MPUM - Projekt
Rozpoznawanie gatunków muzycznych

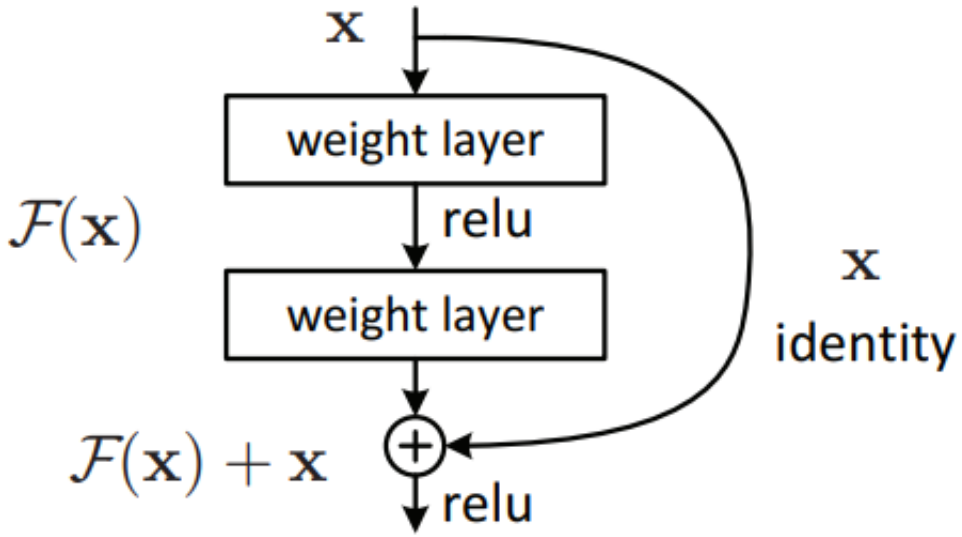
Julia Biały, Jakub Dziarkowski

15 czerwca 2022

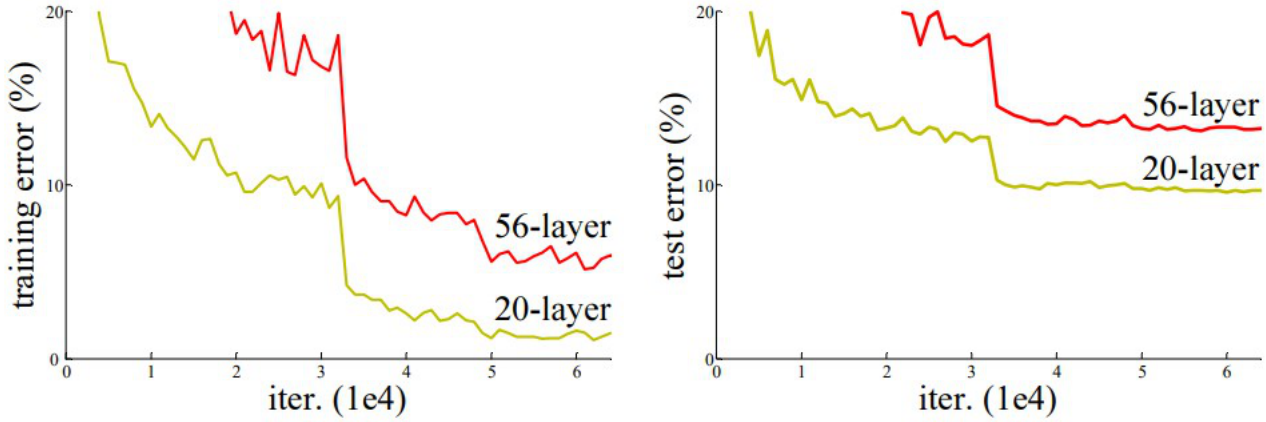
Wstęp
W naszym projekcie prezentujemy trzy różne podejścia do problemu rozpoznawania gatunków muzycznych. Pierwszym z nich jest przetworzenie plików dźwiękowych do obrazków i zastosowanie rezydualnych sieci neuronowych (ResNet), zaimplementowanych przez nas przy użyciu komponentów dostępnych w bibliotece Tensorflow np. warstwy konwolucyjnej. Kolejną metodą było zastosowanie algorytmu KNN (K najbliższych sąsiadów) napisanego przez nas od podstaw. W ostatniej kolejności przetestowaliśmy wybrane algorytmy dostępne w bibliotece sklearn: lasy losowe, drzewa decyzyjne, AdaBoost. Zarówno w punkcie drugim jak i trzecim do trenowania i predykcji dźwięków użyliśmy cech wyekstrahowanych z plików dźwiękowych, np. takich jak Mel-frequency cepstral coefficients.

Zbiór danych
Skorzystaliśmy z powszechnie znanego i wykorzystywanego zbioru "GTZAN Dataset - Music Genre Classification" - dostępnego na
https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification
Zawiera on po 100 plików dźwiękowych po 30 sekund dla każdego z 10 gatunków muzycznych (blues, classical, country, disco, hiphop, jazz, metal, pop, reggae, rock). Oprócz tego znajdziemy też reprezentację wizualną każdego dźwięku, oraz pliki features_3_sec.csv i features_30_sec.csv z cechami. Features_3_sec.csv powstał w oparciu o 3-sekundowe fragmenty dźwięków i to właśnie z tych danych korzystaliśmy przy KNN, lasach losowych, drzewach decyzyjnych i AdaBoost. We wszystkich przykładach 80% danych wybieraliśmy jako treningowe, a pozostałe 20% jako testowe.

ResNet
Głęboka sieć rezydualna jest siecią neuronową, która składa się z bloków rezydualnych. Takimi blokami nazywa się zbiór warstw, w którym wystepują tzw "identity shortcut connections" - takie połączenie między warstwami pozwala na ominięcie co drugiej (lub więcej) warstw. Wynik z warstw ominiętych jest wliczany do wyniku warstwy, do której przeskoczono. Widać to dobrze na poniższym rysunku z pracy "Deep Residual Learning for Image Recognition" autorstwa He et al.



ResNet rozwiązuje problem "znikającego gradientu". Żeby zmniejszyć współczynnik błędu w sieci neuronowej, typowo zwiększa się ilość warstw w architekturze. Niestety, wtedy właśnie błąd na zbiorach treningowym i testowym może się zwiększyć, wbrew oczekiwaniom. Dzieje się tak przez to, że propagację wsteczną (obliczanie i poprawianie wag z użyciem gradientów) zaczyna się od końcowej warstwy. Mnożenie gradientów sprawia, że bardzo maleją lub bardzo rosną. Przez to wczesne warstwy nie dość że są wolniejsze, to także osiągają też znacznie gorsze wyniki. Istotne jest to dlatego, że to początkowe warstwy są najważniejsze, bo odpowiadają za podstawowe cechy. Poniżej można zobaczyć wykres błędu na zbiorach testowym i treningowym dla konwolucyjnych sieci neuronowych, 56- i 20-warstwowych. Widoczne jest to, że 56-warstwowa sieć poradziła sobie znacznie gorzej od mniejszej sieci.



Wykorzystywaliśmy wariant bottleneck bloku rezydualnego. Polega on na użyciu w bloku rezydualnym dodatkowych warstw z mniejszą ilością ilością neuronów niż w warstwie pod lub nad nią. Zmniejsza to wymiarowość reprezentacji cech oraz czas trenowania, zwłaszcza bardzo głębokich sieci.

Początkowo planowaliśmy zaimplementować sieć rezydualną od podstaw, bez korzystania z biblioteki Tensorflow i podobnych. Niestety po wstępnych pracach i testach okazało się, że nasze niezoptymalizowane tak dobrze jak w bibliotekach sieci do nauczenia się potrzebują bardzo dużo czasu. To co teraz byliśmy w stanie testować w kilkanaście, kilkadziesiąt minut trwałoby co najmniej kilka, kilkanaście godzin, więc postanowiliśmy się oprzeć o model sieci konwolucyjnych z Tensorflow, a jedynie przejścia rezydualne dodać samemu.

Najpierw jako zbiór obrazków użyliśmy zbioru dostępnego w "GTZAN Dataset - Music Genre Classification". Natrafiliśmy na kilka problemów, z których największym było przetrenowanie - algorytm dość szybko osiągał nawet 99% poprawności na zbiorze treningowym, a jednocześnie miał problemy z przekroczeniem 60% na zbiorze testowym.

Rozwiązaniem na jakie się zdecydowaliśmy było podzielenie każdego dźwięku na 10 części po 3 sekundy, a następnie wygenerowanie Mel spektrogramu dla każdego z nich. Przy okazji usunęliśmy kategorie jazz ze zbioru danych, ponieważ mieliśmy dla niej błędy podczas generowania spektrogramu z którymi nie byliśmy w stanie sobie poradzić.

Parametry takie jak ilość i wielkość etapów, oraz liczba filtrów w każdym etapie dobieraliśmy głównie na podstawie tego, że nasze komputery nie radziły sobie z większymi wartościami, więc nie mieliśmy wiele do wyboru.

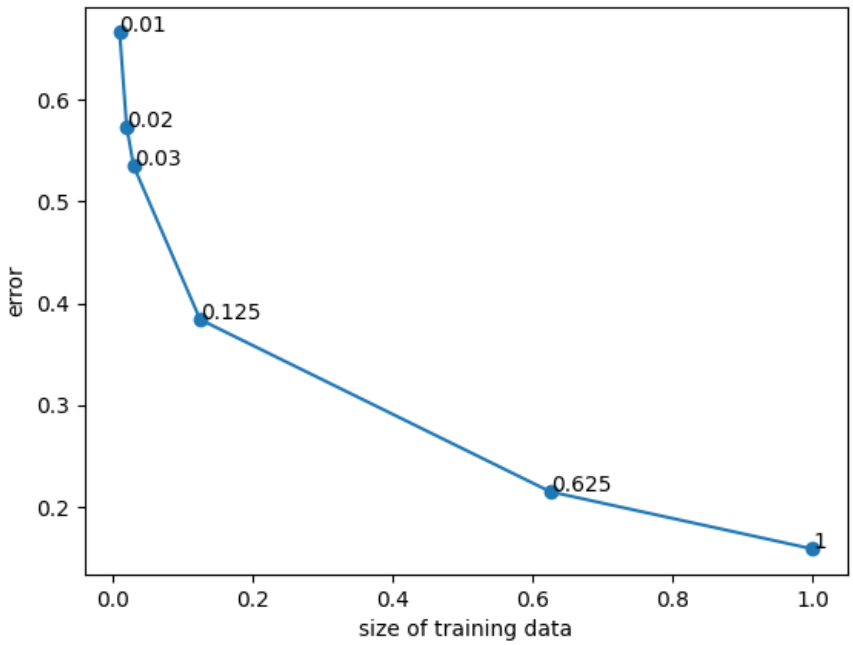
Nasz model rezydualnej sieci neuronowej można podzielić na 3 części - wstępna z 4 warstwami (BatchNormalization, Conv2D, BatchNormalization, Activation(ReLu)). Następna część składa się z etapów z blokami rezydualnymi. Bloki rezydualne w obrębie etapu działają na tych samych wymiarach danych i ilości neuronów w warstwach konwolucyjnych. Natomiast każde przejście do następnego bloku wiąże się ze zmniejszeniem wymiarów danych (szerokości, wysokości) i zwiększeniem ilości neuronów. Każdy blok rezydualny zwraca sumę danych wejściowych i danych przepuszczonych przez kilka serii warstw BatchNormalization, Activation, Conv2D. Część końcowa ma przede wszystkim na celu wypłaszczenie danych i predykcje, składa się z 6 warstw (BatchNormalization, Activation(relu), AveragePooling2D, Flatten, Dense, Activation(softmax)). W warstwach konwolucyjnych oraz Dense zastosowaliśmy regularyzację l2.

Model testowaliśmy tylko i wyłącznie na pełnym zbiorze treningowym, bez sprawdzania co się dzieje dla mniejszych frakcji, a uśredniony błąd klasyfikacji wyniósł 0.1756.

Skrótowy opis warstw: BatchNormalization - skalowanie wartości w danych, Conv2D - warstwa konwolucyjna, szczególnie dobrze sprawdza się w wykrywaniu wzorców w obrazkach np. poprzez tzw. kernel, który sprawia, że w obliczeniu wyjściowej wartości biorą udział też sąsiedzi (np. sąsiednie piksel obrazka), Activation - warstwa z funkcją aktywacji, u nas relu lub softmax, AveragePooling2D - u nas służy do zmniejszania wymiarów, poprzez wzięcie uśrednionej wartości ze zbioru sąsiadujących danych, Flatten - wypłaszczenie, można to sobie wyobrazić jako zamienianie macierzy o wymiarach nxm na wektor o długości nm, Dense - warstwa w pełni połączona (taka jak w zwykłych sieciach neuronowych).

KNN

Podczas predykcji wyniku dla danej ze zbioru testowego obliczamy odległość (odległość w rozumieniu przestrzeni euklidesowej) od wszystkich danych ze zbioru treningowego. Wyznaczaliśmy 16 najbliższych sąsiadów. Wybór 16 sąsiadów dobraliśmy ręcznie. Wyniki uśredniliśmy dla 3 przebiegów algorytmu. Błędy klasyfikacyjne wynoszą odpowiednio:
0.6658, 0.5734, 0.5345, 0.3838, 0.2148, 0.1586.
Poniższy wykres przedstawia krzywą błędu na partycjach zbioru treningowego (0.01, 0.02, 0.03, 0.125, 0.625, 1).

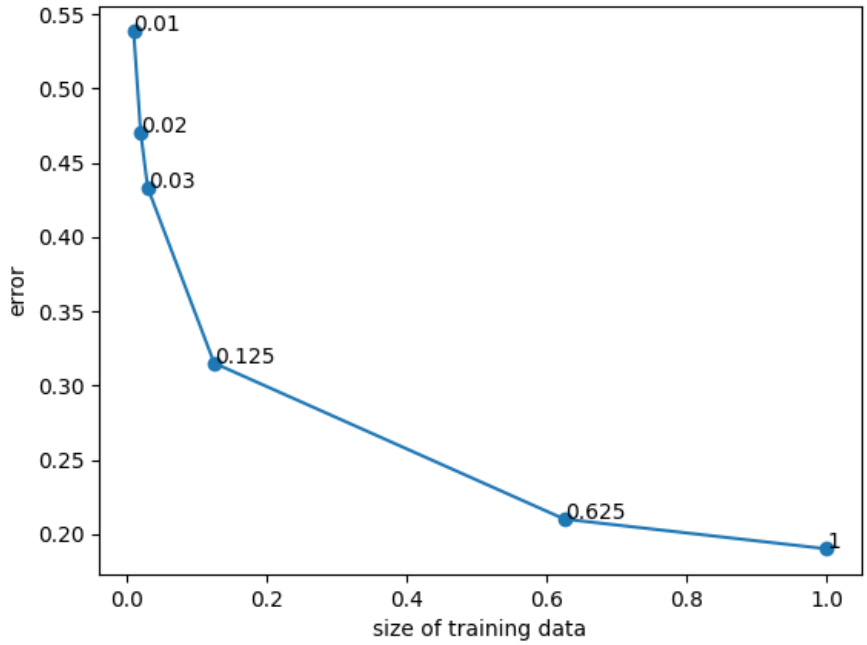


Lasy losowe, Drzewa decyzyjne, AdaBoost

Te algorytmy, podobnie jak knn, były już omawiane na wykładzie, a dodatkowo tutaj skorzystaliśmy z gotowych implementacji z biblioteki, także poniżej wstawiamy tylko rezultaty bez opisu teoretycznego.

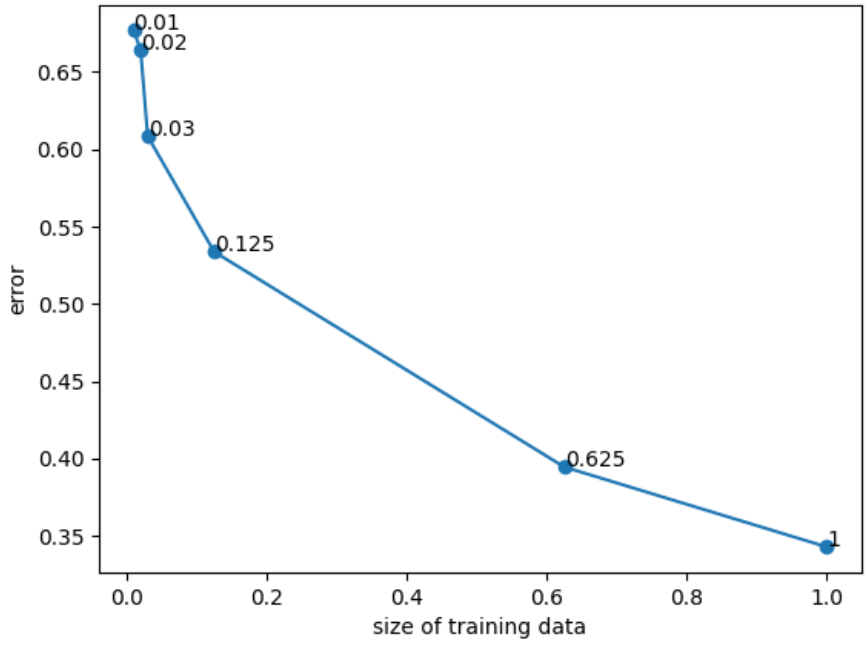
Lasy losowe

Błędy klasyfikacyjne wynoszą odpowiednio:
0.5382, 0.4704, 0.4332, 0.3151, 0.2102, 0.1901.
Poniższy wykres przedstawia krzywą błędu na partycjach zbioru treningowego (0.01, 0.02, 0.03, 0.125, 0.625, 1).



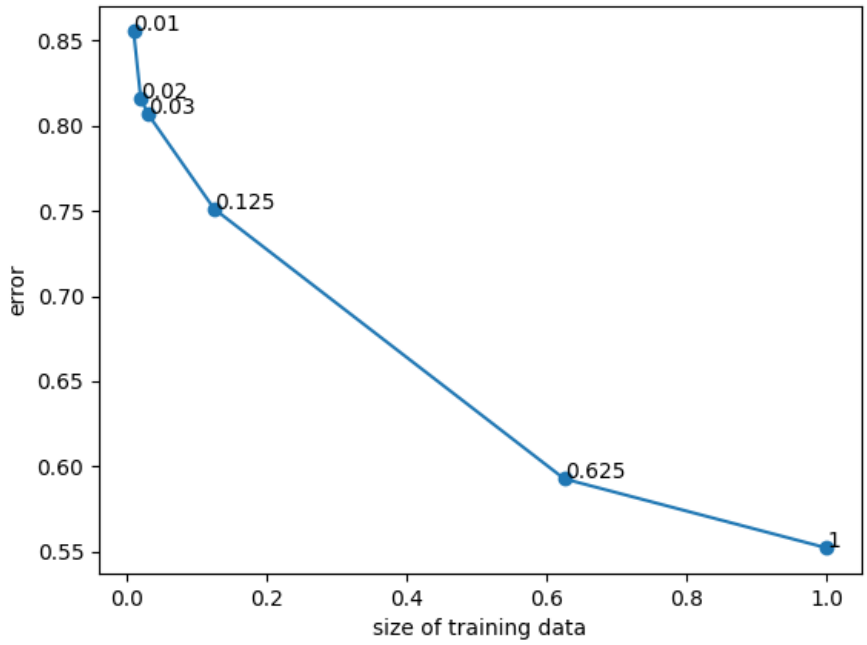
Drzewa decyzyjne

Błędy klasyfikacyjne wynoszą odpowiednio:
0.6768, 0.6641, 0.6084, 0.5337, 0.3945, 0.3428.
Poniższy wykres przedstawia krzywą błędu na partycjach zbioru treningowego (0.01, 0.02, 0.03, 0.125, 0.625, 1).



AdaBoost

Błędy klasyfikacyjne wynoszą odpowiednio:
0.8551, 0.8163, 0.8069, 0.7510, 0.5927, 0.5522.
Poniższy wykres przedstawia krzywą błędu na partycjach zbioru treningowego (0.01, 0.02, 0.03, 0.125, 0.625, 1).



Wnioski

Patrząc tylko na wyniki dochodzimy do wniosku, że najlepiej z tym problemem poradził algorytm KNN, co jest o tyle ciekawe, że to prawdopodobnie najłatwiejszy koncepcyjnie oraz implementacyjnie algorytm spośród porównywanych w naszym projekcie. Trzeba jednak wziąć pod uwagę po pierwsze to, że dane wejściowe do ResNetu różnią się od danych wejściowych do innych algorytmów i być może z inną reprezentacją wizualną niż ta wygenerowana przez nas by sobie poradził lepiej. Warto odnotować również, że mierzyliśmy się z ograniczeniami sprzętowymi, a ResNet jest bardzo rozbudowaną siecią wymagającą dużej mocy obliczeniowej, stąd podejrzewamy, że przy większych możliwościach dostosowania parametrów i ich wielkości udałoby nam się uzyskać znacznie lepsze rezultaty. Z pozostałych trzech algorytmów, to lasy losowe poradziły sobie jeszcze całkiem dobrze tzn. na małych frakcjach zbioru treningowego wyniki były nawet lepsze od KNN, a na pełnym zbiorze niewiele gorsze. Drzewa decyzyjne na całym zbiorze treningowym miały aż 34% błędnych predykcji, a AdaBoost nawet 55%, więc można stwierdzić, że nie nadają się one do rozwiązania tego problemu.