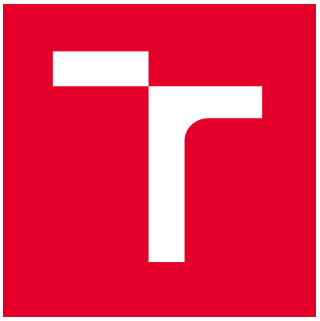


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

DETEKCE QR KÓDŮ POMOCÍ HLUBOKÉHO UČENÍ

QR CODE DETECTION USING DEEP LEARNING

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — **Matěj Černohous**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. Jiří Přinosil, Ph.D.**

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Telekomunikační a informační systémy**

Ústav telekomunikací

Student: Matěj Černohous ***ID:*** 229997

Ročník: 3 ***Akademický rok:*** 2022/23

NÁZEV TÉMATU:

Detekce QR kódů pomocí hlubokého učení

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je provést návrh algoritmu, který pomocí technik hlubokého učení umožní detekovat QR kódy v obrazech. Návrh by měl být koncipován tak, aby bylo možno detekovat QR kódy i za nepříznivých podmínek (více kódů v obraze, nízké obrazové rozlišení, částečné rozmazání, nerovnoměrné osvětlení, atd.). Navržený algoritmus bude implementován ve zvoleném programovacím jazyku. Výsledná implementace ověřena na reálných datech a porovnána s ostatními běžně používanými algoritmy.

DOPORUČENÁ LITERATURA:

Podle pokynů vedoucího práce.

Termín zadání: 6.2.2023 ***Termín odevzdání:*** 26.5.2023

Vedoucí práce: Ing. Jiří Přinosil, Ph.D.

prof. Ing. Jiří Mišurec, CSc.
předseda rady studijního programu

ABSTRAKT

Tato bakalářská práce se zabývá návrhem algoritmu pro detekci a dekódování QR kódů v obrazech využitím technik hlubokého učení. V rámci práce byly zhotoveny 2 datové sady, model neuronové sítě YOLOv7 pro detekci QR kódu v obraze, model neuronové sítě YOLOv4-tiny pro detekci pozičních bodů QR kódu a program v jazyce Python využívající těchto modelů pro čtení QR kódů v obrazech. Pro vyhodnocení byl algoritmus porovnán s jinými řešeními čtení QR kódů.

KLÍČOVÁ SLOVA

konvoluční neuronová síť, YOLOv7, YOLOv4-tiny, QR kód, datová sada, detekce objektů, Python

ABSTRACT

This bachelor thesis deals with the design of an algorithm for detecting and decoding QR codes in images using deep learning techniques. The work involved the construction of 2 datasets, a YOLOv7 neural network model for detecting QR codes in images, a YOLOv4-tiny neural network model for detecting position markers of QR codes, and a Python program utilizing these models to read QR codes in images. For evaluation, the algorithm was compared with other options for QR code reading.

KEYWORDS

convolutional neural network, YOLOv7, YOLOv4-tiny, QR code, dataset, object detection, Python

ČERNOHOUS, Matěj. *Detekce QR kódů pomocí hlubokého učení*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2023, 57 s. Bakalářská práce. Vedoucí práce: Ing. Jiří Přinosil, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Matěj Černohous
VUT ID autora:	229997
Typ práce:	Bakalářská práce
Akademický rok:	2022/23
Téma závěrečné práce:	Detekce QR kódů pomocí hlubokého učení

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Jiřímu Přinosilovi, Ph.D. za uvedení do problematiky, odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam výpisů

Úvod

Obor umělé inteligence, a především potom počítačového vidění, se poslední dekádu výrazně rozrůstá a inovuje. Také je díky jednoduchému přístupu k velkému množství dat skrze internet přístupnější vývoj modelů využívajících umělé inteligence. Rok od roku jsou vývojáři představené nové neuronové sítě, které výkonem a přesností nezanedbatelně předčí ty minulé. Tyto pokroky dělají detekování objektů v obraze pomocí neuronových sítí přístupné lidem, kteří by před 10 lety tento přístup neměli (ať už kvůli nedostatečně výkonému HW nebo nedostatku dat).

Motivací pro tuto práci je fakt, že i přesto, že technologie QR kódů je v dnešní době běžně využívána, často se stále používají zastaralé techniky pro jejich čtení, což může vézt k neschopnosti čtení některých QR kódů. Tato práce se zaměřuje na využití moderních technik počítačového vidění s cílem dosáhnout lepších výsledků při čtení QR kódů v obrazech.

Cílem této práce je navrhnutí a implementace algoritmu pro detekci a čtení QR kódů v obraze i za nepříznivých podmínek. Toto zahrnuje výběr správných neuronových sítí, tvorbu datových sad, natrénování a otestování modelů neuronových sítí na datech z datové sady a implementaci těchto modelů neuronových sítí v algoritmu pro čtení QR kódů v obraze.

V teoretické části práce bude uvedena a rozebrána problematika a terminologie týkající se neuronových sítí. V praktické části bude poté uvedeno konkrétní řešení návrhu a implementace algoritmu, stejně tak jako dosažené výsledky a srovnání s jinými řešeními čtení QR kódů.

1 Teoretická část studentské práce

Tato kapitola práce uvede pojmy umělé inteligence, strojového učení a neuronových sítí. Díky porozumění základní funkcionality a vnitřní struktury umělých neuronových sítí bude praktická část této práce lépe pochopitelná. V této kapitole bude také vysvětlena funkce a použití QR kódů, jejichž detekci v obraze se tato práce věnuje.

1.1 Umělá inteligence

Umělá inteligence (AI) je vědní disciplína tvořena z částí kognitivními vědami, kybernetikou a počítačovými vědami. Zabývá se zkoumáním a modelováním inteligence, a jejím cílem je vývoj SW a HW, jejichž postupy budeme moci považovat za projev lidské inteligence. Jejími základními oblastmi zkoumání jsou: obecné řešení problémů, plánování, reprezentace dat, rozpoznávání, reprezentace znalostí, adaptace a strojové učení. Umělá inteligence je v nynější době hojně aplikována v expertních systémech, zpracování přirozeného jazyka, počítačovém vidění, robotice, neuronových sítích a dalších[1].

Přístup, kdy programátor ručně naprogramuje velké množství explicitních pravidel pro manipulování dat je známý jako symbolické AI a byl v oboru dominantní do roku 1980. Tento přístup je vhodný pro řešení specificky definovaných logických problémů, jako je například hraní šachů. Pro komplexnější problémy (jako například klasifikace obrazu, rozeznávání řeči a překlad jazyka) je takřka neřešitelné přijít na explicitní pravidla. Proto se přešlo k přístupu strojového učení[2].

1.1.1 Strojové učení

Strojové učení (ML) je oborem umělé inteligence zaobírající se algoritmy a technikami pro automatizované řešení komplexních problémů, které je příliš obtížné řešit konvenčními metodami. ML algoritmy nevyžadují explicitně navržený design jejich funkce, ale učí se z množiny označených dat (datové sady). Cílem ML algoritmu tedy je naučit se model, neboli množinu pravidel, z označené sady dat, aby mohl tato pravidla aplikovat pro úspěšné předpovězení označení dat, která nebyla součástí původní sady dat[3].

Dá se tedy říci, že ML algoritmus není explicitně programován, ale spíše trénován. Strojové učení se rychle stalo nejpopulárnější oblastí AI díky dostupnosti rychlejšího HW a větších datových sad[2]. Je hojně uplatňováno pro rozpoznávání a kompresi obrazů, práci s akustickými nebo elektrickými signály, klasifikaci a segmentaci dat, rozpoznávání ručně psaného textu atd.

1.1.2 Počítačové vidění

Počítačové vidění je obor výpočetní techniky zabývající se vývojem autonomních systémů, které by mohly provádět takové vizuální úkony, které zvládají provádět lidé, neboli získání informací z obrazu. Příklady používání počítačového vidění jsou: autonomní navigace, robotická montáž, detekce jevů v obraze, výrobní inspekce, organizování databází, rozpoznávání obličejů apod[4].

Vývoj technik strojového učení a pokroky v oboru hlubokého učení přinesli nemalé pokroky do oboru počítačového vidění, neboť přesnost jejich algoritmů předčila pro účely jako klasifikace, segmentace a optický tok veškeré dřívější metody[5].

Detekce objektů

Detekce objektů je odvětví počítačového vidění zabývající se detekováním objektů v digitálním obraze v závislosti na jejich třídě (například člověk, auto, zvíře, text). Odvětví detekce objektů je nadále vyvíjeno nejen z hlediska akademického, ale i praktického - v oboru bezpečnostním, vojenském, zdravotnickém atd.

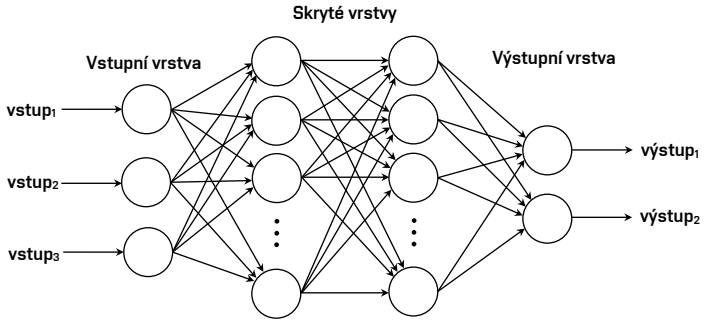
Stejně jako pro celé počítačové vidění, pokroky ve vývoji hlubokého učení urychlily vývoj detekce objektů a většina nejmodernějších objektových detektorů využívá sítě hlubokého učení[5][6].

1.2 Umělá neuronová síť

Umělá neuronová síť je výpočetní systém inspirován biologickými neuronovými sítěmi tvořící mozek. Je tvořena velkým počtem *umělých neuronů*, které mají být modelem neuronů mozku. Tyto umělé neurony jsou mezi sebou propojeny pomocí tzv. *vah*, kde každý vstup do neuronu je umocněn touto váhou, což ovlivní funkci vnitřního výpočtu. Neuronová síť tedy počítá funkci vstupů propagováním[1] vypočtených hodnot vstupních neuronů do výstupních neuronů používáním vah jako přechodných parametrů.

Samotné učení tedy probíhá změnou těchto jednotlivých vah, které propojují jednotlivé neurony, s cílem modifikace výpočetní funkce pro dosažení lepších výsledků[7]. Struktura neuronové sítě je znázorněna na obrázku 1.1. Skládá se z vrstev umělých neuronů, které jsou zapojeny za sebou. Neurony v první vrstvě mají na svém vstupu vstupní data, jedná se tedy o vstupní vrstvu. Každý neuron v další vrstvě má na svém vstupu výstup z každého neuronu předchozí vrstvy, těmto vrstvám se říká vnitřní, nebo také skryté vrstvy. Poslední vrstva se nazývá vrstvou výstupní a představuje výstup neuronové sítě. Neuronové sítě, která čítají více než jednu skrytou vrstvu, se nazývají hlubokými neuronovými sítěmi[8].

[1] propagace - proces přenosu informací mezi jednotlivými vrstvami umělé neuronové sítě



Obr. 1.1: Struktura umělé neuronové sítě s 2 skytými vrstvami

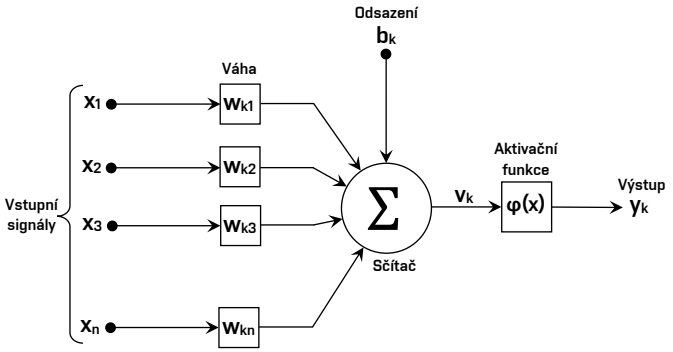
1.2.1 Umělý neuron

Model umělého neuronu je znázorněn na obrázku 1.2. Každý umělý neuron má několik vstupů x, z nichž každý je ohodnocen váhou w. Tyto váhy mohou nabývat kladných i záporných hodnot a určují propustnost daných vstupů. Určitý vstup je tedy vynásoben svou váhou a hodnoty jsou poté sečteny ve sčítači společně s b (bias), zvaným prahová hodnota/odsazení. Tato hodnota slouží k umělému navýšení/snížení vstupu aktivační funkce[9]. Umělý neuron k lze tedy popsat matematickými rovnicemi (1.1) a (1.2).

$$v_k = b_k + \sum_{j=1}^{n} w_{kj} \cdot x_j \tag{1.1}$$

$$y_k = \varphi(v_k) \tag{1.2}$$

První rovnice popisuje funkci sčítače, který sečte hodnotu odsazení b a vstupní signály x_j vynásobeny svými vahami w_{kj}. Tuto hodnotu nazýváme aktivační potenciál v_k. Druhá rovnice popisuje celkový výstup neuronu k, což je aktivační funkce φ, jejíž vstupem je aktivační potenciál v_k[7].



Obr. 1.2: Model umělého neuronu

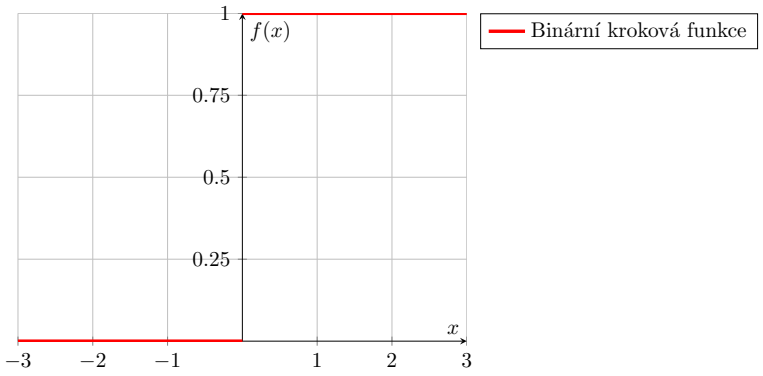
1.2.2 Aktivační funkce

Funkcí aktivační funkce je úprava hodnoty aktivační funkce na takovou hodnotu, aby ji mohl následující neuron přijmout a zpracovat. Bez použití aktivační funkce by byla výstupním signálem lineární funkce (polynom 1. stupně), což by učinilo neuronovou síť jednoduchou, ale nedala by se škálovat na komplikovanější úlohy. V praxi tedy chceme používat aktivační funkci, na jejímž výstupu bude hodnota v intervalu $[0; 1]$ nebo $[-1; 1]$. Při tvorbě samotné neuronové sítě na volbě aktivační funkce (nebo aktivačních funkcí v případě že různé vrstvy mají odlišné aktivační funkce) velmi záleží. Je běžné vyzkoušet více aktivačních funkcí a podle výsledných přesností se vybírá ta nejoptimálnější. Některé používané aktivační funkce jsou:

Binární kroková funkce

Jedná se o nejjednodušší aktivační funkci, která podle určité hodnoty hranice určí, zda bude daný neuron aktivovaný či deaktivovaný. Binární krokové funkce není možné použít pro klasifikaci více tříd a jsou tedy používány jen při binární klasifikaci (rozdělení vstupních dat do 2 diskrétních skupin). Vzhledem k tomu, že tato funkce nemá derivaci, tudíž je její gradient nulový, neumožňuje zpětnou propagaci chyb. Binární kroková funkce je definovaná jako (1.3) a její průběh vidíme na grafu 1.3

$$f(x) = \left\{ \begin{array}{lll} 1, & pro & x \geq 0 \\ 0, & pro & x < 0 \end{array} \right\} \tag{1.3}$$



Obr. 1.3: Průběh binární krokové funkce

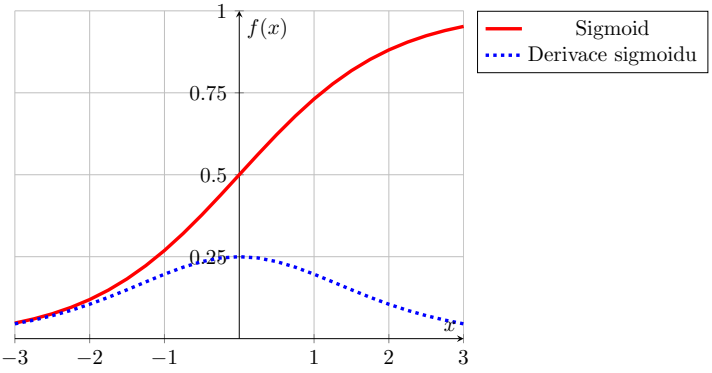
Sigmoid

Sigmoid je aktivační funkce, která transformuje výstup neuronu do intervalu $[0; 1]$. Hodnota tedy nenabývá záporných hodnot. Jedná se o nejčastěji používanou akti-

vační funkci[9]. Je definován vztahem 1.4 a jeho derivace je tedy 1.5. Průběh Sigmoidu a jeho derivace můžeme vidět na grafu 1.4. Problém u této aktivační funkce ale nastává v případě, kdy jsou vstupní hodnoty daleko od nuly. Tehdy se může zpomalovat učení sítě a v hlubších sítí může nastat problém mizijícího gradientu[2] (derivace)[9].

$$f(x) = \frac{1}{1 + \mathrm{e}^{-x}} \tag{1.4}$$

$$f'(x) = \frac{\mathrm{e}^{-x}}{(1 + \mathrm{e}^{-x})^2} \tag{1.5}$$



Obr. 1.4: Průběh Sigmoidu a jeho derivace

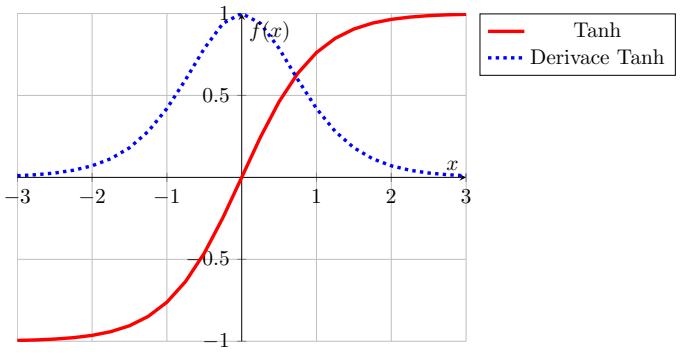
Hyperbolický tangens (tanh)

Hyperbolický tangens (značený také Tanh) je funkce podobná sigmoidu. Výstup neuronu je ale transformován do intervalu $[-1; 1]$. Stejně jako sogmoid je Tanh náchylný na problém s mizejícím gradientem, ale je preferovatelnější před sigmoidem, jelikož je to funkce soustředěná na nulu. Jeho první derivace je ale oproti derivaci sigmoidu strmější a nabývá více hodnot (popsán (1.7)). Díky tomu je funkce preferovavtelnější před sigmoidem[9]. Průběh Hyperbolické tangenty a její derivace je znázorněn v grafu 1.5.

$$f(x) = 2 \cdot f_{sigmoid}(2x) - 1 = \frac{\mathrm{e}^{x} - \mathrm{e}^{-x}}{\mathrm{e}^{x} + \mathrm{e}^{-x}} \tag{1.6}$$

$$f'(x) = \frac{4 \cdot \mathrm{e}^{2x}}{(\mathrm{e}^{2x} + 1)^2} \tag{1.7}$$

[2]mizející gradient: gradient je v zpětné propagaci algoritmu příliš malý, protože má tendenci se průchodem vrstvami snižovat a neovlivní pak parametry modelu



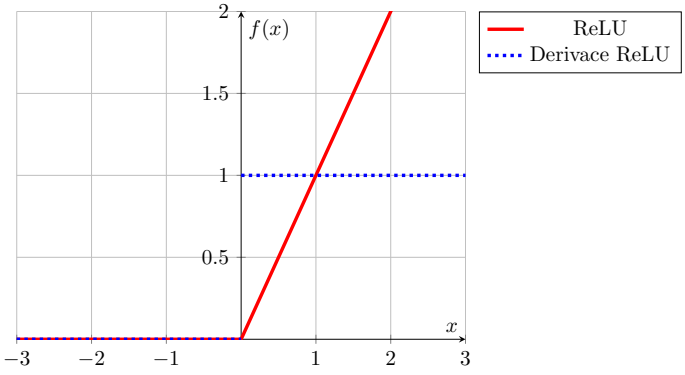
Obr. 1.5: Průběh Hyperbolické tangenty a její derivace

Usměrněná lineární funkce (ReLU)

ReLU (popsaná rovnicemi 1.8 a 1.9) je v neuronových sítí široce používána a v moderních hlubokých neuronových sítích je to standardní aktivační funkce. Je to hlavně díky její efektivnosti a tudíž rychlejšímu zpracování dat. Nevýhodou je ale nulová hodnota gradientu (derivace) pro některé případy, která zapříčiňuje to, že hodnoty vah a odsazení nejsou aktualizovány během algoritmu zpětného šíření chyb[3] v trénování dané neuronové sítě[9]. Průběh aktivační funkce ReLU je znázorněn na obrázku 1.6.

$$f(x) = \left\{ \begin{array}{lll} x, & pro & x \geq 0 \\ 0, & pro & x < 0 \end{array} \right\} \tag{1.8}$$

$$f(x) = \left\{ \begin{array}{lll} 1, & pro & x \geq 0 \\ 0, & pro & x < 0 \end{array} \right\} \tag{1.9}$$



Obr. 1.6: Průběh usměrněné lineární funkce a její derivace

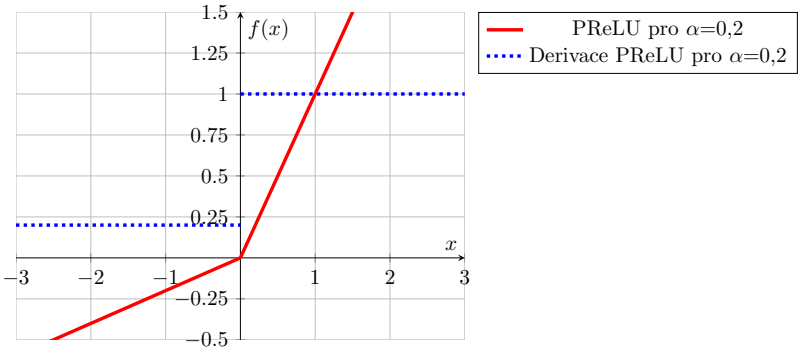
[3]zpětné šíření chyb, neboli Backpropagation, je algoritmus pro trénink dopředné neuronové sítě, s efektivnějším výpočtem gradientu

Parametrická usměrněná lineární funkce (PReLU)

Jedná se o modifikovanou ReLU funkci, která má navíc parametr α pro negativní část (viz rovnice 1.10). Existuje také varianta se zafixovaným parametrem $\alpha = 0,01$ (leaky ReLU - LReLU). Zásadnější je ale varianta, u které lze přímo parametr α učit [8]. Průběh aktivační funkce PReLU je znázorněn na obrázku 1.7.

$$f(x) = \left\{ \begin{array}{lll} x, & pro & x \geq 0 \\ \alpha \cdot x, & pro & x < 0 \end{array} \right\} \tag{1.10}$$

$$f(x) = \left\{ \begin{array}{lll} 1, & pro & x \geq 0 \\ \alpha, & pro & x < 0 \end{array} \right\} \tag{1.11}$$



Obr. 1.7: Průběh parametrické usměrněné lineární funkce a její derivace

Softmax

Aktivační funkce softmax, také nazývaná normalizovaná exponenciální funkce, je kombinací několika sigmoidových funkcí. Je často používána jako poslední aktivační funkce neuronové sítě. Nejčastěji se používá při klasifikaci s více třídami. Jeho funkce je taková, že převede vstupní vektor na vektor rozdělení pravděpodobností (jehož součet je roven 1) [9]. Jeho funkce je dána rovnicí 1.12.

$$f(x)_i = \frac{\mathrm{e}^{x_i}}{\sum_{j=1}^{\mathrm{K}} \mathrm{e}^{x_j}} \tag{1.12}$$

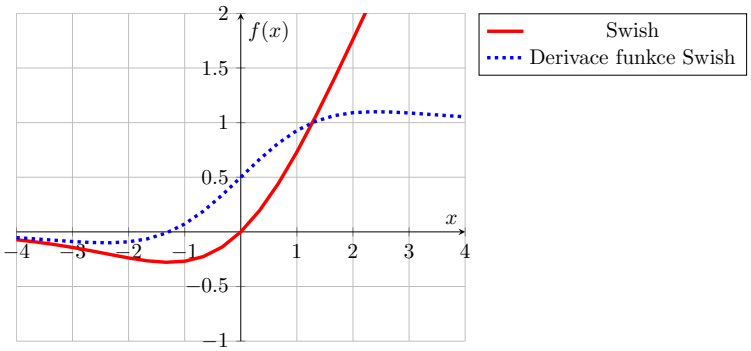
Swish

Aktivační funkce Swish byla vytvořena výzkumníky společnosti Google. V určitých případech se jedná o přímou náhradu funkce ReLU, jelikož u hlubokých neuronových sítí využívajících funkci ReLU přinesla její náhrada funkcí Swish zlepšení přesnosti klasifikace o $0,6 \div 0,9$ %. Mezi hlavní výhody patří jednoduchost a zvýšená přesnost.

Zásluhou toho je fakt, že tato funkce nemá problémy s mizejícím gradientem[9, 10]. Z rovnic 1.13 a 1.14 můžeme vidět její spojitost s funkcí sigmoid a na obrázku 1.8 znázorněný její průběh.

$$f(x) = x \cdot \sigma(x) = x \cdot \frac{1}{1 + \mathrm{e}^{-x}} = \frac{x}{1 + \mathrm{e}^{-x}} \tag{1.13}$$

$$f'(x) = \frac{\mathrm{e}^{x} \cdot (\mathrm{e}^{x} + x + 1)}{(\mathrm{e}^{x} + 1)^2} \tag{1.14}$$



Obr. 1.8: Průběh funkce Swish a její derivace

1.2.3 Trénování neuronových sítí

Po neuronové síti vyžadujeme, aby dosahovala co možná nejlepších výsledků. Proto využíváme schopnosti učení. Toto učení neuronových sítí probíhá úpravami vah a odsazení jednotlivých neuronů tak, abychom minimalizovali výslednou chybu sítě[11]. Základní dělení trénování, nebo také učení, neuronové sítě je:

- **Učení s učitelem**: Neuronové síti jsou předána kromě vstupních dat také správná výstupní data. Neuronová síť následně porovnává vlastní výstupní data se správnými a podle rozdílů určí chybu, podle které se upraví vnitřní parametry. Postup se opakuje do dosažení žádané minimální chyby.
- **Učení bez učitele** Neuronové síti jsou předána pouze vstupní data. Síť si tato data sama na základě odlišných vlastností a vztahů rozdělí do kategorií.
- **Posilované učení** Neuronová síť se učí pomocí propojování svého prostředí. K tomu je využíván tzv. agent, který se učí ze svých předchozích zkušeností a pamatuje si výsledky svých akcí. Na základě nich se rozhoduje při dalších akcích.

V této práci se budeme zabývat neuronovými sítěmi aplikujících učení s učitelem.

Chybová funkce

Chybová funkce (cost function) je funkce vyjadřující rozdíl mezi výstupní hodnotou neuronové sítě a správnou/očekávanou výstupní hodnotou. Je značena C. Některé příklady výpočtu chybové funkce je střední kvadratická chyba MSE(1.15) nebo střední absolutní chyba MAE(1.16). Nejpoužívanější je asi chybová funkce křížové entropie CE(1.17)[13].

$$C_{MSE} = \frac{1}{\mathrm{N}} \cdot \sum_{i=1}^{\mathrm{N}} (y_i - o_i)^2 \tag{1.15}$$

$$C_{MAE} = \frac{1}{\mathrm{N}} \cdot \sum_{i=1}^{\mathrm{N}} |y_i - o_i| \tag{1.16}$$

$$C_{CE} = \frac{1}{\mathrm{N}} \cdot \sum_{i=1}^{\mathrm{N}} \sum_{j=1}^{\mathrm{M}} y_{ij} \cdot log(p_{ij}) \tag{1.17}$$

Kde N je počet trénovacích vzorků.
y_i je očekávaný/správný výstup.
o_i je výstup neuronové sítě.
M je počet kategorií, které má model neuronové sítě rozpoznávat.
p_i je pravděpodobnost, se kterou si neuronová síť myslí, že vzorek je kategorie j.

Zpětná propagace chyb

Algoritmus zpětné propagace chyb se provádí u vícevrstvých neuronových sítí. Probíhá tak, že zprvu se provede dopředný průchod (forward pass) - síti předáme vstupní data, která síť s výchozími hodnotami vah a odsazení zpracuje a na výstupu dostaneme prvotní (pravděpodobně velice chybné) výstupní hodnoty. Ty jsou pomocí chybové funkce porovnány se správnými výstupními hodnotami. Výslednou hodnotu chybové funkce poté předáme neuronové síti ve výstupní vrstvě a dále se předává zpět skrze jednotlivé vrstvy až k vrstvě vstupní. Během předávání se v závislosti na vahách jednotlivých neuronů vypočítá parciální derivace (gradient) chybové funkce, na základě které jsou dané váhy aktualizovány. K tomu se využívá optimalizační algoritmus gradientního sestupu, kdy se po určitých krocích pohybujeme v opačném směru gradientu[12].

Hodnocení modelu

Ve strojovém učení je cílem to, aby byl náš model schopen *generalizovat* (zobecňovat) - neboli mít schopnost dobrého zpracování dat, které model již dřív nezpracovával. Je tedy potřeba spolehlivě hodnotit generalizační schopnost modelu[2].

Trénovací, validační a testovací set

Pro hodnocení modelu použijeme rozdělení dostupných dat do 3 setů: trénovacího, validačního a testového. Během trénování modelu na trénovacích datech je potřeba (kromě úpravy parametrů modelu) konfigurace takzvaných *hyperparametrů* modelu (velikost a počet vrstev). Úprava hodnot hyperparametrů probíhá během učení jako reakce na zpětnovazebný signál vyhodnocení validačních dat[2].

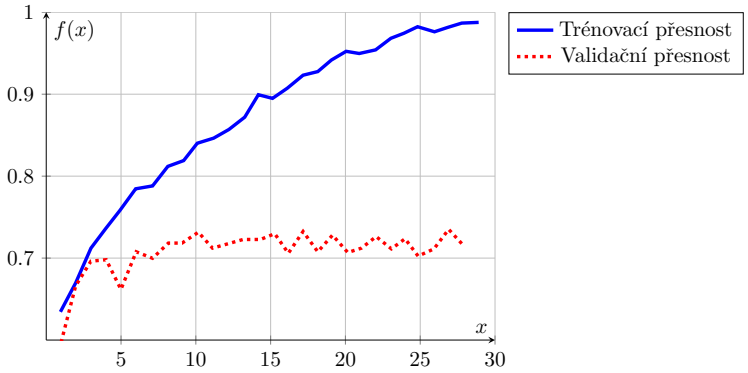
Projití celého trénovacího setu při učení nazývéme *epocha*. Počet epoch pro učení modelu hluboké neuronové sítě bývá v řádu stovek až tisíců (musíme dbát na to, aby nedošlo k *přeučení*, viz 1.2.3). Celé trénovací sady dat bývají ale pro jejich hromadné zpracování příliš velké, proto se provádí jejich rozdělení do menších částí, ty nazýváme *batch* - dávka.

Přeučení

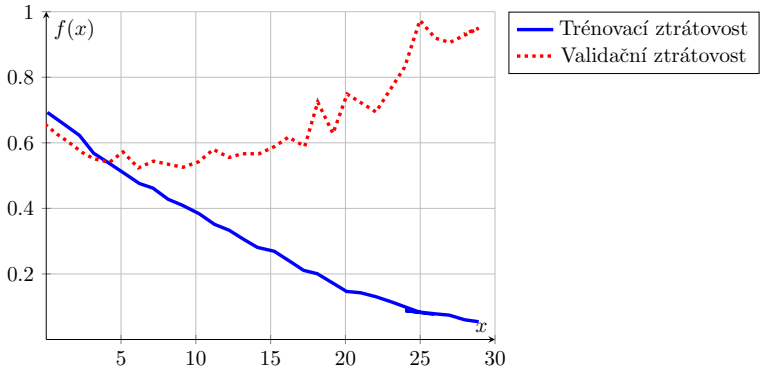
Přeučení (overfitting) se nazývá úkaz, kdy se neuronová síť adaptuje na nežádoucí vlastnosti trénovací sady dat a přichází o schopnost generalizace (schopností pracovat s dříve neviděnými daty). Dochází k němu při dlouhém učení (velkém počtu epoch) nebo při malé velikosti trénovací sady. Příklad přeučení můžeme vidět na grafech 1.9 a 1.10[2].

Přeučení může taktéž proběhnout na úrovní validace, kdy během validace data „prosáknou“, neboť hyperparametry modelu jsou upravovány podle výkonu na těchto validačních datech. Ačkoliv na nich tedy nejsou modely přímo trénovány, můžou při úpravě hyperparametrů „prosáknout“ některé informace o validačních datech do modelu. Tento fenomén se nazývá **informační prosakování**[2].

Jak můžeme vidět v grafu 1.9, zatímco průběhem trénování trénovací přesnost lineárně stoupá, validační přesnost stagnuje. V grafu 1.10 zase můžeme pozorovat, jak trénovací ztrátovost průběhem trénování klesá, ale validační ztrátovost naopak značne stoupat [2]. Přeučení můžeme předejít například augmentací dat, což je technika sloužící k umělému zvýšení množství dat přidáním mírně upravených kopií již existujících dat, čímž si uměle navýšíme velikost trénovací sady dat.



Obr. 1.9: Přeučení - trénovací a validační přesnost [2]



Obr. 1.10: Přeučení - trénovací a validační ztrátovost [2]

1.2.4 Vstupní data

Vstupními daty se rozumí sada anotovaných dat, která slouží pro trénování, validaci a testování algoritmu. Stejně jako volba samotné neuronové sítě, velikost a kvalita datové sady ovlivní přesnost konečného modelu sítě[19].

Datová sada pro detekci objektů

V případě detekce objektů se datovou sadou rozumí větší množství anotovaných obrázků obsahujících objekty, které má síť detekovat. Určitý obrázek je anotován v textovém souboru, ve kterém je pro detekované objekty určeno, kde se na obrázku nachází a o kterou třídu objektu se jedná - toto znázorňujeme *ohraničujícími boxy* (bounding box). Ve vhodné datové sadě by měla být každá třída objektů adekvátně zastoupena, a obrázky by měli být rozmanité. Pro případy testování funkce sítě se používají veřejně dostupné datové sady obsahující velké množství tříd na anotovaných obrázcích. Některými z nich jsou například **ImageNet** (14 milionů obrázků, 20 tisíc tříd), **COCO** (300 tisíc obrázků, 80 tříd) nebo **PASCAL VOC** (3 tisíce

obrázků, 20 tříd). Tyto datové sady jsou vhodné například pro porovnávání rozdílných neuronových sítí pro detekci objektů. V případě, kdy ale chceme detekovat například pouze několik určitých tříd objektů, je lepší volbou tvorba vlastní datové sady. Detekcí méně tříd objektů dosáhneme většího výkonu a můžeme dosáhnout i vyšší přesnosti[19].

Augmentace dat

Pro umělé zvětšení velikosti datové sady se používá takzvaná **augmentace** dat. Jedná se o proces, kdy nad originálními daty provedeme určitou transformaci, kterou získáme více datových vzorků. V případě detekce objektů můžeme datovou sadu obrázků upravit například jejich převrácením, otočením, rozmazáním, ořezáním, změnou jasu, změnou kontrastu, přidáním náhodného šumu atd. Tyto prvky augmentace můžeme zároveň mezi sebou kombinovat pro dosažení lepších výsledků. Neboť cílem augmentace dat je zlepšení generalizační schopnosti modelu, je augmentace dat prováděna pouze pro trénovací část datové sady. Jelikož validační a testovací data nejsou přímo používána ke změně parametrů modelu, augmentace validačních a testovacích dat nemá v tomto ohledu význam. V určitých případech se ale s augmentací validačních nebo testovacích dat setkáme.

1.3 Konvoluční neuronové sítě

Konvoluční neuronové sítě, často také nazývané ConvNet nebo CNN, jsou speciální hluboké neuronové sítě, které se vyznačují vynikajícími výkonem při zpracování obrazových, hlasových a jiných audio signálů. Konvoluční neuronové sítě se vyznačují tím, že mají 3 hlavní vrstvy, kterými jsou:

- Konvoluční vrstva (Convolutional layer)
- Sdružovací vrstva (Pooling layer)
- Plně propojená vrstva (Fully-connected (FC) layer [14]

Konvoluční vrstva

Konvoluční vrstva ve skládá z *filtrů*. Tyto filtry provádí nad vstupními daty matematickou operaci zvanou *diskrétní konvoluce*. Vrstva pracuje s parametry filtru:

1. rozměr - např. v případě černobílého obrázku délka a šířka udávající rozlišení obrázku a hloubka udávající jasové hodnoty hodnoty pixelů.
2. krok/posun (také stride) - udává, jak moc se budeme posouvat po vstupu
3. výplň (také padding) - rozšíření vstupu na takovou velikost, aby byl výstup konvoluje stejných rozměrů jako vstup. Při nepoužití výplně bude mít výstup menší rozměry než vstup.

Výstupem filtru je takzvaná *mapa příznaků*. To jsou naučené příznaky v prostorových dimenzích (jako třeba hrany, tvary). Konvoluční vrstvy často využívají více filtrů, kdy na výstupu každého filtru je mapa příznaků. Konečným výstupem konvoluční vrstvy je tedy více map příznaků - pro 3rozměrná data je konvoluce definovaná vztahem 1.18 [15, 14].

$$y[m,n,o] = \sum_{i=0}^{\mathrm{I}-1} \sum_{j=0}^{\mathrm{J}-1} \sum_{k=0}^{\mathrm{K}-1} h[i,j,k] * \mathbf{x}[m-i, n-j, o-k] \tag{1.18}$$

Kde $\mathbf{x}$ jsou vstupní (3rozměrná) data.
h je konvoluční filtr.
y je výstupní mapa příznaků.
m a n jsou řadnice pixelů ve vstupním obrázku.
o jsou jasové hodnoty pixelů.
I, J a K jsou rozměry konvolučního filtru.

Sdružovací vrstva

Účelem této vrstvy je snížení počtu parametrů, čímž dojde ke snížení výpočetní náročnosti celého modelu. Na jejím vstupu je aktivační mapa předaná minulou vrstvou. Opět se zde nachází filtr, jehož funkce je aplikace agregační funkce a zapsání výsledku do výstupního pole. Existuje více druhů sdružování, často se ale používá sdružení podle maxima - vybrání nejvyšší hodnoty, která je zapsána do výstupní matice. V této vrstvě na úkor snížení výpočetní náročnosti dochází k velké ztrátě informací[16].

Plně propojená vrstva

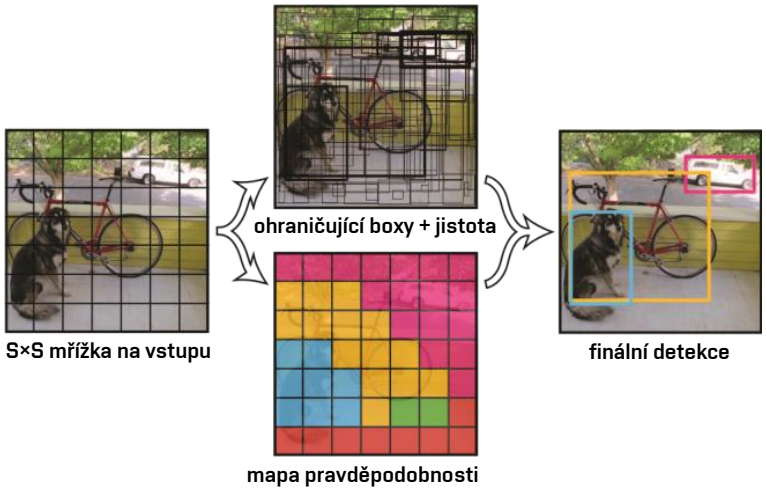
V této vrstvě je, jak je napovězenu v názvu, každý neuron vstupní vrstvy propojen s každým neuronem vrstvy výstupní (jako v běžné dopředné neuronové síti). Tyto vrstvy se používají za posledními konvolučními vrstvami. Transformují vstupní vektorová data na jednorozměrné hodnoty, které poté předají do následujících propojených vrstev. Ty na získané příznaky aplikují svoje váhy, čímž se pokusí určit výstup sítě[15].

1.3.1 YOLO

YOLO (You Only Look Once - Díváš se jen jednou) je konvoluční neuronová síť používáná k detekci objektů v reálném čase. Tento algoritmus provede detekci objektu v jediné regresi, hned po vložení vstupních dat tedy dojde k předpovědi jednotlivých tříd v ohraničujících boxech a určení jejich pravděpodobností. Díky tomu je algoritmus znatelně rychlejší než klasifikační algoritmy, jako je například R-CNN.

Narozdíl od technik detekce posuvným oknem a R-CNN algoritmus YOLO při trénování a testování zpracovává celý obraz, tudíž může implicitně zpracovávat informace o třídách a jejich charakteristikách kontextuálně. Díky tomu tento algoritmus například neoznačuje malé shluky pixelů v pozadí obrázku za objekty, neboť obrázek zpracovává v kontextu[17].

Na obrázku 1.11 je znázorněná takzvaná *sjednocená detekce* (unified detection). Vstupní obrázek je rozdělen mřížkou rozměrů $S \times S$. Pokud střed objektu náleží buňce mřížky, tato buňka je určená pro detekci tohoto objektu. Každá buňka tedy předpoví ohraničující boxy a jejich hodnoty jistoty. Tyto hodnoty jistoty značí, s jakou jistotou model předpovídá výskyt objektu v ohraničujícím boxu. Poté jsou pomocí hodnot pravděpodobností a jistot spočteny *pravděpodobnosti jistot individuálních boxů*, z čehož nám vyplyne pravděpodobnostní skóre pro určitou třídu každého boxu. Tato skóre obsahují pravděpodobnost toho, že určitá třída se nachází v tomto boxu, a také hodnotu jak dobře předpovězený ohraničující box obklopuje objekt[17].



Obr. 1.11: Znázornění sjednocené detekce[17]

Od roku jeho představení (2016) je YOLO vylepšována v nových představených verzích (YOLOv2 - 2017, YOLOv3 - 2018, YOLOv4 - 2020, YOLOv5 - 2020, YOLOv6 - 2022, YOLOv7 - 2022), kde každá z těchto verzí s sebou nesla určitá vylepšení pro dosažení lepších výsledků v oblasti detekci objektů. Do verze YOLOv4 byly algoritmy implementovány pomocí frameworku Darknet, od verze YOLOv5 je použit framework PyTorch.

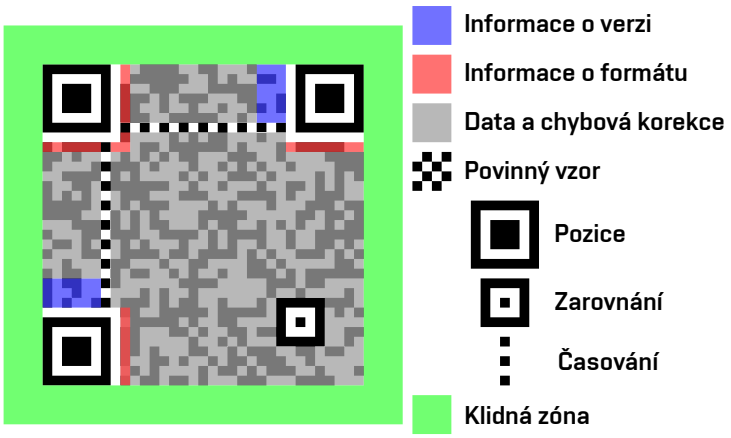
1.4 QR kódy

QR kódem rozumíme 2rozměrný kód navržený na to, aby byl přečten chytrými telefony. Zkratka QR (neboli Quick Response) indikuje, že kód by měl být velice

rychle dekódován, k čemuž byl tento kód navržen. Pomocí černých vzorů na bílém pozadí může být zakódován text, URL adresy nebo další data. V dnešní době jsou QR kódy hojně používány v různorodých odvětvích mimo jiné i díky tomu, že je běžné vlastnit chytrý telefon s fotoaparátem pro jeho přečtení.

Standardizace QR kódů je umožňuje komukoliv číst a zároveň vytvářet. QR kódy jsou standardizované na **verze** 1 až 40, kde každá verze udává jeho rozměry a další vlastnosti. Množství dat které lze QR kódem zakódovat je přímo uměrné jeho velikosti, v praxi se ale nejběžněji setkáme s QR kódy verzí 1 až 5, neboť velikosti těchto verzí jsou většinou dostačující. Na obrázku 1.12 je znázorněna struktura QR kódu verze 3. Jsou na něm ukázány vzory QR kódu a jejich umístění[22].

Kromě běžných verzí QR kódů existují další verze jako Micro QR kód, Logo QR kód, iQR kód, EQR kód a další. Tyto verze jsou používány pro specifické účely. QR kódy jsou v praxi často používány například pro sdílení jednoduchého textu, adres, číselných hodnot, URL odkazů, identifikací a mnoha dalších dat, které lze vyjádřit textem.



Obr. 1.12: Příklad struktury QR kódu[23]

2 Výsledky studentské práce

V této kapitole bude uvedená praktická část práce, stejně tak jako dosažené výsledky a srovnání s jinými řešeními. Budou zde tedy aplikovány poznatky z teoretické části práce a uvedeny nové informace potřebné pro návrh vlastních modelů konvolučních neuronových sítí, stejně jako způsob aplikace těchto modelů pro dekódování QR kódů v obraze.

2.1 Detekce QR kódu v obraze

První částí zpracování práce byl návrh modelu konvoluční sítě pro rozpoznání QR kódu v obraze. K tomuto účelu bylo potřeba zvolit správnou neuronovou síť, zvolit pro ni vhodné parametry a vytvořit datovou sadu, na které byla síť trénována a testována.

2.1.1 Volba neuronové sítě

Volba správného modelu neuronové sítě je velice důležitým krokem, neboť to ovlivní konečné výsledku výkonu modelu. Při výběru vhodné detekční neuronové sítě je třeba dbát na:

1. **rychlost detekce** - tento parametr je primárně důležitý pro modely, které jsou určené k detekci v reálném čase.
2. **přesnost** - v detekci objektů často uvažovaný parametr mAP - střední průměrná přesnost.
3. **hloubka sítě** - vyšší hloubka neuronové sítě bývá spjatá vyšší přesnost, ale stejně tak nižší rychlost a větší požadavky na HW.

Konečnou volbou modelu neuronové sítě byl model **YOLOv7** - v době zpracovávání práce nejmodernější verze neuronových sítí YOLO, který svou rychlostí a přesností překonává veškeré své předchůdce, stejně jako ostatní sítě pracující v reálném čase [18].

2.1.2 Datová sada

V rámci práce byla zkompletována vlastní datová sada[27] obsahující anotované obrázky s QR kódy. Jedná se většinou o fotografie QR kódů, které často obsahují i konvenčně nechtěné nepříznivé vlivy, jako jsou rozmazání, nízké rozlišení, nerovnoměrné osvětlení apod. Námi jsou ale tyto nepříznivé vlivy žádané, neboť chceme, aby natrénovaný model zvládal detekovat QR kódy vyfocené i za těchto podmínek.

Shromáždění dat

Pro dosažení co nejlepší přesnosti modelu je důležité se zaměřit na kvalitu a velikost samotné datové sady. Díky rostoucí popularitě detekce objektů a strojového učení obecně ale roste množství volně dostupných datových sad. Díky tomu můžeme nasbírat velké množství dat a data taktéž budou rozmanitá, neboť pochází z různých zdrojů. Konečná datová sada se skládá z kombinace 7 jiných datových sad online dostupných pod licencí CC BY 4.0[1]. Tyto datové sady [28, 29, 30, 31, 32, 33, 34] v součtu obsahovaly **3576** anotovaných obrázků. Bylo ale potřeba odstranit obrázky, které se napříč datovými sadami vícekrát opakovaly (kvůli předejití přeučení - viz 1.2.3) a taktéž obrázky, které by nebyly pro náš model přínosné (neobsahovaly QR kód, kvůli malému rozlišení byl QR kód nedetekovatelný apod.). Velikost datové sady poté činila **3044** obrázků ve formátu `.jpg`.

Anotace dat

Anotace v detekci objektů značí proces, ve kterém je na obrázku označená oblast, která obsahuje detekované objekty. Anotace jsou ukládány do textových souborů formátu `.txt` stejnojmenných se souborem obrázku. Každý řádek anotačního souboru odpovídá jedné anotaci ve formátu:

```
třída_id střed_x střed_y šířka výška
```

Kde `třída_id` značí, o kterou z rozlišovaných tříd objektů se jedná. V našem případě jediné třídy to bude vždy číslo 0.

`střed_x` značí pozici středu ohraničujícího boxu relativně k šířce obrázku.

`střed_y` značí pozici středu ohraničujícího boxu relativně k výšce obrázku.

`šířka` značí šířku ohraničujícího boxu relativně k šířce obrázku.

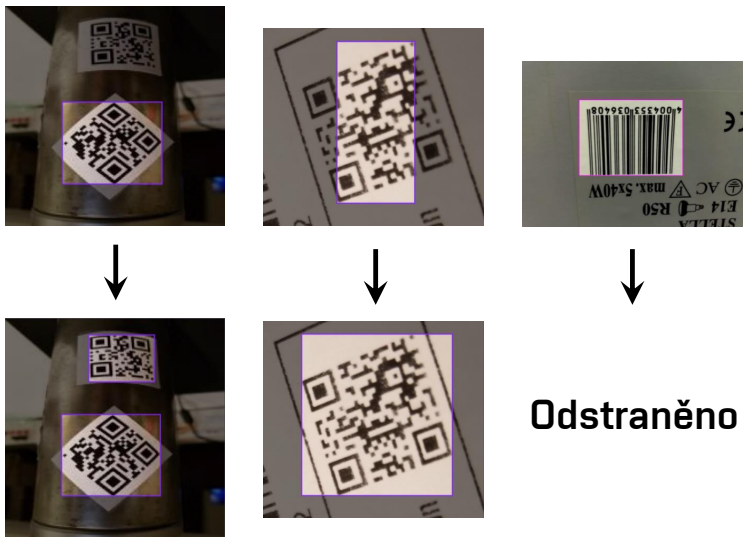
`výška` značí výšku ohraničujícího boxu relativně k výšce obrázku.

Tento formát anotace odpovídá formátu pro modely YOLOv7, ale pro jiné neuronové sítě se může odlišovat. Pro anotaci obrázků se často využívají nástroje nebo programy pro ulehčení a vizualizaci anotací ohraničujícími boxy. Pro tuto práci bylo k tvorbě a úpravě anotací datové sady použito rozhraní **Roboflow** (popsáno v 2.1.2).

Jelikož použité datové sady již byly předem anotovány, bylo potřeba jednotlivé anotace zkontrolovat, doplnit a upravit, aby byly anotace konzistentní napříč celou datovou sadou. Tento proces byl časově náročný, neboť bylo potřeba manuálně prohlédnout každý z obrázků a zkontrolovat (případně i dodělat/upravit) anotaci. Také bylo potřeba odstranit obrázky, které QR kód neobsahovaly vůbec, nebo z obrázku

[1] licence Creative Commons 4.0 umožňuje používání, sdílení a transformaci materiálů při splnění daných licenčních podmínek.

nebyl QR kód takřka rozpoznatelný (např. kvůli velice nízkému rozlišení obrázku nebo jiným nepříznivým podmínkám). Příklad provedených úprav v anotovaných obrázcích je na obrázku 2.1.



Obr. 2.1: Příklady provedených změn v původních anotacích obrázků

Roboflow

Roboflow je rozhraní, které umožňuje volný přístup k nástrojům pro aplikaci počítačového vidění. Kromě jiného Roboflow disponuje velkým množstvím datových sad pro detekci objektů (100 tisíc sad čítajících 100 milionů obrázků) a nástroji pro **anotaci**, **rozdělení** i **augmentaci** obrázků v datových sadách. Datové sady jsou volně přístupné pod licencí CC BY 4.0, což uživatelům umožňuje dostupné datové sady používat, sdílet a upravovat. Neboť proces anotace dat může být zdlouhavý, je volba správného nástroje pro ulehčení a zrychlení tohoto kroku důležitou částí při návrhu algoritmu počítačového vidění.

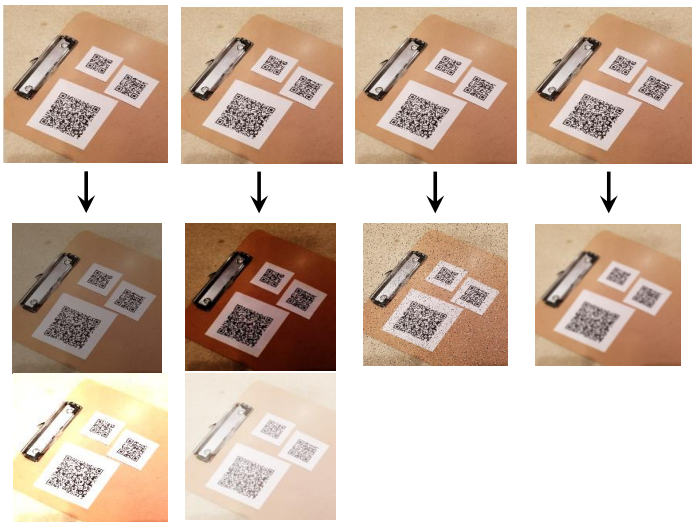
Rozdělení dat

Celková datová sada je potřeba rozdělit na část **trénovací**, **validační** a **testovací** (takzvaný *split*). S rozdělením se lze setkat ve tvaru *train-val-test* (procentuální poměr trénovací, validační a učící části) nebo *train-test* (kdy je validační část poměrově zahrnuta v té trénovací). Volba tohoto rozdělení dat může být důležitou částí pro optimalizaci modelu, ale pro prvotní verzi modelu se nejčastěji můžeme setkat s rozděleními: 80-10-10, 70-15-15, 60-20-20. Po vybrání poměru rozdělení jsou potom obrázky z celkové datové sady náhodně rozděleny do určených částí. Pro prvotní

verzi modelu bylo v této práci vybráno rozdělení **70-20-10**. Z datové sady čítající **3044** obrázků tedy byly pro uvedený poměr obrázky v poměru **2131-609-304**.

Augmentace dat

Pro augmentaci obrázků byly použity k tomu určené nástroje rozhraní Roboflow. Byly přidány augmentační prvky: **rotace**, **oříznutí**, **saturace**, **jas**, **expozice**, **rozmazání** a černobílý **šum** (také nazývám *salt and pepper* - sůl a pepř). Augmentace byly v zájemné kombinaci aplikovány nad množinou trénovací sady pro dosažení **trojnásobné** velikosti trénovací datové sady.



Obr. 2.2: Augmentace obrázku pomocí změny jasu, expozice, šumu a rozmazání

2.1.3 Příprava SW na implementaci YOLOv7

V této kapitole budou popsány důležité kroky potřebné pro možnost spuštění algoritmu YOLOv7. Podle druhu HW nebo SW můžou být konkrétní mezikroky odlišné, proto budou uvedené kroky pouze ty základní a nezbytné.

Stažení souboru YOLOv7

Soubory pro implementaci algoritmu YOLOv7 jsou přístupné[21] pod licencí GPL 3.0[2]. Po jejich stažení můžeme tedy kód upravovat, používat a sdílet, pokud ho uvedeme pod stejnou licencí. V dalších částech práce se tyto soubory uvažují jako umístěné ve složce `yolov7`.

[2]GPL 3.0 je obecná veřejná licence povolující kopírování a modifikaci programů, při splnění licenčních podmínek.

Python

Základním prerekvizitem použití programů modelu YOLOv7 je instalace programovacího jazyku Python (přesněji řečeno kompilaci knihoven programovacího jazyka C, které tvoří Python). Po instalaci je potřebné stáhnutí potřebných **balíčků** a **knihoven** (k tomu lze použít například *pip* - správce balíčku a knihoven pro Python). Seznam potřebných balíčků a knihoven je umístěny ve složce `yolov7` v souboru `requirements.txt`.

CUDA

CUDA je paralelní výpočetní platforma a programovací model vyvíjen společností NVIDIA. Je určena pro rozšíření funkcí výpočetních úkonů na jednotkách GPU. Její použití výrazně urychlí výpočetní úkony lepším využitím GPU. Jejím využitím tedy můžeme pouštět výpočetní programy psané v jazycích jako C, C++, Fortran, Python a MATLAB a díky přesunutí výpočetně náročné části programů na jádra GPU jednotek tak urychlíme chod programů. Pro použití této architektury je nutné stáhnutí programového balíčku CUDA Toolkit[25].

Uložení datové sady a konfigurace cest

Námi vytvořenou datovou sadu je potřeba mít umístěnou na stejném úložišti jako složku `yolov7`, ideálně v podsložce `data` této složky (pro jasnost a přehlednost). Datová sada musí být uložená ve správném formátu, tudíž aby obsahovala složky `train`, `test` a `valid` s rozdělenou datovou sadou, stejně jako soubor `data.yaml`. Každá ze složek `train`, `test` a `valid` by měla obsahovat složky `images` s jednotlivými obrázky a složku `labels` obsahující anotační textové soubory těchto obrázků.

V souboru `data.yaml` se nachází parametry `train`, `test` a `val`, kterým je potřeba přiřadit hodnotu cesty k obrázkům stejnojmenných podmnožin datové sady. Parametr `nc` určuje rozlišovaný počet tříd v datové sadě a parametr `name` jména jednotlivých tříd. Příklad nastavení parametrů:

Výpis 2.1: Datová sada - příklad nastavení parametrů souboru data.yaml

```
train: D:/yolov7/data/datova_sada/train/images
val: D:/yolov7/data/datova_sada/valid/images
test: D:/yolov7/data/datova_sada/test/images

nc: 1
names: ['qr_code']
```

2.1.4 Trénování modelu

Proces trénování slouží k úpravě vah modelu podle vstupních trénovacích dat. Výchozí hodnoty vah můžou být buď náhodné, nebo můžeme použít předtrénované váhy jiného modelu (například modelu trénovaném na datové sadě COCO). Ačkoliv způsobem použití předem trénovaných vah můžeme urychlit trénovací proces a dosáhnout pak vyšší přesnosti, u prvotního modelu bude využito výchozích náhodných vah.

Trénování samotné je prováděno spuštěním Python programu `train.py` (umístěného ve složce `yolov7`) se správnými parametry[24]. Některé důležité parametry:

`--weights` : cesta k počátečním vahám (výchozí hodnota: `'yolo7.pt'`)
`--cfg` : cesta ke konfiguračnímu souboru model.yaml (výchozí hodnota: `''`)
`--data` : cesta k datové sadě (výchozí hodnota: `'data/coco.yaml'`)
`--hyp` : cesta k hyperparametrům (výchozí hodnota: `'data/hyp.scratch.p5.yaml'`)
`--epochs` : počet epoch (výchozí hodnota: `300`)
`--batch-size` : velikost dávky pro všechny GPU (výchozí hodnota: `16`)
`--img-size` : velikost obrázků po předání modelu (výchozí hodnota: `[640, 640]`)
`--rect` : zda používat obdélníkové velikosti obrázků
`--resume` : zda pokračovat v posledním trénování (výchozí hodnota: `False`)
`--device` : zařízení CUDA (`0`, `1`, `2`, `3`, `cpu`) (výchozí hodnota: `''`)
`--workers` : počet podprocesů pro načítání dat (výchozí hodnota: `8`)
`--name` : název složky pro uložení dat (výchozí hodnota: `'exp'`)

Například v příkazovém řádku OS Windows by tedy spuštění trénování mohlo vypadat následovně:

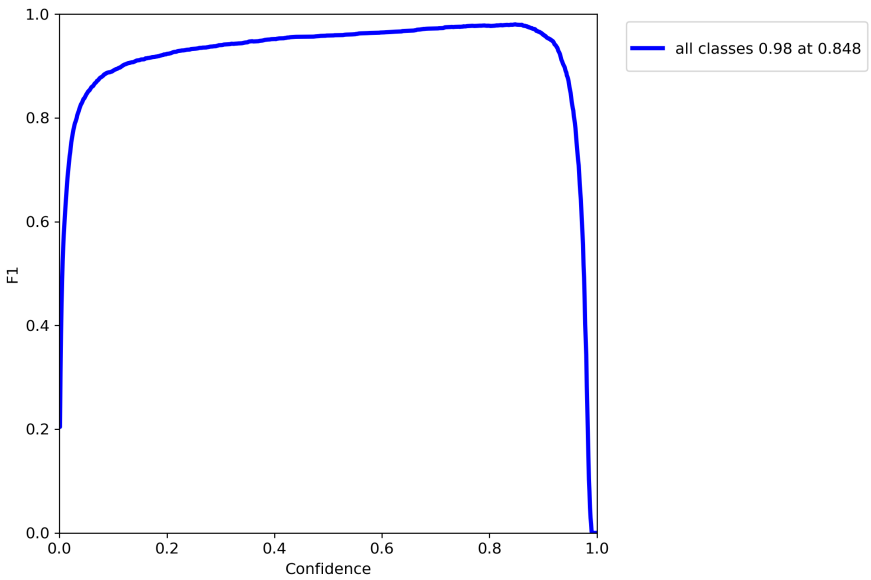
Výpis 2.2: Spuštění trénovacího procesu YOLOv7 v příkazovém řádku Windows

```
cd D:/yolov7
python train.py --workers 4 --device 0 --batch-size 8 --data
   data/datova_sada/data.yaml --img 640 640 --cfg cfg/
   training/yolov7_nc1.yaml --weights '' --name trenovani --
   hyp data/hyp.scratch.p5.yaml
```

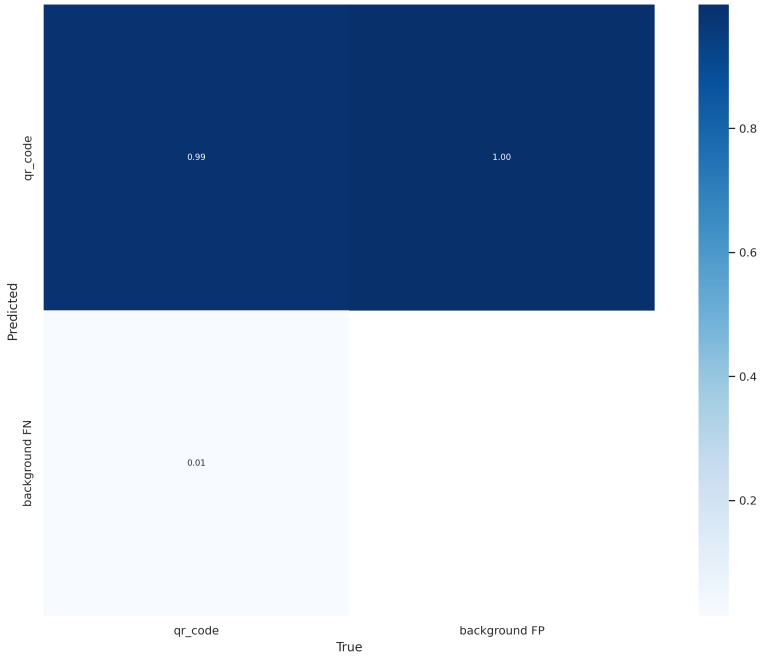
Z důvodu nároků na HW proběhlo trénování modulu na fakultním HW, který byl v rámci této práce zpřístupněn vedoucím práce. HW měl následující parametry:

CPU: AMD Ryzen 9 3900X 12-Core
RAM: 32 GB
GPU: NVIDIA GeForce RTX 2080 Ti

Proces trénování pro nastavených **300 epoch** trval **24,4 hodin**. Dosažené výsledky trénování modelu jsou zobrazené na grafech 2.3 a 2.4.



Obr. 2.3: F1 křivka trénování modelu



Obr. 2.4: Matice záměn trénování modelu

Pro porozumění grafů blíže upřesníme některé termíny:

1. **TP** (**pravdivě pozitivní výsledek**) - Detekování objektu tam, kde se nachází.
2. **FP** (**nepravdivě pozitivní výsledek**) - Detekování objektu tam, kde se nenachází.
3. **FN** (**nepravdivě pozitivní výsledek**) - Nedetekování objektu tam, kde se nachází.
4. **Confidence** (**Jistota**) - Hraniční hodnota u které model vyhodnocuje zda tvar detekuje jako objekt či nikoliv.
5. **Precision** (**Přesnost**) - Poměr správných pozitivních detekcí ku všem detekcím, definovaná vztahem 2.1.
6. **Recall** (**Schopnost rozpoznávání**) - Proporce správně detekovaných pozitiv, definováno vztahem 2.2.
7. **IoU** (**Průnik nad sjednocením**) - Metrika určuje, do jaké míry se 2 plochy (v našem případě předpovězený a správný ohraničující box) protínají 2.3.
8. **AP** (**Průměrná přesnost**) - Průměrná přesnost počítána jako plocha pod křivkou Přesnost-Recall.
9. **mAP** (**Střední průměrná přesnost**) - Parametry průměrné přesnosti napříč všemi třídami modelu, definován vztahem 2.4.
10. $\mathbf{F_1}$ - Parametr pro popsání přesnosti modelu z hlediska statické analýzy, definován vztahem 2.5.

$$Přesnost = \frac{TP}{TP + FP} \tag{2.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.2}$$

$$IoU = \frac{S_{průniku}}{S_{sjednocení}} \tag{2.3}$$

$$mAP = \frac{1}{n} \cdot \sum_{i=1}^{n} AP_i \tag{2.4}$$

$$F_1 = 2 \cdot \frac{mAP \cdot Recall}{mAP + Recal} \tag{2.5}$$

Z křivky F_1 na grafu 2.3 tedy můžeme určit hodnotu jistoty, která optimalizuje přesnost a schopnost rozpoznávání. Tato hodnota bude přibližně **0,82**, neboť při této hodnotě začíná křivka klesat[35].

Na matici záměn 2.4 můžeme pozorovat, že model vyhodnocoval při validaci správně **99 %** objektů, a **1 %** objektů nebylo detekováno.

2.1.5 Testování modelu

Procesem testování modelu už testujeme nyní nastavené váhy modelu na testovací sadě dat. Za předpokladu, že nedošlo k fenoménu **přeučení** a testovací sada dat je stejné kvality jako sady trénovací a validační, bychom bychom očekávat podobných výsledků, jaké byly výsledky validačních dat posledních epoch trénování.

Testování je provedeno obdobně jako trénování, tedy spuštěním Python programu `test.py` (umístěného ve složce `yolov7`) se správnými parametry. Důležité parametry, odlišující se od těch, které byly zmíněny v předchozí kapitole 2.1.4 jsou:

`--weights` : cesta k vahám pro testování (výchozí hodnota: `'yolov7.pt'`)
`--conf` : hranice přesnosti pro vyhodnocení objektu (výchozí hodnota: `0.001`)
`--iou` : hranice IoU pro úspěšnou detekci (výchozí hodnota: `0.65`)
`--task` : kterou z funkcí programu provést (výchozí hodnota: `val`)
`0.65`)
`--save-<formát>` : zda uložit výsledky testování ve vybraném formátu

Například v příkazovém řádku OS Windows by tedy spuštění testování mohlo vypadat následovně:

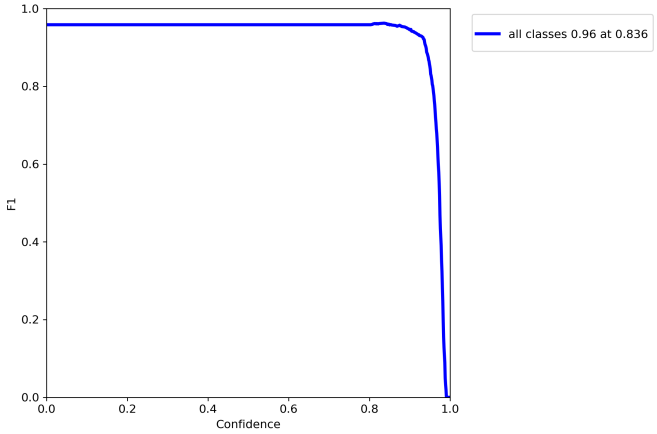
Výpis 2.3: Spuštění testovacího procesu YOLOv7 v příkazovém řádku Windows

```
cd D:/yolov7
python test.py --data data/datova_sada/data.yaml --img 640 --
   batch 32 --conf 0.8 --iou 0.65 --task test --save-txt --
   save-json --device 0 --weights runs/train/trenovani/
   weights/best.pt --name testovani
```

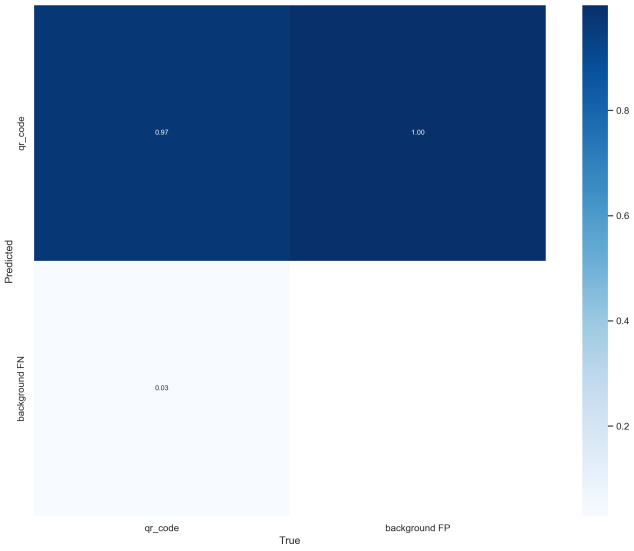
Proces spuštění testovacího programu se zadanými parametry trval řádově **desítky sekund**, přičemž samotné testování na **304** testovacích vzorcích trvalo **7 sekund**.

Dosažené výsledky jsou zobrazeny na grafech 2.5, 2.6 a obrázku 2.7.

Na matici záměn 2.6 můžeme pozorovat, že model vyhodnocoval při testování správně **97 %** objektů, přičemž **3 %** objektů nebyly detekovány. Příklad nedetekovaného QR kódu je červeně zvýrazněn na obrázku výsledků testování 2.6. Důvodem nižší úspěšnosti než při validaci dat ke konci trénovacího procesu může být příliš vysoce nastavená hodnota parametru `--conf`.



Obr. 2.5: F1 křivka testování modelu



Obr. 2.6: Matice záměn testování modelu

Obr. 2.7: Porovnání anotovaných a předpovězených obrázků testování modelu

2.2 Detekce pozičních bodů QR kódu v obraze

Pro další zpracování obrazu s QR kódem bylo potřeba najít každý ze tří pozičních bodů QR kódu.

Jejím vstupem je už samotný QR kód nalezený v obraze předchozí sítí, která obstarává právě detekci QR kódu v obraze. Díky tomu byly na tuto síť kladeny menší nároky na robustnost, neboť bude vstup méně rozmanitý a bude mít menší obrazové rozlišení.

2.2.1 Volba neuronové sítě

Pro volbu samotné neuronové sítě platí stejné požadavky uvedené v podkapitole 2.1.1. V tomto případě je ale vhodnější zvolit více „odlehčenou“ konvoluční síť, neboť vstupy této sítě budou méně rozmanité a menšího rozlišení, než tomu bylo u minulé sítě. První zvažovanou možností byla opět síť YOLOv7 s upravenými parametry sítě. Při úpravě parametrů sítě je ale vždy vhodné otestovat funkčnost takové sítě, což může být při opakované úpravě různých parametrů časově náročné.

Nakonec byl proto zvolen model **YOLOv4-tiny**, což je odlehčená verze modelu YOLOv4. Ačkoliv tato odlehčená verze dosahuje menší přesnosti než by dosahoval model YOLOv4 či YOLOv7, nabízí díky menší architektuře sítě (méně vrstev, menšímu počtu filtrů a sníženému rozlišení výstupu) větší rychlost, a to už jak trénování modelu, tak samotné detekce. Konfigurační soubor toho modelu je volně dostupný pod licencí pro volné užívání a šíření[36].

2.2.2 Datová sada

Pro trénování modelu bylo potřeba zkompletovat další datovou sadu[39], tentokrát s anotacemi jednotlivých pozičních bodů QR kódu, nikoliv už celého QR kódu. Obrázky datové sady byly upraveny do takového formátu, jaké budou výstupem sítě pro detekci QR kódu, neboli ořezané QR kódy již nalezené v obraze.

Shromáždění a úprava dat

Opět bylo využit již existujících datových sad (dostupných online pod licencí CC BY 4.0). Tyto datové sady [37, 38] obsahují anotované poziční body QR kódů v obraze, přičemž se ale kolem těchto QR kódů nachází pozadí a někdy se vyskytuje v obraze více QR kódů, což není žádané. Upravili jsme tedy datovou sadu do žádaného formátu oříznutím nalezeného QR kódu sítí popsanou v kapitole 2.1 a upravili souřadnice anotací, aby odpovídaly oříznutému obrázku.

Skript pro úpravu datové sady

Pro oříznutí obrázku s QR kódem a upravení souřadnic anotací jeho pozičních bodů byl vytvořen Python skript `my_adjust_position_labels.py`. Funkce skriptu je následující:

- Načtení cest k obrázkům a cest k příslušným anotacím obrázků do polí.
- Pro každý obrázek provést následující:
  - Pomocí funkce `my_detect` (blíže popsána v kapitole 2.3.2) nalézt sítí pro detekci QR kódu všechny QR kódy v obrázku, uložit jejich pozice a oříznuté obrázky do polí.
  - Upravit pole pozic QR kódů do vhodného formátu.
  - Načíst anotace pozičních bodů QR kódu z textového souboru do pole, převést pole do vhodného formátu.
  - Pro každý obrázek, kde se nachází právě 1 QR kód a k němu právě 3 poziční body, provést následující:
    - Od originálních souřadnic pozičních bodů odečíst souřadnice nalezeného QR kódu.
    - Tyto nové souřadnice uvést do správného formátu a uložit jako anotaci.
    - Nalezený oříznutý obrázek uložit jako obrázek k této anotaci.

Po dokončení funkce tohoto skriptu jsme obdrželi **1372 obrázků** QR kódů a k nim příslušné anotace jejich pozičních bodů. V rozhraní Roboflow byly následně anotace zkontrolovány a případně upraveny.

Rozdělení dat

V rámci rozdělení dat byl zvolen poměr **trénovací** a **validační** části **80-20**. Pro větší množství dat pro samotné trénování modelu a malé potřebě testování tohoto modelu byla trénovací část vynechána. Po aplikaci augmentace trénovací sady byla tedy konečná velikost datové sady **2613-501**. Celková velikost datové sady je tedy **3114** anotovaných obrázků.

Augmentace dat

Nad trénovací částí dat byla provedena augmentace. K tomu bylo opět použito nástrojů rozhraní Roboflow. Použité augmentační prvky byly: **saturace**, **jas**, **rozmazání** a **černobílý šum**. Díky augmentaci byla velikost trénovací sady ztrojnásobena. Ukázka některých anotovaných obrázků datové sady je na obrázku 2.8.

Obr. 2.8: Datová sada označení pozičních bodů QR kódu

2.2.3 Příprava SW na implementaci YOLOv4

Pro spuštění algoritmu YOLOv4 jsou jiné softwarové požadavky než pro YOLOv7. Opět budou popsány základní a nezbytné kroky pro zprovoznění práce s algoritmem YOLOv4. Primárním rozdílem je, že YOLOv4 nativně využívá framework Darknet, ale je možné ho implementovat i pomocí frameworku PyTorch, podobně jako YOLOv7.

Stažení souborů YOLOv4

Soubory pro implementaci algoritmu YOLOv4 jsou volně přístupné online[36]. Hlavní je stažení konfiguračního soubor `yolov4-tiny.cfg`. Je také možné si stáhnout předtrénované váhy `yolov4.weights`.

Framework

Pro trénování YOLOv4 modelu je potřeba instalovat vhodný framework - Darknet nebo PyTorch. Pro náš model byl využit framework Darknet, neboť už byl nainstalován na fakultním HW, na kterém probíhalo trénování modelu. PyTorch by ale byl validní alternativou. Instalace frameworku se provede stažením Darknet repozitáře[36] a po otevření souboru `darknet.sln` například ve Visual Studio Code zvolíme možnost *Build*.

Python

Pro aplikaci trénovaného modelu v programovacím jazyce Python jsou potřeba knihovny `OpenCV` a `numpy`. Tyto knihovny již byly nainstalovány pro funkčnost YO-

LOv7 a není tak potřeba instalace dalších knihoven.

Úprava datové sady a konfigurace cest

V souboru `yolov4-tiny.cfg` můžeme upravit parametry sítě jako **velikost obrázku** při zpracování (*width* a *height*), velikost **batch** - dávky, **počet filtrů** a další. Níže lze vidět ukázka námi upraveného konfiguračního souboru `yolo-tiny.cfg`. V případě potřeby vyšší rychlosti zpracování obrázku by bylo vhodná zvolit menšího rozlišení obrázku při zpracování - např 192x192 nebo 128x128.

Výpis 2.4: Ukázka konfiguračního souboru yolo-tiny.cfg

```
[net]
# Testing
#batch=1
#subdivisions=1
# Training
batch=32
subdivisions=1
width=416
height=416
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1
...
```

Dále je potřeba nastavit soubory `.data`, `.names` a `.txt`. Soubor `.data` obsahuje počet tříd, cestu k souborům `.names` a `.txt` a dále cestu pro uložení záložních souborů. Soubor `.names` obsahuje jména tříd (v našem případě pouze 1 třída - bod). Dále 2 soubory `.txt` obsahují cesty ke všem obrázků pro trénování a validaci (anotační soubory jsou stejnojmenné s obrázky, tudíž je není potřeba uvádět). Soubory byly uloženy jako `QR_point.data`, `QR_points.names`, `my_train.txt` a `my_valid.txt`.

2.2.4 Trénování modelu

Pro spuštění trénovacího procesu je potřeba v příkazové řádce spustit Darknet program `detector` v módu `train`. Následují parametry `<path_to_data_file>`

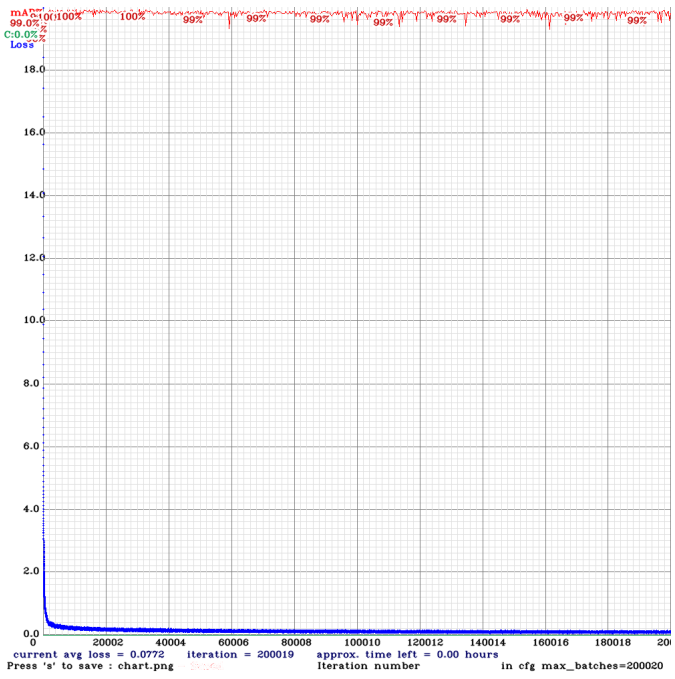
a `<path_to_config_file>`. Dále mohou následovat volitelné parametry jako:

`<- weights <weights_path>` : cesta k počátečním vahám
`<-dont_show>` : nezobrazovat trénovací proces v grafickém okně
`<-map>` : povoluje během trénování počítání parametru mAP
`<-clear>` : příkaz vyčistí váhy a logy z předešlých trénování
`<-gpus <gpus_list>` : specifikuje které jednotky GPU použít při trénování
`map_thresh <value>` : udává hodnotu pro vyhodnocení detekce jako kladného pozitiva při počítání mAP (výchozí hodnota: 0.5)

Výpis 2.5: Spuštění trénovacího procesu YOLOv4-tiny v příkazovém řádku Windows

```
detector train QR_point.data QR_points.cfg -dont_show -map
```

Trénování bylo opět provedeno na fakultním HW, stejně jako v případě trénování modelu YOLOv7. Trénování na 200019 iteracích (přičemž 1 iterace je trénování na 1 batch - dávce obrázků, následná validace a vypočtení mAP) trvalo necelé **3 dny**. Tento čas by mohl být zkrácen například zmenšením rozlišení obrázku při zpracování nebo zmenšením počtu iterací. Dosažené výsledky trénování modelu jsou zobrazené na grafu 2.9

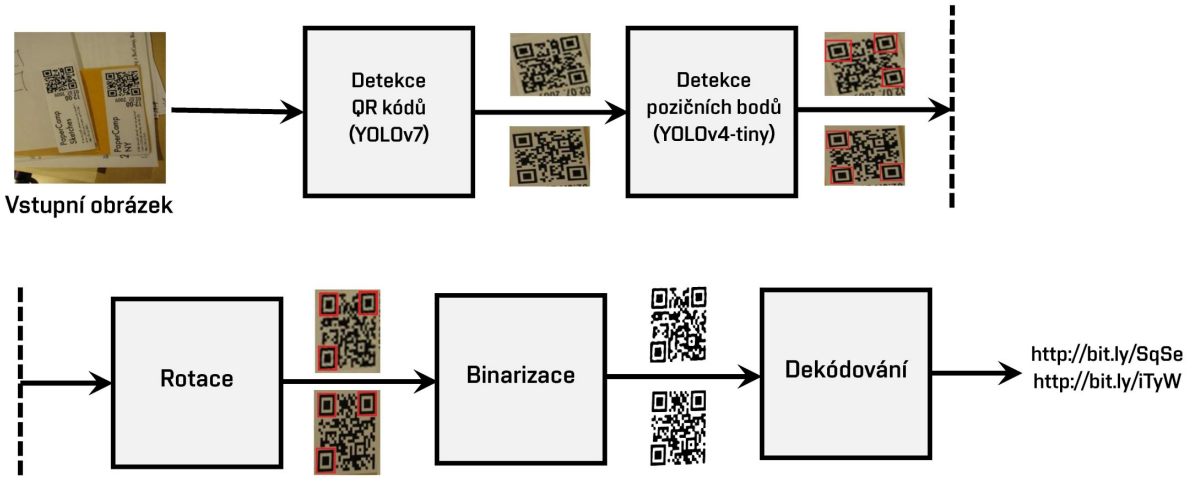


Obr. 2.9: Závislost chybovosti na iteracích trénování

Z grafu závislosti chybovosti na iteracích při trénování 2.9 lze odvodit, že při dokončení trénování byla **mAP přes 99 %**.

2.3 Čtení QR kódu

Pro aplikaci natrénovaných modelů neuronových sítí pro čtení QR kódů byl vytvořen Python program `my_read_qr.py`. Blokové schéma tohoto programu je znázorněno na obrázku 2.10.



Obr. 2.10: Blokové schéma programu `my_read_qr.py`

Program `my_read_qr.py` je uspořádán modulárně voláním funkcí z programu `my_functions.py`. Cesty ke klíčovým souborům jsou uloženy v programu `my_paths.py`. Účel a funkce jednotlivých bloků budou rozebrány v následujících podkapitolách.

2.3.1 Externí knihovny

V rámci programu je použito několik externích knihoven. Využití funkcí externích knihoven umožňuje nejen opětovné využití kódu, ale také rychlejší vývoj kódu a vyšší optimalizaci funkcí. Využity byly knihovny:

- `PIL` : (Python Imaging Library) je knihovna umožňující práci s obrázky v jazyce Python. Umožňuje především načítání, zobrazení, uložení a základní manipulaci s obrázky.
- `numpy` : Poskytuje funkce pro práci s multidimenzionálními poli. Využíváme možnosti převodu obrázku na multidimenzionální pole a následnou manipulaci s ním.
- `cv2` : Knihovna OpenCV je rozsáhlou knihovnu pro počítačové vidění, její hlavní využití je pro možnost obrazové detekce pomocí natrénovaného YOLOv4-tiny modelu.
- `math` : Knihovna matematických funkcí byla využita pro funkci výpočtu druhé odmocniny.

- `imutils` : Knihovna pro manipulaci s obrázky a videem. Byla využita funkce rotace, jejíž výstup byl kvalitnější než obdobná funkce knihovny `PIL`.
- `pyzbar` : Knihovna pro dekódování QR kódu, její funkce `decode` byla využita v poslední části programu `my_read_qr.py`.

2.3.2 Detekce QR kódu

Programu je na začátku programu `my_read_qr.py` dána cesta k obrázku, na kterém má být provedeno čtení QR kódů. Tato cesta je předána funkci `my_detect`, která navrátí pole souřadnic všech nalezených QR kódů v obrázku. Funkce `my_detect` je definována v programu `my_det.py`, což je upravený program `detect.py` sloužící pro obrazovou detekci pomocí YOLOv7 modelu. Tento program byl upraven především v tom, že veškeré volitelné parametry (kromě cesty k obrázku) byly nastaveny „napevno“. Dále bylo nastaveno, aby po nalezení QR kódů byly jejich souřadnice kromě vytisknutí také navráceny jako pole jednotlivých souřadnic a jejich hodnot jistoty.

Mezikrok 1

Mezi hlavními „bloky“ uvedenými ve schématu 2.10 jsou v programu prováděný různé mezikroky potřebné pro zpracování dat nebo jejich uvedení do vhodného formátu. Před blokem *Detekce pozičních bodů* jsou podle polí souřadnic navrácených minulým blokem **vyříznuty z vstupního obrázku QR kódy**, které jsou následně uloženy do pole obrázků.

V rámci oříznutí obrázků je kolem oříznutých QR kódů uměle přidána „**klidná zóna**“ (viz obrázek 1.12). Pokud není kolem QR kódu dostatek místa pro klidnou zónu, je pozadí klidné zóny vyplněno barvou nejsvětlejšího pixelu v oříznutém obrázku. Toto řešení se usvědčilo vhodné pro krok *Binarizace*.

Následující funkce programu jsou prováděny pro každý oříznutý QR kód samostatně.

2.3.3 Detekce pozičních bodů

V oříznutém QR kódu jsou využitím natrénovaného YOLOv4-tiny modelu detekovány jeho 3 poziční body. K tomu je využito externí knihovny `OpenCV`, díky jejíž funkcím lze využít obrazovou detekci modelem YOLOv4-tiny pro detekci pozičních bodů. Souřadnice těchto bodů jsou poté uloženy do vícerozměrného pole.

Mezikrok 2

Před blokem *Rotace* je pole pozičních bodů setříděno do pořadí podle jejich poloh v neotočeném QR kódu. Byla pro to napsána funkce, která si podle vzdálenosti jednotlivých bodů mezi sebou a vlastnosti přepony trojúhelníku vyvodí, který z bodů je bod rohový. Dále funkce podle souřadnic zbylých 2 pozičních bodů a zvážení všech možných rotací určí, který z bodů se má nacházet v levém dolním a který v pravém horním rohu. Funkce potom vrátí pole pozičních bodů setříděné ve správném pořadí.

2.3.4 Rotace

Pomocí souřadnic pozičních bodů jsou využitím funkcí externí knihovny `numpy` vypočteny úhly, který mezi sebou poziční body vzájemně svírají. V závislosti na souřadnicích pozičních bodů je poté vybrány nejvhodnější úhel, o který je následně obrázek QR kódu otočen a navrácen na výstupu funkce.

Princip výpočtu rotačního úhlu je zobrazen v rovnicích 2.6 a 2.7. V rovnici 2.6 (respektive 2.7) je pomocí funkce *arctan* vypočten úhel mezi kladnou osou x (respektive y) a přímkou spojující dva body $[x_1; y_1]$ a $[x_2; y_2]$. Funkce využívá tohoto principu, že v závislosti na rozpoložení souřadnic v prostoru vybírá body, podle kterých bude vypočten rotační úhel.

$$\text{úhel}_1 = \arctan\left(\frac{x_1 - x_2}{y_1 - y_2}\right) \tag{2.6}$$

$$\text{úhel}_2 = \arctan\left(\frac{y_1 - y_2}{x_1 - x_2}\right) \tag{2.7}$$

Mezikrok 3

Před blokem *Binarizace* je nyní rotovaný QR kód opět oříznut, aby byla zachována správná klidná zóna. Po rotaci jsou také znovu detekovány poziční body, neboť je toto řešení implementačně jednodušší, než přepočítávání jejich nových pozic po rotaci.

2.3.5 Binarizace

Krok binarizace obstarává převedení obrázku do podoby, kdy je každý z jeho pixelů vyhodnocen jako buď černý, nebo bílý. Pro binarizaci je nejdůležitější metoda nalezení **prahové hranice** pro určení toho, zda bude pixel vyhodnocený jako černý či bílý. Pro účel binarizace byly napsány 2 funkce zvažující **2 různé přístupy**. První funkce vypočte **průměrný jas pixelu** v oblasti QR kódu a podle této hodnoty určí hranici pro černý/bíly pixel. Druhá funkce používá takzvanou **adaptivní hranici** -

hranice pro černý/bílý pixel není stejná pro celý obrázek, ale je určena jako jas průměrného pixelu v okolí 51 pixelů (tato hodnota byla odvozena testováním různých hodnot). Nad obrázkem QR kódu jsou volány obě tyto funkce zvlášť, aby mohly jejich výsledky být porovnány v kapitole 2.4.

2.3.6 Dekódování

Po nalezení a zpracování QR kódu předchozími funkcemi jsou obě binarizované verze QR kódu předány dekódovací funkci z externí knihovny `pyzbar`. Při úspěšném dekódování QR kódu je jeho obsah vypsán do konzole. Pro srovnání je dekódovací funkce knihovny `pyzbar` použita nad vstupním obrázkem obsahující QR kódy ještě před jeho zpracováním. Výsledky srovnání jsou uvedeny v kapitole 2.4. Příklad možného výpisu konzole jednoho běhu programu je zobrazen na obrázku 2.11. Jak lze z výpisu vidět, z počítače byl nahrán obrázek, na kterém byl detekován 1 QR kód. Ten byl potom zpracován a při dekódování dat se podařilo data dekódovat jen při binarizaci metodou průměrného jasu pixelu.

```
PS C:\Users\mefit\Desktop\BAK\yolov7> python -u "c:\Users\mefit\Desktop\BAK\yolov7\my_read_qr"
Processing image: C:\Users\mefit\Desktop\BAK\comparison\pics\76.jpg
YOLOR  2022-10-21 torch 1.12.1+cu113 CUDA:0 (NVIDIA GeForce RTX 2060, 6143.6875MB)

Fusing layers...
RepConv.fuse_repvgg_block
RepConv.fuse_repvgg_block
RepConv.fuse_repvgg_block
IDetect.fuse
Model Summary: 314 layers, 36481772 parameters, 6194944 gradients
1 qr_code, Done. (23.0ms) Inference, (5.0ms) NMS
Done. (0.226s)

QR codes detected:
1

Now processing QR code number 1:

Reading Raw image:
[]

Reading transformed qr code: (Binarization with average threshold)
['GH69-28945C']

Reading transformed qr code: (Binarization with adaptive threshold)
[]
```

Obr. 2.11: Příklad konzolového výpisu běhu programu `my_read_qr.py`

2.4 Srovnání algoritmů

Pro vyhodnocení detekční schopnosti algoritmu byly jeho výsledky srovnány s jinými řešeními detekce QR kódů v obraze. Vytvořený algoritmus byl porovnán se třemi online webovými aplikacemi pro čtení QR kódů. Byl také porovnán s čtečkou

z externí knihovny `pyzbar`, která je sama o sobě v určitých případech schopna detekovat a dekódovat QR kódy v obraze ještě před samotným zpracováním tohoto QR kódu. Vytvořený algoritmus bude v porovnání zastoupen dvakrát, neboť vzájemně porovnáme oba způsoby binarizace (viz kapitola 2.3.5). S algoritmem tedy budou porovnávány následující možnosti čtení QR kódu:

- **Spikerog BR** : Online Barcode Reader - online čtečka QR a čárkových kódů, která zvládne detekovat a číst více QR kódů v jednom obrázku [40]. Vyvinuto společnosti Spikerog SAS.
- **Inlite BR** : Inlite Barcode Reader Web API - online čtečka QR a čárkových kódů společnosti Inlite Research [41].
- **ZXing Decoder**: ZXing Decoder Online - online čtečka QR kódů, taktéž dostupná jako open-source knihovna pro čtení a generování QR kódů [42].
- **Pyzbar Decoder**: Knihovna `pyzbar` využívá knihovny `zbar` programovacího jazyka C pro extrakci a čtení QR kódů z obrázků [43].

2.4.1 Proces srovnání

Pro férové srovnání řešení bylo použito ručně vybraných **100 obrázků** obsahujících QR kódy. Obrázky z této srovnávací datové sady byly postupně vloženy na vstup každého z algoritmů. Za každý QR kód na obrázku, který algoritmus úspěšně dekódoval a zobrazil dekódovanou hodnotu, dostal 1 bod. Výsledky byly poté vyneseny do tabulky a z celkového počtu QR kódů byla vyhodnocena úspěšnost každého algoritmu.

2.4.2 Výsledky srovnání

Na **100 obrázcích** srovnávací sady se vyskytovalo dohromady **135 QR kódů**. Výsledek porovnání je vidět na tabulce níže.

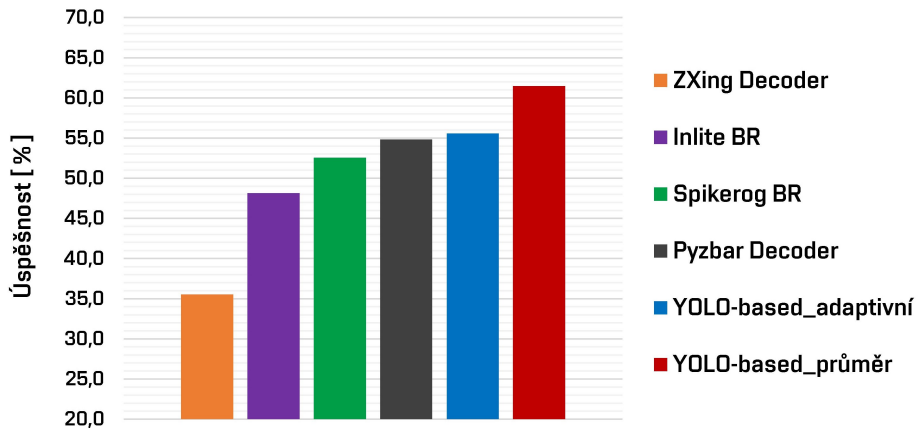
Algoritmus	**Úspěšnost [QR kódů]**	**Úspěšnost [%]**
YOLO-based$_{průměr}$	83	61,5
YOLO-based$_{adaptivní}$	75	55,6
Pyzbar Decoder	74	54,8
Spikerog BR	71	52,6
Inlite BR	65	41,8
ZXing Decoder	48	35,6

Obr. 2.12: Tabulka srovnání algoritmů pro čtení QR kódů v obraze

Jak lze vidět z tabulky 2.12, nejlépe si v čtení QR kódů v obraze vedl námi vypracovaný YOLO-based algoritmus s binarizací pomocí průměrného jasu s úspěšností **61,5 %** správně dekódovaných QR kódů. Na druhém místě se nachází YOLO-based algoritmus s adaptivní binarizací s úspěšností 55,6 %, následovaný knihovnou Pyzbar a dále stránkami Spikerog BR, Inlite BR a ZXing Decoder v tomto pořadí.

Veškerá data srovnání jsou ve složce `comparison` v příloze této práce. Na obrázku 2.14 lze vidět sloupcový graf znázorňující procentuální úspěšnou detekci QR kódů jednotlivých čteček. Dále lze na obrázku 2.13 vidět porovnání výstupů jednotlivých algoritmů pro několik obrázků ze srovnávací datové sady.

Obr. 2.13: Porovnání výstupů jednotlivých algoritmů



Obr. 2.14: Sloupcový graf úspěšnosti jednotlivých algoritmů

2.4.3 Vyhodnocení

Z dat srovnání vyplývá, že z porovnávaných algoritmů čtení QR kódů v obraze si vytvořený YOLO-based algoritmus vedl nejlépe s úspěšností **61,5 %**. I když si samotná čtečka Pyzbar vedle ve výsledku lépe, než zbylé online čtečky, její integrací v YOLO-based algoritmu jsme zvýšili její schopnost dekódovat QR kódy v obraze.

Pro další zlepšení vytvořeného algoritmu by bylo možné vyzkoušet více způsobů binarizace obrazu a sofistikovanější tranformace obrazu, neboť vytvořený algoritmus často selhal při čtení QR kódů na zakřiveném povrchu.

Závěr

Tato práce se zabývá návrhem algoritmu pro čtení QR kódu v obraze pomocí technik hlubokého učení. Toho je docíleno neuronovými sítěmi YOLOv7, YOLOv4-tiny a programem psaným v jazyce Python.

Teoretická část práce uvádí pojem umělé inteligence a strojového učení, a dále rozebírá základní vlastnosti a funkčnost neuronových sítí. Důležitými částmi jsou pojednávání o konvolučních neuronových sítí, datových sadách a uvedení algoritmu YOLO. V rámci této kapitoly jsou také vysvětleny základní vlastnosti QR kódů a jejich praktické využití.

Praktická část práce popisuje proces tvorby vlastní datové sady a nutné kroky pro trénování a testování modelů YOLOv7 a YOLOv4-tiny. V rámci práce byly dále zkompletovány 2 datové sady. Natrénované modely neuronových sítí byly poté integrovány do programu Python, ve kterém byl nalezený QR kód transformován a dekódován. Tento algoritmus je nakonec porovnán s několika jinými řešeními čtení QR kódů v obraze, přičemž náš algoritmus si vedl ze všech nejlépe s úspěšností dekódování QR kódů 61,5 %.

Hlavním přínosem práce byla tvorba datových sad pro detekci QR kódů a jejich pozičních bodů v obraze, natrénování modelů neuronových sítí YOLO nad těmito datovými sadami, integrace těchto modelů do Python programu pro čtení QR kódů v obraze a porovnání konečného algoritmu s jinými algoritmy pro čtení QR kódů v obraze.

Literatura

[1] KUČEROVÁ, Helena. Česká terminologická databáze knihovnictví a informační vědy (TDKIV): *Umělá inteligence* [online]. Národní knihovna ČR 2003. [cit. 17.11.2022]. Dostupné z URL: <https://aleph.nkp.cz/F/?func=direct&doc_number=000000137&local_base=KTD#tail>.

[2] CHOLLET, François. *Deep learning with Python*. Shelter Island, NY: Manning, [2018]. ISBN 978-161-7294-433.

[3] REBALA, G.; RAVI; A.; CHURIWALA, S. *An Introduction to Machine Learning*. Cham, Springer, [2019]. ISBN 978-3-030-15729-6.

[4] HUANG, T. S. *Computer Vision: Evolution and Promise*, Illinois: University of Illinois at Urbana-Champaign

[5] JIAO, L.; ZHANG, F.; LIU, F.; YANG, S.; LI, L.; FENG, Z.; QU, R. *A Survey of Deep Learning-Based Object Detection*, IEEE Access. 7: 128837–128868 DOI: <https://doi.org/10.1109%2FACCESS.2019.2939201>. S2CID: <https://api.semanticscholar.org/CorpusID:198147317>.

[6] ZHENGXIA, Z.; ZHENWEI, S.; YUHONG, G.; JIEPING, Y. *Object Detection in 20 Years: A Survey*, 2019 DOI: <https://doi.org/10.48550/arXiv.1905.05055>. arXiv: <https://arxiv.org/abs/1905.05055>.

[7] HAYKIN, S. *Neural Networks and Deep Learning*, Cham, Springer, [2018]. [cit. 21.11.2022]. ISBN 978-3-319-94463-0.

[8] GOODFELLOW, I., BENGIO, Y. a COURVILLE, A. *Deep Learning*, MIT Press, [2016]. [cit. 23.11.2022]. <http://www.deeplearningbook.org>.

[9] SHARMA, S. *Activation functions in neural networks*, Science, [2022]. [cit. 23. 11. 2022]. Dostupné z URL: <https://www.ijeast.com/papers/310-316,Tesma412,IJEAST.pdf>.

[10] NWANKPA, Ch. E.; IJOMAH, W.; GACHAGAN, A.; MARSHALL, S.. *Functions: Comparison of Trends in Practice and Research for Deep Learning*, [online]. [cit. 24. 11. 2022] Dostupné z URL: <https://arxiv.org/pdf/1811.03378.pdf>.

[11] DIKE, H. U., ZHOU, Y., DEVEERASETTY, K. K. a WU, Q. *Unsupervised Learning*, Based On Artificial Neural Network: A Review. In: 2018 IEEE International. Conference on Cyborg and Bionic Systems (CBS). 2018, s. 322–327. [cit. 24. 11. 2022] DOI: 10.1109/CBS.2018.8612259.

[12] ZHOU, X. *Understanding the Convolutional Neural Networks with Gradient Descent and Backpropagation*, Journal of Physics: Conference Series. IOP Publishing. [cit. 24. 11. 2022] DOI: 10.1088/1742-6596/1004/1/012028. Dostupné z URL:
<https://doi.org/10.1088/1742-6596/1004/1/012028>.

[13] YATHISH, V. *Loss Functions and Their Use In Neural Networks*, Towards Data Science, [2022] [cit. 24. 11. 2022]. Dostupné z URL:
<https://towardsdatascience.com/loss-functions-and-their-use-in-neural-networks-a470e703f1e9>.

[14] IBM Cloud Education *Convolutional Neural Networks*, New York, United States: IBM Cloud Education, [online]. New [2020] [cit. 26. 11. 2022]. Dostupné z URL:
<https://www.ibm.com/cloud/learn/convolutional-neural-networks>.

[15] ALBAWI, S., MOHAMMED, T. A. a AL ZAWI, S. *Understanding of a convolutional neural network*, International Conference on Engineering and Technology (ICET), [2017]. [cit. 03. 12. 2022]. DOI: 10.1109 ICEngTechnol.2017.8308186.

[16] DUMOULIN, V., VISIN, F. *A guide to convolution arithmetic for deep learning*, Cornell University [online]. Mila: Université de Montréal, [2018] [cit. 03. 12. 2022]. Dostupné z URL:
<https://arxiv.org/abs/1603.07285>.

[17] REDMON, J., DIVVALA, S., GIRSHICK, R., FARHADI, A. *You Only Look Once: Unified, Real-Time Object Detection*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), [2016]. [cit. 03. 12. 2022]. Dostupné z URL:
<https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Redmon_You_Only_Look_CVPR_2016_paper.html>.

[18] WANG, C., BOCHKOVSKIY, A., LIAO, H.M. *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*, [online]. [2022], [cit. 05. 12. 2022]. Dostupné z URL: <https://arxiv.org/abs/2207.02696>. DOI: <https://doi.org/10.48550/arXiv.2207.02696>.

[19] ZHOU, X., GONG, W., FU, W., DU, F. *Application of deep learning in object detection*, International Conference on Computer and Information Science, [2017]. [cit. 03. 12. 2022]. DOI: 10.1109/ICIS.2017.7960069.

[20] *About Roboflow*, Roboflow, Inc., [2022]. [cit. 08. 12. 2022]. Dostupné z URL: <https://roboflow.com/about>.

[21] WONG, K. *Implementation of paper - YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*, Github, [online]. [2022]. [cit. 08. 12. 2022]. Dostupné z URL: <https://github.com/WongKinYiu/yolov7>.

[22] TIWARI, S. *An Introduction to QR Code Technology*, International Conference on Information Technology (ICIT), [2016]. [cit. 09. 12. 2022]. DOI: 10.1109/ICIT.2016.021. Dostupné z URL: <https://ieeexplore.ieee.org/abstract/document/7966807>.

[23] Bobmath *QR Code Structure Example 3*, Wikimedia Commons, [online]. [2013]. [cit. 09. 12. 2022]. Dostupné z URL: <https://commons.wikimedia.org/wiki/File:QR_Code_Structure_Example_3.svg>

[24] DWYER, B. *How to Train YOLOv7 on a Custom Dataset*, Roboflow, Inc., [2022]. [cit. 08. 12. 2022]. Dostupné z URL: <https://blog.roboflow.com/yolov7-custom-dataset-training-tutorial/>.

[25] *About CUDA*, NVIDIA Corporation, [2022]. [cit. 09. 12. 2022]. Dostupné z URL: <https://developer.nvidia.com/cuda-zone>.

[26] LEBIEDZINSKI, P. *A Single Number Metric for Evaluating Object Detection Models*, Towards Data Science, [online]. [2022]. [cit. 09. 12. 2022]. Dostupné z URL: <https://towardsdatascience.com/a-single-number-metric-for-evaluating-object-detection-models-c97f4a98616d>.

[27] ČERNOHOUS, M. *xcerno30 QR codes Dataset*, Roboflow, Open Source Dataset [online]. [2022], [cit. 05. 12. 2022]. Dostupné z URL: <https://universe.roboflow.com/matej-cernohous/xcerno30-qr-codes>.

[28] capstonepercobaan1 *Data-set Dataset*, Roboflow, Open Source Dataset [online]. [2022], [cit. 05. 12. 2022]. Dostupné z URL: <https://universe.roboflow.com/capstonepercobaan1/data-set-bp3kt>.

[29] XU, L. *QR Code Dataset*, Roboflow, Open Source Dataset [online]. [2022], [cit. 05. 12. 2022]. Dostupné z URL: <https://universe.roboflow.com/lihang-xu/qr-code-oerhe>.

[30] *qr codes detection Dataset*, Roboflow, Open Source Dataset [online]. [2022], [cit. 05. 12. 2022]. Dostupné z URL: <https://universe.roboflow.com/roboflow-qsmu6/qr-codes-detection>.

[31] *Qr Dataset*, Roboflow, Open Source Dataset [online]. [2022], [cit. 05. 12. 2022]. Dostupné z URL: <https://universe.roboflow.com/qr-t0f6c/qr-nz97x>.

[32] *qrcode Dataset*, Roboflow, Open Source Dataset [online]. [2022], [cit. 05. 12. 2022]. Dostupné z URL: <https://universe.roboflow.com/qrcode/qrcode-0mr1u>.

[33] *qrcode-300 Dataset*, Roboflow, Open Source Dataset [online]. [2022], [cit. 05. 12. 2022]. Dostupné z URL: <https://universe.roboflow.com/qrcode-ali3n/qrcode_300>.

[34] *qrcodes-same-split-oneclass Dataset*, Roboflow, Open Source Dataset [online]. [2022], [cit. 05. 12. 2022]. Dostupné z URL: <https://universe.roboflow.com/qrcodes/qrcodes-same-split-oneclass>.

[35] LEBIEDZINSKI, P. *A Single Number Metric for Evaluating Object Detection Models*, Towards Data Science, [online]. [2022]. [cit. 09. 12. 2022]. Dostupné z URL: <https://towardsdatascience.com/a-single-number-metric-for-evaluating-object-detection-models-c97f4a98616d>.

[36] BOCHKOVSKIY, WANG, LIAO *YOLOv4: Optimal Speed and Accuracy of Object Detection*, [online]. [2020], [cit. 15. 5. 2023]. Dostupné z URL: <https://github.com/AlexeyAB/darknet/tree/master>.

[37] *custom-object-i5sli-dataset*, Roboflow, Open Source Dataset [online]. [2022], [cit. 15. 5. 2023]. Dostupné z URL: <https://universe.roboflow.com/yolo-jlfwl/custom-object-i5sli>.

[38] *real-qr-code-dataset*, Roboflow, Open Source Dataset [online]. [2022], [cit. 15. 5. 2023]. Dostupné z URL: <https://universe.roboflow.com/maxime-myrand/real_qr_code>.

[39] *xcerno30 QR Code pos*, Roboflow, Open Source Dataset [online]. [2022], [cit. 15. 5. 2023]. Dostupné z URL: <https://universe.roboflow.com/xcerno30qrcodecrop/xcerno30-qr-code-pos>.

[40] *Online Barcode Reader*, Spikerog SAS, [online]. [2018]. [cit. 16. 5. 2023]. Dostupné z URL: <https://www.onlinebarcodereader.com/>.

[41] *Inlite Barcode Reader Web API*, Inlite Research, [online]. [2023]. [cit. 16. 5. 2023]. Dostupné z URL: <`https://wabr.inliteresearch.com/`>.

[42] *ZXing Decoder Online*, Inlite Research, [online]. [2022]. [cit. 16. 5. 2023]. Dostupné z URL: <`https://zxing.org/w/decode.jspx`>.

[43] *Pyzbar 0.1.9*, Python community, [online]. [2023]. [cit. 16. 5. 2023]. Dostupné z URL: <`https://pypi.org/project/pyzbar/`>.

Seznam symbolů a zkratek

AI	artificial intelligence
apod.	a podobně
atd.	a tak dále
CNN	Convolutional neural network
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
ML	machine learning
HW	hardware
FN	False Negative
FP	False Positive
GPL	General Public License
GPU	Graphics Processing Unit
IoU	Intersection over Union
JPEG	Joint Photographic Experts Group
mAP	mean Average Precision
MSE	Mean Squared Error
např.	například
PReLU	Parametric Rectified Linear Unit
QR	Quick Response
R-CNN	Region Based Convolutional Neural Networks
RAM	Random Access Memory
ReLU	Rectified Linear Unit
TP	True Positive
URL	Uniform Resource Locator

A Obsah elektronické přílohy

- `yolov7` – kořenový adresář přiloženého archivu yolov7.zip
  - `cfg` – konfigurační .yaml soubory různých verzí YOLO
  - `data` – datové sady samotné nebo odkazy na ně
  - `models` – základní a experimentálními Python programy YOLO
  - `runs` – data ze spuštěných Python programů
    - `test` – data z testování modelů
      - `testovani` – data a výsledky z testování modelu
    - `train` – data z trénování modelů
      - `trenovani` – data a výsledky z trénování modelu
        - `weights` – váhy natrénovaného modelu
  - `tools` – poznámky k různým funkcím a vlastnostem YOLO
  - `utils` – pomocné Python funkce a programy
  - `train.py` – Python program pro trénování modelu YOLOv7/YOLOv7x
  - `train_aux.py` – Python program pro trénování modelu YOLOv7w6/YOLOv7e6
  - `test.py` – Python program pro testování modelu YOLOv7
  - `detect.py` – Python program pro detekci objektů modelem YOLOv7
  - `hubconf.py` – Python program pro konfiguraci hub modelu frameworku PyTorch
  - `export.py` – Python program pro export souborů na jiné systémy/platformy
  - `traced_model.pt` – PyTorch soubor umožňující trasování
  - `requirements.txt` – výpis knihoven a balíčků využívaných Python programy
  - `LICENSE.md` – textový soubor s licencí
  - `README.md` – textový soubor s informacemi k YOLOv7
  - `0.8.1` – soubor uživatelského manuálu přístupný Unix programy
  - `1.7.0` – soubor referencí přístupný v OS Linux
  - `my_adjust_position_labels.py` – Python skript pro úpravu anotací poz. bodů
  - `my_det.py` – upravený Python program pro detekci objektů modelem YOLOv7
  - `my_functions.py` – Python program pomocných funkcí
  - `my_paths.py` – Python konfigurační soubor s cestami k souborům
  - `my_read_qr.py` – Python program pro čtení QR kódů v obraze

- `datove_sady` – kořenový adresář přiloženého archivu datove_sady.zip
  - `xcerno30_QR` – adresář datové sady pro detekci QR kódu
    - `test` – soubory testovací datové sada
      - `images` – obrázky testovací datové sady formátu JPEG
      - `labels` – textové anotační soubory obrázků testovací datové sady
    - `train` – soubory trénovací datové sada
      - `images` – obrázky trénovací datové sady formátu JPEG
      - `labels` – textové anotační soubory obrázků trénovací datové sady
    - `valid` – soubory validační datové sada
      - `images` – obrázky validační datové sady formátu JPEG
      - `labels` – textové anotační soubory obrázků validační datové sady
    - `data.yaml` – konfigurační soubor datové sady
    - `README.dataset.txt` – textový soubor s informacemi k datové sadě od autora

- `README.roboflow.txt` ..... textový soubor s informacemi k sadě od Roboflow
- `xcerno30_QR_pos` ..... adresář datové sady pro detekci pozničních bodů QR kódu
  - `train` ..... soubory trénovací datové sada
  - `valid` ..... soubory validační datové sada
  - `README.dataset.txt` .. textový soubor s informacemi k datové sadě od autora
  - `README.roboflow.txt` ..... textový soubor s informacemi k datové sadě od Roboflow

`yolov4` ..... kořenový adresář přiloženého archivu yolov4.zip
- `backup` ..... Adresář zálažních souborů trénovacího procesu
  - `chart.png` ..... Graf ztrátovosti trénovacího procesu
  - `Yolo-tiny_best.weights` ..... Soubor nejlepších vah natrénovaného modelu
  - `Yolo-tiny_final.weights` ..... Soubor finálních vah natrénovaného modelu
  - `Yolo-tiny_last.weights` ..... Soubor posledních vah natrénovaného modelu
- `detect_v4.py` ..... Python program pro detekci objektů modelem YOLOv4
- `my_train.txt` ..... soubor cest k obrázkům trénovací datové sady
- `my_valid.txt` ..... soubor cest k obrázkům validační datové sady
- `QR_points.data` ..... konfigurační a informační soubor YOLOv4-tiny
- `QR_points.names` ..... soubor jmen objektů modelu YOLOv4-tiny
- `yolo-tiny.cfg` ..... konfigurační soubor parametrů sítě YOLOv4-tiny

`comparison` ..... kořenový adresář přiloženého archivu comparison.zip
- `srovnavaci_sada` ..... obrázky srovnávací datové sady
- `Comparison.xlsx` ..... MS Excel dokument s daty srovnání čteček QR kódů