Neural Networks: Convolutional Nets, Regularization, Data augmentation Dropout and Batch Normalization

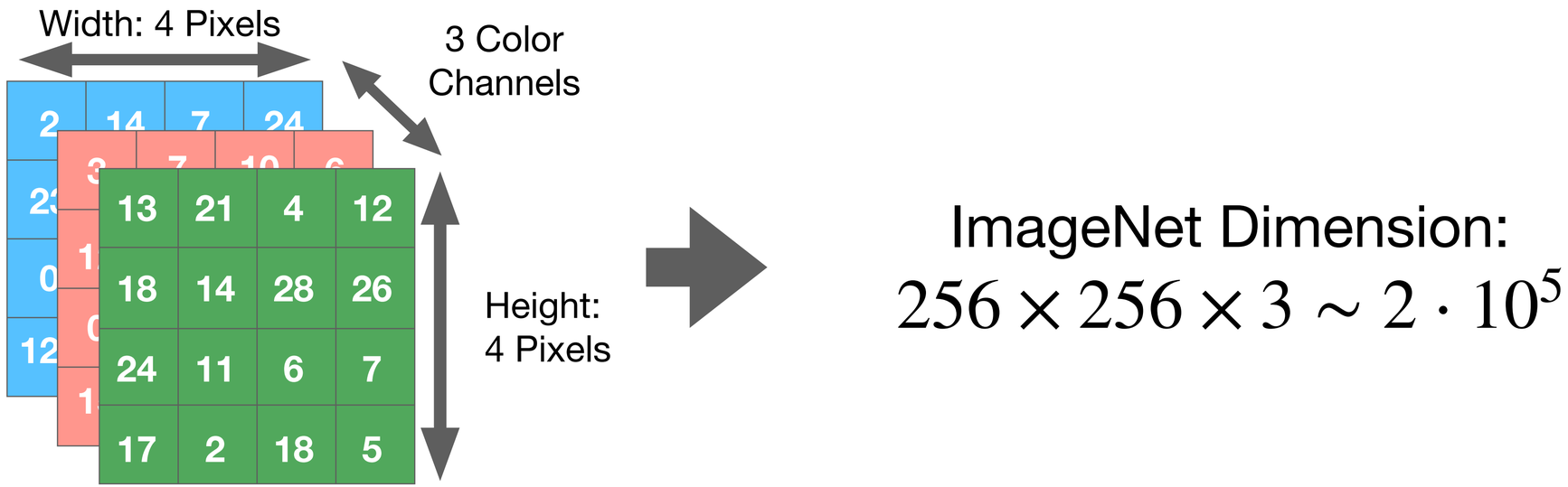
Machine Learning Course - CS-433
Nov 15, 2022
Nicolas Flammarion

EPFL

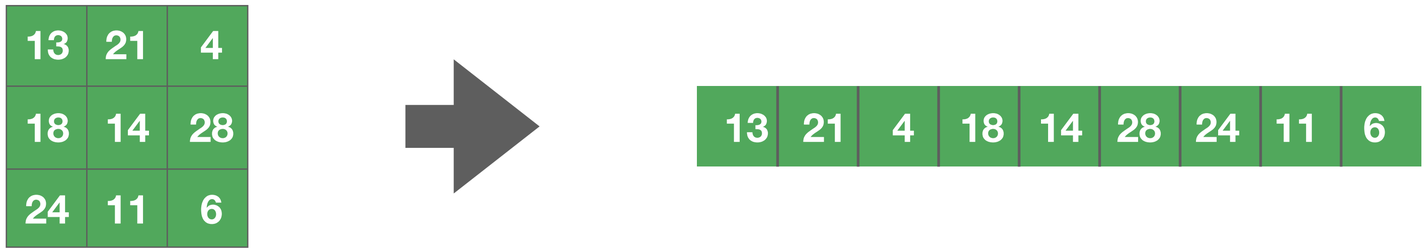
Convolutional Networks

Fully connected NNs have many parameters and do not capture spatial dependencies

- Fully connected NNs have $O(K^2L)$ parameters: training requires much data



- Fully connected NNs conceive a picture as a flattened vector and forget initial spatial dependencies

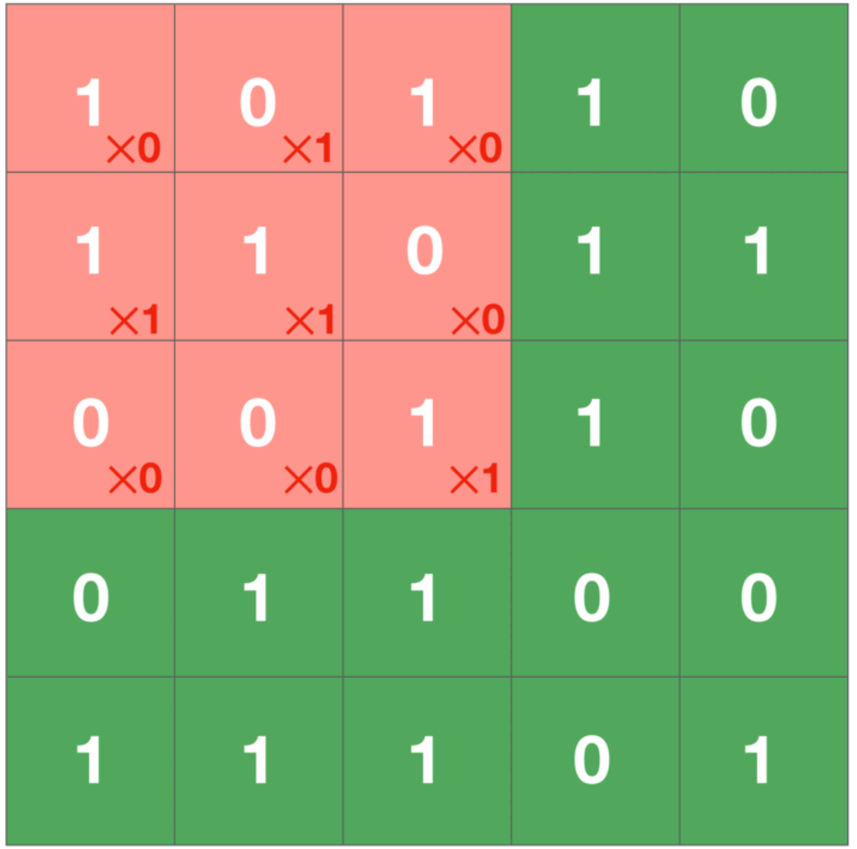


Flattening of a 3×3 picture into a 1×9 vector

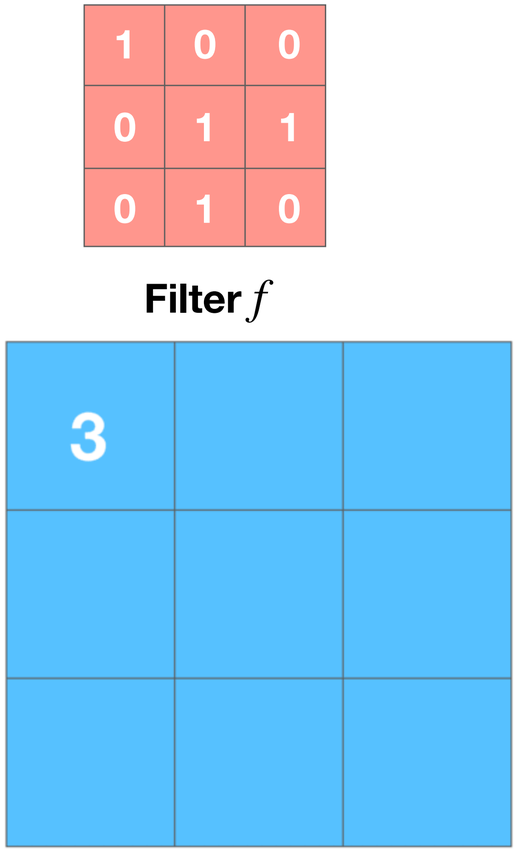
Convolution

$$x^{(1)}_{n,m} = \sum_{k,l} f_{k,l} \cdot x^{(0)}_{n-k,m-l}$$

- We consider local filter: $f_{k,l} \neq 0$ for small values of $|k|$ and $|l|$
  - ➡ $x^{(1)}_{n,m}$ only depends on the value of $x^{(0)}$ close to (n, m)
  - ➡ f represents the learnable weights
- We use the same filter at every position - **weight sharing**
- **Shift invariance** - shifted input results in shifted output



Image

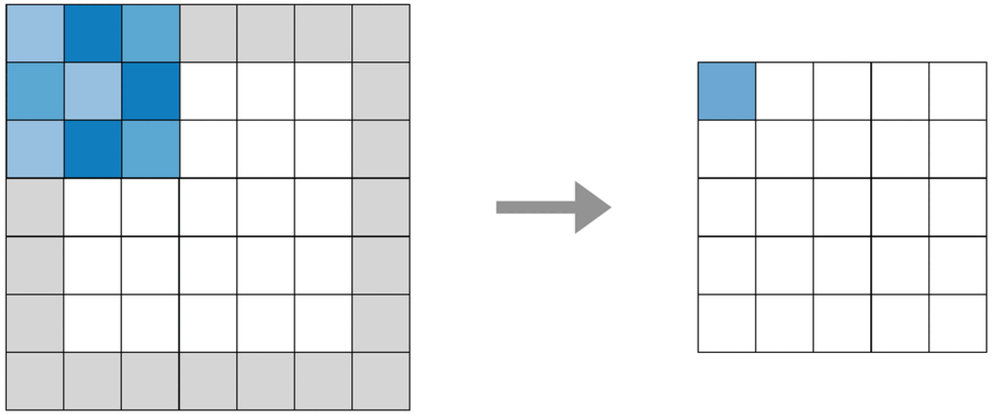


Convolved Feature

➡ **Few parameters** and **features generalize between locations**

Handling of borders

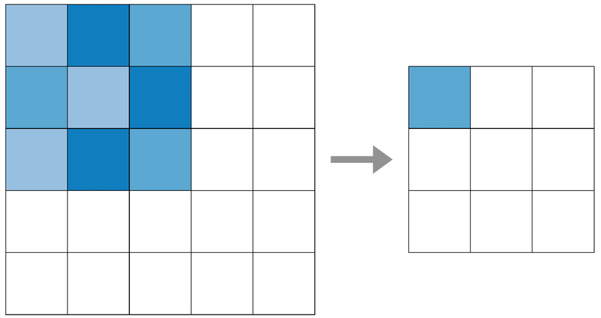
Zero padding:



Add zeros to each side of the boundaries of the input

➡ Convolved feature is of the same dimensionality as the input

Valid padding:

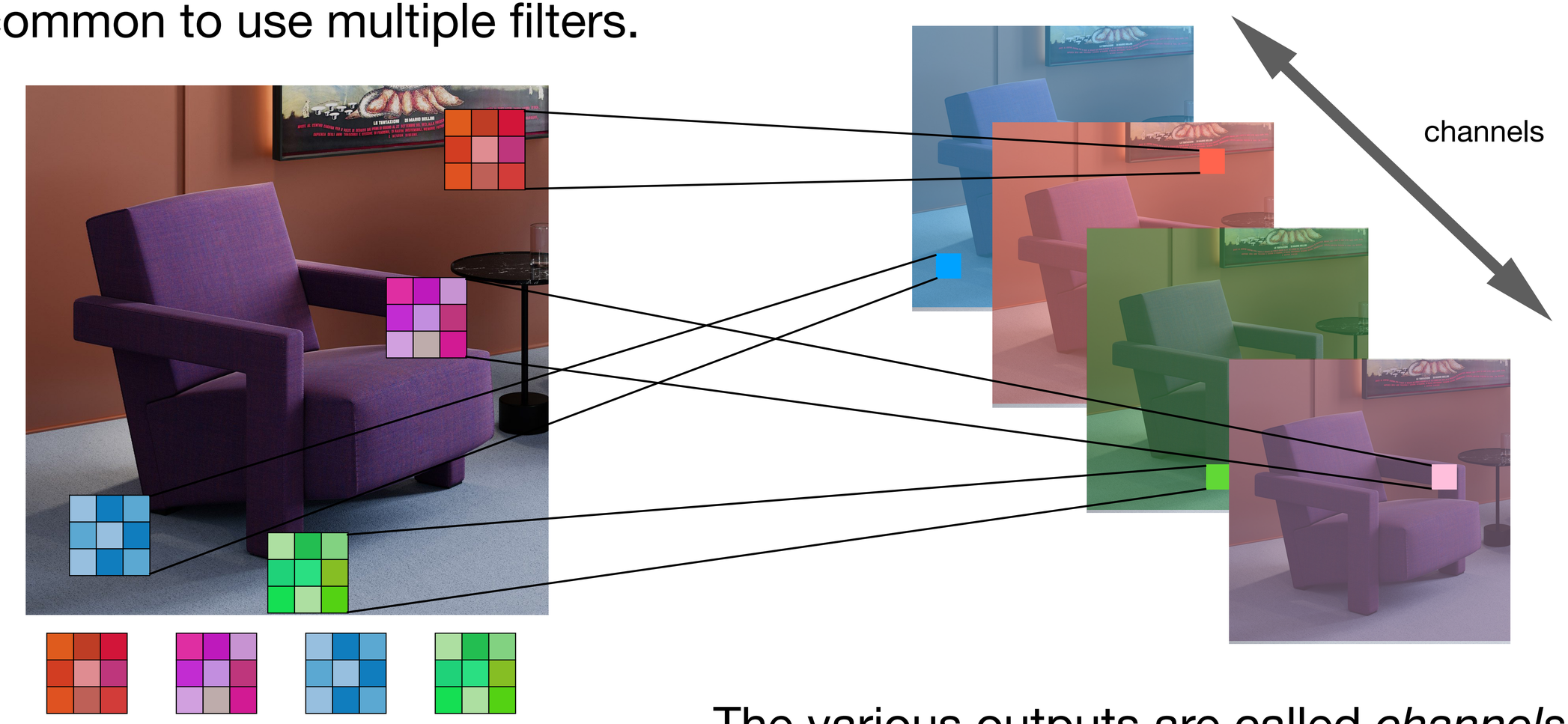


Perform the convolution only for positions so that the whole filter lies inside the original data

➡ Dimension of the convolved feature is smaller than the input's one

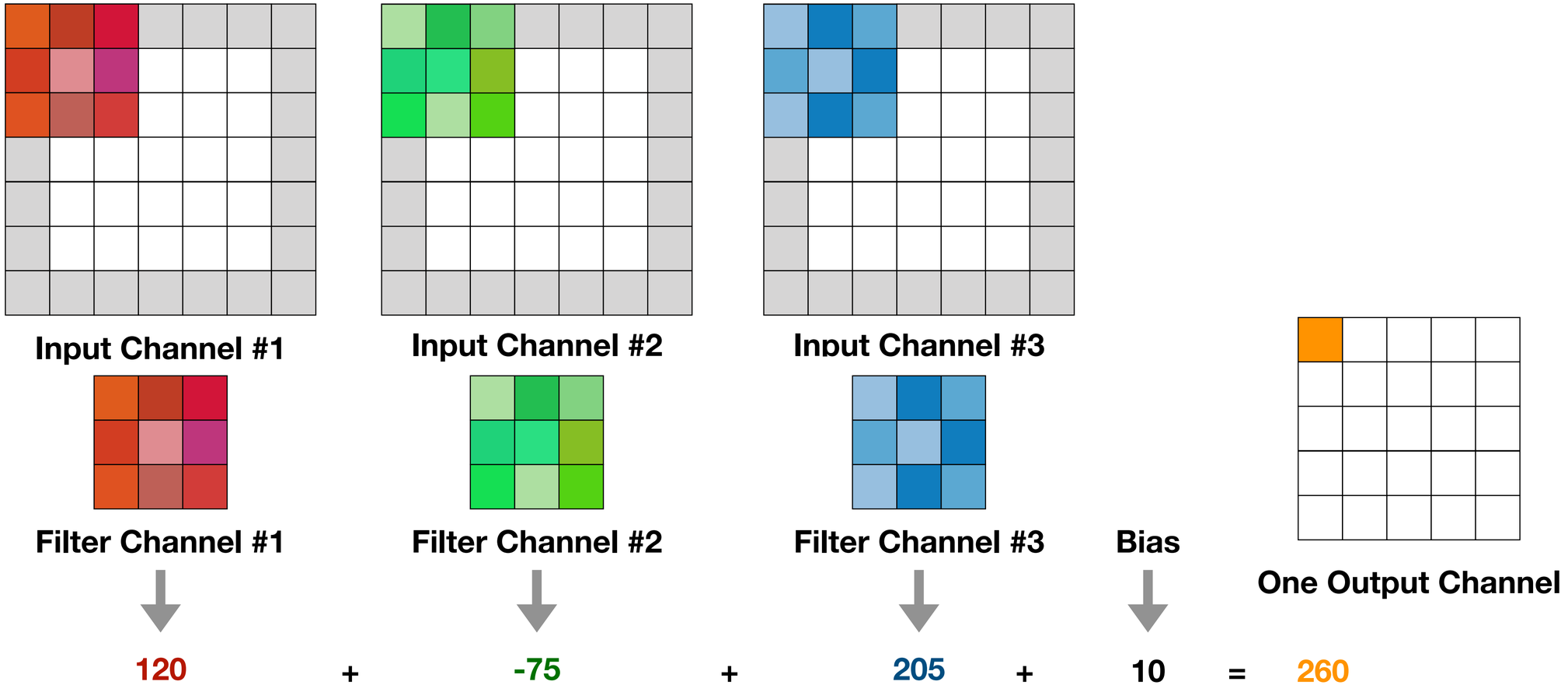
Multiple Channels

It is common to use multiple filters.



The various outputs are called *channels*

Convolutional Layer

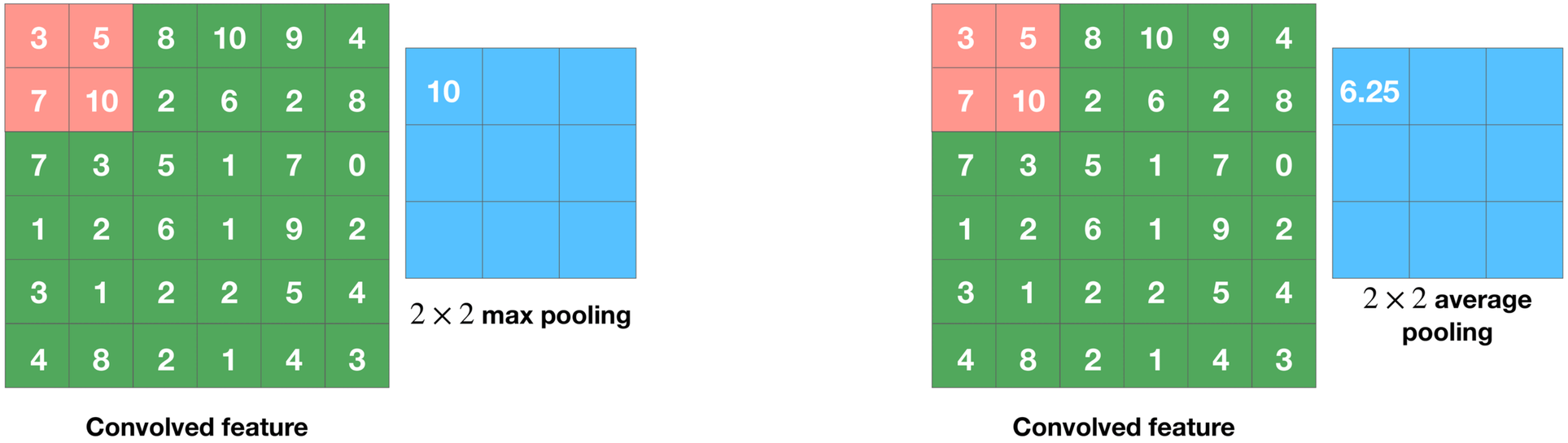


- The filters and the bias are the learnable parameters of the layer
- Each output channel has their own independent filters and bias
- Hyper-parameters of the convolutional layer: size, padding, stride

Pooling: often applied after the convolutional layer

Max pooling: returns the maximum value of the portion of the convolved feature covered by the kernel

Average pooling: returns the average value of the portion of the convolved feature covered by the kernel

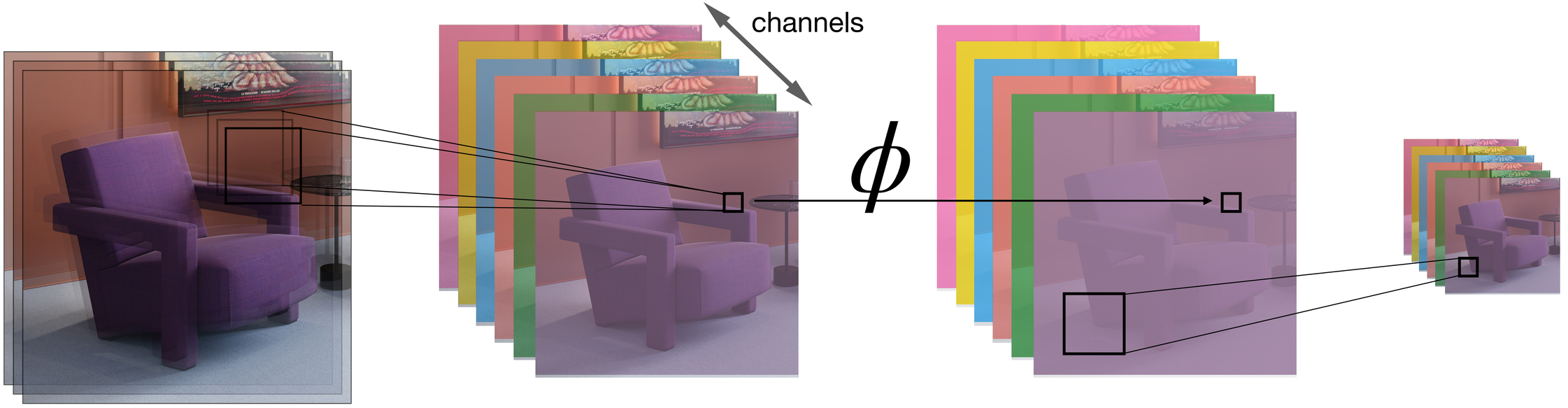


Pooling is a downsampling operation which reduces the spatial size of the convolved feature

Remark: Pooling layers have no learnable parameters

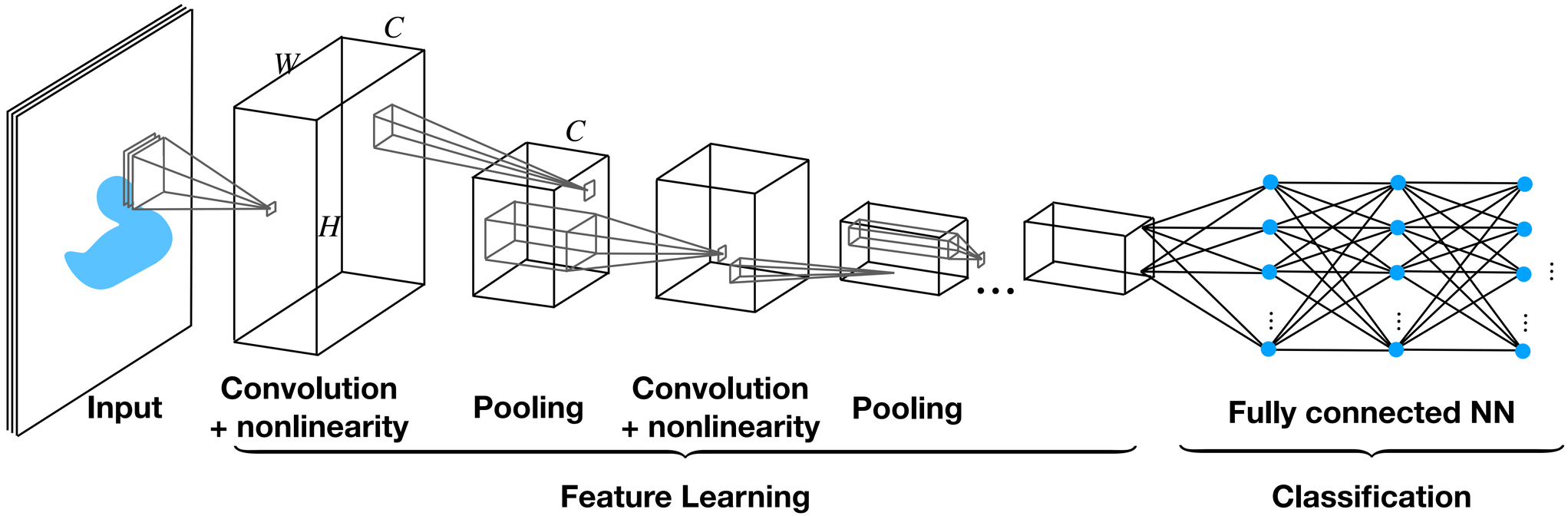
Hyper-parameters: size, type, stride

Non-linearity and convolutional NNs



Important: We include a non-linearity like ReLU after the convolutional layer

Convolutional NNs are composed of succession of convolutional and pooling layers



Receptive field (area of input that affects a given neuron) increases with depth:

- First layers extract the low-level features, e.g., edges, colors
- Next layers extract high-level features, e.g., objects

➡ ConvNet reduces the images into a form easier to process without loosing essential features

Backpropagation with weight sharing

Weight sharing is used in CNN: many edges use the same weights

Training:

1. Run backpropagation ignoring the weights are shared (considering each weight to be independent variable).
2. Once the gradient is computed, sum up the gradients of all edges that share the same weight

Why: let $f(x, y, z) : \mathbb{R}^3 \to \mathbb{R}$ and $g(x, y) = f(x, y, x)$

$$\left(\frac{\partial g}{\partial x}(x, y), \frac{\partial g}{\partial y}(x, y)\right) = \left(\frac{\partial f}{\partial x}(x, y, x) + \frac{\partial f}{\partial z}(x, y, x), \frac{\partial f}{\partial y}(x, y, x)\right)$$

Chain rule

Normalization Layers

Batch Normalization - Train Fwd

Normalize channels of batch $\mathbf{X} \in \mathbb{R}^{N \times C \times H \times W}$ to have zero mean, unit variance:

$$\hat{\mathbf{X}} = \frac{\mathbf{X} - \mu(\mathbf{X})}{\sqrt{\sigma^2(\mathbf{X}) + \varepsilon}}$$

with small hyperparameter $\varepsilon \in \mathbb{R}_{\geq 0}$ for stability and $\mu(\mathbf{X}), \sigma(\mathbf{X}) \in \mathbb{R}^{1 \times C \times 1 \times 1}$ where:

$$\mu(\mathbf{X}) = \frac{1}{NHW} \sum_{n,h,w} \mathbf{X}_{n,\cdot,h,w}, \quad \sigma^2(\mathbf{X}) = \frac{1}{NHW} \sum_{n,h,w} \left(\mathbf{X}_{n,\cdot,h,w} - \mu(\mathbf{X})\right)^2$$

Final output: $\mathbf{Y} = \text{BN}(\mathbf{X}) = \gamma \odot \hat{\mathbf{X}} + \beta$ for learnable $\gamma, \beta \in \mathbb{R}^{1 \times C \times 1 \times 1}$

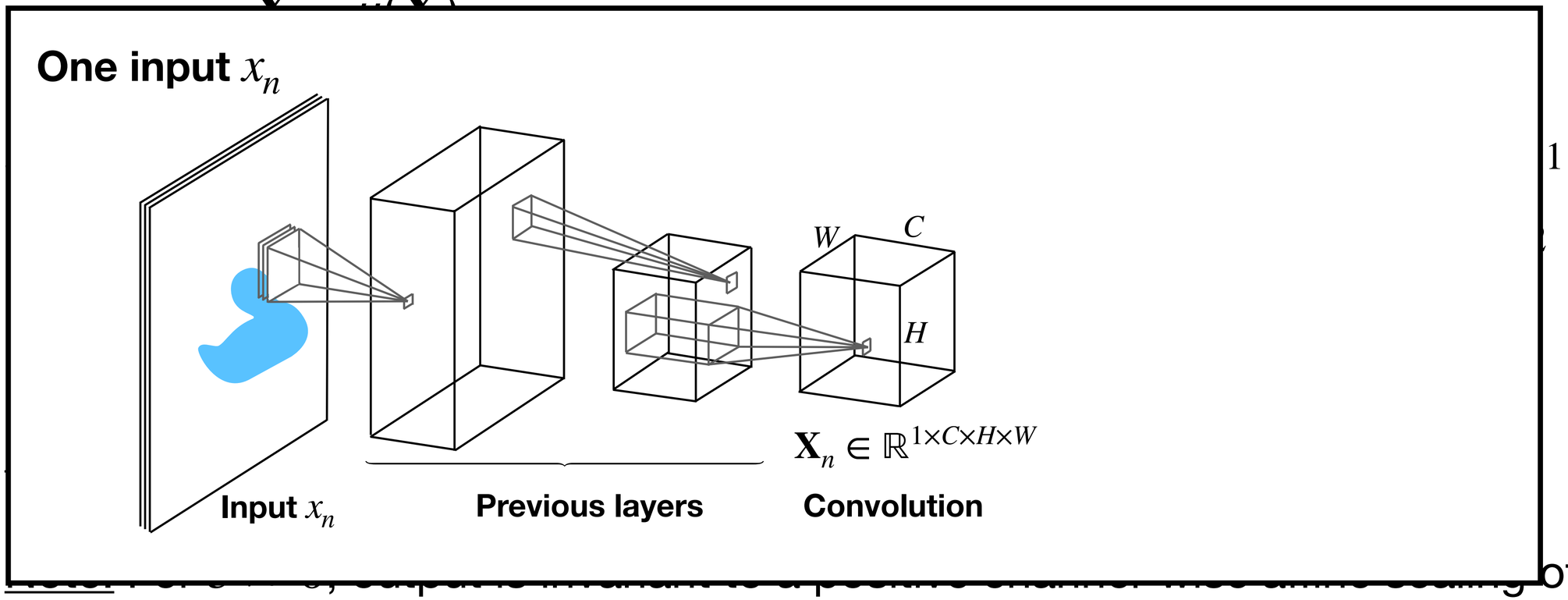
Note: For $\varepsilon \approx 0$, output is invariant to a positive channel-wise affine scaling of $\mathbf{X}$:

$\text{BN}(a \odot \mathbf{X} + b) = \text{BN}(\mathbf{X})$ for $a \in \mathbb{R}_{>0}^{1 \times C \times 1 \times 1}$ and $b \in \mathbb{R}^{1 \times C \times 1 \times 1}$

No need to include a bias before batch normalization

Batch Normalization - Train Fwd

Normalize channels of batch $\mathbf{X} \in \mathbb{R}^{N \times C \times H \times W}$ to have zero mean, unit variance:



^{1} where:

of $\mathbf{X}$:

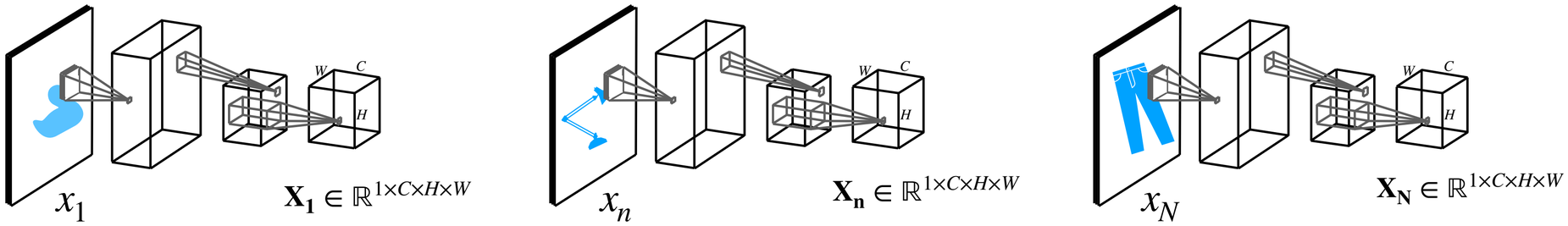
$$\text{BN}(a \odot \mathbf{X} + b) = \text{BN}(\mathbf{X}) \text{ for } a \in \mathbb{R}_{>0}^{1 \times C \times 1 \times 1} \text{ and } b \in \mathbb{R}^{1 \times C \times 1 \times 1}$$

No need to include a bias before batch normalization

Batch Normalization - Train Fwd

Normalize channels of batch $\mathbf{X} \in \mathbb{R}^{N \times C \times H \times W}$ to have zero mean, unit variance:

Batch of input $x_1, \cdots, x_N$



$$\mathbf{X} = (\mathbf{X}_1, \cdots, \mathbf{X}_n, \cdots, \mathbf{X}_N) \in \mathbb{R}^{N \times C \times H \times W}$$

Note: For $\varepsilon \approx 0$, output is invariant to a positive channel-wise affine scaling of $\mathbf{X}$:

$$\text{BN}(a \odot \mathbf{X} + b) = \text{BN}(\mathbf{X}) \text{ for } a \in \mathbb{R}^{1 \times C \times 1 \times 1}_{>0} \text{ and } b \in \mathbb{R}^{1 \times C \times 1 \times 1}$$

No need to include a bias before batch normalization

Batch Normalization - Train Fwd

Normalize channels of batch $\mathbf{X} \in \mathbb{R}^{N \times C \times H \times W}$ to have zero mean, unit variance:

$$\hat{\mathbf{X}} = \frac{\mathbf{X} - \mu(\mathbf{X})}{\sqrt{\sigma^2(\mathbf{X}) + \varepsilon}}$$

with small hyperparameter $\varepsilon \in \mathbb{R}_{\geq 0}$ for stability and $\mu(\mathbf{X}), \sigma(\mathbf{X}) \in \mathbb{R}^{1 \times C \times 1 \times 1}$ where:

$$\mu(\mathbf{X}) = \frac{1}{NHW} \sum_{n,h,w} \mathbf{X}_{n,\cdot,h,w}, \quad \sigma^2(\mathbf{X}) = \frac{1}{NHW} \sum_{n,h,w} \left(\mathbf{X}_{n,\cdot,h,w} - \mu(\mathbf{X})\right)^2$$

Final output: $\mathbf{Y} = \text{BN}(\mathbf{X}) = \gamma \odot \hat{\mathbf{X}} + \beta$ for learnable $\gamma, \beta \in \mathbb{R}^{1 \times C \times 1 \times 1}$

Broadcasted elementwise multiplication

Note: For $\varepsilon \approx 0$, output is invariant to a positive channel-wise affine scaling of $\mathbf{X}$:

$\text{BN}(a \odot \mathbf{X} + b) = \text{BN}(\mathbf{X})$ for $a \in \mathbb{R}_{>0}^{1 \times C \times 1 \times 1}$ and $b \in \mathbb{R}^{1 \times C \times 1 \times 1}$

No need to include a bias before batch normalization

Batch Normalization - Backprop

View μ, σ as functions and differentiate through them to get gradients:

Gradient of loss w.r.t. $\mathbf{X}$

Remove mean of $\delta_{\hat{\mathbf{X}}}$

$$\delta_{\mathbf{X}} = \frac{1}{\sqrt{\sigma^2(\mathbf{X}) + \varepsilon}} \left(\delta_{\hat{\mathbf{X}}} - \mathbf{1} \odot \sum_{n,h,w} \frac{\delta_{\hat{\mathbf{X}}_{n,\cdot,h,w}}}{NHW} - \hat{\mathbf{X}} \odot \sum_{n,h,w} \delta_{\hat{\mathbf{X}}_{n,\cdot,h,w}} \frac{\hat{\mathbf{X}}_{n,\cdot,h,w}}{NHW} \right)$$

Remove approx projection of $\delta_{\hat{\mathbf{X}}}$ on $\hat{\mathbf{X}}$ (exact if $\varepsilon = 0$)

$$\delta_\gamma = \sum_{n,h,w} \delta_{\mathbf{Y}_{n,\cdot,h,w}} \hat{\mathbf{X}}_{n,\cdot,h,w}, \qquad \delta_\beta = \sum_{n,h,w} \delta_{\mathbf{Y}_{n,\cdot,h,w}}, \qquad \delta_{\hat{\mathbf{X}}} = \gamma \odot \delta_{\mathbf{Y}}$$

<u>Key takeaways:</u>

- $\delta_{\mathbf{X}} \perp \mathbf{1}$ and for $\varepsilon \approx 0$ we also have $\delta_{\mathbf{X}} \perp \mathbf{X}$
- Hence for $\mathbf{X} = \text{conv}(\mathbf{W}, \mathbf{Z}) + b$ we get $\delta_{\mathbf{W}} \perp \mathbf{W}$ and $\delta_b = 0$
- Without differentiation through μ, σ training might be numerically unstable

Batch Normalization - Inference

Requires a sufficiently large and uncorrelated batches to get good estimate μ, σ

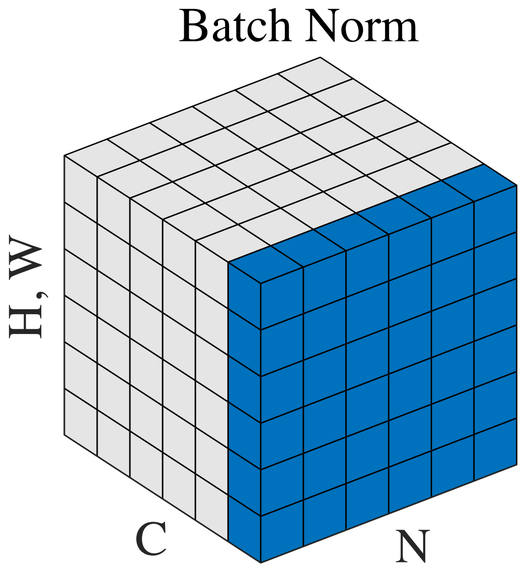
Batch dependency undesirable for inference:
- May not have an (uncorrelated) batch
- Prediction for one sample should not depend on others

Solution:
- Estimate $\hat{\mu} = \mathbf{E}[\mu]$ and $\hat{\sigma} = \mathbf{E}[\sigma]$ during training use $\hat{\mu}, \hat{\sigma}$ for inference
- Commonly use exponential moving averages in practice

Different behavior between train and test:
- In PyTorch use model.train() and model.eval() to switch mode

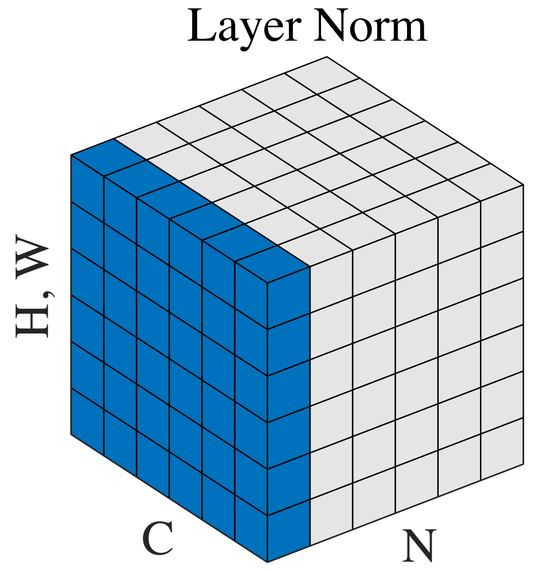
Normalization in General

Layer Normalization:
- Very common alternative, transformers and text data
- Compute μ and σ over C,H,W instead of N,H,W
- No batch dependency, use same form for inference



Benefits of Normalization:
- Stabilizes activation magnitudes / reduces initialization impact
- Augmentation effect from noisy μ, σ in batch norm
- Can down-weigh residual branches (see later)
- Stabilizes and speeds up training, allows larger learning rates

Used in almost all modern deep learning architectures
- Often inserted after every convolutional layer, before non-linearity



From arXiv:1803.08494

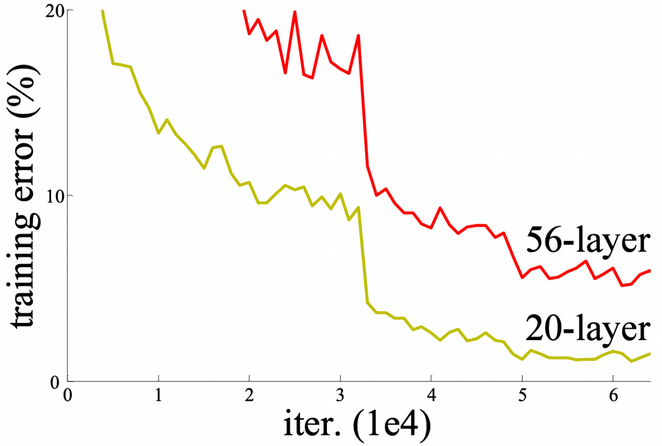
Residual Networks

Skip Connections and Residuals

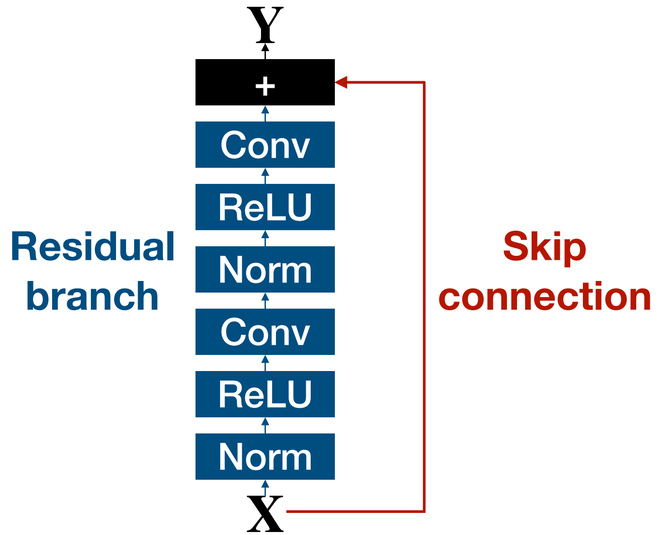
We expect lower train loss with more layers:
— Later layers only have to learn identity
— ResNet paper observed this is not always the case

ResNet solution:
— Add a skip connection around some layers
— Go from $\mathbf{Y} = F(\mathbf{X})$ to $\mathbf{Y} = R(\mathbf{X}) + \mathbf{X}$
— $R(\mathbf{X}) = \mathbf{Y} - \mathbf{X}$ is called a residual branch
— If size($\mathbf{Y}$) $\neq$ size($\mathbf{X}$): Need extra ops on skip connection

Skip connections fix the observed issue
— Enabled training of very deep networks, 100s of layers
— Used in almost all modern neural networks



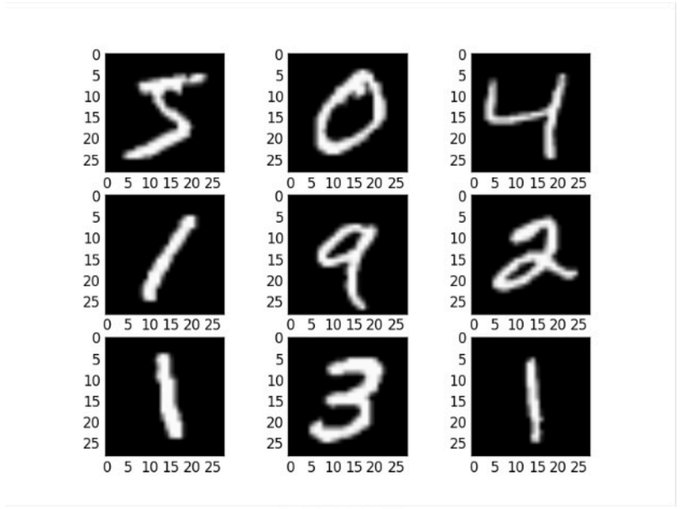
Observation from arXiv:1512.03385
Deeper (plain) networks harder to train



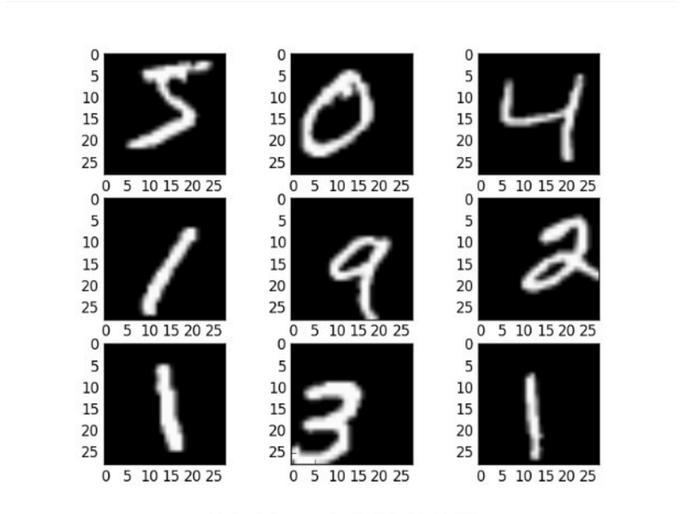
Example residual block

Data augmentation

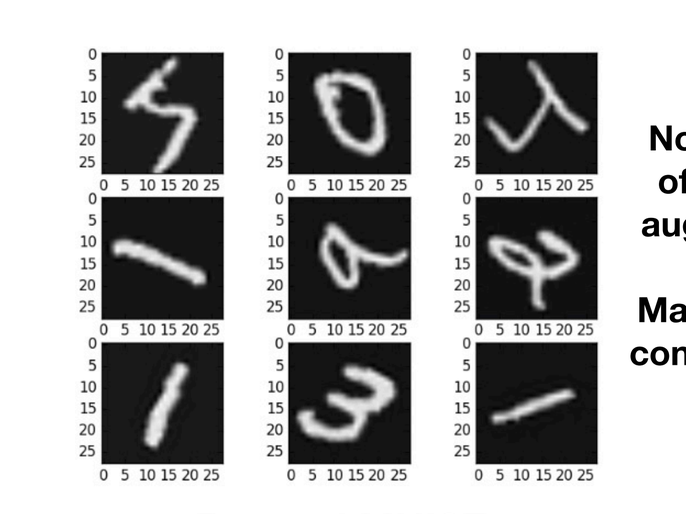
Data augmentation: generate new data from the data



MNIST



Shifted MNIST



Rotated MNIST

Note dangers of excessive augmentation!

May eventually confuse 6 and 9

Transformation $\tau : \mathbb{R}^d \rightarrow \mathbb{R}^d$ which preserves the labels (i.e., $y_x = y_{\tau(x)}$)

$$S = S_{train} \cup \{(\tau(x_i), y_i)\}_{i=1}^{n}$$

- We train on more data
- Encourages models to be invariant to τ
- It can be seen as regularization
- These transformations are task and dataset specific

Data augmentation: pictures can also be cropped, resized, or perturbed by a small amount of noise

(a) Original

(b) Crop and resize

(c) Crop, resize (and flip)

(d) Color distort. (drop)

(e) Color distort. (jitter)

(f) Rotate $\{90°, 180°, 270°\}$

(g) Cutout

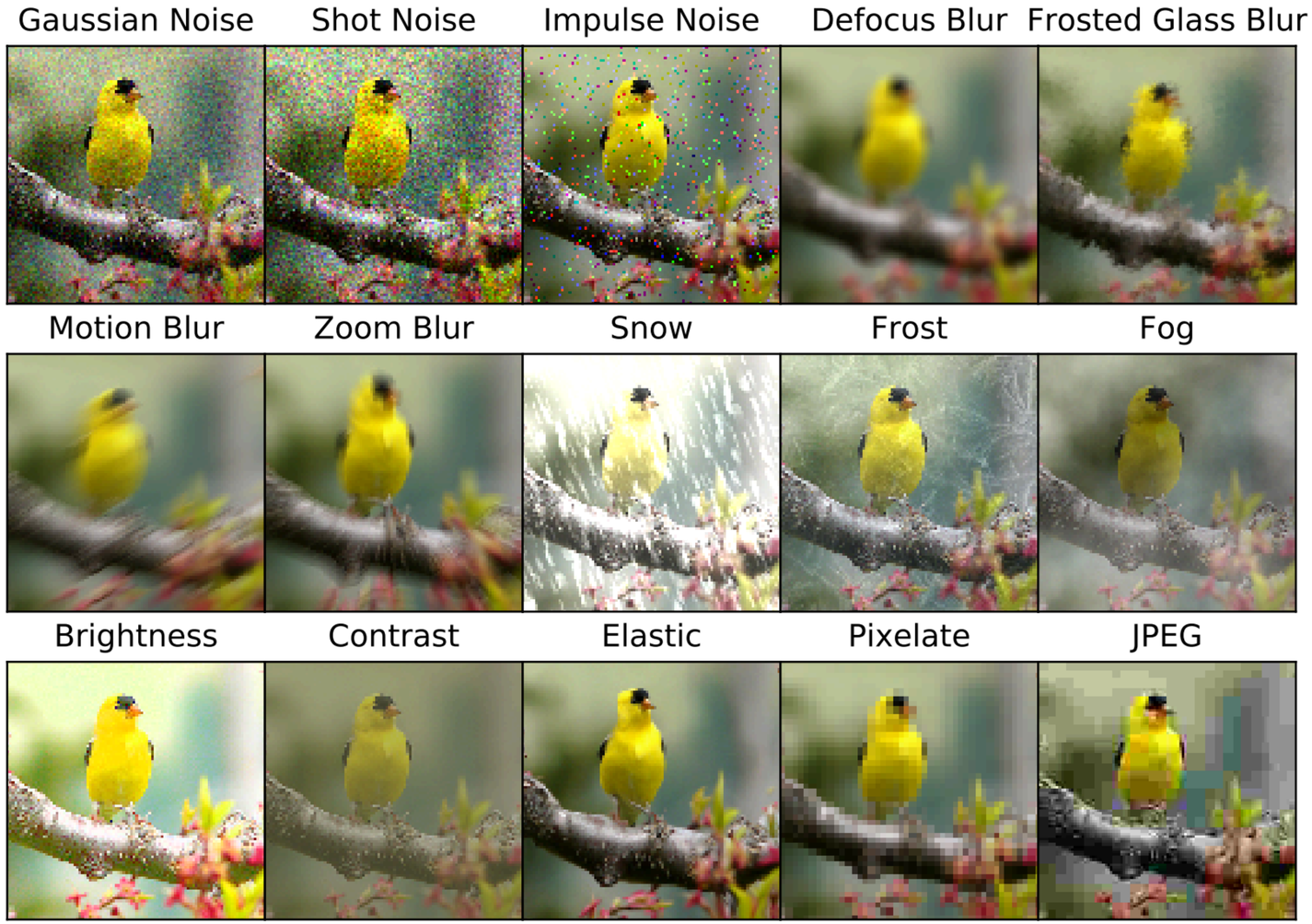
(h) Gaussian noise

(i) Gaussian blur

(j) Sobel filtering

T Chen, S Kornblith, M Norouzi, G Hinton. A simple framework for contrastive learning of visual representations. ICML, 2020

Data augmentation: generated corruptions



D. Hendrycks and T. Dietterich. Benchmarking Neural Network Robustness to Common Corruptions and Perturbations. ICLR, 2019

Weight Decay

Weight decay: ℓ_2-regularization for NNs

It is customary to regularize weights but not the bias terms:

$$\min \mathscr{L} + \frac{\mu}{2} \sum_l \|\mathbf{W}^{(l)}\|_F^2$$

Favors small weights and may help generalization and optimization

Optimization (GD):

$$\begin{aligned}\left(w_{i,j}^{(l)}\right)_{t+1} &= \left(w_{i,j}^{(l)}\right)_t - \gamma \nabla \mathscr{L} - \gamma\mu \left(w_{i,j}^{(l)}\right)_t \\ &= \underbrace{(1-\gamma\mu)}_{\text{Weight decay}} \left(w_{i,j}^{(l)}\right)_t - \gamma \nabla \mathscr{L}\end{aligned}$$

A. Krogh and J. A Hertz. A simple weight decay can improve generalization. NeurIPS, 1992
S. Bos and E Chug. Using weight decay to optimize the generalization ability of a perceptron. ICNN, 1996.

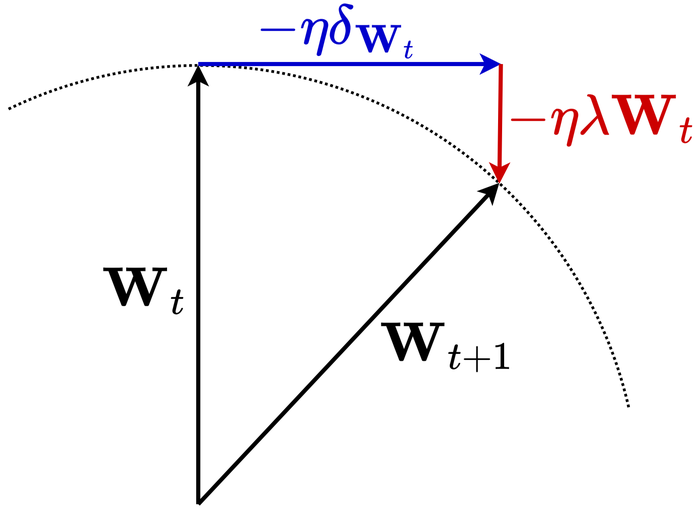
Weight decay + Normalization

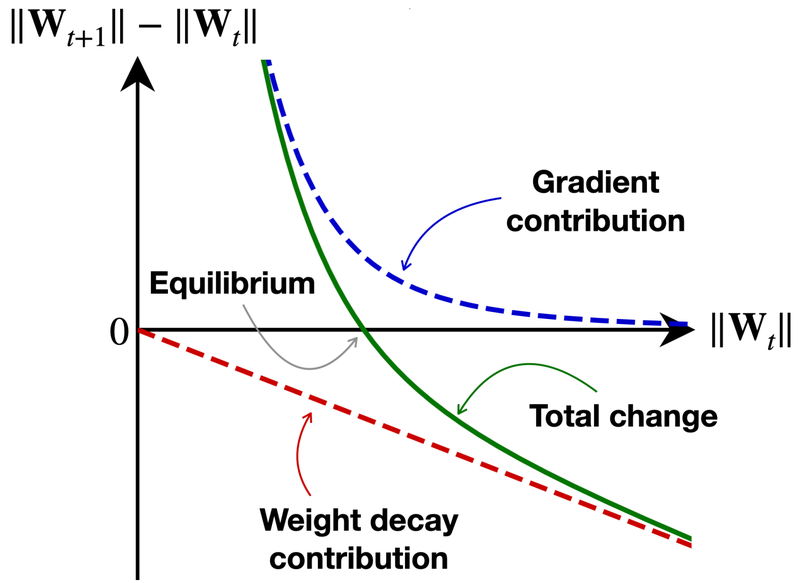
Weight decay interacts with normalization and GD

With batch normalization where $\varepsilon \approx 0$:
- $\text{BN}(\mathbf{WX}) = \text{BN}(\alpha\mathbf{WX})$ for $\alpha \in \mathbb{R}_{>0}$
- $\mathbf{W}$ scale invariant, **no direct regularization effect from WD**
- Note that some layers may not be normalized

Loss gradient and WD orthogonal:
- Loss gradient makes weight norm grow
- Weight decay decreases it
- Forces can balance out in weight norm equilibrium

WD changes how fast scale invariant weight vectors rotate:
- Effect is similar to modifying the learning rate for such weights
- May affect layers differently (layer-wise adjustment of lr)



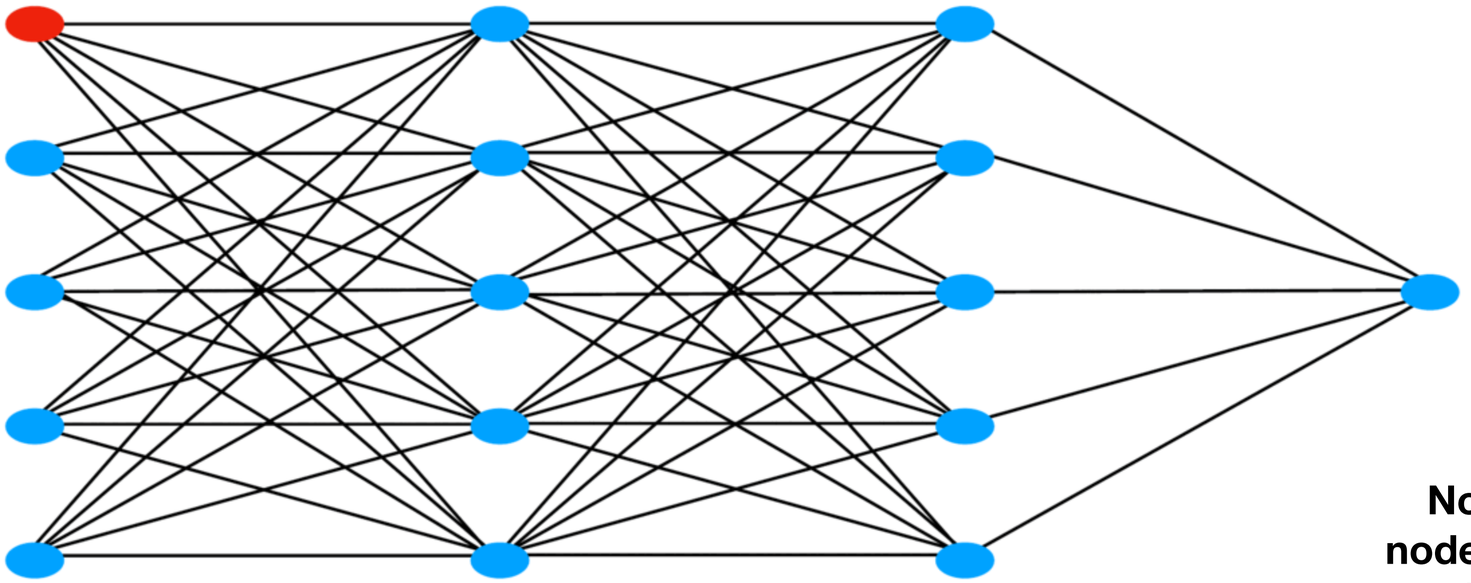


Wan, Ruosi, et al. "Spherical Motion Dynamics: Learning Dynamics of Normalized Neural Network using SGD and Weight Decay.", NeurIPS 2021

Dropout

Dropout: randomly drop nodes

Def: At each training step, retain with probability $p^{(l)}$ the nodes in layer (l):

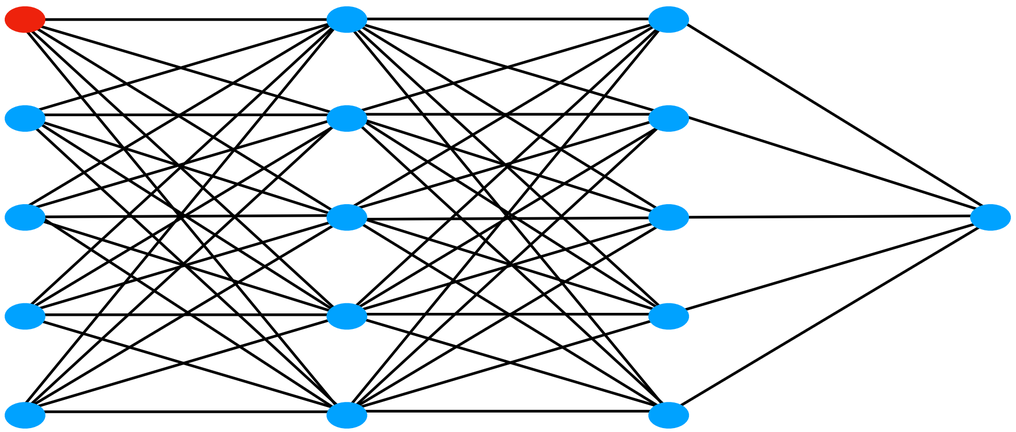


Note: In practice we drop nodes independently for each element of a mini-batch

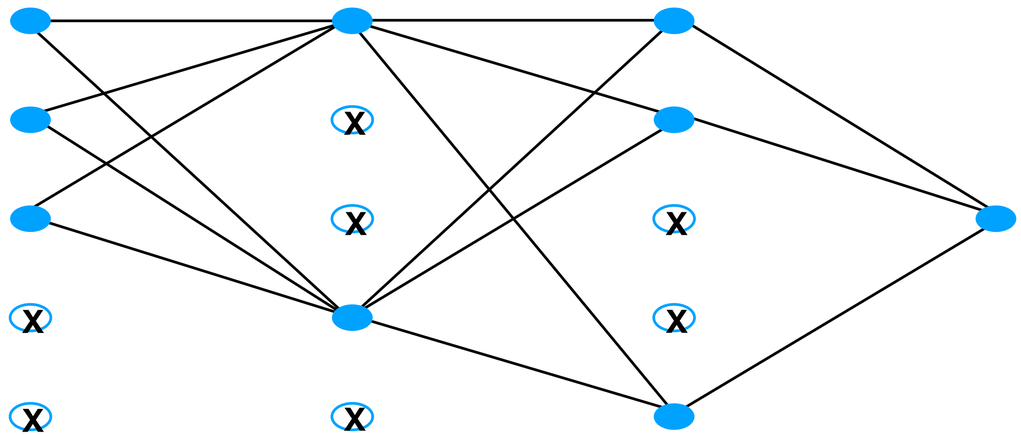
N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting, JMLR, 2014

Dropout: training phase

Def: At each training step, retain with probability $p^{(l)}$ the nodes in layer (l):



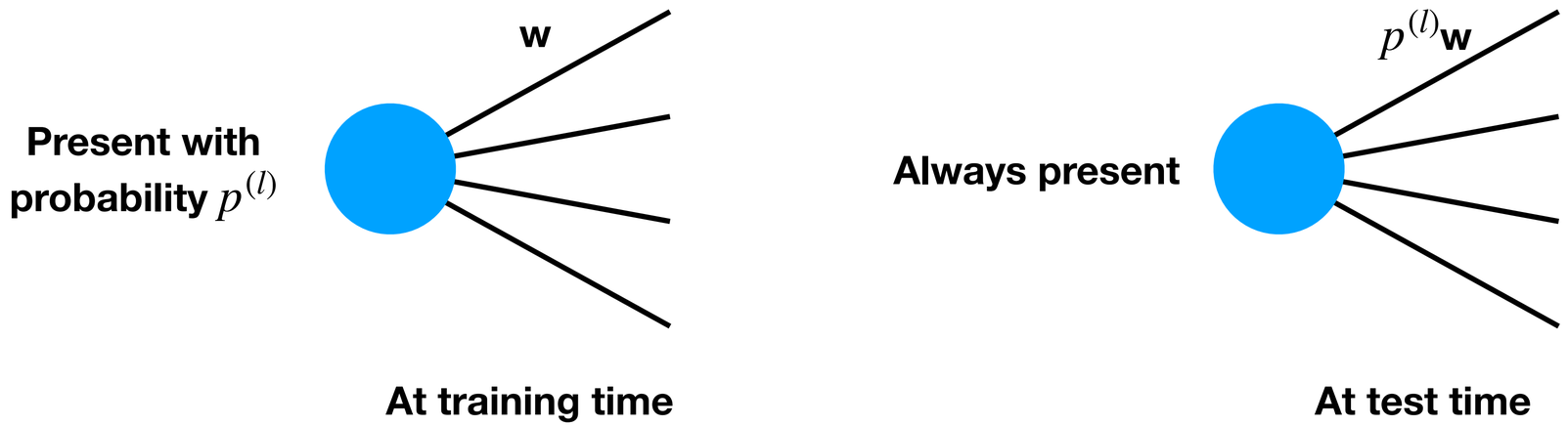
Original network



Random subnetwork

Run one step of SGD on the subnetwork and update the weights

Dropout: testing phase



When testing:

- use all nodes
- but scale each of them by the factor $p^{(l)}$

➡ The expected output (under the distribution to drop nodes at training time) is the same as the actual output as test time

Note: Variance is generally not preserved and as a result Dropout often works poorly with normalization

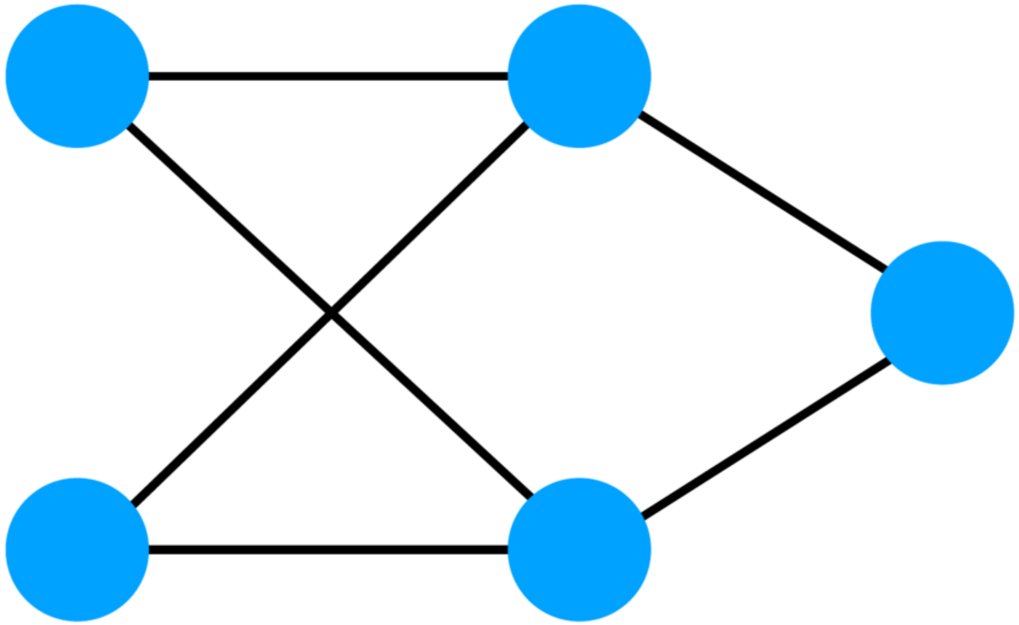
Remark: the weights rescaling can be implemented at training time: after each weight update, scale the weight by $1/p^{(l)}$ - way it is implemented in practice

Benefits of Dropout

- Experimentally, it helps to avoid overfitting by preventing *co-adaptations*
- It is a technique of ensemble averaging:
  - It can be done explicitly by training many NNs and then predicting using the average of the output

    ➡ *bagging* - impractical
  - It can be done implicitly by training a collection of 2^{KL} subnetworks with weight sharing, where each subnetwork is trained very rarely

    ➡ *dropout*



All the subnetworks of a given simple network

Conclusion

Entangled effects of various methods: CIFAR10

model	# parameters	Random crop	Weight decay	Train accuracy	Test accuracy
Inception	1,649,402	Yes	Yes	100.0	89.05
		Yes	No	100.0	89.31
		No	Yes	100.0	86.03
		No	No	100.0	85.75
Inception w/0 BatchNorm	1,649,402	No	Yes	100.0	83.00
		No	No	100.0	82.00
Alexnet	1,387,786	Yes	Yes	99.90	81.22
		Yes	No	99.82	79.66
		No	Yes	100.0	77.36
		No	No	100.0	76.07
MLP 3x512	1,735,178	No	Yes	100.0	53.35
		No	No	100.0	52.39
MLP 1x512	1,209,866	No	Yes	99.80	50.39
		No	No	100.0	50.51

C Zhang, S Bengio, M Hardt, B Recht, O Vinyals. Understanding deep learning requires rethinking generalization. ICLR, 2017

Entangled effects of various methods: ImageNet

model	Data aug	Dropout	Weight decay	Batch Norm	Skip Connections	Top-1 train	Top-1 test
ResNet200 (v2)	Yes	No	Yes	Yes	Yes	?	79.9
Inception (v3)	Yes	Yes	Yes	Yes	No	92.18	77.84
	Yes	No	No	Yes	No	92.33	72.95
	No	No	Yes	Yes	No	90.60	67.18(72.57)
	No	No	No	Yes	No	99.53	59.80(63.16)
VGG19	Yes	Yes	Yes	No	No	?	72.7

( ) : best test accuracy during training, i.e., with early stopping

C Zhang, S Bengio, M Hardt, B Recht, O Vinyals. Understanding deep learning requires rethinking generalization. ICLR, 2017