Picognizer: 電子音の検出および認識のための JavaScript ライブラリ

栗原 一貴* 板谷 あかり* 植村 あい子† 北原 鉄朗†

概要. 我々は今や様々な電子音に囲まれて生活しており，それらを検出・認識し情報処理を行うことは Maker 文化の発展，ディジタルゲーム拡張およびゲーミフィケーションの構成の上で有意義であるが，エンドユーザプログラマが個々の検出・認識対象について教師つき学習により認識器を構築するのは現状ではまだ容易ではない．そこで再生毎の音響的変動が小さいという電子音の特徴を活かし，Dynamic Time Warping 等の伝統的なテンプレートベースのパターンマッチングアルゴリズムにより電子音の検出・認識を行う JavaScript ライブラリを実装する．基礎的な性能評価を行うとともに，ユースケースを例示することでその有用性を示す.

1 はじめに

我々はディジタルゲームの効果音をはじめとして，交通誘導，マナー啓発，広告手段，家電や情報機器の状態通知等の様々な電子音に日々囲まれて生活している．ここで電子音とは計算機等の機械によって再生された，毎回の音響的変動の小さい音と定義する．人間の音声を録音し再生した電子音声も，電子音の一種である．これらを人間へのシグナルとしてだけではなく，電子音検出・認識を通じて計算機システムへの入力として再利用する手段を提供することで，まだ IoT 化されていない生活環境の要素を IoT の世界に接続する事が可能になり，そこを起点として，生活を豊かにする様々なアイディアが実現できるであろう．これは HCI・EC 研究コミュニティ，および近年盛んになっている Maker 文化の発展に貢献が期待できる研究領域である.

本研究では，電子音の中でディジタルゲームの効果音の検出・認識に注目する．効果音の認識を起点にしたディジタルゲーム拡張は，既存ディジタルゲームに新たなエンタテインメント価値を付与したり，非ゲーム的目的，たとえば社会善の達成を実現するような拡張的システム開発を容易にすることができる[1].

ゲーム内の効果音をはじめとする電子音に対象を絞れば，これらを検出する上で高度なパターン認識技術は必ずしも必要ない．再生される音源は基本的に毎回同じであり，音響的特徴の変動が十分に近いと期待できるため，テンプレートベースの古典的な手法でも実用上十分な精度が得られると考えられる．しかし，こういった古典的なテンプレートマッチングによる電子音の検出を Web 上で簡単に実現するためのライブラリは存在しなかった.

本研究では,そのような用途で活用が期待できる，電子音の検出･認識を手軽に実現する JavaScript ライブラリ，Picognizer を提案する．既知の検出対象の音響信号とマイクから得られた音響信号からパワースペクトル等の特徴量を抽出して，Dynamic Time Warping[2]等のマッチングアルゴリズムで求めたコスト（近さ）を比較し，閾値以下であった場合に当該音が検出されたとみなす.「認識」とは複数の候補から尤もらしいものを選ぶことだが，Picognizer においては複数の検出対象とマイク入力音とのコストを算出し，それらを比較選別すればよい．以後は記述を簡便にするため，Picognizer の機能について言及する際,「検出・認識」ではなく「検出」と表記する.

検出判定後はたとえば myThings[3]や Sony MESH[4]などのインターネットサービス・IoT デバイス等へ通知することができ，多様な応用が可能である（図 1）．ライブラリは JavaScript のみで記述されているため，Web ブラウザがあればインストール不要で動作する.

本論文では，プロトタイプの実装，および音源の単純さとゲーミフィケーション構成への応用可能性

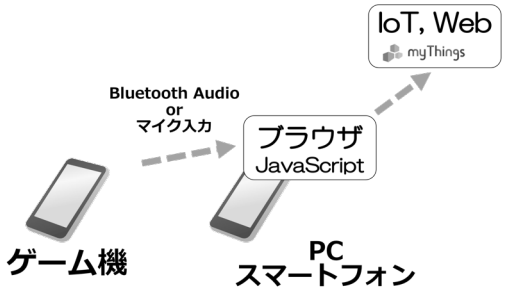

図 1. Picognizer の概要

Copyright is held by the author(s).

* 津田塾大学, † 日本大学

のバランスがよい検出対象として，任天堂ファミリーコンピュータのようないわゆる 8bit 音源を対象にした認識器の評価実験について報告する．

なお，本稿は ACE'17 で発表予定の論文[5]を再編成したものである．

2 関連研究

2.1 音響イベント検出

音声や音楽以外の音の認識は，従来は環境音認識 (environmental sound recognition) という名称で進められてきたが，近年は，音響イベント検出（audio event detection / sound event detection）と呼ばれ，様々な研究が進められている[6,7,8,9,10]．また，他の関連分野と同様に，深層学習を用いた研究も増えつつある[11,12]．

2.2 Web 上の音の検出・認識プラットフォーム

一方，音声認識に関しては，近年 Web 上の API が数多く公開され，これらを活用して音声認識対応の Web アプリケーションなどを簡単に作成できる状況が確立されつつある[13,14,15]．これらは，大規模な学習データと最新の機械学習技術を用いて一般の人の音声を認識することを目的としており，特定の音の検出を目的としているわけではない．

また，Shazam [16] のように，スマートフォンのマイク越しに音楽を入力すると，その曲名やアーティスト名を答えてくれるサービスも登場している．

2.3 音の検出・認識機能の IoT デバイスへの適用

音の検出や認識機能を組み込んだ IoT デバイスも登場している．著名なものとして，Amazon Echo [17]，Listnr [18]などが挙げられる．これらは生活環境に置かれた IoT マイクデバイスが常時周囲の音を取得し，クラウド型の認識エンジンを経て，結果に応じた IoT 機器の連携等を行うものである．主に音声認識を想定したアプリケーションが提案されているが，ユーザが独自の認識エンジンを追加することにより，様々な応用が可能なオープンシステムとなっている．本研究の提案システムも，追加の認識エンジンとして導入しこれらの機器を活用することは可能であり，共存できるものと考えられる．

MagicKnock [19]もハードウェア構成としては IoT マイクデバイスと同様だが，置かれた面へのユーザのノック動作の振動パターンを集音，認識し，IoT 機器への入力として扱うシステムである．

2.4 プログラミング環境

Sikuli[20]は，マウスで特定のアイコンをクリックする，などの GUI オートメーションを実現する Python プログラミング環境である．アイコン画像などをスクリーンショット機能で撮影し，それをプログラムコードに直接貼り付けることで,「この画像を対象にする」という指示が行える．実行時は画像のテンプレートマッチングにより照合が行われる．音声を扱う本研究とは，映像を扱っている点で相違点があるが，既存システムを改変することなく，外付けすることで機能拡張するという点で共通点があり，Sikuli と本研究の提案システムを併用することで映像と音響を併用した既存システム拡張を行うことが可能となる．

加藤らは Sikuli を発展させ，人間の身体動作やロボットの姿勢等のスナップショットデータをマッチング対象とする拡張を行っている[21]．

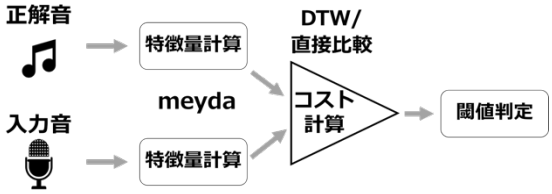

図 2. Picognizer の処理フロー

3 Picognizer

3.1 実装

Picognizer は JavaScript のみで記述された簡易な電子音検出ライブラリである．getUserMedia によりマイク入力を扱える Web ブラウザで動作する．PC では Mac および Windows の Firefox と Chrome で，スマートフォンでは Android の Firefox で動作を確認している．

Picognizer の処理フローは図 2 のようになっている．検出対象となる「正解」の電子音源を wav 形式または mp3 形式で入力すると，音響特徴量計算ライブラリである Meyda[22]によりパワースペクトル，MFCC 等，17 種類の特徴量の組み合わせからなる特徴量ベクトルが抽出される．その際，正規化やローパスフィルタなどの一般的な処理も行われる．また, 検出開始するとマイク入力から音声を取得し，同様に特徴量ベクトルに変換される．

正解の特徴量ベクトルと入力の特徴量ベクトルは Dynamic Time Warping 法，および直接比較法によって比較され，コストが計算される． 直接比較法とは，入力音声と正解について，Dynamic Time Warping のように伸縮を考慮せず，ナイーブにフレームごとに比較しコストを算出する手法である．コストが閾値以下であれば，検出されたとみなして事後処理を行う．

なお，Dynamic Time Warping 法，直接比較法のいずれにおいても，各フレームにおける特徴量ベクトル間の距離関数を定義する必要がある．電子音検出では，BGM などの影響で「正解の音声が含まれ

ているが他の音も鳴っている」という状況も検出対象としたいため，次式のような非対称な距離関数 *Asym_Dist* を定義して用いた．

$$Asym\_Dist(target, input) = Dist(target, Mask(target,input))$$

ここで，*target* は正解音の特徴ベクトル，*input* は入力音声の特徴ベクトル，$Dist(\boldsymbol{A},\boldsymbol{B})$はベクトル $\boldsymbol{A}$，ベクトル $\boldsymbol{B}$ の間のユークリッド距離を表し，$Mask(\boldsymbol{A},\boldsymbol{B})$はベクトル $\boldsymbol{A}$ とベクトル $\boldsymbol{B}$ の要素ごとの最小値からなるベクトルである．すなわちその i 番目の要素 $Mask_i(\boldsymbol{A},\boldsymbol{B})$は以下のように定義される．

$$Mask_i(\boldsymbol{A},\boldsymbol{B}) = min(\boldsymbol{A}_i,\boldsymbol{B}_i)$$

表 1．ブラウザ版 Picognizer に指定するパラメータ．

パラメータ	説明
src	認識対象音源ファイルのURL (.wav or .mp3)
st,et	認識対象音源中で認識に使う箇所の開始時刻と終了時刻
surl	認識時に実行されるJavaScriptファイルのURL
cri	認識判定を行うコストのしきい値
mode	コスト計算アルゴリズム (“dtw” :DTW or “direct” :直接比較)
wfunc	窓関数(default: "hamming")
ft	Meydaで抽出する特徴量名の配列 (default: [“powerSpectrum”])
frame	特徴量抽出の周期 (default: 0.02)
dur	コスト計算の周期 (default: 1.0)

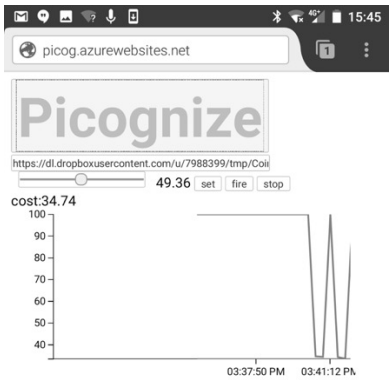

図 3 ブラウザ版の Picognizer の画面

3.2 使用方法

3.2.1 Web サイトによる使用

以下に我々が準備した Web サイトでの Picognizer の使用方法を示す．

Picognizer はインストール不要であり，以下のように URL にアクセスするだけで活用可能である．

https://qurihara.github.io/picognizer/script.html?cri=10&surl=http://xxx.net/bg_red.js&src=http://zzz.com/target.mp3

表 1 に，クエリ文字列として指定可能なパラメータの一覧を示す．この例では，src パラメータで指定した音源ファイルを正解とし，cri パラメータをコストの閾値とし，コストが閾値を下回った時に実行する JavaScript ファイルを surl パラメータで指定している．図 3 にこの URL にアクセスした時の画面の様子を示す．「Picognize」ボタンをクリックもしくはタップし，ブラウザにマイク利用の許可を与えると動作が開始される．画面下部にはコストの時系列と閾値がグラフ化されており，スライドバーで適切なレベルに閾値を調整可能である．

以下は検出時に画面の背景を赤く変化させるスクリプト bg_red.js の内容である．

```
setup = function(){
  console.log('Initialized.');
}
onfire = function(){
  document.bgColor = 'red';
}
```

ここで，setup 関数はサイトのロード時に実行され，onfire 関数は検出時に実行される．このように Picognizer は構成した電子音検出に関する全ての情報を URL に組み込めるため，保存および第三者との共有が容易である．

任意の JavaScript を実行可能であることは汎用的である反面，セキュリティ上の懸念がある．そこで，用途をより限定することでセキュリティ対策を図る使用法を開発した．Picognizer の主な活用方法は，何らかの電子音検出を起点として多様な IoT 機器，ネット上のサービスと接続することであろう．その際は検出時に MQTT メッセージブローカである Meshblu サーバへのトリガー情報の送信機能のみをもち，surl パラメータを持たない以下のような URL を使用する．

https://qurihara.github.io/picognizer/trigger.html?cri=33&myuuid=XXXXXX&mytoken=YYYYY&server=192.168.0.1&src=http://zzz.com/target.mp3

のような記法で URL を指定する．server, myuuid, mytoken は Meshblu に接続するためのアカウント情報を指定しているクエリ文字列である．これにより，同じ Meshblu サーバに接続している各種 IoT 機器との連携，および様々なネット上のサービス同士の連携を容易に行うことができる myThings との接続が可能である．

3.2.2 直接的なコーディング

前節による手法ではなく，直接的に JavaScript をコーディングすることも可能である．特徴量パラメータの設定，複数音源の同時検出（すなわち認識）など，より詳細な挙動の制御が可能である．以下はそのコード例である．

```
var Pico = require('picognizer');
var P = new Pico; //インスタンス化
var option = { //検出パラメータ
    windowFunc: 'hamming',
```

```
    mode: 'direct',
    feature:[ 'mfcc'],
    framesec:0.1,
    duration:1.0
};
P.init(option); //初期化
P.oncost(['audio1.mp3','audio2.mp3'],
function(cost){
        //コスト算出時のイベントハンドラ
});
```

4 基礎性能評価

本論文では研究の初期段階としていわゆる 8bit 音源のゲームの代表例である「スーパーマリオブラザーズ」の音源を用いて評価実験を行った．8bit 音源のレトロゲームは音源が単純であり検出が容易でありながら，余剰自由度[1]（外部要素を導入できる余地）が大きく，ゲーミフィケーション構成が比較的容易である特徴を持つため，実用的な対象である．

4.1 テストデータセット

本実験では以下の 2 つの実験を行った．

1. BGM なしで対象の効果音を検出する．
2. BGM が付加された状態で対象の効果音を検出する．

各実験においてテスト用データセットは次のように構築した．検出対象には，コイン取得音（coin），ジャンプ音（jump），1up きのこ取得音（1up）の 3 種の効果音を選択した．35 秒のサンプリング周波数 22050Hz のモノラル音源を生成し，3 種の効果音のうち 1 種がランダムな箇所に，少なくとも 5 回以上含まれるようにした．また，検出性能低下を人為的に引き起こすために，スーパーマリオブラザーズの 9 種類の効果音から，「非対象音」として無作為に選び，音源に繰り返し混合した．複数の効果音の重なりはない．この実験 1 のデータに対し，3 種の BGM のうち 1 種を 0dB，5dB の SN 比で付加し，それぞれを実験 2 用とした．なお BGM には，地上ステージ（outdoor），地下ステージ（underground），マリオが無敵になるときの BGM（star）の 3 種を選択した．そして，評価ツール用に，各入力音源の検出対象音と非対象音全てについて，オンセット（再生開始）時刻のラベル付けを全て手動で行った．

4.2 評価方法

各実験において，コスト計算アルゴリズムに関し，DTW と直接比較法の 2 通りを用いて比較した．我々のインフォーマルな予備実験において，様々な音響特徴量の中でパワースペクトルがスーパーマリオブラザーズの効果音に対して最もよい検出性能だったため，音響特徴量にはパワースペクトルのみを使用した． 各実験では，パワースペクトルを 40ms ごとに音響特徴ベクトルとして抽出し，検出対象の効果音と入力音とのコストを算出する．再現率，適合率，F 値は Python の評価ツール mir_eval[23]を用いて算出した．各効果音で検出されたオンセット時刻の許容誤差は，正解のオンセット時刻から±50ms である．

なお，手動で行う閾値の設定については，各実験で最も高い再現率を得られるように定めた．この設定により誤検出は増えるが，ディジタルゲーム拡張の応用を考えた場合，画像照合を扱う Sikuli などの他の技術と Picognizer を統合し，適合率を高めることが可能であると考えたからである．

表 2 実験 1・実験 2 における DTW の結果

検出対象	BGM	S/N [dB]	F値	適合率	再現率
coin	outdoor	clean	0.750	0.750	0.750
		5	0.625	0.625	0.625
		0	0.400	0.429	0.375
jump	underground	clean	0.545	1.000	0.375
		5	0.400	1.000	0.250
		0	0.222	1.000	0.125
1up	star	clean	0.444	0.500	0.400
		5	0.444	0.500	0.400
		0	0.100	0.067	0.200

表 3 実験 1・実験 2 における直接比較法の結果

検出対象	BGM	S/N [dB]	F値	適合率	再現率
coin	outdoor	clean	1.000	1.000	1.000
		5	1.000	1.000	1.000
		0	0.933	1.000	0.875
jump	underground	clean	0.941	0.889	1.000
		5	0.889	0.800	1.000
		0	0.889	0.800	1.000
1up	star	clean	1.000	1.000	1.000
		5	1.000	1.000	1.000
		0	1.000	1.000	1.000

4.3 評価結果と考察

DTW を用いた結果を表 2 に，直接比較法を用いた結果を表 3 に示す．表中の S / N [dB]について，「clean」は実験 1 の結果に対応し，5dB と 0dB は実験 2 の結果に対応する．

実験を通して直接比較法は，F 値が多くの条件下で 1.0 となり，最小でも 0.889 であったため，実用的な検出性能を持っていると考えられる．再生毎の音響的変動の小さい電子音の性質に直接比較法が適合し高性能が得られたと推測される．しかし，直接比較法は検出対象音の再生タイミングとコスト計算開始タイミングのずれ，あるいは計算機の処理能力不足によるフレーム欠損があった場合にそれを考慮することができないため，使用にあたっては今日市販されている通常性能のノートブック PC 程度あるいはそれ以上の計算速度を持つマシンを採用し，表

1 の「frame」と「dur」パラメータをより小さく設定する必要がある．これは，そのずれを小さくすべく，頻繁に特徴量抽出とコスト計算を行う設定である．

一方，DTW は直接比較よりも 1 回のコスト計算に多くの計算量を必要とするが，先述のような時間的ずれやフレーム欠損に対して一定のロバスト性を備えている．よってユーザは，直接比較法で膨大な計算をする能力がないスマートフォンなどの低速機を活用する際には，「frame」と「dur」パラメータを大きな値に調整して，DTW を使用することで平均計算量を削減することができる可能性がある．ただしこの場合レイテンシ（検出までの時間の遅れ）は増加する.マシンパワーに基づくこのようなパラメータ最適化戦略の定量的評価は，今後の課題である．

5　応用シナリオ例

本章では Picognizer をディジタルゲーム音検出に適用した場合の応用シナリオ例について述べる．デモ映像 (https://youtu.be/-2MtArZAtfg) には記載例以外の事例も収録されているので参照されたい．

図 4 は，スーパーマリオブラザーズのプレイ中にコインを取得した際，コイン取得の効果音を Picognizer で認識し，実世界で硬貨を 1 枚排出するシステムの例である．硬貨の排出は，Sony MESH の GPIO タグに接続したサーボモータを操作することによって実現している．硬貨の排出には，第三者が用意した硬貨であれば賞金，ユーザ自身が用意した硬貨であれば課金もしくは罰金の意味合いを付与可能である．栗原は[1]において，スーパーマリオブラザーズにおけるコイン取得を慈善団体への web 上での寄付行為に連動させることで，寄付行為に対する参加障壁を下げる社会的目的を達成する Coins For Two というシステムを提案したが，本例はその実世界版の実装である．イベント会場などで来場者に課金の上でゲームをプレイしてもらい，排出されたコインを募金箱に収納することで運用する．

本例のように，既存ゲームの拡張を行う際，新たに付与される要素が単純なエンタテインメント価値の増強ではなく，何らかの社会的目的の達成に寄与する場合，それを栗原は Toolification of Games と定義している．これはゲーミフィケーションの派生概念であり，Picognizer は宿主となる既存ゲームのソースコードを入手することなく Toolification of Games の構築を簡便に行える開発基盤と言える．

図 5 は，Nintendo Switch のゲーム「1-2-Switch」における，ガンマン（QuickDraw）というミニゲームを自動操作するシステムである．これは「Fire」という掛け声を聞いた後に対戦相手より早くコントローラを水平にしてボタンを押すことで，勝利となる．本システムは，「Fire」の掛け声を Picognizer で検出し，IoT ステッピングモータの Webmo を用いてゲームコントローラを回転させるものである．筐体とボタン押下用の突起をブロック玩具で作成した．本例では音声のみを手がかりに進行するゲームを扱ったが，画像照合による GUI オートメーション開発環境である Sikuli と組み合わせることで，より複雑なゲームの自動操作が可能であるだろう．

図 4 スーパーマリオブラザーズを用いた Toolification of Games 事例，「Coins For Two」．コイン取得時の効果音を検出し，コインサーバーから硬貨が排出される．

図 5 QuickDraw の自動操作の例．「Fire」の掛け声が検出されると Nintendo Switch のコントローラが IoT モータ Webmo によって回転し，銃を発射する入力が行われる．

6　課題と今後の展望

6.1　パラメータ設定支援

Picognizer は伝統的なテンプレートマッチング手法により，正解とする音響データと入力信号の間のコスト（近さ）を算出するところまでが機械的に行われ，コストがどの値以下であれば「検出」と判断するかは，ユーザの手作業により設定される．現在はスライドバーとコストのグラフ表示による閾値 UI が提供されているが，よりきめ細かい支援が可能である．たとえば準備作業として正解音響データの発生時と非発生時を実施もしくはエミュレートして，おおよそのコストの変動幅を見積もり，仮の閾値を推薦することなどである．

また，音響学に詳しくないユーザのために，対象となる音響データの性質と使用する計算機の性能，

およびニーズにもとづき，特徴量や検出パラメータを推薦するウィザードの実装も有意義であろう．すなわち，人の声なのか，ディジタルゲームの効果音なのか，計算負荷を上げてでも再現率を上げたいのか，非力な計算機で扱いたいのか，などを対話により取得することである．これらは今後の課題である．

6.2 より複雑な音響特性をもつ電子音への拡張

本論文では，認識対象として任天堂ファミリーコンピュータのようないわゆる 8bit 音源を対象にし，環境音ノイズのない状況下での評価について報告した．電子音といっても対象は多様であり，その全てに対し高性能の検出器を構成するのは容易ではない．研究の第一歩として，音源の単純さとゲーミフィケーション構築への応用可能性のバランスがよい対象として，これを選定した．より複雑・高度な音源への適用も現状のプロトタイプである程度可能であるが，それらへのチューニングや性能検証は今後の課題である．

6.3 教師付き学習に向けて

本研究では，ユーザの身近にある電子音について，教師付き学習を用いずテンプレートマッチする手法について扱ったが，構築したブラウザ版 Picognizer が広く用いられるようになれば，ユーザにとって関心のある様々な電子音の検出に関するデータをサーバ側に蓄積可能である．これに深層学習等の教師付き学習技術を用いることにより，より汎用の電子音検出・認識サービスを構築できる可能性がある．

謝辞

本研究は JSPS 科研費 JP15H02735, JP16H02867, および JP17H00749 の助成を受けた．

参考文献

[1] 栗原一貴: Toolification of Games:既存ゲームの余剰自由度の中で非ゲーム的目的を達成するゲーミフィケーション周辺概念の提案と検討," 情報処理学会論文誌, Vol. 58, No.14, pp.1-13 (2017).

[2] Sakoe et al.: Dynamic Programming Algorithm Optimization for Spoken Word Recognition, IEEE Trans. on Acoustics, Speech, and Signal Processing, Vol.26, No.1, pp.43-49 (1978).

[3] myThings, https://mythings.yahoo.co.jp/, last accessed 2017/07/28.

[4] Sony MESH, http://meshprj.com/, last accessed 2017/07/28.

[5] Kurihara et al. : Picognizer: A JavaScript Library for Detecting and Recognizing Synthesized Sounds, Proc. of ACE'17, (2017) in printing.

[6] Cai et al.: Highlight Sound Effects Detection in Audio Stream, Proc. of ICME'03, vol. III, pp.37-40 (2003).

[7] Pikrakis et al.: Gunshot Detection in Audio Streams from Movies by Means of Dynamic Programming and Bayesian Networks, Proc. of ICASSP'08, pp.21-24 (2008).

[8] Harma et al.: Automatic Surveillance of the Acoustic Activity in Our Living Environment, Proc. of ICME'05, (2005).

[9] Imoto et al.: Acoustic Event Analysis based on Hierarchical Generative Model of Acoustic Event Sequence, IEICE Transactions on Information and Systems, Vol. E990D, No.10, pp.2539-2439 (2016).

[10] Lu et al.: Sparse Representation based on a Bag of Spectral Exemplars for Acoustic Event Detection, Proc. of ICASSP'14, pp.6255-6259 (2014).

[11] Wang et al.: Audio-based Multimedia Event Detection Using Deep Recurrent Neural Networks, Proc. of ICASSP 2016, (2016).

[12] Cakir et al.: Convolutional Recurrent Neural Networks for Polyphonic Sound Event Detection, IEEE/ACM Trans. on Audio, Speech, and Language Processing, Vol.25, No.6, pp.1291-1303 (2017).

[13] Bing Speech API, https://azure.microsoft.com/ja-jp/services/cognitive-services/speech/, last accessed 2017/07/28.

[14] Google Cloud Speech API, https://cloud.google.com/speech/, last accessed 2017/07/28.

[15] IBM Watson Speech to Text, https://www.ibm.com/watson/developercloud/speech-to-text.html, last accessed 2017/07/28.

[16] Shazam, https://www.shazam.com/, last accessed 2017/07/28.

[17] Amazon Echo, https://www.amazon.com/dp/B00X4WHP5E, last accessed 2017/07/28.

[18] Listnr, https://listnr.cerevo.com/ja/, last accessed 2017/07/28.

[19] MagicKnock, http://magicknock.com/, last accessed 2017/07/28.

[20] Tom et al.: Using GUI Screenshots for Search and Automation, Proc. of UIST'09, pp.183-192 (2009).

[21] Kato et al.: Picode: Inline Photos Representing Posture Data in Source Code, Proc. of CHI'13, pp.3097-3100 (2013).

[22] Rawlinson et al.: an Audio Feature Extraction Library for the Web Audio API. In The 1st Web Audio Conference (WAC). Paris, France (2015).

[23] Raffel et al.: mir_eval: A Transparent Implementation of Common MIR Metrics, Proc. of MIR'14, (2014).