POLITECHNIKA GDAŃSKA

Imię i nazwisko autora rozprawy: Maciej Blaszke
Dyscyplina naukowa: Informatyka Techniczna i Telekomunikacja

ROZPRAWA DOKTORSKA

Tytuł rozprawy w języku polskim: Identyfikacja instrumentu muzycznego z nagrania fonicznego za pomocą sztucznych sieci neuronowych

Tytuł rozprawy w języku angielskim: Identification of a musical instrument from an audio recording using artificial neural networks

Promotor *podpis*	Drugi promotor *podpis*
prof. dr hab. inż. Bożena Kostek	<Tytuł, stopień, imię i nazwisko>
Promotor pomocniczy *podpis*	Kopromotor *podpis*
<Tytuł, imię i nazwisko>	<Tytuł, stopień, imię i nazwisko>

Gdańsk, rok 2023

ROZPRAWA DOKTORSKA

Identyfikacja instrumentu muzycznego z nagrania fonicznego za pomocą sztucznych sieci neuronowych

mgr inż. Maciej Blaszke

Promotor: prof. dr hab. inż. Bożena Kostek

Gdańsk, 2023

POLITECHNIKA GDAŃSKA

Autor rozprawy doktorskiej: Maciej Blaszke

Ja, niżej podpisany(a), oświadczam, iż jestem świadomy(a), że zgodnie z przepisem art. 27 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2021 poz. 1062), uczelnia może korzystać z mojej rozprawy doktorskiej zatytułowanej:
Identyfikacja instrumentu muzycznego z nagrania fonicznego za pomocą sztucznych sieci neuronowych
do prowadzenia badań naukowych lub w celach dydaktycznych.[1]

Świadomy(a) odpowiedzialności karnej z tytułu naruszenia przepisów ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych i konsekwencji dyscyplinarnych określonych w ustawie Prawo o szkolnictwie wyższym i nauce (Dz.U.2021.478 t.j.), a także odpowiedzialności cywilno-prawnej oświadczam, że przedkładana rozprawa doktorska została napisana przeze mnie samodzielnie.

Oświadczam, że treść rozprawy opracowana została na podstawie wyników badań prowadzonych pod kierunkiem i w ścisłej współpracy z promotorką prof. dr hab. inż. Bożena Kostek

Niniejsza rozprawa doktorska nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadaniem stopnia doktora.

Wszystkie informacje umieszczone w ww. rozprawie uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami, zgodnie z przepisem art. 34 ustawy o prawie autorskim i prawach pokrewnych.

Potwierdzam zgodność niniejszej wersji pracy doktorskiej z załączoną wersją elektroniczną.

Gdańsk, dnia 01.09.2023 r. ..... Blaszke .....
podpis doktoranta

Ja, niżej podpisany(a), wyrażam zgodę/nie wyrażam zgody* na umieszczenie ww. rozprawy doktorskiej w wersji elektronicznej w otwartym, cyfrowym repozytorium instytucjonalnym Politechniki Gdańskiej.

Gdańsk, dnia 01.09.2023 r. ..... Blaszke .....
podpis doktoranta

**niepotrzebne usunąć*

[1] Art. 27. 1. Instytucje oświatowe oraz podmioty, o których mowa w art. 7 ust. 1 pkt 1, 2 i 4–8 ustawy z dnia 20 lipca 2018 r. – Prawo o szkolnictwie wyższym i nauce, mogą na potrzeby zilustrowania treści przekazywanych w celach dydaktycznych lub w celu prowadzenia działalności naukowej korzystać z rozpowszechnionych utworów w oryginale i w tłumaczeniu oraz zwielokrotniać w tym celu rozpowszechnione drobne utwory lub fragmenty większych utworów.
2. W przypadku publicznego udostępniania utworów w taki sposób, aby każdy mógł mieć do nich dostęp w miejscu i czasie przez siebie wybranym korzystanie, o którym mowa w ust. 1, jest dozwolone wyłącznie dla ograniczonego kręgu osób uczących się, nauczających lub prowadzących badania naukowe, zidentyfikowanych przez podmioty wymienione w ust. 1.

POLITECHNIKA GDAŃSKA

OPIS ROZPRAWY DOKTORSKIEJ

Autor rozprawy doktorskiej: Maciej Blaszke

Tytuł rozprawy doktorskiej w języku polskim: Identyfikacja instrumentu muzycznego z nagrania fonicznego za pomocą sztucznych sieci neuronowych

Tytuł rozprawy w języku angielskim: Identification of a musical instrument from an audio recording using artificial neural networks

Język rozprawy doktorskiej: polski

Promotor rozprawy doktorskiej: prof. dr hab. inż. Bożena Kostek

Data obrony:

Słowa kluczowe rozprawy doktorskiej w języku polskim: Identyfikacja instrumentów muzycznych, sztuczne sieci neuronowe, głębokie uczenie, impulsowe sieci neuronowe

Słowa kluczowe rozprawy doktorskiej w języku angielskim: Musical Instrument Identification, Artificial Neural Network, Deep learning, Spiking Neural Network

Streszczenie rozprawy w języku polskim:

Celem rozprawy jest zbadanie algorytmów do identyfikacji instrumentów występujących w sygnale polifonicznym. W części teoretycznej przywołano podstawy przetwarzania sygnałów fonicznych w kontekście ekstrakcji parametrów sygnałów. Dokonano analizy rozwoju metod uczenia maszynowego z uwzględnieniem podziału na sieci neuronowe pierwszej, drugiej i trzeciej generacji. Opisano powszechnie stosowane metody archiwizowania wyników oraz artefaktów treningu sztucznych sieci neuronowych. Na podstawie przeglądu literatury zaproponowano tezy rozprawy. W części eksperymentalnej opisano zgromadzone zbiory nagrań instrumentów muzycznych oraz sposób ich przekształcenia do formy zbioru treningowego, walidacyjnego i ewaluacyjnego. Przeprowadzono eksperymenty sprawdzające zasadność zaproponowanej koncepcji na mniejszym zbiorze danych. Zaimplementowano modele sieci neuronowych odpowiadające stanowi wiedzy oraz autorskie architektury. Bazując na wstępnych wynikach przeprowadzono eksperymenty na pełnym zbiorze danych. W badaniach tych zostały wykorzystane modele sieci neuronowych drugiej oraz trzeciej generacji. W końcowej części zawarto analizę uzyskanych wyników oraz omówiono wnioski z niej płynące, służące udowodnieniu tez badawczych. Przedstawiono również najważniejsze osiągnięcia rozprawy oraz kierunki rozwoju badań, mających na celu identyfikację instrumentów muzycznych w sygnale polifonicznym.

Streszczenie rozprawy w języku angielskim:

The purpose of the dissertation work is to propose algorithms for identifying instruments present in a polyphonic signal. The theoretical part recalls the basics of phonic signal processing in the context of signal parameter extraction. The development of machine learning methods is recalled, specifically focusing on first-, second-, and third-generation neural networks. The methods commonly used for archiving the results of training artificial neural networks are described. Based on the literature review, the theses of the dissertation are formulated. In the experimental part the available musical instrument recording datasets are described. Also, the way how to transform the datasets into training, validation, and evaluation sets is shown. First, experiments are conducted to

test the validity of the proposed concept on a smaller dataset. Models corresponding to state-of-the-art and novel architectures are implemented. Extended experiments are proposed and performed on a full training dataset. The second and third generation neural network models are implemented. The final section includes an analysis of the results obtained and discusses the conclusions drawn to prove the research theses. The most important achievements of this work and the directions of research development aimed at identifying musical instruments in polyphonic signals are also contained.

PODZIĘKOWANIA

Chciałbym podziękować mojej promotorce Pani Profesor Bożenie Kostek za poprowadzenie mojej naukowej pracy w kierunku, którego owocem jest niniejsza rozprawa.

Dziękuję Mamie, dzięki której pracy mogłem dotrzeć tu gdzie jestem.

Dziękuję Żonie, która wykazała się ogromem cierpliwości, gdy zajmowałem się pracą.

Dziękuję rodzinie i przyjaciołom za każde słowo i gest wsparcia, ogrom motywacji i wiarę w osiągnięcie wymarzonego celu. Nie sposób wymienić wszystkich, ale każdy wie do kogo skierowane są te podziękowania.

STRESZCZENIE

Celem rozprawy jest zbadanie algorytmów do identyfikacji instrumentów występujących w sygnale polifonicznym z wykorzystaniem sztucznych sieci neuronowych. W części teoretycznej przywołano podstawy przetwarzania sygnałów fonicznych w kontekście ekstrakcji parametrów sygnałów wykorzystywanych w treningu sieci neuronowych. Dodatkowo dokonano analizy rozwoju metod uczenia maszynowego z uwzględnieniem podziału na sieci neuronowe pierwszej, drugiej i trzeciej generacji. Opisano także powszechnie stosowane metody archiwizowania wyników treningu sztucznych sieci neuronowych oraz wystąpień artefaktów. Na podstawie przeglądu literatury zaproponowano tezy rozprawy. W części eksperymentalnej w pierwszej kolejności opisano dostępne i zgromadzone zbiory nagrań instrumentów muzycznych oraz sposób ich przekształcenia do formy zbioru treningowego, walidacyjnego i ewaluacyjnego. Przeprowadzono eksperymenty sprawdzające zasadność zaproponowanej koncepcji na mniejszym zbiorze danych. Na tym etapie zaimplementowano modele sieci neuronowych odpowiadające stanowi wiedzy w kontekście identyfikacji instrumentów muzycznych w celu porównania wyników z zaproponowanym modelami. Bazując na wstępnych wynikach, zaproponowano i przeprowadzono rozszerzone eksperymenty na zbiorze danych treningowych zawierających 13 instrumentów muzycznych, w tym rzadko występujących w typowym instrumentarium oraz w bazach muzycznych. W badaniach tych zostały wykorzystane modele sieci neuronowych drugiej oraz trzeciej (sieci impulsowe) generacji, w tym własne propozycje modeli. W końcowej części zawarto analizę uzyskanych wyników oraz omówiono wnioski z niej płynące, służące udowodnieniu tez badawczych. Przedstawiono również najważniejsze osiągnięcia rozprawy oraz kierunki rozwoju badań, mających na celu identyfikację instrumentów muzycznych w sygnale polifonicznym.

Słowa kluczowe: Identyfikacja instrumentów muzycznych, sztuczne sieci neuronowe, głębokie uczenie, impulsowe sieci neuronowe

DYSCYPLINA: Informatyka techniczna i telekomunikacja

DYSCYPLINA DODATKOWA:

ABSTRACT

The purpose of the dissertation work is to propose algorithms for identifying instruments present in a polyphonic signal using artificial neural networks. The theoretical part recalls the basics of phonic signal processing in the context of signal parameter extraction used in training neural networks. In addition, the development of machine learning methods is recalled, specifically focusing on first-, second-, and third-generation neural networks. The methods commonly used for archiving the results of training artificial neural networks and the occurrence of artifacts in the process are also described. Based on the literature review, the theses of the dissertation are formulated. In the experimental part, the first part describes the available musical instrument recording datasets. Also, the way how to transform the datasets into the form of 2D signal representation for training, validation, and evaluation is shown. First, experiments are conducted to test the validity of the proposed concept on a smaller dataset. At this stage, neural networks corresponding to state-of-the-art in the context of musical instrument identification are implemented to compare the results with the proposed models. Based on the preliminary results, extended experiments are proposed and performed on a training dataset containing 13 musical instruments, including those rarely found in typical instrumentation and music databases. At this research stage, the second and third (spiking neural networks) generation neural network models are implemented, including the novel model proposals. The final section includes an analysis of the results obtained and discusses the conclusions drawn to prove the research theses. The most important achievements of this dissertation work and the directions of research development aimed at identifying musical instruments in polyphonic signals are also contained in the final section.

Keywords: Musical Instrument Identification, Artificial Neural Network, Deep learning, Spiking Neural Network

DISCIPLINE: **ADDITIONAL DISCIPLINE:**

Information and communication technology

WYKAZ WAŻNIEJSZYCH SKRÓTÓW I OZNACZEŃ

AMT – Automatyczna transkrypcja muzyczna (ang. Automatic Music Transcription)

ANN – Sztuczne sieci neuronowe (ang. Artificial Neural Networks)

AUC ROC – Wielkość pola powierzchni pod krzywą (ang. Area Under Curve) ROC (ang. Receiver Operating Characteristic Curve)

CNN – Splotowe sieci neuronowe (ang. Convolutional neural networks)

CQT – Transformacja stałego Q (ang. Constant-Q transform)

CRNN – Splotowo-rekurencyjne sieci neuronowe (ang. Convolutional Recurrent Neural Networks)

DBN – Probabilistyczne modele generatywne (Deep Belief Networks; sieci głębokiego przekonania)

FFT – Szybka transformacja Fouriera (ang. Fast Fourier Transform)

FV – Wektor cech (ang. Feature Vector)

GMM – Gaussowskie modele mieszane (ang. Gaussian Mixture Models)

HMM – Ukryte modele Markowa (ang. Hidden Marcov Models)

ICA – Analiza składowych niezależnych (ang. Independent Component Analysis)

IDFT – Odwrotna dyskretna transformacja Fouriera (ang. Inverse Discrete Fourier Transform)

k-NN – k-najbliższych sąsiadów (ang. k-Nearest Neighbor)

LRAP – Uśredniona precyzja rankingu etykiet (ang. Label ranking average precision)

LSTM Sieci rekurencyjne z warstwą LSTM (Long Short-Term Memory Networks, LSTMs; długa pamięć krótkoterminowa)

MFCC – Współczynniki mel-cepstralne (ang. Mel-Frequency Cepstral Coefficients)

MLPs – Perceptron wielowarstwowy (Multilayer Perceptrons, MLPs)

MIR – Automatyczne wyszukiwanie muzyki (ang. Music Information Retrieval)

NMF – Nieujemna Faktoryzacja Macierzy (ang. Nonnegative Matrix Factorization)

RBF – Sieci radialne (Radial Basis Function Networks, RBFNs)

RNN – Rekurencyjne sieci neuronowe (ang. Recurrent Neural Netoworks)

SNN – Impulsowe sieci neuronowe (ang. Spiking Neural Networks)

SOM – Sieci samoorganizujące (Self-Organizing Maps, SOMs; sieci Kohonena)

SPR – Statystyczne rozpoznawanie wzorców (ang. Statistical Pattern Recognition)

SVM – Maszyna wektorów nośnych (ang. Support Vector Machine)

VQT – Transformacja zmiennego Q (ang. Variable-Q transform)

WLPC – Zmodyfikowane liniowe kodowanie predykcyjne (ang. Warped linear predictive coding)

Zero-padding – wypełnianie/uzupełnianie zerami ciągu próbek (ang. *zero-padding*)

SPIS RYSUNKÓW

Rysunek 1.1 Schemat blokowy podstawowej struktury głębokiej sieci neuronowej realizującej zadanie klasyfikacji bądź identyfikacji 19
Rysunek 1.2 Układ rozdziałów pracy 25
Rysunek 2.1 Przykład działania przetwornika analogowo-cyfrowego 26
Rysunek 2.2 Przykład reprezentacji różnych długości cyfrowego sygnału fonicznego .. 27
Rysunek 2.3 Przykład działania kwantyzatora 28
Rysunek 2.4 Porównanie widma sygnału analogowego i cyfrowego przy zadanej częstotliwości próbkowania 29
Rysunek 2.5 Przykład przebiegu sygnału cyfrowego i odpowiadającemu mu widma ... 30
Rysunek 2.6 Przykład działania FFT z zastosowanym zero-paddingiem 32
Rysunek 2.7 Przykład wyznaczania widma sygnału z zastosowaniem okna Blackmana oraz bez okna 33
Rysunek 2.8 Przykład spektrogramu dla rzeczywistego sygnału fonicznego 34
Rysunek 2.9 Przykład wykresu wodospadowego 35
Rysunek 2.10 Funkcja pokazująca zależność pomiędzy postrzeganą wysokością dźwięku w melach a rzeczywistą częstotliwością bodźca [99][137] 38
Rysunek 2.11 Zobrazowane przykłady dla wyliczonych przy pomocy pakietu Librosa reprezentacji sygnału: 1. spektrogram, 2. spektrogram w skali melowej, 3. chromagram, 4. reprezentacja Tonnetz 44
Rysunek 3.1 Szkic biologicznego neuronu 46
Rysunek 3.2 Przykłady różnych funkcji aktywacji używanych w neuronie 48
Rysunek 3.3 Schemat działania głębokiej sieci neuronowej [142] 49
Rysunek 3.4 Równoczesne zapisy (4 sekundy) czasów aktywacji 30 neuronów z kory czołowej małpy, wykonane przez Krügera i Aiple'a (1988). Każda aktywacja jest oznaczona krótkim pionowym paskiem, z osobnym rzędem dla każdego neuronu. Dla porównania zaznaczono długość interwału 100 milisekund dwoma pionowymi liniami [62] 53
Rysunek 3.5 Typowy kształt funkcji aktywacji naturalnego neuronu (EPSP i IPSP) [62] 55
Rysunek 3.6 Typowy kształt funkcji progowej dla neuronu biologicznego 56
Rysunek 4.1 Sekcja opisu eksperymentu w MLflow 63
Rysunek 4.2 Wizualizacja funkcji kosztu na zbiorze treningowym i ewaluacyjnym w MLflow 66
Rysunek 4.3 Krzywa ROC i pole pod krzywą 67
Rysunek 5.1 Rozkład dźwięków w notacji MIDI na klawiaturze fortepianowej 70
Rysunek 5.2 Zdjęcie przedstawiające rav vast [162] 77
Rysunek 5.3 Zdjęcie przedstawiające santur wraz z pałeczkami używanymi do gry na instrumencie [164] 78
Rysunek 5.4. Zdjęcie przedstawiające harfę [166] 79

Rysunek 5.5 Schemat działania systemu do przygotowywania danych treningowych dla modeli sieci neuronowych 83
Rysunek 5.6 Histogram instrumentów występujących w miksie zbioru treningowego .. 84
Rysunek 5.7 Rozkład liczby instrumentów muzycznych w miksie zbioru treningowego 84
Rysunek 5.8 Histogram instrumentów występujących w miksie zbioru testowego..... 85
Rysunek 5.9 Rozkład liczby instrumentów muzycznych w miksie zbioru testowego..... 85
Rysunek 5.10 Histogram instrumentów występujących w miksie zbioru ewaluacyjnego 86
Rysunek 5.11 Rozkład liczby instrumentów w miksie zbioru ewaluacyjnego 87
Rysunek 6.1 Przykładowe spektrogramy wybranych instrumentów i przygotowanego miksu instrumentów..... 88
Rysunek 6.2 Histogram instrumentów występujących w miksie 89
Rysunek 6.3 Liczba instrumentów w miksach..... 90
Rysunek 6.4 Architektura modelu 91
Rysunek 6.5 Miary uzyskane przez algorytm na zbiorze treningowym 92
Rysunek 6.6 Miary uzyskane przez algorytm na zbiorze walidacyjnym 92
Rysunek 6.7 Krzywe ROC dla każdego z testowanych instrumentów..... 93
Rysunek 6.8 Krzywe AUC ROC obliczone w fazie treningu 95
Rysunek 6.9 Krzywe AUC ROC obliczone w fazie walidacji..... 96
Rysunek 6.10 Krzywe ROC dla zmodyfikowanych instrumentów..... 97
Rysunek 6.11 Zredukowane mapy ciepła dla ostatnich warstw splotowych, dla każdego z identyfikowanych instrumentów..... 98
Rysunek 7.1 Architektura modelu CRNN zaadaptowanego na podstawie pracy Gururaniego i współautorów [38] 100
Rysunek 7.2 Wartości funkcji kosztu na zbiorach treningowym oraz walidacyjnym, jak również miary AUC ROC na zbiorze walidacyjnym 103
Rysunek 7.3 Macierz pomyłek dla modelu CRNN [%] 104
Rysunek 7.4 Architektura modelu CNN_1 zaadaptowanego na podstawie pracy Gururaniego i współautorów [38] 105
Rysunek 7.5 Wartości funkcji kosztu na zbiorze treningowym oraz walidacyjnym, jak również miary AUC ROC na zbiorze walidacyjnym 108
Rysunek 7.6 Macierz pomyłek dla modelu CNN_1 [%]..... 109
Rysunek 7.7 Architektura modelu CNN_2 zaadaptowanego na podstawie pracy Kratimenosa i współautorów [55] 110
Rysunek 7.8 Wartości funkcji kosztu na zbiorze treningowym oraz walidacyjnym, jak również miary AUC ROC na zbiorze walidacyjnym 113
Rysunek 7.9 Macierz pomyłek dla modelu CNN_2 [%]..... 114
Rysunek 8.1 Schemat blokowy ścieżki przetwarzania pojedynczego instrumentu w modelu o charakterze modularnym (FMCNN) 117
Rysunek 8.2 Architektura testowanego modelu modularnego..... 118

Rysunek 8.3 Wartości funkcji kosztu na zbiorze treningowym oraz walidacyjnym, jak również miary AUC ROC na zbiorze walidacyjnym 119
Rysunek 8.4 Macierz pomyłek dla modelu FMCNN [%] 120
Rysunek 8.5 Schemat blokowy modelu w architekturze SNN 122
Rysunek 8.6 Wartości funkcji kosztu na zbiorze treningowym oraz walidacyjnym, jak również miary AUC ROC na zbiorze walidacyjnym 123
Rysunek 8.7 Macierz pomyłek dla modelu SNN [%] 125

SPIS TABEL

Tabela 1.1 Przegląd metod realizujących zadania związane z klasyfikacją i identyfikacją sygnału muzycznego 20
Tabela 2.1 Opis deskryptorów standardu MPEG-7 [135] 36
Tabela 3.1 Porównanie głównych cech różniących TensorFlow, Pytorch i Keras 52
Tabela 4.1 Opis elementów sekcji opisu eksperymentu w MLflow 64
Tabela 4.2 Opis elementów sekcji parametrów eksperymentu w MLflow 64
Tabela 5.1 Liczba próbek instrumentów z zbioru GOOD SOUNDS wykorzystanych w ramach tworzenia zbioru treningowego i testowego na potrzeby eksperymentów 71
Tabela 5.2 Liczba próbek fortepianu z zbioru UMA Piano wykorzystanych w ramach tworzenia zbioru treningowego i testowego na potrzeby eksperymentów 72
Tabela 5.3 Liczba próbek instrumentów z zbioru Philharmonia Orchestra (all-samples) wykorzystanych w ramach tworzenia zbioru treningowego i testowego na potrzeby eksperymentów 72
Tabela 5.4 Liczba próbek instrumentów z zbioru Slakh wykorzystanych w ramach tworzenia zbioru treningowego i testowego na potrzeby eksperymentów 73
Tabela 5.5 Liczba próbek instrumentów z zbioru MedleyDB [12] wykorzystanych w ramach tworzenia zbioru treningowego i testowego na potrzeby eksperymentów 74
Tabela 5.6 Liczba próbek instrumentów z zbioru nagranego w ramach dyplomu magisterskiego [58] wykorzystanych w ramach tworzenia zbioru treningowego i testowego na potrzeby eksperymentów 75
Tabela 5.7 Liczba próbek instrumentów z zbioru nagranego w ramach dyplomu inżynierskiego [73] wykorzystanych w ramach tworzenia zbioru treningowego i testowego na potrzeby eksperymentów 76
Tabela 5.8 Liczba próbek instrumentów z zbioru nagranego w ramach dyplomu inżynierskiego wykorzystanych w ramach tworzenia zbioru treningowego i testowego na potrzeby eksperymentów 79
Tabela 5.9 Liczebność wybranych instrumentów w poszczególnych zbiorach danych . 80
Tabela 6.1 Wartości miar dla poszczególnych instrumentów 94
Tabela 6.2 Macierz pomyłek [%] 94
Tabela 6.3 Porównanie pomiędzy modelem bazowym i zmodyfikowanym 94
Tabela 6.4 Porównanie wyników pomiędzy modelem ujednoliconym i zmodyfikowanym 96
Tabela 6.5 Wyniki z rozróżnieniem na poszczególne instrumenty 97
Tabela 7.1 Podstawowe parametry treningu modelu CRNN 102
Tabela 7.2 Wyniki ewaluacji wytrenowanego modelu CRNN 103
Tabela 7.3 Podstawowe parametry treningu modelu CNN_1 107
Tabela 7.4 Wyniki ewaluacji wytrenowanego modelu CNN_1 108
Tabela 7.5 Podstawowe parametry treningu modelu CNN_2 112

Tabela 7.6 Wyniki ewaluacji wytrenowanego modelu CNN_2 113
Tabela 8.1 Opis bloku dla pojedynczego instrumentu w modelu rozproszonym 115
Tabela 8.2 Podstawowe parametry treningu modelu FMCNN 118
Tabela 8.3 Wyniki ewaluacji wytrenowanego modelu rozproszonego 119
Tabela 8.4 Opis bloku dla pojedynczego instrumentu w modelu impulsowym 121
Tabela 8.5 Podstawowe parametry treningu modelu impulsowego 123
Tabela 8.6 Wyniki ewaluacji wytrenowanego modelu impulsowego 124
Tabela 9.1 Porównanie parametrów modeli state-of-the-art z badanymi i zaproponowanymi architekturami 126
Tabela 9.2 Porównanie macierzy pomyłek dla modeli state-of-the-art z badanymi i zaproponowanymi architekturami 127
Tabela 9.3 Porównanie wynikowych miar dla modeli state-of-the-art z badanymi i zaproponowanymi architekturami 127

SPIS TREŚCI

2

WYKAZ WAŻNIEJSZYCH SKRÓTÓW I OZNACZEŃ 6

SPIS RYSUNKÓW 8

SPIS TABEL 11

1. Wprowadzenie 17

1.1. Geneza pracy 17

1.2. Cel pracy 22

1.3. Tezy rozprawy doktorskiej 22

1.4. Zawartość rozprawy 23

2. Podstawy przetwarzania sygnału fonicznego w kontekście zastosowania w uczeniu maszynowym 26

2.1. Sygnał cyfrowe i metody konwersji sygnału analogowego 26

2.2. Kwantyzacja sygnału analogowego 28

2.3. Zjawisko aliasingu i częstotliwość Nyquista 28

2.4. Dyskretna Transformacja Fouriera 29

2.5. Odwrotna Dyskretna Transformacja Fouriera 31

2.6. Rozdzielczość częstotliwościowa: wypełnianie zerami i funkcje okna 31

2.7. Reprezentacja czasowo-częstotliwościowa sygnału 33

2.8. Parametryzacja sygnałów fonicznych 35

2.8.1. Deskryptory standardu MPEG-7 35

2.8.2. Reprezentacja 2D sygnałów muzycznych 37

2.8.3. Pakiet Librosa 41

3. Rozwój metod uczenia maszynowego − Sztuczne sieci neuronowe 45

3.1. Biologiczne Sieci Neuronowe 45

3.2. Sieci neuronowe pierwszej generacji 46

3.2.1. Krótki rys historyczny 47

3.2.2. Matematyczny model neuronu 47

3.3. Sieci neuronowe 2. generacji − sieci głębokie 49

3.3.1. Zasada działania modeli głębokich 49

3.3.2. Przykładowe narzędzia wykorzystywane w treningu głębokich sieci neuronowych 50

3.4. Sieci neuronowe 3. generacji − SNN (Impulsowe Sieci Neuronowe) 52

3.4.1. Geneza impulsowych sieci neuronowych 52

3.4.2. Model neuronu impulsowego w oparciu o model neuronu biologicznego 54

3.4.3. Przykładowe narzędzia wykorzystywane w treningu impulsowych sieci neuronowych 56

4. Metodologia archiwizowania i porównywania wyników eksperymentów 59

4.1. Komponenty oferowane przez platformę MLFlow 59

4.2. Sposoby zapisywania plików artefaktów w MLFlow 61

4.3. Metody skalowania w MLFlow 61

4.4. Przykłady zastosowania MLFlow 62

4.5. Opis implementacji logowania artefaktów i parametrów 63

4.5.1. Rejestrowany opis treningu 63

4.5.2. Rejestrowane parametry 64

4.5.1. Rejestrowane metryki 65

4.6. Wybrane miary oceny jakości klasyfikacji 66

4.6.1. Precyzja 66

4.6.1. Czułość 66

4.6.2. Miara F1 67

4.6.3. AUC ROC 67

4.6.4. LRAP 67

4.6.5. Dokładność 68

5. Zbiory uczące i testowe dla badanych algorytmów 69

5.1. Interpretacja zapisu nutowego w formacie MIDI 69

5.2. Rzeczywiste nagrania instrumentów 71

5.2.1. Baza nagrań „GOOD SOUNDS” 71

5.2.2. Baza nagrań UMAPiano 71

5.2.3. Baza nagrań Philharmonia Orchestra 72

5.2.4. Baza nagrań SLAKH 73

5.2.5. Baza nagrań MedleyDB 74

5.2.6. Nagrania zrealizowane w Katedrze Systemów Multimedialnych 75
5.3. Założenia struktury zbioru treningowego 80
5.4. Sposób przygotowania danych treningowych 81
5.5. Histogramy opisujące zbiory danych 83
5.5.1. Zbiór treningowy 83
5.5.2. Zbiór testowy 84
5.5.3. Zbiór ewaluacyjny 86
6. Eksperyment sprawdzający 88
6.1. Zbiór treningowy i ewaluacyjny 88
6.2. Architektura modelu 90
6.3. Trening modelu 91
6.4. Wyniki oceny i dyskusja 92
6.5. Redefinicja architektury modelu 94
6.6. Dyskusja uzyskanych wyników 98
7. Zadania związane z klasyfikacją i identyfikacją instrumentów w sygnale fonicznyczym 99
7.1. Implementacja modelu w architekturze CRNN 99
7.1.1. Opis architektury modelu CRNN 99
7.1.2. Wynik działania algorytmu CRNN 102
7.1. Implementacja modelu w architekturze CNN_1 104
7.1.1. Opis architektury modelu CNN_1 104
7.1.2. Wynik działania algorytmu CNN_1 107
7.2. Implementacja modelu w architekturze CNN_2 109
7.2.1. Opis architektury modelu CNN_2 109
7.2.2. Omówienie wyników działania algorytmu CNN_2 112
8. Identyfikacja instrumentu muzycznego 115
8.1. Scenariusz przeprowadzonych eksperymentów 115
8.2. Model rozproszony 115
8.2.1. Opis architektury identyfikatora 115
8.2.2. Trening identyfikatora 118
8.2.3. Ewaluacja identyfikatora 119

8.3. Identyfikator impulsowy 120
8.3.1. Opis architektury identyfikatora instrumentów muzycznych 121
8.3.2. Trening impulsowego identyfikatora instrumentów muzycznych 123
8.3.3. Ewaluacja impulsowego identyfikatora instrumentów muzycznych 124
9. Analiza wyników 126
10. Wnioski i podsumowanie 128
10.1. Wnioski dotyczące trudności w realizacji rozprawy 129
10.2. Osiągnięcia rozprawy doktorskiej 129
10.3. Dalsze kierunki prac 130
11. WYKAZ LITERATURY 131
12. Dodatek A: SKRYPTY przygotowujące rysunki z rozdziału 2 143

1. WPROWADZENIE

1.1. Geneza pracy

Identyfikacja złożonych sygnałów dźwiękowych, w tym sygnałów muzycznych, jest złożonym procesem pomimo dużych postępów w tym obszarze badawczym w ostatniej dekadzie [4][26][39][55][81]. Wynika to z wysokiej entropii informacji zawartej w sygnałach dźwiękowych, dużej różnorodności źródeł, sposobu miksowania nagrań, trudności analitycznego opisu, szczególnie w przypadku sygnałów złożonych z wielu źródeł [12][14][118], itd. Stąd też w literaturze widać wiele kierunków i algorytmów służących do separacji i identyfikacji dźwięków z materiału muzycznego [3][8][21][41][61]. Wykorzystują one głównie analizę spektralną i cepstralną, pozwalającą na wykrycie częstotliwości podstawowej i jej harmonicznych oraz przypisanie odzyskanych wzorców do konkretnego instrumentu [41]. Wiąże się to jednak z ograniczeniami − kosztem zwiększenia rozdzielczości czasowej zmniejsza się rozdzielczość częstotliwościowa i odwrotnie. Ponadto należy zauważyć, że algorytmy te nie zawsze pozwalają na ekstrakcję instrumentów perkusyjnych i innych sygnałów nieharmonicznych, co może stanowić źródło zakłóceń dla algorytmu, utrudniających jego działanie oraz zmniejszających dokładność i wiarygodność wyników.

Kolejny problem w procesie identyfikacji stanowi artykulacja muzyczna, np. glissando (płynne przejście pomiędzy następującymi po sobie dźwiękami) czy tremolo (szybka modulacja amplitudy dźwięku) powodują przesunięcia częstotliwości w widmie, jak również stany przejściowe − transjenty mogą generować dodatkowe składowe w widmie sygnału lub wzmacniać niektóre składowe, jak to się dzieje w przypadku przedęć oktawowych. Innym ważnym czynnikiem jest współbrzmienie dźwięków, bowiem muzyka w kulturze zachodniej wspiera się w dużym stopniu na kadencji rozwiniętej z dysonansowo-konsonansowego formy [119]. Oczywistą konsekwencją układu współbrzmień jest nakładanie się tonów harmonicznych w widmie, co stanowi problem dla większości algorytmów.

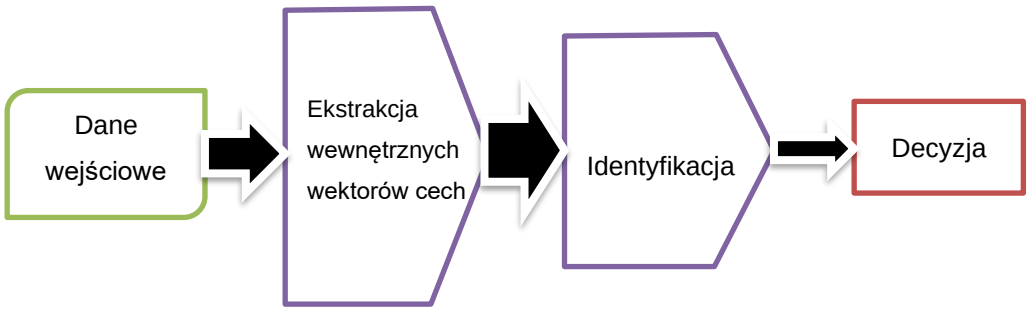
Należy pamiętać, że nagrywanie instrumentów muzycznych jest uwarunkowane zarówno akustyką wnętrza, jak również mikrofonem czy techniką mikrofonową. Ogromne znaczenie ma to, w jaki sposób dany instrument jest rejestrowany. Istnieją oczywiście ogólne zalecenia dotyczące rejestracji danego instrumentu muzycznego, szczególnie gdy dotyczą warunków studyjnych. Nagrania muzyki w studiach o specjalnie do tego celu zaprojektowanej akustyce lub rejestracji muzyki podczas koncertu zwykle realizowane są przez doświadczonych inżynierów dźwięku. W takim przypadku identyfikacja dźwięku instrumentu muzycznego w nagraniu jest stosunkowo łatwa zarówno dla ludzkiego ucha, jak procesu automatycznego rozpoznawania instrumentu muzycznego. Jednak w ciągu kilku ostatnich dekad nagrywanie instrumentów muzycznych i ich przetwarzanie uległo zmianie. Obecnie muzyka jest nagrywana wszędzie i za pomocą dowolnych dostępnych rejestratorów, w tym smartfonów. Z kolei zadanie automatycznej identyfikacji stało się o wiele bardziej pożądane w różnych zastosowaniach. Wynika to z faktu, że identyfikacja instrumentów muzycznych ma znaczenie w wielu obszarach, które nie są ściśle

związane z muzyką, np. w automatycznym tworzeniu dźwięku do gier, organizacji muzycznych serwisów społecznościowych, separacji miksów muzycznych na pojedyncze ścieżki, amatorskich nagraniach, itp. Co więcej, pojawiła się interesująca koncepcja, która opiera się na założeniu, że każda wibrująca część instrumentu może być użyta do pomiaru jego właściwości fizycznych. Nasuwa się więc pytanie czy dźwięk instrumentu muzycznego może być wykorzystany do wnioskowania o fizycznych właściwościach jego podsystemów [10]. Tworzenie nowych interfejsów dla ekspresji muzycznej (NIME, New Interfaces for Musical Expression) [120] jest kolejnym paradygmatem związanym z kreowaniem dźwięku oraz nowym sposobem ekspresji dźwięku instrumentów muzycznych i ich wydajności [104]. Wreszcie, w kontekście tworzenia muzyki należy wspomnieć o inteligentnych instrumentach muzycznych, należących do klasy urządzeń IoT (Internet of Things) [108][109]. Turchet i in. opracowali silnik dźwiękowy wykorzystujący cyfrowe przetwarzanie sygnału, fuzję czujników i wbudowane techniki uczenia maszynowego do klasyfikacji pozycji, dynamiki i barwy każdego uderzenia inteligentnego cajonu [108].

W ogólności, zarówno instrumenty klasyczne, jak syntetyczne, wpisują się w potrzebę identyfikacji dźwięku i dalszych zastosowań, np. obliczeniowej analizy sceny dźwiękowej (CASA, Computational Auditory Scene Analysis) [17,89], interakcji człowiek-komputer (HCI, Human Computer Interface) [75], postprodukcji muzycznej [121][122], wyszukiwania informacji muzycznych [19][57][83][123][124], automatycznego miksowania muzyki [54][64][68], systemów rekomendacji muzycznych [24][94][98], itd.

Analizując dostępny stan wiedzy, zauważyć można jednak tendencję do skupiania się na identyfikacji instrumentów sygnale muzycznym poddanym w pierwszej kolejności separacji źródeł [41]. Drugim często spotykanym podejściem, związanym z identyfikacją jest rozpoznawanie dominującego instrumentu w danym fragmencie muzycznym [4,16,39,81]. Brakuje jednak prac, w których autorzy podchodzą do identyfikacji bardziej kompleksowo, poprzez rozpoznawanie wszystkich występujących instrumentów w danym utworze muzycznym. Związane jest to również z ograniczoną dostępnością predefiniowanych zbiorów treningowych odpowiadających temu zadaniu. Większość istniejących zbiorów, które można wykorzystać do treningu w uczeniu maszynowym, jak np. baza SLAKH (ang. Synthesized Lakh) [125] jest przygotowana do automatycznego miksowania bądź separacji źródeł, jednocześnie ograniczając się do czterech podstawowych grup instrumentów takich jak perkusja, bas, gitara i wokale bądź pozostałe instrumenty. W związku z tym problemem prace odnoszące się do bardziej dokładnej identyfikacji pracują na bardzo ograniczonych zbiorach instrumentów.

Aktualnie najpopularniejszą metodą w wszelkich zadaniach rozpoznawania różnych klas obiektów − zarówno w dziedzinie dźwięków, jak i obrazów − są głębokie sieci neuronowe [4][39][60][101]. Ich budowa w założeniu bazuje na schemacie, w którym autorzy poszczególnych prac wprowadzają swoje modyfikacje do istniejącej struktury podstawowej. Sieć jest podzielona na dwie części: ekstrahującą dodatkowe parametry i klasyfikującą, tak jak to zostało przedstawione na rysunku 1.1.



Rysunek 1.1 Schemat blokowy podstawowej struktury głębokiej sieci neuronowej realizującej zadanie klasyfikacji bądź identyfikacji

Zadaniem pierwszej części jest dodatkowe przeanalizowanie danych wyjściowych, aby wyznaczyć tzw. wewnętrzne wektory cech (ang. *High Level Features*), które mapują przygotowane przez użytkownika dane na takie, które są łatwiej interpretowalne przez kolejne elementy sieci głębokiej. Zazwyczaj część ta składa się z różnej liczby warstw splotowych z różną konfiguracją – zależną od wizji autora, połączonych ze sobą warstwami normalizacji oraz zmniejszania rozmiaru poprzez łączenie ze sobą sąsiadujących elementów przy pomocy określonej funkcji, np. maksimum czy średniej. Należy podkreślić, że dla wszystkich klas część ta jest wspólna, przez co ekstrahowane parametry muszą być jednocześnie bardzo szczegółowe i ogólne, aby każda z badanych klas była równie łatwo identyfikowana [28][58][59].

Druga część, bazując na przygotowanych danych, dokonuje właściwej identyfikacji bądź klasyfikacji. Zazwyczaj liczba wyjść odpowiada ilości badanych klas, a każde z nich interpretowane jest jako prawdopodobieństwo przynależności do konkretnej klasy. W zależności od podejścia autorów danej pracy oraz często typu wejścia najczęściej spotyka się warstwy w pełni połączone (ang. *Fully Connected*) [39][60] lub rekurencyjne typu długoterminowa pamięć krótkoterminowa (LSTM, Long-Short Term Memory) [4,26].

W ostatnich latach coraz częściej zaczynają się pojawiać bardziej innowacyjne podejścia, odchodzące od sieci głębokich, a powracające do korzeni sieci neuronowych, których podstawą była biologiczna komórka nerwowa [102,][126]. Odpowiedzią na te poszukiwania okazały się sieci impulsowe [35][62], które nie tylko odwzorowują mechanizm ważenia i sumowania wejść, jak w biologicznym neuronie, ale jednocześnie to, w jakim czasie i w jakim jego odstępie pojawiają się kolejne informacje. Podejście to pozwala na zbliżenie zachowania zaprojektowanej sieci do rzeczywiście funkcjonujących w naturze połączeń nerwowych.

Rozwój metod identyfikacji źródeł w sygnale fonicznym

Aby w pełni przedstawić genezę niniejszej rozprawy doktorskiej, należy odnieść się do rozwoju metod dotyczących klasyfikacji/identyfikacji źródeł (w tym instrumentów muzycznych) w sygnale fonicznym na przestrzeni ostatnich dekad. Identyfikacja instrumentów muzycznych odgrywa istotną rolę w różnych zadaniach klasyfikacyjnych w obszarze sygnałów fonicznych. Jednym z przykładów może być klasyfikacja gatunków muzycznych. W tym kontekście

stosowano wiele algorytmów, takich jak np. SVM (ang. Support Vector Machine), drzewa decyzyjne, zbiory przybliżone, sieci neuronowe (ANN, Artificial Neural Networks), a w ostatnim czasie również sieci splotowe (CNN, Convolutional Neural Networks) czy sieci rekurencyjne (RNN, Recurrent Neural Networks), uzyskując wyniki zawarte w przedziale (60%−96%) [20][76][90][91][110]. Należy zauważyć, że gatunek muzyczny jest warunkowany przez instrumenty występujące w utworze muzycznym. Na przykład wiolonczelę i saksofon można często spotkać w muzyce jazzowej, natomiast banjo jest ściśle związane z muzyką country. W tabeli 1.1 przedstawiono przegląd różnych algorytmów i zadań opisanych powyżej wraz z uzyskanymi wynikami (definicje miar i wskaźników przedstawionych w tabeli 1.1 zostały podane w rozdziale 4.6).

Tabela 1.1 Przegląd metod realizujących zadania związane z klasyfikacją i identyfikacją sygnału muzycznego

Autorzy	Rok	Zadanie	Typ danych wejściowych	Algorytm	Miary
L. Zhong, E. Cooper, J. Yamagishi, N. Minematsu [117]	2023	Rozpoznawanie instrumentu dominującego	Sygnał foniczny	CNN	LRAP – 0,814 F1 micro – 0,674 F1 macro – 0,584
D. Szeliga, P. Tarasiuk, B. Stasiak, P.S. Szczepaniak [101]	2022	Klasyfikacja pojedynczego instrumentu	FFT	CNN	Dokładność – 75,47%
Dewi C, Chen R [29]	2022	Rozpoznawanie instrumentu dominującego	MFCC i FV	Transformer	F1 micro – 0,66 F1 macro – 0,62
K. Avramidis, A. Kratimenos, C. Garoufis, A. Zlatintsi, P. Maragos [4]	2021	Rozpoznawanie instrumentu dominującego	Raw audio	RNN, CNN i CRNN	LRAP – 0,747 F1 micro – 0,608 F1 macro – 0,543
Kratimenos, A., Avramidis, K., Garoufis, C., Zlatintsi, A., & Maragos, P. [55]	2021	Identyfikacja instrumentu	CQT	CNN	LRAP – 0,805 F1 micro – 0,647 F1 macro – 0,546
Zhang F. [116]	2021	Identyfikacja gatunku	MIDI	RNN	Dokładność – 89,91% F1 macro – 0,9
Mandal P., Nath I., Gupta N., Jha M.K., Ganguly D.G., Pal S [65]	2020	Identyfikacja gatunku	MFCC	ANN	brak informacji
Shreevathsa P. K., Harshith M., A. R. M. Ashwini [96]	2020	Klasyfikacja pojedynczego instrumentu	MFCC	ANN i CNN	ANN Dokładność – 72,08% CNN Dokładność – 92.24%
M. Blaszke, D. Koszewski, S. Zaporowski [15]	2019	Klasyfikacja pojedynczego instrumentu	MFCC	CNN	Precision – 0,99 Recall – 1,0 F1 score – 0,99
Das O [27]	2019	Klasyfikacja pojedynczego instrumentu	MFCC and WLPC (Warped	Regresja Logiczna i SVM	Dokładność 100%

			linear predictive coding)		
S. Gururani, C. Summers, A. Lerch [38]	2018	Identyfikacja instrumentu	MFCC	CNN i CRNN	AUC ROC – 0,81
Rosner A., Kostek B. [90]	2018	Identyfikacja gatunku	FV	SVM	Dokładność – 72%
K. Choi, G. Fazekas, M. Sandler, K. Cho [26]	2017	Tagowanie audio	MFCC	CRNN	ROC AUC – 0,65 – 0,98
Y. Han, J. Kim, and K. Lee [39]	2017	Rozpoznawanie instrumentu dominującego	MFCC	CNN	F1 macro – 0,503 F1 micro – 0,602
J. Pons, O. Slizovskaia, R. Gong, E. Gómez, X. Serra [81]	2017	Rozpoznawanie instrumentu dominującego	MFCC	CNN	F1 micro – 0,503 F1 macro – 0,432
Bhojane S.B., Labhshetwar O.G., Anand, K., Gulhane S.R. [11]	2017	Klasyfikacja pojedynczego instrumentu	FV	k-NN	brak informacji
Lee J., Kim T., Park J., Nam J. [39]	2017	Rozpoznawanie instrumentu dominującego	Raw audio	CNN	AUC ROC – 0,91 Dokładność – 86% F1 – 0,45%
P. Li, J. Qian, T. Wang [60]	2015	Identyfikacja instrumentu	Raw audio, MFCC, and CQT	CNN	Dokładność – 82,74%
Giannoulis D., Benetos E., Klapuri A., Plumbley M. D. [36]	2014	Identyfikacja instrumentu	SDL	Missing feature approach with AMT	F1 – 0,31
Bosch, J. J., Janer, J., Fuhrmann, F., Herrera, P. [16]	2012	Rozpoznawanie instrumentu dominującego	Raw audio	SVM	F1 micro – 0,503 F1 macro – 0,432
T. Heittola, A. Klapuri, T. Virtanen [41]	2009	Rozpoznawanie instrumentu w sygnale polifonicznym	MFCC	NMF I GMM	F1 score – 0,62
T. Heittola, A. Klapuri, T. Virtanen [42]	2009	Identyfikacja instrumentu	MFCC	NMF i GMM	F1 – 0,59
S. Essid, G. Richard, B. David [32]	2006	Klasyfikacja pojedynczego instrumentu	MFCC and FV	GMM I SVM	Dokładność – 93%
B. Kostek [53]	2004	Klasyfikacja pojedynczego instrumentu	FV	ANN	Dokładność – 72,24%
A. Eronen [30]	2003	Klasyfikacja pojedynczego instrumentu	MFCC	ICA I HMM	
T. Kitahara, M. Goto, H. Okuno [48]	2003	Klasyfikacja pojedynczego instrumentu	FV	Funkcja dyskryminacyjna oparta na regule decyzyjnej Bayesa	Dokładność – 79,73%
Tzanetakis G., Cook P [110]	2002	Identyfikacja gatunku	FV and MFCC	SPR	Dokładność – 61%
B. Kostek, A. Czyżewski [52]	2001	Klasyfikacja pojedynczego instrumentu	FV	ANN	Dokładność – 94,5%
A. Eronen and A. Klapuri [31]	2000	Klasyfikacja pojedynczego instrumentu	FV		Dokładność – 80%

J. Marques, P. J. Moreno [67]	1999	Klasyfikacja pojedynczego instrumentu	MFCC	GMM i SVM	Błąd klasyfikacji – 17%

1.2. Cel pracy

Przegląd dostępnych w literaturze metod wskazuje na znaczne braki w kontekście identyfikacji instrumentów muzycznych. Pierwszym z nich jest brak ujednoliconego zbioru treningowego i ewaluacyjnego. Powoduje to, że algorytmów nie można łatwo porównać między sobą bezpośrednio w kontekście jakości. Widać również, że autorzy stosują zróżnicowany zestaw metryk, co dodatkowo utrudnia to zadanie. Ostatnim zidentyfikowanym przez autora problemem jest niska skuteczność identyfikacji instrumentów muzycznych w zestawieniu z algorytmami klasyfikującymi instrumenty bądź gatunki muzyczne.

Celem przedstawionej pracy jest opracowanie algorytmów, które pozwolą na skuteczną identyfikację 13 instrumentów występujących w sygnale polifonicznym. Innowacyjnym podejściem w projektowaniu zaproponowanego rozwiązania, jest podzielenie całej sieci na moduły, z których każdy ma za zadanie przeprowadzenie osobnej identyfikacji przypisanego do niej instrumentu. Ma to w szczególności zapewnić lepszą jakość identyfikacji. Narzędziem wybranym do przeprowadzenia tych eksperymentów są sztuczne sieci neuronowe, które dominują w literaturze wśród rozwiązań uzyskujących najwyższą skuteczność w dziedzinach identyfikacji i klasyfikacji.

Jednocześnie istotnym aspektem jest zbadanie skuteczności w kontekście zadań związanych z identyfikacją innowacyjnej technologii SNN (ang. *Spiking Neural Networks*), nazwanej *„sieciami neuronowymi trzeciej generacji”* [35]. Sieci neuronowe 3. generacji są określane w literaturze w języku polskim jako pulsujące [127] lub impulsowe [102]. W niniejszej pracy będzie stosowane nazwa: sieci impulsowe. Zgodnie z założeniami sieć tego typu pozwoli ona na osiągnięcie wysokich wyników przy jednoczesnym zmniejszeniu rozmiarów sieci neuronowej. Dodatkową zaletą jest możliwość zastosowania dedykowanych akceleratorów takich jak Intel Loihi [128]. Ponieważ miara LRAP (Label ranking average precision) odpowiada jakości identyfikacji w kontekście zadania z wieloma klasami występującymi w danej próbce, dlatego została zastosowana jako kryterium oceny w procesie ewaluacji uzyskanych wyników.

1.3. Tezy rozprawy doktorskiej

W pracy zaproponowano następujące tezy badawcze:

1. **Możliwe jest przygotowanie i wytrenowanie takiej sieci neuronowej, która zrealizuje identyfikację instrumentów występujących w sygnale fonicznym, uzyskując wynik miary opisującej jakość identyfikacji w kontekście zadania z wieloma klasami występującymi w danej próbce (ang. *Label ranking average precision*, LRAP) wyższy niż 0,85, co przewyższa aktualny stan wiedzy.**

2. **Zastosowanie w zadaniu identyfikacji instrumentów sieci neuronowej składającej się z mniejszych podsieci wyspecjalizowanych w odniesieniu do badanego instrumentu pozwala zwiększyć skuteczność identyfikacji definiowaną miarą LRAP do poziomu 0,89.**

3. **Sieci neuronowe impulsowe 3. generacji (ang. *Spiking Neural Networks, SNN*) pozwalają na uzyskanie skuteczności identyfikacji instrumentu miary LRAP na poziomie 0,84 przy jednoczesnym dwukrotnym zmniejszeniu liczby parametrów sieci neuronowej.**

1.4. Zawartość rozprawy

Praca została podzielona na dziesięć rozdziałów, których struktura została przedstawiona na rysunku 1.2. W pierwszej kolejności podano genezę pracy z przeglądem rozwiązań w kontekście klasyfikacji/identyfikacji instrumentów muzycznych, cel oraz tezy rozprawy doktorskiej. W przeglądzie stanu wiedzy odniesiono się do tematyki zbliżonej do rozprawy, czyli klasyfikacji instrumentu, klasyfikacji gatunku oraz identyfikacji instrumentu wraz z opisem i wartościami miar uzyskanych przez autorów prac. Rozdział drugi obejmuje podstawy cyfrowego przetwarzania sygnałów, skupiając się przede wszystkim na aspektach przekształceń sygnału z analogowego na cyfrowy oraz z dziedziny czasu na dziedzinę częstotliwości i odwrotnie. Stanowią one podstawę nie tylko analizy sygnałów fonicznych, ale również parametryzacji sygnałów fonicznych. Oba te aspekty są bardzo istotne w kontekście uczenia maszynowego. Dalsza cześć tego rozdziału odnosi się do różnych metod parametryzacji sygnałów fonicznych, tj. przywołuje deskryptory standardu MPEG-7, reprezentacje widmowe dwuwymiarowe (2D), jak również podaje przykłady parametrów zaimplementowanych w bibliotece Librosa [129].

Trzeci rozdział skupia się na krótkim opisie rozwoju uczenia maszynowego, tj. sztucznych sieciach neuronowych (ANNs, Artificial Neural Networks), splotowych sieciach neuronowych (CNN, Convolutional Neural Networks) oraz sieci 3. generacji − impulsowych sieciach neuronowych (SNN, Spiking Neural Networks).

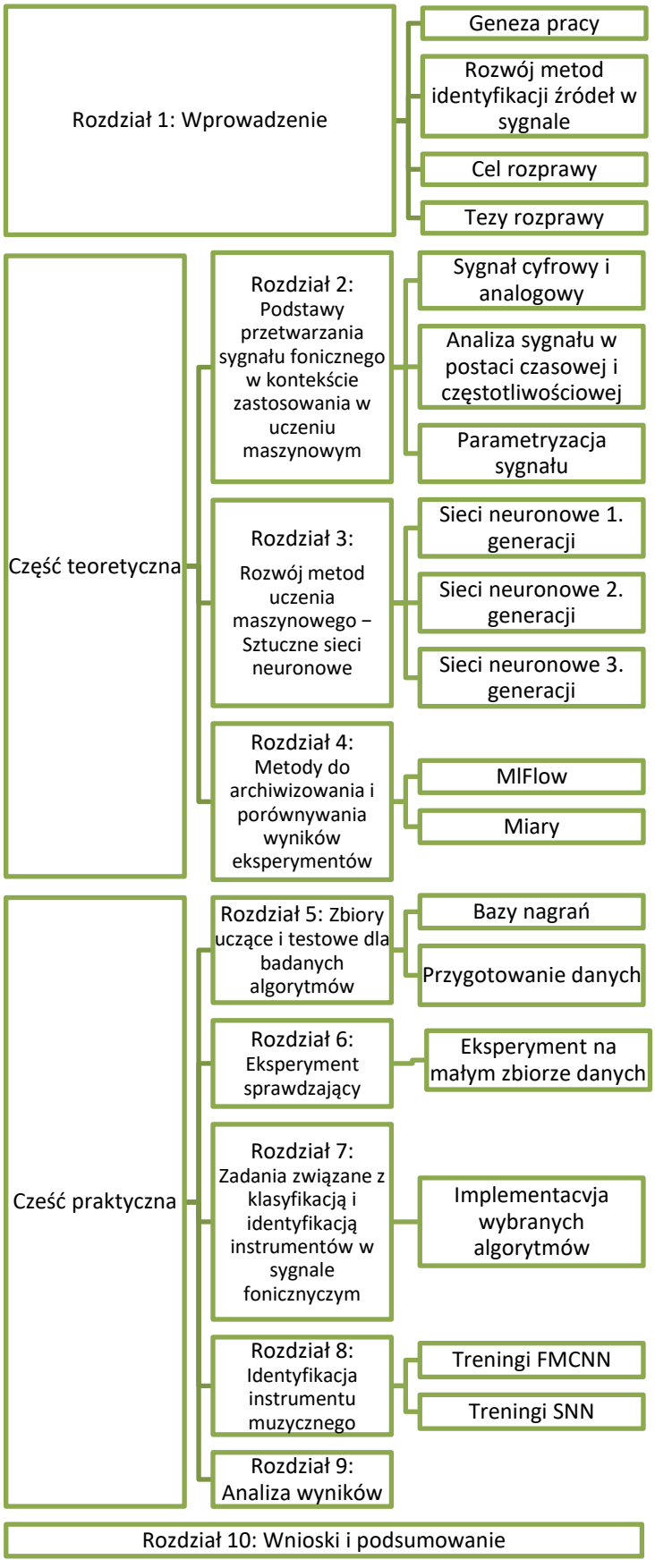
W rozdziale czwartym opisano metodologię archiwizowania i porównywania wyników eksperymentów przy pomocy narzędzia MLFlow. Podano definicje wybranych miar stosowanych w ewaluacji procesu klasyfikacji i identyfikacji.

Rozdział piąty przedstawia, jakie zbiory danych zostały wykorzystane oraz nagrania zrealizowane w Katedrze Systemów Multimedialnych, Wydziału Elektroniki, Telekomunikacji i Informatyki Politechniki Gdańskiej w kontekście wykorzystania ich w eksperymentach. Dalsza część tego rozdziału odnosi się do przygotowania zebranych danych w formie zbiorów treningowych, ewaluacyjnych oraz testowych.

Kolejny rozdział opisuje eksperyment sprawdzający, w którym zawarto architekturę wykorzystywanego modelu, opis treningu, uzyskane wyniki oraz modyfikację architektury. Zadania związane z klasyfikacją i identyfikacją instrumentów w sygnale fonicznym zostały podane w rozdziale 7. W omawianym rozdziale zawarto opis wybranych, przedstawionych w

literaturze algorytmów wraz z ich implementacją i uzyskanymi wynikami. Podano w nim też metody powiększania zbioru danych dla celów klasyfikacji instrumentów w muzyce polifonicznej. Rozdział ten można potraktować jako eksperymenty wstępne, gdyż obejmują przygotowanie i przetestowanie metod w odniesieniu do stanu wiedzy (state-of-the-art), propozycję modyfikacji ww. metod w celu poprawy jakości identyfikacji oraz podział dużego modelu sieci neuronowej na mniejsze podsieci, wyspecjalizowanych w pojedynczych instrumentach.

Rozdział 8. zawiera opis eksperymentu głównego, tj. przygotowanie modelu impulsowej sieci neuronowej będącej przykładem sieci trzeciej generacji. Stanowi on identyfikator impulsowy. Na podstawie przedstawionych wyników przygotowano analizę wyników (rozdział 9) oraz wnioski płynące z eksperymentów wraz podsumowaniem, które znalazły się w rozdziale 10. Rozdział ten zawiera również propozycje kierunków rozwoju prowadzonych badań.



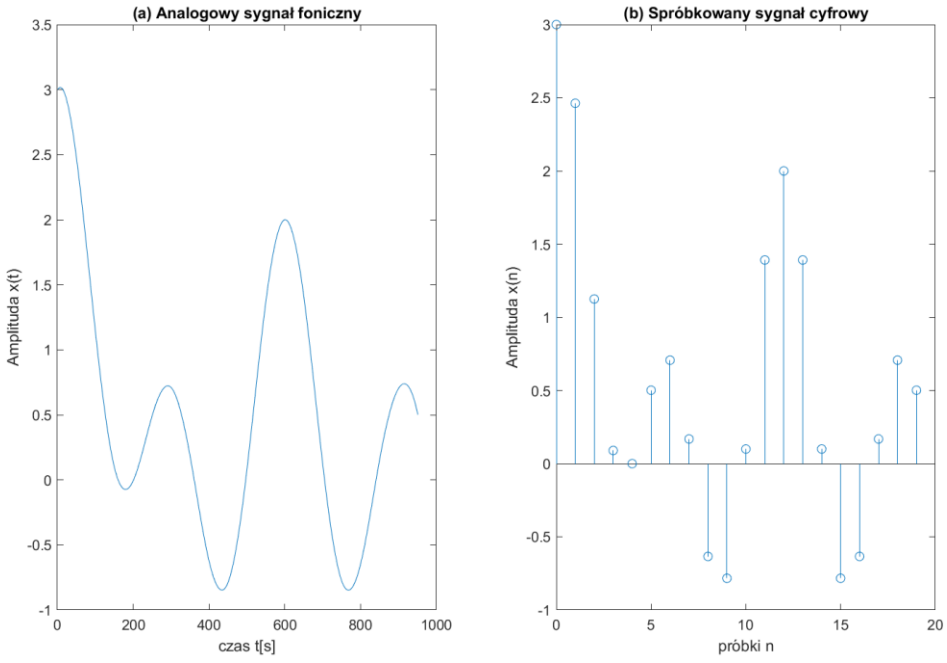
Rysunek 1.2 Układ rozdziałów pracy

2. PODSTAWY PRZETWARZANIA SYGNAŁU FONICZNEGO W KONTEKŚCIE ZASTOSOWANIA W UCZENIU MASZYNOWYM

W niniejszym rozdziale przywołano pokrótce podstawowe metody analizy sygnału fonicznego [130] w dziedzinie czasu, częstotliwości oraz czasowo–częstotliwościowej. Ponieważ metody te stanowią ogólnie znane zagadnienia, dlatego ich opis zostanie ograniczony do podania informacji, które są szczególnie przydatne w kontekście parametryzacji sygnału muzycznego i obszaru zajmującego się wyszukiwaniem informacji muzycznej (ang. *Music Information Retrieval*) [123][124][131][132]. W dalszej części rozdziału omówiono przykłady parametryzacji sygnałów przy pomocy deskryptorów standardu MPEG–7, jak również stosowanych obecnie reprezentacji 2D sygnału (postaci dwuwymiarowej). Przedstawione zagadnienia ujęto w kontekście analizy dźwięków instrumentów muzycznych.

2.1. Sygnał cyfrowe i metody konwersji sygnału analogowego

Cyfrowa reprezentacja analogowego sygnału audio jako ciągu liczb jest osiągana przez przetwornik analogowo-cyfrowy (ADC). ADC wykonuje próbkowanie amplitudy sygnału analogowego x(t) na równomiernie oddalonej siatce wzdłuż poziomej osi czasu i kwantowanie amplitudy do stałych próbek reprezentowanych przez liczby x (n) wzdłuż pionowej osi amplitudy. Przykład działania przetwornika analogowo-cyfrowego został przedstawiony na rysunku 2.1.

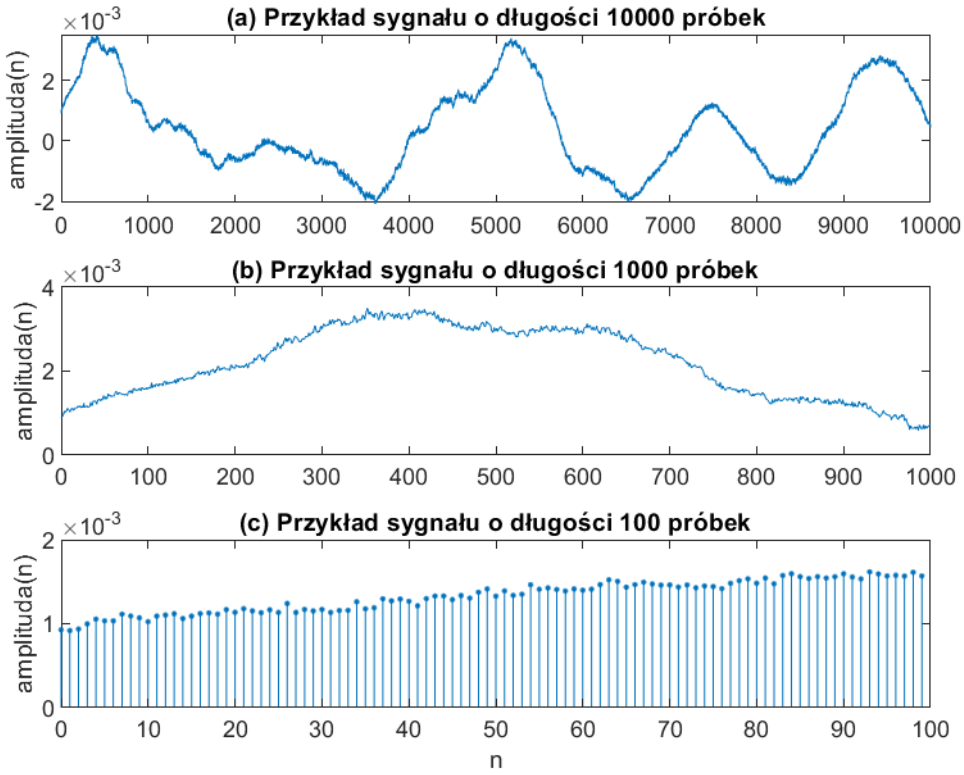


Rysunek 2.1 Przykład działania przetwornika analogowo-cyfrowego

Próbki są przedstawione jako pionowe linie z kropkami na górze. Sygnał analogowy *x(t)* oznacza amplitudę sygnału w ciągłym czasie *t* w sekundach. W następstwie działania przetwornika ADC, sygnał cyfrowy (czas dyskretny i skwantowana amplituda) jest sekwencją (strumieniem) próbek *x(n)* reprezentowanych przez liczby w dyskretnym indeksie czasowym *n*.

Odległość czasowa pomiędzy dwiema kolejnymi próbkami jest określana jako interwał próbkowania T (okres próbkowania), a odwrotność to częstotliwość próbkowania *fs=1/T*. Częstotliwość próbkowania odzwierciedla liczbę próbek na sekundę w hercach [Hz]. Zgodnie z twierdzeniem o próbkowaniu należy ją dobrać jako dwukrotność największej częstotliwości *fmax* (szerokość pasma sygnału) zawartej w sygnale analogowym, czyli *fs>2·fmax*. W przypadku wymuszenia stałej częstotliwości próbkowania *fs*, to musimy się upewnić, że sygnał wejściowy, który ma być próbkowany, ma szerokość pasma zgodną z *fmax=fs/2*. Jeśli nie, należy odrzucić wyższe częstotliwości poprzez filtrowanie za pomocą filtra dolnoprzepustowego, który przepuszcza tylko wszystkie częstotliwości niższe niż *fmax*.

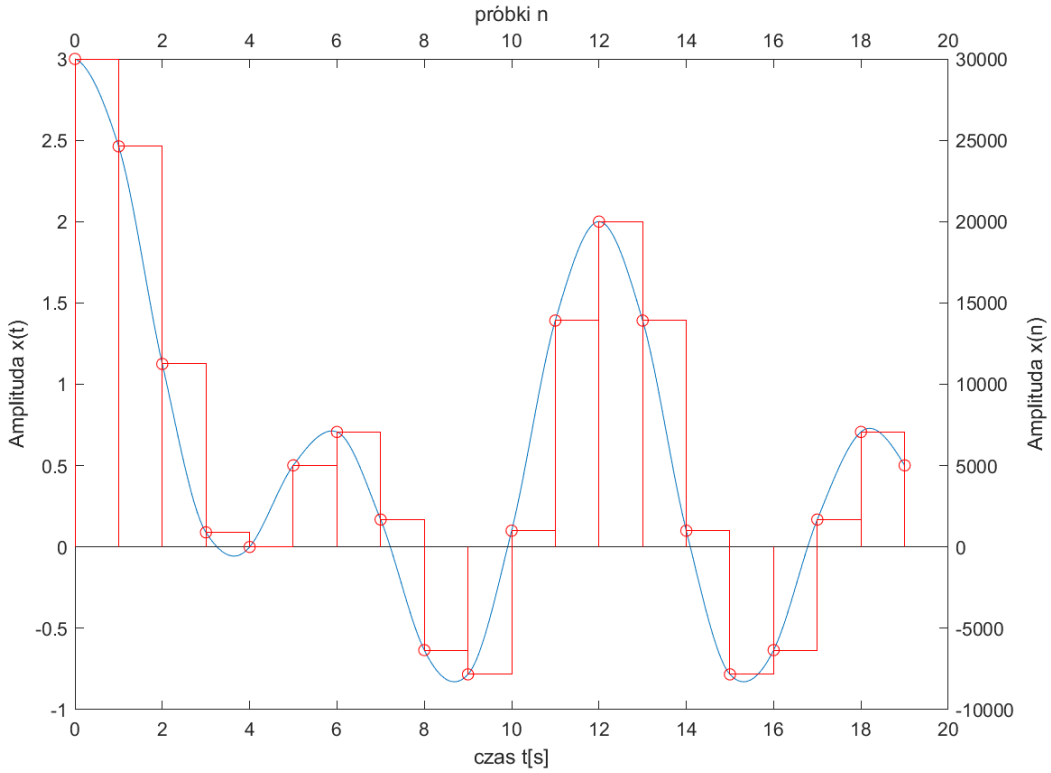
Na rysunku 2.2 przedstawiono różne sposoby przedstawienia sygnałów. Górna część (wykres (a)) pokazuje 10000 próbek, środkowa (wykres (b)) – pierwsze 1000 próbek, a dolna (wykres (c)) – pierwsze 100 próbek z cyfrowego sygnału audio. Tylko wtedy, gdy liczba próbek na wykresie jest wystarczająco mała, dla sygnału cyfrowego można zastosować reprezentację graficzną w postaci linii z kropką. Na wykresach widać również, zależność pomiędzy ilością wyświetlonych próbek a ilością informacji przeniesioną przez wykres – im próbek jest mniej, tym bardziej szczegółowo można przyjrzeć się zmienności sygnału w danym fragmencie, ale kosztem całościowego oglądu.



Rysunek 2.2 Przykład reprezentacji różnych długości cyfrowego sygnału fonicznego

2.2. Kwantyzacja sygnału analogowego

Na rysunku 2.3 przedstawiono dwa różne formaty skali pionowej dla cyfrowych sygnałów audio w porównaniu do oryginalnego – analogowego sygnału. Kwantyzacja amplitud do stałych liczb z zakresu -32768...32767 jest oparta na 16-bitowej reprezentacji amplitud próbek, która pozwala na skwantowanie 216 wartości z zakresu -215...215-1. Dla ogólnej reprezentacji w–bitowej zakres liczb wynosi od -2^{w-1} do $2^{w-1}-1$. Taka reprezentacja nazywana jest reprezentacją liczb całkowitych. Jeśli podzieli się wszystkie liczby całkowite przez maksymalną wartość bezwzględną, na przykład 32768, dojdziemy do znormalizowanej skali pionowej na rysunku 2.3, która mieści się w zakresie $-1...1-Q$, gdzie Q jest wielkością kroku kwantyzacji. Może to być obliczone z wzoru $Q = 2^{-w(-1)}$, co prowadzi do $Q = 3{,}0518 \times 10^{-5}$ dla w=16. Na rysunku 2.3 pokazane są również formaty skali poziomej, a mianowicie oś czasu ciągłego, oś czasu dyskretnego i znormalizowana oś czasu dyskretnego, które będą normalnie używane. Po tym wąskim opisie można zdefiniować sygnał cyfrowy jako sygnał dyskretny w czasie i o dyskretnej amplitudzie, który powstaje w wyniku próbkowania sygnału analogowego i kwantowania amplitudy na ustaloną liczbę wartości amplitudy. Sygnał cyfrowy jest reprezentowany przez ciąg liczb $x(n)$. Rekonstrukcja sygnałów analogowych może być przeprowadzona przez przetworniki cyfrowo–analogowe.

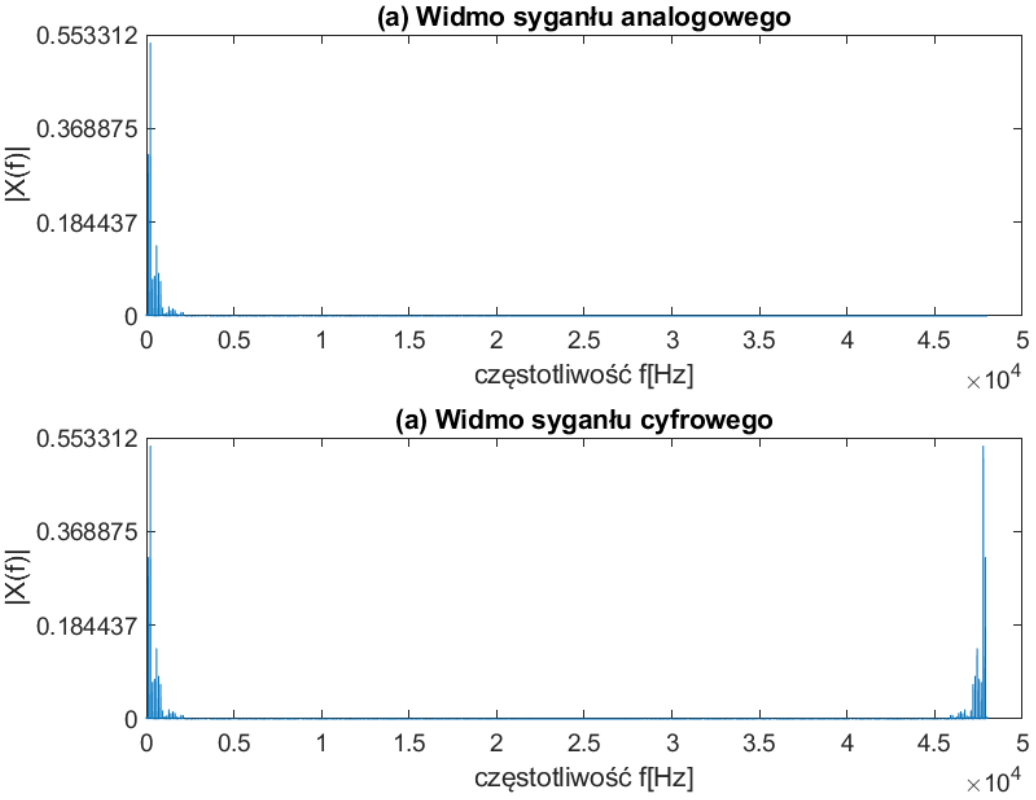


Rysunek 2.3 Przykład działania kwantyzatora

2.3. Zjawisko aliasingu i częstotliwość Nyquista

Widmo sygnału pokazuje rozkład energii sygnału w funkcji częstotliwości. Górna część rysunku 2.4 przedstawia widmo krótkiej szczeliny czasowej analogowego sygnału fonicznego.

Częstotliwości zawierają się w przedziale do 20 kHz. Próbkowanie i kwantowanie sygnału analogowego z częstotliwością próbkowania *fs=40 kHz* prowadzi do uzyskania odpowiadającego mu sygnału cyfrowego. Widmo sygnału cyfrowego o tej samej szczelinie czasowej pokazane jest w dolnej części rysunku 2.4. Operacja próbkowania prowadzi do powielenia widma sygnału analogowego w paśmie podstawowym [77]. Zawartość częstotliwościowa od 0 Hz do 20 kHz sygnału analogowego pojawia się teraz również od 40 kHz do 60 kHz, a jej odwrócona wersja od 4 kHz w dół do 20 kHz. Reprodukcja tego pierwszego obrazu widma sygnału z częstotliwością podstawową przy 40 kHz pojawi się teraz również przy całkowitych wielokrotnościach częstotliwości próbkowania *fs*=40 kHz. Widmo sygnału cyfrowego od 0 do 20 kHz wykazuje dokładnie taki sam kształt jak widmo sygnału analogowego. Rekonstrukcję sygnału analogowego z sygnału cyfrowego uzyskuje się przez filtrację dolnoprzepustową sygnału cyfrowego, odrzucając częstotliwości wyższe niż *fs/2*=20 kHz. Jeśli rozpatrzy się widmo sygnału cyfrowego w dolnej części rysunku 2.4 i odrzuci się wszystkie częstotliwości wyższe od 20 kHz, to nastąpi powrót do widma sygnału analogowego w górnej części rysunku. Opisane zjawisko nosi nazwę aliasingu, a połowa częstotliwości próbkowania *fs/2* nazywa się częstotliwością Nyquista.



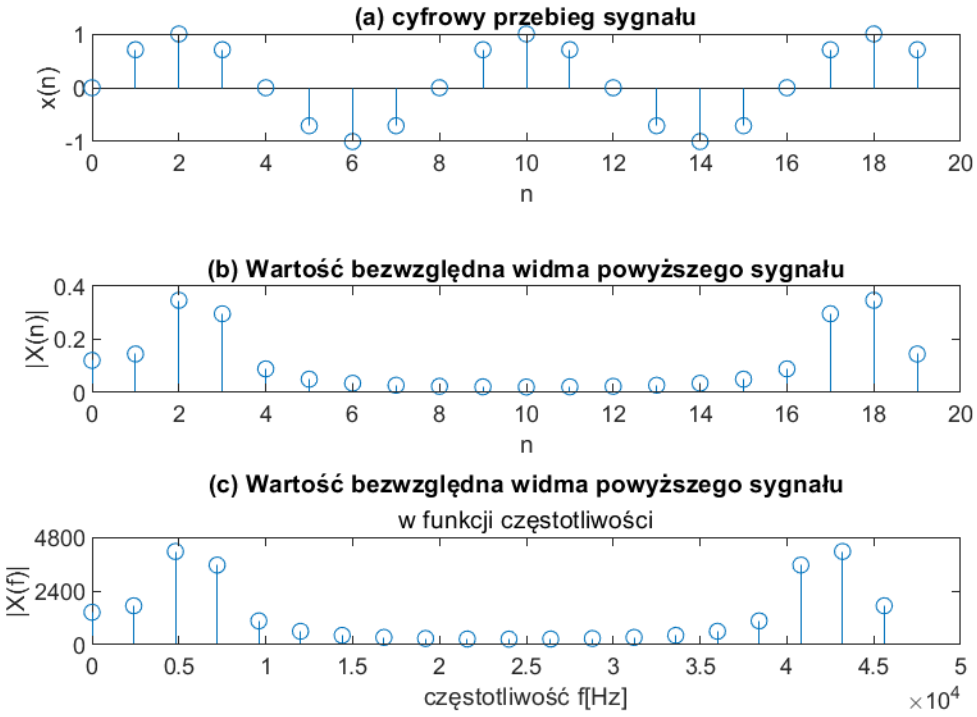
Rysunek 2.4 Porównanie widma sygnału analogowego i cyfrowego przy zadanej częstotliwości próbkowania

2.4. Dyskretna Transformacja Fouriera

Widmo sygnału cyfrowego można obliczyć za pomocą dyskretnej transformacji Fouriera (DFT), która jest określona wzorem:

$$X(k) = DFT[x(n)] = \sum_{n=0}^{N-1} x(n)e^{-j2\pi k/N} \qquad k = 0, 1, \dots, N-1 \tag{2.1}$$

Szybka wersja powyższego wzoru nosi nazwę szybkiej transformacji Fouriera (FFT). FFT pobiera *N* kolejnych próbek z sygnału *x(n)* i wykonuje operację matematyczną, aby wyliczyć *N* próbek *X(k)* widma sygnału. Rysunek 2.5 przedstawia wyniki działania 20-punktowej FFT zastosowanej do 20 próbek sygnału sinusowego. Wynik jest normalizowany przez wartość N zgodnie z formułą $X = \left|\frac{FFT(x)}{N}\right|$.



Rysunek 2.5 Przykład przebiegu sygnału cyfrowego i odpowiadającemu mu widma

N próbek $X(k) = X_R(k) + jX_I(k)$ jest wartością zespoloną z częścią rzeczywistą $X_R(k)$ i częścią urojoną $X_I(k)$, z której można obliczyć wartość bezwzględną będącą amplitudą widma ,przedstawioną równaniem 2.2 oraz fazę widma przedstawioną równaniem 2.3:

$$|X(k)| = \sqrt{X_R^2(k) + X_I^2(k)} \qquad k = 0, 1, \dots, N-1 \tag{2.2}$$

$$\varphi(k) = arctan\frac{X_I(k)}{X_R(k)} \qquad k = 0, 1, \dots, N-1 \tag{2.3}$$

Na rysunku 2.5 widać również, że algorytm FFT prowadzi do uzyskania *N* równomiernie oddalonych punktów częstotliwości, które dają *N* próbek widma sygnału począwszy od 0 *Hz* z krokiem $\frac{fs}{N}$ aż do $\frac{N-1}{N}fs$. Te punkty częstotliwościowe są zdefiniowane jako $k\frac{fs}{N}$, gdzie *k* znajduje się w zakresie 0, 1, 2, …, *N*-1. Widmo amplitudowe |*X(f)*| jest często wykreślane na logarytmicznej

skali amplitudy zgodnie z wzorem $20log_{10}\left(\frac{X(f)}{0.5}\right)$, co daje 0 *dB* dla sinusoidy o maksymalnej amplitudzie 1. Ta normalizacja jest równoważna z $20log_{10}\left(\frac{X(k)}{N/2}\right)$. Na rysunku 2.5 w postaci wykresu *c* przedstawiono taką reprezentację.

2.5. Odwrotna Dyskretna Transformacja Fouriera

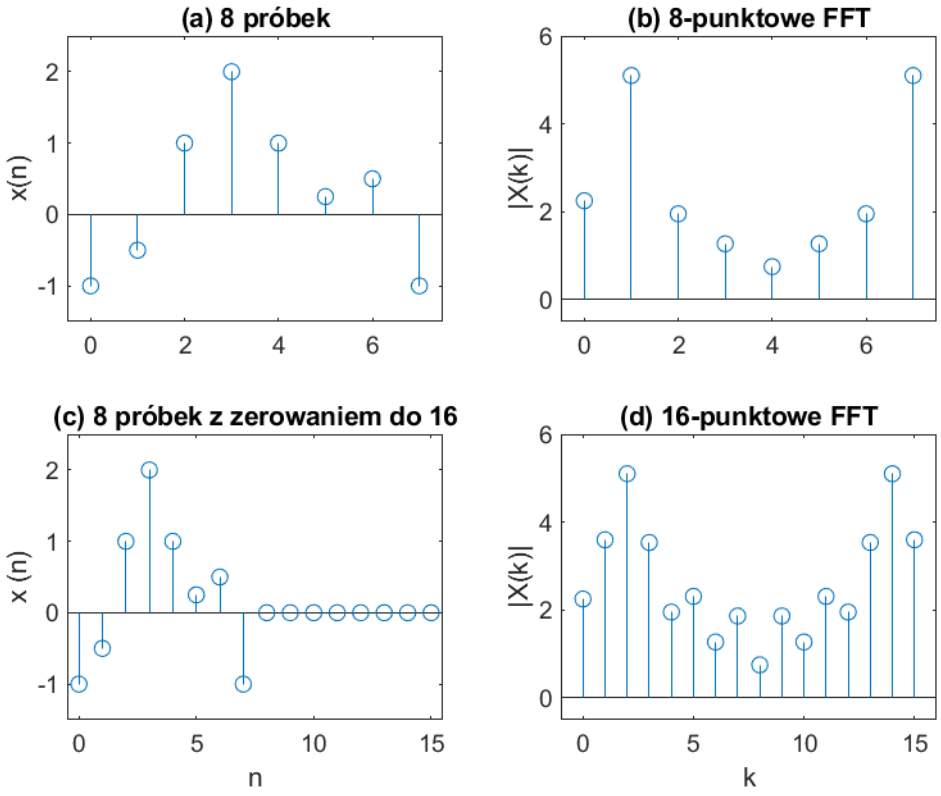
Podczas gdy DFT jest używana jako transformacja z dziedziny czasu dyskretnego do dziedziny częstotliwości dyskretnych w analizie widma, odwrotna dyskretna transformacja Fouriera (IDFT) pozwala na transformację z dziedziny częstotliwości dyskretnych do dziedziny czasu dyskretnego. Algorytm IDFT opisany jest wzorem 2.4.

$$X(n) = IDFT[X(K)] = \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{j2\pi nk/N} \qquad n = 0, 1, \dots, N-1 \tag{2.4}$$

Szybka wersja IDFT nosi nazwę odwrotnej szybkiej transformacji Fouriera (IFFT). Biorąc *N* liczb zespolonych o wartości złożonej o własności $X(k) = X*(N-k)$ w dziedzinie częstotliwości, a następnie wykonując IFFT, otrzymuje się *N* próbek czasu dyskretnego *x(n),* które mają wartość rzeczywistą.

2.6. Rozdzielczość częstotliwościowa: wypełnianie zerami i funkcje okna

Aby zwiększyć rozdzielczość częstotliwościową dla analizy widma, wykorzystywanych jest więcej próbek dla algorytmu FFT. Typowe liczby dla rozdzielczości FFT to *N* = 256, 512, 1024, 2048, 4096 i 8192. W przypadku wyznaczania widma z 64 próbek, aby zwiększyć rozdzielczość częstotliwościową z *fs*/64 do *fs*/1024, należy rozszerzyć sekwencję 64 próbek audio poprzez dodanie próbek zerowych aż do długości 1024, a następnie wykonać 1024–punktową FFT. Operacja ta nosi nazwę wypełniania/uzupełniania zerami (ang. *zero-padding)* i jest zilustrowana na rysunku 2.6. Górna lewa część pokazuje oryginalną sekwencję ośmiu próbek, górna prawa część pokazuje odpowiadający jej ośmiopunktowy wynik FFT. Dolna lewa część ilustruje dodanie ośmiu zerowych próbek do oryginalnej 8–próbkowej sekwencji do długości *N* = 16. Dolna prawa część ilustruje widmo wielkości $|X(k)|$ wynikające z 16-punktowej FFT sekwencji z zerowym wypełnieniem o długości *N*=16. Zauważyć można wzrost rozdzielczości częstotliwościowej pomiędzy 8-punktową a 16-punktową FFT. Pomiędzy każdym punktem częstotliwości górnego spektrum obliczany jest nowy blok częstotliwości w dolnym spektrum. Wartości *k* = 0, 2, 4, 6, 8, 10, 12, 14 16-punktowej FFT odpowiadają punktom *k* = 0, 1, 2, 3, 4, 5, 6, 7 ośmiopunktowej FFT. Te *N* wartości częstotliwości pokrywa zakres częstotliwości od 0 *Hz* do $\frac{N-1}{N}fs$.



Rysunek 2.6 Przykład działania FFT z zastosowanym zero-paddingiem

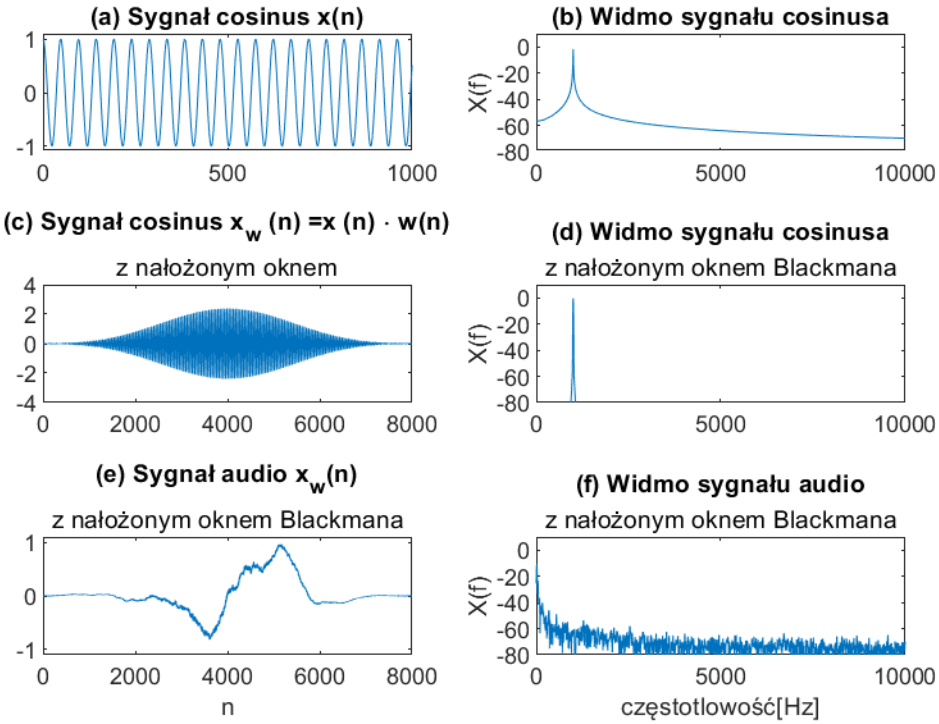
Efekt przecieku występuje w wyniku wycięcia *N* próbek z sygnału. Efekt ten jest pokazany w górnej części rysunku 2.7. Widmo kosinusowe jest rozmazane wokół częstotliwości. Możliwe jest zmniejszenie efekt wycieku poprzez wykorzystanie z funkcji okna, takich jak okno Blackmana czy okno Hamminga i ważenie *N* próbek audio przez funkcję okna. Funkcje okna zostały przedstawione w równaniach 2.5 i 2.6.

$$w_B(n) = 0.42 - 0.5\cos\left(\frac{2\pi n}{N}\right) + 0.08\cos\left(\frac{4\pi n}{N}\right), \tag{2.5}$$

$$w_H(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{N}\right) \tag{2.6}$$

$$n = 0, 1, \ldots, N-1$$

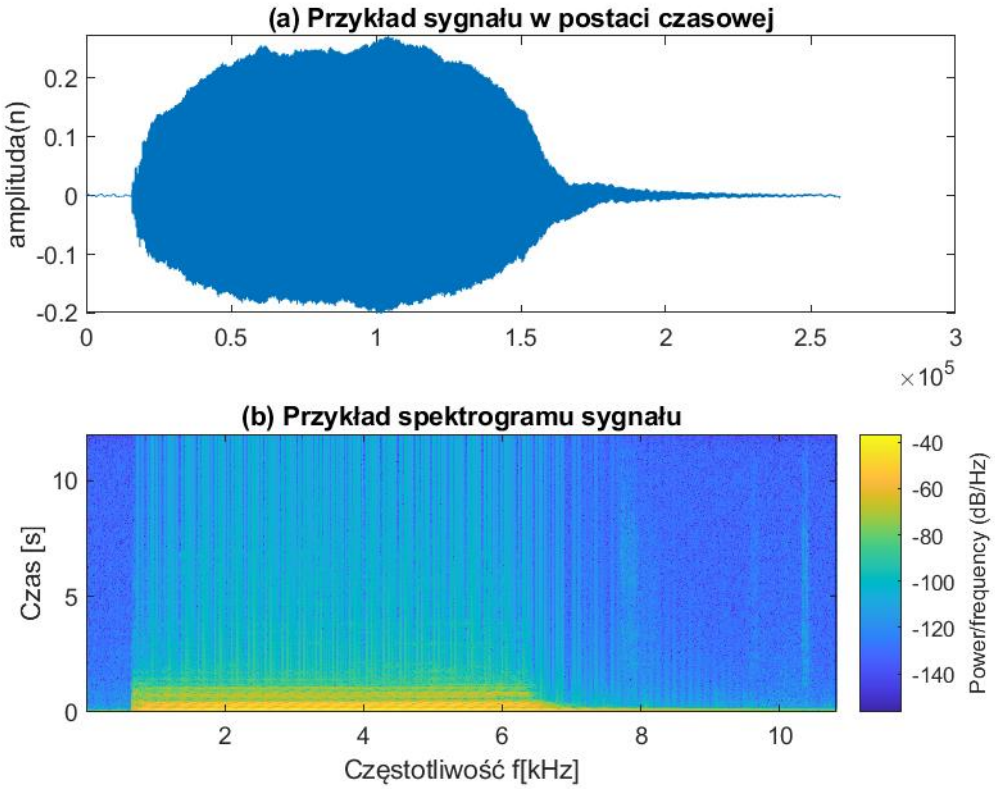
Ważenie sygnału odbywa się zgodnie z $x_w = w(n) \cdot x(n)/(\sum_k w(k))$, przy czym $0 \le n \le N-1$, a następnie wykonuje się FFT sygnału ważonego. Sygnał kosinusowy ważony przez okno i odpowiadające mu widmo pokazane jest w środkowej części rysunku 2.7. W dolnej części rysunku 2.7 pokazano segment sygnału audio ważonego oknem Blackmana oraz odpowiadające mu widmo uzyskane za pomocą FFT.



Rysunek 2.7 Przykład wyznaczania widma sygnału z zastosowaniem okna Blackmana oraz bez okna

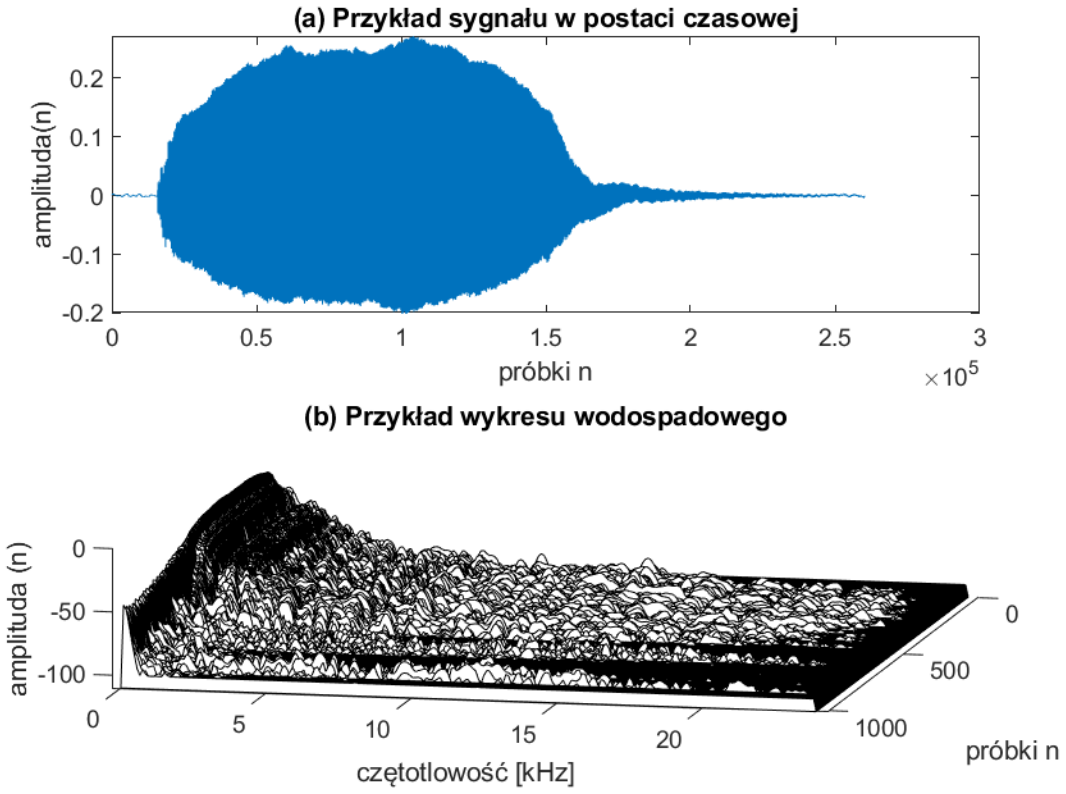
2.7. Reprezentacja czasowo-częstotliwościowa sygnału

Jednym z typów reprezentacją czasowo-częstotliwościową jest spektrogram, który daje oszacowanie krótkotrwałej, zlokalizowanej w czasie zawartości częstotliwościowej sygnału. W celu uzyskania spektrogramu sygnał jest dzielony na segmenty o długości *N,* które są mnożone przez okno i wykonywana jest operacja FFT. W celu zwiększenia lokalizacji czasowej widm krótkookresowych można zastosować nakładanie się ważonych segmentów. Szczególną wizualną reprezentacją widm krótkookresowych jest spektrogram przedstawiony na rysunku 2.8. Czas rośnie liniowo na osi poziomej, a częstotliwość na osi pionowej. Każda pionowa linia reprezentuje wartość bezwzględną *X(f)* nad częstotliwością za pomocą wartości w skali natężenia koloru żółtego. Pokazane są tylko częstotliwości do połowy częstotliwości próbkowania.



Rysunek 2.8 Przykład spektrogramu dla rzeczywistego sygnału fonicznego

Inną reprezentacją czasowo-częstotliwościową krótkotrwałej transformacji Fouriera sygnału *x(n)* jest reprezentacja wodospadowa przedstawiona na rysunku 2.9.



Rysunek 2.9 Przykład wykresu wodospadowego

2.8. Parametryzacja sygnałów fonicznych

Bazując zarówno na postaci czasowej, jak i częstotliwościowej sygnału fonicznego można wyznaczyć parametry, które szczegółowo opisują jego strukturę i zmienność. Jednym z przykładów, który dokładnie opisuje, w jaki sposób można sparametryzować sygnał jest opracowany w roku 2000 standard MPEG-7 (opublikowany w 2002 r. jako standard ISO/IEC 15938 [133]) omówiony szerzej w niniejszym rozdziale [45][134]

Parametry sygnału można wyznaczyć w różnych środowiskach obliczeniowych, np. w Matlabie, Excelu, przy pomocy biblioteki Librosa [129] zaimplementowanej w języku programowania Python. Przykłady wykorzystania biblioteki Librosa zawarto w dalszej części rozdziału.

2.8.1. *Deskryptory standardu MPEG-7*

Celem standardu MEPG-7 (ISO/IEC 15938) jest opis zawartości danych multimedialnych, umożliwiający przeszukiwanie treści zakodowanych obiektów [135]. MPEG-7 definiuje język opisu zawartości obiektów multimedialnych w odniesieniu do danych multimedialnych (sygnał wizyjny, obraz, sygnał foniczny, meta dane oraz dane tekstowe) [50].

Deskryptory standardu MPEG-7 można podzielić na sześć podstawowych grup [134][136]:

1. Basic – deskryptory dedykowane do każdego typu sygnałów, bazują na okresowo próbkowanych wartościach sygnału.
2. Basic Spectral – cztery deskryptory wyznaczone na podstawie pojedynczej analizy czasowo–częstotliwościowej sygnału.
3. Spectral Basis – dwa deskryptory będące niskowymiarową projekcją wielowymiarowej przestrzeni widmowej.
4. Signal Parameters – dwa deskryptory opisujące okresowość sygnału.
5. Timbral Temporal – dwa deskryptory opisujące barwę dźwięku w czasowych segmentach sygnału.
6. Timbral Spectral – pięć deskryptorów opisujących barwę dźwięku, bazując na liniowo–częstotliwościowym widmie sygnału.

Parametry te należą do klasy parametrów niskopoziomowych. Rozwinięcie poszczególnych grup deskryptorów znajduje się w tabeli 2.1.

Tabela 2.1 Opis deskryptorów standardu MPEG-7 [135]

Grupa	Deskryptory niskiego poziomu	Akronim	Opis
Basic	AudioWaveform	AW	Obwiednia sygnału przygotowana szczególnie w celu wizualizacji.
	AudioPower	AP	Moc sygnału uśredniona w czasie.
BasicSpectral	AudioSpectrumEnvelope	ASE	Krótkookresowe widmo sygnału wyznaczone dla pasm częstotliwości w skali logarytmicznej.
	AudioSpectrumCentroid	ASC	Środek ciężkości widma sygnału wyznaczonego dla logarytmicznych wartości częstotliwości. Wskazuje, w której części widma znajduje się najwięcej energii.
	AudioSpectrumSpread	ASS	Deskryptor opisujący jak blisko środka ciężkości rozłożona jest energia widma sygnału.
	AudioSpectrumFlatness	ASF	Odchylenie od płaskiego kształtu widma w danych pasmach częstotliwości.
SpectralBasis	AudioSpectrumBasis	ASB	Seria potencjalnie zależnych od czasu funkcji bazowych wyznaczonych przy pomocy dekompozycji znormalizowanego widma przedstawionego w skali decybelowej.
	AudioSpectrumProjection	ASP	Deskryptory będące projekcją AudioSpectrumBasis na

			zredukowaną liczbę funkcji bazowych.
SignalParameters	AudioHarmonicity	AH	Informacja o harmoniczności sygnału.
	AudioFundamentalFrequency	AFF	Częstotliwość podstawowa sygnału.
TimbralTemporal	LogAttackTime	LAT	Czas, w jakim amplituda sygnału rośnie od ciszy do wartości maksymalnej.
	TemporalCentroid	TC	Punkt w czasie, w którym skumulowana jest energia sygnału.
TimbralSpectral	SpectralCentroid	SC	Środek ciężkości widma, wskazujący, wokół której częstotliwości skupiona jest energia.
	HarmonicSpectralCentroid	HSC	Średnia ważona harmonicznych szczytów w widmie, wyrażająca środek ciężkości widma, oparty jedynie na sygnale nie będącym szumem.
	HarmonicSpectralDeviation	HSD	Odchylenie prążków od środka ciężkości widma wyrażonego w skali decybelowej.
	HarmonicSpectralSpread	HSS	Rozrzut wartości prążków widma wokół środka ciężkości.
	HamonicSpectralVariation	HSV	Znormalizowana korelacja amplitud wartości harmonicznych widma pomiędzy dwoma następującymi po sobie ramkami sygnału.

Warto zauważyć, że w literaturze tematu znajdują się odniesienia do zarówno do nisko-, jak i wysokopoziomowych parametrów sygnałów muzycznych (np. barwa dźwięku, dźwięk jasny, ciemny), a także do poszukiwania powiązań pomiędzy parametrami nisko- i wysokopoziomowymi (tzw. *semantic gap*) [22][23]. Dlatego w badaniach dotyczących inteligentnego przetwarzania muzyki widać oba typy parametrów w wektorze cech [4][11][55][65][90].

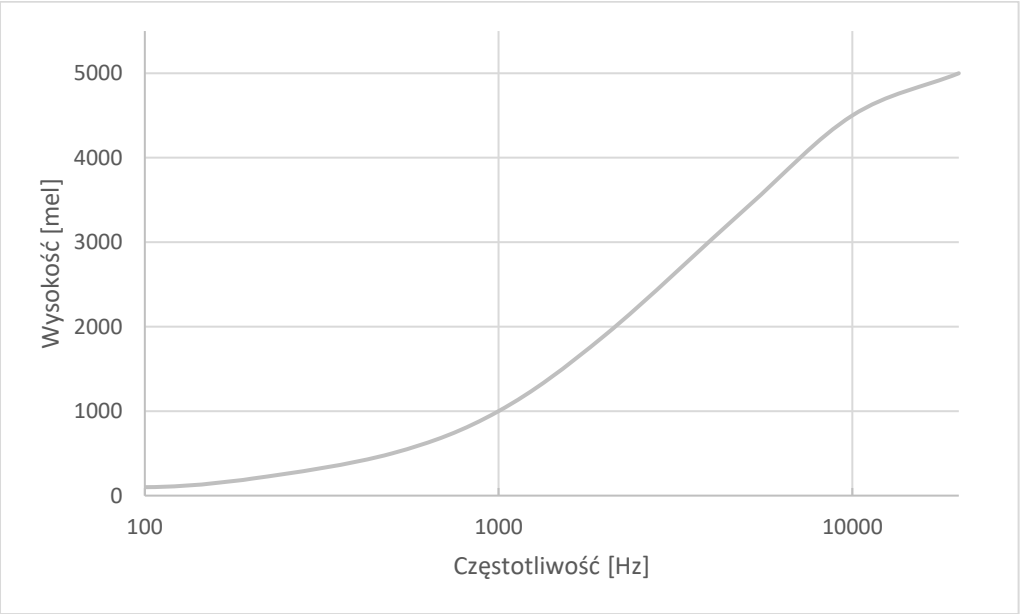
Przykłady obliczania i wizualizacji parametrów zostały przedstawione w kolejnych rozdziałach. W szczególności rozdział 2.8.3 przywołuje język Python i jego biblioteki, w tym pakiet Librosa, które pozwalają na łatwą implementację parametrów sygnałów fonicznych.

2.8.2. *Reprezentacja 2D sygnałów muzycznych*

W rozdziale 2.7 zaprezentowano opis i przykład podstawowej reprezentacji sygnału w postaci dwu wymiarowej jaką jest spektrogram. Poniżej zawarto opisy innych stosowanych metod przedstawienia sygnału w postaci 2D, jakimi są spektrogramy w skali melowej (tzw. mel-spektrogramy), mel-cepstrogramy czy chromagramy.

Mel-spektrogramy

Skala melowa jest percepcyjną skalą wysokości dźwięków ocenianych przez słuchaczy jako równe pod względem odległości od siebie. Punkt odniesienia pomiędzy tą skalą a pomiarem częstotliwości jest definiowany poprzez przypisanie percepcyjnej wysokości 1000 melów tonowi o częstotliwości 1000 Hz przy 40 dB powyżej progu słyszenia. Powyżej około 500 Hz coraz większe interwały są oceniane przez słuchaczy jako dające równe przyrosty wysokości dźwięku. Rysunek 2.10 przedstawia funkcję przejścia pomiędzy skalą melową a częstotliwościową [99]. Spektrogram w skali melowej (mel-spektrogram) jest więc reprezentacją, w której częstotliwości zostają przekonwertowane do skali melowej.



Rysunek 2.10 Funkcja pokazująca zależność pomiędzy postrzeganą wysokością dźwięku w melach a rzeczywistą częstotliwością bodźca [99][137]

Mel-cepstrogramy

Cepstrum można traktować jako widmo z widma logarytmicznego. Aby je wyznaczyć, stosuje się następujące przekształcenie [51]:

$$\hat{x}(l) = \frac{1}{M}\sum_{m=1}^{M} \ln\left(S_k\,(m)\right) e^{(2\pi j)(m-1)\frac{l-1}{M}} \tag{2.7}$$

gdzie $S_k(m)$ jest widmem ramki $k, l = 1, \dots, M$, gdzie M jest długością tej ramki [51].

Cepstrum w skali melowej zostało zaproponowane dla celów rozpoznawania mowy w 1976 roku przez Mermelsteina [72]. Ideą cepstrum w skali melowej jest imitowanie ludzkiego systemu słuchowego. Uzyskuje się to poprzez zastosowanie do sygnałów mowy przetwarzania opartego na banku filtrów. Bank filtrów powstaje poprzez ułożenie filtrów trójkątnych liniowo w przestrzeni częstotliwości mel i w ten sposób imituje nieliniową i logarytmiczną percepcję

ludzkiego systemu słuchowego. Analiza cepstrogramu w skali melowej (znanego również jako Mel-Frequency Cepstral Coefficients, MFCC) stała się dominującą techniką w rozpoznawaniu sygnału mowy [15][26][27][38][65][96]. Sposób obliczania cepstrogramu dla skali melowej jest taki sam, jak w przypadku skali liniowej. Filtracja sygnału jest wykonywana przed operacją logarytmowania (równanie (2.7)).

Chromagramy

Wysokość dźwięku jest jedną z podstawowych cech w przetwarzaniu sygnałów akustycznych. Od dawna muzycy zwracali uwagę na fakt, że wysokość dźwięku składa się z dwóch komponentów: wysokości tonu i jego barwy [5][95]. Wysokość tonu jest wrażeniem percepcyjnym, względem której można uporządkować dźwięki na skali muzycznej. Barwa tonu zawiera wszystkie zmiany wysokości dźwięku w obrębie jednej oktawy. W literaturze [9] wysokość dźwięku jest często ilustrowana jako spirala, której wymiar pionowy to wysokość dźwięku, a wymiar kołowy to barwa dźwięku. Patterson [79] uzasadnił tę samą zależność dla częstotliwości, można ją opisać jako:

$$f = 2^{c+h} \tag{2.8}$$

gdzie *c* jest barwą ($c \in < 0, 1 >$), *f* – częstotliwością, a *h* − wysokością dźwięku.

Równanie (2.8) oznacza, że odległość między dwoma częstotliwościami zależy od dwóch wartości, tj. barwy − *c* i wysokości dźwięku − *h*, a nie od samej częstotliwości − *f*. Zgodnie z tym równaniem, chromatyczność można opisać następującym wzorem:

$$c = log_2(f) - h \tag{2.9}$$

Wysokość częstotliwości może być wyrażona jako:

$$h = \lfloor log_2(f) \rfloor \tag{2.10}$$

gdzie $\lfloor * \rfloor$ oznacza funkcję zaokrąglenia w dół [112].

Chromagram jest reprezentacją rozkładu energii sygnału na chromatykę i czas. Najczęściej jest wykorzystywany w zastosowaniach związanych z przetwarzaniem muzyki. Zazwyczaj stosuje się podział przedziału [0, 1] na 12 części. Zgodnie z tym podziałem wektor barwy sumuje energię widmową na 12 kubełków odpowiadających 12 półtonom w obrębie oktawy. Półton jest najmniejszym interwałem muzycznym powszechnie stosowanym w zachodniej muzyce tonalnej [56]. Czas jest przedstawiony wzdłuż osi x, kubełki chromatyczne są przedstawione wzdłuż osi y, a kolor wskazuje intensywność sygnału względem czasu i kubełków. Przykład chromagramu przedstawiony jest na rysunku 2.11 w rozdziale 2.8.3.

Transformacja CQT

Transformacja Constant-Q (CQT) umożliwia obliczenie czasowo-częstotliwościowej reprezentacji sygnału na podstawie sygnału w dziedzinie czasu. W wyniku transformacji obliczana jest ilość mocy sygnału w przedziałach częstotliwości. Wyznaczone kubełki częstotliwościowe są rozmieszczone geometrycznie, a ich współczynniki Q są równe. Pokazuje ona sygnał w przestrzeni dwuwymiarowej z większą rozdzielczością częstotliwościową przy niskich częstotliwościach i większą rozdzielczością czasową przy wyższych częstotliwościach. Można przyjąć, że definicyjnie CQT jest tożsama z transformatą falkową, ale preferowana jest osobna nazwa, ponieważ podkreśla fakt stosunkowo wysokich współczynników Q − od 12 do 96 na oktawę.

Przedstawiony kod przedstawia, w jaki sposób obliczono transformację CQT:

```
# Obliczenie ilości oktaw
n_octaves = int(np.ceil(float(n_bins) / bins_per_octave))
n_filters = min(bins_per_octave, n_bins)

# Obliczenie górnej częstoliwości granicznej
freqs = interval_frequencies(
    n_bins=n_bins,
    fmin=fmin,
    intervals=intervals,
    bins_per_octave=bins_per_octave,
    sort=True,
)
freqs_top = freqs[-bins_per_octave:]

alpha = calculate_alpha(freqs=freqs, bins_per_octave=bins_per_octave)

# Wyznaczenie długości falek
lengths, filter_cutoff = filters.wavelet_lengths(
    freqs=freqs,
    sr=sr,
    window=window,
    filter_scale=filter_scale,
    gamma=gamma,
    alpha=alpha,
)

# Iterowanie wzdłóż oktaw
my_y, my_sr, my_hop = y, sr, hop_length
cqt_resp = []

for i in range(n_octaves):
    if i == 0:
        sl = slice(-n_filters, None)
    else:
        sl = slice(-n_filters * (i + 1), -n_filters * i)
    freqs_oct = freqs[sl]
    alpha_oct = alpha[sl]

    fft_basis, n_fft, = vqt_filter_fft(
        my_sr,
        freqs_oct,
        filter_scale,
```

```
            norm,
            sparsity,
            window=window,
            gamma=gamma,
            dtype=dtype,
            alpha=alpha_oct,
        )

        cqt_resp.append(
                __cqt_response(my_y, n_fft, my_hop, fft_basis, pad_mode,
dtype=dtype)
        )

    CQT = trim_stack(cqt_resp, n_bins, dtype)

    return CQT
```

W kontekście przedstawionego kodu proces ten obejmuje obliczanie i stosowanie filtrów VQT (Variable-Q transform) dla każdej oktawy, a następnie łączenie ich odpowiedzi dla każdej oktawy w celu skonstruowania ostatecznej reprezentacji CQT. Odpowiedź filtra VQT uzyskuje się przez splot wejściowego sygnału audio x(n) z podstawą FFT. Operację splotu można wyrazić matematycznie w sposób opisany wzorem 2.11.

$$y(m) = \sum_m x(n-m)h(m) \tag{2.11}$$

gdzie $h(m)$ jest przebiegiem bazowym FFT w dyskretnym indeksie czasowym m.

2.8.3. *Pakiet Librosa*

Twórcy pakietu Librosa w odpowiedzi na rozwijający się obszar badań dotyczący wyszukiwania informacji muzycznej (MIR) oraz problemami ze stabilnością, skalowalnością i łatwością użycia deskryptorów sygnałów fonicznych postanowili przygotować własne narzędzie. Jego zadaniem było przede wszystkim ułatwienie badaczom z dziedziny przetwarzania sygnałów fonicznych wykorzystanie języka programowania Python w prosty i przejrzysty sposób [71].

Koncepcja budowy pakietu Librosa

Pakiet Librosa wykorzystuje kilka kluczowych koncepcji – m.in. zapewnia niską barierę zaznajomienia się z pakietem w przypadku naukowców zaznajomionych ze środowiskiem MATLAB. W szczególności, ułatwia to stosunkowo płaski układ pakietów, a w ślad za *scipy* [138], wykorzystano typy danych i funkcje numpy, a nie hierarchię klas abstrakcyjnych [113].

Pakiet ten zapewnia standaryzację interfejsów, nazw zmiennych i (domyślnych) ustawień parametrów w różnych funkcjach analizy. Zadanie to było utrudnione przez fakt, że implementacje referencyjne, pochodzą od różnych autorów i często są zaprojektowane jako jednorazowe skrypty, a nie właściwe funkcje biblioteczne z dobrze zdefiniowanymi interfejsami. Kolejną cechą pakietu Librosa jest zachowanie kompatybilności wstecz z istniejącymi implementacjami referencyjnymi. Osiągnięto to poprzez testy regresji dla numerycznej równoważności wyjść. Ponadto, funkcje są zaprojektowane jako modułowe, pozwalając na dostarczanie własnych

funkcji, gdy jest to konieczne. Pozwala to badaczom na wykorzystanie istniejących funkcji bibliotecznych podczas eksperymentowania z ulepszaniem poszczególnych komponentów. Wszystkie te zabiegi wydają się proste i oczywiste, ale w praktyce monolityczne projekty i brak interoperacyjności pomiędzy różnymi bazami kodu historycznie utrudniają takie podejście.

Finalnie, autorzy dążą do czytelnego kodu, dokładnej dokumentacji i wyczerpujących testów. Wszystkie prace rozwojowe prowadzone są na GitHubie. Zastosowano nowoczesne praktyki rozwoju oprogramowania, takie jak ciągłe testowanie integracji i pokrycia. Wszystkie funkcje są zaimplementowane w Pythonie, dokładnie udokumentowane przy użyciu pakietu Sphinx i zawierają przykładowy kod demonstrujący użycie. Implementacja jest w większości zgodna z zaleceniami PEP-8 [139], z małym zestawem wyjątków dla nazw zmiennych, które sprawiają, że kod jest bardziej zwięzły bez poświęcania przejrzystości: np. *y* i *sr* są preferowane w stosunku do bardziej dosadnych nazw, takich jak *audio_buffer* i *sampling_rate* [71].

Przetwarzanie sygnałów fonicznych w pakiecie Librosa

Sygnał foniczny w pakiecie Librosa jest reprezentowany jako jednowymiarowa tablica numpy, oznaczana jako *y*. Zazwyczaj towarzyszy mu częstotliwość próbkowania (oznaczana jako *sr*), która oznacza częstotliwość (w Hz), z jaką próbkowane są wartości sygnału.

Domyślnie, podczas ładowania plików audio stereo, funkcją *Librosa.load()* dokonuje sumowania kanałów do mono poprzez uśrednienie lewego i prawego kanału, a następnie ponownie próbkuje się sygnał monofoniczny do domyślnej częstotliwości *sr=22050 Hz*.

Większość metod analizy audio działa nie na natywnej częstotliwości próbkowania sygnału, ale na małych ramkach sygnału, które są oddzielone długością skoku (w próbkach). Domyślne ramki i długości skoków są ustawione odpowiednio na 2048 i 512 próbek. Przy domyślnej częstotliwości próbkowania 22050 Hz, odpowiada to nakładającym się ramkom o długości około 93 ms, oddalonym od siebie o 23 ms. Ramki są domyślnie wyśrodkowane, więc indeks ramki *t* odpowiada wycinkowi opisanemu wzorem 2.12:

$$y[(t * hop\_length - frame\_length / 2): (t * hop\_length + frame\_length / 2)], \quad (2.12)$$

– gdzie *hop_length* oznacza długość skoku, a *frame_length* długość ramki. Jeśli nie określono inaczej, wszystkie analizy z oknami przesuwnymi domyślnie używają okna Hanna. W przypadku analiz, które nie używają ramek o stałej szerokości (takich jak transformacja Constant–Q), domyślna długość okna 512 jest zachowana, aby ułatwić wyrównanie wyników. Większość analiz cech zaimplementowanych przez *Librosa* produkuje dwuwymiarowe dane wyjściowe przechowywane jako numpy.ndarray, np. *S*[*f, t*] może zawierać energię w określonym paśmie częstotliwości *f* przy indeksie klatki *t*. Zastosowano konwencję, zgodnie z którą ostatni wymiar dostarcza indeks w czasie, np. *S*[:, 0], *S*[:,1] dostęp do cech w pierwszej i drugiej ramce. Tablice cech są zorganizowane w pamięci w układzie kolumnowym (w stylu Fortranu), tak by wspólne wzorce dostępu do pamięci.

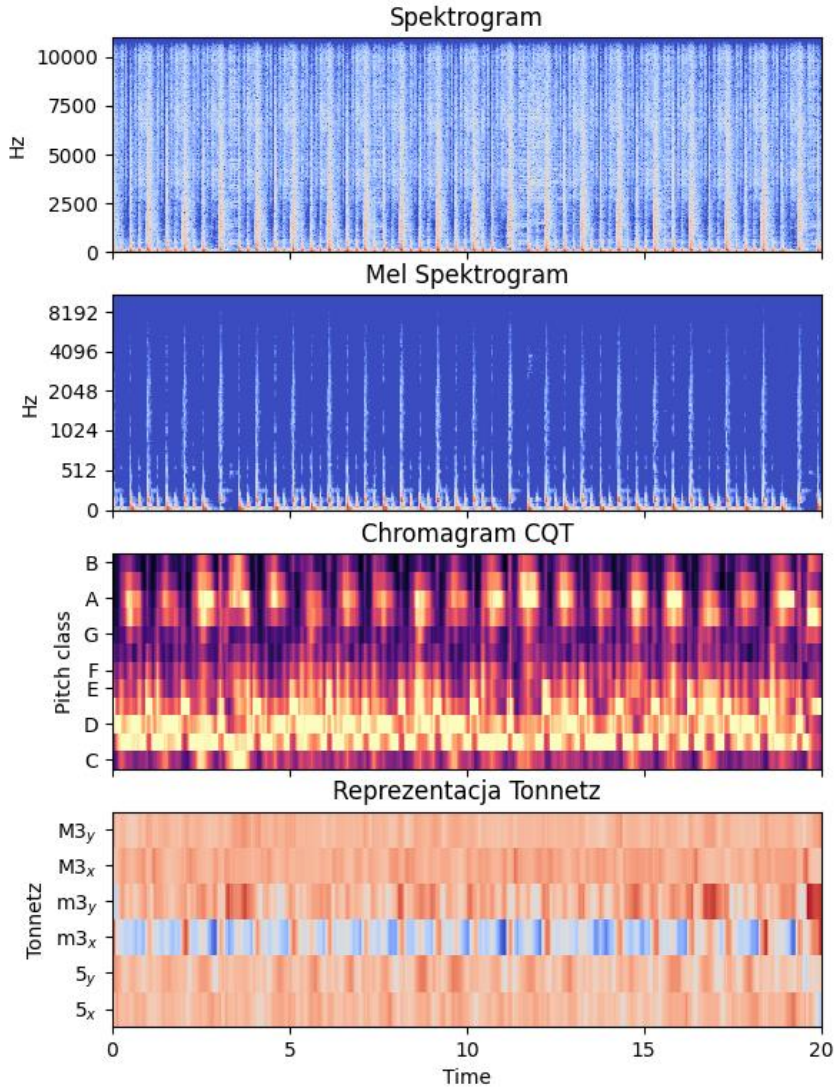
Domyślnie wszystkie analizy oparte na wysokości dźwięku są przygotowane jako odnoszące się do 12-zakresowej równomiernie rozłożonej skali chromatycznej z dostrojeniem A440 równym 440,0 Hz. Analizy wysokości dźwięku i klasy wysokości są ułożone w taki sposób, że zerowy blok odpowiada dźwiękowi C dla klasy wysokości dźwięku lub C1 (32,7 Hz) dla pomiarów wysokości absolutnej [71].

Opis parametrów spektralnych z pakietu Librosa

Reprezentacje spektralne − rozkłady energii na zbiorze częstotliwości − stanowią podstawę wielu technik analizy w MIR (Music Information Retrieval) i ogólnie w cyfrowym przetwarzaniu sygnałów. Moduł *Librosa.feature* implementuje różne reprezentacje spektralne, z których większość oparta jest na krótkotrwałej transformacie Fouriera.

Skala częstotliwości wyrażona w melach jest powszechnie używana do reprezentacji sygnałów audio, ponieważ zapewnia przybliżony model ludzkiej percepcji częstotliwości [100]. Zarówno spektrogram w skali melowej (*Librosa.feature.melspectrogram*), jak i powszechnie używane współczynniki mel-cepstralne (MFCC) (*Librosa.feature.mfcc*) są dostępne algorytmicznie w ramach pakietu. Domyślnie, skale mel są zdefiniowane tak, by odpowiadały implementacji dostarczonej przez *Slaney's auditory toolbox* [140], ale mogą być dopasowane do zestawu narzędzi *Hidden Markov Model Toolkit* (HTK) poprzez ustawienie flagi *htk=True* [115].

Podczas gdy reprezentacje w skali melowej są powszechnie używane do odwzorowania aspektów związanych z barwą sygnałów muzycznych, zapewniają one słabą rozdzielczość tonów i klas wysokości. Reprezentacje klasy wysokości dźwięku są często używane do kodowania harmonii przy jednoczesnym tłumieniu różnic w wysokości oktawy, głośności lub barwy. Dostępne są dwie elastyczne implementacje chromagramów: jedna używa analizy STFT ze stałym oknem (*chroma_stft*), a druga używa analizy transformacji CQT (stałego Q) ze zmiennym oknem (*chroma_cqt*). Alternatywną reprezentację wysokości dźwięku i harmonii można uzyskać za pomocą reprezentacji Tonnetz, która szacuje centroidy tonalne jako współrzędne w sześciowymiarowej przestrzeni interwałowej za pomocą metody Harte'a [40]. Rysunek 2.11 ilustruje różnicę pomiędzy reprezentacjami STFT, mel-spektrogramem, chromagramem i reprezentacją Tonnetz. Warto zauważyć, że reprezentacje sygnału przedstawione na rys. 2.1 mają postać 2D.



Rysunek 2.11 Zobrazowane przykłady dla wyliczonych przy pomocy pakietu Librosa reprezentacji sygnału: 1. spektrogram, 2. spektrogram w skali melowej, 3. chromagram, 4. reprezentacja Tonnetz

Poza cechami w skali melowej i chromatycznej, podmoduł cech zapewnia możliwość uzyskania szeregu statystyk spektralnych, w tym środek ciężkości widma (ang *spectral_centroid*), rozpiętość częstotliwości występujących w widmie (ang. *spectral bandwidth)*, częstotliwość poniżej której znajduje się określona część energii (ang. *spectral rolloff)* [49] i „szczyty i doliny" widmowe z ich różnicą w podpasmach (ang. *spectral contrast)* [46].

Wreszcie podmoduł cech dostarcza kilka funkcji do implementacji powszechnych transformacji cech czasowych w MIR. Obejmuje funkcję delta, która zapewnia wygładzoną estymację pochodnej czasowej; stack_memory, która konkatenuje wejściową tablicę cech z opóźnionymi w czasie kopiami samej siebie (efektywnie symulując n–gramy cech) oraz sync, która stosuje podaną przez użytkownika funkcję agregacji (np. numpy.mean lub medianę) w określonych przedziałach kolumn.

3. ROZWÓJ METOD UCZENIA MASZYNOWEGO − SZTUCZNE SIECI NEURONOWE

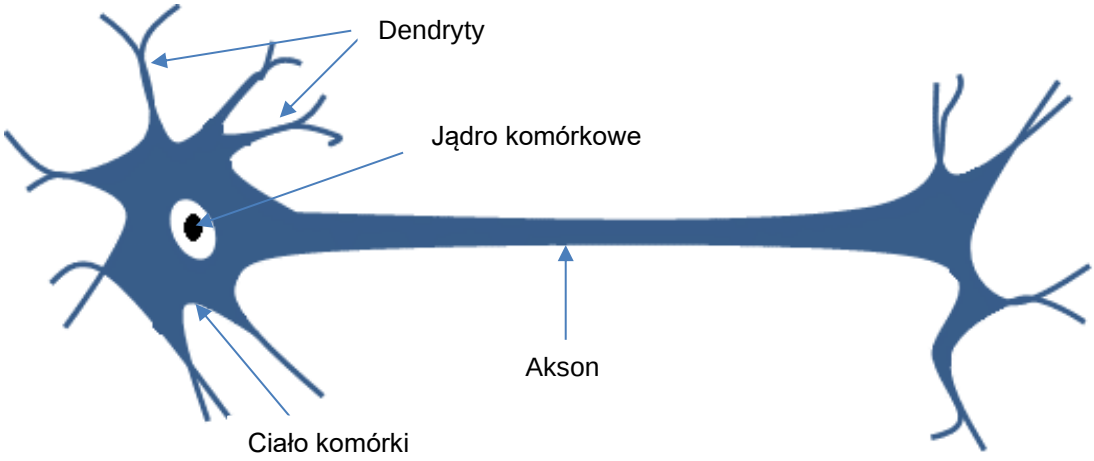
Niniejszy rozdział przywołuje pokrótce podstawy uczenia maszynowego, ze szczególnym uwzględnieniem sztucznych sieci neuronowych. W pierwszej kolejności podany zostanie krótki rys historyczny dotyczący sieci tzw. pierwszej generacji wraz z odniesieniem do komórki nerwowej i jej działania, modelu matematycznego neuronu oraz przykładów architektur sieci. Kolejny podrozdział przedstawia sieci 2. generacji wraz z przykładami narzędzi do ich implementacji i ich porównanie. Ostatni podrozdział dotyczy sieci 3. generacji, które pojawiły się stosunkowo niedawno. W szczególności przedstawione zostaną podstawy biologiczne sieci impulsowych oraz przykłady wykorzystywanych narzędzi.

Goodfellow, Bengio i Courville opisują w swojej pracy z roku 2016 trójfalowy rozwój głębokiego uczenia [37][105]. Pierwsza fala, obejmująca lata 1940–1960, składała się z wczesnych sieci neuronowych i została określona jako fala cybernetyki. Fala ta składała się przede wszystkim z perceptronu Rosenblatta, który został rozwinięty z idei wzmacniania synaptycznego Hebba oraz neuronu McCullocha-Pittsa [25][87][126]. Kluczową ideą były odmiany stochastycznego zejścia gradientowego [141]. Jednak uważa się, że ta faza rozwoju sieci została zatrzymana przez książkę „Perceptrons” [73], która w zasadzie zatrzymała wszelkie badania nad sieciami neuronowymi na 15 lat, a okres ten określa się mianem "Zimy AI" (*ang. AI – Artificial Intelligence*). Druga fala w latach 1980–1990, rozpoczęła się wraz z odkryciem propagacji wstecznej przez Rumelharta i innych i została określona jako fala konekcjonizmu [92]. Ten trend umożliwił trening wielowarstwowych sieci neuronowych, kluczowy element, którego brakowało w erze Rosenblatta. Trzecia fala trwająca od 2006 roku, prawdopodobnie rozpoczęła się od głębokiej sieci Hintona i jest określana jako głębokie uczenie [43]. Kluczową ideą głębokiego uczenia jest hierarchia wielu warstw w sieci neuronowej [47].

3.1. Biologiczne Sieci Neuronowe

Neuron (lub komórka nerwowa) jest specjalną komórką biologiczną, która przetwarza informacje. Składa się z somy, czyli ciała komórkowego oraz aksonu i dendrytów rozchodzących się i przypominających drzewa. Ciało komórki posiada jądro, które zawiera informacje o cechach dziedzicznych oraz plazmę, w której znajduje się mitochondrium potrzebne do produkcji energii. Neuron otrzymuje sygnały (impulsy) od innych neuronów poprzez swoje dendryty (odbiorniki) i przekazuje sygnały generowane przez jego ciało komórkowe wzdłuż aksonu (nadajnika), który ostatecznie rozgałęzia się na sploty i podłańcuchy. Na końcach tych nitek znajdują się synapsy. Synapsa jest to elementarna struktura i jednostka funkcjonalna pomiędzy dwoma neuronami (aksonem jednego neuronu i dendrytem drugiego). Gdy impuls dociera do synapsy, uwalniane są pewne substancje chemiczne zwane neuroprzekaźnikami. Neuroprzekaźniki dyfundują przez szczelinę synaptyczną, aby wzmacniać lub hamować sygnał, w zależności od fazy aktywacji neuronu, neuronu receptora lub przejawów własnej aktywności neuronu receptorowego, charakteryzującą się emisją impulsów elektrycznych. Efektywność synapsy może być

regulowana przez sygnały przechodzące tak, że synapsy mogą uczyć się od działań, w których uczestniczą. Zależność od poprzedniego stanu działa jak pamięć. Szkic przedstawiający biologiczny neuron znajduje się na rysunku 3.1.



Rysunek 3.1 Szkic biologicznego neuronu

Kora mózgowa u ludzi jest dużym płaskim arkuszem neuronów o grubości około 2 do 3 milimetrów, o powierzchni około 2200 cm^2, czyli około dwa razy więcej niż powierzchnia standardowej klawiatury komputerowej. Kora mózgowa zawiera około 10^{11} neuronów, co w przybliżeniu odpowiada liczbie gwiazd w Drodze Mlecznej [18]. Każdy neuron jest połączony z od 10^3 do 10^4 innych neuronów. W sumie, ludzki mózg zawiera około 10^{14} do 10^{15} połączeń.

Neurony komunikują się poprzez bardzo krótki ciąg impulsów, zwykle trwających pojedyncze milisekundy. Przekazywana informacja jest modulowana na częstotliwości transmisji impulsów. Częstotliwość ta może się wahać od kilku do kilkuset Hz. Złożone decyzje percepcyjne takie jak rozpoznawanie twarzy, są zazwyczaj podejmowane przez ludzi w ciągu kilkuset milisekund. Są one podejmowane przez sieć neuronów, których szybkość działania wynosi tylko kilka milisekund. Oznacza to, że obliczenia nie mogą trwać dłużej niż około 100 etapów szeregowych. Innymi słowy, mózg uruchamia równoległe programy, które mają około 100 kroków dla takich zadań percepcyjnych. Jest to znane jako reguła stu kroków [33]. Te same rozważania dotyczące czasu pokazują, że ilość informacji przesyłanych z jednego neuronu do drugiego innego neuronu musi być bardzo mała (kilka bitów). Wynika z tego, że krytyczna informacja nie jest przesyłana bezpośrednio, ale przechwytywana i dystrybuowana w połączeniach − stąd nazwa, model konekcjonistyczny, wykorzystywany w opisie ANN [18][103].

3.2. Sieci neuronowe pierwszej generacji

Pierwsza generacja sztucznych sieci neuronowych skupia się wokół przeniesienia biologicznego neuronu na płaszczyznę matematyczną i zastosowanie jako pojedynczej jednostki w skomplikowanie strukturze umożliwiającej wykonywanie złożony obliczeń. Niniejszy rozdział opisuje historię powstania sztucznych sieci neuronowych oraz matematyczny model neuronu.

3.2.1. *Krótki rys historyczny*

W badaniach nad sztucznymi sieciami neuronowymi (SNN); ang. Artificial neural networks; ANN) wystąpiły trzy okresy wzmożonej aktywności. Pierwszy szczyt w latach 40. XX wieku zawdzięcza się m.in. McCullocha i Pittsa [69]. Drugi miał miejsce w latach 60. wraz z twierdzeniem Rosenblatta o zbieżności perceptronu [88] oraz pracą Minsky'ego i Paperta pokazującą ograniczenia prostego perceptronu [74]. Wyniki Minsky'ego i Paperta stłumiły entuzjazm większości badaczy. Wynikający z tego zastój w badaniach nad sieciami neuronowymi trwał prawie 20 lat do początku lat 80., po tym okresie sieci neuronowe ponownie spotkały się z dużym zainteresowaniem badaczy [103]. Główne osiągnięcia, które przyczyniły się do odrodzenia tej tematyki dotyczą podejścia energetycznego Hopfielda [44] w 1982 r. oraz algorytm uczenia się metodą wstecznej propagacji dla wielowarstwowych perceptronów, zaproponowany po raz pierwszy przez Werbosa [114]. Algorytm ten był wielokrotnie modyfikowany, a następnie spopularyzowany najpierw przez Werbosa, a następnie przez Rumetharta [93] w 1986 roku. Anderson i Rosenfeld przedstawiają szczegółowy historyczny opis rozwoju ANN [2].

3.2.2. *Matematyczny model neuronu*

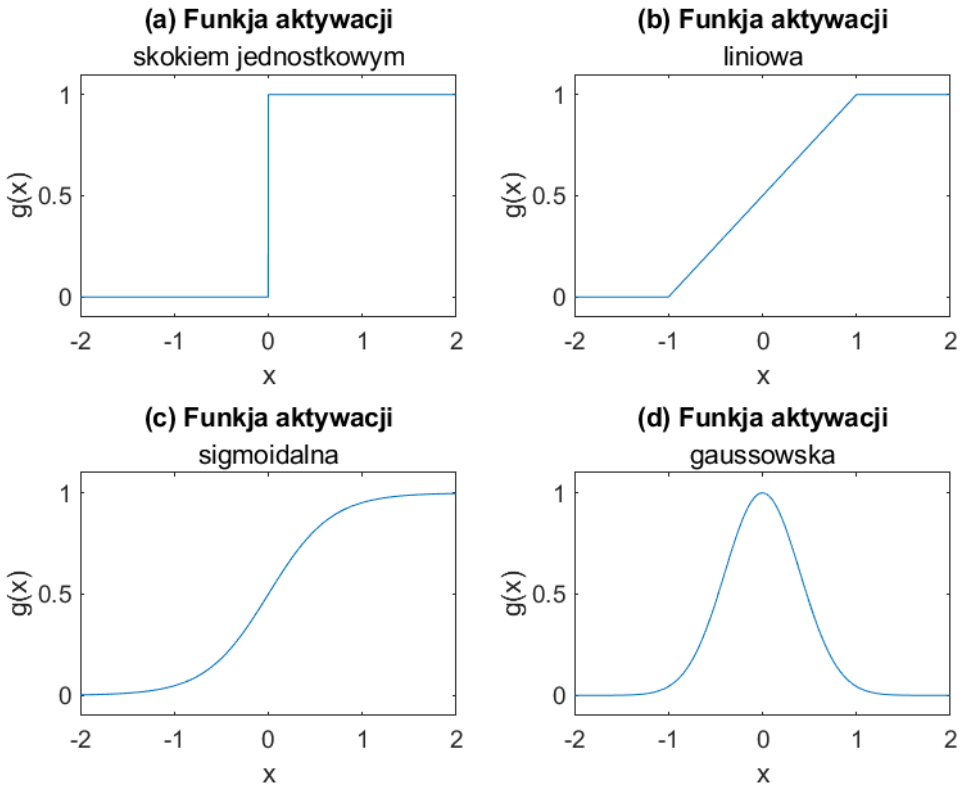
McCulloch i Pitts zaproponowali binarną jednostkę progową jako matematyczny model neuronu. Neuron oblicza ważoną sumę swoich *n* sygnałów wejściowych, $x_j = 1, 2,...,n$ i generuje wyjście równe 1, jeśli suma ta jest powyżej pewnego progu *u*. W przeciwnym razie otrzymuje się wyjście 0. Matematycznie rozpisać neuron rozpisać można funkcją przedstawioną równaniem (3.1), gdzie θ reprezentuje funkcję skoku jednostkowego, a w_j jest wagą synapsy związaną z *j*-tym wejściem [18][33][18]:

$$y = \theta\left(\sum_{j=1}^{n} w_j x_j - u\right) \tag{3.1}$$

Dla uproszczenia notacji próg traktowany jest jako kolejna waga $w_0 = -u$ dołączoną do neuronu ze stałym wejściem $x_0 = 1$. Dodatnie wagi odpowiadają synapsom pobudzającym, natomiast ujemne wagi modelują synapsy hamujące. McCulloch i Pitts udowodnili, że w zasadzie odpowiednio dobrane wagi pozwalają synchronicznemu układowi takich neuronów wykonywać uniwersalne obliczenia. Istnieje prosta analogia do biologicznego neuronu: przewody i połączenia modelują aksony i dendryty, wagi połączeń reprezentują synapsy, a funkcja progowa przybliża aktywność w somie. Model McCullocha-Pittsa zawiera jednak szereg założeń upraszczających, które nie odzwierciedlają prawdziwego zachowania biologicznych neuronów.

Neuron McCullocha-Pittsa został uogólniony na wiele sposobów. Oczywiste jest użycie innych funkcji aktywacji niż funkcja progowa; takich jak liniowa, sigmoidalna czy gaussowska, jak pokazano na rysunku 3.2. Funkcja sigmoidalna jest zdecydowanie najczęściej używana w sztucznych sieciach neuronowych. Jest to funkcja ściśle rosnąca, która wykazuje gładkość i ma pożądane właściwości asymptotyczne. Standardową funkcją sigmoidalną jest funkcja logistyczna, zdefiniowana wzorem (3.2), gdzie β jest parametrem nachylenia.

$$g(x) = \frac{1}{1 + e^{-\beta x}} \tag{3.2}$$



Rysunek 3.2 Przykłady różnych funkcji aktywacji używanych w neuronie

Architektury sieci – podział

Sztuczne sieci neuronowe można postrzegać jako ważone grafy skierowane, w których sztuczne neurony są węzłami, a krawędzie skierowane (z wagami) stanowią połączenia między wyjściami neuronów a wejściami neuronów. Na podstawie budowy i sposobu działania sieci neuronowe można pogrupować w następujący sposób [103]:

- Sieci jednokierunkowe (*feed-forward*), w których grafy nie mają pętli sprzężenia zwrotnego (jednowarstwowe, wielowarstwowe),
- Sieci rekurencyjne (lub ze sprzężeniem zwrotnym), w których pętle występują z powodu połączeń zwrotnych,
- Sieci komórkowe.

W najbardziej powszechnej rodzinie sieci typu *feed-forward*, zwanej perceptronem wielowarstwowym, neurony są zorganizowane w warstwy, które mają między sobą jednokierunkowe połączenia. W ogólności, sieci typu *feed-forward* są statyczne, to znaczy, że produkują tylko jeden zestaw wartości wyjściowych, a nie sekwencję wartości z danego wejścia. Sieci typu *feed-forward* są pozbawione pamięci w tym sensie, że ich odpowiedź na wejście jest

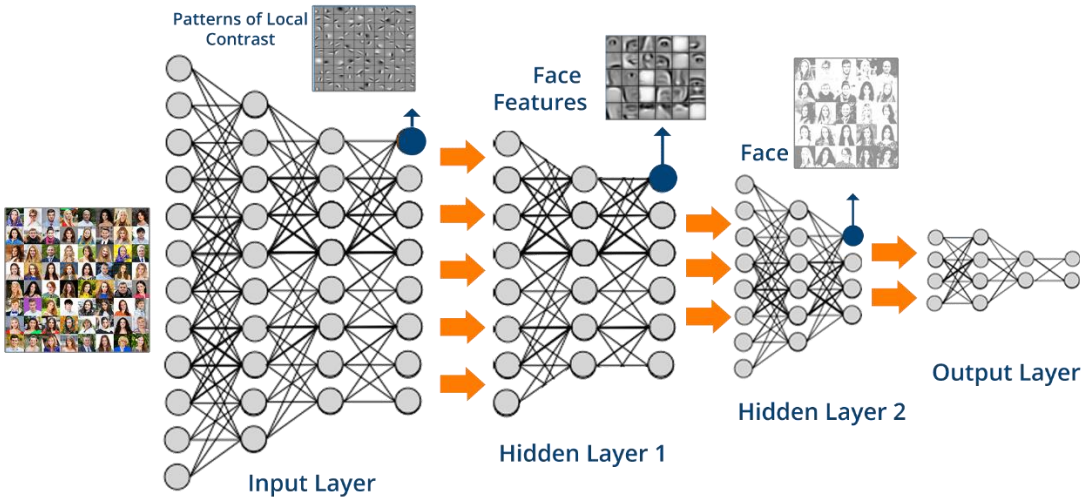
niezależna od poprzedniego stanu sieci. Z kolei, sieci rekurencyjne lub ze sprzężeniem zwrotnym można traktować jako systemy dynamiczne. Kiedy pojawia się nowy wzorzec wejściowy, obliczane są wyjścia neuronów. Ze względu na ścieżki sprzężenia zwrotnego, wejścia do każdego neuronu są następnie modyfikowane, co prowadzi do wejścia sieci w nowy stan.

3.3. Sieci neuronowe 2. generacji − sieci głębokie

Warto zauważyć, że wciąż dominującym nurtem w dziedzinie sztucznych sieci neuronowych są sieci 2. generacji. Ten typ sieci zostanie omówiony w kolejnych podrozdziałach.

3.3.1. *Zasada działania modeli głębokich*

Modele głębokich sieci neuronowych uczą się, odkrywając konkretne struktury w danych zadanych im w trakcie trwania eksperymentu. Aby to osiągnąć, buduje się sieci składające się z wielu warstw przetwarzających, a których każda reprezentuje kolejny poziom abstrakcji. Przykład budowy sieci zaprogramowanej do rozpoznawania twarzy zaprezentowano na rysunku 3.3. Przedstawiono na nim, w jaki sposób kolejne warstwy realizują różne aspekty analizy obrazów.



Rysunek 3.3 Schemat działania głębokiej sieci neuronowej [142]

Uczenie głębokie różni się zasadniczo od konwencjonalnego uczenia maszynowego. Ekspert w danym obszarze musiałby mieć odpowiednie umiejętności w celu zaprojektowania konwencjonalnego systemu uczenia maszynowego. W przypadku uczenia głębokiego wystarczy dostarczyć systemowi bardzo dużą ilość danych, np. zdjęćw kontekście rozpoznawania twarzy, a system może autonomicznie nauczyć się cech, które reprezentują twarz.

W wielu zadaniach, takich jak widzenie komputerowe, rozpoznawanie, tłumaczenie maszynowe i robotyka, wydajność systemów głębokiego uczenia znacznie przewyższa wydajność konwencjonalnych systemów uczenia maszynowego. Nie oznacza to, że budowanie systemów głębokiego uczenia jest stosunkowo łatwiejsze w porównaniu z konwencjonalnymi

systemami uczenia maszynowego. Chociaż rozpoznawanie cech jest autonomiczne w głębokim uczeniu, jednocześnie tysiące hiperparametrów muszą być dostrojone, aby model głębokiego uczenia stał się efektywny. Wymaga to zarówno dużych zasobów obliczeniowych, jak i odpowiedniej ilości danych treningowych [143].

3.3.2. *Przykładowe narzędzia wykorzystywane w treningu głębokich sieci neuronowych*

TensorFlow

TensorFlow jest narzędziem opracowanym przez Google w kontekście uruchamiana zadań związanych z uczeniem maszynowym, uczeniem głębokim czy analizą statystyczną i predykcją. Oprogramowanie zostało stworzone przez zespół Google Brain Team i udostępnione na zasadach kodu otwartego [144].

Oprogramowanie TensorFlow obsługuje zestawy danych, które są ułożone w formie grafu jako węzły obliczeniowe. Krawędzie łączące węzły w grafie mogą reprezentować wielowymiarowe wektory lub matryce, tworząc tzw. tensory. Ponieważ programy TensorFlow używają architektury przepływu danych, która pracuje z uogólnionymi wynikami pośrednimi obliczeń, dlatego szczególnie dobrze nadają się do pracy z bardzo aplikacjami przetwarzania równoległego, takich jak na przykład sieci neuronowe [145].

Narzędzie zawiera zestawy zarówno wysokopoziomowych, jak i niskopoziomowych interfejsów (API). Gdy tylko jest to możliwe, Google zaleca używanie rozwiązań wysokiego poziomu, aby uprościć rozwój i programowanie aplikacji. Wiedza o tym, jak korzystać z niskopoziomowych API − zwanych *TensorFlow Core* − może być cenna dla eksperymentowania i inspekcji (debugowania) aplikacji.

Aplikacje TensorFlow mogą być uruchamiane na konwencjonalnych procesorach (CPU) lub wydajnych procesorach graficznych (GPU), a także na własnych jednostkach przetwarzania tensorowego (TPU) firmy Google, które są niestandardowymi urządzeniami zaprojektowanymi specjalnie w celu przyspieszenia zadań TensorFlow. Pierwsze jednostki TPU firmy Google, zaprezentowane publicznie w 2016 roku, były wykorzystywane wewnętrznie w połączeniu z TensorFlow do uruchamiania niektórych aplikacji i usług zdalnych firmy, w tym jej algorytmu wyszukiwania RankBrain i technologii mapowania Street View [146][147].

Tensorflow Lite

TensorFlow Lite to zestaw narzędzi, które umożliwiają uczenie maszynowe na bezpośrednio na docelowym urządzeniu, pomagając programistom uruchamiać ich modele na urządzeniach mobilnych, czy systemach wbudowanych i brzegowych [148].

Twórcy kierowali się kilkoma głównymi założeniami, tworząc to narzędzie [148]:

1. Zoptymalizowane jest w kontekście uczenia maszynowego na urządzeniach, poprzez uwzględnienie pięciu kluczowych ograniczeń: opóźnienia (nie ma wymiany danych w obie strony z serwerem), prywatności (żadne dane osobowe

nie opuszczają urządzenia), łączności (łączność z Internetem nie jest wymagana), rozmiaru (zmniejszony rozmiar modelu i pliku binarnego) oraz zużycia energii (wydajne wnioskowanie i brak połączeń sieciowych).

2. Obsługa wielu platform, w tym urządzeń z systemami Android i iOS, wbudowanych systemów Linux i mikrokontrolerów.
3. Obsługa różnych języków, w tym Java, Swift, Objective-C, C++ i Python.
4. Wysoka wydajność dzięki akceleracji sprzętowej i optymalizacji modeli.
5. Kompleksowe przykłady dla typowych zadań uczenia maszynowego, takich jak klasyfikacja obrazów, wykrywanie obiektów, szacowanie pozycji, odpowiadanie na pytania, klasyfikacja tekstu itp. na wielu platformach.

PyTorch

PyTorch został opracowany przez zespół Facebook AI, należący aktualnie do korporacji Meta w kontekście algorytmów rozumienia mowy naturalnej. Podobnie jak TensorFlow został udostępniony przez autorów na licencji otwartego oprogramowania, ale w dalszym ciągu twórcy odpowiadają za większość zawartości repozytorium.

Główną filozofią przyświecającą twórcom było zapewnienie użytkownikom elastyczności w pracy z udostępnionym narzędziem, przez co programowanie z użyciem PyTorcha jest bardzo zbliżone do natywnego Pythona. Pozwala to w łatwy sposób dokonać inspekcji (debbugowania) kodu i weryfikować jego poszczególne komponenty, ale kosztem pewnych niestabilności mogących pojawić się w kolejnych etapach treningu modeli.

Aktualnie wielu naukowców rozpoczynających swoją karierę z uczeniem maszynowym sięga właśnie po PyTorch'a ze względu na:

- Łatwe w użyciu API.
- Wsparcie Pythona − jak wspomniano powyżej, PyTorch płynnie integruje się z pythonową filozofią pisania kodu oraz jest bardzo podobny do *numpy*.
- Dynamiczne wykresy obliczeniowe − zamiast predefiniowanych wykresów z określonymi funkcjonalnościami, PyTorch zapewnia ramy, dzięki którym możliwe jest budowanie wykresów obliczeniowych w miarę upływu czasu, a nawet ich zmianę w trakcie pracy algorytmu.

Keras

Keras jest prostym interfejsem, mającym za zadanie udostępnić skomplikowane biblioteki obliczeniowe i ich możliwości w przystępnej formie. Podobnie jak opisane w poprzednich rozdziałach, TensorFlow i PyTorch został napisany w języku Python i udostępniony na licencji otwartego oprogramowania. Pierwotnie możliwe było wybranie biblioteki, nad którą pracował

Keras z CNTK, TensorFlow, PlaidML i Theano. Od wersji 2.4 wprowadzona została jednak zmiana pozwalająca na wykorzystanie jedynie pierwszego z wymienionych narzędzi.

Keras jest narzędziem modułowym, przyjaznym dla użytkownika i rozszerzalnym. Nie obsługuje obliczeń niskiego poziomu, natomiast przekazuje je do innej biblioteki zwanej w repozytorium *Backend*.

Porównanie narzędzi TensorFlow, PyTroch i Keras

Poniższa tabela (tabela 3.1) prezentuje porównanie głównych cech każdego z opisanych powyżej narzędzi do treningu i inferencji algorytmów uczenia maszynowego. W zależności od potrzeb każdy z naukowców pracujących przy zagadnieniach związanych z sztuczną inteligencją może dobrać odpowiednie dla siebie narzędzie [149].

Tabela 3.1 Porównanie głównych cech różniących TensorFlow, Pytorch i Keras

	TensorFlow	*Pytorch*	*Keras*
Poziom API	Wysoki i niski	Niski	Wysoki
Architektura	Nie jest łatwa do użycia	Złożona, trudna w odbiorze	Prosta, konkretna, łatwa w odbiorze
Debbugowanie	Trudne do uruchomienia	Łatwe	Trudne na etapie treningu, jednak ze względu na uproszczone API biblioteka bardzo skrupulatnie weryfikuje trening przed uruchomieniem i zwraca dokładne opisy błędów
Próg wejścia	Wysoki	Średni	Niski
Wydajność	Wysoka	Wysoka	Średnia
Język implementacji	C++, CUDA, Python	Lua	Python

3.4. Sieci neuronowe 3. generacji − SNN (Impulsowe Sieci Neuronowe)

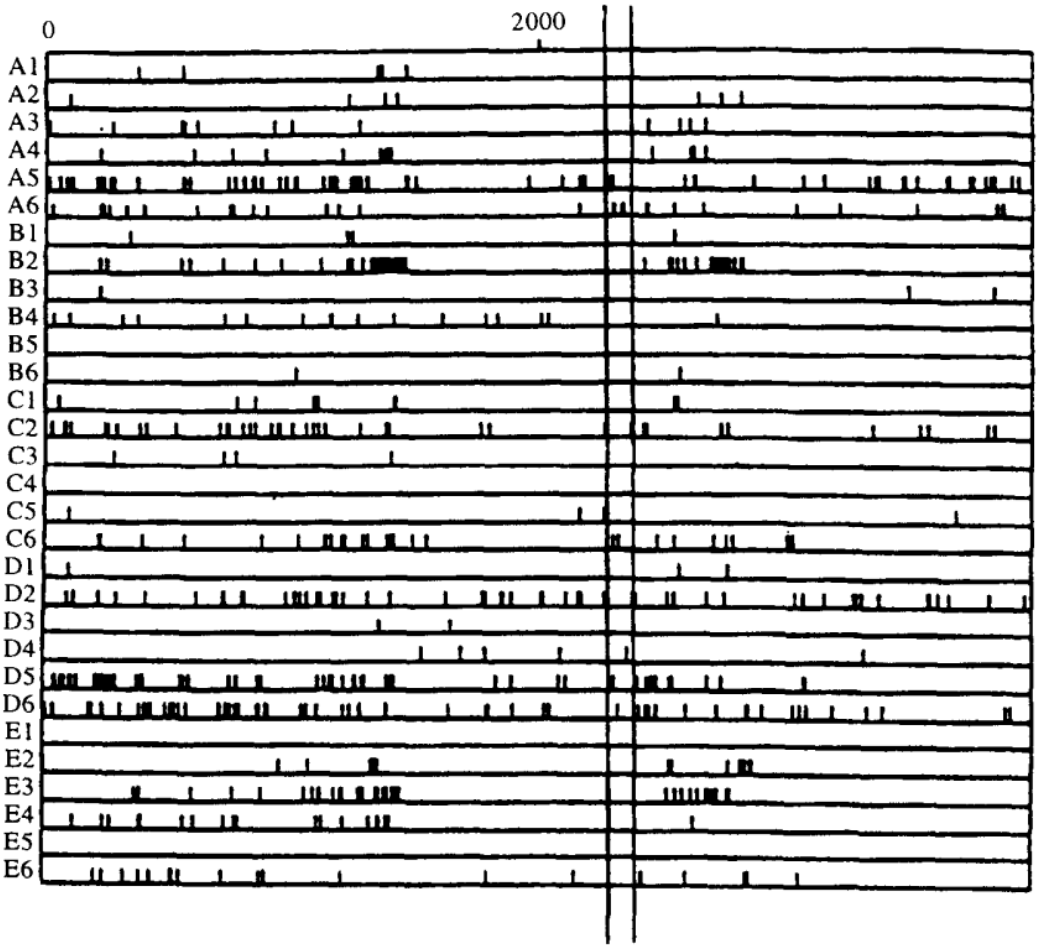
Powrotem do klasycznej koncepcji sieci neuronowych z jednoczesnym wykorzystaniem wiedzy zdobytej przez lata wykorzystywania sieci głębokich mają być Impulsowe Sieci Neuronowe. Poniższy rozdział skupia się na ich genezie oraz zasadach działania.

3.4.1. *Geneza impulsowych sieci neuronowych*

Wyjście sigmoidalnej funkcji aktywacji w sieci neuronowej drugiej generacji może być postrzegane jako reprezentacja aktualnego poziomu aktywacji neuronu biologicznego. Ponieważ wiadomo, że neurony biologiczne, zwłaszcza w wyższych obszarach korowych, aktywują się z różnymi częstotliwościami pośrednimi pomiędzy ich minimalną i maksymalną częstotliwością,

sieci neuronowe drugiej generacji są, w odniesieniu do tej "interpretacji szybkości aktywacji", biologicznie bardziej realistyczne niż modele pierwszej generacji.

Jednakże, przynajmniej w odniesieniu do szybkich obliczeń analogowych wykonywanych przez sieci neuronów w korze mózgowej, sama "interpretacja częstotliwości aktywacji" stała się wątpliwa. Perrett, Rolls i Caan (1982) oraz Thorpe i Imbert (1989) [80,][106] wykazali, że analiza wzorów wzrokowych i klasyfikacja wzorów może być przeprowadzona przez ludzi w ciągu zaledwie 100 milisekund, pomimo faktu, że wymaga minimum 10 etapów synaptycznych od siatkówki do płata skroniowego (patrz rysunek 3.4) Taka sama szybkość przetwarzania wzrokowego została zmierzona przez Rollsa i Tovee'ego (1994) [85] u makaków. Ponadto wykazali oni, że pojedynczy obszar korowy zaangażowany w przetwarzanie wzrokowe może zakończyć swoje obliczenia w ciągu zaledwie 20-30 milisekund. Z drugiej strony, częstotliwość aktywacji neuronów biorących udział w tych obliczeniach jest zwykle niższa niż 100 Hz, a zatem potrzeba co najmniej 20-30 milisekund, aby uzyskać aktualną częstotliwość aktywacji neuronu. Tak więc kodowanie zmiennych analogowych przez częstotliwość aktywacji wydaje się dość wątpliwe w kontekście szybkich obliczeń korowych.



Rysunek 3.4 Równoczesne zapisy (4 sekundy) czasów aktywacji 30 neuronów z kory czołowej małpy, wykonane przez Krügera i Aiple'a (1988). Każda aktywacja jest oznaczona krótkim pionowym paskiem, z osobnym rzędem dla każdego neuronu. Dla porównania zaznaczono długość interwału 100 milisekund dwoma pionowymi liniami [62]

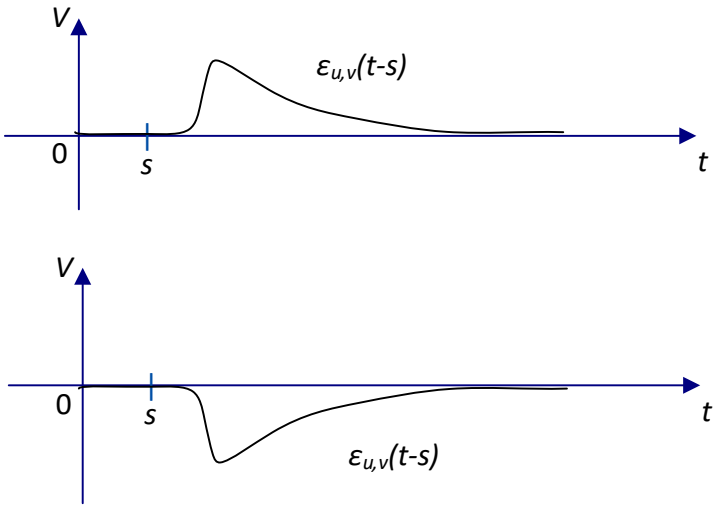
Eksperymentalne wyniki przywołane z neurobiologii doprowadziły do powstania trzeciej generacji modeli sieci neuronowych, które wykorzystują neurony impulsowe jako jednostki obliczeniowe.

Można również postrzegać sieci neuronowe pierwszej generacji jako abstrakcyjne modele obliczeń cyfrowych na sieciach neuronów impulsowych, gdzie bit „1" jest kodowany przez aktywację neuronu w pewnym krótkim oknie czasowym, a „0" przez brak aktywacji tego neuronu w tym oknie czasowym [111]. Jednakże, zgodnie z tym schematem kodowania, sieć neuronowa zapewnia dość dobry model dla sieci neuronów impulsowych tylko wtedy, gdy czasy aktywacji wszystkich neuronów dostarczających bity wejściowe dla innego neuronu impulsowego są zsynchronizowane (do kilku milisekund). Znajduje to odniesienie do biologicznych systemów neuronowych, gdzie występuje tak silnie zsynchronizowana aktywność [1][6].

3.4.2. *Model neuronu impulsowego w oparciu o model neuronu biologicznego*

Modele matematyczne dla neuronów impulsowych sięgają prac Lapique'a (1907) [107]. Istnieje wiele wariantów tego modelu, które zostały opisane i porównane w niedawnym przeglądzie [34]. Te modele matematyczne dla neuronów impulsowych nie zapewniają pełnego opisu niezwykle złożonej funkcji obliczeniowej biologicznego neuronu. Podobnie jak jednostki obliczeniowe poprzednich dwóch generacji modeli sieci neuronowych, są to uproszczone modele, które skupiają się na kilku aspektach działania neuronów biologicznych. Jednak w porównaniu z poprzednimi dwoma modelami są one znacznie bardziej realistyczne. W szczególności, znacznie lepiej opisują rzeczywiste wyjście biologicznego neuronu, a co za tym idzie, pozwalają na zbadanie na poziomie teoretycznym możliwości wykorzystania czasu jako zasobu do obliczeń i komunikacji. Podczas gdy czas kroków obliczeniowych jest zwykle "trywializowany" w modelach z poprzednich dwóch generacji (albo przez założoną synchronizację, albo przez założoną stochastyczną asynchroniczność), czas poszczególnych kroków obliczeniowych odgrywa kluczową rolę w obliczeniach w sieciach neuronów impulsowych.

W najprostszym (deterministycznym) modelu neuronu impulsowego zakłada się, że neuron aktywuje się zawsze wtedy, gdy jego potencjał P_v (który modeluje elektryczny potencjał membranowy w "strefie wyzwalania" neuronu v) osiągnie pewien próg Θ_v. Ten potencjał P_v jest sumą tak zwanych pobudzających potencjałów postsynaptycznych (EPSP, ang. excitatory postsynaptic potential) i hamujących potencjałów postsynaptycznych (IPSP, ang. inhibitory postsynaptic potential), które wynikają z odpalenia innych neuronów u, które są połączone przez "synapsę" z neuronem v. Aktywacja "presynaptycznego" neuronu u w czasie s przyczynia się do potencjału P_v w czasie t w ilości, która jest modelowana przez termin $w_{u,v} \cdot \varepsilon_{u,v}(t-s)$, który składa się z wagi $w_{u,v} \geq 0$ i funkcji aktywacji $\varepsilon_{u,v}(t-s)$. Biologicznie realistyczne kształty takich funkcji odpowiedzi zaznaczono na rysunku 3.5.



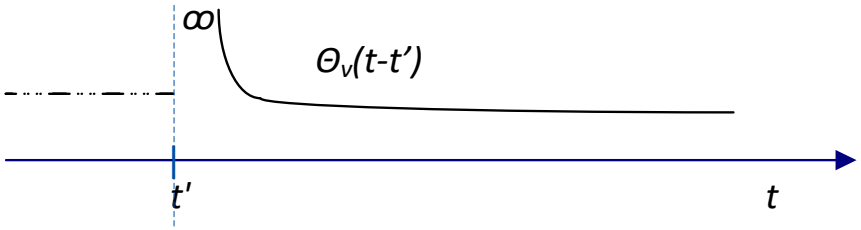
Rysunek 3.5 Typowy kształt funkcji aktywacji naturalnego neuronu (EPSP i IPSP) [62]

"Waga" $w_{u,v} \geq 0$ w określeniu $w_{u,v} \cdot \varepsilon_{u,v}(t-s)$ odzwierciedla siłę (zwaną w neurobiologii skutecznością) synapsy pomiędzy neuronem *u* i neuronem *v*. W kontekście uczenia się można zastąpić $w_{u,v}$ funkcją $w_{u,v}(t)$.

Ograniczenie $w_{u,v}$ do wartości nieujemnych jest motywowane założeniem, że biologiczna synapsa jest albo pobudzająca albo hamująca oraz że nie zmienia ona swojego znaku w trakcie procesu uczenia się. Ponadto, dla większości biologicznych neuronów *u*, albo wszystkie funkcje odpowiedzi $\varepsilon_{u,v}(t-s)$ dla neuronów postsynaptycznych *v* są pobudzające (tj. dodatnie), albo wszystkie są hamujące (tj. ujemne).

Matematycznie wygodniej jest założyć, że potencjał P_v ma wartość 0 przy braku potencjałów postsynaptycznych, a wartość progowa θ_v jest zawsze większa od 0. W typowym neuronie biologicznym spoczynkowy potencjał błonowy wynosi około -70 mV, próg aktywacji spoczynkowego neuronu wynosi około -50 mV, a potencjał postsynaptyczny (tj. EPSP lub IPSP) zmienia przejściowo potencjał błony synaptycznej co najwyżej o kilka mV.

Jeśli neuron *v* aktywował się w czasie *t'*, to nie aktywuje się ponownie przez kilka milisekund po *t'*, niezależnie od tego jak duży jest jego aktualny potencjał $P_v(t)$. Zjawisko to zwane jest absolutnym okresem refrakcji. Następnie przez kilka kolejnych milisekund jest nadal niezdolny do aktywacji, tj. wymaga ona większej wartości potencjału $P_v(t)$ niż zwykle – jest to względny okres refrakcji. Oba te efekty refrakcji są modelowane przez odpowiednią funkcję progową $\theta_v(t-t')$, gdzie *t'* jest czasem ostatniej aktywacji neuronu *v*. W deterministycznej (tj. wolnej od szumów) wersji modelu neuronu impulsowego zakłada się, że neuron *v* aktywuje się zawsze, gdy $P_v(t)$ przecina od dołu funkcję $\theta_v(t-t')$. Typowy kształt funkcji $\theta_v(t-t')$ dla neuronu biologicznego zaznaczono na rysunku 3.6.



Rysunek 3.6 Typowy kształt funkcji progowej dla neuronu biologicznego

3.4.3. *Przykładowe narzędzia wykorzystywane w treningu impulsowych sieci neuronowych*

Model NengoDL

Model NengoDL jest symulatorem dla modeli Nengo (Neural Engineering Object). Oznacza to, że przyjmuje sieć Nengo jako dane wejściowe i pozwala użytkownikowi symulować tę sieć przy użyciu pewnego bazowego narzędzia obliczeniowego (w tym przypadku TensorFlow). Nengo jest sposobem graficznego i skryptowego opisu wielkoskalowych modeli sieci neuronowych przygotowanym na cele symulacyjne.

W praktyce oznacza to, że kod do budowy modelu Nengo jest dokładnie taki sam, jak w przypadku standardowego symulatora Nengo. Zmienia się jedynie to, że do wykonania modelu używa się innej klasy Symulatora [150].

Jednak NengoDL nie jest po prostu duplikatem symulatora Nengo. Dodaje on również szereg unikalnych funkcji, takich jak:

- optymalizacja parametrów modelu poprzez metody treningowe dla sieci głębokich (z wykorzystaniem Keras API);
- większa szybkość symulacji, zarówno na CPU, jak i GPU;
- automatyczna konwersja z modeli Keras do sieci Nengo;
- wstawianie kodu TensorFlow (poszczególnych funkcji lub całych architektur sieci) bezpośrednio do modelu Nengo.

KerasSpiking

KerasSpiking dostarcza narzędzi do trenowania i uruchamiania sieci neuronowych typu impulsowego bezpośrednio w ramach pakietu Keras. Główną cechą jest metoda *keras_spiking.SpikingActivation*, która może być użyta do przekształcenia dowolnej funkcji aktywacji w jej impulsowy odpowiednik, np.:

```
inp = tf.keras.Input((5,))
dense = tf.keras.layers.Dense(10)(inp)
act = tf.keras.layers.Activation("relu")(dense)
model = tf.keras.Model(inp, act)
```

w strukturę zawierającą dodatkowy wymiar czasu i zmieniającą funkcję aktywacji.:

```
inp = tf.keras.Input((None, 5))
dense = tf.keras.layers.Dense(10)(inp)
act = keras_spiking.SpikingActivation("relu")(dense)
model = tf.keras.Model(inp, act)
```

Modele z warstwami *SpikingActivation* mogą być optymalizowane i oceniane w taki sam sposób jak każdy inny model Keras. Będą one automatycznie korzystać z "treningu świadomego impulsowości sieci" w KerasSpiking: używając aktywacji impulsów na w przód i klasycznej (różniczkowalnej) funkcji aktywacji na przejściu do tyłu.

KerasSpiking zawiera również różne narzędzia wspomagające trening modeli impulsowych, takie jak dodatkowe regularyzatory i warstwy filtrujące [151].

Porównanie NengoDL z KerasSpiking

W pierwszej kolejności, porównując ze sobą opisane narzędzia, należy zwrócić uwagę na to, że KerasSpiking został zaprojektowany jako lekka, minimalna implementacja zachowania impulsowego, która integruje się bardzo przejrzyście z Keras. Pakiet został zaprojektowany w taki sposób, aby każdy mógł rozpocząć i uruchomić budowanie modelu impulsowego z bardzo małym narzutem obliczeniowym.

NengoDL zapewnia znacznie bardziej solidne, w pełni funkcjonalne narzędzia do budowania modeli impulsowych. Posiada więcej typów neuronów, więcej typów synaps, bardziej złożone architektury sieci. Jednak wszystkie te dodatkowe funkcje wymagają bardziej znaczącego odejścia od bazowego API TensorFlow/Keras. Ponadto praca z NengoDL jest bardziej wymagająca, a wynikowy kod wygląda mniej jak standardowy kod Keras.

Jedną ze szczególnie istotnych różnic jest to, że KerasSpiking nie integruje się z resztą ekosystemu Nengo (np. nie może uruchamiać modeli zbudowanych za pomocą Nengo API, a modele zbudowane za pomocą KerasSpiking nie mogą działać na innych platformach Nengo). W przeciwieństwie do tego, NengoDL może uruchomić każdy model Nengo, a modele zoptymalizowane w NengoDL mogą być uruchamiane na innych platformach Nengo (takich jak niestandardowy sprzęt neuromorficzny, jak NengoLoihi) [152].

Podsumowanie

W podsumowaniu niniejszego rozdziału warto zauważyć, że najczęściej stosowanymi modelami sieci neuronowych są obecnie [182]:

1) Perceptron wielowarstwowy (Multilayer Perceptrons, MLPs);

2) Sieci radialne (Radial Basis Function Networks, RBFNs);

3) Sieci splotowe (Convolutional Neural Networks, CNN);

4) Sieci rekurencyjne (Recurrent Neural Networks, RNNs);

5) Sieci rekurencyjne z warstwą LSTM (Long Short-Term Memory Networks, LSTMs; długa pamięć krótkoterminowa);

6) Restricted Boltzmann Machines (RBMs);

7) Sieci samoorganizujące (Self-Organizing Maps, SOMs; sieci Kohonena);

8) Generatywne sieci przeciwstawne (Generative Adversarial Networks, GANs);

9) Autoenkodery (Autoencoders Deep Learning Algorithm);

10) Probabilistyczne modele generatywne (Deep Belief Networks; sieci głębokiego przekonania).

4. METODOLOGIA ARCHIWIZOWANIA I PORÓWNYWANIA WYNIKÓW EKSPERYMENTÓW

Ze względu na konieczność uruchamiania wielu różnych treningów przy zmianie niejednokrotnie pojedynczych parametrów, ale również architektur badanych sieci neuronowych, niezbędne jest wykorzystanie narzędzi pozwalających w prosty sposób zapisać oraz porównać wyniki. Przykładem jest platforma MLflow, który pozwala na zapisywanie w trakcie treningu zarówno wyznaczanych metryk, jak i ogólnych parametrów [153]. Ze względu na wykorzystanie tej platformy do zapisu przebiegu eksperymentów przeprowadzonych w ramach niniejszej rozprawie doktorskiej zostanie ona dokładniej omówiona,

4.1. Komponenty oferowane przez platformę MLFlow

Platforma MLflow składa się z czterech komponentów nazwanych: *Tracking, Projects, Models i Model Registry*. Każdy z tych modułów może być wykorzystywany indywidualnie − na przykład, można wyeksportować modele w formacie MLflow bez używania Tracking lub Projects. Jednak z założenia komponenty te są zaprojektowane tak, aby dobrze ze sobą współpracowały.

Podstawową filozofią MLflow jest nakładanie jak najmniejszych ograniczeń na sposób pracy: został on zaprojektowany do współpracy z dowolną biblioteką uczenia maszynowego i wymaga minimalnych zmian w celu zintegrowania z istniejącym kodem. Jednocześnie MLflow dąży do tego, aby każdy kod napisany w jego formacie był powtarzalny i możliwy do wykorzystania przez wielu autorów.

Uczenie maszynowe wymaga eksperymentowania z szeroką gamą zbiorów danych, kroków przygotowania danych i algorytmów w celu zbudowania modelu, który maksymalizuje docelową metrykę. Po zbudowaniu modelu, należy go wdrożyć do systemu produkcyjnego, monitorować jego wydajność, a także stale trenować go na nowych danych i porównywać z alternatywnymi modelami.

Proces śledzenia wydajności uczenia maszynowego staje się zatem wyzwaniem z kilku powodów:

1. Trudno jest śledzić eksperymenty, kiedy opisane są tylko plikami zapisanymi na dysku. Utrudnia to możliwość potwierdzenia, które dane, kod i parametry złożyły się na uzyskanie konkretnego wyniku.
2. Trudno jest odtworzyć kod. Nawet skrupulatne śledzenie wersji kodu oraz użytych parametrów nie pozwala na uchwycenie całego środowiska uruchomieniowego (na przykład zależności bibliotek), aby ponownie uzyskać ten sam wynik.
3. Nie ma standardowego sposobu pakowania i wdrażania modeli. Każdy zespół tworzy własne podejście dla każdej biblioteki uczenia maszynowego, której używa, a związek między modelem a kodem i parametrami, które go wytworzyły, jest często tracony.

4. Nie ma centralnego magazynu do zarządzania modelami (ich wersjami i przejściami między etapami). Zespół tworzy wiele modeli. W przypadku braku centralnego miejsca do współpracy i zarządzania cyklem życia modelu, zespoły stają przed wyzwaniem, jak zarządzać etapami życia modelu: od rozwoju do archiwizacji lub produkcji, z odpowiednimi wersjami, adnotacjami i historią.

Chociaż poszczególne biblioteki do uczenia maszynowego dostarczają rozwiązań dla niektórych z wymienionych problemów (na przykład obsługi modeli), to aby uzyskać najlepszy rezultat, zwykle należy wypróbować wiele podejść. Platforma MLflow pozwala na trenowanie, ponowne wykorzystanie i wdrażanie modeli z dowolną biblioteką i konfigurowanie ich w powtarzalne kroki, które inni naukowcy, zajmujący się danym problemem i danymi mogą zestaw model-biblioteka wykorzystać jako "czarną skrzynkę", nie mając nawet wglądu, która biblioteka została wykorzystana.

Platforma MLflow dostarcza cztery komponenty, które pomagają zarządzać sposobem pracy w uczeniu maszynowym:

1. *Tracking* to API i UI, które służą do rejestrowania parametrów, wersji kodu, metryk i artefaktów podczas uruchamiania kodu uczenia maszynowego oraz do późniejszej wizualizacji wyników. MLflow Tracking można używać w dowolnym środowisku (na przykład jako samodzielny skrypt lub notatnik) do rejestrowania wyników do plików lokalnych lub na serwer, a następnie porównywania wielu przebiegów. Zespoły mogą również używać go do porównywania wyników od różnych użytkowników.

2. Projekty MLflow stanowi standardowy format pakowania kodu wielokrotnego użytku w dziedzinie nauki o danych. Każdy projekt jest katalogiem z kodem lub repozytorium Git i używa pliku deskryptora, aby określić zależności i sposób uruchamiania kodu. Na przykład projekty mogą zawierać plik conda.yaml do określenia środowiska Python Conda. Używając MLflow Tracking API w projekcie, MLflow automatycznie zapamiętuje wersję projektu (na przykład Git commit) i wszystkie parametry. W ten sposób można łatwo uruchomić istniejące projekty MLflow z GitHub lub własnego repozytorium Git i połączyć je w wieloetapowe eksperymenty.

3. Modele MLflow oferują konwencję pakowania modeli uczenia maszynowego, a także szereg narzędzi pomagających w ich wdrażaniu. Każdy model jest zapisywany jako katalog zawierający dowolne pliki oraz plik deskryptora. Na przykład, model TensorFlow może być załadowany jako TensorFlow DAG lub jako funkcja Pythona do zastosowania na danych wejściowych. MLflow dostarcza narzędzi do wdrażania wielu typowych modeli na różne platformy: na przykład każdy model obsługujący funkcjie Python’a może być wdrożony do serwera REST opartego na Dockerze, na platformy chmurowe, takie jak Azure

ML i AWS SageMaker oraz jako funkcja zdefiniowana przez użytkownika w Apache Spark do wnioskowania wsadowego i strumieniowego.

4. *Flow Registry* oferuje scentralizowany magazyn modeli, zestaw API i UI w celu wspólnego zarządzania pełnym cyklem życia modelu MLflow. Dostarcza on informacji o pochodzeniu modelu (który eksperyment MLflow i uruchomienie wytworzyły model), wersjonowaniu modelu, przejściach między etapami (na przykład z etapu inscenizacji do produkcji lub archiwizacji) oraz adnotacjach.

4.2. Sposoby zapisywania plików artefaktów w MLFlow

Określając położenie pliku z artefaktem w API MLflow, składnia zależy od tego, czy odwołuje się do API Śledzenia, Modeli czy Projektów. W przypadku interfejsu API śledzenia lokalizacja artefaktu jest określana za pomocą krotki (identyfikator, ścieżka względna). W przypadku interfejsów API Modele i Projekty lokalizację artefaktu można przedstawić w następujący sposób:

`/Users/me/path/to/local/model`

`relative/path/to/local/model`

`<schemat>/<schemat-zależny-ścieżka>. Na przykład:`

`s3://my_bucket/path/to/model`

`hdfs://<host>:<port>/<ścieżka>`

`run:/<mlflow_run_id>/run-relative/path/to/model`

`models:/<model_name>/<model_version>`

`models:/<model_name>/<stage>`

`mlflow-artifacts:/path/to/model podczas uruchamiania serwera śledzącego w trybie --serve-artifacts proxy.`

Na przykład:

1. Tracking API

```
mlflow.log_artifacts("<mlflow_run_id>", "/path/to/artifact")
```

2. API Modeli

```
mlflow.pytorch.log_model("runs:/<mlflow_run_id>/run-
relative/path/to/model", registered_model_name="mymodel")

mlflow.pytorch.load_model("models:/mymodel/1")
```

4.3. Metody skalowania w MLFlow

Dane stanowią kluczowy element w uzyskaniu satysfakcjonujących wyników w uczeniu maszynowym, dlatego MLflow został zaprojektowany w taki sposób, aby skalować do dużych

zbiorów danych, dużych plików wyjściowych (na przykład modeli) i prowadzenia dużej ilości eksperymentów. W szczególności, MLflow wspiera skalowanie w czterech wymiarach:

1. Pojedynczy eksperyment MLflow może być wykonywany na rozproszonym klastrze, na przykład przy użyciu Apache Spark. MLflow zawiera wbudowane API do uruchamiania przebiegów na Databrickach.
2. MLflow wspiera uruchamianie wielu przebiegów równolegle z różnymi parametrami, na przykład do strojenia hiperparametrów. Można po prostu użyć API *Projekty* do uruchomienia wielu przebiegów i API *Śledzenie* do ich śledzenia.
3. Projekty MLflow mogą pobierać dane wejściowe i zapisywać dane wyjściowe w rozproszonych systemach przechowywania danych, takich jak AWS S3 i DBFS. MLflow może automatycznie pobrać tego typu pliki lokalnie dla projektów, które mogą działać tylko na plikach lokalnych lub podać projektowi URI rozproszonego magazynu, jeśli to obsługuje. Oznacza to, że można pisać projekty, które budują duże zbiory danych.
4. MLflow Model Registry oferuje dużym organizacjom centralny hub do wspólnego zarządzania pełnym cyklem życia modelu. Wiele zespołów w organizacji tworzy setki modeli, a każdy z nich ma swoje eksperymenty, przebiegi, wersje, artefakty i przejścia między etapami. Centralny rejestr ułatwia odkrycie modelu i jego przeznaczenia przez wiele zespołów w dużej organizacji.

4.4. Przykłady zastosowania MLFlow

Poniżej przedstawiono pokrótce filozofię tworzenia i wykorzystania modeli/danych, ich wdrażania, a także sposobu udostępniania.

MLflow można wykorzystać na wiele sposobów, tj. przez indywidualnego naukowca pracującego samodzielnie czy dużą organizację. MLflow Tracking może być używany do śledzenia eksperymentów lokalnie na własnej maszynie, organizowania kodu w projektach do ponownego wykorzystania w przyszłości oraz tworzenia modeli, które inżynierowie produkcyjni mogą następnie wdrożyć za pomocą narzędzi wdrożeniowych. MLflow Tracking domyślnie odczytuje i zapisuje pliki w lokalnym systemie plików, więc nie ma potrzeby wdrażania serwera.

Zespoły mogą wdrożyć serwer MLflow Tracking w celu rejestrowania i porównywania wyników przez wielu użytkowników pracujących nad tym samym problemem. Dzięki ustaleniu konwencji nazywania parametrów i metryk, mogą oni wypróbować różne algorytmy do rozwiązania tego samego problemu, a następnie uruchomić te same algorytmy ponownie na nowych danych, aby porównać modele w przyszłości. Co więcej, każdy może pobrać i uruchomić inny model.

Duże organizacje mogą współdzielić projekty, modele i wyniki za pomocą MLflow. Każdy zespół może uruchomić kod innego zespołu za pomocą MLflow Projects, co pozwala na wykorzystanie danych i modeli oraz porównanie wyników pomiędzy zespołami nad tym samym

zadaniem. Ponadto, zespoły inżynierskie mogą łatwo przenosić procesy z działu badań i rozwoju do produkcji. Inżynierowie produkcyjni mogą w ten sam sposób wdrażać modele z różnych bibliotek ML, przechowywać je w formie plików w wybranym przez siebie systemie zarządzania i śledzić, z którego uruchomienia pochodzi dany model.

Z kolei naukowcy i twórcy oprogramowania Open Source mogą publikować kod na GitHubie w formacie MLflow Project, co ułatwia każdemu uruchomienie ich kodu za pomocą polecenia mlflow run github.com/....

Twórcy bibliotek ML mogą tworzyć modele w formacie MLflow Model, aby automatycznie wspierać ich wdrażanie za pomocą wbudowanych narzędzi MLflow. Ponadto, twórcy narzędzi wdrożeniowych (na przykład dostawca chmury budujący platformę obsługową) mogą automatycznie obsługiwać wiele różnych modeli.

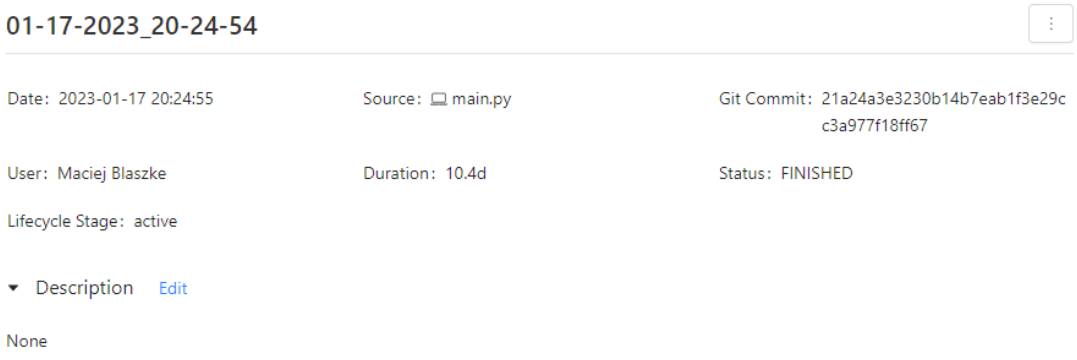
4.5. Opis implementacji logowania artefaktów i parametrów

Zaimplementowanie w eksperymentach automatycznego logowania opisu poszczególnych uruchomień, zarówno z zakresie konfiguracji danego treningu, jak i wyliczonych w trakcie jego trwania metryk, pozwala na proste porównanie uzyskanych wyników, proste odtworzenie każdego z eksperymentów i jednocześnie wybranie najlepszego modelu.

Na potrzeby niniejszej rozprawy doktorskiej wykorzystano jednocześnie domyślnie skonfigurowane automatyczne logowanie, jak i dodano zestaw indywidualnych parametrów i metryk. Kolejne podrozdziały opisują kolejno, jakie elementy opisu treningu zostały zapisane, parametry treningu oraz finalnie metryki.

4.5.1. *Rejestrowany opis treningu*

Podstawowym elementem wyświetlającym się w ekranie eksperymentu MLflow jest sekcja opisu pokazana na rysunku 4.1.



Rysunek 4.1 Sekcja opisu eksperymentu w MLflow

Zaprezentowana sekcja zawiera szereg różnych elementów opisujących eksperyment. Ich znaczenie zostało wyjaśnione w tabeli 4.1.

Tabela 4.1 Opis elementów sekcji opisu eksperymentu w MLflow

Nazwa	***Opis***
Date	Data uruchomienia treningu
Source	Nazwa pliku, z którego został uruchomiony trening
Git Commit	Wartość sumy kontrolnej SHA dla aktualnego stanu repozytorium w systemie kontroli wersji
User	Nazwa użytkownika, który uruchomił trening
Duration	Długość treningu
Status	Status treningu

4.5.2. *Rejestrowane parametry*

Kolejną sekcją w eksperymencie MLflow są parametry. Cześć z nich definiowana jest automatycznie, ale użytkownik może je również dodawać indywidualne. Przykładem takiego parametru może być „dataset_config”. Dokładny opis parametrów znajduje się w tabeli 4.2.

Tabela 4.2 Opis elementów sekcji parametrów eksperymentu w MLflow

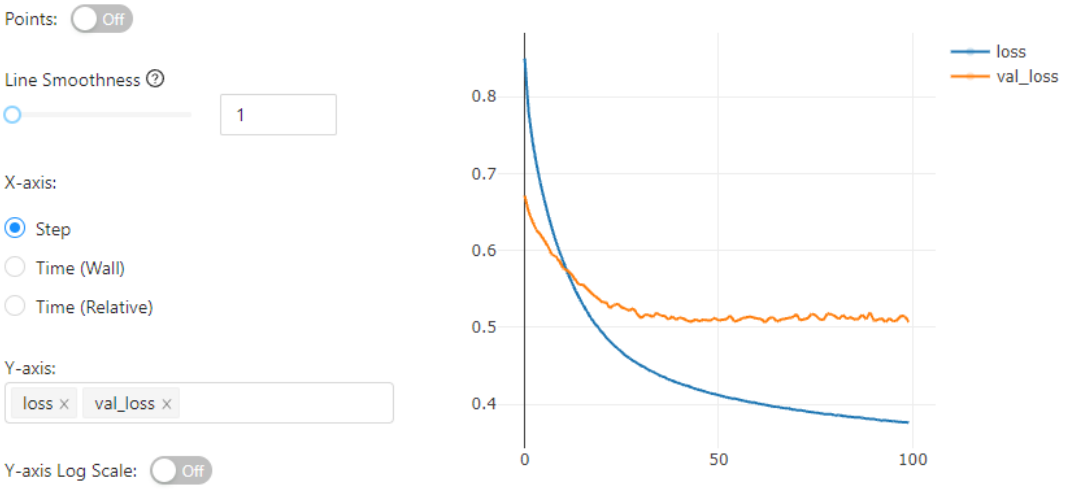
Nazwa parametru	***Opis parametru***
class_weight	Wektor „ważący” klasy w zbiorze treningowym
dataset_config	Ścieżka do pliku konfiguracyjnego opisującego zbiór danych wykorzystany w treningu
epochs	Maksymalna liczba epok w treningu
initial_epoch	Pierwsza epoka, z której trening był uruchamiany (opisuje wznowienie już istniejącego treningu)
max_queue_size	Maksymalna wielkość kolejki zbuforowanych danych
min_delta	Minimalna wartość zmiany monitorowanego parametru, aby uznać wcześniejsze przerwanie treningu
monitor	Metryka monitorowana w kontekście wcześniejszego przerwania treningu
opt_decay	Opóźnienie zmiany współczynnika uczenia dla optymalizatora
opt_epsilon	Mała wartość zmiennoprzecinkowa używana do utrzymania stabilności numerycznej

opt_learning_rate	Wartość początkowa współczynnika uczenia się
opt_name	Nazwa optymalizatora
opt_rho	Wartość szybkości zmiany współczynnika uczenia się
patience	Liczba epok, w których monitorowana jest zmiana parametru monitorowanego w kontekście wcześniejszego przerwania treningu
prepareModel_module	Nazwa modułu wykorzystanego do przygotowania modelu sieci neutronowej
restore_best_weights	Flaga oznaczająca przywrócenie jako wynikowego modelu najlepszej epoki zgodnie z monitorowanym parametrem
sample_weight	Wagi dla każdej próbki do zastosowania przy obliczaniu metryki wydajności modelu
shuffle	Flaga oznaczająca losowanie kolejności elementów w zbiorze po zakończonej epoce treningu
steps_per_epoch	Maksymalna liczba kroków na epokę
use_multiprocessing	Flaga oznaczająca wykorzystanie wielu wątków procesora
validation_freq	Częstotliwość walidacji modelu w odniesieniu do epok treningowych
validation_split	Stosunek wielkości podzbioru walidacyjnego do treningowego – w przypadku, gdy nie jest ten podział ustalany z góry
workers	Liczba procesów uruchomionych na procesorze w ramach treningu

4.5.1. *Rejestrowane metryki*

Podczas treningu sieci neuronowej możliwe jest jednoczesne monitorowanie wielu różnych metryk. W MLflow są one zbierane w postaci tabeli, wyświetlając ostatnią zapamiętaną wartość, ale jednocześnie możliwe jest wyświetlenie całej historii wartości dla dowolnej z nich. Dodatkowo MLflow umożliwia zestawienie wielu różnych metryk na jednym wykresie.

Przykładowy widok przedstawiający porównanie funkcji kosztu na zbiorze treningowym i walidacyjnym został przedstawiony na rysunku 4.2. Istnieje możliwość wyświetlenia punktów, włączenia wygładzania linii, podania opisu osi rzędnych oraz włączenia logarytmicznej skali osi odciętych.



Rysunek 4.2 Wizualizacja funkcji kosztu na zbiorze treningowym i ewaluacyjnym w MLflow

4.6. Wybrane miary oceny jakości klasyfikacji

W procesie oceny jakości klasyfikacji bierze się m.in. pod uwagę liczbę obserwacji poprawnie/niepoprawnie zaklasyfikowanych do klasy pozytywnej/negatywnej. Są to odpowiednio wskaźniki: TP (ang. *True Positive*) – liczba obserwacji poprawnie zaklasyfikowanych do klasy pozytywnej TN (ang. *True Negative*) – liczba obserwacji poprawnie zaklasyfikowanych do klasy negatywnej FP (ang. *False Positive*) – liczba obserwacji zaklasyfikowanych do klasy pozytywnej podczas, gdy w rzeczywistości pochodzą z klasy negatywnej FN (ang. *False Negative*) – liczba obserwacji zaklasyfikowanych do klasy negatywnej podczas, gdy w *rzeczywistości* pochodzą z klasy pozytywnej. Określa się też zdolność klasyfikatora do wykrywania klasy pozytywnej/negatywnej, itp. Wybrane miary zostaną podane w kolejnych podrozdziałach.

4.6.1. *Precyzja*

Precyzja (ang. *Precision*) to stosunek przykładów prawdziwie pozytywnych do wszystkich przykładów zidentyfikowanych jako badana klasa. Definicja tej miary została przedstawiona w równaniu (4.1).

$$precyzja = \frac{Prawdziwie\ pozytywne}{Prawdziwie\ pozytywne + Fałszywie\ pozytywne} \tag{4.1}$$

4.6.1. *Czułość*

Czułość (ang. *Sensitivity*) to stosunek przykładów prawdziwie pozytywnych do wszystkich przykładów w badanej klasie, definiowana jest wzorem (4.2):

$$czułość = \frac{Prawdziwie\ Pozytywne}{Prawdziwie\ pozytywne + Fałszywie\ negatywne} \tag{4.2}$$

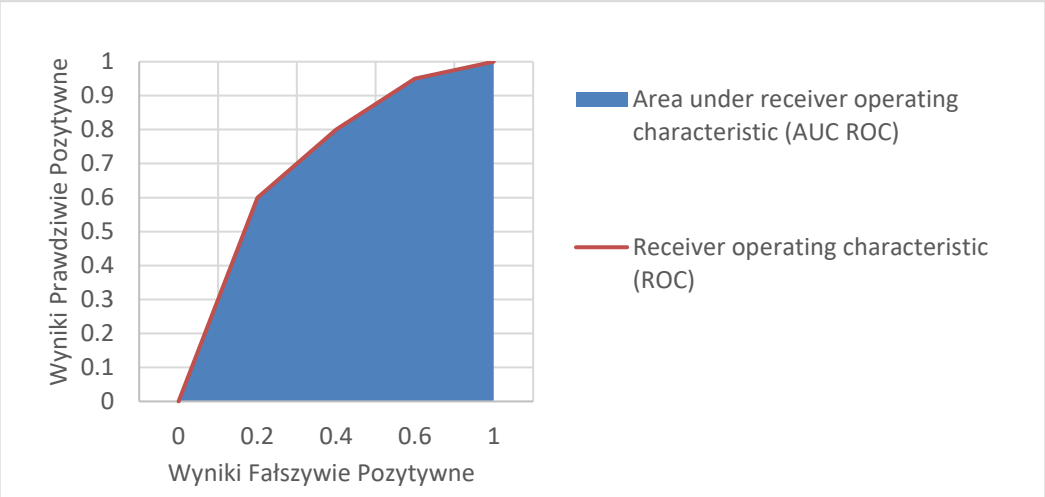
4.6.2. *Miara F1*

Miara F1 (ang. F1 *score*) [154] stanowi średnią harmoniczną precyzji i czułości. Dokładna definicja przedstawiona jest w równaniu (4.3) [52].

$$F1 = 2 \cdot \frac{precyzja \cdot czułość}{precyzja + czułość} \tag{4.3}$$

4.6.3. *AUC ROC*

Krzywa ROC (ang. *Receiver Operating Characteristic*) ilustruje związek pomiędzy czułością a specyficznością (ang. *specificity*) dla danego modelu, tj. kompromis pomiędzy wynikami prawdziwie i fałszywie pozytywnymi w funkcji różnych progów decyzyjnych. ROC i AUC ROC (ang. *Area Under Curve*) są zilustrowane na rysunku 4.3. Ilustracja z rys. 4.3 stanowi wizualizację znaczenia wyników prawdziwie i fałszywie pozytywnych w procesie identyfikacji.



Rysunek 4.3 Krzywa ROC i pole pod krzywą

4.6.4. *LRAP*

Średnia precyzja rankingu etykiet (ang. *Label ranking average precision*, LRAP) wyznacza średnią precyzję modelu wykonującego identyfikację wielu klas. LRAP zasadniczo zadaje pytanie dla każdej z podanych próbek, jaki procent etykiet o wyższej randze odpowiadał prawdziwej (faktycznej) etykiecie [155]. Należy pamiętać, że miara LRAP faworyzuje lepszą rangę etykiet powiązanych z każdą próbką, dlatego jest właśnie wykorzystywana w problemach rankingowych z wieloma etykietami.

Biorąc pod uwagę binarną macierz wskaźników etykiet prawdy $y \epsilon \{0,1\}^{n_{przykłady} * n_{klasy}}$, wynik połączony z każdą klasą oznaczony jest jako $\hat{f} \epsilon \{\mathbb{R}\}^{n_{przykłady} * n_{klasy}}$.

Sama miara wyznaczana jest zgodnie z równaniem 4.4.

$$LRAP(y,\hat{f}) = \frac{1}{n_{przykłady}} * \sum_{i=0}^{n_{przykłady}} \frac{1}{\|y_i\|_0} \sum_{j:y_{ij}=1} \frac{|L_{ij}|}{ranga_{ij}}, \text{ gdzie}$$
$$L_{ij} = \{k: y_{ik} = 1, \hat{f}_{ik} \geq \hat{f}_{ij}\} \text{ i} \quad (4.4)$$
$$ranga_{ij} = |\{k: \hat{f}_{ik} \geq \hat{f}_{ij}\}|$$

4.6.5. *Dokładność*

Dokładność (ang. *accuracy*) opisuje liczbę próbek, które zostały prawidłowo zaklasyfikowane. Definicja tej miary została przedstawiona w równaniu (4.5) [156].

$$dokładność = \frac{TP + TN}{TP + TN + FP + FN} \quad (4.5)$$

5. ZBIORY UCZĄCE I TESTOWE DLA BADANYCH ALGORYTMÓW

Sztuczne sieci neuronowe do prawidłowego działania potrzebują w pierwszej kolejności przejść przez etap treningu, którego bazą są zbiory uczące i testowe. Niniejszy rozdział opisuje zarówno dane źródłowe do przygotowania właściwych zbiorów, jak i właściwe zbiory. W pierwszej kolejności zostanie krótko przywołany format danych muzycznych oraz bazy muzyczne, wykorzystywane w prowadzonych badaniach.

5.1. Interpretacja zapisu nutowego w formacie MIDI

Celem wprowadzania formatu MIDI było łatwe opisanie sposobu wymiany danych oznaczonych znacznikiem czasu między różnymi programami na tym samym lub różnych komputerach [157]. Jednym z głównych celów projektowych jest zwarta reprezentacja, co czyni ją bardzo odpowiednią dla formatu plików na dysku, ale może sprawić, że będzie nieodpowiednia do przechowywania w pamięci w celu szybkiego dostępu przez program sekwencyjny.

Pliki MIDI zawierają jeden lub więcej strumieni MIDI z informacją o czasie dla każdego zdarzenia. Obsługiwane są struktury utworów, sekwencji i ścieżek, informacje o tempie i metrum. Nazwy ścieżek i inne informacje opisowe mogą być przechowywane z danymi MIDI. Ten format obsługuje wiele ścieżek i wiele sekwencji, więc jeśli użytkownik programu obsługującego wiele ścieżek zamierza przenieść plik do innego, ten format może na to pozwolić.

Specyfikacja definiuje 8-bitowy strumień danych binarnych używany w pliku. Dane mogą być przechowywane w pliku binarnym, wyrównanym co do długości symboli (nibbilizowanym), 7-bitowym dla wydajnej transmisji MIDI, konwertowanym na Hex ASCII lub tłumaczonym symbolicznie na plik tekstowy do druku. Ta specyfikacja dotyczy tego, co znajduje się w strumieniu 8-bitowym, ale nie odnosi się do tego, w jaki sposób plik MIDI będzie przesyłany pomiędzy urządzeniami.

Na rysunku 5.1 przedstawiono rozkład dźwięków na klawiaturze fortepianowej w zestawieniu z oznaczeniem dźwięku z notacji muzycznej oraz oznaczeniem w notacji MIDI.

Nazwa Oktawy	Oznaczenie dźwięku w notacji muzycznej	Oznaczenie dźwięku z notacji MIDI z oznaczeniem na klawiaturze fortepianowej
Oktawa subkontra	A_2 B_2	A0 A#0 B0
Oktawa kontra	C_1 D_1 E_1 F_1 G_1 A_1 B_1	C1 C#1 D1 D#1 E1 F1 F#1 G1 G#1 A1 A#1 B1
Oktawa wielka	C D E F G A B	C2 C#2 D2 D#2 E2 F2 F#2 G2 G#2 A2 A#2 B2
Oktawa mała	c d e f g a b	C3 C#3 D3 D#3 E3 F3 F#3 G3 G#3 A3 A#3 B3
Oktawa razkreślna	c^1 d^1 e^1 f^1 g^1 a^1 b^1	C4 C#4 D4 D#4 E4 F4 F#4 G4 G#4 A4 A#4 B4
Oktawa dwukreślna	c^2 d^2 e^2 f^2 g^2 a^2 b^2	C5 C#5 D5 D#5 E5 F5 F#5 G5 G#5 A4 A#5 B5
Oktawa trzykreślna	c^3 d^3 e^3 f^3 g^3 a^3 b^3	C6 C#6 D6 D#6 E6 F6 F#6 G6 G#6 A6 A#6 B6
Oktawa czterokreślna	c^4 d^4 e^4 f^4 g^4 a^4 b^4	C7 C#7 D7 D#7 E7 F7 F#7 G7 G#7 A7 A#7 B7
	c^5	C8

Rysunek 5.1 Rozkład dźwięków w notacji MIDI na klawiaturze fortepianowej

5.2. Rzeczywiste nagrania instrumentów

5.2.1. *Baza nagrań „GOOD SOUNDS”*

Zbiór danych Good Sounds został stworzony w ramach projektu *Pablo* [158], częściowo finansowanego przez firmę KORG. Zawiera on monofoniczne nagrania dwóch rodzajów ćwiczeń muzycznych: pojedynczych nut i skal.

Nagrania zostały wykonane w studiu nagraniowym Uniwersytetu Pompeu Fabra/Phonos przez 15 różnych profesjonalnych muzyków, z których wszyscy posiadają wykształcenie muzyczne i mają pewne doświadczenie w nauczaniu muzyki. 12 różnych instrumentów zostało nagranych przy użyciu jednego lub maksymalnie 4 różnych mikrofonów (w zależności od sesji nagraniowej). Dla wszystkich instrumentów cały zestaw grywalnych półtonów w instrumencie jest nagrywany kilka razy z różnymi charakterystykami tonalnymi. Każda nuta jest zapisywana w osobnym monofonicznym pliku audio w formacie .flac przy częstotliwości 48 kHz i 32 bitach [156].

Pliki audio są zorganizowane w jednym katalogu dla każdej sesji nagraniowej. Oprócz plików dołączony jest jeden plik bazy danych SQLite [86][158].

Liczba próbek instrumentów wykorzystanych w ramach niniejszej pracy została przedstawiona w tabeli 5.1.

Tabela 5.1 Liczba próbek instrumentów z zbioru GOOD SOUNDS wykorzystanych w ramach tworzenia zbioru treningowego i testowego na potrzeby eksperymentów

Instrument	Liczba próbek	Zakres dźwięków (wg. Notacji MIDI)
Bas	159	E1 – G4
Wiolonczela	945	C2 – A4
Klarnet	1688	D3 – D6
Trąbka	934	B3 – C#5
Skrzypce	1383	G3 – G6
Saksofon altowy	720	C#3 – A5
Saksofon tenorowy	680	G#2 – E5
Saksofon barytonowy	288	C2 – C#3
Saksofon sopranowy	334	G#3 – E6
Obój	247	A#3 – E6
Flet piccolo	388	D5 – G7

5.2.2. *Baza nagrań UMAPiano*

Baza *UMAPiano* zawiera akordy o liczbie polifonii od 2 do 10, o trzech poziomach dynamiki (*Forte, Mezzo, Piano*) i trzech różnych stylach gry (*Normal, Staccato, Pedal*) [7]. Zawiera również indywidualne nagranie każdego z 88 klawiszy fortepianu z tymi samymi poziomami dynamiki i stylami gry. Nagrane akordy obejmują pełny zakres gry na fortepianie. Łączna liczba

nagranych plików wynosi 275040, jest to więc duża baza danych, która zawiera bardzo znaczną liczbę akordów wszystkich typów [7].

Liczba próbek fortepianu wykorzystanych w ramach niniejszej pracy została przedstawiona w tabeli 5.2.

Tabela 5.2 Liczba próbek fortepianu z zbioru UMA Piano wykorzystanych w ramach tworzenia zbioru treningowego i testowego na potrzeby eksperymentów

Instrument	Liczba próbek	Zakres dźwięków (wg notacji MIDI)
Fortepian	792	A#0 – G7

5.2.3. *Baza nagrań Philharmonia Orchestra*

Zbiór *Philharmonia* (zwana też all-samples) zawiera nagrania dźwięków instrumentów muzycznych światowej klasy orkiestry symfonicznej z siedzibą w Londynie, prowadzonej przez głównego dyrygenta Santtu-Matiasa Rouvali. Nagrania zostały dokonane w sali Southbank Centre's Royal Festival Hall [159].

Biblioteka próbek dźwiękowych została specjalnie nagrana przez członków orkiestry i jest ogólnie dostępna. Zawiera wszystkie standardowe instrumenty orkiestrowe, a także gitarę, mandolinę, banjo i szeroki zestaw różnych instrumentów perkusyjnych. Próbki nadają się do tworzenia każdego rodzaju muzyki bez względu na styl [160].

Liczba próbek instrumentów wykorzystanych w ramach niniejszej rozprawy doktorskiej została przedstawiona w tabeli 5.3.

Tabela 5.3 Liczba próbek instrumentów z zbioru Philharmonia Orchestra (all-samples) wykorzystanych w ramach tworzenia zbioru treningowego i testowego na potrzeby eksperymentów

Instrument	Liczba próbek	Zakres dźwięków (wg. Notacji MIDI)
Bandżo	74	C3 – E6
Klarnet basowy	944	C2 – C6
Fagot	720	B1 – G5
Wiolonczela	889	C2 – C7
Klarnet	846	D3 – C7
Kontrabas	710	A#0 – D4
Rożek angielski	691	E3 – B5
Kontrabas	852	C1 – G4
Flet	878	C4 – F7
Waltornia	652	A#1 – F5
Gitara	106	E2 – E6
Mandolina	80	G3 – A6

Obój	596	A#3 – A#6
Instrumenty perkusyjne	148	–
Saksofon	732	D3 – F6
Puzon	831	E2 – F6
Trąbka	485	E2 – E6
Tuba	972	B0 – F4
Altówka	973	C3 – F#7
Skrzypce	1502	G3 – B7

5.2.4. *Baza nagrań SLAKH*

Syntezowany zestaw danych Lakh (Slakh) jest zbiorem danych zapisanych jako ścieżki pliku fonicznego oraz synchronizowanych plików MIDI, służący do separacji źródeł muzyki i automatycznej transkrypcji wielu instrumentów. Poszczególne ścieżki MIDI są syntetyzowane z *Lakh MIDI Dataset v0.1* [161] przy użyciu profesjonalnych instrumentów wirtualnych opartych na próbkach Wynikowy dźwięk jest miksowany razem, tworząc utwór muzyczny. Wydanie Slakh2100, zawiera 2100 automatycznie zmiksowanych ścieżek i towarzyszących im, wyrównanych plików MIDI, zsyntezowanych z 187 ścieżek skategoryzowanych w 34 klasach [125].

Slakh jest dostarczany przez *Mitsubishi Electric Research Lab (MERL)* i *Interactive Audio Lab w Uniwersytecie Northwestern* [66].

Liczba próbek instrumentów wykorzystanych w ramach niniejszej pracy została przedstawiona w tabeli 5.4.

Tabela 5.4 Liczba próbek instrumentów z zbioru Slakh wykorzystanych w ramach tworzenia zbioru treningowego i testowego na potrzeby eksperymentów

Instrument	**Liczba próbek**
Akordeon	67
Banjo	17
Bas	2513
Fagot	14
Wiolonczela	39
Klarnet	83
Instrumenty perkusyjne	2107
Flet	445

Gitara	5482
Harfa	70
Obój	93
Fortepian	3495
Flet piccolo	41
Skrzypce	21
Altówka	90
Puzon	141
Trąbka	159
Tuba	39

5.2.5. *Baza nagrań MedleyDB*

Baza MedleyDB [14] została wydana w 2014 r. jako zbiór nagrań wielośladowych opracowany w celu wsparcia badań MIR. Zapewnia on miks stereo oraz zarówno źródłowe, jak i przetworzone nagrania wielośladowe dla każdego utworu w zbiorze danych. Zbiór danych obejmuje szeroki zakres gatunków i składa się głównie z pełnowymiarowych utworów o profesjonalnej lub prawie profesjonalnej jakości dźwięku. Przykładowe zadania w których został wykorzystany to separacja źródeł [97], ekstrakcja melodii [13], rozszerzania danych [70] oraz jako materiał źródłowy do badań nad poznaniem muzyki [84].

Podczas gdy zbiór danych okazał się być cennym zasobem, proces zbierania danych i adnotacji okazał się trudny, przez co w finalnej wersji pojawiło się kilka rodzajów błędów. Druga wersja zbioru oznaczona jako *MedleyDB 2.0* rozwiązuje te problemy poprzez stworzenie strony oraz aplikacji, która pomaga zautomatyzować proces i zapobiec błędom. Narzędzia te zostały wykorzystane do stworzenia drugiego zestawu danych, rozszerzając całkowitą kolekcję do 254 utworów [12].

Liczba próbek instrumentów wykorzystanych w ramach niniejszej pracy została przedstawiona w tabeli 5.5.

Tabela 5.5 Liczba próbek instrumentów z zbioru MedleyDB [12] wykorzystanych w ramach tworzenia zbioru treningowego i testowego na potrzeby eksperymentów

Instrument	**Liczba próbek**
Akordeon	5
Banjo	2
Bas	181
Klarnet basowy	2

Fagot	24
Wiolonczela	30
Klarnet	32
Kontrabas	42
Instrumenty perkusyjne	284
Flet	24
Róg	12
Gitara	280
Harfa	6
Obój	8
Fortepian	96
Flet piccolo	4
Skrzypce	23
Altówka	87
Saksofon	32
Puzon	11
Trąbka	20
Tuba	2

5.2.6. *Nagrania zrealizowane w Katedrze Systemów Multimedialnych*

***Nagrania zrealizowane w ramach pracy dyplomowej* [63]**

Praca dyplomowa magisterska, zrealizowana w Katedrze Systemów Multimedialnych (KSM) na Wydziale Elektroniki, Telekomunikacji i Informatyki (ETI) Politechniki Gdańskiej (PG), zawiera przegląd instrumentów muzycznych ze względu na ich budowę, zastosowanie i możliwości brzmieniowe, ze szczególnym uwzględnieniem artykulacji skrzypiec [63]. W pracy zawarto także dyskusję na temat sposobów nagrań instrumentów solowych. Celem pracy było dokonanie nagrań próbek brzmień kilkunastu instrumentów muzycznych. Nagrania te zostały wykorzystane jako podzbiór w dalszej analizie.

W tabeli 5.6 przedstawiono liczbę próbek wraz z zakresem dźwięków dla poszczególnych instrumentów zarejestrowanych w ramach nagrań.

Tabela 5.6 Liczba próbek instrumentów z zbioru nagranego w ramach dyplomu magisterskiego [58] *wykorzystanych w ramach tworzenia zbioru treningowego i testowego na potrzeby eksperymentów*

Instrument	Liczba próbek	Zakres dźwięków (wg notacji MIDI)
Altówka	361	C3 – C7

Fagot	273	A#1 – B4
Klarnet basowy	237	D3 – F6
Obój	227	A#3 – G6
Puzon basowy	214	E1 – F4
Puzon tenorowy	235	E2 – E5
Róg	229	D2 – D5
Rożek angielski	195	E3 – G#5
Saksofon altowy	225	C#3 – A#5
Saksofon barytonowy	223	C#2 – A#4
Saksofon sopranowy	221	G#3 – C#6
Skrzypce	375	G3 – G7
Trąbka	225	E3 – A5
Tuba B	104	F1 – C#4
Tuba F	159	F1 – C#4
Wiolonczela	363	C2 – A#5

***Nagrania zrealizowane w ramach projektu dyplomowego inżynierskiego* [78]**

W ramach realizacji projektu inżynierskiego [78], którego autor niniejszej rozprawy był konsultantem i wspomagał realizację nagrań, zostały nagrane trzy instrumenty: gitara, klarnet i skrzypce. Proces zbierania próbek oraz ich dalsza obróbka zostały przedstawione w pracy inżynierskiej.

Celem pracy było zebranie informacji o wybranych bazach dźwięków instrumentów muzycznych oraz przygotowanie scenariuszy nagrań kilku wybranych instrumentów, z uwzględnieniem artykulacji muzycznej. Część praktyczna pracy dotyczyła przygotowania bazy wybranych instrumentów muzycznych, zgodnie z wytycznymi dotyczącymi pomieszczenia studyjnego oraz sprzętu nagrywającego − rodzajów mikrofonów i ich rozmieszczenia. W nagraniach uwzględniono charakterystyczne cechy barwy dźwięku każdego z instrumentów oraz artykulację muzyczną.

W tabeli 5.7 przedstawiono liczbę próbek wraz z zakresem dźwięków dla poszczególnych instrumentów zarejestrowanych w ramach nagrań.

Tabela 5.7 Liczba próbek instrumentów z zbioru nagranego w ramach dyplomu inżynierskiego [73] *wykorzystanych w ramach tworzenia zbioru treningowego i testowego na potrzeby eksperymentów*

Instrument	**Liczba próbek**	**Zakres dźwięków (wg notacji MIDI)**
Klarnet	81	D3 – C7
Gitara	319	E2 – E6

Skrzypce	253	G3 – B7

***Nagrania zrealizowane w ramach projektu dyplomowego inżynierskiego* [82]**

W kolejnej pracy studenckiej [82], której autor niniejszej rozprawy doktorskiej był konsultantem i wspomagał realizację nagrań, zrealizowano nagrania wybranych instrumentów muzycznych z uwzględnieniem charakterystyki tych instrumentów. Rejestracja została wykonana z użyciem różnej artykulacji gry. Celem pracy było też stworzenie repozytorium muzycznego dźwięków wybranych instrumentów. W ramach nagrań zostały zarejestrowane trzy instrumenty: harfa, santur i rav vast [82]. Ponieważ nagrane instrumenty (tj. santur i rav vast) rzadko kiedy znajdują się w zbiorach muzycznych, dlatego pokrótce przedstawiono ich charakterystykę.

Rav vast

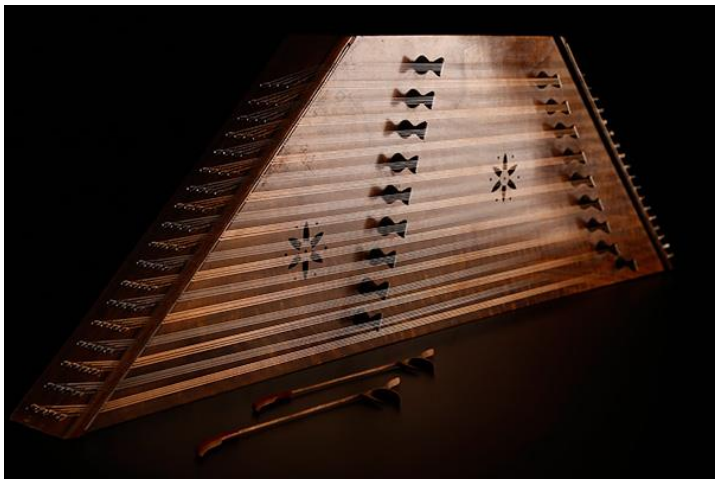
Rav vast jest to metalowy bęben z idealnie wyciętymi stalowymi językami i niepowtarzalnym brzmieniem. Został zbudowany przez rosyjskiego inżyniera, Andreya Remyannikova w 2013 roku, zainspirowany handpanem, okazał się niepowtarzalnym wśród podobnych instrumentów. Posiada hipnotyczne, spokojne, medytacyjne, mistyczne. Obecnie istnieje 21 różnych skal rav vast i 7 skal rav pan i twórca stale pracuje nad nowymi. Każda z nich reprezentuje pewien nastrój, inspirowany określoną kulturą lub harmonią [162]. Na rysunku 5.2 pokazano zdjęcie nagrywanego instrumentu.

Rysunek 5.2 Zdjęcie przedstawiające rav vast [162]

Santur

Santur jest to instrument strunowy z rodziny cymbałów młoteczkowych lub cytra uderzana, występujący w różnych formach w południowo–wschodniej Europie, na Bliskim Wschodzie i w Azji Południowej. Pokrewne instrumenty, znane pod różnymi nazwami, takimi jak węgierski cymbał i chiński yangqin, występują w Europie Środkowej i Zachodniej oraz w Azji Wschodniej. Chociaż dokładne pochodzenie santur pozostaje sporne, powszechnie uważa się, że instrument ten pochodzi z Persji [163].

Santur składa się z płaskiej trapezoidalnej drewnianej ramy lub skrzyni, w poprzek której rozciągnięte są metalowe struny, w które uderza się małymi drewnianymi młotkami lub młotkami. Struny są zwykle przymocowane do metalowych haków lub kołków po lewej stronie pudełka i do kołków strojeniowych po prawej. W zależności od regionu i tradycji, w której gra się na tym instrumencie, liczba strun waha się zazwyczaj od około 72 do ponad 100. Większość strun jest strojona w zestawach po trzy, cztery lub pięć strun (jeśli nie są nastrojone na tę samą wysokość, struny w danej konfiguracji są nastrojone w oktawach). Współczesne instrumenty zazwyczaj posiadają ruchome drewniane mostki, ułożone w dwóch rzędach, które są mniej więcej równoległe do prawej i lewej strony instrumentu. W większości przypadków, każdy mostek podtrzymuje jeden kurs strun. Zdjęcie przedstawiające instrument znajduje się na rysunku 5.3.



Rysunek 5.3 Zdjęcie przedstawiające santur wraz z pałeczkami używanymi do gry na instrumencie [164]

Santur był używany w różnych kontekstach muzycznych na całym obszarze jego występowania. Dla przykładu w Iranie, Iraku, Turcji i Arabii był częścią tradycji muzyki klasycznej. Na wyspach Morza Egejskiego w Grecji jest ważnym elementem muzyki ludowej. W Azji Południowej santur jest silnie związany z regionem Kaszmiru w północno–zachodniej części subkontynentu indyjskiego, gdzie był tradycyjnie grany w muzyce sufickiej, zazwyczaj w małym zespole z bębnami i innymi instrumentami strunowymi. W połowie XX wieku indyjski wirtuoz santur Shiv Kumar Sharma zaadaptował instrument do tradycji klasycznej Hindustani, a tym samym na scenę koncertową. Od tego czasu santur często pojawia się na ścieżkach dźwiękowych filmów i w wykonaniach muzyki popularnej [163].

Harfa

Harfa jest to instrument strunowy, w którym rezonator, jest prostopadły lub prawie prostopadły do płaszczyzny strun. Każda struna wytwarza jedną nutę, przy czym gradacja długości strun od krótkich do długich odpowiada gradacji od wysokich do niskich dźwięków. Rezonator jest zwykle wykonany z drewna lub skóry. W harfach łukowych lub w kształcie łuku, szyjka rozciąga się od korpusu i tworzy z nim łuk. W harfach kątowych, korpus i szyjka tworzą kąt. W harfach ramowych (głównie w Europie) korpus i szyja są ustawione pod kątem i połączone

kolumną, filarem lub przedsionkiem, który usztywnia się przed napięciem strun. Harfy bez przedsionka są naciągane przy stosunkowo niskim napięciu, co powoduje niższą wysokość dźwięku niż w przypadku harf ramowych. Współczesna harfa pedałowa o podwójnym działaniu łączy podstawową strukturę i brzmienie starożytnych harf ze skomplikowanym mechanizmem w celu uzyskania pełnego zakresu chromatycznego [165]. Rysunek 5.4 przedstawia zdjęcie harfy.

Rysunek 5.4. Zdjęcie przedstawiające harfę [166]

Harfy były szeroko stosowane w starożytnym basenie Morza Śródziemnego i na Bliskim Wschodzie, choć rzadko w Grecji i Rzymie; przedstawienia przetrwały z Egiptu i Mezopotamii z około 3000 r. p.n.e. Wiele z nich było granych w pozycji pionowej i skubanych palcami obu rąk, ale Mezopotamia miała również harfy poziome. Umieszczone na kolanach gracza, struny w kierunku gracza, były skubane za pomocą plektrum. Harfy poziome są widoczne w Indiach dopiero od 800 roku, ale najwyraźniej zostały wyparte z Bliskiego Wschodu około 600 roku. W tym samym czasie harfy łukowe przestały być używane na Bliskim Wschodzie, ale przetrwały do czasów obecnych w Afryce, Myanmar (d. Birma) i kilku odosobnionych miejscach. Harfy kątowe przetrwały do XIX wieku w Iranie [165].

Podsumowanie zebranych nagrań

W tabeli 5.8 przedstawiono liczbę próbek wraz z zakresem dźwięków dla poszczególnych instrumentów zarejestrowanych w ramach nagrań.

Tabela 5.8 Liczba próbek instrumentów z zbioru nagranego w ramach dyplomu inżynierskiego wykorzystanych w ramach tworzenia zbioru treningowego i testowego na potrzeby eksperymentów

Instrument	Liczba próbek	Zakres dźwięków (wg notacji MIDI)
Harfa	655	G#1 – C#7

Santur	539	A2 – D6
Rav vast	180	B2 – C#4

5.3. Założenia struktury zbioru treningowego

Zbiór danych przygotowany w kontekście eksperymentów wykorzystuje nagrania zawarte we wszystkich bazach opisanych w rozdziale 5.2. Dzięki takiemu podejściu możliwe było zwiększenie różnorodności danych, tj.:

1. rodzaju instrumentu,
2. jakości nagrania,
3. typu nagrania,
4. zastosowanych mikrofonów,
5. wnętrz, w których rejestrowano nagrania,
6. muzyków grających na danym instrumencie.

Spośród wszystkich dostępnych instrumentów wybrano 13 posiadających największą liczebność próbek, z uwzględnieniem instrumentów nietypowych takich, jak santur i rav vast. Dokładne dane dotyczące liczebności wybranych instrumentów w opisanych bazach danych przedstawiono w tabeli 5.9.

Tabela 5.9 Liczebność wybranych instrumentów w poszczególnych zbiorach danych

	GOOD SOUNDS [158]	*UMA Piano* [7]	*Philharmonia Orchestra* [160]	*Slakh* [125]	*MedleyDB* [12][14]	*Zbiór nagrań zrealizowanych w KSM, ETI, PG* [78]	*Suma*
Bas	159			2513	181		2853
Wiolonczela	945		889	39	30	363	2266
Klarnet	1688		846	83	2	81+237*	2937
Flet	388		878	445	24		1735
Gitara			106	5482	280	319	6187
Harfa				70	6	655	731
Instrumenty perkusyjne		792	148	2107	284		3331
Fortepian				3495	96		3591
Rav vast						182	182
Santur						539	539
Saksofon	2022		732		32	669	3455
Trąbka	934		485	159	20	225	1823
Skrzypce	1383		1502	21	23	253+375*	3557

*oznacza realizację nagrań w różnych warunkach studyjnych

Ze względu na to, że przygotowany zbiór składa się zarówno z utworów muzycznych z określoną aranżacją (bazy MedleyDB i SLAKH) [12][14][125], jak i sztucznie przygotowanych miksów pojedynczych dźwięków różnych instrumentów (bazy Philharmonia Orchestra, UMA Piano, GOOD SOUNDS i prace dyplomowe) [7][63][78][82][158][160], dla każdej z tych grup nagrań zastosowano nieco odmienne podejście.

W przypadku utworów muzycznych − każdy z nich został podzielony na 4-sekundowe fragmenty. Jeśli poziom sygnału instrumentu w wyodrębnionym fragmencie był niższy niż 60 dB, to taki instrument był wykluczany ze zbioru. Pozwoliło to na obniżenie kosztów obliczeniowych i zwiększenie zmienności liczby instrumentów w zmienności miksu danego utworu. Dodatkowo każdy pojedynczy przykład (4 sekundy) ma losowo dobrane wzmocnienie dla wszystkich instrumentów z osobna.

Bazując na pojedynczych dźwiękach instrumentów, w pierwszej kolejności wylosowano instrumenty, a następnie konkretne próbki, które miały się znaleźć w danym przykładzie. Następnie wygenerowano sygnał o długości 4 sekund, składający się z samych zer. Do tego sygnału w losowych jego miejscach zostały dodane poszczególne próbki instrumentów. Dzięki takiemu zabiegowi nie występuje przypadek, w którym sygnał użyteczny znajduje się jedynie na początku próbki. Jeśli plik bazowy wykorzystany do sumowania miał mniej lub więcej niż 4 sekundy, to wybierany został z niego losowy fragment tej długości i umieszczany w wynikowym miksie.

Aby uzyskać powtarzalność wyników treningu, cały zbiór danych został podzielony na trzy części, z zastrzeżeniem, że pojedynczy utwór audio może być wykorzystany w tylko jednej z nich:

1. Zbiór treningowy – 200 000 przykładów;
2. Zbiór walidacyjny – 20 000 przykładów;
3. Zbiór ewaluacyjny – 20 000 przykładów

5.4. Sposób przygotowania danych treningowych

Bazując na zebranych bazach danych, kolejnym istotnym elementem jest opracowanie metodologii przygotowania ich do dalszego przetwarzania, tj. jako danych treningowych dla sieci neuronowych.

W celu przygotowania zbioru uczącego w pierwszej kolejności dokonano indeksowania wszystkich nagrań, tak aby dalsze elementy algorytmu mogły w szybki sposób jedynie tworzyć odpowiednie połączenia bez potrzeby wyszukiwania plików. Skrypt przygotowujący dane do treningu, oprócz poprawnie zaindeksowanych danych, wymagał pliku konfiguracyjnego zawierającego parametry:

1. classes – instrumenty, który mają zostać wykorzystane w eksperymencie,
2. samplerate – częstotliwość próbkowania,

3. signal_mixed_len_in_sec – długość wynikowych próbek,
4. mix_instrument_counts – liczbę instrumentów występujących w miksie,
5. clean_dataset – parametr informujący o usunięciu danych w ścieżce docelowej w przypadku, gdy nie jest ona pusta,
6. trainset_count – liczbę próbek w zbiorze treningowym,
7. devset_count – liczbę próbek w zbiorze walidacyjnym,
8. evalset_count – liczbę próbek w zbiorze ewaluacyjnym
9. seed – początkowe ziarno, będące punktem startowym dla algorytmu losującego,
10. dataset_path – ścieżkę w której ma zostać zapisany zbiór,
11. recordings_list – plik zawierający bazę indeksów nagrań.

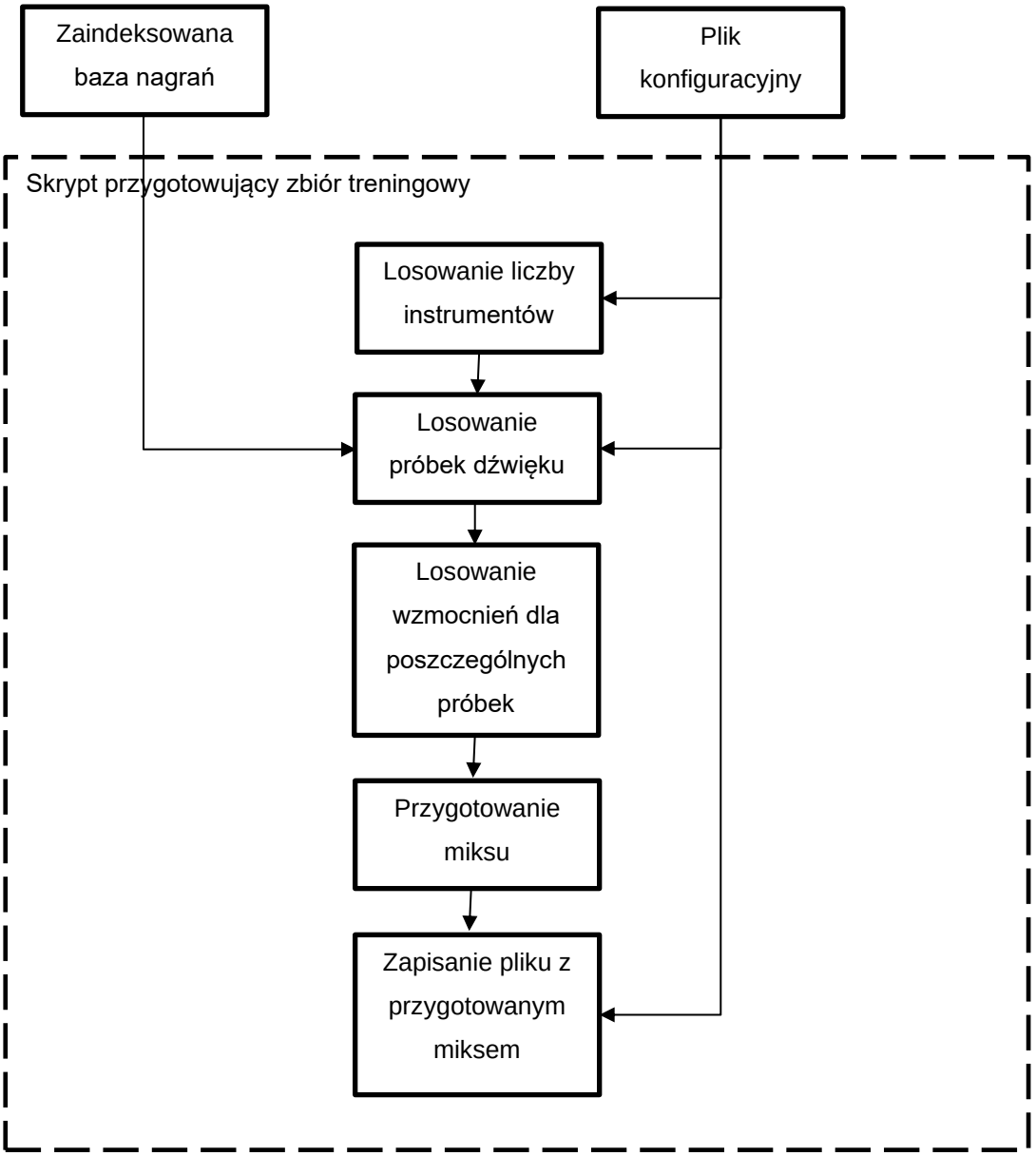
Przykładowa konfiguracja wykorzystana w eksperymentach znajduje się poniżej:

```
{
    "classes" : ["Bass", "Cello", "Clarinet", "Flute", "Guitar", "Harp",
"Drums", "Piano", "Rav_vast", "Santur", "Saxophone", "Trumpet", "Violin"],
    "samplerate" : 16000,
    "signal_mixed_len_in_sec" : 4,
    "mix_instrument_counts" : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
    "clean_dataset" : false,
    "trainset_count" : 200000,
    "devset_count" : 20000,
    "evalset_count" : 20000,
    "seed" : 111111,
    "dataset_path" : "H:\\dataset_v2",
    "recordings_list": "H:\\dataset_v2\\recordings.json"
}
```

Bazując na zaindeksowanych zbiorach oraz na opisanej wyżej konfiguracji, przygotowano zbiór danych, w którym losowana była liczba instrumentów występujących w miksie, a następnie zgodnie z nią losowane były konkretne instrumenty wybrane z zadeklarowanego podzbioru i odpowiadające próbki nagrań. Aby utrudnić zadanie, dodatkowym losowanym parametrem było wzmocnienie sygnałów źródłowych przed ich zsumowaniem, przez co w każdej próbce nagrania miały różne poziomy. Przygotowane w ten sposób miksy zostały zapisane w postaci struktur danych kompatybilnych z pakietem numpy, zawierające:

1. Sygnały referencyjne poszczególnych instrumentów po przeskalowaniu przez wylosowane wzmocnienia.
2. Wektor etykiet, które instrumenty zostały wykorzystane w danej próbce.
3. Zsumowany sygnał.

Dokładny schemat działania narzędzia do przygotowywania danych został przedstawiony na rysunku 5.5.



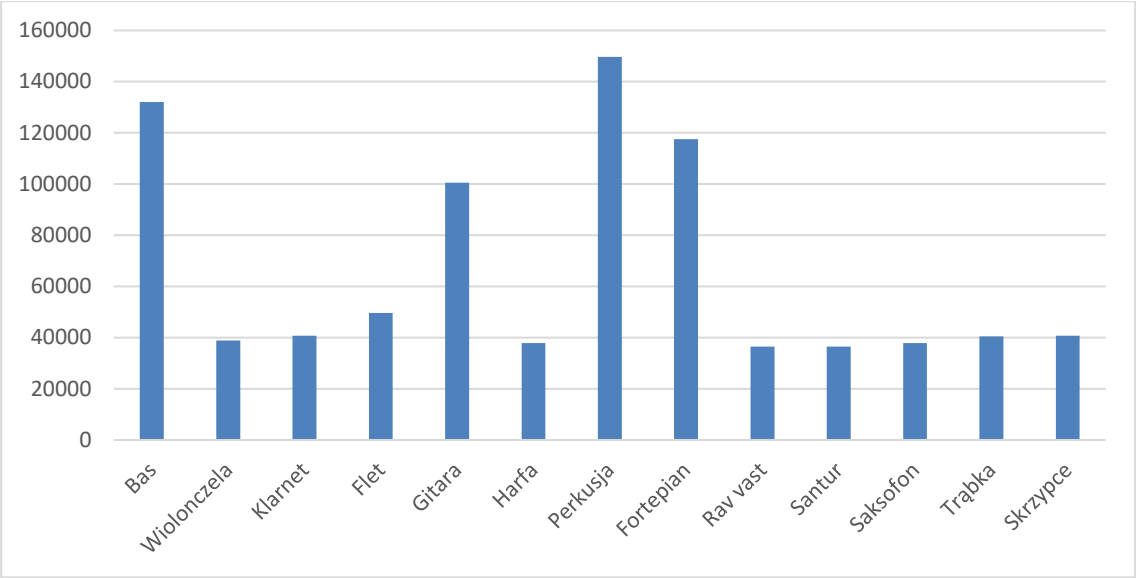
Rysunek 5.5 Schemat działania systemu do przygotowywania danych treningowych dla modeli sieci neuronowych

5.5. Histogramy opisujące zbiory danych

Aby dokładniej zobrazować, jak rozkładają się próbki w poszczególnych podzbiorach, w poniższych podrozdziałach przedstawiono histogramy obrazujące liczebność wystąpienia danego instrumentu oraz liczbę instrumentów muzycznych w miksie.

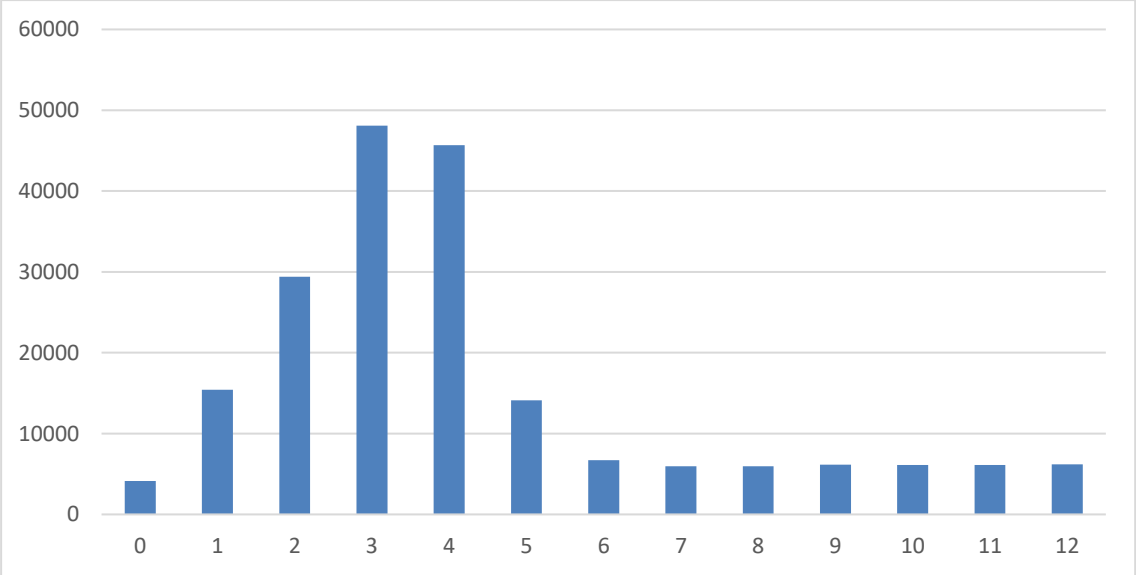
5.5.1. *Zbiór treningowy*

Rysunek 5.6 przedstawia histogram instrumentów muzycznych (tj. liczebność wystąpienia danego instrumentu) w miksie dla zbioru treningowego. Większość instrumentów występuje w zbalansowanej liczbie około 40000 przykładów. Dla basu, gitary, perkusji i fortepianu daje to odpowiednio 131985, 100491, 149622 oraz 117500 próbek dźwiękowych.



Rysunek 5.6 Histogram instrumentów występujących w miksie zbioru treningowego

Rysunek 5.7 przedstawia z kolei rozkład liczby instrumentów muzycznych występujących w miksie dla zbioru treningowego. Obrazuje on szeroki przekrój przykładów, tj. takich, w których nie ma żadnego instrumentu, a skończywszy na 12. Najwięcej występuje próbek składających się z 3 instrumentów – 48077, następnie z 4 instrumentów – 45693 i kolejno 2, 1 i 5 instrumentów, na które składa się odpowiednio 29402, 15431 i 14114 próbek dźwiękowych. Dla pozostałych przypadków liczba instrumentów wynosi ok. 6000.

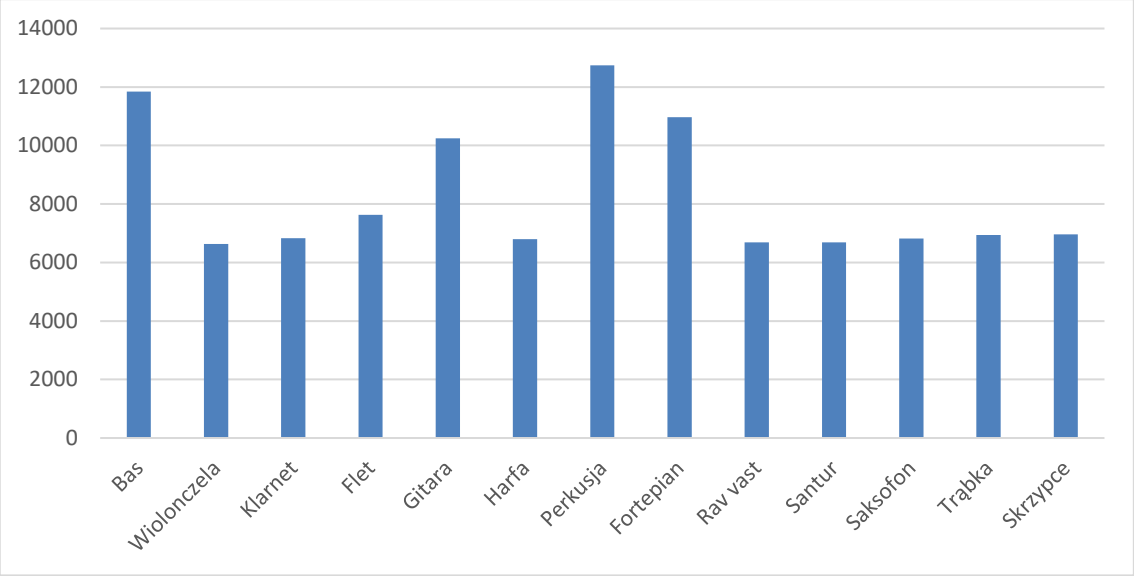


Rysunek 5.7 Rozkład liczby instrumentów muzycznych w miksie zbioru treningowego

5.5.2. *Zbiór testowy*

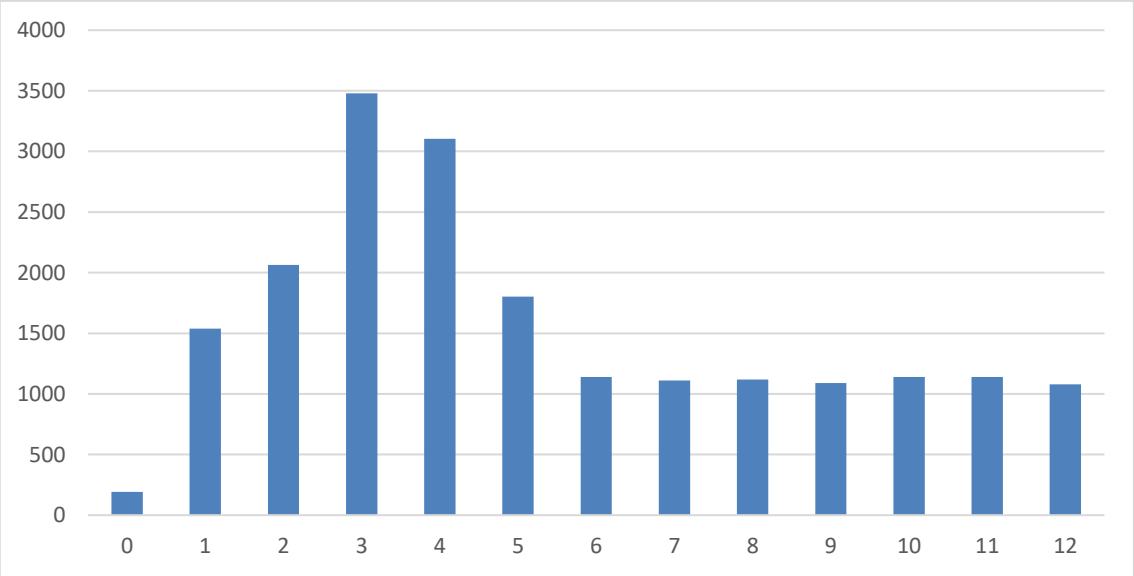
Rysunek 5.8 przedstawia histogram poszczególnych instrumentów muzycznych (tj. liczebność wystąpienia danego instrumentu) występujących w miksie dla zbioru testowego. Podobnie, jak w zbiorze treningowym, większość instrumentów występuje w zbalansowanej

liczbie, tj. około 6500 przykładów. Zbiór treningowy dla basu, gitary, perkusji i fortepianu zawiera najwięcej próbek i wynosi odpowiednio 11847, 10250, 12737 i 10971.



Rysunek 5.8 Histogram instrumentów występujących w miksie zbioru testowego

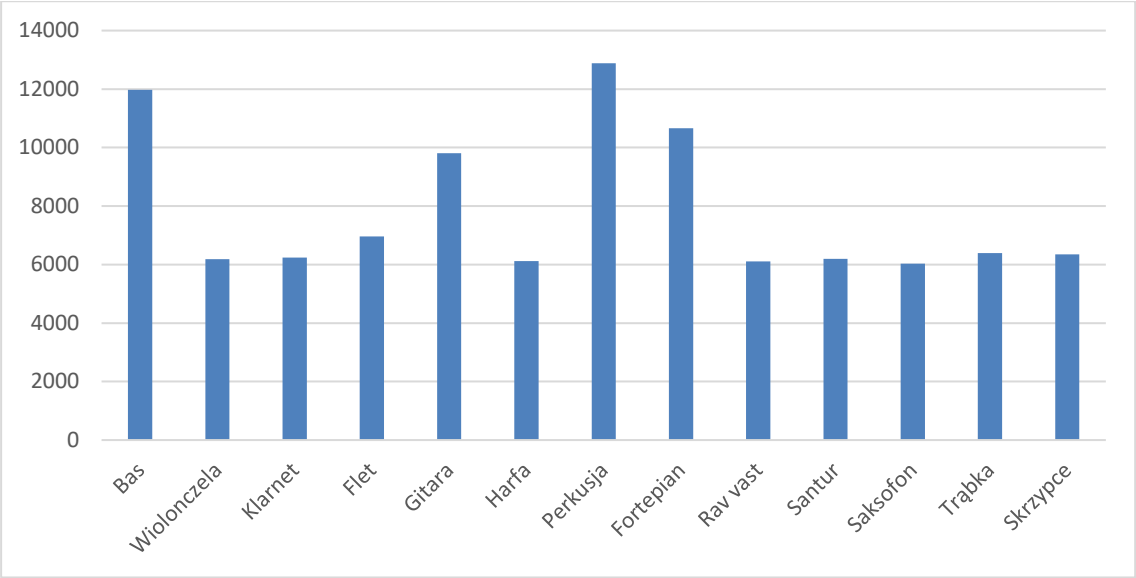
Rysunek 5.9 przedstawia rozkład liczby instrumentów występujących w miksie dla zbioru testowego. Podobnie, jak w przypadku zbioru treningowego, obrazuje on szeroki przekrój przykładów, tj. takich, w których nie ma żadnego instrumentu, a skończywszy na 12. Najwięcej występuje próbek składających się z 3 instrumentów – 3479, następnie z 4 instrumentów – 3103 przykłady i kolejno 2, 5 i 1, które odpowiadają 2064, 1802 i 1538 przykładom próbek dźwiękowych. W 192 próbkach nie zawarto żadnego instrumentu. Dla pozostałych przypadków liczba instrumentów wynosi ok. 1100.



Rysunek 5.9 Rozkład liczby instrumentów muzycznych w miksie zbioru testowego

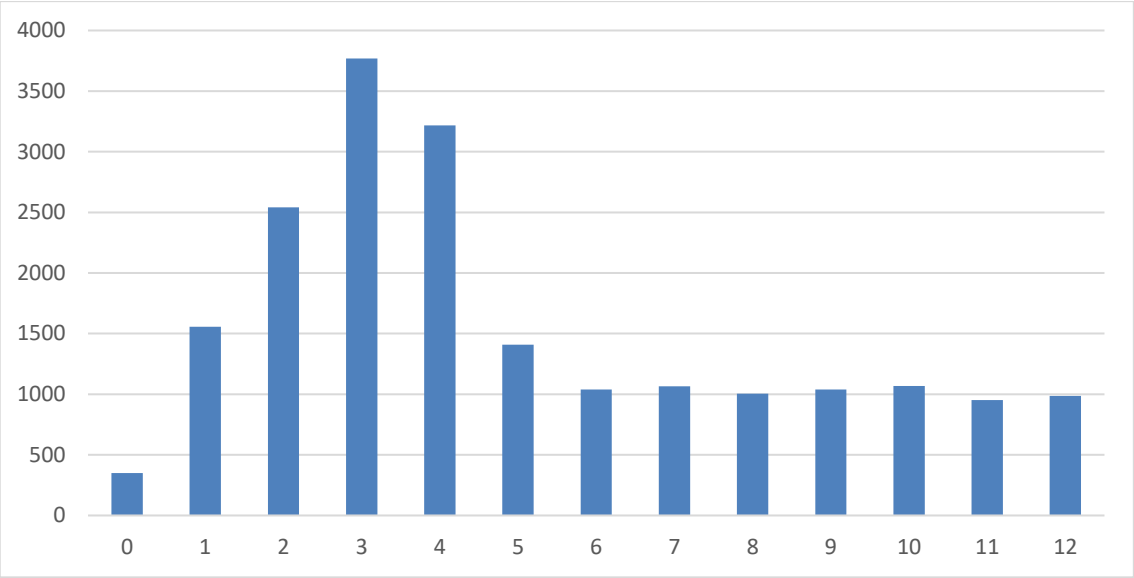
5.5.3. *Zbiór ewaluacyjny*

Rysunek 5.10 przedstawia histogram poszczególnych instrumentów (tj. liczebność wystąpienia danego instrumentu) występujących w miksie dla zbioru ewaluacyjnego. Przedstawia on rozkład analogiczny do zbioru treningowego i testowego – większość instrumentów występuje w zbalansowanej liczbie około 6000 przykładów, zaś dla basu, gitary, perkusji i fortepianu liczby te wynoszą odpowiednio 11979, 9807, 12882 i 10665.



Rysunek 5.10 Histogram instrumentów występujących w miksie zbioru ewaluacyjnego

Rysunek 5.11 przedstawia rozkład liczby występujących w miksie instrumentów dla zbioru ewaluacyjnego. Podobnie, jak w poprzednich przypadkach, rozkład jest zbliżony do zbioru treningowego i testowego. Przykłady zawierają szeroki przekrój liczby instrumentów, tj. takich, w których nie ma żadnego instrumentu, a skończywszy na 12. Najwięcej występuje próbek składających się z 3 instrumentów – 3770, następnie z 4 instrumentów – 3218 i kolejno 2, 1 i 5 instrumentów, na które składa się odpowiednio 2542, 1557 i 1408 przykładów próbek dźwiękowych. Najmniej, bo 351, przykładów nie zawiera żadnego instrumentu. Dla pozostałych przypadków liczba instrumentów wynosi ok. 1000.



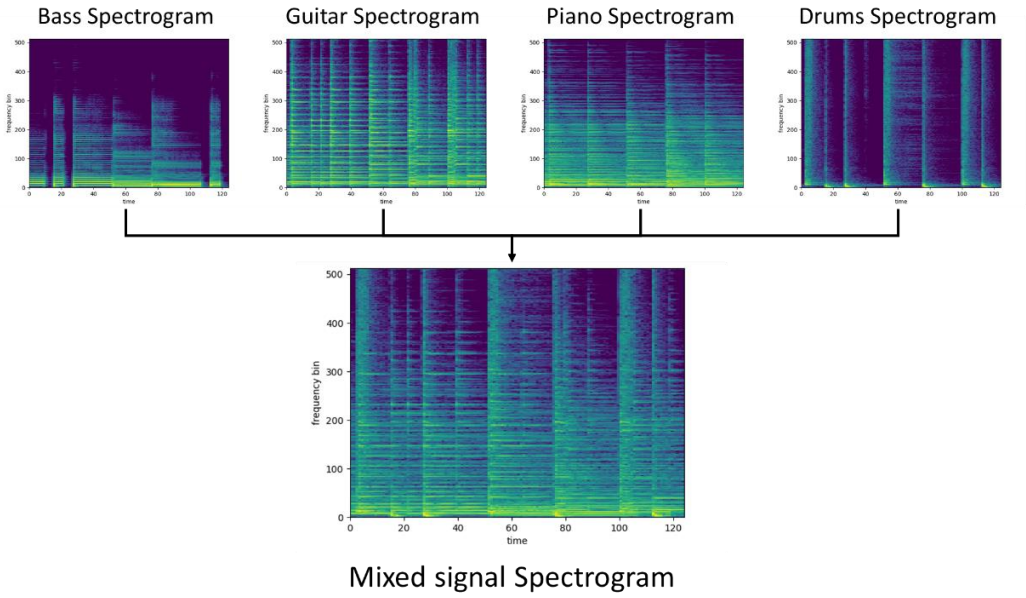
Rysunek 5.11 Rozkład liczby instrumentów w miksie zbioru ewaluacyjnego

6. EKSPERYMENT SPRAWDZAJĄCY

W ramach eksperymentów własnych zaproponowano dodatkowe eksperymenty sprawdzające zasadność zaproponowanych tez zanim podjęto pracę nad eksperymentami w pełnej skali przeprowadzono wstępne eksperymenty bazujące na uproszczonej wersji zbioru treningowego. Poniższy rozdział opisuje przygotowany zbiór, zastosowane architektury sieci neuronowych oraz uzyskane wyniki.

6.1. Zbiór treningowy i ewaluacyjny

W badaniach wykorzystano zbiór danych Slakh, który zawiera 2100 ścieżek dźwiękowych z wyrównanymi plikami MIDI oraz oddzielne ścieżki instrumentów wraz z tagami [167]. Ze wszystkich dostępnych instrumentów do eksperymentu wybrano cztery: bas, perkusję, gitarę i fortepian. Po dokonaniu selekcji każdy utwór został podzielony na 4-sekundowe fragmenty. Jeśli poziom sygnału instrumentu w wyodrębnionym fragmencie był niższy niż -60 dB, to instrument ten był wykluczany z przykładu. Pozwoliło to na obniżenie kosztów obliczeniowych i zwiększenie zmienności liczby instrumentów w zmienności miksu. Dodatkowo każdy przykład ma losowo dobrane wzmocnienie dla wszystkich instrumentów z osobna. Przykładowe spektrogramy wybranych instrumentów oraz korespondującego z nimi miksu przedstawiono na rysunku 6.1.



Rysunek 6.1 Przykładowe spektrogramy wybranych instrumentów i przygotowanego miksu instrumentów

Przykłady były następnie zapisywane przy użyciu formatu NumPy na plikach zawierających zmiksowane sygnały, odniesienia do instrumentów oraz wektory etykiet wskazujących, jakie instrumenty zostały użyte w miksie [168].

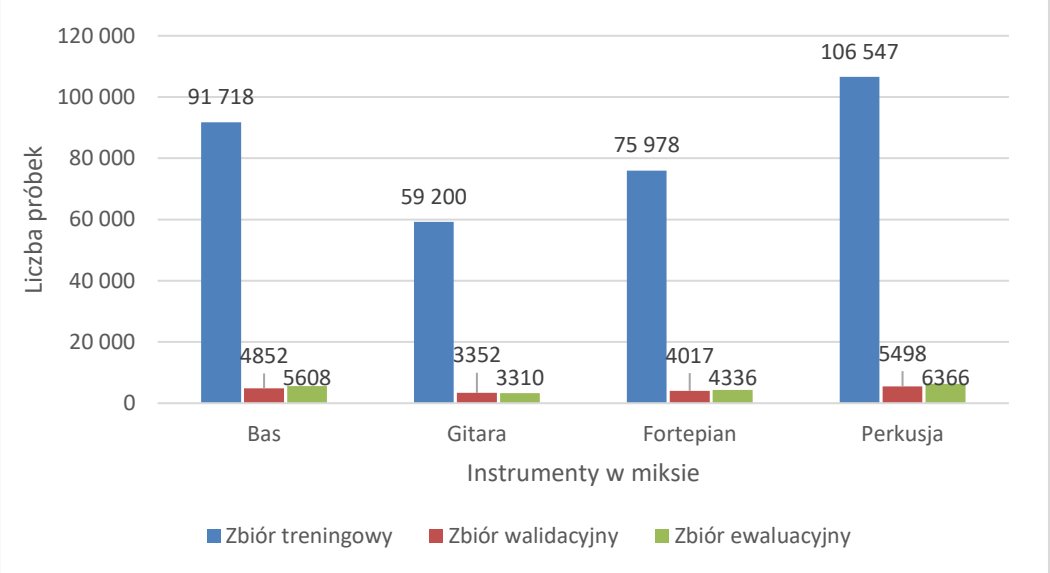
Aby uzyskać powtarzalność wyników treningu, cały zbiór danych został podzielony na trzy części, z zastrzeżeniem, że pojedyncza ścieżka audio może być wykorzystana w tylko jednej z nich:

1. Zbiór treningowy – 116 413 przykładów;
2. Zbiór walidacyjny – 5970 przykładów;
3. Zbiór ewaluacyjny – 6983 przykładów.

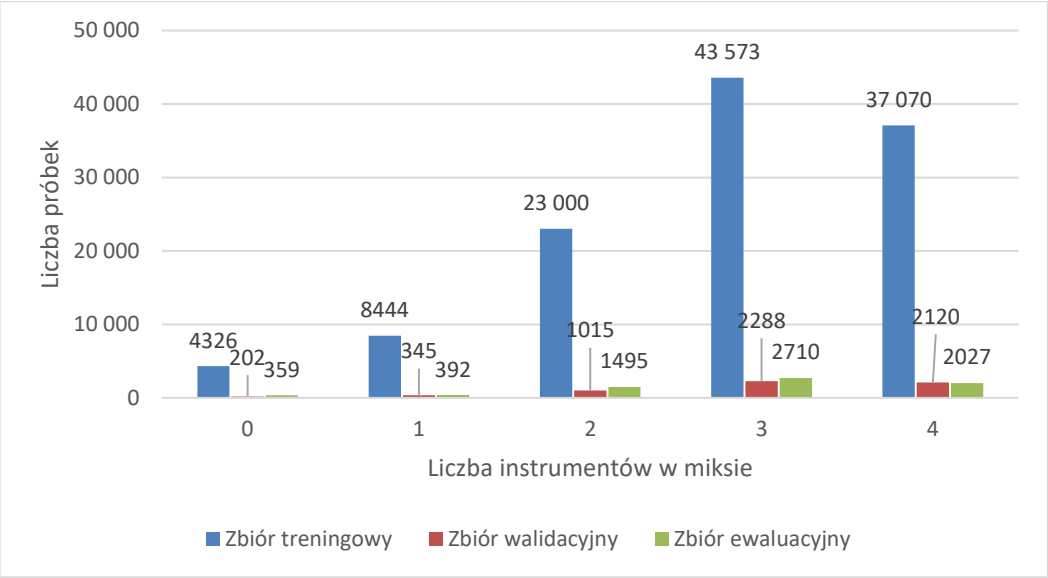
Liczba wystąpień poszczególnych instrumentów muzycznych w miksie jest podobna, aby model nie miał tendencji do podejmowania decyzji z biasem na jeden z nich. Do algorytmu treningowego przekazywany jest wektor wag klas, aby zrównoważyć różnice pomiędzy instrumentami. Obliczone wagi są następujące:

4. Bas – 0,65
5. Gitara – 1,0
6. Fortepian – 0,78
7. Perkusja – 0,56

Ponadto, liczba instrumentów w danej próbce również jest zróżnicowana. Ze względu na strukturę utworów muzycznych, największa część (około 1/3 wszystkich przykładów w zbiorze danych) zawiera trzy instrumenty. Pozostałe części wypełniają cztery, dwa, a następnie jeden instrument. Dodatkowo, na wejście algorytmu wprowadzane są próbki muzyki, które nie posiadają żadnego instrumentu, aby system wiedział, że taki przypadek również może mieć miejsce. Histogramy klas instrumentów w miksach oraz liczba instrumentów w miksie zostały przedstawione na rysunkach 6.2 i 6.3.



Rysunek 6.2 Histogram instrumentów występujących w miksie



Rysunek 6.3 Liczba instrumentów w miksach

6.2. Architektura modelu

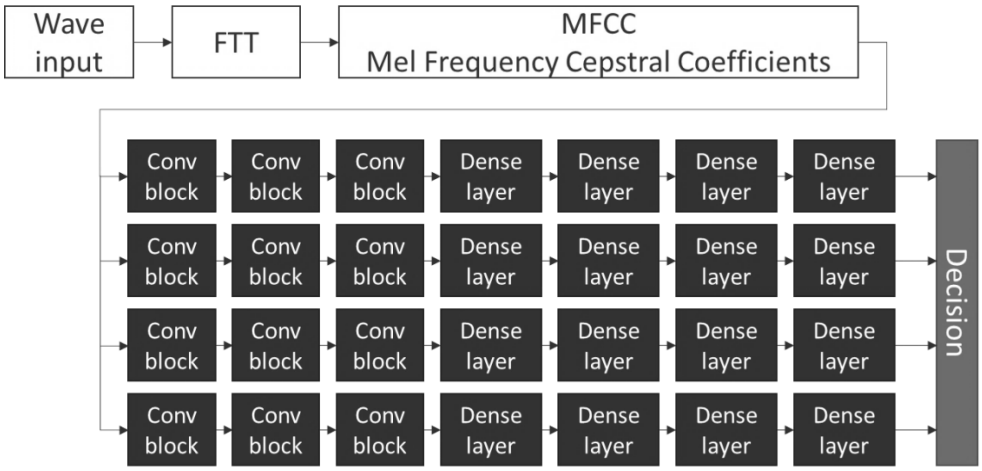
Proponowana sieć neuronowa została zaimplementowana przy użyciu narzędzia Keras i funkcjonalnego API [169]. Model początkowo wytwarza współczynniki MFCC (Mel-Frequency Cepstral Coefficients) z surowego sygnału audio przy użyciu wbudowanych metod Keras [170]. Parametry tych operacji są następujące:

– 1024 próbki długość okna Hamminga;

– 512 próbek krok okna;

– 40 współczynników MFCC.

W przeciwieństwie do innych metod, w których pojedynczy model dokonuje identyfikacji lub klasyfikacji wszystkich instrumentów, zastosowany model wykorzystuje zestawy indywidualnych, identycznie zdefiniowanych podmodeli − jeden na instrument. Proponowana architektura zawiera na początku dwuwymiarowe warstwy splotowe. Liczba filtrów wynosiła odpowiednio 128, 64 i 32 z (3, 3) jądrami i funkcją aktywacji ReLU [171]. Dodatkowo do modelu włączono 2-wymiarowe wyciąganie wartości maksymalnej oraz normalizację po każdym splocie [171–175]. Do uzyskania decyzji użyto czterech gęstych warstw, zawierających odpowiednio 64, 32, 16 i 1 jednostkę [176]. Model zawiera 706182 parametrów, które podlegają procesowi treningu. Topologia sieci została przedstawiona na rysunku 6.4.



Rysunek 6.4 Architektura modelu

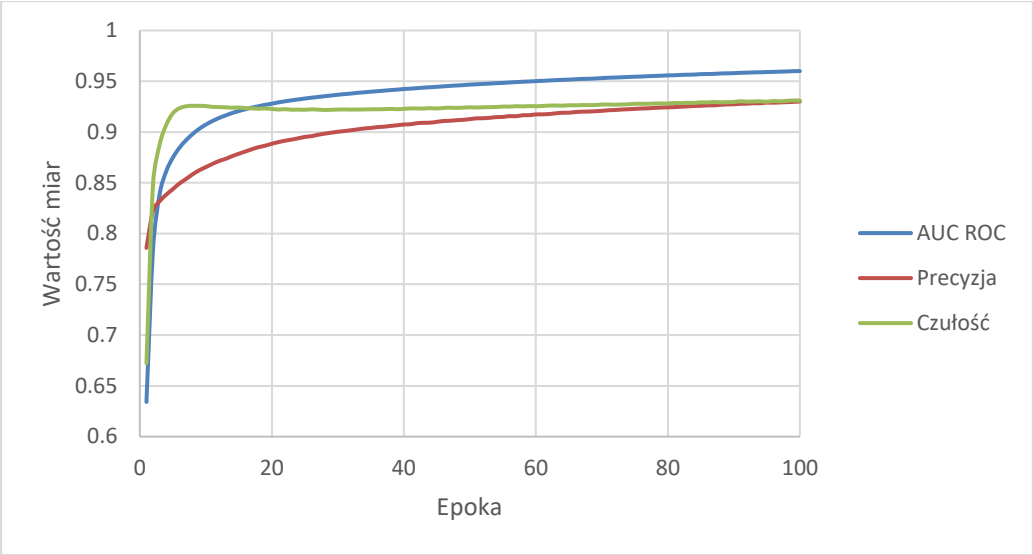
Poniżej przedstawiono uproszczony kod przygotowania modelu. Każdy instrument ma swoją własną funkcję przygotowania modelu, w której można utworzyć nowy model lub załadować wstępnie wytrenowany. W ostatniej operacji wyjścia ze wszystkich modeli są konkatenowane i ustawiane jako wyjście całości.

```
def prepareModel(input_shape):
dense_outputs = []
input = Input(shape = input_shape)
mfcc = prepareMfccModel(input)
dense_outputs.append(prepareBassModel(mfcc))
dense_outputs.append(prepareGuitarModel(mfcc))
dense_outputs.append(preparePianoModel(mfcc))
dense_outputs.append(prepareDrumsModel(mfcc))
        concat = Concatenate()(dense_outputs)
model = Model(inputs = input, outputs = concat)
return model
```

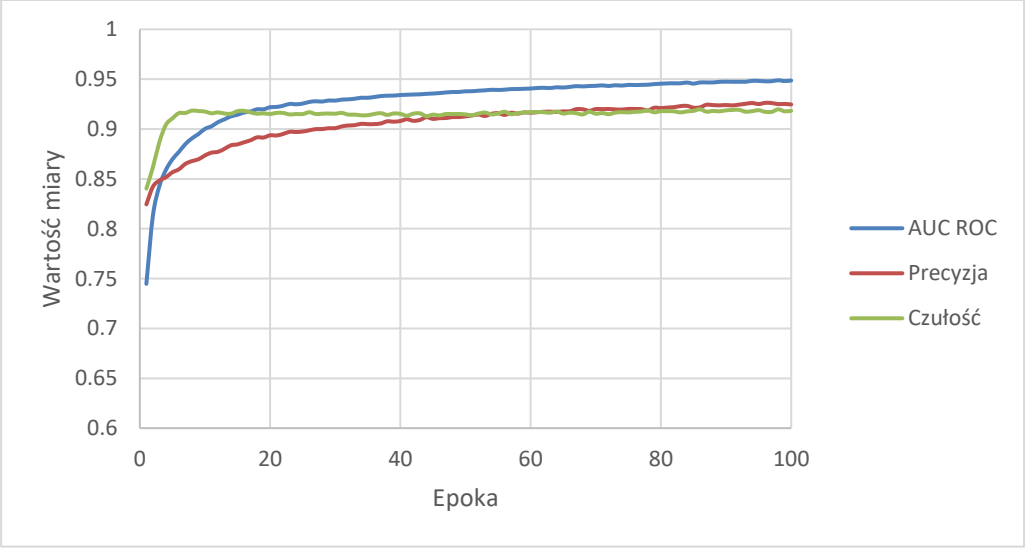
6.3. Trening modelu

Trening został przeprowadzony przy użyciu narzędzia Tensorflow dla języka Python. Model był trenowany przez 100 epok z użyciem błędu średniokwadratowego (MSE) jako funkcji kosztu [177]. Dodatkowo podczas treningu obliczano precyzję, czułość oraz AUC ROC (ang. Area Under the Receiver Operating Characteristic Curve) [154][178]. Najlepszy model był wybierany na podstawie miary AUC ROC opisanej w rozdziale 4.6.3.

Precyzja, czułość oraz AUC ROC obliczone podczas treningu zostały przedstawione na rysunkach 6.5 i 6.6. Na zbiorze treningowym czułość zaczyna się od wartości 0,67 i wzrasta do 0,93. Precyzja zaczyna się od wyższej wartości, 0,78, i wzrasta przez cały proces treningu do wartości 0,93. AUC ROC buduje się od najniższej wartości, 0,63, ale rośnie do najwyższej wartości 0,96. Wartości miar dla zbiorów walidacyjnych są podobne, jak dla zbiorów treningowych. Podczas treningu obie miary rosną do wartości 0,95 i 0,97 dla zbioru treningowego i odpowiednio 0,95 i 0,94 dla zbioru walidacyjnego. Wskaźnik czułości zaczyna się od 0,84 i wzrasta do 0,93, precyzja od 0,82 do 0,92, a AUC ROC od 0,74 do 0,95.



Rysunek 6.5 Miary uzyskane przez algorytm na zbiorze treningowym



Rysunek 6.6 Miary uzyskane przez algorytm na zbiorze walidacyjnym

Trening został przeprowadzony przy użyciu pojedynczej karty graficznej RTX2070 z procesorem AMD Ryzen 5 3600 i 32 GB pamięci RAM. Czas trwania pojedynczej epoki to około 8 min przy zastosowaniu wieloprocesowego ładowania danych i przy wielkości zestawu danych równym 200.

6.4. Wyniki oceny i dyskusja

Ewaluacja została przeprowadzona z wykorzystaniem zbioru 6983 przykładów przygotowanych ze ścieżek audio, które nie były zawarte w zbiorach treningowych i walidacyjnych. Czas przetwarzania pojedynczego przykładu wynosił ok. 0,44 s, zatem algorytm działa ok. 10 razy szybciej niż w czasie rzeczywistym. Uśrednione wyniki dla poszczególnych miar przedstawiają się następująco:

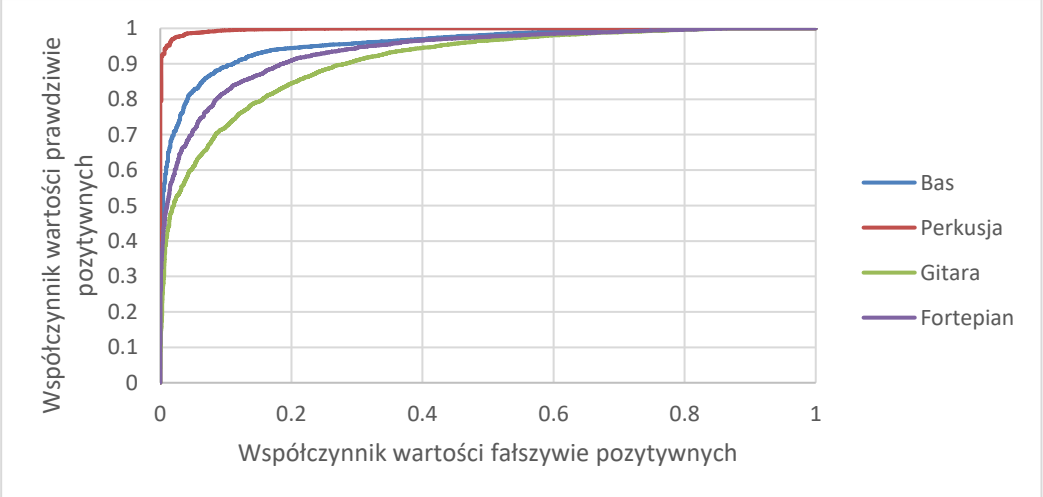
- Precyzja – 0,92;

- Czułość – 0,93;
- AUC ROC – 0,96;
- F1 – 0,93.

Poszczególne składowe precyzji i czułości są następujące:

- Prawdziwie pozytywne – 17759;
- Prawdziwie negatywne – 6610;
- Fałszywie pozytywne – 1512;
- Fałszywie negatywne – 1319.

Na podstawie uzyskanych wyników przeprowadzono również bardziej szczegółowe analizy, w kontekście rozróżniania (identyfikacji) poszczególnych instrumentów. Krzywe ROC zostały przedstawione na rysunku 6.7. Wskazują one, że najłatwiejszą do rozpoznania klasą jest perkusja, dla której można uzyskać współczynnik identyfikacji prawdziwie pozytywnych na poziomie 0,95 przy stosunkowo niskim współczynniku fałszywie pozytywnych wynoszącym około 0,01. Nieco gorzej algorytm radzi sobie z identyfikacją basu, ponieważ, aby uzyskać podobną skuteczność, współczynnik identyfikacji fałszywie pozytywnych musiałby wynosić 0,2. W przypadku gitary i fortepianu, to aby osiągnąć skuteczność około 0,9, należy przyjąć współczynnik identyfikacji fałszywie pozytywnych na poziomie odpowiednio 0,27 i 0,19.



Rysunek 6.7 Krzywe ROC dla każdego z testowanych instrumentów

Szczegółowe wartości precyzji, czułości oraz liczby prawdziwych i fałszywych detekcji przedstawiono w tabeli 6.1. Porównując te wyniki z wykresem ROC na rysunku 6.7, można zauważyć, że model lepiej radzi sobie z rozpoznawaniem perkusji, a także basu. Obserwując wartości miar uzyskanych dla gitary, można zauważyć, że model wykazuje podobną tendencję dla próbek oznaczonych jako fałszywe negatywne i fałszywe pozytywne. W przypadku fortepianu zachodzi odwrotna zależność, tzn. więcej próbek jest oznaczanych jako fałszywie pozytywne. Tabela 6.2 przedstawia macierz pomyłek dla przeprowadzonego eksperymentu.

Tabela 6.1 Wartości miar dla poszczególnych instrumentów

Miara	Bas	Perkusja	Gitara	Fortepian
Precyzja	0.94	0.99	0.82	0,87
Czułość	0.94	0.99	0.82	0,91
F1	0.95	0.99	0.82	0,89
Prawdziwie pozytywne	5139	6126	2683	3811
Prawdziwie negatywne	1072	578	2921	2039
Fałszywie pozytywne	288	38	597	589
Fałszywie negatywne	301	58	599	361

Tabela 6.2 Macierz pomyłek [%]

		Rzeczywisty instrument występujący w miksie [%]			
		Bas	Gitara	Fortepian	Perkusja
Wyznaczony przez model instrument	Bas	81	8	7	0
	Gitara	4	69	13	0
	Fortepian	5	12	77	0
	Perkusja	3	7	6	82

6.5. Redefinicja architektury modelu

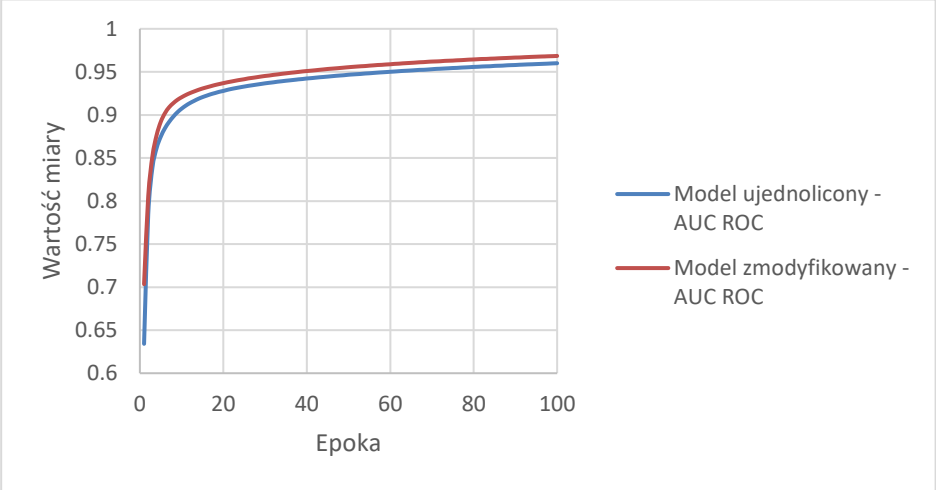
Wykorzystując możliwości architektury modelu do łatwej zamiany całych bloków na indywidualne instrumenty, przeprowadzono dodatkowy eksperyment. Podmodele dla perkusji i gitary zostały zmienione odpowiednio na mniejszy i większy. Szczegółowe porównanie struktury bloków przed i po wprowadzonych zmianach przedstawia tabela 6.3.

Tabela 6.3 Porównanie pomiędzy modelem bazowym i zmodyfikowanym

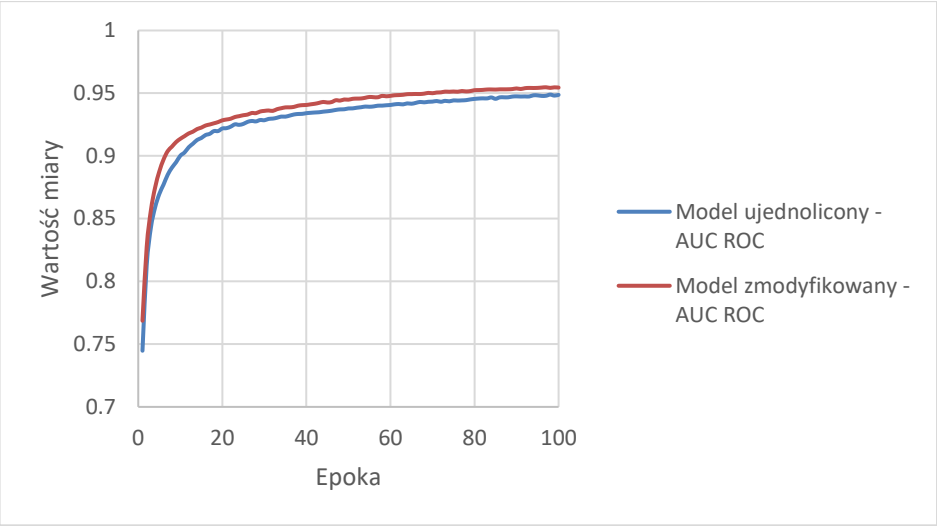
Numer bloku	Podmodel ujednolicony	Podmodel dla gitary	Podmodel dla perkusji
1	• Splot 2D: • Jądro przekształcenia − 3 × 3 • Liczba filtrów − 128 • Wybranie wartości maksymalnej: • Jądro przekształcenia 2 × 2 • Normalizacja	• Splot 2D: • Jądro przekształcenia − 3 × 3 • Liczba filtrów − 256 • Wybranie wartości maksymalnej: • Jądro przekształcenia 2 × 2 • Normalizacja	• Splot 2D: • Jądro przekształcenia − 3 × 3 • Liczba filtrów − 64 • Wybranie wartości maksymalnej: • Jądro przekształcenia 2 × 2 • Normalizacja

2	• Splot 2D: • Jądro przekształcenia − 3 × 3 • Liczba filtrów − 62 • Wybranie wartości maksymalnej: • Jądro przekształcenia 2 × 2 • Normalizacja	• Splot 2D: • Jądro przekształcenia − 3 × 3 • Liczba filtrów − 128 • Wybranie wartości maksymalnej: • Jądro przekształcenia 2 × 2 • Normalizacja	• Splot 2D: • Jądro przekształcenia − 3 × 3 • Liczba filtrów − 32 • Wybranie wartości maksymalnej: • Jądro przekształcenia 2 × 2 • Normalizacja
3	• Splot 2D: • Jądro przekształcenia − 3 × 3 • Liczba filtrów − 32 • Wybranie wartości maksymalnej: • Jądro przekształcenia 2 × 2 • Normalizacja	• Splot 2D: • Jądro przekształcenia − 3 × 3 • Liczba filtrów − 64 • Wybranie wartości maksymalnej: • Jądro przekształcenia 2 × 2 • Normalizacja	• Splot 2D: • Jądro przekształcenia − 3 × 3 • Liczba filtrów − 16 • Wybranie wartości maksymalnej: • Jądro przekształcenia 2 × 2 • Normalizacja
4	• Warstwa w pełni połączona: • Units − 64	• Warstwa w pełni połączona: • Units − 64	• Warstwa w pełni połączona: • Units − 64
5	• Warstwa w pełni połączona: • Units − 32	• Warstwa w pełni połączona: • Units − 32	• Warstwa w pełni połączona: • Units − 32
6	• Warstwa w pełni połączona: • Units − 16	• Warstwa w pełni połączona: • Units − 16	• Warstwa w pełni połączona: • Units − 16
7	• Warstwa w pełni połączona: • Units − 1	• Warstwa w pełni połączona: • Units − 1	• Warstwa w pełni połączona: • Units − 1

Krzywe AUC ROC obliczone w fazie treningu i walidacji przedstawiono na rysunkach 6.8 i 6.9. Krzywe treningowe i walidacyjne wyglądają podobnie, jednak zmodyfikowany model osiąga lepsze wyniki o około 0,01.



Rysunek 6.8 Krzywe AUC ROC obliczone w fazie treningu



Rysunek 6.9 Krzywe AUC ROC obliczone w fazie walidacji

Ocena nowego modelu została przygotowana w oparciu o te same warunki, co w rozdziale 6.3. Porównanie wyników dla pierwszego podmodelu i modeli po modyfikacjach przedstawiono w tabeli 6.4, natomiast w tabeli 6.5 przedstawiono wyniki w podziale na instrumenty. Ze względu na zaokrąglanie wartości miar do dwóch miejsc po przecinku, różnice nie są widoczne. Jednak porównując wartości identyfikacji prawdziwie pozytywnych, prawdziwie negatywnych i fałszywie pozytywnych, wszystkie te miary są wyższe dla zmodyfikowanego modelu niż dla modelu ujednoliconego o około 200 przykładów. Jedynie wartość fałszywych negatywnych przykładów jest gorsza w 61 przykładach. Patrząc na wyniki dla zmienionych instrumentów, można zauważyć, że mniejszy model dla perkusji wypada podobnie w porównaniu z modelem zmodyfikowanym w kontekście zmiany podmodelu dla perkusji i gitary. Większy model uzyskany dla gitary prezentuje lepsze wyniki w zakresie precyzji i wskaźnika F1.

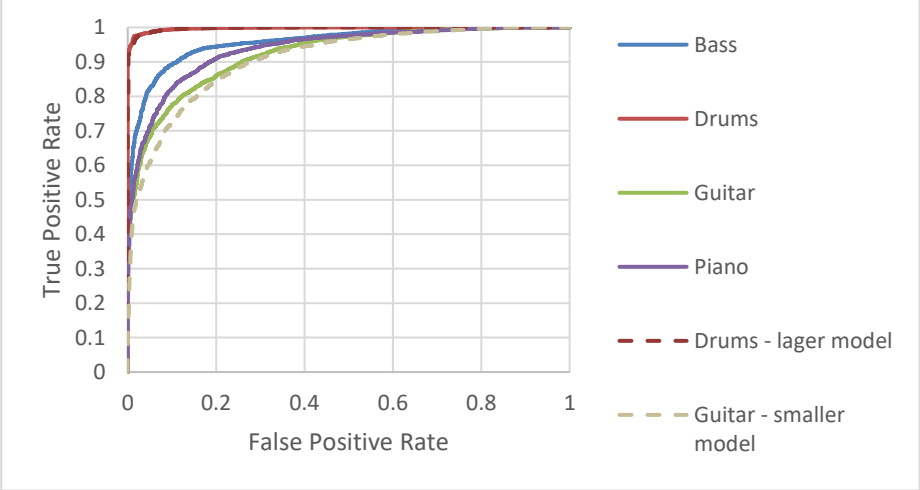
Tabela 6.4 Porównanie wyników pomiędzy modelem ujednoliconym i zmodyfikowanym

Miara	Model ujednolicony	Model zmodyfikowany
Precyzja	0,92	0,93
Czułość	0,93	0,93
AUC ROC	0,96	0,96
F1	0,93	0,93
Prawdziwie pozytywne	17759	17989
Prawdziwie negatywne	6610	6851
Fałszywie pozytywne	1512	1380
Fałszywie negatywne	1319	1380

Tabela 6.5 Wyniki z rozróżnieniem na poszczególne instrumenty

	Perkusja		Gitara	
Miara	***Model ujednolicony***	***Model zmodyfikowany***	***Model ujednolicony***	***Model zmodyfikowany***
Precyzja	0,99	0,99	0,82	0,86
Czułość	0,99	0,99	0,82	0,8
F1	0,99	0,99	0,82	0,83
Prawdziwie pozytywne	6126	6232	2683	2647
Prawdziwie negatywne	578	570	2921	3150
Fałszywie pozytywne	38	47	597	444
Fałszywie negatywne	58	51	599	659

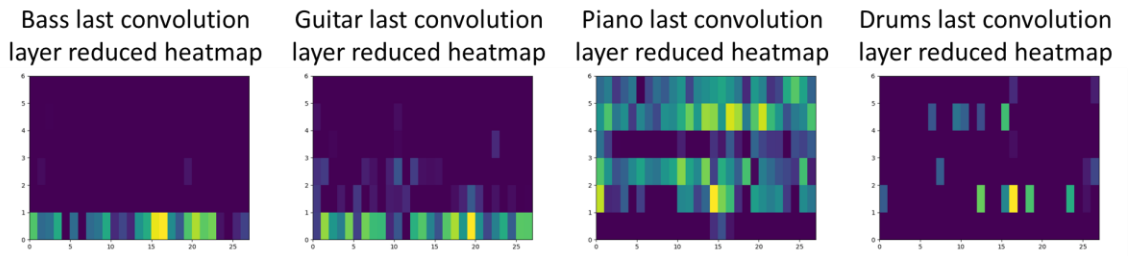
Krzywe ROC dla modeli ujednoliconych i zmodyfikowanych przedstawiono na rysunku 6.10. Można zauważyć, że dla zmodyfikowanych modeli dla perkusji i gitary mniejszy model dla perkusji ma niemal identyczny kształt krzywej ROC. Natomiast model gitary wykazuje lepsze wyniki przy zastosowaniu większego modelu, np. współczynnik przykładów prawdziwie pozytywnych wzrasta z 0,72 do 0,77, natomiast współczynnik przykładów fałszywie pozytywnych jest równy 0,1.



Rysunek 6.10 Krzywe ROC dla zmodyfikowanych instrumentów

Rysunek 6.11 przedstawia zredukowane mapy ciepła dla ostatnich warstw splotowych dla identyfikowanych instrumentów dla jednego z przykładów ze zbioru danych ewaluacyjnych. Porównując mapy ciepła między sobą, można zauważyć, że model dla basu skupia się głównie na niższych częstotliwościach dla całego sygnału, gitarowy na niskich i średnich

częstotliwościach, fortepianowy na średnich i wysokich częstotliwościach, ale również na całym sygnale, i wreszcie perkusyjny na wszystkich częstotliwościach i krótkich sygnałach.



Rysunek 6.11 Zredukowane mapy ciepła dla ostatnich warstw splotowych, dla każdego z identyfikowanych instrumentów

6.6. Dyskusja uzyskanych wyników

Przedstawione wyniki pokazują, że możliwe jest określenie instrumentów występujących w danym fragmencie nagrania muzycznego z dokładnością 93% i wynikiem F1 0.93 przy użyciu prostej sieci splotowej opartej na współczynnikach mel-cepstralnych (MFCC).

Eksperyment pokazuje również, że skuteczność identyfikacji zależy od badanego instrumentu. Łatwiejsza do zidentyfikowania jest perkusja, natomiast gitara i fortepian uzyskują słabsze wyniki rozpoznania.

Obecny stan wiedzy w dziedzinie rozpoznawania dźwięku koncentruje się na rozpoznawaniu pojedynczych lub dominujących w miksie instrumentów oraz klasyfikacji gatunkowej. W odniesieniu do wyników tych zadań, można uzyskać dokładność około 100%, ale wyniki rozpoznawania instrumentów muzycznych przynoszą niższe wartości miar, np. AUC ROC około 0,91 lub F1 w przybliżeniu 0,64. Zaproponowane w ramach rozprawy doktorskiej rozwiązanie może osiągnąć AUC ROC na poziomie około 0,96 oraz F1 na poziomie około 0,93, przewyższając pozostałe metody.

Dodatkową różnicą w stosunku do metod stanu wiedzy (*state-of-the-art*) jest elastyczność modelu. Przedstawione wyniki pokazują, że operacja przełączania podmodelu pozwala np. na zmniejszenie rozmiaru modelu w przypadku, gdy instrument jest łatwo identyfikowalny bez wpływu na architekturę pozostałych identyfikatorów. Tym samym możliwa jest oszczędność mocy obliczeniowej w porównaniu z modelem o dużej, zunifikowanej architekturze. Z drugiej strony, podmodel może być zwiększony w celu poprawy wyników dla instrumentu prezentującego gorszą jakość bez wpływu na pozostałe badane instrumenty.

7. ZADANIA ZWIĄZANE Z KLASYFIKACJĄ I IDENTYFIKACJĄ INSTRUMENTÓW W SYGNALE FONICZNYCZYM

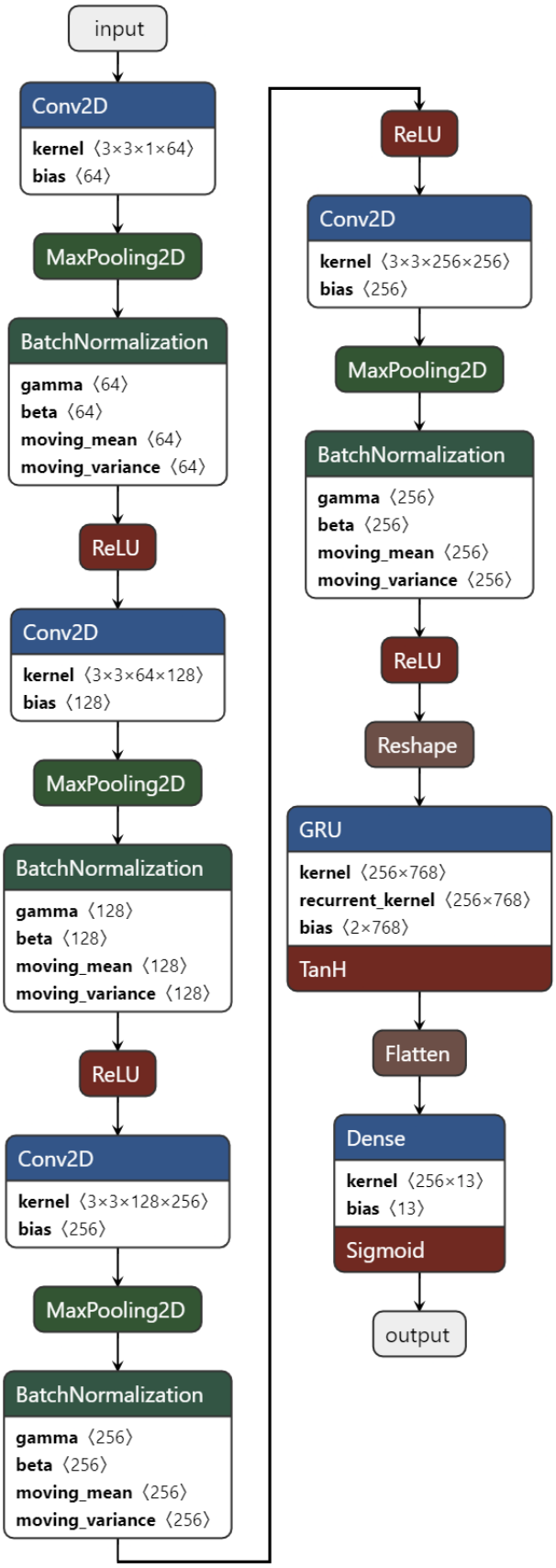
W niniejszym rozdziale omówione zostaną przykładowe metody klasyfikacji oraz identyfikacji instrumentów muzycznych zaprezentowane na przestrzeni ostatnich pięciu lat. Bazując na przygotowanych przez autora danych, przeprowadzone zostały eksperymenty sprawdzające zaproponowane w literaturze rozwiązania.

7.1. Implementacja modelu w architekturze CRNN

7.1.1. *Opis architektury modelu CRNN*

W eksperymentach przeprowadzonych przez Gururaniego i współautorów [38] najwyższą skuteczność uzyskano dla sieci splotowo-rekurencyjnej (CRNN), dlatego na potrzeby porównania przygotowano (zaadaptowano) właśnie taką architekturę. Na wejście modelu przekazywany jest cepstrogram w skali melowej składający się z 96 współczynników. Zastosowano czterokrotne zestawienie dwuwymiarowej warstwy splotowej z warstwą normalizacji, ekstrakcji wartości maksymalnych i funkcji aktywacji ReLU. Warstwy splotowe zostały zbudowane w oparciu o jądro przekształcenia o wymiarze (3, 3) i zawierały kolejno: 64. 128, 256 i 256 filtrów. Cześć rekurencyjna modelu CRNN została zrealizowana pojedynczą warstwą GRU składającą się z 256 jednostek.

Wyjściem z modelu jest warstwa w pełni połączona posiadająca tyle wyjść, ile model obsługuje klas – w trenowanym przypadku 13. Funkcja aktywacji dla tej warstwy jest sigmoidalna. Na rysunku 7.1 przedstawiono schemat blokowy modelu, w którym zawarto wartości hiperparametrów oraz szczegóły implementacji.



Rysunek 7.1 Architektura modelu CRNN zaadaptowanego na podstawie pracy Gururaniego i współautorów [38]

Szczegółowy kod opisujący definicję testowanej architektury przedstawiono poniżej:

```
    input = Input(shape=input_shape)

    mfcc = MFCC(num_mel_bins=96)(input)

    conv_blocks = []

    recurrent_layers = []

    recurrent _outputs = []

    conv_blocks.append(Conv2D(64, (3, 3), padding="valid", dilation_rate=(1,
1), name=f'Conv2D_64')(mfcc))

    conv_blocks.append(MaxPool2D((2, 2))(conv_blocks[-1]))

    conv_blocks.append(BatchNormalization()(conv_blocks[-1]))

    conv_blocks.append(ReLU()(conv_blocks[-1]))

    conv_blocks.append(Conv2D(128, (3, 3), padding="valid", dilation_rate=(1,
1), name=f'Conv2D_128')(conv_blocks[-1]))

    conv_blocks.append(MaxPool2D((2, 2))(conv_blocks[-1]))

    conv_blocks.append(BatchNormalization()(conv_blocks[-1]))

    conv_blocks.append(ReLU()(conv_blocks[-1]))

    conv_blocks.append(Conv2D(256, (3, 3), padding="valid", dilation_rate=(1,
1), name=f'Conv2D_256')(conv_blocks[-1]))

    conv_blocks.append(MaxPool2D((2, 2))(conv_blocks[-1]))

    conv_blocks.append(BatchNormalization()(conv_blocks[-1]))

    conv_blocks.append(ReLU()(conv_blocks[-1]))

    conv_blocks.append(Conv2D(256, (3, 3), padding="valid", dilation_rate=(1,
1), name=f'Conv2D_256_2')(conv_blocks[-1]))

    conv_blocks.append(MaxPool2D((2, 2))(conv_blocks[-1]))

    conv_blocks.append(BatchNormalization()(conv_blocks[-1]))

    conv_blocks.append(ReLU()(conv_blocks[-1]))

    conv_blocks.append(Reshape([conv_blocks[-1].shape[1]*conv_blocks[-
1].shape[2], conv_blocks[-1].shape[3]])(conv_blocks[-1]))



    recurrent_layers.append(GRU(256, name=f'GRU_256')(conv_blocks[-1]))

    recurrent_layers.append(Flatten()(recurrent_layers[-1]))

    recurrent _outputs = Dense(output_shape[0], activation='sigmoid',
name=f'output')( recurrent_layers[-1])
```

```
model = Model(inputs=input, outputs= recurrent_outputs)
```

7.1.2. *Wynik działania algorytmu CRNN*

Trening modelu przeprowadzono w środowisku TensorFlow z wykorzystaniem pojedynczej karty graficznej RTX2070 i trwał on około 58 godzin. Podstawowe parametry treningu przedstawiono w tabeli 7.1.

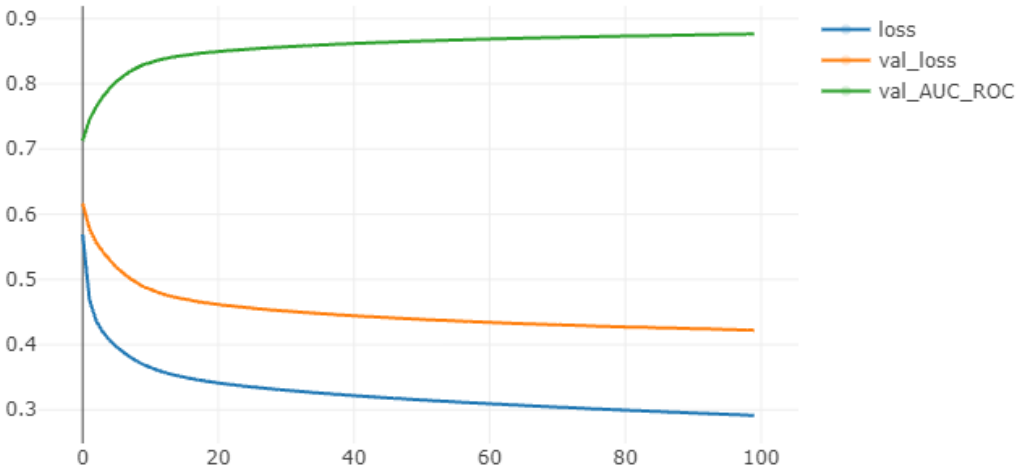
Tabela 7.1 Podstawowe parametry treningu modelu CRNN

Parametr	***Wartość***
Liczba epok	100
Współczynnik uczenia	0,001
Optymalizator	Adadelta [179]
Wielkość zestawu danych	32
Funkcja kosztu	binarna entropia krzyżowa [180]

Podczas treningu wartości funkcji kosztu na zbiorze treningowym i walidacyjnym zaczynały odpowiednio od poziomu 0,57 i 0,62, i spadły do poziomu 0,29 i 0,42. Dodatkowo

podczas treningu po każdej epoce wyznaczana była wartość miar AUC ROC na zbiorze walidacyjnym. Po pierwszej epoce osiągnęła ona wartość 0,71, a w końcowej fazie treningu 0,88.

Dokładne przebiegi funkcji kosztu na zbiorach treningowym oraz walidacyjnym, jak również miarę AUC ROC na zbiorze walidacyjnym przedstawiono na rysunku 7.2.



Rysunek 7.2 Wartości funkcji kosztu na zbiorach treningowym oraz walidacyjnym, jak również miary AUC ROC na zbiorze walidacyjnym

Wyniki ewaluacji przedstawiono w tabeli 7.2.

Tabela 7.2 Wyniki ewaluacji wytrenowanego modelu CRNN

Miara	***Wartość***
Prawdziwie pozytywne	78906
Prawdziwie negatywne	129225
Fałszywie negatywne	23016
Fałszywie pozytywne	28853
Precyzja	0,732
Czułość	0,774
AUC ROC	0,89
F1	0,753
LRAP	0,815

Dla modelu CRNN wyznaczono również macierz pomyłek, która została zaprezentowana na rysunku 7.3.

	Bas	Wiolonczela	Klarnet	Flet	Gitara	Harfa	Instrumenty perkusyjne	Fortepian	Rav vast	Santur	Saksofon	Trąbka	Skrzypce
Bas	85	8.4	8.8	9.2	17	11	8.9	14	11	11	8.7	7.8	10
Wiolonczela	14	76	16	17	20	23	18	18	23	23	19	16	20
Klarnet	14	17	75	17	21	22	16	18	23	22	19	16	21
Flet	14	15	17	72	21	21	16	19	22	21	17	15	19
Gitara	11	10	11	10	74	13	11	17	13	13	11	9.1	12
Harfa	13	14	15	16	19	68	14	16	20	19	16	14	19
Instrumenty perkusyjne	8.5	7.6	8	7.9	17	9.9	88	13	9.7	9.5	7.9	7.1	9.2
Fortepian	11	9.8	11	11	21	13	10	83	13	13	11	9.3	12
Rav vast	13	14	15	16	18	20	15	16	66	19	16	14	18
Santur	13	15	16	16	18	20	15	17	20	67	16	14	19
Saksofon	13	17	18	18	21	23	16	17	23	22	78	17	22
Trąbka	14	16	17	17	20	22	16	18	22	21	19	80	21
Skrzypce	14	15	17	16	20	22	18	18	22	21	18	16	76

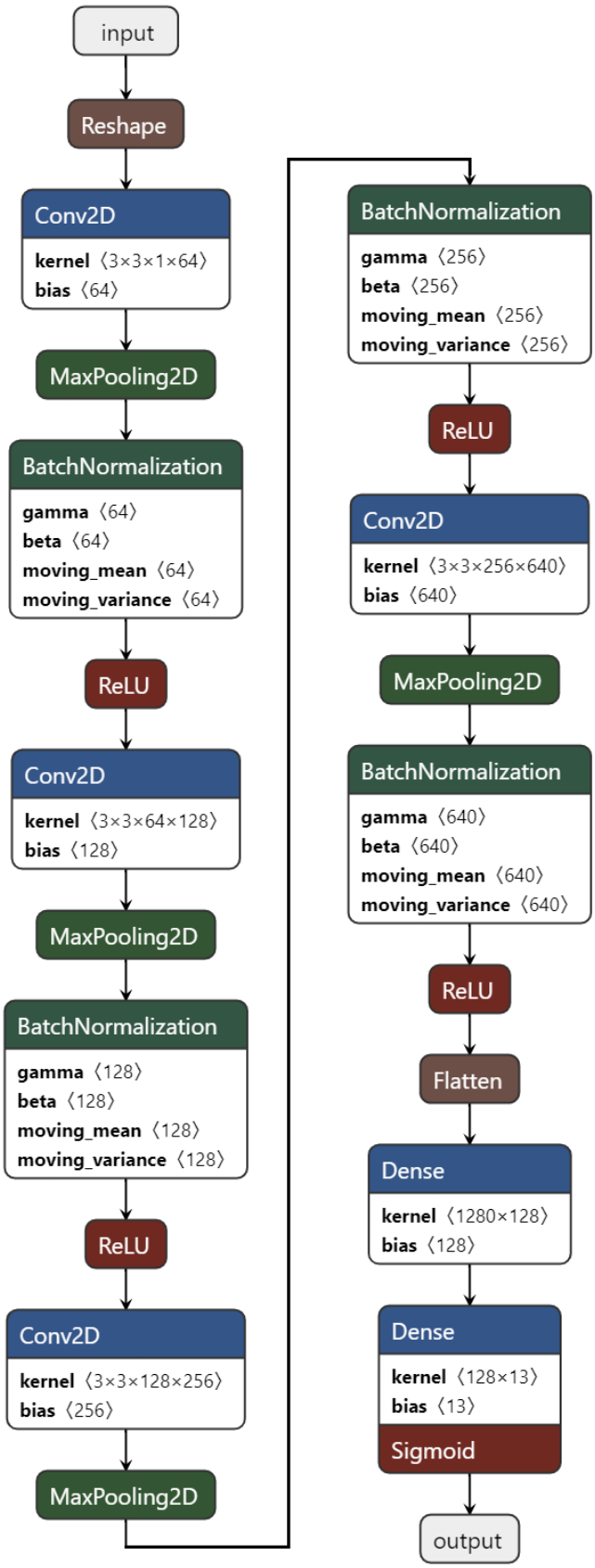
Rysunek 7.3 Macierz pomyłek dla modelu CRNN [%]

7.1. Implementacja modelu w architekturze CNN_1

7.1.1. *Opis architektury modelu CNN_1*

W eksperymentach przetestowano również model splotowy sieci neuronowej (CNN), który w eksperymentach Gururaniego i współautorów [38] uzyskał wyniki nieznacznie słabsze od opisanej wcześniej architektury CRNN. Poniższa część skupia się na przetestowaniu opisanej sieci CNN_1.

Na wejście modelu również przekazywany jest spektrogram w skali melowej składający się z 96 współczynników. Dalej zastosowano czterokrotne zestawienie dwuwymiarowej warstwy splotowej z warstwą normalizacji, ekstrakcji wartości maksymalnych i funkcji aktywacji ReLU. Warstwy splotowe zbudowane były w oparciu o jądro przekształcenia o wymiarze (3, 3) i zawierały kolejno 64. 128, 256 i 640 filtrów. Ostatnia część modelu CNN została zrealizowana pojedynczą warstwą w pełni połączoną składającą się z 128 jednostek. Wyjściem z modelu jest warstwa w pełni połączona posiadająca tyle wyjść, ile model obsługuje klas – w trenowanym przypadku 13. Funkcja aktywacji dla tej warstwy jest sigmoidalna. Na rysunku 7.4 przedstawiono schemat blokowy modelu.



Rysunek 7.4 Architektura modelu CNN_1 zaadaptowanego na podstawie pracy Gururaniego i współautorów [38]

Szczegółowy kod opisujący definicję testowanej architektury przedstawiono poniżej:

```
    input = Input(shape=input_shape)

    mfcc = MFCC(num_mel_bins=96)(input)

    conv_blocks = []

    dense_layers = []

    dense_outputs = []

    conv_blocks.append(Conv2D(64, (3, 3), padding="valid", dilation_rate=(1,
1), name=f'Conv2D_64')(reshape))

    conv_blocks.append(MaxPool2D((2, 2))(conv_blocks[-1]))

    conv_blocks.append(BatchNormalization()(conv_blocks[-1]))

    conv_blocks.append(ReLU()(conv_blocks[-1]))

    conv_blocks.append(Conv2D(128, (3, 3), padding="valid", dilation_rate=(1,
1), name=f'Conv2D_128')(conv_blocks[-1]))

    conv_blocks.append(MaxPool2D((2, 2))(conv_blocks[-1]))

    conv_blocks.append(BatchNormalization()(conv_blocks[-1]))

    conv_blocks.append(ReLU()(conv_blocks[-1]))

    conv_blocks.append(Conv2D(256, (3, 3), padding="valid", dilation_rate=(1,
1), name=f'Conv2D_256')(conv_blocks[-1]))

    conv_blocks.append(MaxPool2D((3, 3))(conv_blocks[-1]))

    conv_blocks.append(BatchNormalization()(conv_blocks[-1]))

    conv_blocks.append(ReLU()(conv_blocks[-1]))

    conv_blocks.append(Conv2D(640, (3, 3), padding="valid", dilation_rate=(1,
1), name=f'Conv2D_640')(conv_blocks[-1]))

    conv_blocks.append(MaxPool2D((3, 3))(conv_blocks[-1]))

    conv_blocks.append(BatchNormalization()(conv_blocks[-1]))

    conv_blocks.append(ReLU()(conv_blocks[-1]))

    conv_blocks.append(Flatten()(conv_blocks[-1]))



    dense_layers.append(Dense(128, name=f'Dense_128')(conv_blocks[-1]))

    dense_outputs = Dense(output_shape[0], activation='sigmoid',
name=f'output')(dense_layers[-1])
```

```
model = Model(inputs=input, outputs=dense_outputs)
```

7.1.2. *Wynik działania algorytmu CNN_1*

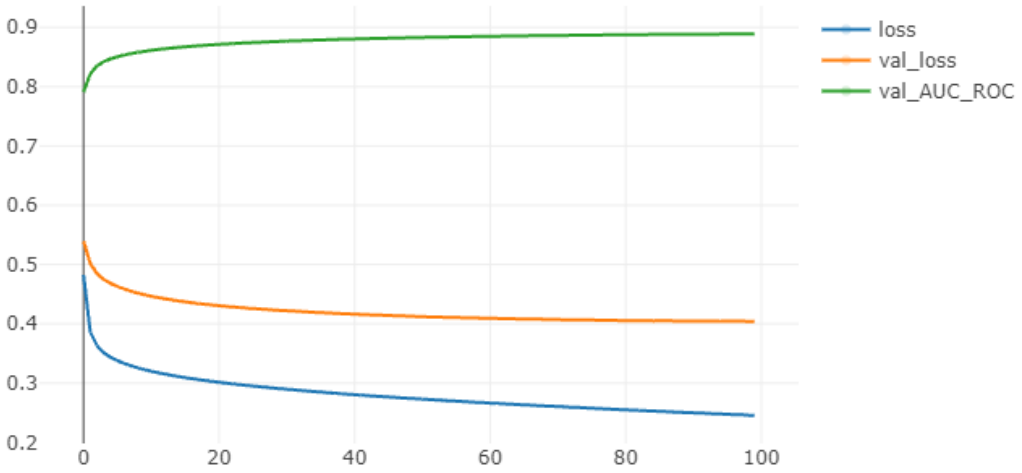
Trening modelu przeprowadzono w środowisku TensorFlow z wykorzystaniem pojedynczej karty graficznej RTX2070 i trwał on około 58 godzin. Podstawowe parametry przedstawiono w tabeli 7.3.

Tabela 7.3 Podstawowe parametry treningu modelu CNN_1

Parametr	***Wartość***
Liczba epok	100
Współczynnik uczenia	0,001
Optymalizator	Adadelta [179]
Wielkość zestawu danych	32
Funkcja kosztu	binarna entropia krzyżowa [180]

Podczas treningu wartości funkcji kosztu na zbiorze treningowym i walidacyjnym zaczynały się odpowiednio od poziomu 0,48 i 0,54, i spadły do poziomu 0,25 i 0,4. Dodatkowo podczas treningu − po każdej epoce wyznaczana − była wartość miary AUC ROC na zbiorze walidacyjnym. Po pierwszej epoce osiągnęła ona wartość 0,79, zaś w końcowej fazie treningu 0,89.

Dokładne przebiegi funkcji kosztu na zbiorze treningowym oraz walidacyjnym, jak również miary AUC ROC na zbiorze walidacyjnym przedstawiono na rysunku 7.5.



Rysunek 7.5 Wartości funkcji kosztu na zbiorze treningowym oraz walidacyjnym, jak również miary AUC ROC na zbiorze walidacyjnym

Na koniec treningu przeprowadzono ewaluację, której wyniki przedstawiono w tabeli 7.4. Osiągnięta przez model wartość miary LRAP na poziomie 0.853 udowadnia pierwszą tezę badawczą.

Tabela 7.4 Wyniki ewaluacji wytrenowanego modelu CNN_1

Miara	***Wartość***
Prawdziwie pozytywne	79478
Prawdziwie negatywne	131810
Fałszywie negatywne	22444
Fałszywie pozytywne	26268
Precyzja	0,752
Czułość	0,78
AUC ROC	0,902
F1	0,765
LRAP	0,853

Dla modelu CNN_1 wyznaczono również macierz pomyłek, która została zaprezentowana na rysunku 7.6.

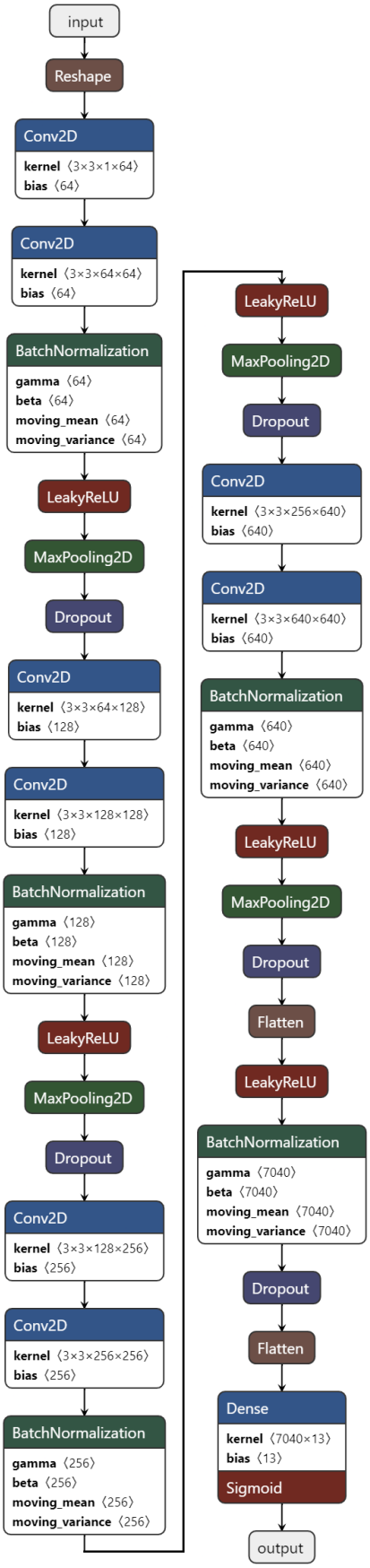
	Bas	Wiolonczela	Klarnet	Flet	Gitara	Harfa	Instrumenty perkusyjne	Fortepian	Rav vast	Santur	Saksofon	Trąbka	Skrzyp
Bas	85	6.4	6.5	7.1	14	10	6.8	9.6	11	11	7.2	6.1	8.1
Wiolonczela	11	75	13	15	17	20	13	13	21	21	15	13	16
Klarnet	11	14	73	16	17	20	13	12	22	21	16	13	17
Flet	11	13	14	76	17	19	13	13	20	20	14	12	15
Gitara	8.4	8.4	8.5	8.7	74	12	8.7	12	13	12	9.1	7.1	9.6
Harfa	10	12	12	13	16	67	11	11	20	19	13	11	15
Instrumenty perkusyjne	7	6.4	5.9	6.4	13	8.9	88	8.8	9.3	9.4	6.5	5.4	7.7
Fortepian	8.1	7.8	7.8	8.2	17	12	7.8	84	12	12	8.8	7.5	9.1
Rav vast	9.5	12	12	13	15	19	12	11	69	19	13	11	15
Santur	9.7	13	13	13	15	19	12	12	20	70	13	11	15
Saksofon	10	14	14	14	18	20	13	13	21	20	80	14	18
Trąbka	10	13	13	13	17	20	13	12	20	21	15	79	17
Skrzypce	11	13	13	13	17	19	14	13	20	20	15	13	75

Rysunek 7.6 Macierz pomyłek dla modelu CNN_1 [%]

7.2. Implementacja modelu w architekturze CNN_2

7.2.1. *Opis architektury modelu CNN_2*

Jak wspomniano wcześniej, w ramach eksperymentów, na podstawie pracy Kratimenosa i współautorów [55], zaimplementowano głęboki model CNN_2, który został opisany w poniższej części rozdziału. Na wejście modelu przekazywany jest sygnał foniczny przekształcony transformacją CQT. Dalej zastosowano czterokrotne zestawienie podwójnej dwuwymiarowej warstwy splotowej z warstwą normalizacji, funkcji aktywacji LeakuReLU, ekstrakcji wartości maksymalnych i warstwy porzucającej. Warstwy splotowe zbudowane były w oparciu o jądro przekształcenia o wymiarze (3, 3) i zawierały kolejno 64, 64, 128, 128, 256, 256, 640 i 640 filtrów. Pozostała cześć modelu została zrealizowana kolejno przy pomocy warstw: w pełni połączonej z 1024 jednostkami, warstwy aktywacji LeakyReLU, normalizacji i warstwy porzucającej. Wyjściem z modelu jest warstwa w pełni połączona posiadająca tyle wyjść, ile model obsługuje klas – w trenowanym przypadku 13. Funkcja aktywacji dla tej warstwy jest sigmoidalna. Na rysunku 7.7 przedstawiono schemat blokowy modelu sieci CNN_2, wykorzystanej w eksperymencie.



Rysunek 7.7 Architektura modelu CNN_2 zaadaptowanego na podstawie pracy Kratimenosa i współautorów [55]

Szczegółowy kod opisujący strukturę zaimplementowanego modelu przedstawiono poniżej:

```
    input = Input(shape=input_shape)

    conv_blocks = []

    dense_layers = []

    dense_outputs = []

    conv_blocks.append(Conv2D(64, (3, 3), padding="valid", dilation_rate=(1,
1), name=f'Conv2D_64')(reshape))

    conv_blocks.append(Conv2D(64, (3, 3), padding="valid", dilation_rate=(1,
1), name=f'Conv2D_64_1')(conv_blocks[-1]))

    conv_blocks.append(BatchNormalization()(conv_blocks[-1]))

    conv_blocks.append(LeakyReLU()(conv_blocks[-1]))

    conv_blocks.append(MaxPool2D((2, 2))(conv_blocks[-1]))

    conv_blocks.append(Dropout(0.2)(conv_blocks[-1]))

    conv_blocks.append(Conv2D(128, (3, 3), padding="valid", dilation_rate=(1,
1), name=f'Conv2D_128')(conv_blocks[-1]))

    conv_blocks.append(Conv2D(128, (3, 3), padding="valid", dilation_rate=(1,
1), name=f'Conv2D_128_1')(conv_blocks[-1]))

    conv_blocks.append(BatchNormalization()(conv_blocks[-1]))

    conv_blocks.append(LeakyReLU()(conv_blocks[-1]))

    conv_blocks.append(MaxPool2D((2, 2))(conv_blocks[-1]))

    conv_blocks.append(Dropout(0.2)(conv_blocks[-1]))

    conv_blocks.append(Conv2D(256, (3, 3), padding="valid", dilation_rate=(1,
1), name=f'Conv2D_256')(conv_blocks[-1]))

    conv_blocks.append(Conv2D(256, (3, 3), padding="valid", dilation_rate=(1,
1), name=f'Conv2D_256_1')(conv_blocks[-1]))

    conv_blocks.append(BatchNormalization()(conv_blocks[-1]))

    conv_blocks.append(LeakyReLU()(conv_blocks[-1]))

    conv_blocks.append(MaxPool2D((2, 2))(conv_blocks[-1]))

    conv_blocks.append(Dropout(0.2)(conv_blocks[-1]))

    conv_blocks.append(Conv2D(640, (3, 3), padding="valid", dilation_rate=(1,
1), name=f'Conv2D_640')(conv_blocks[-1]))

    conv_blocks.append(Conv2D(640, (3, 3), padding="valid", dilation_rate=(1,
1), name=f'Conv2D_640_1')(conv_blocks[-1]))
```

```
    conv_blocks.append(BatchNormalization()(conv_blocks[-1]))
    conv_blocks.append(LeakyReLU()(conv_blocks[-1]))
    conv_blocks.append(MaxPool2D((2, 2))(conv_blocks[-1]))
    conv_blocks.append(Dropout(0.2)(conv_blocks[-1]))
    conv_blocks.append(Flatten()(conv_blocks[-1]))
    conv_blocks.append(LeakyReLU()(conv_blocks[-1]))
    conv_blocks.append(BatchNormalization()(conv_blocks[-1]))
    conv_blocks.append(Dropout(0.5)(conv_blocks[-1]))
    conv_blocks.append(Flatten()(conv_blocks[-1]))

    dense_outputs = Dense(output_shape[0], activation='sigmoid',
name=f'output')(conv_blocks[-1])

    model = Model(inputs=input, outputs=dense_outputs)
```

7.2.2. *Omówienie wyników działania algorytmu CNN_2*

Trening modelu przeprowadzono w środowisku TensorFlow z wykorzystaniem pojedynczej karty graficznej RTX2070 i trwał on około 250 godzin. Podstawowe parametry modelu przedstawiono w tabeli 7.5.

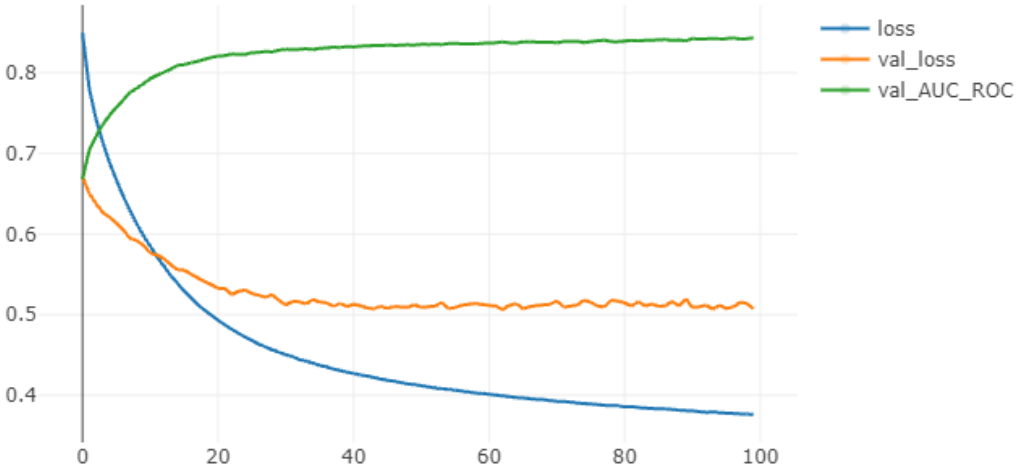
Tabela 7.5 Podstawowe parametry treningu modelu CNN_2

Parametr	***Wartość***
Liczba epok	100
Współczynnik uczenia	0,001
Optymalizator	Adadelta [179]
Wielkość zestawu danych	16
Funkcja kosztu	binarna entropia krzyżowa [180]

Podczas treningu wartości funkcji kosztu na zbiorze treningowym i walidacyjnym zaczynały odpowiednio od poziomu 0,84 i 0,67, i spadły do poziomu 0,38 i 0,51. Dodatkowo

podczas treningu po każdej epoce wyznaczana była wartość miary AUC ROC na zbiorze walidacyjnym. Po pierwszej epoce osiągnęła ona wartość 0,67, a w końcowej fazie treningu 0,84.

Dokładne przebiegi funkcji kosztu na zbiorze treningowym oraz walidacyjnym, jak również miary AUC ROC na zbiorze walidacyjnym przedstawiono na rysunku 7.8. Na koniec treningu przeprowadzono ewaluację, której wyniki przedstawiono w postaci tabeli 7.6.



Rysunek 7.8 Wartości funkcji kosztu na zbiorze treningowym oraz walidacyjnym, jak również miary AUC ROC na zbiorze walidacyjnym

Tabela 7.6 Wyniki ewaluacji wytrenowanego modelu CNN_2

Miara	***Wartość***
Prawdziwie pozytywne	86188
Prawdziwie negatywne	113232
Fałszywie negatywne	15734
Fałszywie pozytywne	44846
Precyzja	0,752
Czułość	0,78
AUC ROC	0,862
F1	0,74
LRAP	0,769

Dla modelu CNN_2 wyznaczono również macierz pomyłek, która została zaprezentowana na rysunku 7.9.

	Bas	Wiolonczela	Klarnet	Flet	Gitara	Harfa	Instrumenty perkusyjne	Fortepian	Rav vast	Santur	Saksofon	Trąbka	Skrzyn
Bas	87	13	13	15	22	15	11	21	16	15	12	15	16
Wiolonczela	18	88	22	26	23	30	21	25	32	31	25	29	32
Klarnet	17	23	87	26	23	29	21	24	32	31	25	29	32
Flet	17	22	25	83	24	28	20	27	30	29	24	27	29
Gitara	12	15	16	17	77	18	13	24	18	17	14	17	19
Harfa	15	20	21	23	21	80	18	22	28	27	21	25	29
Instrumenty perkusyjne	9.8	12	12	13	20	13	81	19	14	13	10	13	14
Fortepian	12	14	14	17	26	17	14	89	18	17	14	16	18
Rav vast	14	20	21	23	20	27	18	22	82	26	21	25	29
Santur	15	20	22	23	20	27	18	22	28	83	22	25	29
Saksofon	17	25	26	28	24	31	21	25	32	31	87	31	34
Trąbka	16	22	24	26	23	29	20	24	30	29	25	91	31
Skrzypce	16	20	23	24	22	28	21	24	29	29	24	29	86

Rysunek 7.9 Macierz pomyłek dla modelu CNN_2 [%]

8. IDENTYFIKACJA INSTRUMENTU MUZYCZNEGO

Pierwszym etapem w łańcuchu przetwarzania zbudowanym na potrzeby niniejszej pracy jest identyfikator potrafiący ocenić, jakie instrumenty występują w nagraniu muzycznym. Na jego podstawie możliwe jest odpowiednie ustawienie wag kolejnej sieci neuronowej, tym razem realizującej już właściwą separację instrumentu. Poniższy rozdział opisuje budowę oraz działanie takiego identyfikatora.

8.1. Scenariusz przeprowadzonych eksperymentów

Zaprezentowane w pracy podejście do problemu identyfikacji instrumentu muzycznego ma charakter iteracyjny, oznacza to kolejne etapy przetwarzania i wykorzystania sieci neuronowej, ostatecznie przygotowanej jako wynik całościowego eksperymentu. W początkowym etapie wykorzystano architektury zaproponowane w literaturze jako metody *state-of-the-art* zarówno w kontekście klasyfikacji, jak i identyfikacji instrumentów muzycznych, ale zmodyfikowane w taki sposób, aby oznaczać jednocześnie wiele instrumentów występujących w danej próbce dźwiękowej. W kolejnym kroku, również bazując na wytypowanej technologii, przygotowano model składający się z osobnych identyfikatorów dla każdego z badanych instrumentów, tworząc tym samym sieć nazwaną jako model rozproszony, tj. FMCNN (ang. *Fully Modular Convolutional Neural Network).* W ostatnim etapie przygotowano nową sieć w technologii SNN.

8.2. Model rozproszony

Analizując zasadę działania opisanych w literaturze identyfikator, zaproponowano rozwiązanie mające na celu bardziej skuteczną identyfikację poszczególnych instrumentów. Ideą zaproponowanego identyfikatora jest rozbicie modelu na mniejsze części, z których każda odpowiada za tylko jeden instrument. Takie podejście ma zapewnić lepszą ekstrakcję cech wysokiego poziomu z podanego na model wejścia, a co za tym idzie dokładniejsze wyniki identyfikacji.

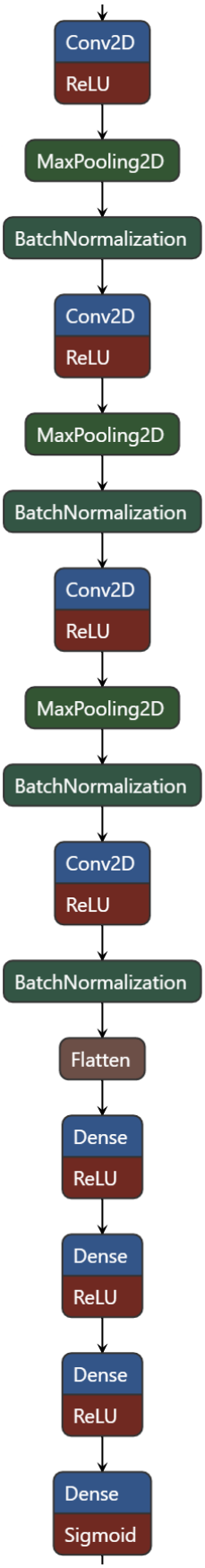
8.2.1. *Opis architektury identyfikatora*

Zaproponowana architektura wykorzystuje klasyczne sieci splotowe, ale jak wspomniano wcześniej, każdy z instrumentów ma osobny tor przetwarzania, który został zaprezentowany na rysunku 8.1. Wejściem na taki blok jest sparametryzowany sygnał, złożony z 40 współczynników mel-cepstralnych (MFCC). Kolejne warstwy ułożone są zgodnie z tabelą 8.1.

Tabela 8.1 Opis bloku dla pojedynczego instrumentu w modelu rozproszonym

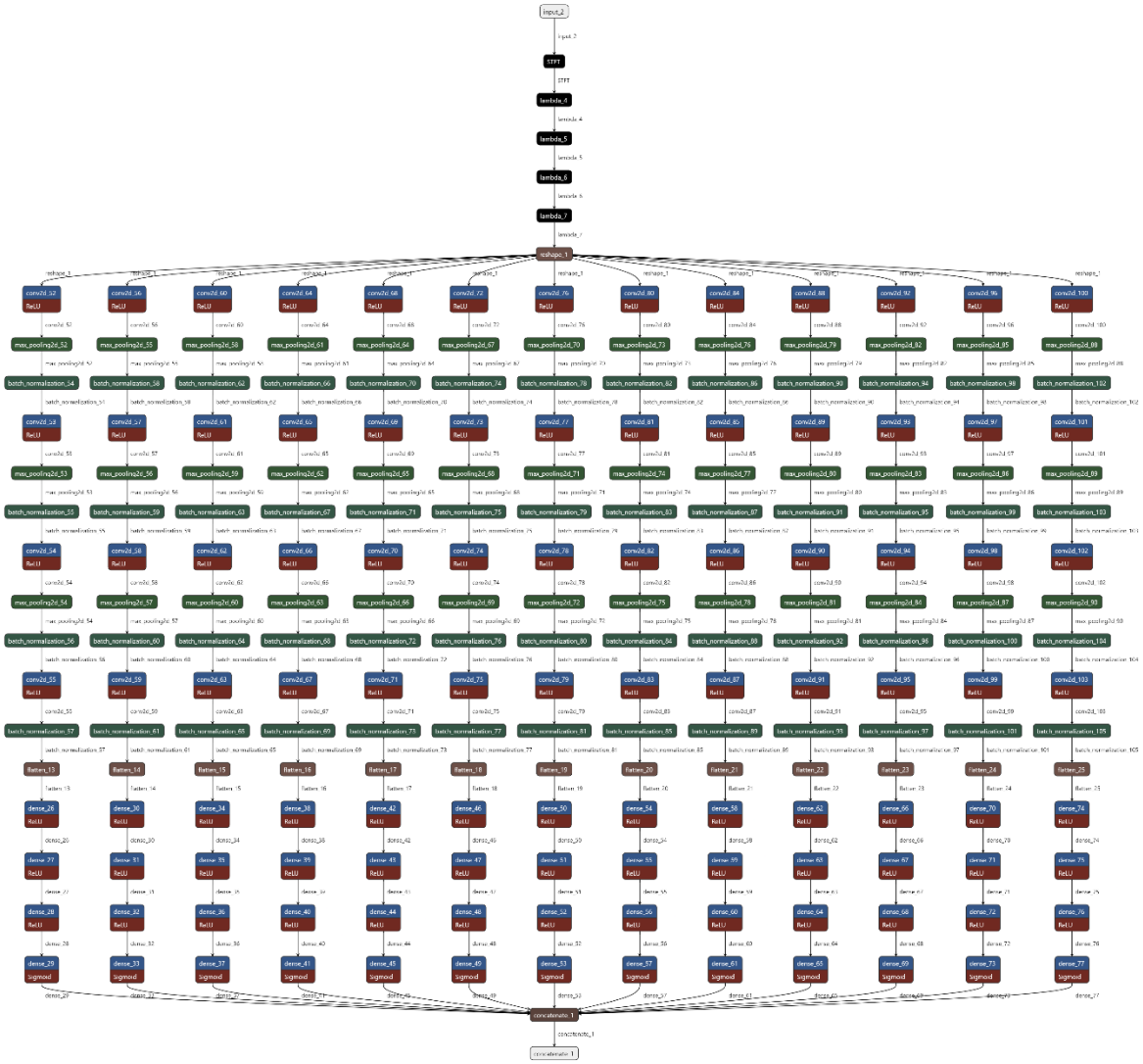
Typ warstwy	*Jądro przekształcenia*	*Liczba jednostek*	*Funkcja aktywacji*
Splot dwuwymiarowy	3 x 3	64	
Ekstrakcja wartości maksymalnej	2 x 2		

Normalizacja			
Splot dwuwymiarowy	3 x 3	32	
Ekstrakcja wartości maksymalnej	2 x 2		
Normalizacja			
Splot dwuwymiarowy	3 x 3	16	
Ekstrakcja wartości maksymalnej	2 x 2		
Normalizacja			
Splot dwuwymiarowy	3 x 3	8	
Normalizacja			
W pełni połączona		64	ReLU
W pełni połączona		32	ReLU
W pełni połączona		16	ReLU
W pełni połączona		1	sigmoida



Rysunek 8.1 Schemat blokowy ścieżki przetwarzania pojedynczego instrumentu w modelu o charakterze modularnym (FMCNN)

Wykorzystując dokładnie tyle bloków, ile instrumentów będzie rozpoznawanych w eksperymencie (tj. 13), zbudowano model, którego architektura została przedstawiona na rysunku 8.2.



Rysunek 8.2 Architektura testowanego modelu modularnego

8.2.2. *Trening identyfikatora*

Trening modelu przeprowadzono w środowisku TensorFlow z wykorzystaniem pojedynczej karty graficznej RTX2070 i trwał on ok. 29 godzin. Podstawowe parametry treningu przedstawiono w tabeli 8.2.

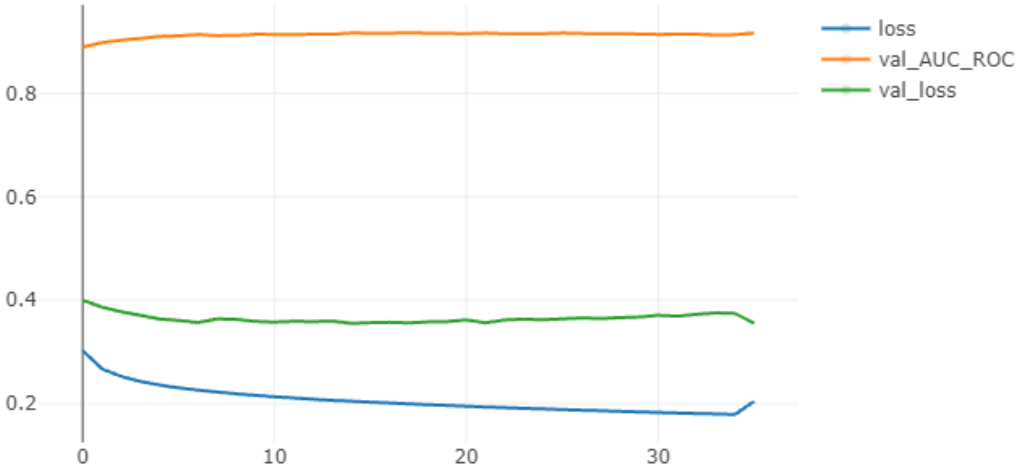
Tabela 8.2 Podstawowe parametry treningu modelu FMCNN

Parametr	***Wartość***
Liczba epok	100

Współczynnik uczenia	0,001
Optymalizacja	Adam [181]
Wielkość zestawu danych	32
Funkcja kosztu	binarna entropia krzyżowa [180]

Podczas treningu wartości funkcji kosztu na zbiorze treningowym i walidacyjnym startowały odpowiednio od poziomu 0,3 i 0,4, i spadły do poziomu 0,18 i 0,37. Dodatkowo podczas treningu po każdej epoce wyznaczana była wartość miary AUC ROC na zbiorze walidacyjnym. Po pierwszej epoce osiągnęła ona wartość 0,89, a w końcowej fazie treningu 0,92.

Dokładne przebiegi funkcji kosztu na zbiorze treningowym oraz walidacyjnym, jak również miary AUC ROC na zbiorze walidacyjnym przedstawiono na rysunku 8.3.



Rysunek 8.3 Wartości funkcji kosztu na zbiorze treningowym oraz walidacyjnym, jak również miary AUC ROC na zbiorze walidacyjnym

8.2.3. *Ewaluacja identyfikatora*

Wynik ewaluacji badanego modelu przedstawiono w tabeli 8.3. Osiągnięta przez model wartość miary LRAP na poziomie 0,895 udowadnia trzecią tezę badawczą.

Tabela 8.3 Wyniki ewaluacji wytrenowanego modelu rozproszonego

Miara	***Wartość***
Prawdziwie pozytywne	82301

Prawdziwie negatywne	135940
Fałszywie negatywne	19621
Fałszywie pozytywne	22138
Precyzja	0,752
Czułość	0,78
AUC ROC	0,926
F1	0,798
LRAP	0,895

Dla modelu FMCNN wyznaczono również macierz pomyłek, która została zaprezentowana na rysunku 8.4.

	Bas	Wiolonczela	Klarnet	Flet	Gitara	Harfa	Instrumenty perkusyjne	Fortepian	Rav vast	Santur	Saksofon	Trąbka	Skrzy
Bas	86	5.1	5.6	6.3	11	9.3	3.3	5.9	10	9.7	4.6	3	6.8
Wiolonczela	7.3	76	11	16	15	19	6.8	8.8	21	20	10	6	14
Klarnet	7.5	13	72	17	17	19	6.3	8.2	21	19	9.9	6	14
Flet	7.3	11	12	81	15	18	6	9.2	20	18	9.3	5.6	12
Gitara	5.1	6.2	6.8	7.7	83	11	4.6	6.9	12	11	5.9	3.5	8.1
Harfa	6.4	10	10	13	14	70	5.5	7.6	19	18	8.4	5.2	12
Instrumenty perkusyjne	4.3	5	5.1	5.9	11	8.6	91	5.4	9.5	8.8	4.4	2.8	6.5
Fortepian	5.3	6.2	6.5	7.2	14	11	4	86	12	11	5.9	3.6	7.6
Rav vast	6.2	9.8	10	13	14	18	6.4	7.6	73	17	8	5	12
Santur	6.4	10	11	13	14	18	6.2	7.9	19	76	8.4	5.1	12
Saksofon	7.8	12	12	14	17	20	7.1	9.2	22	19	78	6.6	15
Trąbka	7.1	12	12	13	16	20	6.8	9	21	19	10	77	14
Skrzypce	7.1	11	11	13	15	19	7.3	8.7	21	19	10	6.3	81

Rysunek 8.4 Macierz pomyłek dla modelu FMCNN [%]

8.3. Identyfikator impulsowy

Kolejnym badanym algorytmem jest impulsowa sieć neuronowa (SNN). W kolejnych podrozdziałach zostanie przedstawiony proces treningu, ewaluacji oraz identyfikacji.

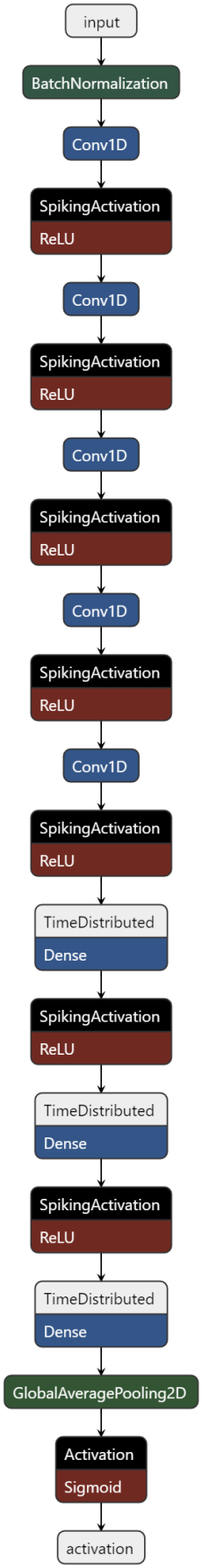
8.3.1. *Opis architektury identyfikatora instrumentów muzycznych*

Zaprojektowany model wykorzystuje jako połączenia synaptyczne jednowymiarowe warstwy splotowe, a jako ciało neuronu funkcję aktywacji opartą na ReLU. Wejściem modelu jest sekwencja ramek MFCC, zawierająca 40 współczynników w skali melowej. Zgodnie z ideą sieci impulsowych warstwy ułożone są kolejno w szeregu synapsa, ciało komórki, synapsa, …, ciało komórki. W tabeli 8.4 zaprezentowano dokładny opis kolejnych warstw wraz z ich parametrami.

Tabela 8.4 Opis bloku dla pojedynczego instrumentu w modelu impulsowym

Typ warstwy	*Jądro przekształcenia*	*Liczba jednostek*	*Funkcja aktywacji*
Normalizacja			
Splot jednowymiarowy	3	40	
Impulsowa funkcja aktywacji			ReLU
Splot jednowymiarowy	3	64	
Impulsowa funkcja aktywacji			ReLU
Splot jednowymiarowy	3	128	
Impulsowa funkcja aktywacji			ReLU
Splot jednowymiarowy	3	126	
Impulsowa funkcja aktywacji			ReLU
Splot jednowymiarowy	3	512	
Czasowo zależna warstwa w pełni połączona		128	
Impulsowa funkcja aktywacji			ReLU
Czasowo zależna warstwa w pełni połączona		128	
Impulsowa funkcja aktywacji			ReLU
Czasowo zależna warstwa w pełni połączona		128	
Warstwa uśredniająca wartość po czasie			
Funkcja aktywacji			Sigmoida

Rysunek 8.5 przedstawia schemat blokowy zaproponowanej architektury sieci impulsowej (SNN).



Rysunek 8.5 Schemat blokowy modelu w architekturze SNN

8.3.2. *Trening impulsowego identyfikatora instrumentów muzycznych*

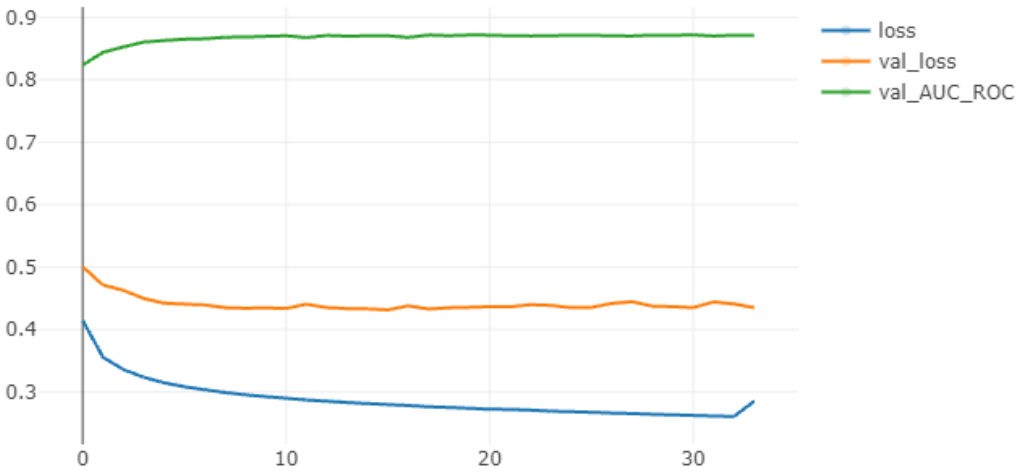
Trening modelu przeprowadzono w środowisku TensorFlow oraz KerasSpiking z wykorzystaniem pojedynczej karty graficznej RTX2070 i trwał on ok. 106 godzin. Podstawowe parametry przedstawiono w tabeli 8.5.

Tabela 8.5 Podstawowe parametry treningu modelu impulsowego

Parameter	***Wartość***
Liczba epok	100
Współczynnik uczenia	0,001
Optymalizator	Adam [181]
Wielkość zestawu danych	32
Funkcja kosztu	binarna entropia krzyżowa [180]

Podczas treningu wartości funkcji kosztu na zbiorze treningowym i walidacyjnym startowały odpowiednio od poziomu 0,41 i 0,5, i spadły do poziomu 0,26 i 0,45. Dodatkowo podczas treningu po każdej epoce wyznaczana była wartość miary AUC ROC na zbiorze walidacyjnym. Po pierwszej epoce osiągnęła ona wartość 0.82, a w końcowej fazie treningu 0.97.

Dokładne przebiegi funkcji kosztu na zbiorze treningowym oraz walidacyjnym, jak również miarę AUC ROC na zbiorze walidacyjnym przedstawiono na rysunku 8.6.



Rysunek 8.6 Wartości funkcji kosztu na zbiorze treningowym oraz walidacyjnym, jak również miary AUC ROC na zbiorze walidacyjnym

8.3.3. *Ewaluacja impulsowego identyfikatora instrumentów muzycznych*

Wynik ewaluacji modelu przedstawiono w tabeli 8.6.

Osiągnięta przez model wartość miary LRAP na poziomie 0.844 przy jednoczesnym dwukrotnym zmniejszeniu rozmiarów sieci udowadnia drugą tezę badawczą.

Tabela 8.6 Wyniki ewaluacji wytrenowanego modelu impulsowego

Miara	***Wartość***
Prawdziwie pozytywne	74872
Prawdziwie negatywne	131893
Fałszywie negatywne	27050
Fałszywie pozytywne	26185
Precyzja	0,752
Czułość	0,78
AUC ROC	0,887
F1	0,738
LRAP	0,844

Dla modelu SNN wyznaczono również macierz pomyłek, która została zaprezentowana na rysunku 8.7.

	Bas	Wiolonczela	Klarnet	Flet	Gitara	Harfa	Instrumenty perkusyjne	Fortepian	Rav vast	Santur	Saksofon	Trąbka	Skrzyp
Bas	81	6.8	6.9	8.1	16	8.3	6.6	13	9.3	9.5	5.2	5	6.3
Wiolonczela	13	70	12	17	20	17	14	18	20	20	12	11	13
Klarnet	13	14	68	18	21	18	14	18	20	20	12	11	14
Flet	13	13	15	76	20	17	13	19	18	19	11	10	12
Gitara	9.3	7.7	8.2	9.3	76	9.6	8	15	11	11	6.8	6.2	7.6
Harfa	12	12	13	15	18	54	11	16	17	18	9.7	9.2	11
Instrumenty perkusyjne	8.8	6.8	6.5	7.5	15	7.4	83	12	8.3	8.6	5	4.7	6.1
Fortepian	9.4	7.8	8	9.2	18	9.9	7.3	80	11	11	6.5	6	7.2
Rav vast	11	12	12	14	18	15	12	16	61	16	9.4	9	11
Santur	11	11	12	14	18	15	12	16	17	69	9.6	8.8	12
Saksofon	13	14	15	16	21	18	14	18	19	19	71	11	14
Trąbka	13	14	14	15	20	18	12	17	19	19	12	75	13
Skrzypce	14	13	14	15	20	17	14	17	18	19	12	11	65

Rysunek 8.7 Macierz pomyłek dla modelu SNN [%]

9. ANALIZA WYNIKÓW

W niniejszym rozdziale − na podstawie wyników zaprezentowanych w rozdziałach 7 i 8 – zestawiono tabele 9.1, 9.2 i 9.3, podsumowujące takie aspekty eksperymentów, jak parametry badanych architektur sieci neuronowych, bezpośrednie wyniki ewaluacji wytrenowanych modeli oraz wyznaczone na ich podstawie miary.

Pierwszym analizowanym aspektem w kontekście przeprowadzonych eksperymentów jest rozmiar badanej architektury. W tabeli 9.1 zaprezentowano porównanie liczby warstw, parametrów oraz czasu treningu poszczególnych modeli. Zauważyć można, że trening pozornie największego i najbardziej skomplikowanego modelu, jakim jest zaproponowana sieć typu FMCNN trwał najkrócej. Wynika to z faktu zastosowania relatywnie małych, ale wyspecjalizowanych warstw, dzięki czemu obliczenia mogły zostać wykonane szybciej.

Kolejną obserwacją na podstawie tabeli 9.1 jest długi czas treningu sieci SNN, mimo że ma ona najmniej parametrów. W tym przypadku spowodowane jest to zastosowaniem komórek pamięci wewnątrz ciała komórki. Spowalnia to cały proces uczenia, gdyż wagi są uzależnione od stanu wewnątrz neuronu, który jest przeliczany dla sekwencji.

Najdłuższym czasem treningu charakteryzuje się model CNN (CNN_2) zaadaptowany ze stanu wiedzy [55], który składa się z dużych warstw splotowych. Powoduje to spowolnienie obliczeń. Dodatkowo tak duży model wymagał obniżenia wielkości zestawu danych przekazywanych jednocześnie, co zwiększa czas obliczeń ze względu na częstszą komunikację pomiędzy procesorem obliczającym miary oraz funkcje kosztu a kartą graficzną, na której uruchamiany jest model.

Tabela 9.1 Porównanie parametrów modeli state-of-the-art z badanymi i zaproponowanymi architekturami

Model	*Liczba parametrów*	*Liczba warstw*	*Długość treningu [h]*
State-of-the-art CNN_1 [38]	2 014 861	26	58
State-of-the-art CRNN [38]	1 360 653	27	58
State-of-the-art CNN_2 [55]	6 430 541	32	250
FMCNN	26 173 069	255	29
SNN	599 665	25	106

Kolejnym analizowanym elementem jest macierz pomyłek dla każdego z zaproponowanych modeli. Składa się ona z liczby przykładów, które zostały zidentyfikowane jako prawdziwie pozytywne i negatywne oraz fałszywie pozytywne i negatywne. Wyniki te zostały

przedstawione w tabeli 9.2. Najwięcej próbek zidentyfikowanych prawdziwie pozytywnie wykazał model CNN_2 zaadaptowany ze stanu wiedzy [55], ale jednocześnie wykazuje on tendencję do przypisywania próbek fałszywie pozytywnych. Zaproponowany model FMCNN bardzo dobrze poradził sobie zarówno w kontekście próbek rozpoznanych jako prawdziwie pozytywne i prawdziwie negatywne. Architektura oparta na sieci impulsowej wykazuje tendencję do odrzucania przykładów, co widać w liczbie zidentyfikowanych prawdziwie i fałszywie negatywnie.

Tabela 9.2 Porównanie macierzy pomyłek dla modeli state-of-the-art z badanymi i zaproponowanymi architekturami

Model	Prawdziwie pozytywne	Prawdziwie negatywne	Fałszywie negatywne	Fałszywie pozytywne
State-of-the-art CNN_1 [38]	79478	131810	22444	26268
State-of-the-art CRNN [38]	78906	129225	23016	28853
State-of-the-art CNN_2 [55]	86188	113232	15734	44846
FMCNN	82301	135940	19621	22138
SNN	74872	131893	27050	26185

Bazując na wynikach przedstawionych w tabeli 9.2, wyznaczono miary takie jak precyzja, czułość, F1, LRAP oraz wartość AUC pod krzywą ROC (tabela 9.3). Spośród badanych modeli najwyższą jakość identyfikacji osiągnął model FMCNN, wykazując drugą (najwyższą) wartość jedynie dla czułości. Drugą architekturą jest sieć CNN zaadaptowana ze stanu wiedzy [38]. Warto zwrócić uwagę w stronę modelu impulsowego (SNN), który pomimo swojego rozmiaru osiągnął bardzo dobre wyniki dając rezultaty lepsze niż większość modeli state-of-the-art.

Tabela 9.3 Porównanie wynikowych miar dla modeli state-of-the-art z badanymi i zaproponowanymi architekturami

Model	Precyzja	Czułość	AUC ROC	F1	LRAP
State-of-the-art CNN [38]	0,752	0,78	0,902	0,765	0,853
State-of-the-art CRNN [38]	0,732	0,774	0,89	0,753	0,815
State-of-the-art CNN_2 [55]	0,658	0,891	0,862	0,74	0,769
FMCNN	0,791	0,811	0,926	0,798	0,895
SNN	0,741	0,735	0,887	0,738	0,844

10. WNIOSKI I PODSUMOWANIE

Zaprezentowane w niniejszej rozprawie eksperymenty skupiły się wokół analizy możliwości zastosowania sztucznych sieci neuronowych do celów automatycznej identyfikacji instrumentów muzycznych w sygnale, złożonym z wielu instrumentów. Przegląd literatury wskazuje na ograniczoną liczbę prac zajmujących się tym tematem. Większość autorów swoją sferę zainteresowania przeniosła na detekcję instrumentu dominującego, na przykładzie zbioru IRMAS [4][29][39][81][117]. Jednocześnie zaprezentowane przez autorów wyniki wskazują na dużą trudność zarówno w przypadku identyfikacji dominującego, jak i wszystkich instrumentów.

Pierwszym etapem rozprawy było sprawdzenie słuszności koncepcji identyfikatora rozproszonego na bardzo ograniczonym zbiorze treningowym (jedynie cztery instrumenty), która potwierdziła możliwość zastosowania tego typu modelu w zadaniu identyfikacji wielu instrumentów jednocześnie. Dla małego zbioru danych osiągnięto wartości miary AUC ROC na poziomie około 0,96 oraz wskaźnika F1 na poziomie około 0,93. Wartości te przewyższyły metody *state-of-the-art,* które dla tych miary osiągnęły wartości odpowiednio ok. 0,91 i ok. 0,64.

Bazując na zdobytych doświadczeniach, przygotowano pełny zbiór treningowy zawierający większą liczbę instrumentów, w tym nietypowe takie jak santur i rav vast. Takie działanie miało za zadanie zwiększenie trudności eksperymentu oraz sprawdzenie potencjału zaproponowanych rozwiązań. Wykorzystując ten zbiór, w pierwszej kolejności wytrenowano modele zaadaptowane ze stanu wiedzy (*state-of-the-art*), aby mieć dokładny punkt odniesienia dla architektur FMCNN oraz SNN. Jednocześnie już podczas tych eksperymentów pokazano, że odpowiednio przygotowany zbiór treningowy pozwala na osiągnięcie wartości miary LRAP wyższej niż 0,8, co pozwoliło na potwierdzenie pierwszej tezy rozprawy w brzmieniu:

1. *Możliwe jest przygotowanie i wytrenowanie takiej sieci neuronowej, która zrealizuje identyfikację instrumentów występujących w sygnale fonicznym, uzyskując wynik miary opisującej jakość identyfikacji w kontekście zadania z wieloma klasami występującymi w danej próbce (ang. Label ranking average precision, LRAP) wyższy niż 0,85, co przewyższa aktualny stan wiedzy.*

W kolejnym etapie badań, bazując na wcześniejszych doświadczeniach, przeprowadzono szereg eksperymentów mających na celu przygotowanie i wytrenowanie modelu zgodnego z zaproponowaną architekturą FMCNN. Proces ewaluacji wskazał wysoką zdolność do identyfikacji instrumentów w sygnale polifonicznym, udokumentowaną osiągnięciem miary LRAP na poziomie 0.895. W ten sposób udowodniono drugą tezę rozprawy doktorskiej:

2. *Zastosowanie w zadaniu identyfikacji instrumentów sieci neuronowej składającej się z mniejszych podsieci wyspecjalizowanych w odniesieniu do badanego instrumentu pozwala zwiększyć skuteczność identyfikacji definiowaną miarą LRAP do poziomu 0,89.*

Ostatnią badaną technologią były sieci impulsowe, traktowane obecnie jako sieci następnej generacji. Ich zastosowanie pozwoliło na osiągnięcie wysokiej jakości identyfikacji (miara LRAP na poziomie 0.844 dla 13 rozpoznawanych instrumentów muzycznych), przy

jednoczesnym znacznym zmniejszeniu liczby parametrów modelu. W stosunku do architektury modelu CNN zaadaptowanego ze stanu wiedzy [38], której wyniki są najbliższe wytrenowanemu modelowi, liczba parametrów zmniejszyła się prawie czterokrotnie. Osiągnięcie tak wysokiego wyniku jakości identyfikacji przy tak małym modelu pozwala na udowodnienie trzeciej tezy rozprawy:

3. *Sieci neuronowe 3. generacji (ang. Spiking Neural Networks, SNN) pozwalają na uzyskanie skuteczności identyfikacji instrumentu na poziomie miary LRAP 0,84 przy jednoczesnym dwukrotnym zmniejszeniu liczby parametrów sieci neuronowej.*

10.1. Wnioski dotyczące trudności w realizacji rozprawy

Podczas realizacji eksperymentów napotkano szereg problemów na różnych etapach realizacji, z których każdy wymagał nieco odmiennego sposobu rozwiązania. Pierwszym z nich była ograniczona moc obliczeniowa zasobów komputerowych, na których prowadzono eksperymenty. Skutkowało to w szczególności dwoma problemami:

1. Trening modeli trwał bardzo długo, co powodowało, że kolejne eksperymenty, modyfikujące pierwotne założenia należało zawiesić nawet na kilkanaście dni;
2. Już na etapie projektowania niezbędne było wprowadzenie optymalizacji modeli, tak aby można było je pomieścić w pamięci karty graficznej, oraz aby trening został wykonany w akceptowalnym czasie.

Kolejnym problemem był brak dostępu do wytrenowanych modeli ze stanu wiedzy (*state–of–the–art*), które miały zostać zaadaptowane w badaniach prowadzonych przez autora rozprawy doktorskiej. Wymagało to przygotowania implementacji, bazując jedynie na opisie zawartym w publikacji [38][55], a następnie wytrenowaniu opisanych architektur. Równocześnie wykorzystane w literaturze zbiory treningowe były bardzo okrojone i proste. Wymagało to dokładnego zagłębienia się w dane i przygotowania nowego zbioru danych.

Ostatnim problemem było przeprowadzenie treningu modelu impulsowego SNN, ze względu na zupełnie nowe podejście w trenowaniu sieci oraz ograniczoną dostępność narzędzi, które umożliwiają obsługę tego typu architektur.

10.2. Osiągnięcia rozprawy doktorskiej

W czasie realizacji rozprawy autor uzyskał kilka osiągnięć pozwalających na rozwój dziedziny związanej z tematyką pracy:

1. Przygotowanie zbioru danych, którego zastosowanie w treningu pozwoliło na uzyskanie wyższych wyników dla metod state-of-the-art niż zaprezentowane przez autorów algorytmów.
2. Realizacja eksperymentów na dużych sieciach neuronowych z wykorzystaniem sprzętu klasy komercyjnej, co pozwoliło na zbadanie większych struktur sieci neuronowych.

3. Zaproponowanie modelu sieci neuronowej posiadającego znacznie mniejszy rozmiar niż metody state-of-the-art przy jednoczesnym braku utraty jakości identyfikacji.
4. Zaproponowanie architektury modelu działającego w technologii sieci impulsowych określanych mianem „sieci neuronowych 3. generacji".
5. Zaproponowanie innowacyjnej architektury FMCNN znacznie przewyższającej jakością działania metody state-of-the-art.
6. Przygotowanie zróżnicowanego i rozbudowanego zbioru treningowego przy pomocy zaimplementowanych automatycznych narzędzi miksujących.
7. Przygotowanie pełnego eksperymentu składającego się z treningu i ewaluacji zaproponowanych modeli.
8. Zaimplementowanie systemu logowania wyników treningów i ewaluacji za pomocą narzędzia MlFlow.

10.3. Dalsze kierunki prac

Zaprezentowane w rozprawie wyniki wskazują na wiele obiecujących kierunków rozwoju przeanalizowanych typów sieci neuronowych. Pierwszym krokiem może być rozszerzenie zbioru treningowego zarówno o pojedyncze dźwięki nowych instrumentów, jak i dodanie kolejnych utworów muzycznych.

Architektura FMCNN (model modularny) zaproponowana w rozprawie doktorskiej może znaleźć zastosowanie w zadaniach identyfikacji danych nie tylko w dziedzinie przetwarzania sygnałów fonicznych. Możliwe jest zastosowanie jej np. w kontekście identyfikacji obiektów w obrazie.

Można zauważyć, że modele SNN, traktowane jako sieci neuronowe 3. generacji, z jednej strony powracają do korzeni uczenia maszynowego i sztucznych sieci neuronowych, z drugiej zaś wykorzystują doświadczenia badaczy z ubiegłych trzech dekad, co wskazuje na duże możliwości w kontekście przetwarzania sygnałów fonicznych. Można przewidywać, że w najbliższych latach modele te będą intensywnie badane w kontekście ich możliwości i ograniczeń w tym obszarze i pokrewnych dziedzinach.

Literatura opisująca metody transkrypcji instrumentów w sygnale monofonicznym oraz separacji źródeł w sygnale polifonicznym wskazuje na wykorzystanie metod klasyfikacji instrumentów jako jeden z elementów całego proponowanego rozwiązania. Co za tym idzie, warte rozważenia jest zweryfikowanie jakości identyfikacji instrumentów w kontekście wyżej wymienionych zadań.

11. WYKAZ LITERATURY

[1] M. Abeles, H. Bergman, E. Margalit, E. Vaadia, Spatiotemporal firing patterns in the frontal cortex of behaving monkeys, J Neurophysiol, 70, 4, 1629–1638, 4, 1993, doi:10.1152/JN.1993.70.4.1629.

[2] J.A. Anderson, E. Rosenfeld, Neurocomputing: Foundations of Research, MIT Press, 1988.

[3] S. Araki, F. Nesta, E. Vincent, Z. Koldovský, G. Nolte, A. Ziehe, A. Benichoux, The 2011 Signal Separation Evaluation Campaign (SiSEC2011): - Audio source separation, 7191 LNCS, 414–422, 2012, doi:10.1007/978-3-642-28551-6_51.

[4] K. Avramidis, A. Kratimenos, C. Garoufis, A. Zlatintsi, P. Maragos, Deep Convolutional and Recurrent Networks for Polyphonic Instrument Classification from Monophonic Raw Audio Waveforms, Institute of Electrical and Electronics Engineers Inc., 2021-June, 3010–3014, 2021, doi:10.1109/ICASSP39728.2021.9413479.

[5] A. Bachem, Tone height and tone chroma as two different pitch qualities, North-Holland, 7, C, 80–88, C, 1950, doi:10.1016/0001-6918(50)90004-7.

[6] W. Bair, C. Koch, Temporal precision of spike trains in extrastriate cortex of the behaving macaque monkey, Neural Comput, 8, 6, 1185–1202, 6, 1996, doi:10.1162/NECO.1996.8.6.1185.

[7] A. Barbancho, I. Barbancho, L. Tardon, E. Molina, Description of the UMA Piano Chord Database, 17–34, 2013, doi:10.1007/978-1-4614-7476-0_3.

[8] D. Barry, R. Lawlor, B. Lawlor, E. Coyle, Sound Source Separation: Azimuth Discrimination and Sound Source Separation: Azimuth Discrimination and Resynthesis, 2004.

[9] M.A. Bartsch, G.H. Wakefield, Audio Thumbnailing of Popular Music Using Chroma-Based Representations, 7, 1, 96–104, 1, 2005, doi:10.1109/TMM.2004.840597.

[10] H.C. Bhakta, V.K. Choday, W.H. Grover, Musical Instruments As Sensors, ACS Omega, 3, 9, 11026–11032, 9, 2018, doi:10.1021/ACSOMEGA.8B01673.

[11] S.S. Bhojane, O.G. Labhshetwar, K. Anand, S.R. Gulhane, Musical Instrument Recognition Using Machine Learning Technique. International Research Journal of Engineering and Technology, 2017.

[12] R.M. Bittner, J. Wilkins, H. Yip, J.P. Bello, Medleydb 2.0 : New Data and a System for Sustainable Data Collection.

[13] R. Bittner, J. Salamon, S. Essid, J. Bello, Melody Extraction by Contour Classification, 2015.

[14] R. Bittner, J. Salamon, M. Tierney, M. Mauch, C. Cannam, J. Bello, MedleyDB: A Multitrack Dataset for Annotation-Intensive MIR Research, 2014.

[15] M. Blaszke, D. Koszewski, S. Zaporowski, Real and Virtual Instruments in Machine Learning - Training and Comparison of Classification Results, 2019.

[16] J.J. Bosch, J. Janer, F. Fuhrmann, P. Herrera, A Comparasion of Sound Segregation Techniques for Predominant Instrument Recognition in Musical Audio Signals, 2012.

[17] G.J. Brown, M. Cooke, Computational auditory scene analysis, Academic Press, 8, 4, 297–336, 4, 1994, doi:10.1006/CSLA.1994.1016.

[18] S. Brunak, B. Lautrup, Neural Networks — Computers with Intuition, WORLD SCIENTIFIC, 1990, doi:10.1142/0878.

[19] J.A. Burgoyne, I. Fujinaga, J.S. Downie, Music Information Retrieval, John Wiley & Sons, Ltd, 213–228, 2015, doi:10.1002/9781118680605.ch15.

[20] J. Burred, Hierarchical approach to automatic musical genre classification, 52, 7/8, 724–739, 7/8, 2014.

[21] J.F. Cardoso, Blind signal separation: Statistical principles, 86, 10, 2009–2025, 10, 1998, doi:10.1109/5.720250.

[22] Ò. Celma, Foafing the Music: Bridging the Semantic Gap in Music Recommendation, Springer, Berlin, Heidelberg, 4273 LNCS, 927–934, 2006, doi:10.1007/11926078_67.

[23] O. Celma, P. Herrera, X. Serra, A Multimodal Approach to Bridge the Music Semantic Gap., 2006.

[24] H.C. Chen, A.L.P. Chen, A music recommendation system based on music and user grouping, Springer, 24, 2–3, 113–132, 2–3, 2005, doi:10.1007/S10844-005-0319-3/METRICS.

[25] Y. Choe, Hebbian Learning, Springer, New York, NY, 1–5, 2014, doi:10.1007/978-1-4614-7320-6_672-1.

[26] K. Choi, G. Fazekas, M. Sandler, K. Cho, Convolutional recurrent neural networks for music classification, IEEE, 2392–2396, 2017, doi:10.1109/ICASSP.2017.7952585.

[27] O. Das, Musical Instrument Identification with Supervised Learning, 2019.

[28] L. Deng, J. Chen, Sequence classification using the high-level features extracted from deep neural networks, Institute of Electrical and Electronics Engineers Inc., 6844–6848, 2014, doi:10.1109/ICASSP.2014.6854926.

[29] C. Dewi, R.C. Chen, Combination of Resnet and Spatial Pyramid Pooling for Musical Instrument Identification, Sciendo, 22, 1, 104–116, 1, 2022, doi:10.2478/CAIT-2022-0007.

[30] A. Eronen, Musical instrument recognition using ICA-based transform of features and discriminatively trained HMMs, 133–136, 2003.

[31] A. Eronen, A. Klapuri, Musical Instrument Recognition Using Cepstral Coefficients and Temporal Features, 753–756, 2000.

[32] S. Essid, G. Richard, B. David, Musical Instrument Recognition by pairwise classification strategies.

[33] J.A. Feldman, M.A. Fanty, N.H. Goddard, Computing with Structured Neural Networks, 21, 3, 91–103, 3, 1988, doi:10.1109/2.34.

[34] W. Gerstner, Time structure of the activity in neural network models, American Physical Society, 51, 1, 738–758, 1, 1995, doi:10.1103/PhysRevE.51.738.

[35] S. Ghosh-Dastidar, H. Adeli, Third generation neural networks: Spiking neural networks, Springer Verlag, 61 AISC, 167–178, 2009, doi:10.1007/978-3-642-03156-4_17/COVER.

[36] D. Giannoulis, E. Benetos, A. Klapuri, M. Plumbley, Improving instrument recognition in polyphonic music through system integration, 2014, doi:10.1109/ICASSP.2014.6854599.

[37] I. Goodfellow, Y. Bengio, A. Courville, J. Heaton, Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning, Springer, 19, 1, 305–307, 1, 2017, doi:10.1007/S10710-017-9314-Z.

[38] S. Gururani, C. Summers, A. Lerch, Instrument Activity Detection in Polyphonic Music using Deep Neural Networks., 569–576, 2018.

[39] Y. Han, J. Kim, K. Lee, Deep Convolutional Neural Networks for Predominant Instrument Recognition in Polyphonic Music, Institute of Electrical and Electronics Engineers Inc., 25, 1, 208–221, 1, 2017, doi:10.1109/TASLP.2016.2632307.

[40] C. Harte, M. Sandler, M. Gasser, Detecting Harmonic Change In Musical Audio.

[41] T. Heittola, A. Klapuri, T. Virtanen, Musical Instrument Recognition in Polyphonic Audio Using Source-Filter Model for Sound Separation., 327–332, 2009.

[42] T. Heittola, T. Virtanen, A. Klapuri, Musical Instrument Recognition in Polyphonic Audio Using Source-Filter Model for Sound Separation. MUSICAL INSTRUMENT RECOGNITION IN POLYPHONIC AUDIO USING SOURCE-FILTER MODEL FOR SOUND SEPARATION, 2009.

[43] G.E. Hinton, S. Osindero, Y.W. Teh, A fast learning algorithm for deep belief nets, 18, 7, 1527–1554, 7, 2006, doi:10.1162/NECO.2006.18.7.1527.

[44] J.J. Hopfield, Neural networks and physical systems with emergent collective computational abilities., Proceedings of the National Academy of Sciences, 79, 8, 2554–2558, 8, 1982, doi:10.1073/PNAS.79.8.2554.

[45] J. Hunter, An overview of the MPEG-7 description definition language (DDL), 11, 6, 765–772, 6, 2001, doi:10.1109/76.927438.

[46] D.N. Jiang, L. Lu, H.J. Zhang, J.H. Tao, L.H. Cai, Music type classification by spectral contrast feature, Institute of Electrical and Electronics Engineers Inc., 1, 113–116, 2002, doi:10.1109/ICME.2002.1035731.

[47] K.G. Kim, Book Review: Deep Learning, Korean Society of Medical Informatics, 22, 4, 351–354, 4, 2016, doi:10.4258/HIR.2016.22.4.351.

[48] T. Kitahara, M. Goto, H. Okuno, Musical Instrument Identification Based on F0Dependent Multivariate Normal Distribution, 5, 421–424, 2003.

[49] A. Klapuri, M. Davy, Signal processing methods for music transcription, Springer US, 1–440, 2006, doi:10.1007/0-387-32845-9/COVER.

[50] R. Koenen, F. Pereira, MPEG-7: A standardized description of audiovisual content, Elsevier Science B.V., 16, 1, 5–13, 1, 2000, doi:10.1016/S0923-5965(00)00014-X.

[51] G. Korvel, P. Treigys, G. Tamulevičius, J. Bernatavičienė, B. Kostek, Analysis of 2D Feature Spaces for Deep Learning-Based Speech Recognition, Audio Engineering Society, 66, 12, 1072–1081, 12, 2018, doi:10.17743/JAES.2018.0066.

[52] B. Kostek, A. Czyżewski, Representing Musical Instrument Sounds for Their Automatic Classification, 49, 9, 768–785, 9, 2001.

[53] B. Kostek, Musical Instrument Classification and Duet Analysis Employing Music Information Retrieval Techniques, Institute of Electrical and Electronics Engineers Inc., 92, 4, 712–729, 4, 2004, doi:10.1109/JPROC.2004.825903.

[54] D. Koszewski, T. Görne, G. Korvel, B. Kostek, Automatic music signal mixing system based on one-dimensional Wave-U-Net autoencoders, Springer Science and Business Media Deutschland GmbH, 2023, 1, 1–17, 1, 2023, doi:10.1186/S13636-022-00266-3/TABLES/6.

[55] A. Kratimenos, K. Avramidis, C. Garoufis, A. Zlatintsi, P. Maragos, Augmentation methods on monophonic audio for instrument classification in polyphonic music, European Signal Processing Conference, EUSIPCO, 2021-January, 156–160, 2021, doi:10.23919/EUSIPCO47968.2020.9287745.

[56] E.W. Large, J.C. Kim, N.K. Flaig, J.J. Bharucha, C.L. Krumhansl, A Neurodynamic Account of Musical Tonality, University of California Press, 33, 3, 319–331, 3, 2016, doi:10.1525/MP.2016.33.3.319.

[57] O. Lartillot, P. Toiviainen, T. Eerola, A Matlab Toolbox for Music Information Retrieval, 261–268, 2008, doi:10.1007/978-3-540-78246-9_31.

[58] Q. V. Le, Building high-level features using large scale unsupervised learning, 8595–8598, 2013, doi:10.1109/ICASSP.2013.6639343.

[59] G. Lee KAIST, Y.-W. Tai, J. Kim KAIST, Deep Saliency With Encoded Low Level Distance Map and High Level Features, 660–668, 2016.

[60] P. Li, J. Qian, T. Wang, Automatic Instrument Recognition in Polyphonic Music Using Convolutional Neural Networks, 2015.

[61] F. Lluís, J. Pons, X. Serra, End-to-end music source separation: is it possible in the waveform domain?

[62] W. Maass, Networks of Spiking Neurons: The Third Generation of Neural Network Models, 10, 9, 1659–1671, 9, 1997.

[63] Maciej Madej, Katalog Dźwięków Instrumentów Muzycznych, Politechnika Gdańska, 1998.

[64] B. De Man, J.D. Reiss, R. Stables, Ten Years Of Automatic Mixing, 2017.

[65] P. Mandal, I. Nath, N. Gupta, M.K. Jha, D.G. Ganguly, S. Pal, Automatic music genre detection using artificial neural networks, Springer, 1125, 17–24, 2020, doi:10.1007/978-981-15-2780-7_3/COVER.

[66] E. Manilow, G. Wichern, P. Seetharaman, J. Le Roux, Cutting Music Source Separation Some Slakh: A Dataset to Study the Impact of Training Data Quality and Quantity, Institute of Electrical and Electronics Engineers Inc., 2019-October, 45–49, 2019, doi:10.48550/arxiv.1909.08494.

[67] J. Marques, P.J. Moreno, A Study of Musical Instrument Classification Using Gaussian Mixture Models and Support Vector Machines, 1999.

[68] M.A. Martínez-Ramírez, W.-H. Liao, G. Fabbro, S. Uhlich, \ Chihiro Nagashima, Y. Mitsufuji, Automatic music mixing with deep learning and out-of-domain data, 2022.

[69] W.S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity, Kluwer Academic Publishers, 5, 4, 115–133, 4, 1943, doi:10.1007/BF02478259.

[70] B. McFee, E.J. Humphrey, J. Bello, A Software Framework for Musical Data Augmentation, 2015.

[71] B. Mcfee, C. Raffel, D. Liang, D.P.W. Ellis, M. Mcvicar, E. Battenberg, O. Nieto, librosa: Audio and Music Signal Analysis in Python, 2015.

[72] P. Mermelstein, Distance measures for speech recognition, psychological and instrumental, 1976.

[73] M. Minsky, S.A. Papert, Perceptrons: An Introduction to Computational Geometry, The MIT Press, 2017, doi:10.7551/MITPRESS/11301.001.0001.

[74] M. Minsky, S.A. Papert, Perceptrons: An Introduction to Computational Geometry, The MIT Press, 2017, doi:10.7551/MITPRESS/11301.001.0001.

[75] A.F. Norcio, J. Stanley, Adaptive Human-Computer Interfaces: A Literature Survey and Perspective, 19, 2, 399–408, 2, 1989, doi:10.1109/21.31042.

[76] S. Oramas, F. Barbieri, O. Nieto, X. Serra, Multimodal Deep Learning for Music Genre Classification, 2018, doi:10.5334/tismir.10.

[77] S.J. Orfanidis, Introduction to signal processing, Prentice Hall, 1996.

[78] A. Otczyk, Przegląd informacji o bazach instrumentów muzycznych i rejestracja wybranych instrumentów, 2020.

[79] R.D. Patterson, Spiral Detection of Periodicity and the Spiral Form of Musical Scales, SAGE PublicationsSage CA: Thousand Oaks, CA, 14, 1, 44–61, 1, 2016, doi:10.1177/0305735686141004.

[80] D.I. Perrett, E.T. Rolls, W. Caan, Visual neurones responsive to faces in the monkey temporal cortex, Exp Brain Res, 47, 3, 329–342, 3, 1982, doi:10.1007/BF00239352.

[81] J. Pons, O. Slizovskaia, R. Gong, E. Gómez, X. Serra, Timbre analysis of music audio signals with convolutional neural networks, Institute of Electrical and Electronics Engineers Inc., 2017-January, 2744–2748, 2017, doi:10.23919/EUSIPCO.2017.8081710.

[82] K. Ratajski, Repozytorium dźwięków i brzmień instrumentów muzycznych - rejestracja wybranych instrumentów, 2021.

[83] Z. Raś, A. Wieczorkowska, Advances in Music Information Retrieval, 2010, doi:10.1007/978-3-642-11674-2.

[84] J. Robinson, R.S. Hatten, Emotions in Music, Oxford Academic, 34, 2, 71–106, 2, 2012, doi:10.1525/MTS.2012.34.2.71.

[85] E.T. Rolls, M.J. Tovee, Processing Speed in the Cerebral Cortex and the Neurophysiology of Visual Masking, Royal Society, 257, 1348, 9–15, 1348, 1994.

[86] O. Romani Picas, H. Parra Rodríguez, D. Dabiri, H. Tokuda, W. Hariya, K. Oishi, X. Serra, A real-time system for measuring sound goodness in instrumental sounds, Audio Engineering Society, 2015.

[87] F. Rosenblatt, The perceptron: A probabilistic model for information storage and organization in the brain, 65, 6, 386–408, 6, 1958, doi:10.1037/H0042519.

[88] F. Rosenblatt, Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms, Springer, Berlin, Heidelberg, 245–248, 1986, doi:10.1007/978-3-642-70911-1_20.

[89] D.F. Rosenthal, H.G. Okuno, Computational Auditory Scene Analysis, CRC Press, 2019.

[90] A. Rosner, B. Kostek, Automatic music genre classification based on musical instrument track separation, Springer New York LLC, 50, 2, 363–384, 2, 2018, doi:10.1007/s10844-017-0464-5.

[91] A. Rosner, B. Schuller, B. Kostek, Classification of music genres based on music separation into harmonic and drum components, Institute of Fundamental Technological Research, 39, 4, 629–638, 4, 2014, doi:10.2478/aoa-2014-0068.

[92] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back-propagating errors, Nature Publishing Group, 323, 6088, 533–536, 6088, 1986, doi:10.1038/323533a0.

[93] D.E Rumelhart, J. Mcclelland, J. L., Parallel distributed processing: explorations in the microstructure of cognition. Volume 1. Foundations, 1986.

[94] M. Schedl, Deep Learning in Music Recommendation Systems, Frontiers Media S.A., 5, 457883, 2019, doi:10.3389/FAMS.2019.00044/BIBTEX.

[95] R.N. Shepard, Circularity in Judgments of Relative Pitch, Acoustical Society of AmericaASA, 36, 12, 2346, 12, 2005, doi:10.1121/1.1919362.

[96] P.K. Shreevathsa, M. Harshith, M.A. Rao, Ashwini, Music instrument recognition using machine learning algorithms, Institute of Electrical and Electronics Engineers Inc., 161–166, 2020, doi:10.1109/ICCAKM46823.2020.9051514.

[97] A.J.R. Simpson, G. Roma, M.D. Plumbley, Deep Karaoke: Extracting Vocals from Musical Mixtures Using a Convolutional Deep Neural Network, Springer Verlag, 9237, 429–436, 2015, doi:10.48550/arxiv.1504.04658.

[98] Y. Song, S. Dixon, M. Pearce, A Survey of Music Recommendation Systems and Future Perspectives.

[99] S.S. Stevens, J. Volkmann, E.B. Newman, A Scale for the Measurement of the Psychological Magnitude Pitch, 8, 3, 185–190, 3, 1937, doi:10.1121/1.1915893.

[100] S.S. Stevens, J. Volkmann, E.B. Newman, A Scale for the Measurement of the Psychological Magnitude Pitch, Acoustical Society of AmericaASA, 8, 3, 185, 3, 2005, doi:10.1121/1.1915893.

[101] D. Szeliga, P. Tarasiuk, B. Stasiak, P.S. Szczepaniak, Musical Instrument Recognition with a Convolutional Neural Network and Staged Training, Elsevier, 207, 2493–2502, 2022, doi:10.1016/J.PROCS.2022.09.307.

[102] A. Świetlicka, Politechnika Poznańska. Wydawnictwo., Stochastyczny model biologicznej sieci neuronowej oparty na kinetycznych schematach Markowa, Wydawnictwo Politechniki Poznańskiej, 2015.

[103] R. Tadeusiewicz, Sieci Neuronowe, 1994.

[104] A. Tanaka, Sensor-Based Musical Instruments and Interactive Music, Oxford University Press, 2011, doi:10.1093/OXFORDHB/9780199792030.013.0012.

[105] C.C. Tappert, Who Is the Father of Deep Learning?, 343–348, 2019, doi:10.1109/CSCI49370.2019.00067.

[106] S. Thorpe, M. Imbert, Biological Constraints On Connectionist Modelling, 1989.

[107] H.C. (Henry C. Tuckwell, Introduction to theoretical neurobiology, Cambridge University Press, 1988.

[108] L. Turchet, A. McPherson, M. Barthet, Real-time hit classification in a smart cajón, Frontiers Media S.A., 5, 382729, 2018, doi:10.3389/FICT.2018.00016/BIBTEX.

[109] L. Turchet, A. McPherson, C. Fischione, Smart instruments : Towards an ecosystem of interoperable devices connecting performers and audiences, 498–505, 2019.

[110] G. Tzanetakis, P. Cook, Musical genre classification of audio signals, 10, 5, 293–302, 5, 2002, doi:10.1109/TSA.2002.800560.

[111] Leslie. Valiant, Circuits of the mind, Oxford University Press, 237, 1994.

[112] G.H. Wakefield, Wakefield, G. H., Mathematical representation of joint time-chroma distributions, SPIE, 3807, 637–645, 1999, doi:10.1117/12.367679.

[113] S. Van Der Walt, S.C. Colbert, G. Varoquaux, The NumPy array: A structure for efficient numerical computation, 13, 2, 22–30, 2, 2011, doi:10.1109/MCSE.2011.37.

[114] P. Werbos, Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Science. Thesis (Ph. D.). Appl. Math. Harvard University, 1974.

[115] S. Young, G. Evermann, D. Kershaw, G. Moore, J. Odell, D. Ollason, D. Povey, V. Valtchev, P. Woodland, The HTK Book, 1995.

[116] F. Zhang, Research on Music Classification Technology Based on Deep Learning, Hindawi, 2021, 7182143, 2021, doi:10.1155/2021/7182143.

[117] L. Zhong, E. Cooper, J. Yamagishi, N. Minematsu, Exploring Isolated Musical Notes as Pre-training Data for Predominant Instrument Recognition in Polyphonic Music, APSIPA ASC, 2023.

[118] IRMAS: a dataset for instrument recognition in musical audio signals - MTG - Music Technology Group (UPF), (dostęp: 06.2023), https://www.upf.edu/web/mtg/irmas.

[119] Dysonans, (dostęp: 06.2023), https://opoka.org.pl/biblioteka/I/IM/dysonans.html.

[120] NIME 2023, (dostęp: 06.2023), https://www.nime2023.org/.

[121] What is „Post Production”?, (dostęp: 06.2023), https://www.sweetwater.com/insync/post-production/.

[122] What Is Music Post-Production? – Joey Sturgis Tones, (dostęp: 06.2023), https://joeysturgistones.com/blogs/learn/what-is-music-post-production.

[123] Music Information Retrieval - an overview | ScienceDirect Topics, (dostęp: 08.2022), https://www.sciencedirect.com/topics/computer-science/music-information-retrieval.

[124] Music Information Retrieval - MTG - Music Technology Group (UPF), (dostęp: 08.2022), https://www.upf.edu/web/mtg/music-information-retrieval.

[125] Slakh | Demo site for the Synthesized Lakh Dataset (Slakh), (dostęp: 01.2023), http://www.slakh.com/.

[126] McCulloch-Pitts Neuron — Mankind's First Mathematical Model Of A Biological Neuron | by Akshay L Chandra | Towards Data Science, (dostęp: 01.2023), https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1.

[127] Pulsujące sieci neuronowe (Spiking Neural Network) ŁUKASZ ALEKSIEJEW (pod kierunkiem Joanna Grabska-Chrząstowska).

[128] Neuromorphic Computing - Next Generation of AI, (dostęp: 09.2022), https://www.intel.pl/content/www/pl/pl/research/neuromorphic-computing.html.

[129] LibROSA — librosa 0.6.3 documentation, (dostęp: 06.2023), https://librosa.github.io/librosa/.

[130] sygnał foniczny - Encyklopedia PWN - źródło wiarygodnej i rzetelnej wiedzy, (dostęp: 11.2022), https://encyklopedia.pwn.pl/haslo/sygnal-foniczny;3981871.html.

[131] Madmom documentation — madmom 0.17.dev0 documentation, (dostęp: 08.2022), https://madmom.readthedocs.io/en/latest/.

[132] Notes on Music Information Retrieval, (dostęp: 08.2022), https://musicinformationretrieval.com/index.html.

[133] ISO - ISO/IEC 15938-15:2019 - Information technology — Multimedia content description interface — Part 15: Compact descriptors for video analysis, (dostęp: 01.2023), https://www.iso.org/standard/75399.html.

[134] MPEG-7 | MPEG, (dostęp: 01.2023), https://mpeg.chiariglione.org/standards/mpeg-7.

[135] ISO - ISO/IEC 15938-1:2002 - Information technology — Multimedia content description interface — Part 1: Systems, (dostęp: 07.2022), https://www.iso.org/standard/34228.html.

[136] MPEG 7 standard, https://mpeg.chiariglione.org/standards/mpeg-7.

[137] Mel, (dostęp: 06.2023), https://www.sfu.ca/sonic-studio-webdav/handbook/Mel.html.

[138] SciPy, (dostęp: 01.2023), https://scipy.org/.

[139] PEP 8 – Style Guide for Python Code | peps.python.org, (dostęp: 06.2023), https://peps.python.org/pep-0008/.

[140] Auditory Toolbox, (dostęp: 01.2023), https://engineering.purdue.edu/~malcolm/interval/1998-010/.

[141] Stochastic Gradient Descent — Clearly Explained !! | by Aishwarya V Srinivasan | Towards Data Science, (dostęp: 01.2023), https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31.

[142] Deep-Neural-Network-What-is-Deep-Learning-Edureka.png (1926×987), (dostęp: 01.2023), https://cdn.edureka.co/blog/wp-content/uploads/2017/05/Deep-Neural-Network-What-is-Deep-Learning-Edureka.png.

[143] What Is Deep Learning? - How It Works | NetApp, (dostęp: 01.2023), https://www.netapp.com/artificial-intelligence/what-is-deep-learning/.

[144] GitHub - tensorflow/tensorflow: An Open Source Machine Learning Framework for Everyone, (dostęp: 01.2023), https://github.com/tensorflow/tensorflow.

[145] TensorFlow, (dostęp: 01.2023), https://www.tensorflow.org/.

[146] CPU vs GPU vs TPU: Understanding the difference b/w them, (dostęp: 01.2023), https://serverguy.com/comparison/cpu-vs-gpu-vs-tpu/.

[147] Google RankBrain: The Definitive Guide, (dostęp: 01.2023), https://backlinko.com/google-rankbrain-seo.

[148] TensorFlow Lite, (dostęp: 08.2023), https://www.tensorflow.org/lite/guide.

[149] Pytorch Vs Tensorflow Vs Keras: Here are the Difference You Should Know, (dostęp: 01.2023), https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article.

[150] Deep learning integration for Nengo — NengoDL 3.6.0.dev0 docs, (dostęp: 01.2023), https://www.nengo.ai/nengo-dl/introduction.html.

[151] KerasSpiking — KerasSpiking 0.3.0 docs, (dostęp: 01.2023), https://www.nengo.ai/keras-spiking/v0.3.0/index.html.

[152] KerasSpiking versus NengoDL — KerasSpiking 0.3.1.dev0 docs, (dostęp: 01.2023), https://www.nengo.ai/keras-spiking/nengo-dl-comparison.html.

[153] Concepts — MLflow 1.27.0 documentation, (dostęp: 07.2022), https://mlflow.org/docs/latest/concepts.html.

[154] Classification: Precision and Recall | Machine Learning Crash Course | Google Developers, (dostęp: 02.2023), https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall.

[155] Multilabel Ranking Metrics-Label Ranking Average Precision | ML - GeeksforGeeks, (dostęp: 07.2023), https://www.geeksforgeeks.org/multilabel-ranking-metrics-label-ranking-average-precision-ml/.

[156] Klasyfikacja: dokładność | Machine Learning | Google for Developers, (dostęp: 07.2023), https://developers.google.com/machine-learning/crash-course/classification/accuracy?hl=pl.

[157] Standard MIDI file format, updated, (dostęp: 06.2023), https://www.music.mcgill.ca/~ich/classes/mumt306/StandardMIDIfileformat.html.

[158] GOOD-SOUNDS: Good-sounds.org dataset - MTG - Music Technology Group (UPF), (dostęp: 01.2023), https://www.upf.edu/web/mtg/good-sounds.

[159] Home | Philharmonia, (dostęp: 01.2023), https://philharmonia.co.uk/.

[160] Sound samples | Philharmonia, (dostęp: 01.2023), https://philharmonia.co.uk/resources/sound-samples/.

[161] The Lakh MIDI Dataset v0.1, (dostęp: 01.2023), https://colinraffel.com/projects/lmd/.

[162] Happy Drum (Tank Drum), Price: Buy Steel Tongue Drum Kit with magical deep sound in the store of hand musical instruments, (dostęp: 07.2022), https://ravvast.com/.

[163] santūr | musical instrument | Britannica, (dostęp: 07.2022), https://www.britannica.com/art/santur.

[164] santoor01.jpg (600×400), (dostęp: 07.2022), https://www.rhythmitica.com/Application/public/filemanager/userfiles/user2/persian-instruments/santoor01.jpg.

[165] harp | Description, History, & Facts | Britannica, (dostęp: 07.2022), https://www.britannica.com/art/harp-musical-instrument.

[166] Brilliant!™ Harpsicle® Harp — Harpsicle® Harps, (dostęp: 07.2022), https://harpsicleharps.com/harps/brilliant-harpsicle-harp.

[167] Slakh | Demo site for the Synthesized Lakh Dataset (Slakh), (dostęp: 01.2023), http://www.slakh.com/.

[168] numpy.savez — NumPy v1.22 Manual, (dostęp: 01.2023), https://numpy.org/doc/stable/reference/generated/numpy.savez.html.

[169] The Functional API, (dostęp: 01.2023), https://keras.io/guides/functional_api/.

[170] tf.signal.fft | TensorFlow Core v2.7.0, (dostęp: 01.2023), https://www.tensorflow.org/api_docs/python/tf/signal/fft.

[171] tf.keras.layers.Conv2D | TensorFlow Core v2.7.0, (dostęp: 01.2023), https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D.

[172] Module: tf.keras.layers | TensorFlow Core v2.3.1, (dostęp: 12.2022), https://www.tensorflow.org/api_docs/python/tf/keras/layers.

[173] tf.keras.activations.relu | TensorFlow Core v2.3.1, (dostęp: 12.2022), https://www.tensorflow.org/api_docs/python/tf/keras/activations/relu.

[174] tf.keras.layers.BatchNormalization | TensorFlow Core v2.7.0, (dostęp: 01.2023), https://www.tensorflow.org/api_docs/python/tf/keras/layers/BatchNormalization.

[175] tf.keras.layers.MaxPool2D | TensorFlow Core v2.7.0, (dostęp: 01.2023), https://www.tensorflow.org/api_docs/python/tf/keras/layers/MaxPool2D.

[176] tf.keras.layers.Dense | TensorFlow Core v2.7.0, (dostęp: 01.2023), https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense.

[177] tf.keras.losses.MeanSquaredError | TensorFlow Core v2.7.0, (dostęp: 01.2023), https://www.tensorflow.org/api_docs/python/tf/keras/losses/MeanSquaredError.

[178] Classification: ROC Curve and AUC | Machine Learning Crash Course | Google Developers, (dostęp: 01.2023), https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc.

[179] tf.keras.optimizers.experimental.Adadelta | TensorFlow v2.12.0, (dostęp: 04.2023), https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/experimental/Adadelta.

[180] tf.keras.losses.BinaryCrossentropy | TensorFlow v2.12.0, (dostęp: 04.2023), https://www.tensorflow.org/api_docs/python/tf/keras/losses/BinaryCrossentropy.

[181] tf.keras.optimizers.Adam | TensorFlow v2.12.0, (dostęp: 05.2023), https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam.

[182] Top 10 Deep Learning Algorithms in Machine Learning [2023]; https://www.projectpro.io/article/deep-learning-algorithms/443] (dostęp: 06.2023).

12. DODATEK A*: SKRYPTY PRZYGOTOWUJĄCE RYSUNKI Z ROZDZIAŁU 2

Matlab:

```
%% Setup environment
close all
clear all
clc

%% Prepare signal
[signal, fs] = audioread("K:/Doktorat/Dane do treningu/good-
sounds/sound_files/cello_margarita_reference/akg/0010.wav");
signal_time = 0:1/fs:((length(signal)-1)/fs);

%% Rysunek 1 - Przykład przebiegu czasowego w różnej skali
time_signal = figure (1);
subplot (3,1,1);
plot(0:9999,signal(1:10000));
ylabel('amplituda(n)');
title('(a) Przykład sygnału o długości 10000 próbek');
subplot (3,1,2);
plot(0:999,signal(1:1000));
ylabel('amplituda(n)');
title('(b) Przykład sygnału o długości 1000 próbek');
subplot (3,1,3);
stem(0:99,signal(1:100),'.');
ylabel('amplituda(n)');
title('(c) Przykład sygnału o długości 100 próbek');
xlabel('n');

saveFigure(time_signal, '../Obrazy', 'przebieg_czasowy_sygnalu')

%% Rysunek 2 - Przykład działania FFT na sztuczyn przykładzie

fft_example = figure(2);
x1 = [-1 -0.5 1 2 1 0.25 0.5 -1];
x2 (16)=0;
x2 (1:8)=x1;
subplot (221);
stem (0:1:7, x1); axis ([-0.5 7.5 -1.5 2.5]);
ylabel('x(n)');
title('(a) 8 próbek');
subplot (222);
stem (0:1:7, abs (fft (x1))); axis ([-0.5 7.5 -0.5 6]);
ylabel('|X(k)|');
title('(b) 8-punktowe FFT');
subplot (223);
stem (0:1:15, x2); axis ([-0.5 15.5 -1.5 2.5]);
xlabel('n');
ylabel('x (n)');
title('(c) 8 próbek z zerowaniem do 16');
subplot (224);
```

```
stem (0:1:15, abs (fft (x2))); axis ([-1 16 -0.5 6]);
xlabel('k');ylabel('|X(k)|');
title('(d) 16-punktowe FFT');

saveFigure(fft_example, '../Obrazy', 'przykład_działania_FFT')

%% Rysunek 3 – przykład działania FFT na rzeczywistym przykładzie sygnału
N=8000;
x=cos(2*pi*1000* (0:1:N-1)/48000)';
fft_and_window_example = figure (3);
W=blackman(N);
W=N*W/sum(W); % scaling of window
f=((0:N/2-1)/N) *fs;

xw=x.*W;
subplot (3,2,1);
plot(0:N-1,x);
axis([0 1000 -1.1 1.1]);
title('(a) Sygnał cosinus x(n)')
subplot (3,2,3);
plot(0:N-1, xw);
axis([0 8000 -4 4]);
title('(c) Sygnał cosinus x_w (n) =x (n) \cdot w(n)')
subtitle('z nałożonym oknem')
X=20*log10(abs (fft(x, N))/(N/2));
subplot (3,2,2);
plot(f, X(1:N/2));
axis([0 10000 -80 10]);
ylabel('X(f)');
title('(b) Widmo sygnału cosinusa')
Xw=20*log10(abs (fft (xw, N))/(N/2));
subplot (3,2,4);
plot (f, Xw (1:N/2));
axis([0 10000 -80 10]);
ylabel('X(f)');
title('(d) Widmo sygnału cosinusa')
subtitle('z nałożonym oknem Blackmana')
s=0.6*(signal(1:N).*W)/max(abs(signal(1:N)));
subplot (3,2,5);
plot (0:N-1, s);
axis ([0 8000 -1.1 1.1]);
title('(e) Sygnał audio x_w(n)')
subtitle('z nałożonym oknem Blackmana')
xlabel('n');
Sw=20*log10(abs(fft(s, N))/(N/2));
subplot (3,2,6);
plot(f, Sw(1:N/2));
ylabel('X(f)');
title('(f) Widmo sygnału audio')
subtitle('z nałożonym oknem Blackmana')
axis ([0 10000 -80 10]);
xlabel('częstotliwość[Hz]');

saveFigure(fft_and_window_example, '../Obrazy',
'przykład_działania_FFT_z_oknem')
%% Rysunek 4 – Przykład działania ADC
N = 20;
x = sin(2*pi*1000*(0:1:N-1)/48000)+ ...
        cos(4*pi*1000*(0:1:N-1)/48000)+ ...
```

```
        cos(8*pi*1000*(0:1:N-1)/48000)+ ...
        cos(16*pi*1000*(0:1:N-1)/48000);

adc_figure = figure (4);
subplot (1,2,1);
plot(interp1(0:N-1,x,0:N/1000:N-1, 'spline'))
title('(a) Analogowy sygnał foniczny')
ylabel('Amplituda x(t)');
xlabel('czas t[s]');
subplot (1,2,2);
stem(0:N-1 ,x);
title('(b) Spróbkowany sygnał cyfrowy')
ylabel('Amplituda x(n)');
xlabel('próbki n');

saveFigure(adc_figure, '../Obrazy', 'przykład_działania_ADC')

%% Rysunek 5 – Przykład działania kwantyzatora
N = 20;
x = sin(2*pi*1000*(0:1:N-1)/48000)+ ...
        cos(4*pi*1000*(0:1:N-1)/48000)+ ...
        cos(8*pi*1000*(0:1:N-1)/48000)+ ...
        cos(16*pi*1000*(0:1:N-1)/48000);

quantizer_figure = figure (5);
t = tiledlayout(1,1);
ax1 = axes(t);
plot(ax1,0:N/1000:N-1,interp1(0:N-1,x,0:N/1000:N-1,"maksima"))
ylabel('Amplituda x(t)');
xlabel('czas t[s]');
ax2 = axes(t);
hold on
stem(ax2,0:N-1 ,(x)*10000, 'r');
stairs(ax2,0:N-1 ,(x)*10000, 'r')
ax2.XAxisLocation = 'top';
ax2.YAxisLocation = 'right';
ylabel('Amplituda x(n)');
xlabel('próbki n');
ax2.Color = 'none';
ax1.Box = 'off';
ax2.Box = 'off';
yticklabels(-10000:5000:30000)

saveFigure(quantizer_figure, '../Obrazy', 'przykład_działania_kwantyzatora')

%% Rysunek 6 – Przykład działania FFT na sygnale sinusoidalnym

fft_sin_example = figure(6);
fs = 48000;
N = 20;
x = sin(0.25*pi*(0:1:N-1));
subplot (311);
stem(0:N-1 ,x);
ylabel('x(n)');
xlabel('n');
title('(a) cyfrowy przebieg sygnału');
subplot (312);
stem(0:N-1 ,abs(fft(x))/N);
ylabel('|X(n)|');
```

```
xlabel('n');
title('(b) Wartość bezwzględna widma powyższego sygnału');
subplot (313);
stem(0:fs/N:fs-fs/N, abs(fft(x))/N);
yticklabels(0:fs/N:fs-fs/N)
ylabel('|X(f)|');
xlabel('częstotliwość f[Hz]');
title('(c) Wartość bezwzględna widma powyższego sygnału');
subtitle('w funkcji częstotliwości')


saveFigure(fft_sin_example, '../Obrazy', 'przykład_działania_FFT_na_sinusie')
%% Rysunek 7 – Przykład spektrogramu

spectrogram_example = figure(7);
subplot (2,1,1);
plot(0:length(signal)-1,signal);
ylabel('amplituda(n)');
xlabel('próbki n');
title('(a) Przykład sygnału w postaci czasowej');
subplot (2,1,2);
spectrogram(signal', 512, 128, 512, fs/2, 'yaxis')
ylabel('Czas [s]');
xlabel('Częstotliwość f[kHz]');
title('(b) Przykład spektrogramu sygnału');

saveFigure(spectrogram_example, '../Obrazy',
'przykład_działania_spectrogramu')


%% Rysunek 8 Przykład wykresu wodospadowego
waterfall_example = figure(7);
subplot(211);
plot(signal);
ylabel('amplituda(n)');
xlabel('próbki n');
title('(a) Przykład sygnału w postaci czasowej');
subplot (212);
start = 256;
steps = 256;
N = 512;
clippingpoint = 20;
baseplane = -100;

windoo=blackman(N);
windoo=windoo*N/sum(windoo);
n=length(signal);
rest=n-start-N;
nos=round(rest/steps);
if nos>rest/steps
    nos=nos-1;
end

x=linspace (0, fs/1000, N+1);

z=x-x;
cup=z+clippingpoint;
cdown=z+baseplane;
```

```
signal=signal+0.0000001;
for i=1:1:nos
    spek1=20.*log10(abs(fft(windoo.*signal(1+start+i*steps:
start+N+i*steps))))./ (N)/0.5);

    spek=[-200; spek1(1:N)];

    spek=(spek>cup').*cup' + (spek<=cup').*spek;
    spek=(spek<cdown').*cdown' + (spek>=cdown').*spek;

    spek(1)=baseplane-10;
    spek(N/2)=baseplane-10;
    y=x-x+(i-1);
    if i==1
        p=plot3(x(1:N/2), y(1:N/2), spek(1:N/2), 'k');
        set (p, 'Linewidth', 0.1);

    end
        pp=patch(x(1:N/2), y(1:N/2), spek(1:N/2), 'w', 'Visible', 'on');
        set(pp, 'Linewidth', 0.1);
end
set (gca, 'DrawMode', 'fast');
axis ([-0.3 fs/2000+0.3 0 nos baseplane-10 0]);
set (gca, 'Ydir', 'reverse');
view (12,40);

zlabel('amplituda (n)');
ylabel('próbki n');
xlabel('czętotliowość [kHz]');
title('(b) Przykład wykresu wodospadowego');
saveFigure(waterfall_example, '../Obrazy',
'przykład_działania_wykresu_wodospadowego')


%% Rysunek 9 – Przykład działania FFT z filtrem antyaliasingowym

antyaliasing_filter_example = figure(6);
fs = 48000;
N = 200;
x = sin(0.25*pi*(0:1:N-1))+ ...
    sin(0.5*pi*(0:1:N-1));
x=signal;
N=length(signal);
spec = abs(fft(x))/N;
subplot (211);
plot(0:fs/N:fs-fs/N, vertcat(spec(1:length(spec)/2),
zeros(int32(length(spec)/2),1)));
yticklabels(0:fs/N:fs-fs/N)
ylabel('|X(f)|');
xlabel('częstotliwość f[Hz]');
title('(a) Widmo sygnału analogowego');
subplot (212);
plot(0:fs/N:fs-fs/N, spec);
yticklabels(0:fs/N:fs-fs/N)
ylabel('|X(f)|');
xlabel('częstotliwość f[Hz]');
title('(a) Widmo sygnału cyfrowego');
```

```
saveFigure(antyaliasung_filter_example, '../Obrazy',
'przykład_działania_filtra_antyaliasingowego')
```

Python:

Rysunek 2.10

```
import Librosa
import Librosa.display
import matplotlib.pyplot as plt
import numpy as np

y, sr = Librosa.load("I:\\Doktorat\\Dane do
treningu\\musdb18hq\\test\\Zeno - Signs\\drums.wav", offset=25.0,
duration=20.0)
y=y/max(abs(y))

# Rysunek 2.10
fig, ax = plt.subplots(nrows=4, sharex=True, sharey=False)
Librosa.display.specshow(Librosa.amplitude_to_db(np.abs(Librosa.stft(y)))
,
                         y_axis='hz', x_axis='time', ax=ax[0])
ax[0].set(title='Spektrogram')
ax[0].label_outer()
Librosa.display.specshow(Librosa.amplitude_to_db(Librosa.feature.melspect
rogram(y=y, sr=sr)),
                         y_axis='mel', x_axis='time', ax=ax[1])
ax[1].set(title='Mel spektrogram')
ax[1].label_outer()
Librosa.display.specshow(Librosa.feature.chroma_cqt(y=y, sr=sr),
                         y_axis='chroma', x_axis='time', ax=ax[2])
ax[2].set(title='Chromagram')
ax[2].label_outer()
Librosa.display.specshow(Librosa.feature.tonnetz(y=y, sr=sr),
                         y_axis='tonnetz', x_axis='time', ax=ax[3])
ax[3].set(title="Reprezentacja Tonnetz")
ax[3].label_outer()
plt.show()
```