



Univerzitet u Sarajevu
Elektrotehnički fakultet u Sarajevu
Odsjek za računarstvo i informatiku



Detekcija umora korištenjem EEG tehnologije

Digitalno procesiranje signala
PROJEKT

Studentice:
Zerina Ahmetović, 19043
Lamija Gutić, 18977

Mentorica:
vanr. prof. dr Amila Akagić

Sarajevo, juni 2024.

SADRŽAJ

1. UVOD.....	3
2. KORIŠTENE TEHNOLOGIJE I SETOVI PODATAKA	4
2.1 Korištene biblioteke	4
3. METODE PREDPROCESIRANJA EEG SIGNALA.....	4
3.1 Učitavanje signala	4
3.2 Prikaz originalnog EEG signala	5
3.3 Primjena filtera na EEG signale.....	7
3.3.1 Notch filter	7
3.3.2 Butterworth filtri.....	11
3.3.3 Median filter	19
4. PREDPROCESIRANJE EEG SIGNALA.....	21
5. EKSTRAKCIJA ZNAČAJKI I ANALIZA SIGNALA.....	22
5.1 Ekstrakcija statističkih značajki.....	22
5.2 Transformacije.....	23
5.2.1 Wavelet transformacija.....	23
5.2.2 FFT (Fast Fourier Transform)	23
6. KLASIFIKACIJA I NEURALNA MREŽA.....	24
6.1 Kreiranje i arhitektura CNN-a.....	24
6.2 Callback rano zaustavljanje	25
7. ANALIZA GLAVNOG PROGRAMA.....	26
8. GRAFIČKA ANALIZA USPJEHA NEURALNE MREŽE.....	28
8.1 Performanse modela kroz epohe	29
9. ZAKLJUČAK.....	30
10. REFERENCE.....	31

1. UVOD

Umor ili manjak koncentracije predstavlja značajan problem u današnjem užurbanom svijetu, posebno kada su u pitanju svakodnevne situacije poput učešća u saobraćaju. Naravno, to nije jedini primjer gdje nedostatak koncentracije ili umor mogu predstavljati prepreku. Tu spadaju i zahtjevni industrijski zadaci. Ipak, manjak koncentracije prilikom vožnje predstavlja daleko najveći i najopasniji potencijalni problem [1].

Najveći uzrok umora kod vozača i drugih učesnika u saobraćaju su mentalna iscrpljenost i nedostatak sna, zbog čega postoji mogućnost da vozači 'odlutažu' u mislima tokom dugih neometanih vožnji, posebno onih na dugim autocestama i drugim pravcima gdje ne postoji onoliko prekida koliko se susreće u gradu, gdje prirodno ima više raskrsnica i pješačkih prelaza.

Postoji nekoliko metoda za mjerenje umora i pospanosti koji se najviše baziraju na samoprocjeni, međutim objektivne metode za mjerenje umora i manjka koncentracije se oslanjaju na fiziološke karakteristike i karakteristike ponašanja i one su dosta pouzdanije. Ove metode uključuju i algoritme za kompjutersku viziju koji koriste kamere da bi se detektovala promjena na licima vozača.

Umor i pospanost karakteriziraju iscrpljeni izrazi lica, često zijevanje, produženo zatvaranje očiju, brzo treptanje i oboren položaj glave. Ipak, identificiranje umora koristeći ponašanje poput treptaja, pokreta usana, učestalosti zijevanja i facijalnih karakteristika može prouzrokovati lažne detekcije. Posebna važnost pridaje se otvorenosti/zatvorenosti očiju zajedno sa analizom moždanih signala. Glavni razlog ovakvog pristupa jeste što loša osvjetljenost može uzrokovati netačnu detekciju. Međutim, znajući kako je ljudski mozak odgovoran za sve tjelesne odgovore, moždana aktivnost se može koristiti za ranu detekciju umora.

Cilj našeg projekta je razvijanje programa koji obrađuje EEG setove podataka sa elektroda i iz njih detektuje stepen umora kod ispitanika na osnovu toga da li su im oči otvorene ili zatvorene. Pokazano je da se za ove svrhe koristi neinvazivni *brain-computer interfejs* (BCI) kako bi se na osnovu moždane aktivnosti izmjerila budnost i umor zbog njegove dobre vremenske rezolucije [1]. Za ovu svrhu se mogu koristiti razne metode, poput analize frekvencija, korelacione analize, analize u vremenskom domenu sa tehnikama mašinskog učenja, i slično. Ova procedura uključivat će predprocesiranje EEG signala kako bi se isti podaci mogli pripremiti za obradu i ekstrakciju značajki, te bi kao takvi u finalnoj fazi klasifikacije bili pogodni za treniranje konvolucijske neuralne mreže (*engl. Convolutional Neural Network - CNN*), koja predstavlja centralni dio ovog projekta.

2. KORIŠTENE TEHNOLOGIJE I SETOVI PODATAKA

Za izradu ovog projekta koristio se programski jezik Python (verzija 3.12), te radno okruženje PyCharm.

Uslijed nedostupnosti podataka sa elektroencefalografskih uređaja i nemogućnosti prikupljanja vlastitog seta, korišteni su setovi podataka dostupni na internetu.

Upotrijebljena su dva seta podataka koji se mogu pronaći na sljedećim linkovima sa web stranice Kaggle:

- [Eye State Classification EEG Dataset](#)
- [Neuroheadstate Eye-State Classification](#).

Oba seta podataka predstavljaju podatke prikupljane sa elektroda (svaka kolona predstavlja jednu elektrodu), dok kolona 'eye-state' predstavlja da li je oko otvoreno (1) ili zatvoreno (0). Ovi podaci pohranjeni su u fajlovima .csv formata.

2.1 Korištene biblioteke

U projektu su korištene sljedeće biblioteke:

1. NumPy (za numeričku analizu)
2. Pandas (za učitavanje setova podataka)
3. Matplotlib (za prikaz grafova datih setova podataka)
4. SciPy (za implementaciju filtera u fazi predprocesiranja)
5. PyWavelets (za implementiranje Wavelet transformacije)
6. Tensorflow (za kreiranje modela mašinskog učenja)
7. SkLearn (za dalju implementaciju modela mašinskog učenja).

3. METODE PREDPROCESIRANJA EEG SIGNALA

Da bi se EEG signali mogli efikasno obraditi, potrebno je da se ti podaci predprocesiraju. Razlog ovome su artefakti koji se pojavljuju tokom akvizicije podataka, poput raznih šumova koji nastaju uslijed ljudske reakcije, treptaja i pokreta oka, ali i vanjskih šumova poput onih koji nastaju uslijed rada elektroenergetske mreže.

3.1 Učitavanje signala

Za učitavanje EEG signala iz .csv fajla koristit ćemo pandas biblioteku, i naš set podataka učitat ćemo na sljedeći način:

```

import pandas as pd

def load_eeg_data(file_path, header='infer', column_names=None):

    try:
        # Učitamo CSV fajl u DataFrame
        eeg_data = pd.read_csv(file_path, header=header)

        # Ako je zaglavlje None, postavimo imena kolona
        if header is None and column_names is not None:
            eeg_data.columns = column_names

        return eeg_data
    except Exception as e:
        print("Došlo je do greške prilikom učitavanja fajla: {e}")
        return None

file_path = 'train.csv'
eeg_data = load_eeg_data(file_path)

if eeg_data is not None:
    print(eeg_data.head())

```

Kao što vidimo, kako bismo olakšali kasnije pozive učitavanja EEG podataka, za ovu svrhu smo napravili funkciju `load_eeg_data`. Za potrebe učitavanja EEG data i kreiranje *dataframe*-a koristili smo Pandas biblioteku. Također, ukoliko zaglavlje nije definisano, ova funkcija će postaviti imena kolona.

3.2 Prikaz originalnog EEG signala

Sljedeći korak u našem programu je skiciranje/prikaz originalnog EEG signala. Ovo ćemo postići korištenjem matplotlib biblioteke i kreiranjem nove funkcije `plot_eeg` kako bismo funkcije za plotanje prilagodili EEG tipu podataka. Naši ulazni podaci bit će set sa kojim radimo, zatim kanali koje želimo prikazati, potom početni i krajnji uzorak, frekvencija uzorkovanja (sampling rate) i naziv grafa.

```

def plot_eeg(data, channels=None, start=0, end=None, sampling_rate=1, title='EEG
Signal'):
    """
    Funkcija za plotanje EEG signala.

    :param data: DataFrame sa EEG podacima.
    :param channels: Lista kanala za plotanje. Ako je None, plotaju se svi kanali.
    :param start: Početni uzorak za plotanje.
    :param end: Krajnji uzorak za plotanje. Ako je None, plotaju se svi uzorci do
    kraja.
    :param sampling_rate: Frekvencija uzorkovanja u Hz (koristi se za kreiranje
    vremenske ose).
    :param title: Naslov grafa.
    """
    if channels is None:
        channels = data.columns

    if end is None:
        end = len(data)

```

```
# Kreiranje vremenske ose
time = (start + np.arange(end - start)) / sampling_rate

plt.figure(figsize=(15, 8))
for channel in channels:
    if channel in data.columns:
        plt.plot(time, data[channel].iloc[start:end], label=channel)
    else:
        print(f"Channel {channel} not found in data columns.")

plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title(title)
plt.legend()
plt.show()
```

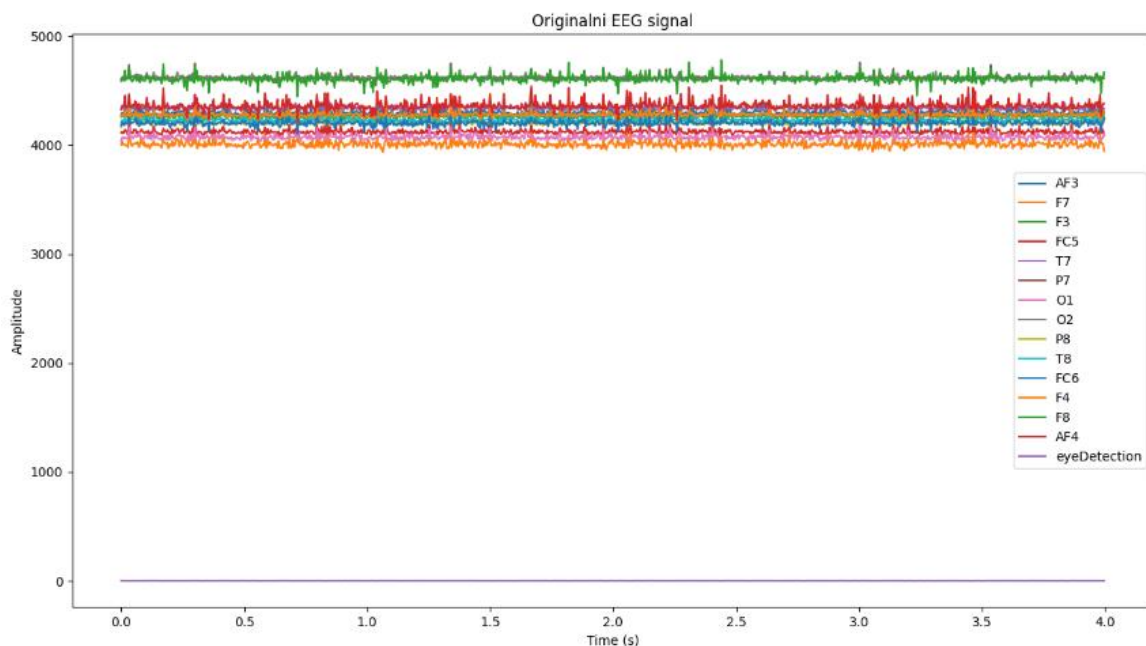
Da bismo prikazali originalni EEG signal, pozvat ćemo ovu funkciju u glavnom programu:

```
def main():
    file = 'train.csv'
    eeg_data = load_eeg_data(file)

    if eeg_data is not None:
        plot_eeg(eeg_data, channels=None, start=0, end=1000, sampling_rate=250,
        title='Originalni EEG signal')

if __name__ == '__main__':
    main()
```

Ono što dobijemo nakon poziva ove funkcije je:



Slika 1: Originalni EEG signal

Pozivom funkcije `plot_eeg` dobili smo grafik datog seta podataka za sve elektrode u vremenskom periodu od 4 sekunde.

3.3 Primjena filtera na EEG signale

Da bi se EEG signal mogao kvalitetno obraditi i pripremiti za dalju analizu, potrebno je da odstranimo artefakte koji mogu nastati uslijed različitih faktora. U našem seminarskom radu već je spomenuto da postoje dvije vrste artefakata koji ometaju rad signala: unutrašnji i vanjski. Unutrašnji artefakti podrazumijevaju bilo kakve šumove koji nastaju uslijed normalnih fizioloških funkcija, poput treptaja oka ili srčanog ritma. Vanjski (ekstrafiziološki) artefakti uključuju sve šumove koji su rezultat eksternih utjecaja. Najčešće korištena metoda za uklanjanje ovakvih artefakata je filtriranje signala. U ovom odjeljku analizirat ćemo *notch filter*, *butterworth* filtre i *median filter*.

3.3.1 Notch filter

Notch filter je vrsta linearnog frekvencijskog filtra koji je dizajniran da priguši ili ukloni određenu frekvenciju ili uski opseg frekvencija iz signala, dok ostale frekvencije propušta nesmanjene. Ovi filtri su posebno korisni za uklanjanje specifičnih nepoželjnih frekvencija, kao što su šum ili smetnje, iz audio, elektronskih ili komunikacionih sistema.

U ovom programu, notch filter će se koristiti za uklanjanje šuma koji uzrokuje rad elektroenergetske mreže. Ono što prvo moramo odrediti je frekvencija smetnji koje želimo ukloniti. U našem slučaju, to je 50 Hz budući da je to frekvencija rada elektroenergetske mreže.

Za kreiranje funkcije notch filtra koristit ćemo `iirnotch` funkciju iz SciPy biblioteke.

```
def notch_filter(data, freq, fs, quality_factor=30):  
    nyq = 0.5 * fs  
    freq = freq / nyq  
    b, a = iirnotch(freq, quality_factor)  
    filtered_data = data.apply(lambda x: filtfilt(b, a, x), axis=0)  
    return filtered_data
```

Ulazni parametri ove funkcije su set podataka koji će biti filtriran, frekvencija koja treba biti eliminisana, frekvencija uzorkovanja i faktor kvaliteta filtera. Sve frekvencije su zadate u Herzima (Hz). Posljednji parametar određuje širinu frekventnog opsega koji će biti eliminisan. Za potrebe ovog programa podrazumijevana vrijednost je 30.

U ovoj funkciji se prvo izračunava vrijednost Nyquist frekvencije (polovina frekvencije uzorkovanja). U odnosu na tu vrijednost, vrši se normalizacija frekvencije koja treba biti eliminisana. Nakon normalizacije, generiraju se parametri filtera pri čemu se koristi funkcija `iirnotch`. Ova funkcija generirat će koeficijente `b` i `a` koji definiraju notch filter. Na samom kraju, vrši se primjena filtera uz upotrebu `filtfilt` funkcije iz SciPy.signal biblioteke koja primjenjuje filter dva puta, naprijed i nazad, kako bi se izbjegla fazna distorzija. Primjenu vršimo koristeći `apply` metodu iz Pandas biblioteke koja primjenjuje funkciju na svaku kolonu dataframe-a.

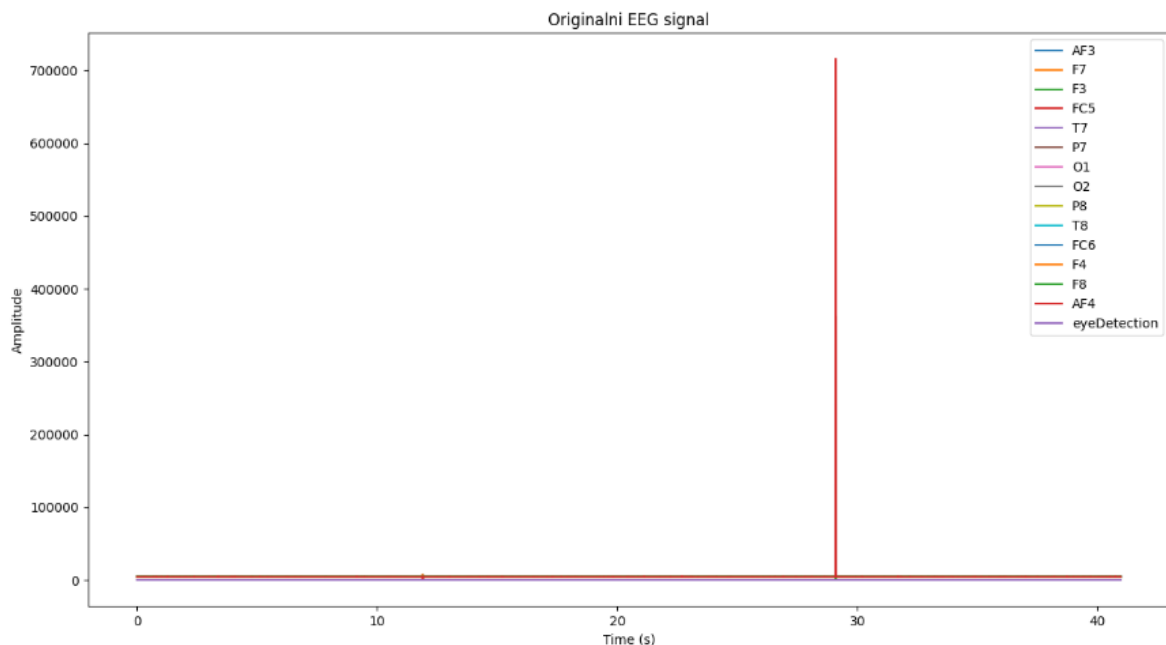
U glavnom programu na sljedeći način možemo upotrijebiti notch filter nad kompletnim setom podataka:

```
def main():
    file_path = 'train.csv'
    eeg_data = load_eeg_data(file_path)
    filtered_data = notch_filter(eeg_data, 50, 256)

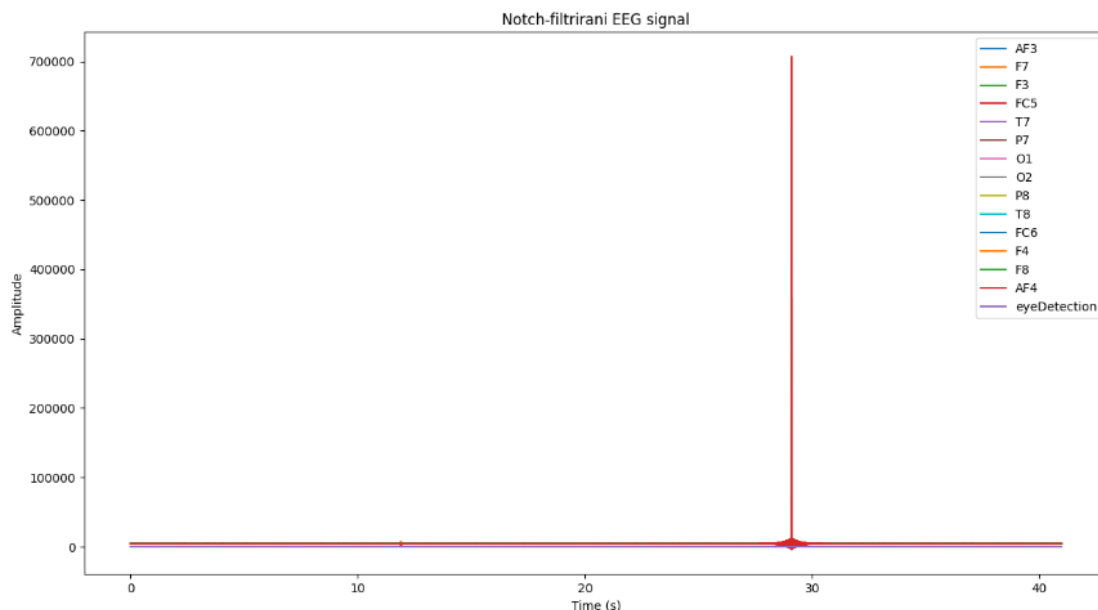
    if eeg_data is not None:
        plot_eeg(eeg_data, channels=None, sampling_rate=256, title='Originalni EEG signal')
        plot_eeg(filtered_data, channels=None, sampling_rate=256, title='Notch-filtrirani EEG signal')

if __name__ == '__main__':
    main()
```

Pokretanjem programa dobijaju se sljedeći grafovi:



Slika 2: Originalni EEG signal (kompletan set podataka)



Slika 3: Notch-filtrirani EEG signal (kompletan set podataka)

Pored prikaza filtriranog EEG signala koristeći notch filter, ovu funkciju je dalje moguće iskoristiti i za prikaz filtriranog signalnog spektra u frekventnom domenu. Da bismo to postigli, prvo je potrebno da kreiramo funkciju za plotanje u frekventnom domenu. Ova funkcija će kreirati graf koji će na x-osi prikazivati frekvenciju signala, dok će na y-osi biti prikazana spektralna gustina snage. Da bismo to postigli, koristit ćemo Welchovu metodu iz SciPy biblioteke. Ove dvije vrijednosti se na grafu prikazuju koristeći semilogy koji daje logaritamski prikaz y-ose.

```
def plot_frequency_domain(data, fs, title):
    plt.figure(figsize=(10, 6))
    for column in data.columns:
        f, Pxx = welch(data[column], fs, nperseg=1024)
        plt.semilogy(f, Pxx, label=column)
    plt.title(title)
    plt.xlabel('Frequency (Hz)')
    plt.ylabel('Power Spectral Density (V^2/Hz)')
    plt.legend()
    plt.show()

# Set parameters
fs = 256 # Example sampling frequency, replace with actual one if different
freq_to_remove = 50 # Example frequency to remove, typically 50 or 60 Hz

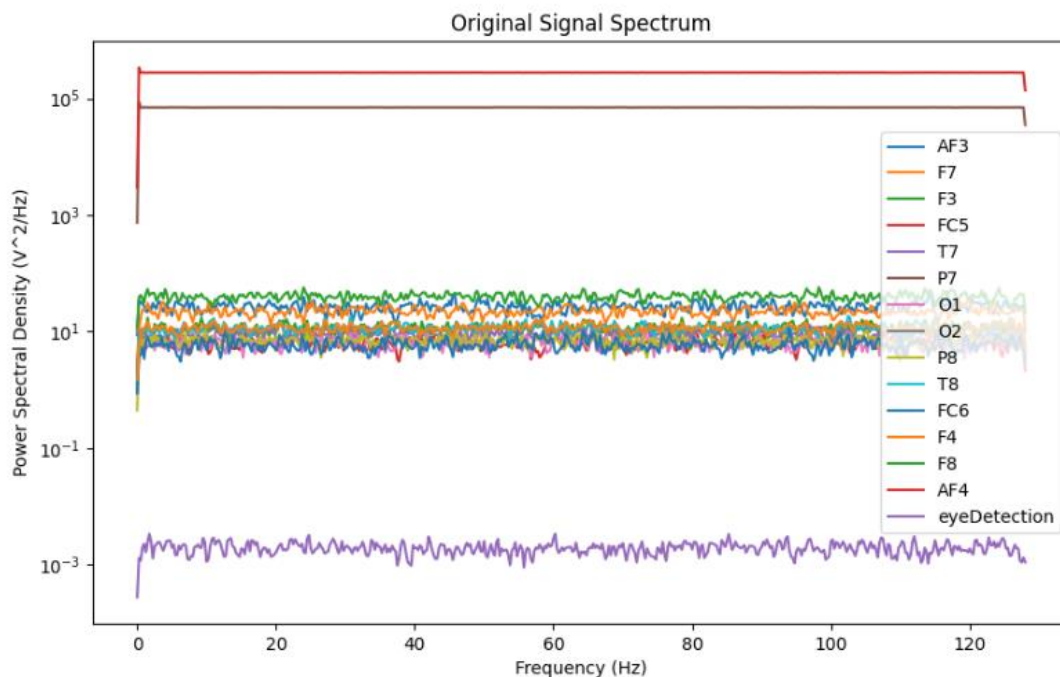
# Load data
file_path = 'train.csv'
data = load_data(file_path)

# Plot original frequency-domain signal
plot_frequency_domain(data, fs, 'Original Signal Spectrum')

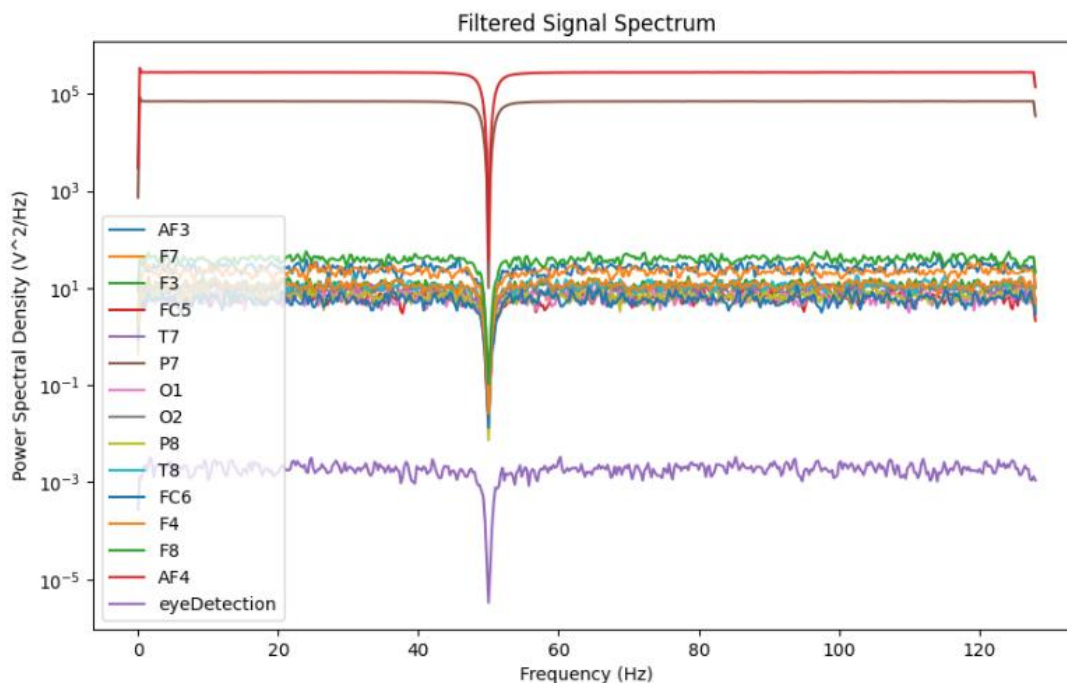
# Apply notch filter
filtered_data = notch_filter(data, freq_to_remove, fs)

# Plot filtered frequency-domain signal
plot_frequency_domain(filtered_data, fs, 'Filtered Signal Spectrum')
```

Korištenjem ove funkcije nad originalnim setom podataka i filtriranim setom podataka dobijaju se:



Slika 4: Originalni signalni spektar (x-osa: frekvencija u Hz; y-osa: spektralna gustina snage)



Slika 4: Filtrirani signalni spektar (x-osa: frekvencija u Hz; y-osa: spektralna gustina snage)

Na prikazu signalnog spektra iznad jasno se vidi razlika za vrijednost frekvencije 50 Hz što ukazuje na djelovanje notch filtra na korišteni set podataka.

3.3.2 Butterworth filtri

Butterworth filter je filter koji funkcionira na principu ograničavanja frekvencija unutar kojih se mogu kretati amplitude nekog signala. Ključna karakteristika Butterworth filtra je da nema oštarih prelaza između propusnog i nepropusnog opsega frekvencija, što znači da je funkcija glatka u obliku i nema oscilacija u odzivu.

Postoji četiri vrste butterworth filtera:

- Bandpass butterworth filter
- Lowpass butterworth filter
- Highpass butterworth filter
- Bandstop butterworth filter.

Funkcije za ove filtre će imati dosta toga zajedničkog, a između ostalog to je i računanje Nyquistove frekvencije koja predstavlja polovinu frekvencije uzorkovanja, nakon čega se vrši normalizacija graničnih vrijednosti.

3.3.2.1 Bandpass butterworth filter

Butterworth band-pass filter je vrsta Butterworth filtera koji propušta signale unutar određenog frekvencijskog opsega, dok blokira signale izvan tog opsega.

Programski kod za Butterworth bandpass filter funkciju:

```
def butter_bandpass(lowcut, highcut, fs, order=4):  
    nyq = 0.5 * fs  
    low = lowcut / nyq  
    high = highcut / nyq  
    b, a = butter(order, [low, high], btype='band')  
    return b, a  
  
def butter_bandpass_filter(data, lowcut, highcut, fs, order=4):  
    b, a = butter_bandpass(lowcut, highcut, fs, order=order)  
    filtered_data = data.apply(lambda x: filtfilt(b, a, x), axis=0)  
    return filtered_data
```

Funkcija `butter_bandpass` kreira bandpass filter koristeći `lowcut` i `highcut` parametre kao frekvencijske granice za signale koji će ostati zadržani. Unutar funkcije vrši se normalizacija ovih vrijednosti u odnosu na Nyquistovu frekvenciju. Za kreiranje ovog filtra koristili smo funkciju `butter` iz SciPy biblioteke. Ova funkcija vraća koeficijente filtra `b` i `a`.

Funkcija `butter_bandpass_filter` primjenjuje bandpass filter na set EEG podataka koristeći prethodno kreiranu `butter_bandpass` funkciju koja generiše koeficijente bandpass filtra.

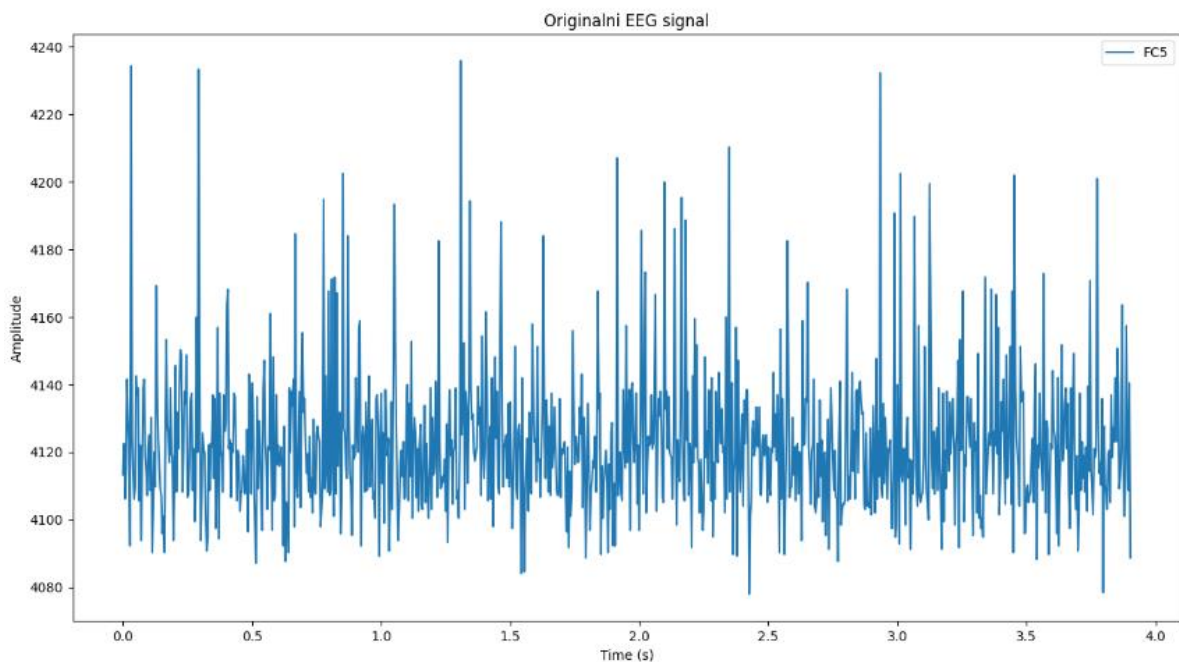
Nakon što su generisani koeficijenti, filter se primjenjuje na date podatke uz upotrebu `filtfilt` funkcije iz SciPy biblioteke. Funkcija vraća filtrirane podatke.

Kod za generiranje grafova bandpass filtra:

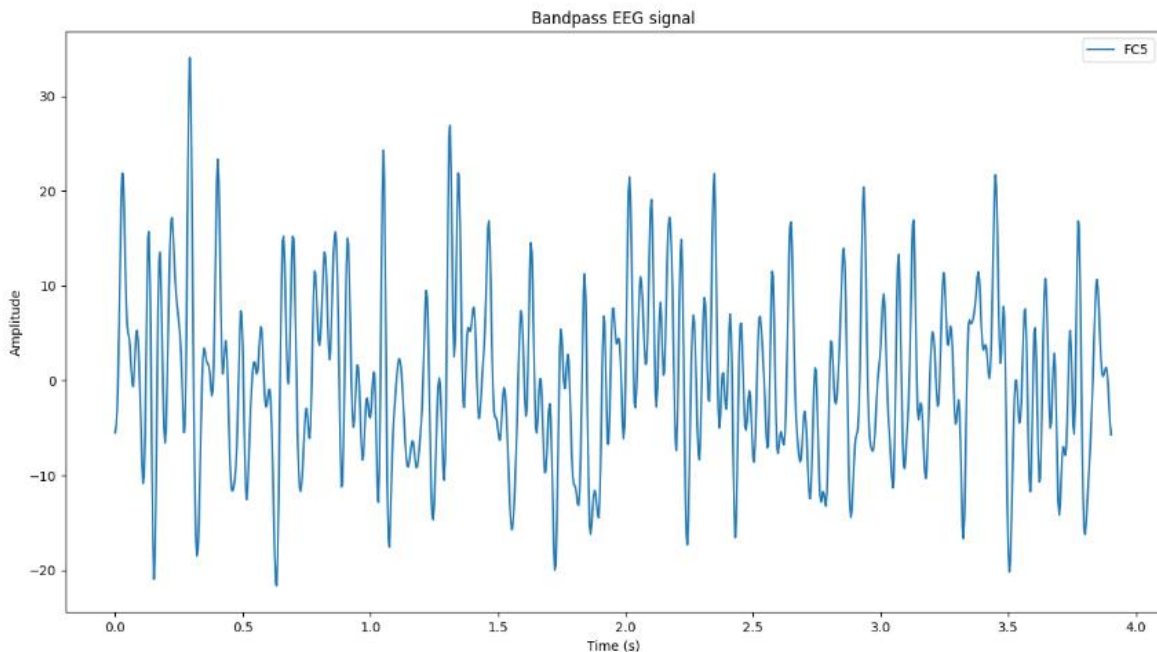
```
def main():
    file_path = 'train.csv'
    eeg_data = load_eeg_data(file_path)
    filtered_data = butter_bandpass_filter(eeg_data, 0.4, 30, 256)

    if eeg_data is not None:
        plot_eeg(eeg_data, channels=['FC5'], start=0, end=1000, sampling_rate=256,
title='Originalni EEG signal')
        plot_eeg(filtered_data, channels=['FC5'], start=0, end=1000,
sampling_rate=256, title='Bandpass EEG signal')

if main is not None:
    main()
```



Slika 6: Originalni EEG signal za FC5 elektrodu u vremenskom periodu 4s



Slika 7: Bandpass-filtrirani EEG signal za FC5 elektrodu u vremenskom periodu 4s

Ono što možemo primijetiti nakon primjene bandpass filtra je da se amplituda na y-osi sada kreće u datom opsegu (gornja granična frekvencija 30 Hz) i da je signal dosta više pročišćen nego originalni, što je jedna od osnovnih karakteristika Butterworth filtera. Ovaj filter pokazao se korisnim pri analizi sna gdje je potrebno izolirati različite frekvencijske opsege vezane za različite faze sna.

3.3.2.2 Lowpass butterworth filter

Lowpass Butterworth filter je vrsta filtera koja se koristi za prolaz signala niže frekvencije, dok se visoke frekvencije potiskuju ili filtriraju. Za razliku od već spomenutog bandpass filtra, ovaj filter će imati samo donju graničnu vrijednost. Ovi filtri su korisni za uklanjanje visokofrekventnog šuma koji može potjecati od elektromiografskih artefakata (npr. mišićni pokreti), električnih smetnji ili drugih izvora koji utječu na EEG signal.

Na sličan način kreiramo funkciju za lowpass filter:

```
def butter_lowpass(cutoff, fs, order=4):
    nyq = 0.5 * fs
    normal_cutoff = cutoff / nyq
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    return b, a

def butter_lowpass_filter(data, cutoff, fs, order=4):
    b, a = butter_lowpass(cutoff, fs, order=order)
    y = filtered_data = data.apply(lambda x: filtfilt(b, a, x), axis=0)
    return y
```

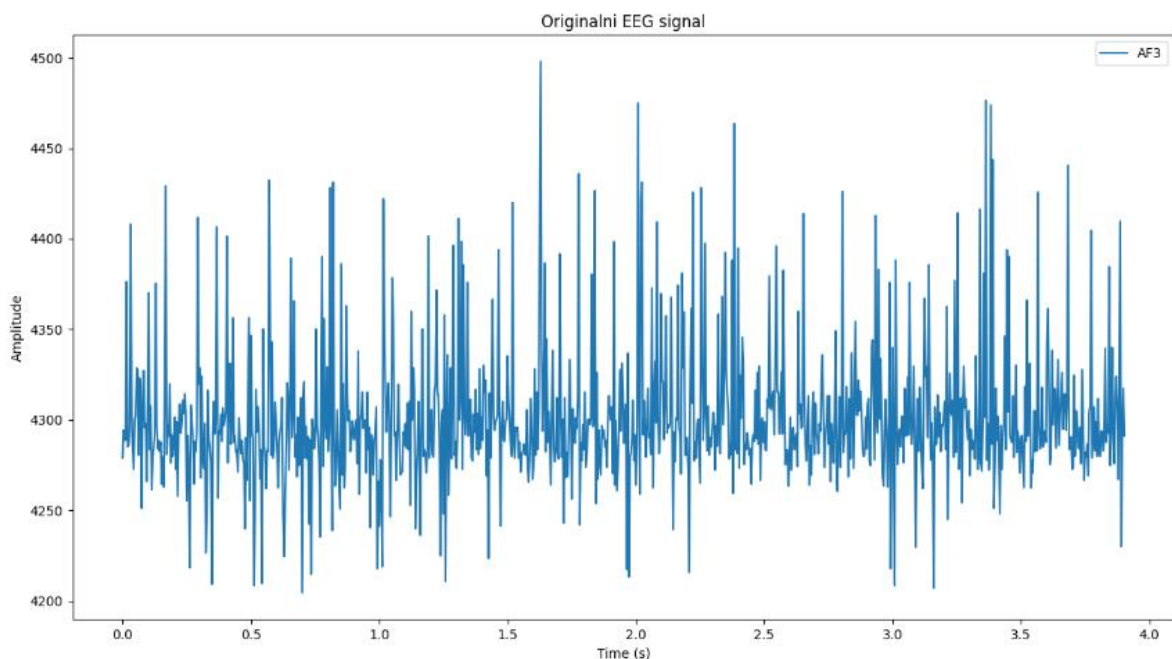
U glavnom programu testiramo ovaj filter:

```
def main():
    file_path = 'train.csv'
    eeg_data = load_eeg_data(file_path)
    filtered_data = butter_lowpass_filter(eeg_data, 17, 256)

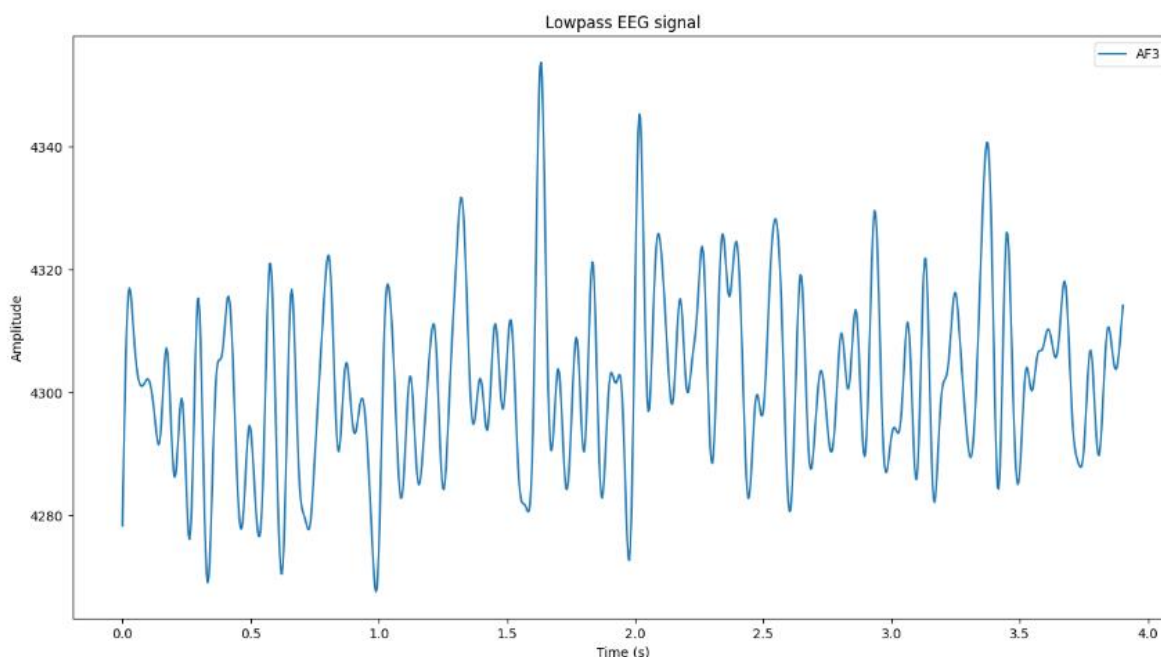
    if eeg_data is not None:
        plot_eeg(eeg_data, channels=['AF3'], start=0, end=1000, sampling_rate=256,
title='Originalni EEG signal')
        plot_eeg(filtered_data, channels=['AF3'], start=0, end=1000,
sampling_rate=256, title='Lowpass EEG signal')

if main is not None:
    main()
```

Nakon pokretanja, dobijamo sljedeće grafove:



Slika 8: Originalni EEG signal za AF3 elektrodu u vremenskom periodu 4s



Slika 9: Lowpass-filtrirani EEG signal za FC5 elektrodu u vremenskom periodu 4s

Sa y-ose vidimo da je amplituda na grafu filtriranog signala niža što ukazuje na to da su visoke frekvencije uklonjene ovim filtrom. Možemo primijetiti da je signal sa ove elektrode dosta 'gladi' i jasniji za čitanje nakon primjene lowpass Butterworth filtra, ali da je glavna karakteristika filtra dobrim dijelom zadržana što može biti korisno u kliničkoj analizi jer se ovim filtrom mogu izdvojiti osobine signala koje su vezane za niskofrekventne aktivnosti mozga. Smanjenjem šuma i visokofrekventnih artefakata, lowpass filter može poboljšati odnos signal-šum čineći prave signale mozga lakšim za interpretaciju.

3.3.2.3 Highpass butterworth filter

Highpass butterworth filter radi suprotno od lowpass filtra; koristi se za propuštanje visokih frekvencija, dok se niske frekvencije uklanjaju ili potiskuju. Najčešće se koristi za propuštanje frekvencija iznad 0.5-1 Hz.

Highpass filter funkciju implementiramo sa jednom razlikom u odnosu na lowpass filter, a to je da za btype koristimo ključnu riječ 'high' umjesto 'low':

```
def butter_highpass(cutoff, fs, order=4):
    nyq = 0.5 * fs
    normal_cutoff = cutoff / nyq
    b, a = butter(order, normal_cutoff, btype='high', analog=False)
    return b, a

def butter_highpass_filter(data, cutoff, fs, order=4):
    b, a = butter_lowpass(cutoff, fs, order=order)
    y = data.apply(lambda x: filtfilt(b, a, x), axis=0)
    return y
```

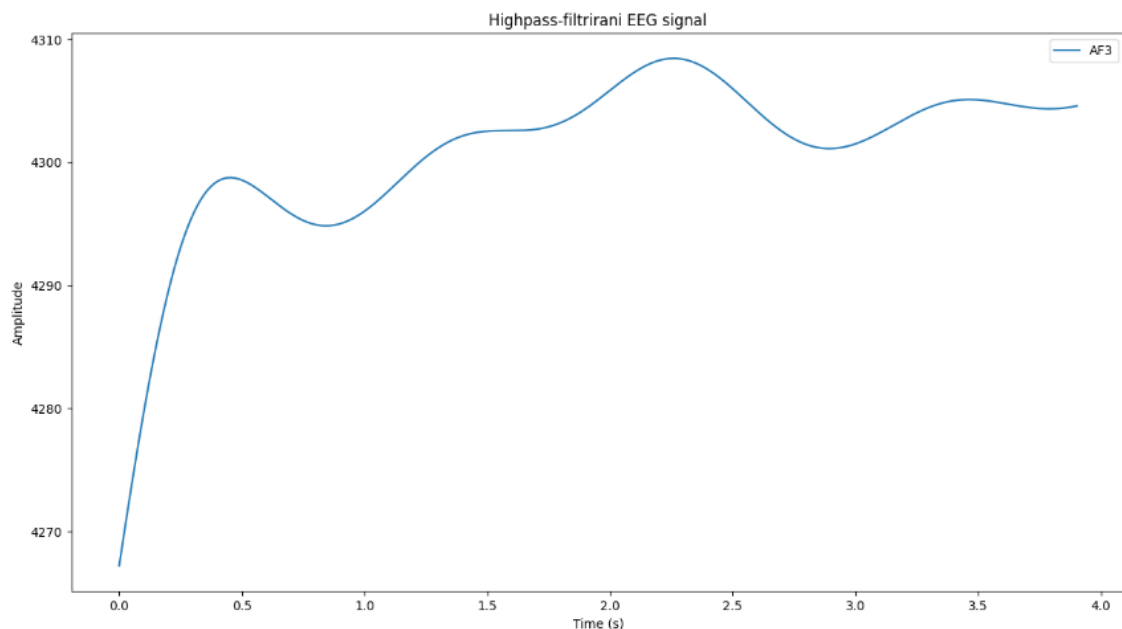
Ovaj filter u glavnom programu možemo primijeniti na sličan način kao i lowpass filter:

```
def main():
    file_path = 'train.csv'
    eeg_data = load_eeg_data(file_path)
    filtered_data = butter_highpass_filter(eeg_data, 1, 256)

    if eeg_data is not None:
        plot_eeg(eeg_data, channels=['AF3'], start=0, end=1000, sampling_rate=256,
        title='Originalni EEG signal')
        plot_eeg(filtered_data, channels=['AF3'], start=0, end=1000,
        sampling_rate=256, title='Highpass-filtrirani EEG signal')

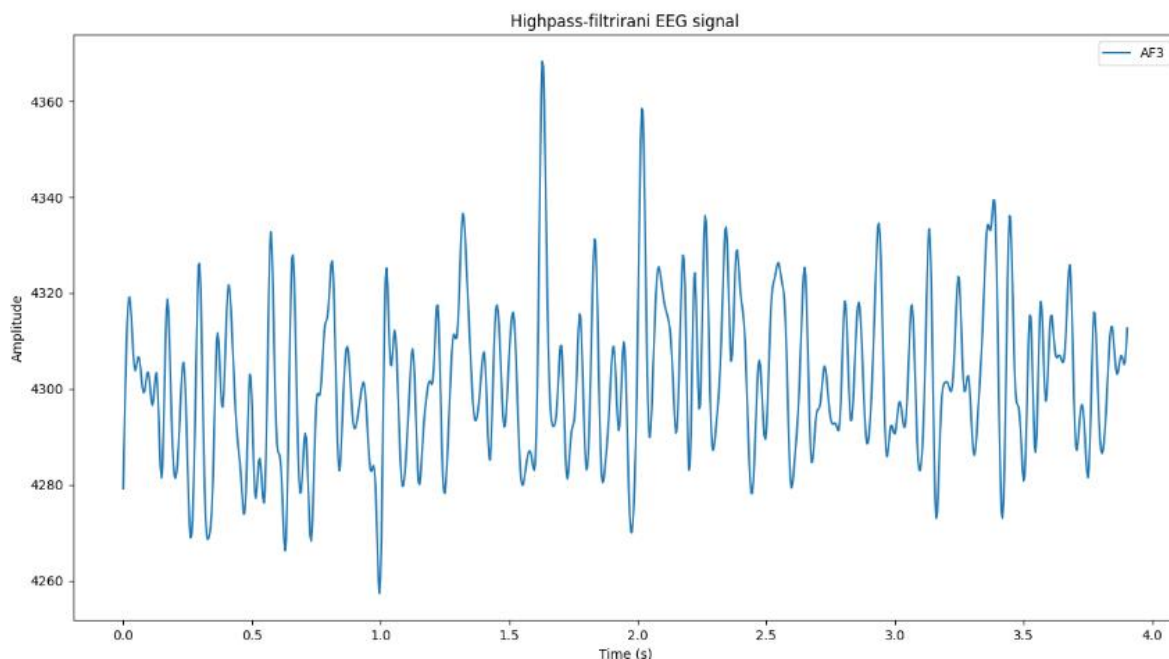
if main is not None:
    main()
```

Pokretanjem dobijamo sljedeći rezultat (originalni signal je isti kao na Slici 8):



Slika 10: Highpass-filtrirani EEG signal za FC5 elektrodu u vremenskom periodu 4s i cutoff vrijednost 1

Može se primijetiti značajna deformacija signala u odnosu na original zbog vrlo niske cutoff vrijednosti čime se gubi dosta informacija o signalu jer signal sada ima dosta manje varijacija. Ovako filtrirani signal nam neće reći puno o samoj karakteristici signala. Ipak, ako promijenimo cutoff vrijednost na 25, dobijamo sljedeći graf:



Slika 11: Highpass-filtrirani EEG signal za FC5 elektrodu u vremenskom periodu 4s i cutoff vrijednost 25

Za ovu cutoff vrijednost, glavna karakteristika signala je zadržana, dok su šumovi uklonjeni. Ovaj filtrirani signal je korisniji za analizu visokofrekventnih komponenti, što može biti važno za određene tipove EEG analiza, poput otkrivanja epileptičnih aktivnosti ili drugih brzih moždanih događaja.

3.3.2.4 Bandstop butterworth filter

Bandstop filter zaustavlja sve frekvencije koje se nalaze unutar nekog opsega dok se sve ostale propuštaju. Notch filter je jedna vrsta bandstop filtra koja ima vrlo uzak frekvencijski pojas koji treba zaustaviti.

Programski kod za bandstop butterworth filter:

```
def butter_bandstop_filter(data, lowcut, highcut, fs, order=4):
    nyquist = 0.5 * fs
    low = lowcut / nyquist
    high = highcut / nyquist
    sos = butter(order, [low, high], btype='bandstop', output='sos')
    filtered_data = sosfilt(sos, data, axis=0)

    return pd.DataFrame(filtered_data, columns=data.columns)
```

Unutar funkcije definirali smo Nyquistovu frekvenciju (polovina frekvencije uzorkovanja) i širinu opsega nakon čega smo primijenili bandstop filter koristeći butter funkciju.

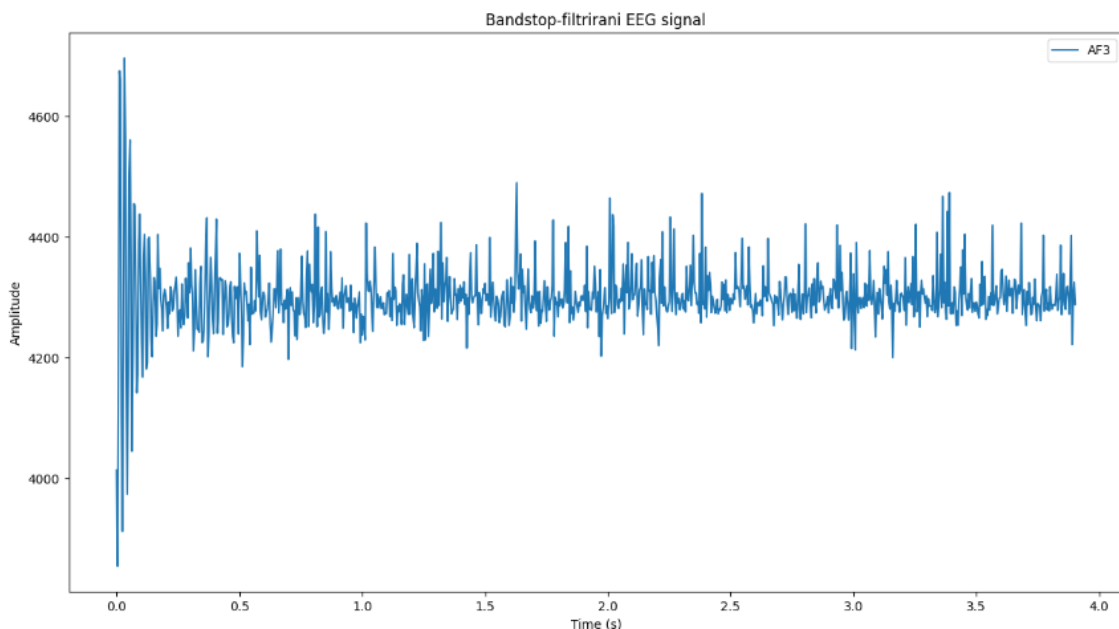
Primjena bandstop filtra u glavnom programu:

```
def main():
    file_path = 'train.csv'
    eeg_data = load_eeg_data(file_path)
    filtered_data = butter_bandstop_filter(eeg_data, 49, 51, 256)

    if eeg_data is not None:
        plot_eeg(eeg_data, channels=['AF3'], start=0, end=1000, sampling_rate=256,
        title='Originalni EEG signal')
        plot_eeg(filtered_data, channels=['AF3'], start=0, end=1000,
        sampling_rate=256, title='Bandstop-filtrirani EEG signal')

if main is not None:
    main()
```

Pokretanjem dobijamo sljedeći graf (originalni signal je isti kao na Slici 8):



Slika 12: Bandstop-filtrirani EEG signal za FC5 elektrodu u vremenskom periodu 4s i cutoff vrijednost 1

Nakon primjene bandstop filtra vidimo značajnu promjenu u izgledu signala. Za veći dio vremenskog perioda, amplituda signala je niža, dok su se sve visokofrekventne vrijednosti 'suzbile' na početku. Iako je frekvencijski pojas poprilično uzak, signal je deformisan u odnosu na početni. Ukoliko povećano highcut vrijednost i proširimo opseg, signal će biti još više deformiran zbog zaustavljanja šireg opsega frekvencija.

3.3.3 Median filter

Za razliku od prethodno spomenutih notch i butterworth filtera, **median filter** je nelinearna tehnika digitalnog filtriranja. Ovaj filter zamjenjuje svaki uzorak u signalu sa median vrijednosti definiranog skupa susjednih uzoraka. Posebno je efektivan za uklanjanje 'salt-and-pepper' šumova, kao i drugih vrsta impulsa (spikes). Važna razlika u odnosu na linearne filtere je to da ovaj filter može dobro sačuvati rubove i fine detalje EEG valnog oblika.

Ovaj filter se jednostavno implementira koristeći medfilt funkciju iz SciPy biblioteke:

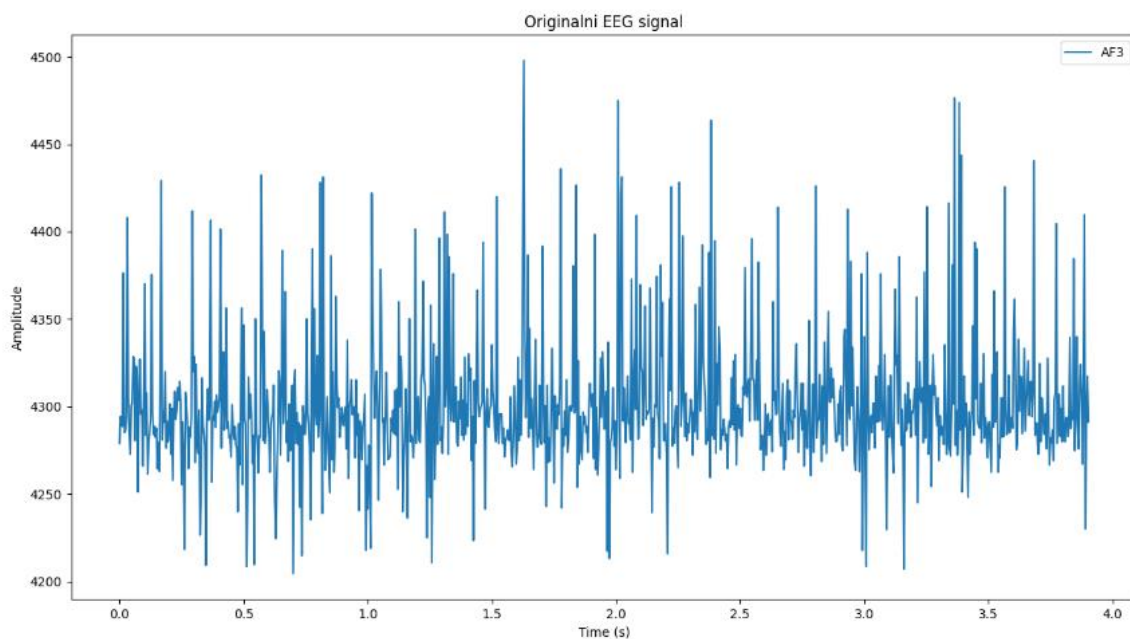
```
def apply_median_filter(data, kernel_size=3):  
    return data.apply(lambda x: medfilt(x, kernel_size), axis=0)
```

U našoj funkciji `apply_median_filter` su dva ulazna parametra, `data` i `kernel_size`. Kernel size se odnosi na veličinu prozora koji se koristi za izračunavanje srednje (median) vrijednosti susjednih uzoraka. Na primjer, kako je zadana vrijednost 3, ovaj filter će u obzir uzeti trenutni uzorak i po jedan uzorak prije i poslije njega za izračunavanje srednje vrijednosti. Iz ovog razloga, `kernel_size` mora biti neparan broj, inače je nemoguće primijeniti filter. Zbog zadržavanja i propuštanja informacija, vrlo je važno da se ova vrijednost izabere pažljivo kako filter ne bi odstranio korisne informacije.

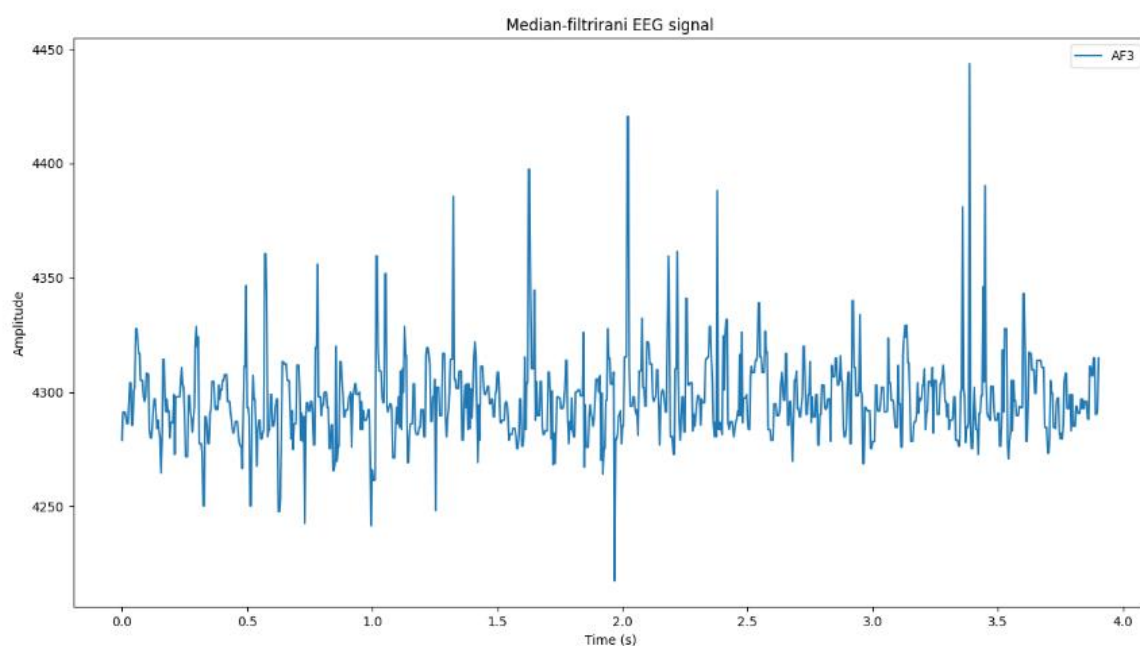
Program za primjenu filtra:

```
def main():  
    file_path = 'train.csv'  
    eeg_data = load_eeg_data(file_path)  
    filtered_data = apply_median_filter(eeg_data)  
  
    if eeg_data is not None:  
        plot_eeg(eeg_data, channels=['AF3'], start=0, end=1000, sampling_rate=256,  
title='Originalni EEG signal')  
        plot_eeg(filtered_data, channels=['AF3'], start=0, end=1000,  
sampling_rate=256, title='Median-filtrirani EEG signal')  
  
if __name__ == '__main__':  
    main()
```

Pokretanjem dobijamo sljedeće grafove:

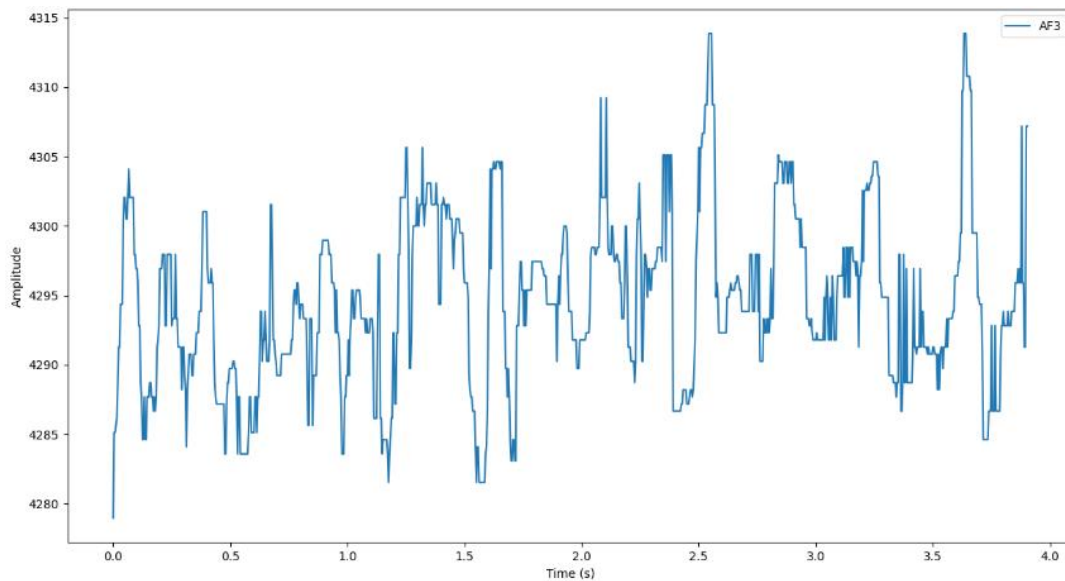


Slika 8: Originalni EEG signal za AF3 elektrodu u vremenskom periodu 4s



Slika 13: Median-filtrirani EEG signal za AF3 elektrodu u vremenskom periodu 4s
(kernel_size = 3)

Može se primijetiti da se na drugom grafiku nalazi dosta manje 'špiceva' (spikes) što ukazuje na to da je dosta šumova odstranjeno ovim filtrom, ali da sa ovom kernel_size vrijednošću ovaj filter i nije najefikasniji. Ipak, veća vrijednost kernel_size može dovesti do zamagljivanja detalja u originalnom signalu, što vidimo da je slučaj za kernel_size = 19:



Slika 13: Median-filtrirani EEG signal za AF3 elektrodu u vremenskom periodu 4s (kernel_size = 19)

4. PREDPROCESIRANJE EEG SIGNALA

Nakon analize i testiranja djelovanja različitih filtera na dataset, najbolje se pokazao notch filter. Ovaj filter koristimo kako bismo uklonili frekventni šum (target frekvencija 50Hz), te će se u svrhu dalje pripreme podataka za neuralnu mrežu koristiti upravo taj filter.

```
def preprocess_data(data, fs=250):
    # Drop rows with any NaN values
    data = data.dropna()

    # Extract labels
    y = data['eyeDetection'].values

    # Apply Notch filter to remove power line noise at 50Hz
    filtered_data = notch_filter(data.drop(columns=['eyeDetection']).values, 50,
    fs)

    # Extract statistical features
    statistical_features = extract_statistical_features(filtered_data)

    # Apply Wavelet Transform
    wavelet_features = wavelet_transform(filtered_data)

    # Apply Fourier Transform
    fourier_features = fourier_transform(filtered_data)

    # Combine all features
    X = np.hstack((statistical_features, wavelet_features, fourier_features))

    # Standardize the features
    scaler = StandardScaler()
    X = scaler.fit_transform(X)
    return X, y
```

Potrebne biblioteke za ovu funkciju su NumPy, Pandas, SciPy i PyWavelets.

5. EKSTRAKCIJA ZNAČAJKI I ANALIZA SIGNALA

Faza ekstrakcije značajki ima za cilj maksimizaciju omjera signala i šuma, te preraditi podatke kako bi daljnja obrada bila što efikasnija. Ova faza predstavlja ključnu etapu koja direktno utiče na postupak klasifikacije.

Kao dobar izbor pokazale su se kombinacija Wavelet i Fourierove transformacije uz praćenje pojedinih statističkih značajki. Na kraju je izvršena i standardizacija značajki kako bi se pružio bolji uvid u samu distribuciju signala.

5.1 Ekstrakcija statističkih značajki

Ekstrakcija statističkih značajki daje osnovne informacije o obliku, distribuciji i rasipanju signala koji se obrađuje. Slična svrha je funkcije [extract_statistical_features](#), koja kao parametar prima dataset koji se obrađuje, te nakon izvršene analize i obrade signala, vraća 2D niz u kojem su sadržane potrebne statističke informacije.

Varijable koje su korištene pri analizi EEG signala su:

- **mean**: srednja vrijednost signala po osi 1 (po svakom uzorku),
- **std**: standardna devijacija (mjera rasipanja signala),
- **min_val**: minimalna vrijednost signala,
- **max_val**: maksimalna vrijednost signala,
- **median**: medijan signala (centralna vrijednost),
- **q25**: 25. percentil, vrijednost ispod koje pada 25% uzoraka,
- **q75**: 75. percentil, vrijednost ispod koje pada 75% uzoraka,
- **iqr**: interkvartilni opseg, razlika između 75. i 25. percentila,
- **skewness**: koeficijent asimetrije, pokazuje koliko je distribucija signala asimetrična,
- **kurtosis**: kurtosis, mjera "šiljatosti" distribucije signala,
- **features**: kombinacija svih gore navedenih karakteristika u jedan 2D niz gdje je svaki red niz karakteristika za jedan uzorak.

```
def extract_statistical_features(data):
    mean = np.mean(data, axis=1)
    std = np.std(data, axis=1)
    min_val = np.min(data, axis=1)
    max_val = np.max(data, axis=1)
    median = np.median(data, axis=1)
    q25 = np.percentile(data, 25, axis=1)
    q75 = np.percentile(data, 75, axis=1)
    iqr = q75 - q25
    skewness = np.apply_along_axis(lambda x: pd.Series(x).skew(), 1, data)
    kurtosis = np.apply_along_axis(lambda x: pd.Series(x).kurt(), 1, data)
    features = np.vstack((mean, std, min_val, max_val, median, q25, q75, iqr,
skewness, kurtosis)).T
    return features
```

5.2 Transformacije

5.2.1 Wavelet transformacija

Na obrađeni EEG dataset primjenjuje se diskretna wavelet transformacija.

Wavelet transformacija se pokazala kao dobra pri analizi nekonzistentnih obrazaca podataka, budući da koristi varijabilne prozore. Dobro opisuje svojstva signala unutar određenog frekvencijskog i lokaliziranog vremenskog domena. Generalno, wavelet je tehnika spektralne procjene pri kojoj se bilo koja funkcija izražava kao beskonačni niz waveleta. Definisana funkcija `wavelet_transform` kao parametre prima sljedeće vrijednosti:

- **data** - podaci koje je potrebno obraditi,
- **wavelet** - tip waveleta koji se koristi (u projektu je odabran `db5` što označava Daubechies wavelet sa 5 nultih momenata)
- **level** - nivo dekompozicije, određuje koliko puta će signal biti dekomponovan.

U `coeffs` se sačuvaju koeficijenti wavelet dekompozicije za svaki signal iz `data`. `features` predstavljaju kombinaciju koeficijenata svih nivoa dekompozicije za svaki uzorak signala, ovo je ujedno i vrijednost koju funkcija vraća.

```
def wavelet_transform(data, wavelet='db5', level=4):  
    coeffs = [pywt.wavedec(d, wavelet, level=level) for d in data]  
    features = np.array([np.concatenate(c) for c in coeffs])  
    return features
```

5.2.2 FFT (Fast Fourier Transform)

Funkcija `fourier_transform` je funkcioniše slično kao i wavelet, na proslijeđenom parametru `data` vrši prigodnu transformaciju, te u povratnoj vrijednosti `fft_features` skladišti informacije o amplitudama Fourier-ove transformacije za svaki signal. Korištena podfunkcija `fft` omogućava prelazak iz vremenskog u frekvencijski domen.

```
def fourier_transform(data):  
    fft_features = np.abs(fft(data, axis=1))  
    return fft_features
```

Kombinacijom waveleta i FFT-a dobijamo bolje izražene karakteristike EEG signala, budući da je wavelet generalno bolji pri detekciji naglih i kratkotrajnih promjena u signalu, a FFT je ipak prigodniji za identifikaciju dominantnih frekvencija tokom dužih perioda.

6. KLASIFIKACIJA I NEURALNA MREŽA

Potrebni paketi za instalaciju su:

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Dropout, Conv1D, MaxPooling1D, Flatten
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import KFold
```

Konvolucijska neuralna mreža (engl. CNN) se ističe u odnosu na druge algoritme mašinskog učenja budući da može autonomno ekstrahovati značajke u velikom obimu, čime se zaobilazi potreba za ručnim inženjeringom značajki i na taj način poboljšava efikasnost.

Arhitektura CNN-a sadrži sljedeće nivoe mreže:

- konvolucijski
- **MaxPooling**
- dropout i
- dense nivoe.

6.1 Kreiranje i arhitektura CNN-a

Definisana je funkcija **create_cnn_model** koja na osnovu proslijeđenog parametra *input_shape* kreira konvolucijsku neuralnu mrežu (engl. *Convolutional Neural Network - CNN*). Inicijalizacija modela na `Sequential()` uz pomoć `tensorflow.keras.Sequential` postizemo slaganje slojeva jedan po jedan, sekvencijalno. Nakon inicijalizacije, potrebno je dodavati konvolucijske slojeve:

```
model.add(Conv1D(64, kernel_size=3, activation='relu', padding='same',
input_shape=input_shape))
```

Navedeni isječak koda dodaje 1D konvolucijski sloj sa 64 filtera, *kernel_size* postavlja na 3.

Model sadrži ukupno tri 1D konvolucijska sloja, sa filterima 64, 128 i 256 respektivno (ostali parametri ostaju neizmijenjeni). Nakon svakog konvolucijskog sloja potrebno je dodati 1D MaxPooling sloj sa priloženim parametrima. Uloga MaxPoolinga jeste smanjivanje dimenzionalnosti podataka, a da pritom zadrži najvažnije informacije. Ovime se sprječava pojava overfitting-a (prevelike usklađenosti podataka, do nje dolazi kada model). Dropout sprječava pretreniravanje modela nasumičnim isključivanjem neurona tokom treninga. Postavljanjem parametra na 0.3 postiže se nasumično isključivanje 30% ulaznih jedinica.

```
model.add(MaxPooling1D(pool_size=2, strides=1, padding='same'))
model.add(Dropout(0.3))
```

Finalna dorada arhitekture modela, `Flatten()` transformiše multidimenzionalne izlaze u jednolinijski vektor za ulaz u potpuno povezane slojeve, te priprema za upotrebu `Dense` - omogućava sloju da nauči nelinearne kombinacije karakteristika. Postavka **activation='sigmoid'** se koristi za binarnu klasifikaciju, daje izlaz između 0 i 1.


```

model.add(Dropout(0.4))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

```

U završnoj fazi vrši se optimizacija modela upotrebom Adam optimizatora (koji koristi adaptivno učenje) iz `tensorflow.keras.optimizers`, te kompajliranje i praćenje tačnosti samog modela. **'Binary_crossentropy'** je prikladan izbor za binarnu klasifikaciju.

```

optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='binary_crossentropy',
metrics=['accuracy'])

```

Potpuni kod za kreiranje pomenutog CNN-a u prilogu:

```

def create_cnn_model(input_shape):
    model = Sequential()
    model.add(Conv1D(64, kernel_size=3, activation='relu', padding='same',
input_shape=input_shape))
    model.add(MaxPooling1D(pool_size=2, strides=1, padding='same'))
    model.add(Dropout(0.3))
    model.add(Conv1D(128, kernel_size=3, activation='relu', padding='same'))
    model.add(MaxPooling1D(pool_size=2, strides=1, padding='same'))
    model.add(Dropout(0.3))
    model.add(Conv1D(256, kernel_size=3, activation='relu', padding='same'))
    model.add(MaxPooling1D(pool_size=2, strides=1, padding='same'))
    model.add(Dropout(0.4))
    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(1, activation='sigmoid'))
    optimizer = Adam(learning_rate=0.001)
    model.compile(optimizer=optimizer, loss='binary_crossentropy',
metrics=['accuracy'])
    return model

```

Dakle, ova CNN mreža je dizajnirana za binarnu klasifikaciju jednodimenzionalnih podataka. Ima tri konvolucijska sloja, svaki praćen `'MaxPooling1D'` slojem i `'Dropout'` slojem, što pomaže u smanjenju pretreniranja. Nakon konvolucijskih slojeva, podaci se poravnavaju i propuštaju kroz gust sloj sa 512 neurona, nakon čega slijedi izlazni sloj sa sigmoid aktivacijom za binarnu klasifikaciju. Model je kompajliran sa Adam optimizatorom i binarnom funkcijom gubitka.

6.2 Callback rano zaustavljanje

Funkcija `custom_early_stopping` se koristi za zaustavljanje treniranja modela prije nego što se završi predefinirani broj epoha ukoliko se zadovolji određeni uvjet tačnosti. Svrha ove funkcije je poboljšati efikasnost treniranja tako što će zaustaviti treniranje kada se postigne zadovoljavajući nivo performansi, čime se štedi vrijeme i resursi.

Prosljeđuju se sljedeći parametri:

- **History** - predstavlja objekt koji sadrži zapise o metrikama treninga za svaku epohu modela,
- **Threshold** - prag tačnosti koji model treba postići da bi se zaustavilo treniranje (postavljeno na 0.94, dakle kada dostigne tačnost od barem 94% model prestaje sa treniranjem).

Funkcija vraća True ukoliko je došlo do rane obustave, u protivnom vraća False.

```
def custom_early_stopping(history, threshold=0.94):  
    accuracies = history.history['accuracy']  
    if any(acc >= threshold for acc in accuracies):  
        return True  
    return False
```

7. ANALIZA GLAVNOG PROGRAMA

Glavni dio programa, odnosno main, sastoji se od nekoliko primarnih cjelina:

1. **Učitavanje podataka** iz EEG_Eye_State_Classification.csv file-a i postavljanje frekvencije uzorkovanja ($fs = 250$)
2. **Preprocesiranje podataka** (preprocess_data funkcija, podrazumijeva korištenje notch filtera, wavelet i FFT transformacije i već pomenute obrade signala),
3. **Preoblikovanje podataka** kako bi bili pogodni za CNN format podataka,
4. **Kreiranje KFold objekta za unakrsnu validaciju s 5 podjela** (folds), te se inicijalizira lista za spremanje tačnosti modela na svakom foldu (ovime je procjena tačnosti modela pouzdanija, svaki uzorak podataka koristi se i za treniranje i za validaciju, što omogućava sveobuhvatniju i precizniju procjenu performansi modela),
5. **Kreiranje i treniranje modela**
 1. Podjela podataka na training i test setove,
 2. Treniranje modela do 100 epoha i provjera da li je potreban callback (rana obustava treniranja),
 3. Bilježi se evaluacija modela u listu,
6. **Finalna evaluacija modela**,
7. **Grafički prikaz (plot) tačnosti modela**,
8. **Spremanje modela za daljnju upotrebu i dodatna testiranja**.

Glavni dio programa:

```
def main():
    # učitavanje dataseta
    data = load_data('EEG_Eye_State_Classification.csv')
    fs = 250 #frekvencija uzorkovanja

    X, y = preprocess_data(data, fs)

    # priprema podataka za CNN
    X = X.reshape((X.shape[0], X.shape[1], 1))

    # KFold cross-validation
    kfold = KFold(n_splits=5, shuffle=True, random_state=42)
    accuracies = []

    for train_index, test_index in kfold.split(X):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

        # kreiranje i treniranje modela
        model = create_cnn_model(X_train.shape[1:])

        for epoch in range(100): # manuelna iteracija po epohama
            history = model.fit(X_train, y_train, epochs=1, batch_size=32,
validation_data=(X_test, y_test))
            if custom_early_stopping(history, threshold=0.94):
                print(f"Early stopping at epoch {epoch+1}")
                break

        # evaluacija modela
        loss, accuracy = model.evaluate(X_test, y_test)
        accuracies.append(accuracy)
        print(f"Fold Accuracy: {accuracy}")

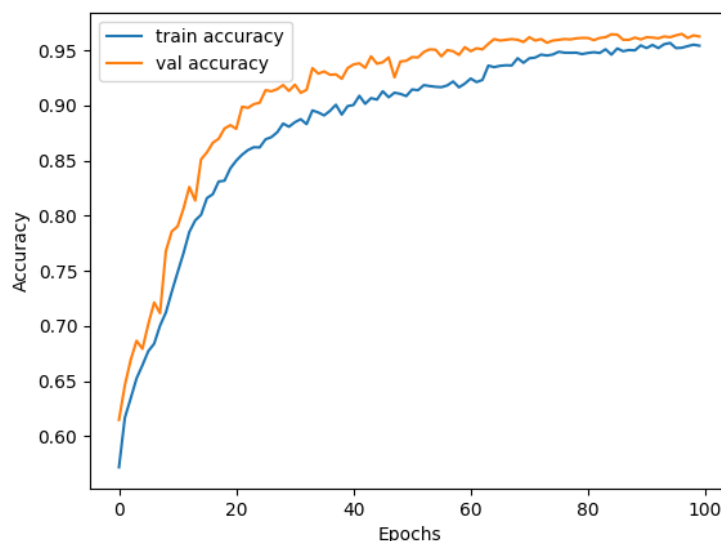
    print(f"Mean Cross-Validation Accuracy: {np.mean(accuracies)}")

    # plot sa prikazom tacnosti
    plt.plot(history.history['accuracy'], label='train accuracy')
    plt.plot(history.history['val_accuracy'], label='val accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()

    # sacuvati model za testiranje
    model.save('trained_model.h5')

if __name__ == "__main__":
    main()
```

8. GRAFIČKA ANALIZA USPJEHA NEURALNE MREŽE



Slika 14: Graf koji prikazuje tačnost treniranja i validacijsku tačnost
U prilogu je graf koji prikazuje tačnost treniranja (**train accuracy**) i validacijsku tačnost (**val accuracy**) kroz epohe treniranja CNN-a (Slika 14).

Značajne karakteristike koje možemo očitati sa grafa su:

1. **Porast tačnosti kroz epohe**

Tačnost na trening setu postepeno raste sa porastom epohe, što ukazuje na prilagođavanje modela podacima. Tačnost na validacijskom setu također raste, ali malo brže nego na trening setu u početnim epohama, i dostiže vrhunac nakon otprilike 40-60 epoha.

2. **Konvergencija**

I train i val accuracy konvergiraju prema kraju epoha, iz čega zaključujemo da model postiže stabilnu tačnost i na trening i na validacijskom setu.

3. **Prag tačnosti 0.94**

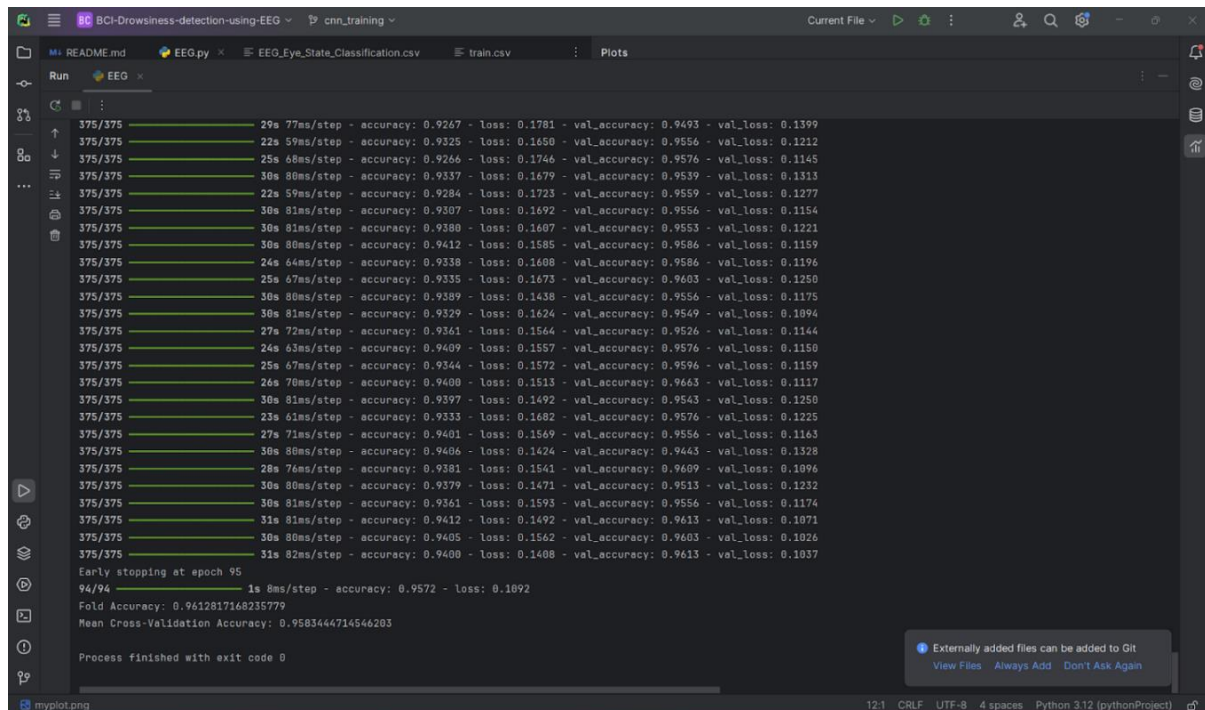
Validacijska tačnost dostiže i prelazi željeni prag od 0.94 nakon približno 25 epoha (na ovo utiče `custom_early_stopping` funkcija).

4. **Moguć blagi overfitting (pretreniranje)**

Val accuracy je veća od train nakon prvih 20-30 epoha, što može ukazivati na blagi overfitting. Ipak, razlika nije isuviše velika, te se može zaključiti da model dobro generalizira na validacijskom setu.

8.1 Performanse modela kroz epohe

Priložen je manji uvid u performanse samog modela po epohama neposredno pred terminiranje. Za svaku epohu je prikazana tačnoost (accuracy) i gubitak (loss). Primjetno je da tačnost postepeno raste, a gubitak postepeno opada, što indicira da model uspješno uči iz datih EEG podataka. Rano zaustavljanje se dogodilo u 95. epohi, budući da je tada validacijska tačnost prevazišla prag od 0.94 (time je ispunjen uslov u funkciji za terminiranje `custom_early_stopping`). Prosječna tačnost kroz svih 5 foldova KFold unakrsne validacije je 0.96128, što je indikacija konzistentnih performansi modela. Na kraju treniranja, model je postigao **tačnost od 0.9583 na testnom skupu, s gubitkom od 0.1092**.



```
375/375 29s 77ms/step - accuracy: 0.9267 - loss: 0.1781 - val_accuracy: 0.9493 - val_loss: 0.1399
375/375 22s 59ms/step - accuracy: 0.9325 - loss: 0.1650 - val_accuracy: 0.9556 - val_loss: 0.1212
375/375 25s 68ms/step - accuracy: 0.9266 - loss: 0.1746 - val_accuracy: 0.9576 - val_loss: 0.1145
375/375 30s 88ms/step - accuracy: 0.9337 - loss: 0.1679 - val_accuracy: 0.9539 - val_loss: 0.1313
375/375 22s 59ms/step - accuracy: 0.9284 - loss: 0.1723 - val_accuracy: 0.9559 - val_loss: 0.1277
375/375 30s 81ms/step - accuracy: 0.9307 - loss: 0.1692 - val_accuracy: 0.9556 - val_loss: 0.1154
375/375 30s 81ms/step - accuracy: 0.9380 - loss: 0.1607 - val_accuracy: 0.9553 - val_loss: 0.1221
375/375 30s 80ms/step - accuracy: 0.9412 - loss: 0.1585 - val_accuracy: 0.9586 - val_loss: 0.1159
375/375 24s 64ms/step - accuracy: 0.9338 - loss: 0.1608 - val_accuracy: 0.9586 - val_loss: 0.1194
375/375 25s 67ms/step - accuracy: 0.9335 - loss: 0.1673 - val_accuracy: 0.9603 - val_loss: 0.1250
375/375 30s 80ms/step - accuracy: 0.9389 - loss: 0.1438 - val_accuracy: 0.9556 - val_loss: 0.1175
375/375 30s 81ms/step - accuracy: 0.9329 - loss: 0.1624 - val_accuracy: 0.9549 - val_loss: 0.1094
375/375 27s 72ms/step - accuracy: 0.9361 - loss: 0.1544 - val_accuracy: 0.9526 - val_loss: 0.1144
375/375 24s 63ms/step - accuracy: 0.9409 - loss: 0.1557 - val_accuracy: 0.9576 - val_loss: 0.1150
375/375 25s 67ms/step - accuracy: 0.9344 - loss: 0.1572 - val_accuracy: 0.9596 - val_loss: 0.1159
375/375 26s 70ms/step - accuracy: 0.9400 - loss: 0.1513 - val_accuracy: 0.9663 - val_loss: 0.1117
375/375 30s 81ms/step - accuracy: 0.9397 - loss: 0.1492 - val_accuracy: 0.9543 - val_loss: 0.1250
375/375 23s 61ms/step - accuracy: 0.9333 - loss: 0.1682 - val_accuracy: 0.9576 - val_loss: 0.1225
375/375 27s 71ms/step - accuracy: 0.9401 - loss: 0.1569 - val_accuracy: 0.9556 - val_loss: 0.1163
375/375 30s 88ms/step - accuracy: 0.9406 - loss: 0.1424 - val_accuracy: 0.9443 - val_loss: 0.1328
375/375 28s 76ms/step - accuracy: 0.9381 - loss: 0.1541 - val_accuracy: 0.9609 - val_loss: 0.1096
375/375 30s 80ms/step - accuracy: 0.9379 - loss: 0.1471 - val_accuracy: 0.9513 - val_loss: 0.1232
375/375 30s 81ms/step - accuracy: 0.9361 - loss: 0.1593 - val_accuracy: 0.9556 - val_loss: 0.1174
375/375 31s 81ms/step - accuracy: 0.9412 - loss: 0.1492 - val_accuracy: 0.9613 - val_loss: 0.1071
375/375 30s 80ms/step - accuracy: 0.9405 - loss: 0.1542 - val_accuracy: 0.9603 - val_loss: 0.1026
375/375 31s 82ms/step - accuracy: 0.9400 - loss: 0.1408 - val_accuracy: 0.9613 - val_loss: 0.1037

Early stopping at epoch 95
94/94 1s 8ms/step - accuracy: 0.9572 - loss: 0.1092
Fold Accuracy: 0.9612817168235779
Mean Cross-Validation Accuracy: 0.958344714546203

Process finished with exit code 0
```

Slika 15: Performanse modela kroz epohe

Kompletnan source code je moguće pronaći na linku <https://github.com/inferno73/EEG-drowsiness-detection>.

9. ZAKLJUČAK

Projekat nastoji prikazati prigodne metode za procesiranje EEG signala, uz upotrebu raznih filtera i transformacija, te izvršiti usporedbu istih i detaljniju analizu. Ekstrakcijom značajki nastoji se izvršiti dodatno procesiranje i poboljšanje podataka, praveći ih što adekvatnijim za obradu neuralne mreže. Pretpostavlja se korištenje konvolucijske neuralne mreže kao glavnog klasifikatora. Model pokazuje impresivne performanse sa visokim tačnostima na trening i validacijskom setu, te niskim gubicima. Rano zaustavljanje je efikasno spriječilo pretreniranje, osiguravajući da model generalizira dobro na neviđenim podacima. Prosječna tačnost unakrsne validacije od 0.9583 potvrđuje pouzdanost i stabilnost modela.

Rezultati su zadovoljavajući, ali ostavljaju prostora za dodatno poboljšanje mreže. Za daljnja poboljšanja, može se razmotriti:

1. Fino podešavanje hiperparametara kao što su veličina batcha, stopa učenja, i konfiguracija modela.
2. Isprobavanje različitih arhitektura CNN-a ili dodatnih slojeva.
3. Eksperimentiranje s različitim metodama predobrade podataka i ekstrakcije značajki.

Tako utreniranu mrežu bi pogodno bilo povezati sa odgovarajućim interfejsom za upravljanje, formirajući jedan kompletan BCI (*engl. Brain Computer Interface*) model.

10. REFERENCE

- [1] S. Arif, S. Munawar, and H. Ali, "Driving Drowsiness Detection Using Spectral Signatures of EEG-based Neurophysiology," *Frontiers in Physiology*, vol. 14, Mar. 2023, doi: <https://doi.org/10.3389/fphys.2023.1153268>.
- [2] IBM, "What are Convolutional Neural Networks? | IBM," *www.ibm.com*, 2023. <https://www.ibm.com/topics/convolutional-neural-networks>
- [3] H. U. Amin *et al.*, "Feature extraction and classification for EEG signals using wavelet transform and machine learning techniques," *Australasian Physical & Engineering Sciences in Medicine*, vol. 38, no. 1, pp. 139–149, Feb. 2015, doi: <https://doi.org/10.1007/s13246-015-0333-x>.
- [4] Z. Kelta, "An Introduction to Convolutional Neural Networks (CNNs)," *DataCamp*, Nov. 2023. <https://www.datacamp.com/tutorial/introduction-to-convolutional-neural-networks-cnns> (accessed Jun. 10, 2024).