UNIVERZITET U BEOGRADU

ELEKTROTEHNIČKI FAKULTET



**Algoritmi za prepoznavanje muzičkih numera**

MaSTER RAD

Mentor: Kandidat:

Dr Dragana Šumarac Pavlović, profesor Uroš Stojanović, 3269/2023

Beograd, 2025.

Sadržaj

[1. Uvod 1](#_Toc212547587)

[2. Shazam 2](#_Toc212547588)

[3. Teorijski osnov 3](#_Toc212547589)

[3.1 Zvuk 3](#_Toc212547590)

[3.2 Furijeova transformacija 4](#_Toc212547591)

[3.3 Spektrogram 5](#_Toc212547592)

[3.4 Hash-ovanje i njegova primena u aplikaciji Shazam 6](#_Toc212547593)

[3.5 Autoenkoder 7](#_Toc212547594)

[4. Arhitektura i implementacija sistema 9](#_Toc212547595)

[4.1 Obrada zvuka 9](#_Toc212547596)

[4.2 Kreiranje digitalnog otiska (*Fingerprinting*) 10](#_Toc212547597)

[4.3 Baza podataka 11](#_Toc212547598)

[4.4 Prepoznavanje 12](#_Toc212547599)

[5. Performanse i evaluacija 13](#_Toc212547600)

[5.2 Tačnost sistema pri identifikaciji zašumljenih lokalnih fajlova 15](#_Toc212547601)

[5.3 Tačnost sistema pri korišćenju različitih dužina segmenata 19](#_Toc212547602)

[5.4 Tačnost sistema pri analizi specifičnih delova numera 24](#_Toc212547603)

[5.5 Vreme prepoznavanja 28](#_Toc212547604)

[5.6 Ograničenja sistema 28](#_Toc212547605)

[6. Zaključak 29](#_Toc212547606)

[7. Literatura 30](#_Toc212547607)

[8. Spisak slika 31](#_Toc212547608)

[9. Spisak tabela 32](#_Toc212547609)

[10. Prilog 33](#_Toc212547610)

[BuildDatabase 33](#_Toc212547611)

[Echo 35](#_Toc212547612)

[addNoise 38](#_Toc212547613)

[echoNeuro 38](#_Toc212547614)

# Uvod

Potreba za prepoznavanjem muzičkih numera i analizom njihove međusobne sličnosti postaje sve prisutnija u savremenom digitalnom okruženju. Ova potreba se javlja u različitim kontekstima, od ličnih i razonodnih aktivnosti, kao što je identifikacija pesme koju čujemo u prolazu, do profesionalnih i komercijalnih namena poput otkrivanja plagijata i zaštite autorskih prava.

Tokom poslednjih decenija, razvijen je niz softverskih rešenja koja omogućavaju prepoznavanje muzike na osnovu kratkih audio uzoraka. Najpoznatiji među njima su *Shazam*, *SoundHound* i *Adobe Audition*, koji koriste različite metode digitalnog otiskivanja zvuka i brzo poređenje sa velikim bazama podataka. Osnovna funkcionalnost ovih sistema ogleda se u formiranju jedinstvenog digitalnog otiska pesme i njegovom upoređivanju sa već postojećim otiscima.

Rad koji sledi fokusira se na teorijske osnove i praktičnu implementaciju pojednostavljene verzije *Shazam* sistema. Cilj rada je da kroz jasan primer objasni način na koji se muzički otisak može formirati, obraditi i uporediti, koristeći otvorene softverske biblioteke u programskom jeziku *Python*.

Za razliku od originalnog *Shazam* algoritma, koji koristi tzv. konstelacione mape frekvencijskih vrhova iz spektrograma i njihovo heširanje radi optimizacije memorije i brzine, ovaj rad koristi Mel-frekventne kepstralne koeficijente (MFCC) kao osnovu za kreiranje otiska. MFCC koeficijenti predstavljaju sažetak spektralnih karakteristika signala i široko se koriste u obradi govora i prepoznavanju muzike. U kombinaciji sa dodatnim koracima kao što su redukcija šuma, normalizacija i klizna prozorska funkcija, dobija se efikasna metoda prepoznavanja muzike koja je jednostavna za implementaciju i testiranje.

Poseban segment rada bavi se analizom ograničenja ovakvih sistema. Unošenjem šumova u već postojeće numere formiramo signale koji simuliraju odredjenu količinu ambijentalnog zvuka. Na taj način vrši se poređenje između sposobnosti algoritma u različitim uslovima. Pored ovoga algoritam će biti testiran za različite veličine isečaka pesama, kao i na različitim delovima pesama.

Ovaj rad ne samo da pruža tehnički uvid u to kako funkcionišu sistemi poput *Shazam*-a, već kroz sopstvenu implementaciju i testiranje omogućava dublje razumevanje prednosti i ograničenja ovakvih tehnologija u simuliranim uslovima.

# Shazam

Danas je *Shazam* najpoznatija aplikacija za prepoznavanje muzike u svetu. Razvijena od strane britanske kompanije *Shazam Entertainment Ltd.*, aplikacija je prvi put predstavljena 2002. godine i odmah je privukla veliku količinu pažnje zbog svoje sposobnosti da identifikuje pesme na osnovu kratkih audio uzoraka, čak i usled pozadinske buke, kompresije i lošeg kvaliteta originalnog snimka. Sistem je uspešno funkcionisao sa bazom od više miliona muzičkih numera, a vremenom su dodate i komercijalne funkcionalnosti kao što su kupovina pesama, kao  i automatsko praćenje muzičkog sadržaja u cilju zaštite autorskih prava.

Osnovni princip rada *Shazam* algoritma zasniva se na formiranju digitalnog otiska pesme. Za razliku od klasičnih spektrogramskih analiza koje gledaju celu zvučnu sliku, *Shazam* izdvaja samo najizraženije vrhove u spektrogramu (frekvencijske pikove) i koristi njihovu vremensku i frekvencijsku raspodelu za kreiranje jedinstvenih heš-tokena. Svaki token predstavlja kombinaciju jednog zvučnog vrha i njegovog odnosa prema drugom bliskom vrhu u istom segmentu, uključujući razliku u vremenu i frekvenciji.

Ovi tokeni se potom hešuju kako bi se smanjila količina podataka i omogućilo brzo pretraživanje kroz veliku bazu poznatih pesama. Dobijeni digitalni otisci moraju zadovoljiti nekoliko važnih kriterijuma kako bi sistem bio efikasan:

* **Vremenska invarijantnost** – otisak treba da bude nezavistan od tačne pozicije uzorka unutar pesme, što znači da isti uzorak iz sredine ili kraja numere treba da bude prepoznat isto kao i onaj sa početka.
* **Vremenska lokalizovanost** – svaka heš kombinacija se računa iz parova lokalnih zvučnih događaja u bliskom vremenskom okviru.
* **Robusnost** – sistem mora biti otporan na šumove, kompresiju, eho i druge degradacije u kvalitetu snimka.
* **Entropijska bogatost** – visoka entropija heš-tokena omogućava bolju razliku među pesmama i smanjuje broj lažno pozitivnih identifikacija.

Pomoću ovako definisanog sistema *Shazam* može da izvrši pretragu među milionima otisaka u roku od nekoliko sekundi i identifikuje pesmu sa visokim nivoom tačnosti, čak i ako je korisnik snimio samo nekoliko sekundi melodije u bučnom okruženju.

Zahvaljujući ovom pristupu, *Shazam* je postavio standard u oblasti prepoznavanja muzike, a njegov algoritam je postao osnova za razvoj mnogih drugih sistema koji pokušavaju da ponude sličnu funkcionalnost, bilo u komercijalne, akademske ili eksperimentalne svrhe.

# Teorijski osnov

## 3.1 Zvuk

Zvuk je svaka vremenski promenljiva mehanička deformacija u elastičnoj sredini. Zvuk nam služi kao sredstvo komunikacije i medium za prenos informacija. Osnovni oblici zvučnih signala su govor, muzika i ambijentalni zvuk. U akustici zvuk sagledavamo u dva domena: vremenskom i frekvencijskom. Vremenski domen nam pretežno služi  za snimanje i reprodukciju zvuka. U obradi signala najčešće koristimo frekvencijski domen zbog bolje preglednosti i lakše manipulacije nad signalom. Prelaz iz vremenskog domena u frekvencijski izgleda tako što signal koji je analogni skup vrednosti u vremenu razložimo u sumu sinusoida različitih frekvenci. Karakteristika da svaki signal možemo da razložimo u sumu sinusoida potiče iz Furijeovog reda. Karakteristike zvuka koje nas zanimaju gledano iz frekvencijskog domena su frekvence i njihove amplitude. Ove vrednosti dobijamo primenom Furjeove transformacije na signal.

Mikrofon je ulazni elektroakustički pretvarač koji je prilagođen radu u vazduhu kao mediju. Mikrofon pretvara zvučni pritisak, koji mu je ulazna veličina, u električni signal na njegovom izlazu. Ovi uređaji rade u digitalnom domenu. Pri snimanju bilo kog zvuka, membrana mikrofona osciluje i pretvara mehaničku energiju u analogni električni signal. Analogno-digitalni konvertori vrše konverziju iz analognog signala u digitalni kroz procese semplinga i kvantizacije. Sempling je proces kojim odabiramo da ubeležimo vrednost na svakih T sekundi. Ovim postupkom dobijamo 1/T odbiraka u sekundi, koji ljudskom uhu pri reprodukciji zvuče približno identicno kao originalni signal za 1/T > 44100Hz. Kvantizacija je proces diskretizacije snaga signala. Deljenjem skale jačine signala na konačno mnogo vrednosti koje možemo dodeliti vrednosti amplitude mi ga kvantizujemo. Na slici ispod je prikazan signal pre i nakon konverzije iz analognog u digitalni.

A graph of a signal

Description automatically generated

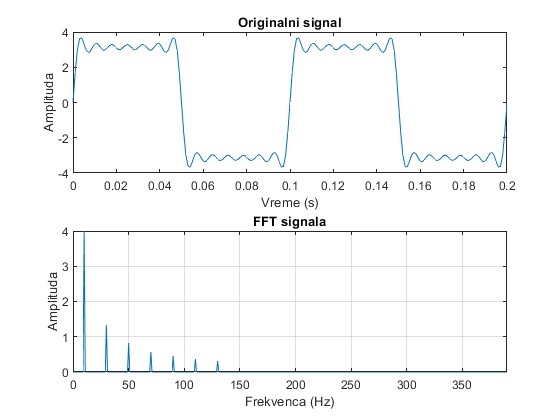
Slika 3.1. Analogni i digitalni signal

## 3.2 Furijeova transformacija

Korišćenjem kvantizovanog i semplovanog vremenskog oblika signala možemo da čuvamo podatke o signalu u digitalnoj formi, ali radi lakše obrade signala koristimo frekvencijski opseg. Signal konvertujemo iz vremenskog u frekvencijski oblik pomoću Furijeove transformacije. Furijeova transformacija signala f(t) računa se na sledeći način:

za analogni signal. Fjω je kompleksna veličina. Njen moduo nazivamo sespektralna gustina amplituda, a argument spektralna gustina faza. Za digitalni signal Furijeova trasformacija glasi:

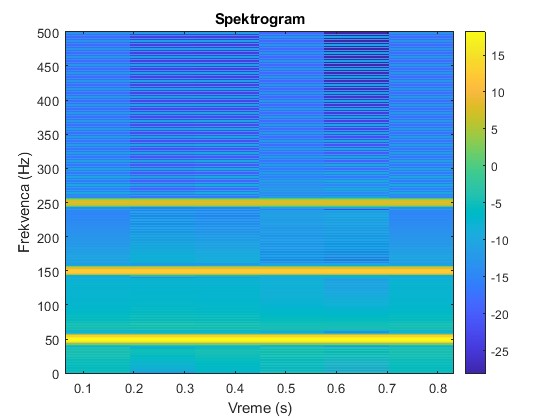
Furijeovom transformacijom dobijamo prikaz frekvenci koje su prisutne u našem signalu i njihove amplitude.



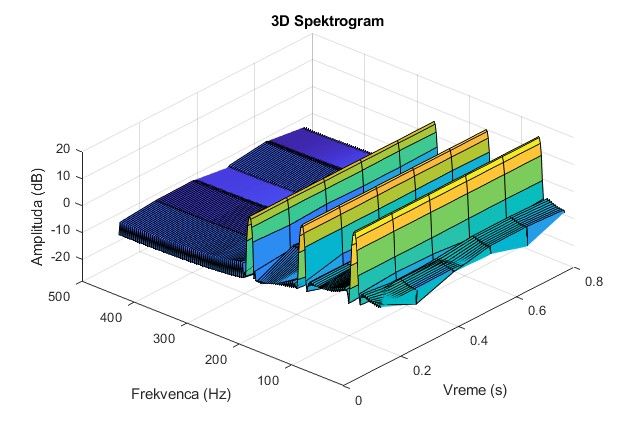
Slika 3.2. Signal u vremenskom i frekvencijskom obliku

## 3.3 Spektrogram

Spektrogram je vizuelna reprezentacija frekvenci u signalu i njihovih promena u vremenu. Na Y osi nam se nalaze frekvence, na X osi vreme, a boje predstavljaju intenzitet, odn. amplitude na datim frekvencama. Osnova jednostavnijih programa za prepoznavanje numera koristi mel-spektrogram, koji je kombinacija obicnog spektrograma, mel baze filtara i dB skale.



Slika 3.3. Spektrogram



Slika 3.4. 3D prikaz spektrograma

Sa slika 3 i 4 vidimo istaknute prisutne frekvence iz signala. Pozicije ovih vrhova su nam ključni podaci koje koristimo radi repliciranja pojednostavljene verzije *Shazam* algoritma za kreiranje otiska jedne numere.

## 3.4 Hash-ovanje i njegova primena u aplikaciji Shazam

Hash-ovanje je proces konvertovanja složenih podataka u kraći i sažet oblik, tzv. hash vrednost, koja može efikasno da se koristi za poređivanje, pretragu i identifikaciju. U kontekstu prepoznavanja muzike, hash-ovanje omogućava brzo pronalaženje pesama u velikim bazama podataka bez potrebe za direktnim poređenjem celokupnih audio zapisa.

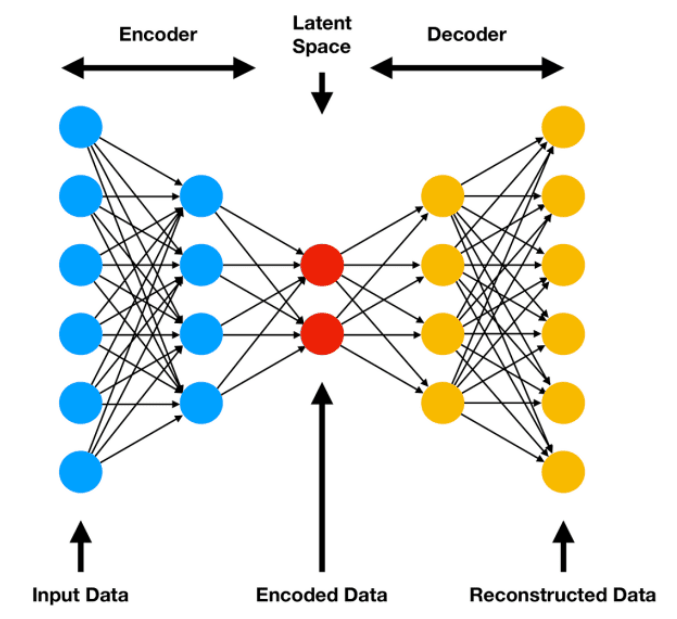
U aplikaciji *Shazam*, hash-ovanje se koristi za kreiranje skraćenih jedinstvenih otisaka (fingerprint) svake pesme. Algoritam nakon identifikacije vrhova u spektrogramu audio snimka i formiranja parova vrhova koji od informacija sadrže frekvencije prvog i drugog vrha i vremensku razliku između njih konvertuje te podatke u skraćeni oblik pomoću hash algoritma.

Ovi podaci se kombinuju u hash vrednost, koja predstavlja sažet i robustan opis karakterističnog dela pesme. Kada korisnik snimi deo pesme, algoritam izračunava hash vrednosti za snimak i znatno brže ih upoređuje sa hash vrednostima u bazi, čime se pronalazi najverovatnija podudarnost. Ova metoda omogućava brzo, pouzdano i memorijski efikasno prepoznavanje muzike, čak i u prisustvu buke i kratkih snimaka.

U našem kodu nećemo koristiti keširanje zato što koristimo znatno manju bazu sa kojom poredimo i uvodjenje ovakvog algoritma ne donosi ubrzanja u ovom slučaju.

## 3.5 Autoenkoder

Autoenkoder predstavlja specifičnu arhitekturu veštačkih neuralnih mreža čija je osnovna namena učenje efikasne reprezentacije podataka. Za razliku od klasičnih mreža, čiji je cilj klasifikacija ili predikcija na osnovu ulaznih podataka, autoenkoder se obučava tako da reprodukuje sopstveni ulaz. Međutim, poenta nije u samom kopiranju, već u činjenici da mreža mora da prođe kroz latentni prostor smanjene dimenzionalnosti, čime se ona primorava da nauči najvažnije osobine i obrasce koji definišu podatke. Upravo ta sposobnost razlikovanja bitnog od nebitnog čini autoenkodere veoma korisnim u brojnim oblastima, od smanjenja dimenzionalnosti i kompresije podataka, preko detekcije anomalija, do obrade slike i zvuka.

Osnovna struktura autoenkodera sastoji se od tri dela: enkodera, latentnog sloja i dekodera. Enkoder ima zadatak da mapira ulazne podatke u prostor manje dimenzionalnosti, pri čemu uklanja redundantne i manje važne informacije. Latentni prostor predstavlja srž sistema i sadrži kompresovanu verziju ulaza. Dekoder zatim pokušava da na osnovu te kompresovane reprezentacije rekonstruiše podatke u obliku što sličnijem originalu. Ovakav proces podseća na način na koji čovek pamti informacije. Pamte se ključne karakteristike i obrasci, dok se nebitni detalji zanemaruju.

Slika 3.5. Arhitektura autoenkodera

Trening autoenkodera zasniva se na principu minimizacije funkcije greške, koja meri razliku između originalnog i rekonstruisanog podatka. Najčešće korišćena mera greške je srednja kvadratna greška (MSE – *mean squared error*), iako se u zavisnosti od vrste podataka i problema, koriste i druge funkcije gubitka. Proces obuke podrazumeva više epoha, pri čemu se mreža iterativno prilagođava, menjajući težine neurona putem algoritama optimizacije poput stohastičkog gradijentnog spusta ili Adam metode. Tokom vremena, mreža postaje sve bolja u pronalaženju reprezentacija koje verno čuvaju informacije potrebne za kasniju rekonstrukciju.

Različite varijante autoenkodera razvijene su u skladu sa različitim istraživačkim potrebama. Klasični autoenkoder predstavlja najjednostavniji oblik i koristi se kao osnova za razumevanje koncepta. Sparse autoenkoder uvodi regularizaciju kako bi aktivacije neurona u latentnom prostoru bile retke, čime se mreža dodatno usmerava da uči efikasnije reprezentacije. Variacione verzije autoenkodera (VAE) dodaju probabilistički pristup, omogućavajući generisanje novih podataka sličnih treniranom skupu. Pored ovih, razvijeni su i konvolucioni autoenkoderi, koji su naročito prilagođeni za obradu slika, kao i kontraktivni autoenkoderi, koji latentni prostor čine otpornijim na male promene u ulazu.

Posebnu pažnju u ovom radu zaslužuje *denoising* autoenkoder (DAE). Njegova osnovna ideja jeste učenje sistema da ukloni šum iz podataka. U praksi se ulazni podaci namerno kontaminiraju određenom vrstom šuma, a mreža se trenira da iz takvog ulaza rekonstruiše čisti original. Na taj način mreža ne samo da uči da prenosi informacije iz ulaza na izlaz, već i da razlikuje korisne podatke od smetnji koje ih narušavaju. Ovakav pristup ima značajnu primenu u audio i video obradi, medicinskoj dijagnostici, kao i u svim oblastima gde su podaci često izloženi degradaciji ili smetnjama.

Teorijski posmatrano, *denoising* autoenkoder se sastoji od istih delova kao i klasičan autoenkoder, ali sa specifičnim načinom pripreme podataka. Tokom treninga, mreži se na ulaz prosleđuje kontaminiran podatak, dok cilj ostaje originalna, čista verzija istog. Enkoder u tom slučaju ima zadatak da kroz latentnu reprezentaciju pronađe obrasce koji su stabilni i otporni na šum, dok dekoder mora da generiše izlaz koji je što bliži neiskvarenoj verziji. Time mreža implicitno uči da razlikuje relevantne informacije od šuma, odnosno da uspostavlja filter između korisnog i neželjenog sadržaja.

U ovom radu se nećemo fokusirati na dublju primenu autoenkodera. Autoenkoderi su pored *Hashing* algoritma najbolje naredne nadogradnje za ovaj sistem. U kodovima postoji implementacija osnovne verzije DAE autoenkodera, koji se koristi za otklanjanje šuma.

# Arhitektura i implementacija sistema

## 4.1 Obrada zvuka

Prvi korak u implementaciji sistema za prepoznavanje muzike je snimanje zvuka i njegovo osnovno preprocesuiranje. Ovaj deo sistema odgovoran je za prikupljanje audio uzorka iz spoljašnje sredine putem mikrofona ili iz već postojećih fajlova, njegovo čišćenje od šuma i pripremu za kasniju analizu i poređenje sa bazom podataka.

U našem testiranju smo se fokusirali na simulirane audio signale formiranje normalizovanjem audio signala pesama koje imamo, potom dodavanjem šuma različitih koeficijenata na ovako dobijen signal. Četiri nivoa šuma koje smo koristili su:

* Bez šuma (0% snage korisnog signala)
* Nizak nivo šuma (0.1% snage korisnog signala)
* Srednji nivo šuma (1% snage korisnog signala)
* Visok nivo šuma (10% snage korisnog signala)

Šum smo formirali korišćenjem *random* funkcije i množenjem dobijenog signala sa koeficijenom pre dodavanja na već postojeće signale u programskom jeziku *Python*. Uz njih korišćene su i biblioteke *librosa* i *scipy*. Biblioteka *librosa* nam omogućuje da iz lokalnih fajlova unesemo amplitudske vrednosti u kod. Audio signal sagledamo kao mono kanal, kako bi se pojednostavio proces upoređivanja sa otiscima iz baze, koji su takođe jedno-kanalni. Iako stereo signal može sadržati dodatne informacije o prostornosti zvuka, za prepoznavanje melodije one nisu relevantne.

Jedan od čestih izazova u radu sa zvukom iz realnog sveta jeste prisustvo pozadinske buke. Tokom snimanja, mikrofon registruje i neželjene zvukove iz okoline, kao što su šum ventilatora, govor u pozadini i saobraćaj. Ovi faktori mogu negativno uticati na tačnost prepoznavanja. U našem slučaju svi ovi faktori su pokriveni generičkim šumom koji smo simulirali i dodali na koristan signal. Kako bi se umanjio uticaj ovih smetnji moguća je implementacija funkcija za redukciju šuma, koristeći biblioteku *noisereduce*. Ova biblioteka koristi statističke metode procene spektralne gustine buke kako bi identifikovala i suzbila neželjene frekvencijske komponente. Po potrebi, dodatno se može koristiti i visokopropusni filter kako bi se eliminisali niskofrekventni šumovi, ispod 60 Hz, koji često potiču od električnih uređaja i fizičkog kontakta sa mikrofonom. Za naše potrebe nismo dalje otklanjali šum radi testiranja sposobnosti sistema u ovim uslovima.

Svaki signal se normalizuje pre svake dalje obrade. Ovaj proces podrazumeva skaliranje amplituda tako da maksimalna vrednost signala dostigne određeni zadati nivo, u našem slučaju ±1. Normalizacija omogućava da se signali različitih jačina dovedu na uporedivu skalu, čime se izbegava da neki signali dominiraju prilikom upoređivanja otisaka. Normalizacija se vrši prema L2 normi, euklidskoj dužini vektora, što dodatno pomaže prilikom izračunavanja udaljenosti među otiscima u kasnijim fazama.

Za potrebe prepoznavanja, umesto da se koristi celokupan audio uzorak kao jedinstveni otisak, signal se segmentiše pomoću kliznog prozora. Svaki prozor pokriva određeni vremenski interval, a pomera se unapred za fiksiran broj uzoraka. Na svaki segment se posebno primenjuju algoritmi za izdvajanje otiska, čime se omogućava:

* robusnost na šumove u samo jednom delu snimka,
* mogućnost poređenja delova pesme, ne samo cele,
* višestruki pokušaji prepoznavanja iz jednog uzorka.

Na ovaj način značajno se povećava tačnost prepoznavanja i otpornost na lokalne smetnje.

## 4.2 Kreiranje digitalnog otiska (*Fingerprinting*)

Nakon što je zvuk snimljen, uklonjen šum, normalizovan i segmentisan, svaki pojedinačni segment se koristi za kreiranje digitalnog otiska, numeričke reprezentacije karakterističnih osobina zvučnog signala. Ovi otisci predstavljaju osnovu za prepoznavanje muzike omogućujući efikasno poređenje snimljenog audio uzorka sa uzorcima iz baze poznatih numera.

Za formiranje otiska, u ovom radu koristi se tehnika ekstrakcije mel-frekventnih kepstralnih koeficijenata (MFCC). MFCC je široko primenjen u sistemima za obradu govora i muzike, jer efikasno sažima oblik spektralnog sadržaja signala u mali broj amplituda koje zadržavaju ključne informacije o njegovoj zvučnoj strukturi.

Računanje MFCC obuhvata sledeće korake:

* **Primena STFT (Short-Time Fourier Transform)** na segment signala, čime se dobija spektrogram.
* **Primena mel filter banke** na spektrogram, koja naglašava frekvence na način sličan ljudskom sluhu.
* **Logaritmovanje dobijenih vrednosti** kako bi se istakle relativne razlike između komponenti.
* **Diskretna kosinusna transformacija (DCT)**, kojom se sažima informacija i dobija niz MFCC koeficijenata.

Rezultat ovog procesa je vektor dužine 13 do 40, koji predstavlja otisak tog audio segmenta.

Za pesme u bazi podataka, svaka numera se učitava iz WAV fajla, obrađuje na isti način kao snimak: redukcija šuma, normalizacija i segmentacija. Zatim se za svaki segment računaju i čuvaju njegovi MFCC koeficijenti. Dobijeni niz vektora za svaku pesmu se čuva u JSON formatu kao mapa.

Ovaj pristup omogućava:

* **brz pristup** otiscima bez ponovnog učitavanja WAV fajlova,
* **kompatibilnost** sa različitim frekvencijama snimanja,
* **štednju memorije** i optimizaciju vremena pretrage.

U okviru sistema implementirana je i mogućnost vizuelizacije spektrograma i MFCC koeficijenata pomoću biblioteka matplotlib i librosa.display. Ova vizualizacija služi za dijagnostiku i provere, te omogućava pregled frekvencijske strukture signala i potvrdu kvaliteta otiska.

## 4.3 Baza podataka

Baza podataka predstavlja ključni deo sistema za prepoznavanje muzike, jer sadrži unapred obrađene i sačuvane otiske poznatih numera, koji služe kao referenca prilikom poređenja sa snimljenim uzorkom.

Za potrebe ovog rada, baza je realizovana kao JSON fajl koji omogućava jednostavno čuvanje i unošenje podataka u strukturisanom formatu. Struktura JSON baze je sledeća:

* Ključevi su nazivi audio fajlova (pesama).
* Vrednosti su liste vektora MFCC koeficijenata, gde svaki vektor predstavlja otisak jednog segmenta te pesme.

**{**

**"naziv\_pesme\_1.wav": [**

**[mfcc\_vektor\_1\_segmenta],**

**[mfcc\_vektor\_2\_segmenta],**

**...**

**],**

**"naziv\_pesme\_2.wav": [**

**[mfcc\_vektor\_1\_segmenta],**

**[mfcc\_vektor\_2\_segmenta],**

**...**

**],**

**...**

**}**

U bazi se čuvaju MFCC otisci za svaki segment svake pesme. Svaki otisak je niz realnih brojeva koji opisuju frekvencijsku strukturu audio signala u određenom vremenskom prozoru. Snimci se obrađuju tako što se segmentuju na delove određene dužine za svaki segment se računaju MFCC koeficijenti, zatim se ti vektori normalizuju i skladište. Čuvanjem ovih otisaka unapred, izbegava se potrebno vreme za ponovno obrađivanje kompletnih audio fajlova prilikom svakog prepoznavanja, što ubrzava rad sistema.

Proces kreiranja baze podataka je sledeći: prvo se učitaju pesme iz foldera *songs/*  kao WAV fajlove. Zatim se na svakom pojedinačnom fajlu vrši redukcija šuma i normalizacija. Audio signal se deli na preklapajuće vremenske segmente i za svaki segment se računaju MFCC koeficijenti. Vektori MFCC se skaliraju da imaju jediničnu dužinu. Dobijeni nizovi vektora se upisuju u JSON datoteku, pod ključem koji odgovara imenu pesme.

Ovim postupkom se obezbeđuje efektivna i brzo dostupna baza za poređenje pri prepoznavanju novih audio uzoraka. Za naše potrebe kreirane su četiri baze, po jedna za svaku dužinu segmenta koje ćemo dalje koristiti od 1, 2, 4 i 8 sekundi.

## 4.4 Prepoznavanje

Prepoznavanje muzike zasniva se na poređenju otiska snimljenog audio uzorka sa otiscima pesama iz baze podataka. Cilj ovog procesa je identifikacija pesme koja najviše odgovara snimku, odnosno pronalaženje najboljeg poklapanja.

Za poređenje otisaka koriste se metrike koje procenjuju sličnost između numeričkih nizova. Jedna od najčešće korišćenih metrika je euklidsko rastojanje, koje meri razdaljinu između dva vektora u višedimenzionalnom prostoru. Što je ta udaljenost manja, to su vektori sličniji. Druga često korišćena metoda je kosinusna sličnost, koja meri ugao između dva vektora i daje vrednost u opsegu od -1 do 1, pri čemu 1 označava maksimalnu sličnost.

U ovom radu mi se više fokusiramo na euklidsko rastojanje. Njega računamo sledećom formulom:

Otisak snimka, MFCC vektor, se zatim poredi sa otiscima svih segmenata svih pesama u bazi. Za svaku pesmu se beleži najmanje rastojanje između nekog njenog segmenta i segmenta iz snimka. Na osnovu tih vrednosti se zatim računa ukupna mera sličnosti između snimka i cele pesme kao suma udaljenosti ovih segmenata i najmanje razlike na odredjenom segmentu.

Kao konačan rezultat, pesma sa najizraženijim podudaranjem se označava kao najverovatniji kandidat. Dodatni pragovi, poput maksimalne dozvoljene udaljenosti ili minimalnog broja poklapanja, koriste se kako bi se smanjio broj lažno pozitivnih rezultata. Ukoliko nijedna pesma ne zadovoljava ove uslove, smatra se da snimak nije prepoznat ili da pesma ne postoji u bazi. Na ovaj način se postiže ravnoteža između tačnosti i otpornosti sistema na šum, distorziju i kratke promene u signalu.

# Performanse i evaluacija

Evaluacija performansi sistema za prepoznavanje muzičkih numera predstavlja ključni deo razvoja, jer omogućava razumevanje njegovih prednosti, ograničenja i potencijalnih oblasti za poboljšanje. U okviru ovog poglavlja razmatra se preciznost sistema u različitim uslovima snimanja, kao i izazovi koji nastaju usled promena u zvučnom signalu. Transformacijom snimka u digitalni i primenom MFCC funkcije dobijamo otisak trenutne pesme.

U ovoj proceni performansi koriste se snimci bez šuma, direktno importovani iz racunara u program. Svi snimci su normalizovani i izvedene su četiri varijante svakog od snimaka prema količini šuma koja je dodata. Dalje u tekstu referenciran kao nezašumljeni snimak je osnovna pesma bez ikakvih promena, tj. isti signal koji se koristi pri kreiranju baze. Drugi tip signala koji ćemo koristiti je signal sa malom količinom šuma, reda veličine 0.1% snage normalizovanog signala. Treći tip je signal sa srednjom količinom šuma koji ima 1% snage normalizovanog signala i četvrti tip je signal sa velikom količinom šuma koji ima 10% snage normalizovanog signala. Svi šumovi su formirani korišćenjem nasumičnih podataka, funkcije *random* u *Python* programskom jeziku, koji su potom skalirani na svoje respektivne jačine i dodati originalnom signalu. Odatle poredimo dobijeni otisak sa bazom podataka i tražimo najsličniji otisak. Pri kreiranju baza i snimanju zvuka koji poredimo sa bazom segmenti numera su varirani prema dužini i delu pesme iz koje su izvučeni segmenti. Prema dužini segmenti su podeljeni u kategorije od 1s, 2s, 4s i 8s. Segmenti su uzimani iz nasumičnih delova sledećih segmenata pesama:

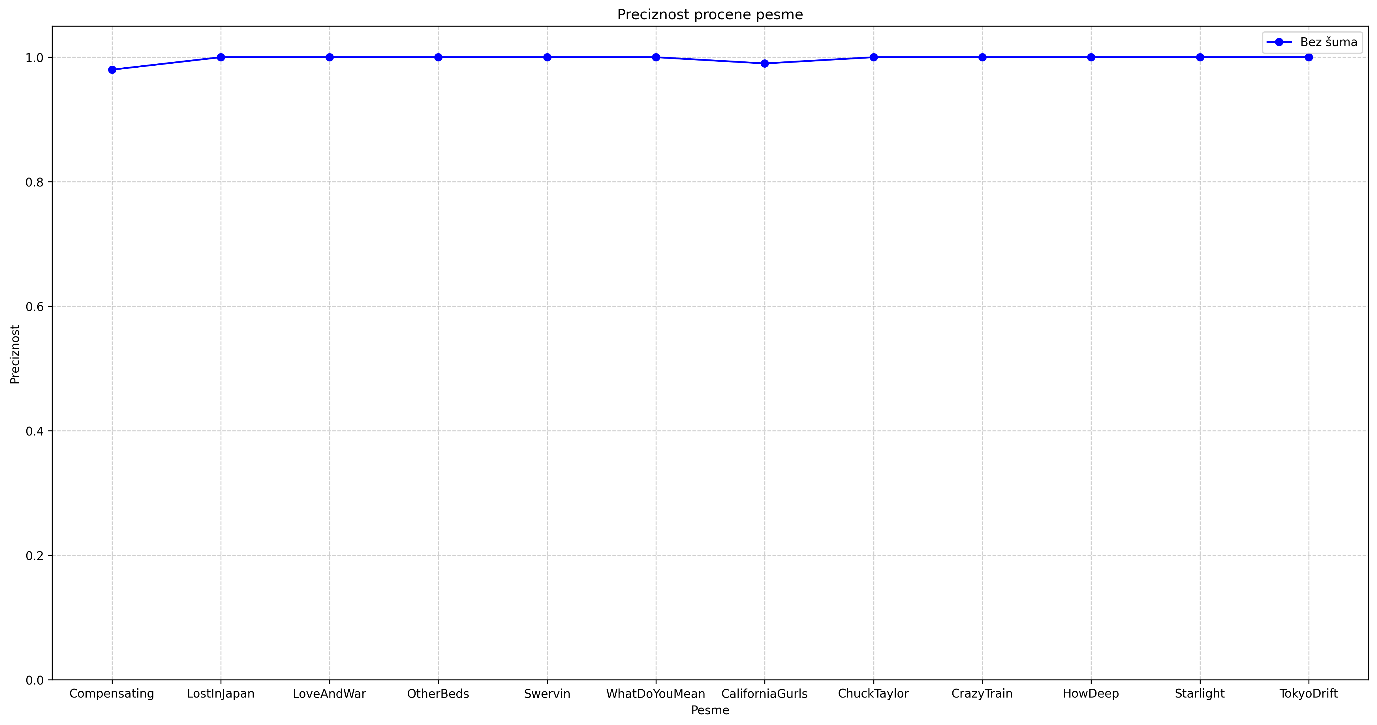
* Početak pesme
* Refren
* Strofa
* Instrumental (ako postoji u pesmi)

Korišćeno je 12 test pesama iz rok, pop, hip-hop, *Rhythm and bass* i elektro žanrova. Svaka od pesama je reformatirana u .wav format i preslušane su pre testiranja radi potvrde očuvanja kvaliteta.

**5.1 Tačnost sistema pri identifikaciji lokalnih fajlova**

Korišćenjem lokalnih fajlova bez prisustva šuma, sistem pokazuje visoku preciznost u procesu identifikacije. Testiranje je sprovedeno tako što je za svaku od 12 pesmama urađeno 100 iteracija. Testovi su ponovljeni na uvodnom delu pesme, strofi, refrenu i instrumentalu za svaku od pesama, kao i za različite dužine snimaka koje koristimo pri kreiranju otiska pesme.

Rezultati pokazuju da je sistem postigao prosečnu tačnost od 99.75%, što ukazuje na pouzdanost metode u idealnim uslovima, kada zvučni signal nije izložen spoljnim ometanjima. Ovo potvrđuje da MFCC otisak, u kombinaciji sa bazom podataka, može da obezbedi stabilne i konzistentne rezultate pri radu sa lokalnim fajlovima. Pri testiranju korišćeni su nasumični delovi pesme, a rezultati nam ukazuju na to da je metod klizajućeg prozora dobro pokrio sve slučajeve.

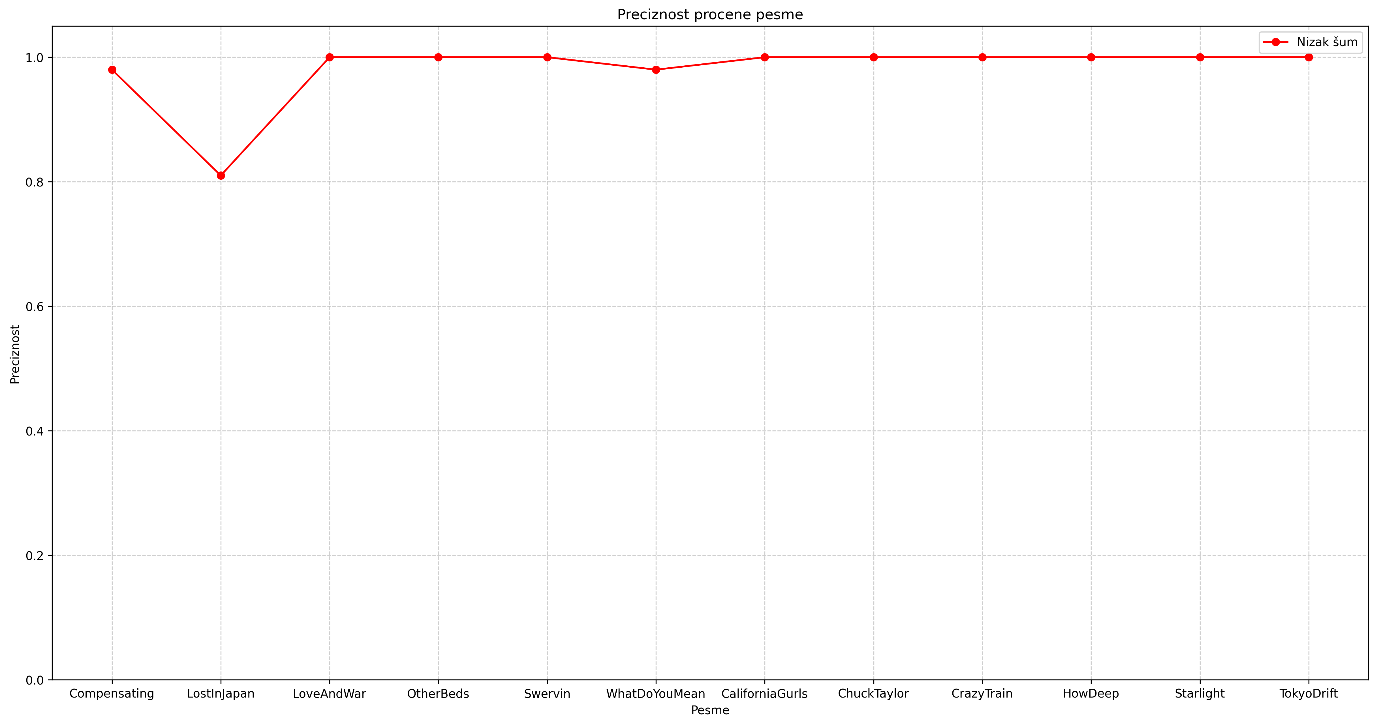


Slika 5.1. Preciznost prepoznavanja pesama iz lokalnih fajlova

Kao što je prikazano na grafu iznad, preciznost je odlična i program je pouzdan pri idealnim uslovima. U narednom segmentu ćemo prikazati pouzdanost programa pri različitim uslovima i uporediti međusobno slučajeve prema pesmama i količini šuma.

## 5.2 Tačnost sistema pri identifikaciji zašumljenih lokalnih fajlova

Kreirani fajlovi sa šumom preko pesme u različitim količinama: 0.001, 0.01 i 0.1 faktor zašumljenja. Testirano je 100 puta za svaku količinu šuma: malo, srednje, mnogo šuma. Testovi su ponovljeni za svaku pesmu.

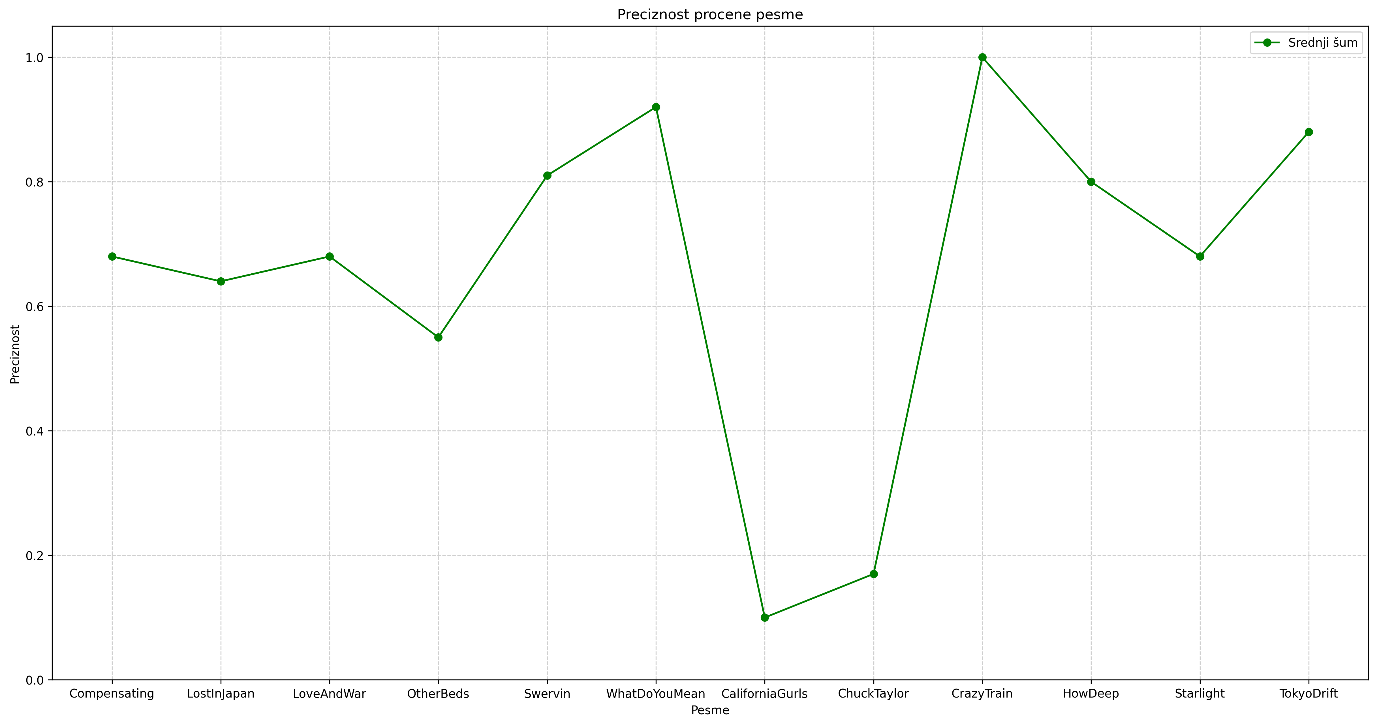


Slika 5.2. Preciznost prepoznavanja blago zašumljenih pesama iz lokalnih fajlova

Pri unosu manje količine šuma, vidimo da je preciznost minimalno oslabila. Preciznost na nivou baze je 98%, što je i dalje odličan rezultat i smatramo ga pouzdanim. Ovakvi rezultati pokazuju da sistem za identifikaciju pesama uspešno prepoznaje obrasce čak i kada je signal delimično degradiran, što ga čini pogodnim za realne uslove rada, gde je prisustvo šuma gotovo uvek neminovno.

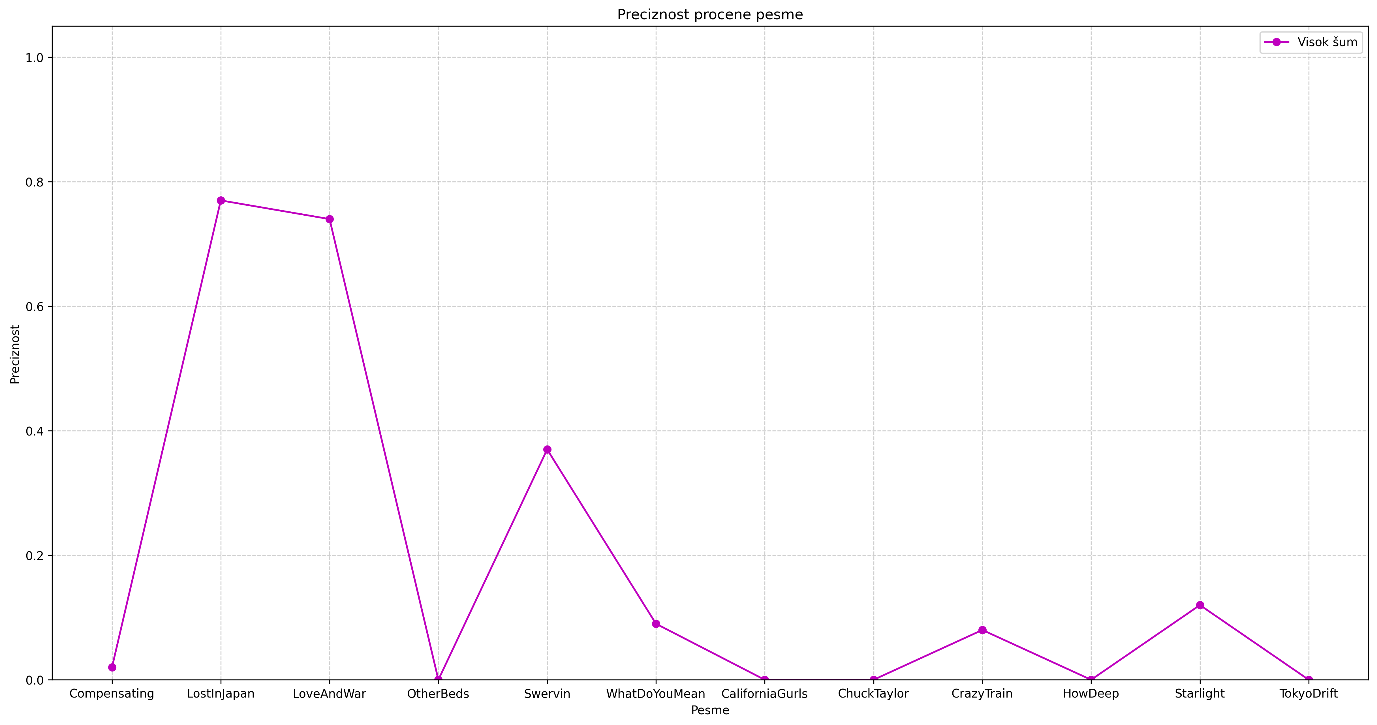
Važno je napomenuti da se visok stepen preciznosti održava zahvaljujući upotrebi robusnih metoda za ekstrakciju karakteristika, kao što su audio otisci (*fingerprints*) i spektrogramske reprezentacije. One omogućavaju softveru da pronađe jedinstvene šablone u pesmama koji su otporni na promene u jačini zvuka, kvalitetu snimka ili spoljašnje smetnje.

Pored same preciznosti, značajan pokazatelj pouzdanosti sistema je i brzina pretrage u bazi podataka. U našem slučaju, identifikacija se obavlja gotovo trenutno, što je od ključnog značaja za krajnjeg korisnika. Softver je optimizovan tako da u realnom vremenu poredi otiske ulaznog signala sa velikim brojem referentnih zapisa, a da pri tome ne dolazi do vidljivog usporavanja.



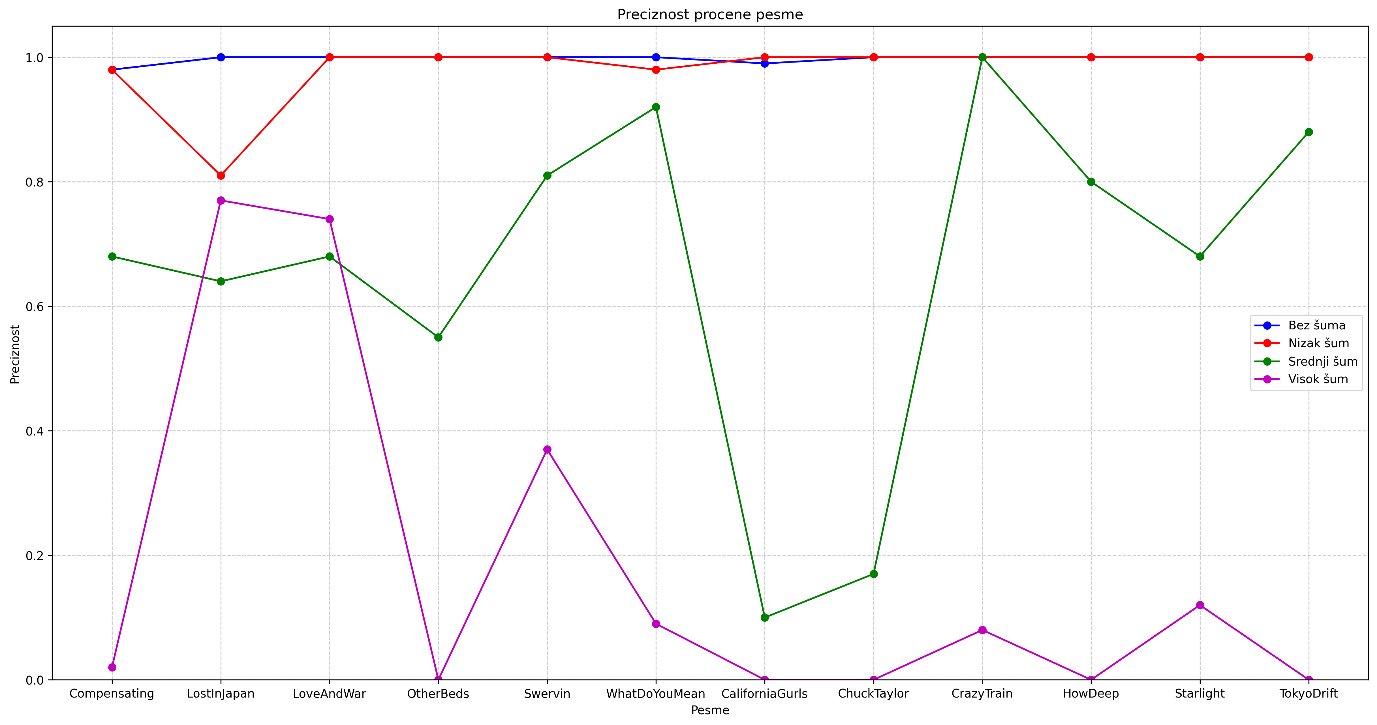
Slika 5.3. Preciznost prepoznavanja umereno zašumljenih pesama iz lokalnih fajlova

Sa slike iznad vidimo da pri srednjem nivou šuma je zabeleženo značajno opadanje preciznosti sistema, pri čemu tačnost prepoznavanja više nije ujednačena. Rezultati pokazuju da uspešnost identifikacije varira od pesme do pesme, što ukazuje na to da određene kompozicije imaju otpornije akustične karakteristike od drugih. Primećeno je da pesme sa izraženijom melodijskom linijom ili jačim ritmičkim obrascima zadržavaju viši nivo prepoznatljivosti, dok su složenije i produkcijski bogatije numere imale veće oscilacije u preciznosti. Ovi nalazi sugerišu da struktura i frekvencijski raspon pesme imaju značajan uticaj na efikasnost identifikacije u prisustvu šuma. Time se otvara prostor za dalja istraživanja o tome kako se sistem može dodatno optimizovati u zavisnosti od tipa muzike. Preciznost ovog sistema je pala na 66%. Pojedine pesme sistem prepoznaje u samo 10% slučajeva, a pojedine u preko 85%. Rezultati nam ukazuju na postojanje karakteristika koje su dovoljno unikatne za pesmu da pri korišćenju tog segmenta pesma je jednoznačno klasifikovana.



Slika 5.4. Preciznost prepoznavanja jako zašumljenih pesama iz lokalnih fajlova

Pri visokom nivou šuma performanse sistema dodatno opadaju, što se vidi po znatno smanjenoj preciznosti prepoznavanja pesama. U ovakvim uslovima, identifikacija postaje znatno otežana jer šum maskira ključne karakteristike signala, pa dolazi do većeg broja pogrešnih ili neprepoznatih rezultata. Ipak, primećeno je da pojedine pesme, naročito one sa dominantnim melodijskim linijama ili jasno izraženim ritmom, i dalje mogu biti prepoznate sa preciznošću iznad 70%. To pokazuje da sistem zadržava određeni nivo robusnosti, ali da je njegova pouzdanost u velikoj meri uslovljena akustičkim svojstvima same kompozicije, kao i nivoom šuma. Ovi rezultati naglašavaju potrebu za dodatnim unapređenjem metoda filtriranja i obrade podataka, kako bi identifikacija bila stabilnija i u ekstremno nepovoljnim uslovima.



Slika 5.5. Preciznost prepoznavanja pesama iz lokalnih fajlova sa varirajućom količinom šuma

Na slici su prikazani rezultati merenja preciznosti sistema u zavisnosti od količine šuma u ulaznim podacima. Može se jasno uočiti da je pri niskom nivou šuma preciznost gotovo nepromenjena u odnosu na snimke bez šuma i kreće se oko 98%, dok se kod srednje količine šuma preciznost značajno smanjuje i pokazuje izražene oscilacije između različitih pesama. Pri visokom nivou šuma rezultati su najlošiji, ali je zanimljivo da pojedine kompozicije i dalje dostižu preciznost iznad 80%. Kako smo primetili da je preciznost pepoznavanja za odredjene pesme viša za viši nivo šuma u odnosu na srednji nivo šuma pokrenut je program sa dodatnim ispisima radi dubljeg razumevanja. Ovaj test je pokazao da program pri visokom nivou šuma najčešće pogađa pesme “*Lost in Japan*” i “*Love and war*” što ukazuje da su rezultati za ove dve pesme nevalidni. Uzimanjem ovoga u obzir preciznost sistema pri visokom nivou šuma je svega 7%. Grafički prikaz stoga pruža jasan uvid u opadanje performansi sistema sa porastom nivoa šuma, ali i ističe razlike među samim pesmama. U tabeli ispod su svi ovi grafovi predstavljeni brojčano.

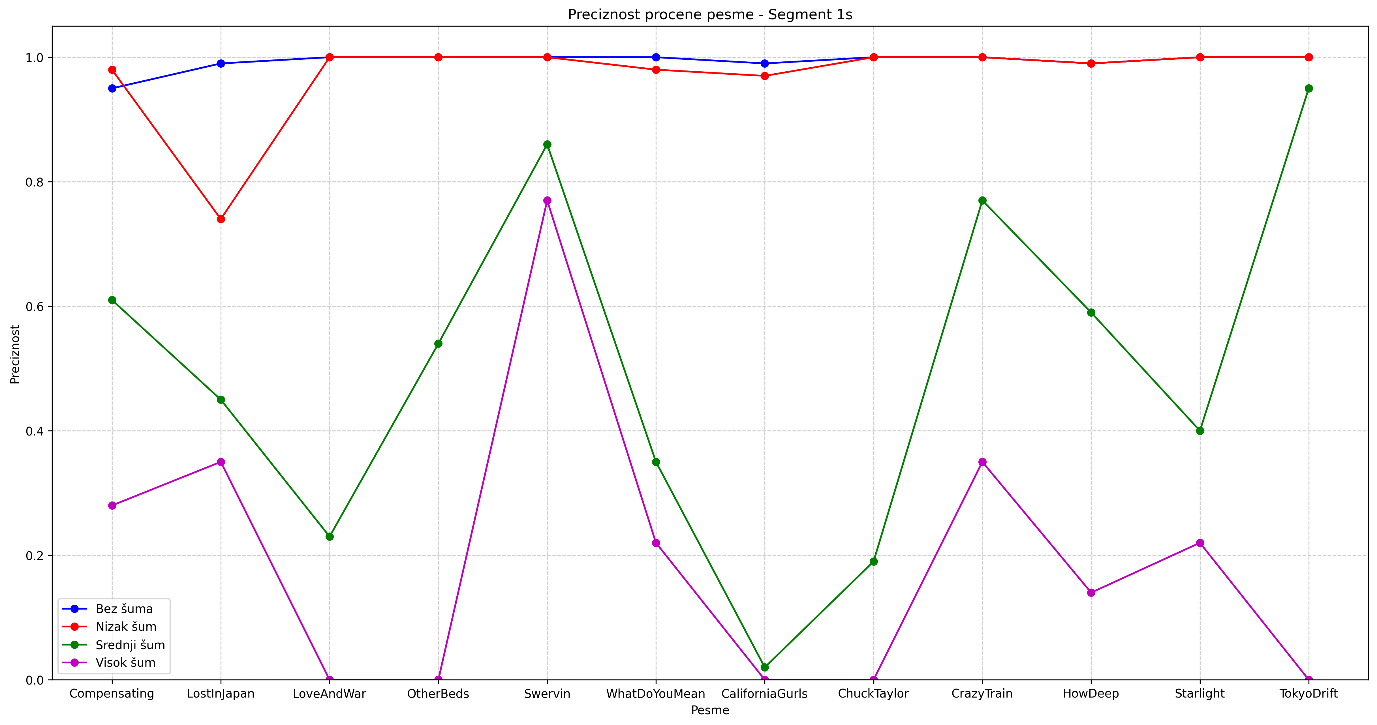
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Naziv Pesme** | **Bez šuma** | **Malo šuma** | **Srednja količina šuma** | **Mnogo šuma** |
| Compensating | 0.98 | 0.98 | 0.68 | 0.02 |
| Lost in Japan | 1.0 | 0.81 | 0.64 | 0.77 |
| Love and War | 1.0 | 1.0 | 0.68 | 0.74 |
| Other Beds | 1.0 | 1.0 | 0.55 | 0.0 |
| Swervin | 1.0 | 1.0 | 0.81 | 0.37 |
| What do you mean | 0.99 | 0.98 | 0.92 | 0.09 |
| California Gurls | 1.0 | 1.0 | 0.1 | 0 |
| Chuck Tailor | 1.0 | 1.0 | 0.17 | 0 |
| Crazy Train | 1.0 | 1.0 | 1.0 | 0.08 |
| How Deep | 1.0 | 1.0 | 0.8 | 0 |
| Starlight | 1.0 | 1.0 | 0.68 | 0.12 |
| Tokyo Drift | 1.0 | 1.0 | 0.88 | 0 |
| **PROSEK** | **0.9975** | **0.9808** | **0.6592** | **0.1825** |

Tabela 5.1. Preciznost prepoznavanja pesama iz lokalnih fajlova sa varirajućom količinom šuma

## 5.3 Tačnost sistema pri korišćenju različitih dužina segmenata

Jedan od ključnih faktora koje koristimo pri kreiranju baze podataka i samih otisaka pesama je dužina segmenta pesme koju koristimo pri izračunavanju svakog otiska. Variranjem ovog faktora dobijamo otiske koji su precizniji za manje segmente pesama, ali koriste znatno manju količinu podataka pri svojoj kreaciji za manju dužinu segmenta. Povećanjem dužine segmenta imamo više podataka za svaki, ali takodje imamo opštiji otisak što nije uvek bolje.

U testiranju je algoritam testiran na dužinama segmenata od 1s, 2s, 4s i 8s. U svim slučajevima je preklapanje segmenata 0.5s. Svih 12 pesama je testirano 100 puta za svaku od ovih dužina u odnosu na bazu podataka kreiranu otiskivanjem segmenata istih dimenzija. Kao i u prethodnom poglavlju pokriveni su slučajevi bez šuma, sa manjom, srednjom i većom količinom šuma za svaku od dužina segmenata.

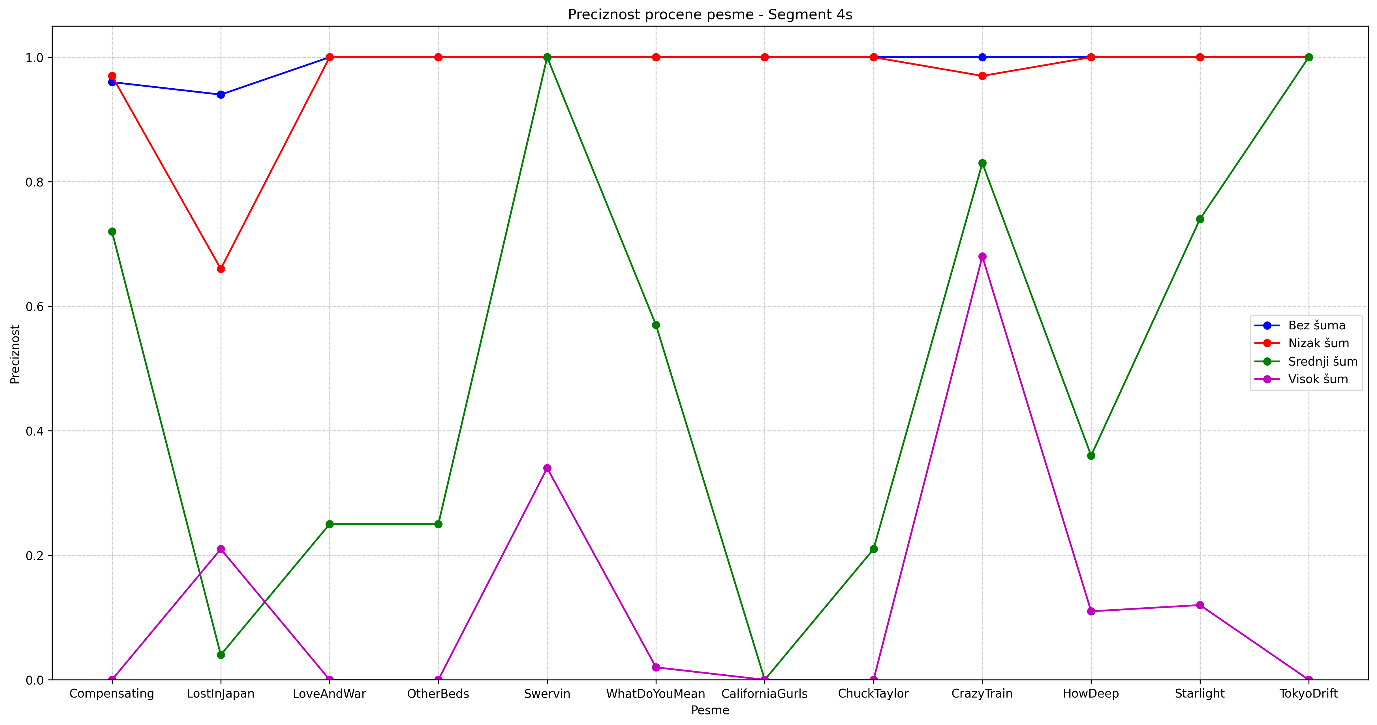


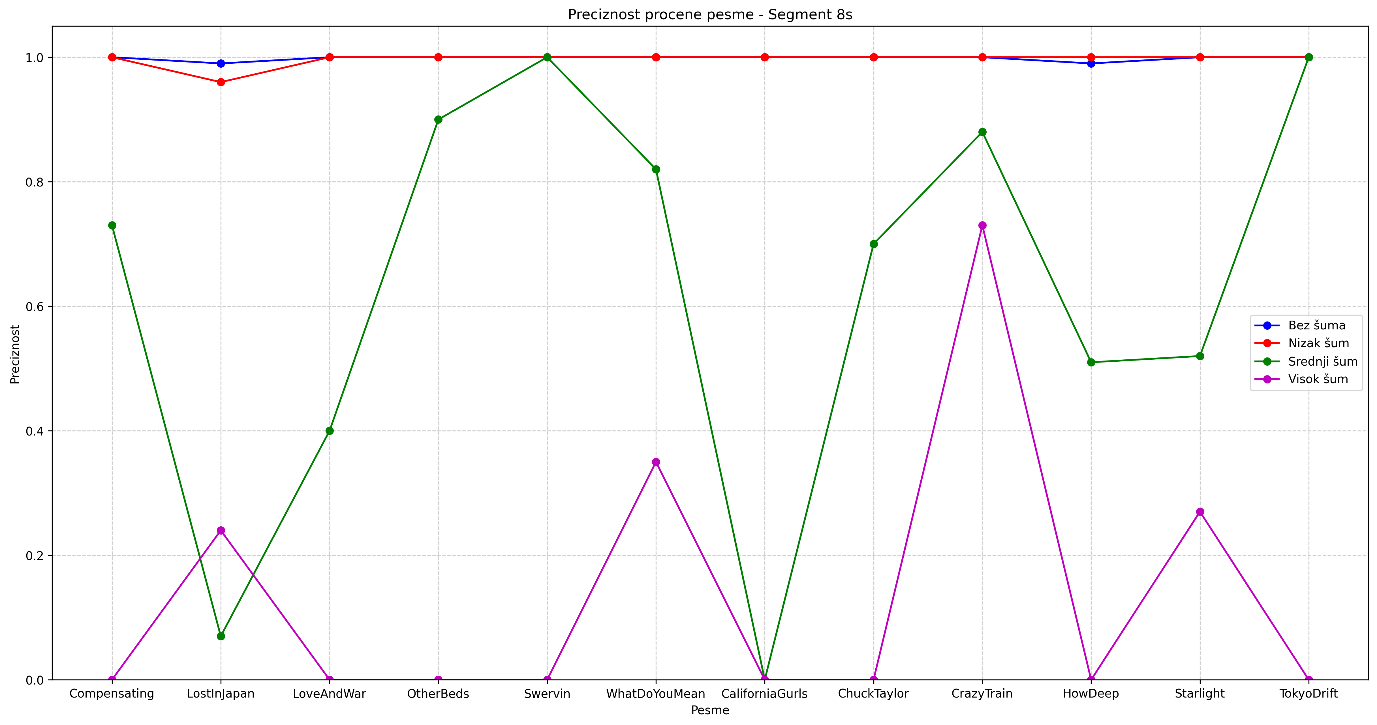
Slika 5.6. Preciznost prepoznavanja pesama iz lokalnih fajlova sa varirajućom količinom šuma – dužina segmenta 1s

A graph with different colored lines

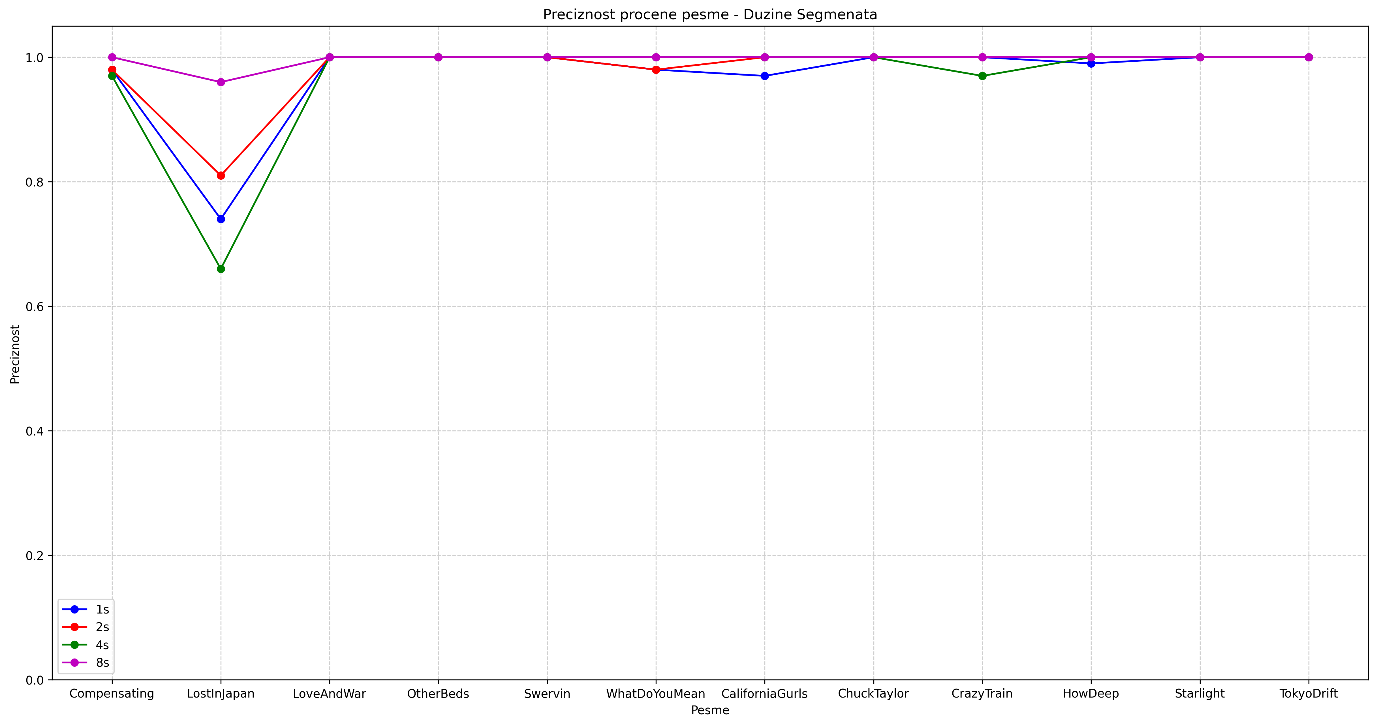
AI-generated content may be incorrect.Slika 5.7 Preciznost prepoznavanja pesama iz lokalnih fajlova sa varirajućom količinom šuma – dužina segmenta 2s

Na slikama 5.6 i 5.7 su prikazane preciznosti prepoznavanja pesama za sve nivoe šuma za dužine segmenata od 1 i 2 sekundi respektivno. Sa slika vidimo da su za otiskivanje bolji rezultati sa segmentima od 2 sekunde. Nezavisno od dužine segmenta u ova dva slučaja rezultati za nezašumljene signale i signale sa niskim nivoom šuma, koeficijenta zašumljenja 0.01, su odlični.

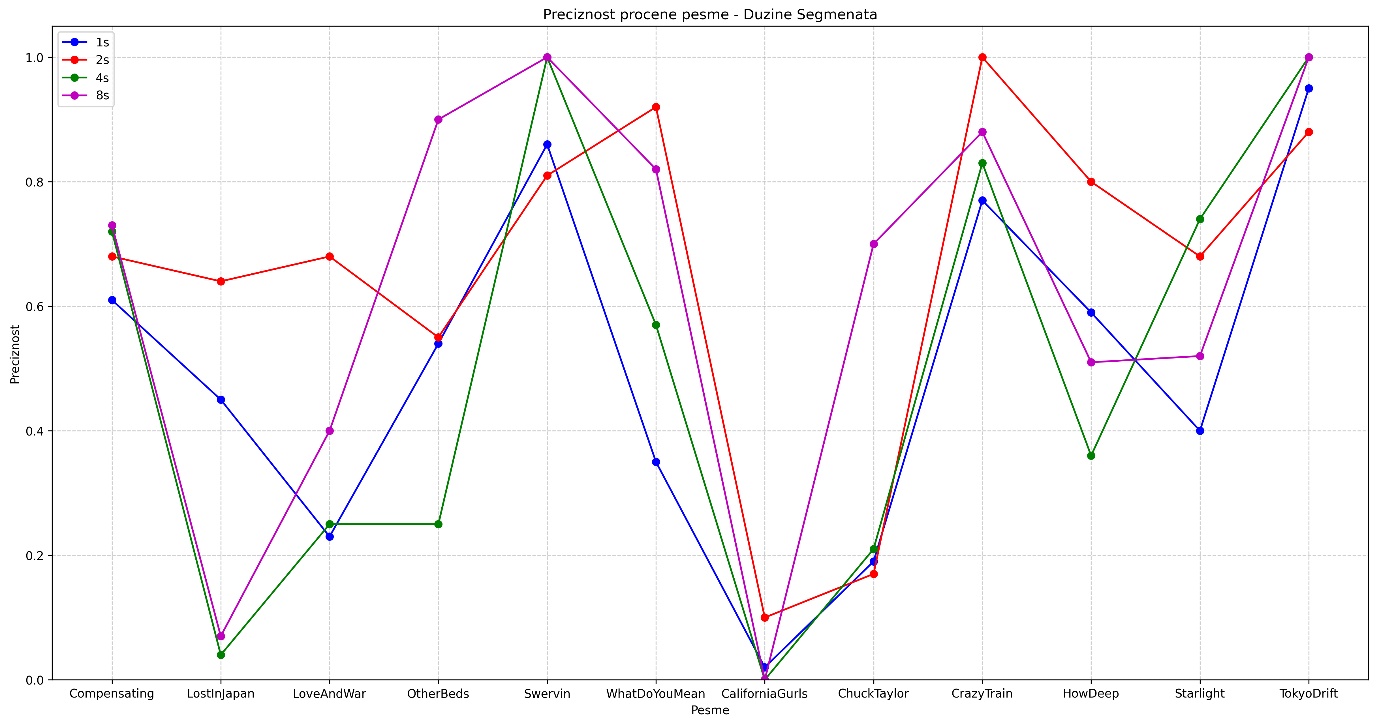
Slika 5.8 Preciznost prepoznavanja pesama iz lokalnih fajlova sa varirajućom količinom šuma – dužina segmenta 4s

Slika 5.9 Preciznost prepoznavanja pesama iz lokalnih fajlova sa varirajućom količinom šuma – dužina segmenta 8s

Na slikama 5.8 i 5.9 su prikazane preciznost prepoznavanja pesama za sve nivoe šuma za dužine segmenata od 4 i 8 sekundi respektivno. Kao i u slučaju za dužinu segmenta od 1 i 2 sekunde vidimo da su nezavisno od dužine segmenta u svim slučajevima rezultati za nezašumljene signale i signale sa niskim nivoom šuma, koeficijenta zašumljenja 0.01, odlični. Do sada nam najbolje rezultate za sigale sa srednjim nivoom šuma pokazuju dužine segmenta od 2 i 8 sekundi. Daljim povećanjem dužine segmenta nije skalabilno na većim bazama, jer varijansa takodje raste.



Slika 5.10 Preciznost prepoznavanja pesama iz lokalnih fajlova sa varirajućom dužinom segmenta – 0.001 koeficijent zašumljenja

Slika 5.11 Preciznost prepoznavanja pesama iz lokalnih fajlova sa varirajućom dužinom segmenta – 0.01 koeficijent zašumljenja

Na slikama 5.10 i 5.11 su prikazane preciznosti algoritma za različite dužine segmenata na maloj i srednjoj količini zašumljenja. Najmanju varijansu i najprecizniju procenu imamo korišćenjem segmenta dužine 2 sekunde. U skoro polovini slučajeva je algoritam sa segmentima dužine 8 sekundi precizniji nego korišćenjem segmenat od 2 sekunde, ali znatno više varira samim tim nije pouzdan, uz to što u prosečnom slučaju daje malo gore rezultate. U produžetku se nalaze tabele sa rezultatima ovih merenja.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Naziv Pesme** | **Dužina segmenta 1s** | **Dužina segmenta 2s** | **Dužina segmenta 4s** | **Dužina segmenta 8s** |
| Compensating | 0.98 | 0.98 | 0.97 | 1.0 |
| Lost in Japan | 0.74 | 0.81 | 0.66 | 0.96 |
| Love and War | 1.0 | 1.0 | 1.0 | 1.0 |
| Other Beds | 1.0 | 1.0 | 1.0 | 1.0 |
| Swervin | 1.0 | 1.0 | 1.0 | 1.0 |
| What do you mean | 0.98 | 0.98 | 1.0 | 1.0 |
| California Gurls | 0.97 | 1.0 | 1.0 | 1.0 |
| Chuck Tailor | 1.0 | 1.0 | 1.0 | 1.0 |
| Crazy Train | 1.0 | 1.0 | 0.97 | 1.0 |
| How Deep | 0.99 | 1.0 | 1.0 | 1.0 |
| Starlight | 1.0 | 1.0 | 1.0 | 1.0 |
| Tokyo Drift | 1.0 | 1.0 | 1.0 | 1.0 |
| **PROSEK** | **0.9717** | **0.9808** | **0.9667** | **0.9967** |

Tabela 5.2. Preciznost prepoznavanja pesama iz lokalnih fajlova sa varirajućom dužinom segmenta – malo šuma

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Naziv Pesme** | **Dužina segmenta 1s** | **Dužina segmenta 2s** | **Dužina segmenta 4s** | **Dužina segmenta 8s** |
| Compensating | 0.61 | 0.68 | 0.72 | 0.73 |
| Lost in Japan | 0.45 | 0.64 | 0.04 | 0.07 |
| Love and War | 0.23 | 0.68 | 0.25 | 0.4 |
| Other Beds | 0.54 | 0.55 | 0.25 | 0.9 |
| Swervin | 0.86 | 0.81 | 1.0 | 1.0 |
| What do you mean | 0.35 | 0.92 | 0.57 | 0.82 |
| California Gurls | 0.02 | 0.1 | 0 | 0.0 |
| Chuck Tailor | 0.19 | 0.17 | 0.21 | 0.7 |
| Crazy Train | 0.77 | 1.0 | 0.83 | 0.88 |
| How Deep | 0.59 | 0.8 | 0.36 | 0.51 |
| Starlight | 0.4 | 0.68 | 0.74 | 0.52 |
| Tokyo Drift | 0.95 | 0.88 | 1.0 | 1.0 |
| **PROSEK** | **0.4967** | **0.6592** | **0.4975** | **0.6275** |

Tabela 5.3. Preciznost prepoznavanja pesama iz lokalnih fajlova sa varirajućom dužinom segmenta – srednji nivo šuma

## 5.4 Tačnost sistema pri analizi specifičnih delova numera

Sistem je testiran na različim segmentima pesama prema strukturi pesme. Svih dvanaest pesmama su pored otiskivanja na nasumičnim delovima podeljene u segmente koji predstavljaju:

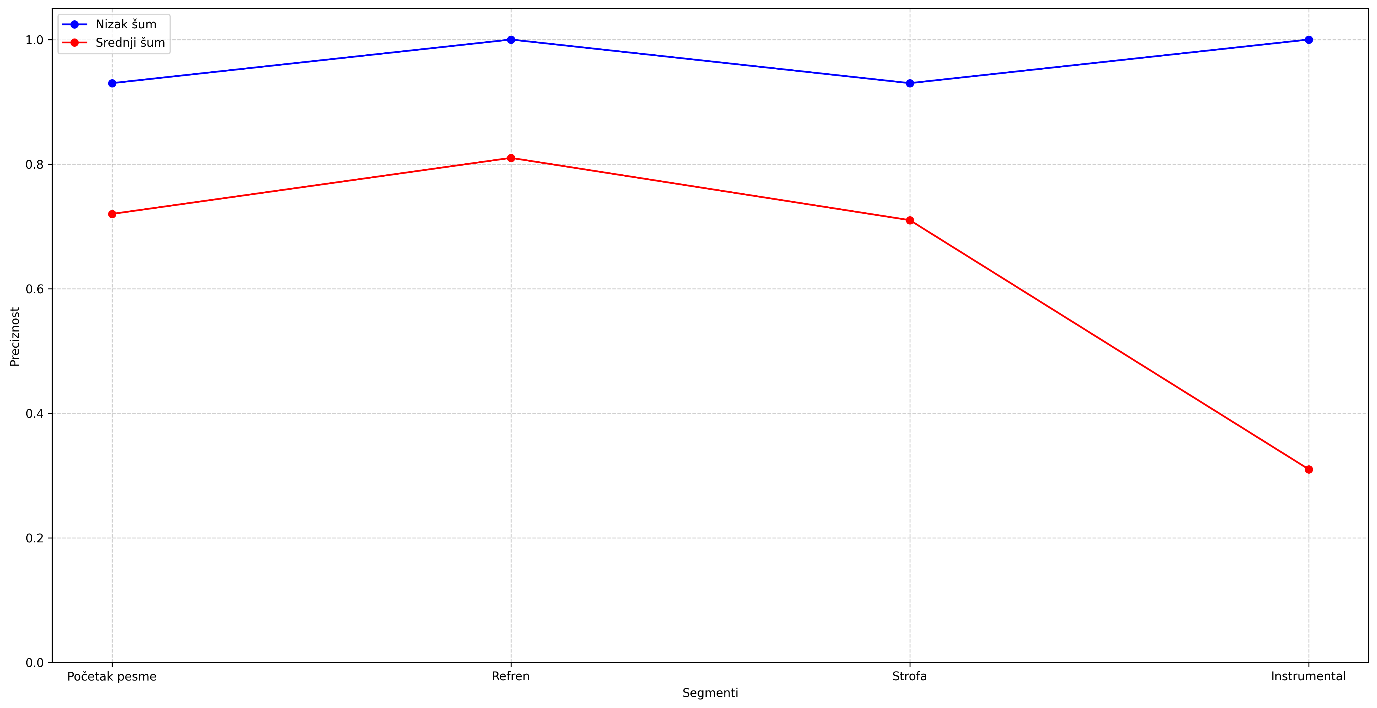
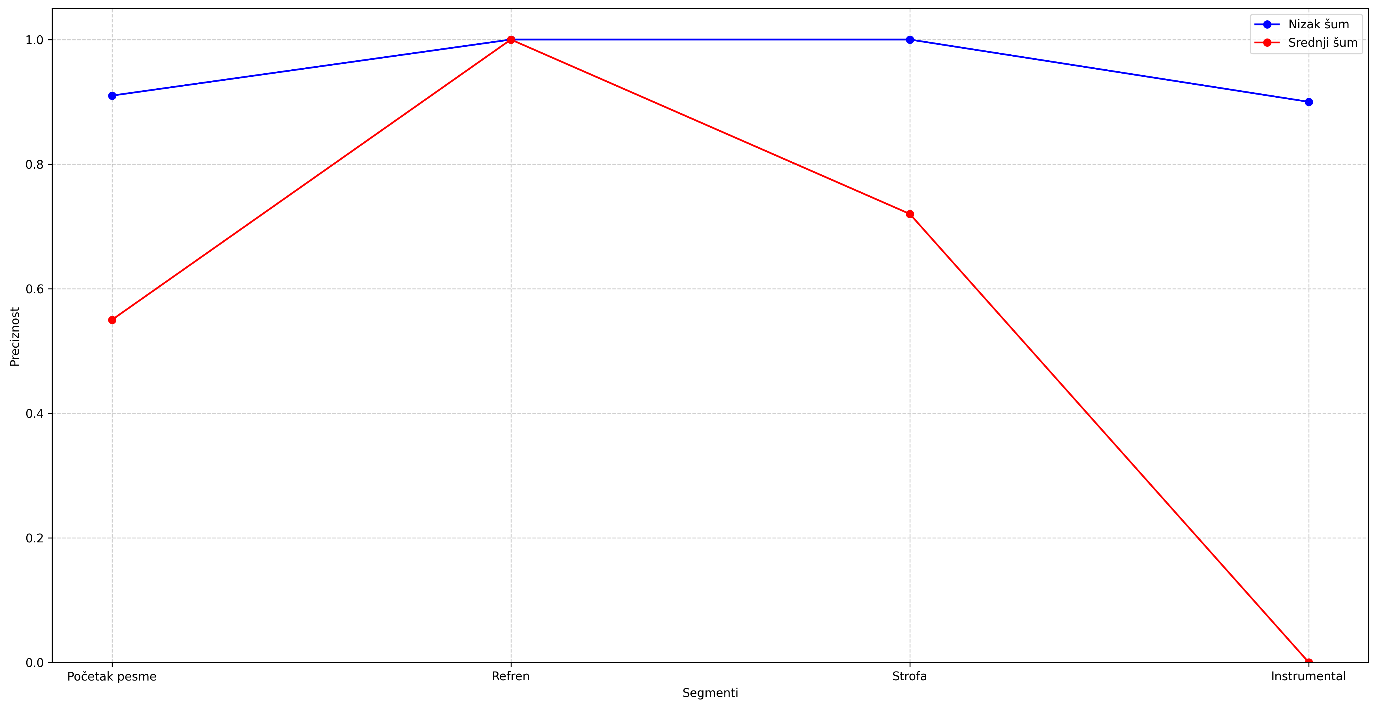
* Početak pesme
* Refren
* Strofu
* Instrumental (ako postoji u pesmi)

Prema ovim podelama koje su obeležene u tabeli 5.4 je program pokrenut da radi identifikaciju sto puta u datom segmentu svake pesme.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Naziv pesme** | **Početak pesme** | **Refren** | **Strofa** | **Instrumental** |
| Compensating | 0:05 - 0:25 | 1:58 - 2:16 | 0:19 – 0:30 | 2:57 – 3:15 |
| Lost in Japan | 0:05 - 0:25 | 0:40 – 0:57 | 0:23 – 0:37 | 0:01 – 0: 20 |
| Love and War | 0:05 - 0:25 | 1:48 – 2:10 | 0:20 - 0:35 | 1:02 – 1:10 |
| Other Beds | 0:05 - 0:25 | 0:49 – 1:18 | 0:02 - 0:32 | x |
| Swervin | 0:05 - 0:25 | 0:10 - 0:31 | 0:33 - 0:59 | 2:56 - 3:04 |
| What do you mean | 0:05 - 0:25 | 0:15 - 0:45 | 0:48 - 1:04 | 0:01 - 0:12 |
| California Gurls | 0:05 - 0:25 | 0:52 - 1:20 | 2:24 - 2:59 | x |
| Chuck Tailor | 0:05 - 0:25 | 1:00 - 1:18 | 0:25 - 0:42 | x |
| Crazy Train | 0:05 - 0:25 | 1:15 - 1:26 | 0:40 - 1:08 | 0:19 - 0:35 |
| How Deep | 0:05 - 0:25 | 0:30-0:50 | 0:01-0:21 | x |
| Starlight | 0:05 - 0:25 | 1:02-1:17 | 0:15-0:52 | x |
| Tokyo Drift | 0:05 - 0:25 | 0:11-0:41 | 0:43-0:57 | 0:01-0:10 |

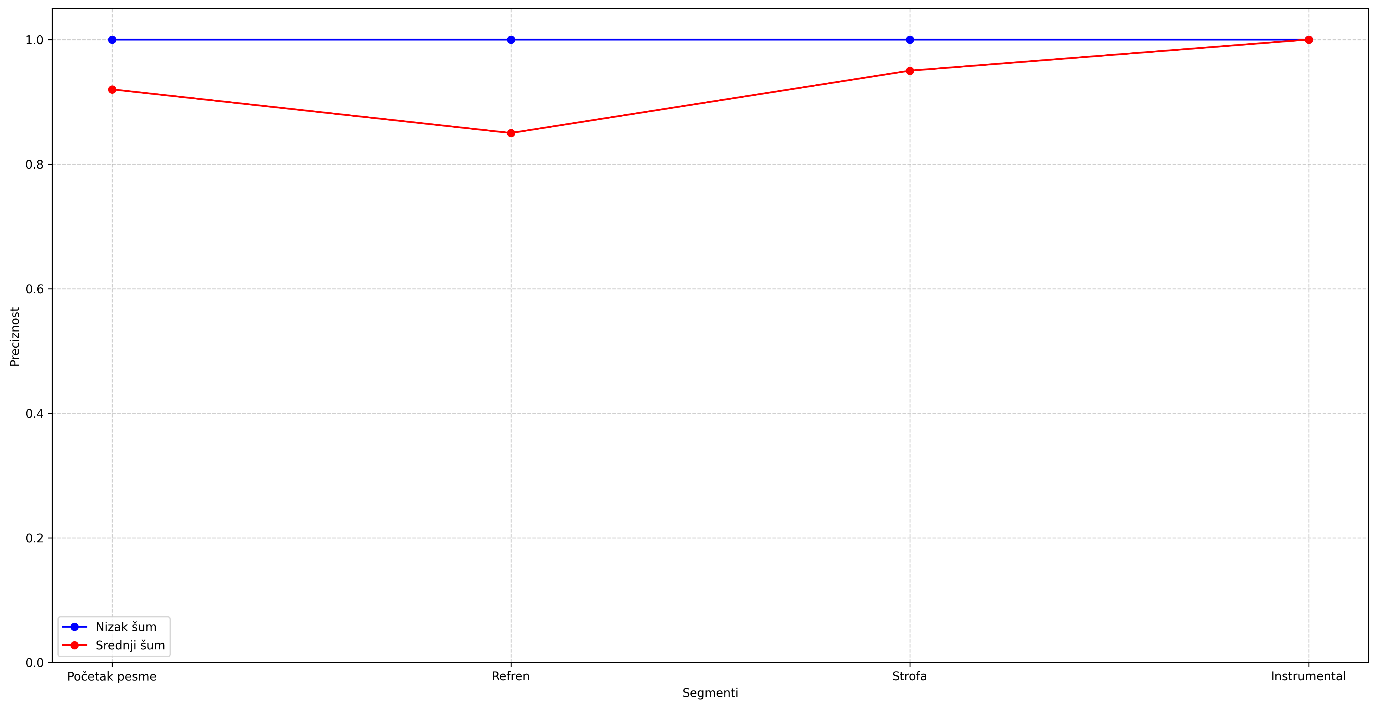
Tabela 5.4 Vremena trajanja segmenata pesama

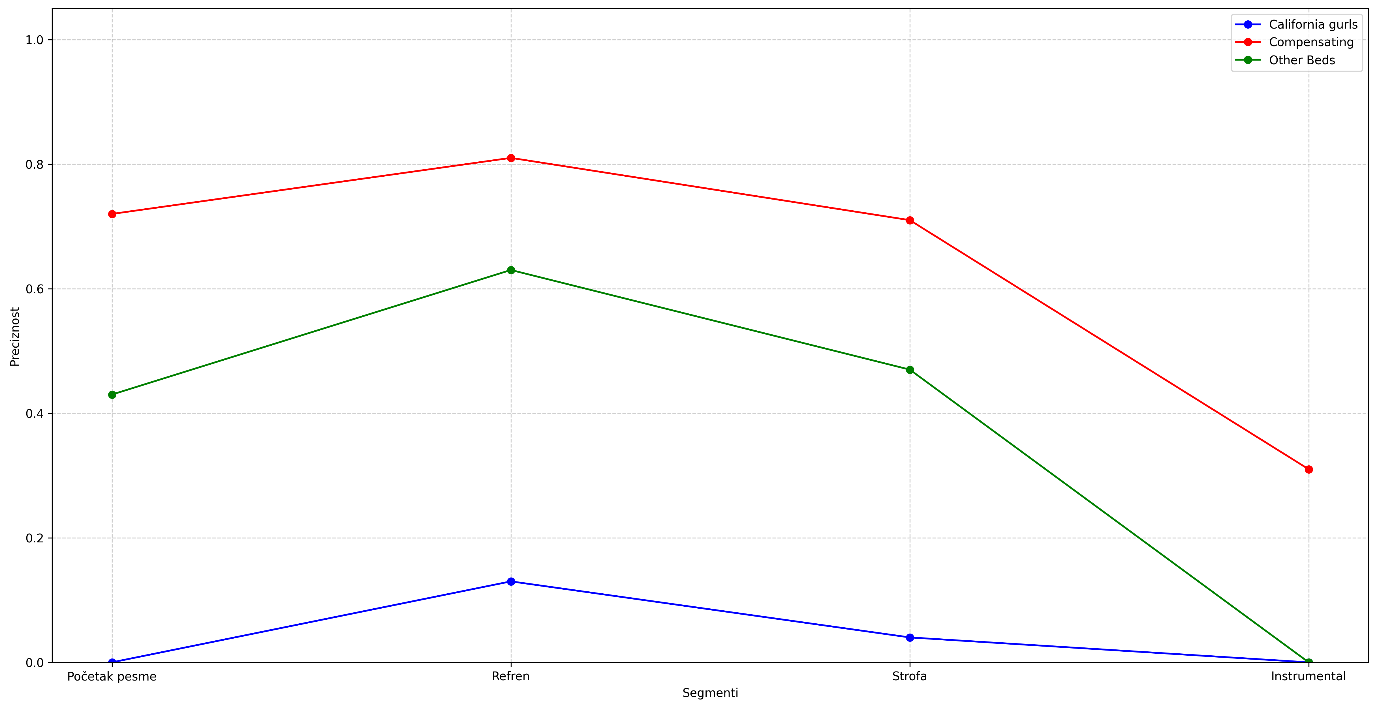
Za ovaj deo evaluacije sistema smo koristili samo širinu segmenta od 2 sekunde, jer se pokazala kao najpreciznija. Pored ovog, merenja su vršena samo na signalima sa malom i srednjom količinom šuma, kako su očekivani rezultati za sve prethodne testove približno identični na signalima bez šuma i onima sa malom količinom, a rezultati signala sa velikom količinom šuma su pokazano nepouzdani.

Slika 5.12 Preciznost prepoznavanja pesme “*Compensating*” na različitim delovima numere

Slika 5.13 Preciznost prepoznavanja pesme “*What do you mean*” na različitim delovima numere

Na slikama 5.12 – 5.15 vidimo da je generalno pravilo da se pesme preciznije odredjuju na segmentima refrena nego strofa. U zavisnosti od pesme instrumentali najviše variraju po parametru preciznosti u odnosu na druge segmente, kao i počeci pesama. Pesme koje imaju sporije i tiše uvode kao što je “*What do you mean*” imaju nižu preciznost prepoznavanja na tom segmentu. Sa druge strane pesme kao što je “*Swervin*” koje imaju glasniji i dinamičniji početak imaju visok nivo preciznosti i u tom segmentu.

Slika 5.14 Preciznost prepoznavanja pesme “*Swervin*” na različitim delovima numere



Slika 5.15 Preciznost prepoznavanja pesma na različitim delovima

Na slici 5.15 instrumentali prikazani nulama ne postoje, pa su samim time predstavljeni kao nule. Ovi grafovi prate pravila da preciznost na početku zavisi od dinamičnosti uvoda, kao i da refren generalno ima malo viši nivo distinkcije u odnosu na strofu i samim time imamo bolju preciznost analizom tih delova pesama. U tabelama ispod se nalaze rezultati merenja za svih dvanaest pesama na sva četiri segmenta za signale sa malom i srednjom količinom šuma. U tabelama je takodje odsustvo instrumentala predstavljeno nulom. Celokupni rezultati iz tabela takodje potvrdjuju ova pravila. Pokazuju da to iako je pouzdano pravilo postoje izuzeci, kao što su “*Swervin*” i “*Other beds*”. Iz tabela takodje vidimo da su i pri prisustvu niskog šuma instrumentali i početak najviše varirajući u zavisnosti od dinamičnosti. Pesma “*Lost in Japan*” ima vrlo tih početak u kome se polako uvode glas i instrumentali i samim time ima najgoru preciznost.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Naziv pesme** | **Početak pesme** | **Refren** | **Strofa** | **Instrumental** |
| Compensating | 0.93 | 1.0 | 0.93 | 1.0 |
| Lost in Japan | 0.83 | 1.0 | 1.0 | 0.84 |
| Love and War | 1.0 | 1.0 | 1.0 | 1.0 |
| Other Beds | 1.0 | 1.0 | 1.0 | 0 |
| Swervin | 1.0 | 1.0 | 1.0 | 1.0 |
| What do you mean | 0.91 | 1.0 | 1.0 | 0.9 |
| California Gurls | 1.0 | 1.0 | 0.99 | 0 |
| Chuck Tailor | 1.0 | 1.0 | 1.0 | 0 |
| Crazy Train | 1.0 | 1.0 | 1.0 | 1.0 |
| How Deep | 1.0 | 1.0 | 1.0 | 0 |
| Starlight | 1.0 | 1.0 | 1.0 | 0 |
| Tokyo Drift | 1.0 | 1.0 | 1.0 | 1.0 |

Tabela 5.5 Preciznost odredjivanja numere na odredjenim delovima pesama za nizak nivo šuma

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Naziv pesme** | **Početak pesme** | **Refren** | **Strofa** | **Instrumental** |
| Compensating | 0.72 | 0.81 | 0.71 | 0.31 |
| Lost in Japan | 0.12 | 0.71 | 0.27 | 0.08 |
| Love and War | 0.1 | 0.65 | 0 | 1.0 |
| Other Beds | 0.43 | 0.63 | 0.47 | 0 |
| Swervin | 0.92 | 0.85 | 0.95 | 1.0 |
| What do you mean | 0.55 | 1.0 | 0.72 | 0 |
| California Gurls | 0 | 0.13 | 0.04 | 0 |
| Chuck Tailor | 0.29 | 0.18 | 0.14 | 0 |
| Crazy Train | 0.86 | 1.0 | 1.0 | 1.0 |
| How Deep | 0.72 | 0.83 | 0.67 | 0 |
| Starlight | 0.42 | 0.51 | 0.58 | 0 |
| Tokyo Drift | 0.95 | 0.96 | 1.0 | 1.0 |

Tabela 5.6 Preciznost odredjivanja numere na odredjenim delovima pesama za srednji nivo šuma

Sa druge strane kod pesama “*Love and War*” i “*California gurls*” primećujemo drastičan pad na početku pesama pri uvodjenju srednje količine šuma, 1% snage signala. Ovo nam ukazuje da sintisajzeri koji su dodatno reverbovani se najlakše utope u umerenoj količini šuma.

## 5.5 Vreme prepoznavanja

Brzina sistema predstavlja važan faktor za njegovu praktičnu primenu, naročito u realnom vremenu. Prosečno vreme potrebno za obradu audio uzorka dužine od oko 5 sekundi i pronalaženje najbliže pesme u bazi bilo je ispod **1 sekunde**. Ova brzina omogućava korisnicima brzo dobijanje rezultata, što je ključno za aplikacije zasnovane na interakciji u realnom vremenu, ali je u trenutnoj verziji aplikacije moguće samo sa manjim bazama podataka. Pri poredjenju sa većim bazama potrebno je koristi neku vrstu hashing algoritma.

## 5.6 Ograničenja sistema

Uprkos dobrim performansama u očekivanim uslovima sa nižom količinom šuma, sistem se suočava sa nekoliko izazova koji mogu negativno uticati na tačnost prepoznavanja:

* Pri srednjoj količini šuma preciznost već drastično opada. Potrebno je koristiti filtre ili autoenkodere koji bi taj šum otklonili.
* Sekvencijalni tip pretrage sistema je moguć samo na manjim bazama podataka. Za skaliranje baze potrebno je uvesti *Hashing* algoritam
* Najveću preciznost nam pruža korišćenje prozorske funkcije od 2 sekunde, ali umeren broj pesama sadrži segmente te dužine u kojima se nalazi minimalna količina sadržaja prema kojima ne možemo da je odredimo. Uvodjenjem varijabilnih prozorskih funkcija ili korišćenjem više kraćih otisaka bi produžilo vreme potrebno za odredjivanje numere, ali povećalo preciznost i pokrilo ove slučajeve.

Ova ograničenja ukazuju na prostor za dalja istraživanja i unapređenja, kao što su primena naprednih tehnika obrade signala, adaptivnih algoritama ili modela mašinskog učenja za povećanje otpornosti sistema na promene i smetnje, kao i nadogradnja i skalabilnost sistema. U prilogu se nalaze kodovi koji sadrže osnove ovih koncepata i omogućuju lakšu nadogradnju.

# Zaključak

Predmet rada predstavlja *Fingerprinting* metodologija i njena primena. *Fingerprinting* je proces u kome treba da iz muzičke numere izdvojimo samo najbitnije podatke koji su potrebni za njihovu identifikaciju. Ovi podaci nam takođe ukazuju na sličnosti između numera. Proces počinje kvantifikovanjem zvuka i diskretizacijom nivoa signala. Odatle signal pretvaramo u njegov frekvencijski oblik pomoću furijeove transformacije. Frekvencijski oblik signala predstavljamo spektrogramom iz kog možemo da izvučemo podatke o istaknutim vrhovima i kategorizujemo muzičke numere. Korišćenjem prozorske funkcije pesme delimo na manje segmente radi kreiranja više otisaka za različite delove pesme.

Cilj rada je softverska implementacija *Fingerprinting* metoda na gore naveden način kao i testiranje preciznosti ovakvog sistema na bazi podataka od dvanaest numera. Korišćen je *Python* programski jezik za realizaciju implementacije. Parametri po kojima je planirano variranje su:

· Segment pesme (početak, strofa, refren, instrumental)

· Dužina prozora za kreiranje otiska (1, 2, 4, 8 sekundi)

Pored ovoga normalizovani signali koje treba da prepoznamo sadrže različite količine šuma: 0, 0.001, 0.01 i 0.1 u odnosu na sve signale koji su skalirani na 1.

Implementacija je razvijena i testirana pokretanjem skripte sto puta za svaku numeru variranu po svakom od navedenih parametara. U implementaciji se koriste baze podataka koje su formirane korišćenjem prozorske funkcije i *Fingerprinting* metoda za svaku od pesama za dužine prozora od 1, 2, 4 i 8 sekundi. Poređenjem postojećih otisaka iz baze sa novodobijenim iz uzorka koji trenutno obrađujemo ističemo najsličniji i njega uzimamo za pretpostavku koja je numera u pitanju.

Rezultati pokazuju visok nivo preciznosti pri odsustvu šuma i pri prisustvu manje količine šuma, preko 98%. Povećanjem količine šuma dobijamo rezultate koji variraju u zavisnosti od pesme i njene jedinstvenosti. Nivoi šuma ispod 0.01, u našoj kategorizaciji srednji nivo šuma, a u realnom svetu proporcionalno ambijentalnom nivou tiše prostorije, nam daju odlične rezultate.

U većini slučajeva algoritam preciznije prepoznaje refren od strofe, a instrumentali i početak pesme variraju preciznost u zavisnosti od svoje dinamičnosti i broja prisutnih instrumenata. Dužina segmenta od 8 sekundi se pokazuje kao najbolja za najveći broj numera dok se dužina od 2 sekunde pokazala kao najpouzdanija u prosečnom slučaju prema preciznosti i varijabilnosti.

Rad je moguće proširiti primenom Hashing algoritama radi dodatne kompresije i brže pretrage na većim bazama podataka, kao i implementacijom neuralnih mreža za bolju kategorizaciju i još veću uštedu prostora uz rašumljvanje signala.

# Literatura

1. Dragana Sumarac Pavlovic, Miomir Mijic, “Elektroakustika”, Beograd 2017
2. Avery Li-Chun Wang, “An Industrial-Strength Audio Search Algorithm”, Shazam Entertainment, Ltd., 2003.
3. Avery Wang, “The Shazam music recognition service”, 2006.
4. Conor Murphy, “Music-Recognition-Tab”, 2022
5. Pradeep Kumar Khanal, Achyut Kayastha, Ashish Khatakho, “Music recognition using deep learning”, *Tribhuvan University Institute of engineering*
6. Blog Ane Harris na sajtu Medium

<https://medium.com/@anaharris/how-does-shazam-work-d38f74e41359>

1. Clanak Jovana Jovanovica na stranici Toptal

<https://www.toptal.com/algorithms/shazam-it-music-processing-fingerprinting-and-recognition>

[8] Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, Douglas Eck, [A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music](https://arxiv.org/abs/1803.05428)

[9] Colin Raffel. "Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching". [PhD Thesis, 2016](http://colinraffel.com/publications/thesis.pdf).

[10] Colin Raffel and Daniel P. W. Ellis. [Intuitive Analysis, Creation and Manipulation of MIDI Data with pretty\_midi](http://colinraffel.com/publications/ismir2014intuitive.pdf). In Proceedings of the 15th International Conference on Music Information Retrieval Late Breaking and Demo Papers, 2014.

# Spisak slika

[1] Slika 3.1. Analogni i digitalni signal

[2] Slika 3.2. Signal u vremenskom i frekvencijskom obliku

[3] Slika 3.3. Spektrogram

[4] Slika 3.4. 3D prikaz spektrograma

[5] Slika 3.5. Arhitektura autoenkodera

[6] Slika 5.1. Preciznost prepoznavanja pesama iz lokalnih fajlova

[7] Slika 5.2. Preciznost prepoznavanja blago zašumljenih pesama iz lokalnih fajlova

[8] Slika 5.3. Preciznost prepoznavanja umereno zašumljenih pesama iz lokalnih fajlova

[9] Slika 5.4. Preciznost prepoznavanja jako zašumljenih pesama iz lokalnih fajlova

[10] Slika 5.5. Preciznost prepoznavanja pesama iz lokalnih fajlova sa varirajućom količinom šuma

[11] Slika 5.6. Preciznost prepoznavanja pesama iz lokalnih fajlova sa varirajućom količinom šuma – dužina segmenta 1s

[12] Slika 5.7 Preciznost prepoznavanja pesama iz lokalnih fajlova sa varirajućom količinom šuma – dužina segmenta 2s

[13] Slika 5.8 Preciznost prepoznavanja pesama iz lokalnih fajlova sa varirajućom količinom šuma – dužina segmenta 4s

[14] Slika 5.9 Preciznost prepoznavanja pesama iz lokalnih fajlova sa varirajućom količinom šuma – dužina segmenta 8s

[15] Slika 5.10 Preciznost prepoznavanja pesama iz lokalnih fajlova sa varirajućom dužinom segmenta – 0.001 koeficijent zašumljenja

[16] Slika 5.11 Preciznost prepoznavanja pesama iz lokalnih fajlova sa varirajućom dužinom segmenta – 0.01 koeficijent zašumljenja

[17] Slika 5.12 Preciznost prepoznavanja pesme “*Compensating*” na različitim delovima numere

[18] Slika 5.13 Preciznost prepoznavanja pesme “*What do you mean*” na različitim delovima numere

[19] Slika 5.14 Preciznost prepoznavanja pesme “*Swervin*” na različitim delovima numere

[20] Slika 5.15 Preciznost prepoznavanja pesma na različitim delovima

# Spisak tabela

[1] Tabela 5.1. Preciznost prepoznavanja pesama iz lokalnih fajlova sa varirajućom količinom šuma

[2] Tabela 5.2 Preciznost prepoznavanja snimljenih pesama nakon otklanjanja šuma

[3] Tabela 5.3. Preciznost prepoznavanja pesama iz lokalnih fajlova sa varirajućom dužinom segmenta – srednji nivo šuma

[4] Tabela 5.4 Vremena trajanja segmenata pesama

[5] Tabela 5.5 Preciznost odredjivanja numere na odredjenim delovima pesama za nizak nivo šuma

[6] Tabela 5.6 Preciznost odredjivanja numere na odredjenim delovima pesama za srednji nivo šuma

# Prilog

## BuildDatabase

import os  
import json  
import numpy as np  
import librosa  
  
DB\_PATH = "songs/" # Folder containing reference songs  
DB\_JSON = "database2.json" # Change name  
  
  
def extractSlidingFingerprints(filePath, segmentDuration=2.0, overlap=0.5): # Change duration  
 y, sr = librosa.load(filePath, sr=None)  
 hop = int((1 - overlap) \* segmentDuration \* sr)  
 win = int(segmentDuration \* sr)  
 fingerprints = []  
 for start in range(0, len(y) - win, hop):  
 segment = y[start:start+win]  
 mfcc = librosa.feature.mfcc(y=segment, sr=sr, n\_mfcc=20)  
 fp = np.mean(mfcc, axis=1)  
 fp = normalizeVector(fp)  
 fingerprints.append(fp)  
 return fingerprints  
  
  
def extractFingerprint(filePath):  
 try:  
 y, sr = librosa.load(filePath, sr=None)  
 if y is None or sr is None:  
 print(f"Failed to load audio: {filePath}")  
 return None  
 mfcc = librosa.feature.mfcc(y=y, sr=sr, n\_mfcc=20)  
 if mfcc is None or mfcc.size == 0:  
 print(f"Failed to extract MFCC: {filePath}")  
 return None  
 return np.mean(mfcc, axis=1).tolist()  
 except Exception as e:  
 print(f"Error processing {filePath}: {e}")  
 return None  
  
  
def normalizeVector(vec):  
 vec = np.array(vec)  
 norm = np.linalg.norm(vec)  
 return (vec if norm == 0 else vec / norm).tolist()  
  
  
def buildDatabase():  
 database = {}  
 for file in os.listdir(DB\_PATH):  
 if file.endswith(".wav"):  
 path = os.path.join(DB\_PATH, file)  
 #fingerprint = extractFingerprint(path)  
 # if fingerprint is None:  
 # continue  
 #normalizedFingerprint = normalizeVector(fingerprint)  
 #database[file] = normalizedFingerprint  
 database[file] = extractSlidingFingerprints(path)  
 with open(DB\_JSON, "w") as f:  
 json.dump(database, f)  
 print(f"Saved {len(database)} songs into {DB\_JSON}.")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 buildDatabase()

## Echo

import os  
import json  
import sounddevice as sd  
import scipy.io.wavfile as wav  
import time  
import numpy as np  
import librosa  
# from keras.models import load\_model  
  
SR = 22050  
N\_MFCC = 20  
  
  
dbPath = "database2.json"  
  
  
# denoiser = load\_model("mfcc\_denoiser.h5", compile=False)  
# wave\_denoiser = load\_model("wave\_denoiser.h5", compile=False)  
  
def extractFingerprintDenoiseWave(path):

# Load full clip at 22.05 kHz  
 y, sr = librosa.load(path, sr=SR, mono=True)  
  
 # If clip is shorter than 17 seconds, pad it  
 target\_len = int(17 \* SR)  
 if len(y) < target\_len:  
 y = np.pad(y, (0, target\_len - len(y)))

# Denoise in chunks (model expects fixed input length)   
 chunk\_size = 44096  
 denoised = []  
 for i in range(0, len(y), chunk\_size):  
 chunk = y[i:i + chunk\_size]  
 if len(chunk) < chunk\_size:  
 chunk = np.pad(chunk, (0, chunk\_size - len(chunk)))  
 chunk\_in = np.expand\_dims(chunk, axis=(0, -1))  
 denoised\_chunk = wave\_denoiser.predict(chunk\_in, verbose=0).flatten()  
 denoised.append(denoised\_chunk)  
 denoised\_wave = np.concatenate(denoised)

# Pick random 2-second segment between 5–15 seconds  
 start\_sec = np.random.uniform(5, 15)  
 start\_sample = int(start\_sec \* sr)  
 end\_sample = start\_sample + int(2 \* sr)  
 segment = denoised\_wave[start\_sample:end\_sample]  
 if len(segment) < int(2 \* sr):  
 segment = np.pad(segment, (0, int(2 \* sr) - len(segment)))

# Extract MFCC fingerprint   
 mfcc = librosa.feature.mfcc(y=segment, sr=sr, n\_mfcc=N\_MFCC)  
 mfcc\_vec = np.mean(mfcc, axis=1)  
 return mfcc\_vec.tolist()  
  
  
def loadDatabase():  
 if not os.path.exists(dbPath):  
 print(f"Database file not found at {dbPath}")  
 database = {}  
 else:  
 with open(dbPath, "r") as f:  
 database = json.load(f)  
 print(f"Loaded {len(database)} songs from {dbPath}")  
 return database  
  
  
def extractFingerprint(filePath):  
 off=np.random.uniform(20, 100)  
 y, sr = librosa.load(filePath, sr=None, mono=True, offset=off, duration=2.0)  
 mfcc = librosa.feature.mfcc(y=y, sr=sr, n\_mfcc=20)  
 return np.mean(mfcc, axis=1).tolist()  
  
  
def normalizeVector(vec):  
 norm = np.linalg.norm(vec)  
 return vec if norm == 0 else vec / norm  
  
  
def recordAudio(filename="sample.wav", duration=3, fs=22050):  
 print("Recording (stay silent for the first second)...")  
 audio = sd.rec(int(duration \* fs), samplerate=fs, channels=1, dtype='float32')  
 sd.wait()  
 print("Recording finished.")  
  
 audio = np.squeeze(audio)  
 audio = audio[:fs] # First 1 second  
  
 # print("Reducing noise...")  
 # reduced = nr.reduce\_noise(y=audio, y\_noise=audio, sr=fs)  
  
 wav.write(filename, fs, (audio \* 32767).astype(np.int16))  
 print(f"Saved to {filename}")  
  
 time.sleep(2)  
 print("Playing back the recording...")  
 try:  
 sd.play(audio, fs)  
 sd.wait()  
 except Exception as e:  
 print("Playback failed:", e)  
  
  
def matchSong(samplePath, database):  
 sampleFp = normalizeVector(np.array(extractFingerprint(samplePath)))  
 bestMatch, minDistance = None, float("inf")  
  
 for song, fps in database.items():  
 for fp in fps:  
 fp = np.array(fp)  
 distance = np.linalg.norm(sampleFp - fp)  
 # if distance < 0.2:  
 # print(f"{song}: {distance}") # Debugging output  
 if distance < minDistance:  
 minDistance = distance  
 bestMatch = song  
  
 # if minDistance > 0.1: # Threshold to reject poor matches  
 # return "No good match found"  
 print(bestMatch)  
 return bestMatch  
  
def matchSongWithEmbeddings(sample\_embeddings, database):  
 bestMatch, minDistance = None, float("inf")  
 for song, fps in database.items(): # fps = list of segment embeddings  
 for fp in fps:  
 fp = np.array(fp)  
 for seg\_emb in sample\_embeddings: # compare each segment of the sample  
 distance = np.linalg.norm(seg\_emb - fp)  
 if distance < minDistance:  
 minDistance = distance  
 bestMatch = song  
 return bestMatch  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # recordAudio()  
 songs = ["Compensating", "LostInJapan", "LoveAndWar", "OtherBeds", "Swervin", "WhatDoYouMean",  
 "CaliforniaGurls", "ChuckTaylor", "CrazyTrain", "HowDeep", "Starlight", "TokyoDrift"]  
 # songs=["CaliforniaGurls"]  
 database = loadDatabase()  
 if database:  
 # match = matchSong("sample.wav", database)  
 hitCount=[]  
 repeat=100  
 for song in songs:  
 hitCountPer=0  
 for i in range(repeat):  
 match = matchSong("noise3/"+song+".wav", database)  
 if str(match) == song + ".wav":  
 hitCountPer+=1  
 hitCount.append(hitCountPer/repeat)  
 print(hitCount)  
 # print(f"Best match: {match}")

## addNoise

import numpy as np  
import librosa  
import soundfile as sf  
  
songs = ["Compensating", "LostInJapan", "LoveAndWar", "OtherBeds", "Swervin", "WhatDoYouMean", "CaliforniaGurls", "ChuckTaylor", "CrazyTrain", "HowDeep", "Starlight", "TokyoDrift"]  
  
for song in songs:  
 # Load wav file  
 y, sr = librosa.load("songs/"+song+".wav", sr=None) # y = audio time series, sr = sample rate  
  
 # Function to add Gaussian noise  
 def add\_noise(y, noise\_factor=0.005):  
 noise = np.random.randn(len(y)) # generate random noise  
 y\_noisy = y + noise\_factor \* noise  
 return np.clip(y\_noisy, -1.0, 1.0) # keep values in [-1, 1]  
  
 # Apply noise  
 y\_noisy = add\_noise(y, noise\_factor=0.1)  
  
 # Save to new file  
 sf.write("noise3/"+song+".wav", y\_noisy, sr)

## echoNeuro

import json  
import numpy as np  
import librosa  
from tensorflow.keras.models import load\_model  
from tensorflow.keras.utils import register\_keras\_serializable  
  
DB\_JSON = "database.json"  
MODEL\_PATH = "base\_network.keras"  
SEGMENT\_DURATION = 3.0  
OVERLAP = 0.5  
SR = 22050  
  
@register\_keras\_serializable()  
def euclidean\_distance(vectors):  
 x, y = vectors  
 sum\_square = tf.reduce\_sum(tf.square(x - y), axis=1, keepdims=True)  
 return tf.sqrt(tf.maximum(sum\_square, tf.keras.backend.epsilon()))  
  
def normalize\_vector(vec):  
 norm = np.linalg.norm(vec)  
 return vec if norm == 0 else vec / norm  
  
def extract\_segments(file\_path, segment\_duration=SEGMENT\_DURATION, overlap=OVERLAP):  
 y, sr = librosa.load(file\_path, sr=SR)  
 hop = int((1 - overlap) \* segment\_duration \* sr)  
 win = int(segment\_duration \* sr)  
 segments = []  
 for start in range(0, len(y) - win, hop):  
 seg = y[start:start + win]  
 mfcc = librosa.feature.mfcc(y=seg, sr=sr, n\_mfcc=20)  
 fp = np.mean(mfcc, axis=1)  
 fp = normalize\_vector(fp)  
 segments.append(fp)  
 return np.array(segments)  
  
def load\_database():  
 with open(DB\_JSON, "r") as f:  
 return json.load(f)  
  
def find\_best\_match(file\_path):  
 base\_network = load\_model(MODEL\_PATH, compile=False, custom\_objects={'euclidean\_distance': euclidean\_distance})  
 database = load\_database()  
  
 segments = extract\_segments(file\_path)  
 query\_embeddings = base\_network.predict(segments, verbose=0)  
  
 best\_match = None  
 best\_distance = float('inf')  
  
 for song, embeddings in database.items():  
 db\_embeddings = np.array(embeddings)  
 for q\_emb in query\_embeddings:  
 distances = np.linalg.norm(db\_embeddings - q\_emb, axis=1)  
 min\_dist = np.min(distances)  
 if min\_dist < best\_distance:  
 best\_distance = min\_dist  
 best\_match = song  
  
 print(f"Best match: {best\_match} with distance {best\_distance:.4f}")  
 return best\_match  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 import tensorflow as tf  
 # Example usage:  
 find\_best\_match("sample.wav")