
Klasyfikacja dźwięków na podstawie zebranych danych

Jakub Cichy, Szymon Rychel, Kamil Szprych

18 czerwca 2019

Spis treści

1	Wstęp	2
1.1	Cel projektu	2
2	Realizacja	2
2.1	Tworzenie zbioru danych	2
2.2	Głębokie sieci konwolucyjne	2
2.3	Nasz model sieci	3
2.4	Ekstrakcja cech z dźwięków	3
2.5	Działanie programu	4
2.5.1	Pliki csv	4
2.5.2	Pliki wav	4
2.5.3	Wykorzystane biblioteki	4
2.5.4	Interfejs konsolowy	5
2.6	Aplikacja mobilna	6
3	Wyniki	9

1 Wstęp

1.1 Cel projektu

Celem projektu było oprogramowanie sieci neuronowej do klasyfikacji dźwięku, oraz samodzielne stworzenie zbiorów uczących i testujących. Następnie wyeksportowanie modelu, celem użycia go w aplikacji mobilnej.

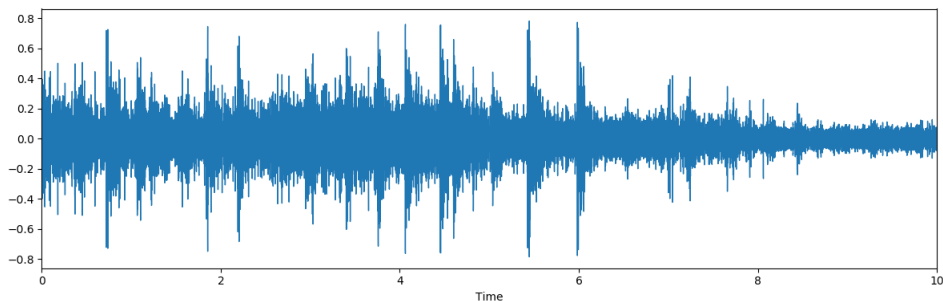
2 Realizacja

2.1 Tworzenie zbioru danych

Do realizacji zadania konieczne było zgromadzenie odpowiedniej ilości próbek dźwiękowych. Postanowiliśmy zebrać dźwięki z obszarów miejskich - wybraliśmy trzy kategorie:

- Rynek wrocławski
- Pasaż Grunwaldzki/Rondo Regana
- wnętrze tramwaju

Na przełomie kilku tygodni sukcesywnie nagrywaliśmy dźwięki z tych miejsc używając mikrofonów w naszych telefonach. Następnie odfiltrowaliśmy te nagrania które zawierały dużo zbędnych informacji, jak np. głośne rozmowy ludzi dookoła, dźwięki syreny, czy zbyt duży szum wiatru. Następnie podzieliśmy zgromadzone nagrania na 500 próbek po 10 sekund każda co daje w sumie ponad 83 minuty nagrań. Większość próbek trafiła do zbioru uczącego, kilka zostawiliśmy w celach testowania. Poniżej wykres amplitudowy dla jednej z próbek:



Rysunek 1: Wykres amplitudy sygnału dźwiękowego dla próbki z Pasażu Grunwaldzkiego

2.2 Głębokie sieci konwolucyjne

Głębokie sieci konwolucyjne to jedna z klas głębokich sieci neuronowych stosowane najczęściej do pracy z obrazami i dźwiękami. Potrafią one filtrować różne części danych i wyodrębiać ważne cechy w procesie klasyfikacji. Warstwa konwolucyjna składa się z filtrów wykrywających pewne niskopoziomowe cechy obrazu. Neurony w warstwie wejściowej grupowane są w obszary o stałym rozmiarze, współdzielą ze sobą wagi, oraz są one połączone z neuronem z warstwy wyżej tworząc filtr, który w procesie uczenia odpowiada za wykrywanie pewnej cechy.

Warstwy typu *pooling* odpowiadają za zmniejszanie wymiaru danych, poprzez łączenie wyjścia jednej warstwy z neuronem kolejnej warstwy. U nas zastosowany został *Max Pooling* czyli wyliczanie maksymalnej wartości z grupy sąsiadujących neuronów z poprzedniej warstwy. Chronią również sieć przed przeuczeniem.

Do wsparcia ostatecznej klasyfikacji stosuje się tzw. *Dense layers* w których wszystkie neurony z warstwy poprzedniej połączone są ze wszystkimi neuronami z warstwy kolejnej.

2.3 Nasz model sieci

Zastosowaliśmy model z czterema warstwami konwolucyjnymi. Każda warstwa zawiera dwa razy większą liczbę filtrów niż poprzednia, w naszym przypadku jest to odpowiednio 16, 32, 64 oraz 128. Każda warstwa zawiera wyżej wspomniany *Max Pooling* oraz tak zwany *Dropout*.

Dropout jest techniką opatentowaną przez Google, której celem jest redukcja zjawiska przeuczenia sieci. Polega na usuwaniu pewnych połączeń między neuronami, z pewnym prawdopodobieństwem. Dzięki temu sieć będzie starała się dostrzegać niezależne cechy i brak jednej z nich nie będzie stanowił problemu. *Dropout rate* został ustawiony na 0.2, czyli usuwanie 20% połączeń. Każda warstwa ma funkcję aktywacji 'relu' od *Rectified Linear Unit*, postaci:

$$f(x) = \max(0, x)$$

Jest ona najczęściej stosowana w sieciach konwolucyjnych - w porównaniu z innymi, przyspiesza proces uczenia oraz obliczenia.

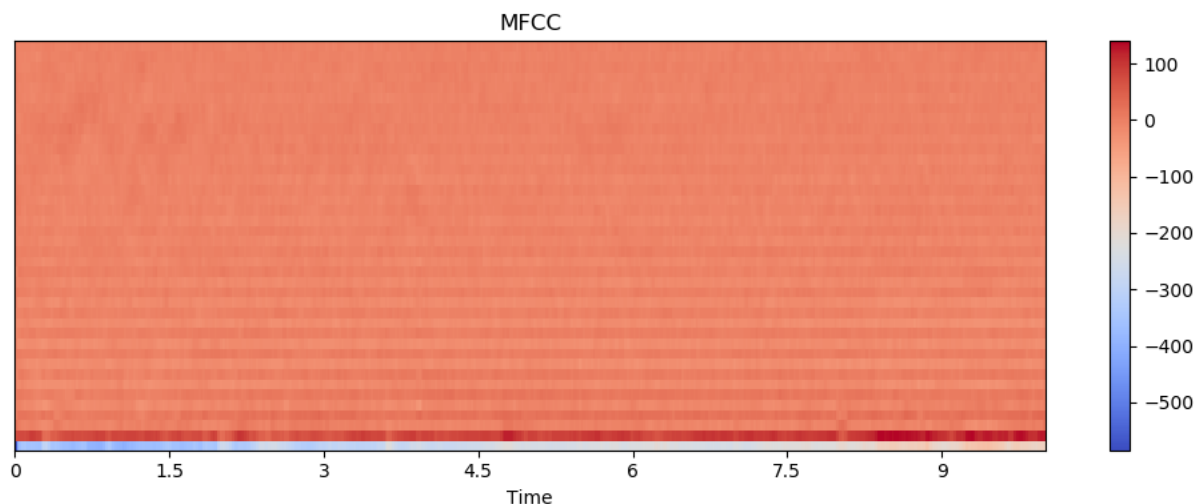
2.4 Ekstrakcja cech z dźwięków

Istotnym momentem w procesie tworzenia sieci neuronowych jest odpowiedni dobór cech danych wejściowych, tak aby sieć była w stanie odnaleźć w nich pewne wzorce i podobieństwa, jednocześnie będąc w stanie odróżnić od siebie poszczególne grupy obiektów. W tym celu, często stosowane są m.in. Transformata Fouriera, dyskretna transformata cosinusowa czy różnego typu filtry częstotliwościowe.

W przypadku problemów dotyczących przetwarzania dźwięków, rozpoznawania odgłosów, mowy i tym podobnych, bardzo często wykorzystywany jest zestaw cech o akronimie *MFCC* (ang. *Mel Frequency Cepstral Coefficients*) Proces ekstrakcji cech, wygląda w tym przypadku następująco:

- Sygnał przepuszczany jest przez transformatę fouriera
- Następnie, widmo częstotliwościowe przepuszczane jest przez filtry Melowe, w celu uzyskania prążków widma (liczba pasm filtru odpowiada liczbie prążków)
- Obliczana jest funkcja logarytmiczna dla poszczególnych prążków
- Poprzednie wartości przepuszczane są przez dyskretną transformatę cosinusową
- Amplitudy otrzymanego „widma” (dokładniej cepstrum) to MFCC

Taki wektor, w naszym przypadku, posiada 40 wartości, wg. których klasyfikowane są podane do sieci dźwięki. W naszej aplikacji, w celu ekstrakcji wspomnianych cech modelu MFCC, wykorzystana została biblioteka języka Python o nazwie Librosa. Odbywa się to zatem za pomocą następującej funkcji `librosa.feature.mfcc(y=audio, sr=samplerate, nmfcc=40)`. Biblioteka stanowi zestaw funkcji przydatnych przy przetwarzaniu dźwięków i pozyskiwaniu z nich przeróżnych wartości liczbowych, przydatna m.in. przy konstruowaniu sieci neuronowych rozpoznających mowę.



Rysunek 2: MFCC dla próbki z Pasażu Grunwaldzkiego

2.5 Działanie programu

2.5.1 Pliki csv

CSV (ang. Comma separated values) to format pliku, powszechnie stosowany w uczeniu maszynowym, do katalogowania danych. W pierwszym wierszu pliku, umieszczane są etykiety (ang. Labels) kolejnych w sekwencji danych. W przypadku opisywanej aplikacji, w pliku .csv znajdują się dwie etykiety: `fileName`, określająca nazwę pliku dźwiękowego oraz `fileLabel`, określającą, jakiemu miejscu odpowiada dane nagranie.

2.5.2 Pliki wav

Pliki audio, najczęściej zapisywane są w formacie .mp3 ze względu na mniejszy rozmiar plików, wynikający z zastosowanej kompresji danych. DO projektu, wybrany został format plików .WAV. Głównym czynnikiem decyzyjnym, był brak kompresji) Fakt ten może być istotny przy wydzielaniu cech z nagrania. Pliki, przed wykorzystaniem w programie, zostały odpowiednio posortowane w celu wykluczenia próbek silnie zakłóconych mową czy innymi, niepowołanymi odgłosami. Po tym, próbki podzielone zostały na dziesięciosekundowe fragmenty, każdy o jednakowej długości.

2.5.3 Wykorzystane biblioteki

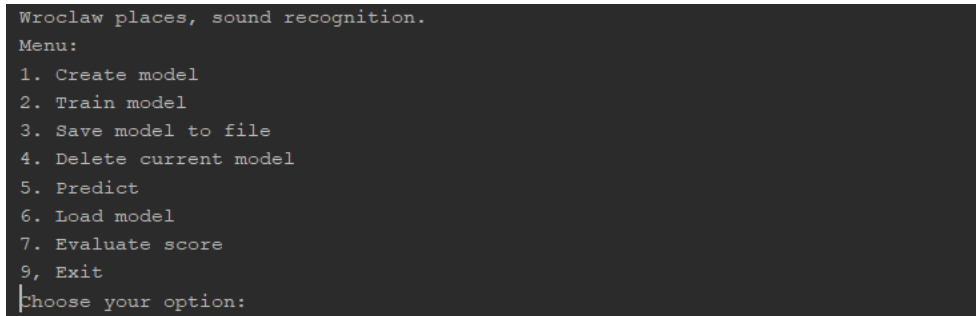
- *Pandas* zestaw użytecznych struktur i narzędzi, przydatnych do przeprowadzania bardziej zaawansowanych obliczeń i modyfikacji danych. Bardzo przydatne do wykonywania operacji macierzowych, wektorowych i przechowywania wartości przy pracy z sieciami neuronowymi.
- *Keras* biblioteka zawierająca liczne modele, metody, funkcje, struktury pozwalające tworzyć, analizować, modyfikować i usprawniać sieci neuronowe.
- *Matplotlib* – biblioteka pozwalająca na tworzenie i wyświetlanie wykresów, grafów.
- *Librosa* biblioteka dostarczająca dużą ilość funkcji, narzędzi, struktur służących do obróbki audio. Narzędzie niemalże niezbędne przy tworzeniu sieci neuronowych, operujących na plikach dźwiękowych. Za pomocą tej biblioteki odbywa się ekstrakcja cech w modelu MFCC.

- *Scikit/Sklearn* zestaw bibliotek zawierające narzędzia przydatne do przeprowadzania w programie bardziej zaawansowanych i naukowych badań i kalkulacji

Scikit/Sklearn – zestaw bibliotek zawierające narzędzia przydatne do przeprowadzania w programie bardziej zaawansowanych i naukowych badań i kalkulacji

2.5.4 Interfejs konsolowy

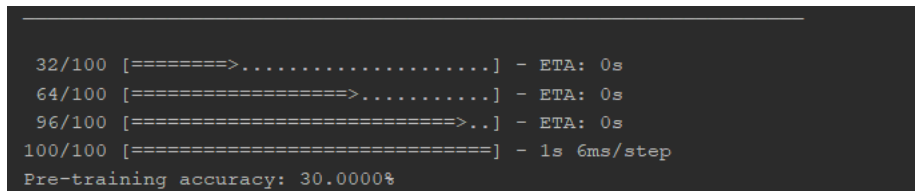
Po uruchomieniu programu, następuje automatyczna ekstrakcja cech z próbek opisanych w pliku .csv, do którego odnośnik znajduje się w kodzie programu. Program dzieli też próbki na ciąg uczący i testowy, w proporcji 80:20 Po wykonaniu tej czynności, wyświetlane jest konsolowe menu, pozwalające na wybór opcji przedstawionych na zdjęciu.



```
Wroclaw places, sound recognition.  
Menu:  
1. Create model  
2. Train model  
3. Save model to file  
4. Delete current model  
5. Predict  
6. Load model  
7. Evaluate score  
9. Exit  
|choose your option:
```

Rysunek 3: Menu programu

Funkcja *Create model* – tworzy nowy model sieci, zgodny z opisem w punkcie o sieci wykorzystanej w aplikacji. Stworzony model posiada swoją skuteczność, jednak jest ona dużo niższa, niż po wyuczeniu sieci. W przypadku opisywanego programu, wartość ta wynosi zazwyczaj między 20% a 40%.



```
32/100 [=====>.....] - ETA: 0s  
64/100 [=====>.....] - ETA: 0s  
96/100 [=====>..] - ETA: 0s  
100/100 [=====] - 1s 6ms/step  
Pre-training accuracy: 30.0000%
```

Rysunek 4: Opcja Create model

Funkcja *Train model* – przeprowadza proces uczenia sieci. Użytkownik proszony jest o podanie ilości epok i rozmiaru pojedynczej paczki danych. Na tej podstawie przeprowadzane jest uczenie sieci. W jego trakcie, możliwe jest obserwowanie przebiegu uczenia dla każdej epoki, dla każdego zestawu z osobna i w postaci wyniku ostatecznego dla danej epoki. Sprawdzoną doświadczalnie wartością parametrów jest: numepochs = 40 oraz numbatchsize = 50. Dla takich wartości, sieć osiąga w procesie uczenia skuteczność bliską 90%, zachowując stosunkowo krótki czas uczenia (ok. 5 minut).

```

400/400 [=====] - 7s 18ms/step - loss: 0.3437 - acc: 0.8750 - val_loss: 0.3690 - val_acc: 0.9000
Epoch 12/60

 50/400 [==>.....] - ETA: 5s - loss: 0.4409 - acc: 0.8800
100/400 [=====>.....] - ETA: 4s - loss: 0.3843 - acc: 0.8800
150/400 [=====>.....] - ETA: 4s - loss: 0.4118 - acc: 0.8533
200/400 [=====>.....] - ETA: 3s - loss: 0.4072 - acc: 0.8400
250/400 [=====>.....] - ETA: 2s - loss: 0.3775 - acc: 0.8520
300/400 [=====>.....] - ETA: 1s - loss: 0.3922 - acc: 0.8467
350/400 [=====>.....] - ETA: 0s - loss: 0.3557 - acc: 0.8629
400/400 [=====] - 7s 17ms/step - loss: 0.3435 - acc: 0.8725 - val_loss: 0.3441 - val_acc: 0.9200
Epoch 13/60

```

Rysunek 5: Opcja Train model

Funkcja *Save model to file* – pozwala wyeksportować aktualny model sieci do pliku. Ścieżka i nazwa pliku podawana jest przez użytkownika, po wcześniejszym zapytaniu.

Delete current model – powoduje usunięcie aktualnego modelu sieci.

Predict – funkcja prosi o ścieżkę do pliku dźwiękowego, a następnie przeprowadza predykcję. Predykcja wyświetlana jest jako procentowa wartość podobieństwa do każdej z klas dostępnych w sieci (tutaj 3 klasy). Wartości te sumują się do 1, dla każdej próbki. Klasa z największym procentowym podobieństwem staje się predykcją programu.

```

Choose your option: 3
Insert file path: test/test_tramwaj3.wav
The predicted class is: tramwaj

pasaz      : 0.31244271993637084960937500000000
rynek      : 0.30556949973106384277343750000000
tramwaj     : 0.38198781013488769531250000000000

```

Rysunek 6: Opcja Predict

Load model – pozwala załadować do programu wcześniej utworzony model sieci i kontynuować dalszą pracę aplikacji na tym modelu

Evaluate model – pozwala wyświetlić informacje o skuteczności sieci dla ciągu uczącego i testującego

```

Choose your option: 7
Training Accuracy: 0.965
Testing Accuracy: 0.97

```

Rysunek 7: Opcja Evaluate model

2.6 Aplikacja mobilna

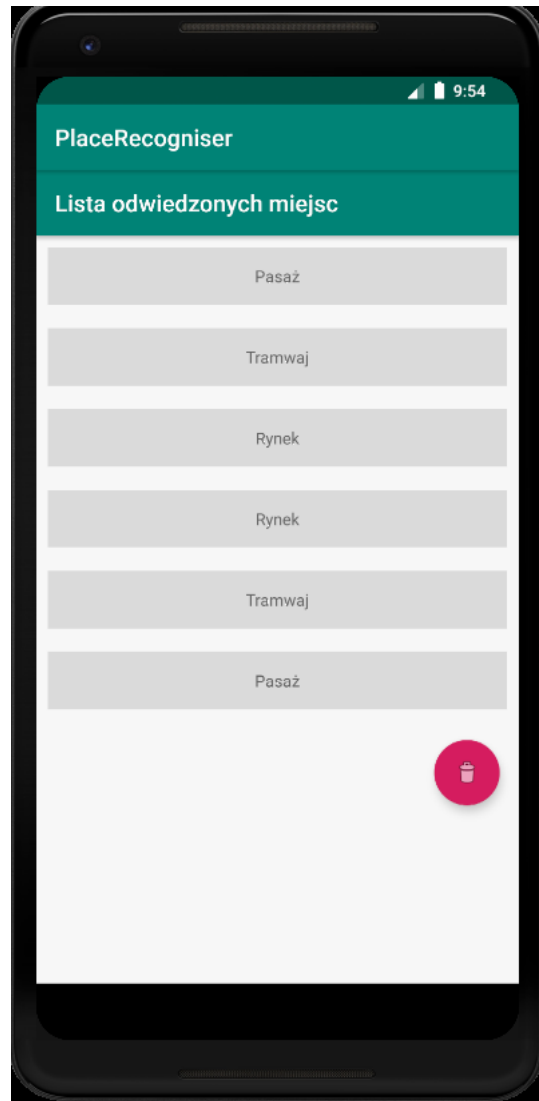
Podjeliśmy również próbę napisania aplikacji mobilnej do naszej sieci, która pozwoliłaby na nagranie dźwięku i otrzymanie na tej podstawie predykcji. Udało nam się zaimplementować prosty interfejs takiej aplikacji, oraz funkcję pobierania i zapisywania 10 sekundowego dźwięku. Jednak nie udało nam się wyeksportować modelu z Pythona do Javy. Poniżej kilka zdjęć interfejsu.



Rysunek 8: Interfejs aplikacji



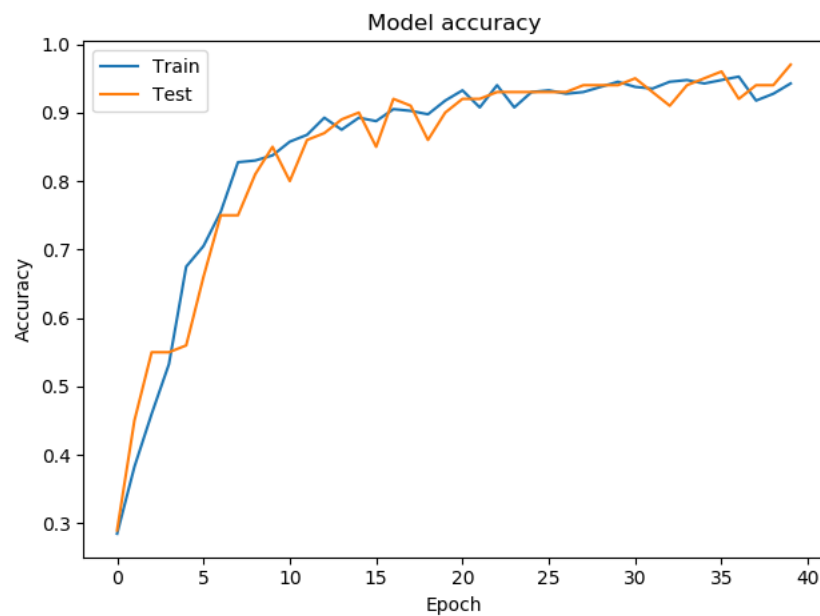
Rysunek 9: Interfejs aplikacji



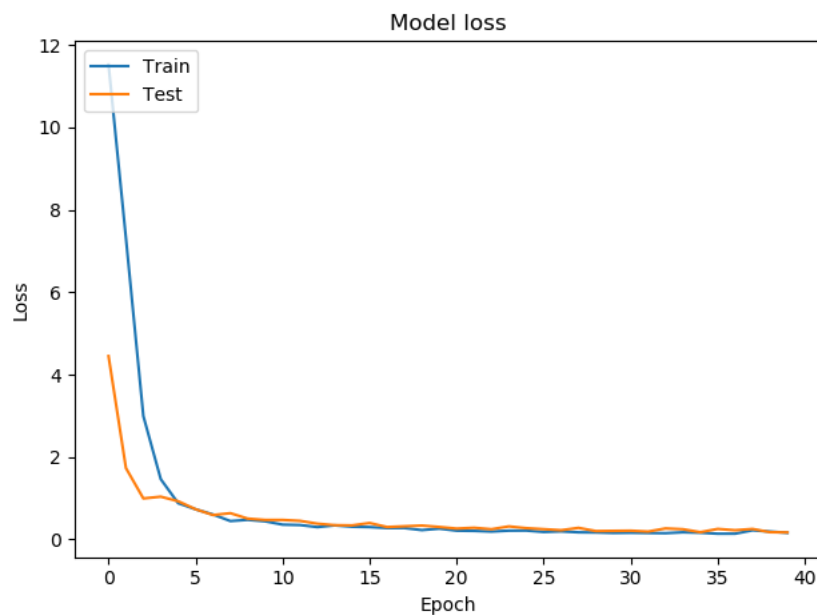
Rysunek 10: Interfejs aplikacji

3 Wyniki

Sieć osiąga bardzo zadowalającą skuteczność na poziomie 90% na etapie uczenia. Poniżej wykresy na których widzimy poziom skuteczności oraz wartość błędu w kolejnych iteracjach dla zbiorów uczących i testowych.



Rysunek 11: Wykres skuteczności w kolejnych iteracjach



Rysunek 12: Wykres błędu w kolejnych iteracjach