Indirect Deconvolution Algorithm

Marat Dukhan
Google Research
Mountain View, CA, USA
Email: maratek@google.com

***Abstract*—Neural network frameworks today commonly implement Deconvolution and closely related Convolution operator via a combination of GEMM (dense matrix-matrix multiplication) and a memory transformation. The recently proposed Indirect Convolution algorithm suggests a more efficient implementation of Convolution via the Indirect GEMM primitive - a modification of GEMM where pointers to rows are loaded from a buffer rather than being computed assuming constant stride. However, the algorithm is inefficient for Deconvolution with non-unit stride, which is typical in computer vision models. We describe a novel Indirect Deconvolution algorithm for efficient evaluation of the Deconvolution operator with non-unit stride by splitting Deconvolution with a large kernel into multiple subconvolutions with smaller, variable-size kernels, which can be efficiently implemented on top of the Indirect GEMM primitive.**

***Keywords*-deconvolution, transposed convolution, segmentation**

I. Introduction

When the Deconvolution operator was proposed for semantic segmentation [1] in 2014, it soon developed into a standard component of convolutional neural networks for dense prediction tasks, including semantic [1]–[4], instance [5], and panoptic [6] segmentation, and generative adversarial networks [7]. In these models the Deconvolution operator serves as generalized upsampling function to predict the output image from a small-scale representation. Unlike bi-linear and bi-cubic upsampling operators, Deconvolution can use all channels of the input image to predict each channel of an output pixel, and learns interpolation weights during neural network training.

Deconvolution is related to a more common Convolution operator in three ways.

1) Deconvolution with unit stride (i.e. without upsampling properties) is equivalent to Convolution, up to permutation of parameters. While this case is mathematically valid, it has no practical applications in convolutional neural networks, and thus **we consider only Deconvolutions with non-unit stride in this paper**.
2) The forward pass (i.e. predicting outputs from inputs) of Deconvolution operator is equivalent to backward (i.e. computing gradients of input from gradients of output) pass of Convolution operator. Conversely, the backward pass of the Deconvolution operator is equivalent to the forward pass of a Convolution operator. **We focus on the inference use-case and the forward pass of Deconvolution** in this paper, and leave backward pass to algorithms designed for the forward pass of Convolution operator [8]–[11]
3) Deconvolution can be represented as a combination of an exotic (different from nearest-neighbour, bi-linear, bi-cubic) upsampling and a Convolution operator. The non-standard upsampling pulls input pixels apart according to the stride parameter and fills the spaces between them with zeroes. Fig. 1 illustrates such input upsampling for Deconvolution with 2x2 stride.

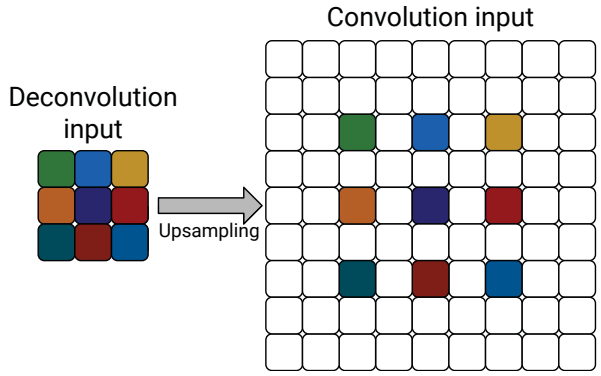

Figure 1. Upsampling of 3x3 stride-2 Deconvolution input to pass into 3x3 Convolution operator. The white pixels in the Convolution input represent zeroes.

Despite similarities to the Convolution operator and vast amount of research on its optimal implementation [8]–[10], [12]–[15], Deconvolution received little attention and popular machine learning frameworks (PyTorch, TensorFlow, Caffe) implement it through a combination of GEMM (dense matrix-matrix multiplication) and COL2IM memory transformation. COL2IM is the inverse of the IM2COL transformation.

We build upon recent research on Indirect Convolution algorithm, and bring two contributions to the study of efficient algorithms for convolutional neural networks:

- We suggest how the Indirect Convolution algorithm can be adapted to directly evaluate Deconvolution via implicit up-sample + Convolution decomposition.

978-1-7281-7445-7/20/$31.00 ©2020 IEEE
DOI 10.1109/IPDPSW50202.2020.00154

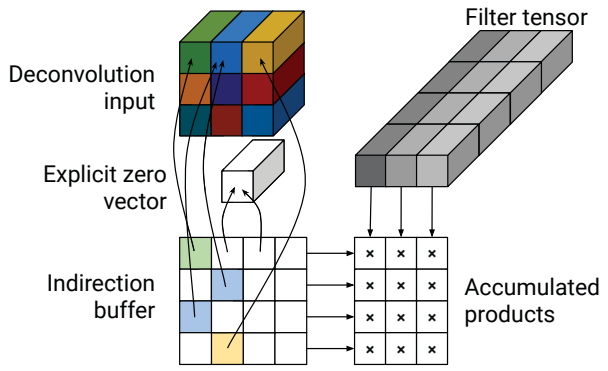

Figure 2. Calculation of Deconvolution with Indirect Convolution algorithm. The white pixels represent zeroes from upsampling.

- We introduce a novel Indirect Deconvolution algorithm that operates by decomposing Deconvolution into multiple smaller Convolutions with variable kernel size. The resulting algorithm outperforms both the Indirect Convolution algorithm adapted to Deconvolution, and the traditional implementation of Deconvolution based on GEMM and col2im transformation.

II. Deconvolution via Indirect Convolution Algorithm

The Indirect Convolution Algorithm [11] calculates the output of a Convolution operator in NHWC layout via the Indirect GEMM primitive - a modification of GEMM which reads pointers to rows of input pixels from an indirection buffer. For a convolution with kernel size $K_h \times K_w$, the algorithm initializes an indirection buffer with pointers to $K_h K_w$ input pixels contributing to each output pixel, and the Indirect GEMM primitive loads pointers from the buffer and accumulates $K_h \times K_w$ dot products of each input row with the corresponding row of filter.

A naive application of the Indirect Convolution algorithm to Deconvolution would explicitly up-sample the Deconvolution input as illustrated on Fig. 1, and invoke the algorithm for the Convolution computation. However, indirection buffer provides a key to a more efficient implementation: rather than explicitly materialise the up-sampled input, we can do so implicitly, by initializing entries of the indirection buffer as if they were pixels of the up-sampled input. Entries corresponding to pixels of original input in the up-sampled image would point directly to the pixels of the original input, and entries corresponding to the filling zeroes in the up-sampled input would point to an explicit zero vector - a constant array with zero-initialized elements.

The optimized implementation of the Indirect Convolution algorithm, illustrated on Fig. 2, avoids the memory and runtime overhead of explicit upsampling. Unfortunately, the Indirect Convolution algorithm involves another inefficiency: it performs computations on zero elements in the implicitly up-sampled input. The overhead due to this inefficiency is very significant, and commonly exceeds 100%, as demonstrated in Table II.

Deconvolution	3x3 stride-2	2x2 stride-2	4x4 stride-4
Overhead	125%	300%	1500%

Table I
Asymptotic overhead in computing Deconvolution via the Indirect Convolution algorithm due to computations on zero elements.

III. Indirect Deconvolution Algorithm

As the Convolution kernel slides over up-sampled Deconvolution input, the positions of the non-zero input pixels and corresponding kernel elements form a regular pattern, which repeats every $S_w \times S_h$ pixels, where S_w is the width stride and S_h is the height stride of the Deconvolution operator. This pattern is the motivation for the Indirect Deconvolution algorithm: decompose Convolution over up-sampled Deconvolution input into several subconvolutions. Each subconvolution would correspond to particular offset x, y within the stride-sized window, where $0 \le x < S_w$ and $0 \le y < S_h$, and include only the non-zero pixels of the up-sampled inputs, and the corresponding kernel elements. The subconvolution corresponding to offset x, y produce output pixels $x + S_w \cdot n$ and $y + S_h \cdot m$ for all integer n and m within the output image bounds. Fig. 3 illustrated this algorithm for a 3×3 Deconvolution with 2×2 stride. The computation is being decomposed into $2 \times 2 = 4$ subconvolutions, which jointly produce the output of the original Deconvolution. As the 3×3 kernel size does not exactly divide by the stride size, these subconvolutions operate with different kernel sizes.

IV. Experimental Evaluation

Device	Chipset	Microarchitecture
Galaxy S8	Exynos 8895	Exynos-M2
Pixel 2	Snapdragon 835	Cortex-A73
Pixel 3a	Snapdragon 670	Cortex-A75

Table II
Specifications of mobile devices for performance evaluation. The microarchitecture is specified for the big core.

Platforms We evaluate the algorithms for Deconvolution on three Android ARM64 mobile devices with characteristics listed in Table IV. These devices features processors with two types of cores: high-performance (big) cores and low-power (little) cores, and for the benchmarks in this section we pinned evaluation to a single big core.

Workload We used parameters of Deconvolution operators in the two mobile-optimized architectures for image segmentation – ENet [3] and ESPNet [4].

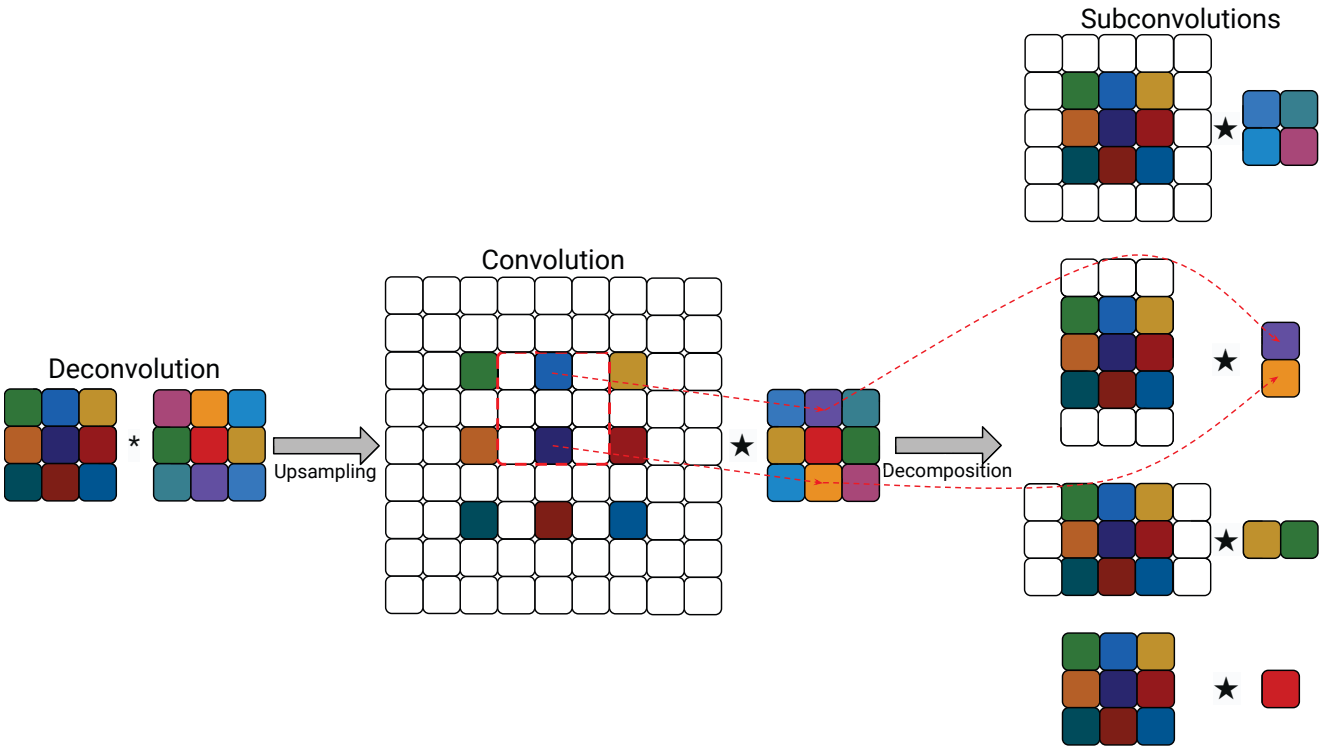

Figure 3. Calculation of Deconvolution with Indirect Deconvolution algorithm. The white pixels in the represent zeroes from upsampling.

Implementation For a baseline, we use the implementation of the traditional GEMM+COL2IM algorithm from the TensorFlow Lite framework. The matrix-matrix multiplication (GEMM) functionality in this implementation is provided by the highly optimized Ruy library and includes expert-tuned micro-kernels in assembly.

We implemented the Indirect Deconvolution algorithm on top of the Indirect GEMM primitive available in the XNNPACK library. Similarly to GEMM primitive in Ruy, these Indirect GEMM primitives are implemented by optimization experts in ARM64 assembly. However, to control for potential difference in the quality of low-level optimization between the two libraries, we also benchmark the same implementation of the Indirect Deconvolution algorithm with a compiler-generated Indirect GEMM primitive.

Additionally, we evaluate an implementation of the Indirect Convolution algorithm adapted to Deconvolution as described in Sec. II. Our implementation of the Indirect Convolution algorithm for Deconvolution leverage the same assembly-optimized Indirect GEMM primitive as the Indirect Deconvolution algorithm.

Protocol We employ the Google Benchmark framework to estimate sustained performance on each workload. We replicate each micro-benchmark 25 times, and report the median performance (in GFLOPS) of the 25 runs, and its 20% and 80% quantiles.

In each run of the micro-benchmarks we simulate the cache state during neural network inference: we evict filter, bias, output tensors, and indirect buffers from cache, and prefetch the input tensor into first-level data cache. For both the Indirect Convolution algorithm and the Indirect Deconvolution algorithm we initialize the indirection once and reused it across Deconvolution computations.

Results Fig. 4 presents detailed performance of the four evaluated implementations for each Deconvolution operators in the ENet and ESPNet networks. The proposed Indirect Deconvolution algorithm consistently outperforms the traditional GEMM+COL2IM algorithm across all Deconvolution operators and all devices. Even when the Indirect Deconvolution algorithm is implemented on top of compiler-generated Indirect GEMM primitive, rather than a one in optimized assembly, the Indirect Deconvolution algorithm surpass GEMM+COL2IM in all but one case, confirming that the advantage can not be attributed to low-level differences in the implementation. Moreover, the Indirect Convolution algorithm, which employs assembly version of the Indirect GEMM primitive, under-performs both the proposed Indirect Deconvolution algorithm and the traditional GEMM+COL2IM algorithm. This latter fact can be explained by substantial overhead in the Indirect Convolution algorithm due to computations on zero inputs.

Table IV summarize the speedup of the Indirect Decon-

Network	Galaxy S8	Pixel 2	Pixel 3a
ENet	40.9%	25.5%	33.1%
ESPNet	71.1%	48.0%	62.9%

Table III
SPEEDUP OF THE INDIRECT DECONVOLUTION ALGORITHM OVER THE GEMM+COL2IM ALGORITHM.

volution algorithm over the traditional GEMM+COL2IM algorithm across the three devices and two segmentation architectures.

V. CONCLUSION

The Indirect Deconvolution algorithm is a modification of the Indirect Convolution algorithm for efficient computation of the forward pass of Deconvolution operator. Unlike the Indirect Convolution algorithm, the Indirect Deconvolution algorithm skips most computations on zero inputs and consistently outperforms the traditional GEMM+COL2IM algorithm by $25 - 71\%$.

To foster reproducibility, we released an open-source implementation of the proposed Indirect Deconvolution algorithm and the Indirect Convolution algorithm adapted for Deconvolution as a part of XNNPACK library at GitHub. com/google/XNNPACK.

REFERENCES

[1] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[2] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[3] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "Enet: A deep neural network architecture for real-time semantic segmentation," *arXiv preprint arXiv:1606.02147*, 2016.

[4] S. Mehta, M. Rastegari, A. Caspi, L. Shapiro, and H. Hajishirzi, "ESPNet: Efficient spatial pyramid of dilated convolutions for semantic segmentation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 552–568.

[5] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[6] A. Kirillov, R. Girshick, K. He, and P. Dollár, "Panoptic feature pyramid networks," *arXiv preprint arXiv:1901.02446*, 2019.

[7] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[8] M. Cho and D. Brand, "MEC: memory-efficient convolution for deep neural network," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 815–824.

[9] A. Anderson, A. Vasudevan, C. Keane, and D. Gregg, "Low-memory GEMM-based convolution algorithms for deep neural networks," *arXiv preprint arXiv:1709.03395*, 2017.

[10] E. Georganas, S. Avancha, K. Banerjee, D. Kalamkar, G. Henry, H. Pabst, and A. Heinecke, "Anatomy of high-performance deep learning convolutions on simd architectures," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2018, pp. 830–841.

[11] M. Dukhan, "The Indirect Convolution Algorithm," *arXiv preprint arXiv:1907.02129*, 2019.

[12] J. Zhang, F. Franchetti, and T. M. Low, "High performance zero-memory overhead direct convolutions," *arXiv preprint arXiv:1809.10170*, 2018.

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

[14] N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, and Y. LeCun, "Fast convolutional nets with fbfft: A gpu performance evaluation," *arXiv preprint arXiv:1412.7580*, 2014.

[15] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4013–4021.

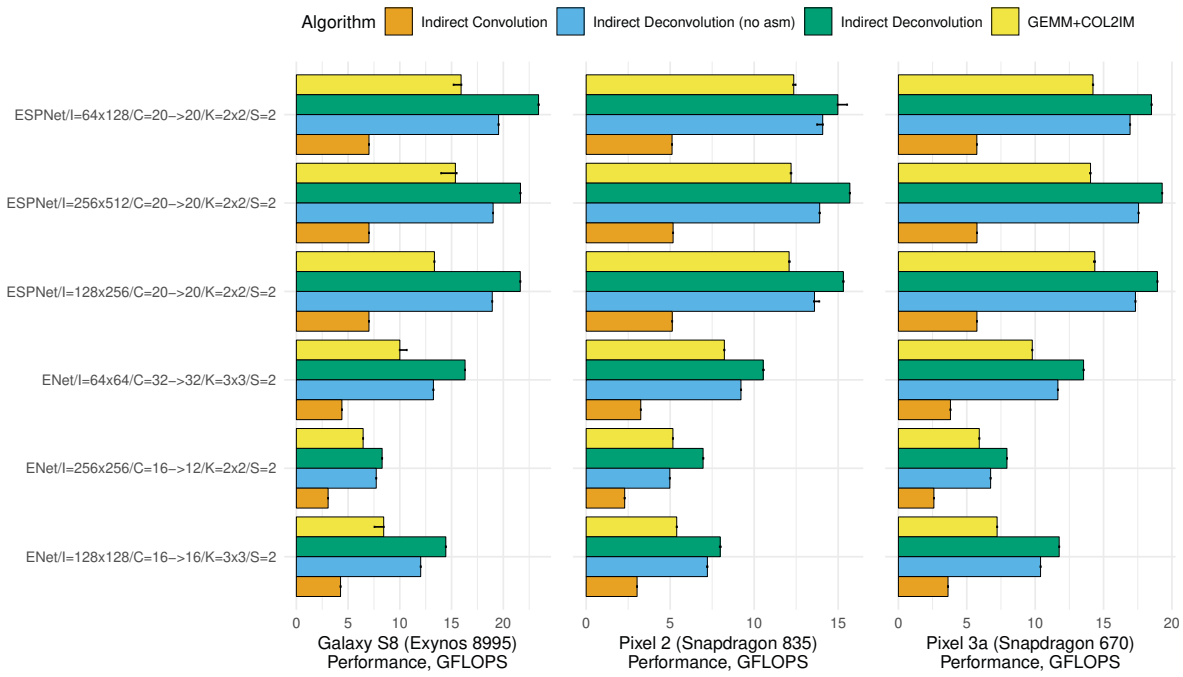

Figure 4. Performance of the Indirect Deconvolution algorithm, the Indirect Convolution algorithm and the GEMM+COL2IM algorithm on the Deconvolution operators of ENet and ESPNet architectures. Error bars show 20% and 80% performance quantiles.