Sveučilište u Rijeci

Tehnički fakultet

Diplomski studij računarstva

Klasifikacija glasova pomoću neuronske mreže

Izvješće projekta

Kolegij: Računalna obrada govora i jezika

Student: Dino Drokan

Ak. god.: 2022./2023.

Rijeka, lipanj 2022.

Sadržaj

1. Uvod

Ovaj projekt uključuje razvoj neuronske mreže za prepoznavanje glasova hrvatskog jezika pomoću konvolucijskih neuronskih mreža i spektrograma audio zapisa. Konvolucijske neuronske mreže su se pokazale kao efikasan alat u području klasifikacije slika te se trenutno prilagođavaju za primjenu na audio podacima. Korištenje spektrograma audio signala na ulazu neuronske mreže omogućava neuronskom modelu sličnu interpretaciju podataka kao i kod slika. Što omogućuje pronalazak uzoraka u vremensko frekvencijskoj domeni signala.

Cilj projekta je razviti, istrenirati te testirati model neuronske mreže koji će moći za ulazni audio signal, odnosno audio isječak, prepoznati o kojem se izgovorenom glasu radi. Odnosno, klasificirati ulazni signal. Podaci korišteni u projektu su podaci iz VEPRAD baze podataka. Kao prvi korak u rješavanju zadatka, potrebno je transformirati podatke u format prikladan za ulaz u neuronsku mrežu. Audio datoteke se učitavaju, uzorkuju, ekstrahiraju se oznake te se izrađuju odgovarajući spektrogrami. Zatim se konstruira neuronska mreža sa odgovarajućom arhitekturom te se model trenira na testnom i validacijskom setu podataka. Set podataka je, prije treniranja modela, promiješan i podijeljen u trening, validacijski i testni podskup podataka. Tijekom testiranja modela isprobani su različiti parametri fourierove transformacije, dužine audio isječka te kompleksnosti neuronske mreže.

2. Programski alati

U projektu se koristi programski jezik Python uz podršku Tensorflow knjižnica funkcija. Napravljeno je conda virtualno okruženje u kojemu su instalirane određene verzije knjižnica funkcija tensorflow, tensorflow-io i python interpretera. Dodatno su instalirane knjižnice funkcija za podizanje jupyter poslužitelja te još popratne knjižnice funkcija: numpy, matplotlib, os, re, audio. Razvoj je napravljen u Dataspell integriranom razvojnom okruženju. Potrebno je bilo instalirati CUDA driver-e za podršku izvođenja programskog koda na grafičkoj kartici.

Python

Python je high-level programski jezik koji je poznat po svojoj jednostavnoj sintaksi i čitljivosti. Jezik je kreirao Guido van Rossum te je prvi put objavljen 1991. godine. Python podržava module i pakete što potiće modularnost programa i ponovnu upotrebu koda. Python-ova standardna knjižnica funkcija, koja je lako dostupna putem raznih modula i paketa, omogućuje programerima da se bave širokim spektrom programskih zadataka bez potrebe za stvaranjem novih alata iz nule. Jezik ima raznoliku primjenu, od web i poslovnih aplikacija do edukacijskih i znanstvenih primjena. Lagano se ući te je zato često prvi izbor kao programski jezik za učenje programiranja. Filozofija dizajna jezika naglašava čitljivost koda, sa sintaksom koja programerima omogućuje izražavanje koncepata u manje redaka nego što je to moguće u jezicima kao što su C++ ili Java. Python koristi dinamičko tipkanje, što znači da se ne mora deklarirati tip podataka varijable kod definicije varijable, nego Python automatski određuje vrstu podataka na temelju vrijednosti koja je dodijeljena. Štoviše, upravljanjem memorijom u Pythonu upravlja skupljač smeća, a jezik pruža strukture podataka visoke razine poput lista i rječnika koji su prikladni za pisanje složenih programa. Sve te značajke doprinose tome da Python postane jedan od najpopularnijih i najkorištenijih programskih jezika na svijetu.

TensorFlow

TensorFlow je knjižnica funkcija otvorenog koda početno razvijena od strane Google-a. Nudi podršku za razvoj programa koji koriste strojno i duboko učenje. Nudi mogućnost izvršavanja koda na različitim platformama (CPU, GPU, TPU) te na različitim računalima (desktop, server, cluster). Stoga je dobar odabir za učenje tehnika strojnog učenja i za samu produkciju finalne aplikacije ili programa. High-level API-i TensorFlow-a, kao na primjer Keras, olakšavaju izradu i treniranje modela strojnog učena. Pomoću api-a definiranje, treniranje i evaluacija modela postaju relativno lagan problem. Knjižnica ima ogromnu, aktivnu zajednicu korisnika i suradnika koji neprestano poboljšavaju i

proširuju njezinu funkcionalnost. Snažna podrška od strane Googlea i široka upotreba u praksi također osigurava redovita ažuriranja, ispravke grešaka i značajke. Štoviše, TensorFlow pruža opsežnu dokumentaciju i resurse za učenje, uključujući niz tutorijala i vodiča koji pomažu novim korisnicima da započnu, a iskusnim ML praktičarima da dublje prodru u napredne teme.

Conda virtualna okruženja

Conda je sustav, otvorenog koda, koji se koristi za menadžment paketa i radnih okolina. Korisnicima omogućuje izradu okolina koje sadrže datoteke, pakete, i druge ovisne pakete na način da paketi i datoteke unutar jedne okoline ne interferiraju sa datotekama i paketima u drugim okolinama. Okoline mogu imati instalirane različite verzije Python-a i paketa, što smanjuje konflikte između paketa te omogućuje lakši i fleksibilniji rad na više projekata te omogućuje konzistentno razvojno i izvršno okruženje. Svako Conda okruženje je potpuno zaseban direktorij koji sadrži određene conda pakete. To omogućuje dijeljenje okruženja na različitim računalima, što je posebno korisno u timskom okruženju jer programeri mogu jednostavno replicirati okruženja drugih kolega. Štoviše, olakšava debug-iranje problema jer se okruženje može replicirati i dijeliti s drugima. Osim toga, Conda također upravlja paketima iz raznih repozitorija, pružajući veliki izbor paketa koji se mogu koristiti u različitim okruženjima.

Python knjižnice funkcija

NumPy, što je kratica za Numerical Python, temeljni je paket za znanstveno računarstvo u Python-u. Pruža podršku za velike, višedimenzionalne liste i matrice, zajedno s velikom zbirkom matematičkih funkcija za manipulaciju listi i matrica. Budući da je osnovna knjižnica funkcija za znanstveno računanje, NumPy se koristi za zadatke poput numeričke integracije, interpolacije, optimizacije, linearne algebre i za još mnogo toga. NumPy-ove mogućnosti operacija nad listama čine osnovu mnogih drugih knjižnica funkcija u znanstvenom ekosustavu Python, uključujući Pandas i SciPy.

Matplotlib je Python knjižnica funkcija koja se koristi za generiranje grafikona i dijagrama. Sa svojim velikim skupom funkcionalnosti, sposobna je za izradu histograma, raspršenih dijagrama, linijskih dijagrama, stupčastih grafikona, dijagrama pogrešaka i još mnogo toga. Također podržava brojne ekstenzije, poznate kao toolkits, od kojih svaka doprinosi dodatnim mogućnostima crtanja. Knjižnice funkcija os i re u Pythonu, koriste se za interakciju s operativnim sustavom i izvođenje operacija regularnih izraza. Modul os uključuje brojne funkcije za interakciju sa datotečnim sustavom, upravljanje procesima i čitanje varijabli iz okoline. Modul re pruža funkcionalnosti povezane s regularnim izrazima koji se koriste za manipulaciju nizovima znakova i podudaranje uzoraka. Audio se općenito povezuje s knjižnicama funkcija kao što su PyAudio ili Librosa, koje se koriste za zadatke obrade zvuka kao što su čitanje i pisanje audio datoteka, obrada audio signala i ekstrakcija audio značajki.

DataSpell i Jupyter server

DataSpell je integriarno razvojno okruženje razvijeno od strane JetBrain firme. DataSpell je napravljen kako bi olakšao eksplorativnu analizu podataka i razvoj modela strojnog učenja. DataSpell nudi podršku za Jupyter bilježnice. Jupyter server je okruženje na kojemu se izvršava jupyter bilježnica. Server i bilježnica omogućuju da podaci ostanu u memoriji nakon izvršavanja blokova koda, što omogućuje interaktivni i olakšani razvoj.

3. Implementacija projekta

U sljedećim poglavljima je prezentirana i objašnjena implementacija projekta.

Upoznavanje sa podacima

Podaci korišteni za projekt su podaci iz VEPRAD baze podataka. VEPRAD baza podataka sadrži audio snimke vremenske prognoze i vijesti sa Hrvatskog nacionalnog radija. Uz audio snimke, baza sadrži i odgovarajuće transkripcije audio zapisa.

Korišteni skup podataka sadrži oko 2000 audio isječaka te odgovarajuću transkripciju za svaki isječak. Audio isječci su u formatu .wav datoteke. Transkripcije su zapisane u .lab datotekama te je struktura jedne .lab datoteke sljedeća: [početak izgovorenog glasa] [kraj izgovorenog glasa] [oznaka izgovorenog glasa]. Početak i kraj izgovorenog glasa izraženi su u vremenskim jedinicama gdje 1 jedinica predstavlja 0.1 mikrosekundu. Vremenske jedinice su relativne prema početku audio zapisa, odnoso označavaju vrijeme koje je prošlo od početka audio zapisa.

Obrada podataka

Prvi korak je prikupljanje vrijednosti svih glasovnih oznaka. Te stoga prvi blok koda radi upravo to. Na početku se kreira prazna list naziva "labels" koja će se koristit za pohranu jedinstvenih vrijednosti oznaka pronađenih u lab datotekama. Zatim se putanja do mape koja sadrži lab datoteke definira u folderPath varijabli, a listdir funckija se koristi za dobivanje liste svih naziva datoteka u direktoriju. Skripta zatim iterira kroz naziv svake datoteke u listi. Za svaku datoteku kreira puni put do datoteke, pokušava je otvoriti i učitava sve retke datoteke. Ako datoteka nije pronađena, javlja se FileNotFoundError exception i ispisuje se "File not found".

Skripta zatim prelazi korz svaki redak u trenutnoj datoteci. Uklanja znak novog retka (\n) s kraja svakog retka, a zatim primjenjuje pretraživanje regularnog izraza koje hvata sve znakove nakon drugog razmaka. Pronađen karakter je pohranjen u vraijablu newLabel. Skripta tada provjerava je li vrijednost newLabel varijable već unutar liste oznaka varijable label. Ako nije, dodaje newLabel u listu.

Na kraju se stvara rječnik pod nazivom labelsDict. Ovaj rječnik uparuje svaku oznaku (koja služi kao ključ rječnika) s njenim indeksom (koji služi kao vrijednost rječnika). Funkcija enumerate() koristi se za uparivanje svake oznake s njezinim indeksom. Ovaj se rječnik zatim ispisuje na konzolu. Konačni rezultat je sortirana lsita jedinstvenih oznaka i rječnik koji svaku oznaku preslikava u njezin položaj u sortiranoj listi.

Sljedeći blok koda se koristi za obradu audio datoteka, odnosno, izradu audio uzoraka te dodijelu oznake svakom uzorku. Konkretno, skripta čita audio datoteke i datoteke s oznakama, izdvaja i obrađuje relevantne dijelove audiozapisa na temelju vremenskih oznaka danih u datotekama s oznakama i dodaje obrađeni audio uzorak zajedno s odgovarajućom oznakom u listu.

Skripta počinje inicijalizacijom prazne liste, podataka, koja će pohraniti parove obrađenih audio uzoraka i njihove odgovarajuće oznake.

U sljedećem većem dijelu skripte, kod iterira preko parova naziva audio datoteka i datoteka oznaka pohranjenih u labAndWavFilesFixed listi. Za svaki par pokušava otvoriti audio datoteku koristeći AudioIOTensor metodu iz TensorFlow knjižnice funkcija, koja stvara tenzor iz audio datoteke i sprema brzinu uzorkovanja zvuka (eng. sample rate) u varijablu rate koja se nalazi u Tensor objektu. Također, otvara odgovarajuću label datoteku i učitava sve retke. Nazivi audio datoteka i label datoteka ispisuju se u konzoli, a za svaki redak u datoteci oznake, vremenske oznake i vrijednost oznake (glas) izdvajaju se pomoću regularnih izraza. Ove vremenske oznake predstavljaju početnu i završnu točku relevantnog audio uzorka unutar audio datoteke i pretvaraju se u indekse uzoraka množenjem s brzinom uzorkovanja i dijeljenjem s 10000000.

U završnom dijelu bloka skripte, kod reže audio tenzor prema izračunatim početnim i završnim indeksima i provjerava duljinu uzorka. Ako je duljina manja od 3000, a oznaka nije jedna od navedenog skupa neželjenih oznaka, tenzor uzorka i njegova odgovarajuća oznaka dodaju se kao tuple u listu podataka. Sve iznimke koje se dogode tijekom obrade para datoteka se hvataju, brojač pogrešaka naziva "er" se povećava, poruka o pogrešci se ispisuje zajedno s iznimkom, a obrada prelazi na sljedeći par datoteka.

Sljedeći blok koda obavlja dva zadatka: pronalaženje maksimalne duljine audio uzoraka, a zatim zero padding svih uzoraka kako bi odgovarali definiranoj maksimalnoj duljini.

U prvom dijelu bloka koda, varijabla max se inicijalizira na nulu i stvara se prazna lista maxValues. Kod zatim prolazi kroz svaki uzorak u listi sa podacima. Za svaki uzorak određuje veličinu (duljinu) uzorka. Ako je trenutna veličina veća od trenutnog maksimuma, max se ažurira na novu trenutnu veličinu i veličina se dodaje u listu maxValues. Ispisuje se i indeks trenutnog uzorka. Ovaj dio skripte odgovoran je za pronalaženje maksimalne duljine među svim audio uzorcima u listi podataka.

U drugom dijelu bloka skripte, kod izvodi zero-padding na audio uzorcima. Iterira kroz dubinsku kopiju liste podataka. Duboka kopija koristi se za sprječavanje izmjena izvorne liste tijekom iteracije, što bi moglo dovesti do neočekivanih rezultata. Za svaki uzorak, kod izračunava veličinu uzorka, zatim stvara novi tenzor, naziva zerosTensor, nula s duljinom koja je jednaka razlici između maksimalne i trenutne veličine uzorka. Taj se tenzor zatim povezuje s izvornim tenzorom uzorka duž vremenske osi (os=0), što rezultira novim tenzorom naziva newTensor koji ima istu duljinu kao maksimalna duljina uzorka. Ovaj novi tenzor zamjenjuje izvorni tenzor uzorka u listi podataka.

Zatim, sljedeći blok koda vizualizira audio formu prvih 10 audio uzoraka glasa b u skupu podataka. Vizualizacije pokazuju sličnosti između različitih audio uzoraka glasa b te pokazuju da su svi audio uzorci iste veličine, odnosno duljine. Dva audio uzorka se reproduciraju kako bi se potvrdilo da su oznake pravilno dodijeljene audio uzorcima.

Sljedeći blok koda definira funkciju naziva get_spectrogram, a zatim funkciju primjenjuje na listu uzoraka audio podataka kako bi se svaki uzorak transformirao u odgovarajući spektrogram.

Funkcija get_spectrogram kao argument prima zvučni valni oblik i vraća njegov spektrogram, što je vizualni prikaz spektra frekvencija u audio signalu kako se mijenjaju s vremenom. Kratkovremenska Fourierova transformacija (STFT), koju izvodi tf.signal.stft funkcija, pretvara valni oblik u niz okvira i izračunava Fourierovu transformaciju svakog okvira. Parametri frame_length=128 i frame_step=64 određuju veličinu okvira i veličinu koraka, odnosno duljinu skoka između okvira.

Izlaz STFT-a je kompleksni broj, pa se funkcija tf.abs koristi za dobivanje veličine kompleksnih brojeva, odnosno njezin realni dio, čime se dobiva spektrogram magnitude. Ovo je prikaz stvarne vrijednosti koji pokazuje kako se snaga (ili veličina) različitih frekvencija mijenja tijekom vremena u audio signalu.

Na kraju, spektrogramu se dodaje nova os kako bi odgovarao očekivanom formatu unosa za konvolucijske neuronske mreže, gdje je ulaz često 4D tenzor s oblikom (`batch_veličina`, `visina`, `širina`, `kanali`). U slučaju slika u sivim tonovima, broj kanala je 1.

Nakon definiranja funkcije, skripta zatim iterira kroz svaki audio uzorak iz liste podataka. Za svaki uzorak primjenjuje funkciju get_spectrogram. Važno je napomenuti da se podaci valnog oblika pretvaraju u float tip podataka prije primjene funkcije, budući da funkcija STFT zahtijeva ulaze s float tipom podataka. Rezultirajući spektrogram, zajedno s izvornom oznakom uzorka, zatim se dodaje u novu listu podataka naziva spectData.

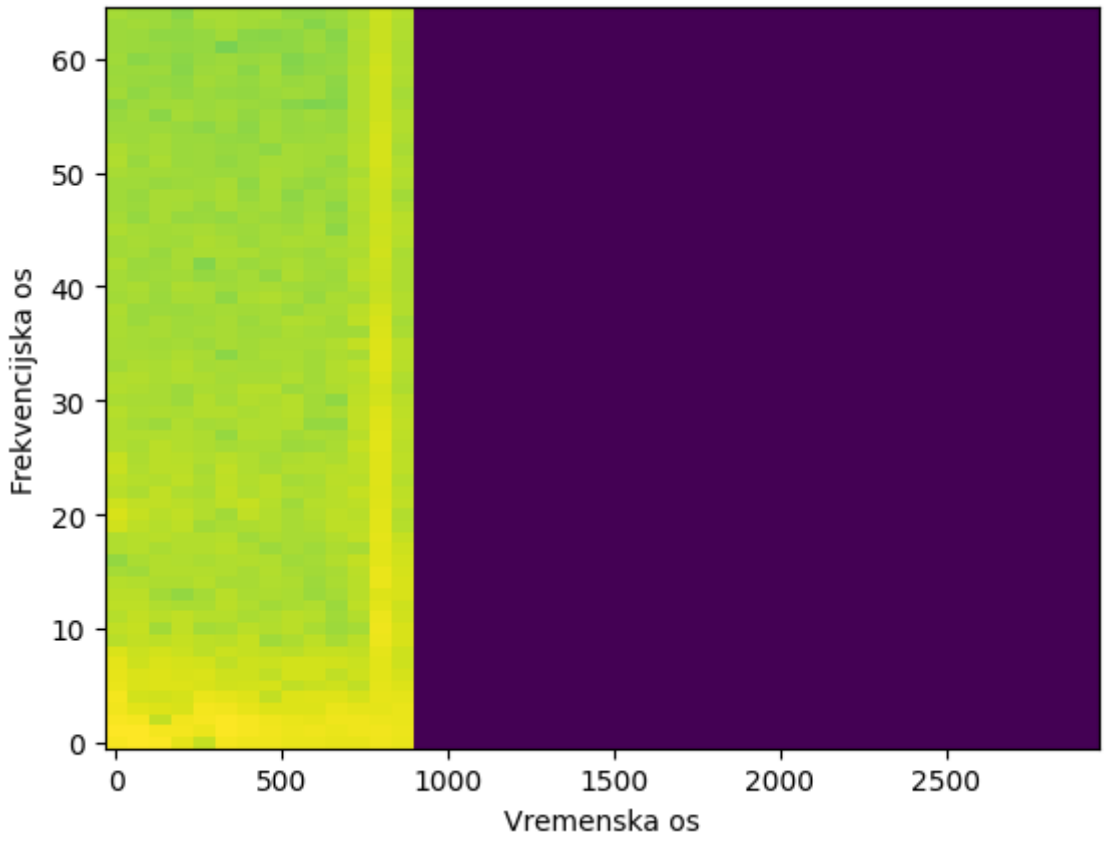
Blok prethodno opisanog koda:

```
def get_spectrogram(waveform):
 #STFT.
 spectrogram = tf.signal.stft(
 waveform, frame_length=128, frame_step=64)

 spectrogram = tf.abs(spectrogram)
 # Dodavnaje jos jedne dimenzije
 # shape (`batch_size`, `height`, `width`, `channels`).
 spectrogram = spectrogram[..., tf.newaxis]
 return spectrogram


spectData = list()
for index, sample in enumerate(data):
 print(index)
```

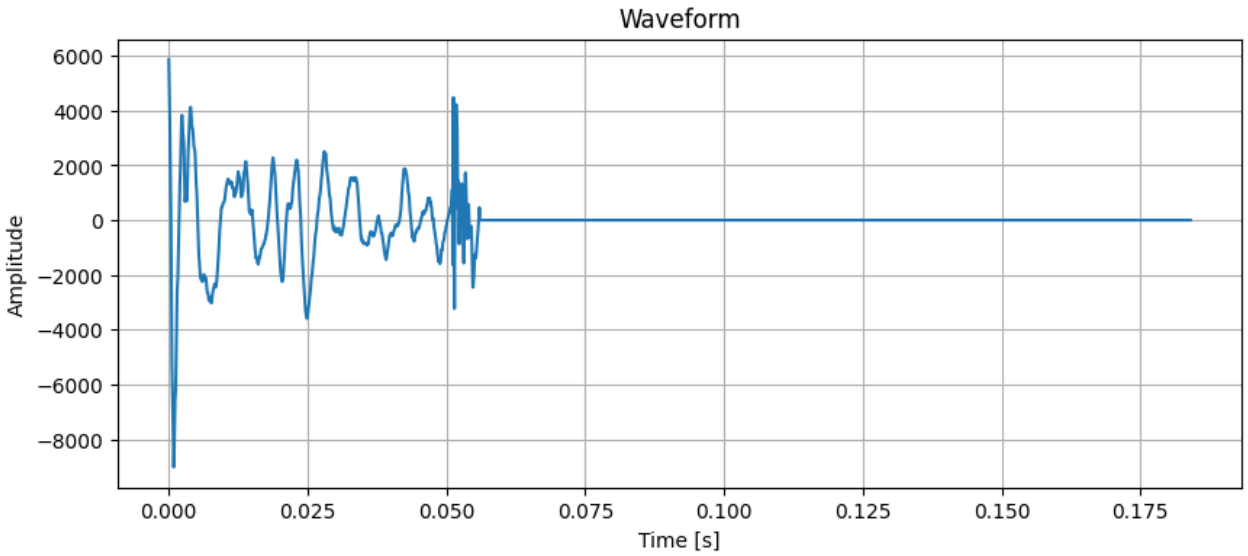
```
 spectrogram = get_spectrogram(sample[0].numpy().astype(float))
 spectData.append([spectrogram, sample[1]])
```

Zatim se radi vizualizacija spektrograma glasa b sa sljedećim kodom:

```
import matplotlib.pyplot as plt
def plot_spectrogram(spectrogram, ax):
 if len(spectrogram.shape) > 2:
 assert len(spectrogram.shape) == 3
 spectrogram = np.squeeze(spectrogram, axis=-1)
 log_spec = np.log(spectrogram.T + np.finfo(float).eps)
 height = log_spec.shape[0]
 width = log_spec.shape[1]
 X = np.linspace(0, np.size(spectrogram), num=width, dtype=int)
 Y = range(height)
 ax.pcolormesh(X, Y, log_spec)

i = 0
for index, sample in enumerate(spectData):
 if sample[1] == 'b':
 fig, ax = plt.subplots()
 plot_spectrogram(sample[0].numpy(), ax)
 plt.show()
 i += 1
 if i == 10:
 break
```



Slika 1. Spektrogram audio uzorka glasa b. X os predstavlja vrijeme, a Y os frekvencije (od manjih prema većima)



Slika 2. Valni oblik audio uzorka glasa b

Nakon izrade audio uzoraka i izrade spektrograma potrebno je podatke zapisati u strukturu TensorFlow Dataset. TensorFlowov tf.data API omogućuje izgradnju složenih ulaznih pipeline-ova od jednostavnih dijelova koji se mogu ponovno koristiti. TF skupovi podataka su vrlo učinkoviti, jednostavni za korištenje i moćni, što ih čini ključnom komponentom za svaki program strojnog učenja koji radi s TensorFlow-om.

Primarna klasa u API-ju tf.data je klasa tf.data.Dataset, koja predstavlja niz elemenata, gdje je svaki element u nizu lista ili tuple gdje je jedan element TensorFlow tenzor. Stvaranje skupa podataka može biti jednostavno poput korištenja jedne od metoda kao što su Dataset.from_tensors, Dataset.from_tensor_slices ili čak iz izvora podataka poput tekstualnih ili CSV datoteka pomoću tf.data.TextLineDataset ili tf.data.experimental.CsvDataset metoda. Nakon što postoji objekt tipa tf.data.Dataset, može se transformirati u novi skup podataka lančanim pozivima metoda na objektu. Na primjer, može se primijeniti transformacije po elementima kao što je map(za primjenu funkcije na svaki element) i transformacije s više elemenata kao što su batch (za grupiranje više elemenata zajedno) ili shuffle (za nasumični redoslijed elemenata).

Jedan od moćnih aspekata tf.data API-ja je njegova sposobnost rukovanja velikim količinama podataka, podacima koji ne stanu u memoriju, pa čak i podacima distribuiranim na više uređaja. To čini stvaranjem ulaznog pipeline-a koji učinkovito upravlja memorijom i paralelnim računanjem.

Za izradu skupa podataka koristi se funkcija tf.data.Dataset.from_tensor_slices() koja kao argumente prima listu sa spektrogramima i listu sa oznakama spektrograma. Te se zatim izrađuju trening, validacijski i testni skup podataka. Prvi korak je miješanje skupa podataka pomoću funkcije shuffle(). Parametar buffer_size funkcije shuffle određuje broj elemenata iz skupa podataka koji se trebaju učitati u međuspremnik prije miješanja. Veća veličina međuspremnika omogućuje bolje miješanje, ali koristi više memorije i traje malo duže, dok manja veličina međuspremnika ima manje savršeno miješanje, ali je učinkovitije u pogledu resursa. Početni parametar koristi se kako bi miješanje bilo deterministično. Kada se osigura početna vrijednost, podaci će se svaki put miješati na isti način.

Nakon što se skup podataka pomiješa, dijeli se na skupove podataka za trening, validaciju i testiranje. Varijabla total_size sadrži ukupan broj uzoraka podataka prisutnih u skupu podataka. Veličina skupa za trening je 70% od ukupnog broja, veličina skupa za validaciju je 20%, a preostalih 10% je za skup za testiranje. Metode take() i skip() koriste se za dijeljenje skupa podataka u ova tri podskupa. Metoda take() dohvaća određeni broj elemenata s početka skupa podataka, dok metoda skip() preskače

određeni broj elemenata s početka skupa podataka.Na kraju, kod ispisuje duljine skupova podataka za trening, validaciju i testiranje, kako bi potvrdio da je proces razdvajanja ispravno izveden.

Kod procesa raspodijele skupa podataka:

```
# Shuffle the dataset
dataset_shuffled = tfDataset.shuffle(buffer_size=1000, seed=42)

# Izracun duljina train, val i test datasetova
total_size = len(tfDataset)
train_size = int(0.7 * total_size) # 70% for training
val_size = int(0.2 * total_size) # 20% for validation
test_size = total_size - train_size - val_size # Remaining 10% for testing

# raspodjela
train_dataset = dataset_shuffled.take(train_size)
test_and_val_dataset = dataset_shuffled.skip(train_size)
val_dataset = test_and_val_dataset.take(val_size)
test_dataset = test_and_val_dataset.skip(val_size)

print(len(list(train_dataset)), len(list(val_dataset)), len(list(test_dataset)))
```

Sljedeći blok koda grupira skupove podataka za trening, validaciju i testiranje. Grupiranje se odnosi na proces grupiranja određenog broja uzoraka podataka u jednu seriju (eng. Batch), što je važan postupak u treningu modela strojnog učenja, posebno u neuronskim mrežama. U TensorFlowu se funkcija batch() koristi za dijeljenje skupa podataka u serije određene veličine. Ovdje je veličina serije postavljena na 8, što znači da će svaka serija sadržavati 8 uzoraka iz skupa podataka. Grupiranje podataka ključno je za efikasnost izvođenja. Umjesto unosa podataka u model jedan po jedan uzorak, što može biti računalno neučinkovito, model prima podatke u serijama. To omogućuje modelu da paralelizira svoje proračune, što dovodi do bržeg vremena obuke. Štoviše, grupiranje također pomaže u generalizaciji modela tijekom obuke osiguravajući da svaka serija ima mješavinu svih klasa podataka. Skupovi train_dataset, val_dataset i test_dataset se dijele u serije od po 8 uzoraka, što ih čini spremnima za trening modela, validaciju i testiranje.

Izrada modela, treniranje i testiranje

U nastavku je opisan postupak izrade modela i njegovog treniranja.

Na početku bloka koda, definira se ulazni oblik podataka u model. Ulazni oblik se dobiva iz oblika tenzora zapisanog u skupu podataka. Sloj normalizacije također je ugrađen u model. Ovaj sloj je namijenjen standardizaciji ulaznih podataka na temelju srednje vrijednosti i varijance podataka u skupu podataka za obuku. Sloj se prilagođava trening podacima pomoću funkcije adapt(). Svrha ove normalizacije je osigurati da model prima ulaze koji su homogeni i unutar određenog raspona, što pomaže u poboljšanju stabilnosti i izvedbe procesa učenja.

Što se tiče arhitekture modela, razmatrane su tri odvojene konfiguracije: jedna s osnovnom razinom složenosti, druga s umjerenom složenošću i završna s visokom složenošću. U svakoj konfiguraciji, konvolucijska neuronska mreža usvojena je kao primarna arhitektura. Osnovni model koristi jednostavnu arhitekturu s dva konvolucijska sloja, nakon čega slijedi sloj naziva MaxPooling2D, sloj za regulaciju, sloj za izravnavanje oblika izlaza značajki i dva gusta sloja za klasifikaciju. Model umjerene složenosti proširuje ovu osnovnu strukturu, povećavajući broj konvolucijskih slojeva i dodajući skupnu normalizaciju nakon svakog konvolucijskog sloja kako bi se poboljšalo učenje. Konačni, najsloženiji model dodatno unapređuje strukturu dodavanjem više konvolucijskih slojeva, nakon kojih slijedi normalizacija batch-eva. Ovaj model također uvodi višu stopu ispadanja kako bi se spriječio overfitting i uključuje gušće povezane slojeve na kraju za konačnu klasifikaciju. Svaka od ovih konfiguracija dizajnirana je s ciljem optimiziranja sposobnosti modela da pravilno klasificira ulazne spektrograme u odgovarajuće oznake.

Svi ovi modeli dizajnirani su pomoću Sekvencijalnog API-ja iz TensorFlowovog Keras modula. Modul pruža jednostavne funkcije za definiranje modela neuronske mreže kao niza slojeva. Korištenjem ovog modula, arhitektura modela može se lako čitati, modificirati i razumjeti, čime se olakšavaju daljnji eksperimenti i poboljšanja.

Definicija modela s osnovnom razinom složenosti:

```
model = models.Sequential([
 layers.Input(shape=input_shape),
 # Downsample
 layers.Resizing(32, 32),
 # Normalize.
 norm_layer,
 layers.Conv2D(32, 3, activation='relu'),
 layers.Conv2D(64, 3, activation='relu'),
 layers.MaxPooling2D(),
 layers.Dropout(0.25),
 layers.Flatten(),
 layers.Dense(128, activation='relu'),
 layers.Dropout(0.5),
 layers.Dense(num_labels),
])
```

Model se zatim prevodi pomoću funkcije model.compile() uz odgovarajuće funkcije za optimizaciju, gubitak te parametar metrike. Funkcija optimizacije je postavljena na: tf.keras.optimizers.Adam(). Metrika modela se mjeri sa točnošću, odnosno, koliko često model predvidi točnu oznaku (glas) spektrograma.

Model se zatim trenira kroz 10 epoch-a, gdje je jedan epoch jedan prolazak modela kroz cijeli dataset. Varijacije modela i trening parametara te rezultati su opisani u poglavlju rezultati.

Sljedeći segment koda testira model na podskupovima testnog dataseta. Podskupovi podataka sadrže podatke (glasove) iz sljedećih kategorija: "Palatali", "Samoglasnici", "C", "a", "c", "b" i "h". U početku se stvaraju prazne liste za svaku kategoriju za pohranjivanje spektralnih podataka i oznaka. Za svaku kategoriju stvorene su dvije liste: jedna za spektralne podatke i jedna za oznake. Zatim, kod iterira kroz testni skup podataka. Za svaku oznaku provjerava pripada li oznaka jednoj od unaprijed definiranih kategorija. Ove kategorije određene su skupinom brojeva oznaka koje predstavljaju različite zvukove. Na primjer, kategorija "Palatali" uključuje zvukove s oznakama 13, 11, 5, 4, 3, 2, 1, 0, 22. Ako oznaka zvuka pripada jednoj od kategorija, spektralni podaci i oznaka pridodaju se odgovarajućim listama za tu kategoriju. Ovaj se postupak ponavlja za sve parove spektralnih podataka i oznaka u testnom skupu podataka, čime se podaci odvajaju u sedam kategorija. Nakon kategorizacije podataka, kod pretvara svaki skup listi (spektralni podaci i oznake) u skup podataka TensorFlow pomoću tf.data.Dataset.from_tensor_slices. Ova funkcija stvara skup podataka rezanjem ulaznih tenzora duž njihove prve dimenzije. To se radi za svaku kategoriju, što rezultira u sedam zasebnih skupova podataka TensorFlow. Potom se svaki skup podataka grupira u grupe od osam elemenata pomoću metode batch(8). Konačno, izvedba modela se procjenjuje na svakom od sedam skupova podataka. Funkcija model.evaluate izračunava gubitak i metriku za skup podataka i vraća ih kao listu.

Rezultati

Treniranje modela je napravljeno više puta sa različitim vrijednostima parametara. Parametri koji su varirani su: duljina vremenskog prozora i koraka kod STFT, duljina audio uzorka, kompleksnost neuronske mreže. Broj epoch-a je 10.

Za duljinu uzorka manju od 1000 i STFT frame_length i frame_step od 128 i 64, performanse modela stalno se poboljšavaju tijekom treninga, točnost nad testnim skupm podatka je 61,72% uz gubitak od 1,1685. Kada je ograničenje duljine uzorka povećano na manje od 2000 s istim STFT parametrima (128 i 64), točnost modela (testni skup podatka) značajno se poboljšala na 69,86%, a gubitak se

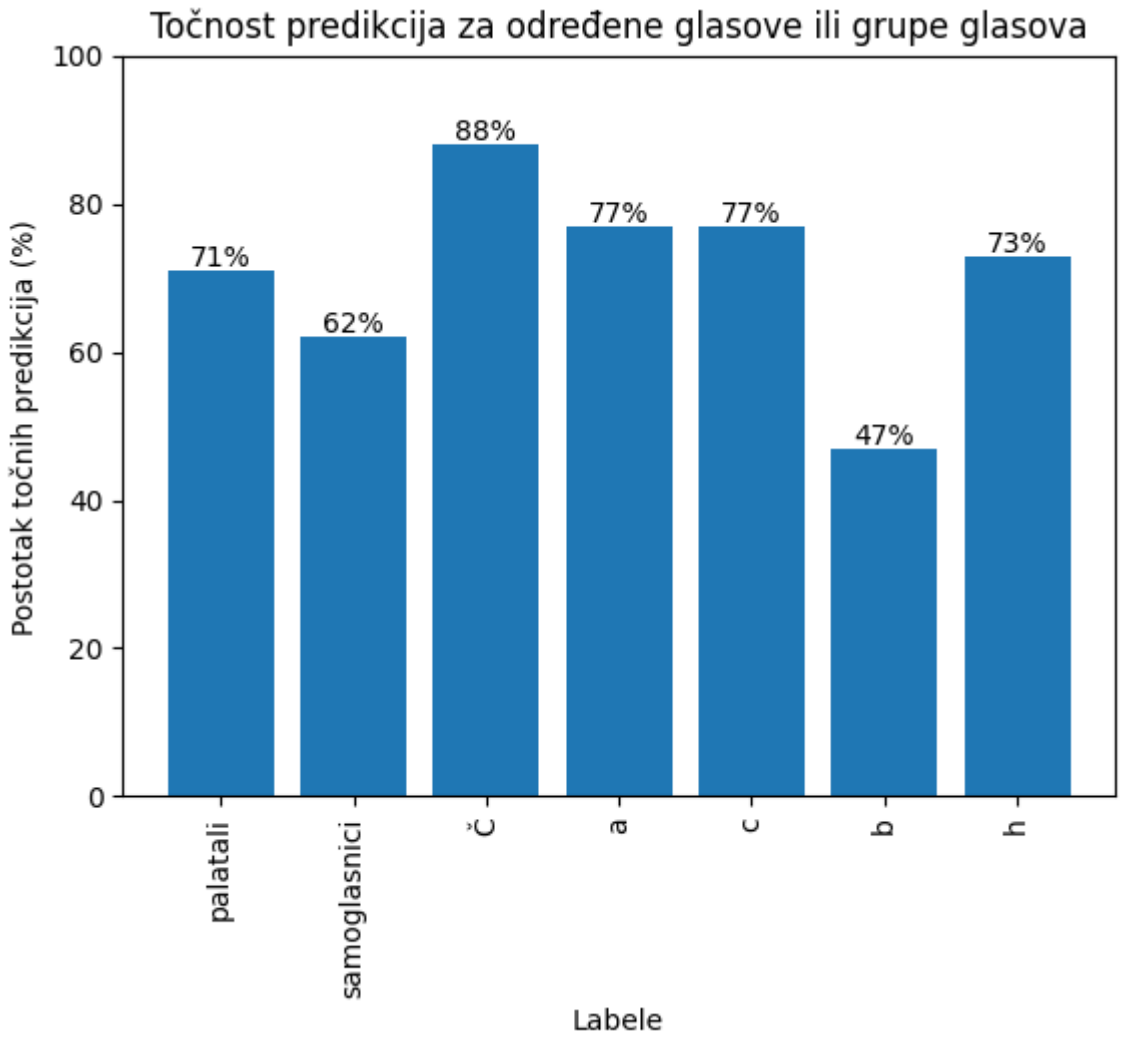
smanjio na 0,8990. Modificiranje STFT parametara na 64 i 32 uz zadržavanje duljine uzorka manje od 2000 dovelo je do najveće zabilježene točnosti od 71,10% uz gubitak od 0,8571. Ovo sugerira da manja duljina okvira i veličina koraka mogu dovesti do boljih performansi modela s većom veličinom uzorka. Međutim, kako su se duljina STFT okvira i veličina koraka povećavali (256/128 i 512/480), performanse modela su se smanjivale s točnostima od 68,11% odnosno 56,88%, što ukazuje da veće duljine okvira i veličine koraka nisu idealni za ovaj specifični model i podatke. Srednja kompleksnost mreže se pokazala kao najboja, manja i veća kompleksnost su davale lošije rezultate od srednje.

Najbolja konfiguracija (sample length < 2000, _length=64, frame_step=32, srednja kompleksnost mreže) je bila dodatno ponovno trenirana sa 15 epoch-a. Povećanje broja epoch-a nije utjecalo na bolju točnost modela koja je ostala ista 71,10%.

Tablica 1: Metrike modela tijekom treniranja te metrika na testnom skupu podataka .

Epoha	Train_loss	Train_acc	Val_loss	Val_acc
1	2.0000	0.3970	1.6134	0.4957
2	1.5377	0.5188	1.1672	0.6160
3	1.3998	0.5640	1.1424	0.6280
4	1.2852	0.5989	1.4466	0.6601
5	1.2288	0.6200	1.0337	0.6508
6	1.1715	0.6357	1.0031	0.6637
7	1.1325	0.6436	0.9554	0.6792
8	1.0987	0.6582	0.9239	0.6885
9	1.0668	0.6666	0.9244	0.6945
10	1.0434	0.6747	0.8803	0.7032
Testni skup podataka	**Loss: 0.8571**	**Accuracy: 0.7110**		

Model je dodatno testiran na podskupovima testnog seta podataka. Svaki podskup sadrži uzorke određenog glasa ili skupa glasova.



Slika 3. Statistički prikaz točnosti modela na određenim grupama glasova

4. Zaključak

Ovaj projekt imao je za cilj razviti model strojnog učenja koji je sposoban predvidjeti izgovorene glasove iz audio zapisa hrvatskog jezika. Cilj je bio iskoristiti snagu dubinskog učenja, posebno konvolucijske neuronske mreže (CNN), za zadatke prepoznavanja govora. Projekt je uključivao različite faze, uključujući prethodnu obradu audio podataka, razvoj CNN modela, obuku modela i procjenu njegove izvedbe na novim podacima.

Audio podaci su prikupljeni i obrađeni na odgovarajući način. Svaki audio isječak izrezan je u audio uzorke koji predstavljaju izgovorene glasove, a koji su služili kao podaci za obuku i testiranje modela. Uzorcima su dodijeljene oznake na temelju izgovorenog glasa. Bitan dio obrade podatka je bio pretvorba audio uzoraka iz vremenske domene u vremensko frekvencijsku domenu, korištenjem Fourierove transformacije.

Projekt je koristio model konvolucijske neuronske mreže zbog njegove dokazane učinkovitosti u zadacima prepoznavanja uzoraka, uključujući prepoznavanje govora. Ovaj je model razvijen pomoću biblioteke Keras i obučen na prethodno obrađenim audio podacima. Arhitektura modela, parametri obuke i hiperparametri pažljivo su odabrani kako bi se uravnotežila složenost modela, učinkovitost

učenja i troškovi računanja. Nakon obuke modela, provedena je faza evaluacije pomoću zasebnog skupa testnih podataka koje model prethodno nije vidio. Ovaj korak je ključan u procjeni sposobnosti modela da generalizira podatke iz obuke na nove podatke. Metrike kao što su točnost i gubitak korišteni su za procjenu izvedbe modela. Ove mjere pružile su vrijedan uvid u učinkovitost modela i njegov potencijal za primjenu u stvarnom svijetu. Uvježbani model pokazao je svoju praktičnu korisnost predviđajući nove, neviđene audio podatke. S obzirom na novi audio isječak, model je mogao predvidjeti izgovoreno slovo (sa točnošću od 70%), čime je ispunio izvorni cilj projekta.

Zaključno, ovaj projekt naglašava potencijal strojnog učenja, posebno CNN-a, u području prepoznavanja govora. Uspjeh modela u predviđanju izgovorenih slova iz hrvatskih audio zapisa dokaz je moći metodologija dubokog učenja. No, uvijek ima prostora za poboljšanja i napredak.

5. Literatura

[1] Basic classification: Classify images of clothing [Online] - https://www.tensorflow.org/tutorials/keras/classification

[2] But what is a neural network? - https://www.youtube.com/watch?v=aircAruvnKk&ab_channel=3Blue1Brown

[3] Artificial neural network - https://en.wikipedia.org/wiki/Artificial_neural_network?useskin=vector

[4] Audio Data Preparation and Augmentation - https://www.tensorflow.org/io/tutorials/audio

[5] Simple audio recognition: Recognizing keywords - https://www.tensorflow.org/tutorials/audio/simple_audio

[6] Audio Deep Learning Made Simple: Sound Classification, Step-by-Step - https://towardsdatascience.com/audio-deep-learning-made-simple-sound-classification-step-by-step-cebc936bbe5