VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

IMPLEMENTACE JEDNODUCHÉHO ROZPOZNÁVAČE ŘEČI PRO ANDROID

IMPLEMENTATION OF SIMPLE SPEECH RECOGNIZER IN ANDROID

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **PETR FLAJŠINGR**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. IGOR SZŐKE, Ph.D.**
SUPERVISOR

BRNO 2018

Zadání bakalářské práce/20485/2017/xflajs00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Flajšingr Petr**
Obor: Informační technologie
Téma: **Implementace jednoduchého rozpoznávače řeči pro Android**
Implementation of Simple Speech Recognizer in Android
Kategorie: Softwarové inženýrství

Pokyny:

1. Nastudujte základy implementace aplikací pro Android (SDK, NDK) a základy rozpoznávání řeči.
2. Implementuje jednoduchý rozpoznávač řeči (extrakce příznaků, forward pass přes neuronovou síť, dekodér). Pokud to bude možné, využijte dostupných knihoven.
3. Tam kde to půjde, využijte low level implementace pro urychlení (NDK, RenderSript).
4. Vyhodnoťte úspěšnost a hlavně časové a paměťové nároky. Navrhněte směry dalšího vývoje.
5. Vyrobte A2 plakátek a cca 30 vteřinové video prezentující výsledky vaší práce.

Literatura:

- Dle pokynů vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Bod 1 a část bodů 2 a 3 ze zadání.

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

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

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

Vedoucí: **Szőke Igor, Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

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

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato práce se zabývá implementací a optimalizací rozpoznávače řeči pod operačním systémem Android. Pokrývá implementaci nahrávání zvukového signálu, následnou extrakci příznaků pomocí Mel bank filtrů a neuronové sítě. Také obsahuje informace o implementaci dynamického dekodéru. Práce se věnuje převážně implementaci v nízkoúrovňových nástrojích jako jsou Android NDK a Renderscript a vyhodnocuje úspěšnost rozpoznávače a jeho paměťové a časové nároky.

Abstract

The subject of this thesis is an implementation and optimization of speech recognizer for operating system Android. This work covers implementation of recording of an audio signal and the subsequent feature extraction using Mel filter banks and neural network. It also contains information about implementation of dynamic decoder. The work focuses on implementation in low-level tools such as Android NDK and Renderscript and evaluates the success rate of the recognizer and its memory and time requirements.

Klíčová slova

rozpoznávání řeči, Android, NDK, Renderscript, neuronové sítě, dekodér, extrakce příznaků

Keywords

speech recognition, Android, NDK, Renderscript, neural networks, decoder, feature extraction

Citace

FLAJŠINGR, Petr. *Implementace jednoduchého rozpoznávače řeči pro Android*. Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Igor Szőke, Ph.D.

Implementace jednoduchého rozpoznávače řeči pro Android

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Igora Szőke, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . . . .
Petr Flajšingr
15. května 2018

Poděkování

Chtěl bych poděkovat vedoucímu práce Ing. Igoru Szőke, Ph.D. za odborné konzultace a pomoc při řešení této práce. Dále také děkuji Danielu Jarošovi za korekturu tohoto textu.

Obsah

Kapitola 1

Úvod

Rozpoznávání řeči je oblast, která je už po dlouhou dobu aktuální a neustále se posouvá kupředu. Je velice užitečné pro usnadnění mnoha činností, ať už se jedná o převod přednášek do textové podoby nebo osobního asistenta v mobilních zařízeních. Vzhledem k nárustu výpočetní kapacity v zařízeních, jako jsou smartphony, je možnost využít tyto služby téměř kdykoliv a právě tímto se zabývá tato práce.

Ačkoliv se výkon mobilních zařízení zvyšuje velice rychle, nemohou dosáhnout výkonu běžných počítačů. Kvůli tomuto je nutné se při vývoji výpočetně náročných aplikací soustředit na jejich optimalizaci a paralelizaci využitých algoritmů, jelikož mobilní zařízení většinou obsahují procesory s velkým počtem výpočetních jader. Specificky systém Android, na který je tato práce zaměřena, umožňuje využití několika nízkoúrovňových nástrojů jak pro paralelizaci tak pro běžné zrychlení exekuce programu.

Druhá kapitola této práce se zabývá vysvětlením metod použitých pro implementaci rozpoznávače a jejich souvislosti. Vysvětluje, jakým způsobem je signál uložen v diskrétních zařízeních, kterými je drtivá většina námi využívaných výpočetních systémů. Popisuje, jakým způsobem jsou obecně zvuková data zpracována, také postup extrakce příznaků a nakonec základní principy rozpoznávání spojité řeči za využití dekodéru implementovaného pomocí skrytých Markovových modelů.

Kapitola třetí se věnuje návrhu řešení práce. Obsahuje informace o použitých technologií jak přímo pro implementaci v systému Android, tak i pro obecný vývoj. Také jsou zde popsány využité knihovny a funkce třetích stran a je zde obsažen popis struktury knihovny s krátkým vysvětlením funkce jednotlivých tříd.

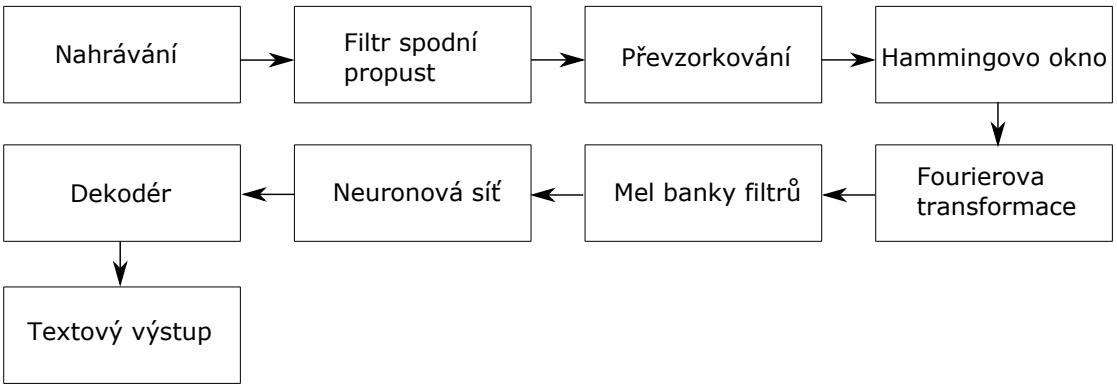
Čtvrtá kapitola je věnována samotné implementaci. Popisuje specifika nahrávání zvukové stopy, extrakce příznaků a detekce řečové aktivity. Také popisuje implementaci dekodéru a postup vyhledávání nejpravděpodobnějších sekvencí slov. Také je zde vysvětleno předávání informací mezi vlákny programu a komunikace s virtuálním prostředí Javy z nativního kódu. Poslední sekce je věnována demonstrační aplikaci.

Poslední, pátá kapitola, se zabývá testováním algoritmů a jejich verifikaci proti referenční implementaci. Srovnává implementace v různých jazycích a zhodnocuje úspěšnost rozpoznávání řeči a celkovou výpočetní a paměťovou náročnost programu.

Kapitola 2

Úvod do řešené problematiky

Tato kapitola si dává za cíl seznámit čtenáře s metodami využitými v této práci. Popisuje průběh zpracování od nahrávání zvukové stopy po dekodér. Pořadí těchto operací je zobrazeno na obrázku 2.1.



Obrázek 2.1: Kroky zpracování signálu řeči

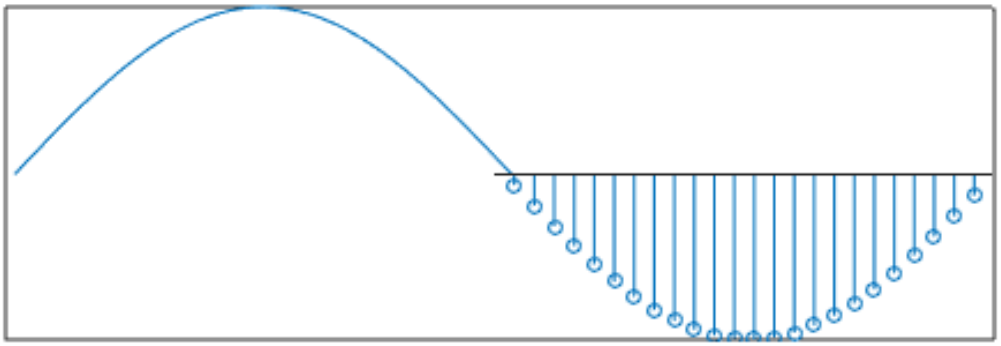
Jsou zde uvedeny základní informace o způsobu uložení signálu v diskrétních zařízeních a udává informace o správných krocích pro uložení signálu řeči a jeho předzpracování před extrakcí příznaků. Dále je zde popsán postup extrakce příznaků pro rozpoznávání řeči pomocí Mel bank filtrů a techniky využité k její následné klasifikaci za využití neuronové sítě. Poslední část kapitoly vysvětluje princip dekodéru, jazykového a akustického modelu a také popisuje zpracování spojité řeči.

2.1 Zvukový signál

Zvukový signál je reprezentací vibrací vzduchu či jiného přenosového média buď analogovou nebo digitální formou.

2.1.1 Digitální reprezentace

Zvuk je v běžných zařízení nahráván analogově, ale abychom byli schopni ho ukládat v paměti a dále s ním nějak pracovat, je nutné ho převést do diskrétní podoby. Toho je dosaženo za pomocí vzorkování spojitého signálu, jehož zdrojem může být například mikrofon v mobilním zařízení, s využitím analogově digitálního převodníku.



Obrázek 2.2: Porovnání spojitého a diskrétního signálu

Přímo v paměti je běžně jeden vzorek digitálního signálu uložen jako 8 bitů či jejich násobek, to se nazývá bitová hloubka. Hodnota vzorku určuje amplitudu signálu v určitém okamžiku.

Další vlastností je vzorkovací frekvence. Ta definuje, kolik vzorků audia je v jedné sekundě záznamu.

Také je možné signál reprezentovat v mono nebo stereo formátu. Stereo umožňuje uložení zvuku tak, aby se posluchači mohlo zdát, že zvuk přichází z určitého směru, ovšem, v případě reálně nahrávaného zvuku, je nutné využití dvou mikrofonů. Proto, že v rozpoznávání řeči by tohle přineslo více nevýhod než výhod, např. dvojnásobné množství dat ke zpracování, je vhodnější data nahrávat pouze na jeden kanál.

2.1.2 Řeč

Vzhledem k tomu, že cílem je rozpoznávání řeči a ne jiného druhu zvukového signálu, je možné výrazně omezit, které frekvence a jejich kombinace jsou pro nás podstatné.

Rozsah frekvence zvuku vydávaného lidským řečovým ústrojím se, u průměrného člověka, pohybuje v rozmezí 300 Hz až 3400 Hz. Díky tomu není nutné pro přenos signálu řeči používat vyšší frekvence. I když v krajních případech může omezení šířky frekvenčního pásma vést k horšímu přenosu některých zvuků - např. hlásky "s" a "f" mohou znít v telefonním pásmu (4 kHz) posluchači stejně [2].

2.1.3 Převzorkování

Běžně používanou vzorkovací frekvencí pro rozpoznávání řeči je 8000 Hz. Tato frekvence je využívána z důvodu urychlení zpracování dat - data pro účely rozpoznávání zachovávají dostatečnou kvalitu řeči v signálu (dáno Shannonovým teorémem popsaným níže), ale výrazně snižuje množství dat ke zpracování [4]. Další důvod využití této frekvence je historický, jelikož byla a stále je používána pro signál přenášený v telefonické komunikaci

Zpravidla je ovšem většina zvukových záznamů nahrávána ve vyšší frekvenci, je proto nutné snížit vzorkovací frekvenci záznamu. Toho je dosaženo pomocí převzorkování.

Před převodem z vyšších frekvencí je ovšem vhodné použít filtr dolní propust. Po aplikaci dolní propusti je z původního signálu odebrán každý N-tý vzorek, kde N je podílem původní vzorkovací frekvence a frekvence požadované.

FIR filtr - dolní propust

Filtr s konečnou odezvou je diskrétní lineární filtr, který se v konečném čase ustálí na hodnotě 0. V této implementaci je využit pro aplikaci filtru dolní propust při převzorkování za účelem odstranění nechtěných frekvencí a vyhnutí se aliasingu signálu. Spodní propust ponechává v signálu pouze frekvence pod určitou úrovní, proto je využit k izolaci mluveného slova. FIR filtr lze vyjádřit rovnicí 2.1

$$y[n] = \sum_{k=0}^{N} h[k]x[n-k]$$

kde x je vstupní signál, h je impulzní odezva, y je výstupní signál a N je řád filtru (2.1)

Shannonův teorém

Tento teorém udává, že vzorkovací frekvence signálu musí být dvakrát větší, než nejvyšší frekvence, kterou chceme přenášet. Proto je v našem případě využita vzorkovací frekvence 8 kHz. V případě nedodržení tohoto pravidla dochází v signálu k aliasingu a kvalita informací v něm přenášených je výrazně snížena.

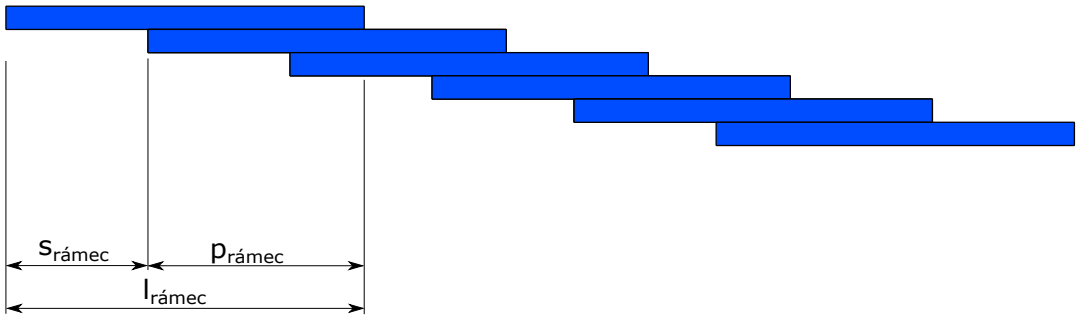
2.2 Extrakce příznaků

Extrakce příznaků je obecně v rozpoznávání proces zaměřený na snížení množství dat nutných k jejich klasifikaci, tedy výraznému snížení dimenzionality vstupních dat. Menší množství dat velmi urychluje proces rozpoznávání a šetří cenný procesorový čas. Dále extrakce poskytuje možnost se zbavit irelevantních dat a soustředit se pouze na ty, co jsou nezbytné.

2.2.1 Segmentace

Prvním krokem při zpracování zvukového signálu pro extrakci příznaků je segmentace. Vstupem této části je audio signál, jehož předzpracování bylo popsáno v předchozí podkapitole.

Signál je nutné rozdělit na malé části, aby bylo možné určit, v jakém stavu bylo v daný moment řečové ústrojí člověk (např. vyslovení hlásky "a"). Tyto části jsou nazývány rámce. Běžná délka rámce je 20-25 milisekund a jejich překryv má velikost 10-15 ms.



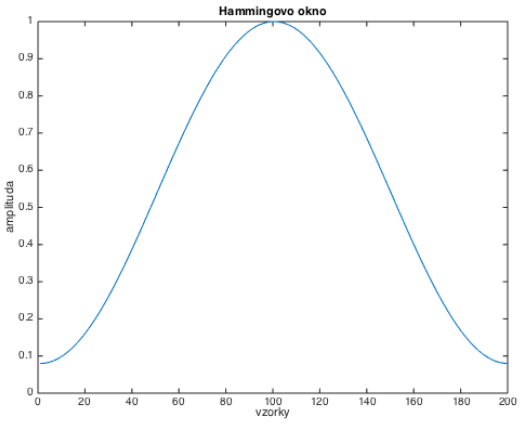
Obrázek 2.3: Reprezentace rámců - l_{ramec} - délka rámce, s_{ramec} - posun, p_{ramec} - překryv

Specificky v této implementaci je délka rámce 25 ms a překryv 15 ms.

Okenní funkce

Po rozdělení signálu na části je na každý rámec aplikována okenní funkce. Důvodem jejich využití je získání lepších výsledků při použití Fourierovy transformace. V této implementaci je využito Hammingovo okno. Tato funkce oslabuje amplitudu signálu na okrajích rámce.

$$w(n) = \alpha - \beta \cos(\frac{2\pi n}{N-1}) \text{ kde } \alpha = 0.54, \beta = 0.46 \tag{2.2}$$



Obrázek 2.4: Hammingovo okno

2.2.2 Fourierova transformace

Fourierova transformace umožňuje rozdělení vstupního signálu na periodické frekvence, ze kterých je složen. Díky tomu můžeme zjistit, ve kterých frekvencích má signál nejvíce energie. Jelikož každý tón vydávaný lidským hlasovým ústrojím má svoje specifické frekvenční složky, je jejich rozložení nesmírně důležité pro jejich automatizované rozpoznávání [1].

$$X_k = \sum_{n=0}^{N-1} x_n e^{\frac{-i2\pi kn}{N}} \tag{2.3}$$

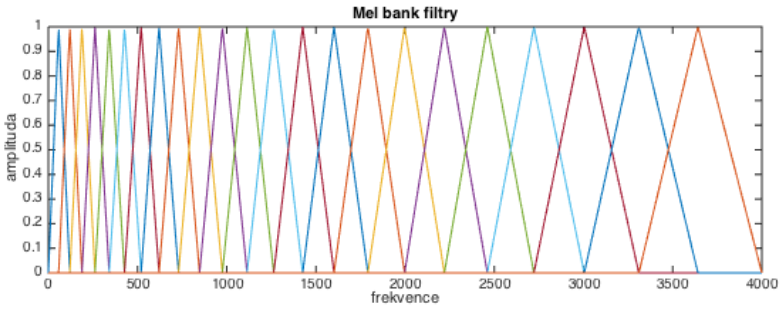
Rychlá Fourierova transformace

FFT je speciální verzí Diskrétní Fourierovi transformace, která je široce využívaná pro její výpočet. Oproti DFT, jejíž složitost je $O(N^2)$, dosahuje FFT složitosti $O(N \log N)$, což výrazně snižuje dobu výpočtu.

2.2.3 Mel bank filtry

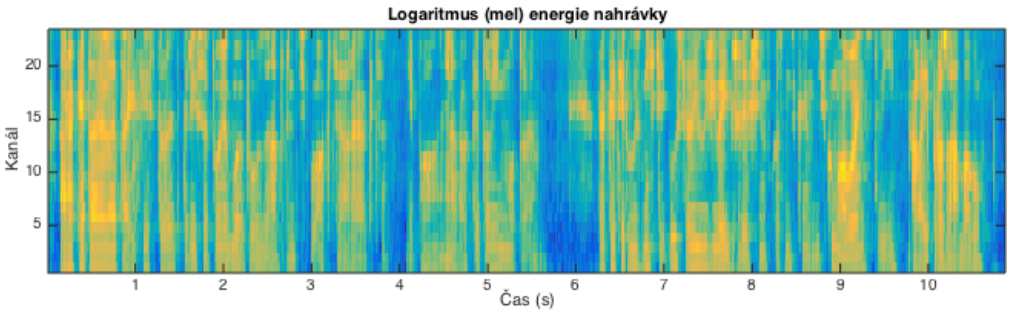
Vstupem mel bank filtrů je Fourierova transformace jednotlivých rámců signálu. Pomocí trojúhelníkových okenních funkcí jsou energie v každé části spektra namapovány na mel měřítko. Po výpočtu funkcí je každá z nich logaritmizována a tyto hodnoty se využívají v dalších výpočtech.

Parametry okenních funkcí použitých pro výpočet energií jsou závislé na počtu použitých filtrů, ze kterých chceme počítat energii, vzorkovací frekvenci signálu a velikosti vstupních dat. Také je možné dále omezit frekvence, které nás nezajímají tím, že změníme počáteční frekvenci nejnižšího okna, respektive konečnou frekvenci okna nejvyššího .



Obrázek 2.5: Mel bank filtry - 24 filtrů, vzorkovací frekvence 8 kHz

Výstupem je matice logaritmů energií v signálu. Tímto jsme dosáhli podstatného snížení dimenzionality oproti původním datům. Po vypočtení energií je každý kanál normalizován pomocí odečtení průměru energií v daném kanálu přes celou nahrávku za účelem normalizace energie mluvy řečníka.



Obrázek 2.6: Matice logaritmu energií nahrávky

Mel měřítko

Člověk, v porovnání s počítačem, vnímá zvukové frekvence řeči rozdílně. Posluchači by se mohlo zdát, že rozdíl frekvencí je výrazně nižší, než ve skutečnosti. Se zvyšující se frekvencí je tento jev výraznější [6].

Mel měřítko vzniklo přesně z tohoto důvodu, aby bylo možné lépe aproximovat jakým způsobem člověk ve skutečnosti zvuk vnímá. Převod mezi frekvencí a mel je prováděn pomocí rovnice 2.4. Pro převod z mel měřítka do frekvence je možné použít rovnici 2.5.

$$m = 2595 \log_{10}(1 + \frac{f}{700}) = 1127 \ln(1 + \frac{f}{700}) \tag{2.4}$$

$$f = 700(10^{\frac{m}{2595}} - 1) = 700(e^{\frac{m}{1127}} - 1) \tag{2.5}$$

2.3 Neuronová síť

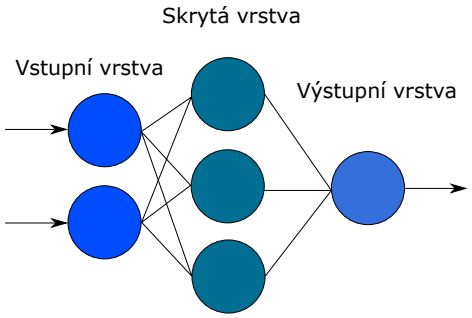
Neuronové sítě jsou využívány jako nástroj, který se dokáže na základě trénovacích dat naučit klasifikovat vstupní data. Specificky v rozpoznávání řeči mohou být použity přímo pro její rozpoznání nebo, jako v případě této práce, pro extrakci příznaků.

2.3.1 Struktura

Neuronové sítě se skládají z částí - neuronů - které provádí základní matematické operace. Každý neuron má na svém vstupu jednu nebo více hodnot. Množství těchto hodnot závisí na typu sítě a její velikosti.

Typ sítě, který je pro tuto práci podstatný je tzv. feedforward. Tento typ je nejjednodušším typem neurální sítě. Jeho vrstvy jsou striktně odděleny a informace se v něm mohou pohybovat pouze jedním směrem - tedy od vstupních uzlů k výstupním.

Vrstvy se dělí na tři typy. Prvním z nich je vrstva vstupní. Ta na vstup přijímá data určená ke klasifikaci - v našem případě výstup Mel bank filtrů. Vrstva zpracuje data a předá je tzv. skryté vrstvě. Těchto vrstev může být virtuálně neomezené množství. Poslední typ, výstupní vrstva, udává výsledek klasifikace. Počet výstupních uzlů určuje kolik různých tříd je síť schopna klasifikovat. V případě rozpoznávání řeči mohou být výstupem např. pravděpodobnosti tzv. phoneme posteriors, tedy klasifikace fonémů řeči. Vizualizace jednoduché neuronové sítě je na obrázku 2.7.



Obrázek 2.7: Grafická reprezentace jednoduché neuronové sítě

Neuron

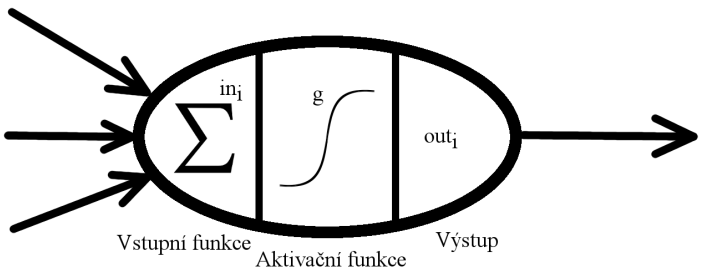
Neuron je základní výpočetní jednotkou sítě. Na úrovni vstupní vrstvy každý neuron odpovídá jedné části vstupních dat. Na svůj vstup přijímá součet všech spojení na něj napojených

z předchozí vrstvy (rovnice 2.7) a - v některých implementacích - k nim přičítá tzv. bias. Ke každému ze zmíněných spojení je přiřazena váha, kterou je přenášená hodnota vynásobena. Po sečtení dat je výsledek odeslán do aktivační funkce. Nejběžnější z nich je sigmoidální přenosová funkce - rovnice 2.6.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.6}$$

$$out_i = S(\sum_{i=1}^{N}(\omega_i x_i) + \Theta) \tag{2.7}$$

kde x je vstup neuronu, ω je váha spojení, Θ je bias a $S()$ je aktivační funkce



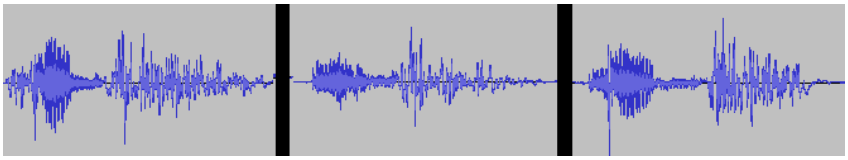
Obrázek 2.8: Grafická reprezentace neuronu

Dopředná neuronová síť

Typ sítě využitý v této práce se nazývá dopředná neuronová síť (feed forward). Jedná se o nejjednodušší a zároveň také nejběžněji využívaný způsob implementace. Data v síti prochází pouze jedním směrem a síť obsahuje obvykle alespoň jednu skrytou vrstvu. Zpravidla bývají všechny uzly sousedních vrstev propojeny. Grafická reprezentace malé sítě tohoto typu je zobrazena na obrázku 2.7.

2.4 Dekodér

Účelem dekodéru je nalezení nejpravděpodobnější sekvence slov $\tilde{W}$ podle pozorované sekvence příznaků O. Vzhledem k velké varianci toho, jak mohou být slova vyřčena různými řečníky (například různá délka mluvy, barva hlasu nebo přízvuk) není možné jednoznačně určit, které slovo bylo vyřčeno. Tento problém je ilustrován na obrázku 2.9 a je z něj jasné, že ani jeden řečník nevysloví slovo nikdy stejně, proto jsou využity dále popsané metody. Informace uvedené v této podkapitole byly čerpány z [3].



Obrázek 2.9: Slovo "one" třikrát vyřčeno stejným řečníkem

Rovnice 2.8 popisuje jakým způsobem je získána nejpravděpodobnější posloupnost slov $\tilde{W}$. Cílem je tedy na základě pozorovaných příznaků O nalézt posloupnost slov, která jim nejlépe odpovídá, tedy $P(W|O)$.

$$\tilde{W} = \underset{W}{\operatorname{argmax}} P(W|O) \tag{2.8}$$

Aplikací Bayesova pravidla získáme rovnici 2.9. Tato úprava umožňuje rozdělení problému do několika částí, které lze modelovat odděleně. První z nich je apriorní pravděpodobnost výskytu slov $P(W)$, která reprezentuje jazykový model (2.4.2), druhou je pravděpodobnost $P(O|W)$ jenže nese informaci o akustickém modelu (2.4.1). Hodnoty používané pro výpočet těchto pravděpodobností je nutné znát před samotným rozpoznáváním, je tedy nutné je natrénovat pomocí textových a řečových dat. Pravděpodobnost $P(O)$ nemá žádnou spojitost s hledaným slovem W a proto není při výpočtech využívána. Namísto toho je pro hledání slov použita rovnice 2.10.

$$\tilde{W} = \underset{W}{\operatorname{argmax}} \frac{P(W)P(O|W)}{P(O)} \tag{2.9}$$

$$\tilde{W} = \underset{W}{\operatorname{argmax}} P(W,O) = \underset{W}{\operatorname{argmax}} P(W)P(O|W) \tag{2.10}$$

2.4.1 Akustický model

Akustický model je používán k reprezentování vztahů mezi zvukovým signálem a fonémy nebo jinými jednotkami, ze kterých je řeč složena. Je vytvářen za pomocí zvukových nahrávek a jejich přepisů. Jeho cílem je získat co nejlepší odhad podmíněné pravděpodobnosti $P(O|W)$ (rovnice 2.10).

Základním předpokladem akustického modelování je to, že při mluvě je lidské řečové ústrojí v dostatečně malém časovém rozsahu v jednom z konečného počtu stavů. Díky tomu je možné rozdělit slova na malé jednotky určené těmito stavy - v tomto případě fonémy.

Jak již bylo zmíněno v předchozí sekci, klasifikaci výstupu Mel bank filtrů obstarává neuronová síť, jejímž výstupem jsou pravděpodobnosti jednotlivých fonémů. Tyto pravděpodobnosti jsou zpracovány v dekodéru, kde jsou jednotlivé výslovnosti slov modelovány pomocí Markovových modelů.

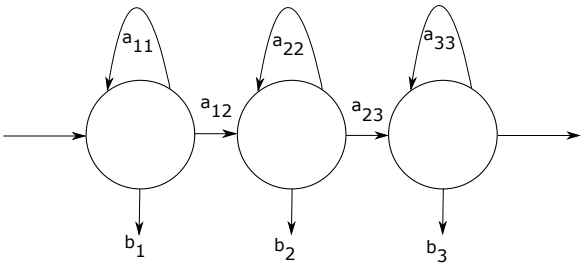
Skryté Markovovy modely

Nejpoužívanějším způsobem akustického modelování jsou v současné době skryté Markovovy modely (Hidden Markov Models). Jedná se o o model stochastického procesu, který generuje v diskrétních časových okamžicích vektor pozorování O a zároveň také mění svůj stav podle pravděpodobností přechodů a_{ij}. Pravděpodobnosti přechodu určují s jakou pravděpodobností přechází model ze stavu s_i do stavu s_j. Tato pravděpodobnost je definována rovnicí 2.11. Zároveň jsou tyto pravděpodobnosti v modelu konstantní, nemění se tedy s časem a jejich součet je jedna (rovnice 2.12).

$$a_{ij} = P(s(t+1) = s_j | s(t) = s_i) \tag{2.11}$$

$$\sum_{j=1}^{N} a_{ij} = 1 \tag{2.12}$$

Při zpracování řeči jsou používány modely, ve kterých lze vyjádřit časový posun. Vzhledem k tomu, že jsou data zpracována po rámcích, reprezentuje přechod jeden časový krok. Ukázka takového modelu je vyobrazena na obrázku 2.10.



Obrázek 2.10: Jednoduchý Markovův model

Konečným výstupem akustického modelu je vektor pravděpodobností výskytu modelovaných jednotek.

2.4.2 Jazykový model

Jazykový model obsahuje slovník, který určuje, jaká slova rozpoznávač podporuje a také obsahuje pravidla pro zřetězení zmíněných slov. Důvodem jeho využití je tedy co nejpřesnější odhad $P(W)$ pro rozpoznávanou posloupnost slov (rovnice 2.10). Tuto pravděpodobnost je možné vyjádřit vztahem v rovnici 2.13.

$$P(W) = \prod_{i=0}^{k} P(w_i) \tag{2.13}$$

Předchozí rovnice ovšem popisuje pouze výpočet pravděpodobnosti pro unigramový jazykový model - tedy model, který definuje pouze pravděpodobnost výskytu jednotlivých slov. Tento model samozřejmě zlepšuje přesnost rozpoznávání, ovšem běžně používané jazykové modely obsahují n-gramy vyššího řádu.

N-gramy definují pravděpodobnost výskytu slova na základě historie $n-1$ slov. Rovnice 2.14 popisuje výpočet za využití bigramů, tedy pravděpodobnost výskytu slova závisí na

jednom předchůdci. To, že pravděpodobnost závisí pouze na historii nám umožňuje provádět rozpoznávání již během promluvy a ne až po jejím skončení.

$$P(W) = \prod_{i=1}^{k} P(w_i|w_{i-1}) \tag{2.14}$$

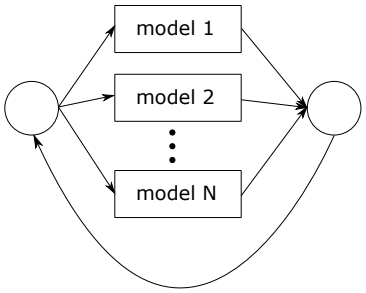
Model ovšem nemusí nutně obsahovat záznamy o všech slovních kombinacích. Vzhledem k tomu, že není možné získat dostatek trénovacích dat na pokrytí všech kombinací, je v bigramových modelech a modelech vyššího řádu využívána tzv. backoff pravděpodobnost. Ta zajišťuje, že kombinacím, které nebyly nalezeny, není přiřazena nulová pravděpodobnost. V případě nenalezení záznamu pro bigram je použita rovnice 2.15.

$$P(w_i|w_{i-1}) = P(w_i)B(w_{i-1}) \tag{2.15}$$

Kde $P(w_i)$ je unigramová pravděpodobnost a $B(w_{i-1})$ je backoff pravděpodobnost předcházejícího slova.

2.4.3 Rozpoznávání spojité řeči

Rozpoznávání spojité řeči s sebou oproti rozpoznávání izolovaných slov přináší jisté problémy. Rozpoznávač izolovaných slov funguje poměrně přímočaře - modelem jsou posílány získané příznaky a všechny informace z modelu vystupující, krom těch, které zůstanou na konci, jsou odstraněny. V případě rozpoznávání spojité řeči to ovšem není tak jednoduché a je nutné provádět mezislovní přechody.



Obrázek 2.11: Rozpoznávací síť pro spojitou řeč. Jednotlivé modely reprezentují slova v síti, N je počet slov.

Obrázek 2.11 zobrazuje jednoduchý model pro spojitou řeč. Jednotlivé vnitřní modely reprezentují slova ve slovníku rozpoznávače. Na místě výstupu je tzv. mezislovní přechod, který posílá informace zpět na začátek modelu. Kvůli tomu je zvýšena jak paměťová (je nutné ukládat historii slov, kterými se v průchodu modelem prošlo), tak i výpočetní náročnost.

Kapitola 3

Implementace

Tato kapitola popisuje implementaci rozpoznávače řeči. Seznamuje čtenáře s technologiemi využitými v programu, externími knihovnami a funkcemi nutnými pro chod aplikace. Kapitola také obsahuje popis struktury knihovny a informace o jejím rozhraní. Vysvětluje, jakým způsobem byly vytvořeny jednotlivé části rozpoznávače. Pro maximalizaci rychlosti výpočtu je celý proces implementován v NDK, tedy jazyku C++ a v Renderscriptu.

3.1 Použité technologie

Společnost Google, která je vývojářem otevřené platformy Android, poskytuje vývojářům několik různých nástrojů pro tvorbu jejich aplikací. Využité nástroje pro implementaci projektu jsou popsány v této podkapitole.

3.1.1 Android Studio

Android Studio je oficiálním vývojovým prostředím (dále IDE) pro platformu Android. Je založen na IDE IntelliJ IDEA od firmy JetBrains. Podporuje nejpoužívanější operační systémy, tedy Windows, macOS a Linux.

Toto IDE podporuje jak vývoj v Javě, na Androidu nejpoužívanějším jazyku, tak i v nativním prostředí, tedy C++ nebo C. Dále také obsahuje nástroje pro tvorbu uživatelského rozhraní, které byly použity pro tvorbu demonstrační aplikace

3.1.2 Android NDK

Z důvodu neoptimální rychlosti interpretace bytecode pod Androidem je vývojářům umožněno psaní nativního kódu přeloženého specificky pro architektury využívané v mobilních zařízeních.

Překlad nativního kódu je v Android Studio prováděn za pomoci nástroje CMake, případně ndk-build. Ndk-build je ovšem podporován jen z důvodu zpětné kompatibility se staršími projekty. V tomto projektu byl využit CMake.

3.1.3 JNI

Java Native Interface definuje způsob, jakým kód napsaný v Javě interaguje s nativním kódem psaným v C++. Umožňuje tedy připravit rozhraní pro nativní kód.

3.1.4 Renderscript

Renderscript[1] je jazyk založený na standardu C99 jazyka C. Je určen pro spouštění výpočetně náročných operací. Hlavní silou Renderscriptu je jeho soustředění na paralelizaci a velkou výhodou v tomto směru je to, že paralelizace je řízena systémem. Díky tomuto je kód v něm napsaný přenositelný. Zároveň může Renderscript runtime zadat některé části výpočtu i GPU zařízení.

I přes to, že hlavním účelem vytvoření Renderscriptu bylo urychlení zpracování grafických operací, je možné ho využít i pro naše účely. Je to velice užitečný nástroj jak pro paralelizaci extrakce příznaků, tak i pro zrychlení výpočtu neuronové sítě.

Pro generování C++ tříd pro použití skriptů je použit Gradle, ovšem pro překlad zdrojového kódu je nutné využít ndk-build.

3.1.5 OpenSL ES

OpenSL je multi-platformní knihovnou pro zpracování zvukového signálu. Dodává nástroje pro zaznamenání a přehrání zvuku a mnoho dalších vlastností, které v tomto projektu nejsou podstatné.

V NDK se nachází poněkud ošizená verze této knihovny, pro účely této implementace jsou ale její funkce dostatečné. Hlavní nevýhodou této verze je zbytečně složité používání v nativním kódu.

3.1.6 Git

Git je verzovací systém využíván pro týmový vývoj. I přes to, že je tento projekt vypracováván samostatně, je Git velice užitečným nástrojem. Využívám ho převážně pro zálohu zdrojových kódů na vzdáleném serveru a také pro jeho přenos mezi zařízeními.

Jako hostovací server pro svoje úložiště jsem zvolil GitHub[2] kvůli předchozích zkušenostem s tímto serverem ať už pro osobní nebo školní projekty.

3.2 Externí knihovny a funkce

3.2.1 FIR filtr

Pro aplikaci filtru dolní propusti jsem využil třídy Filter[3]. Tato třída implementuje tři základní filtry pro zpracování zvukové stopy - dolní, horní a pásmovou propust. Pro převzorkování vstupního zvukového záznamu je v programu využit pouze filtr dolní propust. Zdrojový kód je distribuován pod modifikovanou BSD licencí.

3.2.2 KissFFT

KissFFT[4] je velmi malá knihovna pro výpočet FFT(Rychlá Fourierova Transformace) napsaná v jazyce C. Byla vybrána z důvodu jednoduchosti jejího použití a naprosto dostatečné rychlosti výpočtu. Knihovna je distribuovaná pod modifikovanou BSD licencí.

[1] https://developer.android.com/guide/topics/renderscript/compute.html

[2] https://github.com

[3] https://cardinalpeak.com/blog/a-c-class-to-implement-low-pass-high-pass-and-band-pass-filters/

[4] http://kissfft.sourceforge.net/

3.3 Struktura knihovny

Program je tvořen jako knihovna pro použití v Android SDK. Pro použití v jiných aplikací tedy němu tedy tvořené žádné grafické rozhraní, slouží pouze pro použití v nějakém větším celku, např. aplikace pro ovládání zařízení za pomocí hlasových příkazů.

Zdrojové kódy knihovny jsou děleny do logických celků, z nichž každý zastává jeden úkol. Následuje popis jednotlivých částí a důležitých tříd v nich obsažených. Diagramy třídy jsou v příloze B.

Feature_extraction Obsahuje třídy obstarávající extrakci příznaků (příloha B.2). Třída `AudioFrame` poskytuje metody pro operace nad rámci audio signálu. Dále také obsahuje třídy `RSMelFilterBank` a `RSNeuralNetwork`, které využívají Rendescript k aplikaci Mel bank filtrů a výpočtu neuronové sítě.

Voice_Activity_Detection Pouze třída `VoiceActivityDetector` pro detekci řečové aktivity (příloha B.1).

Decoder Složka s definicemi tříd pro dynamický dekodér (příloha B.3). Hlavní třídou tohoto balíčku je `ViterbiDecoder`, která zaobaluje funkce všech ostatních tříd. Dále je zde obsažena třída `HMMGraph`, která definuje strukturu rozpoznávací sítě a také třída `Token` reprezentující tokeny algoritmu Token Passing.

Threads Zahrnuje třídy pro vlákna a komunikaci mezi nimi a dále také callbacky do virtuálního prostředí Javy (příloha B.4).

Utility Třídy s různými účely, např. čtení WAV souboru nebo nahrávání audio signálu (příloha B.5).

Dále je zde obsažena třída `SpeechRecognitionAPI` poskytující rozhraní pro využívání knihovny. Umožňuje programátorovi využít možnosti nahrávání a současného rozpoznávání zvukového signálu nebo signálu z WAV souboru. Obdobná třída je implementována i v Javě. Obě tyto rozhraní využívají návrhového vzoru observer pattern[5] pro notifikaci posluchačů o změnách v rámci rozpoznávání (změna detekované aktivity řeči, rozpoznání věty a dokončení rozpoznávání).

3.4 Nahrávání a zpracování zvukové stopy

Pro nahrávání zvukového signálu jsem využil knihovnu OpenSL. Audio je nahráváno do bufferu ve vzorkovací frekvenci 48 kHz a v bitové hloubce 16 bitů. Při nahrání určité délky nahrávky, v tomto případě délka překryvu rámce, tedy 10 ms, je vyvolán callback a data jsou předána do vlákna pro extrakci příznaků.

Po předání dat je signál převzorkován na požadovanou frekvenci - 8 kHz. Dále je převeden z datového typu short int na float, jelikož další části rozpoznávače pracují s reálnými čísly.

[5] http://w3sdesign.com/?gr=b07&ugr=proble

3.4.1 Zpracování ze souboru

V případě zpracování ze souboru je proces poněkud jednodušší. Jediné operace, které je nutné vykonat je kontrola formátu vstupního souboru. Tato implementace podporuje pouze soubory uložený ve formátu WAV, které splňují následující požadavky: vzorkovací frekvence musí být 8 kHz, audio obsahuje pouze jeden kanál (mono nahrávka), bitová hloubka je 16 bitů a uložená data jsou ve formátu PCM.

Následné zpracování probíhá obdobně jako při nahrávání. Do vlákna extrakce příznaků je odesílán signál rozdělený na části o délce 10 ms.

3.5 Extrakce příznaků

Extrakce je prováděna souběžně s nahráváním zvuku. V prvotních fázích implementace nebylo možné provádět kompletní extrakci zároveň s nahráváním, ovšem po využití průběžné normalizace energií popsané níže byla tato limitace odstraněna.

Segmentace zvukového signálu je částečně řešena za pomoci callback funkce, kterou vyvolává vlákno pro nahrávání audio signálu, jelikož je předána délka záznamu odpovídající délce překryvu jednotlivých rámců. Pro účely uložení dat rámce a jeho další zpracování je vytvořena třída AudioFrame.

Dalším krokem extrakce je aplikace Hammingova okna (2.2.1). Jedná se o jednoduchou operace násobení, kterou není nutné nijak optimalizovat. Koeficienty okna je nutné počítat pouze jednou a to při inicializaci rozpoznávače.

Následuje aplikace Rychlé Fourierovy transformace. Knihovna KissFFT přijímá na vstup rámec signálu a vrací výsledek FFT o určité délce. Vstupní data transformace mají délku 256 vzorků - jedná se o rámec signálu (200 vzorků) a 56 nulových hodnot.

Všechny tyto kroky jsou implementovány v již zmíněné třídě AudioFrame. Ta v sobě uchovává dílčí výsledky a uvolňuje paměť v momentě, kdy už data nejsou potřebná.

Následujícím, výpočetně náročnějším krokem je výpočet mel bank filtrů. K akceleraci výpočtu je využit Renderscript. Díky Rendersriptu je každý kanál mel bank počítán paralelně a rychlost výpočtu je výrazně vyšší, než při sekvenčním výpočtu. Aplikaci mel bank filtrů obstarává třída RSMelFilterBank. Aplikace jednoho filtru se řídí algoritmem 1, je implementována v Renderscriptu a zpracování všech filtrů mel bank může potenciálně běžet souběžně.

Algoritmus 1: Aplikace mel filtru

```
Function MelFilter(index_filtru, fft_vstup):
    for i ← 0 to Délka FFT dat do
        result = result + mel_filtry[index_filtru][i] * (fft_vstup[i].real +
          fft_vstup[i].imaginary)
    end for
    if result < 1 then
        result = 0
    else
        result = log(result)
    end if
    return result;
```

Parametry jednotlivých filtrů jsou počítány před zahájení extrakce a tento výpočet není nutné opakovat. Po aplikaci všech filtrů jsou výsledky odeslány do detektoru řečové aktivity, který je popsán v následující sekci.

Normalizace získaných dat probíhá pouze na rámcích detekovaných jako aktivní a je řešena odečtením celkového průměru v jednotlivých filtrech, tedy:

$$y_n = x_n - \sum_{i=0}^{n-1} h_i \tag{3.1}$$

Kde x_n je vstup, y_n je výstup a h jsou předchozí vstupy. Průměr samozřejmě není v každém kroku vypočítán na základě všech předchozích dat, ale je využíván tzv. klouzavý průměr (angl. moving average, rovnice 3.2).

$$AVG_{n+1} = \frac{n \cdot AVG_n + x_{n+1}}{n+1} \tag{3.2}$$

Kde AVG_{n+1} je nový průměr, n je počet zprůměrovaných hodnot, x_{n+1} je nová hodnota pro výpočet.

3.6 Detekce řečové aktivity

Důvody pro využití detekce řečové aktivity jsou dva. Prvním z nich je zlepšení normalizace řečníka, která je popsána v předchozí kapitole. Druhým důvodem je možnost deaktivovat výpočet neuronové sítě a dekodér v časových okamžicích, ve kterých není mluva detekována. To samozřejmě výrazně snižuje časovou náročnost zpracování.

Samotná detekce je implementována poměrně jednoduchým způsobem. Aktivita je určována podle výstupů mel bank filtrů, tedy energií nahrávky v určitých částech spektra. V případě, že je součet všech logaritmů energií nižší než 0, je rámec považován za ticho. Formální popis je v rovnici 3.3.

$$f(\vec{n}) = \begin{cases} true & \text{jestli } \sum_{i=0}^{N} n_i > 0 \\ false & \text{jinak} \end{cases} \tag{3.3}$$

kde N je velikost vstupního vektoru a n je vstupní vektor

Určení aktivity pouze pro jednotlivé rámce by ovšem nebylo dostatečné. Často by docházelo k tomu, že za aktivní segmenty by mohly být považovány i velmi malé časové úseky. Proto detektor umožňuje nastavit hranice, kterými je určen minimální časový limit pro přechod z jednoho stavu do druhého. Vhodné hodnoty byly zjištěny experimentálně. Pro přechod ticho→řeč se ukázala vhodnou hodnota 7 rámců a pro přechod řeč→ticho 25 rámců.

Z důvodu zpožděných přechodů detektor obsahuje vlastní zásobník předchozích rámců, které jsou dále využity v případě přechodu do aktivního stavu. Pokud by tento zásobník nebyl implementován, mohlo by docházet k velké ztrátě informací.

3.7 Klasifikace

Klasifikace dat na fonémy je implementována prostřednictvím neuronové sítě. Již natrénovaná neuronová síť mi byla dodána vedoucím práce, proto v této práci není její tréno-

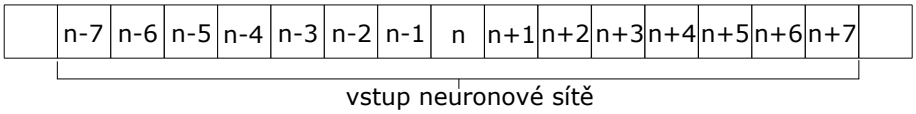
vání pokryto. Tato neuronová síť byla trénovaná na trénovací sadě TED-LIUM pocházející z Kaldi receipe [5].

Tabulka 3.1: Parametry neuronové sítě

Vrstva	Velikost	Aktivační funkce
Vstup	360	-
1	500	sigmoida
2	500	sigmoida
3	80	-
4	500	sigmoida
5	46	softmax

Síť používá jako aktivační funkci sigmoidu popsanou v podkapitole 2.3. Pouze třetí, bottleneck, vrstva nevyužívá žádné aktivační funkce. Na konečný výstup sítě je aplikována funkce softmax pro normalizaci výstupních hodnot (rovnice 3.4). Jelikož dekodér pracuje s logaritmy pravděpodobností, je při výpočtu výsledek funkce softmax zlogaritmován.

$$y_n = \frac{\exp(x_n)}{\sum_{i=0}^{N} \exp(x_i)} \tag{3.4}$$



Obrázek 3.1: Vstupní data neuronové sítě

Vstupem sítě jsou výsledky mel bank filtrů. Z matice výsledků jsou postupně brány jednotlivé rámce a jejich okolí – přesněji 7 rámců. V případě, že některé rámce nejsou dostupné (např. při začátku promluvy) je vícekrát využit první resp. poslední dostupný rámec. Data je dále také nutné přeorganizovat, jelikož síť očekává, že kanály jednotlivých mel bank budou přímo následovat.

Samotná síť je implementována ve třídě RSNeuralNetwork. K urychlení výpočtu je opět využit Renderscript. Bohužel v současné verzi Renderscriptu není možné interně synchronizovat vlákna, proto bylo nutné algoritmus sítě rozdělit do více částí, jinak by mohlo dojít k desynchronizaci dat a výsledky výpočtu by byly nekonzistentní – výpočet další vrstvy by mohl začít předtím, než se připraví data z vrstvy předchozí. Při výpočtu výstupů každé vrstvy je tedy nutné za pomocí NDK čekat na dokončení výpočtu. Data ovšem není nutné kvůli tomuto nijak zpracovávat a mohou zůstat v prostředí Renderscriptu, takže tento proces nepřidává téměř žádnou režii.

Paralelizace algoritmu neuronové sítě je řešena následujícím způsobem. Namísto výpočtu pomocí maticových operací jsou Renderscriptu předány hodnoty naučené sítě a dat pro zpracování a hodnoty jsou vypočteny pouze pro daný neuron. Poté je potenciálně pro každý neuron spuštěno jedno vlákno - záleží na tom, jak se Renderscript Runtime rozhodne práci rozdělit, tohle nemůže programátor nijak řídit. Oproti sekvenčnímu výpočtu v C++ je tento přístup přibližně 5-krát rychlejší. Bližší informace o srovnání rychlosti výpočtu lze nalézt v podkapitole 4.2.

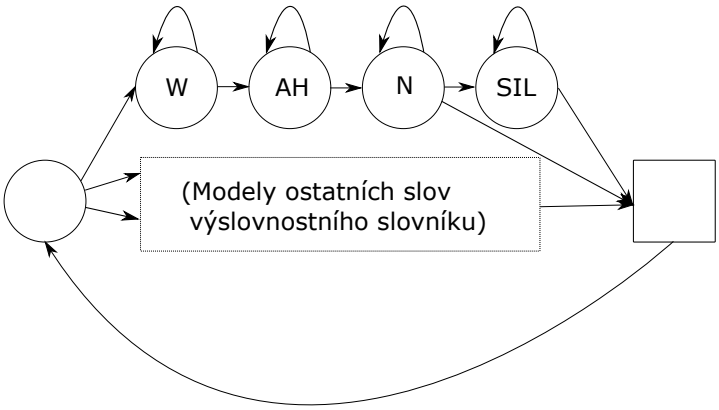
3.8 Dekodér

Jak již bylo zmíněno v podkapitole 2.4, dekodér je implementován za využití skrytých Markovových modelů (dále HMM). HMM jsou reprezentovány formou stromu se zpětnými ukazateli. Každý uzel grafu tedy obsahuje ukazatele jak na všechny svoje přímé předchůdce, tak i přímé následovníky.

V této implementaci je za účelem snížení paměťové náročnosti implementován dynamické dekodér. Ten, na rozdíl od dekodéru statického, vytváří pouze unigramovou rozpoznávací síť a získává hodnoty n-gramových pravděpodobností z jazykového modelu uloženého paměti ve vyhledávací struktuře, řešení uložení jazykového modelu je popsáno v sekci 3.8.2.

Každý uzel grafu v sobě udržuje informace o pravděpodobnostech přechodu do dalších stavů - přesněji logaritmy těchto pravděpodobností - dále také index výstupu neuronové sítě, který určuje jaký foném daný uzel přijímá na svém vstupu. Z důvodu urychlení přístupu k nejlépe hodnocenému tokenu je zde také obsažen ukazatel na nejlepší token.

V grafu jsou definovány dva uzly se speciálním účelem. První z nich je počáteční uzel, ten slouží k předávání tokenů vstupním stavům jednotlivých slov. Druhým je výstupní uzel, ten je využit pro přidání záznamu Word link record (3.8.1) a výpočtu $P(W)$. Zjednodušená grafická reprezentace grafu je na obrázku 3.2.



Obrázek 3.2: Grafická reprezentace rozpoznávací sítě se slovem "one"a přidaným SIL na konec slova

Celý graf je vytvořen při inicializaci rozpoznávače na základě slovníku a jazykového modelu uloženého v paměti zařízení. Pro každé slovo je na konec přidán alternativní uzel přijímací na svém vstupu pravděpodobnost výskytu ticha. Lepším řešením by pravděpodobně bylo přidat na výstup všech slov sdílený uzel přijímající ticho, jelikož tokeny v každém uzlu SIL již nepřijmou před výstupem do dalšího slova žádný jiný foném. Tím by se dal snížit počet stavů HMM grafu o počet slov ve slovníku a o stejné množství i počet aktivních tokenů.

3.8.1 Token passing

Token passing je široce využívaným algoritmem pro zjištění nejpravděpodobnější cesty skrze HMM. Lze jej využívat jak pro rozpoznávání izolovaných slov nebo, jako v případě této

práce, pro spojitou řeč. Popis lze nalézt v algoritmu 2. Popis algoritmu byl převzat z článku [7].

Algoritmus 2: Token passing

Inicializace:
Počáteční stavy modelu obsahují token s ohodnocením 0;
Ostatní stavy obsahují token s nekonečnou cenou.
Algoritmus:
for $t \leftarrow 0$ **to** T **do**
  foreach *stav* i **do**
    Nakopíruj token ze stavu i do všech napojených stavů a přepočítej jeho ohodnocení.
  end foreach
  Odstraň původní tokeny.
  foreach *stav* i **do**
    Najdi ve stavu i token s nejlepším ohodnocením a odstraň zbytek.
  end foreach
  foreach *výstupní stav* i **do**
    Vytvoř nový záznam Word link record pro každý aktivní stav i.
    Navaž záznam na ten uložený v tokenu.
    Změň ukazatel cesty tokenu na tento nově vytvořený záznam.
  end foreach
end for
Výsledek:
Z výstupních stavů vyber nejlépe ohodnocený token. Tento token určuje nejpravděpodobnější cestu grafem.

Velmi důležitou částí tohoto algoritmu je tzv. **Viterbiho kritérium**. To udává, že v rámci jednoho stavu lze uchovat pouze token s nejlepším ohodnocením a ostatní zahodit. Tokeny s horším ohodnocením nemohou být v dalších krocích nikdy hodnoceny lépe, než ten momentálně nejlepší a proto by byl jejich výpočet zbytečný.

V první verzi dekodéru byly tokeny implementovány jako objekty, které se při každém přechodu v paměti alokovaly a dealokovaly. Tento přístup se ukázal být velmi neefektivní kvůli vysoké režii, která tím byla do programu uvedena. Z tohoto důvodu bylo nutné tento problém vyřešit jinak.

Ve finální verzi jsou v rámci každého uzlu grafu alokovány tokeny již při vytvoření rozpoznávacího grafu a to tak, že pro každý uzel je vytvořeno tolik tokenů, kolik je do stavu vedeno přechodů. Díky tomu mohou být tokeny uchovány v paměti neustále, i když to způsobuje vyšší paměťovou náročnost. Namísto vymazání tokenu (jak aplikací Viterbiho kritéria tak i pruningu) je pouze nastaven příznak neaktivity. Tento přístup vyžaduje procházení grafem od konce, jelikož nové hodnoty tokenů závisí na předcházejících uzlech. Díky tomuto přístupu je také možné aplikovat Viterbiho kritérium již po dokončení výpočtu v každém uzlu, jelikož nově vypočtené hodnoty nejsou v tomto kroku dále využity.

Všechny pravděpodobnosti jsou v rozpoznávači reprezentovány jejich logaritmy pro možnost využití rychlého sčítání namísto pomalého násobení (rovnice 3.5). Po dokončení výpočtu není nutné logaritmy opět převádět do běžných pravděpodobností, stačí vybrat ten nejlépe ohodnocený.

$$\log(P_1 P_2) = log(P_1) + log(P_2) \tag{3.5}$$

Pruning

S využitím pruningu je možné opět snížit časovou náročnost programu. Pruning, podobně jako Viterbiho kritérium, umožňuje deaktivaci těch tokenů, které mají malou šanci na úspěch, alespoň v daném časovém okamžiku. Nevýhodou jeho využití je ovšem možnost deaktivace tokenů, jež mohou v dalších krocích získat lepší ohodnocení. Něco takového ovšem nelze předvídat. Typy pruningu využité v této implementaci jsou **beam pruning** a **live states pruning**. Popis vykonání pruningu je v algoritmu 3.

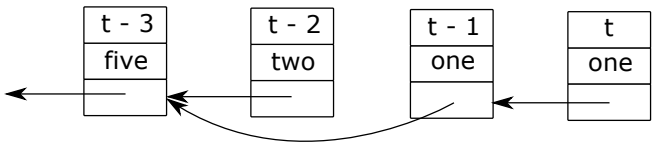
Algoritmus 3: Pruning

```
/* Seznam "aktivní tokeny" obsahuje všechny aktivní tokeny po aplikaci
   Viterbiho kritéria                                                 */
```
Procedure `ApplyPruning()`**:**
Seřaď aktivní tokeny podle jejich ohodnocení, od nejlepšího.
`// beam pruning`
hranice = ohodnocení nejlepšího tokenu + konstanta beam pruningu
foreach *aktivní token t* **do**
  Pokud je ohodnocení tokenu t nižší než hranice, deaktivuj ho.
end foreach
`// live states pruning`
Deaktivuj všechny tokeny krom N nejlepších.

Word link record

Důležitou součástí rozpoznávání spojité řeči pomocí token passingu je vhodné uložení historie slov pro výpočet $P(W)$. V prvotní fázi jsem využíval objektů z knihovny STL[6], tento přístup se ovšem osvědčil jako velice neefektivní. V některých případech bylo nutné kopírovat kompletní historii a to způsobovalo velkou režii. Proto je historie ve finální verzi implementována za pomocí word link record.

Word link record v principu funguje jako jednosměrně vázaný seznam, ovšem pouze zpětně. Jednotlivé záznamy slov jsou reprezentovány nezávislými objekty. Pro každý token procházející slovním přechodem je vytvořen nový záznam a je navázán na poslední záznam uložený v daném tokenu. Záznamy v historii mohou být sdíleny v historii více tokenů, jak je vidět na obrázku 3.3 – každý záznam bez příchozího ukazatele je poslední v historii a je posledním záznamem v některém z tokenů.



Obrázek 3.3: Příklad provázání word link record

Po dokončení rozpoznávání lze snadno zpětným průchodem záznamy zjistit, kterými slovy token s nejlepším ohodnocením prošel. Kvůli usnadnění dealokace již nevyužívaných

[6] http://en.cppreference.com/w/cpp/container

záznamů obsahuje každý objekt počítadlo referencí. World link record se tedy sám vymaže z paměti poté, co již není v historii žádného tokenu.

3.8.2 N-gramy

Jak již bylo zmíněno v 2.4.2, v této implementaci jsou pro jazykový model využity n-gramy. Specificky jsou podporované jazykové modely s bigramy. Informace o jazykovém modelu jsou uloženy v souboru formátu ARPA.

Při inicializaci rozpoznávače jsou N-gramy načteny ze souboru uloženého v paměti zařízení. Pro rychlejší přístup k potřebných datům je každé slovo reprezentováno svým ID, tedy indexem, pod kterým se nachází ve vektoru všech slov. To zajišťuje přímý přístup do paměti při hledání daného slova. Každé slovo v této struktuře využívá k uložení bigramových pravděpodobností hashovací tabulku pro rychlé dohledání potřebného záznamu (opět podle ID slova). Hashovací tabulka sice zabírá více paměti než primitivnější způsoby uložení, ale výrazně snižuje dobu přístupu k potřebnému záznamu.

V případě nenalezení bigramu v hashovací tabulce je dohledána backoff pravděpodobnost předchozího slova přímým přístupem pomocí ID slova.

Ohodnocení přechodu mezi slovy je také měněno tzv. Word insertion penalty a je možné nastavit i škálování pravděpodobností n-gramů. Výpočet $P(w)$ se řídí rovnicí 3.6.

$$P(w_i) = P(w_i|w_{i-1})\text{LM scale} + \text{word insertion penalty} \tag{3.6}$$

3.9 Vlákna

Pro zrychlení výpočtu a využití více-jádrových CPU obsažených v Android zařízeních je exekuce rozpoznávání rozdělena do několika vláken.

Prvním z nich je vlákno, které dodává vstupní audio data. V případě nahrávání z mikrofonu se tedy jedná o nahrávací vlákno, které předává každých 10 ms nahraného audio signálu. Čtení ze souboru funguje obdobným způsobem. Tato data jsou předávána do vlákna extrakce příznaků. Další vlákno obstarává extrakci příznaků (3.5) a detekci řečové aktivity(3.6) a posílá výsledek do vlákna neuronové sítě. Třetím vláknem je klasifikace za využití neuronové sítě (3.7) a posledním je samotný dekodér (3.8).

Předávání dat mezi vlákny

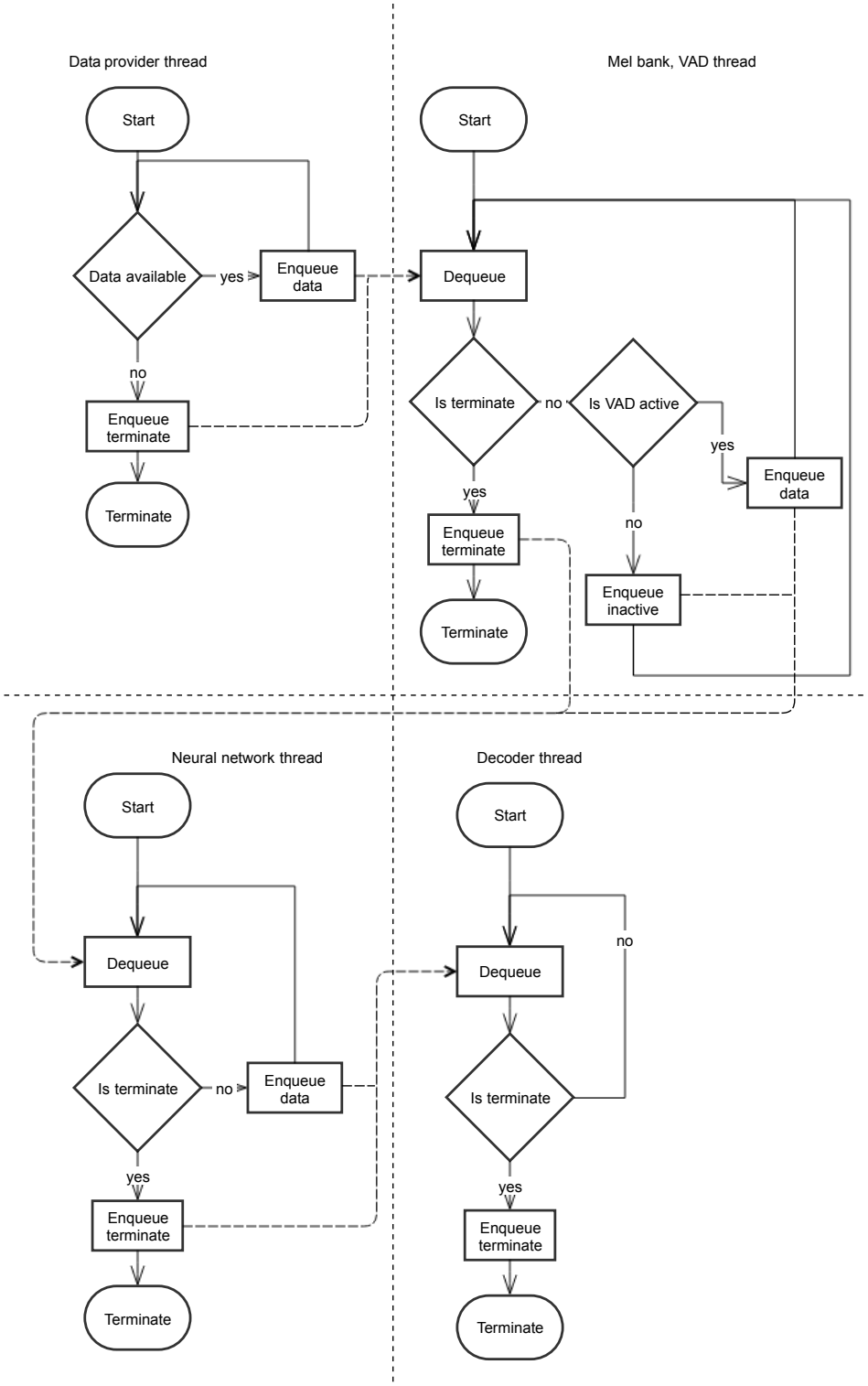
Jelikož mezivláknová komunikace může způsobovat neočekávané chování, pokud není správně implementována, byla pro účely předávání dat vytvořena třída `SafeQueue`. Tato třída za pomocí zamykacích objektů (mutexů) umožňuje přístup k objektu fronty pouze jednomu vláknu při souběžném přístupu. Dále tato třída využívá `std::condition_variable` pro notifikaci vláken čekajících na další data. Tato implementace je inspirována návrhovým vzorem Producer/Consumer[7]. Zjednodušený diagram komunikace mezi vlákny je na obrázku 3.4.

Typ předávaných dat se řídí výčtovým typem. Tyto typy jsou:

- DATA - audio data
- INACTIVE - detekováno ticho

[6] http://www.speech.sri.com/projects/srilm/manpages/ngram-format.5.html
[7] http://www.ni.com/white-paper/3023/en/

- TERMINATE - konec vstupních dat, ukončení vláken



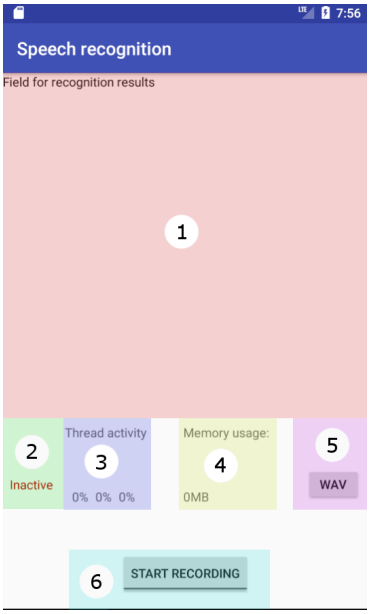
Obrázek 3.4: Zjednodušená grafická reprezentace mezivláknové komunikace

3.9.1 Komunikace s JVM

Pro odesílání výsledků rozpoznávání a notifikování o událostech, které v kódu nastaly je nutné implementovat funkce, které dokáží předávat data z nativního kódu do JVM. Pro tento účel byla využita knihovna JNI (3.1.3). Při tvorbě rozpoznávače je uložena globální reference na objekt, který poskytuje rozhraní k použití rozpoznávače. Tento objekt v sobě obsahuje privátní metody, které jsou volány z nativního kódu. Tímto způsobem je možné i předávat data pomocí parametrů funkcí tak, jak je běžné.

3.10 Demo aplikace

Pro účel předvedení funkčnosti rozpoznávače byla vytvořena jednoduchá Android aplikace. Tato aplikace umožňuje rozpoznávání jak přímým audio vstupem z vestavěného mikrofonu zařízení, tak i z audio nahrávek uložených v jeho paměti. Cílem nebylo vytvořit aplikaci s příjemným uživatelským rozhraním, ale spíše demonstrovat funkčnost a zobrazit důležité informace.



Obrázek 3.5: Demo aplikace

UI aplikace se dělí na šest základních částí, následuje jejich popis:

1. pole pro výsledky rozpoznávání,
2. aktuální stav detekce řečové aktivity,
3. aktivita jednotlivých vláken rozpoznávače (zleva mel bank filtry a detekce řečové aktivity, neuronová síť, dekodér),
4. aktuální využití paměti aplikací, včetně grafických prvků a paměti alokované v JVM,
5. tlačítko pro demonstraci rozpoznávání z WAV souboru,
6. tlačítko pro zapnutí/vypnutí nahrávání z mikrofonu.

Aplikace je lokalizována pro český a anglický jazyk a sama se přizpůsobí nastavení systému.

Kapitola 4

Testování a vyhodnocení

V této kapitole je popsán proces verifikace výsledků implementovaných částí a srovnání implementací některých částí rozpoznávače v jazycích Java, C++ a Renderscript. Také je zde obsaženo zhodnocení úspěšnosti rozpoznávání a vyhodnocení časových a paměťových nároků rozpoznávače.

Všechna měření v této kapitole (krom hodnocení word error rate) byla prováděna na zařízení Lenovo A7020a48[1]. Verze operačního systému Android na tomto zařízení je 6.0. Jedná se o zařízení starší a střední cenové třídy, proto se dá předpokládat, že na novější zařízeních poběží rozpoznávač přinejmenším stejně rychle.

4.1 Verifikace algoritmů

Pro verifikaci výsledků jsem využil program MATLAB[2] a funkce v něm obsažené, případně skripty vytvořené komunitou. Dále byly výsledky porovnány s referenční implementací dodanou vedoucím práce. Referenční kód byl dodán v několika jazycích, pro srovnání jsem využil verze napsané v jazycích Python a Java.

Kód napsaný v Javě bylo možné spuštět přímo na platformě Android a proto byl hlavním zdrojem kontrolních dat. Také tato implementace využívá datový typ float, díky čemuž výsledky lépe odpovídají této implementaci.

Algoritmy napsané v C++ odpovídají výstupům referenční implementace, přesně podle očekávání. Implementace v Renderscript ovšem vykazují jistou odchylku oproti referenčním výstupům i přes to, že je ve skriptech definována nejvyšší možná přesnost výpočtu. Odchylka je ale velmi malá, v řádu desetin procent, tedy zanedbatelná.

4.2 Srovnání implementací

Srovnání rychlosti výpočtu bylo provedeno na dvou výpočetně nejnáročnějších částech extrakce příznaků - výpočet mel bank filtrů a neuronové sítě.

Pro účel srovnání rychlostí byla vytvořena audio nahrávka o délce 11 sekund, převzorkovaná na vzorkovací frekvenci 8 kHz. Pro přesné údaje byl každý algortimus spuštěn 50x a na testovacím zařízení bylo spuštěno minimum procesů, které by mohly výsledky testů ovlivnit.

[1] www.mobilespecs.net/phone/Lenovo/Lenovo_Vibe_K5_Note_A7020a48.html
[2] www.mathworks.com/products/matlab.html

Tabulka 4.1: Rychlosti výpočtů

	Mel banky filtrů	Neuronová síť
Java	1837 ms	22009 ms
C++	447 ms	12223 ms
Renderscript	229 ms	2735 ms

Jak je vidět v tabulce 4.1, rychlost výpočtu je v nativním kódu výrazně vyšší než ve virtuálním prostředí Javy. Tohle samozřejmě není žádným překvapením.

Pro srovnání rychlosti C++ a Renderscript slouží především výpočet mel bank filtrů. Vzhledem k výraznému rozdílu jsem se rozhodl Renderscript použít i pro výpočet neuronové sítě, implementace v C++ ovšem není příliš optimalizovaná a slouží spíše ke srovnání s Javou.

Díky Renderscriptu je tedy možné provádět obě testované části - jak mel filtr banky, tak výpočet neuronové sítě - zároveň s nahráváním zvukové stopy i bez výrazného zatížení CPU či GPU, pokud ho v daném zařízení Renderscript Runtime využívá.

Srovnání rychlosti výpočtu neuronové sítě bylo prováděno na menší síti použité v začátku vývoje pro srovnání s referenční implementací.

4.3 Hodnocení úspěšnosti rozpoznávání

Pro hodnocení úspěšnosti byl využit tzv. word error rate (WER). Výpočet se řídí rovnicí 4.1. Pro výpočet WER jsem využil skriptu asr_evaluation[3].

$$WER = \frac{S + D + I}{N} \tag{4.1}$$

Kde S je počet substitucí, D počet smazaných slov, I počet vložených slov a N je počet referenčních slov.

Přesnost rozpoznávače je tedy následně dána rovnicí 4.2.

$$WAcc = 1 - WER = \frac{N - S - D - I}{N} \tag{4.2}$$

Měření těchto hodnot bylo prováděno spouštěním extrakce příznaků přímo na Android zařízení, vzhledem k využití Renderscriptu a nemožnosti tedy tuto část spustit na jiném operačním systému než Android. Kód dekodéru je ovšem implementován v c++14 a pro ušetření času a možnosti využít většího slovníku než na mobilním zařízení.

Velikost slovníku využita pro hodnocení:

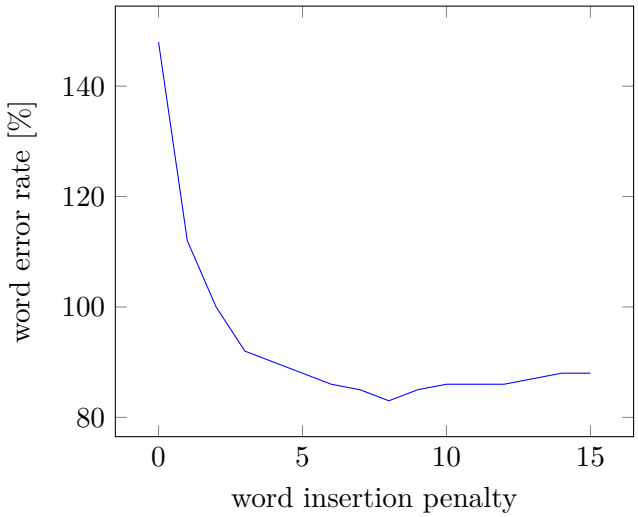
Tabulka 4.2: Velikost slovníku

unigramy	bigramy	výslovnostní slovník
10003	86128	12046

Pro testovací data byla využita sada TED-LIUM obsahující nahrávky TED přednášek. Tato testovací sada obsahuje přibližně 3 hodiny nahrávek.

[3] https://github.com/belambert/asr-evaluation

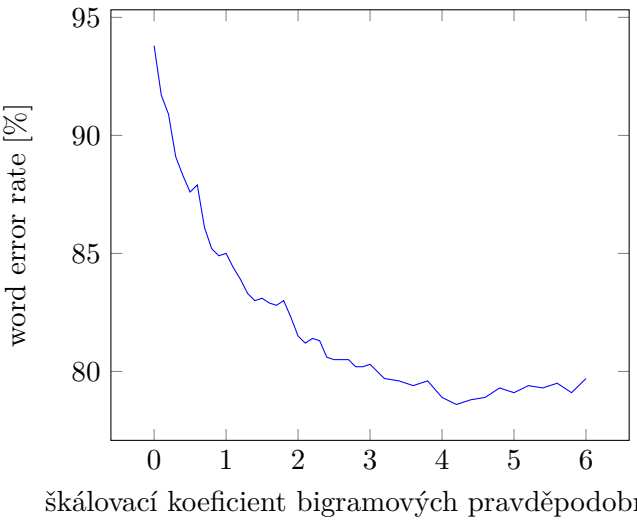
Prvním experimentem bylo zjištění vhodné hodnoty pro word insertion penalty. Důvod využití této hodnoty je popsán v 3.8.2. Pro testování byly zvoleny hodnoty v intervalu $\langle 0, -15 \rangle$ s krokem 1 a škálovací koeficient jazykového modelu 1. Výsledky testů jsou v obrázku 4.1.



Obrázek 4.1: Vliv word insertion penalty na WER

Z grafu vyplývá, že při nepoužití této konstanty je rozpoznávání na velice špatné úrovni. Rozpoznávač bez ní rozpoznává jako vhodná i velice krátká slova a výstup je prakticky nečitelný. Jako optimální se ukázala být hodnota $word\_insertion\_penalty = 8$ a ta je použita i v následujících testech. Vyšší hodnoty vykazují mírné zhoršení kvality rozpoznávání.

Další experiment pokrývá nalezení vhodné hodnoty škálovacího koeficientu jazykového modelu. Otázkou je, zda-li je nutné tento koeficient vůbec používat. V testech byly použity hodnoty z intervalu $\langle 0, 6.0 \rangle$ s krokem 0.1. Nejvhodnější hodnotou se ukázalo být 4.1.



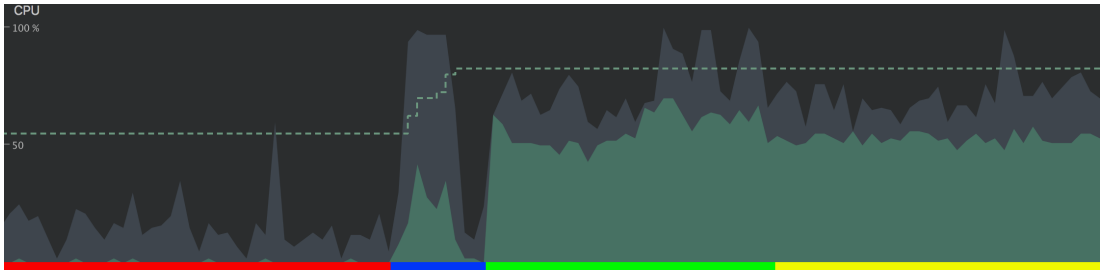
Obrázek 4.2: Vliv škálování jazykového modelu na WER

Další změny parametrů již neměli žádný pozitivní vliv na kvalitu rozpoznávání. Průměrný word error rate rozpoznávače je tedy přibližně 80%.

Jak již bylo zmíněno na začátku této sekce, velikost slovníku použita k vyhodnocení obsahuje přibližně 10000 slov. Tento slovník sloužil především ke zhodnocení kvality samotného dekodéru, pro použití na mobilních zařízení je ale taková velikost slovníku pro tuto implementaci nereálná. Reálné hodnoty pro použití v mobilních zařízeních jsou popsány v následující sekci.

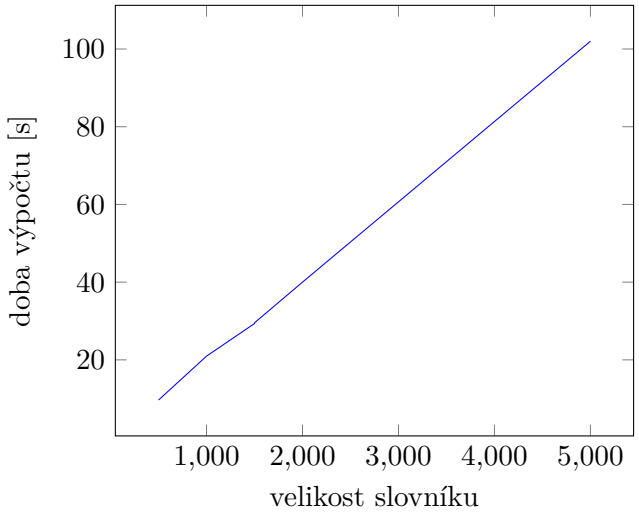
4.4 Časová a paměťová náročnost

Obrázek 4.3 zobrazuje aktivity procesoru v různých fázích rozpoznávání. Červená barva určuje část, ve které probíhá pouze detekce řečové aktivity (a k tomu nutný výpočet mel filtrů). Z grafu je jasně vidět, že tato část má minimální nároky na procesor. Modrá část zobrazuje začátek zpracování dat pomocí neuronové sítě při detekování řečové aktivity před samotnou aktivací dekodéru. Zeleně vyznačená oblast kombinuje jak extrakci příznaků, tak i souběžné dekódování dat. Žlutě zvýrazněná oblast již určuje pouze dekódování.



Obrázek 4.3: Aktivita CPU při rozpoznávání, pořízeno pomocí Android Profiler

Z tohoto obrázku je jasně vidět, že zdaleka výpočetně nejnáročnější částí celé implementace je dekódování. Výsledky testů doby výpočtu lze nalézt v grafu 4.4. Z grafu je vidět, že zdvojnásobení velikosti slovníku vede přibližně k dvojnásobně delší době výpočtu. Počet bigramů nemá na výpočetní dobu příliš velký vliv, při velikosti slovníku 1500 slov a počtu bigramů 77939 resp. 179832 je rozdíl v době výpočtu pouze 0.2 s.



Obrázek 4.4: Vliv velikosti slovníku na dobu výpočtu

Z grafu tedy vyplývá, že nejvhodnějším kandidátem pro rozpoznávání v reálném čase je slovník o velikosti 1500 slov.

Tabulka 4.3 zobrazuje využití paměti nativním kódem po načtení a alokování zdrojů pro různé slovníky. Z tabulky je jasně vidět, že program vyžaduje velmi malé množství operační paměti pro svoje fungování.

Tabulka 4.3: Využití paměti při různých velikostech slovníku

Velikost slovníku	Počet bigramů	Využití paměti [MB]
20	0	8
500	63706	11.2
2000	81741	14
5000	83128	19.4
10000	83128	38.9

Kapitola 5

Závěr

Cílem práce bylo implementovat jednotlivé části rozpoznávače řeči za použití nízkoúrovňových technologií platformy Android, cíl byl tedy splněn.

V teoretické části dokumentu bylo popsáno, jak jsou zvuková data reprezentována ve výpočetních systémech a jakým způsobem se s nimi pracuje. Je zde vysvětlen princip extrakce příznaků a vysvětluje jednotlivé kroky zmíněné extrakce. Dále popisuje princip dekódování za využití skrytých Markovových modelů a n-gramových jazykových modelů.

V implementační části je uvedeno, jakým způsobem jsou data získána a dále zpracována. Blíže specifikuje, jak jsou data předávána v rámci programu a vysvětluje, jakým způsobem jsou reprezentována. Také poukazuje na některé nedostatky implementace. Dále jsou v dokumentu uvedena srovnání implementace extrakce příznaků v různých jazycích, přičemž bylo zjištěno, že nejlepší možností je kombinace C++ a Renderscriptu.

Implementace extrakce příznaků, která byla realizována za použití NDK a Renderscriptu dodává stejné výsledky jako referenční implementace a doba jejich výpočtu je více než dostačující pro souběžné zpracování s nahráváním zvukového signálu. Výpočetně nejnáročnější částí rozpoznávače je dekodér.

Kvůli vysoké výpočetní náročnosti dekódování audia je největším možným slovníkem pro rozpoznávání řeči v reálném čase slovník o velikosti přibližně 1500 slov. Tato velikost slovníku není zdaleka vhodná pro diktování, proto by bylo vhodnější použít tuto implementaci spíše s menším slovníkem, například pro řízení programu jednoduchými příkazy a přidaným post-processingem pro upřesnění rozpoznávání. Word error rate rozpoznávače se pohybuje na rozmezí $70 - 100\%$. Paměťová náročnost rozpoznávače je velmi malá a má velkou rezervu pro případné změny, které by využily větší množství paměťového prostoru.

Pravděpodobně nejlepším krokem dalšího vývoje by bylo přepracování dekodéru a prozkoumání možnosti využití Renderscriptu v jeho implementaci. Dále by bylo vhodné přidat možnost získání nejen nejlepšího odhadu výsledku rozpoznávání, ale několika nejlepších. Důležitým dalším krokem je také samozřejmě zvýšení úspěšnosti rozpoznávače.

Literatura

[1] Bracewell, R.: *The Fourier Transform & Its Applications*. McGraw-Hill Science/Engineering/Math, 1999, ISBN 0073039381.

[2] Gelfand, S.: *Essentials of Audiology*. Thieme, 2011, ISBN 9781604061550.

[3] Psutka, J.; Müller, L.; Matoušek, J.; aj.: *Mluvíme s počítačem česky*. Academia, 2006, ISBN 80-200-1309-1.

[4] Rabiner, L.; Schafer, R.: *Theory and Applications of Digital Speech Processing*. Pearson, 2011, ISBN 9780136034285.

[5] Rousseau, A.; Deléglise, P.; Estève, Y.: Enhancing the TED-LIUM Corpus with Selected Data for Language Modeling and More TED Talks. In *LREC*, 2014.

[6] Stevens, S. S.: A Scale for the Measurement of the Psychological Magnitude Pitch. *Acoustical Society of America Journal*, ročník 8, 1937, doi:10.1121/1.1915893.

[7] YoungN, J.; RussellJ, H.; ThorntonCambridge, H. S.: Token Passing : a Simple Conceptual Model for ConnectedSpeech Recognition. 1989.

Seznam obrázků

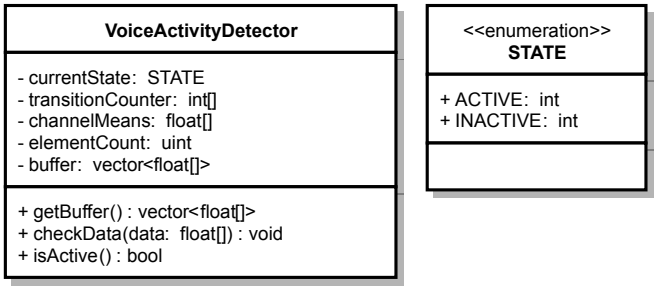
Příloha A

Obsah přiloženého paměťového média

- `README.TXT` – Obsahuje informace o aplikaci a její instalaci
- `BUILD_MANUAL.TXT` – Návod k překladu projektu
- `doc/`
  - `thesis/` – Text této práce a zdrojové kódy LaTeX
  - `doxygen/` – Dokumentace zdrojového kódu C++
  - `javadoc/` – Dokumentace zdrojového kódu Java
- `poster/` – Plakát
- `video/` – Demonstrační video
- `src/` – Zdrojové kódy projektu
- `utils/` – Pomocný program
- `config/` – Dodatečné soubory nutné k běhu projektu

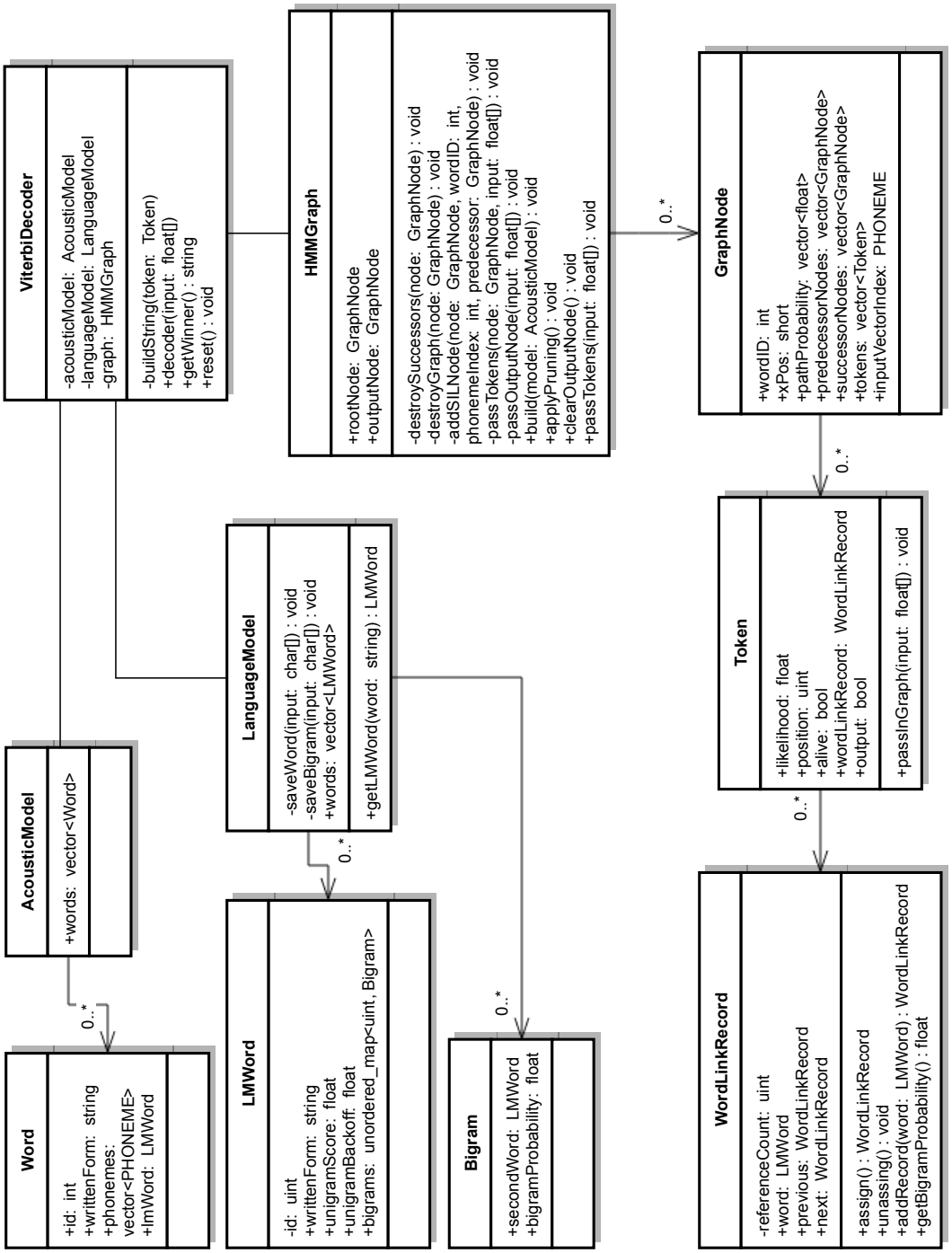
Příloha B

Diagramy tříd

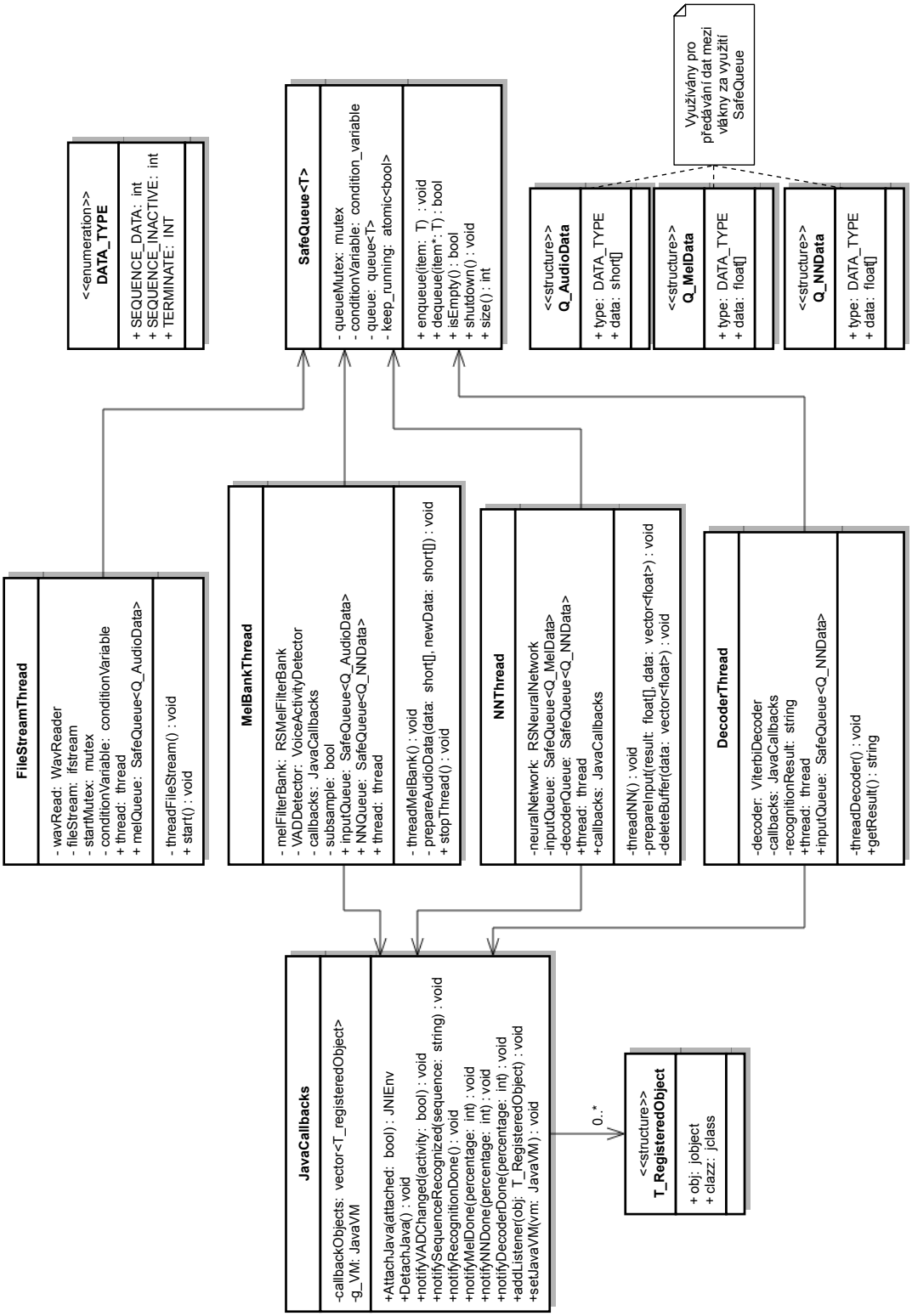


Obrázek B.1: Diagram tříd detekce řečové aktivity

Obrázek B.2: Diagram tříd extrakce příznaků



Obrázek B.3: Diagram tříd dekodéru



Obrázek B.4: Diagram tříd vláken

Obrázek B.5: Diagram pomocných tříd