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 S PRVKY POČÍTAČOVÉHO VIDĚNÍ

MOBILE APPLICATION WITH COMPUTER VISION ELEMENTS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **TOMÁŠ DYK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JAKUB ŠPAŇHEL**
SUPERVISOR

BRNO 2018

Zadání bakalářské práce/21201/2017/xdykto00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Dyk Tomáš**
Obor: Informační technologie
Téma: **Mobilní aplikace s prvky počítačového vidění**
Mobile Application with Computer Vision Elements
Kategorie: Zpracování obrazu

Pokyny:

1. Prostudujte základy zpracování obrazu. Zaměřte se zejména na problematiku detekce a rozpoznání tištěných i ručně psaných číslic.
2. Prostudujte a analyzujte problematiku detekce a rozpoznání tištěných a ručně psaných číslic a dále prostudujte problematiku návrhu a tvorby aplikací pro Android.
3. Navrhněte mobilní aplikace pro snímání vytištěných křížovek sudoku s možností dohrání na mobilním zařízení.
4. Implementujte minimalistickou použitelnou verzi řešené aplikace.
5. Otestujte aplikaci na testovacím vzorku uživatelů.
6. Analyzujte zpětnou vazbu uživatelů a navrhněte+realizujte iterativní úpravy řešené aplikace.
7. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Dle pokynů vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3 zadání,
- rozpracovaný bod 4.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Špaňhel Jakub, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2
L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Bakalářská práce popisuje vývoj mobilní aplikace pro načítání a vyřešení sudoku. Mobilní aplikace je určena pro operační systém android. Uživatel načte sudoku do aplikace tím, že sudoku vyfotí v novinách nebo časopise. Aplikace následně detekuje pozici sudoku na fotografii, rozpozná čísla v jednotlivých polích sudoku a umožní uživateli toto sudoku v mobilním telefonu vyřešit. Všechny nevyřešené sudoku, včetně nově nahraných, se v aplikaci ukládají, aby je uživatel mohl kdykoli dořešit.

Abstract

Bachelor thesis describes development of mobile applications for loading and solving sudoku. Mobile application is developed for Android. The user loads a sudoku into the application by taking a picture of sudoku from a newspaper or a magazine. The application then detects the sudoku position in the photo, recognizes the numbers in each sudoku field and allows the user to solve the sudoku in the mobile phone. All unresolved sudoku, including newly uploaded, is stored in the application so that the user can solve it at any time.

Klíčová slova

počítačové vidění, android, mobilní aplikace, sudoku, neuronové sítě

Keywords

computer vision, android, mobile application, sudoku, neural network

Citace

DYK, Tomáš. *Mobilní aplikace s prvky počítačového vidění*. Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jakub Špaňhel

Mobilní aplikace s prvky počítačového vidění

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jakuba Špaňhela. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Dyk
15. května 2018

Poděkování

Rád bych poděkoval vedoucímu bakalářské práce Ing. Jakubovi Špaňhelovi za jeho rady, podnětné připomínky a čas, který mi věnoval na konzultacích. Dále bych chtěl poděkovat mé rodině za podporu při studiu a při psaní bakalářské práce. V neposlední řadě bych chtěl poděkovat všem respondentům, kteří mi poskytli zpětnou vazbu při testování mobilní aplikace.

Obsah

Úvod

Počítačové vidění je poměrně nový počítačový obor. V poslední době se však velmi rozšířil. Počítačové vidění se již dnes využívá v mnoha oborech a jeho význam stále roste. Policejní složky například využívají počítačové vidění při hledání osob nebo identifikace vozidel pomocí bezpečnostních kamer, v lékařství se využívá při vytváření 3D modelu orgánů. Velký význam má i při vývoji plně autonomního auta. Prvky počítačového vidění se dokonce začali objevovat i u aplikací pro mobilní telefony.

Myslím si, že obor počítačového vidění má velkou budoucnost, a proto jsem si chtěl vyzkoušet naprogramovat aplikaci s prvky počítačového vidění. Již několik let se zabývám řešením sudoku, proto jsem chtěl vytvořit bakalářskou práci s tímto tématem. Napadlo mě zkombinovat tyto dvě věci a vytvořit aplikaci, která by uživateli umožnila načíst sudoku z novin pomocí vestavěného fotoaparátu v mobilním telefonu a umožnit uživateli v něm sudoku vyřešit. Nápad na tuto aplikaci jsem dostal při cestování v městské hromadné dopravě, když jsem si četl noviny a uviděl v nich zadání sudoku. Chtěl jsem jej vyřešit, ale bohužel jsem s sebou neměl žádné psací potřeby. Uvědomil jsem si, že podobný problém by mohlo mít víc lidí a možnost vyplnění sudoku v mobilním telefonu by tento problém vyřešila.

Práce je rozdělena do sedmi kapitol. První čtyři kapitoly poskytují základní teoretické informace o metodách využitých při vývoji této aplikace. Zbylé kapitoly popisují vlastní návrh řešení a jeho implementaci. První kapitola se zabývá principy zpracování digitálního obrazu. Popisuje základní algoritmy využívané při zpracování obrazu. Druhá kapitola popisuje základní principy fungování neuronových a konvolučních sítí, dostupné knihovny pro jejich implementaci a popis architektury pro rozpoznávání ručně psaných čísel. Kapitola číslo tři popisuje operační systém Android, jeho architekturu a informace potřebné pro vývoj aplikace na této platformě. Čtvrtá kapitola vysvětluje principy metod používaných při řešení sudoku. Pátá a šestá kapitola popisuje návrh a samotnou implementaci aplikace, použité knihovny a metody. V kapitole o implementaci aplikace je i část, ve které je popsáno testování mobilní aplikace a vyhodnocení uživatelského dotazníku.

Kapitola 1

Zpracování digitálního obrazu

Zpracování obrazu je forma zpracování signálu. Je těžké přesně definovat pojem zpracování obrazu, jelikož nelze přesně určit hranici, kde začíná a končí zpracování, analýza a porozumění obrazu. Jedna z možností je definovat zpracování obrazu jako proces, na jehož vstupu je obraz a výstupem je obraz vhodně upravený pro další zpracování [24]. Pro zpracování digitálního obrazu se využívá počítačových algoritmů jako například transformace obrazu, segmentace obrazu, detekce hran nebo změny barev v obraze. Některé z nich jsem popsal podrobněji v následujících sekcích.

Počítačové vidění se zabývá pochopením obrazových dat počítačem a k tomu využívá výše uvedené fáze – zpracování, analýza a porozumění obrazu. Obrazová data mohou být například fotky, videa, záznamy z několika kamer nebo data ze skenerů. Počítač se snaží popsat obrázek a najít v něm objekty nebo se snaží z dostupných obrazových dat zrekonstruovat model. U nalezených objektů se snaží zjistit a popsat jejich vlastnosti, které se pak využívají při dalším zpracování [21]. K hledání a popisování objektů se velmi často používá umělá inteligence, například konvoluční neuronové sítě, které budou popsány v kapitole 2.

1.1 Adaptivní prahování

Prahování patří mezi nejjednodušší metody, které se využívají k předzpracování obrazu pro složitější operace. Prahování určuje pro každý pixel jeho výslednou barvu (bílou nebo černou) na základě jasu daného pixelu. Princip prahování lze zapsat funkcí:

$$f(c) = \begin{cases} A & \text{pro } c < \textit{hodnota prahu} \\ B & \text{pro } c \geq \textit{hodnota prahu} \end{cases}$$

kde c je intenzita barvy daného pixelu, $f(c)$ je jeho výsledná hodnota, A(bílá) a B(černá) jsou nové hodnoty pixelu.

U klasického prahování je nastavena globální hodnota prahu, která platí pro všechny pixely v obraze. U adaptivní prahování je obraz rozdělen na menší čtvercové oblasti. Pro každou oblast je pak vypočítána hodnota prahu, která platí pro všechny pixely v dané oblasti. Pro výpočet hodnoty prahu existuje několik funkcí. Nejjednodušší je vypočítání prahu na základě střední hodnoty dané oblasti. Další možností je každému pixelu v dané oblasti přiřadit určitou váhu. Váha se určuje na základě gaussova okna. Prahová hodnota se pak určí na základě váženého součtu těchto pixelů [20].

1.2 Gaussovo rozostření

Gaussovo rozostření se také velmi často využívá při předzpracování obrazu, jelikož pomáhá zbavit obraz šumu. Zbavení obrazu šumu je důležité pro přesnější detekci hran. U detekce hran hraje úroveň šumu zásadní roli, jelikož může razantně snížit efektivitu hranového detektoru. Gaussovo rozostření je velmi efektivní k odstranění gaussova šumu. Pro gaussovo rozostření se využívá Gaussova funkce, které má pro dvě dimenze tvar

$$G(x, y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

kde $G(x, y)$ je výsledná hodnota pole konvolučního jádra, x je vzdálenost od počítaného pixelu na x-ové ose, y je vzdálenost na y-ové ose a σ je směrodatná odchylka Gaussova (normálního) rozložení. Pro dvě dimenze připomínají výsledné hodnoty v matici obrysy kruhů, které směřují od středu matice. Výsledná hodnota pixelu se vypočítá jako vážený průměr hodnot pixelů v okolí. Výpočet se provádí pro každou barevnou složku obrazu. Váhy pixelů se získají z konvolučního jádra. Velikost konvolučního jádra také určuje velikost okolí pixelu.

U okrajů obrázku nastává problém, jelikož konvoluční jádro zasahuje mimo obrázek. Tento problém se řeší například rozšířením obrazu. To znamená, že bod mimo obraz bude mít stejnou hodnotu jako jemu nejbližší bod v obraze. Další metodou je zrcadlení. Vnější bod získá hodnotu bodu, který je mu nejblíže a má stejnou vzdálenost od okraje obrázku. Další možností je nebrat v potaz body mimo obraz a pro krajní body vypočítat výslednou hodnotu jako vážený průměr jen z dostupných pixelů. Mezi často požívané metody patří i stočení obrazu. Chybějící pixely za hranou obrazu přebírají hodnoty pixelů z opačného konce obrazu [2].

1.3 Cannyho hranový detektor

Algoritmy pro detekci hran se snaží nalézt takové body, u kterých se ostře mění jas. Tyto body pak shlukovat do přímek nebo křivek, které se nazývají hrany. Algoritmy se využívají k nalezení struktury objektů v obraze. Výsledkem pak bývá obvykle obraz, který má černé pozadí a do něj jsou zaznačeny pouze nalezené hrany z původního obrazu. Hrany bývají zaznačeny bílou barvou. Obraz lze pak uložit jako binární, čímž se zmenší jeho velikost. V případě, že je nutné uchovat i směr hrany, jsou hrany označeny barevně.

Princip tohoto detektoru vymyslel John Canny a v roce 1986 ho popsal ve své publikaci *A Computational Approach to Edge Detection* [12]. Snažil se vyvinout detektor hran, který by dosahoval vynikajících výsledku v definovaných kritériích. První kritérium je malá chybovost při hledání hran, tedy nalezení pouze takových hran které opravdu existují. Druhé kritérium je správné určení pozici hrany, to znamená, aby vzdálenost mezi nalezenou a skutečnou hranou byla co nejmenší. Posledním kritériem je jednoznačnost detekce hran. Algoritmus tedy bude detekovat určitou hranu pouze jednou.

Algoritmus lze rozdělit do čtyř hlavních fází – eliminace šumu, nalezení velikosti a směru gradientu, zpřesnění detekovaných hran a finální korekce hran prahováním pomocí hystereze.

1.3.1 Eliminace šumu

Jak již bylo zmíněno, pro detekci hran je potřeba zredukovat šum v obraze. Pro eliminaci šumu se využívá gaussovo rozostření, které bylo podrobněji popsáno v podkapitole 1.2.

1.3.2 Výpočet velikosti a směru gradientu

Pro určení velikosti a směru přechodu se může využít například robertsův, prewittův nebo sobelův filtr. V knihovně OpenCV se pro cannyho detektor hran využívá sobelův filtr. Jelikož v tomto projektu využívám právě tuto knihovnu, bude zde popsán podrobněji pouze sobelův filtr.

Sobelův operátor (sobelův filtr) vypočítá pro každý pixel vektor přechodu intenzity barev v obraze. Výsledkem tohoto filtru je obraz s barevně označenými hranami, kde jednotlivé barvy určují úhel hrany. Sobelův filtr využívá dvě konvoluční jádra pro výpočet velikosti gradientu přechodu ve vertikálním a horizontálním směru. Konvoluční jádra mají rozměry 3x3. Díky malým rozměrům není výpočetně náročný. Jádra obsahují tyto hodnoty:

$$G_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, G_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

Celková velikost intenzity přechodu se pak vypočítá pomocí vztahu

$$G = \sqrt{G_x^2 + G_y^2}$$

Směr (úhel) přechodu se vypočítá pomocí vztahu

$$\theta = \arctan\left(\frac{G_y}{G_x}\right)$$

Úhel je zaokrouhlen na hodnotu jednoho ze čtyř hodnot úhlu $(0°, 45°, 90°, 135°)$. Podle velikosti úhlu přechodu se určí, jakou barvu bude mít daný pixel.

1.3.3 Zpřesnění nalezených hran

Pro splnění druhého kritéria optimální detekce hran, musí dojít ke zpřesnění nalezených hran. V této fázi algoritmu dojde k odebrání bodů, které nedosahují lokálního maxima. Tímto krokem dojde ke ztenčení nalezených hran a tedy i k jejich zpřesnění.

U každého pixelu se zkontroluje zda velikost jeho intenzity přechodu ($G_{x,y}$) je lokálním maximem. Lokální maximum se určuje porovnáním dvou sousedních pixelů. Tyto pixely se vyberou na základě směru přechodu (úhlu $\theta_{x,y}$). Jestliže má úhel $\theta_{x,y}$ hodnotu 0° porovnávají se hodnoty intenzity přechodu z polí $G_{x-1,y}$ a $G_{x+1,y}$, pří hodnotě úhlu 45° hodnoty $G_{x-1,y+1}$ a $G_{x+1,y-1}$, při 90° se využijí hodnoty $G_{x,y-1}$ a $G_{x,y+1}$ pro porovnání lokálního maxima a při 135° pak hodnoty $G_{x-1,y+1}$ a $(G_{x+1,y-1})$.

1.3.4 Prahování pomocí hystereze

V této části algoritmu dochází k rozhodnutí, které body jsou skutečnými hranami a které by se měly odstranit. K identifikaci skutečných hran se využívá prahování hysterezí. Zadají se dvě prahové úrovně – minimální a maximální práh. Pixel, který má hodnotu intenzity přechodu pod úrovní minimálního prahu, se vyhodnotí jako málo viditelná hrana a nebude se tedy považovat za hranu. Pixel s hodnotou intenzity přechodu nad úrovní maximálního prahu se vyhodnotí jako skutečná hrana. U pixelů, které mají hodnotu mezi těmito dvěma úrovněmi, se musí navíc určit, zda jsou tyto pixely pokračováním skutečné hrany. Body vyhodnocené jako pokračování skutečné hrany se vyhodnotí jako skutečná hrana, ostatní body se vyloučí ze zpracování. Výsledkem je binární obraz s vyznačenými skutečnými hranami.

1.4 Dostupné knihovny

Pro zpracování obrazu existuje několik knihoven, které implementují algoritmy popsané v předchozích sekcích. Mezi nejznámější a nejpoužívanější patří OpenCV a FastCV.

1.4.1 OpenCV

Open source Compuer Vision library, neboli zkráceně OpenCV, je knihovna zaměřená na zpracování obrazu v reálném čase. Knihovna je napsána v jazyce C++ a je vydávána pod licencí BSD. Díky dostupným rozhraním ji lze využít v jazycích C++, Python, Java a MATLAB na podporovaných operačních systémech Windows, Linux, Mac OS, iOS a Android. Díky přenositelnosti a jednoduchému použití patří OpenCV mezi nejrozšířenější knihovny pro zpracování obrazu.

Knihovna je rozdělena do několika modulů. Hlavní modul (`core`), definuje základní datové struktury a operace s nimi. Tento modul je potřebný pro funkci ostatních částí knihovny. Modul `imgproc` slouží ke zpracování digitálního obrazu, který zahrnuje například funkce pro transformaci obrazu, změny nebo konverzi barev. Dále knihovna obsahuje modul `objdetect` pro detekci objektů v obraze na základě klasifikátorů. Později byl přidán i modul pro klasifikaci a shlukování dat, který využíval strojového učení. Modul `video` slouží pro analýzu videa, jako například sledování objektů v záznamu, odfiltrování pozadí. OpenCV od verze 2.3.0 obsahuje i `GPU` modul. Tento modul nabízí využití GPU, tedy využití výpočetního výkonu grafické karty, která je specializována pro výpočty související s grafikou. Výpočty implementovaných algoritmů z tohoto modulu jsou několikanásobně rychlejší díky tomu, že jsou prováděny na optimalizované jednotce. Využitím těchto algoritmů dosahuje knihovna až 30-ti násobnému zrychlení. Modul využívá platformu CUDA od firmy nVIDIA, která umožňuje paralelní výpočty [11].

1.4.2 FastCV

FastCV je knihovna vyvíjena speciálně pro mobilní zařízení. Knihovna obsahuje nejčastěji využívané funkce pro zpracování obrazu. Tyto funkce jsou optimalizovány pro mobilní zařízení obsahující ARM nebo Snapdragon procesor. Díky optimalizacím dosahuje na mobilních zařízeních několika násobně rychlejších operací s obrazem oproti knihovně OpenCV [1].

Kapitola 2

Vícevrstvé neuronové sítě

Neuronové sítě jsou inspirovány nervovou soustavou živých organismů. Nervová soustava se skládá ze vzájemně propojených nervových buněk – neuronů. V případě neuronových sítích lze princip těchto buněk napodobit modelem vzájemně propojených umělých neuronů. Model neuronu je možné zapsat jednoduchou funkcí, jako vážený součet všech vstupů neuronu. Výsledná hodnota neuronu je ještě upravena aktivační funkcí. Aktivační funkce se liší u jednotlivých modelů umělého neuronu. Nejčastěji se používá sigmoidální, skoková, hyperbolická nebo radiální aktivační funkce. Model obecného neuronu lze tedy zapsat funkcí ve tvaru

$$y = S\left(\sum_{i=1}^{N}(w_i x_i) + b\right) \tag{2.1}$$

kde x_i je potenciál neuronu z předchozí vrstvy, w_i je jeho váha, $S()$ je aktivační funkce a b je bias neuronu.

Architekturu neuronové sítě lze rozdělit na vstupní, vnitřní a výstupní vrstvy. Vnitřní část může obsahovat i několik vrstev. Pro tuto část se používá označení „black box“, jelikož informace v těchto vrstvách člověku nedávají smysl. Vrstvy neuronů jsou mezi sebou propojeny tak, že každý neuron z jedné vrstvy je propojen se všemi neurony z vrstvy následující. Jednotlivá spojení mezi neurony mají svoji vlastní váhu (významnost). Váhy neuronů lze chápat jako „znalosti“, které již neuronová síť získala. Neuronová síť funguje na principu dopředného šíření informací. Síti jsou předány parametry, které se předají do vstupní vrstvy. Odtud postupují do následující vrstvy, ve které se vypočítá potenciál jednotlivých neuronů podle rovnice 2.1. Potenciály neuronů jsou pak využity v následující vrstvě. Tento výpočet probíhá postupně až do výstupní vrstvy.

Učení sítě (tedy nastavování váhy jednotlivých spojení) probíhá tak, že se síti předá testovací množina dat a porovnávají se výstupy sítě s očekávaným výsledkem. Na základě tohoto výsledku jsou upravovány vazby mezi neurony tak, aby výstup neuronové sítě odpovídal očekávanému výsledku. Tohoto chování je docíleno pomocí principu backpropagation [22].

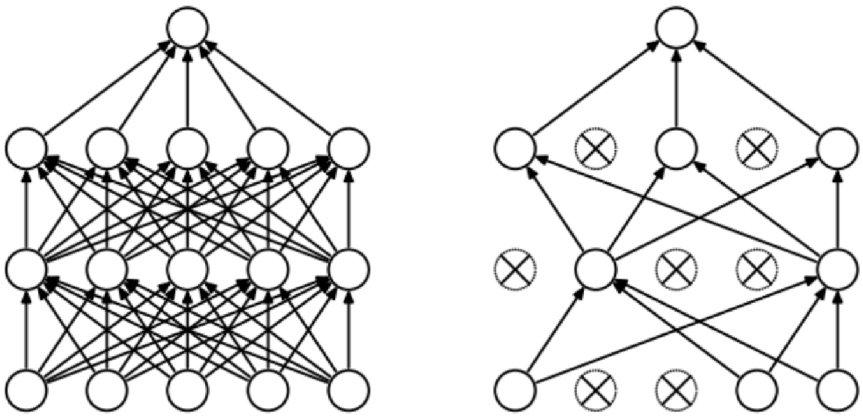
2.1 Backpropagation

Backpropagation lze přeložit jako zpětné šíření. Tato technika se využívá při trénování neuronových sítí k upravení vah mezi neurony tak, aby výstup neuronové sítě odpovídal požadovanému výsledku. Základním principem této techniky je porovnat výstup sítě pro

určitý podnět s jeho očekávaným výsledkem. Rozdíl výstupů se pak v určitém měřítku využije k opravení váhy spojení mezi neurony. Opravování vah probíhá od výstupní vrstvy po vstupní vrstvu sítě. Backpropagation se snaží nalézt takové váhy, aby pro daný vstup byla chybovost co nejmenší – tedy aby se rozdíl výstupu sítě a očekávaného výstupu blížil nule.

2.2 Přeučení sítě

Při učení může nastat problém s přeučením sítě. V takovém případě se síť naučí správně reagovat jen na podměty, které byly v trénovací množině dat. Výsledky z trénování proto bývají velice slibné, ale při testování není síť schopna správně reagovat na reálná data na vstupu. Tomuto lze předcházet vytvořením různorodějšího trénovacího datasetu. Další možností zamezení přeučení sítě je nastavit neuronové síti při učení dropout úroveň. Správná úroveň dropoutu zajistí, aby na sebe neurony nebyly závislé, což napomáhá lepšímu zobecnění neuronové sítě. Dropout funguje tak, že se při učení náhodně vybere určitý počet neuronů, které pro aktuální vstup nebudou aktivní – nebudou vůbec provázány s ostatními neurony. Počet neuronů je určen na základě úrovně dropoutu. Princip dropoutu lze vidět na obrázku 2.1.



(a) Standard Neural Net (b) After applying dropout.

Obrázek 2.1: Princip dropoutu při učení neuronové sítě. Na levém obrázku je vidět klasické zapojení neuronů, na pravém obrázku lze vidět deaktivace náhodně vybraných neuronů při aplikaci metody dropout[1].

2.3 Konvoluční neuronová síť

Konvoluční neuronová síť se principiálně podobá neuronovým sítím. Rozdíl je v tom, že konvoluční neuronová síť obsahuje alespoň jednu konvoluční vrstvu. Konvoluční neuronová síť se využívá především pro zpracování obrazu, protože konvoluční vrstvy umožňují zpracovávat společně body, které spolu souvisejí. Takto se jednodušeji detekují důležité vlastnosti v obraze (například hrany) [18].

[1] Obrázek byl převzat z webu https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5.

2.4 Dostupné knihovny

Pro jednodušší práci s vytvářením neuronových sítí bylo vytvořeno několik knihoven. Většina je naprogramována v jazyce Python nebo C++. Mezi nejpoužívanější knihovny patří Tensorflow a Caffe. Obě zmíněné knihovny nabízí možnost výběru trénování sítě na GPU nebo CPU.

Tensorflow

Tensorflow je knihovna zaměřené na vývoj aplikací, které využívají strojové učení. Knihovna byla vyvinuta výzkumným týmem Google Brain. Knihovna je vydávána pod licencí Apache 2.0. První verze knihovny byla vydána v listopadu v roce 2015.

Společně s TensorFlow je dodávána i webová aplikace TensorBoard. TensorBoard pomáhá při vývoji neuronových sítí. Využívá se k vizualizaci architektury neuronové sítě a kontrolování průběhu učení sítě. Aplikace umožňuje snadnější hledání chyb v neuronové síti. Tensorflow si při učení sítě průběžně ukládá data, které se pak použijí v TensorBoard. Díky tomu pak lze zjistit, jak se postupem času měnily váhy neuronů v jednotlivých vrstvách, přesnost sítě a mnoho dalších parametrů [10].

Caffe

Caffe je knihovna, která se primárně využívá k vytváření konvolučních neuronových sítí. Knihovna je napsána v jazyce C++ s vytvořeným rozhraním pro využití v jazyce Python. Caffe bylo vyvinuto výzkumnou skupinou *Berkeley Artificial Intelligence Research*. Knihovna je vydána pod licencí BSD.

Architektura neuronové sítě se ukládá do speciálního souboru typu `prototxt` za využití knihovny protobuf vyvinuté firmou Google. Takto uložená data jsou pro člověka jednoduše čitelná a přehlednou formou zobrazují strukturu sítě. Programátor může upravovat parametry jednotlivých vrstev sítě přímo v tomto souboru za použití textového editoru [16].

2.5 Architektura LeNet-5

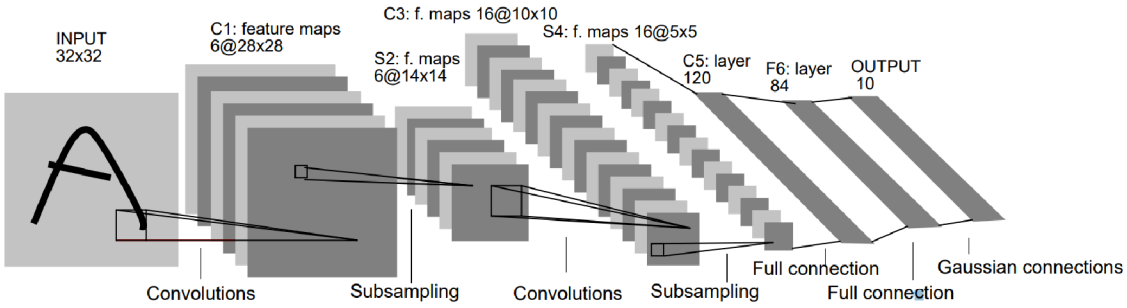
Yann LeCun popsal poprvé v publikaci *Gradient-Based Learning Applied to Document Recognition* [17] v roce 1998 architekturu konvoluční neuronové sítě LeNet-5. Tato síť byla speciálně navržena pro rozpoznávání ručně psaných čísel.

Vstupem sítě je obrázek o rozměrech 32x32 pixelů. Konvoluční síť je složena ze sedmi vrstev. Přesnou podobu sítě lze vidět na obrázku 2.2.

2.6 Dataset

Dataset pro trénování neuronové sítě lze chápat jako soubor dat, na kterém se neuronová síť „učí“ určovat správnou hodnotu výstupu pro podobný vstup.

[2] Obrázek architektury sítě LeNet-5 byl převzat z publikace *Gradient-Based Learning Applied to Document Recognition* [17].



Obrázek 2.2: Architektura konvoluční neuronové sítě LeNet-5 určené pro rozpoznávání ručně psaných čísel[2].

2.6.1 MNIST

Dataset MNIST je databáze ručně psaných čísel. Dataset obsahuje přibližně 60000 vzorků určených pro trénování a dalších 10000 vzorků určených k testování natrénované sítě. Všechny vzorky jsou normalizovány a uloženy ve stupních šedi o fixní velikosti 28x28 pixelů. Číslice je umístěna do středu vzorku na základě vypočítání středu podle váhy jednotlivých pixelů. Obrázky napsaných číslic zaznamenalo 250 dobrovolníků.

Dataset je rozdělen do čtyř souborů. První dvojice souborů obsahuje data pro natrénování sítě a druhá dvojice pro její otestování. Dvojice obsahuje jeden soubor s uloženými vzorky číslic a druhý nese informace, jaká číslice se na daném vzorku vyskytuje. Data jsou v souborech uložena ve speciálním formátu. Soubor se vzorky obrázků na začátku obsahuje čtyři řídící parametry uložené po 32 bitech v následujícím pořadí: konstanta určující typ souboru (hodnota 2051), informace o počtu obrázků, výška obrázku (28) a šířka obrázku (28). Dále soubor obsahuje hodnoty jednotlivých pixelů (po 8 bitech) uložených po řádcích. Soubor se štítky (hodnoty číslic v odpovídajících obrázcích) má první dva řídící parametry shodné jako soubor s obrázky, ale za nimi hned následují štítky pro jednotlivé obrázky uložené po 8 bitech (obsahující čísla 0-9) [3].

2.6.2 EMNIST

V roce 2017 byla vydána databáze EMNIST, která je rozšířením databáze MNIST. Rozšíření EMNIST ponechalo stejný formát jako původní dataset MNIST. Vzorky jsou také vycentrované a mají stejnou velikost 28x28 pixelů. Databázi je možné stáhnout ve dvou formátech. První je ve stejném formátu jako databáze MNIST a druhá verze, která byla vytvořena pro snazší načítání, je ve formátu pro Matlab nebo Python. Pro načtení tohoto formátu stačí pouze zavolat systémovou funkci [13].

Kapitola 3

Android

Android je operační systém založený na linuxovém jádře. Vyvíjí jej společnost *Google* a *Open handset alliance*. Záměrem Googlu bylo vytvoření komplexní platformy, která by byla spustitelná na většině mobilních zařízeních. Za tímto účelem vznikla organizace *Open handset alliance*. *Open handset alliance* je skupina 84 firem, mezi které patří největší softwarové firmy (Google), mobilní operátoři (T-Mobile, Vodafon), výrobci polovodičů (například Intel, nVIDIA, NXP) a výrobci mobilních telefonů (například Samsung, Asus). Firmy mohou dávat podměty pro vylepšení operačního systému, což umožňuje rychlejší vývoj platformy.

Tento operační systém byl primárně určen pro chytré mobilní telefony, nyní se však rozšířil a využívá se i v tabletech, chytrých televizích a chytrých hodinkách. Nyní je nejrozšířenějším operačním systémem u chytrých mobilních telefonů. V roce 2017 činil podíl mobilních telefonů s operačním systémem Android $85,9\%$[1].

Android studio je oficiálním vývojovým prostředím pro vývoj aplikací pro Android. Na konci roku 2014 byla vydána první oficiální verze tohoto vývojového prostředí. Vývojové prostředí mimo jiné obsahuje i emulátor mobilního telefonu pro jednodušší testování aplikace. Pro sestavování projektu se využívá automatický systém Gradle [14].

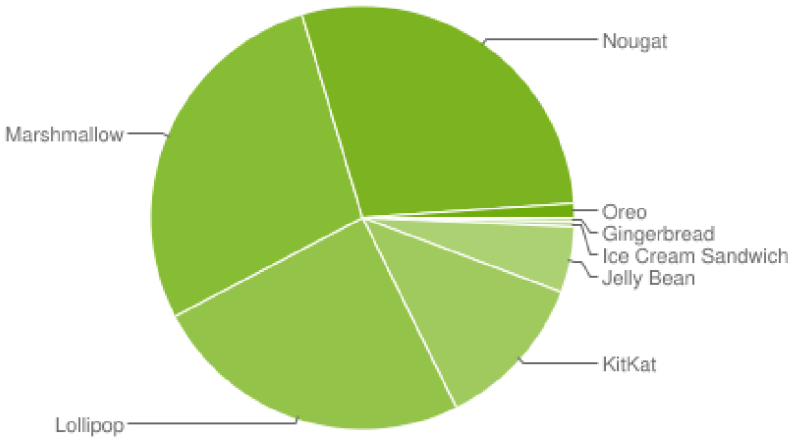
3.1 Architektura operačního systému

Architektura operačního systému Android je rozdělena do pěti vrstev – linuxové jádro, nativní knihovny, android runtime, aplikační framework a samotné aplikace. Jak již bylo zmíněno, operační systém Android využívá pro fungování linuxové jádro. Tato část slouží jako rozhraní mezi hardwarovou a softwarovou částí zařízení. Linuxové jádro se stará o správu paměti, spravování procesů a ovladačů (displej, fotoaparát, WiFi a další). Další vrstvou jsou nativní knihovny. Knihovny jsou napsané v programovacím jazyku C nebo C++. Do této vrstvy spadá mimo jiné i knihovna OpenCV. Vrstva Android runtime obsahuje hlavní knihovny napsané v jazyce Java a Android runtime (ART). Jednotlivé aplikace se spouštějí jako instance virtuálního stroje ART. Vrstva Android runtime je na stejné úrovni jako vrstva nativních knihoven. Aplikační framework poskytuje vývojářům důležité stavební bloky pro správné fungování aplikace. Do této vrstvy se řadí správce aktivit, notifikací, zdrojů, systém zobrazení a mnoho dalších. Poslední vrstvou jsou již samotné aplikace [8].

[1] https://www.statista.com/statistics/263453/global-market-share-held-by-smartphone-operating-systems/

3.2 Verze Androidu

Jednotlivé verze operačního systému android jsou vydávány i s krycími jmény. Krycí jména jsou tématická a jejich název souvisí se sladkostmi. Nejnovější verze Androidu je verze 8.1.0, která nese krycí jméno Oreo. Procentuální využití jednotlivých verzí je vidět na obrázku 3.1. Nejvyužívanější verzí je Android 6.0, který aktuálně využívá 28.1% zařízení s operačním systémem Android. Další používané verze jsou Android 7.0 s 22.3% a Android 5.1 s 19.2% [6].



Obrázek 3.1: Graf reprezentující podíl využití jednotlivých verzích androidu[2].

3.3 Životní cyklus aktivity

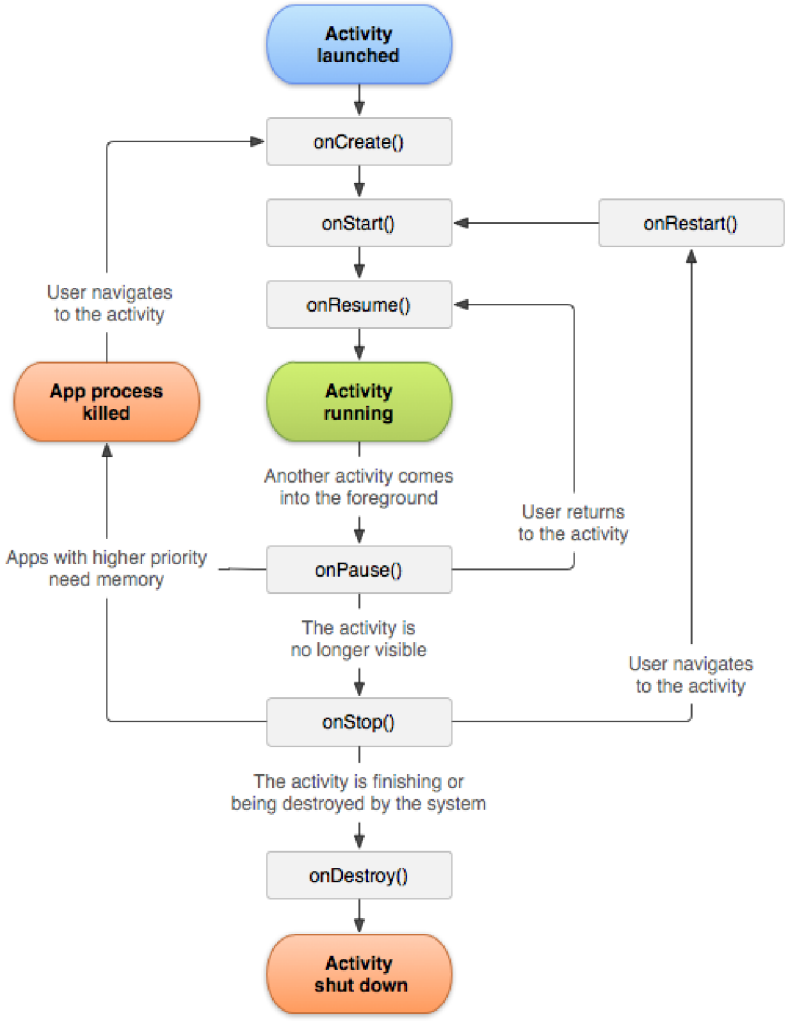
Android aplikace se skládají z aktivit. Aktivita je jejich základní komponenta. Lze si ji představit jako organizátora, který řídí co se stane na dané „obrazovce“. Aktivita poskytuje okno, na které se vykreslí grafické rozhraní „obrazovky“, a definuje jaké akce se provedou při uživatelově interakci s aplikací. Aktivity mohou spouštět další aktivity (i aktivity z jiných aplikací). Spuštění aktivity se rozumí vytvoření její instance.

S přechody mezi aktivitami je spojen životní cyklus aktivity. Jelikož jsou všechny aktivity potomky třídy `Activity`, mají předdefinované callback metody, které určují do jaké fáze životního cyklu aktivita pokročila. Třída `Activity` poskytuje šest základních callback funkcí – `onCreate`, `onStart`, `onResume`, `onPause`, `onStop` a `onDestroy`. Přepsáním implementace jednotlivých callback metod může programátor definovat, co se má v dané fázi aktivity provést.

Programátor musí implementovat callback funkci `onCreate`, která se spouští při startu aktivity. V této funkci by měly být příkazy, které se mají provést pouze jednou a to na začátku životního cyklu aktivity, například namapování a uložení dat do proměnných aktivity. Callback metoda `onStart` se zavolá hned po vytvoření aktivity. V této fázi se uživateli zobrazí samotná aktivita. Probíhá zde například vykreslení grafického rozhraní aktivity. Pak aktivita vstupuje do fáze klasického běhu. V případě, že dojde k přerušení běhu aktivity jinou aktivitou, dojde k vyvolání callback události `onPause` nebo případně `onStop`. Jakmile se aktivita dostane opět do popředí, vrátí se aktivita do fáze `onResume` nebo případně až do

[2]Graf byl převzat z webu https://developer.android.com/about/dashboards/index.html.

fáze `onStart`. Při ukončení aktivity (například systém potřebuje uvolnit více místa paměti nebo se aktivita sama programově ukončila) se vyvolá callback událost `onDestroy` a dojde k uvolnění instance aktivity. Přesný průběh volání callback funkcí zle vidět na obrázku 3.2.



Obrázek 3.2: Obrázek zobrazuje životní cyklus aktivity na operačním systému Android s odpovídajícími callback funkcemi[3].

3.4 Google play

Google play je oficiální distribuční služba pro zařízení s operačním systémem Android. Google play nabízí ke stažení hry a aplikace, dále také poskytuje online distribuci elektronických knih, hudby a filmů.

Pro možnost vydávat programy nebo hry ve službě google play je nutné si zaregistrovat účet pro vývojáře. To se provede tak, že se uživatel přihlásí pomocí svého google účtu, případně si vytvoří google účet. Následně je nutné přijmout smlouvu pro vývojáře na

[3]Obrázek byl převzat z webu https://developer.android.com/guide/components/activities/activity-lifecycle.

adrese https://play.google.com/apps/publish/signup a zaplatit jednorázový registrační poplatek, který činí 25 $.

Po aktivaci účtu se vývojáři nabídne možnost vytvořit a publikovat aplikaci. Může nastavit typ vydání aplikace podle její verze – interní testování, alfa, beta nebo produkční verze. Dále google play nabízí propracované statistiky vydaných aplikací. Vývojář vidí v přehledné tabulce jeho aplikace, jejich celkový počet stažení a hodnocení uživatelů. Po kliknutí na konkrétní aplikaci se zobrazí stránka s podrobnějšími statistikami, kde lze nalézt například informace o počtu aktuálních instalacích, komentáře uživatelů k aplikaci, počet odinstalování aplikace a počet závažných chyb, které se objevily v aplikace a vedly k jejímu ukončení [7].

Kapitola 4

Sudoku

Sudoku je logická hra, ve které je cílem doplnit do předvyplněného hracího plánu čísla 1 až 9 tak, aby se žádné z čísel neopakovalo ve stejném řádku, sloupci ani čtverci. Na obrázku 4.1 je příklad jednoho z možných zadání sudoku. Hrací plán je v základní variantě rozdělen na 9 čtverců, které jsou na obrázku 4.1 barevně rozlišeny. Tyto čtverce se dále skládají z 9 polí (3x3 pole). Hrací plán má tedy celkem 81 polí.

	1	2	3	4	5	6	7	8	9
a			4	8					
b		9		4	6			7	
c		5					6	1	4
d	2	1		6			5		
e	5	8		7		9		4	1
f			7			8		6	9
g	3	4	5					9	
h		6			3	7		2	
i						4	1		

Obrázek 4.1: Příklad zadání hry sudoku.

Další varianty sudoku se liší hodnotami, které se doplňují do polí, nebo typem hracího pole. Hrací pole se může lišit velikostí (např. 16x16, 25x25) nebo tvarem „čtverců“.

Složitost sudoku lze definovat podle složitosti technik potřebných k jeho vyřešení nebo podle počtu nezadaných polí v sudoku. Myslím si, že obtížnost není přímo závislá na počtu nezadaných polí. Může být zadáno sudoku jen s několika prázdnými poli a k vyřešení bude potřeba využít jedny z komplikovaných technik jako je například X-Wing. Proto považuji rozdělení obtížnosti na základě nutných technik k jeho vyřešení jako přesnější. Techniky lze tedy rozdělit do pěti skupin – jednoduché, střední, pokročilé, mistr a nejobtížnější. Mezi jednoduché techniky patří naked single nebo hidden single. Díky těmto technikám lze vyřešit většinu sudoku, které se objevují v novinách. Středně obtížné techniky už pomáhají spíše jen s redukováním kandidátů než přímo s doplněním čísla do pole. Do této kategorie spadá například locked candidate a na podobném principu založené techniky. Do kategorie pokročilých technik spadají například naked pairs, triplet ... a hidden pairs, triplet ....

Do technik pro úroveň mistr se řadí například X-Wing. Nejobtížnější technika je zkoušet dosazovat možné kandidáty do polí, dokud se nenajde správná kombinace kandidátů [4].

Jelikož se tato práce zabývá pouze základní variantou hry sudoku, budou v následujících sekcích popsány pouze algoritmy pro řešení této varianty hry. Popisované algoritmy vychází z publikace *The Mathematics of Sudoku* [15].

4.1 Backtracking

Backtracking se řadí mezi algoritmy řešící problém „hrubou silou“. Algoritmus spočívá v postupném doplňování možných čísel do všech volných políček v hracím poli, dokud se nenajde správná kombinace čísel vyhovující pravidlům sudoku. Tento algoritmus poskytuje jistotu nalezení řešení pro všechny korektně zadané sudoku. Další výhodou je snadná implementace v porovnání s algoritmy řešící sudoku pomocí komplexních eliminačních technik. Naopak jeho velkou nevýhodou je jeho časová náročnost.

4.1.1 Princip algoritmu

Algoritmus nejprve najde první volné pole. Sestaví pro něj množinu všech přípustných čísel, které se na dané pole mohou doplnit (v případě hry na obrázku 4.2 se pro pole $a1$ sestaví množina čísel 2, 7, 9). Vybere z nich první číslo (číslo 2) a postupuje na další volné pole (v tomto případě na pole $a2$). Pro něj také sestaví množinu kandidátů čísel na doplnění (množina tedy obsahuje číslo 9). Algoritmus postupuje tímto principem dále, dokud nenarazí na pole, do kterého není možné doplnit žádné číslo. Algoritmus se vrátí na předcházející pole, odstraní z množiny kandidátů číslo, které doplnil. Místo něj dosadí následující možnost (u pole $a1$ by to bylo číslo 7) a pokračuje na následující pole, pro které znovu sestaví množinu kandidátů. Takto algoritmus postupně zkouší a doplňuje čísla do polí, dokud nenajde spravný výsledek. V případě špatně zadaného sudoku (neřešitelného), algoritmus skončí jakmile první volné pole sudoku bude mít prázdnou množinu kandidátů.

	1	2	3	4	5	6	7	8	9
a	2 7 9	2 9	1	3	8	2 5 6 7 9	5 9	4	5 7 9
b	5	4	6	7 9	7 9	1	3 9	2	3 7 8 9
c	2 3 7 8 9	2 8 9	2 3 7 8	2 4 5 6 7 9	2 5 6 7 9	2 4 5 6 7 9	1 3 5 9	1 5 6 7 8	3 5 7 8 9
d	6	1 2 8	2 7 8	1 2 5 7	1 2 5 7	2 5 7 8	4	9	2 3 5 7
e	4	2	5	2 7 6 9	3	2 7 6 9	8	7	1
f	1 2 7 8	3	9	1 2 4 5 7	1 2 5 7	2 4 5 7 8	2 5	5 7	6
g	1 2 3 8 9	1 2 5 8 9	2 3 4 8	1 2 5 6 7 9	1 2 5 6 7 9	2 5 6 7 9	1 2 5 9	1 5 8	2 5 8 9
h	1 2 9	7	2	8	1 2 5 9	2 5 9	6	3	4
i	1 2 8 9	6	2 8	1 2 5 9	4	3	7	1 5 8	2 5 8 9

Obrázek 4.2: Ukázka doplnění všech přípustných kandidátu do jednotlivých polí sudoku[1].

[1] Obrázek sudoku hry byl převzat z publikace *The Mathematics of Sudoku* [15].

4.2 Eliminační techniky

Eliminační techniky si na začátku vytvoří pro každé prázdné pole množinu kandidátů. Na rozdíl od backtrackingu se tento přístup snaží zredukovat počet těchto možností tak, aby pro každé pole zbyla jen jedna možnost pro doplnění. Na obrazku 4.2 je příklad sudoku, ve kterém jsou již sestaveny množiny kandidátů pro všechna nevyplněná pole. Principy některých základních eliminačních technik zde budou rozebrány podrobněji.

Výhodou tohoto přístupu je jeho rychlost a efektivita. Naopak nevýhodou je, díky jeho komplexnosti, náročná implementace. Při nepokrytí všech eliminačních technik, nemusí vést vždy ke kompletnímu vyřešení sudoku, jelikož v některých polích může zůstat více kandidátů.

4.2.1 Naked single

Naked single je pojmenování čísla, které se nachází v množině kandidátu jako jediné. Takové číslo lze hned doplnit do hracího plánu. Na obrázku 4.2 tomuto stavu odpovídá množina kandidátů pro pole $e2$, $e8$ nebo $h3$.

4.2.2 Naked pair, triplet, ...

Tato technika je podobná technice naked single s rozdílem, že u této techniky dochází pouze k redukci kandidátů v ostatních polích. Toto pravidlo lze uplatnit u polí, které mají na stejném řádku, sloupci nebo čtverci stejné kandidáty a žádné jiné. V takovém případě pak lze odebrat vyskytující se kandidáty z ostatních polí ze stejného řádku, sloupce nebo čtverce. U sudoku 4.2 lze tento jev pozorovat u polí $b4$ a $b5$, které mají jediné dva kandidáty 7 a 9. Tyto kandidáty lze odstranit z polí $a6, b7, b9, c4, c5$ a $c6$.

Tuto techniku lze také aplikovat i u třech a více kandidátů v poli. V takovém případě nemusí skupina polí obsahovat v každém poli všechny kandidáty, ale může se objevovat takzvaná závislost kandidátů. Takový případ lze vidět například na obrázku 4.2. V polích $a2, a8$ a $a9$ se dohromady objevují pouze tři kandidáti. V jednotlivých polích se, ale objevují vždy pouze dva kandidáti, díky kterým jsou tato pole provázána. Na základě této techniky lze odebrat z pole $a4$ číslo 1 a z pole $a5$ čísla 1 a 3.

4.2.3 Hidden single

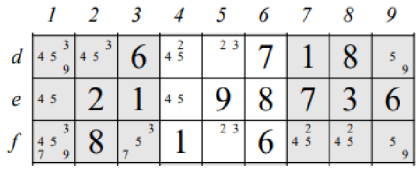
Hidden single je kandidát, který se vyskytuje pouze jednou v celém řádku, sloupci nebo čtverci. Množina však může obsahovat i další čísla. Tento jev můžeme také pozorovat na ukázkovém sudoku 4.2 například na poli $d9$. V tomto poli je jeden z kandidátů (číslo 3), které se nevyskytuje v žádné jiné množině kandidátů polí z tohoto čtverce, proto ho lze doplnit.

4.2.4 Hidden pair, triplet, ...

Tento princip se velmi podobá technice naked pair, triplet popsané v části 4.2.2. Jediný rozdíl mezi těmito technikami je v tom, že v případě hidden pair, triplet se v polích mohou vyskytovat i další kandidáti. Tito kandidáti se díky tomuto pravidlu mohou odstranit.

4.2.5 Locked candidate

Uzamčení kandidáta nastává v případě, že se ve čtverci nachází stejný kandidát ve stejném řádku nebo sloupci. Potom lze tohoto kandidáta odstranit z ostatních polí nacházejícím se na stejném řádku nebo ve stejném sloupci. Tento případ lze pozorovat na obrázku 4.3 v posledním čtverci. Jediná možnost kam lze doplnit číslo 2 je na pole $f7$ a $f8$. Číslo 2 lze tedy odebrat z kandidátů pro pole $f5$. Stejný případ nastává i pro číslo 4 u polí $f7$ a $f8$ a tento kandidát se odebere z pole $f1$.

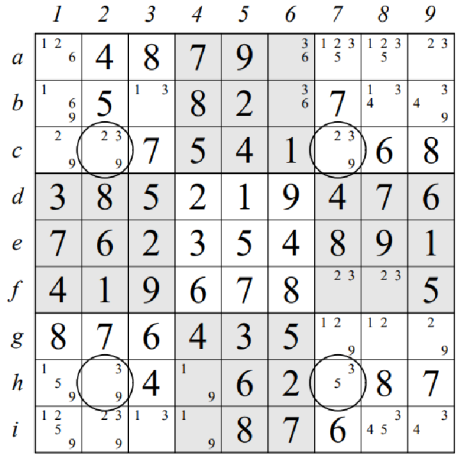


Obrázek 4.3: Ukázka případu uzamčení kandidáta (čísla 2) v posledním čtverci[2].

4.2.6 X-Wing

Techniku X-Wing lze aplikovat v případě, že se objevuje stejný kandidát ve čtyřech různých čtvercích a tyto pole se nachází na dvou stejných řádcích a sloupcích. Pole tedy vytváří pomyslný obdélník, případně čtverec. Pole samozřejmě mohou obsahovat i jiné kandidáty. Princip techniky spočívá ve faktu, že v protilehlých rozích pomyslného obdélníku se musí vyskytovat daní kandidáti. Z ovlivněných řádků lze odstranit daného kandidáta, kromě rohů obdélníka.

V obrázku 4.4 v polích $c2, c7, h2$ a $h7$ se objevuje stejný kandidát číslo 3. Na tyto pole tedy lze aplikovat techniku X-Wing, a proto lze odstranit z polí $a7, f7$ a $i2$ kandidáta číslo 3.



Obrázek 4.4: Ukázka případu, ve kterém lze aplikovat techniku X-Wing na označených polích[3].

[2]Obrázek sudoku hry byl převzat z publikace *The Mathematics of Sudoku* [15].
[3]Obrázek sudoku hry byl převzat z publikace *The Mathematics of Sudoku* [15].

Kapitola 5

Návrh aplikace

V této sekci bude popsán návrh algoritmů detekce hracího plánu sudoku, rozpoznání jednotlivých čísel v polích a grafický návrh aplikace. Aplikace měla v původním návrhu řešit pouze načtení sudoku z novin a nalezení správného řešení. V průběhu testování jsem se na základě připomínek od uživatelů, kteří testovali první verzi aplikace, rozhodl pro doplnění funkcionality o ukládání her a možnosti je později dohrát. Důvod této změny bude popsán v sekci 7.

5.1 Detekce sudoku

Postup detekování hracího pole sudoku, jsem rozvrhl do několika fází. Tato část aplikace je postavena hlavně na principech zpracování digitálního obrazu.

Cílem první fáze je správné nalezení hran. U hledání hran je nutné obraz předzpracovat tak, aby v obraze byla minimální hladina šumu. Dále jsem se ještě rozhodl upravit obraz pomocí prahování. Poslední funkcí využitou v této fázi je aplikace samotného detektoru hran. Výstupem z této části je binární obraz s vyznačenými hranami.

Druhou fází detekce je nalezení rohů hracího plánu na fotografii. Tuto část jsem vyřešil nalezením obrysů všech útvarů na fotografii a vybráním takového útvaru, který zabírá největší plochu. U takového obrysu se předpokládá, že se na něm nachází hrací plán sudoku. Pro takový obrys se následně určí jeho krajní body – rohy.

Ve finální fázi dochází k transformaci obrazu hracího pole. Tímto se opraví nerovnosti podkladu, na kterém je sudoku vytištěno (například ohnutí listu novin). Výstupem navrženého algoritmu je obraz hracího plánu bez nevýznamného okolí, které bylo pořízeno společně s hracím plánem.

5.2 Detekce a rozpoznávání čísel

Nalezené hrací pole se rozdělí do 81 částí (9x9). Jednotlivá pole jsou následně zpracována pro lepší detekci obrysů případných číslic v poli. Z nalezených obrysů se podle plochy vybere ten největší. Jestliže obsah obrysu překoná stanovený práh, považuje se tento obrys za číslo. V takovém případě je obraz pole ještě upraven a vložen na vstup konvoluční neuronové sítě k rozpoznání číslice v poli. Architektura sítě je inspirována architekturou LeNet-5, která byla popsána v sekci 2.5.

5.3 Databáze pro ukládání sudoku her

Aplikace ukládá všechny načtené hry sudoku i data o těchto hrách. Pro uložení dat jsem se rozhodl vytvořit jednoduchou databázi, do které se budou ukládat požadované informace. Databázi jsem si vybral z důvodu snadného rozšíření ukládaných dat o nové položky.

V databázi jsou uchovány následující informace

- identifikační číslo sudoku, podle kterého lze sudoku nalézt
- cestu k souboru s vyfoceným sudoku – fotka se využije jako náhledový obrázek k sudoku hře
- datum a čas pořízení fotografie sudoku
- zadání sudoku
- aktuální stav rozehraného sudoku
- procento políček zbývající k vyřešení

Po kompletním dořešení se sudoku vymaže z databáze. Tato funkce byla do aplikace doplněna na základě zpětné vazby uživatelů, kteří si nepřáli znovu řešit již vyřešené sudoku.

5.4 Řešení sudoku

Algoritmus pro řešení sudoku jsem navrhl tak, že využívá základní eliminační techniky a princip backtrackingu. Z eliminačních technik jsem se rozhodl využít techniky naked single a hidden single. Za použití těchto technik lze většinu sudoku her vyřešit, v opačném případě tyto techniky alespoň poslouží k minimalizaci počtu kandidátů jednotlivých polí sudoku.

V případě, že se sudoku nepodaří vyřešit jen pomocí těchto technik, využije se principu backtrackingu. Doplní se jeden z kandidátů a opět se sudoku začne řešit pomocí eliminačních technik s nově doplněným číslem. Když pro zvoleného kandidáta neexistuje řešení, zvolí se následující kandidát jako u backtrackingu. Postupujeme tak dlouho, dokud nenajdeme takovou kombinaci čísel, které splňují pravidla sudoku. Abych ověřil, zda pro dané sudoku neexistuje více možných řešení, rozhodl jsem se neukončit algoritmus hledání hned po nalezení prvního řešení, ale pokračovat v hledání dalších řešení. V případě, že by algoritmus našel více řešení, znamenalo by to, že zadání sudoku je chybné nebo došlo ke špatné detekci čísel z fotografie hracího pole sudoku. V tomto případě aplikace upozorní uživatele na špatné zadání hry.

5.5 Grafické rozhraní aplikace

Návrh aplikace musí splňovat zásady intuitivnosti a jednoduchosti ovládání bez nutnosti studování návodu. Aplikace je proto navržena tak, aby ji mohlo využívat široké spektrum uživatelů. U této aplikace je to obzvláště důležité. Řešení sudoku je oblíbenou zábavou lidí různých věkových kategoriích, proto jsem se snažil navrhnout ovládání tohoto programu tak, aby byl snadno ovladatelný pro každého uživatele. Nechtěl jsem, aby složité ovládání odradilo některé zájemce od využívání této aplikace.

(a) Řešení sudoku (b) Fotoaparát (c) Seznam nedořešených her

Obrázek 5.1: Návrh grafického rozhraní finální verze mobilní aplikace pro načítání sudoku.

Aplikaci jsem rozdělil do tří částí – aktivit. Jedna z částí se zabývá grafickou podobou hracího pole sudoku. Další z částí je samotný fotoaparát pro načtení fotky sudoku. V této části se také řeší samotná detekce pozice sudoku na fotografii a detekce čísel v polích sudoku. Poslední část, kterou jsem přidal na základě zpětné vazby testerů z prvního kola testování rozebraného v sekci 7, je seznam rozehraných her sudoku. Jednotlivé části aplikace zde popíši podrobněji.

V aplikaci jsem vytvořil menu pro navigaci. Menu obsahuje tlačítka pro spuštění aktivit fotoaparátu nebo historie her, neobsahuje však tlačítko pro spuštění aktivity s hracím plánem sudoku. Na tuto aktivitu se lze dostat pouze načtením sudoku pomocí fotoaparátu nebo ze seznamu nedohraných her.

5.5.1 Hrací pole

Vzhled hracího pole jsem se snažil navrhnout moderně. Hrací pole jsem navrhl jen s hlavními čárami rozdělující hrací pole na jednotlivé čtverce. Pozadí hracího plánu jsem zvolil bíle pro lepší kontrast čísel od pozadí. Pro zadaná pole v sudoku jsem zvolil světle šedou barvu s jemným přechodem do bílé barvy na okrajích. Uživatel pak rychle a jednoduše dokáže rozeznat prázdná a zadaná pole. Aktuálně zvolené pole se liší od ostatních polí silnější čárou rámečku a barvou podkladu. Tlačítka pro zadávání čísel jsou umístěna ve spodní části, což umožňuje jednoduché ovládání pomocí palce. Zanechal jsem základní šedou barvu tlačítek, aby odpovídala barevnému konceptu aplikace. Zbylé tlačítka sloužící pro vymazání čísla, upravení zadání sudoku a nápovědu (doplnění správného čísla do hracího pole) jsem umístil do horní části obrazovky, jelikož se u těchto tlačítek nepředpokládá jejich časté používání. Přesunem těchto tlačítek se uvolnilo více místa pro tlačítka zadávání čísel, takže mohou být širší a proto jednodušší na stisknutí. Přesnou grafickou podobu lze vidět na obrázku 5.1a.

5.5.2 Fotoaparát

Jelikož načtení práce s fotoaparátem mobilu trvá poměrně dlouho dobu, rozhodl jsem se část detekce pozice sudoku rozdělit do dvou podoken – první část sloužící pro vyfotografování sudoku a druhá část pro zobrazení výsledku nalezeného sudoku. Podokno s fotoaparátem je svázáno s aktivitou, po vyfotografování dojde k jejímu překrytí druhým podoknem s nalezeným hracím polem. V případě, že uživatel neodsouhlasí hrací pole, zruší se pouze tato vrstva a uživatel bude mít hned možnost znovu vyfotit obrázek bez čekání na nastartování fotoaparátu.

Grafický návrh fotoaparátu vzhledově odpovídá běžně používaným aplikacím s fotoaparátem. Pro uživatele je pak práce s fotoaparátem jednoduchá a známa. Přes celou obrazovku se uživateli zobrazí obraz z fotoaparátu a nad touto vrstvou se nachází plovoucí tlačítko spouště pro vyfotografování sudoku.

V druhé části dojde k načtení fotografie. Fotografie obsahuje kromě hracího plánu sudoku i jeho blízké okolí. V rámci detekce sudoku dojde k oříznutí nevýznamného okolí a uživateli je zobrazena pouze část obsahující sudoku v horní části obrazovky. Spodní část obrazovky pak obsahuje dvě tlačítka – OK a Cancel, kterými uživatel rozhodne, zda došlo ke správnému rozpoznání sudoku nebo je potřeba sudoku vyfotografovat a zpracovat znovu. Návrh obrazovky s fotoaparátem lze vidět na obrázku 5.1b.

5.5.3 Historie her

Historie her je navržena jako seznam jednotlivých her, ve kterém se po kliknutí na položku hry, otevře daná hra a uživatel ji může dohrát. Hry jsou řazeny pod sebou podle data vyfocení sudoku. Jednotlivé položky obsahují náhled fotky sudoku, datum a čas nahrání sudoku a kolik procent nezadaných políček již uživatel vyplnil. Přesnou grafickou podobu lze vidět na obrázku 5.1c.

Kapitola 6

Implementace

Jelikož je vývoj a odhalování chyb programu na počítači mnohem snazší než na mobilním telefonu, rozhodl jsem se v první fázi vývoje rozdělit projekt do částí. Pro jednotlivé části jsem naprogramoval samostatně spustitelné aplikace pro počítač, které řešily daný problém. Po odladění chyb jsem jednotlivé části spojil do jedné velké aplikace, která již byla spustitelná na operačním systému android.

Mobilní aplikace je naprogramována v programovacím jazyku Java. V projektu využívám několik knihoven. Část pro detekci pozice hracího plánu využívá knihovnu OpenCV, která byla popsána v sekci 1.4.1. Pro rozpoznání čísel jsem se rozhodl využít knihovnu Tensorflow a pro vytvoření a správu databáze jsem využil knihovnu Room.

Od verze operačního systému Android 6.0 se změnily funkce, které pracovaly s kamerou mobilního telefonu. Zvažoval jsem tedy, zda bych měl implementovat i speciální třídu pro fotoaparát, která by podporovala i nižší verze operačního systému. Jelikož procento zařízení, které využívají verze Android 6.0 nebo novější, je 57.7% a s postupem času se toto procento bude dále zvyšovat. Rozhodl jsem se, že moje aplikace bude podporovat operační systém od verze Android 6.0.

6.1 OpenCV

Knihovnu OpenCV jsem do projektu zahrnul pomocí aplikace OpenCV Manager, která je dostupná na portálu Google Play[1]. OpenCV Manager obsahuje všechny verze knihovny OpenCV 2.4.x a OpenCV 3.x přímo optimalizované pro konkrétní procesor mobilního telefonu. Využitím OpenCV Manager se zmenší velikost výsledné aplikace, protože není nutné s aplikací dodávat i verzi OpenCV pro všechny podporované procesory. OpenCV Manager také zajistí, že aplikace bude využívat nejaktuálnější vydání používané verze OpenCV.

Po prvním spuštění aplikace je uživatel vyzván, aby si stáhl aplikaci OpenCV Manager. Tím se zajistí možnost využívat funkce pro zpracování obrazu z této knihovny. Po stáhnutí a nainstalování může uživatel pokračovat a vyfotit obrázek sudoku hry. Při odmítnutí nainstalovat OpenCV Manager, aplikace neumožní uživateli vyfotografovat sudoku.

[1] https://play.google.com/store/apps/details?id=org.opencv.engine

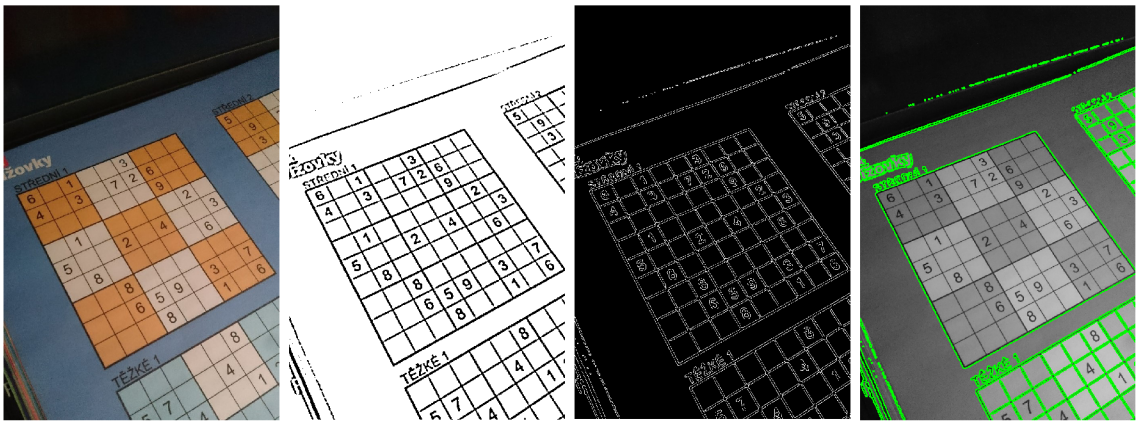
6.2 Detekce sudoku

Detekci hracího pole sudoku jsem implementoval podle návrhu popsaného v sekci 5.1. Pro lepší orientaci v kódu, jsem rozdělil detekci sudoku do fází, kde každé fázi odpovídá jedna metoda.

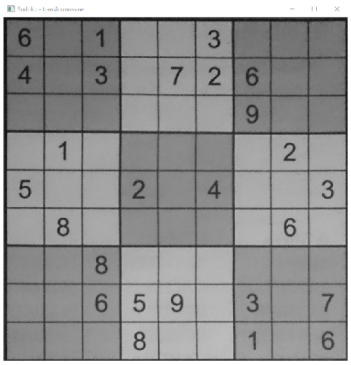
Abych dosáhl přesnější detekce hran, zkoušel jsem různé funkce zpracování obrazu. Nejlepší výsledky jsem dosáhl s kombinací několika funkcí. První krok algoritmu je změna velikosti fotografie tak, aby každá byla široká 700 pixelů (výška fotografie se dopočítá tak, aby zůstal zachován původní poměr stran). Tím je zajištěno, že algoritmus bude pracovat vždy se stejně velkými fotografiemi. Pro vyhlazení obrazu algoritmus využívá funkci `GaussianBlur`. Velikost konvolučního jádra gaussova rozostření jsem zvolil 11x11. Po několika experimentech jsem zjistil, že tato velikost je ideálním kompromisem mezi časem výpočtu a odstíněným šumem z obrazu. Dalším krokem je aplikace adaptivního prahování (OpenCV funkce `adaptiveThreshold`). To se osvědčilo hlavně u sudoku, ve kterém byly jednotlivé čtverce rozlišeny barvami. U adaptivního prahování jsem zvolil velikost okolí 11x11, ze kterého se počítá hodnota prahu tohoto okolí, a pro výpočet hodnoty prahu jsem využil klasického průměrování hodnot v okolí (parametr `ADAPTIVE_THRESH_MEAN_C`).
Poslední funkcí využitou v této fázi je Cannyho detektor hran (OpenCV funkce `Canny`). U této funkce jsem zvolil velikost Sobelova filtru jako 5x5. Hodnotu minimálního prahu hystereze jsem vybral 150 a maximální práh je třikrát větší, jelikož v dokumentaci k OpenCV se tento poměr uvádí jako nejvhodnější. Díky použití těchto funkcí se výrazně zlepšily výsledky hledání hran.

K nalezení obrysů jsem se rozhodl využít funkci `findContours` z knihovny OpenCV s parametry `RETR_EXTERNAL` a `CHAIN_APPROX_SIMPLE`. Parametr `RETR_EXTERNAL` zajistí, že funkce vrátí pouze vnější obrysy, což ulehčí následné zpracování nalezených obrysů. Pole, do kterého funkce uloží nalezené obrysy, projde algoritmus a snaží se najít obrys s největší plochou – obrys na kterém je hrací plán. Při hledání obrysu jsem využil parametr `CHAIN_APPROX_SIMPLE`, který by měl zajistit pouze přibližné uložení obrysů. Použití tohoto parametru by mělo zajistit, že jsou v poli uchovány pouze body, na kterých došlo k velké změně směru. I přesto mohou být obrysy uloženy jako několik bodů. Proto je nutné ještě body obrysu aproximovat, aby bylo možné obrys zapsat pouze čtyřmi body – pouze rohy obrysu. Pro aproximaci využívám funkci `approxPolyDP`. U této funkce se v dokumentaci OpenCV doporučuje využít hodnotu epsilon jako 0.05-ti násobek obvodu obrazce. Jestliže ani po aproximaci bodů algoritmus nenalezne přesně 4 body, považuje se detekce hracího plánu jako neúspěšná a uživatel je požádán o opakované vyfocení plánu.

Pro korektní transformaci je nejprve nutné správně seřadit rohy. Funkce `getPerspectiveTransform`, kterou jsem se rozhodl použít, očekává rohy zadané v protisměru hodinových ručiček se začátkem v levém horním rohu. Princip algoritmu pro správné setřídění nalezených rohů hracího plánu spočívá v hledání minima a maxima součtu nebo rozdílu souřadnic bodů. Pro levý horní roh platí, že součet souřadnic bude nejmenší ze součtu souřadnic ostatních bodů, pro pravý spodní roh platí, že tento součet bude největší. Pozice zbylých bodů se určí rozdílem x-ové a y-ové souřadnice. Pravý horní roh má tento rozdíl největší a levý spodní roh nejmenší. Funkce `getPerspectiveTransform` vytvoří transformační matici, která se posléze využije jako vstup funkce `warpPerspective` pro transformaci nalezeného sudoku. Jednotlivé fáze algoritmu jsou znázorněny na obrázku 6.1.



(a) Originální fotografie (b) Výsledek adaptivního prahování (c) Výsledek cannyho hranového detektoru (d) Znázornění nalezených hran v obraze



(e) Výsledek algoritmu pro detekci pozice hracího plánu

Obrázek 6.1: Fotografie zachycují kroky potřebné pro správnou detekci hracího plánu sudoku ve fotografii.

6.3 Detekce a rozpoznání čísel

Hrací pole se rozdělí na 9x9 částí – polí. Jednotlivá pole se pak postupně pošlou na zpracování. Nejprve je obraz upraven pomocí funkcí `GaussianBlur` a `adaptiveThreshold`. V upraveném obrazu pole se pomocí funkce `findContours` najdou všechny obrysy. Kolem každého obrysu je vytvořen ohraničující obdélník. Na základě velikosti plochy takto vytvořeného obdélníku se nalezne největší z nich. Obsah největšího obdélníku se porovná s prahem, který určuje zda se na tak velké ploše může nacházet číslice. Hodnotu prahu jsem určil 300. V případě, že je plocha menší než tato hodnota, předpokládá se, že pole neobsahuje žádnou číslici, a pokračuje se na další políčko. V opačném případě se toto pole upraví, aby mělo velikost 32x32 pixelů a je předáno konvoluční neuronové síti pro analýzu.

Konvoluční síť jsem trénoval na několika datasetech. Jako hlavní trénovací sadu jsem zvolil dataset MNIST, který byl popsán v sekci 2.6.1. Síť jsem pak ještě dotrénoval na fotografiích jednotlivých polí z hracího plánu sudoku. Tyto fotografie polí jsem vytvořil pomocí testovacího programu, který bude popsán v sekci 7.3. Ručně jsem k těmto obrázkům vytvořil popis, jaké číslo se na dané fotografii nachází. Síť dosáhla při trénování přesnosti 95,6% na mém datasetu.

6.4 Řešení sudoku

Při řešení sudoku se nejprve zkontroluje, zda je sudoku zadané korektně a splňuje všechny pravidla sudoku. Špatně zadané sudoku způsobí ukončení řešení sudoku s příslušnou chybovou návratovou hodnotou. U korektního sudoku se postupuje dále k naplnění polí všemi kandidáty a následně k samotnému řešení sudoku pomocí eliminačních technik. Řešení pomocí eliminačních technik skončí v případě, že již nelze odstranit další kandidáty – tedy v aktuálním průchodu sudoku hry nedošlo k žádnému odebrání kandidáta. Zkontroluje se tedy, zda eliminační techniky nalezly řešení. V takovém případě funkce skončí s návratovým kódem pro úspěšně nalezené řešení. V opačném případě se přistupuje k hledání řešení s využitím backtrackingu. Při tomto způsobu řešení se musí nejprve uložit aktuální stav vyplněných polí v sudoku a kandidáti pro jednotlivá pole. Rekurzivně se zavolá tato funkce na řešení sudoku a na základě návratového kódu se rozhodne, jaký bude další postup. Jestliže funkce nenalezla správné řešení, obnoví se původní stav doplněných polí v sudoku a odpovídající kandidáti. Do pole se doplní následující kandidát a algoritmus se pokusí nalézt řešení pro tohoto kandidáta. V případě, že další kandidát již neexistuje, funkce skončí s kódem pro nevyřešené sudoku. I po nalezení správného řešení se funkce vyvolá s dalším kandidátem, aby se potvrdila korektnost zadání.

6.5 Databáze pro ukládání sudoku her

Databázi jsem implementoval pomocí knihovny `Room`. Knihovna nabízí velmi jednoduché a rychlé vytvoření databáze. Pro správné fungování knihovny `Room` je nutné navíc vytvořit několik tříd. V hlavní třídě, která je uvozena příkazem `@Dao`, se definuje samotná databáze. Třída navíc slouží i jako přístupový bod k takto vytvořené databázi, jelikož spravuje všechny její instance v aplikaci. Další třída obsahuje operace nad touto databází. Operace jsou definovány formou klasických funkcí s rozdílem, že na předchozím řádku zdrojového kódu těchto funkcí se vyskytuje klíčové slovo `@Query(...)`, které obsahuje dotaz v jazyce SQL. Knihovna sama doplní do SQL dotazu parametry z funkce a výsledek dotazu namapuje na návratový typ funkce. Tabulky databáze jsou definovány v jednotlivých třídách, které jsou uvozeny příkazem `@Entity` a obsahují jméno a datový typ sloupců databáze [9].

6.6 Grafické rozhraní aplikace

Při implementaci grafického rozhraní aplikace jsem dodržoval stanovený návrh popsaný v sekci 5.5. K tomu jsem využil základních objektů API Androidu. V této sekci budou popsány pouze implementace obtížnějších částí grafického rozhraní.

Menu jsem vytvořil pomocí objektů `DrawerLayout` a `NavigationView`.

6.6.1 Hrací pole

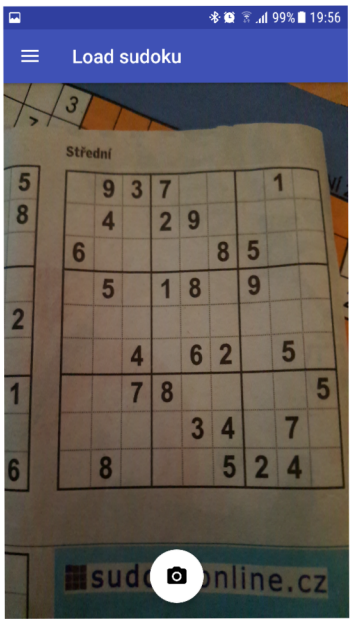
Pole hracího plánu sudoku jsou generovány dynamicky do objektu `TableLayout`. Při generování je jim nastaven text (číslice), funkce, pro zpracování události kliknutí a případně je nastaveno pozadí podle toho, zda se jedná o zadané pole. Funkce pro zpracování kliknutí změní vzhled pole a uloží jeho adresu, aby aplikace věděla, kam se má případně zapsat uživatelem zadané číslo. Pro zadání číslice uživatel použije tlačítka s číslicemi ve spodní části aktivity.

6.6.2 Fotoaparát

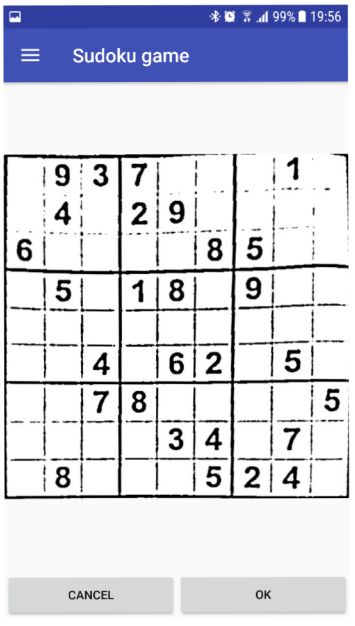
Při implementaci fotoaparátu jsem využil demo aplikace na webu android studia [5]. Náhled obrazu z fotoaparátu se zobrazuje ve vytvořeném objektu `AutoFitTextureView`. Detekovaný obraz hracího pole se pak zobrazí v objektu `ImageView` ve zcela novém podokně společně s potvrzovacími tlačítky.

6.6.3 Historie her

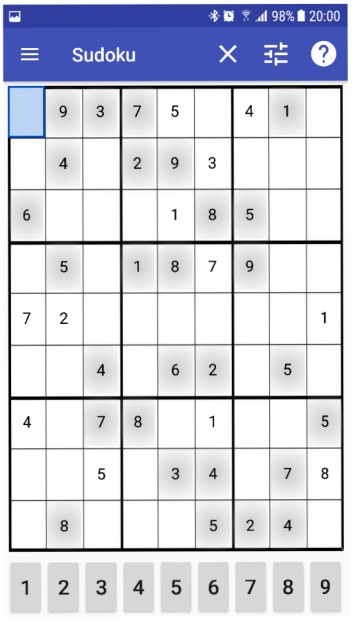
Seznam položek her je implementován pomocí objektu `ListView`. Pro správné fungování tohoto listu jsem implementoval adaptér, který zajišťuje navázání dat z databáze do položek seznamu.



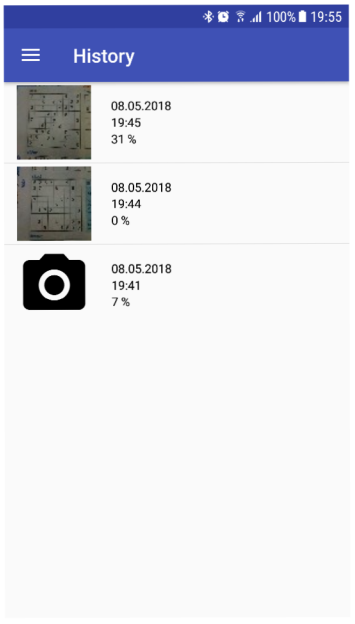
(a) Fotoaparát aplikace



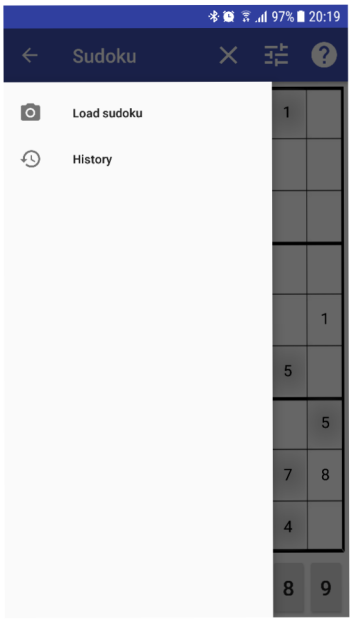
(b) Zobrazení detekovaného sudoku



(c) Prostředí pro řešení sudoku



(d) Výsledek algoritmu pro detekci pozice hracího plánu



(e) Menu aplikace

Obrázek 6.2: Fotografie zobrazují grafické rozhraní výsledné aplikace pro mobilní telefon.

Kapitola 7

Uživatelské a aplikační testování

Pro jednodušší testování a organizaci projektu v první fází vývoje jsem se rozhodl vytvořit několik menších aplikací, které jsou samostatně spustitelné na počítači. První aplikace se zabývá vyřešením sudoku. Aplikaci jsem naprogramoval v jazyku Java, aby se mohla jednodušeji spojit s finální aplikací. Cíl druhé aplikace byl detekovat pozici hracího pole sudoku z obrázku načteného z počítače. Tuto aplikaci jsem napsal v programovacím jazyku C++.

Nejprve jsem samostatně otestoval tyto dílčí aplikace zabývající se konkrétními problémy. Pro snadnější ověření funkčnosti jsem vytvořil automatické testy, které kontrolovaly zda výstup aplikace odpovídá očekávanému výsledku. Při těchto testech jsem současně zjišťoval časovou náročnost řešení a procento špatných výsledku. Testy budou podrobněji popsány pro jednotlivé části aplikace. Po odladění aplikačních chyb jsem přešel k uživatelskému testování, kdy uživatelé testovali kompletní aplikaci.

7.1 Uživatelské testování grafické podoby aplikace

Jako první jsem vytvořil mobilní aplikaci, která obsahovala pouze grafickou podobu aplikace podle návrhu uvedeného v sekci 5.5. Aplikace umožňovala pouze přepínání obrazovek, aby si uživatel vyzkoušel intuitivnost ovládání a mohl mně poskytnout zpětnou vazbu a případné připomínky ke grafickému návrhu a jejímu ovládání.

Aplikaci jsem nechal vyzkoušet 20 kamarádům. Po otestování jsem se každého z nich zeptal, co si myslí o grafickém návrhu, co by chtěl v aplikaci změnit, co by navrhoval přidat nebo odebrat. Z této diskuze vzešel nápad na přidání funkce ukládání nedohraných sudoku her. Další připomínky byly k výběru ikon na obrazovce, kde se řeší sudoku. Ikony se testerům zdáli zavádějící a neintuitivní.

Tyto připomínky jsem považoval za přínosné a proto jsem je začlenil do druhé verze návrhu aplikace, která již obsahovala obrazovku se seznamem nedohraných her a s upravenými ikonami. Aplikaci jsem nechal opět otestovat uživatele. Do druhého kola jsem zahrnul i nové testery. Uživatelé tentokrát neměli žádné výhrady a byli s grafickou podobou aplikace spokojeni.

7.2 Testování části řešení sudoku

Pro zkontrolovaní správnosti a rychlosti algoritmu řešení sudoku jsem vytvořil testovací program. Jako vstup testovacího programu jsem využil zadání sudoku her z datasetu dostupného na webu Kaggle [19]. Tento dataset obsahuje 1000000 zadání sudoku i s jejich

řešením. V průměru mají jednotlivé sudoku hry 42,9 nevyplněných polí. V popisu datasetu se uvádí, že jsou rovnoměrně zastoupeny všechny úrovně obtížnosti. Dataset je uložen ve formátu csv. Zadání a vyřešené sudoku jsou mezi sebou odděleny čárkou.

Testovací program je napsán v jazyku Java. Program využívá vytvořené třídy pro vyřešení sudoku a kromě správnosti programem nalezeného řešení také testuje kolik času bylo potřeba k vyřešení daného sudoku. Všechny sudoku z datasetu program vyřešil správně. Průměrná časová náročnost vyřešení sudoku her byla 13,5 ms. Při hledání řešení sudoku algoritmus nejprve využije eliminační metody, v případě ze se nenalezne řešení, využije se principu backtrackingu. U většiny testovaných sudoku se řešení našlo jen za pomoci eliminačních technik.

Vzhledem k tomu, že doba nalezení řešení sudoku je výrazně odlišná za použití pouze eliminačních technik v porovnání s využitím backtrackingu, rozhodl jsem se doplnit do výstupu programu i podrobnější statistiku. Ve statistice se samostatně počítá doba pro nalezení řešení sudoku s využitím pouze eliminačních technik a s využitím backtrackingu. Pro 21.8% sudoku z datasetu bylo nutné využít principu backtrackingu. Průměrná doba tohoto řešení byla 16,4 ms a u zadání, ve kterém stačilo využít eliminační techniky, byla průměrná doba řešení 12,9 ms.

7.3 Testování části zjištění pozice hracího plánu ve fotografii

Testovací program pro detekci pozice hracího plánu sudoku jsem naprogramoval v jazyce C++. Pro otestovaní jsem využil fotky sudoku z volně dostupného repositáře na serveru Github [23], který obsahoval více než 200 fotografií sudoku. Repositář obsahuje fotografie pořízené z různých mobilních zařízení v různém rozlišení. Dalších přibližně 50 fotografií různých zadání sudoku jsem pořídil sám. Mým cílem bylo získat co nejvíce zadání sudoku s různým vzhledem, abych mohl lépe otestovat stabilitu a přesnost mého detektoru. Proto jsem sudoku fotil z několika různých novin a časopisů.

První fáze spočívala v ručním zadání pozice rohů hracího plánu sudoku. Pro tento účel jsem vytvořil program, který postupně zobrazoval všechny fotografie se sudoku. Aplikace čeká, až uživatel určí pozici rohů hracího plánu jednotlivých fotografií se sudoku, tím že na ně klikne. Po zadání rohů u všech fotografií aplikace vygeneruje csv soubor. Jednotlivé řádky csv souboru představují jednu fotografii, první sloupec obsahuje název fotografie a v dalších osmi polích jsou uloženy souřadnice zadaných rohů.

Druhá fáze již spočívala v samotném testování přesnosti detekce hracího plánu. V testovacím programu jsem porovnával souřadnice rohů nalezené mým detektorem s ručně zadanými souřadnicemi z csv souboru nebo případně nerozpoznání hracího plánu. Nerozpoznání hracího plánu se projeví po aproximaci nalezeného obrysu plánu tím, že počet výsledných bodů nebude roven čtyřem. Nesprávná detekce pozice hracího plánu byla způsobena dvěma problémy – nízkým rozlišením fotografie a malou vzdáleností mezi hracím plánem a úvodním textem sudoku.

Algoritmus správně určil pozici rohů u 71,4% fotografií. Takto nízká úspěšnost byla způsobeno nízkým rozlišením některých fotografií. Po odstranění fotografií, které měly na šířku méně než 700 pixelů, se toto procento zvětšilo na 97,3%. Další problémy při detekci nastaly pouze v případě, že nad hracím plánem sudoku byl text a mezera mezi sudoku a textem byla velmi malá. V tomto případě funkce pro nalezení vnějších obrysů nezaznamenala tuto mezeru a vyhodnotila hrací plán a text jako jeden objekt, čímž došlo ke špatnému vypočítání rohů sudoku a tedy i ke špatné transformaci fotografie.

Testovací program navíc vytvořil fotografie o rozměrech 28x28 pixelů, které reprezentovaly jednotlivá pole z nalezených sudoku her. Program využíval pouze taková pole, která v sobě obsahovala nějaké číslo. Takto vytvořené obrázky se ukládaly pro pozdější otestování úspěšnosti natrénované konvoluční neuronové sítě pro rozpoznávání čísel.

7.4 Testování části rozpoznávání čísel z fotografie

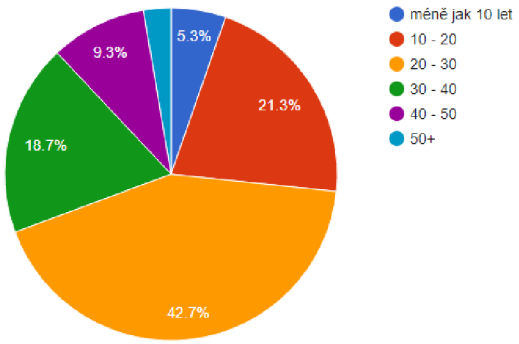
Program pro testování úspěšnosti rozpoznávání čísel z fotografie je napsán v jazyce Python. Program využíval zbytek fotografií čísel, které vytvořil testovací program z kapitoly 7.3 Ručně jsem k těmto obrázkům vytvořil popis, jaké číslo se na dané fotografii nachází. Program dosáhl na této trénovací sadě úspěšnosti 93.7%.

7.5 Uživatelské testování finální aplikace

Testování finální verze aplikace probíhalo již na mobilních telefonech. Tento typ testování probíhal ve dvou fázích. V první fázi jsem se zaměřil na otestování funkčnosti celé aplikace. Snažil jsem se zjistit, zda proběhlo spojení jednotlivých části úspěšně. Ve druhé fázi jsem se zaměřil na celkový dojem z aplikace.

Pro první fázi testování jsem opět využil pouze mých kamarádů, aby zjistili, zda v aplikaci fungují základní funkce. Po odladění základních chyb jsem přešel do finální fáze testování.

Do druhé fáze testování aplikace jsem zahrnul více lidí. V tomto finálním testování jsem se snažil získat testovací uživatele z různých věkových skupin. Pro snadnější vyhodnocení uživatelského testování jsem vytvořil online dotazník pomocí webové služby Google forms. Přesné otázky dotazníku lze vidět v příloze A. Dotazník jsem koncipoval tak, že uživatelé vybírají na kolik procent jsou s mým řešením aplikace spokojeni. Do dotazníku jsem zahrnul 7 kritérií umožňující hodnotit aplikaci z různých pohledů. Stupnice je od 0 do 10, kde 0 je špatné a 10 je výborné. Výsledky jsem prezentoval v procentuálním vyjádření ze zadaných hodnot. Testování se celkem zúčastnilo 74 lidí. Na obrázku 7.1 lze vidět zastoupení věkových skupin, které se zúčastnily finálního testování. Ohlasy na aplikaci byly velmi pozitivní. Vzhled aktivity pro řešení sudoku se uživatelům líbil na 87,7%. Celkový vzhled aplikace si však vysloužil 94,3%. Aplikace se zdála uživatelům intuitivní z 89,1%. Úspěšnost detekce hracího pole sudoku byla u uživatelů 96,5%. Rozpoznání čísel v polích pak bylo úspěšné z 92,6%.



Obrázek 7.1: Graf zastoupení věkových skupin v testování.

Kapitola 8

Závěr

Výsledkem této práce je mobilní aplikace vyvinutá pro operační systém android. Aplikace umožňuje uživateli načíst sudoku do mobilního telefonu. Tento postup je velice jednoduchý. Uživatel jen vyfotí sudoku v novinách. Aplikace nalezne pozici hracího pole sudoku ve fotografii, rozpozná jednotlivé pole a číslice v zadaných polích. Uživatel pak může načtené sudoku vyřešit v mobilním telefonu vyplněním prázdných polí. Nevyřešené sudoku se uloží v mobilním telefonu, dokud je uživatel nevyřeší. Pro řešení sudoku si uživatel může vybrat ze seznamu rozehraná nebo nově načtená sudoku.

Součástí bakalářská práce je také vyhodnocení uživatelského testování aplikace. Uživatelé hodnotili aplikaci pozitivně.

Aplikaci by bylo možné ještě v budoucnu vylepšit. Krom zjevných vylepšení jako jsou například přesnější detekce pozice hracího pole nebo vylepšení neuronové sítě pro rozpoznávání číslic v poli, by se mohla aplikace rozšířit o možnost vygenerovat zadání sudoku, aby uživatel nemusel vždy načítat sudoku z novin. Dále by se do aplikace mohl vložit element soutěžení. K tomuto rozšíření by bylo potřeba připojení k internetu. Aplikace by po spuštění kontaktovala server a stáhla by aktuální soutěžní zadání sudoku – pro každý den by se generovalo nové zadání. Jakmile by uživatel vyřešil sudoku, jeho čas by se odeslal na server, kde by se uchovávalo průběžné pořadí všech soutěžících. V režimu soutěže by byla funkce nápovědy vypnuta.

Literatura

[1] FastCV Computer Vision SDK. online, navštíveno 2018-02-14.
URL https://developer.qualcomm.com/software/fastcv-sdk

[2] Smoothing Images. online, navštíveno 2018-04-28.
URL https://docs.opencv.org/3.1.0/d4/d13/tutorial_py_filtering.html

[3] THE MNIST DATABASE of handwritten digits. online, navštíveno 2018-02-09.
URL http://yann.lecun.com/exdb/mnist/

[4] Techniques For Solving Sudoku. online, 2014, navštíveno 2018-01-20.
URL http://www.sudokuoftheday.com/techniques/

[5] Android Camera2Basic Sample. online, 2016, navštíveno 2018-01-04.
URL https://developer.android.com/samples/Camera2Basic/

[6] Distribution dashboard. online, 2018, navštíveno 2018-03-02.
URL https://developer.android.com/about/dashboards/

[7] Jak používat službu Play Console. online, 2018, navštíveno 2018-03-23.
URL https://support.google.com/googleplay/android-developer/answer/6112435?hl=cs

[8] Platform Architecture. online, 2018, navštíveno 2018-05-01.
URL https://developer.android.com/guide/platform/

[9] Save data in a local database using Room. online, 2018, navštíveno 2018-03-27.
URL https://developer.android.com/training/data-storage/room/

[10] Abadi, M.; Agarwal, A.; Barham, P.; aj.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015, software available from tensorflow.org.
URL https://www.tensorflow.org/

[11] Bradski, G.: The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[12] Canny, J.: A Computational Approach to Edge Detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, ročník 8, č. 6, Červen 1986: s. 679–698, ISSN 0162-8828, doi:10.1109/TPAMI.1986.4767851.
URL http://dx.doi.org/10.1109/TPAMI.1986.4767851

[13] Cohen, G.; Afshar, S.; Tapson, J.; aj.: EMNIST: an extension of MNIST to handwritten letters. *arXiv preprint arXiv:1702.05373*, 2017.

[14] Conder, S.; Darcey, L.: *Android Wireless Application Development*. Addison-Wesley Professional, 2009, ISBN 9780321627094.

[15] Davis, T.: The Mathematics of Sudoku. online, 2012, navštíveno 2018-02-24.
URL http://www.geometer.org/mathcircles/

[16] Jia, Y.; Shelhamer, E.; Donahue, J.; aj.: Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[17] Lecun, Y.; Bottou, L.; Bengio, Y.; aj.: Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, ročník 86, č. 11, Nov 1998: s. 2278–2324, ISSN 0018-9219, doi:10.1109/5.726791.

[18] O'Shea, K.; Nash, R.: An Introduction to Convolutional Neural Networks. 11 2015.

[19] Park, K.: 1 million Sudoku games. online, 2016, navštíveno 2017-11-17.
URL https://www.kaggle.com/bryanpark/sudoku/data

[20] Roy, P.; Dutta, S.; Dey, N.; aj.: Adaptive thresholding: A comparative study. In *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, July 2014, s. 1182–1186, doi:10.1109/ICCICCT.2014.6993140.

[21] Szeliski, R.: *Computer Vision: Algorithms and Applications*. Texts in Computer Science, Springer London, 2010, ISBN 9781848829350.

[22] Vondrák, I.; škola báňská Technická univerzita Ostrava. Fakulta elektrotechniky a informatiky, V.: *Umělá inteligence a neuronové sítě*. VŠB - Technická univerzita Ostrava, 2009, ISBN 9788024819815.

[23] Wicht, B.; Hennebert, J.: Camera-based Sudoku recognition with deep belief network. In *Soft Computing and Pattern Recognition (SoCPaR), 2014 6th International Conference of*, IEEE, 2014, s. 83–88.
URL https://github.com/wichtounet/sudoku_dataset

[24] Young, I.; J. Gerbrands, J.; Van Vliet, L.; aj.: Fundamentals Of Image Processing. 07 2004.

Příloha A

Uživatelský dotazník

Dotazník k testování mobilní aplikace Sudoku

*Povinné pole

1. **Věk** *

Označte jen jednu elipsu.

- ◯ méně jak 10 let
- ◯ 10 - 20
- ◯ 20 - 30
- ◯ 30 - 40
- ◯ 40 - 50
- ◯ 50+

2. **Jak často používáte mobilní telefon** *

Označte jen jednu elipsu.

	0	1	2	3	4	5	6	7	8	9	10	
Nepoužívám	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Často

3. **Vzhled obrazovky s hracím polem sudoku**

Označte jen jednu elipsu.

	0	1	2	3	4	5	6	7	8	9	10	
Nelíbí se mně	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Líbí se mně

4. **Celkový vzhled aplikace**

Označte jen jednu elipsu.

	0	1	2	3	4	5	6	7	8	9	10	
Nelíbí se mně	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Líbí se mně

5. **Intuitivnost ovládání (přehlednost a jednoduchost ovládání)**

Označte jen jednu elipsu.

	0	1	2	3	4	5	6	7	8	9	10	
Špatné	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Dobré

6. **Procentuální úspěšnost automatické detekce pozice hracího pole sudoku**
Označte jen jednu elipsu.

	0	1	2	3	4	5	6	7	8	9	10	
nedošlo ke správnému rozpoznání hracího plánu	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	vždy došlo ke správnému rozpoznání hracího plánu

7. **Procentuální úspěšnost automatického rozpoznávání číslic v hracím plánu**
Označte jen jednu elipsu.

	0	1	2	3	4	5	6	7	8	9	10	
nedošlo ke správnému rozpoznání číslic	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	vždy došlo ke správnému rozpoznání číslic

8. **Doporučil byste aplikaci svým známým**
Označte jen jednu elipsu.

- ◯ Ano
- ◯ Ne

9. **Připomínky (nedostatky aplikace, nápady na vylepšení)**

Děkuji za vyplnění dotazníku