VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DETEKCE GRAFFITI TAGŮ V OBRAZE

DETECTION OF GRAFFITI TAGS IN IMAGE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MAREK MOLISCH**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JAKUB ŠPAŇHEL**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Molisch Marek**
Program: Informační technologie
Název: **Detekce graffiti tagů v obraze**
Detection of Graffiti Tags in Image
Kategorie: Zpracování obrazu

Zadání:

1. Prostudujte základy zpracování obrazu. Zaměřte se zejména na problematiku obecné detekce objektů a textu.
2. Vyberte vhodné metody a navrhněte řešení problému detekce graffiti tagů v obraze.
3. Posbírejte vhodnou datovou sadu reálných fotografií graffiti tagů pro vyhodnocení vaší implementace.
4. Experimentujte s vaší implementací a případně navrhněte vlastní modifikace metod.
5. Porovnejte dosažené výsledky a diskutujte možnosti budoucího vývoje.
6. Vytvořte stručný plakát a video prezentující vaši bakalářskou práci, její cíle a výsledky.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění prvních tří bodů zadání.
- Rozpracovaný čtvrtý bod zadání.
- Odevzdání rozepsaného textu práce.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Špaňhel Jakub, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 30. října 2020

Abstrakt

Cílem této práce je porovnat současné architektury modelů, zodpovědné za detekci objektů a použít je pro úlohu detekce graffiti tagů. Pro tyto účely byly v řešení vybrány state-of-the-art modely, které jsou podporovány frameworkem Tensorflow. Architektura Faster R-CNN byla nejpřesnější a architektura SSD nejrychlejší. Také byly provedeny experimenty s graffiti tagy z Athén na datasetu STORM, kde se zjistilo, že ke graffiti tagům je žádoucí přistupovat jako k objektům a ne jako k písmu.

Abstract

The goal of this work is to compare today's architecture of object detection models and use them for the purpose of graffiti tag detection. State-of-the-art models, which are compatible with the Tensorflow framework, were used. Faster R-CNN architecture was found to be the most accurate and SSD architecture to be the fastest. Experiments with graffiti tags from Athens in the STORM dasater showed, that it is better to approach graffiti tags as objects rather than writings.

Klíčová slova

detekce objektů, graffiti tagy, konvoluční neuronové sítě, Faster R-CNN, SSD, CenterNet, EfficientDet, Tensorflow

Keywords

object detection, graffiti tags, convolutional neural networks, Faster R-CNN, SSD, CenterNet, EfficientDet, Tensorflow

Citace

MOLISCH, Marek. *Detekce graffiti tagů v obraze*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jakub Špaňhel

Detekce graffiti tagů v obraze

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jakuba Španhěla. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Marek Molisch
12. května 2021

Poděkování

Hlavní dík patří mému vedoucímu práce, panu Ing. Jakubovi Špaňhelovi za cenné rady, pomoc a dohled při psaní této bakalářské práce. Také bych rád poděkoval mému kamarádovi Danielovi za psychickou podporu během zkouškových období po celé studium a také Anně za pomoc s korekturou. V neposlední řadě nesmím zapomenout na kamarády Adama a Mateje, se kterými jsme se navzájem v tvorbě prací podporovali.

Obsah

Kapitola 1

Úvod

Graffiti nás provázejí na každém kroku už od 80. let, nejprve jako určitá forma protestu mladých proti režimu, později i jako forma umění. Zde se však liší pohled široké veřejnosti, někteří to vnímají jako pouhý vandalismus a vadu na kráse měst a vesnic, ostatní to vnímají jako formu pouličního umění. Ať už jste na té či oné straně, dle zákonů většiny zemí světa se jedná o protiprávní čin a nic nenasvědčuje tomu, že by se situace měla změnit.

Z tohoto důvodu je na místě ověřit, zda-li jsou současné technologie schopné rozeznat (a jak dobře) tuto pouliční činnost prezentující se na fasádách budov, zdí či mostů. Ve větších městech můžete dnes potkat auta, které snímají SPZ značky zaparkovaných aut a zjišťují tak, zda-li zaparkované auto má na dané místo povolení či zaplacené parkovné. Každé auto je registrované na nějakého majitele a díky tomu lze určit, komu případně poslat pokutu.

Pokud by byl tento postup aplikován i na graffiti, dalo by se zjistit, jakému *writerovi* či skupině pravděpodobně patří *tagy* na zkoumaném území a s dalšími informacemi tuto problematiku řešit. Jedním z článků tohoto řetězce by bylo nutné zajistit rychlou a správnou detekci (později pak identifikaci) graffiti tagů, čímž se zabývá tato práce. V kapitole 2 bude postupně probráno co to jsou vůbec graffiti a jejich krátká historie, následovat bude teorie a problematika počítačového vidění, konkrétně detekce objektů. V kapitole 3 jsou rozebrány architektury modelů detektorů, v kapitole 4 pak jejich trénování a popis datasetu. Dosažené výsledky jsou dostupné v kapitole 5 a shrnutí celé práce pak v kapitole 6.

Kapitola 2

Problematika graffiti a detekce objektů

Tato sekce je primárně zaměřena na popsání obecné detekce objektů a nejnovější přístupy k řešení této problematiky, což je jedna z disciplín kterou se zabývá *computer vision* (počítačové vidění). Oblast počítačového vidění jde raketovým tempem dopředu, což potvrzují vědecké práce, které vycházejí i několikrát měsíčně od různých autorů a vždy posunou tuto oblast o mílové kroky vpřed. Menší důraz je v této sekci kladen na obecné popsání graffiti a graffiti tagů, ale pro uvedení této práce do kontextu je nutné znát alespoň obecné informace o této problematice.

2.1 Graffiti

Graffiti je druh umění, se kterým se můžeme setkat každý den, ať už je to v centru velkoměsta nebo na okraji vesnice, dostalo se prakticky všude. Pojem graffiti zastřešuje bezpočet forem *writingu*, ať už se jedná o tagy jednotlivců či skupin, *throw-ups* což jsou sofistikovanější tagy (těmito dvěma druhy se zabývá tato práce) nebo vyloženě forma umění jako je *stencil* či *wildstyle*, které lze vidět na obrázku 2.1.

Obrázek 2.1: Ukázka uvedených stylů graffiti.

2.1.1 Tagy

Graffiti tagy slouží k označení daného místa writerem, popřípadě k identifikaci jeho díla, tedy dalo by se říct, že se jedná o jeho umělecký podpis. Tato tradice sahá až do 70. let v New Yorku, kde se nejprve rozrostla v chudších čtvrtí se zvýšenou kriminalitou [18]. K jeho zhotovení se využívá klasický sprej nebo zvýrazňovač a skoro vždy se jedná o text psaný jedním tahem. V našich končinách se jedná o formu sebevyjádření, ovšem v ulicích měst, kde jsou aktivity gangu a jiných zločineckých organizací, se tímto označují území, kde gangy působí nebo jako forma jeho zabírání. Pohled na graffiti už od jeho existence polarizoval společnost a to proto, že někteří tuto formu umění vnímají jako ničení veřejného prostranství a bezduchý vandalismus, zatímco jiní to obhajují jako formu pouličního umění. Jistým řešením tohoto problému jsou veřejné, legální plochy ve větších městech, které jsou určeny právě k posprejování a potagování writery. Nicméně i přes dostupnost těchto ploch se můžeme setkat s tagy na každém kroku, ať už to jsou posprejované vlaky, fasády domů či výlohy obchodů.

2.2 Strojové učení a detekce

Tato kapitola se zabývá obecným návrhem neuronových sítí, vysvětluje co je to perceptron a konvoluční neuronová síť, jaké formy učení neuronových sítí jsou k dispozici a jak sestavený detektor či klasifikátor ohodnotit na základě přesnosti.

2.2.1 Neuronová síť

Neuronová síť je výpočetní model, a jak již název napovídá, inspirována skutečnou sítí neuronů v lidském mozku. Myšlenka přiblížení se přírodě zde byla již od první poloviny dvacátých let, v roce 1943 již zde byl první návrh a později v roce 1957 vznikl první perceptron.

Perceptron

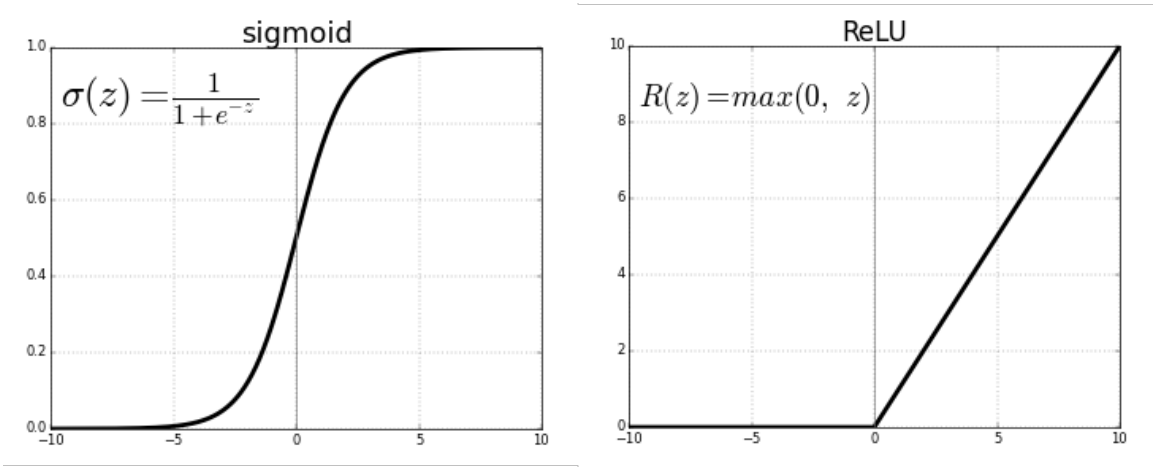
Perceptron je jednoduchý, binární klasifikátor, který se skládá pouze z jednoho neuronu. Na vstupu má m číselných hodnot a ke každé této hodnotě je přiřazená váha w. Váhy se s číselnými hodnotami vstupu vynásobí a sečtou. Tato hodnota slouží jako vstup aktivační funkce, jejímž výstupem je 0 nebo 1 (nebo také -1 a 1, podle vybrané aktivační funkce). Výstup aktivační funkce je poté odeslán do další vrstvy.

Aktivačních funkcí je více druhů, například pro funkci perceptronu jako binárního klasifikátoru to je skoková aktivační funkce, která vrací pro vstup menší než daný práh 0, pro větší pak 1. Dalším možným druhem aktivační funkce je sigmoid, který má na výstupu reálné číslo na intervalu $< 0,1 >$, výstup perceptronu se poté interpretuje jako pravděpodobnost jevu. V současnosti je nejpoužívanější aktivační funkce ReLU, která má výstup 0 pokud je hodnota na vstupu menší než 0 a jinak vrací hodnotu vstupu. Průběh těchto funkcí je vidět na obrázku 2.2. [28]

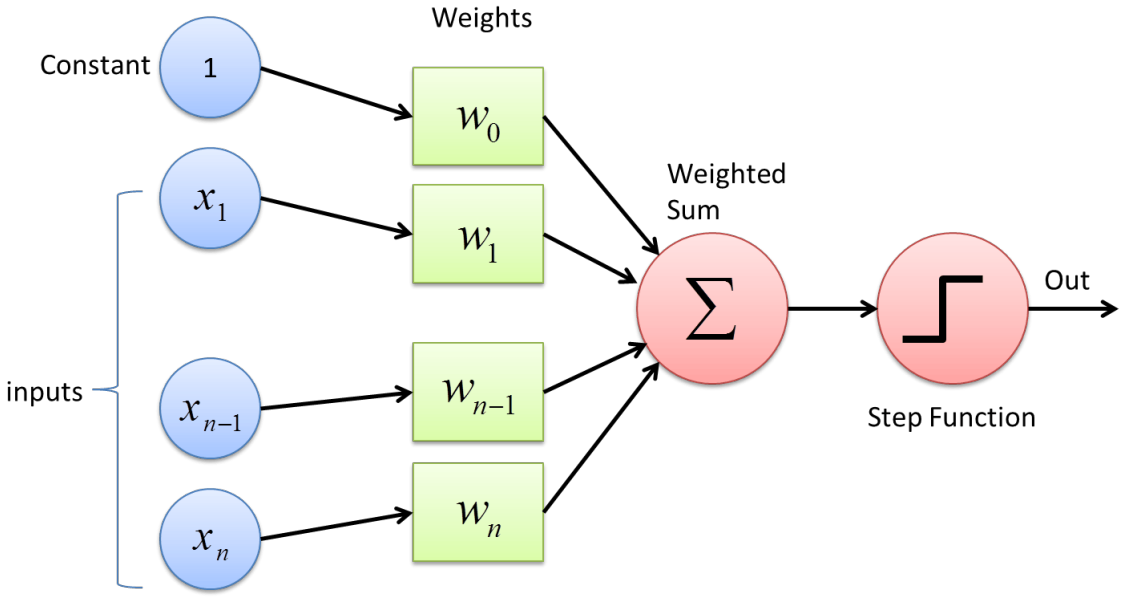
Perceptron je matematický popsán rovnicí:

$$f(x) = \delta(\sum_{i=1}^{m}(w_i x_i) - b),$$

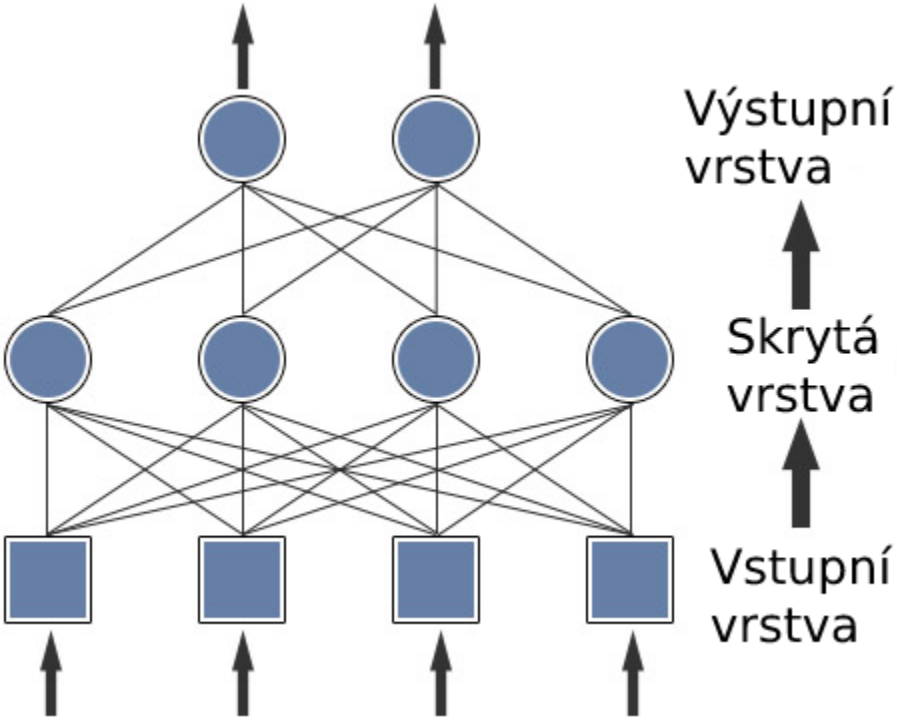
kde $f(x)$ je výsledná hodnota perceptronu, δ aktivační funkce, m počet vstupů, w váha vstupu, x hodnota vstupu a b bias. Schéma perceptronu je uvedeno na obrázku 2.3.



Obrázek 2.2: Funkce sigmoid a ReLU (převzato z [28]).



Obrázek 2.3: Schéma perceptronu (převzato z [29]).



Obrázek 2.4: Druhy vrstev, ze kterých se skládá neuronová síť (upraveno z [15]).

Vrstvy sítě

Stejně jako v lidském mozku jsou neurony propojeny mezi sebou, taktéž jsou i propojeny neurony v neuronových sítích. Každý vstup ze vstupní vrstvy se přivádí do každého uzlu (neuronu) v první skryté vrstvě a odtud do každého uzlu v dalších skrytých vrstvách. Pokud je zde více než jedna skrytá vrstva, nazýváme tento proces jako *deep learning* (hluboké učení). V poslední skryté vrstvě jsou poté propojeny všechny výstupy uzlů s uzly ve výstupní vrstvě. Na jednu vrstvu může být libovolný počet uzlů. Výběr správného počtu uzlů a vrstev je důležitý později při optimalizaci neuronové sítě tak, aby síť byla přizpůsobena námi zvolenému úkolu. Jak je patrné z diagramu 2.4, data jdou pouze jedním směrem a to ze vstupu na výstup, takové síti se říká *feed-forward*. Existují také sítě, které umožňují, aby data putovala směrem z výstupu ke vstupu, ale tato práce se zabývá pouze feed-forward návrhem a jeho modifikaci v podobě *convolutional neural network* (konvoluční neuronové sítě). [15]

2.2.2 Učení neuronových sítí

Neuronové sítě jsou komplexní, ale jejich vnitřní struktura je plně adaptivní. Taková adaptace se nazývá učení a tento proces si lze představit jako modifikaci vah neuronů neuronové sítě tak, aby se optimalizovala na zvolenou úlohu. K změně vah dochází, když se porovná výsledek s realitou a výsledek je odlišný. Pro učení lze použít několik strategií, pro tuto práci je použita strategie s názvem kontrolované učení (někdy uvedeno jako učení s učitelem či učení s dozorem), tedy *supervised learning*. V této sekci bylo čerpáno z článku [16].

Kontrolované učení

Algoritmus učení spadá do této kategorie, pokud je požadovaný výstup pro síť poskytnut zároveň se vstupem. Poskytnutím neuronové sítě vstupním i výstupním párem je možné vypočítat chybu na základě jejího cílového výstupu a skutečného výstupu. Hodnota chyby je využita pro aktualizování hodnoty vnitřních vah pomocí metody *back propagation* (zpětné propagace).

Učení bez dozoru

V tomto přístupu má neuronová síť k dispozici pouze sadu vstupů a je její odpovědností najít nějaký smysl a rysy v poskytovaných datech na vstupu bez jakékoli externí pomoci. Tento typ přístupu učení se často používá při data miningu a používají ho také doporučovací algoritmy kvůli jejich schopnosti předvídat preference uživatele na základě preferencí jiných podobných uživatelů, které seskupují podle podobných rysů.

Posilované učení

Posilované učení je podobné kontrolovanému učení v tom, že je poskytována určitá zpětná vazba, avšak místo poskytnutí cílového výstupu je udělena odměna na základě toho, jak dobře systém fungoval. Cílem posilovaného učení je maximalizovat odměnu, kterou systém obdrží metodou pokus-omyl. Tato metoda učení silně souvisí s tím, jak učení funguje v přírodě, například zvíře si může pamatovat kroky, které dříve provedlo a které mu pomohlo najít potravu (odměnu).

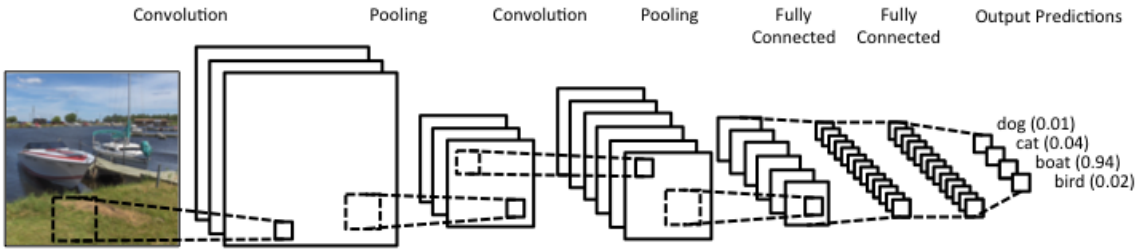
2.3 Konvoluční neuronové sítě

Konvoluční neuronové sítě (CNN) mají stejně jako feed-forward architektura neurony, které se, jak už bylo zmíněno, skládají ze vstupů, vah, váženého součtu, aktivační funkce a výstupu. Jelikož je taktéž využito kontrolovaného učení, chybová funkce nastavuje váhy a mění tak strukturu jednotlivých vrstev. Při tvorbě této kapitoly bylo čerpáno ze zdrojů [17, 2, 27].

Na rozdíl od feed-forward architektury nejsou všechny vrstvy plně propojené, ale neurony jsou spojeny jen s těmi, které jsou si v následující vrstvě prostorově blízké.

U feed-forward architektury se vstupní data překonvertují na jedno-dimenzionální vektor a tím se ztratí informace o lokalitě příznaků. Konvoluční neuronové sítě na rozdíl od toho používají konvoluční vrstvy, které definují filtr (někdy se také uvádí *window* nebo *kernel*), který se aplikuje postupně na celý obrázek část po části. Tento filtr se parametrizuje podle toho, jaké příznaky na obrázku hledáme, například hrany. Výstup této vrstvy známe pod pojmem *feature map* (mapa příznaků). Pokud je spojeno více konvolučních vrstev za sebou (mezi kterými jsou nelineární aktivace (například sigmoid nebo ReLU jak už bylo zmíněno)), lze pak detekovat složitější příznaky. Uvedeno na příkladu, detekce čtverců by se skládala z několika konvolučních vrstev za sebou, které by detekovaly horizontální a vertikální rovnoběžky, kde všechny rovnoběžky mají totožnou velikost a celkem se protnou ve čtyřech bodech. Tímto způsobem, tedy rozložení problému na jednodušší podproblémy, lze detekovat mnohem složitější útvary než jen čtverec, například auta, zvířata nebo graffiti.

Pooling layers jsou vrstvy, které mají za úkol převzorkovat mapy příznaků, na rozdíl od jejich extrakce, a často se střídají s vrstvami konvolučními (například mřížka 6×6 pixelů má po aplikování *poolingu* (sdružování) velikost 3×3 pixely, čímž redukuje její velikost při zachování podstatných informací). Fungují na podobném principu jako konvoluční vrstvy.



Obrázek 2.5: Po složení všech výše uvedených vrstev vzniká model konvoluční neuronové sítě (převzato z [17]).

Dělí se na *max pooling*, který pomocí filtru vrátí nejvyšší hodnotu na daném místě, dále *average pooling* který vrátí průměrnou hodnotu a *minimum pooling* pro nejmenší hodnoty.

Vhodnější přístup k tomuto problému nabízejí *strided convolutions*, které také provádějí převzorkování, ale na rozdíl od explicitního definování, zda se bude jednat o min/average/-max pooling a jeho velikost, je tato operace předmětem učení spolu s modelem, a tak dochází k vhodnějšímu převzorkování.

Na konci konvoluční neuronové sítě se pak nachází plně propojené vrstvy, které mají za úkol zpracovat informace ze všech předchozích vrstev a zajistit tak, aby výstup modelu byla například pravděpodobnost, o jaký objekt se na obrázku jedná. A jak už bylo zmíněno, toho je docíleno pomocí vah, které neurony mají.

Příklad navrhnuté konvoluční neuronové sítě, která se skládá z výše uvedených vrstev, lze vidět na obrázku 2.5.

2.3.1 Hodnocení kvalit detekce konvolučních neuronových sítí

Jelikož existuje mnoho architektur konvolučních neuronových sítí, je nutné je mezi sebou rozlišit na základě výkonu nebo přesnosti (v případě této práce) detekce a klasifikace. Při tvorbě této kapitoly bylo čerpáno z článku [39].

Kategorie detekce a intersection over union

Pro přehlednost jsou zde uvedeny čtyři kategorie detekce:

- True positive (TP) – objekt byl detekován a v obraze se nachází,
- True negative (TN) – objekt detekován nebyl, jelikož se v obrazu žádný nevyskytuje,
- False positive (FP) – objekt byl detekován, ale v obraze se žádný nenachází,
- False negative (FN) – objekt detekován nebyl, ale v obraze se nachází.

IoU, známé také jako Jaccardův index je metrika, která udává do jaké míry se překrývají dva *bounding boxy* detekovaných objektů. Toho je dosaženo pomocí vzorce:

$$IoU = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})},$$

kde je podělen průnik sjednocením predikovaného boxu a *ground truth* boxu. Spolu s *confidence score* (skóre jistoty) tvoří kritéria pro rozhodnutí, zda-li je detekce true positive (TP) nebo false positive (FP). Pseudokód tohoto rozhodnutí je následující:

```
for each detection that has a~confidence score > threshold:
   among the ground-truths, choose one that belongs to the same class and
      has the highest IoU with the detection

   if no ground-truth can be chosen or IoU < threshold:
      the detection s a~false positive
   else:
      the detection is a~true positive
```

Jak naznačuje pseudokód, detekce je považována za TP, pouze pokud splňuje tři kritéria: *confidence score* (skóre jistoty) > *threshold* (prahová hodnota); predikovaná třída odpovídá třídě ground truth boxu a predikovaný bounding box má IoU větší než prahovou hodnotu (standardně 0,5) s ground truth boxem. Porušení kterékoliv z uvedených podmínek má za následek označení detekce jako FP.

Pokud je skóre jistoty detekce nižší než prahová hodnota, detekce se počítá jako false negative (FN) a pokud je skóre jistoty detekce, která nemá nic detekovat, nižší než prahová hodnota, detekce se počítá jako true negative (TN). Při detekci objektů a vyhodnocení modelu se na true negative obvykle nebere ohled.

Precision a recall

Precision (přesnost) je metrika odpovídající podílu počtu případů, kdy byl objekt detekován správně ku počtu všech detekcí, správných či nikoliv, nebo ji lze popsat jako schopnost nalézt pouze objekty, které se v obraze skutečně vyskytují:

$$precision = \frac{TP}{TP + FP}.$$

Oproti tomu *recall* (míra případů, kdy měl být objekt detekován) je metrika kterou lze vyjádřit jako podíl počtu případů, kdy objekt byl detekován správně, a počtu všech očekávaných detekcí nebo popsat jako schopnost nalézt všechny objekty, které se v obraze vyskytují:

$$recall = \frac{TP}{TP + FN}.$$

Z těchto dvou hodnot lze vykreslit křivku, které se říká *precision-recall* a znázorňuje závislost precision na recall při změně confidence thresholdu na intervalu confidence threshold $\epsilon < 0.01, 1 >$.

Mean average precision

I když lze precision-recall křivku použít k vyhodnocení přesnosti detektoru, není to ideální. Lepší způsob řešení je tuto křivku převést do číselné hodnoty, které se říká *average precision* (AP, neboli průměrná přesnost) a jedná se o zprůměrování všech precision hodnot přes všechny hodnoty recallu. AP lze také interpretovat jako plochu pod interpolační precision-recall křivkou, vyjádřeno vzorcem:

$$AP = \sum_{i=1}^{n-1} (r_{i+1} - r_i) p_{interp}(r_{i+1}).$$

Výpočet AP zahrnuje pouze jednu třídu, avšak při detekci objektů se obvykle setkám s více než jednou třídou. Mean average precision (mAP) je definována jako průměr AP ve všech třídách K:

$$mAP = \frac{\sum_{i=1}^{K} AP_i}{K}.$$

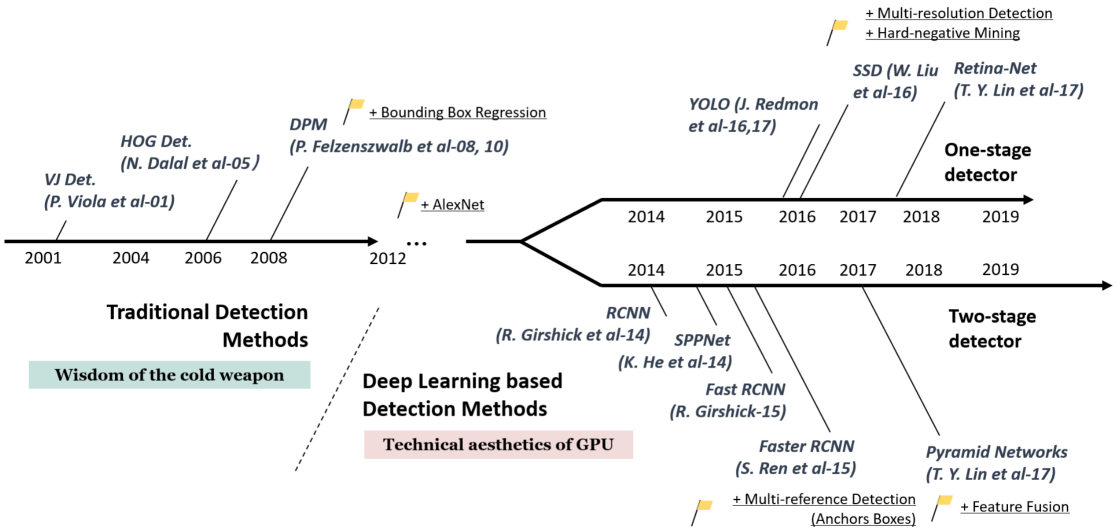
COCO metriky

Pro účely soutěže detektorů nad COCO datasetem byla vytvořena skupina metrik, přičemž nejdůležitější z nich pro tuto práci je $mAP^{IoU=.50:.05:.95}$, která označuje průměrné mAP přes IoU prahové hodnoty v intervalu $IoU\epsilon < 0.5, 0.95 >$ s krokem 0.05. Jedny z dalších metrik jsou $mAP^{IoU=.50}$ (identická s metrikou Pascal VOC) a $mAP^{small,medium,large}$, které počítají mAP na detekovaných objektech s velikostí menších než 32^2, 32^2 až 96^2, respektive větších než 96^2.

Kapitola 3

Modely detekce objektů

Historie vývoje detekce objektů je poměrně pestrá a lze ji dělit na dvě éry: před rokem 2014, kdy se nepoužívalo hluboké strojové učení, a na éru po roce 2014, kde se začaly používat konvoluční neuronové sítě (viz obrázek 3.1). Rozmach použití této metody lze přisuzovat pokroku ve výzkumu a exponenciálnímu zvyšování výpočetního výkonu. V této sekci jsou zběžně popsány architektury před nástupem konvolučních neuronových sítí a do detailu jsou poté rozebrány aktuální modely a jejich architektury.



Obrázek 3.1: Časová osa znázorňující vývoj detekce objektů (převzato z [40]).

3.1 Detekce v reálném čase bez CNN

První velký milník v této oblasti lze přisuzovat dvojici P. Viola a M. Jones, kterým se v roce 2001 podařilo vyvinout detektor v reálném čase pro detekci lidských tváří (poprvé se tak stalo bez omezení barvy pleti, avšak bylo nutné mít obličej situovaný přímo proti kameře). Detektor běžel na (z dnešního pohledu) nevýkonném procesoru Intel Pentium 3, ale i přes to dokázal pracovat s rozlišením 384×288 pixelů a se snímkovou frekvencí 15 FPS. [36]

3.1.1 HOG

HOG, neboli *histogram of oriented gradients* (histogram orientovaných gradientů) je jeden ze způsobů, jak z obrázku extrahovat rysy (tedy provést *feature extraction* a z těchto rysů vytvořit feature mapu). Důvod proč tato metoda vznikla, byla detekce chodců. Pro každý pixel v obrázku se spočítá jeho gradient, směrový vektor a jeho velikost. Obrázek se poté rozdělí na mřížky, které mají velikost 8×8 pixelů. V každé mřížce se pixely rozdělí do 9 kategorií podle směru vektoru a jeho velikostí, a tím vzniká histogram. Poté je nad obrázkem provedena metoda *sliding window* o velikosti 16×16 px, a nad každým oknem se převedou vypočtené histogramy do normalizovaného jedno-dimenzionálního vektoru a tyto vektory, které se řetězí, poté tvoří feature mapu. [37]

3.1.2 DPM

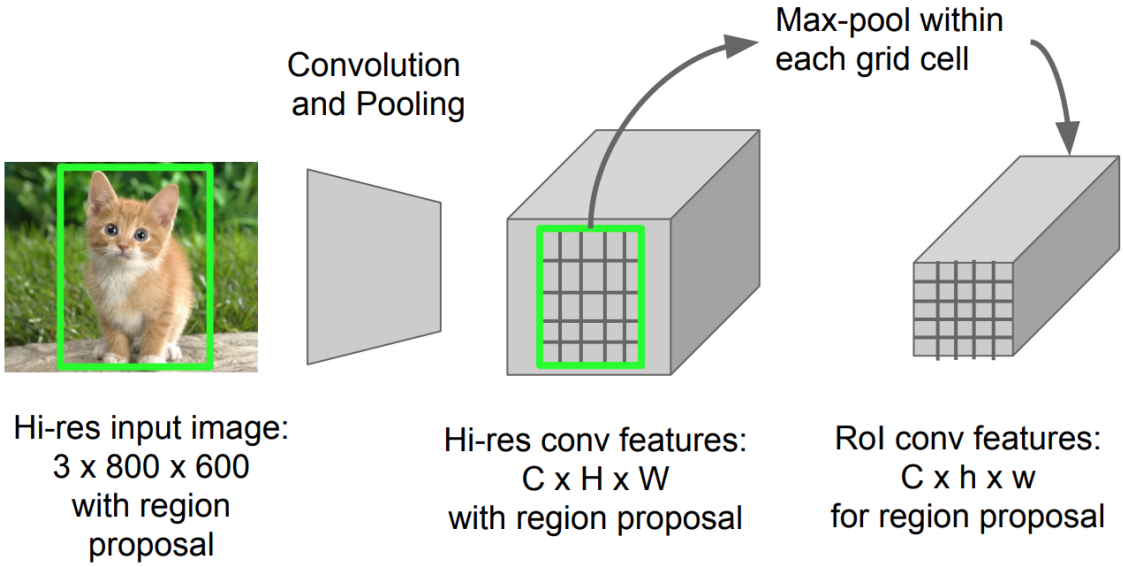
DPM byl nejlepší z tradičních architektur pro detekci objektů (vyhrál soutěž detektorů VOC v letech 2007 až 2009), tedy bez použití neuronových sítí. Tento model byl původně navržen jako rozšíření detektoru HOG a to hlavně pro detekci chodců v pohybu, kde se ohýbají jednotlivé končetiny a proto zde už HOG, který detekuje celý objekt naráz, nestačil. Jak už bylo řečeno, oproti HOG se snaží o detekci různých částí objektu a jejím následním spojením do hledaného objektu. Například detekci auta lze považovat za detekci jeho okna, dveří, kapoty, kol atd., které tvoří jeden objekt. Tento model se skládá z *root* (kořenového) a dalších dílčích filtrů. Místo ručního zadávaní parametrů pro dílčí filtry, které jsou zodpovědné za detekci jednotlivých části objektů, je zde poprvé využito přístupu učení s učitelem a také například regrese bounding boxů. I když z dnešního hlediska je tento model pomalý a ne tak přesný jako ty současné, ale jeho návrh přinesl nové poznatky, které se využívají i v současných konvolučních neuronových sítí. [40]

3.2 R-CNN

R-CNN (2014) byl první model, který aby se vyhnul použití metody sliding window, tak v prvním kroku rozděluje vstupní obrázek na cca. 2000 zón pomocí algoritmu *selective search*, tzn. region proposals (i když tento model není závislý na použitém algoritmu.) Výstupem první fáze je obrázek rozdělený na regiony, u kterých je potřeba zjistit, zda-li se v nich nachází nějaký objekt zájmu. K tomu slouží *feature extractor* AlexNet, který byl na svou dobu technologickou špičkou. Zde nastává problém u R-CNN, jelikož je jako feature extractor použit AlexNet, je potřeba tuto CNN nejprve natrénovat. Vstup AlexNetu je vždy stejný, a to $227 \times 227 \times 3$ (šířka, výška, RGB kanál), proto je nejprve potřeba všechny regiony normalizovat na stejnou velikost. Na výstupu AlexNet je připojen klasifikátor SVM (který je ale potřeba nejprve natrénovat na již natrénované CNN AlexNet), který má n výstupů, kde n je počet možných klasifikovaných objektů a každý výstup je skóre, jak moc si je jistý model tím, že se jedná o námi hledaný objekt. [12, 7]

3.3 Fast R-CNN

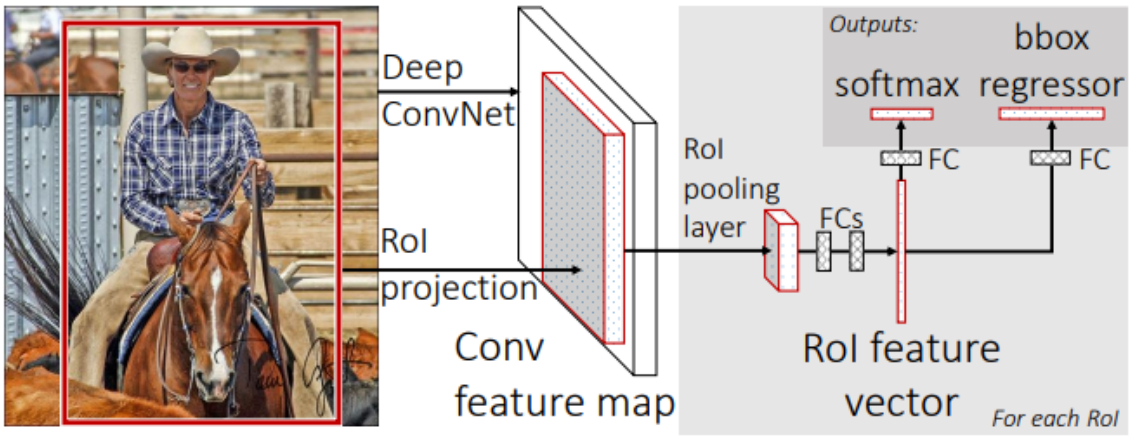
R-CNN má tři hlavní nevýhody: 1) díky použití feature extractoru AlexNet je nejprve potřeba celý obrázek rozdělit na cca 2000 regionů a nad každou aplikovat její průchod 2) části modelu: CNN v podobě AlexNet a SVM se musí trénovat zvlášť a jelikož SVM závisí na vstupu z AlexNetu, nelze je trénovat paralelně 3) každá *feature map* pro každý region



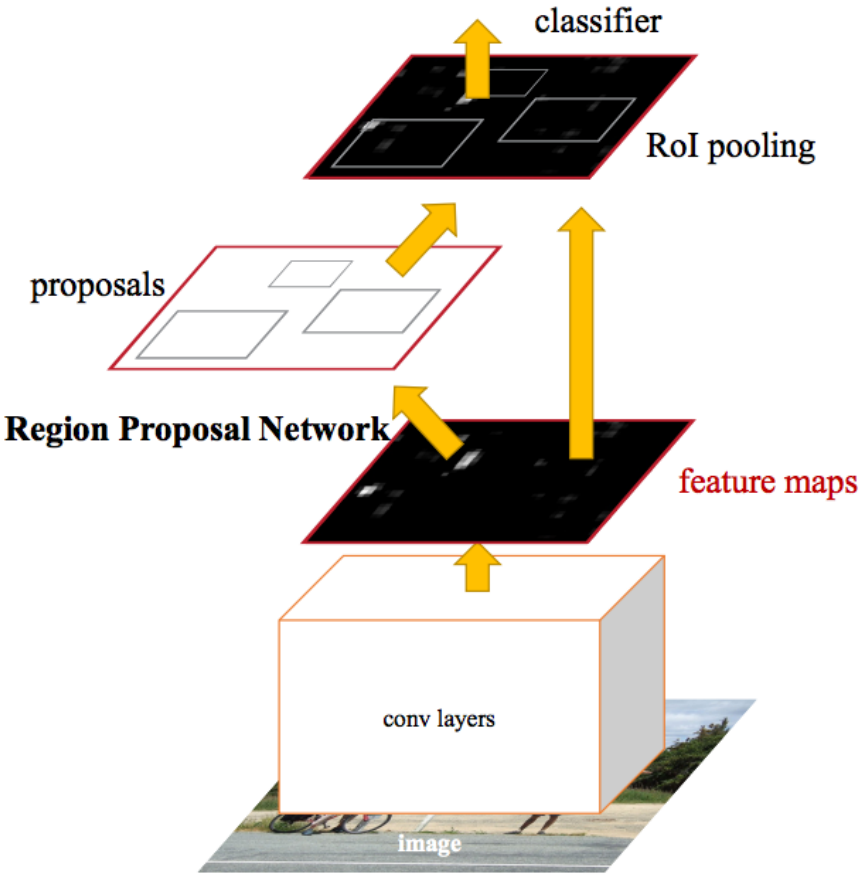
Obrázek 3.2: Princip metody RoI pooling (převzato z [20]).

se musela ukládat do paměti – vysoká paměťová náročnost. Tyto problémy řeší modifikace R-CNN: Fast R-CNN.

Aby se zamezilo aplikování algoritmu selective search na každý region, jde zde využita metoda zvaná *RoI pooling* (Region of Interest Pooling), která provede *max pooling* nad určitým počtem regionů, například 7×7 (počet regionů je nastavitelný hyperparametr) čímž značně zrychlí celý proces detekce (až $100\times$). Princip této metody je ukázán na obrázku 3.2. Třetí problém, tedy tři modely (CNN v podobě AlexNet, klasifikátor SVM a model regrese), které na sobě závisí a nelze je učit paralelně, je vyřešeno také a to spojením do jednoho modelu. SVM klasifikátor je vyměněn za *softmax* vrstvu a k němu je paralelně připojená vrstva regrese, která generuje souřadnice bounding boxů. Schéma modelu je uvedeno na obrázku 3.3. [11, 10]



Obrázek 3.3: Architektura modelu Fast R-CNN (převzato z [11]).



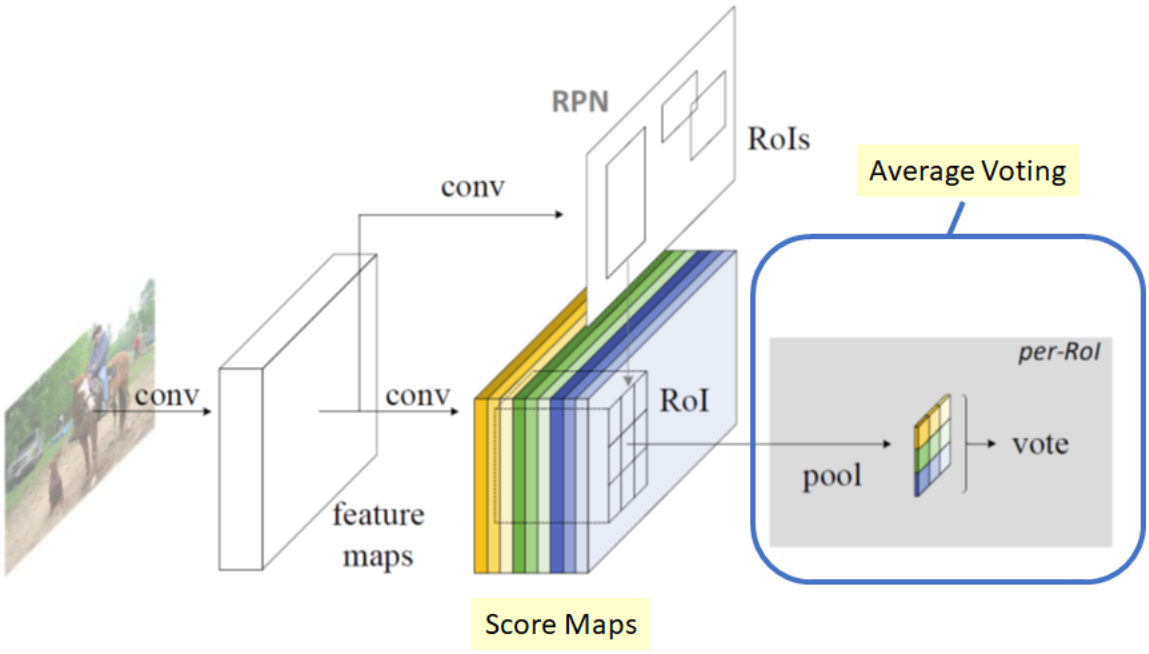
Obrázek 3.4: Architektura modelu Faster R-CNN (převzato z [23]).

3.4 Faster R-CNN

U modelu Fast R-CNN byly návrhy regionu závislé na příznacích obrázků, které byly spočítány při průchodu konvoluční neuronové sítě. Faster R-CNN tedy využije těchto předešlých výpočtů a díky tomu přeskočí pomalou část celého procesu: algoritmus selective search. To je docíleno pomocí přidání plně konvoluční sítě nad CNN, tzv. RPN (Region Proposal Network). Ta pracuje na principu okna, které se posouvá nad mapou příznaků. Z každého vyhodnoceného okna jsou dva výstupy: počet potencionálních ohraničujících rámečků **k** a jejich skóre. Uvedu příklad: pokud chceme detekovat auta, víme, že jejich rámeček bude spíše horizontální a ve tvaru obdélníku, naopak úzký vertikální obdélník pravděpodobně auto nebude. Z těchto znalostí vyvodí obecné poměry stran, které se nazývají *anchor boxes*. Ke každému z nich pak RPN přidělí skóre a jeden bounding box. Schéma modelu je uvedeno na obrázku 3.4. [26, 24]

3.5 R-FCN

R-FCN stejně jako R-CNN využívá dvě fáze pro detekci: *region proposal* a *region classification*. Nejprve se pomocí plně konvoluční sítě RPN (Region Proposal Network) navrhnou regiony výskytu objektu, a nad těmito regiony se provede RoI pooling, ale oproti Faster

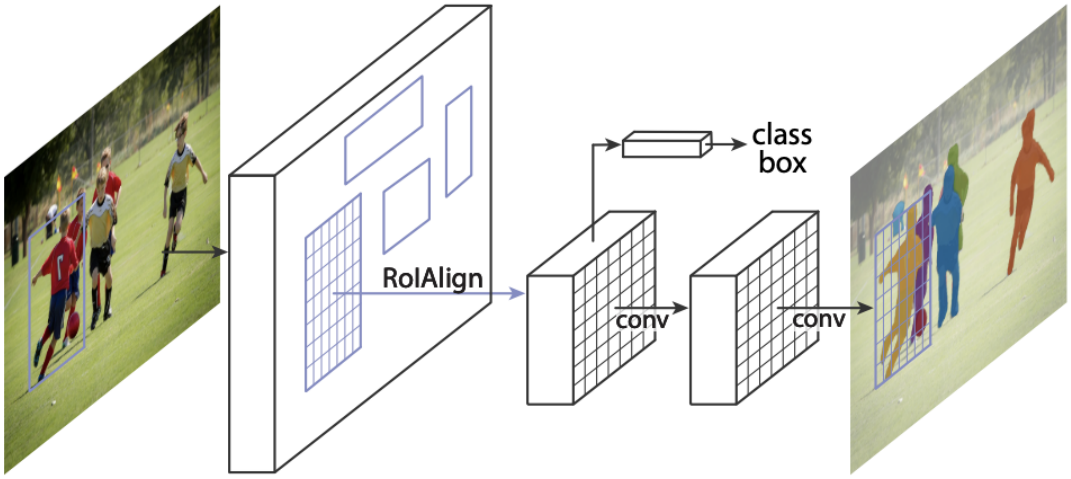


Obrázek 3.5: Architektura modelu R-FCN (převzato z [34]).

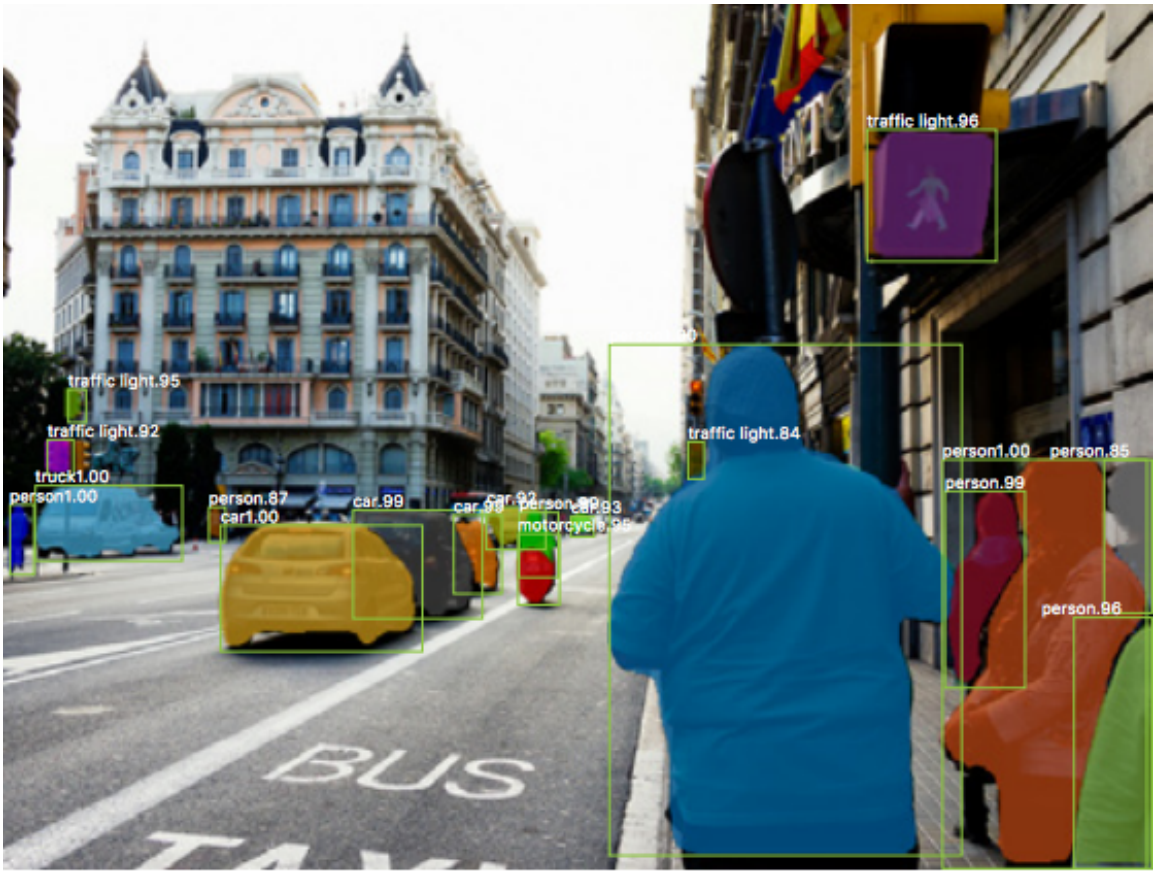
R-CNN již zde nejsou plně propojené vrstvy a komplexní část architektury je přemístěna před RoI pooling, která má za cíl generovat mapy skóre (*score maps*). Všechny návrhy oblastí, které nastanou po RoI pooling, už využívají mapy skóre vypočtené v předchozím kroku které použijí pro *average voting*, takže není potřeba mít v modelu další konvoluční vrstvy a namísto toho se jedná o jednoduchý výpočet. Tímto se docílí dalšího zrychlení oproti Faster R-CNN s podobnými výsledky. Schéma modelu je uvedeno na obrázku 3.5. [34, 5]

3.6 Mask R-CNN

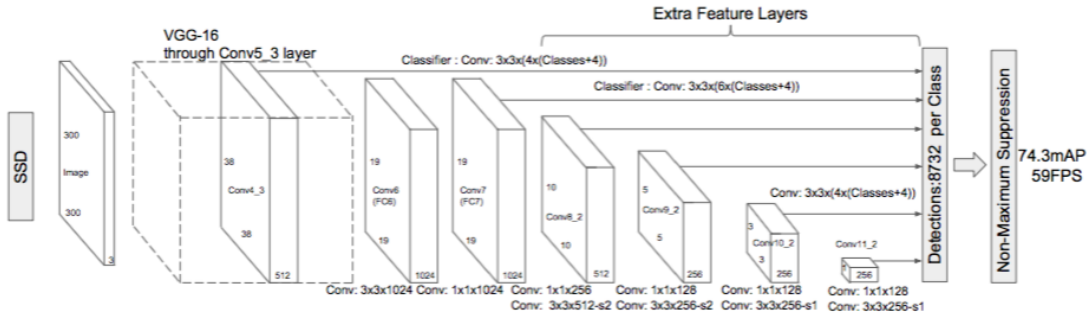
Poslední z rodiny CNN je model Mask R-CNN, který využívá sémantické segmentace, tedy detekovaný objekt je zvýrazněn na úrovni pixelů (klasifikované objekty jsou rozděleny podle barev) namísto bounding boxes, jak je u všech předešlých metod. To je umožněno přidáním větve k modelu Faster R-CNN, jejímž výstupem je matice specifikující, zda daný pixel je součástí objektu (hodnota 1) nebo není (hodnota 0), jinak známo jako *binary mask* (binární maska). Oproti Faster R-CNN má tedy tři výstupy: bounding box, název třídy a binární masku objektu. Namísto metody RoI pooling je zde použita metoda RoI align, a to z důvodu zvýšení přesnosti (jednotlivé pixely oproti ohraničení objektu pomocí bounding boxu), jelikož nedochází k zaokrouhlování hodnot. Příklad, jak detekované objekty vypadají je na obrázku 3.7, architektura modelu pak na obrázku 3.6. Tento model nebyl otestován, jelikož podle [8] má výstupní maska pro úlohu detekce graffiti přesnost v řádu jednotkách procent. [13, 24]



Obrázek 3.6: Architektura modelu R-CNN (převzato z [5]).



Obrázek 3.7: Detekce a klasifikace pomocí Mask R-CNN (převzato z [5]).



Obrázek 3.8: Model SSD300 (převzato z [9]).

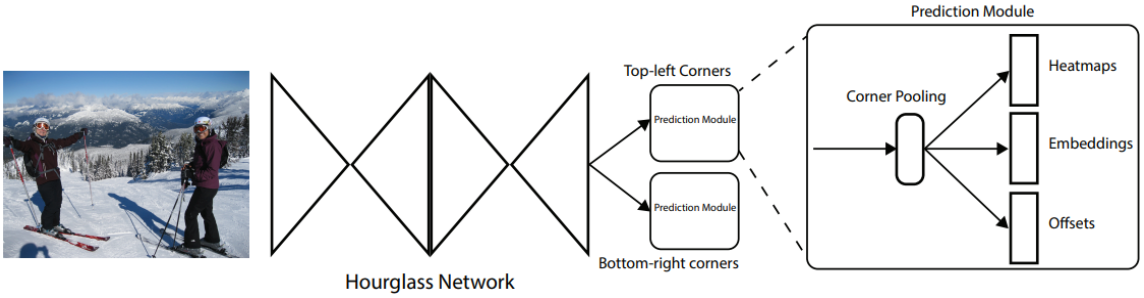
3.7 SSD

Doposud zmíněné modely byly vždy dvoufázové: nejprve se navrhnou regiony zájmu a v druhé fázi se nad těmito jednotlivými regiony vyhodnocuje, zda-li obsahují nějaký objekt. SSD je navrženo tak, aby k detekci a klasifikaci objektu stačil pouze jeden průchod, což značně urychlí celý proces (Faster R-CNN 73.2% mAP při 7 FPS vs SSD300 74.3% mAP při 59 FPS), SSD tedy vzniklo k detekci objektu v reálném čase (použití například v automobilech pro detekci aut na silnici).

SSD model, jehož schéma je na obrázku 3.8, je postaven na architektuře VGG-16, ale zahazuje její plně propojené vrstvy. VGG-16 zde slouží pouze jako feature extractor (a lze ho vyměnit za kterýkoliv jiný). Vrstvy, které následují po, fungují následovně: obrázek je rozdělen do mřížek velikostí 19×19 až 1×1 (19×19 detekuje menší objekty a 1×1 naopak ty největší, zároveň se liší i počet bounding boxů a jejich poměry stran na buňku) a jednotlivé vrstvy konvoluční neuronové sítě fungují jako detektory. Na výstupu každé buňky je pak pozice a třída objektu: pokud se nenalezne žáden objekt, buňka vrací třídu objektu jako pozadí a výsledek se zahodí. SSD používá tzv. default (anchor) boxes při učení, které mají předem definovaný poměr stran a úroveň přiblížení (ne každý objekt bude mít stejnou velikost, i když bude mít stejný poměr stran). Na každou buňku na každé vrstvě se jich vygeneruje několik (tento počet je předem taky daný) a pomocí IoU se vybere ten, který nejvíc odpovídá ground truth boxu. Této informace se poté využívá při samotné detekci, tedy jednotlivé vrstvy a buňky jsou specializované na hledání objektů, které odpovídají nejčastějšímu výskytu ground truth boxů ve fázi učení. Každá buňka pak predikuje offset nalezeného bounding boxu k default boxu a taky skóre, které určuje jak moc si je detektor jistý třídou objektu v rámci bounding boxu. Na výstupu se pak aplikuje *non-max supression* a zůstanou jen bounding boxy, které mají největší skóre, tedy jistotu, že se jedná o hledaný objekt. [9, 21, 14]

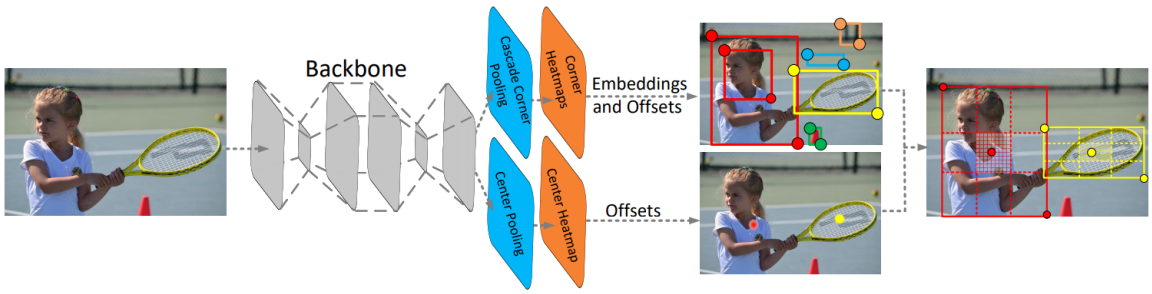
3.8 CornerNet

CornerNet je, podobně jako SSD nebo YOLO, jednofázový detektor, který ale nepoužívá default boxy. Namísto detekování středu objektu a vykreslení bounding boxu kolem něj se snaží o detekci hran objektu (zde vzniká problém, že v okrajích bounding boxu většinou není okraj objektu, k tomu využívá *corner pooling layer* (vrstva pro sdružování okrajů, o tom později)) a to pomocí dvojici bodů *paired keypoints* (párové klíčové body). Jako feature extractor jde zde využit Hourglass Network. Na jeho výstupu jsou umístěny dva moduly,



Obrázek 3.9: Model architektury CornerNet (převzato z [19]).

které se starají o lokaci levého horního a pravého dolního rohu, jak je vidět na obrázku 3.9. Moduly se skládají z corner poolingu, což je metoda vytvořena právě pro tento model. Má za úkol propagovat predikovaným, hraničním bodům informaci o svém okolí a tak zahodit ty, které netvoří budoucí bounding box objektu. Výstup corner poolingu pak tvoří heatmapy (obsahuje hodnoty, které tvoří levé horní rohy a pravé dolní rohy spolu s třídou objektu), *embeddings* (to jsou hodnoty, podle kterých se model snaží spojit levý horní roh se správným pravým dolním rohem aby tvořily bounding box jednoho objektu) a offset. Ten je potřeba kvůli tomu že Hourglass Network provádí mimo jiné i upscaling a downscaling, čili jeho hodnota posune výsledné bounding boxy z heatmapy tak, aby odpovídaly původnímu obrázku. [19]



Obrázek 3.10: Model CenterNet (převzato z [6]).

3.9 CenterNet

Model CenterNet staví na modelu CornerNet a přidává jeden bod navíc a to střed hledaného objektu, body se kterými pracuje se nazývají *keypoint triplets* (trojice klíčových bodů). Motivace pro toto rozšíření byla, že model CenterNet má problémy s rozeznáváním malých objektů kvůli špatným proporcím bounding boxů. Od Fast(er) R-CNN si naopak z části propůjčuje metodu RoI poolingu, ale stále zůstává jednofázovým detektorem, RoI pooling používá k lokalizaci a extrahování features z centru objektu (metoda se zde nazývá center pooling). Z CornerNetu je zde rozšířena metoda corner pooling, která se nazývá *cascade corner pooling* a řeší citlivost corner poolingu na hrany, které ale nejsou okrajové. Architekturu modelu lze viděl na obrázku 3.10. [6, 30]

Stage i	Operator $\mathbf{F}_i$	Resolution $\mathbf{H}_i \times \mathbf{W}_i$	#Channels $\mathbf{C}_i$	#Layers $\mathbf{L}_i$
1	Conv3 × 3	224 × 224	32	1
2	MBConv1, k3 × 3	112 × 112	16	1
3	MBConv6, k3 × 3	112 × 112	24	2
4	MBConv6, k5 × 5	56 × 56	40	2
5	MBConv6, k3 × 3	28 × 28	80	3
6	MBConv6, k5 × 5	14 × 14	112	3
7	MBConv6, k5 × 5	14 × 14	192	4
8	MBConv6, k3 × 3	7 × 7	320	1
9	Conv1 × 1 & Pooling & FC	7 × 7	1280	1

Tabulka 3.1: Architektura modelu EfficientNet-B0.

3.10 EfficientNet

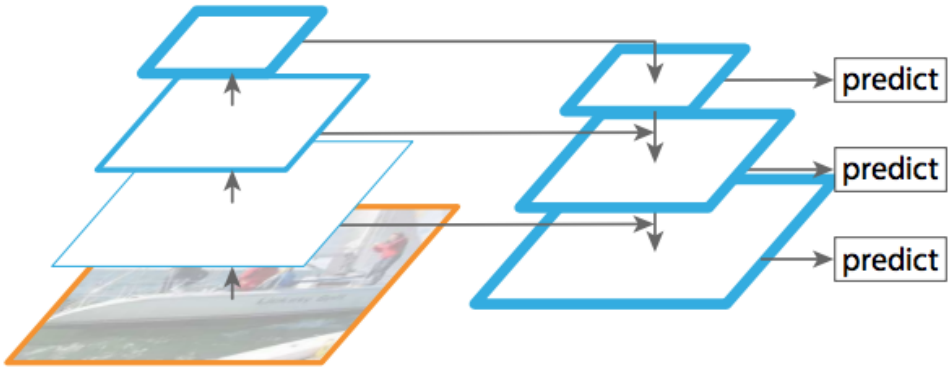
Jeden z problému při vytváření nových state of the art detektorů je škálovatelnost, tedy jaké parametry modelu změnit, aby se dosáhlo lepší přesnosti. Problém ale nastává u velkých a komplexních modelů, kde při zvýšení jednoho ze tří parametrů (šířka, výška, rozlišení) nedochází skoro k žádnému zlepšení za ceny nárůstu požadavků na výkon. Modely se obvykle škálují jedním ze tří způsobů: 1) do šířky, tedy rozšíření počtu kanálů (informací, které produkují jednotlivé vrstvy), např. MobileNet 2) do výšky, tedy zvýšit počet vrstev, např. ResNet 3) zvýšit rozlišení vstupního obrázku. Autoři EfficientNetu navrhují, aby se škálovalo ve všech třech oblastech a tím se docílilo značnému zlepšení přesnosti. Autoři uvádějí, že k optimalizaci jejich vlastní architektury EfficientNet-B0 (B0 určuje *baseline*, tedy model, na kterém se dál testuje škálovatelnost, jeho architektura je popsána na obrázku 3.1), je postavený z *inverted residual* bloků a pro jeho prvotní nastavení byl použit vzorec

$$ACC(m) \times [FLOPS(m)/T)]^{\omega},$$

kde $ACC(m)$ je přesnost modelu, $FLOPS(m)$ je počet operací s plovoucí desetinnou čárkou za sekundu, T je požadovaný počet operací s plovoucí desetinnou čárkou a $\omega = -0.07$ jako hyperparametr, který určuje kompromis mezi přesností a rychlostí. Podle výsledků autorů je dosaženo lepší přesnosti oproti ostatním feature extractorům se značně nižšími požadavkami na výkon zařízení, na kterém modely běží. Dále je v práci popsáno, jak lze vylepšit škálování již existujících feature extractorů pomocí jejich metody, tedy jak už bylo dříve zmíněno, škálovat všechny tři parametry naráz. [31, 35]

3.11 EfficientDet

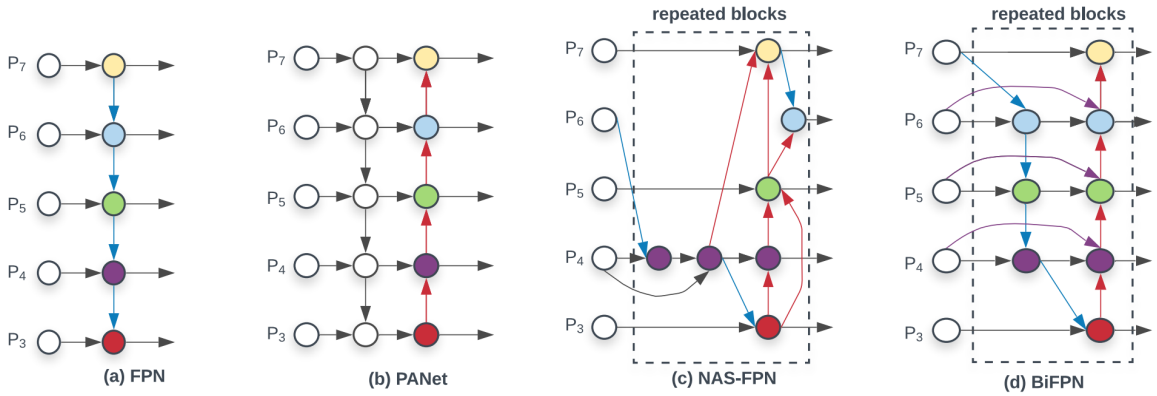
Z poznatků získaných při vytváření feature extractoru EfficientNet, vznikl model EfficientDet, který staví na základech modelu EfficientNet (taktéž jednofázový) a přidává dva nové poznatky: *BiFPN* a *compound scaling*. BiFPN staví na a rozšiřuje FPN (konkrétně model PANet), což je přístup, jak získat *features* z obrázků, které se liší rozlišením (taky samotné objekty se velikostně liší), a na výstupu mít proporcionálně škálované feature mapy na více úrovních. Jak je vidět na obrázku 3.11, jednotlivé vrstvy pyramidy jsou vrstvy konvoluční neuronové sítě a čím je vrstva výše, tím zpracovává *feature* mapy o menším rozlišení s menšími objekty. Při takovém rozdělení dochází k velkým sémantickým mezerám mezi



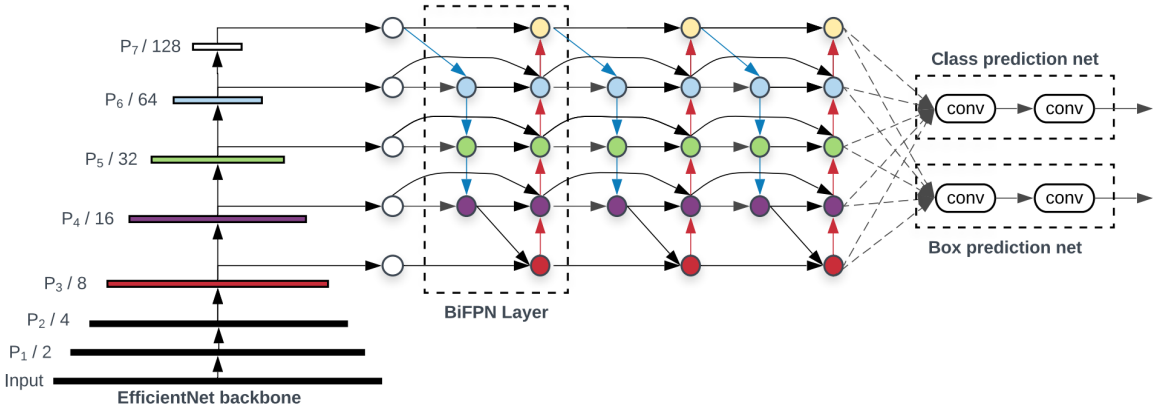
Obrázek 3.11: Princip *feature pyramid network* (převzato z [32]).

jednotlivými vrstvami: u vrstev nižších rozlišení je kladen důraz na tvary, barvy nebo jaké má pixel sousední pixely, u vrstev nižších rozlišení se využívá znalosti z těch vyšších, tedy objekty mají nějaký tvar a barvu a na základě toho je lze označit jako zajímavé a později klasifikovat (například slunce bývá na fotkách kulatého tvaru, je žluté až bílé a leží na obloze, která je do modra). Tento proces kombinování znalostí z jednotlivých vrstev se nazývá *multi-scale fusion*. Pozdější architektury (jako například NAS-FPN nebo PANet) přidávají propagaci z nižších rozlišení do vyšších, nebo i propojení vrstev navzájem mezi sebou, jak je vidět na obrázku 3.12, zde se ale naráží na problém, že aby se tyto vrstvy optimálně propojily, je nutné je učit tisíce hodin. Autoři EffcientDetu tento problém řeší způsoby, že 1. odstraní uzly, které mají pouze jeden vstup (zde je předpoklad, že s jedním vstupem má uzel malý význam pro výsledné váhy), 2. propojí vstupní uzly s výstupnímy, 3. každá vrstva, která obsahuje obousměrné cesty se bere jako jedna vrstva konvoluční neuronové sítě a 4. přidávají jednotlivým feature mapám váhy, které lze trénovat spolu s modelem. Compound scaling se zde stará o rovnoměrné škálování dvou součástí, a to EfficientNetu a BiFPN. Model architektury je popsán na obrázku 3.13.

V předchozím odstavci bylo uvedeno, že optimálně propojit vrstvy trvá tisíce hodin, to by mohlo vést k otázce, jak dlouho trvalo vybrat parametry popisující jednotlivé úrovně tohoto modelu, které jsou popsané v tabulce 3.2. Autoři sdělují, že parametry byly vybrány heuristikou, ale jakou už v článku neuvádějí. [22, 32]



Obrázek 3.12: Princip *multi scale fusion* napříč architekturami (převzato z [32]).



Obrázek 3.13: Architektura modelu EfficientDet (převzato z [32]).

Scale	**Input size** R_{input}	**Backbone network**	**BiFPN** **#channels** W_{bifpn}	**#layers** D_{bifpn}	**Box/class #layers** D_{class}
D0	512	B0	64	3	3
D1	640	B1	88	4	3
D2	768	B2	112	5	3
D3	896	B3	160	6	4
D4	1024	B4	224	7	4
D5	1280	B5	288	7	4
D6	1280	B6	384	8	5
D7	1536	B6	384	8	5
D7x	1536	B7	384	8	5

Tabulka 3.2: Popis velikostí jednotlivých částí modelu EfficientDet podle hodnoty škálování.

Kapitola 4

Trénování detektorů

Pro trénování detektorů byly zvolené dvě prostředí, první bylo lokální v podobě grafické karty NVIDIA RTX 2070 s 8GB VRAM a druhé v podobě Google Colab, kde je k dispozici grafická karta NVIDIA Tesla T4 s 16GB VRAM, ovšem v pozdější fázi implementace byla použita Pro verze, kde jsou k dispozici grafické karty NVIDIA Tesla P/V100, taktéž s 16GB VRAM a je možno mít až čtyři instance naráz, tedy značně se tím urychlilo trénování modelů a jejich evaluace. Původně bylo v plánu využít pouze lokálního prostředí, avšak ukázalo se, že u některých modelů (EfficientDet D4 a výše) i při nastavení *batch size* na 1 není 8GB VRAM dostatečné. Aby model začal detekovat graffiti tagy, je nutné mít model, ze kterého se vychází, předtrénovaný na obecném datasetu [3].

K trénování všech modelů byly použity již předtrénované modely na MS COCO[1] datasetu, který obsahuje až 1,5 miliónů objektů na více než 200 tisících anotovaných obrázcích se 171 kategoriemi objektů. Předtrénované modely se používají, protože již mají přednastavené parametry a váhy uvnitř modelů, například pro RetinaNet-R101 to je 53 miliónů parametrů [32] a tak značně zmenšují potřebný čas (trénovat komplexní modely od nuly by na jedné grafické kartě trvalo dny až týdny), velikost datasetu a ve většině případů i lepší dosažené výsledky. Tomuto procesu se říká *transfer learning* [3].

4.1 Dataset

Pro trénování jednotlivých modelů je použit dataset, který se skládá z fotografií zdí, dveří a dalších objektů, na kterých jsou graffiti tagy. Všechny fotografie jsou pořízené v Brně (viz. mapa 4.3) a rozšiřují tak dataset z minulých let od Martina Fischera a Jana Pavlici [8, 25]. Datasety jsou dva a to z toho důvodu, že jeden sloužit pro trénování detektorů a druhý pro jeho validaci. Dataset pro trénování obsahuje 4804 objektů na 1893 fotografiích a pro validaci čítá 1201 objektů na 487 fotografiích a jejich příklad je na obrázku 4.1. Celkový počet objektů činí 6005 na 2380 fotografií, které byly pro potřebu trénování zmenšeny (i když feature extractory rozlišení fotografií modifikují, pro zachování použitelné velikosti datasetu je nutné fotografie předem zmenšit, mnou dodané fotografie jsou v rozlišení 750×1000). Fotografie byly anotovány programem LabelImg[2], který ukládá anotace ve formátu Pascal VOC do souboru s XML formátováním. Tento formát je nutné převést do formátu TensorFlow a to pomocí upraveného skriptu dostupného z GitHub repozitáře [33], který

[1] https://cocodataset.org/

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

Obrázek 4.1: Příklad fotografií z brněnského datasetu.

převede XML do CSV a poté k informacím z CSV připojí fotografie a uloží je do formátu *tfrecord*, který je binární a se kterým pracuje TensorFlow.

4.1.1 STORM dataset

Pro účely experimentování, zda-li si model naučený na brněnských graffiti poradí s graffiti, které se liší použitým písmem, tedy řecká alfabeta oproti evropské latince, byl vybrán dataset s názvem STORM [4], který obsahuje 1022 fotek graffiti pořízených v Athénách a jejich příklad je na obrázku 4.2. Dataset je rozdělen 80% pro učení a 20% pro validaci. Stejně jako dataset brněnský má různorodost v podobě povrchu, na kterém se graffiti tagy nachází.

4.2 Tensorflow Object Detection API

Pro účely této práce byl vybrán *Tensorflow Object Detection API*, což je open source framework umožňující trénování, evaluaci a exportovaní finálních modelů. Jeho součástí je repozitář, na kterém jsou k dispozici state of the art modely[3], ze kterých jsem vybral EfficientDet D3 a D4, SSD MobileNet v2 320 × 320, SSD ResNet50 V1 FPN 640 × 640, Faster R-CNN ResNet101 V1 640 × 640, CenterNet Resnet101 v1 FPN 512 × 512 a CenterNet HourGlass104 512 × 512.

Jak už bylo zmíněno, modely jsou předtrénované na MS COCO datasetu. Ve většině případů jsou modely předtrénované na několika GPU zároveň a tomu odpovídají i konfigurační soubory (například EfficientDet D3 na TPU[4] s 32 jádry a 256GB VRAM), které je nutné změnit vzhledem k zařízení, na kterém bude evaluace a učení modelu probíhat. Zejména je nutné změnit batch size (počet vzorků z datasetu, které jsou aplikovány na jeden průchod, například na EfficientDet D3 větší hodnota než 2 způsobí, že dojde paměť VRAM i když je

[3] https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md

[4] HW akcelerátor vyvinutý na míru AI aplikacím od společnosti Google.

Obrázek 4.2: Příklad fotografií z datasetu STORM.

k dispozici 16GB), *learning rate* nebo počet tříd (v našem případě 1, jelikož identifikujeme pouze tagy).

Tensorflow Object Detection API generuje checkpointy modelu, ve kterých ukládá aktuální parametry a váhy modelu. Z nejvyššího checkpointu se po skončení učení vygeneruje finální model, na kterém se pak pouští testovací data a evaluace. Během procesu učení lze také na každý nový checkpoint pouštět evaluaci a tak sledovat, zda-li se detektor zlepšuje nebo naopak zhoršuje. Celý proces učení a i evaluaci je možné sledovat z nástroje *Tensorboard*, který vykresluje statistiky modelu během učení a evaluace. Nejdůležitější graf, který je nutné sledovat během učení, je *total loss*, který ukazuje chybu detektoru v čase a lze z něj odhadnout, zda-li je detektor *overfitting*, či *underfitting*, tedy jestli je přeučen nad trénovacím datasetem, nebo naopak, jestli je ještě vhodné pokračovat v učení.

Pro maximální kontrolu nad přesností detektorů bylo provedeno testování na každém vygenerovaném checkpointu. Checkpointy se generují každých 2500 kroků, což zhruba odpovídá půlce datasetu. Pokud model stagnoval s naměřenou hodnotou mAP, pak učení bylo zastaveno a model vyexportován. Pokud se naopak hodnota mAP razantně měnila, učení pokračovalo dále.

Rychlost jednotlivých kroků učení ovlivňuje hlavně nastavený batch size a typ použité grafické karty. Google Colab Pro přiřazuje grafické karty náhodně, ve většině případů byla přiřazena grafická karta NVIDIA Tesla P100, ojediněle V100, která je podstatně rychlejší, v některých typech úloh až dvojnásobně [38].

4.2.1 CenterNet

Model CenterNet s feature extractorem HourGlass104 byl vybrán z toho důvodu, aby se zjistilo, zda-li detekce založená na středu objektů a jeho rohů je vhodná pro tak různorodé objekty jako jsou graffiti. Batch size byl nastaven na 2, přičemž jeden krok trval průměrně 575 ms, celkem bylo provedeno 20 epoch a model byl exportován při kroku 35000, kdy dosáhl největší přesnosti podle COCO metriky $mAP^{IoU=.50:.05:.95}$. Učení trvalo 8 hodin a jedná se tak o druhý nejrychlejší model k naučení.

Obrázek 4.3: Fotky graffiti tagů podle vyfocené lokace.

4.2.2 EfficientDet

Architektura EfficientDet se momentálně pokládá za špičku detekce podle datasetu MS COCO. Jak už bylo řečeno v kapitole o EfficientDet, architekturu lze škálovat a pro účely této práce byly vybrány stupně D3 a D4, viz tabulka 3.2. Rozlišení vstupního obrázku pro D3 je nejblíž průměrné hodnotě rozlišení v celém datasetu, přičemž D4 bylo vybráno pro otestování, zda-li upscale vstupního obrázku, spolu s nárůstem ostatních parametrů, ovlivní výsledné skóre detekce.

D3

Při trénování modelu EfficientDet D3 byl nastaven batch size na 2, přičemž jeden krok trval průměrně 1040 ms, celkově bylo provedeno 41 epoch a model byl exportován při kroku 79200. Učení modelu zabralo 29 hodin, nejdéle ze všech testovaných.

D4

U tohoto modelu musel být nastaven batch size na 1, jinak docházelo ke spadnutí trénování z důvodu nedostatku paměti VRAM. Jeden krok trval průměrně 950 ms, celkově bylo provedeno 20 epoch a export modelu byl proveden při kroku 79200, kde se už ustálila hodnota loss.

4.2.3 Faster R-CNN

U architektury Faster R-CNN existuje mnoho feature extractorů, avšak jelikož se jedná o straší model, byl vybrán feature extractor Inception Resnet v2 se vstupním rozlišením 640×640, který má na datasetu MS COCO naměřenou nejvyšší přesnost. Batch size byl nastaven na 2 a model byl exportován při kroku 50400, přičemž jeden krok trval 780 ms.

4.2.4 SSD

Z architektury SSD byly vybrány dva feature extractory, ResNet50 V1 pro optimální poměr rychlosti/přesnosti a MobileNet V2 FPNLite byl vybrán jako nejrychlejší detektor ze všech testovaných.

ResNet50 V1

Při trénování SSD se zmíněným feature extractorem byl nastaven batch size na 2, model byl exportován při kroku 97500 a celkově bylo provedeno 41 epoch. Jeden krok průměrně trval 242 ms a jednalo se tak o druhý nejrychlejší model, přičemž učení zabralo necelých 7 hodin.

MobileNet V2 FPNLite

Tento feature extractor je optimalizovaný pro mobilní zařízení, proto jeho učení bylo nejrychlejší ze všech a zároveň mělo nejmenší nároky na paměť VRAM, díky tomu bylo možné spustit učení s batch size 16, přičemž jeden krok trval 456 ms a celkově učení zabralo 13 hodin. Díky vysoké hodnotě batch size bylo provedeno 333 epoch, ale přesnost modelu se neměnila (až na velmi malé odchylky v řádu tisícin) už od kroku 20000, kde se již ustálila hodnota funkce loss. Model byl exportován při posledním kroku, tedy 100000, ale bylo možné provést export již při 20000.

Kapitola 5

Dosažené výsledky

V této kapitole jsou vyhodnoceny výsledky z výše uvedeného učení, které klade za úkol čím jak největší přesnost detekce graffiti tagů, potažmo optimálnímu poměru rychlosti detekce a přesnosti. Jak už bylo řečeno, při vyhodnocování všech modelů bylo použito metrik COCO, které v sobě zahrnují i starší PASCAL-VOC. Pro připomenutí, jedná se o $mAP^{IoU=.50:.05:.95}$ (COCO) a $mAP^{IoU=.50}$ (PASCAL-VOC).

5.1 Brněnský dataset

První část experimentů probíhala na brněnském graffiti datasetu. Pokud se vezme do úvahy pouze přesnost, pak model Faster R-CNN s feature extractorem Inception Resnet v2 je z testovaných detektorů nejpřesnější, a to s přesností 55,8% podle COCO metriky, avšak zároveň se jednalo o nejpomalejší model s průměrnou rychlostí 353,9 ms na jeden snímek, neboli 2,8 FPS. V případě nasazení tohoto modelu do aplikace, která by měla za úkol detekovat graffiti v reálném čase, je tento model v podstatě nepoužitelný.

Na druhém místě je pak EfficientDet D3 s feature extractorem EfficientNet, na kterém, jak už bylo řečeno, je celá architektura, včetně škálování, postavena a to s přesností 51,2%. Průměrně jeden snímek trval 95 ms, neboli 10,5 FPS. Pokud zvýšíme převzorkování vstupního obrázku a to na úroveň EfficientDet D4, přičemž vstupní obrázek je ve stále nižším rozlišení, dojde pouze k zhoršení přesnosti a k zpomalení celému procesu. Naměřená přesnost je 43,8% s průměrnou rychlostí na jeden snímek 155,3 ms, neboli 6,4 FPS.

Model CenterNet s feature extractorem HourGlass 104 byl s přesností 46,4% čtvrtý v pořadí, oproti EfficientDet D3 byl pomalejší o 0,2 ms na snímek avšak s přesností horší o necelých 5% a z tohoto důvodu není vhodným kandidátem pro detekci graffiti. Pokud se ale použije feature extractor Resnet101 v1 FPN, dojde ke zvýšení přesnosti o 1,3% při trojnásobné rychlosti: 32 ms se snímkovou frekvencí 31,2 FPS. Jednalo se tak o druhý nejrychlejší a třetí nejpřesnější model.

Model SSD s feature extractorem ResNet50 V1 FPN byl třetím nejrychlejším modelem s časem 46,6 ms na snímek a 21,6 FPS při přesnosti 44,4%. Posledním modelem bylo SSD s feature extractorem MobileNet V2 FPNLite, který zde slouží hlavně jako ukazatel aktuálně nejrychlejšího a hlavně podporovaného modelu pro Tensorflow Object Detection API. S časem na jeden snímek 26,6 ms a 37,6 FPS se jednalo o nejrychlejší model z testovaných a to při přesnosti 35,4%. Všechny tyto data jsou přehledně sepsány v tabulce 5.1 a jednotlivé detekce všech testovaných modelů jsou na obrázku 5.1.

Model	mAP COCO (%)	mAP PASCAL (%)	rychlost (ms)	FPS
CenterNet HourGlass104	46,4	71,2	97,2	10,3
CenterNet Resnet101 v1 FPN	47,7	73,3	32	31,2
EfficientDet D3	51,2	75,7	95,0	10,5
EfficientDet D4	43,8	75,2	155,3	6,4
Faster R-CNN Inception Resnet v2	**55,8**	**82,6**	353,9	2,8
SSD MobileNet v2 FPNLite	35,4	59,4	**26,6**	**37,6**
SSD ResNet50 V1 FPN	44,4	69,6	46,4	21,6

Tabulka 5.1: Naměřené hodnoty pro všechny testované modely.

Dataset learn → dataset test	mAP COCO (%)	mAP Pascal (%)
Brno → Brno	44,4	69,6
Brno → Athény	38,1	57,2
Athény → Brno	33,8	58,7
Athény → Athény	42,2	64,2
Brno + Athény → Brno	44,6	69,5
Brno + Athény → Athény	46,4	69,0
Brno + Athény → Brno + Athény	44,9	69,2

Tabulka 5.2: Naměřené hodnoty s modelem SSD ResNet50 V1 FPN po kombinaci obou datasetů.

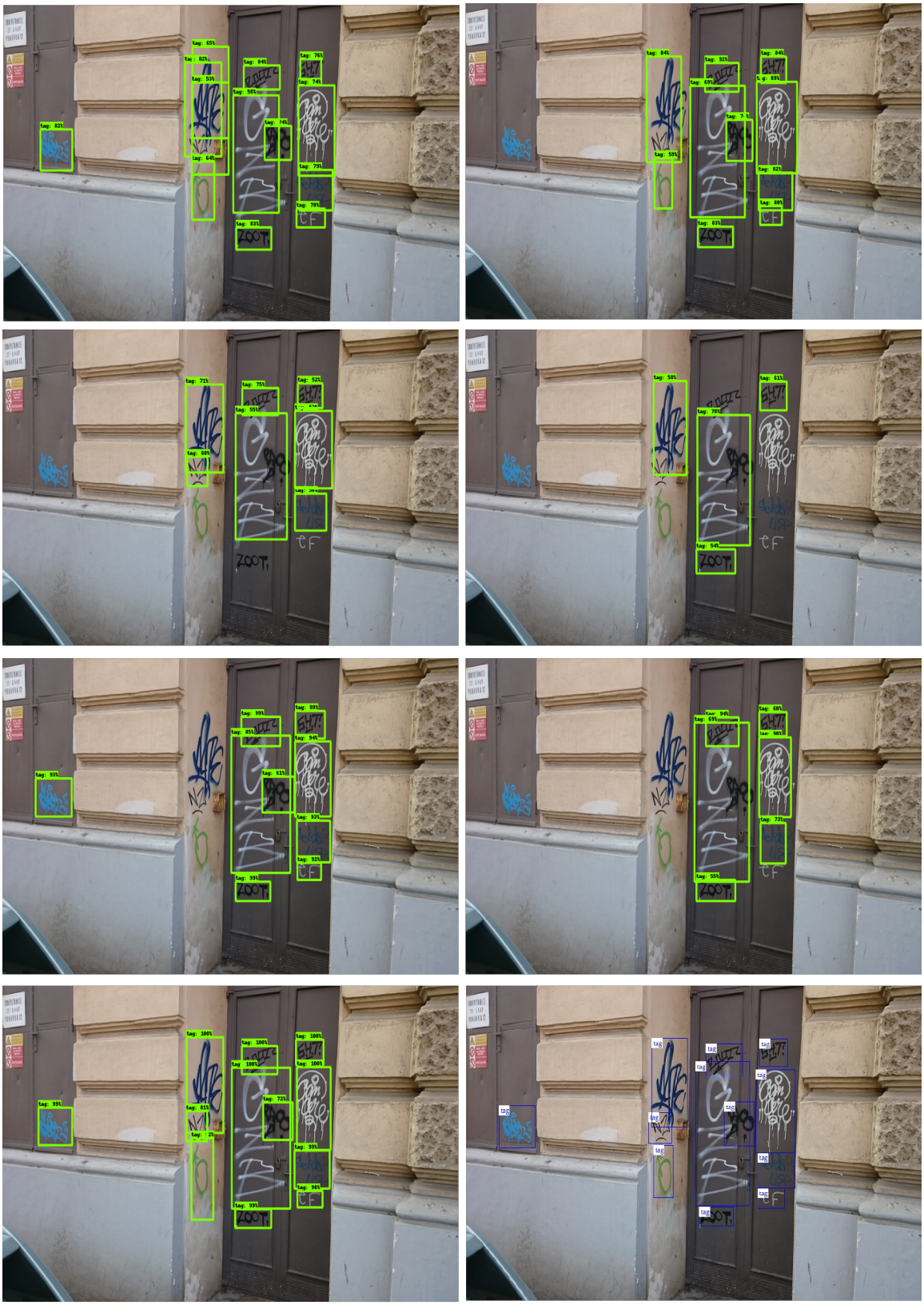
5.2 Experiment s datasetem STORM

Tento experiment byl proveden pro ověření, zda-li je model naučený na graffiti z jedné části světa schopen detekovat graffiti z úplně jiné lokality. Jak už bylo řečeno v sekci 4.1.1, jedná se o dataset s názvem STORM, který byl pořízen na území řeckých Athén. Pro tento experiment byl vybrán model SSD ResNet50 V1 FPN.

Nejprve byl model naučen pouze na datasetu STORM, kde dosáhl přesnosti 42,2%, následně byl dataset vyměněn za brněnský, kde dosáhl přesnosti 44,4% a v posledním kroku byl model naučen na kombinací obou datasetů, kde dosáhl přesnosti 44,9%. Jak je vidět z naměřených dat v tabulce 5.2, učení s datasetem STORM není dostatečné, pokud se model testuje na brněnských datech a naopak. Pokud tyto datasety zkombinujeme, v případě brněnského datasetu dojde k zlepšení o pouhých 0,2%, ale v případě datasetu STORM dojde k zlepšení o 4,2%.

Z těchto výsledků vyplývá, že v případě dostatečně velkého datasetu přidáním dalších fotografií, i když z jiné části světa, dojde k zanedbatelnému zlepšení. Z naměřených dat také vyplývá, že graffiti tagy jsou si velice podobné napříč světem a pokud je k dispozici dostatečně velký dataset, lze ho použít i v jiných částech světa, i když se daná lokalita liší písmem, v případě Řecka to je alfabeta oproti běžné latince.

Z těchto naměřených dat lze také v závěru vyvodit, že v případě úlohy detekce graffiti není vhodné ke graffiti přistupovat jako k písmu, ale nýbrž jako k objektu, jelikož je model schopen detekce, i přestože je naučen na jiném druhu písma a to za cenu zhoršení přesnosti. Zhoršení přesnosti lze ale také přikládat menší velikosti datasetu STORM, oproti tomu brněnskému.



Obrázek 5.1: Detekce dle jednotlivých modelů **1.** EfficientDet D3, **2.** EfficientDet D4, **3.** CenterNet HourGlass104, **4.** CenterNet Resnet101 v1 FPN, **5.** SSD ResNet50 FPN, **6.** SSD MobileNet v2 FPNLite, **7.** Faster R-CNN Inception Resnet v2, **8.** Ground truth.

Kapitola 6

Závěr

Cílem této práce bylo porovnat různé přístupy architektur modelů, snažící se o detekci graffiti tagů v obraze. Tento problém je stížen prostředím, ve kterém se graffiti tagy nacházejí, jako jsou například dveře, zdi s motivem, lavičky atp. Navíc se každý graffiti tag liší barvou, velikostí a tvarem, čímž se stává detekce ještě obtížnějším. Pro tyto účely byl rozšířen dataset z minulých let o mnou pořízené fotografie graffiti tagů.

Vybrané modely jsou state of the art které aktuálně podporuje Tensorflow Object Detection API a jsou předem natrénované na datasetu MS COCO. Každý model se velice liší návrhem architektury a použitím, některé jsou velice pomalé, ale přesné na rozdíl od těch, které slouží pro mobilní zařízení a tím pádem jsou velice rychlé s horší přesností.

Z mnou testovaných detektorů nejlépe dopadl Faster R-CNN Inception Resnet v2 s mAP 55,8% (při použití COCO metriky), který ale zároveň byl nejpomalejší s průměrnou snímkovací frekvencí 2,8 FPS. Nejrychlejší detektor byl SSD MobileNet v2 FPNLite určený pro mobilní zařízení s přesností 35,4% s průměrnou snímkovací frekvencí 37,6 FPS. Jako nejlepší kandidát pro detekci graffiti se ukázal model CenterNet Resnet101 v1 FPN a EfficientDet D3 s přesností 47,7% a 31,2 FPS, respektive 51,2% a 10,5 FPS. Z naměřených výsledků také vyplývá, že nejvhodnější feature extractor pro detekci graffiti tagů je architektura ResNet.

Zároveň se při testování na datasetu s písmem jiným, než je latinka (konkrétně řecká alfabeta) ukázalo, že při dostatečně velkém datasetu se přidáním dalších fotografií zlepší přesnost detektoru zanedbatelně a zároveň, že v případě detekce graffiti není vhodné ke graffiti přistupovat jako k písmu, ale nýbrž jako k objektu, jelikož je model schopen detekce, i přestože je naučen na jiném druhu písma a to za cenu zhoršení přesnosti.

Na závěr je potřeba dodat, že detekce objektů, potažmo celá oblast počítačového vidění, jde každým dnem velice rychle dopředu a již dnes existují pokročilejší modely, které ale nejsou implementovány v použitém Tensorflow Object Detection API, jako je například YOLOv4-P7, CenterNet2 nebo Swin-L [1].

Literatura

[1] *Papers with Code - COCO test-dev Benchmark (Object Detection)*. May 2021 [cit. 2021-03-28]. Dostupné z: https://paperswithcode.com/sota/object-detection-on-coco.

[2] Bansari, S. *Introduction to how CNNs Work*. DataDrivenInvestor, Apr 2019 [cit. 2021-03-28]. Dostupné z: https://medium.datadriveninvestor.com/introduction-to-how-cnns-work-77e0e4cde99b.

[3] Brownlee, J. *A Gentle Introduction to Transfer Learning for Deep Learning*. Sep 2019 [cit. 2021-03-28]. Dostupné z: https://machinelearningmastery.com/transfer-learning-for-deep-learning/.

[4] Charalampos, P., Panagiotis, K., Lazaros, T. a Alexandros, T. *STORM graffiti/tagging detection dataset*. Zenodo, červen 2019 [cit. 2021-03-28]. DOI: 10.5281/zenodo.3238357. Dostupné z: https://doi.org/10.5281/zenodo.3238357.

[5] Dai, J., Li, Y., He, K. a Sun, J. R-FCN: Object Detection via Region-based Fully Convolutional Networks. *CoRR*. 2016, abs/1605.06409. Dostupné z: http://arxiv.org/abs/1605.06409.

[6] Duan, K., Bai, S., Xie, L., Qi, H., Huang, Q. et al. CenterNet: Keypoint Triplets for Object Detection. *CoRR*. 2019, abs/1904.08189. Dostupné z: http://arxiv.org/abs/1904.08189.

[7] Elfouly, S. *R-CNN*. Medium, Nov 2020 [cit. 2021-03-28]. Dostupné z: https://medium.com/@selfouly/r-cnn-3a9beddfd55a.

[8] Fischer, M. Detekce graffiti tagů v obraze. June 2019. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/121881.

[9] Forson, E. *Understanding SSD MultiBox - Real-Time Object Detection In Deep Learning*. Towards Data Science, Jun 2019 [cit. 2021-03-28]. Dostupné z: https://towardsdatascience.com/understanding-ssd-multibox-real-time-object-detection-in-deep-learning-495ef744fab.

[10] Gandhi, R. *R-CNN, Fast R-CNN, Faster R-CNN, YOLO - Object Detection Algorithms*. Towards Data Science, Jul 2018 [cit. 2021-03-28]. Dostupné z: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e.

[11] Girshick, R. Fast R-CNN. Apr 2015. Dostupné z: https://arxiv.org/abs/1504.08083.

[12] Girshick, R., Donahue, J., Darrell, T. a Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. Oct 2014. Dostupné z: https://arxiv.org/abs/1311.2524.

[13] He, K., Gkioxari, G., Dollár, P. a Girshick, R. B. Mask R-CNN. *CoRR*. 2017, abs/1703.06870. Dostupné z: http://arxiv.org/abs/1703.06870.

[14] Hollemans, M. *One-stage object detection*. Jun 2018 [cit. 2021-03-28]. Dostupné z: https://machinethink.net/blog/object-detection/.

[15] Jacobson, L. *Introduction to Artificial Neural Networks - Part 1*. Dec 2013 [cit. 2021-03-28]. Dostupné z: https://www.theprojectspot.com/tutorial-post/introduction-to-artificial-neural-networks-part-1/7.

[16] Jacobson, L. *Introduction to Artificial Neural Networks Part 2 - Learning*. Mar 2014 [cit. 2021-03-28]. Dostupné z: https://www.theprojectspot.com/tutorial-post/introduction-to-artificial-neural-networks-part-2-learning/8.

[17] Jordan, J. *Convolutional neural networks*. Apr 2019 [cit. 2021-03-28]. Dostupné z: https://www.jeremyjordan.me/convolutional-neural-networks/.

[18] KOŠ, F. *Graffiti: Formování mladého writera [online]*. 2018 [cit. 2021-04-28]. Dostupné z: https://is.muni.cz/th/zvh1r/.

[19] Law, H. a Deng, J. CornerNet: Detecting Objects as Paired Keypoints. 2019. Dostupné z: https://arxiv.org/abs/1808.01244.

[20] Li, F.-F., Karpathy, A. a Johnson, J. *Spatial Localization and Detection*. Feb 2016 [cit. 2021-03-28]. Dostupné z: http://cs231n.stanford.edu/slides/2016/winter1516_lecture8.pdf.

[21] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. et al. SSD: Single Shot MultiBox Detector. *Lecture Notes in Computer Science*. Springer International Publishing. 2016, s. 21–37. ISSN 1611-3349. Dostupné z: http://dx.doi.org/10.1007/978-3-319-46448-0_2.

[22] Nain, A. *EfficientDet: Scalable and Efficient Object Detection*. Medium, Nov 2019. Dostupné z: https://medium.com/@nainaakash012/efficientdet-scalable-and-efficient-object-detection-ea05ccd28427.

[23] Nguyen, C., Tran, G. S., Nghiem, T., Doan, N., Gratadour, D. et al. Towards Real-Time Smile Detection Based on Faster Region Convolutional Neural Network. In:. Apr 2018, s. 1–6. DOI: 10.1109/MAPR.2018.8337524. Dostupné z: https://www.researchgate.net/publication/324549019_Towards_Real-Time_Smile_Detection_Based_on_Faster_Region_Convolutional_Neural_Network.

[24] Parthasarathy, D. *A Brief History of CNNs in Image Segmentation: From R-CNN to Mask R-CNN*. Athelas, Apr 2017 [cit. 2021-03-28]. Dostupné z: https://blog.athelas.com/a-brief-history-of-cnns-in-image-segmentation-from-r-cnn-to-mask-r-cnn-34ea83205de4.

[25] Pavlica, J. Detekce graffiti tagů v obraze. June 2017. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/106263.

[26] Ren, S., He, K., Girshick, R. B. a Sun, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *CoRR*. 2015, abs/1506.01497. Dostupné z: http://arxiv.org/abs/1506.01497.

[27] Rohrer, B. *How do Convolutional Neural Networks work?* Aug 2016 [cit. 2021-03-28]. Dostupné z: https://e2eml.school/how_convolutional_neural_networks_work.html.

[28] Sharma, S. *Activation Functions in Neural Networks*. Towards Data Science, Feb 2019. Dostupné z: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[29] Sharma, S. *What the Hell is Perceptron?* Towards Data Science, Oct 2019 [cit. 2021-03-28]. Dostupné z: https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53.

[30] Shin, T. *CenterNet Keypoint Triplets for Object Detection Review*. Towards Data Science, Nov 2019 [cit. 2021-03-28]. Dostupné z: https://towardsdatascience.com/centernet-keypoint-triplets-for-object-detection-review-a314a8e4d4b0.

[31] Tan, M. a Le, Q. V. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *CoRR*. 2019, abs/1905.11946. Dostupné z: http://arxiv.org/abs/1905.11946.

[32] Tan, M., Pang, R. a Le, Q. V. EfficientDet: Scalable and Efficient Object Detection. *CoRR*. 2019, abs/1911.09070. Dostupné z: http://arxiv.org/abs/1911.09070.

[33] Tran, D. *Raccoon Detector Dataset*. GitHub, 2017 [cit. 2021-03-28]. Dostupné z: https://github.com/datitran/raccoon_dataset.

[34] Tsang, S.-H. *Review: R-FCN-Positive-Sensitive Score Maps (Object Detection)*. Towards Data Science, Mar 2019 [cit. 2021-03-28]. Dostupné z: https://towardsdatascience.com/review-r-fcn-positive-sensitive-score-maps-object-detection-91cd2389345c.

[35] Tsang, S.-H. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks (Image Classification)*. Medium, Nov 2020 [cit. 2021-03-28]. Dostupné z: https://sh-tsang.medium.com/efficientnet-rethinking-model-scaling-for-convolutional-neural-networks-image-classification-ef67b0f14a4d.

[36] Viola, P. a Jones, M. Rapid object detection using a boosted cascade of simple features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. Dec 2001. DOI: 10.1109/cvpr.2001.990517. Dostupné z: https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf.

[37] Weng, L. *Object Detection for Dummies Part 1: Gradient Vector, HOG, and SS*. Oct 2017 [cit. 2021-03-28]. Dostupné z: https://lilianweng.github.io/lil-log/2017/10/29/object-recognition-for-dummies-part-1.html#histogram-of-oriented-gradients-hog.

[38] Xcelerit. *Benchmarks: Deep Learning Nvidia P100 vs V100 GPU*. Mar 2018 [cit. 2021-03-28]. Dostupné z: https://www.xcelerit.com/computing-benchmarks/insights/benchmarks-deep-learning-nvidia-p100-vs-v100-gpu/.

[39] Zeng, N. *An Introduction to Evaluation Metrics for Object Detection: NickZeng*. Dec 2018 [cit. 2021-03-28]. Dostupné z: https://blog.zenggyu.com/en/post/2018-12-16/an-introduction-to-evaluation-metrics-for-object-detection/.

[40] Zou, Z., Shi, Z., Guo, Y. a Ye, J. Object Detection in 20 Years: A Survey. 2019. Dostupné z: https://arxiv.org/abs/1905.05055.