Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Aplikace pro stanovení podobnosti obrazců pomocí Hopfieldovy neuronové sítě

Bc. Jakub Jakoubek

Diplomová práce

2019

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2018/2019**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Jakub Jakoubek**
Osobní číslo:	**I17205**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Aplikace pro stanovení podobnosti obrazců pomocí Hopfieldovy neuronové sítě**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

Cílem práce je vytvoření aplikace pro platformu Android, která bude stanovovat podobnosti obrazců. Software umožní nakreslit pomocí dotykového displeje obrazec, který bude následně podroben zkoumání pomocí Hopfieldovy sítě za účelem detekce podobnosti se vzorem uloženým v paměti sítě. Software bude umožňovat uživatelsky příjemný import sady obrázků jako vzory pro rozpoznávání a intuitivní možnosti nastavení parametrů trénování. Součástí práce bude komplexní případová studie se statistickým vyhodnocením vlastností aplikace. Teoretická část:Stručná rešerše problematiky umělých neuronových sítí, rešerše problematiky Hopfieldovy sítě, rešerše podobných aplikací. Stručná rešerše možností vývoje aplikací pro Android OS, rešerše nástrojů implementace umělých neuronových sítí pro mobilní platformy. Praktická část: Tvorba aplikace, komplexní testování aplikace, dokumentace popisující případovou studii, stručná uživatelská příručka.

Rozsah grafických prací:

Rozsah pracovní zprávy: **60 s**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

HAYKIN, Simon S. Neural Networks: a comprehensive foundation New York: Macmillan College Publishing Company, 1994. ISBN 0-02-352761-7
COUGHLIN, James P. a Robert H. BARAN. Neural computation in Hopfield networks and Boltzmann machines. Cranbury, NJ: Associated University Presses, 1995. ISBN 0874134641
AO, Sio-Iong., Burghard B. RIEGER a Mahyar A. AMOUZEGAR. Machine learning and systems engineering, New York: Springer, 2010. ISBN 9789048194193

Vedoucí diplomové práce: **doc. Ing. Petr Doležel, Ph.D.**
Katedra řízení procesů

Datum zadání diplomové práce: **22. října 2018**

Termín odevzdání diplomové práce: **18. května 2019**

Ing. Zdeněk Němec, Ph.D.
děkan

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 17. listopadu 2018

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 14. 5. 2019

Bc. Jakub Jakoubek

PODĚKOVÁNÍ

Tímto bych chtěl poděkovat doc. Ing. Petru Doleželovi, Ph.D., že mi byl dobrým vedoucím a že si na mě našel čas vždy, když jsem potřeboval. Také bych rád poděkoval všem blízkým, kteří mi byli oporou v době psaní této práce.

ANOTACE

Práce se zabývá tvorbou mobilní aplikace schopné rozeznávat obrazce za pomocí Hopfieldovy neuronové sítě. V teoretické části jsou popsané principy neuronových sítí a následně je podrobněji popsaná Hopfieldova síť. Také jsou představeny některé aplikace zabývající se podobnou problematikou. Druhá polovina práce je zaměřena na samotný vývoj aplikace.

KLÍČOVÁ SLOVA

Neuronové sítě, Hopfieldova neuronová síť, rozpoznávání obrazců, Android

TITLE

Application for shape similarity using Hopfield neural network

ANNOTATION

Thesis deals with creation of mobile application capable of recognize drawn pattern with help of Hopfield neural network. Theoretic part describes principles of neural networks and more specifically Hopfield neural network. There are also introduced few applications that deals with similar problematics. Second part of thesis is focused on creation of application itself.

KEYWORDS

Neural networks, Hopfield neural network, drawn pattern recognition, Android

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZKRATEK

OS	Operation System
REST	Representational State Transfer
RPC	Remote Procedure Call
API	Application Programming Interface
CPU	Central processing unit
GPU	Graphic processing unit
NPU	Neural processing unit
SDK	Software development kit
RAM	Random access memory
XML	Extensible Markup Language

ÚVOD

Cílem diplomové práce je vytvořit aplikaci pro mobilní platformu android, jenž dokáže rozpoznat obrazec, který uživatel nakreslí. K rozpoznání obrazce aplikace využije umělou inteligenci v podobě Hopfieldovy neuronové sítě, kterou bude možné naučit libovolné obrazce importované do aplikace v podobě obrázků.

Umělé neuronové sítě a strojové učení obecně jsou výpočetně náročné. Touto prací bych chtěl dokázat, že lze naprogramovat neuronovou síť, která je vhodná pro mobilní zařízení. Rád bych tedy, aby aplikace neměla vysoké nároky na hardware, ale také aby vykazovala takovou přesnost, jaká se dá od dané neuronové sítě očekávat.

V první kapitole bude stručně popsaná problematika umělých neuronových sítí. Dále zde bude popsán základní stavební prvek sítí, neuron, a závěr kapitoly se zaměří na metodiky učení umělých neuronových sítí.

Druhá kapitola se bude podrobněji věnovat Hopfieldově neuronové síti. Bude zde popsán princip učení sítě a stanovení její odezvy, a to včetně příslušných vzorců. Závěr kapitoly bude věnován možným využitím sítě v praxi a také jejímu kapacitnímu omezení.

Třetí kapitola bude obsahovat stručnou rešerši na již existující vybraný software zabývající se rozpoznáváním obrazu.

Čtvrtá kapitola se zaměří na vývoj aplikací pro OS Android. Součástí kapitoly bude také rešerše nástrojů pro implementaci komplexních neuronových sítí na moderní mobilní zařízení.

Praktická část začne pátou kapitolou, kde bude popsán návrh aplikace. Nejprve budou rozebrány jednotlivé případy užití a poté se přejde k návrhu jednotlivých tříd.

Implementace aplikace bude popsána v šesté kapitole. Na začátku budou zmíněny použité technologie a poté budou postupně probrány jednotlivé třídy aplikace z pohledu programátora. Na konci kapitoly bude popsáno, jak aplikace pracuje se soubory.

V předposlední kapitole bude popsáno testování aplikace z hlediska přesnosti neuronové sítě. Také zde budou shrnuty výsledky testování.

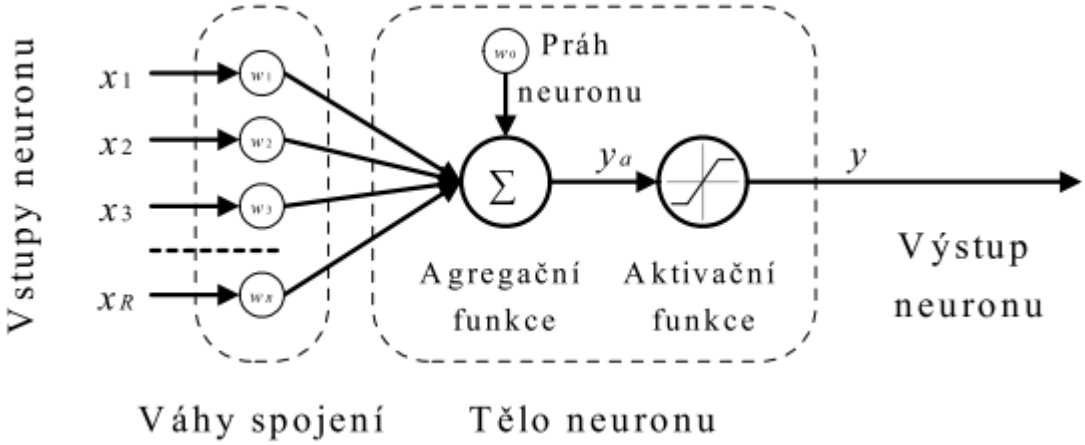
Poslední kapitolou pak bude stručná uživatelská příručka. Ta bude obsahovat popis ovládání aplikace včetně obrázků.

1 UMĚLÉ NEURONOVÉ SÍTĚ

Umělá neuronová síť napodobuje funkce lidského mozku. Na začátku je síť nenaučená, podobně jako mozek u člověka po narození. Poté následuje proces učení, kterému je věnovaná podkapitola Učení neuronové sítě. Nakonec je síť schopná na základě vstupních signálů vygenerovat výstupní signál. Díky schopnosti se učit je neuronová síť vhodná pro úlohy, kde není předem znám matematický model anebo je příliš složitý. Mezi možná uplatnění patří: předpovídání časových řad, rozpoznávání obrazců či zpracování analogových signálů. [1], [2]

1.1 Neuron

Umělý neuron, nebo přesněji řečeno jeho matematický model, je základním stavebním prvkem umělé neuronové sítě. Obdobně jako biologický neuron má i ten umělý několik vstupů, tělo a jeden výstup. Na každém vstupu se přečte signál a vynásobí se jeho váhou. Takto upravené vstupní hodnoty se pak pomocí agregační funkce (nejčastěji sumou) agregují a přičte se k nim takzvaný práh neuronu, hodnota udávající počáteční stav neuronu. Výsledek se poté omezí pomocí aktivační funkce na předem stanovenou mez, například interval <-1;1>. [1], [2]



Obrázek 1: model umělého neuronu [2]

1.2 Topologie sítě

Propojením neuronů vznikne neuronová síť. Počet neuronů a jejich uspořádání vytváří topologii sítě. Spolu s váhami a aktivačními funkcemi pak tvoří komplexní neuronovou síť. Ve většině případů se neurony seskupují do vrstev a propojují se jen s neurony následující vrstvy, tvoří tak takzvanou dopřednou síť. Součástí takové topologie je vrstva vstupní, výstupní a případně jedna

nebo více vrstev skrytých mezi nimi. Mohou existovat i topologie, ve kterých jsou neurony propojené mezi sebou například v mřížce. [1], [2]

1.3 Učení neuronové sítě

Klíčovou vlastností neuronových sítí je jejich schopnost učit se. Učení je proces, během kterého se upravují určité parametry sítě na základě jejího okolí. Nejčastějším parametrem jsou váhy jednotlivých spojení neuronů, ale můžeme se také setkat s procesy, které upravují aktivační funkce nebo dokonce do jisté míry i samotnou topologii sítě.

1.3.1 Hebbův zákon učení

Učení sítě je tedy definováno algoritmem, který stanový změny v síti na základě vstupních signálů (popřípadě i odezvy sítě na dané signály). Nejstarší předlohou pro tyto algoritmy je Hebbův zákon učení, který je pojmenovaný po psychologovi a neurobiologovi Donaldu Hebbovi. Ten roku 1949 vydal knihu [3], ve které uvedl, že spojení dvou neuronů sílí, pokud jsou oba neurony ve stejný čas aktivní. Na základě této myšlenky probíhá učení například u Hopfieldovy sítě, které je věnovaná 3. kapitola.

1.3.2 Učení s učitelem

Učící algoritmy bychom mohli rozdělit do dvou skupin, podle paradigmat, jež využívají. Prvním je učení s učitelem, během kterého síť porovnává své odezvy na vstupní signály s odezvou učitele. V praxi to vypadá tak, že učitel poskytne síti trénovací množinu dat, ve které jsou vstupní signály a k nim ideální odezvy. Síť stanoví pro vstupní signál svou odezvu a poté se upraví na základě rozdílu mezi touto odezvou a odezvou ideální. Síť má tedy po ukončení učení být schopna napodobit reakce učitele. Jednoduchým příkladem učitele by mohla být kalkulačka, na které spočítáme určité množství součtů dvou náhodných čísel od -5 do 5. Tyto náhodná čísla a jejich součty poslouží jako trénovací množina pro síť, která by poté měla být schopna sčítat libovolná čísla v předem zmíněném rozmezí. Přesnost sítě závisí jak na zvolené topologii, tak na množství dat v trénovací množině. [2]

1.3.3 Učení bez učitele

Jiný přístup pak zavádí učení bez učitele, takzvaná samoorganizace. Síť v tomto případě předem nemá žádné informace o ideálních odezvách pro vstupní signály. Jako trénovací množinu tedy dostane pouze vstupní signály. Síť se poté pokusí ve vstupních signálech nalézt podobnosti a na jejich základě je rozčlenit do skupin. Po dokončení učení bude odezvou sítě na vstupní signál skupina, ke které má signál nejblíže. Příkladem může být síť, která jako trénovací množinu obdržela fotografie lidí a aut. Následně bude schopna o libovolné fotce říct, zda se jedná o člověka nebo o automobil. Přesnost sítě závisí na mnoha faktorech, jako je topologie, množství trénovací množiny či reprezentace vstupních dat. [1], [2]

2 HOPFIELDOVA NEURONOVÁ SÍŤ

John Joseph Hopfield během své činnosti na Kalifornským institutu Technologie (oddělení chemie a biologie) publikoval roku 1982 práci zabývající se vlastnostmi paměti biologických organismů a možností aplikovat tyto vlastnosti při tvorbě obsahem adresovatelné (asociativní) paměti v oblasti výpočetních technologií. Výsledkem jeho práce byl matematický popis umělé neuronové sítě dnes známé jako Hopfieldova neuronová síť. [5]

2.1 Učení

Hopfieldova síť se učí pomocí Hebbova zákona o učení. Reprezentace paměti je formou matice o rozměrech R×R, kde R je počet neuronů v síti. Při učení se do paměti přičítají vztahy mezi jednotlivými neurony. Vztah mezi dvěma neurony je definován bipolární hodnotou. Pokud oba neurony jsou pro daný vstup (který musí být definován R bipolárními hodnotami) k zapamatování aktivní či neaktivní, na dané místo v matici se přičte jednička. V případě, že dané neurony nemají stejný stav (aktivní je právě jeden z nich), dané místo v matici se o jedničku sníží. Matice udržuje váhy všech možných dvojic neuronů dvakrát. Je to způsobeno tím, že při učení narazíme na dvojici neuronu *a, b* ale i *b, a*, přestože jsou obě tyto hodnoty stejné. Díky tomuto faktu se dá říct, že matice vah je symetrická. Vztah pro výpočet jednotlivých vah sítě:

$$w_{ij} = \begin{cases} \sum_{p=1}^{N} t_i(p)t_j(p), & \text{pro } i \neq j, \\ 0, & \text{pro } i = j, \end{cases}$$

kde

w_{ij} – hodnota váhy mezi neurony i a j;

i – pořadové číslo neuronu $i = 1 \ldots R$;

j – pořadové číslo neuronu $j = 1 \ldots R$;

p – pořadové číslo vzoru k zapamatování $p = 1 \ldots N$;

N – celkový počet vzorů k zapamatování;

$t_i(p)$ – i-tý prvek p-tého vzoru k zapamatování;

$t_j(p)$ – j-tý prvek p-tého vzoru k zapamatování. [2]

Do již naučené sítě lze přidat nové vzory stejným způsobem, jakým byla naučená na začátku. Stačí do matice vah přičíst vztahy mezi neurony pro nový vzor. Stejným principem lze také

zajistit zapomenutí vzorce. Stačí vztahy mezi neurony daného vzorce z matice odečíst. Tudíž pokud by dva neurony byly současně aktivní anebo neaktivní, dané místo v matici by se o jedničku snížilo. V případě rozdílných stavů neuronů by se pak jednička přičetla. Vzorce pro přidání a zapomenutí obrazce:

$$w_{ij} = w_{ij} + \Delta w_{ij}, \quad \text{pro zapamatování obrazce,}$$
$$w_{ij} = w_{ij} - \Delta w_{ij}, \quad \text{pro zapomenutí obrazce,}$$

$$\Delta w_{ij} = \begin{cases} t_i t_j, & \text{pro } i \neq j, \\ 0, & \text{pro } i = j, \end{cases}$$

kde

t_i – i-tý prvek vzoru pro zapamatování či zapomenutí;

t_j – j-tý prvek vzoru pro zapamatování či zapomenutí. [2]

2.2 Stanovení odezvy sítě

Hopfieldova síť je takzvaná auto asociativní paměť. Výsledná odezva tedy vzniká postupným vybavováním. Síť postupně mění vstupní vzor podle matice vah na vzor, k němuž má nejmenší vzdálenost. Vstupem je tedy v první iteraci vstupní vzor a v dalších iteracích pak výsledek iterace předchozí. Výslednou odezvou je pak vzor, který je již stabilní (dalšími iteracemi se nezmění). Vztah pro stanovení odezvy:

$$y_i(k) = \varphi\left(y_{a,i}(k)\right),$$

$$y_{a,j}(k) = \begin{cases} \sum_{i=1}^{R} w_{ij} x_i, & \text{pro } k = 0, \\ \sum_{i=1}^{R} w_{ij} y_i(k-1), & \text{pro } k > 0, \end{cases}$$

kde

φ – aktivační funkce perceptronu;

i – pořadové číslo neuronu $i = 1 \ldots R$;

j – pořadové číslo neuronu $j = 1 \ldots R$;

x_i – i-tý prvek vstupního vzorce;

$y_{a,j}$ – vstupní potenciál perceptronu. [2]

Výsledný vzor ovšem nemusí odpovídat předpokládanému vzoru. Může nastat, že síť "mylně" vybaví jiný vzor z paměti. Dokonce mohou nastat situace, kdy výsledným vzorem nebude žádný vzor z paměti, takzvaný fantom. Anebo výstupy iterací začnou oscilovat, jinými slovy výstupy nikdy nedosáhnout stabilního stavu, protože budou přeskakovat mezi dvěma různými stavy.

Existují dva přístupy stanovení odezvy jednotlivých iterací, synchronní a asynchronní. Synchronní vybavování stanoví odezvu všech neuronů paralelně, tedy v jedné iteraci se přepočítají všechny neurony. V tomto případě lze využít maticového výpočtu.

$$\mathbf{y_a}(k) = \begin{cases} \mathbf{W^T x}, & \text{pro } k = 0, \\ \mathbf{W^T y}(k-1) & \text{pro } k > 0, \end{cases}$$

kde

W – matice vah;

x – sloupcový vektor reprezentující vstupní vzorec;

$\mathbf{y_a}$ – sloupcový vektor reprezentující výstup sítě.

V případě asynchronního vybavování se v jedné iteraci přepočítá pouze jeden neuron. Neurony se postupně všechny vystřídají v náhodném pořadí. Poté se otestuje, zda vzor nedosáhl stabilního stavu (nezměnil se od předchozí kontroly) a pokud nebude vyhodnocen jako stabilní (nebo že osciluje), tak se znova postupně přepočítají všechny neurony v novém náhodném pořadí. Asynchronní přístup věrněji simuluje biologickou neuronovou síť, jelikož ani v mozku se neaktivují všechny neurony ve stejný moment. [2]

2.3 Omezení sítě

Jak již bylo zmíněno, síť nezaručuje vždy správný výsledek. Přesnost sítě se odvíjí od počtu zapamatovaných vzorů a také záleží na rozdílnosti vzorů. Vzory, které mají větší část hodnot společnou, se od sebe hůře odlišují. Teoretická kapacita sítě byla podle Hopfieldovy práce (1982) přibližně 0,15N. Na téma kapacity Hopfieldovy sítě vznikla další řada studií [4], na základě kterých se doporučená kapacita sítě nejčastěji udává 0,138N. V praxi je pak tato hodnota ještě menší vzhledem k povaze vzorů. Málokdy si totiž chceme zapamatovat čistě náhodné vzory či dokonce vzory sestavené čistě za účelem dosáhnutí maximální možné kapacity sítě. [2], [5]

2.4 Aplikace využívající Hopfieldovu neuronovou síť

Hopfieldova neuronová síť se dá díky svým vlastnostem využít při řešení mnoha různých problémů. Kromě rozpoznávání nakreslených obrazců, kterým se věnuji v této práci, vyšlo také mnoho prací zaměřených na jiná zajímavá využití této sítě. Příkladem může být aplikace pro rozpoznávání obličeje [6], využití sítě pro autentizaci hesel [7] či řešení problému výběru ideální trasy na základě několika aspektů [8].

3 EXISTUJÍCÍ SOFTWARE PRO ROZPOZNÁVÁNÍ OBRAZCŮ

V následujících podkapitolách bude zmíněno několik různých aplikací věnujících se rozpoznávání obrazců. Většina z nich je řešena jako webová služba, což je ideální řešení, neboť umělá inteligence není dokonalá a její podstatná výhoda je, že se může v průběhu času učit a zlepšovat.

3.1 Vision AI

Jedná se o natrénovanou umělou inteligenci určenou pro rozpoznávání informací z obrázků. Nástroj je schopen detekovat jednotlivé objekty a texty v obrázku. Také dokáže přidělit obrázku klíčová slova a do jisté míry určit, zda obrázek obsahuje nějaký nevhodný obsah. Celé řešení se nachází na cloudu a přístup je řešen přes REST a RPC API. Do jednoho tisíce požadavků za měsíc je služba zdarma. [9]

3.2 Reverse image search

Druhým nástrojem od Googlu demonstrujícím rozpoznávání obrazců je samotný vyhledávač, který dokáže podle nahraného obrázku nalézt podobné obrázky a stránky, na kterých se nacházejí.

3.3 Open CV

Open CV není aplikace jako taková, ale jedná se o open-source knihovnu, kterou využívá velké množství jiných aplikací. Knihovna obsahuje přes 2,500 optimalizovaných algoritmů pro strojové učení a rozpoznávání obrazců. Knihovna se pak dá využít například k rozpoznávání tváří, identifikování objektů, hledání podobných obrázků z databáze anebo práci s rozšířenou realitou. [10]

3.4 Orpix computer Vision

Placená webová služba sloužící k rozpoznání aut v reálném čase. Mezi možná využití patří prodej aut, cílená reklama na elektronických billboardech, ale také pomoc při boji s kriminalitou. Služba rozpozná vozidla na snímku a přiřadí jim několik atributů, jako jsou například model, rok výroby a spz značka. Služba je optimalizovaná pro USA, protože její databáze obsahuje téměř všechny tamní vyráběné a dovážené modely aut od roku 2000 do současnosti. [11]

4 VÝVOJ APLIKACÍ PRO ANDROID OS

Mobilních zařízení s Android OS je v dnešní době na trhu velmi mnoho. Jednou z podstatných výhod je poměrně snadný vývoj aplikací pro tato zařízení díky dostupné dokumentaci jejich API a také díky rozsáhlé komunitě. Nevýhodou pak může být velké množství různorodých zařízení a jejich hardware.

4.1 Programovací jazyky

Aplikace pro Android OS se dá vyvíjet hned v mnoha různých jazycích. Oficiálně podporovanými jazyky jsou Java a Kotlin. Výhodou těchto jazyků je, že dokumentace android API obsahuje příklady kódu pro oba tyto jazyky.

Další populární možností je C#, který je podporován užitečnými nástroji, jako je Unity či Xamarin. Díky Unity je velmi dobrou volbou například pro vývoj her. Mezi další možné jazyky pak patří C/C++ a BASIC. Také existuje alternativa v podobě interaktivních webových stránek zabalených do aplikace. [12]

4.2 Vývojové prostředí

Volba vývojového prostředí je úzce spjata s volbou programovacího jazyka. V případě Javy a Kotlinu je ideální volbou Android Studio, které bylo navrženo právě pro účely vývoje aplikací pro Android OS. Hlavní výhodou je integrovaný simulátor mobilních zařízení a možnost spouštět aplikaci přímo v připojeném mobilním zařízení přes ADB (Android Debug Bridge).

Pro C/C++ a C# je pak doporučovanou volbou Visual Studio, které vzniklo právě pro dané jazyky. Existují i jiná vývojová prostředí se kterými lze pracovat, jmenovitě pak třeba Eclipse, který lze poměrně snadno modifikovat, takže vznikla spousta pluginů rozšiřující možnosti vývoje v tomto prostředí o mnohé jazyky.

4.3 Nástroje pro implementaci umělých neuronových sítí pro mobilní platformy

Mobilní zařízení obecně disponují nižším výkonem než stolní počítače či velké servery. Když se vezme v úvahu výpočetní náročnost neuronových sítí a obecně strojového učení, nebyla by mobilní zařízení vhodnou cílovou platformou. Jednodušší neuronové sítě ještě lze počítat

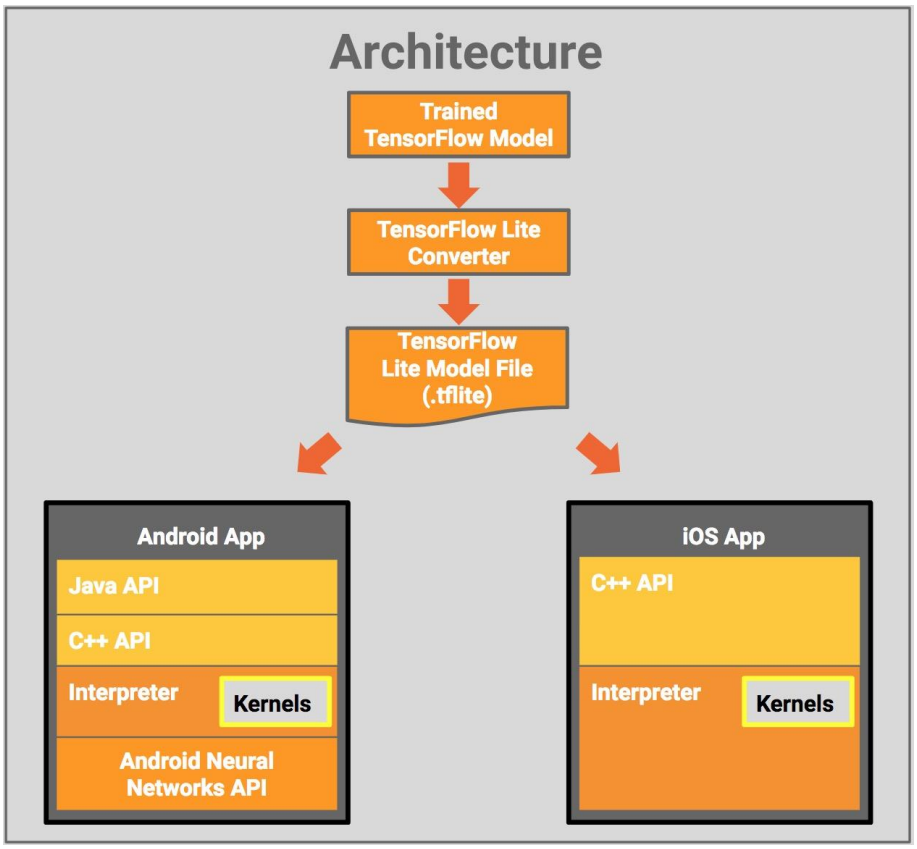
na mobilních CPU, ale co se týče náročnějších výpočtů, tak bylo v minulosti běžnou praxí využití modelu klient-server. Výhodou bylo, že náročné výpočty počítal vzdálený server, na druhou stranu nevýhodou byla potřeba internetového připojení pro komunikaci se serverem.

V posledních dvou letech se začalo pracovat na rozdílném přístupu. Hlavní výrobci mobilních čipů na trhu začali vyrábět čipy co obsahují kromě procesorů CPU a GPU také procesor pro strojové učení NPU, který je mimo jiné zaměřený také na akceleraci výpočtů neuronových sítí. Některé společnosti také začaly vyvíjet frameworky, které usnadňují vývojářům aplikací využití možností hardware, ať už NPU nebo alespoň převedení výpočtů na GPU. [13]

4.3.1 Frameworky pro strojové učení

Core ML je framework využívaný napříč produkty od firmy Apple. Je využit například u digitální asistentky Siri. Kromě toho také Apple nabízí specifické frameworky, jako je například Vision, který je určený pro rozpoznávání obrazu, Natural Language určený pro rozpoznávání jazykových vlastností textu, anebo Create ML, který zjednodušuje tvorbu modelů strojového učení natolik, že to zvládne i člověk bez znalostí z tohoto oboru. [14]

TensorFlow Lite je další z řady frameworků určených pro strojové učení na mobilních zařízení. Jedná se o mobilní verzi většího frameworku TensorFlow, ve kterém lze sestavovat a trénovat různé modely. Tyto modely pak lze převést do Lite verze, kterou lze spustit v aplikaci za pomocí knihovny C++ API, která využívá takzvaného Interpreta, který efektivně využívá hardware zařízení pro výpočty. Takto lze spustit natrénovaný model na zařízeních s operačním systémem Android nebo iOS. V případě androidu lze také využít knihovnu Java API, která obaluje knihovnu C++ API. [15]



Obrázek 2: TensorFlow architektura [15]

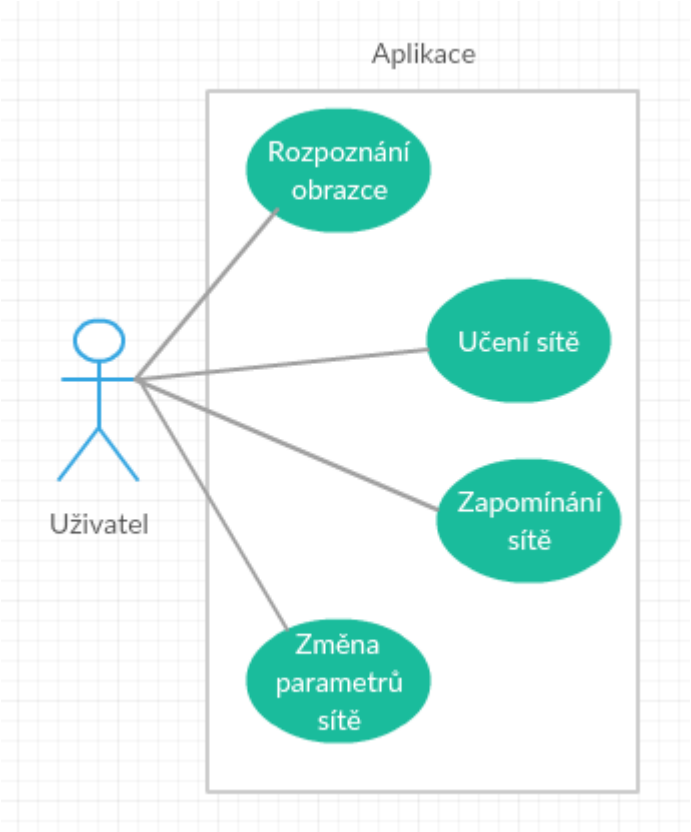
S příchodem Android verze 8.1 (API 27) přibyla knihovna Android Neural Networks API (NNAPI), která poskytuje nízkoúrovňovou vrstvu funkcí pro jiné frameworky (např. TensorFlow Lite). Hlavním cílem NNAPI je zpřístupnit co nejefektivněji hardwarovou akceleraci ostatním frameworkům, které sestavují a využívají modely strojového učení. [16]

Jedna z největších firem vyrábějící mobilní procesory Qualcomm vyvinula vlastní framework (Snapdragon Neural Processing Engine SDK), díky kterému mohou vývojáři naplno využít výkon jejich procesorů. Framework je možné využít v Javě i C++ a dokonce je možné zkonvertovat modely z některých jiných frameworků do formátu vyhovujícím tomuto frameworku. [17]

5 NÁVRH APLIKACE

Prvním krokem při vývoji aplikace je její návrh. Nejprve se provede analýza případů užití, to jsou scénáře, které očekáváme, že uživatel aplikace bude chtít provést. Následně proběhne sestavení diagramu tříd, který vystihne jednotlivé části aplikace, jejich účel a provázanost.

5.1 Případy užití



Obrázek 3: Diagram případů užití

Rozpoznání obrazce – uživatel nakreslí za pomocí dotykového displeje na plátno aplikace obrazec, který následným kliknutím na určené tlačítko podrobí zkoumání neuronové sítě. Celý průběh zkoumání je animovaný v závislosti na nastavení sítě. Po dokončení, pokud byl obrazec úspěšně nalezen, zobrazí originální obrazec uložený v paměti. Pokud neuronová síť výsledek nenalezne (dojde k oscilaci nebo nalezne fantoma), tak ukončí zkoumání a na displeji zůstane vyobrazen poslední výstup sítě.

Učení sítě – uživatel pomocí určeného tlačítka zobrazí možnost importu obrazců. Poté zvolí libovolné množství obrázků z disku a potvrdí jejich import. Aplikace poté zkopíruje obrázky

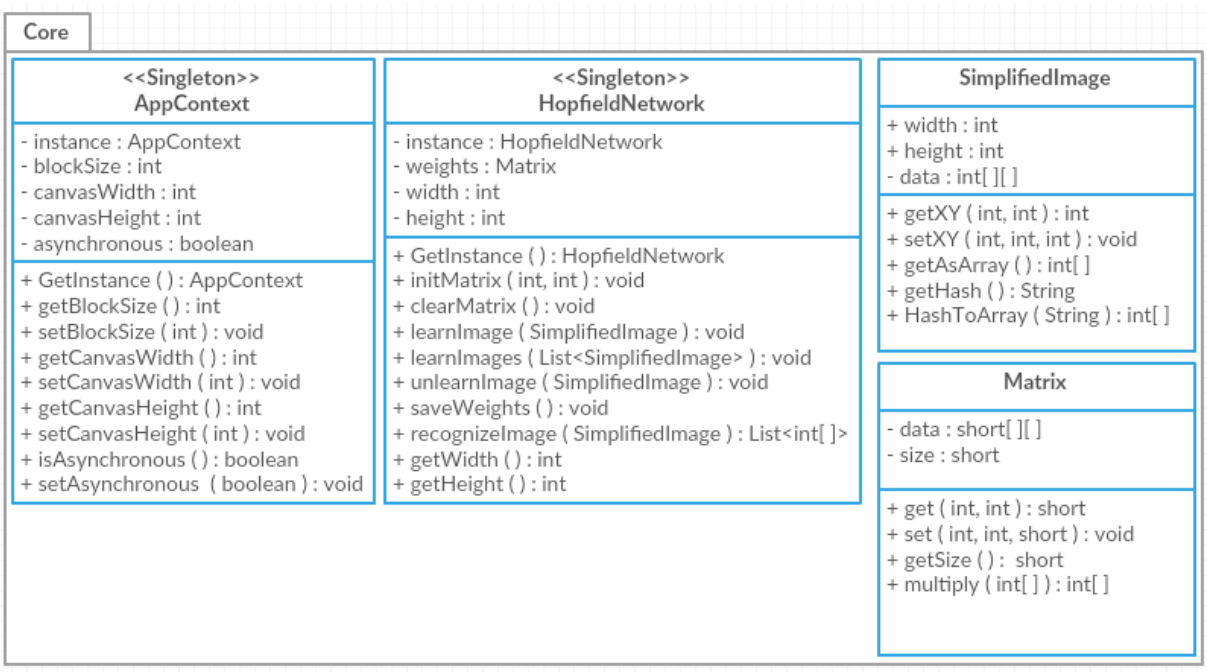
do složky aplikace a tam jim změní velikost, aby odpovídala rozměrům plátna aplikace na daném zařízení. Aplikace poté provede doučení neuronové sítě o dané obrazce. Nakonec uloží neuronovou síť do složky aplikace, aby při dalším spuštění nebylo třeba provádět proces učení znova.

Zapomínání sítě – uživatel určeným tlačítkem zobrazí všechny obrazce uložené v paměti. Poté zvolí libovolné množství obrazců a potvrdí jejich zapomenutí. Aplikace zvolené obrazce odstraní ze své složky a provede jejich odnaučení z neuronové sítě. Nakonec uloží neuronovou síť.

Změna parametrů sítě – uživatel zobrazí určeným tlačítkem možná nastavení aplikace, kde podle potřeby upraví nastavení sítě. Klíčová nastavení sítě jsou počet neuronů a způsob vybavování sítě (synchronní/asynchronní). Následným potvrzením proběhne změna těchto parametrů a v případě změny počtu neuronů také přeučení celé neuronové sítě a její uložení do složky aplikace.

5.2 Diagram tříd

Aplikace je členěna do tří balíčků. Balíček *core* obsahuje třídy, které obstarávají klíčovou logiku okolo neuronové sítě. Balíček *graphics* obsahuje třídy spjaté s vykreslováním a zprávou obrázků. Poslední balíček *activities* obsahuje třídy provázané s jednotlivými okny (aktivitami) aplikace.



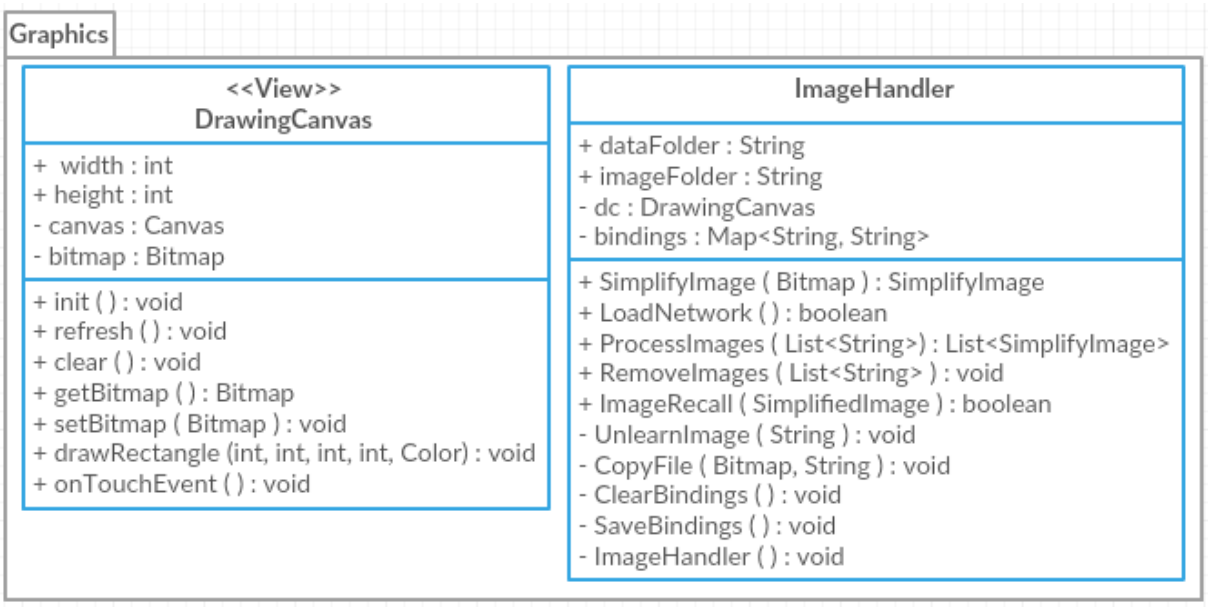
Obrázek 4: diagram tříd – balíček core

Třída *AppContext* slouží k uchovávání potřebných informací napříč aplikací, jako je velikost kreslícího plátna či jednotlivá nastavení. Vzhledem k faktu, že stačí pouze jedna instance této třídy na aplikaci, je vhodné využít návrhový vzor Singleton.

Třída *HopfieldNetwork* zapouzdřuje logiku neuronové sítě, jmenovitě učící algoritmy či algoritmy pro stanovení odezvy. Také udržuje matici vah sítě. Tyto váhy jsou v paměti uloženy v instanci třídy *Matrix*. Stejně jako u třídy *AppContext* je i u třídy *HopfieldNetwork* žádoucí, aby její instance byla v aplikaci pouze jednou, tedy znovu se jedná o Singleton.

Třída *Matrix* zapouzdřuje dvourozměrnou matici, ve které budou uloženy váhy neuronové sítě. Také obsahuje základní maticovou operaci násobení vektorem, která poslouží při stanovení odezvy sítě.

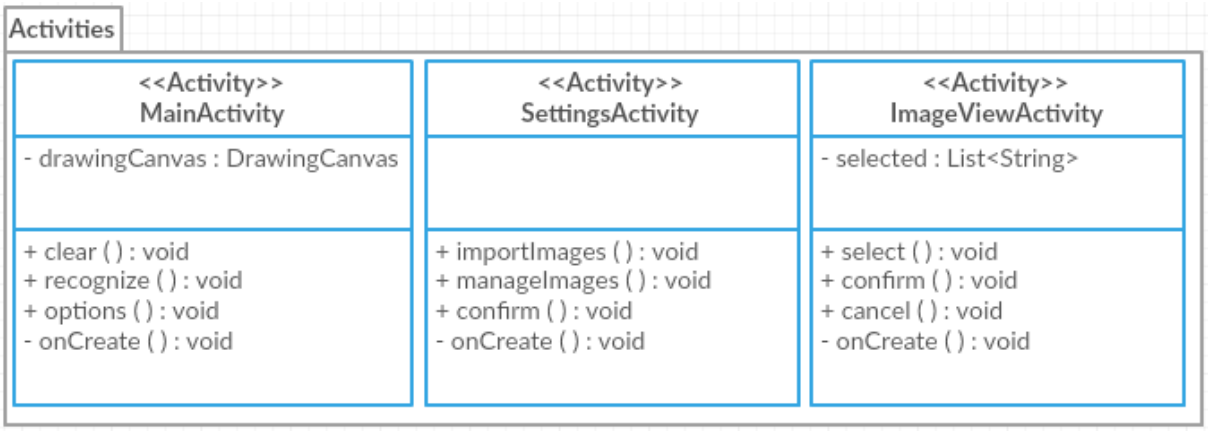
Třída *SimplifiedImage* slouží k optimalizaci uchovávání obrazce v paměti. Obrazec je uchováván v matici bipolárních hodnot o rozlišení odpovídající počtu neuronů v neuronové síti. Třída také umožňuje reprezentovat obrazec ve tvaru jednorozměrného pole či textového řetězce.



Obrázek 5: diagram tříd – balíček graphics

Třída *DrawingCanvas* rozšiřuje grafickou komponentu *View* o potřebné funkčnosti, aby posloužila jako kreslící plátno pro aplikaci. Zpracovává tedy vstup z dotykového displeje a umožňuje překreslit plátno podle potřeby v rámci aplikace.

Třída *ImageHandler* je statická třída zapouzdřující veškerou práci se soubory obrázků. Také udržuje hash jednotlivých naučených obrazců pro případ úspěšného detekování obrazce pomocí neuronové sítě. Třída tedy volá metody tříd *HopfieldNetwork* a *DrawingCanvas*.



Obrázek 6: diagram tříd – balíček activities

Třída *MainActivity* reprezentuje hlavní okno aplikace, ve které se nachází kreslící plátno a tlačítka pro potřebné funkce. Také je zde tlačítko pro přechod do okna nastavení.

Třída *SettingsActivity* reprezentuje okno nastavení, kde lze nastavit jednotlivé parametry sítě a také obsahuje tlačítko pro vyvolání dialogu pro import obrázků a tlačítko pro zobrazení okna s již naučenými obrazci.

Třída *ImageViewActivity* reprezentuje okno pro správu obrazců. Jednak zobrazuje všechny obrazce, které jsou naučeny v neuronové síti, a také umožňuje zvolit obrazce pro zapomenutí.

6 IMPLEMENTACE APLIKACE

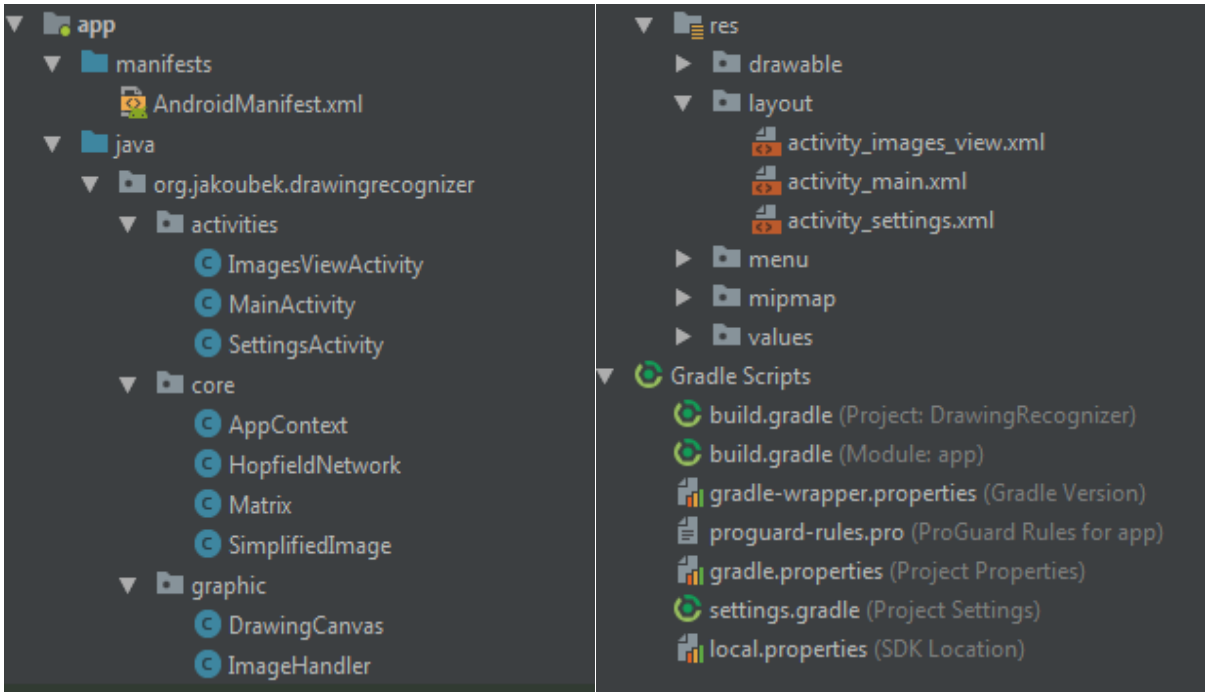
Po dokončení analýzy a sestavení návrhu je možné začít se samotným programováním. Nejprve je třeba zvolit programovací jazyk a vývojové prostředí. Poté se začnou implementovat jednotlivé třídy z návrhu.

6.1 Použité technologie

K vývoji aplikace byly použity volně dostupné nástroje a technologie. Aplikace byla vyvíjena v programovacím jazyce Java a bylo použito vývojové prostředí Android Studio. S výjimkou Android API nebyly použity žádné jiné externí knihovny.

Aplikace byla během vývoje testovaná na mobilním zařízení Huawei CUN-L21 s procesorem quad core 1,3 GHz, pamětí RAM 1 GB, rozlišením displeje 720 × 1280 a operačním systémem Android ve verzi 5.1, což odpovídá Android API 22.

6.2 Struktura projektu



Obrázek 7: Struktura projektu

Projekt se skládá z několika různých souborů. Jsou zde gradle skripty, které obsahují informace, jako například závislosti nebo minimální a cílená verze API, které jsou potřebné k úspěšnému sestavení projektu do instalačního balíčku s příponou apk.

Dalším souborem obsahujícím informace o aplikaci je takzvaný manifest. Jedná se o XML soubor, který obsahuje název balíčku aplikace, popis jednotlivých komponent aplikace (například aktivit), potřebná zvláštní oprávnění a případné požadavky na vyžadovaný hardware a software.

Ve složce src se nacházejí zdrojové kódy aplikace, které byly podle návrhu aplikace rozčleněny do tří balíčků – activities, core a graphic. Poslední klíčovou složkou projektu je res/layout, kde se nacházejí XML soubory popisující rozložení jednotlivých grafických prvků pro každou aktivitu.

6.3 Popis jednotlivých tříd

Projekt obsahuje celkem devět souborů se zdrojovými kódy. Následuje popis jednotlivých tříd včetně ukázek kódu klíčových metod.

6.3.1 Core.AppContext

Singleton třída, která obstarává uchování a přístup k jednotlivým atributům, které jsou potřeba po dobu běhu aplikace v různých dalších třídách. Oproti návrhu byly přidány atributy pro velikost displeje, které jsou využity u realizace responzivního designu. Dále byl přidán atribut pro tloušťku kreslícího „pera“ a atribut pro hladinu šumu, kterým lze poškodit obrazec pro účely testování. Nakonec byly také přidány, kromě getrů a setrů na zmíněné atributy, dvě pomocné metody. První je určená pro výpočet počtu neuronů z velikosti bloku, který je využit v aktivitě nastavení. A druhá, která změní velikost bloku podle daného počtu neuronů, se využívá při načítání sítě z disku.

```
public int calculateNumberOfNeurons(int block){
    return (canvasWidth/block)*(canvasHeight/block);
}
public void reverseCalculations(int numberOfNeurons){
    blockSize=(int)Math.sqrt(canvasHeight*canvasWidth/numberOfNeurons);
}
```

6.3.2 Core.HopfieldNetwork

Singleton třída, která je srdcem celé aplikace. Je zodpovědná za správu a funkčnost neuronové sítě, kterou si udržuje v podobě vah mezi jednotlivými neurony v instanci třídy *core.Matrix*.

První klíčovou metodou této třídy je přiučení neuronové sítě o nový obrazec, který je již v podobě *core.SimplifiedImage*.

```
public void learnImage(SimplifiedImage simplifiedImage){
    long time = System.currentTimeMillis();
    int[] array = simplifiedImage.getAsArray();
    for (int i=0; i<array.length; i++){
        for (int j=0; j<array.length; j++){
            if (i==j){
                weights.set(i,j,0);
            }else{
                weights.increment(i,j,(array[i]*array[j]));
            }
        }
    }
    Log.println(Log.DEBUG,"APP", "Image learned in: "
            +(System.currentTimeMillis()-time)/1000.0+" sec");
}
```

Další klíčovou metodou je zapomenutí obrazce, která je velmi podobná metodě na učení. Rozdíl je ve vstupním parametru, kde metoda očekává obrazec již v podobě jednorozměrného pole hodnot. Samotný algoritmus funguje stejně až na to, že vztah dvou neuronů od vah odečítá.

```
…
  for (int i=0; i<array.length; i++){
    for (int j=0; j<array.length; j++){
        if (i==j){
            weights.set(i,j,0);
        }else{
            weights.increment(i,j,(-(array[i]*array[j])));
        }
    }
}
```

Mezi další důležité metody patří ukládání vah do souboru (viz kapitola 6.4.3) a samozřejmě rozpoznání obrazce neboli stanovení odezvy sítě, kterou popisuje diagram v příloze A. O stanovení odezvy se stará vnitřní třída *AsyncRecognizer*.

6.3.3 Core.HopfieldNetwork.AsyncRecognizer

Jedná se o vnitřní privátní třídu určenou pro stanovení odezvy sítě. Třída rozšiřuje třídu z Android API *android.os.AsyncTask*, která umožňuje spuštění výpočtů na novém vlákně. Díky

tomu lze na novém vlákně počítat odezvu sítě, zatímco na původním vlákně se průběžně vykresluje aktuální odezva sítě, čímž se vyřeší animování průběhu. Pro stanovení odezvy se ve třídě nacházejí dvě metody, jedna pro synchronní a druhá pro asynchronní přístup. Podle nastavení v *core.AppContext* se volí, která z metod se bude volat.

```
        private ArrayList<int[]> syncMode(SimplifiedImage simplifiedImage){
            long time = System.currentTimeMillis();
            ArrayList<int[]> progress = new ArrayList<>();
            int iterations = 0;
            boolean match = false;
            int[] it;
            it = simplifiedImage.getAsArray();
            progress.add(copyArray(it));
            publishProgress(progress.get(0));
            while(!match) {
                it = weights.multiply(copyArray(it));
                if (iterations>=0){
                    if (areSame(it,progress.get(iterations))){
                        match=true;
                    }else if(iterations>1){
                        if(areSame(it,progress.get(iterations-1))){
                            match=true;
                            Log.println(Log.DEBUG,"APP", "Oscillation");
                        }
                    }
                }
                progress.add(copyArray(it));
                publishProgress(progress.get(progress.size()-1));
                iterations++;
                Log.println(Log.DEBUG,"APP", "iteration "+iterations
+" complete.");
            }
            Log.println(Log.DEBUG,"APP", "Image found in: "
+(System.currentTimeMillis()-time)/1000.0+" sec");
            Log.println(Log.DEBUG,"APP", "Number of iterations: "
+iterations);
            return progress;
```

Synchronní přístup aktualizuje všechny neurony během iterace současně. Po dokončení jedné iterace se otestují ukončovací podmínky a odešle se aktuální odezva hlavnímu vláknu k vykreslení.

Oproti tomu asynchronní přístup během iterace aktualizuje jednotlivé neurony postupně v předem daném náhodném pořadí, které se pro každou iteraci náhodně změní. Pokud se po aktualizaci neuronu změní odezva sítě, tak se odešle hlavnímu vláknu k vykreslení. Jelikož tento přístup velmi rychle aktualizoval svou odezvu a hlavní vlákno ji nestíhalo vykreslit, bylo přidáno krátké pozdržení výpočtu po každém odeslání odezvy k vykreslení, konkrétně 2 ms. Stejně jako u synchronního přístupu se po dokončení jedné iterace otestují ukončovací podmínky.

```
private ArrayList<int[]> asyncMode(SimplifiedImage simplifiedImage){
    long time = System.currentTimeMillis();
    ArrayList<int[]> progress = new ArrayList<>();
    ArrayList<Integer> indexes = new ArrayList<>();
    int iterations = 0;
    boolean match = false;
    int[] it;
    it = simplifiedImage.getAsArray();
    progress.add(copyArray(it));
    publishProgress(progress.get(0));
    for (int i = 0; i<it.length; i++){
        indexes.add(i);
    }
    while(!match) {
        Collections.shuffle(indexes);
        for (Integer i : indexes) {
            it = weights.multiply(copyArray(it), i);
            if (weights.asyncChangeFlag){
                publishProgress(copyArray(it));
                try {
                    Thread.sleep(2);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
        if (iterations>=0){
            if (areSame(it,progress.get(iterations))){
                match=true;
                Log.println(Log.DEBUG,"APP", "Vysledek nalezen");
            }else if(iterations>1){
                if(areSame(it,progress.get(iterations-1))){
```

```
                            match=true;
                            Log.println(Log.DEBUG,"APP", "Oscilace
detekována");
                        }
                    }
                }
                progress.add(copyArray(it));
                publishProgress(progress.get(progress.size()-1));
                iterations++;
                Log.println(Log.DEBUG,"APP", "iteration "+iterations
+" complete.");
            }
            Log.println(Log.DEBUG,"APP", "Image found in: "
+(System.currentTimeMillis()-time)/1000.0+" sec");
            Log.println(Log.DEBUG,"APP", "Number of iterations: "
+iterations);
            return progress;
        }
```

6.3.4 Core.Matrix

Jednoduchá třída, která zapouzdřuje dvourozměrné pole hodnot reprezentující matici vah neuronové sítě. Oproti návrhu doznala třída jedné podstatné změny. Metoda pro vynásobení matice s vektorem byla přetížena o metodu potřebnou pro asynchronní stanovení odezvy, která přijímá navíc číselný parametr označující hodnotu vektoru, která se má přepočítat.

```
    public int[] multiply(int[] input){
        int[] result = new int[input.length];
        for (int i = 0; i<size; i++){
            int neuronI = i;
            result[neuronI] = 0;
            for (int neuronJ = 0; neuronJ < size; neuronJ++){
                result[neuronI]+=data[neuronI][neuronJ]*input[neuronJ];
            }
            result[neuronI] = (result[neuronI]>0)?1:-1;
        }
        return result;
    }

    public int[] multiply(int[] input, int index){
            int buff = input[index];
            for (int neuronJ = 0; neuronJ < size; neuronJ++){
```

```
                input[index]+=data[index][neuronJ]*input[neuronJ];
            }
        input[index] = (input[index]>0)?1:-1;
            asyncChangeFlag = buff != input[index];
        return input;
    }
```

6.3.5 Core.SimplifiedImage

Třída zapouzdřuje dvourozměrné pole hodnot reprezentující obrazec v paměti aplikace. Obsahuje metody pro převod obrazce na jednorozměrné pole hodnot či textový řetězec (hash). Také obsahuje statickou metodu pro přímý převod zmíněné hash na jednorozměrné pole číselných hodnot.

6.3.6 Graphic.DrawingCanvas

Třída rozšiřuje *android.view.View* o možnost zpracovat vstup z dotykového displeje. Pro tento účel byla přepsána metoda *onTouchEvent* a byly vytvořeny potřebné pomocné metody, například metoda pro vykreslení čtverce, která se využívá u vykreslení zjednodušeného obrazce či při přidání šumu.

```
public void noise(){
        int block = appContext.getBlockSize();
        for (int x = 0; x<myBitmap.getWidth(); x+=block){
            for (int y = 0; y<myBitmap.getHeight(); y+=block){
                if (Math.random()<appContext.getNoise()){
                    if (Math.random()<0.5){
                       drawRectangle(x,y,x+block, y+block, Color.BLACK);
                    }else{
                        drawRectangle(x,y,x+block, y+block, Color.WHITE);
                    }
                }
            }
        }
    }
```

6.3.7 Graphic.ImageHandler

Třída obsahuje výhradně statické metody. Účelem této třídy je veškerá práce s bitmapovými obrázky. To zahrnuje načítání, kopírování a mazání obrázků, převádění bitmapy na zjednodušený obrazec (*SimplifiedImage*) a také udržování spárovaných hash (naučených obrazců) s názvy souborů příslušných obrázků pro účely zobrazení originálního obrázku při úspěšném stanovení odezvy neuronové sítě. Následuje kód metody pro sestavení zjednodušeného obrazce z plátna *DrawingCanvas* (dc).

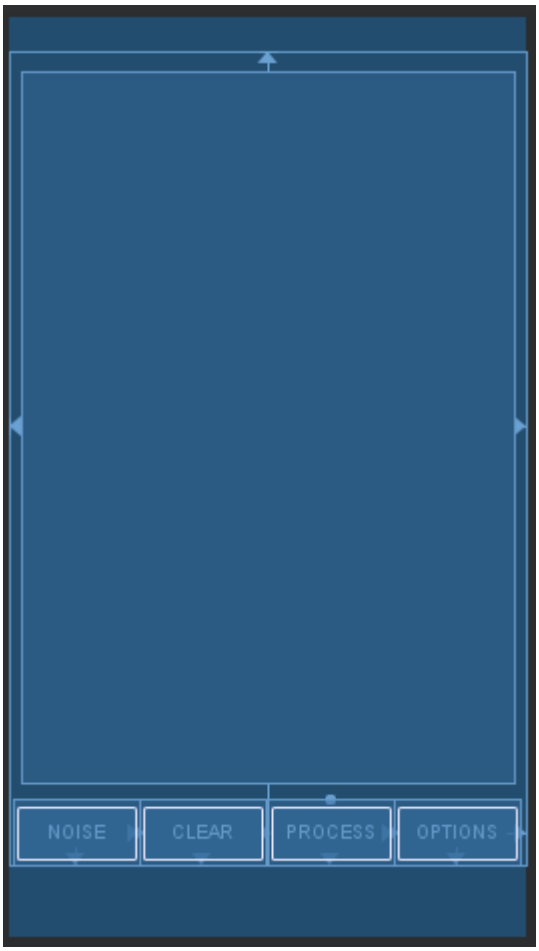
```
    public static SimplifiedImage SimplifyImage() {
        long time = System.currentTimeMillis();
        Bitmap bitmap = dc.getMyBitmap();
        int width = dc.getSimpleX();
        int height = dc.getSimpleY();
        SimplifiedImage result = new SimplifiedImage(width, height);
        int block = appContext.getBlockSize();
        for (int sx = 0; sx < width; sx++) {
            for (int sy = 0; sy < height; sy++) {
                int pxCount = 0;
                for (int x = 0; x < block; x++) {
                    for (int y = 0; y < block; y++) {
                        if (isNonWhite(bitmap, (sx * block + x), (sy *
block + y))) {
                            pxCount++;
                        }
                    }
                }
                if (pxCount > block * block / 2) {
                    result.setXY(sx, sy, 1);
                } else {
                    result.setXY(sx, sy, -1);
                }
            }
        }
        Log.println(Log.DEBUG, "APP", "Image simplified in: " +
(System.currentTimeMillis() - time) / 1000.0 + " sec");
        return result;
    }
```

Navíc je zde metoda pro načtení sítě, která se pokusí načíst ze složky aplikace soubor vah neuronové sítě a soubor, ve kterém se nacházejí názvy souborů naučených obrázků a hash jejich

zjednodušených obrazců. Pokud se tyto soubory nepovede z nějakého důvodu načíst, vybuduje se celá síť nanovo v základním nastavení (velikost bloku zjednodušeného obrazce 25) a provede se učení obrazců, které se nacházejí v dané podsložce aplikace (pokud existuje a není prázdná).

6.3.8 Activites.*

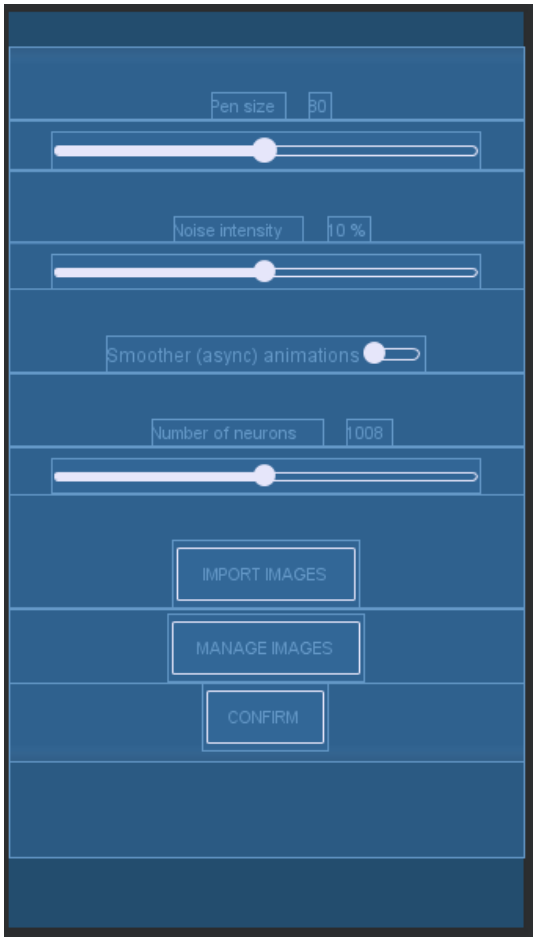
Třídy v balíčku *activities* jsou provázané s layouty pro jednotlivé aktivity a obsahují funkční kódy jednotlivých prvků. Tyto třídy rozšiřují třídu *android.app.Activity* nebo její potomky. Hlavní aktivitu aplikace spravuje třída *activities.MainActivity*. Nachází se zde plátno pro kreslení (*DrawingCanvas*) a čtyři funkční tlačítka – přidání šumu, vyčištění plátna, rozpoznání obrazce a nastavení. Tlačítko nastavení spouští aktivitu nastavení.



Obrázek 8: Grafické rozložení hlavní aktivity

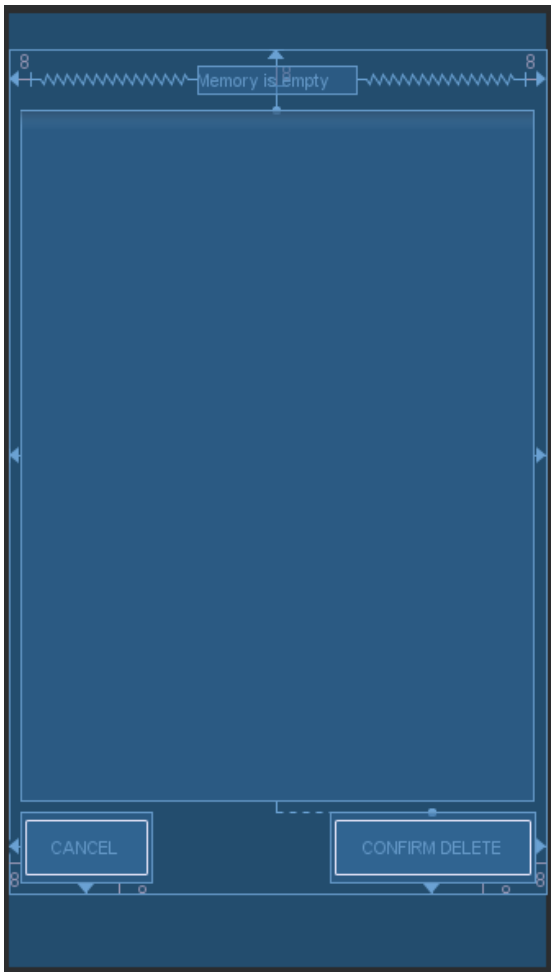
Aktivitu nastavení spravuje třída *activities.SettingsActivity*. V této aktivitě se nachází funkční prvky pro jednotlivá nastavení – tloušťky „kreslícího pera“, intenzity šumu, počtu neuronů a volbu typu vybavování (synchronní/asynchronní). Také jsou zde tlačítka pro import ob-

rázků, správu obrázků a potvrzení navoleného nastavení. Tlačítko import obrázků spouští systémovou aktivitu pro výběr obrázků, která zobrazí dostupné nástroje pro zobrazení a výběr obrázků. Pokud to daný nástroj umožňuje, je možné zvolit několik obrázků najednou. Tlačítko správy obrázků spouští aktivitu pro správu obrázků v paměti aplikace (naučených obrazců).



Obrázek 9: Grafické rozložení aktivity nastavení

Poslední aktivitou je aktivita pro správu obrázků v paměti aplikace. Je spravována třídou *activities.ImagesViewActivity*, která se mimo jiné také stará o dynamické generování prvků typu *android.widget.ImageView*, které zobrazují jednotlivé obrázky ze složky aplikace. Tyto obrázky lze vybrat a následně tlačítkem pro potvrzení smazat (provede se i jejich odnaučení). Také je zde tlačítko pro zrušení (návrat zpět do nastavení).



Obrázek 10: Grafické rozložení aktivity pro správu obrázků

6.4 Práce se soubory

V aplikaci se pracuje s bitmapovými obrázky reprezentující obrazce, se kterými pracuje neuronová síť. Pak se také pracuje se souborem reprezentujícím matici vah neuronové sítě a souborem obsahujícím názvy souborů naučených obrázků a jejich hash (zjednodušeného obrazce).

6.4.1 Bitmapové obrázky

Práce s obrázky začíná v momentě, kdy se zvolí jeden či více takových to obrázků k importu do aplikace. Každý obrázek se otevře v aplikaci, změní se mu velikost, aby odpovídala rozměrům kreslícího plátna (DrawingCanvas), a poté se uloží do podsložky aplikace určené pro obrázky. Dále se obrázek v aplikaci převede na zjednodušený obrazec (SimplifiedImage) a vygeneruje se odpovídající hash. Hash obrazce i s názvem souboru jsou uchovány v paměti pro

potřeby rozpoznání obrazce. Poté, co se takto připraví všechny obrázky, se spustí proces učení sítě.

Když se stanoví odezva sítě, sestaví se z ní zjednodušený obrazec, ze kterého se pak vygeneruje hash. Pokud se tato hash nachází v paměti aplikace, načte se korespondující obrázek a vykreslí se na plátno, čímž se docílí zobrazení výsledku v plném rozlišení, přesto že neuronová síť pracuje s podstatně zjednodušenými obrazci.

Posledním případem, kdy se pracuje se soubory obrázků je jejich odstranění z paměti. Obrázky zvolené v aktivitě správy obrázků se po potvrzení odstraní z podsložky aplikace a dojde také k odstranění jejich názvů a korespondujících hash z paměti aplikace, viz kapitola 6.4.2. Také dojde k jejich odnaučení z neuronové sítě.

6.4.2 Soubor „bindings“

Tento soubor uchovává v textové podobě názvy souborů naučených obrázků a hash jejich zjednodušených obrazců. Při inicializaci aplikace se z tohoto souboru načtou všechna data do paměti aplikace, kde se uchovávají v podobě hash mapy (klíčem je hash, hodnotou název souboru).

Pokud nastane import nových obrázků, tak se po uložení všech nových párů do hash mapy automaticky přeuloží soubor *bindings* aby odpovídal novému stavu paměti. V případě mazání obrázků mohou nastat dvě situace.

Pokud se bude mazat méně než polovina obrázků, tak se pro každý obrázek z hash mapy získá hash jeho zjednodušeného obrazce. Z této hash se pak sestaví obrazec v podobě jednorozměrného pole hodnot, které se pak použije k odnaučení daného obrazce z neuronové sítě. Po odnaučení všech požadovaných obrazců se odstraní jejích záznamy z hash mapy a dojde k aktualizaci souboru *bindings*.

Pokud se bude mazat více než polovina obrázků, tak se z paměti odstraní rovnou a také se kompletně vyčistí hash mapa i celá neuronová síť. Poté se spustí naučení zbývajících obrázků v podsložce, které povede ke stejné situaci, která nastane při importu obrázků.

6.4.3 Soubor „weights“

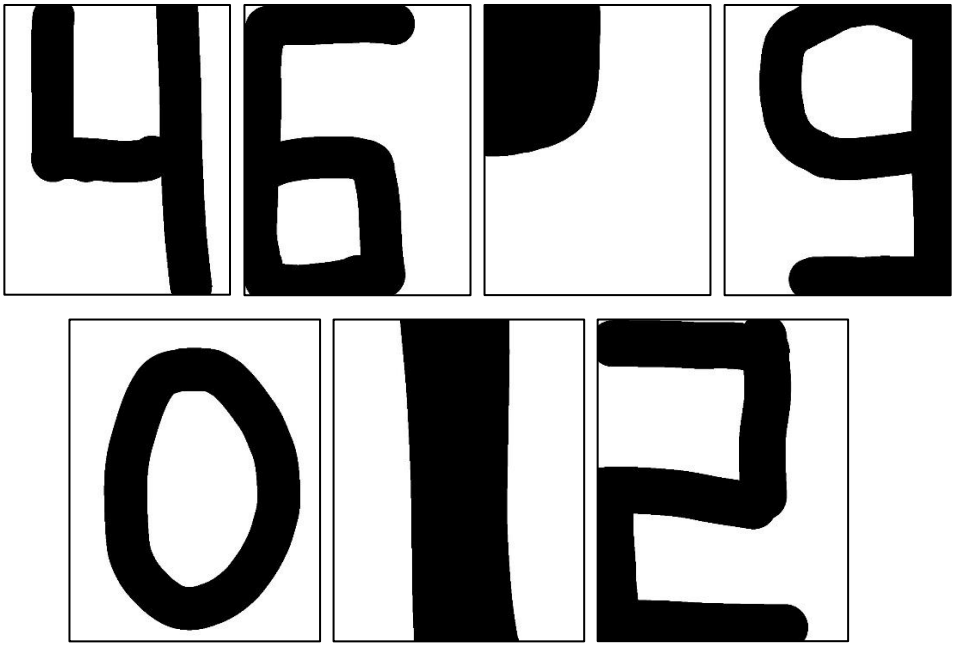
Posledním souborem, se kterým aplikace pracuje, je binární soubor pro uchování vah neuronové sítě. Matice vah obsahuje hodnoty pro všechna spojení mezi neurony sítě. Nabývá tedy

poměrně velkého počtu hodnot (druhé mocnině počtu neuronů) a přesto že se hodnoty ukládají pouze jako *short* (2 B na hodnotu), tak se jedná o velké množství dat. Například pro 6000 neuronů by bylo potřeba něco přes 34 MB. I když se to nezdá jako velké číslo, tak je třeba mít na paměti, že se tento soubor načítá při každém spuštění aplikace a také dochází k jeho přepsání po každé změně sítě (učení/zapomínání obrazců či změna počtu neuronů v nastavení).

Pro zrychlení ukládání souboru se využilo bezztrátové komprese. Připravená matice vah se za pomocí *java.util.zip.ZipOutputStream* zkomprimuje do souboru. Výsledná úspora místa na disku byla více než 95 % a doba ukládání se také znatelně snížila. Vzhledem k potřebě daný soubor před přečtením dekomprimovat se doba spouštění aplikace mírně zvýšila.

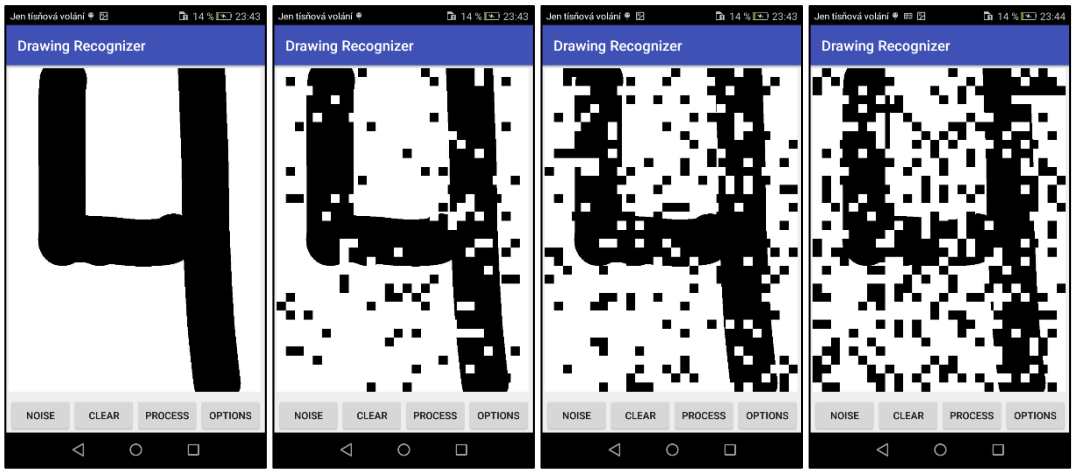
7 TESTOVÁNÍ APLIKACE

Pro testování funkčnosti neuronové sítě se zvolilo 7 testovacích obrazců. Každý obrazec byl poškozen určitým množstvím šumu a poté byl podroben zkoumání ve snaze nalézt původní obrazec. Tento postup byl několikrát zopakován pro každý obrazec. Takto se otestoval pro různé množství neuronů, šumu a také pro synchronní a asynchronní způsoby stanovení odezvy.



Obrázek 11: Testovací sada obrázků

Celkem bylo provedeno více než 2500 pokusů o rozpoznání poškozeného obrazce. Během tohoto testování se sestavily celkem 3 neuronové sítě o počtech neuronů 63, 252 a 1008. Každá z těchto sítí byla naučena stejnou testovací sadou obrazců. U každé sítě se testoval synchronní i asynchronní způsob vybavování a obrazce se poškozovali různými hladinami šumu – 15, 30 a 45 %.



Obrázek 12: Ukázka jednotlivých úrovní šumu

7.1 Výsledky

Příloha B obsahuje tabulku popisující celé testování. Během tohoto testování se ukázalo, že rozdíl mezi synchronním a asynchronním vybavování je minimální, pouze 0,56 %. V následující tabulce bude zobrazena efektivita sítě na základě počtu neuronů a úrovni poškození obrazce.

Tabulka 1: Výsledná úspěšnost testování

Parametry testování	15 % šumu	30 % šumu	45 % šumu
63 neuronů	80,36 %	71,79 %	54,64 %
252 neuronů	97,5 %	92,14 %	88,57 %
1008 neuronů	100 %	98,93 %	97,86 %

Z výsledků je patrné, že počet neuronů ovlivňuje funkčnost sítě. Při 63 neuronech síť dokázala v 80 % případů rozeznat z mírně poškozený obrazec. Při vyšším poškození obrazce úspěšnost začala značně klesat. Hlavním důvodem takto nízkých hodnot byla nevhodně zvolená testovací sada obrázků, protože po jejich zjednodušení na 63 bipolárních hodnot některé obrazce již neměly dostatek unikátních hodnot. Jmenovitě šlo obrazec číslice 4, který měl oproti ostatním mnohem nižší úspěšnost rozeznání a obrazec číslice 6, který se nepovedlo úspěšně rozeznat při žádném z pokusů. Teoretická kapacita této sítě je pouze 8 obrazců a ty by musely mít velmi rozdílné podoby, aby bylo dosaženo dostatečného počtu unikátních hodnot mezi nimi.

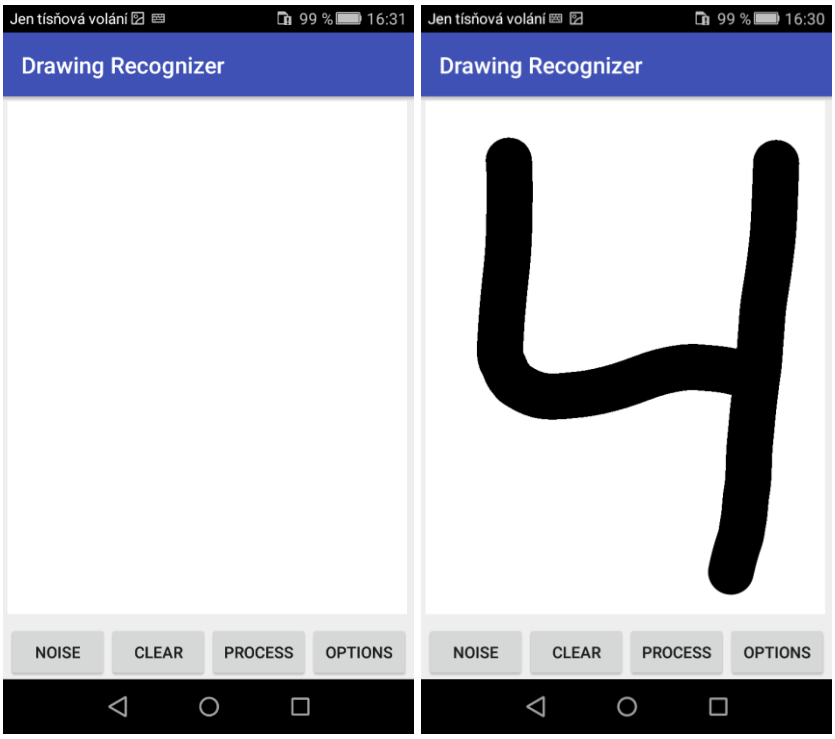
Při počtu 252 neuronů se již výsledky mnohem zlepšily. I když byly obrazce velmi poškozeny (45 % šumu), tak síť rozeznala přes 88 % obrazců. Obrazec číslice 6 byl sice oproti ostatním obtížnější rozeznatelný, ale i tak dosahovala úspěšnost testování u tohoto obrazce 73 %.

Poslední testovaná síť měla 1008 neuronů. V tomto případě i při vysokém poškození (45 % šumu) síť dosahovala úspěšnosti přes 97 %. Taková to hodnota není zas tak překvapující, protože testovací sada obsahovala pouze 7 obrazců a teoretická kapacita sítě při tomto počtu neuronů je podstatně vyšší.

8 UŽIVATELSKÁ PŘÍRUČKA

Po zapnutí aplikace se zobrazí hlavní obrazovka, kde lze na plátno kreslit za pomocí dotykového displeje. Ve spodní části obrazovky jsou tlačítka pro ovládání aplikace.

- **Noise** – Přidá šum do obrazce na plátně, vhodné pro testování neuronové sítě.
- **Clear** – Vymaže celé plátno.
- **Process** – Podrobí obrazec na plátně zkoumání neuronové sítě. Plátno se začne překreslovat na základě průběžných výstupů z neuronové sítě. Pokud se úspěšně nalezne obrazec, který byl naučen, zobrazí se na plátně v plném rozlišení.
- **Options** – Zobrazí veškerá možná nastavení aplikace.

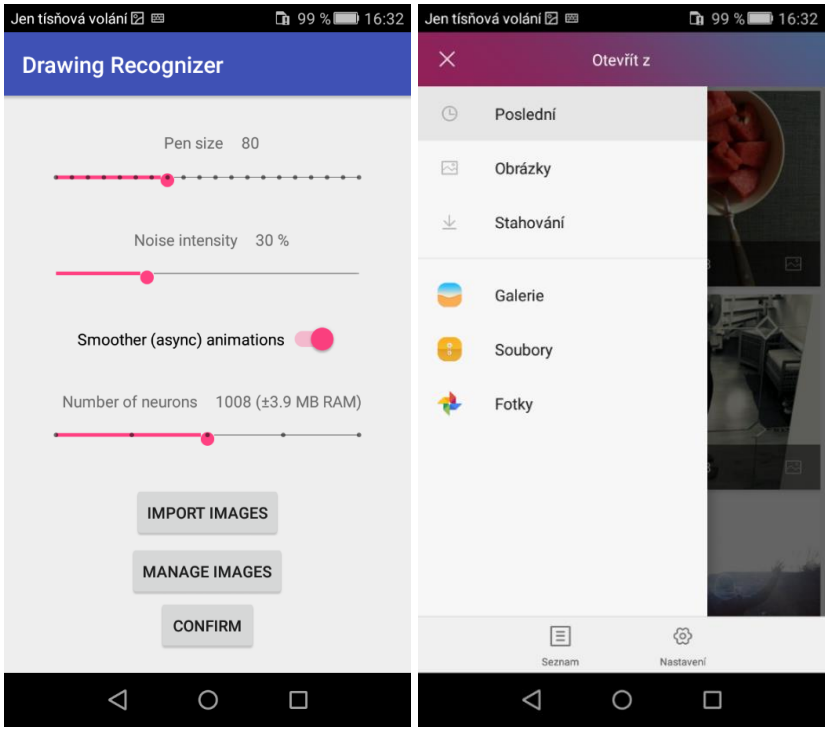


Obrázek 13: Hlavní obrazovka aplikace

Na obrazovce nastavení lze nastavit hned několik parametrů aplikace. Změny se projeví až po potvrzení tlačítkem a pokud není žádoucí uložit změny, lze se vrátit na hlavní obrazovku pomocí systémového tlačítka zpět.

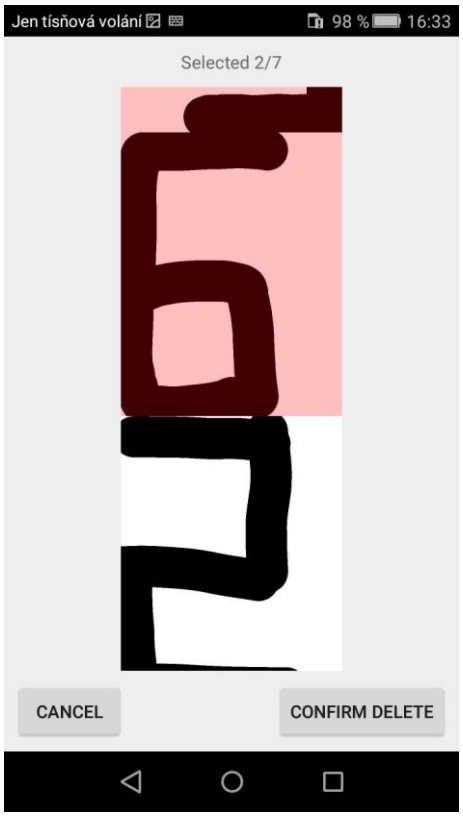
- **Pen size** – Nastaví tloušťku vykreslovaného tahu, hodnota udává počet pixelů.

- **Noise intensity** – Nastaví množství šumu, který lze do obrazce přidat. Pro nastavení konkrétní hodnoty lze stisknout zobrazovanou hodnotu pro vyvolání dialogu.
- **Smoother animations** – Přepínač, kterým lze zapnout asynchronní výpočet odezvy neuronové sítě. To také ovlivňuje animaci, asynchronní výpočet vytváří dojem hladkého průběhu. Synchronní výpočet se vykresluje po jednotlivých iteracích, takže je animace skoková a kratší.
- **Number of neurons** – Nastaví počet neuronů pro síť. V popisku se také zobrazuje odhadovaná paměťová náročnost, pokud zařízení není schopné poskytnout dostatek paměti, aplikace se vypne.
- **Import images** – Zobrazí obrazovku pro import obrázků. Tato obrazovka se liší podle toho, jaké nástroje pro import obrázků systém nabízí.
- **Manage images** – Zobrazí obrazovku, ve které lze spravovat obrazce v paměti.
- **Confirm** – Potvrdí změny, nevztahuje se na import/mazání obrázků, tyto změny se provádí automaticky pro potvrzení daných akcí.



Obrázek 14: Obrazovky nastavení a importu obrázků

Obrazovka pro správu obrázků zobrazuje obrázky v paměti. Obrázky se zobrazují pod sebou a lze je gestem posouvat. Tyto obrázky lze také vybrat, čímž se zvýrazní. Potvrzením se pak vybrané obrázky z paměti odstraní a provede se jejich odnaučení z neuronové sítě.



Obrázek 15: Obrazovka pro správu obrázků

ZÁVĚR

V rámci diplomové práce jsem vytvořil a otestoval aplikaci pro OS Android. Aplikace umožňuje rozeznávat nakreslené obrazce za pomocí Hopfieldovy neuronové sítě. Tuto síť je možné naučit nové obrazce, odnaučit ty staré a také měnit dva její klíčové parametry, počet neuronů a způsob, jakým síť přistupuje k vybavování obrazce. Aplikace má jednoduché a intuitivní ovládání. Pokud by s ovládáním aplikace přeci jen měl někdo problém, je součástí práce také uživatelská příručka, která popisuje veškeré ovládání.

Největší výzvou během vývoje aplikace byla optimalizace. Aplikaci jsem během celého vývoje testoval na starším telefonu s Androidem 5.1 a na dnešní poměry slabým výkonem. Díky tomu jsem musel optimalizovat některé náročnější části aplikace. Jmenovitě třeba velikost, jakou neuronová síť zabírá na disku. Několik desítek MB sice není v dnešní době mnoho, ale pokud se daný soubor přepisuje při každé změně neuronové sítě, tak to může zpomalit aplikaci. Tento problém elegantně vyřešila komprese.

Na druhou stranu jsem díky tomu docílil své snahy o vytvoření aplikace s neuronovou sítí, která se dá využít na mobilních zařízení. Důkazem toho může být například testování, během kterého jsem na výše zmíněném telefonu provedl přes 2 500 pokusů o rozpoznání obrazce a baterie se nevybila. Toto testování také prokázalo, že navržená neuronová síť pracuje s očekávanou přesností.

Přesto že jsem s výsledkem mé práce spokojen, tak pokud bych v budoucnu měl naprogramovat nějakou další mobilní aplikaci využívající neuronové sítě, využil bych některý z moderních frameworků. Obzvláště pokud by šlo o aplikaci, která řeší více komplexní úlohy. Nejen že tyto frameworky mnohdy zjednodušují implementaci umělé inteligence, ale také zpravidla vývojáři zpřístupňují optimalizované využití hardwarové akcelerace, která se čím dál více objevuje na moderních mobilních čipech.

POUŽITÁ LITERATURA

[1] HAYKIN, Simon. *Neural Networks: a comprehensive foundation*. 2. vydání. Singapore: Prentice Hall, 1999. ISBN 978-0132733502.

[2] DOLEŽEL, Petr. *Úvod do umělých neuronových sítí pro studenty vysokých škol*. Pardubice: Univerzita Pardubice, 2016. ISBN 978-80-7560-022-6.

[3] HEBB, Donald. *The Organization Of Behavior: A Neuropsychological Theory*. New York: John Wiley & Sons, 1949. ISBN 978-0415654531.

[4] Humayun Karim Sulehria and Ye Zhang, 2008. Study on the Capacity of Hopfield Neural Networks. *Information Technology Journal*, 7: 684-688. 10.3923/itj.2008.684.688.

[5] John Hopfield, 1982. Neural networks and physical systems with emergent collective coputational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79: 2554-2558. ISSN 0027-8424.

[6] Kapoor, Amita & Soni, Neha & Sharma, Enakshi. (2016). *Application of Hopfield neural network for face recognition*. 10.13140/RG.2.2.11433.60008.

[7] Shouhong Wang, Hai Wang. 2008. Password Authentication Using Hopfield Neural Networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38: 265-268. ISSN 1094-6977.

[8] KOJIC, N., I. RELJIN a B. RELJIN. Route Selection Problem Based on Hopfield Neural Network. Radioengineering [online]. Společnost pro radioelektronické inženýrství, 2013, 22(4), 1182-1193 [cit. 2019-05-06]. ISSN 1210-2512. Dostupné z: http://hdl.handle.net/11012/36972

[9] Vision AI. *Google Cloud* [online]. [cit. 2019-05-06]. Dostupné z: https://cloud.google.com/vision/

[10] Open CV [online]. [cit. 2019-05-06]. Dostupné z: https://opencv.org

[11] Vehicle make & model recognition LAW ENFORCEMENT. *Orpix computer vision* [online]. [cit. 2019-05-06]. Dostupné z: http://www.orpix-inc.com/vehicle-make-and-model-recognition-for-law-enforcement/

[12] I want to develop Android Apps. *Android authority* [online]. [cit. 2019-05-06]. Dostupné z: https://www.androidauthority.com/develop-android-apps-languages-learn-391008/

[13] Hardware acceleration for machine learning on Apple and Android devices. *Heartbeat* [online]. [cit. 2019-05-06]. Dostupné z: https://heartbeat.fritz.ai/hardware-acceleration-for-machine-learning-on-apple-and-android-f3e6ca85bda6

[14] Get Ready for Core ML 2. *Apple developer* [online]. [cit. 2019-05-06]. Dostupné z: https://developer.apple.com/machine-learning/

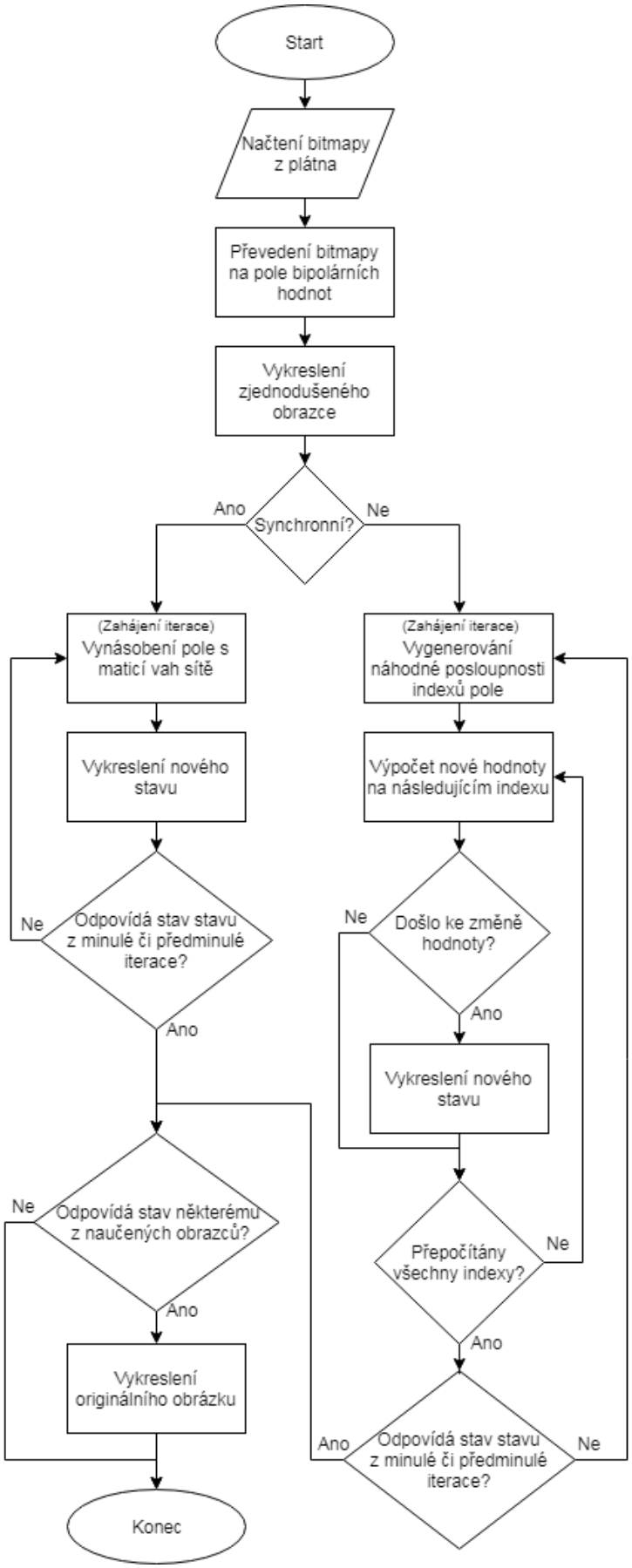
[15] TensorFlow Lite guide. *TensorFlow* [online]. [cit. 2019-05-06]. Dostupné z: https://www.tensorflow.org/lite/guide

[16] Neural Networks API. *Android developers* [online]. [cit. 2019-05-06]. Dostupné z: https://developer.android.com/ndk/guides/neuralnetworks/

[17] Snapdragon Neural Processing Engine SDK. *Qualcomm developer* [online]. [cit. 2019-05-06]. Dostupné z: https://developer.qualcomm.com/docs/snpe/overview.html

PŘÍLOHY

PŘÍLOHA A – DIAGRAM ROZPOZNÁNÍ OBRAZCE



PŘÍLOHA B – TABULKA TESTOVÁNÍ

Počty úspěšných rozpoznání poškozeného obrazce									
Počet neuronů	**styl**	**šum**							
63	**Synchronní**	**15 %**	20/20	19/20	19/20	16/20	0/20	20/20	20/20
		30 %	20/20	14/20	19/20	14/20	0/20	18/20	19/20
		45 %	14/20	12/20	12/20	6/20	0/20	17/20	14/20
	Asynchronní	**15 %**	20/20	17/20	14/20	20/20	0/20	20/20	20/20
		30 %	19/20	15/20	12/20	16/20	0/20	16/20	19/20
		45 %	17/20	14/20	10/20	11/20	0/20	11/20	15/20
252	**Synchronní**	**15 %**	20/20	20/20	19/20	20/20	17/20	20/20	20/20
		30 %	20/20	20/20	15/20	20/20	13/20	20/20	20/20
		45 %	20/20	19/20	14/20	20/20	12/20	20/20	19/20
	Asynchronní	**15 %**	20/20	20/20	20/20	20/20	17/20	20/20	20/20
		30 %	19/20	20/20	17/20	20/20	14/20	20/20	20/20
		45 %	20/20	18/20	13/20	20/20	14/20	20/20	19/20
1008	**Synchronní**	**15 %**	20/20	20/20	20/20	20/20	20/20	20/20	20/20
		30 %	20/20	20/20	20/20	20/20	19/20	20/20	20/20
		45 %	20/20	20/20	19/20	20/20	19/20	20/20	20/20
	Asynchronní	**15 %**	20/20	20/20	20/20	20/20	20/20	20/20	20/20
		30 %	20/20	20/20	19/20	20/20	19/20	20/20	20/20
		45 %	20/20	20/20	18/20	20/20	18/20	20/20	20/20