Etykietowanie muzyki (music tagging) za pomocą metod sztucznej inteligencji

PDI1 - Pracownia dyplomowa inżynierska

Olga Krupa

Czerwiec 2022

1 Analiza literatury

1.1 Artist Similarity with Graph Neural Networks

Jest to praca naukowa opisująca sieć GNN służąca do wyszukiwania podobieństw między artystami. Wykorzystywany jest w systemach rekomendacji. Podobieństwo możemy badań na podstawie harmonii, melodii czy też współwystępowanie pozycji na listach odtwarzania. Wykorzystują metody grafowe oraz embedding. Funkcja loss - triplet loss ma na celu przybliżyć podobnych artystów, a oddalić negatywne przypadki. Wykorzystywany jest dataset OLGA. Porównujący wyniki dla GNN oraz DNN, grafy lepiej prognozują prawdopodobieństwo artystów.

1.2 Leveraging Hierarchical Structures for Few-Shot Musical Instrument Recognition (ISMIR 2021)

Rozpoznawanie instrumentów muzycznych. W przypadku gdy dataset jest ograniczony, jednak chcielibyśmy nauczyć naszą sieć rozpoznawać nowy dźwięk jedynie za pomocą kilku próbek. Wykorzystywany jest system hierarchii, który po wyznaczeniu hierarchii instrumentów umożliwi naukę systemu za pomocą kilku próbek nowej klasy który ma wspólnego przodka w naszej wyznaczonej hierarchii. Wykorzystujemy sieci prototypowe. Początkowo tworzymy sieć na najbardziej szczegółowym poziomie. Następnie wykorzystujemy predefiniowaną sieć do stworzenia nowego prototypu dla grubszych poziomów. Powtarzamy ten proces, co pozwala na rozpoznanie instrumentów na różnych poziomach. Obliczamy embedding vector dla każdej próbki z podzbioru zbioru S zawierającego wszystkie przykłady dla danej klasy k. Tworzymy rozkład prawdopodobieństwa na zbiorze klas dla danego zapytania. Struktura ma postać drzewa. Hierarchiczne sieci prototypowe są rozszerzeniem sieci prototypowych o hierarchiczne uczenie wielozadaniowe (etykietowanie zapytań). Podejście te mogłabym wykorzystać w swojej pracy do rozpoznawania różnego rodzaju instrumentów.

1.3 BebopNet: Deep Neural Models for Personalized Jazz Improvisations (ISMIR 2020)

Generowanie spersonalizowanej muzyki. Generowanie muzyki porównywane jest do NLP. Na podstawie znaków (nut) przewidywany jest kolejny znak (nuta). Aby stworzyć spersonalizowane improwizacje jazzowe przeprowadzane jest badania oraz uczenie preferencji użytkownika,

a następnie zoptymalizowanie generowania muzyki. Aby nauczyć się preferencji użytkownika wykorzystywane jest recurrent regression model. Nuta reprezentowana jest przez 2 wartości (pitch-duration). BebopNet trenujemy za pomocą datasetu solo jazzowych z próbkami w formacie musicXML. Dostarczane dane to: pitch, duration, offset i chord. Model może zostać różnie zaimplementowany: RNN, CNN, LSTM... Dane reprezentujemy przez warstwy embeddingu. Preferencje użytkowników pobierane są na podstawie datasetu - 124 improwizacji (oryginalne jak i wygenerowane przez BebopNet). Wykorzystując transfer learning, tworzymy model preferencji użytkownika składający się z tych samych warstw jak BebopNet, bez końcowych warstwy fully-connected. Wagi warstw embeddingu pozostają zamrożone.

1.4 MINGUS: Melodic Improvisation Neural Generator Using Seq2Seq (ISMIR 2021)

Generowanie muzyki (jazz). Podczas generowania muzyki możemy wykorzystać techniki wykorzystywane w NLP MINGUS przyjmuje na wejściu odpowiednia sekwencje z wartościami takimi jak Pitch, Duration, Chord, Next Chord, Bass, Beat, Offset Zbudowany jest z 2 równoległych modeli, do przewidzenia wysokości dźwięku oraz czasu trwania. Modele te są takie same, różnią się jedynie wyjściem. Oba modele wykorzystują funkcje crossentropy oraz SGD Generowanie muzyki polega na dostarczeniu sekwencji nut o zadanej długości. Następnie wejście dzielone jest na sekwencje wysokości dźwięku oraz czasu trwania, a następnie wprowadzone do odpowiedniego modelu. Wynikiem działania jest token o największym prawdopodobieństwie z wyjścia modelu. Następnie jest on dołączany do naszej wygenerowanej sekwencji

2 Technologie i metodologie

2.1 Obróbka danych muzycznych - audio

2.1.1 Najważniejsze cech dźwięku

- **Spektrogram**
  Analiza widma sygnału w czasie.

  Z okna wyznaczamy widmo sygnały które odzwierciedla składowe dla poszczególnych częstotliwości.

  Spektrogram odzwierciedla sygnał w czasie i częstotliwości.

  Służy do analizy dźwięków. Jest narzędziem które pozwala analizować dźwięk jako obraz.

- **Spectral roll-off – parametr opadania widma**
  Częstotliwość roll-off jest częstotliwością, poniżej której zawarty jest pewien procent całkowitej energii widma. Jest przydatne w odróżnianiu mowy dźwięcznej od bezdźwięcznej; do wykrywania dźwięków harmonicznych oraz hałaśliwych.

- **Spectral flux**
  Parametr szybkości zmiany widma.

  Wyznaczanie na podstawie widma z bieżącego okna dźwięku z poprzednim.

  Wykorzystywany do opisu barwy dźwięku.

  Wykrywanie początków nut/słów ("atak dźwięku")

- **Spectral centroid**
  "środek ciężkości" widma, czyli jaka częstotliwość jest środkiem ciężkości dźwięku.

  Określamy jasność brzmienia (im wyższa wartość tym jaśniejsze).

  Wykorzystujemy np. do rozróżnienia mowy damskiej i męskiej.

- **Zero-crossing rate**
  Jest to współczynnik przejścia przez zero. Określa jak często obserwujemy zmianę znaku sygnału. Jest to pomiar gładkości sygnału, czyli liczba przejść przez zero w próbce dźwięku.

  Wykorzystywany w klasyfikacji muzyki, wykryciu śpiewu, analizie mowy itp.

- **MFCC - Mel-frequency cepstral coefficients**
  Wektor parametrów - współczynniki dla odpowiednich pasm melowych

  Wektor ten odzwierciedla naturalną odpowiedź układu słuchowego na pobudzenie dźwiękami mowy.

  Skala melowa – filtracja dźwięku filtrami pasmowymi

- **Chromogram**
  Jest to reprezentacja dźwięku (12-elementowa), zwanej wektorem chrominancji. Dźwięk jest reprezentowany przez 12 elementów – półtony w oktawie. Każdy półton jest reprezentowany na spektrogramie odpowiednią barwą, w zależności od aktualnej energii półtonu.

- **Beat track**
  Wykrywanie tempa, wyszukiwanie szczytu sił zgodne z szacowanym tempem

- **HPSS**
  Separacja części perkusyjnej i harmonicznej

- **Zero-crossing rate**
  Jest to współczynnik przejścia przez zero. Określa jak często obserwujemy zmianę znaku sygnału. Jest to pomiar gładkości sygnału, czyli liczba przejść przez zero w próbce dźwięku.

  Wykorzystywany w klasyfikacji muzyki, wykryciu śpiewu, analizie mowy itp.

2.1.2 Biblioteki do ekstrakcji cech dźwięku

- **Librosa**
  Jest pakietem Pythona służącym do analizy muzyki i dźwięku. Dostarcza elementów niezbędnych do tworzenia systemów wyszukiwania informacji o muzyce.

  ```
  y, sr = librosa.load(librosa.util.example_audio_file())
  ```

  Za jej pomocą możemy wyznaczyć takie dane jak :

  – STFT Short-Time Fourier Transform

  ```
  Y = librosa.stft(y)
  ```

  STFT reprezentuje sygnał w dziedzinie czasu i częstotliwości poprzez obliczanie dyskretnej transformaty Fouriera (DFT) w krótkich, nakładających się na siebie oknach

– CQT - transformata constant-Q

```
fmin = librosa.midi_to_hz(36)
hop_length = 512
C = librosa.cqt(y, sr=sr, fmin=fmin, n_bins=72, hop_length=hop_length)
```

Przekształca szereg danych do dziedziny częstotliwości. Wykorzystuje logarytmicznie rozłożoną oś częstotliwości

– Chromogram

```
chroma = librosa.feature.chroma_stft(y, sr=sr)
lub
chroma = librosa.feature.chroma_cqt(y=y, sr=sr)
```

Za pomocą stft/cqt możemy uzyskać transformacje dźwięków na wartości liczbowe - chroma

Chroma jest 12-elementowym wektorem mierzącym energię z wysokości dźwięku. Jest to typ danych wektorowych służący do pomiaru energii dźwięku, zwykle w decybelach (dB).

Wektor cech wskazujący, ile energii każdej klasy wysokości dźwięku C, C#, D, D#, E, ..., B jest obecne w sygnale.

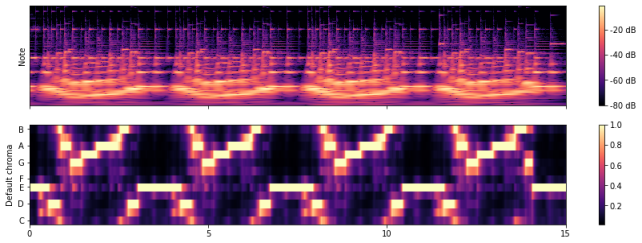

Figure 1: Przekształcenie decybeli na nuty muzyczne

– Melspectogram

```
S = librosa.feature.melspectrogram(y=y, sr=sr)

mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)
```

Jest to spektrogram, w którym częstotliwości są przekształcane na skalę melową.

MFCC (Mel-Frequency Cepstral Coefficients) - cepstrum sygnału przedstawione w skali melowej

Jest wykorzystywany w uczeniu maszynowym dotyczącym plików audio. MFCC przedstawia obwiednie widma. Są to parametry szeroko stosowane w akustyce mowy. Uzyskujemy wektor współczynników cepstrum w odpowiednich pasmach melowych. Odzwierciedlają naturalną odpowiedź układu słuchowego na pobudzenie dźwiękami

mowy, odzwierciedlenie odbioru dźwięku przez ludzkie ucho. Może być wykorzystywana do wykrywania wysokości dźwięku oraz rozpoznawania mowy.

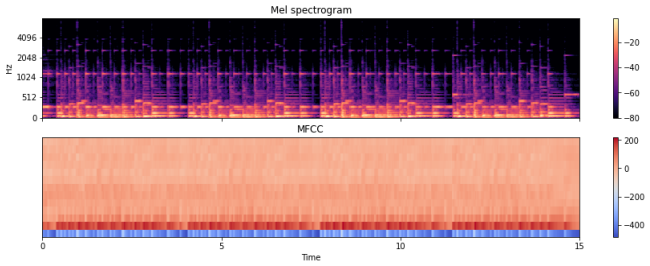

Figure 2: Melspectogram oraz MFCC wyznaczony dla losowej próbki dźwięku

– Beat tracker

```
onset_env = librosa.onset.onset_strength(y, sr=sr, aggregate=np.median)
tempo, beats = librosa.beat.beat_track(onset_envelope=onset_env,sr=sr)
```

Beat track – wykrywanie tempa, wyszukiwanie szczytu sił zgodne z szacowanym tempem.
beat_times jest tablicą znaczników czasowych (w sekundach) odpowiadających wykrytym zdarzeniom pobudzenia, w której sekundzie doszło do pubudzenia Dostajemy przybliżone tempo, oraz rytm/uderzenia w czasie.

– Separacja części perkusyjnej i harmonicznej

```
y_harmonic, y_percussive = librosa.effects.hpss(y)
```

Izolacja składowych perkusyjnych oraz harmonicznych sygnału audio.

– ZCR Zero-crossing rate

```
zcr = librosa.feature.zero_crossing_rate(y)
```

Ilość przejść przez wartość zero w danej ramce. Wykorzystywane jest w kalsyfikacji muzyki, wykrycie śpiewu, analizie mowy itp.

– Spectral flux

```
onset_env = librosa.onset.onset_strength(y=y, sr=sr)
```

Parametr określający szybkość zmian widma, porównanie okien aktualnego z poprzednim. Wykorzystujemy do określenia barw dźwięku. Używany jest również do wykrywania początków nut lub słów, szybkich zmian.

– Spectral centroid

```
cent = librosa.feature.spectral_centroid(y=y, sr=sr)
```

Wskazuje na jakiej częstotliwości skupia się energia widma. Jest obliczane dla każdej ramki w sygnale. Wskazuje ilość wysokiej częstotliwości w dźwięku.

- Pitch (wysokość tonu)

```
pitches, magnitudes = librosa.piptrack(y=y, sr=sr)
```

Wysokość dźwięku to subiektywne postrzeganie fali dźwiękowej przez każdego człowieka, którego nie da się bezpośrednio zmierzyć, określenie które nuty są wyższe, a które niższe.

- Inne możliwości

  * Separacja wokalu od instrumentów
  * Usuwanie ciszy
  * Tworzenie sygnału
  * Tempogram - macierz cech, która wskazuje przewagę określonych temp w danym momencie czasu.
  * Tempo (BMP)
  * Zmiana tempa ...

- **Pydub**
  Jest pakietem Pythona służącym do edycji oraz pracy z plikami audio. Dostarcza elementów umożliwiających edycję pliów audio, tworzenie własnych plików oraz zapis. Pydub umożliwia:

  - Wczytanie pliku z różnych formatów (wav, raw, mp3).

```
from pydub import AudioSegment

song = AudioSegment.from_mp3(sample_file)
```

    Podczas wczytywania plików możemy ustawić takie wartości jak:

    * sample_width - ilość bitów na próbkę
    * channels - kanał mono lub stereo
    * sample_rate
    * start_second - moment od którego zostanie wgrany plik audio
    * duration - ilość sekund do wgrania

  - Odtwarzanie pliku audio
  - Łatwe przycinanie pliku, zapętlanie

```
ten_seconds = 10 * 1000
first_10_seconds = song[:ten_seconds]
last_5_seconds = song[-5000:]
first_10_seconds_loop = first_10_seconds * 2
```

- Zwiększenie oraz zmniejszenie głośności

```
# boost volume by 6dB
first_10_seconds_volume_up = first_10_seconds + 6
# reduce volume by 3dB
first_10_seconds_volume_reduce = first_10_seconds - 3
```

- Złączenie plików audio

```
without_the_middle = first_10_seconds + last_5_seconds
# z przenikaniem
without_the_middle_crossfade = first_10_seconds.append(last_5_seconds,
                    crossfade=5000)
```

- Wykrywanie ciszy
- Stosowanie filtrów
- Tworzenie audio
- Zapis pliku (z tagami)
  Ciekawą możliwością jest zapisywanie plików audio wraz ze znacznikami(tagami) dla kodera.

```
result.export("mashup.mp3", format="mp3",
        tags={'artist':'Various artists', 'album': 'Best of 2011',
        'comments': 'This album is awesome!'})
```

- **Essentia**
  Jest to open-soucowa biblioteka C++ z bindingami do Pythona. Służy do analizy dźwięku oraz wyszukiwania informacji. Jest biblioteką, która dostarcza wiele algorytmów do charakterystyki danych. Tak jak biblioteka Librosa daje wiele narzędzi do pracy z plikami audio, a nawet wykracza poza możliwości tej biblioteki. Wymaga jednak wcześniejszej inicjalizacji algorytmów.

  Algorytmy dostarczane z biblioteki Essentia umożliwiają:

  - Odczytu i zapis prawie wszystkich formatów plików audio (wav, mp3, ogg, flac itd.)
  - PrzetwarzaniE sygnału: FFT, DCT, obcinanie ramek, windowing, obwiednia, wygładzanie
  - Filtry: dolno/wysoko przepustowy ...
  - Deskryptory statystyczne: mediana, średnia, wariancj...
  - Deskryptory w dziedzinie czasu: czas trwania, głośność, częstotliwość przejścia przez zero, logarytm czasu ataku i inne deskryptory obwiedni sygnału
  - Deskryptory spektralne
  - Deskryptory tonalne: dominująca melodia i wysokość dźwięku, , akordy, tonacja i skala...
  - Deskryptory rytmu: detekcja rytmu, BPM...
  - Inne deskryptory wysokiego poziomu
  - Modele uczenia maszynowego: wnioskowanie za pomocą klasyfikatorów SVM i modeli TensorFlow

- **Soundfile**
  Prosta biblioteka do czytania i zapsiu plików audio.

- **Audiomentations**
  Biblioteka do augmentacji danych audio. Zainspirowana biblioteką albumentations. Podstawową funkcją biblioteki audiomentations jest funkcja Compose, dzięki której możemy zastosować podaną sekwencję przekształceń. Przyjmuje ona listę przekształceń, które chcemy wykonać. Podobnie działa funkcja SpecCompose - tylko dla przekształceń spektrogramów.

  Przekształcenia:

  – AddGaussianNoise
    Dodanie szumu gaussowskiego do naszych próbek. Ustawiamy z jakim prawdopodobieństwem zostanie zaaplikowana augumentacja. Przykład użycia:

```
from audiomentations import Compose, AddGaussianNoise

augment = Compose([ AddGaussianNoise(min_amplitude=0.001,
                        max_amplitude=0.15, p=0.5),])

data, samplerate = librosa.load(sample_file)

augmented_samples = augment(samples=data, sample_rate=samplerate)
```

  – TimeStretch
    Rozciąganie w czasie sygnał bez zmiany wysokości dźwięku.

```
augment = Compose([AddGaussianNoise(min_amplitude=0.001,
                max_amplitude=0.015, p=0.5),
                TimeStretch(min_rate=0.8,
                max_rate=1.25, p=1),])
```

  – PitchShift
    Zmiana wysokość dźwięku w górę lub w dół bez zmiany tempa.
  – Shift
    Przesunięcie próbki do przodu lub do tyłu, z nakładaniem lub bez nakładania.
  – SpecChannelShuffle
    Zamienia kanał w spektrogramie wielokanałowym. Funkcja ta może pomóc w zwalczaniu zniekształceń pozycyjnych.
  – SpecFrequencyMask
    Maskowanie zestawu częstotliwości w spektrogramie. Takie działanie okazało się skuteczne w przypadku modeli rozpoznawania mowy.

- **Pedalboard**
  Biblioteka do augmentacji danych audio. Służy do manipulacji dźwiękiem: dodawania efektów, odczytu oraz zapisu danych. Pedalboard to obiekt, który zawiera efekty, które chcemy zastosować do naszego pliku audio.

```
from pedalboard.io import AudioFile
```

```
with AudioFile('sample1.wav', 'r') as f:
  audio = f.read(f.frames)
  samplerate = f.samplerate

from pedalboard import Pedalboard, Chorus, Reverb

board = Pedalboard([Chorus(), Reverb(room_size=0.25)]) # efekty gitarowe
effected = board(audio, samplerate)
```

Pedalboard dostarcza wiele podstawowych przekształceń dźwięku np.

- Efekty gitarowe takie jak Chorus, Distortion, Phaser
- Zwiększanie głośności i zakres dynamiki: Compressor, Gain, Limiter
- Filtry: HighpassFilter, LadderFilter, LowpassFilter
- Efekty przestrzenne:Convolution, Delay, Reverb
- Zmianna wysokości tonu: PitchShift
- Kompresja: GSMFullRateCompressor, MP3Compressor
- Redukcja jakości: Resample, Bitcrush

Objekty klasy Pedalboard zachowują się jak lista, w związku z czym możemy je dodawać tak samo jak do listy. Możemy również w łatwy sposób zmieniać parametry danych efektów zawartych w Pedalboard.

```
board = Pedalboard([
Compressor(threshold_db=-50, ratio=25), # kompresor
Gain(gain_db=30), # wzmacniacz
Chorus(),
LadderFilter(mode=LadderFilter.Mode.HPF12, cutoff_hz=900), # filt Ladder
Phaser(), # efekt gitarowy
Reverb(room_size=0.25),])

#dodanie efektów
board.append(Compressor(threshold_db=-25, ratio=10))
board.append(Gain(gain_db=10))
board.append(Limiter())

#zmiana efektów
board[0].threshold_db = -40
```

Możemy również stworzyć wiele obiektów Pedalboard, które później zostaną wywołane równolegle.

- **Torchaudio**
  Torchaudio to wbudowana w PyTorch biblioteka do przetwarzania plików audio.

  Funkcje dostarczane przez Torchaudio:

- Spectrogram - funkcja tworząca spektrogram z przebiegu audio.
- MelSpectrogram - funkcja tworząca spektrogram MEL z przebiegu audio za pomocą funkcji STFT (Short-time Fourier Transform) w PyTorch.
- Resample - funkcja za pomocą, której możemy przeprowadzić ponowne próbkowanie przebiegu dla innej częstotliwości próbkowania.
- MuLawEncoding - funkcja za pomocą której możemy kodować przebiegi w oparciu o kompandowanie mu-law. Aby skorzystać z tej funkcji potrzebujemy sygnału z przedziału od -1 do 1.
- MuLawDecoding służy do dekodowania przebiegów zakodowanych metodą mu-law.
- Compute_deltas - funkcja służąca do obliczenia współczynnika delta tensora.
- Gain - funkcja, dzięki której możemy zastosować wzmocnienie lub tłumienie do całego przebiegu.
- Diter - funkcja zwiększająca postrzegany zakres dynamiki dźwięku zapisanego w określonej głębi bitowej.
- Zastosowanie filtrów dolno i górnoprzepustowych
- I wiele innych jak: GriffinLim - oblicza kształt fali ze spektrogramu w skali liniowej przy użyciu transformacji Griffina-Lima, ComputeDeltas - oblicza współczynniki delta tensora, ComplexNorm - oblicza normę złożonego tensora, MelScale - zmienia normalny STFT w STFT o częstotliwości Mel, używając macierzy konwersji, AmplitudeToDB - zmienia spektrogram ze skali mocy/amplitudy na skalę decybeli, TimeStretch - rozciąga spektrogram w czasie bez zmiany wysokości tonu dla danej szybkości, FrequencyMasking - zastosuj maskowanie do spektrogramu w dziedzinie częstotliwości, TimeMasking - zastosuj maskowanie do spektrogramu w dziedzinie czasu.

- **Mirdata**
  Mirdata to biblioteka, która ma na celu standaryzację dostępu do zestawów danych audio w Pythonie, eliminując potrzebę pisania niestandardowych Data Loader w każdym projekcie i poprawiając odtwarzalność. Umożliwia pobierania zbiorów danych do wspólnej lokalizacji i wspólnego formatu.

  Lista wszystkich dostępnych Data Loader.
  ['acousticbrainz_genre', 'beatles', 'beatport_key', 'billboard', 'cante100', 'compmusic_jingju_acappella', 'compmusic_otmm_makam', 'da_tacos', 'dagstuhl_choirset', 'dali', 'freesound_one_shot_percussive_sounds', 'giantsteps_key', 'giantsteps_tempo', 'good_sounds', 'groove_midi', 'gtzan_genre', 'guitarset', 'haydn_op20', 'ikala', 'irmas', 'maestro', 'medley_solos_db', 'medleydb_melody', 'medleydb_pitch', 'mridangam_stroke', 'mtg_jamendo_autotagging_moodtheme', 'orchset', 'phenicx_anechoic', 'queen', 'rwc_classical', 'rwc_jazz', 'rwc_popular', 'salami', 'saraga_carnatic', 'saraga_hindustani', 'slakh', 'tinysol', 'tonality_classicaldb', 'tonas', 'vocadito']

  Wybieramy jeden z dostępnych Data Loader za pomocą zainicjowania go. Wszystkie Dataset Loaders zawierają funkcję download(), która pozwala pobrać kanoniczną wersję dataset'u. Domyślnie wszystko jest pobierane/czytane do/z lokalizacji " /mir_datasets/wybrany_dataloader", którą możemy zmienić w initialize za pomocą argumentu data_home=path. Za pomocą funkcji validate() możemy sprawdzić, czy pobrany Dataset jest poprawny (czy nie jest uszkodzony) oraz czy jego lokalna wersja jest zgodna z dostępną wersją kanoniczną.

  Możemy również nie pobierać na raz całego Datasetu, tylko pobrać elementy, które nas interesują.

```
cante100 = mirdata.initialize('cante100')
cante100.download(partial_download=['spectrogram', 'melody', 'metadata'])
```

2.2 Obróbka danych muzycznych - MIDI

MIDI - cufrowy interfejs instrumentów muzycznych. Umożliwia komputerom, kartom dźwiękowym, syntezatorom wymieniać informację między sobą. MIDI jest zapisem nutowym.

Inne notacje

- MusicXML
  Uniwersalny format dla powszechnie stosowanej zachodniej notacji muzycznej, podobnego do formatu MP3, który służy do zapisu muzyki.
- Notacja ABC
  Notacja ABC to prosty, ale wydajny format notacji muzycznej ASCII. Istnieje też wiele pakietów oprogramowania umożliwiających konwersję zapisu abc na MIDI, tworzenie nut, odtwarzanie pliku przez głośnik komputera itp.
- GUIDO
  Format tekstowy, zdolny do reprezentowania wszystkich informacji zawartych w konwencjonalnych partyturach muzycznych.

Biblioteki MIDI w pythonie:

- **Mido**
  Biblioteka Mido służy do obsługi portów oraz messages z pliku MIDI. Mido umożliwia również do tworzenia nowych plików MIDI.

```
from mido import Message, MidiFile, MidiTrack

mid = MidiFile()
track = MidiTrack()
mid.tracks.append(track)

track.append(Message('program_change', program=12, time=0))
track.append(Message('note_on', note=64, velocity=64, time=32))
track.append(Message('note_off', note=64, velocity=127, time=32))

mid.save('new_song.mid')
```

- **music21**
  Biblioteka ta umożliwia obsługę różnych notacji muzycznych.

```
c = converter.Converter()
for sc in c.defaultSubconverters():
    print(sc)
```

```
<class 'music21.converter.subConverters.ConverterABC'>
<class 'music21.converter.subConverters.ConverterBraille'>
<class 'music21.converter.subConverters.ConverterCapella'>
<class 'music21.converter.subConverters.ConverterClercqTemperley'>
<class 'music21.converter.subConverters.ConverterHumdrum'>
<class 'music21.converter.subConverters.ConverterIPython'>
<class 'music21.converter.subConverters.ConverterLilypond'>
<class 'music21.converter.subConverters.ConverterMEI'>
<class 'music21.converter.subConverters.ConverterMidi'>
<class 'music21.converter.subConverters.ConverterMuseData'>
<class 'music21.converter.subConverters.ConverterMusicXML'>
<class 'music21.converter.subConverters.ConverterNoteworthy'>
<class 'music21.converter.subConverters.ConverterNoteworthyBinary'>
<class 'music21.converter.subConverters.ConverterRomanText'>
<class 'music21.converter.subConverters.ConverterScala'>
<class 'music21.converter.subConverters.ConverterText'>
<class 'music21.converter.subConverters.ConverterTextLine'>
<class 'music21.converter.subConverters.ConverterTinyNotation'>
<class 'music21.converter.subConverters.ConverterVexflow'>
<class 'music21.converter.subConverters.ConverterVolpiano'>
<class 'music21.converter.subConverters.SubConverter'>
```

Figure 3: music21 - obsługiwane notacje

Umożliwia tworzenie melodii z zadanej adnotacji.

```
melody = converter.parse("tinynotation: 3/4 c4 d8 f g16 a g f#")
```

- **pretty_midi**
  Biblioteka służy jedynie do pracy z plikami w formacie MIDI.

  ```
  midi_data = pretty_midi.PrettyMIDI(exampl_file.mid)
  ```

  Plik MIDI przechowuje informację o:

  - .instruments - odpowiadają kanałom (instrumentom)
    - .program (rodzaj)
    - .is_drum
    - .name
    - .notes (lista obiektów, nut zawierające takie informacje jak wysokość tonu, głośności, czasu)
    - .pitch_bends
    - .control_changes
  - .key_signature_changes - metainformacja
  - .time_signature_changes

– .lyrics

Są to listy obiektów przechowujące dokładne informację.

Biblioteka pozwala na syntezę dźwięku. Domyślnie synteza dokonana jest sinusoidalnymi falami lub własnymi.

```
idx = 10**6 // 2
wav = midi_data.synthesize()
ipd.Audio(wav[:idx], rate=44100)
```

Metody z biblioteki pretty_midi pozwalają na:

– zmiana tempa
– estymacja tempa
– histogram dla pitch
– chroma
– zmiana czasu na tick (zmiana reprezentacji)
– zmiana reprezentacji tekstowej w liczbową
– zmiana instrumentów dla syntezy

- **pypianoroll**
Służy do pracy z plikami pianoroll. Obsługuje jedynie pliki MIDI. Dane reprezentowane są w postaci macierzy 128XN (długość utworu). Metody oraz reprezentacja danych są podobne do biblioteki pretty_midi.

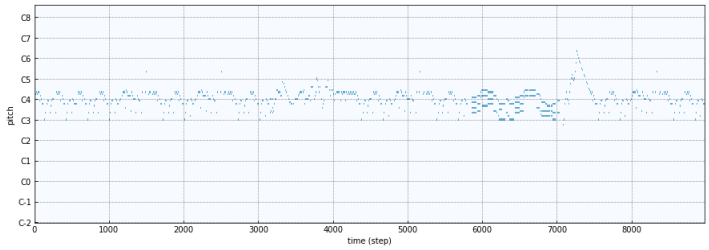

Figure 4: Wizualizacja wszystkich kanałów pliku MIDI

Biblioteka pypianoroll udostępnia metody do modyfikacji: transpozycje, przycięcie utworu, zmiana długości, liczba różnych tonów, zakres tonu, procent zadanych nut w pewnej tonacji.

- **MusPy**
MusPy jest biblioteką, która umożliwia pracę z różnymi danymi, takimi jak: MIDI, MusicXML, ABC, YAML, JSON. Reprezentacja danych zbliżona jest do reprezentacji w wyżej wymienionych bibliotekach, z pewnymi rozszerzeniami np. akordy. Biblioteka MusPy udostępnia 4 podstawowe reprezentacje muzyki symbolicznej:

– pitch (wysokość tonu)

```
music.to_pitch_representation(use_hold_state=True)
```

– reprezentacja pypianoroll

```
music.to_pianoroll_representation(encode_velocity=True)
```

– Zdarzenia (początek/koniec nuty)

```
music.to_event_representation()
```

– reprezentacja nutowa - czas, ton, długość oraz głośność

```
music.to_note_representation()
```

MusPy wspiera pracę na zbiorach danych. Pozwala na automatyczne pobranie, wypakowanie oraz pracę z danymi.Wpiera również konwersję do datasetu PyTorch lub Tensorflow.

Metryki - MusPy udostępnia również metryki do analizy jakości.

Pliki MIDI a pliki audio Praca z plikami MIDI dostarcza potrzebne dane do analizy, łatwiejsza ekstrakcja danych niż z pliku danych. Pracując z plikami audio możemy napotkać się z problemem zaszumionych danych, audio z szumem.

2.3 Tagowanie i analiza muzyki

Potrzeba taggowania oraz analizy muzyki powstała wraz z rozwojem aplikacji związanych ze światem muzyki. Korzystając z takich aplikacji jak Spotify, Youtube music czy Tidal spotykamy się z różnymi playlistami, różnymi gatunkami czy też systemami rekomendacyjnymi. Aby uniknąć ręcznego wprowadzania oraz aktualizacji danych na temat każdego pliku audio powstało zapotrzebowanie na rozwiązanie z dziedziny sztucznej inteligencji. Podstawowe tagi:

- Gatunek
- Nastroje
- Instrumenty
- Wokale (lub same ich wystąpienie)
- Tempo
- Głośność dźwięku

Istnieje wiele innych możliwych tagów, takich jak analiza samego teksu muzyki, wykrycie na jego podstawie nastroju, przekazu. Możemy również tagować artystów, wykorzystywane jest to w systemach rekomendacji.
Metody tagowania muzki:
Podstawowym modelem wykorzystanym w tagowaniu muzyki są sieci konwolucyjne.

- CNN - Możemy w różny sposób wykorzystać sieci konwolucyjne. Analizie możemy poddać cały plik audio oraz dokonać klasyfikacji

- FCN – operacja na short-chunk, czyli krótkich fragmentach, kilkusekundowych. Operujemy na krótkich fragmentach w celu łatwiejszej analizy oraz wychwytywaniu danych tagów. Przykładem jest wystąpienie danego instrumentu tylko w krótkim fragmencie pliku audio. Za pomocą analizy na fragmentach jesteśmy w stanie uzyskać dokładniejszą analizę muzyki.
- Harmoic CNN – wytrenowanie filtrów w sieciach konwolucyjnych na podstawie utworów w celu ułatwienia wyszukiwania przy małej liczbie danych.
- MusiCNN – filtry w CNN zostają manualnie stworzone. Dzięki temu możemy w łatwiejszy sposób znaleźć charakterystyki tonów instrumentów lub wykrycie tempa.
- Sample-level CNN – operują na próbkach w zapisie częstotliwości w czasie. Umożliwia analizę dla każdej próbki.
- CRNN – sieci rekurencyjne. Przetwarzamy dane bazując na danych poprzedzających. Wykorzystywane do przetwarzania mowy i sekwencji.
- Music tagging transformer – połączenie sieci rekurencyjnych z sieciami konwolucyjnymi.

3 Prototyp

W ramach prototypu opierałam się na **implementacji sieci neuronowej CNN.** Przeanalizowałam oraz wprowadziłam we własnej implementacji różne biblioteki do augumentacji danych. Dokonałam analizy oraz wprowadziłam pewne zmiany parametrów sieci oraz biblioteki do augumentacji danych (w tym przypadku została wykorzystana biblioteka **audiomentations**), co umożliwiło mi uzyskanie lepszych wyników niż dla sieci z przykładu.

Rozpoczęłam również próby pracy nad siecią CRNN, rozwiązanie to jest jeszcze w fazie przygotowań.

Link do colaba z implementacją.

W ramach pracy nad deploymentem oraz prezentacji działania w przyszłości wytrenowanych sieci, zapoznałam się z dwoma bibliotekami: gradio oraz streamlit. W ramach przygotowania stworzyłam proste aplikację wykorzystujące podane powyżej biblioteki:

- **Gradio**. - link do repozytorium z aplikacją służącej do klasyfikacji zdjęć. Wykorzystany został model ResNet50.
- **Streamlit**. - link do repozytorium z aplikacją do przewidywania czasu dostawy przesyłki na podstawie czasu zakupu, miasta oraz firmy kurierskiej. Użytkownik ma możliwość wyboru modelu z jakiego będzie korzystał.

Gradio jest biblioteką umożliwiającą aplikacji interfejsu uzytkownika wykorzystujacej modele uczenia maszynowego. Streamlit natomiast jest bardzo wygodny do wprowadzania danych, wyboru różnego typu modeli oraz przedstawienia wyników. W swojej pracy inżynierskiej planuje wykorzystać jedną w powyższych bibliotek do prezentacji wyników zbudowanych sieci.

4 Plan pracy na kolejny semestr

W ramach dalszej pracy nad pracą inżynierską planuje skupić się nad poszukiwaniem DataSetów do wytrenowania sieci w celu uzyskania jak najlepszych wyników dla odpowiednich tagów, takich jak gatunek czy instrument. Planuje również skupić się na implementacji różnych sieci neuronowych służących do tagowania muzyki takich jak CRNN, FCN czy MusicCNN oraz porównać

ich wyniki. Rozważam również próbę wykorzystania sieci prototypowych do tagowania instrumentów.

References

[1] Artist Similarity with Graph Neural Networks,
https://arxiv.org/abs/2107.14541

[2] Leveraging Hierarchical Structures for Few-Shot Musical Instrument Recognition (ISMIR 2021),
https://archives.ismir.net/ismir2021/paper/000027.pdf

[3] MINGUS: Melodic Improvisation Neural Generator Using Seq2Seq (ISMIR 2021),
https://archives.ismir.net/ismir2021/paper/000051.pdf

[4] BebopNet: Deep Neural Models for Personalized Jazz Improvisations (ISMIR 2020),
https://program.ismir2020.net/poster_6-08.html

[5] CNN Music Tagging,
https://music-classification.github.io/tutorial/part3_supervised/tutorial.html

[6] Gradio,
https://gradio.app/

[7] Streamlit,
https://streamlit.io/

[8] Spotify - pedalboard,
https://github.com/spotify/pedalboard

[9] Audiomentations,
https://github.com/iver56/audiomentations

[10] Torchaudio,
https://pytorch.org/audio/main/tutorials/audio_data_augmentation_tutorial.html

[11] Librosa,
https://librosa.org/doc/latest/index.html

[12] Essentia,
https://essentia.upf.edu/documentation.html