2024

Křesťanské gymnázium

Kozinova 1000

102 00 Praha 10 – Hostivař

Detekce dopravních značek na mobilních zařízeních

JAN ŠTAFFA

Prohlášení

Prohlašuji, že jsem maturitní práci zpracoval samostatně a že jsem uvedl všechny použité informační zdroje a literaturu v seznamu použitých zdrojů.

Poděkování

Děkuji všem, kteří mě při práci jakkoliv podpořili, především Mgr. Ivoně Spurné za vedení práce.

Abstrakt

Tato práce se zabývá detekcí dopravních značek na mobilních zařízeních. Cílem práce je vytvořit mobilní aplikaci, která dokáže detekovat dopravní značky v reálném čase a může pomáhat řidičům dělat lepší rozhodnutí při řízení. Pro detekci dopravních značek jsem využil neuronové sítě. Celkem jsem testoval čtyři modely, nejlepších výsledků dosáhl model MobilenetV2. Nejvyšší dosažená úspěšnost byla 80.7% mAP a průměrná rychlost na mobilních zařízeních byla 18.3FPS. Výstupem práce je, kromě samotné funkční aplikace, také datová sada českých dopravních značek, vytvořená kombinací reálných a syntetických dat. Práce se také zabývá pokročilým generováním těchto syntetických dat. Dále jsem testoval modely na nejrůznějších zařízeních a sledoval chování v různých situacích a při různých zátěžích.

Klíčová slova

neuronové sítě, dopravní značky, datová sada, detekce, mobilní aplikace, MobilenetV2, Python

Abstract

This thesis deals with the problém of detection of traffic signs on mobile devices. The goal of the thesis is to create a mobile application that can detect traffic signs in real time and can help drivers make better decisions while driving. I used neural networks to detect traffic signs. I tested a total of four models, the best results were achieved by the MobilenetV2 model. The highest success rate achieved was 80.7% mAP and the average speed on mobile devices was 18.3FPS. The output of the thesis is, in addition to the functional application itself, also a data set of Czech traffic signs, created by a combination of real and synthetic data. The work also deals with the advanced generation of this synthetic data. Furthermore, I tested the models on various devices and observed the behavior in different situations and under different loads.

Key words

neural networks, traffic signs, dataset, detection, mobile app, MobilenetV2, Python

Obsah

Prohlášení 1
Poděkování 2
Abstrakt 3
Klíčová slova 3
Abstract 3
Key words 3
Souhrny objektů 5
Seznam obrázků 5
Seznam grafů 5
Seznam rovnic 5
Seznam tabulek 5
1 - Úvod 6
2 - Teoretická část 8
2.1 Dopravní značení v České republice 8
2.2 Neuronové sítě 8
2.3 Konvoluční neuronové sítě 10
3 - Praktická část 11
3.1 Datová sada 11
3.2 Generování dat 12
3.3 Výsledná datová sada 16
3.4 Výběr modelu 17
3.5 Trénování 19
3.6 Porovnání modelů 21
3.7 Optimalizace 24
3.8 Tvorba mobilní aplikace 25
4 - Výsledky 27
5 - Závěr 31
Zdroje 32

Souhrny objektů

Seznam obrázků

Obrázek 1: Zvolené dopravní značky 8

Obrázek 2: Příklad jednoduché neuronové sítě 9

Obrázek 3: Příklad jednoduché konvoluční neuronové sítě 10

Obrázek 4: Příklad trénovací datové sady (foto autor) 11

Obrázek 5: Příklady úprav značek generátorem 13

Obrázek 6: Příklady vygenerovaných dat 15

Seznam grafů

Graf 1: Podíl zdrojů reálných obrázků ... 12

Graf 2: Podíl zdrojů obrázků pro generování ... 14

Graf 3: Pravděpodobnost výskytu dopravních značek při různých „násobcích“ ... 15

Graf 4: Rozložení tříd (značek) v datové sadě ... 16

Graf 5: Porovnání rozložení tříd(značek) v datové sadě před a po zamíchání ... 17

Graf 6: Loss funkce (vlevo) a learning rate (vpravo) pro model SSD MobilenetV2 320x320 ... 20

Graf 7: Loss funkce modelu SSD MobilenetV2 FPNlite 640x480 ... 21

Graf 8: Příklad PR grafu pro model SSD MobilenetV2 640x480 při hranici IOU 0.5 ... 23

Graf 9: Příklad Confusion matrixu vypočítaného pro model SSD MobilenetV2 320x320 ... 24

Graf 11: Efekt počtu jader na výkon modelu SSD MobilenetV2 320x320 ... 28

Graf 12: Porovnání výkonu modelů na různých zařízeních ... 29

Graf 13: Porovnání Precision Recall grafů sledovaných modelů při různých IOU hranicích ... 29

Graf 14: Porovnání confusion matrixů sledovaných modelů při IOU hranici 0.5 ... 30

Seznam rovnic

Rovnice 1: Precission (vlevo), Recall (vpravo) 22

Rovnice 2: Intersection Over Union 22

Seznam tabulek

Tabulka 1: Hodnoty IOU sledovaných modelů 27

1 - Úvod

Moderní technologie se každým rokem stále posouvají a posledním výkřikem tohoto vývoje je hromadná integrace umělé inteligence do každodenního života lidí. Umělá inteligence se dostává do nejrůznějších zařízení, kde řeší široké spektrum problémů. Jedním z odvětví, která budou z tohoto vývoje nejspíše čerpat nejvíce, je automobilový průmysl. Sen o létajících automatických vozidlech, která nepotřebují lidského řidiče, je možná blíže, než si myslíme. Avšak dnes v tomto sektoru stále vládnou lidští řidiči, kteří musí dodržovat předpisy, které zajišťují bezpečnou přepravu jak pro ně, tak pro ostatní řidiče. To však neznamená, že jim v tom nemohou moderní technologie trochu pomoci.

Prvním vozidlem, které takové technologie využilo, byl model Insignia značky Vauxhall (Opel) v roce 2008, později technologii adoptovaly značky BMW a Mercedes-Benz. Dnes nová vozidla většiny výrobců již detekci dopravních značek zařadila do své výbavy a od roku 2022 je dokonce asistenční systém detekující dopravní značky, do jisté míry, povinnou výbavou všech nových vozidel. (1)

Velký krok pro strojové rozpoznávání dopravních značek nastal v roce 1968, kdy byla, více než padesáti státy, podepsána *Vídeňská úmluva o dopravních značkách a signálech*, která přinesla standardizaci dopravního značení. Díky tomu mohou řidiči bez problémů cestovat do různých zemí a rozumět místnímu značení a stejně tak mohou fungovat i automatické detekční systémy. (2)

Jak ale značky v obrázku najít? Pro člověka, díky jeho výborné schopnosti adaptace a učení, je tento úkol relativně jednoduchý, ale pro počítač, který pouze vykonává přesně stanovené instrukce stále dokola, je velmi obtížný. Kdyby se to tak jen počítač dokázal naučit sám... A také že dokáže, řešením jsou neuronové sítě, které dokáží do jisté míry napodobit biologické chování lidského mozku a dokáží se tak samy učit z předložených dat. Již od začátku práce jsem tušil, že řešení problému detekce dopravních značek využitím neuronových sítí bude krok správným směrem, proto jsem také jiné metody ani nezkoušel.

Cílem této práce je vytvořit mobilní aplikaci využívající neuronové sítě pro detekci a identifikaci dopravních značek v reálném čase z kamery mobilního zařízení. Problematikou detekce dopravních značek se již zabývalo několik prací, ze kterých jsem při tvorbě této práce čerpal. (3)

Jako cíl jsem si dal úspěšnost detekce konkrétních značek v 65 % případů, při rychlosti alespoň 5 FPS (pět snímků za sekundu) na průměrně výkonném mobilním zařízení, tyto cíle byly dosaženy a v některých ohledech dokonce výrazně překonány.

2 - Teoretická část

2.1 Dopravní značení v České republice

V ČR používáme dva druhy dopravního značení: svislé a vodorovné. V této práci jsem se zabýval pouze značením svislým. Celkem takových značek existují vyšší desítky až stovky, vybral jsem proto pouze dvacet, podle mě nejdůležitějších, a jimi jsem se zabýval. Problém pak přináší dodatkové tabulky, které se u některých značek nachází (například vymezení zákazu stání jen na určitou dobu), těmito jsem se také nezabýval, protože porozumění textu je mimo rozsah této práce. (4)

Obrázek 1: Zvolené dopravní značky (4)

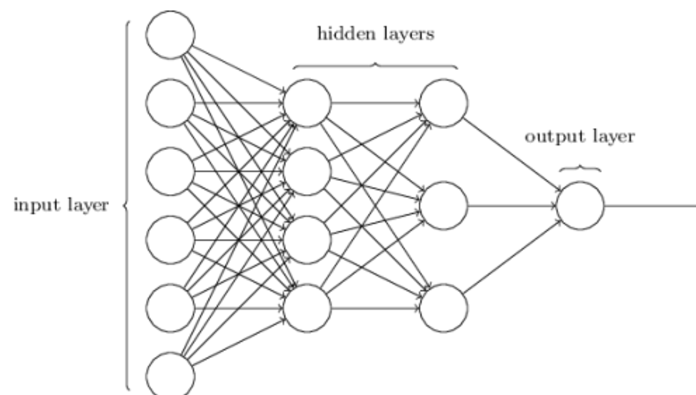
2.2 Neuronové sítě

Neuronové sítě, nebo také umělé neuronové sítě, jsou v informatice a umělé inteligenci zásadním nástrojem pro zpracování dat a řešení složitých úkolů, které by pro běžné algoritmy byly obtížně řešitelné. Jedná se o program inspirující se fungováním lidského mozku a jeho neuronových sítí. V poslední době se staly neuronové sítě středem pozornosti díky své schopnosti samy se učit z dat a adaptovat se na nové situace. Jejich využití mohou být pestrá, od doporučovacích systémů přes chatovací asistenty až po tzv. *computer vision* („počítačové vidění“), které se mimo jiné zabývá detekcí a rozpoznáváním objektů v obrázcích, což je pro tuto práci ideální.

Neuronová síť se skládá z jednotlivých neuronů, které jsou organizovány do vrstev. Na začátku sítě se nachází vstupní („input“*)* vrstva, která má z pravidla stejný počet neuronů,

jako je množství vstupních dat (př. pro obrázek 20x20 pixelů se třemi kanály bude mít 20x20x3 = 1 200 vstupních neuronů – jeden pro každý pixel). Tyto hodnoty jsou často normalizovány na desetinné číslo mezi 0 a 1. Po vstupní vrstvě následuje stanovený počet skrytých („hidden“) vrstev, které mohou obsahovat libovolný počet neuronů. Každý takový neuron má svou váhu („weight“), která ovlivňuje, jak moc je signál z předchozí vrstvy důležitý a aktivační funkci, která normalizuje hodnotu neuronu a zvyšuje komplexitu, což pomáhá zlepšit schopnost sítě pochopit drobné vlastnosti sledovaných dat. (5)

To, jak a co bude neuronová síť dělat, specifikuje široké pole parametrů, jako jsou typy vrstev, váhy, *biases* (konstantní čísla přičtená při jednotlivých operacích v síti) a aktivační funkce. Počet parametrů se velmi liší podle toho, co daná síť má dělat. Běžně se pohybuje v řádech jednotek až desítek milionů, ale může se vyšplhat až do astronomických hodnot, jako třeba 1.76 biliónu parametrů, kterými se může pyšnit ChatGPT 4. Souhrn těchto parametrů se nazývá model.



Obrázek 2: Příklad jednoduché neuronové sítě (6)

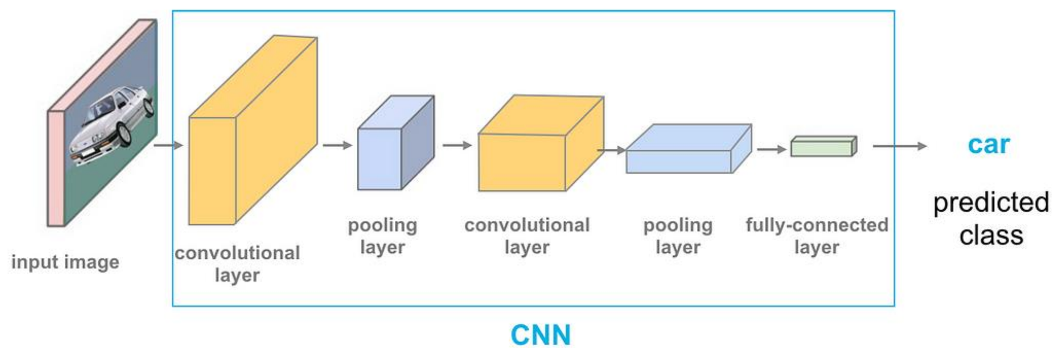
Vývojář může vytvořit model vlastní, ale pro většinu problémů existuje mnoho profesionálně navržených modelů, které jsou většinou mnohem rychlejší a přesnější, než amatérsky navržené modely.

Než ale můžeme začít využívat skvělých vlastností neuronových sítí, musíme je nejprve naučit, co vlastně mají dělat. Pro učení je však potřeba velké množství vstupních dat, v našem případě se jedná o obrázky dopravních situací. Více detailů ohledně datové sady v praktické části.

2.3 Konvoluční neuronové sítě

Konvoluční neuronové sítě („Convolutional Neural Networks“ – CNN) jsou druhy neuronových sítí, které se zabývají analýzou obrázků a následnou detekcí a klasifikací objektů v nich. Díky tomuto faktu jsou pro tuto práci ideální.

Tento druh umělých neuronových sítí funguje na principu segmentace obrázků a následné *feature extraction*, kdy se snaží najít známé vzory v obrázku, nejprve se může jednat pouze o čáry, čtverce, půlkruhy a jiné jednoduché tvary, v dalších vrstvách už se ale může jednat třeba o lidské obličeje nebo písmena a číslice. O tento proces se starají tzv. *convolutional layers („*konvoluční vrstvy“) a následné pooling layers („spojovací vrstvy“), které nalezené vzory zvýrazňují a snižují počet neuronů, před hustě spojenými vrstvami na konci modelu, které se snaží určit přesně o jaký objekt se jedná. (7)



Obrázek 3: Příklad jednoduché konvoluční neuronové sítě (8)

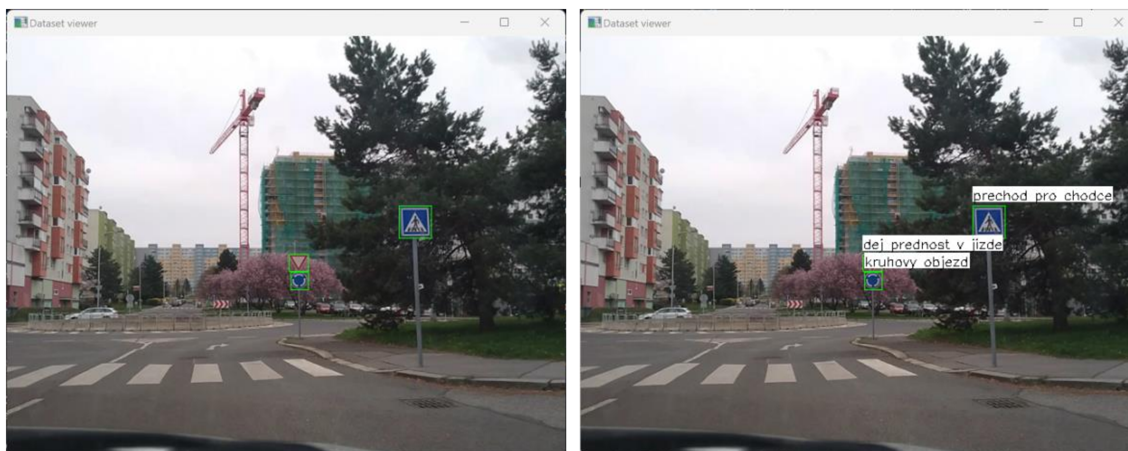
3 - Praktická část

3.1 Datová sada

Jádrem každého modelu jsou data, na kterých je model trénován. Volba těchto dat a jejich množství a různorodost je pro úspěšnost modelu absolutně zásadní. Než začneme vytvářet datovou sadu, je třeba se zamyslet nad tím, co chceme, aby model vlastně dělal. Pro detekci a klasifikaci, což je přesně to, čeho se snažíme docílit, je zapotřebí v datasetu obsáhnout jak samotné vstupní obrázky, tak i tzv. anotace – informaci, kde se v obrázku nachází jaký objekt (značka).

Základní poučkou při stavbě datové sady je, že vstupní (trénovací) data by měla odpovídat datům, které model uvidí při běžném používání. V tomto případě se jedná o záběry z palubních kamer.

Potřebné množství dat je závislé na druhu modelu – pro jednodušší modely, jako mohou být modely detekující pouze jednu třídu objektů, stačí pouze několik set příkladů, pro složitější však může požadovaný počet příkladů dosahovat milionů. V své práci jsem použil datovou sadu obsahující 2 643 obrázků s 4 359 vyznačenými značkami.

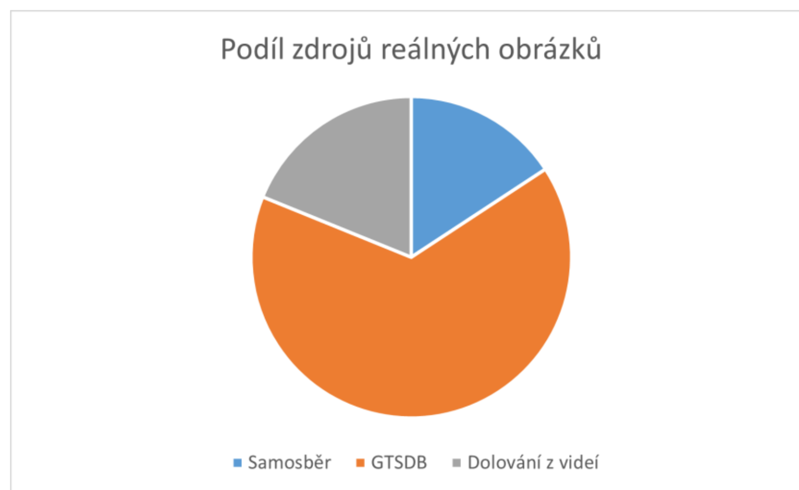


Obrázek 4: Příklad trénovací datové sady (foto autor)

Podobně jako s modely se vyplatí nejprve se podívat, zda již potřebná datová sada už neexistuje. V případě českých dopravních značek jsem bohužel po důkladném průzkumu veřejně dostupnou datovou sadu nenašel. Zato však existuje velmi pěkně zpracovaná datová sada německých dopravních značek – German Traffic Sign Recognition Benchmark (GTSDB) (9). Tato datová sada obsahuje 600 obrázků s 852 vyznačenými značkami rozdělenými mezi 43 tříd, což odpovídá cca 20 příkladům pro každou značku, což je relativně málo.

První generace modelů byla trénována pouze na německých datech, ale ukázalo se, že množství dat není dostatečné, a zároveň se projevily malé a znatelné rozdíly mezi českými a německými značkami, které pro běžného řidiče nejsou problémem, ale neuronovou síť spíše zmatou.

Tuto datovou sadu jsem rozšířil o 56 obrázků, které jsem sám nafotil a 67 obrázků, které jsem vydoloval z online videí. V těchto obrázcích jsem následně anotoval (vyznačil) dopravní značky pomocí programu LabelImg (10).



Graf 1: Podíl zdrojů reálných obrázků

3.2 Generování dat

Podstatou generování dat je využít reálná data a synteticky je rozšířit, aby syntetická data kopírovala reálná data co nejblíže, ale zároveň přidávala určitou náhodnost a variaci. Tímto způsobem se, do určité míry, můžeme vyhnout zdlouhavému a pracnému manuálnímu anotování dat.

Generace dat mi nejprve přišla pro tento případ nepoužitelná, o opaku mě však přesvědčila práce Filipa Kočici z VUT v Brně (3), ve které se autor zabýval velmi podobným problémem a elegantně jej vyřešil právě syntetickým generováním dat.

Před generováním je opět potřeba se zamyslet, na kterých datech vlastně chceme, aby se model učil. Jak jsem již výše popsal, vstupními daty mají být obrázky z úrovně palubních desek s vyznačenými dopravními značkami.

Při generování jsem nejprve vyříznul sledované dopravní značky (viz. obrázek 1) z různých obrázků, buďto stažených z internetu, nebo mnou nafocených, abych získal pouze individuální značky. Celkem jsem takto vytvořil 3 příklady pro každou značku. Myslím, že by datům mohlo

prospět větší množství těchto obrázků, ale kvůli časové náročnosti vyřezávání jsem se raději soustředil na jiné části projektu. Myslím, že přesto výsledné modely fungují velmi dobře.

Dále jsem vytvořil program, který každou z vyřízlých značek náhodně upraví několika způsoby, aby zvýšil variaci a náhodnost, protože, jak jsem výše napsal, chceme, aby syntetická data co nejblíže kopírovala reálná data a v reálných datech ne všechny značky vypadají stejně – některé jsou přesvětlené, ve stínu, zašlé nebo třeba trochu ohnuté. Při generování nových variant značek jsem použil následující úpravy:

- změna pozice značky – vždy
- změna velikosti značky – 50 % případů (1 ze 2)
- rozmazání značky – 20 % případů (1 z 5)
- přidání šumu – 17 % případů (1 z 6)
- oříznutí rohu značky – 7 % případů (1 z 15).

Obrázek 5: Příklady úprav značek generátorem

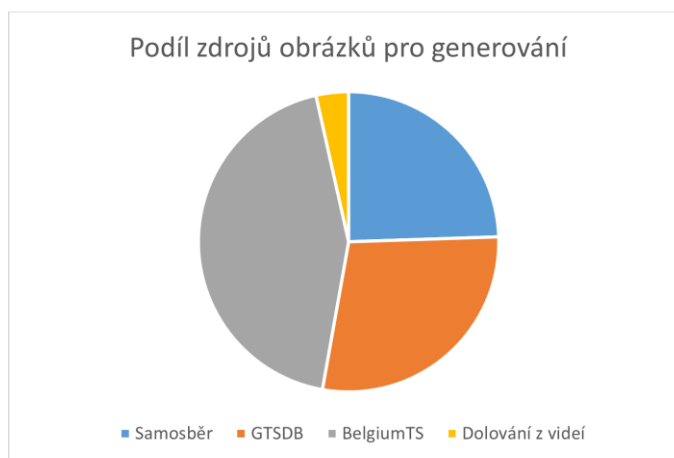
V první verzi datové sady jsem použil pouze první čtyři úpravy a po testování jsem objevil, že modely mají problémy s detekcí značek, které jsou příliš velké, například pokud je značka zabírána z blízkosti z chodníku nebo pokud jsou zčásti zakryté (dokonce stačilo jen aby před značkou byla tenká větvička a model už měl problém ji detekovat). Po tomto testování jsem přidal funkci, kdy s pravděpodobností 5 % (1 z 20) bude značka až 2 krát větší a také výše zmíněnou pátou úpravu, která náhodně ořízne jednu ze čtyř stran nebo rohů. Tyto úpravy problém z velké části vyřešily.

Poté, co byly vygenerovány různé varianty značek, bylo potřeba je vložit do prázdných obrázků různých dopravních situací. Tyto obrázky nesměly obsahovat žádné značky, aby zbytečně

nepletly model při učení. Část těchto obrázků pocházela z Belgické datové sady BelgiumTS (11), která přímo obsahuje sekci „Background images“, kde se nachází obrázky bez značek.

Celkem jsem jako pozadí pro generování využil 500 obrázků pocházejících z datové sady BelgiumTS, 162 obrázků bylo extraktováno z datové sady GTSDB, sám jsem nasbíral dalších 140 obrázků a dalších 20 jsem vydoloval z videí dostupných online. Poté byla část obrázků převrácena podle osy y, a tak se skoro zdvojnásobil počet vhodných obrázků pozadí na 1 144.

Během samotného generování pak byl každý obrázek pozadí využit dvakrát s jinými značkami, a tak byl dosažen celkový počet výsledných obrázků v syntetické části datové sady 2 288.

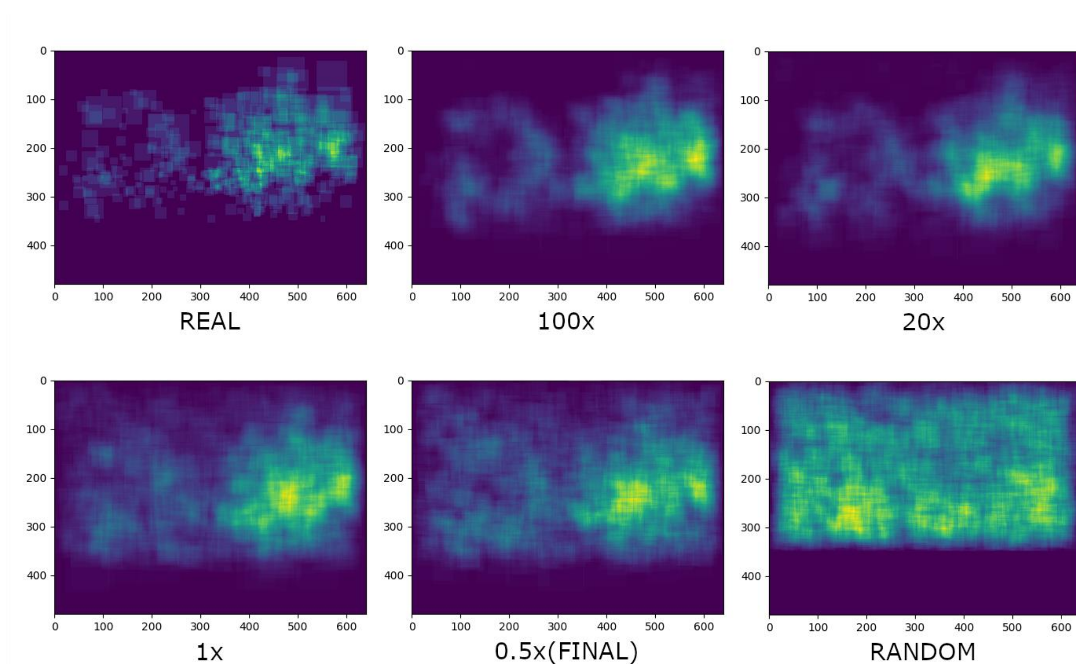


Graf 2: Podíl zdrojů obrázků pro generování

Aby syntetická data co nejlépe kopírovala reálná, vypočítal jsem analýzou reálných dat pravděpodobnost výskytu dopravních značek pro každý pixel obrázku, jinak řečeno jsem zjistil, kde se značky na českých silnicích, z pohledu řidiče, nacházejí nejčastěji. Poté jsem využil těchto dat k úpravě generačního algoritmu, tak aby pozice nově vložené značky měla větší pravděpodobnost nacházet se na místě s častějším výskytem. Vyzkoušel jsem několik konfigurací, kdy jsem zvyšoval a snižoval „násobek“ – kolikrát pravděpodobněji bude značka vložena na pixel s vyšším výskytem než na pixel s nižším. Testované konfigurace byly 100x, 20x, 5x, 1x a 0.5x – viz grafy níže. Protože jsem však chtěl zachovat jistou úroveň náhodnosti a nechtěl jsem značky generovat pouze na místech s nejvyšším výskytem, zvolil jsem nejnižší „násobek“, tedy 0.5. Příklad: na pixelu, kde se v reálných datech značka nacházela 10x, bude pravděpodobnost výskytu značky na tomto pixelu v syntetických datech 5x větší než na místě, kde byla nalezena značka pouze jednou. Dále také byla z generovaných pozic pro vložení značky kompletně vyjmuta oblast spodních 130 pixelů obrázku, neboť zde se nikdy značka v reálných

datech nenacházela a často tuto část obrázku zabírala palubní deska nebo kapota automobilu.

Při vkládání značek do obrázků program ještě automaticky upravil jas značky, aby zapadla na vybrané místo co nejlépe (př. značka ve stínu bude mnohem tmavší než značka na slunci).



Graf 3: Pravděpodobnost výskytu dopravních značek při různých „násobcích“

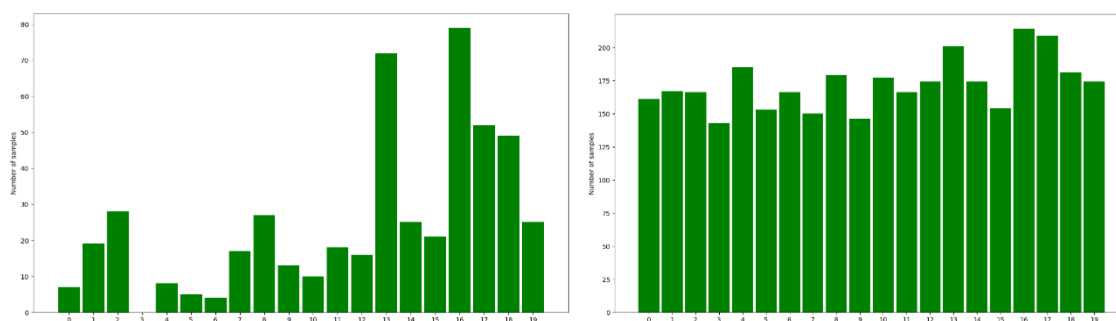
Obrázek 6: Příklady vygenerovaných dat

3.3 Výsledná datová sada

Nakonec jsem spojil reálná a syntetická data, abych získal jednu datovou sadu. Jako cíl jsem si dal dosáhnout poměru 1 reálný obrázek na 5 syntetických obrázků (1:5), ale kvůli vysoké časové náročnosti sběru reálných obrázků a vyznačování značek byl výsledný poměr okolo 1:6.

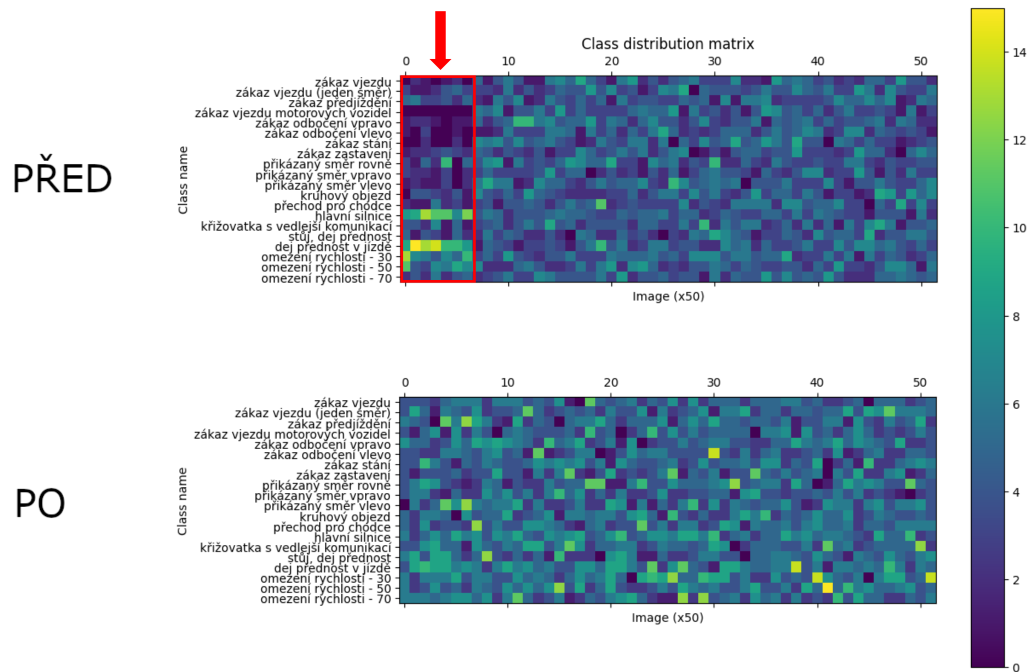
Celkem obsahuje výsledná datová sada 2 643 obrázků s 4 359 vyznačenými značkami, z čehož 355 obrázků je reálných a 2 288 synteticky generovaných.

Generování pomohlo nejen ve zvýšení počtu obrázků, ale také ve vyrovnání počtu výskytů jednotlivých značek. V reálné části datové sady byly některé značky příliš zastoupeny (například „dej přednost v jízdě“), jiné zas nebyly zastoupeny vůbec (pouze „zákaz vjezdu motorových vozidel“). Po generování bylo zastoupení značek v podstatě vyrovnané – rozdíl mezi počtem nejvíce a nejméně zastoupenými značkami byl zhruba 50, což při relativně velké velikosti datové sady není moc.



Graf 4: Rozložení tříd (značek) v datové sadě – reálná část datové sady (vlevo), výsledná datová sada (vpravo)

Než však mohla být nová datová sada prohlášena za hotovou, bylo ještě nutné jednotlivé obrázky zamíchat, protože jak jsem již výše zmínil, reálná část měla některé značky velmi slabě zastoupeny. Bez zamíchání by model viděl nejprve pouze reálné a potom pouze syntetické obrázky, což by mohlo vést ke snížení přesnosti. Cílem tohoto míchání je, aby měl model přístup ke všem zdrojům dat po celou dobu trénování. V následujícím obrázku je patrné, že datová sada před mícháním v prvních zhruba 350 obrázcích neobsahuje některé značky, v dalším obrázku (po zamíchání) už jsou značky rozložené rovnoměrně.



Graf 5: Porovnání rozložení tříd(značek) v datové sadě před a po zamíchání

Výsledná datová sada byla zveřejněna na platformě Github a je volně ke stažení: https://github.com/janstaffa/czech-traffic-signs.

3.4 Výběr modelu

Nejprve jsem k úkolu detekce a klasifikace dopravních značek chtěl využít dva odlišné modely – jeden pouze pro detekci a druhý pouze pro klasifikaci. Pro detekci jsem zvolil volně dostupný model YOLOv8 (12) a pro klasifikaci jsem vytvořil vlastní model pomocí frameworku Tensorflow (13) v jazyce Python. Systém měl fungovat tak, že detekční model najde místo, kde se nachází značka, program následně vyřízne danou oblast a předá nový obrázek jednoduššímu klasifikačnímu modelu, který určí, o jakou značku se jedná. Tento přístup jsem implementoval a přesto, že fungoval v podstatě podle očekávání, bylo evidentní, že výkon takového modelu je nedostatečný pro použití na mobilních zařízeních. Dokonce na relativně výkonném počítači dosahoval pouze zhruba 8 FPS, a to bez ohledu na lineárně se zvyšující časovou náročnost při vyšším počtu značek v obrázku (klasifikační model by musel běžet pro každou značku zvlášť).

Dále jsem zkusil naučit výše zmíněný model YOLOv8 detekovat i klasifikovat značky najednou (odstranil jsem mezikrok s detekcí). Tato konfigurace fungovala překvapivě

dobře a prokázala se jako velmi přesná a v porovnání s předchozím přístupem efektivnější – na počítači model dosahoval 10-13 FPS. Problém však opět nastal při spouštění na mobilních zařízeních, kde model byl stále příliš veliký a výkonem nedostatečný. Vlastně ani není divu, protože model není přizpůsoben k práci na mobilních procesorech a konverze na vhodný formát se ukázala jako velmi komplikovaná.

Výše zmíněné modely, přestože byly neúspěchem, mohou sloužit díky svým odlišným technologiím jako celkem dobré porovnání k výslednému modelu a ukázaly, že volba vhodné architektury modelu je naprosto zásadní a je třeba od začátku myslet na cílovou aplikaci modelu, v tomto případě nutnost provozovat model na mobilních zařízeních.

Pro detekci objektů na mobilních a podobných zařízeních (obecně nazýváno „edge devices“) existuje hned několik modelů, z nichž každý má mnoho variant. Mezi nejznámější, a ty které jsem se rozhodl využít, patří: SSD MobilenetV2, SSD EfficentDet a SSD MobilenetV2 FPNlite. Architektura těchto modelů je velmi složitá, proto ji zde nebudu popisovat, obecný popis konvolučních neuronových sítí je k nalezení v teoretické části práce.

Pro implementaci samotného modelu jsem využil jazyk Python a framework Tensorflow (13), který skrývá většinu implementačních detailů a nabízí velmi pěkně definovaný interface. Díky tomu je vývoj modelů mnohem efektivnější, než kdyby programátor psal model sám. Také nevyžaduje hluboké pochopení fungování detailů modelu a stačí pouze základní znalosti neuronových sítí.

Celkem jsem zvolil čtyři konfigurace pro trénování:

- SSD MobilenetV2 – vstup 640x480
- SSD MobilenetV2 – vstup 320x320
- SSD MobilenetV2 FPNlite – vstup 640x480
- SSD Efficentdet – vstup 512x512.

Vstup zde znamená rozlišení vstupního obrázku, tedy i velikost vstupní vrstvy neuronové sítě. Čím větší je rozlišení, tím přesnější jsou výsledky modelu, ale zpracování trvá déle. Naopak menší vstupní velikost znamená kratší dobu zpracování za cenu nižší přesnosti výsledků. Výstupem modelů jsou tzv. „bounding boxes“ – obdélníky, které určují, kde se v obrázku nachází detekované objekty (značky), skóre přiřazené každému detekovanému

objektu, které říká, s jakou pravděpodobností se jedná o konkrétní značku a samotná předpověď třídy objektu (konkrétní značky).

3.5 Trénování

Před trénováním je třeba připravit data, což kromě tvorby samotné datové sady obsahuje rozdělení na trénovací (data, na kterých se bude model učit), validační (data, podle kterých bude model upravovat své parametry) a testovací (data, která použije vývojář ke zpětnému manuálnímu testování modelu) části. Dobrý poměr pro toto rozdělení se obecně uvádí na 80 % trénovací a 20 % validační + pár testovacích obrázků, přičemž se žádný obrázek nesmí mezi částmi datové sady opakovat. Pokud by byly stejné obrázky v trénovací i validační části, naučil by se model tyto obrázky nazpaměť a jiné by už nedokázal rozpoznat. Já jsem data rozdělil na 2,100 trénovacích, 500 validačních a 43 testovacích obrázků.

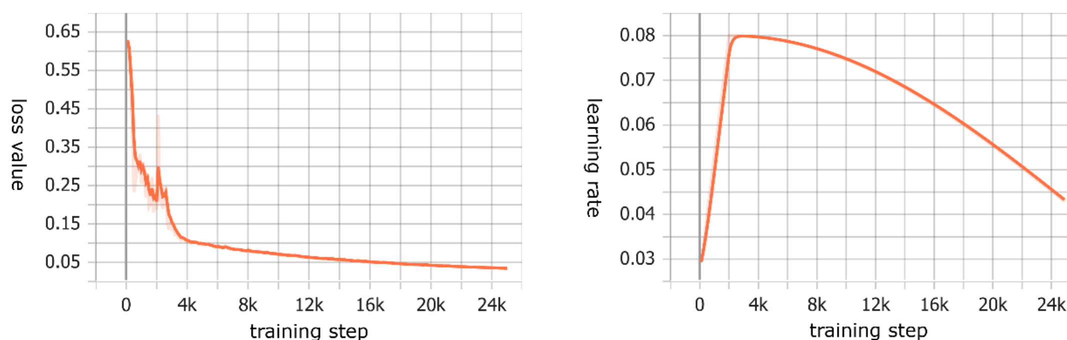
K samotnému trénování existují dva přístupy, buďto se model trénuje od nuly, kdy se musí naučit všechno, nebo se trénuje pomocí tzv. *transfer learning*, tedy v překladu „přenesené učení". Při využití této metody se model netrénuje od nuly, ale využije se již předem vytrénovaný model, který se poté pouze „dotrénuje" na konkrétních datech, v našem případě obrázcích dopravních značek. Výhodou tohoto přístupu je, že model již na začátku trénování dokáže rozpoznávat například jednoduché tvary, čáry nebo v některých případech třeba i znaky. Takto trénovaný model již nepotřebuje pro „dotrénování" tak velké množství dat, jako kdyby měl být trénován od nuly.

Při trénování jsem využil této metody a zvolil jsem předtrénovaný model trénovaný na datové sadě COCO (14). Tato datová sada obsahuje přes 200 tisíc obrázků s 1.5 milionem různých objektů rozdělených do 80 kategorií (př. člověk, auto, židle…). Model tak ještě před začátkem trénování na dopravních značkách dokáže detekovat všechny tyto objekty, mezi které dokonce patří jedna dopravní značka – „stopka".

Samotné trénování je velmi náročná operace, protože model musí mnohokrát projít všechny obrázky v trénovací datové sadě, která na běžném procesoru může trvat desítky hodin. Naštěstí je trénování optimalizováno pro práci na více vláknech, tedy je možné trénovat model na grafické kartě a proces mnohonásobně urychlit, problém však je, že framework Tensorflow a obecně většina podobného softwaru funguje pouze na grafických kartách Nvidia, ke kterým nemám přístup. Existuje však několik cloudových služeb, které nabízí zdarma, nebo velmi levně zapůjčení velmi výkonné grafické karty.

Mezi tyto cloudové služby patří Google Colab (15), který jsem využil. Google Colab nabízí zdarma grafickou Nvidia T4 GPU, která je pro účely trénování neuronových sítí přímo uzpůsobena, bezplatné běhové prostředí však může být kdykoli ukončeno, proto jsem využil relativně levné Colab Pro, které nabízí až 12 hodin nerušeného běhu a širší výběr grafických karet. Nakonec jsem modely trénoval na grafické kartě Nvidia V100, která je o něco rychlejší než T4. Celý proces probíhal bez větších problémů. Trénování každého modelu zabralo okolo 3-5 hodin. Pro srovnání na relativně výkonném moderním procesoru by stejné trénování mohlo trvat okolo 30 hodin.

Cílem samotného trénování je minimalizovat tzv. *loss function*, což je funkce, která se snaží určit, jak daleko je model v aktuální fázi učení od perfektního modelu. Čím je hodnota loss větší, tím více „chyb“ model dělá. Framework Tensorflow nabízí sadu metrik, které během učení ukládá, hlavní takovou metrikou je právě hodnota loss, ale také můžeme sledovat třeba *learning rate* („rychlost učení“). Toto číslo říká, jak moc může model upravit své parametry při každém kroku učení, čím je číslo větší, tím více aagresivněji mění model své parametry. Na začátku trénování je learning rate zvyšován, aby se model dokázal lépe adaptovat na nová data a naučil se jejich základní vlastnosti. Této fázi se říká *warmup phase*, neboli „zahřívací fáze“. Poté se v průběhu trénování *learning rate* snižuje podle nějaké funkce, v našem případě se jedná o nějakou exponenciální funkci. Snižování je zásadní, protože kdyby hodnota zůstala stále stejná mohl by model změnit parametry, které se již dříve naučil správně, a nezlepšoval by se dále. V podstatě čím je model lepší, tím dělá menší kroky v učení.

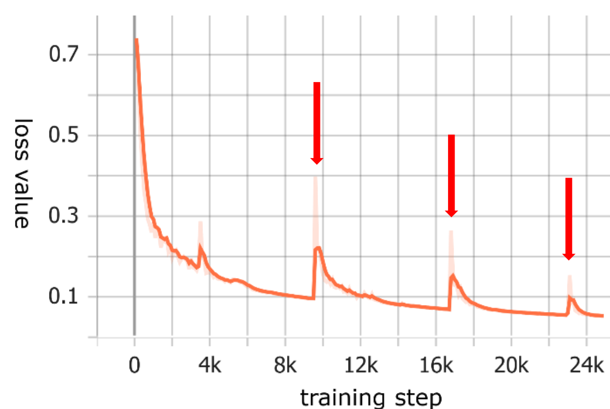


Graf 6: Loss funkce (vlevo) a learning rate (vpravo) pro model SSD MobilenetV2 320x320

Trénování může probíhat vlastně do nekonečna a je na vývojáři, aby zvolil správný moment, kdy ho ukončí. Je třeba dát pozor, aby se model nepřetrénoval na trénovacích

datech. Pokud by to nastalo, model by začal ztrácet schopnost pracovat s novými daty, která ještě neviděl, což nechceme. Naštěstí dobrým indikátorem pro přetrénování je zvyšující se hodnota loss. Během trénování model ukládá tzv. *checkpoints* („záchytné body“) každých cca 1 000 kroků a pokud dojde k přetrénování, může se výsledný model exportovat ze staršího checkpointu, kde ještě nebyl přetrénovaný. Nakonec jsem u většiny modelů zvolil jednotný bod, kdy ukončit trénování v hodnotě 25 000 trénovacích kroků (steps).

V následujícím obrázku je vidět graf loss funkce zakreslený při trénování modelu SSD MobilenetV2 FPNlite 640x480, na kterém můžeme pozorovat jasné skoky v hodnotách loss funkce. Po každém skoku se však hodnota vrátila a dále se opět snižovala, toto chování je očekávané a nejedná se o znak přetrénování.



Graf 7: Loss funkce modelu SSD MobilenetV2 FPNlite 640x480

3.6 Porovnání modelů

Po trénování jsem pro každý model vypočítal několik statistických metrik, které vypovídají o vlastnostech modelu. Základní metrikou pro porovnávání modelů detekující objekty je tzv. mAP (*mean average precisison*). Jedná se o jedno číslo, do kterého je zahrnuta jak přesnost modelu, tak schopnost rozpoznávat objekty.

Pro výpočet mAP potřebujeme nejprve vypočítat hodnoty *precission* a *recall*, neboli „přesnost“ a „odezvu“. Tyto hodnoty spočítáme následujícími výpočty, kde TP = *True Positives* – počet správně detekovaných značek, FP = *False Positives* – počet špatně detekovaných značek a FN = *False Negatives* – počet značek, které nebyly vůbec detekovány:

$$Precission = \frac{TP}{TP+FP} \qquad Recall = \frac{TP}{TP+FN}$$

Rovnice 1: Precission (vlevo), Recall (vpravo) (16)

Precission vypovídá o tom, jak dobře model rozpoznává konkrétní značky, a *recall* o tom, zda model detekuje všechny značky, které má. Hodnoty obou čísel se pohybují v rozmezí od 0 do 1, přičemž perfektní model má hodnotu 1 a netrénovaný model se blíží k 0, tedy vyšší je lepší.

Než můžeme tyto rovnice vypočítat, musíme najít způsob, jak určit, zda se jedná o správně detekovaný objekt. Běžně používanou metodou pro tento výpočet je tzv. IOU (*Intersection Over Union*), jedná se o číslo, které říká jak moc se dva obdélníky v dvou-dimenzionálním prostoru překrývají, protože vlastně neděláme nic jiného než porovnání dvou obdélníků (jeden předpovězený naším modelem a druhý je skutečnou pozicí značky definovanou v testovací datové sadě). Rovnice pro výpočet IOU, kde A a B jsou porovnávané obdélníky vypadá následovně:

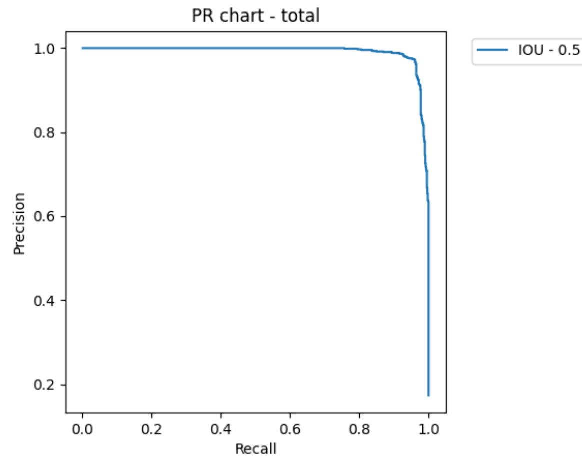
$$IOU = \frac{A \cap B}{A \cup B}$$

Rovnice 2: Intersection Over Union (17)

V podstatě dělíme plochu, kde se obdélníky překrývají, s celkovou plochou obou obdélníků, jinak řečeno pomocí výroků: konjunkce dělená disjunkcí. Výsledkem je číslo od 0 do 1, přičemž 1 znamená, že jsou obdélníky totožné, a 0 znamená, že nemají žádné překrytí.

Nyní můžeme určit hranici IOU, při které budeme detekci považovat za správnou, běžně se používá hodnota 0.5, ale může se použít i postupně se zvyšující sada hodnot, která vypovídá o fungování modelu ještě přesněji.

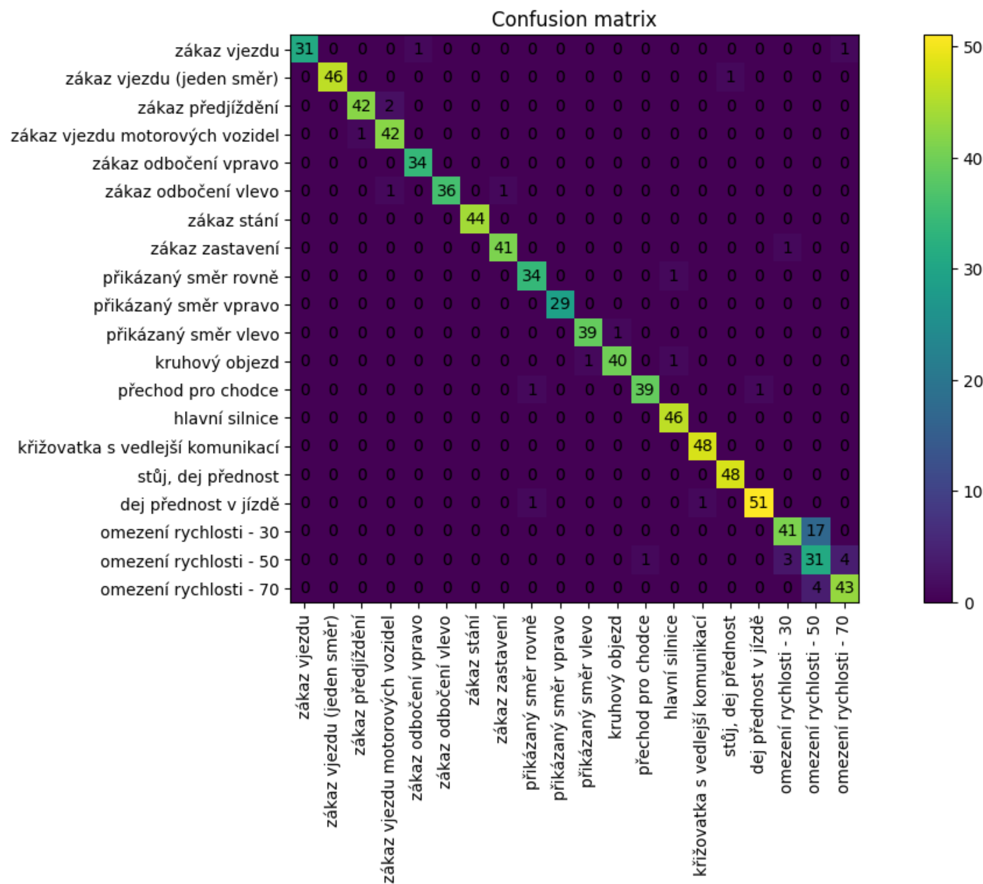
Pokud seřadíme detekované objekty podle jejich skóre a následně postupně vypočítáme hodnoty pro precission a recall, získáme tzv. *Precission Recall graf* (PR graf). Pokud poté integrujeme plochu pod tímto grafem, získáme hodnotu mAP pro daný model při dané IOU hranici. Jedná se opět o číslo od 0 do 1, kde vyšší je lepší. Integrace se dá snadno provést interpolací bodů, zachovávajících pravé úhly na křivce, ze kterých se dá získat série obdélníků, jejichž obsah se dá následně sečíst.



Graf 8: Příklad PR grafu pro model SSD MobilenetV2 640x480 při hranici IOU 0.5

Pro každý testovaný model jsem takto vypočítal hodnotu mAP při hranici IOU 0.5 a sadě hranic od 0.5 do 0.95 zvyšující se po 0.05 (značeno IOU=0.5:0.95:0.05) na testovací datové sadě obsahující 500 obrázků s 848 značkami. Tabulka výsledků k dispozici v sekci výsledky.

Další velmi praktickou metrikou je tzv. *confusion matrix*. Jedná se o matici, kde jsou zobrazeny všechny kombinace tříd (značek) a v každém poli je zapsáno kolikrát došlo k detekci značky podle této kombinace (kolikrát byla detekována značka A jako značka B). Tento graf nám říká, které značky si model nejvíce plete a které naopak rozpoznává bez problémů. Znamením zdravého modelu je „výrazná diagonála", která říká, že model dokáže ve většině případech určit značku správně.



Graf 9: Příklad Confusion matrixu vypočítaného pro model SSD MobilenetV2 320x320 s IOU hranicí 0.5

Z výše uvedeného grafu je patrné, že tento model bude mít problém rozeznat od sebe jednotlivé značky rychlostí, což se díky jejich podobnosti dalo očekávat. Jinak je patrná „výrazná diagonála" a s ostatními značkami nemá problém. Porovnání *confusion matrixů* všech modelů k dispozici v sekci výsledky.

3.7 Optimalizace

Různé procesory jsou optimalizované pro různé operace a my můžeme náš model upravit podle toho, na jakém zařízení budeme výsledný model spouštět. Cílem tohoto projektu je spouštět model na mobilních zařízeních a ty, obecně řečeno, díky své omezené výpočetní kapacitě rychleji počítají s celými čísly než s desetinnými. Dále budu používat pojmy *integer* a *float*. U procesorů nalezených v počítačích a noteboocích je to pak přesně naopak a model optimalizovaný pro operace s *floaty* je mnohem rychlejší. Porovnání ve výsledcích.

Využil jsem operaci quantizace, která převede 4 bytové *floaty* na 1 bytové *integery*, jenže tím sníží počet možností každého čísla více než 16 milionkrát. Tuto ztrátu kompenzuje zaokrouhlením na nejbližší číslo, které je násobkem nového 1 bytového čísla a násobku, který je dynamicky vybrán analýzou parametrů modelu a testováním na reprezentativní datové sadě, která má simulovat reálné využití. Naštěstí tento proces dokáže provést Tensorflow, a vývojář tak musí specifikovat jen několik nastavení.

Impakt quantizace na model je znatelný, velikost modelu byla snížena zhruba 3.5x – není to přesně 4x, protože některé hodnoty musí zůstat stejné mezi quantizovaným a normálním modelem a nejdou quantizovat (například metadata nesoucí informace o modelu). Hlavním přínosem quantizace však bylo výrazné zvýšení výkonu na mobilních zařízeních, kde došlo ke 35% zvýšení rychlosti detekce oproti nequantizovanému modelu. Ze střídmého testování jsem dospěl k závěru, že quantizace nemá znatelný negativní impakt na kvalitu detekce. Některé zdroje uvádí snížení přesnosti modelu po quantizaci o zhruba 5 %, což je zanedbatelné.

Před spuštění na mobilním zařízení bylo ještě nutné model převést do formátu Tensorflow Lite, který je navržen pro běh na *edge devices* (i telefonech). Tato konverze dále optimalizuje model. Také jsem tento model obohatil o metadata – informace určené jak pro počítače tak pro vývojáře, která specifikují tvar vstupu a výstupu modelu, popisují co model vlastně dělá, informují o nutných operacích na vstupních datech před předáním modelu, a další...

3.8 Tvorba mobilní aplikace

Nejprve jsem se pokusil vytvořit aplikaci v jazyce Python, ve kterém je psán kód pro trénování modelu, ale rychle jsem narazil na limitace tohoto přístupu. Proto jsem se rozhodl vytvořit aplikaci nativně pro platformu Android. Jazykem této platformy je Kotlin, který má relativně standardní syntax, takže nedělal problémy. Samotný vývoj probíhal v programu Android studio, se kterým jsem byl velmi spokojen, uživatelské rozhraní bylo intuitivní a mnoho operací šlo dělat graficky. Další výhodou tohoto prostředí je dobře provedená a uživatelsky snadná integrace modelu Tensorflow Lite do aplikace.

Při vývoji jsem adaptoval volně dostupný sample program od Tensorflow a rozšířil ho o funkce potřebné k tomuto projektu.

Aplikace funguje bez problémů, jediným zádrhelem, se kterým se uživatel může setkat, je hlasové hlášení dopravních značek, které je sice česky, ale hlas je evidentně anglický, proto jsou některá hlášení trochu rozbitá a vtipná.

4 - Výsledky

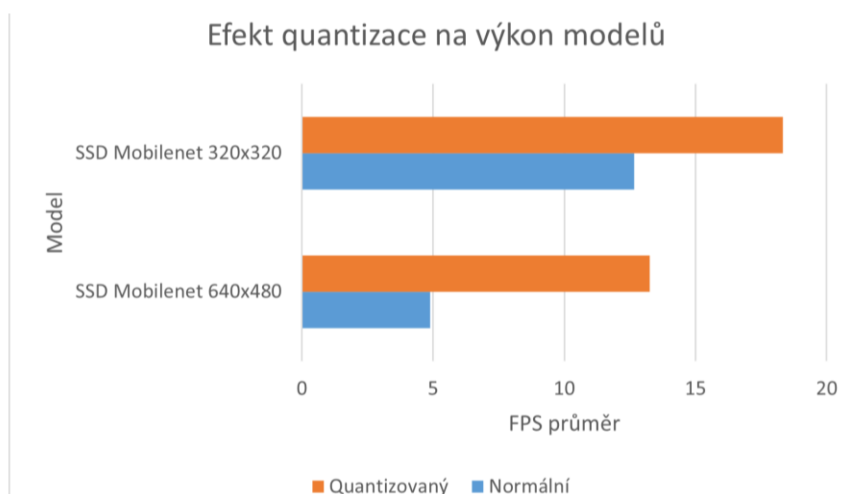
Celkem jsem vytrénoval čtyři modely, po testování jsem dospěl k závěru, že modely Efficentdet a MobilenetV2 FPNlite jsou, kvůli pomalému běhu a nižší přesnosti, pro účely práce nevhodné, proto jsem se soustředil na modely MobilenetV2 640x480 a MobilenetV2 320x320 (přesto přikládám porovnání PR grafů a confusion matrixů všech sledovaných modelů níže). Tyto modely jsem následně optimalizoval pro mobilní zařízení pomocí quantizace.

Z vybraných modelů má každý své výhody a každý vyniká v něčem trochu jiném. Model MobilenetV2 640x480 je rozhodně přesnější než verze se vstupem 320x320, což reflektuje i pokles hodnoty mAP z 0.807 u 640x480 na 0.663 u 320x320 (o 17.8 %). Naopak průměrná rychlost detekce se zvýšila z 18FPS u 640x480 na 52FPS u 320x320 (o 288 % - měřeno na procesoru Intel i5-11320H).

Model	IOU=0.5	IOU=0.5:0.95:0.05
SSD MobilenetV2 640x480	0.952	0.807
SSD MobilenetV2 320x320	0.899	0.663
SSD MobilenetV2 FPNlite 640x480	0.921	0.804
SSD Efficentdet 512x512	0.883	0.505

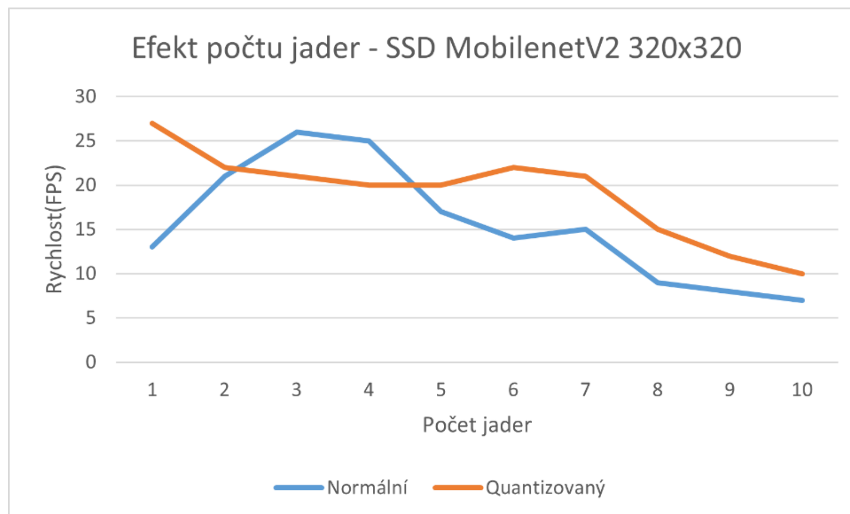
Tabulka 1: Hodnoty IOU sledovaných modelů

Quantizace pak přinesla průměrně 23% zvýšení rychlosti modelu, přičemž vliv quantizace záležel na počtu jader. Obecně počet jader velmi ovlivňoval výkon modelu. Překvapivě dokonce v některých případech překonal normální model ten quantizovaný.



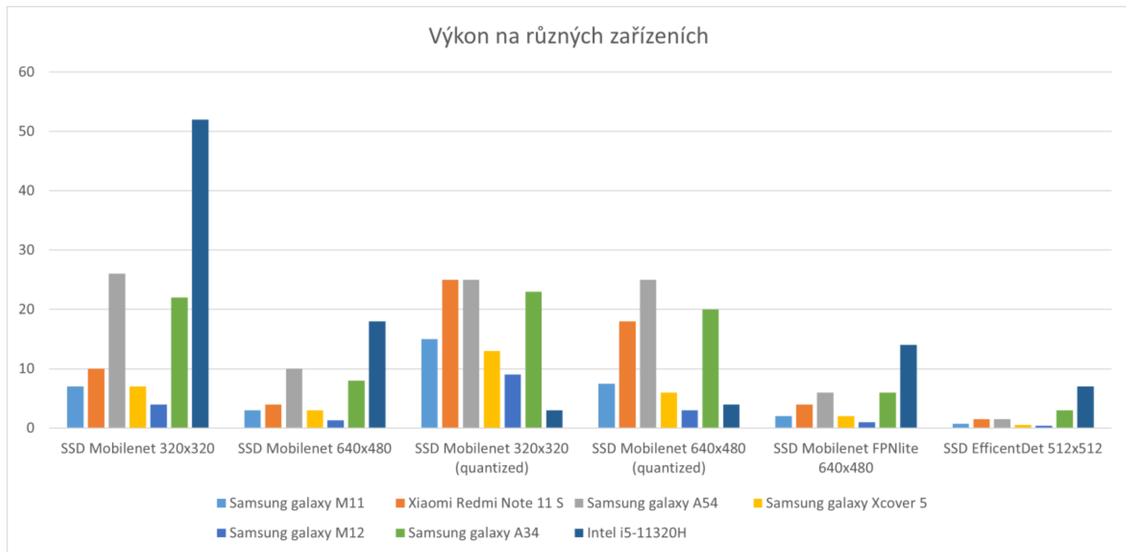
Graf 10: Efekt quantizace na výkon modelů na mobilních zařízeních

Také bylo celkem překvapivé, že vyšší počet jader nemusí znamenat vyšší výkon, v praxi tomu dokonce bylo přesně naopak. Každý model měl maximum výkonu na jiném místě na škále počtu jader a další zvyšování počtu jader už výkon jen snižovalo.

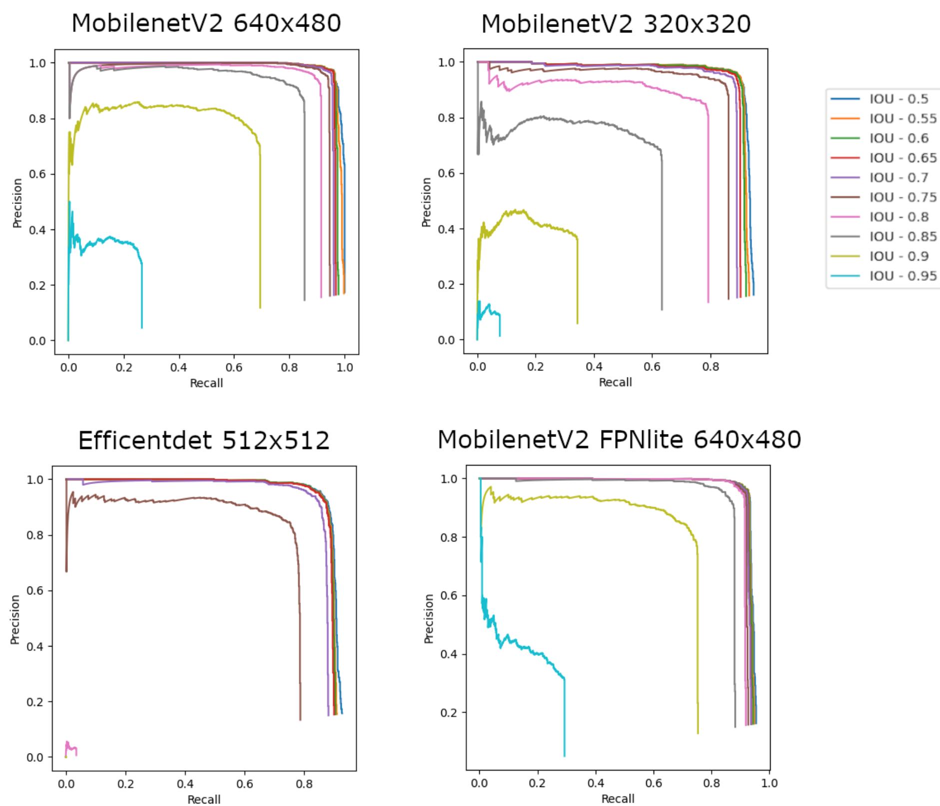


Graf 101: Efekt počtu jader na výkon modelu SSD MobilenetV2 320x320

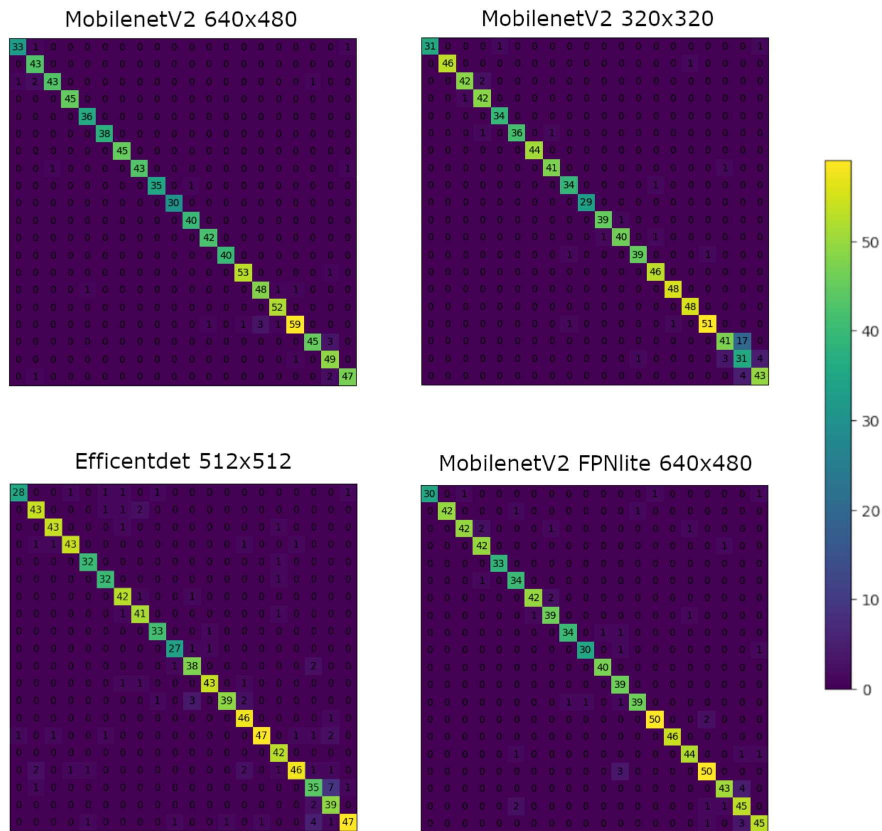
Dalším zajímavostí bylo, že při testování na mobilních zařízeních se rychlost modelu znatelně snížila, když byl vstup prázdný (černá obrazovka). Rychlost zde konzistentně klesala na 10FPS, což v některých případech je až 50% pokles výkonu. Dá se soudit, že se jedná o záměrnou funkci Tensorflow Lite, která se snaží snížit zátěž, když se ve snímku evidentně nenachází žádné objekty k detekci.



Graf 11: Porovnání výkonu modelů na různých zařízeních



Graf 12: Porovnání Precision Recall grafů sledovaných modelů při různých IOU hranicích



Graf 13: Porovnání confusion matrixů sledovaných modelů při IOU hranici 0.5

5 - Závěr

Cílem práce bylo vytvořit mobilní aplikaci, která dokáže detekovat dopravní značky v reálném čase, tato aplikace v průběhu práce vznikla a funguje dle očekávání. Celkem jsem pracoval se čtyřmi modely, nejlepších výsledků dosáhl model MobilenetV2 640x480, jehož úspěšnost je 80.7% mAP. Nejrychlejší pak byl model MobilenetV2 320x320, jehož průměrná rychlost na mobilních zařízeních po quantizaci je 18.3FPS. Vyzkoušel jsem různé optimalizace, které dále zlepšily výkon modelů. Také v průběhu práce vznikla datová sada kombinací reálných dat z několika zdrojů, mých vlastních dat a syntetických dat, pro která jsem vytvořil relativně pokročilý generátor. Hotovou aplikaci jsem následně testoval na různých zařízeních za různých podmínek a zjišťoval jsem, které jevy (ať už hardwarové, nebo softwarové) a jak ovlivňují přesnost a rychlost modelu.

Při práci jsem čerpal z mnoha zdrojů, zvláště bych uvedl Youtube kanál EdjeElectronics (18), Nicholas Renotte (19) a tutorialsEU (20), také mě inspirovala bakalářská práce Filipa Kočici z VUT v Brně (3).

Práce by šla dále rozšířit zejména přidáním pokročilejších funkcí do aplikace, jako je ukládání aktuální maximální rychlosti, nastavení závažnosti jednotlivých značek nebo přidáním českého překladu do hlášení značek. Také by rozhodně stálo za to rozšířit počet sledovaných značek a přidat třeba schopnost číst některé dodatkové tabulky. Dalším prostorem pro zlepšení je datová sada, kde by se mohl zlepšit poměr reálných a syntetických obrázků třeba na 1:1, nebo by mohla být syntetická data kompletně nahrazena reálnými.

Myslím, že práce nabízí nové zajímavé pohledy na problematiku detekce dopravních značek, a věřím, že na výsledky mé práce by se v budoucnu dalo navázat dalším výzkumem v této oblasti.

Zdroje

1. **Přibyl, Martin.** Aktuálně. [Online] https://zpravy.aktualne.cz/ekonomika/auto/nadavate-na-ne-ale-vite-jak-funguji-prehled-funkci-automobil/r~b8fa89c6735211e9819e0cc47ab5f122/v~sl:43fa2a4b70f36b260d20954f986334bc/.

2. **Wikipedia. [Online] https://en.wikipedia.org/wiki/Traffic-sign_recognition.**

3. Kočica, Filip. VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ. ***vut.cz.*** [Online] 2019. https://www.vut.cz/www_base/zav_prace_soubor_verejne.php?file_id=198183.

4. Dopravní-značení.eu. [Online] http://www.dopravni-znaceni.eu/.

5. Wikipedia. [Online] https://en.wikipedia.org/wiki/Neural_network_(machine_learning).

6. [Online] http://neuralnetworksanddeeplearning.com/chap1.html.

7. Convolutional neural network. ***Wikipedia.*** [Online] https://en.wikipedia.org/wiki/Convolutional_neural_network.

8. [Online] https://towardsai.net/p/deep-learning/convolutional-neural-networks-for-dummies.

9. team, GTSDB. [Online] 2011. https://www.kaggle.com/datasets/safabouguezzi/german-traffic-sign-detection-benchmark-gtsdb.

10. HumanSignal. LabelImg. [Online] https://github.com/HumanSignal/labelImg.

11. Timofte, Radu. BelgiumTS. [Online] https://btsd.ethz.ch/shareddata/index.html.

12. Ultralytics. YOLOv8. [Online] https://github.com/ultralytics/ultralytics.

13. Tensorflow. Tensorflow. [Online] https://www.tensorflow.org/.

14. COCO. COCO. [Online] https://cocodataset.org/.

15. Google. Colab. [Online] https://colab.google/.

16. Wikipedia. Precision and recall. [Online] https://en.wikipedia.org/wiki/Precision_and_recall.

17. Rosebrock, Adrian. Intersection over Union (IoU) for object detection. [Online] https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/.

18. Electronics, Edje. Youtube. [Online] https://www.youtube.com/@EdjeElectronics.

19. Renotte, Nicholas. Youtube. [Online] https://www.youtube.com/@NicholasRenotte.

20. tutorialsEU. Youtube. [Online] https://www.youtube.com/@tutorialsEU.

21. Tensorflow. Tensorflow. [Online] https://www.tensorflow.org/.

22. Tenorflow. Post Integer Quantization. [Online] https://www.tensorflow.org/lite/performance/post_training_integer_quant.

23. Evan Juras, EJ Technology Consultants. TensorFlow Lite Object Detection API in Colab. [Online] https://colab.research.google.com/github/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi/blob/master/Train_TFLite2_Object_Detction_Model.ipynb#scrollTo=fF8ysCfYKgTP.

24. —. How to Train TensorFlow Lite Object Detection Models Using Google Colab. [Online] https://www.youtube.com/watch?v=XZ7FYAMCc4M.

25. Cherukuri, Ram. MathWorks. ***What Is int8 Quantization and Why Is It Popular for Deep Neural Networks? .*** [Online] https://www.mathworks.com/company/technical-articles/what-is-int8-quantization-and-why-is-it-popular-for-deep-neural-networks.html.

26. tutorialsEU. Youtube. ***Kotlin & Android 12 Tutorial.*** [Online] https://www.youtube.com/watch?v=HwoxgUPabMk.

27. Renotte, Nicholas. Youtube. ***Tensorflow Object Detection in 5 Hours with Python.*** [Online] https://www.youtube.com/watch?v=yqkISICHH-U.