Practical Tricks for CNNs

Neural Networks Design And Application

Practical tricks

- Batch normalization and local response normalization
- Data augmentation
- Dropout
- Regularization/weight decay
- Pre-train
- Stagewise training

Rescaling images

```
# calculate mu and sig using the training set
d = x_train.shape[1]
mu = numpy.mean(x_train, axis=0).reshape(1, d)
sig = numpy.std(x_train, axis=0).reshape(1, d)

# transform the training features
x_train = (x_train - mu) / (sig + 1E-6)

# transform the test features
x_test = (x_test - mu) / (sig + 1E-6)
```

Rescaling images

```
# calculate mu and sig using the training set
d = x_train.shape[1]
mu = numpy.mean(x_train, axis=0).reshape(1, d)
sig = numpy.std(x_train, axis=0).reshape(1, d)

# transform the training features
x_train = (x_train - mu) / (sig + 1E-6)

# transform the test features
x_test = (x_test - mu) / (sig + 1E-6)
```

Rescaling images

```
# calculate mu and sig using the training set
d = x_train.shape[1]
mu = numpy.mean(x_train, axis=0).reshape(1, d)
sig = numpy.std(x_train, axis=0).reshape(1, d)

# transform the training features
x_train = (x_train - mu) / (sig + 1E-6)

# transform the test features
x_test = (x_test - mu) / (sig + 1E-6)
```

Centering images

Rescaling images

```
# calculate mu and sig using the training set
d = x_train.shape[1]
mu = numpy.mean(x_train, axis=0).reshape(1, d)
sig = numpy.std(x_train, axis=0).reshape(1, d)

# transform the training features
x_train = (x_train - mu) / (sig + 1E-6)

# transform the test features
x_test = (x_test - mu) / (sig + 1E-6)
```

Centering images   Standardize images

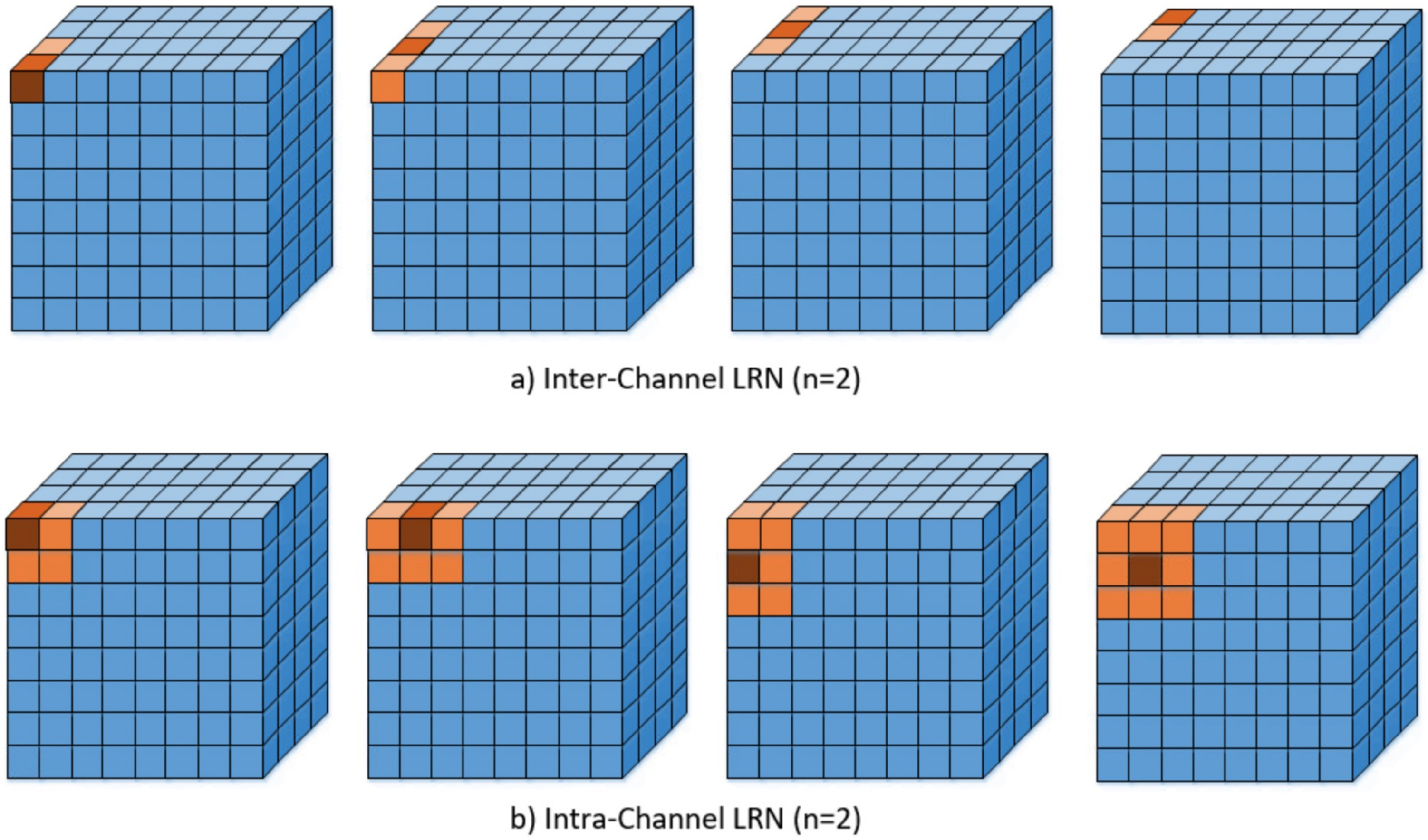
Rescaling images

```
# calculate mu and sig using the training set
d = x_train.shape[1]
mu = numpy.mean(x_train, axis=0).reshape(1, d)
sig = numpy.std(x_train, axis=0).reshape(1, d)

# transform the training features
x_train = (x_train - mu) / (sig + 1E-6)

# transform the test features
x_test = (x_test - mu) / (sig + 1E-6)
```

Centering images   Standardize images

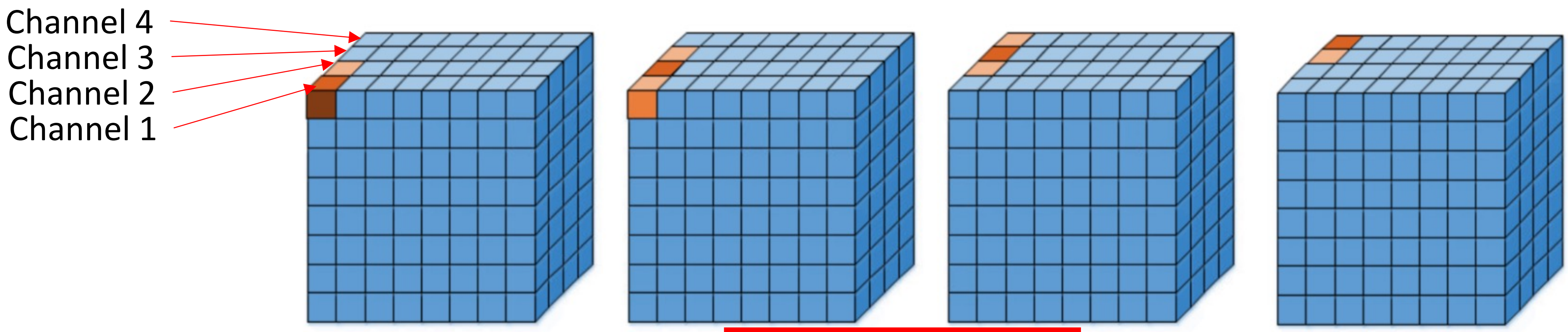
To bound the values of data

Local response normalization



https://towardsdatascience.com/difference-between-local-response-normalization-and-batch-normalization-272308c034ac

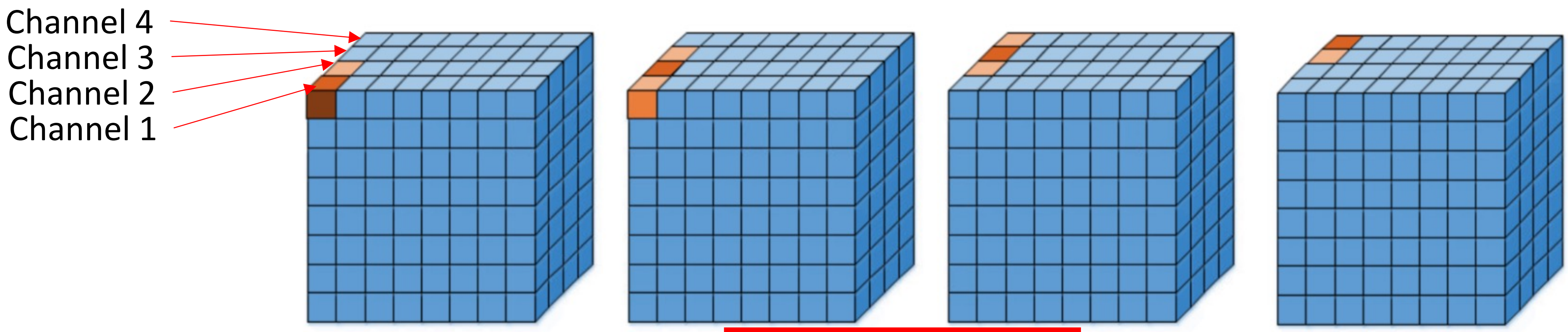
Local response normalization



a) Inter-Channel LRN (n=2)

https://towardsdatascience.com/difference-between-local-response-normalization-and-batch-normalization-272308c034ac

Local response normalization

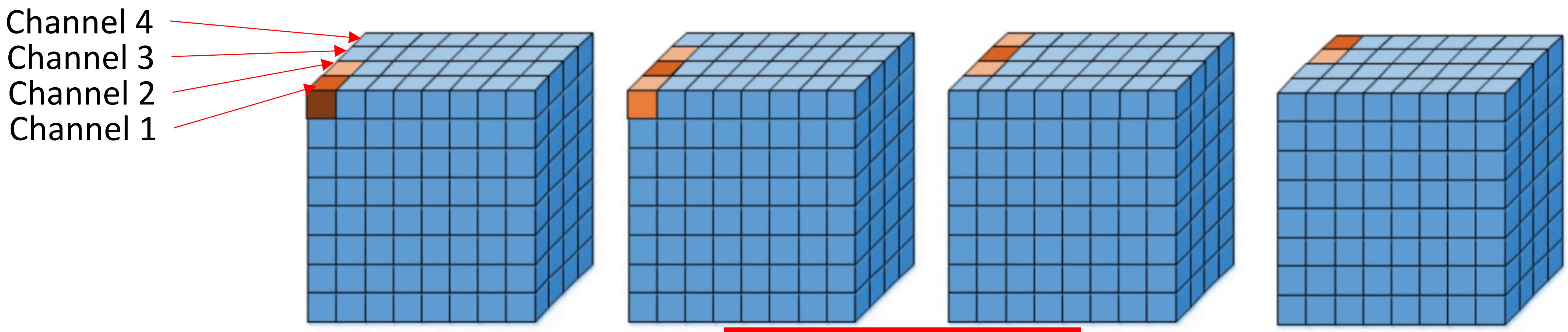


a) Inter-Channel LRN (n=2)

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0, i-n/2)}^{\min(N-1, i+n/2)} (a^j_{x,y})^2 \right)^\beta$$

https://towardsdatascience.com/difference-between-local-response-normalization-and-batch-normalization-272308c034ac

Local response normalization



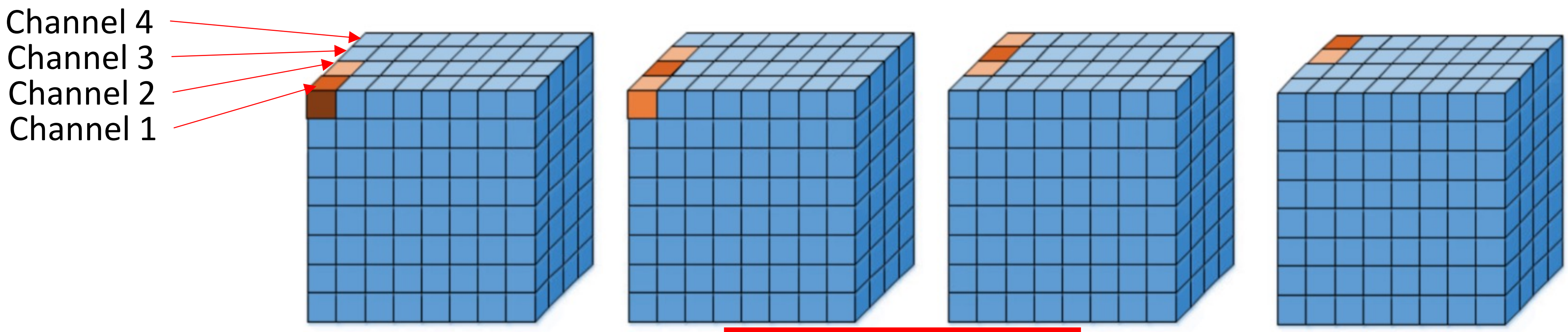
a) Inter-Channel LRN (n=2)

$$\boxed{b^i_{x,y}} = \boxed{a^i_{x,y}} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2\right)^\beta$$

Output value   Original value

https://towardsdatascience.com/difference-between-local-response-normalization-and-batch-normalization-272308c034ac

Local response normalization



a) Inter-Channel LRN (n=2)

$$\boxed{b^i_{x,y}} = \boxed{a^i_{x,y}} / \left(k + \alpha \sum_{j=\max(0,\boxed{i-n/2})}^{\min(N-1,\boxed{i+n/2})} (a^j_{x,y})^2 \right)^{\beta}$$

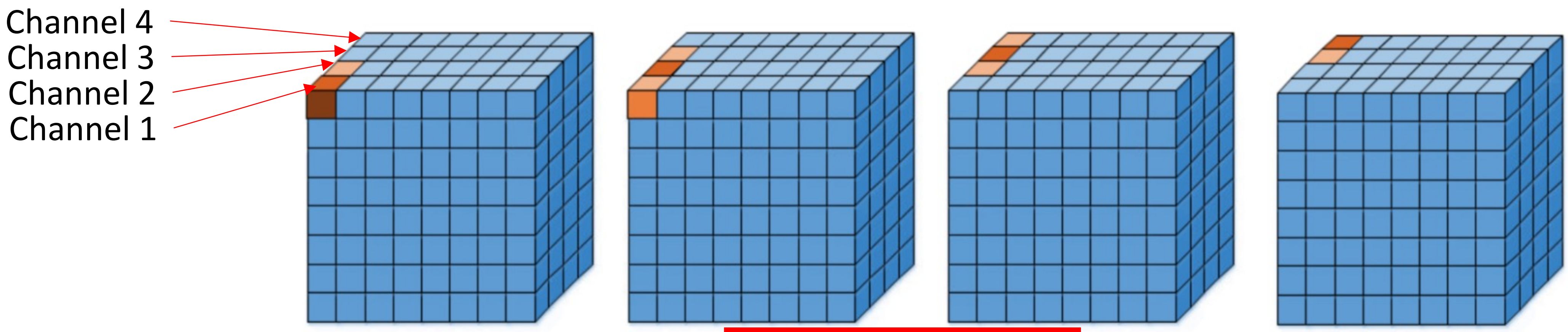
Surrounding elements

Output value

Original value

https://towardsdatascience.com/difference-between-local-response-normalization-and-batch-normalization-272308c034ac

Local response normalization



a) Inter-Channel LRN (n=2)

$$\boxed{b^i_{x,y}} = \boxed{a^i_{x,y}} \Big/ \left(k + \alpha \sum_{j=\max(0,\boxed{i-n/2})}^{\min(N-1,\boxed{i+n/2})} \boxed{(a^j_{x,y})^2}\right)^{\beta}$$

Squared values of surrounding elements

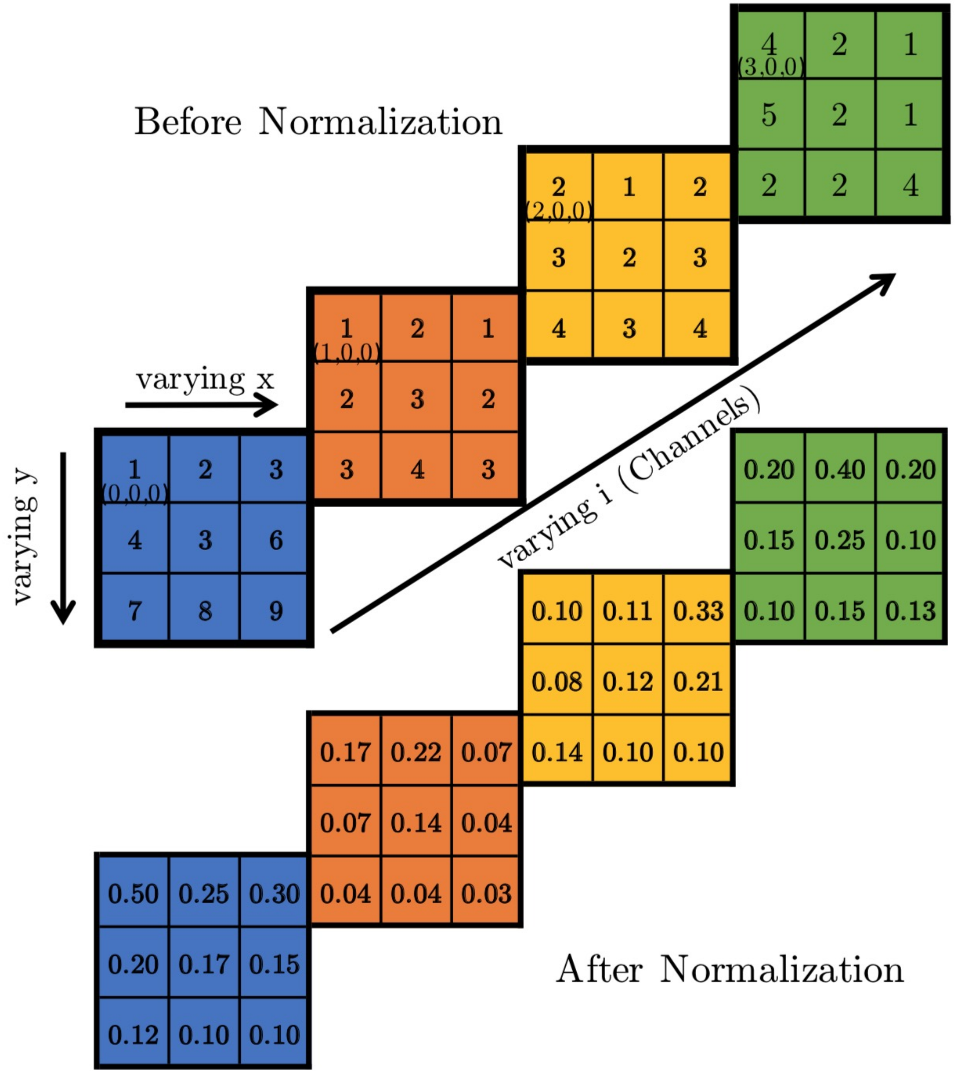
Surrounding elements

Output value

Original value

https://towardsdatascience.com/difference-between-local-response-normalization-and-batch-normalization-272308c034ac

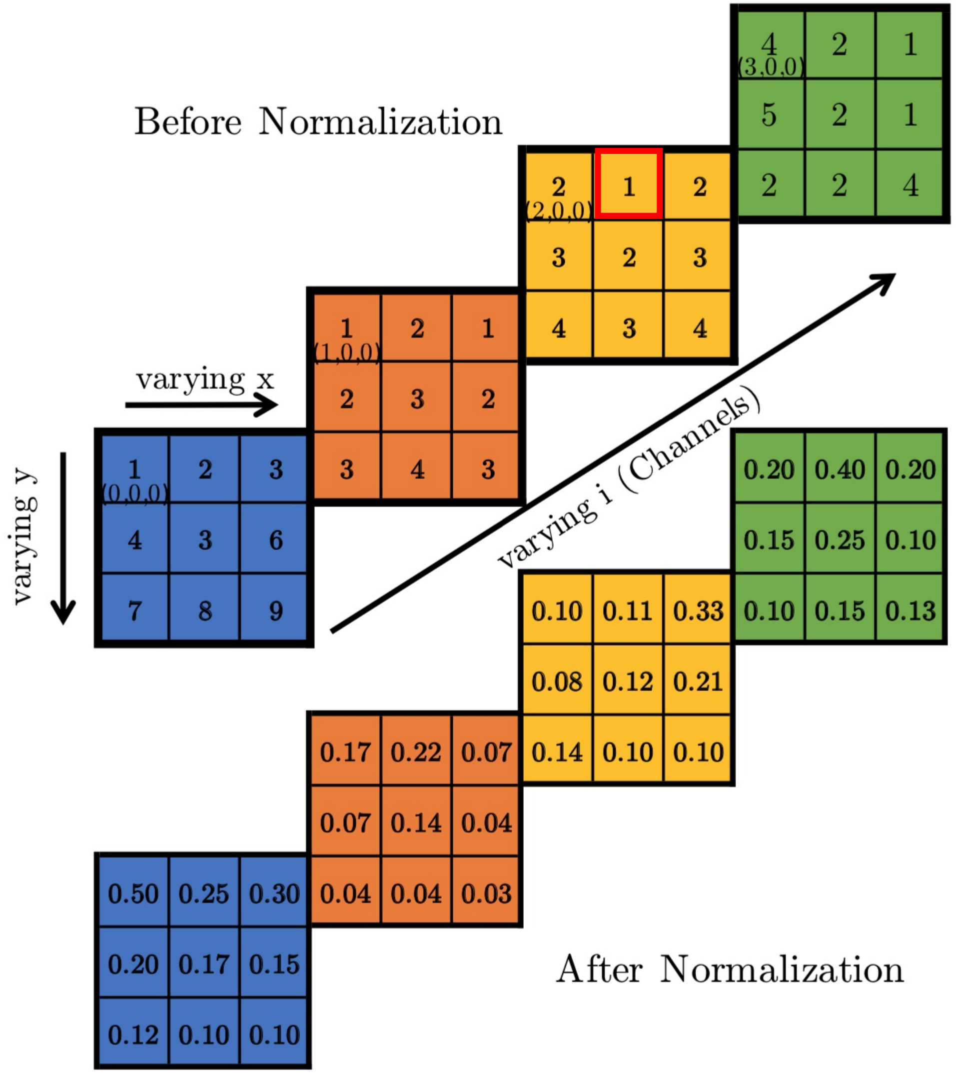
Local response normalization



N=2, K=0, $\alpha = 1, \beta = 1$

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}$$

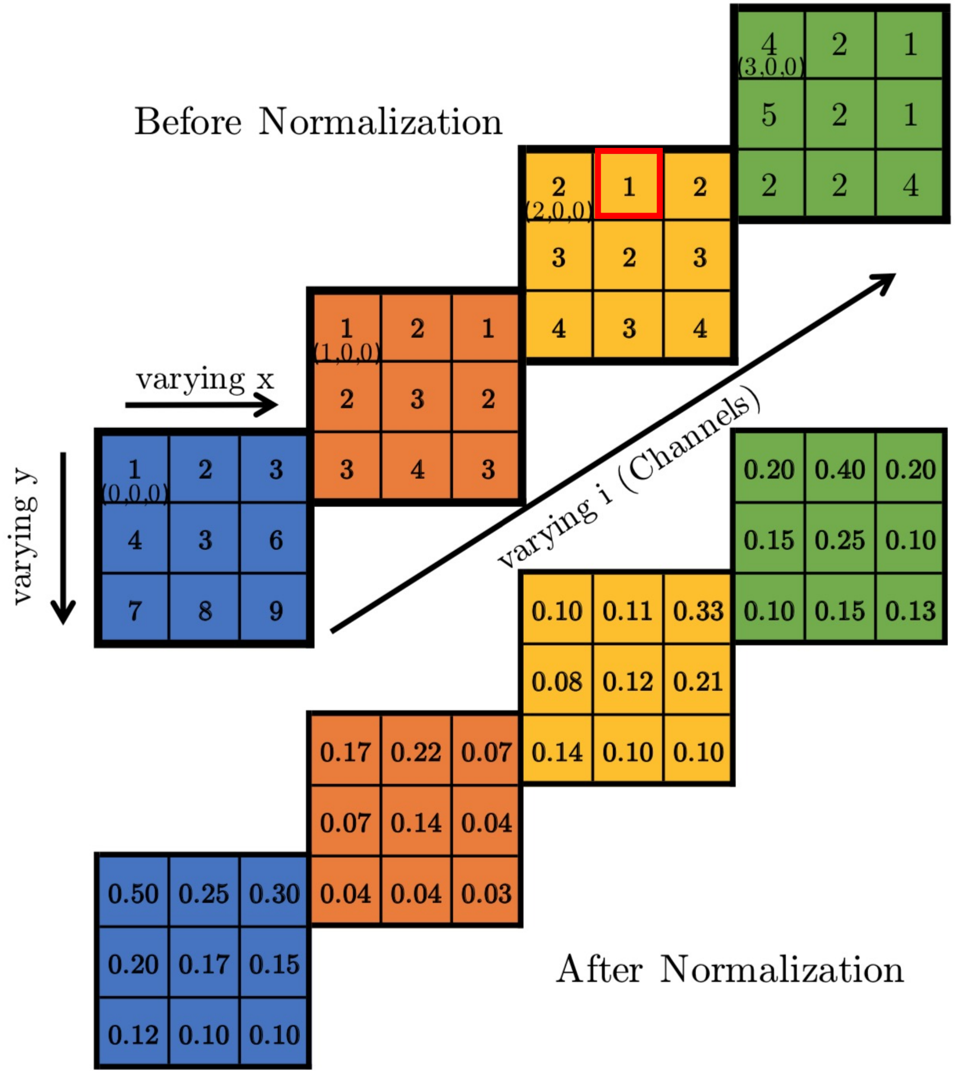
Local response normalization



N=2, K=0, $\alpha = 1, \beta = 1$

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0, i-n/2)}^{\min(N-1, i+n/2)} (a^j_{x,y})^2 \right)^\beta$$

Local response normalization

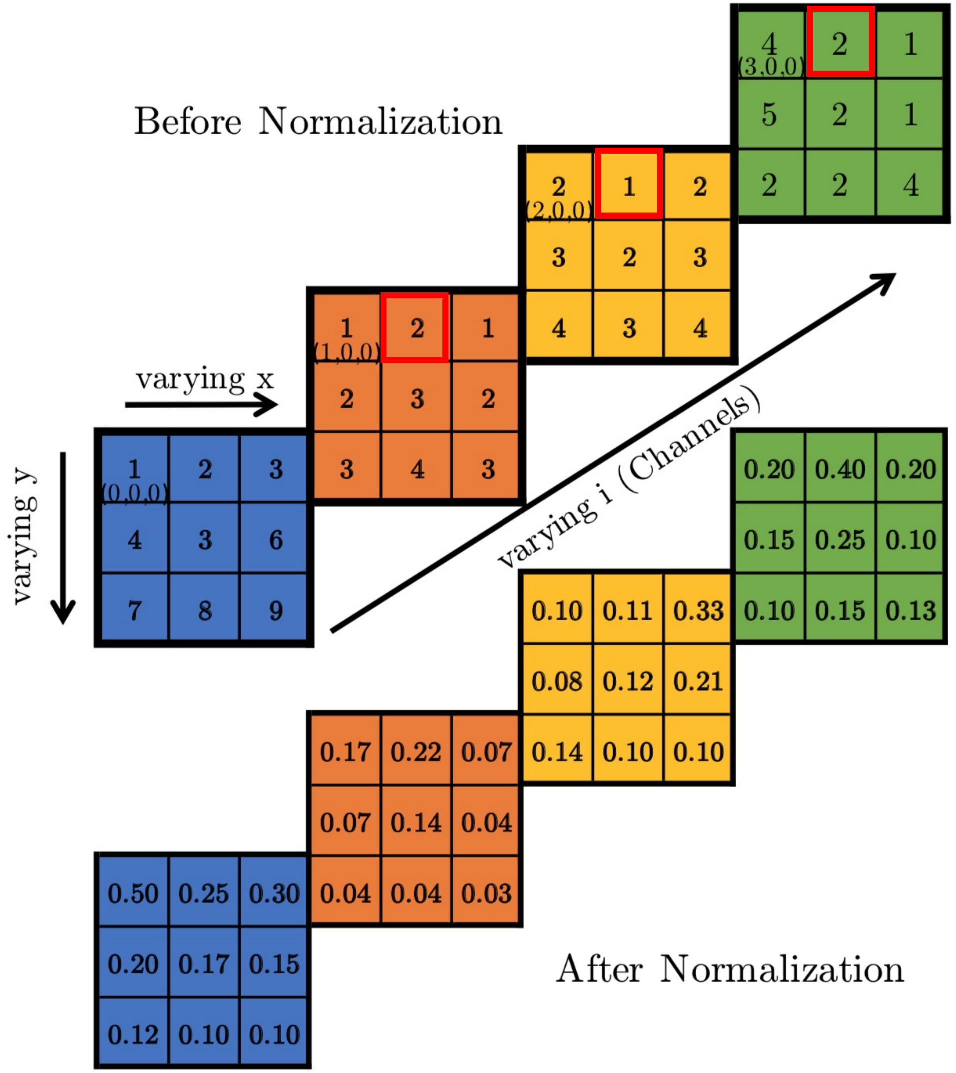


N=2, K=0, $\alpha = 1, \beta = 1$

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}$$

1/(?) = ?

Local response normalization

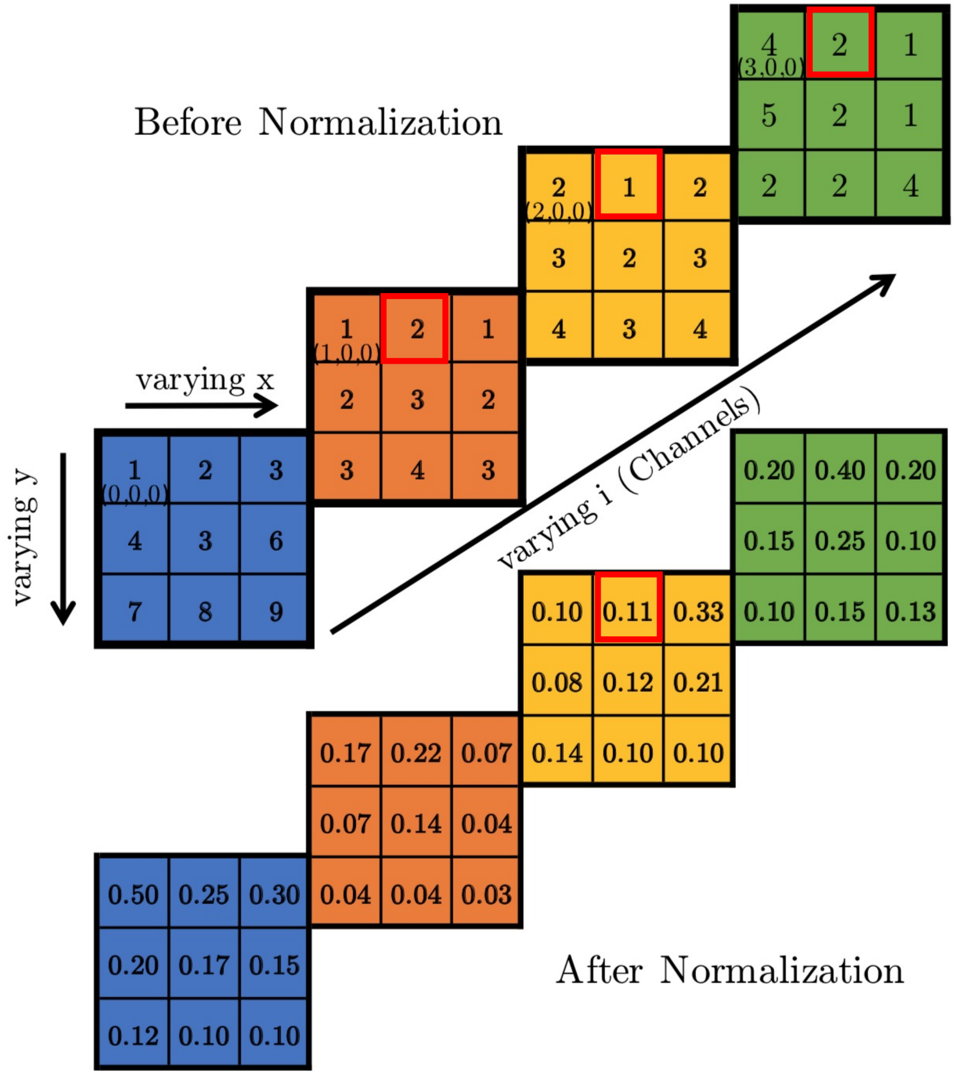


N=2, K=0, $\alpha = 1, \beta = 1$

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}$$

1/(0 + 4 + 1 + 4)=?

Local response normalization

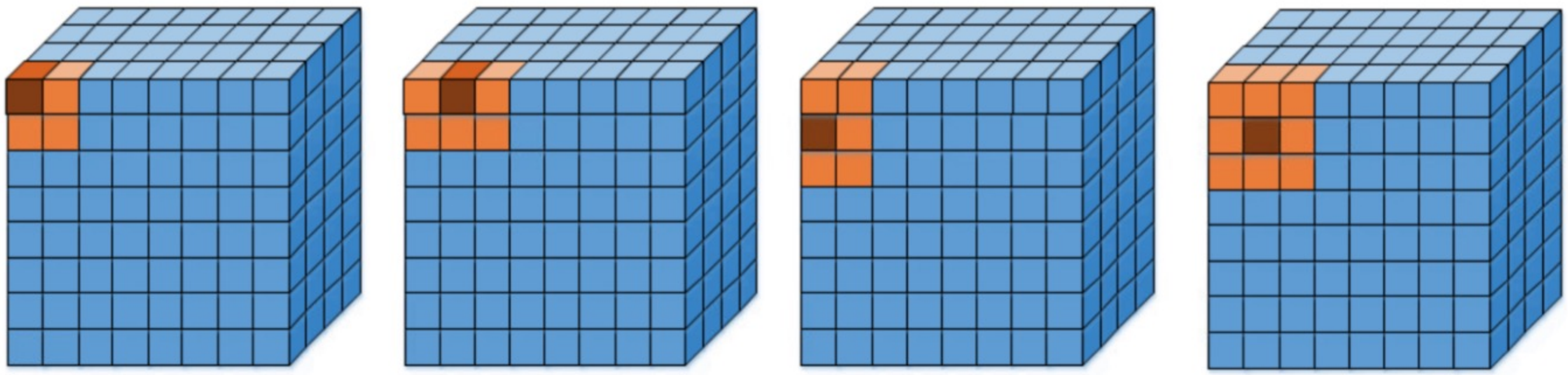


N=2, K=0, $\alpha = 1, \beta = 1$

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^\beta$$

1/(0 + 4 + 1 + 4)=1/9

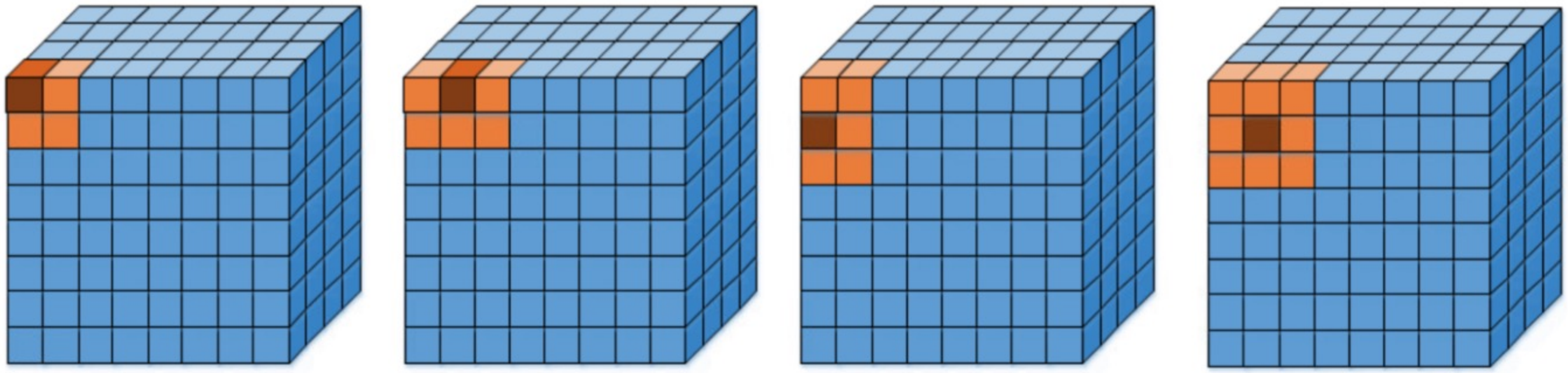
Local response normalization



b) Intra-Channel LRN (n=2)

$$b^k_{x,y} = a^k_{x,y} / \left(k + \alpha \sum_{i=max(0,x-n/2)}^{min(W,x+n/2)} \sum_{j=max(0,y-n/2)}^{min(H,y+n/2)} (a^k_{i,j})^2 \right)^{\beta}$$

Local response normalization

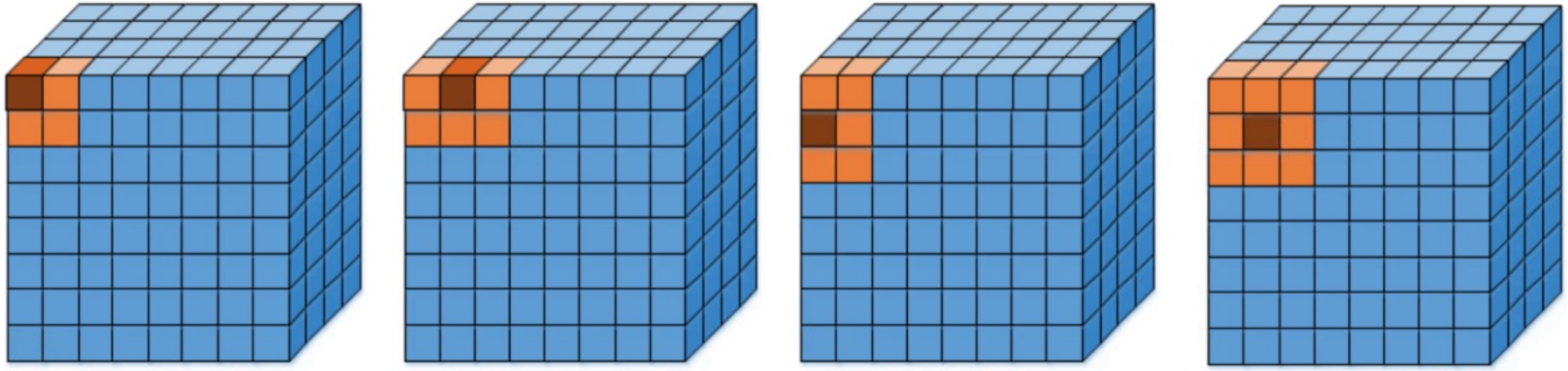


b) Intra-Channel LRN (n=2)

$$b_{x,y}^{k} = a_{x,y}^{k} / \left(k + \alpha \sum_{i=max(0,x-n/2)}^{min(W,x+n/2)} \sum_{j=max(0,y-n/2)}^{min(H,y+n/2)} (a_{i,j}^{k})^2 \right)^{\beta}$$

Inter-channel: $$b_{x,y}^{i} = a_{x,y}^{i} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a_{x,y}^{j})^2 \right)^{\beta}$$

Local response normalization

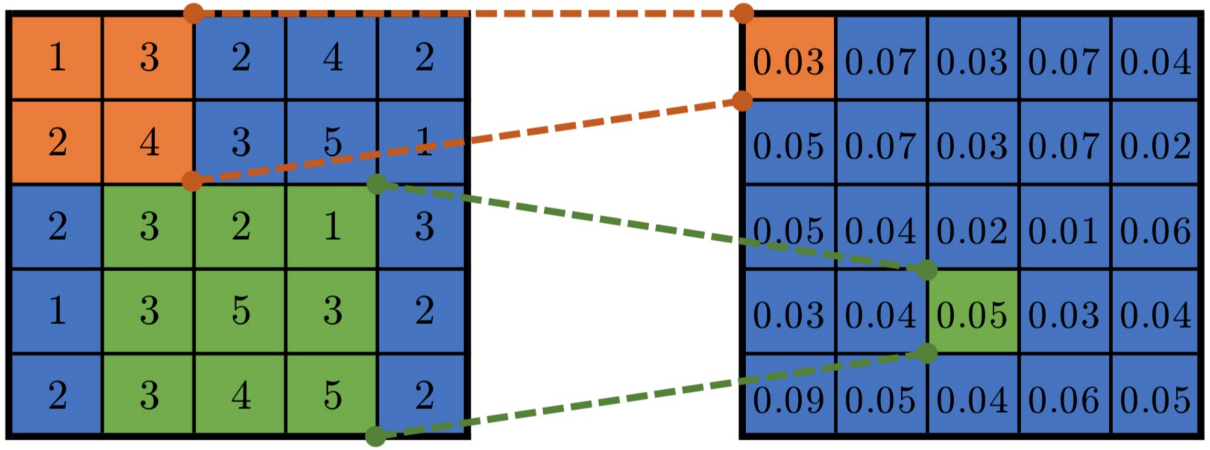


b) Intra-Channel LRN (n=2)

$$b^k_{x,y} = a^k_{x,y} / \left(k + \alpha \boxed{\sum_{i=max(0,x-n/2)}^{min(W,x+n/2)} \sum_{j=max(0,y-n/2)}^{min(H,y+n/2)}} (a^k_{i,j})^2 \right)^{\beta}$$

Inter-channel: $$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \boxed{\sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)}} (a^j_{x,y})^2 \right)^{\beta}$$

Local response normalization



Before Normalization **After Normalization**

$$b^k_{x,y} = a^k_{x,y} / \left(k + \alpha \boxed{\sum_{i=max(0,x-n/2)}^{min(W,x+n/2)} \sum_{j=max(0,y-n/2)}^{min(H,y+n/2)}} (a^k_{i,j})^2 \right)^{\beta}$$

Batch normalization [BN]

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Batch normalization [BN]

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1 \ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Batch normalization [BN]

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Rescaling for a batch

Batch normalization [BN]

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Rescaling for a batch

Batch normalization [BN]

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1 \ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

Rescaling for a batch

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

A linear model as output

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Batch normalization [BN]

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

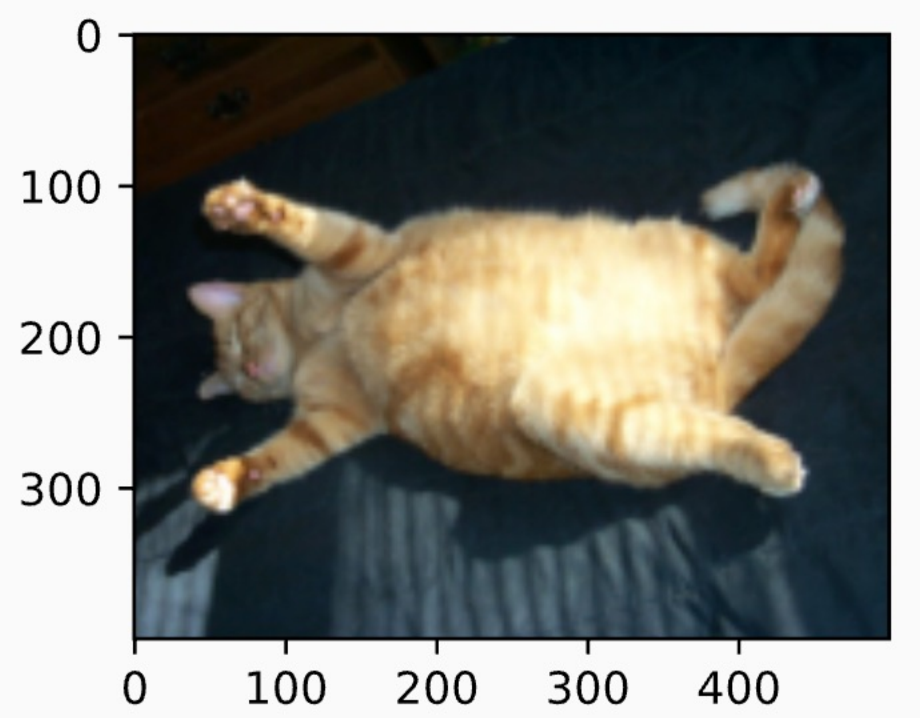
Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

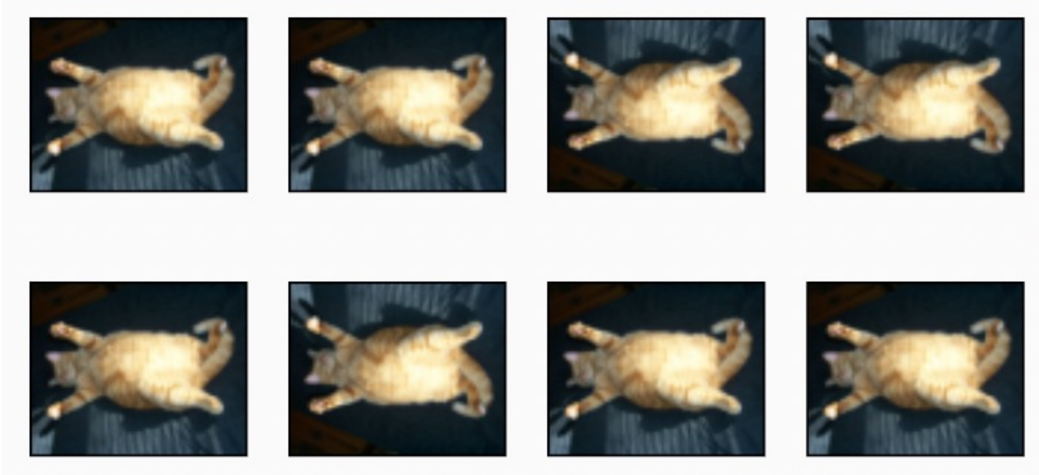
Rescaling for a batch

A linear model as output:
There are two learnable parameters

Data augmentation

- Increase the amount of data by:
  - Adding slightly modified copies of already existing data, or
  - Newly created synthetic data from existing data

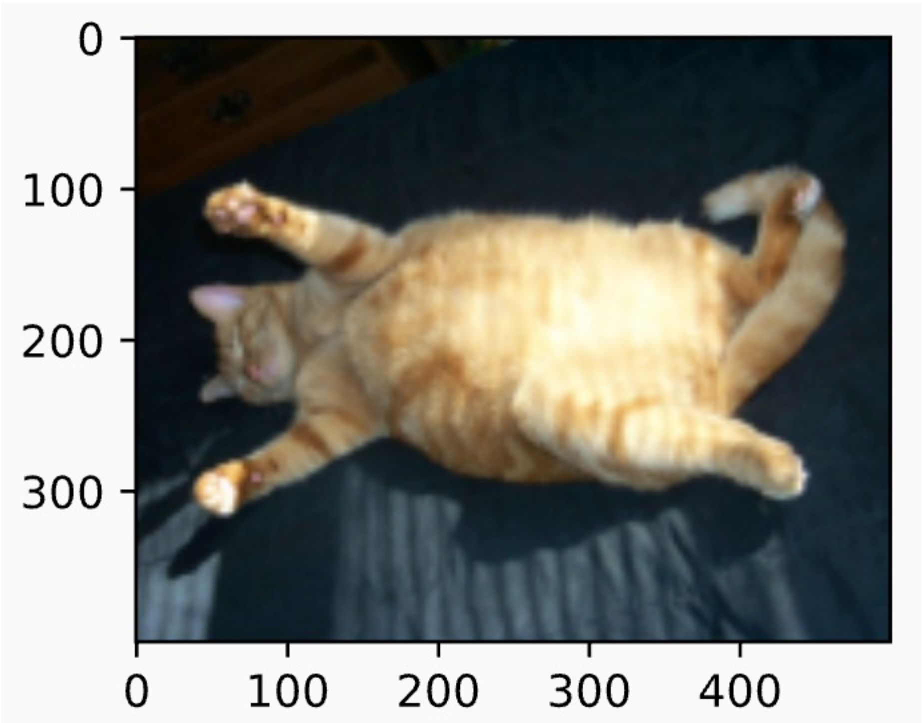


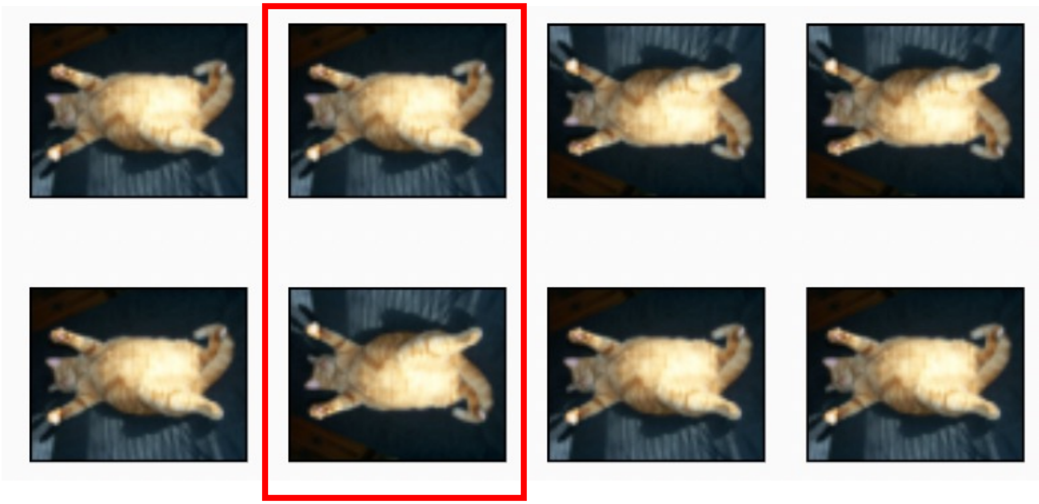


Vertical flipping

Data augmentation

- Increase the amount of data by:
  - Adding slightly modified copies of already existing data, or
  - Newly created synthetic data from existing data

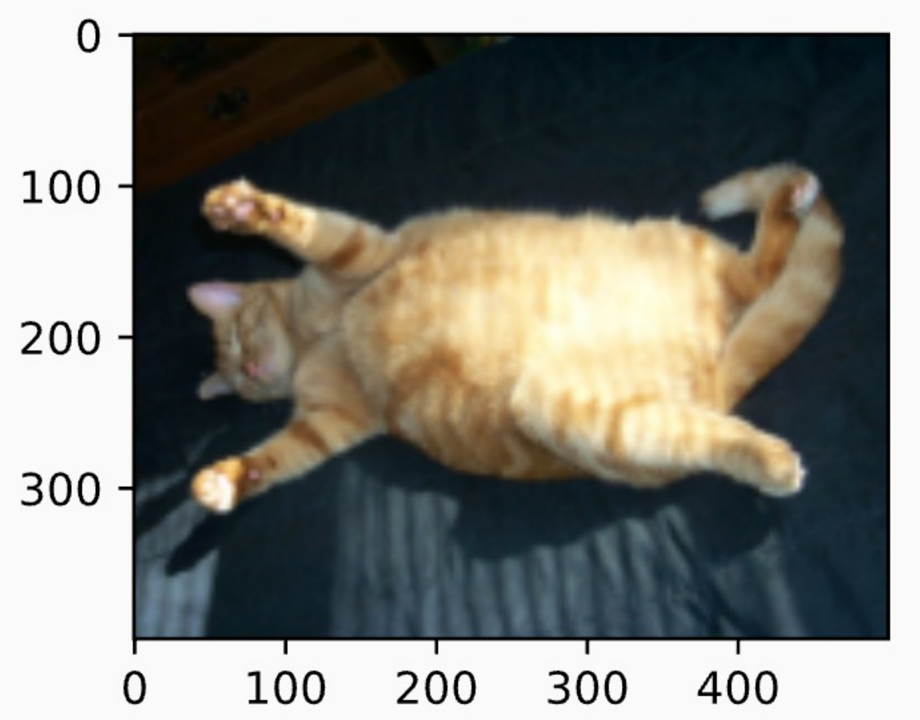


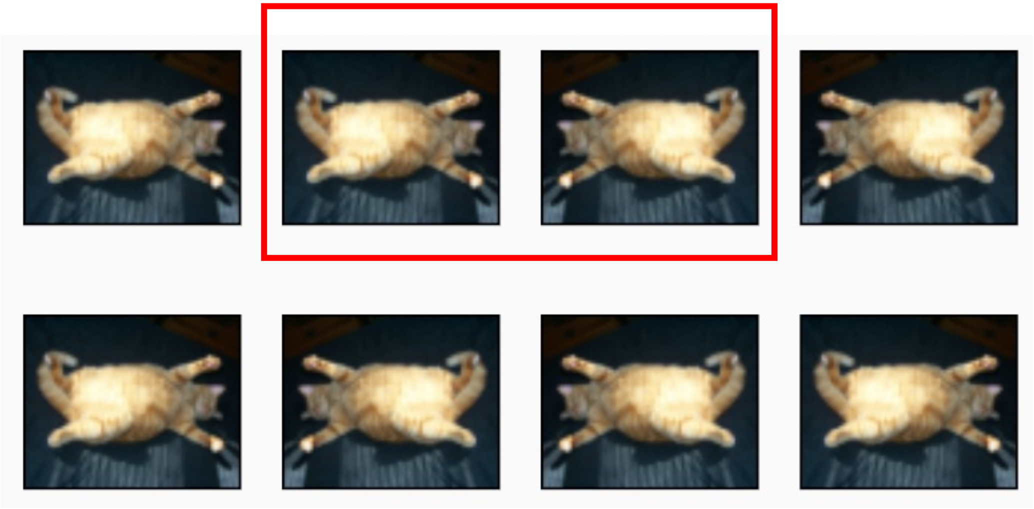


Vertical flipping

Data augmentation

- Increase the amount of data by:
  - Adding slightly modified copies of already existing data, or
  - Newly created synthetic data from existing data

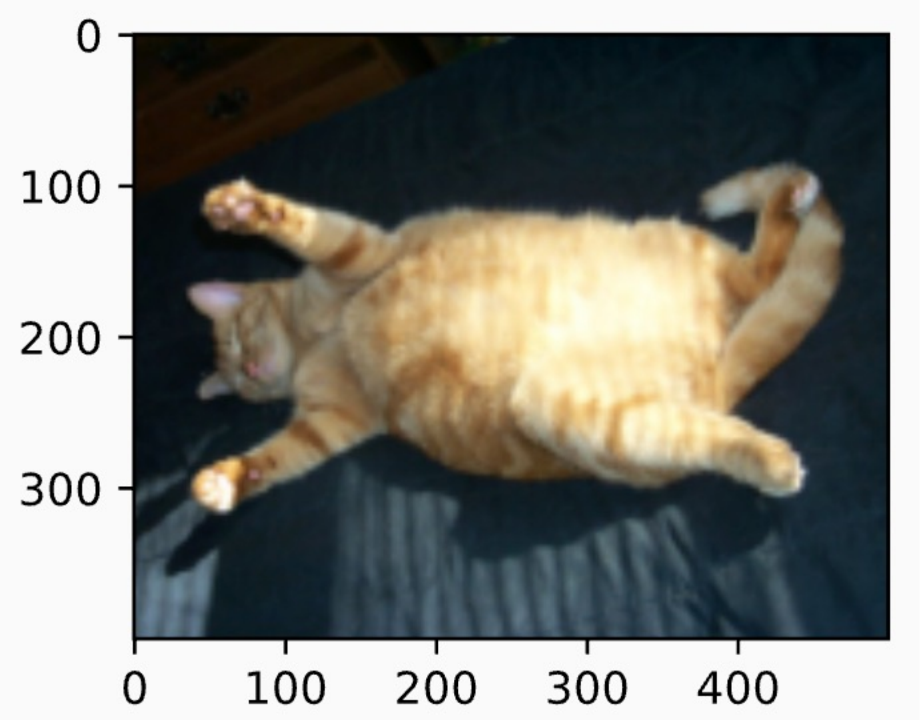


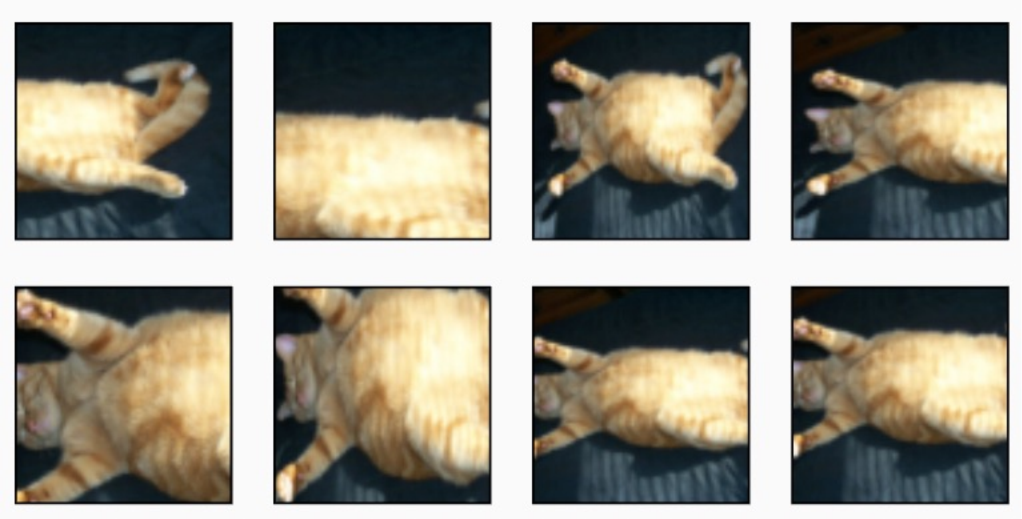


Horizontal flipping

Data augmentation

- Increase the amount of data by:
  - Adding slightly modified copies of already existing data, or
  - Newly created synthetic data from existing data

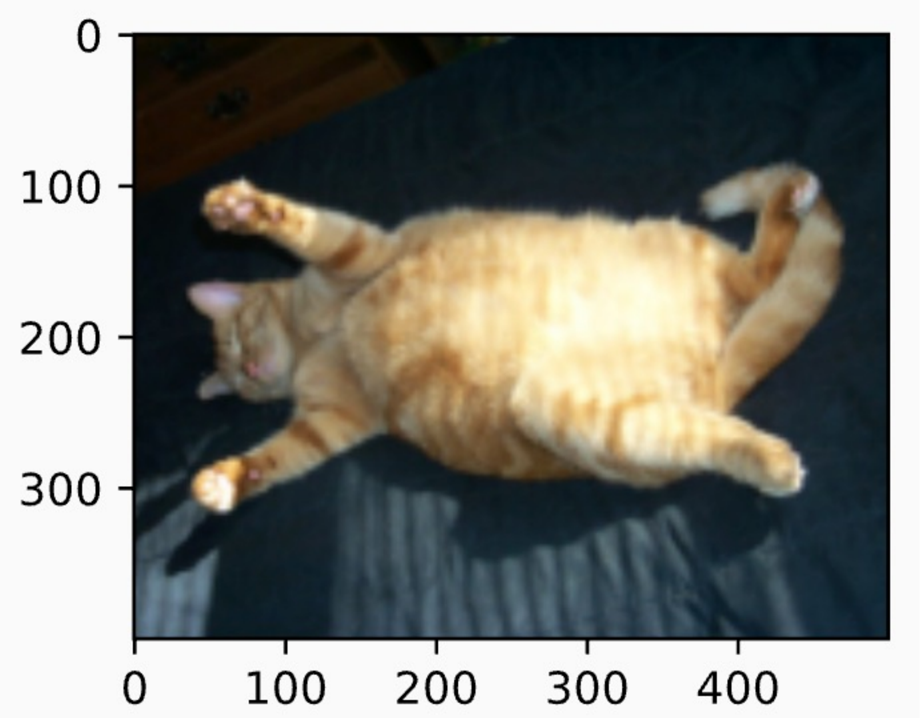


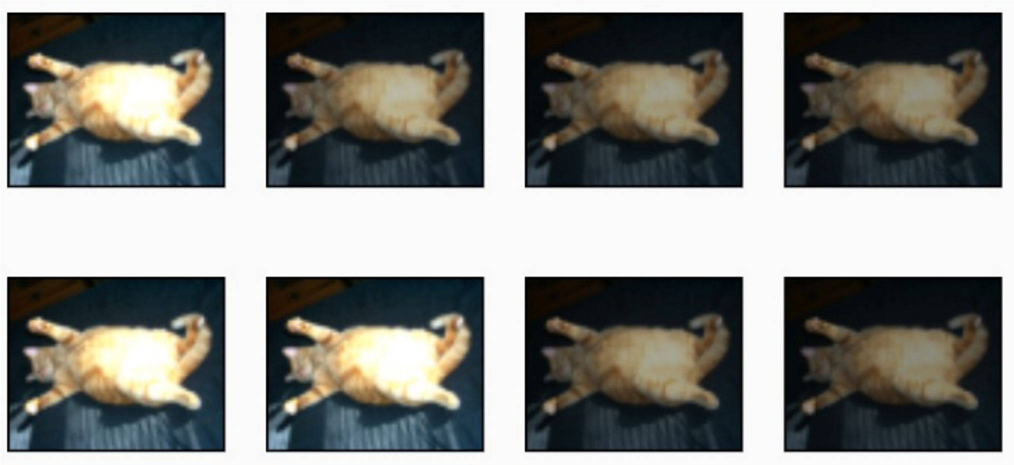


Resize and cropping

Data augmentation

- Increase the amount of data by:
  - Adding slightly modified copies of already existing data, or
  - Newly created synthetic data from existing data

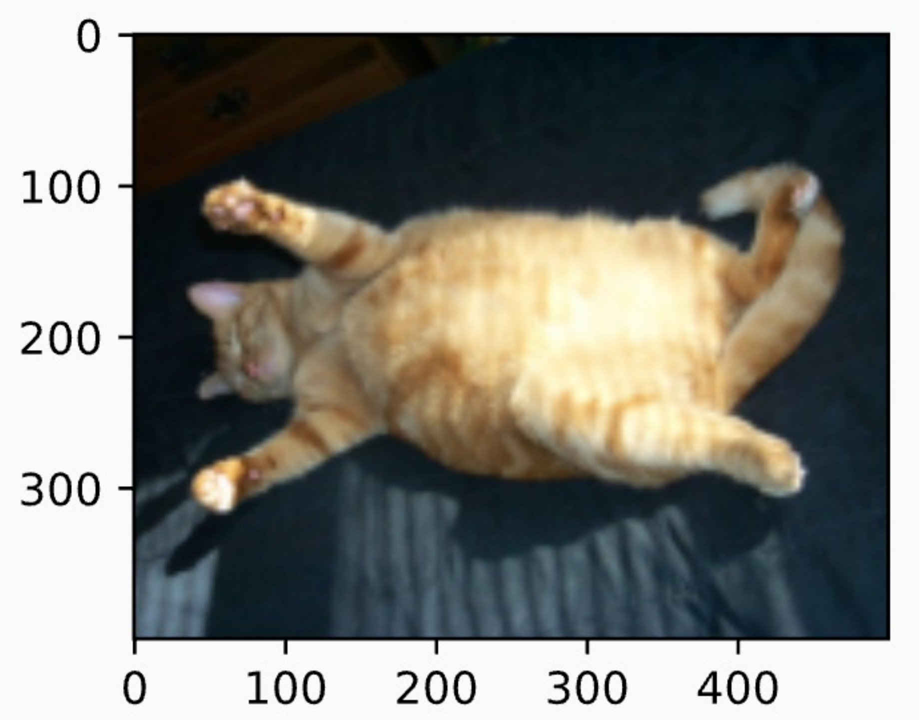




Changing brightness

Data augmentation

- Increase the amount of data by:
  - Adding slightly modified copies of already existing data, or
  - Newly created synthetic data from existing data



Color jitter:
brightness, contrast, saturation, hue

Why data augmentation

The two classes in our hypothetical dataset. The one in the left represents Brand A (Ford), and the one in the right represents Brand B (Chevrolet).

Why data augmentation

The two classes in our hypothetical dataset. The one in the left represents Brand A (Ford), and the one in the right represents Brand B (Chevrolet).

Consider: when our images only contain Ford cars facing left and Chevrolet cars facing right…

Why data augmentation

The two classes in our hypothetical dataset. The one in the left represents Brand A (Ford), and the one in the right represents Brand B (Chevrolet).

Consider: when our images only contain Ford cars facing left and Chevrolet cars facing right…

A Ford car (Brand A), but facing right.

Why data augmentation

The two classes in our hypothetical dataset. The one in the left represents Brand A (Ford), and the one in the right represents Brand B (Chevrolet).

Consider: when our images only contain Ford cars facing left and Chevrolet cars facing right…

A Ford car (Brand A), but facing right.

Our CNN may predict this car (facing right) to Chevrolet…

Why data augmentation

The two classes in our hypothetical dataset. The one in the left represents Brand A (Ford), and the one in the right represents Brand B (Chevrolet).

Consider: when our images only contain Ford cars facing left and Chevrolet cars facing right…

A Ford car (Brand A), but facing right.

Our CNN may predict this car (facing right) to Chevrolet…

Data augmentation:
Gives more variations for data

Why data augmentation

The two classes in our hypothetical dataset. The one in the left represents Brand A (Ford), and the one in the right represents Brand B (Chevrolet).

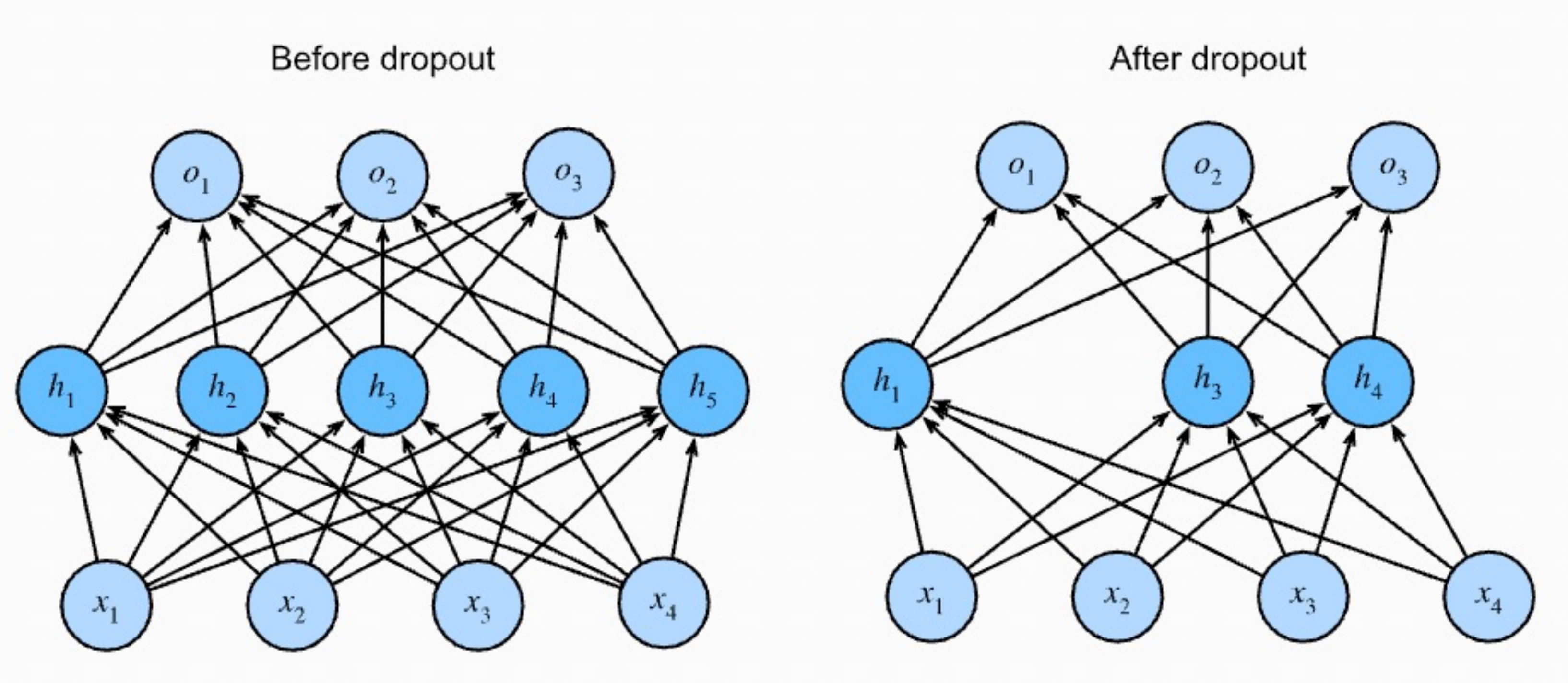
Consider: when our images only contain Ford cars facing left and Chevrolet cars facing right…

A Ford car (Brand A), but facing right.

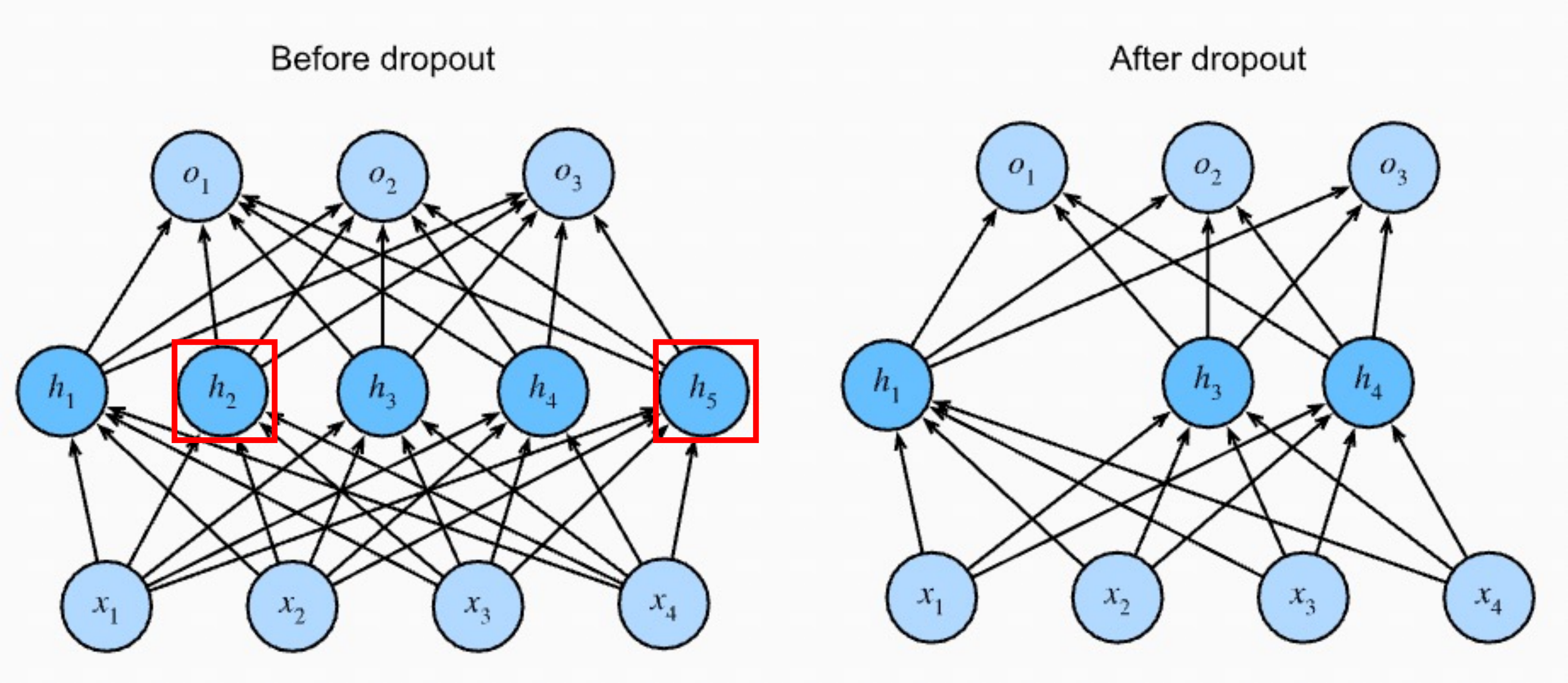
Our CNN may predict this car (facing right) to Chevrolet…

Data augmentation:
Gives more variations for data → better generalization

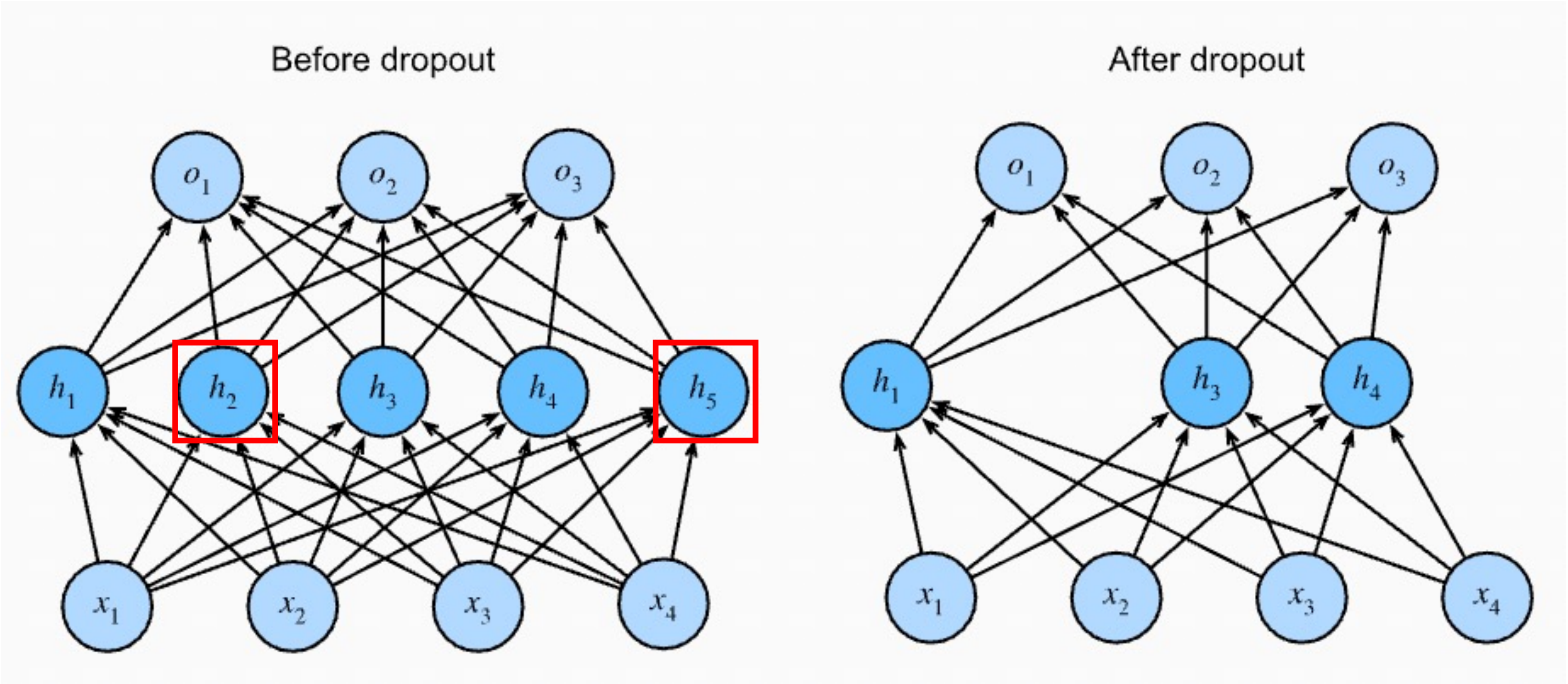
Dropout



Dropout

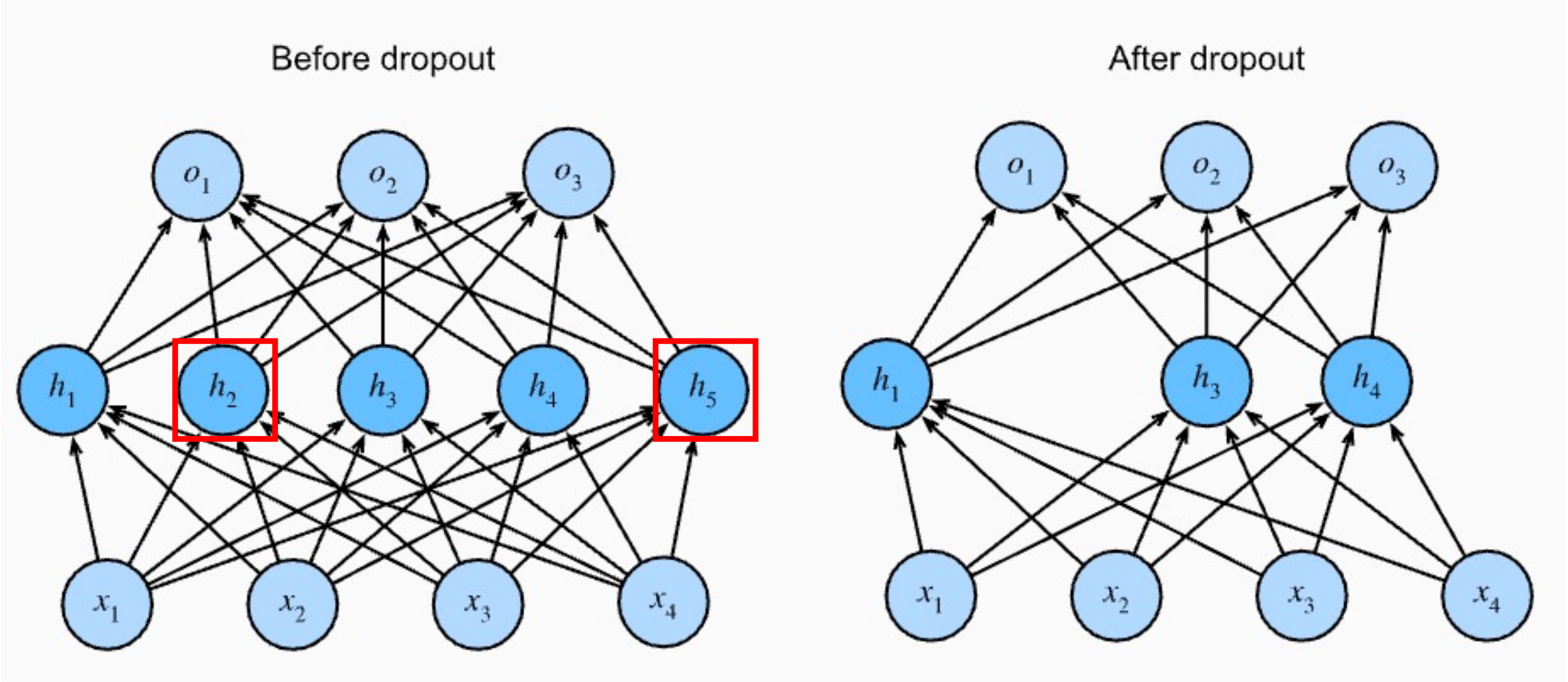


Dropout



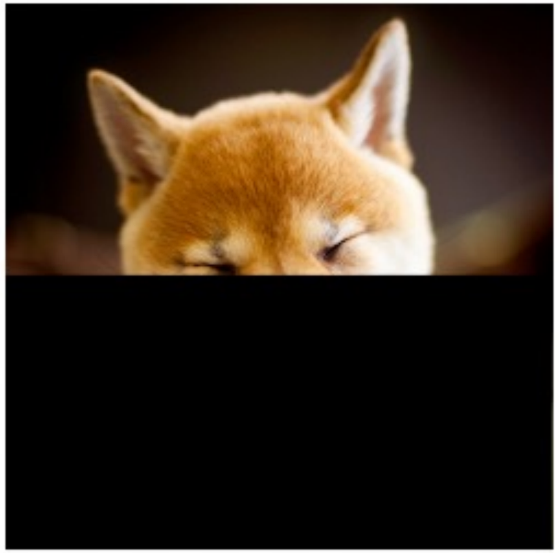
Dropout in training: select an arbitrary percentage of neurons (weights) and mask them

Dropout



Dropout in training: select an arbitrary percentage of neurons (weights) and mask them
Dropout in testing: use all parameters, no dropout

Dropout

Dropout

Why dropout?

Dropout

Why dropout? → alleviate overfitting

Regularization/weight decay

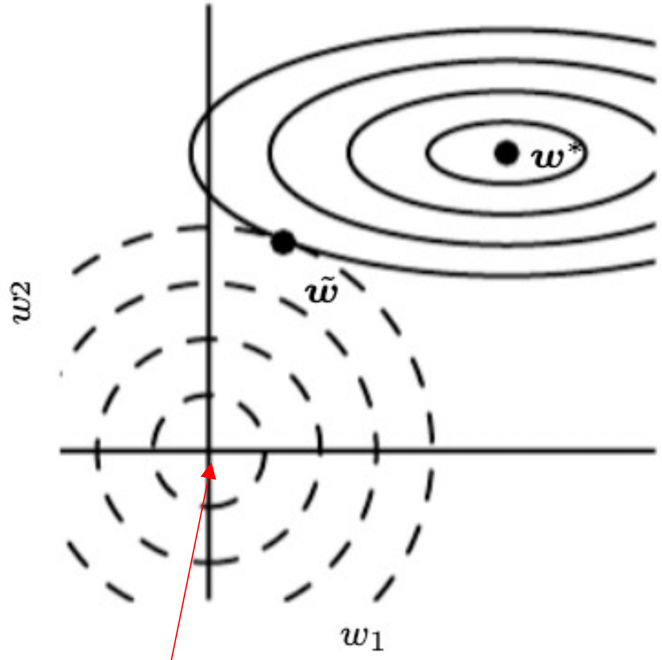
$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta}),$$

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

Regularization/weight decay

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta}),$$

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

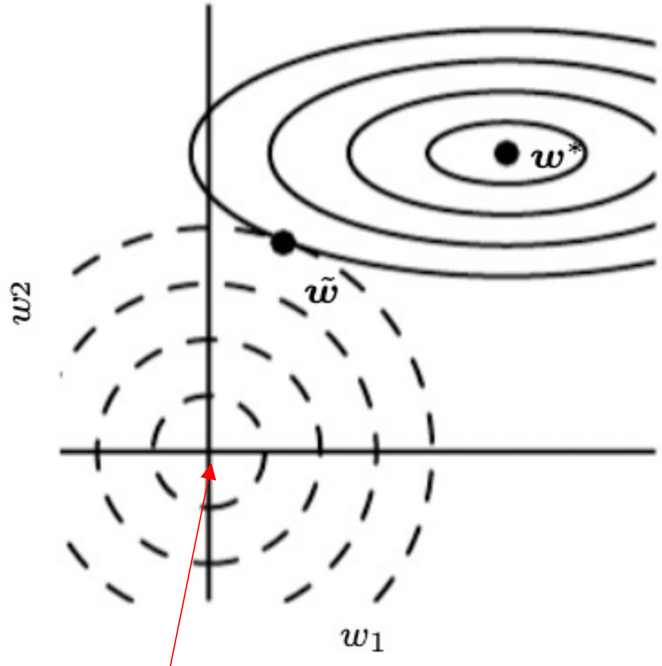


Origin

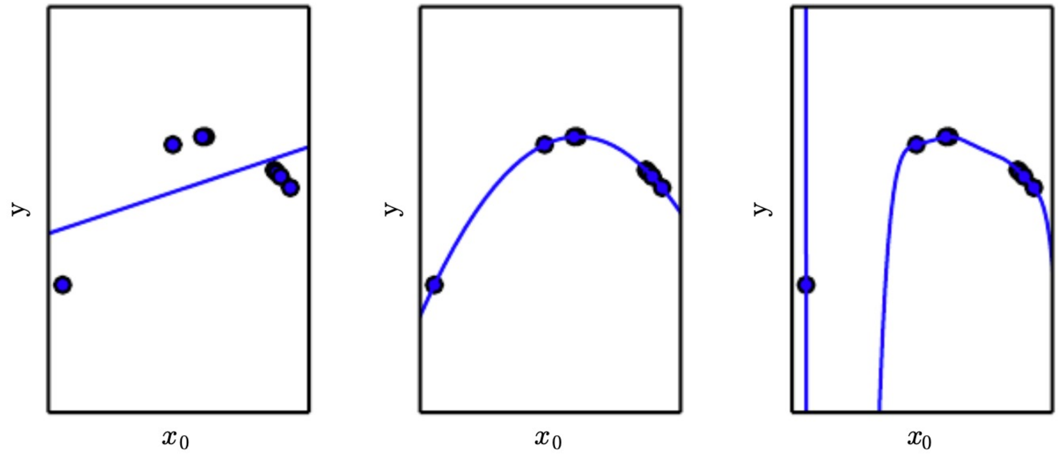
Regularization/weight decay

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta}),$$

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$$



Origin



Linear model

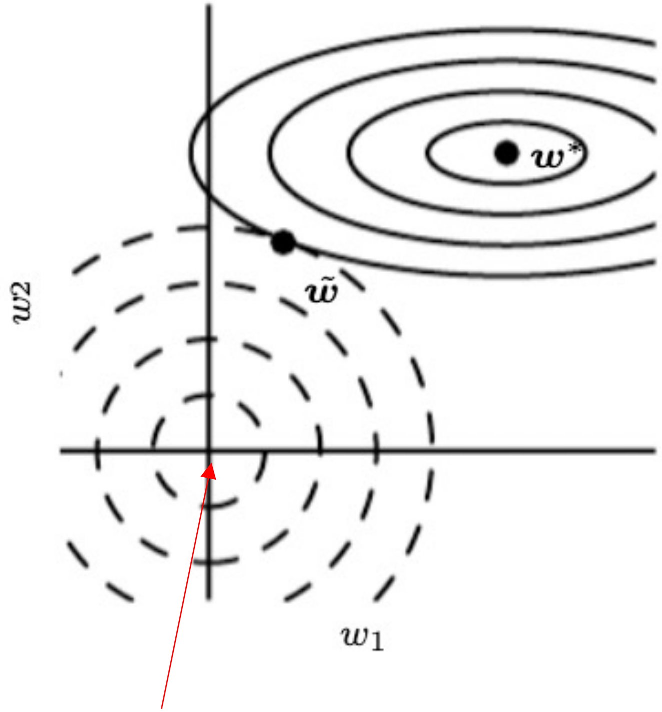
Quadratic model

Polynomial model (9 degree)

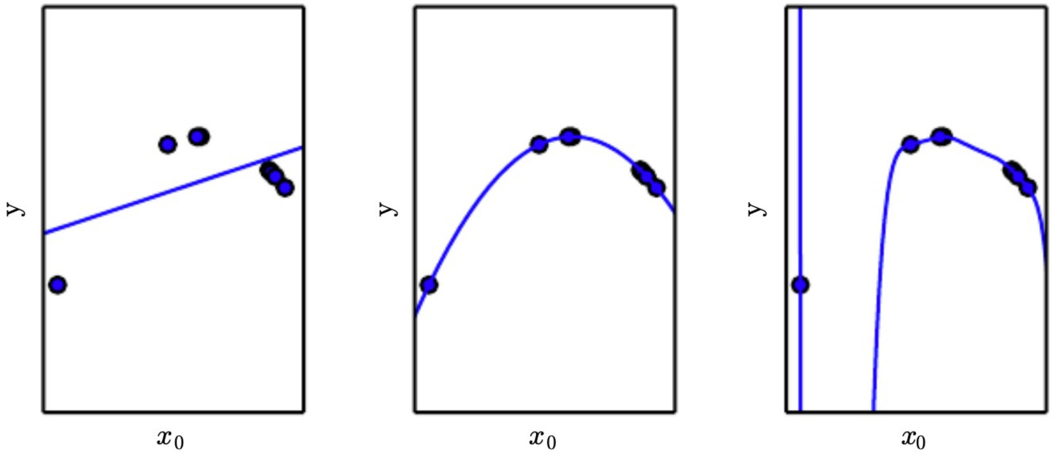
Regularization/weight decay

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta}),$$

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$$



Origin



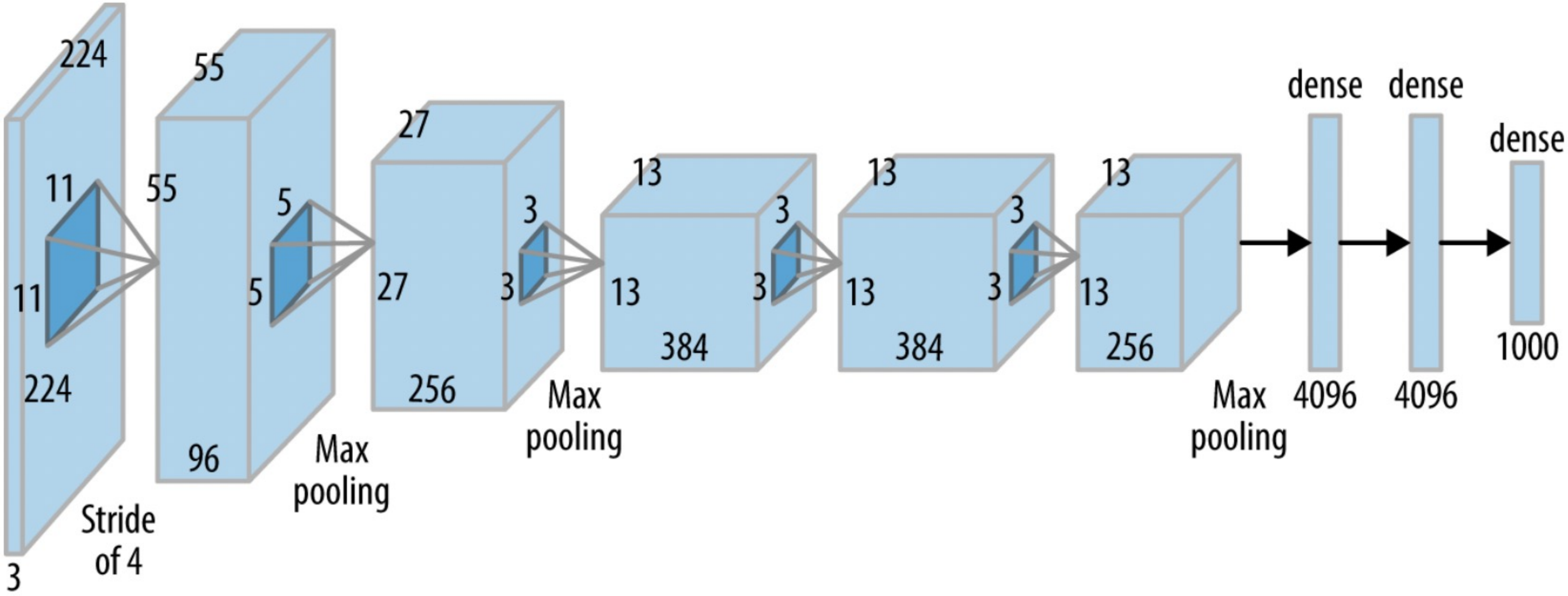
Linear model

Quadratic model

Polynomial model (9 degree)

Improve generalization performance

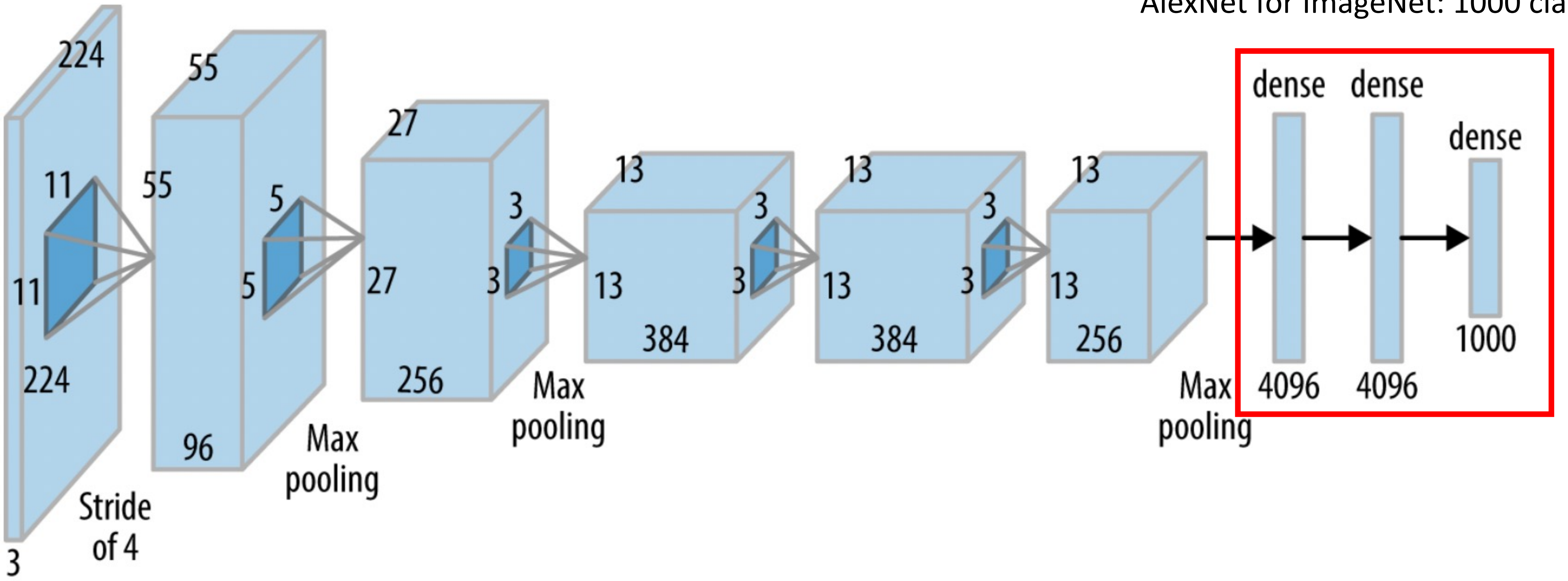
Pre-train



Suppose we have a learned model → weight parameters are determined and fixed

Pre-train

AlexNet for ImageNet: 1000 classes



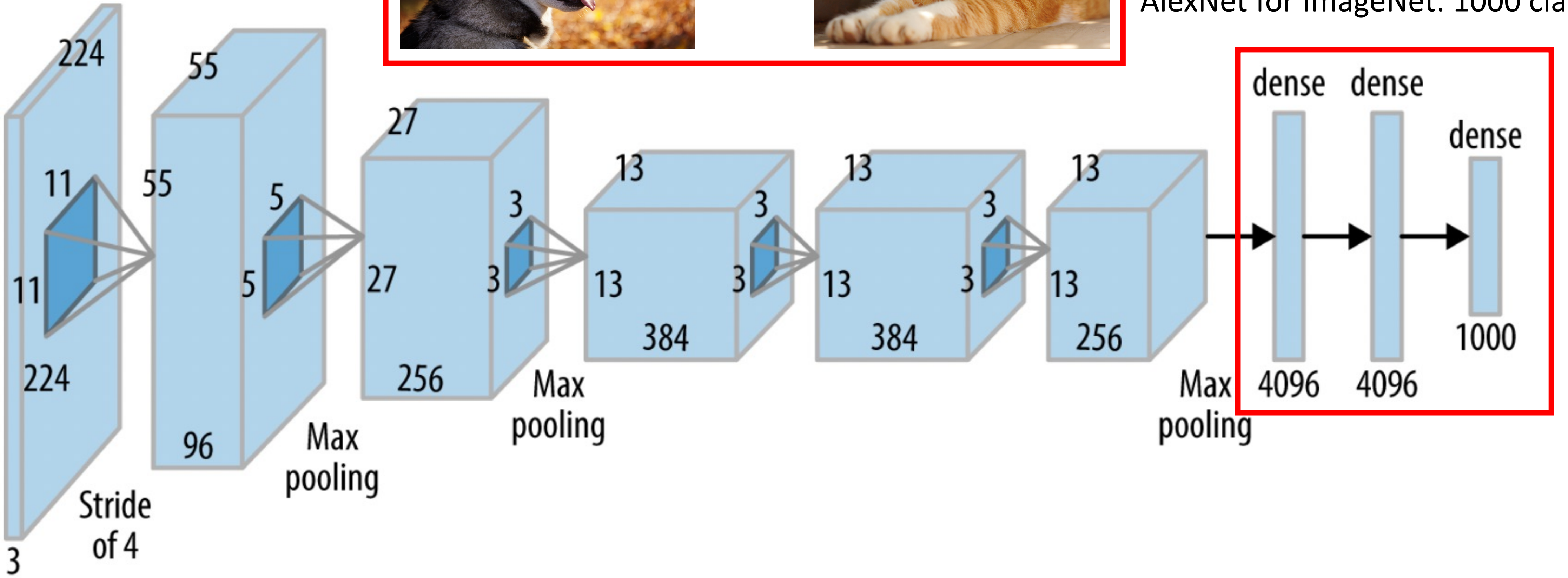
Suppose we have a learned model → weight parameters are determined and fixed

Pre-train

Now: Two classes

vs

AlexNet for ImageNet: 1000 classes

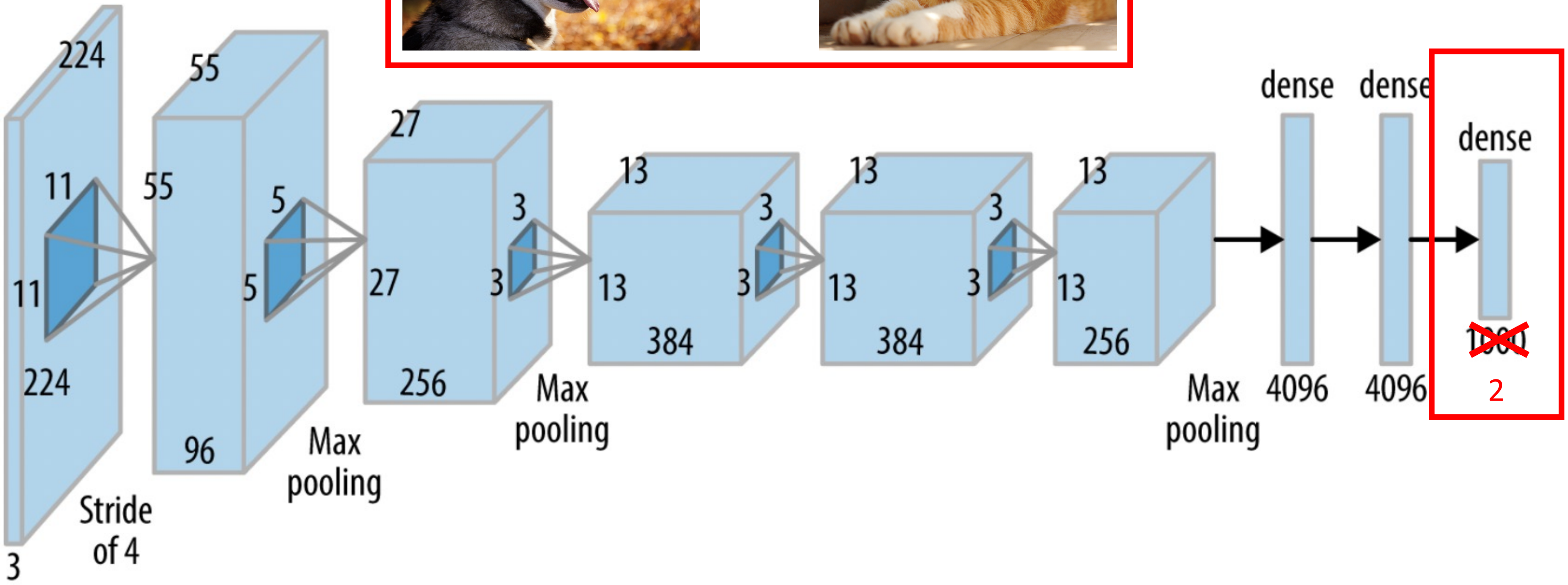


Suppose we have a learned model → weight parameters are determined and fixed

Pre-train

Now: Two classes

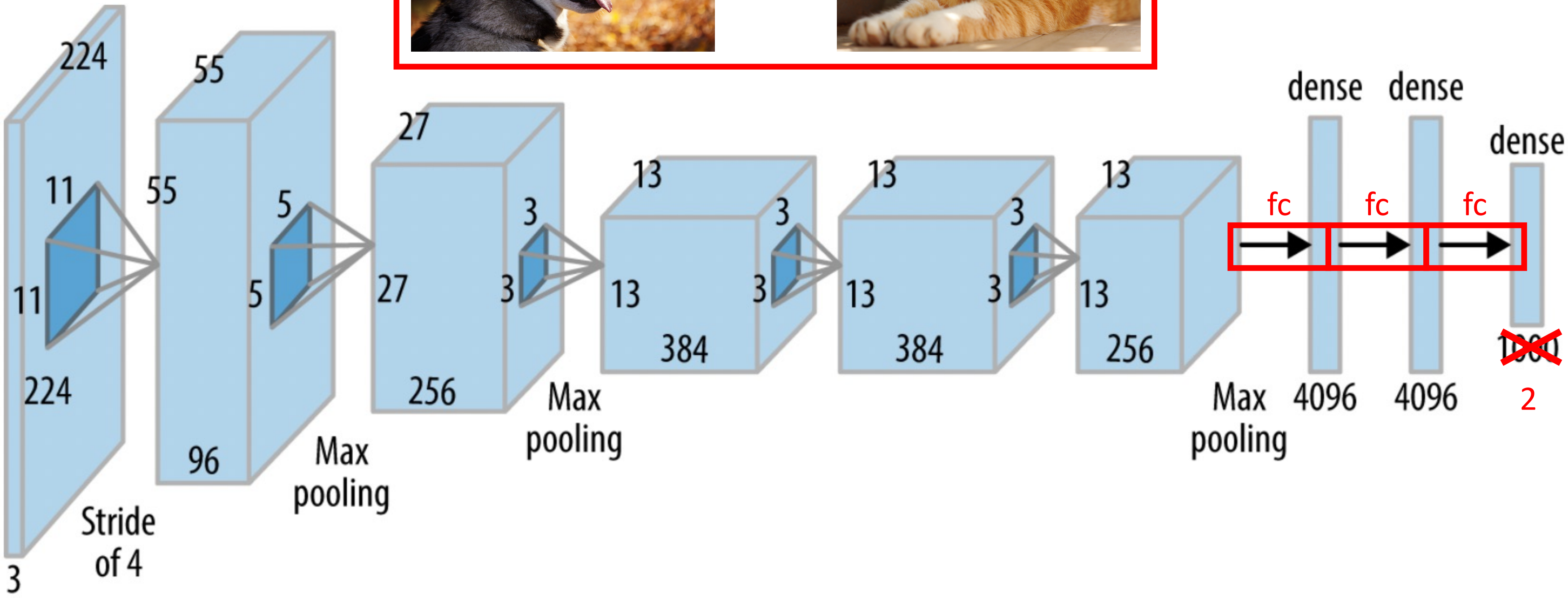
vs



Suppose we have a learned model → weight parameters are determined and fixed

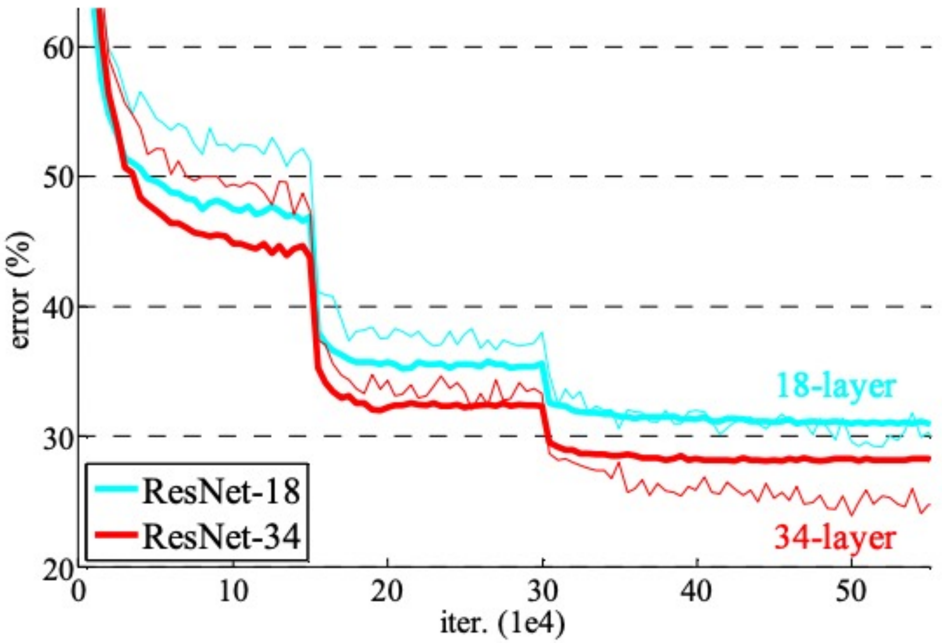
Pre-train

Now: Two classes

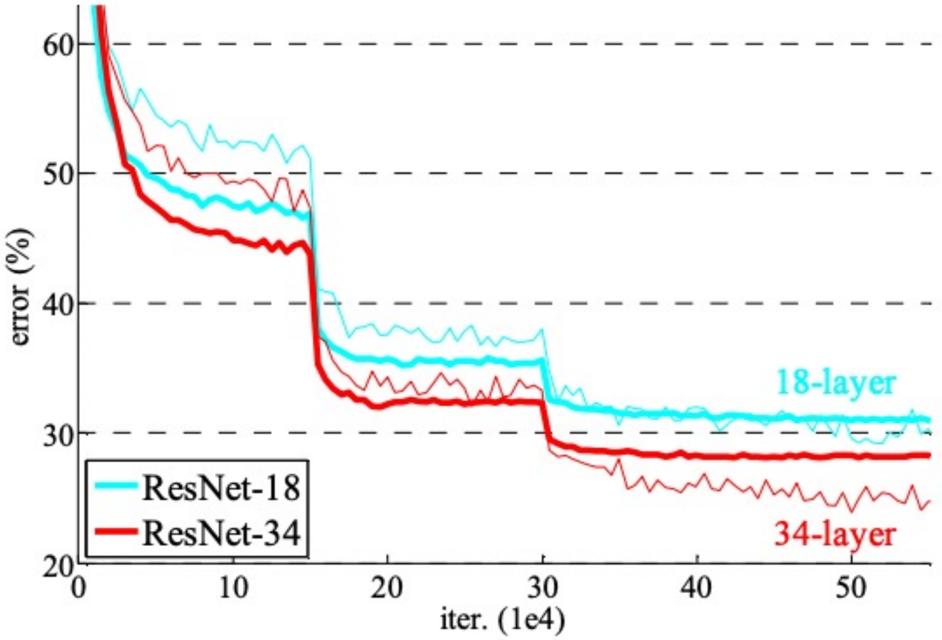


Suppose we have a learned model → weight parameters are determined and fixed

Stagewise/restart training



Stagewise/restart training



One-loop SGD

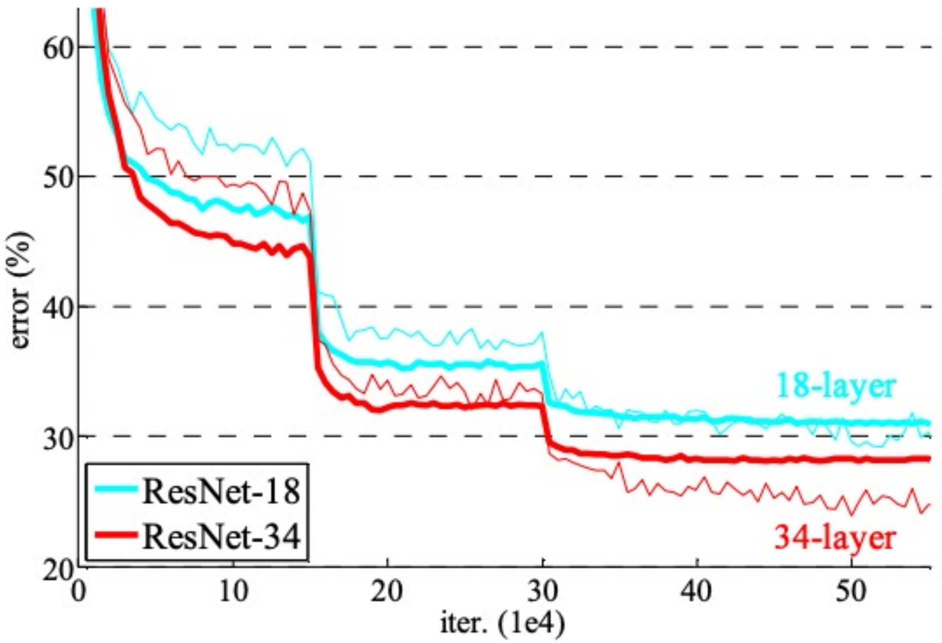
For t=1→T

  Compute stochastic gradients G_t for w_t

  Update $w_{t+1} = w_t - \eta G_t$

Endfor

Stagewise/restart training



Two-loop SGD

For s=1→S

For t=1→T

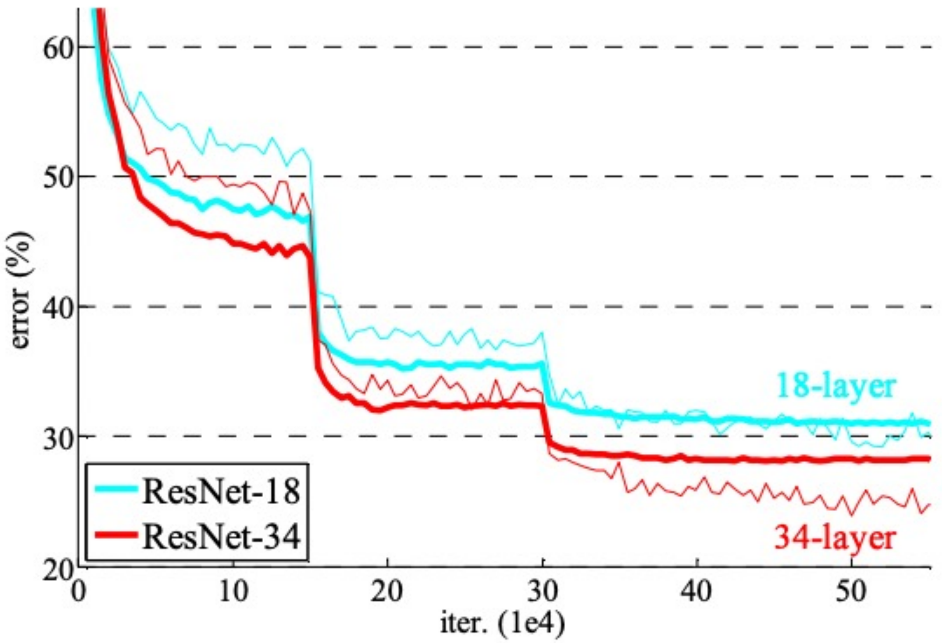
Compute stochastic gradients G_t for w_t

Update $w_{t+1} = w_t - \eta_s G_t$

Endfor

Endfor

Stagewise/restart training



Two-loop SGD

For s=1→S

For t=1→T

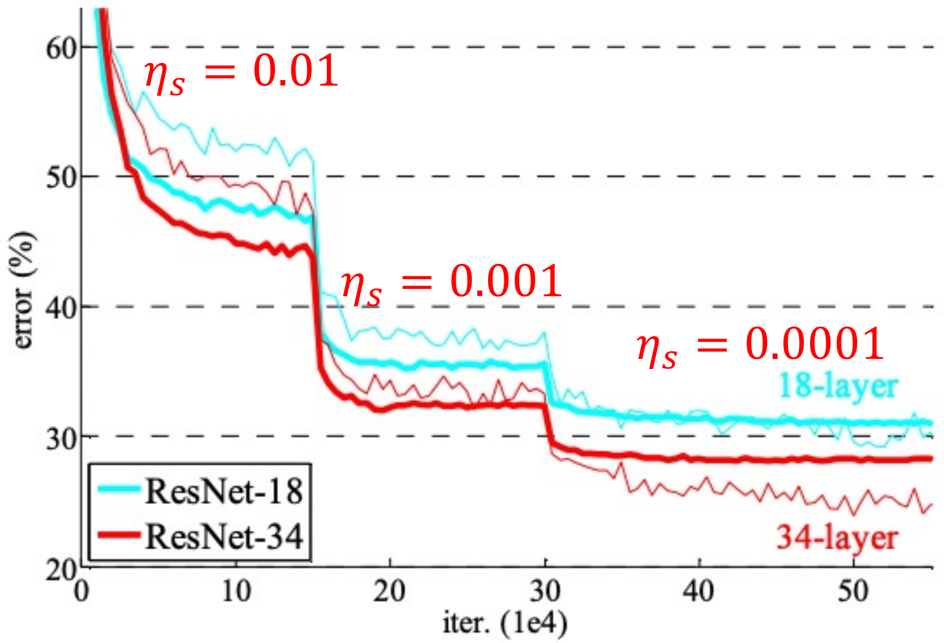
Compute stochastic gradients G_t for w_t

Update $w_{t+1} = w_t - \eta_s G_t$

Endfor

Endfor

Stagewise/restart training



Two-loop SGD

For s=1→S

For t=1→T

Compute stochastic gradients G_t for w_t

Update $w_{t+1} = w_t - \eta_s G_t$

Endfor

Endfor

Many other tricks for CNNs

- Large mini-batch in stochastic gradient descent
- Learning rate warmup [warmup]
- Mixup augmentation [mixup]
- Others, e.g., [BagOfTricks]

References

- [BN] Ioffe, Sergey, and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift.” *arXiv preprint arXiv:1502.03167* (2015).
- [warmup] Goyal, Priya, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. "Accurate, large minibatch sgd: Training imagenet in 1 hour." *arXiv preprint arXiv:1706.02677* (2017).
- [mixup] Zhang, Hongyi, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. "mixup: Beyond empirical risk minimization." *arXiv preprint arXiv:1710.09412* (2017).
- [BagOfTricks] He, Tong, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. "Bag of tricks for image classification with convolutional neural networks." In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 558-567. 2019.