UNIVERZITA MATEJA BELA V BANSKEJ BYSTRICI

FAKULTA PRÍRODNÝCH VIED

JEDNODUCHÉ PRÍKLADY Z UMELEJ INTELIGENCIE POSTAVENÉ NA DOSTUPNÝCH WEBOVÝCH NÁSTROJOCH

Bakalárska práca

fed7309d-48ad-47e6-b758-1dbf4cdd6bd4

Študijný program: Aplikovaná informatika

Študijný odbor: 18. - informatika

Pracovisko: Katedra informatiky

Vedúci bakalárskej práce: doc. Ing. Jarmila Škrinárová, PhD.

V Banskej Bystrici, 2021 **Miroslav Janovják**

3 Nástroje pre prácu s umelou inteligenciu

V nasledujúcej kapitole si predstavíme nástroje, ktoré sme v našej práci použili pri tvorbe príkladov, spojené s umelou inteligenciou.

3.1 Použité nástroje

Pri vytváraní našich príkladov s umelou inteligenciou sme použili niekoľko nástrojov. Na začiatok sme pracovali s nástrojom Machine Learning for Kids, ktorý vytvoril Dale Lane a nachádza sa na stránke https://machinelearningforkids.co.uk/. Tento nástroj sme využili pri trénovaní našich modelov. Tento nástroj sme použili z toho dôvodu, že ponúka jednoduché grafické zobrazenie trénovania modelu umelej inteligencie a jednoduchú prácu s trénovacími vzorkami. Následne sme s týmto modelom pracovali pomocou programovacieho jazyka Python, alebo sme použili nástroj Scratch, ktorý využíva blokové programovanie. Tento nástroj je možné nájsť na stránke https://scratch.mit.edu/.

Taktiež sme použili aj nástroj, ktorý umožňuje vytvárať aplikácie pre mobilné telefóny. Model sme vytrénovali pomocou nástroja Machine Learning for Kids a aplikáciu naprogramovali pomocou nástroja MIT App Inventor. Tento nástroj taktiež podporuje blokové programovanie a je možné ho nájsť na stránke http://appinventor.mit.edu/.

Akonáhle sme vytvorili dostatočné množstvo príkladov s týmito nástrojmi, rozhodli sme sa vytvoriť príklad aj v jednom zo zložitejších nástrojov. Vybrali sme si nástroj TensorFlow, ktorý vytvorila spoločnosť Google. Tento nástroj je možné nájsť na stránke https://www.tensorflow.org/. Vybrali sme si ho preto, pretože je dostupný zadarmo a je využívaný po celom svete, vo viacerých známych spoločnostiach. Ponúka veľké množstvo návodov pre začiatočníkov, ale aj pre pokročilých používateľov. Pomocou tohto nástroja sme vytvorili jednoduchú konvolučnú sieť, ktorá rozpoznáva obrázky pomocou počítačového videnia. Následne sme v programovacom jazyku Python pracovali s touto konvolučnou sieťou. TensorFlow takisto ponúka veľké množstvo rozširujúcich knižníc, ktoré nám výrazne urýchlili a zľahčili prácu s obrázkami, ale aj so samotnou neurónovou sieťou. V nasledujúcej kapitole sú všetky tieto nástroje a v nich vytvorené príklady podrobne opísané a graficky zobrazené.

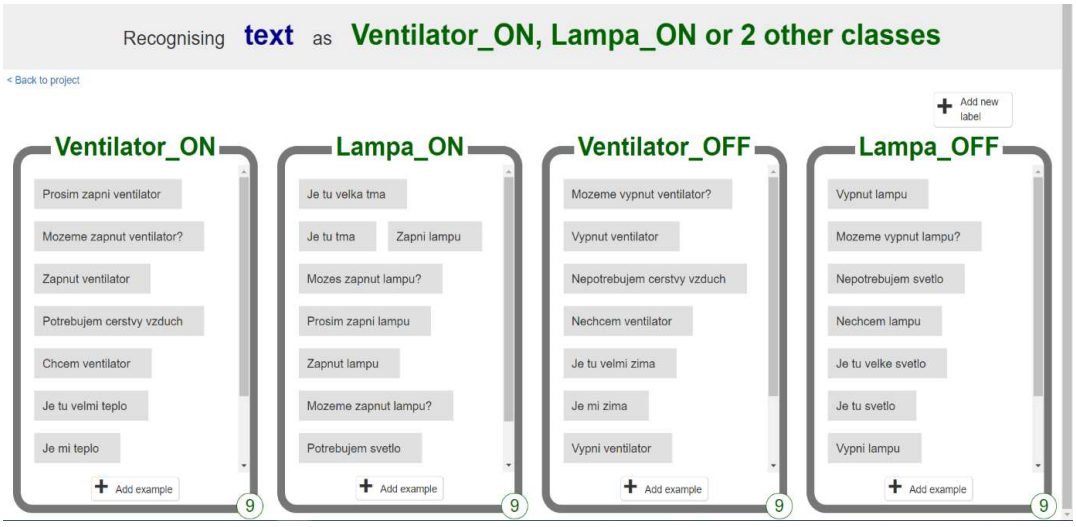
4 Tvorba príkladov z umelej inteligencie

Cieľom tejto kapitoly je predstaviť tvorbu príkladov a algoritmov, ktoré primerane a správne spopularizujú umelú inteligenciu pre širšiu verejnosť.

4.1 Virtuálny asistent

Túto aplikáciu predstavil pán Dale Lane už v roku 2017 (Lane, 2017) počas svojej práce. Rozhodli sme sa ju použiť ako inšpiráciu, pretože aplikácia je veľmi jednoduchá a známa z každodenného života. Ide o virtuálneho mobilného asistenta, ktorého môžete poznať napríklad aj z Vášho smartfónu. Pri Apple smartfónoch ako „Siri“, pri Android smartfónoch ako „Google asistent“.

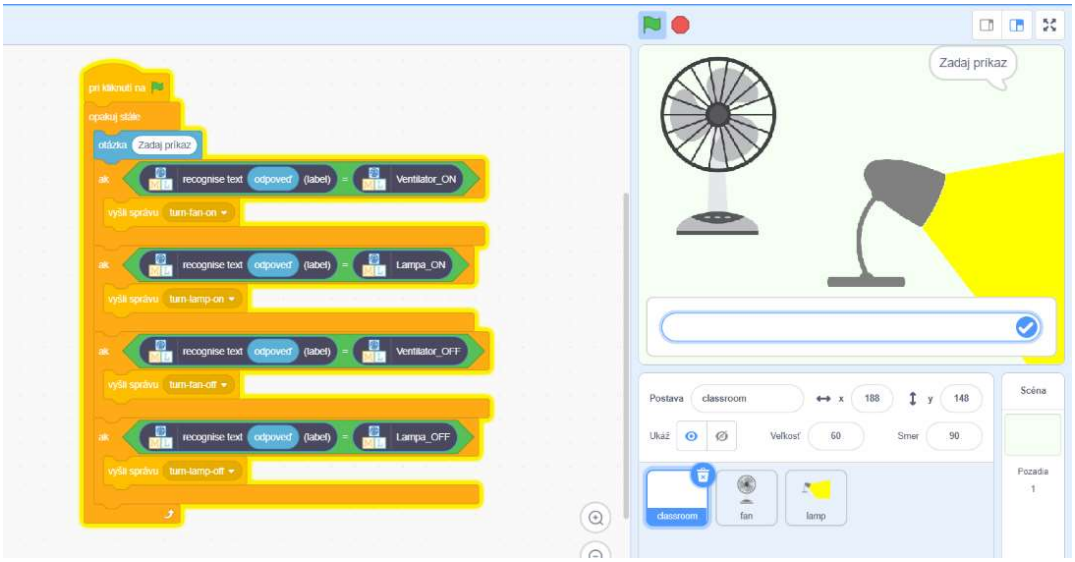
Ako prvé sme museli spísať čo najväčší počet konkrétnych vzoriek, ktoré bude aplikácia očakávať ako vstupný príkaz. V tomto prípade ide o slovné spojenia či vety, ktoré by sme použili, ak by sme od niekoho žiadali zapnutie alebo vypnutie ventilátora a lampy. Sú to napríklad vety ako „Prosím zapni ventilátor“, „Je tu tma“, „Je mi zima“, „Nepotrebujem lampu“ a iné. Pre každú takúto situáciu sa vytvorí samostatný zásobník vzoriek, do ktorého sa vypíšu konkrétne vzorky. Zásobníkmi sú v tomto prípade, ako ukazuje Obrázok 6, obdĺžniky s názvami „Ventilator_ON“, „Lampa_ON“, „Ventilator_OFF“, „Lampa_OFF“. V každom zásobníku sa nachádza deväť konkrétnych vzoriek, ktoré budú použité ako vstup. Výstupom bude reakcia na danú vzorku, to znamená zapnutie alebo vypnutie ventilátora a lampy.



Obrázok 6: Ukážka trénovania modelu pre aplikáciu Virtuálny asistent

Po zozbieraní dostatočného množstva vzoriek do každého zásobníka, je na rade krok kedy sa model všetky dané pojmy naučí. To znamená, že spracuje vstupné vzorky a podľa nich neskôr vykoná správny výstup. Táto akcia učenia trvá bežne niekoľko minút. Na záver, po vytrénovaní modelu je potrebné vytvoriť jednoduchý program. Pre túto aplikáciu Dale Lane použil nástroj „Scratch“, ktorý využíva blokové programovanie.

Obrázok 7 ukazuje funkciu, ktorá po štarte programu vyzve používateľa aby zadal konkrétny vstupný príkaz pre ovládanie lampy a ventilátora. Po zadaní príkazu, vytrénovaný model začne rozpoznávať text. Program prejde ďalej do podmienky, ak príkaz rozpozná ako vzorku zo zásobníka „Ventilator_ON“, ventilátor na výstupe sa začne točiť. Ak príkaz rozpozná ako vzorku zo zásobníka „Lampa_ON“, lampa na výstupe sa rozsvieti. Ak zadaný príkaz patrí do zásobníka vzoriek „Ventilator_OFF“, ventilátor na výstupe sa prestane točiť. Pri príkaze, zo zásobníka „Lampa_OFF“, lampa na výstupe sa zhasne.



Obrázok 7: Ukážka blokového programovania pre aplikáciu Virtuálny asistent v nástroji Scratch

V tejto podkapitole sme si predstavili ako sa trénuje model umelej inteligencie a aké vzorky sú pritom potrebné. Vysvetlili sme pojem zásobník, ktorý budeme ďalej používať v práci s modelmi. Nakoniec v programovacom nástroji „Scratch“, pomocou blokového programovania predstavili aplikáciu Virtuálny asistent, ktorú vytvoril Dale Lane.

4.2 Virtuálny asistent v programovacom jazyku Python

Aplikácia Virtuálny asistent, samozrejme nemusí byť naprogramovaná len pomocou blokového programovania. Preto sme sa sami rozhodli spracovať túto aplikáciu aj do

programovacieho jazyka Python. Na začiatok, je potrebné použiť takzvaný unikátny API kľúč, ktorý slúži na jednoznačnú identifikáciu, ktorý vytrénovaný model sa bude používať. Ten sme získali po vytrénovaní modelu. Obrázok 8 predstavuje zdrojový kód aplikácie Virtuálny asistent.

```
from mltext import classifyText, storeText
from mlmodel import trainModel, checkModel

API_KEY = "b6d7bdf0-64c1-11eb-b850-99d193cae1a2be52ce60-0973-4c69-893d-
0df4d3d2617e"

# KONTROLA CI JE MODEL PRIPRAVENY
status = checkModel(API_KEY)
print (status)

# ROZPOZNAVANIE TEXTU
# text ktory chcem odskusat
test_text = "zapnut lampu"
demo = classifyText(API_KEY, test_text)

label = demo["class_name"]
confidence = demo["confidence"]

# vypis vysledku
print ("Vysledok: '%s' s %d%% istotou" % (label, confidence))

# PRIDAVANIE NOVYCH PRIKLADOV DO MODELU
training_text = "Pokus2"

# konkretny zasobnik kam sa ma dat novy priklad
training_label = "Lampa_ON"

# funkcia na pridanie noveho prikladu
storeText(API_KEY, training_text, training_label)
print ("Vysledok: '%s' pridane do '%s' " % (training_text, training_label))

# VYTRENOVANIE MODELU
# funkcia na vytrenovanie modelu
trainModel(API_KEY)
print ("Trénujem model s novym prikladom")
```

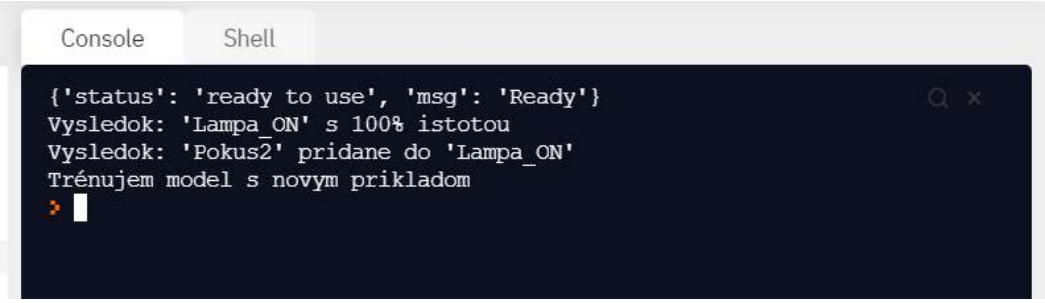
Obrázok 8: Ukážka zdrojového kódu pre Virtuálneho asistenta v jazyku Python

Pre porozumenie kódu je dobré povedať si akú činnosť jednotlivé funkcie vykonávajú:

- checkModel – táto funkcia skontroluje či bol zadaný unikátny API kľúč a vypíše svoj stav na obrazovku,

- test_text – do tejto funkcie je možné medzi úvodzovky vpísať vlastný text, ktorý aplikácia následne spracuje a bude porovnávať s ostatnými príkladmi ktoré boli na začiatku vložené do zásobníkov. Po skončení porovnávania, vypíše funkcia výsledok na obrazovku, v ktorom zásobníku sa otestovaný text nachádza a s akou percentuálnou istotou to tvrdí.
- training_text – táto funkcie slúži na pridávanie nových príkladov do modelu priamo z programu. Do úvodzoviek sa vpíše konkrétny pojem ktorý sa potom zaradí do modelu.
- training_label – podobne ako pri funkcii „training_text“ sa medzi úvodzovky vpíše text. V tomto prípade ale konkrétny zásobník, do ktorého sa má pojem z predchádzajúcej funkcie zaradiť.
- storeText – funkcia spracuje obe požiadavky z predchádzajúcich riadkov a uloží pojem do konkrétneho zásobníka. Na obrazovku vypíše aký pojem a do ktorého zásobníka sa pridáva.
- trainModel – táto funkcia na koniec vytrénuje model s novým príkladom, aby sa s ním mohlo pri ďalších spusteniach pracovať.

Výstup takéhoto programu v jazyku Python ukazuje Obrázok 9. Sú tam vypísané všetky výstupy funkcií. A to, do akého zásobníka patrí testovaný text a s akou istotou to aplikácia tvrdí, aký nový pojem a kam bol pridaný a následné trénovanie modelu s novým príkladom.

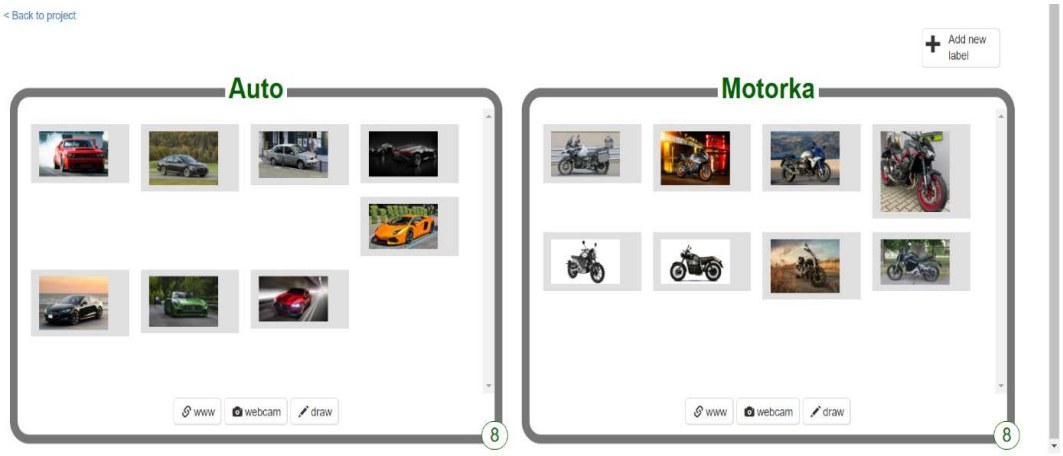


Obrázok 9: Ukážka výstupu aplikácie Virtuálny asistent

V tejto podkapitole, sme vytvorili program v jazyku Python pre aplikáciu Virtuálny asistent. Aplikácia nie je teda striktne zameraná len na programovanie pomocou blokového programovania.

4.3 Rozpoznanie obrázkov áut alebo motoriek

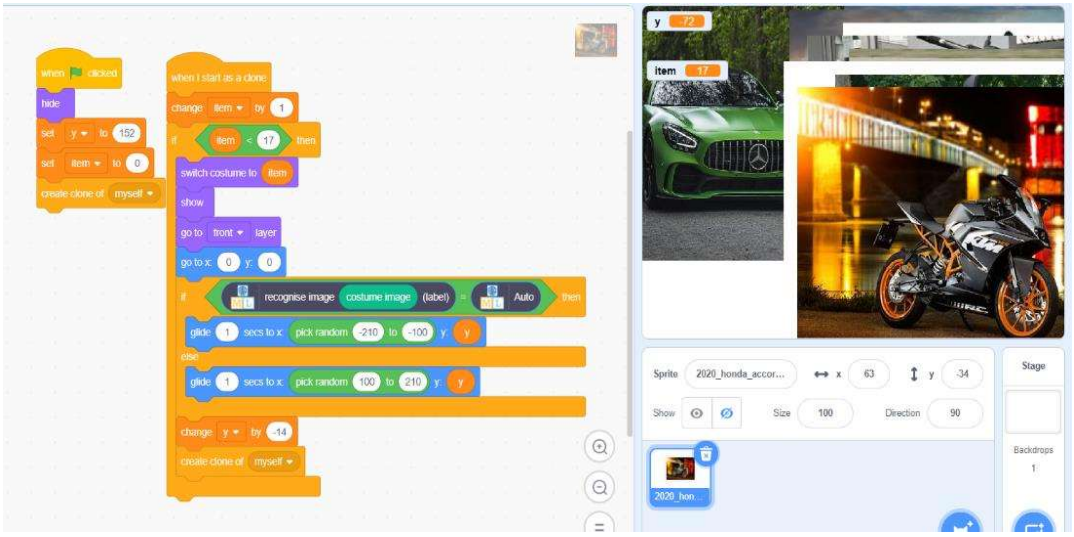
Ďalšia aplikácia je o niečo zložitejšia, keďže namiesto rozpoznávania textu rozpoznáva obrázky, ktoré dostane ako vzorky. Túto aplikácia taktiež vo svojom videu predstavil Dale Lane, počas toho ako študentom vysvetľoval umelú inteligenciu. (Lane, 2018). Spracoval ale aplikáciu, v ktorej medzi sebou porovnával obrázky áut a pohárov. My sme sa ale rozhodli použiť dva rôzne dopravné prostriedky a spracovať aplikáciu, v ktorej porovnávame obrázky áut a motoriek. Tak ako v prípade aplikácie Virtuálny asistent ktorá rozpoznáva text, znova je na začiatok potrebné vytvoriť zásobníky ktoré sa naplnia vzorkami obrázkov áut a motoriek. Ako ukazuje Obrázok 10, oba zásobníky sú naplnené ôsmimi obrázkami, ktoré zobrazujú vstupné vzorky áut rôznych značiek a rôzne typy motoriek. Je dobré zozbierať čo najviac vzoriek s rozdielnymi tvarmi dopravných prostriedkov, taktiež aj s rôznym pozadím obrázku.



Obrázok 10: Ukážka trénovania modelu pre rozpoznanie obrázkov áut a motoriek

Akonáhle má model dostatočné množstvo vzoriek, ďalším krokom je naučenie všetkých týchto vstupných obrázkov. Pri rozpoznávaní obrázkov, môže toto učenie trvať o niečo dlhšie, než pri aplikácii Virtuálny asistent, ktorá rozpoznáva text. Po tomto učení je nakoniec potrebné naprogramovať samotné rozhranie aplikácie. Pri tejto aplikácii sme znova použil nástroj „Scratch“, v ktorom sa používa blokové programovanie. Na začiatok sa všetky obrázky, s ktorými bude aplikácia pracovať uložia do samostatného kontajnera. Tento kontajner bude program postupne prechádzať a triediť obrázky. Ako predstavuje Obrázok 11 je zobrazená funkcia, ktorá má v sebe cyklus a ten predstavuje samotný kontajner obrázkov. Pri prechádzaní týmto cyklom, program obrázky v kontajneri rozpoznáva vďaka vzorkám, ktoré boli poskytnuté modelu pri jeho trénovaní. V tomto cykle sa nachádza ďalšia podmienka, ak obrázok z kontajnera je rozpoznaný ako auto,

obrázok sa presunie na ľavú stranu. V prípade že na obrázku sa nachádza motorka, obrázok z kontajnera bude presunutý na pravú stranu. Cyklus skončí po tom, akonáhle v kontajneri neostanú žiadne neroztriedené obrázky. Výstupom tejto aplikácie sú roztriedené obrázky dopravných prostriedkov, presnejšie autá na ľavej strane a motorky na pravej strane.



Obrázok 11: Ukážka blokového programovania pre triedenie obrázkov áut a motoriek v nástroji Scratch

Pri tejto aplikácii sme prezentovali, že umelá inteligencia dokáže rozpoznávať nielen text ale aj obrázky. Zozbierali sme vzorky dvoch dopravných prostriedkov a pomocou blokového programovania, ukázali ako ich model rozdelí na obrázky aut a obrázky motoriek.

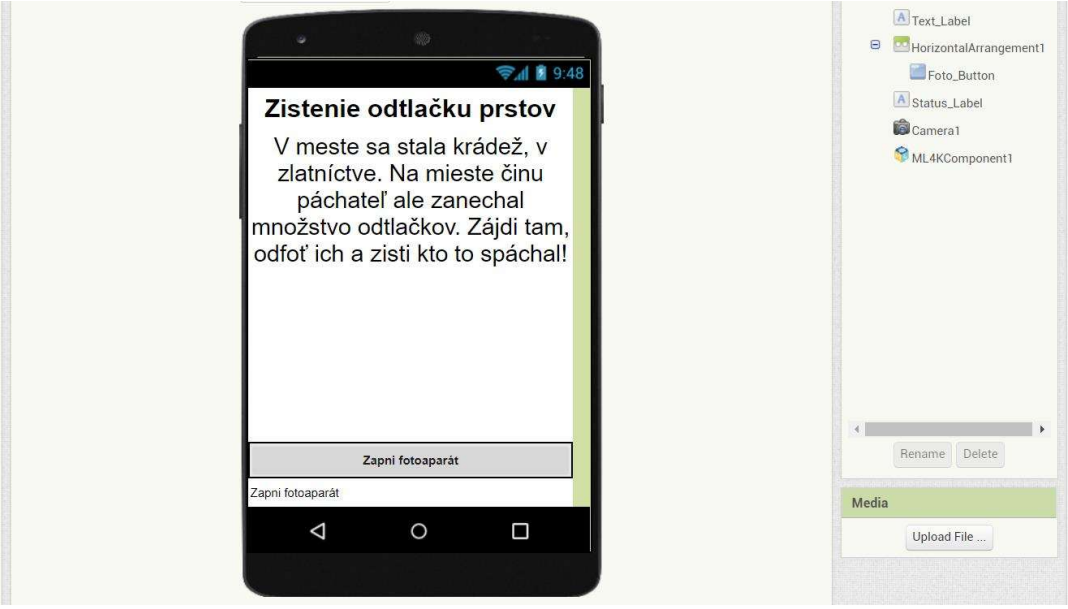
4.4 Rozpoznávanie konkrétnych odtlačkov prstov

V tomto príklade, sme vytvorili mobilnú aplikáciu, ktorá rozpoznáva odtlačky prstov niekoľkých osôb. Už nejde o jeden z príkladov, ktoré vytvoril pán Dale Lane, túto aplikáciu sme vymysleli, navrhli a vytvorili sami. Obrázok 12 ukazuje štyri osoby, z toho každá má pomocou pečiatky otlačený svoj palec na papier, pre účel vytrénovania modelu. Nejde o skutočné mená osôb ktorým patria odtlačky.

Keďže, rozpoznávanie odtlačkov prstov je omnoho zložitejšie než napríklad rozpoznávanie bežných objektov, je potrebné aj väčšie množstvo príkladov jednotlivých odtlačkov. Limit obrázkov pre projekt je ale nastavaný na 100, preto bol zvolený postup že každá osoba má pridelených 25 obrázkov svojho odtlačku palca.

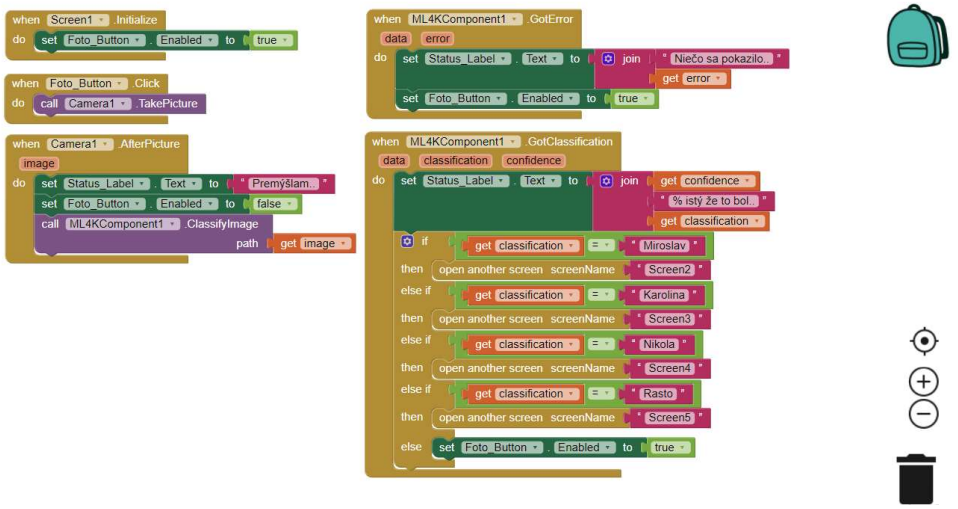
Obrázok 12: Trénovanie modelu pre rozpoznanie odtlačkov prstov

Po zozbieraní dostatočného množstva príkladov odtlačku palca, je možné pristúpiť k vytvoreniu samotného dizajnu aplikácie. Keďže ide o mobilnú aplikáciu, bol použitý nástroj MIT App Inventor. Ako ukazuje Obrázok 13 je zobrazená hlavná stránka aplikácie, ktorá sa zobrazí po spustení. Po stlačení spodného tlačidla sa otvorí fotoaparát telefónu a bude očakávaná fotka nového odtlačku. Po odfotení, sa fotka zanalyzuje a začne sa porovnávať s ostatnými príkladmi odtlačkov, ktoré boli uložené behom trénovania modelu.



Obrázok 13: Hlavná obrazovka aplikácie pre rozpoznávanie odtlačkov prstov

Nástroj MIT App Inventor používa pre tvorbu mobilných aplikácií blokové programovanie. Tento nástroj bol zvolený, pretože je jednoduchý a intuitívny.

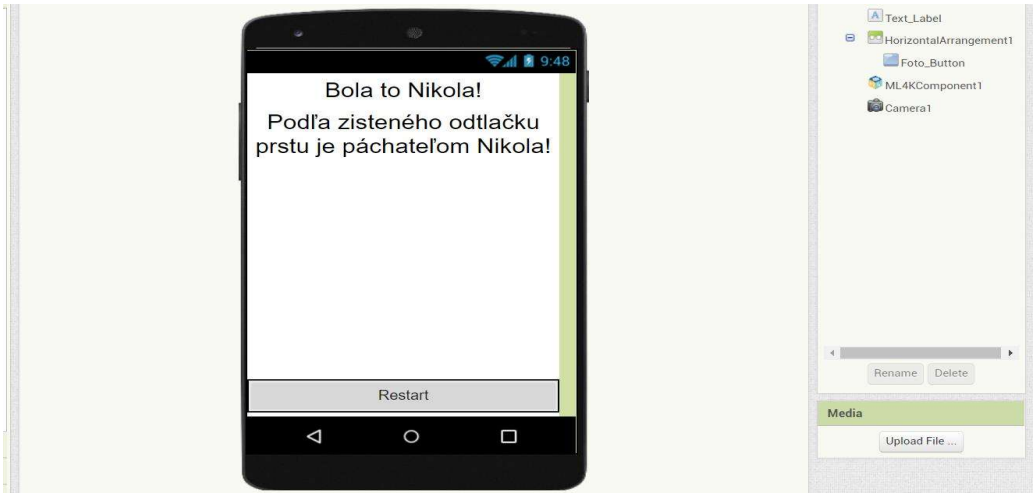


Obrázok 14: Blokové programovanie pre aplikáciu rozpoznávania odtlačkov prstov

Ako ukazuje Obrázok 14, program pomocou blokov pre takúto aplikáciu je veľmi jednoduchý. Skladá sa len z malého množstva funkcií.

- Pri spustení aplikácie sa zobrazí hlavná stránka a tlačidlo pre fotoaparát sa prepne do funkčného stavu,
- Po kliknutí na tlačidlo sa spustí fotoaparát mobilného telefónu,
- Po odfotení odtlačku, aplikácia zobrazí text "Premýšľam.." a začne analyzovať obrázok,
- V prípade vzniku neočakávanej chyby aplikácia vypíše chybovú hlášku,
- Aplikácia obdrží výsledok analýzy, vypíše text komu a s akou percentuálnou istotou odtlačok patrí. V podmienke potom vyhodnotí názov zásobníka s rovnakým odtlačkom a prejde na jednu zo štyroch obrazoviek s menom osoby.

Po naprogramovaní, je možné pristúpiť k testovaniu aplikácie. Po analýze fotky, sa aplikácia rozhodne ktorej osobe patrí otlačok palca a podľa výsledku sa spustí príslušná obrazovka. Obrázok 15 je ukážkou výstupnej obrazovky s menom osoby, ktorej otlačok bol vyhodnotený. Aj keď sú v aplikácii odtlačky palcov až štyroch osôb, výstupné obrazovky sú pre všetkých rovnaké, mení sa len meno osoby ktorej patrí otlačok.



Obrázok 15: Výstupná obrazovka aplikácie pre rozpoznávanie odtlačkov prstov

Navrhli a vypracovali sme aplikáciu ktorej nápad je náš vlastný. Zozbierali sme vzorky odtlačkov prstov od štyroch osôb a vytrénovali model, kde každá osoba má pridelených 25 vzoriek. Následne sme vytvorili mobilnú aplikáciu v nástroji MIT App Inventor.

Analýza a testovanie aplikácie pre rozpoznanie odtlačkov prstov

Pri samotnom používaní a testovaní tejto aplikácie sa ukázalo viacero rôznych a zaujímavých výsledkov. Testovanie je zhrnuté v Tabuľka 1.

Tabuľka 1: Testovanie aplikácie pre rozpoznanie odtlačkov prstov

Testovanie	Počet osôb	Počet vzoriek na osobu	Počet testovacích vzoriek	Počet úspešných rozpoznaní	Počet neúspešných rozpoznaní	Úspešnosť (%)
1.	2	35	10	10	0	100
2.	3	33	15	13	2	86,67
3.	4	25	20	12	8	60

Ako prvé bolo testovanie, odtlačkov len dvoch osôb. Každá z nich mala v modeli priradených 35 fotiek vlastných odtlačkov palca. Pri testovaní nových odtlačkov bolo spravených 10 pokusov, 5 pre každú osobu. Testovanie dopadlo tak, že všetkých 10 pokusov bolo správne vyhodnotených a nový odtlačok bol pridelený správnej osobe. To znamená 100% úspešnosť.

Po tomto úspešnom testovaní bola pridaná tretia osoba. Kvôli obmedzeniu projektu na 100 obrázkov, bol počet príkladov v modeli zmenšení na 33 príkladov pre každú osobu. Znova bolo použitých 5 nových odtlačkov od každej osoby, pre účel testovania. Keďže sa zmenšil počet príkladov v modeli, no úmerne s tým sa zväčšil počet testovaných osôb,

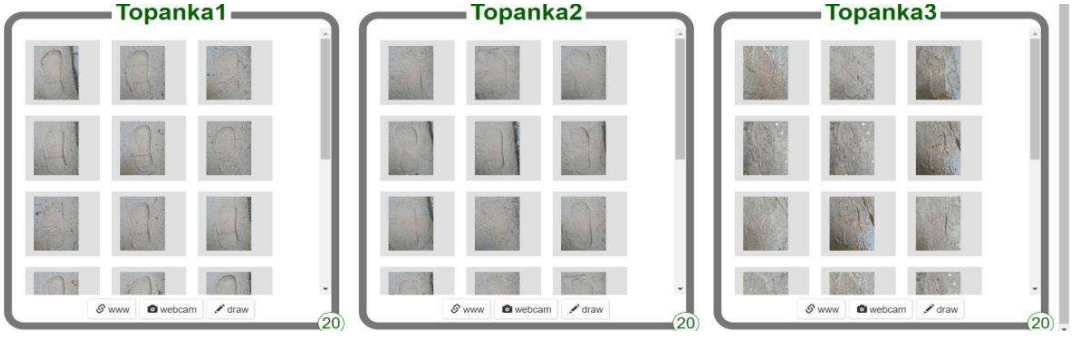
bolo očakávane že sa vyskytnú aj chybné výsledky. Test dopadol podľa očakávaní. Zo všetkých 15 pokusov bolo správne vyhodených 13 prípadov, nesprávny výsledok bol vyhodnotený v 2 prípadoch. To znamená že úspešnosť bola 86.67%.

Ako bolo ale ukázané na začiatku pri predstavovaní aplikácie, nakoniec sa pracuje so štyrmi osobami a ich odtlačkami. To znamená že po pridaní štvrtej osoby do modelu, sa počet príkladov pre každú osobu musel zmenšiť z pôvodných 33 odtlačkov, len na 25 odtlačkov. Testovanie sa tým ešte viac sťažilo, keďže počet príkladov sa znížil o veľký počet ale počet osôb sa znova zvýšil o jednu viac. Znova bolo vyhotovených päť nových odtlačkov od každého, to znamená že dokopy sa testovalo 20 prípadov. Podľa očakávaní, pre zložitosť testu bolo výsledkom viacero chybných pokusov. Z celkového množstva 20 pokusov, bolo 12 úspešných prípadov a 8 neúspešných prípadov. Tento výsledok ukazuje úspešnosť len 60%.

Vďaka týmto testom sa ukazuje pravdivosť tvrdení. Pri trénovaní modelov pre umelú inteligenciu, je vždy potrebné modelu poskytnúť dostatočný počet vzoriek, ktorý bude rozpoznávať. Ako ukázal posledný test, pri veľmi malom počte odtlačkov každej osoby a väčšom počte osôb, je vysoká pravdepodobnosť chybového výsledku. To je ale spôsobené obmedzeným počtom vzoriek, ktoré je možné použiť v nástroji pre trénovanie. V tejto podkapitole sme opísali testovanie aplikácie, na rozpoznávanie odtlačkov prstov.

4.5 Rozpoznávanie odtlačkov topánok

Aplikácia, ktorú sme sa rozhodli spracovať ako ďalšiu, rozpoznáva rôzne druhy a tvary odtlačkov topánok v blate. Ide o náš ďalší vlastný nápad na aplikáciu, ktorá rozpoznáva obrázky. Ako ukazuje Obrázok 16, vytvorili sme model do ktorého sme sa rozhodli dať 3 zásobníky rôznych topánok s 20 vzorkami ich odtlačkov.



Obrázok 16: Trénovanie modelu a vzorky pre rozpoznanie odtlačkov topánok

Po vytvorení dostatočného množstva vzoriek odtlačkov topánok a po úspešnom vytrénovaní modelu, sme sa rozhodli aplikáciu naprogramovať v programovacom jazyku Python. Zdrojový kód pre túto aplikáciu predstavuje Obrázok 17.

```
import requests
import sys

def fotoData(fotoURL):
    data = requests.get(fotoURL).content
    if sys.version_info[0] < 3:
        return data.encode("base64")
    else:
        import base64
        return base64.b64encode(data).decode()

def klasifikuj(obrazokUrl):
    kluc = "d75f9030-adcc-11eb-8716-171ad64b06c236be7759-5b97-4f55-9327-
9166cd9dc856"
    model = "https://machinelearningforkids.co.uk/api/scratch/" + kluc + "/classify"
    odpoved = requests.post(model, json={"data": fotoData(obrazokUrl)})
    if odpoved.ok:
        responseData = odpoved.json()
        topMatch = responseData[0]
        return topMatch
    else:
        odpoved.raise_for_status()

obrazok = klasifikuj(
    "https://lh3.googleusercontent.com/yrvmD1nKjKRor2Set1IiyiyY-xW16DTFIYa-
WHOLNiDA_G6IAa6e22UbGZsOPSMilKUIbQ=s85")

zasobnik = obrazok["class_name"]
istota = obrazok["confidence"]

print("Výsledok: '%s' s %d%% istotou" % (zasobnik, istota))
```

Obrázok 17: Ukážka zdrojového kódu pre aplikáciu na rozpoznanie odtlačkov topánok v jazyku Python

Na začiatok bolo potrebné pridať do programu prídavne knižnice pomocou príkazu import. Nasleduje funkcia „fotoData“. Tato funkcia spracúva internetovú adresu, ktorú dostane ako parameter od inej funkcie. Preto, je potrebne pridať do programu knižnicu „requests“, pretože tato funkcia požaduje internetovú adresu obrázka.

Funkcia klasifikuj najskôr skontroluje API kľuč, ktorý sme dostali pri trénovaní modelu, aby bolo jasné, ktorý model sa používa v programe. Premenná model, v sebe uchováva internetovú adresu kam sa bude posielať obrázok na klasifikáciu, spolu s API kľúčom.

Následne má funkcia v sebe premennú odpoveď. Táto premenná posiela internetovú adresu obrázka, ktorú dostane pri volaní funkcie klasifikuj nižšie. Túto adresu posiela funkcii „fotoData“, ktorá ju spracuje. Model dostane obrázok a ten ho následne začne porovnávať so vzorkami, ktoré dostal pri trénovaní. Program následne pošle odpoveď, pri ktorej ma najväčšiu zhodu obrázka so vzorkami.

Nasleduje používateľské telo programu, premenná obrázok v sebe volá funkciu klasifikuj a dáva jej vstupný parameter, internetovú adresu obrázka. Premenná „zásobník“, v sebe uchováva názvy zásobníkov modelu. Premenná "istota“, je dobrá pre percentuálne porovnanie výsledkov. Tieto premenné sa v programe nachádzajú hlavne, kvôli lepšej prezentácii výpisu výsledku.

Nakoniec je len výpis programu, aký je výsledok porovnávania vzoriek. Výstupom tejto aplikácie je text, do akého zásobníka patrí obrázok z internetovej adresy a s akou percentuálnou istotou to model tvrdí.

V tejto podkapitole sme navrhli a vytvorili aplikáciu, ktorá rozpoznáva odtlačky topánok. Model umelej inteligencie sme naplnili 20 vzorkami troch druhov topánok a následne tento model vytrénovali. Program pre aplikáciu, sme vytvorili v programovacom jazyku Python.

Analýza a testovanie aplikácie pre rozpoznávanie odtlačkov topánok

Túto aplikáciu sme taktiež podrobili testovaniu, využili sme pri tom 5 novovytvorených vzoriek každej topánky. Výsledky testovania prezentuje Tabuľka 2.

Tabuľka 2: Testovanie aplikácie pre rozpoznanie odtlačkov topánok

Testovanie	Počet topánok	Počet trénovacích vzoriek	Počet testovacích vzoriek	Počet úspešných rozpoznaní	Počet neúspešných rozpoznaní	Úspešnosť (%)
1.	3	20	15	12	3	80

Testovanie prebehlo na všetkých 3 druhoch topánok. Každá z topánok mala v modeli pridelených vlastných 20 vzoriek odtlačkov. Pri testovaní nových odtlačkov topánok bolo vyhotovených 15 nových odtlačkov, 5 pre každú topánku z modelu. Ako ukazuje výsledok testu v Tabuľka 2, z 15 testovaných pokusov bolo správne vyhodených 12. To znamená že 3 odtlačky topánok, boli nesprávne vyhodnotené a pridelené k nesprávnemu zásobníku. V percentách, predstavuje takýto výsledok 80% úspešnosť.

Týmto testovaním sme ukázali, že pri trénovaní modelu pre takúto aplikáciu, je ešte potrebné pridať viac vzoriek odtlačkov pre každú topánku. Pri aktuálnom počte vzoriek, 20 pre každý druh topánky, sa môžu vyskytnúť drobné chyby a pár neúspešných pokusov. V našom prípade to bolo 12 úspešných pokusov a 3 neúspešné pokusy.

V tejto podkapitole sme opísali testovanie aplikácie na rozpoznávanie odtlačkov topánok. Výsledky testovania sú zhrnuté v Tabuľka 2 vyššie.

4.6 Rozpoznávanie obrázkov pomocou nástroja TensorFlow

V tejto podkapitole si ukážeme, ako sme navrhli a spracovali aplikáciu, ktorá rozpoznáva dva rôzne predmety, konkrétne herný konzolový ovládač a hodinky. Tentoraz sme sa ale rozhodli aplikáciu spracovať pomocou nástroja, s názvom TensorFlow od spoločnosti Google. Tento nástroj umožňuje vytvoriť si vlastnú neurónovú sieť a pracovať s ňou.

4.6.1 Príprava údajov

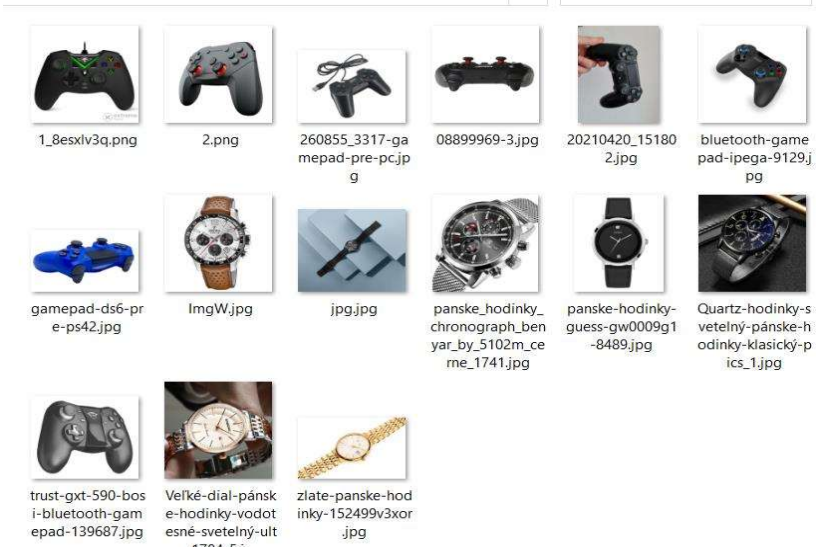
Predtým ako začneme s programovaním samotnej aplikácie, je potrebné vytvoriť si hlavný priečinok, v ktorom sa budú nachádzať všetky vzorky. V našom prípade, vzorky znamenajú obrázky, s ktorými bude naša aplikácia pracovať. Obrázky, ktoré sú určené na trénovanie modelu, validáciu modelu a jeho testovanie. V tomto hlavnom priečinku si vytvoríme tri menšie adresáre s názvom „testing“, „training“ a „validation“. Do adresára „training“, vložíme vzorky herného ovládača a vzorky hodiniek. Konkrétne 25 vzoriek herného ovládača a 25 vzoriek hodiniek vo formáte “JPG“ alebo vo formáte “PNG“. Aby sa vzorky od seba odlišovali, rozhodli sme použiť rôzne druhy ovládačov a hodiniek, ktorých obrázky sme použili z internetu. Obrázok 18 a Obrázok 19 predstavujú trénovacie vzorky. Ako hovorí názov adresára, tieto vzorky budú použité pri trénovaní modelu.

Obrázok 18: Ukážka trénovacích vzoriek hodiniek

Obrázok 19: Ukážka trénovacích vzoriek herných ovládačov

Podobne to spravíme aj pri adresári „validation“, naplníme ho vzorkami hodiniek a herného ovládača. Takisto sme použili obrázky herných ovládačov a hodiniek vo formáte “JPG“ alebo vo formáte “PNG“. Táto vzorka údajov bude použitá na objektívne vyhodnotenie modelu. To znamená, či zapadajú do množiny trénovacích údajov.

Adresár „testing“ využijeme až na konci našej práce, keďže tam umiestnime niekoľko vzoriek, ktorými otestujeme náš model, či je správne vytrénovaný. To znamená, že tam umiestnime vzorky hodiniek a herných ovládačov, ktoré sme nepoužili pri trénovaní modelu. Konkrétne sme tam umiestnili 8 obrázkov herného ovládača a 7 obrázkov hodiniek. Model ich potom bude mať za úlohu správne rozpoznať a pomenovať o aký predmet ide. Obrázok 20 predstavuje vzorky určené na testovanie modelu.



Obrázok 20: Testovacie vzorky

4.6.2 Tvorba a implementácia aplikácie na báze TensorFlow

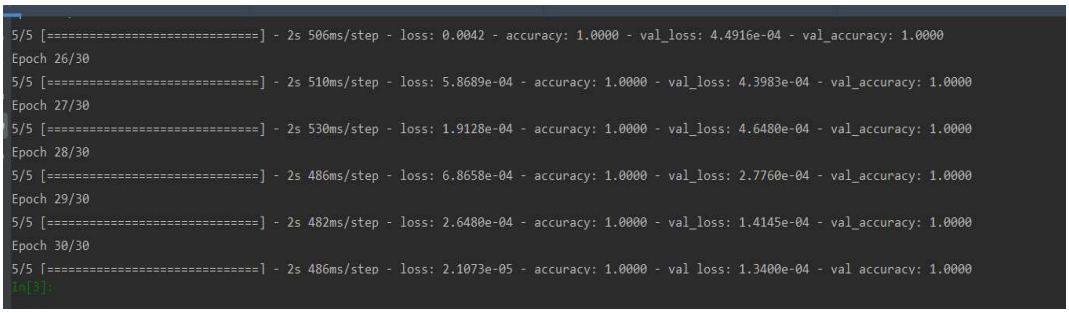
Po zozbieraní všetkých potrebných vzoriek sa presunieme k naprogramovaniu aplikácie. Celý zdrojový kód sa nachádza v Prílohe A. Spoločne s nástrojom TensorFlow sme použili programovací jazyk Python. Na začiatok vložíme všetky potrebné knižnice. V tomto prípade ich je viac, keďže využívame knižnice nástroja TensorFlow a taktiež viaceré knižnice pre prácu s obrázkami a ich zobrazenie na výstupe. Všetky vzorky ktoré sme vložili do priečinkov „training" a „validation", potrebujeme konvertovať do množiny údajov, s ktorými bude neurónová sieť pracovať. Na to využijeme funkciu „flow_from_directory", ktorej ako parameter dáme systémovú cestu k adresárom „training" a „validation". V tejto funkcii nastavíme, aby zmenila veľkosť všetkých vzoriek ktoré sme do adresárov vložili. A to konkrétne na veľkosť 200x200 pixelov.

Akonáhle máme naše vzorky konvertované do množiny údajov, potrebujeme si zadefinovať model. K tomu použijeme takzvanú konvolučnú neurónovú sieť, ktorá je efektívny nástroj v oblastiach rozpoznávania obrazu a klasifikácie. Pre rozpoznávanie používajú takzvané filtre. Platí, že koľko je nastavených filtrov, toľko vznikne výstupných príznakových máp. My sme na začiatok nastavili 16 filtrov, no s každou vrstvou sa počet filtrov zvyšoval. Každý filter má vlastný parameter a to je jeho veľkosť, v našom prípade sme ich nastavili na 3x3. Ako aktivačnú funkciu sme zvolili „ReLU". Ide o nelineárnu funkciu, ktorá sa aplikuje na každú vytvorenú príznakovú mapu. Následne prebehne funkcia „MaxPool". Ide o operáciu ktorá sa používa na zmenšenie príznakových máp. Z každého okna príznakovej mapy, sa ponechá len maximálna hodnota. Výstupy z jednotlivých okien sa spoja a vytvoria menšiu príznakovú mapu. Množina takýchto máp sa použije ako vstup do ďalšej vrstvy neurónovej siete s väčším počtom filtrov a proces sa opakuje.

Po zadefinovaní vrstiev je potrebné model skompilovať. Jedným z argumentov potrebných pre kompiláciu je „optimizer". Zvolili sme „RMSprop" optimizer, ktorý udržiava kĺzavý priemer a tento priemer používa na odhad odchýlky. Tento optimizer má v sebe aj parameter, takzvaný učiaci pomer ktorý je štandardne nastavený na 0.001.

Nasleduje konkrétne trénovanie nášho modelu pomocou funkcie „model.fit". Ako parameter vložíme premennú, ktorá má v sebe naše vzorky určené na trénovanie. Táto funkcia taktiež vyžaduje nastaviť počet „epochs", „steps_per_epoch" a „validation_data". „Epochs" znamená počet iterácii, ktorými model prejde celú množinu vzoriek určených

na trénovanie modelu. „Steps_per_epoch“ znamená, aký počet krokov musí prejsť aby bola úspešne ukončená jedna epocha. V našom programe sme zvolili menší počet krokov, aby sa model trénoval rýchlejšie, ale nastavili sme 30 epoch. To znamená, že model 30 krát prejde celú našu množinu vzoriek určených pre tréning. Obrázok 21 ukazuje ako vyzerá takéto trénovanie. V nastavení pre „validation_data“ sme vložili premennú, ktorá má v sebe naše vzorky určené pre validáciu.



Obrázok 21: Ukážka trénovania modelu pomocou nástroja TensorFlow

Nakoniec, je už len potrebné zadefinovať si premennú, ktorá bude pracovať s adresárom určeným pre testovanie. V adresári „testing“ prebehne cyklus, ktorý všetky testovacie vzorky zmenší na rozmery 200x200 pixelov a pomocou funkcie „plot“ ich postupne zobrazí. Keďže, pracujeme len s dvoma triedami vzoriek, prvá trieda so vzorkami herných ovládačov má hodnotu 0 a druhá trieda so vzorkami hodiniek má hodnotu 1. Do premennej „val“ si zadefinujeme funkciu „model.predict“. Táto funkcia spracuje všetky testovacie vzorky a pomocou podmienky vyhodnotí, či sa jedná o triedu s herným ovládačom alebo o triedu s hodinkami.

4.6.3 Testovanie aplikácie pre rozpoznávanie na báze TensorFlow

Po úspešnom dokončení aplikácie sme sa rozhodli ju podrobiť pár testom. Pri prvom spúšťaní aplikácie, sme jej zadali určitý počet vzoriek pre trénovanie, konkrétne 25 vzoriek pre ovládač a 25 vzoriek pre hodinky. Pri testovaní, to bolo 8 vzoriek pre ovládač a 7 vzoriek pre hodinky. Dôležitým parametrom je aj počet trénovacích epoch a počet krokov jednej epochy. To, sme pri prvom spúšťaní nastavili na 30 epoch a 5 krokov v rámci jednej epochy. Preto sme sa rozhodli aplikáciu podrobiť ďalším testom. Výsledky testovania predstavuje Tabuľka 3.

Tabuľka 3: Testovanie aplikácie pre rozpoznávanie na báze TensorFlow

Testovanie	Počet trénovacích vzoriek	Počet epoch	Počet krokov epochy	Počet úspešných rozpoznaní	Počet neúspešných rozpoznaní	Úspešnosť (%)
1.	20	5	3	9	6	60%
2.	20	10	5	12	3	80%
3.	25	15	3	14	1	93,33%
4.	25	15	5	15	0	100%
5.	30	20	5	15	0	100%

Celkovo prebehli 4 testovania, v ktorých sa menili rôzne parametre, od počtu trénovacích vzoriek po počet epoch.

Pri prvom testovaní sme zvolili, že použijeme 20 vzoriek herných ovládačov a 20 vzoriek hodiniek pri trénovaní. Počet epoch sme nastavili na 5, pričom každá mala 3 kroky. Výsledkom bolo, že po vytrénovaní, model úspešne rozpoznal 9 testovacích vzoriek z 15. To znamená, že dosiahol 60% úspešnosť.

Druhé testovanie prebehlo s rovnakými 20 trénovacími vzorkami, ale zdvihol sa počet epoch na 10. Taktiež sme zmenili počet krokov epochy na 5. Vďaka tejto zmene, počet úspešných rozpoznaní sa zdvihol na 12 z 15. To značí 80% úspešnosť modelu.

V treťom teste sme sa rozhodli zvýšiť nielen počet epoch, ale aj počet vzoriek pre trénovanie. Počet vzoriek sa zvýšil na 25, počet epoch sme zvýšili na 15. Naopak, rozhodli sme sa znovu znížiť počet krokov jednej epochy na 3. Toto testovanie dosiahlo výsledok, 14 úspešne rozpoznaných obrázkov z 15. Model teda dosiahol úspešnosť 93,33%.

Ďalšie testovanie bolo takmer identické ako predchádzajúce. Počet trénovacích vzoriek zostal na 25, taktiež aj počet 15 epoch. V tomto teste sme ale zvýšili počet krokov jednej epochy. To znamená, že model sa trénoval dlhšiu dobu, teda mal viac času na učenie. Vďaka tejto zmene, sa modelu podarilo dosiahnuť 100% úspešnosť, teda 15 úspešne rozpoznaných vzoriek.

V poslednom testovaní sme sa rozhodli zvýšiť počet trénovacích vzoriek, aj počet epoch. Modelu sme poskytli 30 vzoriek pre trénovanie a nastavili mu 20 epoch s krokom 5. Model síce musel rozpoznať viac vzoriek, no mal na to dostatok trénovacieho času. To znamená že znova dosiahol 100% úspešnosť.

Vďaka týmto testom sme preukázali, že pri trénovaní modelu umelej inteligencie, je potrebné poskytnúť dostatočné množstvo trénovacích vzoriek. Pri malom množstve trénovacích vzoriek sa môžu vyskytnúť nesprávne výsledky. To ale nie je jediným parametrom. Taktiež je potrebné nastaviť dostatočné množstvo epoch a krokov v rámci jednej epochy. Pri malom množstve epoch alebo jej krokov, sa model nestihne dostatočne dobre vytrénovať a vďaka tomu sa môžu vyskytnúť chybné rozpoznania obrázka. Správne nastavenie týchto parametrov ale závisí od náročnosti projektu. Pri malom množstve trénovacích vzoriek a testovacích vzoriek, postačí aj nastavenie menšieho množstva krokov a epoch. No pri projektoch s väčších množstvom trénovacích vzoriek a zložitejším testovaním, je potrebné dostatočne dobre nastaviť aj počet epoch a ich krokov. To poskytne modelu viac času, ktorý potrebuje na svoje vytrénovanie.

V tejto podkapitole sme navrhli a vytvorili aplikáciu, ktorá rozpoznáva vzorky herného ovládača a hodiniek. Model umelej inteligencie sme vytvorili pomocou nástroja TensorFlow od spoločnosti Google. Následne, sme aplikáciu naprogramovali v programovacom jazyku Python a opísali zdrojový kód. V poslednom rade sme pristúpili k testovaniu aplikácie. Testovanie a zmeny rôznych parametrov nám ukázali zaujímavé výsledky, zhrnuté v Tabuľka 3.

Bibliografia

Deyi, Li a Du, Yi. 2007. *Artificial Intelligence with Uncertainty.* s.l. : CRC Press, 2007. 1584889993.

European Commission. 2020. *White paper on artificial intelligence: A European approach to excellence and trust.* Brussels : European Commission, 2020.

Goodfellow, Ian , Bengio, Yoshua a Courville, Aaron. 2016. *Deep Learning.* s.l. : MIT Press, 2016.

Kriesel, David. 2007. A Brief Introduction to Neural Networks. [Online] 2007. [Dátum: 24. Jún 2021.] http://www.dkriesel.com/en/science/neural_networks.

Kvasnička, Vladimír, a iní. 1997. *Úvod do teórie neurónových sietí.* s.l. : IRIS, 1997.

Lane, Dale. 2018. Explaining artificial intelligence to high school students at #IBMAoTTHINK2018. *dalelane.co.uk.* [Online] 25. Apríl 2018. [Dátum: 1. Máj 2021.] https://dalelane.co.uk/blog/?m=201804.

—. 2017. Machine Learning for Kids event at Hursley. *dalelane.co.uk.* [Online] 2. August 2017. [Dátum: 25. Apríl 2021.] https://dalelane.co.uk/blog/?p=3537#more-3537.

McCaffrey, James. 2018. *Keras Succinctly.* s.l. : Syncfusion Inc., 2018.

Nielsen, Michael. 2015. *Neural Networks and Deep Learning.* s.l. : Determination Press, 2015.

OECD. 2019. *Recommendation of the Council on Artificial Intelligence.* s.l. : OECD/LEGAL/0449, 2019.

Sharma, Siddharth, Sharma, Simone a Athaiya, Anidhya. 2020. *ACTIVATION FUNCTIONS IN NEURAL NETWORKS.* [Dokument] s.l. : JEAST, 2020. 2455-2143.

Stahl, Bernd Carsten. 2021. *Artificial Intelligence for a Better Future.* s.l. : Springer - eBook, 2021. 3030699773.

Tian, Youhui. 2020. *Artificial Intelligence Image Recognition Method Based on Convolutional Neural Network Algorithm.* s.l. : IEEE Access, 2020.

Príloha A Zdrojový kód programu pre rozpoznávanie obrázkov pomocou nástroja TensorFlow

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
from tensorflow.keras.optimizers import RMSprop

train = ImageDataGenerator(rescale=1 / 255)
validation = ImageDataGenerator(rescale=1 / 255)

train_dataset = train.flow_from_directory('vision/basedata/training/',
                                          target_size=(200, 200),
                                          batch_size=3,
                                          class_mode="binary")

validation_dataset = validation.flow_from_directory('vision/basedata/validation/',
                                                    target_size=(200, 200),
                                                    batch_size=3,
                                                    class_mode="binary")

train_class = train_dataset.class_indices
print(train_class)

model = tf.keras.models.Sequential([tf.keras.layers.Conv2D(16, (3, 3), activation="relu",
input_shape=(200, 200, 3)),
                                    tf.keras.layers.MaxPool2D(2, 2),
                                    tf.keras.layers.Conv2D(32, (3, 3), activation="relu"),
                                    tf.keras.layers.MaxPool2D(2, 2),
                                    tf.keras.layers.Conv2D(64, (3, 3), activation="relu"),
                                    tf.keras.layers.MaxPool2D(2, 2),
                                    tf.keras.layers.Flatten(),
                                    tf.keras.layers.Dense(512, activation="relu"),
                                    tf.keras.layers.Dense(1, activation="sigmoid")
                                    ])

model.compile(loss="binary_crossentropy",
              optimizer=RMSprop(lr=0.001),
              metrics=["accuracy"])

model_fit = model.fit(train_dataset,
                      steps_per_epoch=3,
                      epochs=30,
                      validation_data=validation_dataset)
```

```
dir_path = 'vision/basedata/testing'

for i in os.listdir(dir_path):
    img = image.load_img(dir_path + "//" + i, target_size=(200, 200))
    plt.imshow(img)
    plt.show()

    X = image.img_to_array(img)
    X = np.expand_dims(X, axis=0)
    images = np.vstack([X])
    val = model.predict(images)
    if val == 0:
        print("Gamepad")
    else:
        print("Hodinky")
```