

Prepoznavanje instrumenata u zvučnim  
datotekama  
Tehnička dokumentacija

Travanj 2023.

# Sadržaj

<b>1</b>	<b>Uvod</b>	<b>3</b>
1.1	Korišteni alati . . . . .	3
<b>2</b>	<b>Skup podataka</b>	<b>3</b>
2.1	Analiza podataka . . . . .	4
2.2	Isprobavanje jednostavnih modela . . . . .	4
<b>3</b>	<b>Virtualno okruženje</b>	<b>6</b>
<b>4</b>	<b>Model</b>	<b>6</b>
4.1	Augmentacije . . . . .	7
4.2	Transformacije . . . . .	7
4.3	Treniranje . . . . .	8
4.4	Preciznost . . . . .	9
4.5	Klizeći prozor . . . . .	9
4.6	Tehnički detalji . . . . .	9
<b>5</b>	<b>Aplikacija</b>	<b>10</b>
5.1	Api . . . . .	10
5.1.1	Instalacija . . . . .	10
5.1.2	Pokretanje . . . . .	11
5.1.3	Učitavanje modela . . . . .	13
5.1.4	Tehničke poteškoće . . . . .	14
5.2	Frontend . . . . .	14
5.2.1	Pokretanje . . . . .	14
5.2.2	Opis aplikacije . . . . .	15
5.2.3	Tehničke poteškoće . . . . .	16
<b>6</b>	<b>Izvori</b>	<b>18</b>

# 1 Uvod

U ovom radu rješavat ćemo problem prepoznavanja instrumenata u zvuku. Cilj će biti otkriti parametre koji utječu na konačnu preciznost (accuracy) modela. Sve ćemo testirati na IRMAS datasetu. Problem dijelimo na dva dijela: monofonija i polifonija. Monofonija je jednostavnija i cilj je prepoznati točno jedan instrument koji se pojavljuje na snimci. S druge strane, u polifoniji trebamo detektirati sve instrumente.

Kako su modeli dubokog učenja postali vrlo dobri, odlučili smo koristiti Python biblioteku *fastai* za izradu modela. Kao ulaz u model, pjesme smo prvo transformirali u spektrograme. Za treniranje smo koristili razne augmentacije zvuka: pomicanje zvuka u vremenu, brisanje segmenata zvuka, promjena visine tona i dodavanje šuma. Također smo isprobali različite arhitekture poput *resnet18*, *alexnet* i *vgg16\_bn* te se na kraju odlučili za korištenje *resnet18*.

Osim toga, napravili smo web stranicu te api kojima možemo naš model koristiti u stvarnom svijetu.

U ovoj dokumentaciji fokusirat ćemo se na tehničke detalje rješavanja zadatka. Kako smo točno izradili dataset i kako smo izradili model. Također, navodimo kako se pojedini programi koriste. Za instalaciju potrebnih biblioteka i pokretanje programa napisali smo vrlo detaljne upute u README datotekama.

Izrada modela najlakše će se shvatiti uz pokretanje Jupyter bilježnice `main.ipynb`. Tamo se nalazi kod za augmentacije, transformacije u spektrogram, učitavanje datasea i izradu modela.

## 1.1 Korišteni alati

Koristili smo sljedeće alate:

- GitHub
- Python
  - *fastai*
  - *numpy*
  - Django - za API
- Cuda - za korištenje grafičke kartice prilikom treniranja modela
- React - za frontend aplikaciju

U repozitoriju se nalaze upute za namještanje većine korištenih alata.

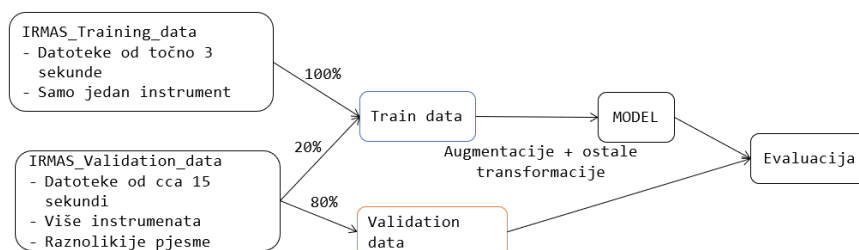
## 2 Skup podataka

Skup podataka s kojim radimo je IRMAS dataset[1]

Skup je podijeljen u dvije skupine. Prva je *IRMAS\_training* koja se sastoji od isječaka od tri sekunde te je samo jedan instrument dominantan i označen.

Druga skupina je *IRMAS\_validation* koja se sastoji od isječaka u trajanju od 5 do 20 sekundi. Također, u njima se pojavljuje od jedan do pet instrumenata, umjesto samo jednog. Isječci su malo raznovrsniji i više nalikuju onima koji bi se mogli pojaviti u stvarnom svijetu.

Mi ćemo dodatno transformirati skup podataka na sljedeći način. Cijeli *IRMAS\_training* i dio *IRMAS\_validation* (oko 20%) uzet ćemo kao skup za treniranje, a ostatak *IRMAS\_validation* će nam biti skup za testiranje.



Slika 1: Skup podataka

Ovo radimo zbog toga što je *IRMAS\_validation* set raznovrsniji te se isplati trenirati model i na njemu. Bitno za napomenuti je da 20% podataka nije slučajno odabrano, već je uzeto prvih 20% podataka. Kako se u ovom skupu može pronaći više isječaka iste pjesme (čak i više od 10), ovime neće dio njih završiti u skupu za treniranje, a dio u skupu za testiranje, već će svi klipovi iste pjesme biti u istome skupu.

## 2.1 Analiza podataka

U ovom poglavlju objasniti će se postupak analize podataka. Ovom i sljedećem poglavlju, kao i poglavljima 2. i 3. u projektnoj dokumentaciji pridružena je *Jupyter bilježnica* u *notebooks/dataAnalysis.ipynb*. Ovdje se, kao i kasnije u projektu (stvaranje modela), koristimo Python bibliotekom *librosa* za učitavanje audio zapisa i njihovo baratanje. *Librosa* ima korisne funkcije kojima se mogu vaditi informacije o nekoj zvučnoj snimci kao što su spektrogrami i spektralna središta. Neke od tih značajki smo precrtali u grafove koristeći *Matplotlib* biblioteku.

## 2.2 Isprobavanje jednostavnih modela

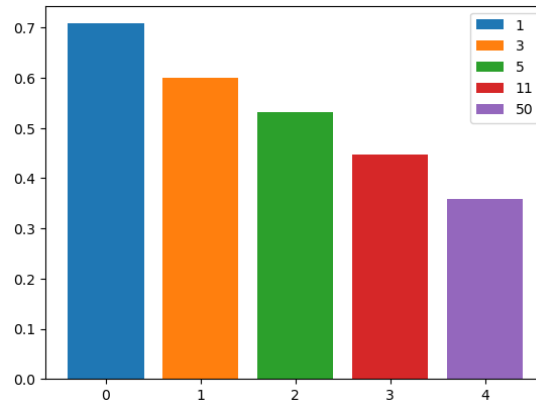
Nakon osnovne analize podataka isprobali smo nekoliko jednostavnih modela.

Za skup podataka kojim se ovi modeli služe koristili smo manji podskup *IRMAS\_training* skupa. Specifično, koristili smo prvih 100 snimka za svaki instrument. Ovi modeli isprobani su samo za rješavanje problema monofonije jer nisu dovoljno dobri za polifoniju, ali svejedno nam daju osjećaj o tome kakav bi pristup trebali koristiti.

Koristili smo *scikit-learn* biblioteku za funkciju koja razdvaja skup podataka na dva dijela - skup za treniranje i skup za testiranje.

Modeli koje smo isprobali:

1. **Uprosječni model** Za ovaj model napravili smo klasu koja ima *fit* metodu koja prima X i y podatke pomoću kojih model podesi težine. Težine se podešavaju tako da uprosječi pripadajuće X vrijednosti za svaki različit y. U slučaju pjesama kao ulaznih podataka, uprosječi sve zvučne zapise (ili čak njihove spektrograme) za svaki instrument.
2. **K-najbližih susjeda** Ova metoda srećom već je implementirana u *scikit-learn* biblioteci kao *KNeighborsClassifier* pa ju nismo morali sami implementirati. Slika 2 prikazuje graf točnosti modela u ovisnosti o različitom parametru K. Zanimljivo je primijetiti da točnost modela opada s većim K-ovima što nije inače slučaj.



Slika 2: Dataset

3. **Multivarijantna logistička regresija** Kao i za prošlu metodu, *scikit-learn* nam opet spašava dan, pa je implementirao logističku regresiju umjesto nas. Koristimo već gotov *liblinear* algoritam kao metodu optimizacije jer radi brže za manje skupove podataka nego unaprijed zadana metoda. Multivarijantna logistička regresija koristi SOFTMAX aktivacijsku funkciju.

U slici 3 smo priredili točnost modela u ovisnosti o korištenom modelu i transformaciji ulaznih podataka.

	Average model	K-neighbours	Logistic regression
X original	0.13	0.17	0.17
X fft	0.28	0.63	0.53
X spec	0.34	0.71	0.69
X centroid	0.19	0.2	0.15
X bandwidth	0.22	0.27	0.16

Slika 3: Različit parametar K i točnost predviđanja

Najbolje predviđa metoda K-najbližih susjeda u kombinaciji sa spektrogramima.

### 3 Virtualno okruženje

Za lakše pokretanje na različitim računalima napravili smo i dva Python virtualna okruženja: jedno za općenito pokretanje Jupyter bilježnica i poslužitelja, a drugo okruženje koristi grafičku karticu kako bi ubrzalo treniranje modela. U ovom poglavlju objasniti ćemo ukratko proces instalacije prvog navedenog okruženja, a za drugo okruženje je više objašnjeno u GPU\_SETUP.md dokumentu u repozitoriju.

U environments direktoriju nalaze se dva .yaml dokumenta. Pozicionirajte se u taj direktorij i unutar naredbenog retka pokrenite:

```
conda env create -f environment.yaml
```

Trebalo bi se stvoriti novo Conda okruženje zvano lumen, to možete provjeriti pokretanjem:

```
conda env list
```

Ako je lumen naveden među okruženjima možete pokrenuti:

```
conda activate lumen
```

Time će se aktivirati okruženje koje ima podršku za jupyter bilježnice i kod poslužitelja.

Ovo okruženje zna usporiti kod poslužitelja (api), iako će i dalje raditi. Otkrili smo da na nekim drugim okruženjima poslužitelj radi brže, ali ovo okruženje smo izveli jer sadrži najmanji broj biblioteka potrebnih za rad aplikacija i mislimo da je najprikladnije.

### 4 Model

Za model smo odabrali raditi sa Resnet18 arhitekturom koristeći fastai[2] biblioteku u Pythonu.

## 4.1 Augmentacije

Mnoge augmentacije koje smo primjenjivali zapravo su inspirirane nekim augmentacijama za problem klasifikacije slika: pomak slike  $\rightarrow$  translacija, šum na slici  $\rightarrow$  šum, crop  $\rightarrow$  tišina. Osim toga, čine se vrlo prirodne za analizu zvuka, a njihovu učinkovitost mjerit ćemo kasnije.

1. **Translacija** Translacija pomiče početak datoteke na malo kasniji trenutak, a dijelove pjesme s kraja pomiče na početak. Možemo ju zamisliti kao rotaciju bitova u binarnom zapisu. Možda je najbitnija od svih augmentacija jer uopće ne uništava podatke, a opet pomaže modelu da se nauči prepoznavati značajke u svakom trenutku zvučnog signala.

2. **Tišina**

Dijelove datoteke ćemo maknuti tako da model nauči prepoznavati instrumente i iz kraćih isječaka zvuka.

3. **Dodavanje šuma**

Na pjesmu ćemo dodati nasumične brojeve prigodne amplitude tako da se stvori otpornost na greške u podacima.

4. **Mijenjanje visine zvuka**

Koristeći alate iz biblioteke *librosa*, pomaknut ćemo visinu zvuka datoteke za nasumičan broj.

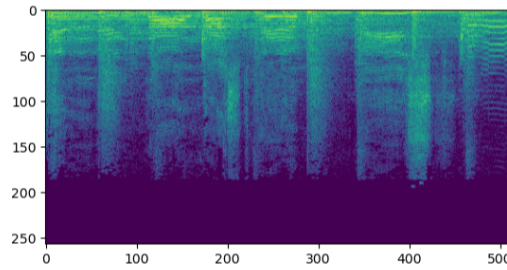
5. **Spajanje više datoteka**

Ova augmentacija vrlo je bitna za poboljšanje prepoznavanja više instrumenata u datotekama. U skupu podataka za treniranje nalaze se samo pjesme sa jednim instrumentom, ali možemo zbrojiti više nasumičnih datoteka i tako dobiti polifoniju.

Sve augmentacije događaju se s određenom vjerojatnošću (uglavnom oko 50%), osim prve dvije koje se događaju uvijek. Za augmentacije se biraju nasumični parametri, poput jačine šuma, promjena visine tona itd.

## 4.2 Transformacije

Nakon odrađenih augmentacija imamo zvučni signal koji je jednodimenzionalan - vektor. Njega zatim pretvaramo u spektrogram što je dvodimenzionalan podatak (matrica). Spektrogram prikazuje jačinu frekvencija koje sviraju u svakom trenutku.



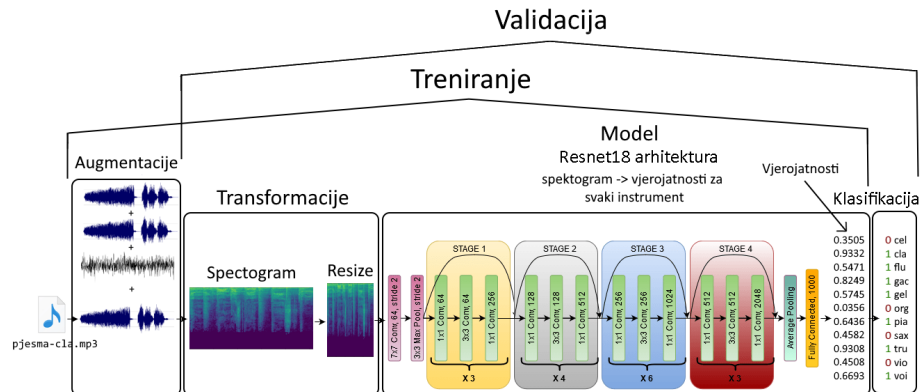
Slika 4: Spektrogram

Nakon toga smanjujemo dimenzije matrice, kako bismo ubrzali proces treniranja. Preciznije, smanjili smo na dimenzije ( $y=256$ ,  $x=156$ ). Ovo smanjivanje sačuva informacije većine frekvencija iz spektrograma, ali stegne sliku u vremenu. Time se najbitnije informacije o datoteci ne gube, ali se ubrzava model. Dobivena matrica bit će konačni ulazni podatak za konvolucijsku neuronsku mrežu.

### 4.3 Treniranje

Treniramo model samo na skupu podataka za treniranje uz upotrebu augmentacija. Pjesme koje traju tri sekunde učitamo cijele, a one koje traju dulje (*IRMAS\_Validation*) odrežemo u nasumično odabrane tri sekunde.

S druge strane, prilikom validacije uvijek biramo središnji isječak pjesme. Dodatno, sve vjerojatnosti koje model izračuna za svaki instrument *odrežemo* na 0.6. To znači da ako je pretpostavio da je vjerojatnost iznad 0.6, to pretvaramo u 1, a inače u 0.



Slika 5: Cijeli model



## 4.4 Preciznost

Preciznost smo mjerili kao Hamming score. Gledali smo XOR predviđanja modela i ispravnih oznaka. XOR će davati 1 samo ako se predikcije razlikuju. Dobivene vrijednosti prvo zbrojimo po svim datotekama, zatim po instrumentima te ih podijelimo sa brojem instrumenata i brojem instrumenata. Tako dobivamo preciznost.

Ovakav način računanja preciznosti ima i svoje mane. Primjerice model koji uvijek vraća da se ne pojavljuje niti jedan instrument imat će preciznost od 84%, zato što se po datoteci u prosjeku nalazi 1.73 instrumenta. Stoga prilikom analiziranja rješenja računali smo lažno pozitivne, lažno negativne i promatrali ih zasebno po instrumentima.

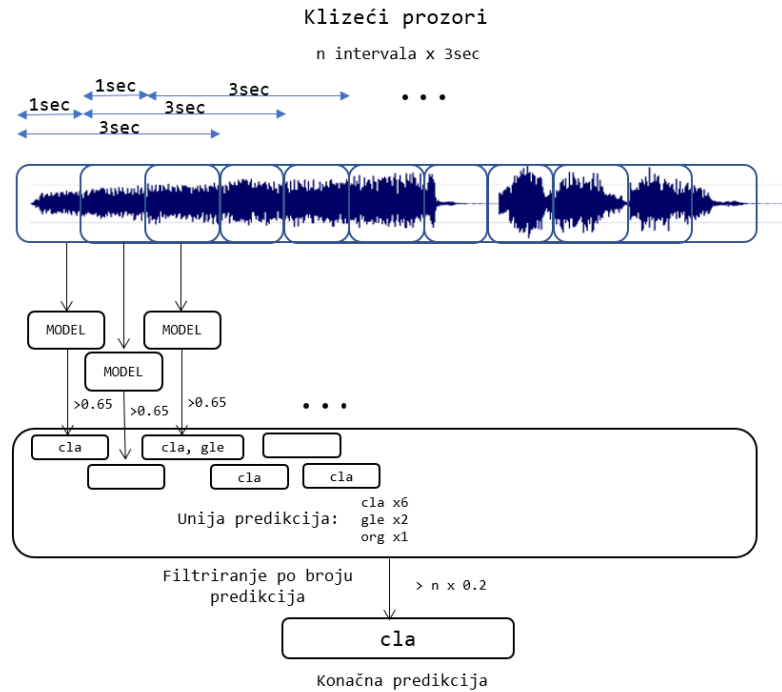
## 4.5 Klizeći prozor

Jednom kada imamo zadovoljavajući model za datoteku od tri sekunde, trebamo način da možemo klasificirati i pjesme dulje od tri sekunde. Ovo smo radili na način da smo uzeli isječke duljine tri sekunde s razmakom od 1 sekunde te smo svaki zasebno evaluirali. Ako se neki instrument pojavio u više od 26% isječaka, dodajemo ga u konačna predviđanja.

Dodatno, koristili smo prag od 0.56 za evaluaciju (veličina na kojoj se predikcije *odsijećaju*). Na donjoj slici prikazan je model kojem je taj prag jednak 0.65, a prag za broj pojavljivanja 20%. Postupak optimizacije pragova opisan je u projektnoj dokumentaciji.

## 4.6 Tehnički detalji

Cijeli gornji postupak napravili smo u jupyter bilježnici `main.ipynb`. Kako pokrenuti bilježnicu detaljno je opisano u README datoteci u repozitoriju. Također, mnoge bitne napomene vezane uz virtualna okruženja koja smo koristili nalaze se u ostatku ove dokumentacije i u README datotekama.



Slika 6: Klizeći prozor

Ovaj način evaluacije pjesme dobiva čak 1% veću preciznost u usporedbi s biranjem samo središnjeg isječka. Jedan posto je čak vrlo značajno u ovom načinu računanja preciznosti. Najbolji rezultat koji dobivamo ima preciznost od čak 90.620%.

## 5 Aplikacija

Ovdje ćemo malo detaljnije proći proces pokretanja Api-a i Frontend-a. Također ćemo navesti neke probleme s kojima smo se susreli i kratki opis aplikacije.

### 5.1 Api

Api smo pisali u Django framework-u jer se činio najprikladniji s obzirom na to da smo model razvijali u Python okruženju, a Django je Python biblioteka upravo za razvijanje rest poslužitelja.

#### 5.1.1 Instalacija

Za pokretanje poslužitelja nekoliko stvari treba biti zadovoljeno. (Ako ste ispravno namjestili virtualno okruženje, ovaj dio možete preskočiti)

1. **Django:** Ovo je osnovni alat za primanje i slanje Http poruka. Može se instalirati sljedećom naredbom:

```
pip install django
```

Osim njega, postoji jedna dodatna biblioteka koju koristimo da uredimo postavke oko CORS-a:

```
pip install django-cors-headers
```

2. **Librosa:** Osnovni alat za baratanje sa audio zapisima, koristili smo ga i u bilježnicama za analizu podataka i razvijanje modela:

```
pip install librosa
```

3. **Fastai:** Ova biblioteka znatno olakšava razvijanje modela, a potrebna je i za njegovo učitavanje. Fastai se koristi torch biblioteku pa je nju posebno instalirati prije:

```
pip install torch  
pip install fastai
```

4. **Jupyter bilježnica:** Većina koda za učitavanje modela napravljena je u jednoj Jupyter bilježnici. Kako bismo izbjegli ponovno pisanje istog koda i u poslužitelju, koristili smo biblioteku koja može pretvoriti Jupyter bilježnicu u običan Python kod.

```
pip install nbformat  
pip install nbconvert
```

### 5.1.2 Pokretanje

Nakon instalacije potrebno je pozicionirati se u `"/lumenback"` i pokrenuti:

```
\$ python manage.py runserver
```

Ako je sve instalirano kako treba, izlaz bi trebao izgledati slično kao na slici 7. Prije toga je uobičajeno da se prikažu neka upozorenja, to se može zanemariti.

```

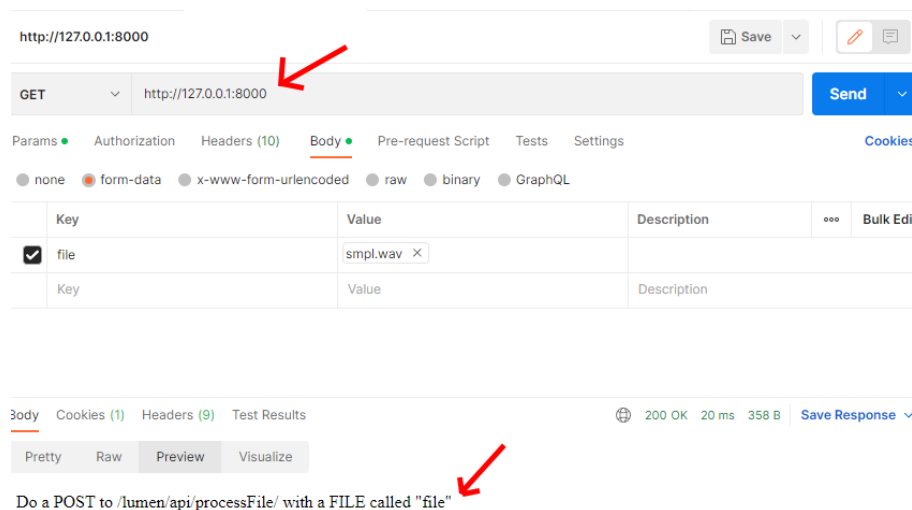
-----
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
May 08, 2023 - 12:20:25
Django version 4.2.1, using settings 'lumenweb.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

```

Slika 7: Izlaz nakon pokretanja programa

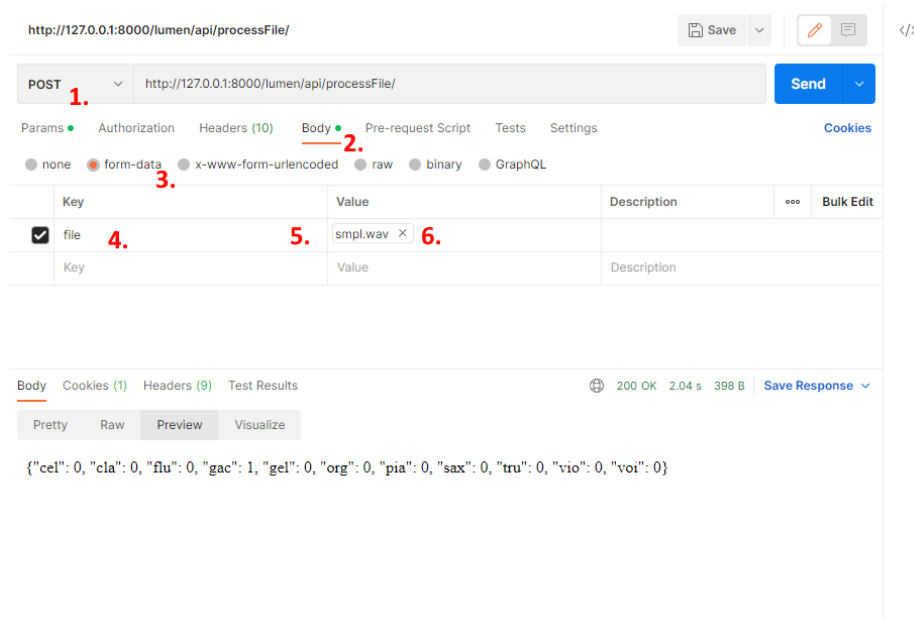
Tu je također napisana adresa na kojoj se stranica vrti, možemo ju otvoriti. Slika 8 prikazuje ispis u Postman agentu koji je koristan alat za slanje i primanje Http requestova i besplatan je (<https://www.postman.com/downloads/>). Sličan ispis bio bi i da smo adresu otvorili u web-pregledniku. Nebitno je što šaljemo u zaglavlju, bitno je da se koristi GET metoda.



Slika 8: Index

Ovo predstavlja popis endpointova koje stranica nudi (samo jedan).

Ako pošaljemo zvučnu datoteku na adresu `"/lumen/api/processFile/"`, dobit ćemo odgovor u obliku JSON-a sa popisom instrumenata na datoteci koju smo poslali (Slika 9). U nastavku je opisan proces slanja datoteke koristeći postman agent (pratite sliku 9) .



Slika 9: `/lumen/api/processFile/`

Promjenimo GET u POST u gornjem lijevom kutu prozora (1.) i promjenimo adresu slanja na `/lumen/api/processFile/`. Kliknemo na Body izbornik ispod adrese (2.), odaberemo form-data izbornik (3.). Napišemo "file" kao vrijednost ključa (4.). Promjenimo tip podataka iz teksta u datoteku (5.). Odaberemo željenu datoteku za slanje (6.). Pošaljemo ovu Http poruku na zadanu adresu klikom na plavi gumb SEND u gornjem desnom kutu.

Vidimo odgovor u obliku JSON-a sa popisom aktivnih instrumenta na poslanoj snimci.

### 5.1.3 Učitavanje modela

Učitavanje modela u Django backendu bilo je svakako zahtjevno. Glavni problem je što smo model izrađivali u Jupyter bilježnici, a backend je morao biti niz običnih Python datoteka. Zato smo odlučili koristiti biblioteku *nbconvert* koja nam pruža mogućnost pretvaranja bilježnica u obične Python datoteke. Tako se prilikom pokretanja Django servera, prvo `main.ipynb` prevodi u python te kasnije njezine funkcije koristimo po potrebi. Za naglasiti je da se ne pretvara cijela bilježnica, već samo dio. Neke ćelije u bilježnici označene su sa oznakom *remove*. One su uglavnom tekstovi i funkcije koje koristimo samo za prikaz u bilježnici te se neće prevesti.

**Prva napomena:** U našem direktoriju nalazi se direktorij `models`. Tamo se moraju nalaziti težine našeg modela `MLBLCLA2_model.pth`. Bez njih program neće moći učitati model i neće se moći pokrenuti. Težine će biti poslone u rješenju, ali se ne nalaze na našem GitHub repozitoriju zbog prevelike veličine.

Upravo zbog toga nismo mogli staviti aplikaciju na neki javni poslužitelj, jer većina samo preuzima kod sa GitHuba.

**Druga napomena:** U rješenju ćemo poslati *dummy* dataset. On je daleko manji i sadrži samo jednu datoteku po folderu dataseta. To je kako bi se struktura direktorija sadržala i na GitHubu. Takvog ćemo ga i poslati jer je potreban za učitavanje modela, a slanje cijelog bi drastično povećalo rješenje. Upute za kopiranje cijelog dataseta na način kako smo ga mi podijelili, nalazi se u README datoteci u datasetu. Isti postupak je opisan i u projektnoj, ali ovdje je detaljniji. Program bi se trebao moći pokrenuti samo koristeći *dummy* dataset, ali tada bilježnica neće ispravno prikazivati preciznost i rezultati će biti drugačiji nego u projektnoj dokumentaciji.

#### 5.1.4 Tehničke poteškoće

Django je dosta rigorozan oko sigurnosti, pa smo imali puno tehničkih poteškoća sa POST metodama. Trebali smo isključiti neke opcije oko CORS-a (Cross-Origin Resource Sharing) da bi bilo tko mogao slati POST metodu na poslužitelj.

Ako želimo napraviti *whitelist*-u sa popisom klijenta koji se smiju koristiti api-jem, django ima opcije za to. Ovu opciju bismo koristili kada bi poslužitelj izišao u produkciju.

Django zna biti spor u kombinaciji sa određenim virtualnim okruženjima. Rješenje nismo našli, ali ovo je problem optimizacije, a ne greška.

## 5.2 Frontend

Ovdje ćemo objasniti postupak pokretanja klijentske strane aplikacije, kratki opis aplikacije i probleme s kojima smo se susreli.

### 5.2.1 Pokretanje

Aplikacija je izrađena u React frameworku i koristi *Vite* za stvaranje projekta. Popis svih dependency-a koje smo koristili nalazi se u package.json. Potrebno je imati Node.js instaliran kako bi se aplikacija pokrenula (<https://nodejs.org/en/download>). Sa Node.js-om trebao bi stići i Node Package Manager (NPM), pomoću kojeg se instaliravaju dodatni dependency-ji. Slika 10 prikazuje verzije Node-a i NPM-a na kojima se razvijala aplikacija. Preporučeno je da se program vrti na ovim verzijama, iako nebi trebalo biti razloga da ne radi na drugim verzijama.

```
(base) E:\source\Lumen\lumenback>npm --version
8.1.0

(base) E:\source\Lumen\lumenback>Node --version
v16.13.0
```

Slika 10: Node and npm versions

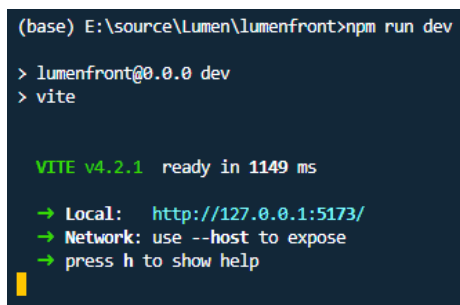
Pozicionirajte se u root klijentske aplikacije ("/lumenfront"). Pokrenite naredbu:

```
\$ npm install
```

Time bi se trebali instalirati svi potrebni moduli za rad aplikacije zapisani u package.json-u. Nakon toga potrebno ju je i pokrenuti:

```
\$ npm run dev
```

Pokazat će se adresa za aplikaciju koju možete otvoriti u web-pregledniku (mi smo koristili Chrome). Na slici 11 se prikazuje kako bi trebao izgledati uobičajeni ispis u terminalu nakon pokretanja.



```
(base) E:\source\Lumen\lumenfront>npm run dev
> lumenfront@0.0.0 dev
> vite

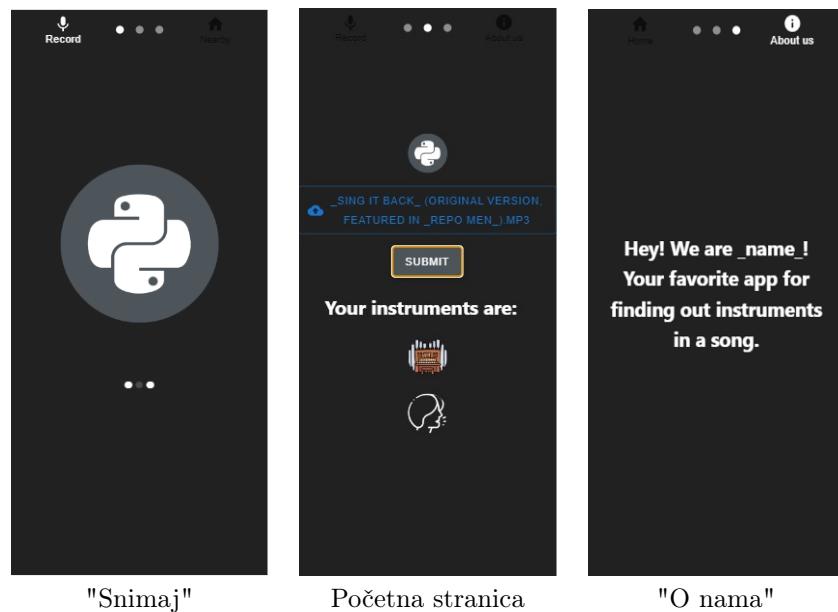
VITE v4.2.1 ready in 1149 ms
  → Local:   http://127.0.0.1:5173/
  → Network: use --host to expose
  → press h to show help
```

Slika 11

### 5.2.2 Opis aplikacije

Aplikacija ima tri dijela:

1. Početna stranica
2. "Snimaj"
3. "O nama"



Slika 12

U zaglavlju se nalazi izbornik s kojim se možemo pomicati lijevo i desno na različite stranice, ovisno o trenutnoj.

Početna stranica nudi opciju učitavanja datoteke sa uređaja, koja se potom šalje poslužitelju. Odgovor od poslužitelja je u obliku JSON-a i prikazuje instrumente prisutne u datoteci. Instrumenti se potom prikazuju u obliku ikonica kao na slici 12.

Stranica "Snimaj" ima gumb za snimanje s kojim se snima zvuk korisnika [gumb pulsira dok je aktivan :) ] i snimku potom pošalje poslužitelju. Odgovor se prikazuje isto kao što bi se prikazao i na prvoj stranici (ispod gumba). Ako korisnikov mikrofoni nije dostupan, na stranici se prikazuje sukladna poruka.

Dok se pjesma šalje sa glavne ili "Snimaj" stranice, tri točkice cirkuliraju na ekranu da indiciraju čekanje na odgovor.

### 5.2.3 Tehničke poteškoće

Nedavno su ukinuli većinu podrške za snimanje gotovo svih tipova zvučnih datoteka i većina web-preglednika trenutno podržava samo .webm. Ovo je problem jer Python biblioteka s kojom radimo ne podržava direktno taj tip datoteke. Zato smo imali problema sa dijelom aplikacije koja snima zvuk. Zasad klijent snimi .webm, dopusti korisniku da preuzme datoteku i istu pošalje poslužitelju koji nažalost još ne prepoznaje taj tip datoteke. Možete pretvoriti preuzetu .webm datoteku u .wav pomoću nekog online converter-a ili pomoću ffmpeg-a (<https://ffmpeg.org/>), novu datoteku poslati poslužitelju na glavnoj stranici aplikacije i pogledati snimljene instrumente.



Ovaj dio programa nije daleko od toga da radi (samo nedostaje pretvornik iz .webm u .wav na klijentskoj ili poslužiteljskoj strani).

## 6 Izvori

- [1] J. J. Bosch, J. Janer, F. Fuhrmann, and P. Herrera. “IRMAS: a dataset for instrument recognition in musical audio signals”. In: (2012).
- [2] J. Howard. *Practical Deep Learning for Coders with fastai and PyTorch*.