INTERSPEECH 2020
October 25–29, 2020, Shanghai, China

Harmonic Lowering for Accelerating Harmonic Convolution for Audio Signals

Hirotoshi Takeuchi[1], *Kunio Kashino*[2], *Yasunori Ohishi*[2], *Hiroshi Saruwatari*[1]

[1]Graduate School of Information Science and Technology, The University of Tokyo
[2]NTT Corporation, Japan

hirotoshi_takeuchi@ipc.i.u-tokyo.ac.jp,
{kunio.kashino.me,yasunori.ooishi.uk}@hco.ntt.co.jp,hiroshi_saruwatari@ipc.i.u-tokyo.ac.jp

Abstract

Convolutional neural networks have been successfully applied to a variety of audio signal processing tasks including sound source separation, speech recognition and acoustic scene understanding. Since many pitched sounds have a harmonic structure, an operation, called harmonic convolution, has been proposed to take advantages of the structure appearing in the audio signals. However, the computational cost involved is higher than that of normal convolution. This paper proposes a faster calculation method of harmonic convolution called Harmonic Lowering. The method unrolls the input data to a redundant layout so that the normal convolution operation can be applied. The analysis of the runtimes and the number of multiplication operations show that the proposed method accelerates the harmonic convolution 2 to 7 times faster than the conventional method under realistic parameter settings, while no approximation is introduced.

Index Terms: Harmonic Lowering, Convolutional Neural Network, Lowering, Harmonic Structure

1. Introduction

Inspired by the success in the image processing fields, convolutional neural networks (CNNs) [1] have been introduced in a variety of audio signal processing. For example, U-Net [2] has been successfully applied to sound source separation [3]. In addition to the two-dimensional one, one-dimensional convolution has also been proposed for the analysis of raw waveforms [4].

In the case of images, it is considered that small convolutional kernels can extract features from images because images usually have local continuity. However, sound spectrograms show little local continuity along the frequency axis. Regarding the relationships of components along the frequency axis, it is well known that a harmonic structure is observed in the case of pitched audio signals.

To incorporate the harmonic structure to CNN-based audio processing, a method called harmonic convolution [5] has been proposed, where harmonics are convoluted with a kernel. It has been reported that it achieves a higher performance in audio restoration and sound source separation tasks than with the normal convolution.

However, a major problem with harmonic convolution is its low speed. With a typical parameter setting, harmonic convolution can be about 10 times slower than normal convolution due to the memory access not always being continuous. Even in normal CNNs, convolution computation often becomes a bottleneck, and therefore, slower convolution could be a more difficult problem when considering the applications.

In this paper, we propose two efficient computational methods of harmonic convolution for a linear and logarithmic frequency axis, respectively. In both cases, no approximation is introduced, and therefore, the same calculation result as the original one is guaranteed. The analysis of the runtime and the number of multiplication operations show that the proposed method accelerates the harmonic convolution 2 to 7 times faster than the conventional method under realistic parameter settings.

Our paper is organized as follows. Section 2 introduces harmonic convolution. Section 3 introduces related works. Section 4 describes our proposed method. Section 5 shows the experimental results. Section 6 concludes this paper.

2. Harmonic Convolution

We introduce the definition of harmonic convolution and its implementation [5].

2.1. Definition

When a waveform signal $x[t]$ is given, the power of discrete short-time Fourier transformation $X[\omega, \tau]$ is given by Eq. (1).

$$X[\omega, \tau] = \|STFT(x[t])\|^2 \tag{1}$$

Then, harmonic convolution of X is defined by Eq. (2).

$$Y_n[\hat{\omega}, \hat{\tau}] = \sum_{k=1}^{K_f} \sum_{\tau=1}^{K_t} X\left[\frac{k\hat{\omega}}{n}, \hat{\tau} - \tau\right] \mathcal{K}[k, \tau] \tag{2}$$

$$Y[\hat{\omega}, \hat{\tau}] = \sum_{n=1}^{N} w_n Y_n[\hat{\omega}, \hat{\tau}], \tag{3}$$

where K_f and K_t are the kernel size along the frequency and time axes, respectively, n is an anchor with which the convolution is executed treating $\hat{\omega}/n$ as the base frequency. Eq. (3) is called anchor mixing because it is mixing the results of various anchors.

2.2. Problem in Harmonic Convolution

Harmonic convolution requires to access the spectrogram data located at frequency $k\hat{\omega}/n$. This is the core part of harmonic convolution, but at the same time, it can become a bottleneck for fast calculation. This is primarily because accessing distant data is costly. Data interpolation needed for the case when the frequency index becomes noninteger also requires additional computation.

2.3. Implementation in previous work

In [5], a type of approximation is introduced to harmonic convolution for computational efficiency, as in Eq. (4). This decomposes harmonic convolution to a combination of normal

http://dx.doi.org/10.21437/Interspeech.2020-3185

convolution on the time axis and the so-called deformable convolution along the frequency axis. Then, the anchor mixing is implemented as a pointwise convolution.

$$\mathcal{K}[k,\tau] \approx \mathcal{K}_f[k]\mathcal{K}_t[\tau] \tag{4}$$

However, we consider it could be difficult for such a decomposed kernel to capture the features of acoustic signal components whose frequencies vary over time.

3. Related Work

3.1. Deep Learning with Harmonic Structures

The harmonic structure appearing in a sound spectrogram of pitched sounds has been utilized in a variety of audio signal processing tasks for a long time [6]. However, when CNNs are used on the spectrograms, a convolutional kernel has usually been too small to cover harmonic structures, which have not been fully utilized. Recently, a harmonic convolution method was proposed [5]. Preprocessing methods using a learnable filter-bank [7] and F_0 estimation and harmonic structure rendering [8] have also been proposed.

3.2. Deformable Convolution

Deformable convolution is a generalized convolution method proposed in [9]. It enables to change reference data with given offsets Δ_x, Δ_y as in Eq. (5). The original motivation of this convolution method was to deal with variations of scale, pose, viewpoint and so on, of an input image with learnable offsets. In this method, if accessing data with a noninteger index is needed, 2-dimensional interpolation is performed.

$$Y[h,w] = \sum_x \sum_y (X[h-(x+\Delta_x(h,x)), w-(y+\Delta_y(w,y))]\mathcal{K}[x,y]) \tag{5}$$

In Section 5, we compare the runtime of the proposed method with that of an implementation based on deformable convolution.

3.3. Separable Convolution

Separable convolution is a method to reduce the number of parameters in convolution by approximating the kernel as in Eq. (6). It is reported that under a certain condition separable convolution can perform almost the same as normal convolution [10, 11]. It is also used in machine translation [12] and with mobile devices [13].

$$\mathcal{K}[c_{out}, c_{in}, h, w] \approx \mathcal{K}_p[c_{out}, c_{in}]\mathcal{K}_d[c_{in}, h, w] \tag{6}$$

As introduced in 2.3, the authors in [5] implemented harmonic convolution by introducing a type of separable convolution.

3.4. Efficient Calculation for Convolutional Neural Networks

When learning CNNs, the computational cost for convolution tends to be a bottleneck. Therefore, a number of methods have been proposed for its efficient calculation [14].

Generally, the approaches are roughly classified into architectural, algorithmic, and implementational ones. Architectural approaches involve the use of specific computational architecture such as cache mechanisms. Algorithmic approaches often reduce the problem by taking advantage of the common parts in the problem itself, such as in fast Fourier transform (FFT). For example, the use of FFT to convert convolutional operations to product operations [15, 16, 17] has been proposed, although the overhead of FFT cannot be ignored with small kernels, Implementational approaches aim at faster execution at the risk of increased memory usage. For example, loop unrolling has been a common computational technique up to now [23, 24, 25]. Another commonly-used method is called lowering, or Im2Col [18, 19]. The lowering enables fast computations on GPUs by transforming input data into a matrix so that convolution is executable in a highly optimized matrix multiplication routine. Efficient lowering methods with a reduced amount of multiplications [20, 21] and memory usage [22] have also been proposed.

We focus on an implementational approach in this paper. Inspired by the lowering, the proposed method transforms input data so that harmonic convolution is executable in the highly optimized normal convolution routine.

4. Method

In this section, we detail our proposed method called Harmonic Lowering to solve the problems described in 2.2. The implementation is available at `https://github.com/taketakeseijin/HarmonicLowering`.

4.1. Harmonic Lowering

The following equations are derived from Eq. (2).

$$X'_n[k,\hat{\omega},\hat{\tau}] = X\left[\frac{k\hat{\omega}}{n}, \hat{\tau}\right] \tag{7}$$

$$\mathcal{K}'[k,1,\tau] = \mathcal{K}[k,\tau] \tag{8}$$

$$Y_n[\hat{\omega},\hat{\tau}] = \sum_{k=1}^{K_f}\sum_{\omega=1}^{1}\sum_{\tau=1}^{K_t} X'_n[k,\hat{\omega},\hat{\tau}-\tau]\,\mathcal{K}'[k,\omega,\tau] \tag{9}$$

In Eq. (9), harmonic convolution is performed by the normal convolution of X'_n with kernel $\mathcal{K}'$. This is an essential point in the proposed method, as illustrated in Fig. 1.

First, by lowering X to X'_n as Eq. (7), harmonic data become accessible continuously. For example, accessing $X'_n[\cdot, f, \cdot]$ is equivalent to accessing $\forall k \in K\ X[kf/n, \cdot]$. In this part, X is shrank k/n times on the frequency axis and set to $X'_n[k]$. Second, $\mathcal{K}$ is transformed to $\mathcal{K}'$ as Eq. (8) to convolute correctly in Eq. (9). Third, X'_n is convoluted with kernel $\mathcal{K}'$ as Eq. (9). Here, the kernel size is 1 along the frequency axis and K_t along the time axis.

We call a sequence of Eqs. (7), (8), and (9) Harmonic Lowering. When setting shrank X to X_n, shrank X is zero padded for missing data of high frequencies, or trimmed to be of the constant size. Shrinking X is done by affine interpolation, but when $n = 1$, shrinking can be performed by striding. As an implementation tip, convolution for X' is quickly performed by treating the first axis as the input channel.

4.2. Logarithmic Harmonic Lowering

The Harmonic Lowering can be done on a logarithmic frequency scale, ignoring the DC components of spectrograms. We call it Logarithmic Harmonic Lowering, which is even simpler and faster than Harmonic Lowering on the linear frequency scale.

Following Eqs. (10), (11), and (12), harmonic convolution on a logarithmic frequency scale can be calculated, as illustrated in Fig. 2. Unlike Harmonic Lowering, Logarithmic Harmonic

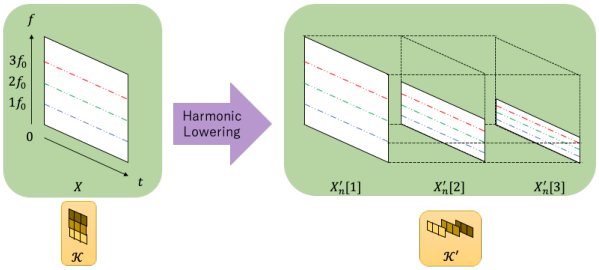

Figure 1: *Harmonic Lowering (3×3, n=1). It consists of shrinking (expanding) and unrolling by* $X'_n[k] = X\left[\frac{k\hat{\omega}}{n}, \hat{\tau}\right]$ *for* $k = 1, \cdots, K_f$. *The zero-padding and trimming are applied when needed. The harmonic elements are accessible by* $X'_n[\cdot, f, \cdot]$. *For example, reference to* $X[1f_0, \cdot], X[2f_0, \cdot], X[3f_0, \cdot]$ *is done by accessing to* $X'_n[\cdot, f_0, \cdot]$.

Lowering only requires shifting $\log k/n$ on the frequency axis when lowering X, which also reduces the amount of computation.

$$X'_n\left[k, \log\hat{\omega}, \hat{\tau}\right] = X\left[\log\frac{k\hat{\omega}}{n}, \hat{\tau}\right] = X\left[\log k/n + \log\hat{\omega}, \hat{\tau}\right] \quad (10)$$

$$\mathcal{K}'[k, 1, \tau] = \mathcal{K}[k, \tau] \quad (11)$$

$$Y_n[\log\hat{\omega}, \hat{\tau}] = \sum_{k=1}^{K_f}\sum_{\omega=1}^{1}\sum_{\tau=1}^{K_t} X'_n\left[k, \log\hat{\omega}, \hat{\tau}-\tau\right]\mathcal{K}'[k, \omega, \tau] \quad (12)$$

4.3. Total Sums and Multiplications

For analysis purposes, here we enumerate the total number of sums and multiplications needed in computing one element of $Y_n[\hat{\omega}, \hat{\tau}]$.

4.3.1. Bilinear Interpolation

Deformable convolution computes bilinear interpolation for each $X[k\hat{\omega}/n, \hat{\tau}-\tau]$. Bilinear interpolation equations are below.

$$\begin{aligned}
f &= \frac{k\hat{\omega}}{n},\ t = \hat{\tau} - \tau,\\
f_{floor} &= floor(f),\ f_{ceil} = f_{floor} + 1,\\
t_{floor} &= floor(t),\ t_{ceil} = t_{floor} + 1,\\
w_{fceil} &= f - f_{floor},\ w_{ffloor} = 1 - w_{fceil},\\
w_{tceil} &= f - t_{floor},\ w_{tfloor} = 1 - w_{tceil},\\
X[f,t] &= w_{ffloor}w_{tfloor}X[f_{floor}, t_{floor}]\\
&\quad + w_{ffloor}w_{tceil}X[f_{floor}, t_{ceil}]\\
&\quad + w_{fceil}w_{tfloor}X[f_{ceil}, t_{floor}]\\
&\quad + w_{fceil}w_{tfceil}X[f_{ceil}, t_{ceil}].
\end{aligned}$$

Ignoring the floor function and computing f, t, bilinear interpolation needs 8 multiplications and 9 sums. This interpolation is executed $k\tau$ times, convolution only requires $k\tau$ sums and $k\tau$ multiplications, and the computing index requires k sums and τ multiplications. The total numbers are as follows:

- Total multiplications: $k\tau(1+8)+k$,
- Total sums: $k\tau(1+9)+\tau$.

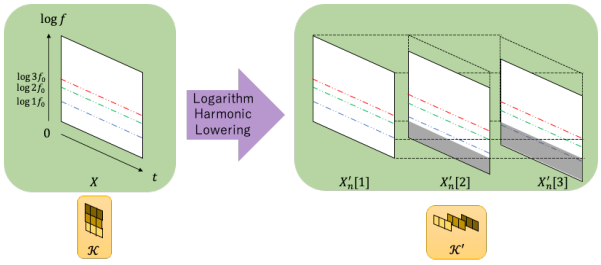

Figure 2: *Logarithmic Harmonic Lowering* $(3 \times 3,\ n = 1)$. *It consists of shifting and unrolling operations by* $X'_n[k] = X\left[\log k/n + \log\hat{\omega}, \hat{\tau}\right]$ *for* $k = 1, \cdots, K_f$. *The zero-padding and trimming are applied when needed. In the above illustration, the grey part of* X'_n *means the trimmed part. The harmonic elements are accessible by* $X'_n[\cdot, \log f, \cdot]$. *For example, reference to* $X[\log 1f_0, \cdot], X[\log 2f_0, \cdot], X[\log 3f_0, \cdot]$ *is done by accessing* $X'_n[\cdot, \log f_0, \cdot]$.

4.3.2. Affine Interpolation

Unless $n = 1$, Harmonic Lowering involves linear interpolation for each $X'_n[k, \hat{\omega}, \hat{\tau}-\tau]$. We can assume affine transformation regarding the frequency axis, and therefore, interpolation weights are predictable. The interpolation is done by the equations below.

$$\begin{aligned}
f &= \frac{k\hat{\omega}}{n},\\
f_{floor} &= floor(f),\ f_{ceil} = f_{floor} + 1,\\
w_{ceil} &= f - f_{floor},\ w_{floor} = 1 - w_{ceil},\\
X[f] &= w_{floor}X[f_{floor}] + w_{ceil}X[f_{ceil}].
\end{aligned}$$

Here, w_{ceil}, w_{floor} are predictable and do not need to be computed, and therefore, convolution requires 2 multiplications and 1 sum. The total numbers are as follows:

- Total multiplications: $k\tau\,(1+2)+k$,
- Total sums: $k\tau\,(1+1)+\tau$.

5. Experiments

5.1. Exp. 1: Kernel size and anchor

To evaluate the acceleration performance, we measured the runtime and the maximum amount of memory usage of harmonic convolution, changing the kernel sizes and the anchor. Throughout this paper, the runtimes were measured 100 times and then averaged, by using random input and gradient output for each time. We compared the following three methods.

1. Deformable convolution
2. Harmonic Lowering
3. Logarithmic Harmonic Lowering

We used a PC with one Nvidia GeForce GTX 1080Ti GPU for the measurement. Experimental parameters are listed in Table 1. The batch size was 16, and the dilation, stride and group parameters were all set to 1.

The results are summarized in Fig. 3. In each bar, the bottom part shows the runtime for the forward calculation, the upper part for backward calculation. The colored dots indicate the maximum memory usage. For reference, we also plots those of the normal convolution. The results showed four points of interest. First, the proposed method was faster than the conventional

Table 1: *Experimental settings. The parameters* $n, C_{in}, C_{out}, S, K, P$ *are anchor, input channel size, output channel size, input spectrogram size, kernel size, and padding size, respectively.*

	n	C_{in}	C_{out}	S	K	P
Setting1	1	16	32	(256,256)	(7,7)	(3,3)
Setting2	1	16	32	(256,256)	(5,5)	(2,2)
Setting3	1	16	32	(256,256)	(3,3)	(1,1)
Setting1a	7	16	32	(256,256)	(7,7)	(3,3)
Setting2a	5	16	32	(256,256)	(5,5)	(2,2)
Setting3a	3	16	32	(256,256)	(3,3)	(1,1)

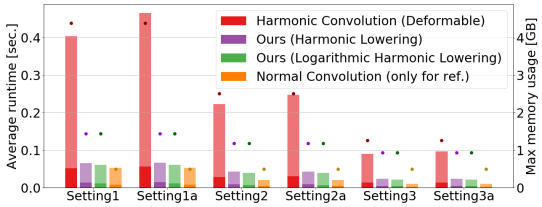

Figure 3: *Averaged runtime for harmonic convolution (1). The bottom, middle and top parts of each bar show the runtimes for forward, backward, and other calculations, respectively.*

method in all cases, and it was about 7 times faster in some cases. Second, when the anchor parameter was greater than 1, the runtime slightly increased. Third, as the kernel size became greater, the ratio of the proposed method's runtime to that of the normal convolution became smaller. Fourth, in all settings, the proposed methods are more efficient than the conventional in the maximum memory usage.

5.2. Exp. 2: Input spectrogram size

We tested how the runtime and the memory usage change when the input spectrogram size was changed. Experimental parameters are listed in Table 2. The remaining setups were the same as those in 5.1.

The results are summarized in Fig. 4, and showed four points of interest. First, in all settings, the proposed method was faster than the conventional method, and the acceleration was about 2 to 7 times, depending on the settings. Second, when the input size became greater, the ratio of the proposed method's runtime to that of the normal convolution became smaller. Third, in setting5, the conventional method was the slowest among the tested settings. Fourth, in all settings, the proposed methods are more efficient than the conventional in the maximum memory usage.

5.3. Discussion

When comparing setting4 with setting5, the channel size of setting4 is 1 and its data access cost is relatively small. In addition, the deformable convolution routine we used is highly optimized while our implementation of the proposed method is not. Thus, there is room for improvement.

For a realistic example, let us assume building a model consisting of setting4∼9 and the pooling layers. Then the estimated runtime of normal convolution is 0.0089 s, by simply adding all the runtimes of setting4∼9, whereas that of the conventional method and Harmonic Lowering are 0.0384 s and 0.0152 s, respectively. That is, the conventional method takes over four times more time compared with the normal model, but with the

Table 2: *Experimental settings. The parameters* $n, C_{in}, C_{out}, S, K, P$ *are anchor, input channel size, output channel size, input spectrogram size, kernel size, and padding size, respectively.*

	n	C_{in}	C_{out}	S	K	P
Setting4	1	1	16	(512,512)	(3,3)	(1,1)
Setting5	1	16	32	(256,256)	(3,3)	(1,1)
Setting6	1	32	64	(128,128)	(3,3)	(1,1)
Setting7	1	64	128	(64,64)	(3,3)	(1,1)
Setting8	1	128	256	(32,32)	(3,3)	(1,1)
Setting9	1	256	512	(16,16)	(3,3)	(1,1)

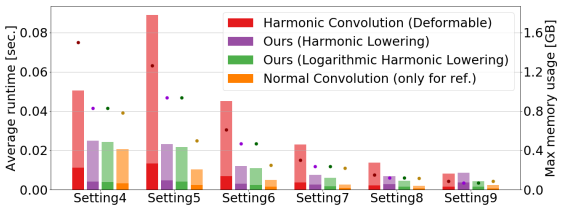

Figure 4: *Averaged runtime for harmonic convolution (2). The bottom, middle and top parts of each bar show the runtimes for forward, backward, and other calculations, respectively.*

proposed method, the required time is less than two times compared with the normal model.

In 5.1, we observed that the runtime for each method changes with the kernel size. To further analyze it, Fig. 5 shows the runtimes for the forward calculation of deformable convolution and Harmonic Lowering, overlapped with the number of multiplications studied in 4.3. It is shown that the number of multiplications corresponds well to the runtime, which implies that the multiplication may account for a major portion of the computational time.

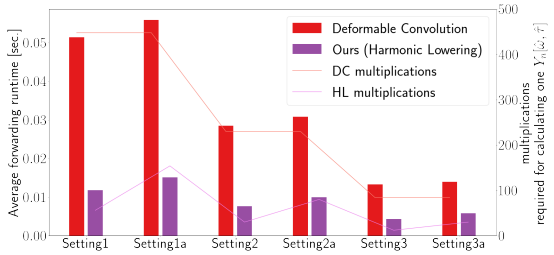

Figure 5: *Runtime and the number of multiplications for the forward calculation required for one element of* $Y_n[\hat{\omega}, \hat{\tau}]$

6. Conclusions

In this paper, we proposed a fast calculation method of harmonic convolution, called Harmonic Lowering, while maintaining the exactly same calculation results. The method is based on the mapping of the input data to a redundant layout so that a normal convolution operation with continuous data access can be applied. The measurement and analysis of the runtimes and the maximum memory usage revealed that the proposed method accelerates the harmonic convolution 2 to 7 times faster and more memory-efficient than the conventional method, under various, realistic parameter settings.

7. References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. NIPS*, 2012, pp. 1097–1105.

[2] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. MICCAI*, 2015, pp. 234–241.

[3] A. Jansson, E. Humphrey, N. Montecchio, R. Bittner, A. Kumar, and T. Weyde, "Singing voice separation with deep U-Net convolutional networks," in *Proc. ISMIR*, 2017, pp. 745–751.

[4] D. Stoller, S. Ewert, and S. Dixon, "Wave-U-Net: A multi-scale neural network for end-to-end audio source separation," in *Proc. ISMIR*, 2018, pp. 334–340.

[5] Z. Zhang, Y. Wang, C. Gan, J. Wu, J. B. Tenenbaum, A. Torralba, and W. T. Freeman, "Deep audio priors emerge from harmonic convolutional networks," in *Proc. ICLR*, 2020.

[6] A. Nehorai and B. Porat, "Adaptive comb filtering for harmonic signal enhancement," in *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 34, no. 5, 1986, pp. 1124–1138.

[7] M. Won, S. Chun, O. Nieto, and X. Serrc, "Data-driven harmonic filters for audio representation learning," in *Proc. ICASSP*, 2020, pp. 536–540.

[8] T. Nakano, K. Yoshii, Y. Wu, R. Nishikimi, K. W. E. Lin, and M. Goto, "Joint singing pitch estimation and voice separation based on a neural harmonic structure renderer," in *Proc. WASPAA*, 2019, pp. 160–164.

[9] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, "Deformable convolutional networks," in *Proc. ICCV*, 2017, pp. 764–773.

[10] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proc. CVPR*, 2017, pp. 1251–1258.

[11] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[12] L. Kaiser, A. N. Gomez, and F. Chollet, "Depthwise separable convolutions for neural machine translation," *arXiv preprint arXiv:1706.03059*, 2017.

[13] P. Zhang, E. Lo, and B. Lu, "High performance depthwise and pointwise convolutions on mobile devices," in *Proc. AAAI*, 2020.

[14] Q. Zhang, M. Zhang, T. Chen, Z. Sun, Y. Ma, and B. Yu, "Recent advances in convolutional neural network acceleration," *Neurocomputing*, vol. 323, pp. 37–51, 2019.

[15] M. Mathieu, M. Henaff, and Y. LeCun, "Fast training of convolutional networks through FFTs," *arXiv preprint arXiv:1312.5851*, 2013.

[16] T. Highlander and A. Rodriguez, "Very efficient training of convolutional neural networks using fast fourier transform and overlap-and-add," *arXiv preprint arXiv:1601.06815*, 2016.

[17] J. H. Ko, B. Mudassar, T. Na, and S. Mukhopadhyay, "Design of an energy-efficient accelerator for training of convolutional neural networks using frequency-domain computation," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2017, pp. 1–6.

[18] K. Chellapilla, S. Puri, and P. Simard, "High performance convolutional neural networks for document processing," in *Workshop on Frontiers in Handwriting Recognition*, 2006.

[19] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cuDNN: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.

[20] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *Proc. CVPR*, 2016, pp. 4013–4021.

[21] H. Park, D. Kim, J. Ahn, and S. Yoo, "Zero and data reuse-aware fast convolution for deep neural networks on gpu," in *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, 2016.

[22] M. Cho and D. Brand, "MEC: memory-efficient convolution for deep neural network," in *Proc. ICML*, 2017, pp. 815–824.

[23] A. Krizhevsky, "cudaconvnet2," https://code.google.com/p/cuda-convnet2/, 2014.

[24] J. Zhang, F. Franchetti, and T. M. Low, "High performance zero-memory overhead direct convolutions," in *Proc. ICML*, 2018, pp. 5776–5785.

[25] A. Gural and B. Murmann, "Memory-optimal direct convolutions for maximizing classification accuracy in embedded applications," in *Proc. ICML*, 2019, pp. 2515–2524.