

UNIwersytet Kardynała Stefana Wyszyńskiego
w Warszawie

Wydział Matematyczno-Przyrodniczy
Szkoła Nauk Ścisłych

Adrianna Jankowska

Nr albumu: 107376

Kierunek studiów: informatyka

Analiza skuteczności metod filtracji i klasyfikatorów rozpoznających symbole zapisu nutowego

Praca licencjacka wykonana
pod kierunkiem naukowym
dr inż. Roberta Kłopotka

Warszawa 2020

.....
Imię i nazwisko studenta/studentki

.....
Nr albumu

.....
Wydział

.....
Instytut

.....
Kierunek

Dziekan Wydziału

.....

OŚWIADCZENIE

Świadomy(a) odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w żadnej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Oświadczam, że poinformowano mnie o zasadach dotyczących kontroli samodzielności prac dyplomowych i zaliczeniowych. W związku z powyższym oświadczam, że wyrażam zgodę na przetwarzanie* moich prac pisemnych (w tym prac zaliczeniowych i pracy dyplomowej) powstałych w toku studiów i związanych z realizacją programu kształcenia w Uczelni, a także na przechowywanie pracy dyplomowej w celach realizowanej procedury antyplagiatowej w ogólnopolskim repozytorium pisemnych prac dyplomowych.

.....
podpis studenta

*Przez przetwarzanie pracy rozumie się porównywanie przez system antyplagiatowy jej treści z innymi dokumentami (w celu ustalenia istnienia nieuprawnionych zapożyczeń) oraz generowanie raportu podobieństwa.

OŚWIADCZENIE

Oświadczam, że niniejsza praca napisana przez

Pana/Panią....., nr albumu została przygotowana pod moim kierunkiem i stwierdzam, że spełnia ona warunki do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

.....
podpis promotora

Załącznik do Zarządzenia nr 14/2010
Rektora UKSW z dnia 15 marca 2010 r.
Załącznik nr 4
do Zarządzenia nr 39/07
Rektora UKSW z dnia 9 listopada 2007r.

Imię i nazwisko autora pracy	
Imię i nazwisko promotora pracy	
Wydział	
Kierunek studiów	
Specjalność	
Tytuł pracy	

Niniejszym oświadczam, że zachowując moje prawa autorskie, udzielam Uniwersytetowi Kardynała Stefana Wyszyńskiego w Warszawie nieodpłatnej licencji niewyłącznej do korzystania z ww. pracy bez ograniczeń czasowych w zakresie rozpowszechniania pracy poprzez publiczne udostępnianie pracy w wersji drukowanej i elektronicznej, w taki sposób, aby każdy zainteresowany mógł mieć do niej dostęp w miejscu, w którym praca jest przechowywana tj.: w Archiwum UKSW lub w Bibliotece UKSW.

/podpis /

Streszczenie

Niniejsza praca dyplomowa ma na celu porównanie skuteczności pięciu klasyfikatorów uczenia maszynowego oraz własnej sieci neuronowej w problemie rozpoznawania symboli w odręcznie spisanym partyturach muzycznych. Hipotezą badawczą pracy jest teoria, iż własnoręcznie budowane sieci neuronowe, nawet nieskomplikowane, osiągają wyższe wyniki w procesie rozpoznawania symboli muzycznych w odręcznej partyturze.

Praca zawiera opis implementacji programu realizującego zagadnienie, metodologię badań oraz wyniki przeprowadzonych eksperymentów. Oceną efektywności danych algorytmów będzie miara poprawności klasyfikacji na różnych obrazach i czas wykonania algorytmu.

Słowa kluczowe: nuty, OMR, Python, uczenie maszynowe, optyczne rozpoznawanie znaków, regresja logistyczna, liniowa analiza dyskryminacyjna, k-najbliższych sąsiadów, drzewo decyzyjne, naiwny Bayes, maszyna wektorów nośnych, klasyfikacja.

Abstract

Optyczne rozpoznawanie muzyki (ang. *Optical Music Recognition*) jest to system służący rozpoznawaniu partytur muzycznych. Choć termin OMR jest ściśle powiązany z optycznym rozpoznawaniem znaków (ang. *Optical Character Recognition*), oba systemy rzadko współdzielą uniwersalne algorytmy do celów rozpoznawania. To, co odróżnia je od siebie, jest fakt, iż systemy OMR próbują automatycznie przekształcić zeskanowane nuty w format czytelny dla komputera. W tymże formacie są również przechowywane dane semantyczne takie jak wysokości nut, czas trwania, instrumentacja czy informacje kontekstowe.

Spis treści

Streszczenie	4
Abstract	4
1 Wstęp	7
2 Historia i problem badawczy	9
2.1 <i>Tło historyczne i problematyka badań w świetle literatury przedmiotu</i>	9
2.2 <i>Zarys problemu badawczego</i>	10
2.3 <i>Podział na etapy</i>	13
2.3.1 <i>Preprocesing obrazu</i>	14
2.3.2 <i>Rozpoznawanie symboli muzycznych</i>	15
2.3.3 <i>Rekonstrukcja zapisu muzycznego i ostateczna konstrukcja reprezentacji</i>	17
3 Zbiór danych wejściowych	19
4 Implementacja programu	23
4.1 <i>Przygotowanie zbioru danych</i>	24
4.2 <i>Wykorzystanie gotowych klasyfikatorów</i>	30
4.3 <i>Zbudowanie własnej sieci neuronowej</i>	32
5 Wyniki	35
5.1 <i>Wyniki testów na gotowych klasyfikatorach</i>	36
5.1.1 <i>Z konwersją danych wejściowych</i>	36
5.1.2 <i>Bez konwersji danych wejściowych</i>	40
5.2 <i>Wyniki testów na własnej sieci neuronowej</i>	43
5.2.1 <i>Z konwersją danych wejściowych</i>	43
5.2.2 <i>Bez konwersji danych wejściowych</i>	47
6 Podsumowanie	51
Bibliografia	53

Rozdział 1

Wstęp

Partytura muzyczna jest głównym artefaktem, służącym do przekazywania ekspresji muzycznej w tradycjach niesłuchowych. Pojawienie się muzycznych systemów typograficznych pod koniec XIX wieku, a ostatnio pojawienie się bardzo wyrafinowanych systemów do edycji rękopisów muzycznych i systemów do układania stron, ilustruje ciągle wchłanianie nowych technologii w ekosystemach dla tworzenia utworów muzycznych i ich części.

Aż do niedawna większość twórców nagrań - kompozytorów muzyki filmowej, teatralnej, koncertowej czy sakralnej - nadal używała tradycyjnego ręcznego zapisu nut za pomocą tuszu i papieru. Wczesne oprogramowanie do pisania muzyki komputerowej opracowane w latach 70. i 80. XX wieku zapewniało doskonałą wydajność, ale z kolei było niewygodne w użyciu [4]. Zatem zapotrzebowanie na solidny i dokładny system optycznego rozpoznawania muzyki (OMR) pozostaje aktualne.

Digitalizacja jest powszechnie stosowana jako możliwe narzędzie do konserwacji, oferujące łatwą duplikację, dystrybucję i przetwarzanie cyfrowe [23]. Potrzebny jest jednak czytelny maszynowo format symboliczny z partytur muzycznych, aby ułatwić operacje takie jak wyszukiwanie, odzyskiwanie i analiza. Ręczna transkrypcja partytur muzycznych na odpowiedni format cyfrowy jest bardzo czasochłonna, dlatego też z pomocą przychodzą algorytmy uczenia maszynowego.

Celem tej pracy jest wymierne porównanie efektywności klasyfikatorów uczenia maszynowego pod kątem miary precyzji klasyfikacji i prędkości przeprowadzenia rozpoznania na podstawie cyfrowych zapisów nutowych. Hipotezą badawczą pracy jest teoria, iż własnoręcznie budowane sieci neuronowe, nawet nieskomplikowane, osiągają wyższe wyniki w procesie rozpoznawania symboli muzycznych w odręcznej partyturze.

Praca w dużej mierze opiera się na informacjach zawartych w artykule naukowym [1].

W kolejnym rozdziale skonkludowany jest problem badawczy, jakim jest rozpoznawanie symboli nutowych i w jaki ogólnopojęty sposób jest on zazwyczaj analizowany za pomocą narzędzi programistycznych. Przybliżone zostaną podstawowe pojęcia z zakresu uczenia maszynowego takie, jak usuwanie szumów, operacje morfologiczne, binaryzacja, klasyfikacja, wygładzanie histogramu. Przywołane zostaną także podstawy ze świata muzyki takie jak nazewnictwo danych rodzajów symboli czy rozróżnienie notacji muzycznych z przeszłych wieków.

Rozdział trzeci w pełni poświęcony jest opisowi zbioru danych, jego pochodzeniu i analizie.

Czwarty rozdział rozpoczyna dogłębną analizę i konwersję danych wejściowych, a następnie w obszerny sposób skupia się na implementacji programu - użytym języku programowania i jego bibliotekach. Zobrazowane zostaną poszczególne metody architektury.

Metodologia badań zamieszczona jest w rozdziale piątym, który przedstawia miary oceny klasyfikatorów.

Rozdział szósty jest wymiernym przedstawieniem wyników eksperymentów za pomocą uśrednionej skuteczności klasyfikatorów oraz uśrednionego czasu wykonania działań.

Ostatni rozdział zaś zawiera ogólne podsumowanie przeprowadzonych testów na zbudowanych modelach i wyciągnięte wnioski z nich płynące.

Rozdział 2

Historia i problem badawczy

2.1. Tło historyczne i problematyka badań w świetle literatury przedmiotu

Pierwsze poważne wzmianki o technicznych próbach optycznego rozpoznawania symboli muzycznych datuje się na późne lata 60' ubiegłego wieku, gdy przy pomocy najnowszych technologii firma MIT zaczęła produkować skanery w cenach akceptowalnych dla instytutów badawczych. Za prekursorów rozwijania technik OMR uważa się naukowców z MIT - Dennisa Pruslina [2] i Davida Prerau'a [3]. Jednym z pierwszych dużych projektów inżynierskich wykorzystujących dobrodziejstwa eksperymentów OMR była budowa japońskiego robota WABOT-2 [4] w 1984 roku, który był w stanie odegrać na keyboardzie melodię odczytaną z wydrukowanego na papierze zapisu nutowego.

Temat OMR stale jest intensywnie badany. Badania te dzielą się na trzy główne kategorie: ocena istniejących metod, odkrywanie nowatorskich rozwiązań i klasyczne podejście do OMR.

Badania nad oceną istniejących metod polegają na łączeniu wyników wielu komercyjnych produktów OMR w celu osiągnięcia maksymalnie wysokiego wyniku ogólnego [7] [8] i nieco węższego pojęcia definicji [9] [10].

Istnieje wiele prac traktujących na temat poszczególnych problemów składowych takich, jak wykrywanie rytmu [11] [12], taktów zawierających nuty [13] i konkretyzacja, dla jakiego instrumentu partytura jest przeznaczona [14] [15]. Analizowano także samą jakość skanowanych obrazów [16].

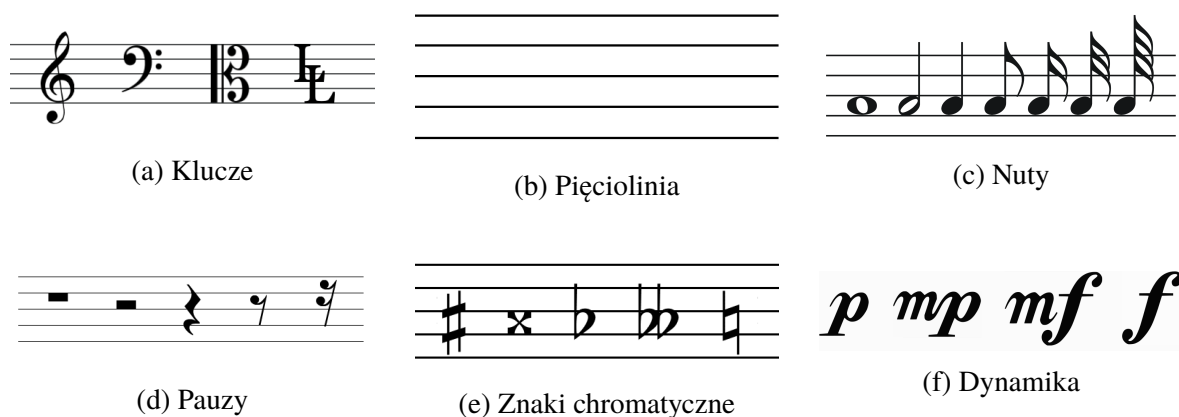
W ostatnim etapie OMR (rekonstrukcji zapisu muzycznego i ostatecznej konstrukcji reprezentacji) niektórzy autorzy używają kilku algorytmów do wykonywania różnych zadań składowych np. jednego algorytmu do wykrywania głowic nut, a innego do wykrywania łodyg. Do ich scalenia użyto systemu głosowania z algorytmem porównawczym najlepszych cech kilku algorytmów OMR dla lepszej produkcji wyników [5] [6].

2.2. Zarys problemu badawczego

Problem przekształcania odręcznie zanotowanego zapisu nutowego w plik graficzny jest współcześnie powszechnym, acz złożonym zadaniem. Głównymi celami systemu OMR są rozpoznawanie, prezentacja i przechowywanie muzyki w formacie odczytywalnym maszynowo. Program OMR powinien zatem być w stanie rozpoznać treść muzyczną i dokonać analizy semantycznej każdego muzycznego symbolu.

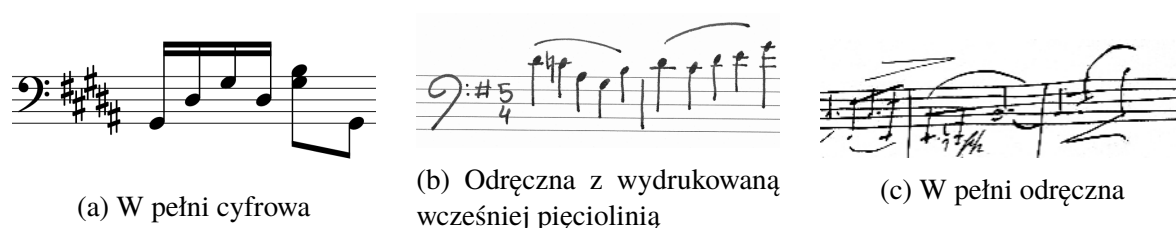
Skupiamy się w niej wyłącznie na analizie zachodniej notacji muzycznej (znanej również jako wspólna notacja muzyczna - CMN (ang. *Common Music Notation*), choć zostały opracowane systemy OMR do rozpoznawania innych rodzajów notacji np. średniowiecznej notacji muzycznej [17] [18]. Zrozumienie każdej notacji muzycznej wymaga znajomości informacji, która notacja ta stara się przekazać. W przypadku CMN są to wysokość, metrum, głośność i tembr.

Rysunek 2.1 ukazuje przykładowe rodzaje symboli wspólnej notacji muzycznej. Klucze (Rys. 2.1a) są to znaki mówiące o sposobie odczytywania dźwięków na pięciolinii (Rys. 2.1b), która jest niczym innym jak przestrzenią, na której dokonuje się zapisu nutowego. Rodzaj nuty (Rys. 2.1c) definiuje długość trwania dźwięku, a jej umiejscowienie względem pięciolinii - jego wysokość. Zestaw pauz (Rys. 2.1d) służy ukazaniu długości przerwy między wybrzmiewającymi dźwiękami, zaś znaki chromatyczne (Rys. 2.1e) mają funkcję obniżać lub podwyższać wysokość dźwięków. Głośność nut dyktują symbole dynamiki (Rys. 2.1f), dzięki którym wiemy, jak głośno zagrać daną partię.



Rysunek 2.1: Przykładowe symbole wspólnej notacji muzycznej. Źródło: [20]

Równie frapującą kwestią jest analiza partytur o różnym stopniu zdigitalizowania. Zauważa się zapisy w pełni zdigitalizowane (Rys. 2.2a) (takie, których zarówno pięciolinia jak i symbole na niej zostały wydrukowane komputerowo), partytury zapisane odręcznie na wydrukowanej wcześniej pięciolinii (Rys. 2.2b) oraz takie stworzone w pełni analogowo (Rys. 2.2c) - zarówno linie pięciolinii jak i nuty naniesiono ręcznie na papier, co nadaje im wyglądu najdalszego od tego, który znamy we współczesnym świecie.



Rysunek 2.2: Stopień zdigitalizowania partytury muzycznej. Źródło: [22]

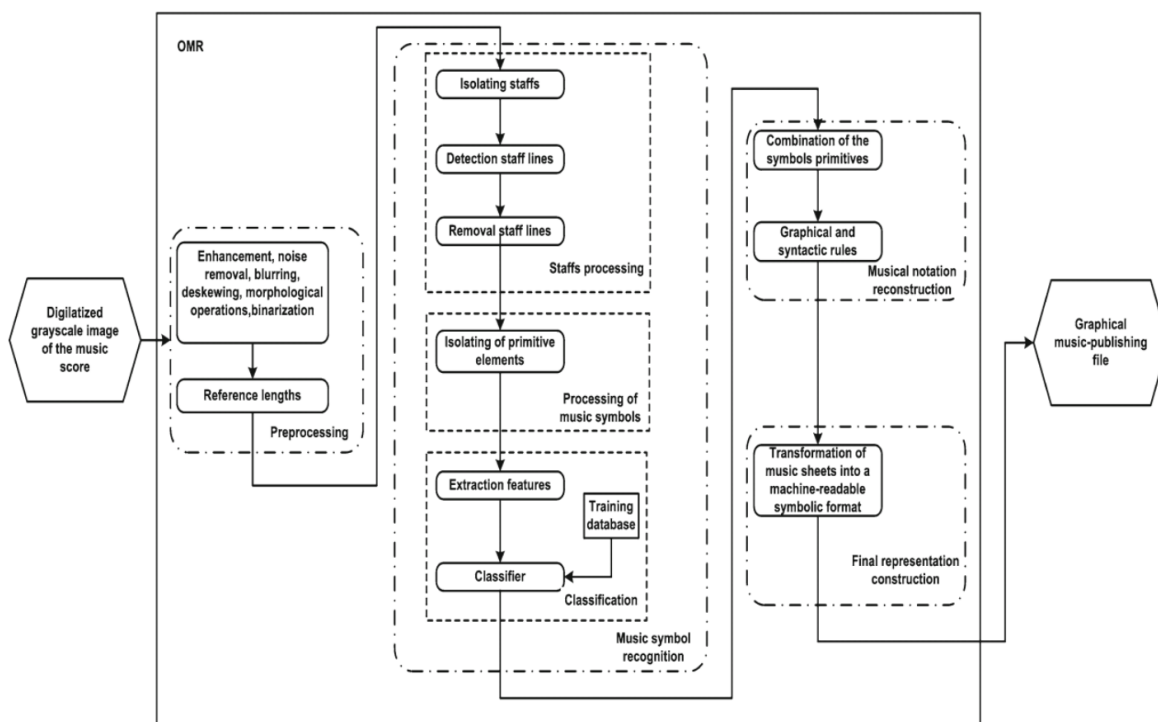
W pracy tej koncentrujemy się na wyodrębnionych uprzednio odręcznie zapisanych symbolach prymitywnych tj. nie bierzemy pod uwagę umiejscowienia symboli względem pięciolinii. Jednakże dla lepszego rozpoznania problematyki badań, w następnym podrozdziale prezentujemy klasyczne, kompleksowe podejście do zagadnienia optycznego rozpoznawania muzyki z podziałem na etapy.

2.3. Podział na etapy

Głównymi czterema etapami, na jaki dzieli się typowy algorytm optycznego rozpoznawania muzyki są (Rys. 2.3):

1. wstępne przetworzenie obrazu
2. rozpoznawanie symboli muzycznych
3. scalenie symboli prymitywnych w semantycznie logiczną całość
4. zbudowanie modelu zapisu nutowego w formacie wynikowym jako symboliczny opis arkusza muzycznego

Dla każdego z opisanych powyżej etapów istnieją różne metody wykonania odpowiedniego zadania.

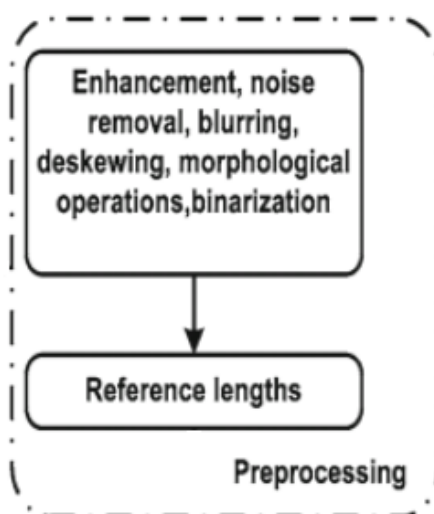


Rysunek 2.3: Schemat kolejnych etapów algorytmu OMR. Źródło: [1]

2.3.1. Preprocessing obrazu

Etap ten najczęściej dzieli się na trzy bazowe części (Rys. 2.4):

- powiększenie odczytanego symbolu do odpowiednich rozmiarów
- usuwanie szumów
- zaostrenie krawędzi, by nie były rozmyte
- wyprostowanie linii z łuków do równoległych prostych
- przeskalowanie, jeśli zdjęcie zostało zrobione pod niewłaściwym kątem
- konwersja skali szarości na skalę czarno-białą
- standaryzacja wielkości obrazu

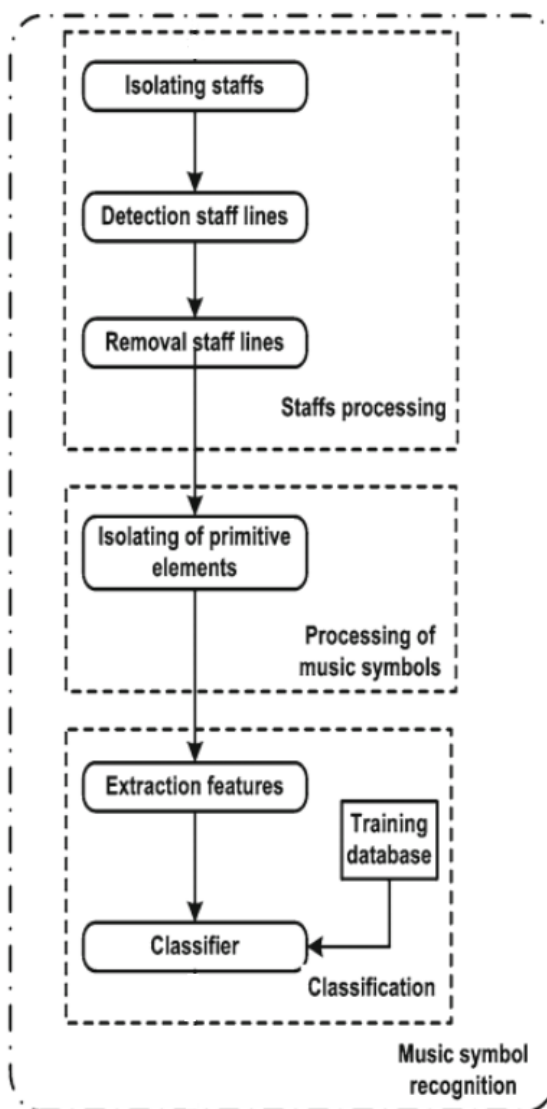


Rysunek 2.4: Przykładowe kroki podejmowane na etapie preprocessingu. Źródło: [1]

2.3.2. Rozpoznawanie symboli muzycznych

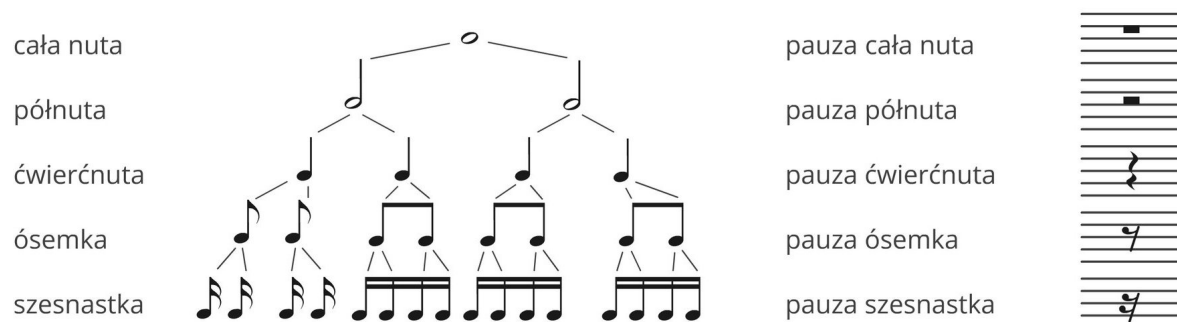
Na etapie wstępnego przetwarzania obrazu do zapisu nutowego (Rys. 2.5) można zastosować kilka technik:

- wykrywanie i usuwanie linii pięciolinii, aby uzyskać obraz zawierający tylko symbole muzyczne
- segmentacja symboli na elementy prymitywne
- klasyfikacja symboli



Rysunek 2.5: Wewnętrzny podział na kolejne podetapy rozpoznawania symboli muzycznych.
Źródło: [1]

Na tym etapie klasyfikatory zwykle otrzymują surowe piksele jako cechy wejściowe. Jednak niektóre prace dotyczą większych elementów, takich jak informacje o podłączonych komponentach np. nuty połączone wspólnym daszkiem lub orientacji symbolu względem pięciolinii (Rys. 2.6).



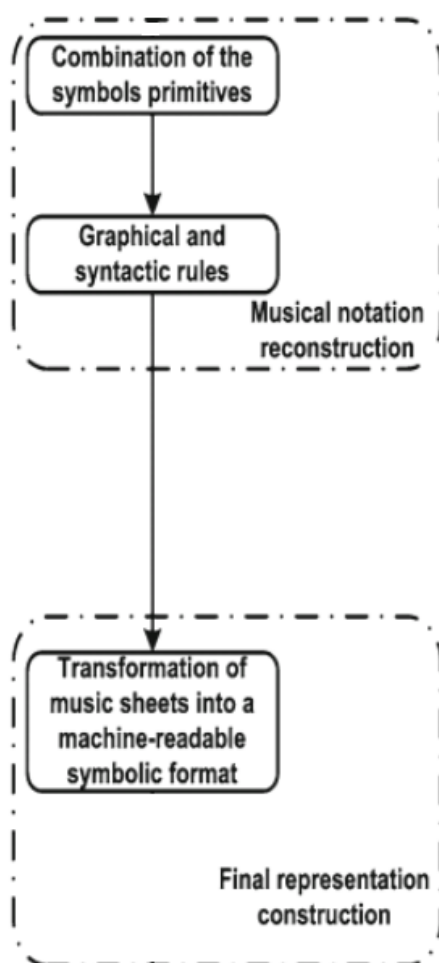
Rysunek 2.6: Przykład nut złożonych z wielu symboli prymitywnych. Źródło: [21]

2.3.3. Rekonstrukcja zapisu muzycznego i ostateczna konstrukcja reprezentacji

Trzeci i czwarty etap algorytmu mogą być ze sobą ściśle powiązane.

Na etapie rekonstrukcji zapisu muzycznego (Rys. 2.7) symbole prymitywne łączą się, tworząc pełnoprawne symbole muzyczne. W tym kroku stosuje się reguły graficzne i składniowe w celu wprowadzenia zweryfikowanych informacji kontekstowych. Wykryte symbole są interpretowane i zaczynają mieć znaczenie muzyczne.

W czwartym i ostatnim etapie tworzony jest format muzycznego pliku wyjściowego z wcześniej wytworzoną informacją (Rys. 2.7). Najczęściej spotykanymi formatami są MIDI i MusicXML.



Rysunek 2.7: Kolejne kroki podejmowane w celu konstrukcji reprezentacji cyfrowej. Źródło: [1]


















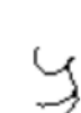
Rozdział 3

Zbiór danych wejściowych

Zbiorem danych wykorzystanym w tej pracy naukowej jest zbiór ”The Handwritten Online Music Symbols (HOMUS) dataset” autorstwa Jorge Calvo-Zaragoza i Jose Oncina [19], dostępny do pobrania na stronie <https://grfia.dlsi.ua.es/homus/>. Celem autorów było dostarczyć korpus danych składających się z różnorodnych próbek odręcznego pisma nutowego.

Zbiór ten składa się z 15200 próbek pojedynczych symboli muzycznych odręcznie zapisanych przez 100 różnych autorów. Jak dowiadujemy się z dokumentacji datasetu [19], byli nimi muzycy ze szkoły muzycznej *Escuela de Educandos Asociacion Musical l’Avanc* w El Campello, Hiszpania oraz szkoły muzycznej *Conservatorio Superior de Musica de Murcia ”Manuel Massotti Littell”* w Murcii, Hiszpania. Wśród tych, spod których piór powstały próbki nut w zbiorze, byli zarówno wykwalifikowani nauczyciele, jak i zaawansowani uczniowie owych szkół. Aby sprawić zbiór bardziej różnorodnym i na swój sposób nieperfekcyjnym, próbki zebrano zarówno od osób biegłych i doświadczonych w odręcznym zapisie nutowym, jak i od takich, które nie reprezentowały dużego dorobku doświadczenia w tej materii.

Uczestnicy zostali poproszeni o czterokrotne narysowanie w ich własnym, indywidualnym stylu 32 rodzajów symboli muzycznych. Świadectwem różnorodności stylów jest Rysunek 3.1, ukazujący 3 rodzaje nut nakreślone przez 5 różnych autorów.

Symbol	Writer 1	Writer 2	Writer 3	Writer 4	Writer 5
					
					
					

Rysunek 3.1: Przykład zróżnicowania stylów pisma różnych muzyków. Źródło: [19]

Na Rysunku 3.2 tabela przedstawia rodzaje elementów zbioru z podziałem na typowe grupy takie jak *nuty*, *klucze*, *metrum*...

Note	whole, half, quarter, eighth, sixteenth, thirty-second, sixty-fourth
Rest	whole/half, quarter, eighth, sixteenth, thirty-second, sixty-fourth
Accidentals	flat, sharp, natural, double sharp
Time signatures	common time, cut time, 4-4, 2-2, 2-4, 3-4, 3-8, 6-8, 9-8, 12-8
Clef	G-clef, C-clef, F-clef
Others	dot, barline

Rysunek 3.2: Przykład zróżnicowania stylów pisma różnych muzyków. Źródło: [19]

Zbiór danych buduje 15200 plików tekstowych rozdyskrebowanych do 100 niezależnych katalogów, z których każdy reprezentuje kolejnego ze 100 muzyków. Każdy z plików składa się z:

- etykiety danego symbolu (ang. *label*)
- jednej lub więcej linii tekstu reprezentujących pojedynczą linię symbolu

Ciągi liczb reprezentujących linie symboli są to współrzędne dwuwymiarowych punktów oddzielone przecinkami. Zestawy współrzędnych są zaś od siebie odseparowane średnikiem. Przerwanie linii tekstu oznacza rozpoczęcie rysowania linii w innym miejscu po "oderwaniu długopisu od kartki".

C-Clef

30,124;30,124;27,134;24,151;23,168;23,181;23,195;25,200;26,198;28,192;28,192;
31,158;31,158;33,152;42,143;49,138;55,134;61,131;64,128;62,129;62,129;
26,162;26,163;29,165;34,168;44,171;49,174;50,177;49,184;45,192;41,205;38,215;36,226;36,226;

Rysunek 3.3: Zawartość przykładowego pliku tekstowego zbioru HOMUS

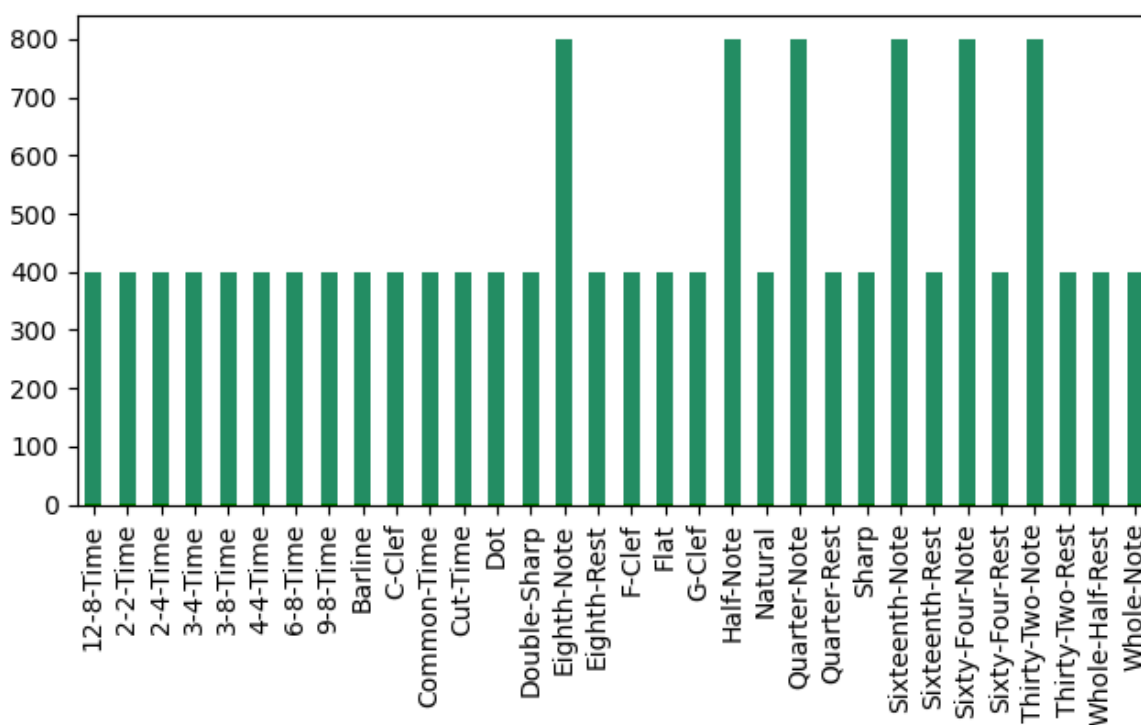
Na przykładzie z Rysunku 3.3 widać, że plik ten reprezentuje klucz C (ang. *C-Clef*). Kolejne trzy linie tekstu są zestawami współrzędnych kolejnych punktów na powierzchni dwuwymiarowej, które należy połączyć prostą linią, aby uzyskać rys figury. W tym przypadku odtworzony symbol wyglądałby tak, jak na Rysunku 3.4. Symbol ten może wydawać się niewyraźny, ponieważ zbiór reprezentują symbole o bardzo niewielkich wymiarach, często zamykających się w rozmiarze 50x100 pikseli. Służy to korzystnie niedużej wadze pobieranego ze strony internetowej pliku (cały dataset waży niespełna 10 MB).



Rysunek 3.4: Wizualna reprezentacja jednego z przykładowych plików tekstowych ze zbioru danych HOMUS

Rysunek 3.5 przedstawia wykres częstości występowania każdego z typów symboli muzycznych w wejściowym zbiorze danych. Widać, że większość rodzajów symboli reprezentowana jest w ilości dokładnie 400 egzemplarzy, zaś niewielka pozostała część w ilości 800. Wśród rodzajów najliczniejszych są same nuty: *półnuta*, *ćwierćnuta*, *ósemka*, *szesnastka*, *trzydziestka* *dwójka* i *sześćdziesiątka* *czwórka*. Nie ma wśród nich *całej nuty*.

Analiza składu zbioru wejściowego umożliwi w części eksperymentalnej sprawdzenie, czy liczność danego rodzaju symbolu w zbiorze jest skorelowana z efektywnością klasyfikatorów. Spodziewamy się osiągnąć wyższy poziom precyzji klasyfikatora w przypadku symboli o liczniejszej reprezentacji.



Rysunek 3.5: Wykres częstości występowania każdego z symbol w datasetcie

Rozdział 4

Implementacja programu

Większość dostępnych w internecie zbiorów danych przeznaczonych do systemów przetwarzania obrazów, np. za pomocą uczenia maszynowego, jest w formie niewielkich rozmiarów grafik w formacie jpg, png lub tiff. Zbiór danych HOMUS zaś przedstawiał enigmatyczną reprezentację w formie plików tekstowych.

Dlatego też pierwszym etapem pracy była analiza datasetu i jego konwersja na format bliższy ludzkiej intuicji niż ciąg liczb, przecinków i średników. O szczegółach tego podejścia czytamy w podrozdziale 4.1.

W drugiej kolejności postanowiono podejściu do zbioru danych bardziej organicznie i pozwolić klasyfikatorom uczyć się na niemal niezmienionej formie danych z plików wejściowych. Szczegóły z tych operacji znajdziemy w podrozdziale 4.2.

Implementacja kodu została napisana w języku Python 3.8.3. Użyto m.in. bibliotek *Keras*, *Pillow*, *Numpy*, *Pandas*, *Matplotlib*, *Pygame* i *Tensorflow*. Badania przeprowadzone zostały na komputerze z procesorem Intel Core i7-8750, 16 GB RAM.

W skład programu wchodzi kilka plików o rozszerzeniu *py*. W pierwszej kolejności należy uruchomić skrypty *noteConverter* oraz *noteConverter_raw* w celu konwersji danych z macierzystej zbioru danych HOMUS na format odczytywalny przez klasyfikatory. Skrypty *testTheModels* i *testYourOwnNeuralNetwork* służą odpowiednio uczeniu gotowych klasyfikatorów i własnej sieci neuronowej. W celu ich uruchomienia, należy otworzyć je w dowolnym środowisku pythonowym, skompletować zestaw potrzebnych bibliotek i skompilować.

4.1. Przygotowanie zbioru danych

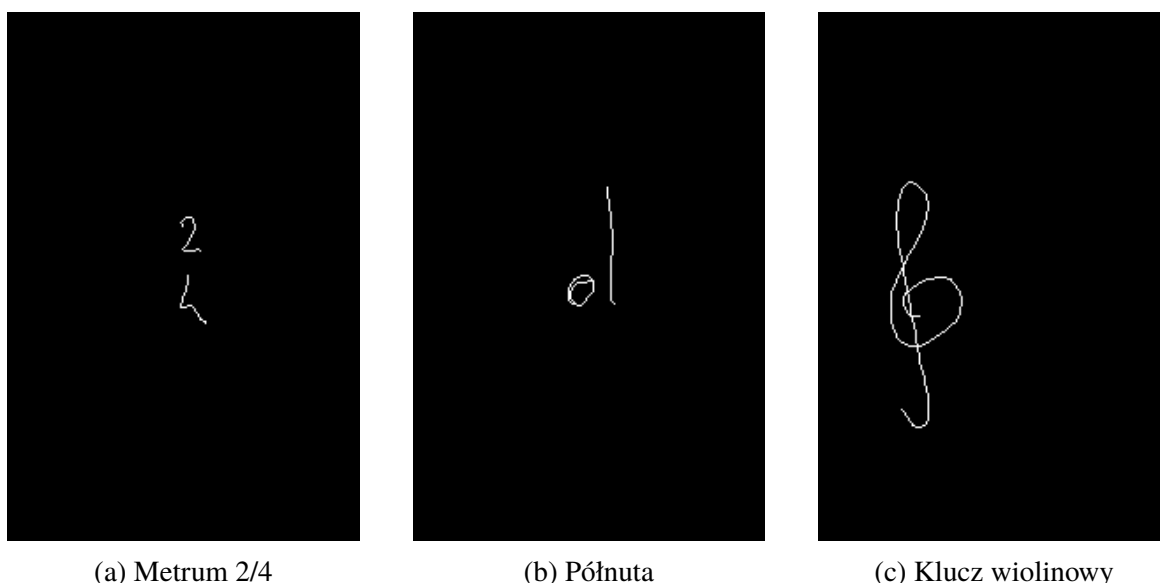
Jako, że sieci neuronowe dobrze radzą sobie z macierzami danych, postanowiono przemianować dane wejściowe na dwie listy - listę etykiet oraz listę macierzy zer i jedynek reprezentujących kolejne piksele grafiki wizualizującej wygląd symbolu muzycznego. Do tego celu jako stan pośredni pomiędzy plikiem tekstowym a listą macierzy użyto formatu graficznego png.

Poniższy kod źródłowy ukazuje metodę *SaveTxtToPng*, która odczytuje z oryginalnego pliku tekstowego jego treść, przechwytuje etykietę danego symbolu, dodając ją do listy *labels*, a następnie w pętli po wszystkich kolejnych liniach pliku rozdziela następne porcje danych na rozdzielne współrzędne dwuwymiarowych punktów. Za pomocą biblioteki *pygame* rysuje linie łączące punkty o danych współrzędnych. Zakończony obraz zapisuje do pliku png w lokalizacji początkowego pliku tekstowego. Dzięki tej operacji szybko przekonaliśmy się o wiarygodności datasetu. Rysunek 4.1 ukazuje przykładowy wynik kompilacji metody *SaveTxtToPng*.

Kod źródłowy

```
def SaveTxtToPng(filepath, labels):
    pointsTable = []
    screen = pygame.display.set_mode((200, 300))
    screen.fill((0, 0, 0))

    with open(filepath, 'r') as source:
        labels.append(source.readline().replace('\n', ''))
        for line in source:
            line = line[:-2]
            pointsTable = line.split(";")
            for i in range(0, len(pointsTable)-2):
                x1, y1 = pointsTable[i].split(",")
                x2, y2 = pointsTable[i+1].split(",")
                pygame.draw.line(screen, (255, 255, 255), (int(x1), int(
                    ↪ y1)), (int(x2), int(y2)))
    pygame.image.save(screen, subdir + os.sep + filename+".png")
```

(a) Metrum 2/4

(b) Półnuta

(c) Klucz wiolinowy

Rysunek 4.1: Wizualizacja nut po konwersji z formatu na tekstowego na graficzny

Obrazy ukazują białe figury na czarnym tle, by w następnym kroku stworzona została lista macierzy zer i jedynek, w których białe piksele miałyby swoją reprezentację za pomocą “1”, a czarne za pomocą “0”.

Kod źródłowy

```
def PngToMatrix(subdir, filename):
    source = Image.open(subdir + os.sep + filename+".png").convert('L'
        ↪ )
    matrix = np.array(source)

    for row in range(len(matrix)):
        for col in range(len(matrix[0])):
            if not matrix[row][col]==0:
                matrix[row][col] = 1
    return matrix
```

Operację tę wykonuje metoda *PngToMatrix*, która każdy znaleziony plik z rozszerzeniem png konwertuje na macierz zer i jedynek.

Jak widać po samych obrazach na Rysunku 4.1, symbole są różnych rozmiarów i znajdują się w różnych częściach przestrzeni dla niej zarezerwowanych, dlatego w następnych krokach wywołano metodę *FindBoundaries*, która zwraca cztery wartości traktowane jako równania prostych pionowych (z lewej i prawej strony symbolu) oraz poziomych (na dole i górze symbolu) wyznaczających granice symbolu na czarnej płaszczyźnie.

Kod źródłowy

```
def FindBoundaries(matrix):
    left = len(matrix)
    right = 0
    top = len(matrix[0])
    bottom = 0

    for row in range(len(matrix)):
        for col in range(len(matrix[0])):
            if matrix[row][col] == 1:
                if col < left:
                    left = col
                if col > right:
                    right = col
                if row < top:
                    top = row
                if row > bottom:
                    bottom = row
    return left, right, top, bottom
```

(a) Macierz przed docięciem

(b) Macierz po docięciu

Za pomocą granicznych wartości, macierz jest docinana do wymiaru tak, żadnej z czterech stron nie było pasu samych zer. Rysunek 4.2 ukazuje ładową macierz przed docięciem oraz przykładową macierz po docięciu wymiaru.

W kolejnym kroku zapewniamy, by do przyszłej sieci neuronowej trafiły paczki danych o identycznych rozmiarach. Znając wymiary przyciętych macierzy, które aktualnie mają różne wymiary, ustala się największe z nich tj. największą wysokość i największą szerokość tak, aby w następnym kroku sprawić każdą z macierzy tego samego wymiaru. Te macierze, których wymiary są mniejsze niż największe ustalone, dopełnia się zerami z dołu i z prawej strony. Tym sposobem macierze wszystkich symboli muzycznych będą miały te same wymiary, dzięki czemu do sieci neuronowej trafią macierze o identycznych wymiarach. Jest to wymóg.

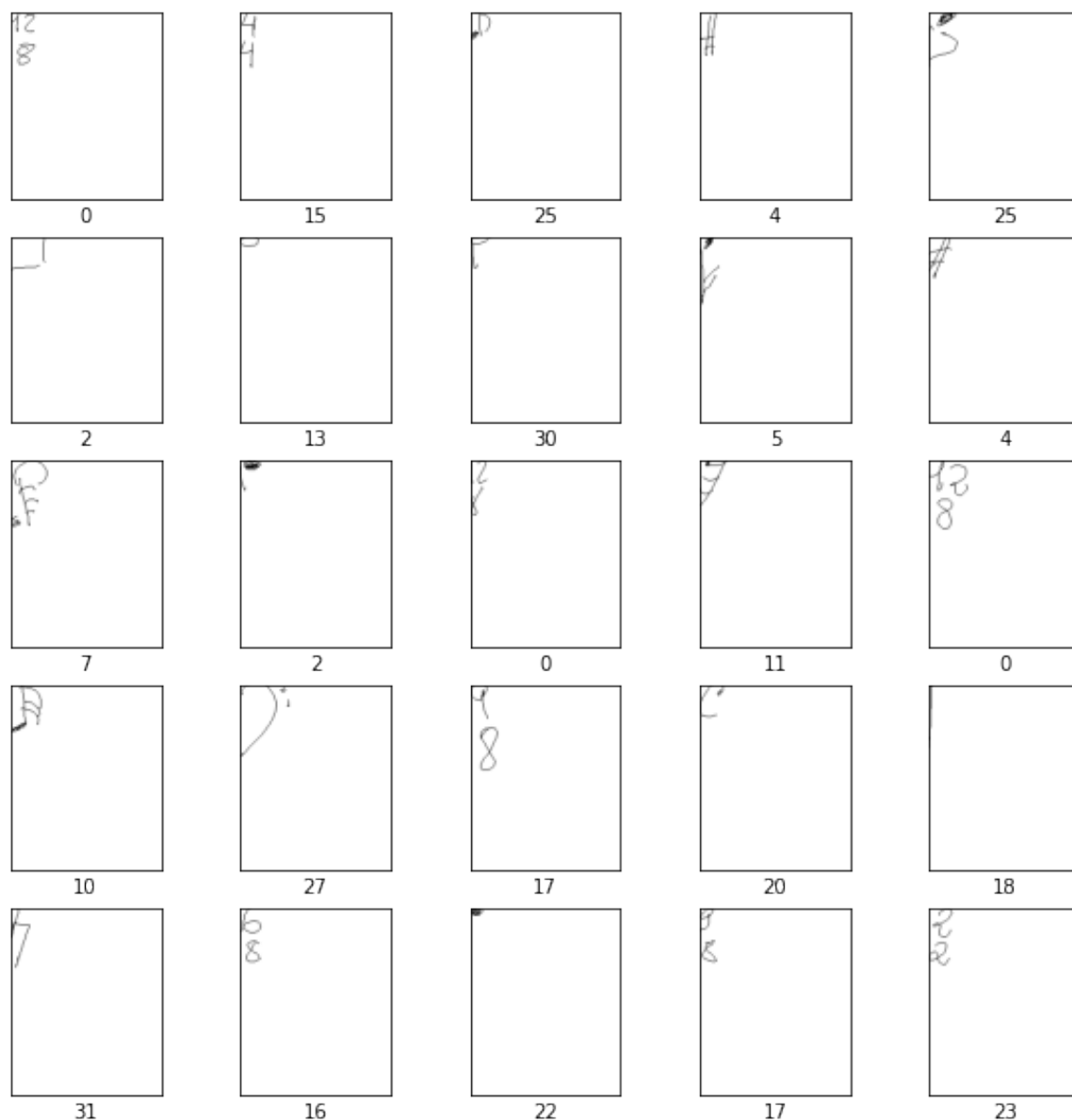
Tworzona jest nowa macierz trójwymiarowa X wypełniana zerami, której konkretne komórki modyfikuje się zamianą na jedynkę, jeśli macierz danego symbolu reprezentuje w tym miejscu jedynkę.

Kod źródłowy

```
X = np.zeros((len(notes), maxHeightCropped, maxWidthCropped), dtype=
    ↪ int)

for i in range(len(notes)):
    for j in range(len(notes[i])):
        for k in range(len(notes[i][j])):
            X[i][j][k] = notes[i][j][k]
```

Aby zweryfikować rzetelność zbioru danych, wyświetlamy pierwsze nuty wraz z ich etykietami. Na rysunku 4.3 zauważa się zróżnicowanie symboli ze względu na kształt i rozmiar, ale także niepokojąco dużą ilość pustej przestrzeni - zer w macierzy. Wynika ona z tego, iż większość symboli była o wiele mniejsza niż ten jeden lub dwa o największej szerokości i wysokości, które to ustanowiły wielkość przestrzeni na tak dużą.



Rysunek 4.3: 25 przykładowych macierzy symboli muzycznych wraz z ich etykietami

4.2. Wykorzystanie gotowych klasyfikatorów

Do implementacji algorytmów uczenia maszynowego użyto sześciu gotowych klasyfikatorów biblioteki *Keras*. Są wśród nich:

- regresja logistyczna (ang. *Logistic Regression*)
- liniowa analiza dyskryminacyjna (ang. *Linear Discriminant Analysis*)
- k-najbliższych sąsiadów (ang. *K-Nearest Neighbours*)
- drzewo decyzyjne (ang. *Decision Tree*)
- naiwny Bayes (ang. *Naive Bayes*)
- maszyna wektorów nośnych (ang. *Support Vector Machine*)

Kod źródłowy

```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size
    ↪ =0.30, shuffle=True, random_state=42)

nsamples, nx, ny = X_train.shape
X_train_2D = X_train.reshape((nsamples, nx*ny))
```

W pierwszej kolejności jednak cały zbiór danych *X* (macierze symboli) i *Y* (etykiety symboli) dzieli się na dwa podzbiory - uczący i testowy. W tym przypadku w stosunku 3:7.

Metodzie *train_test_split* przekazuje się argument *shuffle=True*, dzięki czemu początkowy zbiór w trakcie podziału zostaje wymieszany.

Następnie trójwymiarową tablicę macierzy symboli “spłaszcza się” do dwuwymiarowej, gdzie drugi wymiar równa się iloczynowi drugiego i trzeciego wymiaru tablicy początkowej.

Kod źródłowy

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

models = []
models.append(('LR', LogisticRegression(max_iter = 700)))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
results = []
names = []
scoring = 'accuracy'

for name, model in models:
    kfold = model_selection.KFold(n_splits=10, shuffle=True,
    ↪ random_state=seed)
    cv_results = model_selection.cross_val_score(model, X_train_2D,
    ↪ Y_train, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
```

Powyższy kod realizuje uczenie każdego modelu w pętli po wszystkich elementach tablicy *models* za pomocą metody *cross_val_score*, która za wyznacznik przyjmuje *accuracy* i wewnętrznie dzieli zbiór danych według wyniku kompilacji metody *KFold*.

Jednakże nie wszystkie klasyfikatory udało się uruchomić z ich domyślnymi parametrami. *LogisticRegression* od samego początku zwracało ostrzeżenie, jakoby argument *max_iter* reprezentował zbyt małą wartość. Klasyfikatorowi nie udawało się przeanalizować zbioru danych i jako wynik końcowy zwracał *nan*. Drogą prób i błędów doszło się do optymalnej wartości *max_iter* równej 700, która wystarczała algorytmowi na płynną pracę. *Linear Discriminant Analysis* z kolei zwracał błędy z powodu za małej ilości RAMu komputera (do dyspozycji miał 16 GB RAMu). Mimo wielu prób naprawy błędu, nie udało się doprowadzić klasyfikatora do stanu pełnej kompilacji, dlatego w dalszych badaniach pominięto go.

4.3. Zbudowanie własnej sieci neuronowej

Głównym celem zbudowania własnej sieci neuronowej na potrzeby eksperymentów w ramach tej pracy naukowej było stworzenie sieci konkurencyjnej dla najlepszych klasyfikatorów spośród wybranych. Liczyła się miara precyzji oraz czas wykonania algorytmów.

Poniższy kod źródłowy prezentuje budowę modelu uczącego. Typem sieci neuronowej z biblioteki *Keras* jest sieć *Sequential*, która znana jest z wysokich wyników w przypadku pojedynczego inputu (w naszym przypadku jest nim pojedyncza macierz zer i jedynek reprezentująca symbol) oraz pojedynczego outputu (jest nim skonkretyzowana jedna klasa decyzyjna, do której model przypisuje dany przykład).

Kod źródłowy

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=X[0].shape),
    keras.layers.Dense(128, activation='sigmoid'),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(len(withoutDuplicates), activation='softmax')
])
```

Model zbudowany jest z czterech warstw. Pierwszą z nich jest *Flatten* która konwertuje dwuwymiarową tablicę o wymiarach $A \times B$ na jednowymiarową tablicę pikseli o długości $A*B$, gdzie w naszym w przypadku A i B to wysokość i szerokość matrycy każdej z nut. Dwie kolejne warstwy są warstwami neuronów. *128* i *64* to liczba neuronów w każdej z warstw, wartości parametru *activation* z kolei są to funkcje aktywacji dla danych warstw. Ostatnią warstwą jest warstwa rozdzielająca wynik prawdopodobieństwa na tyle wyjść, ile jest elementów listy *withoutDuplicates*. Reprezentuje ona listę etykiet wszystkich symboli bez powtórzeń. Model zwracać ma 32 wyniki probabilistyczne sumujące się do 1. Funkcja aktywacji *softmax* jest w tym przypadku funkcją normalizacji.

Zanim model będzie gotowy do uczenia, potrzebuje kilka dodatkowych ustawień. Są one dodawane podczas kompilacji:

- *loss function* - mierzy ona, jak precyzyjny jest model podczas uczenia się
- *optimizer* - sposób aktualizacji modelu na podstawie danych, które widzi oraz funkcji straty
- *metrics* - służy do monitorowania kolejnych etapów uczenia i testowania. Bieżący przykład używa stopnia precyzji `textitaccuracy`, czyli miary, z jaką nuty są poprawnie rozpoznawane

Kod źródłowy

```
model.compile(optimizer='adam',  
              loss='sparse_categorical_crossentropy',  
              metrics=['accuracy'])
```

Aby rozpocząć uczenie, wołamy metodę *fit*, która swoją nazwę zawdzięcza dobieraniu (ang. *fitting*) modelu pod dostarczony zbiór danych. W przypadku samodzielnie budowanych sieci neuronowych, jesteśmy w stanie lepiej monitorować przebieg uczenia na każdej z epok, dlatego też pozwolono sieci przeprowadzić analizę danych na aż 200 epokach w celu stworzenia wykresu precyzji od liczby epok (Rysunki 5.8 i 5.12).

Kod źródłowy

```
history = model.fit(X_train, Y_train, validation_split=0.10, epochs  
    ↪ =200, batch_size=10, verbose=1)
```

Metoda *fit* implementuje technologię 10-krotnej walidacji krzyżowej, która służy do określania jakości modelu już w trakcie jego adaptacji, w celu wyeliminowania problemu przeuczenia się.

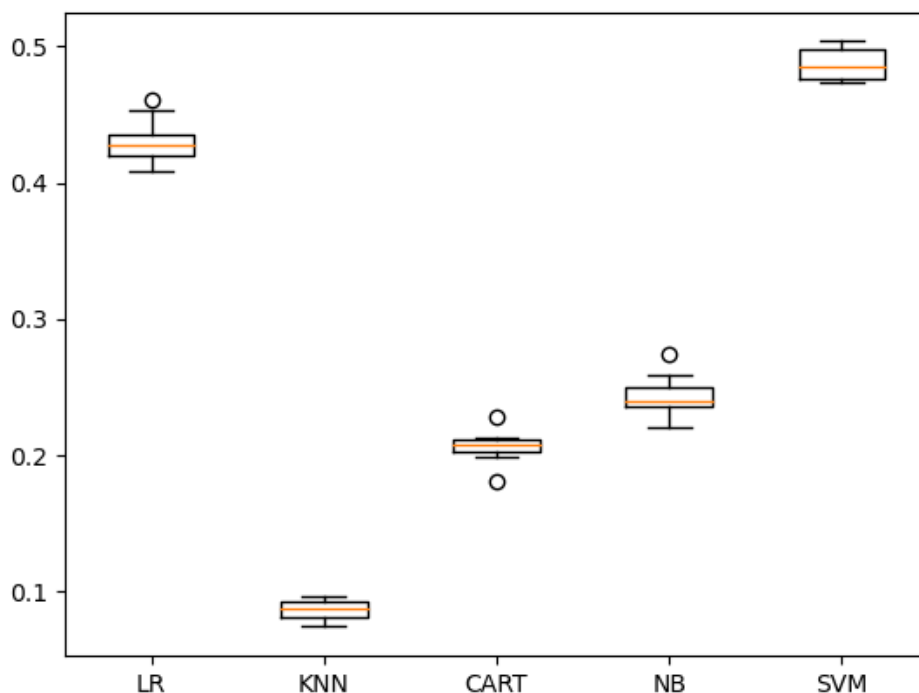
Rozdział 5

Wyniki

Rozdział ten podzielony został na dwie wyraźne części - podrozdział 5.1. prezentuje wyniki badań na gotowych klasyfikatorach biblioteki *Keras*. Sekcje 5.1.1. oraz 5.1.2 pokazują rezultaty pracy algorytmów na odpowiednio przekonwertowanych i nieprzekonwertowanych zbiorach danych wejściowych. Podrozdział 5.2. zaś ukazuje wyniki uzyskane z kompilacji własnej sieci neuronowej.

5.1. Wyniki testów na gotowych klasyfikatorach

5.1.1. Z konwersją danych wejściowych



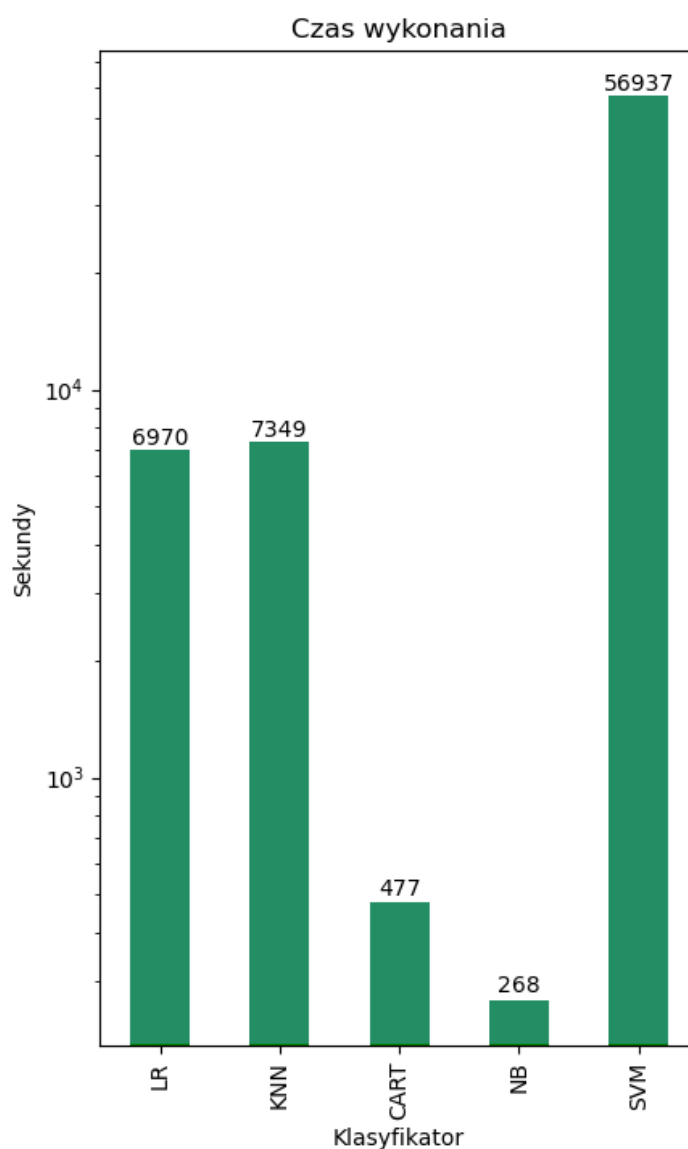
Rysunek 5.1: Wykres skrzynkowy precyzji osiągniętych przez klasyfikatory wyników

Rysunek 5.1 prezentuje wykres skrzynkowy użykanych z uczenia wszystkich modeli wyników precyzji. Na pierwszy rzut oka widać, że zdecydowanie najlepsze z nich osiągnęły klasyfikatory *Logistic Regression* z wynikiem ok. 42% oraz *Support Vector Machine* z wynikiem ok. 49%. Klasyfikatory *Decision Tree* i *Naive Bayes* uzyskały wyniki *accuracy* na poziomie ok. 22%. Najgorzej wypadł zaś *K-Nearest Neighbours* z wynikiem ok. 9% skuteczności.

Dwa najlepsze klasyfikatory (*LR* i *SVM*) osiągnęły, jak na swoje domyślne parametry, bardzo wysokie wyniki. Czy wynik 50% jest satysfakcjonujący? W wymiarze tego, że klas decyzyjnych (rodzajów wszystkich symboli w zbiorze danych) było aż 32, osiągnięcie skuteczności na poziomie połowy zbioru danych jest imponujące. Z drugiej zaś strony, biorąc pod uwagę, że klas decyzyjnych było 32, to hipotetyczny algorytm przyporządkowujący kolejnym symbolom etykiety w sposób absolutnie losowy, osiągnąłby skuteczność na poziomie ok. 3%, co nie wydaje się być tak dalekim wynikiem od tego, jaki osiągnął nasz najmniej skuteczny klasyfikator (*KNN*).

Miara precyzji jednakowoż nie jest jedynym czynnikiem brany pod uwagę przy ocenie wydajności danego algorytmu uczenia maszynowego. Drugim, równie ważnym parametrem, jest czas jego wykonania.

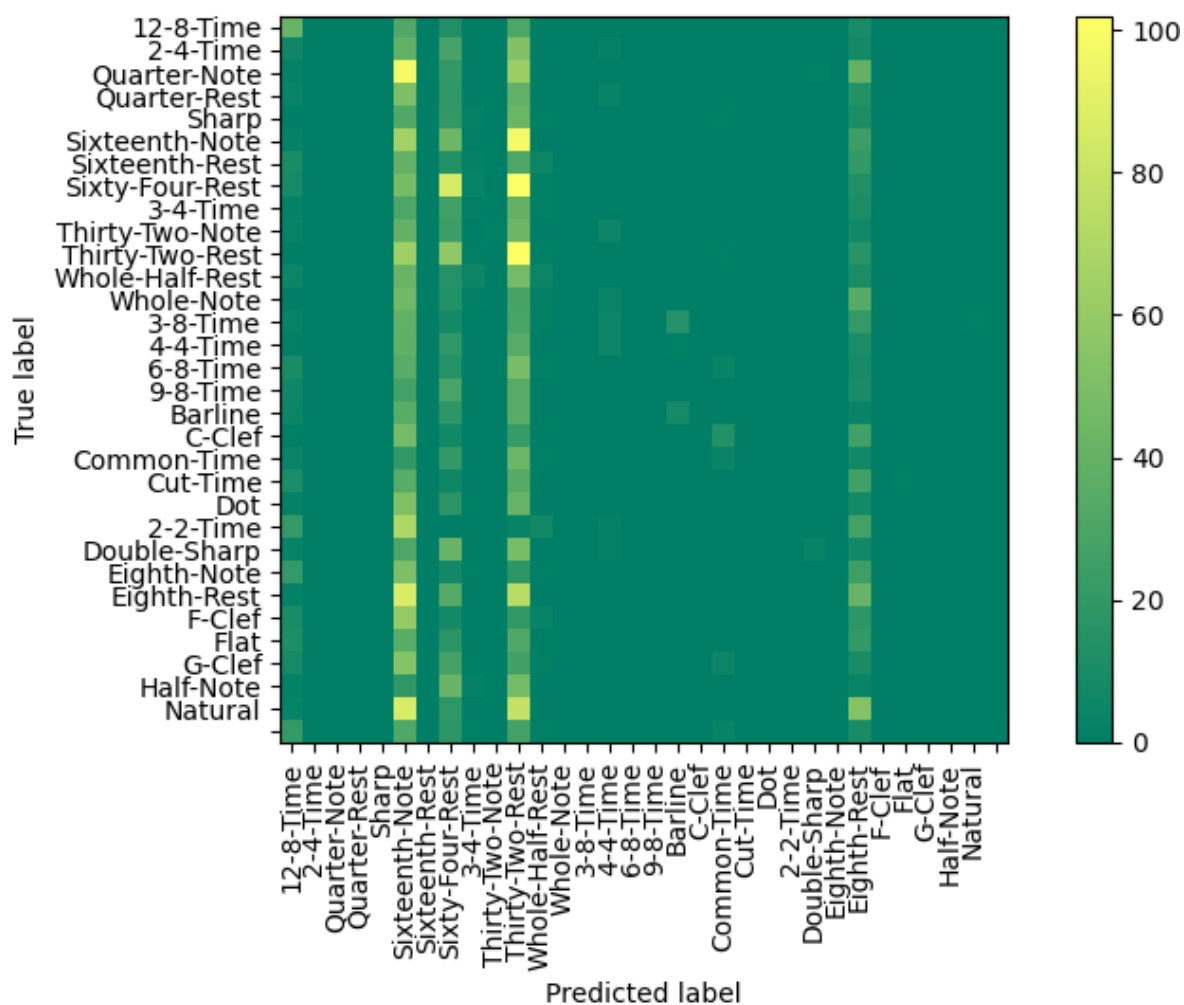
Rysunek 5.2 ilustruje czas uczenia przez każdy z klasyfikatorów wyrażony w sekundach. Warto mieć na uwadze, że pionowa oś czasu jest logarymiczna. Wartości znajdujące się na szczytach słupków prezentują dokładne wartości osiągniętych czasów.



Rysunek 5.2: Czasy wykonywania algorytmów















Przeanalizujemy w parach kolejne klasyfikatory od najefektywniejszych:

- Podczas gdy osiągający wynik ok. 42% *Logistic Regression* potrzebował niespełna 2 godzin (6970 sekund), tak *Support Vector Machine* osiągnął rekordowy wynik ok. 49% w czasie ponad 8-krotnie dłuższym (prawie 16 godzin).
- Dwa kolejne pod względem osiągniętej, bardzo podobnej, skuteczności klasyfikatory - *Decision Tree* i *Naive Bayes* - potrzebowały na kompilację również podobną do siebie ilość czasu - 477 sekund (8 minut) i 268 sekund (4,5 minuty).



Rysunek 5.3: Macierz pomyłek SVM dla danych przekonwertowanych

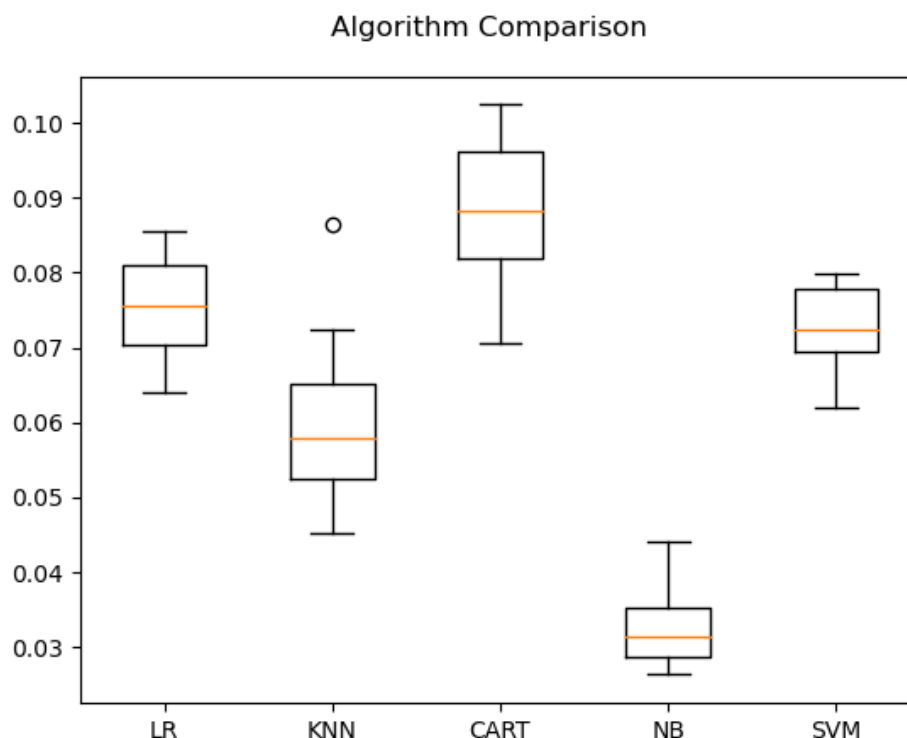
W ramach analizy najskuteczniejszego klasyfikatora (*Support Vector Machine*), stworzono macierz pomyłek (Rysunek 5.3) - tablicę o wymiarach 32 x 32 (32 jest to ilość klas decyzyjnych), która prezentuje, które z klas w jakim stopniu były mylone z innymi i jak często modelowi udało się rozpoznać symbol poprawnie.

Rest symbols		
Rest	Time value	Note
	Whole-note rest	
	Half-note rest	
	Quarter-note rest	
	Eighth-note rest	
	Sixteenth-note rest	
	Thirty-second-note rest	
	Sixty-fourth-note rest	

Rysunek 5.4: Podobieństwo symboli muzycznych. Źródło: [24]

Zauważalnym jest, iż model w dużym stopniu obstawiał symbole *Sixteenth-Note*, *Sixty-Four-Rest*, *Thirty-Two-Rest* i *Eighth-Rest*. Gdy skonfrontować wynik ten z wykresem słupkowym częstości występowania danych symboli w zbiorze danych (Rysunek 3.5), niedostrzegalne jest żadne powiązanie między nimi, ale okazuje się, że symbole najczęściej predykowane przez model oraz te, które w macierzystym zbiorze danych pojawiają się najczęściej są do siebie bardzo podobne graficznie (Rysunek 5.4), dlatego modelowi bardzo łatwo jest je ze sobą pomylić.

5.1.2. Bez konwersji danych wejściowych



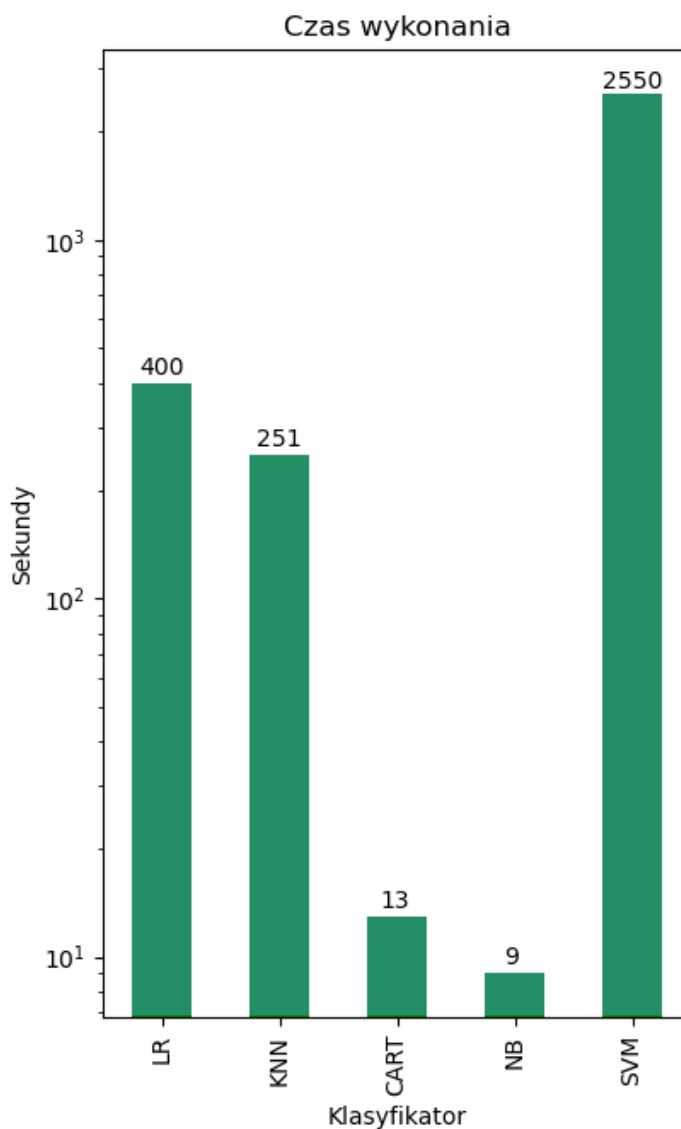
Rysunek 5.5: Wykres skrzynkowy precyzji osiągniętych przez klasyfikatory wyników

Na Rysunku 5.5 w formie wykresu skrzynkowego przedstawione zostały wyniki kompilacji uczenia wszystkich modeli na nieprzekonwertowanych danych wejściowych. Zauważa się znaczny spadek precyzji względem uczenia modeli na przekonwertowanych danych. Znamienne jest również, iż klasyfikatory, które z uprzednim uczeniem na danych przekonwertowanych osiągały blisko 50% skuteczności rozpoznawania (były nimi *LR* i *SVM*), tym razem uzyskały wyniki przeciętne na tle innych klasyfikatorów w przypadku uczenia na niezmiennych danych.

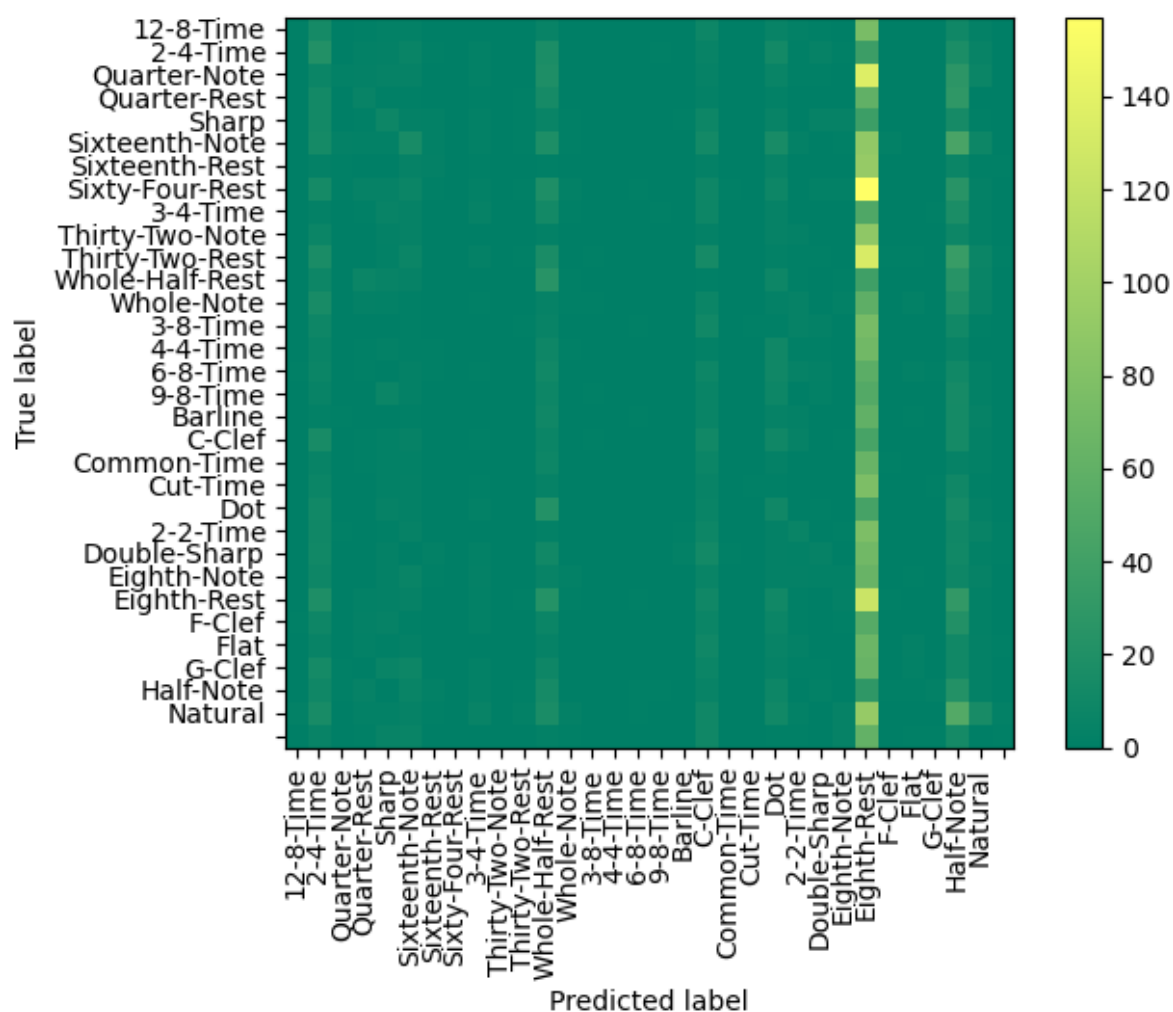
Można powiedzieć, że żaden z modeli nie osiągnął wybitnie wysokiego wyniku, gdyż ani jeden z nich nie przekroczył pułapu wartości precyzji na poziomie zaledwie 9%, który to wynik można interpretować jako relatywnie bliski statystycznemu wynikowi algorytmu losowego (ok. 3%).

Gdy zaś przeanalizujemy wykres słupkowy (Rysunek 5.6) czasu potrzebnego na uczenie każdego z modeli, zauważamy uderzające podobieństwo stosunków wartości każdego klasyfikatora do stosunków wartości z uczenia na danych przekonwertowanych.

Ponownie *Support Vector Machine* kompilował się ok. 7-krotnie dłużej od *Logistic Regression* oraz *K-Nearest Neighbours*, a *Decision Tree* i *Naive Bayes* osiągnęły niemal identyczne czasy odpowiednio 13 i 9 sekund. Czasy uczenia poszczególnych modeli wydają pozostawać w niemal stałych stosunkach proporcjonalności.



Rysunek 5.6: Czasy wykonywania algorytmów



Rysunek 5.7: Macierz pomyłek SVM dla danych niekonwertowanych

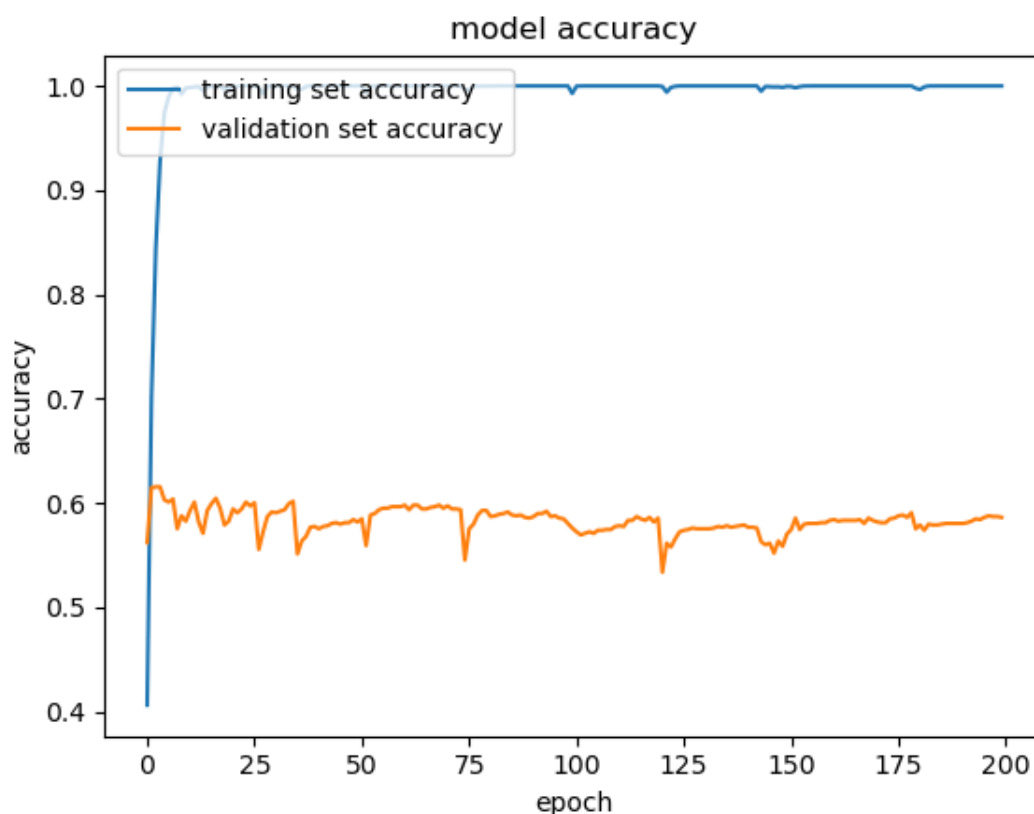
W odróżnieniu do macierzy pomyłek w uczeniu na danych przekonwertowanych, nowa macierz (Rysunek 5.7) rozbłyska jasnym pionowym pasem na wysokości symbolu *Eighth-Rest*, który to model predykuje jako właściwy label zdecydowanie najczęściej.

5.2. Wyniki testów na własnej sieci neuronowej

Nasza sieć, choć niezbyt rozbudowana jak na możliwości programistyczne w kreacji sieci neuronowych, okazała się spełnić postawione oczekiwania. Osiągnęła w relatywnie krótkim czasie uderzająco wysoki wynik precyzji rozpoznawania symboli muzycznych zbioru danych. Zarówno przekonwertowanego na postać macierzy zer i jedynek jak i nieprzekonwertowanego, surowego.

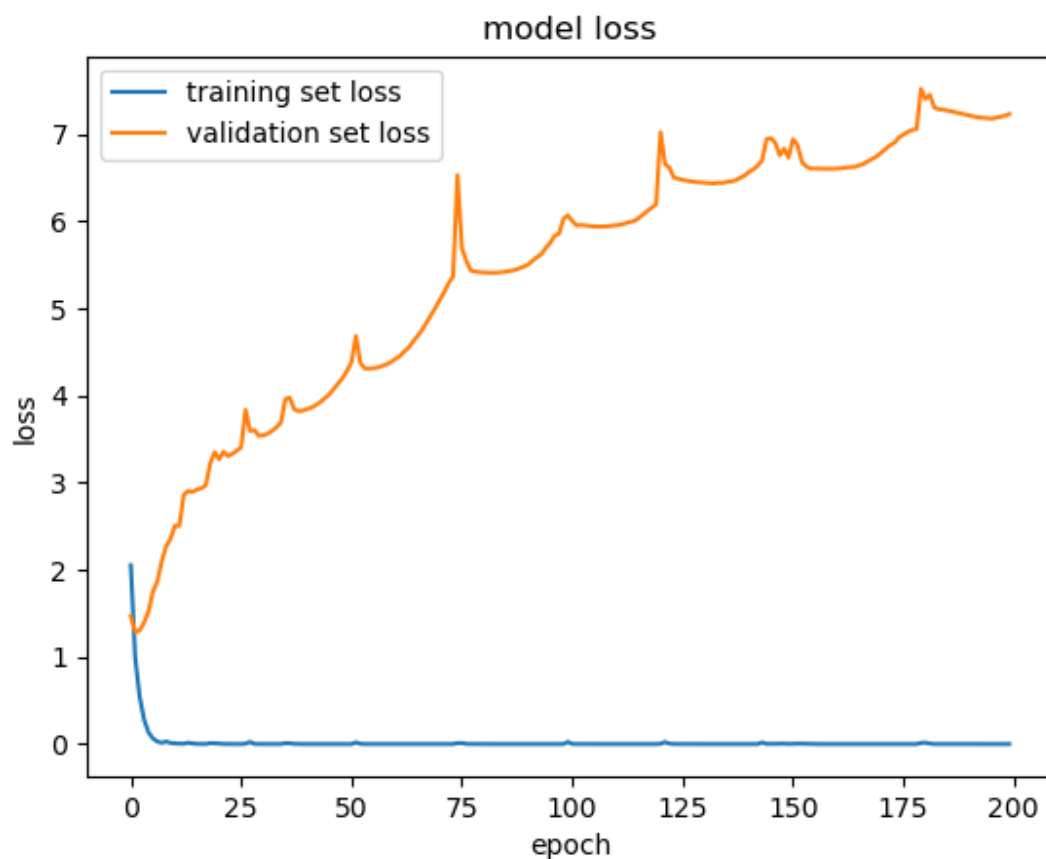
5.2.1. Z konwersją danych wejściowych

W ramach obserwacji postępów nauczania wraz ze wzrostem liczby epok, stworzono wykres prezentujący efekty pracy algorytmu (Rysunek 5.8). Widoczne są na nim krzywe precyzji zbioru uczącego (linia niebieska) i zbioru walidacyjnego (linia pomarańczowa). Znamiennym jest niemal błyskawiczne (w ciągu pierwszych 10 epok) osiągnięcie *accuracy* zbioru uczącego na poziomie blisko 100%. To nie pozwoliło modelowi na znaczny wzrost precyzji na zbiorze testowym. Model wydaje się osiągać maksymalną precyzję w okolicy trzeciej epoki, po czym w trakcie kolejnych 197 epok utrzymuje nieco niższą, stałą wartość ok. 55% precyzji rozpoznawania na zbiorze walidacyjnym.



Rysunek 5.8: Wykres precyzji od liczby epok z danymi przekonwertowanymi

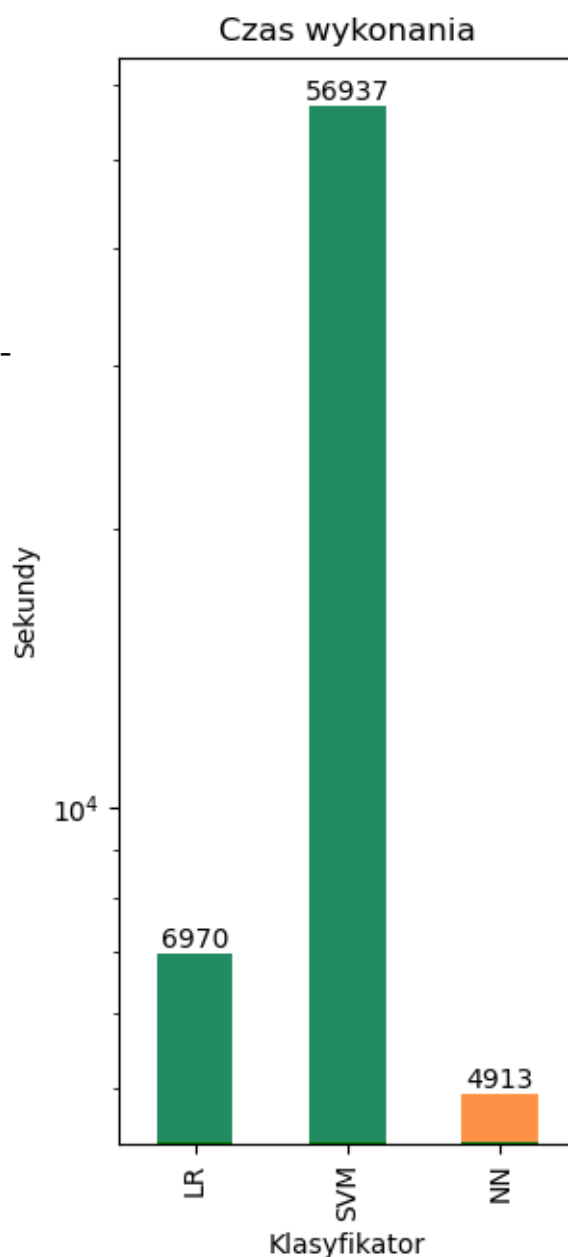
Wynikiem analizy uczenia modelu jest również wykres funkcji straty od liczby epok (Rysunek 5.9). Wartość straty udaje się minimalizować praktycznie do zera w ciągu pierwszych 10 epok. Wartość ta jednak wzrasta funkcją bliską logarytmicznej w przypadku zbioru testowego.



Rysunek 5.9: Wykres strat od liczby epok z danymi przekonwertowanymi

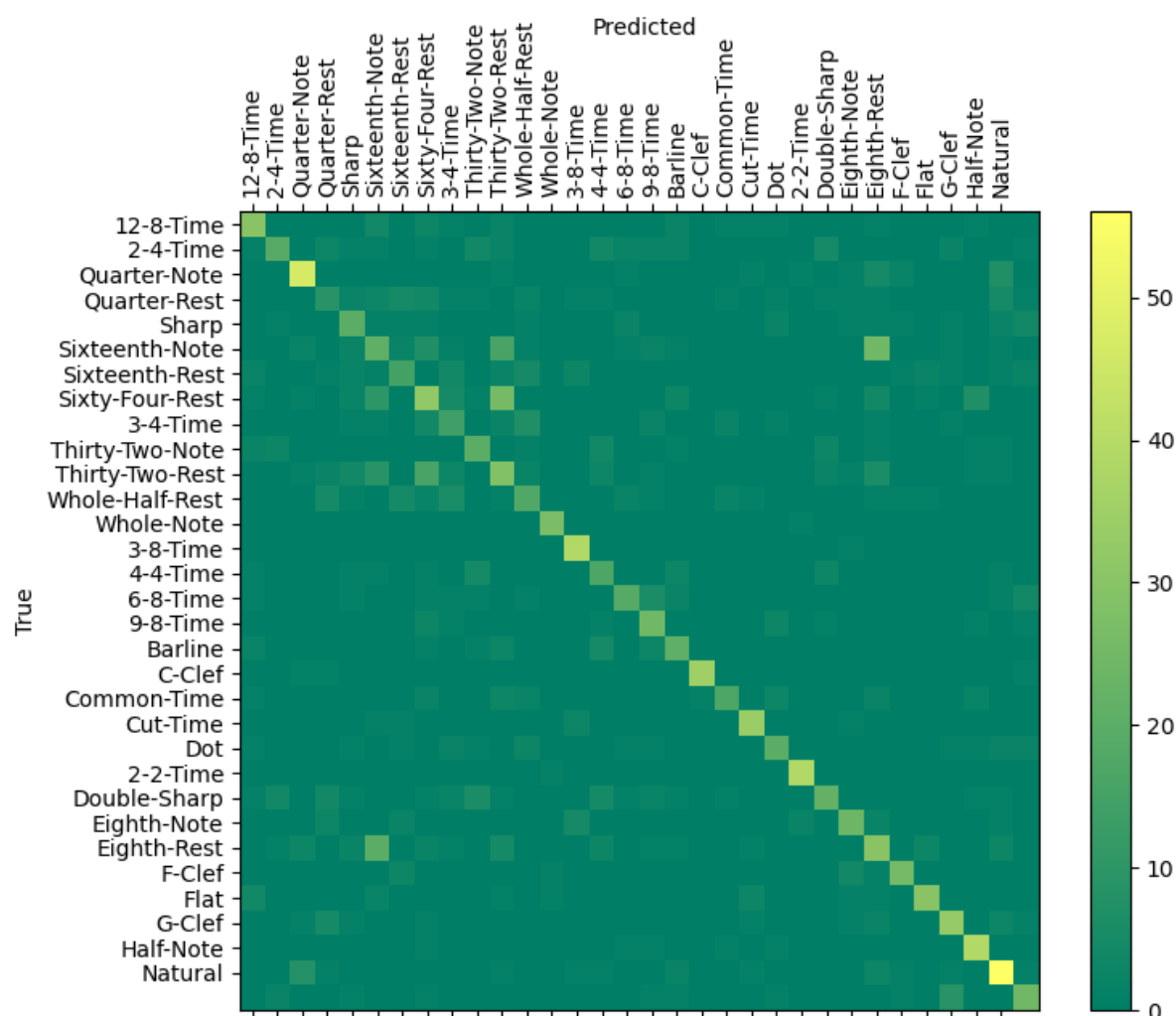
Porównajmy ze sobą czasy wykonania algorytmów uczenia dwóch najskuteczniejszych przetestowanych klasyfikatorów oraz własnej sieci neuronowej (Rysunek 5.10).

Dostrzegalnym jest widoczna różnica w pomiędzy klasyfikatorem *Support Vector Machine* oraz własną siecią (na wykresie zaznaczona jako *NN - NeuralNetwork*, mając na uwadze, iż *SVM* osiągnął wynik precyzji ok. 48%, a własna sieć w szczycie formy (na poziomie ok. 3 epoki) aż 62%. Własna sieć neuronowa osiągnęła wynik o ok. 14 punktów procentowych wyższy w czasie prawie 12-krotnie krótszym.



Rysunek 5.10: Czasy wykonywania algorytmów z danymi przekonwertowanymi

Macierz nieporozumień na Rysunku 5.11 pokazuje nam, iż model sieci neuronowej, rozpoznając symbole błędnie, relatywnie często szacował, że symbolem może być *Sixteenth-Rest*, *Sixteenth-Note*, *Sixty-Four-Rest* lub *4-4-Time*. Wyraźnie jasne punkty poza przekątną kwadratu pokazują, że model często mylił ze sobą *Sixteenth-Note* z *Eighth-Rest* oraz *Thirty-Two-Rest* z *Sixty-Four-Rest*.

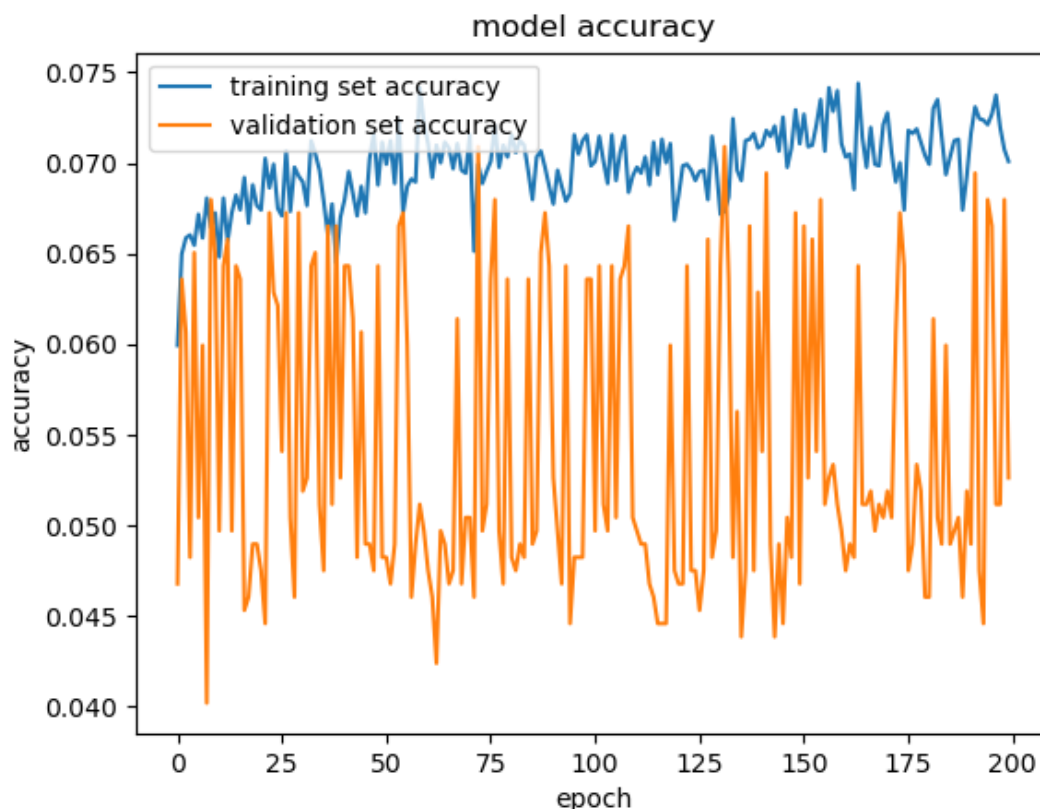


Rysunek 5.11: Macierz pomyłek własnej sieci dla danych przekonwertowanych

5.2.2. Bez konwersji danych wejściowych

W przypadku uczenia własnej sieci neuronowej danymi nieprzekonwertowanymi, wyniki precyzji okazały się nielepsze od wyników, jakie w przypadku danych nieprzekonwertowanych dał najskuteczniejszy gotowy klasyfikator (*CART*).

Wykres osiąganego *accuracy* od liczby epok uczenia (Rysunek 5.12) ukazuje nam fakt, iż przez całe trwanie uczenia, bez znaczenia, ile epok miało się przed sobą, sieć uporczywie walczyła o wyniki w dość gwałtowny sposób naprzemiennie tracąc i zyskując skuteczność. Widać, że w całym czasie działania algorytmu, sieć działała chaotycznie, bez planu, nie znajdując w żadnym momencie sensownych konsekwencji uczenia z poprzecnych epok. Osiągała ona podobne wyniki precyzji zarówno w okolicach epoki 10., jak i 75. czy 150. Finalnie nie przekroczyła pułapu precyzji na poziomie 8%, co w porównaniu z najskuteczniejszym w przypadku danych niekonwertowanych klasyfikatorem *Decision Tree*, który osiągnął ok. 9%, sprawia sieć w zasadzie bezużyteczną.



Rysunek 5.12: Wykres precyzji od liczby epok z danymi niekonwertowanymi

W odróżnieniu do funkcji straty sieci, gdy uczyła się na danych przekonwertowanych, tym razem Rysunek 5.13 ukazuje nam już nie logarytmiczną a hiperboliczną funkcję straty. Sieć jednak, choć dążyła do jej minimalizacji, utrzymywała ją w każdej epoce na bardzo wysokim, jak na sieć neuronową, poziomie.

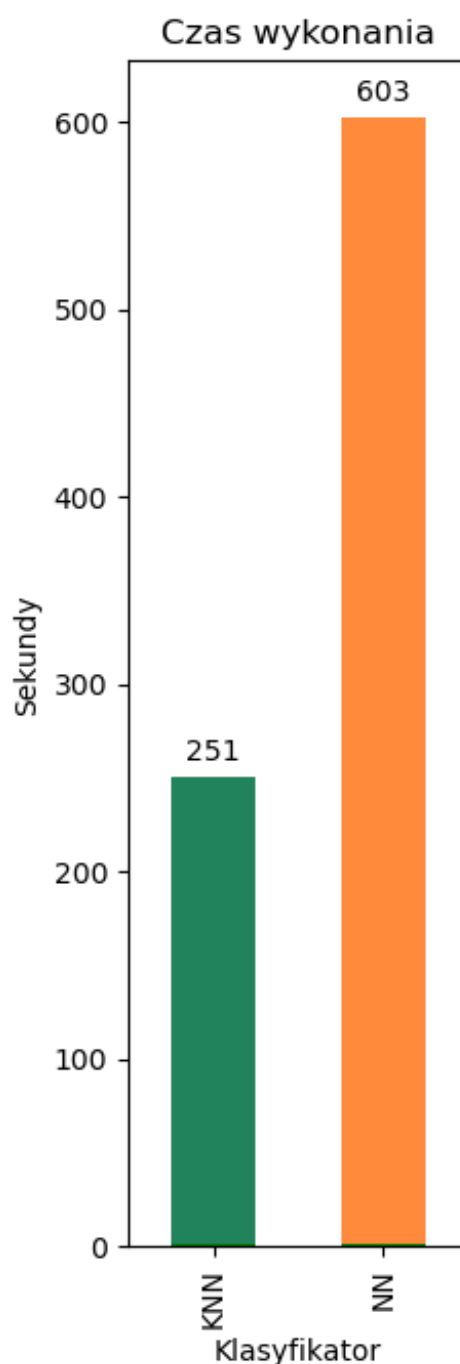


Rysunek 5.13: Wykres strat od liczby epok z danymi niekonwertowanymi

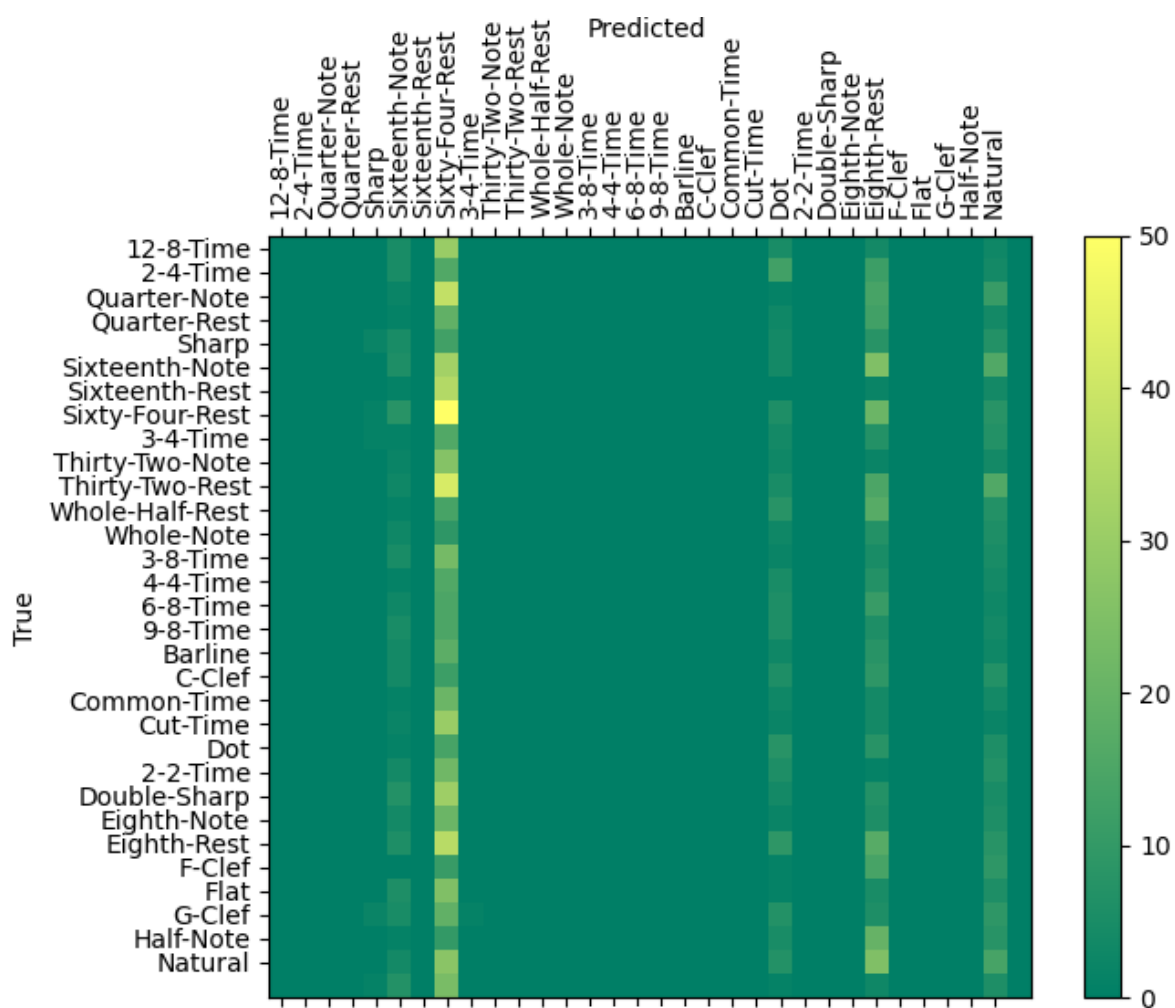
Raz jeszcze warto przyrzeć się długości czasu uczenia sieci neuronowej w porównaniu do gotowych klasyfikatorów. Rysunek 5.14 prezentuje czas kompilacji najlepszego spośród klasyfikatorów (*Decision Tree*) oraz własnej sieci neuronowej. Oś pionowa jest liniowa.

Zbudowanej sieci neuronowej (NN) zajęło ponad 2 razy więcej czasu uczenie na zbiorze danych nieprzekonwertowanych.

Macierz pomyłek z Rysunku 5.15 okazała się być bardzo podobna do macierzy pomyłek klasyfikatora SVM z uczenia na danych przekonwertowanych. Ponownie sieć z dużą częstością oceniała symbole jako *Sixty-Four-Rest*, *Sixteen-Note Eighth-Rest*, ale dodatkowo relatywnie często predykowane były symbole *Dot* i *Natural*. Raz jeszcze okazuje się, że model nie predykuje z większą częstotliwością tych symboli, które w początkowym zbiorze testowym pojawiają się najczęściej.



Rysunek 5.14: Czas wykonywania algorytmów z danymi niekonwertowanymi



Rysunek 5.15: Macierz pomyłek własnej sieci dla danych niekonwertowanych

Rozdział 6

Podsumowanie

W trakcie realizacji implementacji tego projektu, przekonaliśmy się, jak wiele przeszkód (głównie programistycznych, ale i ograniczeń technicznych komputera) stoi na drodze ku płynnej kompilacji systemów optycznego rozpoznawania symboli muzycznych.

Największym ograniczeniem okazał się czas - czas, którego klasyfikatory uczenia maszynowego potrzebują bardzo dużo. Wiele godzin, czasami wiele dni. Stawia to przed programistą nieznosne utrudnienie w postaci niemocy używania sprzętu w czasie uczenia modelu. Komputer jest maksymalnie przeciążony, dlatego jakakolwiek praca na nim jest niemożliwa. Zresztą nie byłoby profesjonalnym, porównywać czasy wykonania poszczególnych algorytmów, wiedząc, że w czasie pracy niektórych z nich komputer był dodatkowo obciążany, co jedynie wydłużałoby czas działania algorytmu.

Dowiedzieliśmy się, że niewielkim nakładem zarówno czasowym jak i programistycznym (o wiele krótszy kod implementacji) wyścig w najskuteczniejszym rozpoznawaniu odręcznych symboli muzycznych bezapelacyjnie wygrywa własnoręcznie zbudowana sieć neuronowa. Przegraną odnoszą klasyfikatory uczenia maszynowego, bo choć niektóre z nich osiągały niezłe, w porównaniu z własną siecią, wyniki precyzji, czas ich kompilacji musiał je dyskwalifikować. Tak oto hipoteza badawcza z rozdziału 1. została udowodniona.

Duży wkład niniejszej pracy polegał na dogłębnej analizie i prezentacji elementów zbioru HOMUS i propozycji jego wykorzystania w uczeniu maszynowym. Pomimo jego niegraficznego formatu udało się zwizualizować część symboli w postaci obrazu i przy okazji sprawdzono jego wiarygodność.

Jednym ze znaczących otwartych problemów jest brak wspólnych metod i mierników, które wymiennie porównałyby wyniki systemów OMR. Jest to bardziej skomplikowana kwestia niż mogłaby się na pierwszy rzut oka wydawać, gdyż systemy optycznego rozpoznawania symboli muzycznych mogą służyć różnym celom - odtwarzaniu dźwięków audio, archiwizacji wyników cyfrowych itp., a wyniki te mogą być przechowywane w różnych formatach.

Zauważamy także potrzebę szerszego edukowania autorów prac naukowych o metodach OMR w kierunku wiedzy muzycznej w celu wspierania procesów rozpoznawania i lepszego zrozumienia problematyki i jej zagadnień.

W tej pracy naukowej skupiliśmy się na porównaniu efektów uczenia gotowych klasyfikatorów uczenia maszynowego oraz własnej sieci neuronowej. Mam nadzieję, że nasz wkład zmotywuje kolejnych badaczy, uczonych, studentów do poszerzania wiedzy na temat systemów OMR i udoskonalania ich.

Dalsze badania nad problematyką OMR będą miały na celu poszukiwanie jak najbardziej optymalnych parametrów wywołania dla danych metod realizujących zagadnienie tak, by osiągały wyższe wyniki precyzji. W ramach pracy nad coraz krótszym czasem kompilacji modeli, zostaną podjęte kroki ku zrównolegleniu pewnych procesów uczenia maszynowego.

Bibliografia

- [1] A. Rebelo, I. Fujinaga, F. Paszkiewicz, A. Marcal, C. Guedes, J. Cardoso, *Optical Music Recognition: state-of-the-art and open issues*, International Journal of Multimedia Information Retrieval, 2012
- [2] D. Pruslin, *Automatic recognition of sheet music*, MIT, 1966
- [3] D. Prerau, *Computer pattern recognition of standard engraved music notation*, Massachusetts Institute of Technology Libraries, 1989
- [4] S. Ohteru, I. Kato, K. Shirai, S. Narita, *The Robot Musician 'WABOT-2* Department of Mechanical Engineering, Japan, 1987
- [5] D. Byrd, M. Schindele, *Prospects for improving OMR with multiple recognizers*, ISMIR, 2016
- [6] I. Knopke, D. Byrd, *Towards musicdiff: a foundation for improved optical music recognition using multiple recognizers*, Austrian Computer Society, 2007
- [7] E. Bugge, K. Juncherr, B. Mathiansen, *Using sequence alignment and voting to improve optical music recognition from multiple recognizers*, University of Miami, 2011
- [8] V. Padilla, A. McLean, A. Marsden, K. Ng., *Improving optical music recognition by combining outputs from multiple sources*, ISMIR, 2015
- [9] L. Pugin, T. Crawford, *Evaluating OMR on the early music online collection*, ISMIR, 2013
- [10] J. Burgoyne, L. Pugin, G. Eustace, I. Fujinaga, *A comparative survey of image binarisation algorithms for optical recognition on degraded musical sources*, Australian Computer Society, 2007
- [11] R. Jin, C. Raphael, *Interpreting rhythm in optical music recognition*, ISMIR, 2012
- [12] M. Church, M. Cuthbert, *Improving rhythmic transcription via probability models applied post-omr*, ISMIR, 2014
- [13] G. Vigliensoni, G. Burlet, I. Fujinaga, *Optical measure recognition in common music notation*, ISMIR, 2013
- [14] Y. Jiang, C. Raphael, *Instrument identification in optical music recognition*, ISMIR, 2015
- [15] V. Thomas, C. Wagner, M. Clausen, *OCR based post-processing of OMR*, ISMIR, 2011

- [16] D. Ringwalt, R. Dannenberg, *Image quality estimation for multi-score OMR*, ISMIR, 2015
- [17] N. Carter, *Segmentation and Preliminary Recognition of Madrigals Notated in White Mensural Notation*, Machine Vision and Applications, 1992
- [18] L. Tardon, S. Sammartino, I. Barbancho, V. Gomez, A. Oliver, *Optical Music Recognition for Scores Written in White Mensural Notation*, Image Video Process, 2009
- [19] J. Oncina, J. Calvo-Zaragoza *Recognition of Pen-Based Music Notation: the HOMUS dataset*, University of Alicante, 2014
- [20] <https://pl.wikipedia.org/>
- [21] <https://epodreczniki.pl/>
- [22] (a) <https://pl.wikipedia.org/>
 (b) <http://www.cvc.uab.es/cvcmuscima/>
 (c) J. Navotny, J. Pokorny, *Introduction to Optical Music Recognition: Overview and Practical Challenges*, Department of Software Engineering, Faculty of Mathematics and Physics Charles University, 2015
- [23] <https://andyzeng.github.io/ip/proj8/>
- [24] https://www.daviddarling.info/encyclopedia_of_music/R/rest.html