卒 業 論 文

畳み込みニューラルネットワークを用いた楽器音識別に関する研究

北海道科学大学 工学部
情報工学科
2-16-3-049
2-16-3-063
指導教員　松﨑 博季
2020 年 (令和 2 年)2 月

目 次

第 1 章　緒言 **1**

第 2 章　機械学習について **2**

2.1　まえがき 2

2.2　機械学習及び，Deep Learning について 2

2.3　畳み込みニューラルネットワークについて 3

第 3 章　学習用音声データの作成並びに楽器音識別システムの概要について **4**

3.1　まえがき 4

3.2　使用したソフトウェア及び開発環境 4

3.3　音声データの作成 5

3.3.1　複数の楽器音の音声データの作成 5

3.3.2　水増し処理 5

3.3.3　ファイルの切り分け 5

3.4　音声ファイルから画像ファイルへの変換 6

3.4.1　オーバーラップ処理と窓関数処理 6

3.4.2　高速フーリエ変換とサウンドスペクトログラム 6

3.4.3　画像ファイルの作成 6

3.5　ラベル付け 9

3.6　ネットワークの構造について 9

3.7　楽器音識別システムについて 9

第 4 章　実験について **11**

4.1　はじめに 11

4.2　実験概要 11

4.3　結果 11

第 5 章　結論 **14**

参考文献 **15**

付 録 A GarageBand を用いた wav ファイルの作成方法 **16**

付 録 B Audacity の使い方 **23**

B.1 ホワイトノイズ 23

B.2 2 音上げ，2 音下げ 24

B.3 テンポアップ 25

付 録 C 本研究に用いたプログラムのソースコード **27**

C.1 ファイルの切り分け 27

C.2 ファイルのリネーム 29

C.3 サウンドスペクトログラム変換 30

C.4 ラベル付け 34

C.5 CNN を用いた学習 36

C.6 楽器音識別システム 39

謝 辞 **41**

第1章　緒言

楽器演奏者は，複数の楽器が同時に演奏されている楽曲から特定の楽器の音だけを聞き取りそれを模倣することがある．この時，特定の楽器音だけをはっきりと聞き取ることができれば問題ないが，他の音に埋もれた状態ではその聞き取りは困難になる．このような場合に楽曲の中から自分が演奏する楽器の音声のみを再生する，といったことは現在不可能である．本研究では，様々な楽器の音が同時に再生される音声ファイルの中から，特徴を見出し，一つの楽器の音を抽出することが出来るのかを，畳み込みニューラルネットワーク (以下 CNN) の技術を用いて試みた結果について報告する．

第2章　機械学習について

2.1　まえがき

本章では本研究に用いた技術である機械学習，及び CNN について基本的な部分を説明する．

2.2　機械学習及び，Deep Learning について

機械学習とは，機械にデータからアルゴリズムを学習させ，予測を行わせることを可能にすることが目的とされる人工知能 (AI) の一種である．機械学習は，教師あり学習，教師なし学習，強化学習の 3 つに大別できる．教師あり学習とは，入力データに対して望ましい出力データを与え，未知のデータを予測することが出来るように学習させるもので，教師なし学習は入力データに対して正解を与えずに，入力データに意味のある構造を見出させると言うものである．そして強化学習とは，機械にエージェントとして行動させることで，目的となる最適解を導き出させると言うものである．Deep Learning とは，生物の神経細胞を模倣した機械学習であるニューラルネットワークを用いて，より多くの層を構造として持つものである．ニューラルネットワークに用いられる層にはいくつか種類がある．学習に用いるデータを読み込む入力層，脳にある神経細胞のニューロンのように前層から送られてきた信号をどのように活性化させて次のノード (値に対して計算が行われる場所) に渡すかを決める活性化関数の層，学習データに対する学習が過剰に行われることにより，学習データ以外の未知のデータに対する正解率が下がってしまう過学習を防ぐために，指定した割合で最適化された信号を切除するドロップアウト層，前層にある全てのノードと次の層にある全てのノードを繋ぐ全結合層，データの次元数を一次元に揃える平坦化層などがある．ニューラルネットワークは，これらの層を活用して自ら特徴量を得て学習を進めることができ，現在では様々な分野で，もはやなくてはならない存在となっている [1,2].

2.3　畳み込みニューラルネットワークについて

本研究では，楽器の音の特徴を学習するために畳み込みニューラルネットワークという技術を用いる，CNN とは，人間の視覚野をヒントにして作られた畳み込み層とプーリング層を有するネットワークである．畳み込み層は画像の一部分に対し注目してフィルターをかけることで特徴を抽出する層であり，プーリング層は注目点がずれていかないように位置の修正を行う層である．人間は，網膜で画像を受け取り，色彩や光量の抽出を行い，大脳皮質に渡す．そして大脳皮質で特徴を抽出し，位置ずれの許容をする．CNN はまさにこの仕組みをニューラルネットワークで実現したものである．その為，ニューラルネットワークにおける画像認識の分野で広く活躍している [1–3].

第3章　学習用音声データの作成並びに楽器音識別システムの概要について

3.1　まえがき

本実験では，音声データをサウンドスペクトログラムというグラフに変換して画像データとして扱うことでこれを学習データとし，CNN を用いて学習を行い，その結果を元に音声データに使用されている楽器の識別を行うことを目的とする．

本章では，実験で扱う音声データの作成方法，及び楽器音識別システムの概要について述べる．

3.2　使用したソフトウェア及び開発環境

本研究に用いたソフトウェア及び開発環境を以下に示す．

Python3.7.4 開発言語

PyCharme 統合開発環境

Google ドライブ オンラインストレージサービス

Google Colaboratory クラウド開発環境

GarageBand 音楽制作ソフトウェア

Audacity サウンド編集ソフト

CNN を用いて学習を行う際，大量の画像データを読み込む為，ローカルの CPU を用いると，途轍もない時間を浪費することになる．その為，データの生成には PyCharme を用い，実験を行う際にはクラウド上で GPU を使用してプログラムを実行することができる Google Coraboratory を用いた．

3.3　音声データの作成

本システムには CNN を使用する為，判別したい楽器音が含まれる大量の学習用音声データが必要になる．そこで，楽器音が長時間再生される音声ファイルを作成し，その音声ファイルを元に水増し処理を施した別の音声ファイルを作成した後に，音声ファイルを切り分けることで，大量の音声データを短時間で得られるようにした．以下に実験に用いるデータの元となる音声データの作成方法を，順に述べていく．

3.3.1　複数の楽器音の音声データの作成

Apple 社製の GarageBand を用いて，Piano，Guitar，Bass，Trumpet，Flute，Marimba および Strings の楽器音の音声ファイルを作成した．いずれも同じフレーズを約 50 分再生する音声ファイルである．音声ファイルのフォーマットは量子化ビット数 16，標本化周波数 44.1kHz，ステレオ形式の WAV 形式である．作成方法及び使用したフレーズ等については付録 A を参照されたい．

3.3.2　水増し処理

まずは GarageBand で作成された音声データに対して 4 種類の処理を Audacity を用いて行なった．Audacity の操作手順については付録 B を参照されたい．1 種類目はホワイトノイズ印加である．ホワイトノイズとは，全ての周波数で同じ強度のノイズである．本研究では元となる音声ファイルに振幅 0.3 のホワイトノイズを印加した音声ファイルを作成した．2 種類目は 2 音上げである．これは元となる音声ファイル内の音声全体に対し，テンポを変えずピッチを 2 音分上げる処理である．3 種類目は 2 音下げである．これは 2 種類目と同様に元となる音声ファイル全体の音声を 2 音分下げる処理である．4 種類目はピッチを変更せずにテンポを変更する処理である．本研究では，テンポの変更率を 100 に設定することで元となる音声ファイル全体の音声を倍速にする処理を施した．これら 4 種類の処理を全てのファイルに施したもの作成した．

3.3.3　ファイルの切り分け

上記で作成した音声データを読み込み，指定した秒数ごとにデータを切り分け，新規に作成したディレクトリに保存した．実行には Python プログラムを用いた．全ての音声データをそれぞれ 2 秒ずつに切り分け，結果約 45,000 個のファイルを作成した．実行したプログラムのソースコードについては，付録 C の C.1 を参照されたい．

3.4　音声ファイルから画像ファイルへの変換

本研究では，CNN を用いてデータを学習して楽器音の識別を行うため，扱うデータを画像ファイルの形式にしなければならない．従って以下に音声ファイルを画像ファイルへと変換する方法を述べていく．

3.4.1　オーバーラップ処理と窓関数処理

オーバーラップ処理とは，データを任意のフレームに分割する処理である．スペクトログラム に変換するために音声データにハニング窓関数をかける．ハニング窓関数とはフレームに分割した波形の両端がなだらかに小さくなっていき，両端が必ず 0 になる関数である．元の波形がどんなに不連続であっても必ず両端の値が 0 になる為，周期性を保つことができる．もし周期性が保たれていないと，この後に施す高速フーリエ変換処理において本来求めたい周波数の他に余分なスペクトルが現れてしまい分解能が悪くなってしまう．しかしこの窓関数をかける事で両端に近い値が減衰してしまい情報が欠損してしまう．そこでオーバーラップ処理が必要になる．オーバーラップしてデータを抽出した波形に対して窓関数をかけると，窓関数で減衰しない部分がずれていくため情報の損失を軽減する事ができる．つまり，精度の良いフーリエ変換をするために窓関数を使用したいが，情報を失いたくない場合にオーバーラップ処理は用いられる．

3.4.2　高速フーリエ変換とサウンドスペクトログラム

高速フーリエ変換 (Fast Fourier Transform)(以下 FFT) とは，あらゆる周期関数は正弦波の足し合わせで表現できるという特性を利用し，周波数成分に分解する事である．FFT を音声データに施し，縦軸は周波数，横軸は時間を表し，色の濃淡でその時点での周波数の強さを表すグラフにしたものがサウンドスペクトログラム である．

3.4.3　画像ファイルの作成

Python で音声ファイルを画像ファイルへと変換するプログラムを作成し，実行した．ソースコードについては付録 C.3 を参照されたい．以下に音声ファイルに施した処理について述べる．

はじめに，GarageBand で作成された音声データがステレオだった為，モノラルに変換した．2 秒ずつに切り分けられた音声ファイルに対してオーバーラップ率 90%，窓幅 (フレームサイズ) を 4096 としてオーバーラップ処理 (3.4.1 参照)，ハニング窓処理 (3.4.1 参照) お

よび FFT(3.4.2 参照) を行った．

オーバーラップ率とは，フレーム同士の重なり具合を示している．例えばオーバーラップ率が 50%の場合は，丁度フレームサイズの半分の位置でフレーム同士が重なることになる．さらに，本実験で用いる音のような広い周波数帯域のデータを CNN で学習し易くするためには，また，デシベル変換と聴覚補正を施す必要がある．人間の可聴音圧域は 20μPa から 200Pa とされている．このままでは音の大きさの範囲が非常に広くなってしまう．デシベル変換を用いる事で 0dB から 140dB で扱う事ができるためスペクトログラム で表示したときに分析を行い易くすることができる．デシベル変換で述べた可聴音圧域と同様に，人間には可聴周波数が存在する．一般に人間の可聴周波数は 20Hz から 20kHz とされている．そのため，仮に音声データに可聴周波数の範囲を越えた音声が含まれていたとしても，人間の耳では聞き取ることができない．そこで，人間の耳では聞き取りにくい低周波数帯や高周波数帯の周波数成分を低下させ，人間の耳の感覚に合う測定値を得られるように補正することが，聴覚補正である．FFT を施す前にデシベル変換と聴覚補正を施す事で，人間が音を聞き分けるのと同様に，システムも楽器音を識別できるような学習データの作成を目指した．

最後に高速フーリエ変換で得られた情報から， 全ての音声データに対するサウンドスペクトログラムを求め，画像データ (ビット深度 32 ビット，幅 640 ピクセル，高さ 480 ピクセル) として保存した．サウンドスペクトログラムの周波数範囲を 0Hz から 10000Hz とした．例として bass の音声ファイルを元に作成したサウンドスペクトログラムを，図 3.1〜3.5 に示す．これらのサウンドスペクトルグラム画像を用いて次のラベル付け処理を行う．

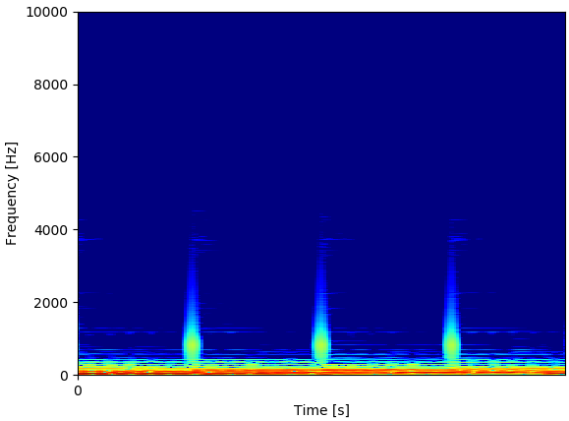

図 3.1: 2 音下げ処理をした bass のサウンドスペクトログラム

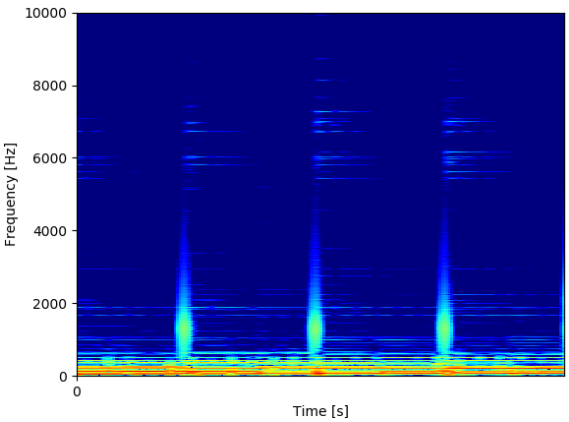

図 3.2: 2 音上げ処理をした bass のサウンドスペクトログラム

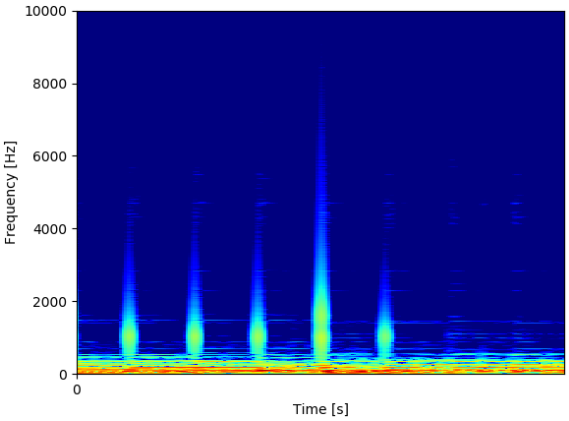

図 3.3: テンポアップ処理をした bass のサウンドスペクトログラム

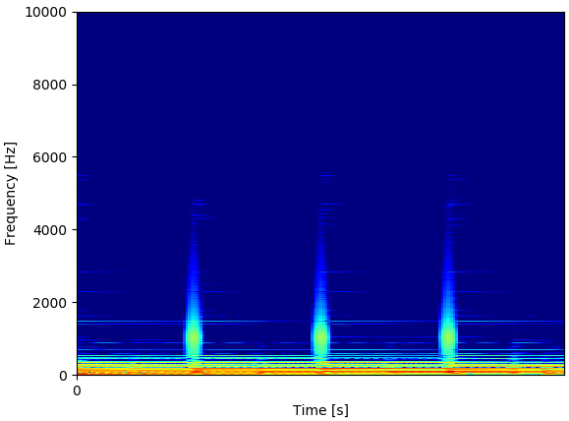

図 3.4: 水増し処理をしていない bass のサウンドスペクトログラム

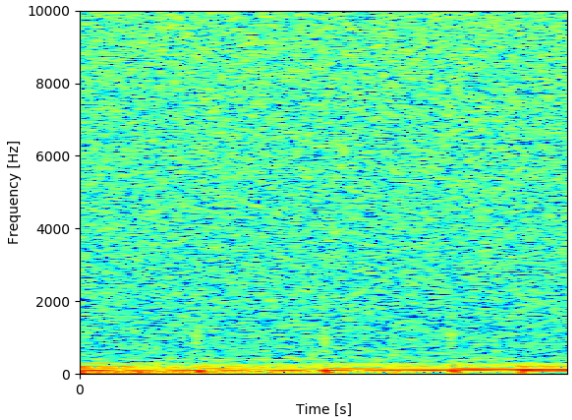

図 3.5: ホワイトノイズ印加をした bass のサウンドスペクトログラム

3.5　ラベル付け

作成した画像データに対してラベルを付け，機械学習に用いることができるようにした．画像データとラベルを対にした画像ファイルを本実験に用いるデータセットとする．作成したプログラムのソースコードについては，付録 C の C.4 を参照されたい．

3.6　ネットワークの構造について

本システムの CNN を用いたネットワークモデルは図 3.6 に示すように，第 2 翔で述べた Conv(畳み込み層) 及び MaxPooling（プーリング層）の他，relu(活性化関数)，Dropout(過学習を防ぐための不活性化選択層)，Flatten(平坦化層)，Dense(全結合層)，softmax(活性化関数) の 7 種類のレイヤから成る全 19 層で構成される [2]．前節で述べた学習用のデータセットで学習後，本システムで任意の複数の楽器音が含まれるサウンドスペクトログラムから楽器音を識別できるかどうか確認を行う．作成したプログラムのソースコードについては，付録 C の C.5 を参照されたい．

3.7　楽器音識別システムについて

先に述べた CNN を用いてデータの学習を行い，得られた重みを用いて与えられたサウンドスペクトログラムが，どの楽器のデータかを識別するシステムを作成した．作成したプログラムのソースコードについては，付録 C の C.6 を参照されたい．

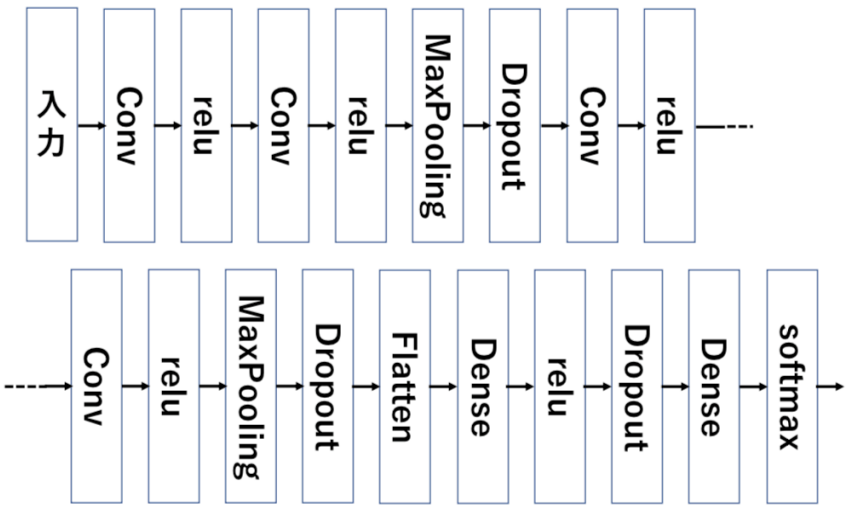

図 3.6: 学習に用いる CNN のネットワークモデル

第4章　実験について

4.1　はじめに

本章では，音声データを元に作成したサウンドスペクトログラムと楽器音識別システムを用いた実験について述べていく．

4.2　実験概要

前章で作成したデータセットから，各楽器音に対して 1000 枚ずつ画像を集め，データ総数 7000 枚で実験を行った (データ総数の違いで結果に差異はなかった為)．学習データとテストデータを 7 対 3 の割合に分けて学習を行った．

4.3　結果

図 4.1 に実験を行なった際の機械学習における精度の推移を，図 4.2 に実験を行なった際の機械学習における損失の推移を，図 4.3 に Piano の音声から作成したサウンドスペクトログラムを，図 4.4 に Strings の音声から作成したサウンドスペクトログラムを，図 4.5 に楽器音識別システムに Piano と Strings が同時に鳴らされている音声データの識別結果を示す．最終的な精度は約 99.9%，損失は約 0.6%となった． 学習を開始してからの精度と損失の推移に着目した際，これは元の音声データが同じフレーズの繰り返しであったことに起因すると考えられる．この事から，同じ種類の楽器でも音色の違う音声データや，様々なフレーズを使用した音声データを実験に用いる事で，より多くの楽器を分類できるようになると考える．また，2 種類の楽器が同時に鳴っている音声データの識別結果から，この楽器音識別システムは，より強く特徴がサウンドスペクトログラムに表れている楽器のみを認識していると考えられる．複数種類の楽器が同時に鳴っている音声データも学習させ，実験するべきだと考えられる．

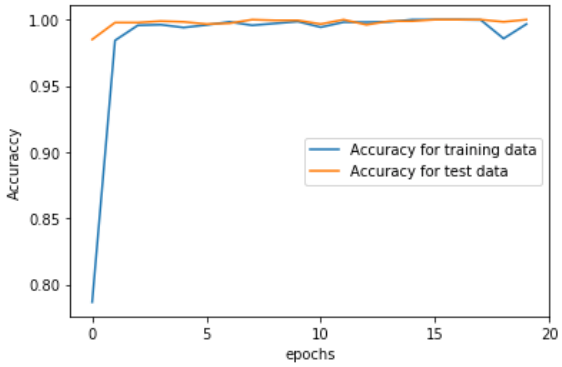

図 4.1: 学習における精度の推移

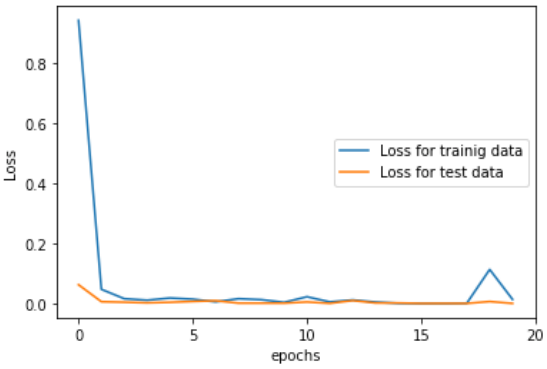

図 4.2: 学習における損失の推移

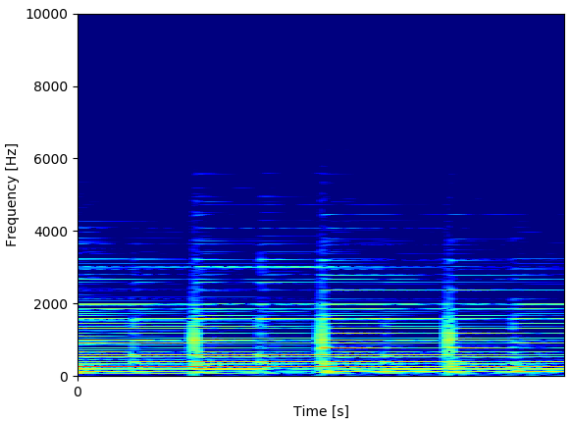

図 4.3: Piano のサウンドスペクトログラム

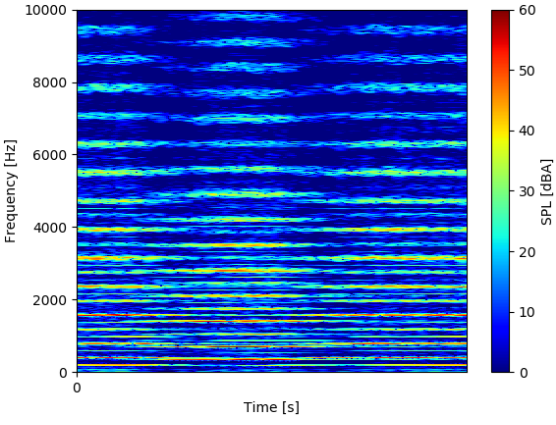

図 4.4: Strings のサウンドスペクトログラム

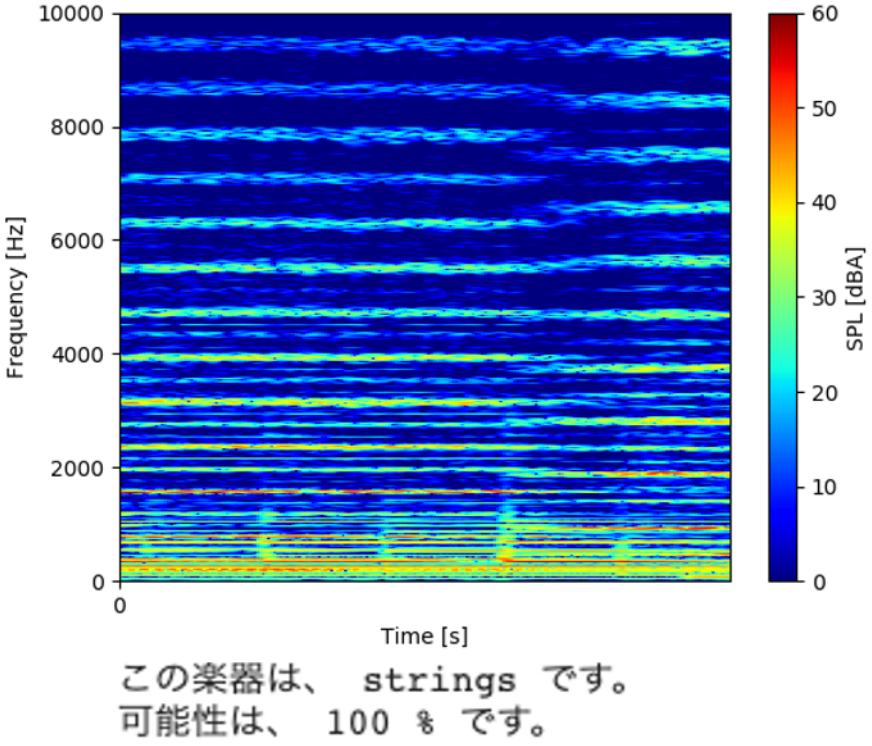

図 4.5: Piano と Strings の合成音からなる
サウンドスペクトログラムを識別した結果

第5章　結論

本研究では，CNN の技術を用いて楽器音から楽器を識別する事は可能であるという事を明らかに出来た．しかし，実験に用いたデータの種類が乏しく，2 種類類の楽器音が同時に再生される実験データに対して両方の楽器が使われていることが想定できる (今回の場合，使われている楽器 2 種類の内どちらか一方を指して 50%の可能性を示す) ようなシステムを作成する事が出来なかった．今後は (同じギターでもアコースティックギターやエレキギターなどがあるように) より多い種類の楽器音による音声データを用意して，楽器音識別システムを用いた実験を重ね，ネットワークモデルのチューニングをしていく事で，まずは複数の楽器が演奏に用いられていることを認識させることができるようにしたい．最終的には，2 種類よりも多くの楽器音が同時に再生される音声ファイルでも，それぞれの楽器音を識別できるようなシステムを目指していきたい．

参考文献

[1] Antonio Gulli, Sujit Pal:”直感 Deep Learning- Python × Keras でアイデアを形にするレシピ”, (大 串正矢, 久保隆宏, 中山光樹訳), オライリー・ジャパ ン (2018)

[2] 斎藤 康毅，”ゼロから作る Deep Learning-Python で学ぶディープラーニングの理論と実装”，オライリー・ジャパン (2016)

[3] AGIRobots，”CNN の勉強（I）-網膜と一次視覚野の構造からネオコグニトロンへ”，https://agirobots.com/blog/ml/retina-v1-neocognitron/

[4] WATLAB，”Python で FFT をする前にオーバーラップ処理をしよう!”，https://watlab-blog.com/2019/04/17/python-overlap/

[5] 川又 政征, ”MATLAB で学ぶディジタル信号処理の基礎ー第 2 回 離散フーリエ変換と高速フーリエ変換ー”, http://www.mk.ecei.tohoku.ac.jp/jspmatlab/pdf/matdsp2.pdf

[6] HELLO CYBARNETICS，”FFT とは何か，フーリエ変換との関連と絶対抑えるべき注意点”，https://www.hellocybernetics.tech/entry/2017/03/22/043855

付 録A　GarageBandを用いたwavファイルの作成方法

以下にApple社製のソフトウェア，GarageBandを用いて楽器音の音声ファイルをwavファイル形式で作成する手順を示す．

GarageBandのメニューからファイルを選択し，新規プロジェクトを押下すると，図A.1に示す画面が表示される．ウインドウ右下にある選択ボタンを押下する．

図 A.1: プロジェクト選択画面

すると図A.2に示す画面が表示される．ソフトウェア音源を選択し，ウインドウ右下にある作成ボタンを押下する．

図 A.2: トラックタイプ選択画面

すると図 A.3 に示す画面が表示される．ウインドウ右上に 3 つ並ぶアイコンの内，真ん中にある，赤枠で囲んだループバックアイコンを押下する．

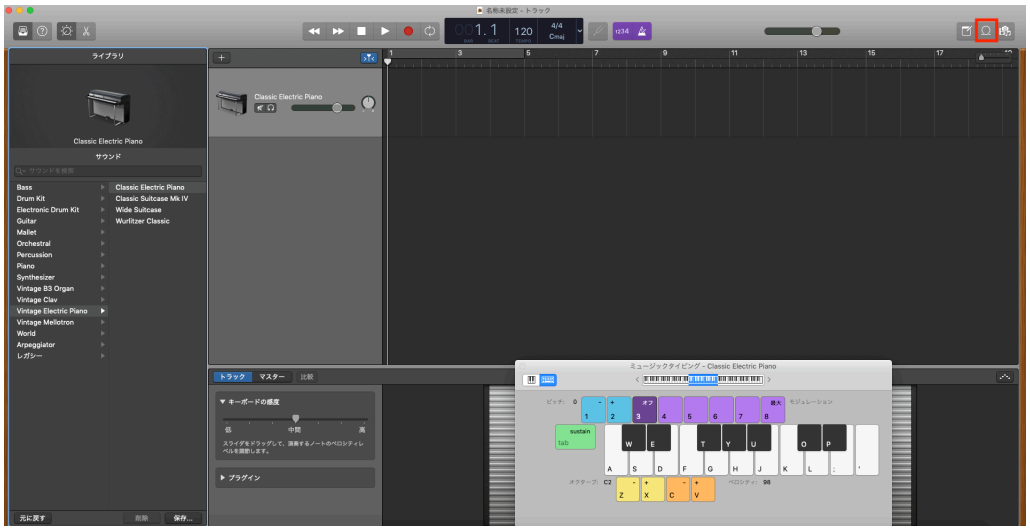

図 A.3: 音楽作成画面

すると図 A.4 に示す画面が表示される．赤枠で囲んだインストゥルメントを押下する．

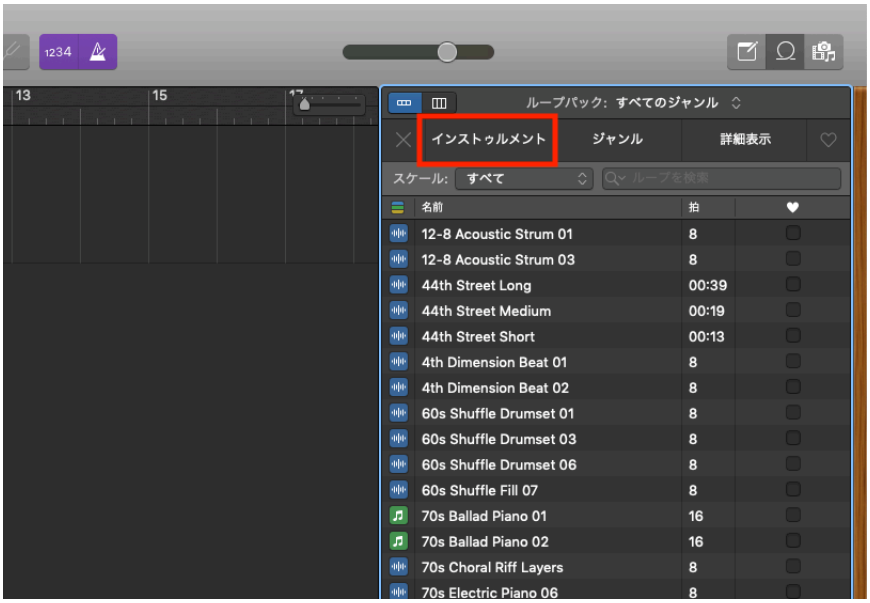

図 A.4: ループバック選択画面 1

すると図 A.5 に示すように楽器の種類でループを検索できる画面が表示される．赤枠で囲んだアイコンを押下する．

図 A.5: ループバック選択画面 2

すると図 A.6 に示すようにループタイプで検索結果を絞り込むことができる画面が表示される．本実験では MIDI ループのみを選択する．

図 A.6: ループバック選択画面 3

検索結果画面から MIDI ループを選択し，一度押下すると選択した MIDI ループを試聴することができる．本実験では 70s Ballad Piano 01 を使用した．図 A.7 に示すようにドラッグ＆ドロップする．

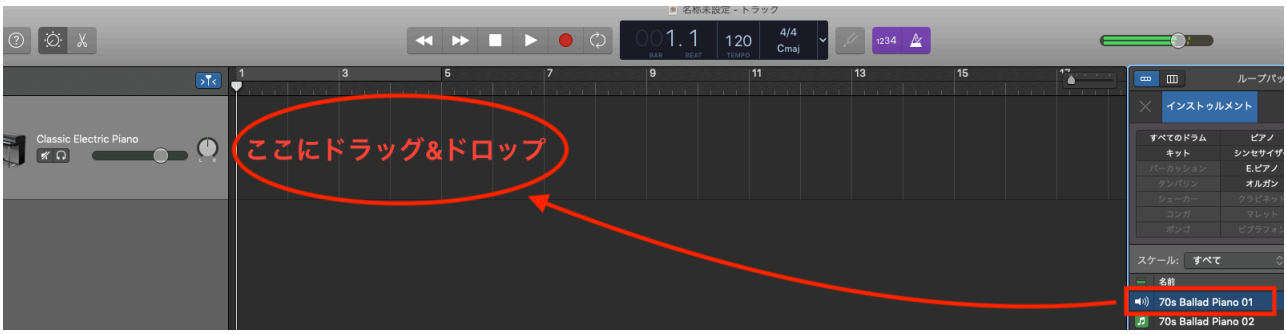

図 A.7: 使用する MIDI ループを
リージョンへドラッグ＆ドロップ

すると図 A.8 に示すようにトラックのリージョンに MIDI ループのスコアが表示される．本実験ではこのスコア部分を選択し，コピー&ペーストを繰り返すことで，図 B.9 に示すように長い間音声を再生することができるオーディオトラックを作成した．

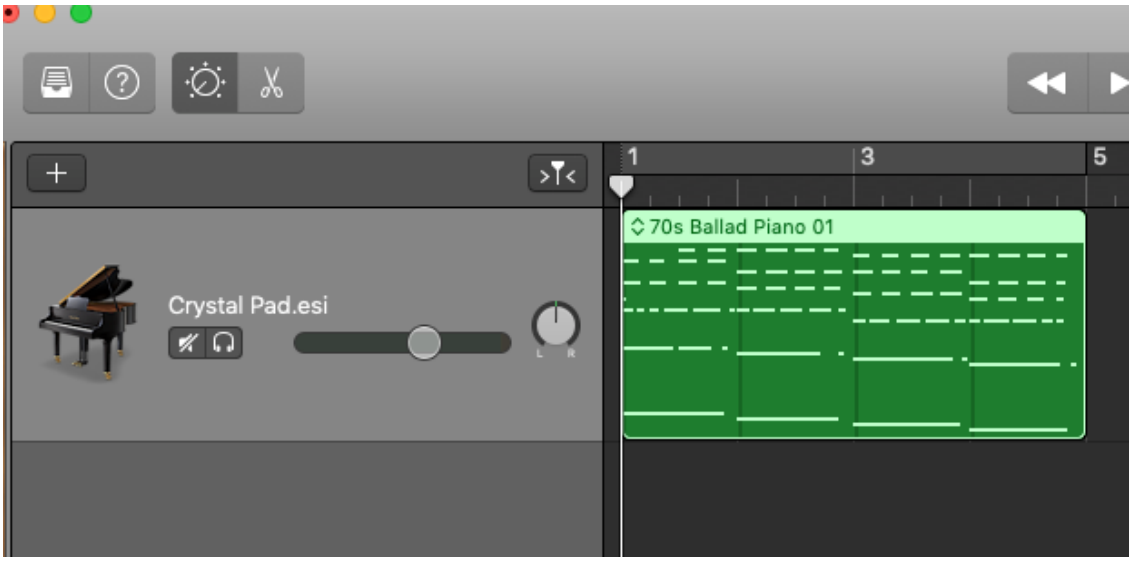

図 A.8: スコアが表示されたトラック

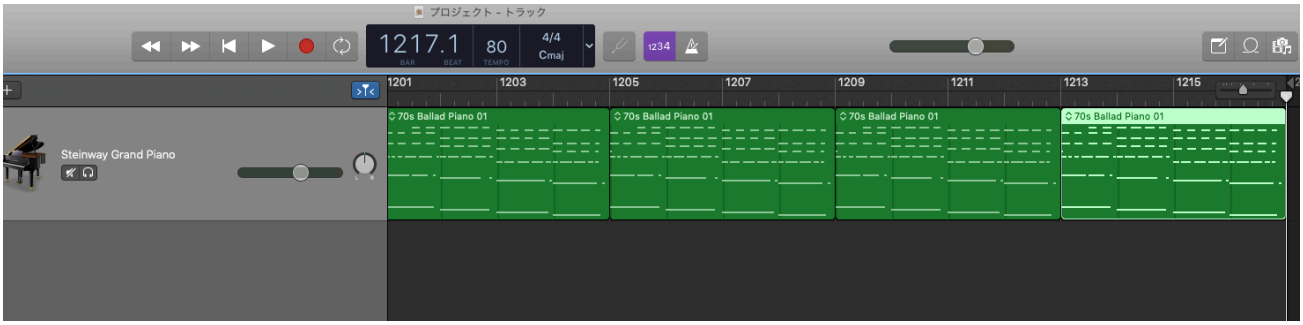

図 A.9: 出来上がったトラック

また，ウインドウ左上にあるライブラリアイコン (図 A.10 赤枠参照) を押下し，別の楽器を選択することで，再生される楽器音を変更することができる (図 B.11 参照)．本実験では，Piano，Guitar，Bass，Trumpet，Flute，Marimba および Strings の楽器音を用いた．

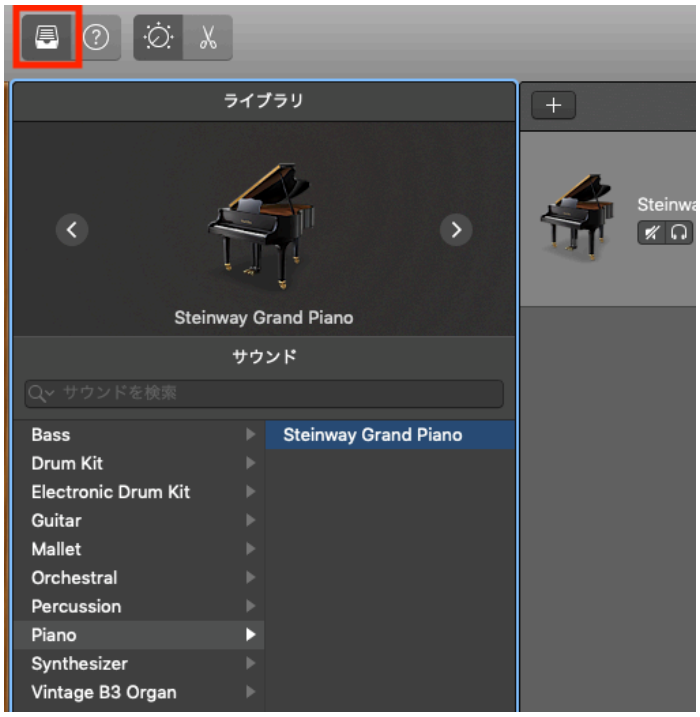

図 A.10: ライブラリアイコンと
その下に表示される楽器一覧

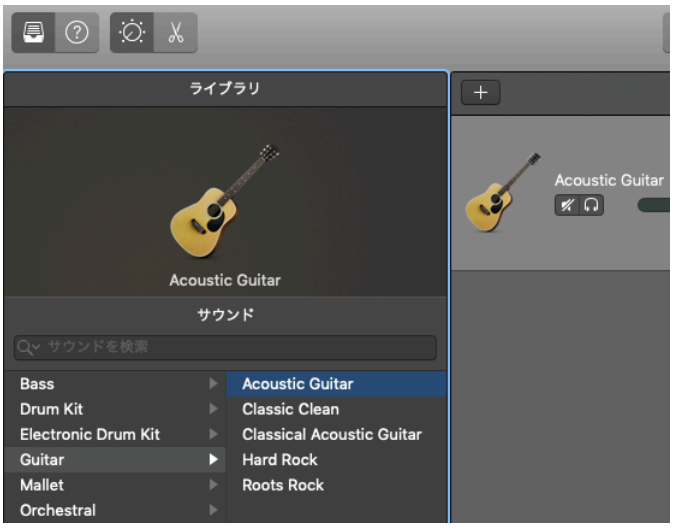

図 A.11: 楽器変更後

作成したトラックを wav ファイル形式で保存する方法を以下に示す．

GarageBand のメニューから共有を選択し，曲をディスクに書き出すを押下すると，図 A.12 に示す画面が表示される．

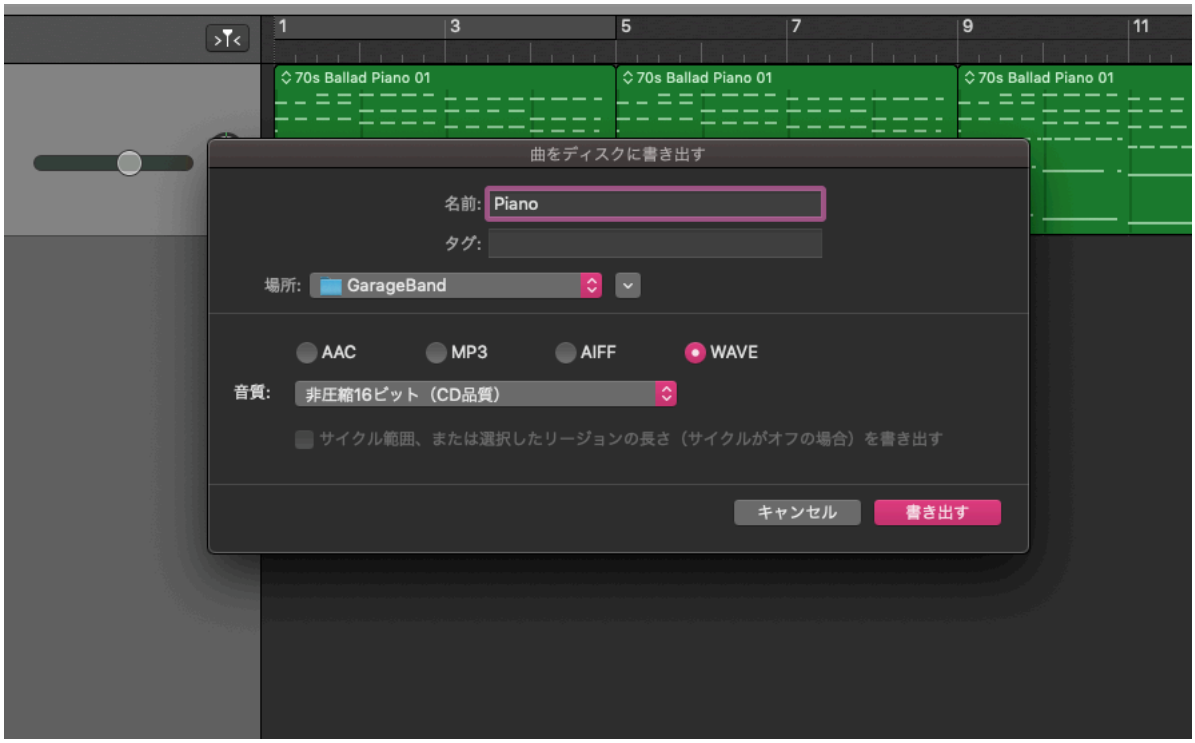

図 A.12: 作成した楽器音再生トラックを wav ファイル形式で書き出す

任意の名前を入力して任意の保存場所を選択し，WAVE を選択した後にウインドウ右下にある書き出しボタンを押下することで，作成したトラックを wav ファイル形式で保存することができる．

以上の手順を楽器を変えて繰り返すことで本研究で使用した楽器音の wav ファイルを作成した．

付 録B　Audacityの使い方

本研究では楽器音に対して 4 種類の水増し処理を施した．水増し処理を施す為の Audacity の操作手順を以下に示す．

B.1　ホワイトノイズ

Audacity のメニューからファイルを選択し，ホワイトノイズを加えたいファイルを選択する．図 B.1 のように表示される．

メニューのトラックから新しく追加を選択し，ステレオトラックを選択すると図 B.1 のように表示される．

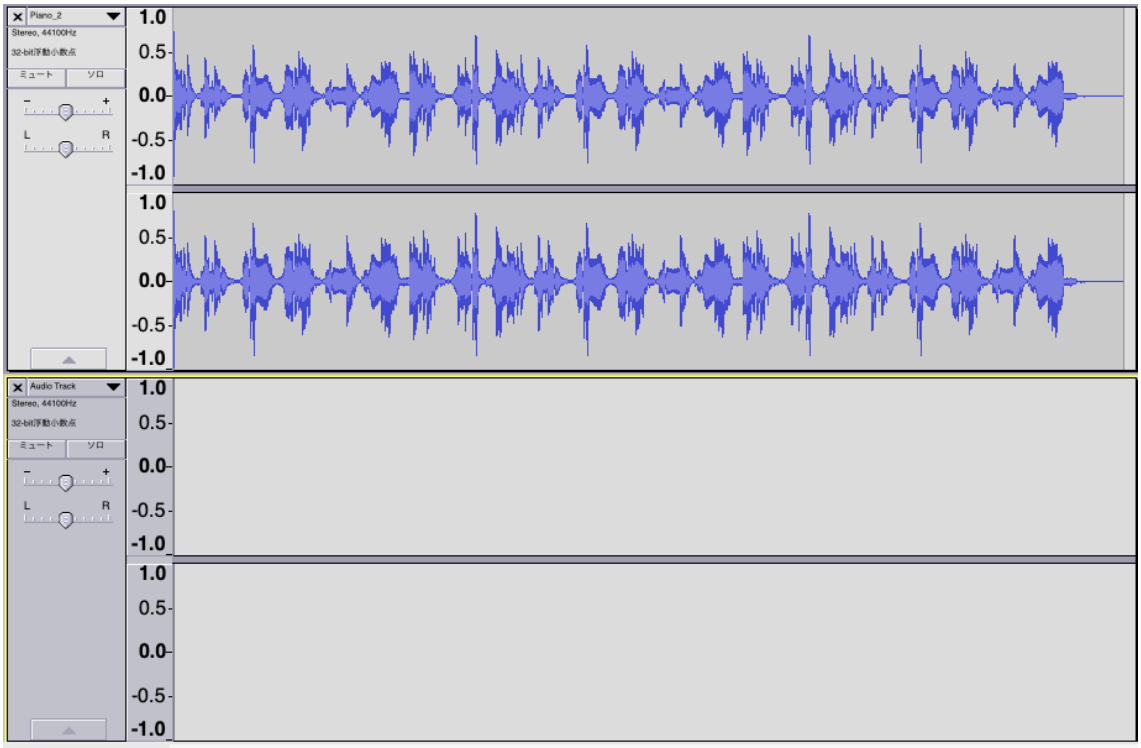

図 B.1: トラック追加後の画面

追加したトラックが選択されていることを確認した後，メニューのジェネレーターを選択しノイズを選択する．

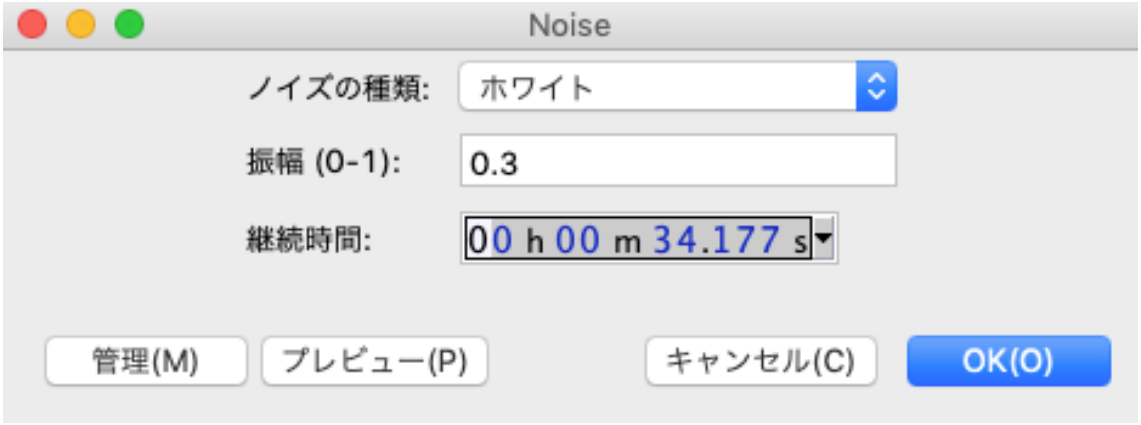

図 B.2: ホワイトノイズの設定画面

本研究では図 B.2 のような設定にした．なお継続時間は読み込んだ音声ファイルの時間に揃える．設定が完了したら OK ボタンを押下する．

結果図 B.3 のようになる．音声を確認すると元の音声とホワイトノイズがミックスされている．

メニューのファイルを選択し，オーディオの書き出しを行い音声を保存する．

B.2　2 音上げ，2 音下げ

Audacity のメニューからファイルを選択し，2 音上げまたは 2 音下げの処理をしたい WAV ファイルを選択する．

次に開いたトラックを選択した状態でメニューのエフェクトからピッチの変更を選択する．

図 B.4 に示すような Change Pitch のウィンドウが表示される．

このウィンドウ内の音程差の値を変更する．図 B.4 では 2 音上げを例にしているため音程差の値が 4 になっている．

また音程差の値を-4.00 に設定すると 2 音下げた音声にする事ができる．

メニューのファイルを選択し，オーディオの書き出しを行い音声を保存する．

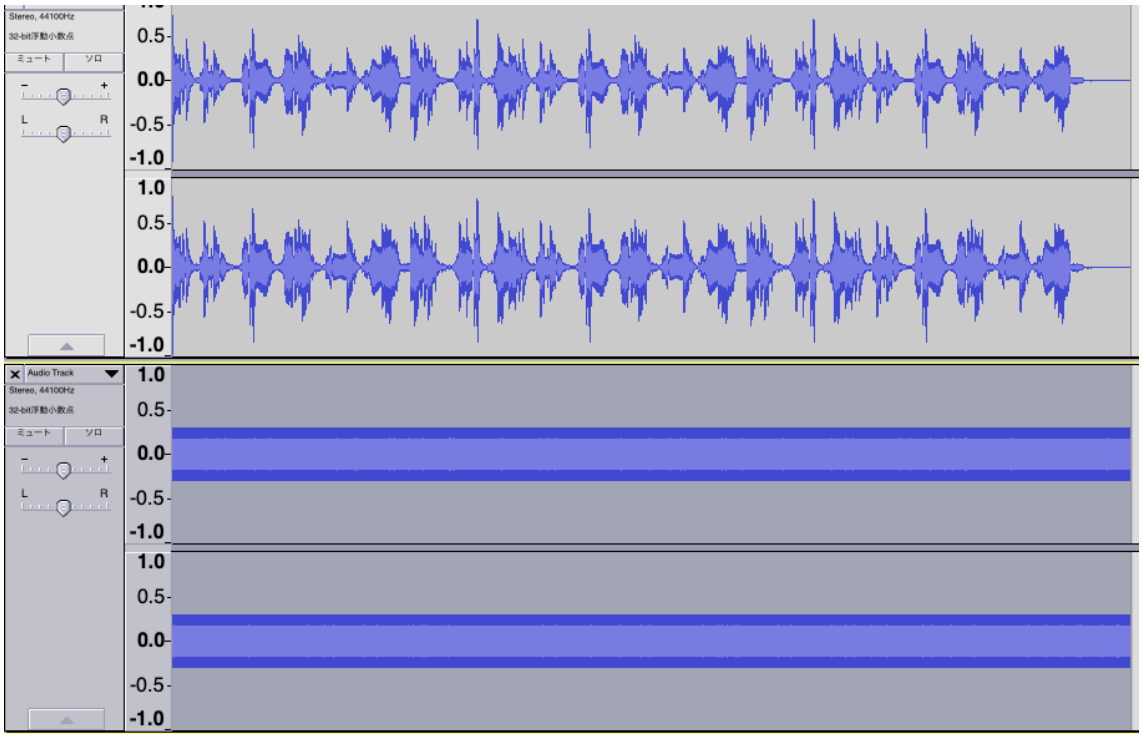

図 B.3: 元の音声にホワイトノイズが追加された画面

B.3　テンポアップ

Audacity のメニューからファイルを選択し，テンポアップ処理をしたい WAV ファイルを選択する．

次に開いたトラックを選択した状態でメニューのエフェクトからテンポの変更を選択すると，図 B.5 のような Change Tempo のウィンドウが表示される．

変更率の値を変更する．本研究では図 B.5 のように値を 100 に設定した．

メニューのファイルを選択し，オーディオの書き出しを行い音声を保存する．

図 B.4: 音程を変更する画面

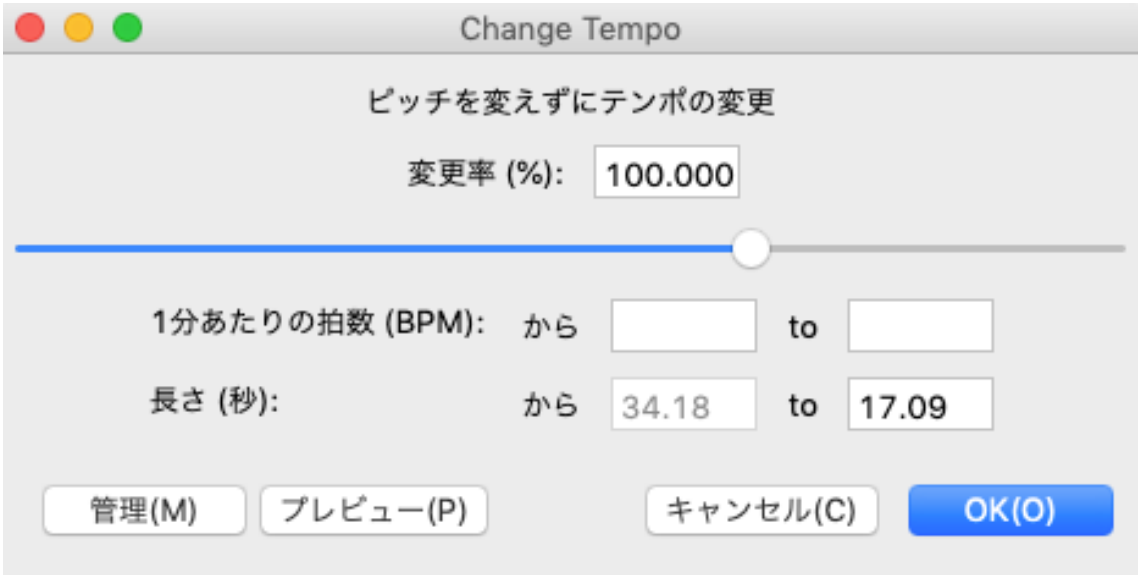

図 B.5: テンポを変更する画面

付 録C　本研究に用いたプログラムのソースコード

C.1　ファイルの切り分け

以下に音声ファイルを任意の長さに切り分けるプログラムのソースコードを示す.

```
import wave
import struct
import math
import os
from scipy import fromstring, int16


# 同じ名前のディレクトリがないか確認
file = os.path.exists("output")
print(file)

if file == False:
    # 保存先のディレクトリの作成
    os.mkdir("output")



def cut_wav(filename, time): # WAV ファイルを切り取る
    # time の単位は[sec]

    # ファイルを読み出し
    wavf = filename + '.wav'
    wr = wave.open(wavf, 'r')

    # wave ファイルが持つ性質を取得
    ch = wr.getnchannels()
    width = wr.getsampwidth()
    fr = wr.getframerate()
    fn = wr.getnframes()
    total_time = 1.0 * fn / fr
    integer = math.floor(total_time) # 小数点以下切り捨て
    t = int(time) # 秒数 [sec]
    frames = int(ch * fr * t)
    num_cut = int(integer//t)
```

```
35
36      # 確認用
37      print("Channel: ", ch)
38      print("Sample width: ", width)
39      print("Frame Rate: ", fr)
40      print("Frame num: ", fn)
41      print("Params: ", wr.getparams())
42      print("Total time: ", total_time)
43      print("Total time(integer): ", integer)
44      print("Time: ", t)
45      print("Frames: ", frames)
46      print("Number of cut : ", num_cut)
47
48
49      # wav の実データを取得し、数値化
50      data = wr.readframes(wr.getnframes())
51      wr.close()
52      X = fromstring(data, dtype=int16)
53      print(X)
54
55      for i in range(num_cut):
56          print(i)
57          # 出力データを生成
58          outf = 'output/' + str(i) + '.wav'
59          start_cut = i*frames
60          end_cut = i*frames + frames
61          print(start_cut)
62          print(end_cut)
63          Y = X[start_cut:end_cut]
64          outd = struct.pack("h" * len(Y), *Y)
65
66          # 書き出し
67          ww = wave.open(outf, 'w')
68          ww.setnchannels(ch)
69          ww.setsampwidth(width)
70          ww.setnframerate(fr)
71          ww.writeframes(outd)
72          ww.close()
73
74  print("input filename = ")
75  f_name = input()
76  print("cut time = ")
77  cut_time = input()
78  cut_wav(f_name, cut_time)
```

C.2　ファイルのリネーム

本研究では，データセットの総数を変えるためにファイルの移動を行うことが多々あった．しかし，ファイルを切り分けたままだと違うファイルなのに同じ名前がついている場合があり，同じフォルダ内に同名のファイルを保存することはできないため，名前をつけ直す必要があった．そこでフォルダ内にある全てのファイルの名前を一括でつけ直すプログラムを作成した．以下にソースコードを示す．

```
1  import glob
2  import os
3
4
5  path = "C:/trumpet_v2_tempoup100_2sec_spe" # 名前を変えたいファイルがある
       フォルダの絶対パス
6  files = glob.glob(path + '/*')         # フォルダの中の全てのファイル
7
8  # ファイルリストを絶対パスから取得
9  print(glob.glob(path + '/*'))
10
11 for i, f in enumerate(files):
12     os.rename(f, os.path.join(path, '{0:04d}'.format(i) + # 数字の桁を決
           める
13                               '_trumpet_temp.png')) # '~~.png'で後半の名前
                                     を決める
```

C.3　サウンドスペクトログラム変換

以下に音声ファイルを画像ファイルに変換するプログラムのソースコードを示す.

```
import numpy as np
import matplotlib.pyplot as plt
import wave
import os
import fnmatch
from scipy import signal
from scipy import fftpack
import soundfile as sf

#WAV ファイル読み込み
def wavload(path):
    data,samplerate=sf.read(path)
    return data,samplerate

#オーバーラップ処理
def ov(data, samplerate, Fs, overlap):
    Ts = len(data) / samplerate
    Fc = Fs / samplerate # フレーム周期
    x_ol = Fs * (1 - (overlap / 100)) # フレームずらし幅
    N_ave = int((Ts - (Fc * (overlap / 100))) / \
                (Fc * (1 - (overlap / 100)))) # 抽出するフレーム数

    array = []

    for i in range(N_ave):
        ps = int(x_ol * i) # 切り出しする位置を初期化
        array.append(data[ps:ps + Fs:1])
        final_time = (ps + Fs) / samplerate # 最終時刻
    return array, N_ave, final_time

#ハニング窓
def hanning(data_array, Fs, N_ave):
    han = signal.hann(Fs) # ハニング窓作成
    acf = 1 / (sum(han) / Fs) # 振幅補正係数(Amplitude Correction Factor
        )

    for i in range(N_ave):
        data_array[i] = data_array[i] * han

    return data_array, acf

#デシベル(dB)変換
def db(x, dBref):
    y = 20 * np.log10(x/dBref)
```

```
44      return y
45
46  #聴覚補正 (A 特性補正)
47  def aweightings(f):
48      if f[0] == 0:
49          f[0] = 1
50      else:
51          pass
52      ra = (np.power(12194, 2) * np.power(f, 4)) / \
53           ((np.power(f, 2) + np.power(20.6, 2)) * \
54            np.sqrt((np.power(f, 2) + np.power(107.7, 2)) * \
55                    (np.power(f, 2) + np.power(737.9, 2))) * \
56            (np.power(f, 2) + np.power(12194, 2)))
57      a = 20 * np.log10(ra) + 2.00
58      return a
59
60  #FFT 処理
61  def fft_ave(data_array, samplerate, Fs, N_ave, acf):
62      fft_array = []
63      fft_axis = np.linspace(0, samplerate, Fs) # 周波数軸を作成
64      a_scale = aweightings(fft_axis) # 聴感補正曲線を計算
65
66      # FFT をして配列に dB で追加、窓関数補正値をかけ、(Fs/2)の正規化を実施。
67      for i in range(N_ave):
68          fft_array.append(db \
69                                  (acf * np.abs(fftpack.fft(data_array[i]) / (
                                        Fs / 2)) \
70                                   , 2e-5))
71
72      fft_array = np.array(fft_array) + a_scale # 型を
            ndarray に変換し A 特性をかける
73      fft_mean = np.mean(fft_array, axis=0) # 全てのFFT 波形の平均を計算
74
75      return fft_array, fft_mean, fft_axis
76
77  #変換した画像データを格納するディレクトリの作成
78  file = os.path.exists("output")
79  if file == False:
80
81      os.mkdir("output")
82
83  ###############
84  #メイン処理
85  ###############
86  path='guitar_and_strings_2sec' #パスを指定 (ディレクトリ名を指定)
87  file_num=len(fnmatch.filter(os.listdir(path),'*.wav')) #ディレクトリ内の
        ファイル数
88
```

```
89
90 for a in range(file_num):
91     data,samplerate=wavload(path+'/'+str(a)+'.wav')
92
93     #処理する音声ファイルがステレオのため、Lch と Rch に分割
94     data_l=data[:,0]
95     data_r=data[:,1]
96     #入力をモノラル化
97     data=(0.5*data_l)+(0.5*data_r)
98
99     x=np.arange(0,len(data))/samplerate
100
101    Fs=4096 #フレームサイズ
102    overlap=90 #オーバーラップ率
103    # オーバーラップ抽出された時間波形配列
104    time_array, N_ave, final_time = ov(data, samplerate, Fs, overlap)
105
106    # ハニング窓関数をかける
107    time_array, acf = hanning(time_array, Fs, N_ave)
108
109    # FFT をかける
110    fft_array, fft_mean, fft_axis = fft_ave(time_array, samplerate, Fs,
           N_ave, acf)
111
112    # スペクトログラムで縦軸周波数、横軸時間にするためにデータを転置
113    fft_array = fft_array.T
114
115    # グラフを描画
116    fig = plt.figure()
117    ax1 = fig.add_subplot(111)
118
119    # データをプロットする。
120    im = ax1.imshow(fft_array, \
121                    vmin=0, vmax=60,
122                    extent=[0, final_time, 0, samplerate], \
123                    aspect='auto', \
124                    cmap='jet')
125
126    # カラーバーを設定する。
127    cbar = fig.colorbar(im)
128    cbar.set_label('SPL [dBA]')
129
130    #軸設定
131    ax1.set_xlabel('Time [s]')
132    ax1.set_ylabel('Frequency [Hz]')
133
134    # スケールの設定をする。
135    ax1.set_xticks(np.arange(0, 50, 5))
```

```
136     ax1.set_yticks(np.arange(0, 20000, 2000))
137     ax1.set_xlim(0, 2)
138     ax1.set_ylim(0, 10000)
139
140     plt.savefig('output/' + str(a) + '.png')
141     plt.close()
142 else:
143     print('Finish!!')
```

C.4　ラベル付け

以下に音声ファイルにラベルを付けるプログラムのソースコードを示す.

```
import matplotlib.pyplot as plt
import os
import cv2
import random
import numpy as np

DATADIR = "C:/dataset2" # 読み込むディレクトリの絶対パスを入力
CATEGORIES = ["bass", "guitar", "piano"]
training_data = [] # リスト作成

def create_training_data():
    for class_num, category in enumerate(CATEGORIES):
        path = os.path.join(DATADIR, category)
        for image_name in os.listdir(path):
            try:
                img_array = cv2.imread(os.path.join(path, image_name)) # 読み込み
                training_data.append([img_array, class_num]) # ラベル

            except Exception as e:
                pass
create_training_data()

random.shuffle(training_data) # シャッフル

X_train = [] # 画像データ
y_train = [] # ラベル

# データセット作成
for feature, label in training_data:
    X_train.append(feature)
    y_train.append(label)

# numpy 配列に変換
X_train = np.array(X_train)
y_train = np.array(y_train)

# 確認
for i in range(0, 4):
    print("学習データのラベル: ", y_train[i])
    plt.subplot(2, 2, i+1)
    plt.axis('off')
    if y_train[i] == 0:
        plt.title(label='bass')
```

```
44      elif y_train[i] == 1:
45          plt.title(label='guitar')
46      elif y_train[i] == 2:
47          plt.title(label='piano')
48
49
50      img_array = cv2.cvtColor(X_train[i], cv2.COLOR_BGR2RGB)
51      plt.imshow(img_array)
52
53  plt.show()
```

C.5　CNN を用いた学習

以下に CNN のプログラムのソースコードを示す.

```
import keras
from keras.models import Sequential
from keras.layers.convolutional import Conv2D
from keras.layers.pooling import MaxPooling2D
from keras.optimizers import Adam
from keras.layers.core import Dense, Activation, Dropout, Flatten
from keras.layers import Input, Add, BatchNormalization,
    GlobalAveragePooling2D
from keras.models import Model
from keras.utils import plot_model
from keras.callbacks import TensorBoard
from keras.utils import np_utils
from sklearn.model_selection import train_test_split
from PIL import Image
import glob
import cv2
import numpy as np
import os
import tensorflow as tf
import keras.backend.tensorflow_backend as ktf
import matplotlib.pyplot as plt

folder = ["bass", "flute", "guitar", "piano", "strings", "trumpet"]
image_size = 256 # リサイズ

X = [] # 画像データ
Y = [] # 正解ラベルのデータ

# numpy 形式に変換
for index, name in enumerate(folder):
    dir ="/content/drive/My Drive/7000_div/" + name
    files = glob.glob(dir + "/*.png")
    for i, files in enumerate(files):
        image = Image.open(files)
        image = image.convert("RGB")
        image = image.resize((image_size, image_size)) # リサイズ
        data = np.asarray(image)
        X.append(data)
        Y.append(index)

X = np.array(X)
Y = np.array(Y)

# 画像データを 0~1の範囲に変換
```

```
X = X.astype('float32')
X = X / 255.0

# 正解ラベルの形式を変換
Y = np_utils.to_categorical(Y, 7)

# 学習用データとテストデータに分割
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.30)

# CNN を構築
model = Sequential()

model.add(Conv2D(32, (3, 3), padding='same', input_shape=X_train.shape
    [1:]))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(7))
model.add(Activation('softmax'))

# コンパイル
model.compile(loss='categorical_crossentropy', optimizer='Adam', metrics
    =['accuracy'])


# 訓練
history = model.fit(X_train, y_train, epochs=20, batch_size=20,
    validation_data=(X_test, y_test))
model.summary()

# 評価　& 結果出力
print(model.evaluate(X_test, y_test))

# 学習の様子をグラフへ描画
```

```
89 # 正解率の推移をプロット
90 plt.plot(history.history['acc'])
91 plt.plot(history.history['val_acc'])
92 plt.legend(['Accuracy for training data', 'Accuracy for test data'], loc
       ='right')
93 plt.xlabel("epochs")
94 plt.ylabel("Accuraccy")
95 plt.xticks([0, 5, 10, 15, 20])
96 plt.show()
97
98 # ロスの推移をプロット
99 plt.plot(history.history['loss'])
100 plt.plot(history.history['val_loss'])
101 plt.title('Loss')
102 plt.legend(['Loss for trainig data', 'Loss for test data'], loc='right')
103 plt.xlabel("epochs")
104 plt.ylabel("Loss")
105 plt.xticks([0, 5, 10, 15, 20])
106 plt.show()
107
108 # ネットワークモデルの可視化
109 plot_model(model, to_file='/content/drive/My Drive/7000_div/7000model.png
       ')
110
111 model.save_weights('/content/drive/My Drive/7000_div/photos-model-light.
       hdf5')
```

C.6　楽器音識別システム

以下にCNNのプログラムのソースコードを示す.

```
1
2  import keras
3  from keras.models import Sequential
4  from keras.layers.convolutional import Conv2D
5  from keras.layers.pooling import MaxPooling2D
6  from keras.optimizers import Adam
7  from keras.layers.core import Dense, Activation, Dropout, Flatten
8  from keras.layers import Input, Add, BatchNormalization,
       GlobalAveragePooling2D
9  from keras.models import Model
10 from keras.utils import plot_model
11 from keras.callbacks import TensorBoard
12 from keras.utils import np_utils
13 from sklearn.model_selection import train_test_split
14 from PIL import Image
15 import glob
16 import cv2
17 import numpy as np
18 import os
19 import tensorflow as tf
20 import keras.backend.tensorflow_backend as ktf
21 import matplotlib.pyplot as plt
22
23 folder = ["bass", "flute", "guitar", "piano", "strings", "trumpet"]
24 image_size = 256 # リサイズ
25 image_color = 3 # color dimention
26
27
28 # CNNのモデル
29 model = Sequential()
30
31 model.add(Conv2D(32, (3, 3), padding='same', input_shape=X_train.shape
       [1:]))
32 model.add(Activation('relu'))
33 model.add(Conv2D(32, (3, 3)))
34 model.add(Activation('relu'))
35 model.add(MaxPooling2D(pool_size=(2, 2)))
36 model.add(Dropout(0.25))
37
38 model.add(Conv2D(64, (3, 3), padding='same'))
39 model.add(Activation('relu'))
40 model.add(Conv2D(64, (3, 3)))
41 model.add(Activation('relu'))
42 model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(7))
model.add(Activation('softmax'))

# コンパイル
model.compile(loss='categorical_crossentropy', optimizer='Adam', metrics
    =['accuracy'])
# 学習済みの重みを読み込む
model.load_weights('/content/drive/My Drive/7000_div/photos-model-light.
    hdf5')


def check_photo(path):
  # 画像を読み込む
  image = Image.open(path)
  image = image.convert("RGB")
  image = image.resize((image_size, image_size))
  plt.imshow(image)
  plt.show()
  # データに変換
  x = np.asarray(image)
  x = x.reshape(-1, image_size, image_size, image_color)
  x = x / 255
  # 予測
  pre = model.predict([x])[0]
  idx = pre.argmax()
  per = int(pre[idx] * 100)
  return (idx, per)

def check_photo_str(path):
  idx, per = check_photo(path)
  # 答えを表示
  print("この楽器は、", folder[idx], "です。")
  print("可能性は、", per, "% です。")

if __name__ == '__main__':
  check_photo_str('/content/drive/My Drive/mini_test/0000_bass_n.png')
  check_photo_str('/content/drive/My Drive/mini_test/0000_piano_n.png')
  check_photo_str('/content/drive/My Drive/mini_test/0000_flute_n.png')
  check_photo_str('/content/drive/My Drive/mini_test/1090_bass_n.png')
  check_photo_str('/content/drive/My Drive/Piano_and_Strings/0001
      _Piano_and_Strings_n.png')
```

謝 辞

本研究は，筆者が北海道科学大学工学部情報工学科在学中に行ったものである．

◇

本研究を進めるにあたり，終始，御指導，御鞭撻をいただいた北海道科学大学工学部情報工学科，松﨑博季教授に心より感謝致します．

また，同ゼミの良き研究仲間として御助言，御協力頂いたゼミ生に深く感謝致します．

最後に本研究の遂行にあたり著者を常に支援し応援してくれた両親，並びに親族のみなさまに心より感謝申しあげます．