Glasovno upravljanje uređajima

Milković, Filip

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:643552

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-18**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

GLASOVNO UPRAVLJANJE UREĐAJIMA

Diplomski rad

Filip Milković

Osijek, 2022.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 19.09.2022.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Filip Milković
Studij, smjer:	Diplomski sveučilišni studij Automobilsko računarstvo i komunikacije
Mat. br. Pristupnika, godina upisa:	D-51ARK, 09.10.2020.
OIB studenta:	42713081538
Mentor:	Doc.dr.sc. Ivan Vidović
Sumentor:	,
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Tomislav Matić
Član Povjerenstva 1:	Doc.dr.sc. Ivan Vidović
Član Povjerenstva 2:	Dr. sc. Petra Pejić
Naslov diplomskog rada:	Glasovno upravljanje uređajima
Znanstvena grana diplomskog rada:	**Procesno računarstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U ovom radu potrebno je istražiti postojeće metode i sustave za prepoznavanje glasa. Nakon istraživanja potrebno je implementirati jednu odabranu metodu ili sustav te omogućiti upravljanje proizvoljno odabranim uređajima putem glasovnih naredbi. Upravljanje je potrebno testirati na hrvatskom i engleskom jeziku.
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	19.09.2022.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 27.09.2022.

Ime i prezime studenta:	Filip Milković
Studij:	Diplomski sveučilišni studij Automobilsko računarstvo i komunikacije
Mat. br. studenta, godina upisa:	D-51ARK, 09.10.2020.
Turnitin podudaranje [%]:	12

Ovom izjavom izjavljujem da je rad pod nazivom**: Glasovno upravljanje uređajima**

izrađen pod vodstvom mentora Doc.dr.sc. Ivan Vidović

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD 1
2. POSTOJEĆA RJEŠENJA 3
3. ZVUK 6
   3.1. Digitalni zvuk 6
   3.2. Osnovna frekvencija 7
   3.3. Kvaliteta zvuka 7
4. NEURONSKE MREŽE ZA PREPOZNAVANJE GOVORA 8
   4.1. Općenito o umjetnim neuronskim mrežama 8
      4.1.1. Aktivacijske funkcije u neuronu 10
      4.1.2. Stopa učenja 15
      4.1.3. Problem nestajućih gradijenata 16
      4.1.4. Optimizatori 18
   4.2. MEL spektrogram i MFCC 19
   4.3. Rekurentna neuronska mreža 21
      4.3.1. Problem dugotrajne ovisnosti informacije 22
   4.4. LSTM mreže 23
   4.5. Konvolucijska neuronska mreža 24
      4.5.1. Arhitektura 24
      4.5.2. Konvolucijski sloj 25
      4.5.3. Sloj sažimanja 26
      4.5.4. Potpuno povezani sloj 26
5. REALIZACIJA SUSTAVA 27
   5.1. Shema sustava 29
   5.2. Upravljanje uređajima *Speech Recognition* modulom 29
      5.2.1. Raspoznavanje govora na engleskom jeziku 30
      5.2.2. Raspoznavanje govora na hrvatskom jeziku 34
      5.2.3. Usporedba između engleskog i hrvatskog jezika 35
   5.3. Upravljanje uređajima konvolucijskom neuronskom mrežom (CNN) 38
      5.3.1. Python moduli neophodni za predobradu i treniranje mreže 38
      5.3.2. Set podataka korišten za treniranje 38
      5.3.3. Programski kod za obradu podataka 39
      5.3.4. Dizajn i arhitektura mreže 42
      5.3.5. Programski kod za treniranje mreže 43
      5.3.6. Rezultat treninga 48

5.3.7. Rezultati mjerenja točnosti izgovorenih riječi 48
5.3.8. Integracija modela i Raspberry Pi uređaja 49
5.3.9. Glavna aplikacija za upravljanje uređajima 50
6. ZAKLJUČAK 56
LITERATURA 58
SAŽETAK 60
ABSTRACT 61
ŽIVOTOPIS 62
PRILOG 7.1. Programski kod - SpeechRecognition Engleski 63
PRILOG 7.2. Programski kod - SpeechRecognition Hrvatski 66
PRILOG 7.3. Programski kod za stvaranje trening značajki 69
PRILOG 7.4. Programski kod za trening modela 72
PRILOG 7.5. Programski kod za stvaranje Lite verzije modela 76
PRILOG 7.6. Programski kod glavne aplikacije za upravljanje uređajima 77

1. UVOD

Raspoznavanje govora je jedna funkcionalnost koju ljudi teško razumiju i ne znaju što predstavlja, a svakodnevno ju koriste. Široka je upotreba uređaja koji koriste raspoznavanje govora, do te razine da ju neki uređaji koriste kao osnovnu funkcionalnost. Kao primjer može se uzeti mobitel, jer većina korisnika Iphone ili Samsung uređaja imaju uključene Siri, odnosno Bixby opcije. To su funkcionalnosti mobilnih uređaja koje stalno moraju prisluškivati okolinu kako bi reagirali na komande buđenja, kao što je „Hey Siri“. Uz to danas sve više ljudi koristi komunikacijske uređaje kao što je Amazon Alexa – Home Assistant ili Google Assistant. Ti uređaji uvelike olakšavaju svakodnevni život brojnim funkcionalnostima. Stoga se ovaj rad bazira na izgradnji i implementaciji jednog takvog uređaja, s kojim se može komunicirati i koji može izvršavati zadatke glasovnim naredbama.

U radu se obrađuju dva načina na koji se može govorom upravljati uređajima. Prvi način je prema već postojećim modulima za raspoznavanje govora koji daju mogućnost odabira jezika. Napravljena je aplikacija koja na temelju izgovorene naredbe upravlja jednostavnim uređajima spojenim na Raspberry Pi računalo. Drugi način upravljanja napravljen je pomoću konvolucijske neuronske mreže, koja je prebačena na Raspberry Pi zajedno s programskim kodom koji ju koristi u svrhu klasificiranja izgovorenih riječi. Za svaku izgovorenu riječ neuronska mreža vraća postotak koliko je sigurna u točnost izgovorene riječi. Na temelju tog postotka programski kod određuje kojim uređajima će upravljati i kako.

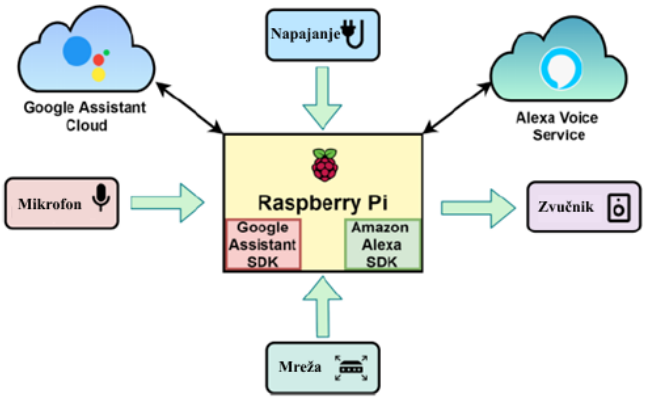
Nakon prvog uvodnog poglavlja, u drugom poglavlju opisuju se postojeća rješenja za upravljanje uređajima pomoću govornih naredbi. Svako rješenje se opisuje i uspoređuje u odnosu na ostala. Treće poglavlje prolazi kroz teoriju zvuka i njegovu reprezentaciju u digitalnom svijetu. Četvrto poglavlje se odnosi na teoriju umjetnih neuronskih mreža. Cijelo poglavlje bazira se na metodama, algoritmima i tehnikama koji su korišteni pri izradi zadatka. Također se spominju mogući problemi prilikom dizajniranja mreže te kako ih se može izbjeći. Bitno je naglasiti da se sve metode i problemi odnose na neuronske mreže koje se isključivo bave audio signalima. U petom poglavlju je opisan i realiziran projektni zadatak. Kao što je ranije rečeno, zadatak je odrađen na dva različita načina s istim hardverskim postavkama. Prvi način je pomoću već postojećeg Python modula koji se povezuje s Google oblakom preko kojeg se koristi već postojeća neuronska mreža. U drugom načinu dizajnirana je neuronska mreža koja zamjenjuje Google mrežu iz prvog primjera. Mreža se prebacuje na uređaj zajedno s programskim kodom koji će ju koristiti, nakon čega se isti kod pokreće i na osnovu izgovorenih naredbi upravlja uređajima. Posljednje poglavlje predstavlja

zaključak u kojem su iznesene činjenice o danom problemu, analiza rezultata te moguća poboljšanja i promjene u sustavu za raspoznavanje govora.

2. POSTOJEĆA RJEŠENJA

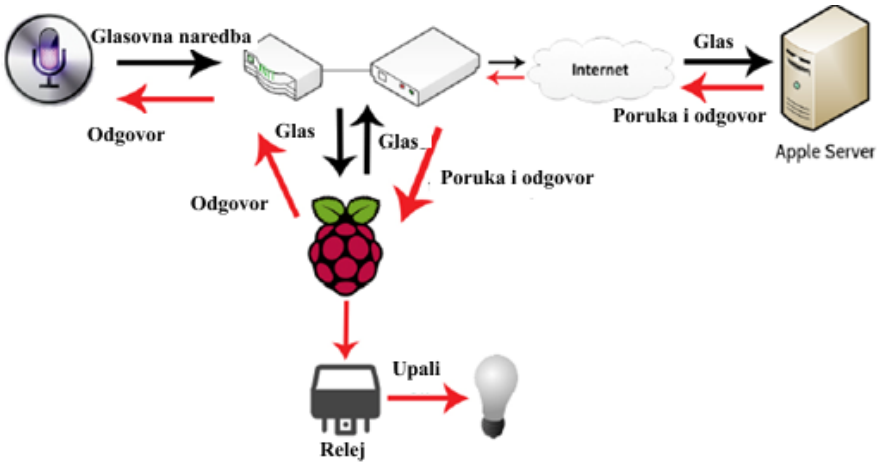
Autori u [1] koriste besplatni Deepspeech servis napravljen od strane Mozzile. Upotreba servisa kao što su *Google Assistant, Amazon Alexa* i sl. može koštati i do 150$ po određenom broju pristupa. Trenutno je dostupno nekoliko verzija Deepspeech-a, s različitim funkcionalnostima. Servis je baziran na rekurentnoj neuronskoj mreži koja na ulazu prima i obrađuje signal u obliku spektrograma. Sagrađen je od pet slojeva od kojih su tri obični slojevi, a neuroni mreže imaju ReLU aktivaciju. Izvršni uređaj je Raspberry Pi 3B+, kojemu je najbolje odgovarala Deepspeech v0.6.0 zbog niske složenosti i male potrošnje resursa.

U [2] također se koristi Rasperry Pi 3B+ verzija, ali u ovom slučaju se pristupa naredbama s *Google Assistant* i *Alexa Voice* oblaka (engl. *cloud*). Sustav pri pokretanju automatski pokreće *Google Assistant SDK* i *Amazon Alexa Service*, koji osluškuju riječ za početak komunikacije. Shema sustava se može vidjeti na slici 2.1. Zvučnik u ovom slučaju služi kao potvrda o izgovorenoj naredbi. Raspberry Pi može upravljati svim pametnim uređajima koji su spojeni na istu mrežu, kao što su pametni prekidači, pametne utičnice, pametne žarulje i sl.



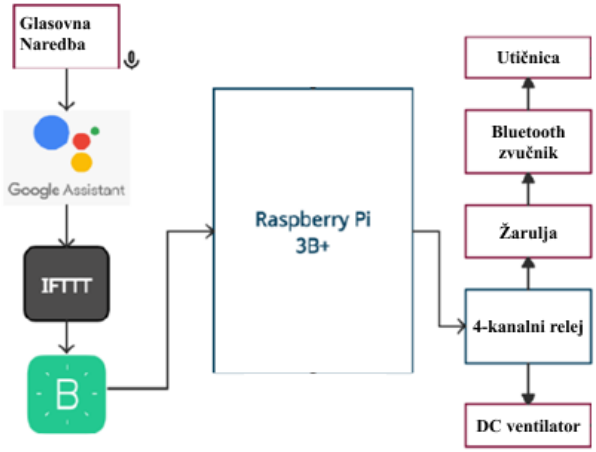
Slika 2.1. Sustav upravljanja uređajima.

Autori u [3] ne koriste fizički mikrofon spojen na Raspberry Pi, nego koriste iOS Siri za upravljanje uređajima kao što su releji i motor driveri. Siri aplikacija generira glasovne pakete od strane korisnika, koji se šalju na Apple server za tekstualni prijevod i za odgovor na poslani upit. Na slici 2.2. je prikazan način rada SiriProxy ekstenzije koja je kreirana za pravljenje biblioteka naredbi, pomoću koje se upravlja GPIO portovima Raspberry Pi uređaja. Sustav je u potpunosti funkcionalan s razinom efikasnosti od 93.33%, pri čemu mu je prosječna latencija 2.12s.



Slika 2.2. SiriProxy proces rada.

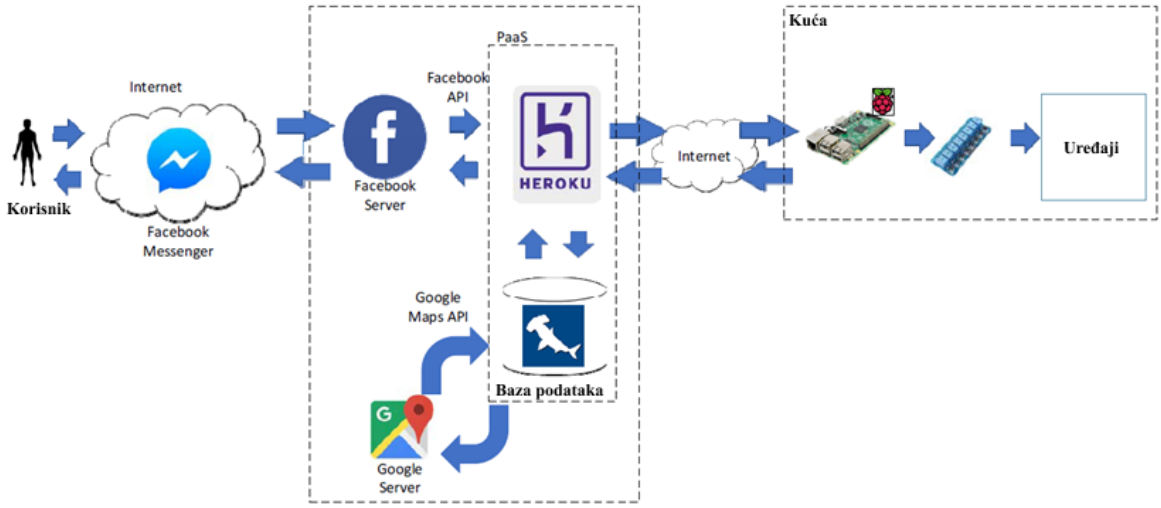
Upravljanje uređajima glasovnim naredbama u [4] je realizirano preko *Google Assistant* servisa. Sustav koristi Raspberry Pi 3B+, *Google Assistant*, Blynk App i IFTTT. Svi spojeni uređaji mogu se upravljati koristeći Smartphone, glasovnim naredbama različitih jezika. Blynk app služi za interakciju s Raspberry Pi te pruža zaštitu sustava. IFTTT pomaže kod povezivanja *Google Assistant* servisa i Blynk app, kako bi se omogućilo upravljanje preko glasovnih naredbi. Za rad se može koristiti onoliko različitih jezika koliko ih ima u IFTTT-u. Kao što se vidi na slici 3.2. za glasovno upravljanje uređajima IFTTT se koristi kao veza između *Google Assistant* i Blynk app. Uređaji kojima se upravlja su utičnica, bluetooth zvučnik, žarulja i DC ventilator preko više-kanalnog releja.



Slika 2.3. Proces od glasovne poruke do izvršnih uređaja za uređaj opisan u [4].

Za razliku od prethodnih primjera, autori u [5] ne koriste direktno glasovne naredbe za upravljanje uređajima. Ideja je da se kućna automatizacija izvodi preko Facebook servera. Korištenjem Chatbot-a koji pruža uslugu grafičkog sučelja te simulira ljudski razgovor. *Messenger platform* je originalni Facebook API korišten u svrhu stvaranja Chatbot-ova za kućnu automatizaciju. Na slici

2.4. se vidi cijeli proces upravljanja udaljenim uređajima. Korisnik komunicira porukama s Chatbot-om koji prosljeđuje te naredbe dalje prema izvršnom uređaju. Budući da mobilni uređaji danas imaju mogućnost raspoznavanja govora, time je omogućeno da se glas pretvara u tekst koji Chatbot može koristiti kao naredbu.



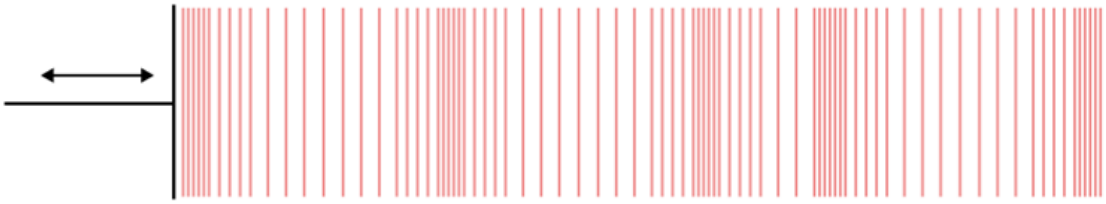
Slika 2.4. Proces upravljanja uređajima.

Problem koji je obrađen u ovom radu, najsličniji je rješenjima opisanim u radovima [1] i [2]. Prva bitna funkcija je obrada zvuka koju detektira mikrofon. Pošto je na Raspberry Pi spojen mikrofon, moraju se koristiti posebni Python paketi koji će obrađivati dobiveni zvuk. Nedostatak *Google Assistance* oblaka je naplata prilikom komercijalne upotrebe, samo u edukacijskim i privatnim svrhama je besplatan. U ovom radu nije korištena rekurentna neuronska mreža iz razloga što cilj rada nije raspoznavanje govora u smislu kreiranja rečenica, nego klasifikacija pojedine izgovorene riječi. Klasificirana riječ predstavlja naredbu na temelju koje se izvodi određena radnja.

3. ZVUK

Riječ zvuk, u svakodnevnom govoru i životu ima vrlo jasno značenje. Predstavlja sve ono što čovjekovo i životinjsko uho čuje. Stručnije rečeno, zvuk je skup longitudinalnih valova (slika 3.1.) koji se emitiraju s određene pozicije u svim smjerovima. Taj val uzrokuje promjene tlaka u prijenosnom mediju što može biti zrak ili voda. Zvuk ne može postojati na mjestima gdje prijenosnog medija nema. Takvi slučajevi su vakuum i svemir, gdje zbog nepostojanja zraka nema promjene tlaka. [6]

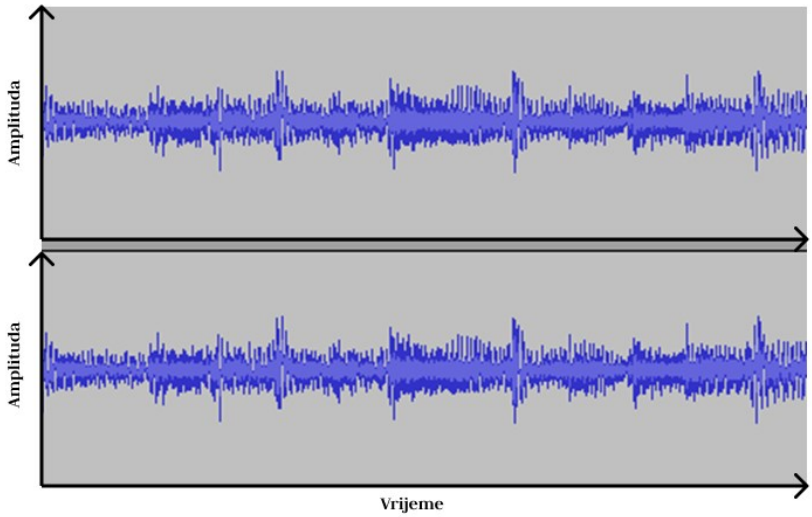
Kada se zvuk generira, stvara se promjena tlaka zraka koju prenosi val. Bubnjić uha detektira tu promjenu tlaka i mozak ju interpretira kao zvuk. [6]



Slika 3.1. Longitudinalni val uzrokuje titranje čestica zraka.

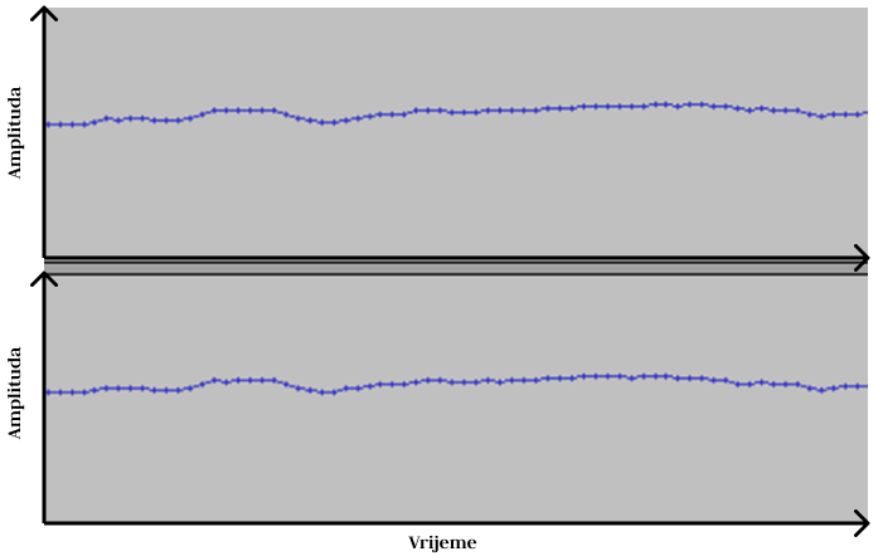
3.1. Digitalni zvuk

Graf valnog oblika za jedan isječak audio signala prikazan je na slici 3.2. Na slici se vide dva valna oblika zato što datoteka sadrži dva audio kanala, što predstavlja stereo snimak. Kod mono snimka postoji samo jedan kanal reprodukcije. [6]



Slika 3.2. Valni oblik stereo audio zapisa.

Uvećanjem jednog dijela iz prethodnog audio zapisa dobije se signal kao na slici 3.3. Svaka prikazana točka predstavlja podatak koji se zove uzorak. Uzorak je amplituda vala u određenom trenutku. [6]



Slika 3.3. Uvećani prikaz audio snimka sa slike 3.2.

U jednoj sekundi audio zapisa, postoji više od tisuću takvih uzoraka. Broj uzoraka koji se uzima po sekundi u audio zapisu zove se brzina uzorkovanja (engl. *sampling rate*) i obično iznosi oko 44100 uzoraka po sekundi (44100 Hz ili 44.1 kHz). Veća brzina uzorkovanja znači više podataka u sekundi, što daje veću vjerojatnost ponovnog sastavljanja audio zapisa. Još neke uobičajene brzine uzorkovanja su 48 kHz, 96 kHz, 192 kHz, 8 kHz. [6]

3.2. Osnovna frekvencija

Osnovna frekvencija tona je broj oscilacija vala u jednoj sekundi. Ako sinusni val oscilira 440 puta u sekundi, uho će prepoznati zvuk koji predstavlja ton *c.* Ako isti sinusni val oscilira 200 puta u sekundi (200Hz) ton će biti dublji, dok suprotno tome ako zvuk oscilira 1200 puta u sekundi (1200Hz) ton će biti viši. [6]

3.3. Kvaliteta zvuka

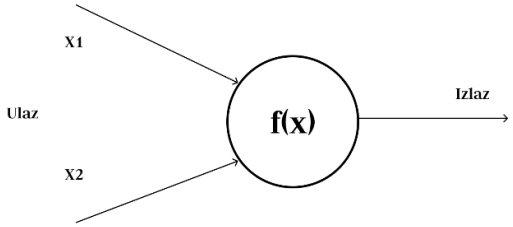
Amplituda vala u određenoj točki obično se mjeri u decibelima. Na raspon i potencijalnu razlučivost amplitude utječe preciznost mjerenja pojedinog uzroka. Kod 8 bitnog valnog oblika, postoji 8 bitova za definiranje zvuka. Kod 16 bitnog, postoji 16 bita za definiranje istog zvuka, što omogućuje detaljniji prikaz signala. Povećanjem bitova za pojedini uzorak povećava se kvaliteta zvuka, ali i veličina same datoteke. [6]

4. NEURONSKE MREŽE ZA PREPOZNAVANJE GOVORA

Umjetne neuronske mreže (engl. *Artificial Neural Networks*) su poprilično jednostavne za razumijevanje, ali su istovremeno komplicirane kada je potrebno realizirati model koji će rješavati dane probleme. Treniranje neuronske mreže može biti zbunjujuće u trenutku određivanja parametara i hiperparametara modela. Stoga su u ovom poglavlju objašnjeni parametri modela za treniranje mreže.

4.1. Općenito o umjetnim neuronskim mrežama

Umjetna neuronska mreža sastoji se od niza jednostavnih umjetnih čvorova koji se zovu neuroni. Umjetni neuron kao i biološki neuron funkcionira tako da se njemu prosljeđuju informacije koje dolaze od izvora signala ili nekog drugog neurona. Svaki neuron ima funkciju u sebi, koja definira prag propagacije signala. Ako suma svih ulaznih signala prelazi tag prag funkcije, neuron propusti signal dalje u sljedeći neuron (slika 4.1.). [7]

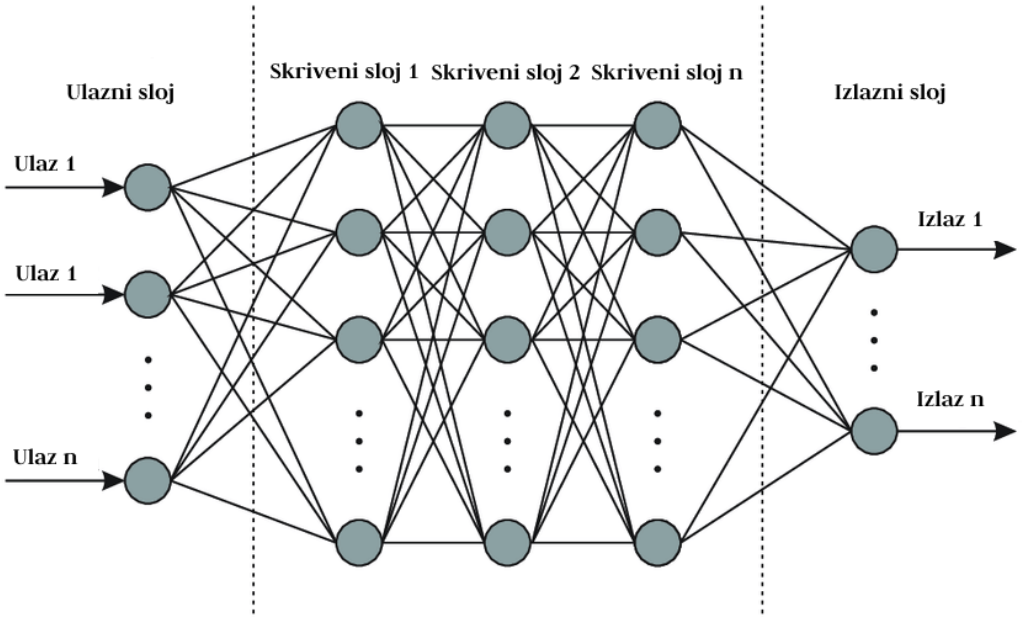


Slika 4.1. Neuron umjetne neuronske mreže.

Neuronska mreža se sastoji od ulaznog, skrivenog i izlaznog sloja (slika 4.2.). Ni jedan sloj nema definiranu granicu koliko bi neurona trebao sadržavati. Ulazni sloj predstavlja sve ulazne signalne koji se predaju neuronskoj mreži na procesiranje, odnosno to su podaci prema kojima se radi predikcija. Skriveni sloj predstavljaju sve neuronske veze između ulaza i izlaza mreže. Broj skrivenih slojeva ovisi o problemu koji se rješava, odnosno o arhitekturi mreže. Uvijek se želi postići najveća učinkovitost mreže, stoga sama arhitektura mora biti kvalitetno dizajnirana. [7]

U većini slučajeva mreža s pet skrivenih slojeva je dovoljna, ali kod problema kao što su procesiranje slika i govora potrebne su mreže sa stotinama slojeva. Za te slučajeve koriste se već postojeći modeli kao što su *YOLO*, *ResNet*, *VGG*, koji dozvoljavaju korištenje svojih dijelova kako bi se istrenirali modeli za specifične slučajeve. Ovom metodom se u konačnici trenira samo nekoliko slojeva koji su dodani naknadno. [8]

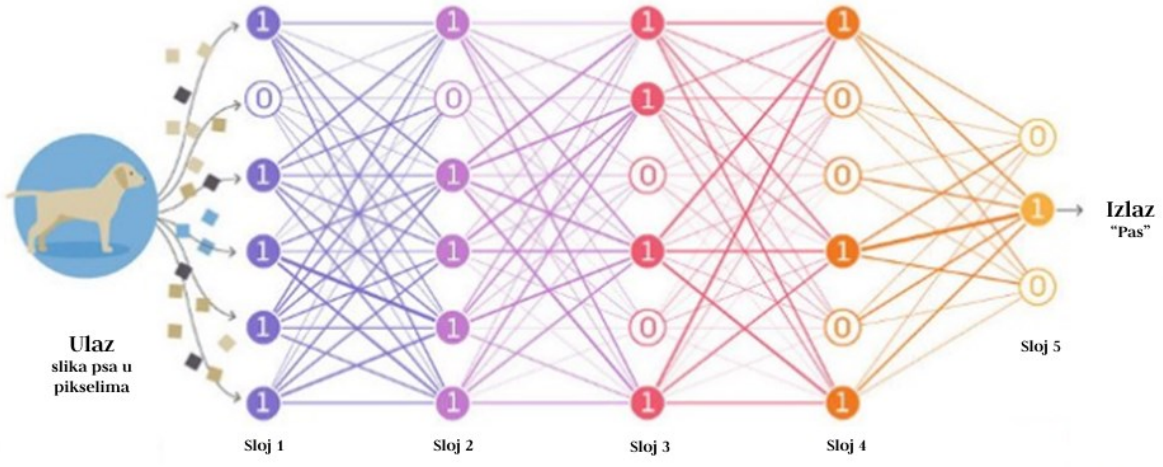
U praksi je dokazano da će mreža biti učinkovitija ako postoji više skrivenih slojeva, a manje neurona u samom sloju, nego da postoji manji broj slojeva koji sadržavaju tisuće neurona. [8]



Slika 4.2. Slojevi umjetne neuronske mreže.

Izlazni sloj sadrži neurone koji određuju rezultat temeljen na ulaznom sloju. Svaki neuron predstavlja jednu predikciju koju je mreža pretpostavila. Što znači da će mreža u svakom slučaju uvijek dati neko rješenje prilikom uzimanja ulaznih podataka. [8]

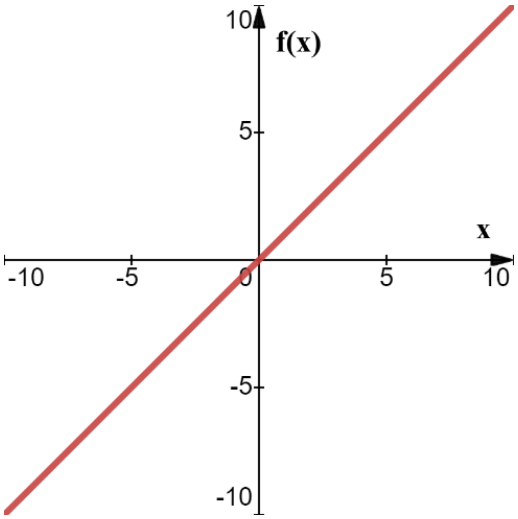
Postoje dvije vrste izlaza umjetne neuronske mreže, a to su regresija i klasifikacija. Regresijske predikcije su podaci kao predviđanje cijena stanova, cijena automobila, potrošnje goriva itd. Klasifikacijske predikcije su slučajevi kao određivanje spam poruka (je ili nije spam – binarna klasifikacija) ili određivanje instanci objekata kao što su auto, pas, mačka itd. (višeklasna klasifikacija). Također kod klasifikacije u izlaznom sloju postoji onoliko neurona koliko je klasa. Za slučaj gdje se trenira raspoznavanje automobila, psa i mačke postojati će tri izlazna neurona (slika 4.3.). [8]



Slika 4.3. Višeklasna klasifikacija.

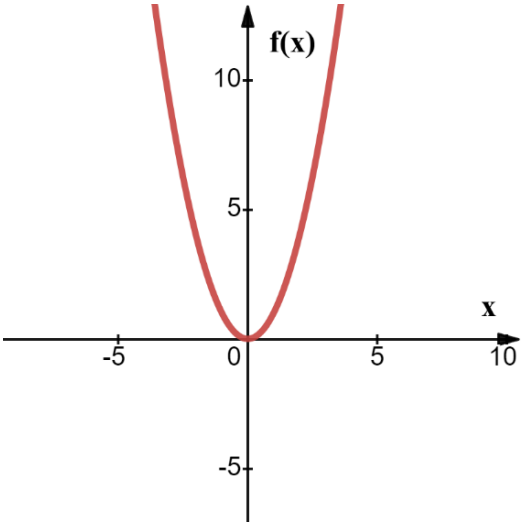
4.1.1. Aktivacijske funkcije u neuronu

Generalno, aktivacijske funkcije se dijele u dvije grupe, a to su linearne i nelinearne aktivacijske funkcije. Linearna aktivacijska funkcija može se vidjeti na slici 4.4. i obilježava ju značajka da joj izlaz neograničen. [9]



Slika 4.4. Linearna aktivacijska funkcija f(x) = x.

Nelinearne aktivacijske funkcije se češće koriste od linearnih, zato što je nelinearnost uobičajena u stvarnim problemima. Stoga se nelinearni model lakše prilagođava različitim ulaznim podacima (slika 4.5.) [9]. Neke od najpoznatijih aktivacijskih funkcija objašnjenje su u nastavku.



Slika 4.5. Nelinearna aktivacijska funkcija.

Sigmoid funkcija

Glavni razlog korištenja sigmoid funkcije je taj što joj izlaz postoji između nula i jedan, stoga je najčešće korištena kod modela za binarnu klasifikaciju. Ako je vrijednost ispod nekog praga 0.5,

izlazni signal iz funkcije je prema slici 4.6. sigurno nula. S druge strane za ulaz vrijednosti pet i više na izlazu daje vrijednost jedan [9]. Jednadžba funkcije i njena derivacija su prikazane izrazima (4-1) i (4-2).

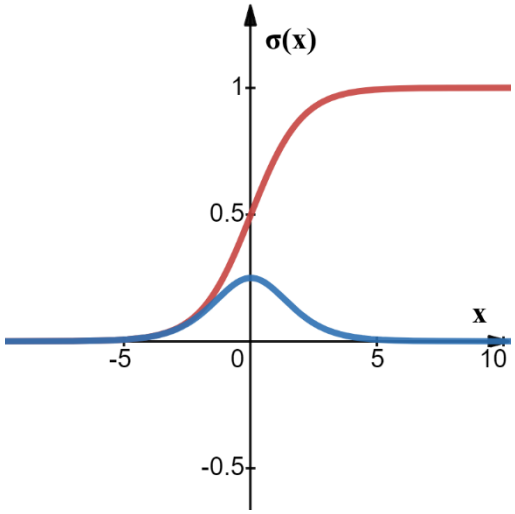
Jednadžba funkcije:

$$\sigma(x) = 1/\,(1 + \ e^{-x}) \tag{4-1}$$

Jednadžba derivacije:

$$\sigma'(x) = \sigma(x) * (1 - \sigma(x)) \tag{4-2}$$

Derivacija funkcije se koristi kod *backpropagation* metode radi ažuriranja i prilagodbe parametara modela. Derivacija će dati nagib grafa koji funkcija opisuje, a ta će vrijednost obavijestiti mrežu treba li povećati ili smanjiti vrijednost pojedinih težina u slojevima. Ovaj opis vrijedi za sve derivacije koje se spominju u ostatku diplomskog rada.



Slika 4.6. Sigmoid funkcija (crvena) i njena derivacija (plava).

Tanh funkcija

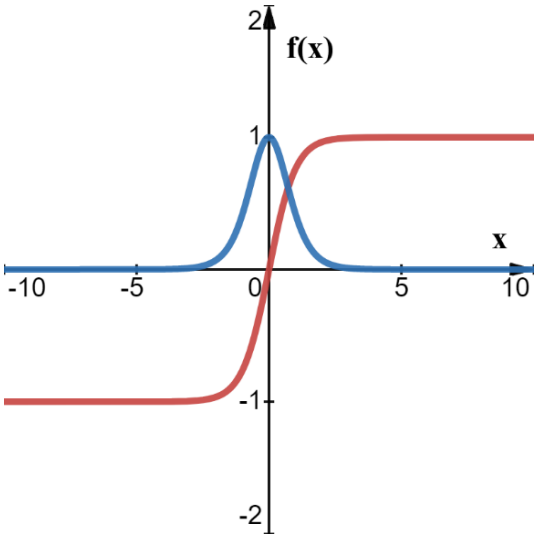
Tanh ili hiperbolički tangens je generalno ista funkcija kao i sigmoid. Prema slici 4.7. se može vidjeti da je izlaz *tanh* funkcije u granicama [-1,1], stoga joj je prednost mapiranje negativnih ulaza u negativne izlaze, vrijednosti oko nule u nulu i pozitivne ulaze u pozitivne izlaze. Ova funkcija se najčešće koristi kod višeklasne klasifikacije [9]. Jednadžba i derivacija funkcije su prikazane izrazima (4-3) i (4-4).

Jednadžba funkcije:

$$\tanh(x) = (e^{x} - e^{-x})/(e^{x} + e^{-x}) \tag{4-3}$$

Jednadžba derivacije:

$$tanh'(x) = 1 - \tanh(x)^2 \tag{4-4}$$



Slika 4.7. Tanh funkcija (crvena) i njena derivacija (plava).

ReLU funkcija

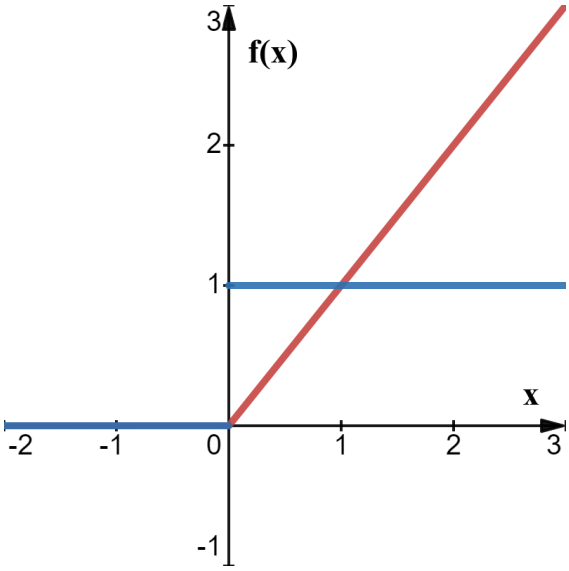
ReLU (engl. *Rectified Linear Unit*) je najčešće korištena aktivacijska funkcija, pogotovo kod konvolucijskih neuronskih mreža i dubokog učenja. Prema slici 4.8. se može vidjeti da je izlaz ove funkcije nula kod svih negativnih ulaza, dok je kod pozitivnih ulaza izlaz jedak ulaznom signalu. Problem ove funkcije je što svi negativni ulazi odmah daju nulu na izlazu, što smanjuje mogućnost ispravnog treniranja modela nad skupom podataka [9]. Jednadžba *ReLU* funkcije prikazana je izrazom (4-5), a derivacija izrazom (4-6).

Jednadžba funkcije:

$$f(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases} \tag{4-5}$$

Jednadžba derivacije:

$$f'(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases} \tag{4-6}$$



Slika 4.8. *ReLU* aktivacijska funkcija (crvena) i njena derivacija (plava).

Leaky ReLU funkcija

Leaky ReLU rješava prethodni problem tzv. mrtvi *ReLU* problem, na način da izlazi za negativne vrijednosti budu negativne vrijednosti, a ne nule kao kod prethodne funkcije. To povećava granice izlaza s $[0, \infty]$ na $[-\infty, \infty]$ [9]. Jednadžba funkcije i njena derivacija dane su izrazima (4-7) i (4-8).

Jednadžba funkcije:

$$f(x) = \begin{cases} x, & x > 0 \\ \alpha x, & x \leq 0 \end{cases} \tag{4-7}$$

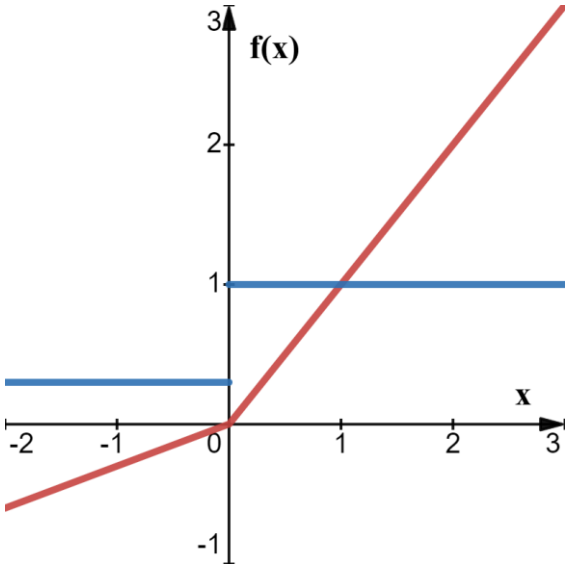
Jednadžba derivacije:

$$f'(x) = \begin{cases} 1, & x > 0 \\ \alpha, & x < 0 \end{cases} \tag{4-8}$$

gdje je:

- a – vrijednost koja omogućuje da negativni ulazi daju izlaz manji od nule. Obično iznosi 0.01.

Na slici 4.9. α je postavljen na 0.3 radi bolje vizualizacije funkcije u koordinatnom sustavu.

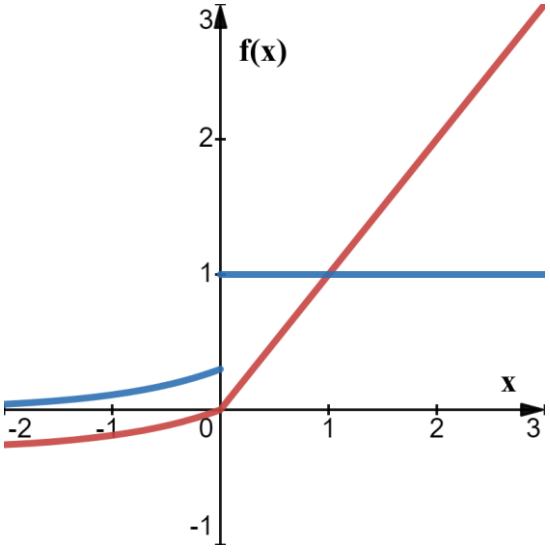


Slika 4.9. *Leaky ReLU* aktivacijska funkcija (crvena) i njena derivacija (plava).

ELU funkcija

ELU (engl. *Exponential Linear Unit*) je najbolja inačica *ReLU* funckije. U odnosu na ostale ima α parametar koji je pozitivan broj (slika 4.10.). Najveća prednost ove funkcije je što se u negativnom dijelu izlaz polako ispravlja sve dok ne dostigne vrijednost $-\alpha$ [9]. *ELU* jednadžbe su prikazane izrazima (4-9) i (4-10).

Na slici 4.10. α je postavljen na 0.3 radi bolje vizualizacije funkcije u koordinatnom sustavu.



Slika 4.10. *ELU* aktivacijska funkcija (crvena) i njena derivacija (plava).

Jednadžba funkcije:

$$f(x) = \begin{cases} x, & x > 0 \\ \alpha(e^x - 1), & x \leq 0 \end{cases} \tag{4-9}$$

Jednadžba derivacije:

$$f'(x) = \begin{cases} 1, & x > 0 \\ \alpha e^x, & x < 0 \end{cases} \tag{4-10}$$

gdje je:

- a – vrijednost koja omogućuje da negativni ulazi daju izlaz manji od nule. Obično iznosi 0.01.

Softmax funkcija

Softmax aktivacijska funkcija izračunava distribuciju vjerojatnosti događaja kod *n* različitih događaja. Generalno, ova funkcija računa vjerojatnost svake klase za sve moguće klase (4-11). Kasnije pomoću izračunatih vrijednosti određuje ciljne klase za ulazni skup podataka (višeklasna klasifikacija) [9].

$$S(y_i) = e^{y_i} / \sum_{k=1}^{K} e^{y_k}, \qquad za\ i = 1, 2, \dots, K \tag{4-11}$$

4.1.2. Stopa učenja

Stopa učenja (engl. *learning rate*) je parametar podešavanja koji u algoritmu optimizacije predstavlja veličinu koraka pri svakoj iteraciji kretanja prema minimumu funkcije gubitaka. Za pronalazak najbolje stope učenja treba krenuti od jako malih vrijednosti (npr. 10^{-6}) i polako ju množiti konstantom dok se ne postigne jako velika vrijednost (npr. 10). Tijekom učenja mreže treba pratiti razvoj modela kako bi se utvrdila najbolja stopa učenja za dani problem. Zatim se ponovo pokrene trening modela, ali ovaj put s idealnom stopom učenja. Dobra stopa učenja je obično pola vrijednosti stope učenja koja uzrokuje probleme odstupanja modela [8]. Izgled grafa dobre stope učenja se može vidjeti na slici 4.11.

Slika 4.11. Prikaz pogreške za određenu stopu učenja.

4.1.3. Problem nestajućih gradijenata

Kako neuronske mreže s više slojeva koriste određene aktivacijske funkcije, gradijenti funkcije približavaju se nuli. To je problem koji uvelike otežava treniranje mreže, jer može uzrokovati da dio mreže, pa i cijela mreža zamre i na izlazu počne davati odstupanje, odnosno pogrešku. [10]

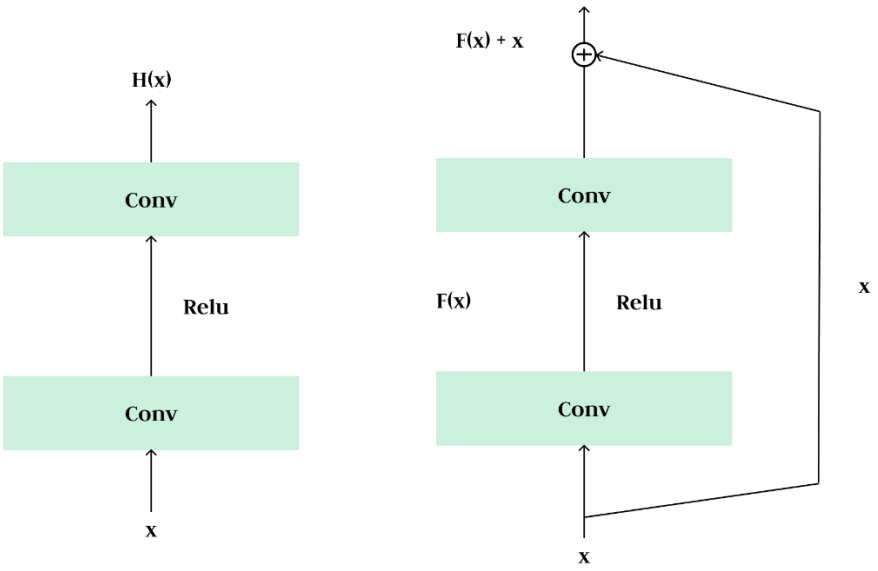
Postoji nekoliko načina za rješavanje ovog problema, a to su: korištenje drugih aktivacijskih funkcija, korištenje rezidualnih mreža, rano zaustavljanje, batch normalizacija i izbacivanje.

a. Druge aktivacijske funkcije

Najjednostavnija i praktična metoda rješavanja problema nestajućih gradijenata je promjena aktivacijske funkcije samog neurona. Do sada se na rješavanju ovog problema najbolje pokazala *ReLU* funkcija, ali i sve njene inačice. [10]

b. Rezidualne mreže

Rezidualne mreže rade na principu rezidualnih blokova (slika 4.12.). Rezidualni blok zbraja ulaznu vrijednost direktno s izlazom, odnosno preskače određene slojeve i aktivacijske funkcije koji su u mogućnosti uzrokovati nestajanje gradijenta. [10]



Slika 4.12. Usporedba bloka klasične mreže (lijeva) i bloka rezidualne mreže (desna).

a. Rano zaustavljanje

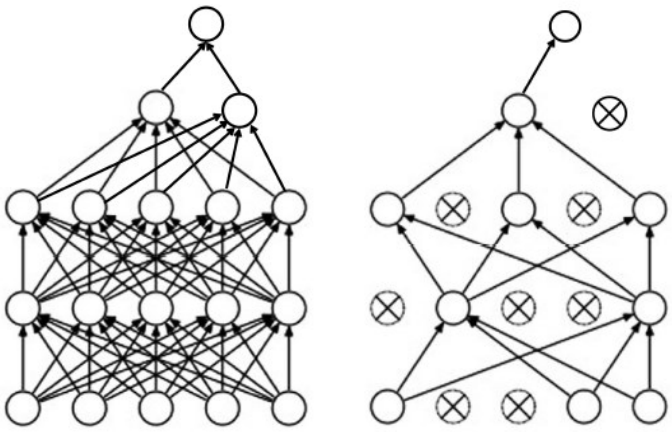
Ova metoda se primjenjuje kod mreža s velikim brojem sakrivenih slojeva, kod mreža s velikim brojem neurona te kod treniranja s velikim brojem epoha. Radi na način da se trening prekida u trenutku kada se prestane poboljšavati karakteristika modela. [10]

b. Batch normalizacija

Radi na principu nultog centriranja (engl. *zero-centering*) i normaliziranja ulaznih vektora koji se zatim skaliraju i premještaju. Ponaša se kao regulator što znači da nisu potrebna L2 regularizacija i izbacivanja (engl. *dropout*). Ova metoda omogućuje korištenje većih stopa učenja, što daje bržu konvergenciju. Jedina i glavna mana joj je povećanje vremena prolaska kroz jednu epohu zbog dodatnih izračuna u svakom sloju. Obično se ubacuje ispred povezanog sloja, konvolucijskog sloja ili prije nelinearnosti. [10]

c. Izbacivanje

Izbacivanje (engl. *dropout*) je jedna on najčešće korištenih pristupa za regularizaciju. Ova vrlo jednostavna tehnika radi na principu nasumičnog isključivanja neurona u svakom sloju tijekom svake epohe treninga (slika 4.13.). Ovim pristupom se želi ravnomjerno raspodijeliti moć odabira značajki na sve neurone i prisiliti model da trenira neovisne značajke. To čini mrežu robusnijom jer se ne može osloniti na određeni skup ulaznih neurona za predviđanje izlaza. Ova metoda također smanjuje prilagođavanje modela trening skupu (engl. *overfitting*) te najveću primjenu ima kod treniranja dubokih mreža. [10]



Slika 4.13. Mreža bez isključivanja (lijeva) i mreža sa isključivanjem (desna). [10]

4.1.4. Optimizatori

Optimizatori su algoritmi, odnosno metode koje se koriste za modifikaciju parametara, kao što su težine i stopa učenja radi smanjenja pogreške i povećanja točnosti prilikom treniranja neuronske mreže. [10]

Neke od metoda optimizacije opisane u nastavku su: Metoda najbržeg spusta, Adagrad, Adam, RMSprop… Uz njih druge često korištene metode su: SGD, SGD s momentom, AdaDelta, Nadam, itd…

a. Metoda najbržeg spusta

Metoda najbržeg spusta (engl. *gradient descent*) je najkorištenija i najosnovnija metoda za optimizaciju mreže. Koristi se kod linearne regresije i algoritama klasifikacije. Predstavlja optimizaciju prvog reda te stoga ovisi o izvodu funkcije prvog reda. Izračunava na koji se način težine trebaju promijeniti kako bi funkcija dosegla svoj minimum. Često koristi metodu povratne propagacije (engl. *backpropagation*), koja pogrešku vraća nazad kroz mrežu kako bi se popravili parametri koji su dali loše rezultate na izlazu [10]. Ažuriranje parametara se izvodi izrazom (4-12).

$$\theta = \theta - \alpha * \theta \nabla J(\theta) \tag{4-12}$$

b. AdaGrad

Glavni problem većine optimizacijskih tehnika je konstantna stopa učenja prilikom treninga. Ova metoda prilikom treniranja mijenja stopu učenja na način da pravi velike promjene parametara kod onih parametara koji se manje pojavljuju, a male promjene parametara kod onih parametara koji se često pojavljuju. [10]

c. Adam

Adam (engl. *Adaptive moment estimation*) je algoritam koji radi estimaciju momenta prvog i drugog reda. Ideja kod ove metode je ne raditi velike promjene jer to može uzrokovati preskakanje minimuma. Iz tog razloga je bolje smanjiti brzinu ažuriranja parametara kako bi se točnije pronašao minimum funkcije. [10]

d. RMSprop

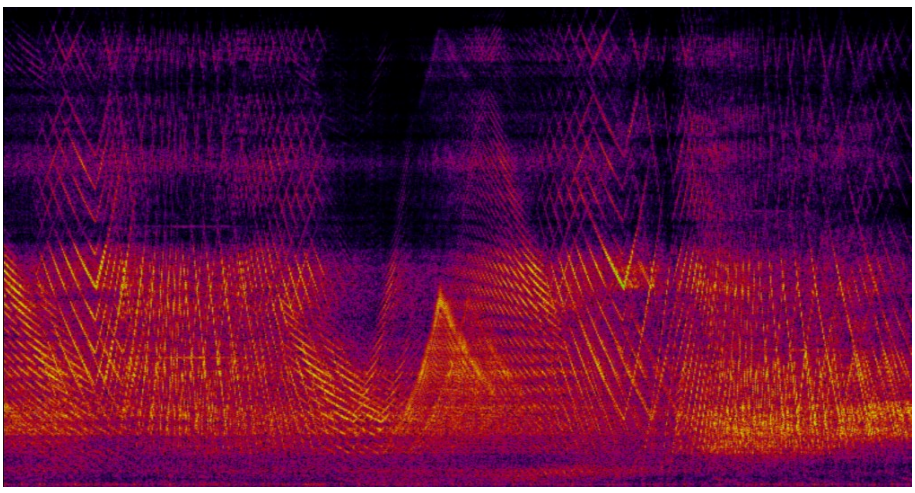
Glavna ideja ovog algoritma je pričuvati prosjek kvadratnih gradijenata za svaku težinu te podijeliti gradijent s korijenom srednje kvadratne vrijednosti. Iz tog razloga se zove RMS (engl. *root mean square*), a jednadžba ažuriranja parametara dana je izrazom (4-14) [10]

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta)\left(\frac{\partial C}{\delta w}\right)^2 \tag{4-13}$$

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{E[g^2]_t}}\frac{\delta C}{\delta w} \tag{4-14}$$

4.2. MEL spektrogram i MFCC

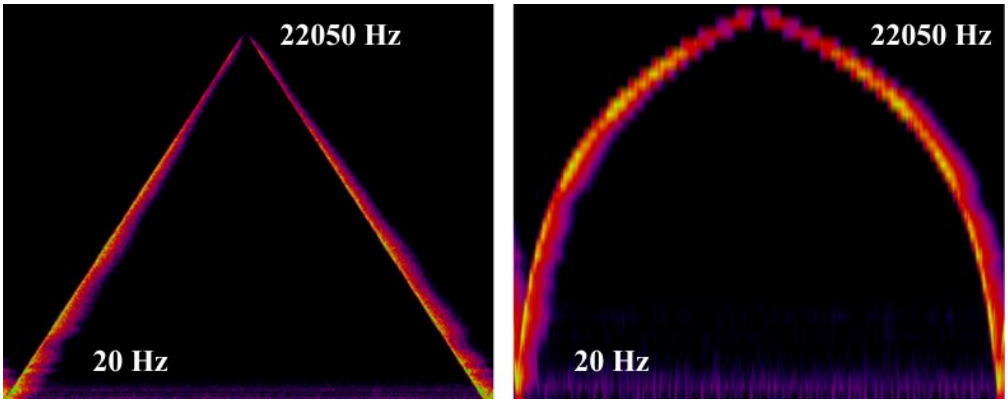
Spektrogram je način vizualizacije audio sadržaja u 2D domeni, gdje x-os predstavlja vrijeme audio zapisa, a y-os predstavlja frekvenciju. Boja predstavlja amplitudu koja ovisi o vremenu i frekvenciji (slika 4.13.). Spektrogram može pomoći u prepoznavanju problema sa zvukom, kao što su nisko i visoko frekvencijski šumovi te šumovi kratkih impulsa (npr. pucketanje). Iz jednog uzorka zvuka spektrogram se dobije primjenom diskretne kosinusne transformacije (engl. *discrete cosine transform - DCT*). DCT izračunava frekvencijske komponente audio signala i pretvara signal u niz amplituda na komponentnim frekvencijama. [11]



Slika 4.13. Spektrogram. [11]

Mel skala je skala tonova koje ljudski sluh općenito percipira kao međusobno jednako udaljene tonove. Kako frekvencija raste, interval između vrijednosti u Mel skali se povećava. Mel spektrogram preslikava vrijednosti u Hz na Mel skalu, odnosno logaritamski prikazuje frekvencije iznad određenog praga. Naziv Mel potječe od riječi melodija i ukazuje na to da se ljestvica temelji na usporedbi između visine tonova. Npr. u linearnom spektrogramu okomiti prostor između 1000 Hz i 2000 Hz je pola okomitog prostora između 2000 Hz i 4000 Hz. U Mel spektrogramu razmak između tih raspona je približno isti. Ovo skaliranje je analogno ljudskom sluhu, gdje se lakše razlikuju slični zvukovi niskih frekvencija u odnosu na slične zvukove visokih frekvencija.

Slika 4.14. prikazuje razliku između linearnog i Mel spektrograma istog tona na različitim frekvencijama. Frekvencija prvo raste s početne vrijednosti od 20 Hz na 22050 Hz, nakon čega pada ponovo na 20 Hz. Lijeva slika prikazuje objektivni signal, dok desna slika prikazuje ljudsku percepciju. Krivulja se na vrhu izravnava što ukazuje na smanjenu diferencijaciju između visokih frekvencija. [12]



Slika 4.14. Linearni spektogram (lijevi) i MEL spektrogram (desni).

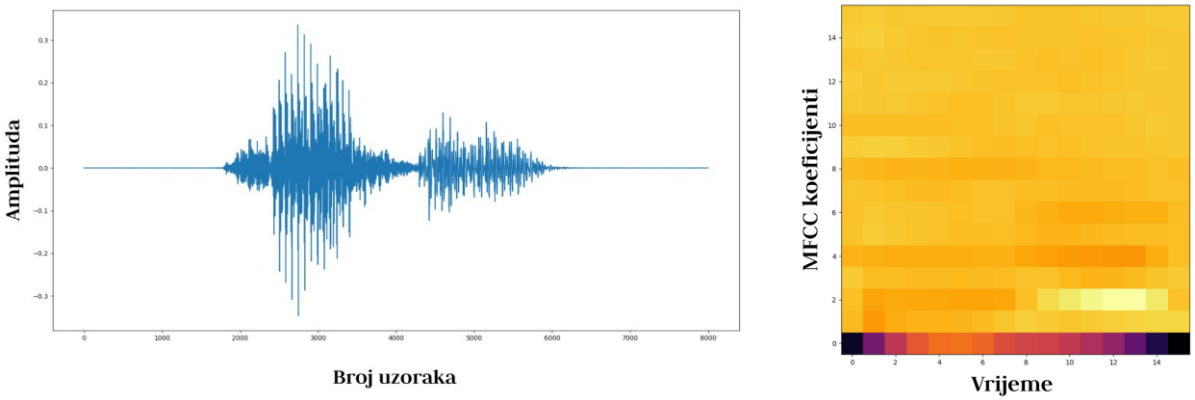
Mel frekvencijski kepstralni koeficijenti (engl. *Mel Frequency Cepstral Coefficients*) ili MFCC predstavljaju spektralne audio značajke. Izračun značajki ovog tipa odvija se u nekoliko koraka. Prvo se signal podijeli na okvire fiksne duljine primjenom funkcije prozora (engl. *windowing*). Cilj je modelirati male dijelove signala (npr. 20ms) koji su stacionarni. Primjenom funkcije prozora (obično Hamming prozor) uklanjaju se rubni efekti, čime se generira kepstralni vektor značajki za svaki okvir. [13]

Sljedeći korak je primjena Diskretne Fourierove Transformacije (DFT) za svaki okvir. Time se zadržava samo logaritam spektra amplitude. Logaritam se uzima jer je dokazano da je percipirana glasnoća signala približno logaritamska. [13]

Nakon DFT treba izgladiti i naglasiti perceptivno značajne frekvencije. To se odrađuje uzorkovanjem većeg broja spektralnih komponenti (npr. 256) na manji broj komponenata (npr. 40), prema Mel frekvencijskoj skali. [13]

Zbog smanjenja broja parametara u sustavu, zadnji korak je primjena transformacije na Mel-spektralne vektore koja uklanja korelaciju vektorskih komponenata. To se može obaviti Karhunen-Loeve (KL) transformacijom, koja se aproksimira diskretnom kosinusnom transformacijom. KL transformacija za svaki okvir računa 13 ili više kepstralnih značajki. [13]

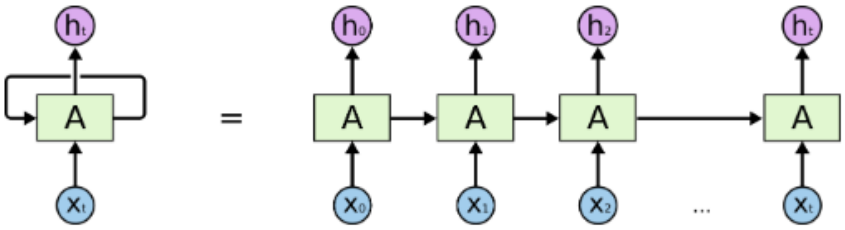
Slika 4.15. prikazuje jedan audio signal engleske riječi sretan (engl. *happy*), a odmah do nje je slika koja prikazuje MFCC sa 16 koeficijenata istog audio signala.



Slika 4.15. Valni oblik signala riječi *happy* i njegov MFCC.

4.3. Rekurentna neuronska mreža

Rekurentne neuronske mreže (engl. *Recurrent neural networks – RNN*) su mreže koje u sebi sadrže povratnu petlju. Time ne dozvoljavaju informaciji da ode, nego ju koriste za sljedeću predikciju. [14]



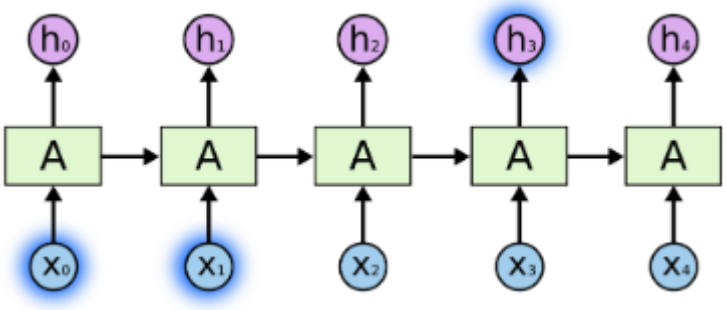
Slika 4.16. Razlika između rekurentnog i klasičnog bloka neuronske mreže. [14]

Na slici 4.16. se s lijeve strane vidi dio rekurentne neuronske mreže, gdje A uzima ulaz x_t i daje izlaz h_t. Povratna petlja ovdje dozvoljava prijenos informacija iz jednog koraka mreže u drugi. Također je vidljivo da se rekurentne neuronske mreže ne razlikuju od klasičnih neuronskih mreža.

Iz tog razloga se dijelovi mreže mogu prikazati kao iste kopije razvučene u niz, od kojih svaka prenosi informaciju u sljedeći dio. [14]

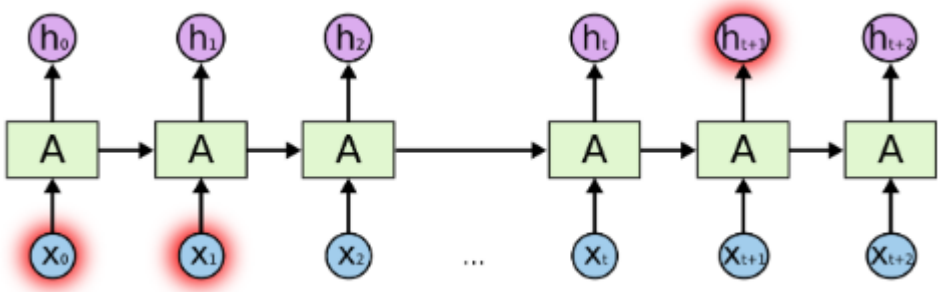
4.3.1. Problem dugotrajne ovisnosti informacije

U nekim slučajevima za dobivanje predikcije mreža uzima informacije u petlji koje su se dogodile u bližoj prošlosti, odnosno izvršava predikciju na osnovi prošlih riječi. Npr. „Po moru plovi brod“ je rečenica gdje je vrlo jednostavno predvidjeti zadnju riječ na osnovi prijašnjih, a da se ne gleda daleko u prošlost prošlih riječi. [14]



Slika 4.17. Primjer kratkotrajne ovisnosti riječi u rečenici. [14]

Za razliku od primjera prikazanog na slici 4.17., na sljedećoj slici (slika 4.18.) vidljiv je primjer gdje je potrebno više konteksta za predikciju. Npr. „Odrastao sam u Francuskoj… Stoga pričam solidno francuski“. U ovom slučaju za riječ francuski mora se znati iz koje je zemlje jezik kako bi se izvršila predikcija. Kod velikih razmaka između predikcije i konteksta klasične RNN ne daju dobre rezultate. [14]



Slika 4.18. Primjer dugotrajne ovisnosti riječi u rečenici. [14]

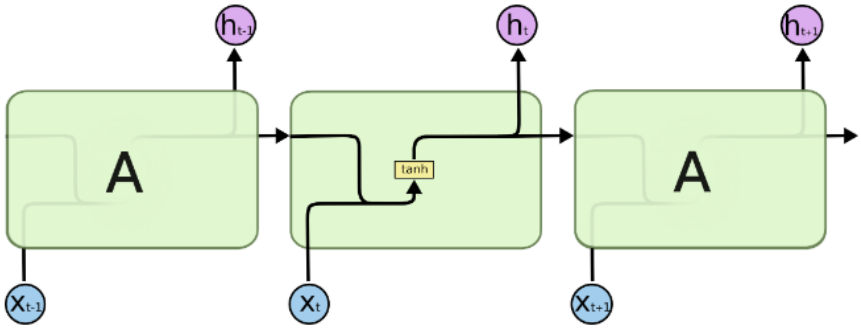
U teoriji RNN su u potpunosti sposobne rješavati probleme ovakvog tipa, ali u praksi zapravo ne daju dobre rezultate. Stoga se koristi posebna vrsta rekurentne mreže koja nema ovakvih problema, a zove se mreža dugo-kratkoročnog pamćenja (engl. *Long Short Term Memory – LSTM*) [14]. Ovaj

tip mreže najviše primjene ima u rješavanju problema kao što su raspoznavanje govora, raspoznavanje jezika, prevođenje…

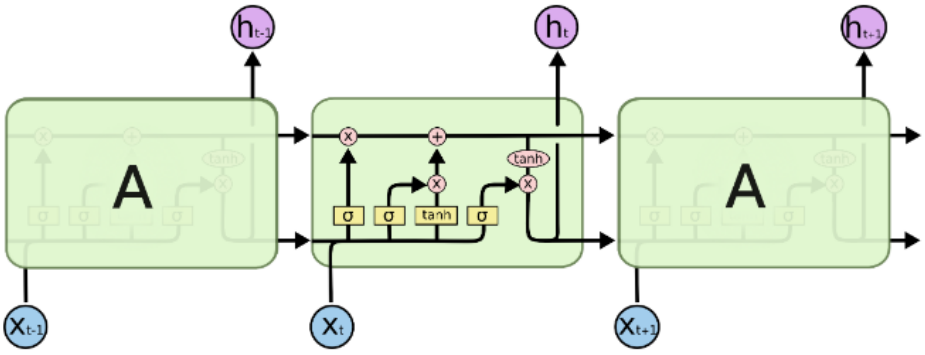
4.4. LSTM mreže

LSTM mreže su specijalne vrste RNN, koje su sposobne učiti na primjerima dugoročnih ovisnosti. Specijalno su dizajnirane kako bi izbjegle problem dugoročnih veza. Pamćenje informacije na duži vremenski period je njihova osnovna svrha. [14]

Kod standardnih rekurentnih mreža modul s povratnom petljom je jednostavne strukture, jedan sloj s *tanh* aktivacijskom funkcijom (slika 4.20.). LSTM moduli s povratnom petljom umjesto jednog sloja imaju četiri koji rade na specifičan način. [14]



Slika 4.20. Standardni rekurentni modul.



Slika 4.21. LSTM modul.

LSTM ima mogućnost uklanjanja ili dodavanja informacije u akumulator stanja (engl. *cell state*), pomoću ulaznih struktura (engl. *gate*). Te strukture su obično put kojim se propušta informacija, a sastoje se od sigmoid aktivacija i operacija množenja (slika 4.21.). Sigmoid sloj na izlazu daje broj između nula i jedan, koji govori koliko svake komponente treba propustiti. Nula ne propušta ništa, dok jedan propušta sve. LSTM ima tri takva ulaza koja pružaju zaštitu i kontrolu akumulatorima stanja. [14]

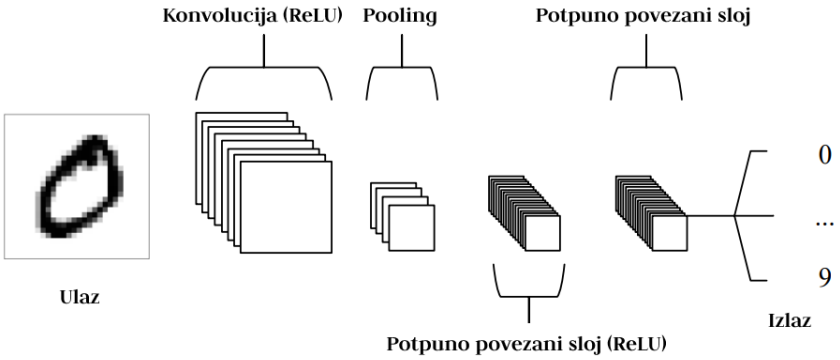
4.5. Konvolucijska neuronska mreža

Konvolucijske neuronske mreže (engl. *Convolutional neural network – CNN*) najveću primjenu imaju u raspoznavanju i detekciji značajki slike i videa, ali su svoju primjenu našle i kod prepoznavanja govora. Jedna od glavnih mana klasičnih neuronskih mreža je što troše puno procesorskog vremena prilikom obrade slike. Na primjer za sliku 64 x 64 koja je u boji, jedan neuron u prvom sloju će obrađivati 12288 (64 x 64 x 3) parametara. [15]

CNN arhitektura se određuje prema specifičnom tipu ulaznog podatka koji će se obrađivati. Što znači da će na primjeru slike, neuron u sloju biti organiziran u tri dimenzije. Prostornu dimenzionalnost ulaza (visina i širina – 2D) i dubina (boja) pojedinog piksela. U praksi bi to značilo da će za ranije navedeni ulazni podatak (64 x 64 x 3), izlazni podatak izgledati 1 x 1 x n, gdje je n broj mogućih klasa. [15]

4.5.1. Arhitektura

CNN su sastavljene od tri tipa slojeva. Konvolucijski slojevi, slojevi sažimanja i potpuno povezani slojevi. Kada se ova tri sloja povežu jedan za drugim, formira se konvolucijska neuronska mreža. Na slici 4.22. se vidi jedna jednostavna CNN arhitektura koja obrađuje, odnosno klasificira jedan broj od nula do devet iz MNIST[1] baze podataka. [15]



Slika 4.22. Jednostavna konvolucijska mreža s pet slojeva.

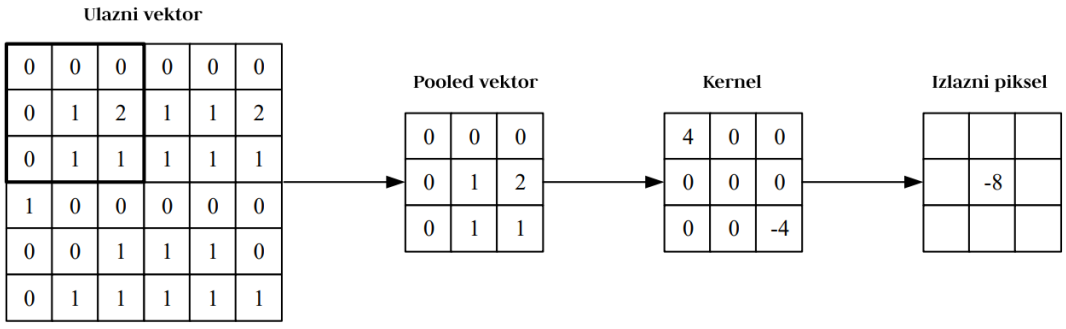
Primjer sa slike 4.22. može biti jednostavno objašnjen u nekoliko koraka.

[1] Baza podataka slika 28x28, ručno pisanih brojeva.

1. Ulazni sloj (ulaz) je slika s određenim brojem piksela koji imaju svoje numeričke vrijednosti. [15]
2. Konvolucijski sloj će odrediti izlaz iz neurona na temelju ulaza, na način da će izvršiti izračun skalarnog produkta između njihovih težina i područja povezanih ulazom. ReLU ima cilj primijeniti aktivacijsku funkciju na izlaz koji je proizveo prethodni sloj. [15]
3. Sloj sažimanja ima jedinstvenu zadaću smanjivanja uzoraka duž prostorne dimenzionalnosti danog ulaza, čime se dodatno smanjuje broj parametara unutar te aktivacije. [15]
4. Potpuno povezani sloj će tada izvršiti svoju standardnu operaciju kao i kod klasičnih neuronskih mreža te će proizvesti rezultat pojedine klase s kojim će se izvršiti klasifikacija. [15]

4.5.2. Konvolucijski sloj

Parametri ovog sloja fokusirani su na upotrebu kernela[2] koji se mogu naučiti. Kerneli su obično male prostorne dimenzionalnosti koji se šire duž dubine ulaza. Kada podatak dođe do konvolucijskog sloja, sloj odrađuje konvoluciju za svaki filtar preko ulaza kako bi izradio 2D aktivacijsku mapu. Prelaskom preko ulaznog podatka, skalarni produkt se računa za svaku vrijednost u tom kernelu (slika 4.23.). Na ovaj način mreža uči kernele da se aktiviraju kada se uoči specifična značajka na specifičnoj poziciji ulaza. Svaki kernel ima odgovarajuću aktivacijsku mapu, koja je složena duž treće dimenzije (dubine) kako bi se formirao puni izlazni volumen iz konvolucijskog sloja. [15]

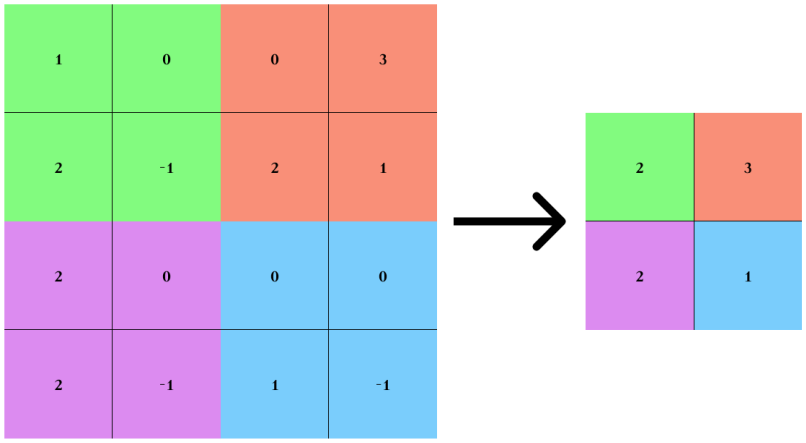


Slika 4.23. Računanje vrijednosti piksela u konvolucijskom sloju.

[2] Matrica konvolucije, maska.

4.5.3. Sloj sažimanja

Sloj sažimanja (engl. *Pooling layer*) ima cilj postupno smanjivati dimenzionalnost, broj parametara i računalnu složenost modela. Ovaj sloj djeluje nad svakom aktivacijskom mapom u ulazu i skalira njezinu dimenzionalnost pomoću MAX[3] funkcije (slika 4.24.). U većini slučajeva ovaj sloj radi s MAX funkcijom i kernelima veličine 2 x 2 koji se pomiču po horizontalnoj liniji za dva mjesta. [15]



Slika 4.24. Sažimanje po maksimalnoj vrijednosti.

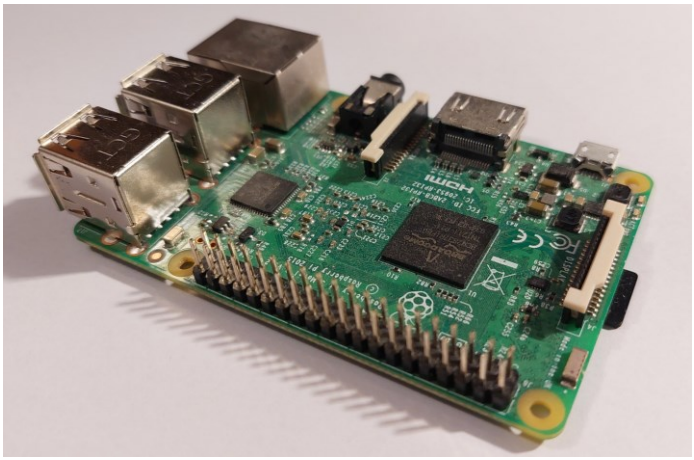
4.5.4. Potpuno povezani sloj

Potpuno povezani sloj sadrži neurone koji su direktno povezani s neuronima u sloju ispred i u sloju iza njega, a da pri tom nisu povezani ni s jednim slojem unutar njih. Ovaj sloj je analogan klasičnom sloju u umjetnim neuronskim mrežama. [15]

[3] Sažimanje po maksimalnoj vrijednosti, postoji i po srednjoj vrijednosti.

5. REALIZACIJA SUSTAVA

Sustav za glasovno upravljanje uređajima izrađen u sklopu ovog diplomskog rada realiziran je na Raspberry Pi 3 Model B (slika 5.1.) razvojnoj ploči s Raspberry Pi operacijskim sustavom, koji se temelji na Linux Debian operacijskom sustavu.



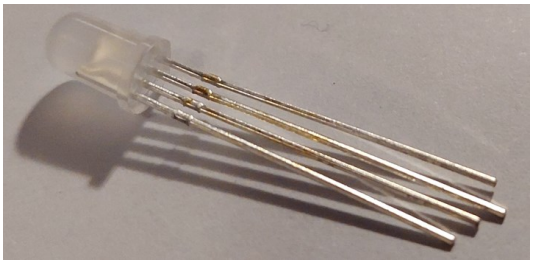
Slika 5.1. Raspberry Pi 3 Model B.

Za obradu zvuka korištena je USB zvučna kartica (slika 5.2.) preko koje je spojen mikrofon.

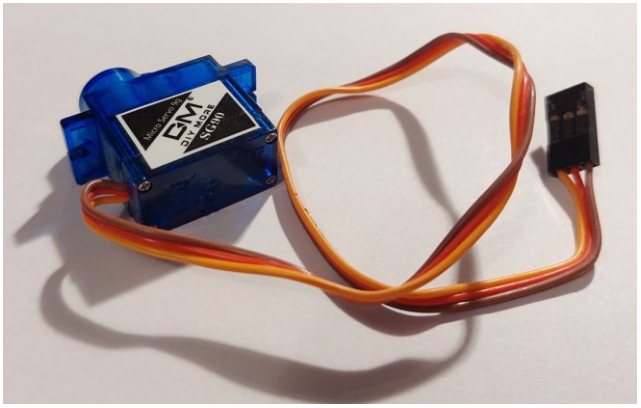
.

Slika 5.2. Axagon USB zvučna kartica.

Izvršni uređaji su RGB LED sa zajedničkom anodom (slika 5.3.) i SG90 servo motor (slika 5.4.).

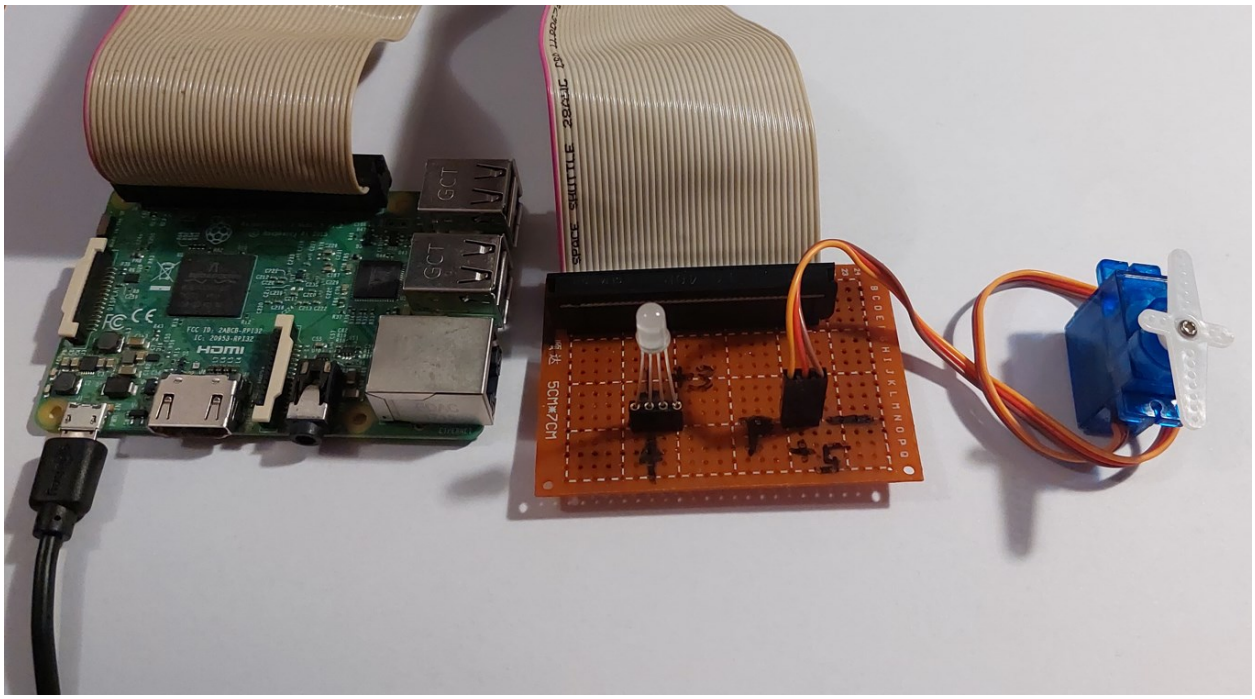


Slika 5.3. RGB LED.



Slika 5.4. SG90 servo motor.

Na slici 5.5. prikazan je Raspberry Pi koji je povezan 20-pinskim konektorom na ploču s izvršnim uređajima. Sustav omogućuje uključivanje i isključivanje LE diode, mijenjanje boje LE diode te mijenjanje kuta zakreta osovine servo motora ovisno o izgovorenim naredbama.

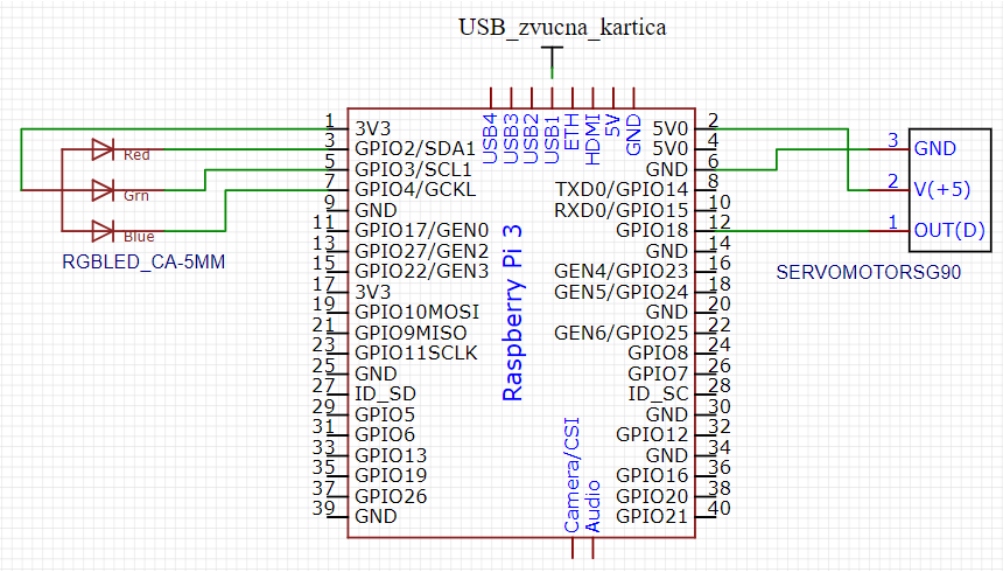


Slika 5.5. Izvršni uređaji spojeni na Raspberry Pi.

U radu su implementirane dvije verzije programskog koda za upravljanje uređajima. Prva implementacija realizirana je korištenjem *SpeechRecognition* Python modula za raspoznavanje govora. Druga implementacija temelji se na konvolucijskoj neuronskoj mreži koja klasifikacijom raspoznaje glasovnu naredbu.

5.1. Shema sustava

Na slici 5.6. se vidi shema sustava realiziranog u sklopu ovog diplomskog rada. Umjesto servo motora SG90 se može koristiti i MG90, koji je zapravo SG s metalnom osovinom. Također na slici se vidi da je motor spojen na napon 5V, a LED na napon 3.3V. Ako se koristi pločica i 20-pinski konektor, onda se jedino treba pripaziti orijentacija konektora, odnosno da je crvena oznaka kabla uvijek bliža lijevom rubu pločice (slika 5.5.).



Slika 5.6. Shema spoja sustava

5.2. Upravljanje uređajima *Speech Recognition* modulom

SpeechRecognition je Python modul za raspoznavanje govora koji radi s nekoliko različitih API-a, kao što su *Google API, Wit.AI, Microsoft Bing Voice Recognition, IBM Speech to Text*. U ovom diplomskom radu je korišten *Google Speech Recognition* iz razloga što daje mogućnost odabira jezika. S jezičnom postavkom se u zadatku usporedila detekcija govora između engleskog i hrvatskog jezika.

Prije instalacije samog modula, moraju se instalirati drugi moduli o kojima *SpeechRecognition* ovisi. U nastavku su navedene upute za instalaciju potrebnih modula.

- Prvo treba instalirati *Python 3.3* verziju ili noviju.
- *SpeechRecognition* modul.

  pip install SpeechRecognition

- *PyAudio* modul koji omogućuje korištenje mikrofona kao ulaznog uređaja.

  pip install pyaudio ili *sudo apt-get install python3-pyaudio*

- *FLAC* encoder koji kodira audio signal i prosljeđuje ga API-u.

  sudo apt-get install flac

- *RPi.GPIO* modul za inicijalizaciju i upravljanje pinovima.

  pip install RPi.GPIO

Cijeli programski kod za raspoznavanje govora na engleskom jezik se može vidjeti u Prilogu 7.1., a za raspoznavanje govora na hrvatskom jeziku u Prilogu 7.2.

5.2.1. Raspoznavanje govora na engleskom jeziku

Programski kod sa slike 5.7. poziva `time` modul, `RPi.GPIO` modul preko kojeg se određuju korišteni izlazni pinovi Raspberry Pi uređaja. Na slici je također prikazan *SpeechRecognition* modul koji omogućuje raspoznavanje govora prilikom pričanja u mikrofon. Također se vide dva rječnika (engl. *dictionary*) koja označavaju naredbe. `commands` rječnik predstavlja uređaj kojim će se upravljati, a `commandList` predstavlja naredbu koju će uređaj izvršiti. `commands` rječnik definira početne vrijednosti uređaja na ugašeno (engl. *off*). `errorWords` i `replacements` su liste koje služe za zamjenu brojeva i riječi. Npr. kod naredbe *position two* često je broj 2 prepoznat kao riječ *two* ili broj 3 kao *three*. Ovo se događa kada kontekst rečenice nije potpun te tada mreža ne može odlučiti radi li se o znamenki ili riječi. Pošto se u ovom programskom kodu koriste isključivo znamenke, za svaku riječ koja predstavlja znamenku je dodijeljena zamjena. Time će aplikacija sigurno prepoznati izgovorenu naredbu.

```
from time import sleep
import RPi.GPIO as GPIO
import speech_recognition as sr

commands = {"led": "off",
"motor": "off"}
commandList = ["on", "off",
"red", "green", "blue", "yellow", "purple",
"position 1", "position 2", "position 3", "position 4", "position 5"]
# Used because script recognizes some numbers as words
errorWords = ["one", "two", "three", "for", "five"]
replacements = ["1", "2", "3", "4", "5"]
```

Slika 5.7. Osnovne naredbe aplikacije.

Nakon toga se definiraju pinovi za crvenu, zelenu i plavu boju RGB LE diode, ali i za upravljački PWM signal servo motora (slika 5.8.). `GPIO.setmode` metoda inicijalizira pinove prema fizičkim pinovima s ploče. Stoga se isti postavljaju kao izlazni pinovi s početnim vrijednostima.

```
RED_PIN = 3
GREEN_PIN = 5
BLUE_PIN = 7

PWM_MOTOR = 12

GPIO.setmode(GPIO.BOARD)

# LED pins
GPIO.setup(RED_PIN, GPIO.OUT)  # RED
GPIO.setup(GREEN_PIN, GPIO.OUT)  # GREEN
GPIO.setup(BLUE_PIN, GPIO.OUT)  # BLUE

GPIO.output(RED_PIN, 1)
GPIO.output(GREEN_PIN, 1)
GPIO.output(BLUE_PIN, 1)

# PWM motor pin
GPIO.setup(PWM_MOTOR, GPIO.OUT)

pwm = GPIO.PWM(PWM_MOTOR, 50)
pwm.start(0)
```

Slika 5.8. Inicijalizacija pinova.

Na slici 5.9. su prikazane metode za promjenu kuta servo motora, promjenu LE diode te `callback` metoda koja preko Google API-a vraća izgovorenu rečenicu. Prvo zamjenjuje svaki broj koji je predstavljen kao riječ sa znakom iz `replacement` liste. Nakon toga prolazi kroz rečenicu i provjerava spominje li se u rečenici jedan od uređaja (LED, motor) te koja naredba je navedena.

```
def setAngle(angle):
    duty = angle / 18 + 2
    GPIO.output(PWM_MOTOR, True)
    pwm.ChangeDutyCycle(duty)
    sleep(1)
    GPIO.output(PWM_MOTOR, False)
    pwm.ChangeDutyCycle(0)

def setColor(red, green, blue):
    GPIO.output(RED_PIN, 1 - red)
    GPIO.output(GREEN_PIN, 1 - green)
    GPIO.output(BLUE_PIN, 1 - blue)

def callback(recognizer, audio):
    try:
        speech = recognizer.recognize_google(audio).lower()
        for word, replacement in zip(errorWords, replacements):
            speech = speech.replace(word, replacement)

        for device in commands.keys():
            if device in speech:
                for command in commandList:
                    if command in speech:
                        commands[device] = command

    except sr.UnknownValueError:
        print("UnknownValueError")
    except sr.WaitTimeoutError:
        print("WaitTimeoutError")
```

Slika 5.9. Metode korištene u aplikaciji.

Nakon odrađene inicijalizacije, slika 5.10. prikazuje početak programskog koda, prvo stvara objekt za raspoznavanje i objekt koji predstavlja mikrofon. Zatim pokreće pozadinsko slušanje s `callback` metodom, koje pri svakoj detekciji zvuka sluša 3 sekunde prije svog ponovnog pozivanja. U međuvremenu, glavni *thread* ispisuje i izvodi naredbe detektirane preko mikrofona. CTRL + C vraća motor u početno stanje te gasi LED prije izlaska iz funkcije.

```
if __name__ == "__main__":
    try:
        recognizer = sr.Recognizer()
        microphone = sr.Microphone()
        with microphone as source:
            recognizer.adjust_for_ambient_noise(source)
        recognizer.listen_in_background(microphone, callback, phrase_time_limit=3)
        ledState = commands["led"]
        motorState = commands["motor"]

        while True:
            print(commands)
            if commands["led"] != ledState:
                ledState = commands["led"]
                if ledState == "on":
                    setColor(1, 1, 1)
                elif ledState == "off":
                    setColor(0, 0, 0)
                elif ledState == "red":
                    setColor(1, 0, 0)
                elif ledState == "green":
                    setColor(0, 1, 0)
                elif ledState == "blue":
                    setColor(0, 0, 1)
                elif ledState == "yellow":
                    setColor(1, 1, 0)
                elif ledState == "purple":
                    setColor(1, 0, 1)

            if commands["motor"] != motorState:
                motorState = commands["motor"]
                if motorState == "on":
                    setAngle(180)
                elif motorState == "off":
                    setAngle(0)
                elif motorState == "position 1":
                    setAngle(30)
                elif motorState == "position 2":
                    setAngle(60)
                elif motorState == "position 3":
                    setAngle(90)
                elif motorState == "position 4":
                    setAngle(120)
                elif motorState == "position 5":
                    setAngle(150)

    except KeyboardInterrupt:
        setColor(0, 0, 0)
        setAngle(0)
```

Slika 5.10. Glavni *thread*, izvođenje naredbi.

5.2.2. Raspoznavanje govora na hrvatskom jeziku

Programski kod za raspoznavanje govora na hrvatskom jeziku gotovo je jednak kao i za engleski jezik. Jedina razlika su hrvatske naredbe, kao što je prikazano na slici 5.11. Također kod hrvatskog jezika nije bilo potrebe za promjenom riječi u znamenke kao u programskom kodu za engleski jezik. Korišteni API točno raspoznaje brojeve, stoga nije bilo potrebe za nikakvom dodatnom zamjenom kao u slučaju programskog koda prikazanog na slici 5.7.

```
from time import sleep
import RPi.GPIO as GPIO
import speech_recognition as sr

commands = {"led": "ugasi",
            "motor": "ugasi"}

commandList = ["upali", "ugasi",
               "crvena", "zelena", "plava", "žuta", "ljubičasta",
               "pozicija 1", "pozicija 2", "pozicija 3", "pozicija 4", "pozicija 5"]
```

Slika 5.11. Naredbe na hrvatskom jeziku.

Kod `callback` metode se prilikom korištenja API-a mora naglasiti korištenje hrvatskog jezika. Sa slike 5.12. se vidi da se predajom parametra *language* jednostavno mijenja govorni jezik.

```
def callback(recognizer, audio):
    try:
        speech = recognizer.recognize_google(audio, language='hr-HR').lower()

        for device in commands.keys():
            if device in speech:
                for command in commandList:
                    if command in speech:
                        commands[device] = command

    except sr.UnknownValueError:
        print("UnknownValueError")
    except sr.WaitTimeoutError:
        print("WaitTimeoutError")
```

Slika 5.12. Postavke parametra za raspoznavanje hrvatskog jezika.

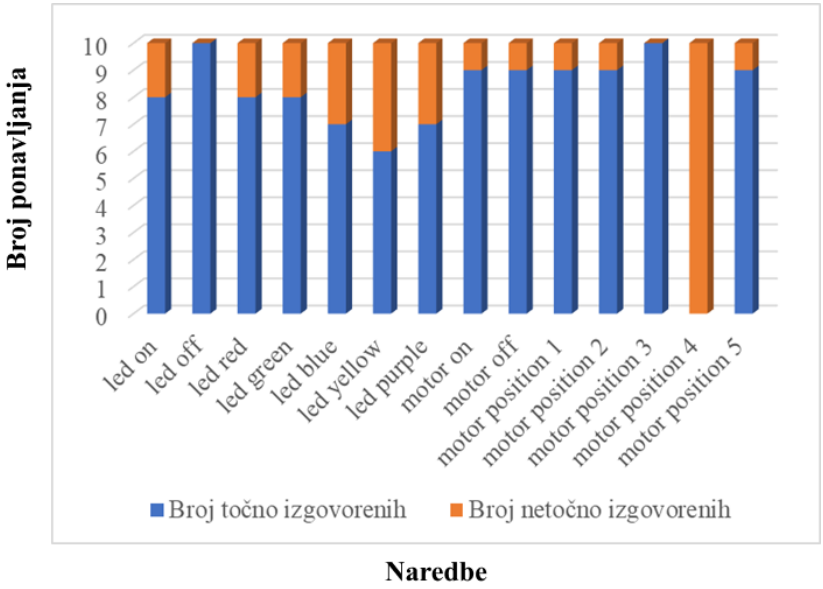
Također još jedna bitna promjena nalazi se u glavnom *threadu*, gdje su se morale izmijeniti detektirane naredbe (slika 5.13.).

```
if __name__ == "__main__":

    try:
        recognizer = sr.Recognizer()
        microphone = sr.Microphone()

        with microphone as source:
            recognizer.adjust_for_ambient_noise(source)

        recognizer.listen_in_background(microphone, callback, phrase_time_limit=3)

        ledState = commands["led"]
        motorState = commands["motor"]

        while True:

            print(commands)

            if commands["led"] != ledState:
                ledState = commands["led"]

                if ledState == "upali":
                    setColor(1, 1, 1)
                elif ledState == "ugasi":
                    setColor(0, 0, 0)
                elif ledState == "crvena":
                    setColor(1, 0, 0)
                elif ledState == "zelena":
                    setColor(0, 1, 0)
                elif ledState == "plava":
                    setColor(0, 0, 1)
                elif ledState == "žuta":
                    setColor(1, 1, 0)
                elif ledState == "ljubičasta":
                    setColor(1, 0, 1)
```

Slika 5.13. Izvođenje naredbi, hrvatski jezik.

5.2.3. Usporedba između engleskog i hrvatskog jezika

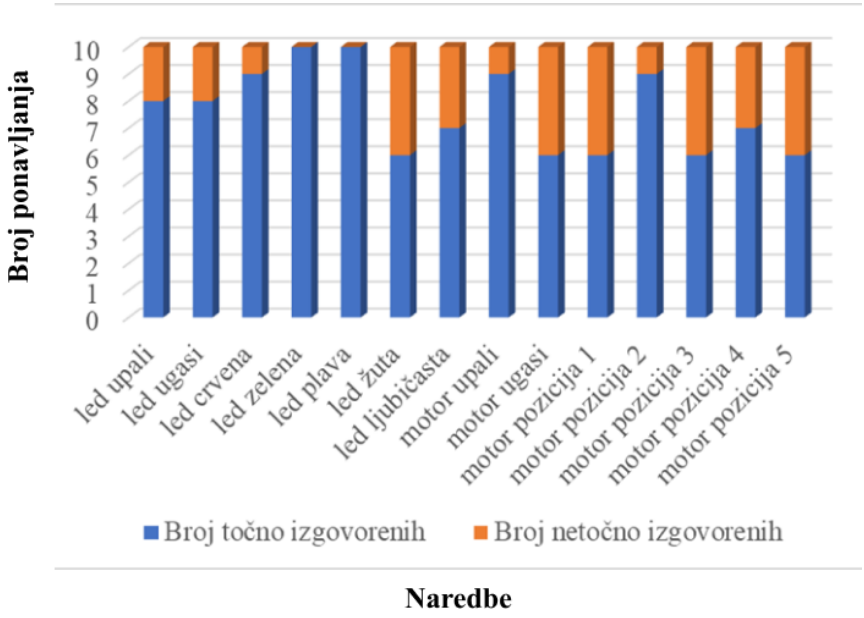
Najveći problem kod engleskog jezika je bio izgovor, stoga bi testiranje s različitim govornicima engleskog i hrvatskog materinskog jezika bilo najispravnije. Prema slici 5.14. i slici 5.15. vidi se da engleska verzija ima bolje rezultate u odnosu na hrvatsku verziju. Najveći problem prilikom prepoznavanja brojeva bio je broj četiri, jer ga je mreža zamijenila za riječ *for* 10 puta. Sve pogreške tijekom testiranja se mogu vidjeti u Tablici 5.1. i Tablici 5.2. koje su napravljane na temelju izgovaranja pojedine naredbe 10 puta.



Slika 5.14. Statistika točnosti izgovorenih naredbi ENG.

Tablica 5.1. Točnost izgovorenih naredbi za engleski jezik

Izgovorena naredba	Točno	Netočno	Prihvaćena naredba
led on	8	2	tv on, turn tv on
led off	10	0	
led red	8	2	the red, led thread
led green	8	2	lady green, enjoy the green
led blue	7	3	2 puta play lady blue, glow
led yellow	6	4	elo, england yellow, L E D E L O, play de la soul
led purple	7	3	2 puta play deep purple, really play deep purple
motor on	9	1	motor position own
motor off	9	1	motor position of
motor position 1	9	1	motor position wand
motor position 2	9	1	8 puta mijenjao je 'two' za 2, oposition to
motor position 3	10	0	10 puta mijenjao je 'three' za 3
motor position 4	0	10	10 puta mijenjao je 'for' za 4
motor position 5	9	1	motor positio file



Slika 5.15. Statistika točnosti izgovorenih naredbi HRV.

Tablica 5.2. Točnost izgovorenih naredbi za hrvatski jezik

Izgovorena naredba	**Točno**	**Netočno**	**Prihvaćena naredba**
led upali	8	2	ajde upali, led bali
led ugasi	8	2	hajde ugasi, hajde ugasi
led crvena	9	1	klet crvena
led zelena	10	0	
led plava	10	0	
led žuta	6	4	lažu te, lažu te, play žuta, lažu te
led ljubičasta	7	3	led ljubičaste, lezbe ljubičaste, led ljubičaste
motor upali	9	1	motor pali
motor ugasi	6	4	2 puta vatrogasci, modro ugasi, motor u klasi
motor pozicija 1	6	4	mother pozicija 1, motor policija, 2 puta modro pozicija 1
motor pozicija 2	9	1	motor opozicija
motor pozicija 3	6	4	moto pozicija 3, 2 puta model pozicija 3, motel pozicija 3
motor pozicija 4	7	3	model pozicija 4, modro pozicija 4, može poziv za četiri
motor pozicija 5	6	4	2 puta model pozicija 5, moto pozicija 5, motor policija

5.3. Upravljanje uređajima konvolucijskom neuronskom mrežom (CNN)

Drugi pristup rješavanju problema glasovnog upravljanja uređajima je kreiranje vlastite konvolucijske neuronske mreže za klasifikaciju riječi. Za dizajn arhitekture neuronske mreže kao i za samo treniranje korišten je *TensorFlow Keras*.

5.3.1. Python moduli neophodni za predobradu i treniranje mreže

Kreirana neuronska mreža trenirana je na Linux Ubuntu 22.04 operacijskom sustavu te su postavke prilagođene takvom okruženju. U nastavku su navedene upute za instalaciju potrebnih paketa.

- Instalirati najnoviju verziju Python 3 s njihove stranice.
- Instalirati *TensorFlow* alat za rad s neuronskim mrežama.

  pip install tensorflow
- Instalirati *librosa* paket koji se koristi za analizu zvuka.

  pip install librosa
- Instalirati *NumPy* biblioteku koja se koristi za obradu višedimenzionalnih nizova i matrica.

  pip install numpy
- Instalirati *Matplotlib* paket koji omogućuje vizualizaciju podataka.

  pip install matplotlib
- Instalirati *python_speech_features* biblioteku za obradu značajki govora.

  pip install python_speech_features

5.3.2. Set podataka korišten za treniranje

Za treniranje mreže koristio se Googleov set podataka (engl. *dataset*) koji je omogućen za slobodno korištenje 2017. godine. Set sadrži 65000+ wav datoteka koje predstavljaju 35 kratkih riječi kao što su: *Yes, No, Up, Down, Left, Right…* Uz riječi dolazi i folder s pozadinskim zvukovima i šumovima, kako bi se mreža prilagodila na smetnje prilikom obrade ulaznog podatka. Svaka datoteka predstavlja jednu sekundu izgovorene riječi, a prilikom snimanja wav datoteka sudjelovalo je više od 1000 različitih ljudi. [16]

Za primjenu u ovom diplomskom radu iz trening skupa izbačeni su pozadinski zvukovi te riječ *Up*. Pozadinski zvukovi nisu bili potrebni iz razloga što se uređaj neće primjenjivati u okolini gdje mu smeta rad bušilice ili vikanje čovjeka u pozadini. Također nisu korišteni jer njihov audio zapis traje duže (do minute) u odnosu na ostale trening datoteke (jedna sekunda) i time zahtijevaju posebnu obradu prije početka treninga. Riječ *Up* je uklonjena jer se često miješala s riječi *Stop*, koja je prvotno bila korištena u samoj aplikaciji.

5.3.3. Programski kod za obradu podataka

U ovom poglavlju su objašnjeni pojedinačni blokovi programskog koda za obradu i izdvajanje značajki iz Googleovog seta podataka. Cijeli programski kod se može vidjeti u Prilogu 7.3.

Na slici 5.16. se vide biblioteke koje su korištene za obradu te lista podataka koja se obrađuje. Također se vidi da se iz ukupnog seta podataka 10% podataka izdvaja za validacijski set i 10% za test set.

```
from os import listdir, path
from os.path import isdir, join
import librosa
import random
import numpy as np
import python_speech_features

# Dataset path
dataset_path = path.abspath('../dataset')

# Create targets list
targets = [name for name in listdir(dataset_path) if isdir(join(dataset_path, name))]

# Remove background noise and up from targets list
targets.remove('_background_noise_')
targets.remove('up')

# Settings
features_file = 'mfcc_dataset.npz'
validation_ratio = 0.1
test_ratio = 0.1
sample_rate = 8000
num_mfcc = 16
len_mfcc = 16
```

Slika 5.16. Postavke za izdvajanje značajki.

U sljedećem dijelu programskog koda definiraju se ulazni vektori. Lista `filenames` sadrži liste u kojima su nazivi svake datoteke za svaku pojedinu riječ, a `y` lista sadrži liste koje predstavljaju

svaku riječ na temelju indeksa te riječi. Npr. riječ *backward* koja ima indeks 0, će u `filenames` listi biti lista u kojoj će se nalaziti svi nazivi datoteka iz direktorija *backward*. Lista `y` sadrži listu koja predstavlja riječ *backward* te se u njoj nalazi indeks riječi *backword* za svaku datoteku iz direktorija. Što znači ako ima 2000 datoteka *backward*, u `filenames` se nalazi lista s 2000 naziva pojedine datoteke, a u `y` listi se nalazi lista s 2000 indeksa 0. Kako su na taj način pojedine datoteke povezane s indeksima riječi koje predstavljaju, prema slici 5.17. se radi miješanje lista. To omogućava da mreža prilikom treninga dobiva nasumične podatke na ulazu.

```
# Create filenames and ground truth vector
filenames = []
y = []

for index, target in enumerate(targets):
    print(join(dataset_path, target))
    filenames.append(listdir(join(dataset_path, target)))
    y.append(np.ones(len(filenames[index])) * index)

# Flatten filename and y vectors
filenames = [item for sublist in filenames for item in sublist]
y = [item for sublist in y for item in sublist]

# Associate filenames and y then shuffle
filenames_y = list(zip(filenames, y))
random.shuffle(filenames_y)
filenames, y = zip(*filenames_y)
```

Slika 5.17. Povezivanje i miješanje trening podataka.

Na sljedećoj slici 5.18. prikazan je programski kod kojim se definira veličina validacijskog seta i trening seta podataka. Također, prikazan je i programski kod za izdvajanje validacijskog seta, trening seta i test seta za `filenames` i `y` liste.

```
# Calculate validation and test set sizes
validation_size = int(len(filenames) * validation_ratio)
test_size = int(len(filenames) * test_ratio)

# Extract validation, test and train sets
filenames_val = filenames[:validation_size]
filenames_test = filenames[validation_size:(validation_size + test_size)]
filenames_train = filenames[(validation_size + test_size):]

# Extract validation, test and train sets
y_orig_val = y[:validation_size]
y_orig_test = y[validation_size:(validation_size + test_size)]
y_orig_train = y[(validation_size + test_size):]
```

Slika 5.18. Izdvajanje validacijskog, testnog i trening seta podataka.

Slika 5.19. prikazuje dvije metode, jedna služi za proračun MFCC-a, a druga za izvlačene značajki. U `calculate_mfcc(path)` se koristi *librosa* paket koji za pojedinu datoteku izdvaja signal preko kojeg se dobiva MFCC tog signala. A `extract_features(filenames, y)` vraća liste s MFCC podacima i njihove indekse.

```
# Create MFCC from given path
def calculate_mfcc(path):
    # Load wavefile
    signal, fs = librosa.load(path, sr=sample_rate)

    # Create MFCC from sound clip
    mfcc = python_speech_features.base.mfcc(signal,
                                    samplerate=fs,
                                    winlen=0.256,
                                    winstep=0.050,
                                    numcep=num_mfcc,
                                    nfilt=26,
                                    nfft=2048,
                                    preemph=0.0,
                                    ceplifter=0,
                                    appendEnergy=False,
                                    winfunc=np.hanning)
    return mfcc.transpose()

# Extract features from file and create output lists
def extract_features(filenames, y):
    out_x = []
    out_y = []

    for index, filename in enumerate(filenames):

        # Create path from given filename and target item
        path = join(dataset_path, targets[int(y[index])], filename)

        # Check if this is wav file
        if not path.endswith('.wav'):
            continue

        # Create MFCC
        mfcc = calculate_mfcc(path)

        # Only keep MFCC with right lenght
        if mfcc.shape[1] == len_mfcc:
            out_x.append(mfcc)
            out_y.append(y[index])

    return out_x, out_y
```

Slika 5.19. Metode za izdvajanje značajki.

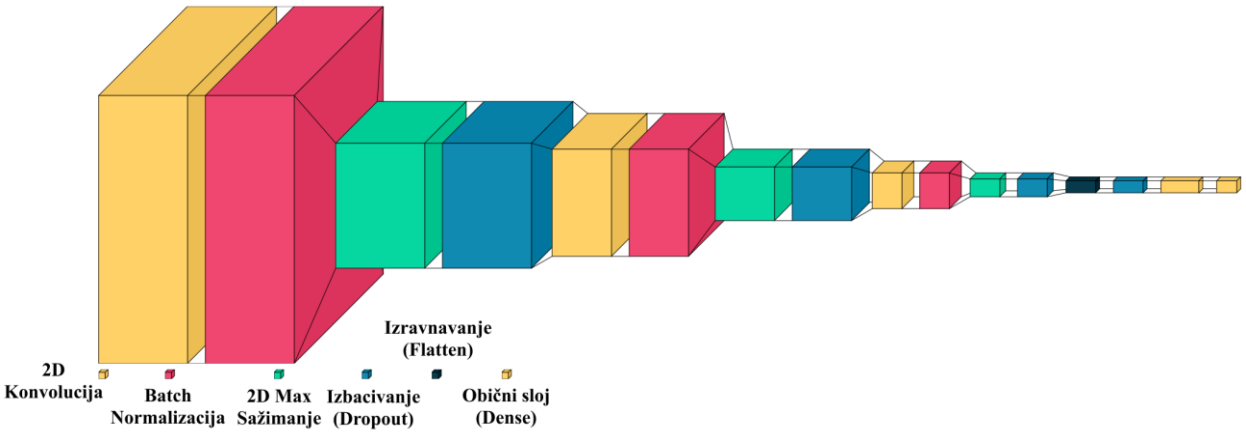
U nastavku programskog koda izdvajaju se značajke koje se spremaju u datoteku pod nazivom *mfcc_dataset.npz*, kao što se može vidjeti na slici 5.20.

```
# Create train, validation, and test sets
x_train, y_train = extract_features(filenames_train, y_orig_train)
x_val, y_val = extract_features(filenames_val, y_orig_val)
x_test, y_test = extract_features(filenames_test, y_orig_test)

# Save features and y
np.savez(features_file,
        x_train=x_train,
        y_train=y_train,
        x_val=x_val,
        y_val=y_val,
        x_test=x_test,
        y_test=y_test)

print(„Feature extraction done …“)
```

Slika 5.20. Stvaranje i spremanje značajki ulaznog seta podataka.

5.3.4. Dizajn i arhitektura mreže

Dizajn mreže se zasniva na konvolucijskoj neuronskoj mreži koja je prikazana na slici 5.21. Mreža je sastavljena od tri konvolucijska sloja, koji su popraćeni slojem Batch normalizacije i slojem sažimanja. Na kraju mreže poslije zadnjeg sažimanja ostaje izbacivanje nakon kojeg slijedi izravnavanje podataka u jedan vektor. Sloj izravnavanja je spojen na klasični skriveni sloj nakon kojeg se dolazi do *softmax* sloja koji određuje jednu od 34 klase.



Slika 5.21. Arhitektura konvolucijske neuronske mreže.

Ulazni oblik u mrežu je jedan MFCC dimenzija 16 x 16. Na slici 5.22. se vidi evaluacija modela te da je ukupan broj parametara 87574 od kojih je 86974 za treniranje.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 15, 15, 150)       750

batch_normalization (BatchN  (None, 15, 15, 150)       600
ormalization)

max_pooling2d (MaxPooling2D  (None, 7, 7, 150)         0
)

dropout (Dropout)            (None, 7, 7, 150)         0

conv2d_1 (Conv2D)            (None, 6, 6, 100)         60100

batch_normalization_1 (Batc  (None, 6, 6, 100)         400
hNormalization)

max_pooling2d_1 (MaxPooling  (None, 3, 3, 100)         0
2D)

dropout_1 (Dropout)          (None, 3, 3, 100)         0

conv2d_2 (Conv2D)            (None, 2, 2, 50)          20050

batch_normalization_2 (Batc  (None, 2, 2, 50)          200
hNormalization)

max_pooling2d_2 (MaxPooling  (None, 1, 1, 50)          0
2D)

dropout_2 (Dropout)          (None, 1, 1, 50)          0

flatten (Flatten)            (None, 50)                0

dropout_3 (Dropout)          (None, 50)                0

dense (Dense)                (None, 64)                3264

dense_1 (Dense)              (None, 34)                2210

=================================================================
Total params: 87,574
Trainable params: 86,974
Non-trainable params: 600
_________________________________________________________________
EVALUATE:
292/292 [==============================] loss: 0.6045 - accuracy: 0.8506
```

Slika 5.22. Evaluacija i sažetak modela.

5.3.5. Programski kod za treniranje mreže

Kod treniranja mreže korišten je *Tensorflow* paket, odnosno *Keras API*, koji predstavlja apstrakciju *TensorFlow-a*. Aplikacija na početku poziva datoteku s već obrađenim značajkama izvedenim iz Googleovog seta podataka. Iz slike 5.23. vidljivo je da se poslije definiranja putanja poziva metoda za učitavanje trening podataka, metoda za definiranje ulaznog oblika, metoda za

izgradnju modela, metoda za trening modela i metoda za stvaranje grafova preciznosti i pogreške te se na kraju ispisuje evaluacija i model se sprema pod nazivom koji je definiran varijablom `model_file`.

```
if __name__ == "__main__":

    # Features and model path
    features_path = path.abspath('./mfcc_dataset.npz')
    model_file = 'model.h5'

    # Extract data
    x_train, y_train, x_val, y_val, x_test, y_test = prepare_data(features_path)

    # Input shape = MFCC of 1 sample (16, 16, 1) = (16, 16)
    input_shape = x_train.shape[1:]

    # Build model
    model = build_model(input_shape)

    # Train model
    history = train(model, x_train, y_train, x_val, y_val)

    # Plot accuracy and loss
    save_plots(history)

    # Evaluate model
    print("EVALUATE:")
    model.evaluate(x=x_test, y=y_test)

    # Save the model as a file
    models.save_model(model, model_file)
```

Slika 5.23. Redoslijed pozivanja metoda.

Pozivanjem metode `prepare_data(features_path)` učitavaju se podaci s predane putanje, u ovom slučaju podaci iz datoteke *mfcc_dataset,npz*. Na slici 5.24. se vidi da se nakon učitavanja setova podataka, *x* podaci transformiraju u oblik od četiri dimenzije, kod kojeg prva dimenzija predstavlja ukupan broj datoteka u setu, druga i treća dimenzija širinu i visinu jednog podatka te četvrta dimenzija predstavlja broj audio kanala. Pošto datoteke imaju samo jedan audio kanal (mono), kao zadnju dimenziju predaje se broj 1, kojeg se može zanemariti. U ovom slučaju jedan MFCC je dimenzija 16 x 16 x 1 što je zapravo isto kao 16 x 16. Na slici se također vide hiperparametri mreže koji se koriste tijekom treninga.

```
import numpy as np
from os import path
import matplotlib.pyplot as plt
from tensorflow import optimizers
from tensorflow.keras import layers, models, regularizers, callbacks

# Parameters
EPOCHS = 200
BATCH_SIZE = 300
PATIENCE = 5
LEARNING_RATE = 0.001
CLASS_NUMBER = 34

def prepare_data(features_path):

    print(„Preparing data…“)

    # Load features
    features = np.load(features_path)

    # Extract features
    x_train = features['x_train']
    y_train = features['y_train']
    x_val = features['x_val']
    y_val = features['y_val']
    x_test = features['x_test']
    y_test = features['y_test']

    # Shape = (batch, height, width, channels)
    x_train = x_train.reshape(x_train.shape[0],
                              x_train.shape[1],
                              x_train.shape[2],
                              1)
    x_val = x_val.reshape(x_val.shape[0],
                          x_val.shape[1],
                          x_val.shape[2],
                          1)
    x_test = x_test.reshape(x_test.shape[0],
                            x_test.shape[1],
                            x_test.shape[2],
                            1)

    return x_train, y_train, x_val, y_val, x_test, y_test
```

Slika 5.24. Priprema značajki za trening.

Nakon pripreme se izgrađuje model, što je prikazano programskim kodom na slici 5.25. Također se vidi da je mreža sagrađena od tri sloja konvolucije (2D konvolucija, Batch normalizacija i 2D sažimanje po maksimalnoj vrijednosti), jednog klasičnog sloja te *softmax* izlaznog sloja. U

slojevima se koristila L2 regularizacija s vrijednosti 0.001 te izbacivanje s vrijednosti 0.1. Prvom sloju se predaje ulazni oblik koji je definiran na slici 5.23., a tijekom treninga se koristi *Adam* optimizator sa stopom učenja 0.001. Za izračun gubitaka koristi se `sparse_categorical_crossentropy` metoda, a metrika je definirana prema točnosti.

```
def build_model(input_shape, class_number=CLASS_NUMBER,
learning_rate=LEARNING_RATE):
    print("Building model...")
    model = models.Sequential()

    # First Layer
    model.add(layers.Conv2D(150, (2, 2), activation='relu', input_shape=input_shape,
                        kernel_regularizer=regularizers.l2(0.001)))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D(pool_size=(2, 2)))

    model.add(layers.Dropout(0.1))

    # Second Layer
    model.add(layers.Conv2D(100, (2, 2), activation='relu',
kernel_regularizer=regularizers.l2(0.001)))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D(pool_size=(2, 2)))

    model.add(layers.Dropout(0.1))

    # Third Layer
    model.add(layers.Conv2D(50, (2, 2), activation='relu',
kernel_regularizer=regularizers.l2(0.001)))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D(pool_size=(2, 2)))

    model.add(layers.Dropout(0.1))

    # Stretch data in one dimension
    model.add(layers.Flatten())
    model.add(layers.Dropout(0.1))

    # Fourth Layer
    model.add(layers.Dense(64, activation='relu'))

    # Output Layer
    model.add(layers.Dense(class_number, activation='softmax'))
    adam = optimizers.Adam(learning_rate=learning_rate)
    model.compile(loss='sparse_categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
    model.summary()
    return model
```

Slika 5.25. Metoda za izgradnju modela.

Nakon izgradnje modela potrebno je trenirati neuronsku mrežu, a to se postiže izvođenjem programskog koda prikazanog na slici 5.26. U metodi za treniranje je definirana varijabla `earlystop` koja se koristi za zaustavljanje treninga u slučaju da se točnost detekcije počne smanjivati.

```
def train(model, x_train, y_train, x_val, y_val, epochs=EPOCHS, batch_size=BATCH_SIZE,
patience=PATIENCE):
    print("Training model...")
    earlystop = callbacks.EarlyStopping(monitor="accuracy", min_delta=0.001,
patience=patience)
    # Start network training
    history = model.fit(x_train,
                        y_train,
                        epochs=epochs,
                        batch_size=batch_size,
                        validation_data=(x_val, y_val),
                        callbacks=[earlystop])
    return history
```

Slika 5.26. Trening metoda.

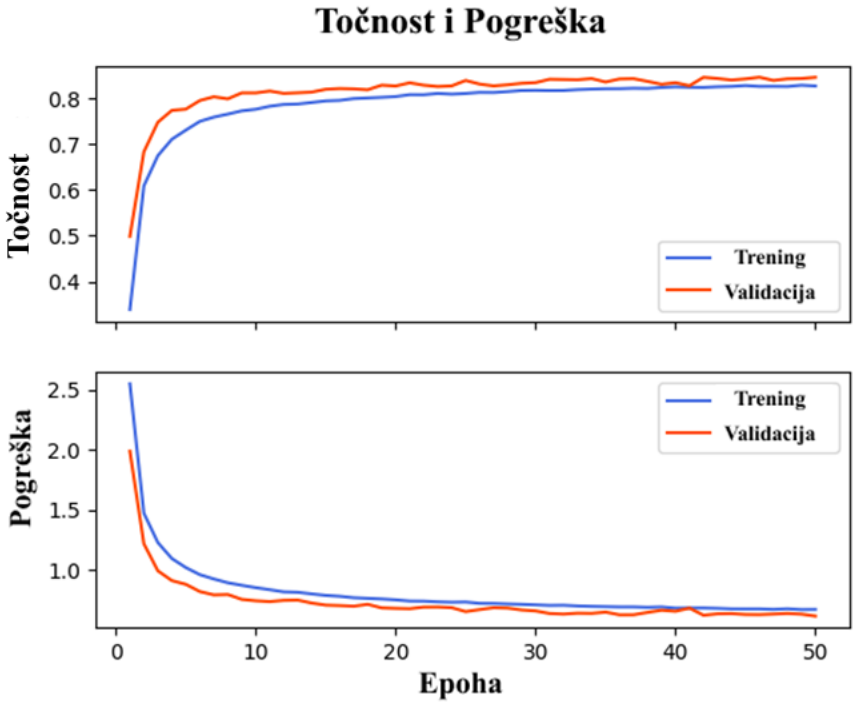
Trening metoda vraća parametre za crtanje i prikaz grafova na temelju točnosti i gubitaka modela tijekom treninga. Na slici 5.27. se može vidjeti `save_plots` metoda koja izdvaja podatke nakon treniranja te crta karakteristike modela koji se spremaju kao datoteka *acc_loss.png*.

```
def save_plots(history, epochs=EPOCHS):
    # Plot training results
    accuracy = history.history['accuracy']
    val_accuracy = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(1, len(accuracy) + 1)

    fig, (accuracyPlt, lossPlt) = plt.subplots(2, sharex=True)
    fig.suptitle("Accuracy and Loss")
    accuracyPlt.plot(epochs, accuracy, color='royalblue', label='Training')
    accuracyPlt.plot(epochs, val_accuracy, color='orangered', label='Validation')
    accuracyPlt.set_ylabel('Accuracy')
    accuracyPlt.set_xlabel('Epochs')
    accuracyPlt.legend()

    lossPlt.plot(epochs, loss, color='royalblue', label='Training')
    lossPlt.plot(epochs, val_loss, color='orangered', label='Validation')
    lossPlt.set_ylabel('Loss')
    lossPlt.set_xlabel('Epochs')
    lossPlt.legend()
    fig.savefig('acc_loss.png')
```

Slika 5.27. Metoda za crtanje grafova.

5.3.6. Rezultat treninga

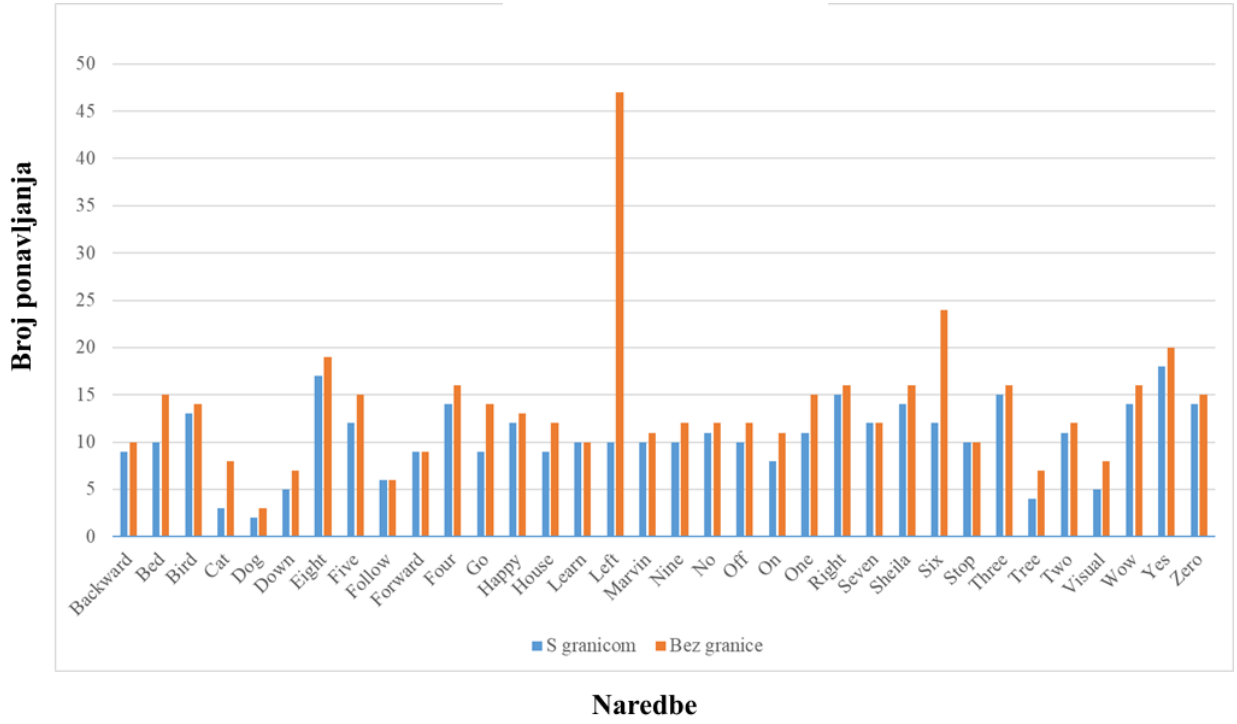
Na slici 5.28. se vide rezultati treniranja modela na temelju točnosti i pogreške tijekom svake epohe. Vidi se da su vrijednosti točnosti i za validaciju i za trening veće od 80%, točnije 82.72% na trening skupu i 84.62% na validacijskom skupu. Također, vidljivo je da su vrijednosti pogreške su manje od 1.0, tj. 0.67 za trening skup i 0.61 za validacijski skup. Evaluacija pogreške iznosi 0.60, a evaluacija točnosti iznosi 85.06%, što se može vidjeti iz slike 5.22. Kompletan programski kod za treniranje mreže se može vidjeti u Prilogu 7.4.



Slika 5.28. Točnost i pogreška tijekom treniranja modela.

5.3.7. Rezultati mjerenja točnosti izgovorenih riječi

U ovom poglavlju napravljena je analiza točnosti klasifikacije na temelju izgovorenih naredbi. Test se izvodio na način da se svaka riječ izgovori 10 puta i zapisuju se postotci točne detekcije. Rezultati mjerenja se uzimaju s definiranom granicom od 50% vjerojatnosti i bez granice, slučaj gdje vjerojatnost može biti manja od 1%. U slučajevima s tako niskim postotkom, dobivene vrijednosti se mogu zanemariti jer mreža uvijek mora dati nekakav izlaz, čak i kada mikrofon ne detektira nikakve vrijednosti. Zbog tog problema na slici 5.29. se vidi da je mreža na izlazu dala rezultat *Left* više od 45 puta, ali pri tim detekcijama vjerojatnosti su bile jako niske. Sve riječi koje imaju detekciju s granicom oko broja 10 su ispravno detektirane i mogu se smatrati riječima koje će mreža u većini slučajeva ispravno klasificirati.



Slika 5.29. Statistika točnosti izgovorenih riječi.

5.3.8. Integracija modela i Raspberry Pi uređaja

Kako bi se model povezao s Raspberry Pi uređajem, potrebno je instalirati dodatne Python module za Raspberry Pi. U nastavku su navedeni potrebni moduli te upute kako ih instalirati.

- *NumPy* → biblioteka koja se koristi za obradu višedimenzionalnih nizova i matrica.

  pip install numpy

- *python_speech_features* → biblioteka za obradu značajki govora.

  pip install python_speech_features

- *RPi.GPIO* modul za inicijalizaciju i upravljanje pinovima.

  pip install RPi.GPIO

- *sounddevice* → modul koji ostvaruje vezu s *PortAudio* bibliotekom te omogućuje korištenje funkcija za snimanje i puštanje *NumPy* nizova koji predstavljaju zvuk.

  python3 -m pip install sounddevice

- *SciPy* → biblioteka za znanstveno računanje, posebne funkcije za obradu slike i zvuka.

  sudo apt-get install python3-scipy

- *TensorFlow Lite* → modul za primjenu neuronskih mreža kod ugradbenih uređaja.

  pip install tflite-runtime

Za korištenje modela u slabijem računalnom okruženu kao što je Raspberry Pi, istrenirani model treba prilagoditi. Ovo je napravljeno korištenjem programskog koda prikazanog u Prilogu 7.5. Programski kod radi na način da se u varijable `keras_model_filename` upiše ime modela dobivenog treningom, a u `tflite_filename` se upisuje naziv modela koji će nastati kao lite verzija. Nakon unosa naziva, programski kod je potrebno pokrenuti iz konzole.

Lite verziju modela treba prebaciti na uređaj u isti folder u kojem se nalazi programski kod koji koristi model i upravlja uređajima.

5.3.9. Glavna aplikacija za upravljanje uređajima

Nakon svih prethodnih instalacija i prilagodbe okruženja rada, na Raspberry Pi uređaju se može pokrenuti aplikacija koja koristi model i upravlja izvršnim uređajima pomoću govora. Pošto u setu podataka za učenje ne postoji riječ LED i motor, LED riječ je predstavljena s riječi *sheila*, a motor riječ je predstavljena s riječi *house*. Ostale naredbe za oba uređaja su: *on, off, one, two, three*.

Prvi dio programskog koda se može vidjeti na slici 5.30., gdje se uz prethodno instalirane pakete definiraju sve riječi koje model može prepoznati.

```
import sounddevice as sd
import numpy as np
import scipy.signal
from time import sleep
import python_speech_features
import RPi.GPIO as GPIO
from tflite_runtime.interpreter import Interpreter

words = [
   'visual', 'five', 'three', 'down', 'learn', 'go',
   'no', 'nine', 'dog', 'left', 'two', 'on', 'backward',
   'right', 'happy', 'sheila', 'follow', 'zero',
   'tree', 'marvin', 'stop', 'wow', 'eight', 'seven', 'bird',
   'cat', 'yes', 'house', 'six', 'bed', 'off', 'forward', 'one', 'four' ]
```

Slika 5.30. Riječi za klasifikaciju.

Nakon toga se programskim kodom prikazanim na slici 5.31. definira spremnik riječi koji se koriste kasnije prilikom detekcije uređaja i naredbi. Također su na slici vidljive postavke korištene

za obradu signala te programski kod za definiranje pomičnog prozora koji izračunava MFCC vrijednosti u pojedinim trenucima audio zapisa.

```
word_buffer = []
prediction_counter = 0

# Settings
word_threshold = 0.5
rec_duration = 0.5
sample_rate = 48000
resample_rate = 8000
channels_num = 1
num_mfcc = 16
model_path = 'lite_model.tflite'

# Sliding window
window = np.zeros(int(rec_duration * resample_rate) * 2)
```

Slika 5.31. Postavke za obradu audio signala.

Sljedeći dio programskog koda odrađuje inicijalizaciju na isti način kao u poglavlju 5.1. Inicijalizacija pinova odrađena je kao na slici 5.8., a metode `setAngle` i `setColor` su definirane kao na slici 5.9. Poslije inicijalizacije pinova, inicijalizira se model (slika 5.32.) preko kojeg se klasificiraju riječi izgovorene u mikrofon.

```
# Load model (interpreter)
interpreter = Interpreter(model_path)
interpreter.allocate_tensors()
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
print(input_details)
```

Slika 5.32. Inicijalizacija modela.

Programski kod počinje u trenutku kada mikrofon počne osluškivati svoje okruženje i prikupljati signale za obradu. Na slici 5.33. vidi se `InputStream` metoda koja u sebi poziva `callback` metodu svakih pola sekunde. `callback` metoda prvo provjerava prisutnost *errora*, zatim se iz `modify_mic_sample` dobiva prozor i broj uzoraka prilagođen datotekama iz skupa koji se koristio kod treniranja mreže (slika 5.34.). Nakon toga se izračunavaju MFCC vrijednosti na temelju kojih se određuje predikcija riječi. Ako klasificirana riječ prelazi prag od 50% točnosti onda se sprema u spremnik riječi. Programski kod provjerava nalaze li se u spremniku dvije riječi, ukoliko se nalaze spremnik se predaje metodi za upravljanje uređajima. Metoda na temelju riječi u spremniku upravlja uređajima te po završetku ispražnjava spremnik. Ukoliko se u roku 5 sekundi spremnik ne napuni s dvije riječi, pokreće se automatsko pražnjenje spremnika.

```
if '__main__' == __name__:
    try:
        # Start listening to the microphone
        with sd.InputStream(channels=channels_num,
                            samplerate=sample_rate,
                            blocksize=int(sample_rate * rec_duration),
                            callback=callback):
            while True:
                pass

    except KeyboardInterrupt:
        SetAngle(0)
        setColor(0, 0, 0)
```

Slika 5.33. Početak aplikacije.

```
# Executes every 0.5 seconds
def callback(recording, frames, time,  status):

    # Notify if errors
    if status:
        print('Error:', status)

    window, new_sr = modify_mic_sample(recording)
    mfccs = get_mfccs(window, new_sr)
    index, word, word_probability = make_prediction(mfccs)
    print(f"index={index:<10}word={words[index]:<10}probability={word_probability}")

    global prediction_counter
    prediction_counter += 1
    print(prediction_counter)

    if word_probability > word_threshold:
        word_buffer.append(word)
        if len(word_buffer) == 2:
            control_devices(word_buffer, index, word_probability)

            # Reset Buffer
            prediction_counter = 0
            print(word_buffer)
            word_buffer.clear()

    # Reset Buffer
    if prediction_counter == 10:
        word_buffer.clear()
        prediction_counter = 0
```

Slika 5.34. `callback` metoda.

Na slici 5.35. se vidi `modify_mic_sample` metoda koja se prva poziva unutar `callback` metode. Radi na način da kao parametar prima signal s mikrofona te pomoću metode `decimate` mijenja broj uzoraka na 8000 i sprema taj novi signal pomoću pomičnog prozora.

```
# Decimate = filter and downsample
def decimate(signal, old_sr, new_sr):

    # Check for downsampling
    if new_sr > old_sr:
        print("Error: target sample rate higher than original")
        return signal, old_sr

    # Downsample by integer factor
    dec_factor = old_sr / new_sr
    if not dec_factor.is_integer():
        print("Error: can only decimate by integer factor")
        return signal, old_sr

    # Do decimation
    resampled_signal = scipy.signal.decimate(signal, int(dec_factor))

    return resampled_signal, new_sr

def modify_mic_sample(recording):

    # Remove second dimension from sample
    recording = np.squeeze(recording)

    # Resample
    recording, new_sr = decimate(recording, sample_rate, resample_rate)

    # Save recording with sliding window
    window[:len(window)//2] = window[len(window)//2:]
    window[len(window)//2:] = recording

    return window, new_sr
```

Slika 5.35. Promjena broja uzoraka ulaznog signala.

Nakon promjene broja uzoraka poziva se metoda za izračun značajki (MFCC) koja radi na isti način kao i kod proračuna značajki podataka za trening, validaciju i test. Metodi `make_prediction` se predaje MFCC izračunat u koraku prije (slika 5.36.). Zatim transformira predani MFCC te taj podatak predaje mreži na temelju kojeg mreža vraća rezultat. U ovom slučaju vraća indeks detektirane riječi, samu riječ te postotak točnosti izgovorene riječi.

```
def get_mfccs(window, new_sr):

    # Compute features
    mfccs = python_speech_features.base.mfcc(window,
                                samplerate=new_sr,
                                winlen=0.256,
                                winstep=0.050,
                                numcep=num_mfcc,
                                nfilt=26,
                                nfft=2048,
                                preemph=0.0,
                                ceplifter=0,
                                appendEnergy=False,
                                winfunc=np.hanning)
    mfccs = mfccs.transpose()
    return mfccs

def make_prediction(mfccs):

    # Make prediction
    in_tensor = np.float32(mfccs.reshape(1, mfccs.shape[0], mfccs.shape[1], 1))
    interpreter.set_tensor(input_details[0]['index'], in_tensor)
    interpreter.invoke()
    output_data = interpreter.get_tensor(output_details[0]['index'])

    # Get index with max probability
    index = np.argmax(output_data)
    word = words[index]
    probability = output_data[0][index]

    return index, word, probability
```

Slika 5.36. Metoda za proračun značajki i predikciju ulazne vrijednosti.

Metodi `control_devices` se predaju tri parametra: spremnik riječi, indeks druge riječi u spremniku te vjerojatnost druge riječi u spremniku. Ako se riječi iz spremnika podudaraju s riječima iz uvjeta upravljanja, onda se na temelju njih izvodi odgovarajuća akcija te se u konzoli ispisuje naredba i njen postotak točnosti. Definicija metode i sama logika odabira akcije za izvođenje se može vidjeti na slici 5.37.

```
def control_devices(word_buffer, index, word_probability):

    # Execute command for specific device
    if "house" in word_buffer and "one" in word_buffer:
        setAngle(30)
print(f"\033[32mindex={index:<10}word={words[index]:<10}probability={word_probability}\
033[0m")
    elif "house" in word_buffer and "two" in word_buffer:
        setAngle(90)
print(f"\033[32mindex={index:<10}word={words[index]:<10}probability={word_probability}\
033[0m")
    elif "house" in word_buffer and "three" in word_buffer:
        setAngle(120)
print(f"\033[32mindex={index:<10}word={words[index]:<10}probability={word_probability}\
033[0m")
    elif "house" in word_buffer and "on" in word_buffer:
        setAngle(180)
print(f"\033[32mindex={index:<10}word={words[index]:<10}probability={word_probability}\
033[0m")
    elif "house" in word_buffer and "off" in word_buffer:
        setAngle(0)
print(f"\033[32mindex={index:<10}word={words[index]:<10}probability={word_probability}\
033[0m")
    elif "sheila" in word_buffer and "one" in word_buffer:
        setColor(1, 0, 0)
print(f"\033[32mindex={index:<10}word={words[index]:<10}probability={word_probability}\
033[0m")
    elif "sheila" in word_buffer and "two" in word_buffer:
        setColor(0, 1, 0)
print(f"\033[32mindex={index:<10}word={words[index]:<10}probability={word_probability}\
033[0m")
    elif "sheila" in word_buffer and "three" in word_buffer:
        setColor(0, 0, 1)
print(f"\033[32mindex={index:<10}word={words[index]:<10}probability={word_probability}\
033[0m")
    elif "sheila" in word_buffer and "on" in word_buffer:
        setColor(1, 1, 1)
print(f"\033[32mindex={index:<10}word={words[index]:<10}probability={word_probability}\
033[0m")
    elif "sheila" in word_buffer and "off" in word_buffer:
        setColor(0, 0, 0)
print(f"\033[32mindex={index:<10}word={words[index]:<10}probability={word_probability}\
033[0m")
```

Slika 5.37. *control_devices* metoda.

Kompletan programski kod za glasovno upravljanje uređajima konvolucijskom neuronskom mrežom se može vidjeti u Prilogu 7.6.

6. ZAKLJUČAK

U sklopu ovog diplomskog rada obrađen je problem upravljanja uređajima pomoću glasovnih naredbi. U ovom radu se želi postići slična funkcionalnost već postojećih, gotovih rješenja kao što su *Alexa, Google, Siri, Bixby, Cortana* itd. Prvi dio rada bavi se izradom aplikacije i sklopovlja za glasovno upravljanje pomoću već gotovih modula, paketa i servisa, koji uvelike olakšavaju predikciju riječi i obradu zvuka. Veliki problem ovog pristupa je naplata prilikom korištenja. Gotovo svi servisi koji su dostupni na tržištu i koji rješavaju ovakve probleme se naplaćuju u slučaju komercijalne upotrebe. Za izradu prve aplikacije u ovom radu korišten je Python paket *SpeechRecognition*, preko kojeg je moguća upotreba različitih alata za prepoznavanje govora kao što su *Microsoft Bing Speech, Google Web Speech API, Google Cloud Speech, IBM Speech to Text…* U ovom diplomskom raduje korišten *Google Speech API*, preko kojeg se pristupa Googleovoj neuronskoj mreži koja iz govora stvara tekst. Također *Google API* ima opciju za odabir jezika, što je poslužilo za usporedbu prepoznavanja govora između hrvatskog i engleskog jezika. Sklopovskim dijelom upravlja Raspberry Pi 3B računalo, koje je na osnovu izgovorenih naredbi odabire izvršni uređaj i zadaje mu određeni zadatak. Test je odrađen na način da se 10 puta izgovori svaka naredba te se mjeri broj točnih prepoznavanja. Prema podacima iz rada engleski jezik je bolje prepoznao naredbe, ali to je slučaj kod jednostavnih riječi. Slaganje rečenica koje imaju dugačke i teško izgovorive nazive, hrvatski jezik za hrvatske govornike definitivno ima prednost. Prilikom testiranja primijećeno je kako aplikacija s engleskim jezikom teško raspoznaje određene riječi zbog hrvatskog naglaska.

Drugi pristup rješavanju problema upravljanja uređajima putem glasovnih naredbi je dizajniranje vlastite neuronske mreže, koja obavlja klasifikaciju na temelju izgovorene riječi. Kada se riječ klasificira, ona postaje naredba kojom se odabire uređaj i zadaje mu se određeni zadatak. Korištena je konvolucijska neuronska mreža s 5 slojeva. Mreža može klasificirati 34 različite riječi, od kojih je 7 korišteno kao naredbe za upravljanje u aplikaciji. Za razliku od prvog rješenja, gdje se govor pretvarao u rečenicu i iz nje se izvlačila pojedina naredba, ovdje se predviđala po jedna riječ koja se spajala u naredbu. Testiranje točnosti je ponovno provedeno na način da se 10 puta ponavlja svaka riječ (naredba) i gledaju se izlazi iz mreže u postotcima. Pošto mreža uvijek mora dati nekakav rezultat, čak i kad mikrofon ne detektira zvuk, mreža u takvim slučajevima izbacuje predikcije u malim postocima. Zbog tog problema u aplikaciji se postavio prag za određeni postotak kod detekcije svake riječi. Iz rezultata testa se vidi da predikcija bez praga nema smisla, jer su u tim slučajevima neke riječi bile detektirane preko 40 puta. Kod testiranja aplikacije primijećeno je da se riječi ne spoje uvijek u naredbu, stoga se naredba nekad ne izvrši.

Glavna razlika između realiziranih aplikacija je složenost. Kod prvog pristupa postoje gotovi paketi koji odrađuju sve potrebno za raspoznavanje govora i vraćaju rečenicu koju je Googleova mreža prepoznala. Kod drugog pristupa se mora prvo obraditi signal koji je dobiven od strane mikrofona, nakon čega se obrađeni signal šalje u mrežu, mreža daje rezultat za koji misli da je točan i taj rezultat se prosljeđuje aplikaciji koja provjerava može li s tim podatkom upravljati uređajem ili ne.

Kao poboljšanje aplikacije moguće je koristiti drugačiju arhitekturu mreže. Također postoji mogućnost da bi drugi parametri za treniranje dali bolju izlaznu karakteristiku. Korištenje rekurentne ili LSTM mreže bi također pomoglo, pogotovo ako se rješava problem raspoznavanja govora koji automatski slaže cijele rečenice. Tijekom implementacije rješenja opisanih u sklopu ovog diplomskog rada, uočeno je da je najvažniji dio pri izgradnji ovakvih sustava dobra baza podataka koja se koristi za treniranje mreže. To je glavni razlog zašto velike, već spomenute firme imaju ogromnu prednost u odnosu na ostale.

LITERATURA

[1] Muhammad Hafidh Firmansyah, Anand Paul, Deblina Bhattacharya, Gul Malik Urfa, „A.I. based Embedded Speech to Text Using Deepspeech“, Travanj 2020, doi: 10.48550/arXiv.2002.12830

[2] Shailesh D. Arya, Dr. Samir Patel, „Implementation of Google Assistant & Amazon Alexa on Raspberry Pi“, Lipanj 2020, doi: 10.48550/arXiv.2006.08220

[3] Ana Marie. D Celebre, Alec Zandrae D. Dubouzet, Ian Benedict A. Medina, Adrian Neil M. Surposa, Reggie C. Gustilo, „Home automation using raspberry Pi through Siri enabled mobile devices“, Prosinac 2015, doi: 10.1109/HNICEM.2015.7393270

[4] Cherry Soni, Maitri Saklani, Gunjan Mokhariwale, Aishwarya Thorat, Kunal Shejul, „Multi-Language Voice Control IOT Home Automation Using Google Assistant and Raspberry Pi“, Siječanj 2015, doi: 10.1109/ACCAI53970.2022.9752606

[5] Tussanai Parthornratt, Dollachart Kitsawat, Pasd Putthapipat, Prapap Koronjaruwat, „A Smart Home Automation Via Facebook Chatbot and Raspberry Pi“, Srpanj 2018, doi: 10.1109/ICEI18.2018.8448761

[6] Intro to Audio Programming, Part1: How Audio Data is Represented https://docs.microsoft.com/hr-hr/archive/blogs/dawate/intro-to-audio-programming-part-1-how-audio-data-is-represented, (12.6.2022.)

[7] Strojno učenje u sustavima autonomnih i umreženih vozila: Uvod u neuronske mreže, Izv.prof.dr.sc. Mario Vranješ, FERIT prezentacija.

[8] Designing Your Neural Networks, Lavanya Shukla, (16.6.2022.) https://towardsdatascience.com/designing-your-neural-networks-a5e4617027ed

[9] Siddharth Sharma, Simone Sharma, „ACTIVATION FUNCTIONS IN NEURAL NETWORKS“, Svibanj 2020, doi: 10.33564/IJEAST.2020.v04i12.054

[10] Anirudha Ghosh, Abu Sufian, Farhana Sultana, Amlan Chakrabarti, Debashis De, „Fundamental Concepts of Convolutional Neural Network“, Siječanj 2020, doi: 10.1007/978-3-030-32644-9_36

[11] Visualizing Sound as an Audio Spetrogram https://developer.apple.com/documentation/accelerate/visualizing_sound_as_an_audio_spectrogram, (25.8.2022.)

[12] Computing the Mel Spectrum Using Linear Algebra https://developer.apple.com/documentation/accelerate/computing_the_mel_spectrum_using_linear_algebra, (25.8.2022.)

[13] Mel Frequency Cepstral Coefficients for Music Modeling, Beth Logan, Studeni 2000, https://www.researchgate.net/publication/2552483_Mel_Frequency_Cepstral_Coefficients_for_Music_Modeling, (25.8.2022.)

[14] Understanding LSTM Networks, 27.8.2015, http://colah.github.io/posts/2015-08-Understanding-LSTMs/, (27.8.2022.)

[15] Keiron O'Shea, Ryan Nash, „An Introduction to Convolutional Neural Networks", Prosinac 2015, doi: https://doi.org/10.48550/arXiv.1511.08458

[16] Speech Commands Dataset, Google AI Blog https://ai.googleblog.com/2017/08/launching-speech-commands-dataset.html, (6.9.2022.)

SAŽETAK

U ovom diplomskom radu opisano je korištenje neuronske mreže za glasovno upravljanje uređajima koji su spojeni preko Raspberry Pi računala. Problem glasovnog upravljanja rješava se na dva načina. U prvom načinu korišteni su gotovi Python moduli koji sadrže sve potrebno za obradu glasovnih naredbi. Za razliku od prvog, u drugom načinu je dizajnirana vlastita neuronska mreža koja je integrirana s Python aplikacijom koja upravlja uređajima preko glasovnih naredbi.

Ključne riječi: Raspberry Pi, konvolucijska neuronska mreža, Python, prepoznavanje govora, glasovno upravljanje.

ABSTRACT

Title: Voice control of devices

This thesis describes the use of a neural network for voice control of devics connected via a Raspberry Pi computer. The problem of voice control is solved in two ways. In the first method existing Python modules were used, which contain everything necessary for processing voice commands. Unlike the first, the second method has it's own neural network that is integrated with Python application for controlling devices via voice commands.

Keywords: Raspberry pi, Neural network, Convolutional neural network, Python, Speech recognition, Voice control.

ŽIVOTOPIS

Filip Milković rođen je 5. srpnja 1997. godine u Osijeku. Završio je osnovnu školu Ivana Filipovića u Osijeku nakon čega upisuje Elektrotehničku i prometnu školu Osijek, smjer tehničar za mehatroniku. Nakon završetka srednjoškolskog obrazovanja upisuje preddiplomski stručni studij Elektrotehnika smjer Automatika na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek. Na istom fakultetu poslije stečene diplome stručnog prvostupnika, upisuje razlikovne obveze smjer Računarstvo. Završetkom razlikovnih obveza iste godine upisuje diplomski sveučilišni studij Automobilsko računarstvo i komunikacije.

PRILOG 7.1. Programski kod - SpeechRecognition Engleski

```
from time import sleep
import RPi.GPIO as GPIO
import speech_recognition as sr

commands = {"led": "off",
            "motor": "off"}

commandList = ["on", "off",
               "red", "green", "blue", "yellow", "purple",
               "position 1", "position 2", "position 3", "position 4", "position 5"]

# Used because script recognizes some numbers as words
errorWords = ["one", "two", "three", "for", "five"]
replacements = ["1", "2", "3", "4", "5"]

RED_PIN = 3
GREEN_PIN = 5
BLUE_PIN = 7

PWM_MOTOR = 12

GPIO.setmode(GPIO.BOARD)

# LED pins
GPIO.setup(RED_PIN, GPIO.OUT)  # RED
GPIO.setup(GREEN_PIN, GPIO.OUT)  # GREEN
GPIO.setup(BLUE_PIN, GPIO.OUT)  # BLUE

GPIO.output(RED_PIN, 1)
GPIO.output(GREEN_PIN, 1)
GPIO.output(BLUE_PIN, 1)

# PWM motor pin
GPIO.setup(PWM_MOTOR, GPIO.OUT)

pwm = GPIO.PWM(PWM_MOTOR, 50)
pwm.start(0)


def setAngle(angle):
    duty = angle / 18 + 2
    GPIO.output(PWM_MOTOR, True)
    pwm.ChangeDutyCycle(duty)
    sleep(1)
    GPIO.output(PWM_MOTOR, False)
    pwm.ChangeDutyCycle(0)

```

```
48.
49. def setColor(red, green, blue):
50.     GPIO.output(RED_PIN, 1 - red)
51.     GPIO.output(GREEN_PIN, 1 - green)
52.     GPIO.output(BLUE_PIN, 1 - blue)
53.
54.
55. def callback(recognizer, audio):
56.     try:
57.         speech = recognizer.recognize_google(audio).lower()
58.         for word, replacement in zip(errorWords, replacements):
59.             speech = speech.replace(word, replacement)
60.
61.         for device in commands.keys():
62.             if device in speech:
63.                 for command in commandList:
64.                     if command in speech:
65.                         commands[device] = command
66.
67.     except sr.UnknownValueError:
68.         print("UnknownValueError")
69.     except sr.WaitTimeoutError:
70.         print("WaitTimeoutError")
71.
72.
73. if __name__ == "__main__":
74.
75.     try:
76.         recognizer = sr.Recognizer()
77.         microphone = sr.Microphone()
78.
79.         with microphone as source:
80.             recognizer.adjust_for_ambient_noise(source)
81.
82.         recognizer.listen_in_background(microphone, callback, phrase_time_limit=3)
83.
84.         ledState = commands["led"]
85.         motorState = commands["motor"]
86.
87.         while True:
88.
89.             print(commands)
90.
91.             if commands["led"] != ledState:
92.                 ledState = commands["led"]
93.
94.                 if ledState == "on":
95.                     setColor(1, 1, 1)
96.                 elif ledState == "off":
97.                     setColor(0, 0, 0)
```

```
98.             elif ledState == "red":
99.                 setColor(1, 0, 0)
100.            elif ledState == "green":
101.                setColor(0, 1, 0)
102.            elif ledState == "blue":
103.                setColor(0, 0, 1)
104.            elif ledState == "yellow":
105.                setColor(1, 1, 0)
106.            elif ledState == "purple":
107.                setColor(1, 0, 1)
108.
109.        if commands["motor"] != motorState:
110.            motorState = commands["motor"]
111.
112.            if motorState == "on":
113.                setAngle(180)
114.            elif motorState == "off":
115.                setAngle(0)
116.            elif motorState == "position 1":
117.                setAngle(30)
118.            elif motorState == "position 2":
119.                setAngle(60)
120.            elif motorState == "position 3":
121.                setAngle(90)
122.            elif motorState == "position 4":
123.                setAngle(120)
124.            elif motorState == "position 5":
125.                setAngle(150)
126.
127.    except KeyboardInterrupt:
128.        setColor(0, 0, 0)
129.        setAngle(0)
```

PRILOG 7.2. Programski kod - SpeechRecognition Hrvatski

```
1.  from time import sleep
2.  import RPi.GPIO as GPIO
3.  import speech_recognition as sr
4.
5.  commands = {"led": "ugasi",
6.              "motor": "ugasi"}
7.
8.  commandList = [ "upali", "ugasi",
9.                  "crvena", "zelena", "plava", "žuta", "ljubičasta",
10.                 "pozicija 1", "pozicija 2", "pozicija 3", "pozicija 4", "pozicija 5"]
11.
12. RED_PIN = 3
13. GREEN_PIN = 5
14. BLUE_PIN = 7
15.
16. PWM_MOTOR = 12
17.
18. GPIO.setmode(GPIO.BOARD)
19.
20. # LED pins
21. GPIO.setup(RED_PIN, GPIO.OUT) # RED
22. GPIO.setup(GREEN_PIN, GPIO.OUT) # GREEN
23. GPIO.setup(BLUE_PIN, GPIO.OUT) # BLUE
24.
25. GPIO.output(RED_PIN, 1)
26. GPIO.output(GREEN_PIN, 1)
27. GPIO.output(BLUE_PIN, 1)
28.
29. # PWM motor pin
30. GPIO.setup(PWM_MOTOR, GPIO.OUT)
31.
32. pwm = GPIO.PWM(PWM_MOTOR, 50)
33. pwm.start(0)
34.
35. def setAngle(angle):
36.     duty = angle / 18 + 2
37.     GPIO.output(PWM_MOTOR, True)
38.     pwm.ChangeDutyCycle(duty)
39.     sleep(1)
40.     GPIO.output(PWM_MOTOR, False)
41.     pwm.ChangeDutyCycle(0)
42.
43.
44. def setColor(red, green, blue):
45.     GPIO.output(RED_PIN, 1 - red)
46.     GPIO.output(GREEN_PIN, 1 - green)
47.     GPIO.output(BLUE_PIN, 1 - blue)
```

```
48.
49. def callback(recognizer, audio):
50.     try:
51.         speech = recognizer.recognize_google(audio, language='hr-HR').lower()
52.
53.         for device in commands.keys():
54.             if device in speech:
55.                 for command in commandList:
56.                     if command in speech:
57.                         commands[device] = command
58.
59.     except sr.UnknownValueError:
60.         print("UnknownValueError")
61.     except sr.WaitTimeoutError:
62.         print("WaitTimeoutError")
63.
64. if __name__ == "__main__":
65.
66.     try:
67.         recognizer = sr.Recognizer()
68.         microphone = sr.Microphone()
69.
70.         with microphone as source:
71.             recognizer.adjust_for_ambient_noise(source)
72.
73.         recognizer.listen_in_background(microphone, callback, phrase_time_limit=3)
74.
75.         ledState = commands["led"]
76.         motorState = commands["motor"]
77.
78.         while True:
79.
80.             print(commands)
81.
82.             if commands["led"] != ledState:
83.                 ledState = commands["led"]
84.
85.                 if ledState == "upali":
86.                     setColor(1, 1, 1)
87.                 elif ledState == "ugasi":
88.                     setColor(0, 0, 0)
89.                 elif ledState == "crvena":
90.                     setColor(1, 0, 0)
91.                 elif ledState == "zelena":
92.                     setColor(0, 1, 0)
93.                 elif ledState == "plava":
94.                     setColor(0, 0, 1)
95.                 elif ledState == "žuta":
96.                     setColor(1, 1, 0)
97.                 elif ledState == "ljubičasta":
```

```
                setColor(1, 0, 1)

            if commands["motor"] != motorState:
                motorState = commands["motor"]

                if motorState == "upali":
                    setAngle(180)
                elif motorState == "ugasi":
                    setAngle(0)
                elif motorState == "pozicija 1":
                    setAngle(30)
                elif motorState == "pozicija 2":
                    setAngle(60)
                elif motorState == "pozicija 3":
                    setAngle(90)
                elif motorState == "pozicija 4":
                    setAngle(120)
                elif motorState == "pozicija 5":
                    setAngle(150)

    except KeyboardInterrupt:
        setColor(0, 0, 0)
        setAngle(0)
```

PRILOG 7.3. Programski kod za stvaranje trening značajki

```
from os import listdir, path
from os.path import isdir, join
import librosa
import random
import numpy as np
import python_speech_features

# Dataset path
dataset_path = path.abspath('../dataset')

# Create targets list
targets = [name for name in listdir(dataset_path) if isdir(join(dataset_path, name))]

# Remove background noise and up from targets list
targets.remove('_background_noise_')
targets.remove('up')

# Settings
features_file = 'mfcc_dataset.npz'
validation_ratio = 0.1
test_ratio = 0.1
sample_rate = 8000
num_mfcc = 16
len_mfcc = 16

# Create filenames and ground truth vector
filenames = []
y = []

for index, target in enumerate(targets):
    print(join(dataset_path, target))
    filenames.append(listdir(join(dataset_path, target)))
    y.append(np.ones(len(filenames[index])) * index)

# Flatten filename and y vectors
filenames = [item for sublist in filenames for item in sublist]
y = [item for sublist in y for item in sublist]

# Associate filenames and y then shuffle
filenames_y = list(zip(filenames, y))
random.shuffle(filenames_y)
filenames, y = zip(*filenames_y)

# Calculate validation and test set sizes
validation_size = int(len(filenames) * validation_ratio)
test_size = int(len(filenames) * test_ratio)

```

```
48. # Extract validation, test and train sets
49. filenames_val = filenames[:validation_size]
50. filenames_test = filenames[validation_size:(validation_size + test_size)]
51. filenames_train = filenames[(validation_size + test_size):]
52.
53. # Extract validation, test and train sets
54. y_orig_val = y[:validation_size]
55. y_orig_test = y[validation_size:(validation_size + test_size)]
56. y_orig_train = y[(validation_size + test_size):]
57.
58. # Create MFCC from given path
59. def calculate_mfcc(path):
60.
61.     # Load wavefile
62.     signal, fs = librosa.load(path, sr=sample_rate)
63.
64.     # Create MFCC from sound clip
65.     mfcc = python_speech_features.base.mfcc(signal,
66.                                             samplerate=fs,
67.                                             winlen=0.256,
68.                                             winstep=0.050,
69.                                             numcep=num_mfcc,
70.                                             nfilt=26,
71.                                             nfft=2048,
72.                                             preemph=0.0,
73.                                             ceplifter=0,
74.                                             appendEnergy=False,
75.                                             winfunc=np.hanning)
76.     return mfcc.transpose()
77.
78. # Extract features from file and create output lists
79. def extract_features(filenames, y):
80.
81.     out_x = []
82.     out_y = []
83.
84.     for index, filename in enumerate(filenames):
85.
86.         # Create path from given filename and target item
87.         path = join(dataset_path, targets[int(y[index])], filename)
88.
89.         # Check if this is wav file
90.         if not path.endswith('.wav'):
91.             continue
92.
93.         # Create MFCC
94.         mfcc = calculate_mfcc(path)
95.
96.         # Only keep MFCC with right lenght
97.         if mfcc.shape[1] == len_mfcc:
```

```
            out_x.append(mfcc)
            out_y.append(y[index])

    return out_x, out_y

# Create train, validation, and test sets
x_train, y_train = extract_features(filenames_train, y_orig_train)
x_val, y_val = extract_features(filenames_val, y_orig_val)
x_test, y_test = extract_features(filenames_test, y_orig_test)

# Save features and y
np.savez(features_file,
         x_train=x_train,
         y_train=y_train,
         x_val=x_val,
         y_val=y_val,
         x_test=x_test,
         y_test=y_test)

print("Feature extraction done ...")
```

PRILOG 7.4. Programski kod za trening modela

```
1.  import numpy as np
2.  from os import path
3.  import matplotlib.pyplot as plt
4.  from tensorflow import optimizers
5.  from tensorflow.keras import layers, models, regularizers, callbacks
6.
7.  # Parameters
8.  EPOCHS = 200
9.  BATCH_SIZE = 300
10. PATIENCE = 5
11. LEARNING_RATE = 0.001
12. CLASS_NUMBER = 34
13.
14. def prepare_data(features_path):
15.
16.     print("Preparing data...")
17.
18.     # Load features
19.     features = np.load(features_path)
20.
21.     # Extract features
22.     x_train = features['x_train']
23.     y_train = features['y_train']
24.     x_val = features['x_val']
25.     y_val = features['y_val']
26.     x_test = features['x_test']
27.     y_test = features['y_test']
28.
29.     # Shape = (batch, height, width, channels)
30.     x_train = x_train.reshape(x_train.shape[0],
31.                               x_train.shape[1],
32.                               x_train.shape[2],
33.                               1)
34.     x_val = x_val.reshape(x_val.shape[0],
35.                           x_val.shape[1],
36.                           x_val.shape[2],
37.                           1)
38.     x_test = x_test.reshape(x_test.shape[0],
39.                             x_test.shape[1],
40.                             x_test.shape[2],
41.                             1)
42.
43.
44.     return x_train, y_train, x_val, y_val, x_test, y_test
45.
46. def build_model(input_shape, class_number=CLASS_NUMBER,
    learning_rate=LEARNING_RATE):
```

```
47.
48.     print("Building model...")
49.
50.     model = models.Sequential()
51.
52.     # First Layer
53.     model.add(layers.Conv2D(150, (2, 2), activation='relu', input_shape=input_shape,
54.                             kernel_regularizer=regularizers.l2(0.001)))
55.     model.add(layers.BatchNormalization())
56.     model.add(layers.MaxPooling2D(pool_size=(2, 2)))
57.
58.     model.add(layers.Dropout(0.1))
59.
60.     # Second Layer
61.     model.add(layers.Conv2D(100, (2, 2), activation='relu',
    kernel_regularizer=regularizers.l2(0.001)))
62.     model.add(layers.BatchNormalization())
63.     model.add(layers.MaxPooling2D(pool_size=(2, 2)))
64.
65.     model.add(layers.Dropout(0.1))
66.
67.     # Third Layer
68.     model.add(layers.Conv2D(50, (2, 2), activation='relu',
    kernel_regularizer=regularizers.l2(0.001)))
69.     model.add(layers.BatchNormalization())
70.     model.add(layers.MaxPooling2D(pool_size=(2, 2)))
71.
72.     model.add(layers.Dropout(0.1))
73.
74.     # Stretch data in one dimension
75.     model.add(layers.Flatten())
76.
77.     model.add(layers.Dropout(0.1))
78.
79.     # Fourth Layer
80.     model.add(layers.Dense(64, activation='relu'))
81.
82.     # Output Layer
83.     model.add(layers.Dense(class_number, activation='softmax'))
84.
85.     adam = optimizers.Adam(learning_rate=learning_rate)
86.
87.     model.compile(loss='sparse_categorical_crossentropy', optimizer=adam,
    metrics=['accuracy'])
88.
89.     model.summary()
90.
91.     return model
92.
```

```
93. def train(model, x_train, y_train, x_val, y_val, epochs=EPOCHS,
    batch_size=BATCH_SIZE, patience=PATIENCE):
94.
95.     print("Training model...")
96.
97.     earlystop = callbacks.EarlyStopping(monitor="accuracy", min_delta=0.001,
    patience=patience)
98.
99.     # Start network training
100.        history = model.fit(x_train,
101.                            y_train,
102.                            epochs=epochs,
103.                            batch_size=batch_size,
104.                            validation_data=(x_val, y_val),
105.                            callbacks=[earlystop])
106.
107.        return history
108.
109. def save_plots(history, epochs=EPOCHS):
110.
111.        # Plot training results
112.        accuracy = history.history['accuracy']
113.        val_accuracy = history.history['val_accuracy']
114.        loss = history.history['loss']
115.        val_loss = history.history['val_loss']
116.
117.        epochs = range(1, len(accuracy) + 1)
118.
119.        fig, (accuracyPlt, lossPlt) = plt.subplots(2, sharex=True)
120.        fig.suptitle("Accuracy and Loss")
121.        accuracyPlt.plot(epochs, accuracy, color='royalblue', label='Training')
122.        accuracyPlt.plot(epochs, val_accuracy, color='orangered', label='Validation')
123.        accuracyPlt.set_ylabel('Accuracy')
124.        accuracyPlt.set_xlabel('Epochs')
125.        accuracyPlt.legend()
126.
127.        lossPlt.plot(epochs, loss, color='royalblue', label='Training')
128.        lossPlt.plot(epochs, val_loss, color='orangered', label='Validation')
129.        lossPlt.set_ylabel('Loss')
130.        lossPlt.set_xlabel('Epochs')
131.        lossPlt.legend()
132.
133.        fig.savefig('acc_loss.png')
134.
135. if __name__ == "__main__":
136.
137.        # Features and model path
138.        features_path = path.abspath('./mfcc_dataset.npz')
139.        model_file = 'model.h5'
140.
```

```
141.    # Extract data
142.    x_train, y_train, x_val, y_val, x_test, y_test = prepare_data(features_path)
143.
144.    # Input shape = MFCC of 1 sample (16, 16, 1) = (16, 16)
145.    input_shape = x_train.shape[1:]
146.
147.    # Build model
148.    model = build_model(input_shape)
149.
150.    # Train model
151.    history = train(model, x_train, y_train, x_val, y_val)
152.
153.    # Plot accuracy and loss
154.    save_plots(history)
155.
156.    # Evaluate model
157.    print("EVALUATE:")
158.    model.evaluate(x=x_test, y=y_test)
159.
160.    # Save the model as a file
161.    models.save_model(model, model_file)
```

PRILOG 7.5. Programski kod za stvaranje Lite verzije modela

```
from tensorflow import lite
from tensorflow.keras import models

keras_model_filename = 'model.h5'
tflite_filename = 'lite_model.tflite'

# Convert model to TF Lite model
model = models.load_model(keras_model_filename)
converter = lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
open(tflite_filename, 'wb').write(tflite_model)
```

PRILOG 7.6. Programski kod glavne aplikacije za upravljanje uređajima

```
import sounddevice as sd
import numpy as np
import scipy.signal
from time import sleep
import python_speech_features
import RPi.GPIO as GPIO
from tflite_runtime.interpreter import Interpreter

words = [
    'visual', 'five', 'three', 'down', 'learn', 'go',
    'no', 'nine', 'dog', 'left', 'two', 'on', 'backward',
    'right', 'happy', 'sheila', 'follow', 'zero',
    'tree', 'marvin', 'stop', 'wow', 'eight', 'seven', 'bird',
    'cat', 'yes', 'house', 'six', 'bed', 'off', 'forward', 'one', 'four'
    ]

word_buffer = []
prediction_counter = 0

# Settings
word_threshold = 0.5
rec_duration = 0.5
sample_rate = 48000
resample_rate = 8000
channels_num = 1
num_mfcc = 16
model_path = 'lite_model.tflite'

# Sliding window
window = np.zeros(int(rec_duration * resample_rate) * 2)

RED_PIN = 3
GREEN_PIN = 5
BLUE_PIN = 7

PWM_MOTOR = 12

GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)

# LED pins
GPIO.setup(RED_PIN, GPIO.OUT) # RED
GPIO.setup(GREEN_PIN, GPIO.OUT) # GREEN
GPIO.setup(BLUE_PIN, GPIO.OUT) # BLUE

GPIO.output(RED_PIN, 1)
GPIO.output(GREEN_PIN, 1)
```

```
GPIO.output(BLUE_PIN, 1)

# Motor pin
GPIO.setup(PWM_MOTOR, GPIO.OUT) # PWM

# PWM init
pwm = GPIO.PWM(PWM_MOTOR, 50)
pwm.start(0)

def setAngle(angle):
    duty = angle / 18 + 2
    GPIO.output(PWM_MOTOR, True)
    pwm.ChangeDutyCycle(duty)
    sleep(1)
    GPIO.output(PWM_MOTOR, False)
    pwm.ChangeDutyCycle(0)

def setColor(red, green, blue):
    GPIO.output(RED_PIN, 1 - red)
    GPIO.output(GREEN_PIN, 1 - green)
    GPIO.output(BLUE_PIN, 1 - blue)

# Load model (interpreter)
interpreter = Interpreter(model_path)
interpreter.allocate_tensors()
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
print(input_details)

# Decimate = filter and downsample
def decimate(signal, old_sr, new_sr):

    # Check for downsampling
    if new_sr > old_sr:
        print("Error: target sample rate higher than original")
        return signal, old_sr

    # Downsample by integer factor
    dec_factor = old_sr / new_sr
    if not dec_factor.is_integer():
        print("Error: can only decimate by integer factor")
        return signal, old_sr

    # Do decimation
    resampled_signal = scipy.signal.decimate(signal, int(dec_factor))

    return resampled_signal, new_sr

def modify_mic_sample(recording):
```

```
    # Remove second dimension from sample
    recording = np.squeeze(recording)

    # Resample
    recording, new_sr = decimate(recording, sample_rate, resample_rate)

    # Save recording with sliding window
    window[:len(window)//2] = window[len(window)//2:]
    window[len(window)//2:] = recording

    return window, new_sr

def get_mfccs(window, new_sr):

    # Compute features
    mfccs = python_speech_features.base.mfcc(window,
                                samplerate=new_sr,
                                winlen=0.256,
                                winstep=0.050,
                                numcep=num_mfcc,
                                nfilt=26,
                                nfft=2048,
                                preemph=0.0,
                                ceplifter=0,
                                appendEnergy=False,
                                winfunc=np.hanning)
    mfccs = mfccs.transpose()

    return mfccs

def make_prediction(mfccs):

    # Make prediction
    in_tensor = np.float32(mfccs.reshape(1, mfccs.shape[0], mfccs.shape[1], 1))
    interpreter.set_tensor(input_details[0]['index'], in_tensor)
    interpreter.invoke()
    output_data = interpreter.get_tensor(output_details[0]['index'])

    # Get index with max probability
    index = np.argmax(output_data)
    word = words[index]
    probability = output_data[0][index]

    return index, word, probability

def control_devices(word_buffer, index, word_probability):

        # Execute command for specific device
        if "house" in word_buffer and "one" in word_buffer:
            setAngle(30)
```

```
print(f"\033[32mindex={index:<10}word={words[index]:<10}probability={word_probability}\
033[0m")
        elif "house" in word_buffer and "two" in word_buffer:
            setAngle(90)

print(f"\033[32mindex={index:<10}word={words[index]:<10}probability={word_probability}\
033[0m")
        elif "house" in word_buffer and "three" in word_buffer:
            setAngle(120)

print(f"\033[32mindex={index:<10}word={words[index]:<10}probability={word_probability}\
033[0m")
        elif "house" in word_buffer and "on" in word_buffer:
            setAngle(180)

print(f"\033[32mindex={index:<10}word={words[index]:<10}probability={word_probability}\
033[0m")
        elif "house" in word_buffer and "off" in word_buffer:
            setAngle(0)

print(f"\033[32mindex={index:<10}word={words[index]:<10}probability={word_probability}\
033[0m")
        elif "sheila" in word_buffer and "one" in word_buffer:
            setColor(1, 0, 0)

print(f"\033[32mindex={index:<10}word={words[index]:<10}probability={word_probability}\
033[0m")
        elif "sheila" in word_buffer and "two" in word_buffer:
            setColor(0, 1, 0)

print(f"\033[32mindex={index:<10}word={words[index]:<10}probability={word_probability}\
033[0m")
        elif "sheila" in word_buffer and "three" in word_buffer:
            setColor(0, 0, 1)

print(f"\033[32mindex={index:<10}word={words[index]:<10}probability={word_probability}\
033[0m")
        elif "sheila" in word_buffer and "on" in word_buffer:
            setColor(1, 1, 1)

print(f"\033[32mindex={index:<10}word={words[index]:<10}probability={word_probability}\
033[0m")
        elif "sheila" in word_buffer and "off" in word_buffer:
            setColor(0, 0, 0)

print(f"\033[32mindex={index:<10}word={words[index]:<10}probability={word_probability}\
033[0m")

# Executes every 0.5 seconds
```

```
def callback(recording, frames, time,  status):

    # Notify if errors
    if status:
        print('Error:', status)

    window, new_sr = modify_mic_sample(recording)
    mfccs = get_mfccs(window, new_sr)
    index, word, word_probability = make_prediction(mfccs)
    print(f"index={index:<10}word={words[index]:<10}probability={word_probability}")

    global prediction_counter
    prediction_counter += 1
    print(prediction_counter)

    if word_probability > word_threshold:
        word_buffer.append(word)

        if len(word_buffer) == 2:
            control_devices(word_buffer, index, word_probability)

            # Reset Buffer
            prediction_counter = 0
            print(word_buffer)
            word_buffer.clear()

    # Reset Buffer
    if prediction_counter == 10:
        word_buffer.clear()
        prediction_counter = 0

if '__main__' == __name__:
    try:
        # Start listening to the microphone
        with sd.InputStream(channels=channels_num,
                            samplerate=sample_rate,
                            blocksize=int(sample_rate * rec_duration),
                            callback=callback):
            while True:
                pass

    except KeyboardInterrupt:
        SetAngle(0)
        setColor(0, 0, 0)
```