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

MOBILNÍ APLIKACE PRO AUTOMATICKÝ ZÁZNAM ŠACHOVÉ PARTIE

MOBILE APPLICATION FOR AUTOMATIC RECORDING OF CHESS GAMES

SEMESTRÁLNÍ PROJEKT
TERM PROJECT

AUTOR PRÁCE — **Bc. ADAM JIRUŠKA**
AUTHOR

VEDOUCÍ PRÁCE — **Doc. Ing. MARTIN ČADÍK, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2019/2020

Zadání diplomové práce

Student: **Jiruška Adam, Bc.**
Program: Informační technologie Obor: Informační systémy
Název: **Mobilní aplikace pro automatický záznam šachové partie**
Mobile Application for Automatic Recording of Chess Games
Kategorie: Zpracování obrazu

Zadání:

1. Seznamte se s problematikou mobilních aplikací pro podporu šachové hry a s metodami pro rozpoznání a zpracování obrazu na mobilních zařízeních.
2. Identifikujte činnosti během hry vhodné pro přenos do mobilního prostředí, diskutujte výhody a nevýhody.
3. Navrhněte a implementujte systém pro podporu šachové hry, který bude v reálném čase s využitím kamery zařízení zaznamenávat a usnadňovat hráčům hru.
4. S mobilní aplikací experimentujte, posuďte její vlastnosti při reálné hře uživatelskou studií i dalšími syntetickými experimenty. Diskutujte možnosti budoucího vývoje.
5. Dosažené výsledky prezentujte formou videa a plakátu, případně článku.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3 zadání.

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

Vedoucí práce: **Čadík Martin, doc. Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 20. května 2020
Datum schválení: 6. listopadu 2019

Abstrakt

Práce se zabývá tvorbou aplikace pro mobilní zařízení, která zaznamenává průběh šachové partie. K tomu využívá rozpoznávání obrazu z kamery zařízení, jež zpracovává. Šachové figurky jsou rozpoznávány pomocí neuronové sítě. Aplikace má využití v záznamu tréninkových nebo i soutěžních partií a jejich následné analýze. Pro analýzu poskytuje zápis partie ve standardní šachové notaci. K partiím si také hráči mohou dávat různé poznámky.

Abstract

This thesis is focused on making application for mobile devices, which records progress of chess game. This is achieved by image recognition on input from camera. Chess figures are classified by neural network. Usage of application is during training or real matches to record games and then for analyzing these games. For analyzing, my application offers record in standard algebraic notation. User can also add notes to every game.

Klíčová slova

šachy, OpenCV, konvoluční neuronové sítě, rozpoznávání obrazu, aplikace pro mobilní zařízení

Keywords

chess, OpenCV, convolutional neural networks, image recognition, application for mobile devides

Citace

JIRUŠKA, Adam. *Mobilní aplikace pro automatický záznam šachové partie*. Brno, 2020. Semestrální projekt. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Ing. Martin Čadík, Ph.D.

Mobilní aplikace pro automatický záznam šachové partie

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana docenta Martina Čadíka. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Adam Jiruška
3. června 2020

Poděkování

Tímto bych chtěl poděkovat vedoucímu práce panu docentu Martinu Čadíkovi za pomoc při řešení práce.

Obsah

Kapitola 1

Úvod

Šachy jsou velmi stará hra, která vznikla kolem 15. století. Vzhledem k omezenému počtu tahů se již mnoho lidí pokoušelo a stále pokouší tuto hru analyzovat a tím si zvýšit šance na výhru. Nejjednodušším způsobem analýzy je zápis šachové partie na papír a poté zkoumání jednotlivých tahů. Zkoumá se, za jaké situace jsou tahy provedeny, jaký mají účinek a také jejich příspěvek k výhře či prohře hráče. Aby si lidé mohli tyto informace snadněji vyměňovat, byla zavedena notace, kterou se všichni snaží při zápisu dodržovat. Na základě těchto notací je v moderní době vyvíjeno velké množství systémů, které hru dokáží podle této notace vizualizovat a zlepšují tak analýzu. Jsou ale také vyvíjeny systémy, které šachové partie analyzují automaticky a například s použitím neuronových sítí jsou schopny v dalších partiích určovat nejlepší možné tahy. Systém záznamu partií se ale vyvíjí mnohem pomaleji. Kromě klasického způsobu zápisu ručně jsou tu ještě dvě možnosti. Jednou z nich je hardwarové řešení přímo v šachovnici, která je speciálně upravená, aby registrovala tahy. Pro ty, co si takovou speciální šachovnici nemohou dovolit, je tu řešení pomocí rozpoznávání obrazu. V současné době jsou metody pro rozpoznávání obrazu již na velmi vysoké úrovni, ale pro dokonalé rozpoznávání je zapotřebí ideálně více úhlů záběru stejné věci. V tomto případě šachovnice.

Cílem této práce je tedy vyvinout aplikaci, která umožní zaznamenat šachovou partii pomocí mobilního zařízení. Zápis šachové partie bude proveden ve standardní algebraické notaci. V aplikaci bude možné si partie pojmenovávat a komentovat. Uživatel také bude moci si partii přehrát ve vizuální podobě. K nahrání záznamu tedy bude stačit pouze připravit šachovnici, nastavit mobil do správné pozice a začít hrát. Pokud je problém s rozpoznáním šachovnice, aplikace uživatele upozorní aby si pozici kamery zařízení upravil a šachovnice mohla být rozpoznána. Získá se tak jednoduše záznam partie, který je možno přehrát v jakémkoliv vizualizačním nástroji dostupném na internetu.

Práce je rozdělena do čtyř částí. První část je teoretická a je v ní popsána problematika rozpoznávání obrazu a jednotlivé algoritmy používané pro jeho zpracování. Dále jsem se zaměřil na konvoluční neuronové sítě používané pro klasifikaci obrazu na mobilních zařízeních a frameworky pro jejich implementaci a trénování. Také zmiňuji základní pravidla šachu a jejich využití při záznamu hry. Poslední sekcí je analýza již existujících řešení problematiky záznamu šachové partie. Druhá kapitola obsahuje návrh řešení. Jako první je zde návrh architektury systému a popis jednotlivých částí uživatelského rozhraní. Dále rozebírám postup při rozpoznávání šachovnice a její rozdělení na jednotlivá políčka. Následuje část věnovaná identifikaci šachových figurek. Tu provádím pomocí konvoluční neuronové sítě určené ke klasifikaci. V následující kapitole je popsána implementace výsledné aplikace. V této části se zabývám detaily v implementaci aplikace na Android, také konkrétní

implementací procesu rozpoznávání šachovnice a samotná sekce se věnuje klasifikaci figurek. Zvlášť je popsána tvorba datasetu, tvorba neuronové sítě a její trénování. Předposlední částí je kapitola o testování celého systému. Testování jsem rozdělil na tři části. Uživatelské testování bylo provedeno nejdříve. Následovalo testování rozpoznávání šachovnice a jednotlivých figurek. Na konci této kapitoly jsou shrnuty výsledky testů a vyhodnocení užitečnosti aplikace. Na závěr jsem uvedl některá možná vylepšení aplikace v budoucnosti, která mě během její tvorby napadla a už nebyl čas a možnost je zapracovat do výsledného řešení. Jde o mechanismy vylepšení úspěšnosti i uživatelské přívětivosti aplikace.

Kapitola 2

Teorie

V této kapitole se budu zabývat teoretickými poznatky v oblastech, kterým se moje práce věnuje. Těmito oblastmi jsou počítačové vidění, konvoluční neuronové sítě, pravidla šachu a analýza již existujících řešení. Co se počítačového vidění týče, je nutno znát rozdílné přístupy k řešení zpracování obrazu. Také blíže vysvětlím některé z algoritmů, které jsem využil ve své práci a nakonec uvedu některé z využití počítačového vidění v dnešním světě. Následuje sekce věnovaná konvolučním neuronovým sítím. Ta se skládá z popisu některých důležitých vrstev těchto sítí, přiblížení frameworků pro práci s nimi a uvedením některých neuronových sítí pro práci na mobilních zařízeních. Je také potřeba znát alespoň základní pravidla šachu a možnosti jejich využití v můj prospěch. Poté uvedu výsledky mého průzkumu již existujících řešení a aplikací v oblasti rozpoznávání šachovnice z obrazu.

2.1 Počítačové vidění

V šedesátých letech dvacátého století se začala rozvíjet nová oblast informačních technologií. Touto oblastí bylo počítačové vidění. Hlavním smyslem této technologie je nechat počítač simulovat lidské vidění a dokázat rozpoznat, co za objekty se nachází na fotografii, kterou vidí, případně na záznamu. Od té doby bylo vynalezeno mnoho a mnoho vylepšení, jejichž aplikace vedla k vylepšení této schopnosti počítačů. V současné době už jsou počítače schopny rozeznávat obličeje, jednotlivé osoby, druhy květin či zvířat a další různé objekty [12].

V této sekci se budu nejprve věnovat počítačovému vidění obecně, nastíním základní princip jeho fungování a také různé možnosti jeho využití. Poté se zaměřím na některé konkrétní algoritmy využívané pro analýzu obrazu. Bude se jednat zejména o algoritmy, které budu v práci využívat. Zde již budu konkrétněji řešit fungování a princip těchto algoritmů.

2.1.1 Principy a přístupy ke zpracování obrazu

Zpracování obrazu je samo o sobě velmi široké téma. Existuje mnoho přístupů k detekci toho, co na obraze je. Skoro každý obraz je potřeba nejprve upravit například na stejnou velikost nebo světlost. Poté je potřeba detekovat základní rysy a k tomuto účelu je spousta metod a algoritmů, jejichž využití závisí na konkrétním předmětu rozpoznávání. Když máme z obrazu detekovány například čáry nebo kružnice, dá se přistoupit k detekci samotných objektů, které se z těchto primitiv skládají.

Základní úpravy obrazu

V této kategorii se nachází hodně druhů úprav. První kategorií jsou geometrické transformace. Ty se využívají k tomu, aby rozpoznávaný obraz dosáhl nějaký předpokládaný tvar. Patří sem zvětšování a zmenšování, otočení, zkosení, odstranění perspektivy a podobně. Dalšími úpravami jsou úpravy hodnot individuálních pixelů. To vede k zvýšení jasu nebo k jeho zmenšení, úpravě kontrastu nebo změně barevné palety obrazu. Také se sem dá zařadit například prahování (thresholding). To je úprava obrazu, kde se určí hranice a podle intenzity pixelů se jejich hodnota upraví buď na 1 nebo na 0 v závislosti na pozici od té hranice. Také bych zmínil filtrování obrazu. Jsou různé typy filtrů, které pomáhají například zvýrazňovat určitá barevná spektra nebo určité intenzity pixelů. Také mohou sloužit k vyhlazení obrazu nebo k opravě některých "rozbitých"pixelů při poškození obrazu [11].

Detekce rysů

Detekce rysů se používá například k párování obrazů. Pokud jsou pořízeny dvě fotografie stejného objektu, mají určité rysy stejné, tudíž pokud tyto rysy dokáže algoritmus rozpoznat, je možné přiřadit dané fotografie k sobě. Rysů je ale potřeba velké množství. Jde zejména o hrany, body, malé oblasti a čáry. Například pro rozpoznávání rohů existuje Harrisův detektor, pro rozpoznávání hran Laplaceův operátor nebo Canny detektor. Rovné čáry se následně rozpoznávají z hran a k tomu slouží Houghova transformace.

2.1.2 Segmentace obrazu

Segmentace obrazu je rozdělování obrazu na skupiny pixelů, které spolu nějakým způsobem souvisí [23]. Nejčastěji jde o stejný objekt. K tomuto účelu se používá velké množství různých algoritmů. Například princip aktivních kontur vyznačí na obrázku pouze obrysy, nerozděluje tedy celý obraz, ale hledá čáry a hrany. Dá se využít například pro sledování (tracking) lidského těla u herních konzolí. Dalším algoritmem je "Split and Merge". Jedná se o opakované provádění prahování. Vybere se nějaká hranice, podle které se obraz prahuje a následně se udělají statistiky jednotlivých regionů. Vybere se další hranice a proces se opakuje. Na základě statistik z jednotlivých prahování se na konci vypočítá, které oblasti k sobě patří a které ne. Ne každý algoritmus funguje pro všechny typy objektů. Proto byl vyvinut například algoritmus "Mean-shift"(posun průměrů). Je založen na zkoumání větších oblastí, například pokud chceme segmentovat celý dům jako jeden objekt a ne omítku zvlášť a okna zvlášť. Tyto větší oblasti hledá a poté segmentuje. Další technikou je "Normalized cuts"(normalizované řezy). Zde se pracuje s navzájem sousedícími pixely a hledají se mezi nimi podobnosti (spojitosti). Na místech, kde jsou tyto podobnosti malé se udělá řez. Tyto řezy poté rozdělí obraz na segmenty [23].

Modernějším způsobem segmentace obrazu je využití konvolučních neuronových sítí. Jednou z nich je Deeplab-v3. Jedná se o síť vyvíjenou společností Google. Ke svému trénování potřebuje speciálně upravený dataset, jako například MS-COCO 2.2.1. Na základě těchto připravených datasetů a hodin trénování dokáží takovéto neuronové sítě sémanticky segmentovat různé objekty jako například lidi, psy, letadla, auta a podobně.

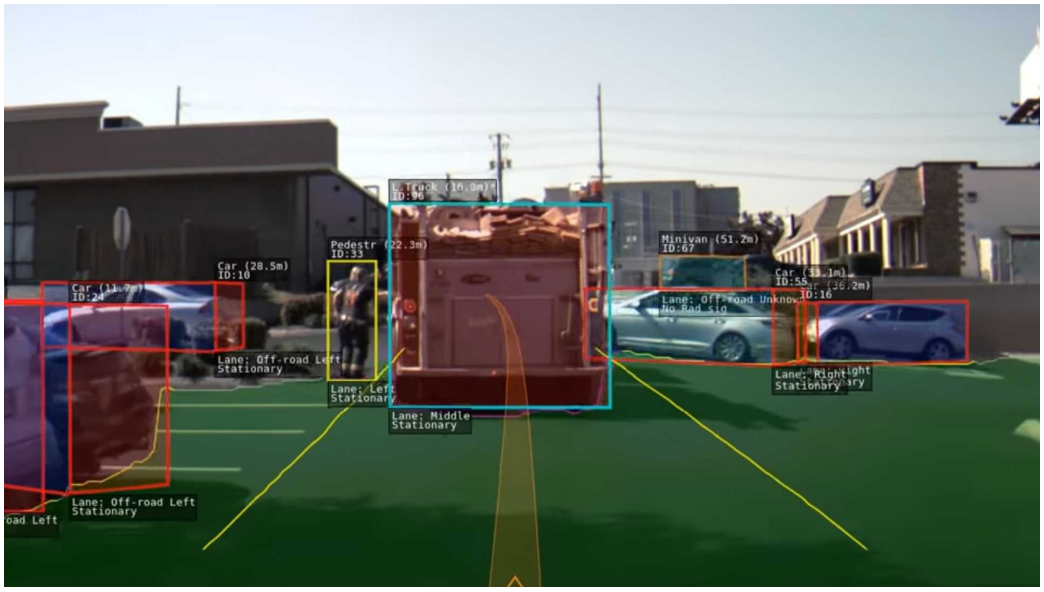
2.1.3 Využití počítačového vidění

Počítačové vidění se využívá čím dál více i v každodenním životě. Jednou z nejvíce se rozvíjejících oblastí je autonomní řízení osobních automobilů. Dále se využívají k detekci

osob v bezpečnostních kamerách, při rozpoznávání například hub nebo rostlin a podobně. Na jednotlivé způsoby využití se zaměřím podrobně níže.

Autonomní řízení osobních automobilů

Jedna z nejzajímavějších a nejrychleji se vyvíjejících oblastí. Hlavní zásluhu na tom má společnost Tesla, jejíž autopilot už je schopen jízdy bez zásahu řidiče [4]. Velkou část v jeho schopnostech hraje právě počítačové vidění. Auto je osazeno několika kamerami a obraz z nich musí být velmi rychle a efektivně segmentován, aby počítač věděl, kudy jet dál, kde se blíží nebezpečí a podobně (obrázek 2.1). Hrají zde roli neuronové sítě, které se stávají lepší a lepší s každým kilometrem, který jakýkoliv řidič Tesly s autopilotem ujede.



Obrázek 2.1: Obrázek z kamery automobilu - jednotlivé objekty jsou rozpoznány a označeny. Převzato z [15].

Detekce poznávacích značek

Jde o velmi jednosměrnou činnost, která ale nachází uplatnění v mnoha případech. K rozpoznání značek se využívají neuronové sítě. Umět lokalizovat poznávací značku auta a následně z ní extrahovat textový obsah se ukázalo jako velmi užitečné například při analýzách dopravy. Další využití je při sledování dodržování pravidel silničního provozu s následným vymáháním pokut. Při krádeži vozidla je také snadnější ho vyhledat na kamerami sledovaných křižovatkách a cestách. Jako poslední bych zmínil možnost flexibilního vjezdu a výjezdu z různých parkovacích domů [10].

Rozeznávání rostlin

Další případ použití neuronových sítí. Ať už jde o rozpoznávání čehokoliv, neuronové sítě se umí z velkého množství dat velmi dobře naučit rozpoznávat objekty. Vybral jsem rozeznávání rostlin, ale existují i aplikace pro rozpoznávání hub, zvířat nebo třeba ručně psaného textu. Jde většinou o aplikace pro širokou veřejnost, která není tolik vzdělaná a neví, kterou houbu například právě v lese našli.

Zdravotnictví

Využití této technologie ve zdravotnictví má dva různé typy aplikací. Speciálně vytrénované neuronové sítě mohou zastávat lékaře v rozpoznávání CT skenů [4]. V současnosti už existují sítě, které dokáží rozpoznat nebezpečný nádor s větší pravděpodobností, než lékař. Druhý způsob využití je v pomoci nemocným. Zrakově postižení nemohou vidět, co je na obrázku před nimi, ani nemohou číst texty. Facebook proto vyvinul neuronovou síť, která dokáže rozpoznat některé věci na obrázku a slepému uživateli je přečíst. Dokonce zvládne rozpoznat i náladu lidí. To umožňuje těmto handicapovaným uživatelům lépe komunikovat v dnešním světě sociálních sítí [4].

Reklama a obchod

Představa, že abychom se v obchodě viděli v oblečení, které si chceme koupit, bez jeho vyzkoušení, je velmi lákavá. Některé obchody již vyvinuly speciální zrcadlo, v němž je zabudovaná kamera a displej (obrázek 2.2). Pomocí toho dokáže detekovat osobu a "navléknout"na ni oblečení podle požadavků. Zákazník si tedy může zkoušet oblečení rychleji než kdy dříve. Dalším využitím by mohlo být například zobrazování reklamy relevantní k osobám, které před digitálním plakátem stojí. Kamera by spočítala, jaký je průměrný věk lidí nebo jejich pohlaví a podle toho zobrazila nejlepší možnou reklamu.

Obrázek 2.2: Chytré zrcadlo - možnost výběru barvy šatů bez nutnosti převlékání. Převzato z [21].

2.1.4 Algoritmy využívané v počítačovém vidění

Nyní rozeberu některé z výše zmíněných algoritmů detailněji. Půjde o algoritmy, které budu ve své práci využívat pro rozpoznání šachovnice.

Laplaceův operátor

Je to operátor, který se ve zpracování obrazu používá k nalezení a zvýraznění hran. Matematicky je Laplaceův operátor definovaný jako divergence gradientu [14]. Znamená to, že tam, kde je v obrazu malá změna stupňů šedi se snaží prostor více vyhladit. Tam, kde se intenzita stupňů šedi mění více skokově, zvýrazní hranu. Pro funkce je operátor definován jako

$$\Delta f = \nabla^2 f = \nabla \cdot \nabla f.$$

Ve zpracování obrazu máme ale diskrétní hodnoty pixelů. Je tedy potřeba mít konvoluční jádro, které dokáže aproximovat hodnoty druhé derivace. Nejčastěji používaná jsou jádra na obrázku 2.3.

0	1	0
1	−4	1
0	1	0

1	1	1
1	−8	1
1	1	1

−1	2	−1
2	−4	2
−1	2	−1

Obrázek 2.3: Nejčastěji používaná konvoluční jádra při aplikaci Laplacova operátoru

Houghova transformace

Houghova transformace [7] se využívá zejména pro detekci čar, je ale možné ji využít i pro detekci elips a kružnic. Pracuje v polárním systému souřadnic, de je každá přímka definována pomocí vzdálenosti přímky od počátku souřadného systému a úhlu vektoru směřujícího k nejbližšímu bodu přímky. Pomocí těchto proměnných se dá rovnice přímky zapsat následovně:

$$y = -\frac{\cos(\theta)}{\sin(\theta)}x + \frac{r}{\sin(\theta)}$$

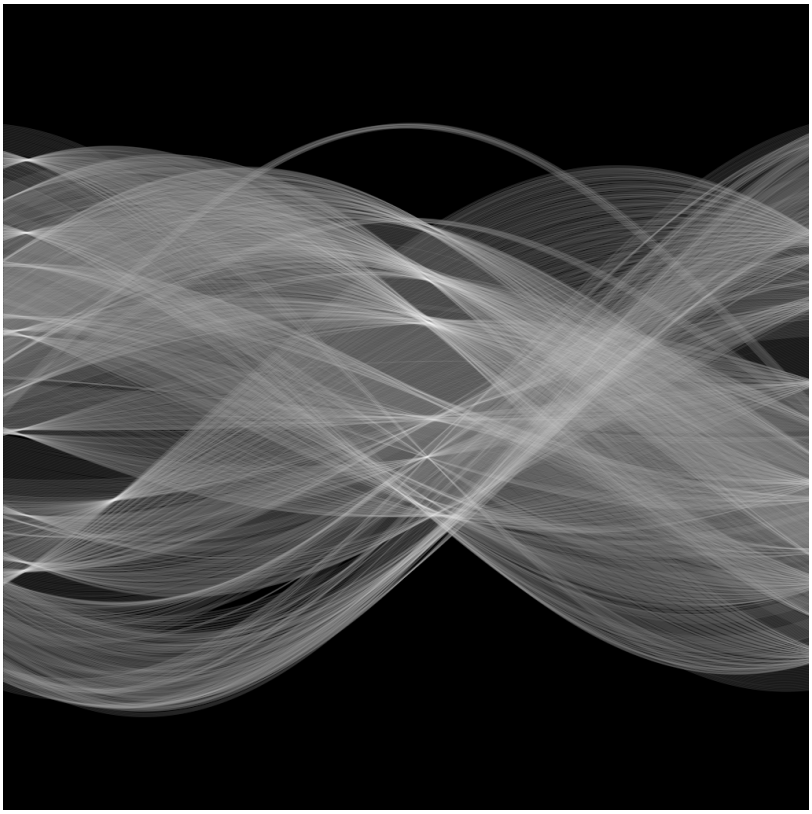
Po úpravě je možné vyjádřit r:

$$r = x \cdot \cos(\theta) + y \cdot \sin(\theta)$$

Tento algoritmus pro každý bod, který by mohl být součástí nějaké čáry spočítá hodnotu r a θ pro všechny přímky, které tím bodem prochází. Výsledkem tohoto je sinusoid. Předpokládejme, že na v obraze existují dva body. Algoritmus tedy vytvoří dva sinusoidy a dá je do jednoho grafu. Místo, kde se protnou, určuje, jaké hodnoty je potřeba dosadit za r a θ, aby vznikla přímka procházející těmito dvěma body. Ve skutečnosti je těchto bodů mnohem více a spojením jejich sinusoidů vzniká graf jako na obrázku 2.4. Body, které v něm jsou patrné, jsou právě ty přímky, které nakonec budou detekovány.

2.2 Neuronové sítě pro klasifikaci obrazu

Pro klasifikaci obrazu se využívají konvoluční neuronové sítě (CNN). To je třída neuronových sítí, které využívá hluboké učení pro rozpoznávání různých typů objektů na obrázcích. Takovéto sítě fungují tak, že na vstupu mají obrázek, neboli matici pixelů. Jejich výstup je buď třída (například "kolo") nebo to, s jakou pravděpodobností se jednotlivé třídy na



Obrázek 2.4: Graf všech možných přímek procházející všemi body - místa s nejvíce průsečíky jsou detekované čáry. Převzato z [27].

obrázku nacházejí. Toho se dá využít, pokud chcete uživateli nabídnout více možností v případě, že by síť chybovala při určení v první řadě. [2] Využití těchto sítí je opravdu velké. Některé způsoby už jsem zmiňoval v sekci 2.1.3.

Konvoluční neuronové sítě se skládají z několika typů vrstev. První je konvoluční vrstva. Úkolem těchto vrstev je extrahovat z obrazu rysy. Vstupem je tenzor a na výstupu se objevují právě tyto rysy. Dalším typem vrstvy je ReLU vrstva. Tyto vrstvy se využívají mezi konvolučními vrstvami a jejich úkolem je vnést nelinearitu do vyhodnocování. Oproti dříve používaným funkcím jsou velmi rychlé na trénování. Také existují tzv. pooling vrstvy. Ty slouží k diskretizaci obrazu, což je sdružování pixelů do větších oblastí, aby se sjednotily informace o rysech daného regionu. Umožňuje to také bránit se přetrénování [2, 18].

2.2.1 Existující datasety pro trénování neuronových sítí

Pokud vytváříme neuronovou síť pro nějakou obecnou úlohu, jako rozpoznávání tváří a textu, nebo si chceme pouze vytvořit nějaký základ pro naši konkrétní úlohu, existují již připravené datasety dostupné online. Samozřejmě existují i datasety vytvořené pro konkrétní úlohy, ale už je mnohem těžší je najít. Například dataset pro rozpoznávání šachových figurek jsem najít nedokázal. Důvodem může být i to, že různých typů figurek je spousta a některé z nich nemají ani žádné společné rysy. Pro přehled zde uvedu některé nejznámější datasety pro trénování neuronových sítí.

MNIST

MNIST je jeden z nejznámějších datasetů vůbec. Je to způsobeno tím, že se zaměřuje na úlohu, kde spousta lidí začíná svoje první pokusy. Jde o ručně psaná čísla a jejich rozpoznávání. Pro trénink práce se sítěmi je to ideální úloha. Dataset se počtem tříd velmi podobá tomu pro šachy. Celkem existuje 10 tříd (číslic). Mezi tyto třídy je rozděleno 70 000 obrázků ručně psaných číslic. Z tohoto celkového počtu je 10 000 určeno pro testování a zbytek pro trénování [8].

ImageNet

První dataset, s kterým jsem se dostal do kontaktu já. Síť MobileNet, kterou jsem používal, byla předtrénována právě na tomto datasetu. Složení je zde velmi odlišné od předchozího zástupce. Každý obraz obsahuje tzv. bounding boxy a ty jsou označeny názvem objektu, který ohraničují. Celkový počet fotografií je přibližně 1 500 000. Je tedy mnohem obecnější, než MNIST a tím pádem je určen pro mnohem víc případů použití [8].

MS-COCO

Tento dataset je specializovaný na segmentaci obrazu. Také slouží k detekci objektů nebo k popisu obrázků. Skládá se z obrazů, které mají k sobě přiděleny jejich segmentované kopie. Každý segment je popsaný podle jeho obsahu. Je tedy možné trénovat síť na různé typy úkolů. Velikost tohoto datasetu je 330 000 fotografií na kterých je celkem 80 různých kategorií objektů. Každá fotografie obsahuje 5 různých objektů [8].

2.2.2 Frameworky pro práci s neuronovými sítěmi

Práce se neuronovými sítěmi není vůbec jednoduchá. Hlavním programovacím jazykem je zde Python. Aby s nimi mohli pracovat i lidé, kteří nemají podrobné porozumění implementačním detailům, existuje několik frameworků, které tuto práci dělají daleko přívětivější a jednodušší. Ve svém rozboru popíšu zejména frameworky, které budu ve své práci používat. Vybral jsem si Tensorflow a Keras. Důvodem bylo, že nejsem velký odborník na práci s neuronovými sítěmi a kombinace těchto dvou knihoven poskytuje velmi snadné prostředí pro vytváření a trénování neuronových sítí. Je také k dispozici velmi dobrá dokumentace, takže se snadno řeší i problémy, které při práci nastávají.

TensorFlow

TensorFlow [1] je open-source knihovna pro práci s neuronovými sítěmi a hlubokým učením. Je vyvíjen Googlem už od roku 2011 jako DistBelief a byl vydán v roce 2015 již pod názvem TensorFlow. Základní verze byla v C++ a Pythonu a uměla pouze část toho, co umí současná. Postupem času byla přidána podpora dalších jazyků, jako JavaScript nebo Java. Také přibyla podpora učení neuronových sítí na mobilních zařízení (EdgeTPU a TensorFlow Lite). V loňském roce přibyla i podpora pro hluboké učení v grafice, a to v TensorFlow Graphics.

Keras

Keras [13] také slouží jako knihovna pro práci s neuronovými sítěmi, ale ještě dále rozšiřuje funkce TensorFlow. Dokáže běžet i na základech jiných knihoven, jako například Microsoft

Cognitive Toolkit, R nebo Theano. Snaží se být co nejvíce uživatelsky přívětivá, a tím pádem není divu, že se mi pro moje první setkání s hlubokým učením zalíbila. Je napsaná v Pythonu, ale má určitou podporu i pro Javu, což je pro mě další výhoda.

Tensorboard

Tensorboard [25] je součástí knihovny Tensorflow. Zmiňuji ho zde proto, že jsem ho využíval pro získání přehledných výsledků z trénování sítí. Je to vizualizační toolkit, který v reálném čase získává data z trénování a zapisuje je do logů. Tyto složky následně Tensorboard dokáže zobrazit v podobě interaktivních grafů, které jsou přehledné, ale přesto obsahují naprosto všechna data z trénování.

2.2.3 Neuronové sítě pro mobilní zařízení

Výpočty, které neuronové sítě provádí, nejsou vůbec snadné. Jsou velmi náročné a hlavně jich je mnoho. Když tedy chceme používat takovou klasifikační síť na mobilním zařízení, je jasné, že nemůžeme vzít kteroukoliv síť, která funguje v prostředí o mnoho výpočetně silnějšího počítače a spustit ji na mobilu. A už vůbec není jednoduché (hlavně z časového hlediska) takovou síť natrénovat. Toto vše je ale možné, pokud při tvorbě sítě přihlédneme k faktu, že se bude používat na mobilu a její architekturu tomu odpovídajícím způsobem přizpůsobíme. Vznikají tím sítě, které zabírají velmi málo místa, nepotřebují zdaleka tolik zdrojů jako ty velké a jejich úspěšnost klasifikace není vůbec špatná.

MobileNet

Jako prvního zástupce těchto sítí jsem vybral sítě MobileNet. Důvody jsou jednak to, že mi byly doporučeny k prozkoumání jako první a také to, že jsem si je nakonec vybral. Vybral jsem si konkrétně síť MobileNetV2. Oproti první verzi má tato verze mnohem větší úspěšnost předpovídání. Také je efektivnější, což by se u druhé verze dalo předpokládat. Sítě tohoto typu fungují na principu hloubkově dělitelných konvolucí. Na datasetu ImageNet, což se používá často jako benchmark konvolučních neuronových sítí dosáhly velmi slušné úspěšnosti.[9]

Google MNasNet

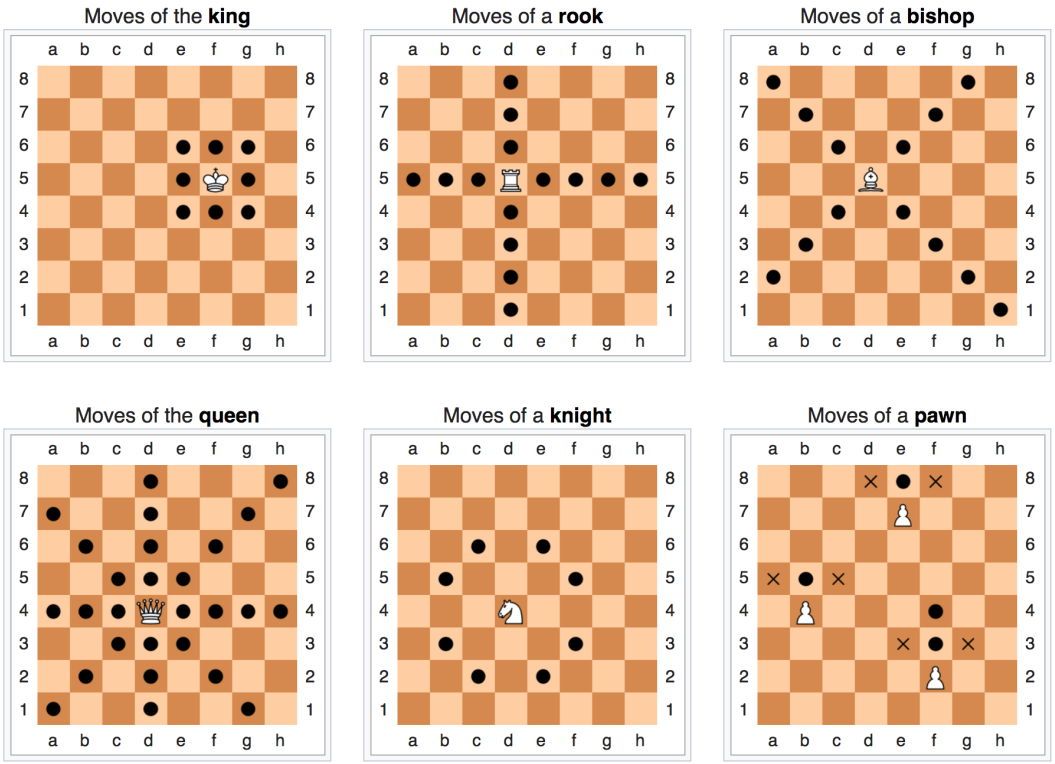
MNasNet je architektura vytvořená stejnou skupinou lidí jako TensorFlow, a to Google Brain. Ta volí zcela jiný přístup než ostatní sítě. Využívají klasickou neuronovou síť, tu trénují a spouští na mobilním zařízení a měří výkon a latenci pomocí TensorFlow Lite. Na základě těchto dat následně optimalizují danou síť pro získání lepších výsledků. Hlavním odlišením je měření latencí na reálných zařízeních. Jejich úspěšnost je velmi podobná úspěšnosti sítí MobilenetV2.[20]

ShuffleNet

Tato konvoluční neuronová síť se chlubí tím, že má velmi malé nároky na výpočetní výkon. Při tom ale přesnost sítě zůstává přibližně stejná. Dociluje toho dvěma novými operacemi, které mají snížit výpočetní náklady a síť tím zrychlit. Jednou z nich je promíchání kanálů, anglicky *channel shuffle*, z čehož síť i dostala své jméno [28].

2.3 Pravidla šachu a jejich využití

Při rozpoznávání šachové partie mohu pravidla šachu využít ve svůj prospěch. Ty totiž jasně definují, jaké jsou další možnosti tahu hráčů. Nemůže se tedy stát, že by se nějaká z figurek pohnula jinak, než je dáno v pravidlech. Dalším z pomocných faktorů je to, že se hraje na omezeném prostoru. Šachovnice má rozměry 8x8 čtverců. Na začátku jsou v prvních dvou řadách od obou hráčů umístěny bílé figurky na jedné straně, černé na straně druhé. Každý typ figurky se může pohybovat pomocí dopředu určené množiny tahů. Například věž se může pohybovat pouze vertikálně a horizontálně, a to o kolik políček hráč chce. Možné tahy jednotlivých typů figurek jsou vidět na obrázku 2.5.



Obrázek 2.5: Přehled povolených tahů pro jednotlivé typy figurek. Převzato z [3].

Těchto pravidel se dá využít při zápisu tahů. Pokud budu mít jakýkoliv stav uprostřed hry, existuje pouze omezený počet stavů, do kterých je možné se příštím tahem dostat. Aplikováním této strategie se dá předejít chybným rozpoznáním některých figurek, které byly při tahu například zakryty jinou figurkou. Je totiž jasné, na která políčka se které figurky mohou dostat během jednoho tahu. Jedním ze základních pravidel je, že vždy táhne pouze jeden hráč a jejich tahy se střídají. Tím pádem může systém dopředu očekávat, která barva figurek změní své rozestavení. Systém se může tedy soustředit střídavě na rozpoznávání bílých a černých figurek. Další pravidla, která se využijí, budou pravidla o pohybu figurek. Porovnáním dvou po sobě jdoucích stavů šachovnice se zjistí, která figurka se pohnula a také kam se pohnula. Pokud ale tento tah není v pravidlech povolený, může systém hráče upozornit, že nedošlo ke správnému rozpoznání figurky a hráči mohou s některými figurkami lehce pohnout, aby kamera dostala lepší pohled.

Standardní šachová notace

Pro zápis tahů se hojně využívá právě tato notace, jejíž anglická zkratka je PGN (Portable game notation) [19]. Jedná se o formát textového souboru, který je jednoduše zpracovatelný a dostatečně krátký. Na začátku takového souboru je sedm povinných tagů. Jedná se o název události, místo události, datum, číslo kola, jména obou hráčů a výsledek hry. Následuje zápis tahů. Zde je formát takový, že je napsáno číslo kola, tah bílých figurek a tah černých figurek. Pravidla zápisu tahu jsou dána takto. První je typ figurky. Jedná se o zkratku v angličtině, takže písmena K (king), Q (queen), R (rook), B (bishop), N(knight). Pokud táhne pěšec, je písmenko vynecháno. Poté jsou souřadnice políčka, na které figurka šla. V některých případech, když mohou dvě stejné figurky na stejné políčko, připíše se před tyto souřadnice ještě z kterého sloupce figurka šla. Výsledný tah střelce na C4 tedy bude zapsán jako Bc4. Pokud probíhá rošáda, je podle jejího typu zapsáno buď "O-O"nebo "O-O-O". V případě, že hráč vezme tahem soupeřovu figurku, je zapsáno ještě "x"mezi typ figurky a souřadnice políčka. Příklad jednoho ze složitějších zápisů, kde mohly obě věže udělat stejný tah a jedna z nich vzala soupeřovu figurku na políčku E5, by tak byl Rxge5. Do této notace se dají přidávat komentáře, které je nutno oddělit od ostatního zápisu složenými závorkami.

2.4 Existující řešení a aplikace

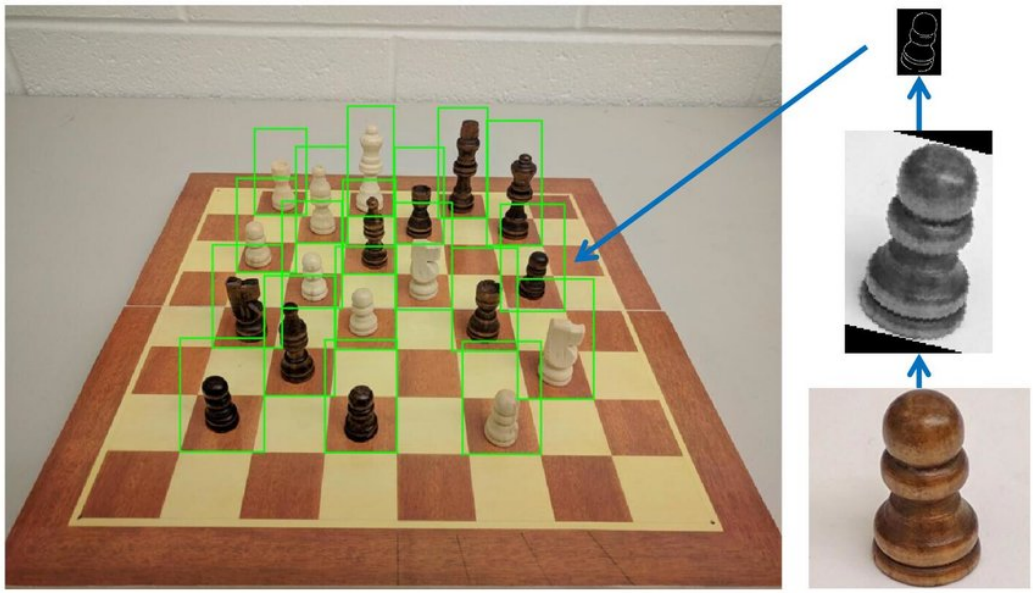
Rozpoznávání rozestavení šachových figurek je vcelku známý problém, a proto není překvapení, že existuje mnoho řešení, které se tímto problémem zabývají. Většina z těchto řešení jsou ale pouze výzkumy nebo pokusy. Nenašel jsem žádnou aplikaci, která by sloužila přímo k účelu záznamu šachové partie na mobilním zařízení. Byly už vytvořeny systémy, které dokáží rozpoznávat současný stav šachové partie a fungují jako soupeř pro kohokoliv, kdo zrovna chce hrát proti umělé inteligenci. Další systémy se soustředí například na rozpoznávání stavu partie pouze podle fotky, kterou přetransformují do digitální podoby šachovnice. V této části tedy jednotlivá řešení rozeberu. V analýze se zaměřím zejména na postup, kterým provádí dílčí části rozpoznávání a zaměřím se jejich silné stránky a případné neúspěšné pokusy, což přispěje k lepšímu návrhu mojí aplikace.

Chess Piece Recognition Using Oriented Chamfer Matching

Tento projekt [24] byl představen na Zimní konferenci aplikací počítačového vidění WACV2018 a je velmi zajímavý tím, že nevyužívá neuronové sítě pro rozpoznávání figurek. Místo toho figurky rozpoznává pomocí šablon. Každá figurka je v databázi nafocena z 12 různých výškových úhlů. Při rozpoznávání je vypočítán normálový vektor každého políčka a podle projekční matice je obraz políčka vždy narovnán, aby mohlo proběhnout rozpoznání podle šablony. Protože zde neprobíhá rozpoznávání figurek pomocí neuronové sítě, bude moje řešení podobné pouze v části rozpoznávání šachovnice. Ukázka fungování je na obrázku 2.6 [24].

Stonewall project

Stonewall [6] je projekt studenta na University of Canterbury. Jeho použití můžete vidět na obrázku 2.7 V rámci předmětu Počítačové vidění se vytvořil systém, který pomocí kamery snímá šachovnici a simuluje jednoho hráče. Simulace chování hráče je zařízena pomocí enginu Stockfish, což je jeden z nejrozšířenějších šachových enginů v dnešní době. K rozpoznávání obrazu se využívá kamera umístěná přímo nad šachovnicí v dané výšce, což hodně

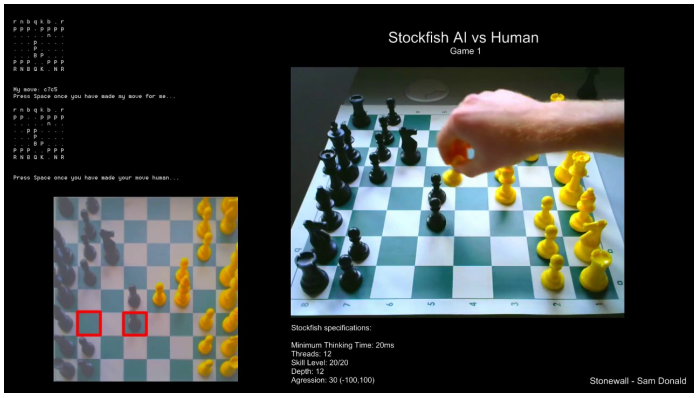


Obrázek 2.6: Chamfer matching - proces rozpoznání figurky a její vyznačení v obraze. Převzato z [24]

usnadňuje rozpoznávání políček na šachovnici. Aplikace tedy není využitelná pro širokou veřejnost, protože potřebuje mít jasně definovanou pozici kamery [6].

Chess Position Recognition from a Photo

Toto je diplomová práce studenta z Masarykovy Univerzity v Brně [17]. V rámci svojí práce vytvořil aplikaci, která dokáže rozpoznat pozice figurek na šachovnici z fotografie. Neřeší tedy na rozdíl ode mě tahy a také mírné změny pozice kamery. K rozpoznávání figurek používá template matching, ale jiným způsobem, než tomu bylo u první aplikace. Pomocí dat získaných z rozpoznání šachovnice si vyrenderuje figurku na dané místo z 3D modelu a pomocí té poté provádí porovnání s výřezem z fotografie [17].



Obrázek 2.7: Stonewall - v levé části obrázku je vidět rektifikovaná šachovnice a rozpoznaný tah. Nahoře je současný stav šachovnice. Převzato z [22].

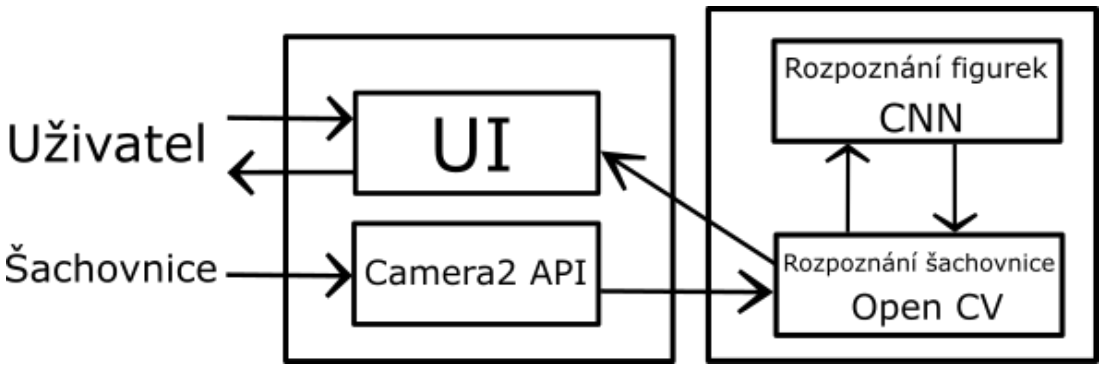
Kapitola 3

Návrh řešení

V této kapitole je popsán celkový návrh aplikace a řešení problémů, které se při rozpoznávání vyskytují. Je zde popsáno uživatelské rozhraní mobilní aplikace včetně všech možností, které bude nabízet. Dále zmíním návrh architektury systému, návrh práce s neuronovou sítí, pomocí které se rozpoznávají figurky a dataset připravený pro tuto síť.

3.1 Architektura systému

Základní rozdělení aplikace by se dalo určit takto. Systém je rozdělen na dvě hlavní části. Jedna část komunikuje s uživatelem a okolím a druhá část provádí výpočty, rozpoznávání a klasifikace. V obrázku 3.1 jsem zaznamenal dle mého názoru nejdůležitější části mnou navrženého systému. Jde o čtyři část. Dvě z nich jsou pro zajištění rozpoznávání a výpočtů na pozadí a druhé dvě jsou pro komunikaci s uživatelem a interakci s prostředím.



Obrázek 3.1: Nákres architektury systému

Nejprve bych se zaměřil na část backendovou. Zde jsou v obrázku vidět dva bloky. Jeden z nich slouží k předúpravě obrazu z kamery a následnému rozpoznání šachovnice. Také zde probíhá příprava vstupů pro neuronovou síť a také zpracování toho, co neuronová síť vrací. Hlavní knihovnou používanou v této části je OpenCV. Druhým blokem je blok klasifikace figurek. Jde o neuronovou síť, která přijímá obrázky jednotlivých políček a vrací typ figurky, který se na daném políčku nachází. Tato část je implementována mimo zbytek architektury a do systému je potom vložen pouze model sítě a natrénované váhy.

Ve frontendové části jsou také dva bloky. Abych dokončil rozbor rozpoznávání šachovnice, začnu blokem kamery. Zde bude nejvíce využíváno Camera2 API, které umožňuje

ovládat kameru zařízení Android. Ta bude využita k snímání šachovnice v určitých časových intervalech. Poslední částí je samotné uživatelské rozhraní aplikace. Je to jediná část, která jakýmkoliv způsobem komunikuje s uživatelem. Zastřešuje zobrazování všech obrazovek, zobrazuje obraz z kamery a přijímá například stisky tlačítek.

Celý proces bude fungovat následovně. Uživatel pomocí interakce s uživatelským rozhraním začne nahrávat svoji hru. Blok pro snímání následně udělá snímek šachovnice a ten zašle do backendové části. Zde se rozpozná šachovnice, připraví vstupy pro neuronovou síť a ty se poté zasílají do bloku klasifikace figurek. Dílčí výstupy jsou vraceny zpět a zpracovávány do konečné podoby stavu šachovnice. Poté je vypočítán poslední udělaný tah a ten je zaslán zpět na uživatelské rozhraní.

3.2 Návrh uživatelského rozhraní

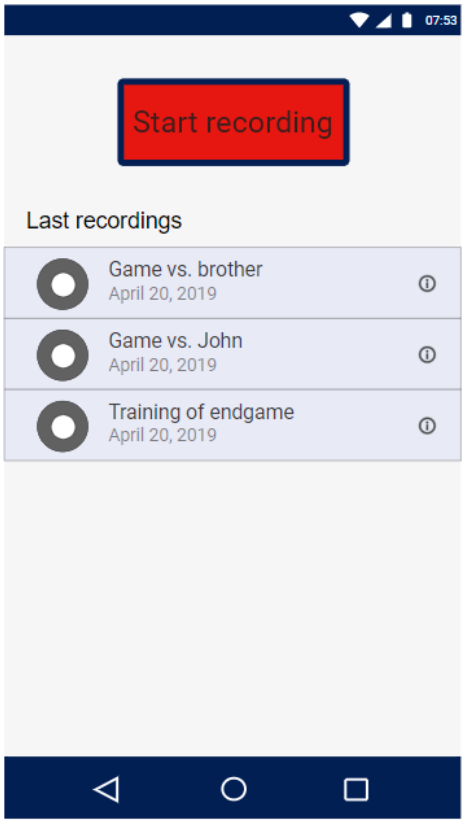
Uživatelské rozhraní aplikace bude velmi jednoduché. Hlavními obrazovkami budou úvodní obrazovka aplikace a obrazovka konkrétní nahrané hry. Spuštění nahrávání jsem navrhl tak, aby uživateli stačilo pouze jednou kliknout a už byl na obrazovce pro nahrávání. K přechodu na obrazovku nahrávání slouží tlačítko *Začít nahrávat*. Kromě tohoto tlačítka bude úvodní obrazovka obsahovat ještě další užitečné odkazy. Další důležitou funkcí je zobrazení nahrané hry. Ta se zobrazí tak, že uživatel vybere ze seznamu nahraných her tu, kterou si chce prohlédnout. Poté se zobrazí obrazovka detailu nahrané hry. Při nahrávání hry bude na obrazovce vidět obraz z kamery, aby bylo možné mobilní zařízení nastavit do správné pozice. Jako hlavní barvu aplikace jsem zvolil tmavě modrou. Do této barvy tedy budou laděny všechny ovládací prvky. Pro návrh uživatelského rozhraní mi posloužila aplikace Moqups [16].

Úvodní obrazovka

Návrh úvodní obrazovky je vidět na obrázku 3.2. Po spuštění aplikace uživatel uvidí právě tuto obrazovku. Je tedy nutné, aby obsahovala všechny důležité informace a umožňovala dostat se ke všem funkcím aplikace co nejrychleji, tedy na co nejméně kliknutí. Stránce dominuje tlačítko pro spuštění nahrávání, tedy nejpravděpodobnější úkon, kvůli kterému si uživatel aplikaci spustí. Tlačítko bude červené, aby se podobalo klasickým tlačítkům pro zahájení nahrávání na kamerách a fotoaparátech. Lemuje ho tmavě modrá barva, aby také zapadlo do barevného schématu aplikace. Pod tímto tlačítkem je seznam naposledy nahraných her. V seznamu je u každé hry možno vidět vlastní název hry a datum, kdy byla hrána. To by mělo uživateli posloužit k jednoznačné identifikaci hry, kterou chce zobrazit.Také se zde nachází tlačítko nápovědy, kde je uživateli vysvětleno, jak má aplikaci ovládat. Jedná se zejména o proces nahrávání hry.

Detail nahrané hry

Návrh této obrazovky je na obrázku 3.3. Při návrhu detailu jsem nijak nevybočoval ze zvyklostí ostatních aplikací. Stránce dominuje název hry, který si uživatel zvolil. Následuje datum, kdy byla partie hrána. Po každé hře si může uživatel zadat, kdo hru vyhrál. To bude také zobrazeno v detailu. Následují poznámky, které si uživatel může volitelně také přidávat k jednotlivým hrám. Poté již zbývá jen hra zapsaná ve standardní algebraické notaci. Notace je zapsaná ve formátu PGN (Portable Game Notation) tak, aby mohla být vložena do jakéhokoliv analyzátoru šachových partií, který tento formát podporuje. Pod



Obrázek 3.2: Wireframe hlavní obrazovky aplikace

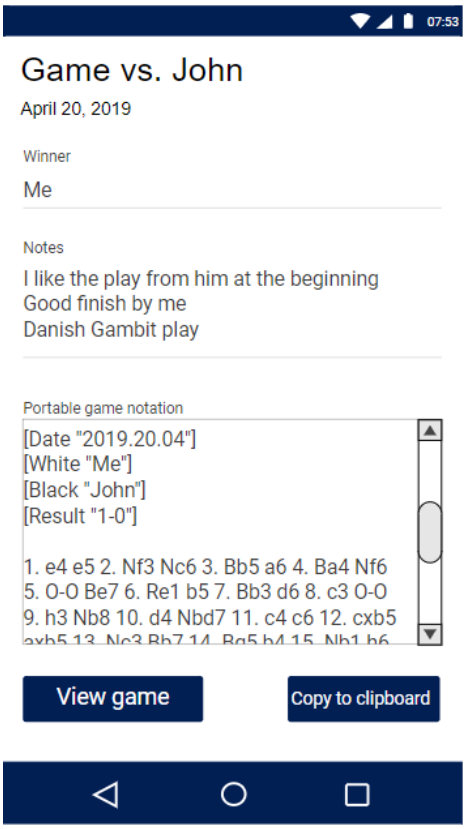
zápisem hry jsou k dispozici ještě tlačítka pro zkopírování notace do schránky zařízení a pro přehrání hry v digitální podobě.

3.3 Rozpoznávání obrazu

Vzhledem k tomu, že šachy jsou velmi pomalá hra (vynechám-li jejich bleskovou verzi), bude aplikace zpracovávat obraz jednou za určitou dobu. Tu bude nutné ještě pomocí experimentování určit, ale předběžně předpokládám každé 2 vteřiny. Takto nezmeškáme žádný tah a zároveň nebude zařízení zbytečně přetěžované kvůli opakovaným zbytečným výpočtům. Každý obraz bude následně nutné zpracovat pomocí algoritmu popsaného v sekci 3.4.2.

V mém řešení budu používat knihovnu OpenCV pro Android. Je to knihovna určená zejména k práci v oblasti počítačového vidění v reálném čase. Je navržena velmi efektivně. To znamená, že při správném využití jejich funkcí by neměl být žádný problém s výkonem na mobilních zařízeních střední a vyšší třídy, ovšem to bude teprve předmětem testování. V případě horšího výkonu bude nutné provádět optimalizace na úkor kvality rozpoznávání. Pro první část práce budu vyvíjet aplikaci v počítačovém prostředí, tudíž nebudu muset dbát tolik na efektivitu kódu. Knihovnu budu využívat pro rozpoznávání hran a čar. Dále rozpoznání rohů šachovnice a přípravu políček pro rozpoznávání jednotlivých figurek [26].

Poté, co budou zpracovány dva po sobě jdoucí snímky, bude nutné vytvořit zápis tahu, který jeden z hráčů provedl (pokud nějaký byl proveden). Zde přijdou na řadu funkce, které ještě upraví to, co bylo rozpoznáno přímo z obrazu. Neuronové sítě dokáží klasifikovat



Obrázek 3.3: Wireframe detailu nahrané hry

s určitou pravděpodobností. Proto mohu těmto funkcím poslat více možností a pomocí znalostí pravidel šachu toto rozpoznání upravit.

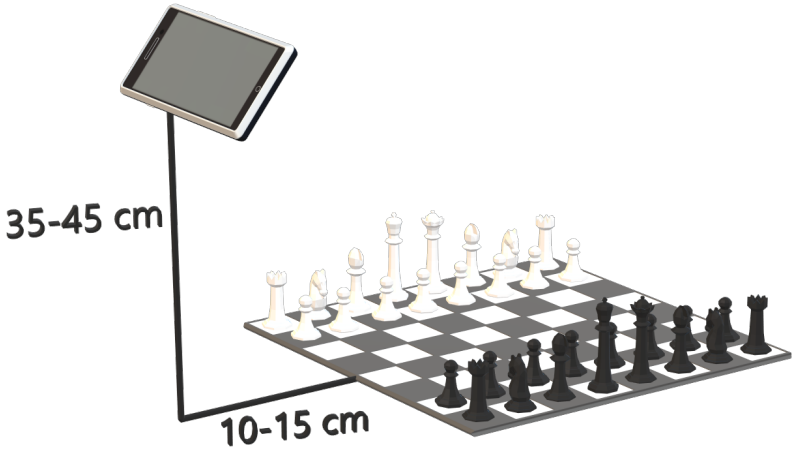
3.4 Rozpoznávání šachovnice

Rozpoznávání šachovnice je prvním klíčem k celkovému úspěchu. Pokud bude úspěšnost tohoto kroku velmi slabá, mnohokrát se ani aplikace nedostane k samotnému rozpoznávání figurek. Proto je důležité mít velmi přesné rozpoznávání. Principiálně se jedná o použití funkcí OpenCV ve správném pořadí a se správnými parametry. Mojí prací je přijít na to, jaké funkce bude vhodné pro tento úkol použít a vyladit správně všechny parametry těchto funkcí.

3.4.1 Podmínky pro úspěšné rozpoznávání

Nejprve bych zde uvedl podmínky, které je potřeba splnit, aby aplikace správně šachovnici rozpoznala. Jedná se o správnou pozici zařízení a okolní podmínky. Nelze totiž očekávat, že šachovnice bude rozpoznána za každých okolností.

Nejdůležitější podmínkou je správné umístění zařízení. Přímo v aplikaci je zobrazeno, v jaké části obrazu by se měla šachovnice nacházet. Zařízení by mělo být umístěno na boční straně šachovnice tak, aby hráč s bílými figurkami byl na levé straně. Pokud by strany byly prohozeny, šachová notace by měla prohozené černé a bílé figurky. Šachovnice má totiž jen jednu správnou orientaci a to je taková, že čísla jdou po bočních stranách vzestupně



Obrázek 3.4: Nejlepší pozice zařízení při snímání šachovnice - přibližně 12cm od strany šachovnice ve výšce asi 40cm

od bílých figurek k černým. Pozici zařízení lze vidět na obrázku 3.4. Kamera by měla být vzdálena asi 10-15cm od šachovnice a měla by být ve výšce 35-45cm. Tolerance, ve které je rozpoznání šachovnice je relativně velká, ale rozpoznání figurek už tak tolerantní není.

Dále by měla být samozřejmě viditelná celá šachovnice, nezakryta rukou nebo nějakým jiným objektem, osvětlení v místnosti musí být rovnoměrné. Pokud bude přes půlku šachovnice přecházet stín, šance na její rozpoznání se značně snižují. Přesnou intenzitu světla jsem neměřil, kamera zařízení je ale schopna si s průměrným osvětlením poradit. Všechny informace potřebné pro práci s aplikací budou uvedeny v interní nápovědě aplikace.

3.4.2 Algoritmus pro rozpoznávání šachovnice

Zde je uveden algoritmus pro rozpoznávání šachovnice:

Algorithm 1: Rozpoznávání stavu šachovnice

Data: Obraz z kamery
Result: Vnitřní reprezentace šachovnice
předúprava obrazu;
rektifikace obrazu;
detekce hran;
rozpoznání čar v obraze;
filtrace čar;
rozdělení obrazu na políčka;
if *počet políček == 64* **then**
  předání políček pro neuronovou síť;
else
  nahlášení chyby uživateli;
end

Algoritmus funguje tak, že nejprve se vezmou data z kamery. Proběhne odstranění šumu a podobné předúpravy. Dále se obraz rektifikuje, aby šly lépe rozpoznat čáry. Dále

následuje detekce hran a pomocí těchto hran a následného filtrování jsou detekovány čáry šachovnice. Pomocí těchto čar je obraz rozdělen na jednotlivá políčka. Pokud je všechno správně rozpoznáno, mělo by být 64 políček. V takovém případě se posílají na klasifikaci neuronové síti. Poté je vytvořena vnitřní reprezentace šachovnice. V opačném případě se nějakým způsobem uživatel upozorní, že šachovnice není rozpoznána.

3.5 Rozpoznávání figurek

Rozpoznávání jednotlivých figurek je velmi náročná úloha. Existuje pro ni více možných přístupů a řešení. Jedním z nich je například rozpoznávání podle šablon nebo segmentace obrazu pomocí neuronových sítí. V navržené implementaci budu také využívat neuronové sítě, ale úloha pro ně bude klasifikace. Šachových figurek je pouze 12 typů, což je v oblasti neuronových sítí malé číslo. Segmentace obrazu by byla zbytečně složitá a vzhledem k tomu, že aplikace má fungovat na mobilních zařízeních, bude potřeba myslet i na efektivitu řešení. Pro práci s neuronovými sítěmi využiji některý z dostupných frameworků, nejspíše Tensorflow [1]. Pomocí nich si stáhnu některou z předtrénovaných sítí a tu následně dotrénuji na vlastním datasetu, jehož sestavení popíši dále. Framework se také uplatní při samotné aplikaci sítě. Vzhledem k tomu, že neuronové sítě vytvářím v jazyku Python (je to dle mého názoru nejjednodušší) a jazykem všech zařízení Android je Java, bude nutné nějak vyřešit kompatibilitu těchto dvou jazyků.

3.5.1 Dataset

Pro natrénování neuronové sítě je potřeba mít relativně velký dataset. Protože úloha, kterou bude neuronová síť vykonávat, není nijak obecná, tak jsem nikde nenašel požadovaný dataset. Tím pádem jsem si ho musel vytvořit sám. Pro každý typ figurky bude potřeba řádově desítky až stovky fotek, to vše připravené ve formátu pro trénování neuronové sítě. To znamená mít tři složky (train, test a valid) a v každé z nich složku pro každou třídu klasifikovaných fotografií zvlášť. Aby byla úspěšnost rozpoznávání vyšší, je nutné mít fotografie z různých místností, různých podkladů šachovnice a také v různou denní dobu, případně v různých typech osvětlení. Sice se fotografie převádí do stupňů šedi, ale všechny tyto rozdíly jsou ve výsledku znát.

Kapitola 4

Implementace aplikace

V této kapitole se zaměřím na to, jakým způsobem jsem uvedený návrh implementoval a kde jsem případně udělal nějaké změny oproti návrhu. Nejprve popíši implementaci samotné aplikace, její funkčnost a přidám screenshoty z aplikace pro lepší představu. Následuje konkrétní použití jednotlivých funkcí pro zpracování obrazu v procesu rozpoznání šachovnice. Dále zde bude část zaměřená na tvorbu, trénování a využití konvoluční neuronové sítě. Budou popsány jak úvodní, tak finální verze sítě spolu s výhodami a nevýhodami. To vše bude doplněno o obrázky a grafy, které průběh zpracování vhodně dokreslují.

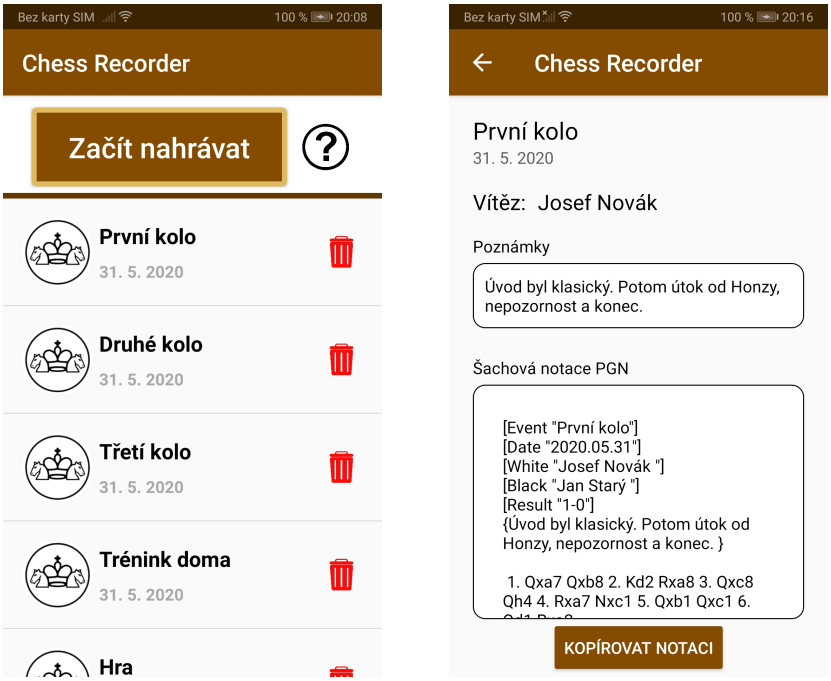
4.1 Mobilní aplikace

Aby mohly být algoritmy prezentovány širší veřejnosti, je potřeba vytvořit mobilní aplikaci, na jejímž pozadí tyto algoritmy poběží. Jako implementační platformu jsem si vybral Android. Důvodem bylo, že je to nejrozšířenější mobilní operační systém na světě. S touto volbou souvisí i implementační jazyk, kterým je Java. Javu jsem si vybral protože už jsem jí na rozdíl od Kotlinu nebo jiných novějších jazyků znal. Aplikace pro Android jsou rozděleny do aktivit. V následujícím textu popíši jednotlivé aktivity mojí aplikace a také například princip ukládání her. Každá aktivita má v principu svoje uživatelské rozhraní (okno) a logiku.

Popis aplikace

Při otevření aplikace se spustí hlavní aktivita. Snímek hlavní obrazovky je na obrázku 4.1 (a). V horní části obrazovky se nachází tlačítko na spuštění nahrávání. Vedle něj je tlačítko s otazníkem pro spuštění aktivity nápovědy. Aplikace funguje ta, že pokud si jí poprvé uživatel spustí, při stisku tlačítka "Začít nahrávat"se stejně nejprve spustí obrazovka s nápovědou. Zbytek obrazovky patří seznamu již nahraných her, protože jde o jediný další důvod, proč by si uživatel aplikaci pouštěl. Tento seznam se načítá z paměti v zařízení. Jsou zobrazeny soubory, které začínají slovem "game", což jsem vybral jako identifikátor. Každá hra má u sebe logo aplikace a kromě základních informací také možnost hru vymazat. Po kliknutí na tlačítko koše se zobrazí modální okno pro potvrzení operace. Oproti návrhu jsem se rozhodl změnit barevné téma aplikace, aby lépe zapadalo do tématu šachu. Dominuje zde hnědá barva se žlutými doplňky.

Další obrazovkou, kterou popíši, je detail nahrané hry. Ten se nachází na obrázku 4.1 (b). Jde o velmi jednoduchou obrazovku, která vypíše všechny informace uložené v příslušném souboru dané hry. Jde o název, datum, jméno vítěze, poznámky a šachovou notaci. Šachová



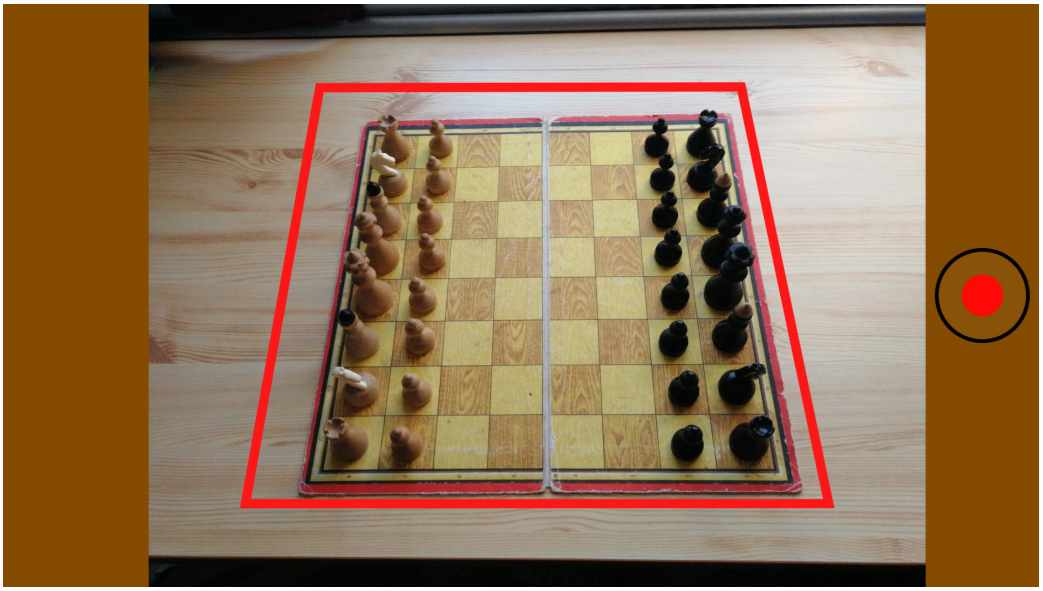
(a) Hlavní obrazovka aplikace (b) Detail nahrané hry

Obrázek 4.1

notace je na obrazovce oddělena ve svém boxu. Ve spodní části obrazovky se nachází tlačítko pro kopírování šachové notace do schránky zařízení. O ukládání her bude sekce dále v textu. O jejich načítání se stará třída `GameReader`. Jedná se pouze o přečtení celého souboru a zprostředkování informací dalším aktivitám, které je mohou potřebovat.

Nejdůležitější aktivitou celé aplikace je nahrávání hry. K tomu slouží aktivita nahrávání. Co se týče uživatelského rozhraní, skládá se ze tří částí. Uprostřed je zobrazován obraz z kamery zařízení v reálném čase. Na této obrazovce je také promítán obrazec, do kterého je nutno mít umístěnou šachovnici při nahrávání. Tento lichoběžník svítí červeně, pokud zařízení nemůže šachovnici detekovat a zeleně, pokud je šachovnice správně rozpoznána. Na pravé straně je panel s tlačítkem pro spuštění a vypnutí nahrávání. Tlačítko funguje naprosto přímočaře jako v jakékoliv jiné aplikaci obsahující vytváření záznamu. Na levé straně je potom panel, do kterého se postupně zapisují tahy provedené hráči. Co se týče pozadí této aktivity, zde se odehrává všechno rozpoznávání a klasifikování. Aktivita je napojena na třídu `RecorderManager`. Tato třída se stará o synchronizaci mezi vytvářením nových stavů a rozpoznáváním tahů. Také iniciuje zápis tahů. Operace s šachovnicí má na starosti balíček tříd `board`. Zde se ve třídě `Chessboard` nachází definice interní reprezentace šachovnice a operací s touto reprezentací. Také je zde třída `MoveRecognizer`, která má metodu pro rozpoznávání tahu. Na vstupu jsou dva stavy šachovnice a tato třída vypočítá, kde přesně byl tah proveden a jaké figurky se na něm podílely. Ověří, zda je tah podle pravidel šachu možný a vrátí text představující reprezentaci tohoto tahu ve standardní algebraické notaci. Aby ale mohla tato třída fungovat, je nutné rozpoznat každý stav šachovnice. K tomu slouží třída `BoardRecognizer`, kterou jsem vyvíjel v prostředí počítačovém. Jedná se o třídu používající různé algoritmy OpenCV v kombinaci s mnou navrženými metodami k tomu, aby rozpoznala šachovnici a lokalizovala jednotlivá políčka. O vytvoření komplet-

ního obrazu šachovnice se stará třída `ChessboardBuilder`. Ta si vezme vyříznuté části šachovnice a prostřednictvím Tensorflow Lite je pošle připravené neuronové síti. Od ní pak zpracuje informaci o typu figurky a vytvoří objekt třídy `Chessboard`.



Obrázek 4.2: Obrazovka nahrávání hry

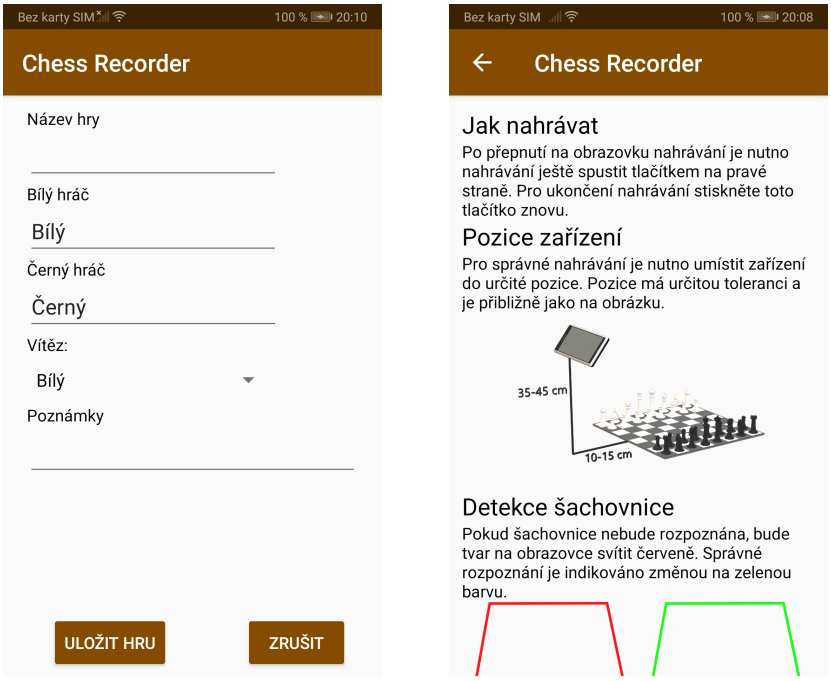
Po ukončení nahrávání je uživateli zobrazena aktivita pro uložení hry (obrázek 4.3 (a)). Zde vyplní všechny informace, které potřebuje ke hře poznamenat a hru uloží. Co se týče uživatelského rozhraní, jedná se o jednoduchý formulář s vstupními políčky pro název hry, jména obou hráčů, možností vybrat vítěze a prostorem pro poznámky. Pod formulářem jsou umístěna tlačítka pro uložení a nebo zrušení a vymazání hry. Způsob, jakým funguje ukládání hry a formát uložených her jsem rozepsal do vlastní sekce 4.1.

Obrazovkou, která je tak trochu mimo běžné používání, je nápověda k nahrávání (obrázek 4.3 (b)). Jedná se o obrazovku, kde je uživateli vysvětleno, jak má umístit zařízení před nahráváním, jak se zachovat při nerozpoznání šachovnice a podobně. Také je zde tlačítko, které může být využito k přímému přechodu na nahrávání.

Ukládání her

Nahrané hry je potřeba nějakým způsobem uložit. V aplikaci je k tomu vytvořena třída `GameWriter`. Pro ukládání her jsem vytvořil vlastní formát obsahující standardní PGN notaci a další data potřebná pro fungování aplikace. Hry jsou ukládány do vnitřní paměti zařízení, protože jejich velikost je velmi malá. Jde jen o textové soubory s délkou přibližně 1000 znaků. Každá hra má v názvu slovo "game", aby šlo tyto soubory odlišit od ostatních, poté následuje univerzální identifikátor, který generuje samo zařízení a nakonec koncovka .chg jako zkratka "chess game". Protože se soubory bude operovat pouze moje aplikace, na pojmenování úplně tak nezáleží.

Formát souborů uložených her je následující. Na začátku souboru jsou informace potřebné pro aplikaci a následuje zápis hry. Každá z těchto informací se zároveň využije i v analyzátorech PGN, protože jejich formát respektuje tagy uvedené v oficiální dokumentaci PGN. První položkou je "Event". Ten jsem si pro své potřeby upravil a je zde název hry. Dále je uvedeno datum, jména obou hráčů, výsledek a poznámky. Pro poznámky má šachová



(a) Ukládání hry (b) Nápověda aplikace

Obrázek 4.3

notace také speciální formát. Ohraničeny jsou složenými závorkami. Po těchto informacích následují jednotlivé tahy hry ve formátu číslo kola, tah bílého hráče a tah černého hráče.

4.2 Rozpoznávání šachovnice

Drtivou většinu rozpoznávání mám implementovanou pomocí knihovny OpenCV. Ta poskytuje všechny potřebné funkce pro získání dat, které budu využívat. Tyto data budu následně analyzovat. Tím mám na mysli například pozici detekovaných čar, které mi funkce z OpenCV vrátí. Pomocí těchto dat budu následně čáry filtrovat a ponechávat si pouze ty relevantní. To samé platí u rozpoznávání čtverců. Funkce OpenCV rozpoznává oblasti. Moje algoritmy poté vyfiltrují pouze ty oblasti, které vyhovují podmínkám, které si stanovím. Více k algoritmům je napsáno v následujících sekcích.

Předzpracování obrazu

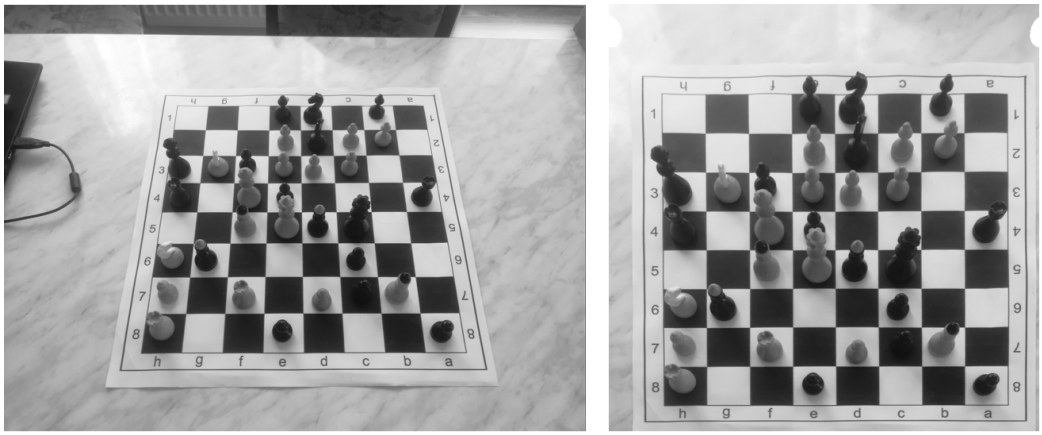
Vzhledem k různým prostředím a podmínkám, ve kterých je obraz pořizován je nutné si obraz nejprve předzpracovat. Tím docílím toho, aby byly všechny obrazy pokud možno co nejvíce podobné před vstupem do dalšího zpracování. Předzpracování obrazu se skládá ze 3 částí.

Nejprve je nutné obraz konvertovat do stupňů šedi. Tím se zajistí, aby měly všechny černé figurky vždy podobnou barvu a bílé figurky také. Pro převod používám adaptivní hranici vypočtenou podle průměrné intenzity pixelů v obraze. Poté následuje sjednocení velikosti obrazu. Pokud by se obraz zpracovával v původní kvalitě a velikosti, zbytečně by se zatěžoval systém mnoha výpočty. Tím, že snížím velikost a kvalitu obrazu, se docílí rych-

lejšího zpracování, což je velmi důležitý parametr aplikace. Posledním krokem je odstranění zbytečného šumu v obraze. Po této úpravě zůstanou v obraze pouze více význačné hrany a detekce hran bude dávat více relevantní výsledky.

Rektifikace obrazu

Po předzpracování přichází rektifikace obrazu. Z obrazu se tak odstraní perspektiva a je snazší poznat rovnoběžné čáry na šachovnici. Tuto část jsem řešil tak, že na obrazovku zařízení při snímaní budu promítat přibližnou polohu, kde se má šachovnice nacházet. To povede k větší úspěšnosti rozpoznávání nejen šachovnice, ale následně i jednotlivých figurek. Úhel snímání je totiž velmi důležitý faktor tohoto rozpoznávání. Podle předem známé přibližné polohy šachovnice určím tedy body, které následně transformuji pomocí funkce *warpPerspective()*. Výsledek tohoto kroku je vidět na obrázku 4.4. Čáry na pravé části obrázku jsou již pěkně vodorovné.



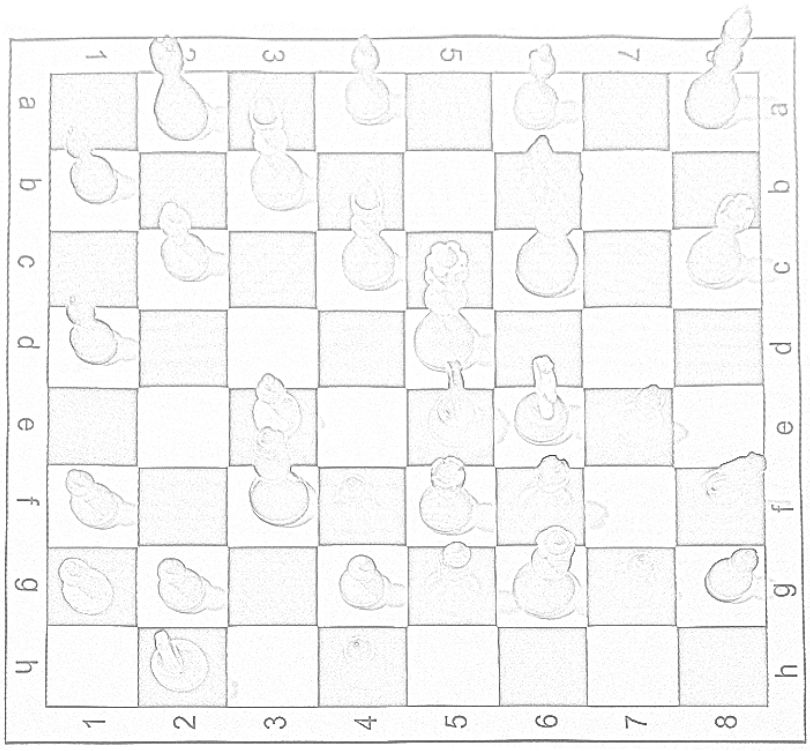
Obrázek 4.4: Nalevo šachovnice před rektifikací, napravo po rektifikaci

Detekce hran

Na takto upraveném vstupním obrazu poté spustím funkci pro detekci hran. Pro tento úkon jsem využil funkci *Laplacian()*. Jedná se o implementaci Laplaceova operátoru v OpenCV. Jeho princip jsem popsal v sekci 2.1.4. Pro správnou detekci je ale potřeba také správně nastavit parametry. Já využívám adaptivní detekci, která přizpůsobí parametry detekce tomu, jak jsou v obrazu zastoupeny jednotlivé odstíny šedé. Díky tomu se nedetekují zbytečně nepodstatné hrany nebo naopak nevynechávají důležité hrany. Výstupem je obraz, na kterém jsou hrany vyznačeny bíle. Protože velké černé plochy nevypadají dobře, pro ukázku jsem barvy invertoval (viz. obrázek 4.5).

Detekce a výběr čar

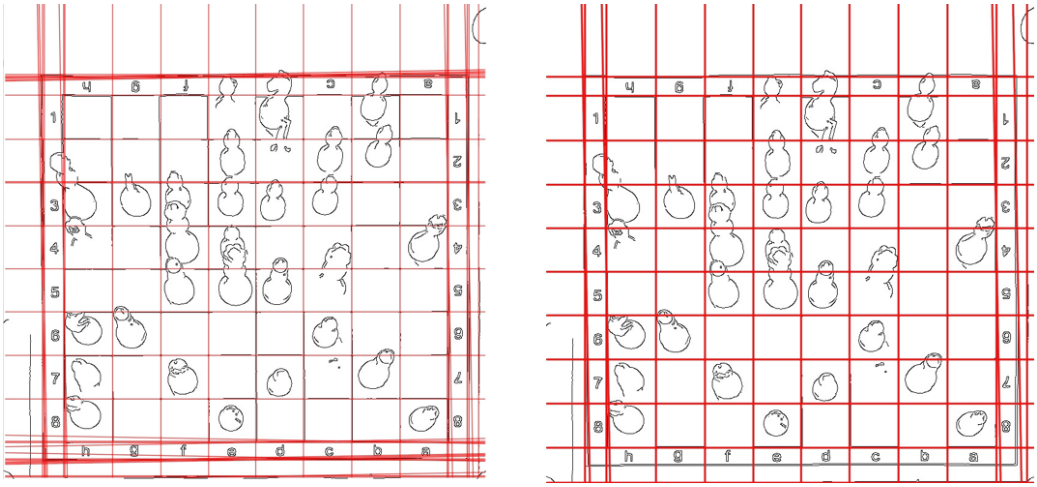
Výstup detekce hran se využije dále, kde se detekují čáry. V OpenCV jsem se rozhodoval mezi dvěma funkcemi pro detekci čar. Jedna z funkcí pracuje vypisuje čáry pomocí polárních souřadnic a druhá pomocí kartézských. Protože v následné filtraci čar a při dalších operacích s nimi se bude lépe pracovat v polární soustavě souřadnic, vybral jsem funkci *HoughLines()*. Tato funkce je implementací Houghovy transformace, jejíž princip je vysvětlen v sekci 2.1.4.



Obrázek 4.5: Obrázek hran rozpoznaných na šachovnici

Poté, co jsou získány všechny čáry v obraze, se provádí jejich filtrace, aby v konečné fázi zbylo pouze 18 čar, a to 9 v jednom směru a dalších 9 ve směru kolmém na první skupinu. Tak je totiž definována šachovnice. V první fázi filtrace se čáry rozdělí na horizontální a vertikální s využitím souřadnic jejich bodů. To, že budou vertikální opravdu vertikální je zaručeno rektifikací obrazu v předchozím zpracování. Současně s rozdělováním probíhá také proces podobný clusteringu. V mojí implementaci je to realizováno jako výběr jedné z čar, které vedou velmi blízko u sebe. Tento krok je vidět na obrázku 4.6. Některé čáry jsou ještě dvojitě, jejich odstranění ale bude uskutečněno dalšími kroky algoritmu. Následně jsou čáry seřazeny zleva doprava, respektive shora dolů.

Tato filtrace ale nezaručuje, že se detekovaly všechny čáry, pouze odstraní čáry přebytečné a nevyhovující předpokladům. Další fází je kontrola, zda byly detekovány všechny čáry. Po rektifikaci je obraz oříznutý přibližně okolo šachovnice. Následující proces probíhá jak pro čáry horizontální, tak pro čáry vertikální. Nejprve se spočítá průměr vzdáleností všech sousedních čar. Tím se zjistí přibližná velikost políčka šachovnice. Poté se určí čára, která je nejblíže ke středu obrazu. S touto znalostí můžeme spočítat, na jakých pozicích by se přibližně měly nacházet detekované čáry. Pokud se ovšem některou z čar nepovedlo detekovat, je nutné ji tam uměle vytvořit. To je se znalostí velikosti políčka velmi snadný úkol. Na konci proběhne kontrola, zda seznamy čar opravdu obsahují 9 a 9 objektů a tím je tato fáze u konce.



Obrázek 4.6: Nalevo čáry před filtrací, napravo sjednocené čáry s několika nedokonalostmi (barvy černá a bílá invertovány)

Detekce a výběr čtverců

V této části se z dvou skupin vzájemně kolmých čar vytvoří čtverce. Systém, který používám, je ten, že se čáry nakreslí do obrazu a následně se použije funkce *findContours()*. Ta detekuje všechny plochy mezi čárami. Bohužel ale detekuje i plochy mimo šachovnici. Ty je nutné v dalším kroku odfiltrovat pryč.

Filtrování čtverců probíhá na podobném principu jako filtrování čar. Nejprve se spočítá průměrná plocha všech oblastí detekovaných v obraze. Protože čtverců správné velikosti je tam hodně, je tímto možné odfiltrovat extrémy. Samozřejmě, že plocha nemůže být jediným parametrem k určení toho, zda je oblast čtverec. Následuje tedy porovnání velikostí hran. Zde si nechávám trochu větší toleranci, protože občas je některá čára detekována trochu jinde, než by měla být. V některých případech ani toto filtrování nestačí, a tak přichází na řadu poslední část. Tou je porovnávání pozic čtverců s pozicemi čar, která ohraničují šachovnici. Tím je zaručeno, že se čtverec nachází uvnitř šachovnice a je tak s největší pravděpodobností i šachovým políčkem.

Zůstane tak 64 čtverců, což odpovídá počtu čtverců na šachovnici. Funkce uloží pozici těchto čtverců a tu poté předá dále. Poslední částí je vyřezání všech pozic na šachovnici a předání těchto výstřižků ke klasifikaci. Protože figurky mají určitou výšku, nestačí vyřezávat pouze čtvereček detekovaný předchozí funkcí. Experimenty jsem zjistil, jaký bude potřeba přesah do okolí čtverce tak, aby byla figurka ve výřezu vždy celá (pokud budeme předpokládat její umístění uvnitř čtverce, a ne na hraně). Nejvyšší figurky, tedy královna a král, přesahují v maximálním případě horní hranu čtverce o jednu a půl výšku čtverce. Do stran není nutný skoro žádný přesah, ten je tam v podstatě pouze pro jistotu a kvůli nepřesnostem v řezání čtverců.

4.3 Klasifikace figurek

4.3.1 Tvorba datasetu

K vytvoření dostatečného datasetu bylo nutné nafotit stovky snímků a ty následně zpracovat a označit. Focení probíhalo na několik etap. Chtěl jsem mít snímky z různých míst s

Tabulka 4.1: Počty figurek v datasetu pro určení typu figurky

Typ figurky	**Počet fotografií**
Bílý pěšec	220
Bílý král	200
Bílá dáma	200
Bílý střelec	200
Bílý jezdec	200
Bílá věž	200
Černý pěšec	220
Černý král	200
Černá dáma	200
Černý střelec	200
Černý jezdec	200
Černá věž	200
Prázdné políčko	250
Celkem	2690

různým osvětlením a v různou denní dobu. Pro zjednodušení a zrychlení procesu jsem vždy rozestavěl šachové figurky do určitého postavení a poté šachovnici vyfotil ze všech čtyř stran, abych pokryl co možná největší počet kombinací vzájemného překrývání a umístění figurek. Po pořízení snímků jsem využil část mojí aplikace, která již v tu dobu uměla rozdělit šachovnici na odpovídající části. Tím jsem tvořil už jednotlivé fotografie pro dataset. Do programu jsem ale přidal část, která mi na základě toho, které tlačítko jsem zmáčkl, správně fotografii zařadila a pojmenovala. I tak je ale tato část práce nesmírně zdlouhavá a monotónní. Takto vytvořený dataset jsem nadále musel upravit tak, aby byly správně rozložené fotografie mezi složky pro trénování, validaci a testování.

Celková velikost datasetu je přibližně 8500 fotografií. Jak se ale v průběhu vývoje ukázalo, ne všechny fotografie byly nakonec použity ve finálním datasetu. Tato velikost je tedy celkový počet fotografií, které jsem během práce využil. Počty všech figurek v datasetu jsou uvedeny v tabulce 4.1. Pro optimální fungování sítě je nutné mít dataset vyrovnaný. Pokud by bylo například mnohem více pěšců, naučila by se síť, že když obrázek klasifikuje jako pěšce, bude mít nejspíše pravdu. Já ale potřebuji, aby byla síť stejně dobře schopná rozpoznat krále i pěšce, je tedy potřeba mít přibližně stejný počet od obou figurek. Protože je ale pěšců ve hře více než ostatních figurek, mám i v datasetu mírně větší zastoupení pěšců. Celkem dataset pro určení typu figurky obsahuje asi 2700 obrázků. Ukázka datasetu je na obrázku 4.7.

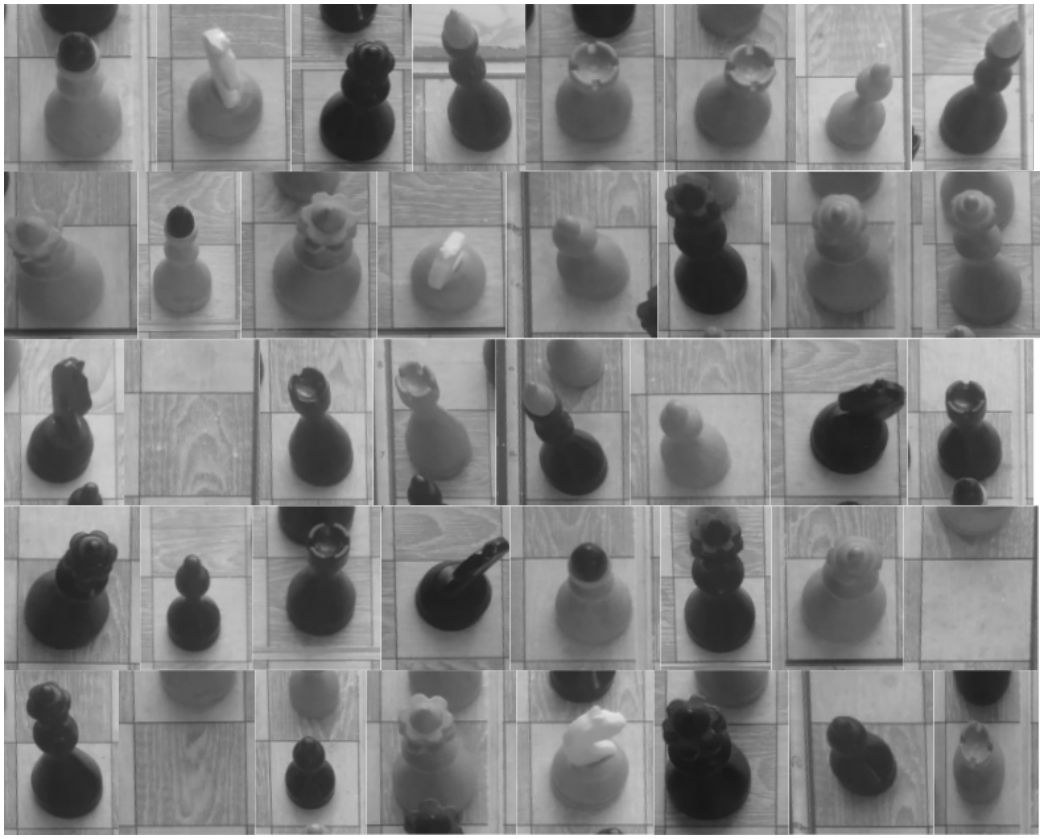
Během vývoje jsem zkusil vytvořit mnoho typů datasetu, rozdělené například pouze na typ figurky místo typu a barvy, nebo obsahující odděleně černá a bílá políčka. Co se ale ukázalo jako užitečné, byl dataset rozdělený pouze na obsazená a prázdná políčka. Ten jsem nakonec využil k natrénování druhé neuronové sítě. Počty obrázků v datasetu jsou v tabulce 4.2.

4.3.2 Tvorba a trénování neuronové sítě

Pro tvorbu a trénování sítě jsem využil Keras, který jsem popsal dříve. Práce se sítěmi v Kerasu je relativně jednoduchou záležitostí. Nejprve je ale nutné mít funkční Tensorflow

Tabulka 4.2: Počty figurek v datasetu pro účely rozpoznání obsazenosti políčka

Typ políčka	**Počet fotografií**
Prázdné políčko	2500
Obsazené políčko	2500
Celkem	5000



Obrázek 4.7: Náhled na dataset - vzorek dat

a další prerekvizity pro práci s Kerasem. Pracoval jsem jak se skripty v Pythonu, tak v interaktivnějším prostředí Jupyter notebooku.

Tvorba, trénování a úpravy datasetu se stále prolínají, ale i přesto se pokusím jednotlivé části od sebe oddělit pro větší přehlednost. Nejprve jsem vytvořil základní dataset a na něm jsem poté zkoušel různé typy sítí. Z tohoto testování jsem si vybral některé architektury sítí a s nimi poté zkoušel experimenty v oblasti stylu trénování. Nakonec jsem ještě upravoval dataset, aby byl větší a vyrovnanější.

Síť MobileNet (verze 1)

Tuto síť jsem si vybral z jednoduchého důvodu - výkon. I když jsou mobilní zařízení čím dál více výkonné, stále není možné na nich používat robustní neuronové sítě, které běží na počítačích. Jak již název napovídá, síť MobileNet je vytvořena tak, aby fungovala i s menšími zdroji. Samozřejmě se za to platí nižší přesností, ale to je pochopitelné. Dobré je, že i když je přesnost menší, poměr velikost/výkon je u těchto sítí velmi dobrý.

Síť, kterou budu používat, byla již trénována na datasetu ImageNet. [5] Protože je ale moje úloha značně odlišná a konkrétní, nebudu využívat tolik z původních hodnot sítě. Pro trénování nejprve odstraním posledních 6 vrstev sítě, což mi přišlo jako nejlepší hodnota. Dále je potřeba vytvořit svoji výstupní vrstvu, která bude určovat jednu ze 13 tříd, které bude síť umět rozpoznávat. Těmito třídami bude 6 figurek jedné barvy, 6 figurek druhé barvy a prázdná pole. Před trénováním je nutné některé vrstvy zmrazit, aby se jejich hodnoty už neměnily a předtrénování bylo tedy k něčemu dobré. Po sérii experimentů jsem zmrazil prvních 30 vrstev z celkových 88. To znamená, že z původního trénování využiji relativně malou část. Je to proto, že všechny výstřižky jsou vcelku podobné a není zde taková diverzita. Trénování těchto menších sítí nezabere až tak dlouho. Tři sta epoch trénování zabralo přibližně hodinu času na mém laptopu.

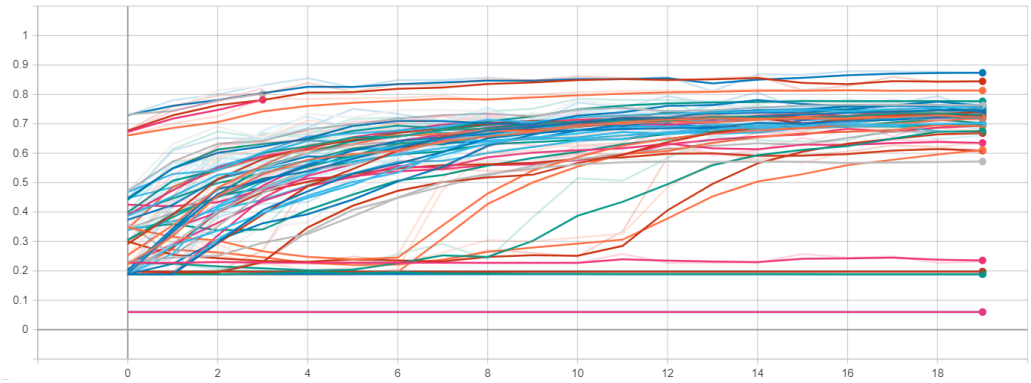
Základní typy sítí (verze 2)

Když síť MobileNet nedokázala dosáhnout rozumné úspěšnosti (data trénování budou analyzována dále), rozhodl jsem se zkusit implementovat vlastní konvoluční neuronovou síť. Začal jsem se základními typy sítí podle různých návodů a doporučení. Tou dobou už jsem měl zprovozněné trénování na grafické kartě, takže trénování jedné sítě byla otázka maximálně několika minut. Důvodem také bylo, že menší sítě nevyžadují tolik epoch na trénování. Také velikost datasetu byla větší a trénování sítě MobileNet se stalo nemožné. Jak se později ukázalo, výsledky těchto sítí nebyly vůbec špatné.

Základní architektura těchto sítí byla následující. Na vstupu je konvoluční vrstva následovaná sdružovací vrstvou. Poté jsem testoval různé počty opakování této kombinace za sebou. Následuje zplošťovací vrstva a za ní také variabilní počet plně propojených vrstev. Výstupní vrstvou byla také plně propojená vrstva. Chtěl jsem vyzkoušet, jak různé počty různých typů vrstev ovlivní úspěšnost rozpoznávání při stejném datasetu. Analýza výsledků byla velmi složitá, ale nakonec jsem dospěl k tomu, že nejlepší je mít 3 konvoluční vrstvy, poté zplošťovací vrstvu následovanou 2 plně propojenými vrstvami. Do těchto počtů nepřičítám vstupní a výstupní vrstvu. Pokud neměla síť žádnou plně propojenou vrstvu kromě výstupní, dopadla zpravidla velmi špatně. Samozřejmě ve všech těchto počtech jsou určité výjimky a hlavně mnoho dalších proměnných, které úspěšnost ovlivňují, ale takový byl hlavní trend během trénování. Pro ilustraci výsledků trénování přikládám graf na obrázku 4.8.

Složitější sítě (verze 3)

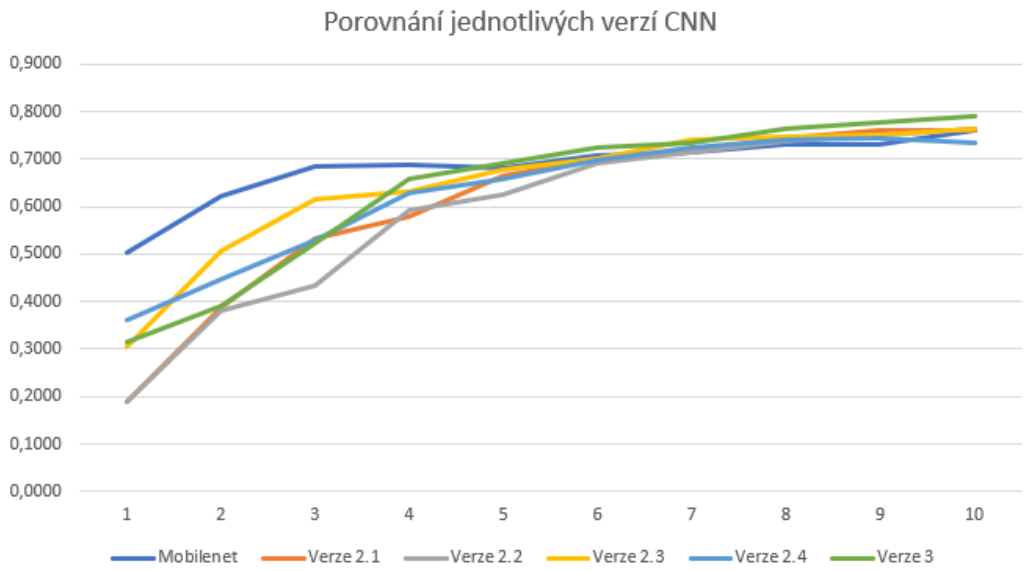
Druhá verze přinesla sice nějaké úspěchy, ale bylo nutno dbát také na celkovou velikost natrénované sítě. Do mobilní aplikace není vhodné dávat sítě, které jsou velké třeba 200 MB. Proto jsem opustil systém zkoušení všech variant a vydal se cestou většího průzkumu a studia a sofistikovanější architektury. Při hledání informací jsem narazil na nějaké tipy, jak takové sítě dělat, a tak jsem vytvořil nakonec tři sítě a zkusil, jakých výsledků dosáhnu. Jedna z těchto sítí byla mnohem lepší než ostatní a tak už od této doby budu zmiňovat pouze tuto. Skládá se z dvou konvolučních vrstev za sebou, poté následuje sdružovací vrstva, poté znovu dvě konvoluční vrstvy a sdružovací vrstva, dále zplošťovací a dvě plně propojené vrstvy. Protože jde o síť, kterou používám, přikládám i vizualizaci (obrázek 4.10).



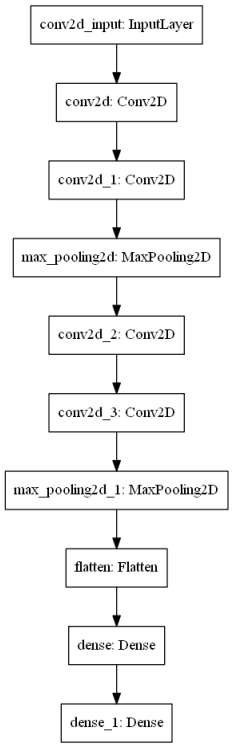
Obrázek 4.8: Graf trénování sítí druhé verze. Pouze pro ilustraci, avšak jde vidět, že mnoho sítí mělo velmi podobné výsledky. Poté můj výběr závisel na dalších ukazatelích, jako například výsledná velikost sítě. To, že některé sítě mají vysokou úspěšnost, je způsobeno přetrénováním. Pro konečné využití jsem trénování zastavil včas. Vygenerováno toolkitem Tensorboard [25].

4.3.3 Výsledná síť

Nejlepším modelem, který jsem byl schopen vytvořit je model verze 3, který jsem popsal v minulé sekci. V této sekci se zaměřím na princip, jak je tento model používán v aplikaci a také ukážu porovnání jednotlivých verzí. Vytvořil jsem graf na obrázku 4.9, který vystihuje rozdíly mezi jednotlivými verzemi sítě. Rozdíly nejsou až tak veliké, ale nejlépe si vedla třetí verze. Je vidět, že každá ze sítí má jiný způsob a rychlost trénování. Síť MobileNet je oproti mým sítím lepší hned od začátku, ale nakonec skončí s nejhorším výsledkem. Je to určeno nejspíš tím, že je optimalizovaná pro obecnější případy použity. Nejvyšší úspěšnost měla poslední vytvořená síť verze 3. Tuto síť jsem využil pro oba zmíněné datasety a vytvořil dva různé modely, které v aplikaci používám. Pro jejich využití je nutné modely z formátu `.h5`, který je typický pro Keras, převést do formátu `.tflite`. Díky tomu je poté možné používat knihovnu Tensorflow Lite pro Android a modely nějakým způsobem použít. V aplikaci je třída, která tyto modely načte a poté už je jen nutno vždy načíst obrázek, upravit jeho velikost a hodnoty pixelů do odpovídajících rozpětí, dotázat se sítě na pole pravděpodobností, následně se vybere třída s nejvyšší pravděpodobností a výsledek se dále zpracuje.



Obrázek 4.9: Porovnání výsledků jednotlivých verzí sítě. Sítě verze 2.1, 2.2, 2.3 a 2.4 jsou velmi podobné a liší se v počtu různých vrstev jak jsem již popsal.



Obrázek 4.10: Vizualizace neuronové sítě verze 3. Vygenerováno pomocí frameworku Keras [13].

Kapitola 5

Testování aplikace

Obsahem této kapitoly bude popis testování aplikace a interpretace výsledků tohoto testování. Proces testování aplikace jsem rozdělil do tří hlavních částí. První z nich je uživatelské testování samotné aplikace bez funkce nahrávání. Druhá a třetí část už se týkají záznamu šachové partie. Jednou z nich je rozpoznávání pouze přítomnosti šachovnice. Další částí je poté rozpoznávání figurek na šachovnici. Měření a experimenty jsem rozdělil z toho důvodu, že tyto dvě části jsou na sobě nezávislé a je možné je samostatně testovat i upravovat a dále vylepšovat.

5.1 Uživatelské testování aplikace

Uživatelské testování je velmi užitečným nástrojem při vývoji uživatelských rozhraní aplikací. Každý člověk s mobilním telefonem zachází trochu odlišně a každý člověk je jinak zručný v jejich ovládání. Proto se u takového testování dokáží odhalit nedostatky a možná vylepšení, které by mě osobně například nenapadly. Nejprve bych rozebral jak probíhalo testování uživatelského rozhraní aplikace. Pro začátek jsem vytvořil funkční předběžný vzhled. Ten jsem nadále upravoval pouze podle svého uvážení. Když byla aplikace upravena k mojí spokojenosti, začal jsem aplikaci ukazovat známým a rodině. Testovací množina nebyla až tak veliká z důvodů stavu v době testování a izolace obyvatelstva. Celkem čítala 17 osob. Skládala se z příbuzných a blízkých přátel. Je proto nutno brát výsledky s rezervou, i když jsem zdůrazňoval aby byli co nejvíce objektivní. Testování probíhalo na mobilním telefonu Honor 9 s verzí Androidu 9.0 Pie.

Úkoly pro uživatelské testování

Testování probíhalo následovně. Uživatel dostal do ruky mobilní telefon se spuštěnou aplikací. Prvním bodem testování bylo zobrazit si nápovědu pro aplikaci. Dále testované subjekty dostaly úkol nahrát krátký úryvek šachové partie. Z důvodů chybějící šachovnice byl tento krok občas nahrazen simulovaným nahráním. Nedílnou součástí tohoto úkolu bylo také vyplnit formulář pro ukládání hry. Poté dostali za úkol svoji hru najít v seznamu ostatních her a zobrazit si její informace detailně. Poté, co se vrátili na hlavní obrazovku byl poslední úkol odstranit záznam, který v minulých krocích vytvořili. Po ukončení těchto úkolů dostali testovaní ještě možnost projít si například znovu nějaké obrazovky a poté byla jejich interakce s aplikací ukončena. Následně dostali několik otázek týkajících se jejich zkušeností a připomínek k aplikaci.

Otázky byly následující:

- Jaké máte zkušenosti s užíváním smartphonů?
- Jak byste celkově ohodnotil/a aplikaci? (stupnice 1-5, 1 je nejlepší)
- Co se vám na aplikaci líbilo nejméně?
- Co se vám na aplikaci líbilo nejvíce?
- Máte nějaké další připomínky nebo nápady k vylepšení aplikace?

Výsledky uživatelského testování

V této sekci rozebírám výsledky uživatelského testování. Nejprve vyhodnotím každou z otázek a poté ještě moje osobní poznámky ohledně věcí, kterých jsem si při testování všiml. Co se první otázky týče, je z přiloženého grafu vidět, že jsem dosáhl rovnoměrného rozložení testovací množiny vzhledem ke zkušenostem s používáním chytrých mobilních telefonů. Mezi uživateli jsou jak úplní začátečníci, tak i lidé, kteří tyto telefony používají již několik let. Další otázka byla zaměřena na celkový pocit z aplikace a její výsledky jsou na obrázku 5.1 stejně jako výsledky první otázky. Výsledky byly relativně příznivé, i když musím brát v potaz to, že příbuzní nebudou hodnotit tak přísně jako cizí lidé. Pomyslně jsem si tyto hodnoty o něco snížil. Dva lidé dali této aplikaci známku 1 a žádnou 5.

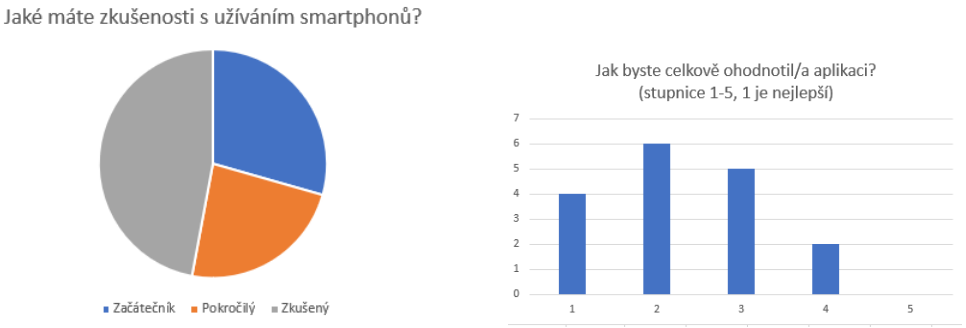
Další dvě otázky měly prozradit nejsilnější a nejslabší stránky mojí aplikace. Pochvalu dostala aplikace za celkovou jednoduchost, barevné provedení a například "propracovaný systém vybírání vítěze hry při ukládání". Byly to spíše takové drobnosti, cokoliv bylo dobré. Co se týče chyb, bylo nutno přepracovat stránku detailu hry, protože její vzhled nebyl moc přívětivý. Jinak negativních věcí moc nebylo zmíněno, proto jsem se musel spolehnout na vlastní pozorování chování uživatelů při testech. Zde jsem zpozoroval několik věcí, které se mi nelíbily. Při prvním spuštění aplikace je zobrazena nápověda, ale nebyla možnost z ní přejít dále na nahrávání. Uživatel se tedy musel vracet zpět na hlavní stránku. Tato skutečnost mi při vývoji unikla, protože jsem měl aplikaci již nainstalovanou a puštěnou několikrát předem. Přidal jsem tedy tlačítko i do nápovědy. Dále jsem musel přidat modální okno při mazání hry a při zrušení ukládání, protože se občas stalo, že uživatel omylem vymazal hru, kterou nechtěl. Dalšími úpravami byly různé grafické detaily a změny funkčnosti některých tlačítek. Ve výsledku jsem s testy byl spokojen, protože jsem byl schopen vyvinout aplikaci, která zvládla veškeré testování bez větších problémů.

5.2 Testování rozpoznávání šachovnice

Proto, aby se daly rozpoznávat jednotlivé figurky, musí být nejdřív identifikována šachovnice. Pokud by byla úspěšnost rozpoznávání příliš nízká, nedocházelo by většinu času k druhé části rozpoznávání. Je tedy nutné mít otestováno, že je šachovnice rozpoznána v drtivé většině případů a právě proto jsem se rozhodl tuto část testovat samostatně.

Průběh testování

Testování probíhalo v iteracích průběžně s vývojem algoritmu. V této části rozeberu dvě hlavní iterace, u kterých jsem měřil úspěšnost na větším vzorku obrázků. Pro účely tohoto testování jsem nafotil přibližně 500 obrázků, které jsem využil v obou zmíněných hlavních



(a) Otázka č. 1

(b) Otázka č. 2

Obrázek 5.1: Uživatelské testování

Tabulka 5.1: Zápis z testování rozpoznávání šachovnice

Iterace	Počet vzorků	Správně	Špatně	Úspěšnost
1	100	38	62	38 %
2	300	147	153	49 %
3	300	180	120	60 %
4	500	331	169	66 %
5	500	450	50	90 %
6	500	463	37	93 %

iteracích. Při první iteraci byla úspěšnost rozpoznávání 65 %. To je samozřejmě velmi málo, a tak následovala bližší inspekce rozpoznávání. Zjistil jsem, že u asi třetiny chybně rozpoznaných obrázků je úplně stejná chyba. Měl jsem totiž implementovaný algoritmus, který doplní případnou chybějící čáru na okraji šachovnice. Ale pokud nebyla detekována jediná z čar uprostřed, nastala zmiňovaná chyba a šachovnice nebyla rozpoznána. Proběhlo tedy vylepšení algoritmu a následné opakování testů. Druhé kolo testů proběhlo již mnohem lépe zejména díky odhalené nedokonalosti první implementace. Úspěšnost rozpoznávání v druhém kole byla 93 %.

Finální výsledky testování

V tabulce 5.1 lze vidět všechny iterace, které v rámci tohoto testování proběhly. Pro zjednodušení jsem vybral dvě hlavní, a to iteraci číslo 4 a iteraci číslo 6. Důvodem je, že první z nich byla těsně před rapidním zlepšením výsledků a druhá je finální verze algoritmu. První tři iterace probíhaly na menším vzorku obrázků z důvodu postupného přidávání fotografií. Celé toto testování bych zhodnotil tak, že konečná verze algoritmu je schopna velmi spolehlivě rozpoznávat pozici šachovnice, pokud není ničím zakryta. Zkoušel jsem dva různé typy šachovnic a oba měly úspěšnost víceméně stejnou. Číslo 93 % úspěšnosti by se ale mělo brát s rezervou, protože v momentě, kdy se zařízení umístí do správné pozice, je úspěšnost rozpoznání dalších snímků velmi blízko 100 %.

5.3 Testování rozpoznání stavu šachovnice

Poslední část testování. Nechal jsem ji nakonec z několika důvodů. Prvním je, že v kontextu celé aplikace je toto poslední, co je prováděno. Druhým důvodem je, že je to nejdůležitější část celé aplikace. Posledním důvodem je to, že toto testování nedopadlo vůbec dobře. Respektive moje snaha vylepšit výsledky v této činnosti nedopadla dobře. Testování dopadlo velmi neuspokojivě a má zásadní vliv na současné fungování aplikace.

Průběh testování

Protože úspěšnost rozpoznání stavu šachovnice je zejména práce neuronové sítě, budu používat výsledky, které jsem získal při trénování jednotlivých sítí. Tyto data jsem získával přímo v průběhu trénování pomocí toolkitu Tensorboard [25]. I když jsem během testování vytvořil přibližně 100 různých typů sítí a každou trénoval třemi různými způsoby, není možné všechna tato data uvádět v jednom grafu. Po ukončení všech testů jsem tedy vyfiltroval ta data, která korespondují s jednotlivými verzemi neuronových sítí popisovaných v části 4.3.2.

Výsledky testování

Interpretace výsledků se dá rozdělit na dvě části. Úspěšnost neuronové sítě při rozpoznání jednoho políčka a úspěšnost aplikace v korektním rozpoznání celé šachovnice. Úspěšnost jednoho rozpoznání mé nejlepší sítě je přibližně 79 %, jak je možno vidět na grafu 4.9. Tento graf blíže rozebírám v sekci 4.3.3. Bohužel schopnost rozpoznávat se stejnou pravděpodobností všechny figurky už nefunguje. Neuronová síť se naučila předpovídat některé z nich tak, aby požadované úspěšnosti dosáhla a jiné z nich nedokáže rozpoznat ani z velmi jasných fotografií. To už vytváří první problém. Druhý problém je ten, že pro získání úspěšnosti rozpoznání úplně všech políček na šachovnici 0.79^64 vyjde v kalkulačce číslo tak malé, že nemohu doufat ve správné rozpoznání všech políček. Konkrétně jde o číslo 0.00000028062984, což je přibližně 0,00002 %. V případě, že bych chtěl dosáhnout celkové úspěšnosti 90 %, musela by být úspěšnost sítě 99,8 %. Bohužel takových výsledků jsem nebyl schopen dosáhnout. Na toto navazuje skutečnost, že rozpoznání opravdu provedených tahů je nemožné. Tahy sice rozpoznávány jsou, ale v jednu chvíli jich je podle neuronové sítě provedeno hned několik, i když se se žádnou figurkou nepohne a pohne se pouze nepatrně s kamerou.

Kapitola 6

Možnosti dalšího vývoje

Ačkoliv jsem se snažil, aby aplikace byla co nejdokonalejší a nejpřesnější, narážel jsem (hlavně z časových důvodů) na problémy, které jsem musel řešit kompromisy. Aplikace se totiž skládá z několika částí, ke kterým je potřeba přistupovat stejně zodpovědně a důkladně. Naskýtají se tedy možnosti, jak aplikaci vylepšit po všech stránkách, ať už jde o úpravu stávajícího kódu nebo i přidání nových funkcí či možností. Rád bych zde moje návrhy na vylepšení uvedl, pokud by někdo chtěl aplikaci dále vyvíjet a vylepšovat.

Prvním kompromisem, který jsem udělal, je nutnost mít mobil v přibližně stejné pozici po celou dobu hry. Ačkoliv je to implementačně velmi náročné, dalo by se upravit aplikaci tak, aby bylo jedno z kterého úhlu se šachovnice zabírá. Nejprve by bylo nutné upravit algoritmus pro rozpoznávání samotné šachovnice, který už s přibližně pevnou pozicí počítá. Dále také vyřezávání jednotlivých figurek a také jejich následné rozpoznávání. To už by vyžadovalo mnohem větší dataset. I tento přístup ale narazí na určité limity (například moc nízká pozice zařízení při snímání), při kterých už jsou figurky zakryty samy sebou na tolik, že nepůjdou separovat a rozpoznat.

Na tento zmíněný limit navazuje další možnost vylepšení. Ten je ještě více implementačně náročný, ale podle mého názoru by o hodně zvýšil přesnost a flexibilitu aplikace. Na rozpoznávání by mohly spolupracovat dvě mobilní zařízení, která by spolu komunikovaly například pomocí bluetooth nebo WiFi. Šachovnice by tedy byla snímána z různých úhlů, což by eliminovalo většinu špatně rozeznaných figurek z důvodu zakrytí jinou figurkou. Takové vylepšení ale zase vyžaduje implementaci správné komunikace mezi zařízeními. Také zde je problém se správností zápisu v šachové notaci. Šachovnice má při hraní pouze jednu správnou orientaci. Tu by bylo nutné nějakým způsobem zadat.

Posledním vylepšením aplikace by mohl být vestavěný přehrávač her. Místo kopírování šachové notace a její vkládání do některého z dostupných analyzátorů by se mohl uživatel jedním tlačítkem dostat na obrazovku, kde by šla hra přehrát.

Kapitola 7

Závěr

Cílem této práce bylo navrhnout a implementovat mobilní aplikaci, která umožní široké veřejnosti nahrávat svoje šachové partie a jejich zápis ukládat ve standardní šachové notaci. V původním plánu bylo vyvinout aplikaci co nejvíce flexibilní, aby mohla fungovat opravdu v jakékoliv pozici a situaci. Ukázalo se ale, že toto není úplně možné z hlediska náročnosti řešení. Pro standardní notaci je nutno, aby byla známá orientace šachovnice a tím pádem původní postavení figurek. Dále také není možné snímat šachovnici z velmi ostrých úhlů, protože vzájemné překrytí figurek je příliš velké. Vznikl tedy návrh aplikace, který umožňuje snímat šachovnici za předem definovaných podmínek, jako například pozice mobilního zařízení.

V rámci práce byl také vytvořen dataset pro trénování neuronových sítí, který by mohl být využit k dalšímu rozvoji aplikace nebo k implementaci podobných aplikací. Jedná se o dataset fotografií políček obsahující jeden z dvanácti typů šachových figurek (rozdílné barvy jsou počítány jako rozdílné typy) a prázdných políček také obou barev. Celková velikost datasetu je přibližně 8500 fotografií.

Co se týče vyhodnocení úspěšnosti aplikace, tak zde je potřeba rozdělit úspěšnost na dvě programově nezávislé části. První z nich je rozpoznávání samotné šachovnice. Zde se úspěšnost pohybuje kolem 93 % s tím, že pokud uživatel nalezne jednou pozici, ve které je šachovnice rozpoznána, šance úspěšného rozpoznání se ještě zvyšuje. Výjimka nastává v případě, kdy se rapidně změní světelné podmínky v místnosti nebo například jeden z hráčů bude zrovna na tahu a zakryje tak část šachovnice svojí rukou. Druhou částí je rozpoznávání figurek. Zde již moje řešení není zdaleka tak uspokojivé. Ani po měsících tréninku a experimentování jsem nedokázal dosáhnout takové úspěšnosti rozpoznávání jednoho políčka, aby rozpoznávání tahů bylo funkční. Maximální hodnoty, ke kterým jsem se dostal byly kolem 80 %. Jenže to je pouze šance na správné rozpoznání jednoho políčka. Pokud si spočítáme šanci na správné rozpoznání celé šachovnice tak vyjde, že je to víceméně zázrak, když se povede vše rozpoznat. Snažil jsem se ještě algoritmicky odstraňovat některé z chyb rozpoznávání, ale 80 % je velmi málo na to, aby se s tím dalo něco provést.

Výsledkem práce je tedy kromě zmíněného datasetu také aplikace, která poskytuje všechny potřebné funkce. Je zde funkční mechanismus nahrávání her, jejich ukládání, prohlížení zápisů a podobně. Při nahrávání aplikace upozorní uživatele, pokud šachovnice nebyla rozpoznána, aby si mohl upravit pozici mobilního zařízení. Vytvořena byla také neuronová síť, která má za úkol rozpoznávat jednotlivé figurky. Bohužel u ní se mi nepodařilo dosáhnout na uspokojivou úspěšnost a tím pádem i celé vytváření zápisu je značně nepřesné. Pokud by se ale do aplikace dodala lepší neuronová síť, která by byla schopna rozpoznat úspěšně 99,9 % figurek, aplikace by byla plně využitelná.

Literatura

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z. et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org. Dostupné z: http://tensorflow.org/.

[2] Bonner, A. *The Complete Beginner's Guide to Deep Learning: Convolutional Neural Networks*. Towards Data Science, Jun 2019. Dostupné z: https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb.

[3] *Wikipedia - Chess*. [cit. 2020-05-28]. Dostupné z: https://en.wikipedia.org/wiki/Chess.

[4] *The Most Exciting Uses of Image Recognition That are Already Changing Our Lives*. 2018. Dostupné z: https://indatalabs.com/blog/uses-image-recognition?cli_action=1590481015.01.

[5] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. et al. ImageNet: A Large-Scale Hierarchical Image Database. In: *CVPR09*. 2009.

[6] Donald, S. *Stonewall - Chess Computer Vision*. 2017. Dostupné z: https://www.youtube.com/watch?v=uDw_vCNbx-I.

[7] Duda, R. O. a Hart, P. E. *Use of the Hough transformation to detect lines and curves in pictures*. Artificial Intelligence Center, 1971.

[8] Dar, P. *25 Open Datasets for Deep Learning Every Data Scientist Must Work With*. [cit. 2020-05-20]. Dostupné z: https://www.analyticsvidhya.com/blog/2018/03/comprehensive-collection-deep-learning-datasets/.

[9] G., A., Menglong, H., Chen, Z. B., Kalenichenko, D., Wang, W. et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. Dostupné z: https://arxiv.org/pdf/1704.04861.pdf.

[10] Gupta, M. [cit. 2020-03-24]. Dostupné z: https://medium.com/data-science-in-your-pocket/vehicle-number-plate-detection-and-ocr-tcs-humain-2019-a253019e52a1.

[11] Jain, R. C., Kasturi, R. a Schunck, B. G. Chapter 4 : Image Filtering. In: *Machine vision*. McGraw-Hill, 1995, s. 112–136.

[12] Jirbandey, A. *A brief history of Computer Vision and AI Image Recognition*. 2018. Dostupné z: https://www.pulsarplatform.com/blog/2018/brief-history-computer-vision-vertical-ai-image-recognition/.

[13] *Keras: the Python deep learning AI*. [cit. 2020-04-08]. Dostupné z: https://keras.io/.

[14] *Laplacian operator*. Dostupné z: https://www.tutorialspoint.com/dip/laplacian_operator.htm.

[15] Mark, K. *It's visualization of data processed by Autopilot's on-board computer*. [cit. 2020-05-28]. Dostupné z: https://insideevs.com/news/346873/video-tesla-autopilot-sees-fire-truck/.

[16] *Online Mockup, Wireframe and UI Prototyping Tool · Moqups*. 2020. Dostupné z: https://moqups.com/.

[17] Orémuš, Z. *Rozpoznání šachové pozice z fotografie [online]*. 2018. Diplomová práce. Masarykova univerzita, Fakulta informatiky, Brno. Dostupné z: DostupnézWWW<https://is.muni.cz/th/meean/>.

[18] O'Shea, K. a Nash, R. An Introduction to Convolutional Neural Networks. *ArXiv e-prints*. Listopad 2015.

[19] *Standard: Portable Game Notation Specification and Implementation Guide*. [cit. 2020-05-28]. Dostupné z: http://www.saremba.de/chessgml/standards/pgn/pgn-complete.htm.

[20] Rodriguez, J. *What's New in Deep Learning Research: Mobile Deep Learning with Google MnasNet*. Towards Data Science, Aug 2018 [cit. 2019-12-05]. Dostupné z: https://towardsdatascience.com/whats-new-in-deep-learning-research-mobile-deep-learning-with-google-mnasnet-cf9844d30ae8.

[21] *Computer Vision and Smart Mirrors*. [cit. 2020-05-28]. Dostupné z: https://media.springernature.com/original/springer-static/image/chp%3A10.1007%2F978-1-4842-3931-5_3/MediaObjects/462363_1_En_3_Fig1_HTML.jpg.

[22] *STONEWALL - CHESS VISION*. [cit. 2020-05-28]. Dostupné z: http://notsamdonald.com/stonewall.

[23] Szeliski, R. *Computer Vision: Algorithms and Applications*. Springer, 2010.

[24] Tang, Y. X. andGongguo a Hoff, W. *Chess Piece Recognition Using Oriented Chamfer Matching with a Comparison to CNN*. Dostupné z: https://inside.mines.edu/~youyexie/paper/WACV2018.pdf.

[25] *Tensorboard: TensorFlow's visualization toolkit*. [cit. 2020-04-08]. Dostupné z: https://www.tensorflow.org/tensorboard.

[26] *OpenCV - Overview*. [cit. 2019-12-9]. Dostupné z: https://www.tutorialspoint.com/opencv/opencv_overview.htm.

[27] vaibhav06891. *Computer Vision Talks: Hough Transforms*. [cit. 2020-05-28]. Dostupné z: http://computervisionwithvaibhav.blogspot.com/2015/09/hough-transforms.html.

[28] Zhang, X., Zhou, X., Lin, M. a Sun, J. *ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices*. Dostupné z: http://openaccess.thecvf.com/content_cvpr_2018/CameraReady/0642.pdf.