VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PODPORA DESKOVÉ HRY NEMESIS NA MOBILNÍM TELEFONU S OS ANDROID

SUPPORTING BOARD GAME NEMESIS ON ANDROID MOBILE PHONE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MIROSLAV ŠTĚPÁNEK**
AUTHOR

VEDOUCÍ PRÁCE — **doc. RNDr. PAVEL SMRŽ, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Štěpánek Miroslav**
Program: Informační technologie
Název: **Podpora deskové hry Nemesis na mobilním telefonu s OS Android**
Supporting Board Game Nemesis on Android Mobile Phone
Kategorie: Počítačová grafika

Zadání:

1. Seznamte se s existujícími aplikacemi pro podporu různých fází deskových her a mechanismy přirozeného zapojení mobilních zařízení do "fyzického světa" her.
2. Analyzujte potřeby hráčů deskové hry Nemesis s cílem určení oblastí, u nichž by přenesení konkrétních činností na obrazovky mobilních telefonů, případně i podpora hraní, přinesly největší užitech.
3. Navrhněte a implementujte systém pro usnadnění hraní a vyhodnocování hry Nemesis na základě rozpoznávání a zpracování obrazu na mobilním telefonu.
4. Vyhodnoťte realizované řešení v adekvátní uživatelské studii v reálném prostředí při hraní hry.
5. Vytvořte stručný plakát prezentující práci, její cíle a výsledky.

Literatura:

- dle doporučení vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- funkční prototyp řešení

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

Vedoucí práce: **Smrž Pavel, doc. RNDr., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 1. listopadu 2021

Abstrakt

Cílem této práce je vytvořit mobilní aplikaci k deskové hře Nemesis určenou na systém Android, která umožní uživateli zjistit informace o herních komponentech při hře. Práce se skládá ze dvou hlavních částí, první je model vytvořený za pomoci knihovny Tensorflow, který zajišťuje detekci těchto komponent. Druhou je pak samotná aplikace, která dostává výsledky od dříve zmíněného modelu a zobrazuje výsledné informace uživateli. Toto usnadňuje uživateli hru a pomáhá ji i urychlit. Výsledný systém je možné modifikovat, tak aby byla aplikace využitelná i pro jiné hry.

Abstract

The aim of this thesis is to create a mobile application for the board game Nemesis designed for the Android system, which will allow the user to find out information about the game components during the game. The solution consists of two main parts the first is a model created with the help of the Tensorflow library, which is responsible for the detection of these components. The second is the application itself, which receives results from the model and displays the resulting information to the user. This makes the game easier for the user and helps to speed it up. The resulting system can be modified so that the application can be used for other games.

Klíčová slova

Konvoluční neuronové sítě, Tensorflow, Tensorflow lite, Android, Klasifikace obrazu, Desková hra Nemesis

Keywords

Convolutional neural networks, Tensorflow, Tensorflow lite, Android, Image classification, Nemesis board game

Citace

ŠTĚPÁNEK, Miroslav. *Podpora deskové hry Nemesis na mobilním telefonu s OS Android.* Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. RNDr. Pavel Smrž, Ph.D.

Podpora deskové hry Nemesis na mobilním telefonu s OS Android

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. RNDr. Pavla Smrže, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Miroslav Štěpánek
9. května 2022

Poděkování

Rád bych poděkoval doc. RNDr. Pavlu Smržovi, Ph.D. za vedení práce, cenné rady a věcné připomínky, které mi pomohly při tvorbě této práce.

Obsah

Kapitola 1

Úvod

Klasické deskové hry se i přes technologický pokrok a snahu o jejich digitalizaci stále těší velké oblibě. Důkazem toho jsou například lokálně pořádané turnaje ve hrách, jako například Scythe nebo Carcassonne. Dále je možné prohlédnout statistiky vlastnictví deskových her na stránce BoardGameGeek[1], která je jedním z hlavních center zdrojů informací pro všechny hráče. Zde se asi zcela jistě nenachází celkový počet všech vlastněných her, ale i přesto tato čísla nejsou zanedbatelná. Aktuálně je nejpopulárnější desková hra Pandemic[2], která má zaregistrovaných přibližně 170 tisíc vlastněných kopií.

Výhodou deskových her je skutečnost, že je může hrát téměř kdokoliv. Ovšem existují různé úrovně náročnosti a některé tituly tedy představují daleko komplexnější situace. Toto bývá většinou doplněno velkým počtem různých dílků a karet. Pravidla těchto titulů se pak obecně pohybují v rámci desítek stran, což pro začínající hráče není vůbec optimální. Pro ty to znamená, že pokud si danou hru chtějí zahrát, musí věnovat značné úsilí studiu pravidel. Pokud eventuálně k samotné hře dojde, nebude pravděpodobně herní zážitek zcela optimální a standardní herní doba bude značně prodloužena. Tento prospekt pak jistě odradí hráče, kterým se za tímto účelem nechce toto úsilí vydávat.

Herní doba těchto složitějších her je většinou určena podle počtu přítomných hráčů. Může se tedy jednat o pár hodin nebo v extrémních případech i o několik dní, pokud není možné hru dohrát za jedno sezení. Ať už jde tedy o herní zážitek, první dojem nebo dobu hraní, je zbytečné prodlužování nežádoucí. Vše zmíněné je tedy jistou motivací pro tvůrce hry, aby hráči co nejvíce zjednodušili její porozumění a přišli s řešením, jak pravidla efektivně vysvětlit.

Řešeních tohoto problému již existuje mnoho, některé hry mají pro tento účel vytvořené zmenšeniny pravidel, kde se vyskytují nejčastěji využívaná pravidla. U jiných si takovou pomoc musí hráč vytvořit sám předem nebo sáhnout po zdrojích vytvořených komunitou. Pro některé hry jsou vytvořeny i mobilní aplikace, které pomáhají s pravidly nebo v jiném aspektu hry, například při počítání bodů.

Tato řešení mají své výhody i nevýhody, například pomocné karty nebo sumarizace návodů zabírají místo navíc. U některých her toto může být problémem, zejména u menších stolů nebo jiných hracích ploch. Výhodou mobilní aplikace je, že má stejné nebo i větší benefity a zabírá daleko méně místa než ostatní řešení. Navíc chytré mobilní telefony v dnešní době vlastní velká část populace, a tak jedinými limitujícími faktory jsou v tomto případě samotná existence aplikace a operační systém, pro který je vytvořena. S ohledem na to,

[1] https://boardgamegeek.com/browse/boardgame?sort=numowned&sortdir=desc
[2] https://boardgamegeek.com/boardgame/30549/pandemic

že je v rámci mobilních telefonů nejvíce zastoupen operační systém Android[3], bude se tato práce věnovat právě jemu.

V následující kapitole 2 bude popsána samotná desková hra Nemesis, na kterou se bude práce zaměřovat. Dále bude v kapitole 3 popsána problematika zpracování snímků za pomoci neuronových sítí a přizpůsobení výsledných modelů pro mobilní zařízení. Kapitola 4 se bude věnovat datovým sadám, jejich generování a rozdělení do dílčích sad. Popsán bude i návrh celého systému v kapitole 5. Struktura modelu, který zajišťuje rozpoznání komponent ve snímku, bude popsána v kapitole 6. Kapitola 7 je věnována aplikaci pro zařízení Android, její struktuře a způsobu vyhodnocování výsledků. V poslední kapitole 8 se nachází analýza výsledků, popsání vzniklých chyb a jejich příčin.

[3] https://www.businessofapps.com/data/android-statistics/

Kapitola 2

Desková hra Nemesis

Stolní hra Nemesis je zasazena do sci-fi prostředí celosvětově známého filmu Alien[1] (Vetřelec) z roku 1979. Byla vytvořena polským autorem Adamem Kwapińskim, po velice úspěšném Kickstarteru v roce 2018. Požadovaná suma peněz byla během několika dnů přesažena a díky tomu vyšlo několik rozšíření během minulých let na současných třináct. Hra byla nominována a získala několik ocenění, což je odraženo i v uživatelském hodnocení, které dosahuje 8.4 z 10 bodů a řadí se tím aktuálně na sedmnáctou příčku v žebříčku všech deskových her. Na následujícím obrázku 2.1 je možné vidět, jak tato hra vypadá.

Obrázek 2.1: Herní situace z deskové hry Nemesis. Převzato z: https://boardgamegeek.com/image/4405998/nemesis

2.1 Popis hry

Hráči se probouzí z kryospánku na lodi Nemesis, která se, jak záhy zjistí, hemží mimozemskými vetřelci. Každý z hráčů hraje za rozdílný charakter, ať už jde o vojáka, který má jako jeden z mála vybavení, aby se mohl vetřelcům postavit, mechanika, který se snaží udržet loď pohromadě opravováním místností nebo vědce, který se snaží odhalit slabiny vetřelců a ulehčit tím tak všem život. Každý z těchto charakterů se tedy velice liší jak ve svém vybavení, které mají k dispozici, tak ve svých schopnostech. Pro každého hráče je cílem přežít, ale také splnit jeden ze svých dvou tajných úkolů, které každý dostane na začátku hry.

[1] https://www.csfd.cz/film/8265-vetrelec/prehled/

Jedná se o částečně kooperativní hru, hráči tedy mohou spolupracovat a společně se snažit překonat různé překážky nebo si naopak škodit a snažit se jeden druhého dostat do problémů. Hra je designovaná tak, aby co nejvíce odpovídala realitě, v jaké se charaktery ocitly i ve filmu a neodpouští tedy sebemenší chyby. Navíc, ať už se hráči nachází v sebelepší situaci, nemusí být dosaženo vítězství ani jedním z nich.

2.2 Popis herních komponent

Hra se skládá z velkého počtu komponent, které můžeme rozdělit do těchto kategorií:

- Karty
- Místnosti
- Značky a žetony
- Figurky
- Ostatní

Karet je ve hře velké množství a zároveň se většina z nich liší, ať už jsou specifické pro daný charakter nebo například popisují zranění, které charakter utržil. Popisy karet jsou však dostatečně podrobné a omezení jejich použití jsou dobře zřetelné, není tedy potřeba je nikterak řešit. Toto platí i pro komponenty z kategorie ostatní.

V případě hexagonů místností je ale situace jiná. Ty jsou sice lehce rozlišitelné a obsahují stručný popis, jakou funkci zajišťují, avšak tento je pro plné pochopení, k čemu daná místnost slouží, nedostatečný. Toto bude tedy jedna z cílových kategorií aplikace.

Značky a žetony budou také rozeznávány v rámci aplikace. Tyto komponenty nemají žádné popisky, hráči si tedy musí pamatovat jejich využití. Zároveň také mohou ovlivňovat stav místností a počet akcí, které má hráč k dispozici.

Obrázek 2.2: Zahrnuté kategorie – Hexagon místnosti, žeton člena posádky a značka ohně.

Figurky hráčů a vetřelců jsou dobře rozeznatelné a nejeví se jako problém. Případů, kde by se dala pomoc aplikace využít sice několik existuje, ale zabíhala už do úplných základů hry a lepším řešením by v tomto případě byla spíše konzultace fyzických pravidel.

Figurky také dosahují výšky několika centimetrů, a hlavně, v případě vetřelců, jsou schopni zakrýt velkou část místnosti a zároveň i některé menší značky. Toto by představovalo pro aplikaci situaci, kde by nemohla rozpoznat některé komponenty. Ze zmíněných důvodů tedy nebudou figurky zahrnuty. Všechny zahrnuté kategorie jsou vyobrazeny na snímku 2.2.

2.3 Existující pomocná aplikace

Pro hru Nemesis již existuje oficiální pomocná mobilní aplikace, která je dostupná jak na zařízeních Android[2], tak IOS[3]. Avšak zaměřuje se na automatizaci některých částí hry, aby tímto způsobem urychlila hru a ušetřila místo na stole. Vývoj aplikace je pravděpodobně ukončen, protože poslední verze byla vydána v lednu 2021. Zřejmě tedy nehrozí scénář, že by byla vydána verze, která by se zabývala stejnou oblastí jako tato práce. Aktuálně se jedná o jedinou mobilní aplikaci, která se hrou zabývá.

[2] https://play.google.com/store/apps/details?id=com.AwakenRealms.Nemesis
[3] https://apps.apple.com/us/app/id1517768907

Kapitola 3

Zpracování obrazu pomocí neuronových sítí

Zdrojem informací pro tuto a následující kapitolu byly: [17, 11, 6].

Rozpoznání a kategorizování různých objektů v rámci běžného života se pro člověka může jevit jako velice triviální úloha, nad kterou většinou nemusí ani příliš přemýšlet. Lidé totiž mají v hlavě superpočítač v podobě mozku, který se postupně v průběhu několika stovek miliónů let pro tuto činnost adaptoval. Mozek obsahuje milióny neuronů, které jsou mezi sebou různě propojené a které mimo jiné dovolují člověku zpracovat obraz a identifikovat v něm nacházející se objekty. Pokud člověk znalostí o daném problému ještě nedisponuje, pak pro její nabytí ve většině případů stačí poměrně malé množství instancí, ve kterých se s tímto problémem setká. Zde mohou sloužit jako příklad právě deskové hry. Pokud hráč danou hru nezná, nejspíše je první hru poněkud zmatený, ve druhé začne využívat některé základní strategie a v následujících hrách už pravděpodobně bude podobně zběhlý jako ostatní hráči.

Avšak v případě programů je tato identifikace kategorií, nacházejících se v obraze, velice odlišná. Obraz je totiž vnímán jako matice pixelů. Pro účel jejího zpracování je sice možné určit různá pravidla, aby se požadovaného výsledku dosáhlo, například určit, jaký vzhled daná komponenta má nebo jakého je tvaru. Takový přístup však není možné využít, pokud je daná komponenta překrytá nebo pokud je ovlivněna jinými rušivými jevy. Tento problém pomáhají řešit neuronové sítě, které jsou jedním z modelů strojového učení. Ty mají schopnost naučit se různé rysy dat, které jim pomohou při rozpoznávání daných objektů. Zmíněný proces učení pak probíhá v epochách, kdy je v ideálním případě program po každé dokončené epoše úspěšnější, podobně jako člověk v případě deskových her.

Za zmínku stojí také již delší dobu probíhající vývoj hybridních her s cílem propojit deskové a počítačové hry. Protože se mladší generace méně socializují a nahrazují skupinové hraní deskových her spíše osamoceným hraním počítačových her, je tato kombinace možným řešením tohoto problému. Důležitým aspektem je to, že pokud některý z hráčů rozšířené reality využít nechce, tak nemusí. Hlavním účelem je udělat hru zajímavější pro větší počet hráčů. Tento efekt byl testován na hře Small Star Empires a ukázal se jako úspěšný [19].

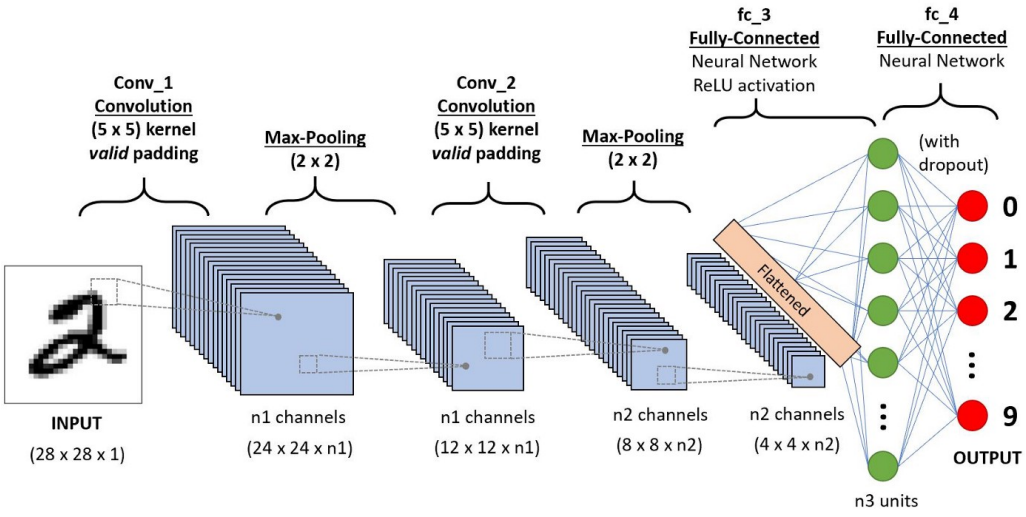
Dalšími příklady mohou být například rozšířená realita pro hru Monopoly [16], která umožňuje přidávat virtuální elementy a transformovat figurky hráčů nebo hra False Prophets, ve které je kombinována projekce veřejných informací a zobrazení soukromých informací na mobilních zařízeních. Cílem je opět maximalizovat sociální interakci a udělat hru zajímavější [15].

Možné je zmínit i další existující mobilní aplikace, které pomocí neuronových sítí a technik zpracování obrazu počítačového vidění, umožňují uživateli například načíst rozložení čísel ve hře Sudoku. Toto rozložení je načítáno přímo z novin a následně umožňuje uživateli tuto hru dohrát na mobilním zařízení [14].

3.1 Konvoluční neuronové sítě

Tento typ neuronových sítí se specializuje na zpracování dat, které jsou rozloženy v mřížce, v tomto případě jde tedy o obrázky. Jde o efektivnější a úspěšnější přístup než u předchozích neuronových sítí, které obsahovaly pouze plně propojené vrstvy. Těchto výsledků dosahuje za využití účinnějších konvolučních a pooling (shromažďujících) vrstev.

Konvoluční vrstvy využívají lokálních receptivních polí, které podle velikosti kernelu propojují určitou oblast, například 3x3, do pouhého jednoho neuronu. Tímto redukují velké množství propojení v rámci sítě. Dále se vrstvy skládají z určitého počtu map, podle zvoleného počtu filtrů. Jedná se vlastně o různé verze daného snímku. Příkladem může být klasický barevný snímek, který se skládá ze tří barevných map. Na všechny tyto verze je následně aplikován určitý filtr, který z obrázku extrahuje jeden specifický typ rysu. Může se jednat například o detekci svislých hran. Výhodou je, že pro filtr není důležité, kde přesně ve snímku se tyto rysy nachází. Znamená to tedy, že pokud se objekt v obrázku pohne na jiné místo, budou stále detekovány stejné rysy a budou vyvozeny stejné výsledky.



Obrázek 3.1: Příklad typické architektury konvoluční neuronové sítě. Převzato z: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Pooling vrstvy pak tyto informace o rysech zjednodušují a v případě použité maxpooling vrstvy jsou použity jen ty nejdůležitější. Toto, zároveň při běžném použití s hodnotou kroku dva, zmenšuje rozlišení obrázku na polovinu, čímž také pomáhá redukovat počet nutných propojení.

Celá tato architektura je obecně zakončena určitým počtem plně propojených vrstev, včetně vrstvy výstupní, podobně jako na tomto obrázku 3.1. Zde je přítomné stále velké

množství propojení, avšak je daleko menší, než dříve a to hlavně díky zmíněným pooling vrstvám.

3.2 Předzpracování dat

Pro optimální funkčnost modelu je potřeba provést několik operací se vstupními daty. Vzhledem k tomu, že model pracuje se snímky dané velikosti, je nutné všechna tato vstupní data škálovat na požadovanou velikost, pokud se v tomto rozlišení již nenachází. Další nutnou operací je normalizace vstupních snímků maximální hodnotou pixelu 255. Toto normalizuje hodnoty pixelů do rozmezí nula až jedna. Pokud by tento krok nebyl proveden, mohlo by dojít k tomu, že by se model naučil zbytečně velké váhy, což by vedlo k nízké stabilitě a horším výsledkům [9]. Touto akcí se zároveň předejde možnosti výskytu explodujícího gradientu, který by trénování zpomalil nebo zcela znemožnil [18].

3.3 Aktivační funkce

V rámci modelu je konvolučním a plně propojeným vrstvám přiřazena určitá aktivační funkce. Tato, jak již název napovídá, určuje, kdy je neuron aktivovaný a jakých hodnot výstup dosahuje. Vstupní hodnoty pro tyto funkce jsou získány z předešlé vrstvy a upraveny váhou, která je na propojení přítomná. Dříve byly pro tento účel využívány funkce `sigmoid` a později `tanh`. Jak se ale následně ukázalo, nebyly vhodné pro použití v hlubších sítích, kvůli problému mizícího gradientu a jejich snadné saturaci. Z těchto důvodů je dnes nahradila velice často využívaná funkce `ReLU` nebo některá z jejich upravených variant, které řeší některé z jejích nedostatků [8]. I přesto, že tato funkce není naprosto dokonalá, jak naznačují různé verze, pomáhá při tvoření hlubších neuronových sítí.

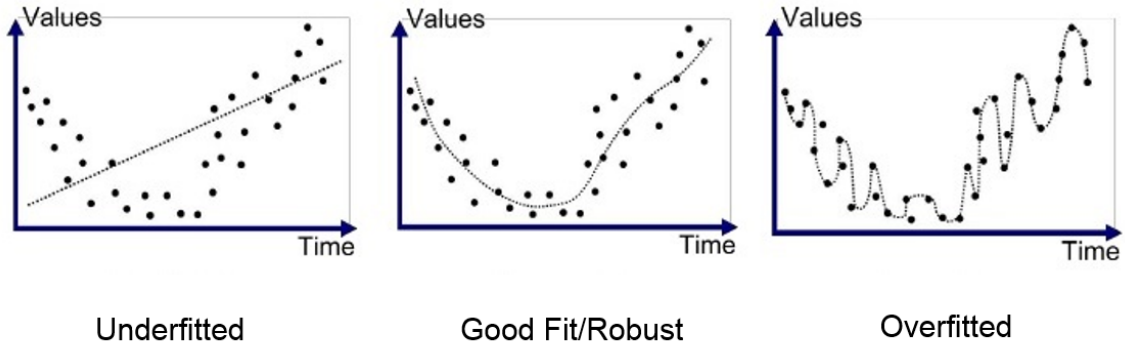
Výstupní vrstva je v tomto ohledu velice specifická, protože určuje výsledky, ke kterým model dospěl. Počet neuronů, který tato vrstva obsahuje, je rozdílný, podle specifického účelu. Zde bude brán v potaz přístup, kdy je pro každou z kategorií určen jeden výstupní neuron. V tomto případě hraje velice důležitou roli, k jakému účelu je model vytvořen.

Pokud má určit, zda se ve snímku nachází právě jedna kategorie z více jak dvou možných, bude se jednat o klasifikaci více tříd (multiclass classification). Pro tento účel se využije funkce `softmax`. Ta přiřadí každému z výstupních neuronů pravděpodobnost, že se ve snímku nachází. Pravděpodobnost všech kategorií se v součtu rovná jedné, což není použitelné pro případ, kdy se požaduje, aby byla detekována více než jedna kategorie.

Pro případ, kdy je třeba detekovat více kategorií v rámci jednoho snímku, je třeba využít klasifikace s více značkami (multilabel classification). Pro tento účel je využito funkce `sigmoid` [10]. Při jejím použití je každému neuronu přiřazena vzájemně nezávislá hodnota, zda se ve snímku nachází či nikoliv.

3.4 Problémy generalizace modelu

Hlavním cílem trénování modelu je, aby byl schopný pracovat s daty, která doposud neviděl a dosahoval správných výsledků. Této schopnosti modelu, jak dobře je schopný pracovat s novými daty, se říká generalizace. Pokud model dosahuje špatných výsledků, je možné, že dochází k jednomu ze dvou fenoménů, které generalizaci mohou ovlivnit. Těmito situacemi jsou přetrénování (overfitting) nebo neschopnost naučení se (underfitting). Porovnání následného chování modelu v důsledku těchto chyb je možné vidět na obrázku 3.2.



Obrázek 3.2: Porovnání funkcionality modelů s vhodnou a nevhodnou kapacitou. Převzato z: https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76

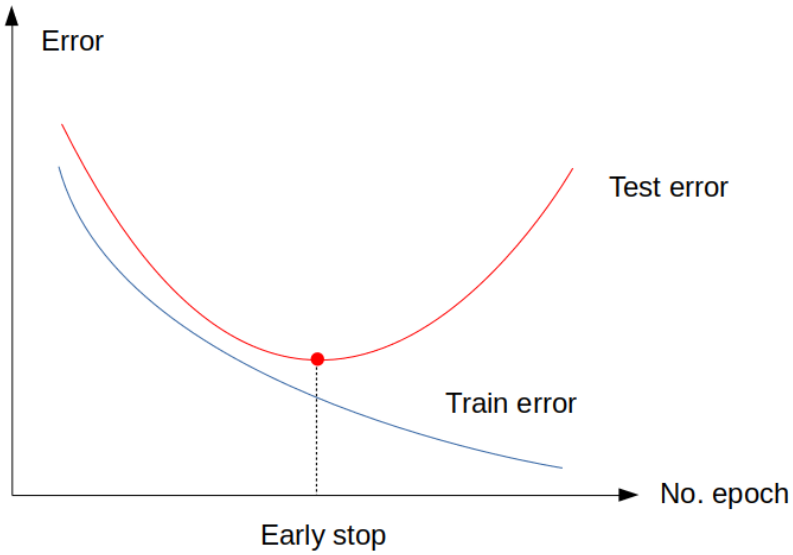
Výskyt přetrénování je možné odhalit pomocí validační sady, na které má model znatelně horší úspěšnost než na té trénovací. Toto znamená, že je kapacita modelu příliš velká a je schopný si zapamatovat rysy trénovací sady, které se však nedají použít při validaci. Neschopnost naučení se je pak úplným opakem, model má v této situaci příliš malou kapacitu a není schopný zachytit rysy z trénovací sady.

Řešení těchto problémů se nabízí mnoho, může například stačit změna hyperparametrů modelu, tak aby kapacita modelu odpovídala nebo se blížila té požadované. V některých případech však toto nestačí a je třeba změnit samotnou architekturu modelu. Cílem tohoto úsilí je také dosáhnutí stejné kapacity, jako je pro řešený problém potřeba, protože jen tehdy bude model dosahovat nejlepších možných výsledků.

Jako určité zmírnění efektu přetrénování je možné v rámci modelu využít regularizace v podobě dropout vrstvy. Ta při každé trénovací epoše zanedbá určité procento vstupů, aby pro model nebylo možné zapamatovat si všechny rysy dat. Alternativně je pro tento účel možné využít L2 a L1 regularizace, které omezí méně důležité rysy dat, aby se je model neučil. Při používání těchto regularizačních metod je však důležité zvolit vhodné hodnoty, protože v opačném případě může dojít k tomu, že se model nebude schopen nic naučit.

Další příčinou přetrénování může být příliš velký počet trénovacích epoch, kdy model dosahuje stále lepších výsledků na trénovací sadě, ale čím dál horších výsledků na validační sadě. Důvodem pro tento trend je skutečnost, že si model zapamatovává zbytečné rysy z trénovacích dat, které s daným problémem přímo nesouvisí a nejsou přítomné ve validačních datech.

Častým řešením je v tomto případě využití praktiky, které se říká early stopping (brzké zastavení trénování). V průběhu trénování se ukládají verze modelu, při kterých dosahoval nejlepších výsledků. Pokud po určitém počtu proběhlých epoch model nezlepší nejlepší výsledek, je trénování zastaveno a je použita verze, která dosahovala nejlepšího výsledku. Graf ukazující efekt této techniky je možné vidět na obrázku 3.3.



Obrázek 3.3: Efekt early stoppingu na výsledek trénování. Po dosažení zvýrazněného bodu je trénování zastaveno a je použit nejlepší dosažený výsledek modelu. Pokud by toto nebylo provedeno, došlo by k přetrénování, jak je vidět na zhoršující se červené křivce. Převzato z: https://guopai.github.io/ml-blog18.html

I přes všechna možná řešení těchto problémů, které mohou nastat, je nutné pamatovat, že nejjednodušším řešením může občas být úprava datové sady. Problémy mohou být často způsobené nějakým nedopatřením, a proto pokud model nepracuje podle očekávání, je dobré provést kontrolu, zda sada odpovídá pravidlům popsaným v kapitole 4. Pokud žádný takový problém neexistuje, ani tak nemusí být větší množství dat ke škodě, protože větší množství relevantních dat může modelu jen pomoci. Zároveň je však nutné dbát na časové požadavky, které jsou od procesu trénování očekávány.

3.5 Vyhodnocování úspěšnosti

Při trénování se model snaží dosáhnout co nejlepšího výsledku, tento výsledek se nazývá ztráta a je získán ztrátovou funkcí. Ta následně určuje, jak moc se model ve svých předpovědích mýlil. Nejlepším výsledkem je v tomto případě její minimalizování. Ztrátové funkce se vybírají podle toho, k jakému účelu má model sloužit. Například pro účely klasifikace obrázku jako jedné kategorie, kde je možné ho zařadit do více než dvou, je možné využít kategorickou ztrátovou funkci. Toto je často spojeno s použitím výstupní vrstvy, která obsahuje aktivační funkcí `softmax`, pro zisk pravděpodobnosti výskytu.

Pokud je však potřeba rozeznat těchto kategorií více v rámci jednoho snímku, je daleko vhodnější užití binární ztrátové funkce. Zde je využívána funkce `sigmoid` a model určuje, zda se kategorie v obrázku nachází či nikoliv. Pro tento případ je však obtížné určit úspěšnost modelu. Výsledky trénování mohou vypadat velice dobře, avšak může například docházet k situacím, kdy má model určit jednu ze tří kategorií jako přítomnou a namísto toho určí všechny. Výsledek bude i přesto velice pozitivní. Namísto standardních metrik je zde nutné sledovat přesnost (precision) a úplnost (recall) [13]. Přesnost určuje, kolik z na-

lezených kategorií bylo určeno správně a úplnost vyhodnocuje, kolik z hledaných kategorií bylo nalezeno.

3.6 Optimalizace modelů pro mobilní zařízení

Modely neuronových sítí dosahují vysoké výpočetní náročnosti, zabírají velké množství paměti a spotřebovávají vyšší množství energie. Existuje tedy několik metod pro zlepšení těchto aspektů na mobilních a okrajových zařízeních, které nedisponují dostatečně velkou výpočetní silou nebo výdrží baterie. Pro tento účel jsou využívány techniky kvantizace (quantization), redukce zbytečných propojení rámci modelu (pruning) a shlukování vah (weight clustering) [1].

Kvantizaci lze rozdělit na dva typy, quantization aware training (trénování s ohledem na kvantizaci) a post-training quantization (kvantizace po dokončení trénování). Při obou přístupech je vytrénovaný model kvantizován a datový typ parametrů je změněn z Float na Int. Přesnost těchto parametrů je tedy výrazně snížena. V případě quantization aware training je tato akce následována přetrénováním modelu v tomto stavu, aby získal zpět svoji správnost. Tato akce není provedena v rámci jednodušší post-training quantization, čímž může docházet k horším výsledkům.

V obou případech může optimalizovaný model dosahovat až čtvrtinové velikosti s několikanásobným zrychlením oproti tomu neoptimalizovanému. Obecně dojde k určité ztrátě správnosti modelu, která bývá pro každý model specifická. Toto snížení je však většinou v zanedbatelných mezích, jak je vidět na následujícím obrázku 3.4. Zároveň je také vidět lepší výkon a správnost modelů, které byly trénovány s ohledem na kvantizaci [3, 2].

Model	Top-1 Accuracy (Original)	Top-1 Accuracy (Post Training Quantized)	Top-1 Accuracy (Quantization Aware Training)	Latency (Original) (ms)	Latency (Post Training Quantized) (ms)	Latency (Quantization Aware Training) (ms)	Size (Original) (MB)	Size (Optimized) (MB)
Mobilenet-v1-1-224	0.709	0.657	0.70	124	112	64	16.9	4.3
Mobilenet-v2-1-224	0.719	0.637	0.709	89	98	54	14	3.6
Inception_v3	0.78	0.772	0.775	1130	845	543	95.7	23.9
Resnet_v2_101	0.770	0.768	N/A	3973	2868	N/A	178.3	44.9

Obrázek 3.4: Porovnání optimalizovaných a neoptimalizovaných modelů při využití knihovny TensorFlow. Z tabulky je jasně vidět lepší výsledný výkon a správnost modelu při využití quantization aware training, oproti jednoduššímu post-training quantization. Převzato z: https://www.tensorflow.org/lite/performance/model_optimization

Dalším způsobem, jak zmenšit velikost modelu a zrychlit jeho inferenci, je redukce méně významných propojení v rámci neuronové sítě. Tohoto je možné dosáhnout redukováním některých vah na nulovou hodnotu nebo úplným odstraněním některých neuronů. Obecně jde však o odstraňování nepotřebných parametrů a uzlů, které nemají vliv na výslednou správnost modelu. Váhy, které jsou vynulovány, bývají většinou už blízko samotné nule, a tak nemají vlastně žádný efekt na výkon modelu. Podobně je tomu u odstraněných neu-

ronů, které v tomto případě nabývají také hodnot blízkých nule, čímž nemají vliv na jeho funkcionalitu [7, 4].

Pro zlepšení komprese modelu je také možné využít shlukování vah. Za tímto účelem se využívá k-means shlukování. Toto zajišťuje, že všechny váhy, které spadají do stejného shluku, budou sdílet stejnou váhu. Aby bylo tohoto výsledku dosaženo, je třeba určit centrální body, kolem kterých se budou tyto váhy shlukovat. Je možné využít náhodné inicializace, lineárního rozdělení nebo například rozdělení podle četnosti výskytu dané váhy [12, 5].

Kapitola 4

Datová sada

Pro trénování jakéhokoliv modelu jsou potřeba data popisující daný problém, ta jsou poté shromažďována do datových sad. Při každé tvorbě nové sady, která má být využita při trénování nebo testování modelu, je potřeba zajistit těchto několik věcí:

- Data jsou relevantní k problému, který se model snaží naučit.
- Všechna data jsou správně kategorizovaná.
- Dat je dostatečné množství a nejsou jednotvárná.

Pokud některé z těchto kritérií není splněno, nebo je splněno jen částečně, může to mít za následek hůře pracující model. Data sice nemusí být vyvážená, což je situace, kdy je snímků jedné kategorie větší množství než ostatních. Zde je pak ale nutné přiřadit kategoriím s menším výskytem při trénování větší váhu, aby měla na proces učení větší efekt než data, kterých je větší počet.

4.1 Generování a augmentace dat

Pokud je cílem vytvoření robustního modelu, je potřeba velké množství relevantních dat. Čím více jich bude k dispozici, tím lepších výsledků bude pravděpodobně dosaženo. Získání většího počtu dat je možné více způsoby, například využitím některých hotových datových sad ze služeb jako ImageNet[1] nebo CIFAR[2], kde je k dispozici velké množství už kategorizovaných snímků. Pokud je ale řešený problém specifičtější a není k dispozici žádný velký zdroj dat, je vhodné provést augmentaci stávajících dat, popřípadě pokud je to možné, nová data generovat.

Augmentace dat je použitelná ve většině případů, jedná se o určité pozměnění originálních dat. Jde například o změnu jasu nebo posun obrázku, čímž bude jeho část oříznuta. Změna jasu je velice důležitá pro správnou funkcionalitu modelu za různých světelných podmínek, pokud nejsou k dispozici relevantní data. Tímto způsobem je možné získat velké množství alternativních dat pro daný model, avšak je nutné augmentaci provádět v rozumných mezích. V opačném případě mohou být data transformována na nepoužitelný šum, kdy získání jakýchkoliv rysů není možné.

[1] https://www.image-net.org/
[2] https://www.cs.toronto.edu/~kriz/cifar.html

4.2 Rozdělení datové sady

Pro účely trénování modelu je vhodné datovou sadu rozdělit. Zde se ve využívaných přístupech vyskytují rozdíly, do kolika sad je vhodné data dělit. Ve všech případech existuje hlavní trénovací sada. Zde přítomná data jsou využita modelem pro naučení jejich hlavních rysů. Tuto sadu je velice vhodné augmentovat a při každé epoše zamíchat, aby bylo dosaženo lepší generalizace. Některé přístupy obsahují fyzicky pouze tuto sadu, která je pak v rámci trénování rozdělena na trénovací a validační.

Validační sada je tedy také přítomna ve všech přístupech, i když ne vždy je získána stejným způsobem. Tato sada slouží, jak bylo dříve zmíněno, jako kontrola, zda model správně generalizuje a nedochází například k přetrénování. K této sadě nemá model přístup, a tak slouží čistě jen pro jeho zhodnocení. Problém tohoto přístupu, kde chybí testovací datová sada, ale tkví ve faktu, že tvůrce modelu, při snaze zlepšit výkon modelu na validační sadě, mění jeho parametry, aby logicky dosáhl co nejlepších výsledků. To však může být problematické ve chvíli, kdy dojde v důsledku těchto změn k přílišnému přizpůsobení parametrů validační sadě. Člověk tedy provede vlastní přetrénování na validační sadě. Aby se tomuto předešlo, je doporučené mít ještě dodatečnou testovací sadu.

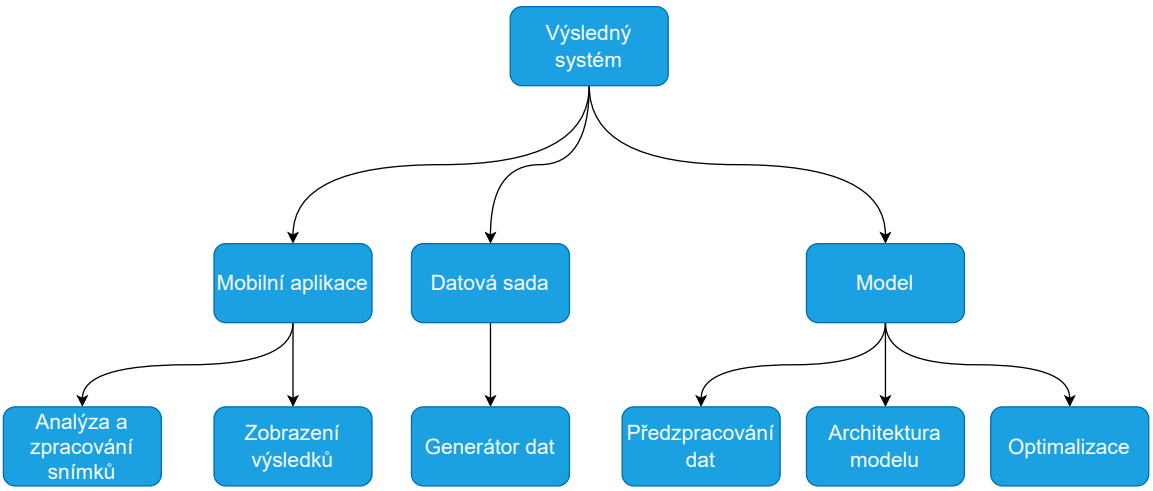
Jedná se tedy vlastně o kontrolu kontroly, avšak pokud se neprovede, nemusí to mít dobrý dopad na úspěšnost modelu. Výjimkou je v tomto ohledu validační sada, na které je provedena augmentace. V tomto případě jsou data pro každou epochu pokaždé trochu jiná a není tedy zdaleka tak snadné jim přizpůsobit parametry modelu.

První zmíněný přístup má tu výhodu, že při každém novém trénování je trénovací a validační, popřípadě i testovací sada jiná. Zároveň je také velká pravděpodobnost, že bude z většiny obsahovat jiná data, vzhledem k hojně využívanému rozdělení sad v poměru 80:20 pro trénovací a validační nebo 80:10:10 pro všechny tři. Stále je však třeba mít na paměti, že pokud existuje šance výskytu stejných nebo podobných dat v trénovací a testovací sadě současně, výsledky trénování nemusí být směrodatné. Z tohoto důvodu může být manuální rozdělení dat do sad lepší volbou.

Kapitola 5

Návrh systému

Tato část se bude zabývat návrhem všech dílčích částí výsledného systému a nástrojů, které pro tento účel bude možné využít. Dále bude třeba zajistit, aby tento systém uživateli v co největší míře ulehčil a urychlil orientaci při hře Nemesis. Výsledný systém se nakonec bude skládat ze tří hlavních částí, jak je možné vidět na schématu 5.1. Těmito částmi jsou aplikace, která bude zajišťovat interakci s uživatelem, model, určený k rozpoznávání vybraných komponent a datová sada, ze které bude model čerpat při trénovacím procesu.



Obrázek 5.1: Schéma navrhovaného systému.

5.1 Model pro detekci komponent hry

V první části bude třeba vytvořit model, který bude schopen kategorizovat vybrané komponenty hry. Tohoto cíle bude dosaženo za využití konvolučních neuronových sítí, které jsou pro tento účel velice vhodné. Aby se předešlo zbytečným komplikacím, budou vstupní data předzpracována, aby odpovídala rozměrům, které model akceptuje a budou normalizována. V rámci modelu budou použity dříve zmíněné aktivační funkce. Pokud bude v rámci procesu trénování docházet k přetrénování nebo naopak k neschopnosti modelu naučit se klíčové rysy dat, bude využito dříve zmíněných technik, aby se těmto chybám předešlo. Výsledný model bude testován a validován za pomoci vhodných datových sad, které budou odpovídat specifikacím z předešlé kapitoly. Takto vytvořený model bude optimalizován, aby

bylo možné jeho co nejefektivnější využití na mobilních zařízeních. Zde bude hlavním cílem zajistit co největší výslednou rychlost modelu s co nejmenší ztrátou.

Vývoj tohoto modelu bude probíhat v programovacím jazyce Python, za využití knihoven Tensorflow a Keras. Tyto knihovny v kombinaci s jazykem Python umožňují rychlou a efektivní tvorbu a úpravu modelů. Jedná se tedy o vhodné nástroje pro iterativní vývoj. Výhodou těchto knihoven je také to, že stejně jako operační systém Android, patří pod firmu Google, což zajišťuje přívětivější možnosti implementace. Zároveň je také nutné využít knihovny Tensorflow lite, která se specializuje na konverzi a optimalizaci modelů pro mobilní a ostatní okrajová zařízení.

5.2 Tvorba datové sady

Druhou částí bude tvorba datové sady, která bude použita pro trénování, validaci a ohodnocení modelu. Zde bude cílem pořídit co největší počet snímků specifikovaných komponent v reálném prostředí. Tato část bude spíše použita pro testování výsledků, protože pravděpodobně nebude dostatečně rozsáhlá. Protože je problém vcelku specifický a neexistuje již předem vytvořená datová sada nebo obecně větší množství kategorizovaných snímků, bude třeba vytvořit generátor snímků zcela nových. Tyto části budou extrahovány z dříve pořízených snímků a v případě potřeby budou upraveny různými efekty nebo částečně překryty, aby se docílilo co největší rozdílnosti v generovaných obrázcích.

5.3 Aplikace pro operační systém Android

Finální částí sytému bude mobilní aplikace určená na platformu Android. Vývoj bude probíhat ve vývojovém prostředí známé aplikace Android studio. Zvoleným programovacím jazykem bude Kotlin, protože oproti dříve využívané Javě obsahuje několik výhod. Oba jazyky jsou navzájem nahraditelné, avšak vytváření programu v Kotlinu zabere daleko menší počet řádků, čímž je výsledný kód více čitelný. Navíc obsahuje méně chyb a je lépe udržovatelný a proto je pro tyto účely více využíván.

Hlavním účelem této aplikace bude snímání daných komponent a následné zobrazení informací s nimi spojenými. Tuto část je dále možné rozdělit na dvě menší části. V obou těchto částech je pak žádoucí udržet co nejjednodušší uživatelské rozhraní. Pokud by bylo příliš složité, mohla by být ve výsledku aplikace pro uživatele matoucí a raději by využil jiný způsob pro získání těchto informací. Zároveň by také měly být obsaženy pouze věci, které jsou pro uživatele důležité a za jejichž účelem si aplikaci instaluje.

Snímání komponent kamerou mobilního telefonu

Tato část aplikace bude kontinuálně získávat snímky z kamery telefonu, aby nebylo nutné snímek pořizovat vícekrát, například z důvodu rozmazání. Při detekci některé z komponent, aplikace vyčká určitou dobu a pokud se detekované komponenty nezmění, zobrazí uživateli stránku s výsledky. V rámci snímání se budou uživateli zobrazovat průběžné výsledky. Toto je vhodné implementovat, aby pro uživatele bylo možné zareagovat, pokud aplikace některé komponenty z jakéhokoliv důvodu nedetekuje. Řešením pak může být například zvolení jiného úhlu snímání. Aktualizace těchto průběžných výsledků bude prováděna pouze v určitém časovém intervalu, aby nebyly změny příliš rychlé a matoucí. Ve výsledku bude moci uživatel v této části provádět pouze akci zapnutí a vypnutí analýzy snímků, protože další akce nejsou pro analýzu třeba.

Zobrazení výsledků snímání

Po dokončení snímání, v případě, kdy byl detekován nenulový počet komponentů, bude uživatel přenesen na stránku se zobrazeným seznamem výsledků. Zde bude moci seznam ještě upravovat, tedy přidávat nebo odebírat komponenty, které byly chybně detekovány. Po potvrzení výsledků bude možné prvky seznamu expandovat, pro zobrazení dalších informací o dané komponentě. Zde se kromě samotného názvu bude zobrazovat také popis, vysvětlující účel dané komponenty. Až uživatel získá všechny informace, které hledal, bude se moci přímo vrátit zpět na část snímání.

Kapitola 6

Implementace modelu

V dřívějších částech bylo popsáno zpracování obrazu za pomocí konvolučních neuronových sítí a nastavení parametrů v rámci těchto sítí. Dále byly zmíněny techniky, které modelu pomáhají v dosáhnutí co nejlepších výsledků. Na závěr byly uvedeny možné optimalizace, které je vhodné provést při využití modelu na mobilním zařízení.

V návrhu tohoto modelu již byly zmíněny některé nástroje, za pomoci kterých bude tohoto cíle dosaženo. Vývoj modelu bude tedy probíhat v programovacím jazyce Python, který je pro tento účel velmi vhodný. Jeho jednoduchost umožňuje rychlé řešení problému, bez zabývání se dalšími, pro řešený problém nepodstatnými, okolnostmi. Velkou výhodou je také ohromné množství dostupných knihoven. V rámci tvorby modelu se bude jednat o knihovny Tensorflow[1] a Tensorflow lite[2]. Generování dodatečných dat bude zajištěno především knihovnou Pillow[3].

6.1 Knihovna Tensorflow

Jedná se o populární knihovnu vytvořenou společností Google, která tuto technologii využívá ve většině svých produktů, například pro vylepšení vyhledávání, překlad nebo různá další uživatelská doporučení. Tato knihovna slouží jako jednoduché a rychlé řešení pro vývoj neuronových sítí. Je možné ji využít pro jazyky Java, C++ a Python. Základní architektura využívání této knihovny je rozdělena na tři části: předzpracování dat, tvorba modelu spojená s trénováním a vyhodnocení výsledného modelu.

Výhodou této knihovny je možnost využití již hotových populárních modelů. Toto je doplněno o možnost načítání již hotových datových sad. Ve výsledku se tedy jedná o velice vhodný nástroj pro rychlou tvorbu modelů. Výsledný model je pak možné využít na jakémkoliv zařízení. Avšak pro případ okrajových a mobilních zařízení je nutné využít již dříve zmíněné doplňující knihovny Tensorflow lite. Za její pomoci je pak snížena velikost modelu, jeho požadavky na paměť a zrychlena výsledná inference, tedy rychlost, s jakou dokáže model zpracovávat data.

[1] https://www.tensorflow.org/
[2] https://www.tensorflow.org/lite/
[3] https://pillow.readthedocs.io/en/stable/

6.2 Generování nových dat

Data a z nich tvořené datové sady jsou zásadní a nedělitelnou součástí procesu trénování modelu. Těchto dat je však třeba mít pro každou kategorii určité množství. Není možné trénovat model obsahující pouze velice malé množství snímků ke každé kategorii a čekat, že výsledky budou při výsledném nasazení dobré. Zde platí pravidlo čím víc, tím líp, avšak není vhodné tento rozsah přehánět, protože s větším počtem trénovacích dat je spojená i delší doba potřebná ke trénování. Vhodným přístupem je tedy postupné přidávání dat a následovné testování, zda je daný počet dat pro model dostačující. Například, pokud by se doba trénování prodloužila na dvojnásobek té původní a úspěšnost modelu by při testování zaznamenala zlepšení pouze rámci desetin procent, je jasné, že nemá cenu trénovat s takto zbytečně velkou sadou, pokud to z nějakého jiného důvodu není nezbytné.

Přestože bylo pořízeno značné množství fotek deskové hry, nejednalo se o zdaleka dostatečný počet, aby bylo tyto data možné využít jak pro účel trénování, tak pro účely validace a testování. Za účelem větší diverzity snímků bylo při pořizování fotek využito různých filtrů. Toto se ve výsledku ukázalo jako nesprávný krok, jelikož z těchto fotek dělal model velice špatné závěry. Nejspíše v tomto případě hrála velkou roli změna barev, což pro trénovaný model znamenalo, že již tak velice podobné komponenty nebyl schopný rozeznat ani podle těchto rysů. Ve výsledku bylo tedy možné využít jen malou část, která nebyla pořízena za použití filtrů. Tyto zbylé snímky pak byly použity hlavně při testování aplikace v emulátorech.

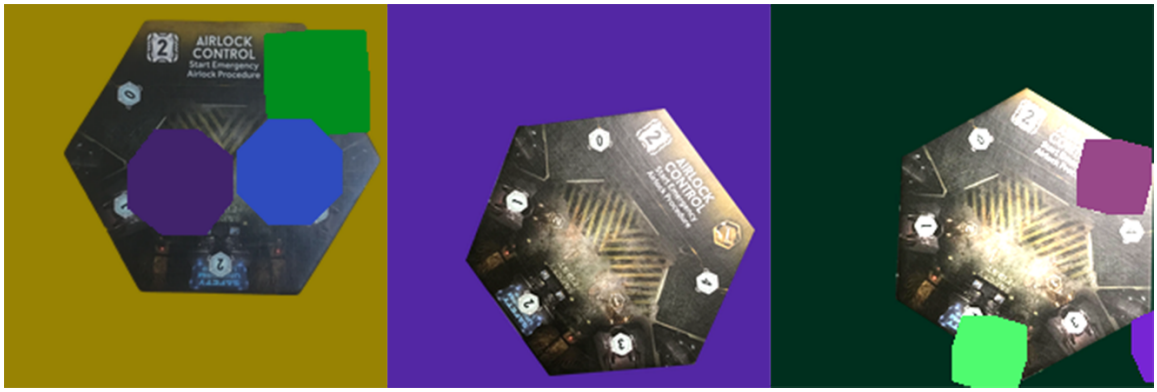
Důsledkem této chyby byl však posun jiným směrem, kterým bylo generování snímků. Za tímto účelem byl vytvořen program pro jejich tvorbu `generate.py`. Nejdříve bylo potřeba z validních pořízených snímků extrahovat dané komponenty. Pro každou komponentu bylo takto extrahováno deset různých obrázků, z fotek, vyfocených v různých prostředích, za různých světelných podmínek. Přesto, že se jedná o vzorek toho samého komponentu, jsou tyto různé změny v prostředí natolik značné, že zásadně mění výsledný vzhled.

Za účelem zlepšení výsledků modelu byly přidány do generovaných snímků náhodně vygenerované překážky. Tato operace byla provedena, aby se dosáhlo co nejlepší generalizace. Přesněji tedy za účelem, aby byl model schopný identifikovat dané komponenty i podle omezeného počtu rysů, tedy i za možnosti, že některá z částí dané komponenty je překrytá.

Těchto vygenerovaných bloků je náhodný počet v rozmezí nula až tři, aby existovaly také snímky bez jejich výskytu a také aby jich nebylo až příliš velké množství, které by mohlo znemožnit rozpoznání jakýchkoliv rysů. Tyto bloky jsou náhodně škálovány, podle velikosti snímku dané komponenty, aby nehrozilo její překrytí. Na blocích je provedena rotace, což mění jejich výsledný tvar na náhodný mnohoúhelník. Tyto mnohoúhelníky jsou náhodných barev, aby se na jejich rysy model zbytečně nezaměřoval. Následně jsou vloženy na náhodné místo v rozmezí dané komponenty. Na závěr jsou provedeny i akce pro změnu jasu, ostrosti a saturace celého snímku, opět za účelem zlepšení generalizace.

Ve výsledku je tedy možné tento program využít pro generování velkého množství nových a unikátních dat ve specifikované velikosti. Obecně je zde snaha zavést do tohoto generování co největší faktor náhody, protože čím větší bude diverzita těchto snímků, tím lepší generalizace a výsledků by měl model dosáhnout. Ve výsledku jsou pak tato data využita pro trénování, validaci a testování. Příklady těchto vygenerovaných dat je možné vidět na obrázku 6.1.

V rámci práce bylo vyzkoušeno několik rozsahů velikostí datových sad. Pro účel ladění modelu byly využity menší sady, které byly ve finále nahrazeny větší výslednou, která obsahuje tisíc snímků ke každé kategorii v rámci trénovací sady.

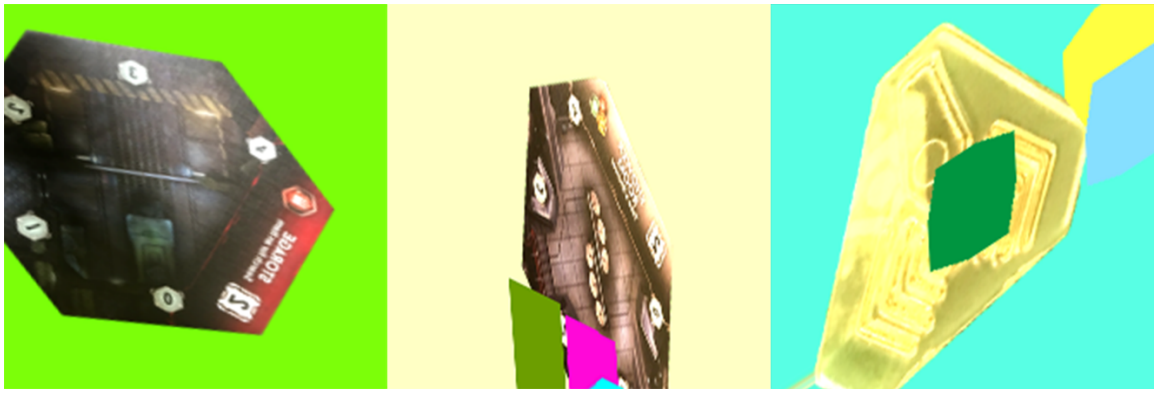


Obrázek 6.1: Příklad vygenerovaných snímků s různými úrovněmi jasu a překrytí.

6.3 Augmentace stávajících dat

Augmentace je další vhodnou operací, pro získání většího množství dat. V tomto případě je využita přímo v rámci hlavního programu pro tvorbu a trénování modelu `train.py`. Zde není potřeba tyto upravené snímky, na rozdíl od těch vygenerovaných, ukládat. Průběh augmentace dat je v programu následující. Uložené snímky se načítají z vybrané složky, ve které se nachází příslušná datová sada. Toto načítání snímků je propojeno s třídou `ImageGenerator`, která zajišťuje vybrané augmentace dat. Během každé epochy jsou poté aplikovány transformace jako například přiblížení, rotace nebo změna jasu.

Všechny tyto provedené úpravy pomáhají generovat nová unikátní data pro vylepšení generalizace modelu. Tyto transformace jsou aplikovány čistě náhodně, je tedy možné, že budou provedeny všechny zároveň, ale je také možné, že nebude provedena žádná. Hodnoty těchto transformací jsou ve většině případů určeny pouze zvoleným rozmezím. To znamená, že stupeň, do jakého bude daná transformace provedena, je také náhodný. Příklad takto augmentovaných snímků je možné vidět na obrázku 6.2.



Obrázek 6.2: Příklad augmentovaných snímků.

6.4 Rozdělení dat do várek

Data, se kterými model dále pracuje, jsou dělena do takzvaných várek (batch). Tuto hodnotu je nutné specifikovat při načítání datové sady. Kromě toho, že je práce s větším počtem várek

náročnější na paměť, protože program musí držet tento počet obrázků načtený v paměti, tak také ovlivňuje do jisté míry schopnost modelu generalizovat. Tento jev je způsoben tím, že model hledá podobné rysy právě v rámci jedné várky. Pokud jsou pak snímky dostatečně rozdílné, je schopný určit vhodné rysy pro jejich rozpoznání. Extrémními případy, které nepůsobí dobrým efektem při trénování modelu, jsou příliš malé nebo příliš velké várky. Pokud je jejich velikost příliš velká, může to znamenat, že se budou stále nahrávat podobné nebo dokonce stejné kombinace snímků. V extrémním případě bude nahrána celá datová sada. Výsledkem tedy bude model, využívající stále ty stejné rysy, které však nemusí být při reálném využití optimální. Naopak, pokud je v rámci jedné várky snímků příliš málo, může být do trénování zahrnut zbytečný šum. V situaci, kdy model určuje rysy, například jen u dvou snímků zároveň, je patrné, že ty, ke kterým dojde, budou pravděpodobně méně použitelné, něž kdyby bylo snímků, třeba často používaných, třicet dva.

V poslední řadě je ještě vhodné vložit další náhodnou proměnou v podobě zamíchání datové sady. Toto nastavení před začátkem každé epochy snímky promíchá, čímž zajistí, že se nebudou ve várkách vyskytovat identická data. Tímto nebude modelu umožněno, aby si zapamatoval přesné výskyty dat v sadě, což by bylo nežádoucí. Toto je samozřejmě za předpokladu dostatečně velké datové sady a vhodné velikosti várky. V případě tohoto modelu je využito standardní velikosti várky třicet dva, avšak bylo experimentováno i s různými dalšími hodnotami. Dostatečný rozsah datové sady je pak zajištěn dříve zmíněným generováním a augmentací.

6.5 Tvorba architektury výsledného modelu

Model konvoluční neuronové sítě je vytvořen v dříve zmíněném programu `train.py`. Tato architektura byla vytvořena specificky pro účel této práce a nejedná se tedy o kopii některé z často využívaných. Určitá inspirace byla získána z architektury VGG-19, avšak v průběhu vývoje se model značně odlišil.

U všech architektur konvolučních neuronových sítí je obecně využito určitého počtu konvolučních vrstev, po kterých následuje pooling vrstva. V tomto modelu jsou využívány maxpooling vrstvy, které vybírají pouze rysy s nejvyšší hodnotou. V kombinaci tyto dva typy vrstev vytváří konvoluční bloky. Celá architektura je následně zakončena plně propojenými vrstvami, což zahrnuje i vrstvu výstupní.

Ladění vytvářených modelu

V případě této práce byl model postupně laděn a vývoj na něm probíhal iterativně. Při každé změně byly tedy provedeny testy, za účelem zjistit, zda model pracuje lépe nebo hůře, než ten předešlý. Tento způsob je dále popsán v kapitole 8, věnující se testování vytvořeného modelu. Postupně byl zjištěn vhodný počet konvolučních bloků a v nich přítomných konvolučních vrstev. Zároveň bylo třeba určit vhodné hodnoty hyperparametrů. Těmito hodnotami byl počet filtrů konvolučních vrstev, neuronů plně propojených vrstev, velikost várky, rychlost učení a několik dalších. Zde bylo vhodných hodnot opět dosaženo postupnými změnami a následným testováním.

Při těchto testech byla hlavní sledovanou metrikou ztráta a správnost modelu na testovacích datech. Průběh trénování byl také zachycen do grafu, díky kterému bylo snadné identifikovat příliš velké změny v modelu, způsobené zbytečně velkou rychlostí učení. Z grafu bylo možné také vyvodit, jaký počet epoch je pro model optimální a jaký je naopak zbytečně dlouhý, což pomohlo určit trpělivost early stoppingu.

Určování parametrů pomocí KerasTuner

Kromě manuálních změn a testů bylo těchto hodnot prvotně dosaženo také automaticky za pomoci nástroje KerasTuner[4], který vybírá hodnoty z daného rozmezí a následně model testuje. Poté, co provede určitý počet testů, porovná dosažené výsledky všech variant modelu a pokračuje v trénování pouze lépe fungující poloviny. Toto je opakováno do té doby, dokud není nalezena jedna nejlepší hodnota. Takto byl například laděn počet filtrů konvolučních vrstev nebo počet neuronů vrstev plně propojených.

Problém rychlosti výsledného modelu

Při testování výsledného modelu v mobilní aplikaci však byla odhalena jeho příliš pomalá inference, což je pro účel analýzy kontinuálního snímání velice nežádoucí. Model obsahoval velké množství parametrů, proto byl pro mobilní zařízení příliš výpočetně náročný. Ve výsledku tento model pracoval pomaleji než jeden snímek za sekundu, což bylo velice nevhodné. Kromě využití vyšší míry optimalizačních technik bylo třeba zmenšit i počet parametrů přítomných v modelu, za účelem dosažení co nejvyšší rychlosti. Avšak toto bylo nutné balancovat společně se správností modelu, protože čím méně parametrů model měl, tím hůře ve výsledku pracoval. Muselo být tedy v tomto ohledu dosaženo jistého kompromisu. Z tohoto důvodu nemohly být využity některé hodnoty získané pomocí KerasTuner.

Problém rychlosti modelu je složitý hlavně v tom, že existuje obrovské množství různých kombinací hardware, na kterých může být model spuštěn. Je sice možné testovat rychlost modelu již po jeho vytvoření na počítači, avšak jediné, k čemu je tyto hodnoty možné využít, je jen jakési porovnání s ostatními verzemi. Rychlost byla testována především na tabletu s verzí Android 8.1, dodatečně byla rychlost otestována i na dalších dvou zařízeních. Prvním byl mobilní telefon s verzí Android 7.1, druhým byl pak novější telefon s Android 11, který oproti předešlému zařízení při testování dosahoval až desetinásobně vyšší rychlosti. Z tohoto je patrné, že optimalizovat model není jednoduché, například i proto, že některé grafické procesory mobilních telefonů nejsou podporované.

Regularizace v rámci modelu

V dřívějších verzích modelu se vyskytovaly dropout vrstvy a L2 regularizace, které měly pomáhat modelu předcházet přetrénování. Tyto metody se pro tento model však ukázaly jako nepotřebné, z důvodu velkého množství dat a využití brzkého zastavení modelu. Jejich jediným pozorovaným efektem bylo zpomalení konvergence modelu, a z tohoto důvodu se ve výsledné verzi již nevyskytují.

6.6 Brzké zastavení procesu trénování pomocí early stopping

Tato technika, zmíněná v dřívější části zaručuje, že nedojde k přetrénování modelu, a tedy k nechtěné ztrátě správnosti. Toto je zajištěno pomocí callbacku (zpětného volání), který po dokončení každé epochy kontroluje, zda model zlepšil svůj dosavadní nejlepší výsledek. Zde je možné vybrat různé metriky, které sledovat. Smysl však dává sledovat pouze validační metriky, protože sledováním trénovací správnosti nebo ztráty by se přetrénování modelu nepředešlo. Na výběr je tedy validační správnost a ztráta. Zde je však nutné uvědomit si, co tyto metriky určují. Jsou sice spolu do jisté míry propojené, avšak mají určité zásadní

[4] https://keras.io/keras_tuner/

rozdíly. Správnost určuje, kolik snímků byl model schopný kategorizovat správně. Avšak, pokud jsou všechny určeny správně jen o trochu lépe, než ty nesprávné, nejedná se o tu nejspolehlivější metriku. Příkladem mohou být případy, kdy jsou rozdíly v pravděpodobnosti detekovaných kategorií velice malé nebo situace, kdy je kategorie sice určena správně, avšak s nízkou úrovní jistoty. Vhodnější je v tomto případě tedy sledovat ztrátu, která je vázaná na použitou ztrátovou funkci modelu. Ta určuje, jak moc se model mýlil ve svých předpovědích, ať už při těch správných nebo nesprávných. Cílem je tedy sledovat, zda model snižuje hodnotu ztráty s postupem v epochách.

Počet epoch, po který má tento callback sledovat, zda se model zlepšuje nebo nikoliv, je také důležitým tématem. Pro tento účel je možné si nastavit různě agresivní limity, které určují, kolik epoch bude early stopping vyčkávat, než zastaví trénovací proces. Zároveň také platí, že tyto limity musí být v rozumných mezích, tedy pokud by čekal pouze jednu epochu, tato metoda ve většině případů ztratí smysl. Tento nízký limit je sice použitelný, avšak ve většině případů nemusí dojít ke zlepšení při každé další dokončené epoše, čímž by toto mohlo procesu trénování spíše uškodit. Druhým extrémem je případ, kdy bude čekat až příliš velký počet epoch, například v řádů stovek. Následkem bude značné prodloužení trénování, za pravděpodobně minimálního zisku ve výkonu modelu. Zde je tedy potřeba zvážit, jak moc velkých posunů je model schopný dosáhnout, pokud mu bude dáno více času a zda mají tyto zisky dostatečnou váhu. Při určování trpělivosti early stoppingu je také třeba vzít v potaz velikost použité datové sady, protože ta značně ovlivňuje délky těchto epoch.

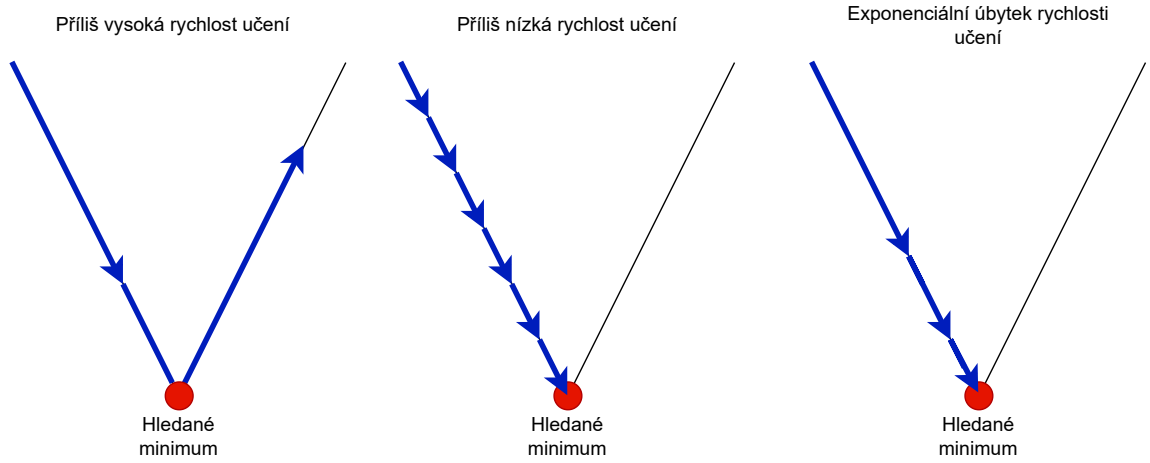
Trpělivost early stoppingu v rámci této práce byla nastavena na dobu deseti epoch, přičemž model dosahuje optimálních výsledků a proces trénování tímto není značně prodloužen i přes velké množství použitých dat.

6.7 Exponenciální úbytek rychlosti učení

Další použitou technikou pro zlepšení procesu učení je exponential decay (exponenciální úbytek) rychlosti učení. Tato technika zajišťuje, že je v průběhu trénování rychlost učení modelu exponenciálně snižována, oproti konstantní hodnotě při přístupu bez jejího využití. Důležitost tohoto přístupu je možné popsat na jednoduchém příkladu 6.3, kde figuruje funkce tvaru V.

Model začíná na okraji této funkce a jeho cílem je najít její minimum. Bez využití této techniky má model konstantní rychlost učení a s velkou pravděpodobností dojde k tomu, že se model minimu přiblíží, ale protože je jeho rychlost učení příliš velká, hledané minimum přestřelí na opačnou stranu. Tímto může svůj aktuální nejlepší výsledek spíše zhoršit. Tento jev se může teoreticky opakovat až do nekonečna a ve výsledku bude model, buď méně přesný, nebo pokud minima dosáhne, zabere mu to značně delší dobu. Jasným řešením by mohlo být snížení rychlosti učení, avšak to by znamenalo snížit tuto hodnotu na takový stupeň, při kterém by model zaručeně minima dosáhl. To by ve výsledku znamenalo opět velice pomalý průběh učení.

Z tohoto příkladu je jasně vidět benefit, který model získá tím, že se bude jeho rychlost učení postupně snižovat. Díky tomu je možné začít i s daleko vyšší hodnotou, aby se model co nejrychleji přiblížil tomuto minimu. Postupně je pak v průběhu epoch jeho rychlost snížena na takovou úroveň, že nemá problém toto minimum lokalizovat.

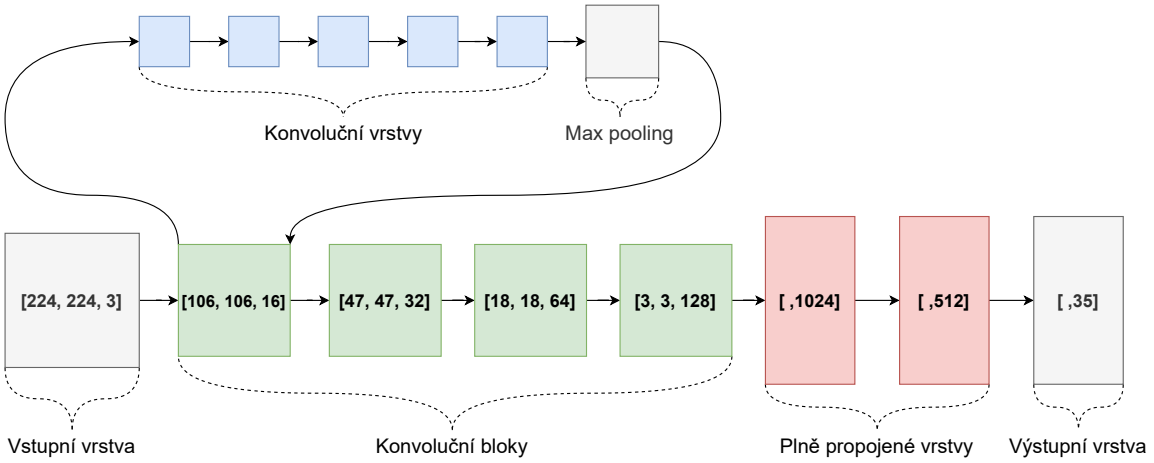


Obrázek 6.3: Příklad využití nevhodných hodnot rychlosti učení a vhodnějšího exponenciálního úbytku učení.

6.8 Výsledná architektura modelu

Schéma výsledné architektury modelu je možné vidět na následujícím obrázku 6.4. Skládá se ze vstupní vrstvy, která přijímá barevné snímky o rozlišení 224 x 224. Následována je čtyřmi konvolučními bloky, které postupně zmenšují rozlišení analyzovaného obrázku a zároveň provádějí extrakci rysů za pomoci stále většího počtu filtrů. Zakončení je zajištěno plně propojenými vrstvami, přičemž výstupní vrstva obsahuje počet neuronů, který je rovný počtu kategorií, které obsahuje daná datová sada.

Tohoto výsledku bylo dosaženo, jak již bylo zmíněno, za postupného iterativního vývoje a testování. Zároveň byl brán ohled na rychlost modelu, aby byl využitelný při snímání v reálném čase. Jako rozumná volba se osvědčil počáteční počet šestnácti filtrů, který je postupně navyšován.



Obrázek 6.4: Schéma výsledné architektury modelu. Vyobrazené hodnoty konvolučních bloků představují stav, ve kterém se analyzovaný snímek nachází, poté co tímto blokem prošel.

6.9 Optimalizace vytvořeného modelu

Jak již bylo ustanoveno dříve, pro zajištění co nejlepší funkcionality modelu na mobilním zařízení, je nutné provést několik optimalizací. První takovou optimalizací je kvantizace, přesněji quantization aware training (trénování s ohledem na kvantizaci) [3]. Tato operace je provedena po vytvoření modelu a dalo by se říci, že se jedná o jistý mezikrok mezi neoptimalizovaným modelem a modelem připraveným pro využití na mobilním zařízení.

Tento mezikrok je vhodné provést, protože je proces konverze datových typů v rámci modelu ztrátový a není zachována původní správnost. Zde jsou změněny dříve používané datové typy Float32 na Int8. Toto samozřejmě ihned sníží správnost modelu, avšak připraví ho to lépe na následující optimalizace. Snížení správnosti není tak velkým problémem, jak by se mohlo zpočátku zdát, jen je nutné provést přetrénování této nové verze. Tento proces je obecně kratší, než normální trénování, protože i přes určitou ztrátu je kvantizovaný model schopný dosahovat lepší správnosti, než zbrusu nový model.

Následně je pokračováno konverzí tohoto kvantizovaného modelu na optimalizovanou lite verzi pomocí knihovny Tensorflow lite. Zde je možné optimalizovat model s cílem, co nejmenší velikosti, nebo naopak co nejvyšší rychlosti. I přes tuto volbu je však ponechán standardní režim optimalizace, protože jsou pro mobilní zařízení vhodné oba tyto benefity. Jedná se tedy o jistý kompromis. Důležitější možností je však nastavení datového typu, který bude konvertovaný model využívat. Zde je možné ponechat Float32, avšak tato možnost neposkytuje žádné z optimalizačních benefitů, tedy nedává pro toto využití smysl.

Na opačné straně spektra je Int8, který nabízí velkou možnost zrychlení a zmenšení velikosti. Ten je však méně přesný a pro správnou konverzi potřebuje navíc reprezentativní datovou sadu, podle které tuto konverzi provede. Jako vhodný kompromis se tedy nabízí Float16, který balancuje mezi snížením velikost modelu a zrychlením inference a zároveň nemá zdaleka takový dopad na správnost konvertovaného modelu. Velikost výsledného modelu zároveň nebyla natolik velká, aby bylo nutné provádět agresivnější optimalizaci datových typů nebo aby bylo nutné odstraňovat některé ze zbytečných propojení v rámci modelu.

Stále je třeba mít na paměti, že se tyto konverze většinou neobejdou bez ztráty na správnosti a je ji tedy vhodné monitorovat, aby nedocházelo ke tvorbě nedostatečně přesných modelů. Tato část však bude plně rozebírána až v následující sekci 8, věnující se testování těchto různých verzí.

Kapitola 7

Mobilní aplikace pro OS Android

Po zhotovení návrhu aplikace a vytvoření modelu, který bude využíván k rozpoznání vybraných komponentů, je na řadě implementace samotné mobilní aplikace. Ta je tvořena dvěma hlavními částmi, jak již bylo naznačeno v návrhu. První je samotné snímání komponent a druhou částí je pak zobrazení výsledků.

Aplikace je určena na operační systém Android a je tedy vytvořena v programu Android studio, který je pro vývoj na tuto platformu určen. Dále je využit programovací jazyk Kotlin, z důvodu lepších vlastností a větší jednoduchosti použití oproti programovacímu jazyku Java, který je také možné využít.

Výsledná aplikace je určena pro zařízení s verzí operačního systému Android 7.0 (API level 24) z roku 2016 a novější. Zde je sice doporučené využívat verzi Android 5.0 (API level 21) z roku 2014, z důvodu kompatibility se skoro všemi aktivními zařízeními. Toto však po testování těchto starších verzí pomocí emulátoru, z důvodu odlišných výsledků od verzí novějších, nebylo možné. Problémem byla menší stabilita aplikace nebo její úplná nefunkčnost pro tyto verze. Toto se samozřejmě dá vyřešit vytvořením alternativního kódu, pro tyto specifické verze nebo za využití zastaralejších přístupů. Avšak největším problémem na těchto verzích byla nefunkční konverze objektu `ImageProxy`[1] na bitmapu, jak bude později zmíněno v kapitole 8. Zde obsahoval obrázek velké množství artefaktů nebo byl úplně překryt barvou, což znamenalo, že tyto snímky byly pro analýzu nepoužitelné. Navíc tyto starší systémy většinou nemají tak výkonný hardware jako nynější telefony, což by mohlo mít omezující efekt na aplikaci. Ve výsledku tak není podporováno doporučených přibližně devadesát devět procent, ale pouze asi devadesát pět procent všech zařízení.

Pro práci s kamerou mobilního zařízení je využito knihovny CameraX[2], která umožňuje zachycovat snímky a provádět jejich analýzu. Tento přístup je velice často využíván v rámci tvorby aplikací, které využívají kameru, včetně aplikací pro klasifikaci obrazu. Funkce pro práci s modelem jsou pak zajištěny knihovnami Tensorflow lite, určenými přímo pro platformu Android.

7.1 Snímání komponent deskové hry

Celá aplikace je poměrně jednoduše strukturována a skládá se pouze z jedné hlavní aktivity, která je doplněna dodatečnými fragmenty. V rámci této aktivity je vykonávána většina akcí,

[1] https://developer.android.com/reference/androidx/camera/core/ImageProxy
[2] https://developer.android.com/training/camerax

včetně analýzy snímků. Před spuštěním analýzy je však nejdříve nutné provést několik dalších akcí.

Nutná nastavení před zahájením snímání

První takovou akcí je nastavení toolbaru, který je využit v rámci celé aplikace pro navigaci a provádění akcí. Tyto dostupné akce se liší podle zobrazeného fragmentu nebo aktivity. V této hlavní aktivitě se zde nachází tlačítko pro spuštění nebo zastavení analýzy snímků, před tímto krokem ale ještě proběhne několik akcí. Aby bylo vůbec možné kameru telefonu využít, je třeba nejdříve zkontrolovat, zda má aplikace povolená přístupová práva ke kameře zařízení. V případě, že tato přístupová práva chybí, je nutné o ně zažádat uživatele.

Další akcí, kterou je nutné provést, je načtení modelu Tensorflow lite a nastavení jeho parametrů. Zde je kontrolováno, zda má zařízení podporovaný grafický procesor. Pokud je k dispozici, je ho využito, protože vede ke zrychlení modelu. V opačném případě je zvýšen počet vláken, na kterých model poběží, pro alespoň nějakou úroveň zrychlení.

Konečně k samotnému snímání je potřeba vytvořit instanci kamery a propojit její výstup se zobrazením náhledu, který umožňuje uživateli vidět, co přesně je aktuálně snímáno. Dále je inicializován analyzátor, který bude snímky v dané velikosti postupně posílat na zpracování modelu. Pro tyto účely je využívána zadní kamera mobilního telefonu, protože tuto mají všechna zařízení a je vhodná pro účel snímání.

Analýza snímků a vyhodnocení výsledků

Po dokončení těchto úkonů je již možné zapnout analýzu snímků. Zde se však potřeba se pozastavit nad formátem analyzovaného obrázku. Analyzátor totiž vrací objekt typu `ImageProxy`, což není úplně vhodný formát. V dřívějších verzích této práce byl využíván méně efektivní přístup, kdy bylo nutné konvertovat originální `Image` objekt na YUV bitmapu a poté teprve na RGB. V aktuálním přístupu je však využito knihovny ML Kit pro zpracování obrazu, který provádí tuto konverzi na bitmapu daleko elegantněji. Tato bitmapa je poté škálována na požadovanou velikost a postupně vysázena do vstupního `Bytebufferu`. Tyto pixely jsou dále ještě normalizovány, podobně jako při trénování modelu.

Pro tento buffer je třeba alokovat určité místo. Toto místo je dáno rozměry obrázku, barevným formátem a využitým datovým typem. V tomto případě je tedy šířka i výška rovna 224, je použit formát RGB, který rozšiřuje velikost na trojnásobek. Na závěr je nutné tuto hodnotu vynásobit ještě čtyřmi, protože jsou tyto hodnoty ve formátu s plovoucí čárkou.

Pro výsledek modelu je také třeba vytvořit `Bytebuffer`, který obsahuje místo pro všech třicet pět možných kategorií. Takto vytvořené buffery jsou poté předány modelu, který pro každou kategorii určí její pravděpodobnost, že se ve snímku nachází. Ke všem těmto výsledkům jsou poté přiřazeny jejich štítky, za předpokladu, že není výsledek ignorován. Tyto štítky pochází z dříve načtených souborů, které obsahují názvy všech detekovatelných tříd a jsou seřazeny ve stejném pořadí jako třídy modelu, ke kterým patří. Toto platí i pro soubory obsahující popisy komponent a odkazy na korespondující obrázky.

Protože není vhodné uživateli ukazovat pravděpodobnost výskytu všech třiceti pěti kategorií, je třeba některé výsledky vyfiltrovat a tím pádem nezobrazovat. Je tedy vhodné určit limit pravděpodobnosti, při kterém bude výsledek zobrazen. V tomto případě je tento limit nastaven na hodnotu osmdesáti procent, která se při testování ukázala jako vhodná. Většina dobře rozpoznatelných komponent dosahuje správnosti přes devadesát devět pro-

cent, avšak některé představují jisté problémy, které jsou diskutovány dále ve vyhodnocení, což je důvodem pro tento nižší limit.

Pořadí těchto výsledků je seřazeno od těch s největší pravděpodobností výskytu, k těm s nejmenší. Takto seřazené výsledky jsou průběžně zobrazovány pomocí `RecyclerView` v dolní části obrazovky, jak je vidět na tomto obrázku 7.1. Toto je prováděno, aby měl uživatel přehled, které komponenty model detekuje, popřípadě pokud žádné nedetekuje. Tímto je uživateli v průběhu umožněno změnit pozici snímání, pro dosažení lepších výsledků.

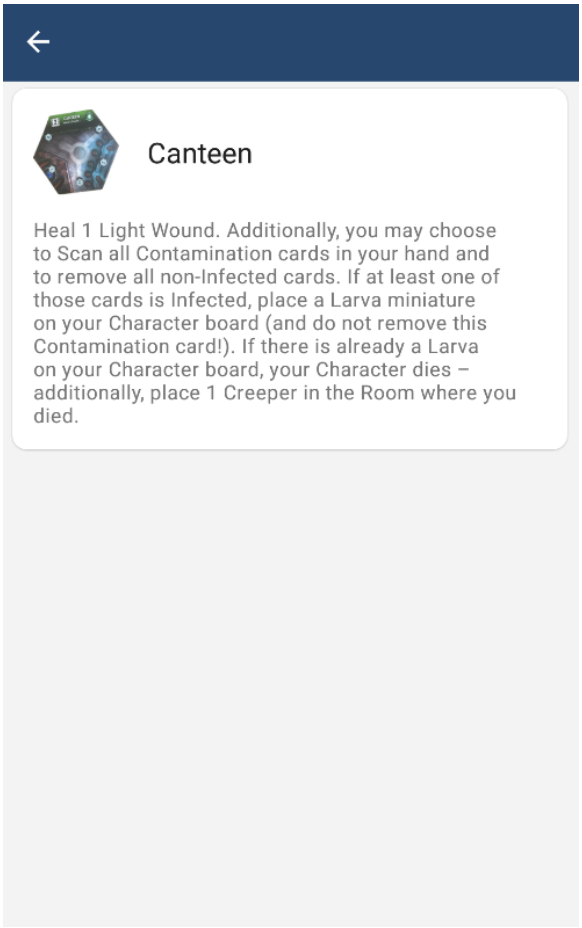
Obrázek 7.1: Průběh analýzy komponenty se zobrazením detekované kategorie a její pravděpodobnosti, že se ve snímku nachází.

Pokud není seznam detekovaných komponent prázdný, je spuštěn automatický odpočet, který uživatele převede do fragmentu se zobrazenými výsledky. Zde je nastaven limit čtyř vteřin, jako kompromis mezi zbytečně dlouhou dobou čekání, a naopak příliš krátkou dobou na správné zachycení scény kamerou telefonu. Zároveň se tato změna na fragment s výsledky provede pouze, pokud obsah seznamu výsledků zůstal identický, tedy pokud nebyla například změněna zaměřená scéna nebo komponenty, které se v ní nachází.

Při přesunu aplikace do zobrazení výsledků je pak automaticky zastavena analýza snímků, jelikož je to výpočetně náročná část aplikace a aktuálně není potřeba, vzhledem k tomu, že výsledky této analýzy se nikam nepromítají.

7.2 Zobrazení výsledků snímání

V této části jde pouze o zobrazení výsledků poskytnutých modelem a informací k nim vázaným. Tato část se dále dělí na dva fragmenty. První umožňuje uživateli mazat výsledky, o které nemá zájem nebo které jsou chybné. Alternativně má uživatel možnost, přes horní panel, přejít do fragmentu druhého, ve kterém může přidávat nové, popřípadě chybějící komponenty. Pokud je uživatel s výsledky spokojen, může je odsouhlasit. Následně pak bude mít možnost tyto karty expandovat, aby mu byl zobrazen jejich popis, jak je možné vidět na obrázku 7.2. Seznamy komponent ve zmíněných fragmentech jsou naplňovány a editovány pomocí adaptéru. Poté, co uživatel získá hledané informace, vrací se, přes horní panel, zpátky na skenování obrázků, do hlavní aktivity.



Obrázek 7.2: Zobrazení informací o komponentě v rámci potvrzených výsledků.

Kapitola 8

Testování a vyhodnocení výsledků

Podstatou této kapitoly bude vyhodnocení výsledků, kterých výsledný model dosahuje a popis testování vytvářených modelů. Tyto výsledky není vhodné odvozovat z malého počtu případů, například pouze z výsledků poskytnutých mobilní aplikací. Kromě nedostatečného počtu vzorků by velkým problémem byla i skutečnost, že by pravděpodobně testy nebyly provedeny nad identickými snímky. Pro tento účel je vhodné využít souhrn neměnných dat, čímž může být již dříve vytvořená testovací datová sada. Samozřejmě je jasné, že tento přístup byl využíván i při testování různých architektur nebo počtu jejich parametrů, aby bylo možné vybrat co nejlépe fungující možnost. Za tímto účelem bylo nutné vytvořit určitý testovací proces, který by zajistil, že testované modely budou hodnoceny stále stejným způsobem, aby mělo testování smysl.

8.1 Proces testování vytvořeného modelu

Určitým způsobem testování jsou i výsledky získané na validační sadě v rámci trénování. Už jen díky těmto výsledkům je možné určit, jak pravděpodobně bude vytvořený model fungovat při zcela nových situacích. Jak již bylo dříve zmíněno, používat pouze validační sadu pro testování výkonu modelu, nemusí být tou nejlepší volbou. Různé změny prováděné v modelu za účelem dosažení co největší hodnoty správnosti a co nejmenší hodnoty ztráty na validačních datech, mohou mít za následek přetrénovaní modelu. Ve výsledku se tedy bude model specializovat na rozpoznání validačních dat, avšak na datech nových bude dosahovat pravděpodobně daleko horších výsledků. Z tohoto důvodu je nutné mít navíc sadu testovací, která na výskyt tohoto jevu vývojáře upozorní.

Testování nově vytvořeného modelu

Poté, co model dosáhne svého nejlepšího výsledku, tedy zmenší ztrátu na nejmenší hodnotu, které byl schopný dosáhnout v rámci nastavené trpělivosti early stoppingu, je třeba provést jeho vyhodnocení na zmíněné testovací sadě. Toto je možné provést více způsoby, avšak jako jednoduchá možnost se nabízí využití metody `evaluate` z knihovny Tensorflow. Tato metoda vyhodnotí zvolené metriky na specifikované datové sadě. Protože je tento model určen pro detekci více tříd současně, je nutné kromě samotné správnosti sledovat i přesnost a úplnost modelu. Díky těmto hodnotám, je pak možné přesněji určit, do jaké míry je model schopen správně identifikovat vybrané snímky.

Tímto krokem však proces testování ještě nekončí. Je stále třeba brát ohled na fakt, že je využíváno kvantizace ve snaze snížit náročnost modelu, jak po stránce výpočetní,

tak po stránce využité paměti zařízení. Aby byl na tento proces výsledný model co nejlépe připraven je, jak již bylo v implementaci řečeno, prováděn quantization aware training. Následkem tohoto přetrénování modelu, při kterém je značně ovlivněna jeho správnost konverzí datových typů, je nová neotestovaná verze modelu. Tuto novou verzi je tedy také nutné otestovat stejným způsobem, na testovacích datech, jako verzi předešlou. Toto testování je prováděno, aby se ověřilo, že se model dokázal dobře adaptovat na provedené změny a dosahuje podobných výsledků jako jeho předchůdce.

Testování optimalizovaného modelu

Nakonec je nutné nezapomenout na skutečnost, že cílem není vytvoření standardního modelu pro využití na výkonných zařízeních, ale naopak modelu optimalizovaného pro využití na mobilních zařízeních. V této části je tedy prováděna konverze z kvantizovaného modelu na model optimalizovaný. Zároveň je po provedení této konverze nutné otestovat tuto nově optimalizovanou verzi oproti předešlé neoptimalizované. Bohužel za tímto účelem již není možné využít metodu `evaluate`, jak v předešlých případech.

Pro tento účel bylo tedy nutné vytvořit funkci, která toto otestování zajistí. V rámci této funkce jsou postupně načítány obrázky ze specifikované sady pomocí knihovny OpenCV2[1]. Na tato data je aplikována normalizace, což je velice důležitý krok, bez kterého by výsledky pravděpodobně nebyly směrodatné. Postupně jsou pak tyto snímky zpracovávány touto optimalizovanou lite verzí modelu. Zároveň jsou tato data testována také za pomocí předešlého kvantizovaného modelu. Toto je prováděno z důvodu, aby bylo možné porovnat výsledky těchto dvou verzí a určit, zda při konverzi došlo k nějaké výrazné neočekávané změně ve správnosti modelu.

Určitou hodnotou, kterou je zde možné sledovat, je doba inference optimalizovaného modelu. Přesto, že takto získané výsledky není možné aplikovat na mobilní zařízení, je alespoň možné získat určitou představu o tom, který z modelů je rychlejší. Zde je tedy sledována průměrná doba, kterou modelu trvá snímek vyhodnotit. Pro určení, zda je výsledek poskytnutý oběma modely stejný, je analyzována výsledná matice, která obsahuje jedno umístění pro každou z trénovaných kategorií.

Tady je možné využít různých metrik pro určení přesnosti modelu. První možností je pozorovat správnost modelů v rámci sledované kategorie. Alternativně je možné brát v potaz pouze kategorii s nejvyšší pravděpodobností a kontrolovat, zda jde o kategorii očekávanou. Zároveň je také vhodné určit nějakou hranici, která bude oddělovat použitelné výsledky, aby nebyl brán jako výsledek i náhodný šum.

V rámci tohoto způsobu se však v průběhu testování objevil jistý problém. I přesto, že modely, jak originální, tak kvantizovaný, dosahovaly přibližně devadesáti osmi procentní úspěšnosti na validačních i testovacích datech, bylo po provedení několika těchto testování jasné, že tento způsob testování není zcela dokonalý. Důvodem byla značně nižší pozorovaná správnost optimalizovaného lite modelu při tomto vyhodnocení. Přesněji šlo asi o deseti až patnácti procentní snížení správnosti. Po otestování různých změn v rámci konverze a kvantizace modelu, bylo dosaženo závěru, že je tento jev způsoben pravděpodobně rozdílným způsobem načítání dat oproti metodám knihovny TensorFlow. Potvrzením této teorie byla skutečnost, že ostatní verze modelu dosahovaly velice podobných výsledků jako optimalizovaný model, při stejném způsobu testování.

Tento fakt ale ve finále nehraje až tak velkou roli, protože jediné, co je porovnáváno, jsou výsledky oproti předešlému kvantizovanému modelu. To tedy znamená, že pokud model do-

[1] https://docs.opencv.org/4.x/

sahuje velice podobných výsledků jako jeho předchůdce, je možné brát dřívější vyhodnocení modelu jako směrodatné. Tento závěr vyplývá z pozorování výsledků, které tyto modely poskytovaly v různých kategoriích. Rozdíly se v těchto výsledcích pohybovaly ve velice malých hodnotách a je tedy možné je brát jako zanedbatelné.

8.2 Testování modelu v rámci mobilní aplikace

I přes daleko větší spolehlivost a důležitost testování na jednotné datové sadě, je dodatečně vhodné otestovat, jak dobře model funguje v rámci mobilní aplikace. V tomto prostředí je totiž daleko jednodušší simulovat rušivé jevy, které se budou pravděpodobně vyskytovat i při jeho běžném využití. Současně je možné otestovat samotný chod mobilní aplikace a její schopnost správně pracovat s poskytnutým modelem.

Typ komponenty	Vyšší úroveň přiblížení	Vyšší úroveň oddálení
Místnost	20/20	14/20
Žeton	12/12	0/12
Značka	2/3	1/3

Tabulka 8.1: Výsledky testování systému při různých úrovních přiblížení nebo oddálení.

V průběhu vývoje bylo vytvořeno několik modelů, jejichž správnost dosahovala přibližně devadesáti osmi procent, jak na datech validačních, tak na datech testovacích. Zároveň bylo dosaženo téměř stoprocentní shody optimalizovaného modelu s verzemi předchozími, což zajišťuje velice podobné chování.

Typ komponenty	Tmavší prostředí	Světlejší prostředí
Místnost	18/20	18/20
Žeton	2/12	10/12
Značka	0/3	3/3

Tabulka 8.2: Výsledky testování systému při různých světelných podmínkách.

Tato úspěšnost se však plně neprojevila při testování na nových datech přímo v mobilní aplikaci. Při tomto testu bylo možné detekovat všechny určené třídy správně, avšak s jistými omezeními. Hlavním problémem byla slabší schopnost identifikovat menší komponenty. Toto sice bylo možné, avšak v závislosti na tom, o kterou specifickou komponentu se jednalo. Správného výsledku pak bylo různě obtížné dosáhnout. Pro dosažení správného výsledku bylo nutné zkoušet různé směry a různé vzdálenosti přiblížení. Výsledky testování různých vzdáleností jsou vidět v tabulce 8.1. Problém pro některé komponenty také představoval stín, který byl při snímání vrhán testovacím zařízením. Tyto výsledky z různých světelných podmínek se nachází v tabulce 8.2.

Typ komponenty	Běžné podmínky	Částečné překrytí	S více komponenty současně
Místnost	20/20	13/20	18/20
Žeton	12/12	-	6/12
Značka	3/3	-	0/3

Tabulka 8.3: Výsledky testování systému při částečném překrytí komponenty a při výskytu více komponent současně. Při částečném překrytí je vidět přibližně jen polovina dané komponenty. Toto není možné provést pro žetony a značky, z důvodu jejich již malé velikosti. Zároveň jsou v rámci hry jedinými komponenty, u kterých tato situace může nastat místnosti, zbytek tedy není testován. V případě více komponentů současně je testováno, zda je model schopný detekovat alespoň testovanou komponentu.

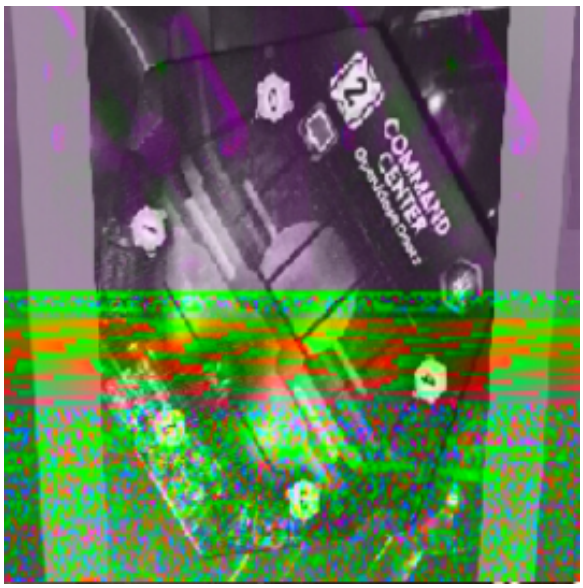
Ke správné identifikaci docházelo občas až po delší době. Opakem tohoto byly větší hexagony místností. U těchto dílků neměl model téměř žádný problém provést správnou identifikaci. Problémem však byly situace, při kterých byly překryté dalšími komponenty. Výsledky těchto testů byly různé, podle počtu a pozice těchto komponent. Model v těchto situacích většinou rozeznal hexagon místnosti správně, avšak ignoroval komponenty na něm se nacházející, podobně jako na tomto obrázku 8.1. Při této situaci navíc platilo, že čím více byla místnost překryta, tím méně byla pro model rozeznatelná. Statistiky z testování překrytí je možné nalézt v této tabulce 8.3.

Obrázek 8.1: Detekce pouhého hexagonu místnosti z více přítomných kategorií.

8.3 Testování funkcionality mobilní aplikace

Mimo to byla samozřejmě testována i samotná mobilní aplikace, avšak toto testování není, na rozdíl od vytvářených modelů, možné automatizovat. Zde je značnou výhodou, že výsledná aplikace není příliš složitá, a tak jde spíše jen o odchycení krajních stavů. Jednou z možných slabin by mohla být všudypřítomná tlačítka, která by teoreticky mohla způsobit některý z chybových stavů, pokud by se uživatel hodně snažil a tato tlačítka například rychle mačkal.

Cílem testování tedy bylo zaměřit se hlavně na možné jevy, které by snižovali stabilitu aplikace. Zároveň byly provedeny testy aplikace na třech různých fyzických zařízeních, jak již bylo dříve zmíněno a několika emulátorech s různými verzemi systému Android. Zde nebyly odhaleny žádné chyby. Je však nutné podotknout, že při brzkých fázích testování byly zjištěny určité rozdíly v chování mezi emulátory a fyzickými zařízeními. Tyto rozdíly se projevovaly hlavně při konverzi získaného obrazu z analyzátoru, jak je možné vidět na obrázku 8.2. Výsledkem byla takřka nefunkční aplikace, která byla velice náchylná k pádům. Tento problém se vyskytoval střídavě, pokud byl jeho výskyt zastaven na emulátorech, začal se objevovat na fyzických zařízeních a opačně. Eventuálně byl nalezen přístup, který nezpůsoboval chyby ani na jedné z platforem. Toto je zde však využito jako jistá ilustrace toho, že není zcela možné říci, jestli aplikace funguje bezproblémově na všech zařízeních s verzí API 24 a výše. Je nutné také pamatovat, že není možné provést otestování všech možných konfigurací, na kterých by se aplikace mohla vyskytnout.



Obrázek 8.2: Ukázka chyby, ke které docházelo při konverzi objektu Image na bitmapu na emulátorech.

8.4 Vyhodnocení vytvořeného modelu a jeho funkčnosti v rámci mobilní aplikace

Cílem této aplikace bylo ulehčit hráčům této hry život tím, že budou moci naskenovat celou situaci v daném hexagonu místnosti. Na základě toho pak budou mít k dispozici informace o komponentech, které se zde nachází a k jakému účelu slouží. Výsledný model měl tedy být

schopen provádět klasifikaci více kategorií v rámci jednoho snímku. Za tímto účelem byly zvoleny specifické aktivační a ztrátové funkce, které jsou pro řešení těchto typů problému určené. Ve výsledku je však vytvořen model určený spíše pro klasifikaci obrázku jako jedné třídy.

I přes tyto skutečnosti je však aplikaci stále možné využít alespoň za účelem rozeznávání jen jedné z přítomných kategorií, což se v rámci této deskové hry nejvíce hodí právě v rámci velice úspěšně detekovaných hexagonů místností.

Toto je pravděpodobně způsobeno nedostatkem relevantních dat, pro vytrénování opravdu robustního modelu. Přestože má generátor dat k dispozici více vzorků pro každou z kategorií, bylo by nejspíše nutné mu poskytnout několikanásobek aktuálního počtu, pro vygenerování více diverzifikované datové sady. Tato data by pak měla být ideálně doplněna velkou datovou sadou snímků z hraní této hry. Toto by bylo ovšem velice časově náročné.

Určitým překvapením je i fakt, že přes provedené augmentace, jak v rámci generování, tak při trénování modelu, pro zajištění dobrého výkonu v různých stupních přiblížení a různých světelných podmínkách, jsou i přesto tyto situace výskytem chyb.

Dalším důvodem pro tento horší výkon může být i ta skutečnost, že si je velké množství dílků velice podobné svým tvarem a některé dokonce i svým obsahem. Tímto je tato úloha pro model daleko těžší, protože z některých komponent může být obtížné extrahovat rysy, které by pro danou komponentu byly unikátní.

8.5 Využití výsledného systému uživateli

Výsledný systém byl testován uživateli v rámci dvou po sobě následujících her deskové hry Nemesis. Aplikace byla testována čtyřmi uživateli, za využití pouze dvou různých mobilních zařízení, z důvodu nekompatibilních operačních systémů. Zde je zároveň nutné podotknout, že všichni čtyři hráči již hru minimálně párkrát hráli a byli tak obeznámeni minimálně se základními koncepty a pravidly.

Při samotné hře se projevily již dříve zmíněné nedostatky, v podobně horšího výkonu při výskytu více kategorií. I přes toto omezení bylo však možné aplikaci využít, hlavně pro identifikaci účelu různých místností. Rozpoznání místností bylo navíc nejčastějším využitím, protože s ostatními komponenty byli již hráči dobře obeznámeni. Míra využití by se pro specifické kategorie mohla u nových hráčů lišit, avšak z tohoto testu je vidět, že funkce těchto dílků místností je obecně nejnáročnější na zapamatování. To je pravděpodobně díky tomu, že je těchto dílků dvacet různých typů a skoro každý má nějaká speciální pravidla.

Ostatní komponenty byly testovány spíše okrajově, protože, jak již bylo zmíněno, to pro zkušenější hráče nemělo takový smysl. V průběhu sezení byla aplikace využívána čím dál méně, protože po několika naskenováních si daný uživatel již zapamatoval, co specifický dílek dělá.

Situace, kdy docházelo k většímu překryvu místností, se kvůli zmíněným omezením ukázaly jako mírně iritující, protože bylo nutné některé komponenty dočasně posunout, aby bylo dosaženo správného výsledku. Obecně se však kromě těchto situací aplikace ukázala jako použitelná a chybné detekce byly případně napraveny uživatelem.

Celkově byly získané výsledky z tohoto sezení v souladu s těmi již dříve uvedenými při vyhodnocení a uživatelé nebyli nuceni hledat tak často v pravidlech. Menší počet her, při kterých byl systém testován, je důsledkem dlouhé hrací doby a poněkud obtížnější přepravou této hry.

8.6 Adaptace systému pro jinou deskovou hru

Přesto, že systém nedisponuje žádnou přímou možností adaptace na jinou deskovou hru, je možné udělat několik menších změn, pro dosažení tohoto výsledku. V rámci trénování modelu je třeba dodat novou datovou sadu k dané deskové hře. Tohoto může být opět dosaženo za pomocí generování. Teoreticky by tedy měl stačit jen menší počet vzorků vybraných součástek pro vytvoření datové sady. Možnou změnou může být i počet filtrů, které obsahují konvoluční vrstvy, avšak toto by pravděpodobně bylo nutné provést jen ve speciálních případech. Samotný proces tvorby a optimalizace modelu by pak měl být dostačující pro jiné využití.

V rámci mobilní aplikace by bylo nutné změnit počet kategorií, které se vyskytují v datové sadě, pro úspěšné alokování místa pro výsledky modelu. Zároveň by bylo nutné nahradit obsah souborů, které obsahují jména, popisy a identifikátory obrázků pro všechny kategorie. S tímto by se musely změnit i zobrazované náhledy daných komponent, které jsou využity při prezentaci výsledků. Kromě těchto několika změn není třeba nijak zásadně měnit strukturu výsledného systému.

Kapitola 9

Závěr

Cílem práce bylo vytvořit systém, pomocí kterého bude uživateli usnadněna desková hra Nemesis. Toto ulehčení mělo být v podobě mobilní aplikace, díky které uživatel naskenuje herní komponenty a aplikace mu o nich následně zobrazí informace, čímž eliminuje nutnost hledání v pravidlech.

Za tímto účelem byly nastudovány různé možnosti pro zpracování obrazu a z nich byly vybrány konvoluční neuronové sítě. Následně byly získány informace o fungování těchto sítí a různé dostupné možnosti pro usnadnění jejich vývoje.

Určení oblasti, ve které by tato aplikace přinesla největší užitek, bylo poměrně jednoduché, protože mám osobně celkem hojnou zkušenost s hraním hlavně komplexnějších deskových her. Z vlastní zkušenosti tedy vím, že pochopení náročnějších titulů pro některé představuje nemalou překážku a neustálým čtením pravidel pak brzdí chod hry.

Na základě těchto informací byl navrhnut systém, který by tento problém eliminoval. Klasifikace komponent byla implementována za pomoci konvolučních neuronových sítí, přičemž data pro vytrénování tohoto modelu byla generována a augmentována kvůli nedostatku dostupných dat. Model pro klasifikaci byl zakomponován do mobilní aplikace, která umožňuje uživateli skenovat dílky pomocí jeho zadní kamery.

Výsledný systém je schopný správně identifikovat všech 35 určených kategorií. Toto bylo potvrzeno v průběhu testování modelu na testovacích datech a při uživatelském testování. Nejlépe byly detekovány hexagony místností, které byly při uživatelském testu nejčastěji analyzovanými komponenty. Jednalo se však pouze o případ, kdy se na snímku nacházela pouze jedna z kategorií. Pokud bylo kategorií přítomných více, nebylo v drtivé většině dosaženo správných výsledků a bylo třeba tyto komponenty naskenovat odděleně. V případě přímého světla nebo tmavšího prostředí byly výsledky horší než za standardních světelných podmínek i přes použitou augmentaci. Pravděpodobnou příčinou těchto nedostatků je menší počet zdrojových snímků, pomocí kterých jsou data generována. Řešením by tedy byl značně větší počet extrahovaných zdrojových snímků nebo vytvoření nové datové sady z dostatečně velkého počtu fotek přímo z procesu hraní hry.

V budoucnu by mohlo být umožněno uživatelům měnit aktivní model, aby byla aplikace využitelná pro větší počet deskových her. Jako vzdálenější cíl by bylo možné systém rozšířit o model hostovaný na serveru, ze kterého by byl vždy do daného mobilního zařízení stažen. Při každé správné klasifikaci by modelu byla odeslána data, která by následně sloužila pro jeho přetrénování, což by pomohlo v jeho adaptaci na nové prostředí a zlepšilo jeho obecnou správnost. Tyto přetrénované verze by pak byly odeslány zpátky do mobilní aplikace.

Literatura

[1] *Model optimization* [online]. [cit. 2022-04-28]. Dostupné z: https://www.tensorflow.org/lite/performance/model_optimization.

[2] *Post-training quantization* [online]. [cit. 2022-05-06]. Dostupné z: https://www.tensorflow.org/lite/performance/post_training_quantization.

[3] *Quantization Aware Training with TensorFlow Model Optimization Toolkit - Performance with Accuracy* [online]. [cit. 2022-04-28]. Dostupné z: https://blog.tensorflow.org/2020/04/quantization-aware-training-with-tensorflow-model-optimization-toolkit.html.

[4] *TensorFlow Model Optimization Toolkit — Pruning API* [online]. [cit. 2022-04-28]. Dostupné z: https://blog.tensorflow.org/2019/05/tf-model-optimization-toolkit-pruning-API.html.

[5] *Weight clustering* [online]. [cit. 2022-05-06]. Dostupné z: https://www.tensorflow.org/model_optimization/guide/clustering.

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

[7] Bandaru, R. *Pruning Neural Networks* [online]. Towards Data Science, září 2020 [cit. 2022-04-28]. Dostupné z: https://towardsdatascience.com/pruning-neural-networks-1bb3ab5791f9.

[8] Brownlee, J. *A Gentle Introduction to the Rectified Linear Unit (ReLU)* [online]. Machine Learning Mastery, leden 2019 [cit. 2022-04-28]. Dostupné z: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.

[9] Brownlee, J. *How to use Data Scaling Improve Deep Learning Model Stability and Performance* [online]. Machine Learning Mastery, únor 2019 [cit. 2022-04-28]. Dostupné z: https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/.

[10] Brownlee, J. *Multi-Label Classification with Deep Learning* [online]. Machine Learning Mastery, srpen 2020 [cit. 2022-04-28]. Dostupné z: https://machinelearningmastery.com/multi-label-classification-with-deep-learning/.

[11] Goodfellow, I., Bengio, Y. a Courville, A. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[12] Han, S., Mao, H. a Dally, W. J. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. *ArXiv: Computer Vision and Pattern Recognition*. 2016.

[13] Karajgi, A. *Evaluating Multi-label Classifiers* [online]. Towards Data Science, listopad 2021 [cit. 2022-04-28]. Dostupné z: https://towardsdatascience.com/evaluating-multi-label-classifiers-a31be83da6ea.

[14] Lazorík, J. *Mobilní aplikace pro naskenování hry Sudoku z novin a její dohrání*. Brno, CZ, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/23648/.

[15] Mandryk, R. L. a Maranan, D. S. False Prophets: Exploring Hybrid Board/Video Games. In: *CHI '02 Extended Abstracts on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, 2002, s. 640–641. CHI EA '02. DOI: 10.1145/506443.506523. ISBN 1581134541. Dostupné z: https://doi.org/10.1145/506443.506523.

[16] Molla, E. a Lepetit, V. Augmented reality for board games. In: *2010 IEEE International Symposium on Mixed and Augmented Reality*. 2010, s. 253–254. DOI: 10.1109/ISMAR.2010.5643593.

[17] Nielsen, M. A. *Neural Networks and Deep Learning*. Determination Press, 2015. http://neuralnetworksanddeeplearning.com/index.html.

[18] Singh, L. *Exploding Gradients / Vanishing Gradients* [online]. MLearning.ai, červen 2021 [cit. 2022-04-28]. Dostupné z: https://medium.com/mlearning-ai/exploding-gradients-vanishing-gradients-97251e4c65d.

[19] T. Rizov, M. T. Design of a board game with augmented reality. In: *FME Transactions*. 2019, sv. 47, č. 2, s. 253–257. DOI: 10.5937/fmet1902253R.