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 [**?**]. 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 optimizirane modele strojnog učenja. Takav procesor bi konstantno akvizirao 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 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

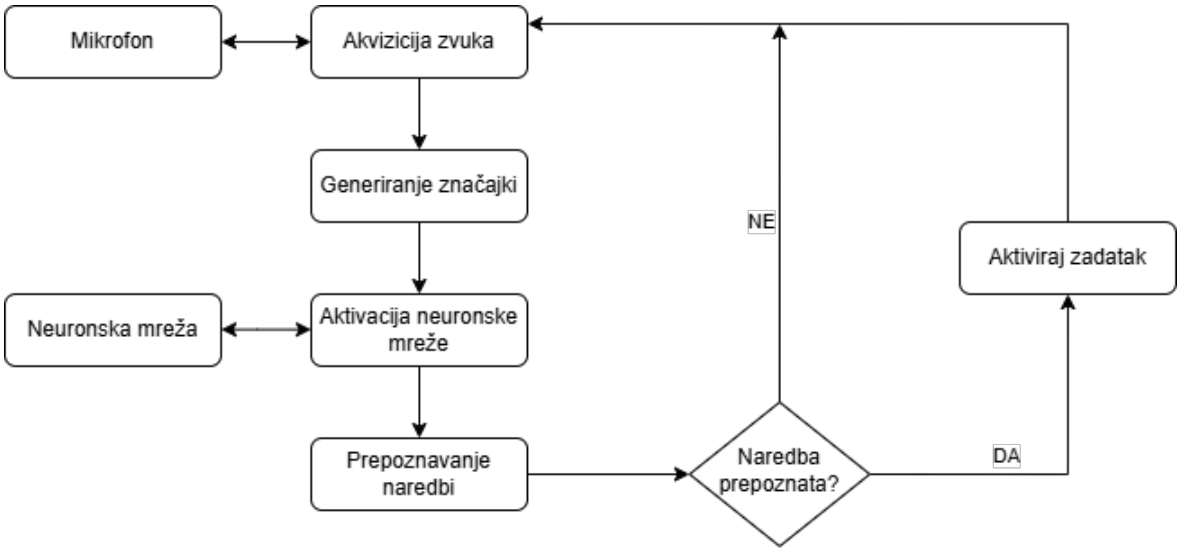
Svrha cjelokupnog sustava je okidanje obavljanja određenog procesa, tj. signalizacija za početak obavljanja nekog posla glasom. Također, sustav mora biti sposoban pokrenuti proces u bilo kojem trenutku, a odaziv bi trebao biti trenutačan što znači da cijeli proces prikupljanja zvuka, obrade te izvršavanja mora raditi kontinuirano u stvarnom vremenu. Srce sustava, tj. dio koji je zadužen za samo prepoznavanje određene naredbe je neuronska mreža trenirana na skupu zvučnih zapisa koji sadrže željene naredbe, a čija je struktura detaljnije objašnjena u poglavlju o neuronskoj mreži **??** Shodno tome, sustav je izgrađen od četiri modularna podsustava implementirana na mikrokontroleru:

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

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

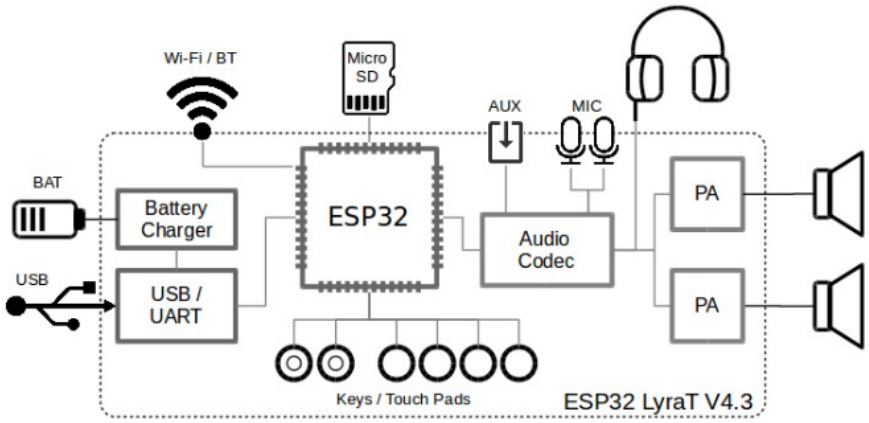
2.1. Akvizicija zvuka

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 [**?**], na kojem je sustav implementiran, na sebi ima već ugrađen mikrofon i audio kodek s kojim je moguće ko-



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

municirati putem I2S protokola. Na slici **??** prikazan je blok dijagram ESP32 razvojne platforme.



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

Najvažniji dio razvojne platforme je upravo ESP32-WROVER-E mikrokontroler na kojem je cjelokupni sustav za prepoznavanje govornih naredbi implementiran. Upravo on je zadužen za komunikaciju s ES8388 audio kodekom [**?**]. 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 **??** 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 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 protočni akvizicijski sustav (engl. *pipeline*), a to je prikazano u programskom kodu **??**

```
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 sustava

Frekvencija otipkavanja postavljena u konfiguraciji iznosi 16000 Hz što je dovoljno za ovakav tip sustava [**?**]. Nakon pokretanja akvizicijskog podsustava, sve što je preostalo

je uzimati nove podatke s kraja protočne strukture podsustava. 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 **??** 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 glavnoj petlji 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 no-

vih 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 pisanjem novog razreda koji prati zadano sučelje.

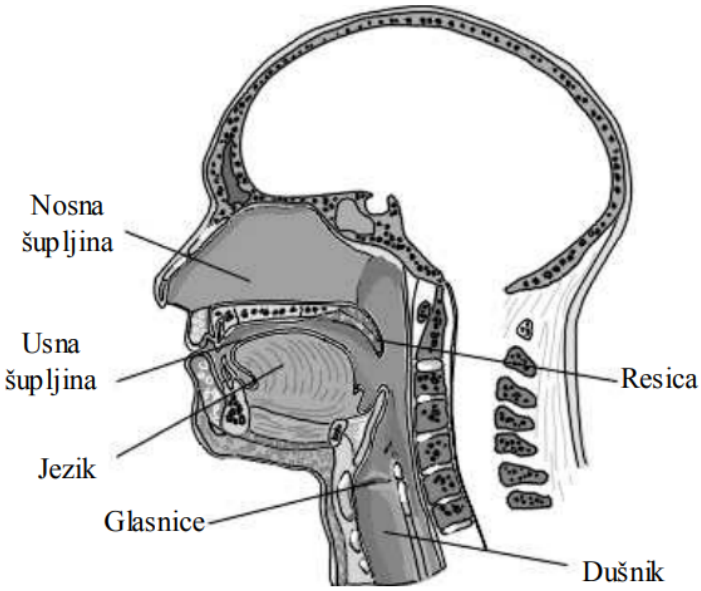
2.2. Generiranje značajki

Generiranje značajki je drugi podsustav spomenut u uvodnom dijelu, također prikazan na slici **??** 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 određivanju 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. Međutim, 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 **??**

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-



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

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) [**?**].

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 slučajnim šumom (engl. *random noise signal*). Opisani sustav može se prikazati jednadžbom **??**

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

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 **??** Množenju u frekvencijskoj domeni istovjetna je konvolucija u vremenskoj (i obratno!).

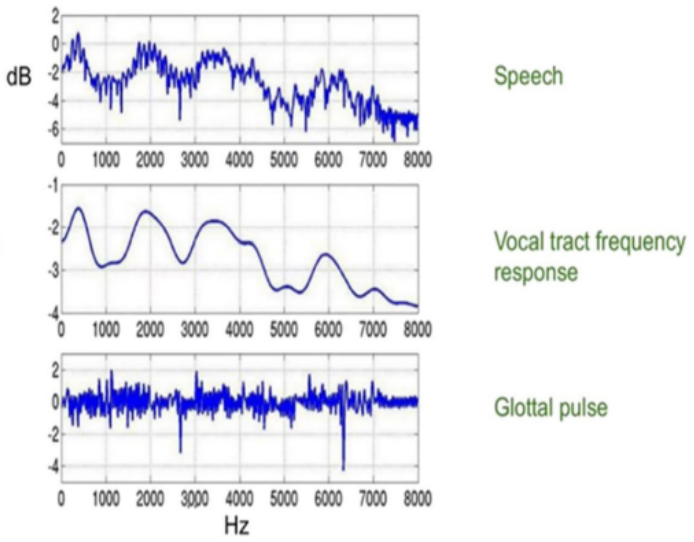
$$x(t) = e(t) * h(t) \tag{2.2}$$

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 konstantna tijekom govora (veliko pojednostavljenje, međutim svi modeli na kojima se temelji ovo područje zasnivaju na ovoj činjenici [**?**, **?**, **?**]). Zbog toga želimo pronaći način za razdvajanje tih dvaju elemenata modela. Ako primijenimo logaritamsku funkciju na sustav opisan u frekvencijskoj domeni proizlazi **??**

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

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

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



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

se koriste Mel kepstralni koeficijenti.

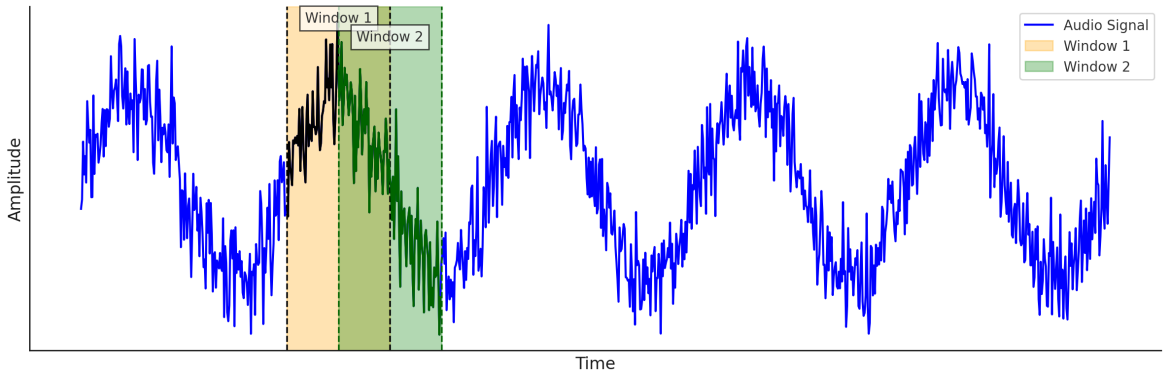
2.2.2. Mel kepstralni koeficijenti

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 [**?**]. 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 s 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 **??** 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 **??** 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. Tako smo dobili okvir ili prozor (engl. *window*) koji se pomiče po akviziranom signalu, tj. svaki sljedeći je jednim dijelom preklopljen preko prošlog što je prikazano na slici **??** Zelenom bojom je obojen prozor u iteraciji nakon prozora obojenog svijetlo smeđom 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 polje 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 **??**

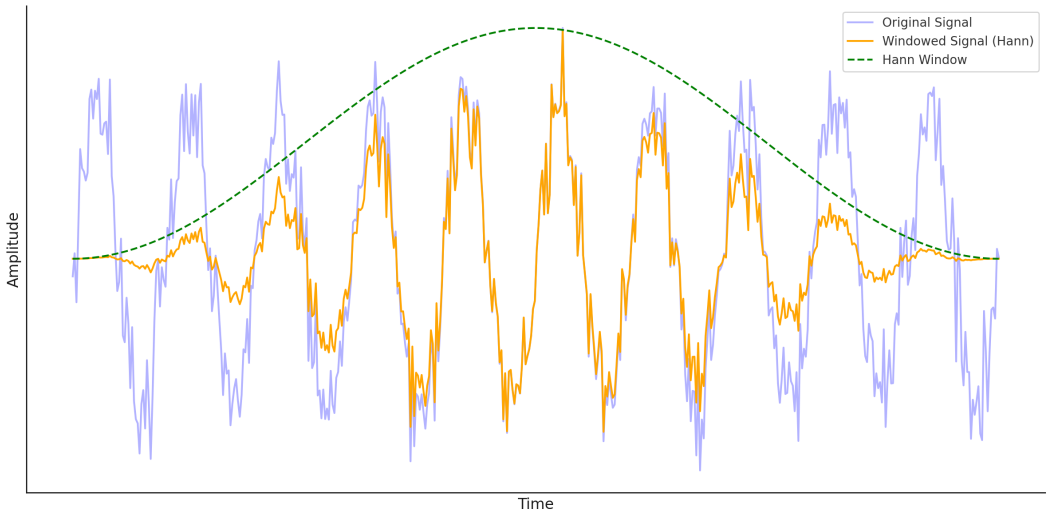
```
/* polje za nove uzorke */
int16_t newSamples[STEP_SIZE];

/* polje za trenutni prozor */
int16_t audioFrame[WINDOW_SIZE] = {0};

/* pomicanje starih podataka prema pocetku aktivnog prozora */
memcpy(audioFrame, audioFrame + STEP_SIZE, (WINDOW_SIZE - STEP_SIZE)*2);

/* stavljanje novih podataka na kraj aktivnog prozora */
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 **??** .

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 **??** 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, međutim u obradi govora najčešće se koristi Hannov prozor (otvor) [**?**]. Na slici **??** 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 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 **??**

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

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, cjelokupni 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 **??**

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

gdje je:

- $X[k]$: spektralni koeficijent na k-toj frekvenciji,

- $x[n]$: signal u vremenskoj domeni,
- N: broj uzoraka signala,

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

```
dsps_fft2r_init_fc32(NULL, WINDOW_SIZE);  // initialization
void FFT::compute(float* frame, float* spectrogram) {
    for(size_t i = 0; i < WINDOW_SIZE; i++) {
        data[2 * i] = frame[i];  // Real part
        data[2 * i + 1] = 0;     // Imaginary part
    }
    dsps_fft2r_fc32_ae32(data, WINDOW_SIZE); // Perform FFT
    dsps_bit_rev_fc32_ansi(data, WINDOW_SIZE); // Bit reversal
    // Magnitude (power) of each frequency bin
    for(size_t i = 0; i < NUMBER_OF_SPECTROGRAM_BINS; i++) {
        float real = data[2 * i];
        float imag = data[2 * i + 1];
        spectrogram[i] = sqrt(real * real + imag * imag);
    }
```

```
15      }
```

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 **??**

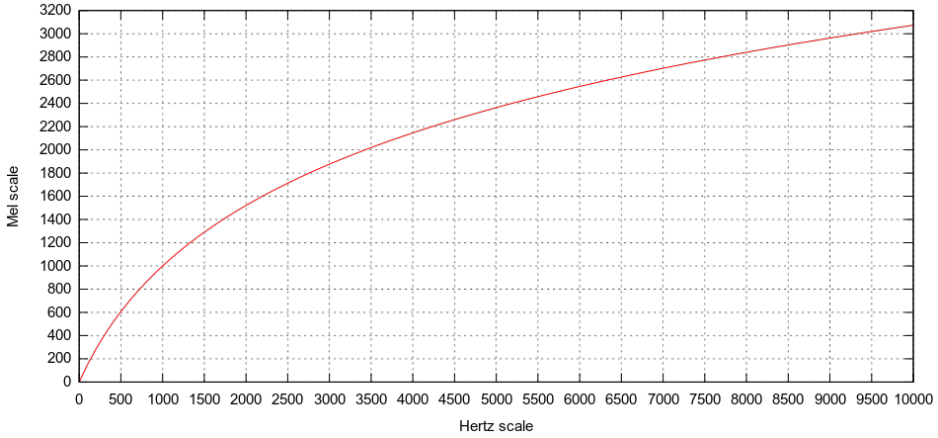
$$m(f) = 2595 \cdot \log_{10}\left(1 + \frac{f}{700}\right) \tag{2.7}$$

gdje je:

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

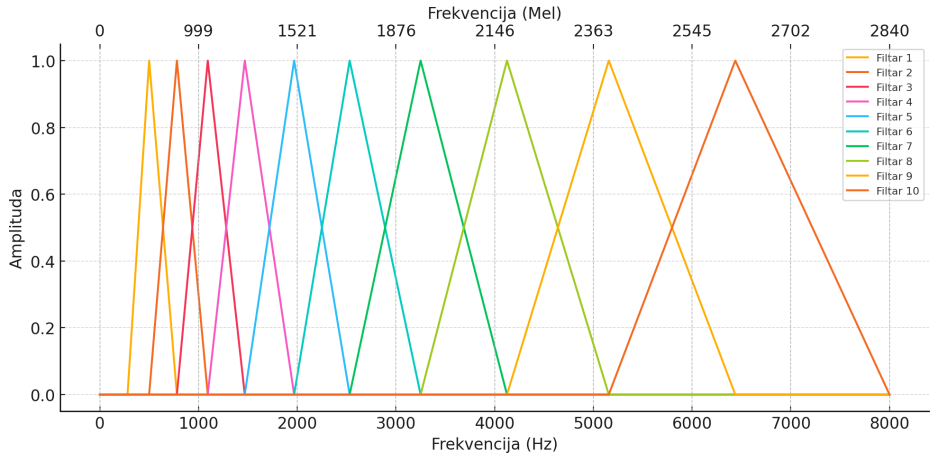
Na slici **??** prikazano je preslikavanje iz Hertz ljestvice u Melovu. Vidljivo je da jednake promjene frekvencije u Hertzima na višim frekvencijama odgovaraju manjoj promjeni na Mel skali nego na nižim frekvencijama (logaritamsko preslikavanje), a to izravno odgovara ljudskoj percepciji koja je osjetljivija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

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 Hertz 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 dato-



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

teci `Configuration.hpp`, a zove se NUMBER_OF_MEL_BINS (u prijevodu broj Melovih kanti). Samo preslikavanje se odrađuje pomoću trokutastih filtara prikazanih na slici **??**



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 i najčešće su korištene u sustavima ovakvog tipa. Naime, informacijski najbogatije komponente se obično nalaze između ove dvije granice. Između njih linearno se (na Mel 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 istoga točno jedan. Nadalje, vrijednosti svake Melove frekvencijske komponente doprinosi vrijednost frekvencijske komponente s Hz ljestvice onoliko koliko iznosi njena vrijednost pomnožena s vrijednošću 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 **??** 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 do-

mene 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 **??**

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

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).

Primjenom DCT-a na ovaj 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 **??** 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 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. Rezultat DCT-a algoritma je vektor prvih NUMBER_OF_MFCCS MFC koeficijenata izuzev prvog. Prvi koeficijent se obično odbacuje

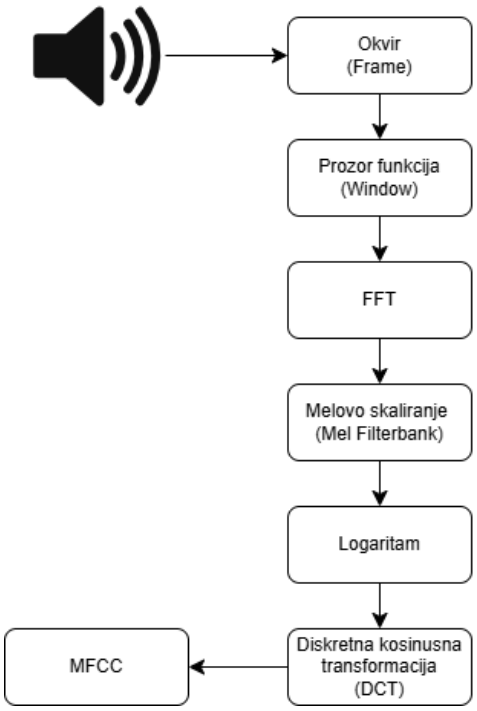
jer sadrži samo energiju signala, a ne nosi dodatne informacije o spektru signala.

```
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: FFT

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 objašnjena u **??** Međutim zašto su kepstralni možda nije intuitivno. Dolazi od toga što smo tijekom 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 puta je to bio spektar spektra. Zbog toga su koeficijenti iz ove domene dobili ime kepstralni (keps je anagram od spek). Ovo sve se može povezati s modelom govora opisanog u **??** Spektar logaritamskog spektra nam omogućuje razdvajanje energija u pojedinim frekvencijskim spektrima te jako dobro razdvaja odziv

vokalnog trakta od kvazi-periodičnog impulsa kao što je prikazano na slici **??** 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 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 **??**



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

2.2.3. Konstrukcija matrice značajki

U poglavlju **??** opisana je konstrukcija MFC koeficijenata iz jednog okvira zvučnog zapisa tipične duljine od oko 20 do 30 ms [**?**]. Na način koji je opisan u **??** 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 struktura te područje primjene podrobnije opisano u **??** 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 onda može biti predana neuronskoj mreži. Možemo reći da je takvim slaganjem dobivena slika zvučnog zapisa proizvoljne duljine. 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 **??**, najbitnije za reći ovdje 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 **??** prikazuje opisanu ovisnost.

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

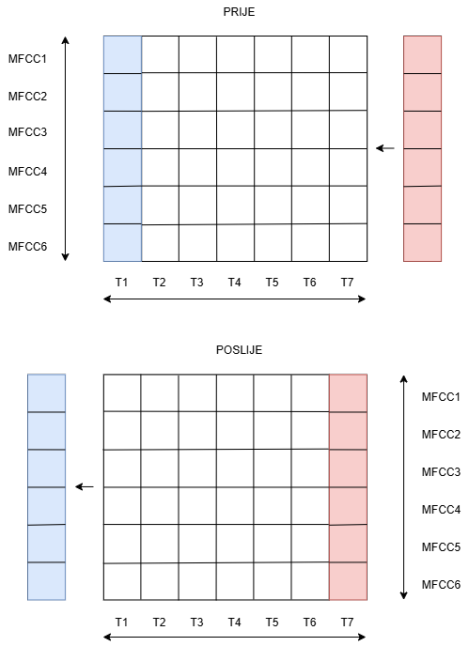
Budući da se konstantno akviziraju novi podaci te generiraju njihove značajke, a veličina dvodimenzionalne matrice koja se predaje neuronskoj mreži ostaje stalna, potrebno je dolaskom značajki novog vremenskog prozora, odbaciti najstariji, ostale pomaknuti prema početku te na kraj matrice dodati najnovije značajke. Algoritam je vrlo sličan onom koji novoakvizirane uzorke dodaje na kraj vremenskog prozora prije generiranja značajki, a prikazan je u odsječku programskog koda **??** U polje `featureSlice` se spremaju novonastale značajke, a polje `featureImage` sadrži trenutačno dvodimenzionalno polje značajki (ovdje je je definirano kao jednodimenzionalno, 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));
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 **??** 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 [**?**]

2.3. Aktivacija neuronske mreže

Aktivacija neuronske mreže (engl. *invoking*) 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 [**?**]. Omogućuje vrlo jednostavno korištenje treniranog modela (treniranog na računalu kako je opisano u poglavlju o treniranju neuron-

ske mreže **??** te konvertiranog u polje koje se može koristiti na mikrokontroleru kako je opisano u dijelu o prilagodbi **??**). 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 **??**

```
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 procesuirala 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 **??** Nije bitan redoslijed dodavanja, samo je potrebno dodati sve što ta mreža sadrži. U suprotnom, inicijalizacija modela neće uspjeti.

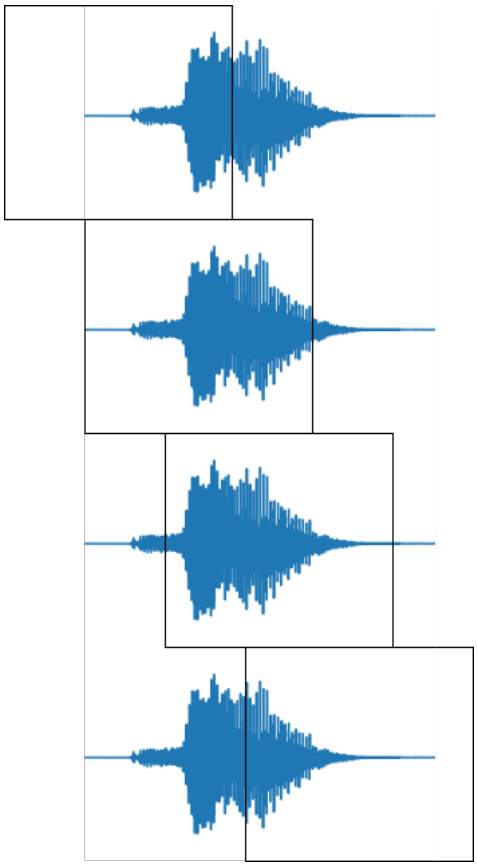
```
resolver.AddConv2D()            // konvolucijski sloj
resolver.AddFullyConnected()    // potpuno povezani sloj
resolver.AddSoftmax()           // softmax funkcija
resolver.AddMaxPool2D()         // sloj za poduzorkovanje
```

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 "provlači" vrijednosti kroz svoju strukturu te na izlazu daje vjerojatnosti da određeni ulaz (matrica značajki) predstavlja naučenu naredbu. Kako je objašnjeno u **??**, 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 **??** 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 bit će 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 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 drugačije vrijeme odziva. Potrebno je maksimizirati broj aktiviranja mreže u određenom vremenu bez gubitaka uzoraka ulaznog signala.

Na slici **??** 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 pre-



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

poznavanje 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).

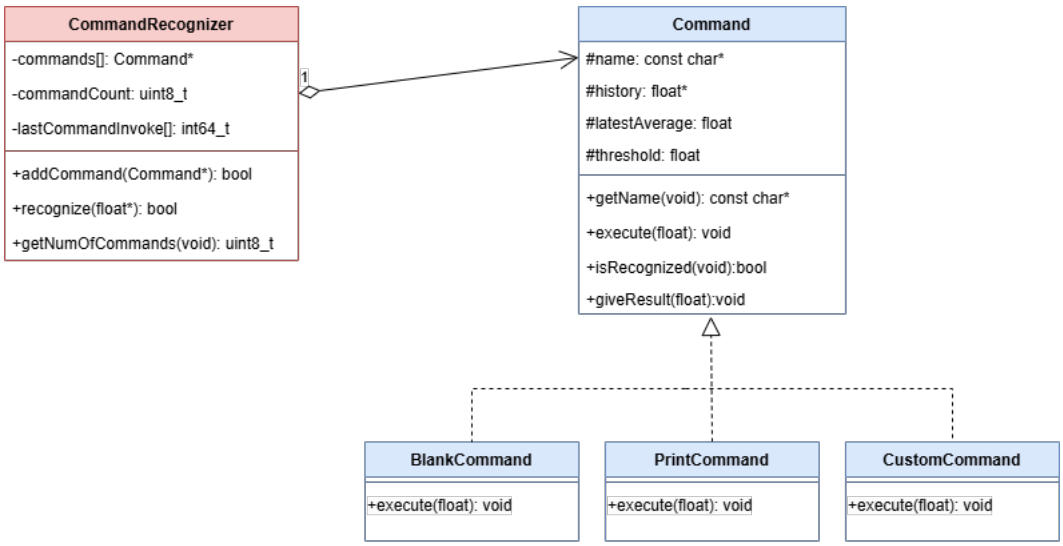
No, takav pristup otvara mjesto drugom problemu. Ne želimo niti višestruko prepoznavanje jednom izgovorene naredbe. Tu na scenu stupa podsustav za prepoznavanje i okidanje zadatka povezanog s određenom naredbom.

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, a to je da se određeni posao (zadatak) aktivira isključivo jednom za jednom izgovorenu na-

redbu. Još jedna stvar na koju treba pripaziti je slučajno aktiviranje nekog posla jer zbog nesavršenosti mreže se može dogoditi da vjerojatnosni izlaz ukazuje na prepoznavanje neke naredbe iako se u stvarnosti nije izgovorila ista. Međutim, u takvim situacijama se očekuje možda jedna ili dvije takve situacije zaredom, a ne 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 **??** 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[**?**]

Prilikom konstrukcije objekta razreda koji implementira sučelje `Command` potrebno je

postaviti parametre za prepoznavanje naredbe (oni se utvrđuju eksperimentalno). Parametrom `historySize` bira se broj uzastopnih vrijednosti vjerojatnosti pojave naredbe nad kojima se računana 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 (potrebno je zbog podataka nad kojima se mreža uči te se zbog toga može 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 **??** Jednom konstruirani objekt naredbe potrebno je dodati u objekt klase CommandRecognizer metodom `bool addCommand(Command* command)` kako bi on bio svjestan postojanja 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 cjelokupni sustav. Ideja je da takva naredba samo pokrene duži posao za koji će onda biti zadužen neki drugi sustav.

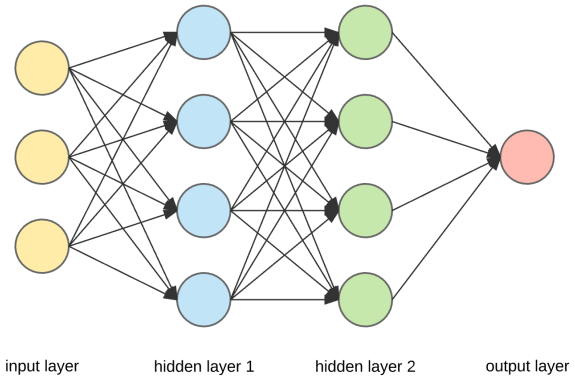
U ovom trenutku opisana je cjelokupna petlja mikrokontrolerskog sustava koja se neprestano iznova izvršava (slika **??**). Jedina stvar koja je dosad uzimana kao "crna kutija" je sama neuronska mreža koja je u biti okosnica cijelog sustava. Sljedeće poglavlje posvećeno je upravo njoj.

3. Neuronska mreža

3.1. Općenito

Neuronska mreža ili preciznije umjetna neuronska mreža (engl. *neural network*) 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 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 [**?**]. 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 [**?**]

Na slici **??** 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 [**?**].

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 (**??**) 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 [**?**]. Kada bi čvor bio modeliran bez aktivacijske funkcije svako preslikavanje koje bi činio bi bilo linearno, a zbog toga što nam svaka kompozicija linearnih funkcija daje opet linearnu funkciju, cjelokupna mreža ne bi bila sposobna modelirati kompleksnije stvari. Sastavnice modela čvora su sljedeće:

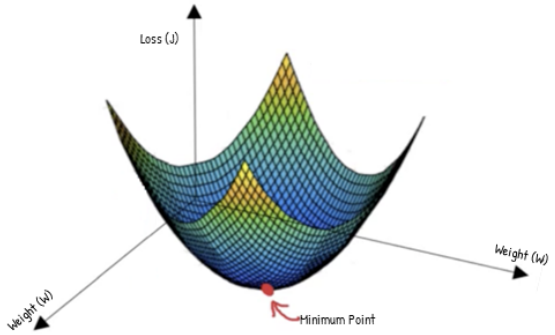
- a: izlazna vrijednost čvora (aktivacija)
- f: aktivacijska funkcija
- w_i: težina i-tog ulaznog čvora (čvor u prijašnjem sloju)
- x_i: vrijednost i-tog ulaznog čvora (njegova aktivacija)
- b: pomak
- n: broj čvorova u prijašnjem sloju koji imaju vezu s modeliranim čvorom

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 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 se neuronska mreža "hrani" označenim skupom 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 izbaci 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 (izlaz svakog sloja je funkcija izlaza prijašnjeg sloja, a kad izračunamo derivaciju funkcije, promjenom njenih ulaza znamo u kojem smjeru će se mijenjati vrijednost same funkcije).

Bez smanjenja općenitosti, funkcija gubitka prikazana je na trodimenzionalnom grafu na slici **??** 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 pomičemo se sve bliže minimumu, a takva vrsta optimizacije naziva se gradijentni spust (engl. *gradient descent*).



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

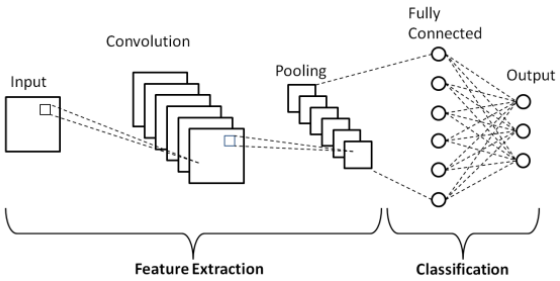
3.2. Konvolucijska neuronska mreža

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 [**?**]. 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 i složenije. Na posljetku organizam može prepoznati cjelokupnu sliku koju gleda očima. Međutim, kakve veze ima klasifikacija slika s problemom prepoznavanja glasovnih naredbi? Naime, generirana matrica značajki iz poglavlja **??** 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 [**?**]. 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 **??**
- *Sloj za poduzorkovanje*: smanjuje dimenzionalnost (vidi **??**)
- *Sloj za poravnavanje*: spaja konvolucijski i potpuno povezani dio (vidi **??**)
- *Potpuno povezani sloj*: povezuje značajke s klasifikacijom (vidi **??**)
- *Izlaz*: izlazni sloj, daje vjerojatnosti klasifikacije (vidi **??**)

Na slici **??** 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.



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

3.2.1. Konvolucijski sloj

Najbitniji dio konvolucijske neuronske mreže je konvolucijski sloj (engl. *convolutional layer*) 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:

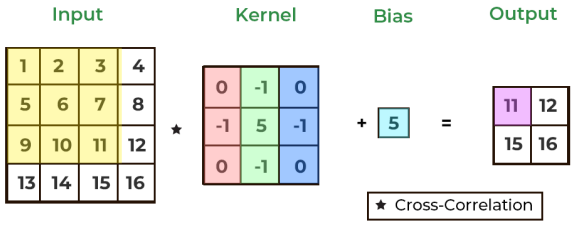
$$h = \omega * x, \tag{3.2}$$

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 **??** 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 [**?**]. 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 **??** pomak iznosi jedan.



Slika 3.4. Konvolucija [**?**]

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

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? 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 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 [**?**]. 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 ovakvog sloja i potpuno povezanog sloja, uzmimo za primjer sliku **??** 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:

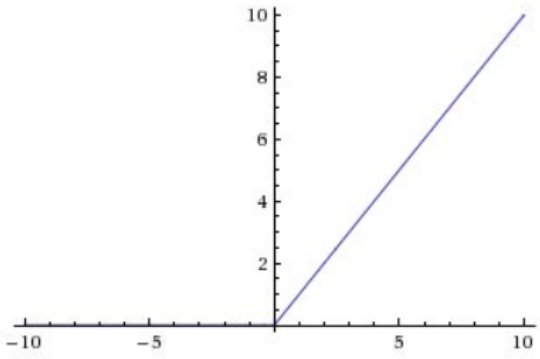
- N: ukupni broj parametara
- n_{in}: broj ulaznih neurona
- n_{out}: broj izlaznih neurona

Također, kao što je opisano u poglavlju **??** 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 [**?**], a za primjenu u konvolucijskim slojevima, najefektivnija se pokazala ReLu (eng. *Rectified linear units*) [**?**]. 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 **??** i **??**, a prikazana je na slici **??**

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

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

Prednosti ReLu funkcije nad drugim aktivacijskim funkcijama je u tome što za nega-



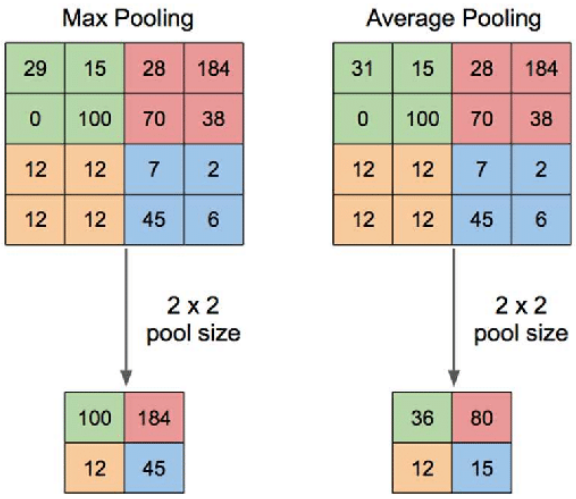
Slika 3.5. ReLu [**?**]

tivne 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 [**?**].

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 [**?**]. Na taj način postupno smanjuje broj parametara, smanjuje broj operacija potrebnih za daljnje računanje te ono najbitnije, kontrolira prenaučenost [**?**]. Na slici **??** prikazane su obje navedene vrste poduzorkovanja.

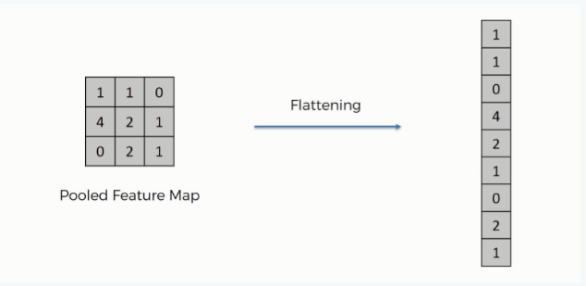
Prosječno poduzorkovanje izglađuje sliku (engl. *smoothing*). Zbog toga oštri detalji slike mogu biti izgubljeni što znači da se određene značajke možda neće prepoznati kada se koristi ova metoda. Maksimalno uzorkovanje odabire piksele s najvećom vrijednošću iz slike, a oni su se pokazali kao najbitnije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



Slika 3.6. Poduzorkovanje [**?**]

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, 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 **??** Na slici **??** prikazana je uloga ovog sloja.



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

3.2.4. Potpuno povezani sloj

Potpuno povezani sloj (engl. *fully connected layer*) detaljnije je pojašnjen u poglavlju **??** 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 tije-

kom 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

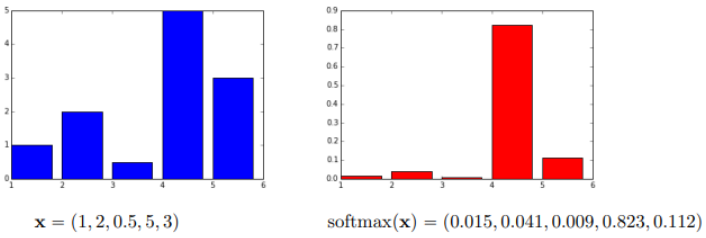
Izlazni (engl. *output layer*) je posljednji sloj potpuno povezanog sloja (a ujedno i cijele neuronske mreže). Ima onoliko neurona koliko želimo imati klasa, pojedina vrijednost neurona predstavlja vjerojatnost pripadnosti klasi. Da bi to stvarno funkcioniralo na takav način, potrebno je odrediti prikladnu 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 [**?**].



Slika 3.8. Softmax [**?**]

3.2.6. Regularizacija

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 implementiran 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. Određene arhitekture su kroz povijest ostale zapamćene zbog toga kako su utjecale na duboko učenje[**?**]:

- **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 (engl. *keyword spotting*), savršen otvoreni skup podataka je skup za treniranje prepoznavanja ograničenog skupa govornih naredbi [**?**]. 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 **??**

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 [**?**].

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 raditi u stvarnom svijetu, jedna od kategorija (klasa) na kojoj model mora biti treniran jest 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 [**?**]. Takvi zvukovi dobro predstavljaju tipičnu okolinu u kojoj bi se sustav mogao nalaziti. 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 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) [**?**]. 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. 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 kojom lako možemo "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 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 okretanjem uzoraka u snimci (nova snimka je obrnuta od originala). Funkcija za augmentaciju prikazana je u odsječku programskog koda **??**

```
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 želimo klasifikaciju, tj. nisu odabrane u podskup naredbi. Ime te kategorije će biti "nepoz-

nato" (engl. *unknown*), a predstavljat će sve riječi koje nisu naredbe cjelokupnog sustava. Ova kategorija je nužna za rad sustava jer 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 **??**

```
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 va-

lidacijski skup. Skup za treniranje koristi se, kao što mu ime kaže, za treniranje neuronske mreže, dok se validacijskim skupom nakon svake epohe (pojam epoha je objašnjen u poglavlju **??**) provjerava točnost modela, podešavaju hiperparametri i sprječava prenaučenost (te podatke model nije "vidio" tijekom treniranja). 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 tokom 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 generirati značajke za svaki zvučni zapis. Detaljni opis generiranja značajki objašnjen je u **??** U dodatku **??** prikazano je generiranje 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 **??** prikazani su valni oblici nasumičnih zvučnih zapisa odabranih kategorija te pripadna matrica značajki generirana na spomenuti način.

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 **??** Uz to, postoji zahtjev za što manjim modelom jer je isti potrebno implementirati na mikrokontrolerskoj platformi koja ima ograničavajuće memorijske resurse. Također, vrijeme potrebno 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, izgradit ćemo mrežu dovoljno jednostavnu da pokrije spomenute uvjete, a s druge strane dovoljno složenu, 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 (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 **??** 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 **??** 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 [**?**]

3.6. Treniranje i vrednovanje modela

Nakon definiranja strukture modela neuronske mreže, na red je došlo treniranje. Podaci su u ovom trenutku podijeljeni u trenažni, 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 **??** 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 u nastavku **??**

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

gdje:

- L je gubitak.

- N je broj uzoraka.
- y_i je oznaka primjera (klasa).
- $p(y_i)$ vjerojatnost predikcije za ispravnu klasu.

Nakon prolaska BATCH_SIZE (u našem 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 našem slučaju 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 **??** 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 (nije nužno stanje nakon posljednje odrađene epohe).

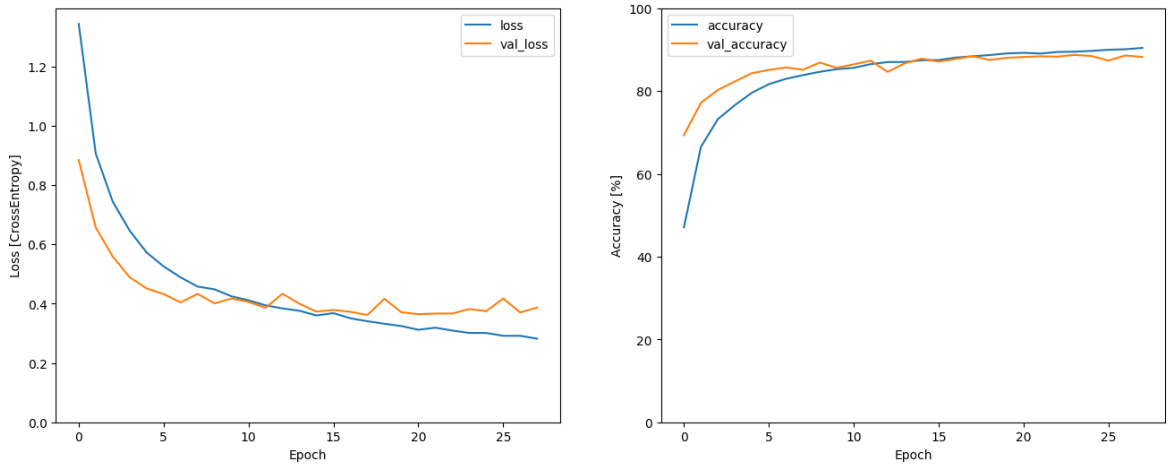
```
history = model.fit(
    train_mfcc_dataset,
    validation_data=validation_mfcc_dataset,
    epochs=EPOCHS ,
    callbacks=tf.keras.callbacks.EarlyStopping(verbose=1, patience=10,
    restore_best_weights=True),
)
```

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 našem 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
Restoring model weights from the end of the best epoch: 18.
Epoch 28: early stopping
```

Na slici **??** prikazana su dva grafa. Lijevi prikazuje funkciju gubitka na trenažnom 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.

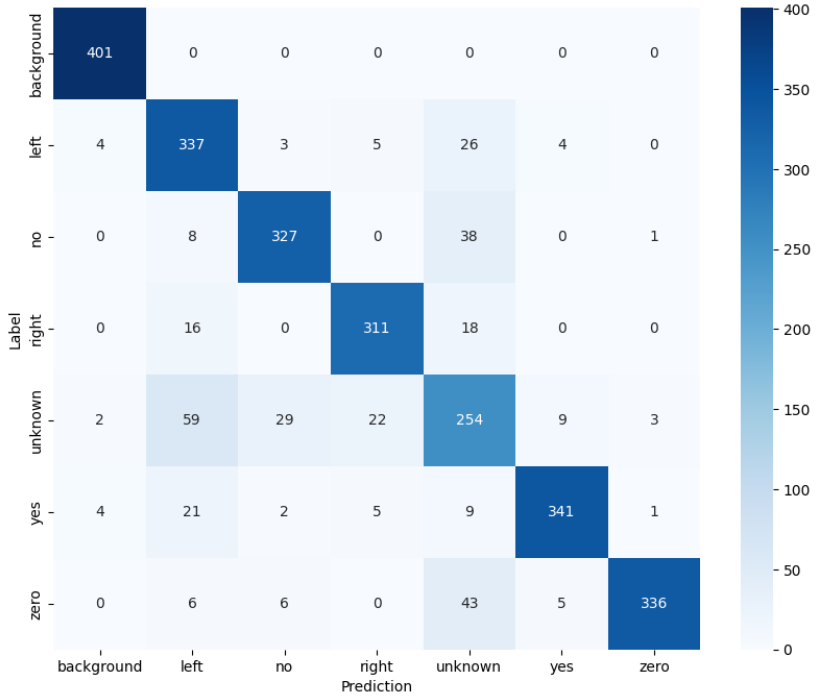


Slika 3.10. Gubitak i točnost modela

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 **??** prikazana je konfuzijska matrica napravljena nad podskupom testnog skupa podataka. Ona prikazuje koliko je puta model pogriješio u klasifikaciji određene klase. Stupci matrice predstavljaju stvarne klase, a reci predviđene klase (izlaz trenira-

nog 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 **??** prikazana je usporedba ovog modela s drugima. Uz naš trenirani model, ubačen je identičan model treniran na samo dvije glasovne naredbe (ukupno četiri klase). Iz tablice je vidljivo da naš 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)
Naš CNN (7 klasa)	88.2	15.7
Naš CNN (4 klase)	93.6	15.6
DNN (Deep Neural Network) [**?**]	84.6	80
CNN (Convolutional Neural Network) [**?**]	91.6	79
LSTM (Long Short-Term Memory) [**?**]	92.9	79.5
CRNN (Convolutional RNN) [**?**]	94.0	79.7
DS-CNN (Depthwise Separable CNN) [**?**]	94.4	38.6
TripletLoss-res15 [**?**]	95.2	237
BC-ResNet-8 [**?**]	98.7	520
WaveFormer [**?**]	98.8	130

Tablica 3.1. Veličine različitih oblika modela i procentualna ušteda

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. Korištenje 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 bismo saželi model još više te ga pretvorili 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 [**?**]. 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 [**?**]. Pretvorbom treniranog modela u ovakav oblik dobiveno je polje podataka spremno za korištenje programskim jezikom C. U isječku programskog koda **??** 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 **??** 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)**	**Smanjenje (%)**
Početni model	173241	100%
TF Lite	41644	24.04%
TF Lite + kvantizacija	15704	9.06%

Tablica 3.2. Veličine različitih oblika modela

4. Implementacija

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 **??** 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 isječku koda **??** 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.

```
1  BlankCommand command_back("BACKGROUND", 1, 0.7);
2  PrintCommand command_left("LEFT", 5, 0.8);
3  PrintCommand command_no("NO", 3, 0.80);
4  PrintCommand command_right("RIGHT", 3, 0.85);
5  BlankCommand command_unknown("UNKNOWN", 1, 0.7);
6  PrintCommand command_yes("YES", 5, 0.85);
7  PrintCommand command_zero("ZERO", 3, 0.85);
8
9  recognizer.addCommand(&command_back);
10 recognizer.addCommand(&command_left);
11 recognizer.addCommand(&command_no);
12 recognizer.addCommand(&command_right);
```

```
13 recognizer.addCommand(&command_unknown);
14 recognizer.addCommand(&command_yes);
15 recognizer.addCommand(&command_zero);
```

Kod 4..1: Stvaranje naredbi

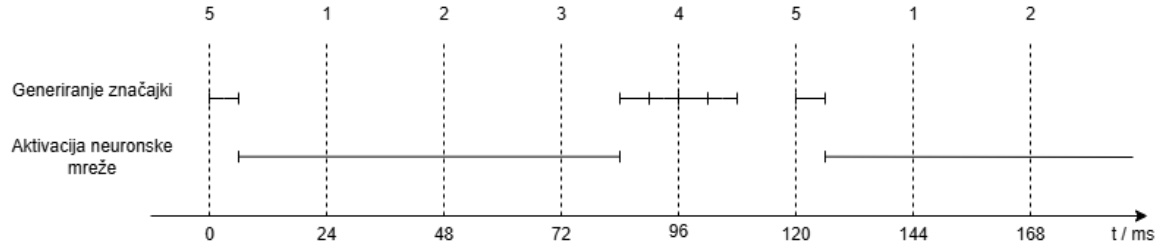
Zbog nesavršenosti skupa na kojem je treniran model i nejednakosti kvalitete zvučnih snimaka, neće svaka naredba biti prepoznata na isti način. Prvo, razlikovat će se pouzdanost vjerojatnosne interpretacije za različite klase. Drugo, uslijed izgovorene naredbe neke će klase imati najveću vjerojatnost različit broj iteracija rada sustava. To se događa zbog toga što sustav neprestano vrsti glavnu petlju opisanu u poglavlju o strukturi sustava **??** i trebao bi 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 **??** Značenje pojedinog parametra detaljno je opisano u **??**

Još jedna stvar koju je potrebno eksperimentalno utvrditi jest 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 **??** prikazana su maksimalna vremena potrebna za odrađivanje poslova pojedinih dijelova sustava.

Dio sustava	**Vrijeme(ms)**
Akvizicija uzoraka	0.016
Generiranje 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 generiranje 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 novih 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 **??** 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), generiraju se značajke nad tim prozorom te aktivira neuronska mreža. Vodoravne linije predstavljaju trajanja generiranja značajki i procesuiranja podataka u neuronskoj mreži. Duljine linija otprilike odgovaraju trajanju procesa: generiranje značajki 6 ms, 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 procesuiranje 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 bismo postavili period aktivacije na 4, 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 svakako da sustav ne bude na rubu stabilnosti jer malim kašnjenjem će se kroz vijek rada uređaja akumulirati kašnjenje i u jednom trenutku će doći do gubljenja podataka. Za točnu provjeru stabilnosti, problem ćemo riješiti analitički.

Zaključak je da je u jednom periodu rada sustava, tj. aktivacije neuronske mreže, potrebno generirati 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 generiranje 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 **??** i $t_{new} = 24$ ms dobivamo 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 uzmemo 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

Sustav za prepoznavanje govornih naredbi u stvarnom vremenu uspješno je implementiran na mikrokontrolerskoj platformi ESP32 Lyrat. Nudi jednostavniju modularnu stukturu od postojećih ([**?**], [**?**]) uz sposobnost pouzdanog prepoznavanja većeg skupa naredbi. Implementirani sustav može prepoznati pet različitih naredbi te aktivirati prikladni posao dodijeljen svakoj naredbi. Također, omogućen je odabir drugačijeg podskupa govornih naredbi, a uz malo više truda moguće je proširiti skup naredbi željenim zvučnim zapisima koji će predstavljati naredbu prilagođenu korisniku. Osim toga, modularna struktura sustava omogućuje laganu prilagodnu pojedinih dijelova za različite namjene. Na taj način stvorena je osnova za implementaciju različitih sustava za prepoznavanje drugih vrsta senzorskih podataka ili čak korištenje drugih vrsta neuronskih mreža. Ono što bi moglo poboljšati sustav u budućnosti je nekorištenje brojeva s pomičnim zarezom u generiranju MFC koeficijenata. Na konkretnom mikrontroleru to ne predstavlja prevelik problem zbog jedinice za operacije s pomičnim zarezom (engl. *Floating Point Unit ili FPU*), ali na drugim mikrokontrolerima bi takav pristup osjetno ubrzao cijeli proces.

Sažetak

Sustav za prepoznavanje govornih naredbi u stvarnom vremenu na rubnim uređajima

Luka Balić

Diplomski rad opisuje razvoj sustava za prepoznavanje govornih naredbi u stvarnom vremenu na ESP32 mikrokontroleru. Rad uključuje opis modularne arhitekture sustava koja koristi Mel kepstralne koeficijente (MFCC), dobivene iz akviziranog zvučnog signala, za treniranje konvolucijske neuronske mreže (CNN). Uz samu implementaciju, rad opisuje teorijsku pozadinu generiranja Mel kepstralnih koeficijenata te strukture i treniranja neuronske mreže. Implementirani sustav može pouzdano prepoznati 5 različitih govornih naredbi te pokrenuti odgovarajući posao povezan s tom naredbom.

Ključne riječi: govorne naredbe; MFCC; CNN; mikrokontroler

Abstract

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

Luka Balić

This master's thesis describes the development of a real-time voice command recognition system on the ESP32 microcontroller, also known as keyword spotting (KWS). The thesis includes a description of a system with a modular architecture that uses Mel-frequency cepstral coefficients (MFCC) extracted from the audio signal to train a convolutional neural network (CNN). In addition to the implementation, the thesis describes the theoretical background of generating MFCCs and the structure and training of the neural network. The implemented system can reliably recognize 5 different voice commands and execute the corresponding task associated with that command.

Keywords: kws; MFCC; CNN; microcontroller

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 [**?**]

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_ */
```

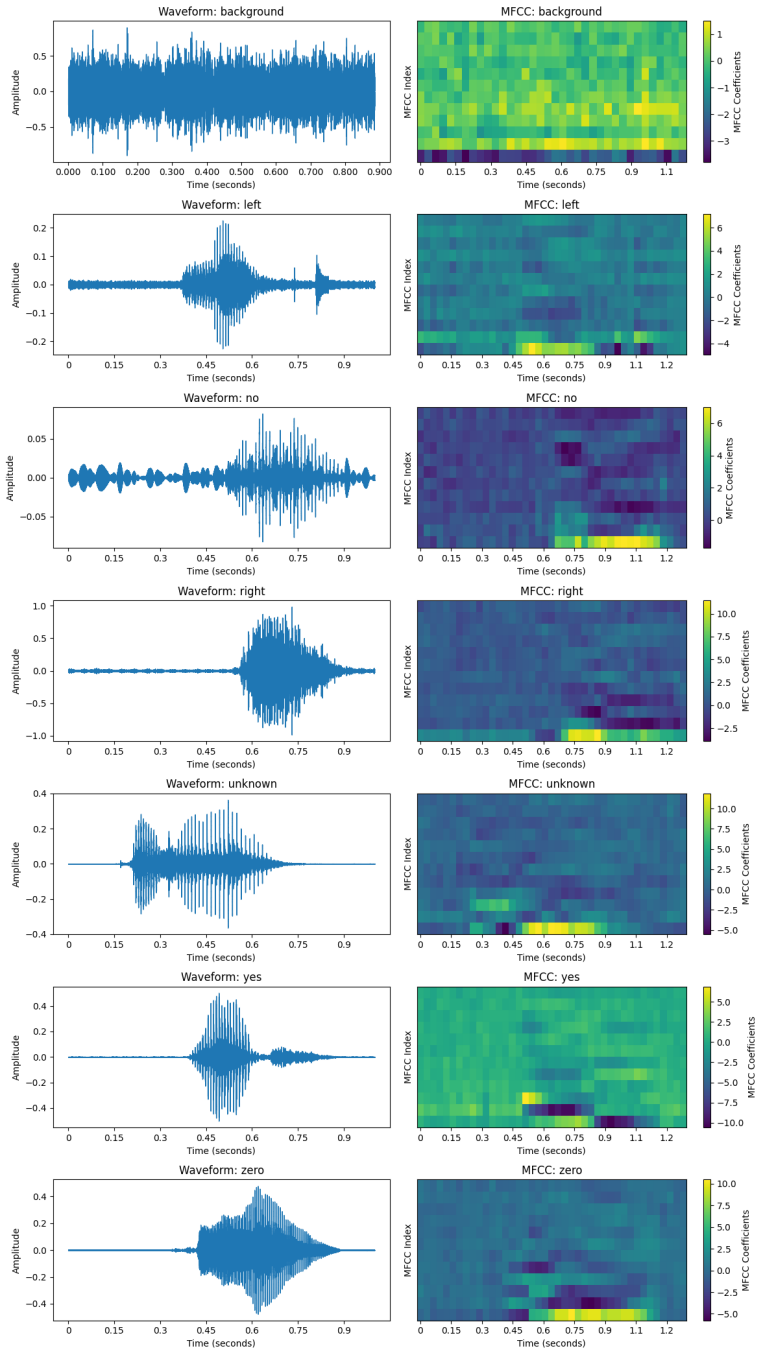
Kod B.1: Konačna konfiguracija

Privitak C: Generiranje značajki

```
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]
```

Kod C.1: Generiranje značajki korišteno u Jupyter bilježnici

Privitak D: Matrice značajki



Slika D1. Zvučni signali i pripadna matrica MFC koeficijenata