Wydział Podstawowych Problemów Techniki
Politechnika Wrocławska

Rozpoznawanie akordów gitarowych w czasie rzeczywistym w oparciu o dźwięk

Maksym Telepchuk
Nr indeksu: 236723

Praca inżynierska napisana
pod kierunkiem
dr inż. Małgorzaty Sulkowskiej

Politechnika
Wrocławska
Wrocław 2019

Spis treści

Wstęp

Praca swoim zakresem obejmuje klasyfikację ścieżek dźwiękowych przy użyciu konwolucyjnych sieci neuronowych w czasie rzeczywistym. W przypadku tej pracy ścieżkami dźwiękowymi będą jednosekundowe nagrania akordów sześciostrunowej akustycznej gitary w standardowym dostrojeniu. Nagrywanie takich ścieżek będzie się odbywać przy użyciu urządzeń mobilnych. Ze względu na dużą ilość możliwych akordów, dla systemu klasyfikacji zostały wybrane 10 podstawowych bardzo popularnych chwytów gitarowych.

Celem pracy jest zaimplementowanie aplikacji mobilnej, która prowadzi nagrywanie ścieżki dźwiękowej oraz co sekundę wyświetla nazwę akordu, który jest aktualnie grany przez użytkownika. Dodatkowo aplikacja powinna zapisywać wyniki dla każdego granego akordu oraz udostępniać łatwy sposób ich przejrzenia.

Istnieje szereg aplikacji o zbliżonej funkcjonalności, przy czym aplikacji rozpoznawania akordów dla gitary akustycznej jest o wiele mniej, niż na przykład tzw. stroików, celem których jest rozpoznawanie jednej struny w przeciwieństwie do kilku strun granych jednocześnie.

Jedną z aplikacji, która zawiera system rozpoznawania akordów, jest Yousician [1] - bardzo popularny serwis (ponad 10 milionów użytkowników) służący uczeniu się grania na różnych instrumentach muzycznych. Jednym z ćwiczeń proponowanych przez Yousician jest zagranie zadanego akordu. Aplikacja nasłuchuje i ocenia, czy akord został zagrany prawidłowo. Jednak taki system daje tylko możliwość stwierdzenia, czy akord należy do konkretnej klasy czy nie.

Inną aplikacją rozpoznającą całe akordy jest MyChord [2], jednak ta aplikacja pozwala tylko na wczytywanie już nagranej wcześniej ścieżki dźwiękowej. Aplikacje Chord detector [3] oraz Chord detector ZAX [4] są najbardziej zbliżone do aplikacji proponowanej w tej pracy, jednak mają one niski wskaźnik poprawnych klasyfikacji, o czym między innym świadczą oceny tych aplikacji. Oprócz tego żadna ze wspomnianych aplikacji nie udostępnia historii swoich klasyfikacji.

Praca składa się z pięciu rozdziałów. W rozdziale pierwszym została przedstawiona analiza problemu oraz opis poszczególnych komponentów implementowanego sytemu. W rozdziale drugim został opisany proces zgromadzenia zbioru danych dla sieci neuronowej oraz teoretyczne aspekty przetwarzania zgromadzonych danych. W rozdziale trzecim porównano algorytmy głębokiego uczenia się, w szczególności sieci neuronowe typu 'feedforward' oraz konwolucyjne sieci neuronowe. W tym rozdziale również została opisana wybrana architektura sieci neuronowej oraz jej proces uczenia się. W rozdziale czwartym opisano proces implementacji systemu aplikacji mobilnej, web serwera dla tej aplikacji oraz parametry dobrane w trakcie uczenia modelu sieci neuronowej. Końcowy rozdział stanowi podsumowanie uzyskanych wyników oraz perspektywy rozwoju przedstawionego systemu.

Analiza problemu

W ostatnich latach istotnie wzrosło zainteresowanie uczeniem maszynowym. Jedną z przyczyn tego stało pojawienie się względnie taniego sprzętu o bardzo wysokiej mocy obliczeniowej. Coraz częściej algorytmy uczenia maszynowego znajdują swoje zastosowania w różnych sferach ludzkiego życia.

Klasyfikacja wieloparametrowych danych jest jednym z zadań, w którym uczenie maszynowe jest bardzo przydatnym. Dla zaimplementowania systemu klasyfikacji został wybrany algorytm konwolucyjnej sieci neuronowej, która przyjmuje dane na wejściu oraz zwraca prawdopodobieństwo należenia do każdej z dostępnych klas. Klasę o najwyższym prawdopodobieństwie traktuje się jako ostateczny wynik klasyfikacji.

Żeby sieć skutecznie przeprowadzała klasyfikację, potrzebuje ona danych, na których będzie 'trenować'. Po trenowaniu warto sprawdzić czy przypadkiem sieć nie dopasowała się tylko i wyłącznie do danych treningowych (tzw. przetrenowanie). Dlatego sieć dodatkowo sprawdzano na zbiorze walidacyjnym. Na podstawie tej walidacji można stwierdzić jak sieć się zachowuje dla nowych danych. Kiedy skuteczność dla zbioru walidacyjnego jest wystarczająco wysoka, ostatecznie sieć jest testowana na tzw. zbiorze testowym.

Ścieżki dźwiękowe podawane do sieci neuronowych w początkowym stanie często wymagają długiego czasu uczenia się, dużej ilości parametrów sieci oraz dużego zbioru danych. Po to wprowadzono warstwę przetwarzania danych, celem której jest ekstrakcja najbardziej istotnych cech oraz zmniejszenie ilości parametrów. Dla dźwięku takim przetwarzaniem będzie wyliczenie spektrogramu w skali melowej. Spektrogram jest funkcją intensywności występujących częstotliwości względem czasu. Podstawowym algorytmem wyliczenia spektrogramu jest transformata Fouriera.

Ponieważ w tej pracy implementowano aplikację mobilną, byłoby nieefektywnym obliczenie klasyfikacji na urządzeniu mobilnym ze względu na duży rozmiar sieci neuronowej. Dlatego został zaimplementowany serwer, który służy interfejsem rzeczywistego klasyfikatora. Łączenie z serwerem odbywa się protokołem HTTP, w ciele zapytania HTTP jest przekazywany nagrany akord, który serwer przetwarza, po czym podaje przetworzone dane do sieci neuronowej. Ostatecznie serwer wysyła odpowiedź HTTP, w ciele której znajduje się wynik klasyfikacji.

Po stronie aplikacji mobilnej, którą nazywano klientem, odbywają się wszystkie procesy związane z nagrywaniem i dzieleniem ścieżek dźwiękowych. Są one wysyłane do serwera, a zwrócone wyniki klasyfikacji są pokazywane użytkownikowi oraz zapisywane do lokalnej bazy danych.

Pracę nad aplikacją rozpoznawania akordów akustycznej gitary można podzielić na pięć części:

1. Zebranie zbioru danych;
2. Przetworzenie ścieżek dźwiękowych do postaci spektrogramów;
3. Wytrenowanie modelu sieci neuronowej na zebranym zbiorze danych;
4. Zaimplementowanie serwera, na którym będą się wykonywać obliczenia;
5. Zaimplementowanie aplikacji mobilnej, która umożliwia nagrywanie i wysyłanie ścieżek dźwiękowych oraz wyświetlanie wyników klasyfikacji.

Dane

2.1 Zgromadzenie danych

W tym rozdziale został opisany proces zgromadzenia zbioru danych potrzebnych do trenowania modelu. Reprezentatywna baza danych to podstawa w modelach głębokiego uczenia. Im bardziej różnorodne dane, tym więcej model ma szansę nauczyć się w procesie trenowania. Uczenie głębokie jest szeroko stosowane dla danych nie strukturyzowanych takich jak obrazy, dźwięk oraz tekst. Takie dane nie są zorganizowane w typowy bazodanowy sposób w wiersze i kolumny, przez co jest ciężej wyliczyć wspólne cechy liniowymi przekształceniami wejściowych danych.

W tym rozdziale zostanie opisany proces zgromadzenia ścieżek dźwiękowych formatów WAV i M4A dla danych wygenerowanych i nagranych manualnie odpowiednio. Akord (chwyt) gitarowy jest kombinacją przyciśniętych strun gitarowych, które są grane jednocześnie. Biciem jest nazywany sposób grania akordu, w którym struny gitarowe są grane po kolej od szóstej do pierwszej struny (z góry do dołu) lub odwrotnie (z dołu do góry). Każda ścieżka dźwiękowa ze zbioru jest długości 1 sekundy oraz zawiera jedno bicie akordu, które się odbywa w pierwszej ćwierci sekundy. Akordy występujące w zbiorze danych są przypisane do jednej klasy ze zbioru {A, Am, C, D, Dm, E, Em, F, G, H7}.

Dla zgromadzenia zbioru danych część danych była nagrana w warunkach zbliżonych do docelowych przy użyciu gitary akustycznej. Pozostałe dane były wygenerowane sztucznie z wykorzystaniem dostępnych zbiorów nagranych oddzielnie gitarowych strun w warunkach studyjnych. W sumie zgromadzony zbiór danych zawiera 3400 ścieżek dźwiękowych, czyli 340 instancji dla każdej klasy.

2.1.1 Dane nagrane

Dla wydajnego modelu głębokiego uczenia dane używane do trenowania muszą bezpośrednio być związane z problemem. Oznacza to, że dane muszą być podobnymi w jak największym stopniu do danych, które taki model ma na celu klasyfikować. Model ma za zadanie rozpoznawanie akordów na podstawie ścieżek nagranych przez przeciętnego użytkownika. Zakłada się, że takie nagrywanie będzie się odbywać we względnie cichym środowisku, bez użycia profesjonalnego sprzętu i z dość blisko położonym mikrofonem.

Przy nagrywaniu danych dla tej części zbioru zostały wybrane odpowiednie warunki. Nagrywanie się odbywało w cichym pokoju, z urządzeniem odbierającym położonym na biurku. W tym przypadku był to mikrofon smartfona Samsung Galaxy A8. Mikrofon nagrywał ścieżki dźwiękowe w formacie M4A z szybkością transmisji bitów 192 kb/s z częstotliwością próbkowania 48 kHz. Mikrofon był obrócony w stronę gitary, położony w odległości 20-30 centymetrów od otworu rezonansowego. W celu wyraźniejszego brzmienia została użyta kostka gitarowa.

M4A to rozszerzenie nazwy pliku używane do reprezentowania skompresowanego pliku audio w kontenerze MPEG-4. Pliki są skompresowane w celu ekonomii miejsca na dysku oraz przyspieszenia transmisji danych do serwera. Jakość takich skompresowanych ścieżek dźwiękowych zależy od szybkości transmisji bitów. Szybkość transmisji bitów (ang. bitrate) odnosi się do liczby bitów przetwarzanych na sekundę. W formacie audio M4A można spotkać różne wartości takie jak 128 kb/s, 192 kb/s, 256 kb/s. Im wyższa jest ta wartość, tym mniej ścieżka dźwiękowa jest skompresowana. 192 kb/s jest powszechnym standardem szybkości transmisji bitów dla większości mikrofonów w nowoczesnych urządzeniach mobilnych.

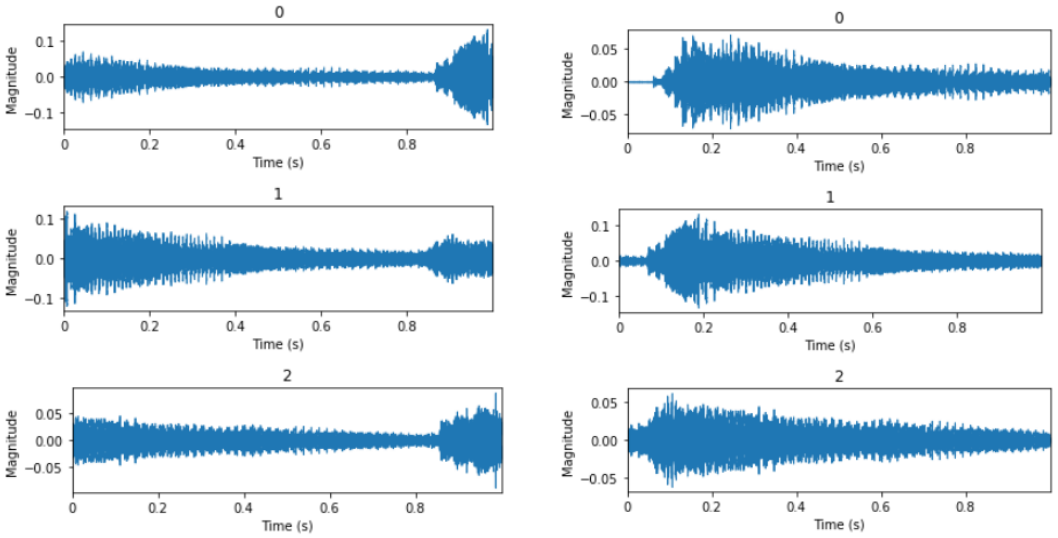
Dla każdej klasy akordu została nagrana ścieżka dźwiękowa o długości ok. 64 sekundy. Pierwsze 2 sekundy ze ścieżki zostały potraktowane jako margines, czyli czas na przygotowanie do nagrywania. W celu zapewnienia możliwości późniejszego precyzyjnego dzielenia ścieżki na równe jednosekundowe kawałki, został użyty metronom, który odbijał jedno bicie raz na sekundę, czym pomagał grać akordy w mniej więcej tych samych odstępach czasowych. Żeby uniknąć dodatkowych szumów, symulacja metronomu była odbierana przez słuchawki.

Dla utrzymywania równowagi w zbiorze,wykorzystano bicia parzyste i nieparzyste - każde parzyste bicie odbywa się z góry do dołu, każde nieparzyste - z dołu do góry. Pozostała część ścieżki była potrakto-

wana jako margines, czyli czas na zatrzymywanie nagrania. W ten sposób zostało nagranych 60 instancji po jednej sekundzie dla każdej klasy akordów.

Dzielenie w sposób opisany powyżej w wielu przypadkach ma swoje wady. Traktowanie marginesu początkowego zawsze z długością 2 sekundy często powodowało to, że na końcu jednej instancji było słychać kawałek początku grania następnej po kolei instancji. Jest to sprzeczne z założeniem, że w każdej jednosekundowej ścieżce jest grany dokładnie jeden akord. Taki efekt był powodowany niedokładnością postrzegania równych odstępów pomiędzy jednym a drugim biciem.

W związku z tym do każdej nagranej 64 sekundowej ścieżki była dobierana wartość początkowego marginesu nieco mniejsza od 2 sekund, żeby uniknąć występowania instancji ze złą końcówką. Wartość ta była dobierana eksperymentalnie i, żeby upewnić się, że w żadnej instancji nie ma tego złego efektu, każda instancja była wizualizowana (rys 2.1).



(a) Odstęp początkowy równy 2s

(b) Odstęp początkowy równy 1.8s

Rysunek 2.1: Trzy pierwsze instancje Am dla różnych odstępów początkowych

Wizualizacja pozwoliła na szybkie sprawdzenie obecności niewłaściwych dźwięków na końcówkach instancji oraz dobieranie indywidualnej wartości marginesu początkowego. W ten sposób eksperymentalnie udało się wyliczyć wartości marginesów początkowych podanych w tablicy 2.1.

A	Am	C	D	Dm	E	Em	F	G	H7
1.8	1.8	1.85	1.82	1.9	1.85	1.8	1.8	1.8	1.88

Tablica 2.1: Wartości odstępów początkowych w sekundach

W końcu dla każdej klasy zostało nagranych 30 instancji każdego akordu granych z góry do dołu oraz 30 z dołu do góry. W sumie dało to zbiór danych zawierający 600 instancji akordów nagranych samodzielnie.

2.1.2 Dane wygenerowane

Żeby model sieci neuronowej lepiej się nauczył klasyfikować dane, potrzebuje on ich jak najwięcej. Samodzielne zgromadzenie wielkiego różnorodnego zbioru może się okazać bardzo czasochłonnym. W dodatku taka samodzielność też jest ograniczona przez dostępne zasoby - ilość źródeł dźwięku, dobry sprzęt do ich nagrywania. W celu zwiększenia zgromadzonego dotychczas zbioru, który się składa tylko z nagranych ścieżek dźwiękowych, został wykorzystany dostępny publicznie zbiór danych "The NSynth Dataset"[5].

„NSynth to zbiór danych audio zawierający 305 979 nut, z których każdy ma unikalną wysokość, barwę i zachowanie. Dla 1006 instrumentów z komercyjnych bibliotek zostały wygenerowane czterosekundowe monofoniczne fragmenty audio 16 kHz, zwane nutami, na podstawie każdego tonu standardowego pianina MIDI o zakresie (21-108), a także pięciu różnych prędkości (25, 50, 75, 100, 127). Nuta była trzymana przez pierwsze trzy sekundy i pozostawiona do rozpadu przez ostatnią sekundę"[5].

Ścieżki w tym zbiorze mają rozszerzenie plików WAV. Format audio WAV jest uważany za „bezstratny" czyli taki, w którym nie jest wykonywana kompresja danych.

Dane w zbiorze są uporządkowane według:

- Grupy źródła, czyli metody produkcji dźwięku dla instrumentu nutowego.
    1. acoustic: nagrane nuty z instrumentów akustycznych
    2. electronic: nagrane nuty z instrumentów elektronicznych
    3. synthetic: syntezowane nuty
- Wysokopoziomowej rodziny instrumentów. Każda nuta ma dokładnie jedną rodzinę.
    1. bass
    2. brass
    3. flute
    4. guitar
    5. keyboard
    6. mallet
    7. organ
    8. reed
    9. string
    10. synth lead
    11. vocal
- Typów jakości: właściwości dźwiękowe ścieżki, do każdej są przypisane kilka lub żaden.
    1. bright : Duża ilość wysokich częstotliwości i silnych harmonik górnych.
    2. dark : Wyraźny brak treści o wysokiej częstotliwości, zapewniający wyciszony i basowy dźwięk.
    3. distortion : Fale, które wytwarzają charakterystyczny chrupiący dźwięk i obecność wielu harmonik. Czasami są w połączeniu z szumem nieharmonicznym.
    4. fast_decay: Obwiednie amplitudy wszystkich harmonik zanikają bardo wcześnie, jeszcze przed punktem 'zaniku' (3 sekunda).
    5. long_release: Obwiednie amplitudy zanikają powoli po punkcie 'zaniku', czasem są wciąż obecne na końcu ścieżki (4 sekunda).
    6. multiphonic: Obecne częstotliwości nadtonu związane z więcej niż jedną częstotliwością podstawową.
    7. nonlinear_env: Modulacja dźwięku z wyraźnym zachowaniem odpowiednim innym niż monotoniczny spadek nuty. Może również zawierać zachowanie filtrujące, a także zachowanie dynamiczne.
    8. percusive: Głośny nieharmoniczny dźwięk na początku nuty
    9. reverb: Akustyka pomieszczenia, której nie można było usunąć z oryginalnej ścieżki.
    10. tempo-synced: Rytmiczna modulacja dźwięku do ustalonego tempa.

Jak widać, nie wszystkie instrumenty w NSynth Database są gitarami. Natomiast do generowania zbioru akordów chciano zastosować tylko nuty nagrane z akustycznych źródeł gitarowych. W sumie jest 13343 takich nut wyprodukowanych przez 37 różnych gitar.

Dla sześciostrunowej gitary akustycznej standardowym dostrojeniem jest (40,45,50,55,59,64) - wektor tonów standardowego pianina MIDI , gdzie 40 - ton szóstej struny, a 64 - ton pierwszej struny. Przy takim dostrojeniu klasyczna sześciostrunowa gitara akustyczna, mająca 20 progów, jest w stanie produkować tony z zakresu (40-84). Zatem nie wszystkie gitary akustyczne ze zbioru NSynth są w stanie zagrać każdą nutę z tego zakresu tonów oraz zakresu prędkości zaznaczonej w opisie zbioru. Co więcej, każdy akord wymaga oddzielnego zestawu tonów, przy czym jedna gitara czasami jest w stanie wyprodukować jeden akord, a nie jest w stanie innego, nawet bardzo podobnego. Oznacza to, że dla symulacji różnych klas akordów będą czasami wykorzystywane różne gitary.

W trakcie badania wygenerowanych ścieżek zauważono, że niektóre instrumenty produkują zbyt zniekształcony, przytłumiony, krótkotrwały lub niewłaściwy dźwięk. Jedyną własnością wykazującą charakter brzmienia każdej z 37 akustycznej gitary jest zbiór typów jakości. Wbrew pozorom, analiza tego pola nie dała jednoznacznych wyników. Często zdarza się tak, że różne nuty nagrane przez tą samą gitarę mają różne zestawy jakości, a czasami jakość dla jednej nuty różni się dla różnych prędkości grania. Też nie jest rzadkim to, że nuta nie ma żadnego przypisanego typu jakości oraz nuty z podobnym zestawem typów jakości mają całkiem inne brzmienie.

W związku z powyższym powstała potrzeba przesłuchania przykładów wygenerowanych ścieżek dla każdego akordu oraz dla każdej gitary. Dla niektórych instrumentów takie sprawdzenie nie było skutecznym w związku z brakiem możliwości grania wszystkich akordów ze zbioru dla niektórych gitar. Jednak nawet na podstawie przykładów z części klas charakter brzmienia gitary był zrozumiały oraz decydował o dalszym wykorzystaniu gitary jako źródła.

Po przesłuchaniu wygenerowanych ścieżek, była podjęta decyzja o rezygnacji z 22 gitar. Sposób wybierania nieodpowiednich gitar różnił się dla każdego instrumentu. Przykładowo, zostały wyrzucone gitary z prawie natychmiastowym wyciszeniem, nuty mające przypisaną jakość 'distortion' i 'multiphonic' też zostały uznane za nieodpowiednie. Również zauważono, że typ jakości 'bright' głównie, ale nie zawsze, występuje w gitarach ze zbyt ostrym metalowym dźwiękiem; niektóre tony jakości 'reverb' zawierały zbyt wielkie zaszumienie; niektóre tony jakości 'dark' były zbyt przytłumione i zawierały bardzo dużo niskich częstotliwości; niektóre tony z brakiem przypisanego zestawu jakości były uznane za nieodpowiednie z powyższych powodów. W wyniku zbiór gitar, wykorzystywanych do generowania akordów został zredukowany z 37 gitar do 15.

Dla symulacji akordu jest potrzebny zestaw ścieżek dźwiękowych dla nagranych strun przyciśniętych na odpowiednich progach oraz sposób ich łączenia. Na przykład, żeby zagrać akord Em czwarta i piąta struna musi być przyciśnięta na drugim progu oraz wszystkie pozostałe struny muszą być grane otwarcie. Oznacza to, że do standardowego dostrojenia jest dodawany wektor (0,2,2,0,0,0). W końcu (40,47,52,55,59,64) jest wektorem tonów potrzebnych do generowania Em.

Dla każdego klasy akordu na podstawie jej wektora tonów były dobierane instrumenty, które są w stanie taki akord zagrać. Instrument jest w stanie zagrać akord wtedy, kiedy zawiera nuty potrzebne do grania każdego tonu w tym akordzie. Model powinien rozpoznawać sposób grania akordów z góry do dołu i z dołu do góry. Dlatego ilość wygenerowanych akordów granych z góry do doły i z dołu do góry jest równa dla każdej klasy.

Przy generowaniu akordu instrument jest pobierany z posortowanej według numeru gitary listy. Takie pobieranie jest cykliczne , czyli po ostatnim instrumencie będzie pobrany pierwszy w liście. W celu stworzenia różnorodnego zbioru, przy generowaniu nowego elementu zbioru, z zakresu prędkości jest wybierana losowa wartość.

W granym akordzie zawsze istnieją małe odstępy pomiędzy graniem osobnych strun; każda następna struna jest grana po kolei. Przy czym odstępy te też mogą się nieco różnić. Przykładowo, żeby zagrać akord Am z zestawu nagranych danych, szacunkowo potrzebne jest od 0.05 do 0.15 sekundy. Ponieważ w akordzie Am jest granych 5 strun, występują 4 odstępy pomiędzy nimi. Żeby naśladować takie zachowanie, odstępy pomiędzy dwiema sąsiednimi strunami zostały losowo wybrane z zakresu (0.0125, 0.0375) w związku z czym długość grania wygenerowanego akordu będzie w takim samym zakresie jak przy akordach nagranych w rzeczywistości. Zatem ścieżki dźwiękowe dla odpowiednich strun zostały nałożone na siebie uwzględniając wspominane odstępy.

Niezbędnym jest dodanie ciszy na początku ścieżki dźwiękowej, ponieważ brak takiego odstępu będzie powodował, że sieć będzie oczekiwała grania akordu od samego początku ścieżki dźwiękowej, co w rzeczywistości rzadko się zdarza. Przy zgromadzeniu nagrywanych danych, początkowy odstęp był w zakresie (0,0.2) sekundy, więc przy wygenerowaniu wartość odstępu była wybrana losowo z tego samego zakresu. Długość każdej wygenerowanej ścieżki została obcięta do pierwszej sekundy, żeby ujednolicić zbiory nagranych i wygenerowanych danych.

W wyniku opisanego procesu zostało wygenerowanych 2800 ścieżek dźwiękowych (280 dla każdej klasy).

2.2 Przetwarzanie dźwięku

Dla każdego zgromadzonego zbioru danych często jest potrzebna dodatkowa warstwa przetwarzania, która umożliwi łatwiejszą identyfikację wzorców, zależności i różnic zawartych w danych. Przetwarzanie danych jest istotnym etapem w uczeniu maszynowym, ponieważ pozwala ono na ekstrakcję cech, które

poprawiają dokładność modelu oraz zapobiegają zbytniemu dopasowaniu modelu do danych. Szeroko stosowanym podejściem do przetwarzania dźwięku przed wrzuceniem ich do modelu uczącego się jest przekształcenie ścieżek dźwiękowych do postaci spektrogramów. Pomagają one wizualizować występujące częstotliwości w ścieżce dźwiękowej. Do utworzenia spektrogramów potrzebny jest zestaw narzędzi matematycznych, głównym z których jest transformata Fouriera.

2.2.1 Transformata Fouriera

W tej części pracy zostaną omówione transformata Fouriera i wszystkie teoretyczne mechanizmy, na których ona się opiera.

„Sygnał jest abstrakcyjnym modelem dowolnej mierzalnej wielkości zmieniającej się w czasie, generowanej przez zjawiska fizyczne" [6]. Innymi słowami, sygnały opisują zmianę zjawisk fizycznych w czasie. Przetwarzanie sygnałów jest zestawem ich przekształceń, które ułatwiają późniejszą analizę i interpretację. Dziedzina cyfrowego przetwarzania sygnałów dotyczy wykonywania tych przekształceń na sygnałach świata rzeczywistego za pomocą urządzeń komputerowych. Komputery są ograniczone pamięciowo oraz są w stanie działać tylko na ograniczonych dyskretnych wartościach (ciągach zer i jedynek). Zanim będzie można przetworzyć ciągły sygnał występujący w rzeczywistości za pomocą komputera, musi zostać on przeskalowany na dyskretną reprezentację. Warto pamiętać, że dyskretny sygnał zawsze będzie przybliżeniem jego ciągłego odpowiednika.

Dla otrzymania dyskretnej reprezentacji sygnału jest stosowane próbkowanie. Próbkowanie (ang. sampling) jest okresowym mierzeniem pewnej wartości. Każdy z takich pomiarów jest nazywany próbką (ang. sample). Zatem każdy sygnał dyskretny jest listą próbek otrzymanych w wyniku próbkowania ciągłych sygnałów z rzeczywistego świata. Ponieważ mierzenie się odbywa okresowo, można wyliczyć częstotliwość próbkowania (ang. sampling rate):

$$S = 1/T,$$

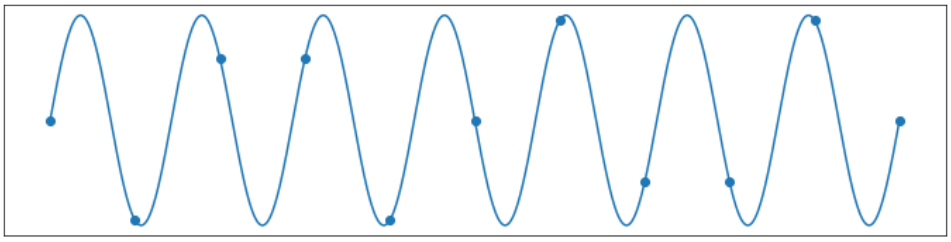
gdzie S - częstotliwość próbkowania, T - czas pomiędzy mierzeniem dwóch sąsiednich próbek (okres mierzenia). Częstotliwość mierzy się w cyklach na sekundę (jednostka częstotliwości znana również jako herc [Hz]).

Przy wyższej częstotliwości próbkowania zbierana jest większa ilość informacji o sygnale, która może być wykorzystana do dokładniejszej analizy. Pobieranie próbek wiąże się z pewnymi kosztami, ponieważ każda próbka musi być przechowywana w ograniczonej pamięci. Zwiększenie częstotliwość próbkowania podnosi koszt. Zatem przy próbkowaniu ważnym jest dokładne zrozumienie, jak często należy próbkować, aby uniknąć utraty informacji która będzie istotna przy przetwarzaniu.

Twierdzenie 2.1 (Nyquista-Shannona o próbkowaniu [7]) *Jeśli funkcja $x(t)$ nie zawiera częstotliwości wyższych niż B Hz, jest ona całkowicie określona przez podanie jej rzędnych w szeregu punktów w odstępach $1/(2B)$ sekund.*

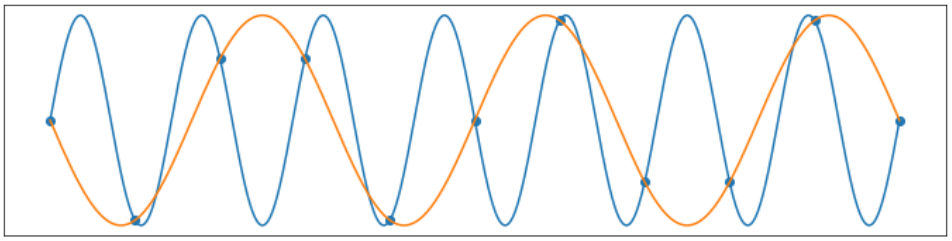
Twierdzenie 2.1 mówi jak uniknąć straty informacji oraz nadużycia pamięci w skutku zmniejszenia lub zwiększenia częstotliwości próbkowania. W praktyce oznacza to, że przy częstotliwości próbkowania S herców, nie jest możliwym dokładne odtworzenie ciągłego odpowiednika dla żadnego sygnału zawierającego częstotliwości większe lub równe $S/2$. Częstotliwość $S/2$ również jest nazywana częstotliwością Nyquista. Przy przetwarzaniu sygnałów wystarczy próbkować z częstotliwością wyższą niż dwukrotność najwyższej składowej tychże sygnałów.

Dla przykładu, na rysunku 2.2 jest pokazany sygnał składający się z fali sinusoidalnej o częstotliwości 7 kHz. Niech częstotliwość próbkowania S wynosi 10 kHz. Niebieskimi punktami jest oznaczonych 11 wymierzonych próbek.



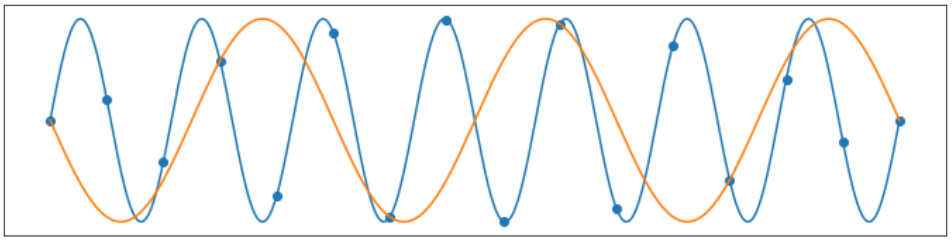
Rysunek 2.2: Sygnał o częstotliwości 7 kHz

Rysunek 2.3 demonstruje, że przy analizie takiego zestawu próbek jest niemożliwym odróżnienie oryginalnego sygnału od sygnału składającego się z fali sinusoidalnej o częstotliwości 3 kHz.



Rysunek 2.3: Aliasing dwóch sygnałów

Dopasowanie do tej samej listy próbek różnych sygnałów nazywa się aliasingiem a dopasowane sygnały aliasami. Dla umożliwienia rozróżnienia sygnałów dla tego przykładu, jest niezbędnym zwiększenie S do wartości powyżej 14 kHz (z tw. 2.1). Na rysunku 2.4 jest pokazany wynik próbkowania sygnału dla $S = 15$ kHz. Przy takiej częstotliwości aliasing nie wystąpi i sygnały będą generować różne listy próbek.



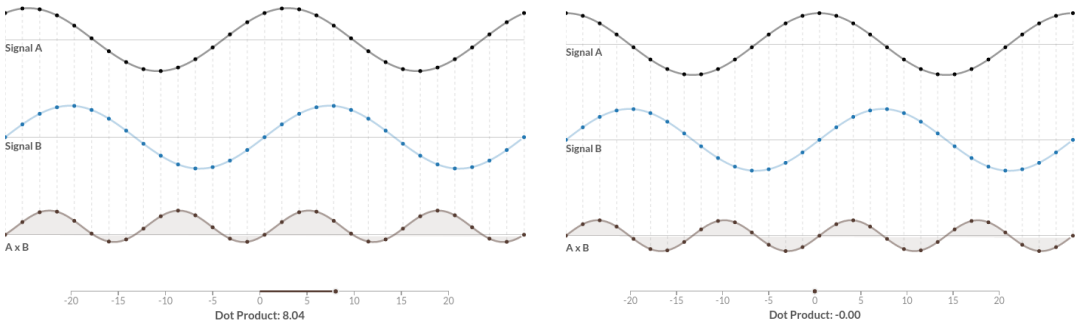
Rysunek 2.4: Dobrze dobrana wartość S

Dla mierzenia stopnia podobieństwa sygnałów wprowadza się termin korelacji. Sygnały wykazują dodatnią korelację, gdy rosną i maleją w tych samych miejscach. Sygnały wykazują ujemną korelację, jeśli jeden rośnie tam, gdzie drugi maleje, i odwrotnie. Sygnały są dekorelowane, jeśli nie ma między nimi dostrzegalnego związku. Do mierzenia korelacji między dwoma sygnałami jest wykorzystywany iloczyn skalarny. Iloczyn skalarny (ang. dot product) sygnału A oraz sygnału B opisuje się wzorem:

$$A \times B = \sum_i A_i \cdot B_i$$

gdzie A_i, B_i są odpowiednimi współrzędnymi wektorów A i B.

Na rysunku 2.5 podano wykresy fal sinusoidalnych o tej samej częstotliwości oraz z różnymi przesunięciami fazowymi i porównano z wykresem ich iloczynu skalarnego.



(a) $\varphi_A = 1$ radian, $\varphi_B = 0$ (b) $\varphi_A = \pi/2, \varphi_B = 0$

Rysunek 2.5: Korelacje fal sinusoidalnych [8]

Wartość iloczynu skalarnego może być traktowana jako obszar pomiędzy jego wykresem (oznaczony jako $A \times B$) a osią współrzędnych. Jak widać na rys. 2.5a, przy różnicy w przesunięciach fazowych

równej 1 radian dwie fale sinusoidalne nadal wykazują dodatnią korelację, o czym mówi wysoka wartość iloczynu skalarnego. Przy różnicy w przesunięciach równej $\pi/2$ (rys. 2.5b) iloczyn skalarny wynosi 0, czyli sygnały są dekorelowane. Ostatnie porównanie prowadzi do wniosku, że fala sinusoidalna i cosinusoidalna są dekorelowane.

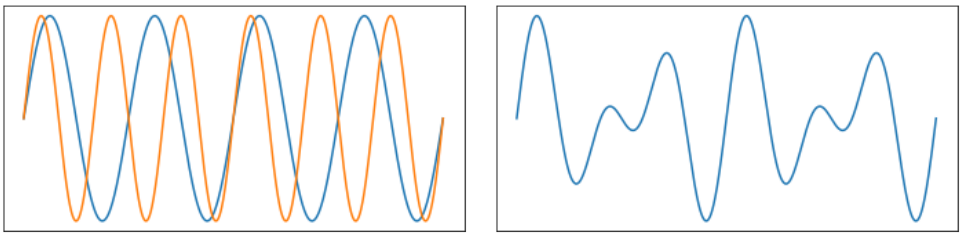
„Dźwięk jest wrażeniem słuchowym, spowodowanym falą akustyczną rozchodzącą się w ośrodku sprężystym" [9]. Fala akustyczna jest okresowa i ma swoją częstotliwość. Można traktować dźwięk jako sygnał, gdzie jest mierzona wartość ciśnienia ośrodka, w którym rozchodzi się dźwięk.

Dźwięki z niską częstotliwością są określane mianem basów, a dźwięki o wysokich częstotliwości są nazywane tonami wysokimi. Osoby o wyjątkowym słuchu słyszą dźwięki od około 20 Hz do 20 kHz . Ponieważ ludzie faktycznie nie słyszą częstotliwości powyżej 20 kHz, częstotliwości powyżej tego limitu są usuwane w większości formatów audio przed próbkowaniem sygnałów. Przez to we wszystkich popularnych formatach audio częstotliwość próbkowania sygnałów ma wartość powyżej 40 kHz, czyli powyżej dwukrotności najwyższej słyszalnej częstotliwości. Skala "słyszalności" nie jest liniowa, tylko logarytmiczna. Dźwięki, na które ludzkie uszy są najbardziej wrażliwe, istnieją w zakresie od około 20 Hz do 8 000 Hz (przykładowo ludzka mowa mieści się w zakresie od 300 do 3000 Hz).

Każdy sygnał dźwiękowy może być traktowany jako suma fal sinusoidalnych o pewnych częstotliwościach zwanych składowymi. Na rysunku 2.6a są podane dwa sygnały sinusoidalne o częstotliwościach 2 kHz i 3 kHz, które rozchodzą się jednocześnie. W wyniku takiego nałożenia tworzy się sygnał, który jest sumą dwóch poprzednich (rys. 2.6b). Funkcja tego sygnału może wyglądać następująco:

$$s = sin(2x) + sin(3x)$$

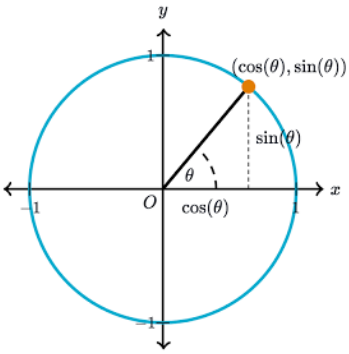
gdzie s - sygnał, który występuje przy próbkowaniu.



(a) 2 kHz i 3 kHz brzmiące jednocześnie (b) Suma brzmiących jednocześnie sygnałów

Rysunek 2.6: Składowe sygnału

Barwa dźwięku to właściwość dźwięku różniąca się od wysokości i intensywności. Różne wzory fali akustycznej wytwarzają dźwięki o różnych barwach. Oznacza to, że różne instrumenty muzyczne mogą grać tą samą nutę, ale dźwięk wytworzony przez każdy taki instrument jest wyraźnie inny. Większość dźwięków muzycznych składa się z częstotliwości podstawowej i dodatkowych harmonik (wielokrotności częstotliwości podstawowej). Barwa dźwięku często zależy od obecności lub nieobecności harmonik. Podstawowa składowa zwykle ma największą intensywność w sygnale, a intensywności harmonik zmniejszają się proporcjonalnie do wzrostu częstotliwości. Fala sinusoidalna nazywana czystym dźwiękiem. Czysty dźwięk zawiera tylko jedną składową.



Rysunek 2.7: Koło jednostkowe [10]

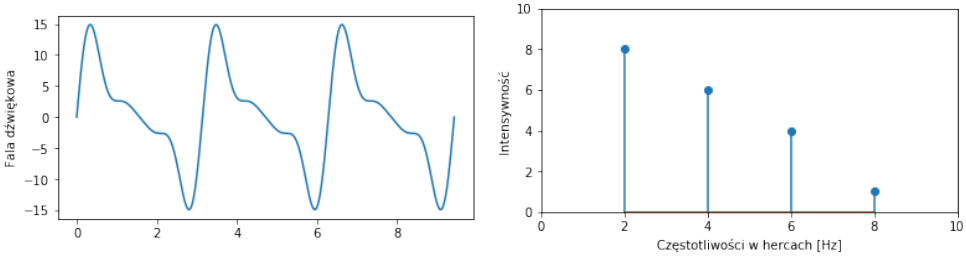
Często przy przetwarzaniu sygnałów występuje również funkcja cosinusa, który może być potraktowany jako sinus z przesunięciem fazowym $\pi/2$:

$$cos(x) = sin(x + \pi/2)$$

Sinus i cosinus są ściśle związane z kołem jednostkowym. Koło jednostkowe (rys. 2.7) jest okręgiem o promieniu 1. Odległości wokół koła są mierzone pod względem fazy. Faza (przesunięcie fazowe) może być traktowana jako kąt φ pomiędzy linią obrotową a osią x i zwykle jest wyrażana w radianach. Dla dowolnego punktu na okręgu jednostkowym współrzędną y punktu jest równa $sin(\varphi)$, a współrzędną x jest równa $cos(\varphi)$.

Sygnał może być reprezentowany na różne sposoby. Do tej pory sygnały występowały w reprezentacji dziedziny czasu. Wykres w dziedzinie czasu pokazuje, jak sygnał zmienia się w czasie, podczas gdy wykres w dziedzinie częstotliwości, pokazuje jakie częstotliwości mieszczą się w sygnale wraz z ich intensywnością.

Na rysunku 2.8a pokazany jest przykład sygnału w dziedzinie czasu. Intensywności dla poszczególnych składowych tego sygnału są zaznaczone na wykresie w dziedzinie częstotliwości (rys. 2.8b). Widać na przykład, że podstawową składową tego sygnału jest częstotliwość 2 o intensywności 8.



(a) Dziedzina czasu

(b) Dziedzina częstotliwości

Rysunek 2.8: Różne reprezentacje sygnału

Reprezentacja w dziedzinie częstotliwości może również zawierać informacje o przesunięciu fazowym, które należy zastosować do każdej sinusoidy, aby móc odtworzyć składowe częstotliwości w celu odzyskania pierwotnego sygnału czasowego.

Transformacje są narzędziami matematycznymi, które służą do zmiany jednej reprezentacji na inną. Transformata Fouriera jest to transformacja, która przekształca reprezentację sygnału w dziedzinie czasu na reprezentację w dziedzinie częstotliwości. Przy przetwarzaniu sygnałów dyskretnych jest stosowana dyskretna transformata Fouriera jako odpowiednik zwykłej transformaty. Dyskretna transformata Fouriera (ang. Discrete Fourier transform, DFT) pobiera sygnał w dziedzinie czasu i tworzy listę liczb zespolonych, które reprezentują stopień wystąpienia różnych częstotliwości w tym sygnale wraz z ich przesunięciem fazowym. Jest ona określona wzorem:

$$DFT[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-\varphi i} \text{ , gdzie } \varphi = k\frac{n}{N}2\pi$$

1. **DFT** jest tablicą **N** liczb zespolonych.
2. **k** jest częstotliwością dla której jest przeprowadzane badanie obecności w sygnale.
3. **x** jest dowolnym sygnałem w dziedzinie czasu o długości N próbek.
4. $\sum_{n=0}^{N-1} x[n] \cdot y[n]$ jest iloczynem skalarnym wektorów x i y. Jest on wykorzystywany do badania korelacji pomiędzy sygnałem wejściowym a sinusoidą o częstotliwości **k**.
5. $e^{-\varphi i}$ jest sposobem opisania pary fal sinusoidalnych i cosinusoidalnych.

$$e^{\varphi i} = cos(\varphi) + sin(\varphi)i \tag{2.1}$$

Przyjmując ze wzoru 2.1 (wzór Eulera) reprezentację kartezjańską, dyskretną transformatę Fouriera można przedstawić następująco:

$$DFT[k] = \sum_{n=0}^{N-1} x[n] \cdot (cos(\varphi) - sin(\varphi)i) \text{ , gdzie } \varphi = k\frac{n}{N}2\pi \tag{2.2}$$

We wzorze 2.2 lepiej jest widoczny związek DFT z sinusem i cosinusem. W transformacie Fouriera jest wykorzystywane wykrywanie korelacji pomiędzy sygnałem wejściowym a falą sinusoidalną i cosinusoidalną o różnych częstotliwościach, w celu wykrycia składowych częstotliwości tego sygnału. Transformata zwraca liczby zespolone, wartości których są stałe bez względu na przesunięcie fazowe przychodzącego sygnału.

Liczby zespolone otrzymane w wyniku zastosowania transformaty Fouriera są nazywane binami. Do każdego bina jest przypisana mu częstotliwość według wzoru:

$$k \cdot \frac{S}{N},$$

gdzie k - numer porządkowy bina, N - długość (w próbkach) wejściowego sygnału, S - częstotliwość próbkowania.

W każdym binie najbardziej istotnym jest jego moduł nazywany magnitudą.

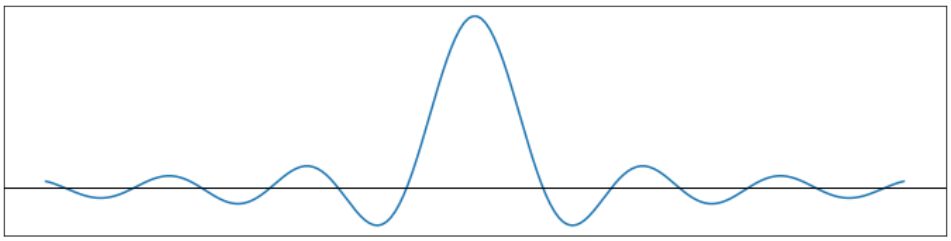
$$||z|| = \sqrt{a^2 + b^2} \text{ ,}$$

gdzie a - część rzeczywista, b - część urojona.

Magnituda bina wykazuje stopień wystąpienia jego częstotliwości w sygnale, ponieważ jest ona równa amplitudzie tej częstotliwości. Biny powyżej częstotliwości Nyquista warto omijać ze względu na aliasing. Biny są równomiernie rozłożone na przedziale od 0 Hz do częstotliwości próbkowania. Na przykład, przy częstotliwości próbkowania $S = 10$ Hz w sygnale o długości $N = 4$ ciągiem częstotliwości dla DFT jest wektor (0 Hz, 2.5 Hz, 5 Hz, 7.5 Hz), a przy $N = 8$ jest to wektor (0 Hz, 1.25 Hz, 2.5 Hz, 3.75 Hz, 5 Hz, 6.25 Hz, 7.5 Hz, 8.75 Hz). Środkowa częstotliwość 5 Hz (częstotliwoscć Nyquista) nie jest zależna od N. Stąd dokładność DFT jest proporcjonalna do długości sygnału wejściowego.

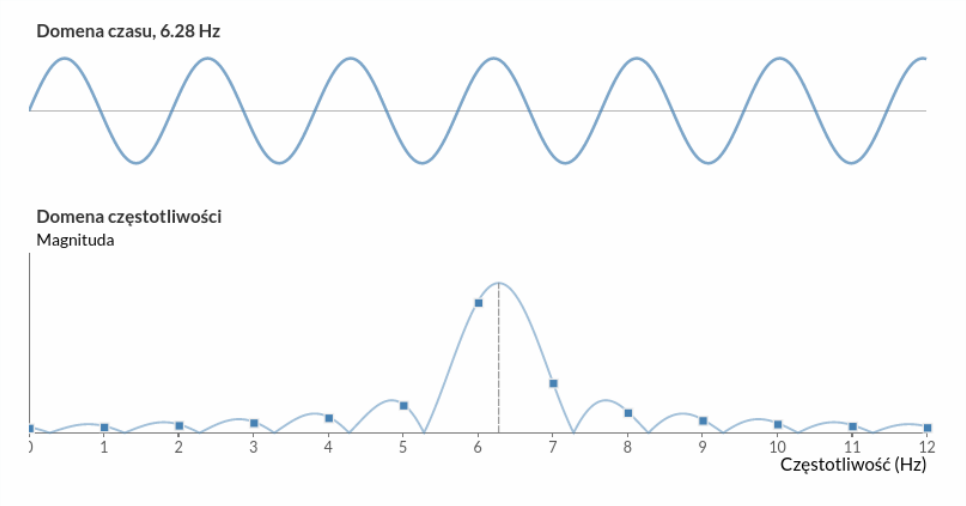
Gdy się okazuje, że jakaś składowa sygnału wejściowego nie pasuje idealnie do żadnej z dostępnych częstotliwości binów, energia związana z częstotliwością tego składnika "rozlewa się" na wszystkie pozostałe biny. Jest to zjawisko znane jako wyciek widmowy. Widmo sygnału jest sposobem na przedstawienie częściowo nieskończenie powtarzającego się sygnału jako sumy poszczególnych częstotliwości. Częściowa nieskończoność tutaj oznacza, że dla dowolnego sygnału w dziedzinie czasu zakłada się, że będzie on zapętlony w nieskończoność w formie, w której on występuje. Ciągłe widmo częstotliwości próbkowanej fali sinusoidalnej jest aproksymowane funkcją sinc (rys. 2.9) określaną następującem wzorem:

$$sinc(x) = sin(x)/x$$



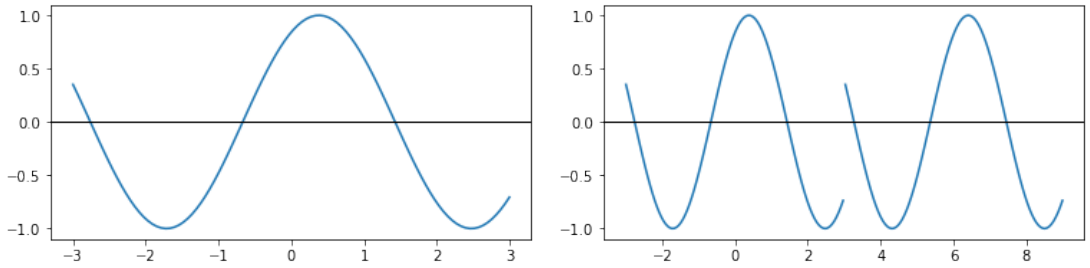
Rysunek 2.9: Funkcja sinc

Dyskretna transformata Fouriera wytwarza próbkowaną wersję ciągłego widma. Dopóki składowe częstotliwości sygnału wejściowego są równe dostępnym częstotliwościom w binach, moduł (magnituda) dla każdego z binów będzie mieć albo zerową wartość, abo będzie równy jednemu z lokalnych maximum krzywej widma. Gdy tylko częstotliwość sygnału wejściowego odchodzi od dostępnych częstotliwości w binach, pozostałe biny przyjmują wartość niezerową ze względu na kształt krzywej ciągłej. Dla przykładu, na rys. 2.10 jest analizowany "czysty" dźwięk z falą sinusoidalną o częstotliwości $f = 6.28$Hz. Przy częstotliwości próbkowania $S = 13$ oraz długości sygnału $N = 13$ częstotliwości przypisywane do binów są liczbami całkowitymi 0,1,...,12 herców. Dostępne częstotliwości nie zawierają wartości f, więc dochodzi do wycieku widmowego.



Rysunek 2.10: Wyciek widmowy [8]

Jedną z przyczyn wycieku są nieciągłości na końcach sygnału. Jak już wspomniano wyżej, zakłada się, że sygnał powtarza swój kształt w nieskończoność. Dla przykładu, na rys. 2.11a jest podany przykład sygnału, który przy powtarzaniu da nieciągłą przerywaną krzywą (rys. 2.11b).

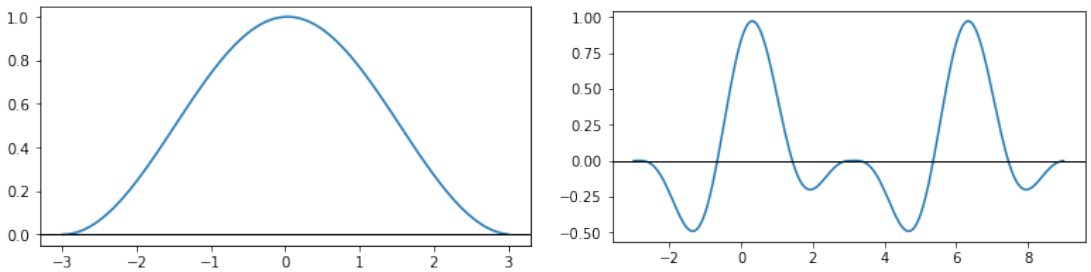


(a) Sygnał wejściowy

(b) Zapętlony sygnał wejściowy

Rysunek 2.11: Przykład nieciągłego sygnału

Aby usunąć nieciągłości, można zastosować okienkowanie do sygnału. Przy okienkowaniu sygnał wejściowy jest mnożony przez sygnał "okna". Po wykonaniu takiej operacji sygnał wejściowy zwęża się do zera na jego końcach, co pozwala usunąć jego nieciągłą okresowość. Popularną funkcją okna przy przetwarzaniu sygnałów jest funkcja Hanna (rys. 2.12a). Należy jednak z tą operacją uważać, ponieważ okienkowanie jest bardzo niszczące dla niskich częstotliwości. Widać, że zokienkowany sygnał zapętla się płynnie, bez rozrywów i przełomów(rys. 2.12b). Pomaga to zmniejszyć wyciek widmowy.



(a) Funkcja Hanna

(b) Zapętlony zokienkowany sygnał wejściowy

Rysunek 2.12: Usuwanie nieciągłości za pomocą okienkowania

Żeby sygnał dźwiękowy z rys. 2.6b "rozbić" na jego składowe z rysunku 2.6a, wystarczy nagrać go z częstotliwością próbkowania $S = 8$ kHz i przeprowadzić na nim transformatę Fouriera, przy założeniu,

że najwyższa składowa sygnału ma częstotliwość niższą od 4 kHz. W wyniku transformaty otrzymamy 8 binów. Biny nr 2, 3, 5 oraz 6 będą mieć niezerowe moduły. Co więcej biny 2 i 6 będą identyczne zarówno jak i biny 3 i 5 (aliasing). Biny powyżej częstotliwości Nyquista są odrzucane. Zatem składowymi sygnału wejściowego są częstotliwości 2 kHz oraz 3 kHz.

2.2.2 Tworzenie spektrogramów

Spektrogram jest podstawowym narzędziem do analizy muzyki. Spektrogram jest wykresem natężenia dźwięku od czasu i częstotliwości. Tworzy się z wykorzystaniem transformaty Fouriera.

Sygnał audio w muzyce jest wysoce niestacjonarny, zmienia się w czasie, więc w celu uproszczenia zakłada się, że w krótkich skalach czasowych sygnał audio statystycznie niewiele się zmienia. Dlatego transformata Fouriera powinna się wykonywać dla ramek sygnału o długości 50-75 ms . Jeśli ramka jest znacznie krótsza, nie ma wystarczającej liczby próbek, aby uzyskać wiarygodne oszacowanie spektralne, jeśli jest ona dłuższa, sygnał zmienia się zbyt mocno w całej ramce.

Krótkotrwała transformata Fouriera (ang. Short Time Fourier Transform, STFT) jest uzyskiwana przez obliczenie transformaty Fouriera dla kolejnych ramek w sygnale. Ramki są częściami sygnału o stałej długości m. Każda taka ramka jest tworzona poprzez przesunięcie indeksu poprzedniej ramki o w próbek. Miejsce, czyli indeks próbki w sygnale, w którym się zaczyna ramka j jest równy $j * w$. Jak można zauważyć, gdy $w < m$, to ramki nakładają się na siebie. Nałożenie jest bardzo przydatne przy przetwarzaniu tego sygnału, ponieważ ze względu na przesunięcie fazowe daje to większą szansę na wykrycie właściwych składowych częstotliwości w miejscach brzegowych ramek.

W tej pracy został zastosowany spektrogram w skali melowej, ponieważ uwzględnia on specyfikę odbierania dźwięku poprzez ludzkie ucho. Często różnicę między dwiema blisko rozmieszczonymi częstotliwościami bardzo trudno zauważyć. Efekt ten staje się bardziej wyraźny wraz ze wzrostem częstotliwości. Z tego powodu amplitudy uzyskane przez STFT dla skupiska przedziałów częstotliwości są sumowane, aby zorientować się, ile energii istnieje w różnych regionach częstotliwości. Taka akcja jest wykonywana przy użyciu banku filtrów melowych.

Pierwszy taki filtr jest bardzo wąski i wskazuje, ile energii istnieje w pobliżu 0 herców. Wraz ze wzrostem częstotliwości filtry stają się coraz szersze, ponieważ ludzkie ucho coraz słabiej postrzega zmiany między kolejnymi częstotliwościami. W ten sposób jest wyliczana ilość energii występująca w różnych zakresach częstotliwości. Wartości z transformaty Fouriera są przekształcane na jednostki decybelowe. Ta operacja sprawia że wyniki są ściślej dopasowane do tego, co faktycznie słyszą ludzie.

Skala melowa wiąże postrzeganą częstotliwość lub wysokość czystego tonu z jego rzeczywistą zmierzoną częstotliwością. Zależność pomiędzy skalą melową, a częstotliwością w heracach określa się wzorem :

$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right) ,$$

gdzie m jest wartością w skali melowej, f jest wartością w hercach [11].

Żeby przejść z powrotem do częstotliwości w hercach stosoje się wzór:

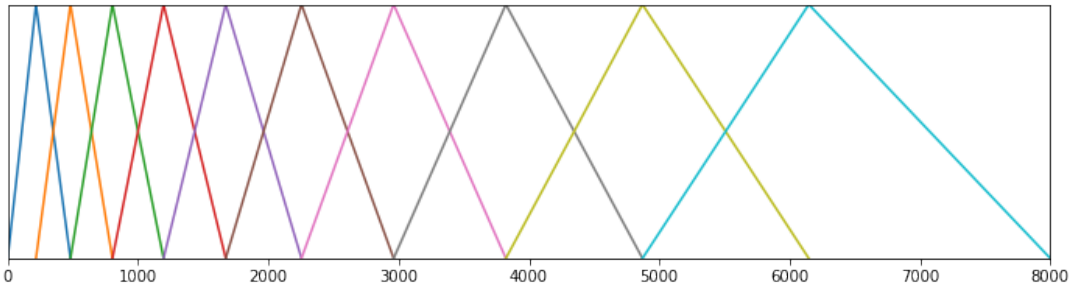
$$f = 700(10^{\frac{m}{2595}} - 1) .$$

Proces obliczenia filtrów melowych wygląda następująco. Pasmo melowe (zakres od 0 do częstotliwości Nyquista w melach) jest dzielone na równe odcinki, nałożone na siebie w połowie, dla których są zapamiętywane wartości dolnej i górnej granicy oraz środka. Dla zapamiętanych wartości jest wyliczana ich wartość w hercach. Zatem jest wykonywane zaokrąglenie otrzymanych częstotliwości do najbliższych dostępnych częstotliwości w binach SFTF poprzez pomnożenie ich przez iloraz długości ramki SFTF i częstotliwości próbkowania.

Wartość j-tego filtru melowego dla k-tej amplitudy można również obliczyć za pomocą wzoru:

$$H_k(j) = \begin{cases} 0 & \text{dla } k < k_d(j) , \\ \frac{k-k_d(j)}{k_c(j)-k_d(j)} & \text{dla } k_d(j) \leqslant k < k_c(j) , \\ \frac{k-k_g(j)}{k_c(j)-k_g(j)} & \text{dla } k_c(j) \leqslant k < k_g(j) , \\ 0 & \text{dla } k_g(j) \leqslant k , \end{cases}$$

gdzie $k_d(j)$, $k_c(j)$ i $k_g(j)$ są odpowiednio indeksami amplitud w SFTF dla dolnej granicy, środka i górnej granicy j-tego filtru. Na rysunku 2.13 jest podany przykład 10 filtrów melowych.



Rysunek 2.13: Bank 10 filtrów melowych

Przeliczenie widma amplitudowego ze skali liniowej (w hercach) na skalę melową dla pojedynczej ramki jest wykonywane według wzoru:

$$A(j) = \sum_{k=0}^{K} |X_k| \cdot H_k(j)$$

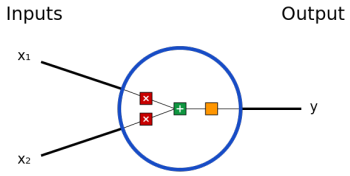
gdzie $A(j)$ jest amplitudą dla j-tego filtru w skali melowej, X_k jest binem SFTF z przypisanej mu k-tej częstotliwości, a $G_k(j)$ jest odpowiednim filtrem melowym. Spektrogram końcowy jest listą takich przeliczonych widm o długości ilości ramek.

Struktura modelu sieci neuronowej

Wykorzystanie sztucznej inteligencji w różnych badaniach oraz dziedzinach nauki znacznie wzrosło w ciągu ostatnich kilku lat. Algorytmy uczenia się maszynowego co roku zbliżają możliwości urządzeń komputerowych do ludzkich. Wizja komputerowa (ang. Computer Vision) jest jedną z wielu dziedzin, w których algorytmy tego typu są szeroko stosowane. Celem tej dziedziny jest umożliwienie postrzegania świata przez maszyny w sposób podobny do ludzkiego. Wizja komputerowa znajduje zastosowania w różnych rodzajach problemów, takich jak rozpoznawanie, analiza i klasyfikacja obrazów i filmów, odtwarzanie mediów, przetwarzanie języka naturalnego. Postępy w dziedzinie wizji komputerowej zostały opracowane i udoskonalone z czasem, przede wszystkim w oparciu o jeden szczególny algorytm - konwolucyjną sieć neuronową. Najlepszą [12] implementacją tego algorytmu dla klasyfikacji obrazów na dzień dzisiejszy przyjęto uważać architekturę ResNet [13].

3.1 Sieci neuronowe typu 'feedforward'

Głębokie uczenie się (ang. Deep Larning) to nazwa, która jest używana w odniesieniu do sieci złożonych z kilku warstw. Warstwy są zbudowane z węzłów. Węzłem nazywa się miejsce, w którym odbywa się obliczenie. Taki węzeł jest ogólnikowo wzorowany na neuronie w ludzkim mózgu, który jest aktywowany wystarczającym bodźcem. Węzeł łączy dane wejściowe z zestawem współczynników nazywanych wagami, które albo tłumią, albo wzmacniają wartości wejścia, przypisując w ten sposób znaczenie wejścia dla wykonania określonego zadania, którego algorytm próbuje się nauczyć. W tej pracy takim zadaniem jest klasyfikacja obrazu, czyli przypisywanie jednej klasy ze zbioru dla każdego wejściowego obrazu.



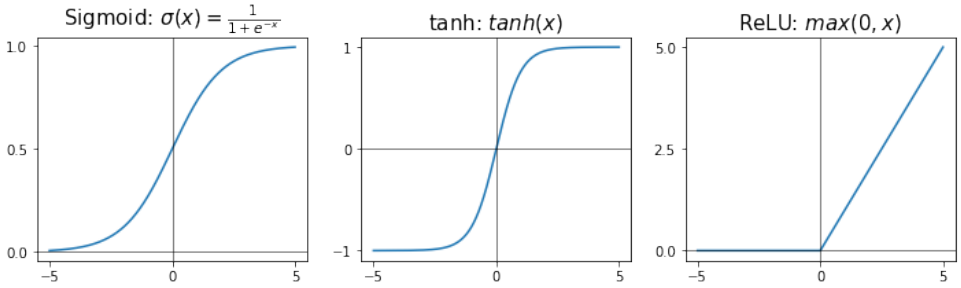
Rysunek 3.1: Węzeł sieci neuronowej [14]

W sieciach neuronowych typu 'feedforward' takie łączenie jest osiągane poprzez mnożenie wag z danymi wejściowymi. Iloczyny otrzymane po przemnożeniu są sumowane, a następnie suma jest przekazywana do tak zwanej funkcji aktywacji węzła, w celu ustalenia, czy i do jakiego stopnia sygnał powinien przejść dalej przez sieć, aby wpłynąć na ostateczny wynik (rys. 3.1).

Wybór funkcji aktywacji różni się w zależności od rodzaju problemu. Najpopularniejszymi funkcjami aktywacji są tanh, sigmoid i ReLU (rys. 3.2). Jednak ReLU oraz jej wariacje są uważane przez praktyków za najbardziej skuteczne funkcje aktywacji. W ReLU negatywne wartości są tłumione poprzez ich zerowanie, przez co nie mają one wpływu na dalszą analizę, a wartości pozytywne wpływają na wynik końcowy w sposób liniowy.

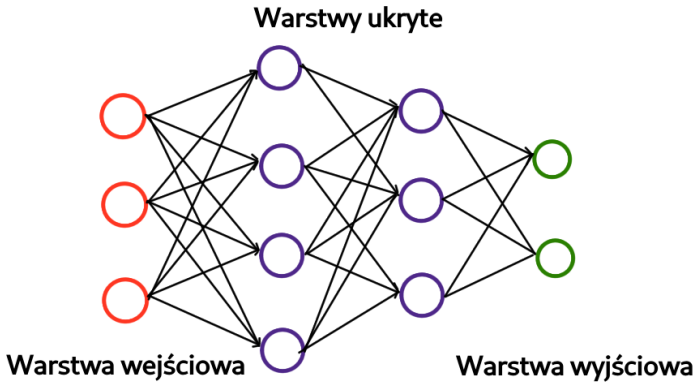
Kombinacja mnożenia macierzy wag i przeznaczonej dla niej warstwy, a następnie wykonywanie ReLU na wyniku mnożenia, ma matematyczną właściwość o nazwie Universal Approximation Theorem [15]. Mówi ona, że jeśli macierze z wagami są wystarczająco duże, i ich, tych macierzy, jest wystarczająca liczba, to taka sieć może rozwiązać dowolną ciągłą funkcję matematyczną z dowolnym poziomem dokładności.

Czasami są przydatne funkcje aktywacji o określonych właściwościach. Tak na przykład funkcja SoftMax ma taką właściwość, że dla wektora wyjściowego każda jego wartość znajduje się w zakresie [0,1] oraz suma jego wartości jest równa 1.



Rysunek 3.2: Funkcje aktywacji

Wektor danych wejściowych jest nazywany warstwą wejściową. Warstwa wyjściowa zawiera wyliczone informacje dotyczące danych w kontekście określonego zadania. W przypadku problemu klasyfikacji każdy węzeł w warstwie wyjściowej będzie odpowiadał prawdopodobieństwu, z którym dane wejściowe należą do odpowiedniej klasy. Ze względu na właściwości SoftMax, ta funkcja aktywacji jest często stosowana przy obliczeniu warstwy wyjściowej. Warstwy pomiędzy wejściową i wyjściową nazywane są warstwami ukrytymi. Wyjście każdej ukrytej warstwy jest jednocześnie wejściem kolejnej warstwy. Na rysunku 3.3 jest podany przykład sieci neuronowej takiego typu. Do każdego połączenia w tym rysunku jest przypisana wartość wagi.



Rysunek 3.3: Struktura sieci neuronowej typu 'feedforward'

W sieciach głębokiego uczenia się każda warstwa węzłów trenuje odrębny zestaw cech opartych na wynikach poprzedniej warstwy. Im głębiej w sieć neuronową, tym bardziej złożone cechy mogą rozpoznać warstwy. Sieci głębokiego uczenia mogą obsługiwać bardzo duże, wielowymiarowe zestawy. Przede wszystkim są one w stanie wykryć zależności w nie strukturyzowanych danych takich jak zdjęcia, wideo i audio. Wykonują one automatyczne wyodrębnianie cech w procesie uczenia się bez interwencji człowieka, w przeciwieństwie do większości tradycyjnych algorytmów uczenia maszynowego.

3.2 Konwolucyjne sieci neuronowe

W sieciach neuronowych typu 'feedforward' szkolenie sieci wielowarstwowej wymaga podania danych wejściowych jako wektora (macierz jednowymiarowa). W przypadku, gdy dane wejściowe są obrazami, macierz obrazu należy spłaszczyć, aby uzyskać kształt wektora. Jednak na naturalnych obrazach istnieją bardzo silne powiązania między sąsiednimi pikselami. Przy spłaszczeniu obrazu te informacje tracą się, sieci jest trudniej interpretować dane podczas fazy trenowania. Dodatkowo, gdy obrazy osiągają bardzo duże wymiary (miliony pikseli w obrazie) intensywność obliczenia w sieciach 'feedforward' byłaby bardzo wysoka. Uzasadnia to zainteresowanie inną architekturą sieci neuronowej w celu rozwiązania tego rodzaju problemu.

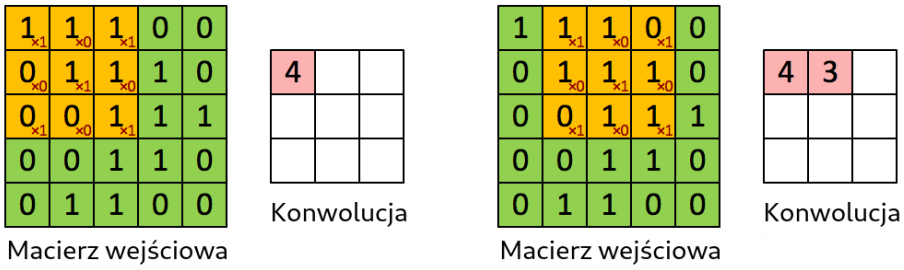
Konwolucyjne sieci neuronowe (ang. Convolutional neural networks, CNN) są w stanie skutecznie uchwycić zależności przestrzenne na obrazie poprzez zastosowanie odpowiednich filtrów. Podstawową operacją dla warstwy konwolucyjnej jest splot. Splot (konwolucja) jest operacją mnożenia filtru z odpo-

wiednim kawałkiem macierzy względem elementów, aby wydobyć z niego pewne cechy określone przez ten filtr. Suma tych iloczynów jest przekazywana dalej po sieci w celu dalszej analizy.

Na rysunku 3.4 dla uproszczenia są pokazane dwa pierwsze kroki w warstwie splotu dla obrazu jednokanałowego o rozmiarze 5×5. W tym przypadku filtr splotu będzie macierzą rozmiaru 3×3:

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}.$$

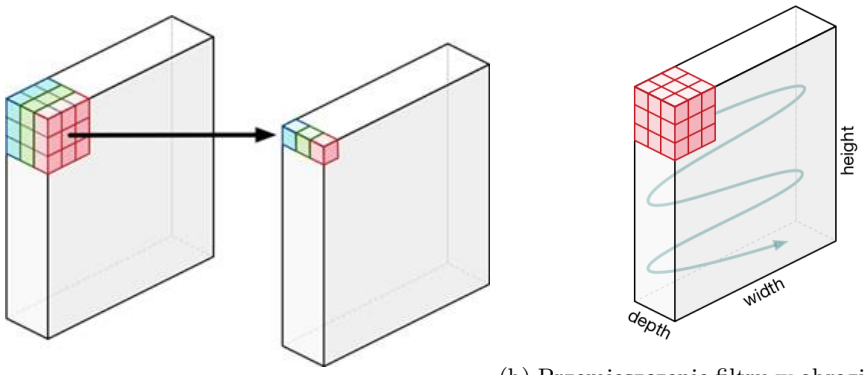
Na początku filtr splotu jest umieszczony w lewym górnym rogu obrazu pokrywając tym samym jego część o wymiarach filtru. Następnie wartości obrazu są przemnażane (pod względem elementu) przez wartości filtru. Ostatecznie te iloczyny są sumowane, a końcowy wynik odpowiada pikselowi obrazu wyjściowego (konwolucji). Po wykonaniu splotu filtr jest przemieszczany o ustaloną stałą wartość nazywaną *długością kroku* w prawo oraz operacja splotu przebiega podobnie. Po dojściu do prawej krawędzi obrazu, filtr wraca do lewej krawędzi oraz przesuwa się o długość kroku w dół. Przy takim przejściu kolejne wartości splotów będą umieszczane w kolejnym rzędzie pikseli nowego obrazu. Proces się powtarza, aż cały obraz zostanie przefiltrowany. W przypadku podanym na rysunku 3.4, długość kroku wynosi 1 oraz konwolucja ma wymiary 3×3.



(a) Splot na pierwszym obszarze (b) Splot na drugim obszarze

Rysunek 3.4: Operacja splotu dla obrazu jednokanałowego [16]

Przy wykonaniu splotu na obrazie trójkanałowym (np. RGB) filtr też jest trójkanałowym. Dla każdego kanału rozważanego obrazu (np. jeśli obraz jest w reprezentacji RGB, to jest wyróżniany czerwony, zielony oraz niebieski kanał) filtrowanie odbywa się oddzielnie, każdemu kanałowi jest przypisany osobny filtr (rys. 3.5a). Często wartości w konwolucji są sumowane względem kanału żeby otrzymać spłaszczoną macierz o głębokości 1.



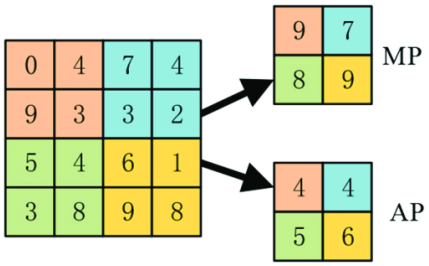
(a) Splot dla obrazu RGB [17] (b) Przemieszczenie filtru w obrazie trójkanałowym [16]

Rysunek 3.5: Warstwa splotu dla obrazu trójkanałowego

Po warstwie konwolucyjnej może dodatkowo występować warstwa poolingu. Warstwa poolingu również jest odpowiedzialna za zmniejszenie przestrzennego rozmiaru konwolucji oraz, podobnie jak warstwa splotu, służy do redukcji złożoności obliczenia przy przetwarzaniu danych poprzez redukcję wymiarowości. Dodatkowo warstwa poolingu służy do znalezienia dominujących cech, które są niezależne od pozycji lub kąta obrotu obrazu, co daje możliwość wytrenowania skuteczniejszego modelu.

Wyróżnia się dwa rodzaje poolingu: maksymalny (ang. max pooling) i uśredniony (ang. average pooling). Przy poolingu jest wybierany rozmiar jądra (ang. kernel) oraz przemieszczenie po obrazie jest takie samo jak w warstwie splotu (rys. 3.5b). Przy maksymalnym poolingu z części obrazu objętej przez jądro do następnej warstwy jest przenoszona maksymalna wartość. Przy uśrednionym poolingu z części obrazu objętej przez jądro jest przenoszona średnia tych wartości. Oba rodzaje poolingu, oprócz tego, że redukują wymiar konwolucji, na różne sposoby radzą sobie z zaszumieniem danych - zachowują one najbardziej znaczące cechy obrazu wejściowego. Na przykład, maksymalny pooling daje możliwość na wybieranie z obszaru najistotniejszych cech oraz odrzucenie pozostałych, traktując ich jako szum. Przy uśrednionym poolingu odbywa się zagładzenie, czyli uśrednienie takich szumów.

Na rysunku 3.6 są podane przykłady wykonania poolingu maksymalnego oraz uśrednionego.

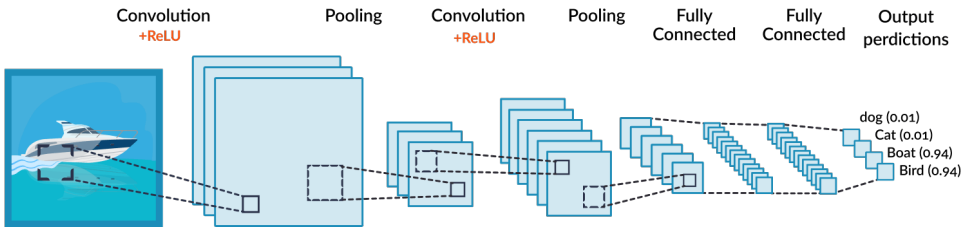


Rysunek 3.6: Przykład wykonania maksymalnego poolingu (MP) oraz uśrednionego poolingu (AP) [18]

Celem warstwy CNN jest wyodrębnienie z obrazu wejściowego cech różnego poziomu. CNN często zawierają wiele warstw oraz przy przejściu do kolejnej warstwy jest wykonywana funkcja aktywacji. Tradycyjnie pierwsza warstwa jest odpowiedzialna za przechwytywanie cech niskiego poziomu, takich jak krawędzie, kolor, gradient itp. Dzięki kolejnym warstwom model dostosowuje się również do cech wysokiego poziomu. Przykładowo druga warstwa CNN może rozpoznać wykryte w poprzedniej warstwie krawędzie połączone w proste figury takie jak kwadraty, trójkąty itp.; trzecia warstwa może wykrywać skupienie prostych figur, z których się składają proste wzorce. Daje to sieć, która ‚widzi' obiekty w sposób podobny do ludzkiego [19].

Kolejną rolą CNN jest redukowanie obrazów do postaci, która jest łatwiejsza do przetworzenia, bez utraty cech, które są niezbędne do poprawnej klasyfikacji obrazu. Jest to szczególnie istotne przy dużych ilościach danych, ponieważ proces uczenia się jest dużo szybszy. Jeżeli jednak przy przejściu między kolejnymi warstwami CNN jest potrzebnym zachowanie wejściowych rozmiarów konwolucji, do niej jest dodawana ramka marginesowa, która zwykle jest wypełniona zerami.

Im więcej jest takich warstw, tym wyższy poziom cech jest wyróżniany. Pierwsze warstwy CNN są intuicyjnie interpretowane, ponieważ są w nich wyróżniane cechy pojawiające się w większości obrazów, ale im głębsza warstwa jest interpretowana, tym bardziej indywidualnym jest sposób ekstrakcji istotnych cech. Na rysunku 3.7 został pokazany przykład sieci konwolucyjnej. Jak widać redukują się szerokość i wysokość obrazu wejściowego. Natomiast głębokość wyjściowej konwolucji zwiększa się z każdą warstwą, ponieważ operacje poolingu i splotu są wykonywane dla kilku filtrów oraz wyniki są przedstawiane jako kolejne kanały obrazu wyjściowego. CNN spłaszcza obraz do wektora wartości zawierającego informacje o wysokopoziomowych cechach obrazu. W tym momencie zachowanie sieci CNN się zmienia i, w celu klasyfikacji, są dodawane dodatkowa warstwa spłaszczenia, która łączy się z każdą wartością tego wektora w sposób jak przy sieciach typu 'feedforward'. Typ takiej warstwy jest nazywany Fully Connected (FC).



Rysunek 3.7: Model CNN [20]

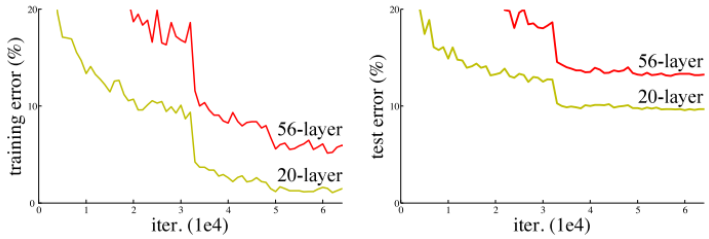
Innymi słowami, wyjście ostatniej warstwy konwolucyjnej jest warstwą wejściową w sieci typu 'feedforward' z jedną warstwą ukrytą. Dodanie warstwy FC jest efektywnym sposobem uczenia się nielinio-

wych kombinacji cech wysokiego poziomu, reprezentowanych przez wynik warstw konwolucyjnych. Wynik warstw FC jest wynikiem ostatecznym CNN.

3.3 Architektura ResNet

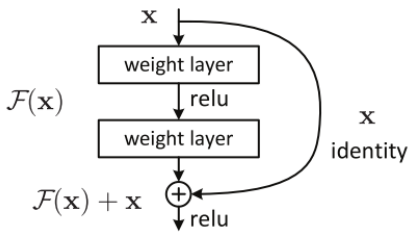
Pojawienie się architektury konwolucyjnej sieci neuronowej ResNet[13] (Residual Network) było jedną z najbardziej przełomowych zmian w dziedzinie wizji komputerowej oraz głębokiego uczenia się w ciągu ostatnich kilku lat. Istnieją różne jej implementacje takie jak ResNet-18, ResNet-34 oraz ResNet-50, gdzie liczby w nazwach odpowiadają ilości warstw w sieci.

W pracy naukowej poświęconej ResNet[13] był zbadany ciekawy efekt niskiej skuteczności dla CNN z dużą ilością warstw. Porównano błąd trenowania dla CNN mającej 20 warstw oraz CNN mającej 56 warstw w której są wykonywane tylko podstawowe sploty z filtrami 3×3 (rys. 3.8). Trenowanie odbywało się na zbiorze CIFAR-10 [21]. 56 warstwowa sieć ma znacznie więcej parametrów, powinno to więc dać lepsze dopasowanie niż w przypadku mniejszej ilości warstw. Można się więc spodziewać, że 56-cio warstwowa sieć szybko sprowadzi błąd trenowania do zera. Jednak, jak się okazuje, sieć ta daje gorsze wyniki, niż sieć 20-to warstwowa.



Rysunek 3.8: Porównanie błędów w zbiorze treningowym oraz testowym dla sieci o różnych głębokościach [13]

W architekturze ResNet problem ten był rozwiązany poprzez wprowadzenie funkcji identyczności jako alternatywnego przejścia pomiędzy grupami warstw konwolucyjnych (rys 3.9). Jeśli w procesie uczenia się okaże się, że kolejna grupa warstw pogarsza wynik końcowy, sieć ma możliwość ominięcia takiej grupy. W związku z czym dodanie kolejnych warstw konwolucyjnych nie będzie pogarszać wyniku końcowego - zawsze można ominąć.



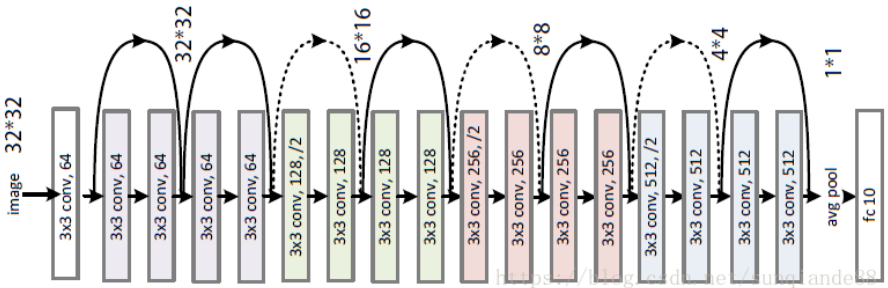
Rysunek 3.9: Blok architektury ResNet [13]

Niech dwie warstwy konwolucyjne będą nazywane blokiem. Niech x będzie wejściem do tego bloku. Zmiany x zachodzące w bloku są określone jako funkcja $F(x)$ (Residual function). W bloku ResNet jednak zamiast określić wyjście jako $x' = F(x)$ określa się je jako $x' = F(x) + x$. Przy takim podejściu w najgorszym scenariuszu 56-cio warswowa sieć będzie przynajmniej tak dobra, jak 20-to warstwowa. Zawsze można wyzerować $F(x)$ poprzez zerowanie wszystkich wag w warstwach konwolucyjnych występujących po 20-tej warstwie, tak że wynik pozostaje bez zmian. W dobrym scenariuszu głębsze sieci lepiej przybliżają odwzorowanie i znacznie zmniejszają błąd.

Takie ominięcie warstw nazywane jest połączeniem tożsamości. Oczywiście przejście po tym połączeniu odbywa się jeśli podwyższa to skuteczność modelu. To zrobiło rewolucję w świecie głębokiego uczenia się, a architektura ResNet-50 wygrała konkurs ImageNet 2015 [22] przez co sprawiedliwie jest nazywana najlepszą architekturą sieci konwolucyjnych na dzień dzisiejszy.

W artykule [23] zbadano przestrzeń funkcyjną funkcji kosztu przy użyciu bloku ResNet oraz bez jego użycia. Wykryto, że bez połączeń tożsamości przestrzeń funkcyjna jest bardzo nierówna, przez co jest większe ryzyko zatrzymania się w lokalnych minimumach.

W tej pracy uznano, że architektura ResNet-18 (rys. 3.10) ma wystarczającą głębokość dla klasyfikacji spektrogramów o małych rozmiarach.



Rysunek 3.10: ResNet-18 [24]

Inne cechy ResNet:

(a) Używanie głównie filtrów rozmiaru 3×3 ,

(b) Dwukrotne zmniejszenie wymiarów konwolucji za pomocą warstw CNN z długością kroku 2 ,

(c) Warstwy globalnego uśrednionego poolingu ,

(d) Warstwa FC z o długości 1000 ,

(e) SoftMax na warstwie wyjściowej .

ResNet zachowuje szerokości i wysokości wyjścia warstwy CNN korzystając ze sposobu dodawania ramki marginesowej. Głębokość jednak jest zwiększana. W odpowiednim momencie długość kroku jest ustawiana na wartość 2, co powoduje dwukrotne zmniejszenie wysokości i szerokości konwolucji.

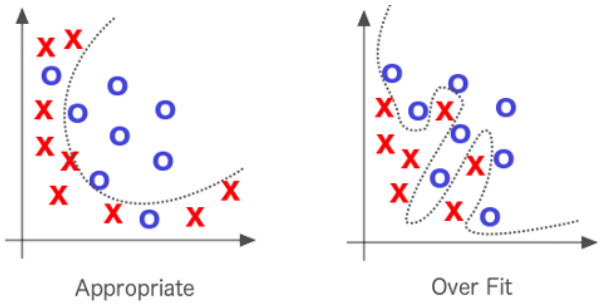
3.4 Specyfiki procesu uczenia się z wykorzystaniem biblioteki fastai

Fastai [25] jest biblioteką, która ma na celu pomóc programistom w realizacji zadań związanych ze sztuczną inteligencją za pomocą głębokiego uczenia. Dokładniej bibliotekę tę opisano w rozdziale 5.

Przepływ uczenia się sieci neuronowej wygląda następująco. Po wykonaniu iteracji klasyfikacji skuteczność modelu jest mierzona za pomocą funkcji kosztu (najpopularniejszą jest średnio kwadratowa funkcja kosztu (ang. MSE)). Następnie jest wykonywana propagacja wsteczna (ang. back propagation), która konfiguruje odpowiednie wagi w oparciu o gradient. O szybkości zmiany parametrów decyduje współczynnik uczenia się (ang. learning rate, lr). Trenowanie polega na sprowadzeniu wartości parametrów do takich, które powodują najmniejszy koszt. Takie sprowadzenie można przedstawić jako przemieszczenie po płaszczyźnie funkcji kosztu do lokalnego minimum. Po wystarczającej liczbie iteracji model potrafi rozróżniać dominujące cechy w obrazach co wpływa na skuteczność klasyfikacji.

Pojęcie przetrenowania się (ang. overfitting) oznacza bardzo dobre dopasowanie się do zbioru danych treningowych oraz bardzo zły wynik dla danych testowych. Często powodem overfitting'u jest ogromna ilość parametrów. Więcej parametrów oznacza więcej nieliniowości, co jest przydatnym ponieważ dane życia rzeczywistego często są nieliniowe. Jednak ważnym jest żeby funkcja, która jest dopasowywana do danych, nie była zbyt skomplikowana (wtedy bardzo wiernie odwzoruje dane treningowe, ale będzie słabo dopasowana do danych rzeczywistych). Na rysunku 3.11 podano przykład overfittingu dla klasyfikacji binarnej.

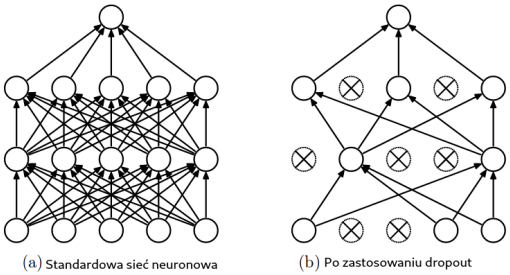
Istnieją różne sposoby „wygładzania" modelu. Jedną z nich jest metoda regularyzacji, która dodatkowo zwiększa wartość funkcji o sumę kwadratów parametrów. Małe parametry powodują bardziej gładki kształt krzywej. Suma kwadratów jest mnożona przez współczynnik λ (przykładowo w bibliotece fastai



Rysunek 3.11: Dobre dopasowanie oraz przetrenowanie [26]

ma domyślną wartość 0.01). Oprócz tradycyjnych sposobów zapobiegania overfittingowi takich jak regularyzacja oraz dodanie zbioru walidacyjnego, biblioteka fastai wykorzystuje metodę 'dropout' po raz pierwszy wspomnianą w [27].

W warstwie FC każdy wierzchołek jest połączony z każdym. Dropout polega na losowym wyrzucaniu pewnego procentu aktywacji warstwy z prawdopodobieństwem p (rys. 3.12). Przy takim podejściu w każdej iteracji są aktywowane różne zestawy węzłów sieci. Oznacza to, że żadna aktywacja nie może zapamiętać jakiejś części danych wejściowych. To właśnie się dzieje przy overfittingu - pewna część modelu w zasadzie uczy się rozpoznawać określony obraz, a nie ogólną cechę właściwą dla tego obrazu. Przy wykorzystaniu metody dropout będzie to bardzo mało prawdopodobne. Dropout stosuje się tylko podczas trenowania modelu. Przy testowaniu tej metody się nie stosuje, ponieważ testowanie ma na celu zmierzyć skuteczność całego modelu. Twórcy metody również sugerują pomnożenie wszystkich wag podczas testowania przez p. Domyślną wartością w bibliotece fastai jest $p = 0.5$.



Rysunek 3.12: Przykład użycia metody dropout z $p = 0.5$ [27]

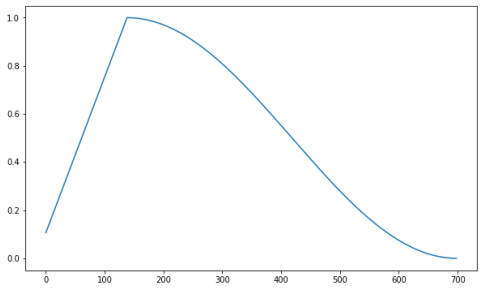
ResNet w fastai jest wytrenowany na zbiorze ImageNet, tak że można wymienić całą macierz FC służącą do klasyfikacji 1000 klas obrazów z ImageNet na inną, która będzie przystosowana do rozwiązania innego problemu klasyfikacji. Zamiast niej biblioteka wprowadza 2 nowe macierze wagowe z ReLU pomiędzy nimi. Istnieją więc pewne wartości domyślne dla rozmiaru pierwszej macierzy, ale w przypadku drugiej macierzy rozmiar jest tak duży, jak tego potrzebuje warstwa wyjściowa.

Podczas trenowania sieci istnieje możliwość częściowego zamrażania sieci. Pojęcie zamrażania sieci oznacza, że aktualizacja parametrów dla warstw konwolucyjnych przy propagacji wstecznej się nie odbywa. ResNet jest wytrenowany na zbiorze ImageNet oraz zakłada się, że pierwsze warstwy, które głównie są odpowiedzialne za cechy niskopoziomowe (wykrywanie krawędzi oraz wzorców geometrycznych) są w mniejszym stopniu zależne od rodzaju klasyfikacji. Zamrażanie odbywa się w celu dobierania sensownych (nie losowych) wag w warstwach FC wraz z zachowaniem dobrze wytrenowanych przydatnych warstw konwolucyjnych.

Po dopasowaniu ostatnich warstw do klasyfikacji specyficznego zbioru danych, część tylnych warstw konwolucyjnych jest 'rozmrażana' i proces uczenia się kontynuuje. Wtedy już zarówno warstwy FC jak i rozmrożone warstwy konwolucyjne dotrenowują się, by dokładniej dopasować się do zadania. Takie rozmrażanie i dotrenowanie możne odbywać się dla kolejnych warstw CNN (najpierw są rozmrażane głębsze warstwy). Ekstrakcja cech w części konwolucyjnej jest polepszana, jednak w większości problemów klasyfikacji w procesie propagacji wstecznej im bliżej początku sieci, tym mniej wartości w filtrach są zmieniane.

Kolejną istotną właściwością procesu uczenia się w fastai jest adaptatywna zmiana współczynnika

uczenia z użyciem metody 'One Cycle Policy' [28]. Epoka jest okresem czasu trenowania CNN w którym sieci jest pokazywany cały zbiór treningowy. 'One Cycle Policy' jest wykonywana dla każdej epoki.



Rysunek 3.13: Wykres zmiany α dla jednej epoki w 'Ony Cycle Policy'

Na wykresie 3.13 jest pokazany wykres wartości współczynnika uczenia się α w obrębie jednej epoki. W bibliotece jest możliwym podanie zakresu α. Tak więc α startując z niskiego poziomu i osiąga swoje maximum w pierwszej ćwiartce epoki, a następnie zmniejsza się przez pozostały czas. Ponieważ na samym początku uczenia się nie wiadomo jak daleko parametry modelu znajdują się od globalnego minimum funkcji kosztu, jest możliwym, że część przestrzeni funkcyjnej na początku epoki będzie bardzo wyboista oraz wielkie skoki na nierównościach z dużym nachyleniem mogą powodować niekontrolowane skoki błędu. Po pewnym czasie parametry sprowadzają się do bardziej gładkiej przestrzeni wag, wtedy można zwiększyć tempo uczenia się, ponieważ gradienty sprowadzają parametry w kierunku minimalnego kosztu. Przy zbliżeniu się do ostatecznej odpowiedzi α znowu się zmniejsza, żeby dokładniej wyznaczyć miejsce globalnego minimum. Podejście w artykule [28] było eksperymentalnym i pokazało ono bardzo dobre wyniki.

Istotnym jest to, że czysta wersja ResNet oczekuje kwadratowego obrazu na wejściu. Jednak fastai wykorzystuje 'adaptatywne' albo 'globalne' warstwy poolingu, które pozwalają pracować z obrazami prostokątnymi.

Implementacja

W celu umożliwienia rozpoznawania akordów w czasie rzeczywistym oraz korzystania z takiego systemu posiadaczom urządzeń mobilnych, została napisana aplikacja mobilna, która ma na celu rozpowszechnienie oraz łatwe korzystanie z klasyfikatora. Aplikacja została podzielona na dwie części - klient oraz serwer. Po stronie klienta odbywa się nagrywanie ścieżek dźwiękowych, wysyłanie ich do serwera oraz zapisywanie otrzymanych wyników do bazy danych. Po stronie serwera odbywa się odbieranie wysłanych ścieżek dźwiękowych, ich przetwarzanie, klasyfikacja na wytrenowanym modelu sieci neuronowej oraz wysyłanie wyniku do klienta.

4.1 Użyte technologie

Jądrem opisanego systemu jest uczenie maszynowe. Język programowania Python stał się niemal synonimem rozwoju sztucznej inteligencji, ponieważ jest szybki, przenośny i skalowalny. Biblioteka `fastai` [25], która swoją drogą jest oparta na `pytorch` [29], zawiera niektóre z najbardziej popularnych algorytmów do klasyfikacji obrazów i zadań w języku naturalnym. W praktyce oznacza to, że modele można tworzyć i uruchamiać w kilku liniach kodu.

Biblioteka `librosa` [30] jest jedną z najbardziej popularnych bibliotek języka Python dla pracy z dźwiękiem. Zawiera ona ważne metody dla przetwarzania dźwięku takie jak odczyt/zapisywanie ścieżek dźwiękowych, zmiana częstotliwości próbkowania oraz przetworzenie ścieżki dźwiękowej do postaci spektrogramu w skali melowej. Do pracy ze ścieżkami jako wektorami przy generowaniu ścieżek została użyta biblioteka `numpy` [31]. Do odczytu plików csv skorzystano z `pandas` [32].

Proces zaprojektowania modelu sieci neuronowej oraz metod przetwarzania ścieżek dźwiękowych wymaga wielorazowego uruchomienia bloków kodu w różnej kolejności. Dla takiego nieliniowego procesu stosuje się narzędzie Jupyter [33], które pozwala na uruchomienie bloków kodu, które współdzielą zmienne, w różnej kolejności. Google Colab [34] jest platformą wzorowaną na Jupyter Notebooks. Dodatkowo Google Colab umożliwia podłączenie dysku Google, gdzie mogą być przechowywane dane, oraz udostępnia bezpłatne korzystanie z potężnej karty graficznej Tesla K80.

Dla pracy z wytrenowaną siecią neuronową serwer również powinien być napisany w języku Python. Django [35] jest popularnym frameworkiem dla aplikacji webowych napisanych w Pythonie.

Android [36] na dzień dzisiejszy jest najpopularniejszym systemem operacyjnym dla urządzeń mobilnych. Stanowi on ponad 75% rynku [37]. Do napisania aplikacji na Androida został użyty język programowania Kotlin [38]. Do automatyzacji kompilacji oraz zarządzania zależnościami zostało użyte narzędzie Gradle [39].

Przy napisaniu aplikacji mobilnej zostały użyte następujące biblioteki. `Retrofit2` [40] oraz `Okhttp3` [40] umożliwiają łączenie po HTTP z serwerem oraz przesyłanie ścieżek dźwiękowych. `MediaRecorder` [41] jest natywnym mechanizmem dla nagrywania oraz zapisywania ścieżek dźwiękowych z podanymi parametrami takimi jak format plików, częstotliwość próbkowania itd. Biblioteki z Android Jetpack [42] takie jak `constraintlayout`, `recyclerview`, `fragment`, `lifecycle`, `room` zostały użyte do zaprojektowania modularnej architektury aplikacji o niskim sprzężeniu (ang. low coupling).

4.2 Model

Model sieci neuronowej został nauczony osobno a wagi wytrenowanej sieci zostały zaimportowane przez serwer.

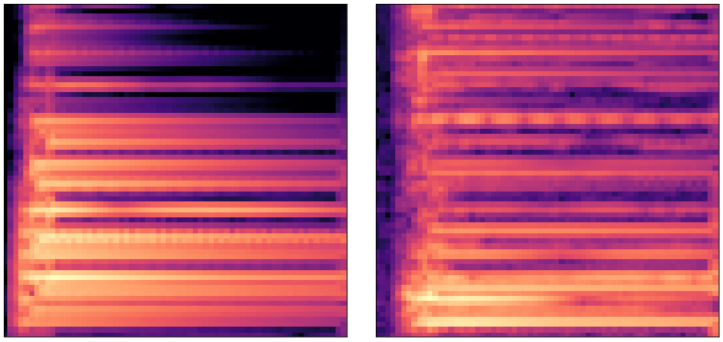
Wygenerowanie 2800 jednosekundowych akordów na Google Colab zajęło około 41 minut.

Częstotliwości podstawowych fali sygnałów produkowanych przez gitarę akustyczną znajdują się w zakresie od 80 Hz do 1200 Hz [43]. Chociaż ich harmoniki mogą znajdować się w zakresie od 2 do 6

kHz, przy trenowaniu częstotliwości powyżej 2 kHz nie polepszały wyniku końcowego, dlatego zakres spektrogramu był ograniczony od 20 Hz do 2 000 Hz.

Ze względu na taki zakres częstotliwości, wszystkie nagrane ścieżki zostały przeskalowane z częstotliwości próbkowania 44.1 kHz do 16 kHz. Przy tworzeniu spektrogramu wykorzystano 64 filtrów melowych, długość ramki wyniosła 62.5 milisekund, a krok przesunięcia ramki wyniósł około 15 milisekund. Z takimi parametrami spektrogram dla jednosekundowej ścieżki dźwiękowej ma rozmiar 64×64 pikseli (rys. 4.1).

Trenowanie modelu dla tych samych meta parametrów (zakres częstotliwości, współczynnik uczenia się itd.) na tym samym zbiorze danych dla modelu ResNet-18 oraz ResNet-34 nie wykazało istotnej różnicy. Architektura ResNet-18 była wybrana jako końcowa ze względu na szybsze analizowanie z powodu mniejszej liczby warstw konwolucyjnych.

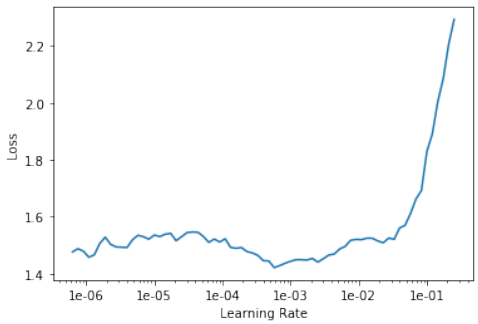


(a) Wygenerowany Am (b) Nagrany Am

Rysunek 4.1: Przykłady spektrogramów

Zbiory danych w `fastai` są przechowywane w obiektach typu `DataBunch`. Interfejs `fastai` pozwala dostosować tworzenie DataBunch, izolując podstawowe części tego procesu w osobnych blokach. Dzięki temu łatwo pogrupować dane na zbiory treningowy oraz walidacyjny według ustalonego podziału (procentowo, na podstawie folderów, na podstawie pliku csv itd).

Uczenie się odbywa za pośrednictwem obiektu typu `learner`, który między innymi przyjmuje takie parametry jak `DataBunch`, typ modelu (np. `resnet18`) oraz metryki. `Learner` posiada metody `freeze()` i `unfreeze()`, którymi można regulować stopień zamrażania modelu. Metoda `fit_one_cycle` przyjmuje jako parametry liczbę epok oraz zakres współczynnika uczenia się α. Jej zadaniem jest aktualizacja parametrów metodą 'One Cycle Policy'. `fastai` zawiera narzędzia do wyliczenia funkcji straty dla różnych α, co pomaga odnaleźć najlepszy współczynnik uczenia się. Na rysunku 4.2 widać, że najmniejszy błąd powodują wartości z zakresu $[5 \cdot 10^{-4}, 5 \cdot 10^{-3}]$.



Rysunek 4.2: Wartość funkcji straty dla różnych α (wykres wygenerowany biblioteką `fastai`)

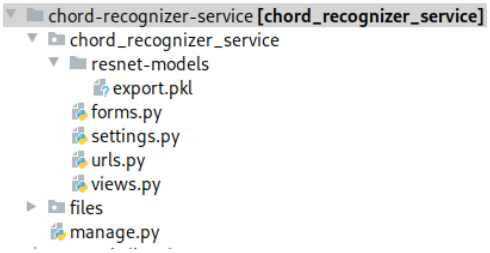
Zbiór treningowy składa się w 75% z danych wygenerowanych oraz w 25% z danych nagranych. Zbiór walidacyjny składa się w 25% z danych nagranych. Zbiór testowy składa się w 50% z danych nagranych.

Dokładność klasyfikacji po dwóch epokach uczenia się na zamrożonym modelu wyniosła ok. 52%; po rozmrożeniu modelu i uczeniu się w ciągu trzech epok dopasowanie do zbioru walidacyjnego wyniosło ok. 99%. Wynik dla zbioru testowego wyniósł ok. 98%.

4.3 Serwer

Modele nowoczesnych sieci neuronowych wymagają znaczących zasobów pamięci oraz mocy obliczeniowej ze względu na bardzo dużą liczbę parametrów oraz kosztowne przetwarzanie danych. Zatem zadanie natychmiastowej klasyfikacji ścieżki dźwiękowej w czasie rzeczywistym będzie wykonywał serwer a wynik będzie zwrócony do klienta.

Framework Django umożliwia napisanie serwera małą ilością kodu. Serwer udostępnia jeden 'endpoint' *http://{ip_serwera:port_serwera}/upload/* na który powinne być wysyłane ścieżki dźwiękowe. Struktura katalogów jest podana na rysunku 4.3.



Rysunek 4.3: Struktura katalogów na serwerze.

Jest to typowa struktura dla danego frameworku. W katalogu `resnet-models` znajdują się wyeksportowane nauczone modele. Katalog files służy jako cache dla plików, które się znajdują w trakcie przetwarzania.

Serwer zawiera instancje `learnera` z biblioteki `fastai` z importowanymi wagami wcześniej wytrenowanych modeli. Po otrzymaniu formy, która zawiera plik w formacie M4A serwer odczytuje tą ścieżkę dźwiękową, przetwarza ją do postaci spektrogramu w skali melowej, podaje do sieci zaś wynik z sieci jest zwracany do klienta jako ciało odpowiedzi HTTP.

4.4 Aplikacja mobilna

4.4.1 Interfejs użytkownika

Aplikacja mobilna służy do nagrywania ścieżek dźwiękowych oraz wysłania ich na serwer. Aplikacja wyświetla wynik klasyfikacji obrazu po otrzymaniu odpowiedzi od serwera. Jeśli w nawiązaniu połączenia wystąpił błąd (np. serwer jest niedostępny), to wynik klasyfikacji jest wyświetlany i zapisywany jako nieklasyfikowany.

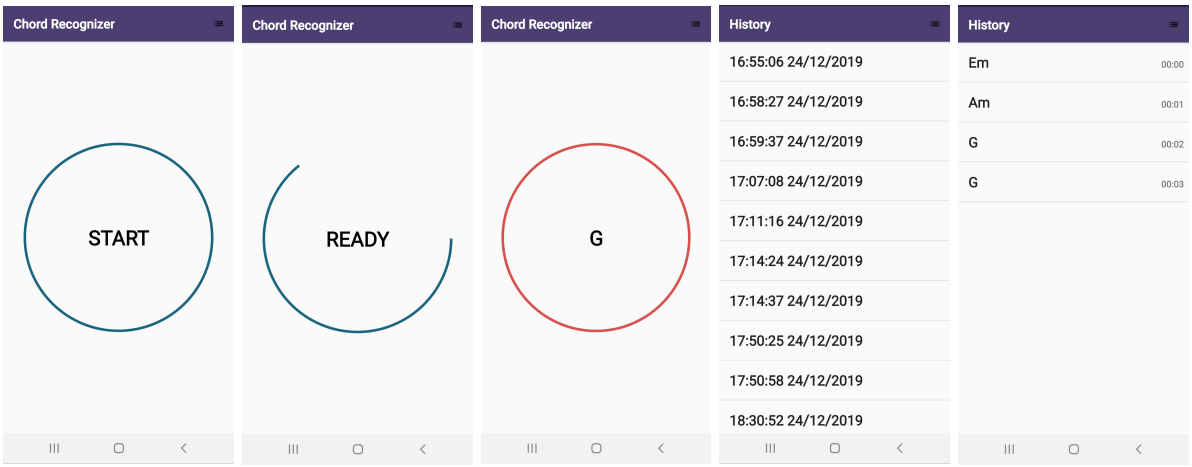
Po otworzeniu aplikacji jest wyświetlany główny widok. Zawiera on pole tekstowe, w którym są wyświetlane wyniki klasyfikacji oraz kontroler, który zarządza całym systemem i ma okrągły kształt. Przy kliknięciu na kontroler okrąg progresu zaczyna wypełniać się a pole tekstowe wyświetla komunikat przygotowania systemu do nagrywania. Po dwóch sekundach od kliknięcia okrąg progresu wypełnia się w całości i zmienia kolor na czerwony informując tym, że rozpoczęło się nagrywanie.

Użytkownik w dowolnym momencie może zacząć nagrywanie lub zatrzymać je klikając na kontroler. Czas od początku do końca nagrywania jest nazywany sesją. Sesja zawiera listę predykcji co do nagranych akordów. Lista wszystkich sesji jest nazywana historią.

W procesie nagrywania w odstępach mniej więcej jednej sekundy są wyświetlane wyniki klasyfikacji dla poprzedniej sekundy. Po kolejnym kliknięciu kontrolera sesja jest zamykana, kolor kontrolera zmienia się na początkowy.

Użytkownik może przejść do okna historii klikając przycisk menu znajdujący się w prawym górnym rogu. Widok historii zawiera listę godzin, w których były rozpoczęte poszczególne sesje. Po kliknięciu na element listy jest wyświetlany widok sesji, który zawiera listę predykcji w odrębnie wybranej sesji wraz z sekundą nagrywania ścieżki dźwiękowej dla danego akordu.

Poszczególne widoki są pokazane na rysunku 4.4. Po kliknięciu kontrolera w widoku 4.4c aplikacja wraca do stanu w widoku 4.4a. Nagrywanie jest automatycznie przerywane przy przejściu do historii lub zamknięciu aplikacji.

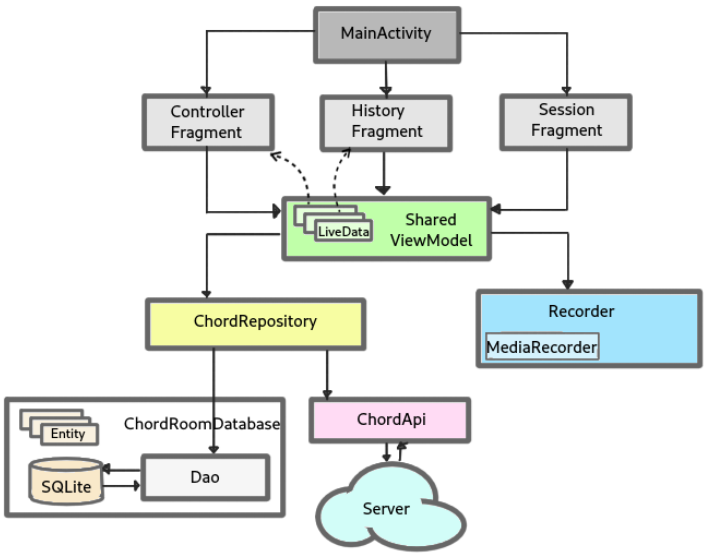


(a) Widok startowy (b) Przygotowanie (c) Rozpoznawanie (d) Widok historii (e) Widok sesji

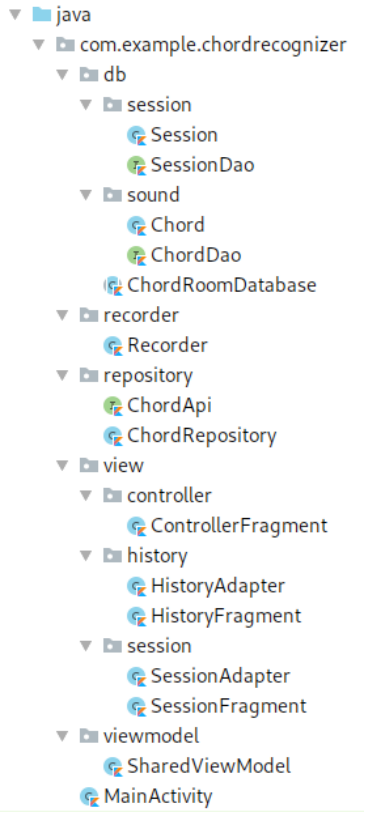
Rysunek 4.4: Interfejs aplikacji

4.4.2 Architektura aplikacji

Jest zalecanym [44] podzielenie architektury aplikacji na komponenty, celem których jest dostarczenie instrukcji dotyczących tej aplikacji wraz z bibliotekami do typowych zadań, takich jak zarządzanie cyklem życia i zapisywanie danych. Na rysunku 4.5 są pokazane zależności pomiędzy komponentami oraz struktura katalogów aplikacji mobilnej. Poniżej znajduje się opis poszczególnych komponentów dla wybranej architektury.



(a) Zależności pomiędzy komponentami



(b) Struktura katalogów aplikacji mobilnej

Rysunek 4.5: Architektura aplikacji mobilnej

1. `ChordRoomDatabase` dziedziczy po klasie `RoomDatabase` i służy do zarządzania lokalnymi trwałymi

danymi. Ten komponent upraszcza pracę z bazą danych SQLite ukrywając wiele automatycznie generowanego kodu. Ten komponent zajmuje się podstawowymi zadaniami, które zwykle są wykonywane za pomocą `SQLiteOpenHelper`. Jest singletonem.

Baza danych `ChordRoomDatabase` używa `DAO` (ang. data access object), który jest odwzorowaniem zapytań SQL na funkcje dla określonej tabeli. Korzystanie z bazy przez inne komponenty odbywa się właśnie przez `DAO`. `Entity` jest klasą, która opisuje tabele w bazie danych. `ChordRoomDatabase` wykorzystuje pola publiczne tej klasy do utworzenia takiej tabeli oraz mapowania obiektów na wpisy w bazie danych.

W tej aplikacji są dwie klasy dziedziczącej po `Entity`:

Session:

- `id`: Long
- `timestamp`: Long
  gdzie `id` jest kluczem głównym, `timestamp` odwzorowuje czas i datę nagrywania sesji (wartość w milisekundach).

Chord:

- `id`: Long
- `sessionId`: Long
- `predictedClass`: String
  gdzie `id` jest kluczem głównym, `sessionId` jest kluczem obcym, a `predictedClass` zawiera klasę przypisaną do ścieżki dźwiękowej.

Domyślnie, aby uniknąć niskiej wydajności interfejsu użytkownika, nie jest dozwolone wysyłanie zapytań z głównego wątku aplikacji. Dlatego przy odczycie Room zwraca obiekty LiveData co umożliwia asynchroniczne przekazywanie danych, a zmiany do bazy danych są automatycznie uruchamiane asynchronicznie w wątku w tle.

2. `ChordApi` jest interfejsem, przez który odbywa się łączenie z serwerem. Zawiera jedną metodę `upload`, która przyjmuje formę zawierającą plik nagranej ścieżki dźwiękowej.

3. `ChordRepository` służy do zarządzania wieloma źródłami danych. Zarządza zapytaniami i pozwala korzystać z wielu serwisów. Repozytorium zawiera referencje do obiektów `DAO` w przeciwieństwie do całej bazy danych, ponieważ zawierają one wszystkie potrzebne metody odczytu /zapisu dla bazy danych.

4. `Recorder` buduje i zarządza natywnym `MediaRecorderem`, który zajmuje się nagrywaniem ścieżek dźwiękowych. Specyfikuje on takie parametry jak długość nagranej ścieżki, format pliku, częstotliwość próbkowania.

5. Rolą `SharedViewModel` jest dostarczanie danych do interfejsu użytkownika i przetwarzanie zmian zachodzących w interfejsie. Działa jako centrum komunikacji między repozytorium (danymi), rekorderem (mikrofonem) a interfejsem użytkownika. Interfejs użytkownika nie wie o pochodzeniu danych. Komponent umieszcza informacje o zmianach w modelu w obiektach `LiveData` oraz przechowuje dane interfejsu użytkownika aplikacji w sposób uwzględniający cykl życia, który przetrwa zmiany konfiguracji. Zarządza danymi i komunikacją z serwerem za pośrednictwem repozytorium, a nagrywaniem za pośrednictwem rekordera.

6. `LiveData` jest posiadaczem danych, które można obserwować. Zawsze przechowuje najnowszą wersję danych i powiadamia swoich obserwatorów o zmianie tych danych. Komponenty interfejsu użytkownika obserwują tylko odpowiednie dane i nie przerywają ani nie wznawiają obserwacji. Z takim mechanizmem aktualizacja interfejsu użytkownika następuje wtedy, gdy dane faktycznie się zmieniają. `LiveData` automatycznie zarządza nadawaniem danych do wszystkich oraz jest stabilnym co do cyklu życia, ponieważ nie zależy od cyklu życia obserwatorów.

7. Fragmenty są komponentami odpowiednich widoków interfejsu użytkownika. Przekazują zmiany, które zachodzą w interfejsie do `SharedViewModel`.

Wyświetlanie danych w `HistoryFragment` oraz `SessionFragment` jest wykonywane w postaci listy umieszczanej w `RecyclerView`. Taki mechanizm wyświetlenia pozwala rozdzielić sposób wyświetlania elementu w liście od pozostałej logiki. Adaptery służą do definiowania sposobu wyświetlania bieżącej zawartości bazy danych w `RecyclerView`.

8. `MainActivity` jest komponentem, który w tej aplikacji zawiera jeden z fragmentów. Przejście pomiędzy widokami to zamiana jednego fragmentu drugim.

Opisana struktura zapewnia niską spójność pomiędzy komponentami oraz odporność na błędy przy zmianach w cyklu życia komponentów interfejsu użytkownika.

Podsumowanie

Komponenty systemu zajmują następującą przestrzeń na dysku:

a) aplikacja mobilna zajmuje 3.54 MB ,

b) kod źródłowy serwera bez zależności zajmuje 18 KB ,

c) wagi wytrenowanego modelu sieci neuronowej zajmuje 46,9 MB.

Jak już było wspomniane w rozdziale o implementacji, model klasyfikował poprawnie ponad 98% danych ze zbioru testowego, który się składał z 300 nagranych akordów.

W rzeczywistości system też wykazuje dobre wyniki, chociaż największym problemem dla modelu są momenty zmiany akordu, ponieważ ciężko mu zdecydować jaki z dwóch akordów jest aktualnie grany. Przy graniu tego samego akordu kilka sekund nie zauważono problemów z klasyfikacją.

Również system jest podatny na błędy dla akordów, które są bardzo podobne. Tak na przykład Em i E, które różnią się tylko jednym tonem na trzeciej strunie, mogą czasem być mylone między sobą.

Żeby zwalczyć ten problem, można udoskonalić system możliwością rozumienia kontekstu. Tak na przykład rekurencyjne sieci neuronowe potencjalnie mogą być przydatne w sytuacji, gdy jeden akord jest grany kilka razy oraz prawdopodobieństwo zmiany akordu musi być brane pod uwagę. Dodatkowo można wziąć pod uwagę popularne wzorce i kolejności akordów występujące w większości piosenek. Taką przykładową kolejnością jest Am-C-G-D.

System wykazuje odporność na błędy związane ze sposobem grania akordu (z góry do doły lub z dołu do góry). Działa on poprawnie nawet wtedy, kiedy w jednej sekundzie jest granych kilka bić tego samego akordu, ponieważ model wybiera najbardziej prawdopodobny wynik klasyfikacji.

Opisane podejście do rozpoznawania akordów w czasie rzeczywistym może być przydatne przy uczeniu się gry na gitarze, łatwym zapisywaniu akordów przy wymyśleniu piosenek lub przy rozpoznawaniu akordów występujących w piosence.

Bibliografia

[1] https://play.google.com/store/apps/details?id=com.yousician.yousician.

[2] https://play.google.com/store/apps/details?id=com.yallafactory.mychord.

[3] https://play.google.com/store/apps/details?id=com.xssemble.chordnamefinder.

[4] https://play.google.com/store/apps/details?id=com.finestandroid.chorddetector.

[5] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Douglas Eck, Karen Simonyan, and Mohammad Norouzi. Neural audio synthesis of musical notes with wavenet autoencoders, 2017.

[6] https://pl.wikipedia.org/wiki/Sygna%C5%82.

[7] C.E. Shannon. *Communication in the Presence of Noise*. IEEE, 1949.

[8] https://jackschaedler.github.io/circles-sines-signals/.

[9] https://pl.wikipedia.org/wiki/D%C5%BAwi%C4%99k.

[10] Khan academy, algebra ii. online course: https://www.khanacademy.org/math/algebra2.

[11] Douglas O'Shaughnessy. *Speech communication : human and machine*. Reading, Mass. : Addison-Wesley Pub. Co., 1987.

[12] https://dawn.cs.stanford.edu/benchmark/.

[13] https://arxiv.org/abs/1512.03385.

[14] https://victorzhou.com/blog/intro-to-neural-networks/.

[15] https://en.wikipedia.org/wiki/Universal˙approximation˙theorem.

[16] https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a

[17] https://embarc.org/embarc˙mli/doc/build/html/MLI˙kernels/convolution˙depthwise.html.

[18] https://www.researchgate.net/publication/328804297˙Multiple˙Sclerosis˙Identification˙by˙14-Layer˙Convolutional˙Neural˙Network˙With˙Batch˙Normalization˙Dropout˙and˙Stochastic˙Pooling.

[19] https://arxiv.org/abs/1311.2901.

[20] https://missinglink.ai/guides/convolutional-neural-networks/convolutional-neural-network-tutorial-basic-advanced/

[21] https://www.cs.toronto.edu/˜kriz/cifar.html.

[22] http://www.image-net.org/.

[23] https://arxiv.org/abs/1712.09913.

[24] https://hackernoon.com/.

[25] https://www.fast.ai/.

[26] https://www.kaggle.com/rafjaa/dealing-with-very-small-datasets.

[27] https://www.cs.toronto.edu/˜hinton/absps/JMLRdropout.pdf.

[28] https://arxiv.org/abs/1803.09820.

[29] https://pytorch.org/.

[30] https://librosa.github.io/librosa/.

[31] https://numpy.org/.

[32] https://pandas.pydata.org/.

[33] https://jupyter.org/.

[34] https://colab.research.google.com/.

[35] https://www.djangoproject.com/.

[36] https://www.android.com/.

[37] https://gs.statcounter.com/os-market-share/mobile/worldwide.

[38] https://kotlinlang.org/.

[39] https://gradle.org/.

[40] https://square.github.io/.

[41] https://developer.android.com/reference/android/media/MediaRecorder.

[42] https://developer.android.com/jetpack.

[43] http://recordingology.com/in-the-studio/guitars/.

[44] https://developer.android.com/jetpack/docs/guide.

[45] https://developer.android.com/studio.

Zawartość płyty CD

Płyta CD zawiera kody źródłowe serwera, aplikacji mobilnej oraz środowiska dla trenowania modelu sieci neuronowej.

W folderze `model` znajduje się plik `training.ipynb`, który powinien być zaimportowany na stronie Google Colab. Jest to interaktywne środowisko dla dobierania parametrów CNN oraz wygenerowania ścieżek dźwiękowych.

Skrypt `model/refactor_nsynth.py` pomaga usunąć wszystkie niepotrzebne dane z Nsynth Database [5] oraz podzielić bazę według instrumentów, tonów oraz prędkości grania, co ułatwia wyszukiwanie potrzebnej ścieżki dźwiękowej przy generowaniu akordów.

W folderze `model/audio` znajdują się niepodzielone nagrane ścieżki dźwiękowe.

Plik `model/paddings.csv` zawiera informacje o początkowych odstępach dla nagranych ścieżek dźwiękowych.

Dla uruchomienia serwera na komputerze musi być zainstalowany `python3` oraz wszystkie moduły opisane w rozdziale implementacji. Uruchamianie serwera wygląda następująco.

```
$ cd ./server
$ python3 manage.py runserver
```

W folderze `serwer/chord_recognizer_service/resnet-models` przy zmienianiu modelu sieci neuronowych nowy model musi być umieszczony jako plik `export.pkl`.

Folder `android` zawiera project Android Studio [45]. Żeby ustawić IP serwera, z którym się komunikuje aplikacja muszą być zmienione następujące parametry:

a) W pliku `app/src/main/res/xml/network_security_config.xml` musi być dodana domena serwera,

b) W klasie `ChordRepository` stała `API_URL` musi być zmieniona na IP serwera.