



**AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE**  
**Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej**

## Praca dyplomowa

*Aplikacja mobilna do kształcania prawidłowej wymowy*

*Mobile application for developing correct pronunciation*

Autor: *Konrad Stalmach*  
Kierunek studiów: *Automatyka i Robotyka*  
Opiekun pracy: *dr inż. Elżbieta Pociask*

Kraków, 2024

*Serdecznie dziękuję mojej promotorce, dr inż. Elżbiecie Pociask, za nieocenione wsparcie, cierpliwość i cenne wskazówki, które były niezbędne podczas tworzenia tej pracy.*

# Spis treści

<b>Słownik terminów .....</b>	4
<b>1. Wprowadzenie.....</b>	5
1.1. Cel pracy .....	5
1.2. Zakres pracy.....	6
<b>2. Przegląd literatury .....</b>	7
2.1. Rola logopedii w terapii mowy.....	8
2.2. Przegląd istniejących aplikacji i technologii wspierających terapię mowy .....	9
2.3. Zalety i wady istniejących rozwiązań.....	11
2.4. Przegląd zastosowań uczenia maszynowego w analizie mowy .....	12
2.5. Podstawy działania sieci neuronowych w kontekście analizy mowy .....	13
2.5.1. Rodzaje sieci neuronowych używanych w analizie mowy .....	13
2.5.2. Kluczowe komponenty sieci neuronowych w analizie mowy .....	14
2.5.3. Proces trenowania sieci neuronowych w analizie mowy .....	14
<b>3. Środowiska pracy .....</b>	15
3.1. Android Studio i Kotlin .....	15
3.2. Visual Studio Code i Python.....	16
<b>4. Tworzenie sieci neuronowej .....</b>	17
4.1. Konsultacja z logopeda i wybór danych wejściowych .....	17
4.2. Proces pozyskiwania danych .....	18
4.3. Przygotowanie danych do obróbki przez sieć.....	20
4.4. Projektowanie architektury sieci neuronowej .....	21
4.4.1. Opis warstw modelu .....	23
4.5. Implementacja modelu.....	24
4.5.1. Kluczowe etapy implementacji .....	24
4.6. Integracja modelu z aplikacją .....	25
4.6.1. Proces integracji.....	25
<b>5. Tworzenie aplikacji "<i>Erkorektor</i>" .....</b>	26
5.1. Ekran główny .....	28

5.2.	Ekran nagrywania .....	29
5.3.	Ekran wyboru analizy .....	30
5.4.	Ekran instrukcji.....	31
5.5.	Ekran analizy słowa .....	33
5.6.	Ekran wyboru ćwiczeń głoski.....	35
5.7.	Ekran wyboru rodzaju ćwiczenia.....	36
5.8.	Ekran ćwiczeń.....	37
<b>6.</b>	<b>Testowanie i wyniki.....</b>	<b>39</b>
6.1.	Początkowy test dla 9 słów.....	39
6.2.	Test dla każdego słowa .....	41
6.2.1.	<i>Szafa</i> .....	42
6.2.2.	<i>Szopa</i> .....	43
6.2.3.	<i>Szelki</i> .....	44
6.2.4.	<i>Kasza</i> .....	45
6.2.5.	<i>Nosze</i> .....	46
6.2.6.	<i>Wieszak</i> .....	47
6.2.7.	<i>Kosz</i> .....	48
6.2.8.	<i>Afisz</i> .....	49
6.2.9.	<i>Gulasz</i> .....	50
6.3.	Końcowy test na aplikacji .....	51
6.3.1.	<i>Szafa</i> .....	52
6.3.2.	<i>Szopa</i> .....	53
6.3.3.	<i>Szelki</i> .....	54
6.3.4.	<i>Kasza</i> .....	55
6.3.5.	<i>Nosze</i> .....	56
6.3.6.	<i>Wieszak</i> .....	57
6.3.7.	<i>Kosz</i> .....	58
6.3.8.	<i>Afisz</i> .....	59
6.3.9.	<i>Gulasz</i> .....	60
<b>7.</b>	<b>Analiza wyników .....</b>	<b>61</b>
<b>8.</b>	<b>Przyszły rozwój aplikacji .....</b>	<b>63</b>
<b>9.</b>	<b>Podsumowanie .....</b>	<b>65</b>
<b>10.</b>	<b>Dodatek A .....</b>	<b>68</b>

# Słownik terminów

W poniższym słowniku znajdują się definicje kluczowych skrótów i terminów użytych w pracy.

**AAC** (*ang. Augmentative and Alternative Communication*) - Alternatywne i wspomagające metody komunikacji

**AI** (*ang. Artificial Intelligence*) - Sztuczna inteligencja

**API** (*ang. Application Programming Interface*) - Interfejs programowania aplikacji

**ASR** (*ang. Automatic Speech Recognition*) - Automatyczne rozpoznawanie mowy

**CNN** (*ang. Convolutional Neural Networks*) - Konwolucyjne sieci neuronowe

**GUI** (*ang. Graphical User Interface*) - Graficzny interfejs użytkownika

**MFCC** (*ang. Mel-Frequency Cepstral Coefficients*) - Parametry Mel-cepstralne

**ML** (*ang. Machine Learning*) - Uczenie maszynowe

**NN** (*ang. Neural Networks*) - Sieci neuronowe

**RNN** (*ang. Recurrent Neural Networks*) - Rekurencyjne sieci neuronowe

**SGD** (*ang. Stochastic Gradient Descent*) - Stochastyczny spadek gradientu

**STFT** (*ang. Short-Time Fourier Transform*) - Krótkoczasowa transformata Fouriera

# **1. Wprowadzenie**

Komunikacja werbalna jest jednym z kluczowych elementów ludzkiego życia, wpływającym na nasze relacje społeczne, zawodowe i osobiste. Prawidłowa wymowa stanowi fundamentalny aspekt tej komunikacji, a jej zaburzenia mogą powodować liczne trudności w życiu prywatnym i zawodowym. Zaburzenia wymowy mogą powodować stres i obniżenie samooceny, co podkreśla znaczenie skutecznej terapii logopedycznej.

Od najmłodszych lat zmagałem się z trudnościami w prawidłowej wymowie głoski "r". Jako dziecko uczęszczałem do logopedy, jednak brak pełnego zrozumienia problemu oraz motywacji do ćwiczeń sprawił, że poprawa była niewielka. Dopiero niedawno, z pełną świadomością i determinacją, postanowiłem poprawić swoją wymowę i ponownie skorzystałem z pomocy logopedycznej. Trafiłem do dr Eweliny Strawa-Kęsek, specjalistki w dziedzinie logopedii, która nie tylko pomogła mi poprawić moją wymowę, ale również wprowadziła mnie w zrozumienie istoty problemu. Jej wsparcie okazało się niezwykle cenne również przy pisaniu tej pracy magisterskiej.

## **1.1. Cel pracy**

Celem niniejszej pracy jest opracowanie aplikacji mobilnej, która wspiera proces kształcenia prawidłowej wymowy. Aplikacja ma na celu nie tylko ułatwienie pracy logopedom, ale również umożliwienie pacjentom samodzielne ćwiczenie wymowy w dogodnym dla nich czasie i miejscu. Głównym zadaniem aplikacji jest nagrywanie, analiza mowy oraz oferowanie ćwiczeń, które pomogą użytkownikom w poprawie wymowy.

Praca koncentruje się na głosce „sz”, ze względu na jej specyficzne właściwości, które czynią ją idealnym obiektem do analizy w kontekście terapii mowy. Konsultacja z logopedą wykazała, że „sz” to głoska, którą można przedłużać w trakcie wymowy, co pozwala na bardziej szczegółowe monitorowanie i analizę problemów artykulacyjnych. Wybór tej głoski miał charakter strategiczny: jeśli uda się opracować skuteczny model dla „sz”, aplikację będzie można w podobny sposób rozszerzyć o inne głoski, co znacząco zwiększy jej użyteczność w terapii logopedycznej. Dzięki temu aplikacja będzie mogła wspierać szerszy zakres ćwiczeń mowy, obejmujących różnorodne dźwięki i głoski.

## **1.2. Zakres pracy**

Zakres pracy obejmuje przegląd literatury dotyczącej istniejących badań i technologii wspierających terapię mowy, ze szczególnym naciskiem na sztuczną inteligencję i uczenie maszynowe w analizie mowy. Praca obejmuje również opracowanie aplikacji wykorzystującej te technologie do wspierania diagnozy i terapii mowy. W ramach projektu opisano proces projektowania, implementacji oraz testowania aplikacji, w tym zbieranie i przygotowanie danych, trenowanie modelu rozpoznawania mowy, integrację modelu z aplikacją oraz analizę wyników testów. Na koniec przedstawiono propozycje dalszego rozwoju aplikacji, z uwzględnieniem możliwych kierunków rozszerzenia funkcjonalności.

## 2. Przegląd literatury

W celu lepszego zrozumienia tematu aplikacji mobilnej wspierającej kształcenie prawidłowej wymowy, przeprowadzono szczegółową analizę różnych dziedzin nauki, kluczowych dla tego zagadnienia. W szczególności, uwzględniono następujące obszary:

1. Rola logopedii w terapii mowy
2. Przegląd istniejących aplikacji i technologii wspierających terapię mowy
3. Zalety i wady istniejących rozwiązań
4. Przegląd zastosowań uczenia maszynowego w analizie mowy
5. Podstawy działania sieci neuronowych w kontekście analizy mowy

Analiza tych dziedzin umożliwiła kompleksowe podejście do tematu, co było kluczowe dla stworzenia efektywnej i funkcjonalnej aplikacji mobilnej wspomagającej terapię mowy. Dzięki szerokiemu przeglądowi literatury możliwe było zidentyfikowanie najskuteczniejszych metod i technologii, które mogą zostać zaadaptowane i rozwinięte w kontekście aplikacji do kształcenia prawidłowej wymowy. [1-14]



Rys. 2.1. Obszary analizy

## 2.1. Rola logopedii w terapii mowy

Logopedia odgrywa kluczową rolę w diagnozowaniu i leczeniu zaburzeń mowy, obejmując szeroki zakres działań mających na celu poprawę zdolności komunikacyjnych pacjentów. Artykuł „Scope of Practice in Speech-Language Pathology” opublikowany przez American Speech-Language-Hearing Association (ASHA) przedstawia szeroki zakres praktyk logopedycznych, które obejmują diagnozę, terapię oraz wsparcie edukacyjne i technologiczne. [15]

### Diagnoza i Ocena

Logopedzi dokonują kompleksowej oceny funkcji mowy i języka, analizując zdolności artykulacyjne, fonologiczne, językowe oraz pragmatyczne pacjentów. Proces ten obejmuje wykorzystanie różnych narzędzi diagnostycznych, które pomagają zidentyfikować specyficzne zaburzenia mowy, takie jak dyslalia, jąkanie, apraksja mowy czy zaburzenia fonologiczne (w moim wypadku był to rotacyzm).

### Interwencja Terapeutyczna

Terapia logopedyczna obejmuje różnorodne metody i techniki, dostosowane do indywidualnych potrzeb pacjenta. Przykłady skutecznych interwencji to ćwiczenia oralne, techniki fonetyczne oraz strategie poprawy płynności mowy. Wczesna interwencja i regularne sesje terapeutyczne mogą znacząco poprawić funkcje mowy i języka u dzieci i dorosłych.

### Wsparcie dla Rodziny i Edukacja

Logopedzi często współpracują z rodzinami pacjentów, zapewniając ciągłe wsparcie i edukację na temat strategii wspomagających rozwój mowy w domu. Terapia może również obejmować szkolenie dla nauczycieli i innych specjalistów. Współpraca ta jest kluczowa dla skuteczności terapii, ponieważ środowisko domowe i szkolne odgrywa istotną rolę w procesie terapeutycznym, aby zapewnić spójne wsparcie w różnych środowiskach życia pacjenta.

### Zastosowanie Nowoczesnych Technologii

Współczesna logopedia korzysta z zaawansowanych technologii, takich jak aplikacje mobilne, systemy wspomagające komunikację (AAC) oraz narzędzia do analizy mowy oparte na sztucznej inteligencji i uczeniu maszynowym. Technologie te mogą wspierać proces terapeutyczny, umożliwiając bardziej precyzyjne i spersonalizowane podejście do leczenia.

## Skuteczność Kliniczna i Przestrzeganie Wytycznych

Przestrzeganie klinicznych wytycznych jest kluczowe dla zapewnienia skuteczności terapii logopedycznej. Wytyczne te, oparte na badaniach naukowych, pomagają logopedom w prowadzeniu interwencji terapeutycznych zgodnie z najlepszymi praktykami, co prowadzi do lepszych wyników leczenia. Przestrzeganie wytycznych oznacza, że logopedzi stosują uznane metody i procedury, co zwiększa prawdopodobieństwo pozytywnych rezultatów terapii.

Warto podkreślić, że nawet najlepszy logopeda nie jest w stanie osiągnąć zamierzonych efektów terapeutycznych bez regularnych ćwiczeń wykonywanych przez pacjenta. Regularna praktyka jest kluczowa dla utrwalenia poprawnych wzorców mowy i eliminacji błędów artykulacyjnych. Z własnego doświadczenia wiem, że brak zaangażowania w ćwiczenia domowe może znaczowo wpływać na efektywność terapii. W dzieciństwie uczęszczałem do logopedy, jednak brakowało mi chęci do regularnych ćwiczeń w domu, co ograniczyło postępy w poprawie mojej wymowy.

## 2.2. Przegląd istniejących aplikacji i technologii wspierających terapię mowy

Nowoczesna technologia odgrywa kluczową rolę we wspieraniu terapii mowy, oferując innowacyjne rozwiązania, które mogą znaczco zwiększyć jej efektywność. Poniżej przedstawiono przegląd kilku popularnych aplikacji dostępnych w *Google Play Store*, które wspierają terapię mowy.

### Otsimo | Speech Therapy SLP

*Otsimo | Speech Therapy SLP* oferuje interaktywne ćwiczenia i gry, które pomagają w poprawie artykulacji oraz rozwijaniu umiejętności językowych u dzieci z różnymi zaburzeniami mowy, takimi jak dyslalia czy opóźnienia rozwoju mowy. Aplikacja dostarcza bogaty zestaw angażujących zadań, które motywują dzieci do nauki poprzez zabawę.

Należy jednak zauważyć, że aplikacja nie oferuje bezpośredniej informacji zwrotnej dotyczącej poprawności wykonywanych ćwiczeń, co może ograniczać możliwość bieżącej korekty błędów przez użytkownika. Przykładowo, nawet gdy aplikacja wymagała powiedzenia słowa "mom" (co w języku angielskim oznacza "mama"), a wypowiedziano coś zupełnie innego, nadal pojawiła się informacja o świetnym wykonaniu zadania. Dodatkowo, wiele funkcji aplikacji jest dostępnych jedynie w wersji płatnej. [16]

### Stamurai: Stuttering Therapy

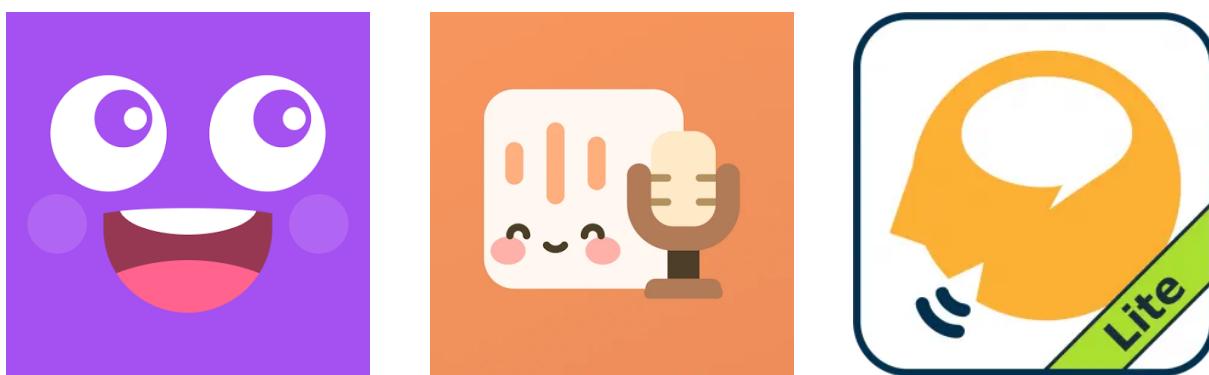
*Stamurai* to aplikacja dedykowana osobom z jąkaniem, oferująca codzienne ćwiczenia mowy i techniki poprawiające płynność wypowiedzi. Aplikacja dostarcza spersonalizowane ćwiczenia, które pomagają użytkownikom stopniowo rozwijać swoje umiejętności komunikacyjne. Dzięki codziennym sesjom praktycznym użytkownicy mogą śledzić swoje postępy za pomocą analiz głosu, regularnych raportów oraz samodzielnego ocen. Aplikacja rejestruje zmiany w płynności mowy, częstotliwości jąkania i innych wskaźnikach, dostosowując program terapeutyczny do indywidualnych potrzeb użytkownika.

Ograniczeniem tego rozwiązania jest fakt, że sporo funkcji aplikacji jest dostępnych tylko w płatnych wersjach, co może być barierą dla niektórych użytkowników. Dodatkowo, brak interakcji na żywo może ograniczać skuteczność terapii w porównaniu do tradycyjnej pracy z logopedą. Obecnie aplikacja nie jest również dostępna w języku polskim, co może stanowić utrudnienie dla osób nieznających języka angielskiego. [17]

### Apraxia Therapy Lite

*Apraxia Therapy Lite* wspiera osoby dorosłe z apraksją mowy, oferując serie filmów instruktażowych pokazujących ruchy ust, które pomagają w nauce poprawnej wymowy. Aplikacja umożliwia użytkownikom wielokrotne odtwarzanie nagrani, co pozwala na dokładne przeanalizowanie i naśładowanie ruchów artykulacyjnych.

Aplikacja nie oferuje analizy obrazu lub wideo pod kątem prawidłowego ułożenia ust, co oznacza, że użytkownicy muszą polegać wyłącznie na własnej ocenie. Dodatkowo, wiele funkcji jest tutaj zablokowanych i dostępnych jedynie w płatnej wersji *Apraxia Therapy Aphasia*. [18]



Rys. 2.2. Od lewej: logo aplikacji *Otsimo*, logo aplikacji *Stamurai*, logo aplikacji *Apraxia Therapy Lite*

## 2.3. Zalety i wady istniejących rozwiązań

W kontekście terapii mowy, istniejące aplikacje i technologie mają swoje unikalne zalety i wady. Poniżej przedstawiono przegląd głównych zalet i wad tych rozwiązań, bazując na analizie dostępnych źródeł oraz przykładach z rynku.

### Zalety

Aplikacje mobilne do terapii mowy oferują znaczną wygodę użytkowania, umożliwiając dostęp do ćwiczeń w dowolnym miejscu i czasie, co może być szczególnie korzystne dla osób z ograniczonym dostępem do specjalistów. Przykładem są aplikacje takie jak *Otsimo* i *Stamurai*, które pozwalają użytkownikom na samodzielne przeprowadzanie ćwiczeń logopedycznych w domowym zaciszu, eliminując potrzebę częstych wizyt u logopedy.

Dodatkowo, personalizacja terapii jest jednym z kluczowych atutów tych aplikacji. Możliwość dostosowania ćwiczeń do indywidualnych potrzeb użytkownika, jak w przypadku *Apraxia Therapy Lite*, przyczynia się do bardziej efektywnej i spersonalizowanej terapii, co zwiększa szansę na poprawę umiejętności językowych.

Wykorzystanie nowoczesnych technologii, takich jak rozpoznawanie mowy i sztuczna inteligencja, pozwala na dokładną analizę postępów użytkownika i dostarczanie cennych informacji zwrotnych. Aplikacje z funkcjami AI mogą śledzić rozwój mowy, dostosowując terapię do bieżących potrzeb użytkownika, co może prowadzić do szybszych i bardziej znaczących postępów.

### Wady

Pomimo licznych zalet, aplikacje mobilne nie zastąpią w pełni interakcji z profesjonalnym logopedą, która jest kluczowa dla skutecznej terapii mowy. Brak bezpośredniego nadzoru specjalisty może ograniczać efektywność terapii, szczególnie w przypadkach, gdzie wymagana jest precyzyjna korekta i natychmiastowe dostosowanie technik terapeutycznych.

Koszty związane z zaawansowanymi funkcjami aplikacji również stanowią istotną wadę. Wiele aplikacji oferuje swoje najważniejsze funkcje wyłącznie w wersjach płatnych, co może stanowić barierę finansową dla użytkowników. Wysokie koszty subskrypcji mogą ograniczyć dostęp do pełnej gamy narzędzi terapeutycznych, co może wpływać na skuteczność terapii.

Kolejnym ograniczeniem jest brak dostępności aplikacji w języku polskim. Większość dostępnych narzędzi logopedycznych jest opracowana z myślą o anglojęzycznych użytkownikach, co utrudnia ich użycie przez osoby posługujące się innymi językami. Brak polskojęzycznych aplikacji logopedycznych oznacza, że osoby te mają ograniczony dostęp do spersonalizowanych narzędzi, co może wpływać na efektywność terapii.

## Innowacyjność proponowanego rozwiązania

Przegląd literatury oraz komercyjnych rozwiązań wykazuje brak aplikacji dla osób posługujących się językiem polskim. Język polski może sprawiać inne problemy niż np. wymowa w innych językach, jak angielski. Polskie głoski, takie jak "sz", "cz", "rz" czy "dż", mają unikalne cechy fonetyczne, które wymagają specyficznych ćwiczeń i technik terapeutycznych. Problemy te są inne niż te napotykane w językach, takich jak angielski, co podkreśla potrzebę narzędzi dostosowanych do specyficznych wymagań fonetycznych języka polskiego. Obecnie na rynku brakuje zaawansowanych aplikacji logopedycznych dedykowanych polskojęzycznym użytkownikom, co sprawia, że moja aplikacja wypełnia ważną niszę. Dzięki temu polscy użytkownicy, zarówno dzieci, jak i dorosli, mogą korzystać z narzędzia dostosowanego do specyficznych potrzeb językowych i kulturowych.

## 2.4. Przegląd zastosowań uczenia maszynowego w analizie mowy

Uczenie maszynowe zrewolucjonizowało analizę mowy, umożliwiając automatyczne ekstrakcje złożonych cech z surowych sygnałów dźwiękowych. Dzięki zastosowaniu głębokich sieci neuronowych, takich jak konwolucyjne sieci neuronowe (CNN) oraz sieci rekurencyjne (RNN), możliwe jest osiągnięcie niespotykanych dotąd wyników w różnych zadaniach związanych z przetwarzaniem mowy. W tym podrozdziale skupiono się na zastosowaniach technik głębokiego uczenia w analizie mowy, ze szczególnym uwzględnieniem CNN, które są szeroko stosowane w aplikacjach zamieniających audio na spektrogramy.

### Automatyczne rozpoznawanie mowy (ASR)

CNN są szeroko stosowane w systemach ASR, gdzie głównym zadaniem jest przekształcenie sygnałów mowy w tekst. Tradycyjnie, sygnały mowy były przetwarzane za pomocą ręcznie projektowanych cech, takich jak MFCC. Wprowadzenie CNN pozwala na automatyczne wyodrębnienie istotnych cech bez potrzeby ręcznego projektowania. W modelach ASR, CNN są często używane do przetwarzania spektrogramów, które są dwuwymiarowymi reprezentacjami sygnału mowy w dziedzinie czasu i częstotliwości. CNN potrafią efektywnie wykrywać lokalne wzorce w spektrogramach, co jest kluczowe dla rozpoznawania fonemów i słów.

### Klasyfikacja dźwięków

Klasyfikacja dźwięków obejmuje identyfikację i klasyfikację różnych dźwięków mowy, takich jak samogłoski, spółgłoski, tony i akcenty. Algorytmy CNN i RNN są szeroko stosowane do analizy cech akustycznych mowy.

## Ocena wymowy

Algorytmy uczenia maszynowego oceniają poprawność wymowy, analizując mowę użytkownika i dostarczając informacje zwrotne na temat dokładności artykulacji. Dzięki tym technologiom użytkownicy mogą otrzymywać spersonalizowane wskazówki, które pomagają w poprawie wymowy, wspierając proces nauki języka lub terapii logopedycznej.

### Analiza sentymenu i emocji w mowie

ML umożliwia analizę emocji i sentymenu w mowie, co jest użyteczne w terapiach zajmujących się problemami emocjonalnymi i behawioralnymi. Technologia *IBM Watson Tone Analyzer* wykorzystuje ML do analizy tonacji mowy i identyfikacji emocji, co jest przydatne w ocenie stanu emocjonalnego pacjentów podczas sesji terapeutycznych.

### Zastosowanie w projekcie

W moim projekcie skupiono się na klasyfikacji wymowy poszczególnych wyrazów na te o prawidłowej wymowie i na te z niepoprawną. Przy użyciu CNN, model analizuje spektrogramy nagrań mowy, klasyfikując je jako poprawne lub niepoprawne. Dzięki temu użytkownik otrzymuje natychmiastową informację zwrotną dotyczącą swojej wymowy, co może znacznie wspomóc proces nauki i korekty błędów.

## 2.5. Podstawy działania sieci neuronowych w kontekście analizy mowy

Sieci neuronowe, szczególnie głębokie sieci neuronowe, stały się podstawą nowoczesnej analizy mowy. Ich zdolność do automatycznego uczenia się z surowych danych audio sprawia, że są niezwykle efektywne w zadaniach takich jak rozpoznawanie mowy, klasyfikacja dźwięków i ocena wymowy. Poniżej przedstawiono podstawowe zasady działania sieci neuronowych w kontekście analizy mowy.

### 2.5.1. Rodzaje sieci neuronowych używanych w analizie mowy

**Konwolucyjne sieci neuronowe (CNN):** Konwolucyjne sieci neuronowe są szeroko stosowane w analizie mowy, szczególnie w rozpoznawaniu wzorców dźwiękowych i przetwarzaniu spektrogramów. CNN są skuteczne w automatycznym wykrywaniu lokalnych wzorców w danych wejściowych poprzez zastosowanie filtrów konwolucyjnych, które przesuwają się po całym obrazie lub spektrogramie, identyfikując kluczowe cechy.

**Rekurencyjne sieci neuronowe (RNN):** Rekurencyjne sieci neuronowe są używane do przetwarzania sekwencyjnych danych, takich jak mowa, ponieważ potrafią przechowywać informacje o poprzednich stanach. Typowym przykładem są sieci LSTM (Long Short-Term Memory) i

GRU (Gated Recurrent Units), które są stosowane w zadaniach rozpoznawania mowy i translacji mowy.

**Sieci Transformer:** Sieci Transformer, takie jak BERT i GPT, rewolucjonizują przetwarzanie języka naturalnego dzięki zdolności do równoległego przetwarzania sekwencji danych. W analizie mowy mogą być używane do zadań takich jak rozpoznawanie mowy i przetwarzanie języka naturalnego.

### 2.5.2. Kluczowe komponenty sieci neuronowych w analizie mowy

**Warstwy konwolucyjne:** Warstwy konwolucyjne są stosowane do ekstrakcji cech z danych wejściowych. W analizie mowy, warstwy te mogą być używane do przetwarzania spektrogramów, identyfikując charakterystyczne wzorce dźwiękowe.

**Warstwy rekurencyjne:** Warstwy rekurencyjne przechowują i przetwarzają sekwencyjne dane, co jest kluczowe w analizie mowy, gdzie kontekst czasowy jest istotny.

**Warstwy normalizacyjne:** Normalizacja danych wejściowych poprawia stabilność i efektywność treningu sieci neuronowej. Jest to szczególnie ważne w analizie mowy, gdzie dane mogą być bardzo zróżnicowane.

**Warstwy dropout:** Warstwy dropout są używane do przeciwdziałania przeuczeniu poprzez losowe wyłączanie neuronów podczas treningu, co zmusza sieć do nauki bardziej uogólnionych wzorców.

### 2.5.3. Proces trenowania sieci neuronowych w analizie mowy

**Przygotowanie danych:** Dane audio są przetwarzane i konwertowane na reprezentacje czasowo-częstotliwościowe, takie jak spektrogramy. Często stosowane są techniki takie jak STFT czy MFCC.

**Definicja architektury sieci:** Wybór odpowiednich warstw i konfiguracji sieci, w tym liczby warstw konwolucyjnych, rekurencyjnych i gęstych, a także parametrów takich jak rozmiar filtrów, liczba neuronów i funkcje aktywacji.

**Trenowanie modelu:** Proces trenowania obejmuje podział danych na zestawy treningowe, walidacyjne i testowe, a następnie optymalizację wag modelu za pomocą algorytmów takich jak Adam czy SGD.

**Ewaluacja i dostrajanie:** Model jest oceniany pod kątem dokładności i efektywności na zestawie testowym, a następnie dostrajany w celu poprawy wydajności. Proces ten może obejmować regulację hiperparametrów, dodawanie warstw regularizacyjnych i dalsze przetwarzanie danych.

### 3. Środowiska pracy

W celu realizacji projektu wybrano odpowiednie narzędzia i środowiska pracy, które umożliwiły efektywne tworzenie, testowanie oraz implementację aplikacji mobilnej oraz powiązanych skryptów do analizy dźwięku. Wybór tych środowisk był kluczowy dla zapewnienia wydajności i komfortu pracy nad projektem.

#### 3.1. Android Studio i Kotlin

Do stworzenia aplikacji mobilnej wybrano środowisko *Android Studio* oraz język *Kotlin*. *Android Studio* to kompleksowe środowisko IDE (Integrated Development Environment), które oferuje szeroki zakres narzędzi niezbędnych do projektowania i budowy aplikacji mobilnych. IDE zapewnia zintegrowane narzędzia do projektowania interfejsów użytkownika, debugowania, testowania aplikacji, a także integracji z różnymi bibliotekami, takimi jak *TensorFlow Lite*, co było istotne w kontekście implementacji modelu uczenia maszynowego w aplikacji. [19]

*Kotlin*, jako język programowania, został wybrany ze względu na jego nowoczesne podejście do programowania, łączące w sobie zalety programowania obiektowego i funkcjonalnego. *Kotlin* jest w pełni interoperacyjny z *Java*, co pozwala na korzystanie z istniejących bibliotek i kodu napisanego w *Java*. Dodatkowo, *Kotlin* zapewnia zwięzły i czytelny kod, co przyspiesza proces tworzenia aplikacji oraz ułatwia jej utrzymanie i dalszy rozwój. [20-21]



Rys. 3.1. Logotypy *Kotlin* (od góry) i *Android Studio* (od dołu)

## 3.2. Visual Studio Code i Python

W przypadku pisania skryptów związanych z nagraniami i modelami, wybór padł na środowisko *Visual Studio Code (VSCode)* oraz język *Python* ze względu na ich kombinację elastyczności, wygody i funkcjonalności. *Python* to język wysokiego poziomu, doskonale nadający się do zadań związanych z przetwarzaniem danych i uczeniem maszynowym. Jego bogata biblioteka narzędzi, takich jak *NumPy*, *TensorFlow* czy *librosa*, pozwala na szybkie i efektywne tworzenie oraz rozwijanie skryptów, które były niezbędne do realizacji projektu. [22]

*VSCode*, z kolei, to edytor kodu, który oferuje wsparcie dla różnych języków programowania, w tym *Pythona*, oraz posiada szeroką gamę rozszerzeń. Ułatwia to pisanie, testowanie i debugowanie kodu dzięki funkcjom takim jak automatyczne uzupełnianie kodu, linting oraz zarządzanie pakietami. Dodatkowo, dzięki wbudowanemu terminalowi, użytkownicy mogą bezpośrednio w środowisku uruchamiać swoje skrypty, co przyspiesza iteracje i procesy testowe.

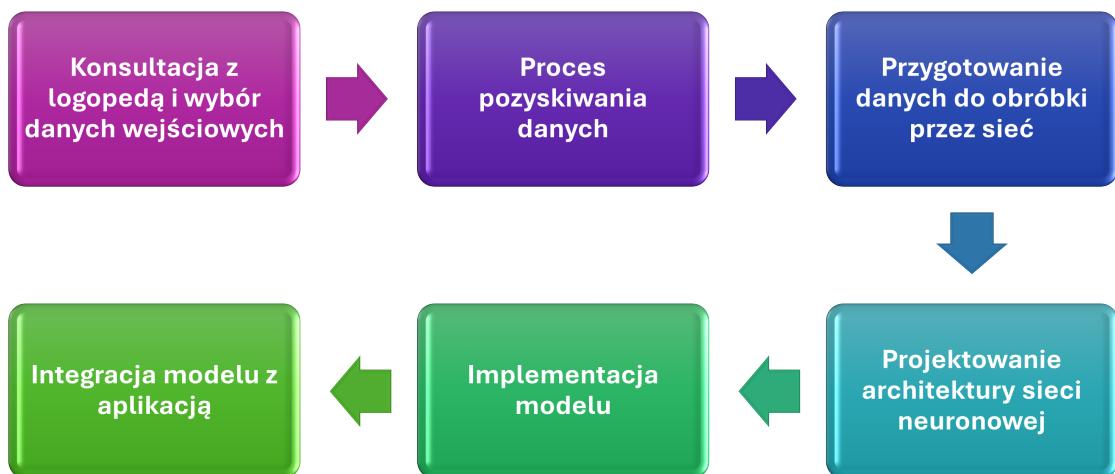
Wybór *Pythona* i *VSCode* był podyktowany potrzebą szybkiego, elastycznego i efektywnego tworzenia kodu, co ma kluczowe znaczenie w dynamicznych projektach związanych z analizą danych i uczeniem maszynowym.



Rys. 3.2. Logotypy *Python* (po lewej) i *Visual Studio Code* (po prawej)

## 4. Tworzenie sieci neuronowej

Proces tworzenia sieci neuronowej wymagał kilku kluczowych kroków, które umożliwiły skuteczne opracowanie aplikacji wspomagającej poprawną wymowę gloski "sz" (Diagram na rys. 4.1.).



Rys. 4.1. Kroki tworzenia sieci neuronowej

### 4.1. Konsultacja z logopedą i wybór danych wejściowych

W celu opracowania skutecznej aplikacji do kształtowania prawidłowej wymowy gloski "sz", skontaktowano się ze wspomnianym wcześniej logopeda, dr Eweliną Strawa-Kęsek z gabinetu "Mowologia". Dr Strawa-Kęsek zaleciła wybranie słów, w których gloska "sz" występuje w nagłosie (na początku wyrazu), śródgłosie (środek wyrazu) i wygłosie (na końcu wyrazu). Zasugerowała również, aby unikać słów zawierających inne dźwięki z tego samego szeregu fonetycznego, co mogłoby utrudnić poprawne rozpoznawanie gloski "sz".

Na podstawie tych wskazówek oraz materiałów zawartych w źródle "Materiał wyrazowo-obrazkowy do utrwalania poprawnej wymowy głosek sz, rz, cz, dz" [23] wybrano następujące wyrazy:

- **Nagłos:** szafa, szopa, szelki
- **Śródgłos:** kasza, nosze, wieszak
- **Wygłos:** kosz, afisz, gulasz

## 4.2. Proces pozyskiwania danych

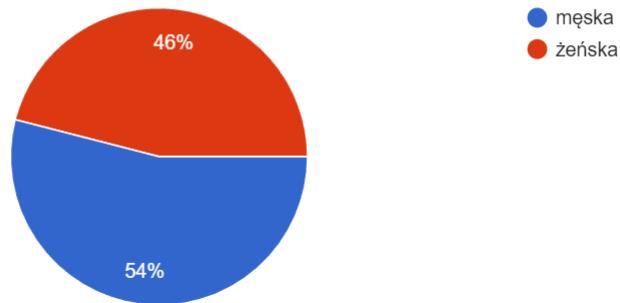
W celu zebrania odpowiednich danych do trenowania sieci neuronowej, przeprowadzono ankietę wśród osób z poprawną wymową głoski "sz". Ankieta miała na celu zebranie nagrani wymowy wybranych wyrazów. Ankieta została przeprowadzona za pomocą formularza *Google* i wysłana do dużej liczby osób. Instrukcje dla uczestników były następujące:

- Znajdź ciche pomieszczenie.
- Uruchom dyktafon.
- Przeczytaj poniższe wyrazy wyraźnie, robiąc lekką przerwę (np. około 1-2 sekundy) między nimi:
  1. *szafa*
  2. *szopa*
  3. *szelki*
  4. *kasza*
  5. *nosze*
  6. *wieszak*
  7. *kosz*
  8. *afisz*
  9. *gulasz*
- Zakończ nagrywanie i zapisz plik (najlepiej w formacie *.wav*, ale mogą być też inne np. *.m4a* lub *.mp3*).
- Wyślij plik.

W ankiecie wzięło udział 50 osób. Zebrane dane demograficzne dotyczące płci, wieku i samooceny poprawności wymowy głoski "sz" przedstawiono na poniższych diagramach (Rys. 4.2.- Rys. 4.4.). Te dane demograficzne pozwoliły na zebranie próbek od zróżnicowanej grupy osób, co jest istotne dla uzyskania reprezentatywnego zestawu danych do trenowania sieci neuronowej. Dzięki temu model mógł być lepiej dostosowany do analizy wymowy w szerokim zakresie demograficznym.

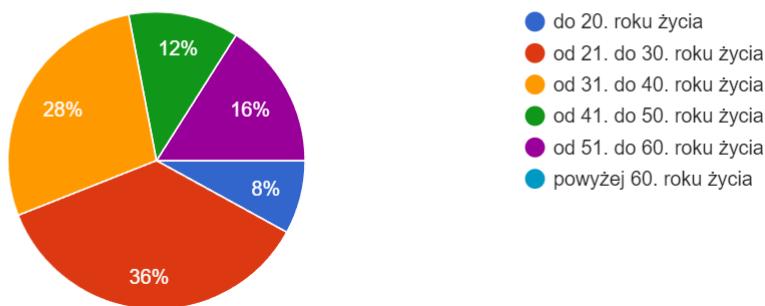
Płeć

50 odpowiedzi

**Rys. 4.2.** Płeć osób ankietowanych

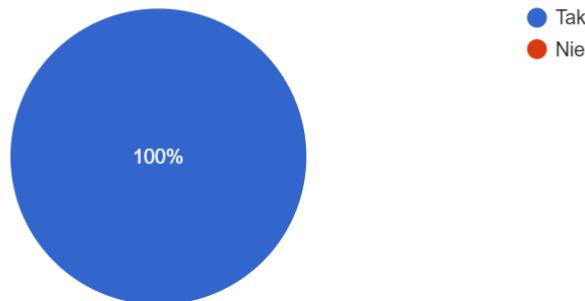
Wiek

50 odpowiedzi

**Rys. 4.3.** Wiek osób ankietowanych

Czy (wg swojej opinii) dobrze wymawiasz głośkę "sz"?

50 odpowiedzi

**Rys. 4.4.** Opinia osób ankietowanych

Oprócz tego, poproszono kilka osób o nagranie dodatkowych próbek, w których próbowali oni celowo zniekształcić wymowę głoski "sz". Na przykład zamiast "szafa", niektórzy uczestnicy mówili "safa". Dzięki tej inicjatywie udało się zebrać 20 nagrań, które zostały następnie włączone do zestawu danych. Te dodatkowe próbki były szczególnie wartościowe, ponieważ umożliwiły modelowi nauczenie się rozpoznawania i klasyfikowania bardziej subtelnych form zniekształceń wymowy.

Jednak najważniejsze były nagrania rzeczywistych zniekształceń wymowy od Pani Doktor i pacjentki. Nagrania od Pani Doktor obejmowały przykłady seplenia bocznego, gdzie dźwięk "sz" wydobywa się z boku ust, oraz seplenia dorsalnego, w którym dźwięk jest produkowany z udziałem tylnej części języka, co tworzy charakterystyczny, zniekształcony dźwięk. Z kolei nagranie od dorosłej pacjentki przedstawało seplenie międzyzębowe, gdzie dźwięk "sz" jest wymawiany z językiem wciśniętym między zęby, co również prowadzi do niepoprawnej artykulacji.

Te nagrania rzeczywistych wad wymowy dostarczyły modelowi kluczowych informacji na temat różnych typów seplenia, co pozwoliło na dokładniejsze i bardziej precyzyjne dostosowanie modelu do wykrywania tego typu wad wymowy.

### 4.3. Przygotowanie danych do obróbki przez sieć

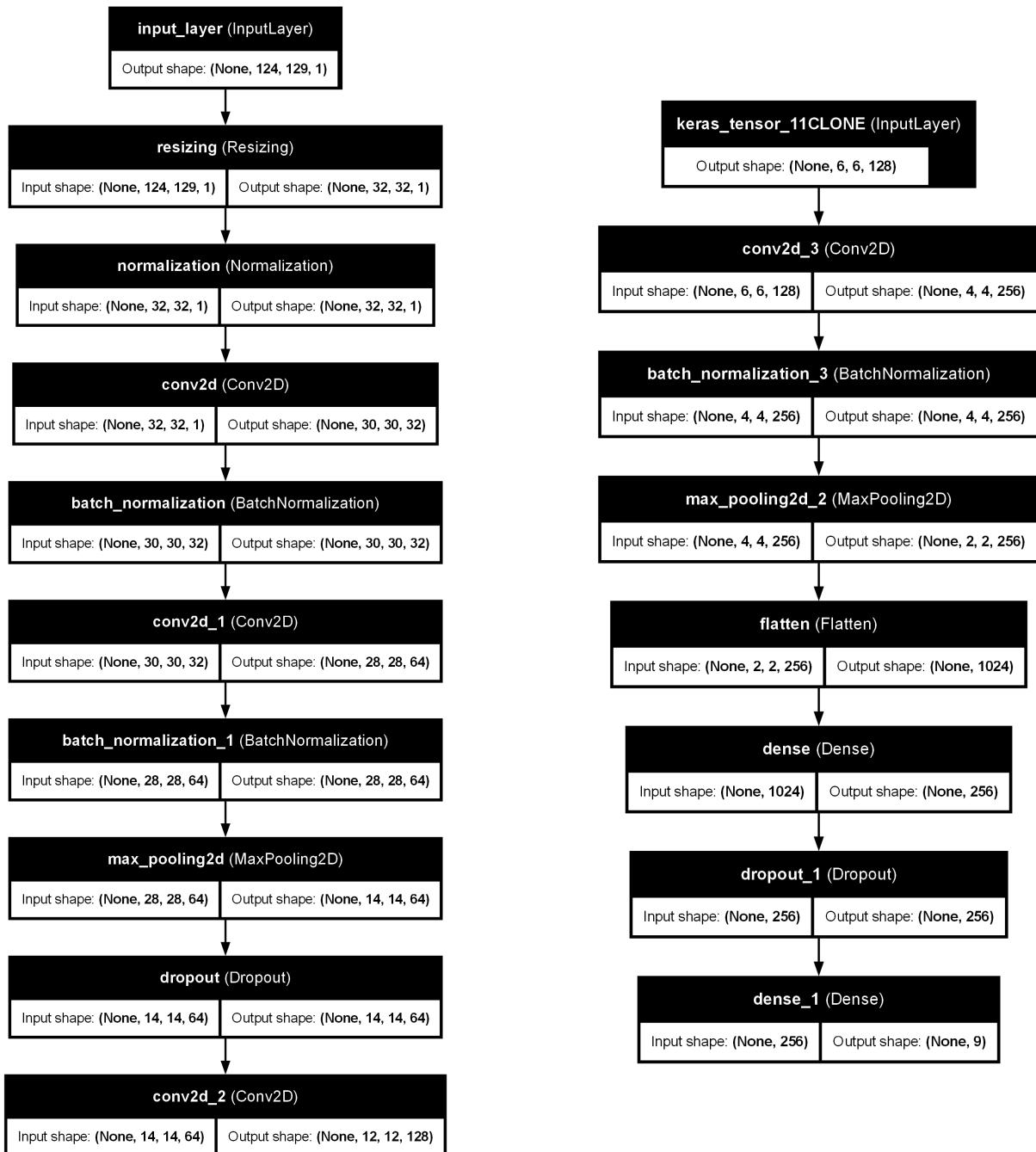
Następnie wszystkie uzyskane nagrania zostały przekonwertowane do formatu .wav. Aby to osiągnąć, opracowano skrypt w *Pythonie*, który automatycznie przekształcał pliki audio oraz sprawdzał, czy nagrania są w mono, czyli zawierają jeden kanał dźwięku, a nie w stereo, gdzie występują dwa kanały. Mono zapewnia jednolity dźwięk, niezależnie od urządzenia odtwarzającego, co jest istotne dla spójnej analizy przez sieć neuronową.

Ze względu na dużą liczbę nagrań, utworzono skrypt, który automatycznie dzielił nagrania na poszczególne słowa, wykrywając momenty ciszy, a następnie zapisywał je w odpowiednich folderach. Parametry, takie jak długość i próg ciszy, były dostosowywane w zależności od nagrań. Po automatycznym podziale każde nagranie było dodatkowo sprawdzane, a w razie potrzeby ręcznie korygowane, aby upewnić się, że podział został przeprowadzony poprawnie i każde nagranie jest gotowe do trenowania modelu.

Aby zwiększyć ilość nagrań w bazie danych, utworzono dodatkowe nagrania poprzez samodzielne modyfikowanie istniejących plików audio. W tym celu zastosowano różne techniki zmiany parametrów dźwięku, takie jak modyfikacja szybkości odtwarzania, zmiana wysokości tonu (półtonów), a także dodanie szumu do nagrań. Każda z tych technik pozwoliła na uzyskanie nowych, unikalnych próbek dźwiękowych, które wzbogaciły bazę danych, zapewniając większą różnorodność nagrań wykorzystywanych do trenowania modelu. Dzięki temu model mógł być bardziej wszechstronny i lepiej radzić sobie z różnorodnymi wariantami wymowy.

## 4.4. Projektowanie architektury sieci neuronowej

W aplikacji wykorzystano konwolucyjną sieć neuronową do analizy nagrani mowy. Sieć została zaprojektowana i zaimplementowana zgodnie z wytycznymi z biblioteki *Tensorflow*. Architektura sieci została dostosowana do specyfiki języka polskiego, koncentrując się na poprawnej wymowie głoski "sz".



Rys. 4.5. Kompletny model podzielony na 2 części

```
1 ...
2
3 # Definicja architektury modelu
4 model = models.Sequential([
5     layers.Input(shape=input_shape),
6     layers.Resizing(32, 32),
7     norm_layer,
8     layers.Conv2D(32, 3, activation='relu'),
9     layers.BatchNormalization(),
10    layers.Conv2D(64, 3, activation='relu'),
11    layers.BatchNormalization(),
12    layers.MaxPooling2D(),
13    layers.Dropout(0.25),
14    layers.Conv2D(128, 3, activation='relu'),
15    layers.BatchNormalization(),
16    layers.MaxPooling2D(),
17    layers.Conv2D(256, 3, activation='relu'),
18    layers.BatchNormalization(),
19    layers.MaxPooling2D(),
20    layers.Flatten(),
21    layers.Dense(256, activation='relu'),
22    layers.Dropout(0.5),
23    layers.Dense(num_labels),
24 ])
25
26 # Kompilacja modelu
27 model.compile(
28     optimizer=tf.keras.optimizers.Adam(),
29     loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
30     metrics=['accuracy'],
31 )
32
33 # Definicja liczby epok do trenowania modelu
34 EPOCHS = 100
35
36 # Trenowanie modelu z użyciem danych treningowych i walidacyjnych
37 history = model.fit(
38     train_ds,
39     validation_data=val_ds,
40     epochs=EPOCHS,
41 )
42
43 ...
```

**Listing 4.1.** Definicja i trenowanie modelu sieci neuronowej w Pythonie

#### 4.4.1. Opis warstw modelu

**Warstwa wejściowa (Input Layer):** Określa kształt danych wejściowych, które model będzie przetwarzał. W tym przypadku dane wejściowe to spektrogramy dźwięków w odpowiednim formacie.

**Warstwy konwolucyjne (Conv2D):** Są to kluczowe warstwy, które wykonują operacje splotu na danych wejściowych. Te warstwy mają za zadanie wyodrębnić istotne cechy z obrazu spektrogramu, takie jak wzory częstotliwościowe charakterystyczne dla poprawnej wymowy głoski "sz". Pierwsza warstwa konwolucyjna ma 32 filtry, druga 64 filtry, a następne warstwy odpowiednio 128 i 256 filtrów, co pozwala na stopniowe zwiększenie złożoności modelu.

**Warstwy normalizacyjne (BatchNormalization):** Te warstwy są używane do normalizacji danych wyjściowych z warstw konwolucyjnych, co przyspiesza trenowanie modelu i poprawia jego stabilność. Normalizacja pomaga w utrzymaniu wartości w sieci w odpowiednim zakresie, co z kolei prowadzi do szybszej konwergencji podczas procesu trenowania.

**Warstwy poolingowe (MaxPooling2D):** Warstwy te redukują wymiarowość danych, skupiając się na najistotniejszych cechach wyodrębnionych przez warstwy konwolucyjne. Pooling pozwala na zmniejszenie ilości parametrów w modelu, co z kolei zmniejsza ryzyko nadmiernego dopasowania (overfittingu).

**Warstwy dropout (Dropout):** Są to warstwy regularizacyjne, które losowo wyłączają niektóre neurony podczas procesu trenowania. Dropout o wartości 0.25 i 0.5 (25% i 50%) pomaga w zapobieganiu przeuczeniu modelu, co zwiększa jego ogólną zdolność do generalizacji na nowych danych.

**Warstwa wyjściowa (Dense):** Końcowa warstwa gęsta (Dense) z odpowiednią liczbą neuronów odpowiada liczbom etykiet (klas) w danych. W tym przypadku, liczba neuronów w tej warstwie odpowiada liczbie możliwych słów, które model ma rozpoznawać.

Model został skompilowany z użyciem optymalizatora *Adam*, co pozwala na szybkie i efektywne dostosowanie wag w sieci. Jako funkcję straty zastosowano *sparseCategoricalCrossentropy*, co jest standardem w przypadku klasifikacji wieloklasowych. Model był trenowany przez 100 epok na zestawie danych treningowych i walidacyjnych.

Powyższa architektura została starannie zaprojektowana, aby efektywnie analizować nagrania mowy, koncentrując się na poprawnej wymowie specyficznej głoski "sz", co jest kluczowym elementem w zastosowaniach logopedycznych aplikacji.

## 4.5. Implementacja modelu

Implementacja modelu sieci neuronowej została przeprowadzona w języku *Python* przy użyciu biblioteki *TensorFlow*. Skrypt zaimplementowany w *notebooku Jupyter* zawierał wszystkie kroki potrzebne do przetworzenia danych wejściowych, trenowania modelu oraz jego ewaluacji. *TensorFlow* został wybrany ze względu na jego wszechstronność i popularność w dziedzinie uczenia maszynowego. Biblioteka ta oferuje zaawansowane narzędzia do budowy, trenowania i wdrażania modeli głębokiego uczenia, a także wspiera szeroki zakres zadań związanych z przetwarzaniem danych. [24-25]

### 4.5.1. Kluczowe etapy implementacji

#### 1. Przygotowanie danych:

- Wczytano zbiory danych zawierające nagrania wymowy słów.
- Pliki audio w formacie WAV zostały przekształcone na spektrogramy za pomocą krótkoterminowej transformaty Fouriera (STFT).
- Zastosowano odpowiednie przekształcenia w celu standaryzacji danych oraz przekształcono je w tensory odpowiednie do dalszego przetwarzania przez model.

#### 2. Definicja modelu CNN:

- Za pomocą *TensorFlow* stworzono model CNN złożony z kilku warstw konwolucyjnych, poolingowych, normalizacyjnych oraz dropout.
- Model został zaprojektowany tak, aby efektywnie analizować spektrogramy i klasyfikować poprawność wymowy słów.

#### 3. Trenowanie modelu:

- Model był trenowany przez 100 epok na zestawie danych treningowych.
- Podczas trenowania monitorowano dokładność modelu na zestawie walidacyjnym, co pozwoliło na bieżące dostosowywanie hiperparametrów w celu osiągnięcia optymalnej wydajności.

#### 4. Ewaluacja modelu:

- Po zakończeniu trenowania model został przetestowany na zestawie testowym.
- Wyniki testów zostały ocenione przy użyciu macierzy błędów, co umożliwiło analizę skuteczności modelu w rozpoznawaniu poprawnej wymowy słów.

## 4.6. Integracja modelu z aplikacją

Integracja modelu uczenia maszynowego z aplikacją mobilną została przeprowadzona w taki sposób, aby zapewnić płynne działanie i łatwość użytkowania. Wytrenowany model został skonwertowany do formatu *TensorFlow Lite*, co pozwoliło na jego efektywne uruchamianie na urządzeniach mobilnych.

### 4.6.1. Proces integracji

#### 1. Konwersja modelu do *TensorFlow Lite*:

- Model sieci neuronowej został przekonwertowany do formatu *TensorFlow Lite*.
- W tym celu użyto narzędzi dostępnych w *TensorFlow*, które umożliwiają optymalizację modelu pod kątem wydajności na mobilnych urządzeniach z ograniczonymi zasobami.

#### 2. Integracja aplikacji z *Pythonem*:

- Aplikacja korzysta z *Chaquopy* do wywoływania funkcji zdefiniowanych w skryptach *Pythona*, co pozwala na generowanie spektrogramów bezpośrednio na urządzeniu mobilnym.
- Skrypt ten przetwarza nagranie audio, generuje odpowiednie wizualizacje oraz przekształca je do formatu akceptowanego przez model *TensorFlow Lite*.

#### 3. Wywoływanie modelu w aplikacji:

- Aplikacja mobilna zaimplementowana w *Kotlinie* używa API *TensorFlow Lite* do wczytania modelu i przetwarzania danych.
- Pliki audio wczytywane przez użytkownika są konwertowane na spektrogramy, które następnie są analizowane przez model.
- Wyniki są zwracane do aplikacji, gdzie są prezentowane użytkownikowi.

#### 4. Testowanie i optymalizacja:

- Po integracji aplikacji z modelem przeprowadzono testy, aby upewnić się, że aplikacja działa zgodnie z oczekiwaniemi na różnych urządzeniach.
- Przeprowadzono również optymalizacje, takie jak kompresja modelu oraz redukcja rozdzielczości spektrogramów, aby zwiększyć wydajność aplikacji.

## 5. Tworzenie aplikacji "Erkorektor"

Aplikacja *Erkorektor* została stworzona jako narzędzie do wspierania terapii logopedycznej. Potrzebna była nazwa, która będzie łatwa do zapamiętania i jednocześnie oddawała funkcjonalność aplikacji. Nazwa *Erkorektor* powstała z mojego zamiłowania do języka hiszpańskiego oraz modyfikacji z "El" na "Er", ze względu na osobiste trudności z wymową głoski "r". Nazwa została specjalnie dobrana tak, aby osoby z problemami wymowy (jak ja) mogły już od samego początku ćwiczyć wymawianie tej trudnej głoski, co czyni ją pierwszym krokiem w procesie terapeutycznym.

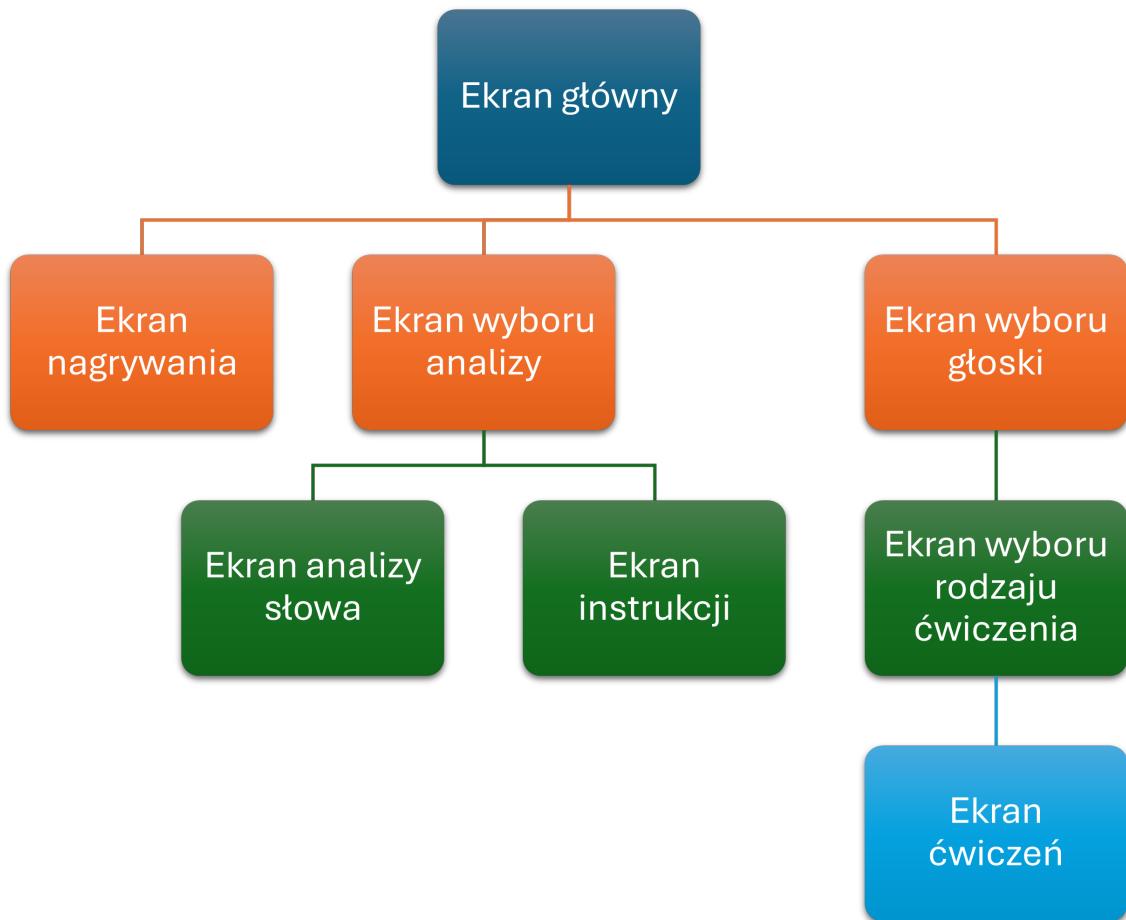


Rys. 5.1. Logo aplikacji

W celu skonfigurowania środowiska aplikacji mobilnej wykorzystano *Android Studio*, *Kotlin* oraz *TensorFlow Lite*, co umożliwia uruchamianie modeli uczenia maszynowego bezpośrednio na urządzeniach mobilnych. Aplikacja została zbudowana z użyciem API level 34 oraz minimalnym SDK 24. API (Application Programming Interface) to zestaw reguł i protokołów, który pozwala różnym elementom oprogramowania na komunikację. W kontekście Androida, API level odnosi się do wersji zestawu narzędzi programistycznych, które są dostępne dla deweloperów. SDK (Software Development Kit) to zestaw narzędzi i bibliotek, które pozwalają na tworzenie aplikacji na określoną platformę, w tym przypadku na Androida. Wybór API level 34 i minimalnego SDK 24 zapewnia kompatybilność aplikacji z szerokim zakresem urządzeń mobilnych.

Zintegrowano *Chaquopy*, co pozwala na bezpośrednie uruchamianie skryptów Pythona w aplikacji Android, ułatwiając integrację z *TensorFlow Lite*. Dodatkowo, za pomocą *Gradle* zarządzano zależnościami projektu. *Gradle* to narzędzie automatyzujące proces budowania aplikacji, które pomaga w zarządzaniu zewnętrznymi bibliotekami, takimi jak *TensorFlow Lite*, oraz optymalizuje proces kompilacji. Dzięki tej konfiguracji aplikacja może działać efektywnie i wydajnie na urządzeniach mobilnych, korzystając z modeli uczenia maszynowego.

Poniżej znajduje się uproszczona mapa oraz szczegółowy opis stworzonych okien aplikacji *Erkorektor*.



Rys. 5.2. Uproszczona mapa aplikacji

## 5.1. Ekran główny

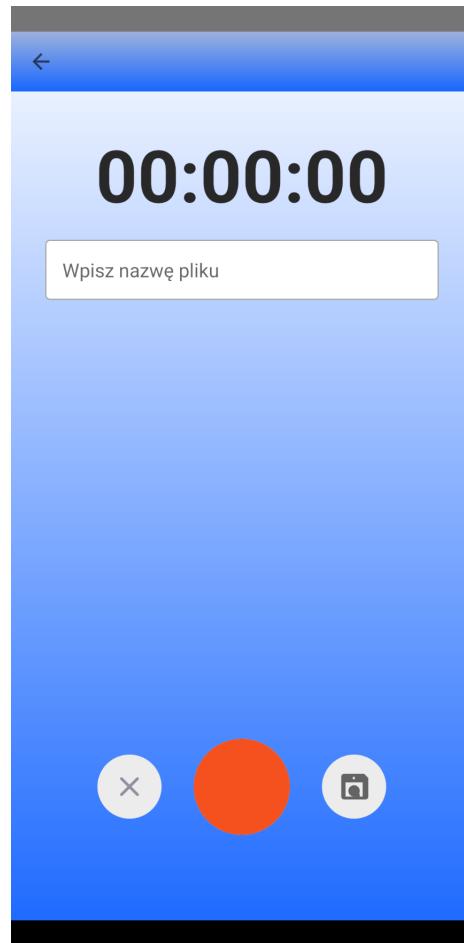


Rys. 5.3. Ekran główny

Ekran główny jest prosty i intuicyjny, pozwalający użytkownikom na szybkie rozpoczęcie pracy z aplikacją. Znajdują się na nim trzy główne przyciski: "NAGRYWANIE", "ANALIZA MOWY" oraz "ĆWICZENIA". Użytkownik może łatwo przejść do nagrywania swojej mowy, analizowania nagrań pod kątem poprawności wymowy oraz wykonywania ćwiczeń logopedycznych. Ekran zawiera:

- **Przycisk "NAGRYWANIE":** Umożliwia użytkownikowi przejście do obszaru nagrywania, gdzie może zarejestrować swoją wypowiedź.
- **Przycisk "ANALIZA MOWY":** Przenosi użytkownika do obszaru analizy, w którym nagranie jest przetwarzane przy użyciu modelu *TensorFlow Lite*, aby ocenić poprawność wymowy.
- **Przycisk "ĆWICZENIA":** Umożliwia dostęp do obszaru ćwiczeń logopedycznych, wspierających użytkownika w doskonaleniu wymowy.

## 5.2. Ekran nagrywania



Rys. 5.4. Ekran nagrywania

Ekran nagrywania jest kluczowym elementem, umożliwiającym użytkownikom nagrywanie ich wypowiedzi. Ekran ten jest zaprojektowany z myślą o prostocie i funkcjonalności, oferując użytkownikom możliwość łatwego nagrywania, zapisywania oraz usuwania nagrań. Ekran zawiera:

- **Obsługa uprawnień:** Aplikacja weryfkuje, czy użytkownik udzielił uprawnień do nagrywania dźwięku. W przypadku braku uprawnień, użytkownik jest proszony o ich przyznanie.
- **Timer:** Wyświetla czas trwania nagrania w czasie rzeczywistym, aktualizując się co sekundę, aby śledzić bieżący czas nagrania.
- **Pole tekstowe do wprowadzania nazwy pliku:** Umożliwia użytkownikowi wprowadzenie nazwy dla nagrania. Jeśli nazwa nie zostanie podana, aplikacja automatycznie generuje ją na podstawie aktualnej daty i godziny.

- **Przycisk nagrywania:** Służy do rozpoczętia i zatrzymania nagrywania. Po naciśnięciu przycisku rozpoczyna się nagrywanie, a ścieżka zapisu pliku zostaje automatycznie ustalona. Ponowne naciśnięcie przycisku zatrzymuje nagrywanie, a urządzenie przygotowuje się do kolejnego nagrania.
- **Przycisk zapisu:** Umożliwia zapisanie nagranego pliku, pod warunkiem że nagranie zostało zakończone i posiada nazwę. Po zapisaniu pliku użytkownik otrzymuje komunikat o pomyślnym zapisaniu.
- **Przycisk usuwania:** Pozwala na usunięcie nagranego pliku. Jeśli plik istnieje, jest usuwany, a użytkownik otrzymuje informację o jego usunięciu. Po usunięciu pliku timer jest resetowany, a nazwa pliku jest czyszczona.
- **Przycisk powrotu do menu:** Umożliwia powrót do ekranu głównego aplikacji, znajduje się w lewym górnym rogu ekranu (symbol strzałki w lewo).

### 5.3. Ekran wyboru analizy

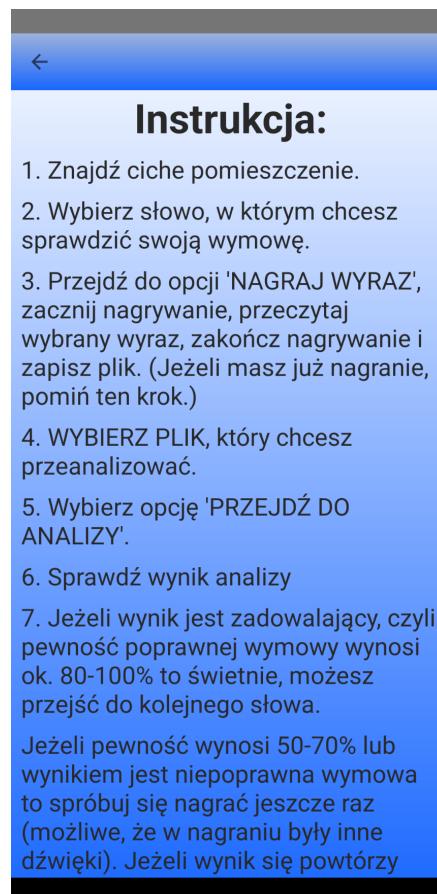


Rys. 5.5. Ekran wyboru analizy

Ekran wyboru umożliwia użytkownikom wybór słowa, które chcą analizować. W tym wypadku użytkownik ma do wyboru kilka przycisków, z których każdy odpowiada innemu słowu zawierającemu głoskę "sz". Ekran wyboru zawiera:

- **Przyciski wyboru słowa:** Każdy przycisk odpowiada jednemu z wybranych słów zawierających głoskę "sz", takich jak "SZAFA", "SZOPA", "SZEŁKI", "KASZA", "NOSZE", "WIESZAK", "KOSZ", "AFISZ", "GULASZ". Po naciśnięciu przycisku użytkownik zostaje przekierowany do ekranu analizy mowy, gdzie może przeprowadzić analizę wymowy wybranego słowa.
- **Przycisk instrukcji:** Przycisk przekierowuje użytkownika do ekranu z instrukcjami korzystania z aplikacji. Umożliwia użytkownikowi zapoznanie się z krokami niezbędnymi do prawidłowego korzystania z analizy.
- **Przycisk powrotu do menu:** Znajduje się w lewym górnym rogu ekranu (strzałka w lewo). Umożliwia użytkownikowi powrót do ekranu głównego aplikacji.

## 5.4. Ekran instrukcji



Rys. 5.6. Ekran instrukcji

Ekran instrukcji w aplikacji został zaprojektowany tak, aby dostarczać użytkownikom przejrzyste wskazówki dotyczące korzystania z aplikacji oraz przeprowadzania analizy poprawności wymowy. Górną część ekranu zawiera pasek narzędzi, który umożliwia powrót do poprzedniego ekranu za pomocą przycisku powrotu, zachowując spójność z resztą aplikacji.

Centralnym elementem ekranu jest szczegółowa instrukcja, która prowadzi użytkownika krok po kroku przez cały proces analizy mowy. Instrukcja składa się z następujących kroków:

1. **Przygotowanie miejsca:** Użytkownik jest zachęcany do znalezienia cichego pomieszczenia, aby uniknąć zakłóceń w nagraniu.
2. **Wybór słowa:** Użytkownik wybiera słowo, które chce przeanalizować pod kątem poprawności wymowy.
3. **Nagrywanie wyrazu:** Użytkownik ma możliwość nagrania swojej wymowy wybranego słowa, korzystając z opcji nagrywania dostępnej w aplikacji.
4. **Wybór pliku do analizy:** Po nagraniu, użytkownik wybiera plik audio, który chce przeanalizować.
5. **Rozpoczęcie analizy:** Użytkownik uruchamia proces analizy wymowy wybranego słowa.
6. **Sprawdzenie wyników:** Aplikacja wyświetla wyniki analizy, oceniąc poprawność wymowy.
7. **Ocena wyników i dalsze kroki:** Instrukcja informuje, co zrobić w zależności od wyników analizy – jeśli wynik jest zadowalający, użytkownik może przejść do kolejnego słowa; w przeciwnym razie jest sugerowane ponowne nagranie lub przejście do ćwiczeń korekcyjnych.

Instrukcja jest napisana w prosty, zrozumiały sposób, aby była dostępna dla użytkowników w każdym wieku.

## 5.5. Ekran analizy słowa

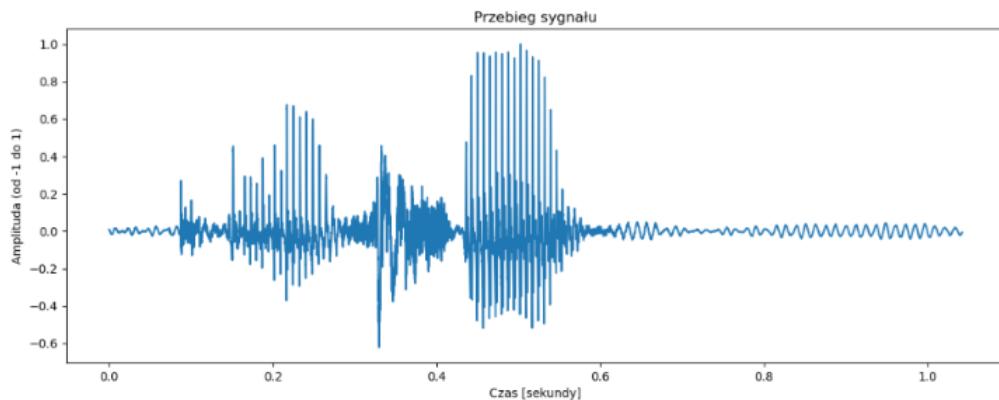


Rys. 5.7. Ekran analizy słowa

Na ekranie analizy słowa użytkownik ma możliwość przeprowadzenia analizy wymowy wybranego słowa na podstawie nagrania audio. Ekran ten został zaprojektowany w taki sposób, aby użytkownik mógł intuicyjnie przeprowadzić proces analizy, od nagrania słowa, przez wybór pliku, aż po wyświetlenie wyników analizy. Elementy ekranu:

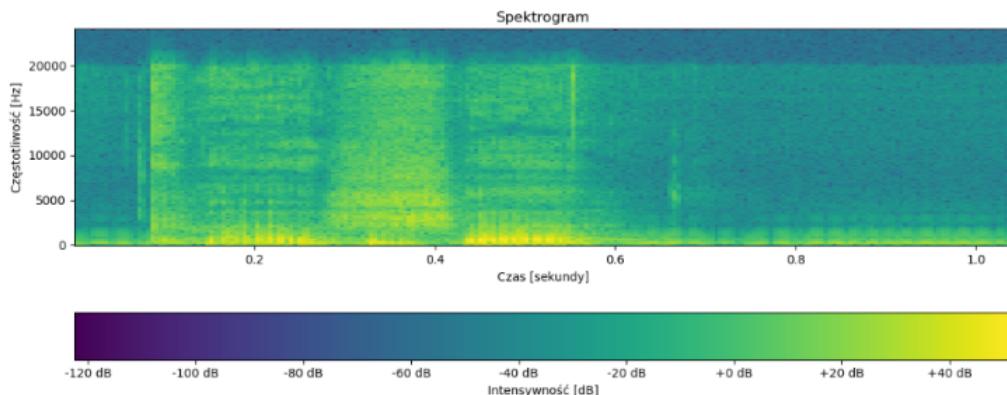
- **Przycisk "NAGRAJ WYRAZ":** Po kliknięciu tego przycisku użytkownik jest przekierowany do ekranu nagrywania, gdzie może zarejestrować swoje nagranie.
- **Przycisk "WYBIERZ PLIK":** Pozwala na wybór już nagranego pliku audio z pamięci urządzenia. Po wybraniu pliku, jego ścieżka jest wyświetlana poniżej, a plik jest gotowy do analizy.
- **Przycisk "PRZEJDŹ DO ANALIZY":** Rozpoczyna proces analizy wybranego pliku audio. Po zakończeniu analizy wyniki są wyświetlane na tym samym ekranie.

- **Wynik analizy:** Po przeprowadzeniu analizy wyświetlany jest przewidywany wynik, który określa, czy wymowa słowa była poprawna (*Dobrze*) lub niepoprawna (*Źle*). Wynik jest również opatrzony wskaźnikiem pewności, wyrażonym w procentach, który informuje o poziomie pewności modelu co do poprawności wymowy.
- **Przebieg sygnału:** Graficzna reprezentacja sygnału dźwiękowego wybranego nagrania, pokazująca, jak dźwięk zmienia się w czasie. Możliwe przybliżanie obrazu w aplikacji.



Rys. 5.8. Przebieg sygnału

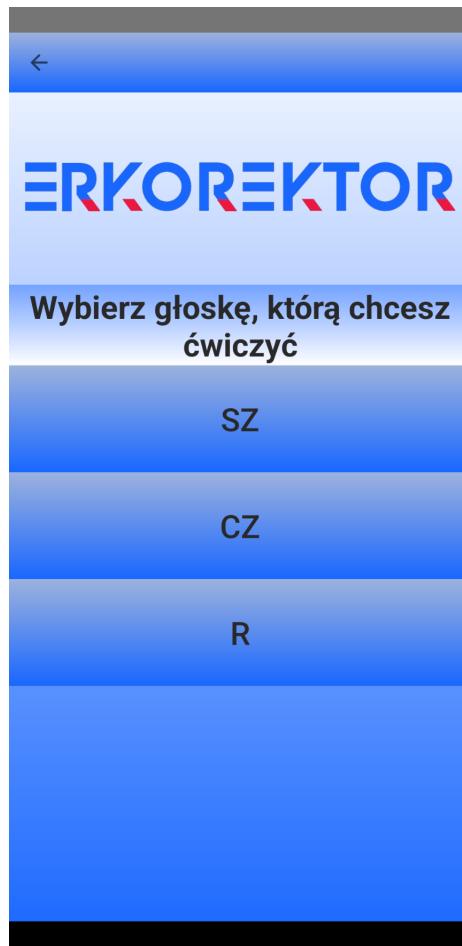
- **Spektrogram:** Dwuwymiarowa wizualizacja spektrum częstotliwości sygnału audio w funkcji czasu. Spektrogram pokazuje, które częstotliwości były obecne w danym momencie nagrania i z jaką intensywnością. Możliwe przybliżanie obrazu w aplikacji.



Rys. 5.9. Spektrogram

Obrazy przebiegu sygnału oraz spektrogramu zostały zawarte w aplikacji z myślą o ich przyszłym zastosowaniu w korekcji wymowy. Choć spektrogramy są już wykorzystywane w modelu do analizy dźwięku, to te wizualizacje mogą w przyszłości wspierać bardziej zaawansowane funkcje aplikacji, takie jak automatyczne wykrywanie błędów w wymowie i sugerowanie konkretnych korekt. Dzięki temu narzędzia te mogłyby znaczco wspierać pracę logopedów i zwiększać efektywność terapii.

## 5.6. Ekran wyboru ćwiczeń głoski

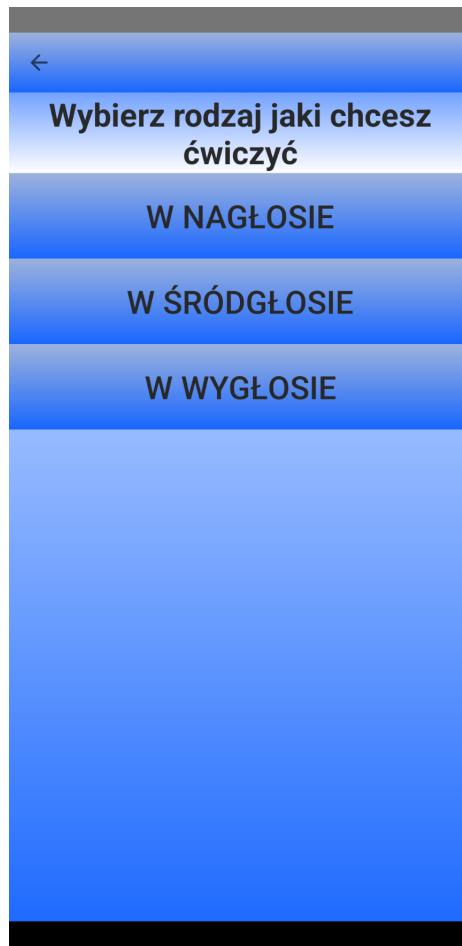


Rys. 5.10. Ekran wyboru ćwiczeń głoski

Ekran wyboru głoski w aplikacji *Erkorektor* umożliwia użytkownikom wybór jednej z dostępnych głosek do ćwiczeń. Ekran zawiera:

- **Przycisk powrotu:** Znajduje się w lewym górnym rogu ekranu (strzałka w lewo). Umożliwia użytkownikowi powrót do ekranu głównego aplikacji.
- **Przyciski wyboru głoski:** Na ekranie użytkownik może wybrać jedną z trzech głosek do ćwiczeń: *SZ*, *CZ*, i *R*. Dla głosek *SZ* i *R* zostały już przygotowane odpowiednie zestawy ćwiczeń. Z kolei dla głoski *CZ* zestawy ćwiczeń są w fazie przygotowania i zostaną dodane w przyszłych aktualizacjach aplikacji.

## 5.7. Ekran wyboru rodzaju ćwiczenia



Rys. 5.11. Ekran wyboru rodzaju ćwiczenia

Ekran pozwala użytkownikowi na wybór, który rodzaj ćwiczeń chce wykonywać, skupiając się na wybranej wcześniej glosce (w tym wypadku dla głoski *sz*). Ekran zawiera:

- **Przycisk powrotu:** Znajduje się w lewym górnym rogu ekranu (strzałka w lewo) i umożliwia użytkownikowi powrót do poprzedniego ekranu, gdzie można wybrać inną głoskę do ćwiczeń.
- **Przyciski wyboru pozycji głoski:** Ekran zawiera trzy przyciski, każdy z nich odpowiada za ćwiczenia związane z pozycją głoski w słowie. Dla głoski *sz* przyciski kierują użytkownika do odpowiednich ćwiczeń: *W NAGŁOSIE* (na początku wyrazu), *W ŚRÓDGŁOSIE* (w środku wyrazu) oraz *W WYGŁOSIE* (na końcu wyrazu).

Dla głoski *r* ćwiczenia te obejmują wymowę np. po literach: *D, F, M, S, Sz, T, W, CH*.

## 5.8. Ekran ćwiczeń



Rys. 5.12. Ekrany ćwiczeń

Ekran ćwiczenia w aplikacji *Erkorektor* umożliwia użytkownikom ćwiczenie wymowy wybranych słów (tutaj SZ w nagłosie). Ekran jest zaprojektowany tak, aby użytkownik mógł łatwo przeglądać zestawy słów i sylab, które mają ćwiczyć. Ekran ćwiczenia zawiera:

- **Przycisk powrotu:** Znajduje się w lewym górnym rogu ekranu (strzałka w lewo). Umożliwia użytkownikowi powrót do ekranu wyboru rodzaju ćwiczenia.
- **Nagłówek:** Wyświetla aktualnie ćwiczony segment, taki jak np. "Sza-", "Szo-", "Szu-", "Sze-", "Szy-". Nagłówek jest umieszczony na środku ekranu, aby użytkownik od razu wiedział, na którym segmencie się skupia.
- **Lista słów:** Pod nagłówkiem znajduje się lista słów do ćwiczenia, odpowiadających aktualnie wybranemu segmentowi. Każdy zestaw słów jest wyświetlany w postaci listy, którą użytkownik może łatwo przeglądać.

- **Przyciski nawigacyjne:** Dwa przyciski nawigacyjne umieszczone na dole ekranu umożliwiają przełączanie między zestawami słów. Przycisk "Poprzednie" pozwala wrócić do poprzedniego zestawu, a przycisk "Następne" przechodzi do kolejnego zestawu.

Przy projektowaniu tych ćwiczeń opierano się na osobistych doświadczeniach, kiedy intensywnie ćwiczono wymowę głoski *r*. W praktyce tej regularnie powtarzano różnorodne słowa, aby wyuczyć odpowiednią pracę języka potrzebną do wymawiania tej głoski. Dzięki tym doświadczeniom opracowano skuteczne strategie, które teraz znalazły zastosowanie w aplikacji. Poniżej znajduje się lista słów, które można znaleźć w tej aplikacji:

**Dla głoski sz w nagłosie:**

- **Sza-**: Szafa, Szalik, Szampon, Szansa, Szałas, Szabla, Szamanka, Szatnia
- **Szo-**: Szok, Szosa, Szopa, Szopka, Szogun, Szop, Szogunat
- **Szu-**: Szum, Szukać, Szuflada, Szumieć, Szubin, Szubak, Szumowina, Szumny
- **Sze-**: Szelki, Sześć, Szept, Szef, Sześciopak, Szelest, Szefelin, Sześćset
- **Szy-**: Szyba, Szybko, Szycie, Szyjka, Szyld, Szybki, Szybowiec, Szyfr

**Dla głoski sz w śródgłosie:**

- **a-sz-**: Pasza, Kasza, Maszyna, Pasztet, Laszka, Baszta, Naszyjnik
- **e-sz-**: Mieszać, Meszek, Pieszy, Leszek, Zesztywnieć, Pieszy, Zeszyt, Leszy, Wieszak
- **i-sz-**: Kiszka, Wiszący, Fiszka, Wisząco, Liszek, Liszaj, Nisza, Cisza
- **o-sz-**: Koszyk, Kosztować, Kosztowny, Koszt, Oszukać, Poszukać, Oszpecić, Nosze
- **u-sz-**: Muszla, Dusza, Kusza, Kuszenie, Muszka, Uszkodzenie, Susza

**Dla głoski sz w wygłosie:**

- **-asz**: Apasz, Gnasz, Ufasz, Znasz, Grasz, Dumasz, Dobywasz, Wydasz, Okiwasz, Zmywasz, Gulasz
- **-esz**: Wiesz, Jesz, Ciesz, Dowiesz, Naciesz, Olejesz, Ugniesz, Dylujesz, Balujesz
- **-isz**: Bawisz, Pisz, Koisz, Wpisz, Zapisz, Ucisz, Kosisz, Spalisz, Dusisz, Afisz
- **-osz**: Kosz, Długosz, Grosz, Klosz, Łosz, Płosz, Kawosz, Smakosz
- **-usz**: Busz, Susz, Tusz, Osusz, Sojusz, Fundusz, Zagłusz, Kapelusz, Ślabeusz

W zestawach ćwiczeń znajdują się również przykłady innych głosek, jak w przypadku ćwiczeń wymowy głoski *r* po literach takich jak *D, F, M, S, Sz, T, W, CH*.

## 6. Testowanie i wyniki

### 6.1. Początkowy test dla 9 słów

Pierwszym krokiem była wstępna weryfikacja działania modelu. Przetestowano go na zestawie danych zawierającym 9 różnych słów: *szafa, szopa, szelki, kasza, nosze, wieszak, kosz, afisz, gulasz*. Ten wstępny test miał na celu ocenę, czy model jest w stanie poprawnie rozpoznać wymowę tych słów przed dodaniem danych zawierających wypowiedzi zniekształcione. Każde z tych słów miało po 500 nagrań. Dla celów trenowania, walidacji i testowania modelu, dane te zostały podzielone w następujący sposób:

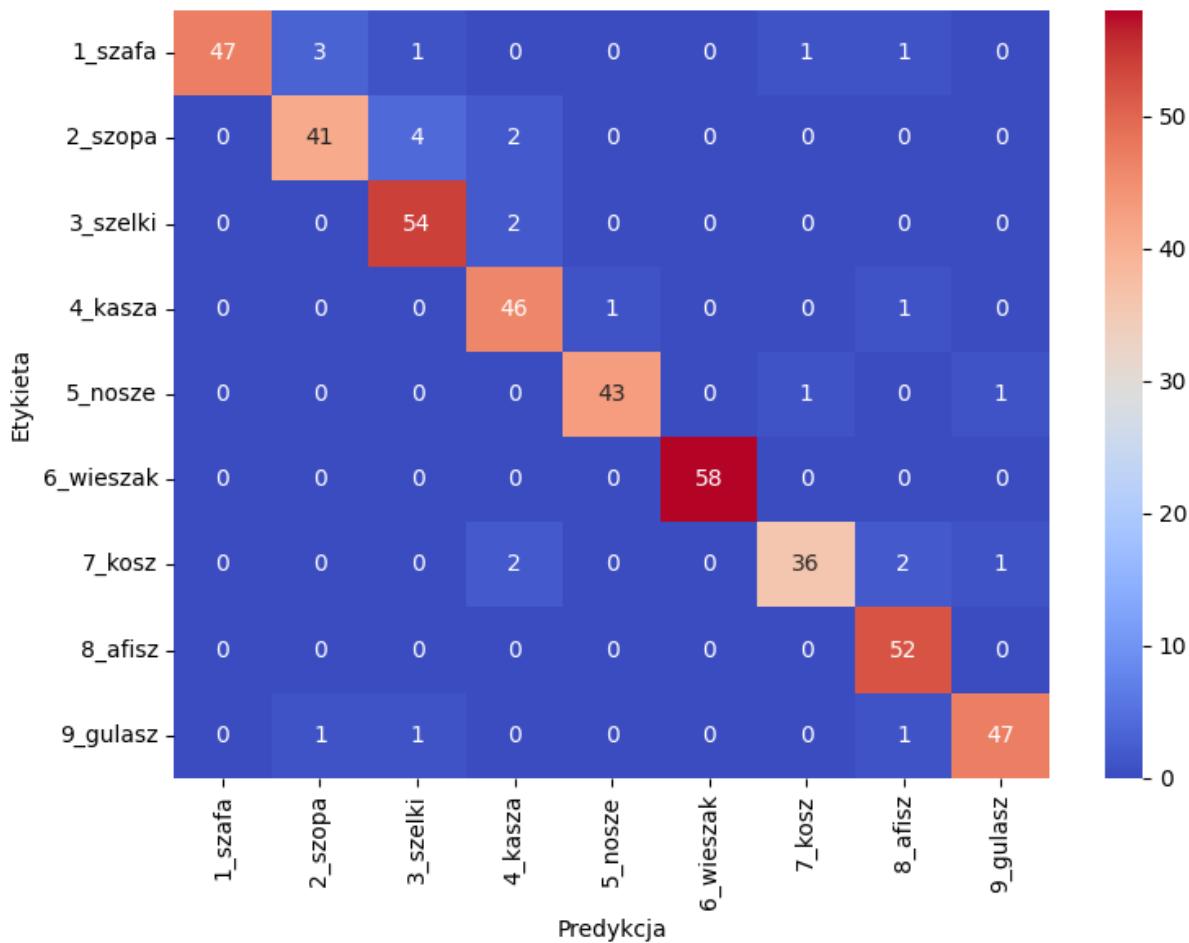
- **Zestaw treningowy:** 3600 nagrań (80% danych), użyty do nauki modelu.
- **Zestaw walidacyjny:** 450 nagrań (10% danych), użyty do monitorowania wydajności modelu w trakcie trenowania i zapobiegania przeuczeniu.
- **Zestaw testowy:** 450 nagrań (10% danych), użyty do ostatecznej oceny skuteczności modelu po zakończeniu procesu trenowania.

Model został poddany 100 iteracjom trenowania (epokom), w trakcie których zoptymalizowano jego zdolność do klasyfikacji wymowy. Każda iteracja umożliwiała modelowi stopniowe doskonalenie umiejętności rozpoznawania wzorców dźwiękowych, co skutkowało poprawą jego wydajności.

Po zakończeniu trenowania, model został przetestowany na zestawie testowym. Wyniki zostały przedstawione w postaci macierzy błędów, co pozwoliło na dokładną analizę wydajności modelu.

Model osiągnął dokładność na poziomie **94%** w rozpoznawaniu poprawności wymowy słów z zestawu testowego.

Większość słów została poprawnie sklasyfikowana, co świadczy o wysokiej efektywności modelu. Zidentyfikowano jednak pewne drobne błędy klasyfikacyjne, zwłaszcza w przypadku słów o podobnych dźwiękach, takich jak *szafa* i *szopa*, czy *szopa* i *szelki*, czyli wyrazy z *sz* w nagłówku. Błędy te wskazują na wyzwania związane z rozróżnieniem podobnych dźwięków, co jest typowym problemem w zadaniach związanych z rozpoznawaniem mowy.



Rys. 6.1. Macierz błędów dla wszystkich słów

Pomimo tych niewielkich błędów, ogólna dokładność modelu wskazuje na jego wysoką skuteczność i użyteczność w kontekście oceny wymowy w aplikacji *Erkorektor*. Wyniki testów potwierdziły skuteczność modelu w prawidłowej klasyfikacji wymowy. Dopiero po tej wstępnej weryfikacji model został rozszerzony o dodatkowe dane zawierające zniekształccone wypowiedzi, co umożliwiło dalsze doskonalenie jego zdolności rozpoznawania mowy.

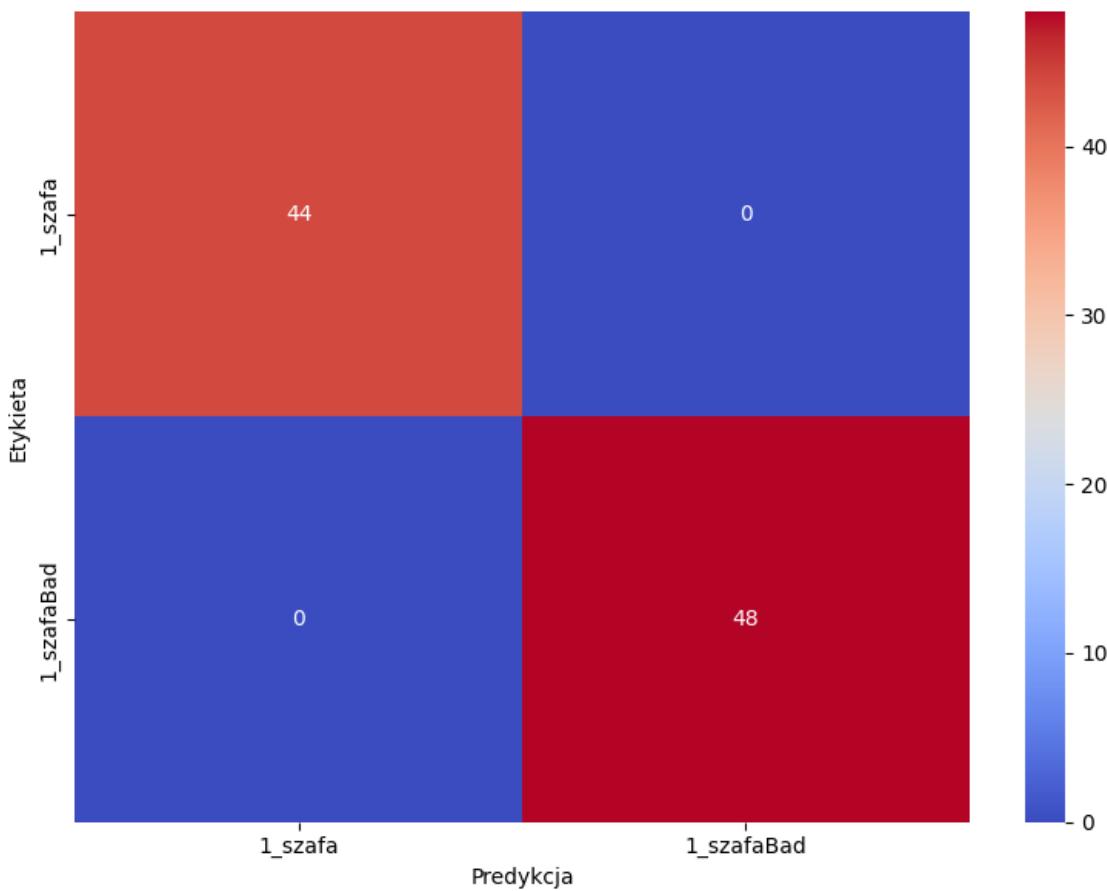
## 6.2. Test dla każdego słowa

Po przeprowadzeniu wstępniego testu modelu na zestawie 9 słów, przystąpiono do szczegółowych testów dla każdego wybranego słowa z osobna, aby sprawdzić, jak model radzi sobie z rozróżnianiem poprawnej i niepoprawnej wymowy. Dla każdego słowa nagrania zostały podzielone na dwie kategorie: **poprawna wymowa** (500 nagrań) i **niepoprawna wymowa** (420 nagrań). Następnie, dla każdego z tych słów, dane zostały podzielone na trzy zestawy:

- **Zestaw treningowy:** 736 nagrań (80% danych), użyty do nauki modelu.
- **Zestaw walidacyjny:** 92 nagrań (10% danych), użyty do monitorowania wydajności modelu w trakcie trenowania i zapobiegania przeuczeniu.
- **Zestaw testowy:** 92 nagrań (10% danych), użyty do ostatecznej oceny skuteczności modelu po zakończeniu procesu trenowania.

Celem tych testów było zbadanie, czy model potrafi skutecznie rozróżnić między poprawną a niepoprawną wymową dla każdego z 9 słów. Po zakończeniu procesu trenowania, który trwał 100 iteracji (epok), model został przetestowany na zestawie testowym. Wyniki każdego testu zostały przedstawione w postaci macierzy błędów, co pozwoliło na szczegółową analizę wydajności modelu.

### 6.2.1. Szafa



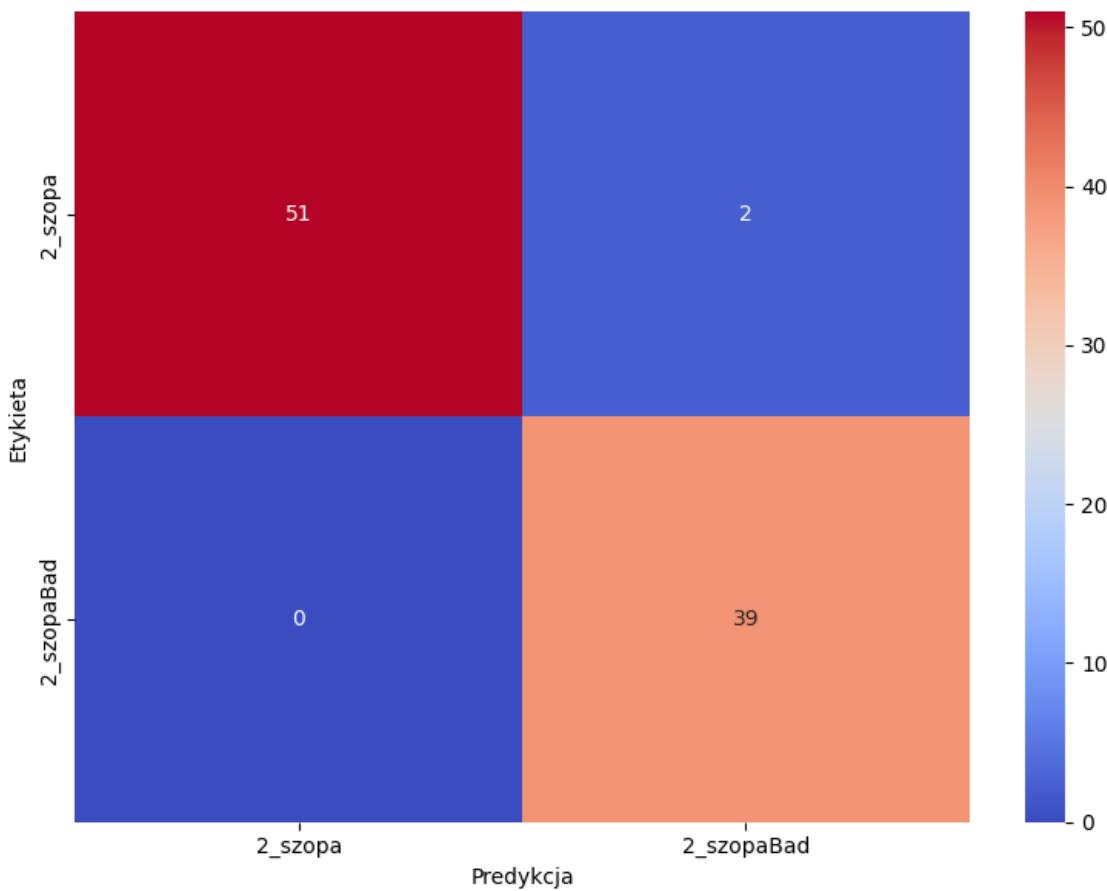
Rys. 6.2. Szafa - Macierz błędów

Macierz błędów przedstawiona na rysunku 6.2 ukazuje skuteczność modelu w klasyfikacji wymowy słowa "Szafa". Na podstawie tej macierzy można zauważyc, że model osiągnął bardzo wysoką dokładność w rozróżnianiu między poprawną a niepoprawną wymową:

- Poprawna wymowa (etykieta: *1\_szafa*) została poprawnie sklasyfikowana w 44 przypadkach, a model nie popełnił żadnych błędów w rozpoznaniu poprawnej wymowy.
- Niepoprawna wymowa (etykieta: *1\_szafaBad*) została poprawnie rozpoznana w 48 przypadkach, co oznacza, że model również nie popełnił żadnych błędów w klasyfikacji tej kategorii.

Model osiągnął dokładność na poziomie **100%** w rozpoznawaniu poprawności wymowy słów z zestawu testowego i wykazał doskonałą skuteczność w rozpoznawaniu zarówno poprawnej, jak i niepoprawnej wymowy słowa "Szafa", co świadczy o bardzo wysokiej precyzji modelu.

## 6.2.2. Szopa



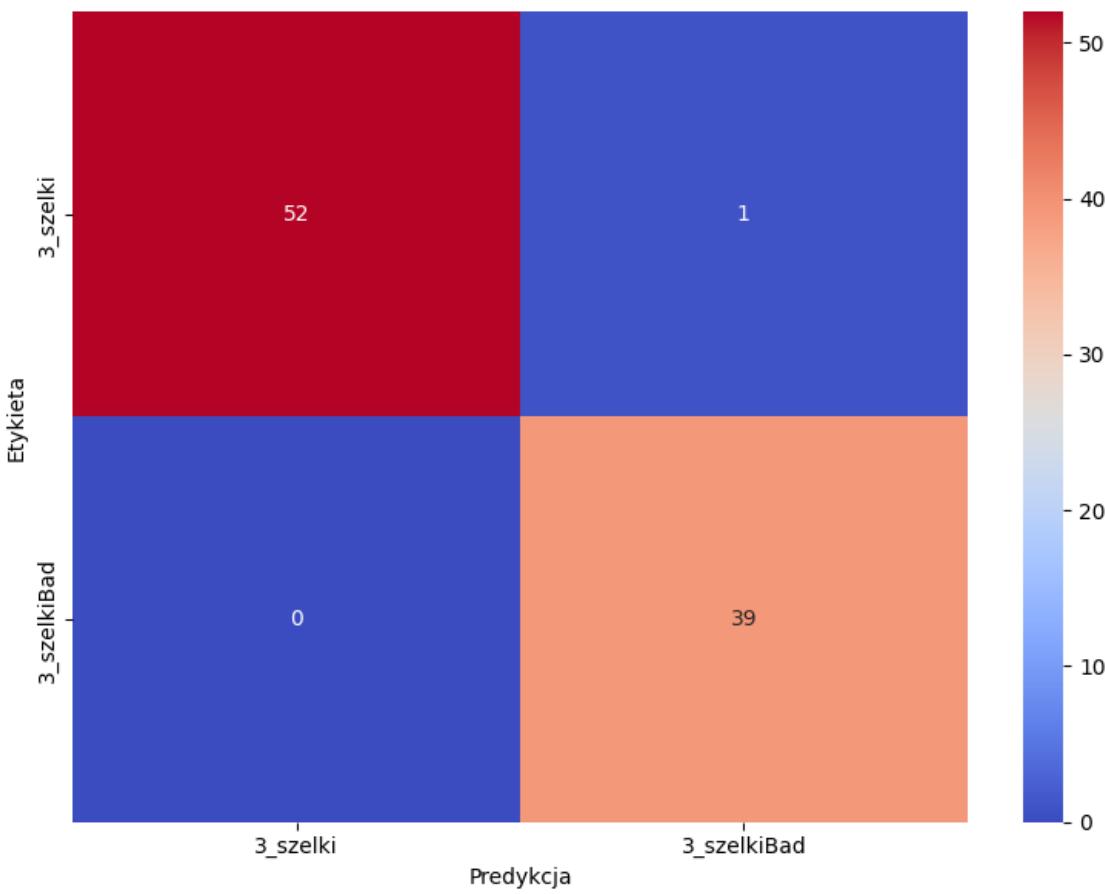
Rys. 6.3. Szopa - Macierz błędów

Macierz błędów przedstawiona na rysunku 6.3 ukazuje skuteczność modelu w klasyfikacji wymowy słowa "Szopa". Na podstawie tej macierzy można zauważyc, że model osiągnął bardzo wysoką dokładność w rozróżnianiu między poprawną a niepoprawną wymową:

- Poprawna wymowa (etykieta: 2\_szopa) została poprawnie sklasyfikowana w 51 przypadkach, a jedynie 2 przypadki zostały błędnie zaklasyfikowane jako niepoprawna wymowa.
- Niepoprawna wymowa (etykieta: 2\_szopaBad) została poprawnie rozpoznana w 39 przypadkach, co oznacza, że model nie popełnił żadnych błędów w klasyfikacji tej kategorii.

Model osiągnął dokładność na poziomie **98%** w rozpoznawaniu poprawności wymowy słów z zestawu testowego i wykazał doskonałą skuteczność w rozpoznawaniu zarówno poprawnej, jak i niepoprawnej wymowy słowa "Szopa", co świadczy o bardzo wysokiej precyzji modelu.

### 6.2.3. Szelki



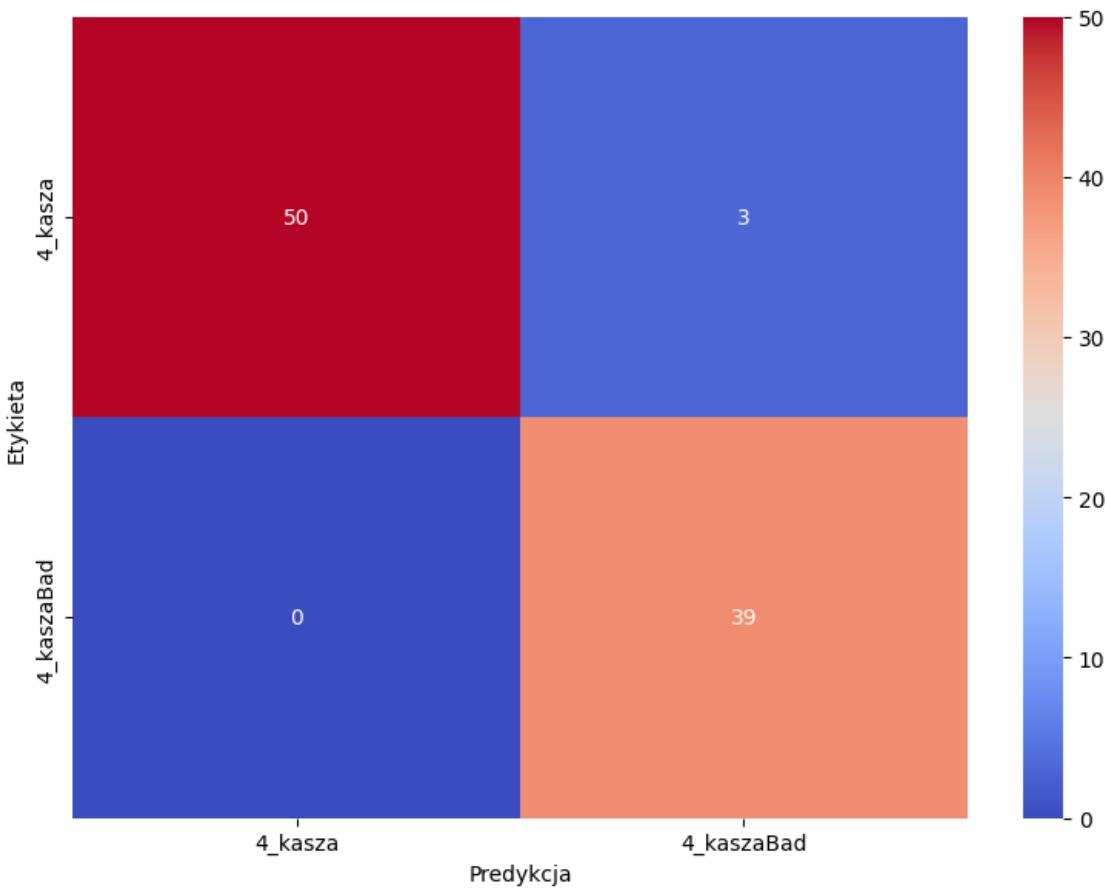
Rys. 6.4. *Szelki* - Macierz błędów

Macierz błędów przedstawiona na rysunku 6.4 ukazuje skuteczność modelu w klasyfikacji wymowy słowa "Szelki". Na podstawie tej macierzy można zauważyc, że model osiągnął bardzo wysoką dokładność w rozróżnianiu między poprawną a niepoprawną wymową:

- Poprawna wymowa (etykieta: *3\_szelki*) została poprawnie sklasyfikowana w 52 przypadkach, a jedynie 1 przypadek został błędnie zaklasyfikowany jako niepoprawna wymowa.
- Niepoprawna wymowa (etykieta: *3\_szelkiBad*) została poprawnie rozpoznana w 39 przypadkach, co oznacza, że model nie popełnił żadnych błędów w klasyfikacji tej kategorii.

Model osiągnął dokładność na poziomie **99%** w rozpoznawaniu poprawności wymowy słów z zestawu testowego i wykazał doskonałą skuteczność w rozpoznawaniu zarówno poprawnej, jak i niepoprawnej wymowy słowa "Szelki", co świadczy o bardzo wysokiej precyzji modelu.

### 6.2.4. Kasza



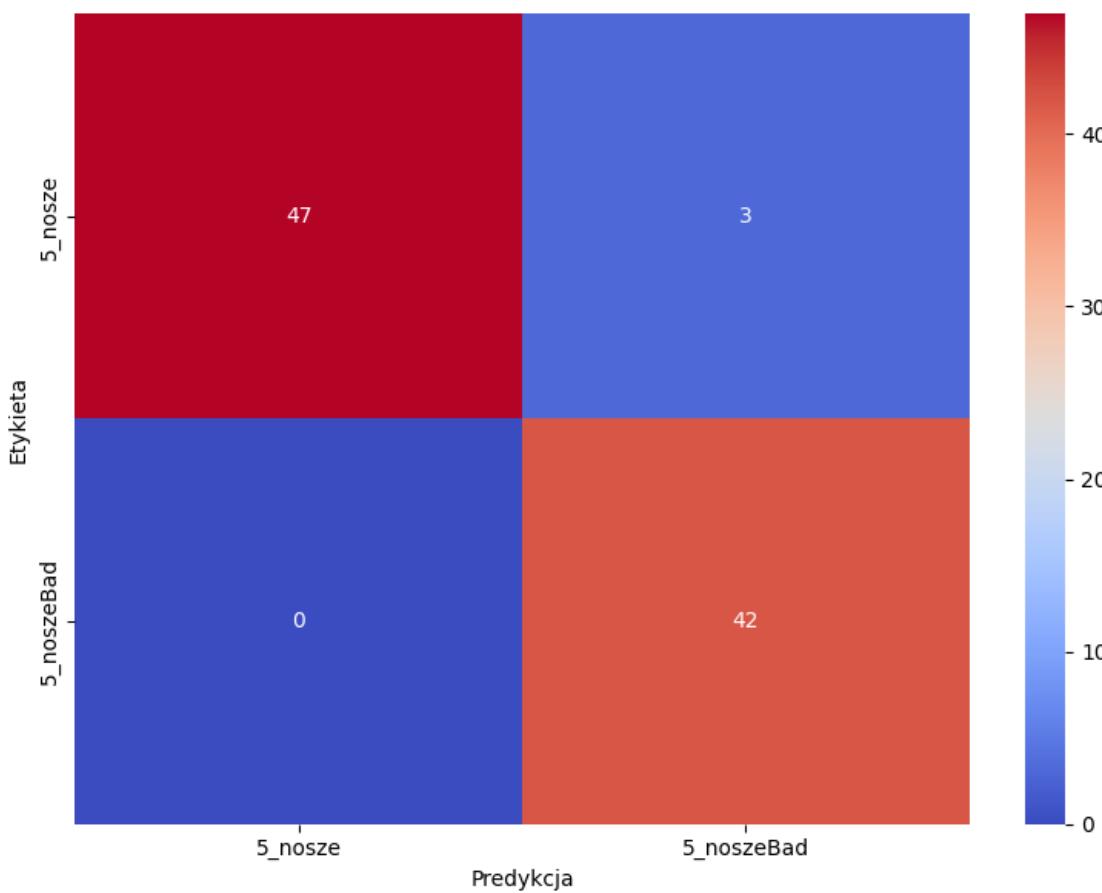
Rys. 6.5. Kasza - Macierz błędów

Macierz błędów przedstawiona na rysunku 6.5 ukazuje skuteczność modelu w klasyfikacji wymowy słowa "Kasza". Na podstawie tej macierzy można zauważyć, że model osiągnął bardzo wysoką dokładność w rozróżnianiu między poprawną a niepoprawną wymową:

- Poprawna wymowa (etykieta: 4\_kasza) została poprawnie sklasyfikowana w 50 przypadkach, a 3 przypadki zostały błędnie zaklasyfikowane jako niepoprawna wymowa.
- Niepoprawna wymowa (etykieta: 4\_kaszaBad) została poprawnie rozpoznana w 39 przypadkach, co oznacza, że model nie popełnił żadnych błędów w klasyfikacji tej kategorii.

Model osiągnął dokładność na poziomie **97%** w rozpoznawaniu poprawności wymowy słów z zestawu testowego i wykazał doskonałą skuteczność w rozpoznawaniu zarówno poprawnej, jak i niepoprawnej wymowy słowa "Kasza", co świadczy o bardzo wysokiej precyzji modelu.

### 6.2.5. Nosze



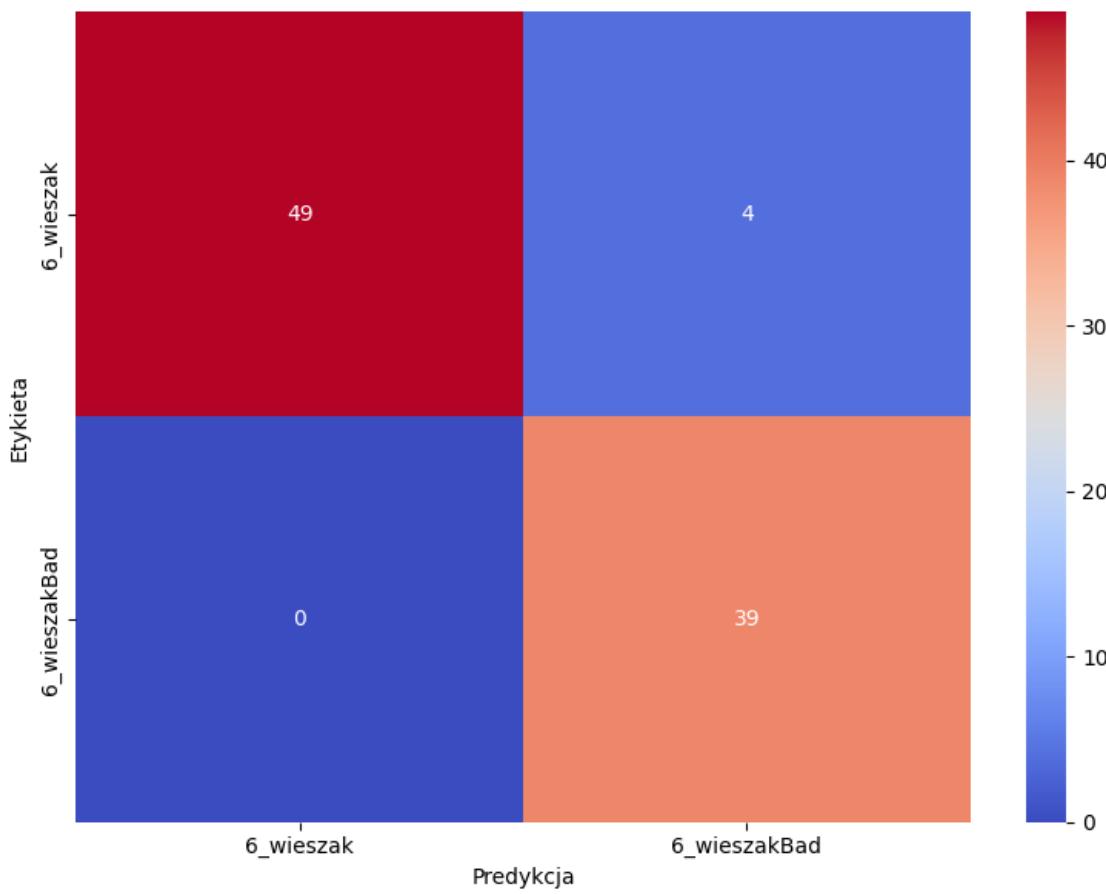
Rys. 6.6. *Nosze* - Macierz błędów

Macierz błędów przedstawiona na rysunku 6.6 ukazuje skuteczność modelu w klasyfikacji wymowy słowa "Nosze". Na podstawie tej macierzy można zauważyc, że model osiągnął bardzo wysoką dokładność w rozróżnianiu między poprawną a niepoprawną wymową:

- Poprawna wymowa (etykieta: *5\_nosze*) została poprawnie sklasyfikowana w 47 przypadkach, a 3 przypadki zostały błędnie zaklasyfikowane jako niepoprawna wymowa.
- Niepoprawna wymowa (etykieta: *5\_noszeBad*) została poprawnie rozpoznana w 42 przypadkach, co oznacza, że model nie popełnił żadnych błędów w klasyfikacji tej kategorii.

Model osiągnął dokładność na poziomie **97%** w rozpoznawaniu poprawności wymowy słów z zestawu testowego i wykazał doskonałą skuteczność w rozpoznawaniu zarówno poprawnej, jak i niepoprawnej wymowy słowa "Nosze", co świadczy o bardzo wysokiej precyzji modelu.

### 6.2.6. Wieszak



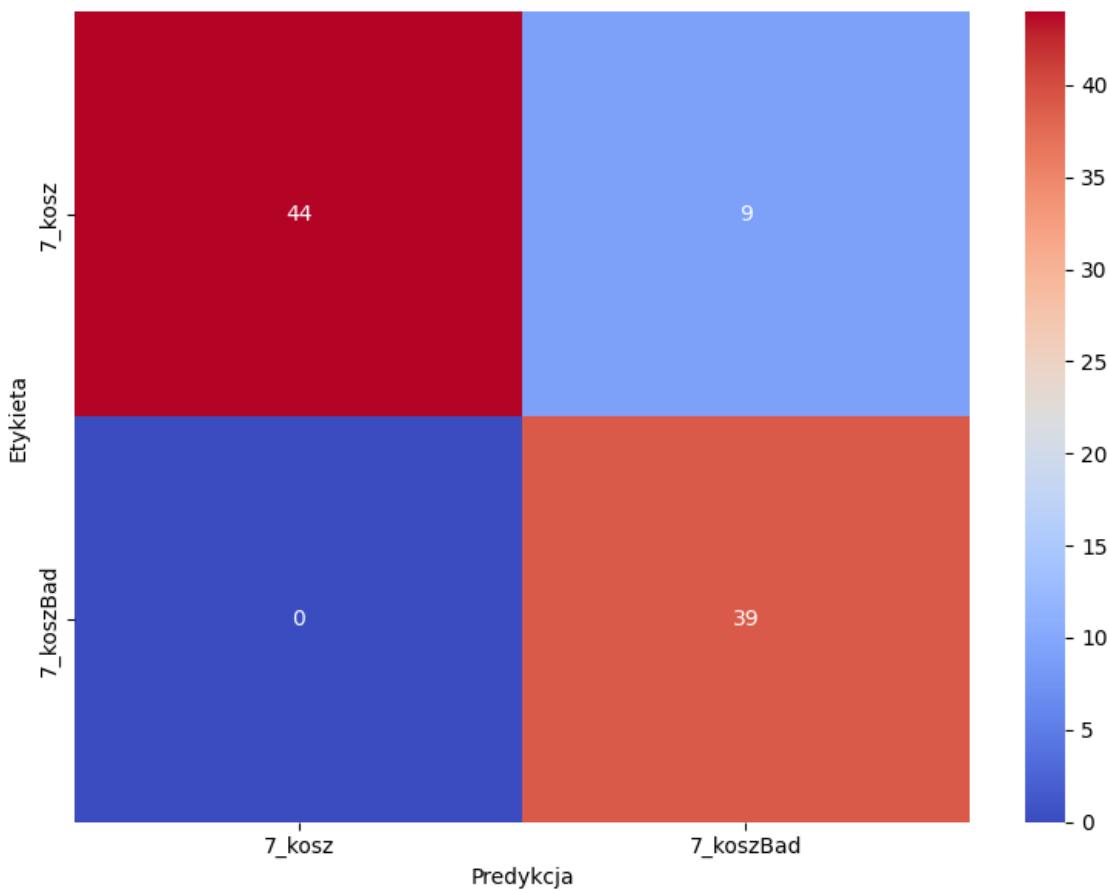
Rys. 6.7. Wieszak - Macierz błędów

Macierz błędów przedstawiona na rysunku 6.7 ukazuje skuteczność modelu w klasyfikacji wymowy słowa "*Wieszak*". Na podstawie tej macierzy można zauważyć, że model osiągnął bardzo wysoką dokładność w rozróżnianiu między poprawną a niepoprawną wymową:

- Poprawna wymowa (etykieta: *6\_wieszak*) została poprawnie sklasyfikowana w 49 przypadkach, a 4 przypadki zostały błędnie zaklasyfikowane jako niepoprawna wymowa.
- Niepoprawna wymowa (etykieta: *6\_wieszakBad*) została poprawnie rozpoznana w 39 przypadkach, co oznacza, że model nie popełnił żadnych błędów w klasyfikacji tej kategorii.

Model osiągnął dokładność na poziomie **96%** w rozpoznawaniu poprawności wymowy słów z zestawu testowego i wykazał doskonałą skuteczność w rozpoznawaniu zarówno poprawnej, jak i niepoprawnej wymowy słowa "*Wieszak*", co świadczy o bardzo wysokiej precyzji modelu.

### 6.2.7. Kosz



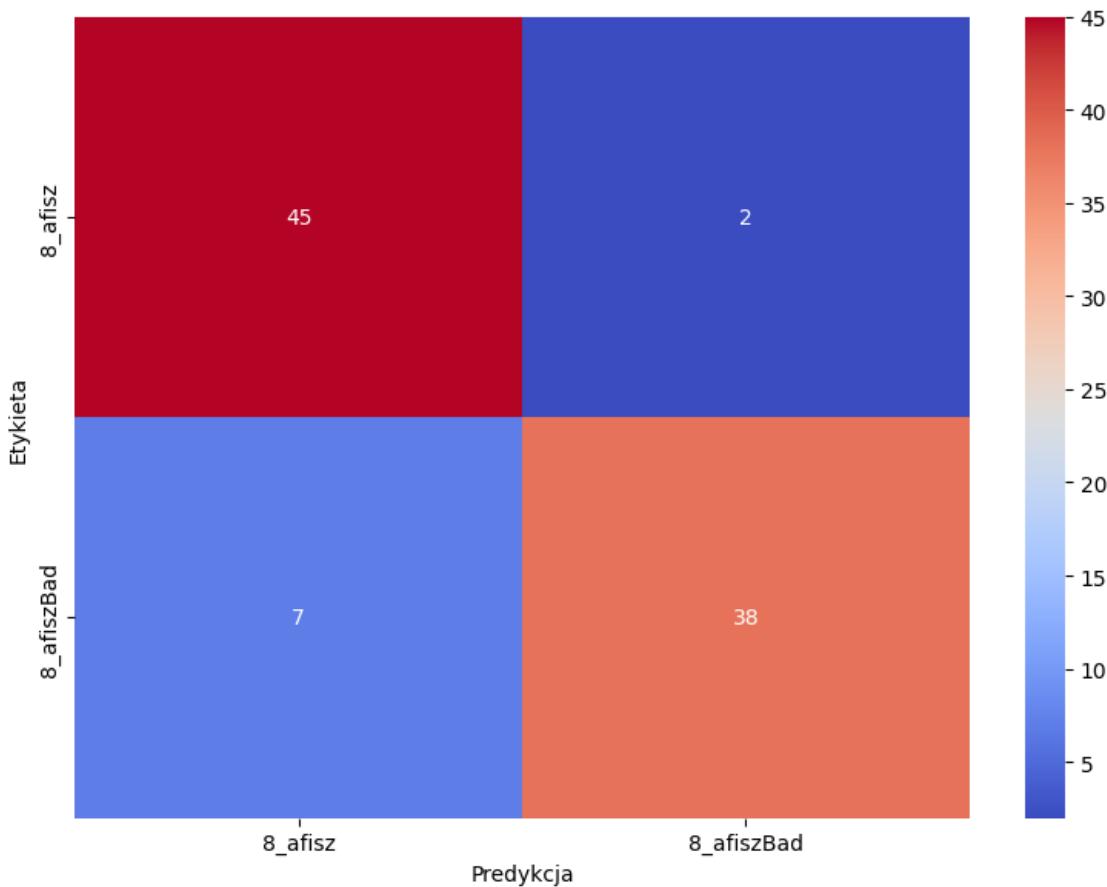
Rys. 6.8. Kosz - Macierz błędów

Macierz błędów przedstawiona na rysunku 6.8 ilustruje wydajność modelu w klasyfikacji wymowy słowa "Kosz". Analiza wyników tej macierzy pokazuje, że mimo ogólnie dobrych rezultatów, model napotkał pewne trudności w rozróżnianiu poprawnej i niepoprawnej wymowy tego słowa:

- *Poprawna wymowa (etykieta: 7\_kosz)* została poprawnie sklasyfikowana w 44 przypadkach, jednak 9 przypadków zostało błędnie zaklasyfikowanych jako niepoprawna wymowa.
- *Niepoprawna wymowa (etykieta: 7\_koszBad)* została poprawnie rozpoznana w 39 przypadkach, bez błędnej klasyfikacji jako poprawna wymowa.

Mimo że model napotkał pewne trudności w rozróżnianiu poprawnej i niepoprawnej wymowy słowa "Kosz", jego ogólna dokładność pozostaje stosunkowo wysoka, osiągając **90%**. Wskaźnik ten świadczy o solidnej wydajności modelu, choć istnieje miejsce na dalsze doskonalenie, aby zwiększyć precyzję i zminimalizować błędy w klasyfikacji.

### 6.2.8. Afisz



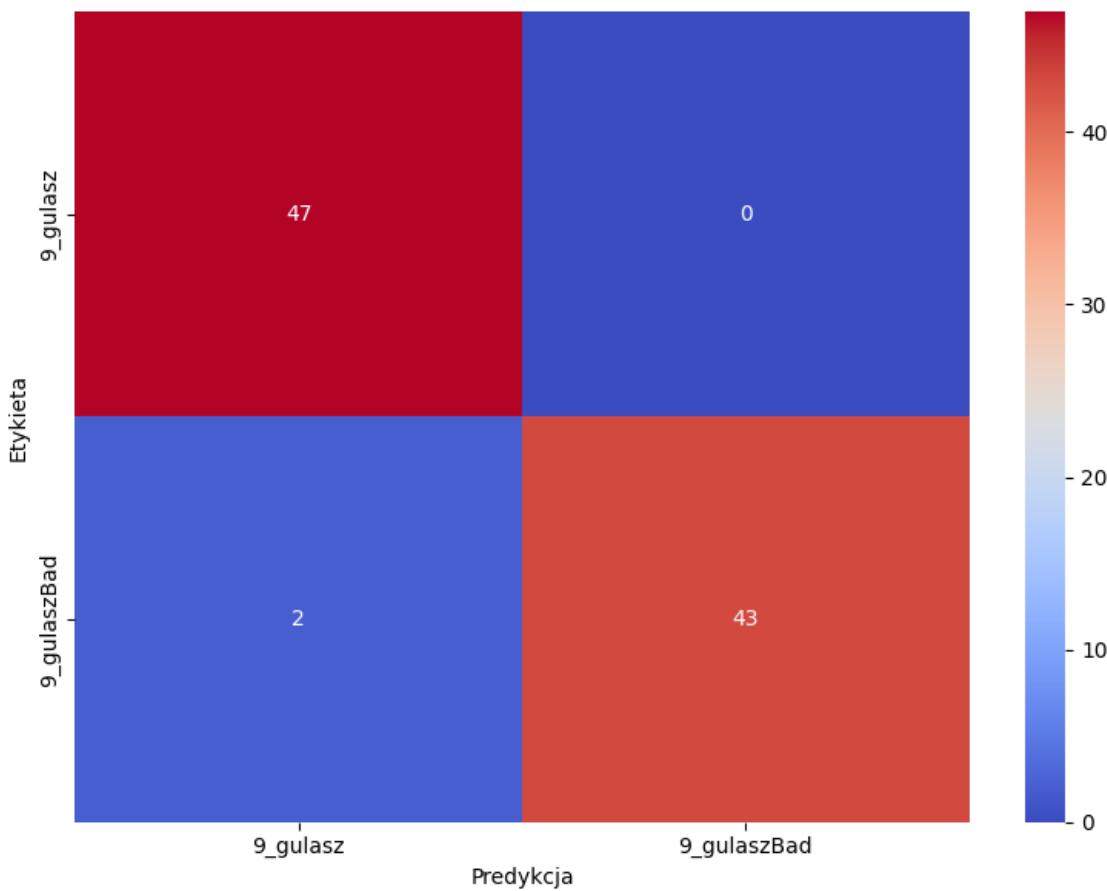
Rys. 6.9. Afisz - Macierz błędów

Macierz błędów przedstawiona na rysunku 6.9 ilustruje wydajność modelu w klasyfikacji wymowy słowa "Afisz". Analiza wyników tej macierzy pokazuje, że mimo ogólnie dobrych rezultatów, model napotkał pewne trudności w rozróżnianiu poprawnej i niepoprawnej wymowy tego słowa:

- Poprawna wymowa (etykieta: 8\_afisz) została poprawnie sklasyfikowana w 45 przypadkach, a 2 przypadki zostały błędnie zaklasyfikowane jako niepoprawna wymowa.
- Niepoprawna wymowa (etykieta: 8\_afiszBad) została poprawnie rozpoznana w 38 przypadkach, jednak 7 przypadków zostało błędnie zaklasyfikowanych jako poprawna wymowa.

Mimo że model napotkał pewne trudności w rozróżnianiu poprawnej i niepoprawnej wymowy słowa "Afisz", jego ogólna dokładność pozostaje stosunkowo wysoka, osiągając **90%**. Wskaźnik ten świadczy o solidnej wydajności modelu, choć istnieje miejsce na dalsze doskonalenie, aby zwiększyć precyzję i zminimalizować błędy w klasyfikacji.

### 6.2.9. Gulasz



Rys. 6.10. Gulasz - Macierz błędów

Macierz błędów przedstawiona na rysunku 6.10 pokazuje skuteczność modelu w klasyfikacji wymowy słowa "Gulasz". Na podstawie tej macierzy można zauważyć, że model osiągnął bardzo wysoką dokładność w rozróżnianiu między poprawną a niepoprawną wymową:

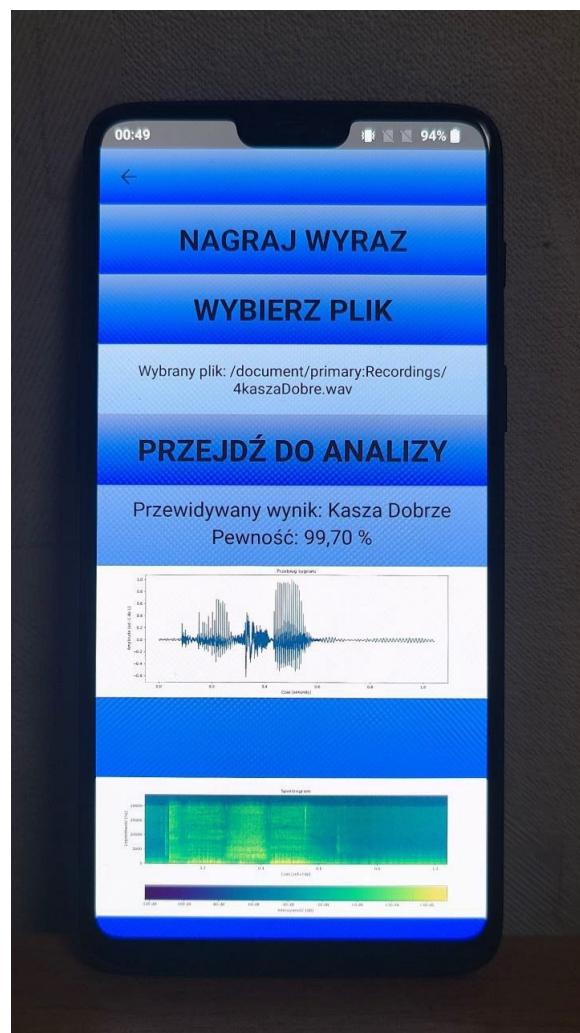
- Poprawna wymowa (etykieta: *9\_gulasz*) została poprawnie sklasyfikowana w 47 przypadkach, bez błędnej klasyfikacji jako niepoprawna wymowa.
- Niepoprawna wymowa (etykieta: *9\_gulaszBad*) została poprawnie rozpoznana w 43 przypadkach, a 2 przypadki zostały błędnie zaklasyfikowane jako poprawna wymowa.

Model osiągnął dokładność na poziomie **98%** w rozpoznawaniu poprawności wymowy słów z zestawu testowego i wykazał doskonałą skuteczność w rozpoznawaniu zarówno poprawnej, jak i niepoprawnej wymowy słowa "Wieszak", co świadczy o bardzo wysokiej precyzji modelu.

## 6.3. Końcowy test na aplikacji

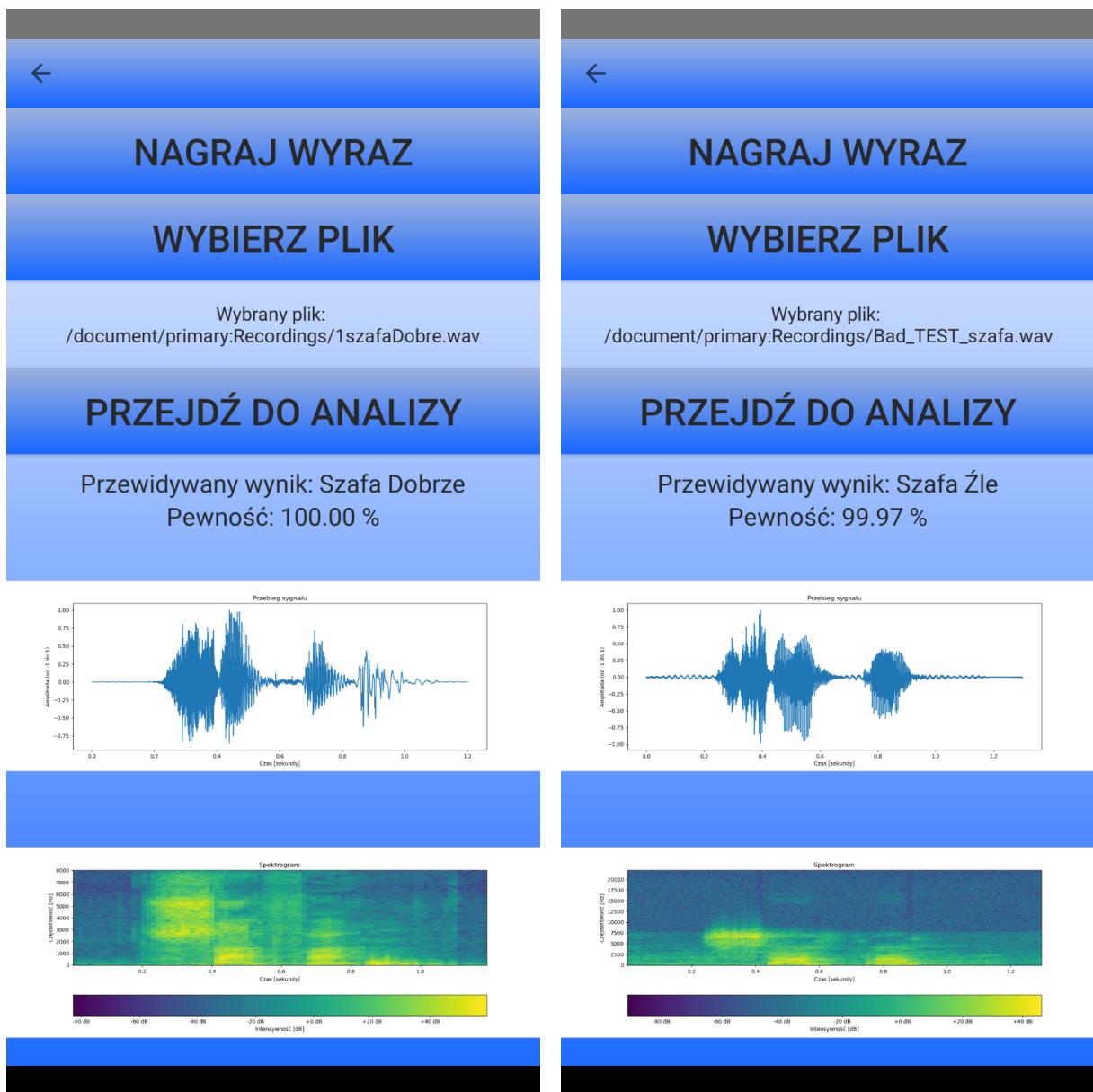
Końcowy test na aplikacji miał na celu ostateczne potwierdzenie poprawności działania modelu, szczególnie w kontekście wyników uzyskanych w poprzednich testach, które były zadowalające. Aby upewnić się, że model działa poprawnie w warunkach rzeczywistych, przeprowadzono analizę na zupełnie nowych nagraniach, które wcześniej nie były używane do trenowania, walidacji ani testowania modelu. Dla tego testu użyto dwóch nagrań: pierwsze nagranie zawierało poprawną wymowę danego słowa, drugie nagranie to zniekształcona wymowa.

Aby sprawdzić działanie aplikacji na urządzeniach mobilnych, projekt został zapisany do formatu APK, a następnie przeniesiony na telefon *OnePlus 6*. Aplikacja została zainstalowana i uruchomiona, co umożliwiło testowanie jej funkcjonalności i wydajności w rzeczywistych warunkach użytkowania. Obecny rozmiar aplikacji wynosi 300 Mb, co obejmuje wszystkie niezbędne komponenty oraz dane potrzebne do działania aplikacji na urządzeniu mobilnym.



Rys. 6.11. Aplikacja uruchomiona na telefonie

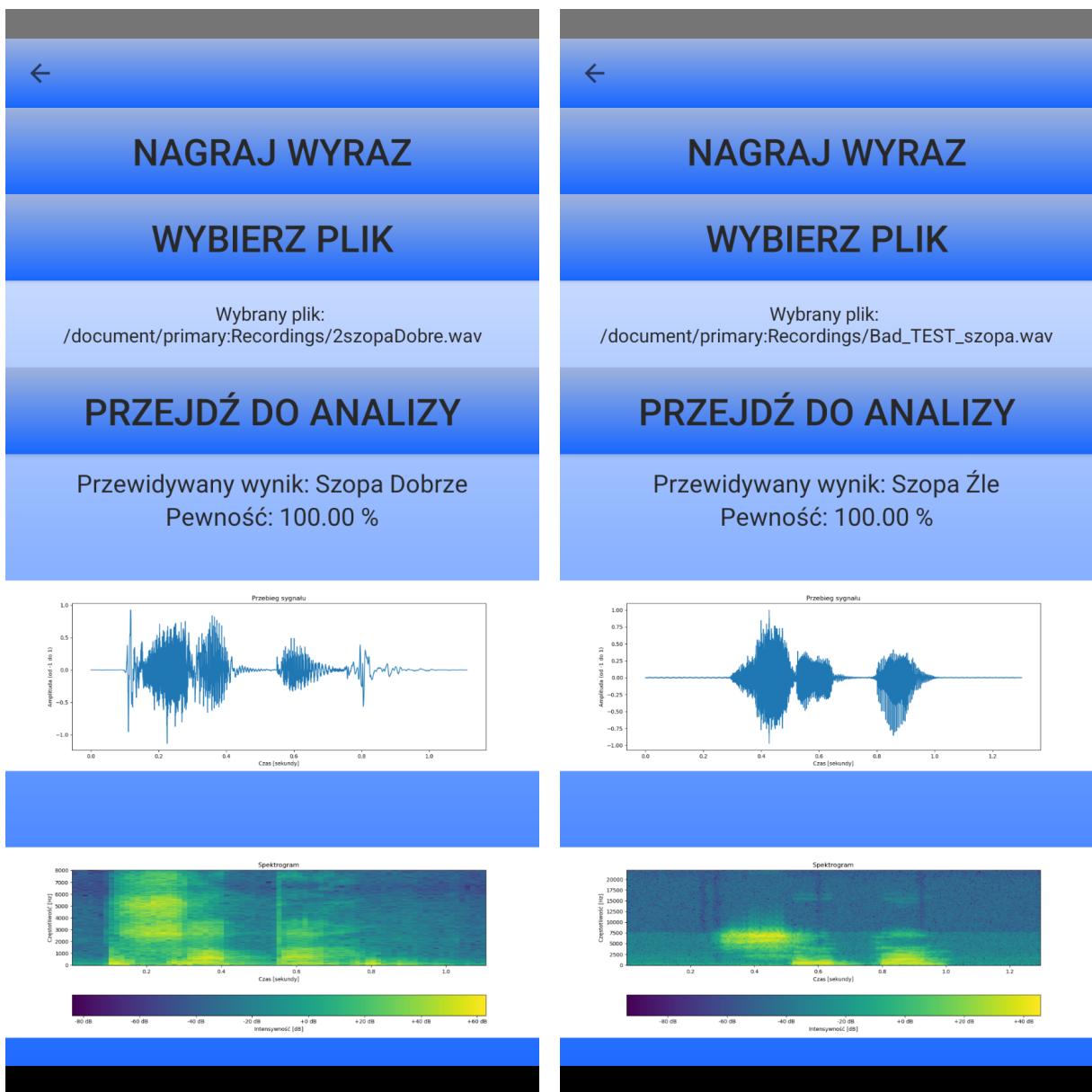
### 6.3.1. Szafa



Rys. 6.12. Wyniki analizy poprawnej (po lewej) i niepoprawnej (po prawej) wymowy słowa "Szafa".

Wyniki tych dwóch analiz potwierdzają, że model jest w stanie skutecznie rozróżniać między poprawną a niepoprawną wymową. Na rysunku 6.12 widzimy, że model poprawnie sklasyfikował nagranie z poprawną wymową jako "Szafa Dobrze" z pewnością 100.00%. oraz prawidłowo rozpoznał zniekształconą wymowę, klasyfikując ją jako "Szafa Źle" z pewnością 99.97%. Te wyniki potwierdzają, że model słowa "Szafa" jest dobrze przygotowany do praktycznego zastosowania w aplikacji *Erkorektor*.

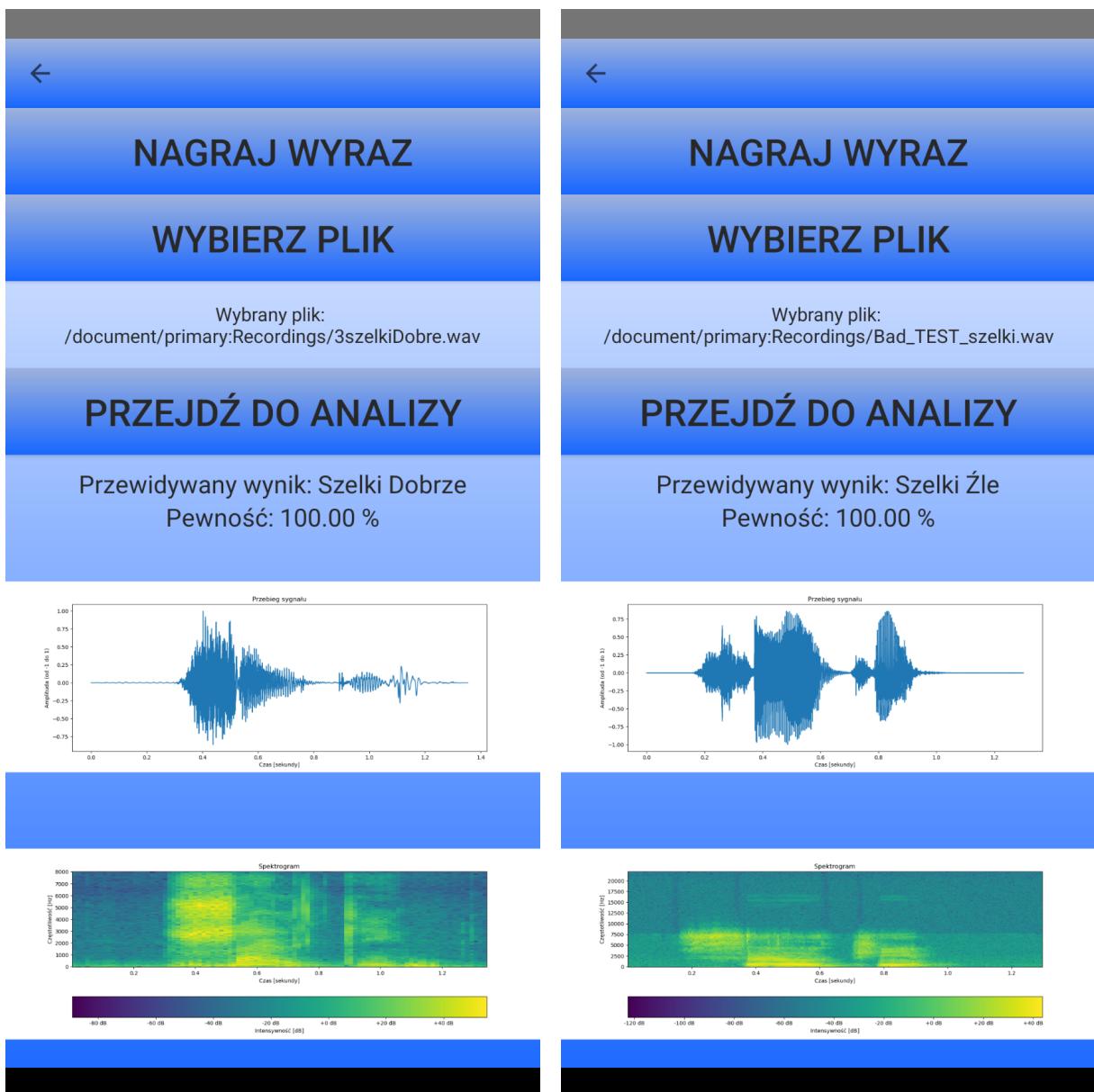
### 6.3.2. Szopa



**Rys. 6.13.** Wyniki analizy poprawnej (po lewej) i niepoprawnej (po prawej) wymowy słowa "Szopa".

Wyniki tych dwóch analiz potwierdzają, że model jest w stanie skutecznie rozróżniać między poprawną a niepoprawną wymową. Na rysunku 6.13 widzimy, że model poprawnie sklasyfikował nagranie z poprawną wymową jako "Szopa Dobrze" z pewnością 100.00% oraz prawidłowo rozpoznał zniekształconą wymowę, klasyfikując ją jako "Szopa Źle" z pewnością 100.00%. Te wyniki potwierdzają, że model słowa "Szopa" jest dobrze przygotowany do praktycznego zastosowania w aplikacji *Erkorektor*.

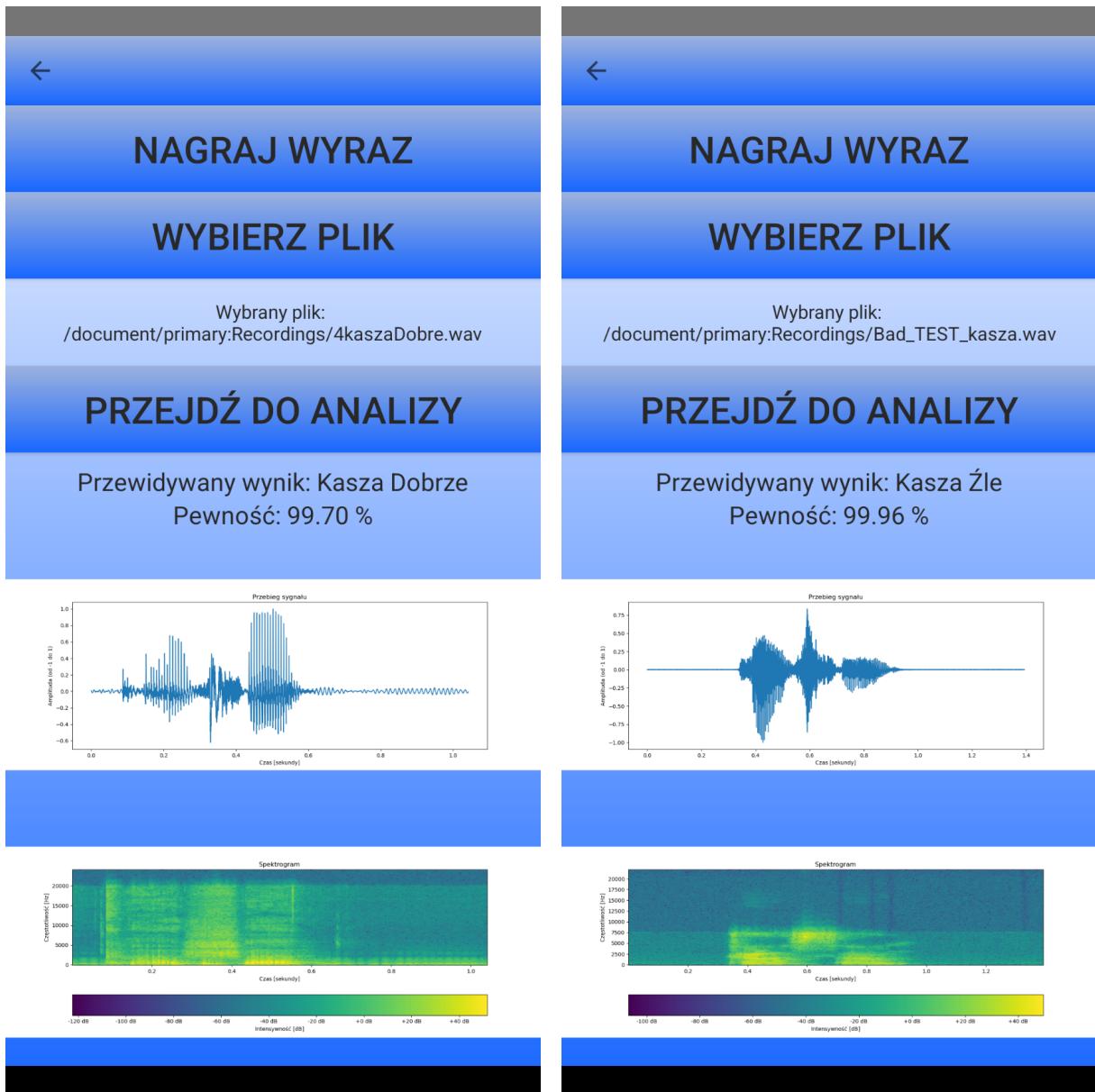
### 6.3.3. Szelki



Rys. 6.14. Wyniki analizy poprawnej (po lewej) i niepoprawnej (po prawej) wymowy słowa "Szelki".

Wyniki tych dwóch analiz potwierdzają, że model jest w stanie skutecznie rozróżniać między poprawną a niepoprawną wymową. Na rysunku 6.14 widzimy, że model poprawnie sklasyfikował nagranie z poprawną wymową jako "Szelki Dobrze" z pewnością 100.00% oraz prawidłowo rozpoznał zniekształconą wymowę, klasyfikując ją jako "Szelki Źle" z pewnością 100.00%. Te wyniki potwierdzają, że model słowa "Szelki" jest dobrze przygotowany do praktycznego zastosowania w aplikacji *Erkorektor*.

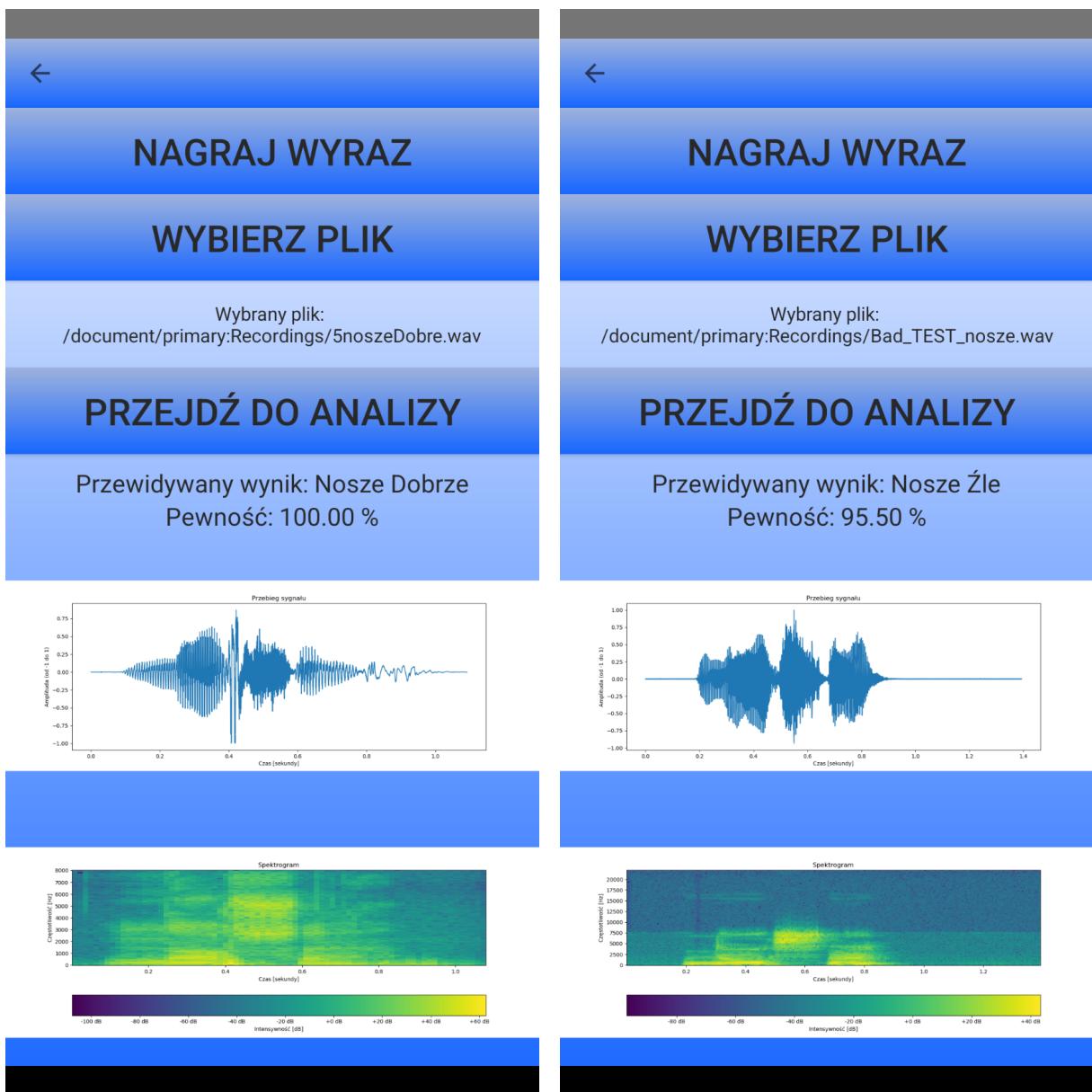
### 6.3.4. Kasza



**Rys. 6.15.** Wyniki analizy poprawnej (po lewej) i niepoprawnej (po prawej) wymowy słowa "Kasza".

Wyniki tych dwóch analiz potwierdzają, że model jest w stanie skutecznie rozróżniać między poprawną a niepoprawną wymową. Na rysunku 6.15 widzimy, że model poprawnie sklasyfikował nagranie z poprawną wymową jako "Kasza Dobrze" z pewnością 99.70% oraz prawidłowo rozpoznał zniekształconą wymowę, klasyfikując ją jako "Kasza Źle" z pewnością 99.96%. Te wyniki potwierdzają, że model słowa "Kasza" jest dobrze przygotowany do praktycznego zastosowania w aplikacji *Erkorektor*.

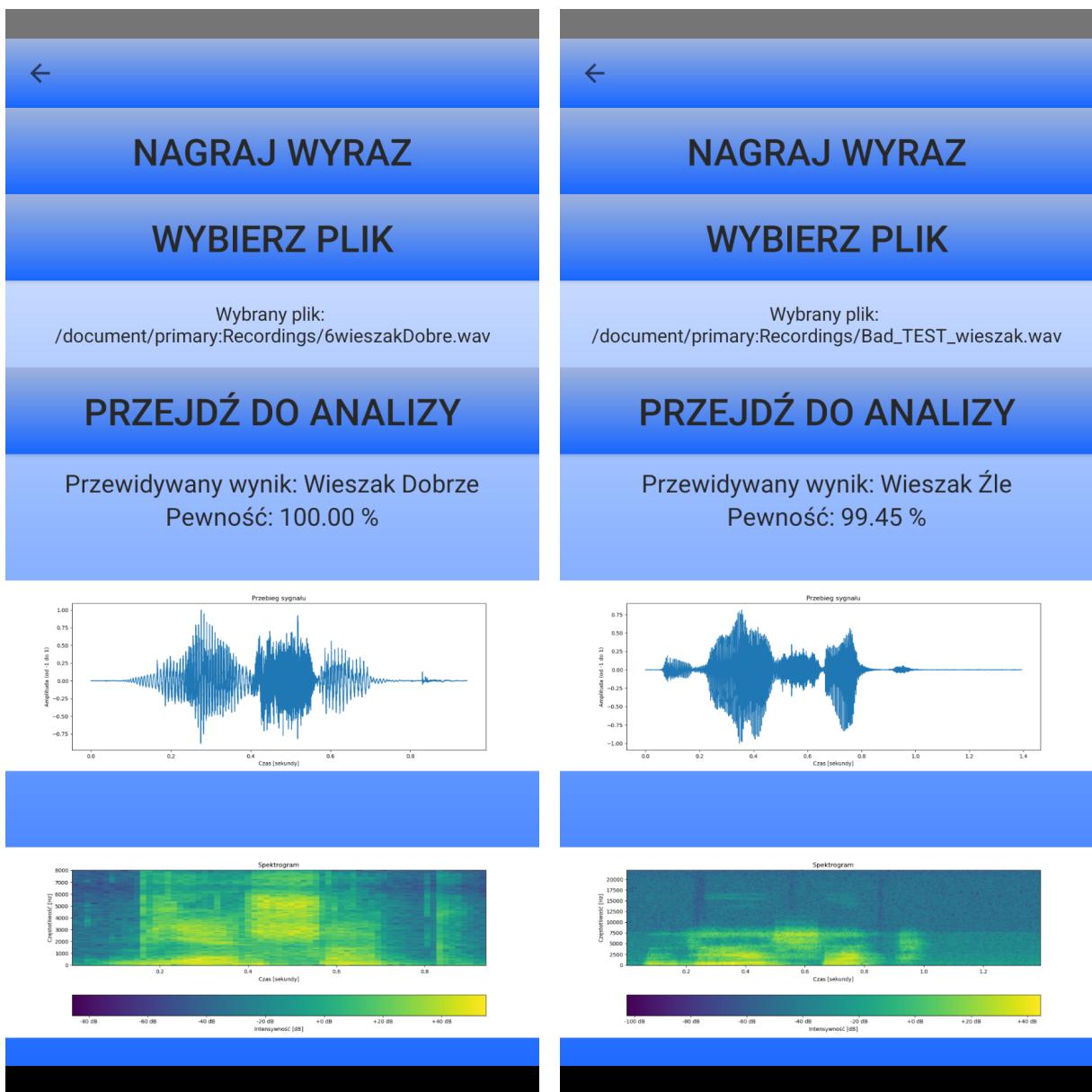
### 6.3.5. Nosze



**Rys. 6.16.** Wyniki analizy poprawnej (po lewej) i niepoprawnej (po prawej) wymowy słowa "Nosze".

Wyniki tych dwóch analiz potwierdzają, że model jest w stanie skutecznie rozróżniać między poprawną a niepoprawną wymową. Na rysunku 6.16 widzimy, że model poprawnie sklasyfikował nagranie z poprawną wymową jako "Nosze Dobrze" z pewnością 100.00% oraz prawidłowo rozpoznał zniekształconą wymowę, klasyfikując ją jako "Nosze Źle" z pewnością 95.50%. Te wyniki potwierdzają, że model słowa "Nosze" jest dobrze przygotowany do praktycznego zastosowania w aplikacji *Erkorektor*.

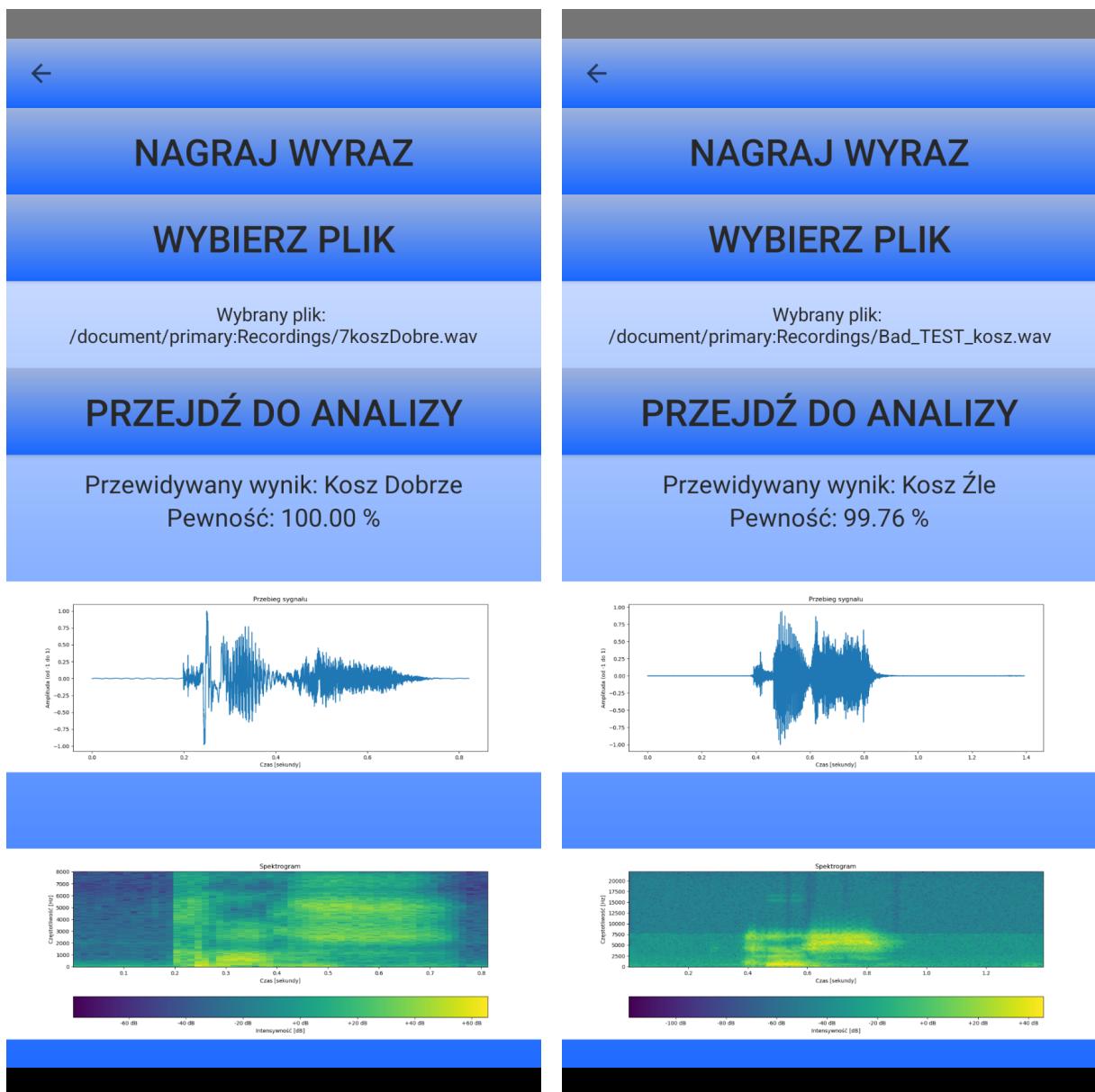
### 6.3.6. Wieszak



Rys. 6.17. Wyniki analizy poprawnej (po lewej) i niepoprawnej (po prawej) wymowy słowa "Wieszak".

Wyniki tych dwóch analiz potwierdzają, że model jest w stanie skutecznie rozróżniać między poprawną a niepoprawną wymową. Na rysunku 6.17 widzimy, że model poprawnie sklasyfikował nagranie z poprawną wymową jako "Wieszak Dobrze" z pewnością 100.00% oraz prawidłowo rozpoznał zniekształconą wymowę, klasyfikując ją jako "Wieszak Źle" z pewnością 99.45%. Te wyniki potwierdzają, że model słowa "Wieszak" jest dobrze przygotowany do praktycznego zastosowania w aplikacji *Erkorektor*.

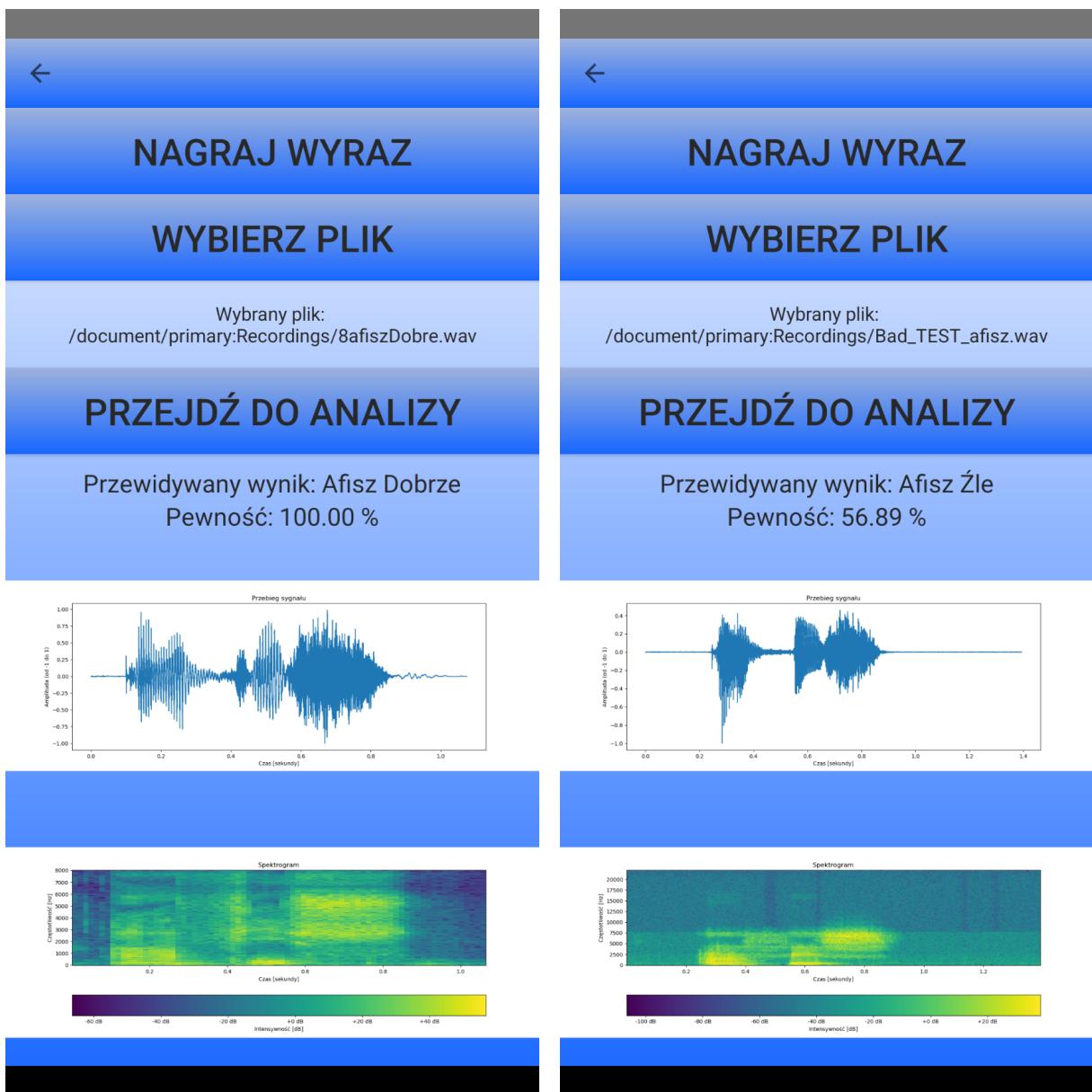
### 6.3.7. Kosz



Rys. 6.18. Wyniki analizy poprawnej (po lewej) i niepoprawnej (po prawej) wymowy słowa "Kosz".

Wyniki tych dwóch analiz potwierdzają, że model jest w stanie skutecznie rozróżniać między poprawną a niepoprawną wymową. Na rysunku 6.18 widzimy, że model poprawnie sklasyfikował nagranie z poprawną wymową jako "Kosz Dobrze" z pewnością 100.00% oraz prawidłowo rozpoznał zniekształconą wymowę, klasyfikując ją jako "Kosz Źle" z pewnością 99.76%. Te wyniki potwierdzają, że model słowa "Kosz" jest dobrze przygotowany do praktycznego zastosowania w aplikacji *Erkorektor*.

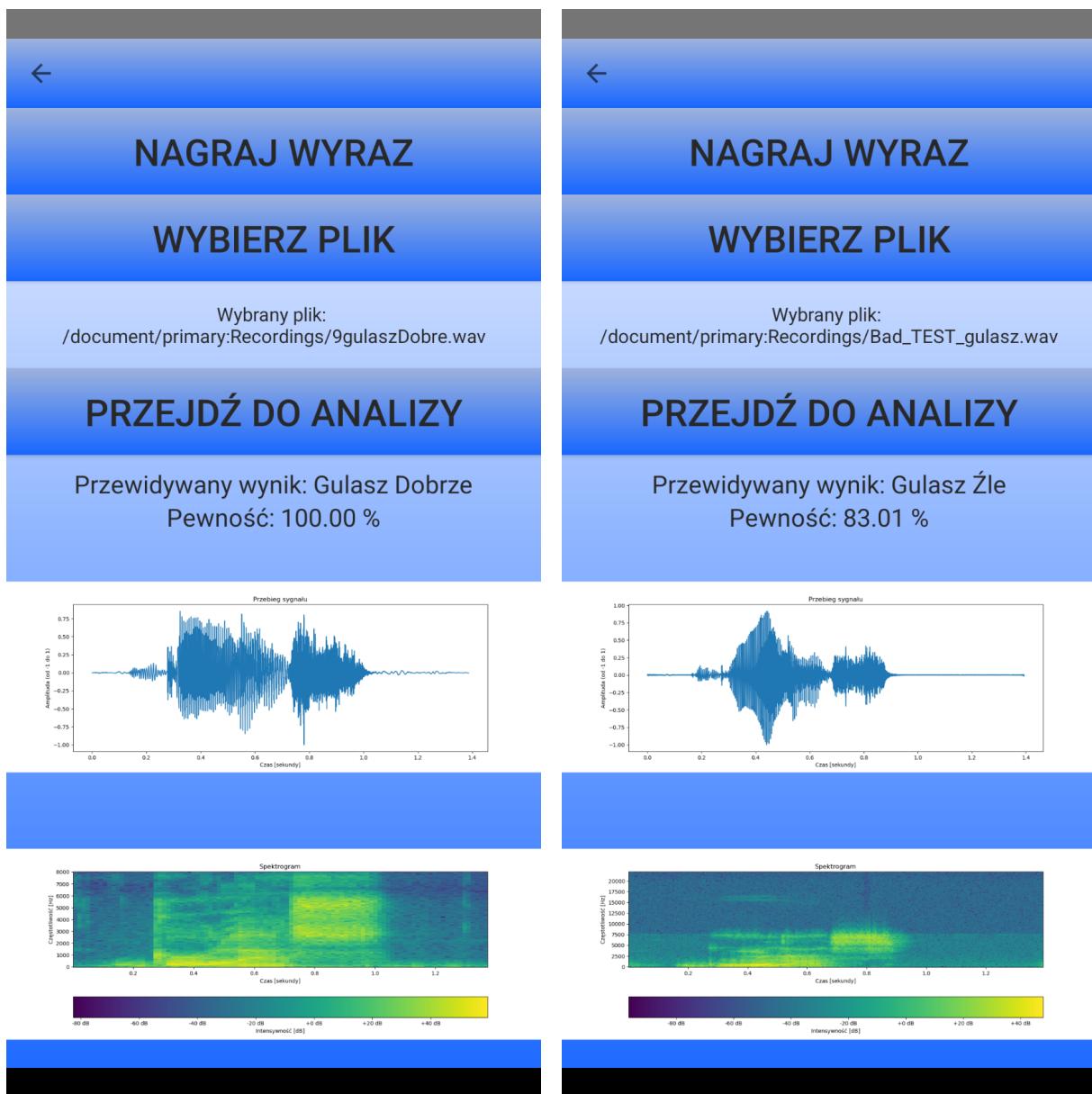
### 6.3.8. Afisz



**Rys. 6.19.** Wyniki analizy poprawnej (po lewej) i niepoprawnej (po prawej) wymowy słowa "Afisz".

Wyniki tych dwóch analiz wskazują, że model jest w stanie odróżniać poprawną wymowę od niepoprawnej, choć z nieco mniejszą pewnością w przypadku bardziej skomplikowanych zniekształceń. Na rysunku 6.19 widzimy, że model bezbłędnie sklasyfikował nagranie z poprawną wymową jako "Afisz Dobrze" z pewnością 100.00%. Jednakże, w przypadku nagrania ze zmodyfikowaną wymową, model poprawnie sklasyfikował je jako "Afisz Źle", choć z niższą pewnością wynoszącą 56.89%. Te wyniki sugerują, że choć model generalnie działa poprawnie, w niektórych bardziej skomplikowanych przypadkach może występować obniżenie pewności klasyfikacji. Mimo to, model słowa "Afisz" wydaje się być dobrze przygotowany do zastosowania w praktyce w aplikacji.

### 6.3.9. Gulasz



**Rys. 6.20.** Wyniki analizy poprawnej (po lewej) i niepoprawnej (po prawej) wymowy słowa "Gulasz".

Wyniki tych dwóch analiz potwierdzają, że model jest w stanie skutecznie rozróżniać między poprawną a niepoprawną wymową. Na rysunku 6.20 widzimy, że model poprawnie sklasyfikował nagranie z poprawną wymową jako "Gulasz Dobrze" z pewnością 100.00% oraz prawidłowo rozpoznał zniekształconą wymowę, klasyfikując ją jako "Gulasz Źle" z pewnością 83.01%. Te wyniki potwierdzają, że model słowa "Gulasz" jest dobrze przygotowany do praktycznego zastosowania w aplikacji *Erkorektor*, chociaż wciąż istnieje pewien margines poprawy w rozpoznawaniu mniej oczywistych zniekształceń wymowy.

## 7. Analiza wyników

W przeprowadzonych testach modelu rozpoznawania wymowy, uzyskane wyniki dla każdego z 9 testowanych słów wskazują na wysoką skuteczność modelu w rozróżnianiu poprawnej i niepoprawnej wymowy. Poniżej przedstawiono krótkie podsumowanie wyników dla każdego słowa:

Słowo	Skuteczność modelu (%)		
	Test dla każdego słowa (6.2.)	Końcowy test na aplikacji (6.3.)	
		Wymowa poprawna	Wymowa niepoprawna
Szafa	100	100.00	99.97
Szopa	98	100.00	100.00
Szelki	99	100.00	100.00
Kasza	97	99.70	99.96
Nosze	97	100.00	95.50
Wieszak	96	100.00	99.45
Kosz	90	100.00	99.76
Afisz	90	100.00	56.89
Gulasz	98	100.00	83.01

**Tabela 7.1.** Wyniki skuteczności modelu dla różnych słów

Ogólnie, model wykazał bardzo wysoką skuteczność w rozpoznawaniu poprawnej i niepoprawnej wymowy, co potwierdza jego przydatność do praktycznego zastosowania w aplikacji *Erkorektor*. Jedynie w kilku przypadkach wystąpiły drobne trudności, które mogą stanowić obszar dalszej optymalizacji. Zauważalnym wzorcem w wynikach jest to, że najniższe wyniki klasyfikacji występowały w przypadku słów, w których kluczowe fonemy znajdowały się w wygłosie, czyli na końcu słowa. Tłumaczy to fakt, że najniższa pewność klasyfikacji w testach na nowych nagraniach również pojawiła się w takich słowach jak "Afisz" i "Gulasz".

---

Jest to zrozumiałe, ponieważ w języku polskim końcowe dźwięki słów mogą być bardziej podatne na zniekształcenia, zarówno w wymowie, jak i w percepji. Dźwięki w wygłosie często są mniej wyraźne, mogą ulegać skróceniu, a także mogą być trudniejsze do rozróżnienia przez model ze względu na ich krótką długość trwania oraz mniejszą energię akustyczną. W konsekwencji rozpoznawanie poprawnej lub niepoprawnej wymowy staje się trudniejsze, co prowadzi do obniżenia pewności klasyfikacji. Z tego powodu, w dalszej optymalizacji modelu szczególną uwagę należy poświęcić treningowi w rozróżnianiu fonemów znajdujących się w wygłosie, aby poprawić jego skuteczność w rozpoznawaniu trudniejszych przypadków.

## 8. Przyszły rozwój aplikacji

Aplikacja *Erkorektor* w obecnej formie wykazuje wysoką skuteczność w rozpoznawaniu i klasyfikacji poprawnej oraz niepoprawnej wymowy słów. Jednakże, aby w pełni wykorzystać potencjał tej technologii oraz dostarczyć użytkownikom jeszcze bardziej zaawansowane narzędzie, można rozważyć kilka potencjalnych kierunków rozwoju.

Jednym z pierwszych kierunków dalszego rozwoju aplikacji jest znaczące powiększenie bazy danych, szczególnie o nagrania od osób z zaburzeniami wymowy. Dotychczasowa baza danych jest solidnym fundamentem, jednak aby model mógł być jeszcze bardziej precyzyjny i wszechstronny, warto by było zebrać większą ilość danych, które uwzględniają różnorodność wymowy wśród osób z różnymi zaburzeniami mowy. Szczególnie cenne będą nagrania od osób sepleniących, jakających się lub mających trudności z artykulacją poszczególnych głosek, ponieważ stanowią one wartościowy materiał do dalszego trenowania modelu.

Skoro aplikacja z powodzeniem rozpoznaje zniekształcenia związane z konkretną głoską, naturalnym kolejnym krokiem mogłoby być rozszerzenie możliwości aplikacji na inne głoski. Rozbudowa modelu o funkcje rozpoznawania zniekształceń w wymawianiu różnych głosek pozwoli na jeszcze szersze zastosowanie aplikacji w terapii logopedycznej oraz edukacji fonetycznej.

Innym potencjalnym kierunkiem rozwoju jest opracowanie modelu, który nie tylko stwierdza, że dana wymowa jest niepoprawna, ale również dokładnie określa rodzaj zniekształceń. Przykładowo, model mógłby rozpoznawać specyficzne typy seplenienia, takie jak seplenie boczne, dorsalne, czy międzyzębowe. Dzięki takiej funkcjonalności, aplikacja mogłaby dostarczać użytkownikom bardziej szczegółowych informacji zwrotnych, co ułatwiłoby skuteczniej szą korektę wymowy.

Kolejnym innowacyjnym pomysłem jest dodanie funkcji ćwiczeń i monitorowania pracy węzła języka. Wiele osób z zaburzeniami mowy musi wykonywać ćwiczenia, które pomagają rozciągnąć węzidełko języka, co jest kluczowe dla prawidłowej artykulacji niektórych dźwięków. Aplikacja mogłaby oferować specjalne ćwiczenia dostosowane do tego celu oraz śledzić postępy użytkownika w czasie, co umożliwiłoby bardziej zindywidualizowaną terapię mowy.

Kolejną potencjalną funkcją jest wprowadzenie ćwiczeń w formie zabaw skierowanych do najmłodszych użytkowników. Dzięki interaktywnym i angażującym aktywnościom, dzieci mogłyby uczyć się poprawnej wymowy w sposób, który jest dla nich atrakcyjny i motywujący.

Zabawy mogłyby być dostosowane do poziomu trudności i potrzeb każdego dziecka, co zwiększyłoby skuteczność nauki.

Dalszy rozwój aplikacji może również obejmować dodanie funkcji analizy postępów użytkowników. Dzięki tej funkcji aplikacja mogłaby śledzić wyniki użytkownika na przestrzeni czasu, identyfikować obszary, które wymagają poprawy, tworzyć raporty oraz dostarczać spersonalizowane sugestie dotyczące ćwiczeń. Taka funkcjonalność umożliwiłaby nie tylko bieżące monitorowanie postępów, ale także motywować użytkowników poprzez wizualizację ich postępów i osiągnięć w nauce poprawnej wymowy. Analiza postępów mogłaby być szczególnie użyteczna w terapii logopedycznej, gdzie regularne śledzenie zmian w wymowie jest kluczowe dla skutecznej interwencji.

Podsumowując, rozszerzenie aplikacji *Erkorektor* o nowe funkcje, takie jak rozbudowana baza danych, analiza różnych głosek, rozpoznawanie specyficznych zniekształceń, ćwiczenia i zabawy dla dzieci, ćwiczenia związane z pracą wędzidełka języka oraz analiza postępów, pozwoli na jeszcze skuteczniejsze wspieranie użytkowników w procesie nauki poprawnej wymowy i terapii logopedycznej.



Rys. 8.1. Przyszły plan rozwoju aplikacji

## 9. Podsumowanie

Niniejsza praca magisterska przedstawia proces projektowania, implementacji oraz testowania aplikacji *Erkorektor*, mającej na celu wspieranie użytkowników w nauce poprawnej wymowy poprzez identyfikowanie i korygowanie błędów artykulacyjnych.

Praca rozpoczęła się od analizy problematyki zaburzeń mowy, ze szczególnym uwzględnieniem trudności w wymowie głosek, takich jak "sz". Na tej podstawie opracowano model rozpoznawania mowy, oparty na konwolucyjnych sieciach neuronowych (CNN). Model został przeszkolony na bazie danych zawierającej zarówno poprawne, jak i zniekształcone wymowy wybranych słów.

Proces trenowania modelu obejmował kilka etapów, od wstępnych testów na ograniczonej liczbie słów, po testy uwzględniające nagrania zniekształconych wymówień. Uzyskane wyniki były satysfakcjonujące i potwierdziły wysoką skuteczność modelu w rozróżnianiu poprawnych oraz niepoprawnych wymówień, co zostało dodatkowo zweryfikowane w końcowych testach aplikacji.

Następnie skupiono się na zastosowaniach modelu w aplikacji mobilnej *Erkorektor*. Aplikacja została zaprojektowana w sposób zapewniający intuicyjność użytkowania, co ułatwia korzystanie z narzędzia wspomagającego naukę wymowy. Ostateczne testy aplikacji na nowych nagraniach, które nie były wcześniej używane w procesie trenowania, potwierdziły skuteczność modelu i gotowość aplikacji do praktycznego zastosowania.

Praca kończy się propozycjami rozwoju aplikacji, takimi jak rozbudowa bazy danych o nagrania osób z różnorodnymi problemami w wymowie, rozszerzenie modelu na inne głoski oraz wprowadzenie funkcji rozpoznawania specyficznych typów zniekształceń. Zasadne wydaje się także dodanie funkcji ćwiczeń w formie interaktywnych zabaw dla dzieci oraz monitorowania pracy wędzidełka języka. Wprowadzenie funkcji analizy postępów i generowania raportów pozwoliłoby na lepsze śledzenie efektów terapii i dostosowanie ćwiczeń do indywidualnych potrzeb użytkowników.

Aplikacja *Erkorektor* stanowi istotny krok w rozwoju narzędzi wspierających terapię logopedyczną, oferując realne korzyści zarówno dzieciom, jak i dorosłym z zaburzeniami mowy. Osobiście zamierzam regularnie korzystać z aplikacji, aby utrwały i jeszcze bardziej udoskonalić swoją wymowę.

## Bibliografia

- [1] Arias-Vergara, T., Klumpp, P., Vasquez-Correa, J. C., Nöth, E., Orozco-Arroyave, J. R., & Schuster, M. *Multi-channel spectrograms for speech processing applications using deep learning methods*, 2020.
- [2] Mehrish, A., Majumder, N., Bhardwaj, R., Mihalcea, R., & Poria, S. *A review of deep learning techniques for speech processing*, 2023.
- [3] Tyagi, S., & Szénási, S. *Semantic speech analysis using machine learning and deep learning techniques: A comprehensive review*, 2023.
- [4] Huang, Y., & Huang, Y. *Detection of Mispronunciation in Non-native Speech Using Acoustic Model and Convolutional Recurrent Neural Networks*, J. Phys.: Conf. Ser., vol. 1952 (2021), 032043.
- [5] Nandal, P., Kadian, Y., Upadhyay, S., & Mudgal, B. P. *Pronunciation Accuracy Calculator using Machine Learning*, Proc. 5th Int. Conf. on Computing Methodologies and Communication (ICCMC 2021).
- [6] Mutqiyyah, R., & Muhammad, A. F. *Developing Mobile App of English Pronunciation Test Using Android Studio*, 2016 International Electronics Symposium (IES).
- [7] Ziółko, B., & Ziółko, M. *Przetwarzanie mowy*. Wydawnictwo Naukowe PWN, Warszawa, 2011. ISBN 978-83-7464-441-9.
- [8] Kasprzak, W. *Rozpoznawanie obrazów i sygnałów mowy*. Oficyna Wydawnicza Politechniki Warszawskiej, 2009. ISBN 978-83-7207-769-1.
- [9] Jurafsky, D., & Martin, J. H. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 2024.
- [10] Goodfellow, I., Bengio, Y., & Courville, A. *Deep Learning*, MIT Press, 2016.
- [11] Węsierska, K., & Podstolec, A. *W świecie logopedii. Tom 2: Studia przypadków*. Wydawnictwo Uniwersytetu Śląskiego, Katowice, 2013.
- [12] Masrom, M., Noor, N. M., Kamaruddin, A. I., & Aziz, M. A. A. *Speech Therapy Mobile Applications for People with Aphasia: PRISMA Review and Features Analysis*. 2021 IEEE National Biomedical Engineering Conference (NBEC)
- [13] Kanimozhiselvi, C. S., & Santhiya, S. *Communication Disorder Identification from Recorded Speech using Machine Learning Assisted Mobile Application*. 2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV)
- [14] Yang, X., Yu, H., & Jia, L. *Speech Recognition of Command Words Based on Convolutional Neural Network*. 2020 International Conference on Computer Information and Big Data Applications (CIBDA)
- [15] American Speech-Language-Hearing Association (ASHA). *Scope of Practice in Speech-Language Pathology*, 2016.
- [16] *Otsimo Speech Therapy*. Aplikacja na Google Play
- [17] *Stamurai Stuttering Therapy*. Aplikacja na Google Play

- [18] Tactus Therapy Solutions. *Apraxia Therapy Lite*, Aplikacja na Google Play
- [19] Android Developers. <https://developer.android.com/?hl=pl>.
- [20] Moskala, M., & Wojda, I. *Android Development with Kotlin*. Packt Publishing, 2017. ISBN 978-1-78712-368-7.
- [21] Kotlin Documentation. <https://kotlinlang.org/docs/>.
- [22] Python Software Foundation. <https://www.python.org/>.
- [23] Śliwerski, W. (Red.). *Materiał wyrazowo-obrazkowy do utrwalania poprawnej wymowy głosek sz, rz, cz, dz*, Oficyna Wydawnicza Impuls, (Wyd. IV, 2011), ISBN 978-83-7587-790-8.
- [24] TensorFlow Documentation. <https://www.tensorflow.org/?hl=pl>.
- [25] Mattmann, C. *Machine Learning with TensorFlow, Second Edition*. Manning, 2020.

## 10. Dodatek A

Ze względu na obszerność kodu źródłowego stworzonego w ramach tego projektu, w niniejszym dodatku zamieszczono jedynie wybrane jego fragmenty. W projekcie utworzono łącznie 9 plików *XML* odpowiedzialnych za definiowanie wyglądu interfejsu użytkownika aplikacji oraz 19 klas w języku *Kotlin*, które odpowiadają za logikę działania aplikacji. Ponadto, projekt zawiera również pliki konfiguracyjne oraz skrypty w języku *Python*, wykorzystywane do trenowania modelu rozpoznawania mowy. Pełna dokumentacja kodu, obejmująca wszystkie pliki *XML*, klasy *Kotlin* oraz skrypty *Python*, została dołączona jako załącznik do pracy.

```
1 <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
2     xmlns:app="http://schemas.android.com/apk/res-auto"
3     xmlns:tools="http://schemas.android.com/tools"
4     android:layout_width="match_parent"
5     android:layout_height="match_parent"
6     tools:context=".RecordingActivity">
7
8     <androidx.appcompat.widget.Toolbar
9         android:id="@+id/myToolbar"
10        android:layout_width="match_parent"
11        android:layout_height="?attr/actionBarSize"
12        android:background="@drawable/gradient"
13        tools:ignore="MissingConstraints"></androidx.appcompat.widget.
14             Toolbar>
15
16     <TextView
17         android:id="@+id/timer"
18         android:layout_width="wrap_content"
19         android:layout_height="wrap_content"
20         android:layout_marginTop="32dp"
21         android:layout_marginBottom="16dp"
22         android:text="00:00:00"
23         android:textSize="68sp"
24         android:textStyle="bold"
25         app:layout_constraintEnd_toEndOf="parent"
26         app:layout_constraintStart_toStartOf="parent"
27         app:layout_constraintTop_toBottomOf="@+id/myToolbar" />
28
29     <EditText
30         android:id="@+id/filenameInput"
```

```

28     android:layout_width="0dp"
29     android:layout_height="wrap_content"
30     android:layout_marginStart="32dp"
31     android:layout_marginTop="16dp"
32     android:layout_marginEnd="32dp"
33     android:background="@drawable/edit_text_background"
34     android:hint="Enter filename"
35     android:imeOptions="actionDone"
36     android:inputType="text"
37     android:padding="16dp"
38     app:layout_constraintEnd_toEndOf="parent"
39     app:layout_constraintStart_toStartOf="parent"
40     app:layout_constraintTop_toBottomOf="@+id/timer" />
41 <LinearLayout
42     android:layout_width="match_parent"
43     android:layout_height="wrap_content"
44     android:layout_marginBottom="80dp"
45     android:gravity="center"
46     android:orientation="horizontal"
47     app:layout_constraintBottom_toBottomOf="parent">
48
49     <ImageButton
50         android:id="@+id/deleteButton"
51         android:layout_width="60dp"
52         android:layout_height="60dp"
53         android:background="@drawable/ic_record"
54         android:src="@drawable/clear_button_off" />
55
56     <ImageButton
57         android:id="@+id/recordButton"
58         android:layout_width="90dp"
59         android:layout_height="90dp"
60         android:layout_marginStart="30dp"
61         android:layout_marginEnd="30dp"
62         android:background="@drawable/ic_record"
63         android:src="@drawable/record_button" />
64
65     <ImageButton
66         android:id="@+id/saveButton"
67         android:layout_width="60dp"
68         android:layout_height="60dp"
69         android:background="@drawable/ic_record"
70         android:src="@drawable/list_button" />
71 </LinearLayout>
72 </androidx.constraintlayout.widget.ConstraintLayout>

```

**Listing 10.1.** activity\_recording.xml

```

1 package com.example.myapplication
2
3 import android.app.Activity
4 import android.content.Context
5 import android.content.Intent
6 import android.graphics.BitmapFactory
7 import android.net.Uri
8 import android.os.Bundle
9 import android.util.Log
10 import android.widget.Button
11 import android.widget.TextView
12 import android.widget.Toast
13 import androidx.appcompat.app.AppCompatActivity
14 import androidx.appcompat.widget.Toolbar
15 import com.chaquo.python.Python
16 import com.chaquo.python.android.AndroidPlatform
17 import com.davemorrissey.labs.subscaleview.ImageSource
18 import com.davemorrissey.labs.subscaleview.SubsamplingScaleImageView
19 import java.io.File
20 import java.io.FileNotFoundException
21 import java.io.FileOutputStream
22 import java.nio.ByteBuffer
23 import java.nio.ByteOrder
24 import org.tensorflow.lite.DataType
25 import org.tensorflow.lite.support.tensorbuffer.TensorBuffer
26 import kotlin.math.exp
27
28 class SpeechAnalizer : AppCompatActivity() {
29
30     private lateinit var fileSelectedText: TextView
31     private lateinit var resultText: TextView
32     private lateinit var confidenceText: TextView
33     private lateinit var signalImageView: SubsamplingScaleImageView
34     private lateinit var spectrogramImageView: SubsamplingScaleImageView
35
36     private val LABEL_SZAFA = listOf("Szafa Dobrze", "Szafa Źle")
37     private val LABEL_SZOPA = listOf("Szopa Dobrze", "Szopa Źle")
38     private val LABEL_SZELKI = listOf("Szelki Dobrze", "Szelki Źle")
39     private val LABEL_KASZA = listOf("Kasza Dobrze", "Kasza Źle")
40     private val LABEL_NOSZE = listOf("Nosze Dobrze", "Nosze Źle")
41     private val LABEL_WIESZAK = listOf("Wieszak Dobrze", "Wieszak Źle")
42     private val LABEL_KOSZ = listOf("Kosz Dobrze", "Kosz Źle")
43     private val LABEL_AFISZ = listOf("Afisz Dobrze", "Afisz Źle")
44     private val LABEL_GULASZ = listOf("Gulasz Dobrze", "Gulasz Źle")
45
46     private val FILE_TYPE_AUDIO = "audio/*"

```

```

47     private val FILE_SELECTED_PREFIX = "Wybrany plik:"
48     private val PYTHON_MODULE_NAME = "model_skrypt"
49     private val PYTHON_ANALYZE_METHOD = "analyze"
50     private val ERROR_MESSAGE_ANALYSIS = "Błąd podczas analizy pliku"
51     private val SUCCESS_MESSAGE_MODEL = "Model załadowany poprawnie"
52     private val UNKNOWN_LABEL = "Nieznana etykieta"
53     private val RESULT_PREFIX = "Przewidywany wynik"
54     private val CONFIDENCE_PREFIX = "Pewność"
55     private val FILE_NOT_FOUND_ERROR = "Nie znaleziono pliku"
56     private val DEFAULT_AUDIO_FILE_NAME = "input_audio.wav"

57
58     override fun onCreate(savedInstanceState: Bundle?) {
59         super.onCreate(savedInstanceState)
60         setContentView(R.layout.activity_analizer)
61
62         val toolbar: Toolbar = findViewById(R.id.myToolbar)
63         setSupportActionBar(toolbar)
64         supportActionBar?.setDisplayHomeAsUpEnabled(true)
65
66         if (!Python.isStarted()) {
67             Python.start(AndroidPlatform(this))
68         }
69
70         fileSelectedText = findViewById(R.id.selectedFile)
71         resultText = findViewById(R.id.resultText)
72         confidenceText = findViewById(R.id.confidenceText)
73         signalImageView = findViewById(R.id.signalImageView)
74         spectrogramImageView = findViewById(R.id.spectrogramImageView)
75
76         findViewById<Button>(R.id.goToRecordingButton).setOnClickListener {
77             val intent = Intent(this, RecordingActivity::class.java)
78             startActivity(intent)
79         }
80
81         findViewById<Button>(R.id.selectFileButton).setOnClickListener {
82             selectFile()
83         }
84
85         findViewById<Button>(R.id.analyzeButton).setOnClickListener {
86             val selectedFileUri = fileSelectedText.tag as? Uri
87             selectedFileUri?.let {
88                 startAnalysis(it)
89             }
90         }
91     }
92
93     private fun selectFile() {

```

```

94     val intent = Intent(Intent.ACTION_GET_CONTENT)
95     intent.type = FILE_TYPE_AUDIO
96     startActivityForResult(intent, REQUEST_CODE_SELECT_FILE)
97 }
98
99     override fun onActivityResult(requestCode: Int, resultCode: Int, data:
100     Intent?) {
101         super.onActivityResult(requestCode, resultCode, data)
102         if (requestCode == REQUEST_CODE_SELECT_FILE && resultCode ==
103             Activity.RESULT_OK) {
104             val selectedFileUri = data?.data
105             selectedFileUri?.let {
106                 fileSelectedText.text = "$FILE_SELECTED_PREFIX ${it.path}"
107                 fileSelectedText.tag = it
108                 fileSelectedText.visibility = TextView.VISIBLE
109             }
110         }
111     }
112
113     fun loadSpectrogramValuesFromFile(filePath: String, width: Int, height:
114     Int): ByteBuffer {
115         val file = File(filePath)
116         val valuesString = file.readText()
117
118         val valuesList = valuesString
119             .lines()
120             .filter { it.isNotBlank() }
121             .mapNotNull {
122                 try {
123                     it.toFloat()
124                 } catch (e: NumberFormatException) {
125                     println("Nieprawidłowa wartość: $it")
126                     null
127                 }
128             }
129
130         if (valuesList.size != width * height) {
131             throw IllegalArgumentException("Liczba wartości w pliku nie
132                 zgadza się z wymiarami spektrogramu")
133         }
134
135         val byteBuffer = ByteBuffer.allocateDirect(width * height * 4)
136         byteBuffer.order(ByteOrder.nativeOrder())
137
138         valuesList.forEach {
139             byteBuffer.putFloat(it)
140         }

```

```

137
138     byteBuffer.rewind()
139
140     return byteBuffer
141 }
142
143 private fun startAnalysis(fileUri: Uri) {
144     try {
145         val filePath = getFilePathFromUri(fileUri)
146
147         val python = Python.getInstance()
148         val pythonFile = python.getModule(PYTHON_MODULE_NAME)
149         val result = pythonFile.callAttr(PYTHON_ANALYZE_METHOD,
150             filePath)
151
152         val resultList = result.asList()
153         if (resultList.size < 3) {
154             Log.e("SpeechAnalizator", "Unexpected result size: ${resultList.size}. Expected at least 3.")
155             showToast(ERROR_MESSAGE_ANALYSIS)
156             return
157         }
158
159         val waveformPath = resultList[0].toString()
160         val spectrogramPath = resultList[1].toString()
161         val spectrogramModel = resultList[2].toString()
162
163         val waveformBitmap = BitmapFactory.decodeFile(waveformPath)
164         val spectrogramBitmap = BitmapFactory.decodeFile(
165             spectrogramPath)
166         signalImageView.setImageBitmap(ImageSource.bitmap(waveformBitmap))
167         spectrogramImageView.setImageBitmap(ImageSource.bitmap(
168             spectrogramBitmap))
169
170         val word = intent.getStringExtra("word") ?: ""
171
172         val labelMap = mapOf(
173             "szafa" to LABEL_SZAFA,
174             "szopa" to LABEL_SZOPA,
175             "szelki" to LABEL_SZELKI,
176             "kasza" to LABEL_KASZA,
177             "nosze" to LABEL_NOSZE,
178             "wieszak" to LABEL_WIESZAK,
179             "kosz" to LABEL_KOSZ,
180             "afisz" to LABEL_AFISZ,
181             "gulasz" to LABEL_GULASZ
182         )

```

```

180
181     val classLabels = labelMap[word.lowercase()] ?: throw
182         IllegalStateException("Unknown word: $word")
183
184     val formattedWord = formatWordForModelClass(word)
185     val modelClassName = "com.example.myapplication.ml.
186         Model$formattedWord"
187     val modelClass = Class.forName(modelClassName)
188     val modelInstance = modelClass.getMethod("newInstance", Context
189         ::class.java).invoke(null, applicationContext)
190
191     val byteBuffer = loadSpectrogramValuesFromFile(spectrogramModel
192         , 124, 129)
193
194     val inputFeature0 = TensorBuffer.createFixedSize(intArrayOf(1,
195         124, 129, 1), DataType.FLOAT32)
196     inputFeature0.loadBuffer(byteBuffer)
197
198     val processMethod = modelClass.getMethod("process",
199         TensorBuffer::class.java)
200     val outputs = processMethod.invoke(modelInstance, inputFeature0
201         )
202
203     val outputTensorBufferMethod = outputs.javaClass.getMethod("getOutputFeature0AsTensorBuffer")
204     val outputFeature0 = outputTensorBufferMethod.invoke(outputs)
205         as TensorBuffer
206
207     modelClass.getMethod("close").invoke(modelInstance)
208     showToast(SUCCESS_MESSAGE_MODEL)
209
210     val outputArray = outputFeature0.floatArray
211     val probabilities = softmax(outputArray)
212
213     val predictedIndex = probabilities.indices.maxByOrNull {
214         probabilities[it] } ?: -1
215     val predictedLabel = classLabels.getOrElse(predictedIndex) {
216         UNKNOWN_LABEL }
217     val confidence = probabilities.getOrElse(predictedIndex) { 0.0f
218         } * 100
219
220     val formattedConfidence = String.format("%.2f", confidence)
221
222     resultText.text = "$RESULT_PREFIX: $predictedLabel"
223     confidenceText.text = "$CONFIDENCE_PREFIX: $formattedConfidence
224         %"
225
226     } catch (e: Exception) {

```

```

214         Log.e("SpeechAnalizator", "Error during analysis", e)
215         showToast("$ERROR_MESSAGE_ANALYSIS: ${e.message}")
216     }
217 }
218
219 fun softmax(logits: FloatArray): FloatArray {
220     val maxLogit = logits.maxOrNull() ?: 0.0f
221     val expValues = logits.map { exp(it - maxLogit) }
222     val sumExpValues = expValues.sum()
223     return expValues.map { it / sumExpValues }.toFloatArray()
224 }
225
226 fun formatWordForModelClass(word: String?): String {
227     return word?.lowercase()?.replaceFirstChar { if (it.isLowerCase())
228         it.titlecase() else it.toString() } ?: ""
229 }
230
231 private fun getFilePathFromUri(uri: Uri): String {
232     val inputStream = contentResolver.openInputStream(uri) ?: throw
233         FileNotFoundException("$FILE_NOT_FOUND_ERROR: $uri")
234     val file = File(filesDir, DEFAULT_AUDIO_FILE_NAME)
235     FileOutputStream(file).use { outputStream ->
236         inputStream.copyTo(outputStream)
237     }
238     return file.absolutePath
239 }
240
241 private fun showToast(message: String) {
242     Toast.makeText(this, message, Toast.LENGTH_SHORT).show()
243 }
244
245 companion object {
246     private const val REQUEST_CODE_SELECT_FILE = 1001
247 }
248 }
```

**Listing 10.2.** SpeechAnalizator.kt

```
1 plugins {
2     id("com.android.application")
3     id("org.jetbrains.kotlin.android")
4     id("kotlin-android")
5     id("com.chaquo.python")
6 }
7
8 android {
9     namespace = "com.example.myapplication"
10    compileSdk = 34
11
12    defaultConfig {
13        applicationId = "com.example.myapplication"
14        minSdk = 24
15        targetSdk = 34
16        versionCode = 1
17        versionName = "1.0"
18
19        testInstrumentationRunner = "androidx.test.runner.
20             AndroidJUnitRunner"
21
22        ndk {
23            abiFilters += "arm64-v8a"
24            abiFilters += "x86_64"
25        }
26
27        buildTypes {
28            release {
29                isMinifyEnabled = false
30                proguardFiles(
31                    getDefaultProguardFile("proguard-android-optimize.txt"),
32                    "proguard-rules.pro"
33                )
34            }
35        }
36        compileOptions {
37            sourceCompatibility = JavaVersion.VERSION_1_8
38            targetCompatibility = JavaVersion.VERSION_1_8
39        }
40        kotlinOptions {
41            jvmTarget = "1.8"
42        }
43        buildFeatures {
44            mlModelBinding = true
45        }
46    }
47 }
```

```

46
47 chaquopy {
48     defaultConfig {
49         pip{
50             install("matplotlib")
51             install("numpy")
52             install("tensorflow==2.1.0")
53             install("protobuf==3.20.0")
54             install("scipy")
55         }
56     }
57     productFlavors { }
58     sourceSets { }
59 }
60
61 dependencies {
62
63     implementation("androidx.core:core-ktx:1.9.0")
64     implementation("androidx.appcompat:appcompat:1.6.1")
65     implementation("com.google.android.material:material:1.10.0")
66     implementation("androidx.constraintlayout:constraintlayout:2.1.4")
67     implementation("org.tensorflow:tensorflow-lite:2.16.1")
68     implementation ("org.tensorflow:tensorflow-lite-support:0.3.1")
69     implementation ("org.tensorflow:tensorflow-lite-gpu:2.16.1")
70     implementation("com.davemorrissey.labs:subsampling-scale-image-view
71         :3.10.0")
72     implementation("org.tensorflow:tensorflow-lite-metadata:0.1.0")
73     testImplementation("junit:junit:4.13.2")
74     androidTestImplementation("androidx.test.ext:junit:1.1.5")
75     androidTestImplementation("androidx.test.espresso:espresso-core:3.5.1")
76 }
```

**Listing 10.3.** build.gradle.kts (Project: MyApplication)

```

1 import scipy.io.wavfile as wavfile
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import os
5 import tensorflow as tf
6
7 def decode_audio(audio_binary):
8     audio, _ = tf.audio.decode_wav(contents=audio_binary)
9     return tf.squeeze(audio, axis=-1)
10
11 def get_spectrogram(waveform):
12     input_len = 16000
13     waveform = waveform[:input_len]
14     zero_padding = tf.zeros([16000] - tf.shape(waveform), dtype=tf.float32)
15     waveform = tf.cast(waveform, dtype=tf.float32)
16     equal_length = tf.concat([waveform, zero_padding], 0)
17     spectrogram = tf.signal.stft(equal_length, frame_length=255, frame_step
18         =128)
19     spectrogram = tf.abs(spectrogram)
20     spectrogram = spectrogram[..., tf.newaxis]
21     return spectrogram
22
23 def preprocess_single_file(file_path):
24     audio_binary = tf.io.read_file(file_path)
25     waveform = decode_audio(audio_binary)
26     spectrogram = get_spectrogram(waveform)
27     spectrogram = tf.expand_dims(spectrogram, axis=0)
28     return spectrogram
29
30 def analyze(file_path, model_path=None):
31     sr, y = wavfile.read(file_path)
32
33     if len(y.shape) > 1:
34         y = y[:, 0]
35
36     y_normalized = y / np.max(np.abs(y))
37
38     plt.figure(figsize=(14, 5))
39     plt.plot(np.linspace(0, len(y_normalized) / sr, num=len(y_normalized)),
40             y_normalized)
41     plt.title("Przebieg sygnału")
42     plt.xlabel("Czas [sekundy]")
43     plt.ylabel("Amplituda (od -1 do 1)")
44     waveform_path = os.path.join(os.path.dirname(file_path), "waveform.png"
45         )
46     plt.savefig(waveform_path)

```

```

44     plt.close()

45

46     from scipy.signal import spectrogram
47     f, t, Sxx = spectrogram(y, sr)
48     plt.figure(figsize=(14, 5))
49     plt.pcolormesh(t, f, 10 * np.log10(Sxx))
50     plt.ylabel('Częstotliwość [Hz]')
51     plt.xlabel('Czas [sekundy]')
52     plt.title('Spekrogram')
53     cbar = plt.colorbar(format='%+2.0f dB', orientation='horizontal', pad
54                         =0.2)
55     cbar.set_label('Intensywność [dB]')
56     spectrogram_path_1 = os.path.join(os.path.dirname(file_path), "spectrogram.png")
57     plt.savefig(spectrogram_path_1)
58     plt.close()

59     spectrogram = preprocess_single_file(file_path)
60     spectrogramFlatten = spectrogram.numpy().flatten()

61

62     spectrogramTxtPath = os.path.join(os.path.dirname(file_path), "spectrogram.txt")
63

64     with open(spectrogramTxtPath, 'w') as f:
65         for item in spectrogramFlatten:
66             f.write("%s\n" % item)

67

68     return waveform_path, spectrogram_path_1, spectrogramTxtPath

```

**Listing 10.4.** model\_skrypt.py