2019 年 2 月 5 日

令和元年度　情報・ネットワーク工学専攻　コンピュータサイエンスプログラム　修士論文発表会

Android 端末における GPU を利用した複数音源分離の比較検討

コンピュータサイエンスプログラム　学籍番号:1831081　成見研究室　杉田陽亮

1　はじめに

スマートフォンは人々に広く普及し多くの場面で利用されており, 基本的に System on a Chip(SoC) と呼ばれるチップが搭載されている. SoC には CPU に加えて Graphics Processing Unit(GPU) などが集約されており, その性能が向上し続けている. また, GPU の高い計算能力を活かして様々な用途で利用する General Purpose computing on GPU(GPGPU) という分野が発展を続けている.

他方, サブスクリプション型のストリーミング配信サービスやバーチャルリアリティ空間におけるライブに代表されるように, 音楽の楽しみ方が多様化している. 今後も同様に音楽を用いたサービスは多く登場すると考えられ, よりよい音楽体験が求められる. しかし, Wi-Fi など無線通信の環境外における高速通信量は個人の契約に依存して有限であり, 通信コストのかからないサービスが求められる.

これらを踏まえ, 本研究では音楽体験として混合音源からそれぞれの音源要素を取り出す複数音源分離を取り扱い, Android 端末上での実装を行う. 処理内部で GPU を利用し, 端末上での高速な処理の実現を目指す. また, GPU 利用の有無による性能を比較し, 有効性を検討する.

加えて, 音源分離を用いた音楽体験の一例として, 4 つの端末上にてそれぞれ 3D モデルを動かしながら音楽を流すことでライブを再現するシステムを提案する.

2　音源分離

音源分離とは複数の音源から構成される混合音源から特定の音源を復元する処理である. 一般的には話者分離等にも用いられるが, 本研究では音楽への適用, 特にバンド楽曲を構成する音源の分離に着目する.

深層学習を利用した音源分離は近年盛んに行われており, その高い分離精度が注目されている. 短時間フーリエ変換 (Short Term Fourier Transform : STFT) を適用することにより, 音源は周波数, 時間, 強さの 3 次元形式へ変換できる. これにより, 同じく 3 次元で構成される画像とほぼ同様の処理が可能となり, 画像処理の分野で高い成果を残したモデルを音源分離へ適用する研究が多く行われている.

代表的な手法として, U-Net 構造 [1] をスペクトログラムを用いて音源分離に適用したものが挙げられる [2]. U-Net は医療分野の画像処理において提案されたモデルであり, スキップ構造を用いることで解像度の高い処理が可能となる. さらに, 2 次元の畳み込みとその逆処理を行う層で構成されており, 単純かつ GPU による高速化が見込める.

本研究ではこの U-Net 構造を利用した深層学習による音源分離の実装を扱う.

3　Android 端末における深層学習推論処理

Android 端末上で深層学習モデルの構築・学習を行うことは計算資源の不足により現実的ではない. そのため事前に用意したモデルを変換し, 推論処理のみを行うことを考える.

TensorFlow は Google 社が主となって開発されているオープンソースの深層学習プラットフォームである [3]. TensorFlow で作成したモデルを Converter にて専用モデルへ変換し, TensorFlow Lite を利用することで Android 上での推論処理が可能となる. また, delegate という機能を利用することで GPU や Neural Network API(NNAPI) 等アクセラレータを選択できる.

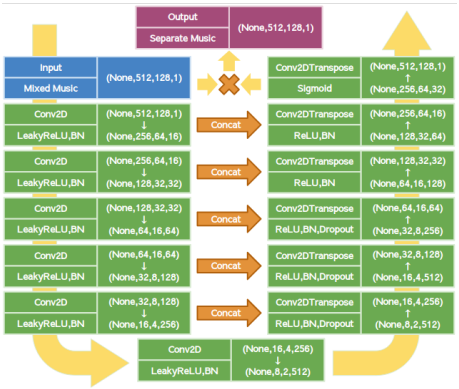

図 1: 深層学習モデル概要

4　既存研究

土橋らは音源分離を利用したタブレット演奏支援アプリを作成した [4]. 本研究とは音源分離を用いる点で共通しているが, 分離処理を事前に行い結果のみを格納しており, 端末上で分離処理を行うという点で異なる.

Miu らは TensorFlow Lite を利用した物体の姿勢推定 Unity アプリを作成した [5]. TensorFlow Lite を利用しスマートフォンでの推論処理を行う点で共通しているが, 推論処理の適用対象が異なる.

5　Android 端末における音源分離の実装

5.1　短時間フーリエ変換による音源処理

モデルの学習と推論にはスペクトログラムを利用した. この際, STFT によって音源をスペクトログラムへ変換した. STFT にはハミング窓を窓関数として利用し, 窓幅は 1024, オーバーラップはその半分とした. この結果の絶対値を取ることで, スペクトログラムを得た.

推論処理を行うことで, 分離音源のスペクトログラムが得られる. STFT の結果の偏角と分離音源スペクトログラムを掛け合わせて複素数へ変換し, 逆短時間フーリエ変換 (inverse STFT : iSTFT) によって分離音源が取得できる.

5.2　深層学習モデルの構築と学習・変換

TensorFlow2.0 における標準的なモデル構築 API である Keras を利用して, 図 1 のモデルを構築した. 学習には MUSDB18[6] を利用し, 4 種の分離対象ごとにモデルを作成した. 作成したモデルは TFLite Converter を利用し TensorFlow Lite 形式のモデルへ変換した. 変換時には通常の単精度 (float32) モデルに加え, 半精度 (float16) モデルへの変換などのオプションも用いたモデルも作成した.

5.3　TensorFlow Lite による推論処理

音源分離アプリは Andoird Studio を利用して作成した. 利用ライブラリに TensorFlow Lite とその GPU 版を追加し, OpenCL の利用時は 2.1.0, OpenGL ES の利用時は 0.0.0-nightly のバージョンを指定した. また, プロジェクト内へ変換後モデルを格納した. Interpreter クラスを利用してモデルを読み込み, 入出力に ByteBuffer を取ることで実行した. Interpreter インスタンス作成時には delegate オプションを指定し GPU や NNAPI の利用を明示した. モデルの推論結果をスペクトログラムとして解釈し, iSTFT をかけることで分離音源を得た.

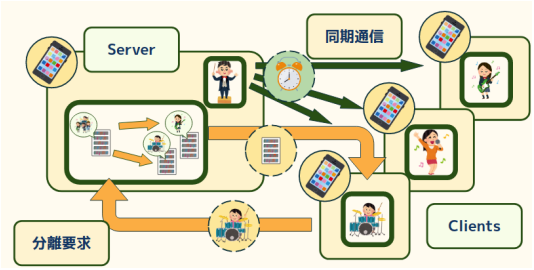

図 2: ライブシステム概要図

表 1: タブレット端末 計算速度結果

	fft time	data	separate	other	sum
CPU	37343	47233	56808	2459	143843
CPU, Half	36516	46997	60740	2579	146832
NN	37156	46743	56687	3858	144444
NN, Half	38080	48127	60545	3808	150560
GPUv2	37644	52456	17725	17726	125551
GPUv1	37776	47763	72538	3928	162004
GPUv1 Half	36377	46949	65271	3929	152526

6 3D モデルライブシステム

端末上での音源分離を利用したアプリケーションとして, 3D モデルを利用したライブ再現システムを提案する. 処理概要を図 2 として示す. サーバ端末が音源分離を行い, クライアント端末が分離音源を受け取り, 再生と同時に楽器演奏モーションを行う.

音源分離は前章で実装した機能をライブラリとして変換し, Unity から呼び出す形とした. 端末間通信には Nearby[7] を利用することで, 外部通信なく端末間ネットワークを構築した.

同期再生には Network Time Protocol(NTP) を基準とした再生時間指定を行い, 規定時間から同時に再生を行う形とした. 3D モデルによる演奏は分離音源の音量がしきい値を超えた場合にモーションを行わせた.

7 評価

まず, PC から Android への深層学習モデル移植にて差異が発生していないかをスペクトログラムにて確認し, 同等の特徴を捉えた分離処理が行えていることを確認した.

次に, 200 秒のモノラル音源を用いて音源分離の処理速度を計測した. 利用端末は MediaPad T2 8 Pro(タブレット端末), Galaxy S8(Galaxy), Pixel 3(Pixel) の 3 台を利用した. また, CPU, GPU(v2 : OpenCL, v1 : OpenGL ES), NNAPI の利用と単精度, 半精度モデルを組み合わせ, 合計 7 つの条件下で測定を行った.

計測結果を端末ごとにそれぞれ表 1, 2, 3 として記した. 表中では処理時間をミリ秒で表した. その結果, 全ての条件下で 200 秒以下で処理が完了し, さらに GPU を利用することで CPU による処理時間と比較し最高約 2.8 倍の高速化を観測した.

最後に Unity から分離処理を同条件で呼び出した際の処理時間を計測した. この際, メモリの関係上処理を一部変更した. また, 計測条件は基準となる CPU と最も高速であった GPUv2 を採用した. 2 つの計測条件を Android アプリ上 (Native) と Unity 上でそれぞれ実行し, 計算時間を測定した. この結果を表 4 に示した. 表中では処理時間をミリ秒で表した.

Unity 利用時は処理時間が増加したが, 増加後も 200 秒以下での処理が実現されている. これにより, Unity に向けた音源分離ライブラリにおいても十分な速度で処理を行うことができることを確認した. また Native での処理時間に着目すると, 処理順の変更により GPU 利用によって CPU 比で最高約 3.2 倍の高速化を観測した.

7.1 各条件における考察

本研究においての推論処理時間は OpenCL による GPU を利用した場合において最も高速となった. ただし, メモリの転送時間が処理時間の多くを占めており, 分離そのものの時間で比較を行うとより高速である. また, 多くの端末での実行を考慮する場合は多少の速度減少が生じる OpenGL ES 版の利用が必要である.

表 2: Galaxy 計算速度結果

	fft time	data	separate	other	sum
CPU	1760	1909	22149	477	26295
CPU, Half	1758	1918	20018	474	24168
NN	2050	2640	33671	929	39289
NN, Half	1880	2594	27530	870	32873
GPUv2	1789	6527	2358	2562	13234
GPUv1	1794	1909	8639	886	13229
GPUv1, Half	1759	1916	22717	833	27225

表 3: Pixel 計算速度結果

	fft time	data	separate	other	sum
CPU	1871	1943	18118	473	22404
CPU, Half	1873	1962	17303	462	21599
NN	1817	2231	115703	915	120667
NN, Half	1692	2376	123946	892	128904
GPUv2	1477	3082	2284	988	7830
GPUv1	1482	1543	10532	712	14269
GPUv1, Half	1508	1561	17220	813	21102

表 4: Native と Unity における計算速度結果

	Native CPU	Unity CPU	Native GPU	Unity GPU
Tablet	112338	159991	93939	99022
Galaxy	26393	33390	8307	17694
Pixel	17651	23710	9082	11054

半精度モデルによる最適化は必ずしも高速な処理に繋がらない. しかし, モデルサイズは 2/3 と小さく抑えられるため, アプリサイズを削減する用途には有効であると言える.

NNAPI による高速化は全く効果を示さなかった. だが, 画像処理において NNAPI は重みを更に削減した量子化 (int8) モデルなどの利用時に効果を発揮する. 精度を落とした状態で音源分離へ適用が可能かどうかの検討は今後の課題である.

8 まとめと今後の課題

Android 端末上での複数音源分離を様々な手法を比較しながら実装した. また, 音源分離の利用例として 3D モデルライブシステムを提案した.

今後の課題として, 音源分離処理の最適化, 異なる音源分離深層学習モデルの検討, リアルタイムな音源分離再生機能の実装, 3D モデルの演奏感向上などが挙げられる.

参考文献

[1] Ronneberger, O., Fischer, P. and Brox, T.: U-Net: Convolutional Networks for Biomedical Image Segmentation, *CoRR*, Vol. abs/1505.04597 (2015).

[2] Jansson, A., Humphrey, E. J., Montecchio, N., Bittner, R. M., Kumar, A. and Weyde, T.: Singing Voice Separation with Deep U-Net Convolutional Networks., *Proceedings of the 18th International Society for Music Information Retrieval Conference*, Suzhou, China, ISMIR, pp. 745–751 (2017).

[3] Abadi, M., Agarwal, A., Barham, P. et al.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems, *CoRR*, Vol. abs/1603.04467 (2016). Software available from tensorflow.org.

[4] 土橋彩香, 池宮由楽, 糸山克寿, 吉井和佳：歌声・調波楽器音・打楽器音分離とユーザ演奏のリアルタイム可視化に基づく音楽演奏練習システム, インタラクション 2016 論文集, 16INT012, pp. 97–105 (2016).

[5] Miu, V. and Fryazinov, O.: Accelerating Handheld Object Pose Estimation on Smartphones in Unity with TensorFlow Lite, *The 16th ACM SIGGRAPH European Conference on Visual Media Production, 56*, BFI Southbank, London, UK (2019).

[6] Rafii, Z., Liutkus, A., Stöter, F.-R., Mimilakis, S. I. and Bittner, R.: The MUSDB18 corpus for music separation, https://doi.org/10.5281/zenodo.1117372 (2017).

[7] Nearby | Google Developers, https://developers.google.com/nearby/?hl=ja. (最終アクセス 2020/01/16).