A Simple and Efficient Implementation of im2col in Convolution Neural Networks

Hao Zhang

zhangh0214@gmail.com

March 2, 2017

1 Introduction

In convolutional nerual networks (CNN), the most time consuming part is the convolution layer. Convolution is usually done by im2col, which convert the 3 D input data tensor and weight tensor into 2 D matrices, then the complicated convolution operation can be done by matrix multiplications. Therefore, the efficiency of im2col operations dettermine the overall speed. In this article, we proposed a simple and efficient implementation of im2col which can take place the Caffe's implementation. When training MNIST on LeNet, we are 20.6% faster than Caffe's implementation.

2 Notations

The notations used throughout this article are described in Tab. 1. The relationships between H_0 and H_1, W_0 and W_1 are

$$H_1 = H_0 + 2P - F + 1\,, \tag{1}$$

$$W_1 = W_0 + 2P - F + 1\,. \tag{2}$$

If padding is not used, P is zero in the above equations. For simplicity, we assume the stride is 1 here, which is common to many start-of-the-art CNN settings.

3 im2col

3.1 Without Padding

The convolution operation is

$$A_{d_1,i_1,j_1} = \sum_{d_0=0}^{D_0-1} \sum_{u=0}^{F-1} \sum_{v=0}^{F-1} W_{d_1,d_0,u,v} \cdot X_{d_0,i'_0,j'_0} \tag{3}$$

Table 1: Notations.

Variable	Dimension	Meaning
D_0, H_0, W_0	$\mathbb{R}$	Input Channels, Height, Width
D_1, H_1, W_1	$\mathbb{R}$	Output Channels, Height, Width
F	$\mathbb{R}$	Kernel Size
P	$\mathbb{R}$	Padding
$\boldsymbol{X}$	$\mathbb{R}^{D_0 \times H_0 \times W_0}$	Input Without Padding
$\phi(\boldsymbol{X})$	$\mathbb{R}^{(D_0 F^2) \times (H_1 W_1)}$	im2col Result of $\boldsymbol{X}$
$\boldsymbol{Z}$	$\mathbb{R}^{D_0 \times (H_0+2P) \times (W_0+2P)}$	Input With Padding
$\phi(\boldsymbol{Z})$	$\mathbb{R}^{(D_0 F^2) \times (H_1 W_1)}$	im2col Result of $\boldsymbol{Z}$
$\boldsymbol{W}$	$\mathbb{R}^{D_1 \times D_0 \times F \times F}$	Weight
$\phi(\boldsymbol{W})$	$\mathbb{R}^{D_1 \times (D_0 F^2)}$	im2col Result of $\boldsymbol{W}$
$\boldsymbol{A}$	$\mathbb{R}^{D_1 \times H_1 \times W_1}$	Output
$\phi(\boldsymbol{A})$	$\mathbb{R}^{D_1 \times (H_1 W_1)}$	im2col Result of $\boldsymbol{A}$
(d_0, i'_0, j'_0)	$\mathbb{R}$	Index of $\boldsymbol{X}$
(p', q')	$\mathbb{R}$	Index of $\phi(\boldsymbol{X})$
(d_0, i_0, j_0)	$\mathbb{R}$	Index of $\boldsymbol{Z}$
(p, q)	$\mathbb{R}$	Index of $\phi(\boldsymbol{Z})$
(d_1, d_0, u, v)	$\mathbb{R}$	Index of $\boldsymbol{W}$
(d_1, p)	$\mathbb{R}$	Index of $\phi(\boldsymbol{W})$
(d_1, i_1, j_1)	$\mathbb{R}$	Index of $\boldsymbol{A}$
(d_1, q)	$\mathbb{R}$	Index of $\phi(\boldsymbol{A})$

$$= \sum_{d_0=0}^{D_0-1} \sum_{u=0}^{F-1} \sum_{v=0}^{F-1} W_{d_1,d_0,u,v} \cdot X_{d_0,i_1+u,j_1+v}, \forall d_1, i_1, j_1 \, . \tag{4}$$

By using im2col,

$$\phi(A)_{d_1,q'} = \sum_{p=0}^{D_0F^2-1} \phi(W)_{d_1,p'} \phi(X)_{p',q'} \tag{5}$$

$$= \sum_{p=0}^{D_0F^2-1} \phi(W)_{d_1,p'} \phi(X)_{p',i_1W_1+j_1} \tag{6}$$

Algorithm 1 im2col Without Padding.

Input: $\boldsymbol{X}$
Output: $\phi(\boldsymbol{X})$

```
for $k = 0$ to $D_0F^2H_1W_1 - 1$
    $p' = k \; / \; (H_1W_1)$
    $q' = k \; \% \; (H_1W_1)$
    $d_0 = (p' \; / \; F) \; / \; F$
    $i'_0 = q' \; / \; W_1 + (p' \; / \; F) \; \% \; F$
    $j'_0 = q' \; \% \; W_1 + p' \; \% \; F$
    $\phi(X)_{p',q'} = X_{d_0,i'_0,j'_0}$
```

$$= \sum_{d_0=0}^{D_0-1} \sum_{u=0}^{F-1} \sum_{v=0}^{F-1} \phi(W)_{d_1,d_0F^2+uF+v} \phi(X)_{d_0F^2+uF+v,i_1W_1+j_1} \tag{7}$$

$$= \sum_{d_0=0}^{D_0-1} \sum_{u=0}^{F-1} \sum_{v=0}^{F-1} \phi(W)_{d_1,(d_0F+u)F+v} \phi(X)_{(d_0F+u)F+v,i_1W_1+j_1} \,. \tag{8}$$

Therefore, the relationships between indices are

$$p' = (d_0F + u)F + v \,, \tag{9}$$
$$q' = i_1W_1 + j_1 \,, \tag{10}$$
$$i'_0 = i_1 + u \,, \tag{11}$$
$$j'_0 = j_1 + v \,. \tag{12}$$

These can be rewriten as

$$u = (p' \; / \; F) \; \% \; F \,, \tag{13}$$
$$v = p' \; \% \; F \,, \tag{14}$$
$$i_1 = q' \; / \; W_1 \,, \tag{15}$$
$$j_1 = q' \; \% \; W_1 \,, \tag{16}$$
$$d_0 = (p' \; / \; F) \; / \; F \,, \tag{17}$$
$$i'_0 = i_1 + u \,, \tag{18}$$
$$j'_0 = j_1 + v \,. \tag{19}$$

Where / means integer division and % means module. When we know the $\phi(\boldsymbol{X})$'s index (p', q'), we can follow the above equations to compute $\boldsymbol{X}$'s index (d_0, i'_0, j'_0), see Alg. 1.

Algorithm 2 im2col With Padding.

Input: $\boldsymbol{X}$

Output: $\phi(\boldsymbol{Z})$

```
for $k = 0$ to $D_0F^2H_1W_1 - 1$
  $p = k \ / \ (H_1W_1)$
  $q = k \ \% \ (H_1W_1)$
  $d_0 = (p \ / \ F) \ / \ F$
  $i_0 = q \ / \ W_1 + (p \ / \ F) \ \% \ F$
  $j_0 = q \ \% \ W_1 + p \ \% \ F$
  if $i_0 \in [P, H_0 + P) \wedge j_0 \in [P, W_0 + P)$
    $\phi(Z)_{p,q} = X_{d_0, i_0 - P, j_0 - P}$
  else
    $\phi(Z)_{p,q} = 0$
```

3.2 With Padding

The relationship between $\boldsymbol{X}$ and $\boldsymbol{Z}$ is

$$Z_{d_0,i_0,j_0} = \begin{cases} X_{d_0,i_0-P,j_0-P} & \text{if } i_0 \in [P, H_0+P) \wedge j_0 \in [P, W_0+P) \\ 0 & \text{Otherwise} \end{cases} . \quad (20)$$

Therefore, once we know (d_0, i_0, j_0), calculate (d_0, i_0', j_0') is easy, see Alg. 2.

4 Experiments

Experiments are done on MNIST using a modified LeNet structure. See Tab. 2. The comparasions of running times can be seen in Tab. 3, 4.

My implementaion is 20.6% faster than Caffe engine. Both forward pass and backward pass of convolution layer need im2col, but Caffe's im2col is somehow slow, which has two-fold **for** loops inside each CUDA kernel loop. On the other hand, my implementation of im2col is simply and efficient, which does not have any **for** loop inside the CUDA kernel loop.

5 Conclusion

In this article, we proposed a simple and efficient implementation of im2col, which can take place the Caffe's implementation. In the experiment of LeNet, our implementation is 20.6% faster compared with Caffe.

References

[1] Jia, Yangqing, et al. "Caffe: Convolutional architecture for fast feature embedding." Proceedings of the 22nd ACM international conference

Table 2: A modified LeNet structure.

Name	Output Channels	Kernel Size	Stride	Pad
conv1	20	5×5	1	0
relu1	-	-	-	-
pool1	-	2×2	2	0
conv2	50	5×5	1	0
relu2	-	-	-	-
pool2	-	2×2	2	0
fc3	500	-	-	-
relu3	-	-	-	-
drop3	-	-	-	-
fc4	10	-	-	-

Table 3: Overall speed of different implementations over different batch size @ NVIDIA K80. Whereas "My Implementation" is the one I rewrite im2col.

Implementations	32	64	128	256
Clean, cuDNN Engine	510.0 iter/s	374.0 iter/s	262.0 iter/s	157.0 iter/s
Clean, Caffe Engine	85.2 iter/s	44.3 iter/s	22.8 iter/s	11.6 iter/s
My Implementation	100.1 iter/s	53.3 iter/s	27.2 iter/s	13.7 iter/s

Table 4: Forward and backward time of different implementations @ batch size = 128, NVIDIA K80.

Implementations	conv1 forw.	conv1 back.	conv2 forw.	conv2 back.
Clean, cuDNN Engine	0.32 ms	0.33 ms	0.43 ms	1.14 ms
Clean, Caffe Engine	6.01 ms	11.49 ms	10.90 ms	14.00 ms
My Implementation	4.63 ms	10.26 ms	9.54 ms	10.94 ms

on Multimedia. ACM, 2014.

[2]. LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.

[3]. Wu, Jainxin. "Introduction to Convolutional Neural Networks". 2016.