Analiza i przetwarzanie dźwięku

Sprawozdanie z projektu 3 - rozpoznawanie mowy

Damian Skowroński

16 czerwca 2023

1 Wprowadzenie

W tej części rozwijam projekt o funkcjonalność rozpoznawania osoby mówiącej oraz co zostało powiedziane. W tym sprawozdaniu skupię się początkowo na dokumentacji kodu, a następnie przedstawię wyniki modeli rozpoznających mowę.

2 Dokumentacja

Ten krok projektu można podzielić na trzy części:

1. uzyskiwanie z nagrań współczynników mel-cepstralnych (MFCC)
2. stworzenie zbioru danych z informacjami ze wszystkich dostępnych nagrań
3. implementacja modeli rozpoznających mowę

W następnych sekcjach opisuję kod dotyczący poszczególnych części.

2.1 Współczynniki mel-cepstralne

Funkcje z tej części znajdują się w pliku *functions/mfcc.py*. Generalnie te funkcje to kolejne kroki, przez które przechodzi nagranie w celu uzyskania MFCC dla każdej ramki. MFCC jest potrzebne ponieważ jest potem głównym (oprócz energii jedynym) atrybutem w modelach.

2.1.1 Preemfaza

Funkcja `preemphasis` implementuje operację preemfazy na sygnale dźwiękowym. Ma ona podkreślać wysokie częstotliwości, zmniejszając amplitudę niskich częstotliwości. Argumenty:

- `signal` - wejściowy sygnał audio w postaci jednowymiarowej tablicy numpy.
- `coeff` - współczynnik preemfazy, który kontroluje stopień wzmocnienia wysokich częstotliwości. Domyślnie ustawiony na 0.97.
- `display` - flaga określające opcję wyświetlenia widgetu pozwalajcego na odsłuchanie nagrania po transformacji w jupyter notebook
- `frame_rate` - częstotliwości próbkowania sygnału audio, wykorzystywana tylko w widget'cie

Funkcja zwraca tablicę numpy po przeprowadzonej operacji preemfazy.

2.1.2 Bank filtrów trójkątnych

Funkcja `get_filter_banks` służy do obliczania banku filtrów trójkątnych wykorzystywanego w ekstrakcji MFCC. Argumenty;

- `n_filters` - liczba filtrów w banku
- `NFFT` - liczba punktów wykorzystanych w FFT (szybkiej transformacji Fouriera)
- `frame_rate` - częstotliwość próbkowania sygnału audio
- `low_freq_mel` - dolna granica zakresu częstotliwości w skali mel
- `high_freq_mel` - górna granica zakresu częstotliwości w skali mel, jeśli nie jest podana, to zostanie obliczona na podstawie `frame_rate`.

Funkcja oblicza indeksy (biny) odpowiadające poszczególnym filtrom w banku filtrów. Następnie dla każdego filtru oblicza wagi dla poszczególnych częstotliwości zgodnie z trójkątnym kształtem. Zwraca bank filtrów jako macierz numpy.

2.1.3 Dyskretna transformacja kosinusowa

Funkcja `get_dct_coefficients` służy do obliczania współczynników dyskretnej transformaty kosinusowej (DCT) dla sygnałów. Argumenty:

- `input_signals` - sygnały (ramki) dla których mają zostać obliczone współczynniki DCT
- `M` - liczba współczynników DCT do obliczenia

Funkcja zwraca obliczone współczynniki.

2.1.4 Energia

Funkcja `get_energy` służy do obliczania energii ramki. Przyjmuje `frame` - ramkę sygnału. Zwraca energię jako sumę kwadratów wartości w ramce.

2.1.5 MFCC pipeline

Funkcja `mfcc_pipeline` jest pipeline'em, który wykonuje sekwencję operacji przetwarzania sygnału audio w celu uzyskania współczynników MFCC i energii. Funkcja przyjmuje ścieżkę do pliku z nagraniem o rozszerzeniu *.wav*. Dużo z parametrów tej funkcji jest ustawione "na sztywno", ponieważ jest przystosowana do konkretnego zastosowania.

2.2 Zbiór cech nagrań

W celu wytrenowania modelu rozpoznawania mowy potrzebny jest zbiór, w którym przechowywane są cechy nagrań. Aby to osiągnąć pobrałem wszystkie dostępne nagrania (tak mi się wydaje, chyba były 24 unikalne osoby). Następnie skryptem *recordings/create_audio_csv.ipynb* wczytuję kolejne nagrania zapisaując w ramce danych informacje dotyczące:

- **person** - osoby, która jest autorem nagrania
- **word_label** - słowa jakie zostało powiedziane
- **mfcc** - macierz współczynników mfcc (rozmiar macierzy: <ilość ramek> × <liczba współczynników mfcc>)
- **energy** - energia każdej z ramki - wektor o długości liczby ramek

Następnie taką ramkę zapisuję w pliku o rozszerzeniu *.pkl*. Wynikowy plik *.pkl* oraz pliki z nagraniami nie znajdują się na repozytorium ponieważ zajmują dużo pamięci.

2.3 Modele rozpoznawania mowy

Dla tej części projektu poświęcony jest folder *models*. Stworzyłem dwa modele do rozpoznawania mowy:

- model rozpoznający która osoba jest autorem nagrania
- model rozpoznający która liczba (od 0 do 10) została powiedziana

Dla obu modeli wykorzystałem prawie identyczną konwolucyjną sieci neuronową z pakietu Tensorflow Keras:

```
model = tf.keras.Sequential([
  layers.Conv1D(32, kernel_size=3, activation='relu', input_shape=input_shape),
  layers.MaxPooling1D(pool_size=2),
  layers.Conv1D(64, kernel_size=3, activation='relu'),
  layers.MaxPooling1D(pool_size=2),
  layers.Conv1D(128, kernel_size=3, activation='relu'),
  layers.GlobalAveragePooling1D(),
  layers.Dense(64, activation='relu'),
  layers.Dense(class_number, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Przy czym w pierwszym modelu `class_number = 24`, bo w moim zbiorze ostatecznie tyle pojawiło się unikalnych autorów, a w drugim modelu `class_number = 11`, bo jest 11 liczb od 0 do 10.

Całe kod związany z modelowaniem zawiera się w pliku *models/models.ipynb*. Korzystałem z jednostki TPU na platformie Google Colab (darmowa wersja), aby sieci szybciej się wytrenowały. Wszystkie użyte pakiety w tym pliku są zainstalowane domyślnie na Google Colab. W pliku znajduje się również kilka linijek kodu, których jedyna funkcjonalność to umożliwienie korzystania z TPU. Kolejne kroki wykonane w celu stworzenia modeli do rozpoznawania mowy to:

1. Wczytuję gotowy plik *.pkl* opisany w poprzedniej sekcji i pozbywam się wierszy z nagraniami zdań - mają one dużo więcej ramek, pojawia się problem z pamięcią RAM w następnym kroku.

2. Do tej pory nagrania mają różne liczby ramek, co sprawia, że kolumny 'mfcc' i 'energy' mają wartości o różnych kształtach. Do sieci potrzebuje inputu o takich samych kształtach. W tym celu znajduję obserwację o największej długości i wykonuję padding (dopełnienie 0 do tej długości) dla pozostałych obserwacji. Po wykonaniu paddingu na kolumnach 'mfcc' i 'energy' łączę je - dołączam wektor 'energy' jako kolumnę 'mfcc' i zapisuję to jako atrybut 'joined'. Teraz każda wartość w kolumnie 'joined' ma taki sam kształt (macierz rozmiaru 1037x17, przy czym 1037 oznacza maksymalną liczbę ramek, a 17 odpowiada 16 współczynnikom MFCC i kolumnie 'energy' dla każdej ramki).

3. Atrybut 'joined' jest trójwymiarową macierzą eksperymentu `X`. Wektor wartości przewidywanych `y` to w zależności od modelu, albo atrybut 'person', albo 'word_label' (przy czym w drugim przypadku zostawiam tylko nagrania odpowiadające liczbom.

4. Wykonuję encoding na wartościahc `y`.

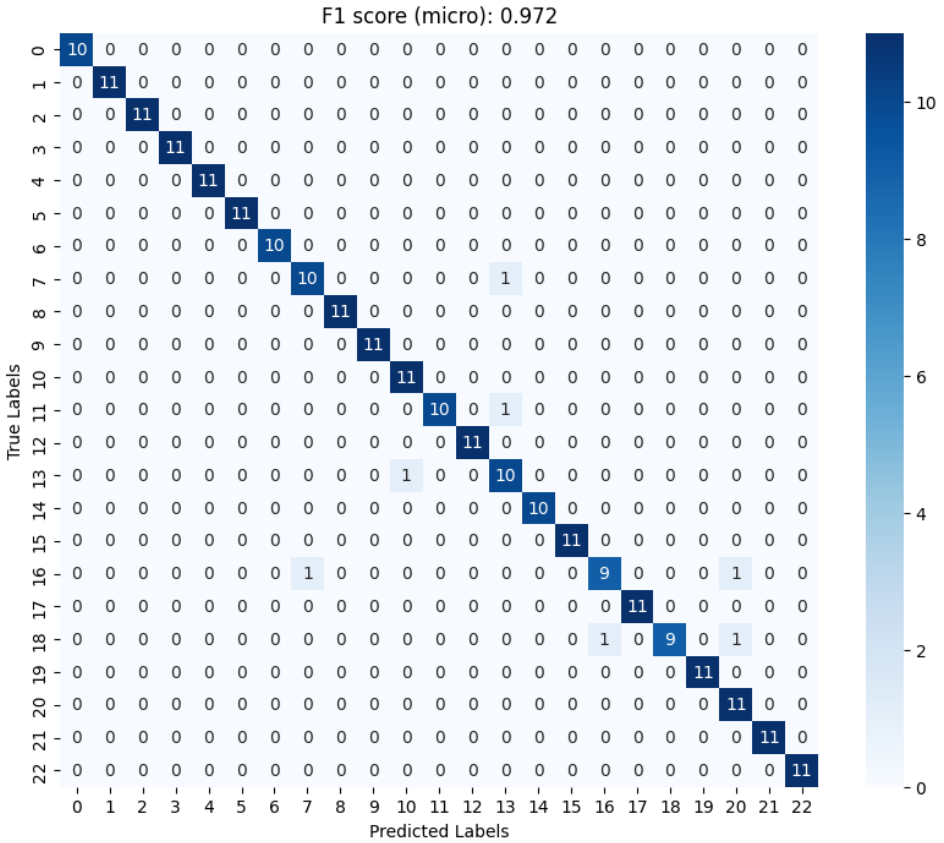
5. Dzielę zbiór na zbiory: treningowy, walidacyjny i testowy używając stratyfikacji ze względu na zmienną przewidywaną.

6. Inicjuję model i używam metody `fit` na 100 epochach. Przy trenowniu modelu wykorzystuję zbiór treningowy i walidacyjny, przy czym zapisuję model o najlepszej wartości `accuracy` dla zbioru walidacyjnego.

7. Zbiór testowy wykorzystuję do oceny modelu.

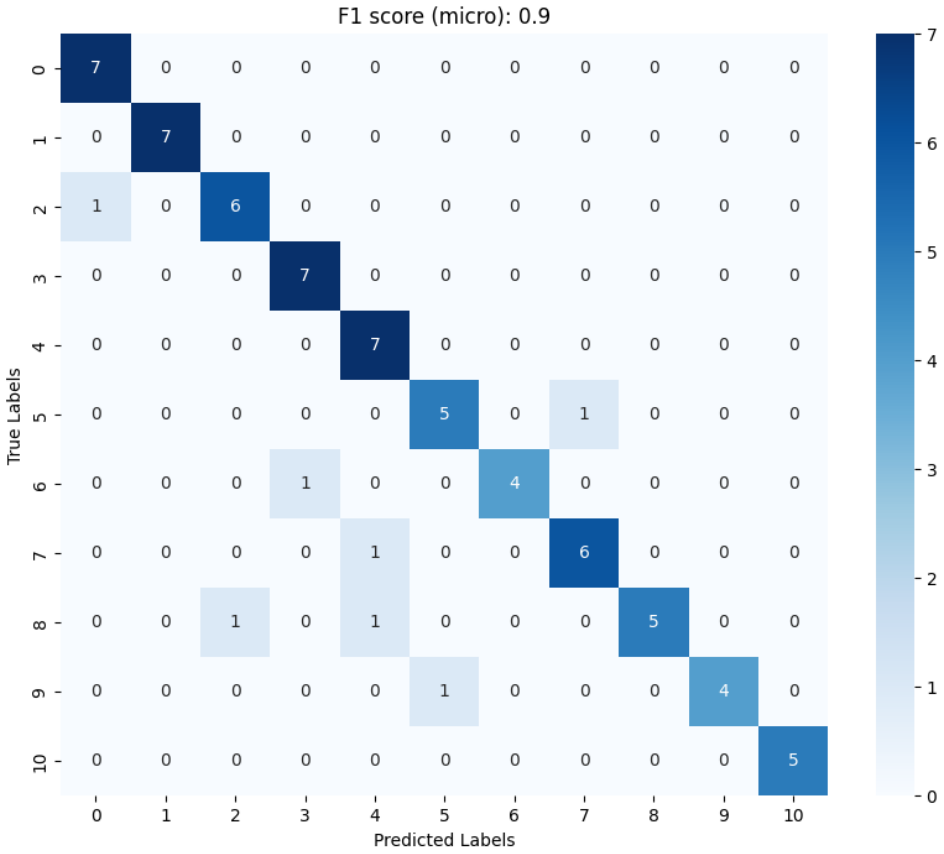
3 Wyniki

W rezultacie otrzymałem dwa modele. Są one zapisane w folderze *models* w plikach *best_model.h5* dla modelu rozpoznającego autora nagrania i *best_model_numbers.h5* dla modelu rozpoznającego liczby. Na rysunku 1 i 2 zostały pokazane wyniki dla obu modeli w postaci macierzy błędu oraz wyniki statystyki mikro F1 dla zbioru testowego.



Rysunek 1: Wyniki klasyfikacji dla modelu rozpoznającego autora nagrania.

Widać, że model rozpoznający autora nagrania radzi sobie świetnie z wynikiem statystyki F1 równym 0.972. Tak naprawdę można zauważyć, że model myli się tylko kilka razy.



Rysunek 2: Wyniki klasyfikacji dla modelu rozpoznającego liczby.

Dla modelu rozpoznającego liczby wynik jest nieco gorszy, ale nadal satysfakcjonujący. Tutaj również widać, że model myli się tylko kilka razy.

4 Wnioski

Ogólnie wyniki są dużo lepsze niż się spodziewałem. Na pewno dla większego zbioru nagrań, sieci mogłyby się dużo lepiej nauczyć. Szczególnie w drugim modelu sieć mogła mieć za mało obserwacji, ponieważ łączenie było ich około 600, co daje tylko po 60 nagrań dla danej liczby podzielone na 3 zbiory.