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, 14. veljače 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 svima...

Sadržaj

1. Uvod

1.1. Tiny ML

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 performanci i neograničavajućih resursa, međutim sve bržim razvojem IoT (eng. 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 (eng. 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.

1.2. Opis problema

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 te bez ikakvog problema komunicirati s osobom koja ih koristi. Za prepoznavanje i obradu ljudskog govora te 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

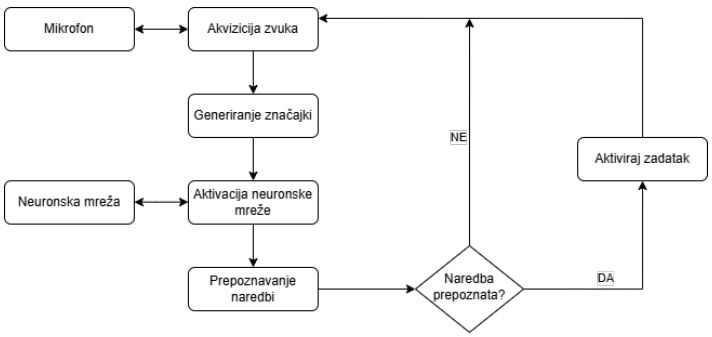
konstantnu internetsku vezu te stabilni i dugotrajni izvor električne energije. Kada bi mobilni uređaji slali konstatan tok audio podataka na server, brzo bi ispraznili bateriju te nepotrebno koristili mobilne podatke za pristup internetu. Zbog toga su takvi sustavi osmišljeni da čekaju naredbu za početak komunikacije, a tek onda uspostave vezu sa serverom. Međutim, i dalje nam ostaje problem konstantne akvizicije ulaznih audio 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 optimizirane modele strojnog učenja. Takav procesor bi konstantno akvizirao podatke s audio ulaza te lokalno, uz pomoć treniranog modela, čekao ključnu riječ te nakon prepoznavanja dao znak cijelom sustavu da se može ”probuditi” iz stanja niske potrošnje te odraditi svoj posao. Ovakvim pristup, postignuta je efikasnost, niska potrošnja, brz odaziv, smanjena je potrošnja internetskih podataka 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 trenucima u kojima želimo.

2. Struktura sustava

Svrha cjelokupnog sustava je okidanje obavljanja određenog procesa, tj. signalizacija za početak obavljanja nekog posla. Također, sustav mora biti sposoban pokrenuti proces u bilo kojem trenutku, a odaziv bi trebao biti trenutan što znači da cijeli proces prikupljanja, obrade te izvršavanja mora raditi u stvarnom vremenu. Shodno tome, sustav je izgrađen od četiri modularna podsustava:

- Akvizicija zvuka
- Generiranje značajki
- Aktivacija neuronske mreže
- Prepoznavanje naredbi

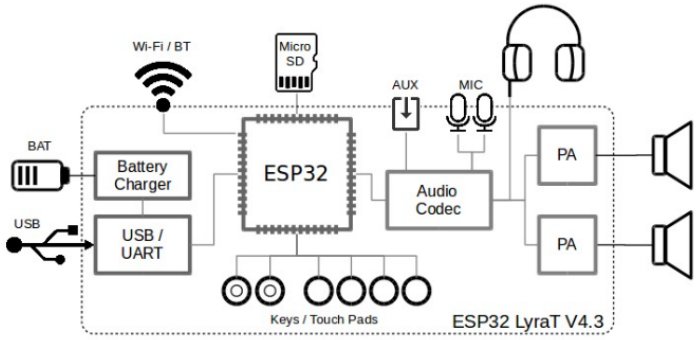
Ideja u pozadini navede podjele je lakša prilagodba cjelokupnog sustava na različite mikrokontrolerske platfome, drugačiju obradu akviziranog zvuka, korištenje drugačije neuronske mreže ili samo kreiranje specifičnog zadatka koji će se aktivirati određenom naredbom. Na slici 2.1. grafički je prikazan opisani sustav.



Slika 2.1. Struktura sustava [2]

2.1. Akvizicija zvuka

Podsustav za akviziciju zvuka je usko vezan uz platformu na kojoj je sustav implementiran zbog toga što je zadužen za komunikaciju sa sustavom koji prikuplja stvarne signale sa senzora (mikrofona). ESP32 Lyrat Development Board [3], na kojem je sustav implementiran, na sebi 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 upravo ESP32-WROVER-E mikrokontroler na kojem je cjelopukni sustav za prepoznavanje govnornih naredbi implementiran. Upravo on je 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 konektor). Mikrokontroler putem I2C sučelja konfigurira kodek, dok konkretne zvučne signale uzima 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.

```
1     audio_board_handle_t board_handle = audio_board_init();
2     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 (Espressif Audio Development Framework). To je biblioteka koja pojednostavljuje razvoj aplikacija vezanih za obradu zvuka kao što je 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 samo pokrenuti akvizicijski sustav, a to je prikazano u programskom kodu 2.1.

```
1    audio_element_handle_t i2s_stream_reader;
2    i2s_stream_cfg_t i2s_cfg = I2S_STREAM_CFG_DEFAULT();
3    i2s_cfg.type = AUDIO_STREAM_READER;
4    i2s_cfg.i2s_port = I2S_NUM_0;
5    i2s_cfg.i2s_config = i2s;
6    i2s_stream_reader = i2s_stream_init(&i2s_cfg);
```

Kod 2..2: Pokretanje akvizicijskog sustava

Frekvencija otipkavanja postavljena u konfiguraciji iznosi 16000 Hz što je dovoljno za ovakav tip sustava [5]. Nakon pokretanja akvizicijskog podsustava, sve što je preostalo je uzimati nove podatke putem I2S sučelja. Konstantno uzimanje novih podataka, tj. komunikacija s ES8388 kodekom, odvojeno je u posebnu dretvu. Na taj način programski je odvojena akvizicija sirovih podataka (sirovi u smislu da nisu još obrađeni ni na kakav način) od ostatka sustava. Ovaj podsustav je proizvođač novih podataka, dok je ostatak sustava potrošač (također jednodretven). Sve što je preostalo je ubaciti pouzdan i efikasan način komunikacije između. Za tu svrhu odabran je kružni međuspremnik (engl. ring buffer). On omogućuje asinkrono pisanje u njega te čitanje iz njega. Implementacija takve strukture dostupna je unutar FreeRTOS-a **??**

```
1    class AudioRecorder
2    {
3        private:
4            uint32_t sampleRate;
5            RingbufHandle_t ringBuffer;
6            TaskHandle_t captureAudioHandle;
```

```
7           static void captureAudioTask(void* pvParameters);
8       public:
9           AudioRecorder(uint32_t sampleRate) : sampleRate(sampleRate),
10                                               ringBuffer(NULL),
11                                               captureAudioHandle(NULL)
    {}
12          uint32_t getSamples(int16_t* samples, size_t numOfSamples);
13  };
```

Kod 2..3: Razred AudioRecorder

Opisani podsustav za akviziciju u programskom kodu zapakiran je u razred imena AudioRecorder 2.1. Prema van, razred nudi metodu potpisa `uint32_t getSamples(int16_t* samples, size_t numOfSamples)` koja omogućuje dohvaćanje proizvoljnog broja `uint16_t` podataka jer se akvizirani zvučni signal sprema upravo u tom obliku. Zbog ovakve strukture ovog podsustava, sve što je potrebno u ostatku sustava je stvaranje objekta razreda AudioRecorder, alociranje spremnika za dohvaćanje podataka te kontinuirano pozivanje opisane funkcije koja kao argumente prima pokazivač na alocirani spremnik te željeni broj audio uzoraka.

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

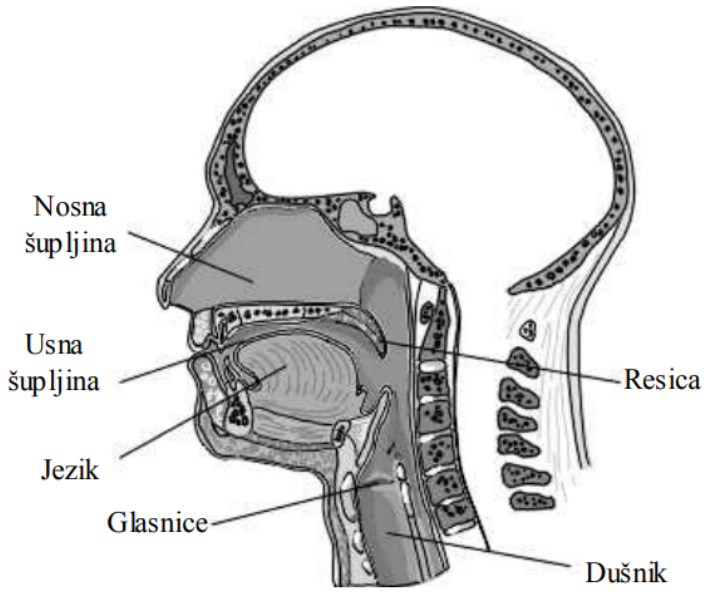
2.2. Generiranje značajki

Generiranje značajki je drugi podsustav spomenut u uvodnom dijelu, također prikazan na slici 2.1. Nakon što je omogućena akvizicija zvučnih uzoraka, potrebno ih je obraditi kako bi se mogli predati neuronskoj mreži na klasifikaciju. Naime, modeli strojnog učenja (kao što je neuronska mreža) rade s različitim primjerima podataka. Razlikovanje primjera temelji se na odredivanju različitih značajki ulaznih podataka. Klasičan primjer koji se koristi kako bi se približio pojam značajki jest model predikcije cijene nekakve nekretnine. U tom slučaju značajke koje bi model mogao koristiti su lokacija,

godina izgradnje, površina, razina energetske učinkovitosti i slično. Medutim, što bi bile značajke našeg ulaznog toka signala? Možemo, ispostavit će se naivno, uzeti amplitudu svake vrijednosti. U slučaju da promatramo period od jedne sekunde signala s već spomenutim otipkavanjem od 16 kHz, broj značajki koje bi naš model morao "progutati" jest 16000. Obraditi toliku količinu podataka u jako kratkom vremenu (sustav mora raditi bez konstantno i bez mrtvog vremena) na resursno ograničenom sustavu kao što je mikrokontroler ne zvuči obećavajuće. Nekako sve upućuje na to da je potrebno na neki način prilagoditi ulazni signal, tj. izvući iz signala bitne informacije i tako smanjiti veličinu podataka.

2.2.1. Model govora

Kako bismo identificirali kojim značajkama bi bilo korisno opisati ljudski govor, potrebno je na neki način modelirati 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.



Slika 2.3. Presjek glave i osnovni dijelovi vokalnog trakta koji sudjeluju u produkciji govornog signala [6]

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

tost 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) [6].

Jednostavni model koji se koristi u području obrade prirodnog govora je da se on 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 slučajnim šumom (engl. random noise signal). Opisani sustav može se prikazati jednadžbom 2.1

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

gdje je:

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

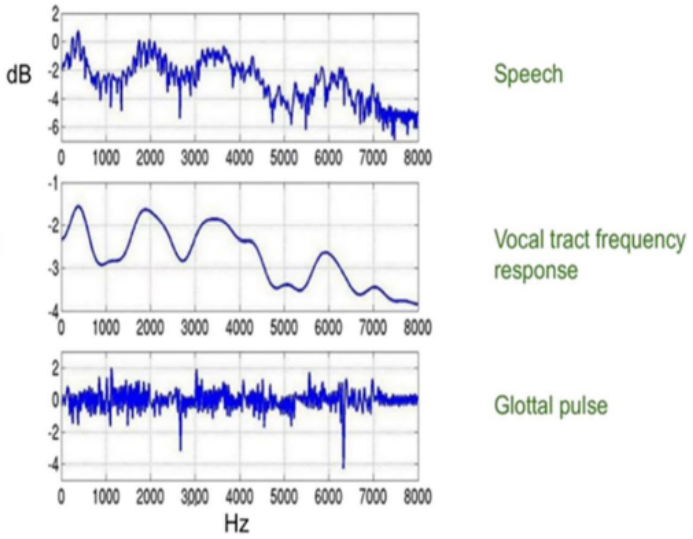
U vremenskoj domeni isti sustav može se prikazati jednadžbom 2.2 Množenju u frekvencijskoj domeni istovjetna je konvolucija u vremenskoj (i obratno!).

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

Cilj ovakvog modeliranja je pronaći bitne informacije u govornom signalu. Pretpostavka na kojoj se temelji daljnji rad je da je skoro sva informacija iz govornog signala sadržana u prijenosnoj funkciji govornog trakta, tj. pobuda (kvazi-periodični impulsi) ostaje konstantnna tijekom govora (veliko pojednostavljenje, međutim svi modeli na kojima se temelji ovo područje zasnivaju na ovoj činjenici [7, 8, 9]). Zbog toga želimo pronaći način za razdvajanje tih dvaju elemenata modela. Ako primijenimo logaritamsku funkciju na sustav opisan u frekvencijskoj domeni proizlazi 2.3

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

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



Slika 2.4. Glasovni signal rastavljen na pobudu i odziv vokalnog trakta [9]

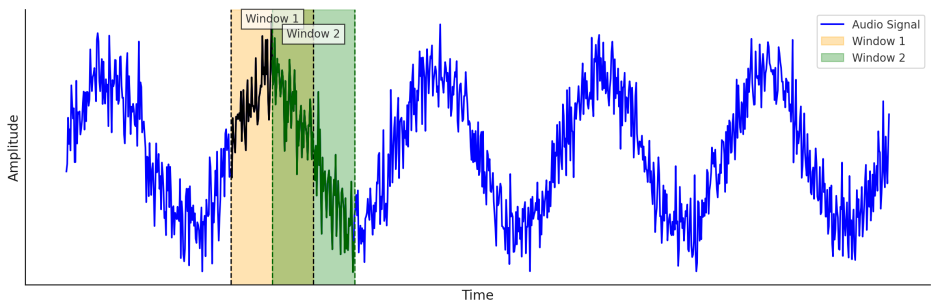
Jedino što preostaje je pronaći način za što efikasniji opis odziva vokalnog trakta. On će u konačnici predstavljati zvučne zapise na kojima će model neuronske mreže biti treniran. U području automatskog prepoznavanja govora te identifikaciji govornika uvelike se koriste Mel kepstralni koeficijenti.

2.2.2. MFCC

Mel kepstralni koeficijenti (engl. Mel frequency cepstral coefficients ili MFCC), tj. mjera euklidske udaljenosti MFCC vektora jedna je od najčešće korištenih mjera u automatskom prepoznavanju govora i govornika [10]. MFC koeficijenti (koji čine MFC vektor) pokazali su se odličnim načinom za spremanje informacije koja je sadržana u govoru, a konkretno predstavljaju kratkotrajni spektar snage glasovnog signala. Naš sustav za prepoznavanje govornih naredbi će koristiti upravo njih za značajke koje predstavljati zvučni signal akviziran pomoću podsustava za akviziciju. Najbolji način za opis ovih koeficijenata je prikaz postupka kojim se dobivaju.

Preklapajući prozori

Izlaz iz podsustava za akviziciju je signal koji predstavlja zvuk iz okoline uređaja, 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 uokviriti, tj. uzimati određeni broj uzoraka, obraditi ih te opet uzeti novije uzorke. Pozadina ovakvog pristupa opisana je u poglavlju 2.2.1. u kojem je predstavljen model nastajanja govora. Naime, kako bi takav model dobro radio, potrebno je promatrati kratke isječke signala u kojima su ton i visina signala stabilni. Također, potrebno je ne uzeti svaki put cijeli okvir novih uzoraka, nego, u svrhu boljeg očuvanja informacije, ostaviti određeni broj uzoraka iz starog okvira. Na taj način, dobili smo prozor (engl. window) koji se pomiče po akviziranom signalu, tj. svaki sljedeći je jednim dijelom preklopljen preko prošlog što je prikazano na slici 2.5. Zelenom bojom je obojan prozor u iteraciji nakon prozora obojanog svijetlosmeđom bojom. Određeni dio uzoraka je isti, a određeni dio su novi uzorci dobiveni od podsustava za akviziciju.



Slika 2.5. Preklapajući prozori

U glavnoj petlji programskog koda inicijalizirano je polja koje sprema nove uzorke te polje koje zaduženo za spremanje trenutnog prozora podataka. Alokacija polja i algoritam pomicanja starih podataka u polju koje sprema trenutni prozor prikazani su u isječku programskog koda 2..4

```
1     int16_t newSamples[STEP_SIZE];
2     int16_t audioFrame[WINDOW_SIZE] = {0};
3
4     /**
5     * Sliding window with overlap. Audio frame holds WINDOW_SIZE samples.
6     * Each update shifts "Samples to Keep" to the start of the buffer,
7     * discards "Old Samples" and makes space at the end for "New Samples
      ."
8     *
9     * Before:
10    * ------------------------------------------------------------
11    * | Old Samples           | Samples to Keep                  |
12    * ------------------------------------------------------------
13    * |<----- STEP_SIZE ----->|<---- WINDOW_SIZE - STEP_SIZE ---->|
14    * ------------------------------------------------------------
15    *
16    * After:
17    * ------------------------------------------------------------
18    * | Kept Samples                       | New Samples         |
19    * ------------------------------------------------------------
20    * |<---- WINDOW_SIZE - STEP_SIZE ---->|<----- STEP_SIZE ----->|
21    * ------------------------------------------------------------
22    */
23    memcpy(audioFrame, audioFrame + STEP_SIZE, (WINDOW_SIZE - STEP_SIZE)
      * 2);
24    memcpy(audioFrame + WINDOW_SIZE - STEP_SIZE, newSamples, STEP_SIZE *
      2);
```

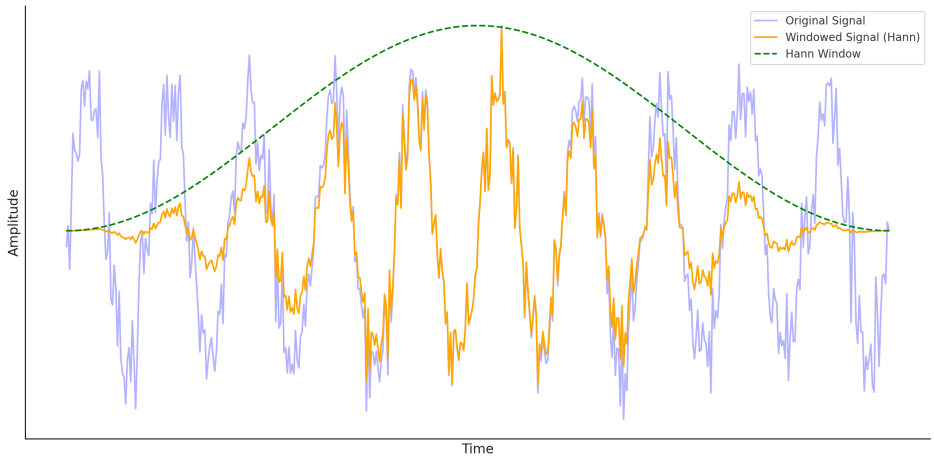
Kod 2..4: Sliding Window with Overlap

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. Prva stvar koja dolazi na red je primjena funkcije 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 sprektru ili spektralnog curenja (engl. spectral leakage). Zbog toga je potrebno pomnožiti originalni signal s funkcijom vremenskog prozora. Postoje različite funkcije koje se koriste u tu svrhu, međutim u obradi govora najčešće se koristi Hannov prozor [11]. 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 diskontinuiranost signala i pojavu nepoželjnih spektralnih komponenti.



Slika 2.6. Hannov prozor

Opisani postupak množenja originalnog signala Hannovim prozorom prikazan je matematičkim izrazom 2.4

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

gdje je:

- $x_w[n]$ signal nakon primjene Hann 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, cijelokupni signal skalirat ćemo 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 je:

- $X[k]$: spektralni koeficijent na k-toj frekvenciji,
- $x[n]$: signal u vremenskoj domeni,
- N: broj uzoraka signala,
- $e^{-j2\pi kn/N}$: osnovni eksponencijalni faktor.

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 cjelokupnog sustava 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 imena `spectrogram` jer spektralnom analizom signala dobijemo nešto što se zove spektrogram. Preciznije, spektrogram bi će biti dvodimenzionalno polje koje sadrži više ovakvih jednodimenzionalnih vektora koje dobijemo nakon Fourierove transformacije jednog prozora.

```
1    // initialization
2    dsps_fft2r_init_fc32(NULL, WINDOW_SIZE);
3    void FFT::compute(float* frame, float* spectrogram) {
4        for(size_t i = 0; i < WINDOW_SIZE; i++) {
5            data[2 * i] = frame[i];  // Real part
6            data[2 * i + 1] = 0;     // Imaginary part
7        }
8        // Perform FFT
9        dsps_fft2r_fc32_ae32(data, WINDOW_SIZE);
10       // Bit reversal
11       dsps_bit_rev_fc32_ansi(data, WINDOW_SIZE);
12       // Magnitude (power) of each frequency bin
13       for(size_t i = 0; i < NUMBER_OF_SPECTROGRAM_BINS; i++) {
14           float real = data[2 * i];
15           float imag = data[2 * i + 1];
16           spectrogram[i] = sqrt(real * real + imag * imag);
17       }
18   }
```

Kod 2..5: FFT

Melovo skaliranje

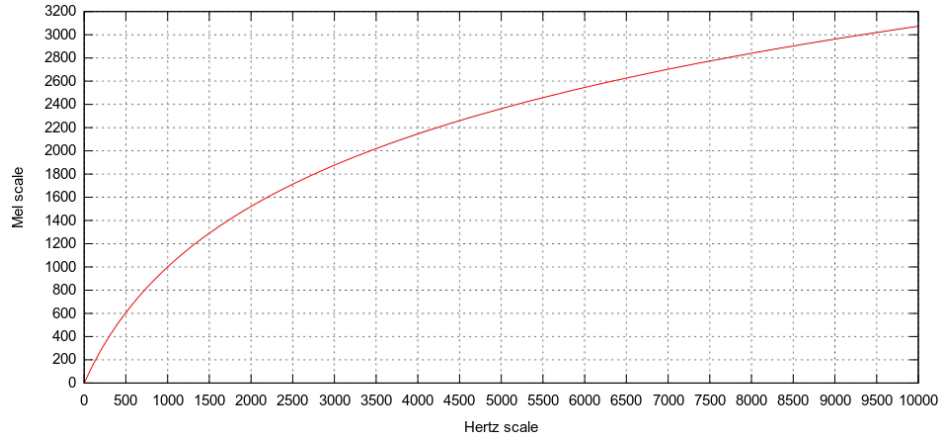
Izlaz iz modula koji se bavi Fourierovom transformacijom je frekvencijski spektar signala. Preciznije svaki element tog vektora je amplituda snage određene frekvencije sadržane u signalu. Te frekvencije su raspoređene linearno. Međutim, čovjek ne percipira jednako male promjene na višim frekvencijama isto kao na nižim. Zbog toga je potrebno na neki način skalirati razmake između frekvencija kako bi bolje odgovarali ljudskoj percepciji. Za potrebe toga osmišljena je 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 je:

- $m(f)$ frekvencija u Mel skali,
- f frekvencija u Hertzima.

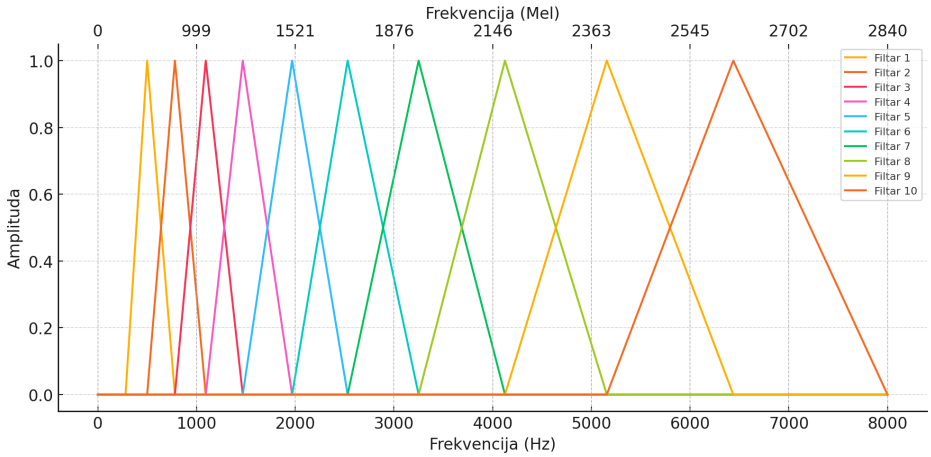
Na slici 2.7. prikazano je preslikavanje iz Hz ljestvice u Melovu. Vidljivo je da jednake promjene frekvencije u Hertzima na višim frekvencijama odgovaraju manjoj promjeni na Mel skali nego na nižim frekvencijama (logaritamsko preslikavanje), a to izravno odgovara ljudskoj percepciji koja je osjetljivija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



Slika 2.7. Melova skala [12]

Budući da je naš sustav diskretan, postoji prirodni broj frekvencijskih komponenti koje dobijemo na izlazu iz Fourierove transformacije (NUMBER_OF_SPECTROGRAM_BINS)

koji izravno ovisi o broju točaka nad kojima se vrši FFT. Sljedeći korak koji je potreban je napraviti preslikavanje iz Hz ljestvice u Melovu. Prije početka preslikavanja, potrebno je odrediti rezoluciju preslikavanja, tj. koliko spektralnih komponenti želimo 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 se odrađuje pomoću trokutaskih 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 "zacementirane" na 80 Hz i 7600 Hz (takve granice su se pokazale prikladne za ovu svrhu). Od donje do gornje granice, linearno (na Mel skali) se 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 istoga točno jedan. Nadalje, vrijednosti svake Melove frekvencijeske komponente doprinosi vrijednost frekvencijske komponente s Hz ljestvice onoliko koliko iznosi njena vrijednost pomnožena s vrijednosti filtra na tom mjestu. Za svaki filtar (koji odgovara jednoj Mel 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, potrebno je svaku vrijednost Melovih frekvencijskih komponenti 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: FFT

Diskretna kosinusna transformacija

Posljednji korak u procesu generiranja 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 najčešćee 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 našem slučaju MFCC)
- $X(n)$: ulazni Mel-frekvencijski spektar,
- N: broj uzoraka u ulaznom spektru,
- k: indeks koeficijenta (MFCC-a).

U kontekstu generiranja MFCC-a, ulaz u DCT dolazi iz Mel-frekvencijskog spektra, koji je već logaritamski skaliran kako bi pratio ljudsku percepciju glasnoće. Primjenom DCT-a na ovaj 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.
3. **Efikasnija obrada**: Kompresijom podataka DCT smanjuje zahtjeve za memorijom i procesorskom snagom (sva informacija prozora podataka sadržana je u ovih 12 ili 13 koeficijenata).

Implementacija diskretne kosinusne transformacije prikazana je u odječku koda 2..7 U konstruktoru razreda DCT inicijaliziraju se koeficijenti potrebni za izračun pojedinog MFCC-a, dok metoda compute računa MFCC-e za konkretni ulzni Mel-frekvencijski spektar. Na taj način razdvojen je izračun 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.

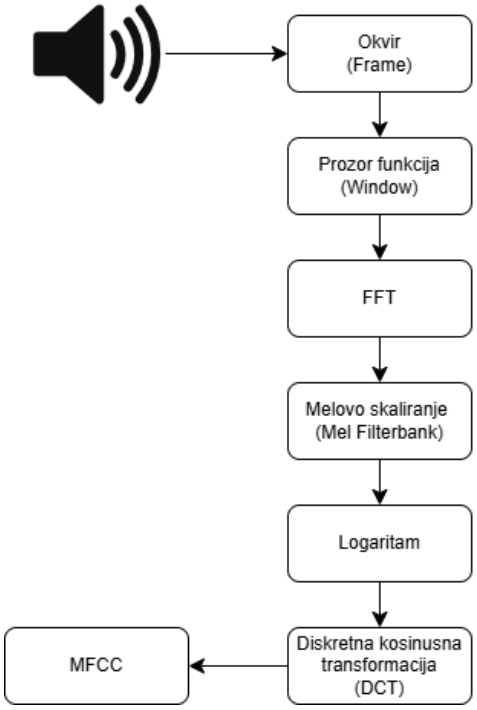
```
DCT::DCT()
{
    const float pi = 3.14159265358979323846f;
    for (int i = 0; i < NUMBER_OF_MFCCS; ++i) {
        for (int j = 0; j < NUMBER_OF_MEL_BINS; ++j) {
            coefficients[i * NUMBER_OF_MEL_BINS + j] = cos(pi * i * (j +
    0.5f) / NUMBER_OF_MEL_BINS);
        }
    }
}

void DCT::compute(const float *input, float *output) {
    for (int i = 0; i < NUMBER_OF_MFCCS; 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);
    }
}
```

Kod 2..7: FFT

Konačni rezultat cijelog procesa generiranja značajki je NUMBER_OF_MFCCS Mel kepstralnih koeficijenata. Zašto se tako zovu? Melovi su jer su bazirani na Melovoj frekvencijskoj skali koja je ubjašnjena u 2.2.2. Međutim zašto su kepstralni možda nije intuitivno. Dolazi od toga što smo tokom njihovih generiranja morali posegnuti za dvama transformacijama. Prva je bila FFT. Ona je signal iz vremenske domene prebacila u frekvencijsku (spektar signala). Druga transformacija je bila upravo DCT. Ona je još jednom računala spektar, međutim ovog putaje je to bio spektar spektra. Zbog toga su koeficijenti it ove domene dobili ime kepstralni (keps je anagram od spek). Ovo sve se može povezati s modelom govora opisanog u 2.2.1. Spektar spektra nam omogućuje razdvajanje energija u pojedinim frekvencijskim spektrima te jako dobro razdvaja odziv vokalnog trakta od kvazi-periodičnog impulsa. Također, informaciju zvučnog prozora duljine

WINDOW_SIZE komprimirali smo u svega NUMBER_OF_MFCCS Mel kepstralnih koeficijenata. Te veličine mogu varirati, a karakteristični iznosi u sustavima ovakvog tipa su redom 512 te 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 kroz cijeli vijek rada uređaja. 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 koristi konvolucijsku neuronsku mrežu čija je stuktura te područje primjene podrobnije opisano u 3.2. One dobro funkcioniraju u klasifikaciji višedimenzionalnih matrica. Ideja je primije-

niti takvu strukturu na značajke koje su prethodno generirane. Uzastopni vektori MFC koeficijenata slagani su u dvodimenzionalnu matricu koja onda može biti predana neuronskoj mreži. Možemo reći da je takvim slaganjem dobivena slika zvučnog zapisa proizvoljne dujine! Što će odrediti koliko će prozora podataka biti u jednom trenutku predano na klasifikaciju? Pa upravo duljina podataka nad kojima je mreža i trenirana! Detaljniji opis skupa podataka za treniranje mreže nalazi se u 3.3., najbitnije za reći ovje je da se radi o skupu zvučnih zapisa u trajanju od jedne sekunde. Zbog toga je potrebno napraviti matricu od onoliko prozora koliko je potrebno da pokriju vrijeme od 1 sekunde. Broj takvih prozora NUMBER_OF_TIME_SLICES će ovisiti o veličini prozora WINDOW_SIZE, broju novih uzoraka u svakoj iteraciji STEP_SIZE te frekvenciji otipkavanja SAMPLE_RATE. Funkcija 2.9 opisuje opisanu ovisnost.

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

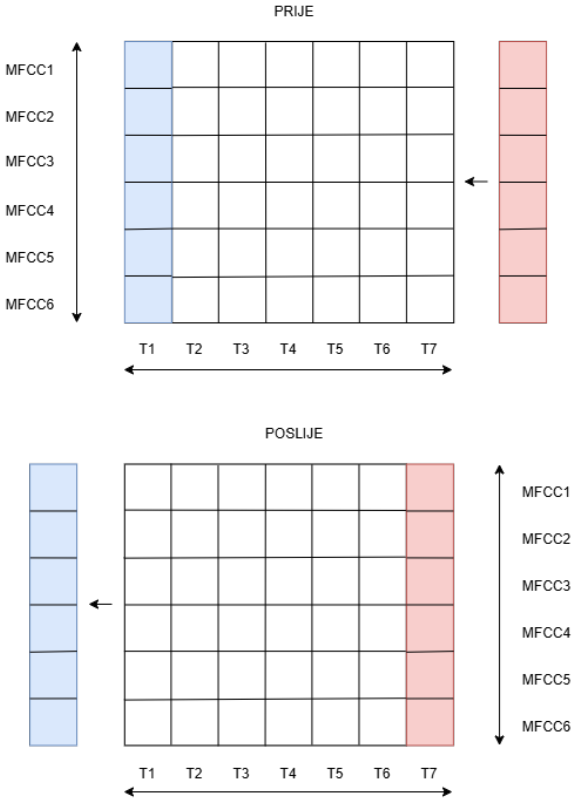
Budući da se konstantno akviziraju novi podaci te generiraju njihove značajke, a veličina dvodimenzionalne matrice koja se predaje neuronskoj mreži ostaje stalna, potrebno je dolaskom značajki novog vremenskog prozora, odbaciti najstariji, ostale pomaknuti prema početku te na kraj matrice dodati najnovije značajke. Algoritam je vrlo sličan onom koji novoakvizirane uzorke dodaje na kraj vremenskog prozora prije generiranja značajki, a prikazan je u odsječku programskog koda 2..8 U polje `featureSlice` se spremaju novonastale značajke, a polje `featureImage` sadrži trenutno dvodimenzionalno polje značajki (ovdje je je definirano kao jedodimenzionalno, međutim memorijski otisak je identičan i, ono najbitnije, takva struktura odgovara ulazu neuronske mreže). Veličina tog polja NUMBER_OF_FEATURES je samo 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));
    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. 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 obojan plavom bojom.



Slika 2.10. Matrica značajki [2]

2.3. Aktivacija neuronske mreže

Aktivacija neuronske mreže (engl. invoking) proces je u kojem se ulaznom sloju neuronske mreže (engl. input layer) predaje matrica značajki. Biblioteka koja je korištena za implementaciju neuronske mreže na mikrokontroleru je TensorFlow Lite **??** Omogućuje vrlo jednostavno korištenje treniranog modela (treniranog na računalu kako je opisano

u ... te konvertiranog u polje koje se može koristiti na mikrokontroleru kako je opisano u ...). 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::MicroInterpreter* interpreter = nullptr;
        TfLiteTensor* model_input = nullptr;
        uint8_t tensor_arena[TENSOR_ARENA_SIZE];
        tflite::MicroMutableOpResolver<N> 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

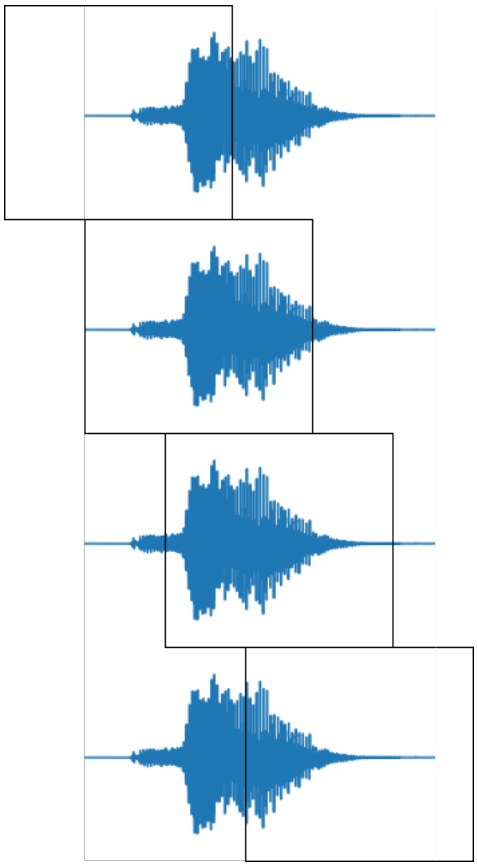
`const tflite::Model* model` predstavlja pokazivač na polje parametara modela, dok `tflite::MicroMutableOpResolver<N> resolver` predstavlja arhitekturu neuronske mreže. Toj strukturi je potrebno dodati sve vrste slojeva i funkcija koje se koriste unutar arhitekture mreže koja je trenirana. Primjer dodavanja nekih vrsta slojeva prikazan je u isječku koda 2..10 Nije bitan redoslijed dodavanja, samo je potrebno dodati sve što ta mreža sadrži.

```
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 "provlači" 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, akviziraju 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 akviziranje novih podataka te cjelokupni sustav neće dobro raditi u stvarnom vremenu. Međutim, dobra vijest je da nije potrebno aktivirati mrežu na svaki novi vremenski korak jer je on svojom duljinom puno manji od duljine uzorka kojeg predstavlja cijela matrica. Drugim riječima, uzastopne verzije ulazne matrice podataka će biti vrlo slične i možemo pričekati nekoliko novih iteracija prije nego li aktiviramo neuronsku mrežu (akviziranje signala i generiranje značajki tipično traje puno kraće od aktivacije). S druge strane, mreža se mora aktivirati dovoljno često jer u suprotnom postoji mogućnost propusta određeni 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 drugačije vrijeme odziva. Potrebno je maksimizirati broj aktiviranja mreže u određenom vremenu bez da se gube uzorci ulaznog signala.

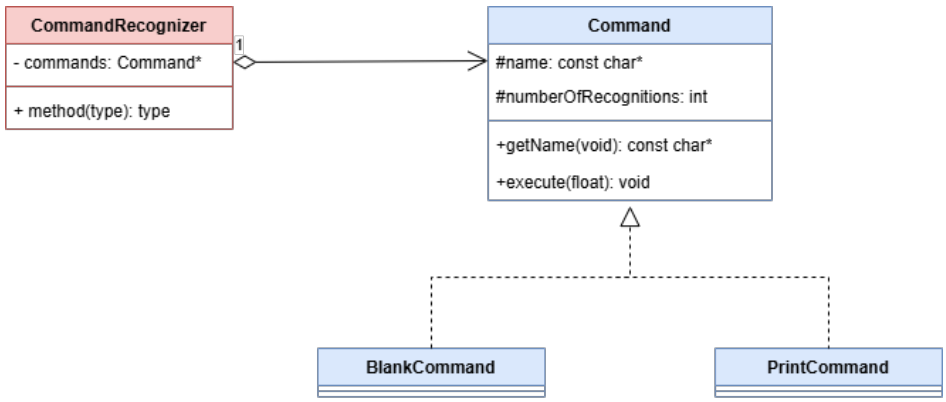


Slika 2.11. Uzastopni vremenski okviri (prozori) [2]

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 smo aktivirali mrežu rjeđe, ne bismo prepoznali naredbu (npr. prvi pa četvrti prozor).

No, takav pristup otvara mjesto drugom problemu. Ne želimo niti višestruko prepoznavanje jednom izgovorene naredbe. Tu na scenu stupa podsustav za prepoznavanje i okidanje naredbi.

2.4. Prepoznavanje naredbi i aktivacija zadatka



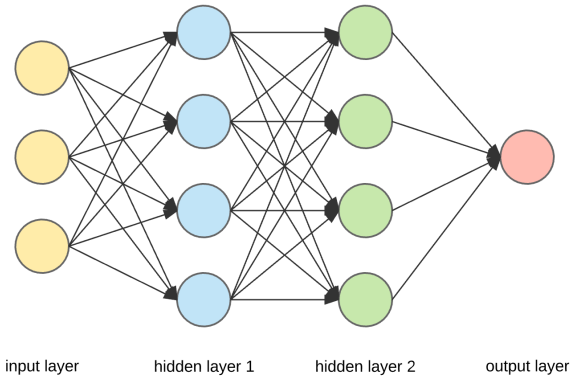
Slika 2.12. Struktura sustava [2]

3. Neuronska mreža

3.1. Općenito

Okosnica cjelokupnog sustava za prepoznavanje izgovorenih naredbi je neuronska mreža. Neuronska mreža ili preciznije umjetna neuronska mreža je računalni model inspiriran biološkom strukturom neurona u mozgu čovjeka. Predstavlja jedan od najkorištenijih modela u dubokom učenju. Sastoji se od čvorova (neurona) i jednosmjernih veza između njih (sinapsa) koji na taj način 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 potpuno povezani sloj (engl. fully connected layer ili dense layer) koji se često koristi kao osnovni građevni blok u umjetnim neuronskim mrežama [13]. 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 [13]

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 kad koristimo model neuronske mreže, gledamo na njega kao na crnu kutiju koja na ulazu prima vrijednosti te na izlazu daje vrijednosti izračunate kroz sve skrivene slojeve [14].

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 nam omogućava modeliranje složenijih nelinearnih odnosa [15]. Kada bi čvor bio modeliran bez aktivacijske funkcije svako preslikavanje koje bi činio bi bilo linearno, a zbog toga što nam svaka kompozicija linernih funkcija daje opet linearnu funkciju, cjelokupna mreža ne bi bila sposobna modelirati kompleksnije stvari. Sastavnice modela čvora su sljedeće:

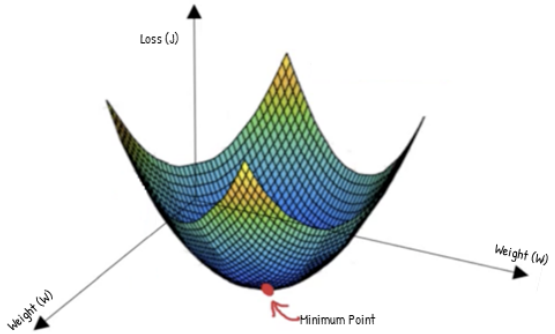
- a: izlazna vrijednost čvora (aktivacija)
- f: aktivacijska funkcija
- w_i: težina i-tog ulaznog čvora (čvor u prijašnjem sloju)
- x_i: vrijednost i-tog ulaznog čvora (njegova aktivacija)
- b: pomak
- n: broj čvorova u prijašnjem sloju koji imaju vezu s modeliranim čvorom

Povezivanjem više ovako definiranih čvorova gradimo neuronske mreže. Ulaz u neuronsku mrežu je informacija na temelju koje će na izlazu iz mreže biti vrijednost izračunata 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 se neuronska mreža "hrani" označenim skupom podataka (označeni skup prestavlja 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 izbaci određenu vrijednost koja se s očekivanom uspoređuje pomoću funckije 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 na način da njihove promjene pomaknu trenutno stanje mreže u smjeru negativne derivacije funckije 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 (eng. backpropagation) jer na izlaz pojedinog sloja utječu njegovi ulazi, tj. izlazi prijašnjeg sloja (izlaz svakog sloja je funkcija izlaza prijašenjeg sloja, a kad izračunamo derivaciju funkcije, pomjenom njenih ulaza znamo 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 funckije. Svakom iteracijom pomičemo se sve bliže minimumu, taakva vrsta optimizacije naziva se gradijentni spust.



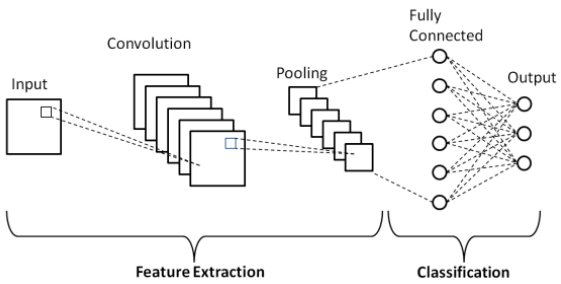
Slika 3.2. Funkcija gubitka [16]

3.2. CNN

Konvolucijska neuronska mreža (eng. Convolutional neural network, CNN) je vrsta umjetne neuronske mreže koja je pogodna je za obradu podataka s rešetkastom 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 [17]. Obrada slike koju vide sisavci funkionira hijerarhijski, tj. u mozgu se ne obrađuje cjelokupna slika odjednom, nego postoje jednostanije stanice koje su zadužene za prepoznavanje osnovnijih oblika koji se nakon toga stapaju u složenije i složenije. Na poslijetku organizam je u mogućnosti prepoznati cjelokupnu sliku koju gleda očima.

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 podataka [18]. Arhitektura najjednostavnije konvolucijske neuronske mreže uključuje:

- *Ulaz* (eng. Input Layer): ulazni sloj modela, prima matrični podatak
- *Konvolucijski sloj*: osnovni sloj modela. Njegov glavni zadatak je ekstrakcija značajki iz ulaznih podataka. Detaljnije je opisan u 3.2.1.
- *Sloj za poduzorkovanje*: smanjuje dimenzionalnost (vidi 3.2.2.)
- *Potpuno povezani sloj*: povezuje značajke s klasifikacijom (vidi 3.2.4.)
- *Izlaz* : izlazni sloj, daje vjerojatnosti klasifikacije (vidi 3.2.5.)



Slika 3.3. Jednostavna CNN [18]

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 povezanih slojeva.

3.2.1. Konvolucijski sloj

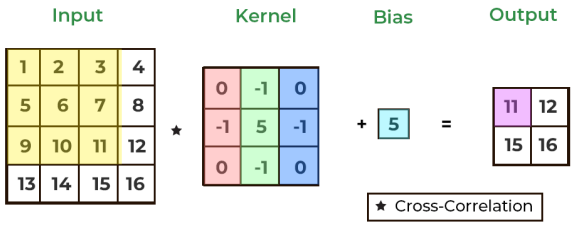
Najbitniji dio konvolucijske neuronske mreže je konvolucijski sloj zbog toga što se u njemu događa konvolucija. Konvolucija (u neuronskim mrežama) je proces kojim iz ulazne matrice podataka (slike) na izlazu dobijemo matricu značajki ili mapu značajki. Neka ulazna matrica bude oblika $x \in M_{mn}(\mathbb{R})$. Umjesto težina (kao kod potpuno povezanog sloja), konvolucijski sloj koristi matricu $\omega \in M_{pr}(\mathbb{R})$ koju nazivamo filtar ili jezgra (eng. kernel) **??** 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 **??** Na slici 3.4. prikazan je proces koji se događa tijekom prolaska ulaznog podatka kroz konvolucijski sloj. Ulaz (eng. input) se konvolucijski množi s jezgrom kako bismo dobili izlaznu matricu značajki [19]. U slučaju prikazanom na slici, ulazna slika je veličine 4×4, dok je jezgra veličine 3×3. Prvo se podmatrica ulaznog podatka veličine jednake veličini jezgre (3×3) skalarno množi s jezgrom. Izlaz je skalarni umnožak na koji se može dodati konstantna vrijednost (eng. 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 (eng. stride). U slučaju na slici 3.4. pomak iznosi jedan.



Slika 3.4. Konvolucija [20]

Rezultat opisanog procesa je mapa značajki koja je manja od ulazne, a njema veličina obrnuto proporcionalno ovisi o veličini jezgre te pomaku [21].

Slično procesu koji se odvija u mozgu čovjeka, opisani 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? Vrlo jednostavno, proces učenja (treniranja mreže) će prepoznati lokalne sličnosti između različitih primjera! Ako strukturiramo mrežu tako da se sastoji od više uzastopnih konvolucijskih slojeva, mreža će prvo naučiti najjednostavnije oblike, a zatim u sljedećem sloju takvim oblicima slagati složenije uzorke. Također, još jedna prednost konvolucijskom 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 [17]. Evo uzmimo, na primjer, dvodimenzionalnu ulaznu sliku. Broj pametara takvog dvodimenzionalnog sloja iznosit će:

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

Gdje:

- N: ukupni broj parametara
- n i m: visina i širina filtra (jezgre)
- C_{in}: Broj ulaznih kanala (npr. 1 za crno-bijele slike, 3 za RGB slike).
- $+1$: konstanta svakog filtra
- C_{out}: Broj filtara (odnosno mapa izlaznih značajki).

Kako bismo bolje dočarali razliku u broju parametara između ovakog sloja i potpuno povezanog sloja, uzmimo za primjer sliku 3.4. Ulazna slika je veličine 4×4, jezgra 3×3, a izlaz 2×2. Neka je slika jednokanalna, a broj jezgri jedan (sve kao na slici). Konvolucijski sloj će 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:

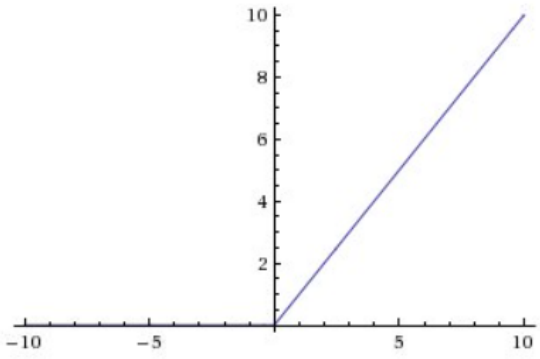
- N: ukupni broj parametara
- n_{in}: broj ulaznih neurona
- n_{out}: broj izlaznih neurona

Također, kao što je opisano u poglavlju 3. vrijednost neurona (čvora) na izlazu it sloja potrebno je provući kroz aktivacijsku funkciju. Postoje različite vrste funkcija koje se koriste u različitim granama strojnog i dubokog učenja [15], a za primjenu u konvolucijskim slojevima, najefektivnija se pokazala ReLu (eng. Rectified linear units) [22]. 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.

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

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

Prednosti ReLu funcije nad nad drugim aktivacijskim funkcijama je u tome za negativne ulaze uopće ne aktivira neuron što izuzetno povećava efikasnost. Druga prednost je u tome što izlaz nelinearno raste s porastom ulazne vrijednosti što znači da nikad neće

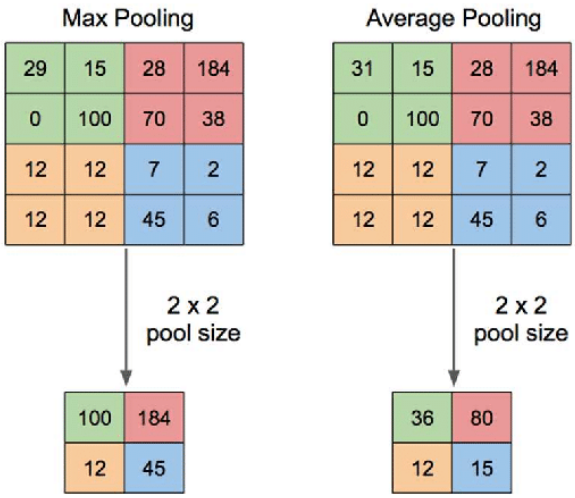


Slika 3.5. ReLu [22]

ući u zasićenje. Ta osobina je važna jer utječe na izgled funkcije gubitka te ubrzava konvergenciju gradijentnog spusta prema minimumu funkcije gubitka [15].

3.2.2. Sloj za poduzorkovanje

Sloj za poduzorkvanje (eng. Pooling layer): služi smanjenju dimenzionalnosti matrice značajki na izlazu iz konvolucijskog sloja. Najčešće korištene tehnike su maksimalno (eng. max pooling) i prosječno poduzorkovanje (eng. average pooling). Radi tako da više susjednih vrijednosti spoji u jednu te tako na svom izlazu da matricu manjih dimenzija [23]. Na taj način postupno smanjuje broj parametara, smanjuje broj operacija potrebnih za daljnje računanje te ono najbitnije, kontrolira prenaučenost [21]. Na slici 3.6. prikazane su obje navedene vrste poduzorkovanja.



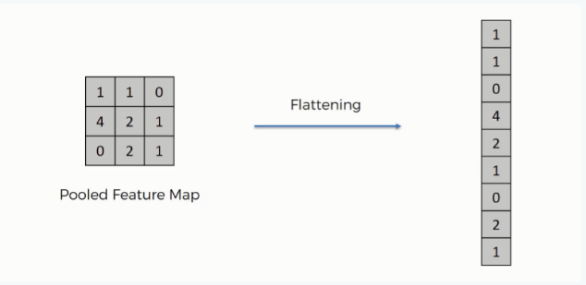
Slika 3.6. Poduzorkovanje [23]

Prosječno poduzorkovanje izglađuje sliku (eng. 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 najsvjetlije piksele iz slike, a oni su se pokazali kao najbitnije značajke jer daju najbolje rezultate [21]. Suprotno tome, minimalno uzorkovanje (eng. min pooling) odabralo bi najtamnije piksele, međutim ono se najrjeđe koristi.

3.2.3. Sloj za poravnavanje

Sloj za poravnavanje (eng. flatten layer) je sloj koji dolazi nakon posjednjeg sloja za poduzorkovanje. Njegova jedina zadaća je poravnati izlaz iz prijašnjeg sloja. Ovaj sloj ništa ne računa, ništa ne uči, jedina zadaća mu je od ulaznih mapa značajki napraviti jedan vektor koji je onda moguće povezati na potpuno povezani sloj. Zbog toga se često izostavi 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 [24]

3.2.4. Potpuno povezani sloj

Potpuno povezani sloj (eng. 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, nemamo problem sa spajanje 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

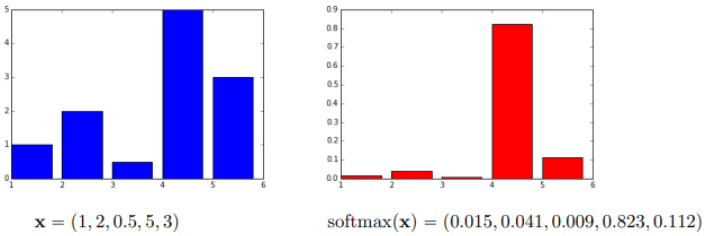
Izlazni (eng. Output Layer) je posljednji sloj potpuno povezanog sloja (a ujedno i cijele neuronske mreže). Ima onoliko neurona koliko želimo imati klasa, pojedina vrijednost neurona predstavlja vjerojatnost pripadnosti klasi. Da bi to stvarno funkcioniralo na takav način, potrebno je odrediti prikladnu aktivacijsku funkciju. Funkcija koja radi baš to 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 [25].



Slika 3.8. Softmax [25]

3.2.6. 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. Određene arhitekture su kroz povijest ostale zapamćene zbog

toga kako su utjecale na duboko učenje[26]:

- **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 3×3 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 (eng. keyword spotting), savršen otvoreni skup podataka je [27]. 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 pravom svijetu. 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 za koji će sustav uspješno klasificirati izgovorenu riječ. Cjelokupni proces pripreme podataka, treniranja i validacije modela popraćen je Jupyter bilježnicom koja se nalazi u GitHub repozitoriju [28].

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. Iz toga proizlazi da jedna od kategorija (klasa) na kojoj će model biti treniran jest pozadinska buka. 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 (eng. white noise) 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 [29]. Međutim, zbog toga što skup podataka zadrži svega nekoliko minuta takvih zvučnih zapisa, potrebno je na neki način dopuniti taj podskup podataka. Naime mreža koju treniramo će preferirati klasu neku od klasa ako takvih primjera ima mnogo više od primjera ostalih klasa. Drugim riječima, skup podataka za treniranje mora biti balansiran (primjera iz svake klase treba biti otprilike jednak broj) [30]. Uz to, za rad u stvarnom svijetu, nije loše u skup dodati zvučni zapis snimljen upravo na sustavu koji će i akvizirati podatke iz okoline. Uz snimke pozadinskih zvukova iz skupa podataka, dodane su snimke snimljenje upravo na sustavu gdje će sve biti implementirano, a zadrže karakteristične pozadinske zvukove okruženja u kojem će se nalaziti uređaj. Budući da su svi zvučni zapisi riječi u skupu podataka duljine od otprilike jedne sekunde, pozadinske zvukove je potrebno izrezati 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 trebamo imati isječaka koji će predstavljati klasu pozadinskih zvukova. Metoda kojoj lako možemo "umnožiti" broj pozadinskih zvukova zove se augmentacija.

AUGMENTACIJA...

Druga kategorija mora biti sastavljena od kombinacije različitih riječi za koje ne želimo klasifikaciju, tj. nisu odabrane u podskup naredbi. Ostatak kategorija bit će odabrane riječi što znači da će ukupan broj klasifikacijskih kategorija biti za dva veći od broja odabranih naredbi.

ODABRANE NAREDBE.....

broj zapisa u svakoj klasi....

podjela na trenig/valid(test)

3.5. Struktura mreže

3.6. Treniranje modela

3.7. Vrednovanje modela

3.8. Convert

4. Zaključak

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

[7] 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

[8] 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

[9] 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

[10] 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

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

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

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

[14] 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.].

[15] 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.

[16] 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.].

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

[18] 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

[19] 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.].

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

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

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

[23] 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

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

[25] 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.

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

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

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

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

[30] “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.].

Sažetak

Sustav za prepoznavanje govornih naredbi u stvarnom vremenu na rubnim uređajima

Luka Balić

Unesite sažetak na hrvatskom.

Ključne riječi: prva ključna riječ; druga ključna riječ; treća ključna riječ

Abstract

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

Luka Balić

Enter the abstract in English.

Keywords: the first keyword; the second keyword; the third keyword

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 audio zapisa različitih riječi u skupu [27]

Privitak B: Configuration.hpp