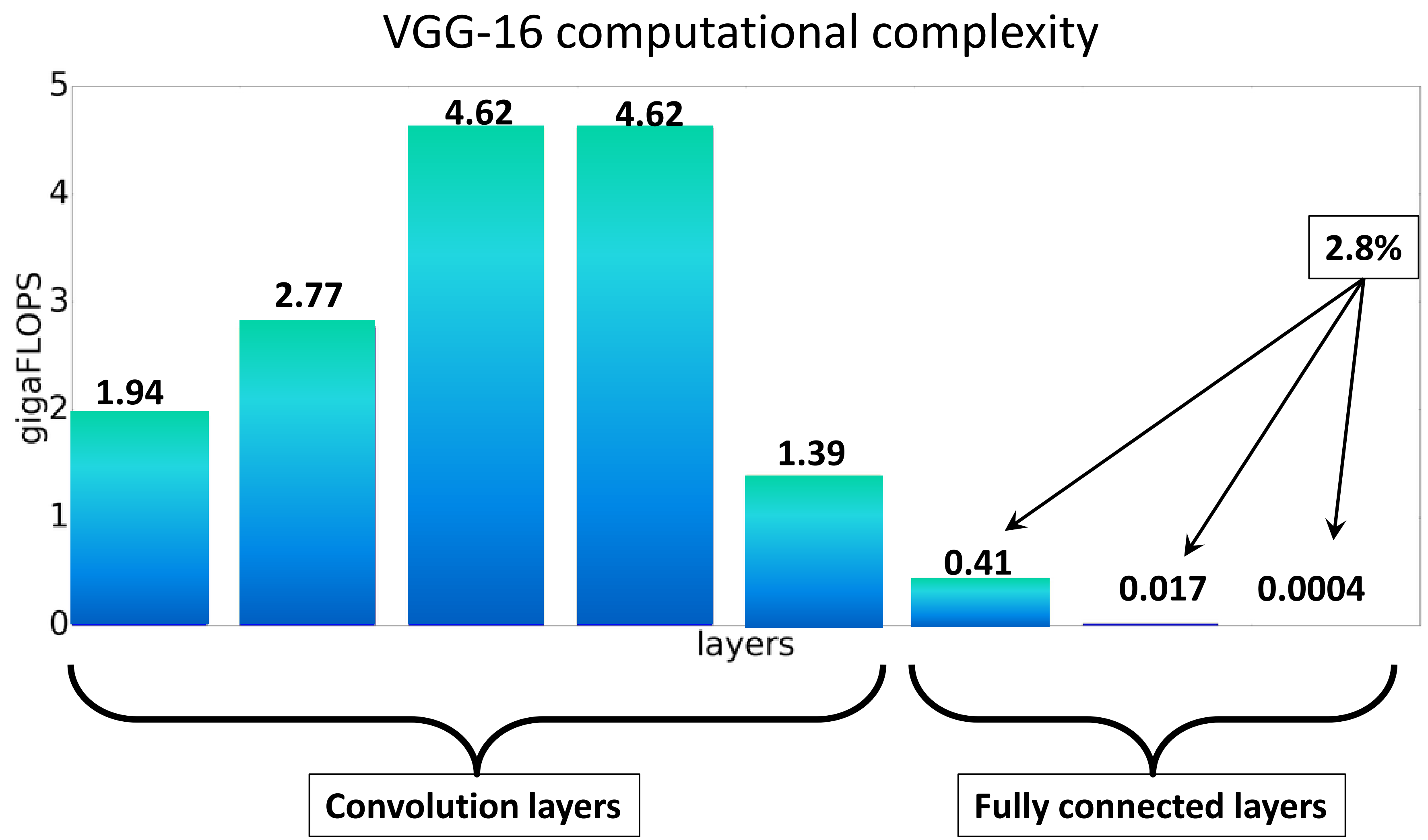
Low rank approximation for Convolution Neural Network

Samsung R&D Institute Ukraine
Vitaliy Bulygin

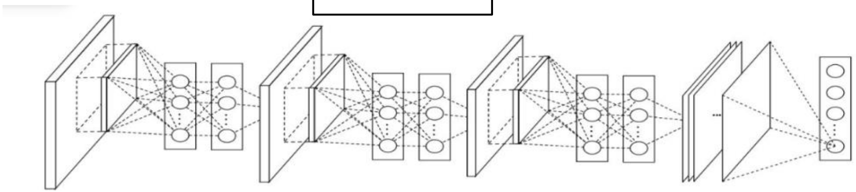

VGG-16 computational complexity
5
4
3
2
1
0
gigaFLOPS
1.94
2.77
4.62
4.62
1.39
0.41
0.017
0.0004
2.8%
layers
Convolution layers
Fully connected layers

A big problem of any CNN approximation model

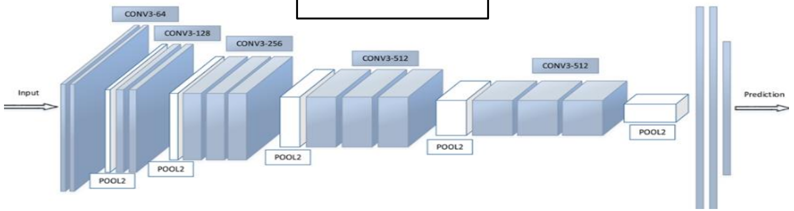
Fine-tuning
or re-train requirement

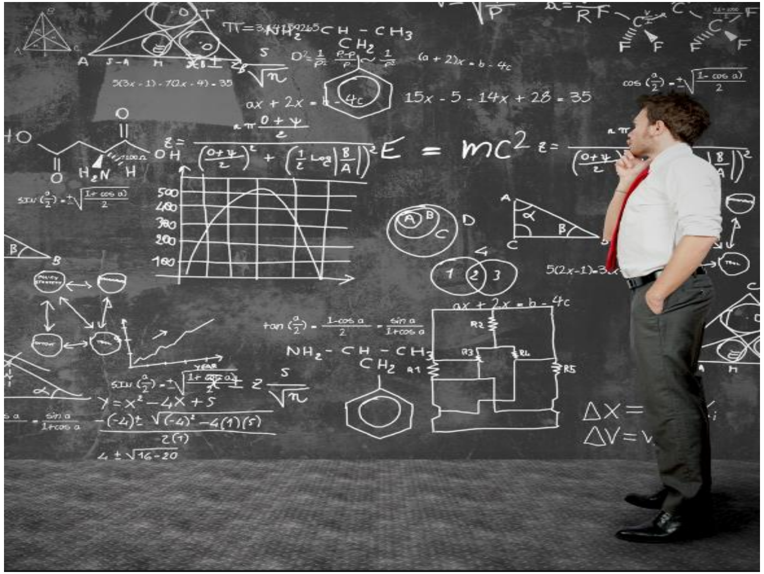
- Choose optimizer coefficients
- way to change them during training process
- batch size
- weight normalization
- dropout, etc

ResNet



VGG-16





A big problem of any CNN approximation model

Fine-tuning requirement → You need to know the learning process of the model

MS-CNN for pedestrian detection **Solver**

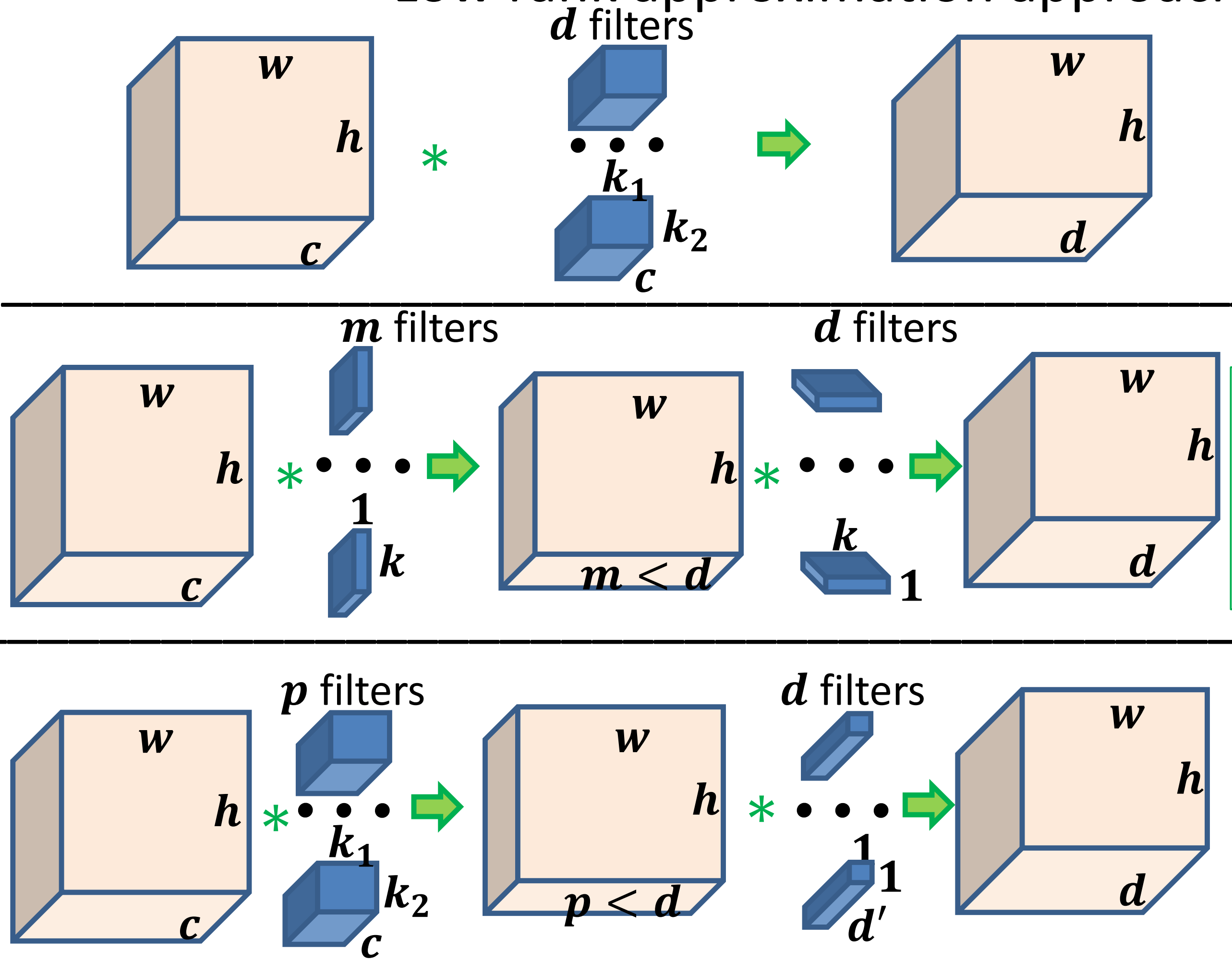
A big problem of any CNN approximation model

Learning process of the
deep convolution neural network
is **dark magic**

It is obtained manually by trial and error

Approximation process changes the model architecture.
Therefore learning process of the exact model
is not correct for approximated model

Low rank approximation approaches



A full rank convolution layer

Jaderberg et al., 2014
Y. Ioannou et al.,2016
Cheng Tai et al., 2016
etc ...

X. Zhang et al., 2015
W. Wen et al.,2017
Lebedev et al., 2015
...

CNN approximation without fine-tuning

Xiangyu Zhang, Jianhua Zou, Kaiming He †, and Jian Sun

Accelerating Very Deep Convolutional Networks for Classification and Detection

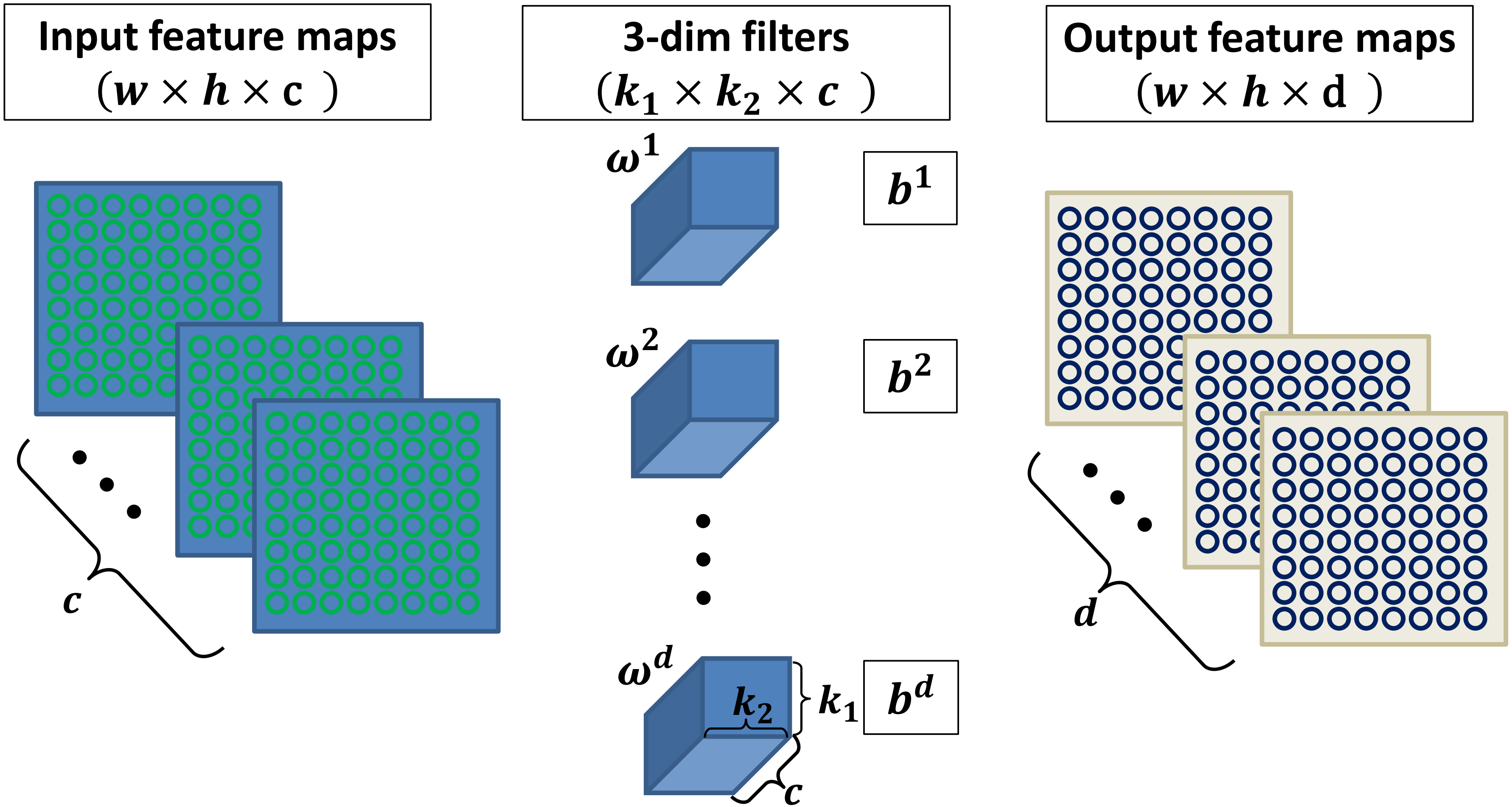
Max Jaderberg, Andrea Vedaldi, Andrew Zisserman

Speeding up Convolutional Neural Networks with Low Rank Expansions

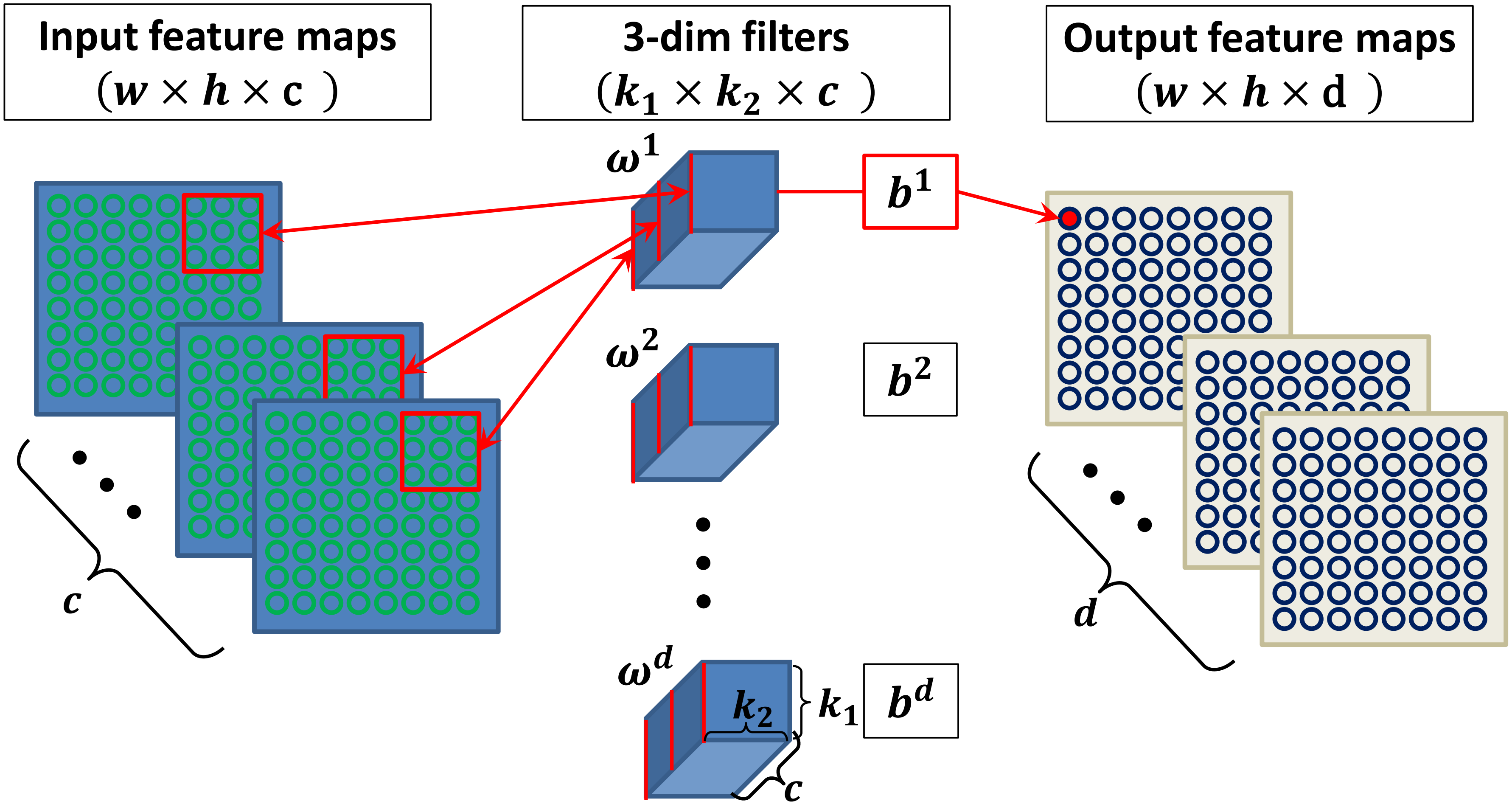
Max Jaderberg, Andrea Vedaldi, Andrew Zisserman

Compression of Deep Convolution Neural Network for Fast and Low Power Mobile Applications

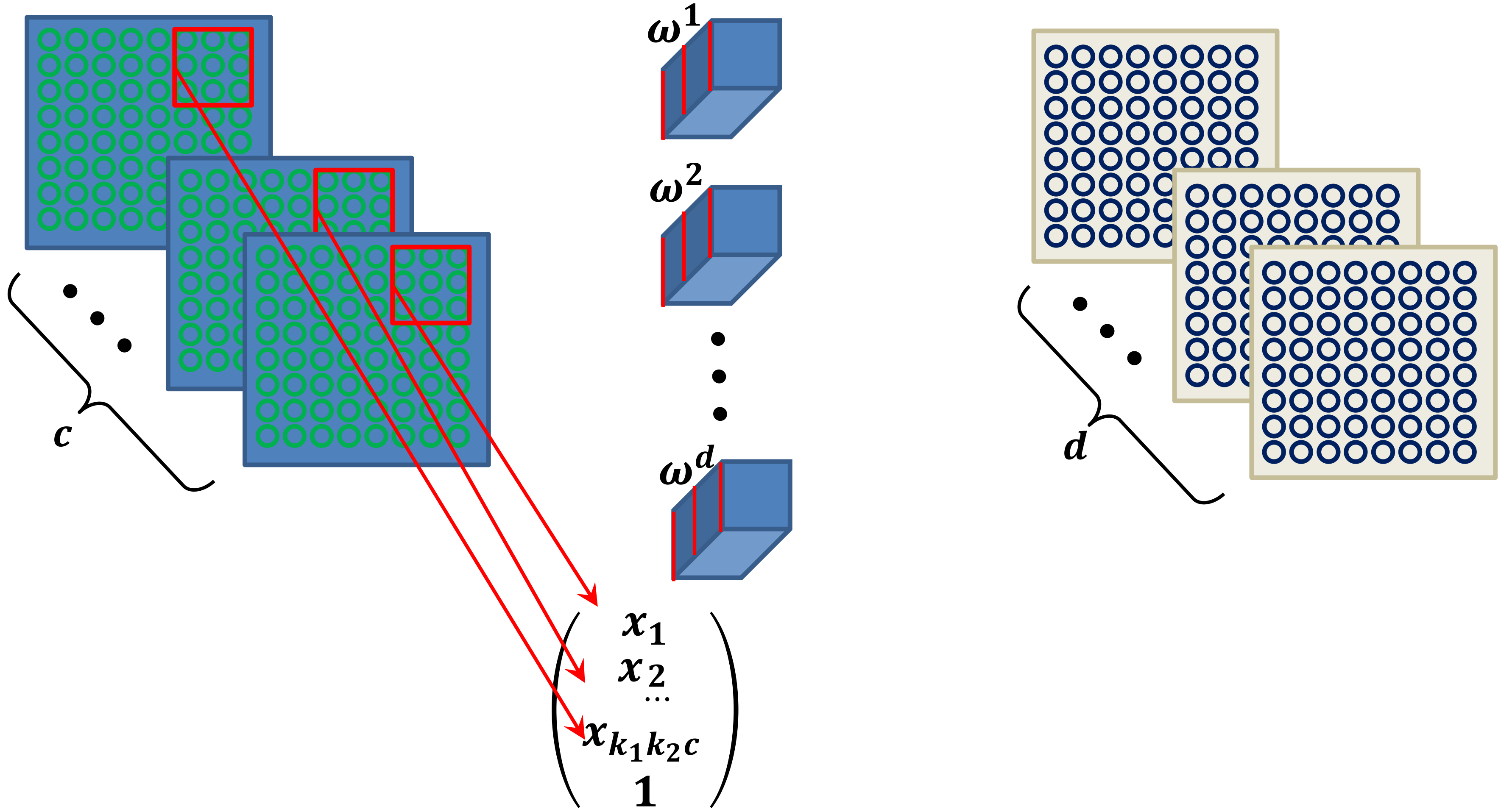
Convolution layer as matrix multiplication



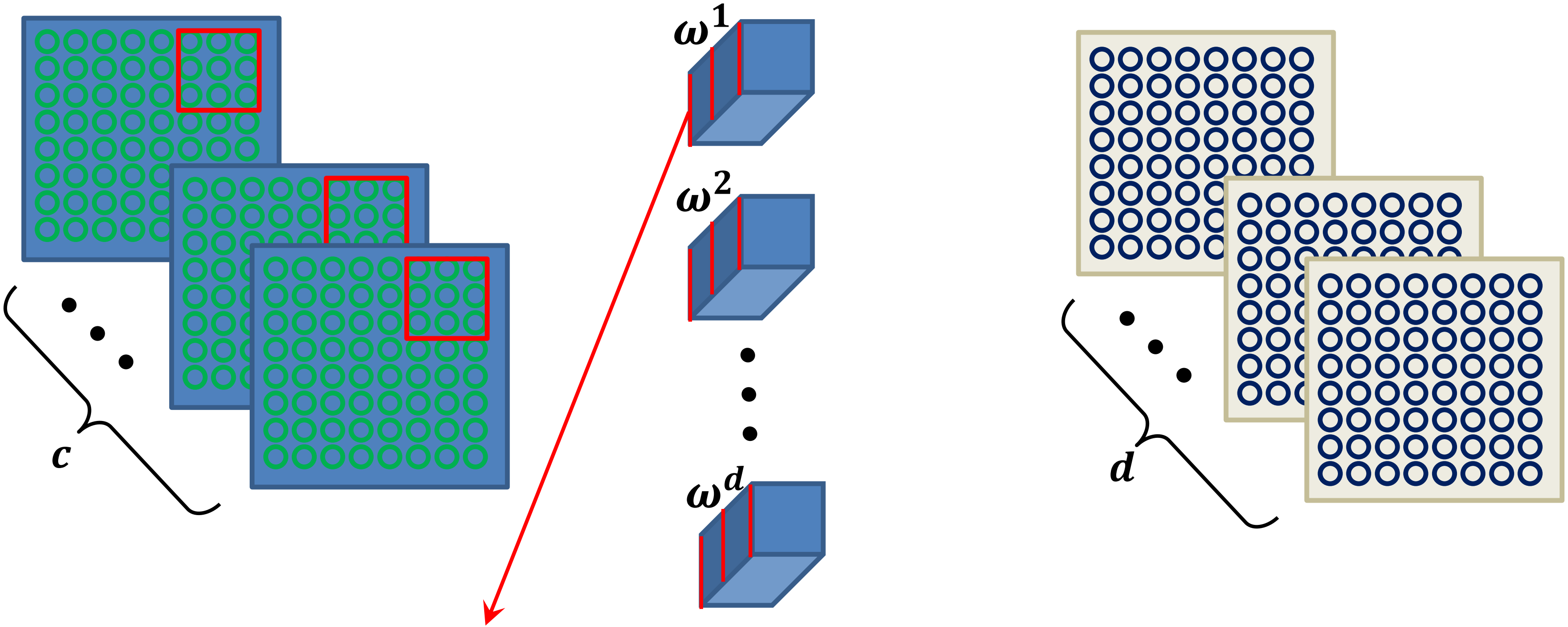
Convolution layer as matrix multiplication



Convolution layer as matrix multiplication



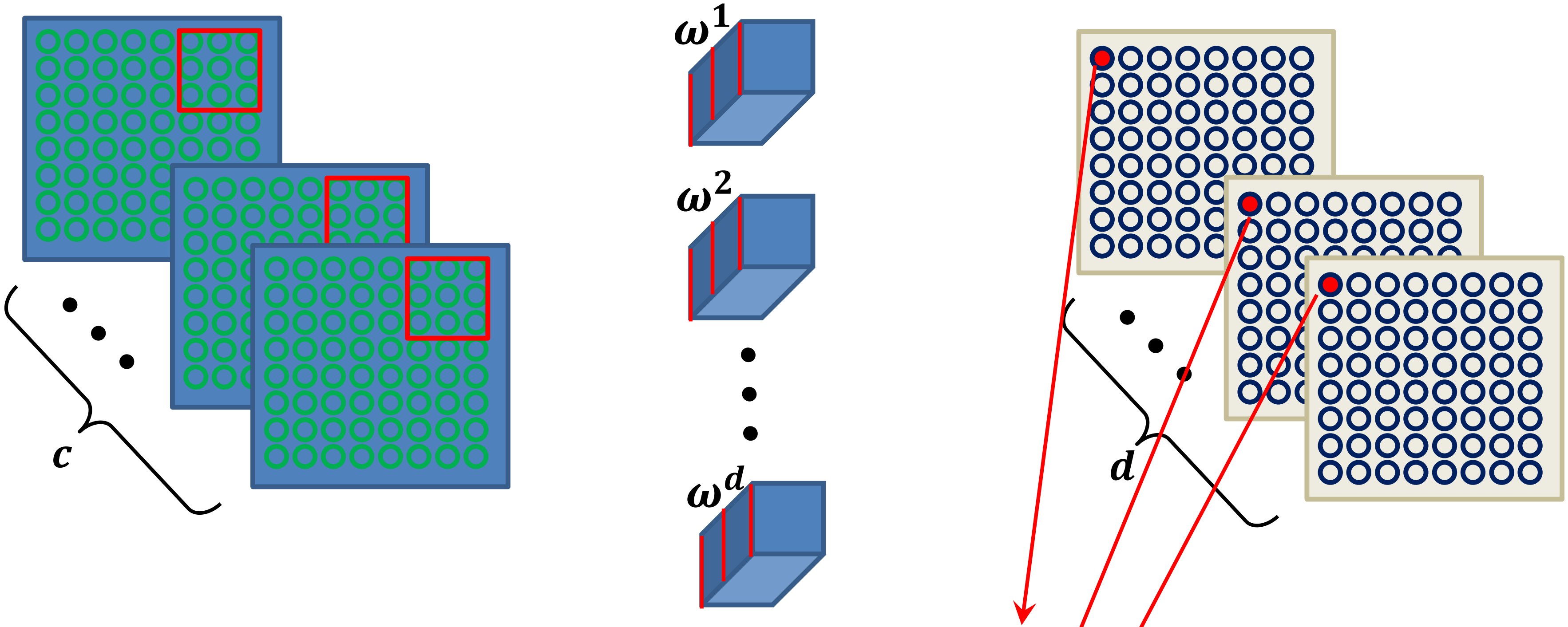
Convolution layer as matrix multiplication



$$(\omega_{11}, \omega_{21}, \cdots, \omega_{k_1 k_2 c}, b_1)$$

$$\begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_{k_1 k_2 c} \\ 1 \end{pmatrix}$$

Convolution layer as matrix multiplication



$$\begin{pmatrix} \omega_{11}, \omega_{12}, \cdots, \omega_{1,k_1k_2c}, b_1 \\ \cdots \\ \cdots \\ \omega_{d1}, \omega_{12}, \cdots, \omega_{d,k_1k_2c}, b_d \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_{k_1k_2c} \\ 1 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \cdots \\ y_d \end{pmatrix}$$

Convolution layer as matrix multiplication

$$\begin{pmatrix} \omega_{11}, \omega_{12}, \cdots, \omega_{1,k_1k_2c}, b_1 \\ \cdots \\ \cdots \\ \omega_{d1}, \omega_{12}, \cdots, \omega_{d,k_1k_2c}, b_d \end{pmatrix} \cdot \left(\vec{x}^1, \cdots, \vec{x}^n\right) = \left(\vec{y}^1, \cdots, \vec{y}^n\right)$$

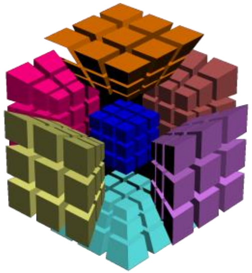
Where $\vec{x}^i \in \mathbb{R}^{k_1k_2c+1}$,

$\vec{y}^i \in \mathbb{R}^d$

$W \in \mathbb{R}^{d,k_1k_2c+1}$

Convolution layer as matrix multiplication

4-dimensional weight filters



$$\begin{pmatrix} \omega_{11}, \omega_{12}, \cdots, \omega_{1,k_1k_2c}, b_1 \\ \cdots \\ \cdots \\ \omega_{d1}, \omega_{12}, \cdots, \omega_{d,k_1k_2c}, b_d \end{pmatrix} \cdot (\vec{x}^1, \cdots, \vec{x}^n) = (\vec{y}^1, \cdots, \vec{y}^n)$$

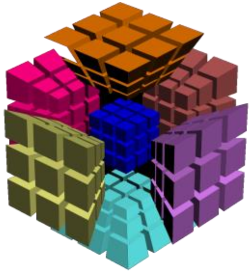
Where $\vec{x}^i \in \mathbb{R}^{k_1k_2c+1}$,

$\vec{y}^i \in \mathbb{R}^d$

$W \in \mathbb{R}^{d,k_1k_2c+1}$

Convolution layer as matrix multiplication

4-dimensional weight filters



$$\begin{pmatrix} \boldsymbol{\omega_{11}}, \boldsymbol{\omega_{12}}, \cdots, \boldsymbol{\omega_{1,k_1k_2c}}, \boldsymbol{b_1} \\ \cdots \\ \cdots \\ \boldsymbol{\omega_{d1}}, \boldsymbol{\omega_{12}}, \cdots, \boldsymbol{\omega_{d,k_1k_2c}}, \boldsymbol{b_d} \end{pmatrix} \cdot (\vec{\boldsymbol{x}}^{\boldsymbol{1}}, \cdots, \vec{\boldsymbol{x}}^{\boldsymbol{n}}) = (\vec{\boldsymbol{y}}^{\boldsymbol{1}}, \cdots, \vec{\boldsymbol{y}}^{\boldsymbol{n}})$$

$\boldsymbol{W} \cdot \boldsymbol{x} = \boldsymbol{y}$

$\boldsymbol{W}$ $\boldsymbol{x}$ $\boldsymbol{y}$

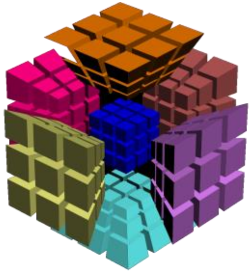
Where $\vec{\boldsymbol{x}}^{\boldsymbol{i}} \in \mathbb{R}^{\boldsymbol{k_1k_2c+1}}$,

$\vec{\boldsymbol{y}}^{\boldsymbol{i}} \in \mathbb{R}^{\boldsymbol{d}}$

$\boldsymbol{W} \in \mathbb{R}^{\boldsymbol{d,k_1k_2c+1}}$

Convolution layer as matrix multiplication

4-dimensional weight filters



$$\begin{pmatrix} \omega_{11}, \omega_{12}, \cdots, \omega_{1,k_1k_2c}, b_1 \\ \cdots \\ \cdots \\ \omega_{d1}, \omega_{12}, \cdots, \omega_{d,k_1k_2c}, b_d \end{pmatrix} \cdot (\vec{x}^1, \cdots, \vec{x}^n) = (\vec{y}^1, \cdots, \vec{y}^n)$$

$W \cdot x = y$

W x y

Number of images $\gg 1$

$n = w \cdot h \cdot N$

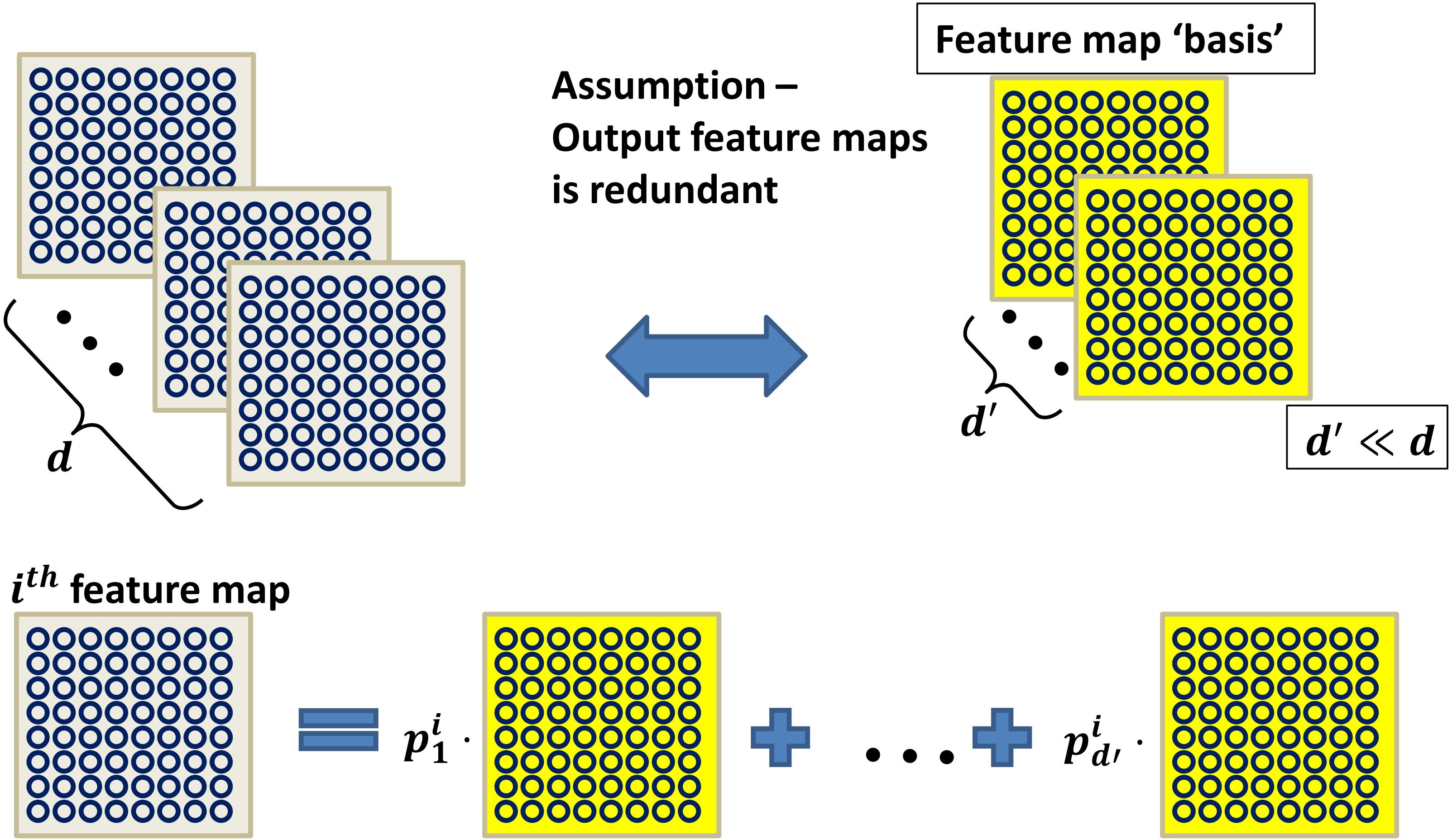
Where $\vec{x}^i \in \mathbb{R}^{k_1k_2c+1}$,

$\vec{y}^i \in \mathbb{R}^d$

$W \in \mathbb{R}^{d,k_1k_2c+1}$

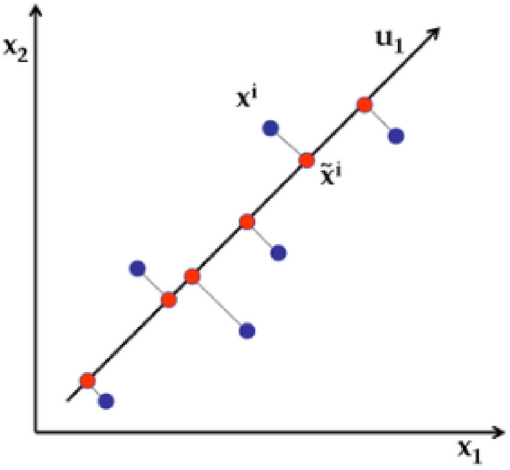
Input feature maps **width** **height**

Low rank approximation of output feature maps



Low rank approximation of output feature maps

PCA gives answer!



Basis $U = (u_1, \cdots u_d)$ is the eigenvectors of yy^t

σ_i is eigenvalues and $\approx$ dispersion of values on y_i axis

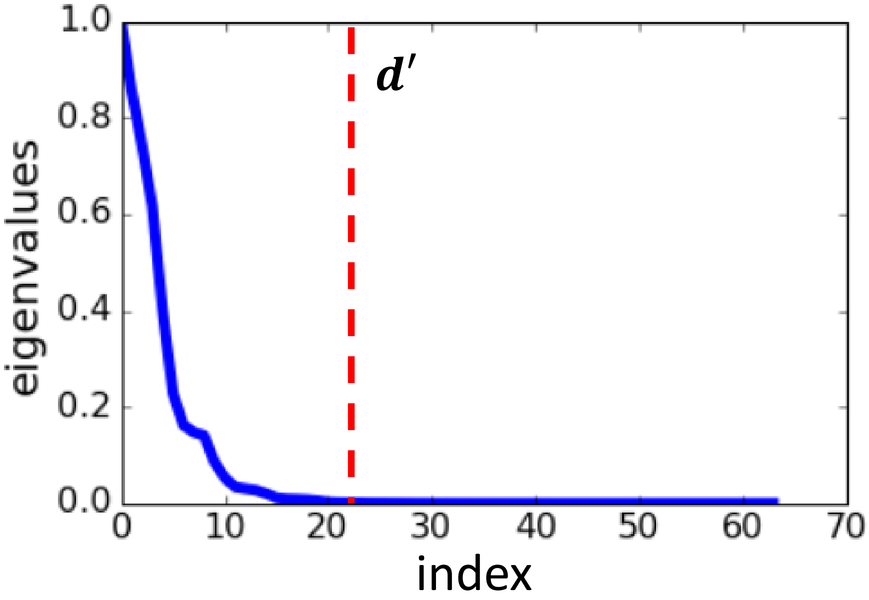
$$yy^t = U \cdot S \cdot U^t = \sum_{i=1}^{d} \sigma_i \cdot u_i^t \cdot u_i,$$

$$\sigma_1 > \sigma_2 > \cdots > \sigma_d$$

Mathematical point of view for d' : $\sum_{i=d'+1}^{d} \sigma_i < \epsilon$

$$\exists \underbrace{u_1, \cdots, u_{d'}}_{\text{Basis}} : \; y^l \approx \sum_{i=1}^{d'} p_i^l \cdot u_i \,,$$

Low rank approximation of output feature maps



yy^t eigenvalues for VGG-16 1st block, 2nd convolution layer, **1000** randomly sampled training images

$$YY^t = U \cdot S \cdot U^t = \sum_{i=1}^{d} \sigma_i \cdot u_i^t \cdot u_i \approx \sum_{i=1}^{d\prime} \sigma_i \cdot u_i^t \cdot u_i$$

$$\sigma_1 > \sigma_2 > \cdots > \sigma_d$$

Separate “heavy” layer on two “light” layers

$$\boxed{\boldsymbol{y} \approx \sum_{i=1}^{d\prime} (\boldsymbol{u}_i, \boldsymbol{y}) \cdot \boldsymbol{u}_i = \boldsymbol{U}_{d'} \cdot \boldsymbol{U}_{d'}^{t} \cdot \boldsymbol{y}}$$

Separate “heavy” layer on two “light” layers

$$\boxed{y \approx \sum_{i=1}^{d'} (u_i, y) \cdot u_i = U_{d'} \cdot U_{d'}^t \cdot y}$$

$$\Longrightarrow$$

$$\boxed{y \approx \overset{\boxed{d \times d'}}{U_{d'}} \cdot \overset{\boxed{d' \times d}}{U_{d'}^t} \cdot \overset{\boxed{d \times n}}{W} \cdot x}$$

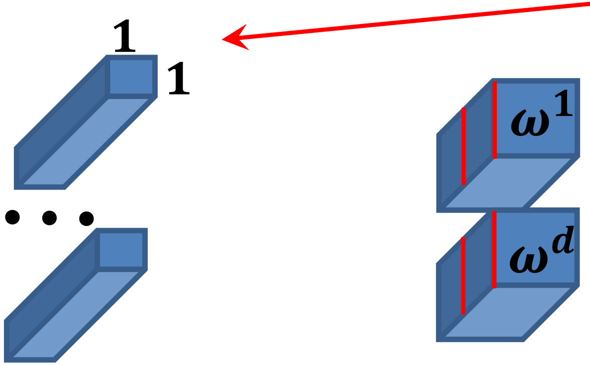
Separate "heavy" layer on two "light" layers

$$y \approx \sum_{i=1}^{d\prime} (u_i, y) \cdot u_i = U_{d'} \cdot U_{d'}^t \cdot y$$

$d \times d'$ $\quad$ $d' \times d$ $\quad$ $d \times n$

$$y \approx \underbrace{U_{d'}}_{P} \cdot \underbrace{U_{d'}^t \cdot W}_{W'} \cdot x$$

$W' \in \mathbb{R}^{d',d}$



$$\begin{pmatrix} \omega_{11}, \omega_{12}, \cdots, \omega_{1,k_1k_2c}, b_1 \\ \cdots \\ \cdots \\ \omega_{d1}, \omega_{12}, \cdots, \omega_{d,k_1k_2c}, b_d \end{pmatrix}$$

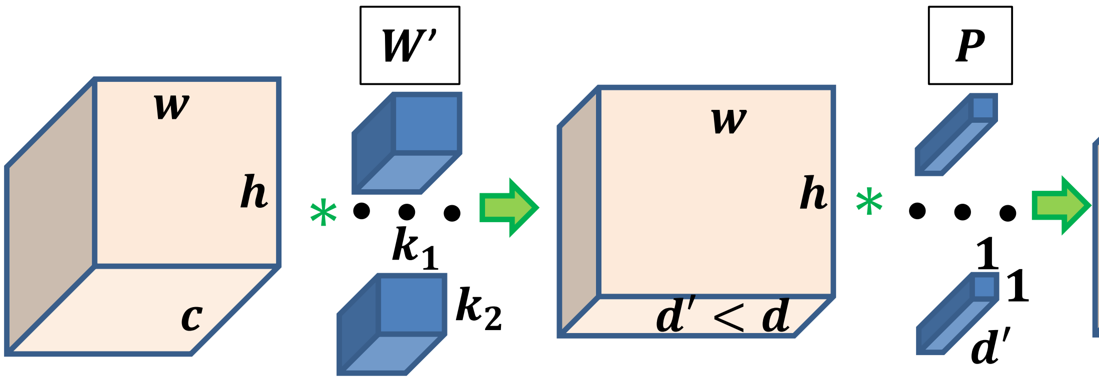
Mathematically we find low-rank matrix A :

$$\|W \cdot x - A \cdot W \cdot x\| \xrightarrow[A]{} min$$

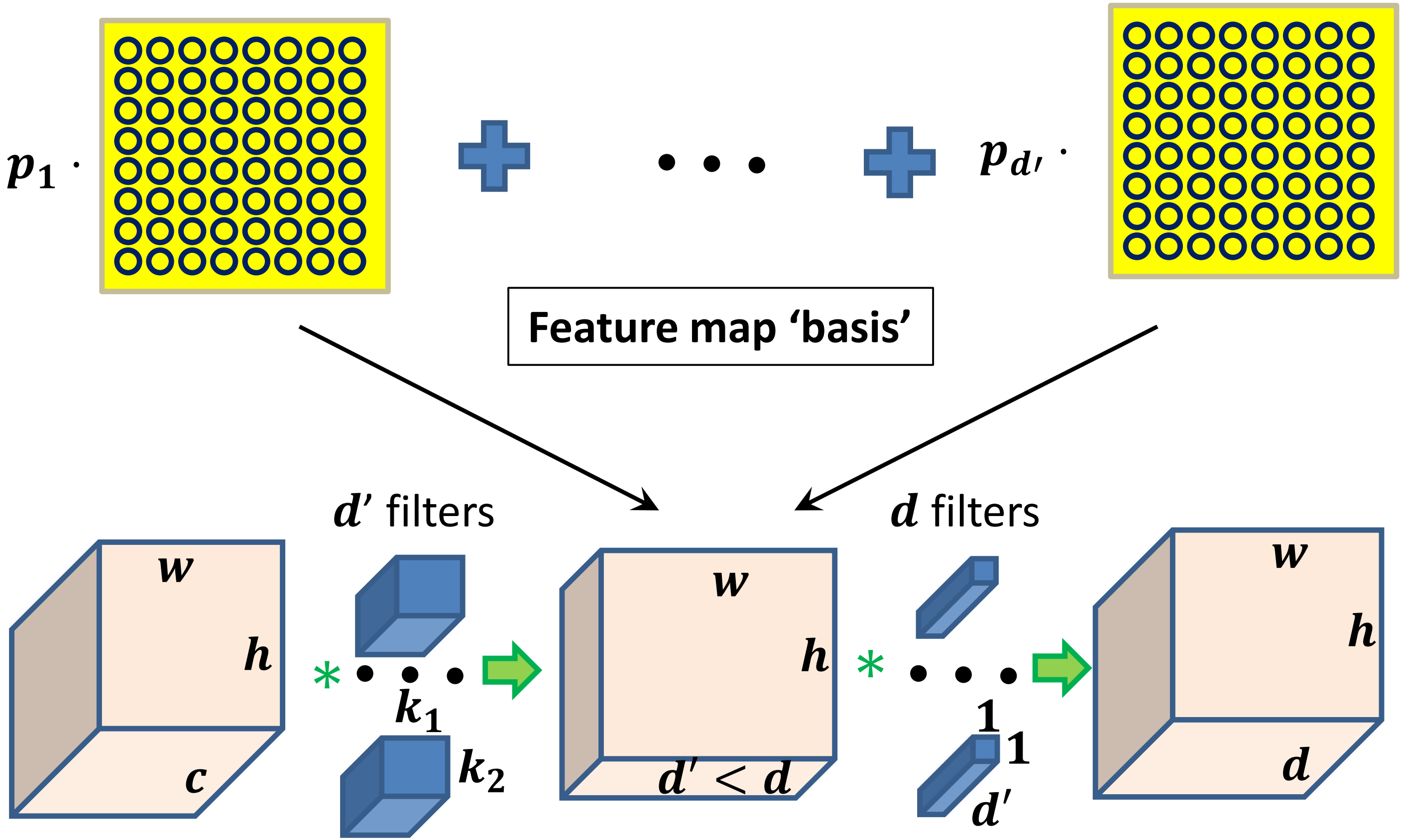
Separate "heavy" layer on two "light" layers

$$y \approx \sum_{i=1}^{d'} (u_i, y) \cdot u_i = U_{d'} \cdot U_{d'}^t \cdot y$$

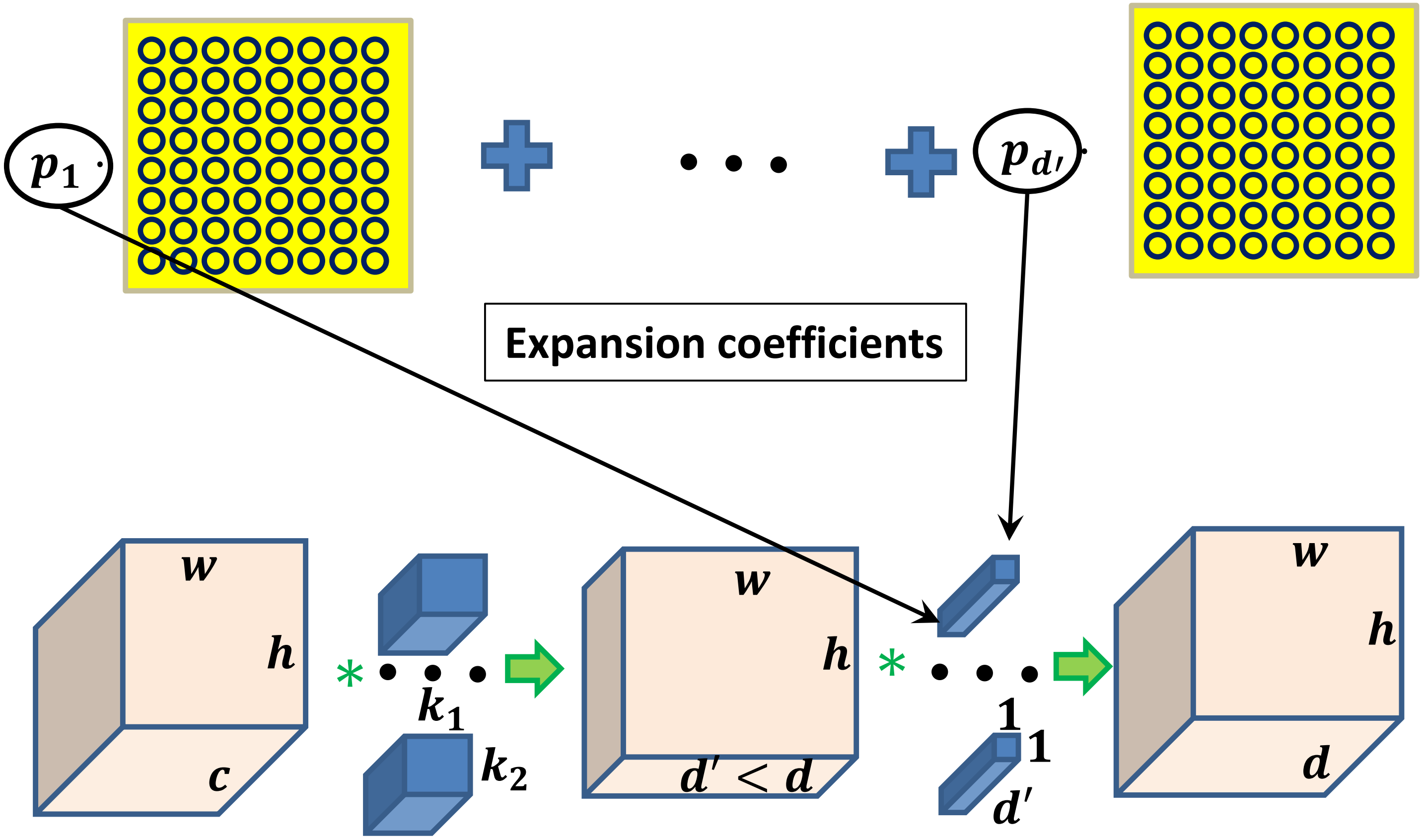
$$y \approx \underbrace{U_{d'}}_{P} \cdot \underbrace{U_{d'}^t \cdot W}_{W'} \cdot x$$



Separate “heavy” layer on two “light” layers



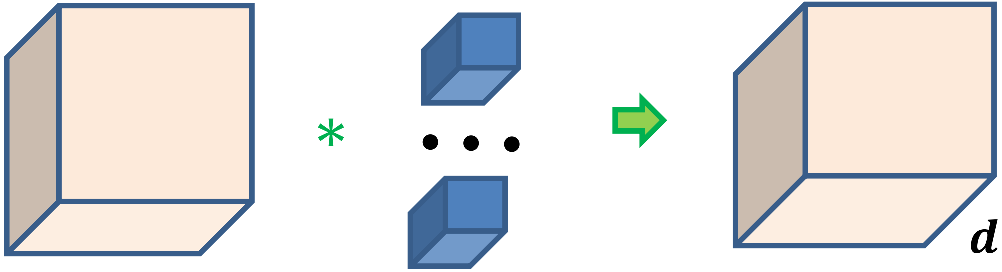
Separate “heavy” layer on two “light” layers



Separate “heavy” layer on two “light” layers

Numerical complexity

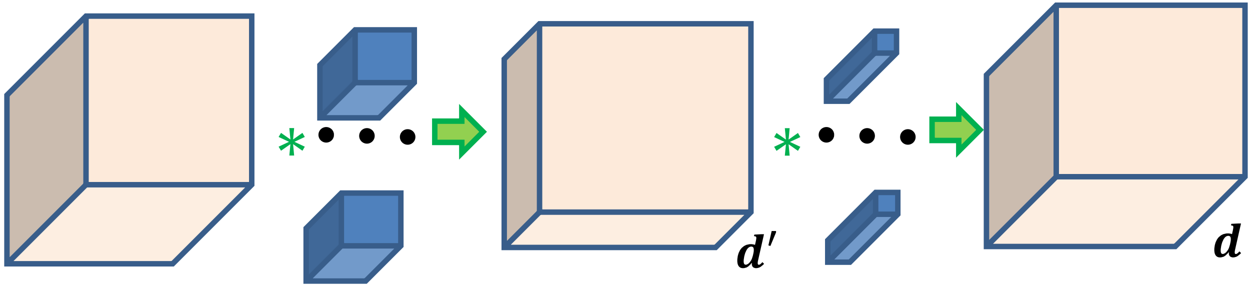
“heavy” layer



$O(d \cdot k_1 \cdot k_2 \cdot c\,)$

Reduce video memory and accelerate on $\approx \boldsymbol{d/d'}$ times

two “light” layers

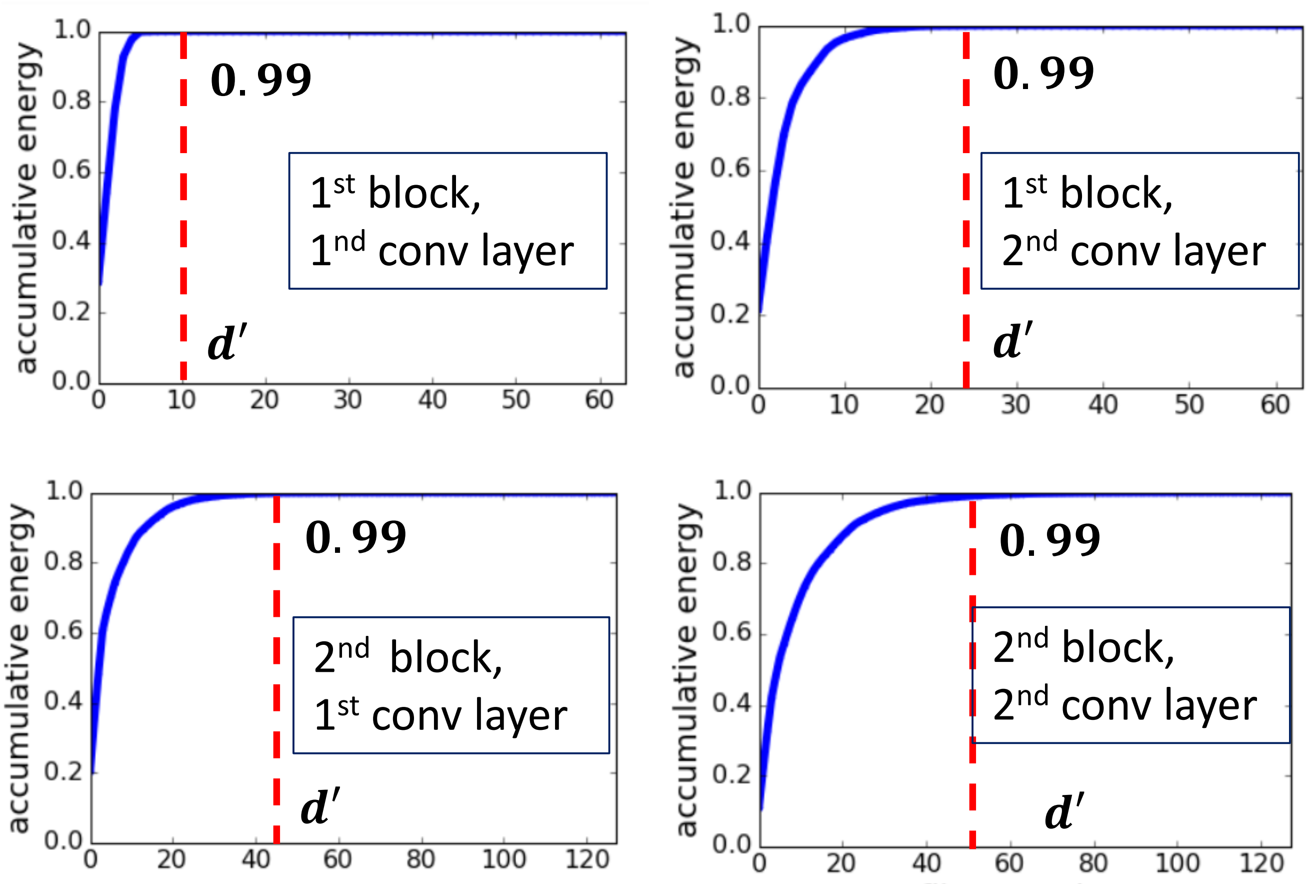


$O(d' \cdot k_1 \cdot k_2 \cdot c)+$
$+O(dd')$

How to choose $\mathbf{d}'$

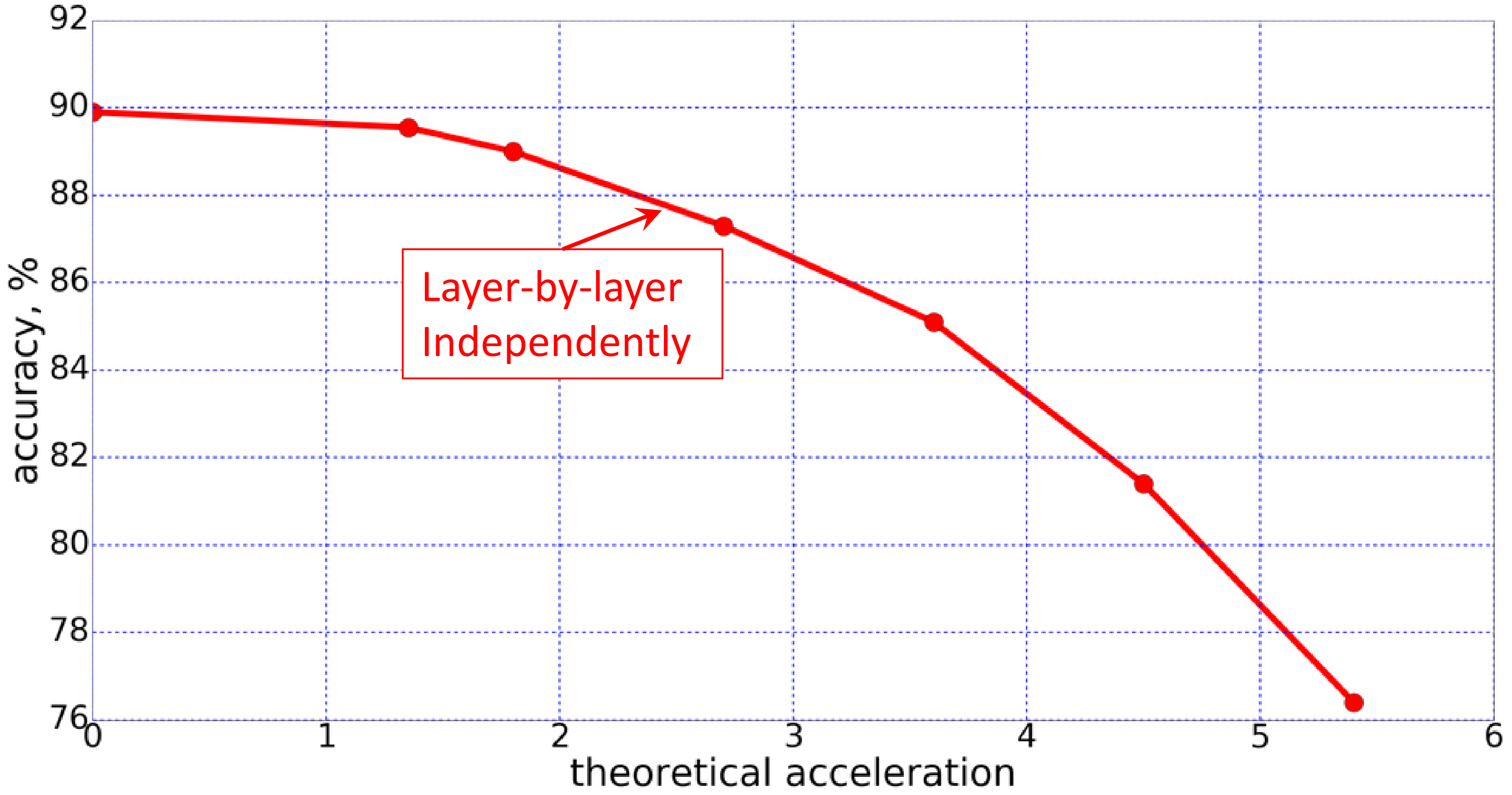
PCA Accumulative energy
$e_j = \sum_{i=1}^{j} \sigma_i / \sum_{i=1}^{d} \sigma_i$

y − Responses from 1000 randomly sampled training images



Results

Top-5 error on ILSVRC-2012 (ImageNet) validation dataset (50K images)

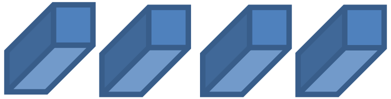


Error accumulation avoiding

Layer-by-layer

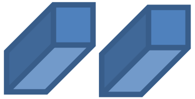
$$\|W \cdot x^n - A \cdot W \cdot x^n\| \underset{A}{\rightarrow} min$$

Exact model $\rightarrow$ $\cdots$ $\rightarrow$ x^n $*$

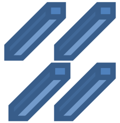


$\longrightarrow$ y^n

Approximate model $\rightarrow$ $\cdots$ $\rightarrow$ $\hat{x}^n = x^n + \epsilon$ $*$

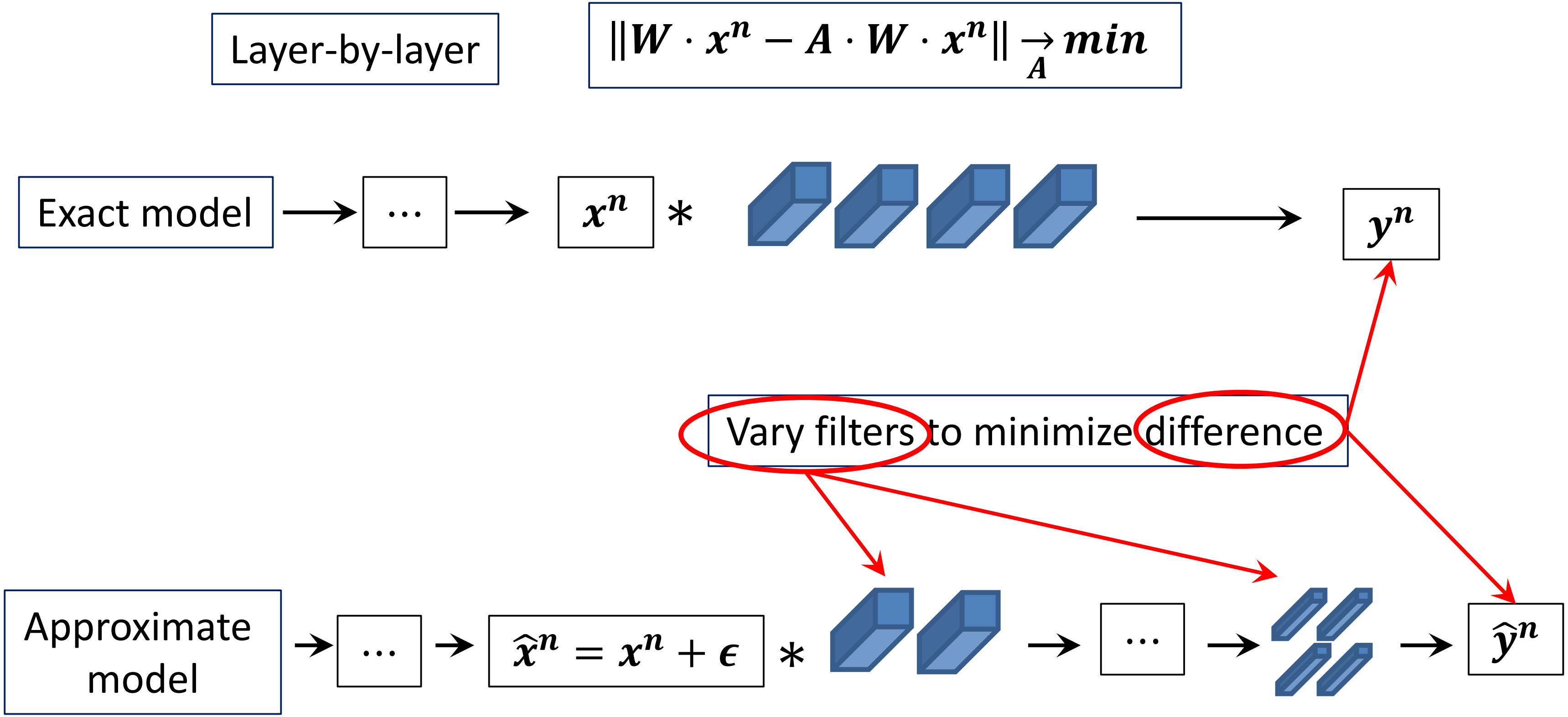


$\rightarrow$ $\cdots$ $\rightarrow$



$\rightarrow$ $\hat{y}^n$

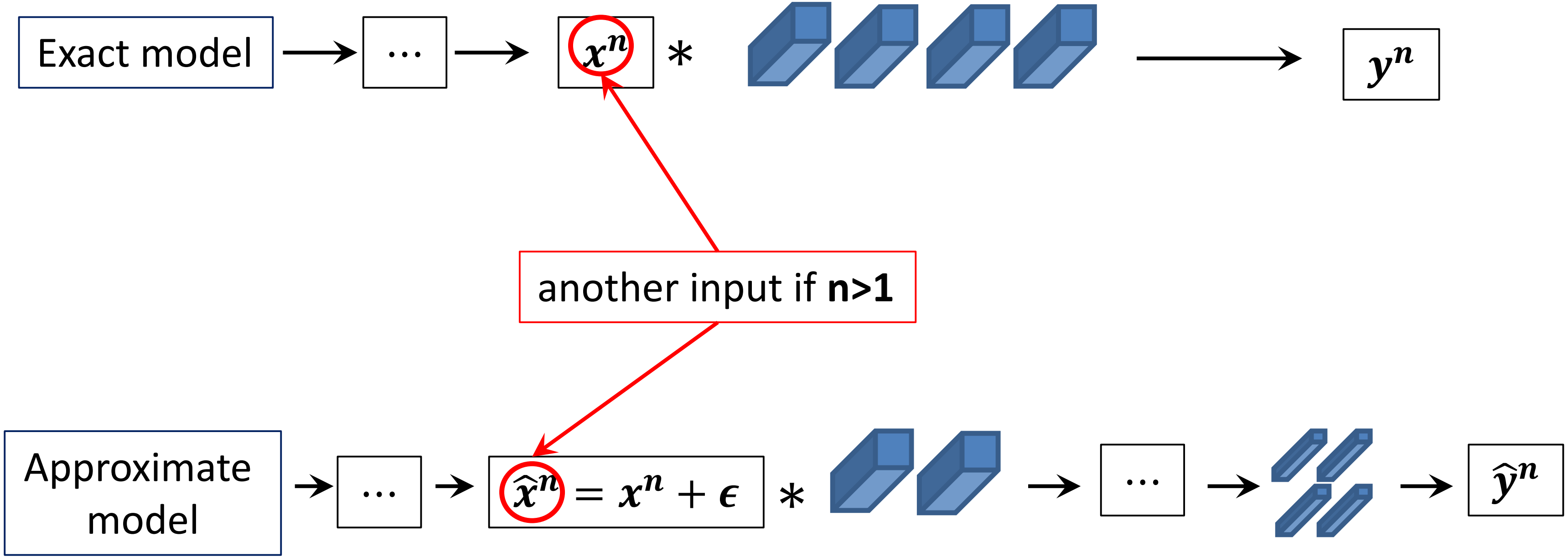
Error accumulation avoiding



Error accumulation avoiding

Layer-by-layer

$$\|\boldsymbol{W}\cdot\boldsymbol{x}^n-\boldsymbol{A}\cdot\boldsymbol{W}\cdot\boldsymbol{x}^n\|\underset{\boldsymbol{A}}{\rightarrow}\boldsymbol{min}$$



Error accumulation avoiding

Layer-by-layer

$$\cancel{\|W \cdot x^n - A \cdot W \cdot \hat{x}^n\| \underset{A}{\rightarrow} min}$$

Taking into account previous layer error

$$\|W \cdot x - A \cdot W \cdot \hat{x}\| \underset{A}{\rightarrow} min$$

Error accumulation avoiding

Layer-by-layer

$$\cancel{\|\boldsymbol{W} \cdot \boldsymbol{x}^{n} - \boldsymbol{A} \cdot \boldsymbol{W} \cdot \boldsymbol{x}^{n}\| \underset{\boldsymbol{A}}{\rightarrow} \boldsymbol{min}}$$

Taking into account previous layer error

$$\|\boldsymbol{W} \cdot \boldsymbol{x} - \boldsymbol{A} \cdot \boldsymbol{W} \cdot \widehat{\boldsymbol{x}}\| \underset{\boldsymbol{A}}{\rightarrow} \boldsymbol{min}$$

Multivariable
Linear regression

$$\boldsymbol{A} = \boldsymbol{W} \cdot \widehat{\boldsymbol{x}} \cdot \boldsymbol{y}^{t} \cdot (\boldsymbol{y} \cdot \boldsymbol{y}^{t})^{-1}$$

Error accumulation avoiding

Layer-by-layer

$$\cancel{\|W \cdot x^n - A \cdot W \cdot \hat{x}^n\| \underset{A}{\rightarrow} min}$$

Taking into account previous layer error

$$\|W \cdot x - A \cdot W \cdot \hat{x}\| \underset{A}{\rightarrow} min$$

Multivariable
Linear regression

$$A = W \cdot \hat{x} \cdot y^t \cdot (y \cdot y^t)^{-1}$$

Inverse matrix does not exist!

Error accumulation avoiding

Layer-by-layer

$$\cancel{\|\boldsymbol{W}\cdot\boldsymbol{x}^{n}-\boldsymbol{A}\cdot\boldsymbol{W}\cdot\boldsymbol{x}^{n}\|\xrightarrow[\boldsymbol{A}]{}\boldsymbol{min}}$$

Taking into account previous layer error

$$\|\boldsymbol{W}\cdot\boldsymbol{x}-\boldsymbol{A}\cdot\boldsymbol{W}\cdot\widehat{\boldsymbol{x}}\|\xrightarrow[\boldsymbol{A}]{}\boldsymbol{min}$$

Multivariable reduced rank regression

$$\boldsymbol{A}=\boldsymbol{W}\cdot\widehat{\boldsymbol{x}}\cdot\boldsymbol{y}^{\boldsymbol{t}}\cdot(\boldsymbol{y}\cdot\boldsymbol{y}^{\boldsymbol{t}})^{-}$$

$(\boldsymbol{y}\cdot\boldsymbol{y}^{\boldsymbol{t}})^{-}$ is generelized inverse matrix

$\boldsymbol{y}\cdot\boldsymbol{y}^{\boldsymbol{t}}$ is low rank $\Rightarrow$ $(\boldsymbol{y}\cdot\boldsymbol{y}^{\boldsymbol{t}})^{-}$is low rank $\Rightarrow$ $\boldsymbol{A}$ is low rank

$$\boldsymbol{A}=\boldsymbol{U}_{\boldsymbol{d}'}\boldsymbol{S}_{\boldsymbol{d}'}\boldsymbol{V}_{\boldsymbol{d}'},\qquad \boldsymbol{d}'<\boldsymbol{d}$$

Error accumulation avoiding

Layer-by-layer

$$\cancel{\|\boldsymbol{W}\cdot \boldsymbol{x}^n - \boldsymbol{A}\cdot \boldsymbol{W}\cdot \boldsymbol{x}^n\| \underset{\boldsymbol{A}}{\rightarrow} \boldsymbol{min}}$$

Taking into account previous layer error

$$\|\boldsymbol{W}\cdot \boldsymbol{x} - \boldsymbol{A}\cdot \boldsymbol{W}\cdot \widehat{\boldsymbol{x}}\| \underset{\boldsymbol{A}}{\rightarrow} \boldsymbol{min}$$

Multivariable reduced rank regression

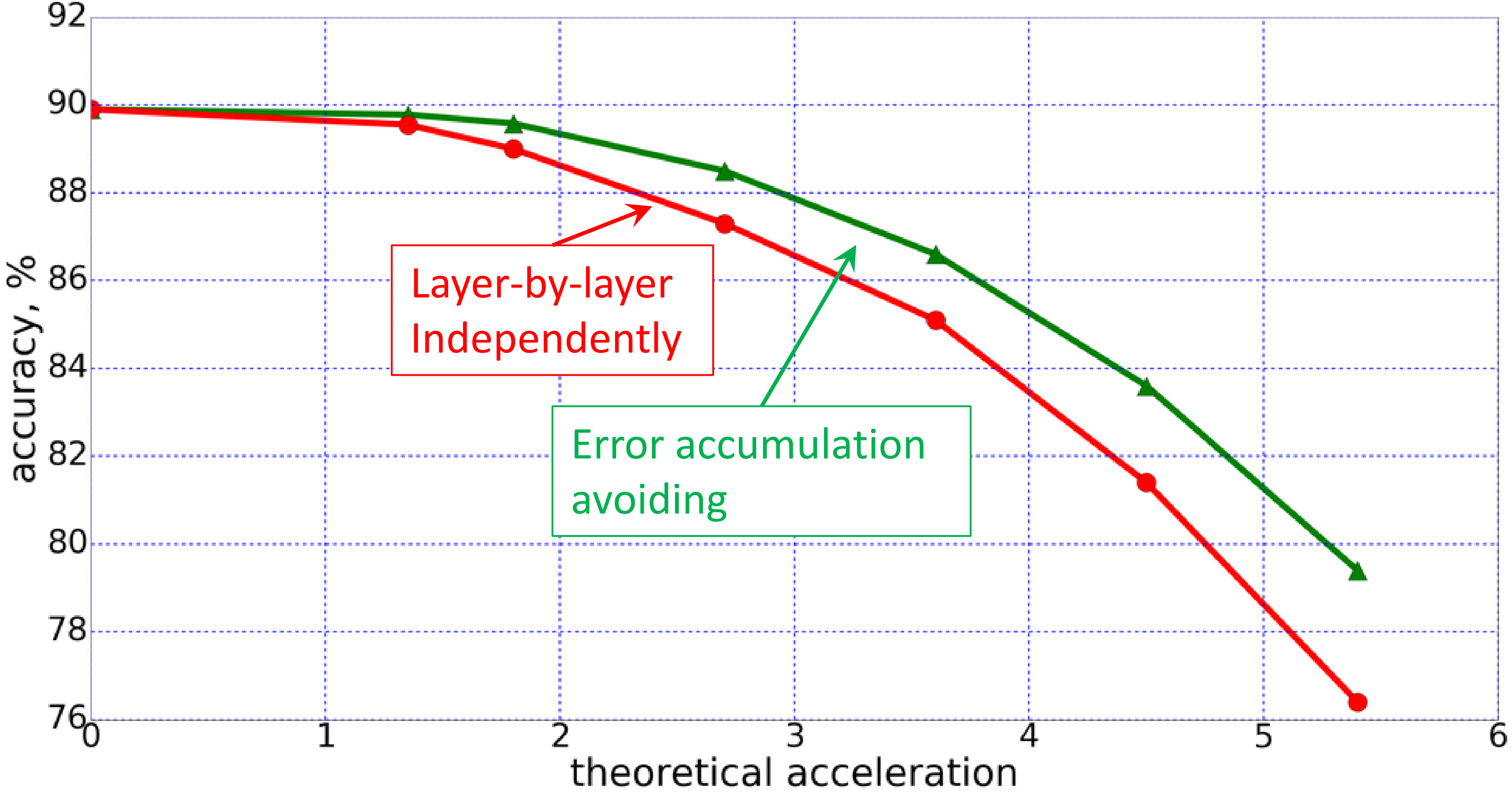
$$\boldsymbol{A} = \boldsymbol{W}\cdot \widehat{\boldsymbol{x}}\cdot \boldsymbol{y}^{\boldsymbol{t}}\cdot (\boldsymbol{y}\cdot \boldsymbol{y}^{\boldsymbol{t}})^{-}$$

$(\boldsymbol{y}\cdot \boldsymbol{y}^{\boldsymbol{t}})^{-}$ is generelized inverse matrix

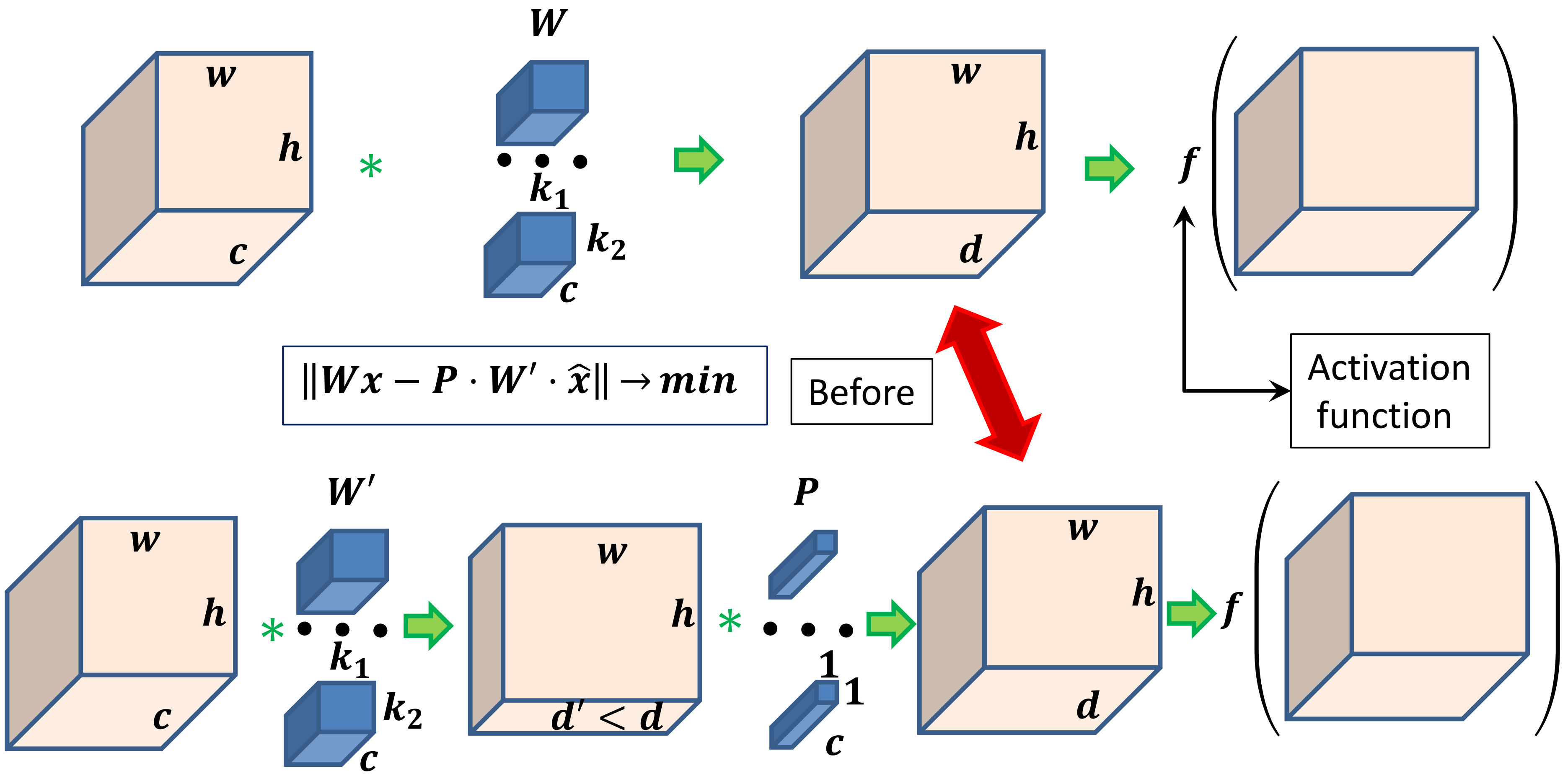
$$\boldsymbol{y} \approx \boldsymbol{A}\cdot \boldsymbol{W}\cdot \widehat{\boldsymbol{x}} = \underbrace{\boldsymbol{U}_{\boldsymbol{d}'}}_{\boldsymbol{P}}\underbrace{\boldsymbol{S}_{\boldsymbol{d}'}\boldsymbol{V}_{\boldsymbol{d}'}\cdot \boldsymbol{W}}_{\boldsymbol{W}' \in \mathbb{R}^{\boldsymbol{d}',\boldsymbol{d}}}\cdot \widehat{\boldsymbol{x}}$$

Results

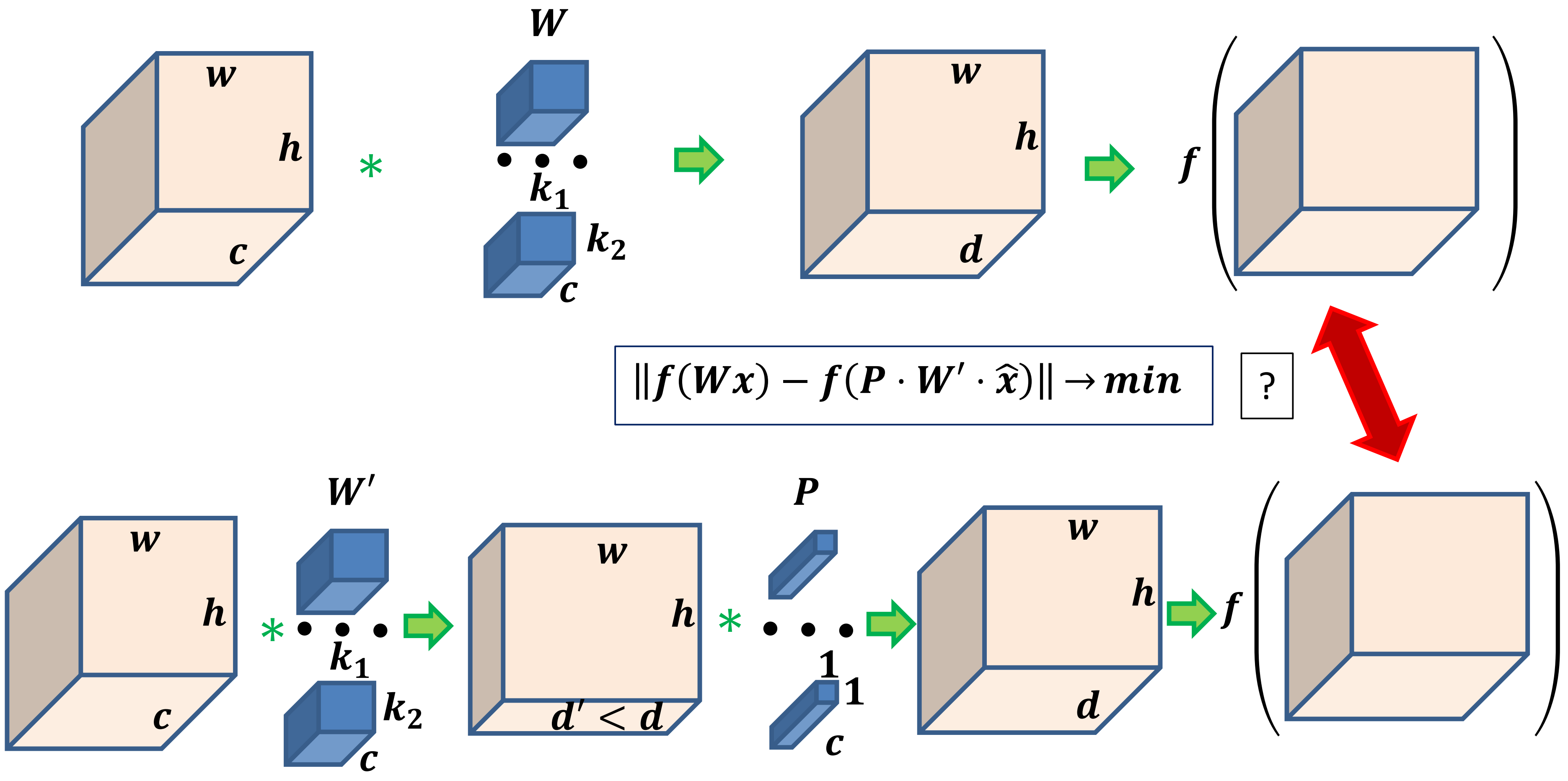
Top-5 error on ILSVRC-2012 (ImageNet) validation dataset (50K images)



Activation function output correspondence



Activation function output correspondence



Activation function output correspondence

We want

$$\|\overset{y}{f(Wx)} - \overset{\hat{y}}{f(P \cdot W' \cdot \hat{x})}\| \to min$$

We have

$$\|Wx - \underbrace{P \cdot W'}_{U_{d'}S_{d'}V_{d'}W} \cdot \hat{x}\| \to min$$

Denote

$$U = U_{d'}\sqrt{S_{d'}}$$

$$V = \sqrt{S_{d'}}V_{d'}$$

Activation function output correspondence

We want

$$L = \|f(y) - f(U \cdot V \cdot \hat{y})\| \underset{U,V}{\longrightarrow} min$$

Gradient descent

$$grad_U L = -2 \cdot \big(f(y) - f(U \cdot V \cdot \hat{y})\big) \cdot (V \cdot \hat{y})^{\mathrm{T}}$$

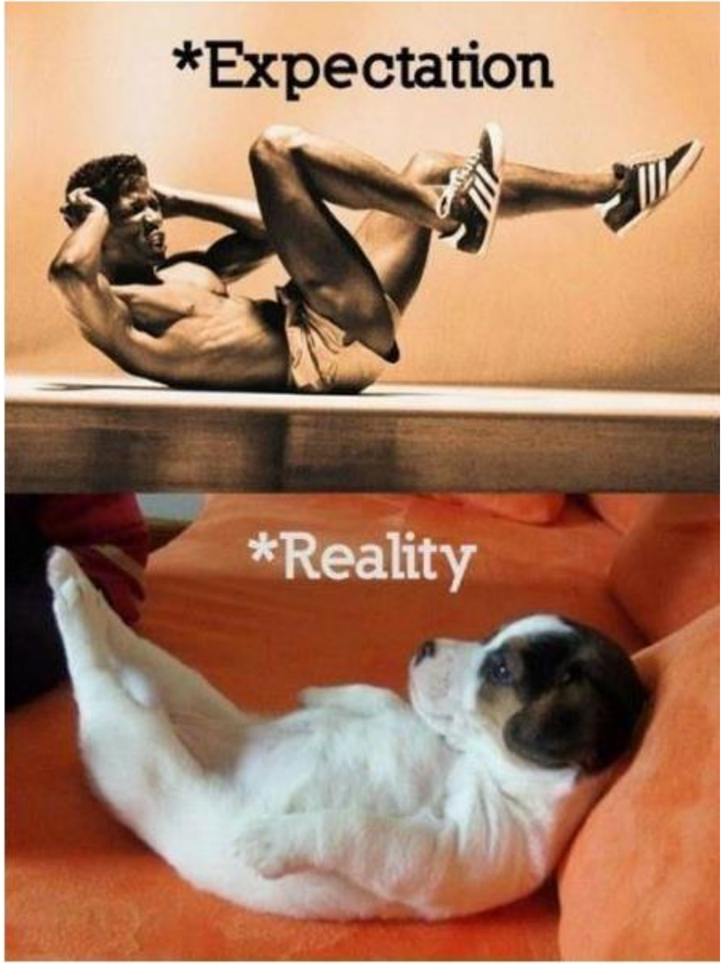
$$grad_V L = -2 \cdot U^T \cdot \big(f(y) - f(U \cdot V \cdot \hat{y})\big) \cdot (\hat{y})^{\mathrm{T}}$$

Solution of the previous problem:

$$L = \|y - U \cdot V \cdot \hat{y}\| \underset{U,V}{\longrightarrow} min$$

$$\begin{cases} U_0 = U \\ V_0 = U \\ U^n = U^{n-1} - \eta_U \cdot \, grad_U L \\ V^n = V^{n-1} - \eta_V \cdot \, grad_V L \end{cases}$$

Layer responses is too heavy!



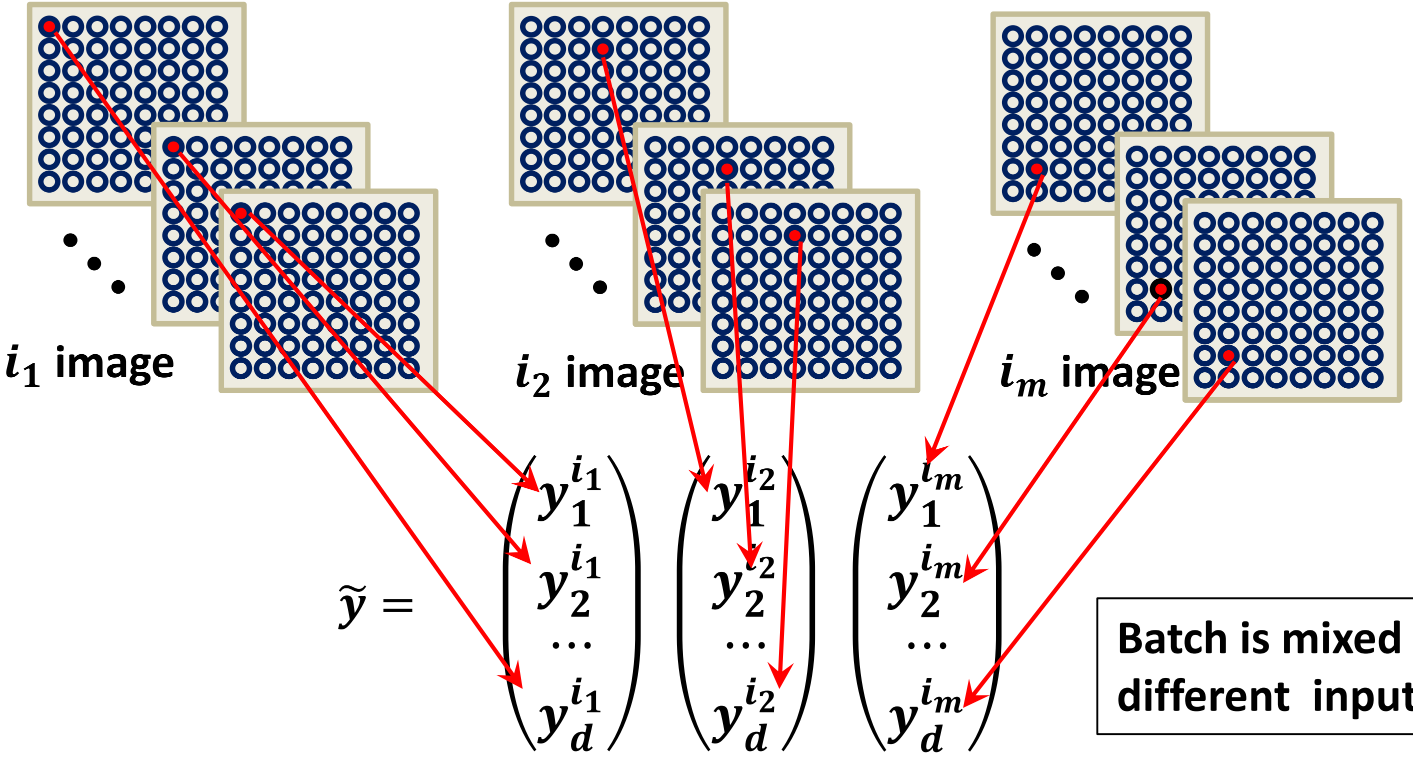
y – responses from 1000 randomly sampled training images 224x224 for 2nd conv layer

$$y \text{ is } \mathbf{64 \times 224 \cdot 224 \cdot 1000 \approx 64 \times 5 \cdot 10^7 = 3.2 \cdot 10^9}$$

Very slow!

Stochastic gradient analogous

$$L = \|f(y_{batch}) - f(U \cdot V \cdot \hat{y}_{batch})\|_2^2 \underset{U,V}{\longrightarrow} min$$



Batch is mixed data from different input!

Algorithm

1) $$L = \|f(y) - f(U \cdot V \cdot \tilde{y})\|_2^2 \xrightarrow[U,V]{} min$$ **Optimization problem**

2) $$\hat{y}_{batch} = \{\hat{y}^{\langle i_1 \rangle}, \dots \hat{y}^{\langle i_m \rangle}\}$$ $$y = \{y^{\langle i_1 \rangle}, \dots y^{\langle i_m \rangle}\}$$ **Take a batch**

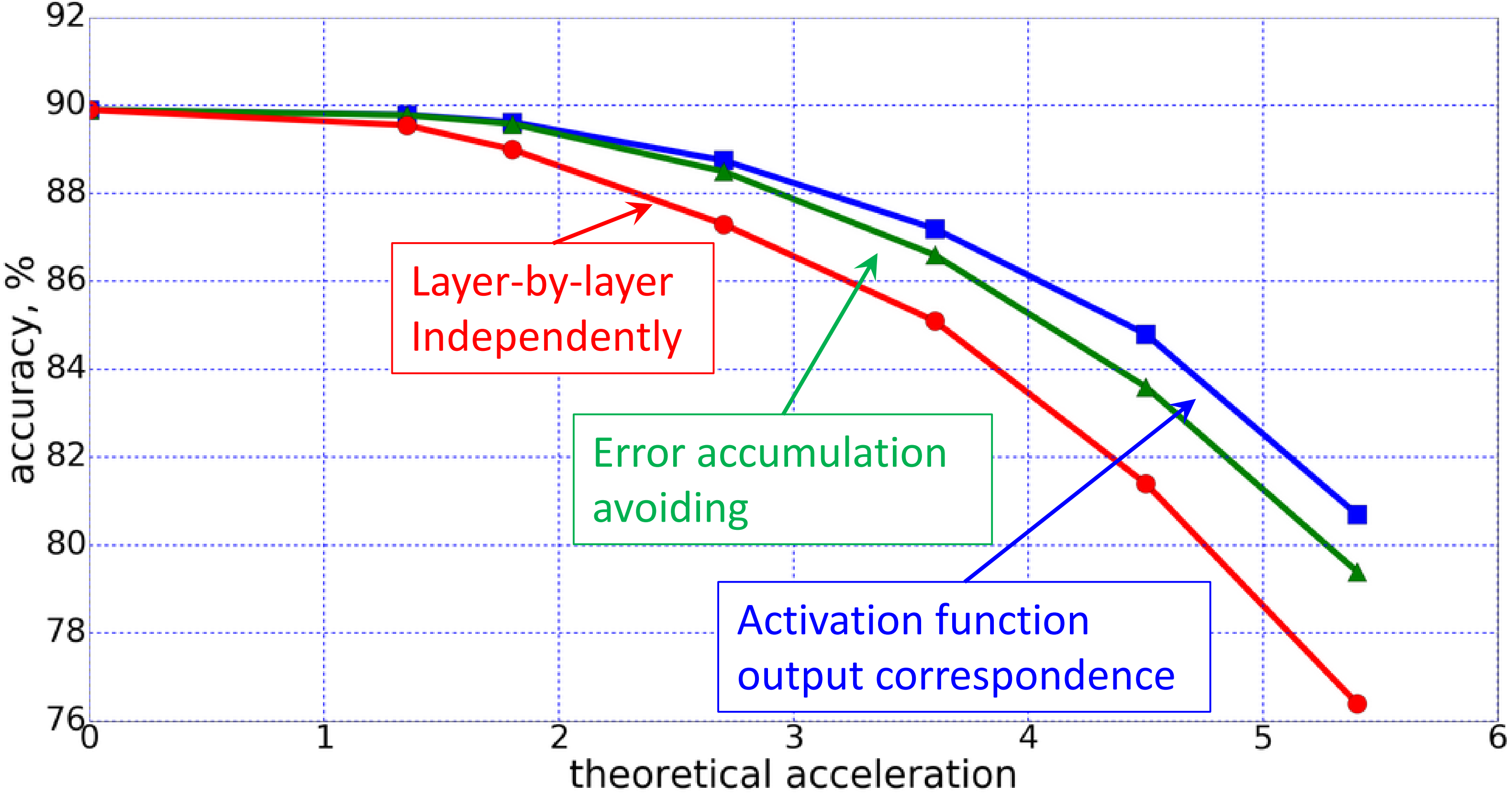
3) $$U_0 = U$$ $$V_0 = U$$ **Initialization**

4) $$v_U^n = \gamma \cdot v_U^n + (1 - \gamma)\, grad_U L$$ $$\eta_U = \eta_V = 10^{-3}, \gamma = 0.9$$ **RSMProp**

5) $$(U^n)_i = \left(U^{n-1}\right)_i - \frac{\eta_U}{\sqrt{(v_U^n)_i}} \cdot (grad_U L)_i$$ $$(V^n)_i = \left(V^{n-1}\right)_i - \frac{\eta_V}{\sqrt{(v_V^n)_i}} \cdot (grad_V L)_i$$ **Gradient descent**

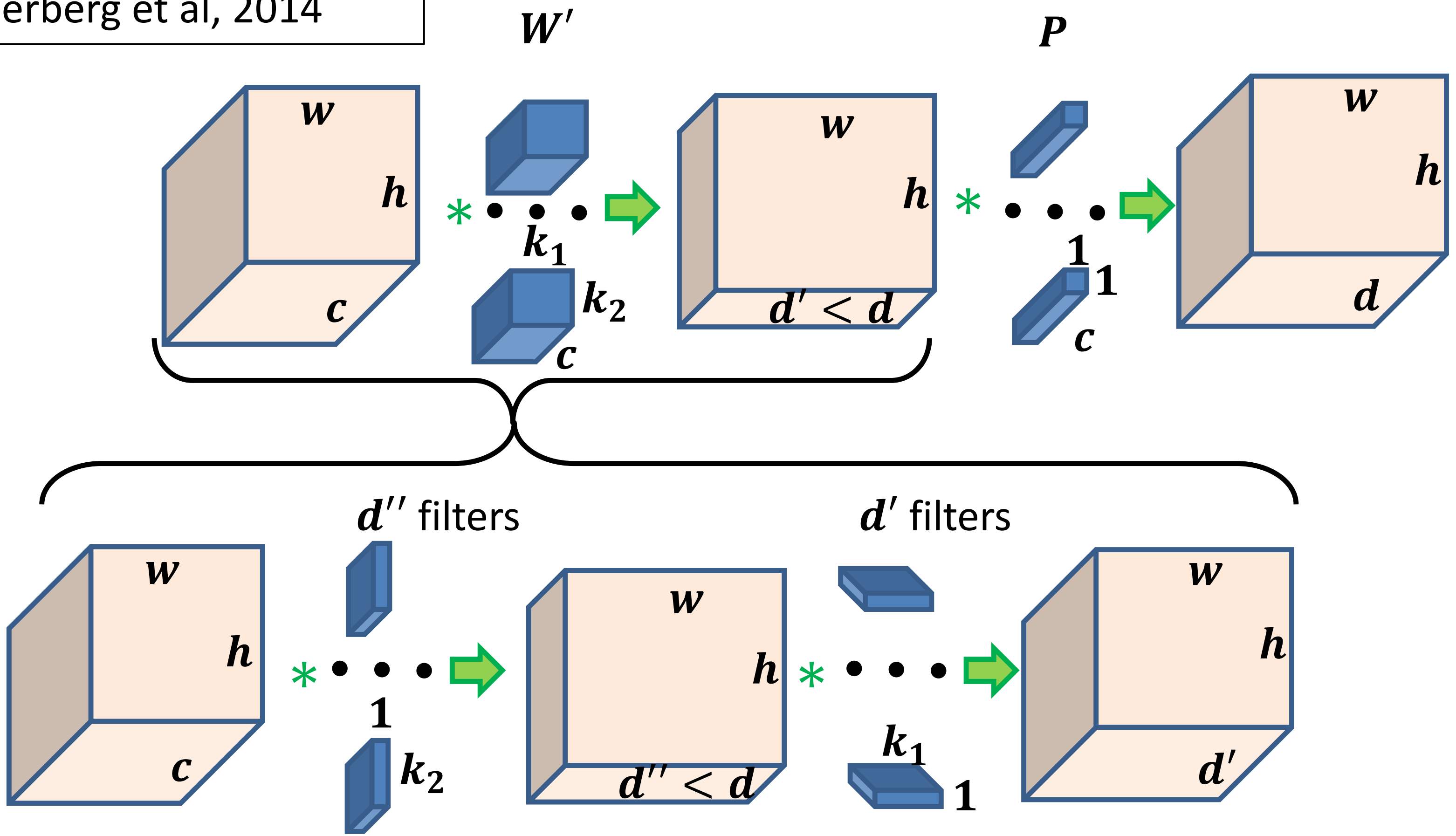
Results

Top-5 error on ILSVRC-2012 (ImageNet) validation dataset (50K images)

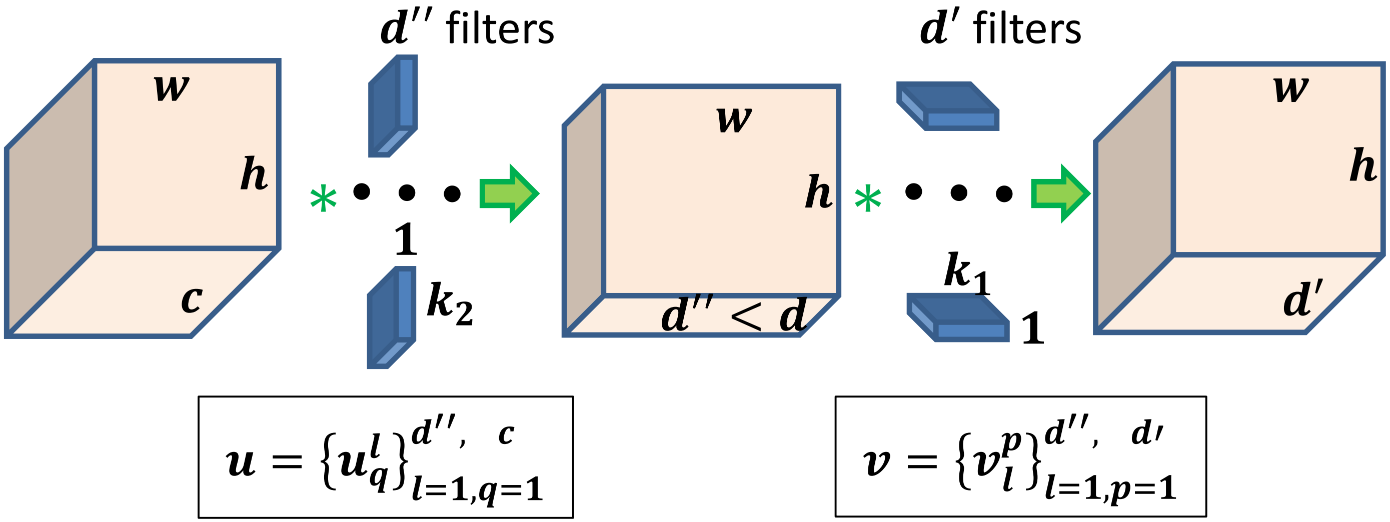


Horizontal and vertical filters

Jaderberg et al, 2014



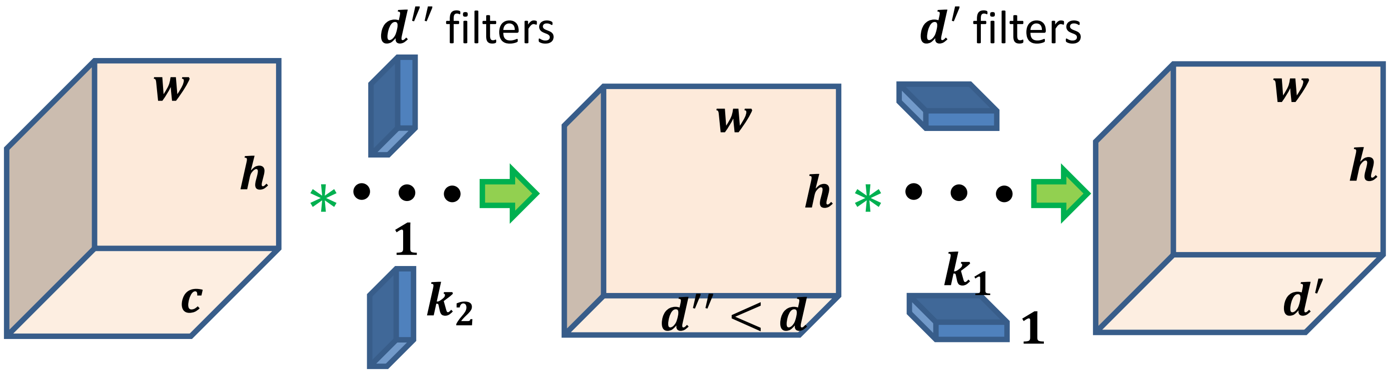
Horizontal and vertical filters



Horizontal and vertical filters

Jaderberg et al, 2014

$$\sum_{q=1}^{c}\sum_{p=1}^{d'}\left\|W_q^p - \sum_{l=1}^{d''} u_q^l * v_l^p\right\|_2^2 \xrightarrow[\{v_l^p\},\{u_q^l\}]{} \min$$



$$u = \{u_q^l\}_{l=1,q=1}^{d'',\ c}$$

$$v = \{v_l^p\}_{l=1,p=1}^{d'',\ d'}$$

Horizontal and vertical filters

Jaderberg et al, 2014

$$L = \sum_{q=1}^{c} \sum_{p=1}^{d\prime} \left\| W_q^p - \sum_{l=1}^{d''} u_q^l * v_l^p \right\|_2^2 \xrightarrow[\{v_l^p\},\{u_q^l\}]{} min$$

$$grad_u L = \sum_{q=1}^{c} \left(W_q^p - v^p \cdot \left(u_q\right)^T \right) \cdot u_q$$

$$grad_v L = \sum_{q=1}^{c} \left(W_q^p - v^p \cdot \left(u_q\right)^T \right)^T \cdot v^p$$

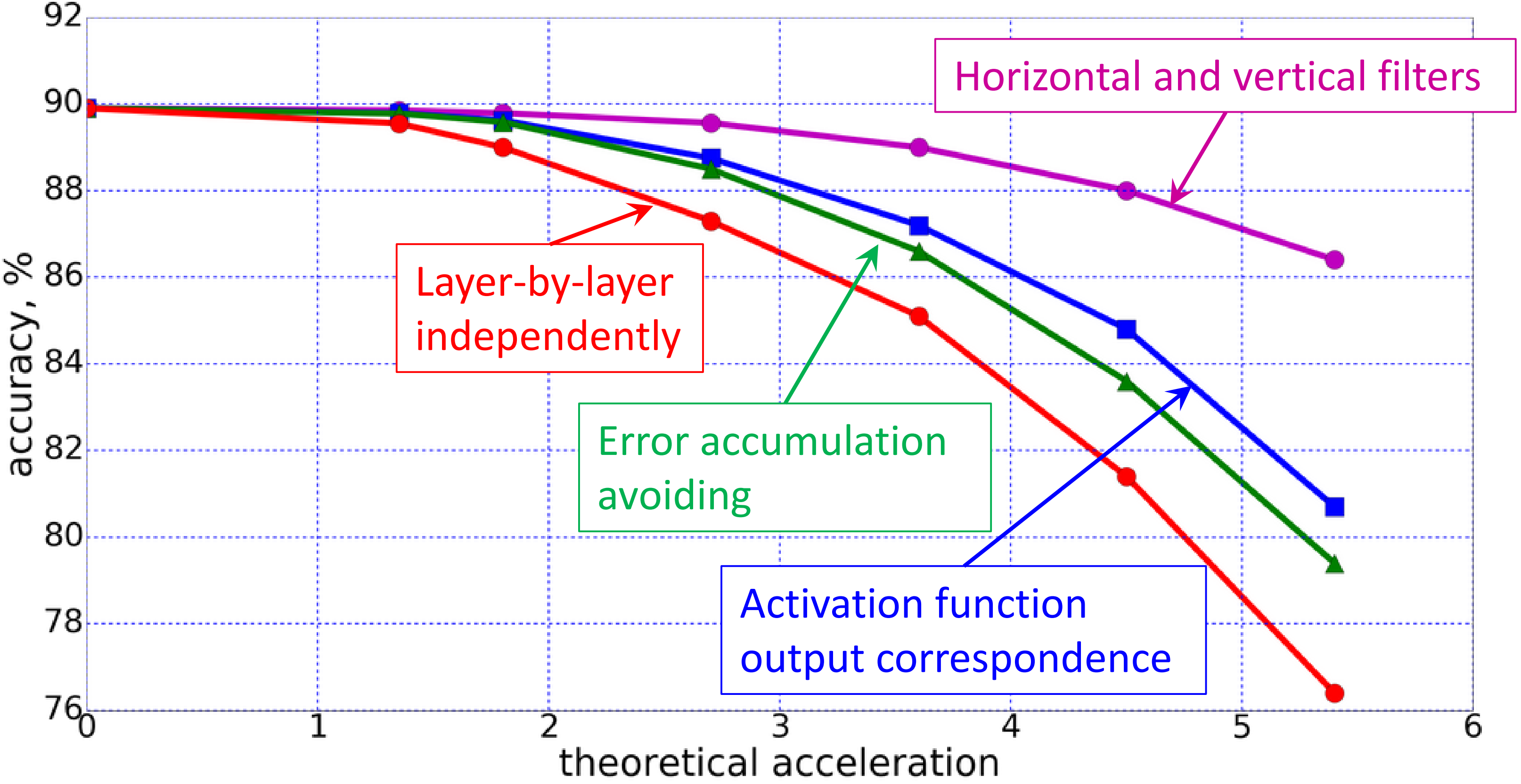
RMSProp

$u_q = \left\{ u_q^1, \cdots, u_q^{d''} \right\} - k_2 \times d''$ matrix

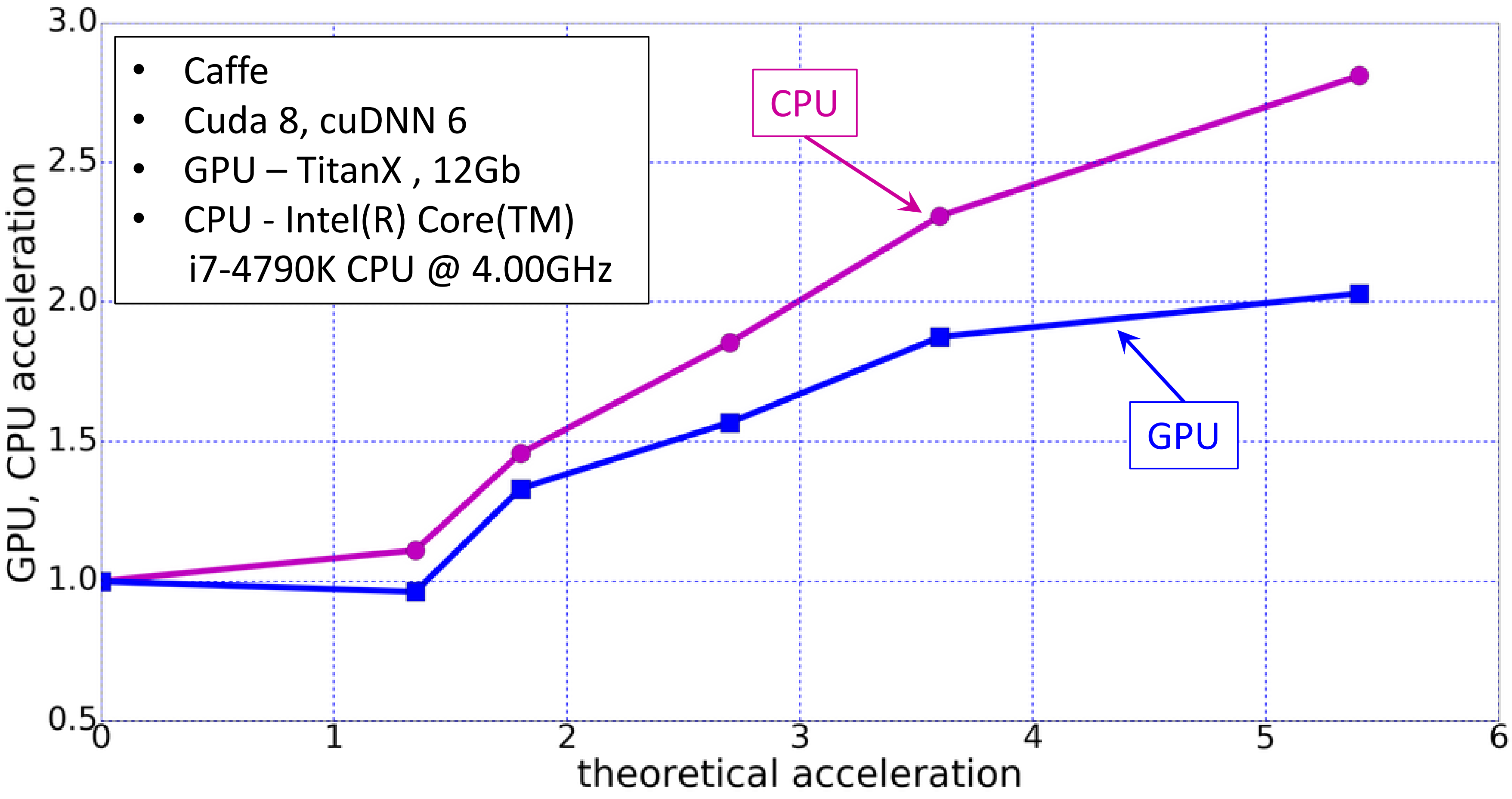
$v^p = \left\{ v_1^p, \cdots, v_{d''}^p \right\} - k_1 \times d''$ matrix

Results

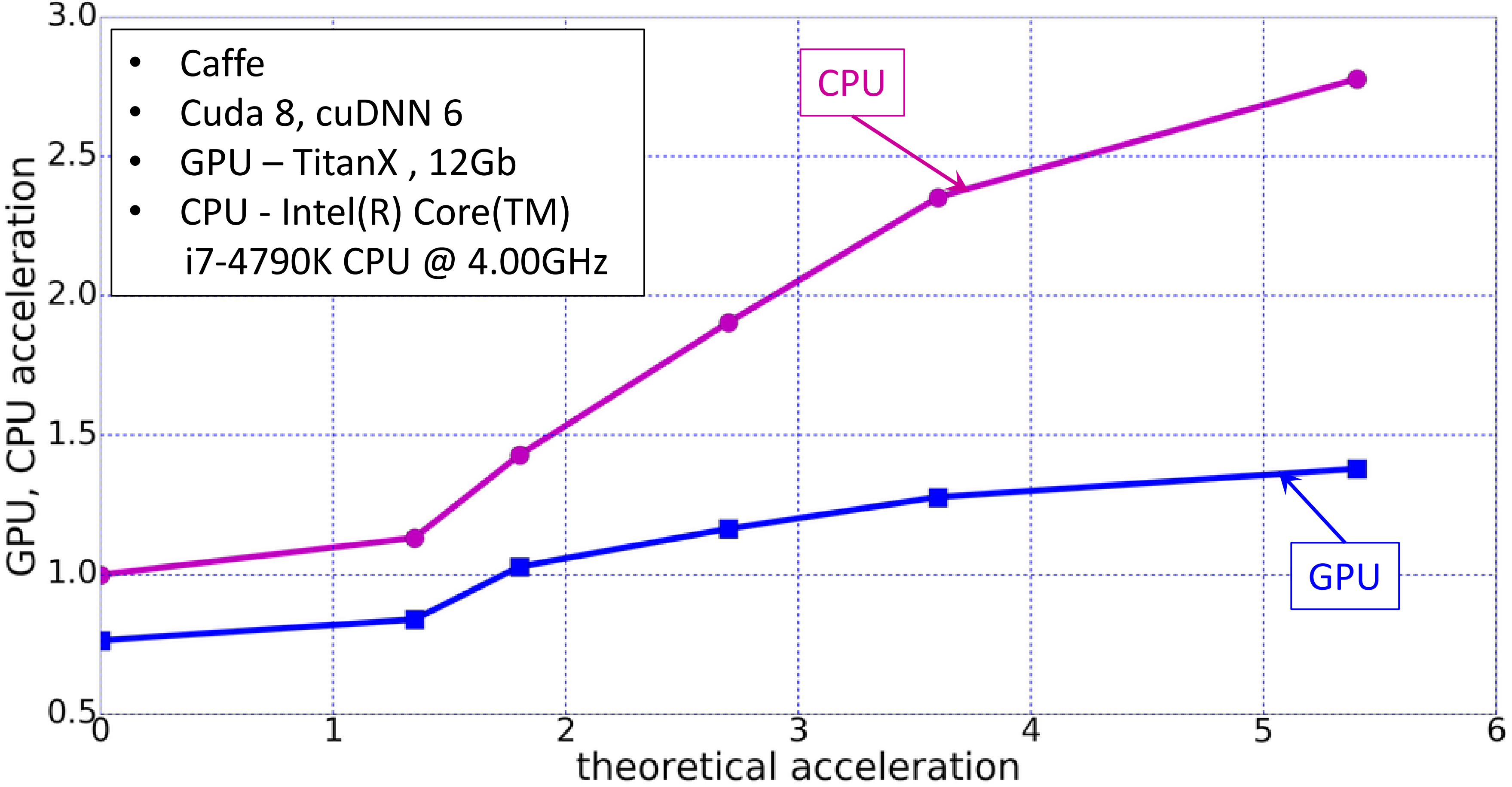
Top-5 error on ILSVRC-2012 (ImageNet) validation dataset (50K images)



Theoretical, CPU, GPU acceleration for 3x3 + 1x1 separation



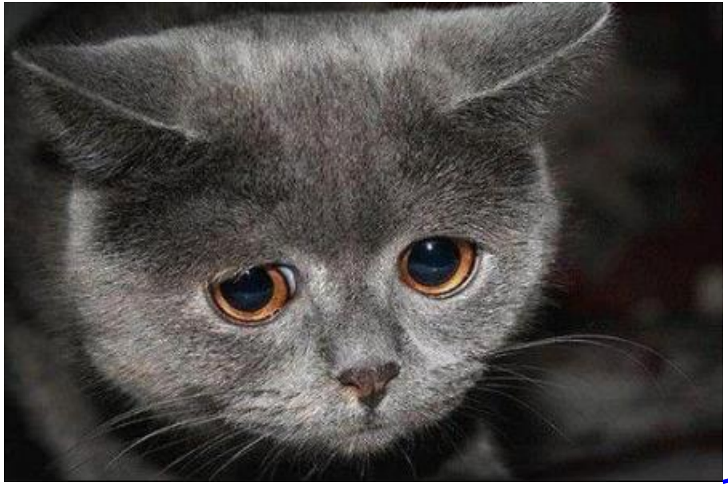
Theoretical, CPU, GPU acceleration for 3x1 + 3x1 + 1x1 separation

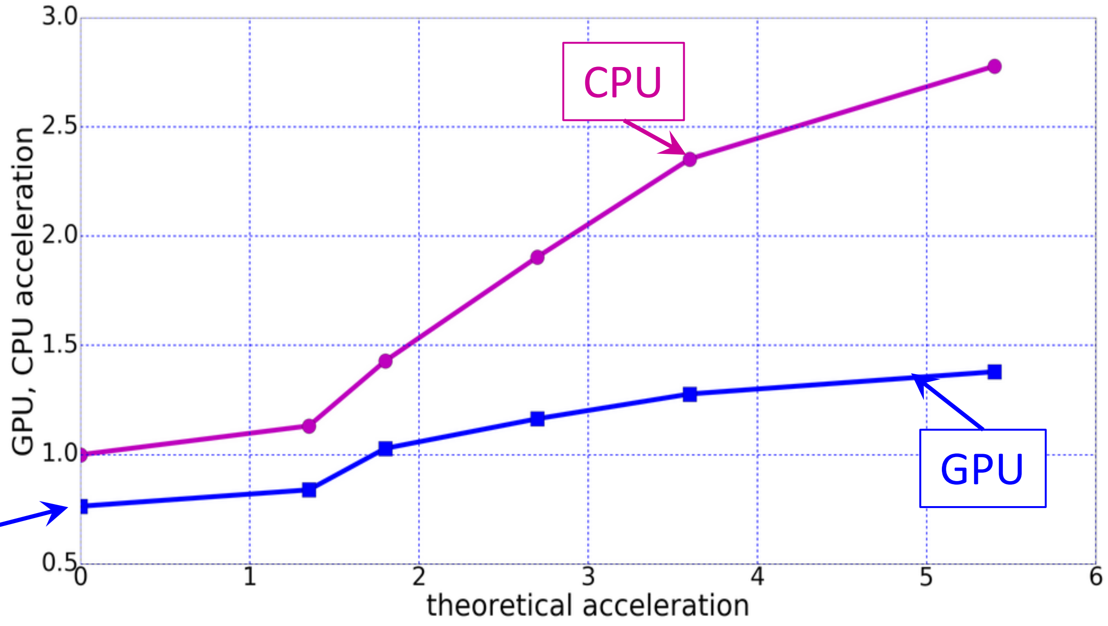


Theoretical, CPU, GPU acceleration for 3x1 + 3x1 + 1x1 separation

GPU very bad performance reasons:

- $1 \times d$ and $d \times 1$ layers are not optimized in cuDNN
- Difference between 1×1 and $d \times d$ layers performance is not at $\times 9$ times , but we optimize only 3×3 layers
- Huge layers is better parallelized on GPU then light ones.



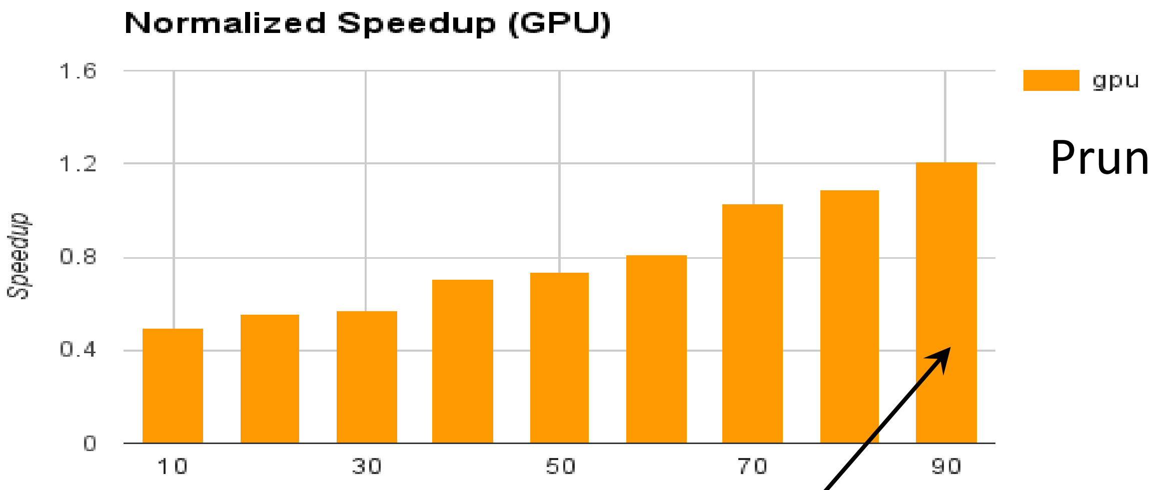


Conclusions

- It is better to avoid CNN learning during approximation process
- Output feature maps are highly correlated
- Take into account approximation not only separate layer but also the whole model too.
- It is better to minimize difference with non-linear responses
- It is easy to obtain good approximation of square filter by horizontal and vertical filters
- It is enough to obtain rule for output feature maps basis only for 1% of the training dataset (ImageNet) to interpolate it for the whole dataset
- CuDNN does not optimize dx1 and 1xd filters
- Low rank approximation is good approach for CPU (for single kernel is much better)

Neural Network Pruning results

SpeedUp of TensorFlow pruning for ConvNet on MNIST



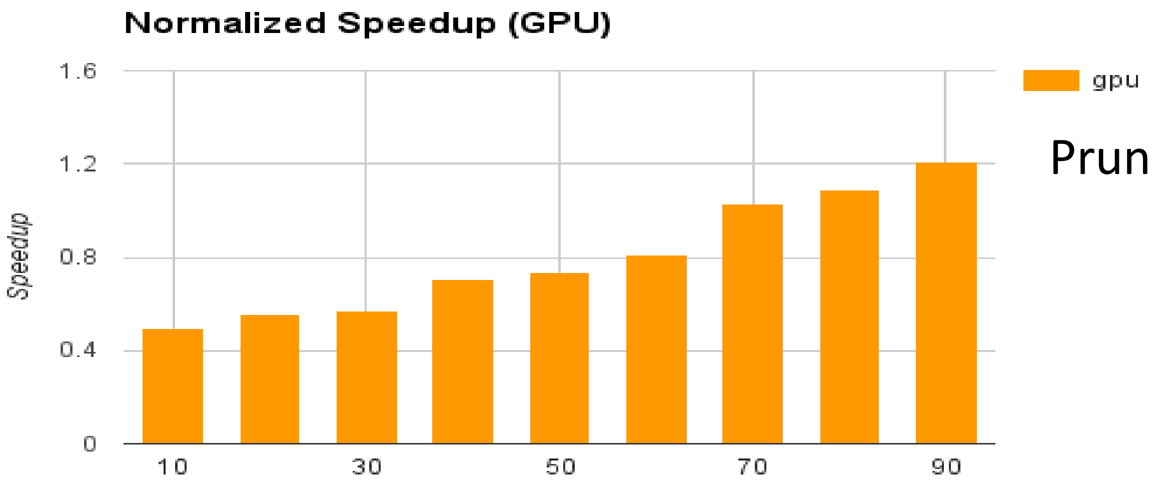
Pruning is only for compression!
It accelerates only
fully-connected layers

1% loss of accuracy

Sparse matrix is a big problem!

Neural Network Pruning results

SpeedUp of TensorFlow pruning for ConvNet on MNIST



Pruning is only for compression!
It accelerates only
fully-connected layers

Song Han papers (mainly conference) and his fantastic 4x acceleration with improved accuracy