VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

DETEKCE TRAMVAJE VE VIDEU POMOCÍ NEURONOVÝCH SÍTÍ

TRAM DETECTION IN VIDEO BY NEURAL NETWORK

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **VOJTĚCH GOLDA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. TOMÁŠ DYK**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

148042

Ústav:	Ústav inteligentních systémů (UITS)
Student:	**Golda Vojtěch**
Program:	Informační technologie
Specializace:	Informační technologie
Název:	**Detekce tramvaje ve videu pomocí neuronových sítí**
Kategorie:	Zpracování obrazu
Akademický rok:	2022/23

Zadání:

1. Prostudujte základy zpracování obrazu. Zaměřte se zejména na problematiku obecné detekce objektů pomocí neuronových sítí.
2. Vyberte vhodné architektury neuronových sítí a navrhněte řešení problému detekce tramvaje z videozáznamu.
3. Navržené řešení implementujte v programovacím jazyce Python.
4. Experimentujte s vaší implementací a případně navrhněte vlastní modifikace zvolených architektur neuronových sítí.
5. Porovnejte dosažené výsledky a navrhněte možnosti budoucího vývoje.

Literatura:

- ACHARYA, Tinku; RAY, Ajoy K. Image processing: principles and applications. John Wiley & Sons, 2005.
- S. Albawi, T. A. Mohammed and S. Al-Zawi, "Understanding of a convolutional neural network," 2017 International Conference on Engineering and Technology (ICET), 2017, pp. 1-6, doi: 10.1109/ICEngTechnol.2017.8308186.
- SZEGEDY, Christian; TOSHEV, Alexander; ERHAN, Dumitru. Deep neural networks for object detection. 2013.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 a 2

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

Vedoucí práce:	**Dyk Tomáš, Ing.**
Vedoucí ústavu:	Hanáček Petr, doc. Dr. Ing.
Datum zadání:	1.11.2022
Termín pro odevzdání:	10.5.2023
Datum schválení:	3.11.2022

Abstrakt

Tato práce se zabývá detekcí tramvají ve videu konvolučními neuronovými sítěmi. Jejich základní princip fungování je popsán. Je vytrénována řada různých architektur. Užitečnost výsledných modelů je následně porovnána. Výstupem je program schopný detekce tramvaje ve videu.

Abstract

This paper deals with tram detection in video using convolutional neural networks. The basic principles of their function are described. A number of distinct architectures are trained. The usefulness of the resulting models is subsequently compared. The output of this paper is a program capable of detecting trams in video.

Klíčová slova

detekce objektů, neuronová sít, konvoluční neuronová sít, zpracování obrazu, detekce tramvaje, strojové učení, PyTorch, gradientní sestup

Keywords

object detection, neural network, convolutional neural network, image processing, tram detection, machine learning, PyTorch, gradient descent

Citace

GOLDA, Vojtěch. *Detekce tramvaje ve videu pomocí neuronových sítí.* Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Dyk

Detekce tramvaje ve videu pomocí neuronových sítí

Prohlášení

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

.....
Vojtěch Golda
29. července 2023

Poděkování

Chtěl bych poděkovat svému vedoucímu Ing. Tomáši Dykovi za odborné a trpělivé vedení. Dále děkuji svojí rodině za podporu při psaní práce.

Obsah

Kapitola 1

Úvod

Městská hromadná doprava je efektivním způsobem přepravy lidí. Tramvaje specificky přináší řadu výhod oproti autobusu a trolejbusu, jako například větší kapacitu, větší bezpečnost cestujících, anebo možnost rychlé reakce na dopravní situaci přidáváním a odebíráním vozů soupravě. Navzdory těmto a dalším výhodám se nachází v jejich využití prostor pro automatizaci. Tím je například detekce obsazenosti jednotlivých kolejí v depu nebo detekce chybné pozice tramvaje. Jedním ze způsobů získání pozice tramvaje, alespoň pro kontext lokálního prostoru, je využití videokamery. Na automatické zpracovaní pro tento účel pořízených snímků je aplikovatelná řada postupů, z nichž předmětem této práce je řešení konvolučními neuronovými sítěmi.

Cílem této práce je rozšiřitelný základ automatizace založený na detekci tramvají ve videu. Takového rozšíření bude dosažitelné jak úpravou zdrojového kódu vzniklého programu, tak napojením programů nových na jeho výstup. V kapitole 2 je popsána funkce a operace konvolučních neuronových sítí, v kapitole 3 specifikace a návrh cílového programu a v kapitole 4 jeho implementace a výsledky testů.

Kapitola 2

Konvoluční neuronové sítě

Některé z problémů, které jsou žádoucí, aby se daly vyřešit počítačem, by se klasickým algoritmickým způsobem řešily velmi těžce. Příkladem jsou problémy popsané v [14], například:

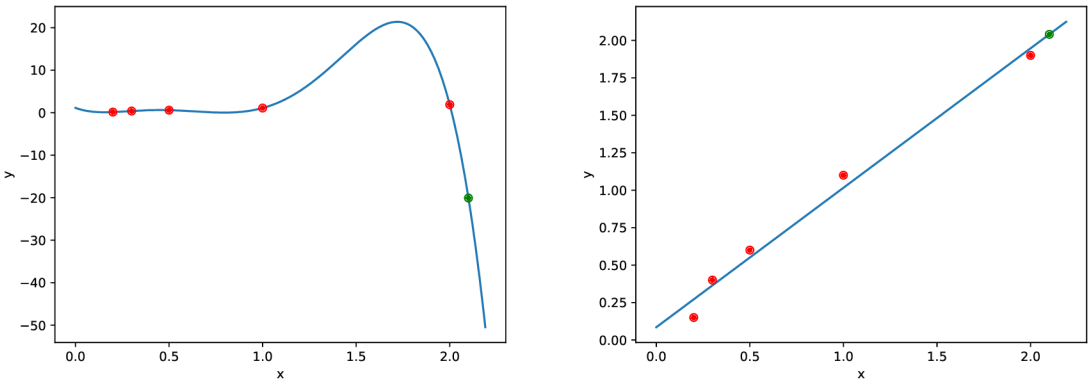
- Nachází se v daném obrázku osoba, jíž hledám?
- Jak se daná akcie zachová v budoucnosti?
- Patří opravdu tento podpis osobě, která je podepsaná?

Alternativním přístupem strojové učení.

2.1 Strojové učení

Strojovým učením se pro účely této práce rozumí soubor metod a algoritmů určených k využití existujících dat k předpovědi nových informací. Jeho výstupem je funkce $f(x)$ jejíž výsledkem je y, kde x je jednotka dat a y je jim přiřazená odpověď. y může být cokoliv, jako například klasifikace do skupin, nově vygenerovaná data a podobně. Zatímco klasickým způsobem by byla nejprve vytvořena funkce a ta až pak aplikovaná na data, zde je započato s množinou dat x a jim odpovídající množinou y a hledaná funkce je vytvořená automaticky. Tomuto procesu se říká *trénování*. Data jsou typicky před započetím trénování rozdělena na nejméně dvě disjunktní podmnožiny, z nichž jedna se stane datovou sadou *trénovací* a druhá datovou sadou *testovací*. [4]

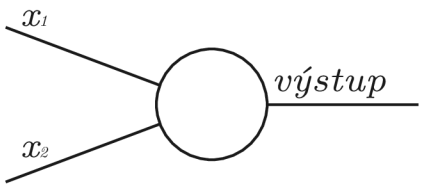
Nejpřímočařejším způsobem, jak dosáhnout strojového učení, je vypočítat z trénovacích dat polynom tato data přesně protínající. To je ale velmi vzácně dostačující řešení. Například pokud mají x a y lineární vztah, ale data jsou naměřená s drobnými nepřesnostmi, vznikne polynom zcela odtržený od reality, jak je vidět na obrázku 2.1. V takovém případě je lepší využít *lineární regrese*. Lineární regrese je akt přiřazení přímky souboru bodů v grafu. Jedním ze způsobů, jak tohoto dosáhnout je využití metody nejmenších čtverců. Tato metoda měří příslušnost dané přímky danému souboru bodů součtem druhých mocnin vzdáleností všech bodů od přímky. Pro přímku podle $f(x) = mx + n$ jsou hledanými hodnotami m a n. Položením částečných derivací výpočtu „ceny" vzhledem k těmto hodnotám nule vznikne soustava rovnic, jejíž vyřešením získáme nejoptimálnější přímku. Na obrázku 2.1 je porovnán příklad souboru bodů proložen přesným polynomem a přímkou získanou metodou nejmenších čtverců spolu s předpovědí odpovídající novému x_i. Metodu lze zobecnit i pro křivky různých tvarů, jako například v případě polynomiální regrese. Relevantními termíny jsou zde *bias* a *variace*, kde bias označuje přesnost řešení pro vzorová



(a) Interpolace přesným polynomem.

(b) Metoda nejmenších čtverců.

Obrázek 2.1: Ilustrace různých metod interpolace stejných dat. Zelený bod je bod odhadnutý danou metodou. I bez znalosti přesného původu dat je zřejmé, že jedna z těchto metod je pro účely strojového učení značně užitečnější.



Obrázek 2.2: Neuron

data a variace označuje rozdíl přesnosti řešení pro zdrojová data s přesností pro data dosud neviděná. Na příkladu 2.1 je vidět, že ačkoliv má první řešení nulový bias, má velmi vysokou variaci, zatímco druhé řešení má relativně nízký bias a variaci. [4]

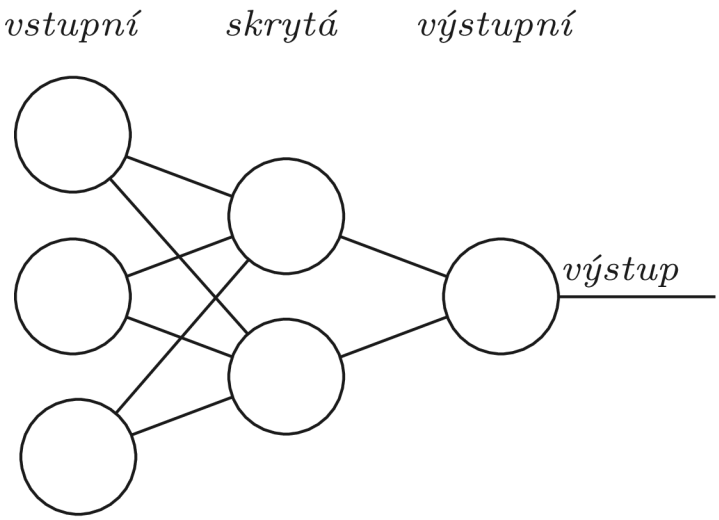
2.2 Neuronové sítě

Jedná se o model inspirovaný reálnými neurony. Neuron má množinu vstupů $x_1, x_2, x_3, \ldots$, které nabývají hodnot z intervalu <0, 1>. S těmi provede matematickou operaci (např. výběr maxima, vypočtení průměru apod.) a její výsledek pošle dál jako svůj výstup, jak je ilustrováno na obrázku 2.2. Nejčatější operací, kterou neuron se vstupy provádí, je následující: každý neuron má jeden bias b a pro každý vstup $x_1, x_2, x_3, \ldots$ váhy $w_1, w_2, w_3, \ldots$ příslušně. Výstup je vypočten podle $\sigma(\sum_j w_j x_j + b)$, kde σ je nelineární funkce, jako například sigmoidní funkce 2.1. V této práci je tato operace použitá všude, kde není uvedeno jinak. [25]

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.1}$$

Jiné nelineární funkce jako například tanh, softplus nebo ReLU mohou a bývají použity. Účelem využití nelinearity je získání schopnosti modelovat nelineární vztahy — skládáním lineárních funkcí je možné získat pouze funkce, které jsou taktéž lineární. [1, 17]

Uspořádáním takových neuronů do *vrstev* a následném propojením těchto vrstev tak, že výstupy jedné vrstvy jsou vstupy vrstvy další (obrázek 2.3), vznikne neuronová síť. Na



Obrázek 2.3: Plně propojená neuronová síť. Označení jednotlivých vrstev se nachází nad nimi.

všech výstupech daného neuronu je stejná hodnota. První vrstvě se říká vrstva *vstupní* a poslední vrstva *výstupní*. Ostatní se nazývají vrstvami *skrytými*. Konvencí je vstupní vrstvu do celkového počtu vrstev nezapočítávat. Síť na obrázku 2.3 by tudíž měla svůj počet vrstev udán jako 2. Účelem vstupní vrstvy je modelace vstupů simulací každé vstupní hodnoty jako neuronu. Dále se vrstvy rozdělují na *plně propojené* a ostatní. Vrstva plně propojená je taková vrstva, jejíž každý neuron je propojený s každým neuronem v předchozí a následující vrstvě. Neuronové sítě určeně pro rozpoznávání a klasifikaci obrazových dat se typicky na obrázek „dívají" vstupní vrstvou uvažováním každého pixelu jako jedné vstupní hodnoty. Vícedimenzionální pole hodnot je tudíž efektivně zpracováváno jako jednodimenzionální. [25] Pro zjednodušení se často používá vektorová notace. Vektor vstupů do sítě bývá označován jako x a má tolik dimenzí, kolik má síť vstupů - každému vstupu náleží jeden prvek vektoru. Nápodobně správné výstupy se shlukují do vektoru označovaném jako $y = y(x)$ pro daný vektor x.

Pro kvantifikaci kvality sítě je vhodné nadefinovat *ztrátovou funkci* C, jako například kvadratickou cenu (rovnice 2.2),

$$C(w, b) \equiv \frac{1}{2n} \sum_x \parallel y(x) - a \parallel^2 \tag{2.2}$$

kde $\parallel$ je operace délky vektoru a a je vektor výsledků výstupní vrstvy. Pro optimálnější neuronovou síť bude mít pro daná data ztrátová funkce menší hodnotu. Sítě jsou optimalizovány snižováním hodnoty ztrátové funkce upravováním specifických hodnot parametrů jako například vah a biasů. Parametrům, jež nejsou optimalizovány zmenšováním ceny (jako například počet vrstev nebo volba nelinearity), se říká *hyperparametry*. [25]

2.2.1 Stochastický gradientní sestup

Gradientní sestup je metoda umožňující numerického nalezení jednoho z lokálních minim některých funkcí. Pro funkci $f(x_1, x_2, \ldots, x_n)$ a n-dimenzionální vektor x symbolizující bod počátku metody existuje n-dimenzionální vektor z určující směr nejstrmějšího klesání, po-

kud je funkce f v bodě x diferencovatelná. Individuální prvky vektoru z jsou dány rovnicí 2.3:

$$z_i = -\lambda \frac{\partial f}{\partial x_i} \tag{2.3}$$

Kde λ je pozitivní konstanta určující rychlost sestupu. Vektoru $(\frac{\partial f}{\partial x_0}, \frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n})$ se často říká *gradient* funkce f, značen jako ∇f. Rovnice 2.3 může tudíž být přepsána jako $z = -\lambda \nabla f$. Opakovanou aktualizací vektoru x podle rovnice 2.4 dojde k postupnému posunu směrem k lokálnímu minimu. [6]

$$x_{k+1} = x_k - \lambda \nabla f_k \tag{2.4}$$

Pro neuronovou síť s váhami w a biasy b, vektorem cvičných vstupů x s náležitým vektorem výstupů y a ztrátovou funkcí $C(w, b)$ by gradientní sestup probíhal opakováním následujících kroků:

- vypočtením ztráty všech cvičných vstupů
- vypočtením všech částečných derivací $\frac{\partial C}{\partial w_i}$ a $\frac{\partial C}{\partial b_i}$
- upravením vah a biasů podle rovnic 2.5 a 2.6

$$w_{k+1} = w_k - \lambda \frac{\partial C}{\partial w_k} \tag{2.5}$$

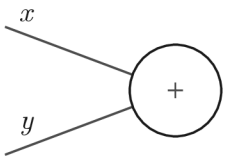
$$b_{k+1} = b_k - \lambda \frac{\partial C}{\partial b_k} \tag{2.6}$$

Postup je opakován, dokud nedojde ke splnění předem určené podmínky, jako například velikost rozdílu mezi optimalizovanými proměnnými jednotlivých iterací. Náročnost výpočtu $\frac{\partial C}{\partial w}$ a $\frac{\partial C}{\partial b}$ se s počtem vstupů x_j (kde j je index specifického cvičného vstupu) však zvedá velmi rychle. Může být efektivnější využít alternativní verze gradientního sestupu, a to *gradientní sestup stochastický*. Místo výpočtu částečné derivace $\frac{\partial C}{\partial w}$ a $\frac{\partial C}{\partial b}$ vzhledem ke všem cvičným vstupům najednou, cvičné vstupy jsou rozděleny do várek a gradientní sestup je aplikován postupně na jednu po druhé. Ačkoliv jednotlivé gradienty neodpovídají přesně gradientu skutečnému, jejich spojením vznikne vlivem zákonů pravděpodobnosti gradient ukazující přibližně správným směrem. [2, 25]

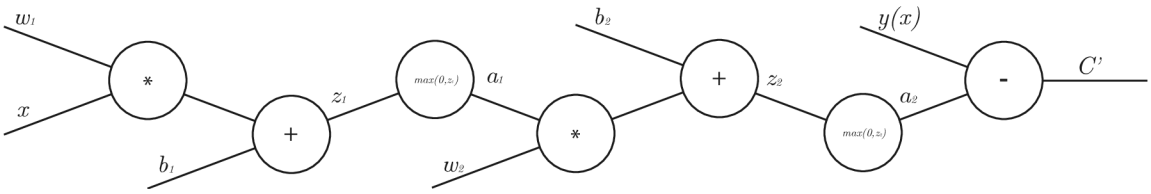
Pro zjednodušení analytického výpočtu $\frac{\partial C}{\partial w}$ a $\frac{\partial C}{\partial b}$ je vhodné využít takzvaného *výpočetního grafu* společně s *algoritmem zpětného šíření chyby*. Výpočetní graf je způsob reprezentace matematické funkce, kde uzly reprezentují jednoduché operace a hrany reprezentují proměnné, jež do operace vstupují. Například graf funkce $f(x, y) = x + y$ by vypadal jako na obrázku 2.4. Graf jednoduché neuronové sítě s jednou skrytou vrstvou, kde se v každé vrstvě nachází pouze jeden neuron s aktivační funkcí zvolenou ReLU a kvadratickou cenou, by vypadal jako na obrázku 2.5. Místo celého výpočtu je znázorněna jenom jeho část pro jeden vstup x_j, jejíž výsledkem je část ceny C_j. Pokud pro zvolenou cenu platí rovnice 2.7 (což pro kvadratickou cenu platí),

$$\sum_j \nabla C_j = \nabla C \tag{2.7}$$

pak je algoritmus zpětného šíření chyby je možné provádět pro každý cvičný vstup x_j zvlášť a jednotlivé přírůstky pak sečíst. Vstupem uzlu jednoduché operace může být také vektor



Obrázek 2.4: Jednoduchý výpočetní graf znázorňující výpočet $f(x, y) = x + y$. Uzel symbolizuje operaci sčítání zatímco jeho dvě hrany symbolizují proměnné x a y.



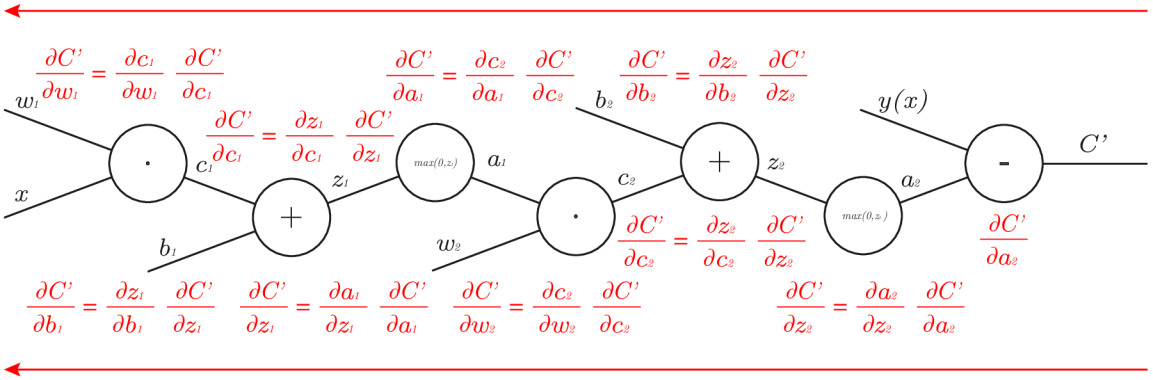
Obrázek 2.5: Výpočetní graf znázorňující výpočet části ceny neuronové sítě s jednou skrytou vrstvou, kde se v každé vrstvě nachází jeden neuron.

nebo matice, jako na obrázku 2.6, kde je znázorněna podobná síť jako výše uvedená s rozdílem, že místo jednoho neuronu se v každé vrstvě nachází neurony tři, společně s postupem algoritmu zpětného šíření chyby. Tento algoritmus nejprve vypočte dopředným směrem část ceny C_j pro daný vstup x_j a poté za pomocí řetězového pravidla (rovnice 2.8) vypočte zpětně jednu po druhé všechny částečné derivace.

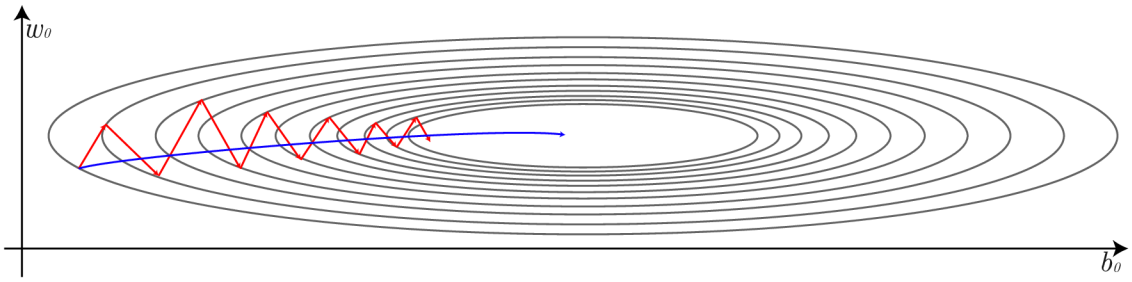
$$\frac{\partial f(g(x))}{\partial x} = \frac{\partial z}{\partial x}\frac{\partial f(z)}{\partial z} \tag{2.8}$$

V uvedeném přikladu jsou specificky zajímavými $\frac{\partial C_j}{\partial w_1}$, $\frac{\partial C_j}{\partial w_2}$, $\frac{\partial C_j}{\partial b_1}$ a $\frac{\partial C_j}{\partial b_2}$, jelikož společně tvoří ∇C_j. [25]

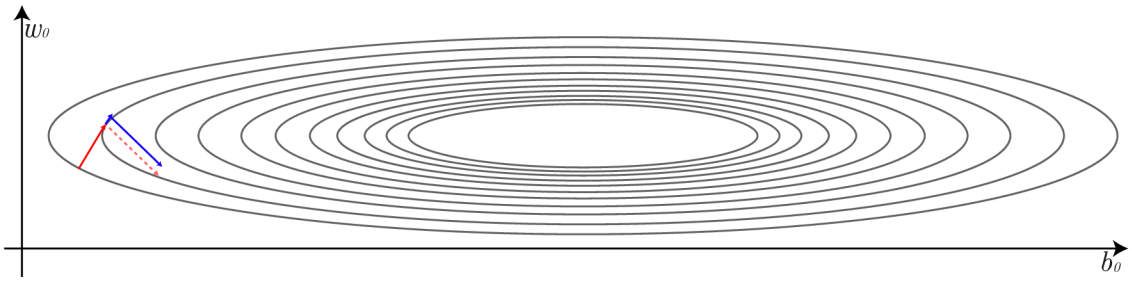
Pro některé tvary funkcí nemusí být čistý stochastický gradientní sestup tím nejefektivnějším způsobem nalezení lokálního minima. Příkladem je ilustrace na obrázku 2.7, kde je znázorněna kontura funkce spolu s možným optimálním sestupem a sestupem typickým pro tento algoritmus. Jelikož stochastický gradientní sestup stráví velké množství času pohybem jiným než nejoptimálnějším směrem, nachází se zde prostor pro zlepšení. Jedním ze



Obrázek 2.6: Výpočetní graf znázorňující výpočet části ceny neuronové sítě s jednou skrytou vrstvou, kde se v každé vrstvě nachází tři neurony, spolu s relevantními částečnými derivacemi. Šipky ukazují směr postupu algoritmu zpětného šíření chyby.



Obrázek 2.7: Kontura funkce a proces optimalizace neuronové sítě skládající se z jedné vrstvy o jednom neuronu. Červenou barvou je znázorněn možný obyčejný stochastický sestup a modrou možné optimálnější nalezení minima.



Obrázek 2.8: Kontura z obrázku 2.7 s prvními kroky algoritmu stochastického gradientního sestupu s momentem. Přerušovanou čarou je vyznačen druhý krok bez momenta a modrou je vyznačen druhý krok s momentem. Tato modifikace nemění velikost ani směr kroku, pouze místo, odkud je proveden.

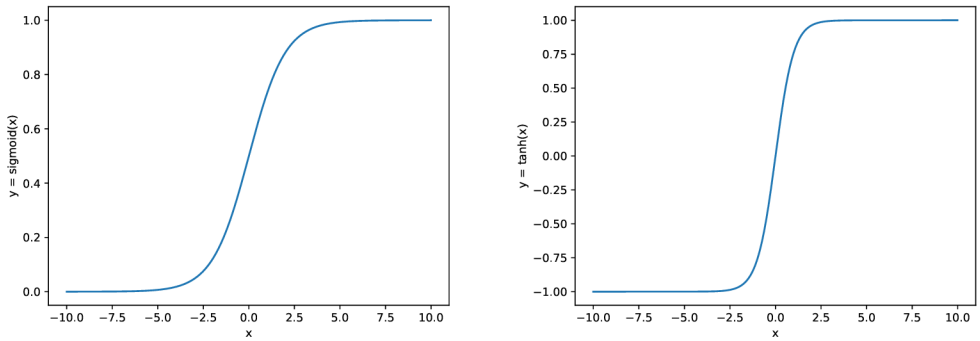
způsobů, jak tohoto dosáhnout, je do rovnice 2.4 přidat tzv. *momentum* člen jak je vidět v rovnici 2.9,

$$\begin{aligned} x_{k+1} &= x_k - \lambda V_k \\ V_k &= \nabla f_k + \beta V_{k-1} \end{aligned} \tag{2.9}$$

kde β je konstanta z intervalu <0,1) určující význam momenta v sestupu. Z rovnice 2.9 vyplývá $V_k = \beta^{k-1}\nabla f_1 + \beta^{k-2}\nabla f_2 + \cdots + \beta\nabla f_{k-1} + \nabla f_k$. Algoritmus si tudíž pamatuje každé využité ∇f, ale jelikož β je konstanta z intervalu <0,1), vliv každého ∇f na výsledný přírůstek exponenciálně klesá s každou iterací. Momentum přidá do algoritmu jistou setrvačnost, „uhlazující" cestu, jež algoritmus využije k dosažení lokálního minima, jak je ilustrováno na obrázku 2.8. Pro neuronovou síť, kde optimalizovanými parametry jsou váhy a biasy jednotlivých neuronů, jsou rovnice 2.5 a 2.6 upraveny na 2.10 a 2.11 příslušně. [28]

$$\begin{aligned} w_{k+1} &= w_k - \lambda V_k^w \\ V_k^w &= \frac{\partial C}{\partial w_k} + \beta V_{k-1}^w \end{aligned} \tag{2.10}$$

$$\begin{aligned} b_{k+1} &= b_k - \lambda V_k^b \\ V_k^b &= \frac{\partial C}{\partial b_k} + \beta V_{k-1}^b \end{aligned} \tag{2.11}$$



Obrázek 2.9: Grafy využívaných aktivačních funkcí.

Alternativou k momentu je *RMSprop* (z anglického *root mean squared propagation*). RMSprop vznikne vydělením ∇f_k variantou kvadratického průměru všech předchozích ∇f, jak je vidět v rovnici 2.12,

$$\begin{aligned} x_{k+1} &= x_k - \lambda \nabla f_k / \sqrt{S_k + \epsilon} \\ S_k &= (\nabla f_k)^2 + \beta S_{k-1} \end{aligned} \tag{2.12}$$

kde ϵ je velmi malá konstanta zabraňující dělení čísly velmi blízkými nule a β je podobně jako výše konstanta z intervalu <0,1) určující rychlost zániku předchozích $(\nabla f_k)^2$. [12]

Kombinací dvou výše uvedených metod vznikne metoda *Adam* (*adaptive moment estimation*) - rovnice 2.13:

$$\begin{aligned} x_{k+1} &= x_k - \lambda V_k / \sqrt{S_k + \epsilon} \\ V_k &= \nabla f_k + \beta_0 V_{k-1} \\ S_k &= (\nabla f_k)^2 + \beta_1 S_{k-1} \end{aligned} \tag{2.13}$$

kde β_0 a β_1 jsou konstanty určující rychlost zániku příslušných členů. Tato metoda má ze všech uvedených nejlepší optimalizační schopnost. [15]

2.2.2 Významné aktivační funkce

Aktivační funkce uvedená v rovnici 2.1 se v dnešní době již používá jen velmi zřídka. Výstup sigmoidní funkce je v celém svém průběhu pozitivní, což z aritmetických důvodů způsobuje

stejnost znaménka gradientu parametrů daného neuronu, a tudíž má negativní efekt na rychlost učení. Řešením je ji nahradit za *tanh* (obrázek 2.9b). Co ale nevyřeší ani *tanh* je fakt, že obě funkce jsou, jak je vidět na obrázcích 2.9a a 2.9b, v jakékoliv nezanedbatelné vzdálenosti vstupu od nuly velmi ploché, důsledkem čehož je v těchto částech funkce $\sigma'(x) \approx 0$. Jelikož jedním z efektů opakované aplikace řetězového pravidla je časté násobení průběžného výsledku částečnou derivací těchto funkcí, jakýkoliv neuron nacházející se svým výstupem v tomto prostoru dosáhne stavu *nasycenosti* — již se dále skoro neučí. [5, 35]

Populární aktivační funkcí je v současné době $ReLU(x) = max(0, x)$. Funkce, jak je vidět na obrázku 2.9c, nemá svoji derivaci alespoň pro $ReLU(x) > 0$ nikdy blízkou nule. Dle [18] je ReLU má velmi nízkou výpočetní složitost a má potenciál i několikrát zrychlit celý proces, nicméně trpí stejnou pozitivností jako sigmoidní funkce, a navíc existuje šance, že během učení vlivem ploché části $ReLU(x) = 0$ daný neuron „umře" — gradient posune jeho parametry do oblasti, kde se už nikdy neaktivuje. *Aktivace* v kontextu ReLU znamená pro daný neuron, že jeho výstup je větší než nula. V síti hojně využívajících tuto aktivační funkci může její část sestávající z takto „mrtvých" neuronů dosáhnout velikosti až desítek procent. Leaky ReLU ($f(x) = 1(x < 0)(\alpha x) + 1(x >= 0)(x)$, kde α je parametr určující sklon $f(x) <= 0$) ilustrována na obrázku 2.9d je pokusem o řešení těchto problémů, ale její konzistentní nadřazenost je neprůkazná. [3, 5]

Existuje také aktivační funkce, jež se dá považovat za zobecnění ReLU, tzv. *maxout*. Jak je vidět v rovnici 2.14,

$$h(x) = max(w_1 \cdot x + b_1, w_2 \cdot x + b_2, \ldots, w_j \cdot x + b_j) \tag{2.14}$$

maxout využívá více než jedné dvojice vektoru vah a biasu, což mu dává vybráním maxima z množiny lineárních funkcí schopnost se přiblížit velkému (až jakémukoliv — v případě jejich dostatečné kvantity) množství nelineárních funkcí. Maxout tak umožnuje neuronové síti nejen se naučit váhy a biasy, ale i aktivační funkci individuálně pro každý neuron. ReLU vznikne zvolením množství linearit na dva a následném nastavení vah i biasu jedné z linearit na nula. Navzdory obecnosti maxoutu jeho vysoká výpočetní složitost jen vzácně stojí za to. Obyčejná ReLU tak zůstává nejvšestrannější volbou aktivační funkce. [10]

V některých případech je žádoucí, aby výstup poslední vrstvy byl vyjádřením pravděpodobnosti, jako například pravděpodobnosti že daný vstup patří do dané kategorie. Jednou z nezbytně nutných vlastností takové vrstvy je, že součet výstupů všech neuronů je jedna, což žádná z dosud uvedených aktivačních funkcí nesplňuje. Jednou z často využitých aktivačních funkcí, jež tohle splňuje, je tzv. *softmax*, uveden v rovnici 2.15:

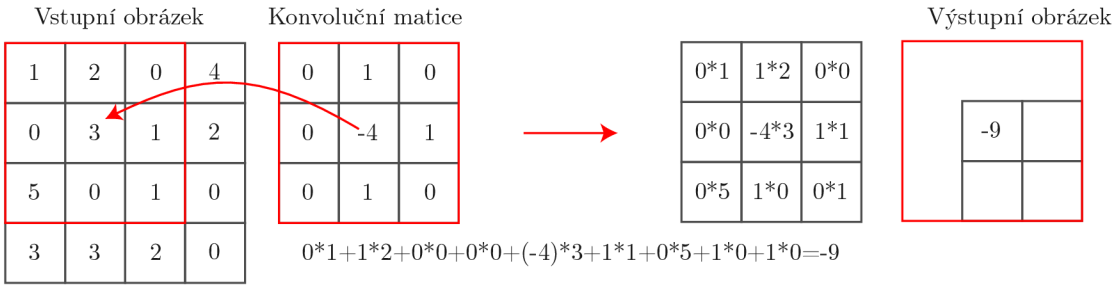
$$a_j = \frac{e^{z_j}}{\sum_k e^{z_k}} \tag{2.15}$$

kde z_k je vážený součet vstupů daného neuronu a k je počet neuronů v dané vrstvě. Výstupy softmax neuronů jsou, jak je vidět v rovnici 2.15, vždy pozitivní (e^x je pozitivní v celém svém průběhu) a tak se vlivem předešle zmíněné vlastnosti dá výstup takové vrstvy považovat za distribuci pravděpodobnosti. [18, 25]

2.3 Konvoluce a neuronové sítě

Konvoluce diskrétních funkcí $f(i, j)$ a $g(i, j)$ je vypočtena rovnicí 2.16. [13]

$$(f \star g)(i, j) = \sum_k \sum_p f(k, p) g(i - k, j - p), \text{ pro } i, j \in \mathbb{Z} \tag{2.16}$$



Obrázek 2.10: Efekt konvoluce na specifický pixel. Pixely na okrajích nelze zkonvulovat a proto je výsledný obrázek menší.

V kontextu počítačové grafiky $f(i,j)$ představuje vstupní obrázek, $g(i,j)$ představuje konvoluční matici a $\star$ je operace konvoluce. Proces konvoluce aplikuje matici na každý pixel, kde sečte násobky pixelů a příslušných hodnot v matici. Hodnoty v konvoluční matici představují váhy vlivu jednotlivých pixelů na pixel výsledný [7]. Výsledek je novým pixelem na zpracovávané pozici. Pro obrázek a konvoluční matici je výpočtem rovnice 2.17 [1]

$$(x \star w)[i,j] = \sum_m \sum_n x[m,n]w[i-m, j-n] \tag{2.17}$$

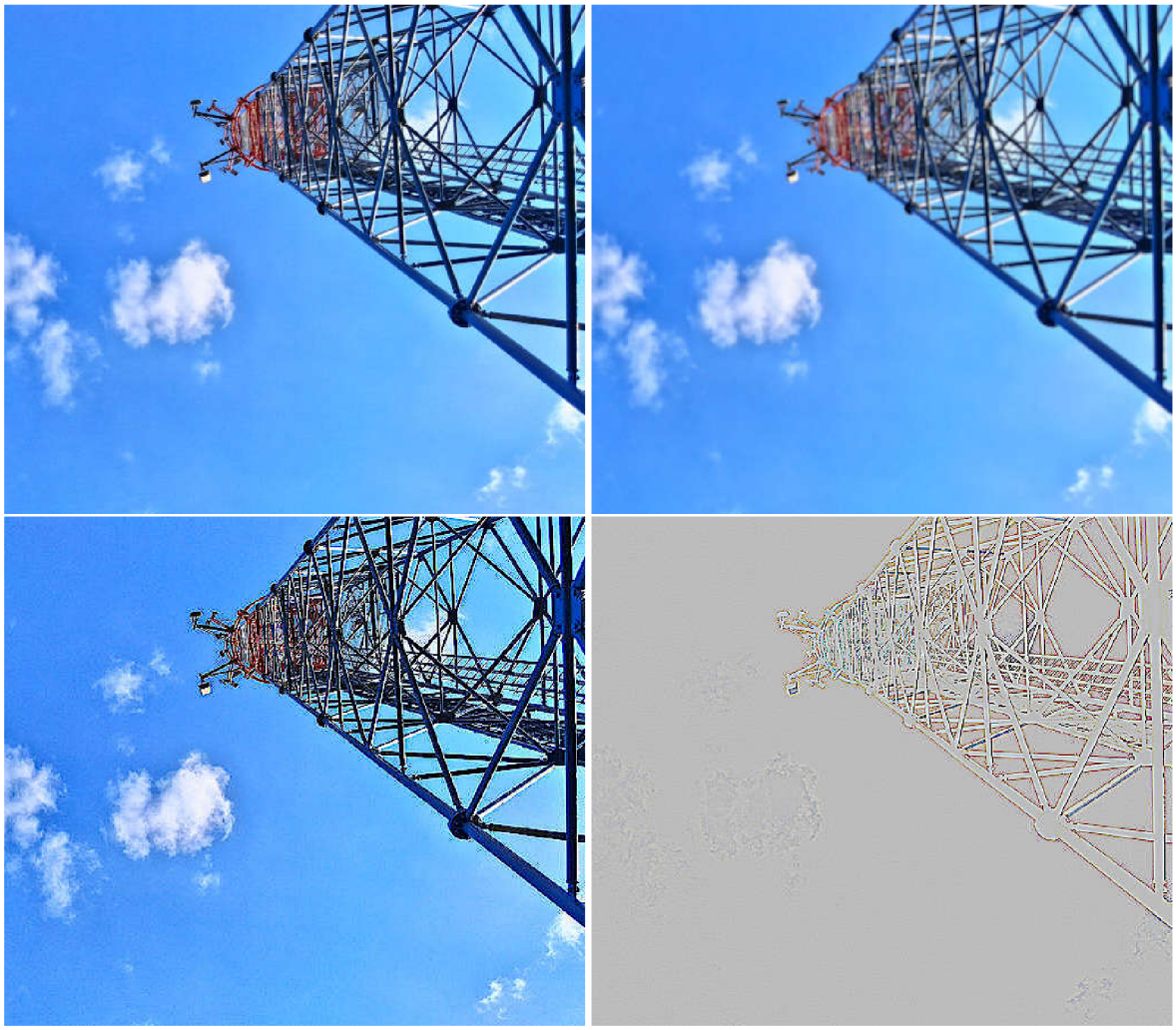
kde x je vstupní obrázek a w konvoluční matice. Obrázek 2.10 znázorňuje tento proces graficky. Obrázky 2.11 znázorňují efekty některých konvolučních matic. Ačkoliv striktně matematická definice konvoluce konvoluční matici před aplikováním převrací, pro účely této práce je tento fakt pro zjednodušení ignorován, protože váhy jsou naučené automaticky, a tak to na výsledku nic důležitého nemění. [13]

Ačkoliv pro malé sítě plná konektivita nezpůsobuje žádné problémy, se zvětšujícím se počtem neuronů nárust parametrů neúnosně zvyšuje náročnost na výpočet. Například pro klasifikaci obrázků o velikosti $32 \times 32 \times 3$ pixelů (třetí rozměr je hloubka, tj. počet RGB kanálů), připojení vstupní vrstvy k pouze jednomu výstupnímu neuronu si vyžádá 3072 vah, připojení ke dvěma 6144 a tak dále. Pro skrytou vrstvu o velikosti 32×32 si plná konektivita vyžádá vah $32 \times 32 \times 32 \times 32 \times 3$ (3145728). [1]

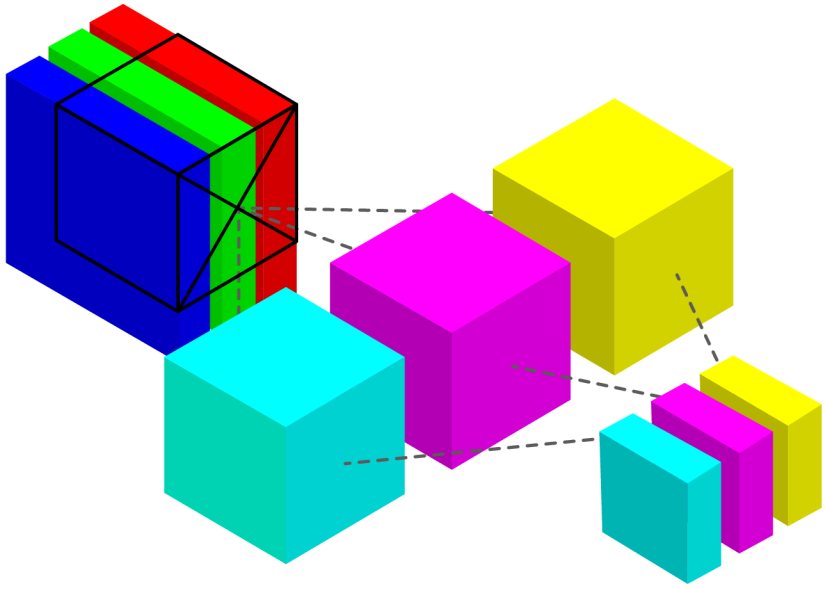
Dobrým způsobem, jak tohle množství omezit, je nepřipojovat každý skrytý neuron ke každému vstupnímu a místo toho „ukázat" každému skrytému neuronu pouze malou část vstupu například o velikosti 5×5. Počet vah se tak zmenší na $5 \times 5 \times 3 \times 32 \times 32$ (76800). Tohle množství může být ještě dále zmenšeno použitím stejných vah pro všechny neurony skryté vrstvy, zmenšujíc počet vah na $5 \times 5 \times 3$ (75). Skrytá vrstva se tak stane vrstvou konvoluční, protože taková vrstva je ekvivalentní malému okýnku počítající vážený průměr s identickými vahami pro všechny pixely a jejich okolí. Této množině vah se často říká *jádro*. Je možné mít více jader zpracovávající jednu část obrázku. Každé takové jádro vyprodukuje vlastní takzvanou *aktivační mapu*. Proces je ilustrován na obrázku 2.12. Síť tak získá schopnost hledat malé rysy, nezávisle na tom, kde se ve vstupním obrázku nachází. [1]

Jádro může nabývat různých rozměrů, kromě výše uvedeného $5 \times 5 \times 3$ například $5 \times 5 \times 1$ (pro šedotónový obrázek) nebo $7 \times 7 \times 3$. Při rozměrech větších než $1 \times 1 \times 1$ není možné provést konvoluci obrazových dat na okrajích, jelikož nejsou žádné pixely, jež je možné přiřadit některým částem jádra, například jak je vidět na obrázku 2.10. To se dá řešit způsoby jako např.:

- Neaplikováním jádra na okrajích. Výsledná aktivační mapa pak bude mít menší rozměry, a to přesně o okraj, který se nepodařilo zkonvulovat. Z výše uvedeného příkladu



Obrázek 2.11: Příklady různých konvolučních matic, v pořadí: identita $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$, rozmazání $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$, zostření $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ a detekce hran $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$.



Obrázek 2.12: Ilustrace konvoluce vícekanálového obrázku více než jedním jádrem. Vzniklé aktivační mapy je možné vnímat jako kanály nového obrázku pro účely například další konvoluční vrstvy. Kvádry barvy červená, modrá a zelená představují jednotlivé kanály vstupního obrázku. Jednotlivé krychle představují různá jádra velikosti $3 \times 3 \times 3$ a příslušně nabarvené kvádry představují výsledné aktivační mapy. Celkový počet násobení pro jeden krok je $3 \times 3 \times 3 \times 3$ (81).

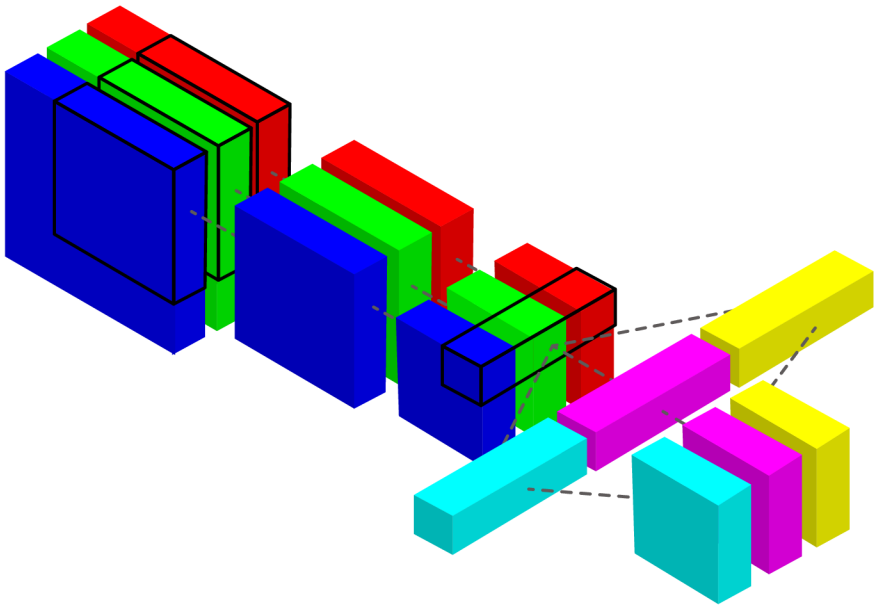
obrázku o rozměrech $32 \times 32 \times 3$ a jádra $5 \times 5 \times 3$ by vznikla aktivační mapa o rozměrech $28 \times 28 \times 1$.

- Rozšířením obrázku o okraj falešný, jako například naplněný nulami, anebo zkopírováním okraje skutečného.

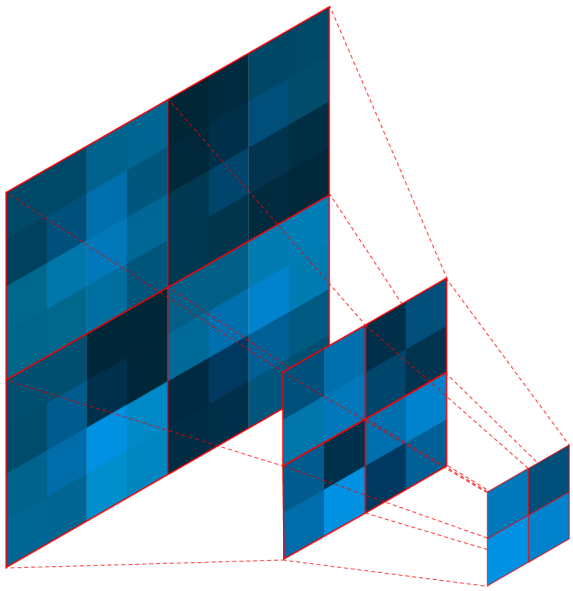
Není také nezbytně nutné konvulovat každý pixel. Konvoluční jádro je možné posouvat o hodnotu kroku větší než 1. V takovém případě je ale potřeba zvolit hyperparametry tak, aby bylo možné zkonvulovat celý obrázek. S obrázkem například o velikosti $7 \times 7 \times 3$ a jádrem $3 \times 3 \times 3$ by krok o velikosti 3 fungoval jen velmi těžko. [1]

V neuronových sítích obsahující konvoluční vrstvy se často nachází také vrstvy typu *max pooling*. Taková vrstva je, podobně jako konvoluční, ekvivalentní pohybujícímu se okýnku s rozdílem, že místo váženého součtu vrstva vybere maximum. Max pooling se setkává s identickými problémy s okraji s identickými řešeními. Často použitou konfigurací je například max pooling s velikostí 2×2, krokem 2×2 a ignorací okrajů. Efekt dvou takových vrstev je ilustrován na obrázku 2.14. Kromě vylepšení výkonnostní složitosti je jednou z vlastností takové a jí podobné max pooling vrstvy redukce sekcí vstupu na jednotlivé hodnoty. Tyto hodnoty je pak možné zpětně namapovat na sekce vstupu a vyvozovat z nich závěry. [20]

Ještě dalším způsobem, jak snížit výpočetní náročnost pro obrázky s víc než jedním barevným kanálem, je místo trojrozměrných jader použít množinu jader dvojrozměrných. V takovém případě bude každému kanálu přiřazeno n dvourozměrných jader a pro m kanálů vznikne $n \times m$ aktivačních map tak, že z každého kanálu vznikne n aktivačních map. Tyto



Obrázek 2.13: Stejná konvoluce jako v obrázku 2.12 s rozdílem, že v tomto případě je využito depthwise separation. Místo tří jader $3 \times 3 \times 3$ je zde využito tří jader velikosti $3 \times 3 \times 1$. Každé z nich je aplikováno na vlastní kanál. Na výslednou množinu aktivačních map jsou následně aplikována 3 jádra velikosti $1 \times 1 \times 3$, z nichž každé vyprodukuje vlastní aktivační mapu. Celkový počet násobení pro jeden krok je $3 \times 3 \times 3 + 3 \times 3$ (36).



Obrázek 2.14: Výsledek dvou max pooling vrstev velikosti 2×2 a krokem 2×2. Jasnější pixel symbolizuje větší hodnotu. Vrstva vybere pixel nejvyšší hodnoty pro každou sekci 2×2. Výstup je dvakrát menší jak ve výšce, tak šířce. Po dvou takových vrstvách zbyde ze vstupu 8×8 pouze 2×2. Každý z těchto pixelů je možné považovat za souhrn informací sekce, ze které pochází, obzvlášť pokud se mezi max pooling vrstvami nachází i vrstvy konvoluční. Tyto sekce jsou vyznačeny červenými rámečky.

aktivační mapy jsou následně rozděleny do n skupin po m a po spojení každé skupiny do trojrozměrné aktivační mapy je na ně aplikováno libovolné množství jader tvaru $1 \times 1 \times m$. Použitím množství těchto jader nezanedbatelně větším, než n je možné hledat rysy v barevné informaci s menší výpočetní cenou, než jádry trojrozměrnými. Tato optimalizace se nazývá depthwise separation a je ilustrována na obrázku 2.13. [30]

2.4 Regularizace

Ačkoliv mají hluboké neuronové sítě velmi dobrou schopnost mapovat komplikované vztahy mezi vstupem a výstupem, při nekompletní datové sadě bude nezanedbatelná část těchto mapování odrážet vztahy nacházející se pouze v datové sadě trénovací. Tyto vztahy se v ní nachází vlivem náhodného rušení a její nekompletnosti, a tudíž jejich naučení snižuje obecnost modelu. Neuronové síti, jež se naučila vztahy a rysy přítomné pouze v trénovací sadě se říká *přetrénovaná*. Ve snaze zmírnit přetrénování neuronových sítí došlo k vyvinutí řady regularizačních technik. [32]

Pojem regularizace je pojem mající velmi neurčitý význam, a tak je třeba přijmout jednu z možných definic. Definicí z [19] je: Regularizace je jakákoliv dodatečná technika určená k vylepšení schopnosti generalizace.

2.4.1 Regularizace datovou sadou

S regularizací se dá začít už na úrovni dat, a to jak transformací existující datové sadě na novou, tak vytvořením nových dat z existujících. Uměle zvětšená datová sada se nazývá *augumentovaná*. Transformace, jejichž kombinovanou aplikací lze z jednotky obrazových dat uměle vytvořit jednotku dat novou, jsou například následující:

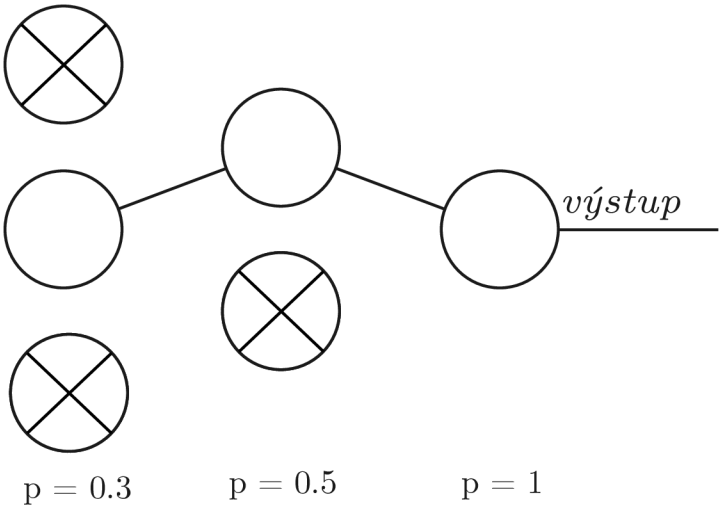
- aplikování náhodného šumu
- roztáhnutí
- rotace
- posunutí

Jejich využitím lze získat datovou sadu několikrát vetší. Je lepší raději získat více unikátních dat, nicméně to není vždy přijatelně snadné nebo cenově únosné. [19]

2.4.2 Regularizace včasným zastavením

Dalším způsobem předejití přetrénování je zastavení trénování ve správný čas. Při trénování při dostatečném počtu iterací dojde k situaci, kdy počet nově nalezených vztahů nacházející se pouze v trénovací sadě převýší počet nově nalezených vztahů platných pro celou řešenou doménu. Metodou měření, zdali k této situaci nedochází, je například periodické testování částí datové sady nevyužité k trénování. Pokud se kvalita sítě nad touto testovací sadou nelepší nebo dokonce zhoršuje, jedná se o možnou indikaci začínající přetrénovanosti sítě. Jelikož je ale testovací sada použita při trénování, objektivita měření finální kvality sítě touto sadou značně upadá. V praxi používaným řešením tudíž je rozdělení dostupné sady na tři části:

- trénovací — optimalizace sítě výpočtem ztráty a následném přepočtu parametrů



Obrázek 2.15: Plně propojená neuronová síť z obrázku 2.3 po aplikaci dropoutu. Příslušné hodnoty p individuálních vrstev jsou uvedeny pod nimi.

- validační — periodické měření relativní přetrénovanosti
- testovací — změření kvality sítě po ukončení trénování

Jedním z typických poměrů rozdělení datové sady je v pořadí například 7:1:2. [25]

2.4.3 Dropout

Jeden ze spíše zřejmých způsobů, jak se vyhnout efektům přetrénování, je vycvičit více než jednu síť a následně vyzkoušet nad testovacími daty sítě všechny. Optimálnějším způsobem, jak dosáhnout podobného efektu je takzvaný *dropout*. Jedná se o následující modifikaci algoritmu zpětného šíření chyby: Pro každou vrstvu je zvolen parametr $p \in <0,1>$. Před každou epochou má každý neuron $1 - p$ jeho vrstvy šanci, že se následující epochy nebude účastnit. (Ilustrováno na obrázku 2.15.) Na konci trénování jsou všechny váhy w vynásobeny jejich příslušným p. Výsledkem je přibližný ekvivalent průměru velkého množství individuálně vytrénovaných sítí. [32]

Aplikace dropoutu je pro každou epochu ekvivalentem sebrání „zřídčeného" vzorku z původní neuronové sítě, jeho individuálním vytrénování a následném zprůměrování jejich vah. Těchto vzorků existuje pro každou síť možných 2^n, kde n je počet neuronů v dané síti. Vzhledem k povaze růstu 2^n je při jakékoliv nezanedbatelné velikosti sítě nepravděpodobné, že bude jeden z těchto vzorků trénován více než jednou. Vzniklý průměr těchto síti je proto průměrem exponenciálního množství různých jednou trénovaných sítí s velkým množstvím sdílených vah. [32]

2.4.4 Regularizace cenovou funkcí

Jedním z důvodů pro výběr dané cenové funkce může být její regularizační efekt. Příbuznou možností je regularizace přidáním regularizačního členu do jinými prioritami zvolené cenové funkce, jako například způsobem uvedeným v rovnici 2.18,

$$C_r(x, y) = C(x, y) + r \tag{2.18}$$

kde r je zmíněný regularizační člen. Tímto členem může být například $\frac{\lambda}{2n}\sum_w w^2$ kde $\sum_w w_2$ je suma druhých mocnin všech vah, n je počet vzorků a λ je významnost tohoto regularizačního členu pro cenu. Protože cílem stochastického gradientního sestupu je minimalizace hodnoty ceny dané konfigurace neuronové sítě, přidáním sumy druhých mocnin vah do ceny dojde i k minimalizaci těchto vah. Preference menších vah má regularizační efekt. [19]

2.5 Zpracování obrazu

Obrazová data je často před jejich využitím pro účely strojového učení výhodné zpracovat. Častou transformací je transformace vyžadující znalost hodnoty pixelů na pozicích, jež nemají korespondující pozici ve zdrojovém obrázku. Příkladem může být zvětšení, zmenšení, anebo rotace. Tyto chybějící hodnoty je třeba vypočítat. K tomuto účelu existuje řada metod, z nichž nejjednodušší je metoda *nearest neighbour*. Tato metoda vypočítá každý neznámý pixel změřením vzdálenosti ke každému pixelu známému. Daný neznámý pixel následně získá hodnotu rovnou známému pixelu geometricky nejbližšímu. Neznámé pixely je také možno interpolovat například lineárně. V interpolovaném jednodimenzionálním obrázku je hodnota neznámého pixelu vypočtena váženou sumou pixelů známých. V tomto případě se jedná o váženou sumu dvou nejbližších známých pixelů. Váhy těchto pixelů jsou dané jejich geometrickou vzdálenosti. Rozšířením této metody do dvou dimenzí vznikne metoda *bilineární interpolace*. Tato metoda vyžaduje pro výpočet neznámého pixelu známé pixely čtyři. Při jejím využití dojde nejprve k jednorozměrným lineárním interpolacím nediagonálních disjunktních párů známých pixelů. Souřadnice těchto dvou nově vzniklých pixelů je třeba vybrat tak, aby jejich lineární interpolací bylo možné získat pixel původně hledaný. Lineární interpolací těchto dvou pixelů pro hledaný pixel je získán výsledek interpolace bilineární. Ještě dalším rozšířením je metoda *bikubické interpolace*, kde zmíněným rozšířením je využití křivek. [34]

Druhou kategorií transformací jsou transformace nevyžadující znalost hodnot neznámých pixelů. V takovém případě jsou hodnoty existujících pixelů přepočteny na nové, například při rozmazání, převodu do stupňů šedi, anebo barevných filtrů. Častým způsobem dosažení těchto transformací je operace konvoluce, jako například v obrázku 2.11. Pro detekci hran může být výhodnější využít takzvaného *Sobelova operátoru*. Jeho aplikací podle rovnic 2.19 a 2.20,

$$G_x = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \star I \tag{2.19}$$

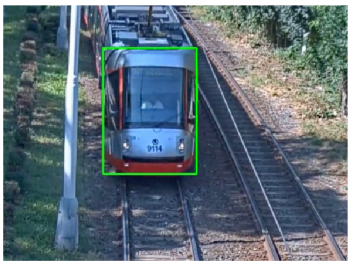
$$G_y = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \star I \tag{2.20}$$

kde $\star$ je operace konvoluce a I je vstupní obrázek, jsou získány obě komponenty gradientu obrázku I. G_x je horizontální komponentou hran a G_y vertikální. Jejich kombinací podle rovnice 2.21

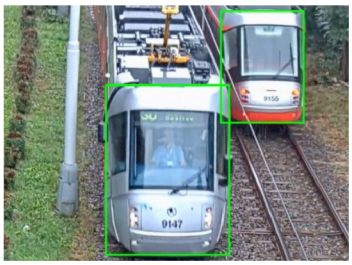
$$\| G \| = \sqrt{[G_x^2 + G_y^2]} \tag{2.21}$$

vznikne matice velikostí gradientů G. Každý pixel G je ohodnocením míry, do které je pixel na stejné pozici ve vstupním obrázku součástí hrany. Hrany detekované tímto způsobem mají často šířku více než jednoho pixelu, což nemusí být žádoucí. Pro řešení tohoto problému je možné využít takzvaný *Cannyho detektor hran*. Některé konvoluční sítě přijímají na svůj vstup obrázek zredukovaný na pouze hrany. [34]

(a) Klasifikace — určení třídy objektu nacházejícím se na vstupním obrázku.



(b) Lokalizace — určení ohraničujícího boxu klasifikovaného objektu.



(c) Detekce objektů — lokalizace jakékoliv množství objektů.

Obrázek 2.16: Ilustrace rozdílů mezi klasifikací, lokalizací a detekcí objektů.

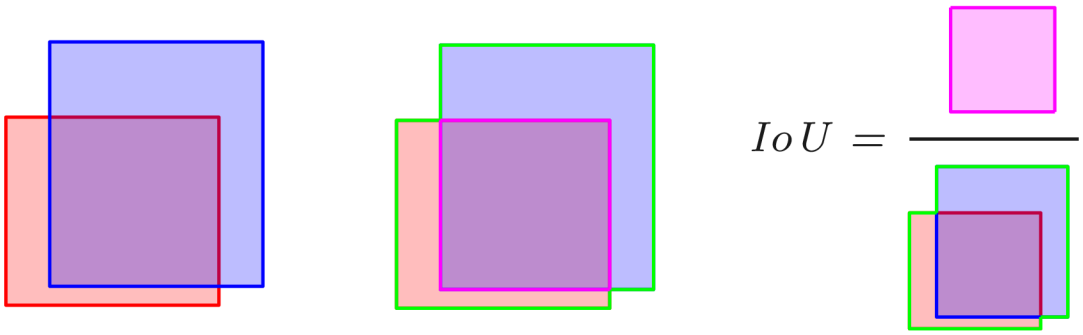
2.6 Detekce objektů

Dosud probírané konvoluční neuronové sítě jsou schopné pouze klasifikace, tzn. určení, jaká z detekovatelných tříd objektů se na vstupním obrázku nachází. To je smysluplnou operací pouze pokud počet objektů patřící do detekovatelných tříd nepřesáhne jedna. Další vykonatelnou operací je lokalizace. Lokalizací se rozumí určení pozice a rozměru klasifikovaného objektu. Dále neuronová síť, schopná určit třídu, pozici i rozměr většího počtu objektů, než jedna, vykonává takzvanou detekci objektů. Ilustrace těchto operací se nachází na obrázku 2.16. Složitost v aplikaci konvolučních neuronových sítí pro detekci objektů se mimo jiné nachází ve faktu, že dimenze výstupní aktivační mapy takové sítě jsou na rozdíl od možného počtu detekovatelných objektů ve vstupním obrázku neměnné. Navzdory tomu jsou různé klasifikátory a nebo jejich podstatná část často nedílnou součástí detektorů objektů. Varianty klasifikátorů, jako například VGG[31], ResNet[11] nebo MobileNet[29], často bývají k tomuto účelu využity. Typickým důvodem pro jejich využití je jejich prokázaná kvalita a efektivita extrakce rysů. [36]

Důležitým konceptem v doméně detekce objektů je *Intersection over Union*, nebo-li *IoU*. IoU mezi dvěmi plochami je dáno vydělením obsahu jejich průsečíku obsahem jejich sjednocení, jak je ilustrováno na obrázku 2.17. Hodnota IoU se vždy nachází v intervalu <0,1>. Častým využitím je výpočet podobnosti mezi předpovězeným boxem a skutečným. Pro vyhodnocení, zdali danou detekci považovat za úspěšnou je třeba zvolit prahovou hodnotu. Pokud je hodnota IoU mezi předpovězeným a skutečným boxem větší než zvolená prahová hodnota, detekci je možné považovat za úspěšnou. [16]

2.6.1 Region-based Convolutional Neural Networks

Společnou vlastností této třídy modelů, často zkracované jako *R-CNN*, je rozdělení procesu detekce na dva kroky — nejprve jsou ve vstupním obrázku nalezeny oblasti s potenciálním objektem a následně jsou s pomocí této informace provedeny samotné detekce. Prvním z nich je model *R-CNN*[9]. Ten nejprve pomocí algoritmu *Selective search* extrahuje ze vstupního obrázku oblasti, ve kterých se potenciálně nachází objekt. Následně individuálně provede lokalizaci na každém z nich konvoluční neuronovou sítí. Selection search je algoritmus umožňující rozdělení všech pixelů ve vstupním obrázku do skupin, kde každá skupina reprezentuje daný objekt. Ackoliv byl tento přístup na svůj čas vysoce úspěšný, trpí řadou nevýhod, jako například vysokou výpočetní cenou algoritmu Selective search, nutnosti zpracovávat každou kandidátní oblast individuálně a nemožností trénovat model

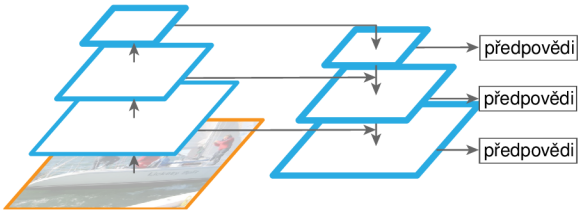


(a) Dva ohraňučující boxy.

(b) Sjednocení dvou boxů (zelená) a jejich průsečík (růžová).

(c) IoU je získáno vydělením obsahu průsečíku obsahem sjednocení.

Obrázek 2.17: Ilustrace získání hodnoty IoU. Inspirováno [16].



Obrázek 2.18: Feature Pyramid Network. Převzato z [21].

jako celek. Další iterací je *Fast R-CNN*[8]. Ta stále využívá algoritmu Selective search, ale nalazené oblasti nejsou konvoluční neuronovou sítí zpracovávány individuálně. Místo toho je konvoluční neuronovou sítí zpracován celý vstupní obrázek. Následně je na výsledné aktivační mapě provedena lokalizace na každé její oblasti korespondující kandidatni oblasti na vstupním obrázku. Odstraněním nutnosti provádět průchod konvoluční neuronovou sítí pro každou kandidátní oblast Fast R-CNN několikrát překoná předchozí R-CNN ve výkonnostní náročnosti. Selective search ale stále představuje velmi vysokou část výpočetního času, což napravuje další model jménem *Faster R-CNN*[27]. Ten již tohoto algoritmu nevyužívá, místo toho je hledání kandidátních oblastí součastí extraktoru rysů. [37]

2.6.2 Singleshot detectors

Tato skupina se vyznačuje pokusem o urychlení sloučením fází zmíněných výše. Významným modelem tohoto typu je *YOLO*. Jak napovídá celé jméno, *You Only Look Once*, vstupní obrázek je použit pouze jednou. Vstupní obrázek je rozdělen do čtvercové mřížky velikosti strany S. Každý detekovaný objekt se považuje za náležicí té buňce, do které náleží jeho střed. Pro každou buňku je předpovězen B počet boxů, ale pro každou buňku je detekovatelný pouze jeden objekt. Pro každý box je mimo souřadnic jeho středu a rozměrů modelem předpovězena také jeho jistota. Ta reprezentuje odhad IoU mezi předpovězeným boxem a skutečným. Kromě boxů je pro každou buňku předpovězeno taktéž C individuálních pravděpodobností, že detekovaný objekt patří do dané třídy. Rozměry výsledné matice tak jsou $S \times S \times (B * 5 + C)$. YOLO oproti R-CNN disponuje jednodušším modelem, nižší časovou

složitostí, a schopností detekce objektů v kontextu celého obrázku. V některých situacích i dosahuje lepších výsledků. [26]

SingleShot Multibox Detector, zkrácene *SSD*, funguje podobně. Součástí návrhu je definice mřížky a množiny výchozích boxů s různými poměry stran a velikostí. Tato množina je stejná pro každou buňku. Při tréningu jsou pak pro každý objekt na vstupním obrázku přiřazené některé z nich. Pro tyto boxy jsou následně vypočteny rozdíly mezi pozicemi a rozměry daných boxů a daného objektu. Ty představují žádoucí výstup sítě, a jsou proto využity pro výpočet ceny. [24]

2.6.3 Feature Pyramid Network

V konvoluční neuronové síti jednotlivé pixely každé další aktivační mapy odpovídají většímu segmentu vstupního obrázku. Ve vrstvách hluboko v jejich síti tak dochází ke ztrátě informací pocházejících z vrstev dřívějších. Jedním ze způsobů, jak tomu předejít, je *Feature Pyramid Network*, zkráceně *FPN*. Postupem bez FPN je zařazení jednotlivých vrstev v sekvenci za sebe, kde poslední vrstva je vrstvou výstupní. Přidaní FPN takové síti je dosaženo nejprve výběrem některých jejích aktivačních map, typicky v různých hloubkách. Ty jsou následně zařazeny (mimo jejich originální účel a pozici) za sebe v opačném pořadí. Každá takto nově vytvořená postranní vrstva je sečtena se součtem všech vrstev předchozích. Každý nově vytvořený součet je pak dále zpracován. Aktivační mapy ovšem mají rozdílný rozměr i počet kanálů, což je před sčítáním potřeba vyřešit. Počet kanálů je konvolucí upraven na společný. Rozměr je upraven převzorkováním. Převzorkování je ekvivalentem vymyšlení rysů ve vstupu se nenacházejících. Ilustrace se nachází na obrázku 2.18. [21]

Příkladem modelu využívající FPN je *RetinaNet*. Každý ze součtů postranních aktivačních map je dále zpracován klasifikátorem a regresorem ohraničujících boxů. Vznikne množina aktivačních map, kde každou hodnotu dané aktivační mapy je možné přiřadit k segmentu na vstupním obrázku. Daný segment však může mít víc než jedno ohodnocení, ve kterém případě je třeba vybrat jedno z nich. Dalším problémem je nerovnováha tříd, tzn. některé z objektů detekovaných tříd se v trénovací sadě nachází v nezanedbatelně rozdílném počtu, než jiné. V tomto případě se jedná o třídu pozadí, která se v typické trénovací sadě nachází ve velmi hojnějším počtu, než objekty tříd jiných. Pro potlačení důsledků této nerovnováhy RetinaNet využívá ztrátové funkce *Focal Loss*, uvedené v rovnici 2.22,

$$FL(p_t) = -(1 - p_t)^{\lambda} log(p_t) \tag{2.22}$$

kde λ je parametrem ovlivňující váhu členu $(1 - p_t)$. Focal Loss vznikne přidáním členu $(1 - p_t)^{\lambda}$ do cross-entropy. Focal Loss má oproti cross-entropy strmější průběh a snižuje ztrátu pro velmi jisté předpovědi. Triviální přepovědi tak mají menší vliv na optimalizaci parametrů. [22]

Dalším příkladem je model *Fully Convolutional One-Stage Object Detector*, nebo-li *FCOS*. Ten přináší oproti RetinaNet řadu změn, jako například dodatečné vrstvy nebo nový „centerness" výstup. Tento výstup udává předpovězenou vzdálenost středu předpovězeného ohraničujícího boxu od středu skutečného objektu. Jeho účelem je získání přenějšího odhadu o kvalitě dané detekce. Dalším důležitým rozdílem je nevyužití výchozích boxů (tzv. „anchors"). [33]

Kapitola 3

Návrh

Cíl této práce stanovuji na program se schopností co nejpřesnějšího určení polohy a velikosti všech tramvají ve vstupním videu. Tato informace bude udaná v pixelech, relativně k levému hornímu rohu každého snímku. Program bude otevřen snadnému pozdějšímu přidání schopnosti uvést vzdálenost a rozměr také v metrech relativně ke kameře za předpokladu, že uživatel uvede dostatečné množství kalibračních detailů. K tomuto účelu volím detekovatelnou část tramvaje na pouze její přední a zadní stranu, protože tyto strany mají značně nižší variabilitu rozměrů než ostatní. Jednotlivé modely tramvají mají variabilitu výšky o dost vetší než variabilitu šířky. Tramvaje se často pohybují vysokou rychlostí, a to způsobuje na výstupu statické kamery jejich rozmazání. Znatelnost tohoto efektu je úměrná blízkosti tramvaje ke kameře a její snímkovací rychlosti. Různé modely tramvají mají také velmi vysokou variaci rysů jejich předních stran. Taktéž jejich barevné provedení může být jakékoliv, a proto budu převádět všechna barevná data do odstínů šedi.

3.1 Cílový program

Využito bude následujících knihoven a technologií:

- CUDA — technologie umožňující hardwarovou akceleraci některých typů výpočtů vybranými grafickými kartami společnosti NVIDIA. Velikost této akcelerace může být i v tisících procent.
- ROCm — ekvivalent CUDA pro některé grafické karty společnosti AMD.
- PyTorch — knihovna určená pro tvorbu, trénování a využití neuronových sítí.
- OpenCV — mimo jiné obsahuje funkce pro manipulaci s obrazovými daty.

Cílovou platformou bude Linux. Program bude podporovat jak CUDA, tak ROCm za předpokladu, že uživatel má systém správně sestavený a nastavený pro využití zvolené technologie. Rozhraní bude pouze příkazovou řádkou. Uživatel bude mít možnost:

- Zvolit, zdali chce pro výpočty využít grafické karty nebo procesoru.
- Dotrénovat zvolený model na vlastních datech a v takovém případě zvolit i parametry trénování jako například velikost dávky, rychlost učení a podobně.
- Zvolit model.

- Použít video nebo webkameru jako vstup a následně jako výstup zvolit kombinaci z:
  - vykreslení vstupního videa obohacené o detekované tramvaje vyznačené obdélníky na obrazovku
  - výstupu zmíněného videa do souboru
  - výstupu čtveřic souřadnic detekovaných tramvají na standardní výstup

Jedinou částí datové sady držené v paměti bude její část bezprostředně potřebná k výpočtu. Minimální podporované množství paměti RAM bude 16GiB. Program se bude skládat z hlavního skriptu řídícího celý proces a vyměnitelných modulů. Před počátkem výpočtu hlavní skript vybere z těchto modulů moduly potřebné pro uživatelem zvolený výpočet. Pro tento skript pak díky kombinaci polymorfismu a kompozice následně skoro nezávisí na jejich konkrétním původu a funkcionalitě.

3.1.1 Detekce využitím předtrénovaných modelů

Pro zjednodušení rozpoznávání obrazu existují různé předtrénované modely. Ty je třeba dotrénovat na specifickém problému. Ačkoliv jejich využití umožňuje dosáhnout velmi dobrých výsledků velmi rychle, není jejich využití plnou náhradou za účelně navržené a od začátku vytrénované modely. PyTorch nabízí řadu předtrénovaných, přednastavených modelů. Těmi jsou:

- SSD
- SSDlite
- RetinaNet
- FCOS
- Faster R-CNN

Tyto knihovní modely nejsou přesnou kopií originálních modelů nesoucí jejich příslušné jméno, jsou na nich pouze založené. Předtrénování proběhlo na datové sadě COCO [23]. Tato datová sada obsahuje více než 80 kategorií. Ačkoliv žádná z nich není tramvaj, nachází se v ní kategorie příbuzné, jako například vlak, autobus a osobní nebo nákladní automobil. Protože detekují více kategorií, než je pro řešení tohoto problému nutné, typickým postupem by bylo nahradit několik posledních vrstev vlastním návrhem lépe se hodící k danému účelu, v tomto případě detekující kategorii pouze jednu. To ale do řešení vnáší zbytečný prostor pro chybu. Rozhodl jsem se raději knihovní návrh neměnit a raději zvolit jednu kategorii, jež bude „přeučena” na detekci tramvaje. Při testování pak budou ostatní kategorie ignorovány. Vrstvy, jež by byly nahrazeny, mají oproti extraktoru rysů nacházejícím se před nimi zanedbatelnou výpočetní cenu.

Indikátorem přetrénovanosti bude počet validačních snímků s nulovým počtem detekcí. Po každé epoše bude model otestován na validační sadě a pokud dosáhne zmíněný počet příliš velké části validační sady, trénování bude zastaveno. Postup zmenšování celkové ceny modelu bude taktéž sledován pro účely pozastavení s manuální úpravou rychlosti učení. Tato rychlost začne pro každou síť na 0.0001. Pro optimalizaci parametrů bude využito stochastického gradientního sestupu s modifikací ADAM.

3.1.2 Detekce na míru navrženou sítí

Síť, jež se skládá pouze z konvolučních a max pooling vrstev, není omezena přesnou délkou vstupu, je pouze omezena nejmenší délkou. Pro binární klasifikační problém, jaký se snažím řešit zde, takový nejmenší vstup vyprodukuje po průchodu sítí právě jeden výstup. Větší vstupy vyprodukují více než jeden. Vzhledem k povaze max pooling vrstev, pro síť tyto vrstvy obsahující tyto jednotlivé výstupy korespondují různým, možně překrývajícím se částem vstupního obrázku. Správnou interpretací takové sítě lze její výstup chápat jako teplotní mapu jednotlivých pravděpodobností, že se na příslušných částech vstupního obrázku nachází tramvaj. Síť je možné se snažit trénovat pro rozpoznání tramvají všech velikostí, to ale v praxi nemá moc velký úspěch. Úspěšnějším postupem je naučení sítě jedné velikosti a při testování vstupní snímek síti předložit v různých škálách, čímž vznikne množina teplotních map, každá z nich popisující šanci že se při dané škále vyskytuje ve vstupu tramvaj. Pro stejnou sekci obrázku tudíž vznikne více ohodnocení. Pro danou sekci je vybrán nejjistější střed a rozměr je odhadnut podle jeho původu.

Pokud se v síti plně propojené vrstvy nachází, vstup je omezen přesným rozměrem. Tradičním umístěním těchto vrstev je konec sítě. Takovou síť se skládá dvou části: extraktor rysů představující většinu parametrů sítě a skládající se pouze z konvolučních a max pooling vrstev následován klasifikátorem skládající se z pouze vrstev plně propojených. Těchto klasifikátorů může mít síť víc než jeden. Jejich význam je dán jejich učením. Například jeden z nich může mít jeden výstup značící přítomnost tramvaje, zatímco druhý může mít výstupy čtyři značící její souřadnice. Klasifikátor se může skládat i z konvolučních vrstev a získat tím identickou schopnost přijímat rozsah rozměrů vstupu. Přirozeně nabízející se architekturou mojí na míru navržené sítě je tudíž jeden extraktor rysů následován dvěma klasifikátory, oba skládající se pouze z konvolučních vrstev — regresor ohraničujících boxů a třídní klasifikátor. Inspirováno architekturou zmíněných předtrénovaných sítí.

Z důvodu nedostatečné velikosti vlastní datové sady a podobnosti tramvají s dalšími dopravními prostředky bude využito i datové sady COCO, a to specificky objektů označených jako `train`, `car`, `bus` a `truck`. Protože tato datová sada obsahuje i množství tramvají pod označením `train`, nemá smysl při jejím použití implementovat rozlišení tramvají od vlaků. Třídní klasifikátor bude mít pro nejmenší vstup čtyři výstupy, každý vyjadřující jistotu že se na vstupu nachází korespondující dopravní prostředek a podobně pro regresor ohraničujících boxů. Síť bude vytrénovaná nejprve na COCO a poté dotrénovaná na vlastní datové sadě obsahující pouze tramvaje. Důvodem je fakt že v COCO jsou dopravní prostředky označené celé, a ne pouze jejich přední strana. Protože vytvoření takové sítě je značně náročné, bude provedeno iterativně, ve fázích se vzestupnou složitostí:

- síť s nepřekrývajícími se segmenty vstupu a bez regresoru ohraničujících boxů
- síť s překrývajícími se segmenty vstupu ale bez regresoru ohraničujících boxů
- kompletní síť
- kompletní síť optimalizovaná zmenšením počtu parametrů a využitím depthwise separation

3.2 Data

Vlastní datová sada sestává z množiny videí. Malou část této množiny jsem natočil sám na telefon Sony Xperia XZ1 s kamerou s rozlišením 19 megapixelů a fokální délkou 25 milimetrů

Obrázek 3.1: Příklady snímků z datové sady.

na Mendlově náměstí v Brně. Zbytek mi byl poskytnut vedoucím. Data je před použitím třeba zpracovat. V datech je třeba vyznačit oblast, kde se nachází tramvaj. Provádět tohle pro video by bylo nepraktické, a tak prvním krokem pro trénování neuronové sítě je vybrání specifických snímků. Na těchto snímcích je následně provedeno tohle vyznačení. K tomuto účelu existuje řada programů, z nichž mojí volbou je *LabelMe*[1]. LabelMe poskytuje grafické rozhraní pro vyznačení objektů na obrázku. Pro moje účely zajímavou funkcí je vyznačení obdélníkem, kde tyto obdélníky jsou určeny koordináty dvou bodů. Obrázek je následně uložen i se všemi obdélníky do souboru JSON. Vyznačenou částí tramvaje je její přední nebo zadní strana. Ohraničující boxy mají svůj poměr stran skoro vždy blízký $2:3$ a pouze zanedbatelná část pixelů nacházející se uvnitř boxu nepatří tramvaji. Snímky jsou vlivem jejich původu zkomprimované s vysokou ztrátou, nicméně rozměry žádného snímku nejsou méně než 1280×720. Všechny mají stejný poměr stran a to $16:9$. Tato datová sada sestává z celkem 339 snímků, na kterých se nachází celkem 363 anotovaných tramvají. Částečně viditelné tramvaje anotovány nebyly. Některé skupiny těchto snímků jsou si velmi podobné, protože pocházejí ze stejného videa. Příklady se nachází na obrázku 3.1. Sada je rozdělena na trénovací, validační a testovací generátorem pseudo-náhodných čísel se šancemi 0.7, 0.1 a 0.2 příslušně. Počty snímků těchto sad jsou 247, 29 a 63.

Dalším využitým zdrojem dat je datová sada COCO. V sadě nabídnuté ke stažení na stránkách COCO[2] se nachází celkem 163957 obrázků, rozděleno na sady trénovací (118287), validační (5000) a testovací (40670). Pro mě užitečnými vzorky jsou pouze takové obsahující alespoň jeden objekt jedné z kategorií `train`, `car`, `bus` a `truck`. Po odstranění obrázků žádný z takových objektů neobsahující zbyde v trénovací sadě 19335 vzorků, zatímco ve validační vzorků 836. Testovací sada nebude využita. Vyznačenou částí objektů jsou objekty celé, pokud jsou celé viditelné. Pokud nejsou, většinou je označena pouze jejich část viditelná

[1] http://labelme.csail.mit.edu/Release3.0/
[2] https://cocodataset.org/#download

Obrázek 3.2: Ohraničující box vlaku na obrázku pocházejícím z datové sady COCO [23]. Poměrně velká část pixelů nacházejících se uvnitř boxu nepatří vlaku.

v popředí, nicméně není výjimkou jejich označení podobným způsobem jako kdyby se na obrázku překážka zabraňující jejich kompletní viditelnosti nenacházela. Dopravní prostředky a obzvláště tramvaje a vlaky mohou dosahovat relativně vysokých délek, a tak pod některými úhly kamery ohraničující box takového objektu bude obsahovat velké množství pixelů patřící pozadí. Příkladem je obrázek 3.2. Obrázky jsou různých rozlišení a poměrů stran. Rozlišení se většinou pohybuje kolem 500×500 a komprese je vysoce ztrátová. Všechny objekty nacházející se v datové sadě jsou umístěné v jednotném seznamu v jednom souboru JSON s pouhou referencí na zdrojový obrázek. Pro jednotnost byl tento soubor převeden na stejný formát, jaký vznikl anotací vlastních dat v programu LabelMe.

Kapitola 4

Řešení

Předmětem této kapitoly je implementace a její testování. Vzniklý program se skládá z části společné pro všechny modely a z části implementované pro každý model inidividuálně. Vzniklé modely jsou po jejich vytrénování porovnány.

4.1 Implementace

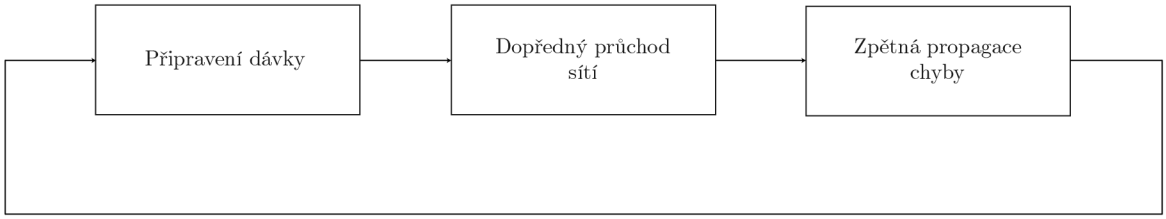
Některé části zdrojového kódu jsou založené na příkladech uvedených v dokumentacích PyTorch[1] a OpenCV[2]. Prvním krokem po přijmutí a zpracování argumentů (což je snadno vyřešeno knihovnou `argparse`) je načtení snímků, ať už pro účely trénování, anebo testování. Pro tohle ideální konstrukcí jazyka Python je generátor. Inicializovaný generátor se chová podobně jako list, s rozdílem že je možné jej projít pouze jednou a pouze sekvenčně. Jeho výhodou oproti standardnímu listu je schopnost nedržet všechna procházená data současně v paměti. Generátor je definován funkcí, která místo klíčového slova `return` používá klíčové slovo `yield`. Každý objekt vrácen pomocí `yield` se pak považuje za jeden prvek generovaného listu. Při požádání inicializovaného generátoru o další prvek v sekvenci je předáno řízení této funkci až po moment, kdy narazí na `yield`, kde je funkce pozastavena a řízení vráceno zpět. Postup se opakuje pro každý prvek v sekvenci až po konec funkce. Generátor je následně vyčerpaný a už nikdy žádný další objekt nevrátí. Tato konstrukce je využita pro načítání snímků ze souborů, načítání snímků z videa a rozdělování dat do dávek. Načítající generátory vrací kromě snímku také slovník obsahující jeho metadata a ohraničující boxy, pokud to dává pro daný snímek smysl.

Pro abstrakci detailů jednotlivých modelů se ve zdrojovém kódu nachází rozhraní pod jménem `ModelUtil`. Nejedná se sice o rozhraní ve striktním slova smyslu, avšak použitím se o rozhraní jedná. Třídy implementující tohle rozhraní musí povinně implementovat:

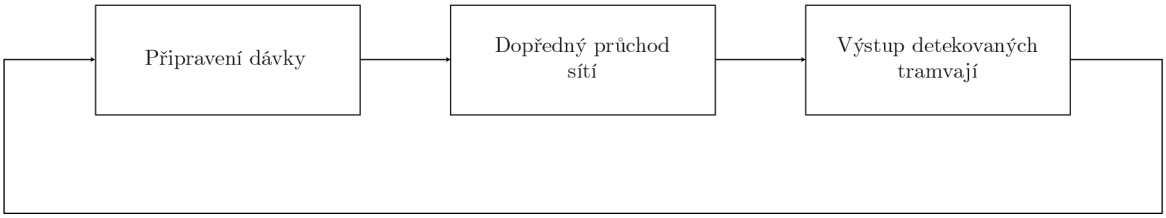
- `get_data` — převod snímků načtených pomocí zmíněných generátorů na data použitelná k trénování
- `get_model` — získání zinicializovaného modelu
- `batch2boxes` — otestování modelu dávkou dat

Nepovinně pak mohou implementovat pomocnou funkci `preprocess`, která slouží třídě implementující `ModelUtil` interně. Díky této abstrakci může program s jednou výjimkou zacházet při trénování a testování se všemi modely identickým způsobem. Výjimkou je převod

[1] https://pytorch.org/docs/stable/index.html
[2] https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html



(a) Blokové schéma hlavní trénovací smyčky.



(b) Blokové schéma hlavní testovací smyčky.

Obrázek 4.1: Bloková schémata trénovací a testovací smyčky.

vygenerovaných numpy matic na matice PyTorch. Důvodem je usnadnění manipulace s třídami `ModelUtil` ve více procesech. Knihovna PyTorch je velmi náročná na použití ve více procesech, a tak je použití této knihovny zanecháno jen hlavnímu procesu. Protože se převáděné datové struktury mezi některými modely liší, dosažení plné abstrakce není možné. Ačkoliv paralelní funkcionalita nebyla implementována, program byl navržen s možností jejího pozdějšího přidání.

Soubor `run.py` obsahuje mimo jiné hlavní trénovací smyčku. Její blokové schéma se nachází na obrázku 4.1a. Tato smyčka pro každou vygenerovanou dávku vypočítá její celkovou cenu a následně provede jeden optimalizační krok. `model` se může nacházet v módu trénovacím nebo testovacím. V trénovacím přijímá jak vstup, tak správný výstup a vrací cenu. Ztrátová funkce je součástí modelu. V testovacím módu přijímá pouze vstup a vrací výstup. Testovací smyčka vypadá podobně — rozdílem je záměna výpočtu a zpětné propagace chyby za výstup souřadnic předpovězených ohraničujících boxů. Její blokové schéma se nachází na obrázku 4.1b.

Jediným zbývajícím problémem je implementace `ModelUtil` pro každý model. Předtrénované modely navzdory mým předpokladům v době návrhu mají tyto implementace velmi podobné, a tak jsou sdílené. Jejich trénování je relativně přímočaré, kde velmi rychle začnou dosahovat dobrých výsledků. Pro model RetinaNet jsou nabízeny verze `v1` a `v2`, z nichž jsem vybral `v2`. Podobně Faster R-CNN nabízí verze `resnet50_fpn`, `resnet50_fpn_v2`, `mobilenet_v3_large_fpn` a `mobilenet_v3_large_320_fpn`, ze kterých jsem se rozhodl vyzkoušet `resnet50_fpn_v2` a `mobilenet_v3_large_320_fpn`. Se vstupem před předložením modelu provádím transformace převodu do jednoho černobílého kanálu a změny rozměrů.

Pro vlastní síť je třeba také navrhnout její architekturu. První a druhá iterace se skládá z jednoho extraktoru rysů a jednoho klasifikátoru. Hyperparametry, jež byly předmětem experimentů, jsou následující:

- počet a kombinace jednotlivých vrstev
- rozměry a krok jednotlivých vrstev
- pro každou konvoluční vrstvu počet jader

Index	Typ vrstvy	Rozměr	Krok	Počet jader
1	Konvoluční	$3 \times 3 \times 1$	$1 \times 1 \times 1$	6
2	Konvoluční	$3 \times 3 \times 6$	$1 \times 1 \times 1$	16
3	Max pooling	$2 \times 2 \times 1$	$2 \times 2 \times 1$	
4	Konvoluční	$3 \times 3 \times 16$	$1 \times 1 \times 1$	32
5	Konvoluční	$3 \times 3 \times 32$	$1 \times 1 \times 1$	64
6	Max pooling	$2 \times 2 \times 1$	$2 \times 2 \times 1$	
7	Konvoluční	$3 \times 3 \times 64$	$1 \times 1 \times 1$	128
8	Konvoluční	$3 \times 3 \times 128$	$1 \times 1 \times 1$	256
9	Max pooling	$2 \times 2 \times 1$	$2 \times 2 \times 1$	
10	Konvoluční	$3 \times 3 \times 256$	$1 \times 1 \times 1$	256
11	Konvoluční	$3 \times 3 \times 256$	$1 \times 1 \times 1$	256
12	Max pooling	$2 \times 2 \times 1$	$2 \times 2 \times 1$	
13	Konvoluční	$3 \times 3 \times 256$	$1 \times 1 \times 1$	256
14	Max pooling	$3 \times 3 \times 1$	$1 \times 1 \times 1$	
15	Konvoluční	$1 \times 1 \times 256$	$1 \times 1 \times 1$	128
16	Konvoluční	$1 \times 1 \times 128$	$1 \times 1 \times 1$	64
17	Konvoluční	$1 \times 1 \times 64$	$1 \times 1 \times 1$	4

Tabulka 4.1: Sekvence vrstev architektury TramNet.

- velikost hodnoty p pro dropout každé vrstvy
- aktivační a cenové funkce

Optimalizace hyperparametrů je otevřená problematika nemající žádný zaručeně fungující postup. Testovaní dané konfigurace je vysoce výpočetně náročná operace, kde jeden test vyžaduje jednotky dnů a kilowatthodin energie. Nejoptimálnější nalezenou konfigurací hyperparametrů je architektura TramNet. Sekvence jednotlivých vrstev této architektury se nachází v tabulce 4.1. Aktivační funkcí každé konvoluční vrstvy je $ReLU()$. Zvolenou cenovou funkcí je kvadratická cena. Tato architektura nemá regresor ohraničujících boxů. Architekturu s tímto regresorem jsem se rozhodl neimplementovat. Všechny vrstvy s rozdílnou velikostí rozměru a kroku s výjimkou posledních třech mají ke svému vstupu přidaný nulový okraj velikosti jedna.

Všechny max pooling vrstvy kromě poslední mají rozměr i krok velikosti 2×2. Těchto vrstev je celkem 4, tudíž každý pixel aktivační mapy 4. max pooling vrstvy koresponduje jednomu nepřekrytému segmentu 16×16 na vstupním obrázku. Poslední max pooling vrstva je rozměru 3×3 a kroku 1×1. Protože má velikost kroku menší než velikost rozměru, každý pixel výsledné aktivační mapy koresponduje překrývajícím se segmentům ve vstupním obrázku. Rozměr těchto segmentů je 72×72 a krok 16×16. Poslední 3 konvoluční vrstvy jsou pro vstup velikosti 16×16 ekvivalentem sekvence 3 plně propojených vrstev. V této architektuře slouží jako klasifikátor.

4.2 Testování

Klasické zhodnocení ve formě počtu správných a nesprávných odpovědí zde není relevantní, protože každý vstupní snímek má pro řešení uvedeného problému množství správných řešení. Daná detekce může být posunutá nebo zvětšená i o jednotky pixelů bez ztráty zdánlivé

Model	AP	AP_{50}	AP_{75}	AR	PDZ
SSD	0.75	1.0	0.85	0.79	25ms
SSDlite	0.8	1.0	0.91	0.83	17ms
RetinaNet	0.86	1.0	0.97	0.89	56ms
FCOS	0.88	1.0	1.0	0.89	55ms
Faster R-CNN ResNet	0.9	1.0	1.0	0.92	91ms
Faster R-CNN MobileNet	0.23	0.65	0.15	0.37	5ms
TramNet	0	0	0	0	6ms

Tabulka 4.2: Výsledky testů. PDZ je zkráceně Průměrná Doba Zpracování jednoho snímku. Celkový počet detekovatelných tramvají v testovací sadě byl 68.

přesnosti. Porovnání manuálního vyznačení tramvaje s předpovězeným přesnou rovností tak nepodá dostatečně přesnou kvantifikaci užitečnosti daného modelu. Často využitým řešením je ohodnocení podle standardu COCO. Tahle metoda začne rozdělením všech detekcí dané třídy na true a false pozitiva, podle toho zdali jejich IoU dosahuje předem zvolené prahové hodnoty anebo ne. Další důležitou hodnotou je počet false negativů, což je počet objektů dané třídy s nulovým počtem detekcí. *Average Precision*, často zkracováno jako *AP*, vyjadřuje proporci správných detekcí a je vypočteno pro danou třídu podle rovnice 4.1. Podobně *Average Recall*, často zkracováno jako *AR*, vyjadřuje proporci detekovaných objektů a je vypočteno pro danou třídu podle rovnice 4.2.

$$\mathrm{AP} = \frac{\text{true pozitiva}}{\text{true pozitiva} + \text{false pozitiva}} \tag{4.1}$$

$$\mathrm{AR} = \frac{\text{true pozitiva}}{\text{true pozitiva} + \text{false negativa}} \tag{4.2}$$

Zvolenou prahovou hodnotu je zvykem zapsat do dolního indexu označení získaného výsledku. Například AP s prahovou hodnotou 0.5 by bylo zapsáno jako $\mathrm{AP}_{0.5}$. AP bez dolního indexu označuje v kontextu standardu COCO průměr všech AP mezi $\mathrm{AP}_{0.5}$ a AP_1 včetně, jejichž prahová hodnota je celočíselným násobkem 0.05. [23]

Další měřenou metrikou byla výpočetní náročnost, udána jako průměrný počet milisekund potřebný ke zpracování jednoho vzorku referenčnímu systému[3]. Do výpočetní náročnosti jsem započítal pouze čas strávený dopředným průchodem neuronovou sítí. Toho bylo dosaženo změřením času před a po průchodu. Všechny rozdíly těchto dvou časových údajů byly následně sečteny a poděleny počtem vstupů. Maximální velikost dávky byla 5 pro všechny modely. Pro některé modely by pro jejich nízkou paměťovou cenu fungovala velikost dávky i mnohonásobně větší. Ačkoliv využití této možnosti by se pozitivně odrazilo na jejich výpočetní náročnosti, pro zachování objektivity porovnání jsem se rozhodl možnosti nevyužít. Fakt, že poslední dávka má jinou velikost, než všechny ostatní není pro jeho zanedbatelný efekt na průměrnou dobu zpracování nijak ošetřen. Pro výpočet metrik podle standardu COCO jsem využil nástroje Object Detection Metrics[4]. Výsledky testů se nachází v tabulce 4.2. Správnost modelu sice koreluje s výpočetní náročností, kde nejlepší ale zároveň nejdražší je model Faster R-CNN ResNet, nicméně poměr správnosti a výpočetní náročnosti se zvedající se výpočetní náročností klesá.

[3]Systémem, na kterém byly modely testované je *Lenovo Legion 5 15ACH6A*. (CPU AMD Ryzen 5 5600H, GPU AMD ATI Radeon RX 6600M, VRAM 8 GiB, RAM 16 GiB).

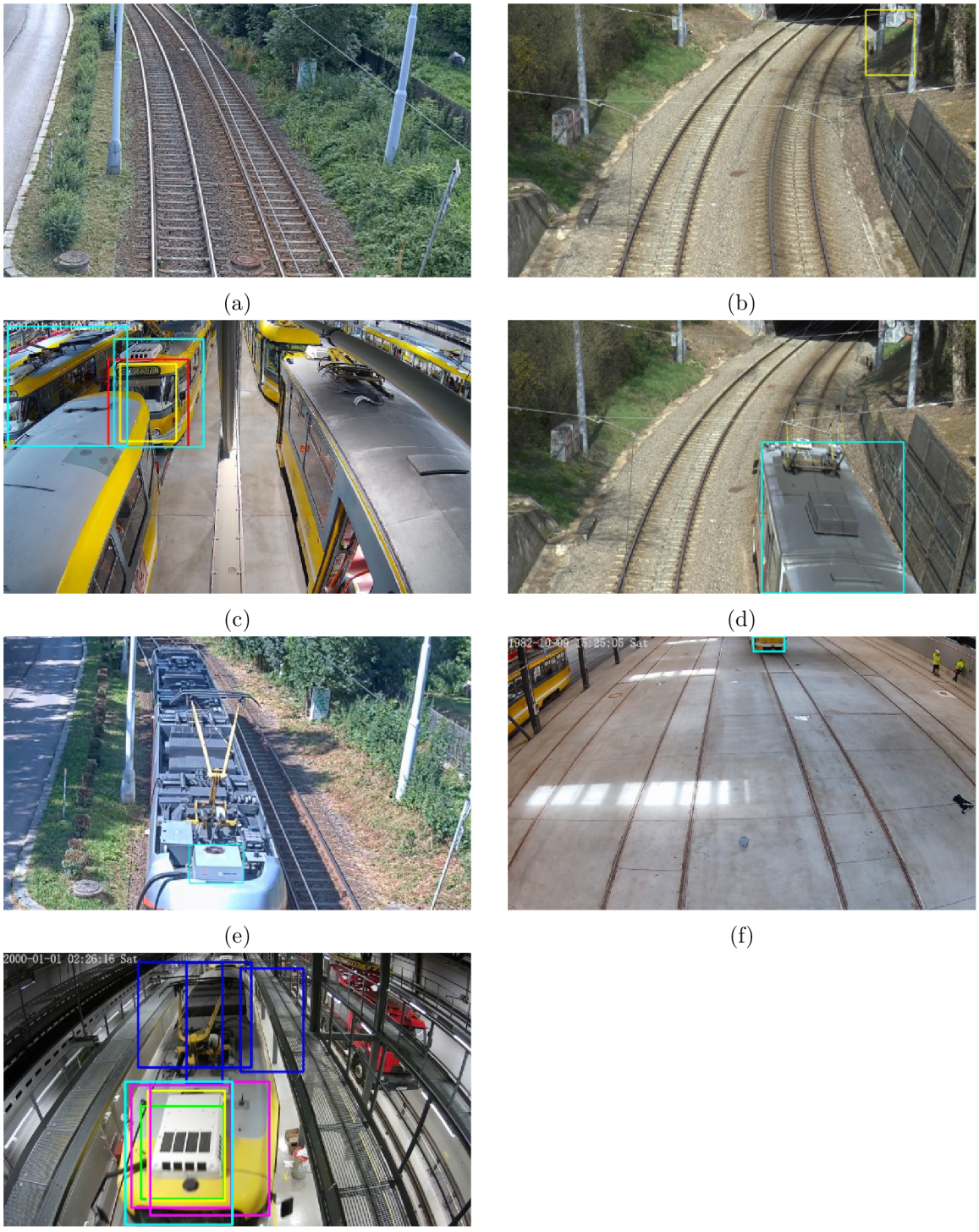
[4]https://github.com/rafaelpadilla/review_object_detection_metrics

Model	Barva
SSDlite	Růžová
SSD	Žlutá
FCOS	Zelená
RetinaNet	Tyrkysová
Faster R-CNN MobileNet	Modrá
Faster R-CNN ResNet	Červená
TramNet	Bílá

Tabulka 4.3: Barevné kódování detekcí jednotlivých sítí.

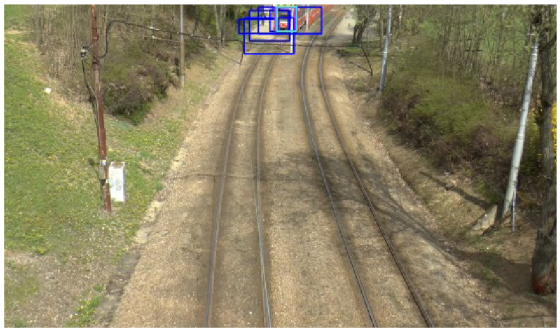
Po dokončení testování jsem znovu prošel dostupné videozáznamy a vybral 17 nových, dosud nepoužitých, snímků. Následně jsem na každý snímek vykreslil detekce všech individuálních sítí. Barevné kódovaní se nachází v tabulce 4.3. Výsledné snímky se nachází na obrázcích 4.2, 4.3 a 4.4. Účelem snímků na obrázku 4.2 bylo otestovat chovaní modelů pro vstupy s nulovým počtem detekovatelných tramvají. S malým počtem výjimek se chovají dle očekávání. Některé modely detekovali i tramvaje z velké části překryté, jako například na obrázcích 4.2c nebo 4.2f. Častým false pozitivem je označení střechy nebo strany tramvaje jako například na obrázcích 4.2c, 4.2d, 4.2e, anebo 4.2g. Taktéž na obrázku 4.4 je vidět velké množství chybně detekovaných bočních stran tramvají. Jak je ale vidět na ostatních příkladech, strany jsou detekovány pouze při dostatečné kolmosti strany ke směru kamery. Pravděpodobnou příčinou je vizuální podobnost všech stran tramvaje obsahujících okna. Model RetinaNet je na false pozitiva tohoto typu nejvíce náchylný. Podobně byly detekovány i dodávky na obrázcích 4.4a a 4.4c, ačkoliv ve druhém případě se jedná o pouze jednu detekci pocházející od Faster R-CNN ResNet.

Pro vysoce hranaté tramvaje má většina modelů vysokou úspěšnost. Detekce jednotlivých modelů jsou si pro tento případ často vysoce podobné, jak je vidět například na snímcích 4.3a, 4.3b, 4.3c a 4.3d. Pro tramvaje obsahující vysoké množství křivek již ale výsledky tak konzistentní nejsou, jako například pro některé z tramvají na snímcích 4.3d, 4.3e a 4.3f. Model SSD má pro některé hůře detekovatelné tramvaje horší výsledky než model SSDlite, jako například snímky 4.3d a 4.4d, avšak SSDlite má větší počet false pozitivů.

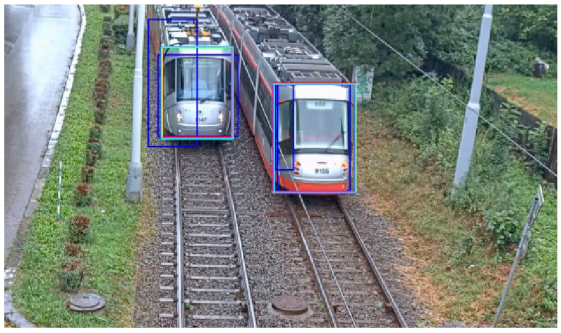


(a) (b) (c) (d) (e) (f) (g)

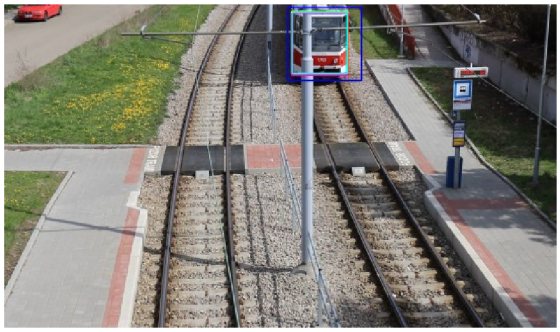
Obrázek 4.2: Všechny detekce všech vytrénovaných sítí na vybraných snímcích obsahujíc nulový počet zcela nepřekrytých tramvají.



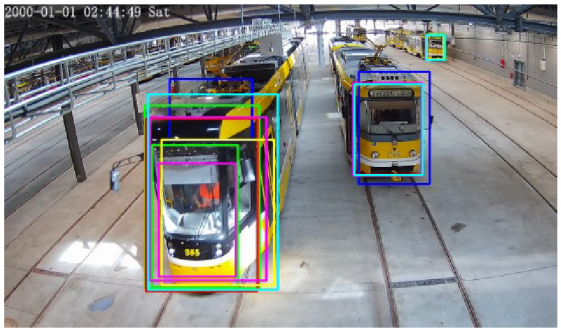
(a)



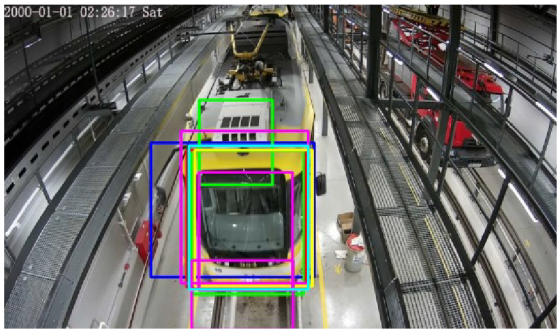
(b)



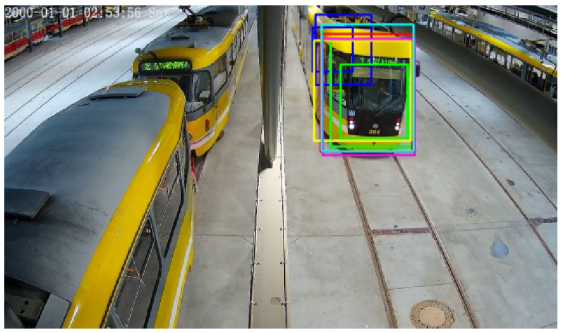
(c) Tramvaj byla detekována navzdory lehké překážce.



(d) Tramvaj nejbližší kameře má vysokou variaci dektekcí. SSDlite (růžová) má správnou detekci kvalitnější než SSD (žlutá).

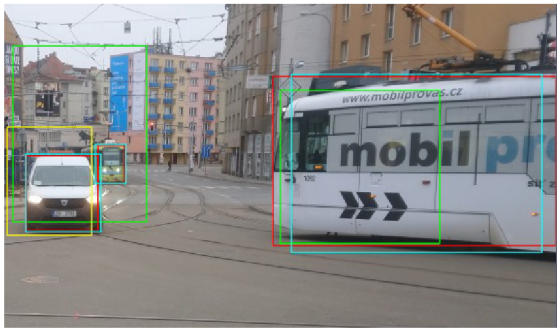


(e)

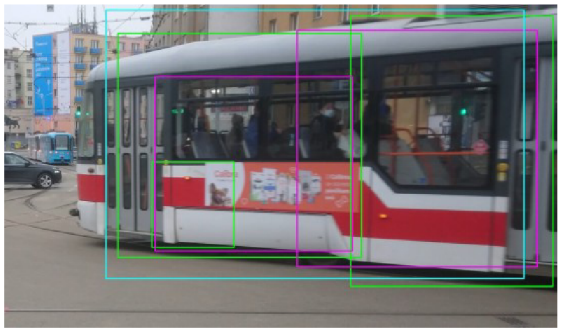


(f)

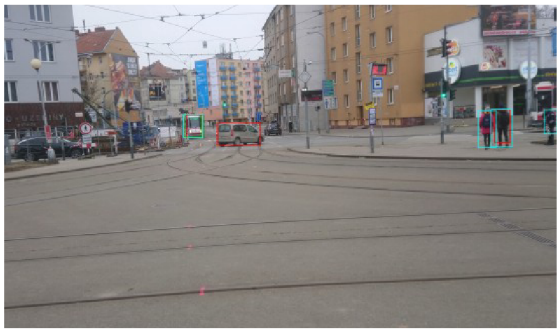
Obrázek 4.3: Všechny detekce všech vytrénovaných sítí na vybraných snímcích obsahujíc tramvaje hůře detekovatelné, anebo tramvají zvýšený počet. Tramvaje na snímcích 4.3a, 4.3b, 4.3c a s výjimkou tramvaje nejbližší kameře 4.3d byly perfektně detekovány všemi předtrénovanými sítěmi kromě Faster R-CNN MobileNet.



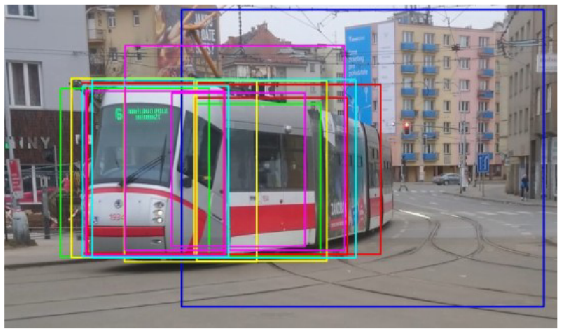
(a) Detekce na snímku obsahujícím tramvaj ze strany, tramvaj velmi vzdálenou a dodávku. FCOS (zelená) nejenže chybně detekoval dodávku jako tramvaj, ale také si ji spojil se vzdálenou tramvají do jednoho objektu.



(b) Detekce na snímku obsahujícím tramvaj ze strany, tramvaj velmi vzdálenou a osobní automobil.



(c) Detekce na snímku obsahujícím velmi vzdálenou tramvaj a dodávku. Vzdálená tramvaj byla detekována sítěmi FCOS, RetinaNet a Faster R-CNN ResNet.



(d) Detekce na snímku obsahujícím tramvaj v zatáčce. Strana tramvaje má alespoň jednu nesprávnou detekci od každého předtrénovaného modelu. Správná detekce modelu SSDlite (růžová) je lepší než správná detekce SSD (žlutá).

Obrázek 4.4: Všechny detekce všech vytrénovaných sítí na vybraných snímcích pořízených ze stejného místa pod podobným úhlem. Všechny kromě 4.4c jsou pro čitelnost ořízlé.

Kapitola 5

Závěr

Cílem bakalářské práce bylo popsat funkci konvolučních neuronových sítí a vytvořit systém schopný detekce pozice a rozměru tramvaje ohraničujícím boxem. Tento cíl byl splněn. Výstupem je aplikace schopná podat informaci o všech tramvajích na vstupu. Došlo k vytrénovaní řady modelů. Tyto modely byly následně porovnány. Uživatel má na výběr ze všech vytrénovaných modelů. Některé tramvaje jsou snadněji detekovatelné než jiné. Ačkoliv cílem bylo detekovat pouze přední a zadní stranu tramvaje, mezi detekcemi se často nachází také část strany boční nebo střechy. Nejkvalitnějším je model Faster R-CNN ResNet, ovšem ten je výpočetně velmi drahý. Modely SSD a SSDlite jsou si kvalitou i výpočetní cenou velmi podobné. Model SSDlite podá v některých výjimečných případech přesnější detekci, zatímco model SSD má menší počet false pozitivů. Stejným způsobem podobné jsou si modely FCOS a RetinaNet, kde RetinaNet je ze všech předtrénovaných modelů nejnáchylnější na detekci nesprávné strany tramvaje, zatímco FCOS často detekuje danou tramvaj více než jednou. Oba jsou ale kvalitnější než obě varianty SSD. Model Faster R-CNN MobileNet má s přehledem největší počet false pozitivů a nepřesných detekcí.

Práce může být dále rozšířena vylepšením datové sady, jako například zvětšením počtu tramvají nebo anotace existujících dat tak, aby ohraničující boxy obsahovaly co nejmenší počet pixelů nepatřící dopravnímu prostředku. Na míru vytvořená architektura může být vylepšena optimalizací hyperparametrů například zněkolikanásobením počtu parametrů a prozkoumáním alternativních aktivačních a cenových funkcí. Zcela neprozkoumaným prostorem je využití segmentace místo ohraničujících boxů a filtrování vzniklých výsledků. Vzniklý program je otevřen možnosti přidání detekce skutečných rozměrů a pozice a optimalizaci zvýšením počtu využitých jader procesoru.

Literatura

[1] Albawi, S., Mohammed, T. A. a Al Zawi, S. Understanding of a convolutional neural network. In: *2017 International Conference on Engineering and Technology (ICET)*. 2017, s. 1–6. DOI: 10.1109/ICEngTechnol.2017.8308186.

[2] Amari, S.-i. Backpropagation and stochastic gradient descent method. *Neurocomputing*. Elsevier. 1993, sv. 5, 4-5, s. 185–196.

[3] Arora, R., Basu, A., Mianjy, P. a Mukherjee, A. Understanding deep neural networks with rectified linear units. *ArXiv preprint arXiv:1611.01491*. 2016.

[4] Bishop, C. M. *Pattern Recognition and Machine Learning*. 1st ed. 2006. Corr. 2nd printing. Springer, 2006. 4 - 12 s. Information science and statistics. ISBN 9780387310732,0387310738.

[5] Bokui, Shen, G., Sethi, K., Zakka, O., Moindrot, R. et al. *CS231n Convolutional Neural Networks for Visual Recognition*. Dostupné z: https://cs231n.github.io/neural-networks-1/.

[6] Curry, H. B. The method of steepest descent for non-linear minimization problems. *Quarterly of Applied Mathematics*. 1944, sv. 2, č. 3, s. 258–261.

[7] Durville, A. a Myers, B. *Computer Vision with Convolution Networks* [online]. 2017 [cit. 2021-12-28]. Dostupné z: https://github.com/OKStateACM/AI_Workshop/wiki/Computer-Vision-with-Convolution-Networks.

[8] Girshick, R. Fast r-cnn. In: *Proceedings of the IEEE international conference on computer vision*. 2015, s. 1440–1448.

[9] Girshick, R., Donahue, J., Darrell, T. a Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, s. 580–587.

[10] Goodfellow, I., Warde Farley, D., Mirza, M., Courville, A. a Bengio, Y. Maxout networks. In: PMLR. *International conference on machine learning*. 2013, s. 1319–1327.

[11] He, K., Zhang, X., Ren, S. a Sun, J. Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, s. 770–778.

[12] Hinton, G., Srivastava, N. a Swersky, K. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*. 2012, sv. 14, č. 8, s. 2.

[13] John F. Hughes, A. v. D. *Computer Graphics: Principles and Practice*. 3rd Edition. Addison-Wesley Professional, 2013. 495–504 s. ISBN 0321399528,9780321399526.

[14] Kaushik, V. *8 Applications of Neural Networks* [online]. 2021 [cit. 2023-03-26]. Dostupné z: https://analyticssteps.com/blogs/8-applications-neural-networks.

[15] Kingma, D. P. a Ba, J. Adam: A method for stochastic optimization. *ArXiv preprint arXiv:1412.6980*. 2014.

[16] Koech, K. E. *Object Detection Metrics With Worked Example* [online]. 2020 [cit. 2023-06-10]. Dostupné z: https://towardsdatascience.com/on-object-detection-metrics-with-worked-example-216f173ed31e.

[17] Krizhevsky, A., Sutskever, I. a Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In: Pereira, F., Burges, C. J. C., Bottou, L. a Weinberger, K. Q., ed. *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2012, sv. 25. Dostupné z: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[18] Krizhevsky, A., Sutskever, I. a Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*. AcM New York, NY, USA. 2017, sv. 60, č. 6, s. 84–90.

[19] Kukačka, J., Golkov, V. a Cremers, D. Regularization for deep learning: A taxonomy. *ArXiv preprint arXiv:1710.10686*. 2017.

[20] LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R. et al. Handwritten Digit Recognition with a Back-Propagation Network. In: Touretzky, D., ed. *Advances in Neural Information Processing Systems*. Morgan-Kaufmann, 1989, sv. 2. Dostupné z: https://proceedings.neurips.cc/paper_files/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf.

[21] Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B. et al. Feature pyramid networks for object detection. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, s. 2117–2125.

[22] Lin, T.-Y., Goyal, P., Girshick, R., He, K. a Dollár, P. Focal loss for dense object detection. In: *Proceedings of the IEEE international conference on computer vision*. 2017, s. 2980–2988.

[23] Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R. et al. *Microsoft COCO: Common Objects in Context*. 2015.

[24] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. et al. SSD: Single Shot MultiBox Detector. In: Leibe, B., Matas, J., Sebe, N. a Welling, M., ed. *Computer Vision – ECCV 2016*. Cham: Springer International Publishing, 2016. ISBN 978-3-319-46448-0.

[25] Nielsen, M. A. *Neural Networks and Deep Learning* [online]. Determination Press, 2015 [cit. 2021-12-28]. Dostupné z: https://neuralnetworksanddeeplearning.com/chap1.html.

[26] Redmon, J., Divvala, S., Girshick, R. a Farhadi, A. You only look once: Unified, real-time object detection. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, s. 779–788.

[27] Ren, S., He, K., Girshick, R. a Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*. 2015, sv. 28.

[28] Rumelhart, D. E., Hinton, G. E. a Williams, R. J. *Learning internal representations by error propagation*. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[29] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A. a Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, s. 4510–4520.

[30] Sifre, L. a Mallat, S. Rigid-motion scattering for texture classification. *ArXiv preprint arXiv:1403.1687*. 2014.

[31] Simonyan, K. a Zisserman, A. Very deep convolutional networks for large-scale image recognition. *ArXiv preprint arXiv:1409.1556*. 2014.

[32] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. a Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*. JMLR. org. 2014, sv. 15, č. 1, s. 1929–1958.

[33] Tian, Z., Shen, C., Chen, H. a He, T. Fcos: Fully convolutional one-stage object detection. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, s. 9627–9636.

[34] Tinku Acharya, A. K. R. *Image Processing: Principles and Applications*. Wiley-Interscience, 2005. ISBN 9780471719984,0471719986.

[35] Wang, P., Fan, E. a Wang, P. Comparative analysis of image classification algorithms based on traditional machine learning and deep learning. *Pattern Recognition Letters*. Elsevier. 2021, sv. 141, s. 61–67.

[36] Weng, L. *Object Detection for Dummies Part 2: CNN, DPM and Overfeat*. 2017 [cit. 15-07-2023]. Dostupné z: https://lilianweng.github.io/posts/2017-12-15-object-recognition-part-2/.

[37] Weng, L. *Object Detection for Dummies Part 3: R-CNN Family*. 2017 [cit. 15-07-2023]. Dostupné z: https://lilianweng.github.io/posts/2017-12-31-object-recognition-part-3/.