

平成 27 年度 修士論文

機械学習を利用した 打楽器の音源同定

早稲田大学 基幹理工学研究科 情報理工・情報通信専攻

5114F023-5

大石 皓太郎

指導 甲藤 二郎 教授

2016 年 2 月 1 日

指導教授印	受付印

目次

1.1	はじめに	5
1.2	研究背景	5
1.3	研究目的	5
1.4	本論文の構成	6
第2章	関連技術／性質	7
2.1	音響信号処理	7
2.1.1	高速フーリエ変換	7
2.1.2	短時間フーリエ変換	8
2.1.3	窓関数	9
2.1.4	スペクトログラム	11
2.2	楽器音の時間周波数構造	11
2.2.1	調波構造	11
2.2.2	非調波構造	13
2.2.3	その他	16
2.3	距離／類似度計算手法	16
2.3.1	距離／類似度について	16
2.3.2	ユークリッド距離	16
2.3.3	データの平均、分散、標準偏差	17
2.3.4	標準化ユークリッド距離	18
2.3.4	共分散と相関係数、分散共分散行列	18
2.3.5	マハラノビス距離	19
2.3.6	マンハッタン距離	21
2.3.7	チェビシェフ距離	21
2.3.8	ミンコフスキー距離	22
2.3.9	コサイン類似度（コサイン距離）	22
2.3.10	相関係数（相関に基づく距離）	22
2.4	k 最近傍法	23
2.4.1	概要	23
2.4.2	最近傍法	23
2.4.3	ボロノイ図	23
2.4.4	kNN 法	25
2.4.4	kNN 法の計算量とその低減法	25
2.5	深層学習(Deep Learning)	25
2.5.1	深層学習の概要	25

2.5.2	畳み込みニューラルネットワーク	26
2.5.3	畳み込み層	26
2.5.4	プーリング層	27
2.5.5	全結合層	28
2.5.6	活性化関数	28
2.5.6.1	シグモイド関数	28
2.5.6.2	ReLU	28
2.5.6.3	ソフトマックス関数	29
2.5.6	誤差逆伝播法	29
第3章	従来手法	33
3.1	特徴量ベクトルを入力とした SOM による教師なしクラスタリング手法	33
3.1.1	概要	33
3.1.2	問題点	34
3.2	テンプレート適応を利用したテンプレートマッチング手法	35
3.2.1	概要	35
3.2.2	処理の流れ	35
3.2.3	問題点	37
3.2.4	提案手法に向けて	37
第4章	k 最近傍法を用いた提案手法	38
4.1	概要	38
4.2	処理の流れ	39
4.2.1	テンプレート側	39
4.2.2	楽曲側	39
4.2.3	マッチング部	40
4.3	発音時刻検出方法	42
4.3.1	打楽器単音の発音時刻検出	42
4.3.2	楽曲に含まれる楽器音の発音時刻検出	45
4.3	時間周波数構造への変換方法	46
4.4	テンプレートと楽曲のマッチング方法	46
4.5	調波音・打楽器音分離ソフトウェア HPSS について	48
第5章	k 最近傍法を用いた実験	49
5.1	実験準備	49
5.2	k 最近傍法の有効性についての検証実験	53
5.2.1	実験概要	53
5.2.2	実験準備	53
5.2.3	実験結果	53

5.3	半径探索に必要な半径の値を定める予備実験	55
5.3.1	実験概要	55
5.3.2	実験準備	55
5.3.3	実験結果	56
5.4	提案手法による実際の楽曲を対象とした認識の実験	57
5.4.1	実験概要	57
5.4.2	実験準備	58
5.4.3	実験結果	58
第 6 章	深層学習を用いた提案手法	60
6.1	概要	60
6.2	処理の流れ	60
6.2.1	教師データ側	60
6.2.2	楽曲側	61
6.2.3	深層学習による識別器作成	61
6.2.4	マッチング部	61
第 7 章	深層学習を用いた実験	63
7.1	実験準備	63
7.2	Caffe について	63
7.2.1	概要	63
7.2.2	ネットワーク定義ファイル	63
7.2.3	solver 定義ファイル	64
7.3	教師データの一部を対象とした認識実験	64
7.3.1	概要	64
7.3.2	予備実験	65
7.3.3	学習率の変更による認識率の変化の調査実験	65
7.3.4	画像データサイズの変更による認識率の変化の調査実験	66
7.4	提案手法による実際の楽曲を対象とした認識の実験	66
7.4.1	概要	66
7.4.2	実験結果	67
第 8 章	総括	69
8.1	まとめ	69
8.2	今後の課題	70
謝辞	71
参考文献	72
発表文献リスト	74

第 1 章 序論

1.1 はじめに

近年のコンピュータの処理能力の向上速度は著しく、それに伴い今まで膨大な計算コストが障壁となっていた分野、例えば音響信号処理などは特に研究が活発になっている。そんな音響信号処理による処理の一つ、音楽演奏に使われている楽器の種類を特定する「音源同定」技術も例外ではない。昔から存在する技術ではあるが、その性能はここ数十年で飛躍的に進歩している。例えば管楽器、弦楽器など明確な音高（音の高さ）が存在する楽器は、その音が含む周波数成分に調波構造と呼ばれる特徴的な性質が存在するため、それを足がかりとした有力な音源同定手法が数多く存在している[1][2]。調波構造を持っているという条件さえ満たしていれば、既存のデータが存在しないような未知の楽器でも既知のどの楽器群に近いのかを判断できるという、カテゴリレベルでの音源同定の研究[3]も行われているほどである。

1.2 研究背景

打楽器を対象とした音源同定に関する研究は、他の楽器を対象としたものに比べるとあまり進んでいない。これの原因としてはまず、音高のない打楽器は他の楽器と違い調波構造を持っていないため、他の楽器と同様の手法が適用できないことが非常に大きい。それに加え、打楽器は同じ楽器種類でも実験において無視できない個体差が存在することが多いにもかかわらず、研究用のデータベースが不足気味であることも理由の一つだと考えられる。従来研究においても、同定対象を単音のみにしていたり[4]、混合音でもドラムセットの構成楽器に限定する[5]など、ある程度制約条件を設定していることが多い。

しかし、実際の音楽演奏は複数の楽器が同時に発音するのが当然であり、使われている打楽器の種類也多岐にわたる。実音源を同定対象にするならば、このような条件でも同定できるほうが望ましい。

1.3 研究目的

本研究では、打楽器音とそれ以外の楽器音を含み、それらが同時に発音することもある混合音を対象に音源同定を行う。予め楽器の材質や奏法などによって打楽器をカテゴリ化しておき、混合音中に含まれる打楽器音について、それぞれの楽器カテゴリに属するか識別することを目的とする。

採譜をする際や聴いた曲を自分で演奏したくなった場合など、楽器が何なのか知りたくなるケースは数多く存在する。しかし、楽曲中に数回しか登場しないような打楽器は数多く存在する。それらを自分の耳で聴きとって楽器を判断するというをしなくとも、この研究によりコンピュータで楽器を自動判別することが可能となる。また、自分が聴いた

ことのない、あるいは聴いたことがあっても楽器名がわからない音でも、データベースに存在しさえすれば楽器名を知ることが可能となる。

1.4 本論文の構成

本論文は 8 つの章から構成されている。まず第 1 章で研究背景、研究目的について述べた。第 2 章では本研究において重要となる性質、技術に関して述べる。第 3 章では研究内容と関連した従来手法について述べ、第 4 章ではその従来手法を元にした、 k 最近傍法による提案手法について述べる。第 5 章では k 最近傍法による提案手法に関する実験について述べる。第 6 章では k 最近傍法による提案手法を元にした、深層学習による新たな提案手法について述べ、第 7 章では、その提案手法に関する実験について述べる。最後に第 8 章では本論文の総括を行い、本論文のまとめとする。

第 2 章 関連技術／性質

2.1 音響信号処理

2.1.1 高速フーリエ変換

フーリエ変換は時間領域の関数を周波数領域に変換する技術である。その一般的な式は

$$X(\omega) = \int_{-\infty}^{+\infty} x(t)e^{-j\omega t} dt \quad (2.1)$$

で表されるが、これを離散化されたデジタル信号の周波数解析などに利用できるようにしたものが離散フーリエ変換(DFT : Discrete Fourier Transform)である。

入力する時間領域の離散信号列を x_0, \dots, x_{N-1} 、出力される周波数領域の離散信号列を X_0, \dots, X_{N-1} とすると、離散フーリエ変換は

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j\frac{2\pi kn}{N}} \quad (2.2)$$

という式で定義される。また、この逆変換にあたる逆離散フーリエ変換(IDFT : Inverse Discrete Fourier Transform)は、

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{j\frac{2\pi kn}{N}} \quad (2.3)$$

という式で定義される。ここで、 e はネイピア数、 j は虚数単位、 π は円周率である。

離散フーリエ変換は直接計算する際、時間計算量が $O(N^2)$ となる。これをコンピュータ上で高速に計算するために考えだされたのが高速フーリエ変換(FFT : Fast Fourier

Transform)である。

例えば代表的な FFT アルゴリズムである Cooley-Tukey 型アルゴリズムでは、分割統治法を使うことにより、データ数が 2 の累乗のときに時間計算量が $O(N \log N)$ となる。一般的にデータ数は 2 の累乗にならないので、素因数が偶数の場合と奇数の場合で別々のアルゴリズムに分岐する。その場合、データ数が $N = \prod n_i$ と素因数分解できるときの時間計算量は $O(N \sum n_i)$ となる。

いずれの場合にしても、高速フーリエ変換を使用することにより、離散フーリエ変換に比べて大幅に計算量を減らすことができ、計算に必要な時間やメモリの大幅な短縮につながる。

2.1.2 短時間フーリエ変換

信号をフーリエ変換することにより信号に含まれる周波数成分と、その相対的な強さを算出して周波数領域の解析が可能となるが、通常の変換では信号の全区間を変換するため時間に関する情報が完全に失われてしまう。定常的な信号であれば問題ないが、一般的な音楽信号は時間変化する非定常的な信号であるため時間的な情報が重要となる。

そこで、時間情報を残しつつ周波数領域に変換する方法として用いられるのが短時間フーリエ変換(STFT: Short-Time Fourier Transform)である。短時間フーリエ変換は、信号に対し窓関数をずらしながらかけ、時間軸方向に短い区間ごとにフーリエ変換を施す。これにより時間軸方向のシフト係数と周波数の2次元の関数として信号を表現できる。

短時間フーリエ変換は次の式で表される。

$$\text{STFT}_{x,w}(t, \omega) = \int_{-\infty}^{\infty} x(\tau)w(\tau - t)e^{-j\omega\tau}d\tau \quad (2.4)$$

ここで $w(t)$ は窓関数である。詳細は次の項で述べる。

短時間フーリエ変換により、時間情報と周波数情報が得られるが、これらの情報の不確定さの間には、窓の大きさに対して常にトレードオフが存在する。これを不確定性原理といい、

$$\Delta x \Delta \omega \geq \frac{1}{2} \quad (2.5)$$

の関係がある。時間分解能・周波数分解能はともに窓の大きさによって決まり、ウインドウサイズが大きいと周波数分解能が良いが時間分解能が悪く、逆にウインドウサイズが小さいと時間分解能は良いが周波数分解能が悪い。

短時間フーリエ変換では全ての周波数に対して同じ大きさの窓が適用される。つまり、分解能が一定となる。短時間フーリエ変換の解像度のイメージ図を Fig.2.1 に示す。

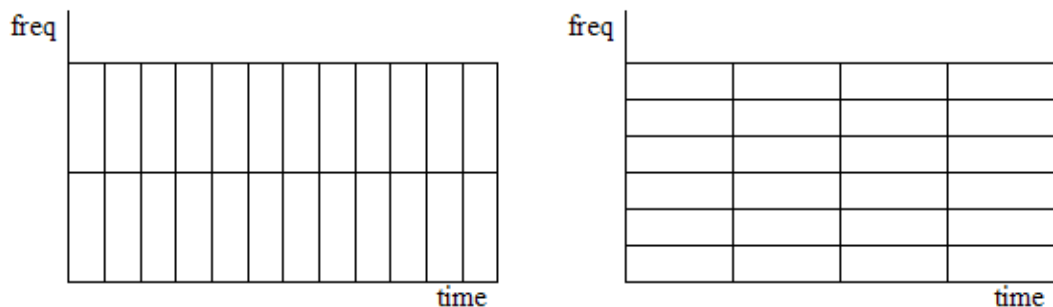


Fig.2.1 短時間フーリエ変換の解像度^[6]
(左は時間分解能が高く、右は周波数分解能が高い)

2.1.3 窓関数

短時間フーリエ変換では時間軸方向に短い区間ごとにフーリエ変換を施すが、フーリエ変換では無限区間で積分を行う必要があるため、この短い区間の信号が無限に繰り返されているものとしてフーリエ変換を行なう。しかし、この短い区間の信号をそのまま繋げただけでは、繋ぎ目の部分が不連続になってしまい、結果に影響を及ぼす。そこで、区間の中心付近では信号がほぼ元のままだが、両端付近では0に近づくように信号を変形させる。この変形を行なうために掛け合わせる関数のことを窓関数という。

窓関数の特徴としては、通常 $t=0$ が中央で1付近の値となり、そこから両端に向かって0に収束していく、山のような形をしている関数である。

以下、最もよく使われる窓関数を3つ挙げる。

・矩形窓

$$w(n) = 1, \text{ if } 0 \leq n \leq N-1$$
$$w(n) = 0, \text{ otherwise}$$

(2.6)

方形窓ともいい、理論上周波数分解能が一番高いが、両端で不連続になる。区間内では元の信号そのままであるため、有限長の信号データはこの窓を全体にかけていると考えることもできる。

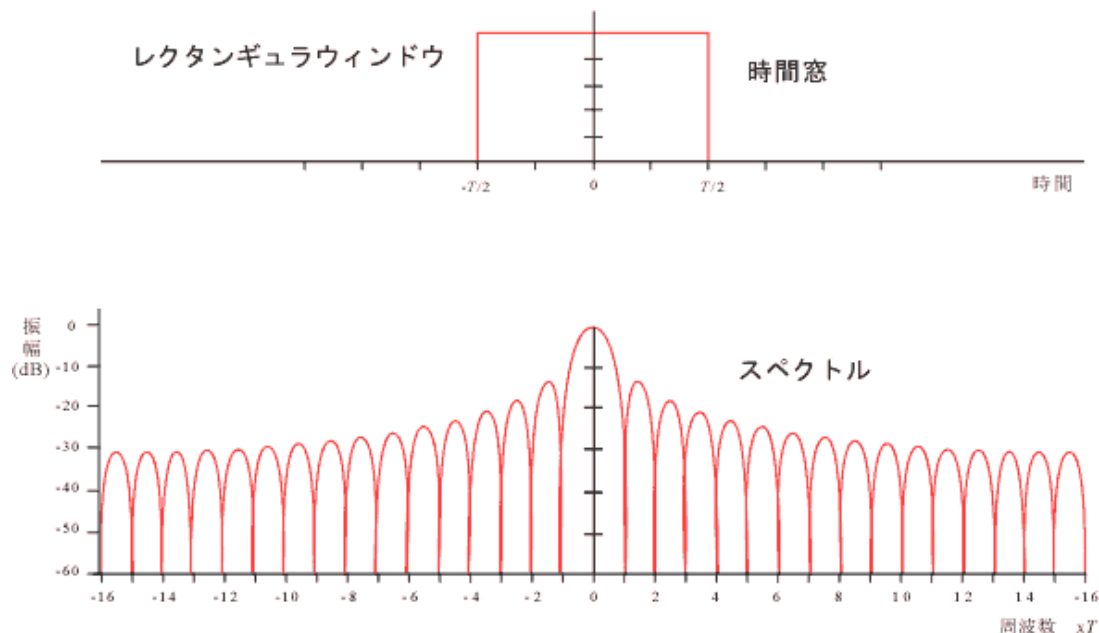


Fig.2.2 矩形窓関数とその周波数スペクトル[7]

・ハニング窓

$$w(n) = 0.5 - 0.5\cos\frac{2\pi n}{N-1}, \text{ if } 0 \leq n \leq N-1$$
$$w(n) = 0, \text{ otherwise}$$

(2.7)

ハン窓と呼ばれることもある。後述のハミング窓よりダイナミック・レンジが広い。

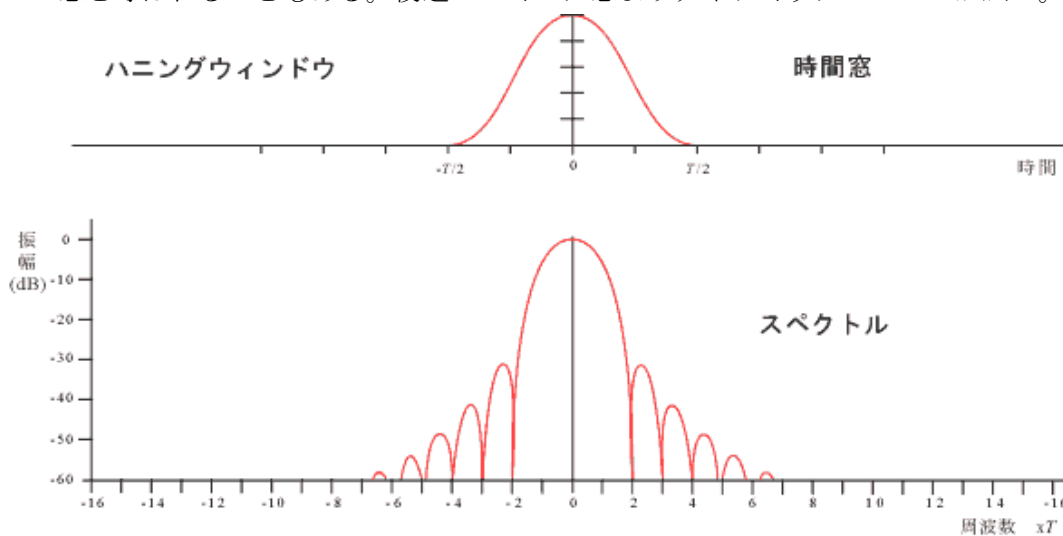


Fig.2.3 ハンニング窓関数とその周波数スペクトル[7]

・ハミング窓

$$w(n) = 0.54 - 0.46 \cos \frac{2\pi n}{N-1}, \text{ if } 0 \leq n \leq N-1$$

$$w(n) = 0, \text{ otherwise}$$

(2.8)

ハンニング窓の改良版として考案された。ハンニング窓より周波数分解能が高い。また、区間の両端で不連続になっている。

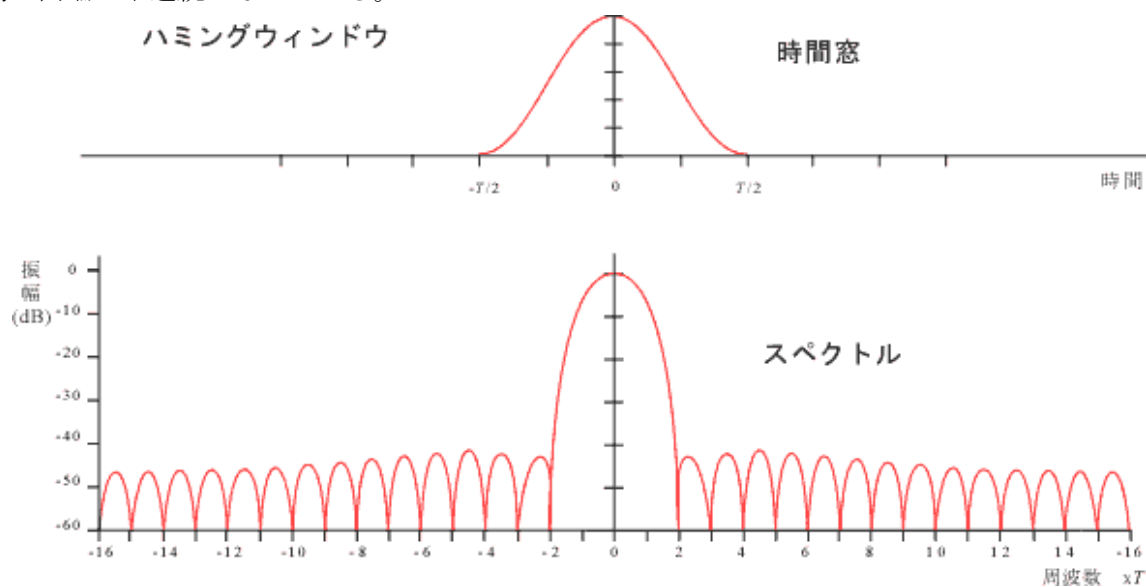


Fig.2.4 ハミング窓関数とその周波数スペクトル[7]

2.1.4 スペクトログラム

短時間フーリエ変換やウェーブレット変換ではそれぞれ時間情報と周波数情報の 2 次元の関数により、それぞれの値に関する振幅の大きさが与えられるが、この結果をグラフ化したものがスペクトログラムである。

スペクトログラムは音韻の弁別に有意な声道の変位に対応するような形で共振成分（フォルマント）が簡単に抽出できることから、音響音声学で広く用いられている。声紋の鑑定などにも使われ、そのためスペクトログラム自体が声紋と呼ばれることもある。

一般的に横軸が時間、縦軸が周波数を表していて、それぞれの値に関する成分の大きさがグラフ上で色や明るさなどにより表現される。周波数軸と成分の大きさは場合により線形目盛と対数目盛が使い分けられるが、本論文では周波数軸は線形目盛、成分の大きさは対数目盛で表すこととする。

スペクトログラムは音楽信号の時間周波数構造を確認する上で視覚的にわかりやすく、本論文の以降の項でも頻繁に登場する。

2.2 楽器音の時間周波数構造

2.2.1 調波構造

第 1 章でも述べたとおり、楽器音はその時間周波数構造にそれぞれ異なる特徴を持っている。その中でも、例えばトランペットなどの金管楽器やフルートなどの木管楽器を含めた管楽器、バイオリンなどの擦弦楽器やギターなどの撥弦楽器を含めた弦楽器が代表的であるが、明確な音高（音の高さ）が存在する楽器は「調波構造」と呼ばれる時間周波数構造を持っている。

これらの楽器音には基本となる周波数成分の他に、周波数が 2 倍や 3 倍などの周波数成分（倍音成分）も一緒に含まれている。例えばクラリネットでは偶数次倍音（2 倍、4 倍…）が奇数次倍音（3 倍、5 倍…）より小さいという特徴がある[8]。また、楽器ごとに長さに差はあるが、一定時間の間、定常的な音を発するため、発音している間はその周波数成分はほとんど時間変化しない。これらの特徴から、時間周波数構造では振幅の大きい部分が時間軸方向に伸びているという状態がいくつかの周波数帯で見られる。

Fig.2.5~2.8 に調波構造を持ついくつかの楽器の単音の波形を、短時間フーリエ変換して得られたスペクトログラムを示す。

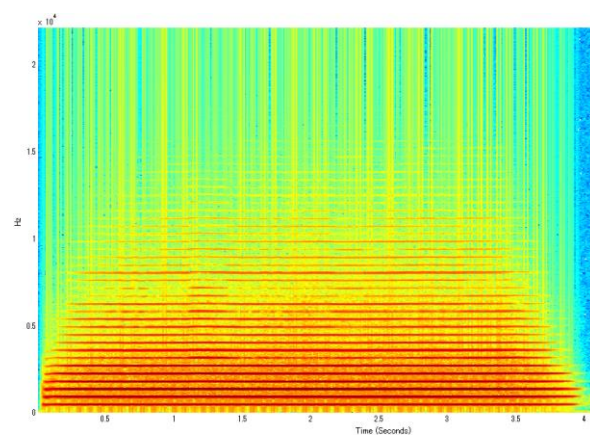


Fig.2.5 トランペットのラの音(440Hz)のスペクトログラム

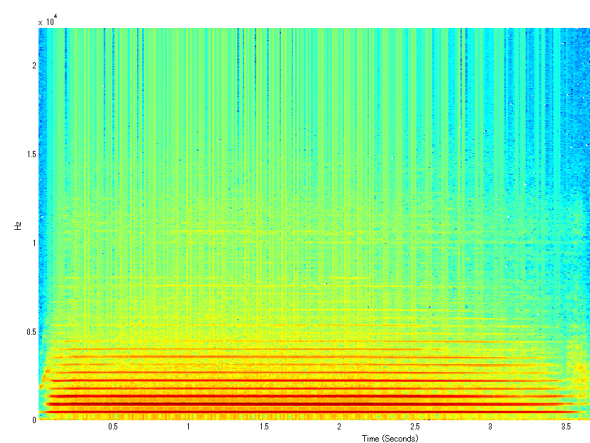


Fig.2.6 フルートのラの音(440Hz)のスペクトログラム

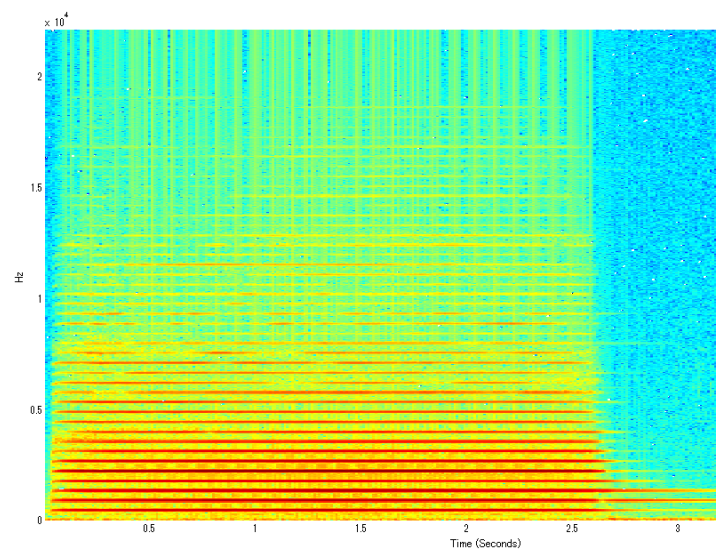


Fig.2.7 バイオリンのラの音(440Hz)のスペクトログラム

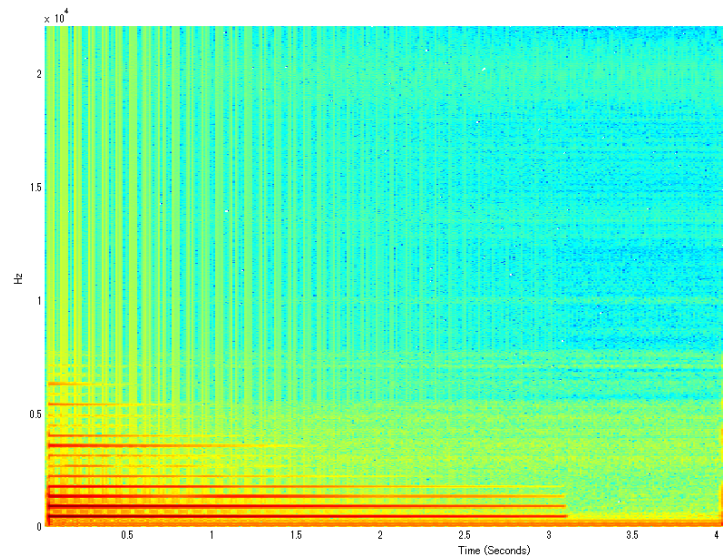


Fig.2.8 エレキギターのラの音(440Hz)のスペクトログラム

このように同じ音高でも楽器によりその時間周波数構造が異なっていることがわかる。この違いを各楽器の特徴量として、音源同定や音源分離などに利用している研究が数多く存在している。

2.2.2 非調波構造

明確な音高が存在する楽器が調波構造を持っている一方で、音高が明確でない多くの打楽器音は調波構造を持っていない。

これらの打楽器音はあらゆる周波数成分を含んでおり（明確な音高を持たない原因）、また多くの打楽器は発音後一瞬で、それ以外も徐々に音が小さくなる。これにより周波数成分が時間変化（減衰）することになる。これらの特徴から、時間周波数構造では振幅の大きい部分が周波数軸方向に伸びているという状態が発音後すぐの時間帯で見られる。

調波構造を持たない楽器の例としてスネアドラムとハイハットシンバルの単音の波形を個体・奏法ごとにいくつか用意し、それを短時間フーリエ変換して得られたスペクトログラムを Fig.2.9~2.12 に示す。

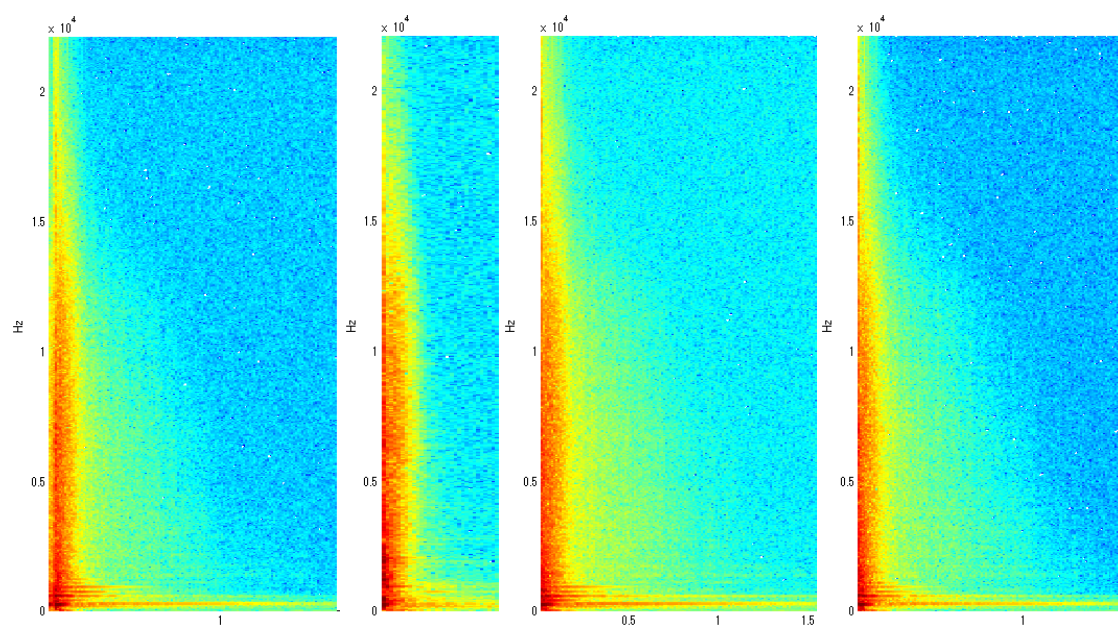


Fig.2.9 スネアドラムを4つの奏法で叩いたときのそれぞれのスペクトログラム

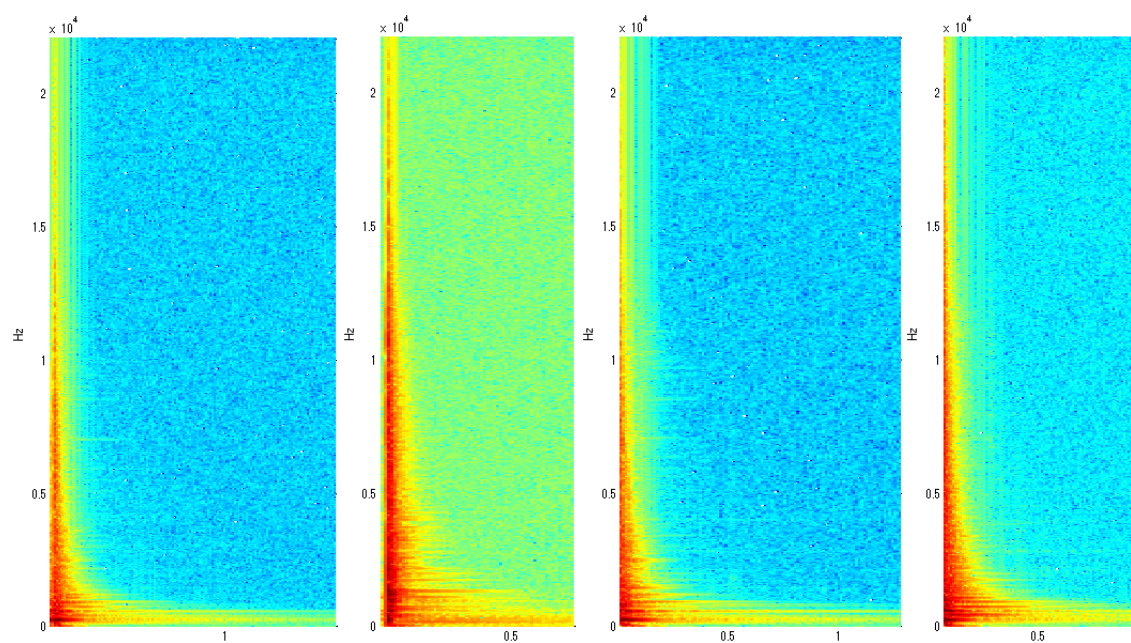


Fig.2.10 2.9とは別のスネアドラムを4つの奏法で叩いたときのスペクトログラム

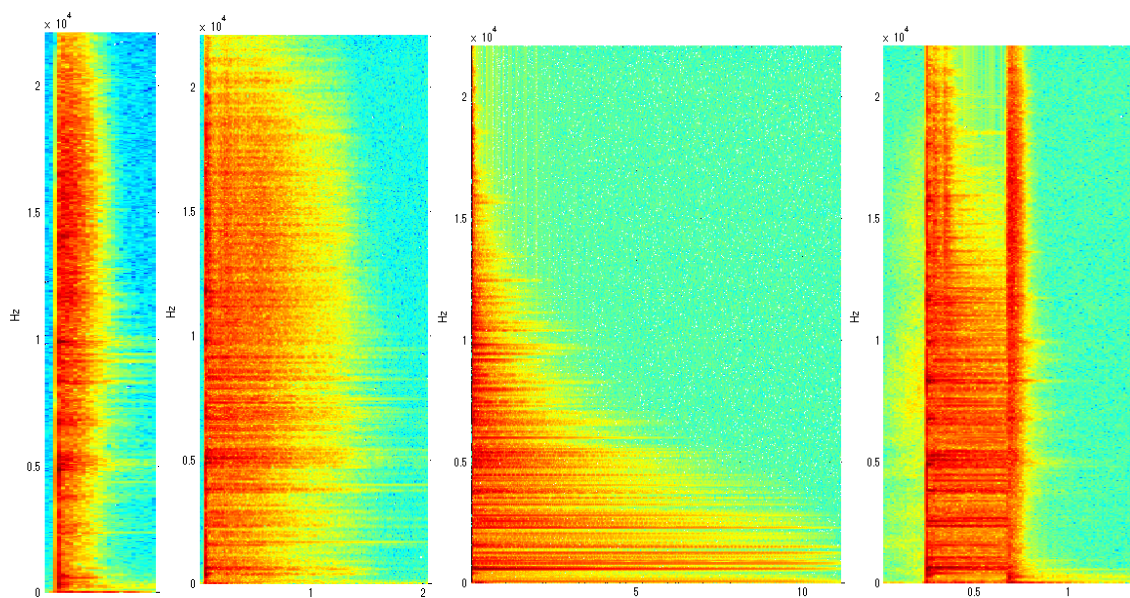


Fig.2.11 ハイハットシンバルを4奏法で叩いたときのそれぞれのスペクトログラム

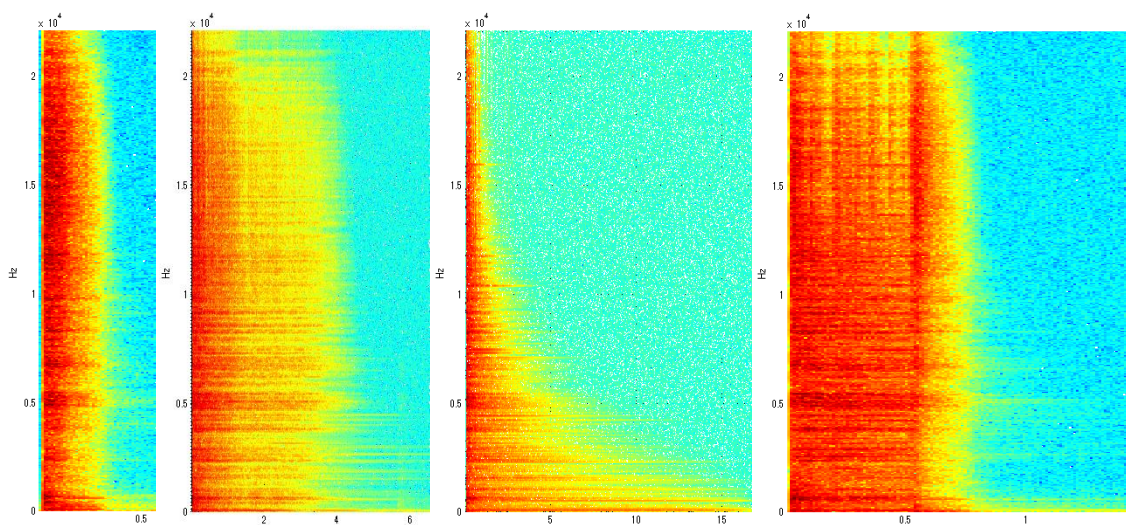


Fig.2.12 2.11とは別のハイハットシンバルを4奏法で叩いたときのスペクトログラム

このように非調波構造のパワー分布は、楽器ごとに大きな違いが存在するだけでなく、同じ楽器でもその個体や奏法によってある程度異なってくることがわかる。次の章でも触れるが、この違いが打楽器音の音源同定において1つの壁になっているといえる。

また、調波構造がスペクトログラムではきれいな縞模様になっていたのに対し、非調波構造のスペクトログラムはぼんやりと広がっている感じになる。

2.2.3 その他

調波構造のみ、または非調波成分のみからなる楽器音も多いが、ピアノのように、調波構造と非調波成分を併せ持つ（鳴り始めに弦をハンマーが叩くときには非調波音を、弦が振動しているときには調波音を発する）楽器音もある[9]。ギターなども、弦を弾く際にわずかに非調波成分が存在している。

2.3 距離／類似度計算手法

2.3.1 距離／類似度について

提案手法で使用するようになるテンプレートマッチングやk最近傍法(kNN法)において、2つのデータが似ている度合いを類似度の大きさや距離の近さといった数値にして表現する方法は非常に重要である。これらの手法以外においても、類似度や距離を計算することで機械学習を用いたさまざまな分析、例えばクラスタ分析などが可能となる。

類似度という概念は、2つの集合の要素が文字通りどれだけ似ているかを数量化したものであり、距離とは、要素同士の離れ具合、したがって非類似度と近い概念だと考えてもよい。

集合 X の直積 $X \times X$ 上の関数 $d: X \times X \rightarrow \mathbb{R}$ が次の条件、

- (1) (正值性) 任意の $x, y \in X$ に対して、 $d(x, y) \geq 0$ である。また任意の $x \in X$ について $d(x, x) = 0$ であり、 $d(x, y) = 0$ であるのは $x = y$ の場合に限る。
- (2) (対称性) 任意の $x, y \in X$ に対して、 $d(x, y) = d(y, x)$
- (3) (三角不等式) 任意の $x, y, z \in X$ に対して、 $d(x, y) + d(y, z) \geq d(x, z)$

を満たすとき、 d を距離あるいは距離関数といい、組 (X, d) を距離空間という（距離の公理）[10]。この距離を満たす定義は無限にあるが、一般に使われている距離となるとある程度限られてくる。

この項では、その中でも特徴量データに適用可能な距離や類似度について、またそれらの計算に必要な前提知識について述べていく。

2.3.2 ユークリッド距離

日常的にも使われているなじみのある距離尺度であり、さまざまな距離尺度の基本ともいえる。

$$\mathbb{R}^n = \{x = (x_1, x_2, \dots, x_n); x_1, x_2, \dots, x_n \in \mathbb{R}\} \quad (2.9)$$

に、二点 $x = (x_1, \dots, x_n)$, $y = (y_1, \dots, y_n)$ の間の距離を

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + \dots + (x_n - y_n)^2}$$

(2.10)

で定義したものがユークリッド距離である。また、ユークリッド距離を定義したこの空間のことを n 次元ユークリッド空間という [11]。

\mathbb{R}^n の要素は同時に n 次元ベクトルとも考えられる。 $d(\mathbf{x}, 0)$ を簡単に $\|\mathbf{x}\|$ で表し、ベクトルの大きさ（長さ）という。距離は $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|$ と表すこともできる。

また、

$$(\mathbf{x}, \mathbf{y}) = x_1 y_1 + \cdots + x_n y_n \quad (2.11)$$

を 2 つのベクトルの内積という。このとき、

$$\|\mathbf{x} + \mathbf{y}\|^2 = \|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 + 2(\mathbf{x}, \mathbf{y}) \quad (2.12)$$

である。内積を用いると、式(2.10)は

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y}, \mathbf{x} - \mathbf{y})} \quad (2.13)$$

と表すことができ、さらに転置行列 $(\mathbf{x} - \mathbf{y})^t$ を用いることで

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})(\mathbf{x} - \mathbf{y})^t} \quad (2.14)$$

と表せる。

2.3.3 データの平均、分散、標準偏差

データの持つ特徴を数量的に表現することを考えた場合、データの分布の中心的位置として、データの平均(mean)

$$\bar{x} := \frac{1}{n} \sum_{i=1}^n x_i \quad (2.15)$$

がよく用いられる [12]。また、データの分布の中心的位置だけでは分布の特徴は捉えきれないので、データのばらつきも考える必要がある。もしすべての値が同じなら、それはもちろん \bar{x} に等しいので、ばらつきを考えるときは値 x_i が \bar{x} からどの程度離れているかを偏差 $x_i - \bar{x}$ の関数で表すことになる。それを $(x_i - \bar{x})^2$ にとるとき、

$$\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2 \quad (2.16)$$

をデータの分散といい(variance)、通常 s^2 で表す。分散は測定単位の 2 乗という単位を持つので、

$$s = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2} \quad (2.17)$$

とすることで、もとの測定単位に戻る。平方根関数は単調増加であるので、ばらつきの尺度になりうる。これを標準偏差(standard deviation)という。計算するときには

$$s = \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2 - \bar{x}^2} \quad (2.18)$$

を使うと便利である。

2.3.4 標準化ユークリッド距離

データ分析の場合、ある項目（次元）のデータが他の項目（次元）のデータに比べて取りうる値が非常に大きいときがある。その場合、距離の違いはほぼその次元の違いになってしまう、他の次元のデータの差異が距離にほとんど反映されなくなる。

これを解決するため、各次元をその次元の取りうる値の標準偏差 s_i で割り、値の分散を標準化する。

$$d(x, y) = \sqrt{\left(\frac{x_1 - y_1}{s_1}\right)^2 + \dots + \left(\frac{x_n - y_n}{s_n}\right)^2} \quad (2.19)$$

このときのユークリッド距離が標準化（正規化）ユークリッド距離である。

ユークリッド距離の場合と同様に、データを n 次元ベクトル化して考えると、式(2.19)は

$$d(x, y) = \sqrt{(x - y)^t V^{-1} (x - y)} \quad (2.20)$$

ここで、 V は i 番目の対角要素が s_i^2 である n 行 n 列の対角行列である。

2.3.4 共分散と相関係数、分散共分散行列

データを n 次元ベクトル化して考えた場合、データはそれぞれの変量の平均をベクトル化した平均ベクトルの周りに分布する。この分布の広がり方を、以下の分散共分散行列 Σ で表す（共分散行列と略記される事が多い）^[13]。

$$\begin{aligned} \Sigma &= \begin{bmatrix} \sigma_{11} & \cdots & \sigma_{1d} \\ \vdots & \ddots & \vdots \\ \sigma_{d1} & \cdots & \sigma_{dd} \end{bmatrix} \\ &= (\sigma_{ij}) = \begin{cases} i = j & \text{分散} \\ i \neq j & \text{共分散} \end{cases} \end{aligned}$$

(2.21)

データが N 個与えられている場合は、 n 番目のデータの i 番目の変量を x_{ni} 、 j 番目の変量を x_{nj} で表せば、共分散は

$$\sigma_{ij} = \frac{1}{N} \sum_{n=1}^N (x_{ni} - \mu_i)(x_{nj} - \mu_j) \quad (2.22)$$

のように表される。

i 番目と j 番目のデータ間の相関係数 ρ_{ij} は、それぞれの標準偏差 σ_i と σ_j 、共分散 σ_{ij} を用いて

$$\rho_{ij} = \frac{\sigma_i \sigma_j}{\sigma_{ij}} \quad (2.23)$$

として定義されるので、 $-1 \leq \rho_{ij} \leq 1$ の範囲の値をとる。式(2.22)から、 x_i が平均 μ_i より大きい（小さい）とき、 x_j も μ_j より大きく（小さく）なる場合が多いと相関係数は正になる。逆に、 x_j は μ_j より小さく（大きく）なる場合が多いと相関係数は負になる。そのような規則性がない場合、相関は 0 になる。

2.3.5 マハラノビス距離

ある項目（次元）と別の項目（次元）の取りうる値に相関がある場合、相関のある方向に対して平行にデータが散らばりやすいので、前述した二つの距離尺度だとその方向の差異が距離を大きく支配してしまう。これを解決するため、相関のある方向に平行な距離を相対的に短く、垂直な距離を相対的に長くしたものがマハラノビス距離である。

2 変量の場合を考える。変量データ x_1, x_2 の平均をそれぞれ μ_1, μ_2 、分散をそれぞれ s_1^2, s_2^2 とする。ここで、簡単のために、データ x_1, x_2 を

$$u_1 = \frac{x_1 - \mu_1}{s_1}, u_2 = \frac{x_2 - \mu_2}{s_2} \quad (2.24)$$

と標準化すると、平均は u_1, u_2 とも 0、分散はともに 1 となる。

さらに、 u_1, u_2 を互いに相関のない変量 z_1, z_2 に変換する。標準化された 2 変量の主成分は相関係数によらず常に同じで、

$$z_1 = \frac{u_1 + u_2}{\sqrt{2}}, z_2 = \frac{u_1 - u_2}{\sqrt{2}} \quad (2.25)$$

となる。このように変換してしまうと、2 つの変量の相関、すなわち「分布が散布図上でどちら向きに傾いているか」はもう考える必要が無い。そこで、散布図上のある 1 点 (z_1, z_2) と

分布の中心 (x_1, x_2 軸上では (u_1, u_2) 、 z_1, z_2 軸上では $(0, 0)$) との「分散で標準化した」ユークリッド距離を、

$$D^2 = \frac{z_1^2}{V(z_1)} + \frac{z_2^2}{V(z_2)} \quad (2.26)$$

で標準化した平方距離の和で表す。この D^2 、またはその平方根である D をマハラノビス距離（またはマハラノビスの汎距離）という [14]。

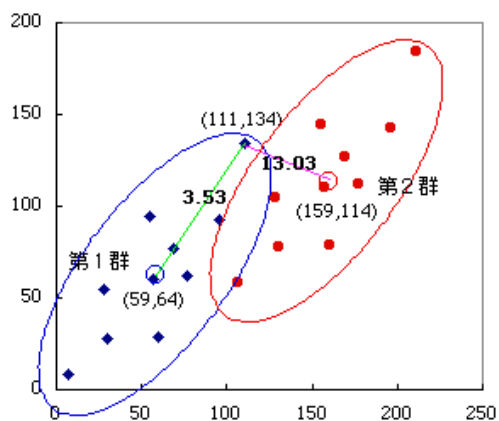
また、 D を行列を用いて式変形すると、

$$\begin{aligned} D &= \sqrt{\left(\frac{u_1 - v_1}{\sigma(z_1)}\right)^2 + \left(\frac{u_2 - v_2}{\sigma(z_2)}\right)^2} \\ &= \sqrt{(u_1 - v_1) \left(\frac{1}{\sigma(z_1)^2}\right) (u_1 - v_1)^t + (u_2 - v_2) \left(\frac{1}{\sigma(z_2)^2}\right) (u_2 - v_2)^t} \\ &= \sqrt{(u_1 - v_1, u_2 - v_2) \begin{bmatrix} \frac{1}{\sigma(z_1)^2} & 0 \\ 0 & \frac{1}{\sigma(z_2)^2} \end{bmatrix} (u_1 - v_1, u_2 - v_2)^t} \\ &= \sqrt{(u - v) \begin{bmatrix} \frac{1}{\lambda_1} & 0 \\ 0 & \frac{1}{\lambda_2} \end{bmatrix} (u - v)^t} \\ &= \sqrt{P^t (x - y) D^{-1} P^t (x - y)^t} \\ &= \sqrt{(x - y) P D^{-1} P^t (x - y)^t} \\ &= \sqrt{(x - y) \Sigma^{-1} (x - y)^t} \end{aligned} \quad (2.27)$$

となる [15]。ただし、 $\sigma(z_1)^2 = \lambda_1, \sigma(z_2)^2 = \lambda_2, P^t x = u, P^t y = v$ であり、転置行列の性質より $P^t x = xP$ であるのを利用する。

式(2.27)は、2 変量の場合に限らず、任意の数の変量で成り立つ。

マハラノビス距離は（標準化）ユークリッド距離を一般化したものであり、（標準化）ユークリッド距離はマハラノビス距離の特殊な場合であるということもできる。具体的には、マハラノビス距離の共分散 $\sigma(x - y)_{ij} = 0$ のときは標準化ユークリッド距離に、さらに分散 $s_i^2 = 1$ のときはユークリッド距離に一致する（このとき、共分散行列は単位行列となる）。



図の(111,134)の点と(59,64)、(159,114)の点それぞれとの距離を測る。緑の線と紫の線がユークリッド距離を表していて、青い楕円と赤い楕円はマハラノビス距離における等距離の範囲を表している。

Fig.2.13 ユークリッド距離とマハラノビス距離の関係^[16]

2.3.6 マンハッタン距離

L1 距離、市街地距離ともいう。マンハッタンや京都のような碁盤の目のような街を移動するときの距離であり、どこを通っても最短距離は等しくなる。図 2.14 に例を示すが、地点 P から地点 Q に行く時には最低でも 10 ブロックを通過しなくてはならない^[17]。

$$d(x,y) = \sum_{i=1}^n |x_i - y_i| \quad (2.28)$$

2 乗していないので外れ値の影響を抑えることができる。

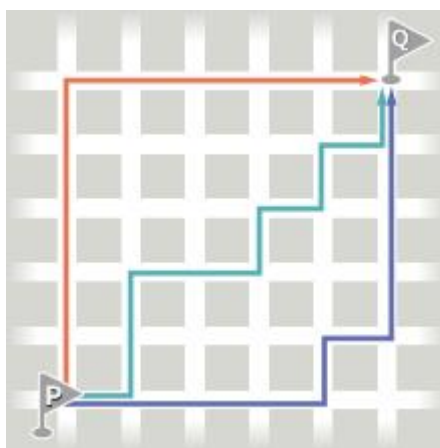


Fig2.14 マンハッタン距離のイメージ図^[17]

2.3.7 チェビシェフ距離

ユークリッド距離が原点を中心に円状に広がっていくのに対し、チェビシェフ距離は斜めも同じ距離と考えるので、正方形状に広がっていく距離となる。同じ次元の変数を、別の次元の変数とみなしたい場合に使う。

$$d(x,y) = \max_i \{|x_i - y_i|\}$$

(2.29)

2.3.8 ミンコフスキー距離

ユークリッド距離を一般化したもので、非常に離れた距離の重みを増やしたり減らしたりできる。

$$d(x, y) = \sqrt[b]{\sum_{i=1}^n |x_i - y_i|^a} \quad (2.30)$$

$a = b = 1$ のときマンハッタン距離に、 $a = b = 2$ のときユークリッド距離に、そして $a = b = \infty$ のときチェビシェフ距離にそれぞれ一致する。

2.3.9 コサイン類似度（コサイン距離）

類似度を計算する対象を n 次元のベクトル $x = (x_1, \dots, x_n)$, $y = (y_1, \dots, y_n)$ としたとき、

$$s_{\cos}(x, y) = \frac{\sum x_i y_i}{\sqrt{\sum x_i^2 \sum y_i^2}} \quad (2.31)$$

で定義したものがコサイン類似度である。

ベクトル空間モデルにおいて、文書同士を比較する際などによく用いられる類似度計算手法である[18]。コサイン類似度はベクトル同士のなす角度の近さを表現するため、その名の通り三角関数のコサインのように、データが非負値の場合は 0 から 1 の範囲、負を含む場合は -1 から 1 の範囲になる。ベクトルの向きが一致しているとき最大値の 1 をとり、直交なら 0、向きが逆ならば最小値の -1 をとる。

1 からコサイン類似度を引くことによりコサイン距離として定義することもできる。コサイン距離は他の距離尺度と同じように、値が小さいほど類似していて、値が大きいほどそうでないという扱い方が可能となる。

2.3.10 相関係数（相関に基づく距離）

コサイン類似度と似ている指標として、ピアソンの相関係数も有名であり、類似度の尺度として使うことができる。

共分散の項でも触れているが、相関係数はデータ x と y の共分散を x と y それぞれの標準偏差で割って正規化したものである。類似度を計算する対象を n 次元のベクトル $x = (x_1, \dots, x_n)$, $y = (y_1, \dots, y_n)$ とし、ベクトル x, y の次元要素の平均をそれぞれ \bar{x}, \bar{y} としたとき、

$$\text{cor}(x, y) = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

(2.32)

で定義される。コサイン距離と同様に、データが非負値の場合は 0 から 1 の範囲、負を含む場合は -1 から 1 の範囲になる。ベクトルの向きが一致しているとき最大値の 1 をとり、直交なら 0、向きが逆ならば最小値の -1 をとる。

相関係数は外れ値の影響を大きく受けるので注意が必要である。また、相関に基づく距離を使用する場合は、コサイン距離同様 1 から相関係数を引いて定義する。

2.4 k 最近傍法

2.4.1 概要

テンプレート (template・鋳型) と呼ばれる学習データすべてと、認識対象となる入力データとの距離を計算する。そして最も近い、つまり最も距離が小さいテンプレートが所属するクラスに入力データも所属するとして識別する方法を最近傍 (nearest neighbor/ NN) 法という。最近傍法は、テンプレートが多ければ多いほど非常に高精度の認識を行うことができる。ただし、トレードオフとして計算時間がかかる。最近傍点だけではなく、k 個の最も近いテンプレートを選び、それらのテンプレートが最も多く属しているクラスに識別する方法を、k 最近傍 (kNN) 法という。kNN 法はパターン認識の分野において、広く用いられている。

2.4.2 最近傍法

K 個のクラスを $\Omega = \{C_1, \dots, C_K\}$ 、i 番目のクラスの学習データ数を $N(i)$ 、その集合を $S_i = \{x_1^{(i)}, \dots, x_{N(i)}^{(i)}\}$ とする。最近傍 (NN) 法では学習データのことをテンプレートとも呼ぶが、これは入力データ x とその学習データ $x_j^{(i)}$ の類似度をユークリッド距離 $d(x, x_j^{(i)}) = \|x - x_j^{(i)}\|$ などの距離尺度で計算するからである。識別規則は、

$$\text{識別クラス} = \begin{cases} \arg \min_i d(x, x_j^{(i)}) & \min_{i,j} d(x, x_j^{(i)}) < t \text{ のとき} \\ \text{リジェクト} & \min_{i,j} d(x, x_j^{(i)}) \geq t \text{ のとき} \end{cases} \quad (2.33)$$

とする。t は、どの学習データとも距離が大きい場合において、リジェクトするために必要となる閾値である。

最近傍法による認識率は、学習データ数 (テンプレートの数 M) が多ければ多いほど良くなる (ただし計算時間はかかる)。

2.4.3 ボロノイ図

入力データに最も近い、つまり最も距離が小さいテンプレートを見つけることが最近傍法の原理である。これについて、逆の視点でテンプレート側から見るとする。Fig2.15 のよ

うに各テンプレートは支配領域（隣接テンプレートと等距離にある境界で囲まれている）をもち、入力データが入った支配領域に対応するテンプレートが最も近いテンプレートということになる。この支配領域をボロノイ領域、その境界をボロノイ境界という。これらは以下のように定義される。

テンプレートの集合を $S = \{x_1, \dots, x_N\} (N \geq 3)$ とする。ボロノイ境界は、 $x_i, x_j \in S$ から等距離の点の集合

$$B(x_i, x_j) = \{x | d(x_i, x) = d(x_j, x)\} \quad (2.34)$$

で定義される。これは、Fig2.16 に示すように、 x_i と x_j を結んだ直線（法線ベクトル n 方向）の中心（平均ベクトル \bar{x} ）を通り、直交する超平面

$$(\bar{x} - x)^T n = 0 \quad (2.35)$$

となる。 $\bar{x} = \frac{x_i + x_j}{2}, n = x_i - x_j$ である。

この超平面は、 d 次元空間を、 x_i を含む半空間

$$D(x_i, x_j) = \{x | d(x_i, x) < d(x_j, x)\} \quad (2.36)$$

と、 x_i を含む半空間 $D(x_j, x_i)$ の 2 つに分割する。 x_i のボロノイ領域は、 x_i を含む半空間の積集合

$$VR(x_i, S) = \bigcap_{x_j \in S, j \neq i} D(x_i, x_j) \quad (2.37)$$

で定義される。定義から $VR(x_i, S)$ は開集合である。このときボロノイ境界を含めた閉包を $\overline{VR(x_i, S)}$ で表す。テンプレート集合 S のボロノイ図（ボロノイモザイクとも呼ばれる）は、

$$V(S) = \bigcup_{x_j \in S, j \neq i} \overline{VR(x_i, S)} \cap \overline{VR(x_j, S)} \quad (2.38)$$

で定義される。

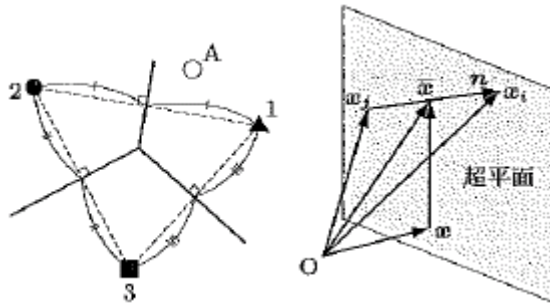


Fig2.15 テンプレートの支配領域^[13] Fig2.16 ボロノイ境界を形作る超平面^[13]

2.4.4 kNN 法

k 個の最も近い、つまり最も距離が小さいテンプレートを選び、それらのテンプレートが最も多く属しているクラスに識別する方法を k 最近傍 (kNN) 法という。クラス選択は投票の形で決定されるため、投票型 kNN 法と呼ばれることもある。

テンプレートの集合を $T_N = \{x_1, \dots, x_N\}$ 、それらが属するクラスの集合を $\Omega = \{C_1, \dots, C_K\}$ 、 i 番目のテンプレートが属するクラスを $\omega_i \in \Omega$ とする。入力 x に最も近い、つまり最も距離が小さい k 個のテンプレートの集合を $k(x) = \{x_{i_1}, \dots, x_{i_k}\}$ とし、これらのテンプレートのうちクラス j に属するテンプレート数を k_j とする。 $k = k_1 + \dots + k_K$ が成り立っている。kNN 法の識別規則は、

$$\text{識別クラス} = \begin{cases} j & \{k_j\} = \max\{k_1, \dots, k_K\} \text{ のとき} \\ \text{リジェクト} & \{k_i, \dots, k_j\} = \max\{k_1, \dots, k_K\} \text{ のとき} \end{cases} \quad (2.39)$$

となる。上記の識別規則では、得票数が同数だった場合はリジェクトとしているが、いずれかのクラスを無作為に識別クラスに決定するようにしても良い。

2.4.4 kNN 法の計算量とその低減法

クラスに番号を $i = 1, \dots, K$ と割り振る。各クラスのテンプレート数をすべて同数とし、それに番号を $j = 1, \dots, M$ と割り振る。またデータの次元数を d とする。kNN 法では入力データが与えられた時、全クラスの全テンプレート $x_j^{(i)}$ と距離を、例えばユークリッド距離の 2 乗を使用する場合、

$$d^2(x, x_j^{(i)}) = (x - x_j^{(i)})^T (x - x_j^{(i)}) \quad (2.40)$$

を計算する必要がある。ユークリッド距離の 2 乗の場合、この距離計算に、ベクトルの差を求めるための KM 回の減算、および積和が必要となる。さらに、距離を昇順にソートする場合でも、ソートせずに距離の小さいテンプレートを検索する場合でも、最低でもおおよそ $KM \log(KM)$ のオーダーの比較、置換が必要となる。

このように、kNN 法はデータの次元が大きくなればなるほど、多くの時間、そして多くの記憶容量が必要であり、実時間で認識を行うには向いてない手法であるといえる。しかし、その制約を緩和しようとする試みが行われてきている。その一例として挙げられるのが、誤り削除型 kNN や圧縮型 kNN、分枝限定法、近似最近傍探索などである。

2.5 深層学習(Deep Learning)

2.5.1 深層学習の概要

深層学習(Deep Learning)とは、深い、すなわち多くの層を持ったニューラルネットワー

クモデルを用いた機械学習の総称である[19]。深層学習では、深い＝層の数が多いニューラルネットワークによって、観測データから本質的な情報を抽出した内部表現(internal representation) (潜在表現(latent representation)や特徴(feature)と呼ぶこともある)を学習する。

機械学習の研究は 1950 年代末期から人工知能の一分野として発展してきた。人間や生物の脳神経系は強力な学習能力を持つことが知られていることから、高度な情報処理の実現を目指して、生物の神経回路網を模倣した人工ニューラルネットワークが長年にわたって研究されてきた[20]。そして、その研究の流れを受けて、近年、大量の電子的データと、強力な分散並列計算を基盤とした深層学習が、音声認識や一般物体認識などのタスクで高い性能を示したことなどを背景として注目され、盛んに研究されている。

ニューラルネットワークモデルにはいろいろな種類があるため、それに対応して、深層学習の方式も様々なものが提案されている。この章では、提案手法の説明に必要な知識に絞って説明していく。

2.5.2 畳み込みニューラルネットワーク

畳み込みニューラルネットワーク(Convolutional Neural Network:CNN)は畳み込みとプーリングを繰り返し、高次の特徴を得る多層ニューラルネットワークである。全結合層だけで構成される多層ニューラルネットワークとは異なり、畳み込み層やプーリング層を持つのが特徴である。各ユニットで、入力と重みの線形和をシグモイド関数などで活性化するという基本動作は変わらない。

全結合層の代わりに畳み込み層とプーリング層を利用することで、学習すべきパラメータ数が減る上、入力画像のある位置におけるエッジの傾きや終点、コーナーなどといった視覚的な特徴をうまく抽出できる。また、畳み込み層における各位置でのカーネルの計算やプーリング層におけるプーリング処理は独立しており並列処理が可能なことから、GPUを用いての計算に適している。

2.5.3 畳み込み層

畳み込み層では様々なカーネルを用いて Fig2.17 に示すような畳み込み処理を行う。画像処理における畳み込み処理とは、注目画素と周辺画素の値にそれぞれ重みを付け、それらの和を出力画像の画素値とするというものである。ここにおけるカーネルとは、一般的に $n \times n$ で重みパラメータを保持する積分核のことで、どのように畳み込むかを示す。

注目する画素を一定間隔(stride)でずらしながら、入力全体にカーネルを適用していく。畳み込みそうでは一般的に様々なバリエーションのカーネルがあり、カーネルの数だけ特徴マップと呼ばれるものを出力する。

$n_i \times n_i$ pixel の画像を、 $n_k \times n_k$ pixel のカーネルを使って畳み込むとする。出力される特徴マップの一边のサイズ n_0 は以下ようになる。

$$n_0 = \frac{n_i - \frac{n_k - 1}{2} \times 2}{stride} = \frac{n_i - n_k + 1}{stride} \quad (2.41)$$

注目画素が必要な周辺画素を持つ事を考慮して、入力画像の内側に $(n_k - 1)/2$ 分の余白を設定している。基本的に n_0 が整数になるように、入力画像サイズ n_i とカーネルのサイズ n_k 、stride のパラメータは決定される。

畳み込み層は、ある一つのカーネルとそれによって得られる特徴マップに注目すると、Fig2.18 のようなニューラルネットワークとして表すことができる。Fig2.18 では 3 種類の矢印でユニット間を接続しているが、これは同じ種類の矢印は同じ重みを持つことを示している。このように、一つのカーネル内で 1 セットの重みを共有しているがゆえに、全結合層よりも扱う重みの数が少なくなる。これはつまり、学習すべきパラメータ数と学習にかかる時間が全結合層に比べて少なくて済むということを意味している。

畳み込み層のパラメータの学習は全結合層のパラメータと同じく誤差逆伝播法を用いる。誤差逆伝播法に関しては後述する。

一般的なニューラルネットワークと同じように、畳み込み処理の出力を保持する各ユニットにおいて活性化を行う。

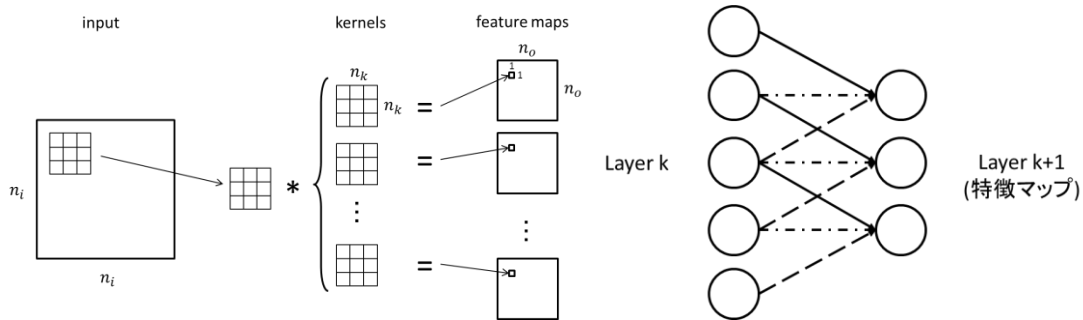


Fig2.17 畳み込み層における処理の概要 Fig2.18 畳み込みニューラルネットワーク

2.5.4 プーリング層

CNNにおいて、畳み込み層で出力された特徴マップはプーリング処理されることが多い。プーリング処理には、データサイズを減らし、対象領域の些細な幾何情報の違いを吸収し、その領域内の特徴をロバストに取得する効果がある。

プーリング処理は入力画像の注目領域における平均値や最大値などを、出力される特徴マップの画素値とするというものである。サブサンプリング処理とも呼ばれる。

4×4 の入力に対して 2×2 の領域ごとに行うプーリング処理を Fig2.19 に示す。この結果、出力として 2×2 の特徴マップを得ている。この場合、 2×2 の領域ごとに処理を行うことによって、データ量が $4 \times 4 = 16$ から $2 \times 2 = 4$ と $1/4$ になっていることがわかる。

CNN では、対象領域の平均値を取る Avg-pooling と最大値を取る Max-pooling がよく利用される。

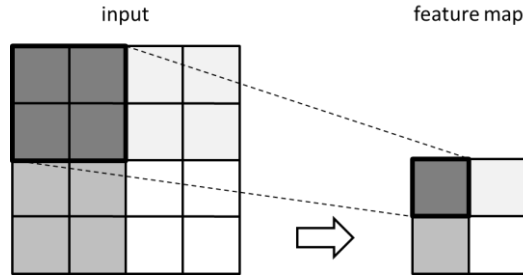


Fig2.19 プーリング処理の概要[21]

2.5.5 全結合層

全結合層は一般的な多層ニューラルネットワークでよく見られる、前層の全ユニットとその層の各ユニット同士がすべて繋がっている層である。j 番目の入力を x_j 、i 番目の出力を y_i 、 x_j と y_i のユニット間の重み係数を w_{ij} 、バイアスを b_i とすると、

$$y_i = f\left(\sum_j w_{ij}x_j + b_i\right) \quad (2.42)$$

という処理になる。この式における f は活性化関数である。

2.5.6 活性化関数

CNN で利用される活性化関数には主にシグモイド関数や ReLU などがある。この項ではそれら活性化関数について説明する。

2.5.6.1 シグモイド関数

シグモイド関数は以下の式で示す非線形関数である。

$$f(x) = \text{sigmoid}(x) = \frac{1}{1 + e^{-x}} \quad (2.43)$$

シグモイド関数は微分しても、以下のようにシグモイド関数で表すことができるという特徴を持っている。

$$f'(x) = \text{sigmoid}(x)(1 - \text{sigmoid}(x)) \quad (2.44)$$

この特徴が後述する誤差逆伝播法を用いて多層ニューラルネットワークのモデルパラメータを最適化する際に意味を持つ。

2.5.6.2 ReLU

ReLU(Rectified Linear Unit)は以下の式に示す区分線形関数である。

$$f(x) = \max(0, x) \quad (2.45)$$

シグモイド関数や \tanh 関数は x の値が大きければ飽和する性質を持つのに対し、ReLU は非飽和な性質を持つ。この ReLU の非飽和な性質は、飽和な性質の活性化を行う場合よりも、勾配法におけるモデルの学習を高速にする。

2.5.6.3 ソフトマックス関数

ソフトマックス関数は以下の式に示す関数である。

$$f_i(a) = \frac{e^{a_i}}{\sum_j^n e^{a_j}} \text{ for } i = 1, \dots, n \quad (2.46)$$

ソフトマックス関数はシグモイド関数を多変量に適応させた関数で、 $f_i(a)$ は $(0,1)$ の範囲の値をとる。 $\sum_{i=1}^n f_i(a) = 1$ であるため、多変量ロジスティック回帰や他クラス分類などにおける離散確率分布としても扱うことができる。ニューラルネットワークにおいては、ある入力 that 各クラスに属する確率を算出するために、しばしば最終層で利用され、これは CNN においても例外ではない。

シグモイド関数は微分してもシグモイド関数で表すことができるという特徴を持っていた。ソフトマックス関数の a_j に関する偏微分も以下のようにソフトマックス関数で表すことができる。

$$\frac{\partial f_i}{\partial a_j} = f_i(\delta_{ij} - f_j) \quad (2.47)$$

δ_{ij} はクロネッカーのデルタで、 i と j が等しい時は 1、それ以外の時は 0 となる。

2.5.6 誤差逆伝播法

誤差逆伝播法は、ニューラルネットワークにデータを入力し、期待する出力の値と実際に出力された値から損失を求め、その損失が小さくなるように書くユニットの重みを更新していく手法である。損失(loss)とはモデルの精度の悪さを表し、損失関数から求められる。

モデルの精度を上げるために求めたいものは、損失関数の最小値をとるようなパラメータである。しかし、ニューラルネットワークにおいて損失関数が依存するパラメータ数はとても大きい。損失関数のパラメータとして各ユニットの重みなどが全て含まれるからである。そのため、損失関数の最小値をとるパラメータが明示的にわからない。そこで、最急降下法を応用して損失が小さくなるように各ユニットの重みを更新していく。

Fig2.20 のような単純なニューラルネットワークについて考える。このニューラルネットワークの出力 $h(x)$ は以下のように表すことができる。

$$h(x) = f(g(x)) = (f \circ g)(x) \quad (2.48)$$

$$o = f(u) = \text{sigmoid}(u) = \frac{1}{1 + e^{-u}} \quad (2.49)$$

$$u = g(x) = W^T x + b \quad (2.50)$$

f は活性化関数、 g は線形和を求める関数であり、今回は活性化関数 f をシグモイド関数としている。

ニューラルネットワークのユニットは一般的に Fig2.20 の(a)のような線形和を求める処理と活性化を行う処理を含んでいるが、今回は説明のため、それらの処理を(b)のように分解している。

損失関数を以下のような二乗和誤差関数とし、入力 $x = (x_1, \dots, x_n)$ に対し、教師信号 t が与えられている時、損失は以下のように算出される。

$$E = \frac{1}{2} \|t - h(x)\|^2 \quad (2.51)$$

ここで、 E を活性化後の値 o で偏微分した値は以下のようになり、誤差信号 δ^o と定義する。

$$\delta^o \equiv \frac{\partial E}{\partial o} = -(t - o) \quad (2.52)$$

今回は単純なニューラルネットワークであるため、活性化関数の出力 o がそのまま $h(x)$ になっていることに注意する。また、 E を線形和の値 u で偏微分した値は以下のようになり、誤差信号 δ^u と定義する。

$$\delta^u \equiv \frac{\delta E}{\delta u} = \frac{\partial E}{\partial o} \frac{\partial o}{\partial u} = \delta^o o(1 - o) \quad (2.53)$$

以上より、重み W とバイアス b の修正量は合成関数の微分における連鎖律を用いて以下のように求めることができる。

$$\Delta W = \frac{\partial E}{\partial W} = \frac{\partial E}{\partial u} \frac{\partial u}{\partial W} = \delta^u x \quad (2.54)$$

$$\Delta b = \frac{\delta E}{\delta b} = \frac{\partial E}{\partial u} \frac{\partial u}{\partial b} = \delta^u \quad (2.55)$$

W と b は以下の式のように最急降下法を適用して更新する。

$$W_{new} = W_{old} - \epsilon \Delta W$$

(2.56)

$$b_{new} = b_{old} - \epsilon \Delta b$$

(2.57)

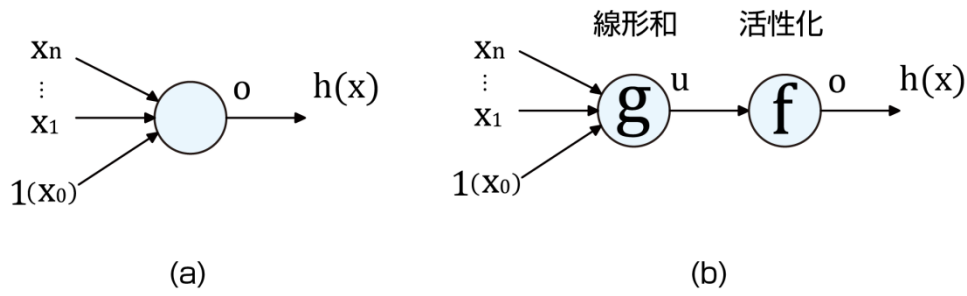


Fig2.20 標準的なニューラルネットワークのユニットとその処理を分解したもの[21]

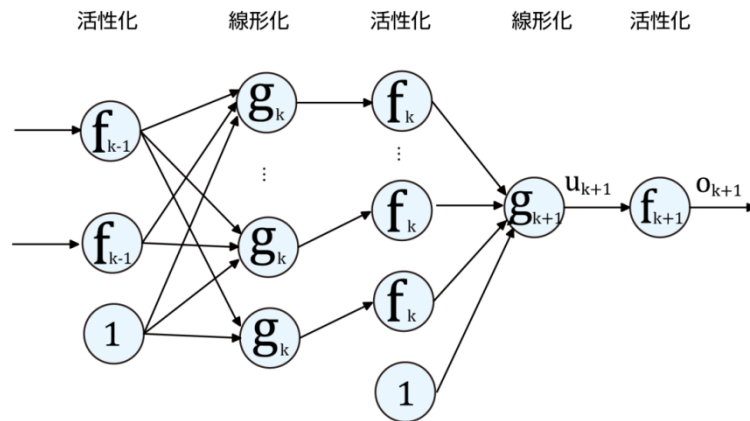


Fig2.21 多層ニューラルネットワークの一部[21]

ここまでは単純なニューラルネットワークを前提に重み \mathbf{W} とバイアス \mathbf{b} を更新することを考えていたが、ここからは Fig2.21 のような多層のニューラルネットワークについて考える。

このとき、この多層ニューラルネットワークの入力と出力は以下ようになる。

$$o^1 = x$$

(2.58)

$$o^{k_{max}} = h(x)$$

(2.59)

便宜上、ここでもすべての活性化関数をシグモイド関数とすると、 o を u で偏微分した値は以下のように表される。

$$\frac{\partial o^{k+1}}{\partial u^{k+1}} = o^{k+1}(1 - o^{k+1}) \quad (2.60)$$

その他、 u を各変数で偏微分した値を以下に示す。

$$\frac{\partial u^{k+1}}{\partial o^k} = W^k \quad (2.61)$$

$$\frac{\partial u^{k+1}}{\partial W^k} = o^k \quad (2.62)$$

$$\frac{\partial u^{k+1}}{\partial b^k} = 1 \quad (2.63)$$

以上より、活性化処理を行う層における誤差信号 δ^{o^k} は以下のように表すことが可能となる。

$$\delta^{o^k} = \frac{\partial E}{\partial o^k} = \begin{cases} -(t - o^k) & \text{if } k = k_{max} \\ \frac{\partial E}{\partial u^{k+1}} \frac{\partial u^{k+1}}{\partial o^k} = \delta^{u^{k+1}} W^k & \text{otherwise} \end{cases} \quad (2.64)$$

このとき、損失関数は先ほどと同じように二乗和誤差関数としている。

したがって、線形和層における重み W とバイアス b の修正量は、先ほどと同じく、連鎖律を用いて以下のように求めることができる。

$$\frac{\partial E}{\partial W^k} = \frac{\partial E}{\partial u^{k+1}} \frac{\partial u^{k+1}}{\partial W^k} = \delta^{u^{k+1}} o^k \quad (2.65)$$

$$\frac{\partial E}{\partial b^k} = \frac{\partial E}{\partial u^{k+1}} \frac{\partial u^{k+1}}{\partial b^k} = \delta^{u^{k+1}} \quad (2.66)$$

以上のように、出力層の方から順番に 1 つ前の層のパラメータを更新していくアルゴリズムが誤差逆伝播法である。

重みの初期値は毎回乱数で初期化する。そのため、同じ設定かつ同じデータを用いたとしても、学習するたびに異なるモデルとなる。

ここでは全結合層を例に誤差逆伝播法について説明したが、畳み込み層を用いても変わらない。畳み込み層の場合、ユニット間の接続がない場所の重みは常に 0 と考えて誤差逆伝播法を適用すれば良い。

プーリング層においては重みパラメータを持たないので更新するものは存在しない。

第3章 従来手法

3.1 特徴量ベクトルを入力とした SOM による教師なしクラスタリング手法

3.1.1 概要

打楽器音の音源同定の従来手法は大きく分けて 2 つ存在するが、そのうちの一つが特徴量ベクトルを使った方法である。ここではその代表的な手法^[4]の概要を述べる。

この手法では、特に膜鳴楽器（膜の振動を胴などに共鳴させるもので、主にさまざまな太鼓類がこれに相当する）の音源同定手法について検討する。まず、同一曲内では各打楽器の特徴量は定常的であると仮定する。抽出した特徴量に対し、教師なしクラスタリングを利用する。教師なし識別方法であるために、学習データは必要ない。また、個体差に富む楽器でも、同一曲内では個体差を考慮しなくてよい。よって、打楽器の個体差による音の違いの問題と、学習データ不足の問題は、この手法においては気にする必要がなくなる。

また、同定対象（楽曲）のクラス数（＝楽器の種類数）を既知とした場合のクラスタリング手法が有効に働くことは既に報告されているが、実際にはクラス数が既知である場合は少ない。そこで、教師なしクラスタリングの実現のために、自己組織化マップ(SOM : Self-Organizing Map)を用いている。

また、すべての音響信号には事前にローパスフィルタ処理を行うことで、周波数帯域を分離し、スペクトルの重なりの問題に対処している。

打楽器の音源同定処理は、膜鳴楽器識別と体鳴楽器（カスタネットやトライアングル、シンバルなど塊や棒状の発音源自体が鳴り響くもの）識別で別々に行う。本手法の処理の流れ図を Fig.3.1 に示す。

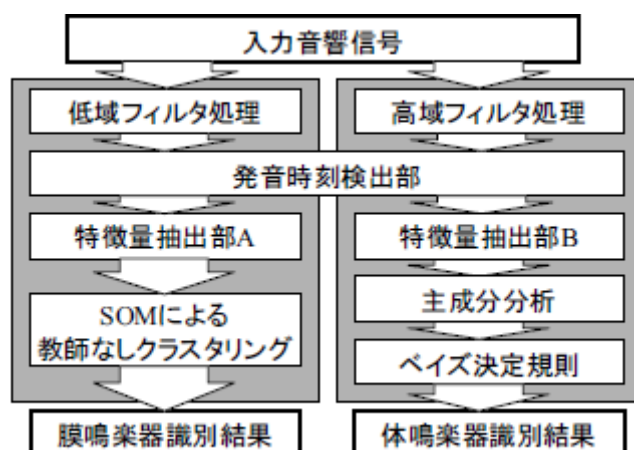


Fig.3.1 音源同定処理の流れ図

膜鳴楽器の認識では、入力音響信号に低域通過フィルタを適用した後、発音時刻検出、特徴量抽出、SOM を利用した教師なしクラスタリングの順に処理を行い、いくつかの候補を出力する。体鳴楽器の認識では、入力音響信号に高域通過フィルタを適用した後、発音時刻検出、特徴量抽出、主成分分析とベイズ決定規則による識別の順に処理を行い、識別結果を出力する。

3.1.2 問題点

まずこの手法では、入力音響信号はドラムセットのみによるドラム演奏であるとされている。そのため、実際の楽曲では当然含まれている調波構造を持つ楽器は考慮されていない。体鳴楽器識別には教師付き統計的識別法を採用しているが、残響の影響や、膜鳴楽器のスペクトルの重なりなどで、抽出する特徴量ベクトル（Table.3.1 に使用する 43 個の特徴量を示す）が変形する問題があるとされている。そのうえ入力音響信号にその他の楽器が含まれれば、これらの影響は更に大きくなると考えられる。この問題は教師あり・教師なしにかかわらず、特徴量ベクトルを抽出する手法全般にいえることである。そして混合音に対する音源同定を単音に比べて桁違いに難しくしている、一番大きな原因であるといえる。

また、教師なしクラスタリング特有の問題としては、未知楽器が含まれている場合は正しいラベル付けが行えないことが挙げられる。この手法では教師なしクラスタリング後に事前知識を用いたラベル付けを行うが、事前知識に含まれているクラス間の関係性は既知楽器の範囲でしかない。そのため未知楽器クラスが 1 つでも存在するとラベルの順序がずれるなど、多くのクラスに影響を及ぼしかねない。

これらのことから、この手法は同定対象が既知の打楽器のみ、かつ同時発音が許されるのは膜鳴楽器 1 つと体鳴楽器 1 つのみという条件下では高性能であるものの、これでは適用できる音響信号が限定的すぎるといえる。

Table.3.1 体鳴楽器の音色を表す 43 個の特徴量

(1)	スペクトルの定常的特徴 (3 個: FT1 - FT3) 周波数重心, 最大パワー周波数, 最大から 5 番目までの パワーを持つ周波数の時間方向の平均
(2)	2-4 次モーメントに関する特徴 (3 個: FT4 - FT6) パワーの分散, 歪度, 尖度の時間方向の平均
(3)	アタック (atk) 区間に関する特徴 (8 個: FT7 - FT14) atk エネルギー, atk 時間, ログ atk 時間, ゼロクロス割合, atk エネルギー/atk 時間, atk エネルギー/最大パワー*atk 時間, atk 時間重心, atk 時間重心/atk 時間
(4)	ディケイ (dec) 区間に関する特徴 (6 個: FT15 - FT20) ゼロクロスの割合, 周波数重心の時間方向の平均値と分散, パワーの分散, 歪度, 尖度の時間方向の平均
(5)	パワー分布に関する特徴 (7 個: FT21 - FT27) 5k-7k, 7k-10k, 10-13k, 13k-16k, 16k-20k [Hz] 各セクション のパワーが占める割合, 5%, 20%以上のパワーの周波数の 占める割合
(6)	残響成分に関する特徴 (2 個: FT28 - FT29) 最大パワーフレーム後 Y(ms) 後までの残響度合い (Y = 100, 200) (エンベロープの面積 / 最大パワー * Y ms)
(7)	MFCC に関する特徴 (13 個: FT30 - FT43) 13 次元+E の MFCC の分散の時間平均

アタックとは最大パワーフレームまでの区間,
ディケイとはそれ以降の区間を指す。

3.2 テンプレート適応を利用したテンプレートマッチング手法

3.2.1 概要

打楽器の音源同定の従来手法を大きく 2 つに分けたうち、もう一つの手法が打楽器単音のパワー分布をテンプレートとしたテンプレートマッチングによる手法である。ここではまず、そのうちの代表的な手法[5]の概要を述べる。

この手法では、それぞれの打楽器ごとに基本テンプレートモデルを 1 つずつだけ必要とするテンプレート適応手法を用いる。まず基本テンプレートモデルを 1 つ与えることで、対象曲中から対応するドラムが発音していると推定される場所を複数探索する。次に探索結果の周辺のパワー分布を手がかりとして、テンプレートモデルの更新を行う処理を繰り返すことで、テンプレート適応を実現している。そして適応後のテンプレートモデルを用いて、距離尺度を改良したテンプレートマッチングにより音源を同定する。

3.2.2 処理の流れ

処理は大きく分けてテンプレート適応部とテンプレートマッチング部の 2 つの部分から構成されている。

テンプレート適応部の処理の流れ図を Fig.3.2 に示す。

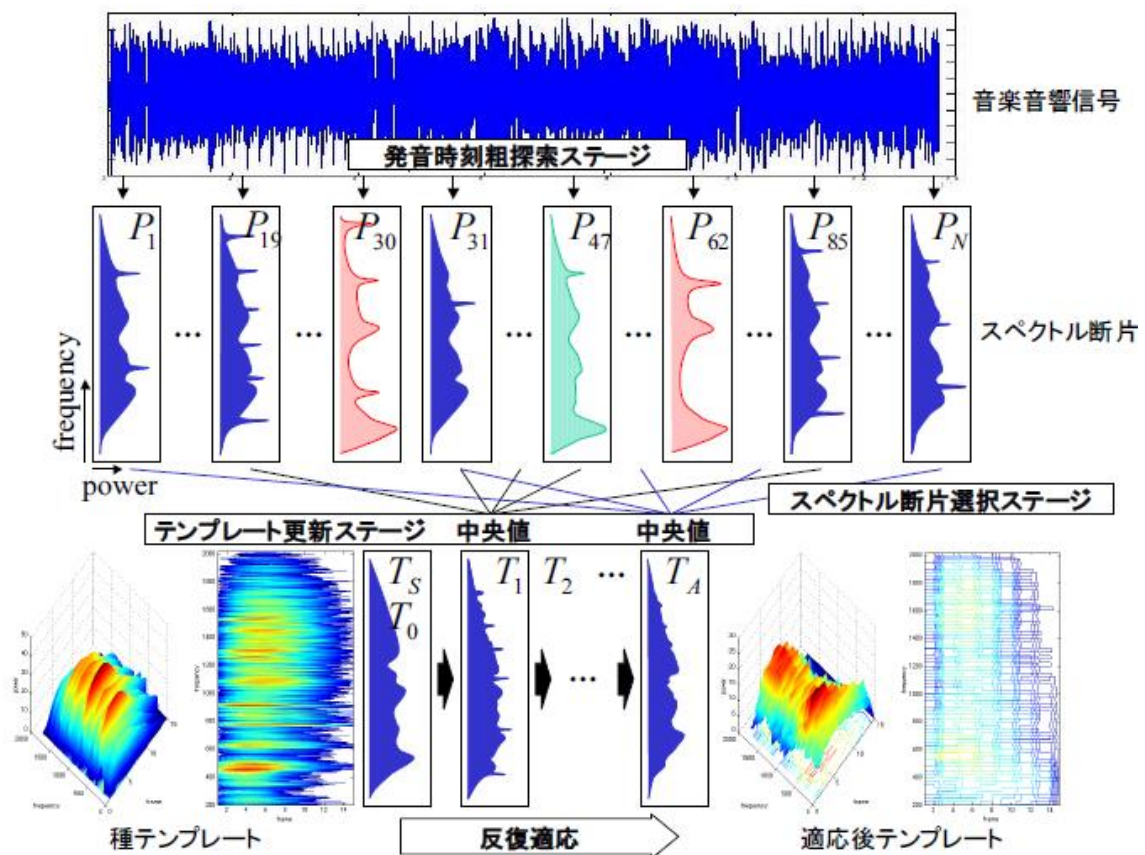


Fig.3.2 テンプレート適応部の処理の流れ図

テンプレート適応部では、ドラムセットを構成するそれぞれの打楽器ごとに用意された基本テンプレートモデル（種モデル）を、対象曲に使用されているドラム音をうまく表現するモデルへと適応させる。1つの種モデルを初期モデルとして与えれば、適応を繰り返すことで、より良いモデルに自動的に成長させることが可能である。

適応過程は、パターン選択（下記(2)(3)）とモデル更新（下記(4)）の2つのステージから構成され、これらが交互に繰り返される。以下に、テンプレート適応部での各処理について簡単に述べる。

- (1) 種モデルを作るために、ドラム単音が収録されたファイルから発音時刻を検出し、そこから一定時間長のパワー分布を切り出す。
- (2) 対象曲から発音時刻を粗探索して、複数の発音時刻候補をあらかじめ求めておく。各発音時刻候補を補正し、補正後の発音時刻候補からモデルと同じ時間長のパワー分布を切り出して、各発音時刻候補から抽出したパターンとする。
- (3) 後述する距離尺度に従って、モデルと各パターンとの距離を計算し、距離の近いものから複数個のパターンを選択する。
- (4) 選択されたパターンの重み付き平均を計算し、次回の適応ループにおけるモデルとする。

テンプレートマッチング部の処理の流れ図を Fig.3.3 に示す。

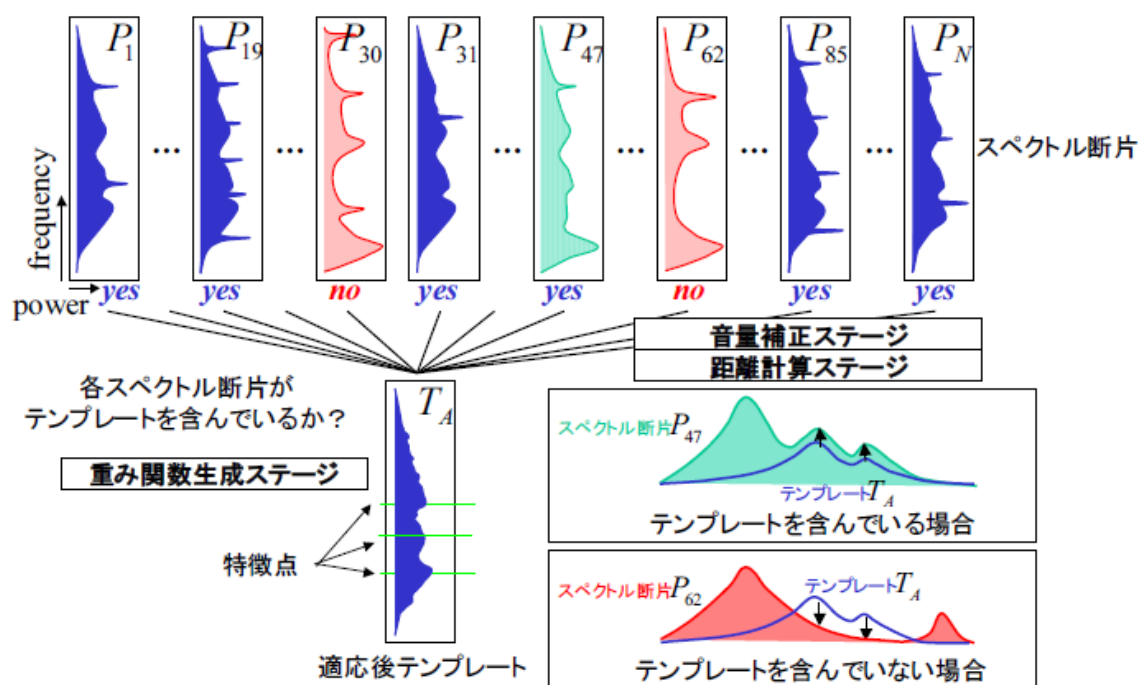


Fig.3.2 テンプレートマッチング部の処理の流れ図

テンプレートマッチング部では、適応後のモデルと各パターンとの距離を計算し、閾値処理を行うことで、そのパターンにモデルが含まれているかを判定する。このとき、識別したいドラム以外の音が多数含まれている場合でも距離が正しく計算されるように、新しい距離尺度を提案する。この距離尺度は、時間一周波数領域において、モデルの特徴的な点の周辺に着目して距離を計算するので、他の楽器の周波数成分が混じっていたり、周波数成分の時間的なゆれがあっても、正しく計算することを狙っている。

3.2.3 問題点

この手法では、調波構造を持つ音を含んだ混合音を実験の対象としているため、実世界の音響音楽信号に対して適用可能な音源同定だといえる。しかし、同定対象とされている打楽器はドラムやシンバルといったドラムセットの構成楽器のみであり、他の打楽器については扱っていない。また、マッチング対象の楽曲にも、含まれる打楽器は先述の楽器のみという条件が存在するので、実世界の楽曲はなんでも扱えるとはいいがたい。

1つや2つ、対象の楽器を増やす程度なら既存の手法のままでも問題がないと思われるが、大幅に増やそうとする場合、この手法が抱える固有の問題が障壁になると考えられる。

この手法ではテンプレート適応が必須の手順となっているが、テンプレート適応は楽曲中に含まれる打楽器音を利用して行っている。つまり、楽曲中に含まれる打楽器についての事前知識が存在することが前提の手法である。もし事前知識なしで使用する場合は、楽曲に使われる頻度の高い楽器に限定する必要がある。そのためドラムセットに限定されていると思われる。

しかし事前知識を用いる場合、ドラムセットに限定する場合、どちらにしてもある程度の制約条件となり、この手法が万能ではない要因となっている。

3.2.4 提案手法に向けて

パワー分布をテンプレートとして扱う手法自体は汎用性が高く、他の打楽器にも同様に適用できる可能性がかなり高い。テンプレートマッチングも認識処理の基本であるので、提案手法としてはこちらの手法をベースにし、障壁の原因となっているテンプレート適応の代わりとしていかに改良していくかがポイントとなる。

第 4 章 k 最近傍法を用いた提案手法

4.1 概要

これまで紹介した従来手法は、いずれも音楽演奏に使われている楽器の種類を特定することもだが、その楽器がどのタイミングで発音しているかということをより重視していた。これは音源同定の 1 つの目標が、「楽曲中で発音された打楽器音 1 つ 1 つをすべて特定する」ということだからである。これを達成しようとする場合、必然的に楽器が発音するタイミングを知る必要がある。従来手法ではこれを達成するためにどうしても特定する楽器数を絞る必要があった。前述したとおり、ポピュラー音楽をマッチング対象とした場合、楽曲中で発音される打楽器音はほとんどがドラムセットであり、これらの打楽器に絞ることで高い同定精度を実現している。

しかし、楽曲に使われている打楽器はもちろんドラムセットだけではなく、ポピュラー音楽にもドラムセット以外の打楽器が用いられていることは多い。楽曲に使われている打楽器名を知りたい場合、これらを除いて考えるというわけにはいかない。

そこで、提案手法では楽曲中で使われている打楽器名を特定するという目的だけに絞り、楽曲中で発音された打楽器音 1 つ 1 つをすべて特定することはしない代わりに、対象とする楽器を打楽器全般に拡張することにする。

具体的な方法としては、まず打楽器単音の音楽ファイルからパワー分布のテンプレートを作成する部分までを 2 つ目の従来手法を踏襲して行う。ただし、作成するテンプレートの種類については、マッチング対象の打楽器を増やすためにデータベースに存在するあらゆる打楽器から作成する。また同じ種類の打楽器でもそれぞれ複数の個体・奏法から作成したテンプレートを用意することで、楽器の個体差によるマッチングへの影響を最小限に抑える。これはテンプレート適応の果たしていた役割の代わりとなる。

テンプレート作成後は、モデル更新を行わずにそのままマッチング部に移行する。k 最近傍法を用いることで、ある打楽器の複数の個体からそれぞれ作成したテンプレートすべて、あるいはある打楽器カテゴリに属する楽器からそれぞれ作成したテンプレートすべて、といった形で包括的に対象楽曲とのテンプレートマッチングを行うことが可能となる。

対象の楽曲からも発音時刻候補ごとにテンプレート側と同じようにパワー分布を切り出す。そしてその発音時刻候補でのパワー分布と各テンプレートとの類似度を計算し、閾値処理によりテンプレートの打楽器が含まれているかどうかを判定する。

場合によっては、認識率を向上させるため、テンプレートとなる打楽器単音、対象楽曲それぞれの音楽信号に何らかの前処理を施すこともある。

4.2 処理の流れ

4.2.1 テンプレート側

まず、テンプレートに使用する打楽器の単音が含まれたサウンドファイルを用意する。この際、なるべく多くの打楽器をマッチング対象にするため、非調波構造という条件さえ満たしていれば、種類にかかわらずできる限り多くのファイルを用意することが望ましい。また、同じ楽器の種類でも、異なる個体・奏法でそれぞれファイルを別々に用意する。これらはいずれのテンプレートにマッチングしても出力時は同じ楽器のものとして扱う。これによりテンプレートと楽曲にそれぞれ使われている同じ種類の楽器の音色に差異があっても、いずれかのテンプレートとマッチングする可能性が高まると期待できる。マッチングしなかったとしても、楽器カテゴリを出力するにあたってカテゴリを推定するための有用な情報を増やすことにつながる。

ここからはそれぞれのサウンドファイルに対し共通の処理を施してテンプレートを作成していく。処理の詳細なアルゴリズムは以降の項で記述する。

用意したサウンドファイルを読み込み、最初に打楽器単音の発音時刻の検出を行う。検出された発音時刻のうち、ピークの高い順に上位 3 点を使用し、そこから一定時間長の波形を切り取る。これがテンプレートのもととなる。つまり、テンプレート数は全部で用意した打楽器単音ファイル×3 個となる。

そして、切り取ったそれぞれの波形を短時間フーリエ変換(STFT)し、時間周波数領域のパワー分布の形にする。これがテンプレートとなり、パワー分布の各点における振幅の大きさがマッチングに利用される。

必要があれば、この時点でテンプレートの振幅の正規化を行う。マッチングの際に距離尺度としてユークリッド距離などを用いる場合、テンプレートと対象楽曲で楽器の振幅に差があると（つまり発音している楽器の音量に差があると）、それらがたとえ全く同じ楽器だったとしても、距離が大きくなってしまう。そして正しくマッチングされない原因となる。

これを解決するのが振幅の正規化である。パワー分布内の振幅の最大値ですべての要素を割ることで、どのテンプレートも振幅の値は必ず 0~1 の範囲になる。

4.2.2 楽曲側

テンプレートマッチングではマッチング対象が、テンプレート側と全く同じ形式のデータでないとマッチングを行うことができない。よって、対象の楽曲に対してもテンプレート側とほぼ同様の手順を行なうことになる。

必要があれば、最初に調波音・打楽器音分離ソフトウェア HPSS^[22]を対象楽曲に対して適用し、調波音を取り除いた楽曲を使用して以降の手順を実行する。

テンプレート側と同様に、まずマッチング対象となる楽曲を読み込み、次に楽器の発音

時刻の検出を行う。ただし楽曲には当然打楽器以外の楽器音も含まれているため、打楽器の発音時刻が正確に検出できるとは限らない。HPSS を適用した場合でも、調波音は完全に取り除けるわけではないので同様である。

そこで発音時刻の補正として、検出された発音時刻それぞれの前後に 5 ヶ所ずつ、微小時間ずらした位置に時刻候補を追加する。これらすべての発音時刻候補からテンプレート側同様に波形を一定時間長切り取る。つまり、楽曲側のマッチング対象データは検出された発音時刻の数 \times 11 個となる。

そして、切り取ったそれぞれの波形をテンプレート側と同様短時間フーリエ変換(STFT)し、時間周波数領域のパワー分布の形にする。

必要があれば、テンプレート側と同様にこちらでもパワー分布内の振幅の最大値ですべての要素を割ることで振幅の正規化を行う。

4.2.3 マッチング部

テンプレート側、楽曲側それぞれから切り取ったパワー分布同士の距離／類似度を計算してマッチングする。類似度なら結果の値が高いほど類似している、距離なら低いほど類似しているというように尺度によって判定方法が異なるが、今回はコサイン類似度をコサイン距離として、相関係数を相関距離として扱うことにより、すべての尺度で判定方法を統一する。

先述したとおり今回は k 最近傍法を利用してテンプレートマッチングを行うが、通常の近傍探索では出力される識別クラスは基本的に 1 つ、得票数にタイを認める場合でも最大クラス数は k の値に等しい数である。

しかし、対象楽曲のそれぞれの発音時刻候補において、複数の打楽器が発音していたり、逆にどの打楽器も発音していないということは珍しくなく、これを考慮しないという訳にはいかない。

そこで、半径探索手法を用いることにする。半径探索手法は適切な半径 r を設定し、対象との距離が r 以下であるテンプレートが最も多く所属するクラスに識別するという k 最近傍探索の派生手法である。その図解を Fig4.1 に示す。

今回はこの手法を応用し、対象楽曲との距離が r 以下であるテンプレート数が、楽器クラスごとに予め設定された閾値以上であれば、その楽器が発音していると判定する。楽器クラスごとの閾値を超えたクラスは 1 つも存在しないが、楽器カテゴリごとに設定された比較的lowめの別の閾値を超えていた場合はその楽器カテゴリであると判定する。どの楽器カテゴリの閾値も超えなかった場合、もしくはそもそも距離が r 以下であるテンプレートが 1 つも存在しない場合、その発音時刻候補では打楽器は発音していないと判定する。

半径探索の処理の流れ図を Fig.4.2 に示す。

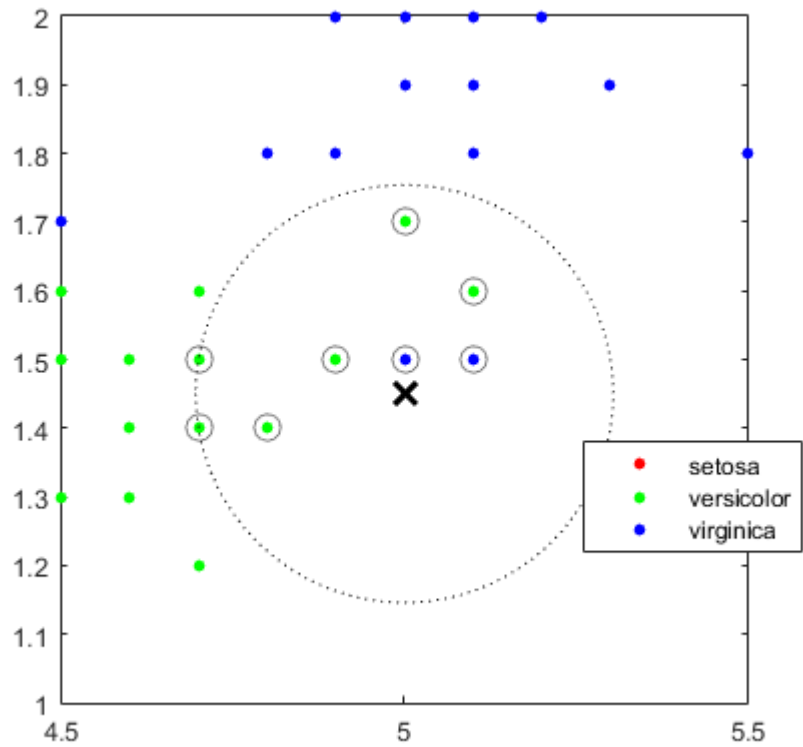


Fig.4.1 半径探索オプションの図解[23]

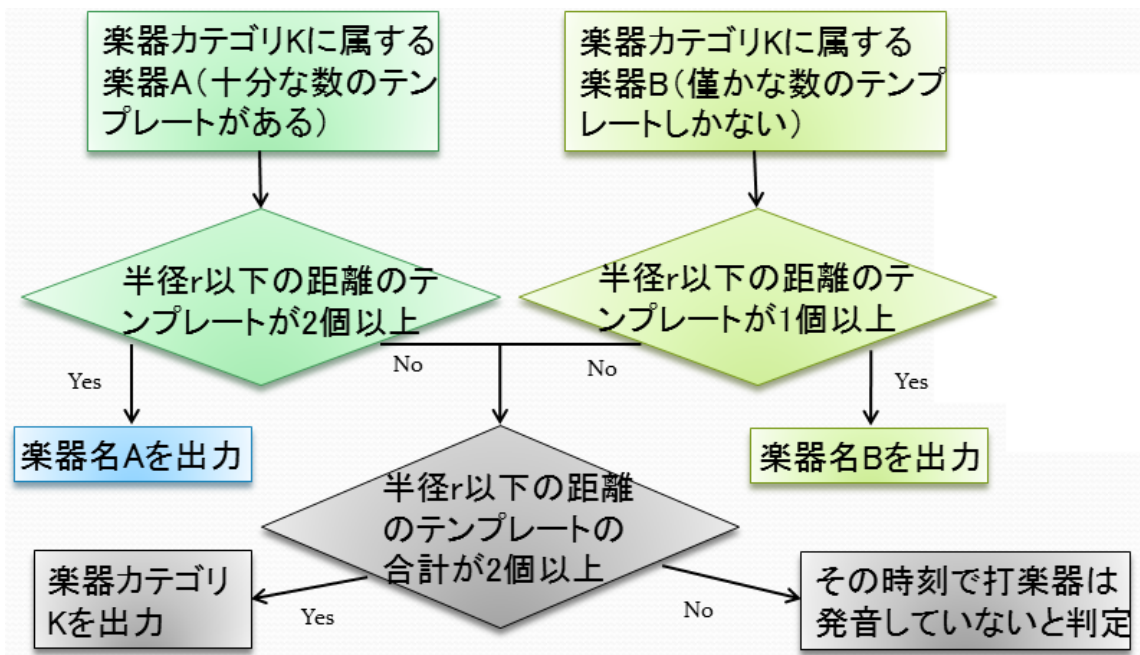


Fig.4.2 半径探索の処理の流れ図

4.3 発音時刻検出方法

4.3.1 打楽器単音の発音時刻検出

読み込んだサウンドファイルの波形から楽器の発音時刻を検出するために、MATLAB のツールボックスの 1 つである MIRtoolbox^[24]に用意されている関数である、`mirpeaks` 関数と `mironsets` 関数を使用した。

`mirpeaks` 関数は波形そのもののピーク、つまり極大となっている時刻を検出する関数である。この関数により打楽器が発音している可能性のある時刻をかなり詳細に検出することができる。

一方、`mironsets` 関数はその名の通り、オンセット (= 発音) を検出するための関数である。波形のエンベロープ (包絡線) のピークを検出することで発音を検出している。この関数により検出される発音時刻候補の数は `mirpeaks` 関数を使った場合に比べて少ないが、残響部分も検出してしまう関係で発音時刻以外の位置が検出されてしまう可能性が非常に高い `mirpeaks` 関数よりも、確実に発音時刻を検出できるといえる。

実験として、スネアドラム単音が収録された `wav` ファイルを読み込んだ波形に対し、`mirpeaks` 関数と `mironsets` 関数をそれぞれ使用して発音時刻を検出し、結果を比較する。

スネアドラム単音は `wav` ファイルに 3 回分収録されているが、そのうちの 1 つ目を再生開始からちょうど 1.0000 秒の地点に調整してある。

Fig.4.3 に読み込んだ `wav` ファイルの波形と、そこに `mirpeaks` 関数により検出された発音時刻位置を赤丸でプロットした図を示す。また、Fig.4.4 に読み込んだ波形のエンベロープと、`mironsets` 関数により検出された発音時刻位置を赤丸でプロットした図を示す。

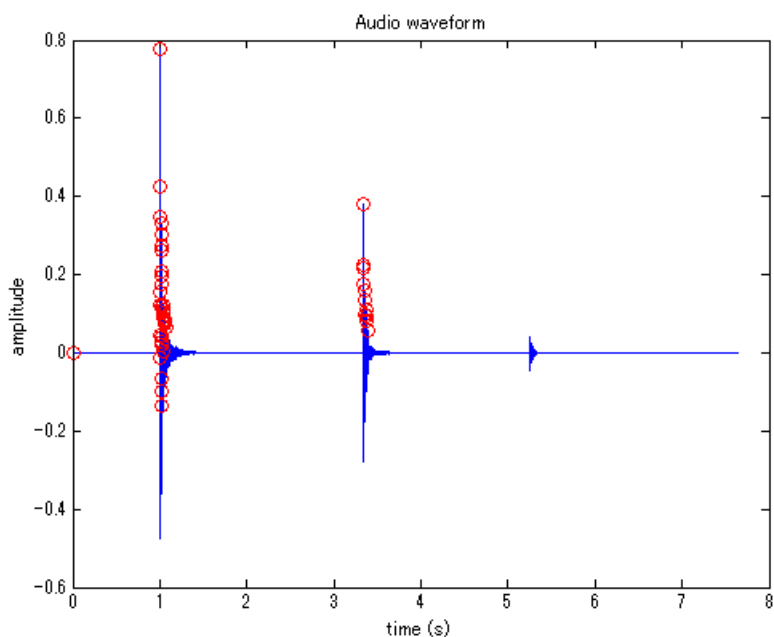


Fig4.3 スネアドラム単音の波形と `mirpeaks` 関数による発音時刻検出結果

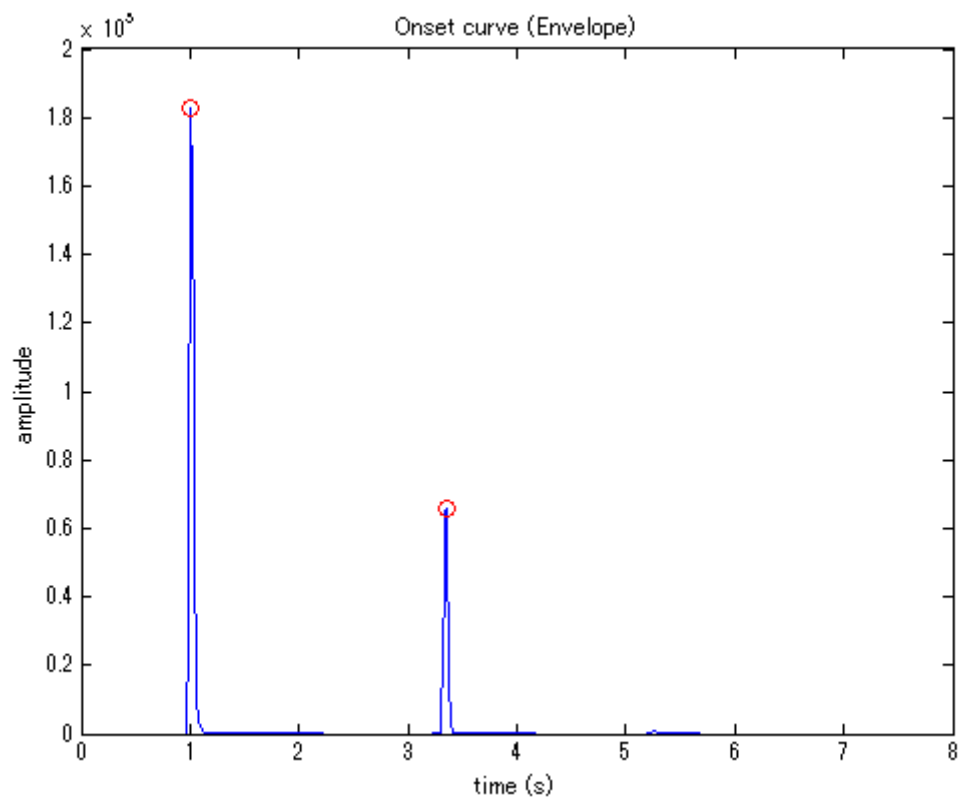


Fig4.4 スネアドラム単音のエンベロープと mironsets 関数による発音時刻検出結果

mirpeaks 関数により検出された発音時刻のうちピークの高い順に上位 10 個と、mironsets 関数により検出された 2 つの発音時刻を比較した表を Table4.1 に示す。

Table.4.1 mirpeaks 関数と mironsets 関数でそれぞれ検出された発音時刻の比較

関数の種類	mirpeaks	mironsets
発音時刻 (秒)	1.0051	1.0113
	1.0098	3.3455
	3.3379	
	1.0068	
	1.021	
	1.0141	
	1.0133	
	1.0166	
	3.3421	
	3.3465	

この比較結果を見ると、**mirpeaks** 関数は実際の発音時刻に対して、その周辺に大量の候補を検出することがわかる。一方、**mironsets** 関数は実際の発音時刻に対して 1 つずつ候補を検出していることがわかる。そして、それぞれの候補のうち最も実際の発音時刻（1.0000 秒）に近いもの（Table4.1 の赤色で示された値）を比較すると、**mirpeaks** 関数のもののほうが近いことがわかる。

この原因としては、打楽器が発音してから最大音量に達するまでにはわずかな時間差が存在する（これをアタックタイム(attack time)という）が、**mironsets** 関数が最大点に到達したときの時刻を検出しているのに対し、**mirpeaks** 関数は立ち上がりの間も発音時刻として検出できるからだと考えられる。

この結果から、テンプレートに使用する打楽器単音の発音時刻検出には **mirpeaks** 関数を使用することとし、検出結果の中からピークの高い順に 3 個の候補を選んで使用する。こうすることでテンプレートの開始時刻に幅をもたせ、バリエーションを増やして楽曲とマッチングさせやすくなる。

発音時刻の位置からは一定時間長の波形を切り取るが、その長さはあらゆる打楽器の発音後音量が減衰する時間までをカバーできるよう考慮して約 0.5 秒間とする。切り取る波形の例を Fig.4.5 に示す。図で赤く網掛けされた部分が切り取る範囲となっている。

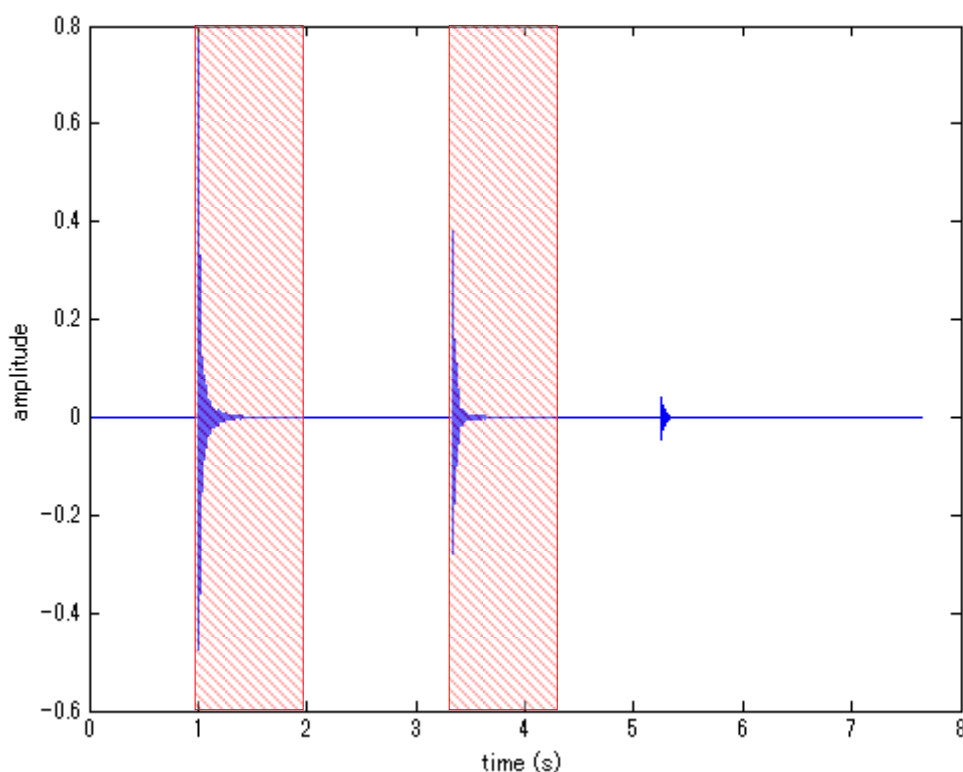


Fig.4.5 切り取る波形の例（網掛け部分が切り取る範囲）

4.3.2 楽曲に含まれる楽器音の発音時刻検出

読み込んだ楽曲の波形から楽器の発音時刻を検出する場合も、打楽器単音の場合と同様に MIRtoolbox に用意されている関数を使用する。ただし、楽曲の場合、何らかの音が鳴っている状態の波形が数分間続くので、`mirpeaks` 関数を使用して発音時刻を検出しようとする膨大な候補が出力される。この候補に対して全てマッチングしようとする非現実的な時間が必要となる。また、長めの楽曲を使用する場合は候補数が多すぎてメモリエラーを起こすこともある。これらの理由から、楽曲に対しては `mironsets` 関数を使用して発音時刻を検出することとする。

Fig.4.6 に `mironsets` 関数を使用して楽曲の楽器の発音時刻を検出した例を示す。

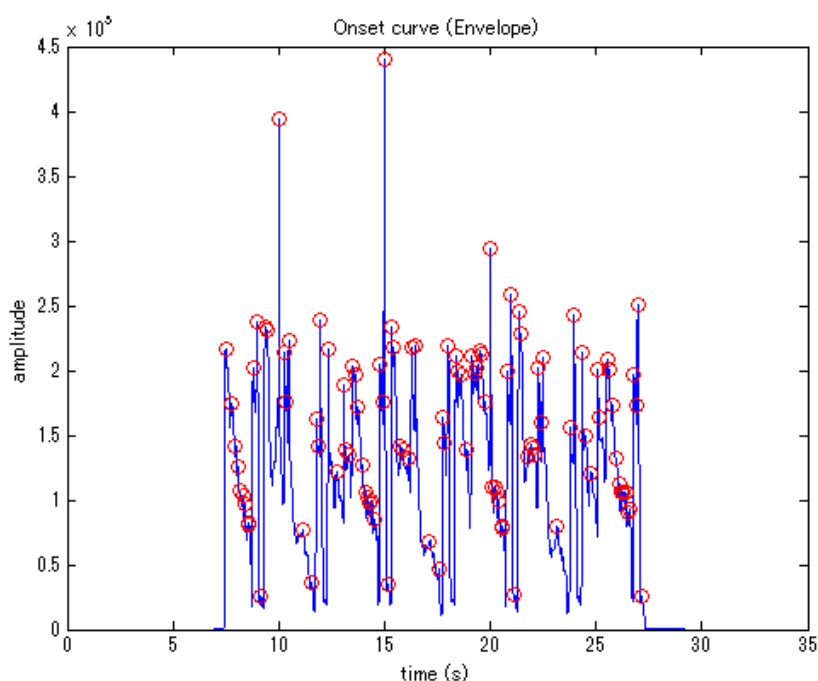


Fig.4.6 `mironsets` 関数を使用した楽曲中の楽器の発音時刻の検出例

楽曲の場合、打楽器以外の楽器音も含まれているため、打楽器音が他の音に隠れて正しく発音時刻を検出できていない可能性がある。そこで、`mironsets` 関数により検出された発音時刻に対し、補正をかけたものを候補として追加する。具体的には、各発音時刻の前後それぞれ 50 マイクロ秒の範囲で、10 マイクロ秒刻みでずらした時間、計 10 個を発音時刻の候補に追加する。

追加した補正後の発音時刻も含めて、すべての発音時刻候補の位置から一定時間長の波形を切り取る。マッチングを行うためにはテンプレートと同じデータ形式である必要があるので、こちらの長さもテンプレートと同様に約 0.5 秒間とする。

4.3 時間周波数構造への変換方法

前の項で切り取った波形を、マッチングで使えるように時間周波数領域のパワー分布の形にする。マッチングで使えるように全く同じ形式のデータを用意するため、ここの手順はテンプレート側と楽曲側で共通したものになる。

変換方法としては短時間フーリエ変換(STFT)を用いるが、このときの設定としては、ハミング窓、ウインドウサイズ 4096 点、フレームシフト長 441 点（サンプリング周波数 44.1kHz の音楽信号で 0.01 秒間）とする。切り取った約 0.5 秒間の波形をこの設定で短時間フーリエ変換すると、時間方向に 42、周波数方向に 2049 の大きさを持つパワースペクトルとなる。このときのパワースペクトルのイメージ図を Fig.4.7 に示す。

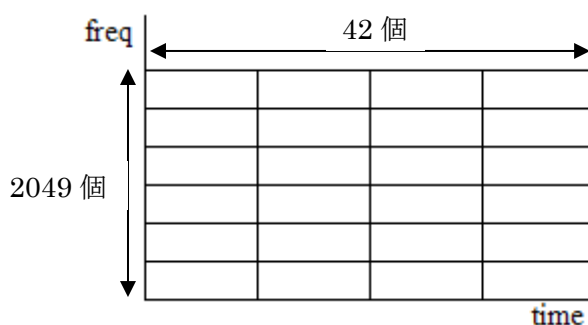


Fig.4.7 パワースペクトルのイメージ図

図を見てもわかる通り周波数分解能が時間分解能に比べてかなり高くなっているが、これはバスドラムなど周波数成分が低帯域に集中する楽器に対応するためである。

バスドラムが周波数成分を持っている帯域のうち、最も低いのは 50Hz 周辺であるとされているが、この周波数帯では僅かな周波数の変化が音の高さに大きく影響する。そのため周波数スペクトルの形式にする場合、周波数分解能をかなり高める必要がある。

スペクトルが周波数方向に 2049 個の大きさを持っている場合、その分解能は約 10.76Hz となるため、低音域にも十分対応できるといえる。

4.4 テンプレートと楽曲のマッチング方法

Fig.4.8, 4.9 にテンプレートと楽曲から切り取った波形を短時間フーリエ変換したスペクトログラムをそれぞれ示す。

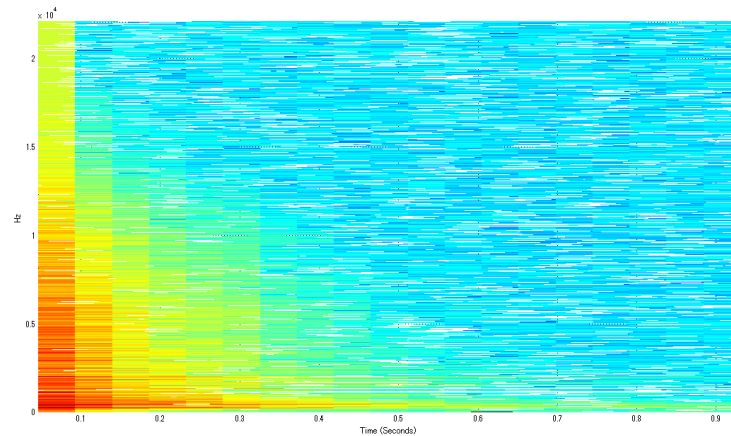


Fig.4.8 テンプレートのスペクトログラム

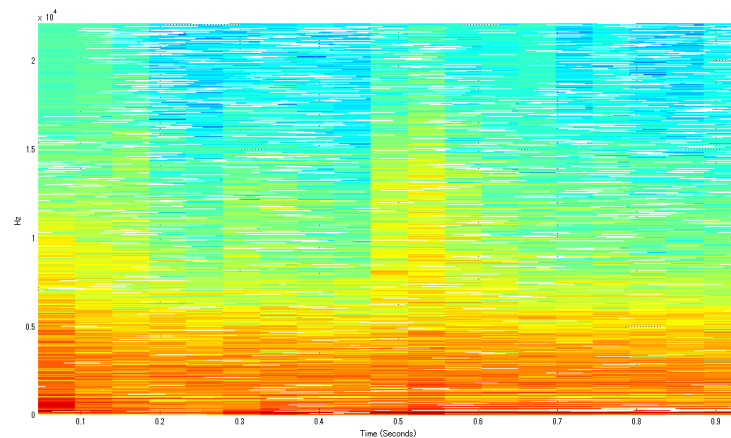


Fig.4.9 楽曲側のスペクトログラム

前の項で述べたとおり、これらは縦（周波数）軸方向に 2049、横（時間）軸方向に 42 の大きさを持っていて、それぞれその時間と周波数に対応する振幅の大きさを表している。提案手法ではこれを $2049 \times 42 = 86058$ 次元のベクトルだと考え、各種距離尺度を使用して距離を計算する。

半径探索に使用する半径については、目安が存在しないため経験則に基づき自力で設定する必要がある。また、距離の値域は当然距離尺度ごとに異なるため、半径も距離尺度ごとにそれぞれ設定する必要がある。設定した半径については、次章に予備実験で得られた結果として掲載している。

楽器ごとに設定する閾値については、十分な数のテンプレートが用意できる楽器とそうでない楽器、大きく 2 つに分けてそれぞれ設定する。

十分な数のテンプレートが用意できる楽器の場合、閾値を 1 に設定すると、次のような問題が起こる可能性が高い。対象楽曲中の複数の楽器音の波形が重なりあった結果、全然関係ない楽器のテンプレートのうちの 1 つとの距離が偶然半径以下になってしまった場合、

その楽器がそのまま発音したと判定されるため、誤認識になる。これは閾値を 2 以上にすることで、その可能性を大幅に抑えられると考えられる。

一方で、同じ種類の楽器にもかかわらず、先述したように個体差や奏法の違いのせいで楽曲とも距離が大きくなってしまうテンプレートも少なからず存在するはずである。閾値を高く設定すると、距離が半径以下に収まっているテンプレートの数が閾値に届かず、その楽器が発音しているにもかかわらず識別されないということになりかねない。

これら 2 つの問題を考慮すると、閾値は基本的に 2、誤認識が目立つ楽器については 3 とするのが望ましいといえる。

少ない数のテンプレートしか用意できない楽器の場合、先述した 2 つの問題の後者が深刻な影響をおよぼすことが予想される。そこで、こちらは閾値を基本的に 1、誤認識が目立つ楽器については 2 とする。

楽器カテゴリの閾値については処理の順番の関係上、単体楽器の場合よりも条件が厳しいと何も出力できないパターンがほとんどを占めてしまうことになるので、「当該楽器カテゴリに属するいずれかの楽器のテンプレートが合計で 2 つ」を閾値とする。

4.5 調波音・打楽器音分離ソフトウェア HPSS について

HPSS(Harmonic/Percussive Sound Separation)は、東京大学の嵯峨山研究室（現 守谷・亀岡研究室）が開発した調波音・打楽器音分離ソフトウェアである。音楽信号内の調波・打楽器という異なる 2 つの成分の異方性に焦点を当て、事前学習や入力音楽信号に関する楽譜や楽器の事前知識を必要とせずに分離を行う。また音楽信号内の調波成分と打楽器の音量バランスを制御するリアルタイムイコライザー機能も持っており、ユーザーはこれらの機能を GUI により利用することができる。その UI を Fig4.10 に示す。

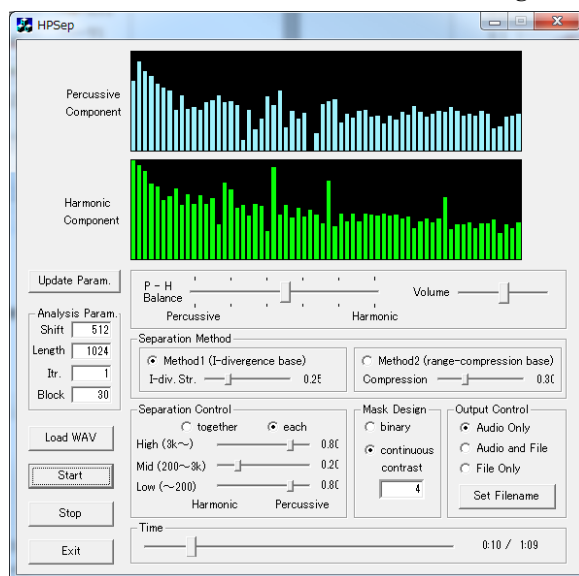


Fig4.10 調波音・打楽器音分離ソフトウェア HPSS の UI

第 5 章 k 最近傍法を用いた実験

5.1 実験準備

最初に、今回の実験で使用するデータについて述べる。

テンプレートに使用する打楽器単音の収録されたデータとしては、産業技術総合研究所より提供されている RWC 研究用音楽データベース(楽器音)^[25]内の RWC-MDB-I-2001 No. 40~44 (打楽器類) を利用している。Table.5.1 に使用している打楽器一覧とその関連情報について示す。なおカテゴリについては後述するが、筆者が独自に付与したものである。

Table5.1 テンプレートとなる打楽器類の内訳 (奏法として数えた場合の一覧)

楽器番号	個体番号	楽器名	楽器記号	楽器名	奏法数	ファイル数	カテゴリ
No. 40	1	邦楽打楽器	ND	長胴太鼓	5	5	膜鳴
			HD	平胴太鼓	4	4	膜鳴
			OD	桶胴太鼓	1	1	膜鳴
			TD	附締太鼓	2	2	膜鳴
			SI	締太鼓	6	6	膜鳴
			DA	大拍子	2	2	膜鳴
			KZ	小鼓	5	5	膜鳴
			AT	当り鉦	4	4	打奏金属製
			CH	チャップパ	2	2	打奏金属製
			DO	銅鑼	2	2	打奏金属製
			SZ	鈴	2	2	鈴
			KM	小木魚	2	2	打奏木製
No. 41	1	コンサートドラムス 1	BD	バスドラム	2	2	バス
			SD	スネアドラム	10	10	スネア
			TT	トムトム	5	5	タム
			Y3	合わせシンバル	1	1	クラッシュ
			YS	サスペンドシンバル	3	3	クラッシュ
			CN	カスタネット	1	1	打奏木製
			TA	トライアングル	3	3	打奏金属製
			WB	ウッドブロック	1	1	打奏木製
			TM	タンバリン	2	2	複合
	2	コンサートドラムス 2	BD	バスドラム	2	2	バス
			SD	スネアドラム	10	10	スネア
			TT	トムトム	5	5	タム
			Y3	合わせシンバル	1	1	クラッシュ
			YF	フィンガーシンバル	1	1	打奏金属製
			CN	カスタネット	1	1	打奏木製
			TA	トライアングル	3	3	打奏金属製
			WB	ウッドブロック	1	1	打奏木製
			TM	タンバリン	2	2	複合
	3	コンサートドラムス 3	BD	バスドラム	2	2	バス

			SD	スネアドラム	10	10	スネア
			Y3	合わせシンバル	1	1	クラッシュ
			BY	シンバル付きバスドラム	2	2	複合
			CN	カスタネット	1	1	打奏木製
			TA	トライアングル	3	3	打奏金属製
			WB	ウッドブロック	1	1	打奏木製
			TM	タンバリン	2	2	複合
No. 42	1	ロックドラムス 1	BD	バスドラム	2	2	バス
			SD	スネアドラム	30	30	スネア
			FT	フロアタム	3	3	タム
			LT	ロータム	3	3	タム
			MT	ミッドタム	3	3	タム
			HT	ハイタム	3	3	タム
			HH	ハイハット	13	13	ハイハット
			YR	ライドシンバル	3	3	クラッシュ
			Y1	クラッシュシンバル 1	3	3	クラッシュ
			Y2	クラッシュシンバル 2	3	3	クラッシュ
			YH	チャイナシンバル	1	1	クラッシュ
			YA	アタックシンバル	1	1	クラッシュ
	2	ロックドラムス 2	BD	バスドラム	2	2	バス
			SD	スネアドラム	30	30	スネア
			FT	フロアタム	3	3	タム
			LT	ロータム	3	3	タム
			HT	ハイタム	3	3	タム
			HH	ハイハット	13	13	ハイハット
			YR	ライドシンバル	3	3	クラッシュ
			Y1	クラッシュシンバル 1	3	3	クラッシュ
	3	ロックドラムス 3	BD	バスドラム	2	2	バス
			SD	スネアドラム	30	30	スネア
			FT	フロアタム	3	3	タム
			LT	ロータム	3	3	タム
			HT	ハイタム	3	3	タム
			HH	ハイハット	13	13	ハイハット
			YR	ライドシンバル	3	3	クラッシュ
			Y1	クラッシュシンバル 1	3	3	クラッシュ
No. 43	1	ジャズドラムス 1	BD	バスドラム	2	2	バス
			SD	スネアドラム	48	48	スネア
			FT	フロアタム	7	7	タム
			HT	ハイタム	7	7	タム
			HH	ハイハット	9	9	ハイハット
			YR	ライドシンバル	3	3	クラッシュ
			Y1	クラッシュシンバル 1	3	3	クラッシュ
			Y2	クラッシュシンバル 2	3	3	クラッシュ
	2	ジャズドラムス 2	BD	バスドラム	2	2	バス
			SD	スネアドラム	54	54	スネア
			FT	フロアタム	8	8	タム

			LT	ロータム	8	8	タム
			HT	ハイタム	8	8	タム
			HH	ハイハット	13	13	ハイハット
			YR	ライドシンバル	3	3	クラッシュ
			Y1	クラッシュシンバル 1	3	3	クラッシュ
			Y2	クラッシュシンバル 2	3	3	クラッシュ
	3	ジャズドラムス 3	BD	バスドラム	2	2	バス
			SD	スネアドラム	42	42	スネア
			FT	フロアタム	6	6	タム
			LT	ロータム	6	6	タム
			HH	ハイハット	13	13	ハイハット
			YR	ライドシンバル	3	3	クラッシュ
			Y1	クラッシュシンバル 1	3	3	クラッシュ
No. 44	1	パーカッション 1	TM	タンバリン	2	2	複合
			CW	カウベル	4	4	打奏金属製
			CA	カバサ	1	1	振奏
			MC	マラカス	1	1	振奏
			WH	ホイッスル	2	2	気鳴
			SW	サンバホイッスル	6	6	気鳴
			TA	トライアングル	2	2	打奏金属製
			SK	シェーカー	1	1	振奏
			ST	スティック	1	1	打奏木製
			GO	ドラ（ゴング）	1	1	打奏金属製
	2	パーカッション 2	BG	ボンゴ	7	7	膜鳴
			TI	ティンバレス	7	7	膜鳴
			AB	アゴゴベル	3	3	打奏金属製
			G1	ギロ（ひょうたん）	4	4	擦奏
			G2	ギロ（鉄）	4	4	擦奏
			G3	ギロ（竹）	4	4	擦奏
			CS	クラベス	2	2	打奏木製
			WB	ウッドブロック	1	1	打奏木製
			CN	カスタネット	4	4	打奏木製
			SR	スルドー	5	5	膜鳴
			VS	ビブラスラップ	1	1	擦奏
			CU	クイーカ	4	4	擦奏
			YF	フィンガーシンバル	1	1	打奏金属製
			YC	クラッシュシンバル	1	1	クラッシュ
			WC	ウィンドチャイム	4	4	鈴
			BE	ベルツリー	1	1	鈴
	3	パーカッション 3	CO	コンガ	6	6	膜鳴
			JB	ジャンベ	2	2	膜鳴
			TK	トーキングドラム	3	3	膜鳴
	4	パーカッション 4	CW	カウベル	4	4	打奏金属製
			BG	ボンゴ	7	7	膜鳴
			CO	コンガ	6	6	膜鳴

			AB	アゴゴベル	4	4	打奏金属製
			CA	カバサ	1	1	振奏
			MC	マラカス	1	1	振奏
			CS	クラベス	2	2	打奏木製
			SR	スルドー	5	5	膜鳴
			SK	シェーカー	1	1	振奏
			WC	ウインドチャイム	4	4	鈴
			TL	タブラ	13	13	膜鳴
5	パーカッション 5		TM	タンバリン	2	2	複合
			CW	カウベル	4	4	打奏金属製
			BG	ボンゴ	7	7	膜鳴
			CO	コンガ	6	6	膜鳴
			AB	アゴゴベル	4	4	打奏金属製
			CA	カバサ	1	1	振奏
			MC	マラカス	1	1	振奏
			CS	クラベス	2	2	打奏木製
			TA	トライアングル	2	2	打奏金属製
			SR	スルドー	5	5	膜鳴
			QJ	キハーダ	1	1	擦奏
			SK	シェーカー	1	1	振奏
			WC	ウインドチャイム	4	4	鈴
			ST	スティック	1	1	打奏木製
			BL	チューブラベル	2	6	打奏金属製
			ME	メトロノーム	2	2	複合
			HA	ハンドクラップ	1	1	打奏木製
			AP	拍手	1	1	打奏木製

また、マッチング対象の楽曲データとしては、市販のドラム練習用 CD に収録されている楽曲を使用する。調波構造を持つ楽器とドラムによって演奏された長さ 1 分程度の楽曲（以降、通常楽曲と呼称する）、それと練習用に元の楽曲からドラムパートを抜いて収録された楽曲（以降、ドラムマイナスイワン楽曲と呼称する）の 2 種類を用途によって使い分ける。

Table.5.2 に打楽器単音のデータファイルと楽曲データファイルのデータ形式について記述する。それぞれのデータ形式は共通したものである。

Table.5.2 打楽器単音と楽曲のデータ形式（共通）

音楽ファイル形式	WAV ファイル
サンプリング周波数	44100Hz
量子化ビット数	16bit
チャンネル数	1（モノラル）

また、前章の提案手法を踏まえた処理の実装、および本章の実験にはすべて MATLAB^[26] を利用している。

5.2 k 最近傍法の有効性についての検証実験

5.2.1 実験概要

本実験は、k 近傍法を用いた提案手法が、各打楽器音を正しい打楽器クラスに分類するという、実験の基本目的に対して有効な手法であるということを確認するのが目的である。またそれと同時に、提案手法を実装したシステムがすべて期待通りに動作するかどうかを確認するという意味合いも存在する。

本実験では、認識対象を楽曲（つまり混合音）ではなくテンプレートと同じような打楽器単音（ただし、テンプレートと認識対象で同じファイルは使用しない）という条件で行う。こうすることで、提案手法が有効であり、かつその手法が正しくシステムに実装できていれば間違いなく高い類似度を示すはずである。そうでない場合、システムか手法のどちらかに問題が存在しているということになる。

5.2.2 実験準備

実験に使用するテンプレート、テストデータともに楽器音データベースのロックドラムスのデータ（Table.1 の No.42）を使用する。スネアドラム、バスドラム、ハイタム、ロータム、フロアタムを'drum'、ハイハットシンバル、ライドシンバル、クラッシュシンバルを'cymbal'として、2 クラスの分類を行う。テンプレートは 150 個（うちドラム 93 個、シンバル 57 個）とし、テストデータはテンプレートで使用しなかった、上記の打楽器を各 1 個ずつ、合計 8 個（うちドラム 5 個、シンバル 3 個）とする。

距離尺度については MATLAB の knnsearch 関数^[23]で利用できるオプションのうち、2 値データ向けの尺度であるハミングとジャカードを除いたものすべて（ユークリッド、標準化ユークリッド、マハラノビス、市街地、ミンコフスキー、チェビシェフ、コサイン、相関、ハミング、スピアマン）を使用して実験を行った。ただし、マハラノビスについては逆行列を求める処理に不具合があり正しく計算できなかったため、結果のグラフや表では空欄としてある。

その他の設定として、本実験では提案手法で述べた半径探索オプション(rangeseach)は使用せず、通常の k 最近傍探索を使用する。探索する最近傍点の数 k は 5、kd ツリーオプションは使用せず全件検索とする。

5.2.3 実験結果

Fig5.1 に、それぞれの距離尺度での分類の正解率を示す。今回はクラス数は 2、最近傍点の数は 5 個なので、最近傍点のうち正しいラベルのものが 3 個以上の場合が正解、2 個以下

の場合は不正解とする。

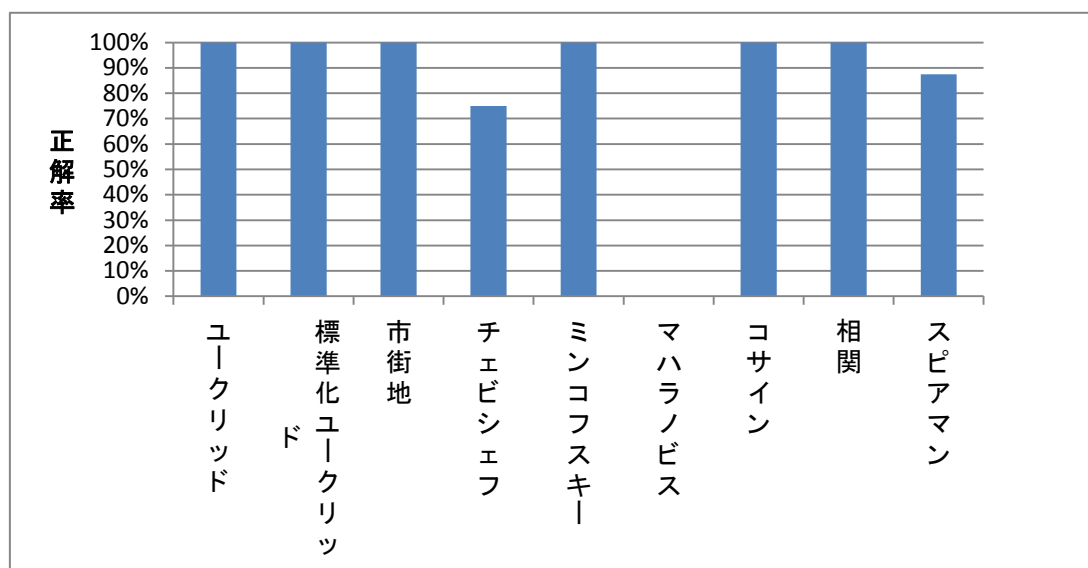


Fig5.1 それぞれの距離尺度での分類結果の正解率

本実験はテンプレート、テストデータともに認識に関係ない音が含まれていない単音、また分類するクラス数も 2 と、認識を行う条件としてはかなり緩くしているので、どの尺度を使ってもかなり高い正解率を得ることができた。実用的な条件にすると正解率が下がることは予想されるが、少なくとも打楽器音分類において k 近傍法を用いた手法が有効であることは確かめられたといえる。

Fig5.1 は最近傍点 5 個から総合して得られた分類結果の正解率であるが、近傍点 1 つ 1 つの分類結果の正解率を Fig.5.2 に示す。つまり、5 個×8 テストデータ=40 個の近傍点のうち、何個がそれぞれのテストデータの正解ラベルと一致しているかということである。

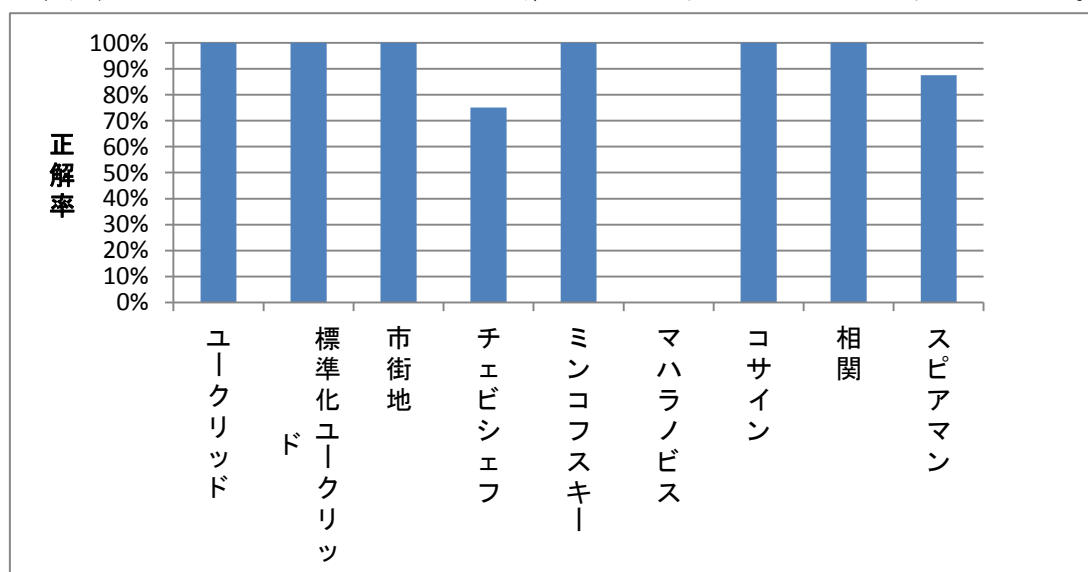


Fig5.2 それぞれの距離尺度での分類結果の正解率

結果は最近傍点 5 個から総合して得られたものと大きな違いはないが、コサイン距離や相関係数において他の距離尺度よりも若干良い結果になっていることがわかる。このことから、切り取ったパワー分布同士のマッチングを行う手法においては、コサイン距離や相関係数がより有効な距離尺度なのではないかと考えられる。

5.3 半径探索に必要な半径の値を定める予備実験

5.3.1 実験概要

前の項で有効性が確かめられた k 最近傍法を実際の楽曲に適用するにあたり、 k 最近傍法の半径探索オプションにおいて重要な役割を担う半径を設定する必要がある。本実験はその半径を定めるという目的で、前章で述べたとおり各距離尺度ごとに行っていく。

ある楽器の残響音が他の楽器音に影響を及ぼさない理想的条件の楽曲を考えた時、Fig5.3 に示すように、各テンプレートと通常楽曲の間の距離は比較的小さくなるため、グラフの下の方に点が集中するはずである。一方で、対象がドラムマイナスイン楽曲の場合は、距離が比較的大きくなるため、グラフの上の方に点が集中するはずである。ドラムマイナスインの場合の中での最小距離よりも小さい値を半径に設定することで、通常楽曲の場合の閾値以下の点を打楽器を同定するための判断材料にすることができる。

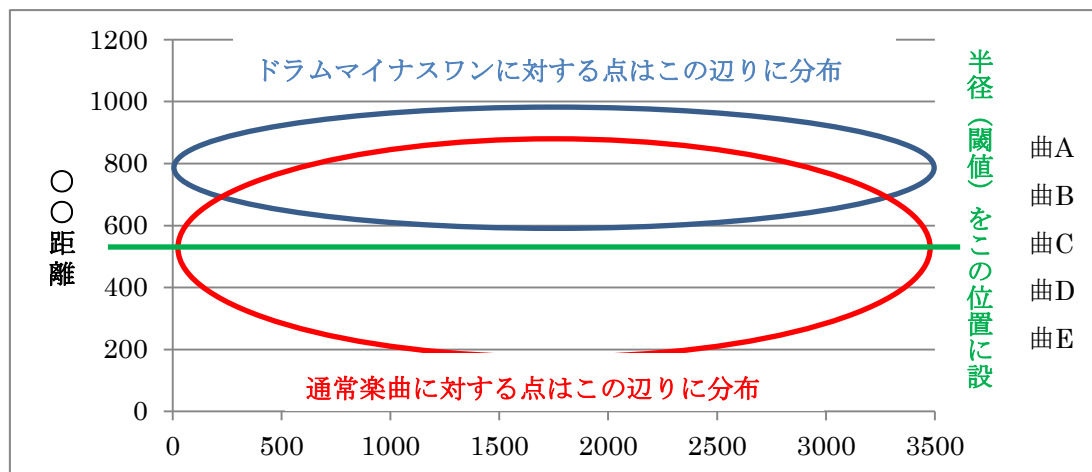


Fig5.3 理想的条件の楽曲の場合の半径の設定方法

5.3.2 実験準備

実験に使用するテンプレートは RWC 楽器音データベースのロックドラムスのデータ (Table.1 の No.42) 全 158 個とし、テストデータには通常楽曲 5 曲とそれらに対応するマイナスイン楽曲 5 曲、合計 10 曲を使用する。

楽曲ごとに検出した楽器発音時刻約 3000~4000 ヶ所からマッチング対象のパワー分布を作成し、それぞれ最近傍点探索 ($k = 1$ の場合) を行う。理想的条件の楽曲ならば、通常楽

曲から最近傍点までの距離は、ドラムマイナスイワン楽曲のそれよりも短くなるはずである。

距離尺度には検証実験で結果を出力できた 8 種類を使用し、その中で通常楽曲とマイナスイワン楽曲ではっきりと差が現れている距離尺度を実際の認識実験で採用、そしてこの実験で求めた半径を使用する。

また前処理として、HPSS を使用する。楽曲から調波成分がある程度取り除かれるため、テンプレートの打楽器音と対象の楽曲との間の距離は小さくなる（=Fig5.3 の点が集中する位置がより下の方になる）と考えられる。

また、ユークリッド、標準化ユークリッド、市街地、チェビシェフの 4 つの距離尺度については、テンプレートの打楽器音と楽曲中の打楽器音に音量差が存在する場合、計算方法の関係でそれが原因となって距離が大きくなってしまふことが考えられる。そこで、これらの距離尺度を使用する場合は結果を改善するために、提案手法で述べた音量正規化により双方の音量差を減らす。

5.3.3 実験結果

実験結果のうち、2 種類の楽曲間ではっきりと差が現れている距離尺度 5 つについて、最近傍点までの距離を散布したグラフを Fig5.4~5.8 にそれぞれ示す。どのグラフも、左側が通常楽曲で右側がマイナスイワン楽曲の結果となっている。また、設定する半径をグラフに破線で示している。

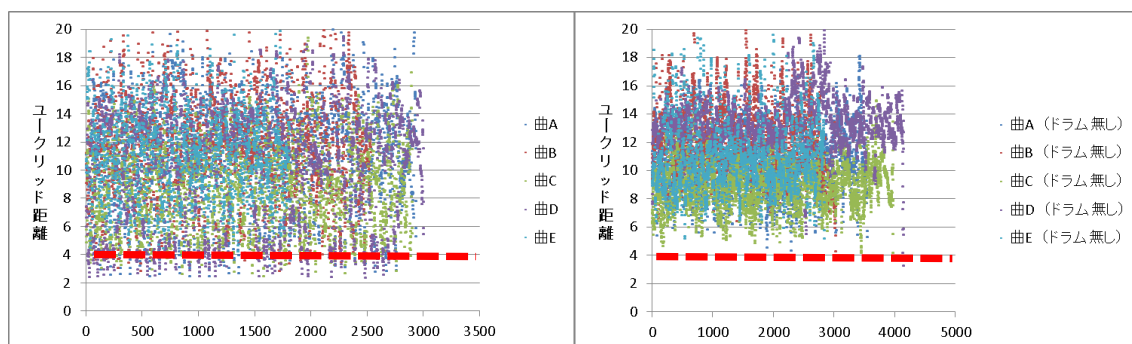


Fig5.4 ユークリッド距離の場合の最近傍点までの距離

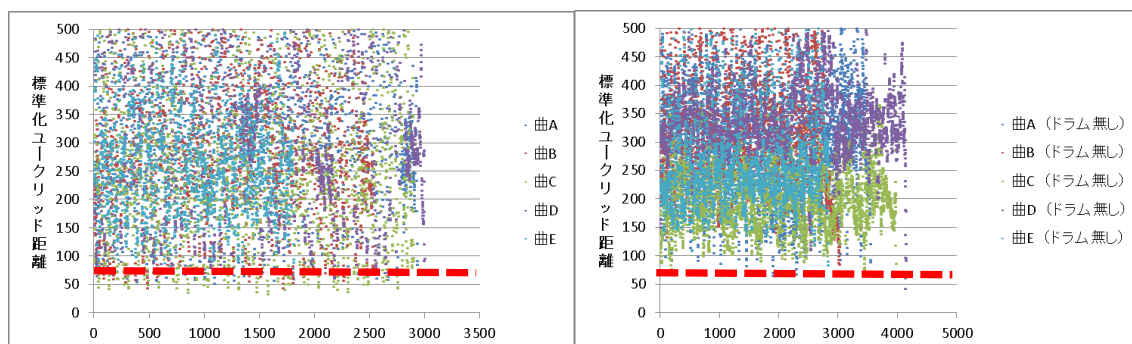


Fig5.5 標準化ユークリッド距離の場合の最近傍点までの距離

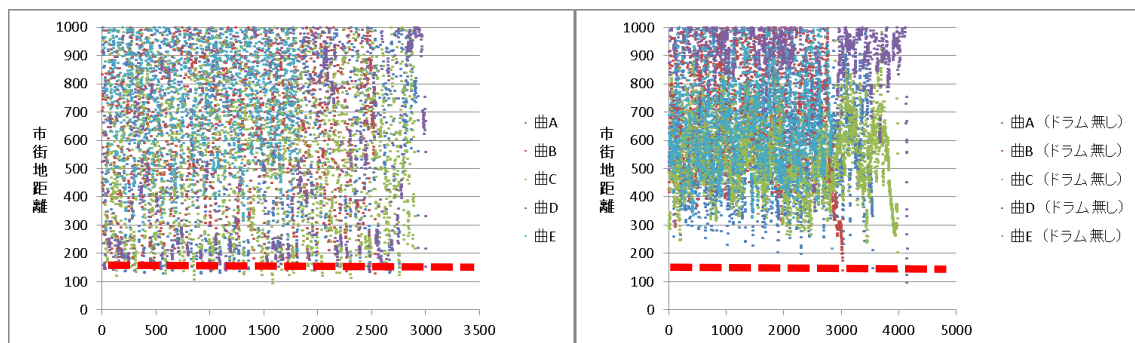


Fig5.6 市街地距離の場合の最近傍点までの距離

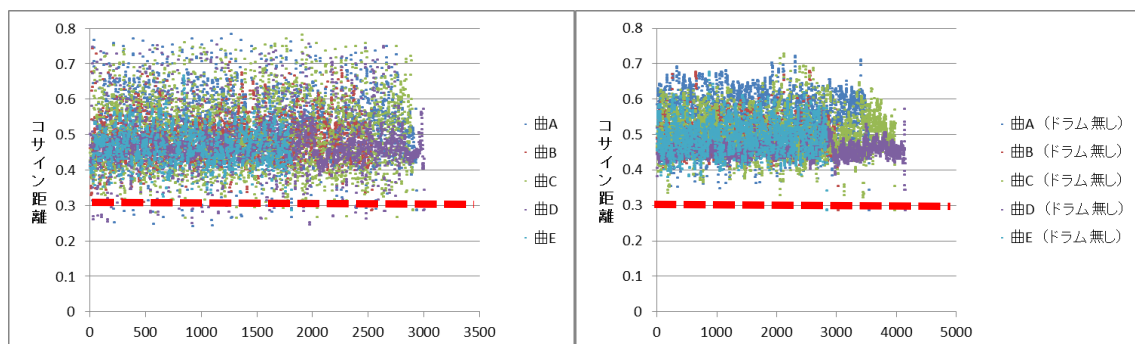


Fig5.7 コサイン距離の場合の最近傍点までの距離

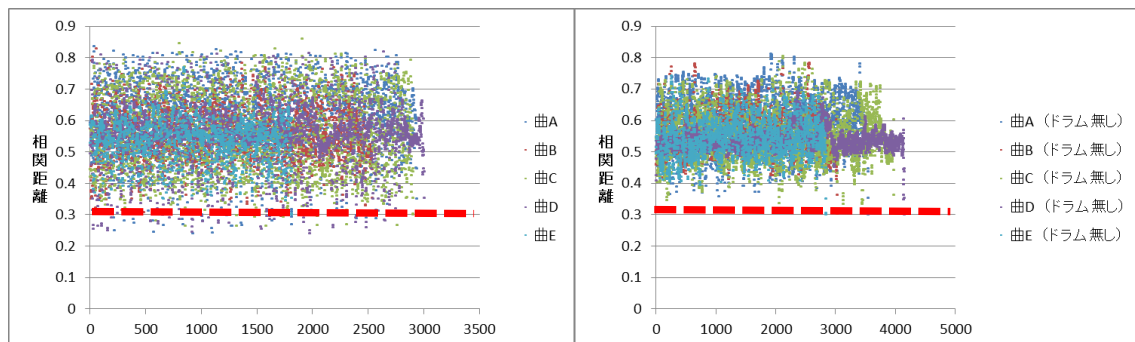


Fig5.8 相関距離の場合の最近傍点までの距離

5.4 提案手法による実際の楽曲を対象とした認識の実験

5.4.1 実験概要

前の項の半径を求める予備実験により、半径探索に使用する半径は Table5.3 のように設定すれば良いという結果が得られた。このうち上の 3 つの距離尺度に関しては振幅の最大値を使った音量正規化を行った場合の設定である。

Table5.3 設定する半径

ユークリッド距離（正規化）	4
標準ユークリッド距離（正規化）	70
市街地距離（正規化）	150
コサイン距離	0.3
相関距離	0.3

本実験ではこの設定を使い、実際の楽曲を対象として認識を行い精度を確認する。

5.4.2 実験準備

認識対象の楽曲は、ドラムマイナスイン楽曲に、ある特定の打楽器音 1 種類を 5 秒毎に 11 ヶ所挿入したものを使用する。楽曲の編集画面は Fig5.9 のようになっている。

このように編集した楽曲を 3 種類のマイナスイン楽曲*10 種類の打楽器音=30 曲作成し、HPSS を前処理として使用した後、分離後の打楽器成分に対して実験を行った。

テンプレートとなる打楽器音は、本実験では楽器音データベースのほぼすべての打楽器音、合計 662 個を使用して実験を行う。

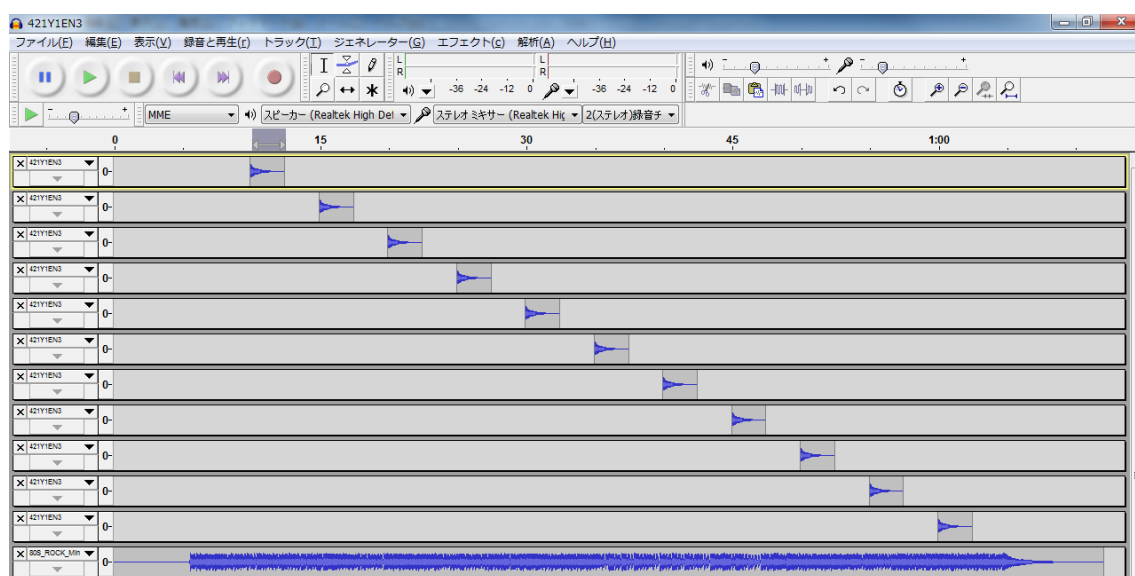


Fig5.9 認識に使用する楽曲の編集画面

5.4.3 実験結果

ユークリッド距離、標準ユークリッド距離、市街地距離では、どの打楽器音を挿入した場合でも出力結果がほぼ変わらず、また無関係だと思われる楽器も多数認識結果として出力された。それに対してコサイン距離、相関距離ではマッチング対象の楽曲に対応した楽器がある程度出力されたので、有効な距離尺度であると言える。そこで 2 つのうち、より

結果の良かったコサイン距離を使用して認識率を計算する。Fig5.10 にコサイン距離を使用した時の認識結果をもとにした打楽器ごとの認識率を示す。また Fig5.11 に比較用として、実験に使用した楽器ごとのテンプレート数を示している。

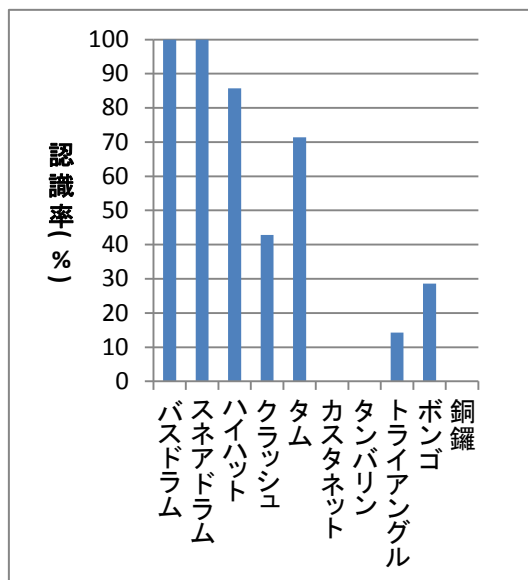


Fig5.10 コサイン距離の認識結果をもとにした打楽器ごとの認識率

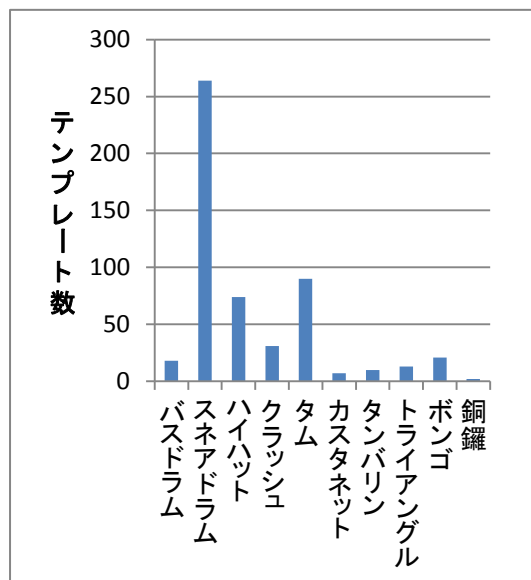


Fig5.11 実験に使用した楽器ごとのテンプレート数

この結果からいえる最も重要なポイントとしては、テンプレートが豊富に用意できる楽器については従来手法に及ばないまでも、70%以上というかなり高い認識率を得ることができた。一方でそうでない楽器に関しては、20%に満たないような低い認識率になってしまった、ということである。実験に使ったテンプレート数のグラフを認識結果と比較してみてもある程度相関があることが一目瞭然である。

他には、膜鳴楽器のほうが体鳴楽器に比べてより高い認識率を示す傾向などが挙げられる。

今後認識率の向上を目指すにあたっては、まず認識率の底上げをするために、認識率の低い楽器に関しては楽器単体ではなく楽器カテゴリごとに扱う、言い換えれば楽器をグループ化することにより、少しでも該当テンプレートの数を増やす必要があるといえる。

加えて、曲によっては付加した打楽器音が他の音に埋もれてしまった可能性が考えられる。調波音が打楽器音に比べて極端に音量が大きいと、両者が混在したときに打楽器音をかき消してしまうため、たとえ HPSS で調波音を分離しても正しく打楽器音を認識できなくなると考えられる。これについても音量調整などにより解決する必要があるといえる。

第 6 章 深層学習を用いた提案手法

6.1 概要

4 章で提案した k 最近傍法による同定手法は、半径探索オプションの半径を設定する際、どうしてもテンプレート数の多い楽器に基準が寄ってしまっていた。こうなると必然的にテンプレート数が少ない楽器の認識率が余計に下がってしまうが、後者を基準として半径を設定するのも難しい。

そこで、楽器ごとのテンプレート数の極端な差にも対応できる方法として、深層学習 (Deep Learning) により作成した識別器を認識処理部に導入する手法を新たに提案する。

具体的な方法としてはまず教師データとなるパワー分布を、 k 最近傍法では数値 (行列データ) として扱っていたが、深層学習ではスペクトログラム、つまり画像として学習を行う。

認識方法に関しては 2 種類の手順が考えられる。1 つ目は認識対象の画像に対し、認識したい楽器 (カテゴリ) の数だけ用意したクラス + 非打楽器音クラスで多クラス識別を行う方法である。この方法では識別器は 1 つしか必要としないため、学習も認識も比較的高速に行うことができる。また、実装も比較的容易に行うことができるという利点もある。しかし、致命的な欠点として一度に 1 つの楽器しか認識ができないという問題がある。4 章でも述べたが、1 つの発音時刻候補の位置で 2 つ以上の打楽器が同時に発音する可能性は十分にあるため、すべての楽器を認識できることが望ましい。そういった意味ではこの方法は、目的を完全に果たすことはできないといえる。

2 つ目は認識対象の画像に、認識したい特定の楽器 (カテゴリ) が含まれているかいないかという 2 クラス (一対他) 識別器を楽器 (カテゴリ) の数だけ作成して認識を行う手法である。この手法であれば、たとえ 1 つの発音時刻候補の位置で 2 つ以上の打楽器が同時に発音していても、理論上すべて認識することが可能である。しかし、必要な識別器の数が大幅に増えるため、それに伴い学習や認識に必要な時間も大幅に増加し、複数の識別器を同時に扱うため実装も比較的難解になるという欠点も存在する。

6.2 処理の流れ

6.2.1 教師データ側

まず打楽器単音の音楽ファイルからパワー分布を作成する部分までは k 最近傍法の提案手法とほぼ共通しているため、これを踏襲して行う。ただし打楽器単音の発音時刻の検出の際、検出された発音時刻のうち、使用するのはピークの高い上位 1 点のみとする。つまり、テンプレートの数は用意した打楽器単音ファイルの数と同じになる。この変更は次章で行う教師データに関する予備実験に配慮するためである。また、振幅の正規化処理は省

略する。これは、スペクトログラムを作成する際に同様の処理が入っているためである。

6.2.2 楽曲側

深層学習による識別においても、認識対象が教師データと同じ形式のデータでないと正しく認識できない。よって、対象の楽曲に対してもほぼ同様の手順を行なうことになる。

相違点としては、楽曲に対しては発音時刻の検出は行わず、波形の切り出し開始点を楽曲の始点から終点まで 0.05 秒刻みでずらしながら以降の処理を行っていく。これは、深層学習は時間軸方向のずれに対してロバストであり、厳密な波形の切り出し開始点を必要としないからである。また振幅の正規化についても、教師データ側と同様の理由で省略する。

6.2.3 深層学習による識別器作成

まず、打楽器音をいくつかのカテゴリに分類する。ドラムセットの構成楽器など、十分な数の教師データが用意できる楽器に関しては楽器単体で 1 つのカテゴリとする。それ以外の、僅かな数の教師データしか用意できない楽器に関しては、楽器の材質や奏法に注目して 8 個のカテゴリを作り、それらに分類した。

作成したカテゴリはバスドラム、クラッシュ、ハイハット、スネアドラム、タム、気鳴、鈴、複合、膜鳴、打奏金属製、擦奏、振奏、打奏木製の合計 13 個である。打楽器の分類先カテゴリは Table5.1 に記載している。

次に学習プログラムに識別したいクラスの画像群を入力して学習させ、識別器を作成する。多クラス識別を行う方法の場合、上記の 13 カテゴリをそのままクラス 0~12 に対応させて入力する。また、非打楽器音クラスとして、トランペットやバイオリンなどの調波音、および無音から作成したスペクトログラム合計 100 枚をクラス 13 に入力する。これにより 14 クラス識別器が作成される。

一方、2 クラス識別を行う方法の場合、上記の 13 カテゴリのうち 1 カテゴリを選択してクラス 0 に入力し、残りの 12 カテゴリをクラス 1 に入力する。これにより選択したカテゴリの楽器が認識対象に含まれていたらクラス 0 を、含まれていなかったらクラス 1 を出力する 2 クラス識別器が作成される。この手順を各カテゴリに対して行い、13 個の識別器を作成する。

6.2.4 マッチング部

楽曲側で作成した各スペクトログラムを識別器に入力し、それぞれ認識を行う。

多クラス識別を行う方法の場合、スペクトログラムを 14 クラス識別器に入力すると、最も発音している可能性が高い打楽器の属するクラスが出力される。打楽器が 1 つも発音していないと識別された場合は非打楽器クラスが出力される。これらをそのまま認識結果とする。識別の流れ図を Fig6.1 に示す。

2 クラス識別を行う方法の場合、スペクトログラムを 13 個の 2 クラス識別器全てに入力

する。各識別器において、対応する打楽器音が認識対象に含まれていると識別された場合はクラス 0 を出力する。含まれていないと識別された場合はクラス 1 を出力する。クラス 0 を出力した識別器に対応する楽器カテゴリをすべて表示し、これを認識結果とする。クラス 0 を出力した識別器が 1 つもない場合も当然存在する。この場合、認識対象に打楽器音が含まれていないという認識結果となる。識別の流れ図を Fig6.2 に示す。

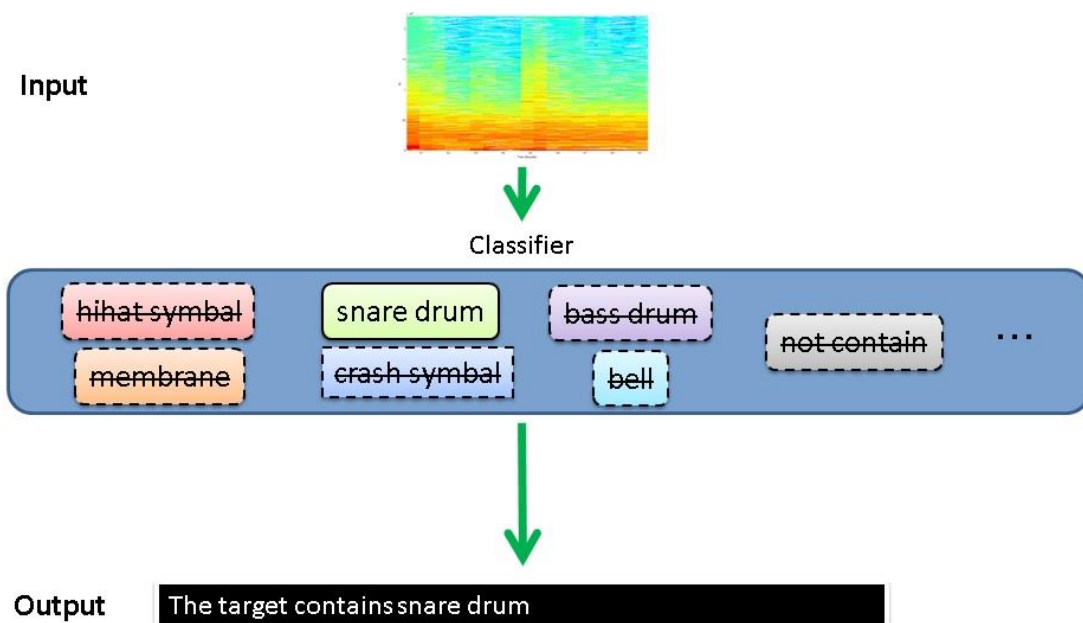


Fig6.1 多クラス識別の流れ図

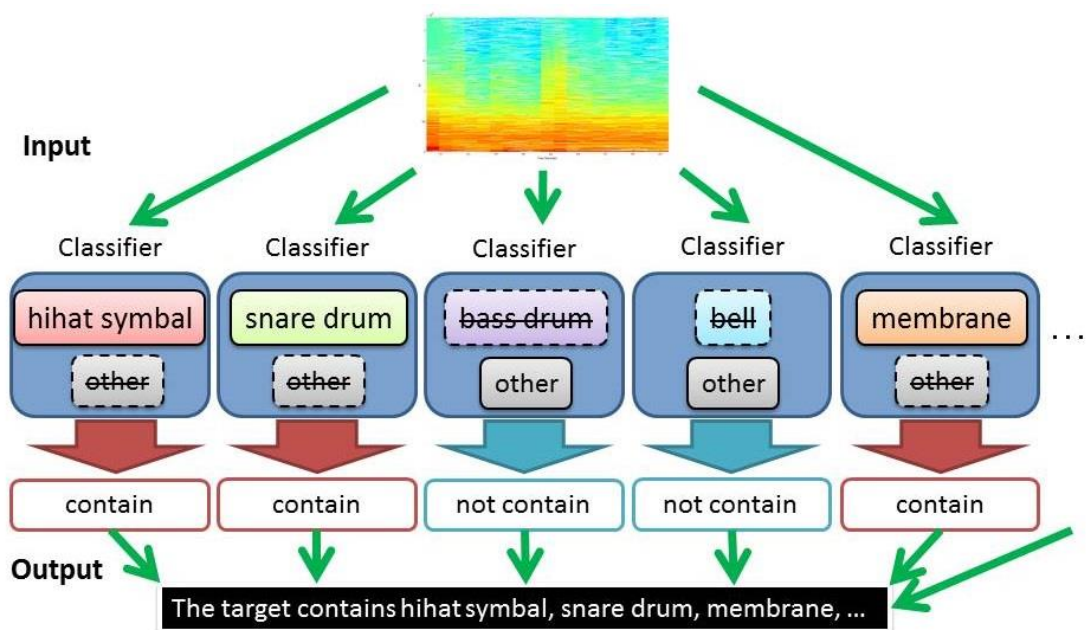


Fig6.2 2 (一対他) 識別の流れ図

第 7 章 深層学習を用いた実験

7.1 実験準備

まず今回の実験で使用するデータについてだが、教師データについては k 最近傍法による実験と同様、Table.5.1 に示す楽器音データベースのすべての打楽器音を利用している。打楽器単音のデータファイル、それから後の項で述べる楽曲データファイルのデータ形式についても、 k 最近傍法による実験と同様、Table.5.2 に示しているとおりの形式である。また、前章の提案手法を踏まえた処理の実装、および本章の実験には Python 用フレームワーク”Caffe”^[27]を使用している都合上、すべて Python を利用している。Caffe については次の項で詳しく説明する。

7.2 Caffe について

7.2.1 概要

Caffe は処理の高速性とモジュール性を考慮した深層学習(Deep Learning)のオープンソースフレームワークであり、主に画像処理分野において利用されている。^[21]コアの部分は C++ で書かれている。CUDA に対応しており、GPU を利用することで高速な処理を可能としている。GPU に比べて処理時間がかかるが、CPU だけで処理するように設定することもできる。

Caffe は Yangqing Jia がカリフォルニア大学のバークレー校(UC Berkeley)の博士課程に在籍していた頃に開発され、現在は UC Berkeley の Berkeley Vision and Learning Center(BVLC)を中心に GitHub 上で開発されている。

Caffe による学習の結果を左右する要素として、教師データの他にネットワーク定義ファイルと solver 定義ファイルの 2 つが挙げられる。それぞれ次の項で詳しく説明していく。

7.2.2 ネットワーク定義ファイル

ネットワーク定義ファイルはどのようなネットワークを使うのかを示すためのファイルである。このファイルを Caffe に与えると、定義されたネットワークを基にモデルの学習や学習済みモデルを用いた分類などを行う。

prototxt を拡張子を持つファイルとして plaintext protocol buffer schema で定義する。layer{...}に囲まれた部分が 1 つの層を表す。この一つ一つの層を接続することによって、Caffe では多層ニューラルネットワークを定義する。

本実験では CIFAR-10^[14]用に作成されたサンプル定義ファイルを使用する。この定義ファイルは CIFAR-10 以外にもさまざまな種類の学習に使用され、その識別器を使用したことにより得られた優秀な認識結果が多数報告されている。

7.2.3 solver 定義ファイル

`solver` はモデルの損失を小さくし、最適化を行う。損失とは、モデルのパフォーマンスの悪さを表す指標である。その `solver` のために用意された `solver` 定義ファイルは、モデルをどのように学習させるのかを設定するためのものである。ネットワーク定義ファイルと同じく、`prototxt` ファイル形式で定義される。各種設定項目の詳細を以下に示す。

- `net`

利用するネットワーク定義ファイルを指定する。

- `test_iter`

評価時に順伝搬処理を何回実行するかを指定する。

- `base_lr`

学習率を計算するときのベースとなる値である。学習率とは 1 回の学習におけるモデルの更新量を調節するパラメータのことである。学習率が大きすぎると学習が収束せず、小さすぎると収束に時間が掛かる。

- `momentum`

損失が極小値に嵌ってしまうのを防ぐために、重みの更新量に前の更新量を加えるが、前の更新量に乗算される計数が `momentum` である。

- `weight_decay`

損失関数の正則化項の重みパラメータである。

- `lr_policy`

学習率の算出方法を指定する。

- `gamma/power`

学習率を計算する際に利用するパラメータである。

- `display`

何回学習するごとに経過を表示するかを指定する。

- `max_iter`

学習回数を指定する。

- `snapshot`

何回学習するごとに学習したモデルパラメータを保存しておくか指定する。

- `solver_mode`

`solver` で処理する際に GPU を利用するかどうかを指定する。

7.3 教師データの一部を対象とした認識実験

7.3.1 概要

Caffe は入力した教師データ画像を用いて学習を行うが、入力したすべてのデータを学習

に使うわけではない。全教師データのうち 90%を学習用データとし、残り 10%はテスト用データとして、学習中の認識率の変化を測定するために使われる。本実験は、実際の楽曲に対する認識に使用する識別器を作成する傍らで、各パラメータをどのように設定した場合が提案手法において最も有効か調査する目的で行う。

7.3.2 予備実験

Caffe では学習時、重みの初期値を毎回乱数で初期化する。そのため同じ設定かつ同じデータを用いたとしても、学習するたびに異なるモデルになる。学習用データとテスト用データを全く同じ組み合わせで 2 回学習を行った場合、また組み合わせを変えて学習を行った場合に認識率に違いはあるのか、検証する予備実験を行った。結果のグラフは割愛するが、どちらの場合も学習経過での認識率の変動には若干違いが見られたものの、最終的に収束した値はほぼ同じという結果になった。この結果から、これ以降の実験では認識率の測定は 1 回のみ行えば良いといえる。

7.3.3 学習率の変更による認識率の変化の調査実験

Caffe で学習に影響を与えるパラメータが多数存在することについては前述した。この実験ではその中でも、一番初めに変更を試してみるべきだと各所で言及されている、学習率に関するパラメータ `base_lr` を小さくする。この変更により認識率がどのように変化するかを調査する。なお、`base_lr` はデフォルトよりも大きくすると学習自体が失敗しやすくなるため、そちらの変更は行わない。

`base_lr` がデフォルトの 0.001 の場合と 0.0001 に変更した場合、それぞれの学習用データに対する `loss` を Fig7.1 に、テスト用データに対する `accuracy` を Fig7.2 にそれぞれ示す。

この結果を見ると、`base_lr` の変更前と変更後で認識率はほとんど同じ値に収束している。このことから、提案手法においては学習率の変更は認識率に影響しないといえる。以降の実験では、`base_lr` の値はデフォルトの 0.001 で固定とする。

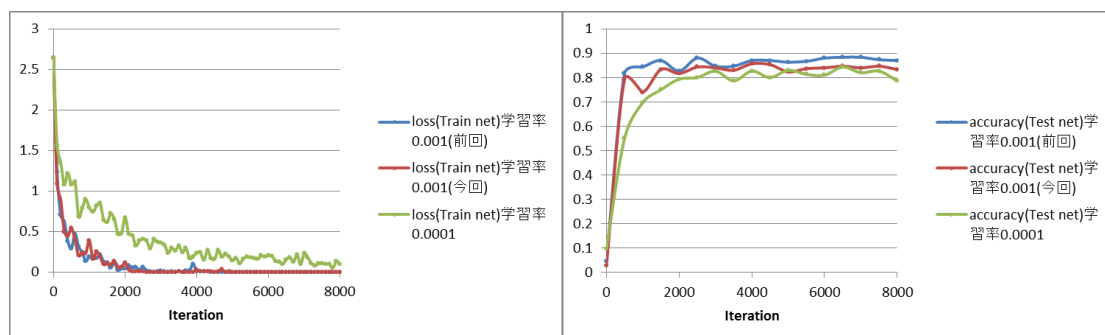


Fig7.1 学習率変更時の学習状況と学習データの `loss` の関係

Fig7.2 学習率変更時の学習状況とテストデータの `accuracy`（正解率）の関係

7.3.4 画像データサイズの変更による認識率の変化の調査実験

Caffe では画像を学習する際、デフォルトでは 32×32 に画像をリサイズしてから学習を行う。しかし、教師データのスペクトログラムは 512×512 を超えるサイズであり、元の $1/256$ に満たないサイズまで小さくして正しく学習が行えるのかという疑問があった。そこで、この実験では学習時の画像のリサイズ設定をデフォルトの 32×32 から 64×64 、 128×128 、 256×256 にそれぞれ変更することで、認識率がどのように変化するかを調査する。

4 種類のリサイズ設定における、それぞれの学習用データに対する loss を Fig7.3 に、テスト用データに対する accuracy を Fig7.4 にそれぞれ示す。

この結果を見ると、リサイズ設定をデフォルトの 32×32 から 64×64 に変更した場合は認識率が向上したが、それ以上の 2 つの設定の場合は逆に認識率が低下していることがわかる。このことから、提案手法においては 64×64 が最も適したリサイズ設定だといえる。学習する画像サイズを 4 倍にすると、それに伴い学習時間もほぼ 4 倍になるという欠点はあるが、以降は基本的に 64×64 の設定で実験を行っていく。

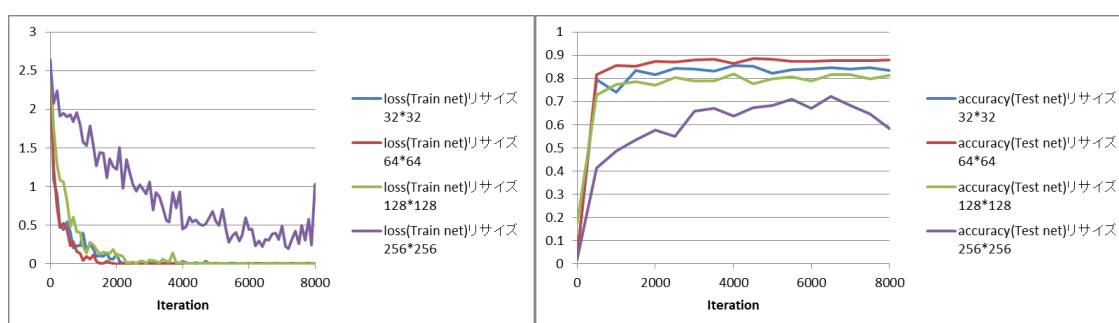


Fig7.3 リサイズ設定変更時の学習状況と学習データの loss の関係

Fig7.4 リサイズ設定変更時の学習状況とテストデータの accuracy（正解率）の関係

7.4 提案手法による実際の楽曲を対象とした認識の実験

7.4.1 概要

本実験では前の項で作成した識別器を使用し、実際の楽曲を対象として認識を行い精度を確認する。

認識対象の楽曲は k 最近傍法の実験と同様、ドラムマイナスイワン楽曲に、ある特定の打楽器音 1 種類を 5 秒毎に 11 ヶ所挿入したものを使用する。挿入する打楽器音は識別する 13 カテゴリから 1 つずつ選択する。各打楽器カテゴリと、それに対応する選択した打楽器の一覧を Table7.1 に示す。

Table7.1 各打楽器カテゴリとそれに対応する選択した打楽器の一覧

打楽器カテゴリ	選択した打楽器
バスドラム	バスドラム
クラッシュ	クラッシュシンバル
ハイハット	ハイハットシンバル
スネアドラム	スネアドラム
タム	ロータム
気鳴	ホイッスル
鈴	鈴
複合	タンバリン
膜鳴	ボンゴ
打奏金属製	トライアングル
擦奏	ギロ
振奏	シェーカー
打奏木製	カスタネット

また k 最近傍法の実験では、調波音が挿入した打楽器音をかき消してしまう問題があったため、今回は音量調整も行っている。このように編集した楽曲を 3 種類のマイナスイオン楽曲×13 種類の打楽器音=39 曲作成し、HPSS を前処理として使用した後、分離後の打楽器成分に対して実験を行う。

7.4.2 実験結果

各打楽器カテゴリそれぞれの、該当の対象楽曲に対する認識率（すべての打楽器挿入点のうち、正しく認識できた数の割合）を Fig7.5 に示す。

これを見ると、タムを除くドラムセット系のカテゴリに関しては、良好な認識結果が得られているといえる。この理由としては k 最近傍法の場合と同様、豊富な学習用データにより十分な学習ができているからだと考えられる。タムに関しては、楽器の材質が膜鳴とほぼ同じなため、この 2 カテゴリ間で誤認識が起こってしまったと考えられる。

それ以外の楽器カテゴリに関しては、あまり良い認識結果は得られなかった。これの全体的な原因の傾向としては、やはり k 最近傍法の場合と同様、十分な数の学習用データがなく学習が中途半端だったことが大きいと思われる。ただし、擦奏や振奏カテゴリに関しては音量調整をしても聞き取りにくい音であるので調波音にかき消されてしまった事も考えられ、また複合カテゴリは楽器の性質上、複数の打楽器が同時に発音した結果誤認識になってしまうということも考えられる。逆に、気鳴楽器は調波音の傾向が強いため、学習用データが少なくても比較的高い認識率になったと考えられる。

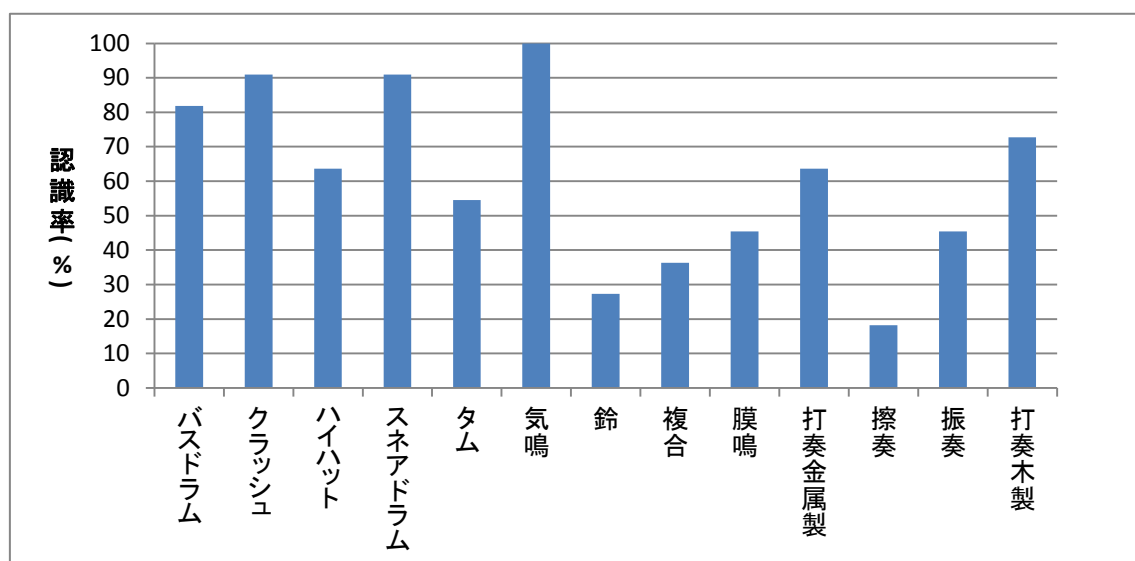


Fig7.5 各打楽器カテゴリそれぞれの該当の対象楽曲に対する認識率

第8章 総括

8.1 まとめ

本論文ではまず、半径探索オプションを利用する k 最近傍法を取り入れた、打楽器を対象とした音源同定手法を提案した。従来手法では対象とする打楽器がドラムセットに限られていたが、それを打楽器全般に拡張して適用できるような改良を行った。

実験を行った結果、ドラムセットの構成楽器に関しては、従来手法にこそ劣るものの概ね良好な認識結果を得ることができた。しかし、それ以外の打楽器については認識率があまり高くなく、課題を残す結果となった。

続いて、上記の提案手法をベースに、認識処理部に深層学習(Deep Learning)によって作成した識別器を使用する音源同定手法を提案した。深層学習を利用することにより、 k 最近傍法による提案手法でネックになっていた楽器ごとのテンプレート数の極端な差の問題を緩和することができる。

実験を行った結果、ドラムセットの構成楽器に関しては、一部の楽器を除いて従来手法とほぼ変わらない良好な認識結果を得ることができた。それ以外の打楽器については相変わらずあまり高い認識率は得られなかったが、 k 最近傍法による提案手法と比較して認識率を改善することは達成した。2つの提案手法の認識結果の比較を Fig8.1 に示す。

これらの結果を踏まえ、「音源同定の対象とする打楽器の種類を拡張する」という目的はある程度果たせたといえる。

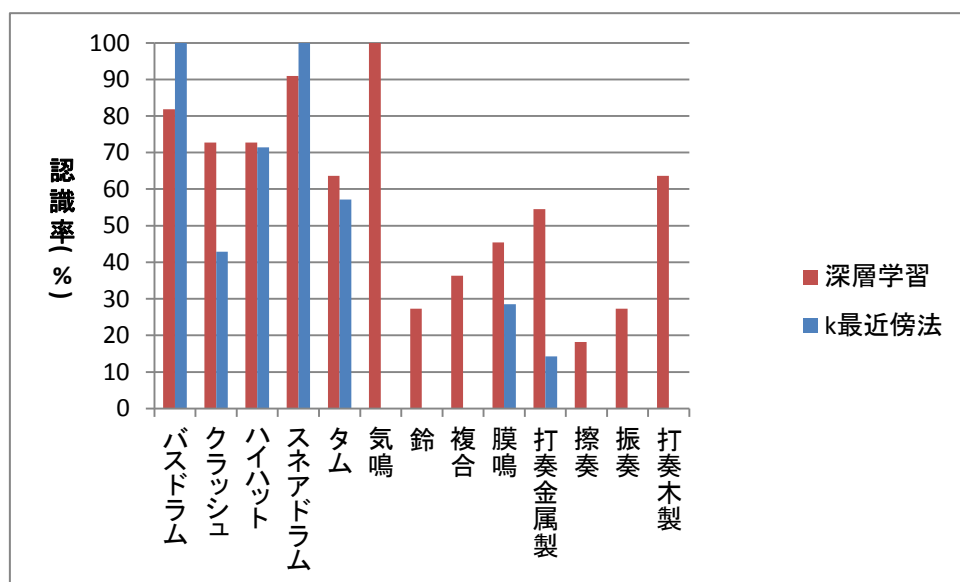


Fig8.1 2つの提案手法の認識結果の比較

8.2 今後の課題

本研究の課題としては、まずドラムセット以外の打楽器の認識率を改善することが第一である。深層学習を取り入れることである程度は改善したが、実用化を考えるとまだ足りず、最低でも 60%程度の認識率は欲しいところである。各パラメータの調整による改善だけでなく、全く別の手法を取り入れることによる根本的な改良も視野に入れて取り組んでいく。

また、深層学習による提案手法では 2（一対他）クラス識別器を使用する手法も提案したが、こちらは非打楽器音区間に対する誤認識の問題が大きく結果を出せる目処が立っていない。複数の打楽器音を認識するためにはこちらの手法は必須であるので、上記の改善に平行して誤認識の問題を解決していきたい。

謝辞

本研究を行うにあたり、日頃よりご指導や大変多くの助言・提案を頂きました甲藤二郎教授に心より御礼申し上げます。

また、研究に関する相談に乗っていただいたり貴重なアドバイスをくださった植村先輩をはじめとする先輩方や研究にご協力いただいた Audio 班の皆様、共に研究に励んだ修士 2 年の同期、並びに日頃よりお世話になりました甲藤研究室の皆様、心より感謝致します。

2016 年 2 月 1 日

大石 皓太郎

参考文献

- [1] 北原 他 “音高による音色変化に着目した楽器音の音源同定：F0 依存多次元正規分布に基づく識別手法” 情処論文誌 Vol.44, No.10, pp.2448-2458, Oct. 2003
- [2] 亀岡 他 “調波時間構造化クラスタリング(HTC)による音楽音響特徴量の同時推定” 情処研究報告 Vol.2005-MUS-61 No.12, pp.71-78, Aug. 2005
- [3] 北原 他 “音響的類似性を反映した楽器の階層表現の獲得とそれに基づく未知楽器のカテゴリレベルの音源同定” 情処論文誌 Vol.45, No.3, pp.680-689, Mar. 2004
- [4] 吉井 他 “自己組織化マップによる教師なしクラスタリングを利用したドラム演奏の自動採譜” 情処会研究報告.2003(82), 43-50, Aug.2003
- [5] 吉井 他 “テンプレート適応を利用した実世界の音楽音響信号に対するドラムスの音源同定” 情処会研究報告.2003(127),55-60,Dec.2003
- [6] “short-time Fourier analysis”
<http://www2.denshi.numazu-ct.ac.jp/staff/jeong/manual/1/1-02.htm>
- [7] “Rectangular window”, “Hanning window”, “Hamming window”
http://www.onosokki.co.jp/HP-WK/c_support/newreport/analyzer/FFT4/fft_12.htm
- [8] “音楽情報処理最前線” <http://www.sigmus.jp/PAPERS/DTMM200902kitahara.pdf>
- [9] “音楽情報処理最前線” <http://www.sigmus.jp/PAPERS/DTMM201004itoyama.pdf>
- [10] “metric space” <http://www.math.ocha.ac.jp/toda/lecture/topsp08/topsp.pdf>
- [11] “Euclidean space”
<http://www.is.titech.ac.jp/~sadayosi/course/past/set09/section2.1.pdf>
- [12] 鈴木武・山田作太郎『数理統計学－基礎から学ぶデータ解析－』内田老鶴圃、2008 年
- [13] 平井有三『はじめてのパターン認識』森北出版、2014 年
- [14] “mahalanobis distance” <http://racco.mikeneko.jp/Kougi/10s/AS/AS08pr.pdf>
- [15] “mahalanobis distance”
<http://www.dna.bio.keio.ac.jp/lecture/postgenome/bi-grad-statis.pdf>
- [16] “mahalanobis distance”
<http://aoki2.si.gunma-u.ac.jp/lecture/Discriminant/mahalanobis.html>
- [17] “Manhattan distance”
http://www.albert2005.co.jp/technology/mining/method3_1.html
- [18] “Cosine similarity”
<http://www.cse.kyoto-su.ac.jp/~g0846020/keywords/cosinSimilarity.html>
- [19] 麻生 英樹、安田 宗樹、前田 新一、岡野原 大輔、岡谷 貴之、久保 陽太郎、ボレガラ ダヌシカ『深層学習－Deep Learning－』近代科学社、2015 年
- [20] 岡谷貴之『機械学習プロフェッショナルシリーズ 深層学習』講談社、2015 年

- [21] 石橋崇司『Caffeをはじめよう』オライリー・ジャパン、2015年
- [22] Nobutaka Ono, Ken-ichi Miyamoto, Hirokazu Kameoka, Shigeki Sagayama, "A Real-time Equalizer of Harmonic and Percussive Components in Music Signals," Proc. of ISMIR, pp.139-144, Sep., 2008.
- [23]"rangesearch"
- <http://jp.mathworks.com/help/stats/classification-using-nearest-neighbors.html>
- [24] "MIRtoolbox"
- <https://www.jyu.fi/hum/laitokset/musiikki/en/research/coe/materials/mirtoolbox>
- [25] 後藤 他 "RWC 研究用音楽データベース: 研究目的で利用可能な著作権処理済み楽曲・楽器音データベース", 情報処理論文誌, Vol.45, No.3, pp.728-738, March 2004.
- [26]"MATLAB" <http://jp.mathworks.com/products/matlab/>
- [27]"Caffe" <http://caffe.berkeleyvision.org/>

発表文献リスト

- [1] 大石皓太郎, 植村あい子, 石倉和将, 甲藤二郎, “未知楽器を含む打楽器を対象とした音源同定”, 電子情報通信学会総合大会, 2014 年 3 月
- [2] 大石皓太郎, 植村あい子, 甲藤二郎, “楽器種類を拡張した打楽器の音源同定”
電子情報通信学会総合大会, 2015 年 3 月
- [3] 大石皓太郎, 植村あい子, 甲藤二郎, “打楽器の音源同定における深層学習を利用した認識率の改善”, 電子情報通信学会総合大会, 2016 年 3 月 (予定)