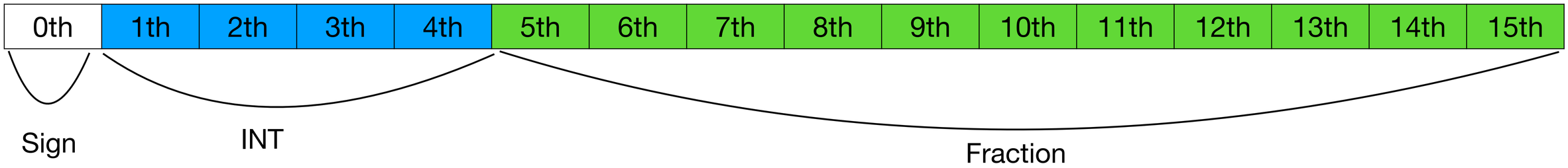
SAF and CNN

What we are exploring:

How the position of Bit Error impacts performance of model.

Quantization

We quantize every weight s to 16 bit.



Models we use:

Lenet, VGG

Observations:

1. For both VGG11 and Lenet5, the first layer is less sensitive to bit error
2. Normalization Layer helps but cannot solve the bit error problem
3. The test accuracy will become random guess even for only 5 percentage of weights are stuck at fault for a particular layer

Example: Save crossbars in Layer7 in VGG19

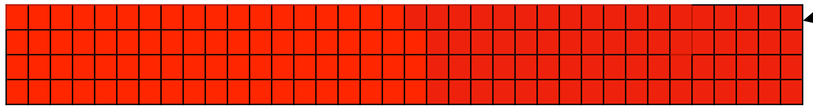
Layer7: (512, 512, 3, 3) = (128*4, 128*36)

144 crossbars with layer sparsity = 1%

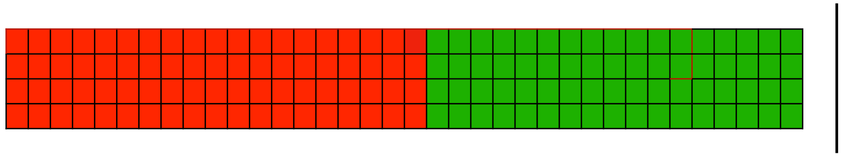
Nonzero Columns: 9044/18432, Nonzero rows: 6366/18432

Each square means a crossbar(128*128)

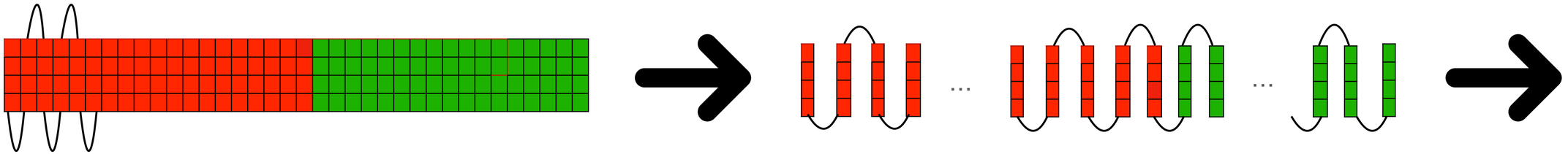
The whole rectangular has 4*36 = 144 crossbars



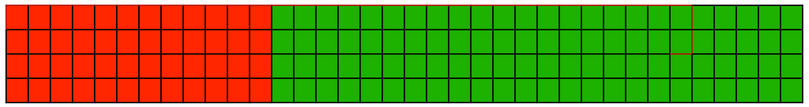
1. In total, move all zero columns forward, we can save 68 crossbars now



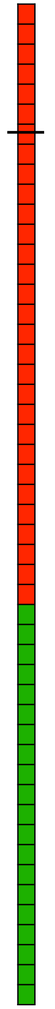
2. Catenate all crossbar and keep move nonzero rows forward



3. Swap rows and move all nonzero rows together



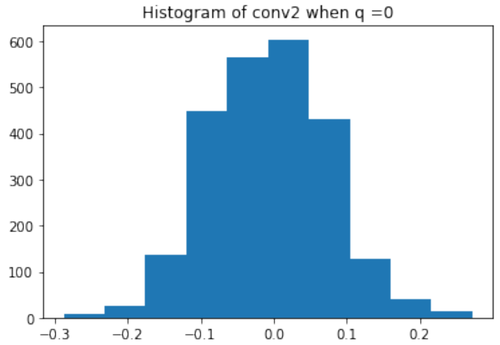
We can save 94 crossbars now



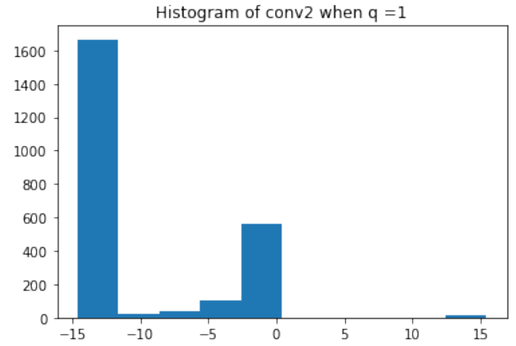
Final Weight distribution for convolution layer2 in Lenet-MNIST

Epochs: 50
Initial learning rate = 0.1
Set q% of weights' the most significant bit as 1 for conv2

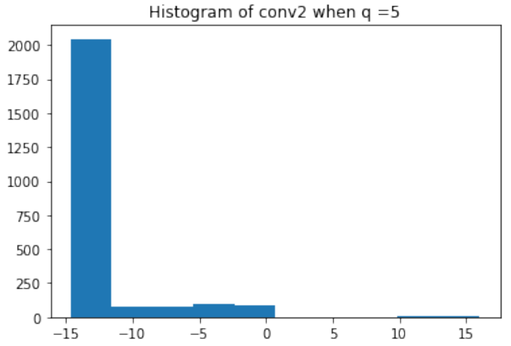
1. Large numbers will lead to change of the distribution
2. For model with normalization, if we have SAF at significant bit, the more epochs we train, the more zeros we will have.
3. Lenet model can be very sparse



Test accuracy = 0.98

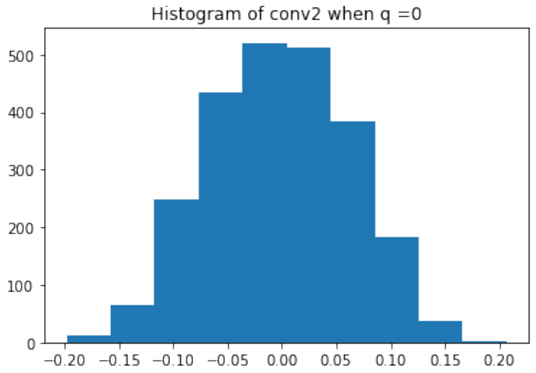


Test accuracy = 0.1 with training loss = 2.3

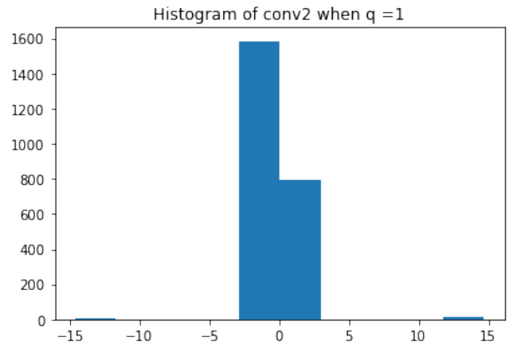


Test accuracy = 0.1 with training loss =nan

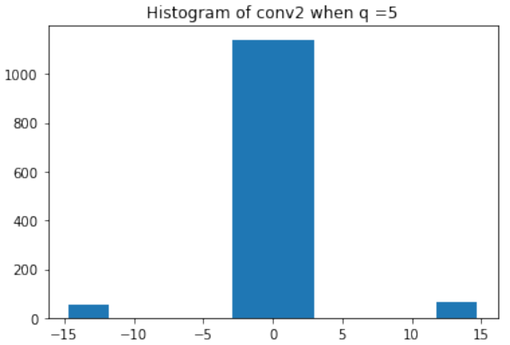
Lenet-MNSIT without normalization



Test accuracy = 0.9805



Test accuracy = 0.9871

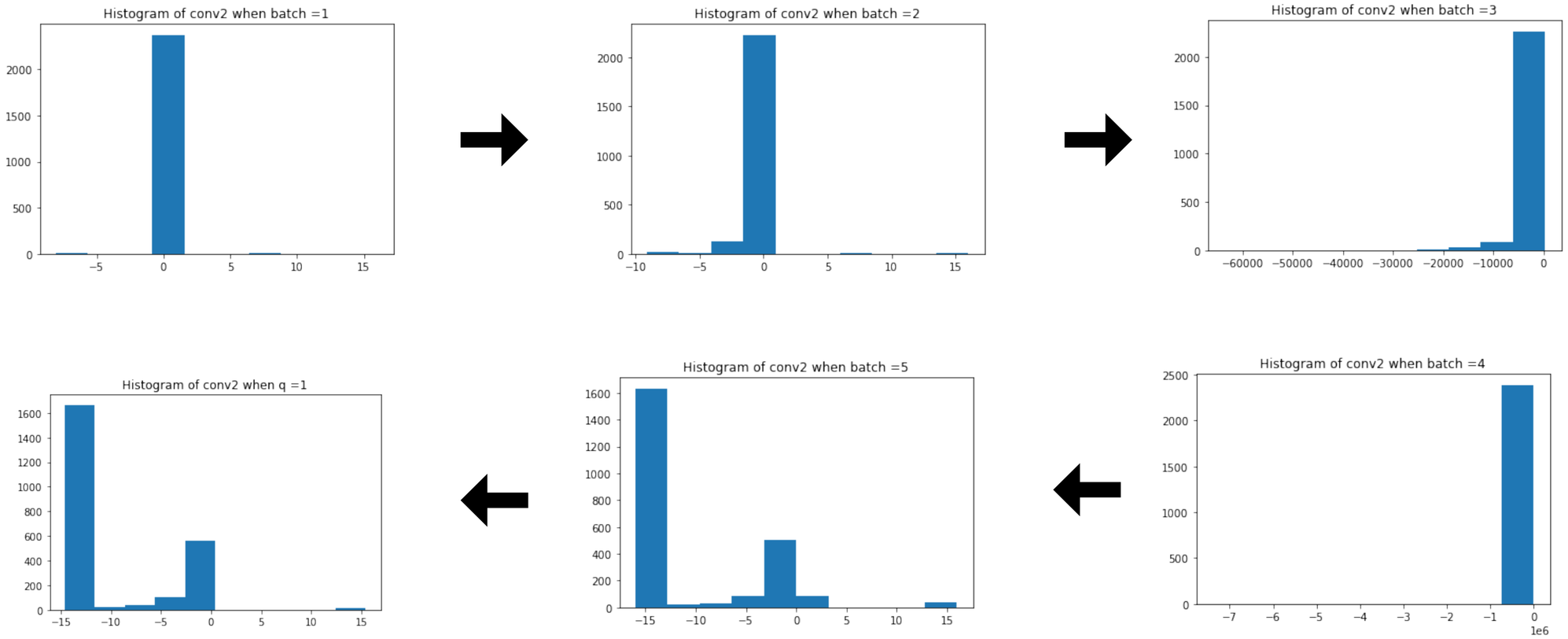


Test accuracy = 0.9766

Lenet-MNIST with normalization

Weight distribution for convolution layer2 in Lenet-MNIST

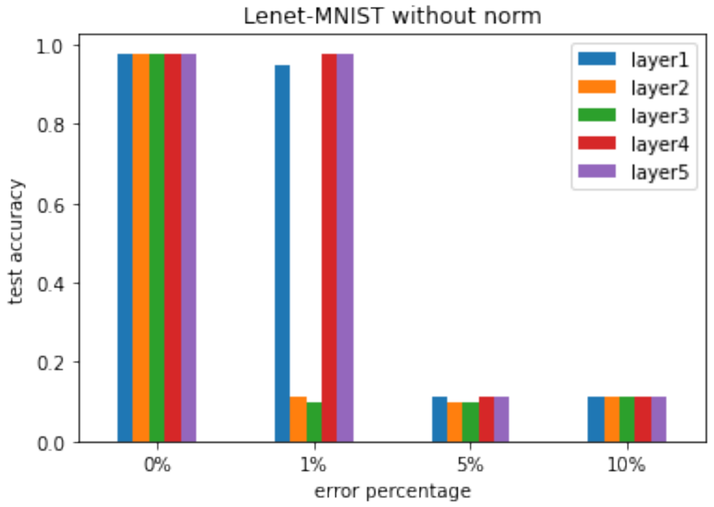
Set 1% of weights' the most significant bit as 1 for layer2. Look at the weights distribution at different batches.
The objective function keep diverging.
After the first epoch, the weight distribution keep the same



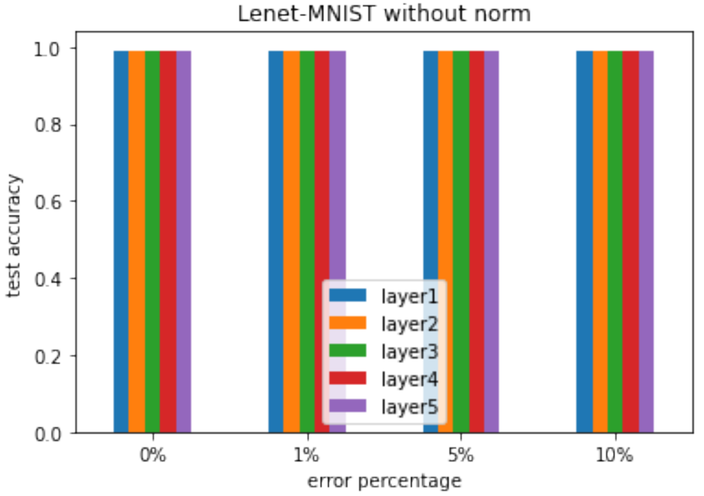
Mask error weight to 0. (Pruning)

Setting:
Initial learning rate = 0.1
Epochs = 50
SDG with momentum = 0
weight_decay = 5e-4
Scheduler: setp_size = 1,
gamma = 0.95
Run in 5 trials with different random seed.

4th of Weight stuck at 1

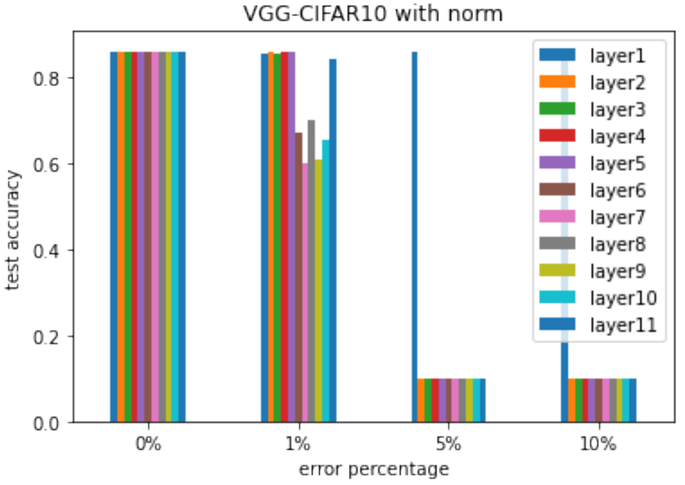


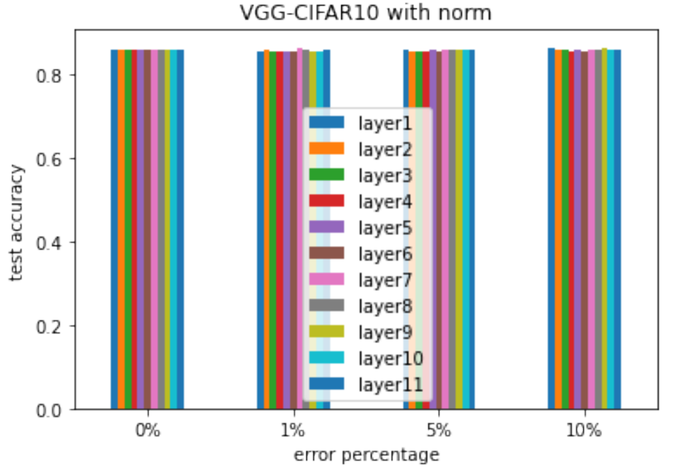
Masking them to 0



After we mask more than 5% of weights for a particular layer, training Loss become nan, test accuracy become random guess.

Objective function diverges to infinity immediately after we set 1th of weight to 1





Recap:

The lottery ticket hypothesis: finding sparse, trainable neural networks

- Main idea: there always exists a sparse subnet that can represent dense network

Iterative pruning with resetting:

1. Randomly initialize a neural network $f(x; m \odot \theta)$ where $\theta = \theta_0$ and $m = 1^{|\theta|}$ is a mask. (where $\theta_0 \sim D_\theta$)
2. Train the network for j iterations, reaching parameters $m \odot \theta_j$.
3. Prune $s\,\%$ of the smallest-magnitude weights, creating an updated mask m' where sparsity of mask $P_{m'} = (P_m - s)\,\%$.(Prune the weights across whole layers)
4. Reset the weights of the remaining portion of the network to their values in θ_0. That is, let $\theta = \theta_0$.
5. Let $m = m'$ and repeat steps 2 through 4 until a sufficiently pruned network has been obtained.

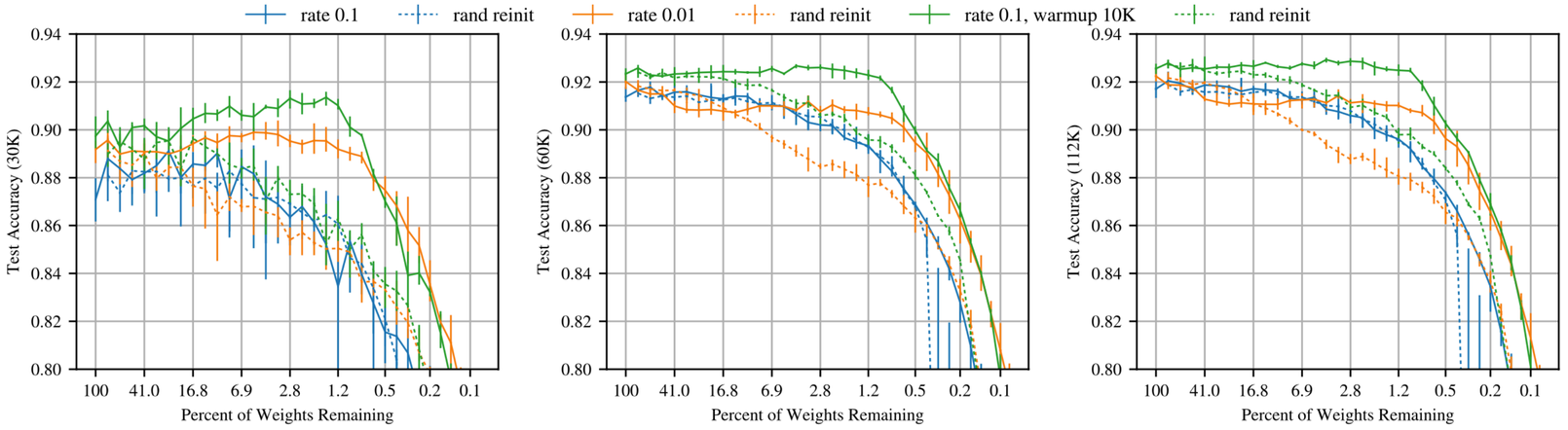
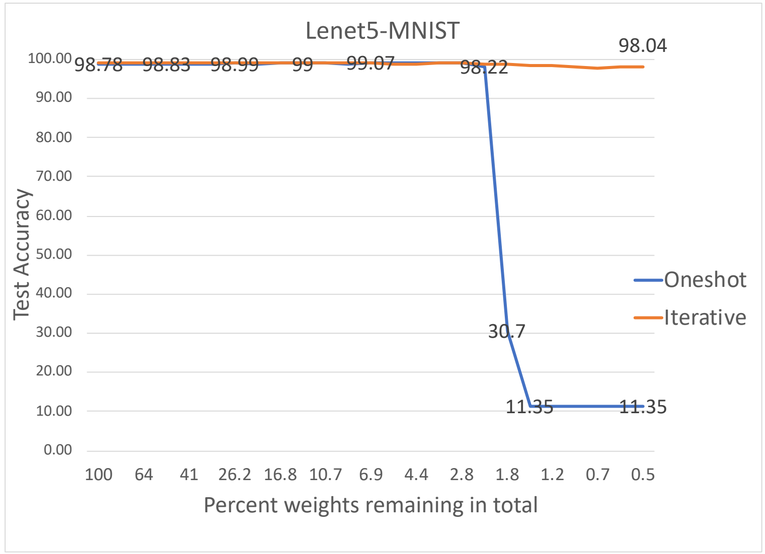
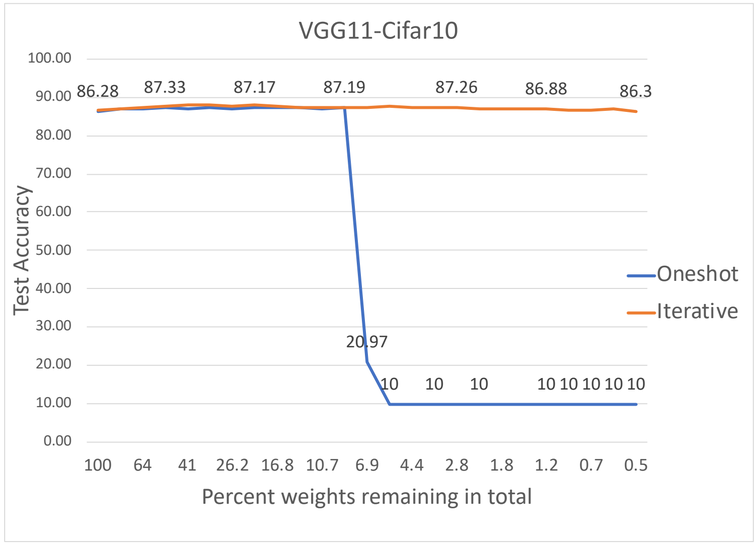
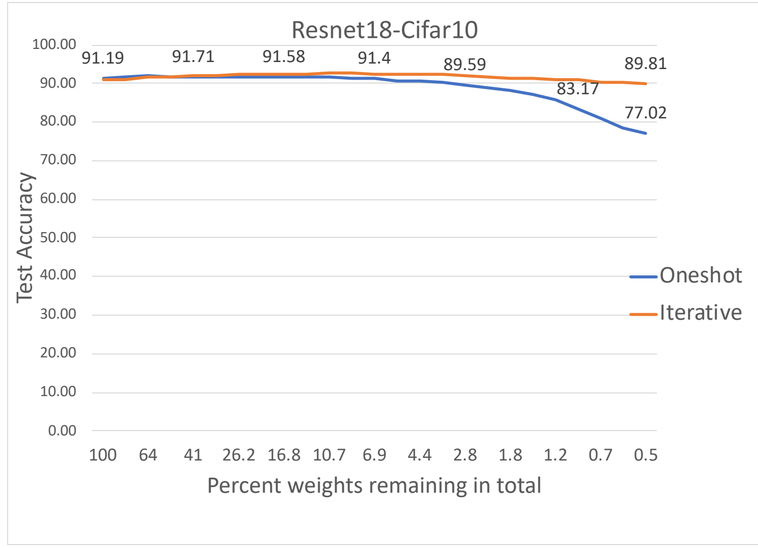


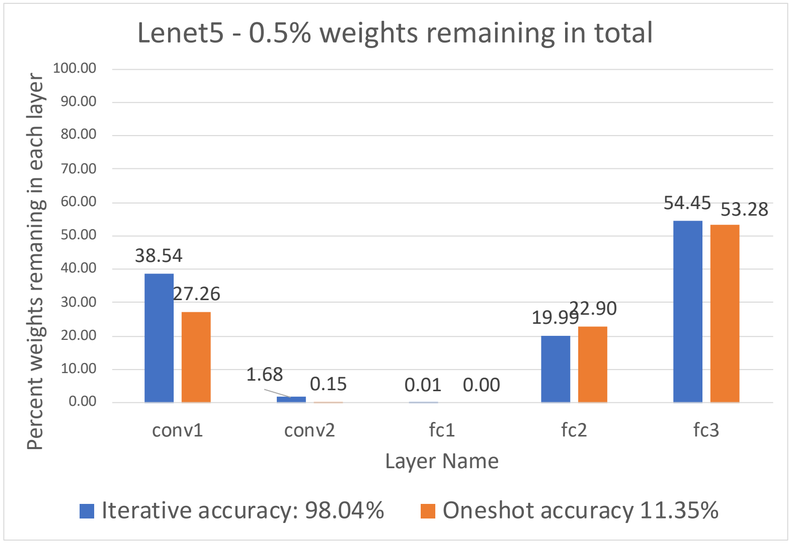
Figure 7: Test accuracy (at 30K, 60K, and 112K iterations) of VGG-19 when iteratively pruned.

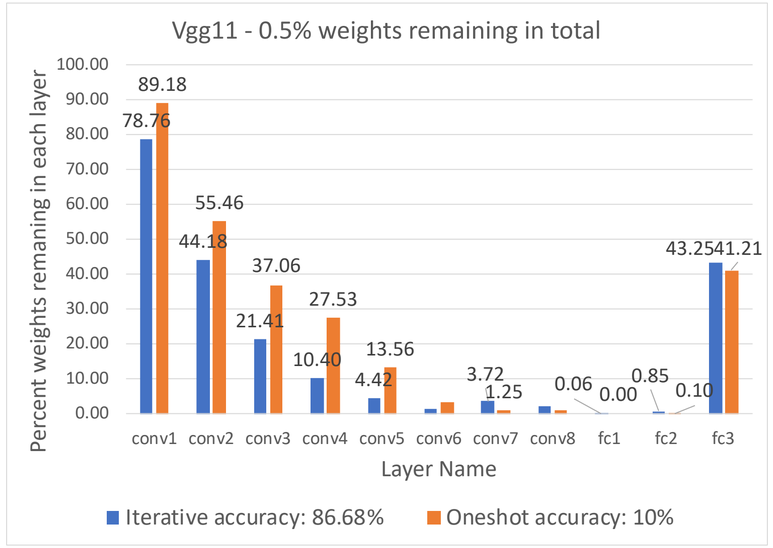
Experiments: find sparse network for Lenet, vgg11, resnet18:

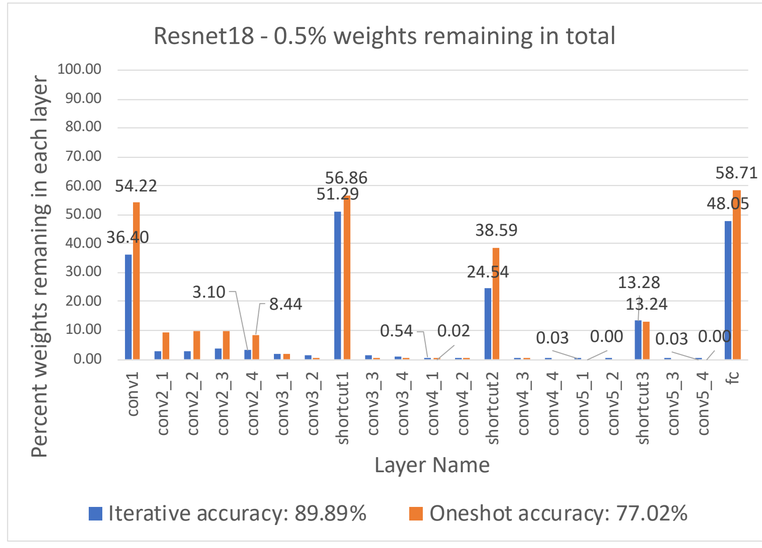












Recap

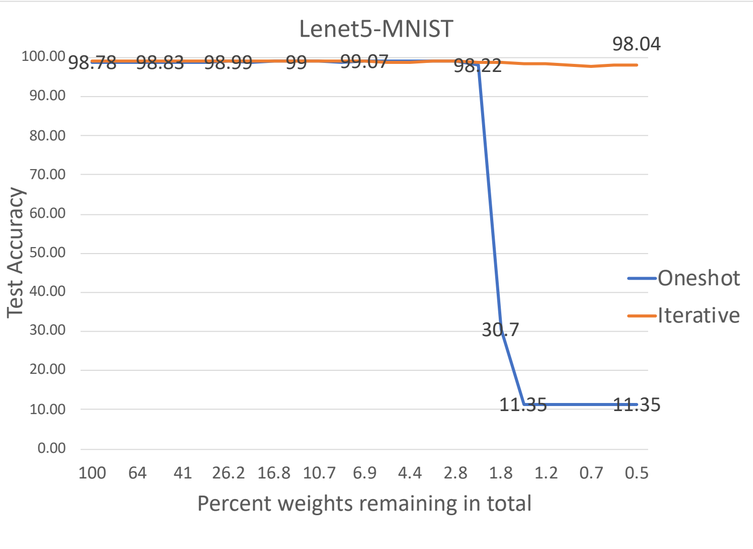
- Accuracy will not drop at all if there are more than 10 percent of weights remaining in total for both pruning methods.
- Oneshot pruning will lead to a big drop if all weights of a particular layer is pruned. Oneshot pruning can be improved. (Set an adaptive or fixed bound for each layer to keep a small amount of weights for each layer)
- The more weights a layer has, the faster the layer is pruned.

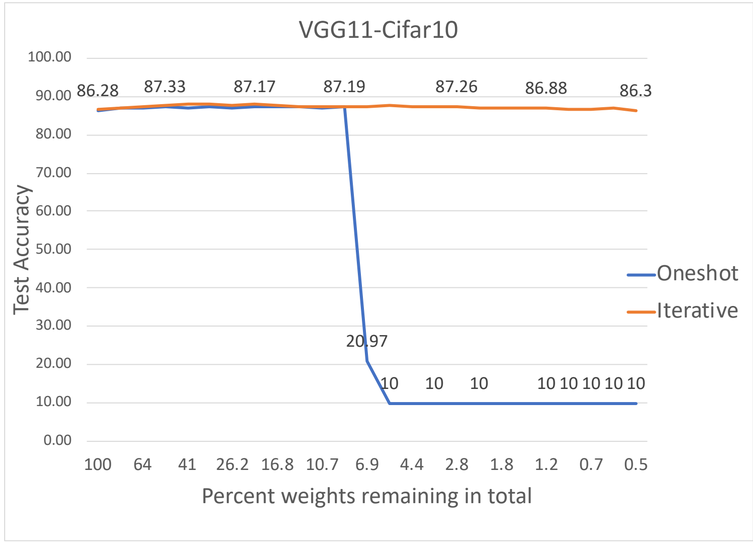
Will do next week:

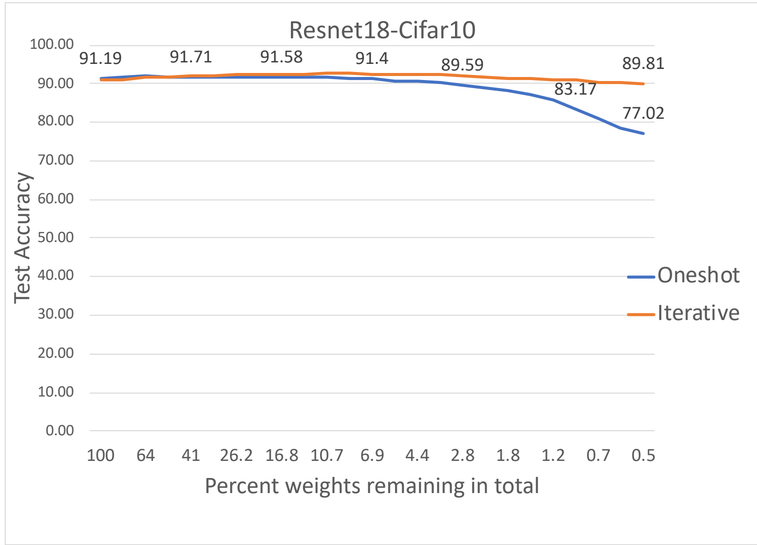
- Add a correction method to prevent oneshot from pruning all weights within a layer
- Apply SAF to the sparse networks with 10% percent weights remaining.
- Do not update weights SAF: keep error weights as initial value

Recap:

- Use oneshot and iterative pruning to get sparse network
- Accuracy will not drop at all if there are more than 10 percent of weights remaining in total for both pruning methods.
- Oneshot pruning will lead to a big drop if all weights of a particular layer is pruned. Oneshot pruning can be improved. (Set an adaptive or fixed bound for each layer to keep a small amount of weights for each layer)



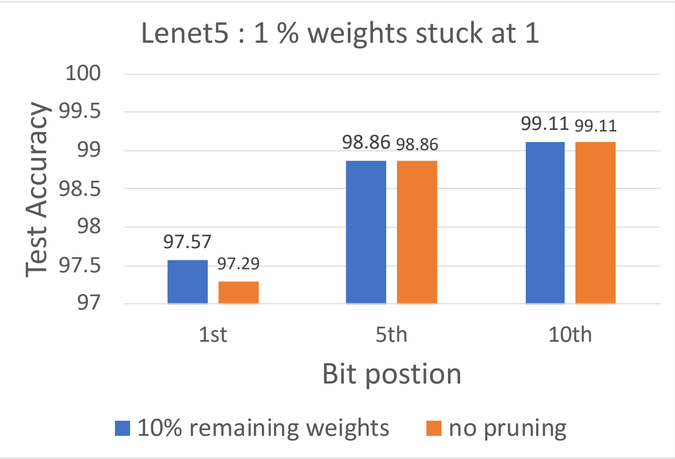


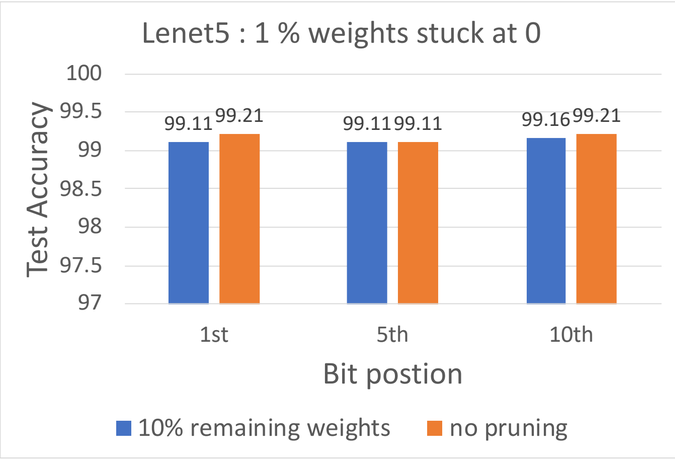


Apply SAF to the sparse networks with 10% weights remaining.

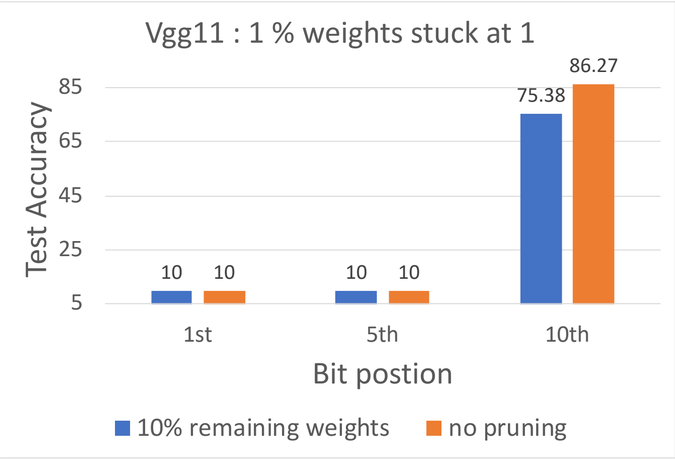
For dense network without pruning, inject 1 % faults to each layer
For sparse network with 10% remaining weights, inject 1% faults to remaining weights of each layer

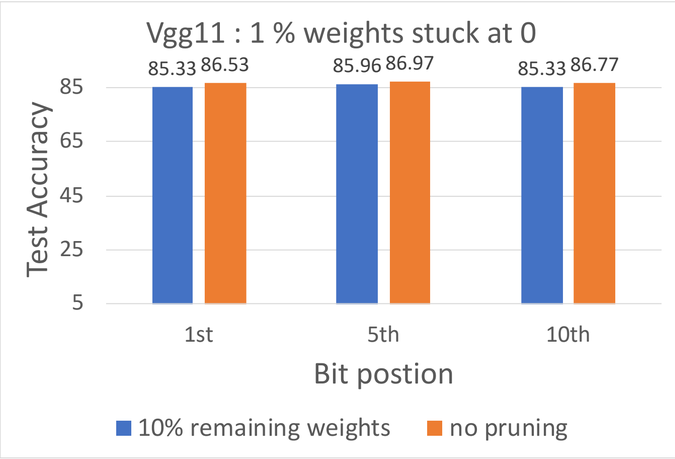
For Lenet sparse network with 10% remaining weights, resistance to faults is similar to network without pruning.





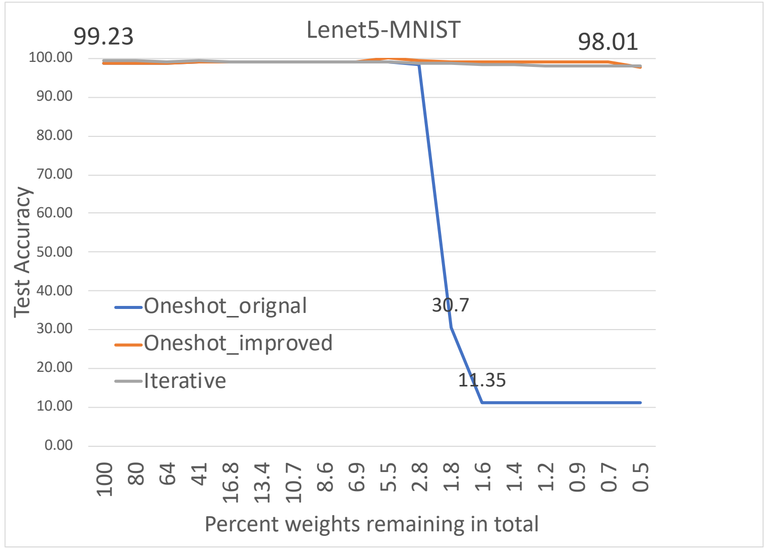
For vgg11 the sparse network, resistance to faults is similar to network without pruning when bit is stuck at 0. But it become more sensitive when bit is stuck at 1.

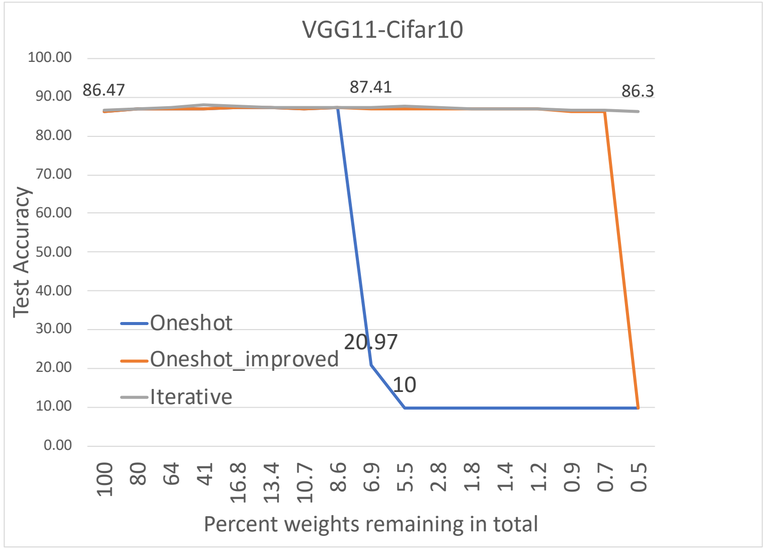




Add a correction method to prevent oneshot from pruning all weights within a layer

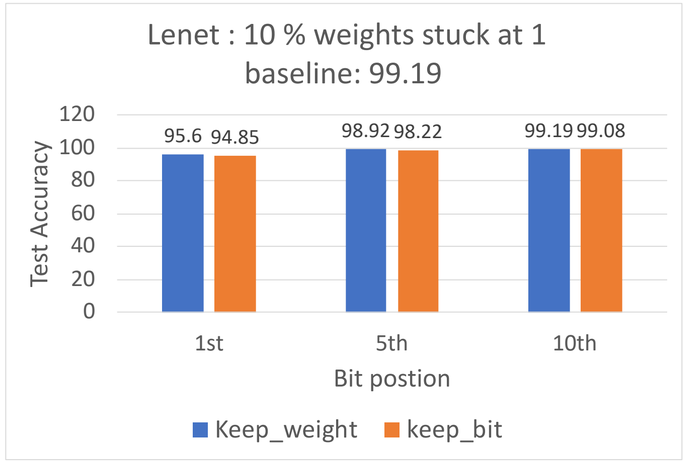
Oneshot_improved: Add a lower bound where there are at least 0.5 % weights remaining for each fully connected layer and 1.2% remaining weights for convolution layer.

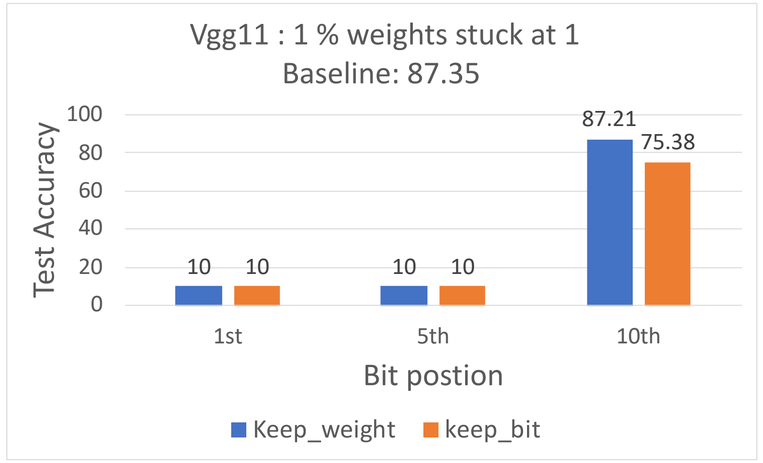


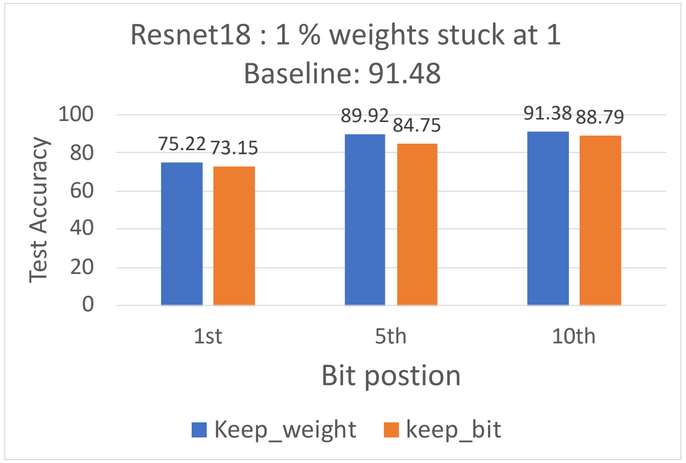


Do not update weights SAF: keep error weights as initial error value

All experiments are based on sparse network with 10% remaining weights
keep_weight means we do not update the weight that has stuck bit
keep_bit means we only do not update the stuck bit. In other words, it is the normal case where we inject SAF to sparse network

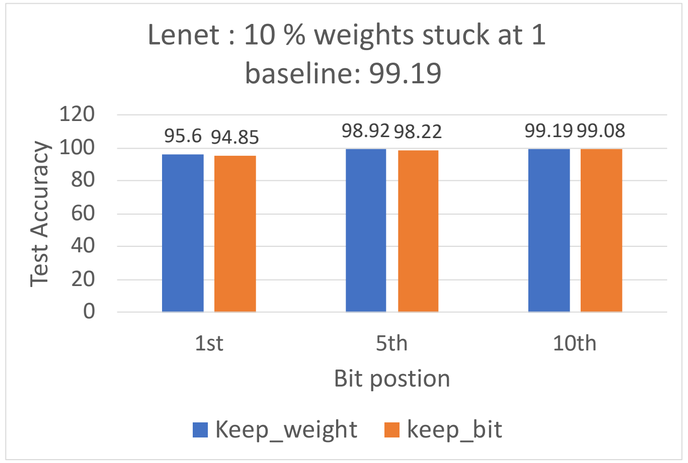


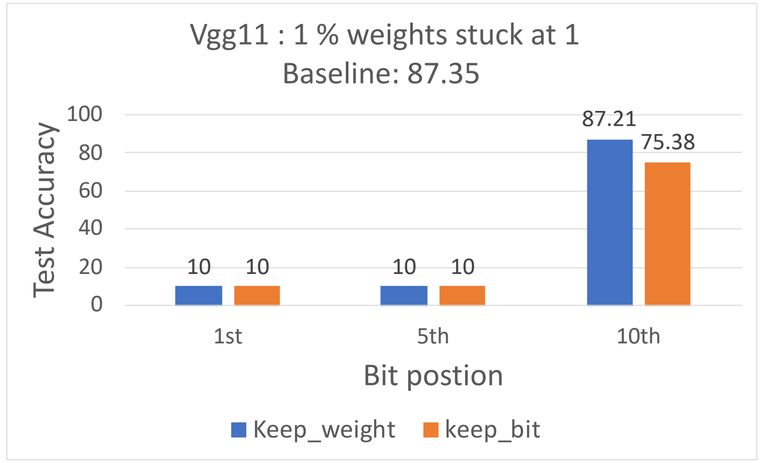


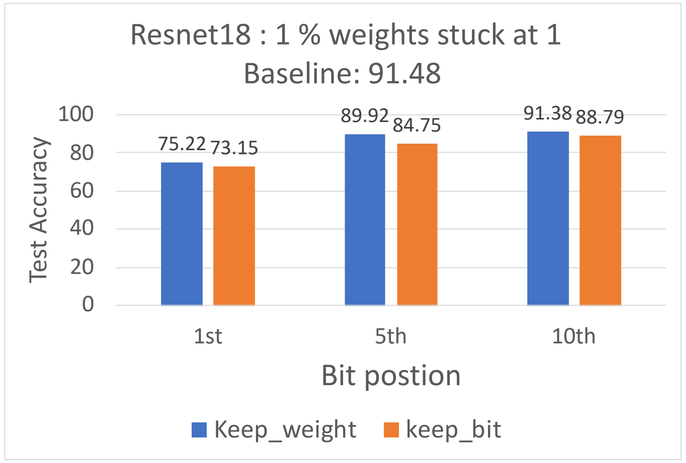


Recap: add faults to sparse network (10% weights remaining)

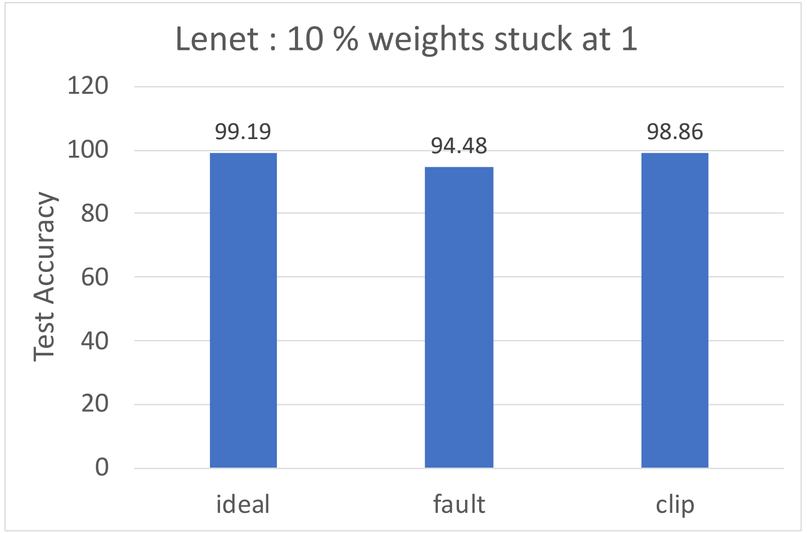
keep_weight means we do not update the weight that has stuck bit
keep_bit means we only do not update the stuck bit. In other words, it is the normal case where we inject SAF to sparse network

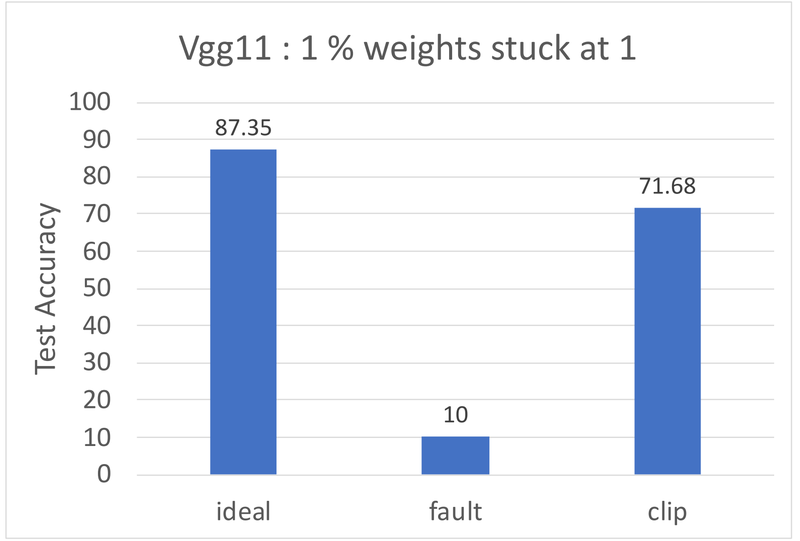






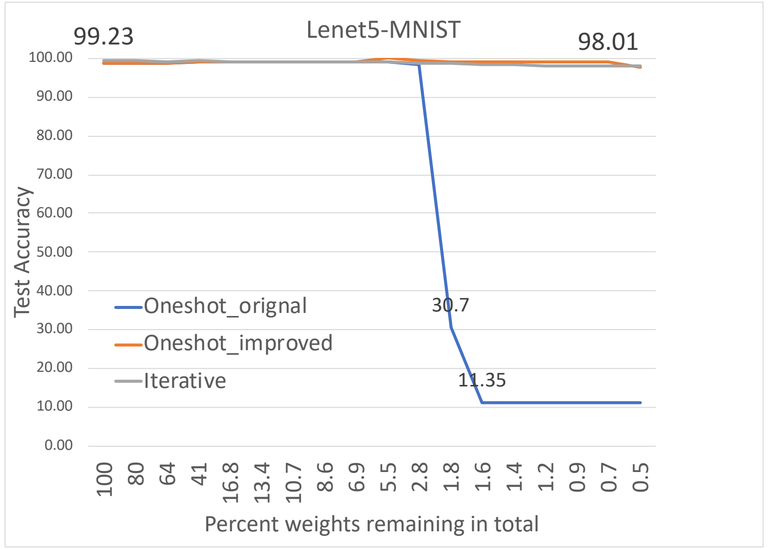
Clip faults to lower values within sparse network

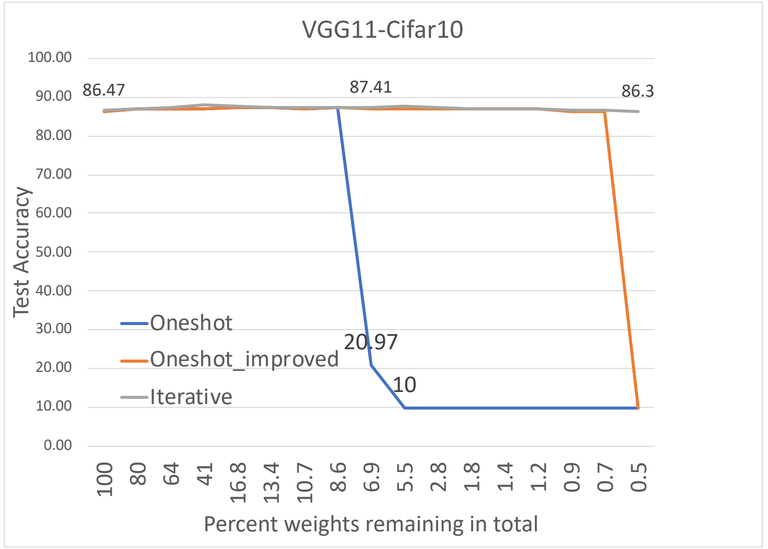




Recap: Add a correction method to prevent oneshot from pruning all weights within a layer

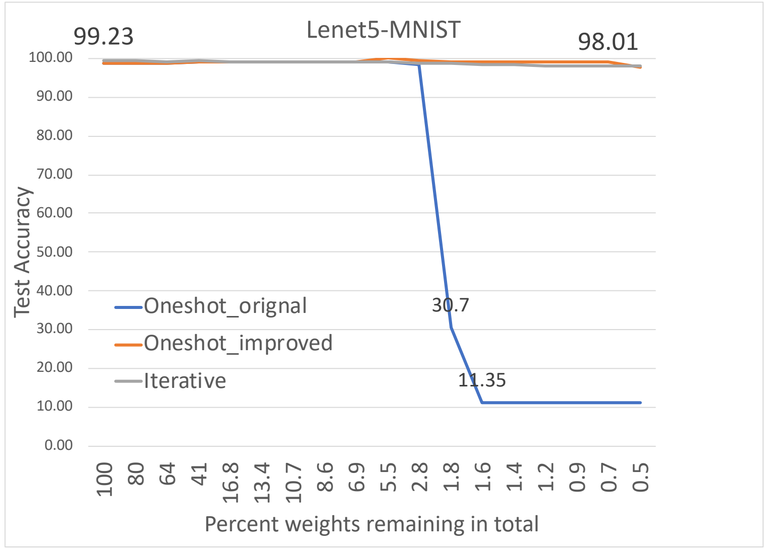
Oneshot_improved: Add a lower bound where there are at least 0.5 % weights remaining for each fully connected layer and 1.2% remaining weights for convolution layer.

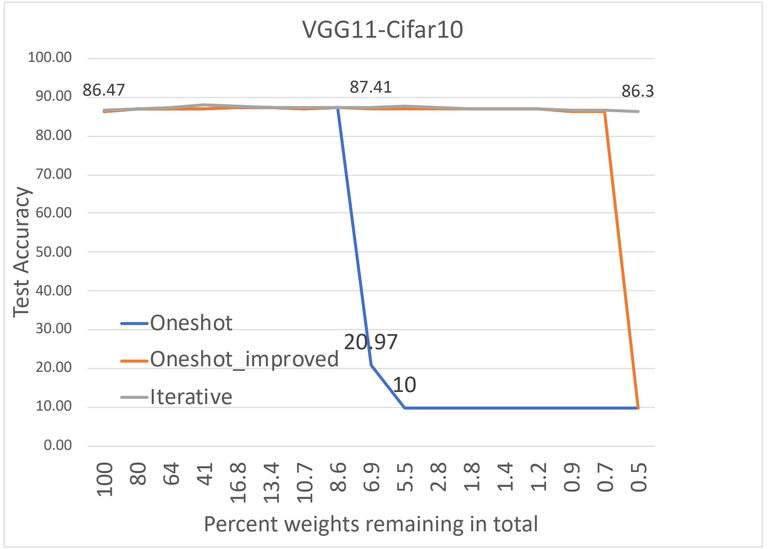




Recap: Add a correction method to prevent oneshot from pruning all weights within a layer

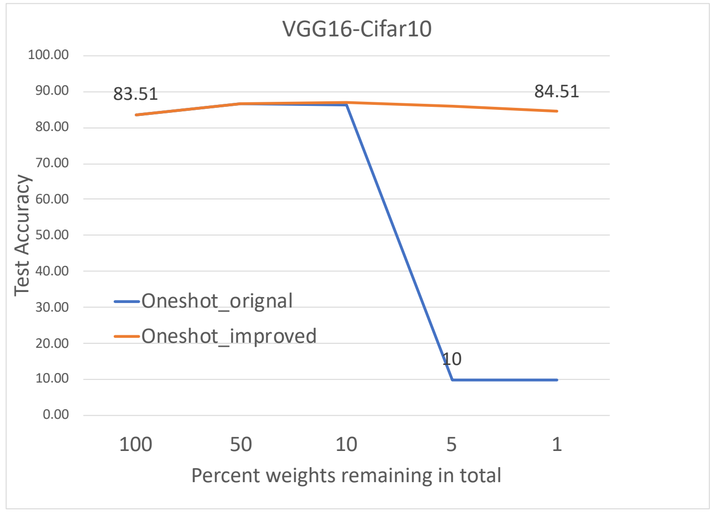
Oneshot_improved: Add a lower bound where there are at least 0.5 % weights remaining for each fully connected layer and 1.2% remaining weights for convolution layer.

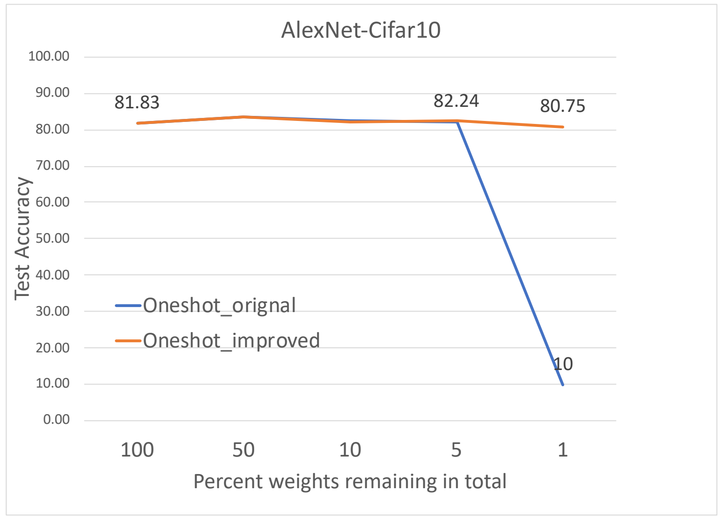


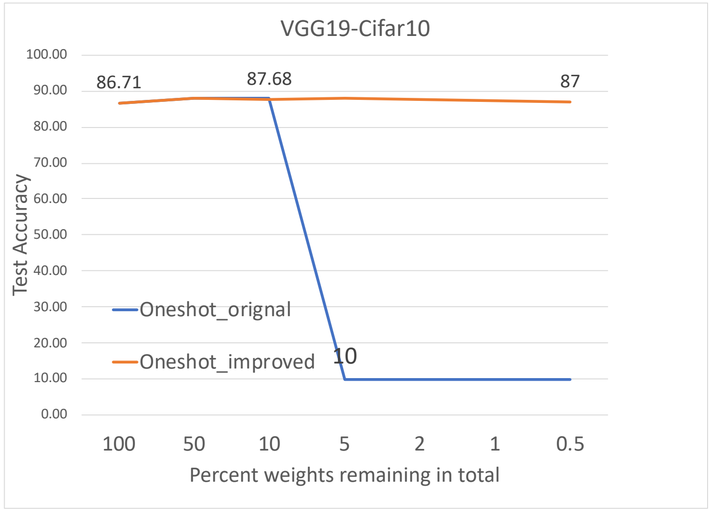


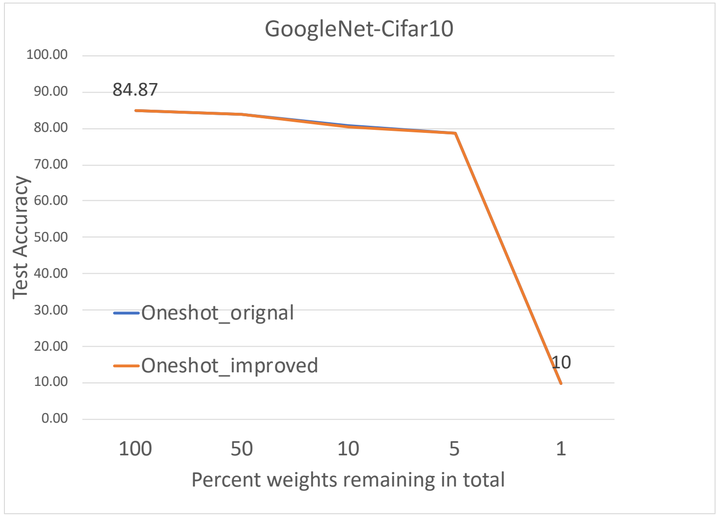
One-shot pruning VS. One-shot_improved pruning

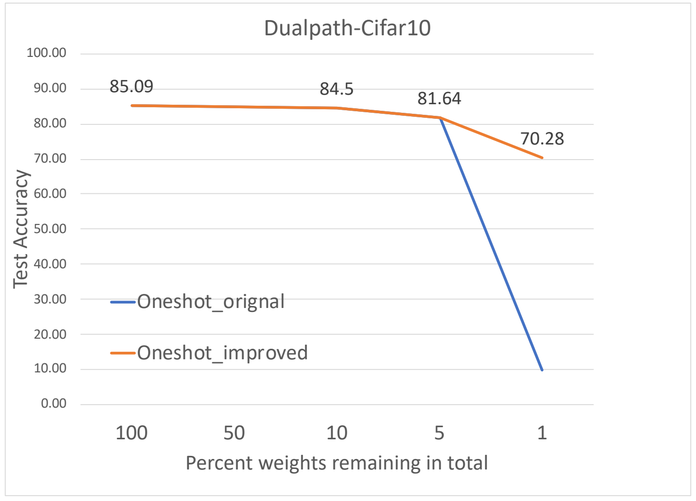
One-shot pruning: prune least significant weights globally once
One-shot_improved pruning: prune least significant weights globally once with a lower adaptive bound

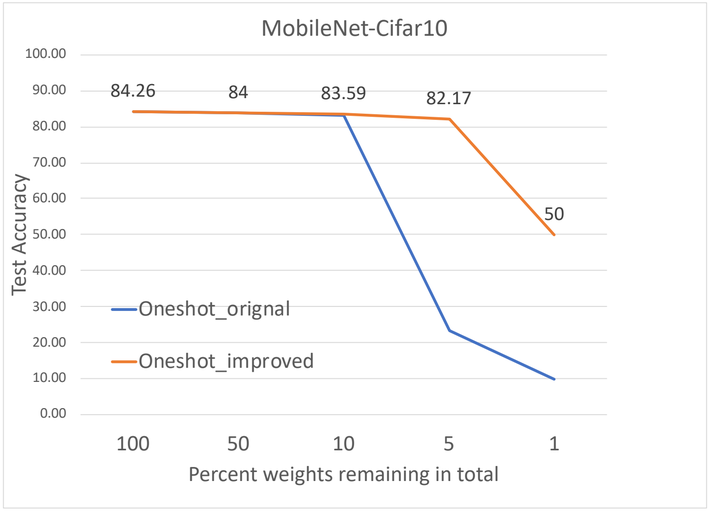








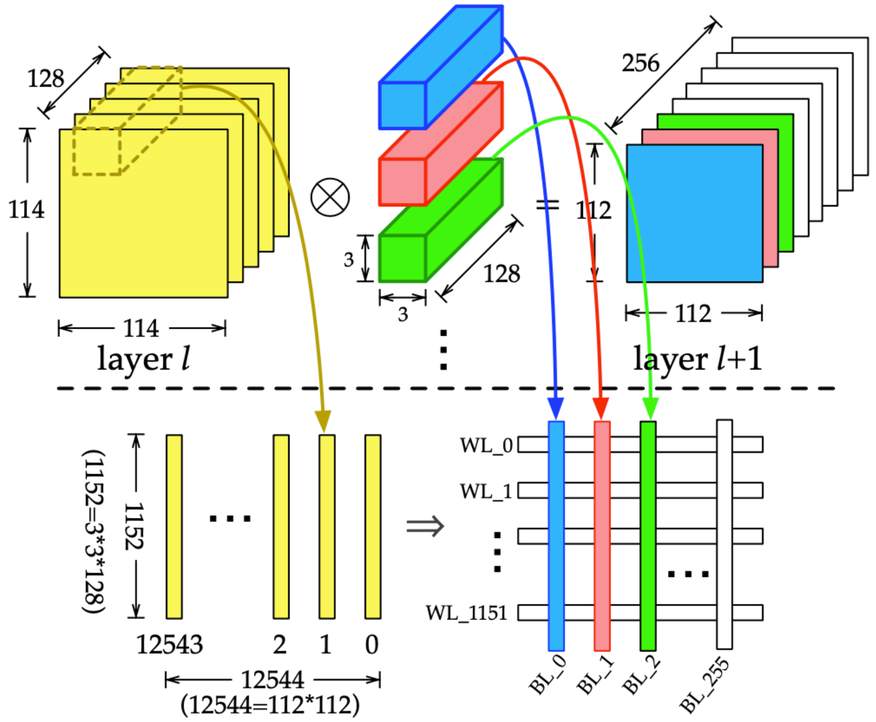




Map the sparse networks to ReRAM

Assume Crossbar size = 128*128

All networks here are sparse with 1.8 % weights remaining



Lenet5 Baseline = 99.13%			Sparsity = nonzero/total =1.8%			
Best acc = 96.5%		weights		Cross bar		
layer name	kernel size	total	nozero	Total	NZ_CB	NZ_iterms
conv1	(6,1,5,5)	150	48	1	1	[48]
conv2	(16,6,5,5)	2400	76	2	2	[66, 10]
fc1	(120, 400)	48000	66	4	4	[20, 23, 20, 3]
fc2	(84, 120)	10080	609	1	1	[609]
fc3	(10, 84)	840	307	1	1	[307]
Sum		61470	1106	9	9	
CrossBar						
layer name	total_cols	nz_cols	nz_cols/total	total_rows	nz_rows	nz_rows/total
conv1	6	6	1.00	25	22	0.88
conv2	32	23	0.72	400	60	0.15
fc1	480	49	0.10	400	63	0.16
fc2	84	84	1.00	120	118	0.98
fc3	10	10	1.00	84	84	1.00
Sum	612	172	0.28	1029	347	0.34

The figure above is from PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning

Map the sparse networks to ReRAM

Assume Crossbar size = 128*128
All networks here are sparse with 10 % weights remaining

Vgg11 Baseline= 86.36			Sparsity = nonzero/total = 3.5%		
Best acc = 86.44					
		weights		CrossBar	
layer name	kernel size	total	nozero	Total	NZ_CB
conv8					
fc1					
fc2					
fc3	(10, 256)	2560	2152	2	2
Sum		9613504	422805	590	590

CrossBar			
layer name	total_cols	nz_cols	nz_cols/total
conv1	64	64	1.00
conv2	640	640	1.00
conv3	2304	2304	1.00

```
Layer1 | (64, 3, 3, 3) | nz_cols/total_cols = 64/64=1.0 | nz_rows/total_rows = 27/27 = 1.0
Layer2 | (128, 64, 3, 3) | nz_cols/total_cols = 640/640=1.0 | nz_rows/total_rows = 576/576 = 1.0
Layer3 | (256, 128, 3, 3) | nz_cols/total_cols = 2304/2304=1.0 | nz_rows/total_rows = 2304/2304 = 1.0
Layer4 | (256, 256, 3, 3) | nz_cols/total_cols = 4608/4608=1.0 | nz_rows/total_rows = 4607/4608 = 0.9997
Layer5 | (512, 256, 3, 3) | nz_cols/total_cols = 9025/9216=0.9792 | nz_rows/total_rows = 9018/9216 = 0.9785
Layer6 | (512, 512, 3, 3) | nz_cols/total_cols = 9526/18432=0.5168 | nz_rows/total_rows = 12736/18432 = 0.6909
Layer7 | (512, 512, 3, 3) | nz_cols/total_cols = 9404/18432=0.5101 | nz_rows/total_rows = 6366/18432 = 0.3453
Layer8 | (512, 512, 3, 3) | nz_cols/total_cols = 9920/18432=0.5381 | nz_rows/total_rows = 8988/18432 = 0.4876
Layer9 | (512, 512) | nz_cols/total_cols = 2048/2048=1.0 | nz_rows/total_rows = 2048/2048 = 1.0
Layer10 | (256, 512) | nz_cols/total_cols = 1024/1024=1.0 | nz_rows/total_rows = 1024/1024 = 1.0
Layer11 | (10, 256) | nz_cols/total_cols = 20/20=1.0 | nz_rows/total_rows = 256/256 = 1.0
In all: nz_cols/total_cols = 48583/75220 = 0.6458 | nz_rows/total_rows = 47950/75355 = 0.6363
```

```
Layer8:  (512, 512, 3, 3) | Sparsity = 0.0108 | CrossBar: left/total: 144/144
Layer9:  (512, 512) | Sparsity = 0.4486 | CrossBar: left/total: 16/16
Layer10:  (256, 512) | Sparsity = 0.5121 | CrossBar: left/total: 8/8
Layer11:  (10, 256) | Sparsity = 0.8406 | CrossBar: left/total: 2/2
Total Sparsity = 422805/9613504 = 0.043980321847268175
```

VGG19-CIFAR10 on different crossbar size 128, 64, 8 and different global density = 4%, 2%,1%

LayerName	
Layer1	(64, 3, 3, 3)
Layer2	(64, 64, 3, 3)
Layer3	(128, 64, 3, 3)
Layer4	(128, 128, 3, 3)
Layer5	(256, 128, 3, 3)
Layer6	(256, 256, 3, 3)
Layer7	(256, 256, 3, 3)
Layer8	(256, 256, 3, 3)
Layer9	(512, 256, 3, 3)
Layer10	(512, 512, 3, 3)
Layer11	(512, 512, 3, 3)
Layer12	(512, 512, 3, 3)
Layer13	(512, 512, 3, 3)
Layer14	(512, 512, 3, 3)
Layer15	(512, 512, 3, 3)
Layer16	(512, 512, 3, 3)
Layer17	(512, 512)
Layer18	(256, 512)
Layer19	(10, 256)
saved/total	

Crossbar. size 128*128			
Total	4%	2%	1%
1	0	0	0
5	2	2	2
5	0	0	0
9	0	0	0
18	0	0	2
36	0	2	14
36	0	2	16
36	0	4	19
72	6	30	59
144	35	95	134
144	24	80	133
144	14	67	133
144	5	52	132
144	4	47	132
144	4	44	132
144	5	49	132
16	0	0	0
8	0	0	0
2	1	1	1
1249	100	475	1041
	8%	38%	83%

Crossbar. size 64*64			
Total	4%	2%	1%
1	0	0	0
9	0	0	0
18	0	0	0
36	0	0	4
72	0	6	28
144	8	38	89
144	10	42	97
144	10	46	105
288	83	186	262
576	285	470	555
576	229	428	555
576	176	391	555
576	114	345	553
576	99	329	552
576	97	319	553
576	106	334	552
64	0	0	0
32	0	0	0
4	3	3	3
4983	1220	2937	4463
	24%	58%	89%

Crossbar. size 8*8			
Total	4%	2%	1%
32	5	5	5
576	28	55	158
1152	239	400	704
2304	801	1205	1781
4608	2592	3411	4097
9216	6498	7813	8683
9216	6544	7868	8759
9216	6613	7991	8858
18432	15749	17443	18217
36864	33767	35941	36696
36864	32721	35480	36693
36864	31621	35078	36692
36864	30054	34561	36676
36864	29524	34352	36672
36864	29463	34237	36677
36864	29679	34401	36672
4096	23	72	431
2048	6	16	108
64	24	24	24
318912	25595	29035	30860
	80%	91.1%	96.7%

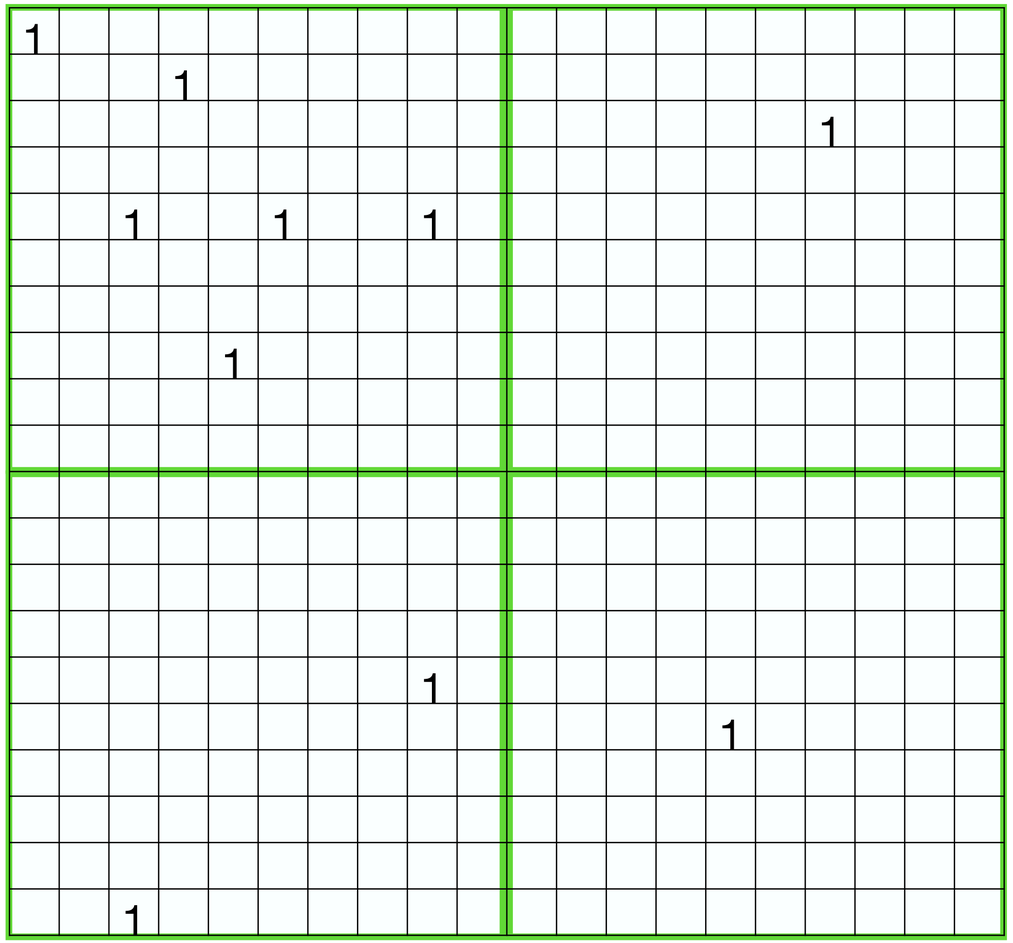
VGG19-CIFAR10 on different crossbar size 128, 64, 8 and different global density = 4%, 2%,1%

1. Assume weights are evenly distributed
2. The probability of a weight is nonzero is the density of a particular layer
3. Independent to each other

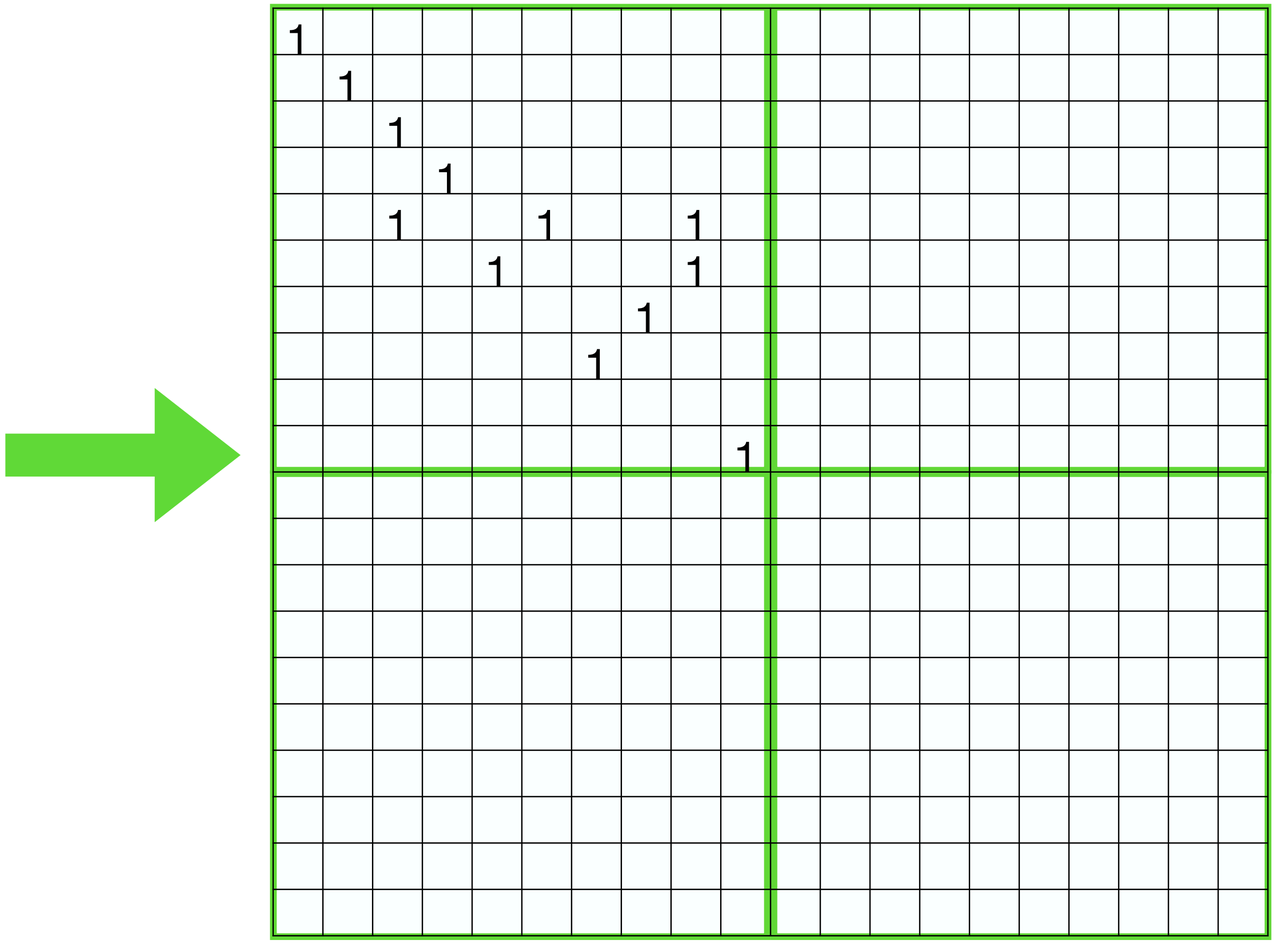
VGG19 -CIFAR10		Global density = 4%	Global density = 2%	Global density = 1%
column size		Layer13 density = 0.025	Layer13 density = 0.008	Layer13 density = 0.0006
8	Theoretical	Pr (zero col)= (1- 0.025)^8 = **81.6%**	Pr (zero col)=(1- 0.008)^8 = **93.7%**	Pr (zero col)=(1- 0.0006)^8 = **99.5%**
	Experimental	**81.5%**	**93.7%**	**99.4%**
64	Theoretical	Pr (zero col)=(1- 0.025)^64 = **19.8%**	Pr (zero col)=(1- 0.008)^64 = **59.8%**	Pr (zero col)=(1- 0.0006)^64 = **96.2%**
	Experimental	**19.7%**	**59.9%**	**96.0%**
128	Theoretical	Pr (zero col)=(1- 0.025)^128 = **4%**	Pr (zero col)=(1- 0.008)^128= **35.7%**	Pr (zero col)=(1- 0.0006)^128 = **92.6%**
	Experimental	**4%**	**35.9%**	**91.9%**

It also work for other CNNs like Resnet18, VGG11, etc.

Reorder the sparse matrix to save more crossbars



Sparse network mapped to crossbar
The matrix is so sparse and we cannot save any crossbar



After reordering, we can save more crossbars

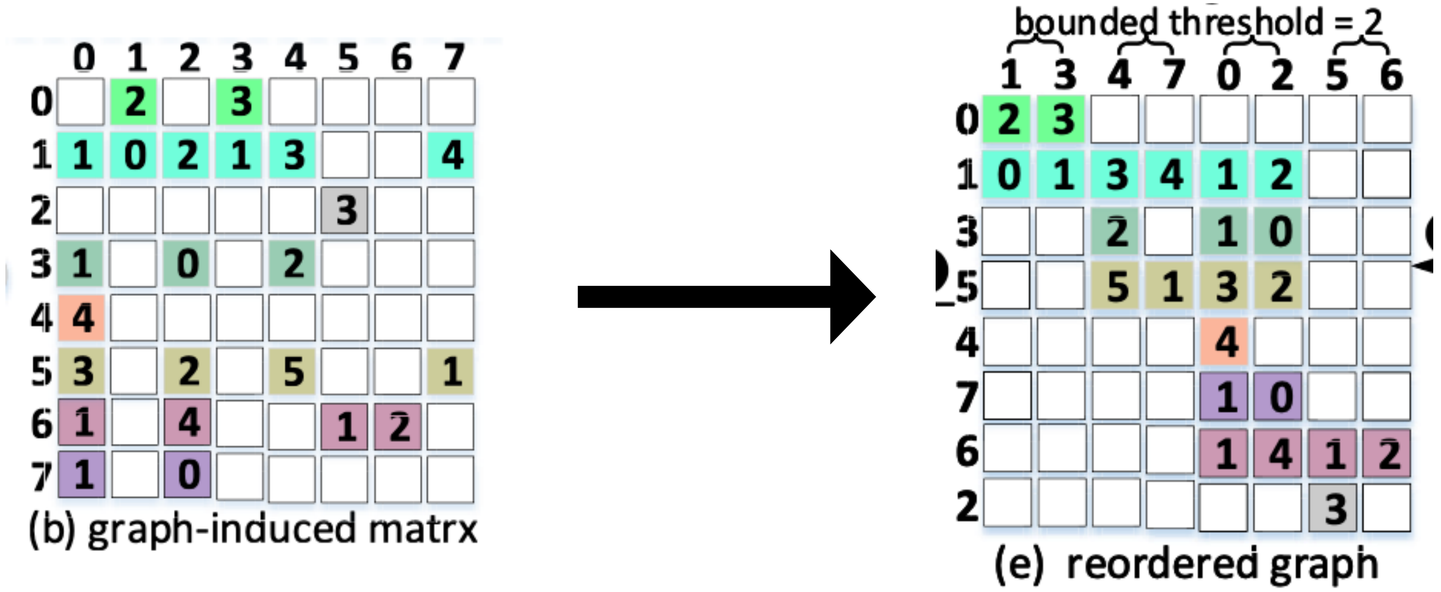
RCM algorithm does not work for our case

Issue:

```
before reordering:
(512, 512) Weights:Left/total: 151725/262144 | CrossBar: Left/total: 16/16
After reordering:
(512, 512) Weights:Left/total: 108111/262144 | CrossBar: Left/total: 10/16
```

This is because RCM algorithm only works for symmetric matrix and it creates an undirected graph for it.
But our sparse matrix is not symmetric.

Solution Implementing Reordering Algorithm in SPARA



Spara: An Energy-Efficient ReRAM-Based Accelerator for Sparse Graph Analytics Applications

Input: *CSR*, *CSC* – The CSR and CSC formats of graph
THRESHOLD – Crossbar size
Output: Mat – The aggregated graph-induced matrix
`/* Initialization */`
Mat$\leftarrow \varnothing$; *DstVSet*$\leftarrow \varnothing$; *SrcVSet* $\leftarrow \{v_0\}$
foreach vertex $v_i \in V$ where $0 \leq i < |V|$ and i is incremented only when v_i finishes visiting all of its out-going edges **do**
  while *true* **do**
    `GetDstVertex()`
    `GetSrcVertex()`
    if |*DstVSet*| = THRESHOLD **then**
      `Append`(*Mat.RowIndexOrder*, *SrcVSet*)
      `Append`(*Mat.ColIndexOrder*, *DstVSet*)
      SrcVSet$\leftarrow \varnothing$; *DstVSet*$\leftarrow \varnothing$
      break

Procedure `GetDstVertex()`
  foreach vertex $v \in$ *SrcVSet* **do**
    for $i \in$ [*CSR.ptr(v), CSR.ptr(v+1)*] **do**
      DstV $\leftarrow$ *col(i)*
      if *DstV* is not visited **then**
        DstVSet $\leftarrow$ *DstVSet* $\cup \{DstV\}$
      if |*DstVSet*| = THRESHOLD **then**
        return

Procedure `GetSrcVertex()`
  foreach vertex $v \in$ *DstVSet* **do**
    for $i \in$ [*CSC.ptr(v), CSC.ptr(v+1)*] **do**
      SrcV $\leftarrow$ *row(i)*
      if *SrcV* is not visited **then**
        SrcVSet $\leftarrow$ *SrcVSet* $\cup \{SrcV\}$

Vertex0:
DstVSet: {1,3}
SrcVertex: {0,1}
Col = {1,3}
Row = {0,1}

Vertex1
DstVSet: {4,7}
SrcVertex: {3,5}
Col = {1,3,4,7}
Row = {0,1,3,5}

Vertex1
DstVSet: {0,2}
SrcVertex: {4,6,7}

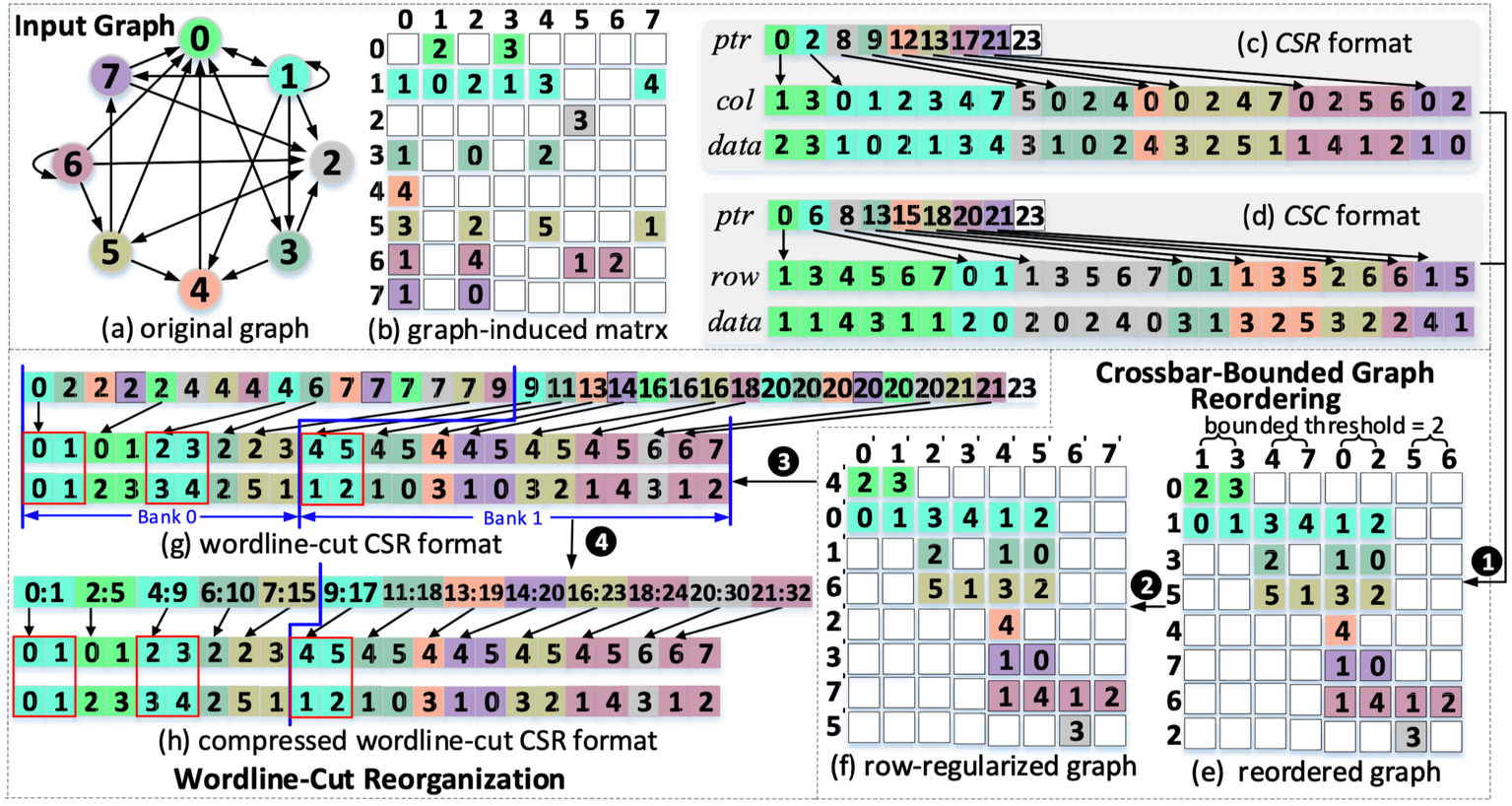
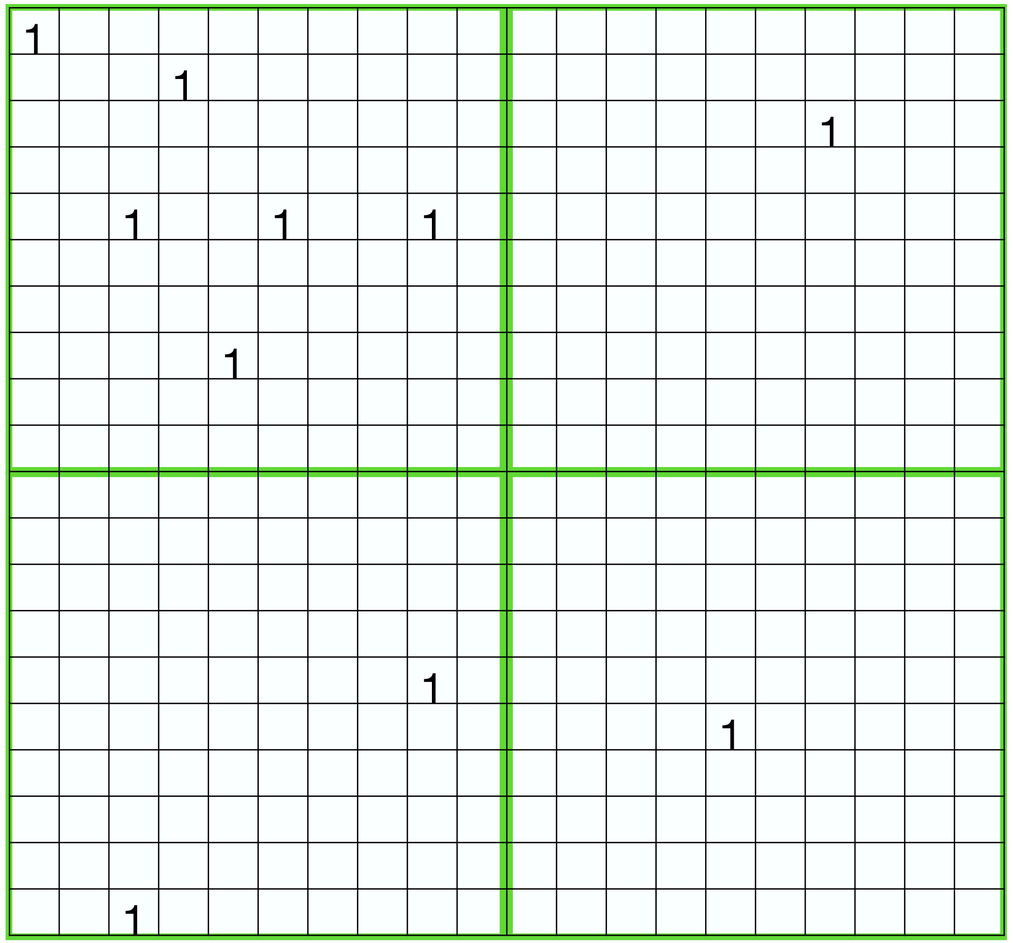
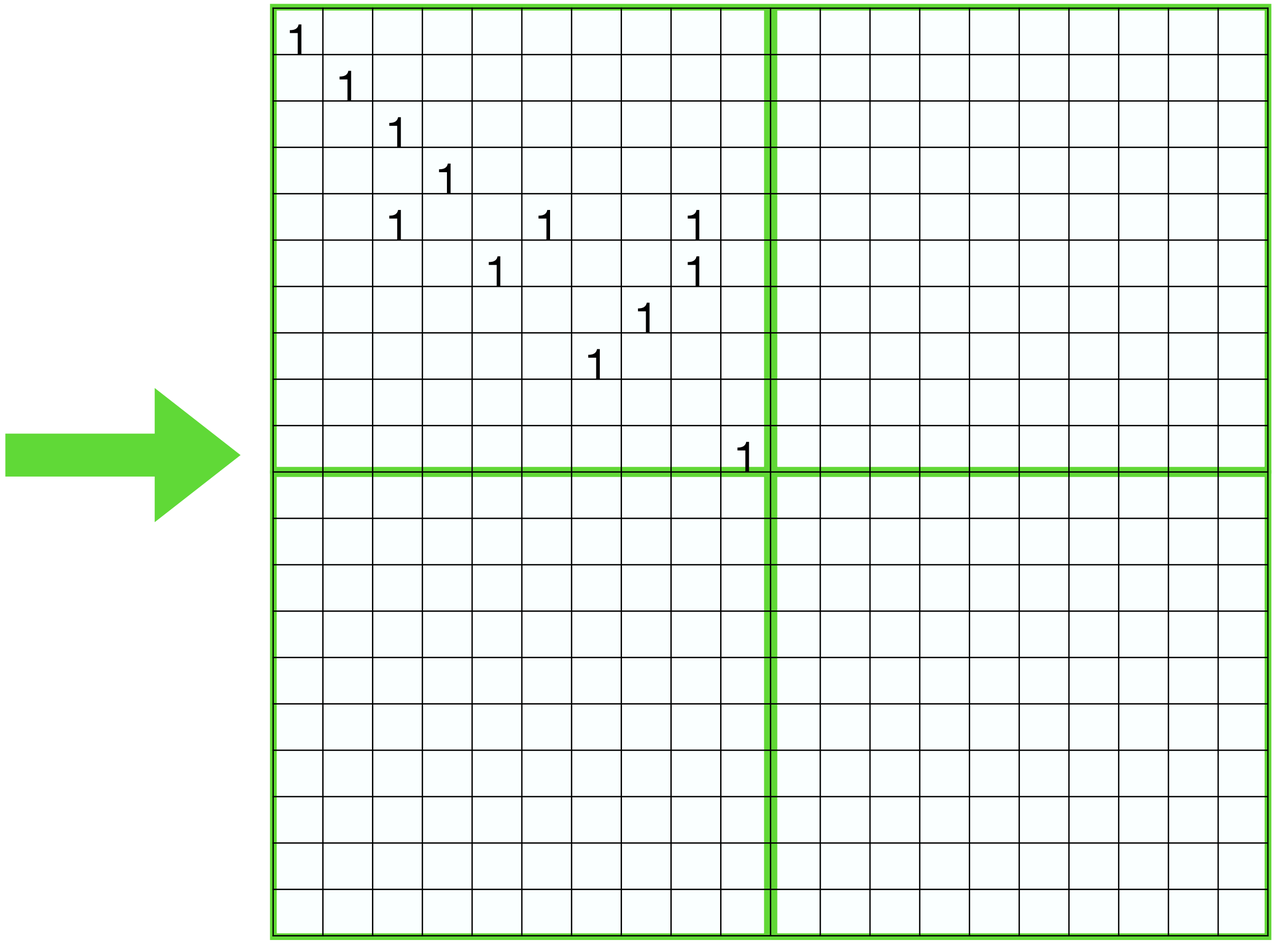


Fig. 3. Crossbar-bounded graph reordering

Reorder the sparse matrix to save more crossbars



Sparse network mapped to crossbar
The matrix is so sparse and we cannot save any crossbar



After reordering, we can save more crossbars

Structured sparsity learning for structures of filters, channels, filter shapes and depth

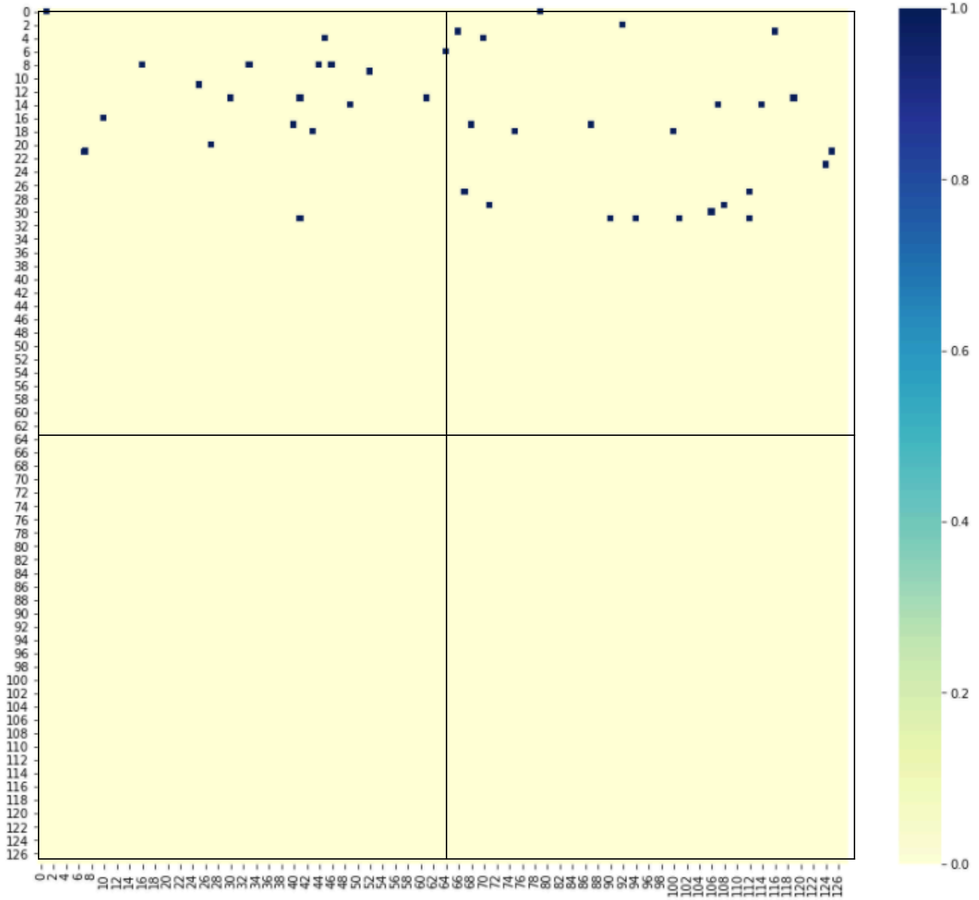
Penalizing unimportant filers and channels. Suppose $\boldsymbol{W}^{(l)}_{n_l,:,:,:}$ is the n_l-*th* filter and $\boldsymbol{W}^{(l)}_{:,c_l,:,:}$ is the c_l-*th* channel of all filters in the l-*th* layer. The optimization target of learning the filter-wise and channel-wise structured sparsity can be defined as

$$E(\boldsymbol{W}) = E_D(\boldsymbol{W}) + \lambda_n \cdot \sum_{l=1}^{L} \left(\sum_{n_l=1}^{N_l} ||\boldsymbol{W}^{(l)}_{n_l,:,:,:}||_g \right) + \lambda_c \cdot \sum_{l=1}^{L} \left(\sum_{c_l=1}^{C_l} ||\boldsymbol{W}^{(l)}_{:,c_l,:,:}||_g \right). \tag{2}$$

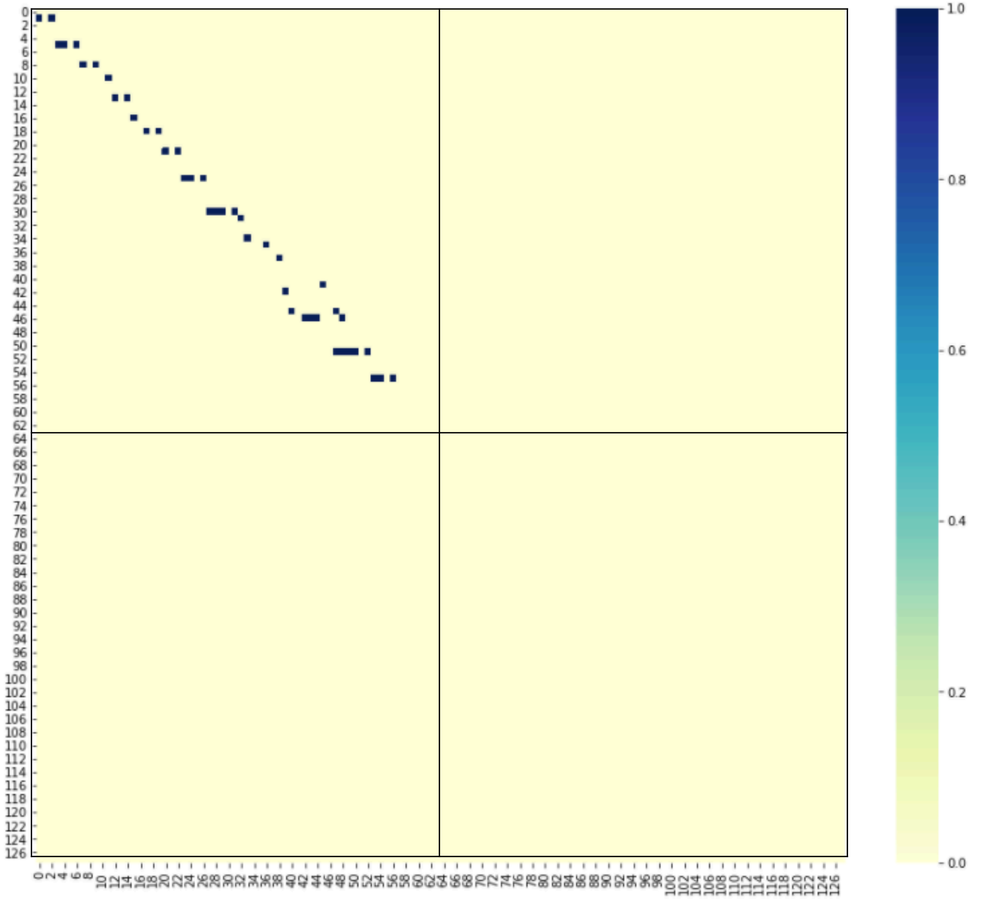
As indicated in Eq. (2), our approach tends to remove less important filters and channels. Note that zeroing out a filter in the l-*th* layer results in a dummy zero output feature map, which in turn makes a corresponding channel in the $(l+1)$-*th* layer useless. Hence, we combine the filter-wise and channel-wise structured sparsity in the learning simultaneously.

Padding problem

Assume we have crossbar size is 64*64
Assume we have a layer: (32, 8, 4, 4) with sparsity 1%. map it to 32 bars with length 128: (32,128).
In order to run this algorithm, let pad this matrix with zeros to (128,128)



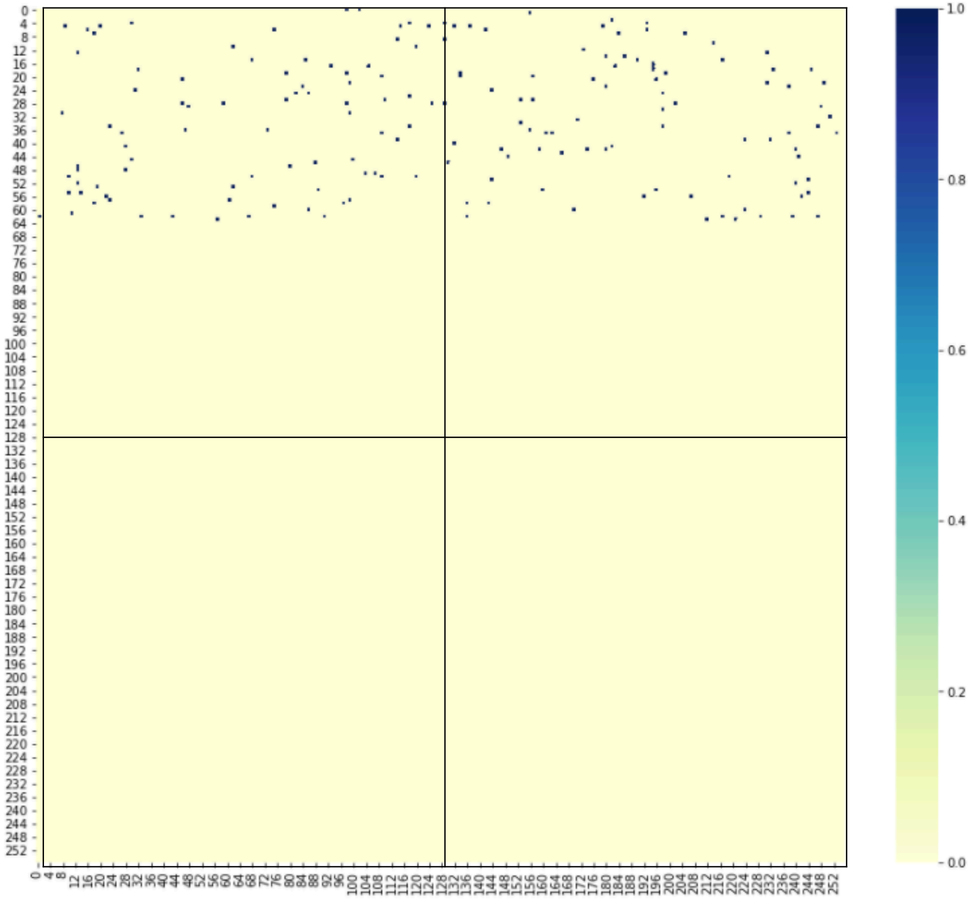
Sparse network mapped to crossbar
The matrix is so sparse and we cannot save any crossbar



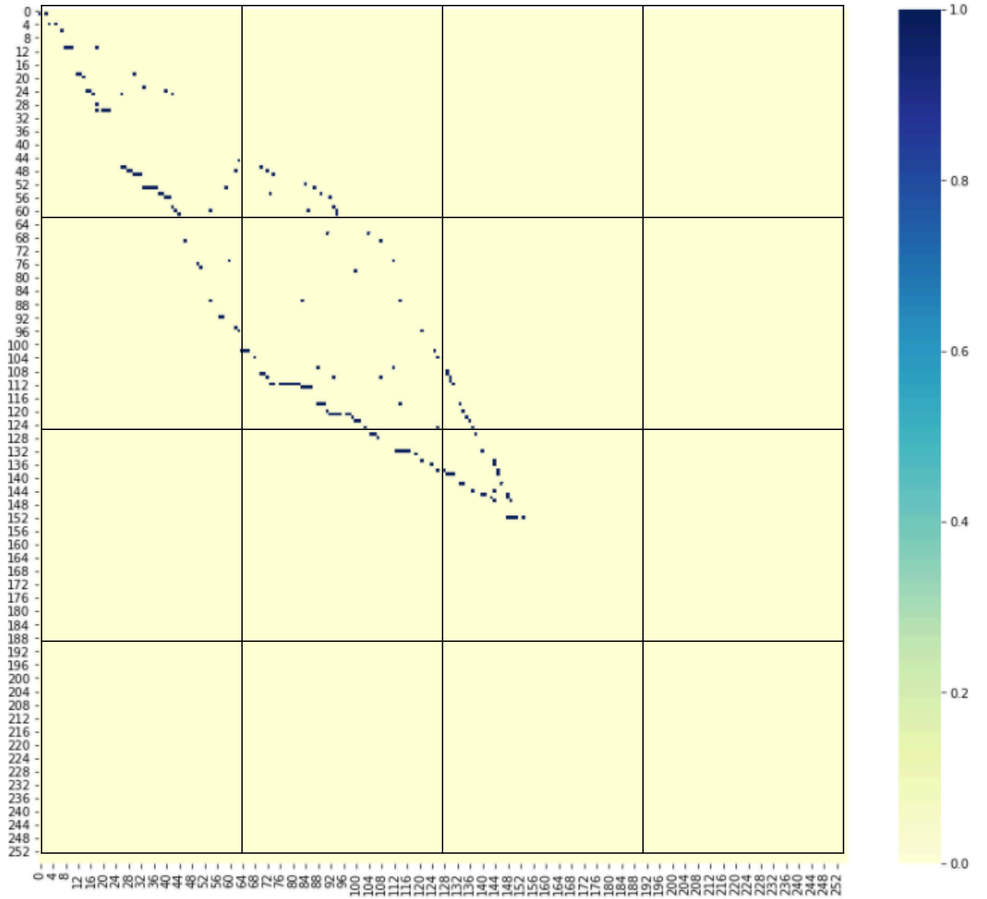
After reordering, we can save ONE crossbars

Padding problem

Assume we have crossbar size is 128*128
Assume we have a layer: (64, 16, 4, 4) with sparsity 1%, map it to 64 bars with length 256: (64,256)
In order to run this algorithm, let pad this matrix with zeros to (256,256)



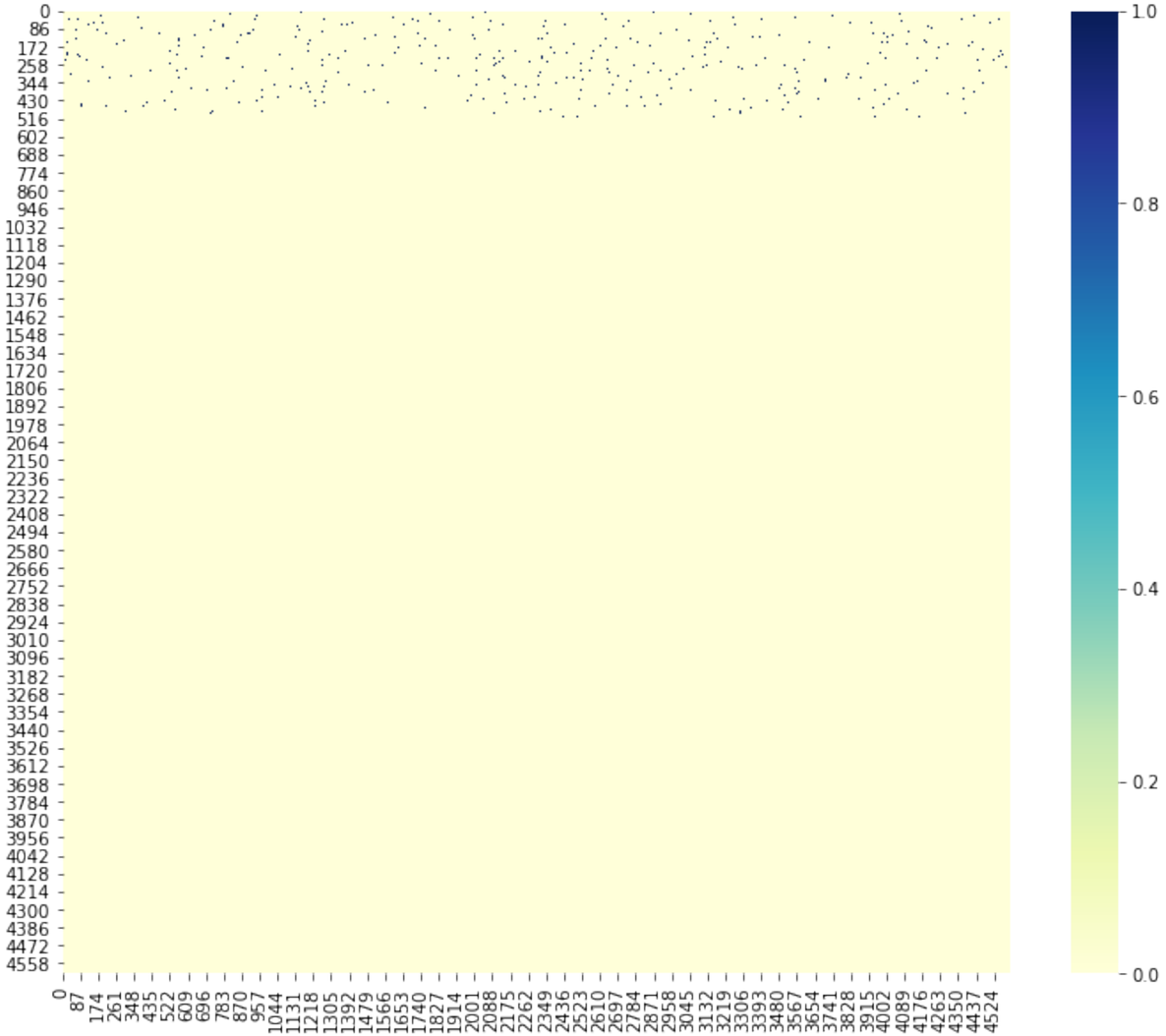
Sparse network mapped to crossbar
The matrix is so sparse and we cannot save any crossbar



After reordering, we use TWO more crossbars

VGG11- Tried to remove the crossbars after mapping

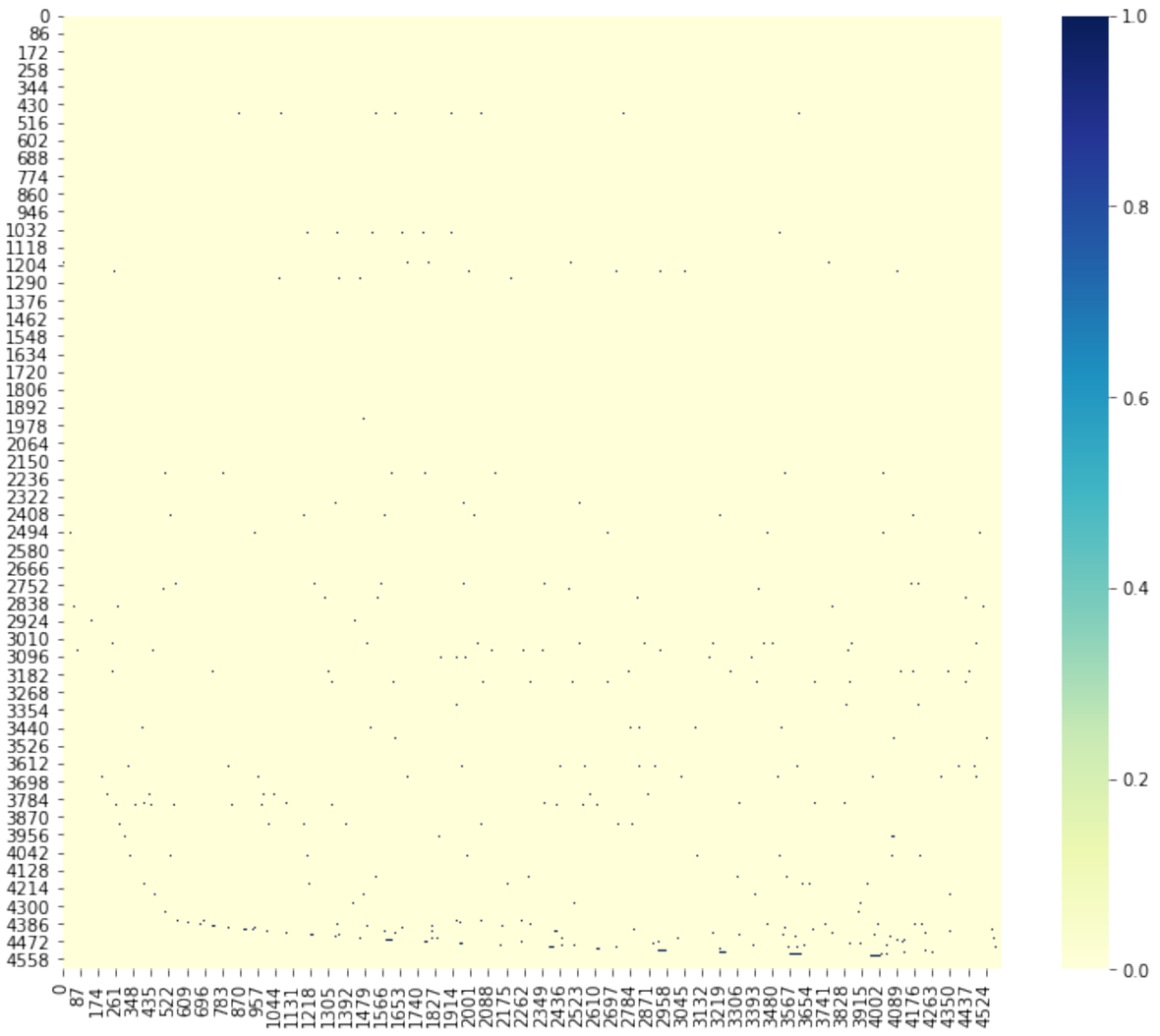
Layer8: (512,512,3,3) Sparsity = 25496 / 2359296 = 1%
Map it to crossbar (512, 4608).



We are using 144 crossbars here

This version of RCM algorithm will not work on very large matrix

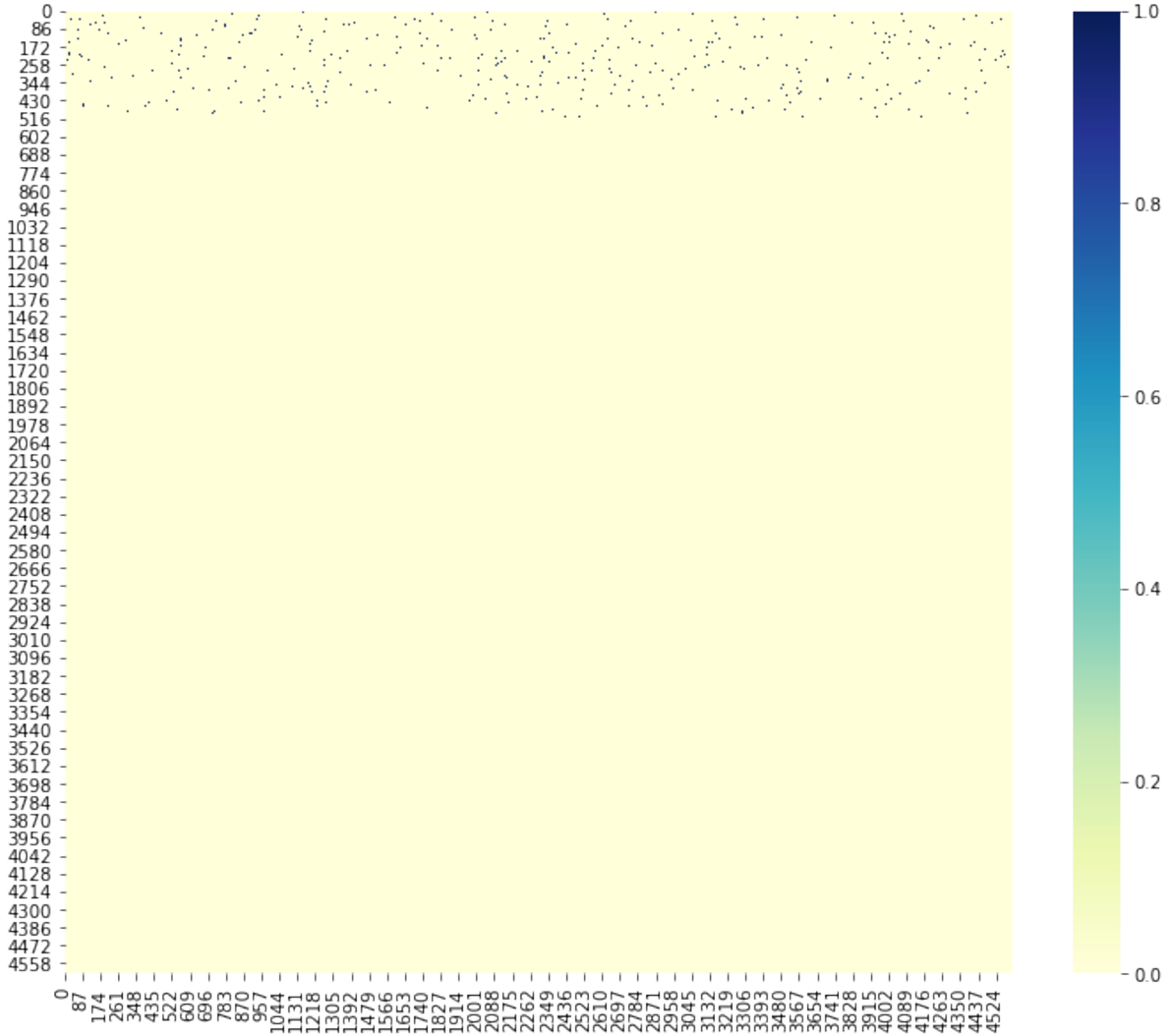
Spicy Package RCM with symmetric_mode = False



After reordering, we are using 1169 crossbars here

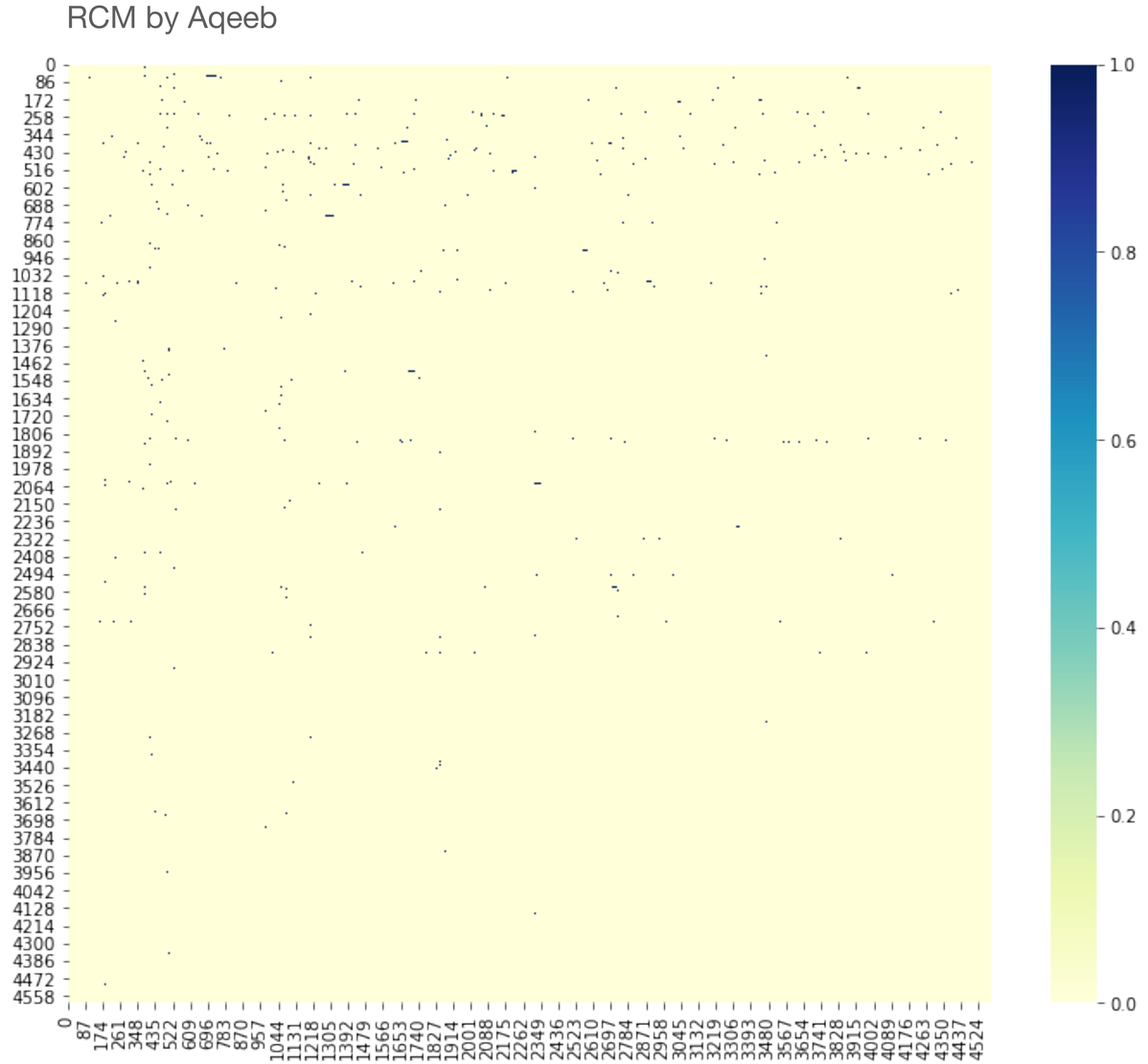
VGG11- Tried to remove the crossbars after mapping

Layer8: (512,512,3,3) Sparsity = 25496 / 2359296 = 1%
Map it to crossbar (512, 4608).



We are using 144 crossbars here

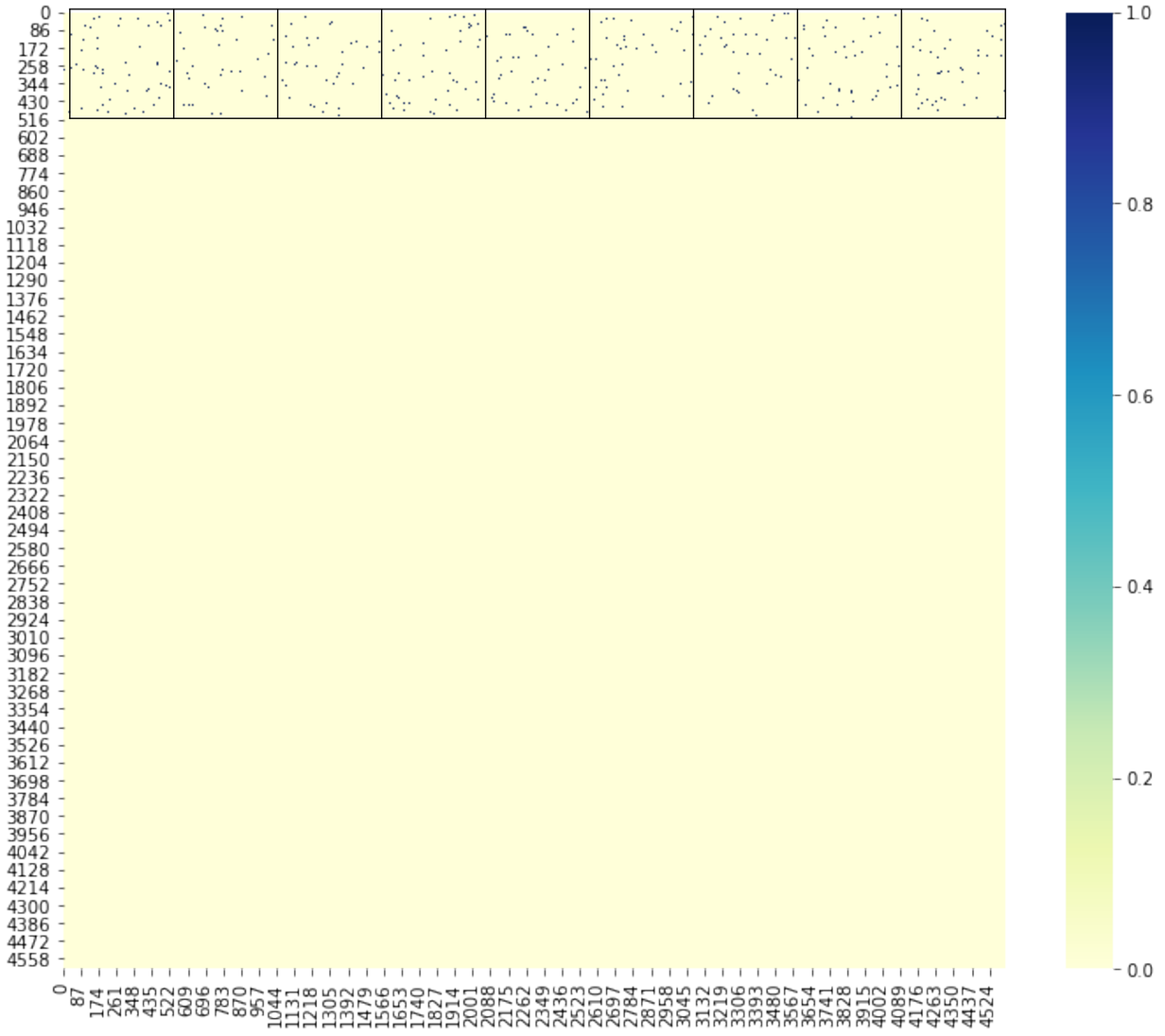
This version of RCM algorithm will not work on very large matrix



After reordering, we are using 1086 crossbars here

VGG11- Tried to remove the crossbars after mapping

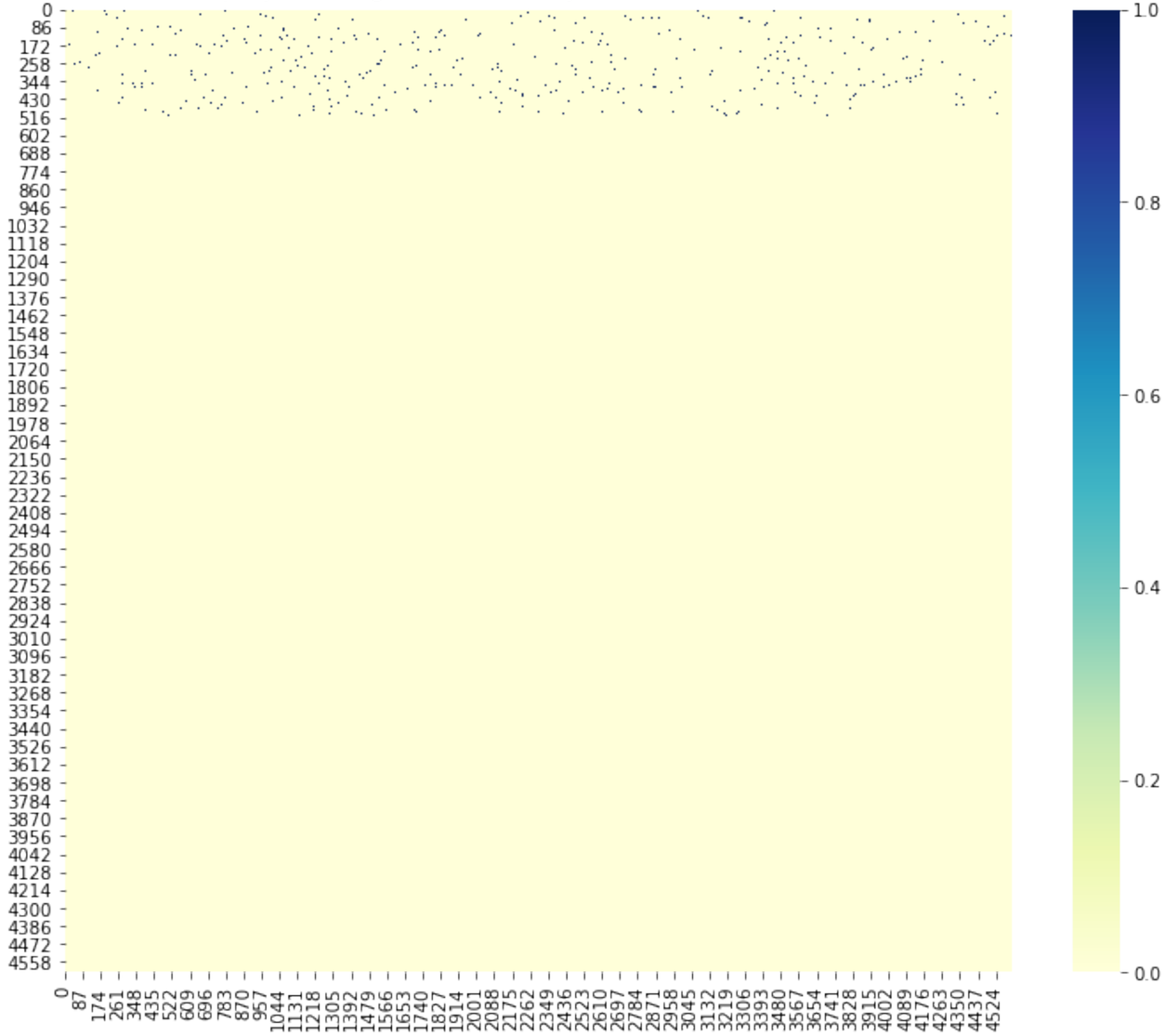
Layer8: (512,512,3,3) Sparsity = 25496 / 2359296 = 1%



We are using 144 crossbar

Do reordering within the rectangular matrix

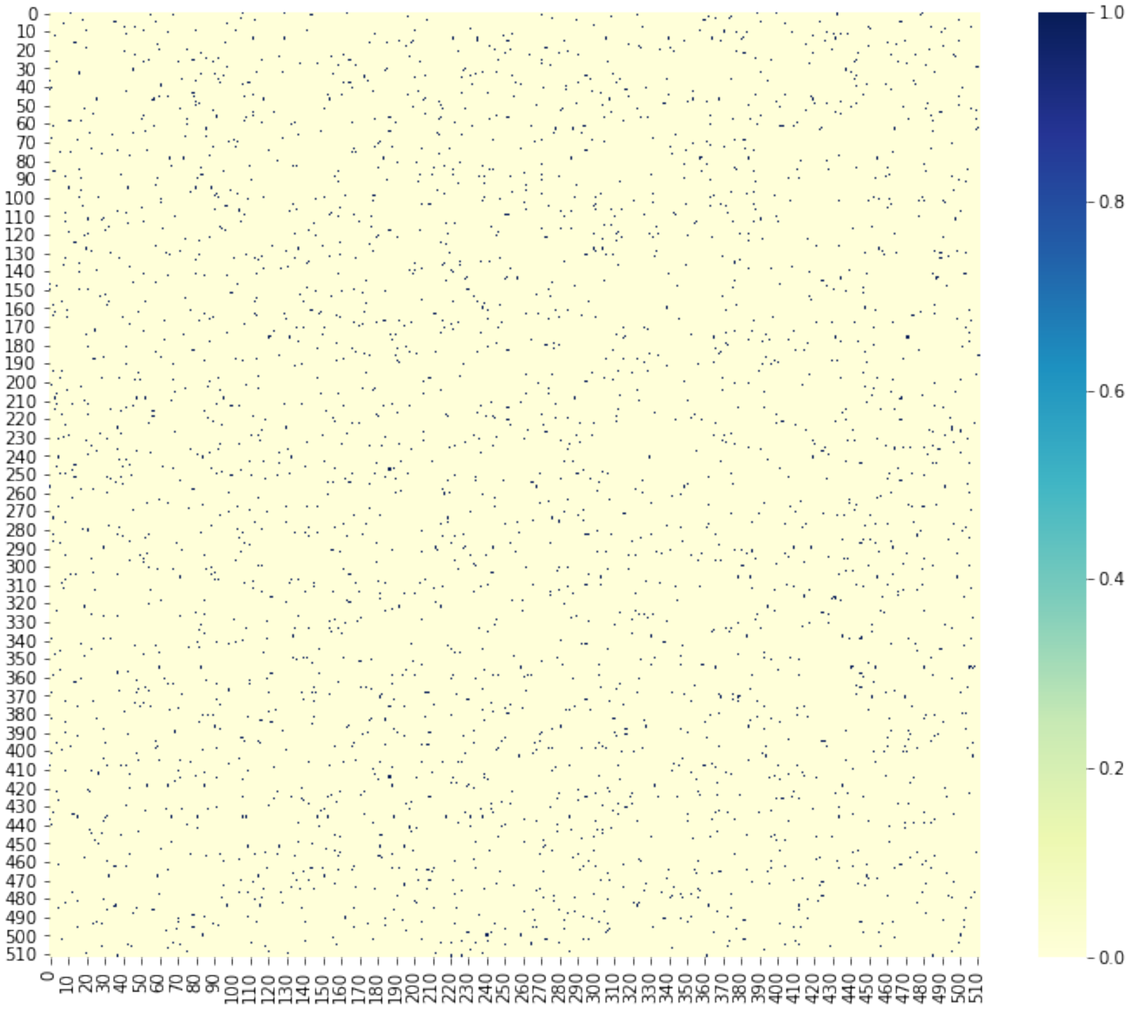
Spicy Package RCM with symmetric_mode = True



After reordering, we are using 143 crossbar

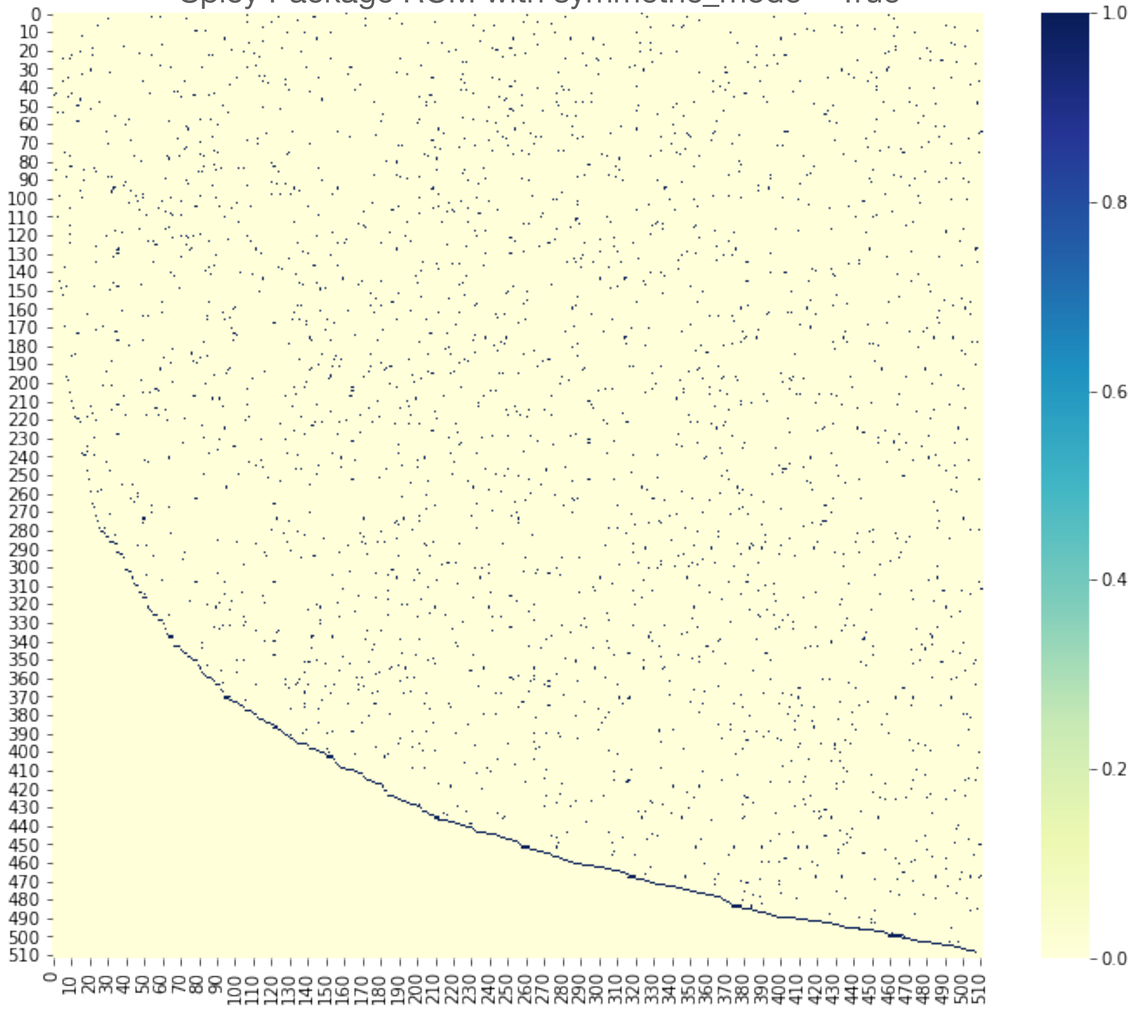
Do reordering to each small square matrix (512,512)

Layer8: (512,512,1,1) *9 Sparsity = 25496 / 2359296 = 1%



We are using 16 crossbar for each square matrix

Spicy Package RCM with symmetric_mode = True



After reordering, we are using 15 crossbar

Vgg11 -cifar10 with global sparsity = 4.39%

Crossbar perspective:

```
Layer1:  (64, 3, 3, 3) | Sparsity = 0.7858 | CrossBar: left/total: 1/1
Layer2:  (128, 64, 3, 3) | Sparsity = 0.3515 | CrossBar: left/total: 5/5
Layer3:  (256, 128, 3, 3) | Sparsity = 0.1504 | CrossBar: left/total: 18/18
Layer4:  (256, 256, 3, 3) | Sparsity = 0.081 | CrossBar: left/total: 36/36
Layer5:  (512, 256, 3, 3) | Sparsity = 0.0357 | CrossBar: left/total: 72/72
Layer6:  (512, 512, 3, 3) | Sparsity = 0.0105 | CrossBar: left/total: 144/144
Layer7:  (512, 512, 3, 3) | Sparsity = 0.0101 | CrossBar: left/total: 144/144
Layer8:  (512, 512, 3, 3) | Sparsity = 0.0108 | CrossBar: left/total: 144/144
Layer9:  (512, 512) | Sparsity = 0.4486 | CrossBar: left/total: 16/16
Layer10:  (256, 512) | Sparsity = 0.5121 | CrossBar: left/total: 8/8
Layer11:  (10, 256) | Sparsity = 0.8406 | CrossBar: left/total: 2/2
Total Sparsity = 422805/9613504 = 0.043980321847268175
```

Columns and rows perspective:

```
Layer1 | (64, 3, 3, 3) | nz_cols/total_cols = 64/64=1.0 | nz_rows/total_rows = 27/27 = 1.0
Layer2 | (128, 64, 3, 3) | nz_cols/total_cols = 640/640=1.0 | nz_rows/total_rows = 576/576 = 1.0
Layer3 | (256, 128, 3, 3) | nz_cols/total_cols = 2304/2304=1.0 | nz_rows/total_rows = 2304/2304 = 1.0
Layer4 | (256, 256, 3, 3) | nz_cols/total_cols = 4608/4608=1.0 | nz_rows/total_rows = 4607/4608 = 0.9997
Layer5 | (512, 256, 3, 3) | nz_cols/total_cols = 9025/9216=0.9792 | nz_rows/total_rows = 9018/9216 = 0.9785
Layer6 | (512, 512, 3, 3) | nz_cols/total_cols = 9526/18432=0.5168 | nz_rows/total_rows = 12736/18432 = 0.6909
Layer7 | (512, 512, 3, 3) | nz_cols/total_cols = 9404/18432=0.5101 | nz_rows/total_rows = 6366/18432 = 0.3453
Layer8 | (512, 512, 3, 3) | nz_cols/total_cols = 9920/18432=0.5381 | nz_rows/total_rows = 8988/18432 = 0.4876
Layer9 | (512, 512) | nz_cols/total_cols = 2048/2048=1.0 | nz_rows/total_rows = 2048/2048 = 1.0
Layer10 | (256, 512) | nz_cols/total_cols = 1024/1024=1.0 | nz_rows/total_rows = 1024/1024 = 1.0
Layer11 | (10, 256) | nz_cols/total_cols = 20/20=1.0 | nz_rows/total_rows = 256/256 = 1.0
In all: nz_cols/total_cols = 48583/75220 = 0.6458 | nz_rows/total_rows = 47950/75355 = 0.6363
```

Summary

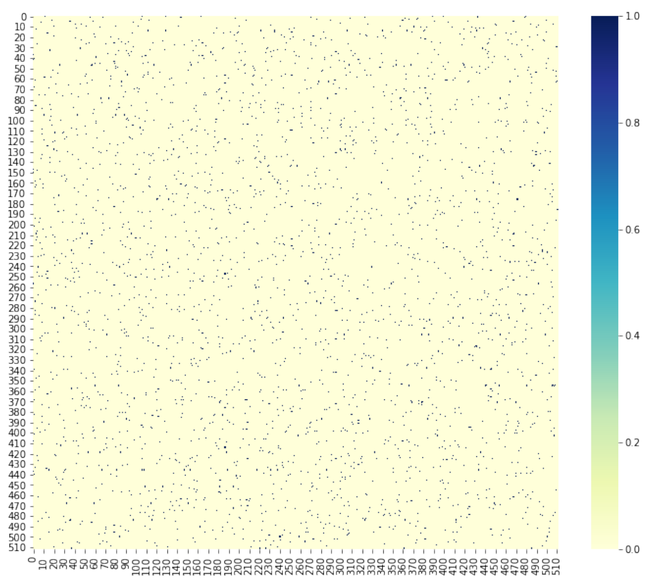
1. We have two versions of RCM:
    1. Do reordering within the rectangular matrix
    2. Do reordering within the square matrix
2. Reordering within the rectangular matrix can save a few crossbar. For instance, we can save 1 crossbars for layer8 in vgg11 (144 crossbar in total). If we do ordering on each small matrix(512,512), we can save 9 crossbar.

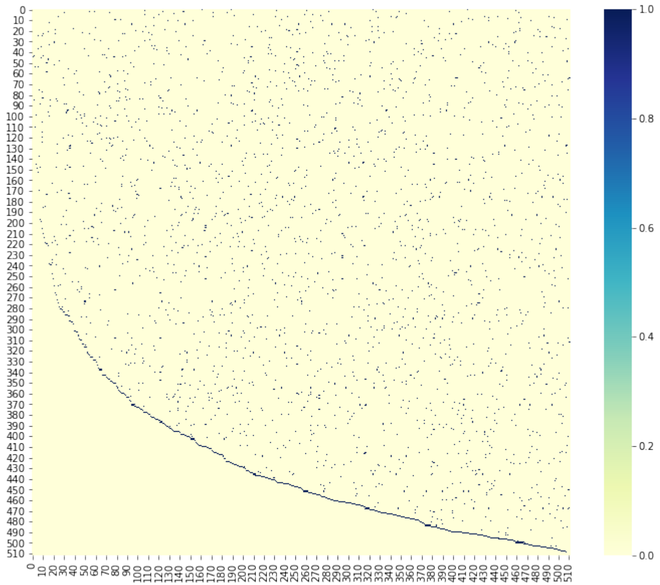
3. Reordering within the square matrix will have worse performance if the matrix is getting larger.

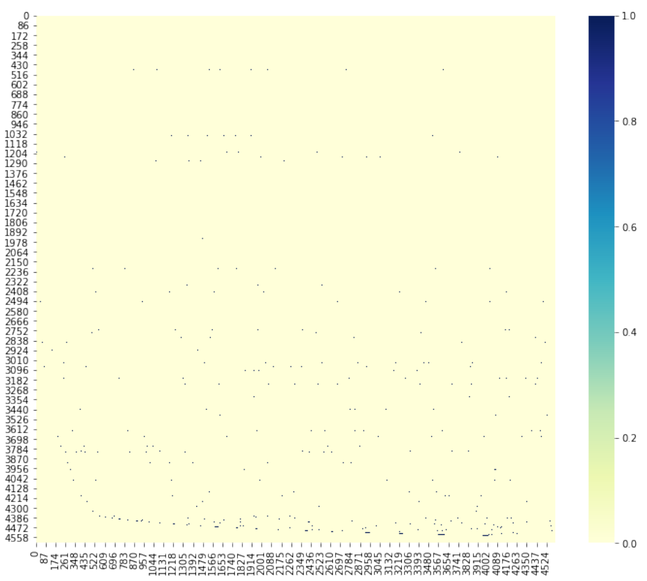
Recap:

1. We have two versions of RCM:
    1. Do reordering within the rectangular matrix
    2. Do reordering within the square matrix
2. Reordering within the rectangular matrix can save a few crossbar. For instance, we can save 1 crossbars for layer8 in vgg11 (144 crossbar in total). If we do ordering on each small matrix(512,512), we can save 9 crossbar.

3. Reordering within the square matrix will have worse performance if the matrix is getting larger.





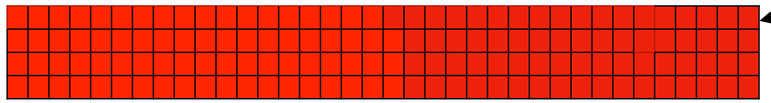


Vgg11-cifar10 with global sparsity = 4.4%

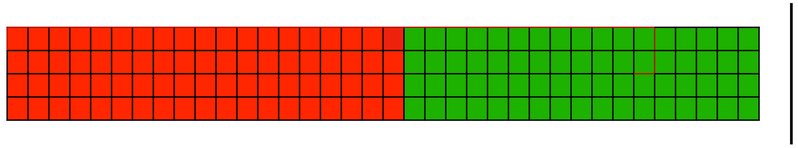
LayerName		Local Weights	Crossbar (size 128*128)	Cols Sparsity	Rows Sparsity	Save
Layer1	(64, 3, 3, 3)	0.786	1	1	1	0
Layer2	(128, 64, 3, 3)	0.352	5	1	1	0
Layer3	(256, 128, 3, 3)	0.150	18	1	1	0
Layer4	(256, 256, 3, 3)	0.081	36	1	0.99	0
Layer5	(512, 256, 3, 3)	0.035	72	0.98	0.98	1
Layer6	(512, 512, 3, 3)	0.011	144	0.52	0.69	69
Layer7	(512, 512, 3, 3)	0.010	144	0.51	0.34	94
Layer8	(512, 512, 3, 3)	0.011	144	0.53	0.49	73
Layer9	(512, 512)	0.449	16	1	1	0
Layer10	(256, 512)	0.512	8	1	1	0
Layer11	(10, 256)	0.841	2	1	1	0
Total		0.044	590	0.65	0.63	

Example1
Layer7: (512, 512, 3, 3) = (128*4, 128*36)
144 crossbars with layer sparsity = 1%
Nonzero Columns: 9044/18432, Nonzero rows: 6366/18432

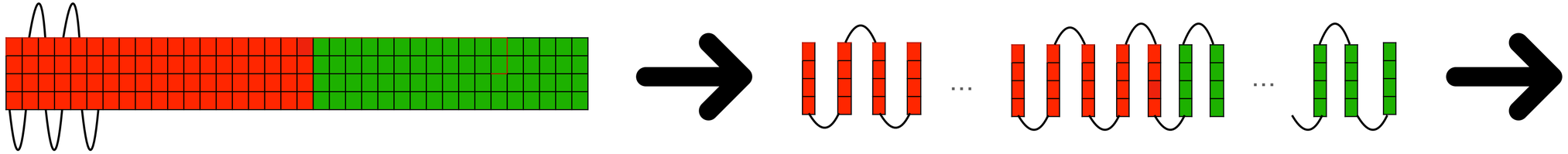
Each square means a crossbar(128*128)
The whole rectangular has 4*36 = 144 crossbars



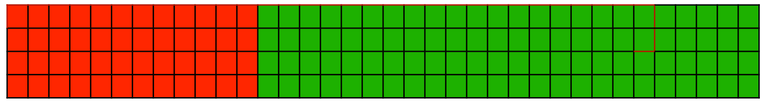
1. In total, move all zero columns forward, we can save 68 crossbars now



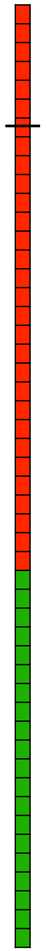
2. Catenate all crossbar and keep move nonzero rows forward



3. Swap rows and move all nonzero rows together



We can save 94 crossbars now



Resnet18-cifar10 with global sparsity = 4.4%

LayerName		Local Weights	Crossbar	Cols Sparsity	Rows Sparsity	Save
Layer1	(64, 3, 3, 3)	0.5711	1	64/64=1.0	27/27=1.0	0
Layer2	(64, 64, 3, 3)	0.5021	5	320/320=1.0	576/576=1.0	0
Layer3	(64, 64, 3, 3)	0.4717	5	320/320=1.0	576/576=1.0	0
Layer4	(64, 64, 3, 3)	0.4811	5	320/320=1.0	576/576=1.0	0
Layer5	(64, 64, 3, 3)	0.4648	5	320/320=1.0	576/576=1.0	0
Layer6	(128, 64, 3, 3)	0.3459	5	640/640=1.0	576/576=1.0	0
Layer7	(128, 128, 3, 3)	0.2592	9	1152/1152=1.0	1152/1152=1.0	0
Layer8	(128, 128, 3, 3)	0.2551	9	1152/1152=1.0	1152/1152=1.0	0
Layer9	(128, 128, 3, 3)	0.2453	9	1152/1152=1.0	1152/1152=1.0	0
Layer10	(256, 128, 3, 3)	0.0952	18	2304/2304=1.0	2304/2304=1.0	0
Layer11	(256, 256, 3, 3)	0.0473	36	4594/4608=0.99	4597/4608=0.99	0
Layer12	(256, 256, 3, 3)	0.0468	36	4597/4608=0.99	4599/4608=0.99	0
Layer13	(256, 256, 3, 3)	0.0519	36	4596/4608=0.99	4603/4608=0.99	0
Layer14	(512, 256, 3, 3)	0.0122	72	6979/9216=0.75	7202/9216=0.78	17
Layer15	(512, 512, 3, 3)	0.0051	144	8517/18432=0.46	8724/18432=0.47	77
Layer16	(512, 512, 3, 3)	0.0031	144	4372/18432=0.23	5829/18432=0.31	109
Layer17	(512, 512, 3, 3)	0.0035	144	5560/18432=0.3	4728/18432=0.25	107
Layer18	(10, 512)	0.7843	4	40/40=1.0	512/512=1.0	0
			687			310

Saved/total = 310/687 = 45%

Resnet18-cifar10 with global sparsity = 1.8%

LayerName		Local Weights	Crossbar	Cols Sparsity	Rows Sparsity	Save
Layer1	(64, 3, 3, 3)		1			0
Layer2	(64, 64, 3, 3)		5			2
Layer3	(64, 64, 3, 3)		5			2
Layer4	(64, 64, 3, 3)		5			2
Layer5	(64, 64, 3, 3)		5			2
Layer6	(128, 64, 3, 3)		5			0
Layer7	(128, 128, 3, 3)		9			0
Layer8	(128, 128, 3, 3)		9			0
Layer9	(128, 128, 3, 3)		9			0
Layer10	(256, 128, 3, 3)		18			1
Layer11	(256, 256, 3, 3)		36			6
Layer12	(256, 256, 3, 3)		36			7
Layer13	(256, 256, 3, 3)		36			14
Layer14	(512, 256, 3, 3)		72			57
Layer15	(512, 512, 3, 3)		144			128
Layer16	(512, 512, 3, 3)		144			136
Layer17	(512, 512, 3, 3)		144			134
Layer18	(10, 512)		4			3
			687			494

Resnet18-cifar10 with global sparsity = 0.8%

LayerName		Local Weights	Crossbar	Cols Sparsity	Rows Sparsity	Save
Layer1	(64, 3, 3, 3)		1			0
Layer2	(64, 64, 3, 3)		5			2
Layer3	(64, 64, 3, 3)		5			2
Layer4	(64, 64, 3, 3)		5			2
Layer5	(64, 64, 3, 3)		5			2
Layer6	(128, 64, 3, 3)		5			0
Layer7	(128, 128, 3, 3)		9			1
Layer8	(128, 128, 3, 3)		9			1
Layer9	(128, 128, 3, 3)		9			1
Layer10	(256, 128, 3, 3)		18			7
Layer11	(256, 256, 3, 3)		36			19
Layer12	(256, 256, 3, 3)		36			20
Layer13	(256, 256, 3, 3)		36			27
Layer14	(512, 256, 3, 3)		72			67
Layer15	(512, 512, 3, 3)		144			138
Layer16	(512, 512, 3, 3)		144			140
Layer17	(512, 512, 3, 3)		144			139
Layer18	(10, 512)		4			3
			687			571

Lenet-mnist with global sparsity = 4.4%

LayerName		Local Weights Sparsity	Cols Sparsity	Rows Sparsity	Crossbar (size 128*128)	Saved
Layer1	(6, 1, 5, 5)	0.4066	6/6=1.00	24/25=0.96	1	0
Layer2	(16, 6, 5, 5)	0.0466	25/32=0.78	80/150=0.53	2	1
Layer3	(120, 400)	0.0048	166/480=0.35	178/400=0.45	4	2
Layer4	(84, 120)	0.1775	84/84=1.00	120/120 = 1.00	1	0
Layer5	(10, 84)	0.6047	10/10=1.00	84/84=1.00	1	0
Total			0.47	0.63	9	3

Layer2: Save 1
Layer3: Save 2
Saved crossbar / total = 3/9 = 33.3%
Left crossbar / total = 6/9 = 66.7%

vgg11-cifar10

LayerName		Crossbar (size 128*128)	Save (sparsity = 10%)	Save (sparsity = 7%)	Save (sparsity = 4%)
Layer1	(64, 3, 3, 3)	1	0	0	0
Layer2	(128, 64, 3, 3)	5	0	0	0
Layer3	(256, 128, 3, 3)	18	0	0	0
Layer4	(256, 256, 3, 3)	36	0	0	0
Layer5	(512, 256, 3, 3)	72	0	0	1
Layer6	(512, 512, 3, 3)	144	0	19	69
Layer7	(512, 512, 3, 3)	144	0	51	94
Layer8	(512, 512, 3, 3)	144	0	21	73
Layer9	(512, 512)	16	0	0	0
Layer10	(256, 512)	8	0	0	0
Layer11	(10, 256)	2	0	0	0
Total		590	0	91	237
Ratio			0%	15%	40%

vgg16-cifar10

LayerName		Crossbar (size 128*128)	Save (sparsity = 4%)	Save (sparsity = 2%)	Save (sparsity = 1%)
Layer1	(64, 3, 3, 3)	1	0	0	0
Layer2	(64, 64, 3, 3)	5	2	2	2
Layer3	(128, 64, 3, 3)	5	0	0	0
Layer4	(128, 128, 3, 3)	9	0	0	0
Layer5	(256, 128, 3, 3)	18	0	0	6
Layer6	(256, 256, 3, 3)	36	0	3	23
Layer7	(256, 256, 3, 3)	36	0	3	25
Layer8	(512, 256, 3, 3)	72	5	25	66
Layer9	(512, 512, 3, 3)	144	38	105	139
Layer10	(512, 512, 3, 3)	144	27	96	140
Layer11	(512, 512, 3, 3)	144	15	83	140
Layer12	(512, 512, 3, 3)	144	12	76	139
Layer13	(512, 512, 3, 3)	144	13	80	139
Layer14	(512, 512)	16	0	0	0
Layer15	(256, 512)	8	0	0	0
Layer16	(10, 256)	2	1	1	1
		928	113	474	820
saved/total			12%	51%	88%

Resnet18-cifar10

LayerName		Crossbar (size 128*128)	Save (sparsity = 4%)	Save (sparsity = 2%)	Save (sparsity = 1%)
Layer1	(64, 3, 3, 3)	1	0	0	0
Layer2	(64, 64, 3, 3)	5	0	2	2
Layer3	(64, 64, 3, 3)	5	0	2	2
Layer4	(64, 64, 3, 3)	5	0	2	2
Layer5	(64, 64, 3, 3)	5	0	2	2
Layer6	(128, 64, 3, 3)	5	0	0	0
Layer7	(128, 128, 3, 3)	9	0	0	1
Layer8	(128, 128, 3, 3)	9	0	0	1
Layer9	(128, 128, 3, 3)	9	0	0	1
Layer10	(256, 128, 3, 3)	18	0	1	7
Layer11	(256, 256, 3, 3)	36	0	6	19
Layer12	(256, 256, 3, 3)	36	0	7	20
Layer13	(256, 256, 3, 3)	36	0	14	27
Layer14	(512, 256, 3, 3)	72	17	57	67
Layer15	(512, 512, 3, 3)	144	77	128	138
Layer16	(512, 512, 3, 3)	144	109	136	140
Layer17	(512, 512, 3, 3)	144	107	134	139
Layer18	(10, 512)	4	0	3	3
Total		687	310	494	571
Ratio			45%	71%	83%

Summary

VGG11 sparsity = 4%
layer6: (512, 512, 3, 3) = (128*4, 128*36): save 69
layer7: (512, 512, 3, 3) = (128*4, 128*36): save 94
layer8: (512, 512, 3, 3) = (128*4, 128*36): save 73

saved (zero crossbars)/total = 40%
Nonzero crossbars/total = 60%

Recap:

VGG11 sparsity = 4%
layer6: (512, 512, 3, 3) = (128*4, 128*36): save 69
layer7: (512, 512, 3, 3) = (128*4, 128*36): save 94
layer8: (512, 512, 3, 3) = (128*4, 128*36): save 73

saved (zero crossbars)/total = 40%
Nonzero crossbars/total = 60%

VGG19-CIFAR10 on different crossbar size 128, 64, 8 and different global density = 4%, 2%,1%

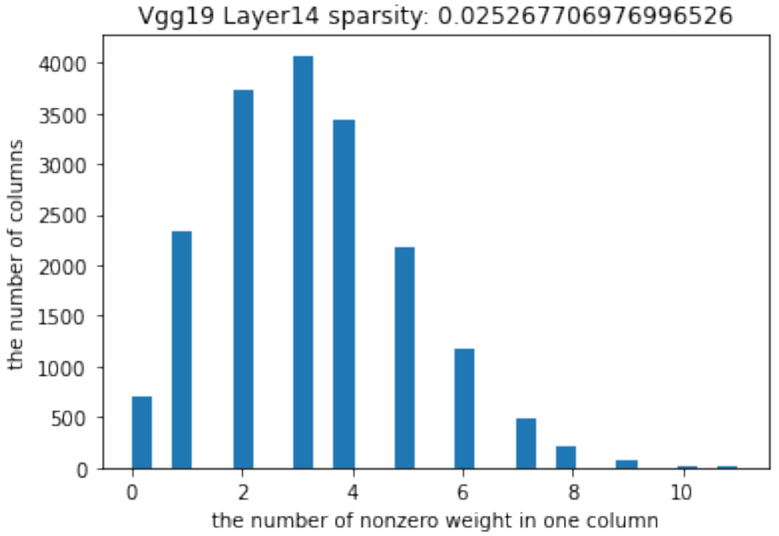
LayerName	
Layer1	(64, 3, 3, 3)
Layer2	(64, 64, 3, 3)
Layer3	(128, 64, 3, 3)
Layer4	(128, 128, 3, 3)
Layer5	(256, 128, 3, 3)
Layer6	(256, 256, 3, 3)
Layer7	(256, 256, 3, 3)
Layer8	(256, 256, 3, 3)
Layer9	(512, 256, 3, 3)
Layer10	(512, 512, 3, 3)
Layer11	(512, 512, 3, 3)
Layer12	(512, 512, 3, 3)
Layer13	(512, 512, 3, 3)
Layer14	(512, 512, 3, 3)
Layer15	(512, 512, 3, 3)
Layer16	(512, 512, 3, 3)
Layer17	(512, 512)
Layer18	(256, 512)
Layer19	(10, 256)
saved/total	

Crossbar. size 128*128			
Total	4%	2%	1%
1	0	0	0
5	2	2	2
5	0	0	0
9	0	0	0
18	0	0	2
36	0	2	14
36	0	2	16
36	0	4	19
72	6	30	59
144	35	95	134
144	24	80	133
144	14	67	133
144	5	52	132
144	4	47	132
144	4	44	132
144	5	49	132
16	0	0	0
8	0	0	0
2	1	1	1
1249	100	475	1041
	8%	38%	83%

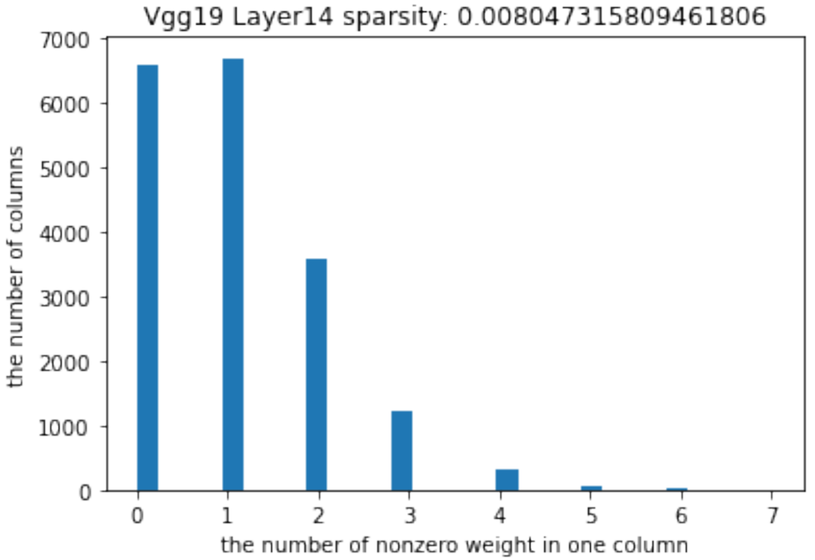
Crossbar. size 64*64			
Total	4%	2%	1%
1	0	0	0
9	0	0	0
18	0	0	0
36	0	0	4
72	0	6	28
144	8	38	89
144	10	42	97
144	10	46	105
288	83	186	262
576	285	470	555
576	229	428	555
576	176	391	555
576	114	345	553
576	99	329	552
576	97	319	553
576	106	334	552
64	0	0	0
32	0	0	0
4	3	3	3
4983	1220	2937	4463
	24%	58%	89%

Crossbar. size 8*8			
Total	4%	2%	1%
32	5	5	5
576	28	55	158
1152	239	400	704
2304	801	1205	1781
4608	2592	3411	4097
9216	6498	7813	8683
9216	6544	7868	8759
9216	6613	7991	8858
18432	15749	17443	18217
36864	33767	35941	36696
36864	32721	35480	36693
36864	31621	35078	36692
36864	30054	34561	36676
36864	29524	34352	36672
36864	29463	34237	36677
36864	29679	34401	36672
4096	23	72	431
2048	6	16	108
64	24	24	24
318912	25595	29035	30860
	80%	91.1%	96.7%

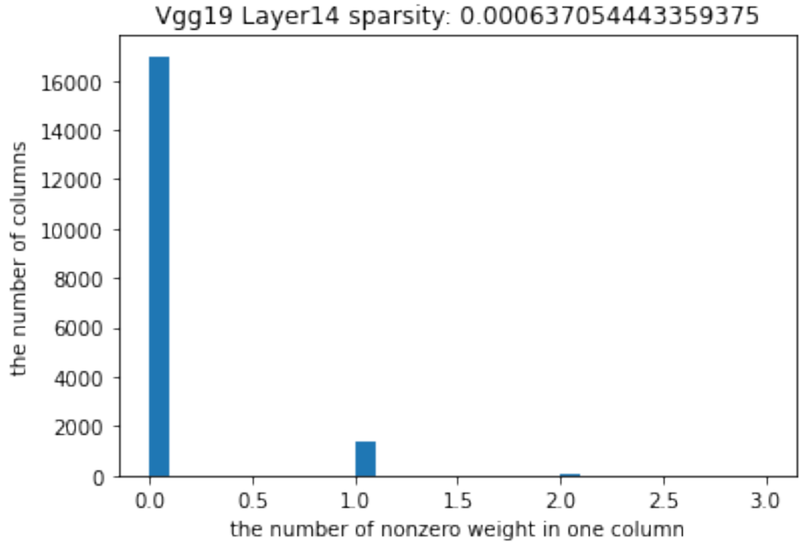
Vgg19 layer13



Global sparsity = 4%



Global sparsity = 2%



Global sparsity = 1%

1. Assume weights are evenly distributed
2. The probability of a weight is nonzero is the density of a particular layer
3. Independent to each other

VGG19 -CIFAR10		Global density = 4%	Global density = 2%	Global density = 1%
column size		Layer13 density = 0.025	Layer13 density = 0.008	Layer13 density = 0.0006
8	Theoretical	Pr (zero col)= (1- 0.025)^8 = **81.6%**	Pr (zero col)=(1- 0.008)^8 = **93.7%**	Pr (zero col)=(1- 0.0006)^8 = **99.5%**
	Experimental	**81.5%**	**93.7%**	**99.4%**
64	Theoretical	Pr (zero col)=(1- 0.025)^64 = **19.8%**	Pr (zero col)=(1- 0.008)^64 = **59.8%**	Pr (zero col)=(1- 0.0006)^64 = **96.2%**
	Experimental	**19.7%**	**59.9%**	**96.0%**
128	Theoretical	Pr (zero col)=(1- 0.025)^128 = **4%**	Pr (zero col)=(1- 0.008)^128= **35.7%**	Pr (zero col)=(1- 0.0006)^128 = **92.6%**
	Experimental	**4%**	**35.9%**	**91.9%**

It also work for other CNNs like Resnet18, VGG11, etc.