Neural Network Approximation

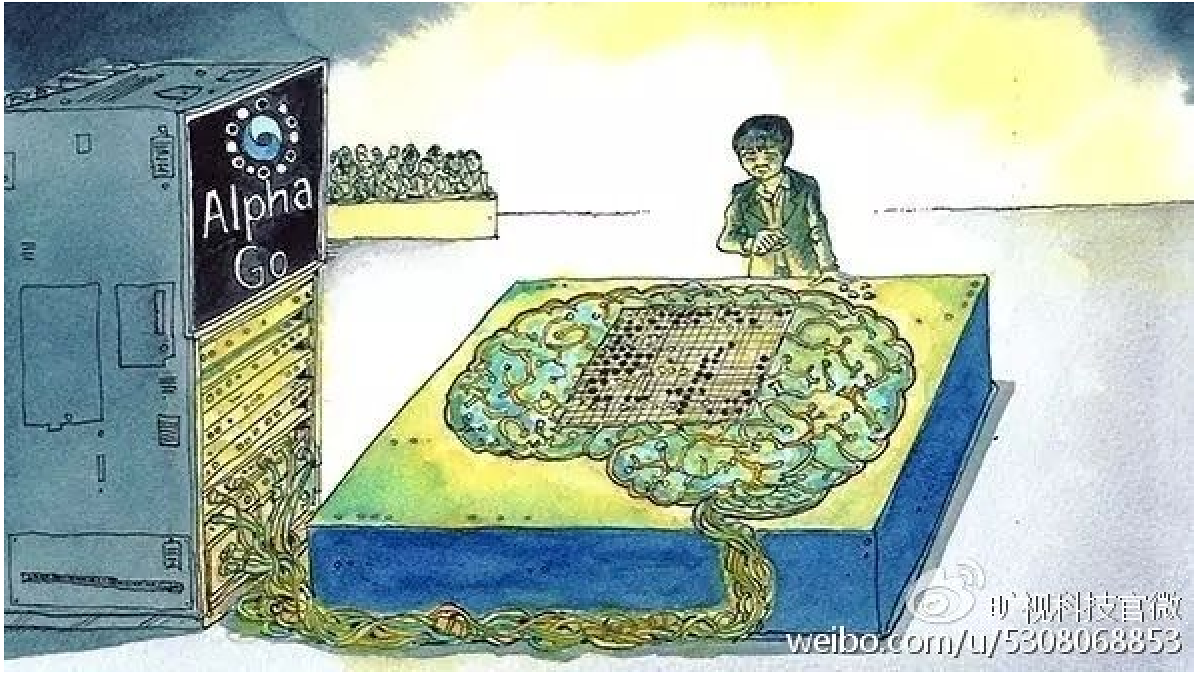
Low rank, Sparsity, and Quantization

zsc@megvii.com

Oct. 2017

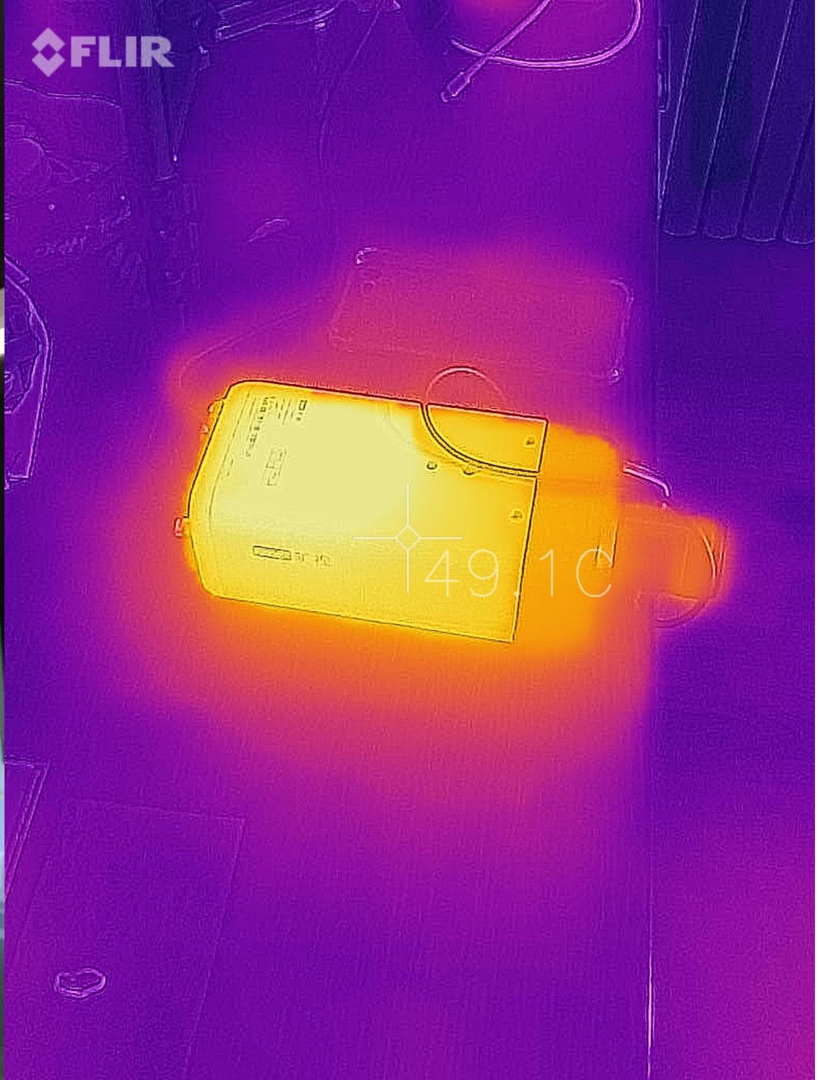
Motivation

- Faster Inference
    - Latency critical scenarios
        - VR/AR, UGV/UAV
    - Saves time and energy
- Faster Training
    - Higher iteration speed
    - Saves life
- Smaller
    - storage size
    - memory footprint



Lee Sedol v.s. AlphaGo

83 Watt 77 kWatt



Face+


6, QA: 0.86
7, QA: 0.87
3, QA: 0.88
4, QA: 0.9
5, QA: 0.9
13, QA: 0.9
15, QA: 0.88
16, QA: 0.81
17, QA: 0.86
24, QA: 0.91
27, QA: 0.87
30, QA: 0.9
32, QA: 0.88
34, QA: 0.89
42, QA: 0.84
50, QA: 0.84
53, QA: 0.83
56, QA: 0.85
60, QA: 0.87
65, QA: 0.89
71, QA: 0.91
81, QA: 0.85
86, QA: 0.9
87, QA: 0.88
92, QA: 0.87
94, QA: 0.83

Neural Network

Machine Learning as Optimization

- Supervised learning
  - θ is the parameter
  - $\hat{y}$ is output, X is input
  - y is ground truth
  - d is the objective function

$$\min_\theta d(y, \hat{y})$$
$$\text{where } \hat{y} = f(X, \theta)$$

- Unsupervised learning
  - some oracle function **r**: low rank, sparse, K

$$\min_\theta r(\hat{y})$$
$$\text{where } \hat{y} = f(X, \theta)$$

Machine Learning as Optimization

- Regularized supervised learning
  - $\min_\theta d(y, \hat{y}) + r(\hat{y})$
    where $\hat{y} = f(X, \theta)$
- Probabilistic interpretation

$$d(y, \hat{y}) = -\log p(y|\hat{y})$$
$$r(\hat{y}) = -\log p(\hat{y})$$
$$\Rightarrow d(y, \hat{y}) + r(\hat{y}) = -\log p(y)$$

  - *d* measures conditional probability
  - *r* measures prior probability
  - Probability approach is more constrained than the optimization-approach due to normalization problem
    - Not easy to represent uniform distribution over [0, \infty]

Gradient descent

$$\min_\theta d(y, \hat{y}) + r(\hat{y})$$
$$\text{where } \hat{y} = f(X, \theta)$$

- Can be solved by an ODE: $\dot{\theta} = -\frac{\partial C(\theta(t))}{\partial \theta}$
  - Discretizing with step length λ we get gradient descent with learning rate λ
  - $\dot{\theta} \approx \frac{\theta_{t+\lambda} - \theta_t}{\lambda} \xrightarrow{\text{Derive}} \theta_{t+\lambda} = \theta_t - \lambda \frac{\partial C(\theta)}{\partial \theta}$
- Convergence proof

$$\frac{\mathrm{d}C(\theta)}{\mathrm{d}t} = \frac{\partial C(\theta)}{\partial \theta} \frac{\mathrm{d}\theta}{\mathrm{d}t} = -\left(\frac{\partial C(\theta)}{\partial \theta}\right)^2 \leq 0$$

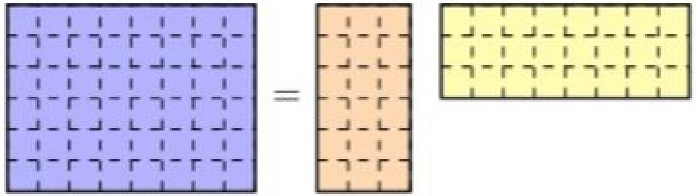
Linear Regression

- $\min_W \|y - Wx\|^2$
  - $\hat{y} = f(X, \theta) = WX$
  - $d(y, \hat{y}) = \|y - \hat{y}\|^2$
  - *x* is input, *\hat{y}* is prediction, *y* is ground truth.
  - ***W*** with dimension (m,n)
  - #param = *m n*, #OPs = *m n*

$$y \approx \hat{y} = Wx$$

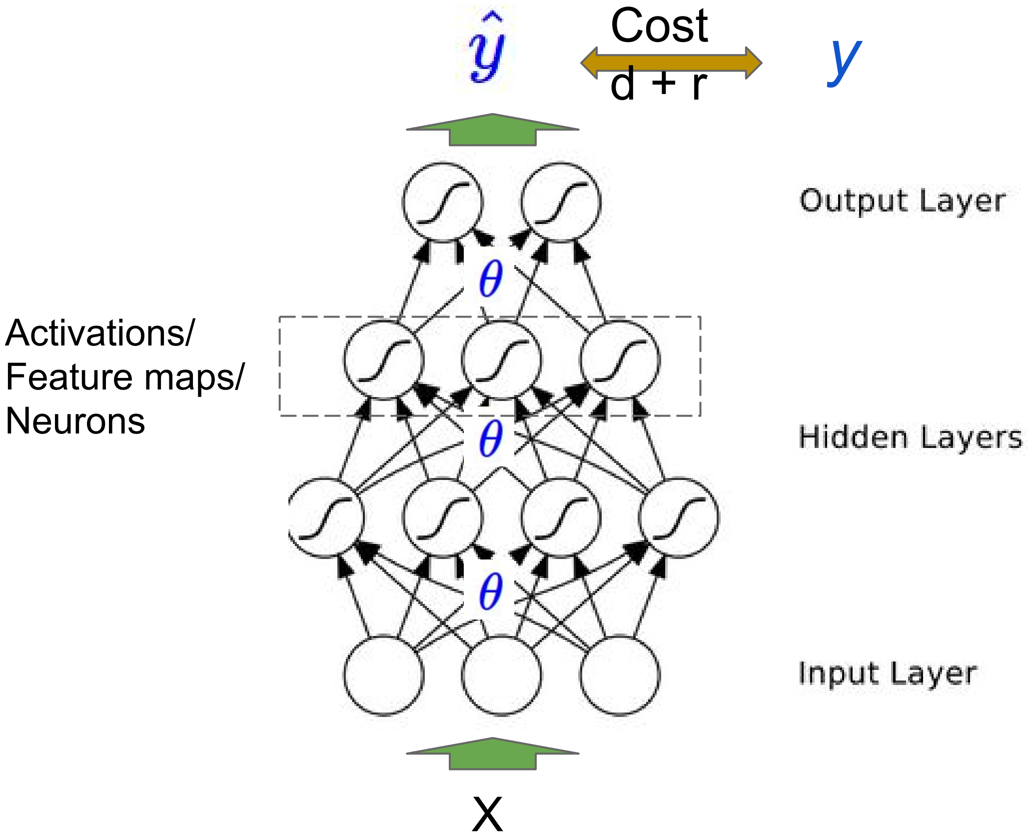
Fully-connected

- $y \approx W_2 f(W_1(x))$
  - In general, will use nonlinearity to increase “model capacity”.
  - Make sense if ***f*** is identity? I.e. f(x) = x?
    - Sometimes, if W_2 is ***m*** by ***r*** and W_1 is ***r*** by ***n***, then W_2 W_1 is a matrix of rank ***r***, which is different from a ***m*** by ***n*** matrix.



- $y \approx W_3(f(W_2 f(W_1(x))))$
  - Deep learning!

Neural Network



Gradient descent

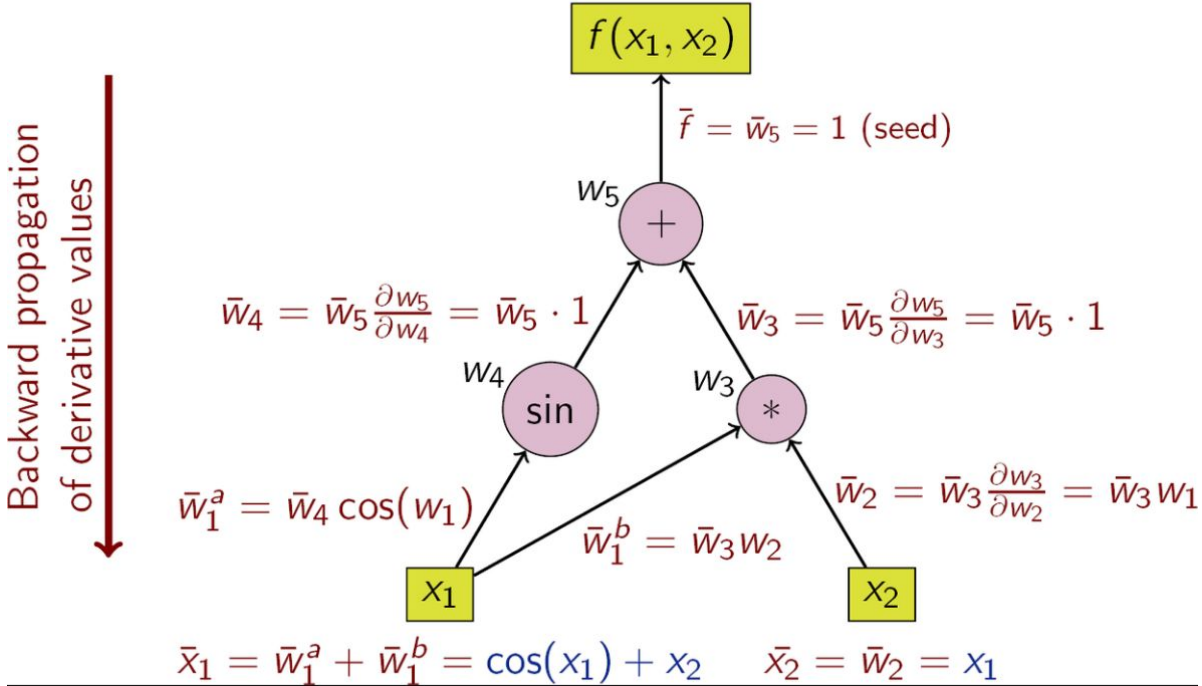
$$\min_\theta d(y, \hat{y}) + r(\hat{y})$$
$$\text{where } \hat{y} = f(X, \theta)$$

- Can be solved by an ODE: $\dot{\theta} = -\frac{\partial C(\theta(t))}{\partial \theta}$
  - Discretizing with step length λ we get gradient descent with learning rate λ
  - $\dot{\theta} \approx \frac{\theta_{t+\lambda} - \theta_t}{\lambda} \xrightarrow{\text{Derive}} \theta_{t+\lambda} = \theta_t - \lambda \frac{\partial C(\theta)}{\partial \theta}$
- Convergence proof

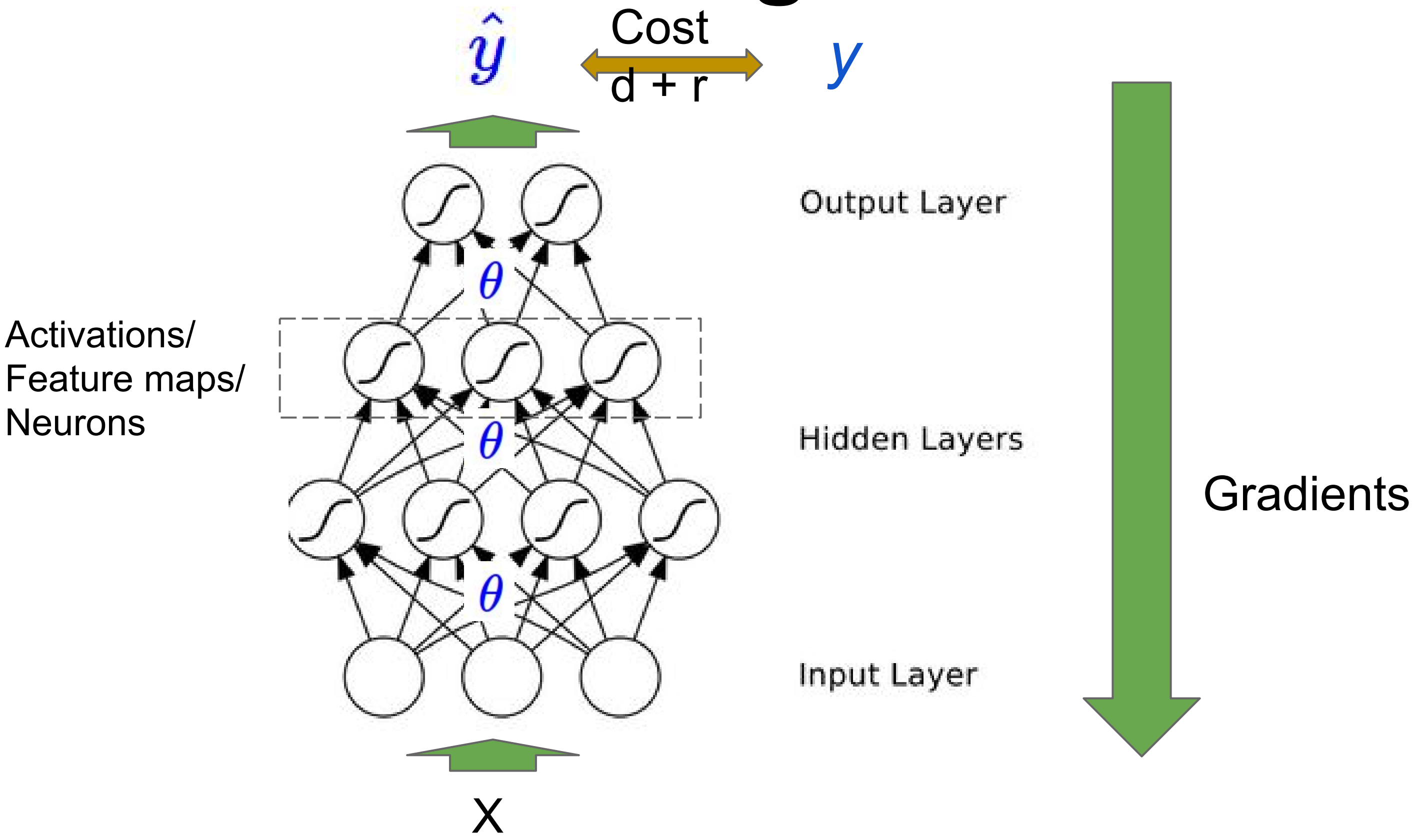
$$\frac{\mathrm{d}C(\theta)}{\mathrm{d}t} = \frac{\partial C(\theta)}{\partial \theta} \frac{\mathrm{d}\theta}{\mathrm{d}t} = -\left(\frac{\partial C(\theta)}{\partial \theta}\right)^2 \leq 0$$

Backpropagation

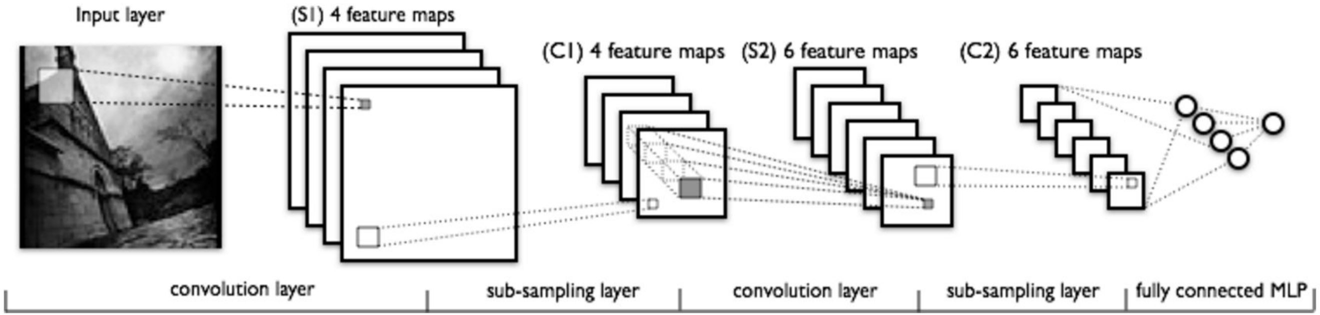
$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial w_1}\frac{\partial w_1}{\partial x} = \left(\frac{\partial y}{\partial w_2}\frac{\partial w_2}{\partial w_1}\right)\frac{\partial w_1}{\partial x} = \left(\left(\frac{\partial y}{\partial w_3}\frac{\partial w_3}{\partial w_2}\right)\frac{\partial w_2}{\partial w_1}\right)\frac{\partial w_1}{\partial x} = \cdots$$

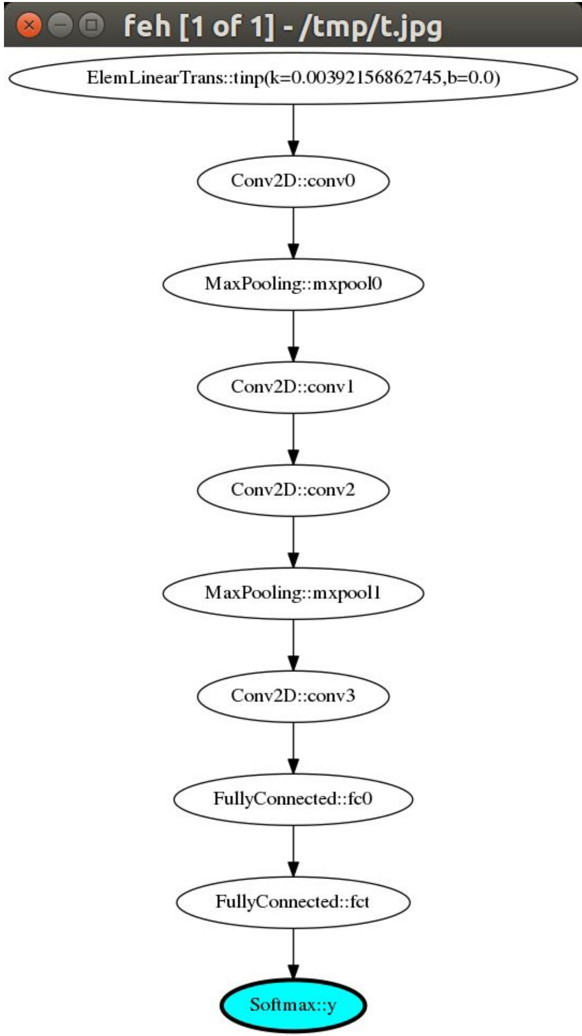


Neural Network Training



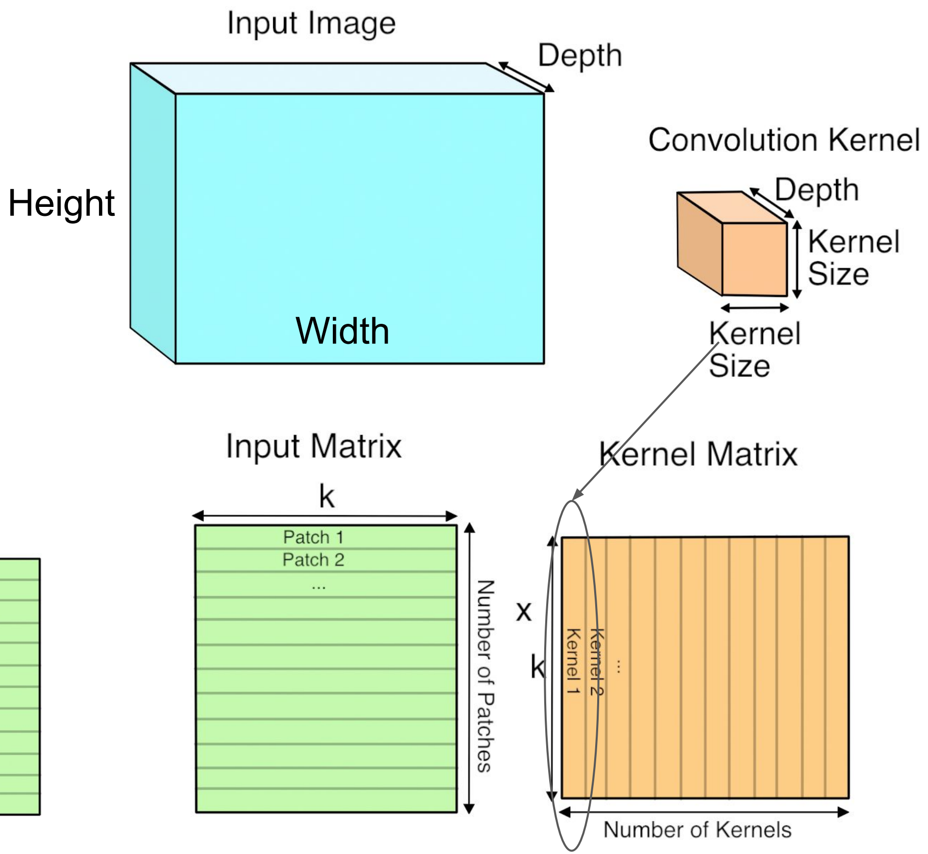
CNN: Alexnet-like



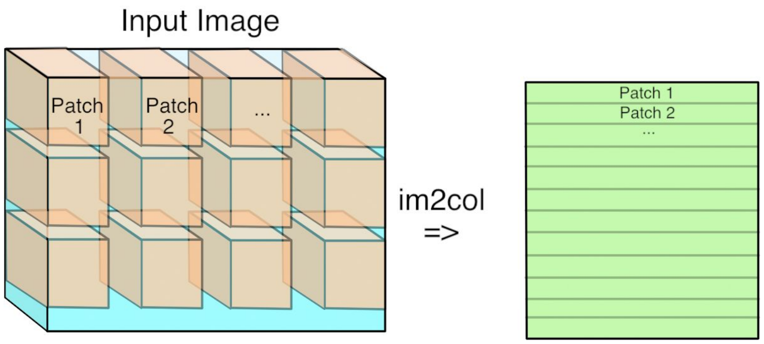


Method 2: Convolution as matrix product

- Convolution
  - feature map <N, C, H’, W’>
  - weights <K, C, H, W>
- Convolution as FC
  - under proper padding, can extract patchs
  - feature map <N H’ W’, C H W>
  - weights <C H W, K>



Kernel stride determines how much overlap



Face++ 旷视

Importance of Convolutions and FC

param_name	shape	#floats	size	perc	
conv0_W	(24, 3, 5, 5)	1800	7.0 KiB	0.21%	
conv0_b	(24,)	24	96.0 B	0.00%	
conv1_W	(32, 24, 3, 3)	6912	27.0 KiB	0.80%	
conv1_b	(32,)	32	128.0 B	0.00%	
conv2_W	(32, 32, 3, 3)	9216	36.0 KiB	1.07%	
conv2_b	(32,)	32	128.0 B	0.00%	
conv3_W	(64, 32, 3, 3)	18432	72.0 KiB	2.14%	
conv3_b	(64,)	64	256.0 B	0.01%	
fc0_W	(1600, 512)	819200	3.1 MiB	95.11%	####################
fc0_b	(512,)	512	2.0 KiB	0.06%	
fct_W	(512, 10)	5120	20.0 KiB	0.59%	
fct_b	(10,)	10	40.0 B	0.00%	
total size		861354	3.3 MiB		

Neupack: inspect_model.py
NeuPeak: npk-model-manip XXX info

Most storage size

Feature map size

layer_name	ispace	ospace	osize	#ops	readable	perc	
tinp	(3, 40, 40)	(3, 40, 40)	4800	0	0.0	0.00%	
conv0	(3, 40, 40)	(24, 36, 36)	31104	2332800	2.2 Mi	32.43%	####################
mxpool0	(24, 36, 36)	(24, 18, 18)	7776	0	0.0	0.00%	
conv1	(24, 18, 18)	(32, 16, 16)	8192	1769472	1.7 Mi	24.60%	###############
conv2	(32, 16, 16)	(32, 14, 14)	6272	1806336	1.7 Mi	25.11%	###############
mxpool1	(32, 14, 14)	(32, 7, 7)	1568	0	0.0	0.00%	
conv3	(32, 7, 7)	(64, 5, 5)	1600	460800	450.0 Ki	6.41%	###
fc0	(64, 5, 5)	(512, 1, 1)	512	819712	800.5 Ki	11.39%	#######
fct	(512, 1, 1)	(10, 1, 1)	10	5130	5.0 Ki	0.07%	
y	(10, 1, 1)	(10, 1, 1)	10	0	0.0	0.00%	
total OPs				7194250	6.9 MiOps		

Most Computation

The Matrix View of Neural Network

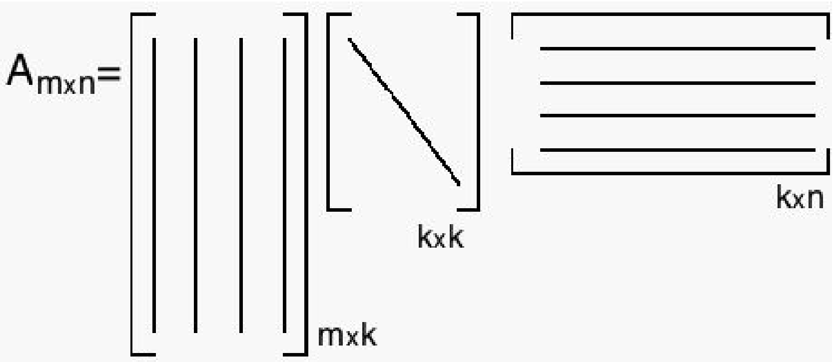
- Weights of FullyConnected and Convolutions layers
  - take up most computation and storage size
  - are representable as matrices
- Approximating the matrices approximates the network
  - The approximation error accumulates.

$$W_a \approx W \Rightarrow f(X, W_a) \approx f(X, W)$$

Low rank Approximation

Singular Value Decomposition

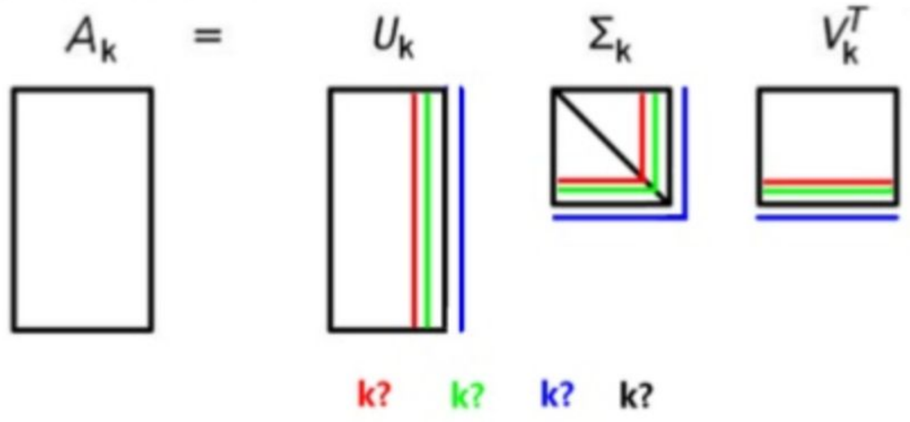
- Matrix deocomposition view
  - A = U S V^T
  - Rows of U, V are orthogonal. S is diagonal.
    - u, s, v^T = np.linalg.svd(x, full_matrices=0,compute_uv=1)
    - The diagonals are non-negative and are in descending order.
    - U^T U = I, but U U^T is not full rank

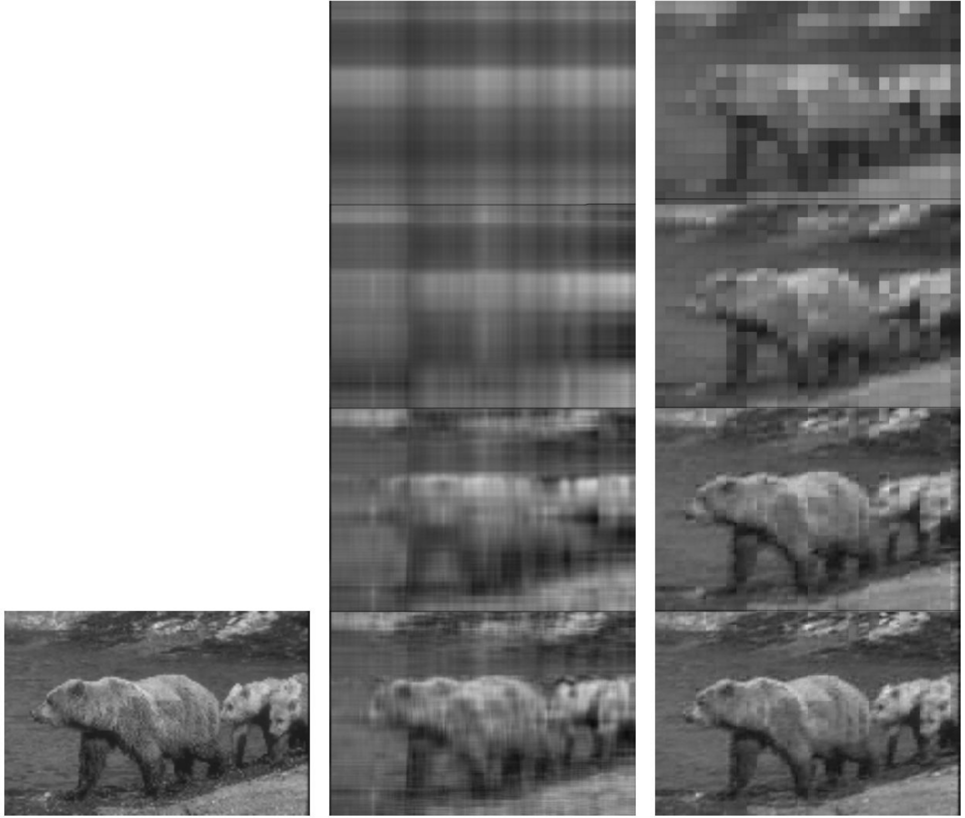


Compact SVD

Truncated SVD

- Assume diagonals of S are in descending order
  - Always achievable
  - Just ignore the blue segments.



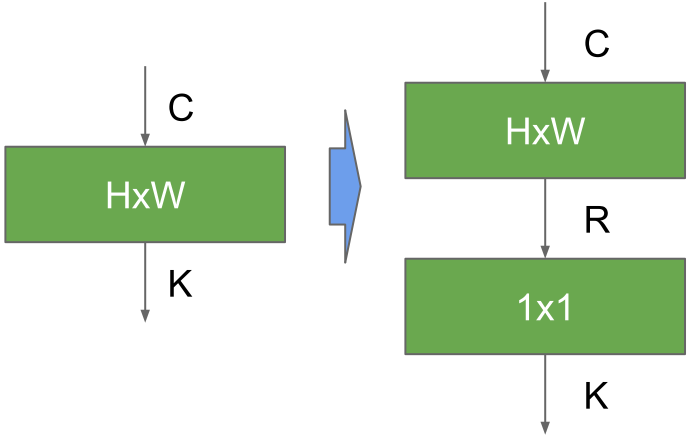


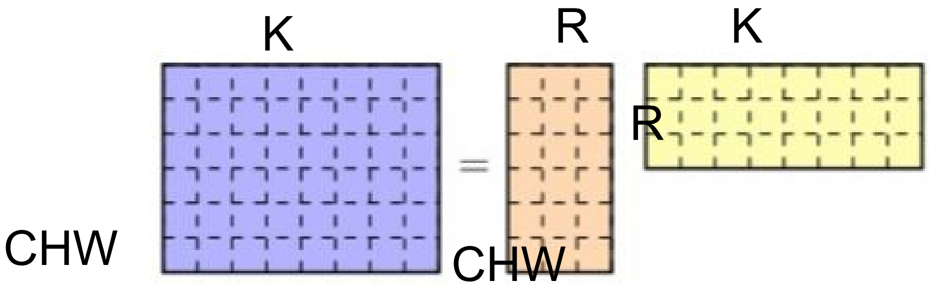
(a) (b) (c)

1: Comparison between approximations by outer product and Kronecker product for an image. The column (a) is the origin image of size 480×320, selected from BSD500 dataset Arbelaez et al. (2011). The column (b) is the SVD approximations of (a) by outer product and the column (c) is the approximation based on Kronecker productVan Loan and Pitsianis (1993), with rank $1, 2, 5, 10$ respectively from top to down. The shape of the right matrix in the Kronecker product is deliberately selected as 20×16 to make the number of parameters equal for each rank.

Matrix factorization => Convolution factorization

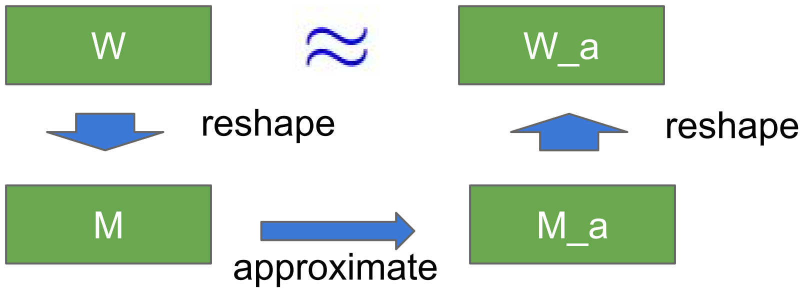
- Factorization into HxW followed by 1x1
    - feature map (N H’ W’, C H W)
    - first conv weights (C H W, R)
    - feature map (N H’ W’, R)
    - second conv weights (R, K)
    - feature map (N H’ W’, K)





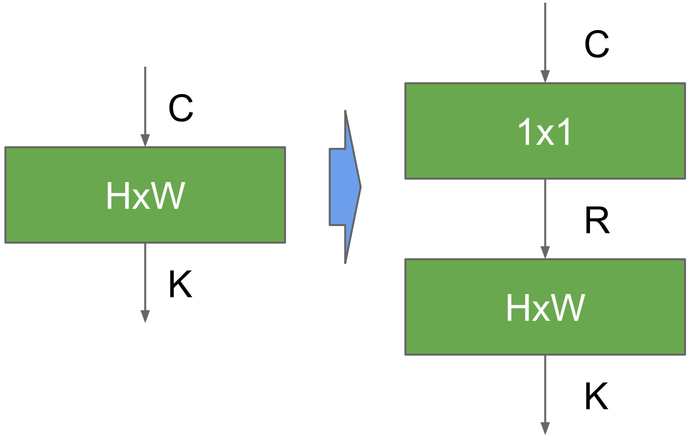
Approximating Convolution Weight

- W is a (K, C, H, W) 4-tensor
  - can be reshaped to a (CHW, K) matrix, etc.
- F-norm is invariant under reshape
  - $\|M - M_a\|_F = \|\text{reshape}(W) - \text{reshape}(W_a)\|_F = \|W - W_a\|_F$



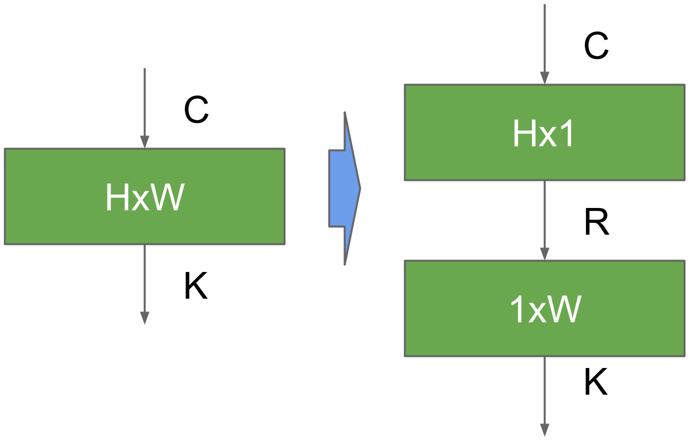
Matrix factorization => Convolution factorization

- Factorization into 1x1 followed by HxW
  - feature map (N H’ W’ H W, C)
  - first conv weights (C, R)
  - feature map (N H’ W’ H W, R) = (N H’ W’, R H W)
  - second conv weights (R H W, K)
  - feature map (N H’ W’, K)
- Steps
  - Reshape (CHW, K) to (C, HW, K)
  - (C, HW, K) = (C, R) (R, HW, K)
  - Reshape (R, HW, K) to (RHW, K)



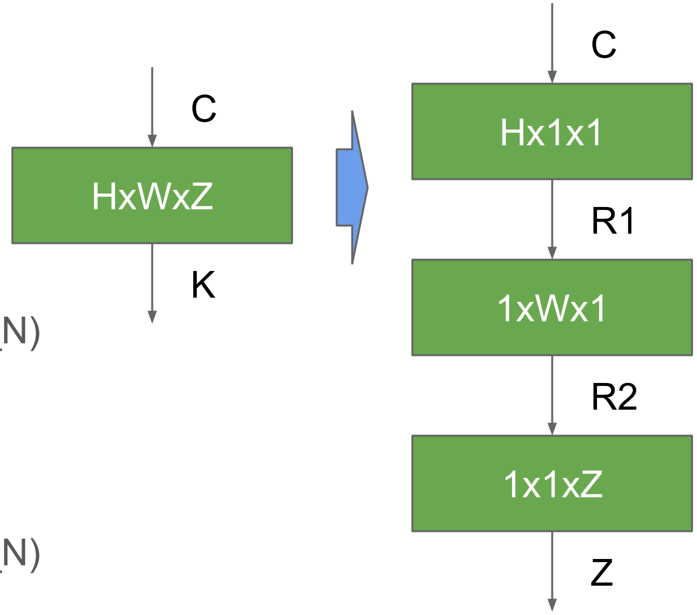
Horizontal-Vertical Decomposition

- Approximating with Separable Filters
- Original Convolution
  - feature map (N H’ W’, C H W)
  - weights (C H W, K)
- Factorization into Hx1 followed by 1xW
  - feature map (N H’ W’ W, C H)
  - first conv weights (C H, R)
  - feature map (N H’ W’ W, R) = (N H’ W’, R W)
  - second conv weights (R W, K)
  - feature map (N H’ W’, K)
- Steps
  - Reshape (CHW, K) to (CH, WK)
  - (CH, WK) = (CH, R) (R, WK)
  - Reshape (R, WK) to (RW, K)

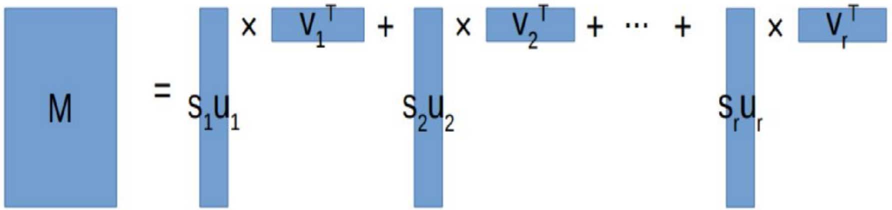


Factorizing N-D convolution

- Original Convolution
  - let dimension be N
  - feature map (N D'_1 D'_2 … D'_Z, C D_1 D_2 … D_N)
  - weights (C D_1 D_2 … D_N, K)
- Factorization into N number of D_i x1
  - R_0 = C, R_Z = K
  - feature map (N D'_1 D'_2 … D'_Z, C D_1 D_2 … D_N)
  - weights (R_0 D_1, R_1)
  - feature map (N D'_1 D'_2 … D'_Z, R_1 D_2 … D_N)
  - weights (R_1 D_2, R_2)
  - ...

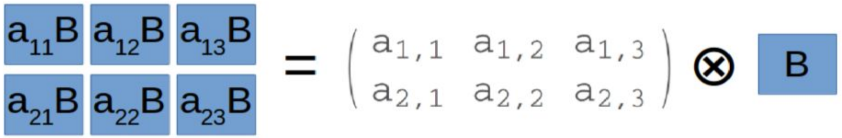


SVD



+

Kronecker Product



- KPSVD:

$$\mathbf{A} \approx \mathbf{A}_r = \sum_{i=1}^{r} \sigma_i \mathbf{U}_i \otimes \mathbf{V}_i.$$

Kronecker Conv

- (C H W, K)
- Reshape as (C_1 C_2 H W, K_1 K_2)
- Steps
  - Feature map is (N C H' W')
  - Extract patches and reshape (N H' W' C_2, C_1 H)
  - apply (C_1 H, K_1 R)
  - Feature map is (N K_1 R H' W' C_2)
  - Extract patches and reshape (N K_1 H' W', R C_2 W)
  - apply (R C_2 W, K_2)
- For rank efficiency, should have
  - R C_2 \approx C_1

Exploiting Local Structures with the Kronecker Layer in Convolutional Networks 1512

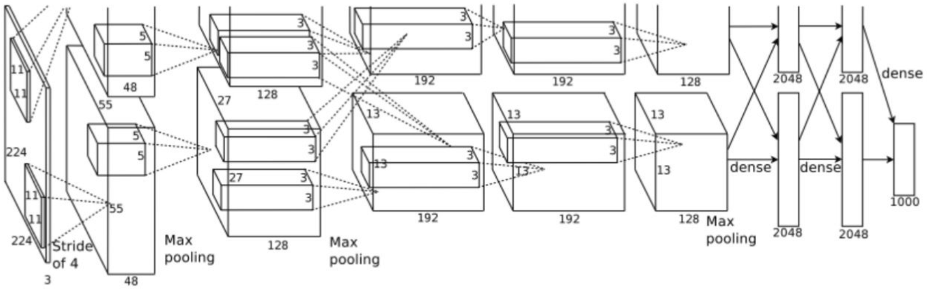
Methods	Configuration (r,o_1,c_1,h_1,w_1)	Validation Error	Speedup
Baseline	/	7.84%	/
KConv-a	1,128,24,9,1; 1,256,64,8,1	8.76%	3.3×
KConv-b	1,128,48,1,9; 1,512,64,1,8	8.69%	3.0×
KConv-c	2,64,24,9,1; 2,256,64,8,1	7.87%	2.9×

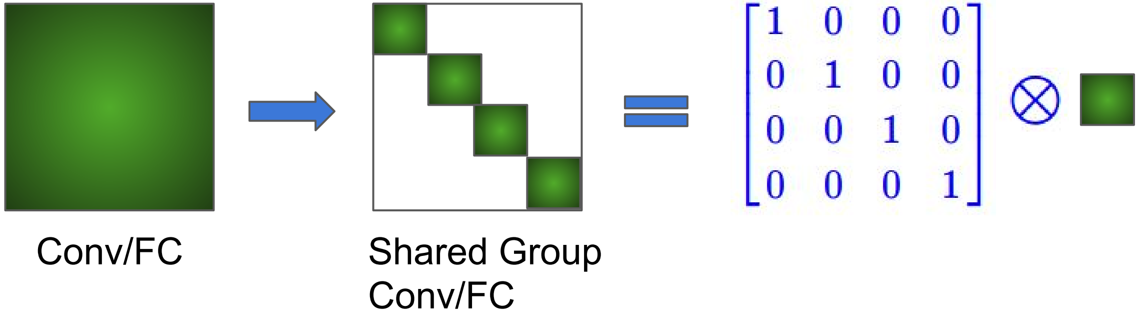
We have also experimented replacing the first convolutional layer with KConv layer. In this case, KConv with $(r, o_1, c_1, h_1, w_1) = (2, 12, 1, 1, 9)$, is found to outperform Jaderberg-style rank-1 filter with $(r, o_1, c_1, h_1, w_1) = (2, 96, 1, 1, 9)$ by 0.83%.

Face++ 旷视

Shared Group Convolution is a Kronecker Layer

AlexNet partitioned a conv





$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \otimes$$

CP-decomposition and Xception

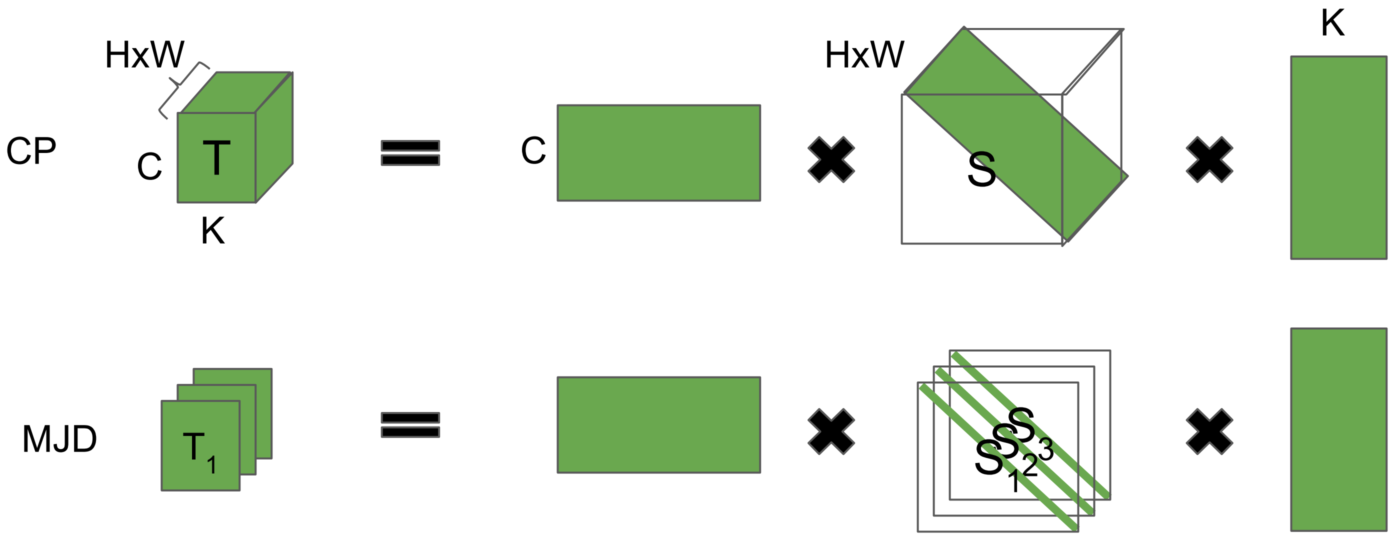
- Xception: Deep Learning with Depthwise Separable Convolutions 1610
- CP-decomposition with Tensor Power Method forConvolutional Neural Networks Compression 1701
- MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications 1704
  - They submitted the paper to CVPR about the same time as Xception.

Layer/Modification	Million Mult-Adds	Million Parameters
Convolution	462	2.36
Depthwise Separable Conv	52.3	0.27
$\alpha = 0.75$	29.6	0.15
$\rho = 0.714$	15.1	0.15

Table 5. Narrow vs Shallow MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
0.75 MobileNet	68.4%	325	2.6
Shallow MobileNet	65.3%	307	2.9

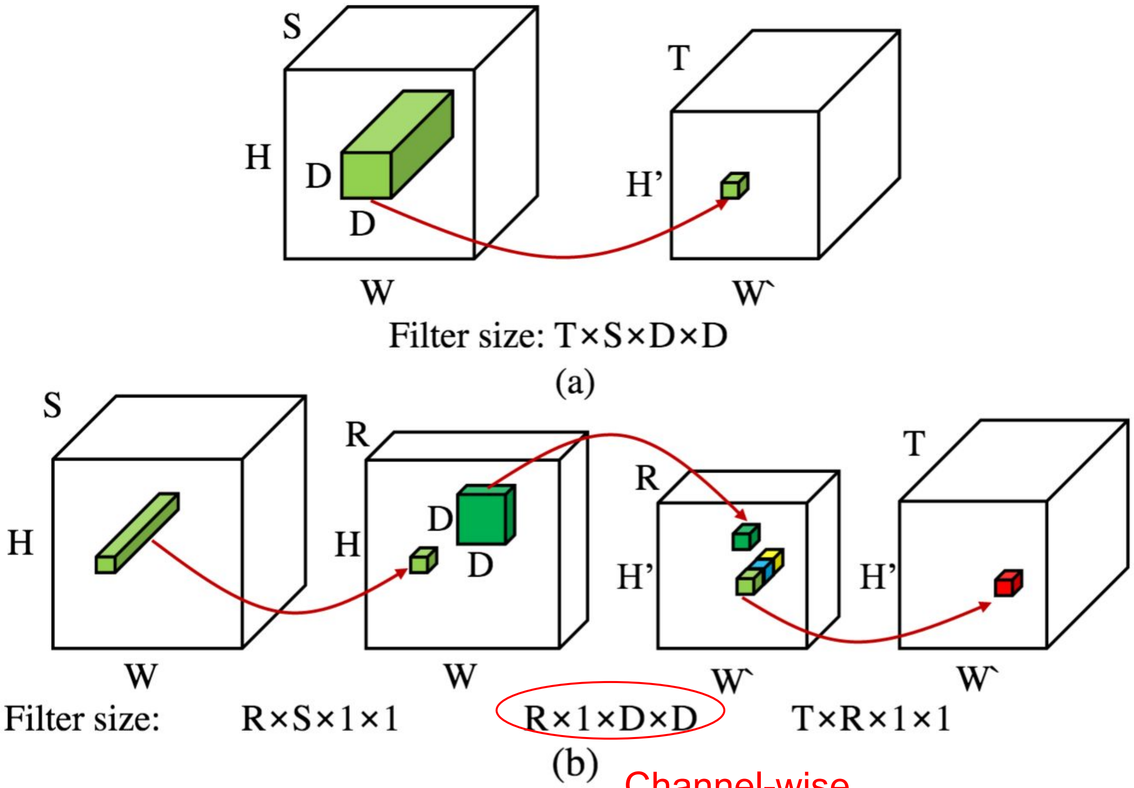
Matrix Joint Diagonalization = CP



CP-decomposition with Tensor Power Method for Convolutional Neural Networks Compression 1701

Convolution

Xception



Channel-wise

Tensor Train Decomposition

Oseledets, 2009:

TT-format

$A(i_1, i_2, \ldots, i_d) \approx$
$\sum_{\alpha_1,\alpha_2,\ldots,\alpha_{d-1}} G_1(i_1, \alpha_1) G_2(\alpha_1, i_2, \alpha_2) \ldots G_d(\alpha_{d-1}, i_d)$

$$f = x_1 + x_2 + \ldots + x_d$$

Canonical rank: d

TT-rank: 2

$$f = \begin{pmatrix} x_1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ x_2 & 1 \end{pmatrix} \cdot \ldots \cdot \begin{pmatrix} 1 & 0 \\ x_{d-1} & 1 \end{pmatrix} \begin{pmatrix} 1 \\ x_d \end{pmatrix}$$

$$f(x_1, \ldots, x_d) = \sin(x_1 + x_2 + \ldots + x_d)$$

FTT-decomposition has form

$$f = \begin{pmatrix} \sin x_1 & \cos x_1 \end{pmatrix} \begin{pmatrix} \cos x_2 & -\sin x_2 \\ \sin x_2 & \cos x_2 \end{pmatrix} \cdot \ldots \cdot$$
$$\begin{pmatrix} \cos x_{d-1} & -\sin x_{d-1} \\ \sin x_{d-1} & \cos x_{d-1} \end{pmatrix} \begin{pmatrix} \cos x_d \\ \sin x_d \end{pmatrix}.$$

Tensor Train Decomposition: just a few SVD's

Require: d-dimensional tensor $\mathbf{A}$ required accuracy ε

Ensure: Cores $G_1, \ldots, G_d$ of the TT-approximation $\mathbf{B}$ to $\mathbf{A}$ in the TT-format with smallest possibles compression ranks $\widehat{r}_k$ such that

$$||\mathbf{A} - \mathbf{B}||_F \leq \varepsilon ||\mathbf{A}||_F,$$

{Initialization}

Compute truncation parameter $\delta = \frac{\varepsilon}{\sqrt{d-1}} ||\mathbf{A}||_F$.

Temporary tensor: $\mathbf{C} = \mathbf{A}$.

$N = \mathrm{numel}(A)$, $r_0 = 1$.

for $k = 1$ **to** $d - 1$ **do**

$\quad C := \mathrm{reshape}(C, [r_{k-1} n_k, \frac{N}{r_{k-1} n_k}])$.

$\quad$ Compute δ-truncated SVD: $C = USV + E, ||E||_F \leq \delta, \quad r_k = \mathrm{rank}_\delta(C)$.

$\quad$ New core: $G_k := \mathrm{reshape}(U, [r_{k-1}, n_k, r_k])$.

$\quad C := SV^\top$.

$\quad N := \frac{N r_k}{n_k r_{k-1}}$.

end for

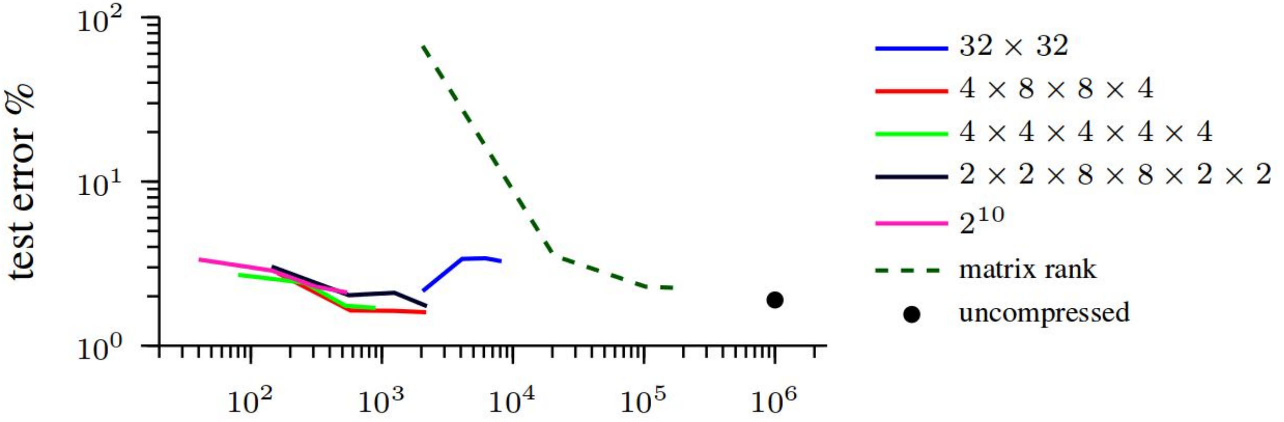
$G_d = C$.

Return tensor $\mathbf{B}$ in TT-format with cores $G_1, \ldots, G_d$.

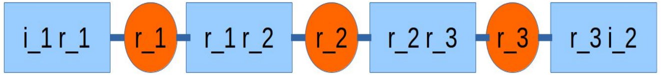
Tensor Train Decomposition on FC

Type	1 im. time (ms)	100 im. time (ms)
CPU fully-connected layer	16.1	97.2
CPU TT-layer	1.2	94.7
GPU fully-connected layer	2.7	33
GPU TT-layer	1.9	12.9

Table 3: Inference time for a 25088×4096 fully-connected layer and its corresponding TT-layer with all the TT-ranks equal 4. The memory usage for feeding forward one image is 392MB for the fully-connected layer and 0.766MB for the TT-layer.

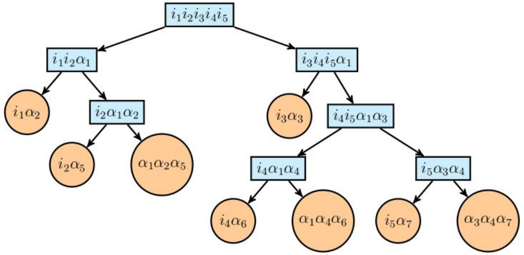


Graph Summary of SVD variants

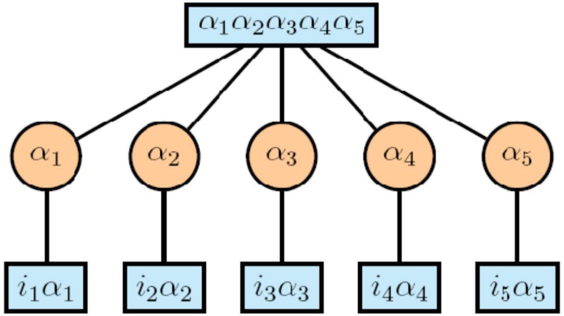


Matrix Product State

EXTENDED TT DECOMPOSITION

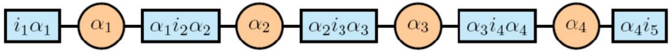


NUMBER OF EXTENDED TT PARAMETERS $= dnr + (d-2)r^3$



$$a(i_1, \ldots, i_d) = \sum_{\alpha_1=1}^{r_1} \ldots \sum_{\alpha_d=1}^{r_d} g(\alpha_1, \ldots, \alpha_d)\ q_1(i_1, \alpha_1) \ldots\ q_d(i_d, \alpha_d)$$

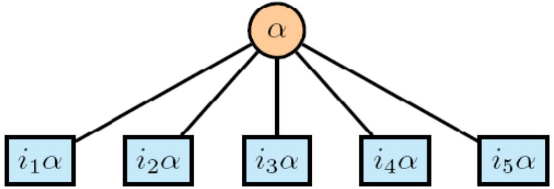
TUCKER DECOMPOSITION



$$a(i_1, \ldots, i_d) = \sum_{\alpha_1, \ldots, \alpha_{d-1}} g_1(i_1, \alpha_1)\, g_2(\alpha_1, i_2, \alpha_2)\ \ldots$$

$$\ldots\ g_{d-1}(\alpha_{d-2}, i_{d-1}, \alpha_{d-1})\, g_d(\alpha_{d-1}, i_d)$$

TENSOR-TRAIN DECOMPOSITION



$$a(i_1, \ldots, i_d) = \sum_{\alpha=1}^{R} u_1(i_1, \alpha) \ldots\ u_d(i_d, \alpha)$$

CANONICAL DECOMPOSITION (PARAFAC, CANDECOMP)

CNN layers as Multilinear Maps

Convolution: $A_{k,c}X_{n,c,h,w,h_k,w_k}$

FC: $A_{k,c}X_{n,c,h,w,1,1}$

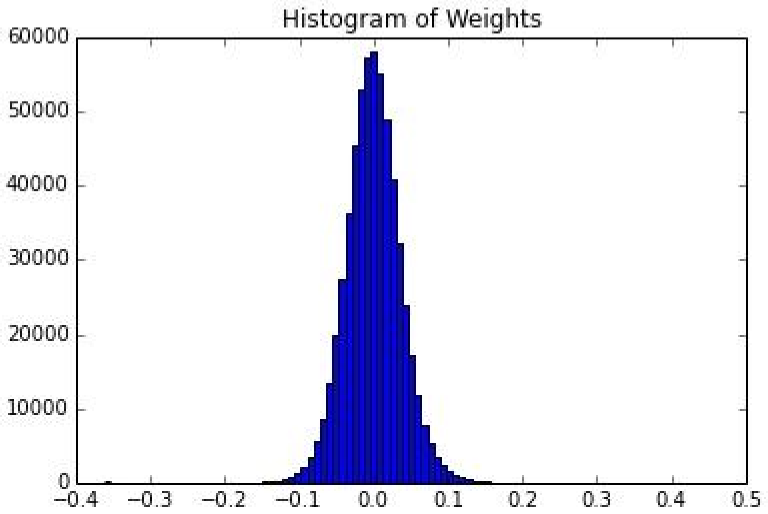
Xception: $A_{k,c}\delta_{k,c}X_{n,c,h,w,h_k,w_k}$

Kronecker: $A_{k_2,c_2}X_{n,c_1,c_2,h,w,h_k,w_k}$

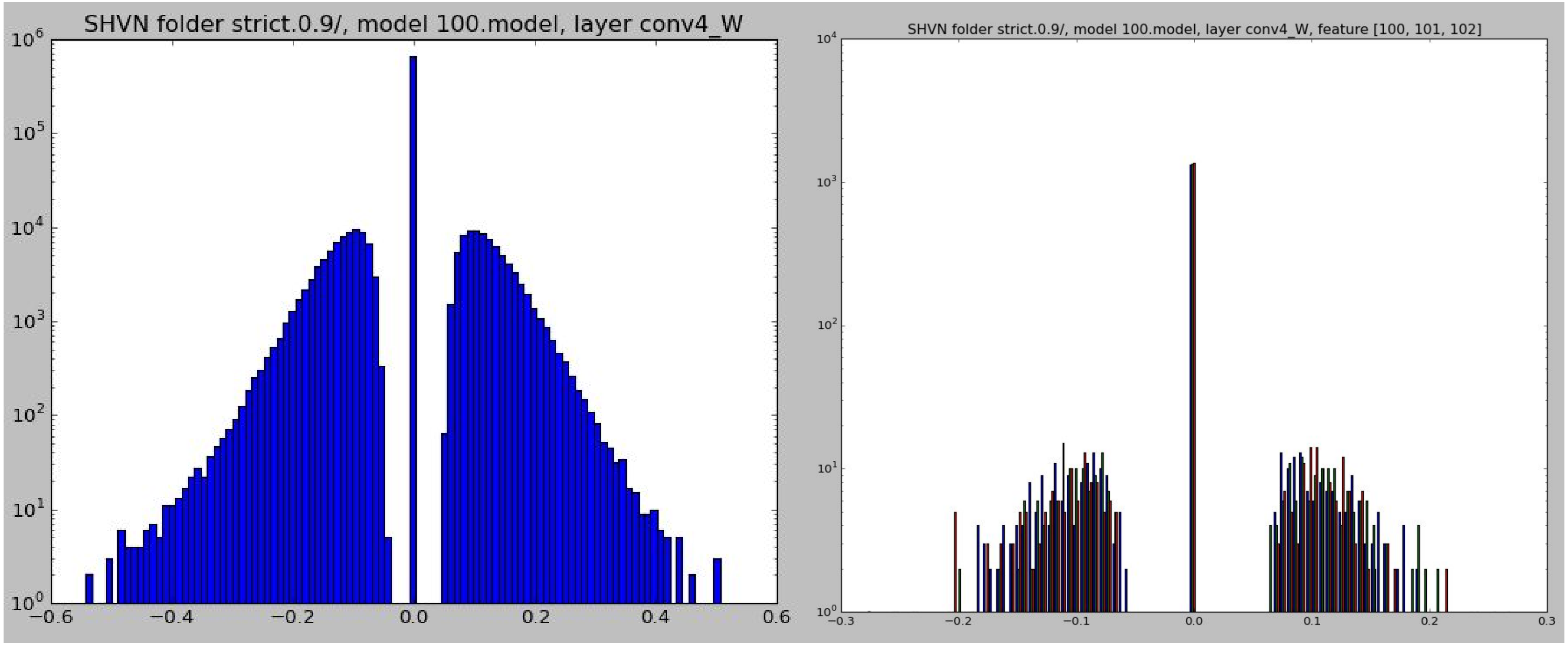
Sparse Approximation

Distribution of Weights

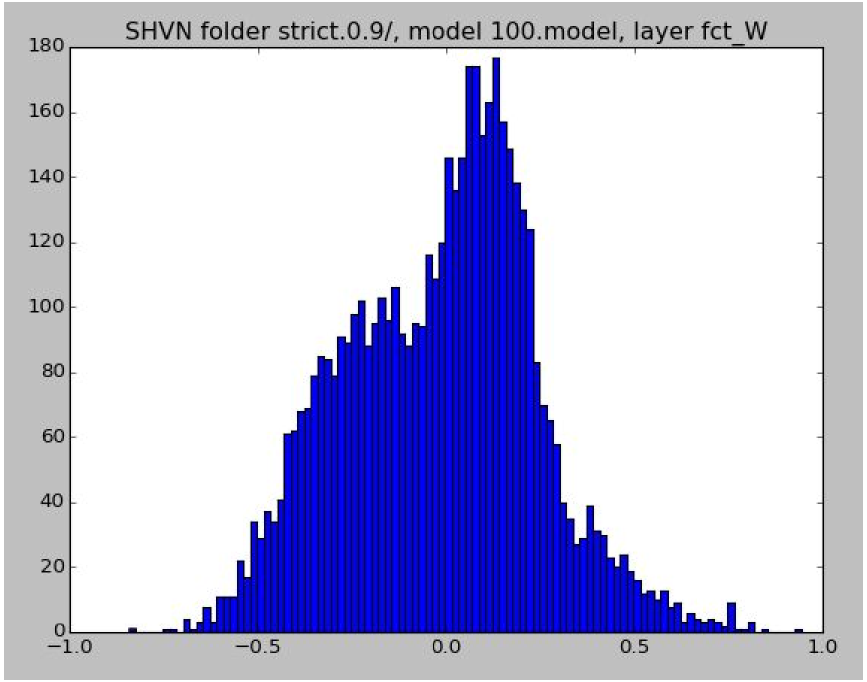
- Universal across convolutions and FC
- Concentration of values near 0
- Large values ***cannot*** be dropped

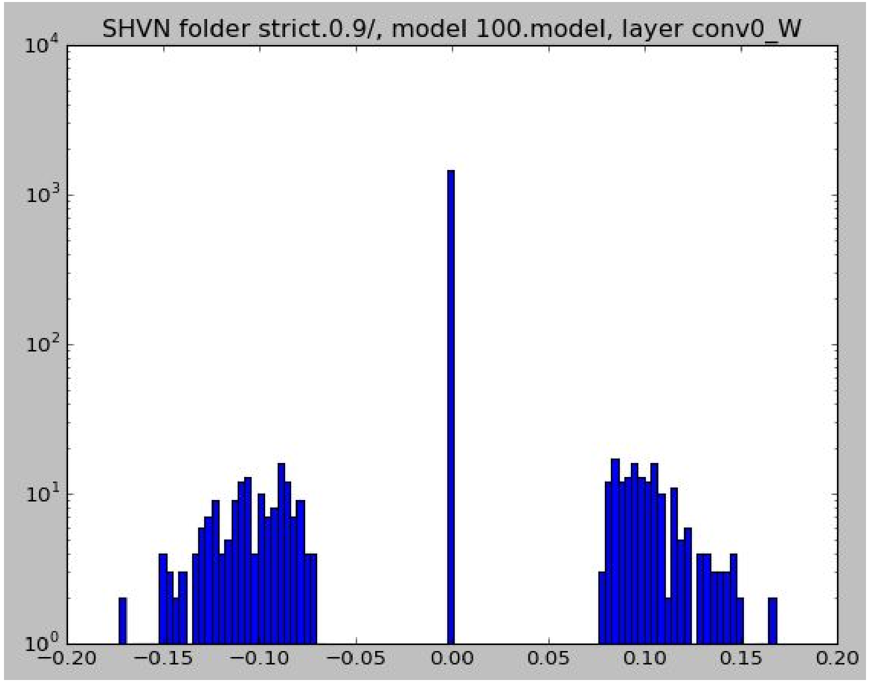


Sparsity of NN: statistics



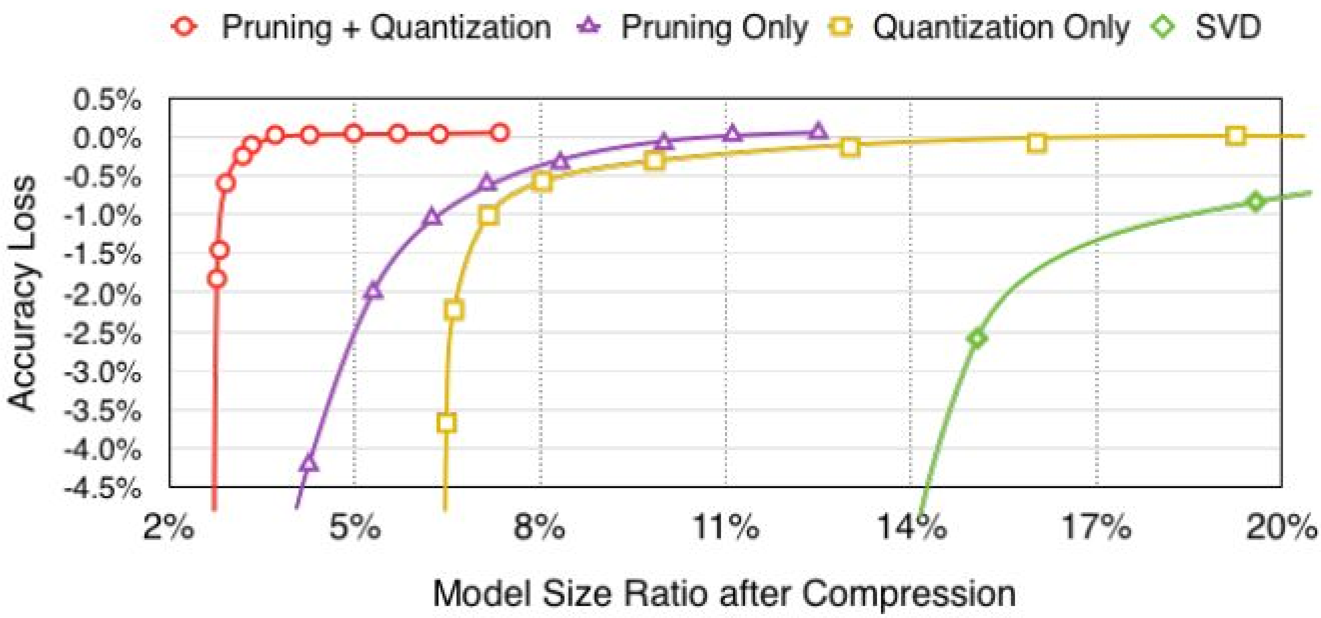
Sparsity of NN: statistics





Weight Pruning: from DeepCompression

Network → Train → Extract Mask M → W => M o W → Train → ...



The model has been trained with exccessive #epoch.

Sparse Matrix at Runtime

- Sparse Matrix = Discrete Mask + Continuous values
  - Mask cannot be learnt the normal way
  - The values have well-defined gradients
- The matrix value look up need go through a LUT
  - CSR format
    - A: NNZ values
    - IA: accumulated #NNZ of rows
    - JA: the column in the row

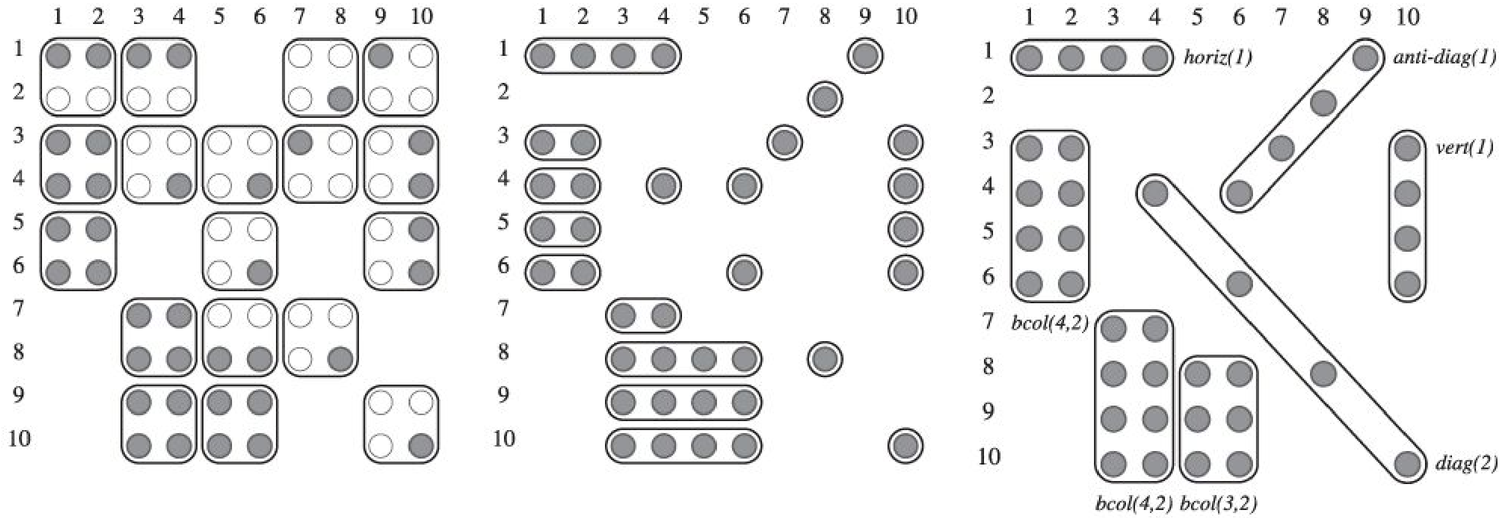
$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 5 & 8 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 6 & 0 & 0 \end{pmatrix}$$

```
A  = [ 5 8 3 6 ]
IA = [ 0 0 2 3 4 ]
JA = [ 0 1 2 1 ]
```

Face++ 旷视

Burden of Sparseness

- ***Lost of regularity of memory access and computation***
    - Need special hardware for efficient access
    - May need high zero ratio to match dense matrix
        - Matrices will less than 70% zero values, better to treat as dense matrices.



(a) BCSR. (b) VBL. (c) CSX.

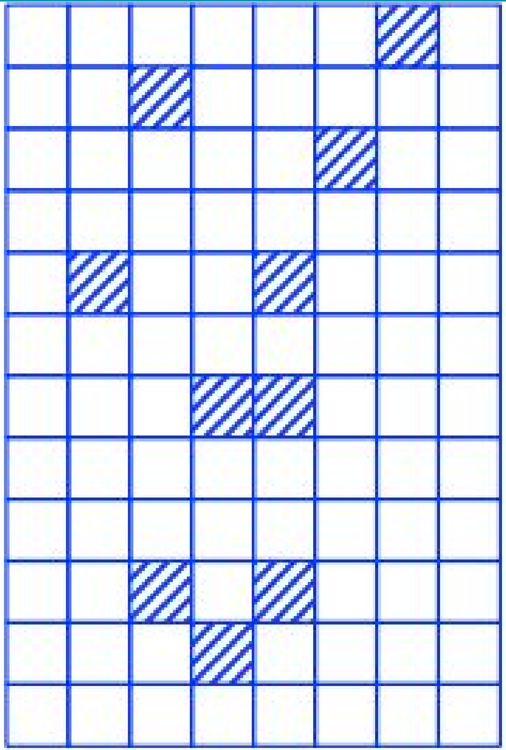
Convolution layers are harder to compress than FC

Table 5: Compression statistics for VGG-16. P: pruning, Q:quantization, H:Huffman coding.

Layer	#Weights	Weights% (P)	Weigh bits (P+Q)	Weight bits (P+Q+H)	Index bits (P+Q)	Index bits (P+Q+H)	Compress rate (P+Q)	Compress rate (P+Q+H)
conv1_1	2K	58%	8	6.8	5	1.7	40.0%	29.97%
conv1_2	37K	22%	8	6.5	5	2.6	9.8%	6.99%
conv2_1	74K	34%	8	5.6	5	2.4	14.3%	8.91%
conv2_2	148K	36%	8	5.9	5	2.3	14.7%	9.31%
conv3_1	295K	53%	8	4.8	5	1.8	21.7%	11.15%
conv3_2	590K	24%	8	4.6	5	2.9	9.7%	5.67%
conv3_3	590K	42%	8	4.6	5	2.2	17.0%	8.96%
conv4_1	1M	32%	8	4.6	5	2.6	13.1%	7.29%
conv4_2	2M	27%	8	4.2	5	2.9	10.9%	5.93%
conv4_3	2M	34%	8	4.4	5	2.5	14.0%	7.47%
conv5_1	2M	35%	8	4.7	5	2.5	14.3%	8.00%
conv5_2	2M	29%	8	4.6	5	2.7	11.7%	6.52%
conv5_3	2M	36%	8	4.6	5	2.3	14.8%	7.79%
fc6	103M	4%	5	3.6	5	3.5	1.6%	1.10%
fc7	17M	4%	5	4	5	4.3	1.5%	1.25%
fc8	4M	23%	5	4	5	3.4	7.1%	5.24%
Total	138M	7.5%(13×)	6.4	4.1	5	3.1	3.2% (31×)	2.05% (49×)

Dynamic Generation of Code

- CVPR'15: Sparse Convolutional Neural Networks
- Relies on compiler for
  - register allocation
  - scheduling
- Good on CPU



Input:

$\mathbf{A}$: 8×12 dense matrix

$\mathbf{B}$: 12×8 sparse matrix

Output:

$\mathbf{C} = \mathbf{A} \times \mathbf{B}$

Operations:

$$\mathbf{c}_7 += \mathbf{a}_1 \times b_{1,7}$$
$$\mathbf{c}_3 += \mathbf{a}_2 \times b_{2,3}$$
$$\mathbf{c}_6 += \mathbf{a}_3 \times b_{3,6}$$
$$\mathbf{c}_2 += \mathbf{a}_5 \times b_{5,2}$$
$$\mathbf{c}_5 += \mathbf{a}_5 \times b_{5,5}$$
$$\mathbf{c}_4 += \mathbf{a}_7 \times b_{7,4}$$
$$\mathbf{c}_5 += \mathbf{a}_7 \times b_{7,5}$$
$$\mathbf{c}_3 += \mathbf{a}_{10} \times b_{10,3}$$
$$\mathbf{c}_5 += \mathbf{a}_{10} \times b_{10,5}$$
$$\mathbf{c}_4 += \mathbf{a}_{11} \times b_{11,4}$$

Channel Pruning

- Learning the Number of Neurons in Deep Networks [1611](#)

$$\min_{\Theta} \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, f(\mathbf{x}_i, \Theta)) + r(\Theta)$$

- Channel Pruning for Accelerating Very Deep Neural Networks [1707](#)
  - Also exploits low-rankness of features

$$\underset{\beta,\mathrm{W}}{\arg\min} \frac{1}{2N} \left\| \mathrm{Y}' - \sum_{i=1}^{c} \beta_i \mathrm{X_i} \mathrm{W_i}^{\top} \right\|_F^2$$

$$\text{subject to } \|\beta\|_0 \leq c'$$

Face++ 旷视

Sparse Communication for Distributed Gradient Descent 1704

Algorithm 1 Gradient dropping algorithm given gradient ∇ and dropping rate R.

```
function GRADDROP(∇, R)
    ∇+ = residuals
    Select threshold: R% of |∇| is smaller
    dropped ← 0
    dropped[i] ← ∇[i]∀i : |∇[i]| > threshold
    residuals ← ∇ − dropped
    return sparse(dropped)
end function
```

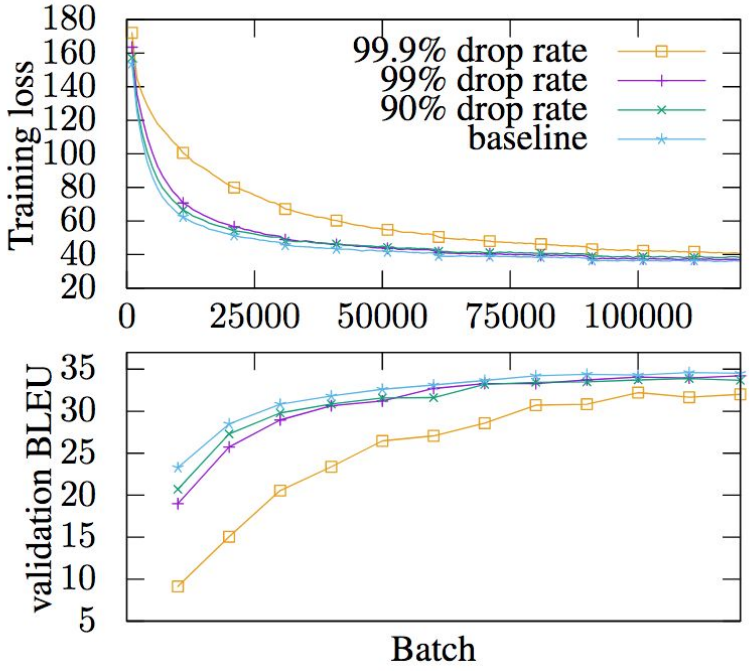


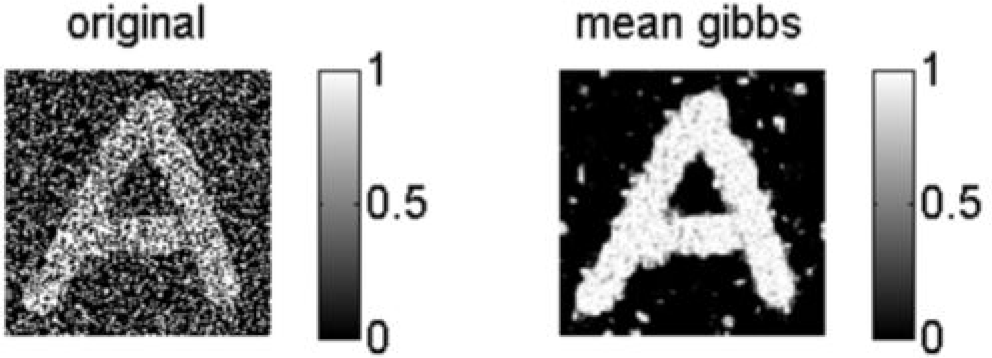
Figure 3: NMT: Training loss and validation BLEU for different dropping ratios.

Quantization

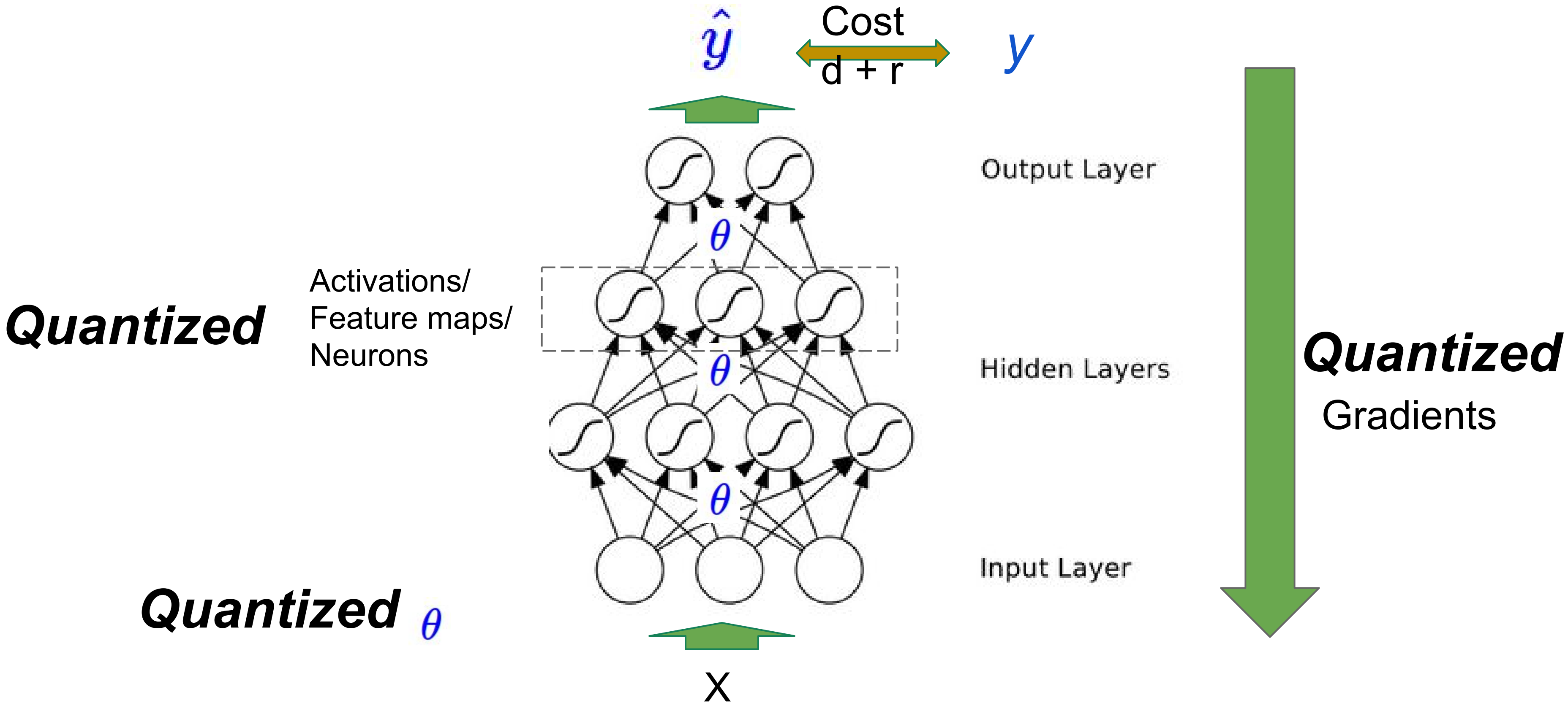
Precursor: Ising model & Boltzmann machine

- Ising model
  - used to model magnetics
  - 1D has trivial analytic solution
  - 2D exhibits phase-transition
  - 2D Ising model can be used for denoising
    - when the mean signal is reliable
- Inference also requires optimization

$$H(\sigma) = -\sum_{<i\ j>} J_{ij}\sigma_i\sigma_j - \mu\sum_j h_j\sigma_j$$

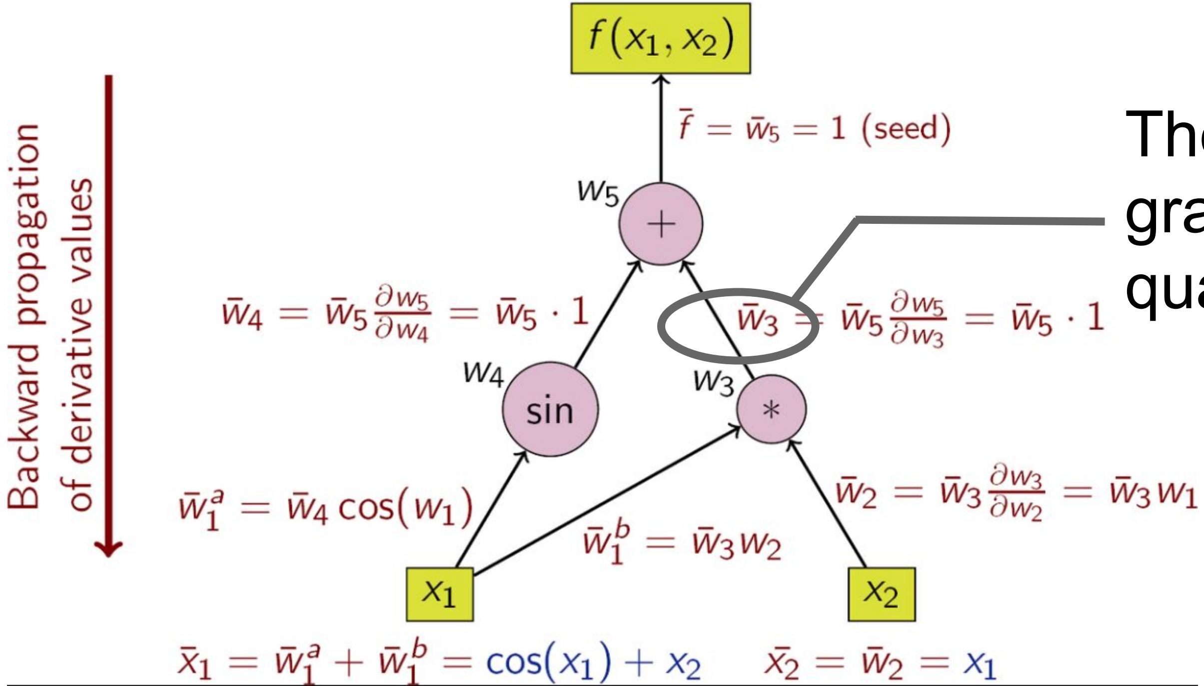


Neural Network Training



Backpropagation

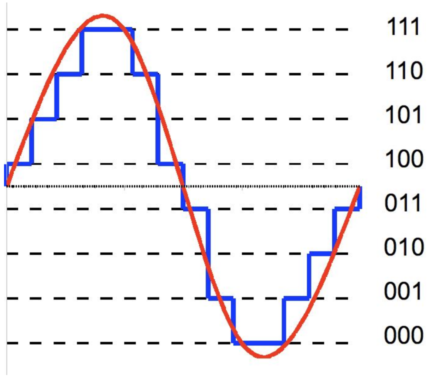
$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial w_1}\frac{\partial w_1}{\partial x} = \left(\frac{\partial y}{\partial w_2}\frac{\partial w_2}{\partial w_1}\right)\frac{\partial w_1}{\partial x} = \left(\left(\frac{\partial y}{\partial w_3}\frac{\partial w_3}{\partial w_2}\right)\frac{\partial w_2}{\partial w_1}\right)\frac{\partial w_1}{\partial x} = \cdots$$



There will be no gradient flow if we quantize somewhere!

Differentiable Quantization

- Bengio '13: Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation
  - REINFORCE algorithm
  - Decompose binary stochastic neuron into stochastic and differentiable part
  - Injection of additive/multiplicative noise
  - Straight-through estimator



Gradient vanishes after quantization.

Quantization also at Train time

- Neural Network can adapt to the constraints imposed by quantization
- Exploits “Straight-through estimator” (Hinton, Coursera lecture, 2012)

$$x \approx \hat{x}$$

$$\Rightarrow$$

$$\frac{\partial}{\partial x} \approx \frac{\partial}{\partial \hat{x}}$$

- Example

$$\textbf{Forward: } q \sim Bernoulli(p) \qquad q \approx \mathrm{E}[q] = p$$

$$\textbf{Backward: } \frac{\partial c}{\partial p} = \frac{\partial c}{\partial q}.$$

Bit Neural Network

- Matthieu Courbariaux et al. BinaryConnect: Training Deep Neural Networks with binary weights during propagations. http://arxiv.org/abs/1511.00363
- Itay Hubara et al. Binarized Neural Networks https://arxiv.org/abs/1602.02505v3
- Matthieu Courbariaux et al. Binarized Neural Networks: Training Neural Networks with Weights and Activations Constrained to +1 or -1. http://arxiv.org/pdf/1602.02830v3.pdf
- Rastegari et al. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks http://arxiv.org/pdf/1603.05279v1.pdf
- Zhou et al. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients https://arxiv.org/abs/1606.06160
- Hubara et al. Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations https://arxiv.org/abs/1609.07061

Binarizing AlexNet

Theoretical

	Network Variations	Operations used in Convolution	Memory Saving (Inference)	Time Saving on CPU (Inference)	Accuracy on ImageNet (AlexNet)
Standard Convolution	Real-Value Inputs 0.11 -0.21 ... -0.34 -0.25 0.61 ... 0.52 Real-Value Weights 0.12 -1.2 ... 0.41 -0.2 0.5 ... 0.68	+ , − , ×	1x	1x	%56.7
Binary Weight	Real-Value Inputs 0.11 -0.21 ... -0.34 -0.25 0.61 ... 0.52 Binary Weights 1 -1 ... 1 -1 1 ... 1	+ , −	~32x	~2x	%53.8
BinaryWeight Binary Input (**XNOR-Net**)	Binary Inputs 1 -1 ... -1 -1 1 ... 1 Binary Weights 1 -1 ... 1 -1 1 ... 1	XNOR , bitcount	~32x	~58x	%44.2

Scaled binarization

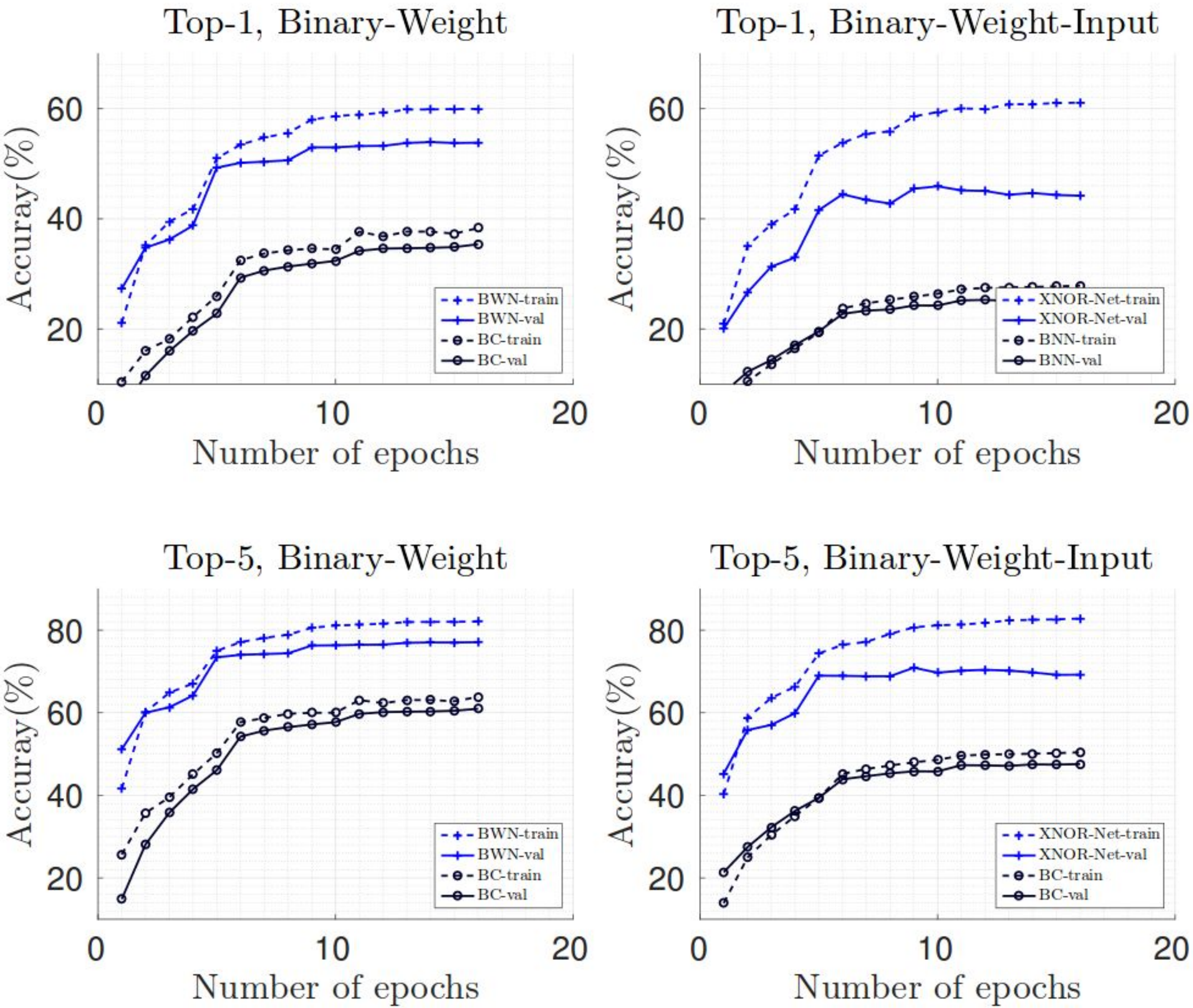
$$\min_{\Lambda \in \text{diagonal}, B \in \{1,-1\}^{m \times n}} ||\Lambda B - W||_F^2$$

- Sol:

$$||\Lambda B - W||_F^2 = ||(\Lambda B) \circ B - W \circ B||_F^2$$
$$= ||\Lambda (B \circ B) - W \circ B||_F^2$$
$$= ||\Lambda \mathbf{1} - W \circ B||_F^2$$

= Varaince of rows of W o B

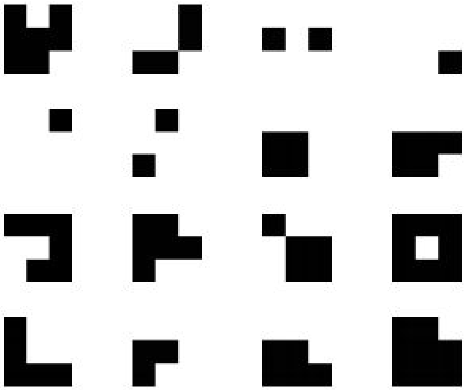
XNOR-Net


Top-1, Binary-Weight
Top-1, Binary-Weight-Input
Top-5, Binary-Weight
Top-5, Binary-Weight-Input
Accuray(%)
Number of epochs
BWN-train
BWN-val
BC-train
BC-val
XNOR-Net-train
XNOR-Net-val
BNN-train
BNN-val

Binary weights network

- Filter repetition
  - 3x3 binary kernel has only 256 patterns modulo sign.
  - 3x1 binary kernel only has only 4 patterns modulo sign.
  - Not easily exploitable as we are applying CHW as filter

Figure 2: Binary weight filters, sampled from of the first convolution layer. Since we have only 2^{k^2} unique 2D filters (where k is the filter size), filter replication is very common. For instance, on our CIFAR-10 ConvNet, only 42% of the filters are unique.



Binarizing AlexNet

Theoretical

	Network Variations	Operations used in Convolution	Memory Saving (Inference)	Time Saving on CPU (Inference)	Accuracy on ImageNet (AlexNet)
Standard Convolution	Real-Value Inputs 0.11 -0.21 ... -0.34 -0.25 0.61 ... 0.52 Real-Value Weights 0.12 -1.2 ... 0.41 -0.2 0.5 ... 0.68	+, −, ×	1x	1x	%56.7
Binary Weight	Real-Value Inputs 0.11 -0.21 ... -0.34 -0.25 0.61 ... 0.52 Binary Weights 1 -1 ... 1 -1 1 ... 1	+, −	~32x	~2x	%53.8
BinaryWeight Binary Input **(XNOR-Net)**	Binary Inputs 1 -1 ... -1 -1 1 ... 1 Binary Weights 1 -1 ... 1 -1 1 ... 1	XNOR, bitcount	~32x	~58x	%44.2

Scaled binarization is no longer exact and not found to be useful

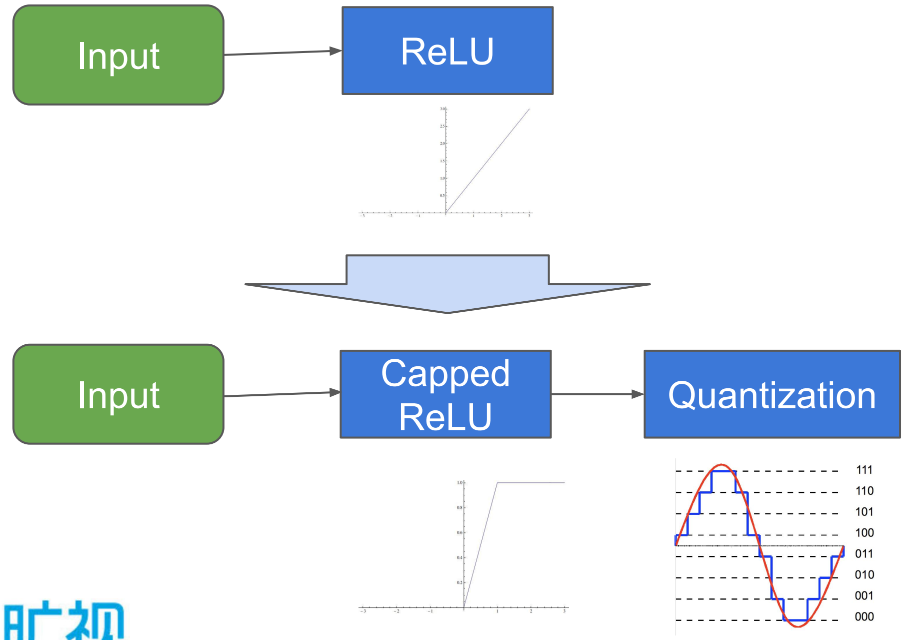
$$\alpha^*, \mathbf{B}^*, \beta^*, \mathbf{H}* = \underset{\alpha, \mathbf{B}, \beta, \mathbf{H}}{\operatorname{argmin}} \|\mathbf{X}^\intercal \mathbf{W} - \beta\alpha \mathbf{H}^\intercal \mathbf{B}\|$$

The solution below is quite bad, like when Y = [-4, 1]

$$\mathbf{C}^* = \operatorname{sign}(\mathbf{Y}) = \operatorname{sign}(\mathbf{X}^\intercal) \operatorname{sign}(\mathbf{W}) = \mathbf{H}^{*\intercal} \mathbf{B}^*$$

Quantization of Activations

- XNOR-net adopted STE method in their open-source our code



DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients

- Uniform stochastic quantization of gradients
  - 6 bit for ImageNet, 4 bit for SVHN
- Simplified scaled binarization: only scalar
  - Forward and backward multiplies the bit matrices from different sides.
  - Using scalar binarization allows using bit operations
- Floating-point-free inference even when with BN
- Future work
  - BN requires FP computation during training
  - Require FP weights for accumulating gradients

Face++ 旷视

Table 1: Comparison of prediction accuracy for SVHN with different choices of Bit-width in a DoReFa-Net. W, A, G are bitwidths of weights, activations and gradients respectively. When bitwidth is 32, we simply remove the quantization functions.

W	A	G	Training Complexity	Inference Complexity	Storage Relative Size	Model A Accuracy	Model B Accuracy	Model C Accuracy	Model D Accuracy
1	1	2	3	1	1	0.934	0.924	0.910	0.803
1	1	4	5	1	1	0.968	0.961	0.916	0.846
1	1	8	9	1	1	0.970	0.962	0.902	0.828
1	1	32	-	-	1	0.971	0.963	0.921	0.841
1	2	2	4	2	1	0.909	0.930	0.900	0.808
1	2	3	5	2	1	0.968	0.964	0.934	0.878
1	2	4	6	2	1	0.975	0.969	0.939	0.878
2	1	2	6	2	2	0.927	0.928	0.909	0.846
2	1	4	10	2	2	0.969	0.957	0.904	0.827
1	2	8	10	2	1	0.975	0.971	0.946	0.866
1	2	32	-	-	1	0.976	0.970	0.950	0.865
1	3	3	6	3	1	0.968	0.964	0.946	0.887
1	3	4	7	3	1	0.974	0.974	0.959	0.897
1	3	6	9	3	1	0.977	0.974	0.949	0.916
1	4	2	6	4	1	0.815	0.898	0.911	0.868
1	4	4	8	4	1	0.975	0.974	0.962	0.915
1	4	8	12	4	1	0.977	0.975	0.955	0.895
2	2	2	8	4	1	0.900	0.919	0.856	0.842
8	8	8	-	-	8			0.970	0.955
32	32	32	-	-	32	0.975	0.975	0.972	0.950

A has two times as many channels as B.
B has two times as many channels as C.
...

Quantization Methods

- Deterministic Quantization

$$Q_k^{det}(\mathbf{X}) = \alpha\, Q_k(\tilde{\mathbf{X}}) + \beta \approx \mathbf{X}$$

- Stochastic Quantizaiton

$$Q_k^{stoc}(\mathbf{X}) = \alpha\, Q_k(\tilde{\mathbf{X}} + \frac{\xi}{2^k - 1}) + \beta \approx \mathbf{X}$$

$$\xi \sim U(-\frac{1}{2}, \frac{1}{2})$$

Injection of noise realizes the sampling.

$$\tilde{\mathbf{X}} = \frac{\mathbf{X} - \beta}{\alpha}$$

$$\alpha = \max(\mathbf{X}) - \min(\mathbf{X})$$

$$\beta = \min(\mathbf{X})$$

Face++ 旷视

Quantization of Weights, Activations and Gradients

- A half #channel 2-bit AlexNet (same bit complexity as XNOR-net)

Quantization Method	Balanced Determini	Unbalanced Deterministic	Stochastic
Weights	**0.469**	0.346	0.120
Activations	0.315	**0.469**	diverge
Gradients	diverge	diverge	**0.469**
XNOR-net		0.442*	
BNN		0.279*	

Quantization Error measured by Cosine Similarity

- Wb_sn is n-bit quantizaiton of real W
- x is Gaussian R. V. clipped by tanh

Wb_s1	Wb_s2	Wb_s3	Wb_s4	Wb_s6	Wb_s8	W	
0.684	0.684	0.699	0.795	0.882	0.888	0.888	xb
0.742	0.742	0.758	0.862	0.957	0.964	0.964	x_2
0.763	0.763	0.780	0.887	0.985	0.991	0.992	x_3
0.768	0.768	0.785	0.893	0.991	0.998	0.998	x_4
0.769	0.770	0.786	0.894	0.993	0.999	1.000	x_6
0.769	0.770	0.786	0.895	0.993	1.000	1.000	x_8
0.769	0.770	0.786	0.895	0.993	1.000	1.000	x

Saturates

Face++ 旷视

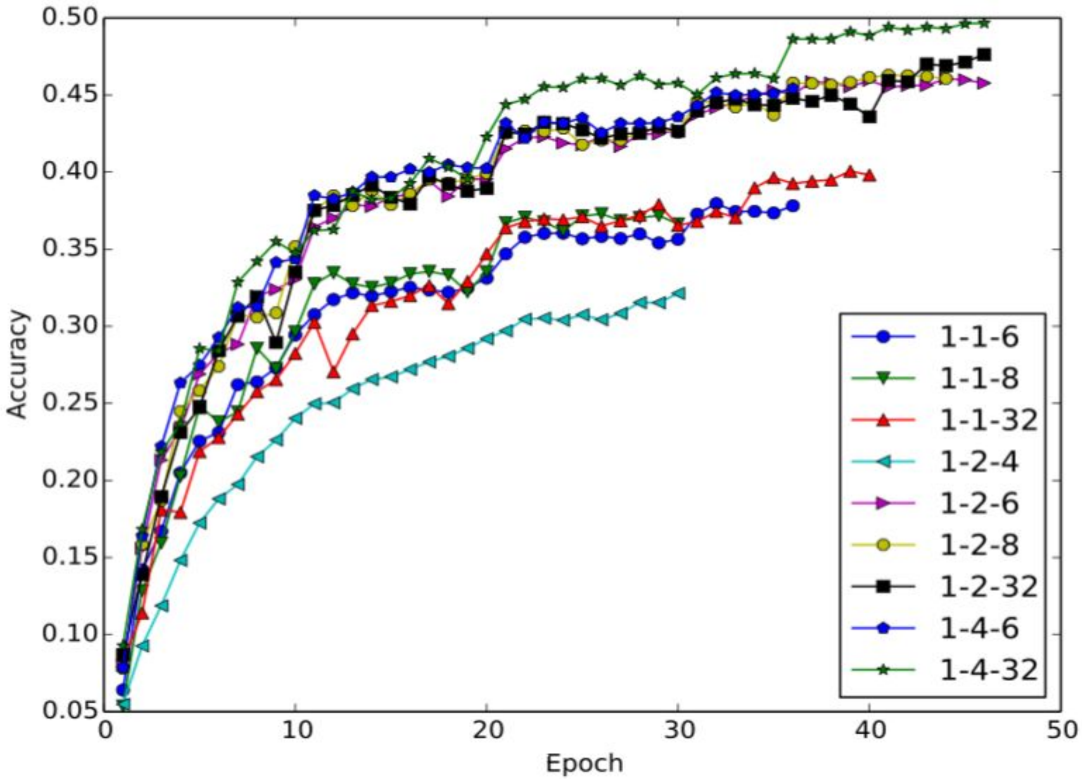
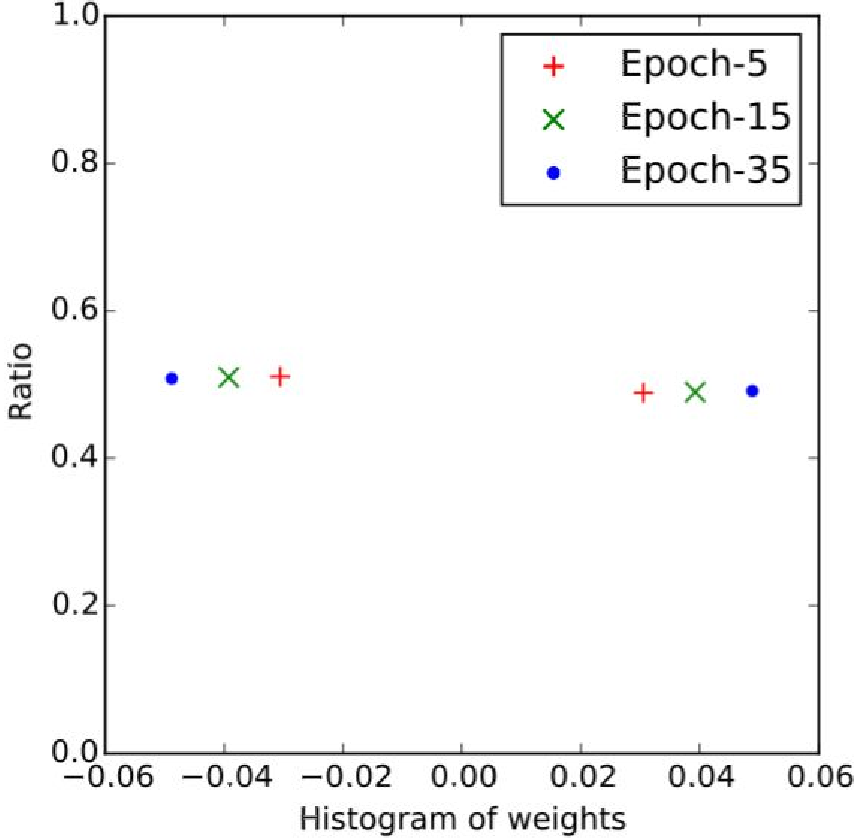
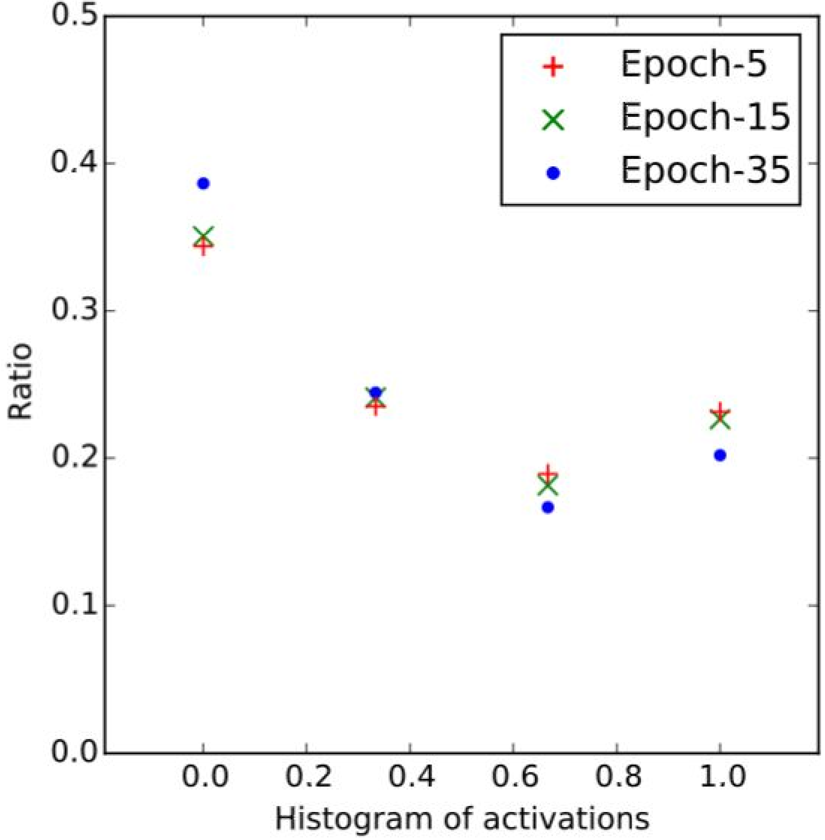


Figure 1: Prediction accuracy of AlexNet variants on Validation Set of ImageNet indexed by epoch number. “W-A-G” gives the specification of bitwidths of weights, activations and gradients. E.g., “1-2-4” stands for the case when weights are 1-bit, activations are 2-bit and gradients are 4-bit. The figure is best viewed in color.



(a)

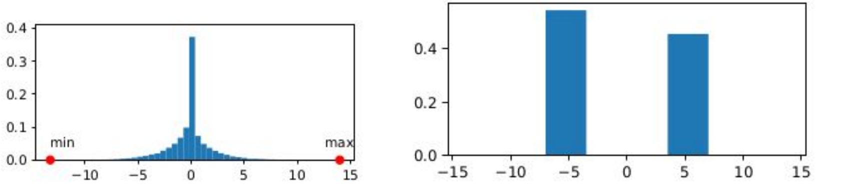


(b)

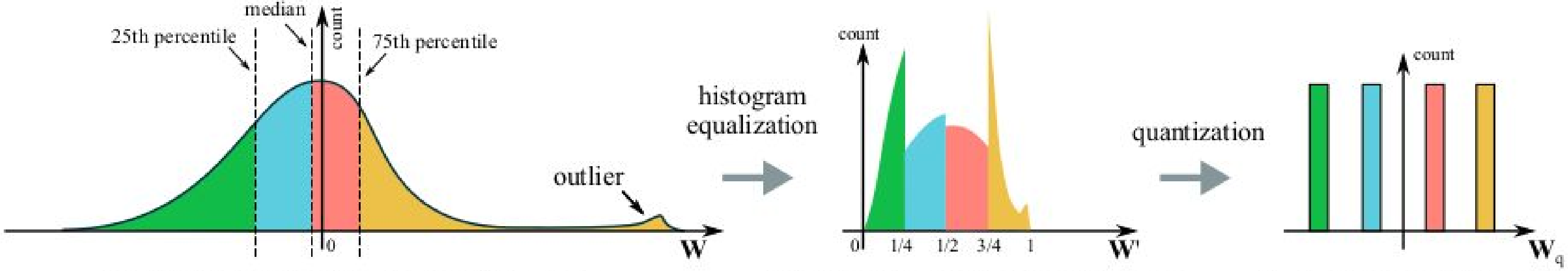
Figure 3: (a) is histogram of weights of layer “conv3” of “1-2-6” AlexNet model at epoch 5, 15

Effective Quantization Methods for Recurrent Neural Networks 2016

Model	weight-bits	activation-bits	PPW	
			balanced	unbalanced
LSTM	2	2	152	164
LSTM	2	3	142	155
LSTM (Hubara et al., 2016a)	2	3		220
LSTM (Hubara et al., 2016a)	4	4		100



(a) floating point copy of weights in QNN after 60 epochs (b) imbalanced quantization (no equalization) (c) balanced quantization with median (before matching value range) (d) balanced quantization with mean (before matching value range)



(a) historgram of floating-point weight values (b) histogram-equalized weight values (c) quantized weight values

Training Bit Fully Convolutional Network for Fast Semantic Segmentation 2016

bit-width (W / A)	mean IoU	Complexity
32 / 32	69.8%	-
8 / 8	69.8%	64
4 / 4	68.6%	16
3 / 3	67.4%	9
2 / 2	65.7%	4
1 / 4	64.4%	4
4 / 1	diverge	4
1 / 2	62.8%	2

Table 5: Results of different bit-width allocated to weight and activation on PASCAL VOC 2012 val set.

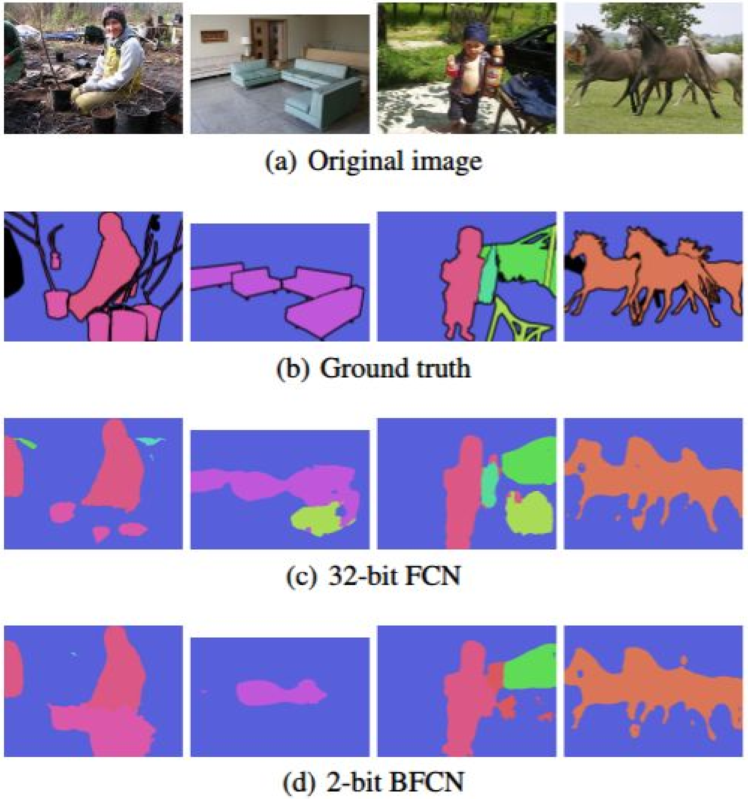
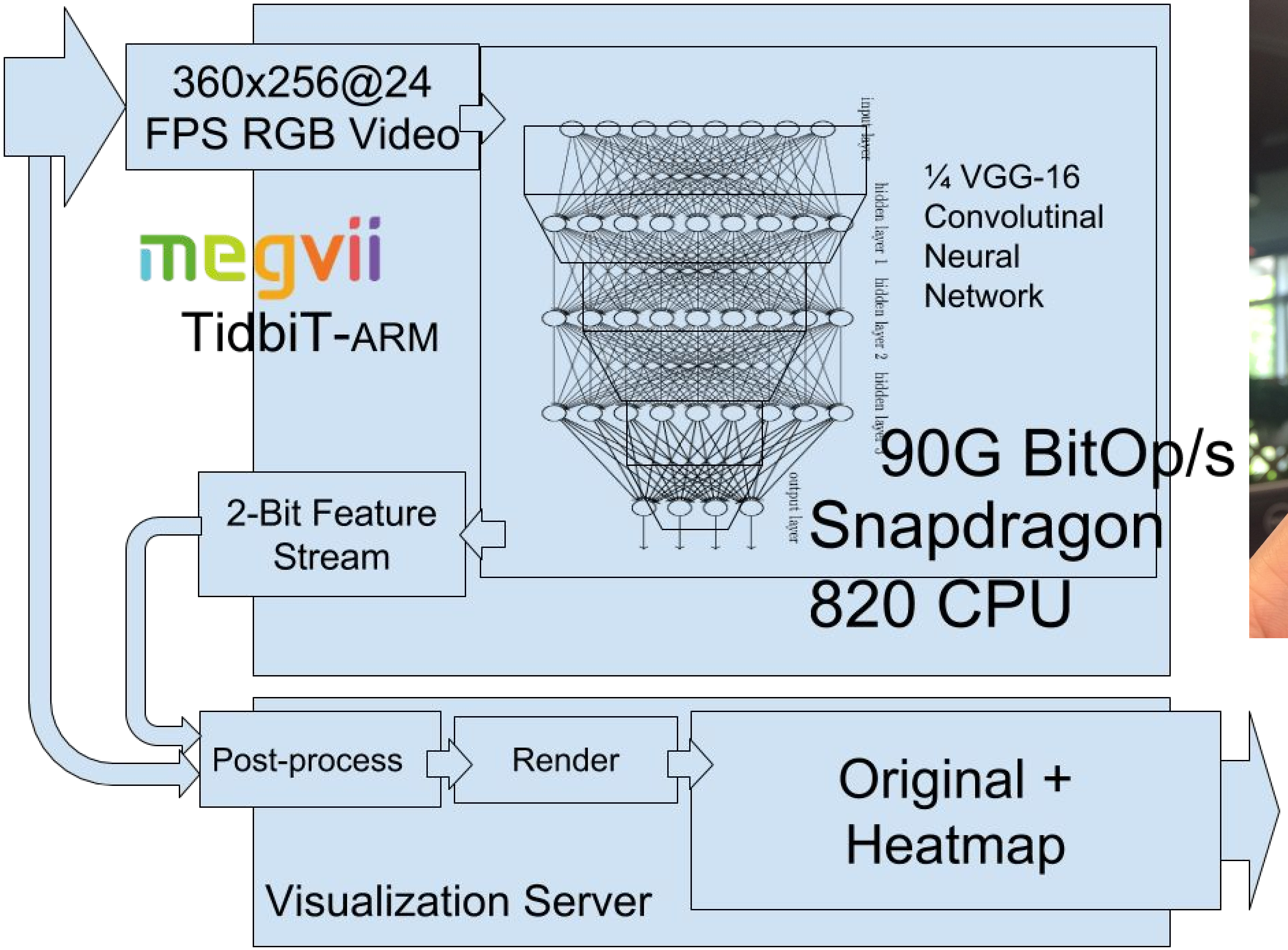
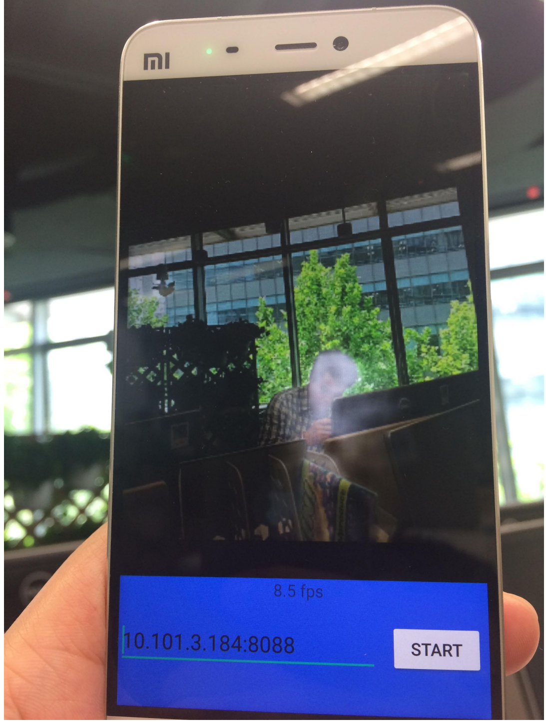


Figure 4: Examples on PASCAL VOC 2012.

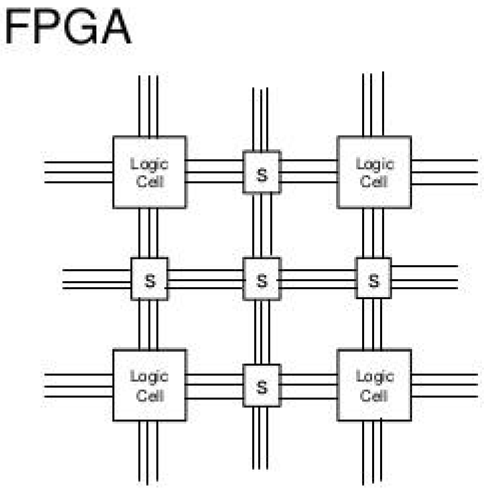


360x256@24
FPS RGB Video
megvii
TidbiT-ARM
input layer
hidden layer 1
hidden layer 2
output layer
¼ VGG-16
Convolutinal
Neural
Network
90G BitOp/s
Snapdragon
820 CPU
2-Bit Feature
Stream
Post-process
Render
Original +
Heatmap
Visualization Server

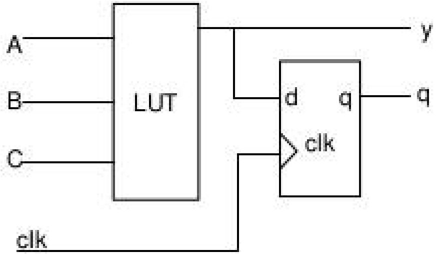


8.5 fps
10.101.3.184:8088
START

FPGA is made up of many LUT's



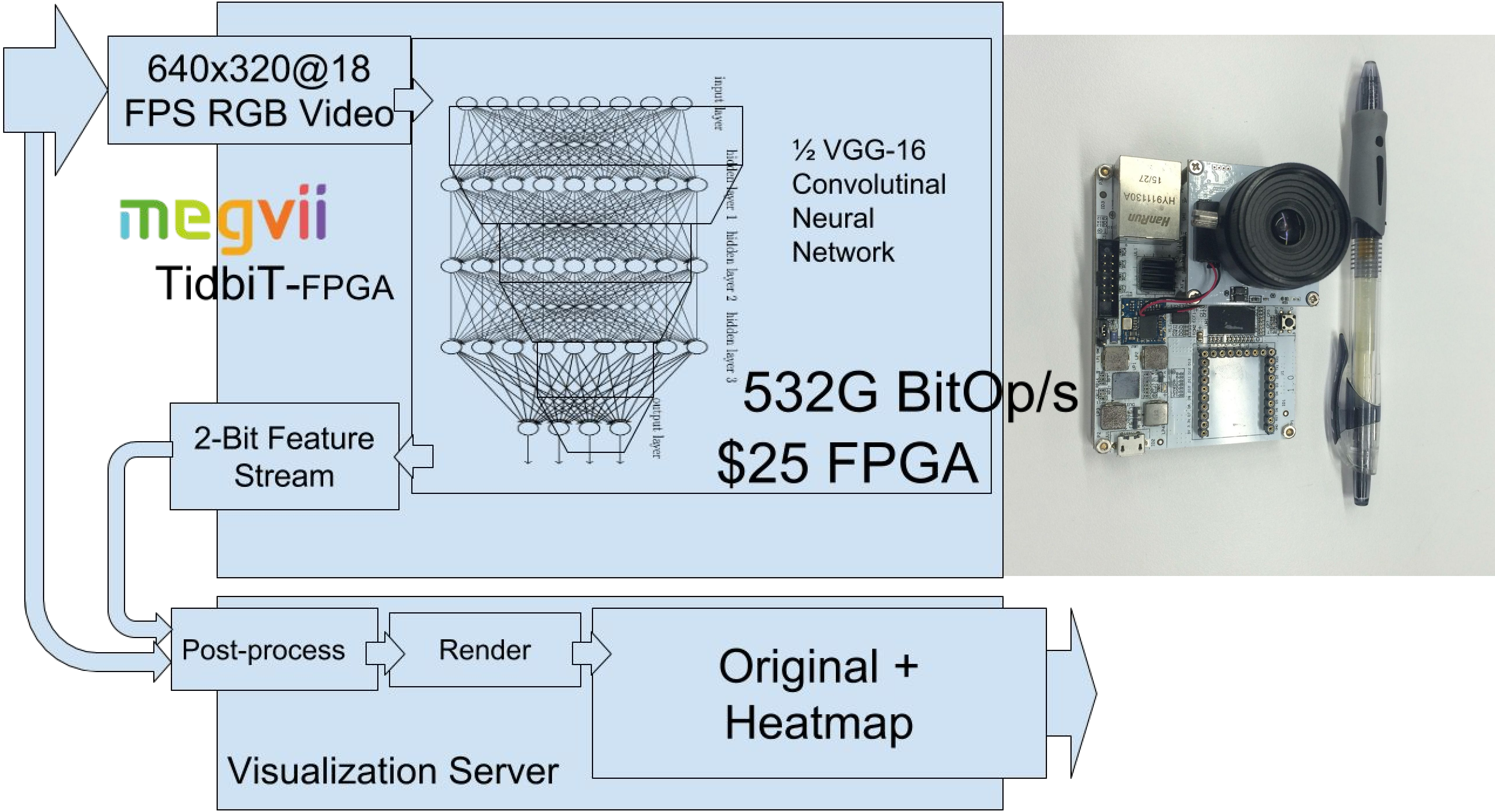
a) Conceptual structure of an FPGA device.



A	B	C	y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

b) Three-input LUT-based logic cell

Flow


640x320@18
FPS RGB Video
megvii
TidbiT-FPGA
input layer
hidden layer 1
hidden layer 2
hidden layer 3
output layer
½ VGG-16
Convolutinal
Neural
Network
532G BitOp/s
$25 FPGA
2-Bit Feature
Stream
Post-process
Render
Original +
Heatmap
Visualization Server

TernGrad: Ternary Gradients to ReduceCommunication in Distributed Deep Learning 1705

- Weights and activations not quantized.

Table 2: Accuracy comparison for *AlexNet*.

base LR	mini-batch size	workers	iterations	gradients	weight decay	DR†	top-1	top-5
0.01	256	2	370K	floating	0.0005	0.5	57.33%	80.56%
				TernGrad	0.0005	0.2	57.61%	80.47%
				TernGrad-noclip ‡	0.0005	0.2	54.63%	78.16%
0.02	512	4	185K	floating	0.0005	0.5	57.32%	80.73%
				TernGrad	0.0005	0.2	57.28%	80.23%
0.04	1024	8	92.5K	floating	0.0005	0.5	**56.62%**	80.28%
				TernGrad	0.0005	0.2	**57.54%**	80.25%

† DR: dropout ratio, the ratio of dropped neurons. ‡ *TernGrad* without gradient clipping.

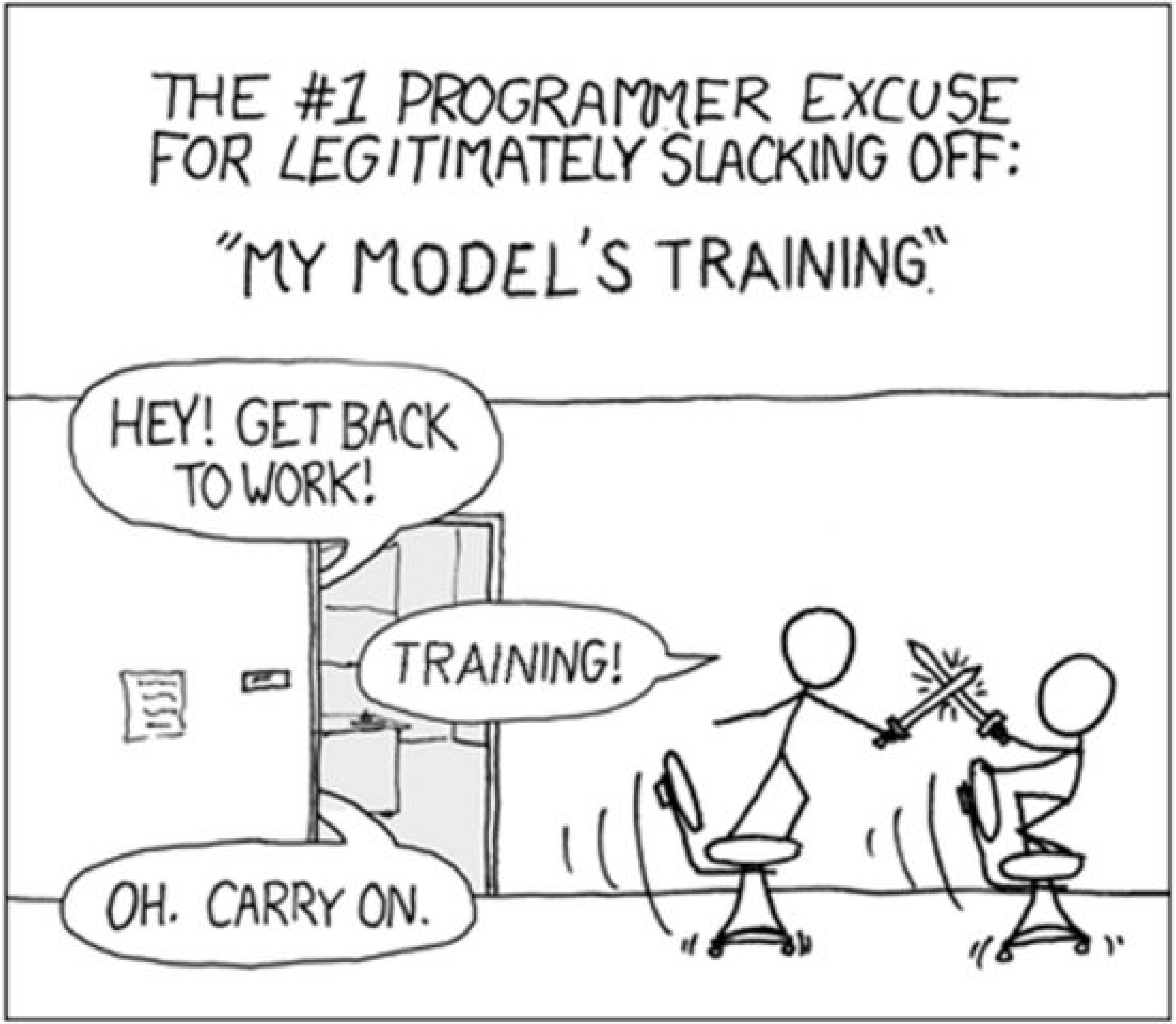
More References

- Xiangyu Zhang, Jianhua Zou, Kaiming He, Jian Sun: Accelerating Very Deep Convolutional Networks for Classification and Detection. IEEE Trans. Pattern Anal. Mach. Intell. 38(10): 1943-1955 (2016)
- ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices https://arxiv.org/abs/1707.01083
- Aggregated Residual Transformations for Deep Neural Networks https://arxiv.org/abs/1611.05431
- Convolutional neural networks with low-rank regularization https://arxiv.org/abs/1511.06067

Backup after this slide

Slide also available at my home page:

https://zsc.github.io/


THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY MODEL'S TRAINING."
HEY! GET BACK TO WORK!
TRAINING!
OH. CARRY ON.

Face++ 旷视

Low-rankness of Activations

- Accelerating Very Deep Convolutional Networks for Classification and Detection