Web aplikacija za trening i evaluaciju modela dubokog učenja koristeći različite skupove podataka

Mirić, Viktor

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Viktor Mirić

Zagreb, 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentori:

Dr. sc. Tomislav Stipančić, dipl. ing.

Student:

Viktor Mirić

Zagreb, 2024

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem mentoru prof. dr. sc. Tomislavu Stipančiću na pruženoj pomoći i pristupačnosti tijekom izrade ovog rada.

Posebno se zahvaljujem svojoj obitelji i prijateljima na podršci tijekom studiranja.

Viktor Mirić

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske ispite studija strojarstva za smjerove:
Proizvodno inženjerstvo, inženjerstvo materijala, industrijsko inženjerstvo i menadžment, mehatronika i robotika, autonomni sustavi i računalna inteligencija

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 - 04 / 24 - 06 / 1**	
Ur.broj: **15 - 24 -**	

DIPLOMSKI ZADATAK

Student: **Viktor Mirić** JMBAG: 0035219948

Naslov rada na hrvatskom jeziku: **Web aplikacija za trening i evaluaciju modela dubokog učenja koristeći različite skupove podataka**

Naslov rada na engleskom jeziku: **A web application for training and evaluating deep learning models using different datasets**

Opis zadatka:

Moderne web tehnologije omogućuju integraciju i dostupnost različitih rješenja temeljenih na umjetnoj inteligenciji, koja su potom spremna za korištenje od strane korisnika bilo gdje u svijetu. Cilj ovog rada je izraditi web aplikaciju za integraciju programskih rješenja koristeći Next.js i PyTorch tehnologije. U web aplikaciju je potrebno integrirati proces razvoja neuronske mreže trenirane na većim količinama dostupnih skupova podataka. Taj proces treba obuhvatiti definiranje arhitekture mreže, postavljanje parametara za treniranje mreže, implementaciju evaluacijskih metrika, te prikaz konačnog rješenja putem web sučelja. Rad je potrebno temeljiti na JavaScriptu/TypeScriptu za tehnologiju kod klijenta (engl. frontend) te koristiti Flask kao tehnologiju s poslužiteljske strane (engl. backend) te PyTorch kao radnu okolinu za realizaciju modela umjetne inteligencije.

U radu je potrebno:

- izraditi web aplikaciju koristeći Next.js i PyTorch tehnologije
- integrirati proces razvoja neuronske mreže, uključujući definiranje arhitekture mreže i parametara treninga
- omogućiti korisniku odabir jednog od dostupnih skupova podataka za trening neuronske mreže
- prikazati finalno rješenje u sklopu aplikacije, uz odgovarajuće korisničko sučelje
- evaluirati funkcionalnost i korisničko iskustvo u stvarnim uvjetima korištenja.

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan:	Datum predaje rada:	Predviđeni datumi obrane:
26. rujna 2024.	28. studeni 2024.	5., 6. i 9. prosinca 2024.

Zadatak zadao:
Izv. prof. dr. sc. Tomislav Stipančić

Predsjednik Povjerenstva:
Prof. dr. sc. Ivica Garašić

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

SAŽETAK

Ovaj rad istražuje mogućnosti olakšavanja korištenja umjetne inteligencije i dubokog učenja, tehnologije koje sve više nalaze primjenu u različitim područjima. Motivacija ovog rada je razviti alat koji će korisnicima pružiti intuitivno sučelje za razvoj modela dubokog učenja bez potrebnog tehničkog znanja u toj domeni.

Kao rješenje, razvijena je web aplikacija koja korisnicima omogućuje jednostavnu izradu i prilagodbu modela dubokog učenja. Sustav je implementiran korištenjem modernih tehnologija. Korisničko sučelje razvijeno je u programskom okviru Next.js, dok je pozadinska logika izrađena uz pomoć programskog okvira Flask. Aplikacija koristi PyTorch za treniranje neuronskih mreža, omogućujući korisnicima prilagodbu arhitekture mreže te vizualizaciju rezultata i performansi modela.

Testiranje i evaluacija aplikacije provedena je na poznatom skupu podataka za klasifikaciju cvijeta perunike. Model je pokazao visoku preciznost, postigavši točnost predikcije od 100% na tesnom skupu podataka.

Ključne riječi: Strojno učenje, duboko učenje, neuronske mreže, web aplikacije, Python, JavaScript, Next.js, Flask, PyTorch

SUMMARY

This paper explores the possibilities of facilitating the use of artificial intelligence and deep learning, technologies that are increasingly being used in various fields. The motivation for this paper is to develop a tool that will provide users with intuitive interface for developing deep learning models without the need for technical knowledge in this domain.

As a result, a web application was developed that allows users to easily create and customize deep learning models. The system is implemented using modern technologies. The user interface is developed in the Next.js framework, while the backend logic is built using the Flask framework. The application uses PyTorch to train neural networks, allowing users to customize the network architecture and visualize the results and performance of the model.

The application testing and evaluation was conducted on a well-known dataset for iris flower classification. The model demonstrated high precision, achieving 100% prediction accuracy on a test dataset.

Key words: Machine learning, deep learning, neural networks, web applications, Python, JavaScript, Next.js, Flask, PyTorch

1. UVOD

Razvoj tehnologija umjetne inteligencije i dubokog učenja otvorio je nove mogućnosti za rješavanje složenih problema i optimizaciju procesa u raznim područjima, od industrije pa sve do medicine. Unatoč njihovoj širokoj primjeni, mnogi korisnici suočavaju se s izazovima implementacije ovih tehnologija zbog složenosti alata i visokih zahtjeva za tehničkim znanjem. Stoga, postoji rastuća potreba za jednostavnim i intuitivnim platformama koje omogućuju širokoj publici pristup potencijalu umjetne inteligencije.

Cilj ovog rada je izraditi web aplikaciju koja integrira proces razvoja neuronskih mreža, omogućujući korisnicima da na temelju vlastitih podataka treniraju prilagođene modele dubokog učenja. Kroz ovu aplikaciju, korisnici će moći učitati skupove podataka, definirati arhitekturu mreže, odabrati parametre za trening te vizualizirati rezultate i performanse modela. Na taj način, rad nastoji smanjiti tehničke prepreke u primjeni umjetne inteligencije.

Aplikacija je razvijena korištenjem suvremenih tehnologija: Next.js-a za frontend, Flask-a za backend, te PyTorcha za implementaciju modela dubokog učenja. Next.js omogućuje izradu korisničkog sučelja koji će uz pomoć REST API-a komunicirati s Flask mikro-servisom na kojemu će biti integriran PyTorch za izvršavanje funkcija dubokog učenja.

Važnost ovog rada leži u pojednostavljenju razvoja prilagođenih modela dubokog učenja, što korisnicima omogućuje bolju iskoristivost njihovih podataka. Rad pruža praktičan alat koji spaja najnovije tehnologije web razvoja i dubokog učenja, potičući širu primjenu umjetne inteligencije u različitim domenama.

U nastavku rada, detaljno će biti objašnjena teorijska osnova, korištene tehnologije, arhitektura sustava te rezultati korištenja aplikacije u stvarnim uvjetima, čime se pruža cjelovit pregled razvijenog rješenja.

2. TEORIJSKA OSNOVA RADA

U ovom poglavlju objašnjavaju se ključni teorijski pojmovi i koncepti potrebni za razumijevanje rada te razvoj i implementaciju web aplikacija koje koriste umjetnu inteligenciju. Prvi dio poglavlja posvećen je definiciji i važnosti web aplikacija, njihovoj ulozi u suvremenom svijetu, te specifičnim primjenama u području umjetne inteligencije. Drugi dio daje pregled osnovnih pojmova iz umjetne inteligencije i dubokog učenja, s naglaskom na prilagođene modele dubokog učenja koji omogućuju rješenja prilagođena specifičnim potrebama korisnika.

2.1. Web aplikacije

Web aplikacije su programska rješenja dostupna korisnicima putem web preglednika (engl. *web browser*), a pohranjene su na udaljenim poslužiteljima. Za razliku od tradicionalnog softvera, koji zahtijeva instalaciju na lokalno računalo, web aplikacije pokreću se putem internetske veze, čime se značajno povećava njihova dostupnost i pristupačnost [1]. Razvoj web aplikacija postao je ključan za mnoge industrije jer omogućuje organizacijama da brzo i učinkovito lansiraju proizvode i usluge na globalno tržište.

Uz kontinuirani rast i širenje interneta tijekom posljednjih desetljeća, web aplikacije stekle su ogromnu popularnost zahvaljujući svojoj fleksibilnosti i jednostavnosti korištenja. Omogućuju korisnicima pristup s različitih uređaja i pojednostavljuju razvoj za više platformi koristeći jedinstvenu kodnu bazu, što omogućuju standardizirane funkcionalnosti web preglednika.

2.1.1. Web preglednici

Web preglednici imaju ključnu ulogu u popularizaciji web aplikacija, jer služe kao univerzalna platforma za njihovo korištenje putem internetskih protokola i standarda. Omogućuju korisnicima dohvaćanje informacija s interneta pomoću internetske veze, pri čemu se podaci prenose putem protokola za prijenos hiperteksta (engl. *Hypertext Transfer Protocol*, akraćeno HTTP), koji definira standarde za prijenos informacija [2].

Svaka web stranica ima jedinstvenu adresu, poznatu kao URL (engl. *Uniform Resource Locator*), putem koje joj se pristupa. Kada korisnik unese URL, web preglednik koristi sustav za traženje domena (engl. *Domain Name System*, skraćeno DNS) kako bi pronašao odgovarajuću IP adresu povezanu s tim URL-om. Nakon što se pronađe IP adresa, preglednik šalje zahtjev poslužitelju na kojem se nalazi tražena web aplikacija [3].

Kada poslužitelj odgovori, web preglednik prima podatke, analizira ih i prikazuje sadržaj pomoću jezika za označavanje hiperteksta (engl. *Hypertext Markup Language*, skraćeno HTML). HTML definira strukturu i izgled web stranica te je osnovni jezik za prikaz web aplikacija.

Tijekom godina, web preglednici su značajno napredovali, uvodeći nove začajke i poboljšavajući korisničko iskustvo. Danas najpopularniji preglednici uključuju Mozilla Firefox, Google Chrome, Microsoft Edge i Apple Safari.

2.1.2. Razvoj web aplikacija

Razvoj web aplikacija uključuje niz faza i koristi različite tehnologije i alate za postizanje kvalitetnog korisničkog iskustva, funkcionalnosti i skalabilnosti. Programske tehnologije i alati za razvoj web aplikacija značajno su napredovali, posebno u JavaScript ekosustavu, koji se relativno brzo mijenja i gdje se konstantno pojavljuju nove programske biblioteke i okviri. Kao rezultat toga osim što se poboljšava iskustvo korisnika, poboljšava se i iskustvo programskih inženjera.

Proces se obično sastoji od planiranja, dizajna, implementacije, testiranja i održavanja [4, 5].

1. **Planiranje i analiza zahtjeva**
   Prvi korak u razvoju web aplikacije je razumijevanje ciljeva aplikacije i potreba korisnika. Definira se glavna funkcija i ključne značajke, te se izrađuje inicijalna struktura projekta.
2. **Planiranje korisničkog sučelja**
   U ovoj fazi se provodi dizajn korisničkog sučelja (UI) i korisničkog iskustva (UX). Vizualizira se tijek interakcije, tok podataka, s ciljem da se osigura intuitivno korištenje aplikacije
3. **Izbor tehnologija i razvojna okruženja**
   Tehnologije odabrane za razvoj web aplikacije ovise o opsegu i zahtjevima projekta i tehničkim preferencijama tima. Današnji uobičajeni programski okviri za korisnička sučelja, tj. frontend bazirani su na JavaScript programskom jeziku, te su neki od najpoznatijih okvira React.js, Vue.js, Angular i Svelte. Za pozadinsku logiku odnosno backend postoji širi izbor što se tiče programskih jezika, te su neki od poznatijih backend programskih okvira Node.js (JavaScript), Django (Python), ASP.NET Core (C#) i Laravel (PHP). Što se tiče odabira tehnologije za bazu podataka, najpopularniji

su PostgreSQL, MongoDB i MySQL. Uz ove osnovne tehnologije, često se odabiru dodatni moduli i biblioteke za rješavanje određenih funkcionalnosti, te alati za kontrolu verzija i implementacije aplikacije.

4. **Frontend razvoj**

   Frontend predstavlja dio aplikacije koji je vidljiv korisnicima i s kojim oni dolaze u interakciju. Razvoj frontend-a uključuje izradu web stranica, oblikovanje elemenata i implementaciju interaktivnosti, te komunikaciju s pozadinskim dijelom aplikacije (backend-om). Da bi se izgradio kvalitetan frontend, potrebno je poznavanje HTML-a za strukturu sadržaja, CSS-a za stilizaciju elemenata i JavaScript-a za dodavanje funkcionalnosti i interaktivnosti [6].

5. **Backend razvoj**

   Backend predstavlja pozadinski dio koji upravlja poslovnom logikom, rukovanjem podacima i autentifikacijom korisnika. Backend se uglavnom odnosi za razvoj aplikacijskih programskih sučelja (engl. *Application Programming Interface*, skraćeno API) koji omogućuju komunikaciju između frontenda i baze podataka te implementiraju sigurnosne mjere za zaštitu podataka korisnika [7].

6. **Integracija i testiranje**

   Testiranje web aplikacije je ključno za otkrivanje i ispravljanje grešaka, te za osiguravanje pravilne funkcije svih komponenti. Bitan pojam kod integracije većih projekata je kontinuirana integracija (engl. *Continuous Integration*, skraćeno CI), što je zapravo praksa automatizacije integracije promjene koda od više suradnika na jednom softverskom projektu, odnosno spajanje promjena koda u središnji repozitorij gdje se pokreću testovi i izgradnja. Automatizirani alati koriste se za potvrdu ispravnosti novog koda prije integracije [8].

7. **Implementacija i održavanje**

   Nakon završetka testiranja, aplikacija se implementira na produkcijsku okolinu, odnosno server ili cloud platformu, kao što su AWS, Google Cloud ili Azure, čime postaje široko dostupna korisnicima. Implementacija uključuje konfiguraciju servera, baze podataka i sigurnosnih postavki.

2.2. Umjetna inteligencija

Umjetna inteligencija (engl. *Artificial Intelligence*, skraćeno AI) odnosi se na sposobnost strojeva i računalnih sustava da izvršavaju zadatke donošenjem odluka i oponašaju ljudsku

inteligenciju. Ova tehnologija obuhvaća niz funkcija, kao što su učenje, prepoznavanje i zaključivanje. Sustavi umjetne inteligencije dizajnirani su za analizu podataka, prepoznavanje uzoraka i donošenje odluka, pri čemu često unapređuju svoje sposobnosti kroz proces poznat kao strojno učenje. AI postaje sve više prisutan i omogućava računalima izvođenje složenih zadataka često brže i preciznije od ljudi, čime se omogućuje automatizacija velikog broja procesa u raznim domenama [9, 10].

2.2.1. Povijesni pregled umjetne inteligencije

Razvoj umjetne inteligencije započeo je sredinom 20. stoljeća s ciljem stvaranja sustava koji bi mogli razmišljati poput ljudi. Inicijalni su se pokušaji fokusirali na simboličko zaključivanje i formalnu logiku, no ograničeni računalni resursi i manjak podataka sprječavali su širu primjenu. Pojam "umjetna inteligencija" službeno je prvi puta spomenut na konferenciji u Dartmouthu 1956. i smatra se formalnim početkom ovog područja [11].

Osamdesetih godina razvijenu si sustavi specijalizirani u donošenju odluka u raznim područjima. Rastuća računalna snaga devedesetih dodatno je ubrzala istraživanja, gdje je nakon toga veliki trenutak bio 1997. godine kada je IBM-ovo super računalo Deep Blue pobijedilo šahovskog prvaka Garryja Kasparova [12].

U 21. stoljeću razvoj strojnog i dubokog učenja omogućilo je širu primjenu AI-a. Značajni događaji poput lansiranja ChatGPT-a 2022. pokazali su potencijal ove tehnologije za svakodnevnu primjenu u raznim domenama.

2.2.2. Strojno učenje

Strojno učenje (engl. *Machine Learning*, skraćeno ML) je podskup umjetne inteligencije koji omogućuje računalima da iz velike količine podataka izvlače obrasce na temelju kojih stječu novo znanje i zapravo uče bez eksplicitnog programiranja svakog specifičnog zadatka. Uključuje razvoj algoritma koji identificira obrasce analizom velikih skupova podataka, kako bi mogao donositi odluke i predviđati ishode na temelju ulaznih podataka [13].

Slika 1. Odnos između strojnog i dubokog učenja

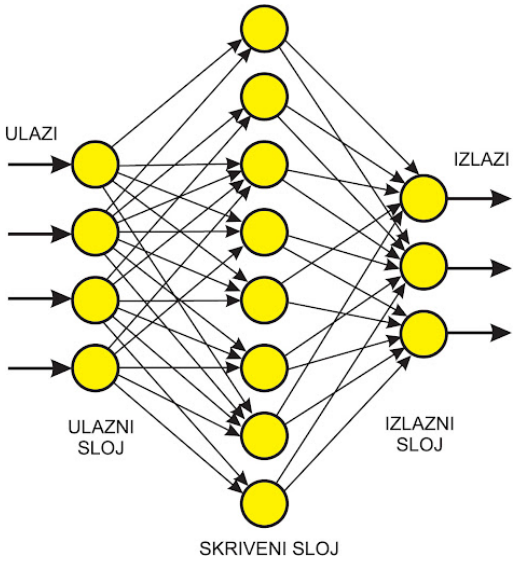
Glavni tipovi strojnog učenja dijele se u tri kategorije:

- Nadzirano učenje (engl. *supervised learning*) – Najčešće primijenjeni princip, radi se o učenju algoritma iz skupa podataka koji sadrži ulazne i izlazne podatke. Kreira se model koji nastoji prepoznati vezu između ulaza i izlaza, kako bi mogao predvidjeti izlazne vrijednosti na temelju ulaznih podataka. Najčešći primjer nadziranog učenja je klasifikacija.
- Nenadzirano učenje (engl. *unsupervised learning*) – Algoritam uči na temelju neoznačenih podataka bez ciljnih vrijednosti. Ovim se algoritmom najčešće otkrivaju skriveni obrasci ili grupiraju podaci.
- Pojačano učenje (engl. *reinforcement learning*) – Pristup gdje algoritam uči iz vlastitog iskustva kroz interakciju s okolinom, od koje dobiva povratne informacije u obliku nagrada ili kazni, omogućujući mu optimizaciju ponašanja u određenim situacijama. Kroz takvo učenje algoritam detektira akcije koje maksimiziraju dobivenu nagradu u nekom procesu.

2.2.3. Duboko učenje

Duboko učenje (engl. *Deep Learning*) je podskup strojnog učenja, koji se temelji na umjetnim neuronskim mrežama za modeliranje složenih obrazaca i relacija između podataka. Neuronske

mreže, kao temelj dubokog učenja, sastoje se od velikog broja međusobno povezanih jedinica koje nazivamo neuroni, organiziranih u slojeve. Ti slojevi uključuju ulazni sloj (koji prima početne, neobrađene podatke), skrivene slojeve (gdje se odvija većina obrade i transformacije informacija), i izlazni sloj (koji generira konačnu izlaznu vrijednost modela) [14].



Slika 2. Slojevi neuronske mreže

Inspirirane načinom na koji ljudski mozak obrađuje informacije, neuronske mreže koriste vlastite vrijednosti koje se nazivaju težine (engl. *weights*) i pomak (engl. *bias*), koje omogućavaju neuronu da „uči" kroz prilagodbu tih vrijednosti na temelju podataka. Težine definiraju važnost svakog ulaza u modelu, a pomaci translatiraju vrijednost izlaza. Neuronske mreže također koriste funkcije aktivacije (kao što su ReLU, Sigmoid, ili Tanh) koje uvode nelinearnost što omogućuje modelu da uči složene relacije unutar podataka.

Neuronska se mreža zapravo može shvatiti funkcijom koja prima ulazne vrijednosti i pretvara ih u određen tip izlaza. Vrijednosti prolaze kroz svaki sloj u kojem neuroni zbrajaju ulaz s vlastitim težinama i pristranosti, zatim se na tu vrijednost primjenjuje aktivacijska funkcija, te se u konačnici vrijednost šalje na sljedeći sloj [15].

Neuronske se mreža treniraju minimizacijom gubitka, koji se odnosi na numeričku metriku koja kvantificira koliko su kvalitetno predviđanja neuronske mreže usklađena s stvarnim vrijednostima. Točnije, mjeri razliku između predviđenih vrijednosti i pravih oznaka, pružajući način za procjenu izvedbe modela. Krajnji cilj tijekom obuke je minimizirati ovaj gubitak, čime se poboljšava točnost modela.

Optimizacijski algoritmi su metode koje se koriste za prilagodbu parametara modela kako bi se minimizirala funkcija gubitka. Ovi algoritmi iterativno mijenjaju težine modela na temelju gradijenta funkcije gubitka [16].

2.2.3.1. Primjene dubokog učenja

Duboko učenje je revolucioniralo brojne industrije zahvaljujući svojoj sposobnosti u obradi velikih količina podataka i donošenja preciznih odluka na temelju njih. Neke od najčešćih oblika primjena su [17]:

- Računalni vid - Duboko učenje omogućilo je razvoj naprednih sustava za prepoznavanje objekata, lica, i uzoraka. Primjeri mogu biti autonomna vozila (računalo iz analize okoline donosi odluke o upravljanju vozila), medicinska dijagnostika (otkrivanje tumora na temelju rendgenskih snimaka) i razvrstavanje otpada (prepoznavanje vrste otpada za danju obradu).
- Obrada prirodnog jezika (engl. *Natural Language Processing*, skraćeno NLP) - NLP modeli temeljeni na dubokom učenju, poput BERT-a i GPT-a, donijeli su velike napretke u razumijevanju i procesiranju jezika. Koristi se u chat-botovima, automatskom prevođenju, analizi sentimenta i prepoznavanju govora.
- Prediktivna analitika - Primjena u slučajevima gdje se na temelju prošlih podataka pokušavaju predvidjet budući trendovi, npr. u financijama i ekonomiji gdje se predviđaju cijena dionica ili drugi oblik vrijednosnica.
- Generativni modeli - Duboko učenje omogućuje stvaranje novog sadržaja, poput slika, glazbe, ili teksta.
- Personalizacija sadržaja - Pristup korištenja dubokog učenja gdje se na temelju korisničkih preferencija ili ponašanju kreiraju personaliziranu preporuku sadržaja za svakog korisnika. Kao primjer, koriste ga velike društvene mreže, zatim platforme poput Netflix-a ili YouTuba i Google za ciljani prikaz reklama.

2.2.4. Učenje neuronskih mreža

Učenje neuronskih mreža temelji se na iterativnom procesu optimizacije koji uključuje dvije ključne faze: propagaciju unaprijed (engl. *forward propagation*) i propagaciju unatrag (engl. *backpropagation*). Cilj ovih koraka je minimizirati pogrešku modela kako bi neuronska mreža

bolje predviđala izlazne vrijednosti na temelju ulaznih, te omogućiti mreži da generalizira na novim nepoznatim podacima [14].

1. Propagacija unaprijed

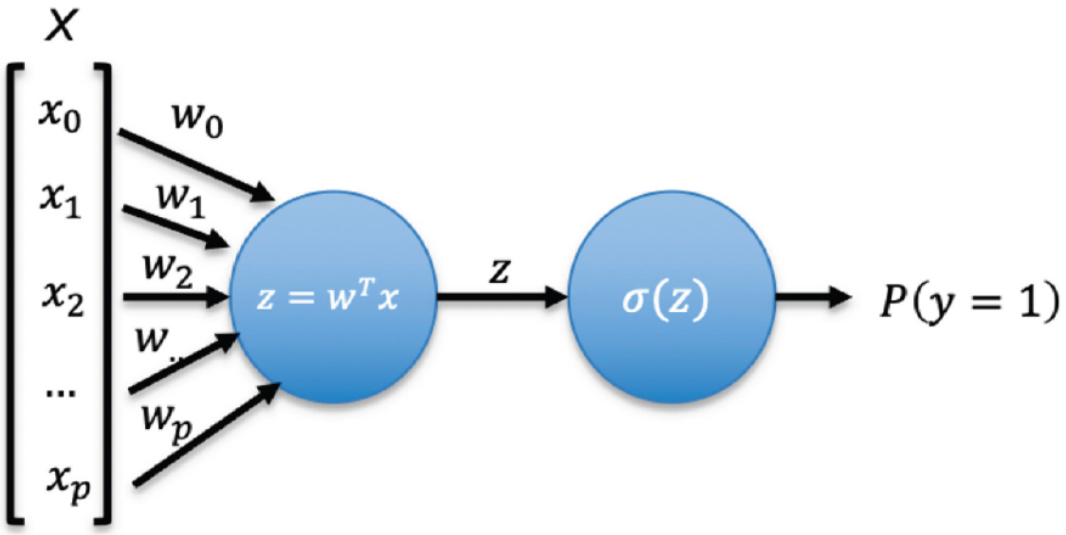
Odnosi se na izračun izlaznih vrijednosti neuronske mreže prolaskom ulaznih podataka kroz sve slojeve, od ulaznog sloja preko skrivenih slojeva do izlaznog sloja. Svaki neuron izračunava ulazne vrijednosti linearnom transformacijom težina i pomaka, nakon čega slijedi primjena aktivacijske funkcije na te vrijednosti [18]:

$$z^{(l)} = W^{(l)} x^{(l-1)} + b^{(l)}$$

$$x^{(l)} = \sigma\,(z^{(l)})$$

Gdje su:

- $W^{(l)}$ - Matrica težina sloja l
- $b^{(l)}$: pomak sloja l,
- $x^{(l-1)}$: aktivacije prethodnog sloja,
- σ: aktivacijska funkcija.



Slika 3. Matematički model neurona

Na kraju propagacije unaprijed, neuronska mreža računa predviđenu vrijednost koja se uspoređuje sa stvarnima izlazom kako bi se izračunala pogreška odnosno gubitak. Kao funkcija gubitka često se koristi srednja kvadratna pogreška (MSE).

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)$$

Gdje su:

- n - broj uzoraka
- y_i - stvarne vrijednosti
- $\hat{y}_i$ − predviđene vrijednosti

2. Propagacija unatrag

Nakon propagacije u naprijed i izračuna pogreške, započinje propagacija unatrag. Cilj ovog koraka je prilagoditi težine i pomake neurona u mreži kako bi se minimizirala funkcija gubitka. Ovo se postiže računanjem gradijenata funkcije gubitka u odnosu na težine koristeći pravilo ulančavanja, gdje gradijenti propagiraju kroz mrežu unatrag počevši od izlaznog sloja prema ulaznom. Gradijent se računa formulom [18]:

$$\frac{\partial \mathrm{C}}{\partial \mathrm{w}} = x^{l-1} \delta^{l}$$

Gdje su:

- x^{l-1}- aktivacija neurona ulazne vrijednosti prema težinama w
- δ^{l} - greška neurona izlaza od vrijednosti težina w

3. Optimizacija gradijentnim spustom

Gradijentni spust (engl. *gradient descent*) je metoda kojom se mijenjaju težine neurona mreže u negativnom smjeru gradijenta čime se kao posljedica smanjuje funkcija gubitka. Nove vrijednosti težina se računaju:

$$W^{l} = W^{l} - \eta \frac{\partial \mathrm{C}}{\partial \mathrm{w}}$$

Gdje je:

- η - stopa učenja (engl. *learning rate*)

Danas se često koriste moderne varijacije gradijentnog spusta, što su metode poput ADAM-a, RMSprop-a ili Nester-ov gradijentni spust, jer omogućuju bržu i stabilniju konvergenciju prilagođavanjem stope učenja za svaku težinu pojedinačno [19].

4. Iterativni proces i generalizacija

Proces učenja sastoji se od ponavljanja propagacije unaprijed i unatrag kroz više iteracija odnosno epoha, pri čemu mreža postepeno smanjuje gubitak i istovremeno poboljšava svoje performanse.

2.2.4.1. Izazovi kod učenja neuronskih mreža

Jedan od najznačajnijih izazova u području dubokog učenja je osiguravanje dovoljno velikih količina kvalitetnih podataka potrebnih za učinkovito treniranje modela. Modeli dubokog učenja zahtijevaju velik i raznolike skupove podataka koji pokrivaju širok raspon uzoraka kako bi model u konačnici mogao generalizirati. Prikupljanje takvih podataka može biti izazovno u domenama gdje podaci nisu lako dostupni, dok obrada i priprema podataka zahtijevaju složene postupke čišćenja, označavanja i standardizacije. Ovi koraci mogu često biti vremenski intenzivni i financijski zahtjevni [20].

Uz brigu za osiguranjem dovoljno velikih i kvalitetnih skupova podataka, potrebno je voditi i brige o pretreniranosti. Pretreniranost (engl. *overfitting*) događa se kada model prekomjerno iterira na trening skupu podataka pa ne uspijeva generalizirat na testnom skupu podataka, odnosno gubitak na trening skupu je nizak, a na testnom skupu visok. Pretreniranost se izbjegava tehnikama poput regularizacijom, povremenim isključivanjem određenih neurona tijekom treninga (engl. *dropout*) ili augmentacijom podataka [21].

Uz velike količine podataka, treniranje modela dubokog učenja zahtijeva značajne računalne resurse. Kako se kod rada s neuronskim mrežama koriste matrice i tenzori što upada u područje linearne algebre, za treniranje modela dubokog učenja koriste se grafički procesori (engl. *Graphics Processing Unit*, skraćeno GPU) i specijalizirani čipovi za izvođenje operacija s tenzorima (engl. *Tensor Processing Unit*, skraćeno TPU). Troškovi nabave i održavanja takve infrastrukture zahtijevaju velike količine kapitala, te se često takve usluge eksternaliziraju korištenjem računalnih resursa u oblaku.

3. PREGLED TEHNOLOGIJA I ALATA

U ovom poglavlju detaljnije ćemo opisati tehnologije i alate korištene za izradu web aplikacije za trening i evaluaciju modela dubokog učenja. Suvremeni razvoj web aplikacija omogućuje širok izbor različitih programskih okvira i biblioteka za izradu aplikacija na web platformi. Ovo je područje izrazito dinamično, s čestim pojavljivanjem novih alata koji olakšavaju rad programskim inženjerima i poboljšavaju korisničko iskustvo.

Poglavlje obuhvaća pregled tri glavne tehnologije korištene u izradi aplikacije: Next.js, Flask i PyTorch. Next.js je moderan okvir za razvoj korisničkih sučelja web aplikacija, omogućavajući prikaz (engl. *render*) na serveru i kreiranje statičnih stranica s izvrsnim performansama. Flask je mikro web okvir za razvoj backend-a, koji omogućava fleksibilnu i brzu implementaciju REST API-ja, što je ključno za komunikaciju između korisničkog sučelja i poslužitelja. Flask je odabran jer se radi o okviru za programski jezik Python, koji je često korišten u razvoju neuronskih mreža. PyTorch, popularna biblioteka za duboko učenje, pruža okruženje za razvoj, treniranje i evaluaciju prilagođenih neuronskih mreža, čime je idealan za integraciju u ovu aplikaciju.

Kroz ovaj pregled tehnologija razmotrit ćemo specifične funkcionalnosti svakog od alata i njihove prednosti za projekt.

3.1. Next.js

Next.js je vrlo moćan i svestran programski okvir koji omogućava brz razvoj skalabilnih aplikacija. Baziran je na React-u, najpopularnijoj programskoj biblioteci za izradu korisničkih sučelja, te ju proširuje full-stack mogućnostima, prikazivanjem na strani poslužitelja (engl. *server side rendering*, skraćeno SSR), pred memoriranjem (engl. *caching*) za bolje performanse i slično [22].

SSR je jedna od ključnih mogućnosti Next.js-a i predstavlja značajku koja je u rastućem trendu u svijetu web programiranja. Ukoliko se ne koristi SSR, JavaScript programski okvir poput React-a bi u potpunosti generirao sadržaj na klijentskoj strani, što znači da bi se korisniku slao uglavnom gotovo prazan HTML dokument, koji bi zatim JavaScript popunjavao izvršavanjem napisanog koda. Kod SSR-a, taj se dio izvršava na strani poslužitelja odnosno servera, što poboljšava brzinu učitavanja stranica, benefite za optimizaciju tražilica (engl. *search engine optimization*, skraćeno SEO), smanjuje računsko opterećenje na strani korisnika te omogućava upravljanjem dinamičkog sadržaja na serverskoj strani prije slanja korisniku [23].

Next.js rješava usmjeravanje (engl. *routing*) među stranicama aplikacije ugrađenom sustavu temeljenom na datotekama, gdje preslikava put do datoteke unutar strukture projekta izravno u URL, što pojednostavljuje upravljanje rutama i navigacijom.

Još jedna od glavni značajki Next.js-a je proširenje Fetch API-a ugrađenog u JavaScript okolinu izvršavanja (engl. *runtime*), koji se koristi za dohvaćanje podataka s poslužitelja putem HTTP protokola. Programski okvir pojednostavljuje korištenje i proširuje ga za memoizaciju zahtijeva, pred memoriranje podataka i revalidaciju. Ovim principom se optimizira dohvaćanje dinamičnih podataka na klijentu i serveru što rezultira većom fleksibilnošću kod razvoja, poboljšanju performansa i većom efikasnošću.

Programski okvir također sadrži ugrađenu podršku za pojednostavljenje stilizacije aplikacija CSS-om putem bilo koje od metoda, optimizira učitavanje slika, fontova i skripti za poboljšavanje temeljnih web pokazatelja, te sadržava poboljšanu podršku za TypeScript koji je danas sveprisutan u web razvoju, s boljom provjerom tipova i učinkovitijom kompilacijom.

3.2. Flask

Flask je mikro programski okvir za Python dizajniran za razvoj aplikacija uz visok stupanj prilagodbe i fleksibilnosti. Klasificira se kao mikro-okvir jer ne zahtjeva određene dodatne programske biblioteke ili alate jer izbjegava korištenje nepotrebnih komponenti po zadanim postavkama. Umjesto toga, fokusira se na pružanje samo osnovnih sastavnih blokova za web razvoj poput URL usmjeravanja, rukovanja HTTP zahtjevima i prikaz generiranih obrazaca poput HTML dokumenata. Ovaj minimalistički pristup omogućava odabir dodatnih proširenja prema potrebi, čineći Flask jako prilagodljivim u različitim slučajevima upotrebe [24].

Modularna arhitektura omogućuje dodavanje funkcionalnosti proširenjem prema potrebi, bilo da se radi o integraciji s bazom podataka, autentifikaciji ili vanjskom integracijom. Ta proširivost osigurava Flask-u da zadovolji potrebe od malih aplikacija, pa sve do većih složenih sustava, jer omogućava započinjanje s osnovnim postavkama i postupno skaliranje po potrebi.

Flask se koristi za razne vrste aplikacija, od jednostavnih API servisa, pa do web platformi s širokim rasponom značajki. Po prirodi je idealan izbor za mikro-servisne arhitekture bez poslužitelja, gdje su učinkovitost i skalabilnost kritični, te ga programeri zbog sposobnosti da podrži RESTful API često koriste za razvoj backend-a za web i mobilnih aplikacija [25].

3.3. PyTorch

PyTorch je programski okvir otvorenog koda (engl. *open source*) za Python i koristi se za strojno učenje, a razvio ga je Facebook-ov istraživački laboratorij za umjetnu inteligenciju. Stekao je značajnu popularnost u zajednicama strojnog učenja i dubokog učenja zahvaljujući svojoj fleksibilnosti i jednostavnosti upotrebe, što ga čini pogodnim za istraživačka i proizvodna okruženja. Još jedan vrlo popularan programski okvir za strojno učenje je kojeg je vrijedno spomenuti je TensorFlow razvijen od strane Google-a.

U dubokom učenju, tenzori su temeljni oblik podataka na kojima je PyTorch izgrađen i vrlo su slični nizovima (engl. *array*) i matricama, s kojima se mogu izvoditi matematičke operacija na većim skupovima podataka. Tenzor se može prikazati kao matrica, vektor, skalar ili neki drugi višedimenzionalni niz [26].

Rad s tenzorima omogućava vrlo brzo izvođenje računskih operacija iako se radi u Pythonu, koji kao jezik visoke razine (engl. *high level*) nije poznat po brzini i učinkovitosti izvođenja takvih operacija. Stvar je u tome da se tenzorima pristupa preko Python API-a, a računski se dio izvodi u kompiliranom C++ kodu optimiziranom za rad na hardveru, odnosno centralnoj procesnoj jedinici (engl. *Central Processing Unit,* skraćeno CPU) i grafičkoj procesnoj jedinici (engl. *Graphical Processing Unit,* skraćeno GPU). To omogućuje učinkovito izvođenje numeričkih izračuna i značajno poboljšava performanse u operacijama strojnog učenja.

Vrlo bitna stvar kod rada s programskim okvirima za duboko učenje je kvalitetna apstrakcija određenih operacija koje su bitne za proces učenja neuronskih mreža, poput PyTorch-ovog Autograd modula. Ovaj modul omogućava automatsko računanje diferencijacija, koji predstavlja temelj učenja dubokih neuronskih mreža, a glavna mu je funkcija praćenje svih operacija koje se izvode na tenzorima kako bi mogao izračunati sumu svih gradijenata tijekom propagacije unatrag. Autograd modul to postiže izradom i praćenjem dinamičkog računskog grafa koji definira izvršavanje operacija u hodu [27].

PyTorch sadrži razne module za pojednostavljenje procesa razvoja neuronskih mreža. Kao primjer, Torch.nn je jedan od temeljnih modula ovog programskog okvira, a se koristi za izgradnju ulaznih, sakrivenih i izlaznih slojeva neuronske mreže, odnosno arhitekture mreže. U Python-u se definira klasa, koja nasljeđuje svojstva i metode od Torch.nn.Module, koji je također klasa. Zatim se unutar inicijalizacijske metode Python-a definiraju slojevi neuronske mreže.

Velika prednost kod korištenja PyTorch-a je integracija s CUDA-om (engl. za *Compute Unified Device Architecture),* što je paralelna je računalna platforma i sučelje za programiranje aplikacija koju je razvila kompanija Nvidia. Omogućuje korištenje resursa GPU-a za opću namjenu, omogućujući poboljšanje performansa u aplikacijama s intenzivnim računskim zahtjevima. Korištenjem CUDA-e mogu se prebaciti operacije izvođenja određenih operacija s CPU-a na GPU, čime se mogu poboljšati performanse i ubrzati vrijeme razvoja. Korištenje grafičke procesorske jedinice u svijetu neuronskim mreža je vrlo korisno, zato što se kod rada s obradom grafike i izvođenju operacija kod neuronskih mreža koristi linearna algebra [28].

PyTorch nudi sveobuhvatan ekosustav prilagođen za razne domene strojnog učenja. Neke od tih programskih biblioteka su:

- TorchVision, koji se koristi za zadatke računalnog vida i pruža pomoćne funkcionalnosti za pertprocesiranje slika i prethodne trenirane modela koji se mogu koristiti kao temelj za razvoj vlastitih prilagođenih modela.
- TorchText za obradu prirodnog jezika, sadrži alate za tokenizaciju i rukovanje podacima.
- TorchAudio je biblioteka za obradu zvuka i signala s PyTorchom. Pruža funkcije za obradu signala i podataka, skupove podataka, implementaciju modela i komponente aplikacije.

PyTorch se koristi u akademskim i industrijskim okolinama zbog svojih performansi, fleksibilnosti, intuitivnog sučelja i aktivne zajednice. Neke istaknute kompanije koje koriste PyTorch za svoje aplikacije su Tesla koja ga koristi za svoj sustav autopilota, OpenAI za razvoj naprednih jezičnih modela poput GPT-3 i GPT-4, te Meta za razvoj algoritama za personalizaciju korisničkog sadržaja [29].

3.4. Ostale tehnologije

Next.js, Flask i PyTorch su glavne tehnologije na kojemu se temelje frontend i backend aplikacije izrađene u ovom radu, uz to koriste se dodatni manji alati za lakšu implementaciju određenih funkcionalnosti, neki od njih su:

- Scikit-learn modul za pomoćne funkcije za upravljanjem trening i test skupova podataka
- React-Query za upravljanje asinkronog stanja frontend aplikacije
- Tailwind CSS za stilizaciju i optimalno generiranje statičnih CSS datoteka
- PapaParse za obradu i čitanje CSV datoteka

- Shadcn UI za intuitivnu izradu korisničkog sučelja
- Recharts za prikaz grafikona putem React komponenata

4. IZRADA APLIKACIJE

U ovom poglavlju detaljno će se opisati plan i koraci izrade aplikacije za trening i evaluaciju modela dubokog učenja koristeći različite skupove podataka. Aplikacija je zamišljena kao rješenje koje omogućava korisnicima da putem intuitivnog web sučelja uvedu skup podataka, definiraju arhitekturu prilagođenih modela dubokog učenja, evaluiraju njihove performanse i zatim koriste model za generiranje predikcija na temelju odabranih ulaznih podataka. Kroz povezivanje Next.js frontend-a, Flask backend platforme i PyTorch programskog okvira za duboko učenje, aplikacija predstavlja pristupačnu platformu za rad s kompleksnim podacima.

Na početku poglavlja predstavit će se arhitektura sustava, uključujući način komunikacije između frontend i backend komponenata te njihovu integraciju. Zatim će se opisati implementacija backend-a koristeći Flask, kao i PyTorch koji će omogućiti trening prilagođenih modela dubokog učenja, kako bi se shvatila pozadinska logika aplikacije, pa će se u nastavku opisati implementacija frontend-a, gdje će se vizualno prikazivati podaci generirani na backend-u.

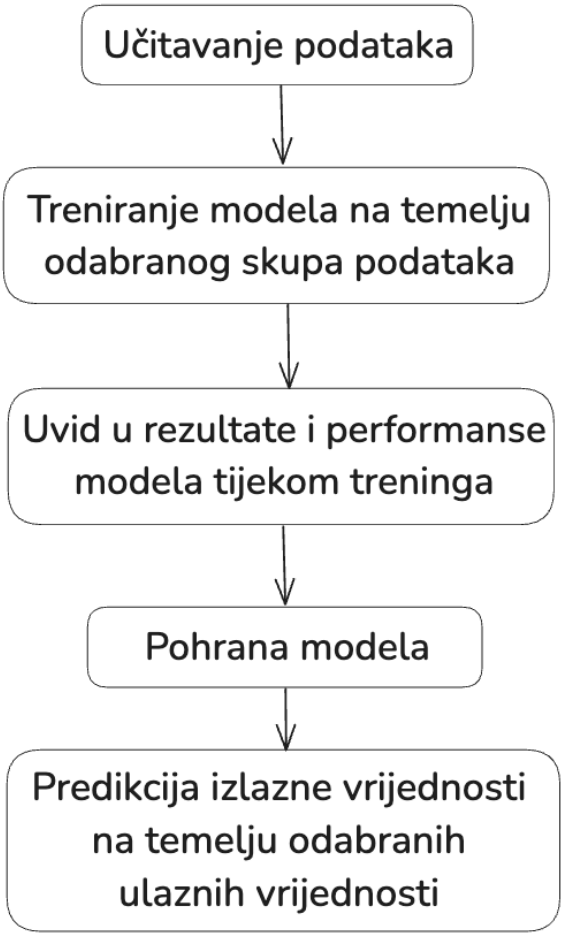
Cilj ovog poglavlja je pružiti čitatelju uvid u tehničke aspekte razvoja aplikacije, naglašavajući izazove i rješenja koja su primijenjena tijekom implementacije.

4.1. Arhitektura aplikacije

Kako bi se mogao definirati tok korištenja (engl. *workflow*) odnosno korisničko iskustvo, potrebno je poznavati glavne značajke na kojima se temelji funkcionalnost aplikacije. Glavne značajke koje korisniku moraju biti dostupne:

- Učitavanje (engl. *upload*) podataka na kojemu će se izraditi klasifikacijski model dubokog učenja, taj će tip podataka biti u obliku CSV (engl. *Comma Seperated Values*) datoteke, što je najčešći oblik za ovu primjenu.
- Pregled svih učitanih podatkovnih skupova, s mogućim pregledom podataka u tabličnom obliku.
- Treniranje modela na odabranom skupu, gdje će biti omogućen odabir ciljne varijable, te odabir hiper parametara modela poput broja i veličina sakrivenih slojeva, broj epoha, stopa učenja, aktivacijska funkcija, omjer trening i test skupa podataka, šum, normalizacija slojeva i nasumičnog generatora za reproducibilnost.

- Prikaz rezultata i performansi treniranja modela, gdje će se korisniku tijekom izvođenja procesa učenja prikazati osnovne metrike poput gubitka i preciznosti, te njihov grafički prikazi kroz vrijeme histogramom.
- Pohrana modela, gdje se određen model nakon treninga može spremiti kako bi se njime radila predikcija.
- Predikcija, gdje se može odabrat prethodno pohranjen model, koji će na temelju nekih odabranih ulaznih podataka generirat određenu izlaznu varijablu.



Slika 4. Struktura toka korištenja aplikacije

Kako je ranije definirano, backend aplikacija će se implementirati pomoću Flask-a, koji će definirati rute za komunikaciju između korisničkog sučelja (frontend-a) i pozadinske logike aplikacije. Ključni razlog korištenja Flask-a u ovom slučaju je njegova sposobnost integracije s PyTorch-om, koji omogućuje izvođenje funkcija dubokog učenja. Iako Next.js predstavlja

full-stack programski okvir koji podržava serversku logiku, Node.js trenutno ne nudi dovoljno robustan i intuitivan okvir ili biblioteku za duboko učenje poput PyTorch-a. Stoga je ovaj aspekt aplikacije izdvojen u zaseban mikro servis, specijaliziran za izvođenje logike vezane uz duboko učenje.

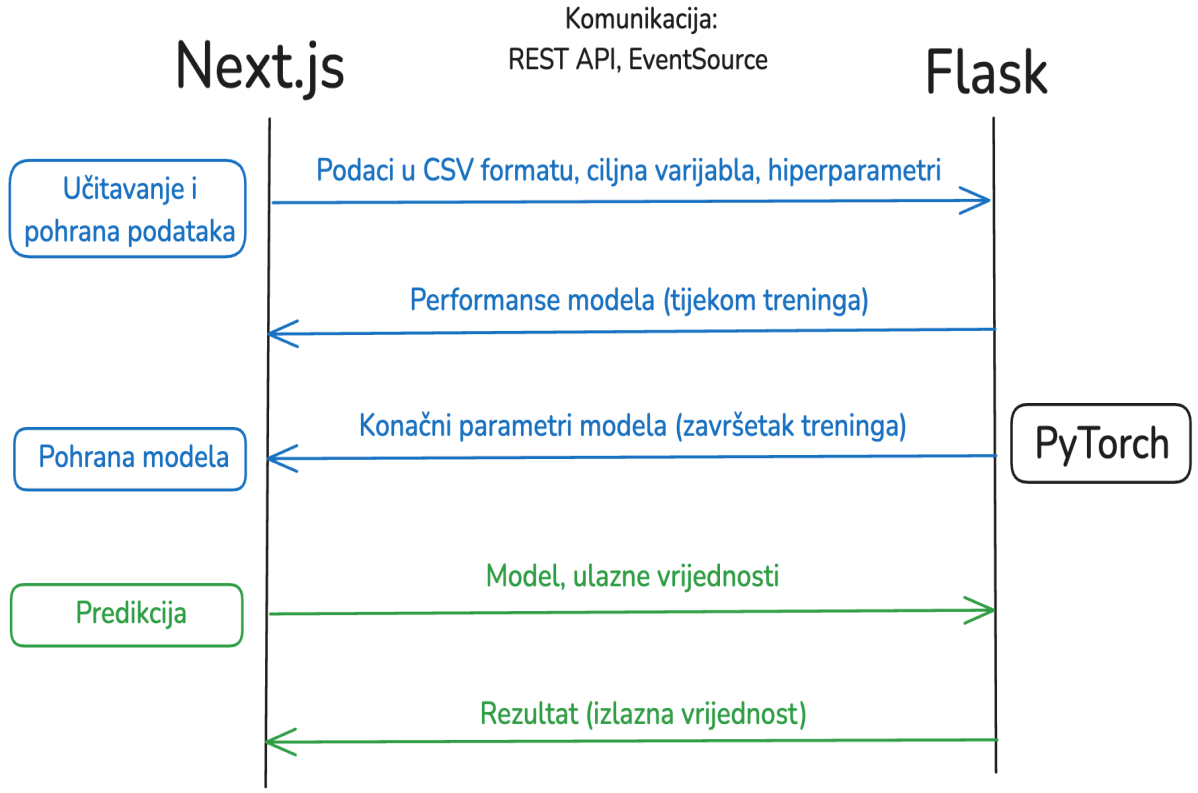
Aplikacija je osmišljena za lokalno izvršavanje, pri čemu će istovremeno raditi dva lokalna servera. Next.js koristi Node.js server, koji omogućuje pokretanje JavaScript koda izvan web preglednika, dok se Flask mikro servis izvršava na WSGI (engl. *Web Server Gateway Interface*) serveru [30, 31].

Aplikacija zahtijeva pohranu dvaju tipova podataka, učitanih podataka u obliku CSV datoteka i modela koje generira PyTorch. Podaci modela sadrže težine i pomake pojedinih neurona. Iako je uobičajeno koristiti baze podataka poput PostgreSQL, MySQL ili MongoDB za pohranu podataka, zbog malog broja podataka i izostanka relacija, podaci će se pohranjivati direktno na serveru.

Pohrana podataka odvijat će se na Node.js serveru, gdje se izvršava Next.js, koristeći ugrađeni fs (engl. *File System*) modul. Učitani podaci bit će spremljeni kao CSV datoteke, dok će modeli biti pohranjeni u JSON (engl. *JavaScript Object Notation*) formatu. Ovakav pristup značajno pojednostavljuje implementaciju aplikacije.

Komunikacija između frontend-a i backend-a će se izvršavati u 3 slučaja:

- Kada korisnik želi trenirati model, nakon što odabere skup podataka, ciljnu varijablu, te ostale hiper parametre moodela, ti se podaci šalju na Flask server koji ih obrađuje i koristi u izvršavanju određenih PyTorch modula.
- Tijekom treninga će se s Flask servera slati povrate informacije o performansi modela, ovo je moguće izvesti uz pomoć Websocket protokola ili EventSource sučelja. EventSource je web sučelje gdje klijent otvara trajnu jednosmjernu vezu s HTTP serverom, koji mu šalje podatke u obliku text/event-stream formata [32].
- Kada se želi raditi predikcija modela na temelju ulaznih podataka, klijent će slati podatke modela i ulaznih vrijednosti Flask mikro-servisu, koji će kreirati model, izvršiti predviđanje izlazne vrijednosti, te vratiti rezultat frontend-u.



Slika 5. Prikaz komunikacije između Next.js i Flask aplikacija

4.2. Backend

Backend aplikacije razvijen je koristeći Flask, programski mikro-okvir koji omogućuje jednostavnu i fleksibilnu implementaciju API-ja za komunikaciju s frontend-om. U ovom radu, Flask poslužitelj može se shvatiti kao mikro-servis koji obavlja ključne funkcije vezane uz obradu podataka, treniranje neuronskih mreža i evaluaciju modela dubokog učenja korištenjem PyTorcha. Funkcionalnosti će se modularno podijeliti, gdje će se funkcionalnosti razvrstati u zasebne komponente. Glavne funkcije koje backend treba ostvariti:

- API komunikacija: Definiranje RESTful API-ja za povezivanje s frontend aplikacijom, uključuje rute za učitavanje podataka i treniranje modela, rutu za vraćanje performansi modela tijekom treninga, te rutu za predikciju izlazne vrijednosti.
- Upravljanje podacima: Modul za prihvaćanje i obradu korisničkih podataka, što su CSV datoteka i hiper parametri, te validacija i pretvorba podataka u formate kompatibilne s PyTorch-om.
- Trening modela: Funkcionalnost za definiranje arhitekture neuronske mreže, prilagodbu hiper parametara i proces treniranja modela.

- Predikcija modela: Implementacija mogućnosti predikcije izlazne vrijednosti na temelju odabranih ulaznih vrijednosti.

4.2.1. Definiranje Flask API ruta

Potrebno je definirati REST API rute koje omogućuju komunikaciju između korisničkog sučelja (frontend-a) i pozadinske logike za izvođenje operacija dubokog učenja.

REST API (engl. *Representational State Transfer Application Programming Interface*) je arhitektura koja se koristi za kreiranje povezanih aplikacija. Omogućuje različitim softverskim sustavima komunikaciju putem interneta, definiranjem skupa pravila i konvencija o tome kako se resursima može pristupiti i manipulirati pomoću standardnih HTTP metoda. REST API metode su:

- GET: Dohvaćanje podataka s poslužitelja.
- POST: Kreiranje podataka na poslužitelju.
- PUT: Promjena postojećeg resursa na poslužitelju.
- DELETE: Uklanjanje resursa s poslužitelja.

Glavna je svrha razdvajanje funkcija klijenta i poslužitelja, dopuštajući im da se samostalno razvijaju uz jasno strukturiran i definiran oblik komunikacije [33].

Na Flask serveru, definirat će se tri ključne rute:

1. Trening modela (POST ruta) - Prima podatke o CSV datoteci, ciljnoj varijabli te hiper parametrima poput broja i veličine skrivenih slojeva, broja epoha, stope učenja i drugih. Ovi se podaci šalju putem HTTP zahtjeva i koriste za iniciranje procesa obuke modela.
2. Izvještavanje o napretku procesa treninga (GET ruta) - Korisnici se povezuju na ovu rutu kako bi dobivali povratne informacije o trenutnim performansama modela tijekom procesa treninga.
3. Predikcija izlazne vrijednosti (POST ruta) - Ova ruta prima podatke o modelu i ulaznim vrijednostima u JSON formatu putem HTTP zahtjeva, a koristi ih za predikciju izlaznih vrijednosti.

Prvo se uvode potrebne biblioteke i klase. Za definiranje ruta i postavljanje servera koristi se Flask, dok se biblioteka CORS implementira za omogućavanje dijeljenja sadržaja između različitih domena (engl. *Cross-Origin Resource Sharing*). Bez nje, aplikacija bi blokirala zahtjeve s frontend strane.

```
from flask import Flask, request, Response
from flask_cors import CORS

app = Flask(__name__)
CORS(app)
```

Slika 6. Uvođenje biblioteka za Flask

Flask klasa se koristi za kreiranje serverske aplikacije. Varijabla app predstavlja instancu Flask klase, kojoj se kao argument prosljeđuje ime trenutnog modula (__name__).

```
if __name__ == "__main__":
    app.run(port=5000 , debug=False)
```

Slika 7. Pokretanje Flask servera

Unutar uvjetnog bloka poziva se metoda run varijable app, čime se osigurava da se Flask server pokreće samo ako se skripta izvršava izravno, a ne kada se uvozi kao modul. Uz to, metodi se prosljeđuju dva parametra konfiguracije, onemogućavanje razvojnih značajki za otklanjanje pogrešaka, te postavljanje aplikacije da radi na portu 5000, koji će služiti za komunikaciju s frontend-om.

```
progress_queue = queue.Queue()
```

Slika 8. Korištenje Python Queue modula

Varijabla progress_queue kreira se pomoću ugrađenog Python-ovog modula za rad s redovima, temeljenog na FIFO principu (*First-In-First-Out*). To znači da se elementi dodaju i uklanjaju iz reda prema redoslijedu kojim su uneseni, pri čemu prvi uneseni element prvi izlazi [34]. Ova varijabla igra ključnu ulogu u praćenju napretka tijekom obuke modela, jer pohranjuje podatke o trenutnom stanju modela, koji se potom prosljeđuju korisniku čim se poveže na odgovarajuću rutu.

```python
@app.post("/api/train-model")
def train_model():
    try:
        params = parse_train_request_params(request)
        train_model_logic(params, progress_queue)
    except Exception as e:
        progress_queue.put(json.dumps({"status": "error", "error": str(e)}))
        return Response('error', status=500)

    return Response('success', status=200)

@app.get("/api/train-progress")
def train_progress():
    def generate():
        while True:
            try:
                progress = progress_queue.get()
                yield f"data: {progress}\n\n"
                if '"status": "complete"' in progress:
                    break
            except queue.Empty:
                yield f"data: {'status': 'waiting'}...\n\n"

    return Response(generate(), mimetype='text/event-stream')

@app.post("/api/predict")
def predict():
    try:
        params = parse_predict_request_params(request)
        result = predict_logic(params)
    except Exception as e:
        return Response(f'error: {str(e)}', status=500)

    return Response(json.dumps({'prediction': result}), status=200)
```

Slika 9. Definiranje Flask ruta

Svaka ruta koristi metode POST ili GET varijable app, kojima se specificira URL ruta i odgovarajuća funkcija koja će biti pokrenuta prilikom pristupa. Sve rute sadrže prefiks "api/", što je česta konvencija za lakšu organizaciju i odvajanje API ruta.

POST rute za treniranje i predikciju modela su relativno slične, obje primaju podatke iz HTTP zahtjeva, obrađuju ih i izvršavaju logiku unutar try/except bloka za rukovanje pogreškama. GET ruta za praćenje napretka treninga prijenos podataka o napretku treninga u stvarnom

vremenu putem EventSource API-a, koji omogućuje slanje događaja od strane poslužitelja prema klijentu.

4.2.2. Kreiranje logike za trening modela u PyTorchu

4.2.2.1. Engine

```
def engine(file,
           label_index,
           iterations,
           learning_rate,
           activation_function,
           progress_queue,
           hidden_layers,
           normalization,
           train_ratio,
           dropout,
           random_seed):
```

Slika 10. Modul za inicijalizaciju i trening modela dubokog učenja

Glavni modul za kreiranje i trening modela dubokog učenja naziva se engine. Taj je modul zapravo funkcija koja kao parametre prima vrijednosti:

- file - CSV datoteka koja sadrži podatke, ti se podaci dalje dijele na trening i test skup podataka.
- label_index - Vrijednost indeksa na kojoj se nalazi ciljna varijabla.
- iterations - Broj iteracija odnosno epoha izvođenja treninga.
- learning_rate - Stopa učenja modela tijekom treninga.
- activation_function - Aktivacijska funkcija čvora koja računa izlaz na temelju njegovih pojedinačnih ulaza i težina, pomaže kod rješavanja složenijih problema koji se ne mogu opisati linearnim odnosima.
- progress_queue - Referenca na prethodno definiranu varijablu reda čekanja, koristi se kako bi se tijekom učenja u varijablu mogle dodati vrijednosti koje pokazuju stanje modela tijekom treninga.

- hidden_layers - Varijabla koja je zapravo niz koji definira broj i veličine skrivenih slojeva neuronske mreže.
- normalization - Booleova vrijednost koja pokazuje treba se koristiti normalizacija slojeva.
- train_ratio - Pokazuje vrijednost omjera između trening i test skupa podataka.
- dropout - Omjer neurona koji će se ignorirati tijekom treninga, pomaže kod regularizacije i overfitting-a.
- random_seed - Može biti nepostojeća vrijednost ili brojka koja se koristi za generiranje identičnih podataka, čime se omogućava reproducibilnost.

```
labels, data, data_by_labels = csv_parser(label_index=label_index, file=file)
X_tr, X_te, y_tr, y_te = to_split_tensor_data(labels, data, label_index, train_ratio, random_seed)
X_tr = X_tr.to(device)
X_te = X_te.to(device)
y_tr = y_tr.to(device)
y_te = y_te.to(device)
```

Slika 11. Obrada podataka

Primljena CSV datoteka i indeks ciljne varijable prosljeđuju se funkciji za raščlanjivanje i obradu podataka. Ta funkcija vraća ulazne i izlazne vrijednosti te pomoćnu varijablu koja specificira tipove podataka u svakom stupcu.

Ciljne vrijednosti, zajedno s ulaznim podacima, indeksom ciljne varijable, omjerom trening i test skupa te brojem za reproducibilnost, predaju se funkciji to_split_tensor_data. Ova funkcija proširuje postojeću funkciju iz biblioteke scikit-learn za podjelu podataka na trening i test skupove, osiguravajući njihovo nasumično miješanje i pravilnu podjelu. Kao rezultat, funkcija vraća podijeljene ciljne i ulazne vrijednosti za trening i test skupove.

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
```

Slika 12. Provjera dostupnost CUDA sučelja

Tenzori trening i test podataka prenose se na specificirani hardver pomoću metode .to(device), gdje je device prethodno definirana varijabla koja ukazuje na podržani tip hardvera. Ako je dostupno CUDA sučelje, računske operacije izvršavat će se na grafičkom procesoru (GPU), dok će se u suprotnom koristiti centralni procesor (CPU).

```
if (random_seed):
    torch.manual_seed(random_seed)
```

Slika 13. Omogućavanje reproducibilnosti

PyTorch metoda za reproducibilnost inicijalizira se samo ako je varijabla definirana, osiguravajući konzistentnost u svim narednim koracima funkcije.

```
model = Model(input_size=len(X_tr[0]),
              output_size=len(labels),
              activation_function=activation_function,
              hidden_layers=hidden_layers,
              normalization=normalization,
              dropout=dropout).to(device)
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

Slika 14. Kreiranja modela, funkcije gubitka i optimizacijskog algoritma

Nakon toga, kreira se model neuronske mreže kojem se kao parametri prosljeđuju veličine ulaznog, izlaznog i skrivenih slojeva, kao i aktivacijska funkcija, normalizacija te stopa isključivanja neurona.

Za izračun gubitka neuronske mreže koristi se funkcija unakrsne entropije (engl. *Cross Entropy Loss Function*). Unakrsna entropija, poznata i kao logaritamski gubitak, ključna je funkcija u klasifikacijskim problemima strojnog učenja jer kvantificira razliku između predviđenih vjerojatnosti modela i stvarnih oznaka klasa podataka [35].

Za optimizaciju parametara koristi se algoritam ADAM (engl. *Adaptive Moment Estimation*), koji kombinira prednosti tehnika Momentum i RMSProp. ADAM je vrlo učinkovit pri radu s velikim skupovima podataka i složenim modelima, budući da dinamički prilagođava stopu učenja tijekom procesa treninga [36]. Ovaj optimizacijski algoritam dostupan je kroz PyTorch modul za optimizaciju, te kao argumente funkcije prima parametre modela neuronske mreže i željenu stopu učenja.

```
train_loop(model, X_tr, y_tr, X_te, y_te, loss_fn, optimizer, iterations, progress_queue, labels, device)
total_params = sum(p.numel() for p in model.parameters())

state_dict = model.state_dict()
json_state_dict = json.dumps(state_dict, cls=TensorEncoder)
```

Slika 15. Inicijalizacije petlje treninga i pohrana konačnog modela

Nakon definiranja klase modela, funkciji train_loop, koja upravlja procesom obuke modela, prosljeđuju se sljedeći argumenti: referenca na model, trening i test skupovi podataka, optimizator, funkcija gubitka, referenca na varijablu reda čekanja, broj iteracija te tip hardvera na kojem se računske operacije izvršavaju.

Osim glavne funkcionalnosti, u modulu se kreiraju pomoćne varijable koje sadrže informacije o modelu kako bi ga bilo moguće pohraniti za kasniju upotrebu. Proces obuke obuhvaćen je blokom try/catch kako bi se osigurala robusnost i omogućilo hvatanje eventualnih pogrešaka u izvođenju koda.

4.2.2.2. Model klasa

```
from torch import nn

You, 2 days ago | 1 author (You)
class Model(nn.Module):
    def __init__(self, input_size, output_size, activation_function, hidden_layers, normalization, dropout):
        super().__init__()
        self.activation = self.get_activation(activation_function)
        self.layer_stack = self.create_layers(input_size, output_size, hidden_layers, normalization, dropout)
```

Slika 16. Klasa modela dubokog učenja

Klasa korištena za inicijalizaciju modela dubokog učenja naziva se Model i nasljeđuje funkcionalnost osnovne klase Module iz PyTorch-ovog modula nn. Klasa Module predstavlja temeljni građevni element za konstrukciju neuronskih mreža u PyTorchu. Sve prilagođene mreže u PyTorchu nasljeđuju ovu klasu, čime se omogućuje jednostavna implementacija i organizacija višeslojnih neuronskih mreža [37].

Prilikom inicijalizacije klase Model, automatski se poziva metoda __init__, koja predstavlja konstruktor metodu u Pythonu. Ova metoda inicijalizira klasu i osigurava da se izvrši konstruktor nadređene klase pomoću metode super().__init__, koja inicijalizira parametre neuronske mreže. Nakon toga, na kreirani objekt se dodjeljuju aktivacijska funkcija i slojevi mreže.

```
def get_activation(self, activation_function):
    if activation_function == 'relu':
        return nn.ReLU()
    elif activation_function == 'sigmoid':
        return nn.Sigmoid()
    elif activation_function == 'tanh':
        return nn.Tanh()
    elif activation_function == 'linear':
        return nn.Identity()
```

Slika 17. Odabir aktivacijske funkcije

Klasa Model dizajnirana je s visokim stupnjem prilagodljivosti. Prihvaća argumente koji definiraju arhitekturu i funkcionalnosti neuronske mreže, poput parametra activation_function što je zapravo znakovni niz (engl. *string*) koji sadrži naziv odabrane aktivacijske funkcije, te omogućuje odabir funkcije koja se pridodaje na izlaz svakog sloja. Podržane aktivacijske funkcije su ReLU, Tanh, Sigmoid i Linear.

```
def create_layers(self, input_size, output_size, hidden_layers, normalization, dropout):
    layers = []
    prev_size = input_size
    for size in hidden_layers:
        layers.append(nn.Linear(in_features=prev_size, out_features=size))
        layers.append(self.activation)
        if normalization:
            layers.append(nn.LayerNorm(size))
        layers.append(nn.Dropout(p=dropout))
        prev_size = size
    layers.append(nn.Linear(in_features=prev_size, out_features=output_size))
    return nn.Sequential(*layers)
```

Slika 18. Kreiranje slojeva neuronske mreže

Metoda create_layers omogućuje modularnu izradu arhitekture mreže. Za svaku vrijednost u nizu hidden_layers dodaje se skriveni sloj s definiranim brojem neurona, odgovarajućom aktivacijskom funkcijom, slojem za normalizaciju (ako je omogućen) i slojem za isključivanje neurona s definiranim postotkom isključivanja. Budući da se radi o klasifikacijskom problemu, na kraju mreže dodaje se izlazni sloj čiji broj neurona odgovara broju jedinstvenih klasa ciljne varijable.

U konačnici se koristi PyTorch-ova metoda nn.Sequential, koja stvara sekvencijalni spremnik slojeva. Ovaj spremnik osigurava da se svi definirani slojevi mreže izvršavaju točnim redoslijedom.

Vraćanjem nn.Sequential(*layers), funkcija generira i vraća sekvencijalni model koji objedinjuje sve definirane slojeve. Ovaj modularni pristup omogućuje jednostavno eksperimentiranje s različitim konfiguracijama i kombinacijama slojeva, čineći proces razvoja mreže fleksibilnim i intuitivnim.

4.2.2.3. Trening petlja

```
from torchmetrics import Accuracy, F1Score, ConfusionMatrix
import json
import torch

def train_loop(model, X_tr, y_tr, X_te, y_te, loss_fn, optimizer, iterations, progress_queue, labels, device):
    accuracy = Accuracy(task='multiclass', num_classes=len(labels)).to(device)
    f1score = F1Score(task="multiclass", num_classes=len(labels)).to(device)
    confmat = ConfusionMatrix(task="multiclass", num_classes=len(labels)).to(device)
```

Slika 19. Petlja treninga neuronske mreže

Modul train_loop odgovoran je za izvođenje petlji kojima se optimiziraju parametri neuronske mreže. Na početku se kreiraju pomoćne varijable za pohranu ključnih mjernih metrika, poput preciznosti, F1 rezultata i matrice konfuzije.

- Preciznost označava omjer ispravno predviđenih ciljnih vrijednosti u odnosu na ukupni broj predikcija.
- F1 rezultat predstavlja harmonijsku sredinu između preciznosti i odziva, s posebnim fokusom na pozitivnu klasu, što ga čini korisnim u situacijama s neuravnoteženim skupovima podataka [38].
- Matrica konfuzije koristi se za procjenu performansi modela. To je matrica dimenzija N x N gdje N označava broj jedinstvenih ciljnih klasa, a svrha joj je usporedba stvarnih vrijednosti s onima predviđenima od strane modela.

Ove se pomoćne funkcije uvode iz pomoćne PyTorch biblioteke koja se naziva TorchMetrics, kako bi se olakšalo precizno mjerenje performansi modela tijekom treninga. Također ih je zbog kompatibilnosti potrebno prebaciti na tip hardvera na kojemu se nalaze tenzori.

Te pomoćne funkcije implementirane su pomoću PyTorch biblioteke TorchMetrics, koja omogućuje precizno i intuitivno mjerenje performansi modela tijekom treninga. Kako bi bile

kompatibilne s tenzorima, sve funkcije potrebno je prebaciti na istu vrstu hardvera na kojem se model izvodi (npr. CPU ili GPU).

```
for i in range(iterations + 1):
    model.train()
    logits = model(X_tr)
    logits_pred = torch.softmax(logits, dim=1).argmax(dim=1)
    acc = accuracy(logits_pred, y_tr).item()*100
```

Slika 20. Predviđanje ciljnih vrijednosti unutar trening petlje

Sljedeće se pokreće glavna petlja treninga, koja se ponavlja onoliko puta koliko je korisnik definirao kroz broj iteracija. Na početku svake iteracije model se stavlja u trening način rada, čime se omogućuje računanje gradijenata i ažuriranje parametara.

Ulazni trening podaci (X_tr) prosljeđuju se modelu, koji generira nenormalizirane izlazne vrijednosti (logits) za svaku klasu ciljne varijable.

Kako bi se ti rezultati pretvorili u vjerojatnosti koje se lakše interpretiraju, koristi se funkcija softmax. Funkcija normalizira vrijednosti duž specificirane dimenzije (dim=1), te se dobivaju izlazne vjerojatosti za svaku klasu ciljne varijable.

Nakon toga, funkcija argmax identificira klasu s najvećom vjerojatnošću za svaki uzorak, vraćajući indekse najviših vrijednosti. Ove predikcije spremaju se u varijablu logits_pred. Na kraju, koristeći stvarne i predviđene vrijednosti, računa se preciznost modela (acc), koja se pretvara u postotak za lakšu interpretaciju.

```
loss = loss_fn(logits, y_tr)
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

Slika 21. Optimizacija modela

Funkcijom backward izvodi se povratna propagacija, gdje se izračunava gradijent gubitka za svaki parametar modela neuronske mreže, vrijednosti tih gradijenata se zatim koriste u optimizacijskoj funkciji koja ažurira sve parametre kako bi se smanjio gubitak.

U procesu optimizacije prvo se koristi funkcija gubitka, koja kvantificira odstupanje između predviđenih vrijednosti modela i stvarnih ciljnih vrijednosti. Funkciji gubitka (loss_fn) prosljeđuju se nenormalizirane vrijednosti (logits) i stvarne ciljne vrijednosti (y_tr), a rezultat je mjera pogreške koju model treba minimizirati.

Prije pokretanja povratne propagacije, potrebno je resetirati gradijente svih parametara modela na nulu pomoću funkcije zero_grad. Ovaj korak osigurava da gradijenti iz prethodnih iteracija ne utječu na ažuriranja u trenutnoj iteraciji, jer bi se inače akumulirani tijekom propagacije.

Sljedeći korak je pozivanje funkcije backward, koja izvodi povratnu propagaciju i izračunava gradijente funkcije gubitka za svaki parametar neuronske mreže. Ti gradijenti se zatim koriste u optimizacijskom algoritmu, koji ažurira vrijednosti parametara modela kako bi smanjio funkciju gubitka.

```
model.eval()
if (i) % (iterations / 20) == 0:
    with torch.inference_mode():
        test_logits = model(X_te)
        test_logits_pred = torch.softmax(test_logits, dim=1).argmax(dim=1)
        test_loss = loss_fn(test_logits, y_te)
        test_acc = accuracy(test_logits_pred, y_te).item()*100
        f1 = f1score(test_logits_pred, y_te)
        confusion_matrix = confmat(test_logits_pred, y_te)

    training_data = {
        "status": "training",
        "iteration": str(i),
        "trainLoss": loss.item(),
        "trainAccuracy": acc,
        "testLoss": test_loss.item(),
        "testAccuracy": test_acc,
        "f1Score": f1.item(),
        "confusionMatrix": confusion_matrix.tolist()
    }
    progress_queue.put(json.dumps(training_data))
```

Slika 22. Računanje performansi na kraju trening petlje

Model se stavlja u način rada evaluacije pomoću funkcije eval. Ovaj način rada onemogućava računanje gradijenata i poboljšava učinkovitost prilikom izvođenja procesa predviđanja.

Evaluacija se izvodi na testnom skupu podataka kako bi se procijenile performanse modela i njegova mogućost generalizacije. Računaju se metrike preciznosti, F1 rezultata i matrice konfuzije. Evaluacijski blok se pokreće periodično nakon svake dvadesete iteracije, kako bi se smanjila učestalost računanja i poboljšala učinkovitost.

Rezultati evaluacije se pohranjuju kao objekt, koji se zatim pretvara u JSON format radi lakšeg prijenosa i kompatibilnosti s frontendom. Ovi podaci šalju se u red čekanja, što omogućuje korisniku njihovo čitanje i vizualizaciju na korisničkom sučelju.

4.2.3. Kreiranje logike za predviđanje izlazne varijable u PyTorch-u

```
import torch
from src.nn.Model import Model
from src.nn.helpers import turn_json_to_torch

def predict_logic(params):

    model = Model(input_size=params['input_size'],
                  output_size=params['output_size'],
                  activation_function=params['activation_function'],
                  hidden_layers=params['hidden_layers'],
                  normalization=params['normalization'])

    state_dict = turn_json_to_torch(params['model_raw'])
    model.load_state_dict(state_dict)
    input = torch.tensor(params['inputsRaw']).float()
    result = model(input).argmax(0).item()
    return result
```

Slika 23. Modul za predviđanje ciljne vrijednosti

Modul za predviđanje ciljne vrijednosti implementira logiku koja omogućuje korisniku predikciju na temelju unaprijed istreniranog modela. Modul prima parametre putem HTTP zahtjeva, koje prosljeđuje Flask web poslužitelj. Na temelju tih parametara, inicijalizira se neuronski model s odgovarajućim postavkama arhitekture

Parametri modela (težine i pomaci) dolaze u JSON formatu unutar HTTP zahtjeva i potrebno ih je pretvoriti u PyTorch tenzore pomoću funkcije turn_json_to_torch. Tako dobivene

vrijednosti unose se u state_dict, a zatim učitavaju u model pomoću metode load_state_dict, čime se osigurava da model ima iste parametre kao i prilikom treninga.

Ulazni podaci za predikciju također se transformiraju u PyTorch tenzore, kako bi bili kompatibilni s modelom. Ti podaci prosljeđuju se modelu, koji generira izlaznu vrijednost. Rezultat predikcije pohranjuje se u varijablu result i vraća kao odgovor korisniku.

4.2.4. Postavljane virtualne okoline i upravljanje uvezenim bibliotekama

Kako bi razvoj aplikacije bio strukturiran i održiv, kreirano je virtualno Python okruženje korištenjem ugrađene funkcionalnosti venv. Virtualno okruženje omogućava izolaciju potrebnih biblioteka i verzija modula, čime se eliminiraju potencijalni konflikti s globalno instaliranim paketima i olakšava reproducibilnost projekta.

U postavljenom virtualnom okruženju moraju biti instalirane sve potrebne biblioteke za rad aplikacije, uključujući PyTorch za implementaciju neuronskih mreža, Flask za izradu web servera te TorchMetrics za mjerenje performansi modela. Instalirani paketi su automatski popisani i spremljeni u datoteku requirements.txt pomoću naredbe pip freeze > requirements.txt. Ova datoteka omogućava drugim korisnicima jednostavno postavljanje identičnog okruženja pomoću naredbe pip install -r requirements.txt.

4.3. Frontend

U ovom poglavlju naglasak će biti na implementaciji frontend dijela aplikacije, koji omogućuje korisnicima interakciju s platformom za trening i evaluaciju modela dubokog učenja. Cilj frontend-a je pružiti intuitivno i funkcionalno web sučelje koje olakšava korisnicima upravljanje podacima i modelima te omogućuje pregled rezultata evaluacije i predikcija.

Frontend aplikacija izrađena je korištenjem Next.js programskog okvira, koji omogućava brz razvoj, visoke performanse i veliku fleksibilnost u kreiranju interaktivnih i modernih korisničkih sučelja. Korištenje React-a u kombinaciji s TailwindCSS-om osigurava jednostavno dizajniranje prilagodljivih i responzivnih elemenata korisničkog sučelja.

Za kvalitetnu implementaciju aplikacije potrebno je definirati glavne funkcije i plan korisničkog toka koji obuhvaća te funkcionalnosti, te se na tim funkcionalnostima temelje dizajn i razvoj korisničkog sučelja. Aplikacija ima četiri ključne funkcije:

1. Upravljanje i pregled podataka.
2. Treniranje i pohrana modela dubokog učenja.

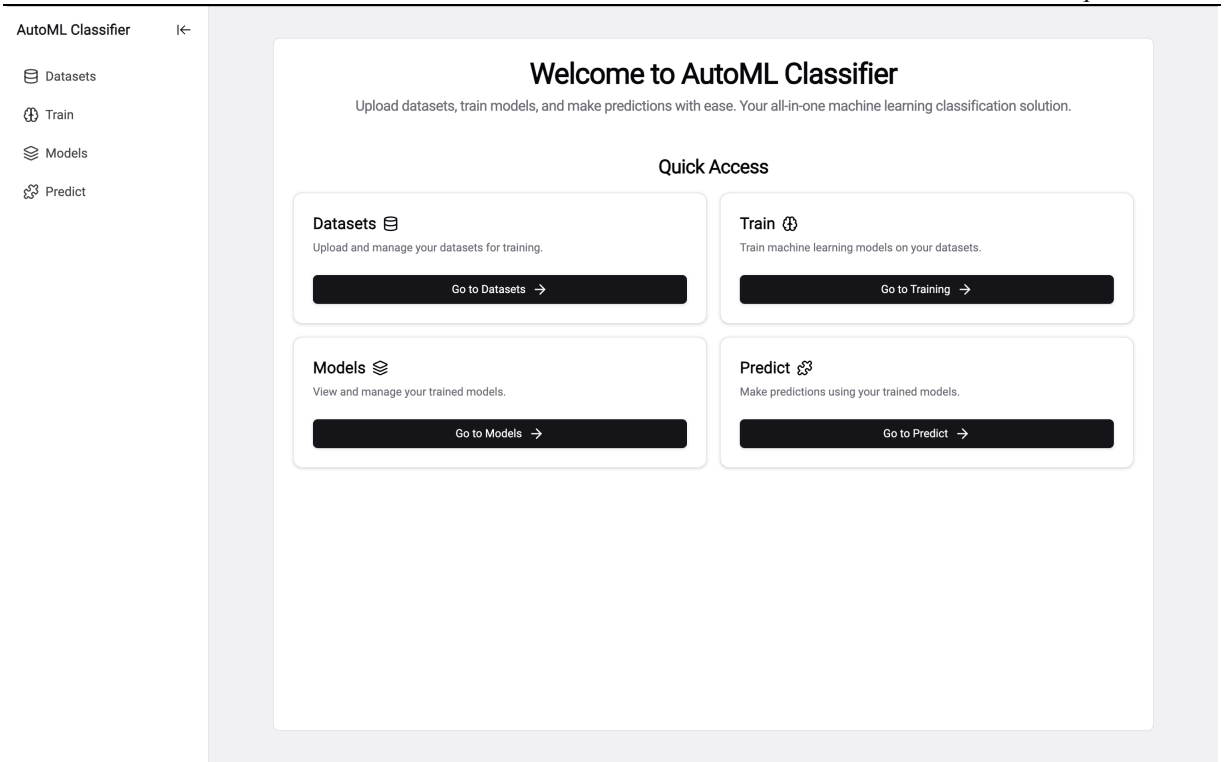
3. Upravljanje i pregled pohranjenih modela.
4. Predikcija ciljne vrijednosti pohranjenog modela na temelju odabranih ulaznih podataka.

Kada su funkcionalnosti jasno definirane, korisnički tok aplikacije osmišljen je prema sljedećim koracima:

1. Korisnik učitava podatke u obliku CSV datoteke.
2. Nakon učitavanja podataka, prikazuje se tablični pregled, gdje korisnik može pregledati i isključiti pojedine stupce prije spremanja podataka.
3. Korisnik ima pregled nad svim spremljenim podacima, s opcijama upravljanja.
4. Na temelju odabranog skupa podataka, korisnik kreira model dubokog učenja.
5. Prilikom kreiranja modela, korisnik odabire ciljnu varijablu i postavlja potrebne hiperparametre.
6. Nakon pokretanja procesa treninga, korisniku se u stvarnom vremenu prikazuje napredak treninga i evaluacija performansi modela.
7. Trenirani model pohranjuje se pod odabranim nazivom.
8. Korisnik ima pregled i mogućnost upravljanja svim pohranjenim modelima.
9. Korisnik odabire model koji želi koristiti za predikciju.
10. Korisnik unosi ulazne podatke i dobiva rezultat predikcije modela.

Tek nakon definiranja ovih koraka se može pristupiti detaljnoj implementaciji korisničkog sučelja i povezivanju s backend-om.

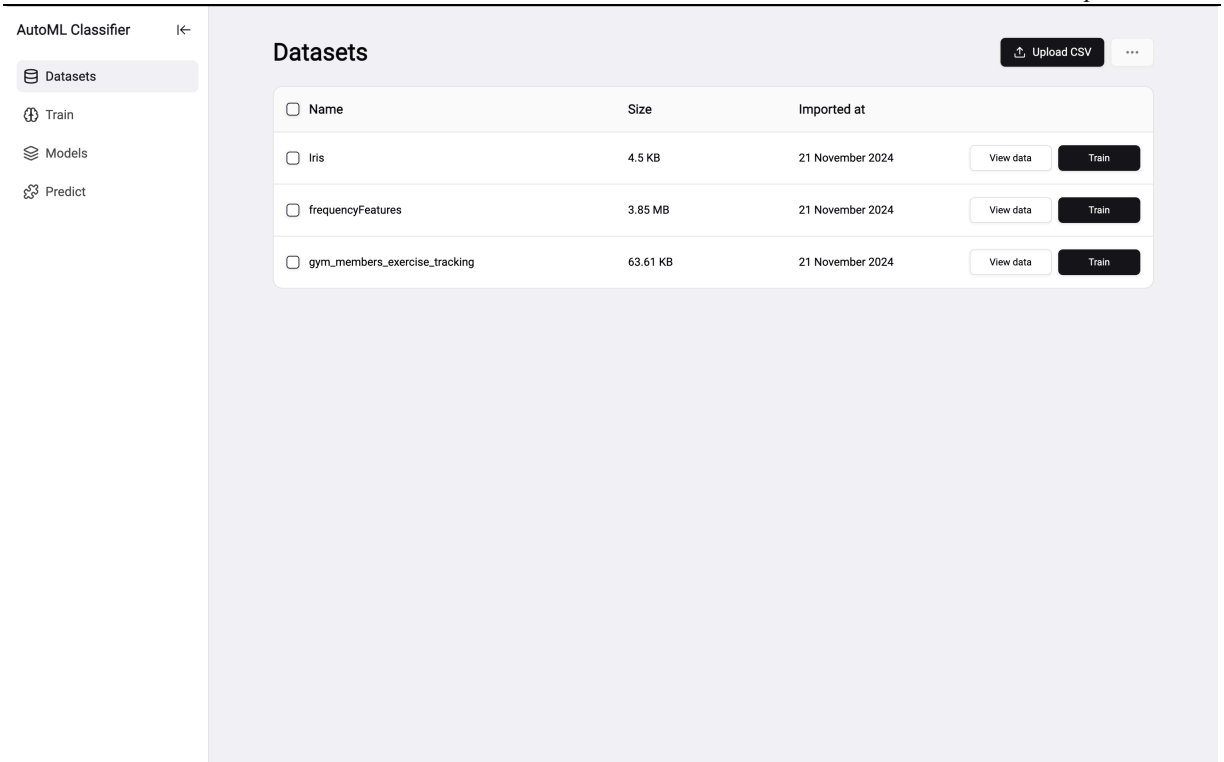
4.3.1. Izvorna stranica i navigacija



Slika 24. Sučelje izvorne stranice i navigacija

Na izvornoj stranici nalazi se naziv aplikacije s kratkim opisom i brzim pristupom svim glavnim funkcionalnostima aplikacije. S lijeve strane se nalazi navigacija koja ustraje među svim rutama aplikacije i omogućava pristup podacima, treniranju modela, pregledu pohranjenih modela i predikciji.

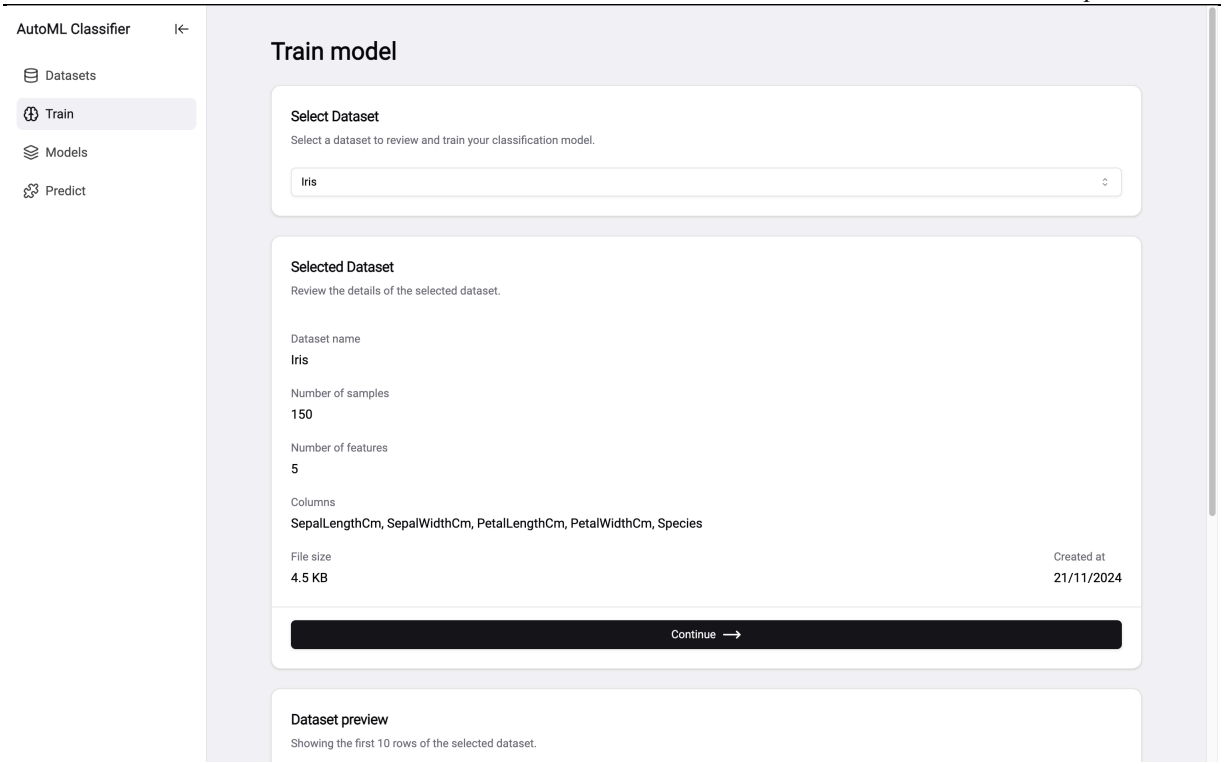
4.3.2. Upravljanje i pregled podataka



Slika 25. Sučelje za upravljanje i pregled podataka

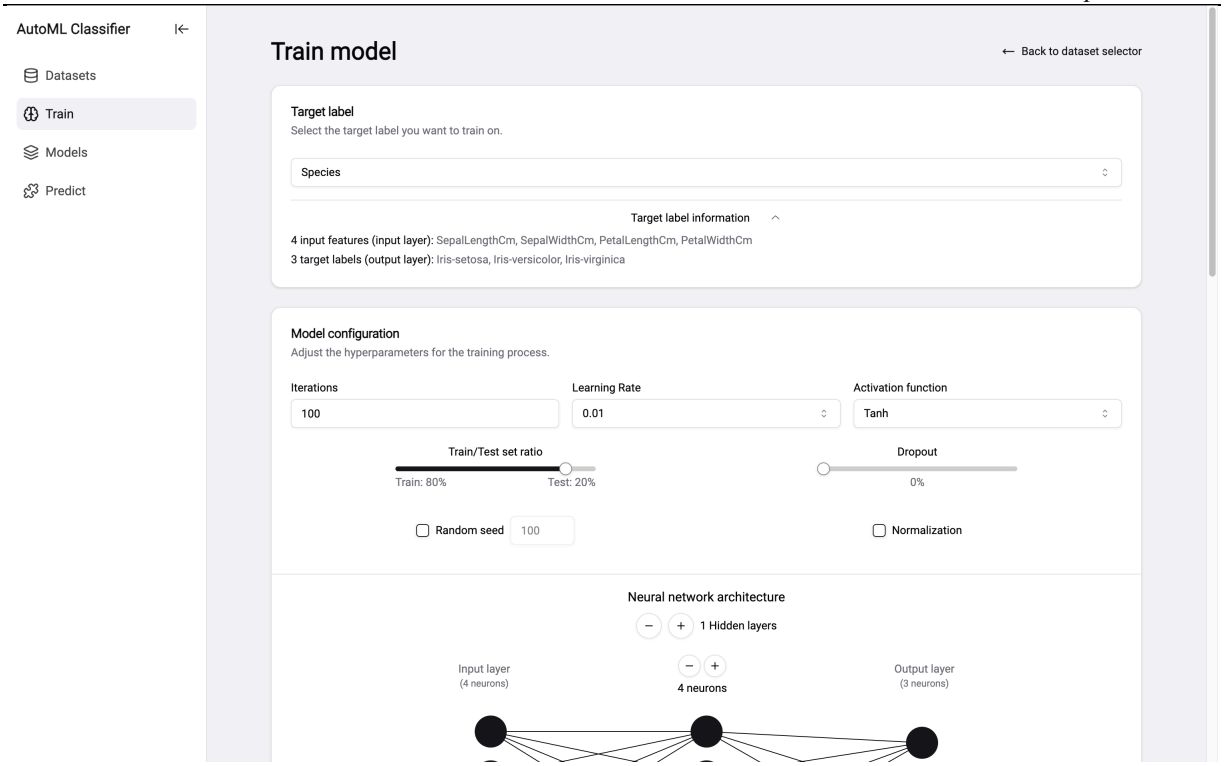
Na stranici skupovi podataka (engl. *datasets*) nalaze se svi pohranjeni skupovi podataka s njihovim imenima, veličinom datoteke i datumom pohrane. Svaki se skup podataka može otvoriti i pregledati u tabličnom obliku, te se pritiskom na gumb train može direktno usmjeriti na trening temeljen na odabranom skupu. Pritiskom na Upload CSV otvara se modul za uvažanje skupa podataka, podaci se također odabirom mogu izbrisati ili skinuti na računalo.

4.3.3. Treniranje modela



Slika 26. Sučelje za odabir podataka za trening modela

Proces treniranja modela započinje na ruti *train model*, gdje korisnik prvo odabire skup podataka koji će se koristiti za kreiranje modela. Odabirom podataka prikazuju se ključne informacije o skupu, uključujući broj uzoraka, broj značajki, imena stupaca, ukupnu veličinu skupa podataka te datum kada su podaci pohranjeni. Ovaj korak omogućava korisniku da ima jasan pregled nad podacima prije nego što započne proces treniranja. Nakon odabira skupa podataka, korisnik se preusmjerava na sučelje za trening modela dubokog učenja.



Slika 27. Sučelje za trening modela i odabir hiper parametara

U sučelju za trening modela potrebno je inicijalno odabrati ciljnu vrijednost modela. Odabirom ciljne vrijednosti ispisuju se ostale ulazne vrijednosti koje predstavljaju ulazne neurone, te sve jedinstvene klase odabrane ciljne vrijednosti koje predstavljaju izlazne neurone.

Dalje se odabire konfiguracija modela gdje se prilagođavaju hiper parametri prema želji korisnika. Moguće je odabrat broj iteracija, stopu učenja, aktivacijsku funkciju, omjer testnog i trening skupa podataka, stopa isključivanja neurona, vrijednost reproducibilnosti, normalizaciju i broj te veličinu pojedinih sakrivenih slojeva mreže.

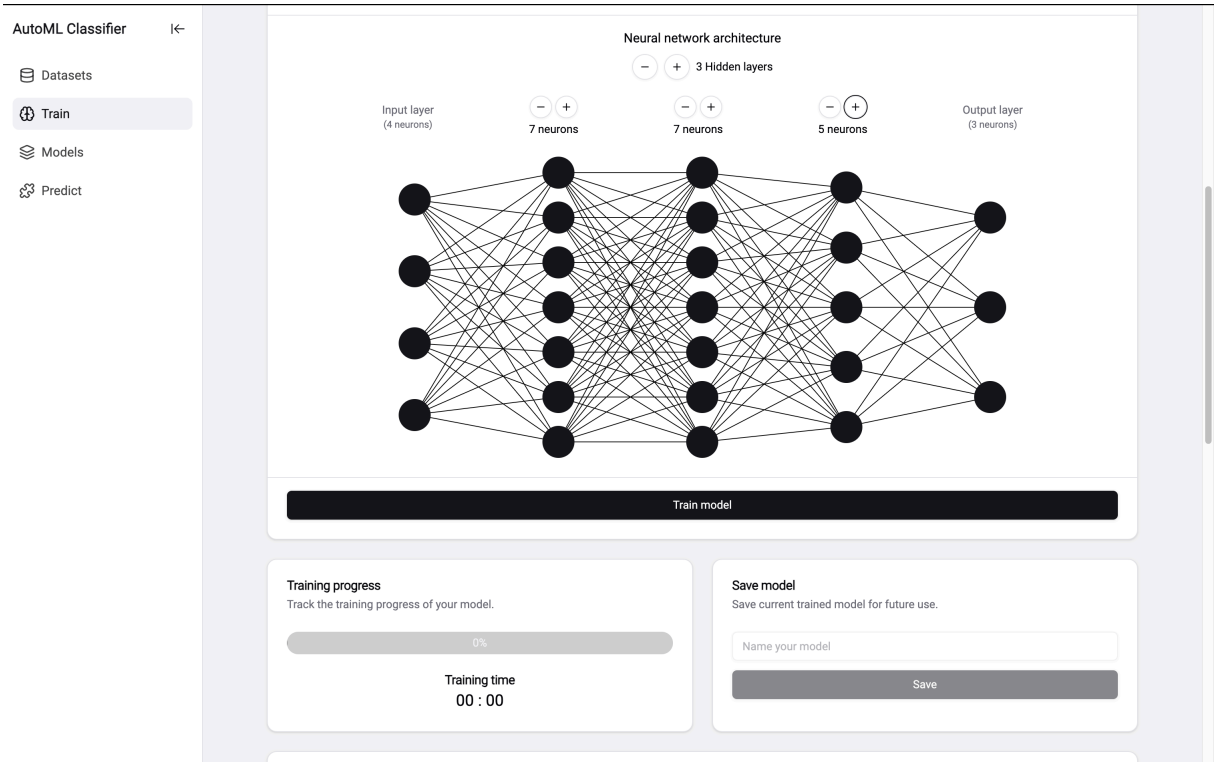
U sučelju za trening modela korisnik prvo odabire ciljnu varijablu, što omogućava aplikaciji da automatski prepozna ulazne značajke koje predstavljaju ulazne neurone modela. Također, ispisuju se sve jedinstvene klase odabrane ciljne varijable koje definiraju izlazne neurone.

Zatim korisnik pristupa konfiguraciji modela, gdje ima mogućnost prilagodbe ključnih hiperparametara. Dostupne opcije su:

- Broj iteracija (epoha).
- Stopa učenja.
- Aktivacijska funkcija.
- Omjer trening i test skupa podataka.

- Stopa isključivanja neurona (engl. *dropout*).
- Vrijednost reproducibilnosti (engl. *random seed*).
- Primjena normalizacije podataka.
- Broj i veličina skrivenih slojeva mreže.

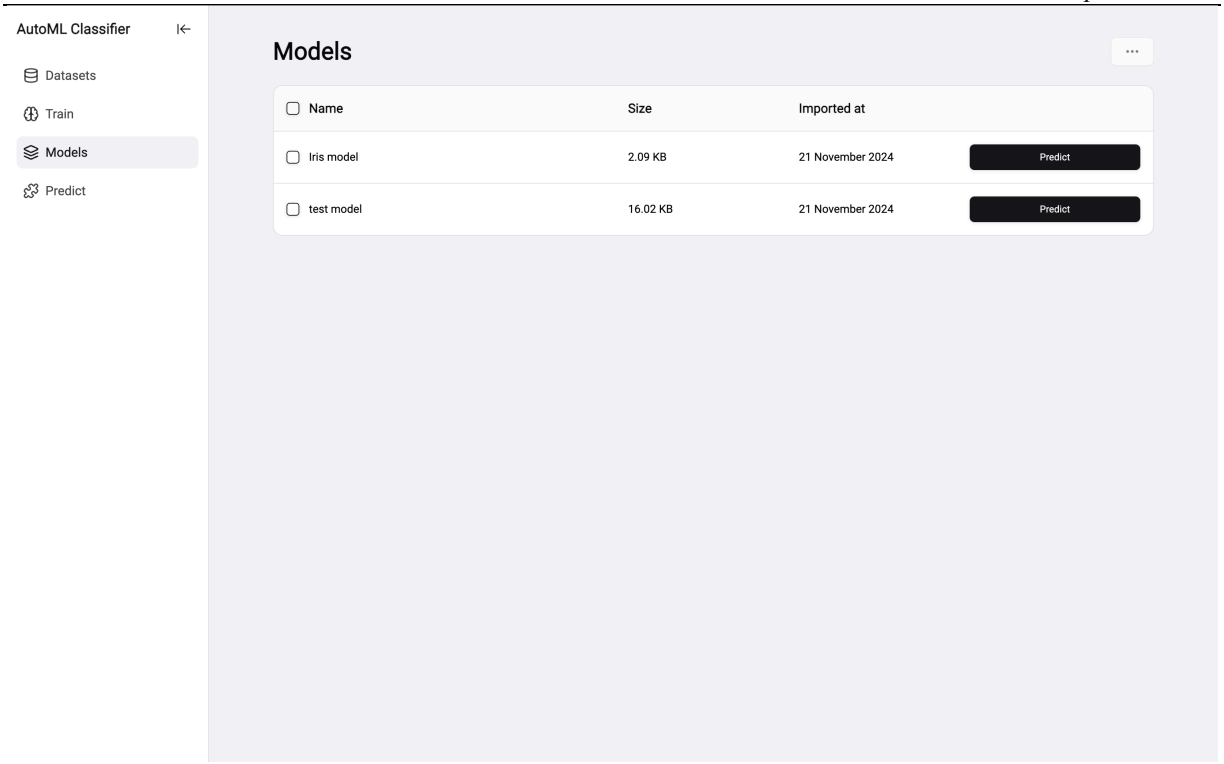
Ovaj dio sučelja omogućava korisniku da oblikuje model prema specifičnim potrebama ili eksperimentalnim ciljevima.



Slika 28. Sučelje za trening modela i kreiranje arhitekture neuronske mreže

Nakon definiranja arhitekture mreže i prilagodbe hiperparametara, pritiskom na gumb *Train model* frontend šalje konfiguracijske podatke backend-u. Backend koristi te informacije za pokretanje procesa treninga modela, a zatim korisniku šalje povratne informacije o napretku i performansama modela tijekom treninga, što korisniku omogućava praćenje i evaluaciju rezultata u stvarnom vremenu.

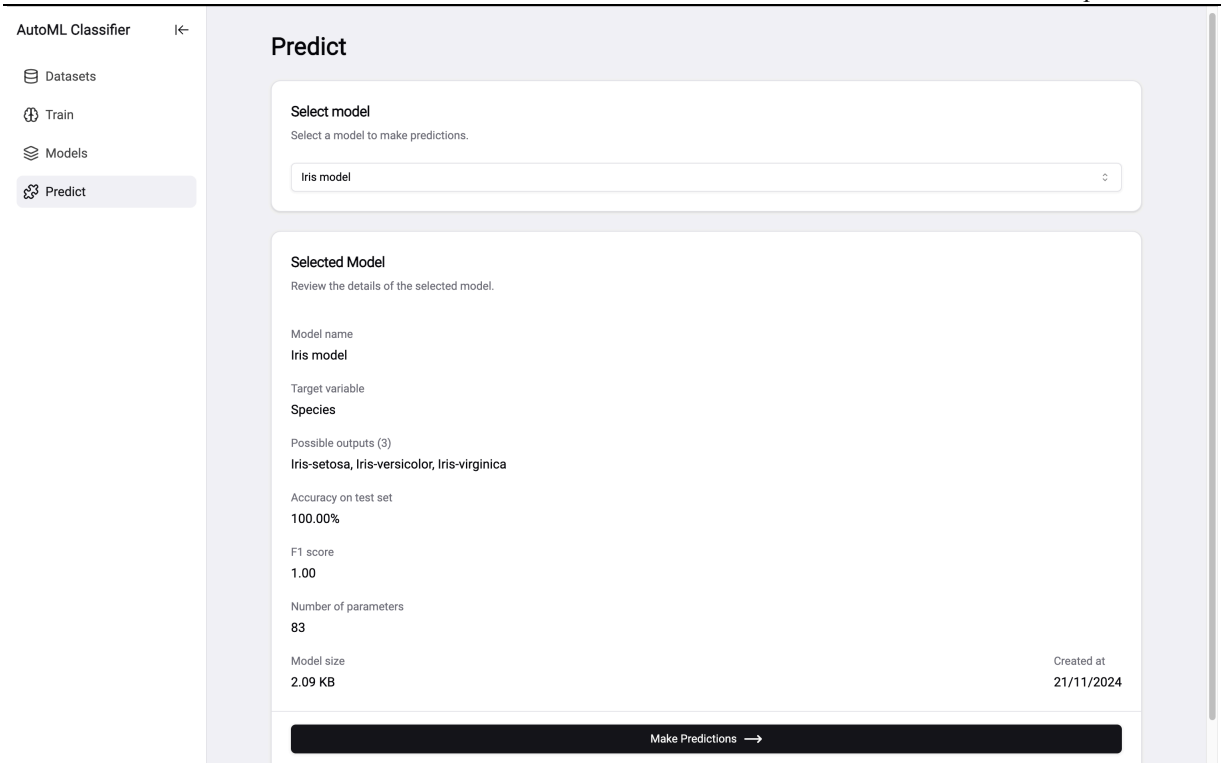
4.3.4. Upravljanje i pregled pohranjenih modela dubokog učenja



Slika 29. Sučelje za upravljanje i pregled pohranjenih modela dubokog učenja

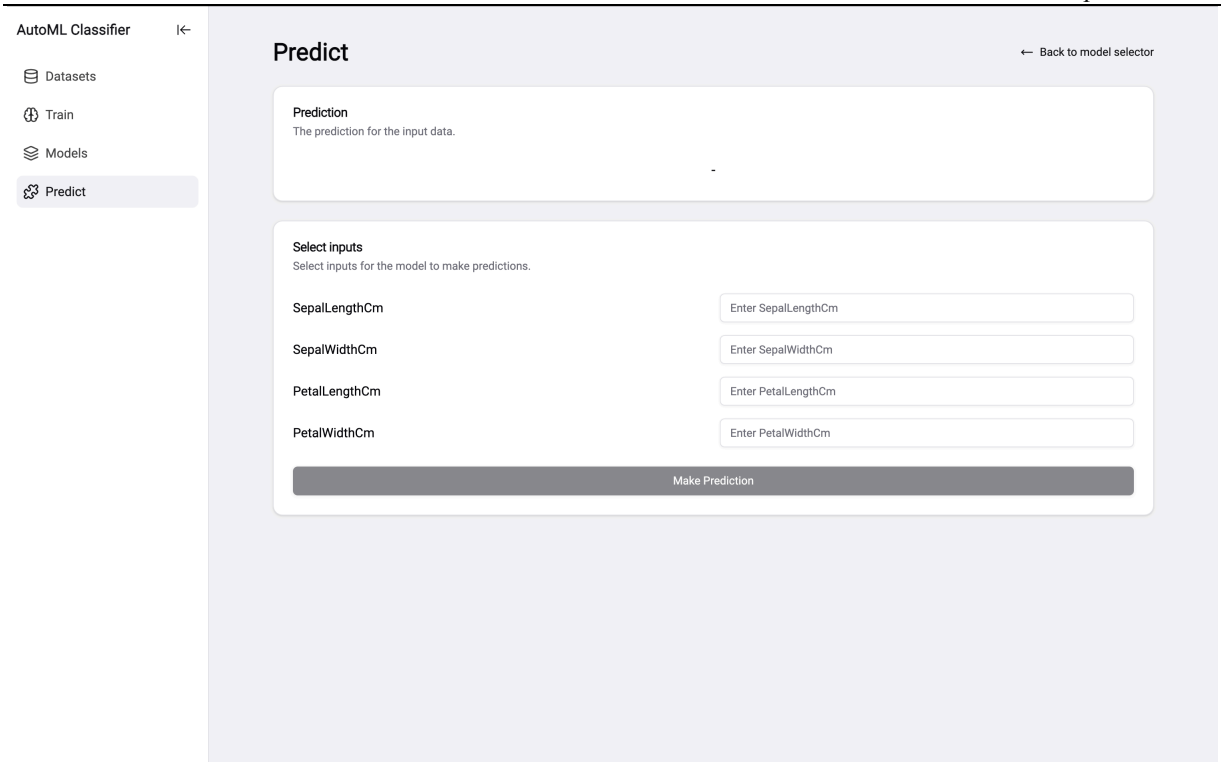
Na sučelju za upravljanje modela se nalaze svi pohranjeni modeli gdje su prikazani njihova imena, veličina datoteke i datum pohrane. Omogućen je odabire i brisanje modela, te se korisnik pritiskom na gumb predict preusmjerava na predikciju uz pomoć odabranog modela.

4.3.5. Predikcija modela



Slika 30. Sučelje za odabir predikcijskog modela

Proces predikcije započinje odabirom već pohranjenog modela. Nakon odabira, korisniku se prikazuju detaljne informacije o modelu, uključujući ime ciljne varijable, moguće rezultate (sve jedinstvene vrijednosti ciljne varijable), preciznost modela na testnom skupu podataka, F1 rezultat, broj parametara neuronske mreže, ukupnu veličinu modela te datum kada je model pohranjen. Ove informacije korisniku omogućuju lakši odabir najprikladnijeg modela za predikciju.



Slika 31. Sučelje za predikciju ciljne vrijednosti

Nakon odabira modela, korisnik prelazi na sučelje za unos ulaznih vrijednosti. Ulazne vrijednosti se prikazuju na temelju njihovog tipa:

- Za brojčane vrijednosti prikazuje se polje koje omogućuje unos isključivo brojeva.
- Za nominalne vrijednosti prikazuje se padajući izbornik s listom svih jedinstvenih vrijednosti atributa.

Ovaj korak osigurava intuitivno iskustvo za korisnika, prilagođeno tipu podataka s kojima radi, čime se minimizira mogućnost pogrešaka pri unosu. Nakon unosa svih potrebnih vrijednosti, korisnik može pokrenuti proces predikcije, a rezultat se prikazuje odmah nakon obrade na backend-u.

5. EVALUACIJA TRENINGA I PREDIKCIJE NA STVARNOM SKUPU PODATAKA

5.1. Odabran skup podataka

Evaluacija funkcionalnosti izrađene aplikacije izvesti će se na poznatom multivarijatnom skupu podataka o cvijetu perunike (engl. *Iris flower dataset*), kojeg je koristio i učinio poznatim britanski biolog i statističar Ronald Fisher [40]. Skup podataka se sastoji od 50 uzoraka svake od tri vrste perunike: Iris setosa, Iris virginica i Iris versicolor.

Za evaluaciju funkcionalnosti izrađene aplikacije korišten je poznati multivarijatni skup podataka o cvijetu perunike (engl. *Iris flower dataset*), koji je popularizirao britanski biolog i statističar Ronald Fisher [40]. Ovaj skup podataka sadrži 50 uzoraka za svaku od tri vrste perunike: *Iris setosa*, *Iris virginica* i *Iris versicolor*.

Svaki uzorak uključuje mjerenja četiri značajke izražene u centimetrima:

- Duljina listova čašice (engl. *sepal length*)
- Širina listova čašice (engl. *sepal width*)
- Duljina latica (engl. *petal length*)
- Širina latica (engl. *petal width*)

Primjer pet uzoraka iz skupa podataka prikazan je u sljedećoj tablici:

Tablica 1. Primjer uzoraka iz skupa podataka o cvijetu perunike

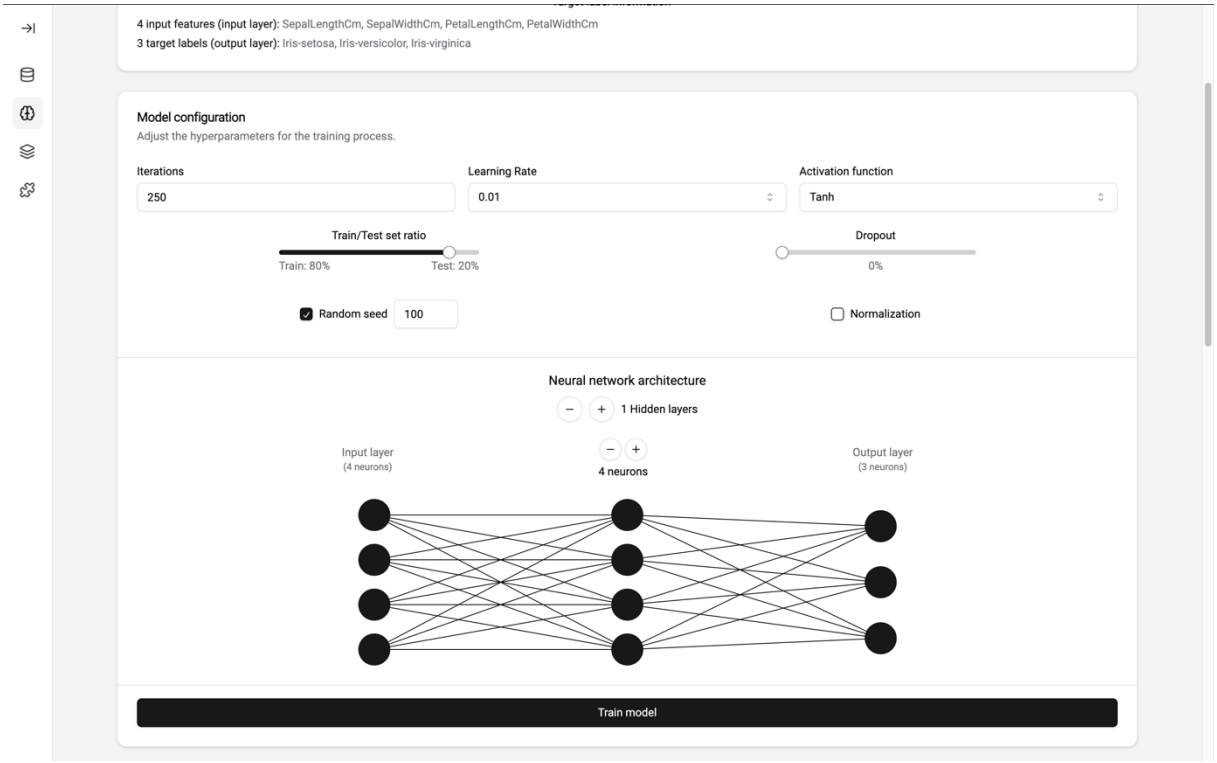
Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	5,1	3,5	1,4	0,2	Iris-setosa
34	5,5	4,2	1,4	0,2	Iris-setosa
70	5,6	2,5	3,9	1,1	Iris-versicolor
111	6,5	3,2	5,1	2	Iris-virginica
148	6,5	3	5,2	2	Iris-virginica

Sljedeće je ovaj skup podataka je uvezen u aplikaciju. Stupac s identifikacijskim oznakama (*Id*) isključen je iz daljnje obrade jer ne sadrži korisne informacije relevantne za klasifikaciju vrsta cvijeta. Ovaj korak osigurava fokus na značajkama ključnim za analizu i trening modela.

5.2. Trening modela

Kao ciljna varijabla odabire se atribut vrste (engl. *Species*), što znači da će ulazne vrijednosti biti duljine i širine listova čašica i latica, a izlazna će vrijednost biti jedna od tri mogućih vrsta: *Iris setosa*, *Iris virginica* ili *Iris versicolor*.

Za ciljni atribut odabrana je varijabla vrsta cvijeta (engl. *Species)*, što znači da su ulazne vrijednosti duljina i širina listova čašica i latica, dok izlazna vrijednost predstavlja jednu od triju mogućih vrsta: *Iris setosa*, *Iris virginica* ili *Iris versicolor*.



Slika 32. Konfiguracija modela cvijeta perunike

S obzirom na to da se radi o relativno malom skupu podataka s ograničenim brojem značajki, nije potrebna složena arhitektura neuronske mreže. Za ovaj model odabrana je sljedeća konfiguracija:

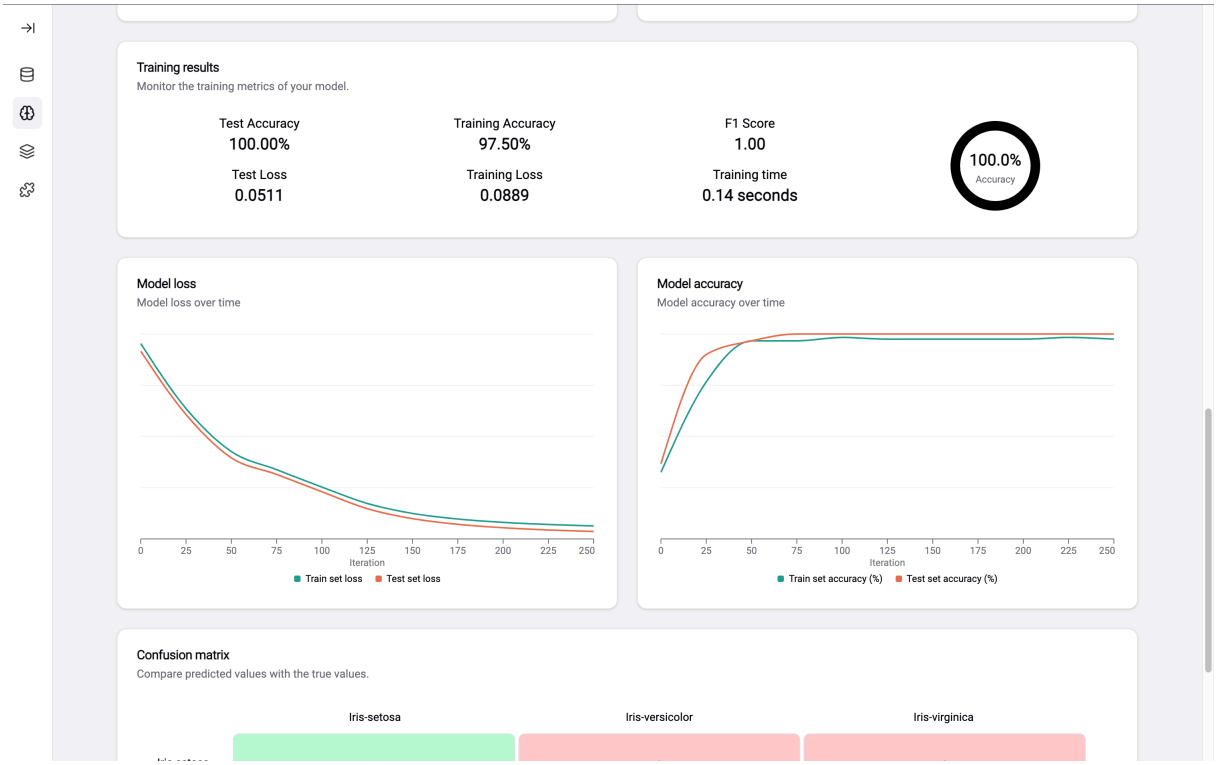
- Broj skrivenih slojeva: 1 (s 4 neurona)
- Broj iteracija: 250
- Stopa učenja: 0.01
- Aktivacijska funkcija: Tanh
- Omjer trening/test skupa podataka: 80/20
- Isključivanje neurona (engl. *dropout*): nije uključeno

- Normalizacija: nije uključena,
- Vrijednost reproducibilnosti (engl. *random seed*): 100

Nakon konfiguracije modela i pokretanja treninga, postupak treninga od 250 iteracija traje svega 0,14 sekundi, a rezultati su sljedeći:

- Preciznost predikcije na testnom skupu podataka: 100%
- Gubitak na testnom skupu: 0.0511
- F1 rezultat: 1.0

Na korisničkom sučelju prikazuju se konačni rezultati u obliku grafova koji prikazuju promjene gubitka i preciznosti kroz iteracije, te konačnu vrijednosti F1 rezultata i vrijeme treniranja.



Slika 33. Rezultati modela cvijeta perunike

Uz osnovne rezultate, generirana je i matrica konfuzije, koja omogućuje vizualnu usporedbu stvarnih vrijednosti s predikcijama modela na testnom skupu podataka. Ova matrica pruža detaljan pregled performansi modela za svaku klasu.

Slika 34. Matrica konfuzije modela cvijeta perunike

5.3. Predikcija modela

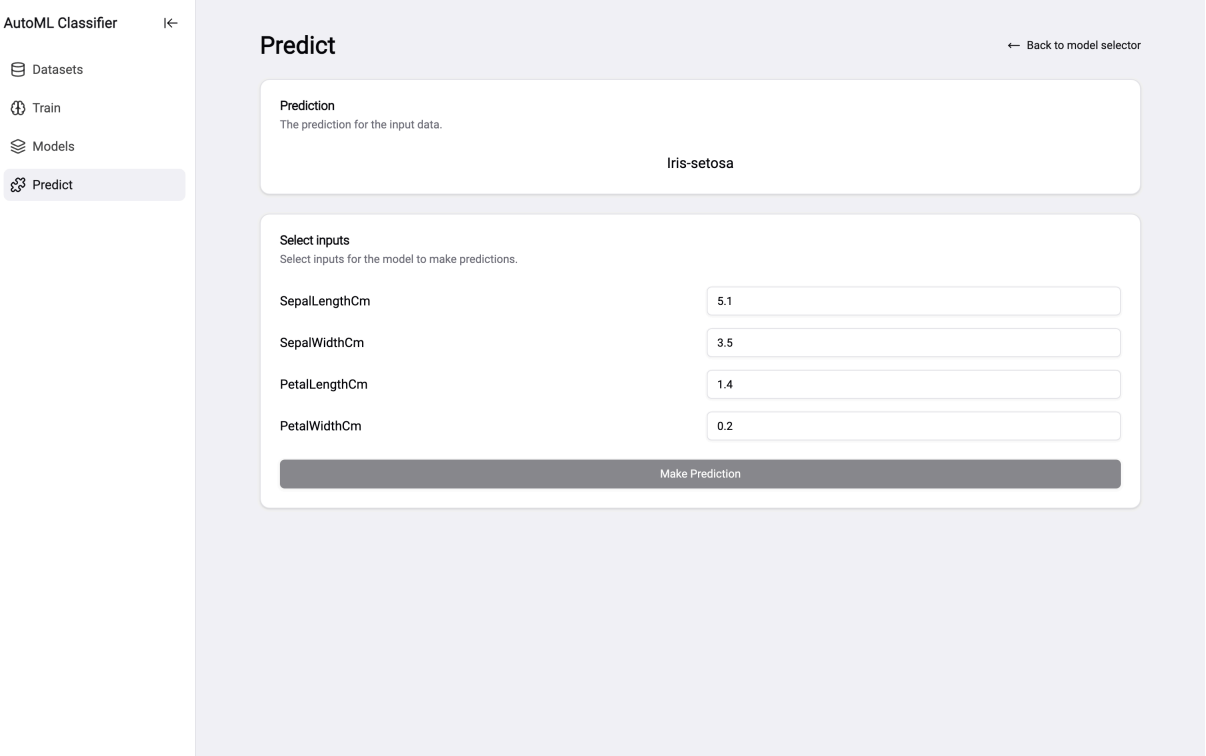
Nakon što model postigne zadovoljavajuće rezultate, omogućuje se njegova pohrana pod odabranim imenom. Na taj način model postaje dostupan za kasniju upotrebu prilikom predikcije. Korisnik može odabrati pohranjeni model, nakon čega mu se prikazuje sučelje za unos vrijednosti atributa i iniciranje predikcije.

Proces predikcije započinje unosom atributa na korisničkom sučelju, dok se obrada predikcije odvija na pozadinskom sustavu (backend), nakon čega se konačni rezultat predikcije vraća i prikazuje korisniku.

Za testiranje funkcionalnosti korišten je uzorak iz skupa podataka cvijeta perunike (uzorak s identifikacijskom oznakom 1). Vrijednosti atributa unesene u sučelje su:

- Duljina listova čašice: 5.1
- Širina listova čašice: 3.5
- Duljina latica: 1.4
- Širina latica: 0.2

Na temelju ovih vrijednosti model je uspješno predvidio vrstu cvijeta kao *Iris-setosa*, što potvrđuje točnost i funkcionalnost razvijenog modela.



Slika 35. Predikcija modela cvijeta perunike

6. ZAKLJUČAK

Razvoj tehnologija umjetne inteligencije i dubokog učenja otvara nove mogućnosti za rješavanje složenih problema u različitim domenama, ali primjena tih tehnologija često zahtijeva napredno tehničko znanje. Cilj ovog rada bio je izraditi web aplikaciju koja pojednostavljuje proces izrade prilagođenih neuronskih mreža, omogućujući korisnicima s ograničenim tehničkim znanjem da treniraju vlastite modele dubokog učenja.

Razvijena aplikacija temelji se na suvremenim tehnologijama, uključujući Next.js za izradu korisničkog sučelja, Flask za poslužiteljsku logiku te PyTorch za implementaciju i treniranje modela dubokog učenja. Korištenjem REST API-ja osigurana je učinkovita komunikacija između frontend-a i backend-a, dok je PyTorch omogućio fleksibilnu implementaciju i trening prilagođenih modela. Aplikacija omogućuje korisnicima učitavanje podataka, odabir arhitekture mreže i postavljanje hiperparametara, a rezultati treniranja i performanse modela vizualiziraju se putem grafova i tablica.

Evaluacija sustava provedena je na skupu podataka *Iris dataset*, gdje je precisnost predikcije na testnom skupu iznosila 100%, s gubitkom od 0,0511. Ovaj rezultat pokazuje točnost i pouzdanost aplikacije u treniranju modela na arbitrarno odabranom skupu podataka.

Budući razvoj platforme usmjerio bi se na proširenje njezinih funkcionalnosti, uključujući podršku za obradu većih skupova podataka, optimizaciju modela za primjenu u različitim domenama te implementaciju aplikacije u produkcijsku okolinu na platformu u oblaku, čime bi se omogućila šira dostupnost i korištenje platforme bez potrebe za lokalnim izvršavanjem.

LITERATURA

[1] TechTarget, What is web application: https://www.techtarget.com/searchsoftwarequality/definition/Web-application-Web-app (pristupljeno 2.11.2024.)

[2] Mozilla, what is a web browser: https://www.mozilla.org/en-US/firefox/browsers/what-is-a-browser (pristupljeno 2.11.2024.)

[3] Cloudflare, What is DNS? | How DNS workshttps://www.cloudflare.com/learning/dns/what-is-dns (pristupljeno 2.11.2024.)

[4] BrowserStack, Web Application Development: Process, Tools, & Examples https://www.browserstack.com/guide/web-application-development-guide (pristupljeno 3.11.2024.)

[5] BairesDevBlog, The Web Application Development Process: https://www.bairesdev.com/blog/web-application-development-process/ (pristupljeno 3.11.2024.)

[6] W3Schools, What is a Front-End Developer?: https://www.w3schools.com/whatis/whatis_frontenddev.asp (pristupljeno 3.11.2024.)

[7] Coursera, What Does a Back-End Developer Do?: https://www.coursera.org/articles/back-end-developer (pristupljeno 6.11.2024.)

[8] Atlassian, What is continuous integration? : https://www.atlassian.com/continuous-delivery/continuous-integration (pristupljeno 6.11.2024.)

[9] Stanford University, Artificial Intelligence Definitions PDF

[10] ISO Standards, What is artificial intelligence (AI)? https://www.iso.org/artificial-intelligence/what-is-ai (pristupljeno 7.11.2024.)

[11] University of Washington, The History of Artificial Intelligence 2006

[12] Coursera, The History of AI: A Timeline of Artificial Intelligence https://www.coursera.org/articles/history-of-ai (pristupljeno 7.11.2024.)

[13] Batta Mahesh, International Journal of Science and Research: Machine Learning Algorithms

[14] John D. Kelleher, The Massachusetts Instutute of Technology: Deep Learning

[15] Mohaiminul Islam, Guorong Chen, Shangzhu Jin: An Overview of Neural Network, American Journal of Neural Networks and Applications

[16] CloudFlare, What is a neural network?: https://www.cloudflare.com/learning/ai/what-is-neural-network/ (pristupljeno 10.11.2024.)

[17] IMB, What is deep learning? https://www.ibm.com/topics/deep-learning (pristupljeno 10.11.2024.)

[18] Michael Nielsen, How the backpropagation algorithm works: http://neuralnetworksanddeeplearning.com/chap2.html (pristupljeno 13.11.2024.)

[19] Robert Kwiatkowski, Gradient Descent Algorithm — a deep dive : https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21 (pristupljeno 14.11.2024.)

[20] GeeksForGeeks, Challenges in Deep Learning: https://www.geeksforgeeks.org/challenges-in-deep-learning/ (pristupljeno 14.11.2024.)

[21] Machine Learning Mastery, How to Avoid Overfitting in Deep Learning Neural Networks: https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/ (pristupljeno 14.11.2024.)

[22] Next.js, Documentation: https://nextjs.org/docs (pristupljeno 15.11.2024.)

[23] Sanity, Server-Side Rendering (SSR) overview: https://www.sanity.io/glossary/server-side-rendering (pristupljeno 15.11.2024.)

[24] Flask, Official Documentation: https://flask.palletsprojects.com/en/stable/ (pristupljeno 16.11.2024.)

[25] GeeksForGeeks, Introduction to Web development using Flask: https://www.geeksforgeeks.org/python-introduction-to-web-development-using-flask/ (pristupljeno 16.11.2024.)

[26] PyTorch, PyTorch documentation: https://pytorch.org/docs/stable/index.html (pristupljeno 17.11.2024.)

[27] PyTorch, How Computational Graphs are Constructed in PyTorch: https://pytorch.org/blog/computational-graphs-constructed-in-pytorch/ (pristupljeno 17.11.2024.)

[28] Nvidia, What Is CUDA?: https://blogs.nvidia.com/blog/what-is-cuda-2/ (pristupljeno 17.11.2024)

[29] BuiltIn, What Is PyTorch?: https://builtin.com/machine-learning/pytorch (pristupljeno 17.11.2024)

[30] Next.js, Runtimes: https://nextjs.org/docs/app/building-your-application/rendering/edge-and-nodejs-runtimes (pristupljeno 18.11.2024)

[31] Flask, Deploying to production: https://flask.palletsprojects.com/en/stable/deploying/ (pristupljeno 18.11.2024)

[32] Mozilla Developer, EventSource API: https://developer.mozilla.org/en-US/docs/Web/API/EventSource (pristupljeno 18.11.2024)

[33] Red Hat, What is a REST API?: https://www.redhat.com/en/topics/api/what-is-a-rest-api (pristupljeno 18.11.2024)

[34] Python, Official documentation, queue — A synchronized queue class: https://docs.python.org/3/library/queue.html (pristupljeno 18.11.2024)

[35] DataCamp, Cross-Entropy Loss Function in Machine Learning: Enhancing Model Accuracy: https://www.datacamp.com/tutorial/the-cross-entropy-loss-function-in-machine-learning (pristupljeno 18.11.2024)

[36] BuiltIn, Complete Guide to the Adam Optimization Algorithm: https://builtin.com/machine-learning/adam-optimization (pristupljeno 18.11.2024)

[37] PyTorch, Module: https://pytorch.org/docs/stable/generated/torch.nn.Module.html (pristupljeno 19.11.2024)

[38] Klu AI, F-Score: What are Accuracy, Precision, Recall, and F1 Score?: https://klu.ai/glossary/accuracy-precision-recall-f1 (pristupljeno 19.11.2024)

[39] Python, Official documentation, venv — Creation of virtual environments: https://docs.python.org/3/library/venv.html (pristupljeno 19.11.2024)

[40] Wikipedia, Iris flower data set : https://en.wikipedia.org/wiki/Iris_flower_data_set (pristupljeno 20.11.2024)

PRILOZI

I. Frontend Github repozitorij diplomskog rada: https://github.com/99vik/auto-ml-classifier

II. Flask-PyTorch microservice Github repozitorij: https://github.com/99vik/auto-ml-classifier-torch-api

III. API rute Flask servera

IV. Engine modul za inicijalizaciju i trening proizvoljnog modela dubokog učenja

V. Klasa za izradu modela dubokog učenja

VI. Petlja za trening modela dubokog učenja

VII. Modul za predviđanje ciljne varijable

```python
from flask import Flask, request, Response
from flask_cors import CORS
from src.utils import parse_train_request_params, parse_predict_request_params
from src.train_model_logic import train_model_logic
from src.predict_logic import predict_logic
import queue
import json
app = Flask(__name__)
CORS(app)

progress_queue = queue.Queue()

@app.post("/api/train-model")
def train_model():
    try:
        params = parse_train_request_params(request)
        train_model_logic(params, progress_queue)
    except Exception as e:
        progress_queue.put(json.dumps({"status": "error", "error": str(e)}))
        return Response('error', status=500)

    return Response('success', status=200)

@app.get("/api/train-progress")
def train_progress():
    def generate():
        while True:
            try:
                progress = progress_queue.get()
                yield f"data: {progress}\n\n"
                if '"status": "complete"' in progress:
                    break
            except queue.Empty:
                yield f"data: {'status': 'waiting'}...\n\n"

    return Response(generate(), mimetype='text/event-stream')

@app.post("/api/predict")
def predict():
    try:
        params = parse_predict_request_params(request)
        result = predict_logic(params)
    except Exception as e:
        return Response(f'error: {str(e)}', status=500)

    return Response(json.dumps({'prediction': result}), status=200)

if __name__ == "__main__":
    app.run(port=5000 , debug=False)
```

Prilog III. API rute Flask servera

```python
import torch
from torch import nn
from src.nn.Model import Model
from src.nn.train_loop import train_loop
from src.nn.helpers import to_split_tensor_data, csv_parser
import json

Viktor Miric, 3 months ago | 1 author (Viktor Miric)
class TensorEncoder(json.JSONEncoder):
    def default(self, obj):
        if isinstance(obj, torch.Tensor):
            return obj.cpu().numpy().tolist()
        return super(TensorEncoder, self).default(obj)

device = 'cuda' if torch.cuda.is_available() else 'cpu'

def engine(file,
           label_index,
           iterations,
           learning_rate,
           activation_function,
           progress_queue,
           hidden_layers,
           normalization,
           train_ratio,
           dropout,
           random_seed):
    try:
        labels, data, data_by_labels = csv_parser(label_index=label_index, file=file)
        X_tr, X_te, y_tr, y_te = to_split_tensor_data(labels, data, label_index, train_ratio, random_seed)
        X_tr = X_tr.to(device)
        X_te = X_te.to(device)
        y_tr = y_tr.to(device)
        y_te = y_te.to(device)

        if (random_seed):
            torch.manual_seed(random_seed)

        model = Model(input_size=len(X_tr[0]),
                      output_size=len(labels),
                      activation_function=activation_function,
                      hidden_layers=hidden_layers,
                      normalization=normalization,
                      dropout=dropout).to(device)
        loss_fn = nn.CrossEntropyLoss()
        optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

        train_loop(model, X_tr, y_tr, X_te, y_te, loss_fn, optimizer, iterations, progress_queue, labels, device)
        total_params = sum(p.numel() for p in model.parameters())

        state_dict = model.state_dict()
        json_state_dict = json.dumps(state_dict, cls=TensorEncoder)
    except Exception as e:
        raise RuntimeError(f'Error: {e}')

    return json_state_dict, data_by_labels, list(map(float, labels)), total_params
```

Prilog IV. Engine modul za inicijalizaciju i trening proizvoljnog modela dubokog učenja

```python
from torch import nn

You, 2 days ago | 1 author (You)
class Model(nn.Module):
        def __init__(self, input_size, output_size, activation_function, hidden_layers, normalization, dropout):
            super().__init__()
            self.activation = self.get_activation(activation_function)
            self.layer_stack = self.create_layers(input_size, output_size, hidden_layers, normalization, dropout)

        def get_activation(self, activation_function):
            if activation_function == 'relu':
                return nn.ReLU()
            elif activation_function == 'sigmoid':
                return nn.Sigmoid()
            elif activation_function == 'tanh':      You, 2 days ago • refactor project
                return nn.Tanh()
            elif activation_function == 'linear':
                return nn.Identity()

        def create_layers(self, input_size, output_size, hidden_layers, normalization, dropout):
            layers = []
            prev_size = input_size
            for size in hidden_layers:
                layers.append(nn.Linear(in_features=prev_size, out_features=size))
                layers.append(self.activation)
                if normalization:
                    layers.append(nn.LayerNorm(size))
                layers.append(nn.Dropout(p=dropout))
                prev_size = size
            layers.append(nn.Linear(in_features=prev_size, out_features=output_size))
            return nn.Sequential(*layers)

        def forward(self, x):
            return self.layer_stack(x)
```

Prilog V. Klasa za izradu modela dubokog učenja

```python
from torchmetrics import Accuracy, F1Score, ConfusionMatrix
import json
import torch

def train_loop(model, X_tr, y_tr, X_te, y_te, loss_fn, optimizer, iterations, progress_queue, labels, device):
    accuracy = Accuracy(task='multiclass', num_classes=len(labels)).to(device)
    f1score = F1Score(task="multiclass", num_classes=len(labels)).to(device)
    confmat = ConfusionMatrix(task="multiclass", num_classes=len(labels)).to(device)

    for i in range(iterations + 1):
        model.train()
        logits = model(X_tr)
        logits_pred = torch.softmax(logits, dim=1).argmax(dim=1)
        acc = accuracy(logits_pred, y_tr).item()*100

        loss = loss_fn(logits, y_tr)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        model.eval()
        if (i) % (iterations / 20) == 0:
            with torch.inference_mode():
                test_logits = model(X_te)
                test_logits_pred = torch.softmax(test_logits, dim=1).argmax(dim=1)
                test_loss = loss_fn(test_logits, y_te)
                test_acc = accuracy(test_logits_pred, y_te).item()*100
                f1 = f1score(test_logits_pred, y_te)
                confusion_matrix = confmat(test_logits_pred, y_te)

            training_data = {
                "status": "training",
                "iteration": str(i),
                "trainLoss": loss.item(),
                "trainAccuracy": acc,
                "testLoss": test_loss.item(),
                "testAccuracy": test_acc,
                "f1Score": f1.item(),
                "confusionMatrix": confusion_matrix.tolist()
            }
            progress_queue.put(json.dumps(training_data))
```

Prilog VI. Petlja za trening modela dubokog učenja

```
import torch
from src.nn.Model import Model
from src.nn.helpers import turn_json_to_torch

def predict_logic(params):

    model = Model(input_size=params['input_size'],
                  output_size=params['output_size'],
                  activation_function=params['activation_function'],
                  hidden_layers=params['hidden_layers'],
                  normalization=params['normalization'])

    state_dict = turn_json_to_torch(params['model_raw'])
    model.load_state_dict(state_dict)
    input = torch.tensor(params['inputsRaw']).float()
    result = model(input).argmax(0).item()
    return result
```

Prilog VII. Modul za predviđanje ciljne varijable