Politechnika Wrocławska

Wydział Informatyki i Telekomunikacji

Głosowa identyfikacja osób

Skład grupy:
Alekseyenko Kristina
Cegielski Szymon
Łupicki Łukasz
Niemczyk Krzysztof

Październik 2021

Spis treści

1 Wstęp **3**
1.1 Wstęp teoretyczny . . . 3
1.2 Założenia projektowe . . . 3

2 Materiały **4**
2.1 Sprzęt . . . 4
2.2 Oprogramowanie . . . 4

3 Analiza i porównanie czujników. Wybór oprogramowania **5**
3.1 Wybór mikrofonu do projektu . . . 5
3.2 Wybór oprogramowania do analizy dźwięku . . . 7
3.3 wykorzystane biblioteki Python . . . 7

4 Analiza próbek dźwięku **8**
4.1 Analiza porównawcza technik rozpoznawania mowy . . . 8
4.2 Analiza dźwięku . . . 8
4.2.1 MFCC . . . 8
4.2.2 STFT . . . 9
4.2.3 Chromagram . . . 9
4.2.4 Mel-scaled spectrogram . . . 9
4.2.5 Spectral contrast . . . 9
4.2.6 Tonnetz . . . 10
4.3 Instalacja bibliotek Arduino . . . 10
4.3.1 biblioteka TMRpcm . . . 10
4.3.2 Pozostałe biblioteki . . . 10

5 Budowa modelu do rozpoznawania głosu **11**

6 Instalacja **14**
6.1 Oprogramowanie . . . 14
6.2 Schemat podłączeniowy . . . 14
6.3 Rozpakowanie programu . . . 14
6.4 Instalacja Arduino . . . 14
6.5 Instalacja Python . . . 15
6.6 Import bibliotek Python . . . 15
6.7 Uczenie modelu . . . 15
6.8 Uruchomienie programu . . . 16

7 Struktura programu **17**
7.1 main.py 17
7.2 script.py 17
7.3 nagrania 17
7.4 Arduino.ino 17

8 Działanie programu **18**

9 Napotkane problemy **19**
9.1 Wywoływanie skryptów Arduino UNO w Python 19
9.2 Ograniczenia dotyczące karty microSD 19
9.3 Bezpośredni odczyt z komputera plików zapisanych w arduino. Szybkość przesyłania przez serial 19
9.4 Ograniczenia bufora 20
9.5 Kodowanie pliku 20
9.6 Problem z rozmiarem karty pamięci 20
9.7 Utrata napięcia 21
9.8 Jakość próbek 21
9.9 Ilość próbek 21

10 Wnioski **22**

1 Wstęp

1.1 Wstęp teoretyczny

Rozpoznawanie głosu staje się coraz powszechniejszą funkcjonalnością w naszych urządzeniach. Korzystamy z niej w telefonach, samochodach oraz coraz częściej także w naszych domach.

Rozpoznawanie głosu (Voice Recognition) to technologia, która umożliwia komputerowi na zidentyfikowanie osoby mówiącej na bazie analizy jej głosu. Zastosowanie Voice Recognition pozwala na aktywację wybranych funkcji interfejsu poprzez wypowiadanie słów aktywacyjnych jedynie przez osobę do tego upoważnioną.

1.2 Założenia projektowe

Celem projektu jest stworzenie oprogramowania oraz konfiguracji sprzętowej służącej do identyfikacji głosowej osób.

- Dźwięk będzie pobierany z mikrofonu ze wzmacniaczem MAX9814 podłączonego do mikrokontrolera Arduino
- Dźwięk będzie nagrywany tylko podczas trzymania wciśniętego przycisku, po zwolnieniu będzie zapisywany do pliku w formacie *.wav
- Pliki z nagranym dźwiękiem będą przesyłane na komputer, który będzie połączony kablem USB z Arduino
- Klasyfikator będzie napisany w języku Python i będzie znajdować się na komputerze,
- Aplikacja będzie rozpoznawać wybrane osoby przy użyciu klasyfikatora i przesyłać wynik w postaci nazwy klasy (użytkownika) z powrotem do mikrokontrolera
- Na podstawie zwróconego wyniku będzie się zapalała dioda o odpowiednim do rozpoznanej osoby kolorze

2 Materiały

Do stworzenia projektu będą wykorzystane poniższe elementy oraz oprogramowanie:

2.1 Sprzęt

- Arduino UNO R3
- Rezystor 220 Ω - 6 sztuk
- Dioda LED 5mm, różne kolory soczewek - 5 sztuk
- Kabel USB
- Komputer stacjonarny/laptop
- Mikrofon MAX9814
- Płytka stykowa
- Przewody połączeniowe - 20 sztuk
- Przycisk tact switch 12x12x7,3
- Czytnik do karty microSD + karta microSD

2.2 Oprogramowanie

- Python
- Arduino IDE

3 Analiza i porównanie czujników. Wybór oprogramowania

3.1 Wybór mikrofonu do projektu

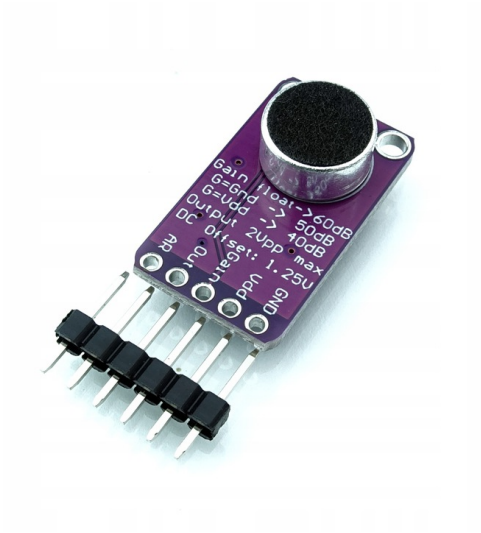
Na potrzeby naszego projektu, do rejestracji dźwięku, będziemy potrzebować mikrofonu o odpowiednich parametrach. Jednym z najważniejszych parametrów jest jego zakres częstotliwości.

Częstotliwość próbkowania określa liczbę próbek sygnału ciągłego na sekundę uwzględnionych w syntezie sygnału dyskretnego. Częstotliwość próbkowania przetwornika A/C bazuje na zegarze próbkującym wyznaczającym chwile czasowe w których przetwornik przetwarza chwilową wartość analogową na cyfrową.

Częstotliwość próbkowania musi być przynajmniej dwa razy większa niż największa częstotliwość sygnału próbkowanego.

Z twierdzenia o próbkowaniu wynika, że pełne odtworzenie sygnału ciągłego w czasie jest możliwe, jeśli częstotliwość próbkowania sygnału cyfrowego jest wyższa niż dwukrotność najwyższej częstotliwości sygnału próbkowanego, lub innymi słowy, kiedy częstotliwość Nyquista (połowa częstotliwości próbkowania) jest wyższa niż najwyższa częstotliwość sygnału próbkowanego. Jeśli sygnał jest próbkowany z niższą częstotliwością, to informacja zawarta w oryginalnym sygnale (analogowym) może nie być w całości odtworzona na podstawie sygnału dyskretnego. Przykładowo, jeśli górna granica pasma sygnału wynosi 100 Hz, to częstotliwość próbkowania większa niż 200 Hz jest wystarczająca, aby uniknąć zjawiska aliasingu i w pełni odtworzyć oryginalny sygnał.

Do naszego projektu wybraliśmy mikrofon MAX9814, ponieważ wyposażony jest on w automatyczną kontrolę wzmocnienia oraz szeroki zakres częstotliwości (20Hz - 20kHz) który zgodnie z twierdzeniem o próbkowaniu z łatwością podwójnie przewyższa zakres częstotliwości generowany przez człowieka (125 – 8000Hz).



Rysunek 1: Mikrofon MAX9814

Specyfikacja:

- Napięcie zasilania: od 2,7 V do 5,5 V
- Zakres częstotliwości: od 20 Hz do 20 kHz
- Programowalna wartość Attack/Release ratio
- Maksymalna wartość wzmocnienia: 40 dB / 50 dB / 60 dB (domyślnie)
- Automatyczne wzmocnienie

3.2 Wybór oprogramowania do analizy dźwięku

3.3 wykorzystane biblioteki Python

Wykorzystane biblioteki:

- pandas
- scikit-learn
- librosa
- numpy
- keras
- tensorflow

Powyższe biblioteki zostały wybrane z kilku powodów. Pandas służy do odpowiedniego wczytania danych (w tym przypadku są to pliki *.wav), scikit-learn do trenowania naszego modelu, librosa jako biblioteka do wyznaczenia cech charakterystycznych z plików dźwiękowych oraz ich transformacji, numpy do pracy na array'ach, keras i tensorflow do stworzenia modelu sieci neuronowych.

4 Analiza próbek dźwięku

4.1 Analiza porównawcza technik rozpoznawania mowy

Podczas formułowania założeń projektowych natrafiliśmy na dylemat związany z wyborem techniki, którą zastosujemy do rozpoznawania mowy. Braliśmy pod uwagę metody graficzne, które analizują spektrogram, w głównej mierze, przy pomocy metod statystycznych takich jak ukryte modele Markova, sieci Bayesowskie, analiza dyskryminacyjna, rozkład Gaussa i wiele innych. Cechą większości z tych metod jest wyszukiwanie pewnych prawidłowości statystycznych bądź występowań charakterystycznych momentów w spektrogramie. Z drugiej strony mamy do wyboru metody analizujące plik dźwiękowy, które przetwarzają dźwięk przy pomocy zadanych metod i na tej podstawie wyodrębniają cechy szczególne. Istnieje wiele transformacji dźwięku i metod jego obróbki, jednak my zdecydowaliśmy się wykorzystać MFCC, STFT, Chromagram, Mel-scaled spectrogram, Spectral contrast i Tonnetz. Nasza decyzja została podjęta ze względu na trudności implementacyjne modeli graficznych oraz konieczność generowania spektrogramu jako danych wejściowych. Na korzyść metod przetwarzających dźwięk należy również wspomnieć o bibliotece librosa, która ma w sobie zaimplementowane metody wyodrębniające te cechy co znacznie ułatwi stworzenie modelu sieci do rozpoznawania mowy.

4.2 Analiza dźwięku

W celu wydobycia cech charakterystycznych dla poszczególnych głosów zostanie wykorzystana biblioteka librosa, która będzie analizować dźwięk nagrania na podstawie następujących cech:

4.2.1 MFCC

Parametry mel-cepstralne (ang. MFCC – MelFrequency Cepstral Coefficients) to parametry szeroko stosowane w akustyce mowy oraz w kompresji sygnałów fonicznych. Powstają z cepstrum sygnału przedstawionego w skali melowej (mel-cepstrum). Uzyskuje się je w następujący sposób:

- Przeprowadza się transformatę Fouriera na zadanym okienku sygnału,
- Odwzorowuje się moce widma uzyskane powyżej na skalę mel, używając trójkątnych okien nakładających się lub alternatywnie, okien nakładających się kosinusowych,
- Sprawdza się logarytmy mocy przy każdej z częstotliwości mel,

- Przeprowadza się transformatę kosinusową logarytmów współczynników uzyskanych z filtracji sygnału, tak jakby były nowym sygnałem.

Wynikiem po powyższych krokach jest uzyskanie wektora parametrów mel-cepstralnych.

4.2.2 STFT

Krótkoczasowa transformata Fouriera (STFT – Short-Time Fourier Transform) stanowi wzorcowy przykład algorytmu analizy czasowo-częstotliwościowej. Umożliwia ona wydobycie z sygnału informacji o tym, jak zmienia się jego widmo w czasie, czyli jednoczesną obserwację jego właściwości zarówno w dziedzinie czasu jak i częstotliwości. Wycinek sygnału (blok próbek o rozmiarze L) przeznaczony do analizy jest sukcesywnie dzielony na segmenty, z których każdy podlega analizie widmowej niezależnie (DFT).

4.2.3 Chromagram

Chromagram jest transformacją czasowo-częstotliwościowych właściwości sygnału w czasowo zmienny prekursor wysokości dźwięku. Transformacja ta oparta jest na obserwacjach percepcyjnych dotyczących systemu słuchowego i wykazano, że posiada kilka interesujących własności matematycznych.Cechy oparte na chromacie, które są również określane jako "profile klas wysokości", są potężnym narzędziem do analizy dźwięku, gdzie dźwięki mogą być sensownie podzielone (często na dwanaście kategorii). Jedną z głównych właściwości cech chromatycznych jest to, że wychwytują one harmoniczną i melodyczną charakterystykę dźwięku, będąc jednocześnie odpornymi na zmiany barwy.

4.2.4 Mel-scaled spectrogram

Jest to spektrogram pomocą którego użytkownik jest w stanie zaobserwować dźwięki przekonwertowane na skalę mel. Skonstruowana jest ona w sposób, dzięki któremu dźwięki o równej odległości od skali mel są odbierane jako równe odległości od ludzi.

4.2.5 Spectral contrast

Polega on na różnicach decybeli między wierzchołkami i dolinami widma dźwięku. Jego zwiększenie może być pewną pomocą przy kompensowaniu skutków zmniejszonej selektywności częstotliwości. Może być ona również wykorzystana do redukcji szumów w momencie, gdy stosunek sygnału do szumu jest wystarczająco wysoki, a doliny są wypełnione sumami.

4.2.6 Tonnetz

Jest to konceptualny diagram siatkowy, przedstawiający przestrzeń tonalną. Przedstawione w nim wierzchołki reprezentują nuty, a trójkąty z nich powstałe tworzą akordy. W innym wypadku, wierzchołki mogą reprezentować akordy, a połączenia między nimi są prezentacją pojedynczego przejścia w tonacji głosu.

4.3 Instalacja bibliotek Arduino

4.3.1 biblioteka TMRpcm

Do obsługi zapisu dźwięku na karcie SD użyto bibliotekę TMRpcm. Umożliwia ona zapis pliku *.wav z mikrofonu (8-bit, 8-32kHz) na karcie SD. W celu instalacji należy dodać ją w programie Arduino IDE (Narzędzia->Zarządzaj Bibliotekami...). Po instalacji, w celu poprawnego działania, w lokalizacji zainstalowanej biblioteki (`C:\Users\...\Dokumenty\Arduino\libraries\TMRpcm`) trzeba edytować plik "pcmConfig.h". W pliku należy odkomentować poniższe linie kodu:

```
//#define buffSize 128  //must be an even number

//#define ENABLE_RECORDING

// #define BLOCK_COUNT 10000UL  // 10000 = 500MB   2000 = 100MB
```

Link do biblioteki:
`https://github.com/TMRh20/TMRpcm`

4.3.2 Pozostałe biblioteki

Arduino IDE -> Narzędzia -> Zarządzaj bibliotekami...

Dodatkowo zainstalowane biblioteki (oprócz domyślnych):

- SD wersja 1.2.4

Podczas podłączanie karty SD o pojemności 2GB występował problem. Karta była widoczna lecz wyświetlany był komunikat o nieprawidłowym sformatowaniu karty. Przy karcie o rozmiarze 8GB problem nie występował.

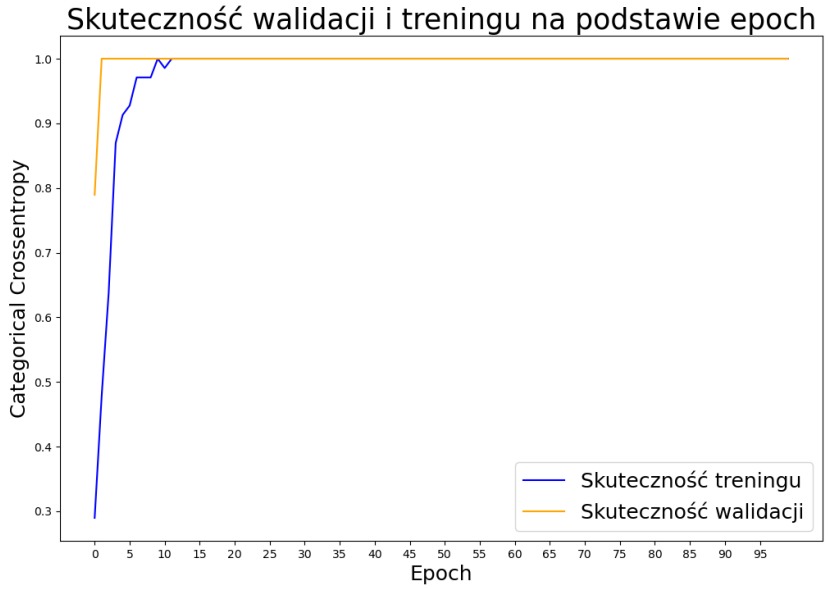
5 Budowa modelu do rozpoznawania głosu

Zdecydowaliśmy się na model sekwencyjny, ze względu na jego przewagę w problemach zależących od ilości klas. W naszym przypadku występują 4 klasy, które odpowiadają każdej z osób biorących udział w projekcie. Po ekstrakcji cech z naszych nagrań zostaje przeprowadzona wstępna obróbka danych za pomocą StandardScaler z biblioteki scikit-learn, by później załadować dane do naszego modelu. Jako dane uczące wykorzystaliśmy nagrania naszych głosów po 25 nagrań dla każdego. Dla każdej z osób ich czas wynosi około 8 minut.

```
# Dzielenie zbiorow na treningowy, testowy i walidacyjny
    X_train = X[:69]
    y_train = y[:69]
    X_val = X[70:89]
    y_val = y[70:89]
    X_test = X[90:99]
    y_test = y[90:99]

# Dopasowywanie danych
    ss = StandardScaler()
    X_train = ss.fit_transform(X_train)
    X_val = ss.transform(X_val)
    X_test = ss.transform(X_test)

# Utworzenie modelu
    model = Sequential()

# Dodawanie warstw
    model.add(Dense(193, input_shape=(193,), activation='relu',
    name='First'))
    model.add(Dropout(0.1))
    model.add(Dense(128, activation='relu', name='Second'))
    model.add(Dropout(0.25))
    model.add(Dense(128, activation='relu', name='Third'))
    model.add(Dropout(0.5))
    model.add(Dense(4, activation='softmax', name='Last'))

# Kompilacja modelu
    model.compile(loss='categorical_crossentropy',
    metrics=['accuracy'], optimizer='adam', run_eagerly=True)
```

```
    early_stop = EarlyStopping(monitor='val_loss', min_delta=0,
    patience=100, verbose=1, mode='auto')

    history = model.fit(X_train, y_train, batch_size=256, epochs=100,
                        validation_data=(X_val, y_val),
                        callbacks=[early_stop])

# Zapisanie modelu do pozniejszego uzytku
    model.save('saved_model')
    score = model.evaluate(X_test, y_test, verbose=0)
```

Po nauce naszego modelu, jego stopień poprawności predykcji wynosi 100, jednak jest to przekłamany wynik ze względu na małą liczbę danych uczących (po 25 nagrań dla osoby).



By nie uczyć modelu od nowa w celu sprawdzenia kolejnego nagrania, stworzyliśmy skrypt do predykcji w którym wystarczy załadować nagranie, a skrypt sprawdzi je pod kątem nauczonego i zapisanego wcześniej modelu sekwencjonalnego.

```
def extract_features():
    X, sample_rate = librosa.load(file_path, res_type='kaiser_fast')
```

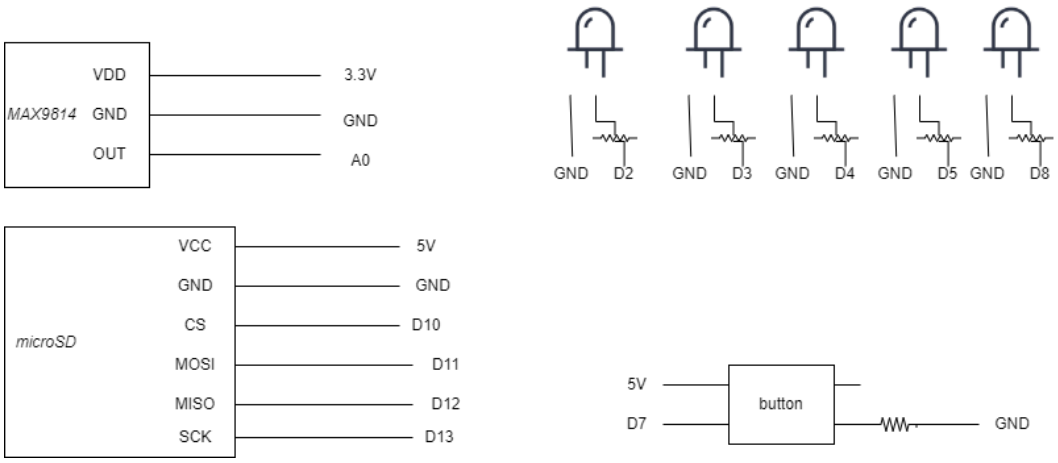
```
    mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate,
    n_mfcc=40).T, axis=0)

    stft = np.abs(librosa.stft(X))

    chroma = np.mean(librosa.feature.chroma_stft(S=stft,
    sr=sample_rate).T, axis=0)

    mel = np.mean(librosa.feature.melspectrogram(X,
    sr=sample_rate).T, axis=0)

    contrast = np.mean(librosa.feature.spectral_contrast(S=stft,
    sr=sample_rate).T, axis=0)

    tonnetz = np.mean(librosa.feature.tonnetz(
    y=librosa.effects.harmonic(X), sr=sample_rate).T, axis=0)

    return mfccs, chroma, mel, contrast, tonnetz


file_path = sys.argv[1]

# Odczytanie modelu
model = load_model('saved_model')

#  Extrakcja cech
with open(file_path, 'r') as f:
    extract = extract_features()

features = [np.concatenate((extract[0], extract[1],
extract[2], extract[3], extract[4]), axis=0)]


#  Predykcja
prediction = model.predict(np.array(features))
print(prediction)
classes = np.argmax(prediction, axis=1)
```

6 Instalacja

6.1 Oprogramowanie

- Python <3.10
- biblioteki Python - requirements.txt
- Arduino IDE
- Projekt był testowany na Windows 10 / 11

6.2 Schemat podłączeniowy



Rysunek 2: Schemat podłączenia

6.3 Rozpakowanie programu

Folder skompresowany należy rozpakować do wybranego folderu.

6.4 Instalacja Arduino

Przed załadowaniem kodu do arduino trzeba otworzyć Arduino IDE, podłączyć Arduino UNO do komputera i sprawdzić, jaki port jest wykorzystywany.

- W razie potrzeby, trzeba zmienić wartość w kodzie python. Domyślnie jest wybrany COM5 (52 linia, script.py).

- Ustawić na komputerze szybkość portu 115200 (Menedżer urządzeń -> Porty COM i LPT -> Ustawienia portu)

Następnie trzeba załadować kod arduino do urządzenia Arduino UNO. Kod do arduino wgrać klikając Plik -> otwórz i wybrać plik Arduino.ino. Należy sprawdzić i doinstalować brakujące biblioteki (rozdział 4.3). Należy pamiętać o ograniczeniach karty pamięci (rozdział 4.3.2).

W Arduino IDE można sprawdzić działanie kodu arduino uruchamiając monitor portu szeregowego (należy ustawić szybkość 115200 w oknie). Podczas odpalania skryptu python monitor portu szeregowego/kreślarka w Arduino IDE musi być wyłączony.

6.5 Instalacja Python

Należy zainstalować interpreter Python, ale w wersji nie wyższej 3.9. Z wersją 3.10 niektóre biblioteki są niekompatybilne.

6.6 Import bibliotek Python

Instalacja wymaganych bibliotek Python odbywa się poprzez wpisanie poniższej komendy w wierszu poleceń. Komendę należy wpisać w głównym katalogu programu.

```
python -m pip install -r requirements.txt
```

6.7 Uczenie modelu

W celu nauczenia modelu nowej osoby należy usunąć pliki z jednego z istniejących folderów i zastąpić je swoimi nagraniami. Należy również pamiętać o tym, że nazwy plików powinny zgadzać się z nazwami usuniętych nagrań. Po wykonaniu tego kroku należy uruchomić plik main.py i odczekać aż model skończy proces uczenia.

Po zamianie plików model wypisze numer klasy, która została uznana za osobę z nagrania. Są to odpowiednio:

- 0 - klasa “Łukasz Łupicki“
- 1 - klasa “Szymon Cegielski“
- 2 - klasa “Krzysztof Niemczyk“
- 3 - klasa “Kristina Alekseyenko“

Po przeprowadzeniu uczenia i testu na nowych nagraniach, należy spojrzeć którą klasę zwrócił program i odpowiednio do zwróconej cyfry porównać z powyższą listą.

6.8 Uruchomienie programu

Program script.py można uruchomić w dowolnym interpreterze lub wpisując w oknie wiersza poleceń:

```
python script.py
```

7 Struktura programu

7.1 main.py

W skrypcie jako pierwszy krok odczytujemy z każdego folderu wszystkie pliki audio, po czym dodajemy do nich dwie dodatkowe kolumny z opisem płci oraz numerem mówcy. Następnie łączymy zebrane pliki w jedną tablice i ją zaszumiamy. Następnie wyodrębniamy z każdego pliku jego cechy i zapisujemy je do osobnej tablicy. W kolejnych krokach dzielimy zapisane cechy wraz z odpowiadającymi im etykietami na trzy zbiory, tj. uczący, walidacyjny oraz testowy. Po ich przeskalowaniu i utworzeniu modelu sieci o czterech warstwach, kompilujemy go i zapisujemy do nowego folderu o nazwie saved_model.

7.2 script.py

W tym pliku otwieramy port połączenia Arduino ze skryptem oraz ładujemy model. W kolejnym kroku zapisujemy plik nagranie.wav do folderu nagrania. Po tym kroku następuje wyodrębnienie cech nagrania oraz predykcja mówcy na ich podstawie. W ostatnim kroku sprawdzamy wartości predykcji, na podstawie których ustalamy czy wartość predykcji danego mówcy była większa od ustalonego przez nas progu poprawności. W przypadku, gdy żaden z wyuczonych mówców nie uzyskał satysfakcjonującej nas wartości, zwracamy informację, że mówca jest nieznany.

7.3 nagrania

W tym folderze znajduje się próbka głosowa odebrana przez port szeregowy z Arduino. Nagranie zapisywane jest w formacie *.wav.

7.4 Arduino.ino

W tym pliku znajduje się kod arduino odpowiedzialny za obsługę mikrofonu, nagrywanie pliku na karcie SD oraz przesyłanie pliku. Naciśnięcie przycisku powoduje rozpoczęcie nagrywania 10 sekundowej próbki dźwięku oraz zapalenie diody. Dźwięk jest zapisywany na włożoną, podpiętą do czytnika kart, kartę SD. Po zapisaniu pliku w formacie *.wav, plik przesyłany jest portem szeregowym do komputera i czeka na odpowiedź z komputera. Po odebraniu sygnału wynikiem predykcji z komputera zapalana jest odpowiednią dioda.

8 Działanie programu

Po wyświetleniu w interpreterze Python poniższych linijek, program jest gotowy do działania.

```
b'loading... SD card\r\n'
b'SD card... OK\r\n'
```

Oznacza to, iż nawiązano połączenie z arduino i odebrano sygnał przez port szeregowy. Naciśnięcie przycisku spowoduje zapalenie się diody led sygnalizującej nagrywanie. Plik dźwiękowy zostaje nagrany na kartę pamięci podpiętą do Arduino. Czas trwania nagrywania wynosi 10 sekund. Taki czas nagrywania spowodowany jest koniecznością przesłania pliku *.wav przez port szeregowy w postaci bajtowej. Podczas nagrywania w otoczeniu powinna panować cisza. Należy mówić w pobliżu mikrofonu. Rozpoczęcie przesyłania poprzedzone jest wyświetleniem:

```
b'START_FILE\r\n'
```

Może to portrwać kilka sekund. Plik zapisywany jest na komputerze w folderze "nagrania" pod nazwą: "nagranie.wav". Po odebraniu pliku portem szeregowym i wyświetleniu:

```
file created
file saved
```

Po odebraniu pliku następuje proces predykcji. W ciągu kilku sekund zapali się dioda oznaczająca mówiącą osobę oraz wyświetli się imię osoby wskazanej w predykcji. Jeśli mówca nie został rozpoznany, zapalą się wszystkie diody. Plik z nagraniem jest usuwany.

9 Napotkane problemy

9.1 Wywoływanie skryptów Arduino UNO w Python

Początkowo założyliśmy, że skrypt napisany w python i klasyfikujący nagrany głos będzie wywoływany z programu na Arduino. Próbowaliśmy to zrealizować używając biblioteki Process, pozwalający wywoływać skrypty zewnętrzne, nie znajdujące się w pamięci Arduino. W trakcie wykonywania projektu okazało się jednak, że wybrany przez nas model urządzenia, Arduino UNO, nie potrafi wywoływać takich skryptów. Ten problem postanowiliśmy rozwiązać przez przeniesienie części kodu z arduino do python, dzięki czemu możemy wykonywać wywoływania w drugą stronę, ze skryptu python do arduino. Do tego została użyta biblioteka serial, która łączy się z portem arduino i wysyła znaki bajtowe do arduino. W kodzie arduino te znaki są odczytywane i wykonywane są odpowiednie akcje.

9.2 Ograniczenia dotyczące karty microSD

W trakcie wykonywania projektu napotkaliśmy też problemy związane z brakiem możliwości odczytu karty microSD w pewnych przypadkach. Po głębszej analizie okazało się, że karta pamięci dla pracy z arduino powinna spełniać konkretne warunki: rozmiar pamięci większy niż 2GB, ale nie większy niż 16Gb oraz formatowanie exFAT.

9.3 Bezpośredni odczyt z komputera plików zapisanych w arduino. Szybkość przesyłania przez serial

Kolejną trudnością był brak możliwości przesłania gotowego pliku dźwiękowego zapisanego na karcie pamięci ardunio do algorytmu na komputerze. Po przeprowadzeniu analizy i konsultacji z prowadzącym podjęliśmy dezycję o przesłanie pliku bajtowo przez port serial. Pierwsze próby zakończyły się niepowodzeniem. Czas przesyłania pliku był nieakceptowalnie długi. Zwiększenie szybkości przesyłania na obydwu urządzeniach rozwiązało problem.

9.4 Ograniczenia bufora

Podczas przesłania pliku, po zwiększeniu szybkości (baud 9600 -> 115200) następowało przepełnienie bufora. Odczytywanie pliku następowało szybciej niż był on odbierany przez komputer. Zastąpienie domyślnej metody pyserial.readline poniższym kodem pozwoliło na ominięcie ograniczeń bufora.

```
class ReadLine:
    def __init__(self, s):
        self.buf = bytearray()
        self.s = s
    def readline(self):
        i = self.buf.find(b"\n")
        if i >= 0:
            r = self.buf[:i+1]
            self.buf = self.buf[i+1:]
            return r
        while True:
            i = max(1, min(2048, self.s.in_waiting))
            data = self.s.read(i)
            i = data.find(b"\n")
            if i >= 0:
                r = self.buf + data[:i+1]
                self.buf[0:] = data[i+1:]
                return r
            else:
                self.buf.extend(data)
```

9.5 Kodowanie pliku

Po rozwiązaniu problemu przepełnionego bufora napotkano na problem związany z kodowaniem pliku. Plik *.wav odczytany i przesłany od Arduino do komputera przez port szeregowy był nie do odtworzenia na komputerze. Problem udało się rozwiązać przesyłając plik bajtowo.

9.6 Problem z rozmiarem karty pamięci

Przy karcie SD o pojemności 2GB występował problem. Karta była widoczna lecz wyświetlany był komunikat o nieprawidłowym sformatowaniu karty. Dla karty o rozmiarze 8GB problem nie występował

9.7 Utrata napięcia

Z powodu dużej ilości podpiętych urządzeń, na płytce prototypowej, przy podpięciu zbyt daleko od wpiętego zasilania diody nie zapalały się. Konieczna była reorganizacja kabli na płytce - trzeba było umieścić je bliżej kabla zasilającego. Spowodowało to spore zagęszczenie okablowania.

9.8 Jakość próbek

Ze względu na panującą sytuację epidemiologiczną i sposób funkcjonowania uczelni - nauczanie zdalne - nie byliśmy w stanie nagrać wszystkich próbek na tym samym mikrofonie. Z tego powodu, nagrania zostały zarejestrowanie na 4 różniących się od siebie mikrofonach, co z kolei przekłada się na ostateczną predykcję mówcy. Fakt zastosowania różnych mikrofonów ukazuje, że ekstrakcja cech i trening na ich podstawie stara się wychwycić cechy szczególne dla konkretnego mówcy. W tym przypadku, oprócz naszych głosów, cechą szczególną są szumy, które są różne dla wszystkich 4 mikrofonów. Model oprócz nauki cech charakterystycznych z naszych głosów nauczył się również cech charakterystycznych z konkretnych mikrofonów, przez co stosując mikrofon kogoś innego niż ten zastosowany do swoich próbek możemy otrzymać nieprawidłową predykcję.

9.9 Ilość próbek

Podczas projektu zdecydowaliśmy się na ilość próbek wynoszącą 100 - po 25 nagrań na osobę. Każdy z nas nagrał 25 plików .wav o długości od 10 do 30 sekund. Po implementacji modelu i jego wytrenowaniu przypuszczaliśmy, że taka ilość próbek jest niewystarczająca i model prawdopodobnie uczy się na pamięć. To tłumaczyłoby zbyt dobre wyniki walidacji i treningu, oscylujące w okolicach stu procent poprawnych predykcji, które nie pokrywały się z obserwacjami doświadczalnymi.

10 Wnioski

W trakcie prac nad realizacją projektu udało się nam zbudować system do rozpoznawania głosu, przesyłanego do komputera z Arduino. Projekt ten okazał się być wymagający, ponieważ napotkało nas sporo problemów kompatybilności i konfiguracji. Strona techniczna działa zgodnie z założeniami, podczas gdy nauczony model rozpoznawania nie zawsze poprawnie identyfikuje osobę. Głównym temu powodem była różnica sprzętu i otoczenia, użytych podczas nagrywania próbek do zbiorów uczącego i testowego. Panująca sytuacja oraz studia zdalne spowodowały, że każdy nagrywał się swoim mikrofonem u siebie w domach, więc oprócz różnicy głosów było też dużo innych dźwięków, które prawdopodobnie trafiły do cech, wyciągniętych z próbek podczas uczenia modelu.

Kolejnym powodem była też zbyt mała ilość próbek. Żeby dobrze nauczyć model, musielibyśmy udzielić dużo więcej czasu i miejsca na dyskach na nagrywanie, zapisywanie oraz uczenie modelu.

Biorąc pod uwagę zasoby czasowe, sprzętowe i ludzkie uważamy, że projekt skończył się sukcesem, a działanie systemu jest zgodne z założeniami.