Data Capybara

Lumen Data Science 2023.
Tehnička dokumentacija

9.5.2023.

Sadržaj

1. Uvod

U današnje vrijeme, razvoj tehnologije pronašao je mnogobrojne primjene u području glazbe i zvuka. Jedna od tih primjena je stvaranje modela za klasifikaciju instrumenata. Takav model može biti koristan u mnogim situacijama, kao što je automatsko prepoznavanje glazbenih instrumenata u snimljenim pjesmama, što može pomoći u organiziranju glazbenih baza podataka ili pretraživanju pjesama prema određenim instrumentima. Također, model za klasifikaciju instrumenata može biti koristan u stvaranju novih glazbenih djela, transkripciji glazbe za potrebe podučavanja, kao i u raznim drugim primjenama u glazbenoj industriji. Cilj ovog projekta je razviti model za klasifikaciju instrumenata koji će biti precizan i učinkovit u prepoznavanju različitih glazbenih instrumenata u audio zapisima.

2. Upute za pokretanje aplikacije

U ovom poglavlju navedene su upute za pokretanje naše aplikacije na Windows ili MacOS/Linux računalima. Kako biste mogli uspješno pokrenuti aplikaciju na vlastitom računalu morate instalirati alate Docker i Docker Compose, upute za instalaciju možete naći na sljedećem linku: https://docs.docker.com/compose/install/.

Nakon uspješne instalacije možete nastaviti po sljedećim uputama:

1. Pozicionirajte se u direktorij *docker/*

```
$ cd LumenDS/docker
```

2. Pokrenite skriptu *serve.sh*

```
$ bash serve.sh
```

Napomena: Ako ste na Windowsu upišite komandu: "docker-compose up"

3. Nakon što se Docker kontejneri pokrenu dobiti ćete sljedeće linkove u terminalu:

```
Starting docker_backend_1 ... done
Starting docker_frontend_1 ... done
Attaching to docker_backend_1, docker_frontend_1
backend_1    | INFO: Started server process [1]
backend_1    | INFO: Waiting for application startup.
backend_1    | INFO: Application startup complete.
backend_1    | INFO: Uvicorn running on http://0.0.0.0:8000
frontend_1   |
frontend_1   |
frontend_1   |
frontend_1   |   You can now view your Streamlit app in your browser.
frontend_1   |
frontend_1   |   Network URL: http://172.20.0.3:8501
frontend_1   |   External URL: http://92.242.240.96:8501
frontend_1   |
```

Stisnite (CTRL + lijevi klik) na link pod nazivom *"Network URL"* (u ovom slučaju: "http://172.20.0.3:8501") i uživajte u korištenju aplikacije!

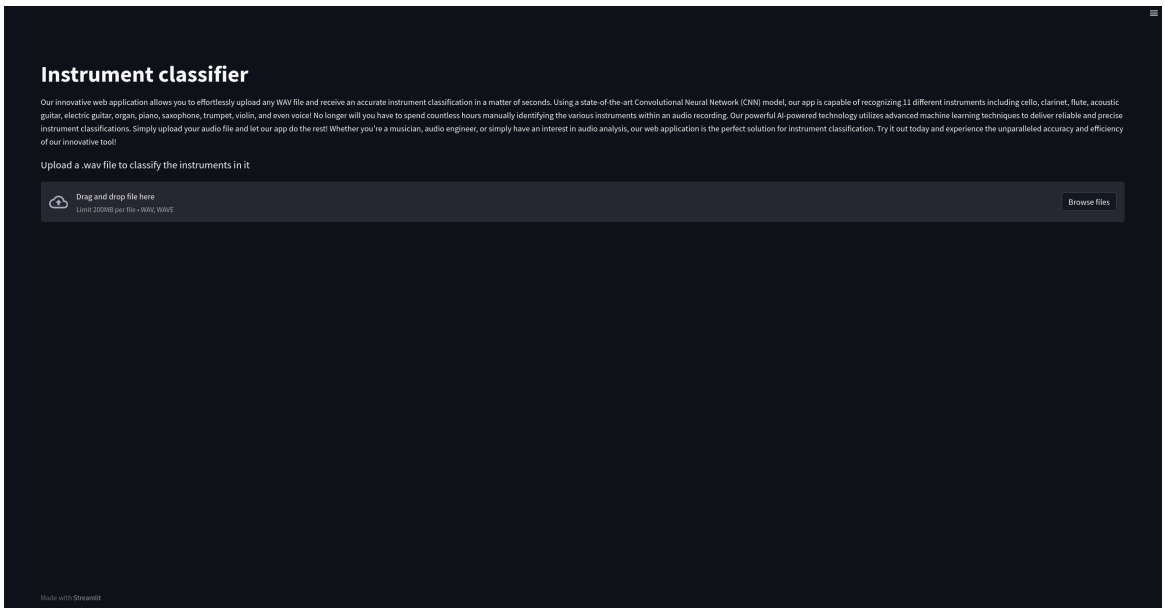
Napomena: Aplikaciju možete zaustaviti klasičnim signalom (CTRL + C) u terminalu ili gašenjem terminala.

3. Aplikacija

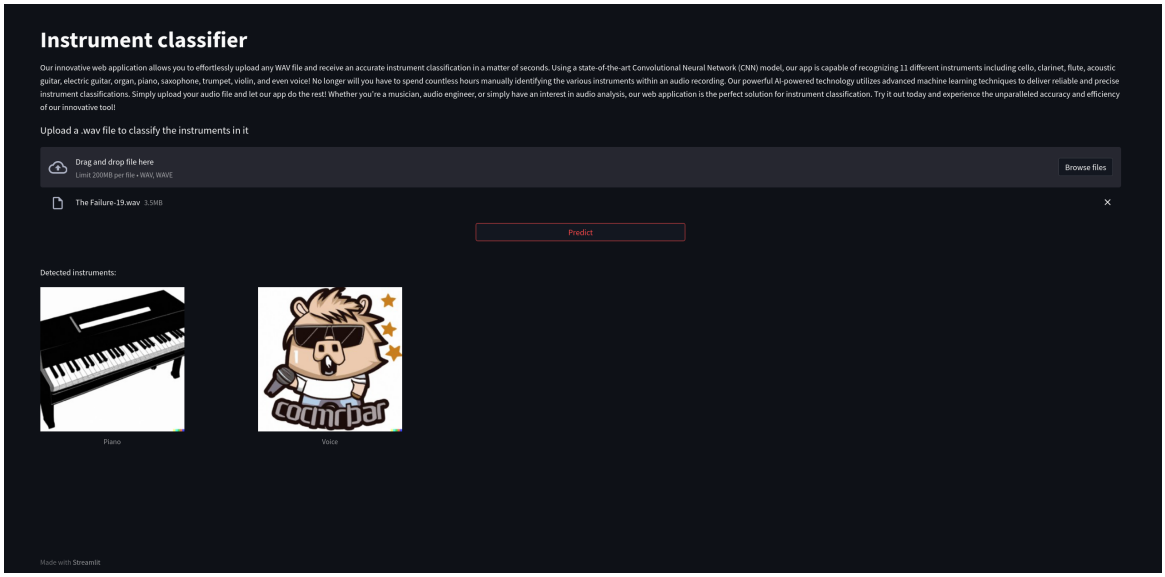
Naša web aplikacija može klasificirati instrumente iz audio datoteka. Koristi konvolucijsku neuronsku mrežu (CNN) za prepoznavanje 11 različitih instrumenata, uključujući violončelo, klarinet, flautu, akustičnu gitaru, električnu gitaru, orgulje, klavir, saksofon, trubu, violinu i glas. Aplikacija pruža jednostavno korisničko sučelje za prijenos .wav audio datoteke i predviđanje instrumenata prisutnih u njoj.

3.1 Frontend

Za razvoj frontenda naše web aplikacije, odlučili smo koristiti Streamlit - Python biblioteku koja olakšava kreiranje interaktivnih web aplikacija. Uz puno gotovih komponenata Streamlit ima sve što je potrebno za brzo i jednostavno sučelje bilo kakvog data science projekta (više o Streamlit-u na: https://docs.streamlit.io/). Također smo koristili alat DALL-E 2 za generiranje slika instrumenata te našeg loga (alat DALL-E 2: https://openai.com/product/dall-e-2). Izgled aplikacije možete vidjeti na sljedećim slikama.



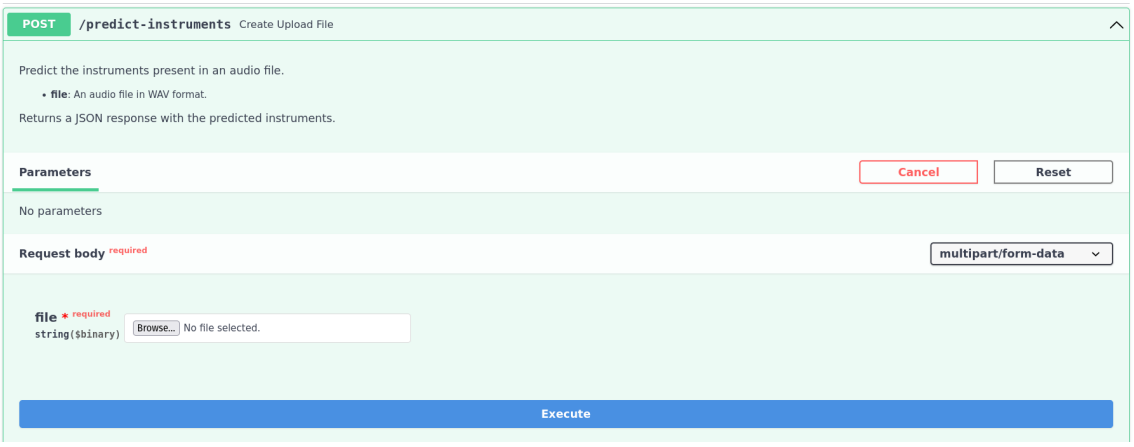
Slika 3.1: Izgled aplikacije



Slika 3.2: Primjer predviđanja

3.2 Backend

Za backend naše web aplikacije koristili smo FastAPI - moderni web framework za razvoj web aplikacija, API-ja i mikroservisa u programskom jeziku Python. Odabrali smo Fast API zbog jednostavnosti korištenja, odličnih performansi te mogućnosti generiranja dokumentacije izravno iz koda pomoću alata Swagger UI. Dokumentacija se generira na endpointu GET */docs* gdje se također može interaktivno testirati sam REST API (više o FastAPI-u na: https://fastapi.tiangolo.com/). Razvijeni REST API sastoji se od dva endpointa: POST */upload-file*, i POST */upload-files* gdje se šalje audio datoteka ili više njih, a vraća se JSON s predikcijama svih klasa instrumenata.



Slika 3.3: Primjer testiranja endpointa

3.3 Docker

Za kontejnerizaciju našeg FastAPI REST API-ja i Streamlit aplikacije, odlučili smo koristiti Docker i Docker Compose. Docker nam omogućava pakiranje i izvođenje našeg koda u izoliranom okruženju, a Docker Compose nam olakšava upravljanje višestrukim kontejnerima (više o dockeru na: https://docs.docker.com/).

Za kontejnerizaciju FastAPI REST API-ja i Streamlit aplikacije, definirali smo Dockerfile datoteke. U ovim Dockerfile-ovima, koristimo *python:3.9-slim-buster* sliku kao osnovu i kopiramo naš kod u */app* direktorij u kontejneru. Također smo instalirali sve potrebne Python pakete korištenjem *pip install* naredbe. Ove definicije kontejnera, s drugim potrebnim konfiguracijskim datotekama, definirane su u Docker Compose datoteci. Naša Docker Compose datoteka definira dvije usluge - FastAPI REST API uslugu i Streamlit uslugu, mapira potrebne portove te stvara *docker network* kako bi kontejneri jednostavnije komunicirali. Nakon što smo sve definirali u Docker Compose datoteci, pokrećemo aplikaciju jednostavnom *docker-compose up* naredbom u terminalu.

Korištenje Docker-a i Docker Compose-a za kontejnerizaciju našeg FastAPI REST API-ja i Streamlit aplikacije omogućava nam jednostavno izvršavanje i upravljanje našom aplikacijom. Naše rješenje je lakše distribuirati i reproducirati na drugim računalima, a korištenje Docker-a pruža nam izoliranu okolinu za izvršavanje našeg koda, što čini našu aplikaciju sigurnijom i pouzdanijom.

4. Razvoj rješenja

U ovom poglavlju predstavit ćemo sve tehnologije korištene za razvoj rješenja te komponente koje smo razvili za:

- Pretprocesiranje
- Treniranje modela
- Validaciju modela

4.1 Organizacija koda

Da bismo osigurali lako upravljanje bibliotekama i alatima koje koristimo u svom projektu, uz jednostavnu instalaciju, uklanjanje ili ažuriranje paketa koristili smo Conda environment u našem projektu (više o Condi na: https://docs.conda.io/en/latest/). S druge strane, da bi svi članovi tima mogli efikasno raditi na projektu koristili smo GitHub repozitorij, a struktura projekta je bila sljedeća:

```
LumenDS/
├── docker/
│   ├── backend/
│   │   ├── model/
│   │   ├── Dockerfile
│   │   ├── main.py
│   │   ├── predict.py
│   │   └── requirements.txt
│   ├── frontend/
│   │   ├── images/
│   │   ├── app.py
│   │   ├── Dockerfile
│   │   └── requirements.txt
│   ├── docker-compose.yaml
│   └── server.sh
├── notebooks/
├── src/
└── README.md
```

Projekt je podijeljen u tri direktorija: *docker/*, *notebooks/* i *src/*. U *docker/* direktoriju je sav kod za uspješno pokretanje web aplikacije, sastoji se od poddirektorija *backend/* i *frontend/* gdje se u svakome od njih nalaze Python datoteke za pokretanje aplikacije te *Dockerfile* i *requirements.txt* datoteka za izgradnju posebnih docker image-a. U direktoriju se također nalazi *docker-compose.yaml* za definiranje posebnih usluga i ostalih postavki kontejnera te skripta *serve.sh* za pokretanje aplikacije. Direktorij *notebooks/* smo koristili za Jupyter bilježnice gdje smo radili razne vizualizacije te testirali dijelove koda.

```
src/
├── lightning-logs/
├── model/
│   ├── models/
│   ├── densenet.py
│   ├── instrument-model.py
│   └── instrument-dataset.py
├── config.py
├── predict.py
├── preprocessing.py
├── train.py
└── validate.py
```

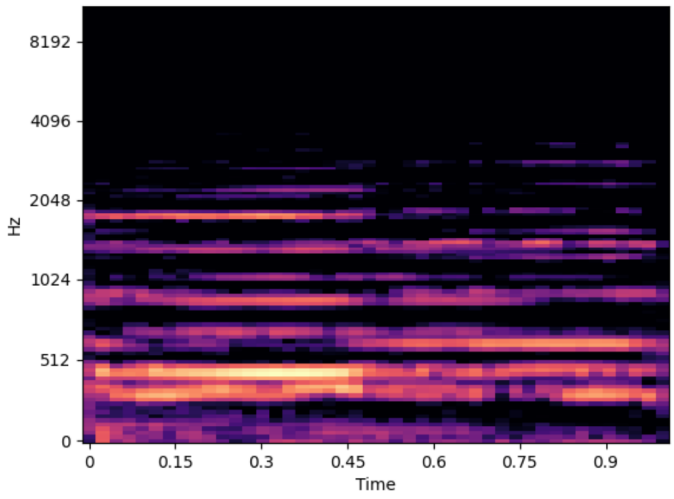
Najvažniji dio rješenja nalazi se u *src/* direktoriju koji sadrži poddirektorije *lightning-logs/* i *model/*. Direktorij *lightning-logs* sadržavao je logove treninga modela iz kojih smo pratili metrike, a u direktoriju *model* smo definirali arhitekture koje smo koristili za treniranje. U datotekama *instrument-model.py* i *densenet.py* se nalaze CNN arhitekture, a u datoteci *instrument-dataset.py* se stvarao dataset za trening iz već pretprocesiranih podataka, modele smo spremali u poddirektorij *models/*.

Ostale datoteke u direktoriju *src/* služile su za pokretanje raznih dijelova rješenja. Skripta *preprocessing.py* pokrenula bi pretprocesiranje definiranog skupa podataka te ga spremila na disk, skripta *train.py* pokrenula bi treniranje definirane arhitekture nad definiranim podacima, skripta *validate.py* pokrenula bi validaciju nad validacijskim skupom podataka ispisujući definirane metrike, skripta *predict.py* stvara json file sa klasifikacijom testnog skupa podataka, a svi parametri ovih skripti mogu se podesiti u datoteci *config.py*.

4.2 Pretprocesiranje

Librosa je popularna Python biblioteka za obradu zvuka i glazbe. Omogućuje jednostavnu manipulaciju i analizu zvučnih datoteka, kao i izvlačenje raznih karakteristika zvuka koje se mogu koristiti u strojnom učenju i drugim analitičkim zadacima (više o librosi na: https://librosa.org/doc/latest/index.html).

U našem projektu, koristili smo librosu za stvaranje log mel spektrograma kao dio procesa pretprocesiranja audio datoteka prije nego što smo ih poslali na treniranje ili klasifikaciju. Također smo koristili librosu za augmentaciju zvuka, što nam je pomoglo da poboljšamo performanse našeg modela strojnog učenja. Ova tehnika nam je omogućila da stvorimo više primjera za trening modela tako da smo manipulirali zvučnim signalima na različite načine, kao što su promjena visine tona, dodavanje šuma ili uklanjanje dijelova signala. Librosa, u kombinaciji s bibliotekom Matplotlib, također omogućava razne mogućnosti vizualizacije audio signala poput spektrograma i ostalih vrsta vremensko-frekvencijskih predstavljanja audio signala.



Slika 4.1: Primjer mel spektrograma prikazanog pomoću librose i matplotliba

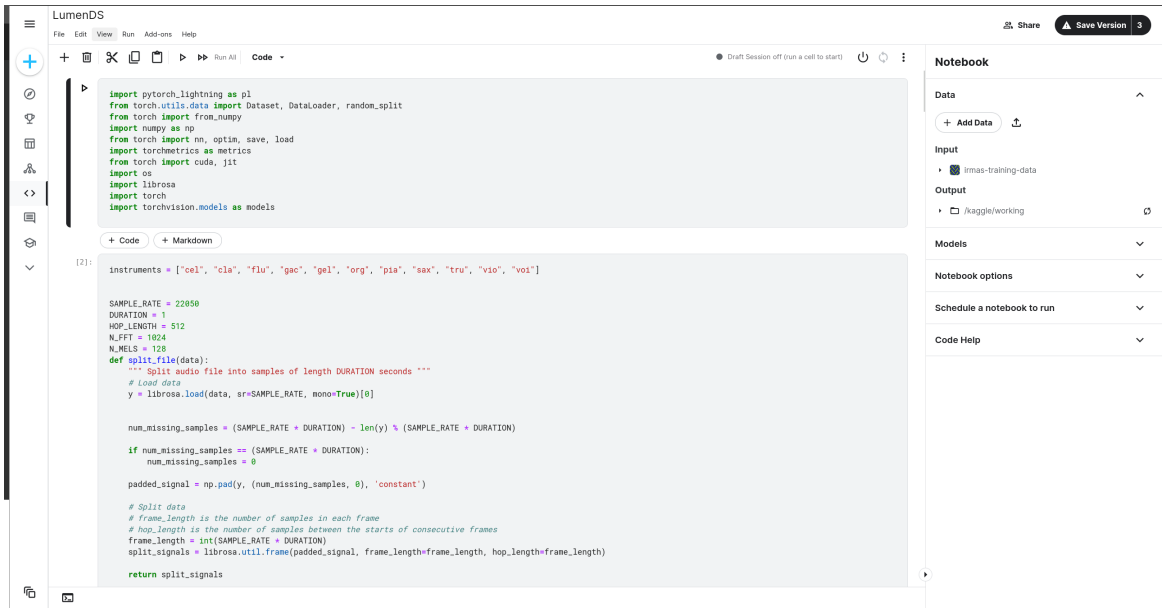
4.3 Model

Konačno smo došli do glavne komponente našeg rješenja, a to je model za klasifikaciju instrumenata. Za razvoj modela koristili smo PyTorch Lightning, framework za istraživanje i razvoj dubokih neuronskih mreža. PyTorch Lightning nam omogućuje da se fokusiramo na razvoj samog modela, dok se brine za sve ostale detalje kao što su inicijalizacija, treniranje i validacija modela, upravljanje optimizatorima i checkpointiranje modela. Jedna od glavnih prednosti PyTorch Lightning-a je njegova modularna arhitektura, koja omogućuje brzo prototipiranje i fleksibilno podešavanje modela. Također, PyTorch Lightning uključuje ugrađene funkcionalnosti koje nam omogućuju brzo eksperimentiranje s različitim optimizatorima, funkcijama gubitka i arhitekturama modela, što čini ovaj okvir idealnim za istraživanje novih ideja i tehnika u području dubokog učenja (više o Pytorch Lightning-u na: https://lightning.ai/docs/pytorch/stable/).

Nakon treniranja modela morali smo ga spremiti, za to smo koristili ONNX (Open Neural Network Exchange) open-source format za razmjenu modela strojnog učenja između različitih okvira za strojno učenje. Za klasifikaciju modela koristili smo alat ONNX Runtime koji omogućuje izvođenje modela strojnog učenja u ONNX formatu na različitim platformama. ONNX Runtime je bio puno bolja opcija za pokretanje modela u kontejneru zbog manje potrebnih računalnih resursa naspram preuzimanja cijele Pytorch biblioteke (više o ONNX na: https://onnx.ai/onnx/intro/).

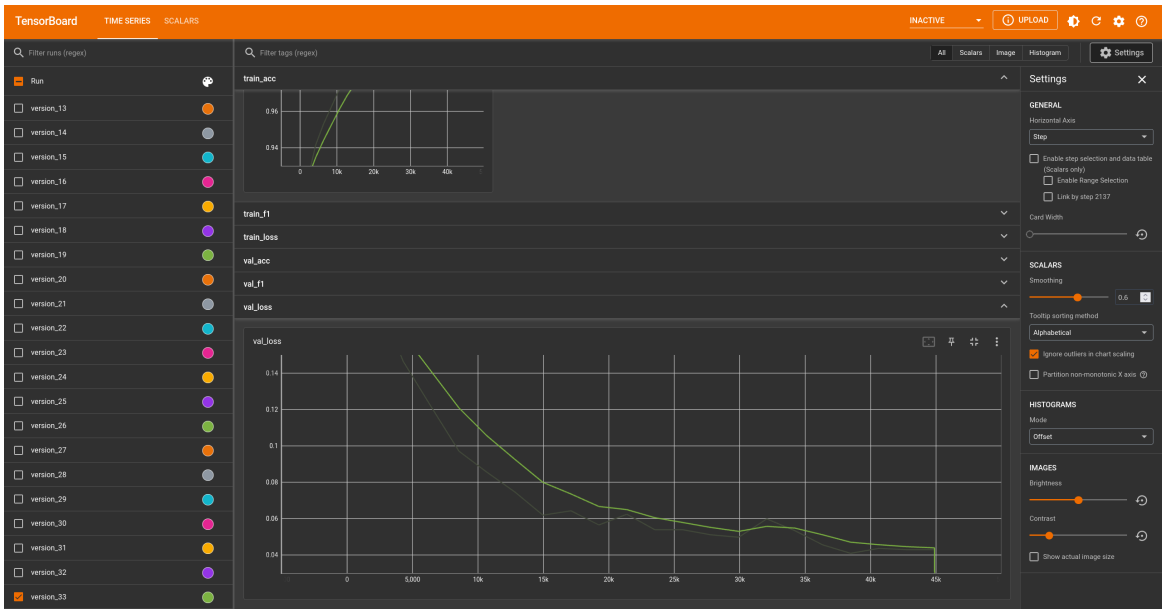
4.3.1 Trening

Kako bi ubrzali proces treniranja i isprobali što više modela odlučili smo trenirati modele u oblaku. Za to smo odabrali platformu Kaggle koja se inače koristi za data science natjecanja, objavu podatkovnih skupova, ali ima i opciju korištenja računalnih resursa poput GPU-a i TPU-a u oblaku. Na Kaggle-u je već bio objavljen IRMAS podatkovni skup pa smo ga samo dodali u naš Kaggle Notebook u ćelije zalijepili naš preprocessing pipeline te kod za definiciju i trening modela te trenirali model u oblaku. Kaggle omogućuje korisnicima 30 sati tjedno besplatnog korištenja GPU-a uz 3 druge opcije različitih akceleratora što je bilo dovoljno za naše potrebe. Iz Kaggle Notebooka lako smo preuzeli naše modele te podatke o treningu koje smo logirali.



Slika 4.2: Prikaz treninga na Kaggle-u

Iz dobivenih logova analizu treninga pojedinog modela vizualizirali smo alatom TensorBoard koji iz logova generiranih pomoću Pytorch Lightning-a omogućuje vizualno uspoređivanje performansi različitih modela i praćenje kako se performanse mijenjaju tijekom treniranja (više o TensorBoard-u na: https://www.tensorflow.org/tensorboard/get-started).



Slika 4.3: Prikaz TensorBoard vizualizacija