Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Bine Repše

Identifikacija glasbe v videoposnetkih s pomočjo zvočnih prstnih odtisov

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Matija Marolt

Ljubljana, 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V okviru diplomskega dela izdelajte rešitev, ki analizira videoposnetek in prepozna glasbena dela, ki v njem nastopajo ter čas in dolžino njihove pojavitve. Pri tem preučite področje zvočnih prstnih odtisov in izberite ustrezno knjižnico za indeksiranje glasbe v videoposnetkih. Rešitev naj omogoča tudi navezavo na zunanjo storitev za prepoznavo glasbe in uporabniku omogoča preprost uporabniški vmesnik za delo.

Rad bi se zahvalil vsem, ki ste mi v kakršni koli obliki pomagali med študijem in mi nudili podporo.

Kazalo

Povzetek

Naslov: Identifikacija glasbe v videoposnetkih s pomočjo zvočnih prstnih odtisov

Avtor: Bine Repše

Diplomsko delo opisuje postopek izdelave aplikacije, ki analizira videoposnetke ter s pomočjo zvočnih prstnih odtisov prikaže vse pesmi, ki jih je v posnetku moč slišati, poleg tega pa prikaže, kdaj se posamezna pesem v posnetku prične in kdaj se konča. Aplikacija uporabniku omogoča tudi uporabo zunanje spletne storitve za odkrivanje pesmi, katerih zvočnih prstnih odtisov uporabnik nima shranjenih v podatkovni bazi. Na to storitev se mora uporabnik naročiti. V primeru, ko pa skladbe niso prepoznane, aplikacija uporabniku nudi možnost vnosa podatkov o teh neprepoznanih skladbah ter shranjevanju le teh v podatkovno bazo za zmožnost prepoznavanja v prihodnosti.
V nalogi opišem obstoječe rešitve za prepoznavanje glasbe ter podrobneje opišem delovanje knjižnice, ki sem jo za ta namen izbral. Opišem tudi celoten postopek implementacije, za zaključek pa prikažem še rezultate testiranja aplikacije ter predlagam možne izboljšave za prihodnost.

Ključne besede: zvočni prstni odtisi, videoposnetki, prepoznavanje.

Abstract

Title: Audio fingerprinting for music identification in videos

Author: Bine Repše

The thesis describes the development process of making an application, the purpose of which is to analyze video files and to recognize the songs that appear in the videos, using audio fingerprints. The application delivers the information about where within the duration of a video a certain song begins and where it ends. The application also enables its user the use of an online service used to recognize songs which could not have been recognized using the original, free library. The user has to subscribe to this online service and pay for its use. In case the application is not able to recognize the songs within the recording it allows the user to write the information about the unrecognized songs and to fingerprint them and save them into the database. The thesis describes the solutions for recognizing music and provides a detailed description of getting from an audio file to a set of audio fingerprints using the library of my choice. It also describes the process of making the application and finally, presents some testing results and proposes some possible improvements.

Keywords: audio fingerprints, video, recognition.

Poglavje 1

Uvod

Prepoznavanje glasbe s pomočjo zvočnih prstnih odtisov je dandanes že precej razvito področje. Uporablja se na več različnih načinov ter z različnimi nameni. Najbolj pogost in vsesplošno znan način uporabe te tehnologije je mobilna aplikacija Shazam, ki uporabniku omogoča, da preko vgrajenega mikrofona zajame nek del skladbe, za katerega mu nato Shazam (v kolikor je skladba prepoznana) tudi vrne podatke o tej skladbi. Drug dober primer je uporaba z namenom preprečevanja kršitev avtorskih pravic. Tako preverjanje izvajajo pri podjetju Youtube in s tem preprečujejo neavtorizirano uporabo glasbe v videoposnetkih uporabnikov. Tretji primer pa je uporaba z namenom evidentiranja predvajanih skladb v bodisi radijskih bodisi televizijskih posnetkih raznih oddaj. To pa je tudi namen te diplomske naloge, in sicer zasnovati aplikacijo, ki kot vhod prejme videoposnetek, vrne pa evidenco skladb, ki so se v videoposnetku pojavile.

Obstoječih rešitev za ta namen je več, razvijajo pa jih različni ponudniki. Med najbolj znanimi in najpogosteje uporabljanimi so Echo Nest, ACRCloud ter BMAT. Echo Nest uporabljajo najbolj znana podjetja na tem področju, kot so Spotify, Vevo, BBC in mnogi drugi, vse rešitve pa imajo tudi veliko manj znanih strank. Echo Nest nudi več storitev, kot so prepoznavanje glasbe, avtomatsko kreiranje seznamov predvajanja, priporočanje glasbe glede na okus poslušatelja, prikaz novih informacij o umetnikih, njihovih soci-

alnih interakcijah, fotografijah v stilu socialnih omrežij, pridobivanje zvočnih prstnih odtisov ter avtomatsko preurejanje skladb in vizualizacijo glasbe. ACRCloud prav tako nudi prepoznavanje glasbe, poleg tega pa omogoča še nadzorovanje radijskih in televizijskih prenosov, v katerih prepoznava glasbo ter oglase, prepoznavanje televizijskih kanalov, prepoznavanje televizijskih vsebin, preprečevanje kršenja avtorskih pravic, prepoznavanje glasbe brez spletne povezave ter štetje števila gledalcev oziroma poslušalcev televizijskih oziroma radijskih oddaj. BMAT omogoča nadzorovanje radijskih in televizijskih prenosov v realnem času, inteligentno avtomatsko kreiranje seznamov predvajanja ter avtomatsko ocenjevanje vokalistov za namene spletnih avdicij.

Cilj te diplomske naloge je razviti aplikacijo, ki omogoča prepoznavanje glasbe v videoposnetkih in nam poleg podatkov o skladbah, ki se v njih nahajajo, pove tudi, kje v posnetku se posamezna skladba začne in kje konča. Če skladbe ni mogoče odkriti, se uporabniku omogoči, da vnese podatke o skladbah, ki jih program ni prepoznal. Na ta način je te skladbe v prihodnosti mogoče prepoznati.
V delu bom predstavil obstoječe rešitve za prepoznavanje glasbe ter se za eno izmed njih odločil, nato pa bom izbrano knjižnico tudi podrobneje opisal. Nato bom opisal postopek implementacije same aplikacije, zaključil pa bom s prikazom rezultatov ter možnimi izboljšavami za prihodnost.

Poglavje 2

Izbira knjižnice za pridobivanje zvočnih prstnih odtisov

Prvi korak pred začetkom razvoja aplikacije je izbira knjižnice, s pomočjo katere bom pridobival zvočne prstne odtise skladb ter jih skupaj z metapodatki shranjeval v lokalno podatkovno bazo.
Ta izbira je bila ključnega pomena, saj je bil od nje odvisen ves nadaljnji razvoj aplikacije. Od te odločitve je bila odvisna tudi izbira tehnologije. Sam sem se najbolj nagibal k izbiri neke knjižnice, ki bi bila napisana v Pythonu, saj imam z njim največ izkušenj in se pri delu z njim dobro znajdem. Spodaj so naštete in opisane rešitve, ki so mi bile na razpolago in med katerimi sem izbiral.

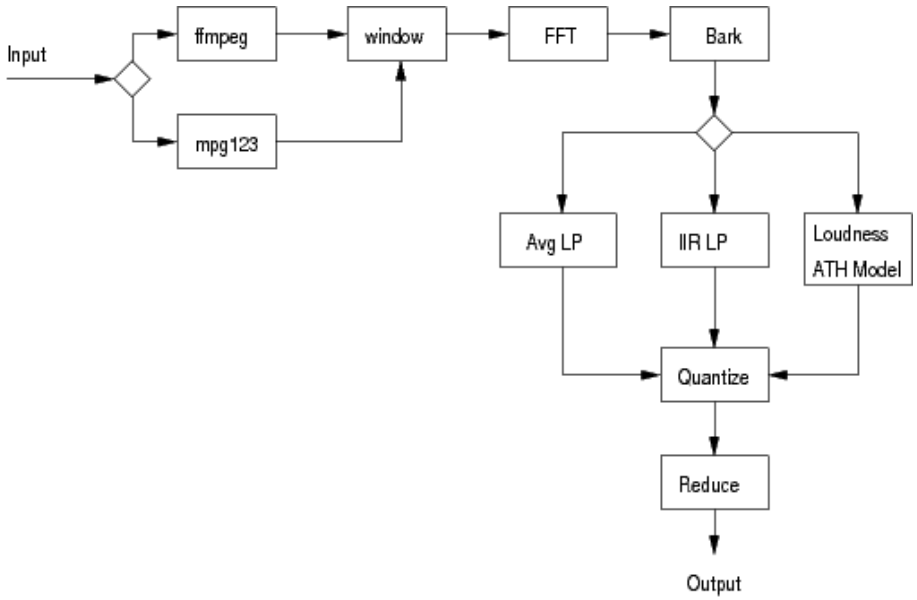
2.1 Echoprint

Echoprint[17] je odprtokodna rešitev za pridobivanje prstnih odtisov ter prepoznavanje glasbe, ki so jo razvili v podjetju The Echo Nest, ki pa se je pred kratkim oziroma v času izdelave mojega diplomskega dela pridružilo podjetju Spotify.
Echoprint zvočne prstne odtise pridobiva tako, da signal najprej pretvori v 11kHz mono signal, nato se na njem izračuna filter, s katerim se poudari vi-

soke ter zaduši nizke frekvence, nato pa se izvede dekompozicija signala na 8 frekvenčnih podpasov, kjer se poiščejo vsi lokalni maksimumi amplitude. Ti se nato podajo zgoščevalni funkciji skupaj s časom pojavitve. Vsaka izmed zgoščenih vrednosti zasede 20 bitov, te pa se indeksirajo v podatkovno bazo skupaj z metapodatki o skladbi. Pri iskanju se nato prešteje število ujemanj zgoščenih vrednosti originalnega posnetka in posnetka, ki ga analiziramo.
Knjižnica ni delovala, saj je ne razvijajo več. Rešitev potrebuje najmanj 20 sekund avdio signala za prepoznavanje pesmi[17], kar je za namene moje aplikacije veliko preveč. Prav tako bi bila vzpostavitev lokalne baze, po kateri bi ujemajoče zvočne prstne odtise lahko iskal, preveč zapletena, saj je echoprint v prvi vrsti namenjen uporabi v obliki spletne storitve. Iz teh razlogov ta rešitev za potrebe diplomske naloge ni bila ustrezna.

2.2 OpenFP

OpenFP[10] je knjižnica, napisana v jeziku C, ki nam omogoča izdelavo datotek, ki vsebujejo zvočne prstne odtise, ki jih je nato potrebno shraniti v nek direktorij. Za iskanje je potrebno pognati OpenFP server ter s funkcijo openfpmatch() poiskati ujemanje z zvočnim prstnim odtisom, ki smo ga pridobili.
Na sliki 2.1 je prikazan postopek pridobivanja zvočnih prstnih odtisov z OpenFP in iz nje je razvidno, da se iz vhodnega zvočnega signala s pomočjo hitre Fourierove transformacije pridobi spektrogram, na katerem se nato opravi dekompozicija signala na frekvenčne podpasove, na njih pa se nato s pomočjo filtrov poudarijo ustrezni deli zvoka, ti pa se nato v obliki 32 bitnih vrednosti shranijo v datoteke, ki predstavljajo zbirke zvočnih prstnih odtisov za posamezne skladbe. Omeniti je potrebno tudi to, da se tri od štirih vrednosti zavržejo z namenom prihranka prostora.
OpenFP bi bil za potrebe diplomske naloge ustrezen, saj je za uporabo zelo preprost in omogoča lokalno shranjevanje zvočnih prstnih odtisov.



Slika 2.1: Shema pridobivanja zvočnih prstnih odtisov z OpenFP[10]

2.3 dejavu

Dejavu[18] je Pythonova knjižnica, ki nam prav tako omogoča pridobivanje zvočnih prstnih odtisov, shranjevanje odtisov v lokalno podatkovno bazo ter prepoznavanje glasbe.

Zvočne prstne odtise pridobiva tako, da s pomočjo hitre Fourierove transformacije pridobi spektrogram, ki ga nato obravnava kot sliko in s pomočjo filtrov na njej izlušči lokalne maksimume amplitude, katere nato kot par (frekvenca, čas) poda zgoščevalni funkciji, od zgoščene vrednosti pa nato ohrani le zadnjih 80 bitov, da s tem skrajša dolžino zgoščene vrednosti s 160 bitov na 80 bitov. S tem se seveda zgubi nekaj informacije, a ne toliko, da iskanje s pomočjo teh zgoščenih vrednosti ne bi bilo uspešno.

Je v prvi vrsti namenjena za uporabo z lokalno podatkovno bazo in nima omejitve za dolžino trajanja posnetka, ki ga prepoznavamo. Je preprosta za vključitev v projekte, napisane v Pythonu in zaradi tega je bila od vseh najbolj primerna, saj z vzpostavljanjem njenega delovanja nisem imel nobenih težav. Po testiranjih ostalih rešitev sem se odločil za dejavu, zaradi

preprostosti uporabe in odlične dokumentacije ter dobrega opisa delovanja.

2.4 ACRCloud

ACRCloud[12] je plačljiva spletna storitev, ki uporabnikom poleg pridobivanja zvočnih prstnih odtisov ter prepoznavanja glasbe nudi še prepoznavanje TV-kanalov ter različne vrste nadzorovanja bodisi televizijskih bodisi radijskih prenosov. Ker je plačljiva, se zanjo nisem odločil, sem jo pa uporabil kot rezervo, v primeru, ko dejavu skladb ne prepozna. Uporabnik mora zanjo plačati naročnino.
Podatkov o postopku pridobivanja zvočnih prstnih odtisov nisem zasledil.
Namestitev je bila preprosta, storitev sem testiral v zastonjskem testnem obdobju in deluje zelo dobro. Skladbe prepoznava s približno dvakrat višjo hitrostjo kot to počne dejavu. Njen namen v moji aplikaciji je prepoznavanje tujih pesmi, ki se nahajajo v videoposnetkih, saj je za lokalno bazo predvideno predvsem prepoznavanje in shranjevanje zvočnih prstnih odtisov slovenskih pesmi.

2.5 AcoustID

AcoustID[4] je brezplačna spletna storitev za prepoznavanje glasbe, v kolikor je aplikacija, v kateri jo uporabljamo, nekomercialna. Pred uporabo je treba aplikacijo oziroma projekt, v katerega jo imamo namen vključiti, samo registrirati.
Podatkov o postopku pridobivanja zvočnih prstnih odtisov nisem zasledil.
Podprta je uporaba v jeziku C, možna pa je tudi uporaba preko Pythona. Skladbe se išče po bazi, sestavljeni iz zvočnih prstnih odtisov, ki so jih prispevali uporabniki. V bazo lahko tudi mi shranjujemo zvočne prstne odtise skupaj z metapodatki o skladbah, ki se shranijo v spletno enciklopedijo glasbe MusicBrainz. Problem pri tej rešitvi je v tem, da je sposobna prepoznavati le na podlagi celotnih datotek in ne kratkih odsekov, kot je to zaželjeno v

mojem primeru.

Poglavje 3

Pregled uporabljenih tehnologij

Spodaj so na kratko opisane vse tehnologije in knjižnice, ki sem jih pri razvoju aplikacije uporabljal. Python je bil glavni programski jezik, v katerem sem pisal aplikacijo. Knjižnici dejavu in ACRCloud sem uporabil za prepoznavanje glasbe, pydub sem uporabil za manipulacijo z avdio posnetki, pyAudioAnalysis pa za izluščevanje odsekov, kjer se v posnetku nahaja glasba. Za oblikovanje uporabniškega vmesnika sem uporabil Pythonovo knjižnico tkinter. Pretvorbo med različnimi formati avdia ter pridobivanje avdia iz videa sem izvajal s pomočjo pretvornika ffmpeg, podatkovno bazo skladb in njihovih zvočnih prstnih odtisov pa s pomočjo MySQL. Kot razvojno okolje sem uporabljal odprtokodni urejevalnik kode Atom, za poganjanje kode pa ukazno vrstico Terminal. Vse skupaj je teklo na operacijskem sistemu OS X Yosemite.

3.1 Python

Python je razširjen visokonivojski splošno namenski dinamičen programski jezik. Poudarjena je berljivost kode, njegova sintaksa pa programerjem omogoča, da za pisanje kode porabijo manj vrstic, kot je to potrebno pri ostalih jezikih, npr. C++ in Java.

3.2 dejavu

Dejavu je Pythonova knjižnica, ki jo je leta 2013 napisal Will Drevo in nam omogoča izračunavanje zvočnih prstnih odtisov za poljubne zvočne posnetke ter shranjevanje le teh v podatkovno bazo MySQL. Knjižnjica nam omogoča tudi prepoznavanje glasbe iz zvočnih datotek s pomočjo primerjanja zvočnih prstnih odtisov neznane datoteke s tistimi, ki so shranjeni v podatkovni bazi.

3.3 ACRCloud

ACRCloud je plačljiva spletna storitev, ki nam omogoča prepoznavanje glasbe, nadzorovanje radijskih oziroma televizijskih prenosov, zaznavanje TV-kanalov ter zaznavanje predvajanih TV-vsebin.

3.4 pydub

Pydub je Pythonova knjižnica, ki jo je leta 2011 razvil James Robert, nam olajša delo z zvočnimi datotekami, saj nam omogoča, da ffmpeg uporabljamo na bolj intuitiven način in da njegove funkcije uporabljamo v Pythonovi kodi in ne v obliki bash ukazov. Z njeno pomočjo lahko zvočne datoteke naložimo v pomnilnik in z njimi manipuliramo na različne načine, lahko jih na primer razrežemo na več kosov, jih obrnemo, stišamo ter izvozimo s poljubnimi lastnostmi in v poljubnem formatu, ki ga podpira ffmpeg.

3.5 pyAudioAnalysis

PyAudioAnalysis je Pythonova knjižnica, ki jo je leta 2014 razvil Theodoros Giannakopoulos in omogoča različne načine analize avdio datotek, kot so: klasifikacija, segmentacija in vizualizacija. Z njo sem na avdio datotekah izvajal segmentacijo govor/glasba.

3.6 tkinter

Tkinter je najpogosteje uporabljana Pythonova knjižnica za implementacijo grafičnih uporabniških vmesnikov. Omogoča intuitivno in hitro implementacijo preprostih grafičnih vmesnikov.

3.7 ffmpeg

Ffmpeg je pretvornik avdia in videa, ki nam omogoča, da pretvarjamo med poljubnimi formati datotek. Z njegovo pomočjo lahko iz videa dobimo zvok v kakršnem koli formatu, ki ga ffmpeg podpira. Z njim lahko tudi spreminjamo mnoge parametre avdio in video datotek, kot so število okvirjev na sekundo, frekvenco vzorčenja in podobno.

3.8 MySQL

MySQL je eden izmed najbolj popularnih odprtokodnih sistemov za kreiranje in upravljanje z relacijskimi podatkovnimi bazami. Je popularna izbira pri razvoju spletnih aplikacij. Uporabljajo ga tudi večja podjetja, kot so: Facebook, Google, Twitter, Youtube in mnogi drugi.

3.9 Atom

Atom je moderen in dostopen odprtokodni urejevalnik kode. Omogoča preprosto instalacijo vtičnikov znotraj same aplikacije in s tem razširi njegovo funkcionalnost ali pa razvijalcu olajša delo. Podpira razvoj v večini danes uporabljanih programskih jezikov.

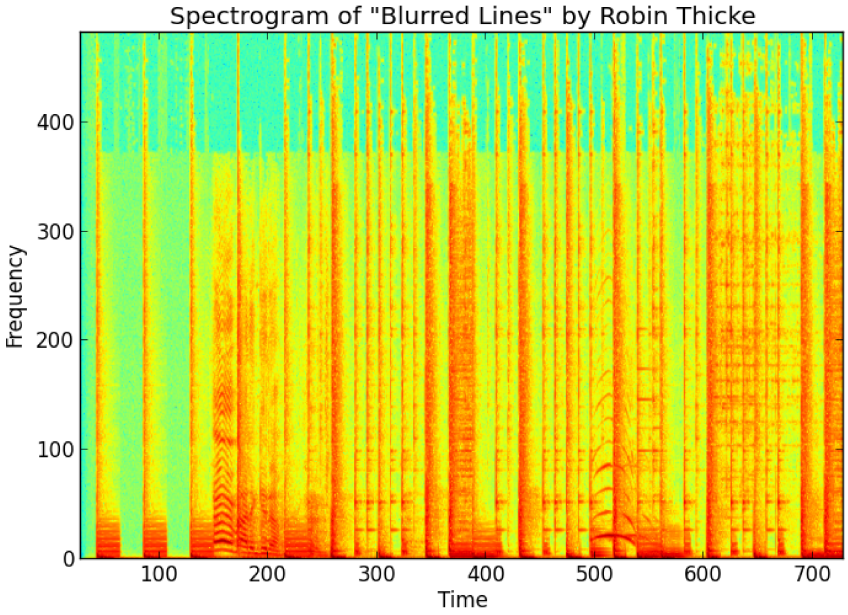
Poglavje 4

Potek pridobivanja in shranjevanja zvočnih prstnih odtisov

V tem poglavju bom podrobneje opisal vse korake, ki jih dejavu opravi na poti do zvočnih prstnih odtisov, primernih za uporabo, in opisal shranjevanje ter iskanje ujemajočih zvočnih prstnih odtisov.

4.1 Spektrogrami

Dejavu v prvem koraku pridobivanja zvočnih prstnih odtisov za vsak zvočni posnetek pridobi spektrogram. Primer spektrograma se nahaja na sliki 4.1. Spektrogram predstavlja dvodimenzionalno polje z amplitudo kot funkcijo časa in frekvence. S pomočjo hitre Fourierjeve transformacije (FFT) se signal pretvori iz časovnega v frekvenčni prostor. Pridobi se amplitudo signala na vsaki izmed frekvenc skozi celoten čas trajanja posnetka. Čas in frekvenca sta izražena diskretno, medtem ko ima amplituda realno vrednost. Na spektrogramu na sliki 4.1 je ta vrednost izražena z barvo, in sicer zelena pomeni najnižjo, rdeča pa najvišjo amplitudo.
Z namenom unikatne reprezentacije vsake posamezne skladbe je potrebno na



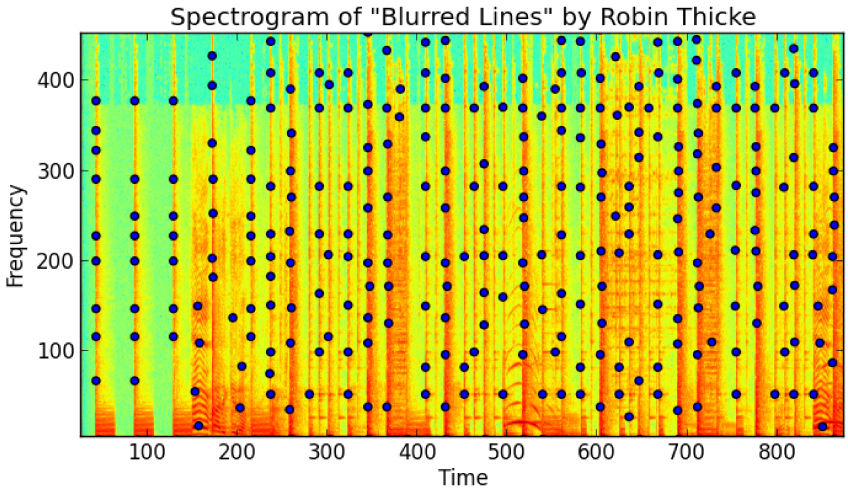
Slika 4.1: Spektrogram [18]

nek robusten način pridobiti edinstvene zvočne prstne odtise.

4.2 Iskanje lokalnih maksimumov amplitude

Lokalni maksimum amplitude je par (čas, frekvenca) z največjo amplitudo v njegovi okolici (slika 4.2). Te poskušamo poiskati zato, ker so ti tisti, ki bodo najverjetneje preživeli šum; iz tega razloga pa je tudi najbolj verjetno, da bomo v posnetku, ki ga analiziramo, naleteli prav na njih, v kolikor gre za ujemajočo se skladbo.

Dejavu te maksimume poišče s pomočjo knjižnice scipy.ndimage[9] ter njenih metod maximum_filter()[14] ter binary_erosion()[13] Teh lokalnih maksimumov se na posamezno pesem najde po več deset tisoč. Ker je teh maksimumov veliko število in ker si zapomnimo zgolj čas in frekvenco, amplitudo pa zavržemo, s tem količino informacije, ki jo imamo, precej zmanjšamo, iz tega razloga pa bo pri nekaterih lokalnih maksimumih amplitude med različnimi



Slika 4.2: Spektrogram z označenimi lokalnimi maksimumi amplitude [18]

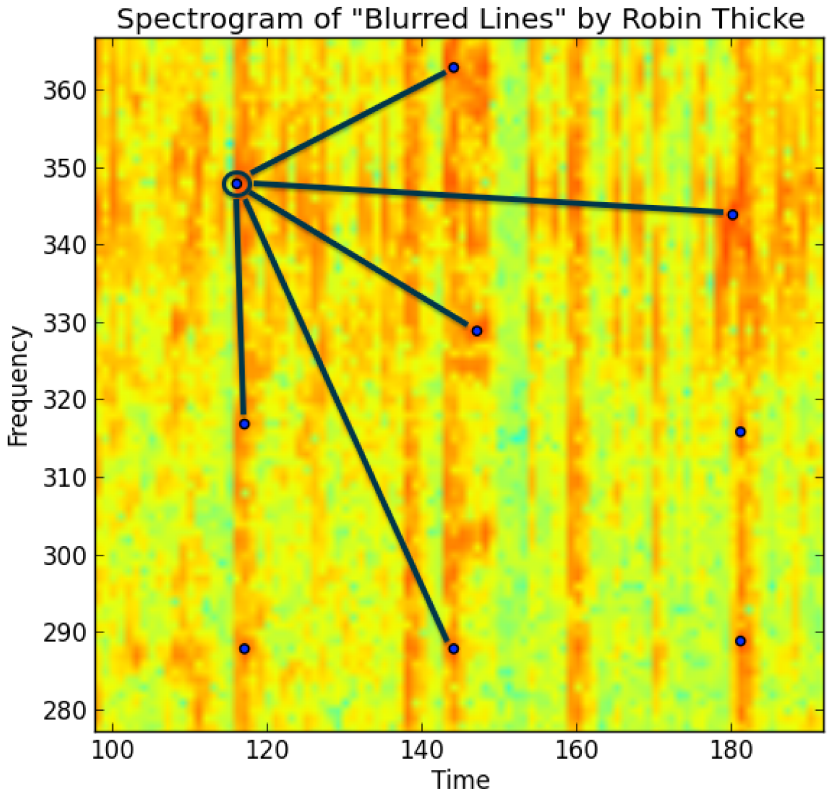
skladbami zelo verjetno prišlo do prekrivanja.

4.3 Pridobivanje zvočnih prstnih odtisov

Rešitev za prekrivanje lokalnih maksimumov je združevanje teh v skupine oziroma kot to počne dejavu v pare (slika 4.3). Iz teh parov lokalnih maksimumov in časovnih razlik med njimi se nato dobi zvočne prstne odtise.

To se naredi z uporabo zgoščevalne funkcije. Zgoščevalne funkcije[7] za vhod vzamejo neke podatke različnih velikosti in jih zgostijo v podatke fiksnih velikosti. Dobra stran teh funkcij je to, da za medsebojno enake vhodne podatke pridelajo enako izhodno zgoščeno vrednost ter da zelo redko pride do tega, da bi za dva različna vhoda dobil enako izhodno zgoščeno vrednost. Zaradi združevanja večih lokalnih maksimumov zvočni prstni odtisi nosijo več informacije.

V primeru dejavu je uporabljena zgoščevalna funkcija SHA-1, kateri so podani podatki o frekvencah para lokalnih maksimumov amplitude ter o časovni razliki med njima. Te podatke se pred zgoščevanjem zapiše v obliki niza, nato pa se jih zgosti s klicem funkcije sha1() iz knjižnice hashlib[8].



Slika 4.3: Združevanje lokalnih maksimumov amplitude [18]

4.4 Shranjevanje zvočnih prstnih odtisov

MySQL tabela zvočnih prstnih odtisov, ki jo uporablja dejavu, vsebuje polja, prikazana na sliki 4.4.

Poleg zgoščene vrednosti ter imena pesmi ima tudi vrednost offset, ki pred-

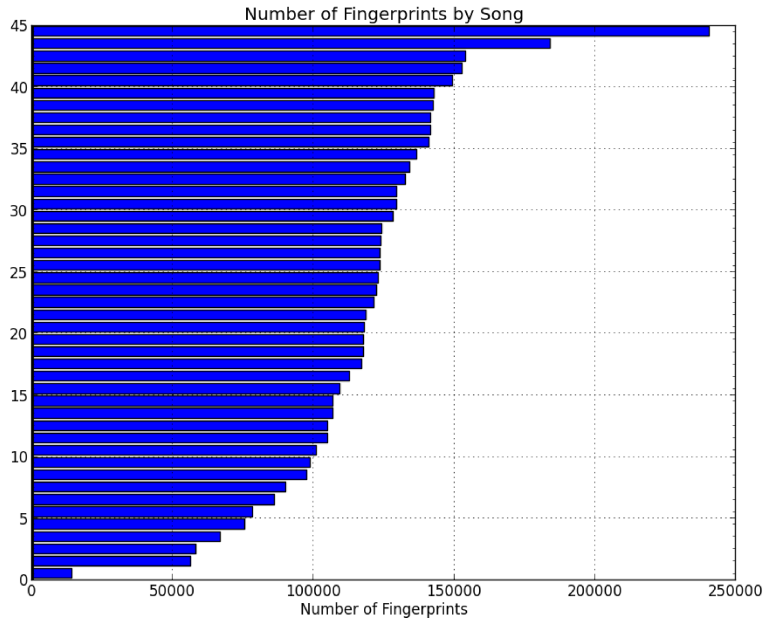
```
CREATE TABLE fingerprints (
    hash binary(10) not null,
    song_id mediumint unsigned not null,
    offset int unsigned not null,
    INDEX(hash),
    UNIQUE(song_id, offset, hash)
);
```

Slika 4.4: Kreiranje tabele zvočnih prstnih odtisov [18]

stavlja časovno koordinato iz tistega dela spektrograma, od koder zgoščena vrednost prihaja oziroma koliko sekund od začetka avdia se nahaja ta zvočni prstni odtis. INDEX za zgoščeno vrednost se uporablja za pohitritev poizvedb. UNIQUE pa poskrbi za to, da nimamo duplikatov posameznih vrednosti v podatkovni bazi. Zgoščeni vrednosti se dolžina skrajša zato, da se prihrani na prostoru, ki ga te vrednosti zasedejo, saj se bo v tabelo shranilo ogromno število zvočnih prstnih odtisov. Na sliki 4.5 je prikazan graf števil zvočnih prstnih odtisov pri neki množici skladb. Iz grafa je razvidno, da si je za vsako pesem potrebno shraniti povprečno več kot sto tisoč zvočnih prstnih odtisov.

Dejavu najprej za vsak zvočni prstni odtis izračuna SHA-1 zgoščeno vrednost, ki je velika 160 bitov. Od te vrednosti se z namenom prihranka na porabljenem prostoru obdrži samo zadnjih 80 bitov, prvih 80 pa se jih zavrže. Binarni zapis zgoščene vrednosti, ki se na koncu shrani v podatkovno bazo, tako zasede 10 bajtov. S tem se prihrani 50 odstotkov prostora, ki bi ga v nasprotnem primeru porabili za shranjevanje zvočnih prstnih odtisov. Dobljena zgoščena vrednost predstavlja zvočni prstni odtis.

S tem, ko se velikost zgoščene vrednosti tako okrni, se seveda izgubi tudi znatna količina informacije, kar pomeni, da se bodo statistično gledano zvočni prstni odtisi med sabo večkrat prekrivali, a še vedno ne tolikokrat, da jih ne



Slika 4.5: Prikaz števila zvočnih prstnih odtisov na množici nekih skladb [18]

bi mogli ločiti.
Poleg tabele za zvočne prstne odtise je potrebna še tabela skladb (slika 4.6).

```
CREATE TABLE songs (
    song_id mediumint unsigned not null auto_increment,
    song_name varchar(250) not null,
    fingerprinted tinyint default 0,
    PRIMARY KEY (song_id),
    UNIQUE KEY song_id (song_id)
);
```

Slika 4.6: Kreiranje tabele skladb [18]

Ta tabela je zelo preprosta, vsebuje namreč le ime skladbe, njen id ter zastavico fingerprinted, ki programu pove, ali je za skladbo potrebno pridobiti zvočne prstne odtise ali ne.

4.5 Iskanje ujemanj zvočnih prstnih odtisov

Ko so enkrat zvočni prstni odtisi skladb shranjeni v podatkovni bazi skupaj s podatki o skladbah, lahko pričnemo z ujemanjem zvočnih prstnih odtisov nekega avdio posnetka s tistimi, shranjenimi v podatkovni bazi.
Ko se za skladbe pred shranjevanjem v bazo pridobijo vsi zvočni prstni odtisi, imajo vrednost offset absolutno, odtisi, pridobljeni s posnetka, ki ga prepoznavamo, pa imajo odmik odvisen od začetka posnetka. Zato se ti odmiki med sabo ne bodo nikoli ujemali, razen v primeru, ko bi se posnetek, katerega prepoznavamo, začel na istem mestu, kot se začne skladba, ki se nahaja v njem. Vemo pa, da so vsi zvočni prstni odtisi posnetka med sabo enako oddaljeni, kot so to v originalnem posnetku, pod predpostavko, da se skladba predvaja z enako hitrostjo kot studijska verzija, katere zvočne prstne odtise imamo shranjene v podatkovni bazi. Zato za vsakega izmed ujemajočih zvočnih prstnih odtisov izračunamo razliko med odmikom tega odtisa v originalni skladbi ter relativnim odmikom posnetka, katerega skladbe želimo prepoznati. Vsa prava ujemanja bodo v tem primeru imela enako razliko. Po tem je potrebno še vsa ta ujemanja prešteti in skladbo napovedati tako, da izberemo tisto, ki ima največ ujemanj.

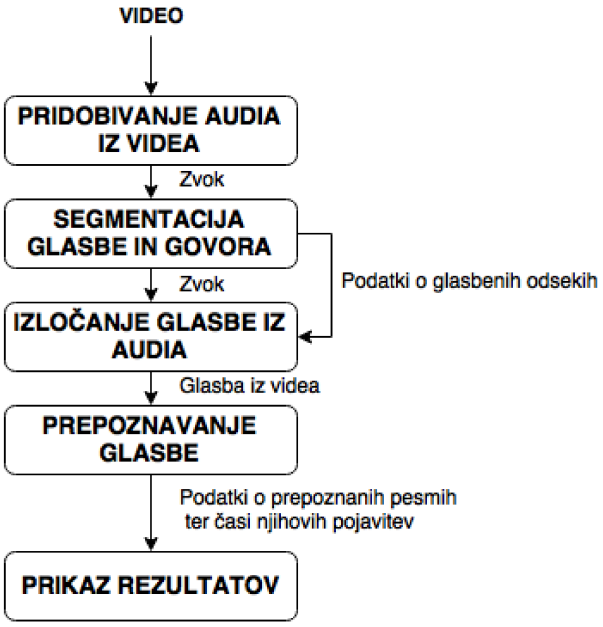
Poglavje 5

Implementacija

Diplomska naloga od nas zahteva, da razvijemo aplikacijo, ki prepozna glasbo, ki se nahaja v videoposnetku ter poleg tega prikaže še podatke o tem, kje se posamezna skladba začne in kje konča. Poleg tega je potrebno izbrati neko zunanjo spletno storitev, ki uporabniku služi kot rezervna rešitev za prepoznavanje glasbe. V primeru, ko tudi ta rešitev odpove, je potrebno uporabniku omogočiti, da vnese podatke o neprepoznanih pesmih zato, da se ti lahko shranijo v podatkovno bazo za prihodnjo uporabo.

Za dosego tega cilja je potrebno najprej pridobiti zvok iz videoposnetka, ki ga dobimo na vhod programa. Ko enkrat imamo ta zvok, je potrebno oziroma zaželjeno na njem opraviti segmentacijo govora in glasbe, saj s tem izvemo, kje v posnetku se nahaja glasba in lahko nato iskanje glasbe poženemo zgolj na odsekih, kjer je glasbo moč najti. S tem prihranimo na času, ne da bi vplivali na natančnost iskanja. Na nek način je potrebno manipulirati z zvokom, in sicer ga razrezati na kratke odseke in nato na vsakem od teh odsekov pognati prepoznavanje glasbe. S tem dobimo približno natančne ocene časov, kjer se posamezne pesmi pojavijo in kje zaključijo. Za prepoznavanje glasbe je treba uporabiti dve rešitvi za preventivo, v primeru, ko ena izmed njih glasbe ne prepozna. Če nobena od rešitev glasbe ne prepozna, se uporabniku omogoči vnos podatkov o pesmih, ki se nato shranijo v podatkovno bazo za prihodnjo uporabo. Vse skupaj mora povezovati nek uporabniški

vmesnik, ki uporabniku omogoča izbiro videoposnetka ter prikaz rezultatov.



Slika 5.1: Shematičen prikaz delovanja rešitve

5.1 Pridobivanje avdia iz videa

Da zvok v videu lahko analiziramo, samega videa ne potrebujemo, zato avdio od njega ločimo in na njem izvajamo nadaljnje delo.
Za pridobivanje avdia iz videa sem uporabil ffmpeg ukaz:

```
ffmpeg -i potDoVidea.mp4 -vn -acodec pcm_s16le -ar 44100 -ac 2 output.wav
```

Stikalo -i pomeni vhodno datoteko.
Stikalo -vn ffmpeg-u pove, da naj v izhodni datoteki ne bo videa.
Stikalo -acodec nam omogoča, da izberemo kodek izhodne avdio datoteke.
Stikalo -ar nam omogoča, da nastavimo frekvenco vzorčenja izhodne avdio datoteke.
Stikalo -ac nam omogoča, da izberemo število kanalov zvoka v izhodni avdio datoteki.

S pomočjo tega ukaza lahko aplikacija pridobi wav format avdia iz kakršnega koli formata video posnetka. Za izhodni format sem izbral format wav zgolj zaradi naslednjega koraka implementacije, in sicer segmentacije govor/glasba, saj knjižnjica pyAudioAnalysis kot vhodni format avdia zahteva format wav.

5.2 Segmentacija glasbe in ostalih delov avdia

Z namenom, da glasbe ne bi iskali tam, kjer je ni moč najti in da bi skrajšali čas izvajanja programa, je potrebno iz originalnega avdio posnetka izrezati dele, kjer se nahaja glasba. To sem naredil s pomočjo knjižnice pyAudioAnalysis. Avdio se razdeli na segmente, dolge po eno sekundo, nato pa se na vsakega izmed njih aplicira prednaučeni nadzorovani klasifikator, da se klasificira pripadnost segmenta enemu izmed dveh vnaprej določenih razredov; v tem primeru sta to govor ter glasba. To se doseže z uporabo funkcije mtFileClassification(), ki vrne seznam ničel in enic, s pomočjo katerih lahko nato avdio posnetek razdelimo na dva dela: tistega, ki vsebuje glasbo in tistega, ki je ne.

5.3 Izločanje glasbe iz avdia

S pomočjo knjižnice pydub glede na podatke, pridobljene iz prejšnjega koraka implementacije, iz originalnega avdia izrežem odseke, kjer se v avdiu nahaja glasba. Te odseke izvozim v mp3 formatu in si zraven shranim še podatke o tem, kje se posamezen odsek začne in kako dolgo traja.

5.4 Prepoznavanje glasbe

Ko iz originalnega avdia izrežem vse odseke, kjer se nahaja glasba, je potrebno na vsakem izmed teh pognati prepoznavanje glasbe. Najprej si posamezen odsek shranim v pomnilnik s pomočjo knjižnice pydub. Nato ta

odsek razdelim na fragmente dolžine 5 sekund in nato na vsakem izmed teh kličem funkcijo recognize(). Ta funkcija za vsakega izmed klicev vrne naslov pesmi, ki jo je prepoznala, ter vrednost confidence, ki mi sporoča, kako dobro je ujemanje. Sam sem na podlagi izkušenj ob testiranju odkril, da v večini primerov, ko je vrednost confidence nižja od 12, naslov pesmi, ki ga funkcija recognize() vrne, ni pravi oziroma bi lahko rekli, da je naključen. Zato takrat v polje rezultatov shranim niz "unknown". Ko pa je vrednost confidence večja ali enaka 12, predpostavim, da je vrnjen rezultat pravi, zato takrat v polje rezultatov tudi shranim vrednost, ki jo dobim s klicem funkcije recognize(). Vrednost confidence predstavlja število ujemanj v odseku, na katerem kličemo funkcijo recognize(). Po koncu izvajanja te funkcije se vrne polje rezultatov, kjer je za vsak pregledani odsek dolžine 5 sekund zabeležen ali naslov pesmi ali pa niz "unknown" v primeru, ko je confidence bil nižji od 12.
Rezultate nato obdelam tako, da si zabeležim, od kdaj do kdaj traja posamezen odsek, nato pa jih shranim v dve novi polji, in sicer polje prepoznanih in polje neprepoznanih skladb. S tem dobim primerno obliko podatkov za prikaz rezultatov.

5.5 Integracija zunanje spletne storitve

Za primere, ko je naša podatkovna baza zvočnih prstnih odtisov pesmi premajhna, da bi nam lahko bila v veliko korist, je dobro imeti nek plan B. Za ta primer je bilo potrebno implementirati možnost uporabe neke zunanje spletne storitve z namenom odkritja čimvečih skladb. Tu sem se po priporočilu mentorja odločil za storitev ACRCloud. ACRCloud je plačljiva storitev, ki svoje storitve prepoznavanja opravlja za veliko število podjetij. Za vključitev njihove storitve in za testiranje sem se naročil na brezplačno preizkusno uporabo storitve. Ob tem sem na njihovi strani registriral svoj projekt in pridobil privatni ključ in ostale podatke, ki so potrebni za omogočanje dostopa do njihove storitve. Te podatke sem nato v kodi shranil v objekt config, ki ga je

nato za uporabo metod razreda ACRCloudRecognizer potrebno podati ob inicializaciji. Nato sem definiral funkcijo, ki kliče funkcijo razreda ACRCloudRecognizer recognizebyfile() ter podatke, ki jih s pomočjo tega klica pridobi, obdelal s pomočjo knjižnice simplejson, ki mi je omogočala, da iz množice podatkov o prepoznani pesmi v obliki stringa pretvorim v Pythonov slovar ter tako pridobim podatek o naslovu pesmi.

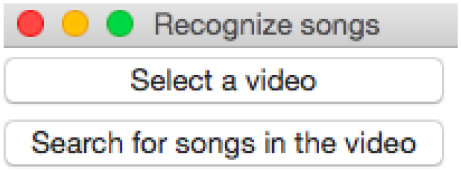
5.6 Uporabniški vnos podatkov o neprepoznanih pesmih

V primeru, ko program ne prepozna vseh pesmi, se uporabniku na uporabniškem vmesniku izpišejo časi pojavitev teh neprepoznanih odsekov, poleg njih pa sem z namenom vnosa podatkov o teh odsekih dodal dve vnosni polji. Ti vnosni polji omogočata, da uporabnik vanju vpiše podatke o izvajalcu ter naslovu pesmi, v kolikor so ti podatki njemu znani. Ko uporabnik vnese podatke, ima možnost, da te shrani v lokalno podatkovno bazo zvočnih prstnih odtisov. S klikom na gumb "Fingerprint", ki se nahaja ob vnosnih poljih, v katere je uporabnik vnesel podatke, se shranijo pripadajoči podatki o izvajalcu in naslovu, hkrati pa se shranijo podatki o začetku in koncu tega odseka. S temi podatki in s pomočjo knjižnice pydub nato iz originalnega avdio posnetka izrežem odsek, za katerega je uporabnik vnesel podatke ter ga izvozim v obliki mp3 datoteke z metapodatki iz pripadajočih vnosnih polj. Na tej mp3 datoteki se nato kliče funkcija fingerprint() knjižnice dejavu, s katero se za to datoteko izračunajo zvočni prstni odtisi, ki se nato skupaj z metapodatki shranijo v lokalno podatkovno bazo, kjer hranimo zvočne prstne odtise skladb, ki jih lahko prepoznamo.

5.7 Uporabniški vmesnik

Ko se aplikacija požene, se najprej prikažeta dva gumba. Prvi za izbiro videoposnetka, drugi pa za začetek prepoznavanja glasbe. Ko uporabnik klikne

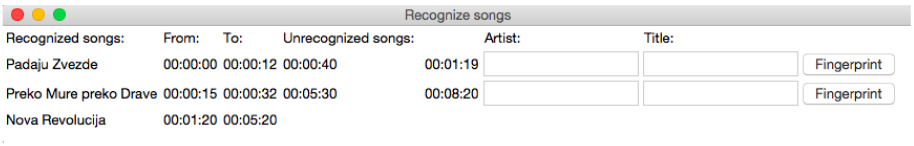
gumb za izbiro videoposnetka, se mu prikaže okno za izbiro datoteke. Ko



Slika 5.2: Uporabniški vmesnik - izbira datoteke

uporabnik izbere neko datoteko, se pot do te datoteke shrani v spremenljivko video. S klikom na gumb za pričetek prepoznavanja glasbe se kliče funkcija analyzeVideo(). Ta funkcija najprej kliče ffmpegov ukaz za pridobivanje avdia iz video datotek. Ko ima avdio datoteko v formatu wav, se na njej kliče segmentacija, ki se shrani v tabelo. Nato se wav datoteka, ki smo jo pridobili iz videoposnetka, pretvori v mp3 format, da se pri pridobivanju zvočnih prstnih odtisov porabi manj pomnilnika. Ta mp3 datoteka in podatki o segmentaciji se nato podajo funkciji extractMusicParts(), ki odseke, kjer se v posnetku pojavi glasba, izreže iz originalnega posnetka in jih začasno izvozi v mapo exports. Nato se na vsakem izmed teh izvoženih odsekov kliče funkcija search, ki po izvedenem iskanju vrne tabelo rezultatov, ki jo nato s funkcijo getDurations() analiziram in pridobim trajanje posameznih odkritih ali neodkritih skladb. Ko je iskanje bilo izvedeno na vseh odsekih, se mapa exports izbriše, skupaj z vsemi izvoženimi odseki. Zatem se rezultate predela v primerno obliko za zapis na uporabniški vmesnik in se za vsakega izmed rezultatov na vmesniku izpišejo podatki v obliki oznak. Na levi strani vmesnika se izpišejo naslovi prepoznanih pesmi ter njihov čas pojavitve v posnetku ter čas, ko se ta skladba v posnetku konča. Na desni strani pa se poleg podatkov o začetku in koncu neprepoznanih skladb prikažeta tudi dve vnosni polji, ki uporabniku omogočata vnos imena izvajalca ter naslova pesmi, v kolikor jo uporabnik pozna ter hoče te podatke shraniti v podatkovno bazo z namenom, da bo te pesmi v prihodnje mogoče prepoznati. Poleg vnosnih polj se nahajajo gumbi za vsako neprepoznano pesem, ki ob kliku kličejo funkcijo fingerprint(). Ta si shrani ime izvajalca

ter naslov pesmi iz soležnih vnosnih polj ter glede na čas začetka in konca neprepoznanega odseka iz originalnega posnetka izreže ta del zvoka in ga izvozi skupaj z metapodatki. Za to izvoženo datoteko se nato pridobijo zvočni prstni odtisi, ki se s pomočjo dejavu-jeve funkcije fingerprintbyfile() shranijo v podatkovno bazo skupaj s pripadajočimi metapodatki. To skladbo je nato mogoče v prihodnje prepoznati.



Slika 5.3: Uporabniški vmesnik - prikaz rezultatov

5.8 Testiranje

5.8.1 Hitrost

Primerjal sem hitrosti izvajanja programa pri uporabi knjižnice dejavu ter pri uporabi storitve ACRCloud. Hitrosti sem testiral na dveh različnih datotekah različnih velikosti. Meritve sem izvajal na prenosnem računalniku Macbook Air, z naloženim operacijskim sistemom OS X Yosemite. Ima Intelov procesor Core i5 s frekvenco 1,8 GHz, 4 GB DDR3 rama s frekvenco 1600 MHz, integriran grafični procesor Intel HD Graphics 4000 s 1024 MB rama ter SSD disk velikosti 128 GB. Rezultate sem dobil tako, da sem za vsako datoteko program pognal trikrat ter povprečil čase trajanja.

Velikost datoteke	`dejavu`	`ACRCloud`
`141,5 MB`	328 sekund	157 sekund
`286,9 MB`	425 sekund	250 sekund

Tabela 5.1: Primerjava hitrosti med dejavu in ACRCloud

5.8.2 Natančnost

Natančnost prepoznavanja skladb z uporabo obeh rešitev, torej dejavu in ACRCloud, je v primeru, ko videoposnetek vsebuje studijske posnetke skladb, brez kakršne koli spremembe v kvaliteti ali spremembe trajanja stoodstotna, saj obe rešitvi vedno vrneta pravilne rezultate, v kolikor njuni podatkovni bazi vsebujeta zvočne prstne odtise teh skladb.
Ker je bilo ob začetku dela predvideno, da se bo aplikacija uporabljala zgolj za prepoznavanje skladb, ki ustrezajo zgornjemu opisu, testiranja na posnetkih z občutno slabšo kvaliteto ali šumom nisem opravljal.

Poglavje 6

Sklepne ugotovitve

6.1 Možne izboljšave

Aplikacija, ki sem jo razvil, zadošča vsem zahtevam, ki so bile specificirane na začetku, ko sem nalogo prevzel. Je pa, kot vedno, ostalo še nekaj prostora za izboljšave.

6.1.1 Hitrost izvajanja

Sedaj v posnetkih preverjam čisto vsak del posnetka in to, sploh s knjižnico dejavu, traja precej dolgo; na primer: od začetka pa do konca izvajanja programa na enajst minut dolgem videoposnetku traja dve minuti. Verjetno bi se trajanje kar občutno skrajšalo, če bi namesto knjižnice dejavu uporabil storitev OpenFP ali pa Echoprint, če bi mi ga uspelo vzpostaviti. Druga možnost za pohitritev algoritma pa je ta, da za potrebe določanja skladbe, ki se predvaja, in njenega trajanja ne bi preverjal čisto vsakega dela avdio posnetka. To sem poskusil že med razvojem, in sicer tako, da sem namesto tega, da posnetek razrežem na pet sekundne odseke in preverim čisto vsakega, iz posnetka vzel po pet sekund dolg odsek vsakih deset ali 15 sekund. S tem se je izvajanje algoritma skrajšalo za približno dve tretjini časa. Problem mojega načina pohitritve pa je v tem, da na tak način ne bi bilo mogoče do-

volj natančno zaznati začetka in konca skladbe, ki jo prepoznavamo. Mogoče bi bila dobra ideja meje med posameznimi skladbami iskati z bisekcijo, saj na ta način prav tako ne bi bilo treba preverjati vsake sekunde posnetka. Bisekcija bi delovala tako, da bi najprej preverila prvih pet sekund posnetka in nato zadnjih pet sekund. Če bi se rezultata prepoznavanja obeh odsekov med seboj ujemala oziroma bi se v obeh odsekih predvajala ista skladba, bi lahko z gotovostjo trdili, da je v celem odseku samo ena skladba in to tista, ki smo jo zaznali na začetku in na koncu. V primeru, ko na koncu rezultat ne bi bil enak, bi enak algoritem pognali na začetku in na odseku, ki se nahaja na polovici posnetka.

6.1.2 Klasifikacija originalnega avdio posnetka

Druga možna izboljšava bi bila izboljšava algoritma za segmentacijo. Sedaj segmentacija dobro deluje na videoposnetkih, ki imajo raznoliko vsebino, kot so na primer posnetki TV-oddaj, saj vsebujejo nekaj govora in nekaj glasbe. Problem nastane, ko bi aplikaciji na primer kot vhod podali videospot za neko skladbo, kjer ni nič govora, ampak samo glasba, saj ta segmentacija, ki sem jo uporabil s pomočjo pyAudioAnalyisis kljub temu, da v posnetku ni govora, na nekaterih delih govor napačno zazna. Rezultat tega je, da aplikacija vrne podatke, ki kažejo, da je, na primer na štirih različnih mestih v posnetku zaznala isto pesem, medtem ko je pravzaprav v videoposnetku zgolj ena. Izboljšani algoritem bi moral na nek drugačen način opravljati klasifikacijo govora in glasbe.

6.1.3 Uporabniški vmesnik

Tretja možna izboljšava pa bi bila na področju prijaznosti do uporabnika in estetike. Pri izdelavi aplikacije se na to nisem toliko osredotočal, saj mi je bil glavni cilj, da aplikacija prikaže podatke, ki jih uporabnik od nje

pričakuje. Za neprepoznane skladb znotraj aplikacije bi na primer lahko omogočil predvajanje teh odsekov, tako da uporabniku ne bi bilo potrebno odpirati originalnega posnetka, se premakniti na čas začetka neprepoznane skladbe in šele takrat pričeti s poslušanjem.

Literatura

[1] Atom. https://atom.io. Dostopano: 2016-08-19.

[2] Atom (text editor). https://en.wikipedia.org/wiki/Atom_(text_editor). Dostopano: 2016-08-19.

[3] Audio recognition media monitoring. https://www.acrcloud.com. Dostopano: 2016-08-19.

[4] Documentation — acoustid. https://acoustid.org/docs. Dostopano: 2016-08-19.

[5] ffmpeg audio format conversions. https://linuxconfig.org/ffmpeg-audio-format-conversions. Dostopano: 2016-08-19.

[6] ffmpeg documentation. https://ffmpeg.org/ffmpeg.html. Dostopano: 2016-08-19.

[7] Hash function. https://en.wikipedia.org/wiki/Hash_function. Dostopano: 2016-08-30.

[8] hashlib. https://docs.python.org/2/library/hashlib.html. Dostopano: 2016-08-30.

[9] Multi-dimensional image processing. http://docs.scipy.org/doc/scipy/reference/ndimage.html. Dostopano: 2016-08-30.

[10] Openfp. http://open-fp.sourceforge.net/index.html. Dostopano: 2016-08-19.

[11] Python (programming language). https://en.wikipedia.org/wiki/Python_(programming_language). Dostopano: 2016-08-19.

[12] Recognize music - acrcloud. https://www.acrcloud.com/docs/tutorials/identify-music-by-sound/. Dostopano: 2016-08-19.

[13] scipy.ndimage.binary_erosion. http://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.binary_erosion.html#scipy.ndimage.binary_erosion. Dostopano: 2016-08-30.

[14] scipy.ndimage.maximum_filter. http://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.maximum_filter.html#scipy.ndimage.maximum_filter. Dostopano: 2016-08-30.

[15] Tkinter - python wiki. https://wiki.python.org/moin/TkInter. Dostopano: 2016-08-19.

[16] Extract audio from video files to wav using ffmpeg. http://savvyadmin.com/extract-audio-from-video-files-to-wav-using-ffmpeg/, 2009. Dostopano: 2016-08-19.

[17] Echoprint - open source music identification. http://echoprint.me, 2012. Dostopano: 2016-08-19.

[18] Will Drevo. Audio fingerprinting with python and numpy. http://willdrevo.com/fingerprinting-and-audio-recognition-with-python/, 2015. Dostopano: 2016-08-19.

[19] Theodoros Giannakopoulos. Mysql. https://en.wikipedia.org/wiki/MySQL. Dostopano: 2016-08-19.

[20] Theodoros Giannakopoulos. pyaudioanalysis: An open-source python library for audio signal analysis. http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0144610#sec002, 2015. Dostopano: 2016-08-19.

[21] Theodoros Giannakopoulos. Fixed-segment segmentation classification. https://github.com/tyiannak/pyAudioAnalysis/wiki/5.-Segmentation, 2016. Dostopano: 2016-08-19.

[22] Antonius Kalker Jaap Haitsma. A Highly Robust Audio Fingerprinting System. *International Symposium on Music Information Retrieval (ISMIR)*, pages 107–115, 2002.

[23] Ton Kalker Jaap Haitsma Pedro Cano, Eloi Batlle. A Review of Algorithms for Audio Fingerprinting. *The Journal of VLSI Signal Processing*, 41(3):271–284, 2005.

[24] James Robert. Manipulate audio with a simple and easy high level interface. http://pydub.com. Dostopano: 2016-08-19.

[25] James Robert. Manipulate audio with a simple and easy high level interface. https://github.com/jiaaro/pydub/. Dostopano: 2016-08-19.

[26] Brian Whitman. The audio fingerprinting at the echo nest faq. http://notes.variogr.am/post/27796385927/the-audio-fingerprinting-at-the-echo-nest-faq, 2015. Dostopano: 2016-08-19.