VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

INTELIGENTNÍ ROZPOZNÁNÍ ČINNOSTI UŽIVATELE CHYTRÉHO TELEFONU

INTELLIGENT RECOGNITION OF THE SMARTPHONE USER'S ACTIVITY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. MICHAL PUSTKA**
AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. MARTIN DRAHANSKÝ, Ph.D.**
SUPERVISOR

BRNO 2018

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2017/2018

Zadání diplomové práce

Řešitel: **Pustka Michal, Bc.**
Obor: Inteligentní systémy
Téma: **Inteligentní rozpoznání činnosti uživatele chytrého telefonu**
Intelligent Recognition of the Smartphone User's Activity
Kategorie: Umělá inteligence

Pokyny:

1. Prostudujte literaturu týkající se sběru a analýzy dat a programování aplikací pro operační systém Android.
2. Nasbírejte data ze senzorů, které jsou osazeny v běžných chytrých telefonech.
3. Analyzujte data ze senzorů a určete, které senzory jsou vhodné pro rozpoznávání běžných činností uživatele.
4. Zvolte vhodnou metodu soft-computingu pro zpracování těchto dat za účelem rozpoznání činnosti uživatele (např. jízda vozem, chůze).
5. Vytvořte knihovnu pro rozpoznávání činnosti uživatele pro platformu Android.
6. Proveďte základní experimenty s touto knihovnou a shrňte dosažené výsledky.

Literatura:

- Lee Y.S., Cho S.B. *Activity Recognition Using Hierarchical Hidden Markov Models on a Smartphone With 3D Accelerometer*. In: Hybrid Artificial Intelligent Systems. Springer Berlin Heidelberg, 2011. pp. 460-467.
- Dash Y., Kumar S., Patle V.K. *A Novel Data Mining Scheme for Smartphone Activity Recognition by Accelerometer Sensor*. In: Proceedings of the 4th International Conference on Frontiers in Intelligent Computing: Theory and Applications (FICTA) 2015. Springer, 2015. p. 131.
- Rabin C., Bock B. *Desired Features of Smartphone Applications Promoting Physical Activity*. Telemedicine and e-Health, 2011, 17.10: 801-803.

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

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% 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: **Drahanský Martin, prof. Ing., Dipl.-Ing., Ph.D.**, UITS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 23. května 2018

L.S.

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Tato práce se zabývá detekcí uživatelských aktivit (např. běhání, chůze, jízda autě atd.) v reálném čase, přičemž k samotné detekci je využito senzorů dostupných v mobilních zařízeních. V rámci práce vznikla aplikace pro sběr dat ze senzorů, pomocí které byla nasbírána data. Dalším produktem této práce je program pro předzpracování naměřených dat a vytvoření datasetu. Součástí práce je vlastní návrh konvoluční neuronové sítě, která slouží ke klasifikaci aktivit. Celou implementační část uzavírá vytvořená knihovna pro detekci aktivit na mobilních zařízeních s Android OS. Spojením všech aplikací vzniká komplexní framework pro vývoj aplikace, využívající detekci uživatelských aktivit. Na závěr jsou provedeny některé zajímavé experimenty pomocí tohoto frameworku (např. vliv konkrétních senzorů na výkon detekce).

Abstract

This thesis deals with real-time human activity recognition (eg, running, walking, driving, etc.) using sensors which are available on current mobile devices. The final product of this thesis consists of multiple parts. First, an application for collecting sensor data from mobile devices. Followed by a tool for preprocessing of collected data and creation of a data set. The main part of the thesis is the design of convolutional neural network for activity classification and subsequent use of this network in an Android mobile application. The combination of previous parts creates a comprehensive framework for detection of user activities. Finally, some interesting experiments were made and evaluated (eg, the influence of specific sensors on detection precision).

Klíčová slova

Senzory, Android, Xamarin, TensorFlow, sběr dat, umělá inteligence, rozpoznávaní aktivit, konvoluční neuronové sítě

Keywords

Sensors, Android, Xamarin, TensorFlow, collection of data, artificial intelligence, human activity recognition, convolutional neural networks

Citace

PUSTKA, Michal. *Inteligentní rozpoznání činnosti uživatele chytrého telefonu.* Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Drahanský Martin.

Inteligentní rozpoznání činnosti uživatele chytrého telefonu

Prohlášení

Prohlašuji, že jsem tuto semestrální práci vypracoval samostatně pod vedením pana prof. ing. Martina Drahanského Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Michal Pustka
21. května 2018

Poděkování

Děkuji vedoucímu mé práce prof. Ing. Martinovi Drahanskému Ph.D. za odborné vedení a cenné připomínky při tvorbě této diplomové práce.

Obsah

Kapitola 1

Úvod

V dnešní éře chytrých mobilních zařízení, které jsou osazeny nejrůznějšími typy senzorů a zároveň roste hardwarový výkon těchto mobilních zařízení, se nabízí možnosti využití nekonvenčních metod umělé inteligence a strojového učení přímo na těchto zařízeních. Pokud chytrá zařízení mají být opravdu „chytrá“, je potřeba, aby zařízení znalo kontext (co uživatel děla, v jaké situaci se nachází). Ke znalosti tohoto kontextu může velmi významně přispět detekce uživatelských aktivit (běh, chůze, jízda autem). Práce se tedy zabývá inteligentním zpracováním dat ze senzorů mobilního zařízení za účelem detekce uživatelských aktivit. Detekce takové aktivity v reálném čase pak může mít široké možnosti použití, např. v oblasti inteligentních domácností, her, detekce životních funkcí záchranářů či v samotném přizpůsobování uživatelského rozhraní zařízení pro různé uživatelské aktivity. Konkrétně se tato práce zabývá detekcí uživatelských aktivit na zařízení s operačním systémem Android. Detekce je založena na datech ze senzorů.

První kapitola se zaobírá rozborem samotných zařízení, senzorů a přístupem k nim z operačního systému Android. Proto, aby mohlo být aplikováno strojové učení, je potřeba nasbírat data ze zařízení od různých uživatelů a tato data označit. Této problematice se věnuje kapitola druhá a třetí, ve které je popsáno vytvoření aplikace pro Android OS pomocí multiplatformního frameworku Xamarin. Čtvrtá kapitola je čistě teoretická, jsou zde uvedeny a rozděleny současné metody detekce aktivit. Dále je zde uveden stručný teoretický zaklad ke konvolučním neuronovým sítím. Šestá kapitola se zabývá implementací a návrhem vlastního detektoru uživatelských aktivit založeného na konvolučních neuronových sítích, je zde představena vlastní architektura sítě a metody, jak zvýšit její výkon. Sedmá kapitola popisuje implementaci knihovny pro detekci aktivit pro platformu Android. Vytvořením knihovny pak vzniká ucelený framework aplikací pro vývoj aplikace v prostředí Xamarin, která využívá detekci aktivit. V poslední kapitole jsou pomocí tohoto frameworku, provedeny některé zajímavé experimenty (výběr senzorů a jejich kombinace, výkon při detekci atd.).

Kapitola 2

Přístup k senzorům na Android OS

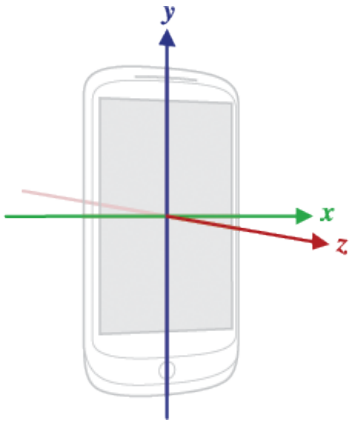
Zařízení s operačním systémem Android obsahují tři druhy vestavěných senzorů: senzory měřící pohyb zařízení, orientaci zařízení a senzory měřící různé vlivy okolního prostředí, např. teplotu či tlak. Krom toho se senzory rozdělují na dva typy, tj. hardwarové a softwarové. Hardwarové senzory si lze představit jako konkrétní realizované obvody se specializací na konkrétní činnost měření na daném mobilním zařízení. Naproti tomu softwarové senzory jsou jakási abstrakce nad jedním či více senzory, v literatuře též označovány za senzory virtuální či syntetické. [13]

2.1 Wakeup a non-wakeup senzory

Pro pochopení této problematiky je nutno se dotknout oblasti jakým způsobem se Android OS snaží šetřit energii baterie. Tato úspora energie probíhá pomocí tří režimu: On, Idle a Suspend v systému na čipu, v anglické literatuře označována jako system of chip (SoC). V režimu On SoC normálně pracuje, v režimů Idle je SoC napájen, ale nevykonává žádné úkoly a v posledním režimu Suspend SoC napájen není. Zařízení se přepne do úsporného režimu Suspend, pokud není nabíjeno a uživatel se zařízením neinteraguje. Při tomto režimu některé senzory přestanou pracovat. Do verze Android 5.0 Lollipop neměli vývojáři možnost zjistit, jestli daný senzor bude snímat i po uvedení zařízení do úsporného režimu, vše záleželo na výrobci zařízení, jakými senzory telefon osadil. Od verze Android 5.0 lze zjistit pomocí systémového api, jestli je senzor tzv. wakeup, tedy může snímat při uvedení zařízení do úsporného režimu, či nikoli a je tedy non-wakeup. V případě, že senzor je non-wakeup, nabízí systém FIFO frontu, kde po dobu, kdy je zařízení v úsporném režimu, zapisuje do této fronty data. Tato fronta je pak poslaná jako dávka a zpracována při uvedení zařízení zpět do běžného režimu. Avšak, každý senzor má tuto frontu omezenou v závislosti na výrobci senzoru. Tedy pokud se fronta přeplní, jsou nejstarší hodnoty přepsány nejnovějšími vzorky. Bohužel u levných a starších zařízení je mnoho senzorů non-wakeup s velikosti fronty nula. To způsobuje, že sbírat data, zatímco je zařízení v úsporném režimu, není možné, což detekci uživatelovy aktivity značně komplikuje. Naštěstí existují neoficiální postupy, které na většině běžných zařízení tento problém řeší. Tyto postupy budou popsány v následující kapitole „Sběr dat z non-wakeup senzorů“. [16] [10]

2.2 Senzory měřící pohyb a orientaci

Mezi senzory měřící pohyb a orientaci patří především akcelerometr, senzor gravitace, gyroskop, senzor měřící geomagnetické pole a další senzory. Senzory pak používají souřadnicový systém, který je definován ve vztahu k obrazovce zařízení v jeho výchozí orientaci. Osa X je horizontální, směřující k pravé straně, osa Y je svislá, směřující nahoru a osa Z směřuje ze středu obrazovky směrem k pozorovateli. Hodnoty osy Z za obrazovkou mají zápornou hodnotu. Souřadnicový systém je ilustrován na obr 2.1. [9]



Obrázek 2.1: Osy mobilního zařízení [13].

Akcelerometr je hardwarový senzor, který měří zrychlení ve třech osách. Na akcelerometr působí dva druhy sil: síla gravitační a síla způsobená pohybem zařízení. Android poskytuje dva typy akcelerometrů. Obecný akcelerometr označený jako `TYPE_ACCELEROMETER` měří zrychlení ve třech osách, které zahrnuje gravitační zrychlení, v jednotkách ms^{-2}, kdežto akcelerometr typu `TYPE_LINEAR_ACCELERATION` měří zrychlení bez vlivu gravitační síly. Obecný akcelerometr je vždy hardwarový senzor a senzor měřící lineární akceleraci je nejčastěji softwarový senzor, pracující nad obecným akcelerometrem a senzorem gravitace. Dalším senzorem je gyroskop, který je rovněž hardwarový senzor, avšak na rozdíl od akcelerometru měří úhlovou rychlost v $rads^{-1}$ ve stejných osách. Dalším důležitým senzorem je senzor gravitace, který měří směr gravitace a gravitační zrychlení v jednotkách ms^{-2} opět ve třech osách. Pokud je zařízení v klidu je výstup senzoru gravitace a akcelerometru identický. Tento senzor může být jak softwarový, tak hardwarový, záleží na výrobci zařízení. Zvláštním senzorem je pak proximity senzor, který měří vzdálenost zařízení od překážky. Nejčastěji se tento senzor využívá pro detekci přiloženého ucha k zařízení při telefonování, pokud je detekována překážka, je vypnut displej, tak aby nedošlo nechtěnému dotyku na ovládací prvky uživatelského rozhraní. Senzor vrací hodnoty v centimetrech, avšak někteří výrobci zařízení implementují senzor tak, aby vracel pouze dva stavy, tedy maximální hodnotu senzoru, pokud není překážka přítomná nebo nulovou hodnotu, pokud je před zařízením překážka. Posledním čistě hardwarovým senzorem této kategorie je senzor měřící geomagnetické pole, který měří geomagnetické pole ve třech osách v jednotkách μT. Ostatní senzory pohybu či orientace jsou již čistě softwarové senzory pracující s daty těchto hardwarových senzorů. [5] [7]

2.3 Senzory měřící okolní prostředí

Android OS v současné době podporuje čtyři druhy senzorů okolního prostředí: senzor teploty, vlhkosti, tlaku a světla. Senzor teploty má pak dvě varianty: teplota samotného prostředí a teplota zařízení. Obě teploty jsou měřeny ve stupních celsia, vlhkost je pak relativní vlhkost prostředí v procentech, tlak je atmosféricky tlak v hekto pascalech a úroveň osvětlení prostředí v luxech. [1]

2.4 Senzor Framework

Základní přístup k senzorům poskytuje Android OS přes tzv. Senzor Framework. Tento framework obsahuje tři třídy: `SensorManger`, `Sensor`, `SensorEvent` a jedno rozhraní `SensorEventListener`. Třída `SensorManger` obsahuje tři důležité metody. Metodu `getListSensors`, která vrací seznam všech dostupných senzorů a dále obsahuje metody registerListener a unregisterListener, které umožnují pro konkrétní senzory registrovat či odregistrovat události `onAccuracyChanged` a `onSensorChanged`. Událost `onSensorChanged` je vyvolána při každém sejmutí hodnoty z vzorku z daného senzoru a událost `onAccuracyChanged` při změně přesnosti senzoru. Instance třídy `SensorEvent` je pak argument metody obsluhující událost `onSensorChanged`. Tato třída obsahuje pole hodnot v plovoucí řadové čárce, což jsou samotné hodnoty veličiny snímané daným senzorem, přesnost, jakým byly hodnoty naměřeny a časové razítko samotného měření. Poslední důležitá třída Sensor, představuje abstrakci nad jednotlivými senzory. [11] Dle [12] lze pomocí této třídy zjistit základní parametry senzoru jako:

- proud, který senzor odebírá při měření
- rozlišení – nejmenší jednotka
- maximální a minimální zpoždění mezi dvěma vzorky
- maximální rozsah hodnot senzoru
- režim snímání, ten může být buď kontinuální na základě času, dále se změnou veličiny měřené senzorem, nebo pouze jedna změna, či jiný speciální režim pro konkrétní senzor
- příznak, jestli je senzor non-wakeup či wakeup
- rezervovaná velikost FIFO a maximální velikost FIFO fronty (tato hodnota nemusí odpovídat rezervované velikosti, jelikož i jiné aplikace mohly rezervovat část fronty pro senzor)
- verze a výrobce senzoru

2.5 Verze Android OS a senzory

Pro různé typy senzorů, jejichž parametry a nastavení byly přidávány postupně s vývojem systému či nových generací zařízení, poskytuje systém řadu funkcí pro detekci dostupných senzorů, v případně se lze explicitně dotázat systému na konkrétní senzor pomocí metody `getDefaultSensor` ve třídě `SensorManager`. [13]

Kapitola 3

Návrh aplikace pro sběr dat

Pro účely sběru dat ze senzorů bylo potřeba zhotovit aplikaci. Samotné požadavky na aplikaci byly: vytvořit aplikaci tak, aby fungovala obecně na jakémkoliv zařízení s operačním systémem Android 4.0 a vyšší, s různými druhy a parametry senzorů. Zároveň měla aplikace data agregovat, jelikož některé senzory mají větší vzorkovací frekvenci a zařízení obecně může mít velké množství senzorů. Agregace pak má za úkol agregovat nasbíraná data tak, aby byla, pokud možno, co nejmenší. Další z požadavků na uživatelské rozhraní aplikace je takový, aby uživatel mohl jednoduše zadat jak předefinovanou aktivitu či sám definovat novou aktivitu, kterou provozuje při sběru dat. Další důležitou částí rozhraní je možnost zadat polohu, kde se zařízení nachází při sběru dat. Jelikož se dopředu předpokládá velké množství uživatelů, kteří budou sbírat data, je potřeba co nejjednodušším způsobem dostat tato nasbíraná data na nějaké centrální úložiště. Z tohoto důvodu byl uvažován požadavek, aby aplikace pomoci FTP protokolu přenesla data na pozadí na předem definované centrální úložiště. Doplňující požadavek byl takový, aby se data přenášela pouze přes WiFi síť a nečerpala uživateli mobilní data.

3.1 Xamarin Framework

Z důvodu rychlejšího vývoje a ambice, aby aplikace mohla být časem multiplatformní, resp. její část, byla aplikace napsána v multiplatformním frameworku Xamarin. Tento framework umožnuje psát aplikace v jazyce C# či využívat veškeré jeho možnosti a používat tzv. Base Class Library (BCL) knihovny z prostředí .NET a další tisíce knihoven postavených nad BCL. Krom toho Xamarin umožnuje používat i nativní knihovny pro dané platformy jak Android, tak iOS. Velkou výhodou frameworku Xamarin je, že kód aplikace neinterpretuje. Zatímco na Androidu překládá Xamarin do mezijazyka, který je následně v tzv. Just In Time (JIT) kompilaci přeložen do nativního strojového kódu, na platformě iOS je aplikace přímo přeložena do ARM assembleru. Z tohoto důvodu Xamarin framework zajišťuje vysoký výkon oproti ostatním multiplatformním frameworkům. Framework Xamarin umožňuje psát mobilní aplikaci dvěma způsoby. První možnost je psát aplikaci pomocí tzv. Xamarin.Forms. Tato technologie umožnuje napsat mobilní aplikaci plně multiplatformní, především její grafické rozhraní. Obsahuje tedy průnik všech platforem, z čehož vyplývá, že nelze využívat veškeré možnosti konkrétní platformy. Tato technologie je využívána hlavně pro tvorbu prototypu a také jednoduchých aplikací pracujících nad daty či aplikací, které velmi málo využívají platformě specifické funkce. Druhá možnost je využití tzv. Xamarin.Android či Xamarin.iOS knihoven, které obalují nativní systémové API da-

ného systému prakticky jedna ku jedné. Tedy systémové funkce a třídy mají stejné názvy, pouze přizpůsobené konvencím jazyka C#. To má výhodu, že vývojáři nejsou závislí na dokumentaci frameworku Xamarin, ale můžou používat oficiální dokumentaci pro danou platformu. Multiplatformní vývoj pod touto technologii probíhá tak, že sdílený kód, především datová vrstva a byznys vrstva je ve formě sdílené knihovny. Samotné uživatelské rozhraní či platformě specifický kód je v konkrétním projektu pro danou platformu. [21] [22]

3.2 Návrh

Z hlediska uživatelského rozhraní aplikaci rozdělíme na dvě části: část, kde uživatel zadává a přidává jeho aktivity či zadává polohu zařízení a část, kde uživatel bude moct měnit nastavení aplikace. Kromě uživatelského rozhraní bude muset aplikace implementovat dvě služby. První služba bude sloužit pro sběr dat ze samotných senzorů zařízení a druhá služba bude sloužit pro přenos nasbíraných dat na centrální úložiště.

3.3 Sběr dat z non-wakeup senzorů

Vzhledem k tomu, že požadavky byly takové, aby aplikace fungovala od Android 4.0 a vyšší, nelze se na příznak wakeup a non-wakeup spolehnout, jelikož tyto příznaky systém podporuje až ve verzi Android 5.0 Lollipop. Je nutno implicitně předpokládat, že všechny senzory jsou typu non-wakeup. Jak již bylo v předchozí kapitole zmíněno, starší a levné zařízení mají nulové FIFO fronty pro ukládání vzorků. Proto je potřeba vyřešit tento problém, když je zařízení v úsporném režimu a senzor na pozadí snímá. Problém lze vyřešit neoficiálním postupem podle článku [48]. Postup má dva kroky. První krok je použití PowerManageru ze systému Android, který může nastavovat tzv. wakup lock, tedy režimy zařízení, do kterých zřízení přejde, pokud uživatel dlouho neinteraguje se zařízením nebo pokud uživatel zařízení uspal. Tato třída nabízí čtyři režimy `FULL_WAKE_LOCK`, `SCREEN_BRIGHT_WAKE_LOCK`, `SCREEN_DIM_WAKE_LOCK` a `PARTIAL_WAKE_LOCK`. Při režimu `FULL_WAKE_LOCK` je zařízení plně aktivní včetně displeje a klávesnice, tento režim je vhodný především aplikace typu hry. V případě dalších dvou režimů je displej zařízení zapnut, avšak klávesnice není aktivní. V posledním režimu je displej a klávesnice vypnuta a na pozadí běží CPU. Do tohoto režimu potřebujeme zařízení dostat, abychom mohli snímat data ze senzorů, avšak pouze toto nestačí. Další krok je pomocí třídy BroadcastReceiver, která umožňuje naslouchat na různé systémové události. Pro tyto účely potřebujeme naslouchat na událost typu `ACTION_SCREEN_OFF`, tato událost indikuje, že zařízení není aktivní a displej je vypnut. Jakmile dojde k této události, je potřeba inicializovat zpožděné vlákno, nejčastěji se uvádí 500ms až 2s. Toto vlákno pak odregistruje naslouchání na senzoru a vzápětí zase naslouchání na senzor zaregistruje. Tímto postupem dojde překonání problému snímaní dat z non-wakeup senzorů a senzor snímá data na pozadí, i když je zařízení neaktivní. Bohužel není nikde garantováno, že uvedený postup bude fungovat na všech zařízeních, avšak z empirických zjištění tento postup fungoval na všech běžných modelech smartphonu a tabletu. Tento postup má pak dva nežádoucí efekty, a to slabší výdrž baterie a sníženou maximální vzorkovací frekvenci senzorů. Vzorkovací frekvence je však i tak dostačující pro účely detekce uživatelovy aktivity.

3.4 Služby

Jelikož dvě stěžejní části aplikace jsou služby (služba zajišťující sběr dat ze senzorů a služba zajišťující zasílaní dat na FTP úložiště), je potřeba popsat realizaci služeb na operačním systému Android. Systém Android nabízí tři typy služeb: `Scheduled`, `Started` a `Bound`. První typ služby slouží k naplánování úkolů, které běží ve vlastním procesu. Tento typ služeb je možno použít pomoci tzv. JobScheduler API představeného v Androidu 5.0 Lollipop. Další typ služby je Started, tento typ služby nejčastěji nic nenavrací a běží buď kontinuálně či vykonává nějakou dlouho trvající operaci, např. nahrávání dat na server, kdy se služba sama ukončí. Poslední typ služby Bound nabízí klient-server rozhraní. Jednotliví klienti mohou připojovat své požadavky službě, klienti mohou být i v rámci jiných procesů. Jakmile tato služba požadavky klientu vykoná, tak zanikne. Důležitou zmínkou je, že implicitně služby běží ve stejném procesu a v hlavním vlákně, které obsluhuje uživatelské rozhraní. Pokud vykonáváme uvnitř služby dlouhotrvající blokující operace, projeví se tato skutečnost nežádoucím zneaktivněním prvků uživatelského rozhraní aplikace. Je několik možností, jak tento nežádoucí efekt eliminovat. První možnost je nastavit explicitně v manifestu aplikace u dané služby vlastnost proces a tím spustit službu v jiném procesu než obsluha uživatelského rozhraní či nastavit příznak isolatedProcess, kdy bude služba spuštěna ve vlastním izolovaném procesu. Další mnohem častější možností je vytvoření nového vlákna uvnitř služby pro blokující operace. Krom výše zmíněného chování je potřeba mít na paměti, že operační systém může kdykoliv danou službu ukončit z důvodu nedostatku operační paměti. Větší pravděpodobnost ukončení služby operačním systémem mají dlouho běžící služby na pozadí, kdežto služby běžící na popředí, tj. služby, o kterých je uživatel systémem notifikován pomocí notifikace v notifikačním centru, systém zřídka kdy ukončuje (většinou se jedná o služby přehrávající audio, bylo by nežádoucí uživateli tyto služby ukončovat). Kromě toho je možnost nastavit návratovou hodnotu při startu služby vyvolání metody `OnStartCommand` jako `START_STICKY`, která určuje, že pokud je systémem služba ukončena, je pak při dostatku prostředků zase vytvořena a spuštěna. Pokud je nastavena tato volba, je vhodné službu navrhnout tak, aby se uměla zotavit při znovuspuštění.[15] [8] [14]

3.5 Doze

Doze je relativně nová technologie pro zvýšení výdrže baterie, představená v Androidu 6.0. Tato technologie funguje tak, že jakmile přestane uživatel interagovat se zařízením displej zařízení se vypne a je spuštěn úsporný režim Doze, který služby na pozadí omezuje především tak, že brání službě přepnout zařízení do plného úsporného režimu. Tato skutečnost pak komplikuje sbírání dat ze senzorů. Naštěstí existují dvě možnosti, jak tento problém vyřešit. První možnost je požádat o výjimku systému, tato možnost je však nevhodná z důvodu restrikcí GooglePlay obchodu s aplikacemi, kde jsou docela přísné podmínky pro tuto žádost. Druhá možnost je spustit službu na popředí tak, aby o ní byl uživatel obeznámen pomocí notifikace. Touto možnost lze použít pro snímaní dat ze senzorů. [6]

Kapitola 4

Realizace aplikace pro sběr dat

Jak již bylo uvedeno výše, k implementaci aplikace v jazyce C# byl použit multiplatformní framework Xamarin konkrétně Xamarin.Android. Hlavním důvodem zvolení tohoto přístupu je pozdější rozšíření aplikace o multiplatformní knihovnu, která bude data ze senzorů klasifikovat.

4.1 Analýza senzorů

Proto, aby mohla být data ze senzorů sbírána je potřeba nejprve analyzovat dostupné senzory a jejich parametry. K tomuto účelu slouží třídy `AnalyzerSensor`, `SensorInfo` a `DeviceInfo`. Třída `AnalyzerSensor` implementuje návrhový vzor jedináček. Tato třída obsahuje kontejner typu Dictionary, což je vlastně hashovací struktura s klíčem typu řetězec, tj. název senzoru s hodnotou typu `SensorInfo`. Při vytváření této třídy se zavolá systémové volání `GetSensorList` nad instancí třídy `SensorManager`, která vrátí všechny dostupné senzory. Samotné objekty senzorů jsou pak obaleny třídou`SensorInfo` a uloženy do již zmíněného kontejneru pod klíčem složeného ze jména a typu senzoru. Třída `AnalyzerSensor` pak poskytuje ostatním třídám přístup k senzorům pomocí metody `GetSensorInfo`, která je přetížena pro dvě varianty, získání senzoru podle klíče či získaní senzoru podle samotného objektu typu `Sensor`. Dále tato třída obsahuje metodu `SaveSensorInfosToFile`, která zavolá nad všemi instancemi typu SensorInfo v hashované struktuře výpis informací o daném senzoru, který je pak následně uložen do souboru. Samotná třída `SensorInfo` pak zahrnuje dvě metody `GetName` a přepsanou metodu `ToString`. Metoda `GetName`, jejíž argument je typu `Senzor`, vrací klíč senzoru, jak bylo již výše uvedeno, ze jména a typu senzoru. Přepsaná metoda `ToString` pak vrací informace a parametry senzoru ve formě řetězce. Informace a parametry senzorů se mohou lišit verzí operačního systému, proto metoda obsahuje podmínky, které volají jen tu část kódu, která je na daném zařízení podporována. Současně třída `SensorInfo` drží odkaz na soubor pro daný senzor, kde se případně uloží informace o senzoru. Pokud soubor neexistuje, je tento soubor implicitně vytvořen. Poslední třída `DeviceInfo` nesouvisí ani tak se senzory, jedná se pouze o doplňující informace o modelu, verzi operačního systému a výrobci zařízení. Tato data budou sloužit pro statistické vyhodnocení testovacích zařízení.

4.2 Virtuální senzory

Virtuálními senzory jsou uvažovány softwarově implementované senzory, které byly vytvořeny speciálně v rámci této práce. Jednou ze skupin virtuálních senzorů jsou senzory s příponou `VirtualInvariant`, které mají oproti původním senzorům částečně přepočítaný souřadnicový systém vzhledem k Zemi. Přepočet se provádí pomocí funkce `GetRotationMatrix` ze systémové třídy SensorManager, která bere na vstup data z gravitačního senzoru (nebo akcelometru), magnetometru a výstupem je rotační matice. V tomto případě necháme vstup z magnetometru konstantní, protože budeme chtít kompenzovat pouze natočení vůči Zemi, ne vůči pólu. Pokud máme rotační matici, můžeme tuto matici invertovat a vynásobit daný výstup ze senzoru. Dalšími variantami virtuálních senzorů jsou `VirtualGravity` a `VirtualLinearAccelerometer`, což jsou aproximace senzorů gravitace a lineárního akcelerometru pro zařízení, které těmito senzory nedisponují. V případě lineárního akcelerometru je používána horní propust nad akcelerometrem a v případě gravitačního senzoru je používána dolní propust nad akcelerometrem. [11] [4]

4.3 Pomocné třídy `FTPHelper` a `ZIPHelper`

Jak již název napovídá, třídy `FTPHelper` a `ZIPHelper` slouží k odesílání souboru přes FTP protokol a komprimaci souboru do formátu ZIP. Obě třídy obsahují tzv. asynchronní metody, což znamená, že každá takto označená metoda vytvoří instanci třídy `Task`. Podle článku [17] je úloha (Task) abstrakce nad vlákny, funguje tak, že plánovač inteligentně vytvoří optimální počet vláken podle toho, jak je CPU výkonný a kolik má logických procesorů. Těmto vláknům pak přiděluje jednotlivé úlohy. Tento princip má několik výhod. Největší výhoda je ta, že dojde ke 100% využití CPU, další výhoda je, že proces vlákna recykluje a ušetří se tedy nemalá režie při vytváření vlákna.

První třída `FTPHelper` má metody `CreateDirectory` a `UploadFileAsync`. Metoda `CreateDirectory` vytvoří na FTP úložišti nový adresář pro ukládaní dat. Metoda `UploadFileAsync`, což je asynchronní metoda, nahraje na zadanou cestu přes FTP protokol soubor předaný parametrem. Obě metody jsou statické a požadují jako parametry heslo, přihlašovací jméno a URL adresu FTP úložiště. Druhá třída `ZIPHelper` obsahuje rovněž asynchronní metodu `ZipFileAsync`, která bere jako argument seznam souborů a cestu k výstupnímu archivu typu ZIP.

4.4 Pomocné třídy `SettingsHelper` a `StorageHelper`

Třída `SettingsHelper` používá k získávání nastavení uložené do perzistentní paměti (např. FTP heslo, uživatelské jméno, adresa úložiště a nastavení snímání senzorů). K tomu třída používá `PreferenceManager`, který ukládá nastavení do paměti ve formě klíče a hodnoty. Třída `StorageHelper` je pomocná třída pro práci s úložištěm zařízení. Metody `IsExternalStorageWritable` a `IsExternalStorageReadable` detekují, jestli je v zařízení namontováno externí úložiště, do kterého je možno zapisovat či z něho číst. Metoda `DeleteRecursive` zase rekurzivně odstraní všechny adresáře a soubory ze zadaného adresáře. Další metoda `Export` zaregistruje všechny soubory vytvořené aplikací do tzv. MediaScanner služby. Tato služba pak umožnuje zpřístupnit uživateli z počítače soubory z mobilního zařízení. Metoda `GetOutputDir` vrátí odkaz na hlavní adresář, kde budou uloženy data aplikace. Metoda `GetFile` navrací odkaz na soubor zadaného jména z hlav-

ního adresáře, pokud soubor neexistuje, vytvoří nový soubor se zadaným jménem. Metoda `DirectorySize` přijme parametr, tj. cesta k adresáři a vrací jeho velikost v megabajtech. Metoda `AvailableSpace` pak vrací dostupné místo na úložišti zařízení. Následující metody `ReadFile` a `WriteFile` umožňují dočíst soubor ve formě pole řádku a zapisovat do souboru řetězec. Metody `DecodeItem` a `EncodeItem` slouží k dekódování, resp. zakódování uživatelovy položky (uživatelova akce, poloha zařízení) do souboru, ve formátu odděleno středníkem: název položky, čas od, čas do. Poslední metody `SaveItemList` a `GetItemList` používají výše zmíněné metody `DecodeItem` a `EncodeItem` k uložení seznamu položek a ke zkonstruování seznamu položek ze souboru.

4.5 Třída `ScanningValue` a `ScanningValuesBuffer`

Třída `ScanningValue` slouží pro držení jedné nasnímané hodnoty ze senzoru. Obsahuje tři důležité vlastnosti: pole nasnímaných hodnot (velikost pole se liší od typu senzoru), čas měření a čas zpracování. Důvod proč uchovávat oba časy je ten, že čas měření není v jednotném formátu. Každý výrobce mobilních zařízení si tento čas interpretuje jinak, např. někdy je tento čas s datem, někdy bez data atd. Proto je potřeba uchovávat i čas zpracování, který si aplikace ukládá sama. Kromě toho třída `ScanningValue` přepisuje metodu `ToString`, která vrací naformátovaný řetězec obsahující uvedené vlastnosti. Druhá třída `ScanningValuesBuffer` slouží jako pomocná datová struktura, do které jsou ukládány dočasně data snímaná ze senzorů. Jádrem této struktury je kontejner typu `Dictionary` s klíčem, který je složený ze jména a typu senzoru. Hodnoty kontejneru jsou pak typu seznam `ScanningValue`. Nad tímto kontejnerem je realizován zámek z důvodu synchronizace vícevláknového přístupu. Dále třída `ScanningValuesBuffer` obsahuje metodu `AddValue`, kterou používá vlákno typu producent. Parametry této metody jsou senzor a naměřené hodnoty zaobalené do objektu typu `ScanningValue`. Poslední metoda `GetCloneAndClear` vytvoří klon třídy `ScanningValuesBuffer` a vrátí ho jako návratovou hodnotu a vymaže z kontejneru původního objektu data. Tato metoda je používána konzumujícím vláknem.

4.6 Služba `CollectorService`

Tato služba slouží ke sběru dat ze senzorů. Jak již bylo popsáno v podkapitole o technologii Doze a službách, musí tato služba běžet na popředí, aby nedošlo během nečinnosti zařízení k jejímu přerušování. Z tohoto důvodu musí služba vytvořit notifikaci v notifikačním centru tak, aby byl uživatel informován, že služba běží. Krom toho se touto permanentní notifikací může uživatel dostat do uživatelského rozhraní aplikace. Služba `CollectorService` v metodě `OnCreate`, která je zavolána bezprostředně po zkonstruování instance služby, nastaví za pomocí systémové třídy `PowerManager` tzv. wake-up režim na Partial. Dále je potřeba inicializovat událost přepnutí zařízení do úsporného režimu (proč je tento postup nutný je popsáno v přechozích podkapitolách). Událost pak při vyvolání se sekundovým zpožděním znovu zaregistruje snímání hodnot ze senzorů. V metodě `OnStartCommand`, která je volána při spuštění služby, je pak zapnut příslušný mechanizmus bránící přepnutí telefonu do úplného úsporného režimu a poté zaregistrování snímaní hodnot ze senzorů. Návratová hodnota metody `OnStartCommand` je nastavená na konstantu Sticky, která udává, že služba bude v případě jejího ukončení systémem znovu spuštěna. Samotná služba pak pracuje ve dvou vláknech jako konzument a producent. První vlákno, resp. úkol (Task) je jako producent vytvořen při vyvolání události `OnSensorChanged`, při sejmutí nové hodnoty z daného

senzoru. Producent vloží novou hodnotu z daného senzoru do mezi-paměti, která je realizována třídou `ScanningValuesBuffer` pomocí metody `AddValue`. Druhá část konzument je časovač spouštějící obslužné vlákno co sekundu, které vytvoří novou kopii mezi-paměti a vymaže ji pomocí metody `GetCloneAndClear`. Nad touto kopií mezi-paměti se provede agregace dat. Ta probíhá tak, že podle nastaveného počtu vzorků uživatelem za sekundu se rozdělí pole nasnímaných hodnot za sekundu daného senzoru. V rámci takto rozdělených částí pole se provede aritmetický průměr vzorků dané části pole. Tyto aritmetické průměry pak nahrazují samotné vzorky dat. Tím dochází k redukci dat a úspoře místa na zařízení. Výsledek agregace je pak ukládán do souboru pro daný senzor. Tento způsob rozdělení služby na dvě části producent a konzument má nejen výhodu v možné agregaci dat, ale má vliv i na úsporu baterie zařízení snížením zápisu dat do souboru. Intervalem časovače, kdy je konzument spuštěn, lze pak ovlivňovat spotřebu baterie a náročnost na operační paměť zařízení. Dále služba obsahuje metodu `OnDestroy`, která je vyvolána při ukončení služby. V této metodě jsou uvolněny zdroje služby, zejména pak zámek, který blokuje přepnutí zařízení do plného úsporného režimu.

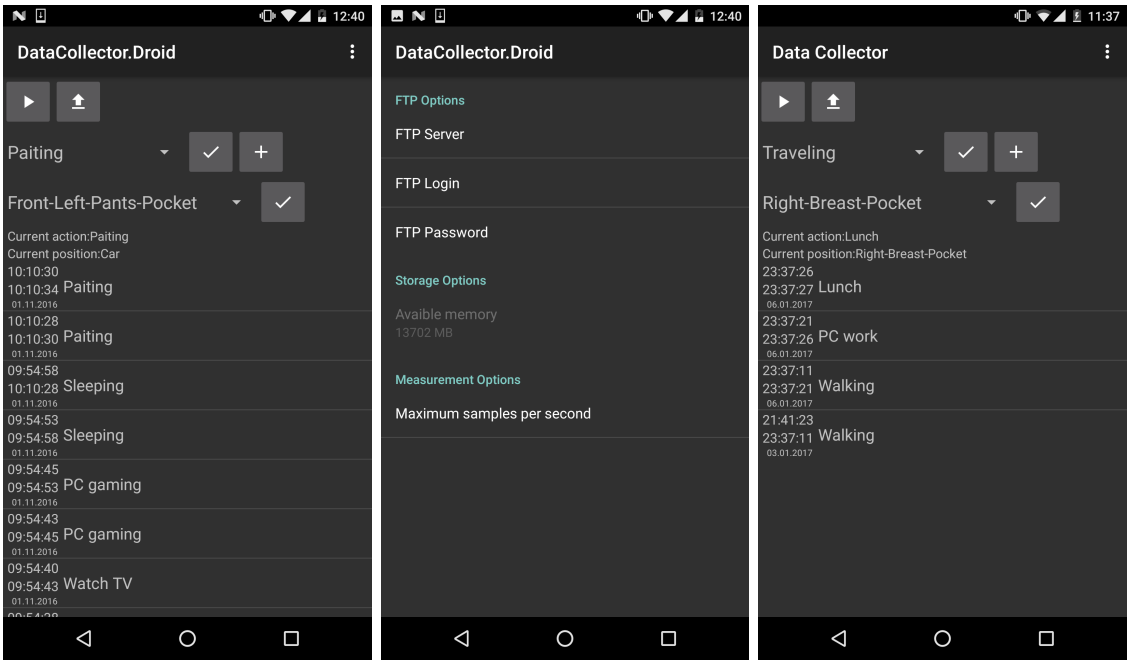
4.7 Služba `BackupService`

Služba `BackupService` slouží k zasílaní dat přes protokol FTP na FTP úložiště. Rovněž tato služba pracuje na popředí a uživatel je o ní informován. Metoda `OnStartCommand` při spuštění této služby zkontroluje, jestli služba `CollectorService` neběží. Pokud tato služba běží je nutno tuto službu pozastavit a nastavit flag, že služba `CollectorService` byla pozastavena, aby po skončení službyBackupService mohla být zase spuštěna. Důvod nutnosti pozastavení služby `CollectorService` je ten, že zapisuje do souborů, které služba `BackupService` bude zálohovat na FTP uložiště. Rovněž služba `BackupService` má nástavnou vlastnost znovuzotavení pomocí návratové hodnoty Sticky metody `OnStartCommand`. Vlastní tělo služby, implementované metodou `DoWork`, nejprve zkontroluje nastavenou uživatelskou aktivitu a polohu zařízení a upraví tento záznam na dva záznamy. Dále se vezmou všechny soubory s daty nasnímanými ze senzorů, jejich informace a parametry, informace o zařízení a záznamy o uživatelských aktivitách a polohy zařízení se zabalí do archivu typu ZIP. K tomuto účelu se využívá třída `ZIPHelper`. Jakmile je vytvořen ZIP archív, je na FTP úložišti vytvořen adresář, jehož název odpovídá unikátnímu identifikátoru zařízení. Tím je docíleno toho, aby zasílaná data uložila správně do složky pro dané zařízení. Po vytvoření adresáře je samotný archív přenesen přes FTP protokol na FTP úložiště. Po přenesení dat jsou data lokálně ze zařízení smazána a pokud běžela služba `CollectorService`, je znovu spuštěna.

4.8 Uživatelské rozhraní

Uživatelské rozhraní lze rozdělit do třech aktivit: `ActionActivity`, `MainActivity` a `SettingsActivity`. Hlavní aktivita `MainActivity` je zobrazena po spuštění aplikace. Rozhraní umožnuje uživateli spustit samotné měření ze senzorů, resp. spuštění služby `CollectorService`. Pokud už služba běží, je zobrazeno hlášení, že je již služba spuštěna. Dále umožnuje synchronizovat data na FTP úložiště. Před synchronizací se zavolá metoda `IsConnectedViaWifi`, která zkontroluje typ připojení a pokud není zařízení připojeno k WiFi síti, zobrazí se uživateli dialog, který ho nabádá k připojení k WiFi. Dále má uživatel možnost zadat uživatelskou aktivitu, a to jak již předdefinovanou, tak novou, kdy je přesmě-

rován do `ActionActivitiy`, kde zadá název nové uživatelské aktivity. Jakmile je nastavená nová uživatelská aktivita, je uložena do souboru předchozí aktivita s časem ukončení v době přepnutí do nové uživatelské aktivity. Tentýž princip je pak u zadávání a vytváření polohy zařízení v době snímaní dat ze senzorů, s tím rozdílem, že zde jsou již položky pouze předem definované. Dalším prvkem rozhraní je pak seznam historie uživatelských aktivit s časem počátku, časem ukončení a datem. Tento seznam je implementován pomocí třídy `ActivityAdapter`, která dědí ze systémové třídy `ArrayAdapter`. Princip adapteru je ten, že se snaží recyklovat UI prvky, tedy jednotlivé UI prvky seznamu jsou vytvořeny a podle pozice seznamu jsou do těchto prvků nahrávány data. Poslední aktivita `SettingsActivity` umožnuje nastavit: uživateli adresu FTP úložiště, přihlašovací jméno, heslo, vzorkovací frekvenci senzorů a možnost kontroly dostupného místa pro uložení dat ze senzorů. Nastavení jsou uložena do perzistentní paměti systému SharePreferences, kde je k nim pak pomocí třídy `SettingsHelper` přístup z ostatních tříd. Realizované rozhraní si lze prohlédnout na obr. 4.1.



Obrázek 4.1: Obrázky jednotlivých obrazovek uživatelského rozhraní.

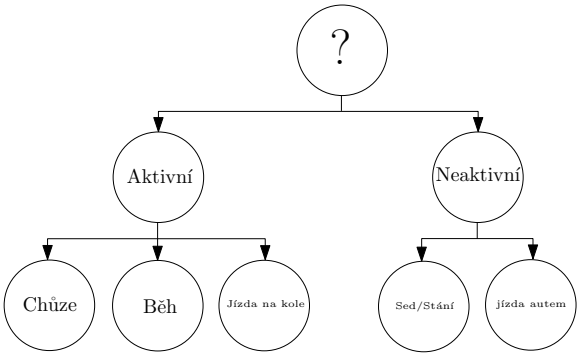
Kapitola 5

Metody detekce aktivit

Problém detekce uživatelských aktivit z dat senzorů lze zjednodušit na problém klasifikace 1D signálů. V anglické literatuře bývá problém detekce uživatelských aktivit označován jako Human Activity Recognition tj. zkráceně HAR. Současné řešení tohoto problému lze rozdělit do několika kategorií, podle toho, zda využívají specializovaný hardware (nejčastěji různé senzory umístěné na končetinách či hrudi), nebo se omezují na data ze senzorů chytrého zařízení a chytrého náramku. Dále lze rozdělit řešení, jestli jsou založené na datech pouze z jednoho senzoru (nejčastěji z akcelerometru), nebo je řešení multimodální a využívá dat z více senzorů. Ve většině publikací je řešení založeno na akcelerometru, gyroskopu případně magnetometru. Z praktického hlediska má smysl se omezit pouze na senzory chytrého zařízení a využit co nejvíce relevantních senzorů z telefonu. [46]

5.1 Nejpoužívanější metody nezaložené na neuronových sítích

Jedna z nejpoužívanějších metod strojového učení v této doméně jsou Rozhodovací stromy, v anglické literatuře označovány jako Decision Tree. Jedno z mnoha řešení využívající Rozhodovací stromy je např.: [44], které nejprve rozdělí aktivity na aktivní (běh, chůze, jízda na kole) či neaktivní (jízda autem, neaktivita) a v druhém kroku predikuje již konkrétní aktivitu uživatele. Nevýhoda Rozhodovacích stromů je obtížná aktualizace modelu (jakmile je model vytvořen je obtížné model aktualizovat o nová trénovací data). Z tohoto důvodu není tato metoda vhodná pro průběžné učení. Obrázek 5.1 nastiňuje způsob řešení. [46]

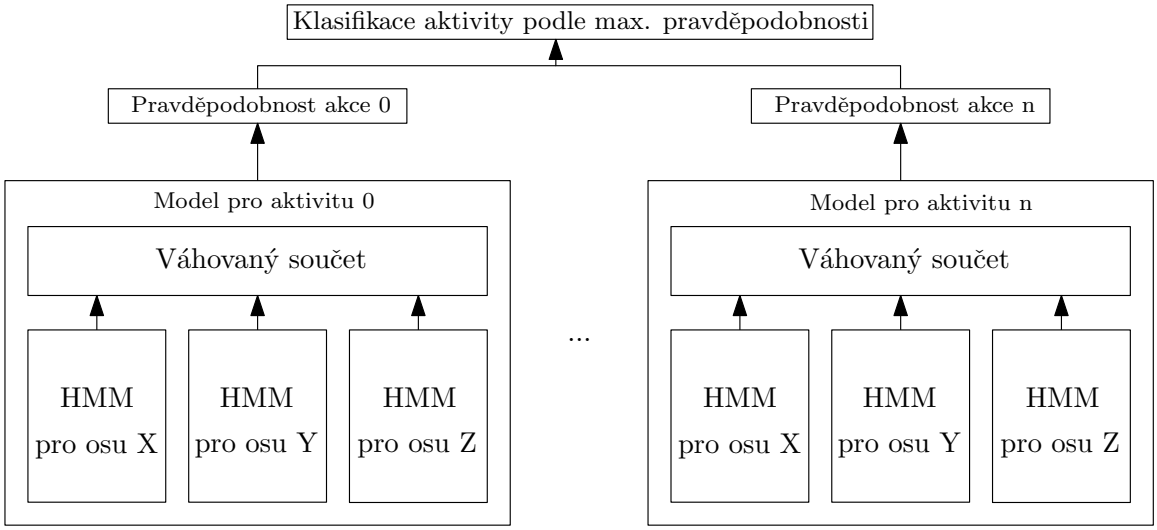


Obrázek 5.1: Hiearchické řešení detekce. Obrázek založen na [44].

Další často používanou metodou je Rozhodovací tabulka, tzv. Decision Table, která obsahuje pravidla a třídy. Pro daný vzorek signálu je pak hledána největší shoda. V porovnání s Rozhodovacími stromy je tato metoda jednoduší na implementaci, avšak ztrácí schopnost hierarchické klasifikace, kdy je nejprve aktivita klasifikovaná jako obecná činnost a poté přesně určena konkrétní aktivita uživatele. [46]

Mezi používané metody se též řadí algoritmus k-nejbližších sousedů, anglicky k-nearest neighbors algorithm, zkráceně K-NN. Tento algoritmus nalezne N nejbližších sousedů k danému vzorku a dle jejich většinové příslušnosti klasifikuje vzorek signálu. [46]

Velmi populární metoda jsou Markovovy skryté modely, anglicky Hidden Markov Models, zkráceně HMM. Nejčastěji se používá hierarchická architektura, kde se nejprve určí, jestli je zařízení v klidu a poté je určena konkrétní aktivita. Pro každou uživatelskou aktivitu pak máme samostatný model a je vybrán model aktivity, jehož skoré je největší. [42] Nebo také jemnější dělení, kdy pro každou uživatelskou aktivitu má každá osa senzoru vlastní model, a skoré dané aktivity je pak váhovaný součet pravděpodobnosti modelů os senzorů viz obr. 5.2. [37]



Obrázek 5.2: Obrázek ukazuje detekci uživatelských aktivit pomoci HMM, kde nejprve jsou separovaně detekovány aktivity na jednotlivých datech z os senzorů a z toho následně vypočítáno skóre pro danou aktivitu. Obrázek čerpá z [37].

Další populární klasifikátor je Support vector machines, zkráceně SVM. Použití této metody je většinou velmi jednoduché, kdy na vstupu je sada příznaků z daného vzorku. Vícetřídní SVM je pak schopen vrátit největší odezvu pro danou třídu. [23]

Ve většině odborných prací, zakládajících své řešení především na: SVM, KNN, HMM či Rozhodovacích stromech, je potřeba nejprve ze signálu získat příznaky. Příznaky můžeme rozdělit do dvou skupin: příznaky v časové doméně, jako např. aritmetický průměr, korelaci mezi osami senzorů, směrodatná odchylka a příznaky ve frekvenční doméně, např. energie ve spektru, spektrální tok a spektrální těžiště. Výčet dalších příznaků je uveden v Tabulce 5.1. [26]

Příznak	Popis	Vzorec
průměr(s)	Aritmetický průměr	$\bar{s} = \frac{1}{N}\sum_{i=1}^{N} s_i$
std(s)	Směrodatná odchylka	$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(s_i - \bar{s})^2}$
mad(s)	Mediánová absolutní odchylka	$median_i(\lvert s_i - median_j(s_j)\rvert)$
max(s)	Maximalní hodnota	$max_i(s_i)$
min(s)	Minimalní hodnota	$min_i(s_i)$
šikmost(s)	Koeficient šikmosti	$E[(\frac{s-\bar{s}}{N})^3]$
špičatost(s)	Koeficient špičatosti	$E[(s-\bar{s})^4]/E[(s-\bar{s})^2]^2$
maxFrekv(s)	Největší frekvenční složka	$argmax_i(s_i)$
výkon(s)	Výkon signálu	$\frac{1}{N}\sum_{i=1}^{N} s_i^2$
entropie(s)	Entropie signálu	$\sum_{i=1}^{N}(c_i log(c_i)), c_i = \frac{s_i}{\sum_{j=1}^{N} s_j}$
iqr(s)	Mezikvartilové rozpětí	$Q3(s) - Q1(s)$
korelace(s_1, s_2)	Pearsonův korelační koeficient	$\frac{COV(s_1,s_2)}{\sigma_1\sigma_2}$
vahPrůměr(s)	Váhovaný průměr signálu	$\frac{\frac{1}{N}\sum_{i=1}^{N} i s_i}{\sum_{j=1}^{N} s_j}$
engPásma(s, a, b)	Energie spektra frekvenčního pásma $[a, b]$	$\frac{1}{a-b+1}\sum_{i=a}^{b} s_i^2$
úhel(s_1, s_2, s_3, v)	Úhel mezi průměrem signálu a vektorem	$tan^{-1}(\lVert[\bar{s_1}, \bar{s_2}, \bar{s_3}] \times v\rVert, [\bar{s_1}, \bar{s_2}, \bar{s_3}] \cdot v)$

Tabulka 5.1: Tabulka znázorňuje nejčastěji používané příznaky pro detekci aktivit, kde N značí velikost vektoru vzorků signálu; Q značí kvartil; s značí vektor diskrétního signálu. Vzorce jsou převzaty z [41].

5.2 Nejpoužívanější metody založené na neuronových sítích

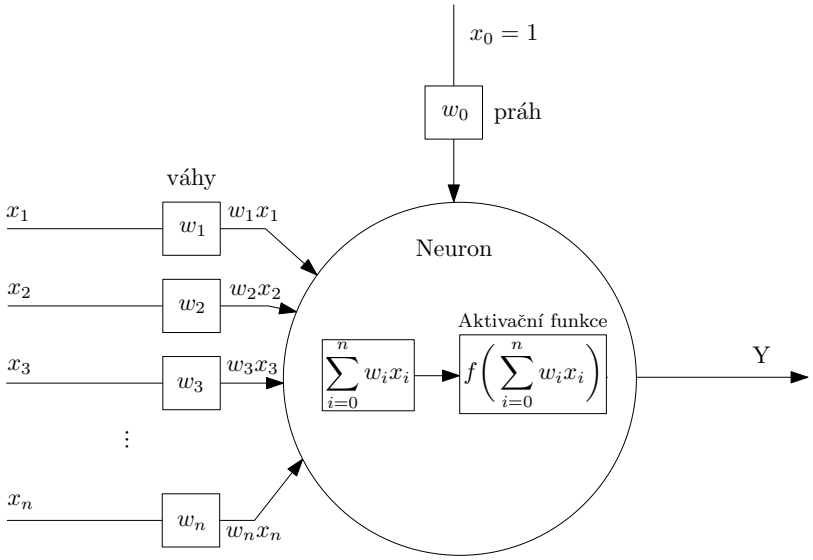
Metody založené na neuronových sítích získaly na popularitě především v posledních dvou letech díky rozmachu frameworků, které implementují neuronové sítě i na mobilní zařízení. Abychom byli schopni jednotlivé přístupy odlišit, vyvodit výhody a nevýhody a vzhledem k založení řešení práce na neuronových sítích, je nutné nejprve vysvětlit teoretické principy a pojmy v dané oblasti.

5.2.1 Neuronové sítě

Neuronové sítě mají za sebou již dlouhou historii. V roce 1943 přišli Warren McCulloch and Walter Pitts s prvním matematickým modelem lidského neuronu v mozku [38]. Na jejich práci později navázal v roce 1958 F. Rosenblatt, který popsal základ všech dnešních neuronových sítí, tzv. Perceptron [43]. V průběhu následujících desetiletí nedošlo v této oblasti k většímu rozvoji. Až v roce 1975 Werbos, P.J. přišel s principem zpětného šíření chyby (Backpropagation) ve vícevrstvých neuronových sítích a tento princip v pozdějších letech odstartoval revoluci [47].

5.2.2 Koncept neuronu

Koncept neuronu je zjednodušený matematický model neuronu v lidském mozku. Základem neuronu jsou axony (vstupy), které vedou signál a přivádí ho na vstup x_n neuronu. Každý vstup x_n má váhu w_n, kterou je násoben. Dále zde figuruje práh (bias) b, což je speciální případ vstupu do neuronu, kde $x_0 = 1$. Suma přes všechny součiny $x_n w_n$ je vstup do tzv. aktivační funkce, kde výstup této funkce je roven výstupu z neuronu. Jako aktivační funkce jsou dnes nejpoužívanější sigmoida či ReLU (Rectified Linear Unit). Na obr. 5.3 je vyobrazen matematický model neuronu. [31]



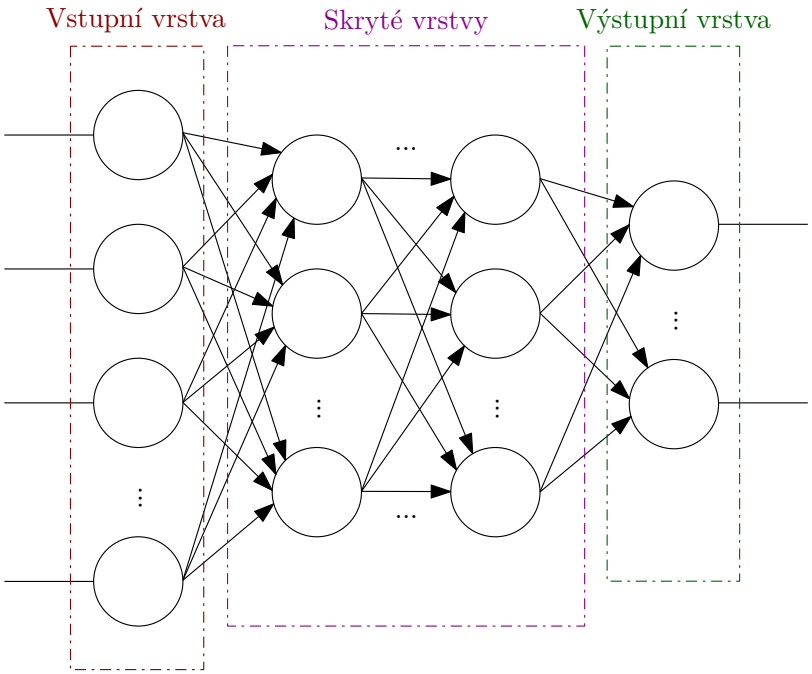
Obrázek 5.3: Obrázek znázorňuje kocept neuronu, kde w_n jsou váhy neuronu, b je práh a f je aktivační funkce. Obrázek založen na předchozím odstavci.

5.2.3 Vícevrstvé dopředné neuronové sítě

Abychom byli schopni klasifikovat i problémy, které nejsou lineárně separovatelné (např. problém XOR [25]), je potřeba definovat vícevrstvou neuronovou síť. Základem takovéto sítě jsou neurony, které jsou organizovány do tzv. vrstev. Síť pak obsahuje tři typy vrstev, a to vstupní, skryté a výstupní vrstvy, kde skrytých vrstev může být až n. Neurony jsou pak napojeny vždy na neurony nadcházející vrstvy, proto přízvisko dopředná. Obecně platí, že čím více vrstev síť má, tím obtížněji taková síť dokáže generalizovat problém. Dopředná neuronová síť je zobrazena na obr. 5.4. [28] [31]

5.2.4 Konvoluční neuronové sítě

Konvoluční neuronové sítě jsou jedna z nejpopulárnějších metod umělé inteligence v současnosti. Jejich síla je v univerzálním použití od zpracování obrazu, dolování dat, až po analýzu řeči či lingvistiku. Mezníkem v této oblasti byl úspěch Alexe Krizhevského [35] z roku 2012, který vytvořil architekturu konvoluční sítě pro klasifikaci obrázků z datasetu ImageNet, později byla tato architektura nazvaná AlexNet. Od té doby začal velký boom konvolučních neuronových sítí.



Obrázek 5.4: Skrytých vrstev může být libovolný počet. Počet výstupních neuronů pak odpovídá počtu tříd klasifikace. Obrázek založen na předchozím odstavci.

Jak již název prozrazuje, základem konvolučních neuronových sítí je konvoluce. Konvoluce dvourozměrných spojitých funkcí f a h (jádro konvoluce) je definovaná následujícím vztahem 5.1. V digitálním obraze se používá diskrétní konvoluce, která je definována analogicky vztahem 5.2. [40]

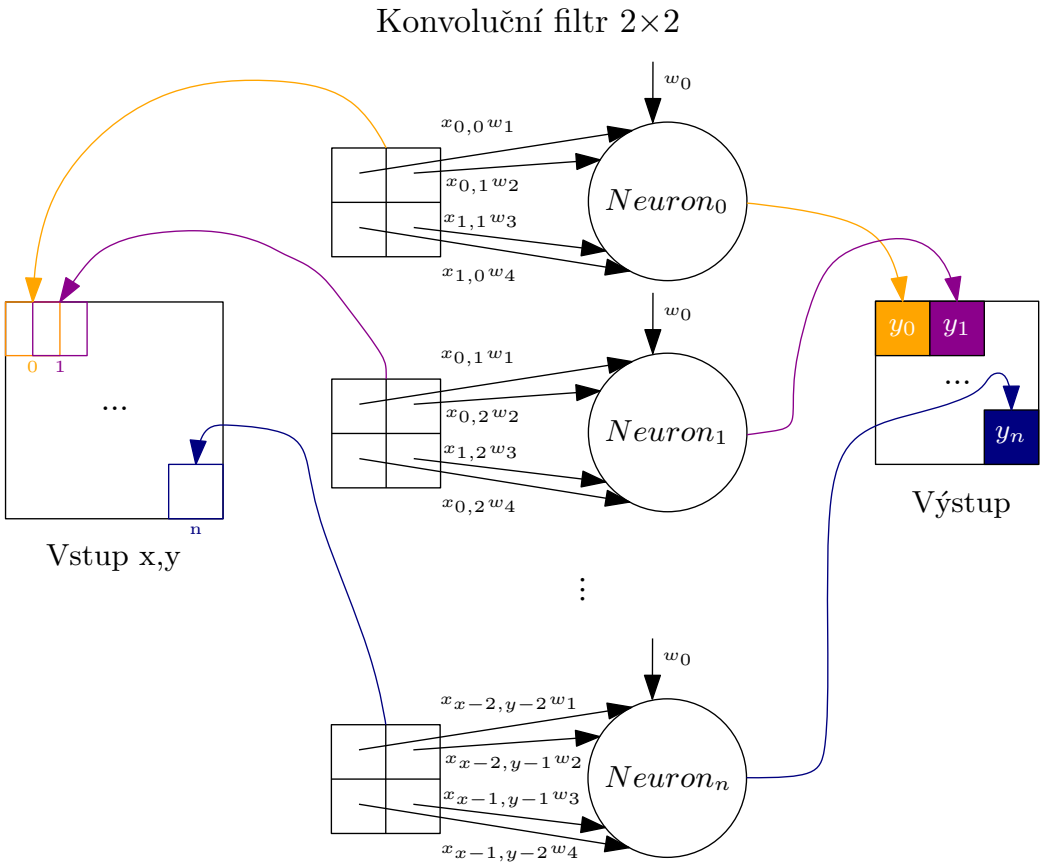
$$f(x,y) * h(x,z) = \int_{\infty}^{\infty} \int_{\infty}^{\infty} f(x-a, y-b)h(a,b)dadb \tag{5.1}$$

$$I(x,y) * h(x,z) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} I(x-i, y-j)h(i,j) \tag{5.2}$$

5.2.5 Vrstvy konvolučních sítí

Konvoluční vrstva implementuje samotnou konvoluci. Základem je tzv. konvoluční filtr (jádro konvoluce) jehož výstupem je příznaková mapa. Velikost filtru je pak rovna počtu vstupů, které jsou připojeny do neuronu, tedy nejedná se o plně propojenou síť. Počet neuronů je roven počtu podoblastí vygenerovaných pomocí konvoluce, např. pro vstupní jednokanálový obrázek 25x25 při velikosti filtru 4x4 a posuv (stride) o 1, obsahuje filtr 22x22 neuronů. Neurony mají v rámci všech podoblastí sdílené váhy a práh, což má za následek dvě výhody. Zaprvé se tím významně redukuje počet parametrů, které jsou konstantní a nezávislé na velikosti vstupu a zadruhé se hledaný příznak definovaný filtrem může nacházet na různých pozicích vstupu. Pokud máme např. velikost filtru 5x5, je pak: 25 + 1 práh = 26 parametrů. Každá vrstva může obsahovat více filtrů, tedy výstupem je pak více příznakových map. Princip konvoluční vrstvy demonstruje obr. 5.5. [28] [31]

Velmi důležitou vrstvou v konvolučních neuronových sítích je tzv. Pooling vrstva. Tato vrstva zajišťuje invarianci vůči menším posunutí vstupu. Dále tato vrstva může agregovat

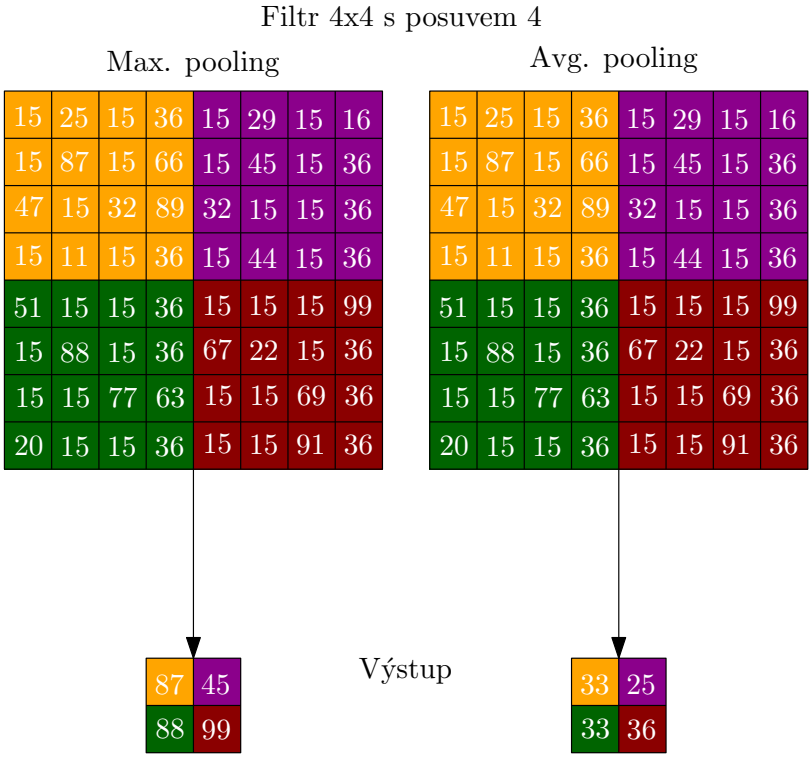


Obrázek 5.5: Obrázek znázorňuje abstrakní pojetí kovnolučního filtru, váhy všech neuronů jsou sdílené. Obrázek založen na předchozím odstavci.

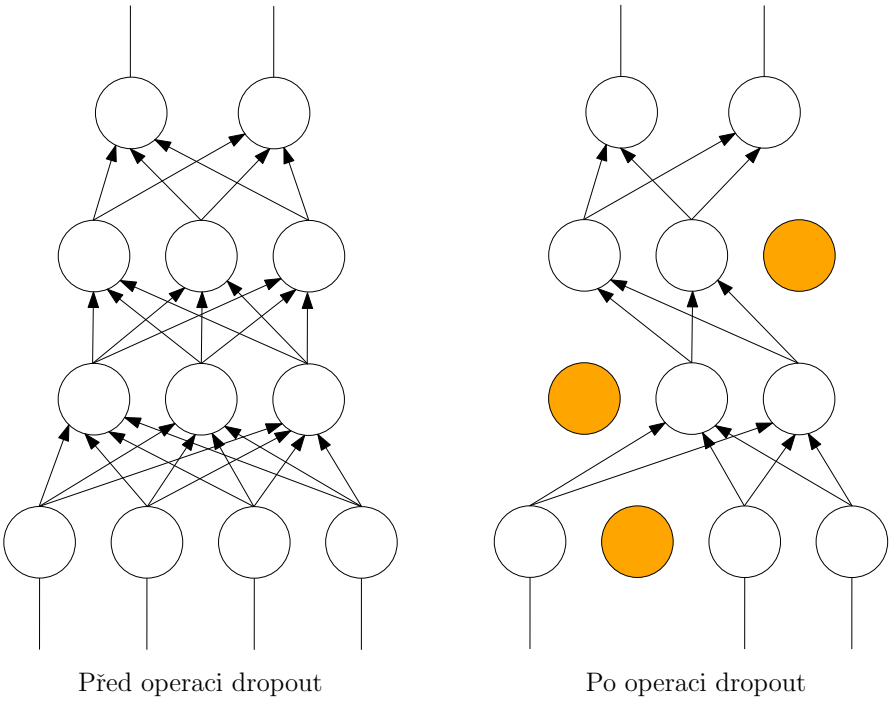
vstup a tím zmenšovat výstup, tj. provádět podvzorkování, anglicky downsampling. To umožnuje efektivnější výpočet v následující vrstvách. Princip této vrstvy je jednoduchý, s nějakým okolím projde vstup a vždy vybere hodnotu v okolí, která je maximální, tj. tzv. Max-Pooling nebo spočítá průměrnou hodnotu okolí, tzv. Avg-Pooling. Oba principy jsou vyobrazeny na obr. 5.6 [28]

Mezi další vrstvy patří také tzv. plně propojená vrstva, která na rozdíl od dříve popisovaných vrstev neslouží k extrakci příznaků, ale ke klasifikaci příznakového vektoru do požadovaných tříd. Jedná se tedy o klasickou dopřednou vícevrstvou neuronovou síť. [31]

Někdy se také uvádí jako vrstva tzv. Dropout. Jde o regularizační techniku, jak předcházet nebo alespoň redukovat přeučení neuronových sítí. Problémem při učení neuronových sítí je fakt, že síť si začne postupným učením pamatovat trénovací vzorky. To má za následek ztrátu generalizace řešení a tedy i velmi špatnou přesnost klasifikace na validační sadě, kde se mohou nacházet jemně či více odlišné vzorky oproti vzorkům v trénovací sadě. S tímto problémem lze bojovat tak, že při trénování síť zapomene část informace, resp. část neuronů je vypnuta a neuronová síť se tak nemůže spoléhat na svou paměť a naučí se lépe generalizovat problém. Obr. 5.7 ukazuje neuronovou síť před operací droupout a po operaci droupout. [45]



Obrázek 5.6: Vpravo můžeme vidět tzv. max. pooling vrstvu, vlevo tzv. avg. vrstvu. Obrázek založen na předchozím odstavci.



Obrázek 5.7: Operace dropout, která část neuronů sítě deaktivuje. Obrázek založen na předchozím odstavci.

Další zajímavou vrstvou je tzv. bach normalizační vrstva, která normalizuje vstupní dávku dat tak, aby průměr byl blízko nule a směrodatná odchylka se blížila k hodnotě 1. Tato technika potom obecně urychluje trénování, trénovaní není tak závislé na inicializaci a síť dosahuje vyšší přesnosti. Obvykle se tato vrstva používá před aktivační funkcí. [32]

Jako poslední vrstva v neuronových sítích bývá tzv. Softmax vrstva. Softmax je normalizovaná exponenciální funkce sloužící k tomu, abychom z výstupních neuronu dostali pravděpodobnost příslušnosti k jednotlivým třídám. Dle [24] lze libovolný k-rozměrný vektor (aktivace výstupní vrstvy neuronů) z převést na stejně rozměrný vektor, kde složky vektoru se pohybují v intervalu $< 0, 1 >$. Tento vektor lze interpretovat jako pravděpodobnost příslušnosti daného vzorku k jednotlivým třídám, což odpovídá vztahu 5.3.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}; j \in 1...K \tag{5.3}$$

5.2.6 1D konvoluční neuronové sítě

Všechny popisované pojmy, zejména vrstvy, byly popsány na 2D vstupních datech (nejčastěji obrázku). Avšak konvoluční neuronové sítě lze přenést i do problému klasifikace 1D vstupních dat (např. časové řady). Tedy konvoluci zjednodušíme následovně 5.4 a máme zde pouze 1D okolí. Všechny další pojmy jsou totožné nebo analogické jako u 2D. [28]

$$I(x) * h(x) = \sum_{i=-k}^{k} I(x-i)h(i) \tag{5.4}$$

5.2.7 Princip učení neuronových sítích

Asi nejpodstatnější část v oblasti neuronových sítí je princip jejich učení. Proto, abychom mohli nějaký model vůbec učit, je potřeba nějakým způsobem změřit, jak moc chybný výstup daný model vrací. K tomuto se v oblasti neuronových sítí používá tzv. loss funkce, neboli funkce chyby. Čím vrací funkce menší hodnoty, tím je přesnější výstup z neuronové sítě. Velmi často je tato funkce implementovaná jako tzv. křížová entropie, anglicky cross-entropy 5.5, kde p požadovaný výstup sítě a q je aktualní výstup sítě pro třídu x [3]. Pokud máme metriku pro vypočítání chyby sítě, nastává problém, jakým způsobem aktualizovat váhy neuronu, abychom zmenšili chybu sítě. Nejprve je potřeba spočítat gradienty ukazující, jak se chyba vyvíjí v hyperprostoru. K tomuto se používá již zmíněný algoritmus zpětného šíření chyby. Tato informace je pak využita optimalizačním algoritmem, který se na základě gradientu chyb snaží co nejlépe aktualizovat váhy neuronů, resp. minimalizovat funkci chyby. Zde se nejčastěji používají dva algoritmy: Stochastic Gradient Descent (SGD) a Adaptive moment estimation (Adam). Oba algoritmy používají koeficient učení nebo tzv. learning rate, ten říká jako moc velká aktualizace vah sítě proběhne v rámci jedné iterace učení. Základní rozdíl mezi SGD a Adam pak je, že původní SGD má konstantní koeficient učení pro všechny váhy, resp. parametry modelu, stejný, kdežto metoda Adam odděluje koeficienty učení pro každý parametr modelu, kde koeficient učení se během trénovaní mění. Obecně platí, že v průběhu učení je vhodné koeficient učení snižovat. V čím pokročilejším stádiu učení sítě je, tím menší váhu by měl nový trénovací vzorek mít. Analogii můžeme najít v případě lidského mozku, kdy mozek má velkou schopnost učit se v nižším věku a postupem času převládájí zkušenosti nad novými poznatky. [27] [33]

$$H(p,q) = -\sum_{x} p(x) log(q(x)) \tag{5.5}$$

5.2.8 Metody detekce aktivit neuronovými sítěmi

Jedna z prvních metod je založena na vícevrstvých neuronových sítích, kde vstup do sítě jsou příznaky, které jsou extrahovány ze signálu senzorů. [49]

Další třídou metod, které klasifikují uživatelskou aktivitu pomocí konvolučních neuronových sítí, jsou metody založené na 1D konvolučních sítích. U této třídy metod je vstupní vektor časové okno diskrétního signálu a složka tohoto vektoru (časový vzorek) je buď euklidovská norma os senzoru 5.6 (jako v článku [36]), sada příznaků, nebo jsou vstupem surová data ze senzorů. V [51] je srovnání klasifikací uživatelských aktivit, kdy vstupem jsou statistické příznaky, příznaky získané PCA analýzou, příznaky získané pomocí konvoluční vrstvy se sdílenými vahami a příznaky konvoluční vrstvy bez sdílených vah. Experiment jasně ukázal, že konvoluční vrstva se sdílenými váhami dokázala najít nejlepší příznaky, na kterých pak plně propojená neuronová síť dokázala nejpřesněji klasifikovat aktivitu. Další variací, jak řešit daný problém pomoci 1D konvoluční sítě, je způsob organizace dat z různých senzorů ve vstupním vektoru. Jednou z možností je přidávat senzory jako kanály vstupního vektoru [50]. Např. pokud máme dva senzory a každý má tři osy x, y, z, vstupní vektor vytvoříme jako 6-kanálový. Konvoluce pak prochází všech 6 kanálu najednou, což nemusí být vždy žádoucí, jelikož data z nesouvisejících senzorů jsou společně agregována. Další možností je pouze 1-kanálový vstupní vektor, kde rozměr x je čas a rozměr y jsou osy senzorů [30] nebo 3-kanálový vektor (pro tři osy) kde osa y jsou jednotlivé senzory.

$$\| x \| = \sqrt{x_1^2 + ... + x_n^2} \tag{5.6}$$

Dalším způsobem, jak řešit klasifikaci aktivit je pomocí 2D konvolučních sítí. Konvoluční filtr prochází data z více senzorů a senzory, které spolu nikterak nesouvisí, jsou odděleny výplní nul o velikosti konvolučního filtru. Tento princip je zvláště vhodný, pokud máme data z více senzorů stejného typu. [29]

5.3 Srovnání metod

Klasické metody jsou v drtivé většině založeny na extrakci příznaků, které mohou být odolnější vůči natočení senzorů. Naproti tomu metody založeny na konvolučních neuronových sítích si dokáží najít příznaky pro daný problém samy a tím zvýšit úspěšnost klasifikace. Je však potřeba velký dataset, kde jsou data ze zařízení nasnímaná v různých polohách senzorů, aby nalezené příznaky byly co nejvíce invariantní vůči natočení zařízení. Vzhledem ke zvyšujícímu se výkonu chytrých zařízení a dostupnosti frameworků, které neuronové sítě implementují efektivně i na mobilních zařízeních, není dnes problém tyto metody použít pro klasifikaci uživatelských aktivit v reálném čase přímo na chytrém zařízení. Pokud srovnáme metody založené na 1D konvoluci a 2D konvoluci, nemá smysl použít 2D konvoluční sítě na mobilním zařízení, jelikož neobsahuje více senzorů stejného typu a tedy benefit 2D konvoluce se zde ztrácí. Nehledě na to, že 2D konvoluční neuronové sítě obsahují více parametrů, modely pak zabírají více místa a jsou vůči 1D konvolučním sítím pomalejší.

Kapitola 6

Implementace a návrh učení

Nosnou části této práce je návrh a implementace učení neuronové sítě pro detekci uživatelských aktivit. Druhou, neméně důležitou částí je program pro předzpracovaní nasbíraných dat pro účel trénovaní a samotná implementace knihovny pro platformu *Xamarin.Android*, která umožní jednoduchou integraci detektoru aktivit do cílové aplikace.

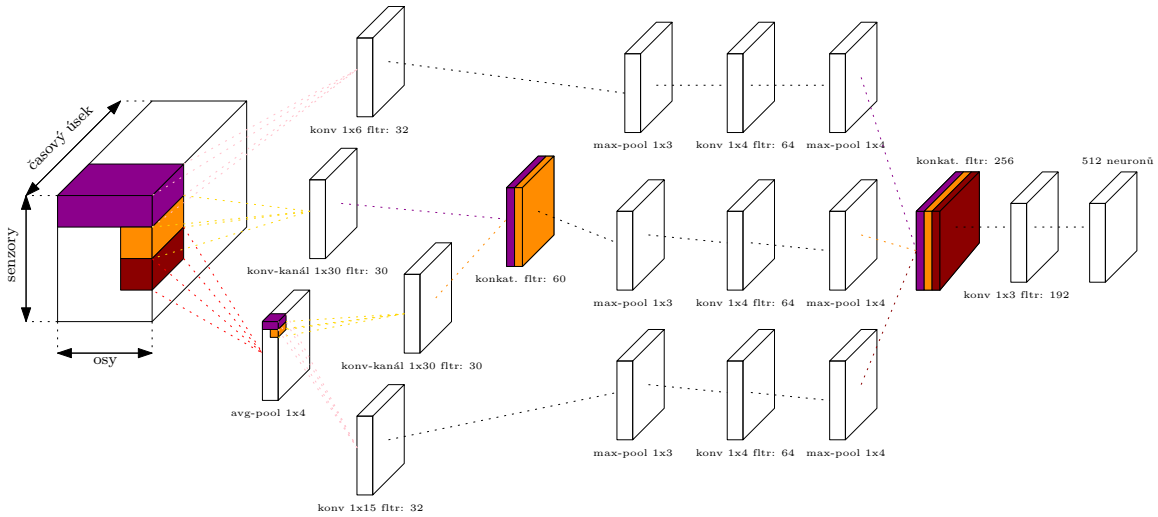
6.1 Návrh konvoluční sítě

Většina existujících řešení založených na neuronových sítích je omezena (nebo je evaluována na omezených datasetech [2]) buď z hlediska typu senzorů (některé senzory nemusí být na mobilních zařízeních realizovatelné, nebo používají více senzorů jednoho typu na různých končetinách), nebo jsou zařízení či senzory umístěny napevno vůči končetinám, což odpovídá laboratorním podmínkám. V běžném provozu je mobilní zařízení volně umístěno např. v kapse kalhot, bundy nebo v ruce. Z tohoto důvodu nelze očekávat, že model bude mít v reálných podmínkách stejnou úspěšnost jako na datasetu, který neodpovídá těmto podmínkám. Proto je lepší navrhnout vlastní síť, která bude lepé uzpůsobena reálným podmínkám. Navržena síť pak bude porovnána s referenční implementaci 1-D konvoluční sítě (obr. 6.2) vycházející z již publikované práce [29].

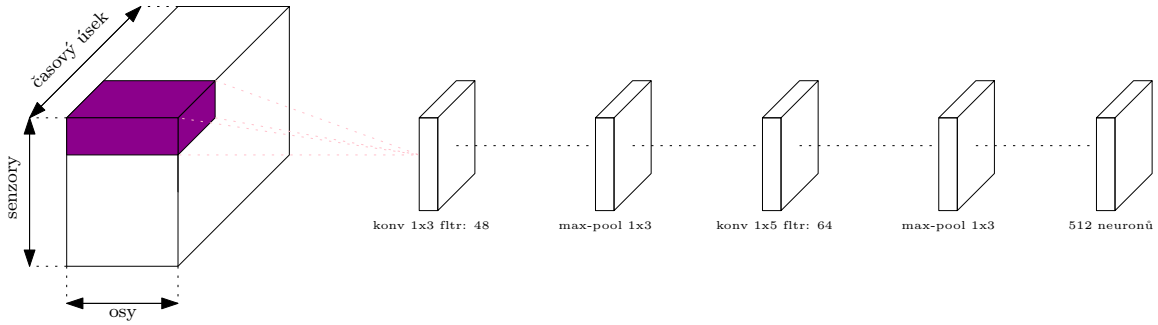
Návrh konvoluční neuronové sítě vychází z 1-D sítě, topologie sítě byla odvozena od výchozí sítě, ke které byly postupně přidávány další vrstvy. Mezivýsledky byly vždy validovány na datasetu, který vznikl v rámci této práce. Výsledný návrh sítě je pak empiricky odvozený model, který dávává nejlepší výsledek na validační sadě.

Obě varianty uvedených sítí používají stejnou organizaci vstupních dat, kde senzory jsou brány jako výška, osy senzorů jako kanály a časové vzorky signálu jako šířka vstupu. Počet kanálu je roven třem, přičemž senzory, které nemají tři osy (např. barometr), mají zbylé dvě osy doplněny nulovými hodnotami pro zarovnání vstupního tenzoru. Navržená síť (obr. 6.1) se skládá ze třech paralelních samostatných větví. První větev nejprve aplikuje na vstupní tenzor Avg-pooling, který aproximuje dolní propust, tedy odstraní ze signálu šum. Na výstup z této vrstvy se aplikují dva typy konvolucí. Konvoluce přes všechny kanály a konvoluce napříč kanály vstupního tenzoru. Tyto dva odlišné přístupy pak hledají příznaky buď v rámci všech os senzorů, tj. konvoluce přes všechny kanály, anebo příznaky pouze na samostatných osách, tj. konvoluce napříč kanály. Druhá větev provádí konvoluci napříč kanály přímo na vstupním tenzoru. V druhé fázi se konkatenuje výstup z druhé větve s částí mezivýsledků první větve. Třetí větev pak provádí konvolucí přes všechny osy na vstupním tenzoru. Všechny větve obsahují trojici bloků: Max-pooling, konvoluci a následně další Max-

pooling operaci. Větve jsou konkatenovány do jednoho vstupu koncové konvoluce. Poslední blok je pak plně propojená vrstva, která obsahuje 512 skrytých neuronů. V této sítí se používají velké konvoluce zejména na výstup z avg-pooling operace (dolní propust), kde menší konvoluce nemá smysl aplikovat.



Obrázek 6.1: Znázornění návrhu topologie sítě, kde blok nejvíce vlevo představuje vstup a blok nejvíce napravo představuje výstup sítě.



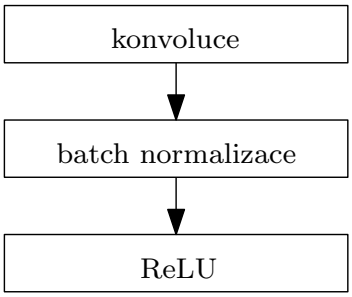
Obrázek 6.2: Zobrazení topologie referenční sítě, která vychází z [29].

V navrhované sítí každý konvoluční blok reprezentuje trojicí bloků, tj. samotné konvoluce, batch normalizace a aktivační funkce ReLU. Blok je vyobrazen na obr. 6.3.

6.1.1 TensorFlow

TensorFlow je komplexní framework pro vývoj a výzkum různých typů či obměn neuronových síti. Za tímto frameworkem stojí společnost Google, která je na poli hlubokého učení v současné době lídr. Samotný framework umožnuje trénování na procesorech, grafických kartách NVIDIA s podporou technologie CUDA či speciálních jednotkách, které implementují neuronové sítě hardwarově (TPU). [19]

Síla tohoto frameworku je zejména v jeho obecnosti. Celý systém je postaven na tom, že na vše se lze dívat jako na tenzory, nad kterými lze provádět operace. Celý systém je organizován do grafu, což umožnuje vytvářet různé komplexní řešení. Zároveň framework poskytuje několik úrovní, pro práci s neuronovými sítěmi: nízkoúrovňové API, vysokoúrovňové



Obrázek 6.3: Schéma konvolučního bloku použitého v navrhované sítí.

API (Estimator API a další) a Keras API. V případě, že programátor používá nízkoúrovňové API, musí vše obstarat sám, zejména inicializaci, vytvoření trénovacích proměnných, detailní propojení tensorů a organizace do vrstev. V tomto případě má programátor úplnou kontrolu nad tvorbu všech vrstev konvoluční neuronové sítě a také umožnuje tvorbu speciálních a komplexních vrstev. Vysokoúrovňové API je pak abstrakcí nad nízkoúroňovou vrstvou, kde jsou předpřipravené nejčastější používané vrstvy. Speciální možnost je Keras API, kde TensorFlow slouží jako implementace tohoto API. Keras je vhodný zejména pro výukové použití. Framework podporuje různé programovací jazyky: Python, C++, Java a Go. Základ je pak napsán C++ a NVIDA CUDA tak, aby poskytoval co nejefektivnější výpočty, práci s maticemi a aby trénování bylo co možná nejvýkonnější, dle možností stroje. Jednou z velkých výhod, zejména pro tuto práci, jejíž cílové řešení je pro mobilní platformy, je odlehčená verze frameworku pro inferenci natrénovaného modelu na mobilních zařízeních s operačními systémy Android a iOS tak, aby mohl provádět detekci v reálném čase.[20] [18]

6.1.2 Implementace trénování a samotné sítě

Pro implementaci bylo použito nízkoúrovňové API frameworku TensorFlow v programovacím jazyce Python. Samotná implementace se skládá ze dvou částí:

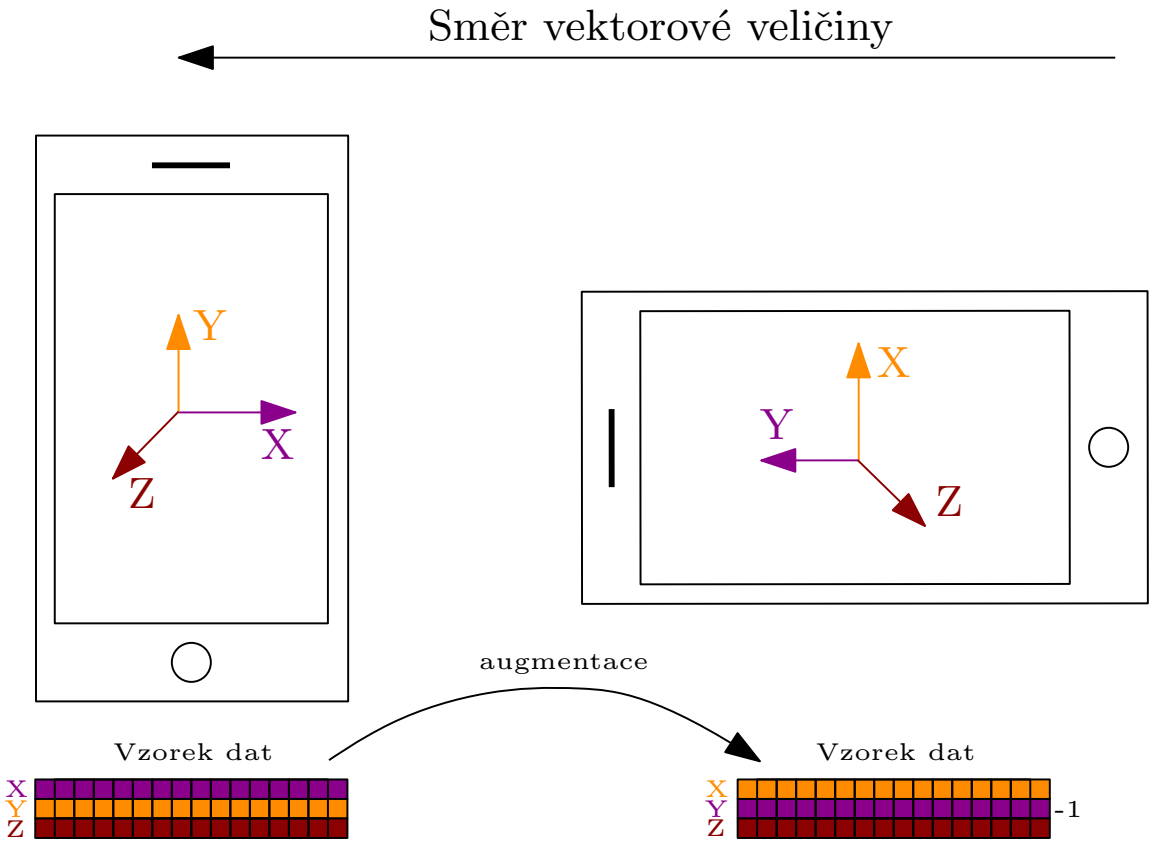
- implementace samotné sítě a její trénování
- třída pro načtení dat

V části implementace neuronové sítě je sada funkcí pro vytvoření konkrétního typu vrstvy a funkce pro vytvoření vah (`weight_variable`) a prahů (`bias_variable`). Všechny trénovací proměnné jsou inicializovány pseudonáhodným generátorem, který je inicializován předem nastavenou hodnotou (seed). To je velmi důležité proto, abychom zajistili deterministické výsledky a mohli tak porovnat výkon trénovaného modelu. Funkce `batch_norm` implementuje batch normalizaci, kde má tato vrstva dva režimy: jeden při trénování, kdy se parametry batch normalizace učí a druhý režim při inferenci sítě, kdy se parametry normalizace nemění. Další důležitá funkce je funkce conv, která implementuje konvoluční vrstvu přes všechny kanály. Jinou variantou je pak `depthwise_conv`, která provádí konvoluci napříč kanály jednotlivě. Poté zde jsou funkce `max_pool` a `avg_pool`, které realizují Max-pooling a Avg-pooling. Následující funkce `full_connected` implementuje plně propojenou vrstvu, která se kombinuje s funkcí dropout, která náhodně vypíná neurony plně propojené vrstvy. Funkce `join_branches`, která konkatenuje vstupy, takže řadí kanály vstupů za se-

bou. Všechny funkce ukládají reference na vstupní/výstupní vrstvy a trénovací proměnné do pole typu slovník (dictionary) pro pozdější rekonstrukci neuronové sítě pro účel exportu.

Samotný graf navrhované sítě je realizován ve funkci build, která vygeneruje graf sítě, připojí ho na vstup a vrátí poslední uzel grafu jako výstup. Trénování sítě se provádí v hlavních částech programu. Nejprve je potřeba vytvořit relaci v TensorFlow pomoci `tf.Session`, v rámci relace se pak načte dataset pomocí třídy data a konstanty z konfiguračního souboru (šířka vstupu, výška vstupu, počet klasifikovaných tříd). Na výstup z grafu se v relaci následně připojuje výpočet křížové entropie pro ztrátovou funkci (loss funkce), optimalizátor Adam (pro prohledávaní stavového prostoru vah), výpočet přesnosti klasifikace dané dávky (batch) a konstrukce tzv. confusion matrix, matice, která zobrazuje počet chyb klasifikace či správnou klasifikaci vztaženou ke třídám. Dále je zde i logika pro načtení uloženého trénování sítě a následné pokračování v trénování. V hlavní smyčce probíhá samotné trénování, kde jsou kontrolní výpisy na standartní výstup (přesnost na validační sadě, přesnost na trénovací sadě, hodnoty loss funkcí a confusion matice), které jsou vypisovány v různých intervalech dle nastavení uživatele. Dalšími funkcemi jsou načtení aktuální dávky trénovacích dat a samotný proces trénování pomoci evaluace uzlů grafu zejména uzlu optimalizátoru. Další důležitou části je export natrénovaného modelu pro další použití na mobilních zařízeních. V tomto případě je nejprve potřeba vytáhnout všechny natrénované parametry z grafu (váhy, prahy atd.), což probíhá pomocí procházení pole network, kde jsou uloženy všechny parametry a posloupnost vrstev. Poté nastává druhá fáze, kdy je graf sítě znovu sestaven dle pole network a jsou z něho odstraněny uzly, které se při inferenci modelu neuplatní (např. dropout). Exportovaný graf je otestován na validační sadě a měl by dosahovat stejné přesnosti jako graf použitý při trénovaní. Skript má pak možnosti nastavení koeficientu učení, počet epoch trénování (doba za kterou se vystřídají všechny trénovací vzorky organizovány do dávky, která je použitá pro jednu iteraci učení), možnost pokračovat v učení sítě od posledního bodu, počet vzorků v dávce a velikost poměru zapomenutých neuronů pomoci operaci dropout.

Druhou části je třída pro načtení datasetu a operace nad ním. Funkce `read_data_sets` nejprve načte konfigurační soubory k datasetu a vyčte potřebné konstanty. Pomocí třídy Dataset jsou poskytovány trénovací a validační data trénovacímu programu. Třída Dataset obsahuje metodu `next_batch`, která při začátku každé epochy náhodně promíchá data a vytvoří dávky. Při každém jejím volání pak vrací aktuální dávku. Dále je zde metoda `next_batch_with_balance`, která funguje podobně, jako předchozí metoda s tím rozdílem, že v dávce jsou ve stejném poměru zastoupeny všechny třídy. To může mít velký vliv na přesnost v případě, že nemáme stejný počet vzorků pro všechny uživatelské aktivity, zejména aktivity dlouhodobé (chůze) vs. krátkodobé (vstání ze židle). Další zajímavou metodou je `augmentation_data`, která jak již název napovídá zavádí augmentaci dat. Myšlenka vychází z toho, že mobilní zařízení (např. v kapse nebo v ruce) může být v různých polohách natočení. Pokud vezmeme vstupní vzorek a zaměníme pořadí os u všech senzorů stejně a případně vynásobíme časové vzorky mínus jedničkou, čímž můžeme simulovat otočení mobilního zařízení. Jedná se tak o prosté násobení matice rotace s maticí hodnot senzoru s krokem o 90 stupňů. Samozřejmě, že tento princip má smysl aplikovat pouze na senzory měřící vektorové veličiny, což není např. barometr, který měří atmosférický tlak. Díky tomuto principu můžeme významně zvýšit trénovací množinu dat a tím zlepšit výkon trénovaného modelu. Princip je znázorněn na obr. 6.4. Augmentace může být použita při generování každé dávky trénovacích dat.



Obrázek 6.4: Vyobrazení simulace otočení telefonu podle osy Z o 90 stupňů, kde naměřená data z osy X jsou přesunuta do osy Y a vynásobená mínus jedničkou (podobně naměřená data z osy Y přesunuta do osy X).

6.2 Implementace předzpracování dat

Proto, abychom mohli trénovat neuronovou síť, je potřeba vytvořit dataset. Pro vytvoření datasetu jsou potřeba nasbíraná data, která je potřeba nějakým způsobem předzpracovat. Implementace sběru dat byla již popsaná v předchozích kapitolách, přičemž aplikace už částečně data předzpracovávala tak, že agregovala data na požadovaný počet vzorků za sekundu. Výstupem je ZIP balíček, ve kterém je pro každý senzor soubor s daty a časovými razítky, dle toho, kdy byla data pořízena. Vstupem programu pro předzpracování dat je tedy množina ZIP souborů (jeden soubor na jedno měření) s daty. Dále je vstupem programu konfigurační soubor obsahující:

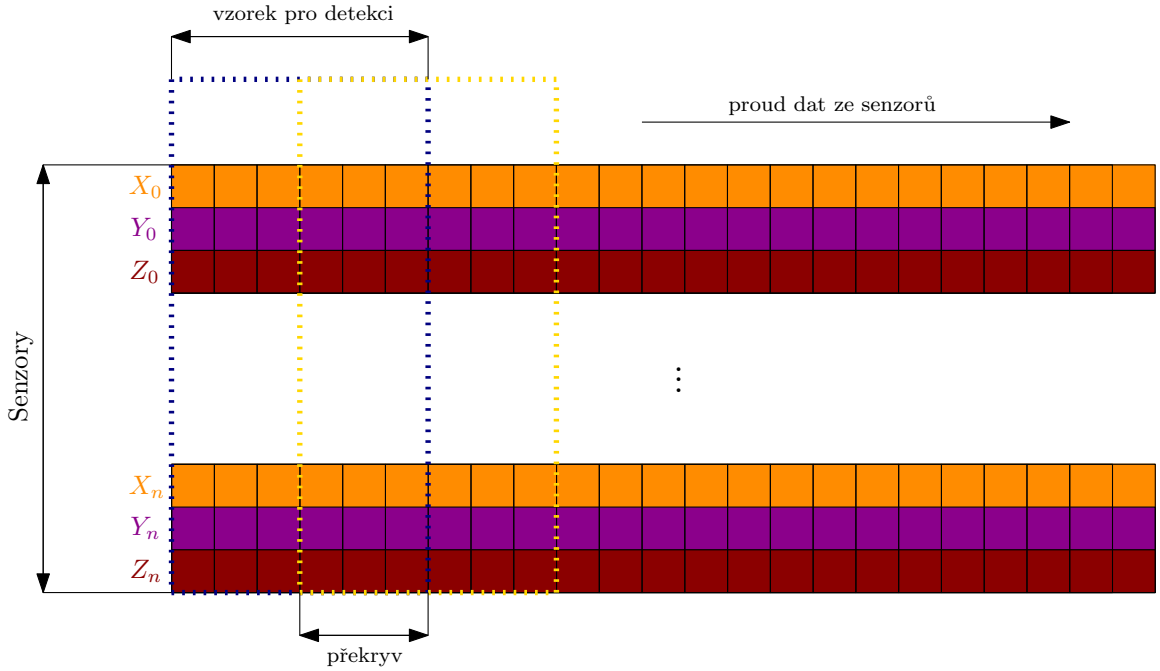
- počet vzorků za sekundu
- délku časového okna (z jak velkého časového úseku bude detektor detekovat uživatelskou činnost)
- překryv vzorků (jaká část vzorku bude použita v následujícím vzorku)
- cesty k dalším konfiguračním souborům

Mezi další konfigurační soubory patří: soubor se seznamem senzorů, které program použije pro tvorbu datasetu, konfigurace knihovny pro detekci (seznam a pojmenování tříd, politiky detekce) a konfigurační soubor s nastavením ořezu intervalů aktivit.

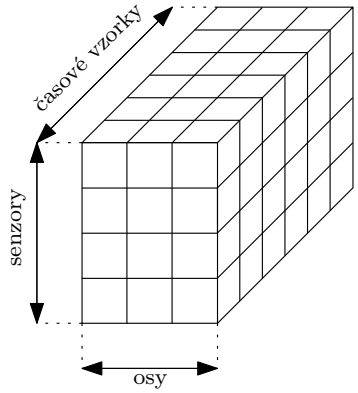
Mezi hlavní funkce programu pro předzpracovaní dat patří samotná tvorba datasetu dle konfiguračních souborů, čištění dat, doplnění chybějících dat a normalizace dat. Všechny tyto činnosti obstarává třída `Parser`, která nejprve načte všechny ZIP balíčky pomocí metody parse, kde dekóduje soubory CSV (Comma Separated Values) s daty pro každý senzor pomocí třídy `CsvReader`. Metoda `parse` vytvoří několik entit. Jedna z entit je třída `Activity`, která reprezentuje uživatelskou aktivitu. V rámci aktivity pak mohou být intervaly s informací, kdy aktivita probíhala (třída `ActivityInterval`). Analogické třídy `Position` a `PositionInterval` dělají totéž, ale pro pozice mobilního zařízení (např. v ruce, kapse atd.). Velmi důležitou entitou je třída `Sensor`, která zapouzdřuje všechny informace o daném senzoru (např. typ senzoru, jméno senzoru a podobně). V této entitě je dekódovaný samotný proud dat ze senzoru, který je reprezentován polem objektů třídy `Sample`, kde jsou k jednotlivým vzorkům přiřazeny pozice a uživatelská aktivita, ke které patří. Dále třída `Sensor` provádí pro senzory barometru a magnetometru rozdíl mezi sousedními vzorky, jelikož pro detekci je vhodnější sledovat změnu těchto veličin, ne jejich absolutní hodnotu. V případě magnetometru se jedná o další virtuální senzor, který může být použit pro detekci.

Nejzásadnější metoda pro hlavní část předzpracování dat je metoda `Export` v třídě `Parser`. Metoda nejprve provede čištění dat, kdy pro každý interval uživatelské aktivity smaže část vzorků na začátku a na konci intervalu. Tato funkcionalita je zde, protože při měření aktivity měřící subjekty (dobrovolníci sbírající dataset) mohou provést nedefinovanou aktivitu na začátku a konci měření (např. zapnou měření dané aktivity a mezi tím dají mobilní zařízení do kapsy). Pomocí konfiguračního souboru můžeme definovat pro každou aktivitu v dané pozici zařízení čas na začátku a konci intervalu, jehož vzorky nebudou zahrnuty do datasetu (např. pokud je aktivita měřená v kapse kalhot, je tento čas vhodné nastavit vyšší, než u aktivity měřené v ruce). Další důležitou části je normalizace dat, zvlášť důležitá pro neuronové sítě, které jsou už z principu velmi citlivé na velké hodnoty ve vstupních datech. Pokud bychom měli např. ve vstupním vektoru data z různých senzorů, kde rozsahy hodnot se podle typu senzorů významně liší, přebyly by příznaky s vyššími hodnotami příznaky s nižšími hodnotami, jelikož mají větší „energii“ v neuronové síti. Normalizace se provádí pomocí tzv. z-score, často nazývaným jako standartní skoré. Ze vztahu 6.1 podle [34] je z-skoré definováno jako podíl rozdílu hodnoty x a průměrné hodnoty μ se směrodatnou odchylkou σ. Pro každou osu senzoru je potřeba vypočítat směrodatnou odchylku a střední hodnotu, obě hodnoty pak pro každou osu uložit do souboru pro pozdější použití detektorem. Poté, co máme mezivýpočty pro normalizaci dat, může metoda Export projít proud dat ze senzoru pomocí tzv. klouzajícího okénka (sliding window) a vytvořit jednotlivé vstupní tenzory do neuronové sítě. Princip je vyobrazen na obr. 6.5. Při tomto procházení může dojít k problémům, kdy vzorky je potřeba synchronizovat s časovými razítky napříč všemi senzory. Když vzorky některého senzoru pro daný okamžik (definovaný časovým razítkem vzorků ostatních senzorů) chybí, je to problém, který může být způsoben opožděním senzoru nebo nižší vzorkovací frekvencí senzoru. V takovém případě je nutné chybějící data doplnit, a to buď následující hodnotou, nebo (pokud nejsou žádné data v okolí k dispozici) doplnit vzorky nulovými hodnotami tak, aby byl tensor zarovnán. Vstupní tensor je vyobrazen na obr. 6.6

$$z = \frac{x - \mu}{\sigma} \tag{6.1}$$



Obrázek 6.5: Vyobrazení postupu zpracovávaní dat pro tvorbu datasetu. Na obrázku je vidět tzv. klouzající okénko s 50 % překryvem vzorku.



Obrázek 6.6: Vstupní tensor do neuronové sítě.

Poté, co jsou vytvořeny vzorky, můžeme tyto vzorky normalizovat pomocí předvypočítaných hodnot směrodatné odchylky a aritmetického průměru dle vztahu 6.1. Metoda `Export` funguje ve dvou režimech: pro tvorbu trénovacího datasetu (kdy se spočítají mezihodnoty pro normalizaci a uloží do souboru) a pro tvorbu validačního datasetu (kdy jsou mezihodnoty pro normalizaci načtené ze souborů). Výsledkem celého programu pro předzpracovaní dat jsou soubory: `settings.zip` (kde jsou uloženy soubory pro nastavení knihovny a pro detekci aktivit), `train.out/test.out` (trénovací tenzory, resp. validační tenzory), `train-labels.out/test-label.out` (třídy k jednotlivým vstupním tenzorům).

Kapitola 7

Implementace a návrh knihovny pro detekci

Základem knihovny pro detekci uživatelské aktivity je framework TensorFlow Mobile, který umožnuje spouštět vytvořený graf (např. konvoluční neuronová síť) v TensorFlow na mobilních zařízeních s operačním systémem Android a iOS. Proto, abychom mohli použít tento framework na platformě Xamarin.Android, je potřeba nejprve obalit samotnou knihovnu TensorFlow ve formátu AAR (Android Archive Library) do Xamarin knihovny pro operační systém Android a zkompilovat jako LibraryProjectZip. Tato knihovna je pak již v klasickém formátu DLL (Dynamic-Link Library).

7.1 Implementace knihovny pro detekci aktivit

Samotnou detekci na cílovém prostředí zajištuje knihovna pro platformu Xamarin.Android, kterou uživatel integruje do své aplikace a pomocí jednoduchých funkcí dostává detekci uživatelských aktivit, na které může reagovat. Proto, aby knihovna mohla fungovat, je potřeba do cílové aplikace (do tzv. Assets) nahrát soubor settings.zip (konfigurace detektoru) a natrénovaný model.

7.1.1 Služba `DetectionService`

Tato služba, která běží na popředí, obstarává celou detekci aktivity. Tak, jako v případě služby pro sběr dat, i zde běží pro každý senzor samostatné vlákno, kde probíhá sběr dat ze senzorů do paměti (producent). Hlavní vlákno je realizované časovačem vyprazdňujícím tuto paměť a zpracovávajícím tyto data (konzument). V hlavním vláknu tedy probíhá agregace dat, tj. zajištění dosažení požadovaného počtu vzorků za sekundu (pokud senzor produkuje více vzorků, než má, jsou jeho vzorky přepočítany na požadovanou vzorkovací frekvenci). V další fázi jsou data normalizována, pomocí načtených souborů směrodatné odchylky a střední hodnoty pro každou osu senzoru z konfiguračního souboru `settings.zip`. Třída `AggregateValueStack` z naměřených vzorků ze senzorů vytváří vstupní tenzor do neuronové sítě. Prakticky provádí tu samou činnost, jako program pro předzpracování dat, s tím rozdílem, že hlídá, jaká data už nejsou aktuální (podle délky okénka, které si lze nastavit v parametrech detektoru) a tyto data maže. Hlavní vlákno se tedy probudí za předem nastavitelnou dobu (jeden z parametrů nastavení detektoru) a zavolá funkci `AggregateValues`, která aktuální data ze senzoru agreguje a vytáhne si pomocí třídy `AggregateValueStack` aktuální vstupní tenzor, který je klasifikován pomocí přetrénovaného modelu. Samotný

model je interpretován pomocí překompilované knihovny TensorFlow pro platformu Xamarin.Android.

Další zajímavou třídou, kterou služba `DetectionService` využívá, je systémová třída `LocalBroadcastManager`. Tato třída slouží k zasílaní zpráv do hlavních vláken, které mohou obsluhovat uživatelské rozhraní aplikace. Toho je využito, pokud cílová aplikace potřebuje dostávat zprávy o aktuální aktivitě uživatele a zároveň měnit uživatelské rozhraní.

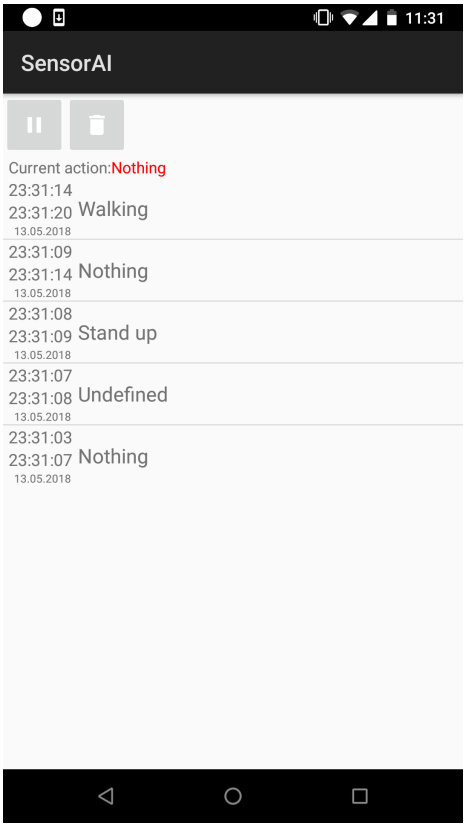
7.1.2 Třída `ActivityDetector`

Samotná služba `DetectionService` má interní viditelnost a nelze ji tedy použít mimo tuto knihovnu. Pro samotné rozhraní knihovny slouží třída `ActivityDetector`. Tato třída umožňuje programátorovi cílové aplikace používat tuto knihovnu. Třída obsahuje čtyři události pro obsluhu detekce aktivit. První událost, na níž může programátor napsat obsluhu, je `ChangedActivity`. Tato událost se vyvolá v případě, že uživatel změnil aktivitu (např. z chůze do běhu) a jako parametr vrací instanci třídy `ActivityResult`, která obsahuje jméno aktuální uživatelské aktivity a pravděpodobnost jakou si je detektor jist touto klasifikací. Další událost `DetectActivity` se vyvolá při spuštění detektoru, který se spouští periodicky (hlavní vlákno v službě `DetectionService`). Parametrem události je pole instancí třídy`ActivityResult`, kde jsou ohodnoceny všechny naučené uživatelské aktivity pravděpodobností detekce. Krom těchto dvou událostí jsou zde analogické varianty `ChangedActivityInBackground` a `DetectActivityInBackground` s tím rozdílem, že předchozí události byly vyvolávaný v kontextu vlákna cílové aplikace, a tedy programátor z nich může měnit uživatelské rozhraní. U variant na pozadí jsou události vyvolány v kontextu volajícího, tj. služba `DetectionService`. Pokud programátor v cílové aplikaci zavolá metodu `EnableEvents`, která zaregistruje pro aplikaci `BroadcastReceiver`, který dostává zprávy, z již zmíněného `LocalBroadcastManager`. Tímto mechanizmem je zaručeno, že události `ChangedActivity` a `DetectActivity` jsou vyvolány ve vlákně cílové aplikace. Avšak, pokud není aplikace aktivní, resp. na popředí (přesněji Android aktivita cílové aplikace), je potřeba `BroadcastReceiver` deaktivovat pomocí metody `DisableEvents`. Z toho plyne, že události jsou vyvolávány, pokud je aplikace aktivní. Nicméně programátor může chtít dostávat události i když aplikace není na popředí (např. uživatel používá jinou aplikaci) nebo dokonce pokud je mobilní zařízení v režimu spánku. Z tohoto důvodu jsou v knihovně události `ChangedActivityInBackground` a `DetectActivityInBackground`. Dále obsahuje třída `DetectionService` zásobník historie detekovaných uživatelských aktivit, kdy cílová aplikace nemusí vůbec běžet a stačí, když běží služba `DetectionService`, která ukládá do zásobníku detekované aktivity. Programátor si pak může kdykoliv data ze zásobníku vytáhnout. Velkou výhodou pak je fakt, že data v zásobníku jsou ukládaná do souboru (tzn. data jsou uložena v zařízení perzistentně). Tatáž třída `ActivityDetector` dělá nad výsledky z detektoru finální analýzu. Programátor si může pomocí konfiguračního souboru detektoru definovat pro každou třídu různé prahy přípustnosti s jakou jistotou je detekce dané třídy brána v potaz. Dále je možné určit, kolik detekovaných vzorků musí být v souslednosti, aby daná třída byla detekována (např. při jízdě v MHD je potřeba tři detekované vzorky za sebou, aby detektor určil s jistotou, že se nejedná o chybnou detekci).

7.1.3 Testovací aplikace

V rámci tvorby knihovny vznikla testovací aplikace, využívající tuto knihovnu. Aplikace může sloužit jako výukový materiál pro pochopení funkce knihovny a její použití v rámci

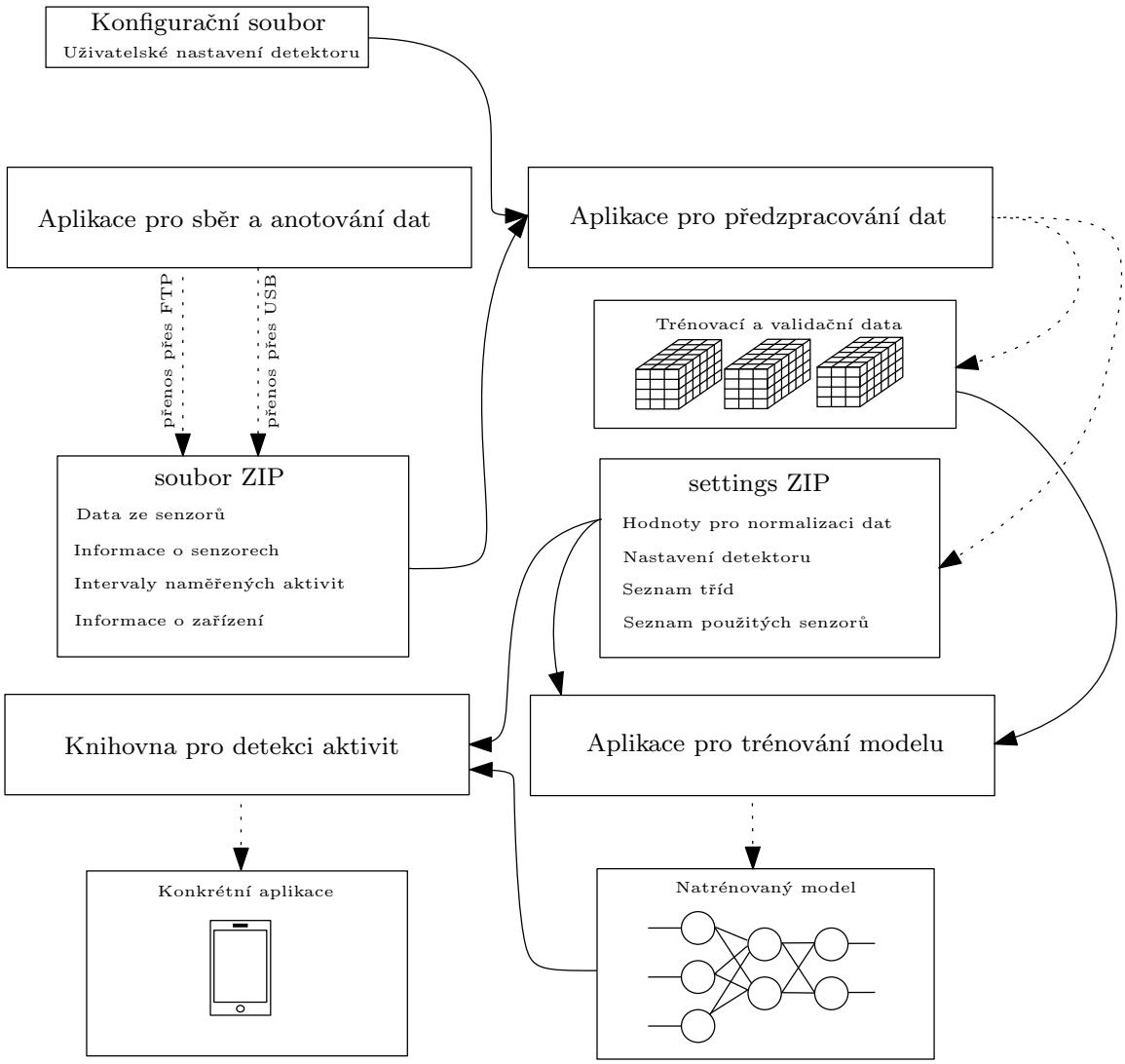
cílové aplikace. Testovací aplikace demonstruje jednoduchý příklad, v kterém aplikace ukazuje uživateli jeho aktuální aktivitu. Zároveň vypisuje aktuální stav zásobníku uživatelských aktivit. Na obr. 7.1. je vidět obrazovka testovací aplikace.



Obrázek 7.1: Obrazovka testovací aplikace.

7.1.4 Framework pro detekci aktivit

Spojením všech implementací této práce vzniká ucelený framework pro detekci uživatelské aktivity na mobilním zařízení, v kterém jsou obsaženy všechny fáze: sběr dat, předzpracování dat, model a jeho trénování a knihovna pro použití na koncovém zařízení. Zároveň je framework vysoce flexibilní a umožnuje programátorovi nasbírat data pro konkrétní oblast použití, definovat si vlastní třídy, vybrat si z široké palety podporovaných senzorů a přizpůsobit si detektor pomoci konfiguračního souboru. Jelikož je framework založen na TensorFlow grafu, není problém ani zcela změnit model klasifikátoru (nejen konvoluční neurnovou síť) pokud bude zachován stejný formát vstupu a výstupu z grafu. Celý framework je vyobrazen na obr. 7.2.



Obrázek 7.2: Schéma frameworku pro detekci uživatelských aktivit na mobilním zařízení.

Kapitola 8

Experimenty

V této kapitole budou popsány některé zajímavé experimenty. Mezi ně patří: Porovnání jednotlivých senzorů a výběr nejlepší kombinace senzorů. V rámci těchto experimentů bude vždy porovnán referenční model s navrženým modelem.

8.1 Nasbíraný dataset

Abychom mohli provádět experimenty, je potřeba získat dataset. Jak již bylo popsáno v předchozí kapitole, většina existujících datasetů nezohledňuje reálné podmínky (senzory z mobilního zařízení, reálné umístění zařízení vzhledem ke končetinám a natočení samotného zařízení), omezuje se na příliš málo senzorů či data jsou již předzpracovaná ve formě příznaků, což omezuje samotnou detekci a analýzu. Proto bylo nutné vytvořit vlastní dataset, abychom zohlednili reálné podmínky v běžném provozu.

Na tvorbě datasetu se podíleli čtyři dobrovolníci: dvě ženy (věk 24 let a 49 let), dva muži (věk 25 let a 57 let). Měření probíhalo na dvou zařízeních s velkou škálou senzorů Nexus 5 a Nexus 5X, které byly zapůjčeny dobrovolníkům. Aktivity a pozice, které byly naměřeny, shrnuje tabulka 8.1. Data z hromadných dopravních prostředků byla nasbírána v Brně, Praze, Vídni a Ostravě. Všechna data byla nasbírána v reálném prostředí bez opasků nebo jiných fixačních pomůcek, které by mobilní zařízení uchytily na pevno ke končetinám. Samotný dataset je rozdělen na dvě části: validační data (89%) a trénovací data (11%). Ty dvě části byly sbíraný odděleně zcela samostatného měření.

Uživatelské aktivity	Pozice zařízení
Běh	Zařízení v pravé kapse kalhot
Chůze	Zařízení v levé kapse kalhot
Chůze po schodech na horu	Zařízení v levé kapse bundy
Chůze po schodech na dolů	Zařízení v pravé kapse bundy
Vstanutí ze židle, pohovky..	Zařízení v ruce na výšku
Sednutí na židli, pohovky...	Zařízení v ruce na šírku
Tramvaj	
Autobus	
Zařízení v klidu	

Tabulka 8.1: Seznam uživatelských aktivit a pozic zařízení naměřené v datasetu.

8.2 Stanovení parametrů pro experimenty

Pro provádění experimentů je potřeba nastavit parametry učení sítě, délka okna detekce a překryv okna. U všech experimentů je použita délka okna detektoru dvě sekundy, jelikož aktivity jako např. sednutí na židli má trvání cca dvě sekundy, a tedy delší časová okna nemá cenu uvažovat. Překryv oken byl pak empiricky zvolen 50 %. V rámci experimentů bylo z datasetu odstraněno část dat ze tříd Tramvaj a Bus, protože pokud autobus či tramvaj nebyly v pohybu data byla označena jako tyto třídy přitom měly být označeny třídou zařízení v klidu. Odstranění proběhlo spuštěním trénování na 10 epoch, kdy takto natrénovany klasifikátor byl použit pro odstranění těchto chybných dat. Dále byly empiricky stanoveny parametry učení neuronové sítě tak, aby sítě dosahovaly co nejlepších výsledků:

- velikost dávky 64 (batch size)
- dropout 60 %
- učící koeficient 0.0001

Základní metrikou, která je použita ve všech grafech je přesnost (precision), která je spočítaná podle jednoduchého vztahu 8.1 podle [39], kde TP je *true positive* a FP *false positive*. Celková přesnost je pak aritmetický průměr všech přesností přes všechny třídy. Vyhodnocení experimentu probíhá nad validační sadou, tj. data, které nebyly použity při trénování.

$$p = \frac{TP}{TP + FP} \tag{8.1}$$

V experimentech jsou senzory označeny zkratkou dle tabulky 8.2.

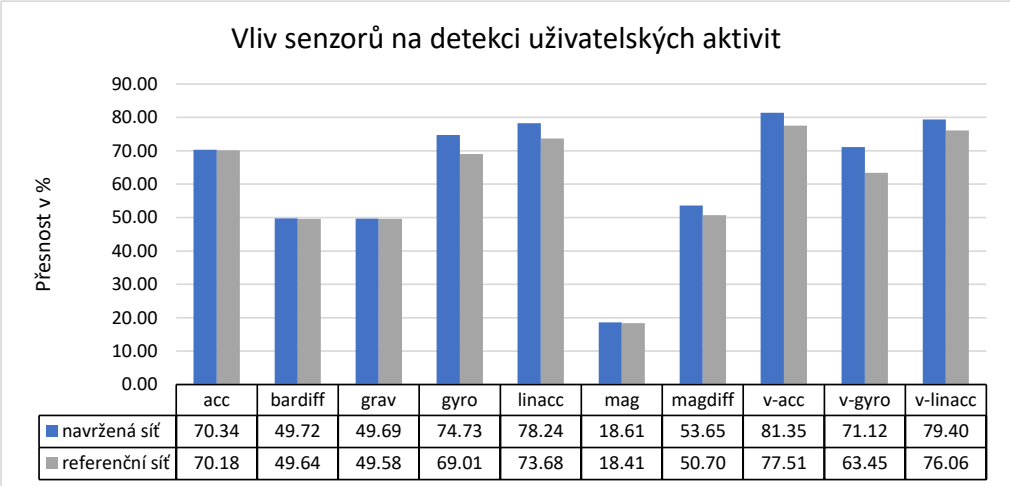
Zkratka senzoru	Cely název
acc	akcelerometr
bardiff	změna tlaku (barometr)
grav	gravitační senzor
gyro	gyroskop
linacc	linearní akcelerometr
mag	magnetometr
magdiff	změna magnetického pole (magnetometr)
v-acc	virtualní akcelerometr (viz. kapitola 4.2)
v-linacc	virtualní linearní akcelerometr (viz. kapitola 4.2)
v-gyro	virtualní linearní gyroskop (viz. kapitola 4.2)

Tabulka 8.2: Seznam zkratek senzorů.

8.3 Porovnání jednotlivých senzorů

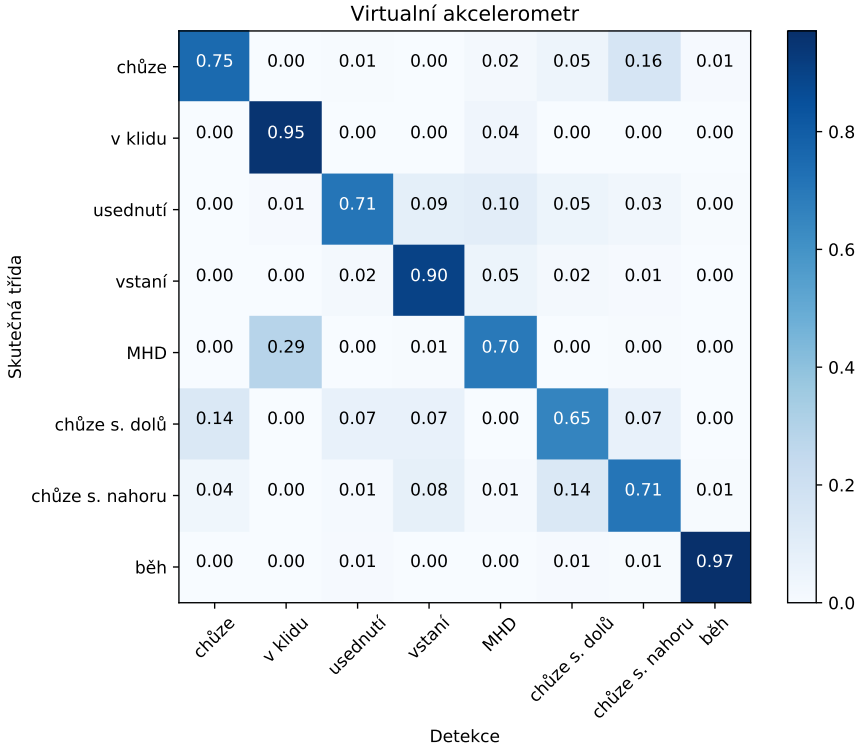
Cílem tohoto experimentu bylo porovnání závislostí jednotlivých senzorů na úspěšnost detekce uživatelské aktivity. Pro každý senzor je natrénován navržený a referenční model, kdy trénování probíhalo po dobu 20 epoch. Na obr. 8.1 jde vidět graf, kde je vyhodnocená přesnost pro jednotlivé senzory. Z grafu vyplývá, že navržená síť překonala referenční síť na

všech senzorech. Jako senzory s nejlepší úspěšnosti pak jsou: virtuální akcelerometr, virtuální lineární akcelometr a lineární akcelometr. Na obr. 8.2 a obr. 8.3 je vidět tzv. confusion matrix nejúspěšnějšího senzoru, resp. nejméně úspěšného senzoru (magnetometr). Dobrý výsledek u linearního akcelerometru je očekavatelný, jelikož zrychlení obsahuje nejvíc informací o uživatelské aktivitě. Naopak magnetometr měřící sílu magnetického pole není vhodný pro detekci, daleko vhodnější se ukazuje změna magnetického pole.

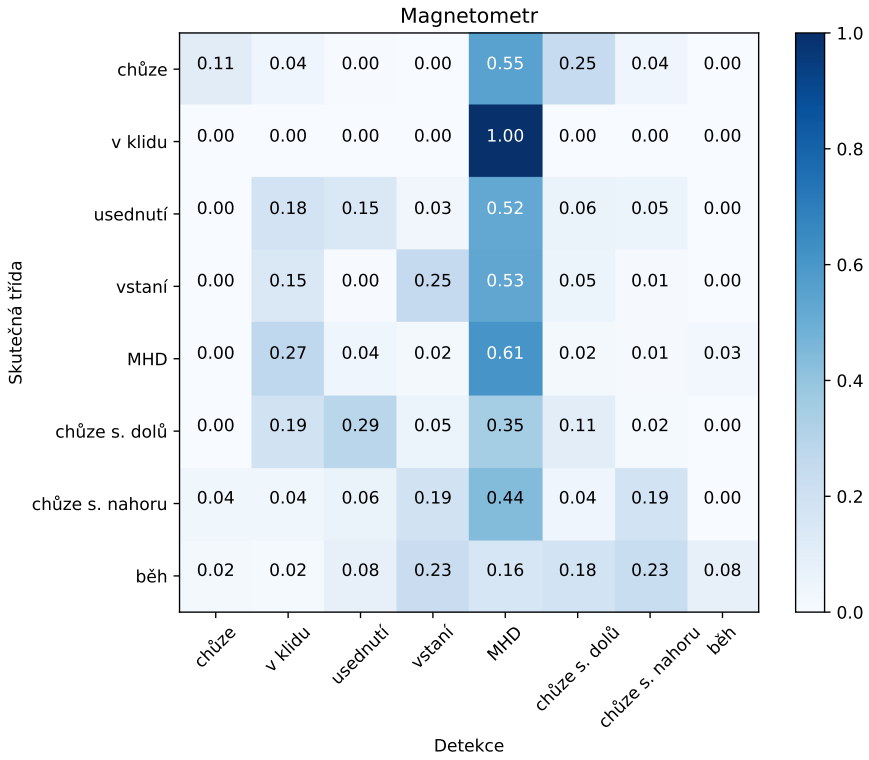


	acc	bardiff	grav	gyro	linacc	mag	magdiff	v-acc	v-gyro	v-linacc
navržená síť	70.34	49.72	49.69	74.73	78.24	18.61	53.65	81.35	71.12	79.40
referenční síť	70.18	49.64	49.58	69.01	73.68	18.41	50.70	77.51	63.45	76.06

Obrázek 8.1: Graf ukazující výsledek experimentu.



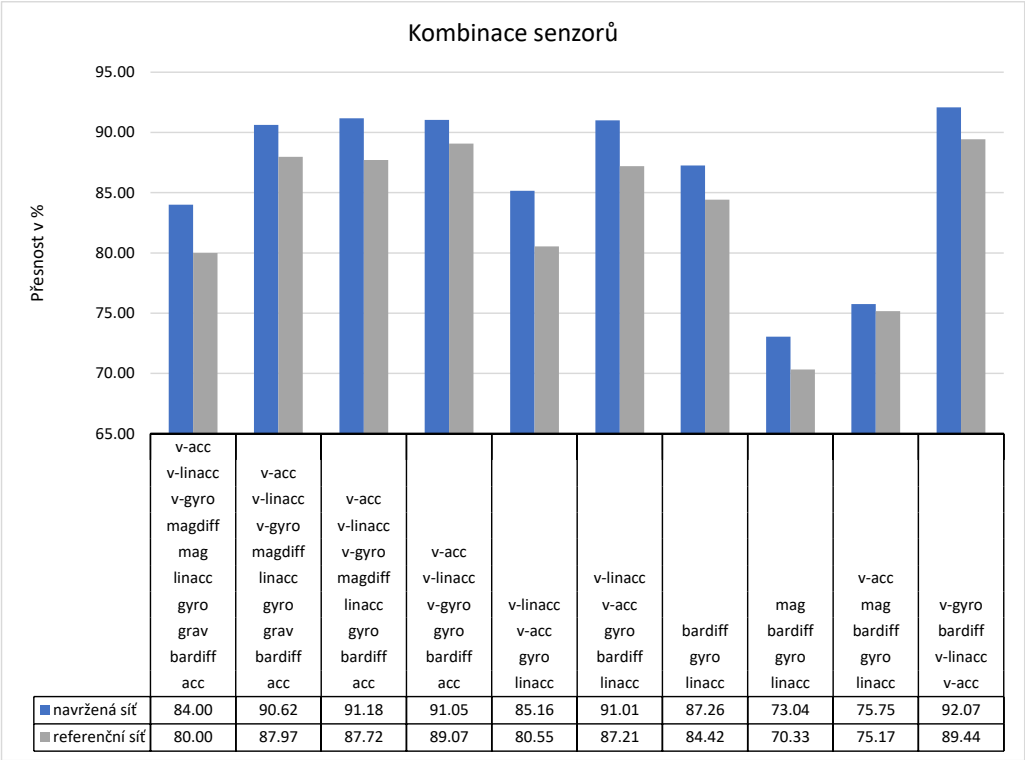
Obrázek 8.2: Confusion matrix virtuálního akcelerometru.



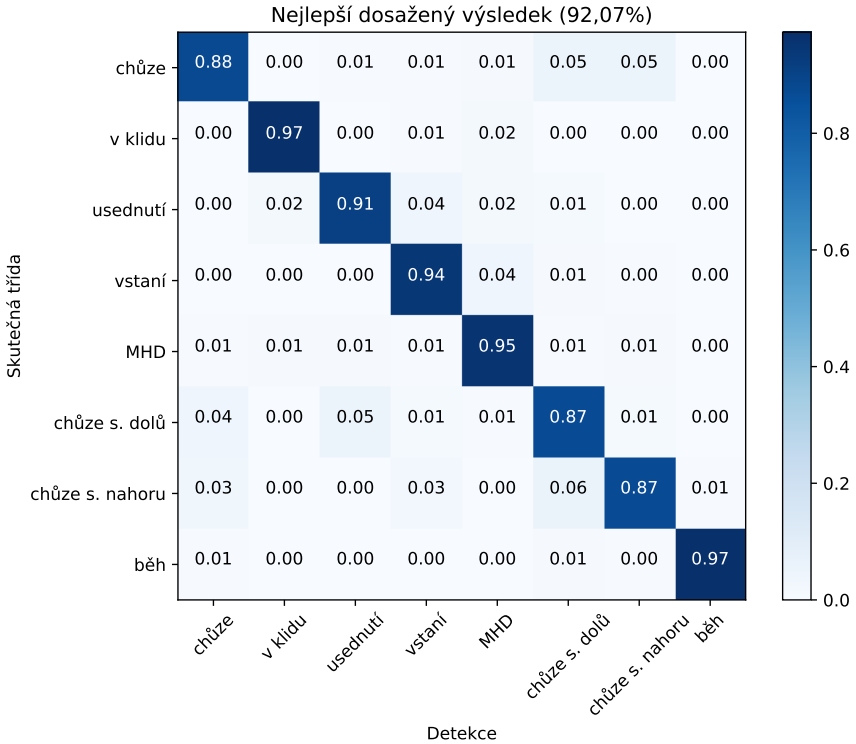
Obrázek 8.3: Confusion matrix magnetometru.

8.4 Výběr kombinace senzorů

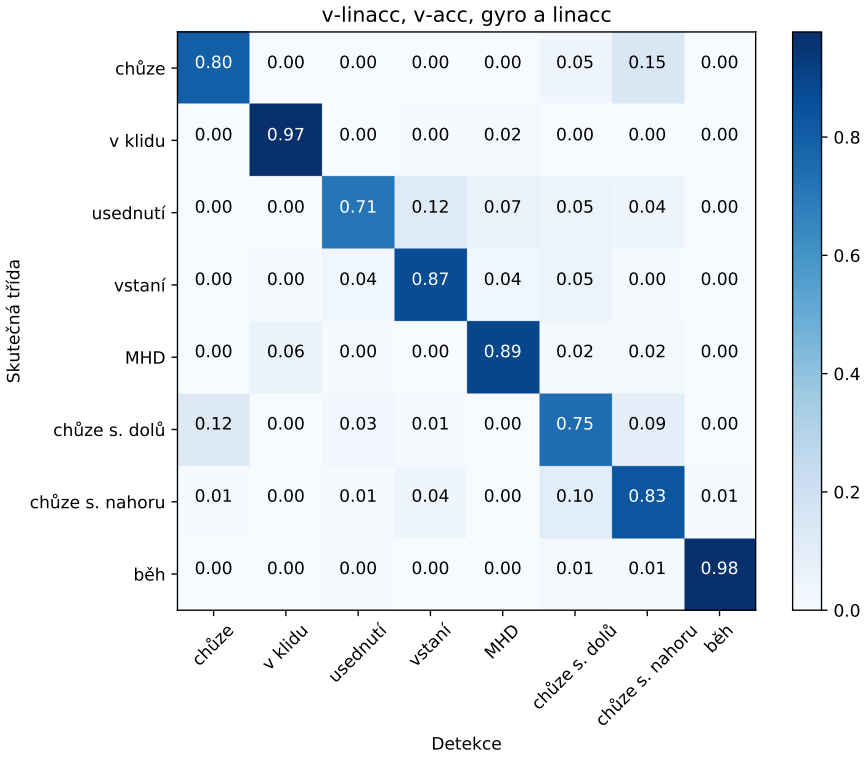
Cílem tohoto experimentu je nalezení co nejvhodnějších kombinací senzorů, které dávají co nejlepší výsledky. Každá kombinace senzorů byla trénována po dobu 20 epoch běhu. Na obr. 8.4 lze vidět graf, kde je vyhodnocení přesnosti pro vybrané kombinace senzorů. Z výsledku jasně vyplývá, že kombinace s magnetometrem dosahují nízké úspěšnosti, což bylo vzhledem k předchozímu experimentu očekávatelné. Další věcí, která stojí za povšimnutí, je vliv barometru. I když samostatně dosahovala data z barometru nízké úspěšnosti, pokud zkombinujeme tento senzor s ostatními senzory, můžeme velmi významně zvýšit úspěšnost detekce. Na obr. 8.6 lze vidět confusion matrix detekce bez barometru a na obr. 8.7 confusion matrix s barometrem. Jak lze na obou obrázcích vidět, barometr napomáhá detekci „vertikálních“ aktivit (vstání, usednutí, chůze po schodech dolů, chůze po schodech nahoru), což lze chápat tak, že když je aktivita uživatelem prováděna, mění se při této aktivitě atmosférický tlak. Na obr. 8.5 lze vidět nejlepší dosažený výsledek (92,07%) na 20 epochách běhu. Jako v předchozím experimentu i zde navrhovaná síť dosahovala lepších výsledků, než referenční síť.



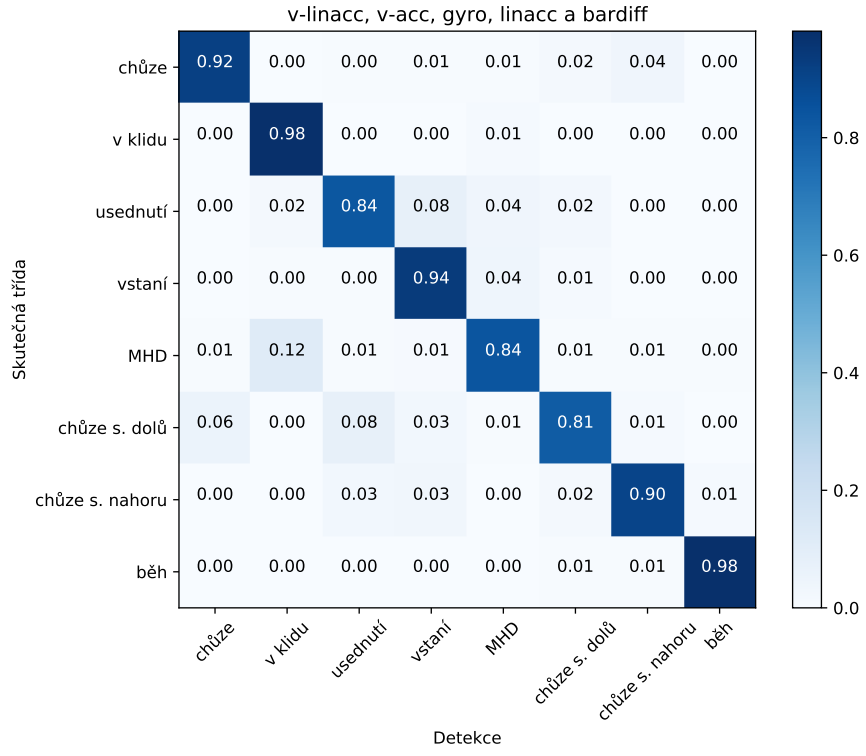
Obrázek 8.4: Graf ukazující výsledek experimentu.



Obrázek 8.5: Nejlepší dosažený výsledek po 20 epochách.



Obrázek 8.6: Bez barometru (atmosferický tlak).



Obrázek 8.7: S barometrem (atmosferický tlak).

8.5 Shrnutí

Z uvedených experimentů vyplývá, že navrhovaná síť překonala ve všech ohledech na daném datasetu referenční síť. Dále se prokázalo, že ač samostatně barometr nedosahuje příliš velké úspěšnosti, při kombinaci s ostatními senzory může významně zlepšit detekci „vertikálních“ aktivit. Další předpoklad, který byl prokázán, je dosažení nejlepších výsledků virtuálním lineárním akcelerometrem (lineární akcelerometr s přepočítaným souřadnicovým systémem vzhledem k Zemi, částečné invariantní vůči natočení zařízení). Senzor, který dosahoval nejhorší přesnosti detekce je pak magnetometr. Zde je výhodnější použití změny magnetického pole než absolutní hodnoty. Z výsledku také vyplývá, že multimodální detektor (detektor založený na více senzorech) má mnohem větší přesnost detekce, než detektor založený na jednom senzoru.

Výsledná navržená síť dosáhla na validační sadě 92,07% úspěšnosti rozpoznání uživatelské aktivity, a to z 8 možných tříd a 6 různých umístění zařízení na těle (např. kapse u kalhot atd.) bez opasků či jiných fixačních pomůcek.

Kapitola 9

Závěr

V této práci jsou obsaženy všechny etapy detekce uživatelských aktivit, tj.: aplikace pro sběr dat, předzpracování dat, navržení a trénování modelu a implementace knihovny pro použití detektoru v cílové aplikaci. V rámci každé etapy se objevily problémy, které bylo potřeba překonat. V etapě tvorby aplikace pro sběr dat bylo potřeba překonat několik problémů, především to, že většina senzorů při úsporném režimu data nesnímají. Situace je komplikovaná, protože jestli senzor data snímá při úsporném režimu, lze detekovat až od Androidu 5.0 a vyšší. Tento problém byl vyřešen tak, že v zařízení je zamezeno přepnutí do plného úsporného režimu a dále bylo použito neoficiálního postupu novou registrací událostí nad senzory. Tento postup funguje na většině běžně dostupných zařízení. Dalším problém byl režim Doze, který je součástí verze Android 6.0 a vyšší. Tento režim pozastavuje služby na pozadí, což je při snímání dat ze senzoru nežádoucí. Režim se podařilo obejít pomocí tzv. služby běžící na popředí. Výsledná aplikace tedy běží i na starších zařízeních s Androidem 4.0 a vyšší.

V etapě předzpracování dat bylo potřeba dataset očistit o nežádoucí data (kraje intervalů měření), data zarovnat do vstupu sítě a normalizovat. Ve fázi návrhu klasifikačního modelu byl navržen model konvoluční neuronové sítě, který v experimentech dosahoval vyšší přesnosti oproti referenčnímu modelu. Zde byl navržen mechanizmus augmentace dat, který se snaží řešit problém s natočením zařízení tím, že data augmentuje a rozšiřuje trénovací množinu dat. V poslední etapě byla vytvořena knihovna pro platformu Xamarin.Android, která umožnuje jednoduchou integraci detektoru do cílové aplikace.

Hlavní přínos této práce spočívá v tom, že spojením všech implementací této práce vzniká komplexní framework pro detekci uživatelských aktivit, který funguje obecně nad všemi dostupnými senzory mobilních zařízení, kde si programátor může určit sadu senzorů, třídy klasifikace a pokročilejší nastavení detektoru. Zároveň framework umožnuje detekci aktivit na pozadí v mobilním zařízení, což není, vzhledem k omezení mobilních platforem, triviální. Dalším přínosem je nasbíraný dataset, který je na rozdíl od většiny dostupných datasetů naměřený v reálném prostředí (bez opasků atd.). Experimenty pak ukazují, který senzor a jaká kombinace senzorů dosahuje nejlepších výsledků detekce. Poslední neméně důležitým přínosem je speciálně vytvořený model neuronové sítě pro účel detekce uživatelských aktivit.

Využití detekce aktivit na mobilním zařízení má široké pole použití, především kontext. Detekcí aktivit získáme kontext, v jakém stavu se uživatel nachází. Kontext je dnes hlavním tématem umělé inteligence. Pokud máme kontext, můžeme lépe překládat, učit systém (sbírat data při anotovaném kontextu) a přizpůsobovat se uživateli. Konkrétními oblastmi použití mohou být např. zdravotnictví, chytré domácnosti a fitness.

Jedna z možností rozšíření této práce je postavení další úrovně detektoru, který by na základě elementárních aktivit detekoval aktivity složitější a komplexnější, např. pomocí tzv. HMM (Hidden Markov Models). Dále by bylo vhodné nasbírat obsáhlejší dataset s více třídami a naměřený větším množstvím dobrovolníků. V implementační části je možnost rozšíření práce o další platformy, především Xamarin.iOS, kde by šel využít sdílený kód s již existující knihovnou.

Literatura

[1] Environment Sensors [online]. 2017 [cit. 2017-1-6]. Dostupné z: https://developer.android.com/guide/topics/sensors/sensors_environment.html.

[2] ETH Zurich Department of Information Technology and Electrical Engineering - Activity Recognition Datasets [online]. 2018 [cit. 2018-05-14]. Dostupné z: https://www.ethz.ch/content/dam/ethz/special-interest/itet/electronics-lab/Research/Dataset/dataset_description.pdf.

[3] Linear Classification. 2018 [cit. 2018-04-02]. Dostupné z: http://cs231n.github.io/linear-classify/.

[4] Low Pass Filter Linear Acceleration [online]. 2018 [cit. 2018-05-17]. Dostupné z: https://github.com/KalebKE/AccelerationExplorer/wiki/Low-Pass-Filter-Linear-Acceleration.

[5] Motion Sensors [online]. 2017 [cit. 2017-1-6]. Dostupné z: https://developer.android.com/guide/topics/sensors/sensors_motion.html.

[6] Optimizing for Doze and App Standby [online]. 2017 [cit. 2017-1-6]. Dostupné z: https://developer.android.com/training/monitoring-device-state/doze-standby.html.

[7] Position Sensors [online]. 2017 [cit. 2017-1-6]. Dostupné z: https://developer.android.com/guide/topics/sensors/sensors_position.html.

[8] Processes and Threads [online]. 2017 [cit. 2017-1-6]. Dostupné z: https://developer.android.com/guide/components/processes-and-threads.html.

[9] Sensor Event [online]. 2017 [cit. 2017-1-6]. Dostupné z: https://developer.android.com/reference/android/hardware/SensorEvent.html.

[10] Sensor: isWakeUpSensor [online]. 2017 [cit. 2017-1-6]. Dostupné z: https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor().

[11] Sensor Manager [online]. 2017 [cit. 2017-1-6]. Dostupné z: https://developer.android.com/reference/android/hardware/SensorManager.html.

[12] Sensor [online]. 2017 [cit. 2017-1-6]. Dostupné z: https://developer.android.com/reference/android/hardware/Sensor.html.

[13] Sensors Overview [online]. 2017 [cit. 2017-1-6]. Dostupné z: https://developer.android.com/guide/topics/sensors/sensors_overview.html.

[14] Service [online]. 2017 [cit. 2017-1-6]. Dostupné z: https://developer.android.com/reference/android/app/Service.html.

[15] Services [online]. 2017 [cit. 2017-1-6]. Dostupné z: https://developer.android.com/guide/components/services.html.

[16] Suspend mode [online]. 2017 [cit. 2017-1-6]. Dostupné z: https://source.android.com/devices/sensors/suspend-mode.html#soc_power_states.

[17] Task-based Asynchronous Programming [online]. 2017 [cit. 2017-1-6]. Dostupné z: https://msdn.microsoft.com/en-us/library/dd537609(v=vs.110).aspx.

[18] TensorFlow - Introduction to TensorFlow Mobile [online]. 2018 [cit. 2018-05-14]. Dostupné z: https://www.tensorflow.org/mobile/mobile_intro.

[19] TensorFlow [online]. 2018 [cit. 2018-05-14]. Dostupné z: https://www.tensorflow.org/.

[20] TensorFlow [online]. 2018 [cit. 2018-05-14]. Dostupné z: https://www.tensorflow.org/get_started/premade_estimators.

[21] Xamarin: Introduction to Mobile Development [online]. 2017 [cit. 2017-1-6]. Dostupné z: https://developer.xamarin.com/guides/cross-platform/getting_started/introduction_to_mobile_development/.

[22] Xamarin: Xamarin.Forms [online]. 2017 [cit. 2017-1-6]. Dostupné z: https://developer.xamarin.com/guides/xamarin-forms/.

[23] Anguita, D.; Ghio, A.; Oneto, L.; aj.: Human Activity Recognition on Smartphones Using a Multiclass Hardware-Friendly Support Vector Machine. *Ambient Assisted Living and Home Care*, 2012: s. 216–223, doi:10.1007/978-3-642-35395-6_30.

[24] Bishop, C. M.: *Pattern recognition and machine learning*. New York: Springer, 2006, ISBN 03-873-1073-8.

[25] Burkill, J.: Artificial Neural Networks – Part 1: The XOr Problem. 2016 [cit. 2018-04-01]. Dostupné z: http://www.mlopt.com/?p=160.

[26] Dargie, W.; Denko, M. K.: Analysis of Error-Agnostic Time- and Frequency-Domain Features Extracted From Measurements of 3-D Accelerometer Sensors. *IEEE Systems Journal*, ročník 4, č. 1, 2010: s. 26–33, ISSN 1932-8184, doi:10.1109/JSYST.2009.2039735.

[27] Dozat, T.: Incorporating Nesterov Momentum into Adam. *ICLR 2016 workshop submission*, 2016.

[28] Goodfellow, I.; Bengio, Y.; Courville, A.: *Deep learning*. Cambridge, Massachusetts: The MIT Press, 2016, ISBN 978-026-2035-613.

[29] Ha, S.; Choi, S.: Convolutional neural networks for human activity recognition using multiple accelerometer and gyroscope sensors. *2016 International Joint Conference on Neural Networks (IJCNN)*, 2016: s. 381–388, doi:10.1109/IJCNN.2016.7727224.

[30] Ha, S.; Yun, J.-M.; Choi, S.: Multi-modal Convolutional Neural Networks for Activity Recognition. *2015 IEEE International Conference on Systems, Man, and Cybernetics*, 2015: s. 3017–3022, doi:10.1109/SMC.2015.525.

[31] Haykin, S. S.: *Neural networks and learning machines*. New York: Prentice Hall, třetí vydání, 2009, ISBN 01-314-7139-2.

[32] Ioffe, S.; Szegedy, C.: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, 2015: s. 448–456.

[33] Kingma, D. P.; Ba, J.: Adam: A Method for Stochastic Optimization. *CoRR*, 2014.

[34] Kreyszig, E.: *Advanced engineering mathematics*. New York: Wiley, Čtvrté vydání, c1979, ISBN 04-710-2140-7.

[35] Krizhevsky, A.; Sutskever, I.; Hinton, G. E.: ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, ročník 60, č. 6, 2017-05-24: s. 84–90, ISSN 00010782, doi:10.1145/3065386.

[36] Lee, S.-M.; Yoon, S. M.; Cho, H.: Human activity recognition from accelerometer data using Convolutional Neural Network. *2017 IEEE International Conference on Big Data and Smart Computing (BigComp)*, 2017: s. 131–134, doi:10.1109/BIGCOMP.2017.7881728.

[37] Lee, Y.-S.; Cho, S.-B.: Activity Recognition Using Hierarchical Hidden Markov Models on a Smartphone with 3D Accelerometer. *Hybrid Artificial Intelligent Systems*, 2011: s. 460–467, doi:10.1007/978-3-642-21219-2_58.

[38] McCulloch, W. S.; Pitts, W.: A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, ročník 5, č. 4, 1943: s. 115–133, ISSN 0007-4985, doi:10.1007/BF02478259.

[39] Olson, D. L.; Delen, D.: *Advanced data mining techniques*. Berlin: Springer, první vydání, 2008, ISBN 35-407-6916-1.

[40] Španěl, M.; Vítězslav, B.: Obrazové segmentační techniky Přehled existujících metod. 2006 [cit. 2018-04-01]. Dostupné z: `http://www.fit.vutbr.cz/ spanel/segmentace/`.

[41] Reyes-Ortiz, J.-L.; Oneto, L.; Samà, A.; aj.: Transition-Aware Human Activity Recognition Using Smartphones. *Neurocomputing*, ročník 171, 2016: s. 754–767, ISSN 09252312, doi:10.1016/j.neucom.2015.07.085.

[42] Ronao, C. A.; Cho, S.-B.: Human activity recognition using smartphone sensors with two-stage continuous hidden Markov models. *2014 10th International Conference on Natural Computation (ICNC)*, 2014: s. 681–686, doi:10.1109/ICNC.2014.6975918.

[43] Rosenblatt, F.: The perceptron. *Psychological Review*, ročník 65, č. 6, 1958: s. 386–408, ISSN 1939-1471, doi:10.1037/h0042519.

[44] Siirtola, P.; Röning, J.: Recognizing Human Activities User-independently on Smartphones Based on Accelerometer Data. *International Journal of Interactive Multimedia and Artificial Intelligence*, ročník 1, č. 5, 2012: s. 38–45, ISSN 1989-1660, doi:10.9781/ijimai.2012.155.

[45] Srivastava, N.; Hinton, G.; Krizhevsky, A.; aj.: Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.*, ročník 15, č. 1, 2014: s. 1929–1958, ISSN 1532-4435.

[46] Su, X.; Tong, H.; Ji, P.: Activity recognition with smartphone sensors. *Tsinghua Science and Technology*, ročník 19, č. 3, 2014: s. 235–249, ISSN 1007-0214, doi:10.1109/TST.2014.6838194.

[47] Werbos, P. J.: *The roots of backpropagation*. New York: Wiley, 1994, ISBN 04-715-9897-6.

[48] Williams, J.: Getting Android Sensor Events While The Screen is Off [online]. 2017 [cit. 2017-1-6]. Dostupné z: `https://nosemaj.org/android-persistent-sensors`.

[49] Yang, J.-Y.; Wang, J.-S.; Chen, Y.-P.: Using acceleration measurements for activity recognition. *Pattern Recognition Letters*, ročník 29, č. 16, 2008: s. 2213–2220, ISSN 01678655, doi:10.1016/j.patrec.2008.08.002.

[50] Zebin, T.; Scully, P. J.; Ozanyan, K. B.: Human activity recognition with inertial sensors using a deep learning approach. *2016 IEEE SENSORS*, 2016: s. 1–3, doi:10.1109/ICSENS.2016.7808590.

[51] Zeng, M.; Nguyen, L. T.; Yu, B.; aj.: Convolutional Neural Networks for Human Activity Recognition using Mobile Sensors. *Proceedings of the 6th International Conference on Mobile Computing, Applications and Services*, 2014: s. 197–205, doi:10.4108/icst.mobicase.2014.257786.

Přílohy

Příloha A

Obsah elektronického nosiče

Obsah CD:

bin/SensorAI.DatasetCreator Aplikace pro předzpracování dat (tvobru datasetu) pro Windows (x64)

bin/SensorAI.Library Zkompilováná knihovna pro detekci uživatelských aktivit pro Xamarin.Android

bin/SensorAI.DataCollector Aplikace pro sběr dat pro Android OS

bin/SensorAI.TestDetector Testovací aplikace využivající knihovnu SensorAI.Library pro Android OS

bin/manual.pdf Manuál k aplikaci SensorAI.DataCollector

bin/readme.txt Obecný manuál ke všem aplikacím a kompilaci

script/SensorAI.Train Trénovací skripty neuronové sítě pro Windows/Linux

src Zdrojové kódy k aplikacím

model Přetrénovaný model s nastavením detektoru

dataset Nasbíraný dataset uživatelských aktivit

doc Text práce v pdf

docsrc Zdrojový text práce v LaTeX