SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

DIPLOMSKI RAD br. 720

# SUSTAV ZA PREPOZNAVANJE GOVORNIH NAREDBI U STVARNOM VREMENU NA RUBNIM UREĐAJIMA

Luka Balić

Zagreb, veljača 2025.

**SVEUČILIŠTE U ZAGREBU**
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

Zagreb, 30. rujna 2024.

# DIPLOMSKI ZADATAK br. 720

| | |
|---|---|
| Pristupnik: | **Luka Balić (0036513380)** |
| Studij: | Računarstvo |
| Profil: | Računalno inženjerstvo |
| Mentor: | prof. dr. sc. Hrvoje Džapo |

Zadatak: **Sustav za prepoznavanje govornih naredbi u stvarnom vremenu na rubnim uređajima**

Opis zadatka:

Proučiti metode i algoritme za analizu i klasifikaciju zvučnih sekvenci u stvarnom vremenu, s naglaskom na pristupe temeljene na dubokom učenju i primjenu u prepoznavanju govornih naredbi. Proučiti programske okvire za razvoj modela prikladnih za prepoznavanje govornih naredbi i implementaciju u ugradbenim računalnim sustavima s ograničenim resursima. Proučiti mogućnosti i ograničenja u primjeni procesora iz porodice ESP32 za implementaciju rješenja za prepoznavanje glasovnih naredbi u stvarnom vremenu te odabrati prikladnu procesorsku porodicu. Implementirati pokazni sustav koji će omogućiti prepoznavanje glasovnih naredbi u stvarnom vremenu. Razviti programsku potporu za mikrokontroler koja će omogućiti snimanje zvuka preko mikrofona, izvođenje modela za prepoznavanje glasovnih naredbi te poduzimanje jednostavnih akcija na svaku prepoznatu naredbu. Detaljno dokumentirati postupak treniranja i prilagodbe modela za izvođenje na ugradbenom računalnom sustavu. Eksperimentalno provesti ispitivanje značajki sustava i uspješnosti prepoznavanja glasovnih naredbi pod različitim uvjetima rada.

Rok za predaju rada: 14. veljače 2025.

*Hvala mojoj obitelji i prijateljima na neizmjernoj podršci tijekom cjelokupnog školovanja.*

# Sadržaj

# 1. Uvod

Modeli strojnog učenja revolucionarizirali su tehnologiju koju koristimo u svakodnevnom životu tako što su omogućili računalima učenje iz podataka kojima raspolažu u svrhu donošenja odluka u situacijama za koje nisu eksplicitno programirana. Tradicionalno su takvi modeli bili namijenjeni za računala visokih performansi i neograničavajućih resursa, međutim sve bržim razvojem IoT (engl. *Internet of Things*) područja te zahvaljujući pristupačnoj cijeni mikrokontrolerskih sustava, počela je prilagodba modela strojnog učenja za takve sustave. Na prvi pogled dva nespojiva svijeta su se susrela te pronašla svoju primjenu u raznovrsnim sustavima. Tiny ML (engl. *Tiny Machine Learning*) je naziv koji se odnosi na implementaciju modela strojnog učenja na uređaje ograničenih resursa kao što su mobilni telefoni i mikrokontroleri. Glavne karakteristike modela namijenjenih za takve sustave su relativno malen memorijski otisak, mogućnost odziva u stvarnom vremenu, smanjenje potrošnje i internetskog prometa te sigurnost [1]. Sustav implementiran kroz ovaj rad ima zadatak prepoznati unaprijed zadane glasovne naredbe te pokrenuti izvršavanje određenog posla vezanog uz specifičnu naredbu.

Okruženi smo digitalnim glasovnim asistentima kao što su Googleov Assistant, Appleova Siri te Amazonova Alexa. Ovakvi sustavi mogu u vrlo kratkom roku pružiti zatražene informacije i bez ikakvog problema komunicirati s osobom koja ih koristi. Za prepoznavanje i obradu ljudskog govora i dohvaćanje bitnih informacija zaduženi su modeli kojima je potrebna velika procesorska moć i dovoljno prostora za pohranu te se zbog toga taj dio posla odrađuje na serverskim računalima. Takav sustav podrazumijeva konstantnu internetsku vezu uz stabilan i dugotrajan izvor električne energije. Kada bi mobilni uređaji slali konstantan tok zvučnih podataka na server, brzo bi ispraznili bateriju te nepotrebno koristili mobilne podatke za pristup internetu. Zbog toga su takvi sustavi osmišljeni da čekaju naredbu za početak komunikacije, a tek onda uspostave

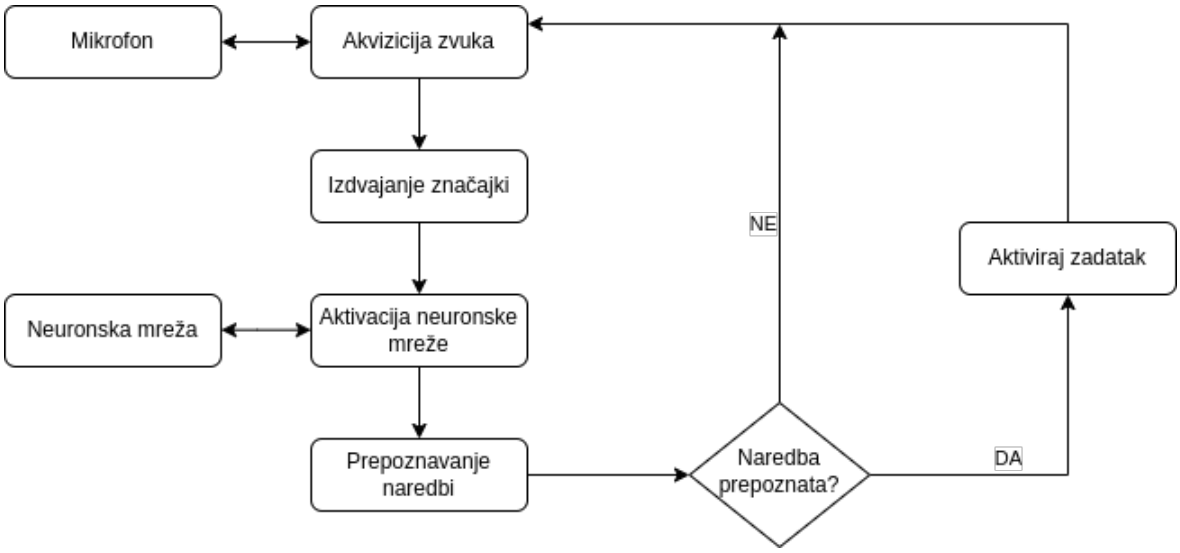
vezu sa serverom. Međutim, i dalje nam ostaje problem konstantne akvizicije ulaznih zvučnih podataka te prepoznavanje naredbe kao što je ”Hey Google” ili nešto slično. U ovoj situaciji savršenu primjenu pronašli su procesori izrazito male potrošnje na kojima je moguće implementirati optimirane modele strojnog učenja. Takav procesor bi konstantno prikupljao podatke s mikrofona te lokalno, uz pomoć treniranog modela, čekao ključnu riječ nakon koje bi dao znak cijelom sustavu da se može ”probuditi” iz stanja niske potrošnje te odraditi svoj posao. Ovakvim pristupom postignuta je efikasnost, niska potrošnja, brz odaziv, smanjena potrošnja internetskog prometa te možda i najvažnija stvar - privatnost. Naime, nema potrebe za konstantnim slanjem glasovnih podataka na server što omogućuje da na server dospiju samo glasovni isječci u željenim trenucima.

# 2. Arhitektura i implementacija sustava

Svrha cjelokupnog sustava je detekcija određenog glasovnog uzorka (glasovne naredbe) koja se može koristiti za okidanje obavljanja određenog procesa, pri čemu se prepoznavanje naredbe implementira na ugradbenom računalnom sustavu niske potrošnje i procesorske moći. Također, potrebno je da sustav može pokrenuti proces u bilo kojem trenutku te bi odaziv trebao biti sa što manjom latencijom, što znači da cijeli proces prikupljanja zvuka, obrade te izvršavanja akcije mora raditi kontinuirano u stvarnom vremenu. Centralni dio sustava koji je zadužen za samo prepoznavanje glasovne naredbe implementiran je u vidu neuronske mreže trenirane na skupu zvučnih zapisa koji sadrže željene naredbe, a čija je struktura detaljnije objašnjena u poglavlju koje opisuje samu neuronsku mrežu 3. Imajući u vidu spomenute ciljeve, sustav je izgrađen od četiri modularna podsustava koja su u cijelosti implementirana na odabranom mikrokontroleru, a to su:

- prikupljanje zvuka,
- izdvajanje značajki,
- aktivacija neuronske mreže,
- prepoznavanje i aktivacija naredbi.

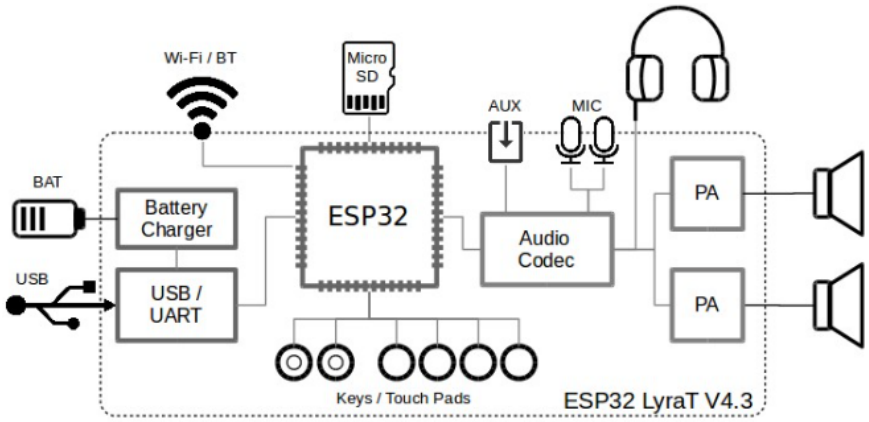
Navedena podjela na podcjeline olakšava prilagodbu cjelokupnog sustava na različite mikrokontrolerske platforme i porodice, što uključuje drugačiju obradu prikupljenog zvuka, korištenje neke drugačije neuronske mreže ili samo aktiviranje specifične akcije koja će se pokrenuti određenom glasovnom naredbom. Na slici 2.1. grafički je prikazan opisani sustav.




**Slika 2.1.** Struktura sustava na mikrokontroleru[2]

## 2.1. Akvizicija zvuka

Podsustav za akviziciju zvuka je usko vezan uz računalnu platformu na kojoj je sustav implementiran jer izravno komunicira sa sustavom koji prikuplja stvarne signale sa senzora (mikrofona). Kao platforma za implementaciju opisanog sustava odabran je razvojni sustav ESP32 Lyrat Development Board [3] koji ima već ugrađen mikrofon i audio kodek s kojim je moguće komunicirati putem I2S protokola. Na slici 2.2. prikazan je blok dijagram ESP32 razvojne platforme.




**Slika 2.2.** ESP32 Lyrat [3]

Najvažniji dio razvojne platforme je ESP32-WROVER-E mikrokontroler na kojem je implementirano cjelokupno programsko rješenje za prepoznavanje govornih naredbi. Upravo on je taj mikrokontroler zadužen za komunikaciju s ES8388 audio kodekom [4]. ES8388 je integrirani sklop koji na spomenutoj platformi služi za analogno-digitalnu

(engl. *ADC*) te digitalno-analognu (engl. *DAC*) pretvorbu, tj. upravljanje audio ulazom (mikrofon) te audio izlazom (AUX priključak). Mikrokontroler putem I2C sučelja konfigurira kodek, dok zvučne signale prikuplja s mikrofona putem I2S sučelja. Budući da je ESP32 Lyrat platforma prilagođena razvoju audio sustava, konfiguracija i puštanje u pogon akvizicije zvuka je vrlo jednostavno. U programskom kodu 2.1 prikazana je inicijalizacija kodeka.

```
    audio_board_handle_t board_handle = audio_board_init();
    audio_hal_ctrl_codec(board_handle->audio_hal,
   AUDIO_HAL_CODEC_MODE_ENCODE, AUDIO_HAL_CTRL_START);
```

Kod 2.1: Inicijalizacija kodeka

Sav posao koji se nalazi iza prikazanih naredbi obavlja ESP-ADF (engl. *Espressif Audio Development Framework*). To je biblioteka koja pojednostavljuje razvoj aplikacija vezanih za obradu zvuka i osnovne zadatke, kao što su akvizicija, kodiranje i dekodiranje različitih formata audio zapisa te komunikacija s računalom ili memorijskom karticom. Nakon inicijalizacije kodeka i definiranja konfiguracije postavki I2S komunikacije, potrebno je pokrenuti protočni akvizicijski sustav (engl. *pipeline*), a to je prikazano u programskom kodu 2.2 .

```
    ESP_LOGI(TAG, "Creating i2s stream");
    audio_element_handle_t i2s_stream_reader;
    i2s_stream_cfg_t i2s_cfg = I2S_STREAM_CFG_DEFAULT();
    i2s_cfg.type = AUDIO_STREAM_READER;
    i2s_cfg.i2s_port = I2S_NUM_0;
    i2s_cfg.i2s_config = i2s;
    i2s_stream_reader = i2s_stream_init(&i2s_cfg);

    ESP_LOGI(TAG, "Creating audio pipeline");
    audio_pipeline_handle_t pipeline;
    audio_pipeline_cfg_t pipeline_cfg = DEFAULT_AUDIO_PIPELINE_CONFIG();
    pipeline = audio_pipeline_init(&pipeline_cfg);
    audio_pipeline_register(pipeline, i2s_stream_reader, "i2s");
    audio_pipeline_run(pipeline);
```

Kod 2.2: Pokretanje akvizicijskog podsustava

Frekvencija otipkavanja postavljena u konfiguraciji iznosi 16000 Hz što je dovoljno za

ovakav tip sustava za prepoznavanje glasovnih naredbi [5]. Nakon pokretanja akvizicijskog podsustava preostalo je uzimati nove podatke s kraja protočne strukture podsustava. Konstantno uzimanje novih uzoraka, tj. komunikacija s ES8388 kodekom, odvojeno je u posebnu dretvu. Tako je programski odvojena akvizicija sirovih uzoraka zvuka od ostatka sustava u kojima se obavlja obrada nad njima. Ovaj podsustav je proizvođač novih podataka, dok je ostatak sustava potrošač, koji je također realiziran u posebnoj dretvi. Kako su prikupljanje podataka i njihova obrada realizirani pomoći dvije odvojene dretve, potrebno je implementirati pouzdan i efikasan način komunikacije između njih. Za tu svrhu odabran je kružni međuspremnik (engl. *ring buffer*) koji omogućuje asinkrono pisanje i čitanje. Implementacija takve strukture dostupna je kao objekta FreeRTOS-a [6], koji je odabran kao operacijski sustav za rad u stvarnom vremenu na kojem je realizirano višedretveno rješenje.

```
class AudioRecorder
{
    private:
        uint32_t sampleRate;
        RingbufHandle_t ringBuffer;
        TaskHandle_t captureAudioHandle;
        static void captureAudioTask(void* pvParameters);
    public:
        AudioRecorder(uint32_t sampleRate) : sampleRate(sampleRate),
                                             ringBuffer(NULL),
                                             captureAudioHandle(NULL) {}
        uint32_t getSamples(int16_t* samples, size_t numOfSamples);
};
```

Kod 2.3: Razred AudioRecorder

Opisani podsustav za akviziciju zvuka u programskom kodu strukturiran je u razred imena `AudioRecorder` 2.3 kako bi ga se lakše koristilo. Razred sadrži metodu `uint32_t getSamples(int16_t* samples, size_t numOfSamples)` koja omogućuje dohvaćanje proizvoljnog broja `uint16_t` podataka jer se akvizirani zvučni signal pohranjuje u tom obliku. U glavnoj petlji sustava potrebno je kreirati objekt razreda `AudioRecorder`, alocirati memorijski spremnik za dohvaćanje podataka i kontinuirano pozivati navedenu funkciju koja kao argumente prima pokazivač na alocirani spremnik i željeni broj audio

uzoraka za dohvat.

Spomenuta modularnost ostvarena je tako što je funkcionalnost prikupljanja novih zvučnih uzoraka sadržana u opisanom razredu. Pokretanje cjelokupnog sustava na drugoj platformi koja ima drugi audio kodek, drugačiji mikrofon ili se radi o drugom mikrokontroleru, bit će moguće implementiranjem novog razreda koji podržava zadano drugačije sučelje.

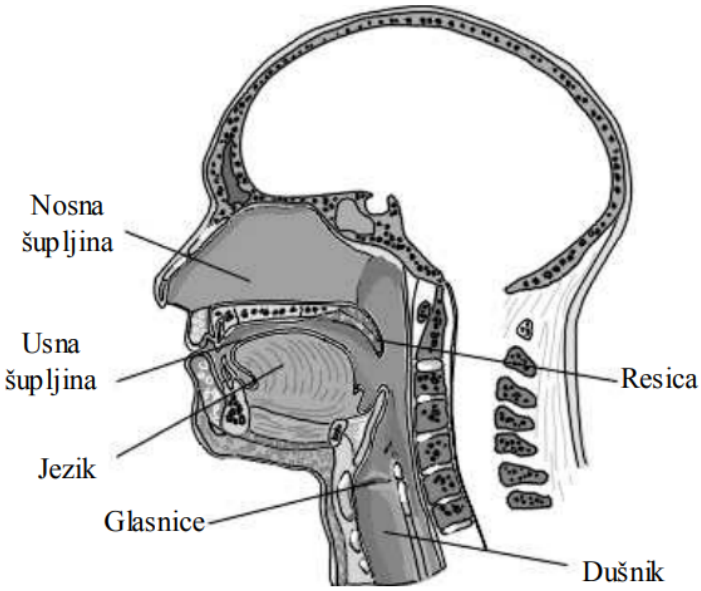
## 2.2. Izdvajanje značajki

Izdvajanje značajki (engl. *feature extraction*) drugi je podsustav implementiranog sustava i prikazan je na slici 2.1. Nakon što je provedena akvizicija zvučnih uzoraka, potrebno ih je obraditi kako bi se mogli predati neuronskoj mreži na klasifikaciju. Naime, modeli strojnog učenja tipično ne rade izravno nad sirovim podacima, već na izdvojenim značajkama dobivenima analizom ulaznih podataka. Jedan tipičan primjer koji se koristi kako bi se približio pojam značajki podataka je model predikcije cijene nekretnine. U tom slučaju kao značajke koje bi prediktivni model mogao koristiti su lokacija, godina izgradnje, površina, razina energetske učinkovitosti i sl. Najjednostavnija značajka kod zvučnog signala koju je moguće izravno izdvojiti je vrijednost uzorka, ali to sasvim sigurno nije dobar odabir jer bi se u model trebalo ubaciti 16000 uzoraka u sekundi, što nije prikladno ne samo za mikrokontroler s ograničenom računalnom moći. Također, same vrijednosti neobrađenih uzoraka ne pružaju dovoljnu distinkciju između različitih klasa, već je potreban sofisticiraniji pristup izdvajanju značajki koji će rezultirati i bitno manjom količinom podataka, što će biti obrazloženo u nastavku.

### 2.2.1. Model govora

Kako bi se identificiralo kojim značajkama bi bilo korisno opisati ljudski govor i time glasovne naredbe, potrebno je razumjeti na koji način se modelira nastanak glasa. Prilikom govorne komunikacije, pluća govornika se pod djelovanjem mišića prsnog koša stišću i potiskuju zrak kroz vokalni trakt čiji su glavni dijelovi prikazani na slici 2.3.

Glasnice (engl. *glottis*) su vrlo značajan organ u procesu formiranja govora. Ponašaju se kao mehanički oscilator koji prelazi u stanje relaksacijskih oscilacija uslijed struje




**Slika 2.3.** Presjek glave i osnovni dijelovi vokalnog trakta koji sudjeluju u produkciji govornog signala [7]

zraka iz pluća koja kroz njih prolazi. Na frekvenciju njihovog titranja utječu brojni parametri, a među najznačajnijim su pritisak zraka iz pluća na ulazu u glasnice i napetost samih glasnica. Takvim periodičkim titranjem, glasnice formiraju periodičku struju zraka, tj. kvazi-periodične impulse (engl. *glottal pulse*) koja zatim prolaze kroz ostatak vokalnog trakta što vodi do stvaranja artikuliranih glasova. U slučaju da su glasnice potpuno opuštene, neće doći do oscilacija i struja zraka iz pluća će neometano prolaziti kroz vokalni trakt (tada se ne formira kvazi-periodični impuls, nego je rezultat prolaska zraka kroz glasnice slučajni šum, a rezultat cijelog procesa je stvaranje neartikuliranih glasova).

S druge strane, vokalni trakt se ponaša kao filtar koji spektralno mijenja karakteristiku pobudnog signala (engl. *vocal tract frequency response*). Geometrijom vokalnog trakta, koja se mijenja ovisno o položaju artikulatora kao što su jezik, usne, čeljust i resica, bit će određen ton (visina i spektralni sastav) formiranog signala (govora) [7].

Jednostavni model koji se koristi u području obrade prirodnog govora je da se govor može prikazati kao izlaz iz linearnog, vremenski promjenjivog sustava čija se svojstva sporo mijenjaju s vremenom. Međutim, ako se promatraju dovoljno kratki segmenti govornog signala, svaki se segment može učinkovito modelirati kao izlaz iz linearnog, vremenski invarijantnog sustava pobuđenog bilo kvazi-periodičnim impulsima bilo slučaj-

nim šumom (engl. *random noise signal*). Opisani sustav može se prikazati jednadžbom (2.1) .

$$X(f) = E(f) \cdot H(f) \tag{2.1}$$

gdje su:

- $X(f)$: odziv sustava (govor),
- $E(f)$: pobuda (kvazi-periodični impuls),
- $H(f)$: prijenosna funkcija vokalnog trakta.

U vremenskoj domeni isti sustav može se prikazati jednadžbom (2.2) . Množenju u frekvencijskoj domeni istovjetna je konvolucija u vremenskoj i obratno:

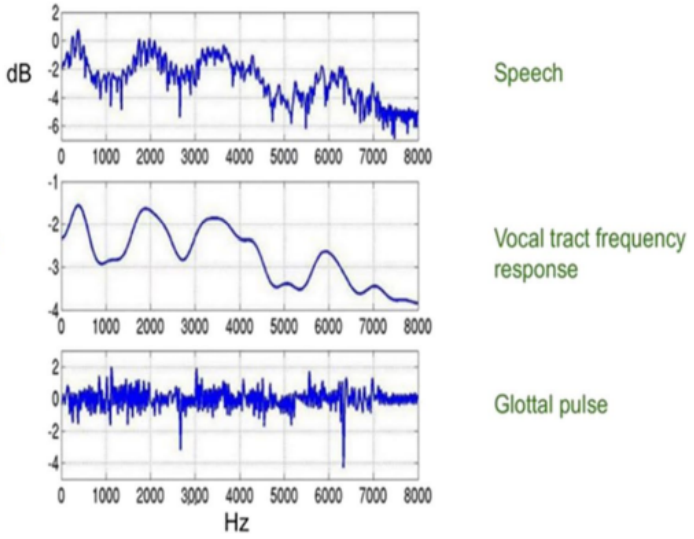
$$x(t) = e(t) * h(t) \tag{2.2}$$

Cilj ovakvog modeliranja je pronaći bitne informacije u govornom signalu. Pretpostavka na kojoj se temelji daljnji rad je da je gotovo sva bitna informacija iz govornog signala sadržana u prijenosnoj funkciji govornog trakta, tj. pobuda (kvazi-periodični impulsi) ostaje konstantna tijekom govora. Premda to predstavlja veliko pojednostavljenje stvarne situacije, gotovo svi modeli na kojima se temelji ovo područje zasnivaju na takvom pristupu [8, 9, 10]. Ako se primijeni logaritamska funkcija na sustav opisan u frekvencijskoj domeni, dobiva se:

$$\begin{aligned} \log(X(f)) &= \log(E(f) \cdot H(f)) \\ \log(X(f)) &= \log(E(f)) + \log(H(f)) \end{aligned} \tag{2.3}$$

Prikazanim formulama uspješno su razdvojeni elementi modela, tj. vidljiv je rastav na pribrojnike ako se primijeni logaritamska skala. Na slici 2.4. prikazan je utjecaj komponenata modela na konačni rezultat, a to je glas (engl. *speech*).

Preostaje pronaći način za što efikasniji opis odziva vokalnog trakta. On će u konač-




**Slika 2.4.** Glasovni signal rastavljen na pobudu i odziv vokalnog trakta [10]

nici predstavljati zvučne zapise na kojima će model neuronske mreže biti treniran. U području automatskog prepoznavanja govora te identifikaciji govornika uvelike se koriste tzv. Mel-kepstralni koeficijenti.

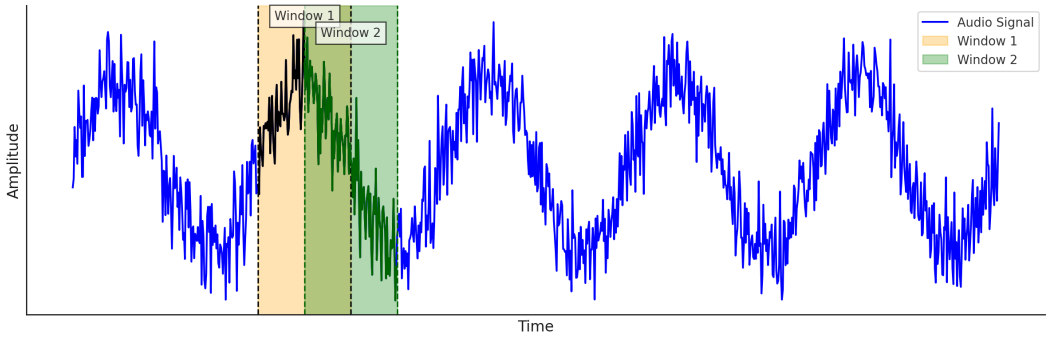
### 2.2.2. Mel-kepstralni koeficijenti

Mel-kepstralni koeficijenti (engl. *Mel frequency cepstral coefficients ili MFCC*) predstavljaju mjeru euklidske udaljenosti MFCC vektora i jedna su od najčešće korištenih mjera u automatskom prepoznavanju govora i govornika [11]. MFC koeficijenti (koji čine MFC vektor) pokazali su se odličnim načinom za pohranjivanje informacije koja je sadržana u govoru, a konkretno predstavljaju kratkotrajni spektar snage glasovnog signala. Sustav koji se razvija u okviru ovog rada za prepoznavanje govornih naredbi će koristiti upravo te koeficijente za opis značajki koje predstavljaju zvučni signal akviziran s pomoću podsustava za akviziciju. Najbolji način za razumijevanje ovih koeficijenata je prikaz postupka kojim se oni dobivaju, što će biti prikazano u nastavku.

#### Preklapajući prozori

Izlaz iz podsustava za akviziciju je signal koji predstavlja zvuk iz okoline uređaja prikupljen senzorom, a nove uzorke je moguće dobiti kontinuiranim pozivima prikladne metode tog podsustava na način opisan u poglavlju 2.1. Kontinuirani dotok novih uzoraka potrebno je podijeliti u vremenske okvire koji se kontinuirano obrađuju. Ovaj pris-

tup opisan je u poglavlju 2.2.1. u kojem je predstavljen model nastajanja govora, a kako bi opisani model dobro radio, potrebno ga je promatrati kroz kratke isječke signala u kojima su ton i visina signala stabilni. Također, potrebno je prilikom uzimanja novog vremenskog okvira postići određeno preklapanje s prethodnim okvirom, odnosno prozorom (engl. *window*) koji se pomiče po akviziranom signalu što je prikazano na slici 2.5. Zelenom bojom je prikazan je prozor u iteraciji koja slijedi nakon prethodnog prozora obojenog svijetlo-smeđom bojom. Jedan dio se preklapa, a određeni dio su novi uzorci dobiveni od podsustava za akviziciju.




**Slika 2.5.** Preklapajući prozori

U glavnoj petlji programskog koda inicijalizirano je polje koje sprema nove uzorke te polje koje zaduženo za spremanje trenutačnog prozora podataka. Alokacija polja i algoritam pomicanja starih podataka u polju koje čuva trenutačni prozor prikazani su u isječku programskog koda 2.4 .

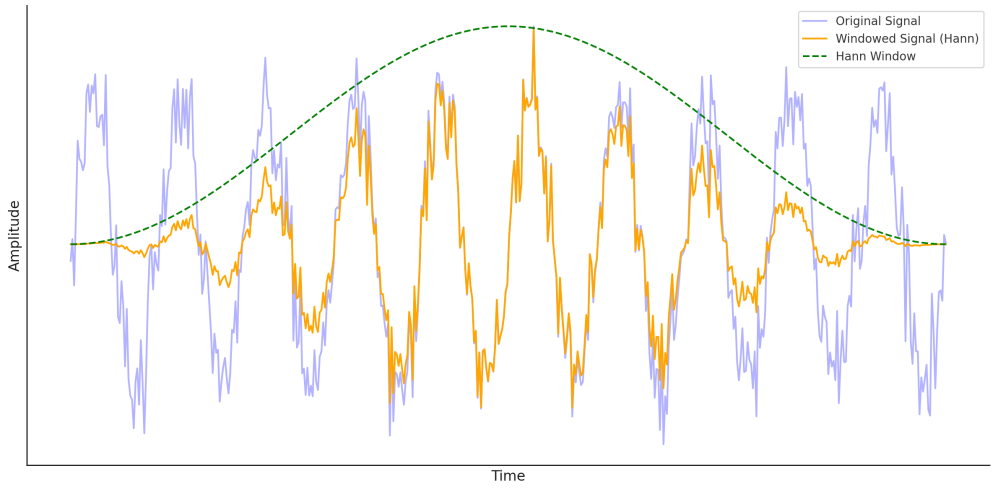
```
1 /* polje za nove uzorke */
2 int16_t newSamples[STEP_SIZE];
3 /* polje za trenutni prozor */
4 int16_t audioFrame[WINDOW_SIZE] = {0};
5 /* pomicanje starih podataka prema pocetku aktivnog prozora */
6 memcpy(audioFrame, audioFrame + STEP_SIZE, (WINDOW_SIZE - STEP_SIZE)*2);
7 /* stavljanje novih podataka na kraj aktivnog prozora */
8 memcpy(audioFrame + WINDOW_SIZE - STEP_SIZE, newSamples, STEP_SIZE*2);
```

Kod 2.4: Algoritam za korištenje novodohvaćenih podataka

`WINDOW_SIZE` predstavlja konfigurabilnu veličinu prozora, dok `STEP_SIZE` predstavlja konfigurabilni broj novih uzoraka koji će biti dohvaćeni (veličina koraka prozora). Spomenute konstante, kao i ostali konfigurabilni parametri, definirani su u datoteci `Configuration.hpp`, a prikazani u dodatku B .

### Funkcija vremenskog otvora

Nakon što je ustanovljen način dohvaćanja sirovih zvučnih uzoraka, potrebno je nad pojedinačnim prozorom podataka napraviti sve što je potrebno kako bismo dobili MFC koeficijente za takav zvučni isječak. Najprije je potrebno primijeniti funkciju vremenskog otvora (engl. *window function*). Budući da je svaki prozor (vremenski otvor) podataka opisan u 2.2.2. jednostavno odrezan od ostatka signala, tj. svega što je ostalo izvan prozora, dolazi do rasipanja energije po frekvencijskom spektru ili spektralnog curenja (engl. *spectral leakage*). Spektralno curenje je dobilo ime po tome što se energija sadržana u jednoj frekvencijskoj komponenti prelijeva u susjedne što čini spektar manje preciznim. Zbog toga je potrebno pomnožiti originalni prozor signala s funkcijom vremenskog otvora. Postoje različite funkcije koje se koriste u tu svrhu, a u obradi govora najčešće se koristi Hannov prozor (otvor) [12]. Na slici 2.6. prikazan je umnožak funkcije Hannovog prozora s funkcijom koja predstavlja zvučni signal. Na konačnom signalu je vidljivo da se rubni dijelovi signala vrijednostima približavaju nuli što sprječava diskontinuitet signala i jačanje spektralnih komponenti koje se nalaze u blizini onih od kojih je signal stvarno sastavljen.




**Slika 2.6.** Hannov prozor

Opisani postupak množenja originalnog prozora signala Hannovim prozorom prikazan je matematičkim izrazom (2.4) .

$$x_w[n] = x[n] \cdot w[n] \tag{2.4}$$

gdje su:

- $x_w[n]$: signal nakon primjene Hannovog prozora,
- $x[n]$: originalni diskretni signal,
- $w[n]$: Hann prozor definiran kao:

$$w[n] = 0.54 - 0.46 \cdot \cos\left(\frac{2\pi n}{N-1}\right) \tag{2.5}$$

za $n = 0, 1, 2, \ldots, N-1$, gdje je $N$ ukupni broj uzoraka u prozoru.

Budući da su uzorci tipa `int16_t` (16-bitni broj), raspon vrijednosti signala prije množenja s funkcijom prozora je $[-2^{15}, 2^{15}-1]$, tj. [-32768, 32767]. Uz množenje signala funkcijom prozora, cjelokupni signal skalirat će se na raspon [-1, 1] dijeljenjem svakog uzorka s 32768. Dobiveni signal veličine `WINDOW_SIZE` i tipa `float` (realni broj) je nakon opisane obrade spreman za spektralnu analizu koja podrazumijeva korištenje diskretne Fourierove transformacije.

### Diskretna Fourierova transformacija

Diskretna Fourierova transformacija (engl. *Discrete Fourier Transform ili DFT*) matematička je funkcija koja se koristi za analizu diskretnih signala u frekvencijskoj domeni. DFT diskretnog signala $x[n]$ s $N$ uzoraka definirana je izrazom (2.6) .

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi kn/N}, \quad k = 0, 1, \ldots, N-1 \tag{2.6}$$

gdje su:

- $X[k]$: spektralni koeficijent na $k$-toj frekvenciji,
- $x[n]$: signal u vremenskoj domeni,
- $N$: broj uzoraka signala.

Međutim, računalno je zahtjevna te se zbog toga koristi puno brži algoritam izračuna frekvencijskih komponenata koji se zove brza Fourierova transformacija (engl. *Fast Fo-*

*urier Transform ili FFT*). To je učinkovita računalna implementacija DFT-a koja značajno smanjuje broj operacija potrebnih za izračun koeficijenata spektra.

Na mikrokontrolerskom sustavu koji se koristi za implementaciju dostupne su funkcije koje izračunavaju spektralne koeficijente za dani signal. Programski kod 2.5 prikazuje inicijalizacijski kod potreban za pravilnu upotrebu FFT-a te metodu `void FFT::compute(float* frame, float* spectrogram)` razreda `FFT` koja izračunava koeficijente koji predstavljaju kvadratnu vrijednost amplitude svake frekvencijske komponente (spektar snage signala) kojih ima `WINDOW_SIZE / 2 + 1` (još definirano kao `NUMBER_OF_SPECTROGRAM_BINS`). Funkcija prima pokazivač na prozor (`float* frame`) te pokazivač na polje u koje će biti spremljen rezultat FFT-a (`float* spectrogram`). Izlazna varijabla je naznačena kao `spectrogram` jer se spektralnom analizom signala dobiva tzv. spektrogram koji predstavlja dvodimenzionalno polje koje sadrži više jednodimenzionalnih vektora koji se dobivaju nakon Fourierove transformacije jednog prozora.

```
1   dsps_fft2r_init_fc32(NULL, WINDOW_SIZE);  // initialization
2   void FFT::compute(float* frame, float* spectrogram) {
3       for(size_t i = 0; i < WINDOW_SIZE; i++) {
4           data[2 * i] = frame[i];  // Real part
5           data[2 * i + 1] = 0;     // Imaginary part
6       }
7       dsps_fft2r_fc32_ae32(data, WINDOW_SIZE); // Perform FFT
8       dsps_bit_rev_fc32_ansi(data, WINDOW_SIZE); // Bit reversal
9       // Magnitude (power) of each frequency bin
10      for(size_t i = 0; i < NUMBER_OF_SPECTROGRAM_BINS; i++) {
11          float real = data[2 * i];
12          float imag = data[2 * i + 1];
13          spectrogram[i] = sqrt(real * real + imag * imag);
14      }
15  }
```

Kod 2.5: FFT

### Melovo skaliranje

Izlaz iz FFT modula je frekvencijski spektar signala. Svaki element izlaznog vektora je amplituda snage određene frekvencije sadržane u signalu. Te frekvencije raspoređene su linearno. Međutim, čovjek ne percipira jednako male promjene na višim frekvencijama

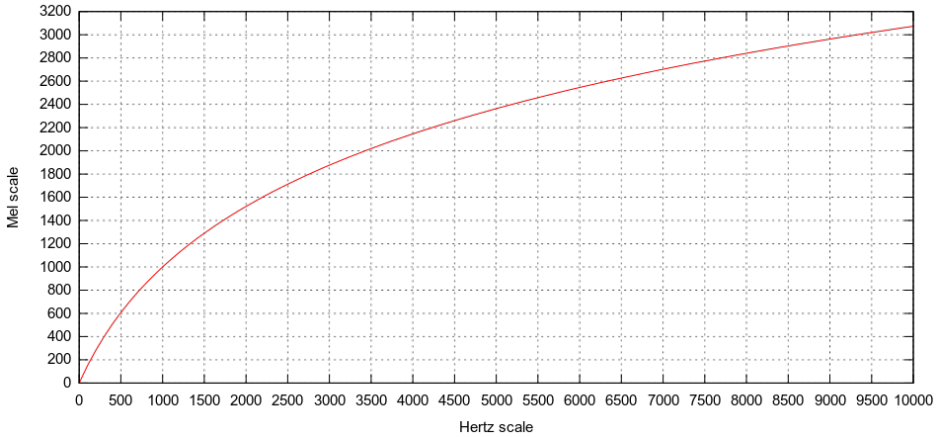
na isti način kao na nižima. Zbog toga je potrebno na neki način skalirati razmake između frekvencija kako bi bolje odgovarali ljudskoj percepciji. Za potrebe toga osmišljena je tzv. Melova skala (engl. *Mel scale*). Preslikavanje na takvu skalu opisano je formulom (2.7) .

$$m(f) = 2595 \cdot \log_{10}\left(1 + \frac{f}{700}\right) \tag{2.7}$$

gdje su:

- $m(f)$: frekvencija na Melovoj skali,
- $f$: frekvencija u Hertzima.

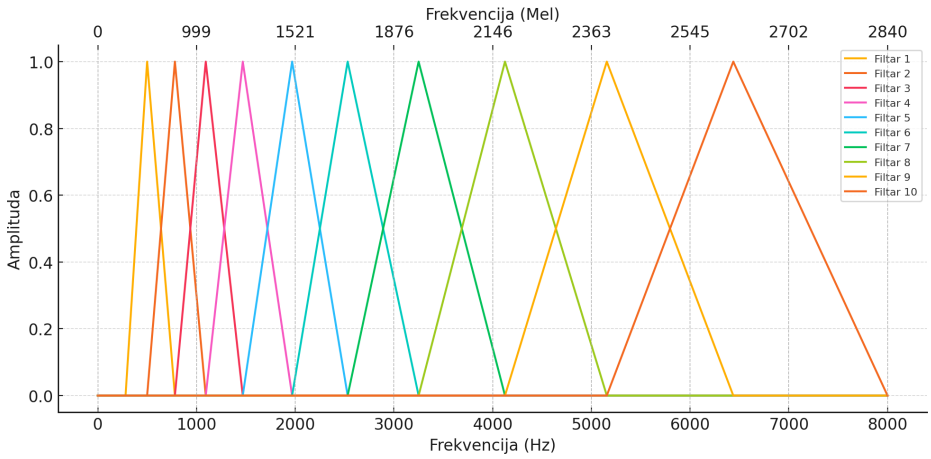
Na slici 2.7. prikazano je preslikavanje iz Hertzove ljestvice u Melovu. Vidljivo je da jednake promjene frekvencije u Hertzima na višim frekvencijama odgovaraju manjoj promjeni na Melovoj skali nego na nižim frekvencijama (logaritamsko preslikavanje), a to izravno odgovara ljudskoj percepciji koja je osjetljivija na promjene u nižim dijelovima spektra.




**Slika 2.7.** Melova skala [13]

Budući da je sustav koji se koristi diskretan, postoji konačan broj frekvencijskih komponenti koje se dobivaju na izlazu iz Fourierove transformacije (`NUMBER_OF_SPECTROGRAM_BINS`) koji izravno ovisi o broju točaka nad kojima se obavlja FFT. Sljedeći korak je preslikavanje iz Hz ljestvice u Melovu. Prije početka preslikavanja, potrebno je odrediti rezoluciju preslikavanja, tj. koliko će biti spektralnih komponenti na novoj Melovoj ljestvici. Taj parametar također je određen u konfiguracijskoj datoteci

`Configuration.hpp`, a zove se `NUMBER_OF_MEL_BINS` (u prijevodu broj Melovih "kanti"). Samo preslikavanje radi se pomoću trokutastih filtara prikazanih na slici 2.8.




**Slika 2.8.** Filtri za Melovu skalu

Za potrebe sustava za prepoznavanje glasa, određuju se donja i gornja granica preslikavanja (`LOWER_BAND_LIMIT` i `UPPER_BAND_LIMIT`) koje su tijekom cijelog razvoja sustava fiksirane na 80 Hz i 7600 Hz. Takve granice su se pokazale prikladne za ovu svrhu i najčešće su korištene u sustavima ovakve vrste. Naime, informacijski najbogatije komponente se obično nalaze između ove dvije granice. Između njih linearno se (na Melovoj skali) raspoređuju sredine filtara čija je širina na svaku stranu točno do sredine susjednog filtra. Izvan tog intervala vrijednost filtra je nula, dok je u središtu pojedinog filtra vrijednost točno jedan. Nadalje, vrijednost svake Melove frekvencijske komponente doprinosi vrijednosti frekvencijske komponente s Hz ljestvice onoliko koliko iznosi njena vrijednost pomnožena s vrijednošću filtra na tom mjestu. Za svaki filtar (koji odgovara jednoj Melovoj frekvencijskoj komponenti) potrebno je provjeriti koliko svaka frekvencijska komponenta s Hz ljestvice doprinosi toj Mel komponenti, tj. koliko energije iz početnog signala odgovara toj komponenti na Melovoj ljestvici. Programski kod koji se bavi opisanim prikazan je u 2.6 . Na kraju cjelokupnog procesa, svaka se Melova frekvencijska komponenta mora logaritmirati jer je to potrebno za sljedeći korak, a to je generiranje MFC koeficijenata.

```
void MelSpectrogram::generate(float *spectrogram, float *melSpectrogram){
    for(int melBin = 0; melBin < NUMBER_OF_MEL_BINS; melBin++){
        melSpectrogram[melBin] = 0.0;
        for(int fftBin=0; fftBin<NUMBER_OF_SPECTROGRAM_BINS; fftBin++){
            float freq = fftBin * hzPerBin;
            float weight = 0.0;

            if(freq >= melPoints[melBin] && freq < melPoints[melBin + 1])
    {
                weight = (freq - melPoints[melBin]) / (melPoints[melBin +
     1] - melPoints[melBin]);
            }else if(freq >= melPoints[melBin + 1] && freq < melPoints[
    melBin + 2]){
                weight = (melPoints[melBin + 2] - freq) / (melPoints[
    melBin + 2] - melPoints[melBin + 1]);
            }
            melSpectrogram[melBin] += spectrogram[fftBin] * weight;
        }
        // Apply log scaling with a safeguard against log(0)
        melSpectrogram[melBin] = logf(fmaxf(melSpectrogram[melBin], 1e-6)
    );
     }
}
```

Kod 2.6: Generiranje Melovog spektrograma

### Diskretna kosinusna transformacija

Posljednji korak u procesu izdvajanja Mel-kepstralnih koeficijenata (MFCC) je primjena diskretne kosinusne transformacije (engl. *Discrete Cosine Transform ili DCT*) na vektor Melovih spektralnih koeficijenata. Ona pretvara signal iz vremenske ili frekvencijske domene u domenu kosinusnih komponenti. Rezultat DCT-a je niz koeficijenata koji predstavljaju energiju signala u različitim frekvencijskim opsezima. Postoje različiti tipovi DCT-a, ali za MFCC se najčešće koristi DCT-II koja je prikazana formulom (2.8) .

$$C(k) = \sqrt{\frac{2}{N}} \sum_{n=0}^{N-1} X(n) \cdot \cos\left[\frac{\pi}{N}\left(n + \frac{1}{2}\right)k\right], \quad k = 0, 1, \ldots, N-1 \tag{2.8}$$

gdje su:

- $C(k)$: koeficijenti DCT-a (u ovom slučaju MFCC),
- $X(n)$: ulazni Mel-frekvencijski spektar,
- $N$: broj uzoraka u ulaznom spektru,
- $k$: indeks koeficijenta (MFCC-a).

Primjenom DCT-a na Melov spektar postiže se:

1. **Dekorelacija podataka**: Komponente Mel-frekvencijskog spektra obično su jako korelirane. DCT uklanja ovu korelaciju, što omogućuje jednostavniju analizu i smanjuje redundantnost.
2. **Redukcija dimenzionalnosti**: DCT generira niz koeficijenata, ali samo prvih nekoliko (obično 12-13) sadrže značajne informacije potrebne za prepoznavanja govora ili zvukova. Ostatak koeficijenata se odbacuje jer sadrže manje bitne informacije ili šum. Također, smanjuju se zahtjevi za memorijom i procesorskom snagom.

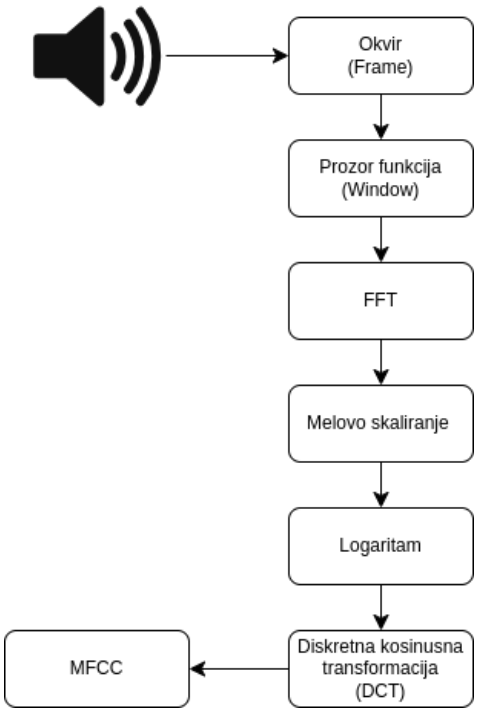
Implementacija diskretne kosinusne transformacije prikazana je u odsječku koda 2.7 . U konstruktoru razreda `DCT` inicijaliziraju se koeficijenti potrebni za izračun pojedinog MFCC-a, dok metoda `void DCT::compute(const float* input, float* output)` računa MFCC-e za konkretni ulazni Mel-frekvencijski spektar. Tako je izračun razdvojen na dva dijela: konstanti dio koji ne ovisi o ulaznom spektru (računa se pri konstrukciji objekta zaduženog za DCT) i dio koji ovisi o ulaznom spektru te ga je zbog toga potrebno računati u svakoj iteraciji. Rezultat DCT algoritma je vektor prvih `NUMBER_OF_MFCCS MFC` koeficijenata, izuzev prvog. Prvi koeficijent se obično odbacuje jer sadrži samo energiju signala, a ne nosi dodatne informacije o spektru signala.

```
DCT::DCT() {
    const float pi = 3.14159265358979323846f;
    for (int i = 0; i < (NUMBER_OF_MFCCS + 1); ++i) {
        for (int j = 0; j < NUMBER_OF_MEL_BINS; ++j) {
            coefficients[i * NUMBER_OF_MEL_BINS + j] = cos(pi * i * (j +
    0.5f) / NUMBER_OF_MEL_BINS);
        }
    }
}
void DCT::compute(const float *input, float *output) {
    float temp[NUMBER_OF_MFCCS + 1];
    for (int i = 0; i < (NUMBER_OF_MFCCS + 1); i++) {
        output[i] = 0.0;
        for (int j = 0; j < NUMBER_OF_MEL_BINS; j++) {
            output[i] += input[j] * coefficients[i * NUMBER_OF_MEL_BINS +
    j];
        }
        output[i] *= sqrt(2.0 / NUMBER_OF_MEL_BINS);
    }
    if(i != 0)  output[i - 1] = temp[i];
}
```

Kod 2.7: Diskretna kosinusna transformacija

Rezultat cijelog procesa izdvajanja značajki prikupljenog zvuka je NUMBER_OF_MFCCS Mel-kepstralnih koeficijenata. Naziv dolazi od toga što su ti koeficijenti temeljeni na Melovoj frekvencijskoj skali koja je opisana u 2.2.2., ali možda nije odmah jasno zašto se zovu i kepstralni. To dolazi od toga što se tijekom njihovog izračuna moralo posegnuti za dvjema transformacijama. Prva je bila FFT koja signal iz vremenske domene transformira u frekvencijsku (spektar signala). Druga transformacija bila je DCT, koja još jednom računala spektar, ali ovog puta je to bio spektar samog spektra. Zbog toga su koeficijenti iz ove domene dobili ime kepstralni (keps je anagram od spek). Ovo se nadalje može povezati s modelom govora opisanim u 2.2.1. Spektar logaritamskog spektra omogućuje razdvajanje energija u pojedinim frekvencijskim spektrima te jako dobro razdvaja odziv vokalnog trakta od kvazi-periodičnog impulsa, kao što je prikazano na slici 2.4. Također, informacija zvučnog prozora duljine WINDOW_SIZE sažeta je u svega NUMBER_OF_MFCCS Mel-kepstralnih koeficijenata. Te veličine mogu varirati, a karakteristični iznosi u susta-

vima ovakvog tipa su redom 512 te 12 ili 13. To je svega oko 2.5% broja početnih značajki (5% ako se promatra memorijsko zauzeće jer `float` tip varijable zauzima 4 bajta, dok `int16_t` zauzima 2 bajta). Blok dijagram cjelokupnog procesa prikazan je na slici 2.9.




**Slika 2.9.** Proces generiranja MFCC-a [2]

### 2.2.3. Konstrukcija matrice značajki

U poglavlju 2.2.2. opisana je konstrukcija MFC koeficijenata iz jednog okvira zvučnog zapisa tipične duljine od oko 20 do 30 ms [5]. Na način koji je opisan u 2.2.2. u sljedećoj iteraciji dohvatit će se određeni broj novih uzoraka koji će sudjelovati u stvaranju novog okvira nad kojim je potrebno generirati nove Mel-kepstralne koeficijente koji odgovaraju tom prozoru na način identičan opisanom. Proces dohvaćanja novih uzoraka te generiranje novih MFCC-a kontinuirano se ponavlja. Rezultat generiranja značajki je jednodimenzionalni vektor koji sadrži `NUMBER_OF_MFCCS` koeficijenata. Tako generirane značajke je potrebno predati podsustavu koji je zadužen za aktivaciju neuronske mreže. Konkretno, sustav za prepoznavanje glasovnih naredbi koristi konvolucijsku neuronsku mrežu čija su struktura i područje primjene podrobnije opisani u 3.2. One dobro funkcioniraju u klasifikaciji višedimenzionalnih matrica. Ideja je primijeniti takvu strukturu na značajke koje su prethodno generirane. Uzastopni vektori MFC koeficijenata slagani su u dvodimenzionalnu matricu koja može biti predana na obradu neuronskoj mreži.

Može se reći da je takvim slaganjem koeficijenata dobivena slika zvučnog zapisa proizvoljne duljine. Koliko će prozora podataka može biti u jednom koraku predano na klasifikaciju određuje duljina podataka nad kojima je mreža i trenirana, što je detaljnije opisano u poglavlju 3.3. Duljina pojedinog zvučnog zapisa u tom skupu iznosi jednu sekundu. Zbog toga je potrebno napraviti matricu od onoliko prozora koliko je potrebno da se pokrije odabrano vrijeme promatranja signala od 1 sekunde, pri čemu će broj takvih prozora `NUMBER_OF_TIME_SLICES` ovisiti o veličini prozora `WINDOW_SIZE`, broju novih uzoraka u svakoj iteraciji `STEP_SIZE` i frekvenciji otipkavanja `SAMPLE_RATE`. Funkcija (2.9) prikazuje opisanu ovisnost.

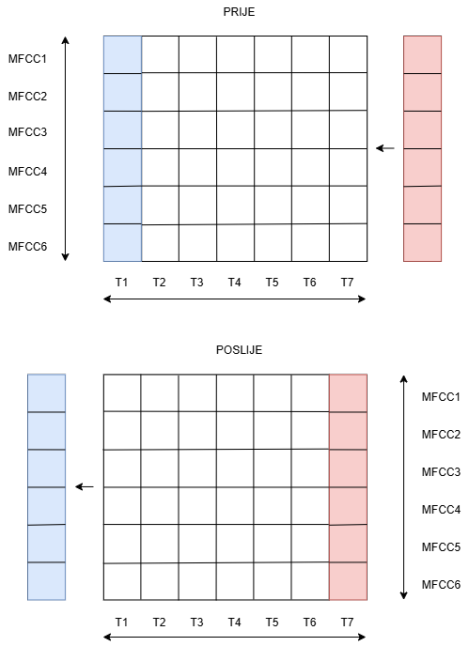
$$\text{NUMBER\_OF\_TIME\_SLICES} = \left\lfloor \frac{\text{SAMPLE\_RATE} - \text{WINDOW\_SIZE}}{\text{STEP\_SIZE}} \right\rfloor + 1 \quad (2.9)$$

Budući da se konstantno akviziraju novi podaci i izdvajaju njihove značajke, a veličina dvodimenzionalne matrice koja se predaje neuronskoj mreži ostaje stalna, potrebno je dolaskom značajki novog vremenskog prozora odbaciti najstariji, a ostale pomaknuti prema početku te na kraj matrice dodati najnovije značajke. Algoritam je sličan onome koji novoprikupljene uzorke dodaje na kraj vremenskog prozora prije izdvajanja značajki, a prikazan je u odsječku programskog koda 2.8 U polje `featureSlice` se pohranjuju novonastale značajke, a polje `featureImage` sadrži trenutačno dvodimenzionalno polje značajki. Ovdje je polje definirano kao jednodimenzionalno, ali je memorijski otisak identičan i takva struktura odgovara ulazu neuronske mreže. Veličina tog polja `NUMBER_OF_FEATURES` je umnožak broja značajki pojedinog prozora `NUMBER_OF_MFCCS` i broja prozora potrebnog za izgradnju matrice `NUMBER_OF_TIME_SLICES`.

```
1    float featureSlice[NUMBER_OF_MFCCS] = {0};
2    float featureImage[NUMBER_OF_FEATURES] = {0};
3
4    memcpy(featureImage, featureImage + NUMBER_OF_MFCCS, sizeof(float) *
    (NUMBER_OF_FEATURES - NUMBER_OF_MFCCS));
5    memcpy(featureImage + NUMBER_OF_FEATURES - NUMBER_OF_MFCCS,
    featureSlice, sizeof(float) * NUMBER_OF_MFCCS);
```

Kod 2.8: Generiranje matrice značajki

Algoritam osvježavanja postojeće matrice značajki prikazan je slikom 2.10. Pokazna matrica se sastoji od sedam vremenskih odsječaka i šest MFC koeficijenata (broj značajki se u stvarnoj implementaciji razlikuje od pokaznog). Crveni vremenski prozor značajki se dodaje je na kraj matrice, dok se istovremeno ostali pomiču prema naprijed te istiskuju najstariji prozor koji je u ovom slučaju prikazan plavom bojom.




**Slika 2.10.** Matrica značajki [2]

## 2.3. Aktivacija neuronske mreže

Aktivacija neuronske mreže proces je u kojem se ulaznom sloju neuronske mreže (engl. *input layer*) predaje matrica značajki generirana na način objašnjen u prethodnom poglavlju. Biblioteka koja je korištena za implementaciju neuronske mreže na mikrokontroleru je TensorFlow Lite [14]. Omogućuje vrlo jednostavno korištenje treniranog modela (treniranog na računalu kako je opisano u poglavlju o treniranju neuronske mreže 3.6. te konvertiranog u polje koje se može koristiti na mikrokontroleru kako je opisano u dijelu o prilagodbi 3.8.). Trenirani model je polje 8-bitnih vrijednosti koje predstavljaju pojedine parametre mreže dobivene upravo treniranjem modela. Razred koji omotava korištenje biblioteke i modela prikazan je u isječku koda 2.9

```
class NeuralNetwork {
    private:
        const tflite::Model* model = nullptr;
        tflite::MicroMutableOpResolver<NUMBER_OF_OPERATORS> resolver;
        float* model_input_buffer = nullptr;
    public:
        NeuralNetwork();
        void giveFeaturesToModel(float* features, size_t numberOfFeatures
    );
        bool invoke(void);
        int numberOfClasses;
        float* outputData;
};
```

Kod 2.9: Razred neuronske mreže

Članska varijabla `const tflite::Model* model` predstavlja pokazivač na polje parametara modela (koji ujedno sadrži i informacije o samoj strukturi mreže), dok `tflite::MicroMutableOpResolver<NUMBER_OF_OPERATORS> resolver` predstavlja arhitekturu neuronske mreže koja koristi informacije iz polja parametara modela kako bi pravilno procesirala dani ulaz. Toj strukturi je potrebno dodati sve vrste slojeva i funkcija koje se koriste unutar arhitekture mreže koja je trenirana. Primjer dodavanja nekih vrsta slojeva prikazan je u isječku koda 2.10 Nije bitan redoslijed dodavanja, samo je potrebno dodati sve što ta mreža sadrži. U suprotnom, inicijalizacija modela neće uspjeti.

```
resolver.AddConv2D()            // konvolucijski sloj
resolver.AddFullyConnected()    // potpuno povezani sloj
resolver.AddSoftmax()           // softmax funkcija
resolver.AddMaxPool2D()         // sloj za poduzorkovanje
```

Kod 2.10: Gradnja arhitekture mreže

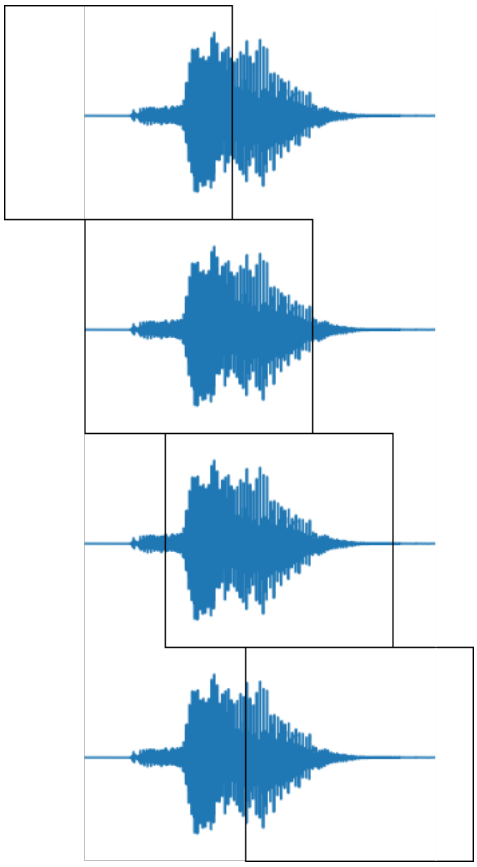
Nakon što je mreža alocirana na pravilan način (struktura polja u kojem se nalaze parametri mreže odgovara strukturi mreže na mikrokontroleru) moguće joj je predati matricu značajki te ju aktivirati. Nakon aktivacije, mreža obrađuje vrijednosti kroz svoju strukturu te na izlazu daje vjerojatnosti da određeni ulaz (matrica značajki) predstavlja naučenu naredbu. Kako je objašnjeno u 3.4., broj kategorija za koje trenirana mreža daje na izlazu vjerojatnosti je za dva veći od broja naredbi koje mreža može prepoznati.

Dodatne kategorije su *"pozadina"* (engl. *background*) i *"nepoznato"* (engl. *unknown*).

Aktivacija neuronske mreže i dobivanje vrijednosti na njenom izlazu, koje predstavljaju vjerojatnosti da ulazna matrica značajki pripada određenoj kategoriji, samo je jedna iteracija u radu sustava čija je struktura prikazana slikom 2.1. Svakom novom iteracijom rada sustava prikupljaju se novi zvučni uzorci, matrica značajki se osvježava, tj. noviji MFC koeficijenti dolaze u matricu. Novi podaci u matrici predstavljaju značajke signala pomaknutog za dohvaćeni broj uzoraka (tipično 20-30 milisekundi) te je potrebno provjeriti kojoj kategoriji pripada taj vremenski prozor. Budući da je od aktivacije mreže do izlaska rezultata u krajnjem sloju mreže potrebno određeno vrijeme, nije moguće aktivirati mrežu na svaki novi vremenski prozor podataka jer će biti zagušeno prikupljanje novih podataka te cjelokupni sustav neće dobro raditi u stvarnom vremenu. Međutim, dobra strana postupka je da nije potrebno aktivirati mrežu na svaki novi vremenski korak jer je svojom duljinom puno manji od duljine uzorka kojeg predstavlja cijela matrica. Drugim riječima, uzastopne verzije ulazne matrice podataka bit će vrlo slične i može se pričekati nekoliko novih iteracija prije nego li se aktivira neuronska mreža (akviziranje signala i izdvajanje značajki tipično traje puno kraće od aktivacije). S druge strane, mreža se mora aktivirati dovoljno često jer u suprotnom postoji mogućnost propuštanja određenih naredbi. Parametar koji brine o tome koliko često se mreža aktivira zove se `NUMBER_OF_NEW_SLICES_BEFORE_INVOKING` i određen je eksperimentalno zbog toga što drugačije strukture mreže imaju drugačija vremena odziva. Potrebno je maksimizirati broj aktiviranja mreže u određenom vremenu bez gubitaka uzoraka ulaznog signala.

Na slici 2.11. prikazani su uzastopni vremenski prozori nad ulaznim zvučnim signalom. Na signalu je vidljiv vremenski odsječak u kojem je izgovorena naredba, a prije i poslije izgovorene naredbe je tišina. U drugom i trećem prozoru očekuje se velika vjerojatnost da je prepoznata izgovorena naredba, dok se u prvom i četvrtom prozoru ne očekuje prepoznavanje određene naredbe (u tim prozorima veću vjerojatnost imaju kategorije *""nepoznato"* ili *""pozadina"*). Upravo zbog prikazanog je potrebno što češće aktivirati mrežu i ne propustiti prepoznavanje naredbe. U slučaju da se mreža aktivira rjeđe, ne bi se prepoznala naredbu (npr. prvi pa četvrti prozor).

No, takav pristup otvara mjesto drugom problemu. Nije poželjno niti višestruko prepoznavanje jednom izgovorene naredbe pa je stoga potreban podsustav za prepoznavanje



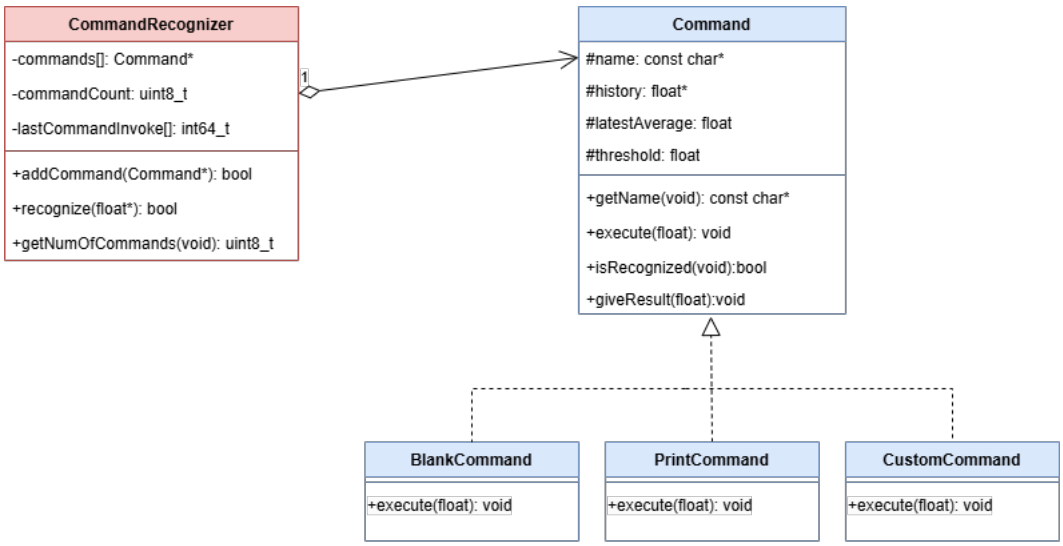
**Slika 2.11.** Uzastopni vremenski okviri (prozori) [2]

i okidanje zadatka povezanog s određenom naredbom.

## 2.4. Prepoznavanje naredbi i aktivacija zadatka

Ako se mreža aktivira dovoljno često, za očekivati je da će najvjerojatniji izlaz iz neuronske mreže u slučaju izgovorene naredbe biti upravo kategorija koja predstavlja tu naredbu. Štoviše, naredba bi trebala biti prepoznata u više uzastopnih iteracija aktiviranja mreže. Međutim, ponašanje sustava u tom slučaju neće biti onakvo kakvo bi trebalo biti, a to je da se određeni posao (zadatak) aktivira isključivo jednom za jednom izgovorenu naredbu. Dodatno treba pripaziti na slučajno aktiviranje nekog posla jer zbog nesavršenosti mreže se može dogoditi da vjerojatnosni izlaz ukazuje na prepoznavanje neke naredbe, iako se ona u stvarnosti nije izgovorila. Međutim, u takvim situacijama se očekuje možda jedna ili dvije takve situacije zaredom, ali ne i više njih. Zbog svega navedenog, potrebno je na neki način obraditi tok vjerojatnosnih izlaza iz neuronske mreže. Kao najbolji način za pokrivanje svih problema pokazalo se uprosječivanje vjerojatnosti pojedinih kategorija.

Na slici 2.12. prikazan je UML dijagram implementiranog podsustava. Upravljački dio posla obavlja se unutar razreda `CommandRecognizer` koji je zadužen za aktiviranje obavljanja konkretnog posla koji je povezan s govornom naredbom. Sadrži listu pokazivača na objekte čiji razredi implementiraju sučelje `Command`. `Command` predstavlja sučelje (apstraktni razred) što znači da nije moguće konstruirati objekt tog razreda, nego da je potrebno naslijediti razredom koji implementira apstraktnu metodu `virtual void execute(float probability)`. Ta metoda je zadužena za obavljanje konkretnog zadatka. Ovakav strukturni obrazac daje mogućnost vrlo lakog implementiranja novih vrsta zadataka (sve što je potrebno je napraviti novi razred koji implementira sučelje `Command`, tj. nadjačava spomenutu metodu). Trenutačno su implementirane dvije vrste naredbi (zadataka): `BlankCommand` koja ne radi ništa (koristi se za kategorije *"pozadina"* i *"nepoznato"*) i `PrintCommand` koja ispisuje ime naredbe i prosječnu vjerojatnost pojavljivanja naredbe.




**Slika 2.12.** Podsustav za prepoznavanje naredbi i aktivaciju zadataka[2]

Prilikom konstrukcije objekta razreda koji implementira sučelje `Command` potrebno je postaviti parametre za prepoznavanje naredbe koji se utvrđuju eksperimentalno. Parametrom `historySize` bira se broj uzastopnih vrijednosti vjerojatnosti pojave naredbe nad kojima se računa prosjek, a parametar `threshold` postavlja prag koji prosječna vrijednost mora prijeći kako bi se naredba aktivirala. Ovim se postiže odvojeno kalibriranje parametara za svaku naredbu koje je potrebno zbog podataka nad kojima se mreža uči. Zbog nejednakosti kvalitete podataka može se dogoditi da se određene naredbe prepoznaju lakše ili teže od drugih. Uz to, objekt razreda `CommandRecognizer` brine

o tome da se naredba aktivira samo ako je prošlo određeno vrijeme nakon zadnje aktivacije. To se postiže parametrom `COOL_DOWN_PERIOD_MS` koji je definiran u datoteci `Configuration.hpp` B Jednom konstruirani objekt naredbe potrebno je dodati u objekt klase `CommandRecognizer` metodom `bool addCommand(Command* command)` kako bi znao za postojanje te naredbe (broj izlaza iz neuronske mreže mora odgovarati broju dodanih naredbi ovim putem). Svaka konkretna implementacija metode `virtual void execute(float probability)` ne smije trajati predugo jer će unijeti kašnjenje u sustav. Ideja je da takva naredba samo pokrene duži posao za koji će onda biti zadužen neki drugi podsustav.

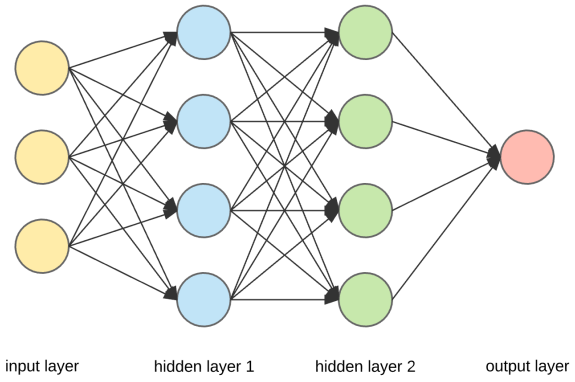
Ovime je opisana cjelokupna petlja mikrokontrolerskog sustava koja se neprestano iznova izvršava (slika 2.1.). Jedini element u ovom opisu koji nije detaljno opisan je sama neuronska mreža koja je u biti okosnica cijelog sustava za prepoznavanje glasovnih naredbi. U sljedećem poglavlju će stoga biti detaljno opisana sama neuronska mreža na kojoj se temelji sustav.

# 3. Neuronska mreža za prepoznavanje govornih naredbi

## 3.1. Načela rada neuronskih mreža

Neuronska mreža ili, preciznije, umjetna neuronska mreža (engl. *neural network*) je računalni model inspiriran biološkom strukturom neurona u mozgu. Predstavlja jedan od najkorištenijih modela u dubokom učenju. Sastoji se od čvorova (neurona) i jednosmjernih veza između njih (sinapsa) koji tako tvore usmjereni graf. Čvorovi su grupirani u slojeve, a svaki od njih je povezan s čvorovima iz susjednog sloja na određeni način. Način na koji su određeni slojevi međusobno povezani određuje vrstu sloja.

Jednostavan primjer strukture neuronske mreže je mreža izgrađena od potpuno povezanih slojeva (engl. *fully connected layer ili dense layer*). Oni se često koristi kao osnovni građevni blok u umjetnim neuronskim mrežama [15]. U potpuno povezanom sloju svaki čvor jednog sloja povezan je sa svakim čvorom susjednog sloja. Ovakva struktura omogućava mreži fleksibilno učenje složenih odnosa između ulaznih i izlaznih podataka.




**Slika 3.1.** Potpuno povezani slojevi [15]

Na slici 3.1. prikazana je struktura neuronske mreže koja se sastoji od ulaznog sloja,

dva potpuno povezana sloja te izlaznog sloja. Srednji slojevi (svi osim ulaznog i izlaznog) se još nazivaju i skriveni slojevi jer kada se koristi model neuronske mreže, obično se na njega gleda kao na crnu kutiju koja na ulazu prima vrijednosti te na izlazu daje vrijednosti izračunate kroz sve skrivene slojeve [16].

Svaka veza između pojedinih čvorova ima određenu vrijednost koju nazivamo težina, a svaki čvor zapravo predstavlja funkciju koja može aktivirati svoj izlaz i vezu sa sljedećim čvorom.

$$a = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{3.1}$$

Jednadžba (3.1) modelira ponašanje pojedinog čvora u mreži. Aktivacijska funkcija $f$ je vrlo bitna u odvajanju bitnih od nebitnih utjecaja pojedinih čvorova na sljedeći čvor. Također, ona omogućava modeliranje složenijih nelinearnih odnosa [17]. Kada bi čvor bio modeliran bez aktivacijske funkcije, svako preslikavanje koje bi činio bi bilo linearno, a zbog toga što svaka kompozicija linearnih funkcija daje opet linearnu funkciju, cjelokupna mreža ne bi bila sposobna modelirati kompleksnije odnose. Sastavnice modela čvora su sljedeće:

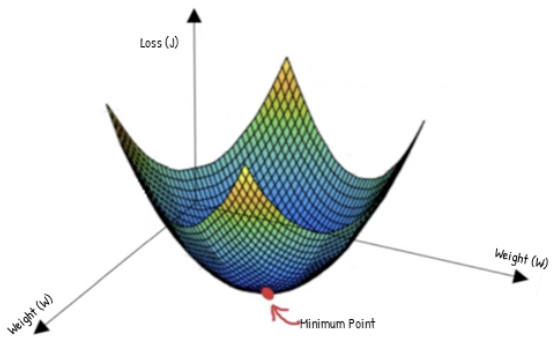
- $a$: izlazna vrijednost čvora (aktivacija)
- $f$: aktivacijska funkcija
- $w_i$: težina i-tog ulaznog čvora (čvor u prijašnjem sloju)
- $x_i$: vrijednost i-tog ulaznog čvora (njegova aktivacija)
- $b$: pomak
- $n$: broj čvorova u prijašnjem sloju koji imaju vezu s modeliranim čvorom

Povezivanjem više ovako definiranih čvorova gradi se neuronska mreža. Ulaz u neuronsku mrežu je informacija na temelju koje će na izlazu iz mreže biti vrijednost izračunata s pomoću svih slojeva u mreži. Kako bi vrijednosti na izlazu iz mreže imale smisla, tj. davale korisnu informaciju, potrebno je trenirati mrežu. Treniranje mreže, u općem slučaju nadziranog strojnog učenja, podrazumijeva korištenje označenog skupa

podataka koji je istog oblika kao i podaci koji će biti na ulazu u mrežu tijekom korištenja same mreže. Arhitektura mreže (vrsta, veličina i broj slojeva) određena je prije samog treniranja, dok se težine, pomaci te samim time i razine aktivacija uče, tj. treniraju. Treniranje je proces u kojem neuronska mreža na svoj ulaz dobiva označeni skup podataka (označeni skup predstavlja podatke za koje znamo što bi mreža trebala dati na izlazu) te provjerava koliko izlazi odstupaju od prave oznake. Na početku su sve težine uglavnom inicijalizirane na nulu. Podatak se predaje ulaznom sloju, prolazi kroz sve skrivene slojeve te na izlazu mreža izbacuje određenu vrijednost koja se s očekivanom uspoređuje s pomoću funkcije gubitka. Takva funkcija predstavlja koliko izlazi iz mreže odstupaju od očekivanih.

Cilj svakog treniranja jest smanjiti vrijednost funkcije gubitka. Stoga se sve težine u mreži ažuriraju tako da njihove promjene pomaknu trenutačno stanje mreže u smjeru negativne derivacije funkcije gubitka (gradijentni spust). Takvim pristupom, mreža kroz iteracije s novim podacima smanjuje funkciju gubitka (efektivno daje sve točnije predikcije). Težine se ažuriraju od izlaznog sloja prema ulaznom (engl. *backpropagation*) jer na izlaz pojedinog sloja utječu njegovi ulazi, tj. izlazi prijašnjeg sloja. Naime, izlaz svakog sloja je funkcija izlaza prijašnjeg sloja, a kad se izračuna derivacija funkcije, promjenom njenih ulaza poznato je u kojem smjeru će se mijenjati vrijednost same funkcije.

Bez smanjenja općenitosti, funkcija gubitka prikazana je na trodimenzionalnom grafu na slici 3.2. Složene arhitekture mreža će imati više dimenzija zbog većeg broja težina $w_i$. Cilj svakog treniranja mreže je doći što bliže minimumu ovakve funkcije. Svakom iteracijom mreža se pomiče sve bliže minimumu, a takva vrsta optimizacije naziva se gradijentni spust (engl. *gradient descent*).




**Slika 3.2.** Funkcija gubitka [18]

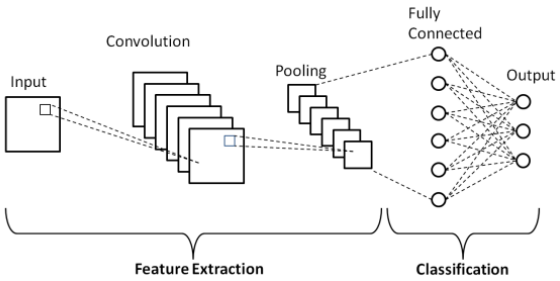
## 3.2. Konvolucijska neuronska mreža

Konvolucijska neuronska mreža (engl. *Convolutional neural network ili CNN*) je vrsta umjetne neuronske mreže koja je pogodna je za obradu podataka s rešetkastom (matričnom) topologijom, a najviše se koristi za rješavanje problema iz područja klasifikacije slika te računalnog vida. Inspirirane su načinom funkcioniranja moždanog korteksa zaduženog za vid kod sisavaca [19]. Obrada slike koju vide sisavci funkcionira hijerarhijski, tj. u mozgu se ne obrađuje cjelokupna slika odjednom, nego postoje jednostavnije stanice koje su zadužene za prepoznavanje osnovnijih oblika koji se nakon toga stapaju u složenije. Naposljetku organizam može prepoznati cjelokupnu sliku koju gleda očima. Veza između prepoznavanja slika i problema klasifikacije glasovnih naredbi možda nije vidljiva na prvu. Međutim, generirana matrica značajki iz poglavlja 2.2. predstavlja upravo dvodimenzionalnu sliku koja se može koristiti kao ulaz u konvolucijsku neuronsku mrežu.

Računalni modeli koji koriste strukturu sličnu opisanoj mogu iz podataka koji su u takvom obliku izvući značajke samostalno što znači da nema potrebe za korištenjem metoda koje eksplicitno izvlače bitne značajke iz matričnih podataka [20]. Arhitektura najjednostavnije konvolucijske neuronske mreže uključuje:

- **Ulaz**: ulazni sloj modela, prima matrični podatak
- **Konvolucijski sloj**: osnovni sloj modela. Njegov glavni zadatak je ekstrakcija značajki iz ulaznih podataka. Vidi 3.2.1.
- **Sloj za poduzorkovanje**: smanjuje dimenzionalnost (vidi 3.2.2.)
- **Sloj za poravnavanje**: spaja konvolucijski i potpuno povezani dio (vidi 3.2.3.)
- **Potpuno povezani sloj**: povezuje značajke s klasifikacijom (vidi 3.2.4.)
- **Izlaz**: izlazni sloj, daje vjerojatnosti klasifikacije (vidi 3.2.5.)

Na slici 3.3. prikazana je opisana struktura. Prva tri sloja grupirana su u dio koji služi za ekstrakciju značajki iz ulaznih podataka, a posljednja dva sloja služe za klasifikaciju. Složenije arhitekture mreže mogu imati veći broj konvolucijskih slojeva (nakon svakog se nalazi sloj za poduzorkovanje) te veći broj složenijih ili manje složenih potpuno po-




**Slika 3.3.** Jednostavna CNN [20]

vezanih slojeva.

### 3.2.1. Konvolucijski sloj

Najbitniji dio ovakvog tipa neuronske mreže je konvolucijski sloj (engl. *convolutional layer*) zbog toga što se u njemu događa konvolucija. Konvolucija (u neuronskim mrežama) je proces kojim se iz ulazne matrice podataka (slike) na izlazu dobije matrica značajki ili mapa značajki. Neka je ulazna matrica oblika $x \in M_{mn}(\mathbb{R})$. Umjesto težina (kao kod potpuno povezanog sloja), konvolucijski sloj koristi matricu $\omega \in M_{pr}(\mathbb{R})$ koju nazivamo filtar ili jezgra (engl. *kernel*) [21]. Svaki konvolucijski sloj može imati proizvoljan broj filtara. Izlazna (u ovom slučaju dvodimenzionalna) mapa značajki tada se računa na sljedeći način:

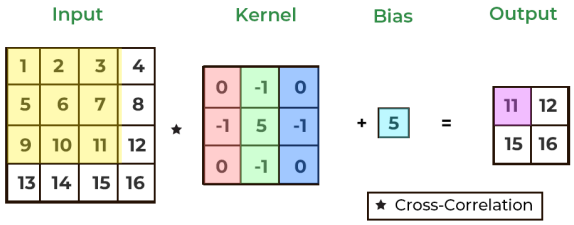
$$h = \omega * x, \tag{3.2}$$

pri čemu $*$ označava operaciju konvolucije te vrijedi:

$$h(i,j) = (\omega * x)(i,j) = \sum_{k=0}^{p-1} \sum_{l=0}^{r-1} x(i+k, j+l)\omega(k,l). \tag{3.3}$$

Formula koja se zapravo koristi naziva se unakrsna korelacija, međutim, zbog sličnosti s formulom konvolucije, mreža nosi takav naziv [22]. Na slici 3.4. prikazan je proces koji se događa tijekom prolaska ulaznog podatka kroz konvolucijski sloj. Ulaz (engl. *input*) se konvolucijski množi s jezgrom kako bi se dobila izlazna matrica značajki [23]. U slučaju prikazanom na slici, ulazna slika je veličine $4\times4$, dok je jezgra veličine $3\times3$. Prvo se podmatrica ulaznog podatka veličine jednake veličini jezgre ($3 \times 3$) skalarno množi s

jezgrom. Izlaz je skalarni umnožak na koji se može dodati konstantna vrijednost (engl. *bias*). Nakon toga se jezgra pomiče po ulaznoj matrici, tj. sljedeći element izlazne matrice je skalarni umnožak jezgre i sljedeće podmatrice ulaznog podatka. Koliko će se jezgra pomaknuti određuje pomak (engl. *stride*). U slučaju na slici 3.4. pomak iznosi jedan.




**Slika 3.4.** Konvolucija [24]

Rezultat opisanog procesa je mapa značajki koja je manja od ulazne, a njena veličina obrnuto proporcionalno ovisi o veličini jezgre te pomaku [25].

Slično procesu koji se odvija u mozgu čovjeka, opisana struktura omogućava hijerarhijsko učenje. Naime, svaki konvolucijski sloj sadrži jezgre koje su zadužene za lokalno pretraživanje određenih uzoraka. Upravo konvolucijom izvlačimo stvari koje su slične između različitih ulaznih slika. Koje jezgre se trebaju koristiti određuje proces učenja (treniranja mreže) koji prepoznaje lokalne sličnosti između različitih primjera. Ako se mreža sastoji od više uzastopnih konvolucijskih slojeva, prvo će naučiti najjednostavnije oblike, a zatim u sljedećem sloju takvim oblicima slagati složenije uzorke. Također, još jedna prednost konvolucijskog sloja je u dijeljenju parametara. Takav sloj nema vezu svakog neurona sa svakim ulaznim, nego se težine dijele unutar određene jezgre. Zapravo se cijeli proces učenja svodi na nalaženje odgovarajućih jezgri koje će prepoznati uzorke [19]. U slučaju dvodimenzionalne ulazne slike, broj parametara ovakvog sloja iznosit će:

$$N = (n \cdot m \cdot C_{\text{in}} + 1) \cdot C_{\text{out}} \tag{3.4}$$

gdje su:

- $N$: ukupni broj parametara,
- $n$ i $m$: visina i širina filtra (jezgre),

- $C_{\text{in}}$: broj ulaznih kanala (npr. 1 za crno-bijele slike, 3 za RGB slike),
- $+1$: konstanta svakog filtra,
- $C_{\text{out}}$: broj filtara (odnosno mapa izlaznih značajki).

Kako bi se bolje dočarala razlika u broju parametara između ovakvog sloja i potpuno povezanog sloja, kao primjer može se uzeti slika 3.4. Ulazna slika je veličine $4 \times 4$, jezgra $3 \times 3$, a izlaz $2 \times 2$. Neka je slika jednokanalna, a broj jezgri jedan (sve kao na slici). Konvolucijski sloj će u tom slučaju imati 10 parametara, dok će potpuno povezani sloj (16 ulaznih vrijednosti, 4 izlazne te 4 konstante za svaki neuron) imati 68 parametara! Formula za broj parametara u tom slučaju je sljedeća:

$$N = n_{\text{in}} \cdot n_{\text{out}} + n_{\text{out}} \tag{3.5}$$

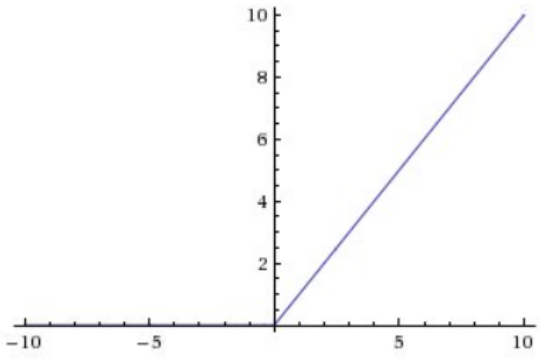
gdje su:

- $N$: ukupni broj parametara,
- $n_{\text{in}}$: broj ulaznih neurona,
- $n_{\text{out}}$: broj izlaznih neurona.

Također, kao što je opisano u poglavlju 3. vrijednost neurona (čvora) na izlazu iz sloja potrebno je provući kroz aktivacijsku funkciju. Postoje različite vrste funkcija koje se koriste u različitim granama strojnog i dubokog učenja [17], a za primjenu u konvolucijskim slojevima, najefektivnija se pokazala ReLu (engl. *Rectified linear units*) [26]. To je funkcija koja vraća nulu ako joj je ulaz negativan, a za svaki pozitivan ulaz samo prosljeđuje istu vrijednost na izlaz. Modelirana je formulama 3.6 i 3.7, a prikazana je na slici 3.5.

$$f(x) = \max(0, x) \tag{3.6}$$

$$f(x) = \begin{cases} 0, & \text{ako } x < 0, \\ x, & \text{ako } x \geq 0. \end{cases} \tag{3.7}$$



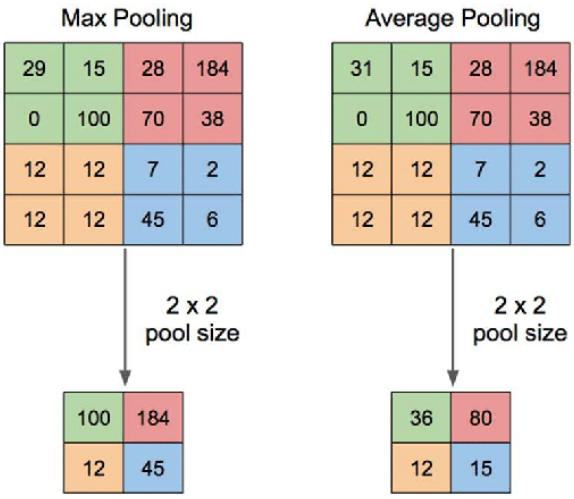

**Slika 3.5.** ReLu [26]

Prednosti ReLu funkcije nad drugim aktivacijskim funkcijama je u tome što za negativne ulaze uopće ne aktivira neuron što izuzetno povećava efikasnost. Druga prednost je u tome što izlaz linearno raste s porastom ulazne vrijednosti što znači da nikad neće ući u zasićenje. Ta osobina je važna jer utječe na izgled funkcije gubitka te ubrzava konvergenciju gradijentnog spusta prema minimumu funkcije gubitka [17].

### 3.2.2. Sloj za poduzorkovanje

Sloj za poduzorkovanje (engl. *pooling layer*) služi smanjenju dimenzionalnosti matrice značajki na izlazu iz konvolucijskog sloja. Najčešće korištene tehnike su maksimalno (engl. *max pooling*) i prosječno poduzorkovanje (engl. *average pooling*). Radi tako da više susjednih vrijednosti spoji u jednu te tako na svom izlazu da matricu manjih dimenzija [27]. Na taj način postupno smanjuje broj parametara, smanjuje broj operacija potrebnih za daljnje računanje te ono najbitnije, kontrolira prenaučenost [25]. Na slici 3.6. prikazane su obje navedene vrste poduzorkovanja.

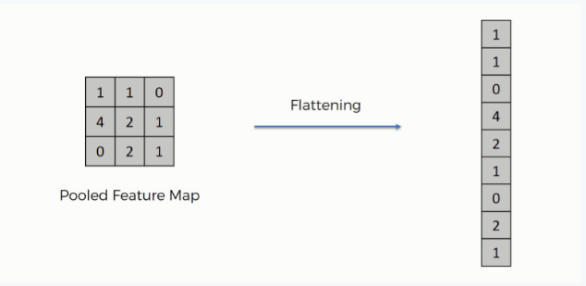
Prosječno poduzorkovanje izglađuje sliku (engl. *smoothing*). Zbog toga oštri detalji slike mogu biti izgubljeni što znači da se određene značajke možda neće prepoznati kada se koristi ova metoda. Maksimalno uzorkovanje odabire piksele s najvećom vrijednošću iz slike, a oni su se pokazali kao najbitnije značajke jer daju najbolje rezultate [25]. Suprotno tome, minimalno uzorkovanje (engl. *min pooling*) odabralo bi piksele s najmanjom vrijednošću, međutim ono se najrjeđe koristi.




**Slika 3.6.** Poduzorkovanje [27]

### 3.2.3. Sloj za poravnavanje

Sloj za poravnavanje (engl. *flatten layer*) je sloj koji dolazi nakon posljednjeg sloja za poduzorkovanje. Njegova jedina zadaća je poravnati izlaz iz prijašnjeg sloja. Ovaj sloj ništa ne računa, niti išta uči, a jedina mu je zadaća od ulaznih mapa značajki napraviti jedan vektor koji je onda moguće povezati na potpuno povezani sloj. Zbog toga se često izostavlja iz skica koje prikazuju strukture CNN-ova, kao što je slučaj na slici 3.3. Na slici 3.7. prikazana je uloga ovog sloja.




**Slika 3.7.** Sloj za poravnjavanje [28]

### 3.2.4. Potpuno povezani sloj

Potpuno povezani sloj (engl. *fully connected layer*) detaljnije je pojašnjen u poglavlju 3. Nakon prijašnjih slojeva koji su služili za izvlačenje značajki iz ulaznih podataka, na red dolazi klasifikacija. Budući da je prethodnik prvom ovakvom sloju sloj za poravnavanje, ne postoji problem sa spajanjem ovog sloja na dosad objašnjenu strukturu. Uloga ovog sloja (ili više ovakvih slojeva) je, najjednostavnije rečeno, klasifikacija. Značajke naučene

tijekom konvolucije se ovdje predaju gustoj mreži neurona koja je sposobna odraditi posao do kraja, tj. naučiti kako različite značajke pridonose određenoj izlaznoj klasi.

### 3.2.5. Izlazni sloj

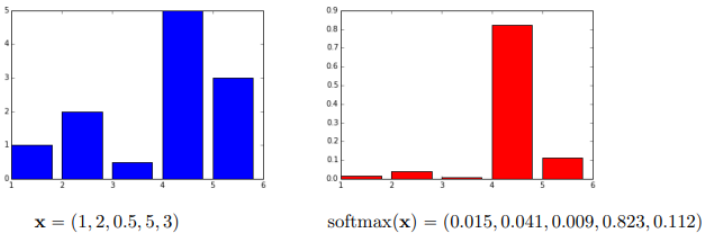
Izlazni (engl. *output layer*) je posljednji sloj potpuno povezanog sloja, a time ujedno i cijele neuronske mreže. Ima onoliko neurona koliko se želi imati klasa, a pojedina vrijednost neurona predstavlja vjerojatnost pripadnosti određenoj klasi. Da bi to stvarno funkcioniralo na takav način, potrebno je odrediti prikladnu aktivacijsku funkciju. Funkcija koja to radi naziva se funkcija *softmax*. Formalno, softmax : $\mathbb{R}^n \to \mathbb{R}^n$, gdje je $k$-ta komponenta izlaznog vektora definirana kao:

$$\text{softmax}_k(x_1, \ldots, x_n) = \frac{\exp(x_k)}{\sum_j \exp(x_j)} \tag{3.8}$$

Funkcija *softmax* radi dvije ključne stvari:

- normalizira sve vrijednosti tako da njihov zbroj bude 1, tj. izlazni vektor predstavlja distribuciju vjerojatnosti,
- pojačava veće vrijednosti (čini ih dominantnijima) i smanjuje manje vrijednosti.

Funkcija nosi naziv *softmax* jer odgovara funkciji max, ali je "meka" u smislu da je neprekidna i diferencijabilna, za razliku od klasične max funkcije [29].




**Slika 3.8.** Softmax [29]

### 3.2.6. Regularizacija odbacivanjem težina

Regularizacija je postupak kojim se smanjuje složenost modela kako bi se spriječila prenaučenost (engl. *overfitting*). Prenaučenost se događa kada model ima preveliku sposobnost prilagodbe podacima za treniranje, ali ne i za podacima za testiranje. U neuronskim

mrežama je, uz sloj za poduzorkovanje, implementirana u obliku sloja koji tijekom treniranja slučajnim odabirom postavlja određeni postotak težina u modelu na nulu (engl. *dropout layer*).

### 3.2.7. Poznate arhitekture konvolucijskih mreža

Arhitektura konvolucijske mreže je ključni faktor koji određuje njene performanse i učinkovitost. Broj konvolucijskih slojeva, izgled istih (broj filtara, njihova veličina, pomak), vrsta slojeva za poduzorkovanje te broj i veličina potpuno povezanih slojeva znatno utječu na brzinu izvođenja i preciznost klasifikacije. Naravno, ne postoji jedan recept koji najbolje radi na svim vrstama ulaznih podataka, nego različite arhitekture daju bolje rezultate u određenim situacijama. Dobro je istaknuti određene arhitekture koje imaju povijesno značenje zbog toga kako su utjecale na područje istraživanja u dubokom učenju[30]:

- **LeNet-5 (1998):** CNN sa 7 slojeva dizajnirana za klasifikaciju rukom pisanih brojeva na slikama veličine 32×32 piksela u sivim tonovima. Koristila se u bankama za čitanje čekova i bila je prvi značajan korak u korištenju CNN-a u stvarnom svijetu.
- **AlexNet (2012):** Proširena verzija LeNet-a s dubljom arhitekturom (5 konvolucijskih i 3 potpuno povezana sloja). Prva mreža koja je koristila ReLU aktivaciju za brže treniranje. Značajno smanjila stopu pogreške na ILSVRC natjecanju i popularizirala duboko učenje.
- **GoogleNet (Inception V1) (2014):** 22-slojna mreža s inovativnim *inception module*, koji koristi male konvolucije za smanjenje broja parametara (sa 60 milijuna na samo 4 milijuna). Pobjednik ILSVRC 2014 s top-5 pogreškom manjom od 7%. Performanse su usporedive s ljudskim prepoznavanjem slika.
- **VGGNet (2014):** Mreža sa 16 konvolucijskih slojeva koja koristi samo $3 \times 3$ konvolucije s povećanim brojem filtara. Iako je jednostavna u dizajnu, ima 138 milijuna parametara, što je čini računalno zahtjevnom za treniranje i implementaciju.

## 3.3. Skup podataka za treniranje

Prvi korak u izgradnji sustava koji koristi bilo kakav tip modela strojnog učenja je odabir i priprema prikladnog skupa podataka na kojem će se taj model trenirati. Budući da je cilj sustava prepoznavanje govornih naredbi (engl. *keyword spotting*), prikladan otvoreni skup podataka za tu svrhu je skup za treniranje prepoznavanja ograničenog skupa govornih naredbi [31]. Riječ je o skupu koji se sastoji od oko 105000 zvučnih isječaka duljine oko jedne sekunde (frekvencija zapisa je 16 kHz) u kojima ljudi izgovaraju jednu od 35 različitih riječi. Također, skup ima nekoliko vrsta pozadinske buke koja je ključna za rad ovakvog sustava u stvarnim uvjetima. U prikupljanju podataka sudjelovalo je oko 2600 ljudi iz cijelog svijeta. U privitku je prikazana tablica koja prikazuje od kojih se riječi skup sastoji te koliko snimaka pojedinih riječi postoji A1.

## 3.4. Priprema podataka

Nakon odabira skupa na kojem će model biti treniran, potrebno je pripremiti podatke. Zbog ograničenih resursa na mikrokontrolerskom sustavu, nije moguće (a niti potrebno za konkretnu namjenu) izgraditi sustav koji će moći prepoznati sve riječi iz skupa. Potrebno je odabrati samo podskup naredbi koje će sustav uspješno klasificirati. Cjelokupni proces pripreme podataka, treniranja i validacije modela popraćen je Jupyter bilježnicom koja se nalazi u GitHub repozitoriju [32].

Projektirani sustav mora moći prepoznati naredbe koje su izgovorene. Zbog toga, mora biti sposoban odbaciti sve ono što nije naredba koja se nalazi u odabranom skupu. Budući da će sustav treba raditi u stvarnim uvjetima, jedna od kategorija (klasa) na kojoj model mora biti treniran je i pozadinska buka. Ona je sveprisutna te zbog toga sustav treba biti otporan takav tip zvukova. Odabrani skup podataka sadrži zvučne zapise kao što je zvuk perilice posuđa, zvuk vode koja teče iz slavine, bijeli šum te ružičasti šum. Bijeli šum je vrsta signala koji u sebi sadrži sve frekvencije i sve imaju isti intenzitet, dok se ružičasti šum također sastoji od svih frekvencija, ali veći intenzitet imaju niže frekvencijske komponente [33]. Takvi zvukovi dobro predstavljaju tipičnu okolinu u kojoj bi se sustav mogao nalaziti. Međutim, zbog toga što skup podataka sadrži svega nekoliko minuta takvih zvučnih zapisa, potrebno je na neki način dopuniti taj podskup podataka. Naime, mreža koju treniramo će preferirati neku od klasa ako takvih primjera

ima mnogo više od primjera ostalih klasa. Drugim riječima, skup podataka za treniranje mora biti balansiran tj. primjera iz svake klase treba biti otprilike jednak broj [34]. Uz to, za rad u stvarnom svijetu, preporučljivo je u skup dodati zvučni zapis snimljen upravo na sustavu koji će i akvizirati podatke iz okoline. Budući da su svi zvučni zapisi riječi u skupu podataka duljine od otprilike jedne sekunde, pozadinske zvukove je potrebno skratiti na identičnu duljinu kako bi mreža mogla primati vrlo precizno definiranu vrstu ulaznih podataka. O broju riječi iz klasa koje su odabrane kao naredbe ovisit će koliko je potrebno isječaka koji će predstavljati klasu pozadinskih zvukova. Metoda kojom se lako može umnožiti broj pozadinskih zvukova zove se augmentacija.

Augmentacija zvuka je proces u kojem se različitim metodama može izmijeniti zvučni zapis. U ovom slučaju koristi se za povećanje broja snimaka na kojima je pozadinska buka. Umjesto da se samo kopiraju uzorci, ovakvim promjenama stvaraju se novi audio zapisi slični onima od kojih su nastali, međutim dovoljno različiti da povećaju robusnost sustava. Metode korištene u umnožavanju danih zvukova pozadinske buke su nasumično ubrzavanje i povećanje ili smanjenje glasnoće, dodavanje jeke (preklapanje originalnog zapisa s istim, ali pomaknutim u vremenu) te okretanje uzoraka u snimci (nova snimka je obrnuta od originala). Funkcija za augmentaciju prikazana je u odsječku programskog koda 3.1

```
def augment_audio(audio: AudioSegment) -> AudioSegment:
    augmented = audio
    if random.random() > 0.5:
      augmented = normalize(speedup(augmented, playback_speed=random.
   uniform(1.1, 1.5)))
    if random.random() > 0.5:
      augmented = augmented + random.uniform(-5, 5)
    if random.random() > 0.5:
      echo = augmented - random.uniform(5, 10)
      augmented = augmented.overlay(echo, position=random.randint(100,
   500))
    if random.random() > 0.5: augmented = augmented.reverse()
    return augmented
```

Kod 3.1: Augmentacija zvuka

Druga kategorija mora biti sastavljena od kombinacije različitih riječi za koje ne že-

limo klasifikaciju, tj. nisu odabrane u podskup naredbi. Ime te kategorije će biti *"nepoznato"* (engl. *unknown*), a predstavljat će sve riječi koje nisu naredbe cjelokupnog sustava. Ova kategorija je nužna za rad sustava jer se treniranjem modela na različitim riječima povećava otpornost sustava na riječi koje se ne nalaze u željenom skupu naredbi. Kada ova kategorija ne bi postojala, povećala bi se mogućnost slučajnog prepoznavanja neke riječi jer bi jedina kategorija koju sustav poznaje, a da ne predstavlja željene naredbe, bila pozadina koja u većini slučajeva nema veliku amplitudu pri akviziciji. Ostale kategorije bit će riječi odabrane kao naredbe sustava što znači da će ukupan broj klasifikacijskih kategorija biti za dva veći od broja odabranih naredbi (broj naredbi + pozadinska buka + nepoznato).

Odabir naredbi koje sustav može prepoznati je proizvoljan, a za primjer na kojem će daljnja obrada biti opisana odabrane su naredbe *"yes"*, *"no"*, *"left"*, *"right"* i *"zero"*. Zbog toga ukupni broj klasifikacijskih kategorija iznosi sedam. Uz ostale konfiguracijske parametre, odabir naredbi omogućen je na početku Jupyter bilježnice za treniranje neuronske mreže. Nakon toga formira se mapa sa sedam datoteka koje predstavljaju sedam klasifikacijskih kategorija. Broj zapisa u svakoj od kategorija odgovarat će broju zapisa u najmalobrojnijoj kategoriji upravo zbog spomenute potrebe za balansiranim skupom podataka za treniranje. Višak naredbi u nekoj od kategorija koje predstavljaju naredbe se neće koristiti, broj zapisa u kategoriji pozadinske buke generirat će se augmentacijom po potrebi, a broj nasumično odabranih zapisa u kategoriji *"nepoznato"* moguće je napraviti proizvoljno velikim.

Učitavanje zvučnih zapisa iz datotečnog sustava pojednostavljeno je korištenjem TensorFlow biblioteke, a prikazano je u odsječku programskog koda 3.2

```
train_set, validation_set = tf.keras.utils.audio_dataset_from_directory(
    directory=commands_dataset,
    batch_size=BATCH_SIZE,
    validation_split=TEST_DATASET_SIZE + VALIDATION_DATASET_SIZE,
    subset='both')
```

Kod 3.2: Učitavanje zvučnih zapisa

Učitavanjem smo dobili dva skupa podataka (engl. *dataset*): skup za treniranje i validacijski skup. Skup za treniranje koristi se, kao što mu ime kaže, za treniranje ne-

uronske mreže, dok se validacijskim skupom nakon svake epohe (pojam epoha je objašnjen u poglavlju 3.6.) provjerava točnost modela, podešavaju hiperparametri i sprječava prenaučenost. Uz to, od validacijskog skupa se još odvoji jedan dio koji se zove testni skup. On služi za konačno testiranje točnosti neuronske mreže jer te podatke mreža nije vidjela niti u jednom trenutku tijekom treniranja. Veličine tih skupova određuju parametri `TEST_DATASET_SIZE` i `VALIDATION_DATASET_SIZE`, a `BATCH_SIZE` predstavlja broj primjera koji će se odjednom davati mreži na treniranje. Spomenute parametre također je moguće podesiti na početku bilježnice.

Nakon što su skupovi podataka učitani, potrebno je izdvojiti značajke za svaki zvučni zapis. Detaljni opis generiranja značajki objašnjen je u 2.2. U dodatku C prikazano je izdvajanje značajki korišteno za ove podatke. Razlika od onog objašnjenog u spomenutom poglavlju je što se ovo izvodi na osobnom računalu i napisano je u programskom jeziku Python. Na slici u dodatku D prikazani su valni oblici nasumičnih zvučnih zapisa odabranih kategorija te pripadna matrica značajki nastala opisanim postupkom.

U ovom trenutku skupovi podataka pripremljeni su za treniranje. Sljedeći korak je definicija konkretne strukture neuronske mreže koja će biti korištena.

## 3.5. Struktura modela neuronske mreže

Principi strukturiranja konvolucijske neuronske mreže za klasifikaciju matričnih podataka poput upravo pripremljenih prate opis u poglavlju o CNN-ovima 3.2. Uz to, postoji zahtjev za što manjim modelom jer ga je potrebno implementirati na mikrokontrolerskoj platformi koja ima ograničavajuće memorijske resurse. Također, vrijeme potrebno za buđenje neuronske mreže, tj. kašnjenje koje unosi mreža implementirana na mikrokontroleru izravno utječe na performanse sustava koji bi trebao raditi u stvarnom vremenu. Imajući to na umu, izgrađena mreža bit će dovoljno jednostavna da zadovolji spomenute uvjete, a s druge strane dovoljno složena, tako da je sposobna pravilno klasificirati ulazne podatke.

Ulazni podaci su matrice dimenzija (32, 41, 12, 1). Podmatrica dimenzija (41, 12) predstavlja matricu značajki pojedinog zvučnog zapisa. U konfiguraciji je odabrano 12 MFC koeficijenata (od 2. do 13.), a zbog veličine prozora (`WINDOW_SIZE`) koja iznosi 512

(32 ms) i veličine koraka (STEP_SIZE) koja iznosi 384 (24 ms) jedna sekunda zapisa se sastoji od 41 vremenskog okvira. Dodatne dimenzije matrice predstavljaju redom broj takvih matrica koje se odjednom daju mreži na treniranje (BATCH_SIZE) te dimenzija slike koja u ovom slučaju iznosi jedan. Konvolucijske neuronske mreže također mogu raditi s višekanalnim matricama kao što su RGB slike. U tom slučaju svaki kanal predstavlja prisutnost određene boje u slici. Matrice značajki generirane nad zvučnim zapisima ponašaju se kao crno-bijele slike gdje svaka vrijednost predstavlja svjetlinu određenog piksela.

Ulazni sloj u neuronsku mrežu prate dva konvolucijska s pripadnim slojevima za poduzorkovanje. Prvi konvolucijski sloj ima 32 jezgre veličine 3x3, a drugi njih 16 iste veličine. Oba sloja za poduzorkovanje rade s matricom veličine 2x2 te pomakom iznosa dva. Slijedi ih podmreža koja se sastoji od dva potpuno povezana sloja s, redom, 8 i 16 neurona te izlazni sloj koji ima točno 7 neurona (svaki za jednu klasifikacijsku kategoriju). Aktivacije svih slojeva su "ReLu", dok izlazni sloj koristi "softmax" aktivaciju. Isječak koda 3.3 prikazuje postupak izgradnje opisane mreže.

```
model = tf.keras.Sequential([
    layers.Input(shape=input_shape),    # Input layer
    layers.Conv2D(32, kernel_size=3, padding='same', activation='relu'),
    layers.MaxPooling2D(pool_size=2, strides=2, padding='same'),
    layers.Conv2D(16, kernel_size=3, padding='same', activation='relu'),
    layers.MaxPooling2D(pool_size=2, strides=2, padding='same'),
    layers.Flatten(),   # Flatten the data for fully connected layers
    layers.Dense(8, activation='relu'),   # Fully connected layer
    layers.Dropout(0.1),                  # Dropout layer with 10% rate
    layers.Dense(16, activation='relu'),  # Fully connected layer
    layers.Dropout(0.1),                  # Dropout layer with 10% rate
    layers.Dense(num_labels, 'softmax'),  # Output layer (softmax)
])
```

Kod 3.3: Struktura mreže

Na slici 3.9. prikazan je model neuronske mreže s pripadnim brojem parametara te oblikom podataka između slojeva. Oblik podataka ima prvu dimenziju neodređenu (na slici "?") zbog toga što se mreža može trenirati s proizvoljnom veličinom grupe (brojem uzoraka koji se odjednom daju mreži).


**Slika 3.9.** Neuronska mreža [35]

## 3.6. Treniranje i vrednovanje modela

Nakon definiranja strukture modela neuronske mreže, na red je došlo treniranje. Podaci su u ovom trenutku podijeljeni u skup za treniranje, validacijski te testni skup. Također, svaki podatak (zvučni zapis) pretvoren je u dvodimenzijsku matricu značajki veličine 41x12.

```
model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=['accuracy'],
)
```

Kod 3.4: Konfiguracija za treniranje

U isječku koda 3.4 prikazana je priprema modela za treniranje. Za optimizacijski postupak odabran je Adam algoritam (engl. *Adaptive Moment Estimation*). Adam algoritam je vrsta gradijentnog spusta koja koristi prilagodljivu stopu učenja. Za funkciju gubitka odabrana je kategorička unakrsna entropija (engl. *Sparse Categorical Crossentropy*). Ona se koristi za treniranje višeklasnih klasifikacijskih modela, a matematički je opisana jednadžbom (3.9) .

$$L = -\frac{1}{N}\sum_{i=1}^{N} \log p(y_i) \tag{3.9}$$

gdje su:

- $L$: gubitak,
- $N$: broj uzoraka,
- $y_i$: oznaka primjera (klasa),
- $p(y_i)$: vjerojatnost predikcije za ispravnu klasu.

Nakon prolaska `BATCH_SIZE` (u ovom slučaju 32) uzoraka kroz mrežu, računa se gubitak na opisani način te se ažuriraju težine mreže (gradijentnim spustom). Prolazak svih uzoraka kroz mrežu označava kraj jedne epohe. Treniranje traje proizvoljan broj epoha, a u ovom slučaju može se konfigurirati varijablom `EPOCHS` U ovom slučaju odabrano ih je 50. Posljednji parametar kojim je konfigurirana mreža je metrika koja se koristi za vrednovanje modela. U ovom slučaju koristi se točnost (engl. *accuracy*) koja predstavlja postotak točno klasificiranih uzoraka u odnosu na ukupan broj uzoraka.

Početak treniranja modela prikazan je u isječku koda 3.5 Modelu su predani testni i validacijski skupovi podataka. Uz to, postavljeni su uvjeti ranijeg zaustavljanja treniranja (engl. *early stopping*) jer se može dogoditi da model konvergira u minimum funkcije gubitka prije isteka predviđenog broja epoha. Nakon što model prestane smanjivati funkciju gubitka na validacijskom skupu, treniranje se smatra završenim, a model se sprema u stanje s najmanjim gubitkom koje nije nužno stanje nakon posljednje odrađene epohe.

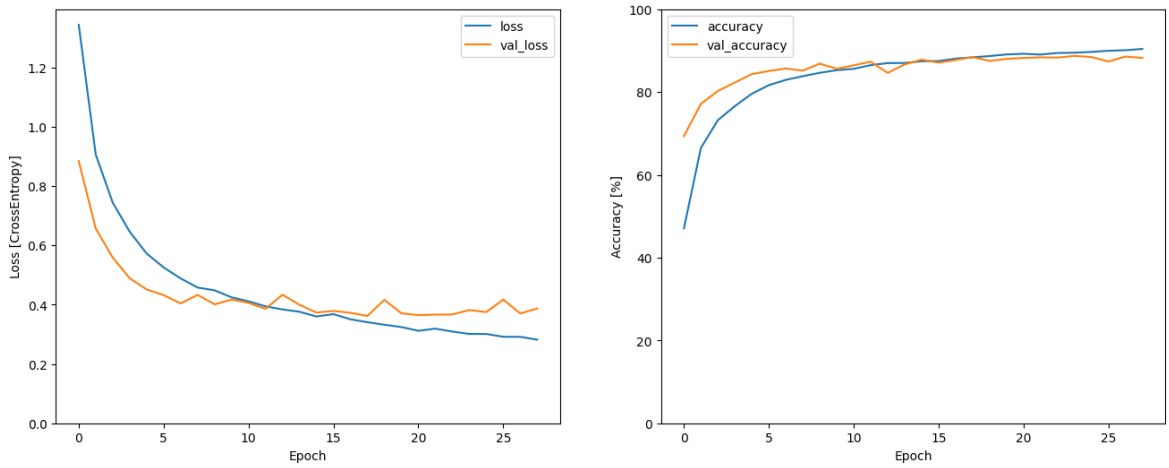
```
history = model.fit(
    train_mfcc_dataset,
    validation_data=validation_mfcc_dataset,
    epochs=EPOCHS ,
    callbacks=tf.keras.callbacks.EarlyStopping(verbose=1, patience=10,
   restore_best_weights=True),
)
```

Kod 3.5: Trening

U nastavku je prikazan proces treniranja. Vidljivo je kako se funkcija gubitka smanjuje s vremenom, a točnost modela raste. Model je konvergirao nakon 20-ak epoha te je uzeo stanje s kraja 18. epohe. U postavkama modela namješteno je da se treniranje ne zaustavi odmah nego da da modelu još određeni broj epoha koji je u ovom slučaju 10 (`patience=10`).

```
Epoch 1/50
662/662 [==============================] - 3s 5ms/step - loss: 1.3427 - accuracy: 0.4707 - val_loss: 0.8837 - val_accuracy: 0.6935
Epoch 2/50
662/662 [==============================] - 3s 5ms/step - loss: 0.9060 - accuracy: 0.6649 - val_loss: 0.6565 - val_accuracy: 0.7714
Epoch 3/50
662/662 [==============================] - 3s 5ms/step - loss: 0.7439 - accuracy: 0.7320 - val_loss: 0.5586 - val_accuracy: 0.8025
Epoch 4/50
662/662 [==============================] - 3s 5ms/step - loss: 0.6457 - accuracy: 0.7659 - val_loss: 0.4887 - val_accuracy: 0.8230
...
Epoch 28/50
662/662 [==============================] - 3s 5ms/step - loss: 0.2817 - accuracy: 0.9040 - val_loss: 0.3869 - val_accuracy: 0.8823
Epoch 28: early stopping, restoring model weights from the end of the best epoch: 18.
```

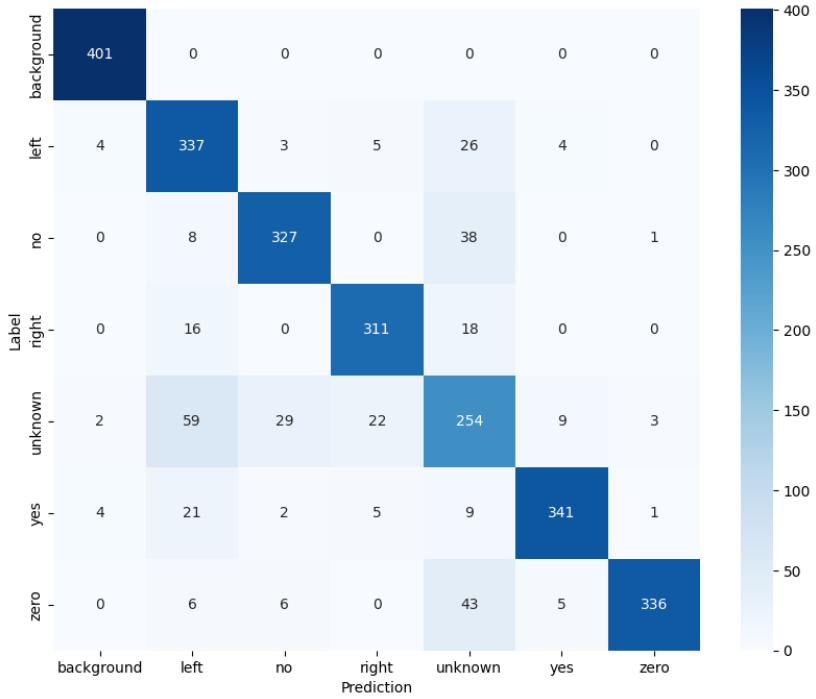
Na slici 3.10. prikazana su dva grafa. Lijevi prikazuje funkciju gubitka na skupu za treniranje i validacijskom skupu, dok desni prikazuje točnost modela na istim skupovima. Obje funkcije prikazane su u ovisnosti o broju odrađenih epoha treniranja. Vidljivo je kako se funkcija gubitka smanjuje s vremenom, a točnost raste. Nakon 20-ak epoha, funkcije se stabiliziraju.




**Slika 3.10.** Gubitak i točnost modela

Konačni iznos funkcije gubitka na testnom skupu iznosi 0,2817, a točnost modela 0,9040, dok na validacijskom skupu iznosi redom 0,3869 i 0,8823. Međutim, konačna ocjena rezultata treniranja modela donosi se na temelju testnog skupa. To su podaci koje model nije vidio niti u jednom trenutku treniranja i predstavljaju podatke kakve će model vidjeti u stvarnom svijetu. Funkcija gubitka na tom skupu iznosi 0,3480, dok točnost iznosi 0,8814.

Na slici 3.11. prikazana je konfuzijska matrica napravljena nad testnim skupom podataka. Ona prikazuje koliko je puta model pogriješio u klasifikaciji određene klase. Stupci matrice predstavljaju stvarne klase, a reci predviđene klase (izlaz treniranog modela). Na dijagonali matrice nalaze se točne klasifikacije, dok se izvan dijagonale se nalaze pogreške. Vidljivo je kako je model najviše griješio u klasifikaciji klase *"unknown"*. To je slučaj zbog toga što se u toj klasi nalaze zvučni zapisi različitih riječi. Drugim riječima, ta klasa je najraznolikija i najteža za klasificirati te je zbog toga ovakav rezultat očekivan.




**Slika 3.11.** Konfuzijska matrica

## 3.7. Usporedba s drugim modelima na istom skupu podataka

Usporediti naučeni model s postojećim modelima nije jednostavno zato što se većina modela temelji na složenijim arhitekturama koje imaju veći broj parametara te uz svoje rezultate ne prilažu način implementacije na mikrokontroleru. Međutim, u tablici 3.1. prikazana je usporedba ovog modela s drugima. Uz implementirani model, ubačen je identičan model treniran na samo dvije glasovne naredbe (ukupno četiri klase). Iz tablice je vidljivo da implementirani model zauzima najmanje memorije uz sličnu točnost za 4 klase te nešto manju za 7 klasa. Točnost bi se jednostavno mogla povećati složenijim potpuno povezanim slojevima na izlazu trenirane mreže, međutim ovo se čini kao najbolji kompromis između veličine mreže i njene točnosti.

| Model | Accuracy (%) | Model Size (KB) |
|---|---|---|
| Implementirani CNN (7 klasa) | 88,2 | 15,7 |
| Implementirani CNN (4 klase) | 93,6 | 15,6 |
| DNN (Deep Neural Network) [36] | 84,6 | 80 |
| CNN (Convolutional Neural Network) [36] | 91,6 | 79 |
| LSTM (Long Short-Term Memory) [36] | 92,9 | 79,5 |
| CRNN (Convolutional RNN) [36] | 94,0 | 79,7 |
| DS-CNN (Depthwise Separable CNN) [36] | 94,4 | 38,6 |
| TripletLoss-res15 [37] | 95,2 | 237 |
| BC-ResNet-8 [38] | 98,7 | 520 |
| WaveFormer [39] | 98,8 | 130 |

Tablica 3.1. Usporedba različitih modela

## 3.8. Prilagodba za implementaciju na mikrokontroleru

Veličina pojedinog sloja treniranog modela neuronske mreže prikazana je u nastavku. Ukupni broj parametara koje mreža ima iznosi 9439 što u memoriji zauzima malo manje od 37 KB.

```
Number of labels: 7
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 41, 12, 32)        320
 max_pooling2d (MaxPooling2D)  (None, 21, 6, 32)       0
 conv2d_1 (Conv2D)           (None, 21, 6, 16)         4624
 max_pooling2d_1 (MaxPooling2D)  (None, 11, 3, 16)     0
 flatten (Flatten)           (None, 528)               0
 dense (Dense)               (None, 8)                 4232
 dropout (Dropout)           (None, 8)                 0
 dense_1 (Dense)             (None, 16)                144
 dropout_1 (Dropout)         (None, 16)                0
 dense_2 (Dense)             (None, 7)                 119
=================================================================
Total params: 9439 (36.87 KB)
Trainable params: 9439 (36.87 KB)
Non-trainable params: 0 (0.00 Byte)
```

Međutim, spremljeni model u memoriji osim vrijednosti parametara mora imati i informaciju o samoj strukturi mreže što znatno povećava sami memorijski otisak. Datoteka s nastavkom ".pb" (engl. *protobuff*) čuva sve informacije potrebne za korištenje treniranog modela, a konkretni model u tom obliku zauzima nešto više od 173 KB. Ko-

rištenje takvog modela na mikrokontroleru nije prihvatljivo niti zbog veličine niti zbog oblika zapisa. Zbog toga je potrebno prilagoditi model. Tensorflow Lite biblioteka omogućava vrlo jednostavnu promjenu formata spremanja informacije o treniranom modelu. Format s nastavkom ".tflite" sažima model na nešto više od 41 KB. Međutim, postoji još nešto što je moguće napraviti kako bi se model sažeo još više te pretvorio u oblik pogodan za korištenje na mikrokontroleru. Spomenuta metoda sažimanja zove se kvantizacija, a oblik u kojem će model biti spremljen zove se *flatbuffer*.

Kvantizacija je metoda optimizacije modela kojom se smanjuje broj bitova potrebnih za spremanje informacije o parametrima modela. To je proces mapiranja brojeva s pomičnim zarezom u cijele brojeve. Ova redukcija preciznosti (s 32 na 8 bitova) pridonosi smanjenju veličine modela i ubrzanju izvođenja, a neznatno utječe na točnost modela [40]. Ovim zahvatom veličina modela smanjena je na 16 KB.

*Flatbuffer* je oblik za serijalizaciju podataka razvijen u Googleu. Dizajniran je za učinkovitu pohranu i pristup podacima [41]. Pretvorbom treniranog modela u ovakav oblik dobiveno je polje podataka spremno za korištenje programskim jezikom C. U isječku programskog koda 3.6 prikazan je proces kojim se model pretvara u opisano polje.

```
# Convert to a C source file
!xxd -i {QUANTIZATION_MODEL} > {MODEL_TFLITE_MICRO}
# Update variable names
REPLACE_TEXT = QUANTIZATION_MODEL.replace('/', '_').replace('.', '_')
!sed -i 's/'{REPLACE_TEXT}'/g_model/g' {MODEL_TFLITE_MICRO}
!cat {MODEL_TFLITE_MICRO}
```

Kod 3.6: Pretvorba u Flatbuffer

Tablica 3.2. prikazuje veličine modela u različitim koracima prilagodbe. Konačni model prihvatljive je veličine za implementaciju na mikrokontrolerskom sustavu, a iznosi svega 9% početne veličine modela.

| **Model** | **Veličina (B)** | **Udio veličine početnog modela (%)** |
|---|---:|---:|
| Početni model | 173241 | 100 |
| TF Lite | 41644 | 24,04 |
| TF Lite + kvantizacija | 15704 | 9,06 |

Tablica 3.2. Veličine različitih oblika modela

# 4. Eksperimentalna provjera rada sustava

Informacije o strukturi i parametrima modela neuronske mreže spremljeni su u obliku polja programskog jezika C. Sustav na mikrokontroleru polje učitava i koristi na način detaljno objašnjen u poglavlju o aktivaciji neuronske mreže na mikrokontroleru 2.3. Kako bi cjelokupni sustav radio u skladu sa zahtjevima, potrebno je u aplikaciji na mikrokontroleru dodati identične naredbe koje su odabrane prilikom treniranja modela. Stvaranje naredbi te njihovo dodavanje objektu zaduženom za prepoznavanje naredbi prikazano je u isječcima koda 4.1 i 4.2 Kategorije *"pozadina"* (engl. *"background"*) i *"nepoznato"* (engl. *"unknown"*) su predstavljene objektima klase `BlankCommand`, dok su ostale naredbe instance klase `PrintCommand`. Rezultat toga je ispis imena naredbe na konzolu u slučaju prepoznavanja govorne naredbe. Detalj koji je ključan za ispravan rad sustava je redoslijed dodavanja naredbi. On mora odgovarati redoslijedu koji je određen pri treniranju modela koji je pak određen redoslijedom učitavanja podataka za treniranje u Jupyter bilježnicu. Nakon preuzimanja podataka s interneta, mape s podacima su poredane abecedno tako da će i krajnji redoslijed naredbi biti takav.

```
BlankCommand command_back("BACKGROUND", 1, 0.7);
PrintCommand command_left("LEFT", 5, 0.8);
PrintCommand command_no("NO", 3, 0.80);
PrintCommand command_right("RIGHT", 3, 0.85);
BlankCommand command_unknown("UNKNOWN", 1, 0.7);
PrintCommand command_yes("YES", 5, 0.85);
PrintCommand command_zero("ZERO", 3, 0.85);
```

Kod 4.1: Stvaranje naredbi

```
recognizer.addCommand(&command_back);
recognizer.addCommand(&command_left);
recognizer.addCommand(&command_no);
recognizer.addCommand(&command_right);
recognizer.addCommand(&command_unknown);
recognizer.addCommand(&command_yes);
recognizer.addCommand(&command_zero);
```

Kod 4.2: Dodavanje naredbi

Zbog nesavršenosti skupa na kojem je treniran model i nejednakosti kvalitete zvučnih snimaka, neće svaka naredba biti prepoznata na isti način. Prvo, razlikovat će se pouzdanost vjerojatnosne interpretacije za različite klase. Drugo, uslijed izgovorene naredbe, klase će imati najveću vjerojatnost različit broj iteracija rada sustava. To se događa zbog toga što sustav neprestano izvodi glavnu petlju (opisanu u poglavlju o strukturi sustava 2.) te bi trebao odraditi nekoliko iteracija tijekom izgovora jedne naredbe. Zbog svega navedenog potrebno je kalibrirati svaku naredbu zasebno. Kalibracija se radi promjenom parametara pri konstrukciji objekta također prikazanog u isječku koda 4.1 Značenje pojedinog parametra detaljno je opisano u 2.4.

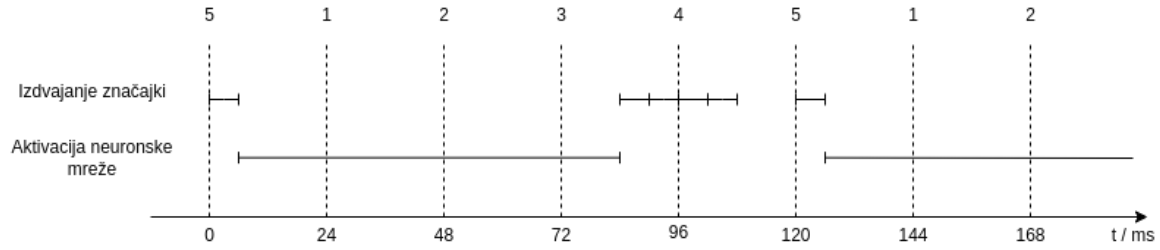
Dodatno što je potrebno eksperimentalno utvrditi je sposobnost sustava da odradi sve potrebne zadatke na vrijeme. Ne smije se dogoditi propuštanje akviziranja novih podataka zbog kašnjenja bilo kojeg drugog dijela sustava jer se time narušava svrha rada cjelokupnog sustava. U tablici 4.1. prikazana su maksimalna vremena potrebna za odrađivanje poslova pojedinih dijelova sustava koja su također utvrđena eksperimentalno.

| **Dio sustava** | **Vrijeme(ms)** |
|---|---:|
| Akvizicija uzoraka | 0,016 |
| Izdvajanje značajki | 4,9 |
| Aktivacija neuronske mreže | 77 |
| Prepoznavanje naredbi i aktivacija posla | 0,028 |

Tablica 4.1. Vrijeme potrebno za određeni posao

Iz prikazanih podataka vidljivo je da je vremenski najzahtjevniji posao aktivacija neuronske mreže, slijedi ga izdvajanje značajki, a akvizicija i prepoznavanje naredbi imaju trajanje zanemarivo u odnosu na prethodna dva. Svaka nova iteracija uzima `STEP_SIZE` novih uzoraka na obradu. U ovom slučaju taj broj iznosi 384 što odgovara 24 ms no-

vih zvučnih podataka. Podešavanje varijable `NUMBER_OF_NEW_SLICES_BEFORE_INVOKING` predstavlja krajnji korak kalibracije sustava, a odnosi se na period aktivacije neuronske mreže. Potrebno ju je postaviti na najmanji mogući broj koji neće narušavati rad sustava.




**Slika 4.1.** Kritičan trenutak rada sustava

Na slici 4.1. prikazan je kritični trenutak rada sustava u kojem je sustav u stanju čekanja na posljednji prozor podataka prije aktivacije neuronske mreže. Svi podaci koji su došli prije tog trenutka su već obrađeni. U trenutku t = 0 ms akviziraju se nova 384 podatka, tj. 24 ms novih podataka. Svi trenuci u kojima je potrebno obraditi nove podatke pojavljuju se s periodom od 24 ms te su na grafu označeni okomitom isprekidanom linijom i brojem koji predstavlja koja iteracija dohvaćanja novih podataka je u pitanju. Nakon dohvaćanja posljednjeg prozora podataka (t = 0 ms), izdvajaju se značajke nad tim prozorom te aktivira neuronska mreža. Vodoravne linije predstavljaju trajanja izdvajanja značajki i procesiranja podataka u neuronskoj mreži. Duljine linija otprilike odgovaraju trajanju procesa: izdvajanje značajki 6 ms, a obrada u neuronskoj mreži 78 ms. Uzeto je malo dulje trajanje kako bi se uzelo u obzir bilo kakvo nepredviđeno mrtvo vrijeme sustava. Vidljivo je da aktivacija i procesiranje podataka unosi u sustav kašnjenje veće od perioda akvizicije novih podataka. Potrebno je provjeriti nakon koliko novih iteracija akvizicije i obrade podataka sustav opet može aktivirati neuronsku mrežu, tj. koliki je period aktivacije.

Ako bi se postavio period aktivacije na četiri, sustav bi trebao odraditi aktivaciju prošlog perioda i obradu svih novih podataka prije trenutka t = 96 ms. U tom trenutku sustav bi se trebao u najgorem slučaju nalaziti u istom stanju kao u početnom trenutku (t = 0 ms). Na grafu je vidljivo da sustav u tom trenutku neće stići obraditi sve što je potrebno, ali je vrlo blizu toga (uzmimo u obzir i grafičko produljenje trajanja obrade podataka i aktivacije neuronske mreže). Ono što je poželjno je da sustav ne bude na rubu stabilnosti jer malim kašnjenjem će se kroz vijek rada uređaja akumulirati kašnjenje i u jednom tre-

nutku će doći do gubljenja podataka. Za točnu provjeru stabilnosti, problem je potrebno riješiti analitički.

Zaključak je da je u jednom periodu rada sustava, tj. aktivacije neuronske mreže, potrebno izdvojiti značajke za zadnji prozor podataka iz prošlog perioda, aktivirati neuronsku mrežu za prošli period te generirati značajke za sve nove prozore podataka u tom periodu, ne uključujući posljednji prozor. Neka je $t_{fg}$ vrijeme potrebno za izdvajanje značajki, $t_{nn}$ vrijeme potrebno za proces obrade podataka u neuronskoj mreži, $t_{new}$ veličina novih podataka u ms, a N broj novih iteracija akvizicije i obrade podataka prije aktivacije mreže (period). Tada vrijedi:

$$t_{fg} + t_{nn} + (N-1) \cdot t_{fg} \leq N \cdot t_{new} \tag{4.1}$$

Iz toga proizlazi da minimalni broj novih iteracija prije aktivacije mreže mora zadovoljavati uvjet:

$$N \geq \frac{t_{nn}}{t_{new} - t_{fg}} \tag{4.2}$$

Za konkretne iznose iz tablice 4.1. i $t_{new} = 24$ ms proizlazi da je $N \geq 4,03$. Rezultat je vrlo blizu mogućnosti korištenja već spomenutog perioda aktivacije mreže koji iznosi 4. Međutim, najmanji mogući period, a da sustav sigurno ostane stabilan, iznosi 5. Time je, ako se uzme u obzir da širina matrice značajki predstavlja jednu sekundu ulaznih podataka i iznosi 41, dobivena brzina koja iznosi 8,2 aktivacije neuronske mreže po sekundi što je sasvim dovoljno za sustav ovakvog tipa.

# 5. Zaključak

U ovom radu uspješno je implementiran sustav za prepoznavanje govornih naredbi u stvarnom vremenu na mikrokontrolerskoj platformi ESP32 Lyrat. U usporedbi s nekim od javno dostupnih sličnih rješenja otvorenog koda (kao. npr. [42], [43]), omogućuje jednostavniju modularnu izvedbu i mogućnost pouzdanog prepoznavanja većeg skupa glasovnih naredbi.

Razvijeno je cjelovito programsko rješenje za mikrokontrolerski sustav koje uključuje različite podsustave: akvizicija zvuka s mikrofona, obrada signala u svrhu izdvajanja značajki govora iz Mel-kepstralnih koeficijanta (MFCC), neuronska mreža za prepoznavanje naredbi i podsustav za aktivaciju radnje u stvarnom vremenu na temelju prepoznate glasovne naredbe. Neuronska mreža implementirana je na mikrokontroleru korištenje biblioteke TensorFlow Lite. U implementaciji posebna je pažnja posvećena niskoj latenciji odziva sustava te robusnosti radi minimiziranja mogućnosti pogrešnog prepoznavanja naredbe ili uzastopnog prepoznavanja više naredbi kada je izgovorena samo jedna. Kako bi se osigurala učinkovita implementacija i povezanost svih dijelova sustava, korišten je operacijski sustav za rad u stvarnom vremenu FreeRTOS.

Implementirani sustav može uspješno prepoznati pet predefiniranih glasovnih naredbi nad kojima je trenirana neuronska mreža i aktivirati odgovarajući posao dodijeljen svakoj od njih. Također, omogućena je mogućnost dinamičkog odabira drugačijeg podskupa govornih naredbi bez potrebe za ponovnim treniranjem mreže, a dodatno je moguće proširiti sam skup naredbi željenim zvučnim zapisima koji će predstavljati naredbu prilagođenu korisniku.

Posebna pažnja posvećena je modularnoj strukturi sustava koja omogućuje jednostavnu prilagodnu pojedinih dijelova sustava za različite namjene ili drugačije sklopovske

platforme. Trenutna implementacija može poslužiti kao osnova za implementaciju različitih sustava za prepoznavanje i drugih vrsta senzorskih podataka ili za korištenje drugih vrsta neuronskih mreža.

Jedno od mogućih budućih poboljšanja sustava odnosi se na izbjegavanje korištenja brojeva u aritmetici s pomičnim zarezom u generiranju MFC koeficijenata korištenjem cjelobrojnih tipova podataka, što bi dodatno pridonijelo brzini izvršavanja i mogućnosti korištenja na manje naprednim sklopovskim platformama. Naime, na mikrokontroleru na kojem je implementiran rad to ne predstavlja problem zbog postojanja jedinice za operacije s pomičnim zarezom (engl. *Floating Point Unit ili FPU*), ali na drugim mikrokontrolerima na kojima ona nije raspoloživa računanje s cjelobrojnim koeficijentima osjetno bi ubrzalo cijeli proces.

# Literatura


[1] K. Pykes, “What is tinyml? an introduction to tiny machine learning”, https://www.datacamp.com/blog/what-is-tinyml-tiny-machine-learning, 2023., [Posjećeno: siječanj 2025.].

[2] “Diagram maker”, https://app.diagrams.net/, [Posjećeno: siječanj 2025.].

[3] “Esp32-lyrat v4.3 getting started guide”, https://docs.espressif.com/projects/esp-adf/en/latest/design-guide/dev-boards/get-started-esp32-lyrat.html, [Posjećeno: siječanj 2025.].

[4] E. Semiconductor, *ES8388 Low Power Stereo Audio Codec with Integrated Headphone Amplifier*, 2025., [Posjećeno: siječanj 2025.]. [Mrežno]. Adresa: http://www.everest-semi.com/pdf/ES8388%20DS.pdf

[5] P. Warden i D. Situnayake, *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. Sebastopol, CA: O’Reilly Media, 2020. [Mrežno]. Adresa: https://www.oreilly.com/library/view/tinyml/9781492052036/

[6] “Freertos (supplemental features)”, https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/system/freertos_additions.html#ring-buffers, [Posjećeno: siječanj 2025.].

[7] D. Petrinović, *Digitalna obrada govora*, Zagreb, 2002., interna zavodska skripta, 08. 02. 2002.

[8] J. Yang, “A low-power keyword spotting chip with multiplier-free mfcc feature extractor”, *IEICE Electronics Express*, 01 2025. https://doi.org/10.1587/elex.22.20250008

[9] S. Patnaik, “Speech emotion recognition by using complex mfcc and deep sequential model”, *Multimedia Tools and Applications*, sv. 82, 09 2022. https://doi.org/10.1007/s11042-022-13725-y

[10] M. S. Sidhu, N. A. A. Latib, i K. K. Sidhu, “Mfcc in audio signal processing for voice disorder: a review”, *Multimedia Tools and Applications*, 2024., [Posjećeno: siječanj 2025.]. https://doi.org/10.1007/s11042-024-19253-11

[11] A. Vasilijević i D. Petrinović, “Perceptual significance of cepstral distortion measures in digital speech processing”, *Automatika: časopis za automatiku, mjerenje, elektroniku, računarstvo i komunikacije*, sv. 52, br. 2, str. 132–146, 2011., izvorni znanstveni članak. https://doi.org/https://hrcak.srce.hr/71297

[12] “Window function”, https://en.wikipedia.org/wiki/Window_function, [Posjećeno: siječanj 2025.].

[13] “Mel scale”, https://en.wikipedia.org/wiki/Mel_scale, [Posjećeno: siječanj 2025.].

[14] TensorFlow, “TensorFlow Lite for Microcontrollers (TFLM)”, [Posjećeno: siječanj 2025.]. [Mrežno]. Adresa: https://github.com/tensorflow/tflite-micro

[15] B. Q. Kevin Ezra, “Introduction to Neural Network”, https://iq.opengenus.org/dense-layer-in-tensorflow/, [Posjećeno: siječanj 2025.].

[16] G. for Geeks, “What is Fully Connected Layer in Deep Learning?” https://www.geeksforgeeks.org/what-is-fully-connected-layer-in-deep-learning/, 2024., [Posjećeno: siječanj 2025.].

[17] V. Labs, “Activation functions in neural networks [12 types and use cases]”, https://www.v7labs.com/blog/neural-networks-activation-functions, 2021., accessed: January 2025.

[18] M. K. Analyticsvidhya, “Gradient Descent vs. Backpropagation: What’s the Difference?” https://www.analyticsvidhya.com/blog/2023/01/gradient-descent-vs-backpropagation-whats-the-difference/, 2023., [Posjećeno: siječanj 2025.].

[19] S. PyCodeMates, https://www.pycodemates.com/2023/06/introduction-to-convolutional-neural-networks.html, [Posjećeno: siječanj 2025.].

[20] V. H. Phung i E. J. Rhee, “A high-accuracy model average ensemble of convolutional neural networks for classification of cloud image patches on small datasets”, *Applied Sciences*, sv. 9, br. 4500, 2019. https://doi.org/doi:10.3390/app9214500

[21] “Tensorflow keras conv2d”, https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D, [Posjećeno: siječanj 2025.].

[22] E. Gračan, “Prepoznavanje uzoraka pomoću neuronskih mreža”, Diplomski rad, Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet, Matematički odsjek, Zagreb, Hrvatska, 2020., uRN:NBN: urn:nbn:hr:217:771401. [Mrežno]. Adresa: https://urn.nsk.hr/urn:nbn:hr:217:771401

[23] J. Brownlee, “How do convolutional layers work in deep learning neural networks?” https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/, 2020., [Posjećeno: siječanj 2025.].

[24] “Apply a 2d convolution operation in pytorch”, https://www.geeksforgeeks.org/apply-a-2d-convolution-operation-in-pytorch/, 2023., [Posjećeno: siječanj 2025.].

[25] D. E. Swapna, “Convolutional neural networks, deep learning”, https://developersbreach.com/convolution-neural-network-deep-learning/, [Posjećeno: siječanj 2025.].

[26] “Rectified linear units (relu) in deep learning”, https://www.kaggle.com/code/dansbecker/rectified-linear-units-relu-in-deep-learning, [Posjećeno: siječanj 2025.].

[27] M. Yani i C. Setianingsih, “Application of transfer learning using convolutional neural network method for early detection of terry’s nail”, *Journal of Physics: Conference Series*, sv. 1201, br. 1, str. 012052, May 2019. https://doi.org/10.1088/1742-6596/1201/1/012052

[28] “Convolutional neural networks (cnn): Step 3 - flattening”, https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening, [Posjećeno: siječanj 2025.].

[29] J. Šnajder, “Logistička regresija 2”, Strojno učenje 1, UNIZG FER, ak. god. 2022./2023., predavanja, v1.10, 2023., sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva.

[30] IndianTechWarrior, https://indiantechwarrior.com/convolutional-neural-network-architecture/, [Posjećeno: siječanj 2025.].

[31] P. Warden, “Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition”, *ArXiv e-prints*, travanj 2018. [Mrežno]. Adresa: https://arxiv.org/abs/1804.03209

[32] L. Balić, “Keyword spotting”, https://github.com/lbalic21/keyword_spotting, 2025., [Posjećeno: siječanj 2025.].

[33] “White noise vs pink noise”, https://getsnooz.com/blogs/snoozweek/white-noise-vs-pink-noise, [Posjećeno: siječanj 2025.].

[34] “Introduction to balanced and imbalanced datasets in machine learning”, https://encord.com/blog/an-introduction-to-balanced-and-imbalanced-datasets-in-machine-learning/, [Posjećeno: siječanj 2025.].

[35] L. Roeder, “Netron: Visualizer for neural network, deep learning, and machine learning models”, https://github.com/lutzroeder/netron, 2025., [Posjećeno: siječanj 2025.].

[36] Y. Zhang, N. Suda, L. Lai, i V. Chandra, “Hello edge: Keyword spotting on microcontrollers”, *arXiv preprint arXiv:1711.07128*, 2017. [Mrežno]. Adresa: https://arxiv.org/abs/1711.07128

[37] R. Vygon i N. Mikhaylovskiy, “Learning efficient representations for keyword spotting with triplet loss”, *ArXiv*, sv. abs/XXXX.XXXXX, 2021., accessed: Feb. 13, 2025. [Mrežno]. Adresa: https://arxiv.org/abs/XXXX.XXXXX

[38] B. Kim, S. Chang, J. Lee, i D. Sung, “Broadcasted residual learning for efficient keyword spotting”, u *Proceedings of Interspeech 2021*. International Speech Communication Association, 2021., str. 4538–4542. https://doi.org/10.21437/Interspeech.2021-383

[39] M. Scherer, C. Cioflan, M. Magno, i L. Benini, “Work in progress: Linear transformers for tinyml”, u *Proceedings of the 2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Valencia, Spain: IEEE, 2024., str. N/A, accessed: Feb. 13, 2025. https://doi.org/10.23919/DATE58400.2024.10546828

[40] MathWorks, “Quantization”, 2025., accessed: Feb. 13, 2025. [Mrežno]. Adresa: https://www.mathworks.com/discovery/quantization.html#:~:text=Quantization%20is%20the%20process%20of,and%20range%20of%20a%20value.

[41] Google, “Flatbuffers documentation”, 2025., accessed: Feb. 13, 2025. [Mrežno]. Adresa: https://flatbuffers.dev/

[42] A. Software, “Ml-kws-for-mcu: Machine learning-based keyword spotting for microcontrollers”, 2024., accessed: Feb. 13, 2025. [Mrežno]. Adresa: https://github.com/ARM-software/ML-KWS-for-MCU

[43] ——, “Avh-tflmicrospeech: Arm virtual hardware for tensorflow lite micro speech”, 2024., accessed: Feb. 13, 2025. [Mrežno]. Adresa: https://github.com/ARM-software/AVH-TFLmicrospeech

# Sažetak


## Sustav za prepoznavanje govornih naredbi u stvarnom vremenu na rubnim uređajima

Luka Balić

U okviru rada razvijen je sustav za prepoznavanje govornih naredbi u stvarnom vremenu temeljen na mikrokontrolerskoj platformi ESP32 Lyrat. Implementacija programskog rješenja koje se izvodi na mikrokontroleru obuhvaća podsustave za akviziciju zvuka, izdvajanje značajki govora iz Mel-kepstralnih koeficijanata (MFCC), konvolucijsku neuronsku mrežu (CNN) za prepoznavanje glasovnih naredbi i podsustav za aktivaciju radnji na temelju prepoznate naredbe u stvarnom vremenu. Za implementaciju neuronske mreže na mikrokontroleru korištena je biblioteka TensorFlow Lite. Za učinkovitu implementaciju rada u stvarnom vremenu korišten je operacijski sustav FreeRTOS. Posebna pažnja posvećena je niskoj latenciji i robusnosti sustava na različite izvore pogrešaka i smetnji. Sustav je implementiran modularno te omogućuje jednostavno konfiguriranje naredbi i akcija, kao i prilagodbu za druge sklopovske platforme.



**Ključne riječi:** obrada zvuka u stvarnom vremenu; prepoznavanje govora; Mel-frekvencijski kepstralni koeficijenti (MFCC); strojno učenje na rubnim uređajima; konvolucijske neuronske mreže (CNN); TensorFlow Lite; mikrokontroleri; ESP32 Lyrat

# Abstract


## A system for recognizing voice commands in real-time on edge devices

Luka Balić

In this thesis a real-time speech command recognition system was developed, based on the ESP32 Lyrat development system. The software for microcontroller includes modules for audio acquisition, speech feature extraction based on Mel-frequency cepstral coefficients (MFCC), a convolutional neural network (CNN) for speech command recognition, and a subsystem for executing actions based on the recognized command in a real time. The TensorFlow Lite library was used for neural network implementation on the microcontroller. FreeRTOS operating system was used to enable efficient implementation of real-time tasks. Particular attention was paid to low-latency system response and the system robustness in terms of various sources of errors and interference. The system was implemented in a modular manner to allow for easy configuration of commands and actions, as well as adaptation to other hardware platforms.



**Keywords:** real-time audio processing; speech recognition; Mel-frequency cepstral coefficients (MFCC); edge AI; convolutional neural networks (CNN); TensorFlow Lite; microcontrollers; ESP32-LyraT

# Privitak A: Skup podataka za treniranje

| Riječ | Broj zapisa | Riječ | Broja zapisa |
|---|---|---|---|
| Backward | 1,664 | Bed | 2,014 |
| Bird | 2,064 | Cat | 2,031 |
| Dog | 2,128 | Down | 3,917 |
| Eight | 3,787 | Five | 4,052 |
| Follow | 1,579 | Forward | 1,557 |
| Four | 3,728 | Go | 3,880 |
| Happy | 2,054 | House | 2,113 |
| Learn | 1,575 | Left | 3,801 |
| Marvin | 2,100 | Nine | 3,934 |
| No | 3,941 | Off | 3,745 |
| On | 3,845 | One | 3,890 |
| Right | 3,778 | Seven | 3,998 |
| Sheila | 2,022 | Six | 3,860 |
| Stop | 3,872 | Three | 3,727 |
| Tree | 1,759 | Two | 3,880 |
| Up | 3,723 | Visual | 1,592 |
| Wow | 2,123 | Yes | 4,044 |
| Zero | 4,052 | | |

Tablica A1. Broj zvučnih zapisa različitih riječi u skupu [31]

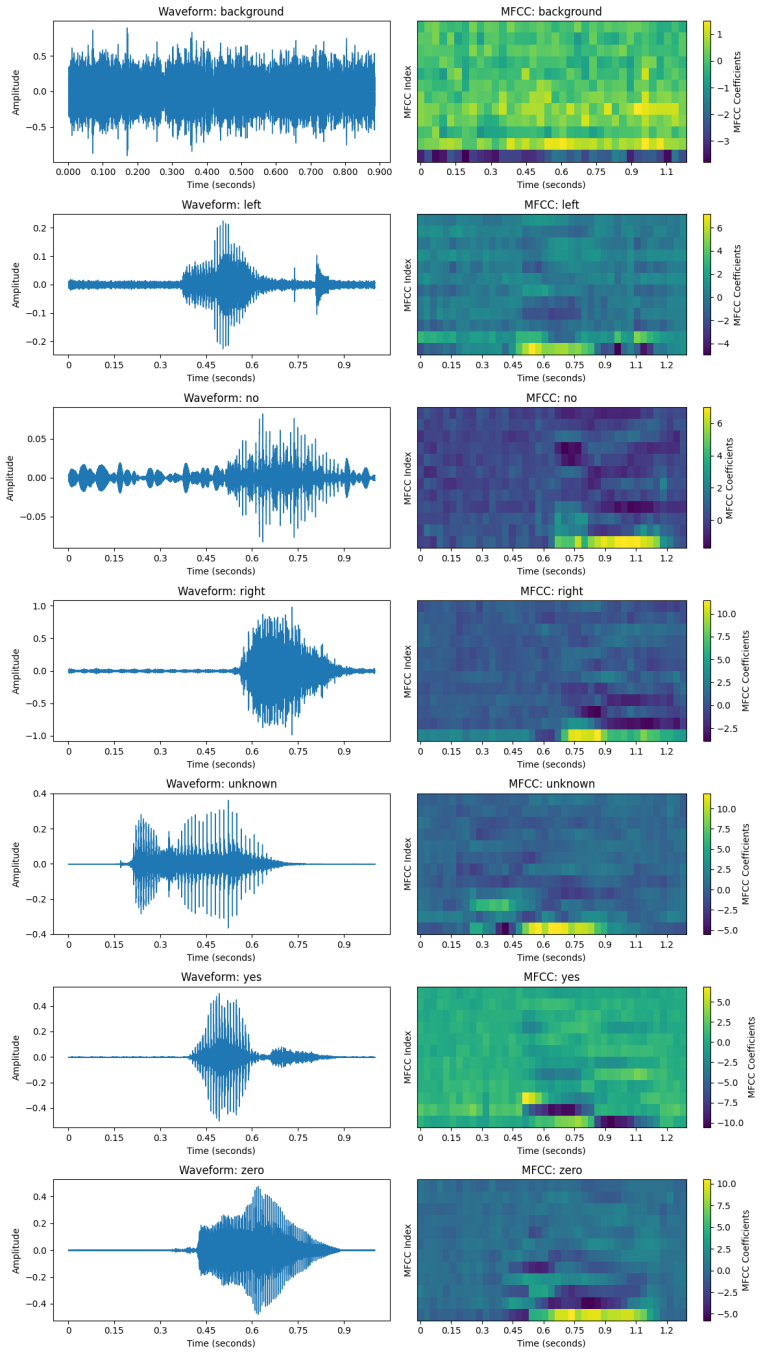
# Privitak B: Configuration.hpp

```
#ifndef _CONFIGURATION_H_
#define _CONFIGURATION_H_

/* Application configuration */

#define SAMPLE_RATE                                 (16000)
#define WINDOW_SIZE                                 (512)
#define STEP_SIZE                                   (384)
#define NUMBER_OF_TIME_SLICES                       ((int)((SAMPLE_RATE -
    WINDOW_SIZE) / STEP_SIZE) + 1)
#define NUMBER_OF_SPECTROGRAM_BINS                  ((WINDOW_SIZE / 2) + 1)
#define NUMBER_OF_MEL_BINS                          (40)
#define LOWER_BAND_LIMIT                            (80.0)
#define UPPER_BAND_LIMIT                            (7600.0)
#define NUMBER_OF_MFCCS                             (12)
#define NUMBER_OF_FEATURES                          (NUMBER_OF_TIME_SLICES *
    NUMBER_OF_MFCCS)
#define NUMBER_OF_NEW_SLICES_BEFORE_INVOKING        (5)
#define COOL_DOWN_PERIOD_MS                         (1500)
#define MAX_COMMANDS                                (10)

#endif /* _CONFIGURATION_H_ */
```

# Privitak C: Izdvajanje značajki

```
# izdvajanje znacajki koristeno u Jupyter biljeznici
def generate_mfccs(audio):
    stft = tf.signal.stft(
        audio,
        frame_length=WINDOW_SIZE,
        frame_step=STEP_SIZE,
        fft_length=WINDOW_SIZE,
        window_fn=tf.signal.hamming_window
    )
    spectrogram = tf.abs(stft)
    numSpectrogramBins = stft.shape[-1]
    linearToMel = tf.signal.linear_to_mel_weight_matrix(
        num_mel_bins=NUMBER_OF_MEL_BINS,
        num_spectrogram_bins=num_spectrogram_bins,
        sample_rate=SAMPLE_RATE,
        lower_edge_hertz=LOWER_BAND_LIMIT,
        upper_edge_hertz=UPPER_BAND_LIMIT
    )
    melSpectrogram = tf.tensordot(spectrogram, linearToMel, axes=1)
    melSpectrogram.set_shape(spectrogram.shape[:-1].concatenate([
   NUMBER_OF_MEL_BINS]))
    logMelSpectrogram = tf.math.log(mel_spectrogram + 1e-6)
    mfccs = tf.signal.mfccs_from_log_mel_spectrograms(logMelSpectrogram)
    mfccs = mfccs[..., 1:NUMBER_OF_MFCCS]
    return mfccs[..., tf.newaxis]
```

# Privitak D: Matrice značajki




**Slika D1.** Zvučni signali i pripadna matrica MFC koeficijenata