Univerzita Hradec Králové

Fakulta informatiky a managementu

BAKALÁŘSKÁ PRÁCE

2021 **Lukáš Bařtipán**

Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informačních technologií

Strojové učení ve vývojovém prostředí Flutter

Bakalářská práce

Autor: Lukáš Bařtipán

Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Karel Mls, Ph.D.

Hradec Králové květen 2021

Prohlášení:

Prohlašuji, že jsem bakalářskou/diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

podpis

V Hradci Králové dne

Poděkování:

Děkuji vedoucímu bakalářské práce Ing. Karlu Mlsovi, Ph.D. za metodické vedení práce, rady a připomínky.

Anotace

Bakalářská práce se zaměřuje na strojové učení pomocí knihovny TensorFlow ve vývojovém prostředí Flutter. Teoretická část je zaměřena na principy umělých neuronových sítí, popis jejich fungování a vysvětlení možností využití. Praktická část se zabývá přípravou dat pro učení neuronové sítě a následný proces učení. Dále je zde popsána implementace mobilní aplikace, která využívá naučené neuronové sítě pro rozpoznání objektu na obrázku.

Annotation

Title: Machine learning in Flutter framework

Bachelor's thesis focuses on machine learning using the TensorFlow library in the Flutter development environment. The theoretical part is focused on the principles of artificial neural networks, a description of their operation and an explanation of the possibilities of use. The practical part deals with the preparation of data for learning a neural network and the subsequent learning process. Furthermore, the implementation of a mobile application is described here, which enables learned neural networks for object recognition in the figure.

Obsah

Seznam obrázků

Seznam použitých zkratek

API – Application Programming Interface

MLP – Multi-layer perceptron

YOLO – You Only Look Once

R-CNN – Region Based Convolutional Neural Networks

NPM – Node Package Manager

1. Úvod

V dnešní době už existuje více způsobů implementace neuronových sítí a jsou nedílnou součásti informatiky v 21. století. Tato práce je zaměřena na využití knihovny TensorFlow a její části Lite pro využití na mobilních zařízeních jako je mobilní telefon, Arduino, Raspery Pi, apod.

Cílem této práce je tedy naučení neuronové sítě a následné použití v mobilní aplikaci, která využívá framework Flutter. Framework Flutter slouží pro multiplatformní vývoj aplikací, nicméně není jediným řešením, co se multiplatformního vývoje mobilních aplikací týče.

Teoretická část nejdříve popisuje principy neuronových sítí a knihovny TensorFlow, její části a formáty dat určené pro učení. Dále se zaměřuje na framework Flutter a jeho součásti. Dále je popsán programovací jazyk Dart, který je hlavním jazykem frameworku Flutter.

V praktické části je popsán postup přípravy dat pro učení neuronové sítě. Přípravu pro přenesené učení již existujícího natrénovaného modelu neuronové sítě. A následné převedení natrénované modelu do formátu tflite, který je určen pro mobilní zařízení. Závěr pak shrnuje důležité poznatky plynoucí z této práce.

2. Cíle práce

Cílem bakalářské práce je vytvoření mobilní aplikace, která využívá neuronovou síť k rozpoznání obrazu, a to za pomocí knihovny TensorFlow a frameworku Flutter. K použití natrénovaného modelu na mobilním zařízení je zde využita část knihovny TensorFlow s názvem Lite.

3. Teoretická část

3.1 Umělé neuronové sítě

3.1.1 Historie neuronových sítí

Neuronové sítě existují od 40. let 20. století a mají dlouhou historii. V průběhu historie se neuronové sítě mnohokrát vynořily z popela diskreditace. W. McCulloch a W. Pitts (1943) jako první navrhli koncept neuronové sítě a neuronu (neuron je základní jednotkou každého nervového systému). Nebyla možnost, jak trénovat tyto umělé neuronové sítě. Váhové matice, které neurony obsahují, musely být u těchto raných neuronových sítí vytvořeny ručně programátory, což zabraňovalo efektivnímu učení a využití neuronové sítě. Neuronové sítě se tím dostaly do nemilosti kvůli časově náročné povaze jejich přípravy [1].

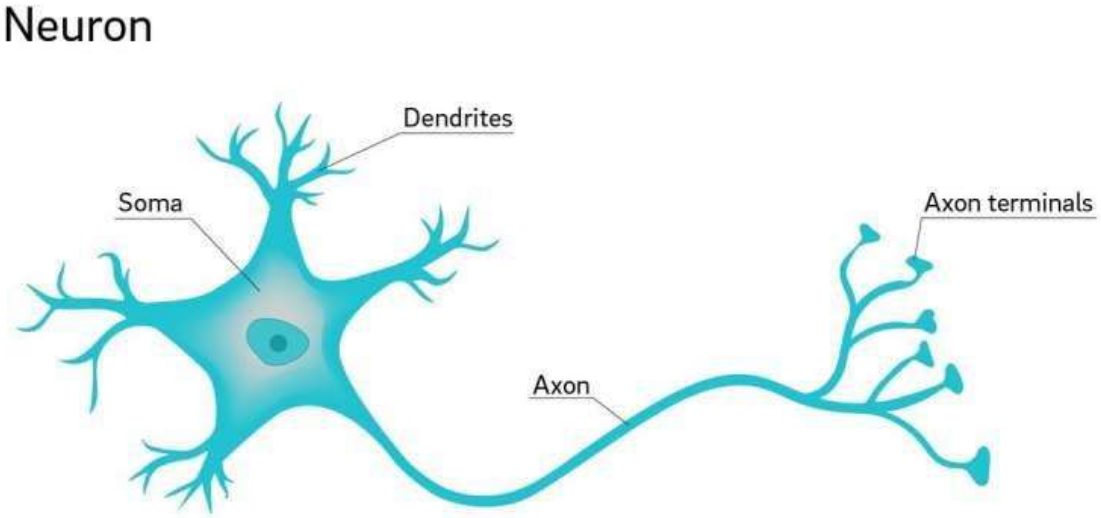
F. Rosenblatt v roce 1958 představil tréninkový algoritmus nazvaný „backpropagation“ (z angličtiny – zpětná propagace), který automaticky generuje váhové matice neuronových sítí jejichž součástí je perceptron. Perceptron může ve skutečnosti obsahovat i několik vrstev neuronů, které napodobují architekturu zvířecích mozků, nicméně je pomalý a s narůstajícím počtem vrstev se stává ještě pomalejší. Přidání výpočetní síly v 80. letech a na počátku 90. let pomohlo neuronovým sítím plnit úkoly, ačkoliv hardwarové a tréninkové algoritmy té doby nebyly schopny efektivně trénovat neuronové sítě s několika vrstvami a tím se dostaly do nemilosti podruhé [1].

V roce 2006 G. E. Hinton představil nový revoluční způsob trénování hlubokých neuronových sítí, jehož výsledkem je třetí růst neuronových sítí [1]. Hluboké neuronové sítě a hluboké učení pokročily natolik, že nyní mohou obdivuhodně fungovat v celé řadě důležitých problémů v oblasti počítačového vidění, rozpoznávání řeči a zpracování přirozeného jazyka. Společnosti jako Google, Microsoft a Facebook je využívají v širokém měřítku [3].

3.1.2 Popis neuronové sítě

Základní složkou neuronové sítě je neuron, podobně jako u nervové soustavy člověka. V neuronových sítích je tento uměle vytvořený neuron nazýván perceptron. S tímto konceptem sítě přišel F. Rosenblatt, který se inspiroval prací W. McCullocha a W. Pittse [2]. Jedná se o prvek neuronové sítě, který sám o sobě nemá žádnou inteligentní funkci. Neuron ze svých vstupů (dendritů) přijme signál a vyhodnotí jejich počet a sílu. Pokud signál přesáhne vnitřní práh neuronu, odešle svůj výstup (axon) do terminálu, který je napojen na další neurony [4].

Samotný neuron sám o sobě netvoří neuronovou síť, slouží pouze jako hradlo. Neuronovou síť tedy tvoří spojení množství neuronů, uspořádání do sítě a upravování síly signálů každého neuronu. Tento proces se realizuje pomocí učení neuronové sítě [2].



Obrázek 1: Neuron a jeho popis

zdroj: Medical Xpress [online] (2018)

Princip neuronové sítě je tedy založen na principu lidského mozku, ale simulovat lidský mozek je aktuálně výpočetně nemožné. Využíváno je tedy jen malé neuronové sítě k specifikovaným úkonům, což dovoluje využívat neuronové sítě na běžném PC nebo na chytrém mobilním telefonu [4].

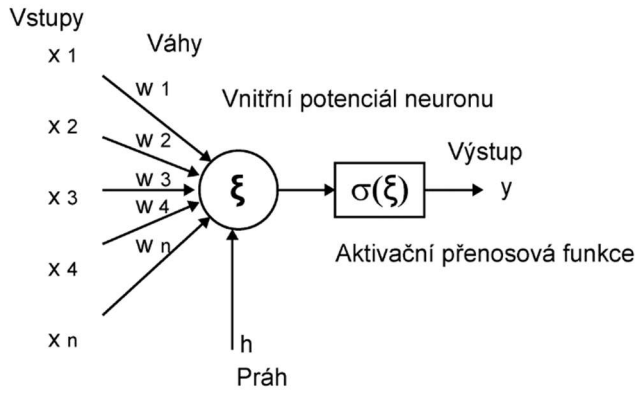
3.1.3 Využití neuronové sítě

Využití těchto neuronových sítí má široké pole uplatnění. Mezi ty hlavní se dají dle [1] zařadit:

- **předpověď** budoucího vývoje na základě naučených dat (např. vývoj akcií, počasí)
- **rozpoznávání** obrazců, textů, identifikace objektů, otisky prstů
- **řízení procesů** v prostředí, které se dynamicky mění (např. autopilot)
- **optimalizační úlohy**

3.1.4 Matematický model neuronu

Neuron je možno rozdělit na několik částí, což jsou synapse $\acute{\omega}_1, \acute{\omega}_2, \ldots, \acute{\omega}_n$ (váhy), vstupy $x_1, x_2, \ldots, x_n$, konstantní hodnota prahu h, vnitřní potenciál neuronu ξ, aktivační přenosová funkce σ, výstup y [1].



Obrázek 2: Model neuronu popsaný McCullochem a Pittsem

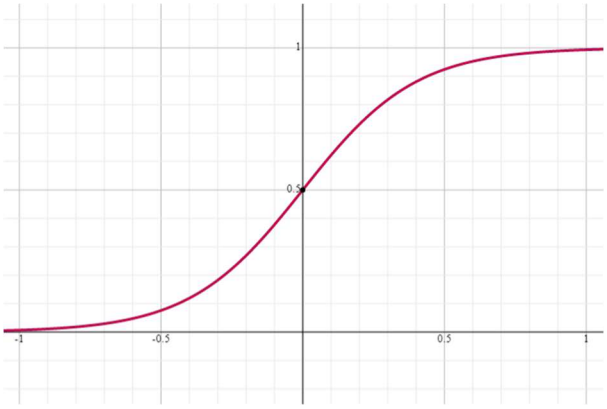
zdroj: Matematická biologie – učebnice [online] (2021)

Neuronový vstup je vektorem prvků, kterým jsou reálná čísla. Podobně může být prahová hodnota reálným číslem. Vnitřní potenciál neuronu je hodnota, která vyplývá z porovnání vážených a sčítaných vstupů s prahem. Váhy jsou opět skutečná čísla, takže mohou být kladná i záporná. Nejjednodušší přechodová funkce je funkce signum, která nám zajišťuje, že neuron je binární, tzn. výstupem mohou být pouze hodnoty -1, nebo 1 [1].

$$\sum_{i=0}^{n}(w_i x_i) - \Theta = x; y = sgn(x)$$

Vyhodnocování součtu může využívat i jiných funkcí, jejichž definiční obor je spojitý, jako příklad se dá použít funkce sigmoidálního přenosu [2].

$$f: y = \frac{1}{1 + e^{-kx}}$$



Obrázek 3: Graf sigmoidální funkce

zdroj: vlastní zpracování, dle Symbolab [online]

Výstupy takovýchto neuronů mají spojitý definiční obor, což je výhodou, protože existuje derivace v každém bodě funkce. Tuto derivaci následně využívá velké množství algoritmů určeních pro učení. Neurony se spojitou funkcí se nazývají spojité neurony [1].

3.1.5 Architektura neuronových sítí

Neuronové sítě jsou tvořeny pomocí vrstev souvisejících neuronů. Většina z nich má alespoň dvě vrstvy: vstupní a výstupní. Neuronová síť přijme vstup do tzv. vstupní vrstvy a vrátí výstup pomocí výstupní vrstvy. Část neuronové sítě mezi vstupem a výstupem je nazývána „black box“, neboli černá skříňka. Jedná se o skryté vrstvy neuronové sítě, mezi kterými probíhá zpracování vstupu. Název černá skříňka vznikl podle toho, že nemáme kontrolu nad tím, co se ve skrytých vrstvách děje [1]. Neuronové sítě mohou využívat různé architektury, a to konkrétně lineární neuronové sítě (LNN), vícevrstvé neuronové sítě (MLP) a neuronové sítě založené na radiální bázové funkci (RBF) [5].

3.1.5.1 Perceptron

Perceptronové sítě jsou jedny z nejstarších neuronových sítí. Jsou zároveň také jedny z nejvíce používaných neuronových sítí. Skládají se z jednoduchého binárního neuronu, který má výstupní skokovou funkci. Je zde možnost i vícevrstvého perceptronu, což je skupina těchto jednoduchých perceptronů. Tato struktura vychází z původní struktury základního modelu neuronové sítě, kterou vytvořili W. McCulloch a W. Pitts [5].

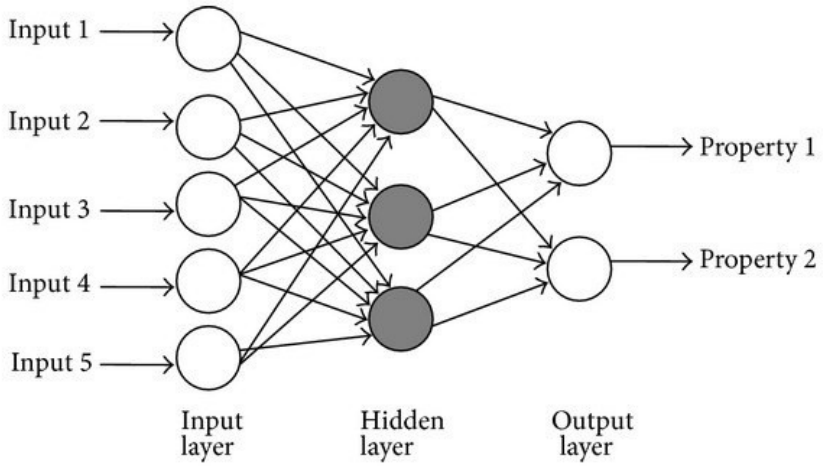
Perceptron se využívá při vytváření jednoduchých logických funkcí. Kde nás omezuje lineární rozdělení množiny, jelikož tento perceptron může nabývat jen binárních hodnot. Proto se tato neuronová síť není schopna naučit vše [5].

Hlavní nevýhodou jednoduchých perceptronů je neschopnost využití funkce XOR, což se stalo velkým problémem na několik let a vývoj neuronových sítí se tím zpomalil. Poté se zjistilo, že pomocí skupiny jednoduchých percetronů, se tento problém vyřeší [5].

3.1.5.2 Vícevrstvé sítě

U architektury vícevrstvé sítě jsou neurony organizovány ve vrstvách. První vrstva představuje vstup a poslední vrstva výstup. Vrstvy mezi nimi se nazývají skryté a může jich být libovolný počet. Vstupní vrstva je pasivní a představuje jen hodnoty vstupu. Zpracování vstupních hodnot se provádí ve skrytých vrstvách. Mezi nejvíce využívané architektury se řadí architektura „Multi-Layer Feedforward“ (MLF). Jedná se

o architekturu s dopředným šířením informací. Vrstvy jsou propojeny pouze jedním směrem od vstupních vrstev po výstupní [5].



Obrázek 4: Vícevrstvá neuronová síť

zdroj: Medium.com [online] (2016)

Při návrhu neuronové sítě nám bohužel nepomůže žádný jednoznačný algoritmus, nicméně existuje mnoho doporučení, jak při návrhu vícevrstvé neuronové sítě postupovat. I tak je při návrhu nutno vycházet z nabytých zkušeností a experimentování. Pro výpočet počtu neuronů v neuronové síti je možno využít těchto vzorců (tyto vzorce neplatí současně):

$$p = \sqrt{n\,m}$$

$$p \geq n + m$$

kde **n** je počet vstupních a **m** počet výstupních neuronů [2].

3.1.6 Učení neuronových sítí

Zjednodušeně lze neuronovou síť popsat jako jednoduchý stroj, který vezme otázku a vymyslí na ni odpověď. Podobně jako lidé přijímající podnět očima, využívající mozek k analýze scény a určení závěru, jaké objekty jsou ve scéně, na kterou se dívají [4].

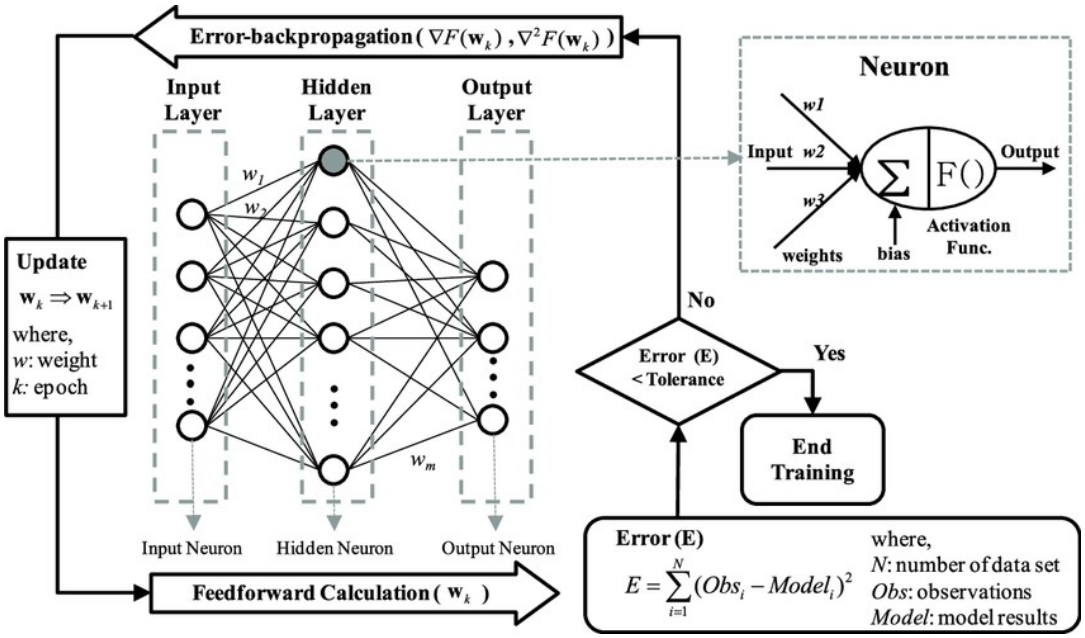
Učení neuronové sítě, lze rozdělit do dvou základních kategorií. Jedná se o učení s dohledem (supervised) a učení bez dohledu (unsupervised). Oba způsoby učení neuronové sítě jsou iterativní procesy. S každou iterací se mění matice vah v neuronové síti, což vede ke snížení šance chybného výstupu („error rate“) na co nejnižší možnou hodnotu, čímž se zvětšuje šance na úspěch [2] .

Pokud je neuronová síť učena s dohledem je zapotřebí mít u vstupních, v tomto případě tréninkových dat, i výstupní hodnotu, která je očekávána [6].

Při učení neuronové sítě bez dohledu (unsupervised) je výstup odhadován. S dostatkem dat je možné najít určité souvislosti, analyzovat jejich strukturu a na základě této analýzy udělat odhad správné hodnoty [6]. U tohoto typu učení nelze určit, jak výrazně se výstup neuronové sítě liší od ideálního, jelikož není znám žádný výstup k porovnání. Je tedy zapotřebí trénovat neuronovou síť a testovat [1].

Výstupní neuron má normalizovanou hodnotu (<0; 1>), a zároveň tréninková data obsahující například údaje o spotřebě vozidla daného typu. V tomto případě je zapotřebí mít tato data v normalizované podobě. V opačném případě by výstupy neuronové sítě byly nekorektní [1].

K učení Multi-layer perceptron (dále jen „MLP“) sítě se z většiny používá tzv. Back Propagation algoritmus (BP). Jedná se o algoritmus zpětného šíření chyby. U netrénované MLP sítě jsou váhy mezi neurony nastaveny náhodně v intervalu <-0,5; +0,5>. Samotný princip trénování neuronové sítě není složitý. Jde o získání množiny dat z tréninkových dat, vložení do sítě a sledování výstupních dat. Poté se výstupní data porovnají se známým výsledkem a upraví se váhy mezi neurony tak, aby byla síť v příštím pokusu přesnější. Tento postup se provádí, dokud není dosaženo požadované přesnosti neuronové sítě [2].



Obrázek 5: Schématický diagram BP algortimu

zdroj: https://www.researchgate.net/profile/Sung-Eun-Kim-4/publication/275721804/figure/fig2/AS:651149220773890@1532257480242/Schematic-diagram-of-backpropagation-training-algorithm-and-typical-neuron-model.png [online] (2015)

https://www.researchgate.net/profile/Sung-Eun-Kim-4/publication/275721804/figure/fig2/AS:651149220773890@1532257480242/Schematic-diagram-of-backpropagation-training-algorithm-and-typical-neuron-model.png

3.1.6.1 Přeučení neuronové sítě

Důležitou součástí při učení MLF sítí je skladba dat, ze kterých se neuronová síť učí. Je potřeba, aby data reprezentovala jen zkoumanou oblast. Nesmíme zde vypustit žádný důležitý vztah mezi vstupními a výstupními daty. Nelze tedy spoléhat na to, že neuronová síť bude poté schopna tento vztah odvodit [1].

Dalším prvkem, který nesmíme zanedbat je doba učení. Pokud učení včas neukončíme, můžeme se dostat do problémů, jelikož neuronová síť se může přeučit. Tento se stav se projeví, tak že neuronová síť se naučí dokonale všechny kombinace předložených tréninkových dat a je schopna na ně reagovat. Nicméně reakce na odchylky od těchto naučených případů je velmi špatná [2].

K takovému problému jsou sítě MLF náchylné, obzvlášť když jsou špatně zvolené počty neuronů ve skrytých vrstvách. Například pokud je ve skryté vrstvě více neuronů, než je potřeba, může i krátký čas zapříčinit přeučení sítě. Problém přeučení

se dá řešit dobrou volbou neuronů ve skrytých vrstvách a také je zde možnost využít algoritmy, které zabrání přeučení sítě a proces učení včas zastaví [2].

3.1.7 Rozpoznávání obrazu

Rozpoznávání a klasifikace obrazu je jedním z nejaktivněji sledovaných oblastí v oborech vědy a inženýrství. Důvody jsou evidentní, schopnost nahradit lidské vizuální schopnosti strojem je velmi efektivní a existují různé aplikace. Hlavní myšlenkou je zkontrolovat obrazovou scénu zpracováním dat získaných ze senzorů. Takové přístroje mohou podstatně snížit pracovní zátěž a zlepšit přesnost rozhodování lidí v různých oblastech včetně armády a obrany, systémů biomedicínského inženýrství, monitorování zdraví, chirurgie, inteligentních dopravních systémů, výroby, robotiky, zábavy a bezpečnostních systémů [7].

Při tréninku modelu pro detekci objektu ve scéně je možné získat nemalé zlepšení výkonu a rychlosti, pokud je obraz nejprve převeden na stupně šedi. Například při detekci automobilů se neuronová síť nebude muset učit, že na barvě objektu záleží. Algoritmus se místo toho může zaměřit na identifikaci tvarů a textur, což může vést k mnohem rychlejšímu učení než učení se zpracováním barev. Obecným pravidlem v neuronových sítích je, že čím více tréninkových dat je poskytnuto, tím lepší výsledky lze očekávat. Nicméně tento princip nelze uplatnit vždy. Když po neuronové síti bude vyžadováno vetší množství funkcí, nelze se spolehnout, že bude poskytovat lepší a přesnější výsledky. Obecně je tento fenomén nazýván jako „kletba dimenzionality“ [6].

3.2 TensorFlow

TensorFlow je open-source softwarová knihovna pro numerické výpočty pomocí dataflow grafu. Uzly v grafu představují matematické operace, zatímco hrany grafu představují vícerozměrná datová pole (tensory) mezi nimi [9].

TensorFlow poskytuje několik rozhraní pro programování aplikací (dále jen „API“), které se liší úrovní kontroly nad programy. Nejnižší úroveň API v TensorFlow

se nazývá TensorFlow Core, která dává programátorovi možnost ovládat každý kus kódu a mít mnohem lepší kontrolu nad vytvořenými modely strojového učení. V TensorFlow je také řada API vyšší úrovně, která programátorům usnadňuje práci, jelikož jim poskytuje jednoduché rozhraní pro často používané úlohy. Všechna vyšší TensorFlow API jsou postavena na základu TensorFlow Core [8].

3.2.1 Tensor

Tensor je centrální datová jednotka v TensorFlow. Tensor se skládá ze sady primitivních datových typů v poli libovolných dimenzí. Tensorové primitivní datové typy jsou celé číslo, plovoucí desetinná čárka, znak, řetězec atd. Řadu dimenzí pole definuje tzv. „tensor rank“. Hodnota (rank) tensoru je rozměr pole tensoru. Tensor hodnoty 0 je určen pro tensory se skalárními hodnotami. Lze si všimnout, že pozice tensoru se liší od pole NumPy. Pole NumPy vrací počet prvků v dimenzi, ale tensorové řady vracejí počet dimenzí. Například délka tohoto pole [1, 2, 3] je 3 a dimenze 1. Tudíž „tensor rank“ je 1 [8].

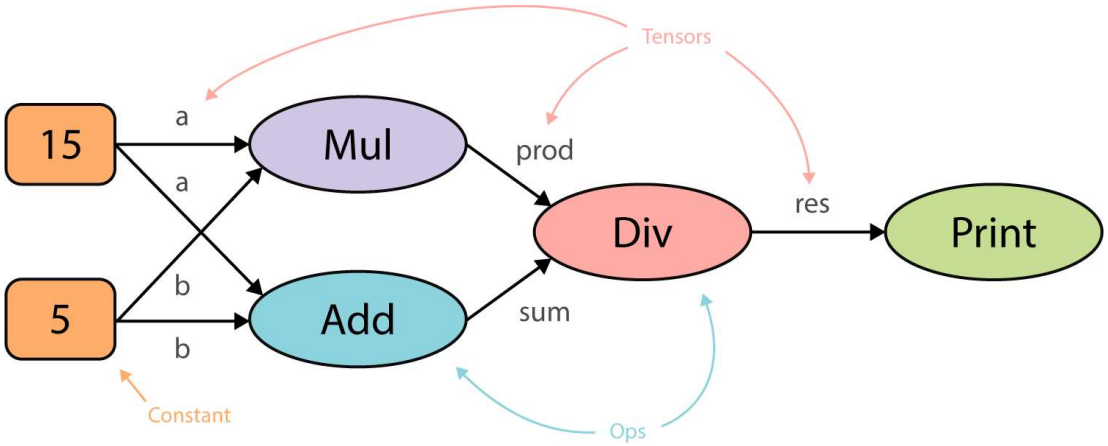
```
5 # Rank 0 tensor, scalar with [] shape
[4, 8] # Rank 1 tensor, [2] shape
[[3, 1, 7], [1, 5, 2]] # Rank 2 tensor, [2, 2] shape
[[[8, 3]], [[11, 9]]]] # Rank 2 tensor, [2, 1, 2] shape
```

Obrázek 6: Příklad Tensor ranku

zdroj: Gad, Ahmed, TensorFlow: A Guide to Build Artificial Neural Networks using Python (2017)

3.2.2 Dataflow graf

Dataflow je programovací model pro paralelní výpočty. V dataflow grafu uzly představují jednotky výpočtu (tensor nebo operaci) a hrany představují vstup a výstup výpočtu. Například operace *tensorflow.matmul* v TensorFlow vytvoří graf s jediným uzlem, který představuje metodu *tensorflow.matmul* připojenou ke dvěma linkám jako vstupy, což jsou dvě matice, které se mají znásobit a jedna linka jako výstup představující výsledek násobení [8]. Na obrázku níže je popsán dataflow graf s více metodami včetně zmiňované *matmul* metody.



Obrázek 7: Příklad Dataflow grafu

zdroj: Dominic E. (2021)

3.2.3 Session

TensorFlow používá třídu *tensorflow.Session* k reprezentaci spojení mezi klientským programem. Typicky program v Pythonu a běhové prostředí C ++. Objekt *tensorflow.Session* poskytuje přístup k zařízením v místním počítači a vzdáleným zařízením pomocí distribuovaného běhového prostředí TensorFlow. Také ukládá informace o *tensorflow.Graph*, takže lze efektivně spustit stejný výpočet vícekrát [9].

3.2.4 Proměnné a zástupné proměnné

Zástupné proměnné se používají k rezervování paměti pro budoucí použití. Jejich hlavní použití je pro vstupní trénovací data modelu. Využívá se, pokud má být stejná operace použita vícekrát, ale na různých vstupních datech. Vstupní data se vloží do zástupné proměnné, ta může nabývat různých hodnot [9].

3.2.5 Formát TFRecord

Formát TFRecord je jednoduchý formát pro ukládání posloupnosti binárních záznamů. Soubor lze číst pouze postupně (iterativně). Každý záznam obsahuje bajtový řetězec pro daný záznam, délku dat a hash CRC32C (32bitový CRC využívající Castagnoliho polynom) pro kontrolu integrity. Slouží k efektivnímu strukturování dat, které je možné použít multiplatformě. Je navržen pro použití s TensorFlow a používá se v celé škále TensorFlow API vyšší úrovně [9].

3.2.6 Protocol buffers

Protocol buffers (.pbtxt) od společnosti Google je jazykově neutrální, platformě neutrální a má rozšiřitelný formát. Jedná se o formát pro strukturování dat podobně jako XML. Nicméně tento formát je menší, rychlejší a jednodušší. Definovat jde individuálně, tak aby data byla v požadované struktuře. [11]

3.2.7 Příklad trénování lineárního modelu

U jednoduchého lineárního modelu existují vstupní data, váhy (weights) a předpětí (bias). Vstupy jeden po druhém budou přiřazeny do zástupné proměnné (placeholder) a její funkce se provede po každém přiřazení. Proměnné se používají pro uchování natrénovaných parametrů. Tudíž vstupní data mají být přiřazena zástupné proměnné, zatímco váhy a předpětí (bias) jsou proměnné. Tento naučený lineární model nemůže posoudit jeho přesnost (error rate). Pro zjištění přesnosti, je

zapotřebí mít základní pravdivostní data, která jsou následně porovnána s výstupem z natrénovaného modelu [9].

K zjištění přesnosti musíme použít funkci ztráty. Lze použít model standardní ztráty pro lineární regresi, která sečte druhou mocninu delt mezi výstupy aktuálně trénovaného modelu a poskytnutými pravdivými daty. Tím se vygeneruje skalární hodnota jako celková chyba trénovaného modelu [9].

```
1. import tensorflow
2.
3. W = tensorflow.Variable([.6], dtype=tensorflow.float32)
   #Weight
4. b = tensorflow.Variable([.2], dtype=tensorflow.float32)
   #Bias
5.
6. x = tensorflow.placeholder(dtype=tensorflow.float32)
   #Input Data
7.
8. linear_model = W * x + b
9.
10.     sess = tensorflow.Session()
11.
12.     #Initializing the variables
13.     init = tensorflow.global_variables_initializer()
14.     sess.run(init)
15.
16.     #Feeding the placeholder with data
17.     print(sess.run(linear_model, feed_dict={x: [1, 2, 3,
   4]}))
18.
19.     sess.close()
```

Obrázek 8: Příklad kódu pro lineární model

zdroj: Ahmed F. Gad. (2017)

TensorFlow tuto práci zjednodušuje, jelikož umožňuje provést vše výše zmíněné automaticky. TensorFlow obsahuje i řadu optimalizátorů, v tomto případě jde o optimalizátor „GradientDescentOptimizer“, který jsou schopen vypočítat pravdivá data a následně určit úspěšnost modelu [9].

3.2.8 Rozpoznání obrazu pomocí přeneseného učení

Přenesené učení mezi modely je dobrým příkladem všestrannosti a opětovného použití hlubokého učení, což je jeden z hlavních důvodů úspěchu oboru. Jedná se o učení již naučeného modelu pomocí jiných dat [12].

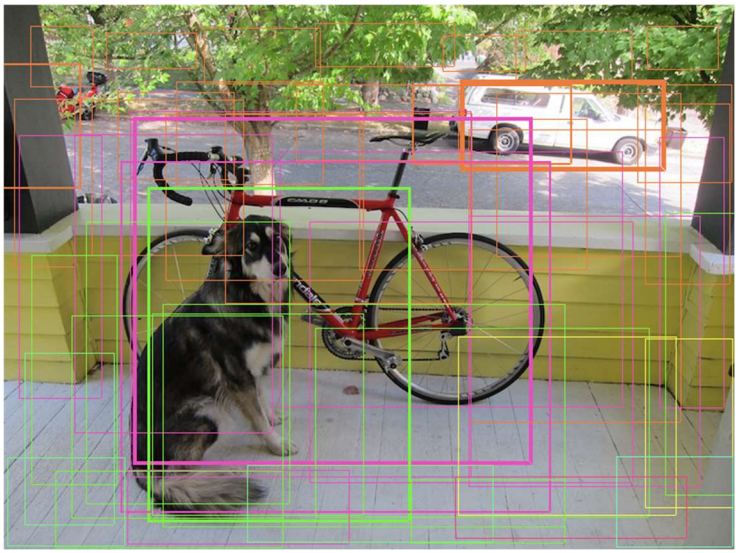
Detekce objektů zahrnuje rozpoznání určitých tříd objektů v obraze. Výstupem z natrénovaného modelu je nejen třída rozpoznaného objektu (o jaký typ objektu se jedná), ale i doplňující informace týkající se umístění objektu uvnitř obrazu (kde se objekt nachází). Jako příklad lze uvést detekci objektů používanou v autonomních vozidlech, kde je analyzován rámec vstupního obrazu, takže systém zobrazuje nejen typy zajímavých objektů, které jsou v obraze přítomny (jako jsou vozidla a chodci), ale také umístění, případný odhad velikostí a pozic těchto objektů v souřadném systému obrazu [12].

Důležitou součástí je příprava dat (obrázků), které je zapotřebí model naučit. Jedná se o získání dat, označení objektů v datech a následně vytvoření souboru typu *record* pro TensorFlow. [7]

3.2.9 Principy rozpoznávání obrazu

3.2.9.1 You Only Look Once

You Only Look Once (dále jen „YOLO“) je nejvíce efektivní princip rozpoznávání obrazu (poslední verze v5 z ledna 2021 [28]). Všechny vstupy do algoritmu jsou převzorkovány na pevnou velikost a rozděleny do mřížky S × S (např. 13 x 13). Každá buňka má následně přiřazenou třídu objektu, který se v ni nachází a má pevný počet B (např. 5) ohraničujících rámečků a jejich odpovídající skóre přesnosti je ihned vypočítáno. Pro každý ohraničující rámeček je také určeno, jaký objekt se v něm nachází. Výstupem jsou tedy ohraničující rámečky a pravděpodobnosti, jaké objekty se zde nachází. Podle příkladových hodnot je zde mřížka 13 x 13, což je 169 buněk. Následně každá buňka může určit 5 ohraničujících rámečků. Neuronovou sítí je následně vyhodnoceno 845 ohraničujících rámečků, nicméně většina z těchto rámečků obsahuje nízké hodnoty pravděpodobnosti objektu. Zde je použit práh přesnosti a následně výstupem jsou ohraničující rámečky, které mají pravděpodobnost objektu nad tímto prahem [13][14].



Obrázek 9: Ukázka rozpoznávání objektu pomocí YOLO (bez prahu přesnosti)

zdroj: https://github.com/pjreddie/darknet/wiki/YOLO:-Real-Time-Object-Detection [online] (2019)

3.2.9.2 Region Based Convolutional Neural Networks

Region Based Convolutional Neural Networks (dále jen „R-CNN“) má mnoho variant, nicméně v základním principu využívá algoritmu selektivního vyhledávání k vytvoření ohraničujících rámečků. V daném obrázku jsou tvořeny náhodné rámečky o různých rozměrech, v náhodných místech a náhodného počtu, kde jejich počet je omezen prahem. Neuronová síť následně vyhodnocuje jen obsah každého rámečku, jako jednotlivý vstupní obrázek pro doučení již existující neuronové sítě, nebo učení nové. V rozsahu rámečku je vypočítáno, jaké pixely spolu souvisí, jakou mají texturu a následně je vyhodnoceno, jaký objekt se zde nachází. Současně jsou upřesněny souřadnice ohraničujícího rámečku. Zde je podobně jako u principu YOLO využito prahu přesnosti [14].

3.2.10 TensorFlow Lite

Pro splnění požadavků na velikost mobilních aplikací zahájil Google v roce 2017 doprovodný projekt k hlavní řadě TensorFlow s názvem TensorFlow Lite. Tato knihovna je zaměřena na efektivní a snadné spuštění modelů neuronových sítí na mobilních zařízeních [22].

TensorFlow Lite se vejde do několika stovek kilobajtů, takže je mnohem jednodušší integrace do aplikace s omezenou velikostí. Má také vysoce optimalizované knihovny pro procesory řady Arm Cortex-A, spolu s podporou pro Android Neural Network API. Další klíčovou výhodou může být podpora pro 8bitovou kvantizaci hodnot v síti. Jelikož model může mít miliony parametrů, 75% zmenšení velikosti z 32bitových floatů na 8bitová celá čísla je samo o sobě užitečné [22].

TensorFlow Lite neumožňuje trénování modelu na mobilním zařízení, ale pouze vyhodnocení dat pomocí již naučeného modelu [22].

3.2.10.1 Konverze modelu do formátu tflite

Formát TensorFlow modelu *tflite* je určený pro využití naučeného modelu na mobilních zařízeních, mikrokontrolerech a minipočítačích, jako například Arduino apod. Konverzi je možné provést pomocí dvou způsobů, dle zdroje [24].

První způsob je pomocí Python scriptu, který je dostupný přímo od vývojářů TFlite. Jedná se script, který využívá knihovny TensorFlow a její části Lite, která umožňuje vytvořit tzv. „converter“, pomocí kterého je následně model převeden do formátu tflite [24].

Druhým způsobem je využití příkazového řádku a nainstalovaného TensorFlow 2 (instalace pomocí příkazu „*pip install --upgrade tensorflow*“). Jedná se o příkaz „*tflite_convert*“, který umožňuje převést uložený model i keras model [24].

3.2.11 TensorFlow.js

TensorFlow.js je jedna z variant TensorFlow knihovny. TensorFlow.js je knihovna, která umožňuje učení neurnových sítí v Javascriptu. Jak jeho název napovídá, TensorFlow.js je navržen tak, aby byl konzistentní a kompatibilní s TensorFlow-Core, základní knihovny pro Python [12].

Při posuzování vhodnosti Javascriptu pro typ aplikace jako jsou neuronové sítě a jejich učení, neměli bychom ignorovat fakt, že JavaScript je jazyk s výjimečně silným ekosystémem. Po celá léta byl JavaScript důsledně hodnocen jako číslo jedna mezi několika desítkami programovacích jazyků z hlediska počtu repositářů a aktivity na GitHubu (viz http://githut.info). Co se týče balíčků poskytovaných pomocí Node package manager (dále jen „npm“), existuje více než 600 000 balíčků (2018), to více než čtyřnásobně zvyšuje počet balíčků v PyPI (www.modulecounts.com). Navzdory tomu, že Python a R mají větší komunitu pro strojové učení a datovou vědu, komunita JavaScriptu buduje podporu pro data související se strojovým učením [12].

3.3 Flutter

Flutter je framework uživatelského rozhraní od společnosti Google. Využívá se pro vytváření moderních, nativních a reaktivních aplikací pro iOS, Android a web. Google také pracuje na verzi frameworku Flutter pro desktop, která je aktuálně v beta verzi, a také pro vestavěná zařízení (Raspberry Pi, automobilový průmysl a další). Flutter je open-source projekt hostovaný na GitHubu s příspěvky od Googlu a komunity. Flutter používá Dart, moderní objektově orientovaný jazyk, který se kompiluje s nativním ARM kódem a produkčním Javascriptovým kódem. Flutter používá vykreslovací modul Skia 2D, který pracuje s různými typy hardwarové a softwarové platformy. Využívá jej také Google Chrome, Chrome OS, Android, Mozilla Firefox, Firefox OS a další [15].

Flutter poskytuje vývojáři nástroje pro vytváření visuálně líbivých a profesionálně vypadajících aplikací a s možností přizpůsobit jakýkoli aspekt aplikace. Do uživatelského rozhraní lze přidat plynulé animace, detekci gest, atd [15].

Během vývoje používá Flutter „hot-reload“, podobně jako například React. Obnoví spuštěnou aplikaci v milisekundách, když se změní zdrojový kód, přidají se nové funkce nebo se upraví ty stávající. Použití „hot-reload“ je skvělý způsob, jak zobrazit změny, které se v kódu provedou [15].

3.3.1 Dart

Dart je základem učení se vyvíjení projektů pomocí frameworku Flutter. Jedná se o objektově orientovaný programovací jazyk (OOP), založený na třídách a syntaxe ve stylu C. Pokud vývojář zná základy jazyků C#, C++, Swift, Kotlin a Java / JavaScript, vývoj v programovacím jazyku Dart není náročný [15].

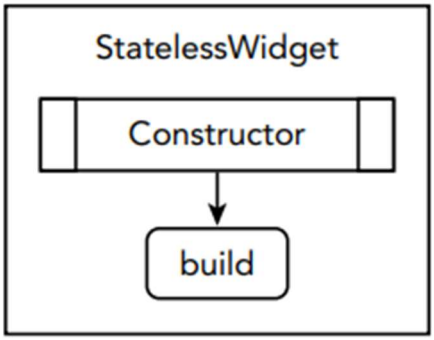
3.3.2 Widgety

Ve frameworku Flutter je téměř všechno widget. Widget je část uživatelského prostředí – například tlačítka, ikony, texty atd. Vzhledem k tomu, že vše je widget, je přirozeným důsledkem to, že Flutter do značné míry není nic jiného než obrovská hierarchie widgetů. Všechny widgety jsou třídy Dart a widgety mají obvykle pouze jeden konkrétní požadavek: musí dodat metodu *build()*. Tato metoda musí vracet objekt, který chceme zobrazit, tudíž další widget. Jsou zde výjimky tzv, „low-level“ widgety. Takové widgety vrací například primitivní typ proměnné, například jedná-li se o widget *Text*, typ proměnné *String*, která obsahuje text k zobrazení [16].

Každý Flutter widget může dědit od různých tříd, které Flutter nabízí. Nicméně jsou zde dvě základní třídy widgetů, od kterých z 99 % bude nový widget dědit. Jedná se o *StatelessWidget* a *StatefulWidget* [16].

3.3.2.1 StatelessWidget

StatelessWidget je postaven na základě své vlastní konfigurace a nemění se dynamicky. Na obrazovce se například zobrazí obrázek s popisem a nezmění se (je statický) [16]. Části UI jako ikony, které zobrazují malé obrázky, textové widgety, které zobrazují řetězce textu, považují se za bez-stavové widgety [16].

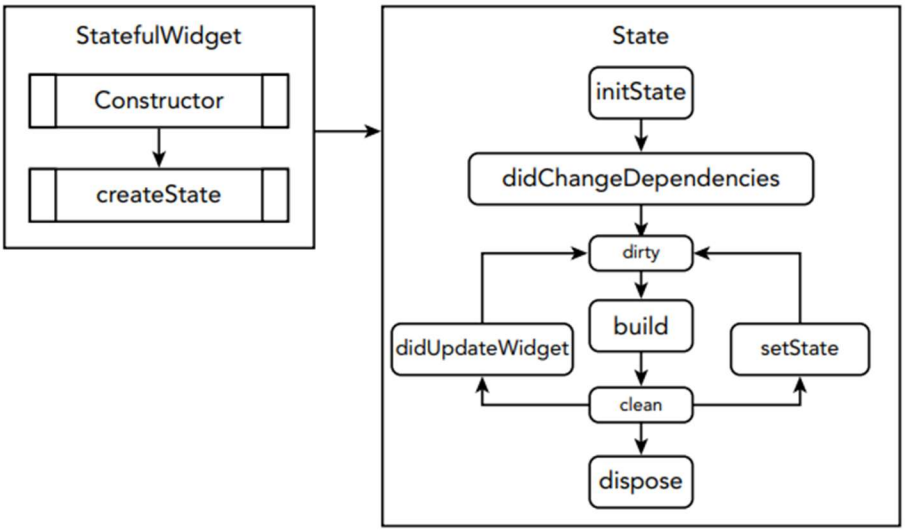


Obrázek 10: StatelessWidget – životní cyklus

zdroj: Marco L. Napoli (2020)

3.3.2.2 StatefulWidget

Opakem ke *StatelessWidget* je *StatefulWidget*. Jedná se widget, který se může měnit dynamicky a má svůj stav. Stavový widget je deklarován dvěma třídami, třídou *StatefulWidget* a třídou *State* [16]. Třída StatefulWidget je překreslena, když se změní konfigurace widgetu, ale třída State přetrvává, což zvyšuje výkon. Pokud je StatefulWidget odebrán ze stromu widgetů, které se vykreslují v danou chvíli a poté v budoucnu vložen zpět do stromu, vytvoří se nový objekt State. Na téměř identickém způsobu funguje například JavaScript knihovna React od Facebooku [16].



Obrázek 11: StatefulWidget – životní cyklus

zdroj: Marco L. Napoli (2020)

3.3.3 Výhody a nevýhody

Stejně jako u každého frameworku, je dobré vyhodnotit výhody a úskalí jakékoli možnosti, kterou bychom mohli zvážit, a Flutter se nijak neliší [16]. Flutter má své problémy a nemusí se hodit na každý projekt, nicméně pokud se jedná o multiplatformní aplikaci, je to jedna z nejlepších variant [16].

3.3.3.1 Výhody

- Funkce „hot-reload“ – automatické obnovení aplikace při změně zdrojového kódu [16].
- Multiplatformní – aplikace vyvíjené pomoci frameworku Flutter fungují jak na Android zařízeních, tak na iOS. Flutter nabízí dvě sady základních widgetů, a to pro Android a iOS [16].
- Dart – Dart je jednoduchý a objektově orientovaný, po překonání úvodního naučení se s tímto jazykem umožňuje vývojářům být velmi rychle produktivní. [16].

3.3.3.2 Nevýhody

- Velikost aplikace – aplikace bývají o něco větší než jejich čistě nativní protějšky, protože musí zahrnovat základní Flutter engine, rámec Flutter, podpůrné knihovny a další zdroje [16].
- Strom widgetů – s velmi hluboce vnořenou hierarchií může být náročné nahlížet do zdrojového kódu a pochopit strukturu [16].

4. Praktická část

Praktická část této bakalářské práce se zabývá strojovým učením pomocí TensorFlow a následné aplikování naučeného modelu v mobilní aplikaci založené na frameworku Flutter.

4.1 Neuronová síť

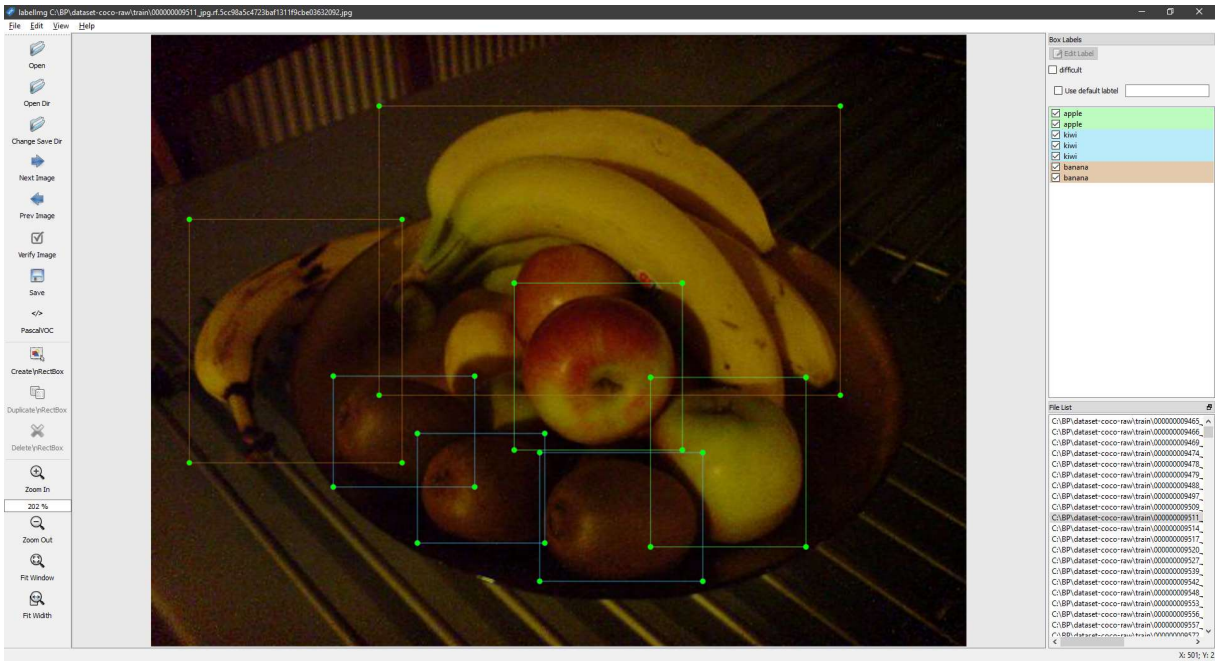
4.1.1 Příprava dat

Pro vytvoření vlastního datasetu (souboru obrázků s jejich popisem), je zapotřebí prvně nasbírat data. Data mohou být vlastní nebo volně dostupná z internetu. Pro sběr obrázků z internetu se dá využít mnoho již existujících scriptů/programů. Obrázky je doporučené si rozdělit na trénovací a testovací v poměru 80 % trénovacích a 20 % testovacích dat. Nicméně je zde i možnost využít už předpřipravených datasetů, které jsou veřejně dostupné.

Zde byla využita možnost veřejných datasetů, konktrétně Open Images Dataset V6. Jedná se o obrázkovou databázi od společnosti Google. V této databázi je možné vyhledat již připravené, označené obrázky určené jak k učení neuronové sítě, tak k testování.

K stažení dat z této databáze je možné využít OIDV4 toolkit [25]. Použití tohoto programu je velice jednoduché, je potřeba specifikovat pouze 3 parametry, a to třídu obrázku (např. *„apple“*), jaký druh dat je požadován (*train / test / validation*) a počet obrázků.

Pokud je dataset vytvářen bez pomoci podobný toolkitů a dostupných datasetů je zapotřebí obrázky nějakým způsobem popsat, aby neuronová síť věděla, co se nachází, na jakém obrázku. Způsobů, jak popsat data je více, záleží také na tom, jaký výsledný formát požadujeme. V této práci byl využit LabelImg [17] a Python script pro převedení popisků dat do formátu csv [18]. Výstupní formát z programu LabelImg je Pascal VOC, nebo yolo (txt). V tomto případě byl vybrán výstupní formát Pacal VOC (xml).



Obrázek 12: Ukázka označení objektů v obrázku

zdroj: vlastní zpracování

```
<object>
        <name>apple</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
                <xmin>285</xmin>
                <ymin>194</ymin>
                <xmax>417</xmax>
        <ymax>325</ymax>
  </bndbox>
</object>
```

Obrázek 13: Ukázka XML kódu označeného obrázku

zdroj: vlastní zpracování

Tyto popisky dat jsou následně převedeny do formátu csv pomocí Python scriptu [18], který už může být použitý k následnému trénování.

Dalším postupem v přípravě datasetu je vytvoření souboru typu TFRecord, na základě, kterého lze model trénovat, a to na jakékoliv platformě. Předtím, než je použit script na vytvoření souboru TFRecord [19], je zapotřebí ho pozměnit k vlastním

potřebám. V tomto případě je nutné upravit funkci *class_text_to_int*, která slouží k převedení názvu zvolené třídy objektu na číslo (index).

```
def class_text_to_int(row_label):
    if row_label == 'apple':
        return 1
    elif row label == 'kiwi':
        return 2
    elif row label == 'banana':
        return 3
```

Obrázek 14: Ukázka změny kódu pro vytvoření souboru TFRecord

zdroj: vlastní zpracování

Pro existující soubor TFRecord je zapotřebí vytvořit tzv. „label map“ soubor, ten slouží k zpětnému převedení z čísla (indexu) třídy na text. Zde využijeme formát Protocol buffers (.pbtxt).

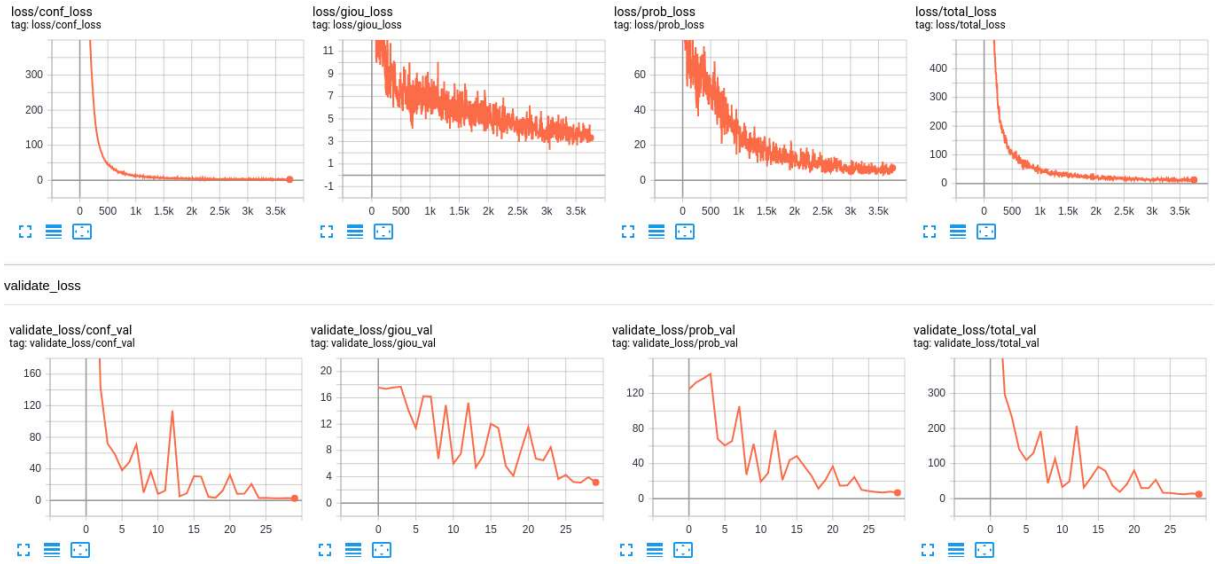
```
item {
  id: 1
  name: 'apple'
}
item {
  id: 2
  name: 'kiwi'
}
item {
 id: 3
 name: 'banana'
}
```

Obrázek 15: Ukázka souboru labelmap.pbtxt

zdroj: vlastní zpracování

4.1.2 Přenesené učení modelu

Byl zvolen postup využívající přeneseného učení modelu z důvodu efektivity a rychlejšího trénování. Jedná se o dotrénování již existujícího před-trénovaného modelu, který je naučený rozpoznávat obraz. Takový model byl trénován na velkém množství dat s velkým rozlišením. Samotné modely je možné získat ze stránek TensorFlow [20].



Obrázek 16: Porovnání přeneseného učení s klasickým učením v prostředí TensorBoard

zdroj: https://raw.githubusercontent.com/pythonlessons/TensorFlow-2.x-YOLOv3/master/IMAGES/tensorboard.png (2021)

Samotné přenesené učení je založeno na existujícím repositáři od TensorFlow [26]. Tento repositář poskytuje v podstatě vše potřebné pro samotný začátek s učením neuronové sítě.

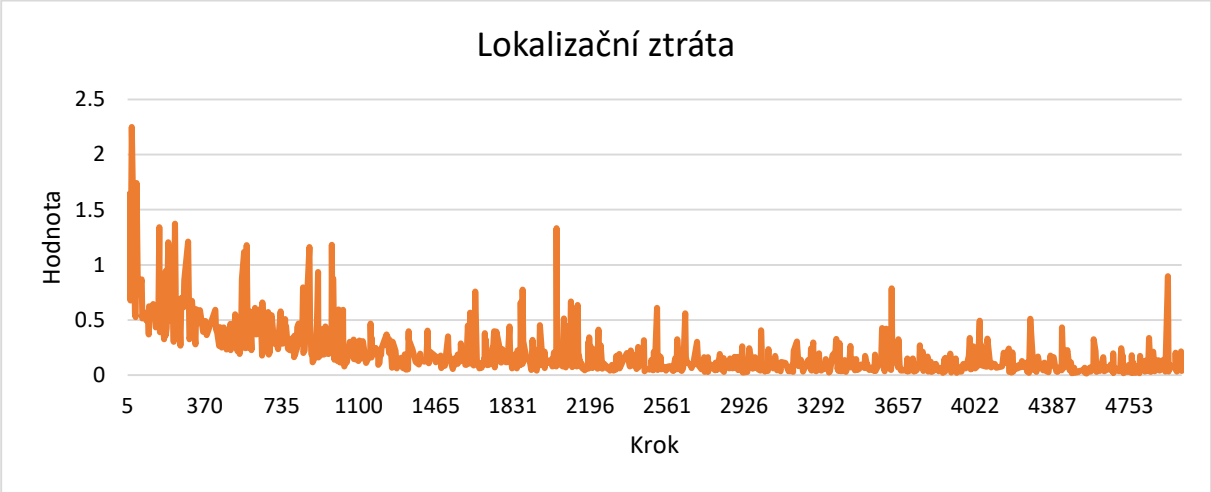
Pro přenesené učení je nutné využít již existujícího modelu. Zde byl zvolen model MobileNet V2 [27], jelikož se jedná o model určený pro mobilní zařízení. Ve staženém repositáři se ve složce *research* nachází složka *object_detection*. V této složce je doporučeno si vytvořit složku s odpovídajícím názvem modelu. Model [27] obsahuje v archivu i konfigurační soubor určený k trénování, a to jak k přenesenému, tak k vlastnímu novému trénování. Přenesené trénování využívá tzv. „checkpoint“. Ten slouží k obnovení natrénovaných vah neurnové sítě. Tyto záchytné body lze využít pro obnovení trénování i například při výpadku elektrické energie apod. V konfiguračním souboru trénování lze upřesnit, po jakém počtů kroků je záchytný bod vytvořen.

Nutným krokem ke spuštění trénování je správné nastavení konfigurace. V konfiguračním souboru se nachází proměnné, které je zapotřebí upravit dle vlastní potřeby. Jedná se o cestu k poslednímu checkpointu, ze kterého učení bude pokračovat, cestu k trénovacím a testovacím datasetům a výstupní cestu pro záchytné body. Důležitou součást je specifikovat i cestu k souboru „labelmap.pbtxt“. Tento

soubor slouží k následnému přeložení výstupní třídy obrázku z neuronové sítě, jelikož výstupem neuronové sítě jsou celočíselné hodnoty (Integer).

Učení (trénování) neuronové sítě lze sledovat pomocí vizualizace TensorBoard. V rozhraní Tensorboard lze nalézt různé grafy, jako například celkovou ztrátovost neuronové sítě, lokalizační ztrátu, přesnost a mnoho dalšího.

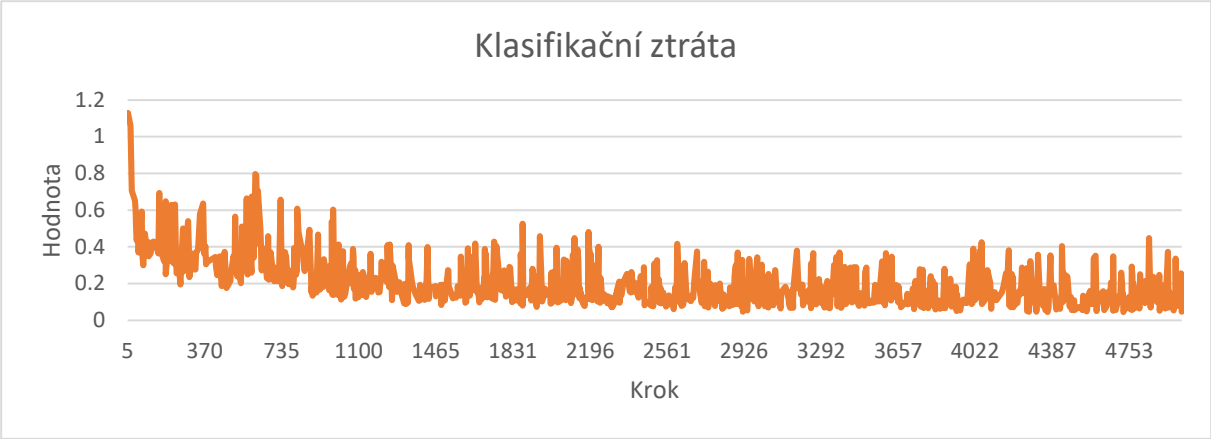
Vlastní model byl učen na jedné třídě objektů, a to konktrétně na detekci lidských očí. Model MobileNet V2 je zde zvolen kvůli efektivnímu fungování na mobilních zařízeních a také pro případnou možnost tento model využít pro detekci v reálném čase. Model je trénován na 5000 kroků.



Obrázek 17: Graf lokalizační ztráty

zdroj: vlastní zpracování

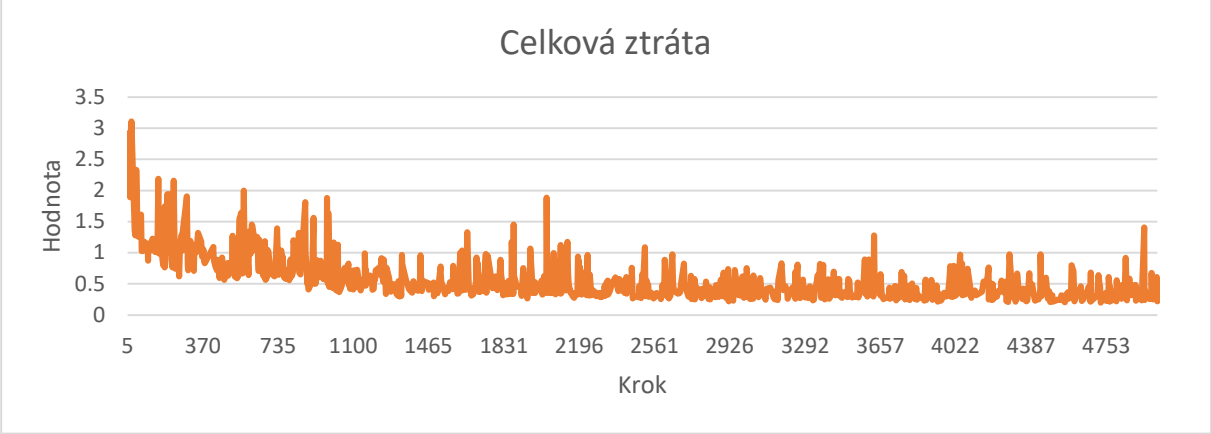
Lokalizační ztráta je hodnota chybovosti v určení lokalizace rozpoznaného objektu na snímku. Při vysoké chybovosti může vést k nepřesnému vykreslení ohraničujících obdélníků tzv. „*bounding boxes*“. Zde v posledních 1000 krocích průměrná hodnota nabývá 0.095.



Obrázek 18: Graf klasifikační ztráty

zdroj: vlastní zpracování

Klasifikační ztráta je hodnota chybovosti při klasifikaci objektu na daném snímku. Průměrná klasifikační ztráta v posledních 1000 krocích z celkových 5000 je 0.144.



Obrázek 19: Graf celkové ztráty

zdroj: vlastní zpracování

Graf celkové ztrát je výsledkem všech ztrát. Zde je průměrná hodnota za posledních 1000 kroků 0.337. Na grafu lze vidět, že k naučení této neuronové sítě by stačila přibližně polovina kroků, jelikož od kroků nad 2500 klesání celkové ztráty zpomalilo a spíše kmitalo.

4.1.3 Konverze modelu do formátu tflite

Formát TensorFlow modelu *tflite* je určený pro využití natrénovaného modelu na mobilních zařízeních, mikroprocesorech a minipočítačích, jako například Arduino apod. Konverzi je možné provést pomocí dvou způsobů, dle zdroje [24]. Zde byla využita varianta převedení modelu pomocí příkazového řádku, tedy TensorFlow funkce „*tflite_convert*“.

4.2 Mobilní aplikace

4.2.1 Struktura aplikace

Jedná se o aplikaci vyvíjenou primárně pro Android zařízení, nicméně jelikož je využíváno frameworku Flutter je zde možnost aplikaci používat na více platformách, které Flutter umožňuje.

Aplikace po spuštění zobrazí hlavní obrazovku, na které se vše odehrává. Nachází se zde tlačítko, které umožní uživateli vybrat snímek k rozpoznání. Vybrání snímku pro rozpoznání probíhá přes již existující aplikaci „Galerie“ v telefonu.

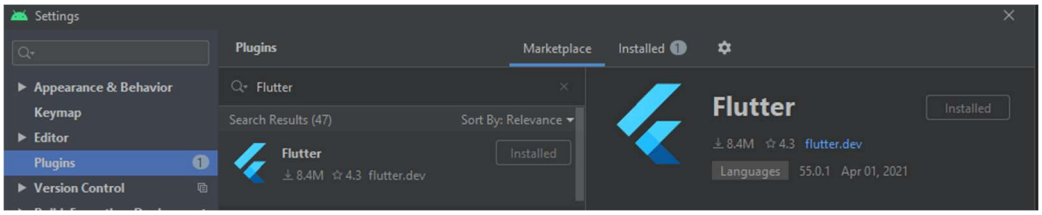
Po vybrání libovolného snímku dojde k přenesení námi vybrané fotografie z galerie na hlavní obrazovku aplikace. Zároveň se samotným vykreslením vybraného obrázku na domovskou stránku aplikace probíhá rozpoznání objektů na snímku. Po dokončení rozpoznávání jsou zobrazeny obdélníky okolo objektů, které naučený model rozpoznal, zároveň s jejich popiskem a procentuální přesností odhadu. Toto vykreslení je založené na natrénovaném modelu, který využívá knihovnu TensorFlow.

4.2.2 Zvolené technologie

Pro vývoj aplikace bylo zvoleno vývojářské prostředí Android Studio, které je založené na vývojářském prostředí IntelliJ IDEA. Android Studio bylo představené firmou Google v roce 2016 [21]. Aplikace také využívá frameworku Flutter, který umožní, aby tato aplikace mohla být používána multiplatformně. Samotný model je do aplikace implementován pomocí TensorFlow Lite.

4.2.3 Implementace

Prvním krokem k vytvoření Flutter aplikace je nainstalování pluginu Flutter do vývojového prostředí, zde se jedná konkrétně o vývojové prostředí Android Studio. V nastavení Android Studia v záložce „Plugins“ a v kartě „Marketplace“ vyhledáme plugin Flutter a nainstalujeme viz. obrázek 19.



Obrázek 20: Instalace pluginu Flutter

zdroj: vlastní zpracování

Po doinstalování vytvoříme Flutter projekt. Do projektu je zapotřebí přidat potřebné balíčky. Za balíčky je zde zodpovědný soubor *pubspec.yaml*, který obsahuje kompletní informace o aplikaci, včetně informací o dodatečných balíčcích a jejich verzích. Do tohoto souboru přidáme balíčky tflite, image a image_picker. Pokud balíček nemá uvedenou konkrétní verzi Android Studio, pracuje s poslední dostupnou verzí balíčku.

```
1.  dependencies:
2.      flutter:
3.          sdk: flutter
4.      image_picker: ^0.6.6
5.      tflite:
6.      image:
```

Obrázek 21: Ukázka přidání balíčků

zdroj: vlastní zpracování

Následuje vytvoření obalujícího widgetu v souboru *main.dart*, ve kterém se bude vše odehrávat. Jedná se o stateless widget, jelikož v tomto hlavním spouštěcím widgetu nepotřebuje držet žádné proměnné.

```
class TFLiteApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: TFLiteHome(),
    );
  }
}
```

Obrázek 22: Ukázka obalujícího staless widgetu

zdroj: vlastní zpracování

```
void main() {
  runApp(TFLiteApp());
}
```

Obrázek 23: Ukázka přidání staless widgetu do main funkce

zdroj: vlastní zpracování

Poté vytvoříme widget, u kterého už je zapotřebí, aby obsahoval stavy. Jedná se o statefull widget, jelikož je zapotřebí aby tento widget měl proměnné, které využijeme dále v aplikaci. Widget obsahuje více stavů, jelikož potřebujeme v proměnných udržet více informací, jako například vybraný obrázek k rozpoznání nebo list rozpoznaných objektů na obrázku. Statefull widget se skládá ze dvou částí, a to ze samotného Statefull widgetu, který vytvoří state a z třídy, která dědí state od Statefull widgetu a zároveň je obsahem widgetu. Tento widget je hlavním widgetem aplikace. Je zde obsažena všechna logika aplikace.

```
class TFLiteHome extends StatefulWidget {
  @override
  _TFLiteHomeState createState() => _TFLiteHomeState();
}
```

```
class _TFLiteHomeState extends State<TFLiteHome> {

File _image;
List _recognitions;
bool _isBusy;

@override
  Widget build(BuildContext context) {
```

```
9.      return Scaffold(
10.        appBar: AppBar(
11.          title: Text("TFLite Object detection"),
12.        ),
13.       bottomNavigationBar: FloatingActionButton(
14.            child: Icon(Icons.image),
15.            tooltip: "Select image",
16.            onPressed: () => selectImage(),
17.        ),
18.      );
19.    }
20.  }
```

Obrázek 24: Ukázka kódu statefull widgetu

zdroj: vlastní zpracování

Pro funkci uložení obrázku do statefull widgetu se zapotřebí využít dodatečného balíčku *image_picker*. Následně vybraný obrázek je pomocí metody *setState()* uložen do stavu widgetu.

```
1. selectImage() async {
2.     var image = await ImagePicker.pickImage(source:
   ImageSource.gallery);
3.     if (image != null) {
4.       setState(() {
5.         _isBusy = true;
6.       });
7.     loadImage(image);
8.     }
9.   }
```

Obrázek 25: Ukázka zdrojového kódu pro vybrání obrázku z galerie

zdroj: vlastní zpracování

```
loadImage(File image) async {
    if (image != null) {
      FileImage(image)
          .resolve(ImageConfiguration())
          .addListener((ImageStreamListener((ImageInfo info, bool _)
{
            setState(() {
              _imageWidth = info.image.width.toDouble();
              _imageHeight = info.image.height.toDouble();
            });
        })));

        setState(() {
          _image = image;
          _isBusy = false;
        });
        makePredictions(image);
      }
    }
```

Obrázek 26: Ukázka zdrojového kódu zpracování a uložení obrázku do stavového widgetu

zdroj: vlastní zpracování

Nutnou součástí celé aplikace je načtení natrénovaného modelu neuronové sítě ve formátu tflite. K načtení modelu je využita funkce z balíčku TFlite. Jedná se o funkci *loadModel()*. Tato funkce požaduje jako parametry cestu k modelu, cestu k názvům tříd objektů, které neuronová síť rozpoznává. Dalšími parametry mohou být například kolik vláken procesoru lze pro model využít apod.

Pro správné načtení modelu do aplikace je nutné přidat soubory modelu neuronové sítě do konfiguračního souboru *pubspec.yaml*. V tomto souboru se mezi zakomentovanými řádky nachází řádek, který obsahuje klíčové slovo „*assets*“. Do této sekce je možné přidat jakékoliv soubory, které aplikace může využívat.

```
# To add assets to your application, add an assets section, like
this:
  assets:
   - assets/tflite/model.tflite
   - assets/tflite/labels.txt
```

Obrázek 27: Ukázka přidání assetů do Flutter projektu

zdroj: vlastní zpracování

```
loadModel() async {
     await Tflite.loadModel(
          model: "assets/tflite/model.tflite",
          labels: "assets/tflite/labels.txt",

     );
  }
```

Obrázek 28: Ukázka funkce pro načtení modelu

zdroj: vlastní zpracování

Důležitým krokem je zajistit, aby model nebyl poškozen kompresí aplikace. K tomu, aby model nebyl poškozen je důležité upravit soubor *build.gradle*, který se nachází *android/app/build.gradle*. Do tohoto souboru je přidána funkce pro zrušení komprese určitého souborového formátu.

```
    aaptOptions {
        noCompress "tflite"
    }

```

Obrázek 29: Přidaná funkce pro zrušení komprese souborového typu tflite

zdroj: vlastní zpracování

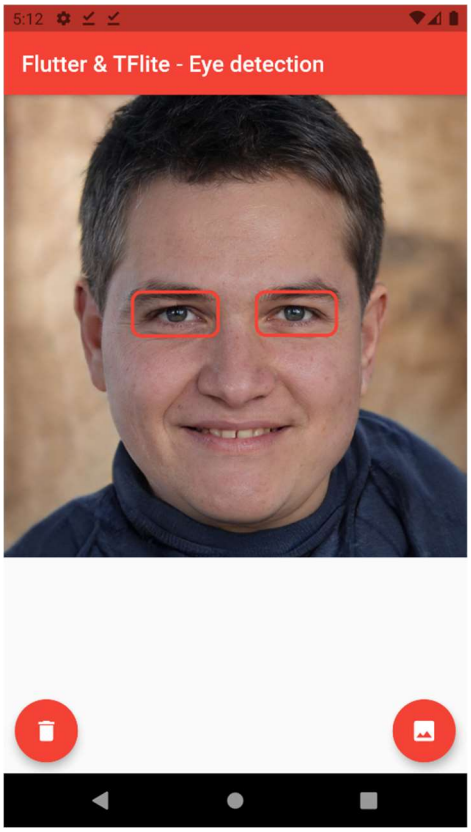
Výsledná aplikace pracuje s natrénovaným modelem, který je schopný rozpoznat lidské oko na obrázku, kde rozpoznané části jsou ohraničeny červeným rámečkem viz. obrázek 30 a obrázek 31.

```
  makeRecognition(File image) async {
    var recognitions = await Tflite.detectObjectOnImage(
        path: image.path,
        threshold: 0.25,
        imageMean: 0.0,
        imageStd: 255.0,
    );

    setState(() {
        _loading = false;
        _recognitions = recognitions;
      });
    }
```

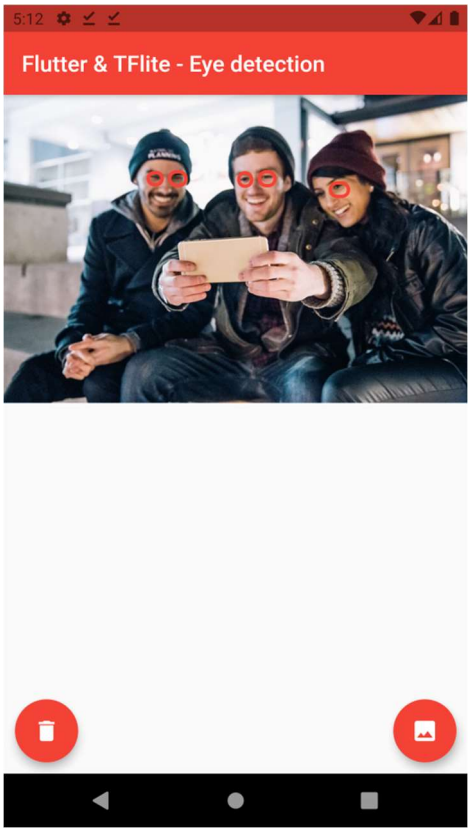
Obrázek 30: Ukázka zdrojového kód pro získání rozpoznaných objektů

zdroj: vlastní zpracování



Obrázek 31: Ukázka výsledné aplikace

zdroj: vlastní zpracování



Obrázek 32: Ukázka výsledné aplikace (více rozpoznání)

zdroj: vlastní zpracování

5. Shrnutí

5.1 Neuronová síť

Nejdůležitějším procesem v učení neuronové sítě pro rozpoznání obrazu je příprava dat. Neuronové sítě je možné naučit pomocí různých druhů datasetů, nicméně nejčastějším způsobem je vytvoření datového balíčku *tfrecord*, což je kompaktnější varianta datasetu. Je nutné dbát na správně popsaná data, pomocí kterých se neuronová síť učí.

Pomocí knihovny TensorFlow je možné vytvořit jakoukoliv neuronovou síť. Ačkoliv se knihovna na první pohled zdá jednoduchá člověk potřebuje určité znalosti týkající se neuronových sítí. Práci s frameworkem TensorFlow mohou komplikovat rozdílné verze s jinými funkcemi, nicméně TensorFlow 2 má už vše logicky uspořádané a je tedy s druhou verzí frameworku snazší pracovat. Zároveň není potřeba mít obavy ohledně starších funkcí z první verze, jelikož většina funkcí je implementována i ve verzi druhé. Co se týče přeneseného učení neuronové sítě, TensorFlow nabízí širokou škálu modelů k přenesenému učení, a to nejen k rozpoznání obrazu. Od přeneseného učení lze čekat rychlejší proces učení neurnové sítě, jelikož přenesené učení upravuje pouze určité vrstvy sítě. Nicméně i tento proces může zabrat i několik desítek hodin, pokud je učen na zařízení s nižším výpočetním výkonem.

Při učení neuronové sítě velice napomáhá pomocný program TensorBoard. Tento program poskytuje náhled do procesu učení. TensorBoard nabízí především přehled zobrazený v grafech, kde je možné nahlédnout na statistiky učení. Při učení rozpoznání obrazu je zde možné nahlédnout i na snímek (více snímků), který aktuálně neuronová síť zpracovává.

5.2 Flutter

Framework Flutter nabízí multiplatformní vývoj na mobilní zařízení, nyní už i na desktop (beta verze). Programování aplikace pomocí frameworku Flutter dokáže být v začátcích velice nepřehledné, nicméně Android Studio nabízí náhled na celý strom vykreslených komponent, což orientaci zlepšuje.

6. Závěry a doporučení

Cílem práce bylo zpracování mobilní aplikace využívající knihovny TensorFlow a frameworku Flutter. Samotný model je přizpůsoben použitím přeneseného učení. Jedná o přeučený model MobileNet V2. Výsledná mobilní aplikace následně pracuje s tímto přeučeným modelem a je tedy schopna rozpoznat objekt na snímku, konktrétně lidské oko (oči).

Neuronové sítě se pomocí knihoven jako je TensorFlow stávají jednoduššími a nabízená možnost využití natrénované neuronové sítě v mobilním zařízení je velkým pokrokem ve sféře neuronových sítí. Při učení vlastní neuronové sítě je dobrým krokem zvážení dostatečného výpočetního výkonu. Zde je doporučeno využít k učení neuronové sítě CUDA jádra grafických karet, čímž je proces učení výrazně urychlen. Učení pouze pomocí CPU není ideální variantou k naučení neuronové sítě.

Frameworku Flutter není přikládána taková pozornost, jako například konkurenčním řešením pro vývoj multiplatformních aplikací. Důvodem může být programovací jazyk Dart, který dokáže být velice nepřehledným. Programovací jazyk může být důvodem, proč jsou konkurenční řešení více populární a také efektivnější. Zde je nutno zmínit, že výběr řešení pro multiplatformní vývoj je velice individuální a záleží na preferencích.

Neuronové sítě mají v mobilních zařízeních veliký potenciál, a to jak ve zpracování obrazu, tak například v předpovídaní další akce uživatele. V dnešní době jsou mobilní zařízení schopné odhadovat, kam se jejich majitel například chce vydat. Neuronové sítě by mohly být podstatnou součástí v budoucnosti informatiky, jak v softwarové, tak hardwarové části.

7. Seznam zdrojů

[1] HEATON, Jeff. *Artificial intelligence for humans, volume 3: Deep learning and neural networks*. St. Louis: Heaton Research, 2015. ISBN 978-1505714340.

[2] BISHOP, Christopher M. *Neural networks for pattern recognition*. Oxford: Oxford Univ. Press, 1995. ISBN 978-019-8538-646.

[3] NIELSEN, Michael. *Neural networks and deep learning* [online]. Determination Press, 2019. Dostupné z: http://neuralnetworksanddeeplearning.com/

[4] RASHID, Tariq. *Make your own neural network: a gentle journey through the mathematics of neural networks and making your own using the Python computer language*. Oxford: [CreateSpace Independent Publishing Platform], [2016?]. ISBN 978-153-0826-605.

[5] CARTWRIGHT, Hugh, ed. *Artificial Neural Networks* [online]. New York, NY: Springer New York, 2015. Methods in Molecular Biology. ISBN 978-1-4939-2238-3. Dostupné z: doi:10.1007/978-1-4939-2239-0

[6] SHUKLA, Nishant. *Machine Learning with TensorFlow*. USA: Manning Publications Co., 2018. ISBN 978-1-61729-387-0.

[7] JAVIDI, Bahram. *Image Recognition and Classification* [online]. CRC Press, 2002. ISBN 9780429213311. Dostupné z: doi:10.1201/9780203910962

[8] GAD, Ahmed F. *TensorFlow: A Guide to Build Artificial Neural Networks using Python*. LAP LAMBERT Academic Publishing, 2017. ISBN 978-620-2-07312-7.

[9] BONNIN, Rodolfo. *Building Machine Learning Projects with TensorFlow*. Birmingham, UK: Packt Publishing, 2016. ISBN 978-1-78646-658-7.

[10] *TFRecord*. Dostupné z: https://www.tensorflow.org/tutorials/load_data/tfrecord

[11] *Protocol buffers* [online]. Dostupné z: https://developers.google.com/protocol-buffers/docs/overview

[12] CAI, Shanqing, Stanley BILESCHI, Eric D. NIELSEN a François CHOLLET. *Deep Learning with JavaScript Neural Networks in TensorFlow.js*. USA: FIRST INTERACT, 2020. ISBN 9781617296178.

[13] REN, Peiming, Wei FANG a Soufiene DJAHEL. A novel YOLO-Based real-time people counting approach. In: *2017 International Smart Cities Conference (ISC2)* [online]. IEEE, 2017, 2017, s. 1-2. ISBN 978-1-5386-2524-8. Dostupné z: doi:10.1109/ISC2.2017.8090864

[14] LI, Min, Zhijie ZHANG, Liping LEI, Xiaofan WANG a Xudong GUO. Agricultural Greenhouses Detection in High-Resolution Satellite Images Based on Convolutional Neural Networks: Comparison of Faster R-CNN, YOLO v3 and

SSD. *Sensors* [online]. 2020. ISSN 1424-8220. Dostupné z: doi:10.3390/s20174938

[15] NAPOLI, Marco L. *Beginning Flutter: A Hands-on Guide to App Development*. Indianapolis: John Wiley & Sons, 2020. ISBN 978-1-119-55082-2.

[16] ZAMMETTI, Frank. *Practical Flutter* [online]. Berkeley, CA: Apress, 2019. ISBN 978-1-4842-4971-0. Dostupné z: doi:10.1007/978-1-4842-4972-7

[17] *LabelImg* [online]. Dostupné z: https://github.com/tzutalin/labelImg

[18] *XML to csv* [online]. Dostupné z: https://github.com/datitran/raccoon_dataset/blob/master/xml_to_csv.py

[19] Generování souboru TFRecord [online]. Dostupné z: https://github.com/datitran/raccoon_dataset/blob/master/generate_tfrecord.py

[20] Před-trénované modely – TensorFlow [online]. Dostupné z: https://www.tensorflow.org/resources/models-datasets

[21] Android Studio [online]. Dostupné z: https://developer.android.com/studio/intro

[22] WARDEN, Pete a Daniel SITUNAYAKE. *TinyML*. USA: O'Reilly Media, 2020. ISBN 978-1-492-05204-3.

[23] *TensorFlow Lite* [online]. Dostupné z: https://www.tensorflow.org/lite/guide

[24] *Konverze modelu do tflite formátu* [online]. Dostupné z: https://www.tensorflow.org/lite/convert

[25] OIDV4 Toolkit [online]. Dostupné z: https://github.com/EscVM/OIDv4_ToolKit

[26] TensorFlow models [online]. Dostupné z: https://github.com/tensorflow/models

[27] MobileNet V2 [online]. Dostupné z: http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_mobilenet_v2_320x320_coco17_tpu-8.tar.gz

[28] Yolo V5 [online]. Dostupné z: https://github.com/ultralytics/yolov5

8. Přílohy

Příloha č.1 Zadání bakalářské práce

UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Akademický rok: 2019/2020

Studijní program: Aplikovaná informatika
Forma studia: Prezenční
Obor/kombinace: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

Jméno a příjmení:	**Lukáš Bařtipán**
Osobní číslo:	**I1700067**
Adresa:	Vodochody 99, Straškov-Vodochody – Vodochody, 41184 Straškov-Vodochody, Česká republika
Téma práce:	Strojové učení ve vývojovém prostředí Flutter
Téma práce anglicky:	Machine learning in Flutter framework
Vedoucí práce:	Ing. Karel Mls, Ph.D. Katedra informačních technologií

Zásady pro vypracování:

Cíl: Cílem bakalářské práce je vytvoření mobilní aplikace, využívající knihovny TensorFlow a frameworku Flutter k rozpoznání obrazu.
Osnova:

- Úvod
- Literární rešerše
- Volba metodologie, způsobu řešení
- Princip neuronových sítí
- Princip rozpoznávání obrazu
- Flutter
- Rozpoznávání obrazu v praxi a vyhodnocování
- Shrnutí výsledků
- Závěry a doporučení
- Seznam použité literatury

Seznam doporučené literatury:

[1] RASHID, Tariq. Make Your Own Neural Network, Createspace Independent Pub, 31. březen 2016, ISBN 1530826608.
[2] DRÁBEK, O.; TAUFER, I.; SEIDL, P. Umělé neuronové sítě – teorie a aplikace (1). CHEMagazín 2005, Sv. XV, 4.
[3] NIELSEN M., Neural Networks and
Deep Learning, Determination Press, 2015 (URL: http://neuralnetworksanddeeplearning.com/)
[4] MARCO, L. Beginning Flutter. John Wiley And Sons, 2019. ISBN 9781119550822.

Podpis studenta: Bařtipán Datum: 29. 4. 2021

Podpis vedoucího práce: Datum:

Příloha č.2 Elektronický archiv (.zip) obsahující:

1) Konfigurační soubor pro přenesené trénování
2) Složku s převedeným modelem do formátu *tflite*
3) Složku se zdrojovým kódem mobilní aplikace