From Matrix Multiplication to Deep Learning: Are We Tuning the Same Kernel?

Yaohung M. Tsai, Jakub Kurzak
Piotr Luszczek, Jack Dongarra

Innovative Computing Laboratory
University of Tennessee, Knoxville

Apr. 15, 2019
9th JLESC Workshop

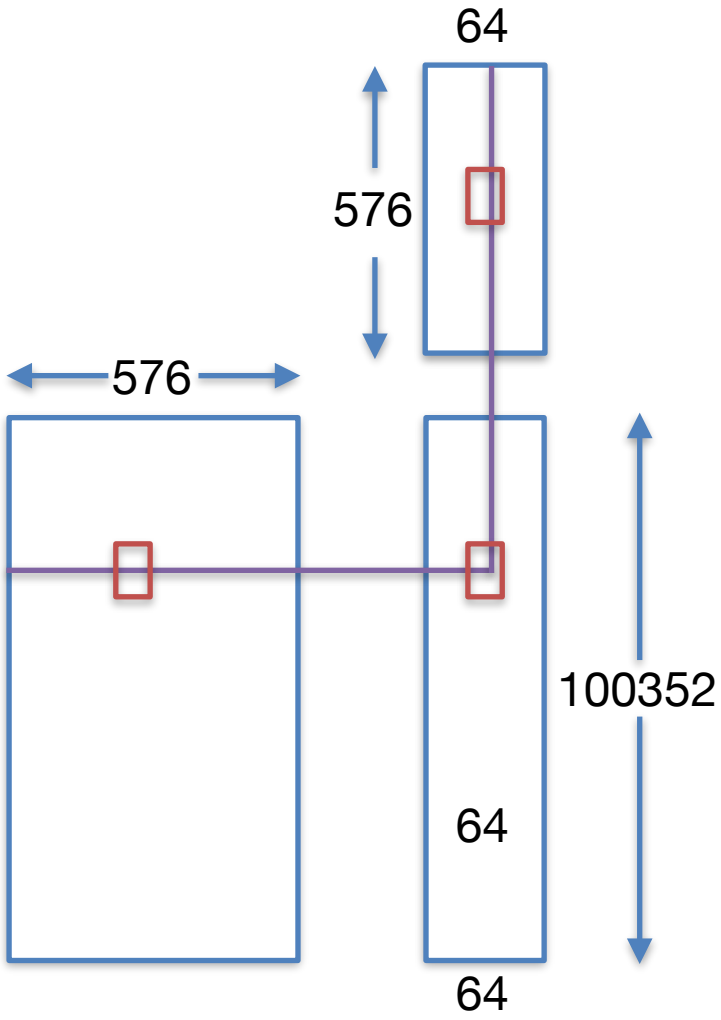
Motivation

- Venders provide low-level optimized libraries (Intel MKL, Nvidia cuBLAS/cuDNN) can usually obtain near-peak performance.
- We still see rooms for improvement for special problem sizes, custom/fused operations.
- Convolutional layers from deep learning were usually casted into GEMMs (GEneral Matrix Multiplication).
- But how similar are they? Can we apply the tuning result from one to the other?

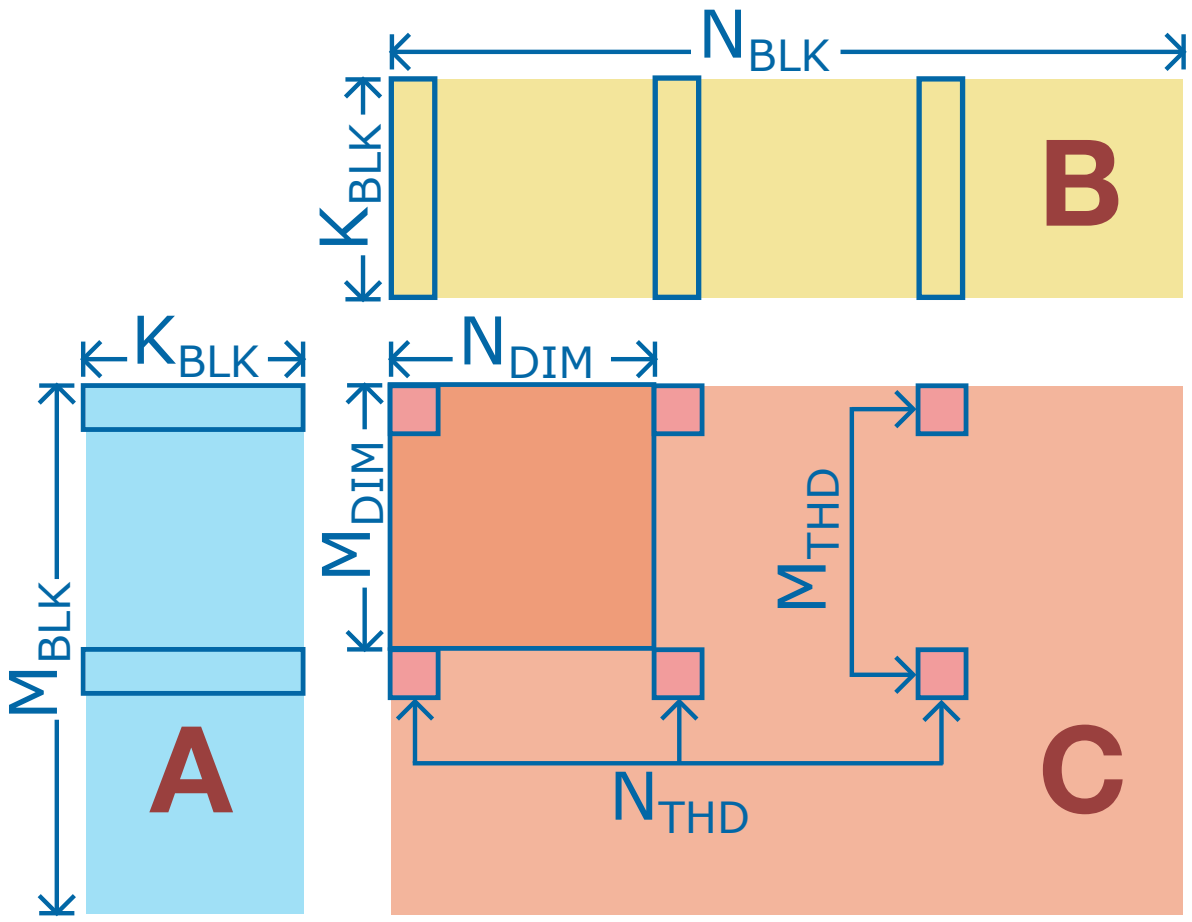
Let's Play with GPU GEMM, First

- Compute $C \leftarrow \alpha AB + \beta C$
- GPU GEMM kernel utilizes shared memory to reuse the data.
- Performance is coming from FMA (fused multiply–add) operations:

$$c \leftarrow a \times b + c$$

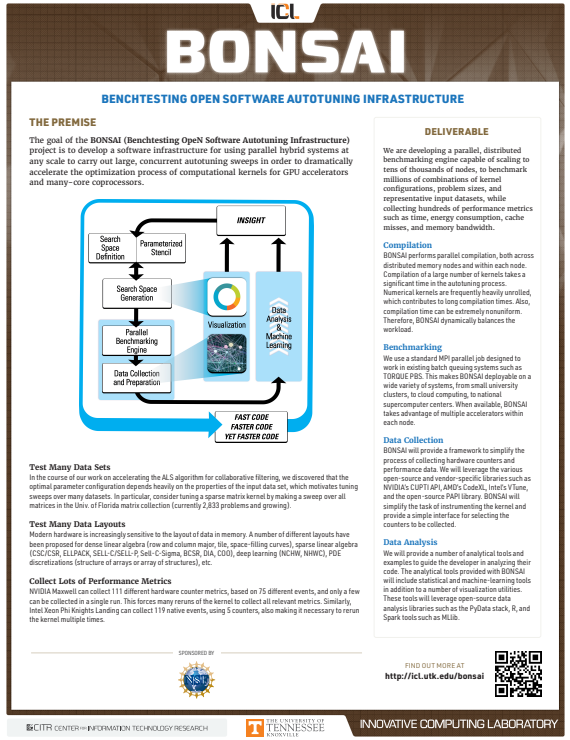


Search Space



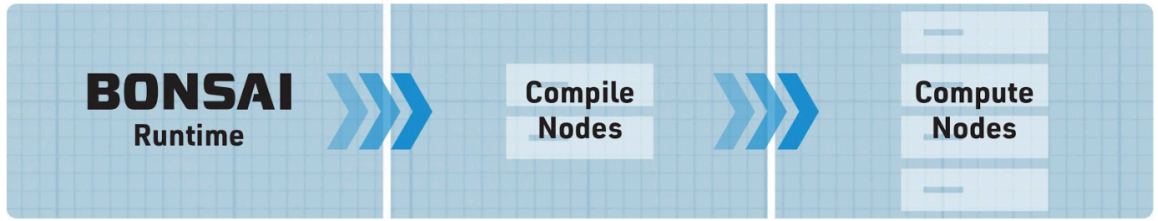
- Hardware level constraints: Ensure there are enough registers and shared memory spaces with minimum occupancy.
- Software level constraints: Problem sizes are divisible by tile sizes to remove imbalance workload.
- Algorithm level constraints: Constraints from the kernel to remove boundary check and utilize all threads.

BONSAI Autotuning Framework



LANAI: LANguage for Autotuning Infrastructure

BONSAI Distributed Runtime



We are using a 4 nodes cluster with 4 Nvidia GTX1060s (Pascal) on each node for all the following results.

ST A1.3 BONSAI: Benchmarking OpeN Software Autotuning Infrastructure *Jakub Kurzak*

GEMM Tuning Result

DIM_M	DIM_N	BLK_M	BLK_N	BLK_K	INDEX	Status	Error	Time	Gflops	Error
Integer	Integer	Integer	Integer	Integer	Integer	String	String	Real	Real	Real
Compile	Compile	Compile	Compile	Compile	Compile	Output	Output	Output	Output	Output
16	8	80	64	16	894	Success	None	0.002331	3174.07	3.69619E-07
16	8	64	64	16	892	Success	None	0.00239648	3087.34	3.69619E-07
8	8	72	64	8	532	Success	None	0.00242787	3047.43	3.69619E-07
32	8	128	64	32	1195	Success	None	0.00245839	3009.6	3.69619E-07
8	8	48	64	8	526	Success	None	0.00247354	2991.16	3.69619E-07
8	8	56	64	8	528	Success	None	0.00249576	2964.53	3.69619E-07
8	8	64	64	8	530	Success	None	0.00249576	2964.52	3.69619E-07
16	8	80	64	32	969	Success	None	0.00251015	2947.53	
32	8	192	64	32	1199	Success	None	0.00254504	2907.13	
32	8	224	64	32	1201	Success	None	0.00255048	2900.92	
16	8	128	64	16	900	Success	None	0.00255777	2892.66	
8	16	88	64	16	1725	Success	None	0.00256759	2881.59	

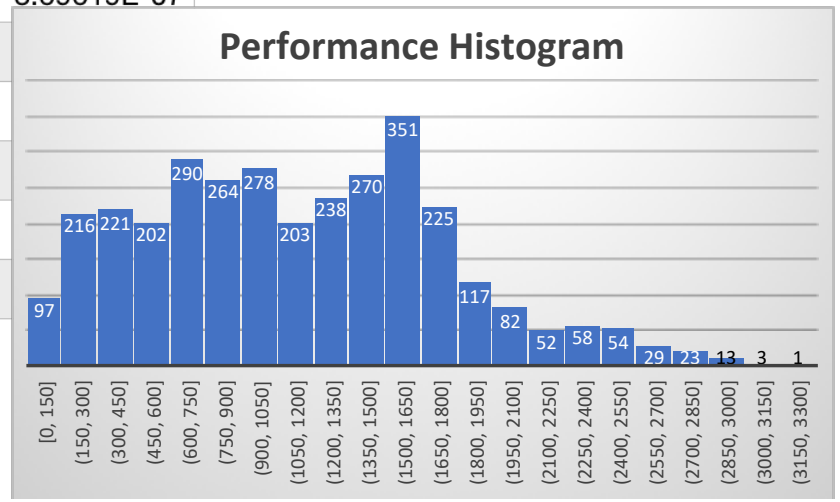
Nvidia GTX 1060

Peak: 3855+ GFlop/s

cuBLAS: 2980 GFlop/s

2945 Tuning points

Done in 5 minutes!!

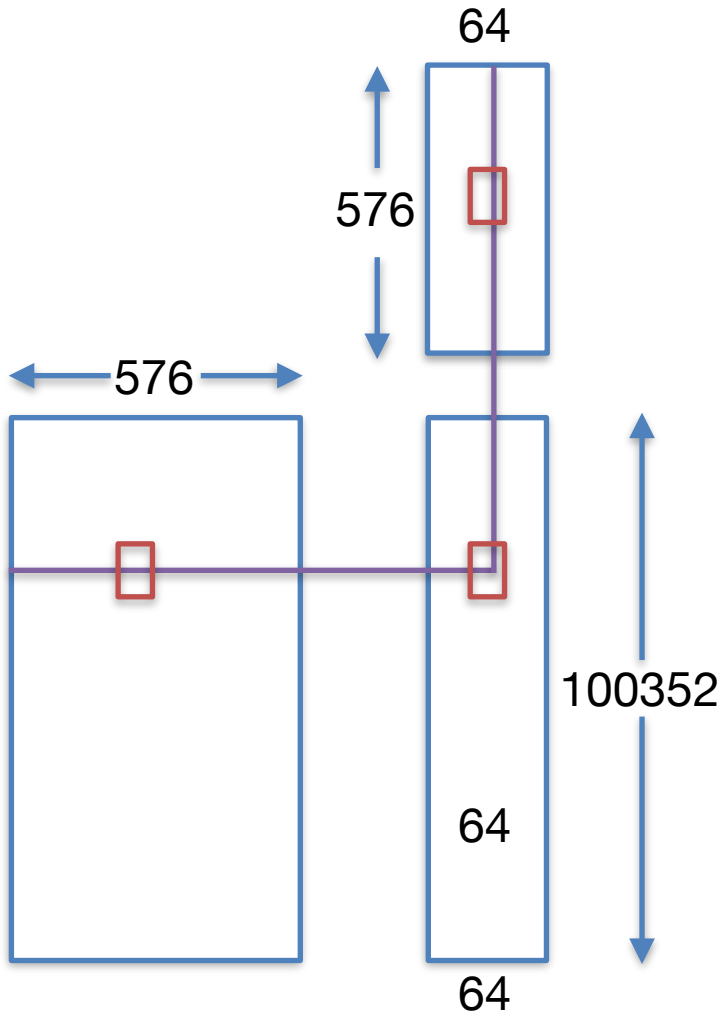


Some Twist?

- Pairwise Euclidean Distance
- Common Operation in Machine Learning

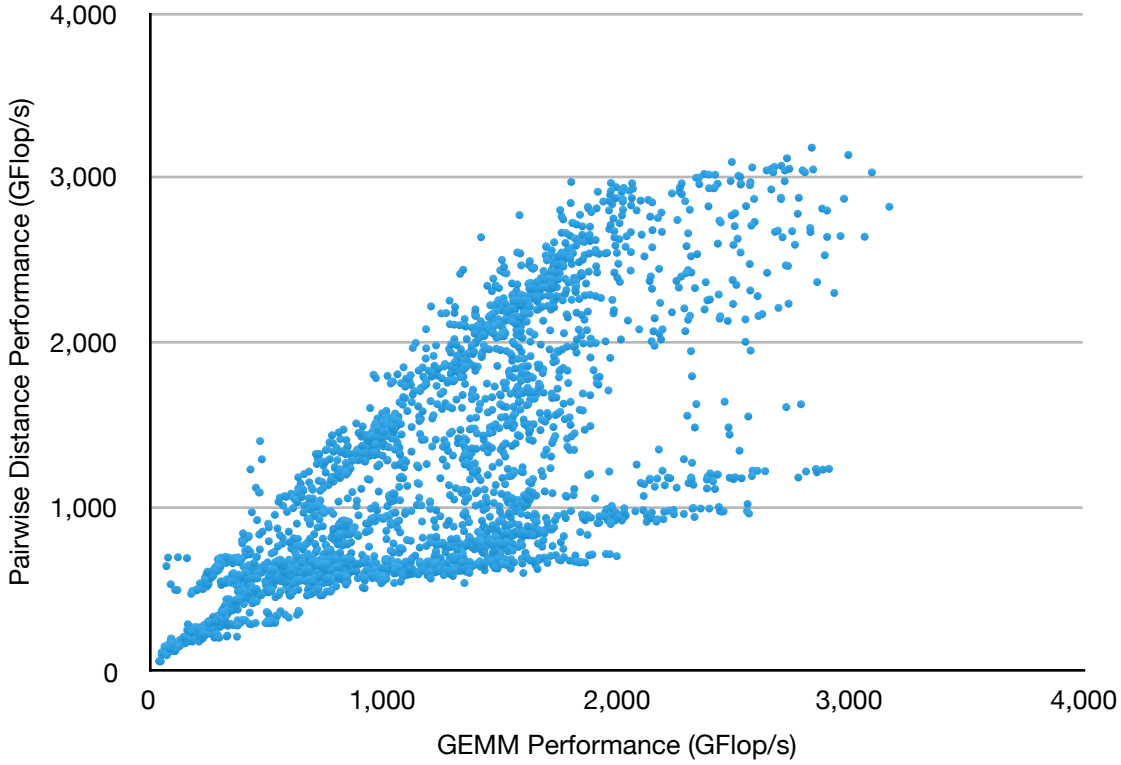
$$C(i,j) += A(i,k) \times B(k,j)$$

$$C(i,j) += (A(i,k) - B(k,j))^2$$



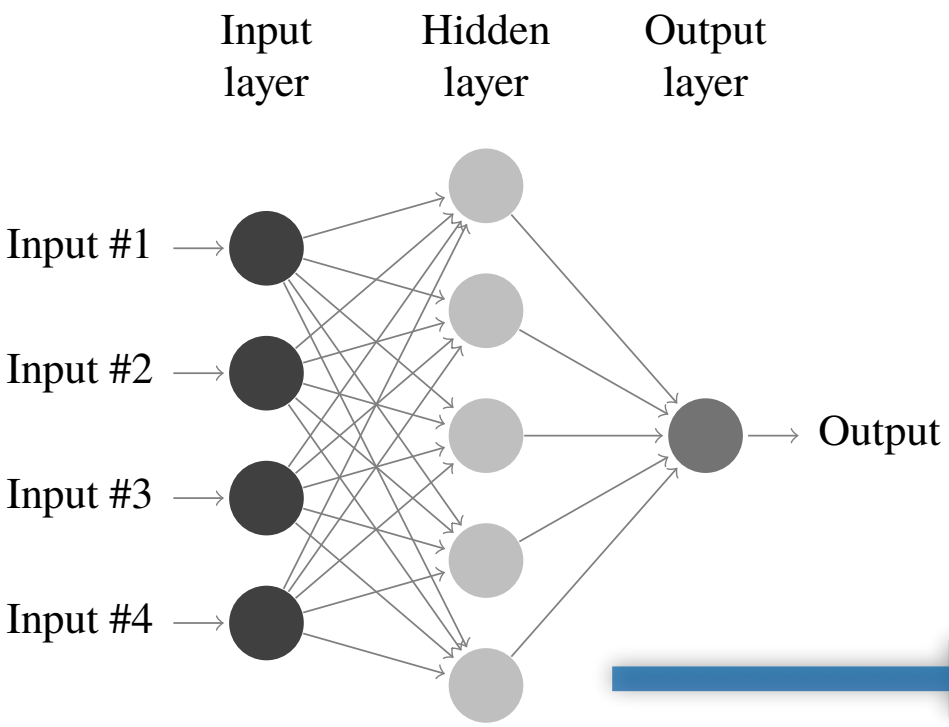
Are they still similar?

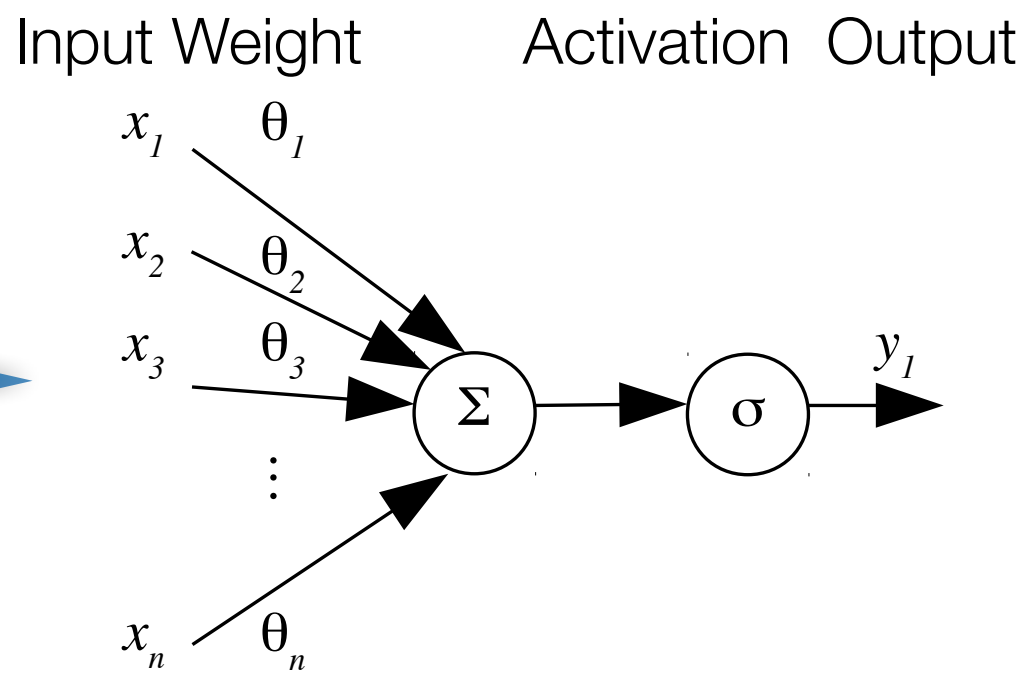
- Each dot represents a set of tuning parameters. x and y axis shows performance from 2 GPU kernels.
- There are cases that behave very differently between two kernels.



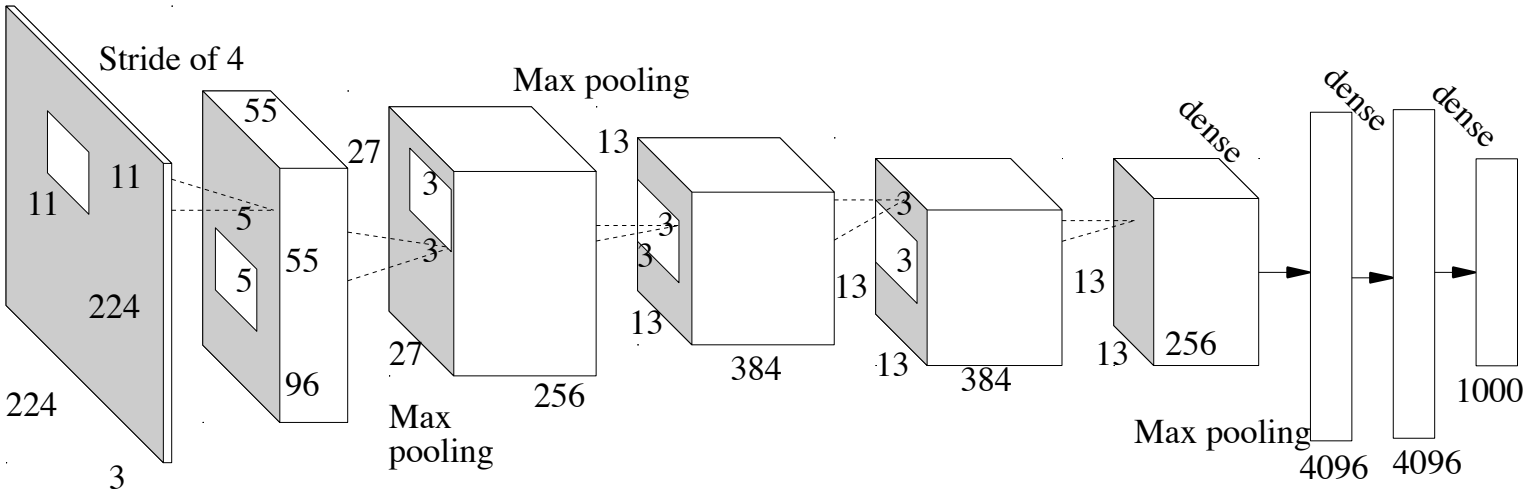
DIM_M	DIM_N	BLK_M	BLK_N	BLK_K	INDEX	GEMM Gflop/s	DIST Gflop/s
16	16	176	64	16	2158	2810.79	3031.2
16	16	176	64	32	2234	2809.41	1616.88

Deep Learning: Background





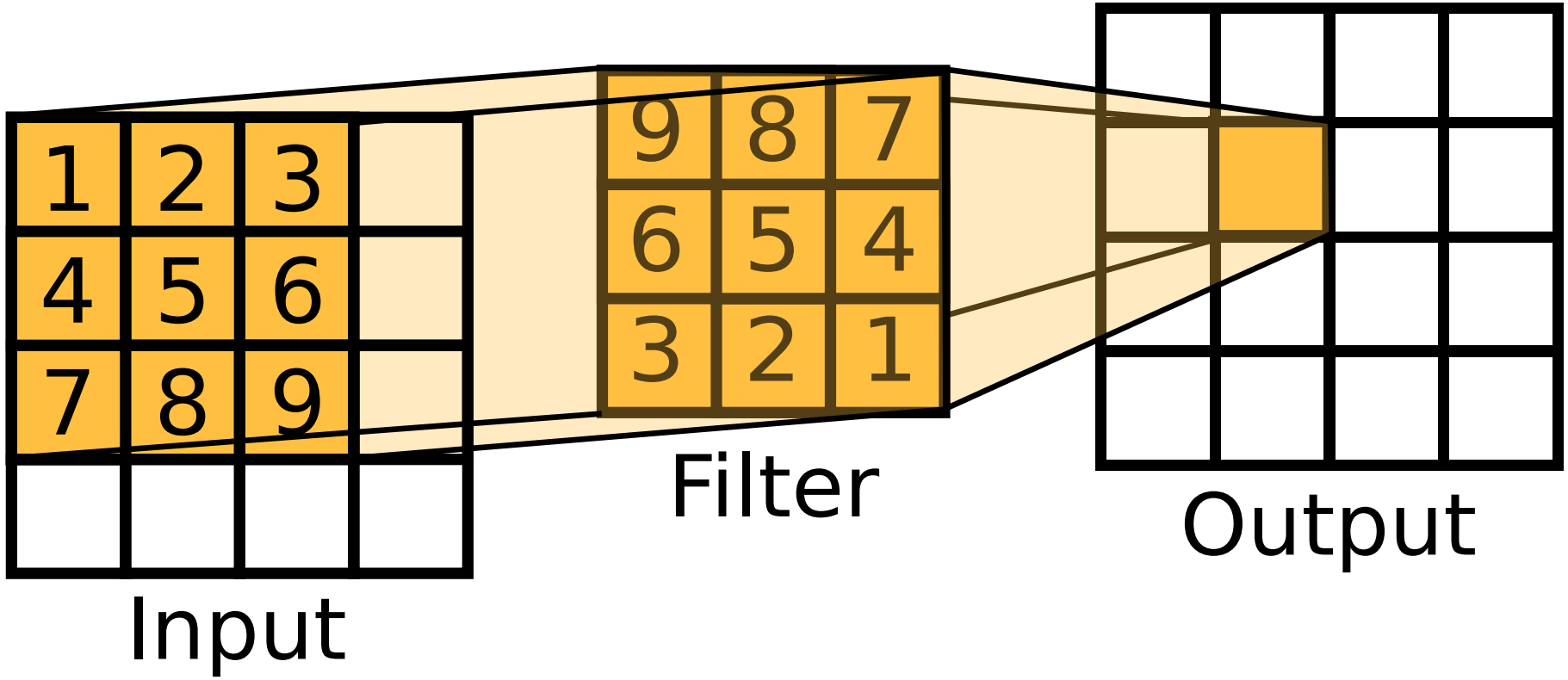
Deep Learning: DNN, Training, Inference



Alexnet for image classification.
Inference: Feed image into network and get the result likelihood vector.
Training: Back propagate the error and update the weights within layers.
Convolutional layers and fully connected layers are the compute intensive parts.

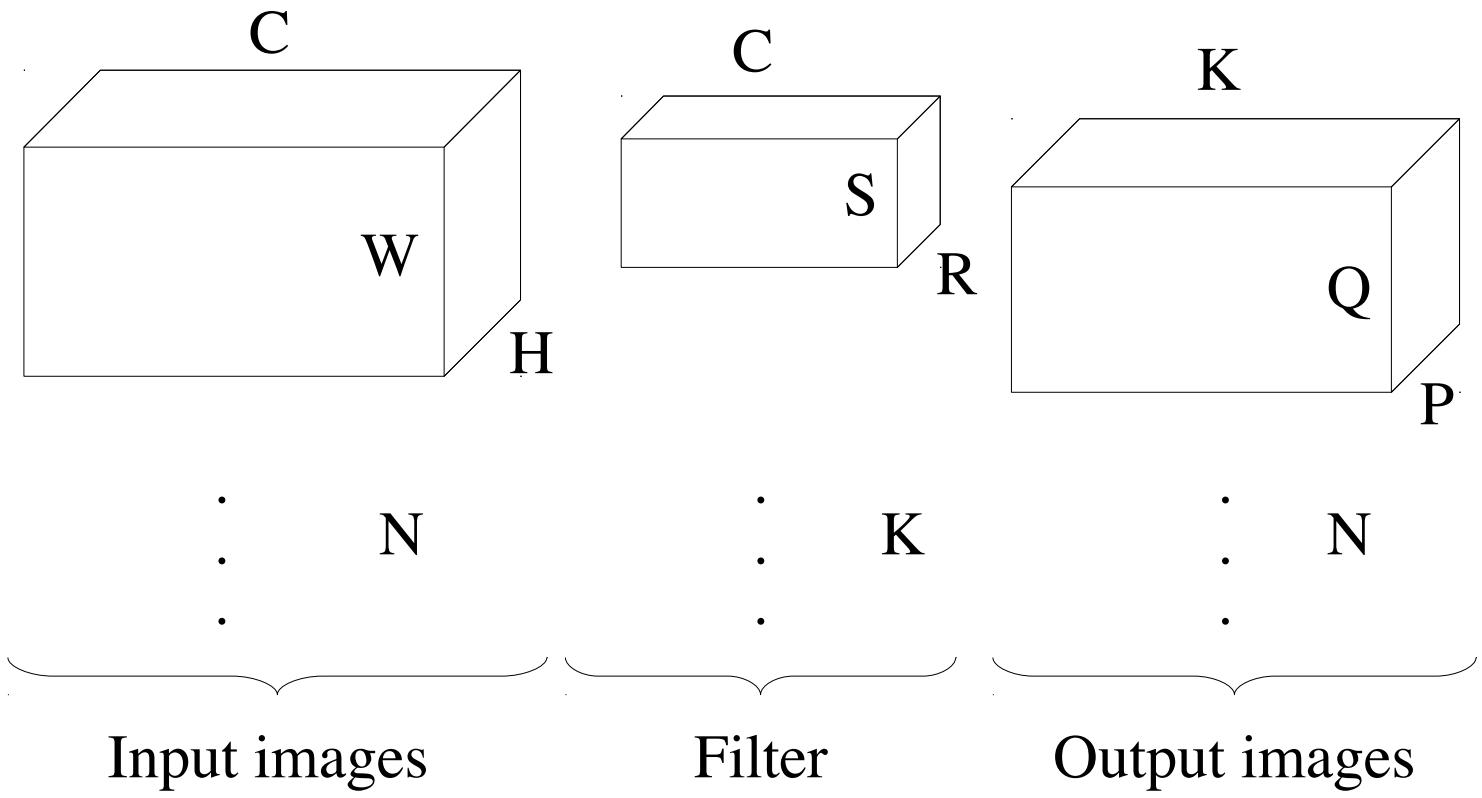
Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

Discrete 2D Convolution



$$\texttt{Output}(x,y) = \Sigma_{i=0}^{r-1}\Sigma_{j=0}^{s-1}[\texttt{Input}(x-i,y-j) \times \texttt{Filter}(i,j)]$$

Dimensions of Convolutional Layer

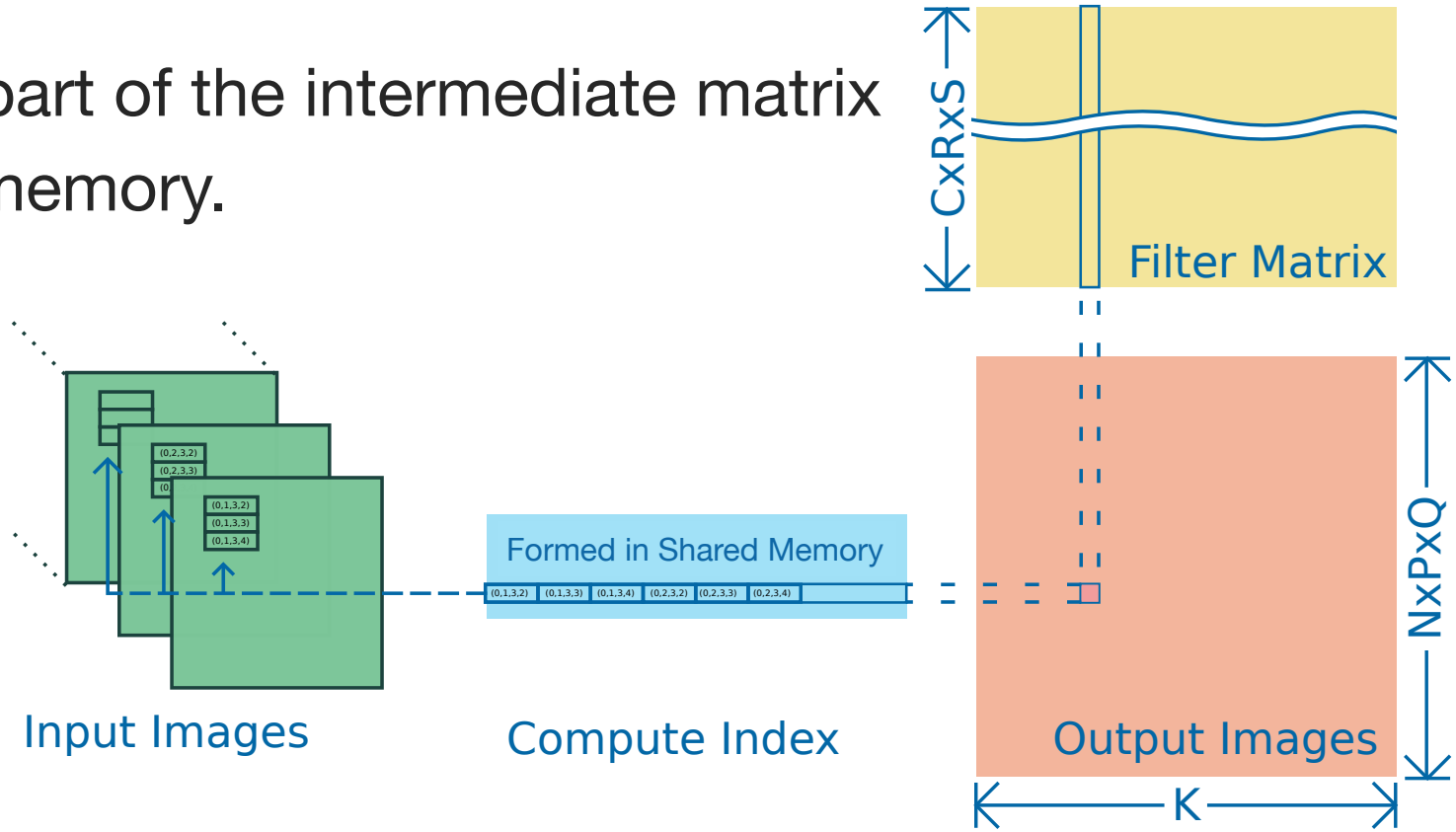


- H x W 2D Input
- N: Batch size
- C: Input features
- P x Q 2D Output
- K: Output features
- R x S 2D Filter

$$O^{(n)}(k) = \Sigma_c \texttt{conv2}\left(I^{(n)}(c), F^{(c)}(k)\right)$$

GPU Kernel Implementation

- Construct part of the intermediate matrix in shared memory.



Tsai, Yaohung M., et al. "Performance-portable autotuning of opencl kernels for convolutional layers of deep neural networks." *Proceedings of the Workshop on Machine Learning in High Performance Computing Environments*. IEEE Press, 2016.

GEMM v.s. Convolutional Layer

```
for (kk = 0; kk < K; kk += blk_k) {
  //Load next submatrix of A into shared memory
  #pragma unroll
  for (n = 0; n < blk_k; n += dim_n_A) {
    #pragma unroll
    for (m = 0; m < blk_m; m += dim_m_A) {
      sA( n+idyA, m+idxA ) = A(m,n);

    }
  }

  //Load next submatrix of B into shared memory
  ...

  __syncthreads();

  //Inner computation loop
  ...
  __syncthreads();

  //Move to next submatrix
  A += blk_k*lda;
  B += blk_k;
}
```

```
for (kk = 0; kk < K; kk += blk_k) {
  //Load next part of Input into shared memory
  int x = p * stride_u + r;
  int y = q * stride_v + s;
  //Check if it's in the padding region
  if( x < pad_h || x >= H+pad_h ||
      y < pad_w || y >= W+pad_w ) {
    #pragma unroll
    for (n = 0; n < blk_k; n += dim_n_A) {
      #pragma unroll
      for (m = 0; m < blk_m; m += dim_m_A) {
        sA( n+idyA, m+idxA ) = 0;
      }
    }
  } else {
    #pragma unroll
    for (n = 0; n < blk_k; n += dim_n_A) {
      #pragma unroll
      for (m = 0; m < blk_m; m += dim_m_A) {
        sA( n+idyA, m+idxA ) =
        A[index+c*N*H*W+r*N*W+s*N+i];;
      }
    }
  }
  //Load next part of Filter into shared memory
  ...
  //Update indices for next submatrix
  s += 16; r += s/S; s = s%S;
  c += r/R; r = r%R;
  __syncthreads();

  //Inner computation loop
  ...
  __syncthreads();

  //Move to next submatrix
  A += blk_k*lda;
  B += blk_k;
}
```

Convolutional Layer Tuning Results

- Using one layer in ResNet: N=128 C=K=64 HxW = 56x56 RxS = 3x3
- The output size and number of Flops is the same as GEMM.

DIM_M	DIM_N	BLK_M	BLK_N	BLK_K	INDEX	Status	Error	Time	GEMM	CONV
Integer	Integer	Integer	Integer	Integer	Integer	String	String	Real	Real	Real
Compile	Compile	Compile	Compile	Compile	Compile	Output	Output	Output	Output	Output
32	8	256	64	32	1058	Success	None	0.0019924	2853.43	3713.49
32	8	224	64	32	1056	Success	None	0.00203969	2909.63	3627.39
32	8	192	64	32	1054	Success	None	0.00207288	2883.96	3569.31
16	8	112	64	16	821	Success	None	0.00215042	2830.04	3440.61
16	8	128	64	16	823	Success	None	0.00215454	2891.44	3434.03
16	8	80	64	16	817	Success	None	0.00216098	3167.65	3423.8
16	4	112	32	16	151	Success	None	0.00218383	1972.38	3387.97
16	8	80	64	32	892	Success	None	0.00221748	2931.88	3336.56
32	8	288	64	32	1060	Success	None	0.00222401	2582.34	3326.76
32	8	160	64	32	1052	Success	None	0.00223656	2644.34	3308.1

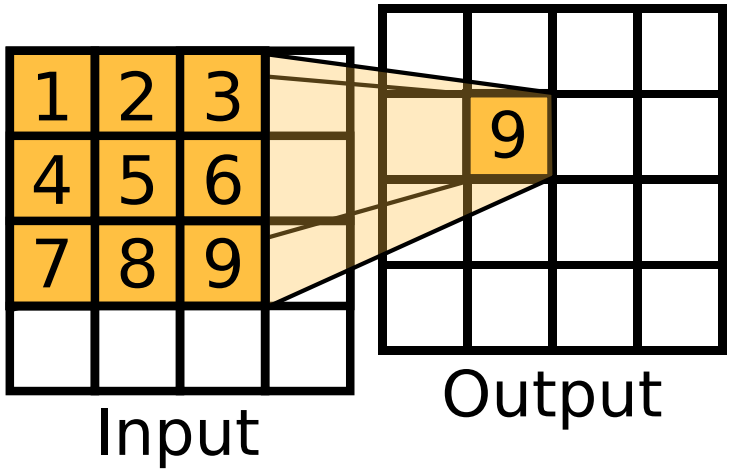
- Note: Winograd convolution has better performance on 3x3 filter layers.

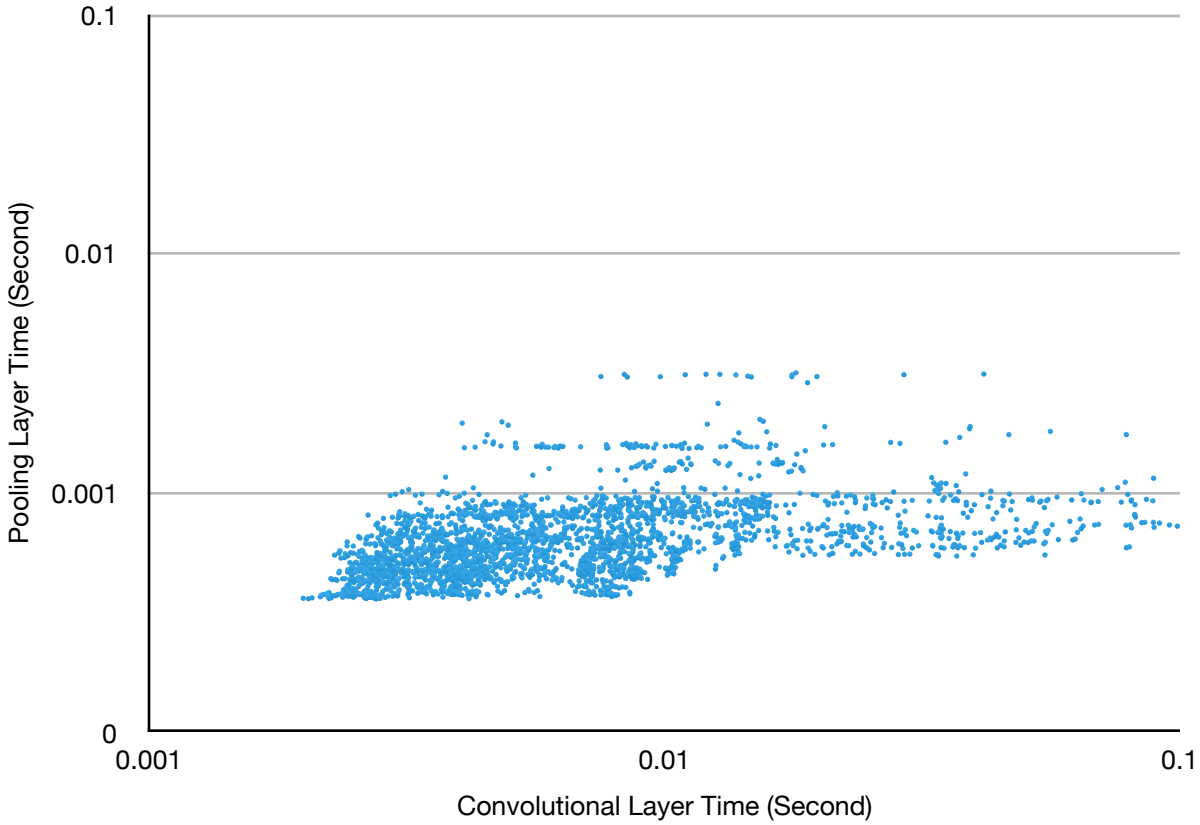
GEMM v.s. Convolutional Layer

- Same story: In general they are proportional to each other. But still some of kernels are good for convolutional layer but not for GEMM.

BOUNS: Pooling Layer

- Pooling layer: output maximum or average of the pooling window
- Similar memory read pattern
- Almost no computation





Lesson Learned and Future Work

- Modeling GPU kernel performance is hard.
- Models still could be used as guidelines for tuning sweep design.

- Improve our BONSAI framework.
- Machine learning model backend for BONSAI.

- This work is funded by NSF through Award Number 1642441.

ST A1.3 BONSAI: Benchmarking OpeN Software Autotuning Infrastructure *Jakub Kurzak*