Automatyczna Transkrypcja Pianina do Pliku Midi

Patryk Gawłowski (263646)[1] **and Jakub Wolski (263619)**[1]

[1] *Student Politechniki Wrocławskiej*

Nadzorowane przez Panią mgr inż. Monikę Wasilewską

Abstract—*Celem tego projektu było opracowanie systemu automatycznej transkrypcji pianina, który konwertuje nagrania audio na pliki MIDI z wykorzystaniem sieci neuronowych. System korzysta z bazy danych MAPS, składającej się zarówno z prawdziwych, jak i syntetycznych nagrań pianina, do trenowania i walidacji modelu transkrypcji. Kluczowe etapy projektu obejmowały wstępne przetwarzanie danych, ekstrakcję cech za pomocą transformaty Constant-Q (CQT) oraz stworzenie struktury danych kompatybilnych z wejściem sieci neuronowej. Zastosowano architekturę konwolucyjnej sieci neuronowej, która uwzględnia okno kontekstu sąsiednich klatek audio w celu poprawy zdolności predykcyjnych modelu. Oceniono różne strategie trenowania, a wydajność modelu mierzono za pomocą metryki F1-Score. Najwyższą wydajność osiągnięto, trenując model od podstaw na pełnym zbiorze danych, uzyskując F1-Score na poziomie 36,56%. Pomimo napotkanych trudności w transkrypcji muzyki polifonicznej, projekt wykazał możliwość automatycznej transkrypcji muzyki za pomocą sieci neuronowych oraz zidentyfikował obszary do przyszłych usprawnień, w tym dynamiczną regulację wskaźnika uczenia się i ulepszoną generację plików MIDI. Wyniki stanowią podstawę do dalszych badań i rozwoju w dziedzinie automatycznej transkrypcji muzyki.*

Keywords—*sprawozdanie z projektu, AI, automatic music transcription (AMT), music information retrieval (MIR), Python*

1. Wstęp

Automatyczna transkrypcja muzyki to proces przekształcania nagrania dźwiękowego (np. utworu muzycznego) na formę zapisu nutowego lub innej symbolicznej reprezentacji muzycznej. Nie jest ona popularnie używana, ze względu na skomplikowaność. Do transkrypcji konieczne jest rozpoznanie grającego instrumentu, tempa utworu, wysokości dźwięków i czasu ich trwania. Aby uprościć to zadanie zdecydowano się ograniczyć zadanie jedynie do transkrypcji pianina. Celem było utworzenie pliku midi na podstawie zadanego pliku audio.

Jako punkt odniesienia na którym się wzorowano uznano artykuł An AI Approach to Automatic Natural Music Transcription [1].

2. Dane

W niniejszej pracy wykorzystano bardzo popularną bazę danych MAPS. Zawiera ona 65 godzin nagrań audio pianina. Część nagrań jest pobrana z prawdziwych instrumentów, a część wygenerowana z plików MIDI. Baza danych dzieli się na cztery główne kategorie:

- ISOL - pojedyncze dźwięki i monofoniczne sekwencje dźwięków
- RAND - akordy zawierające losowe dźwięki
- UCHO - popularne akordy używane w zachodniej muzyce
- MUS - pełne utwory muzyczne

Ponadto nagrania zawarte w bazie były wykonywane przy różnych warunkach akustycznych i na różnych modelach instrumentów. Do nauki i testowania sieci wykorzystaliśmy nagrania z fortepianu Yamaha Mark III nagrywanego bez wpływu otoczenia oraz wtyczki VST Concert Grand D od Native Instruments nagrywanej w studiu. Do każdego nagrania dołączona jest także etykieta w postaci pliku .txt, w której opisane są czas początku dźwięku, koniec, i wysokość jako indeks pozycji w klawiaturze midi. Oprócz tego dołączony jest także sam plik midi odwzorowujący nagranie audio.

Plik MIDI (Musical Instrument Digital Interface) to cyfrowy format przechowywania muzyki, który nie zawiera rzeczywistych dźwięków, lecz informacje o nutach, instrumentach i innych parametrach muzycznych. Przykładowy plik midi został ukazany na zdjęciu 1.

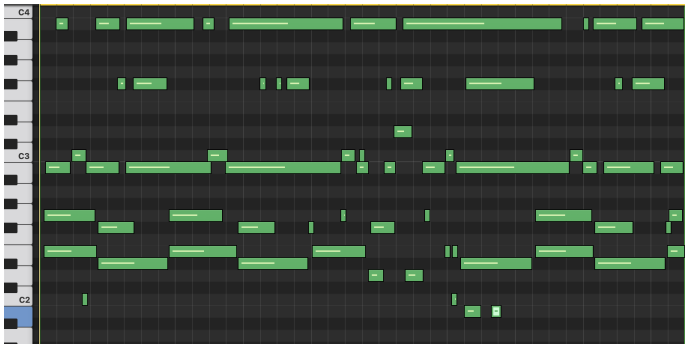


Figure 1. Przykładowy wygląd pliku midi

3. Ekstrakcja Cech

Aby móc skorzystać z danych z bazy MAPS należało je odpowiednio przetworzyć - zarówno pliki *.wav* jak i *.mid*. Poniżej przedstawiono proces ekstrakcji cech z tych plików.

3.1. Preprocessing

Pierwszym krokiem przygotowania danych było zmniejszenie częstotliwości próbkowania plikow *.wav* poprzez tzw. "downsampling" - wykorzystano to aby pozbyć się zbędnej informacji i przyspieszyć proces obliczeń. Dokonano tego w programie za pomocą biblioteki *librosa* [2] i funkcji `load()`:

```
y, sampling_rate=librosa.load(audio, sr=sampling_rate)
```

gdzie `audio` to ścieżka do pliku dźwiękowego a `sampling_rate` to częstotliwość próbkowania z jaką ta funkcja ma pobierać plik. Dla celów tego projektu używano częstotliwość próbkowania 16 kHz.

3.2. CQT

Następnie obliczono dla plików *.wav* transformatę CQT również za pomocą biblioteki *librosa*, tym razem za pomocą instrukcji `cqt()`:

```
cqt = librosa.cqt(y, sr=sampling_rate, hop_length=
    hop_length, n_bins=n_bins)
```

CQT, czyli Constant-Q Transform, jest zaawansowaną metodą przekształcania sygnału dźwiękowego z dziedziny czasu do dziedziny częstotliwościowej. Jest alternatywą dla powszechnie stosowanej FFT, oferującą szereg zalet w kontekście analizy muzyki i innych sygnałów audio. Główną różnicą między CQT a FFT jest sposób grupowania częstotliwości: w CQT pasma częstotliwości są dostosowane proporcjonalnie do wysokości tonu, co lepiej odwzorowuje percepcję ludzkiego słuchu. Dzięki temu, że CQT używa pasm o stałej proporcji, lepiej nadaje się do analizy sygnałów muzycznych, gdzie ważne są różnice proporcjonalne w częstotliwościach, takie jak zmiany w oktawach. Poniżej, na rysunku 2, predstawiono plik muzyczny przekształcony transformatą CQT.

Tak jak autorzy referencyjnego dokumentu ustawiliśmy parametr hop length, określający ilość próbek na klatkę, na 512. Co w rezultacie daje prędkość klatkową $16{,}000/512 = 31.25$ klatek na sekundę.

Dla celów tego projektu przyjęto, że każde pianino posiada 88 klawiszy - od nut A0 (27.5 Hz) do C8 (4186.009 Hz) [3]. Zatem z transformaty CQT otrzymano macierz o wymiarach `(num_frames, 88)`, gdzie `num_frames` to liczba klatek danego pliku muzycznego.

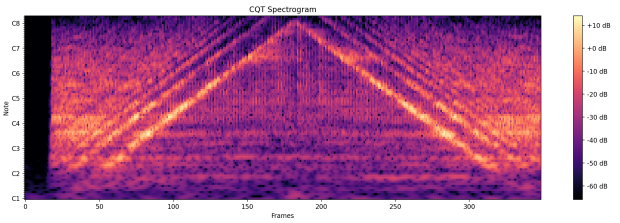


Figure 2. Transformata CQT na pliku z chromatyczną gamą wznoszącą i opadającą

3.3. Grupowanie klatek

Podobnie jak w papierze [1] chciano wykorzystać strukturę uczenia w sieci neuronowej podobną do struktury w sieci LSTM (ang. *Long Short-Term Memory*), co opisano również w rozdziale 4.1, zatem w trakcie przetwarzania i tworzenia klatek z pliku *.wav* automatycznie grupowano je razem z trzema poprzednimi klatkami i trzema kolejnymi klatkami (jeśli one istniały). W ten sposób na wyjściu przetwarzania plików dźwiękowych uzyskano macierz trójwymiarową o wymiarach `(num_frames, 88, 7)`, gdzie `7` wskazuje na liczbę klatek w jednej grupie klatek. Najważniejsza klatka znajduje się na środku grupy podczas gdy reszta przekazuje informacje o otoczeniu tej klatki (quasi-historia).

3.4. Etykiety

W przypadku przygotowania plików *.mid* to wykorzystano je aby stworzyć etykiety do procesów uczenia oraz walidacji. Zatem przekształcono informację o nutach z tych plików na macierz etykiet o wymiarach `(num_frames, 88)`. Aby tego dokonać wykorzystano pakiet Pythona *pretty-midi* [5], który pozwala na analizę, manipulację i przetwarzanie plików MIDI. Wykorzystano obiekty takie jak `pm.instruments[0].notes`, który pobiera nuty instrumentu z listy instrumentów (0 to uniwersalny numer dla pianina w plikach MIDI), `note.start` i `note.end`, które wskazują na czas rozpoczęcia i zakończenia danej nuty oraz `note.pitch`, który podaje informację o wartości granej nuty.

Korzystając z wygenerowanych etykiet można było je wyświetlić za pomocą tzw. *piano roll*, który ukazywał umiejscowienie nut oraz ich czas trwania - tą metodę później wykorzystano do wizualnego sprawdzenia jakości predykcji wykonanej przez nauczoną sieć. Przykładowy piano roll etykiet ukazano na rysunku 3:

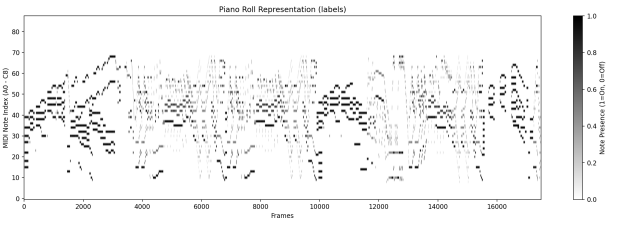


Figure 3. Przykład wygenerowanego piano roll na podstawie etykiet

3.5. Serializacja

Kiedy przetworzono w powyższy sposób wszystkie potrzebne dane z bazy danych zdecydowano się zachować je w jednym pliku aby usprawnić i nie musieć powtarzać tej procedury gdyż może ona momentami trwać długo. Aby dokonać serializacji wykorzystano pakiet Python *pickle* [4], który pozwala zapisać dane w postaci listy, słownika, etc. w jednym pliku za pomocą polecenia:

```
pickle.dump((cqt_array, label_array), f)
```

gdzie `cqt_array` to macierz z danymi z transformaty CQT, `label_array` to macierz z etykietami a `f` to nazwa pliku, do którego te obiekty są zapisywane.

Natomiast aby załadować te pliki i w odpowiedni sposób je rozpakować skorzystano z polecenia:

```
cqt, labels = pickle.load(f)
```

gdzie `cqt` i `labels` odpowiednio przyjmują macierz z danymi od CQT i macierz z etykietami a `f` to nazwa pliku z zserializowanymi danymi.

4. Sieć Neuronowa

4.1. Struktura sieci

Przyjęto architekturę sieci taką samą jak autorzy referencyjnego artukułu. Przedstawia ją rysunek 4.

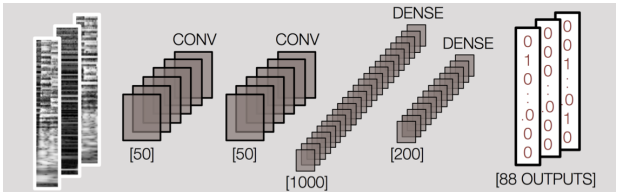


Figure 4. Architektura sieci

Rozważano użycie sieci LSTM, zdecydowano się jednak na zastosowanie sieci konwolucyjnej ze "sztuczną pamięcią". Polega to na tym, że oprócz klatki dla której dokonjemy predykcji na wejście sieci podajemy również kontest w postaci 3 poprzednich i 3 nastepnych klatek.

Pierwszą warstwą jest warstwa konwolucyjna zawierająca 50 filtrów z rozmiarem kernela (25,5). Następnie warstwa MaxPooling o rozmiarze (3,1), funkcją aktywacji tangens hiperboliczny oraz dropout 0.3.

Kolejna warstwa konwolucyjna zawierała również 50 filtrów jednak o rozmiarze (5,3). Ponownie taka sama warstwa MaxPooling i funkcja aktywacji tangens hiperboliczny oraz dropout 0.3. Dla każdej z warstw konwolucyjnych do incjializacji wag użyto inicjalizacji He normal.

Kolejne były dwie, zwykłe, w pełni połączone warstwy. Pierwsza zawierająca 1000 neuronów, druga 200. Obie z sigmoidalną funkcją aktywacji i dropoutem 0.3.

4.2. Uczenie sieci

Przy procesie uczenia sieci zdecydowano się nauczyć sieć na różne sposoby, aby móc je potem ze sobą porównać. Nauczono modele sieci dla następujących baz danych:

- M1 - pojedyncze dźwięki
- M2 - pełna baza danych od zera
- M3 - utwory muzyczne
- M4 - rozszerzenie modelu nauczonego na pojedynczych dźwiękach (M1) o pełną bazę danych

4.3. Kryterium oceny

Do oceny modelu użyto F1-Score, którą można interpretować jako średnią ważoną czułości (R) i precyzji (P), gdzie idealny wynik F1-Score wynosi 1.0, a najgorszy 0.0. Tą metrykę można opisać wzorem 3:

$$R = \frac{TP}{TP + FN} \tag{1}$$

$$P = \frac{TP}{TP + FP} \tag{2}$$

$$\text{F1-Score} = \frac{2 \cdot R \cdot P}{R + P} \tag{3}$$

gdzie

- TP - ilość prawdziwych pozytywów (jest 1 gdzie ma być 1)

- *FP* - ilość fałszywych pozytywów (jest 1 gdzie ma być 0)
- *FN* - ilość fałszywych negatywów (jest 0 gdzie ma być 1)

Używanie F1-Score pozwala nam radzić sobie z ekstremalną nierównowagą klas w danych, ponieważ jest znacznie więcej 0 niż 1 w naszych celach (więcej niegranych nut niż granych). Popularne i najczęściej stosowane *accuracy* nie pozwalałaby nam efektywnie ocenić wydajności naszego modelu (gdyby model przewidywał same zera, dokładność wynosiłaby już około 96%) [1].

4.4. Parametry uczenia

Proces uczenia utrzymano taki sam dla wszystkich przypadków - czyli taki sam schemat i takie same parametry oraz taki sam warunek zatrzymania uczenia.

Dla każdego uczonego modelu skorzystano z funkcji dzielącej zbiór danych na zbiór testowy oraz walidacyjny, która pochodzi z pakietu *sklearn* [6]:

```
cqt_train, cqt_test, label_train, label_test =
    train_test_split(cqt, labels, test_size=0.2)
```

gdzie `cqt_train` i `cqt_test` to odpowiednio zbiór testowy i walidacyjny dla danych z CQT a `label_train` i `label_test` to odpowiednio zbiór testowy i walidacyjny dla etykiet. Podział tych zbiorów był ustawiony na 80% dla zbioru testowego a 20% dla zbioru walidacyjnego.

Do najważniejszych parametrów należały

- `learning_rate` $= 0.01$
- `batch_size` $= 32$

Natomiast każdy model uczył się do momentu aż nie było widać wyraźnej poprawy w kryterium oceny (F1-Score) w kolejnych epokach.

5. Wyniki Uczenia

Poniżej przedstawiono ostateczne wyniki F1-Score po uczeniu każdego z modeli:

- M1: 70.27% F1-Score
- M2: 36.56% F1-Score
- M3: 28.23% F1-Score
- M4: 34.85% F1-Score

Na początkowym modelu M1 (pojedyncze dźwięki) udało się dotrzeć do momentu gdzie metryka wskazywała 70% co było motywującą oznaką aczkolwiek w momencie kiedy się przerzucono na pełną bazę danych, czyli wprowadzono wiele dźwięków w jednym momencie (tzw. polifoniczne dźwięki), ta jakość znacznie spadła. Najlepiej wypadł model M2, który był uczony na pełnej bazie od zera, wyprzedza model M4 o kilka procent jakości zatem nie ma wielkiej różnicy, choć spodziewano się, że może douczony model szybciej i lepiej rozpozna wzorce w polifonicznych dźwiękach.

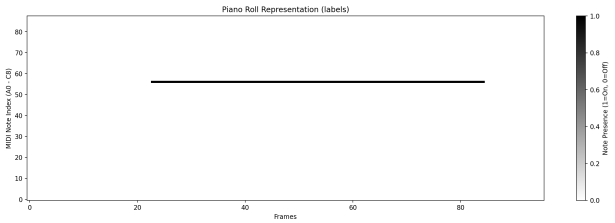
Porównując do wyników autorów papieru [1], na którym bazowano ten projekt, to różnice wyszły całkiem spore. W artykule oznajmiono, że ich ostateczny model osiągnął jakość ok. 55.07% F1-Score, zatem nasz M2 się różni o ok. 19%. Głównymi różnicami pomiędzy tymi modelami były baza, na której model się uczył oraz parametr `learning_rate`, o którym napisano szerzej w rozdziale 7.

6. Predykcje wykonane przez modele

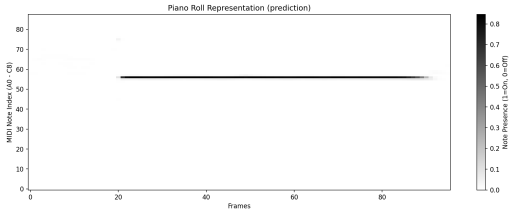
Poniżej przedstawiono predykcje wykonane przez dwa modele (M1 oraz M2) dla plików dzwiękowych o następujących zawartościach:

- pojedyncza nuta
- akord
- chromatyczne gamy wznoszące i opadające
- utwór muzyczny

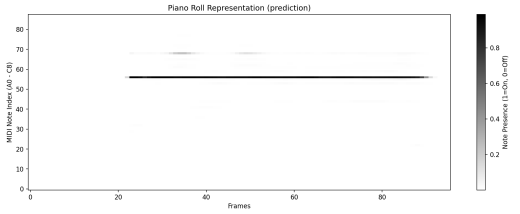
Poniżej przedstawiono ich prawdziwe etykiety oraz predykcje obu modeli w postaci piano roll.



(a) Oryginalna etykieta

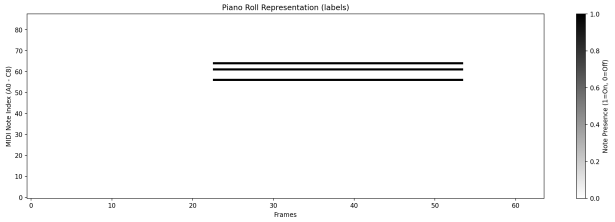


(b) Predykcja M1 (pojedyncze nuty)

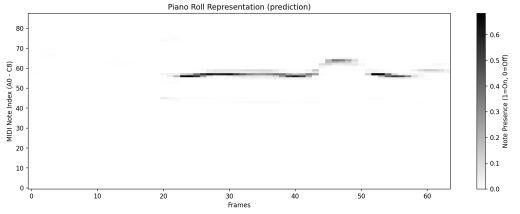


(c) Predykcja M2 (pełna baza danych)

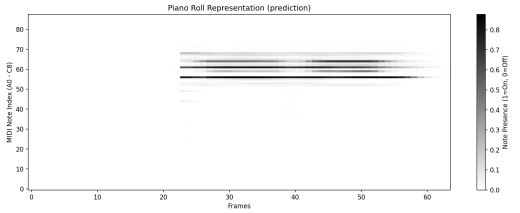
Figure 5. Predykcje dla pliku z pojedynczą nutą



(a) Oryginalna etykieta

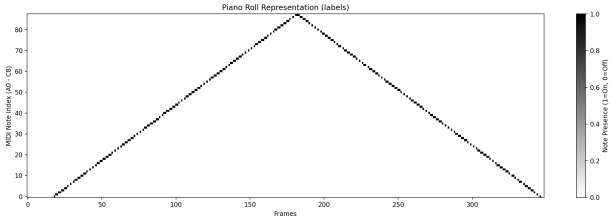


(b) Predykcja M1 (pojedyncze nuty)

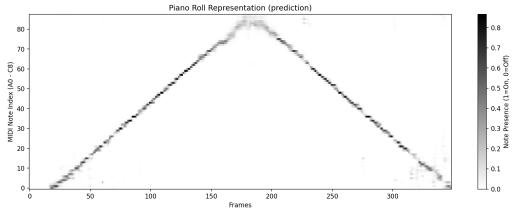


(c) Predykcja M2 (pełna baza danych)

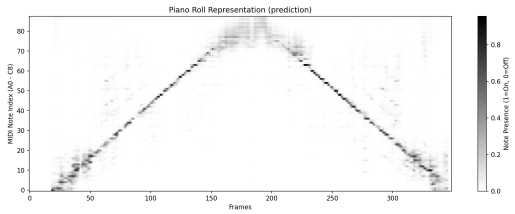
Figure 6. Predykcje dla pliku z akordem



(a) Oryginalna etykieta

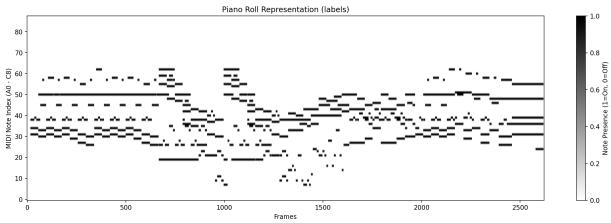


(b) Predykcja M1 (pojedyncze nuty)

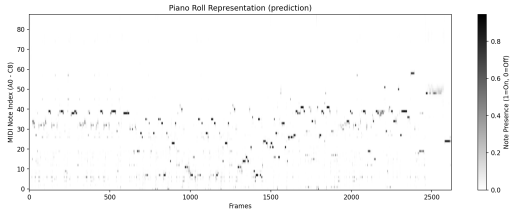


(c) Predykcja M2 (pełna baza danych)

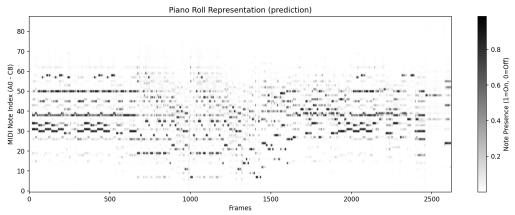
Figure 7. Predykcje dla pliku z chromatyczną gamą wznoszącą i opadającą



(a) Oryginalna etykieta



(b) Predykcja M1 (pojedyncze nuty)



(c) Predykcja M2 (pełna baza danych)

Figure 8. Predykcje dla pliku z utworem muzycznym

Jak można było się spodziewać model nauczony na bazie z pojedynczymi nutami radzi sobie dobrze z pojedynczymi nutami a nie radzi sobie wcale z predykcją wielu nut. Z kolei model wyuczony na pełnej bazie, trochę gorzej spredykował pojedyncze nuty w porównaniu z modelem M1, natomiast znacznie lepiej poradził sobie z dźwiękami polifonicznymi (akord oraz utwór muzyczny).

6.1. Binaryzacja predykowanych macierzy

Kolejnym krokiem przetworzenia wyników z predykcji sieci neuronowej to była binaryzacja predykowanej macierzy nut. Wszystkie wartości w macierzy miały wartość w zakresie od 0 do 1 - im sieć uznała nutę za bardziej prawdopodobną tym komórka macierzy miała większą wartość. Jednak do konwersji do pliku MIDI potrzebne nam są wartości zero-jedynkowe zatem stwierdzono, że dla powyżej pewnego progu nuty są wciśnięte a poniżej - nie grają.

Wartośc tego progu dobrano empirycznie - poprzez sprawdzenie binaryzacji predykcji dla skrajnych wartości i schodząc coraz bliżej środka (0.5). Tym sposobem zauważono, że najlepsze wyniki zachodzą dla wartości progu równym 0.4. Poniżej przedstawiono wyniki skrajnych wartości progu oraz ten wybrany.

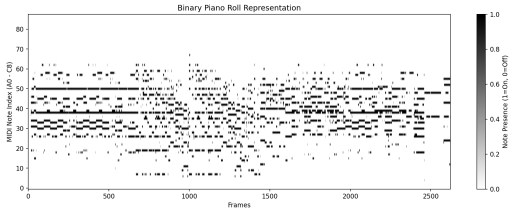


Figure 9. Zbinaryzowana macierz z wartością progu 0.1

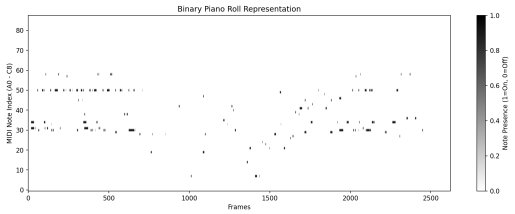


Figure 10. Zbinaryzowana macierz z wartością progu 0.9

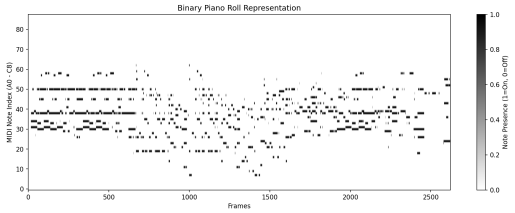


Figure 11. Zbinaryzowana macierz z wartością progu 0.4

6.2. Tworzenie plików MIDI

Z osiągnięta binarną macierzą ostatnim krokiem było przekonwertowanie jej na plik MIDI. Do tego celu skorzystano znowu z pakietu *pretty-midi*. Niestety nie znaleziono autmoatycznej metody, która by się tym zajęła i rozwiązanie jakie rozwinięto było wystarczające do tego, żeby móc wgrać plik do stacji DAW (ang. *Digital Audio Workstation*) i je obrobić ale konwersja nie przechodziła idealnie. Głównymi przeszkodami był fakt, że powstałe pliki MIDI nie posiadały informacji o nutach grające w tym samym momencie - zatem powstawały utwory gdzie w każdym momencie był tylko jeden dźwięk co powodowało nieprzyjemne wrażenie słuchowe i była potrzeba manualnego połączenia wszystkich nut, które powinne być jednym dźwiękiem. Poniżej przedstawiono taki przypadek:

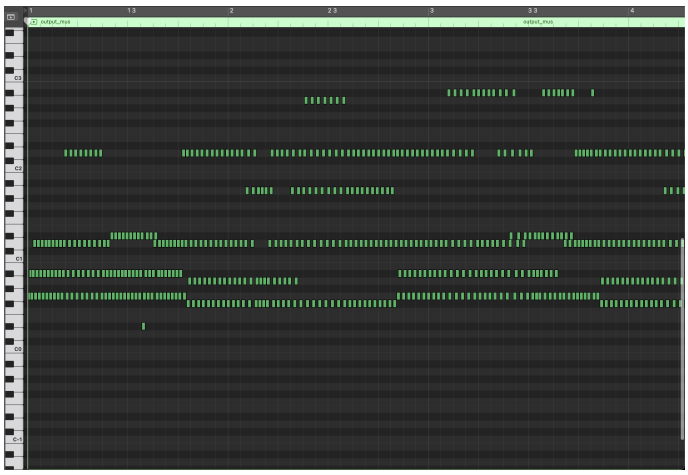


Figure 12. Niepoprawna konwersja z binarnej macierzy do pliku MIDI

7. Perspektywy rozwoju

Wyniki projektu uznano za zadowalające. Udało się sworzyć program realizujący założone cele. Program jednak nie ma wystarczającej jakości, aby nadawać się do celów komercyjnych. Na początku spodziewano się nieco lepszych wyników predykcji zbliżonych do tych osiągniętych przez autorów referencyjnego artykułu. Prawdopodobnie rozbeżność wyników spowodowana była przez różnicę w uczeniu sieci. Autorzy w wraz z postępem w uczeniu sieci zmniejszali parametr `learning_rate` co pozwoliło przełamać zastój w uczeniu i osiągnąć wyższe wyniki. W naszym przypadku sieć uczono cały czas ze stałą wartością `learning_rate`.

Dodatkowym aspektem projektu okazało się być porównanie między różnymi modelami. Nie było to zawarte w początkowych celach jednak zostało uznane za ciekawe i warte zbadania.

Przy większej ilości czasu i rozwoju projektu zdecydowano by się na próbę osiągnięcia wyższej jakości predykcji poprzez technikę dotyczącą `learning_rate` opisaną powyżej. Podjęto by próbę poprawienia fragmentu dotyczącego tworzenia plików midi oraz rozszerzono by program o transkrypcję do notacji nutowej. Kolejnym elementem mogłoby być rozszerzenie projektu o transkrypcję z innych instrumentów niż pianino.

8. Linki

Link do uczelnianego (PWr Gitlab) repozytorium z rozwiniętym kodem projektu.

References

[1] K. S. Michael Bereket, “An ai approach to automatic natural music transcription,” 2017.

[2] *Librosa - audio and music processing in Python*. [Online]. Available: https://librosa.org.

[3] *Piano key frequencies*. [Online]. Available: https://en.wikipedia.org/wiki/Piano_key_frequencies.

[4] *pickle — Python object serialization*. [Online]. Available: https://docs.python.org/3/library/pickle.html.

[5] $pretty_midi0.2.10documentation$. [Online]. Available: https://craffel.github.io/pretty-midi/.

[6] $train_test_split$. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.