Friedrich-Alexander-Universität
Technische Fakultät

RADL – GPU Group

Agenda

01 GEMM Optimization

02 Convolution Optimazition

03 CNN Benchmark

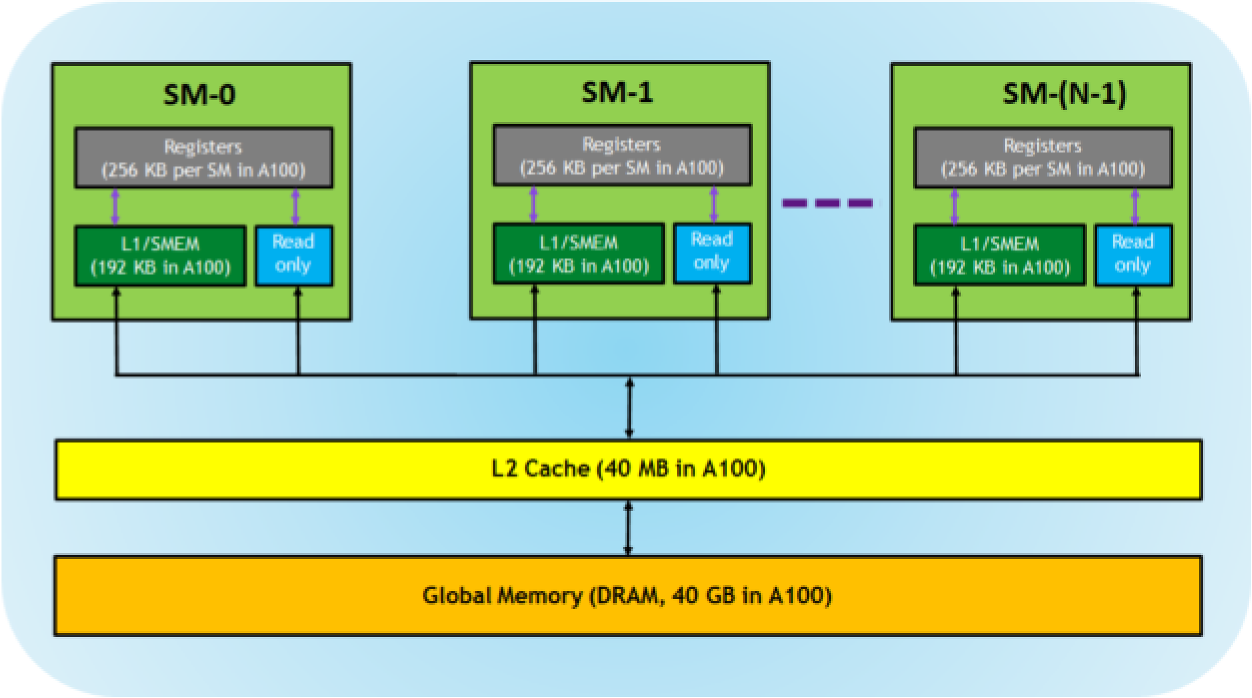
04 Quantization

05 Pruning

06 Conclusion

GEMM Optimization

Memory hierarchy



The following memories are exposed by the GPU architecture:

Registers; L1/Shared memory (SMEM); Read-only memory; L2 cache; Global memory

Optimization Strategy

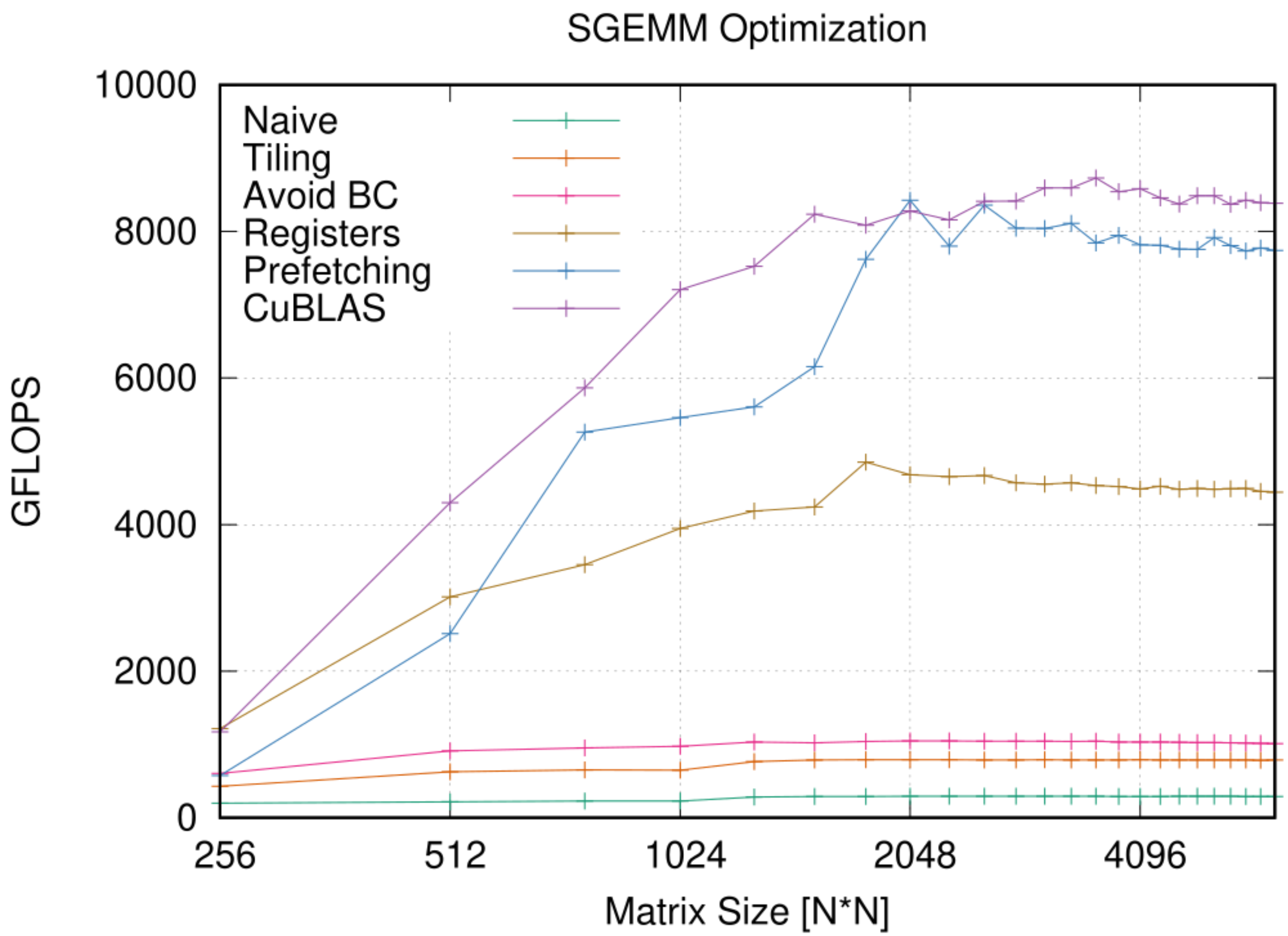
Optimizing GEMM on GPU and CPU platforms share the same idea:

- Hide the memory latency with massive parallelism
- Cache-/register-level data re-use
- Manual prefetching.

The major operations in convolutional neural networks consist of matrix multiplications in both the convolutional and the fully-connected layers.

Benckmarkting

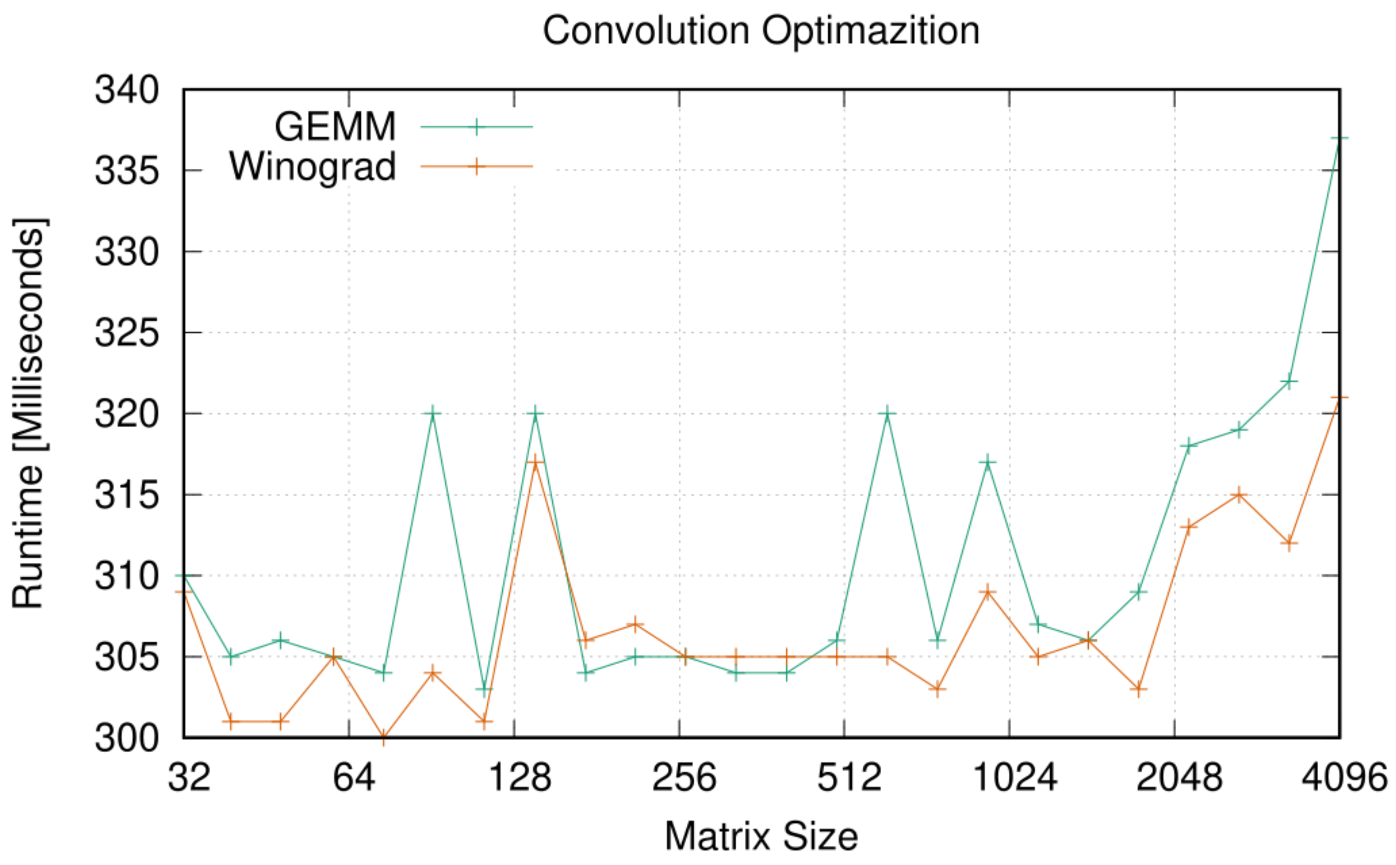
RTX 2080



Convolution Optimazition

Benckmarkting

RTX 2080

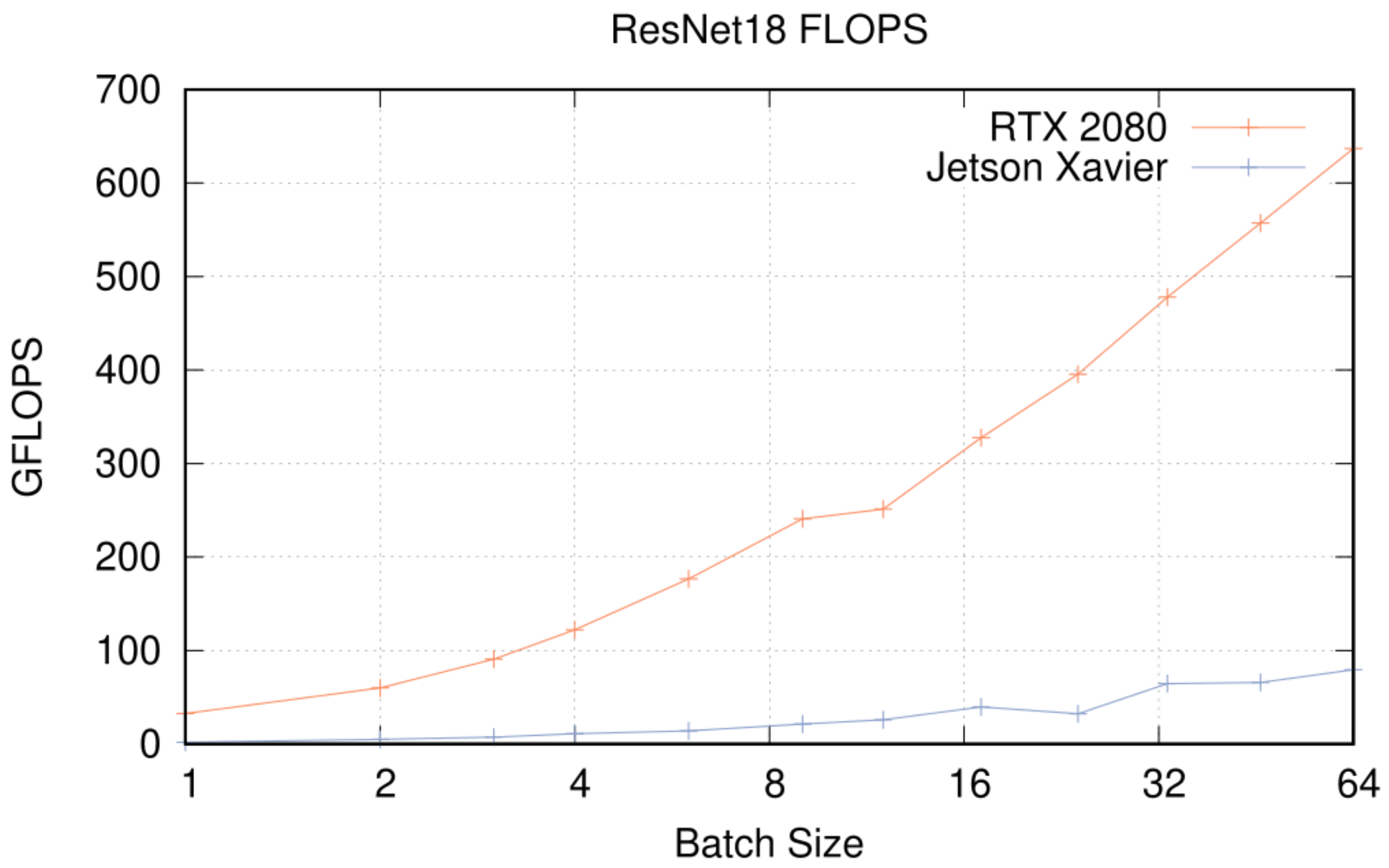


CNN Benchmark

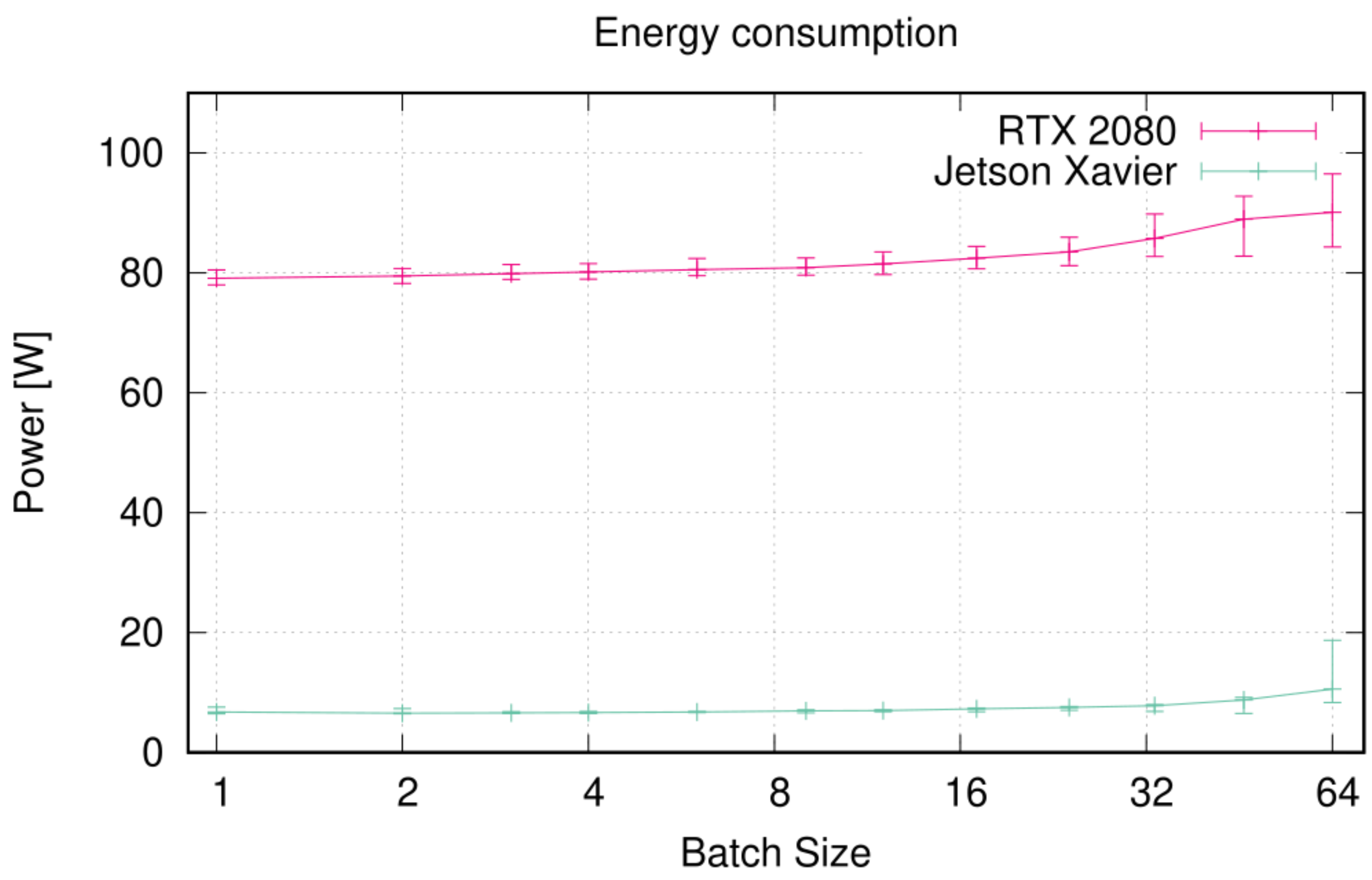
Benchmarketing

- Implemented FC, Conv, ReLU, BN and a Pooling Layer using CUDA
- Benchmarked on GPGPU(RTX 2080) and Embedding GPU(Jetson Xavier)
- Compared the throughput and power consumption of two different platforms
- Model: ResNet18
- Dataset: CIFAR-10

FLOPS


ResNet18 FLOPS
GFLOPS
700
600
500
400
300
200
100
0
1
2
4
8
16
32
64
Batch Size
RTX 2080
Jetson Xavier

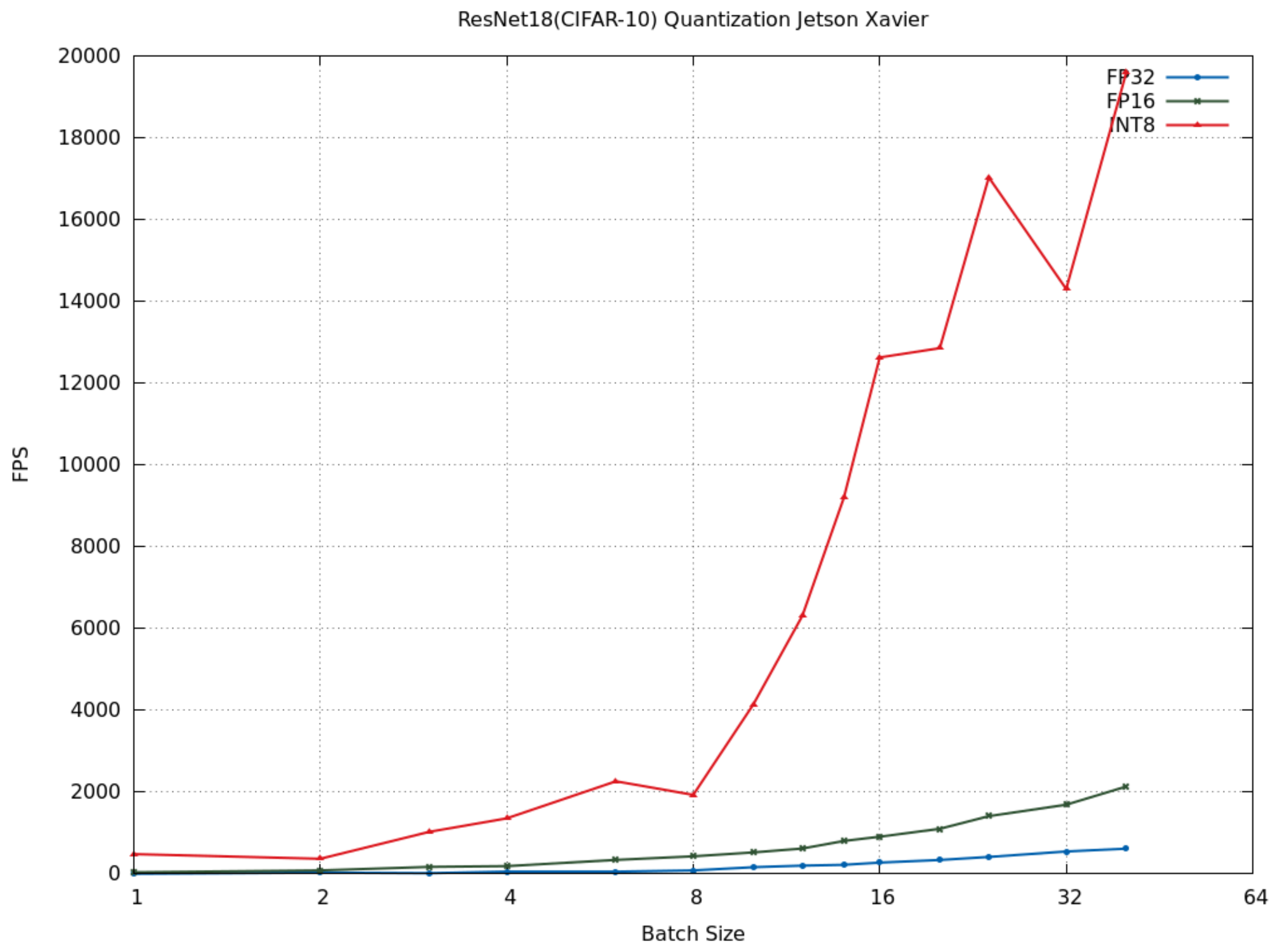
Energy consumption



FLOPS/W

Quantization

Quantization



Pruning

L1/L2 norm

Getting faster/smaller networks is important for running these deep learning networks on mobile devices.

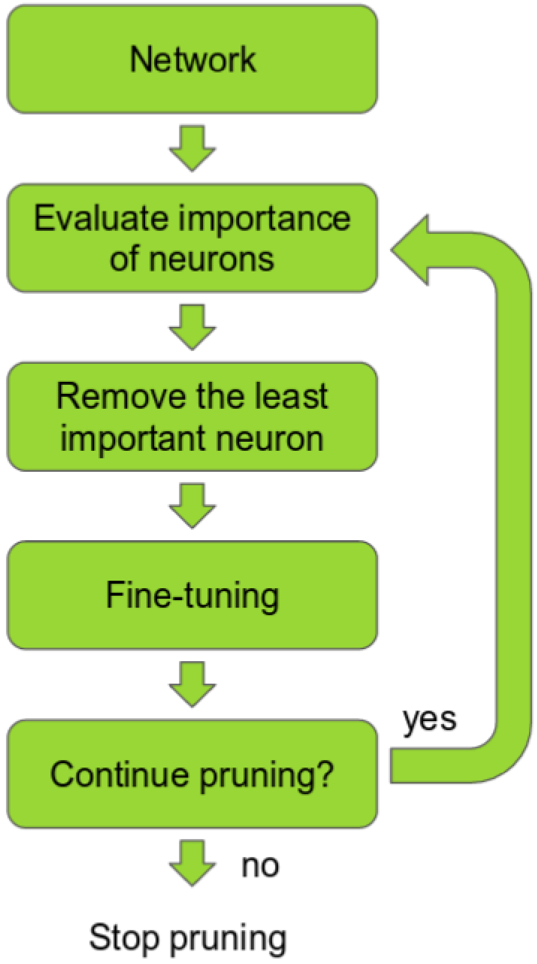
The ranking can be done according to the L1/L2 norm of neuron weights, their mean activations, the number of times a neuron wasn't zero on some validation set, and other creative methods . After the pruning, the accuracy will drop

```
def apply(self, weights, amount=0.0, round_to=1) -> Sequence[int]:  # return index
    if amount<=0: return []
    n = len(weights)

    l1_norm = torch.norm( weights.view(n, -1), p=self.p, dim=1 )

    n_to_prune = int(amount*n) if amount<1.0 else amount
    n_to_prune = round_pruning_amount(n, n_to_prune, round_to)
    if n_to_prune == 0: return []
    threshold = torch.kthvalue(l1_norm, k=n_to_prune).values
    indices = torch.nonzero(l1_norm <= threshold).view(-1).tolist()
    return indices
```

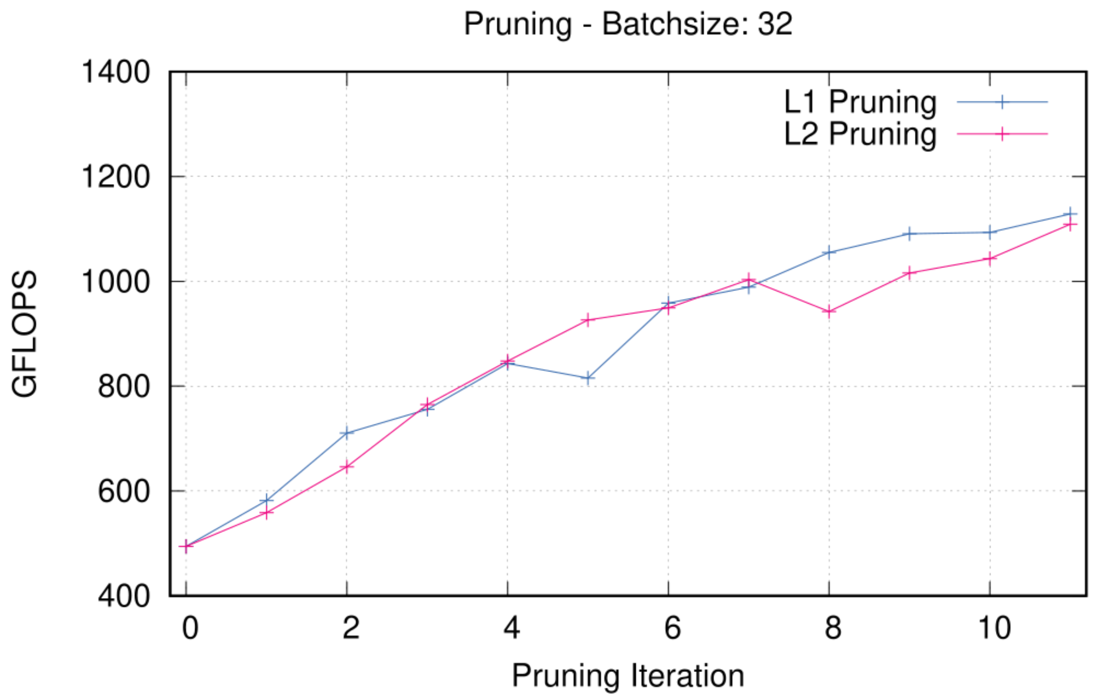
Retraining

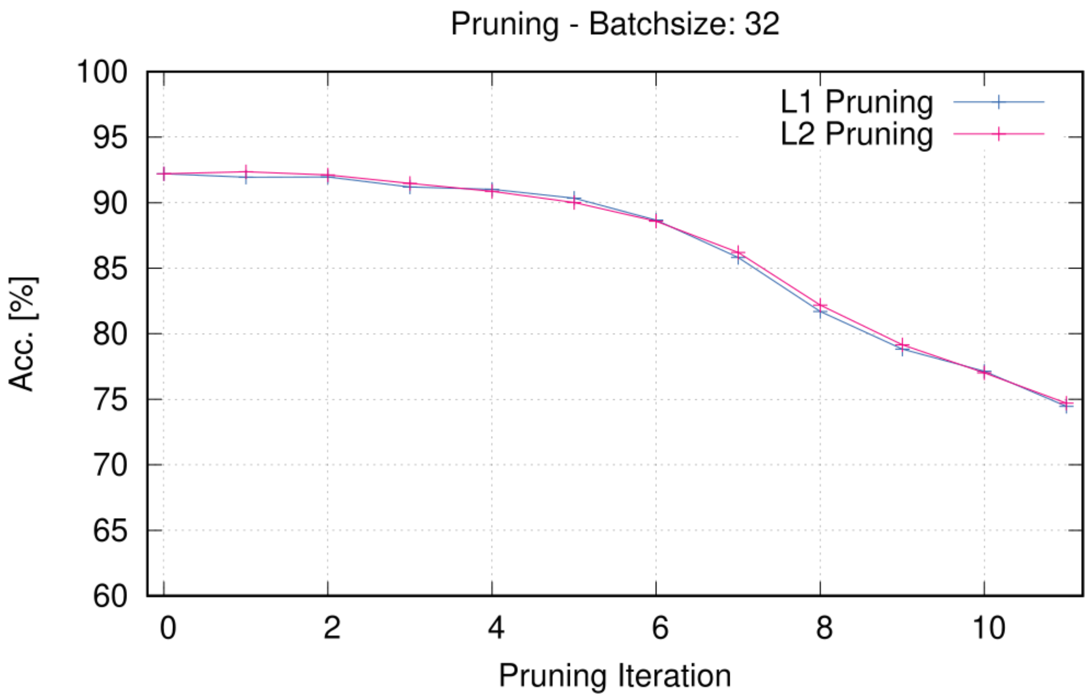
If we prune too much at once, the network might be damaged so much it won't be able to recover.

So, in practice this is an iterative process - often called 'Iterative Pruning': Prune / Train / Repeat.



Pruning Benchmarket

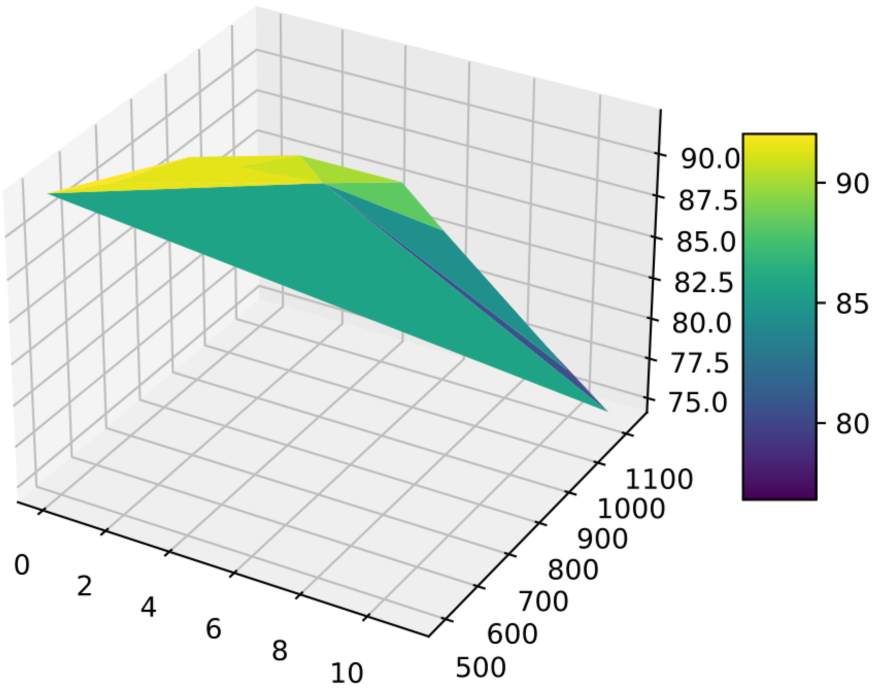




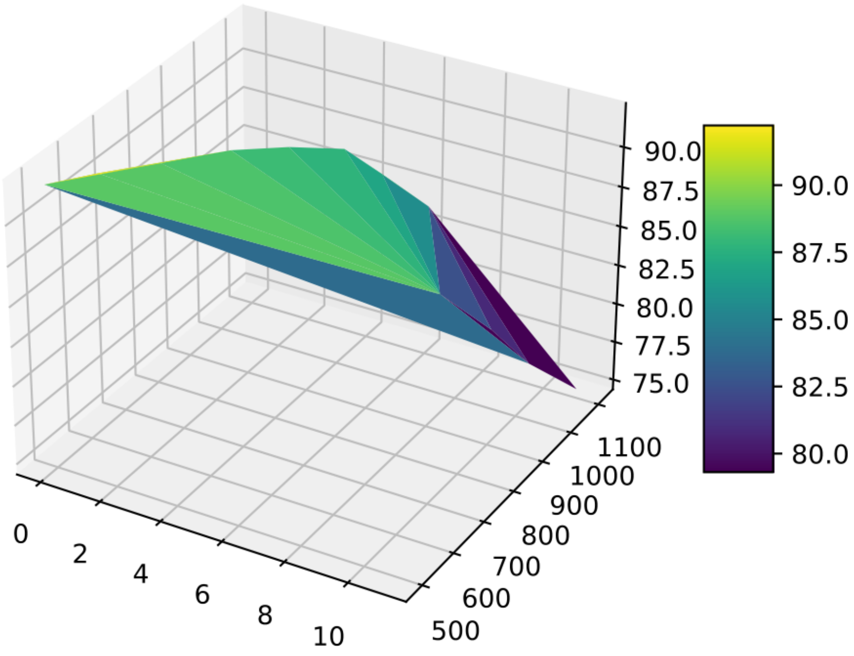
Pruning Benchmarket

Round	Params[K]	Reduced[%]	L1 Acc.[%]	L2 Acc.[%]
0	11181.642	0	92.2	92.2
1	4499.885	59.75	91.94	92.36
2	1904.148	82.97	91.96	92.11
3	850.709	92.39	91.20	91.48
4	408.764	96.34	91.03	90.87
5	210.707	98.11	90.35	90.02
6	118.955	98.93	88.68	88.59
7	70.37	99.37	85.82	86.20
8	46.003	99.58	81.70	82.19
9	32.347	99.71	78.83	79.16
10	23.606	99.78	77.14	77.02
11	17.706	99.84	74.46	74.71

Pruning Benchmarket



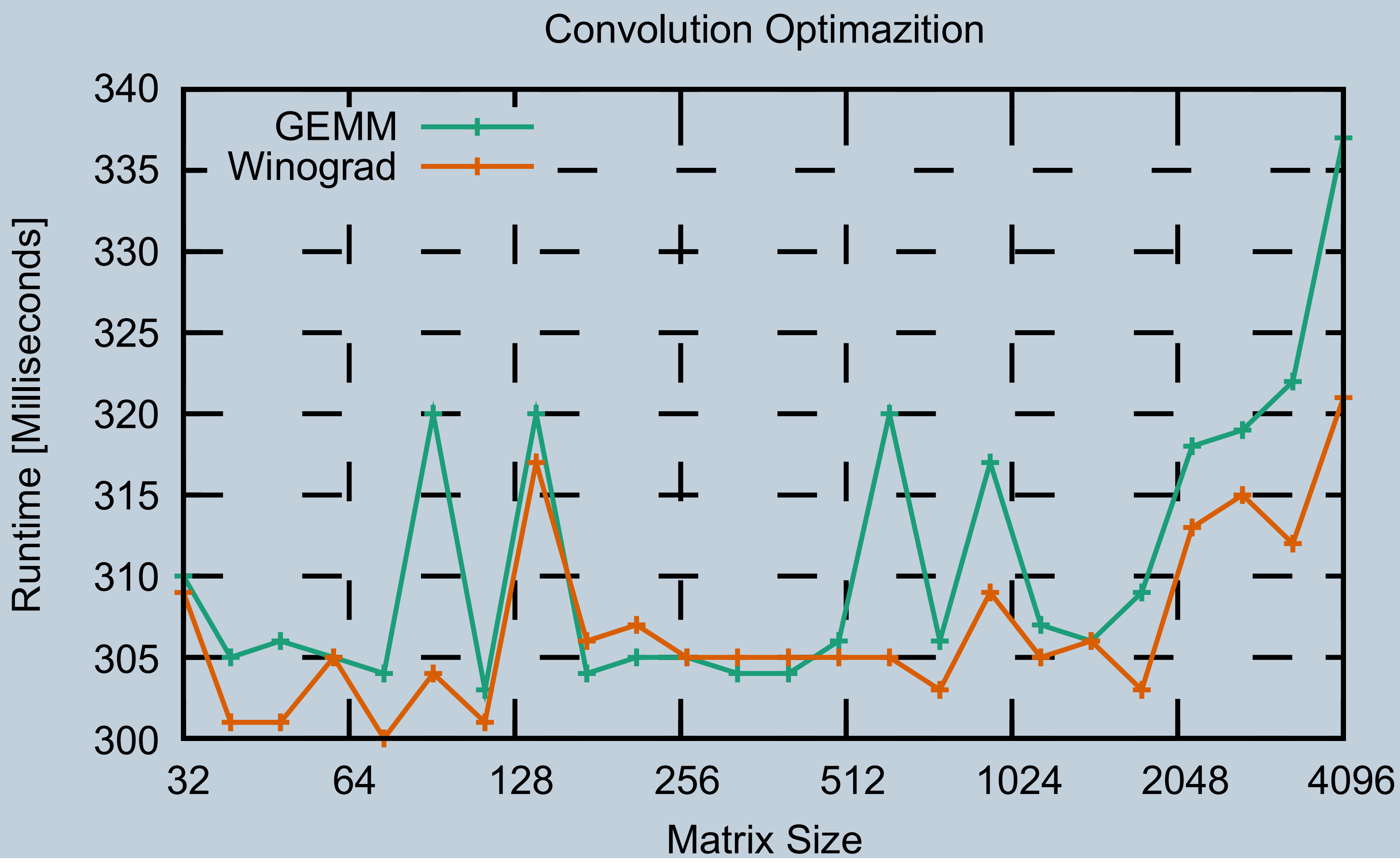
L1 Norm Pruning



L2 Norm Pruning

Conclusion

Conclusion



Convolution Optimazition
GEMM
Winograd
Runtime [Milliseconds]
340
335
330
325
320
315
310
305
300
32
64
128
256
512
1024
2048
4096
Matrix Size

Vielen Dank
für Ihre Aufmerksamkeit!