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

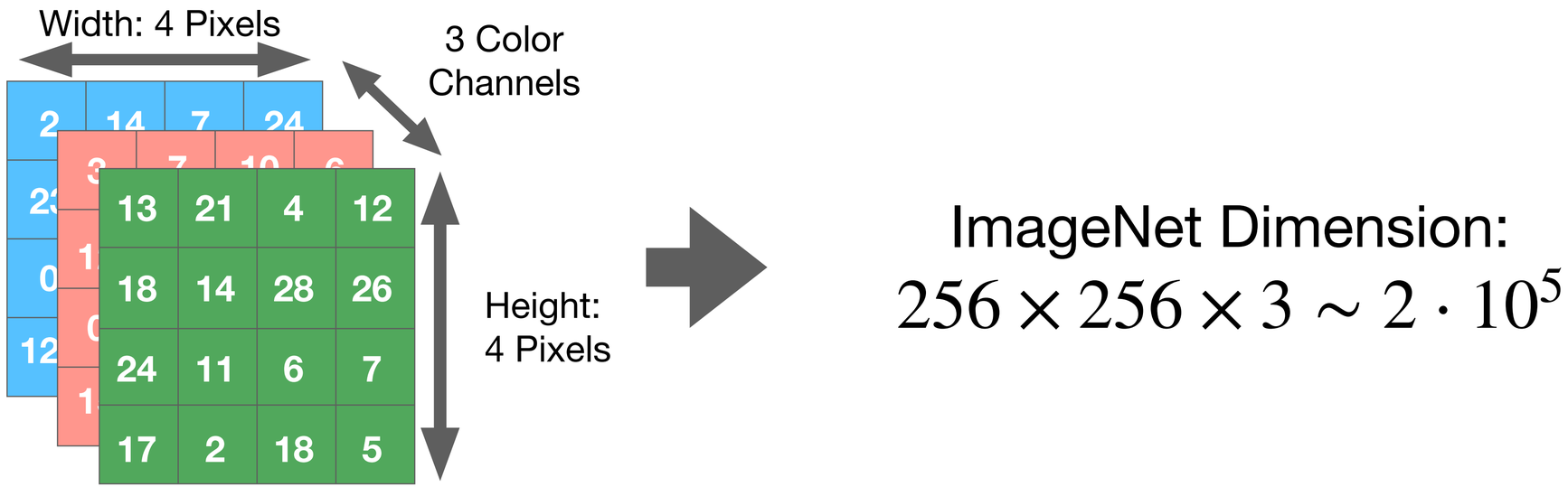
Machine Learning Course - CS-433
Nov 16, 2021
Nicolas Flammarion

EPFL

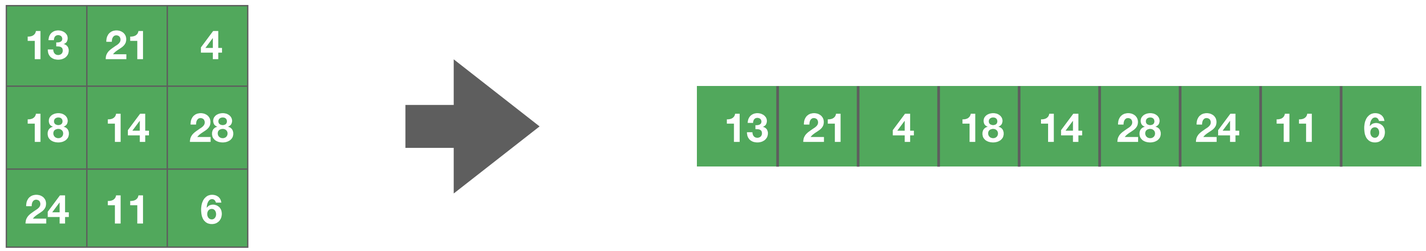
# Convolutional Nets

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

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




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

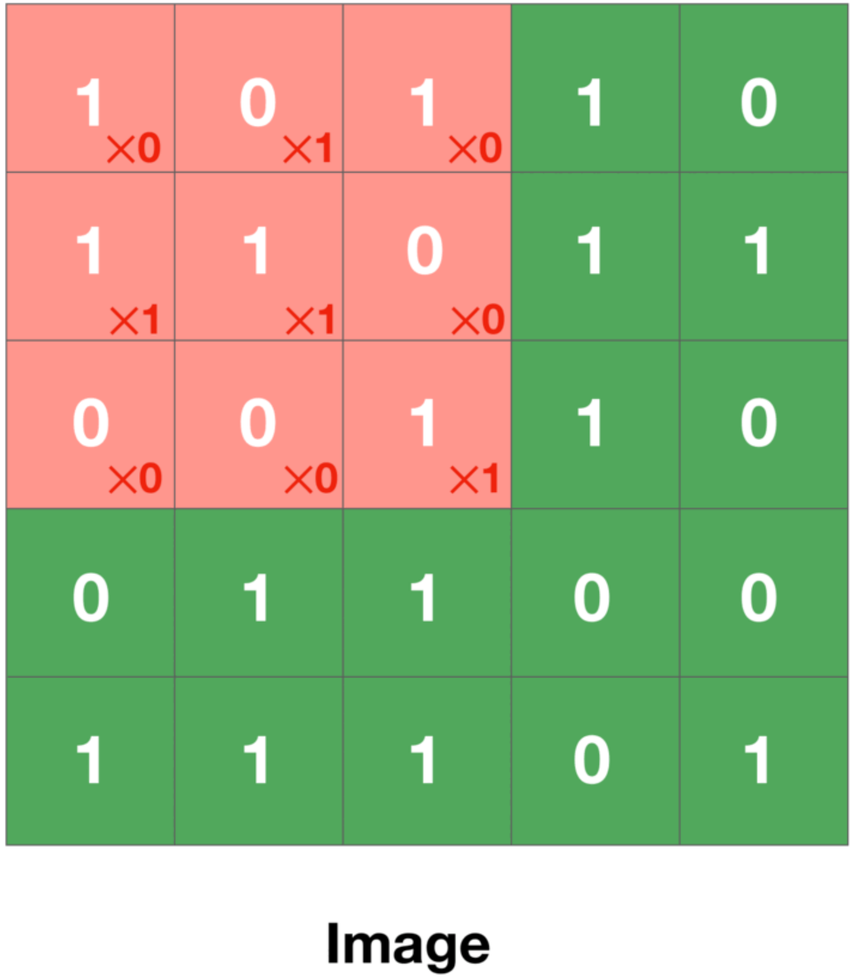


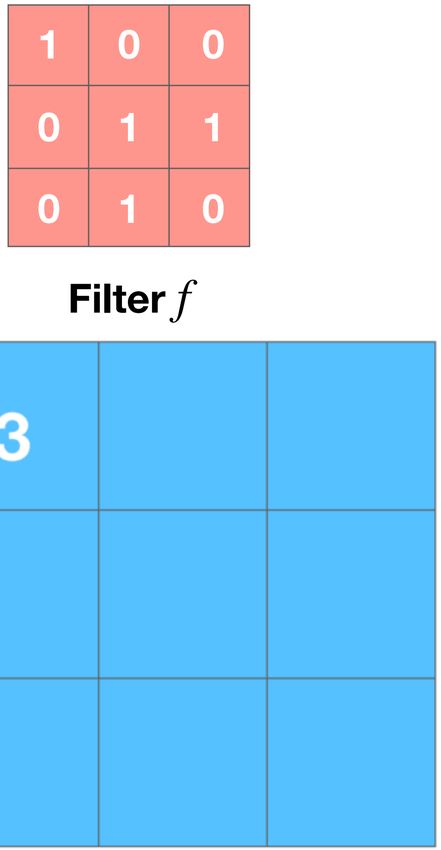

Flattening of a $3 \times 3$ picture into a $1 \times 9$ vector

# Convolutional layer

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

- 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 $(n, m)$
- We should use the same filter at every position - weight sharing

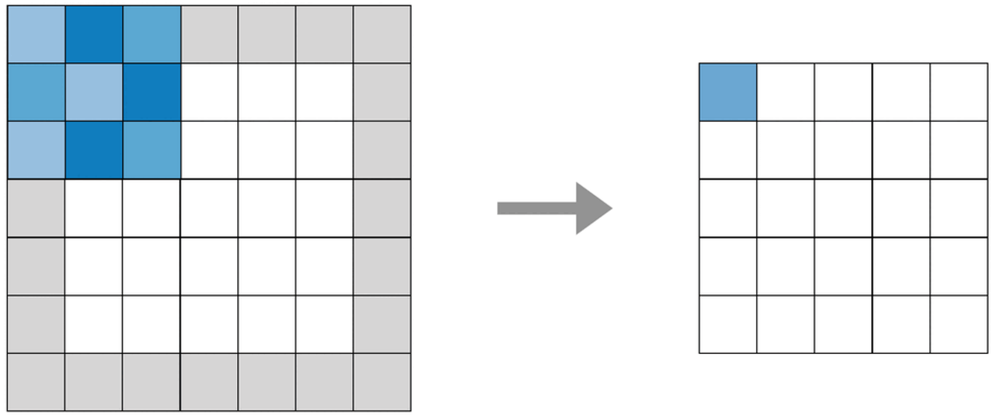



➡ NNs structure is sparse and local with few parameters

# Handling of borders

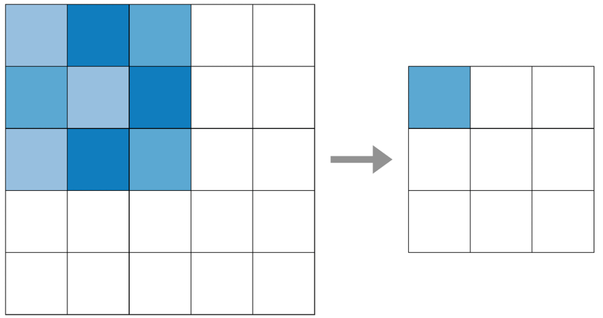
Zero padding:



Add zeros to each side of the boundaries of the input

➡ Convolved feature is of the same dimensionality as the input

Valid padding:

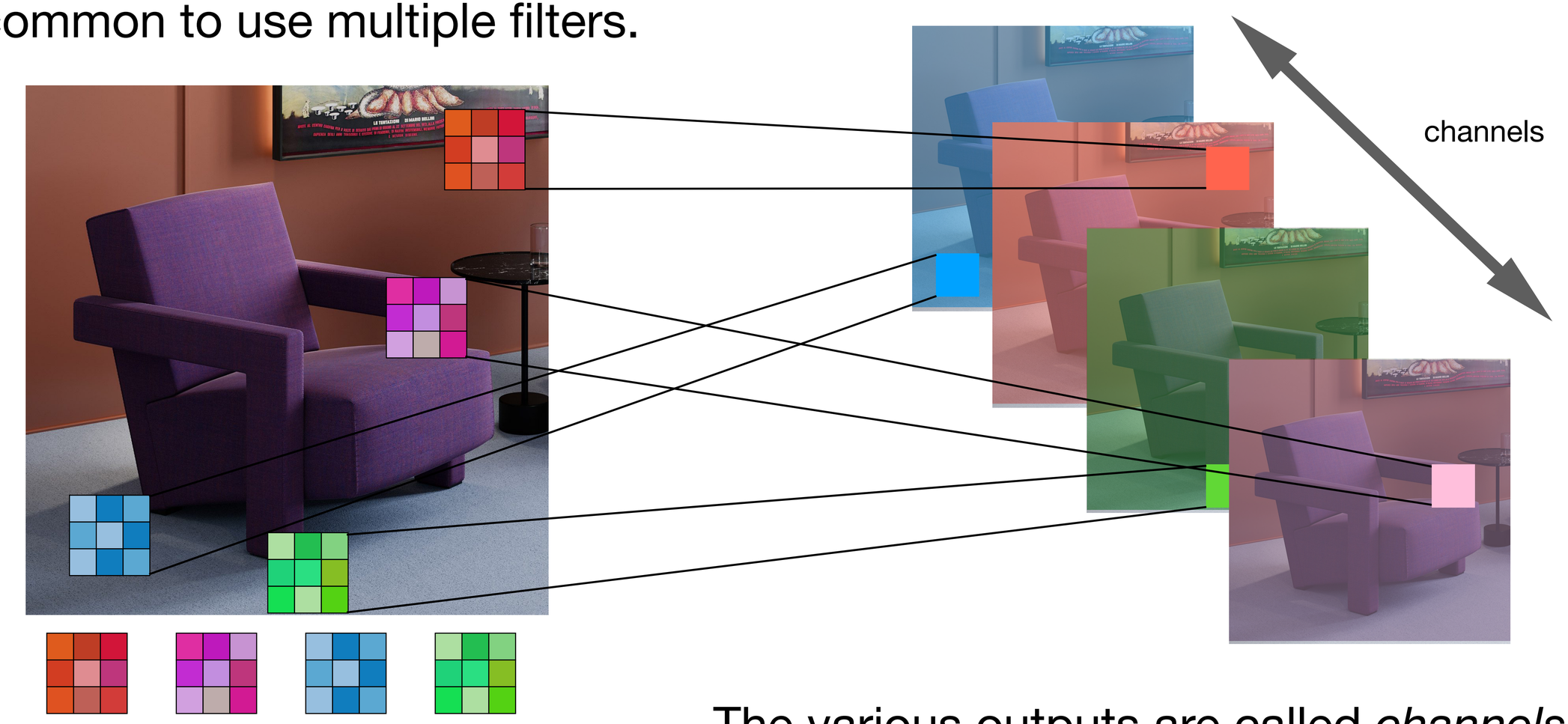


Perform the convolution only for positions so that the whole filter lies inside the original data

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

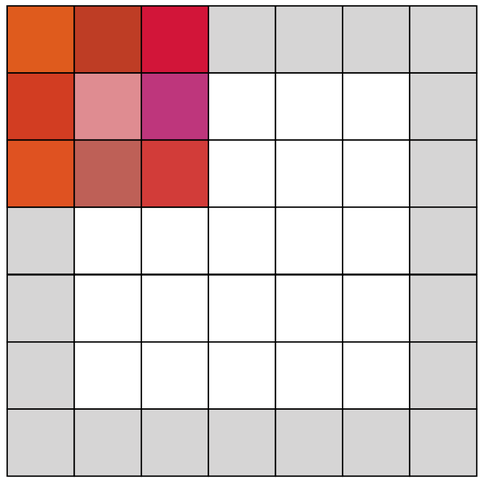
# Multiple Channels

It is common to use multiple filters.

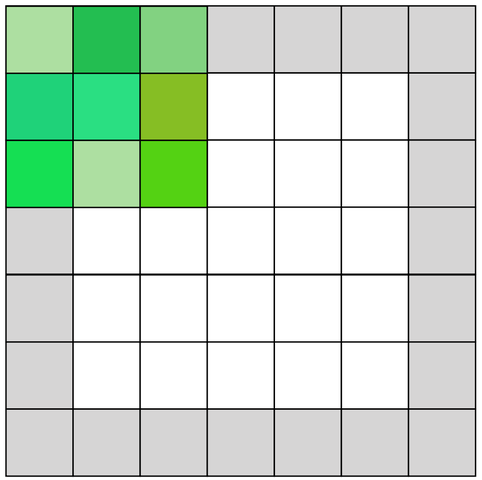



The various outputs are called *channels*

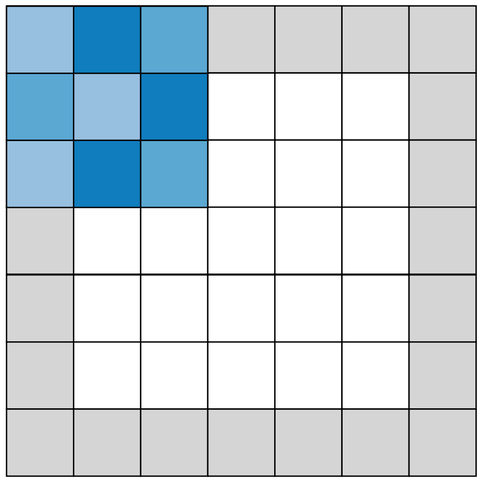
# Convolution on input with multiple channels



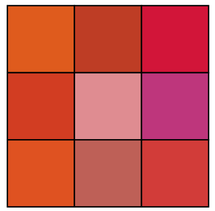
Input Channel #1



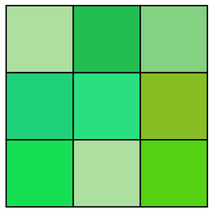
Input Channel #2



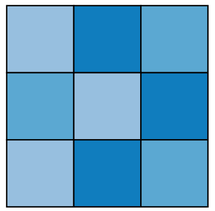
Input Channel #3



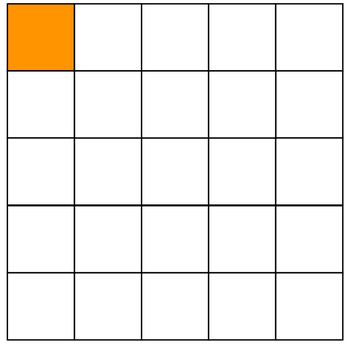
Filter Channel #1



Filter Channel #2



Filter Channel #3



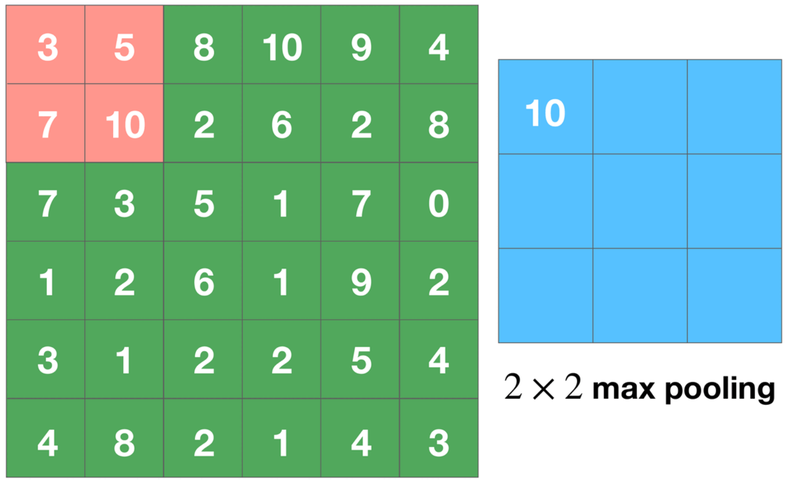
Output

120 + -75 + 205 + 10 = 260

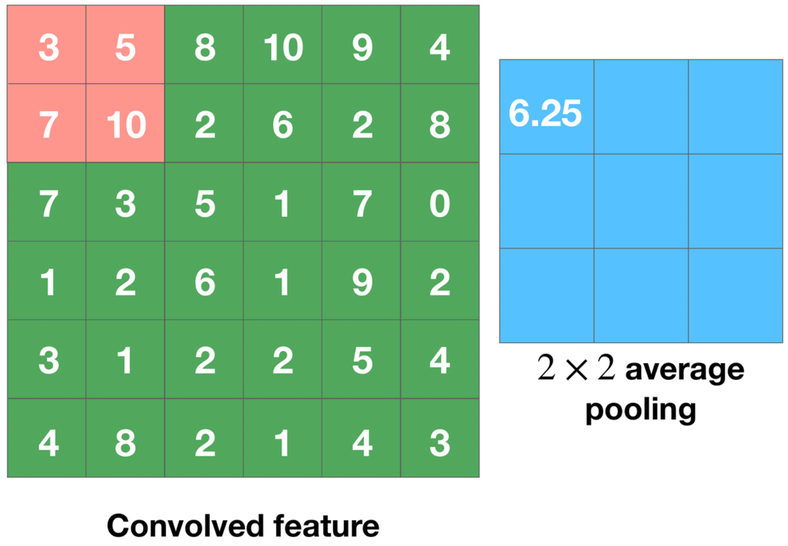
Bias

# Pooling: often applied after the convolutional layer

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

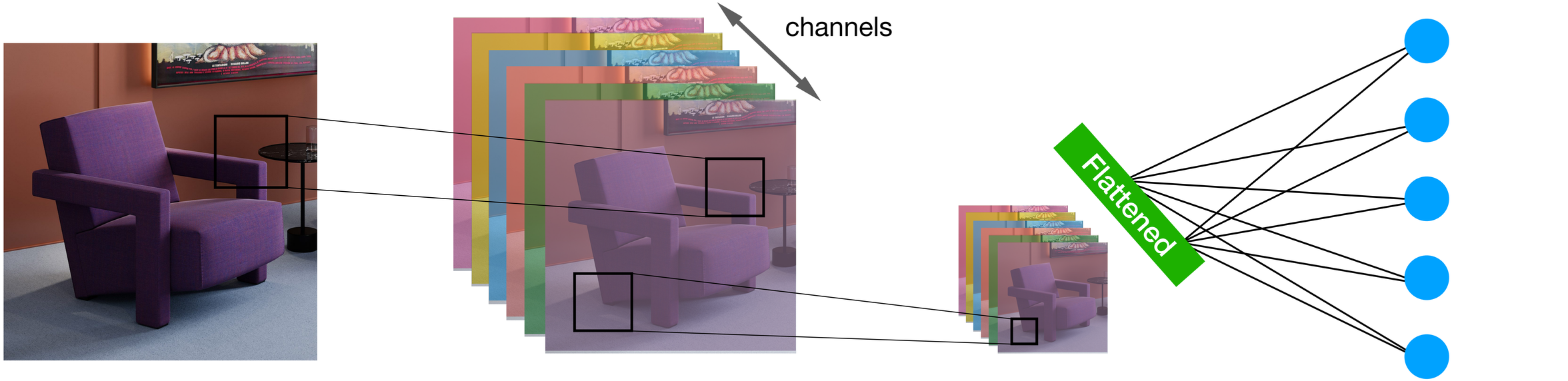



Average pooling: returns the average value of the portion of the convolved feature covered by the kernel



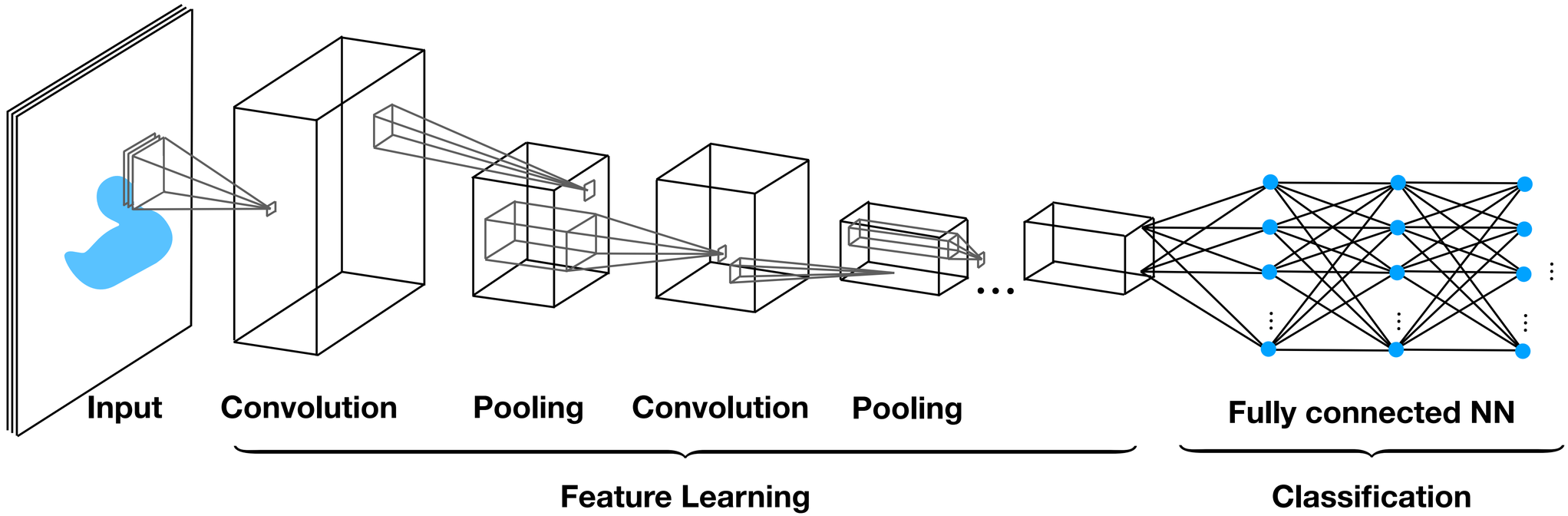

Pooling is a downsampling operation which reduces the spatial size of the convolved feature

# Convolutional NNs are composed of convolutional and pooling layers




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

# Convolutional NNs are composed of succession of convolutional and pooling layers




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

# 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

# Regularization

# Weight decay: $\ell_2$-regularization for NNs

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

$$\min \quad L + \frac{\mu}{2} \sum_{l=1}^{L+1} \|\mathbf{W}^{(l)}\|_F^2$$

➡ It favors small weights and help generalization and optimization

Optimization (GD):

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

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 constraint: $\ell_2$-regularization for NNs

It is also possible to constrain the value of the weight (less used in practice):

$$\min_{\forall l \ \|\mathbf{W}^{(l)}\|_F^2 \leq r} \quad L$$

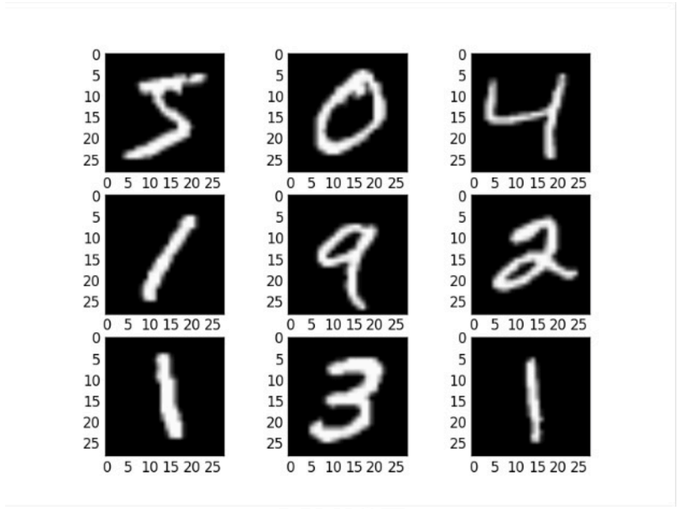
It can be minimized with projected gradient descent

These two formulations are very close (they are even equivalent for convex functions):

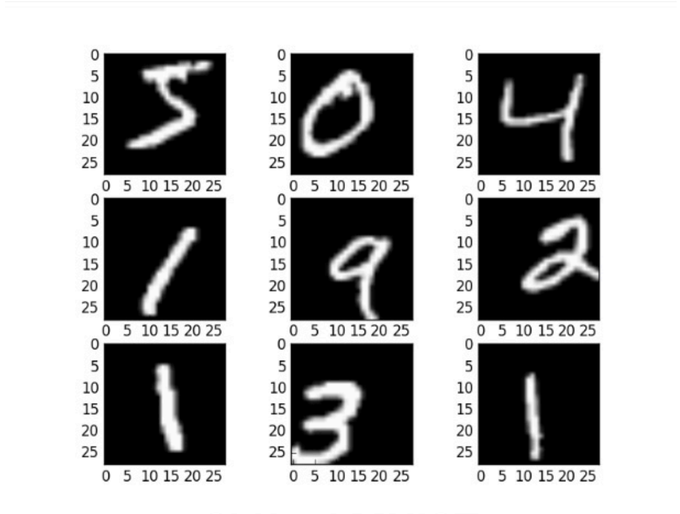
- Penalization is simpler to implement
- Constrained formulation is easier to interpret

# Data augmentation

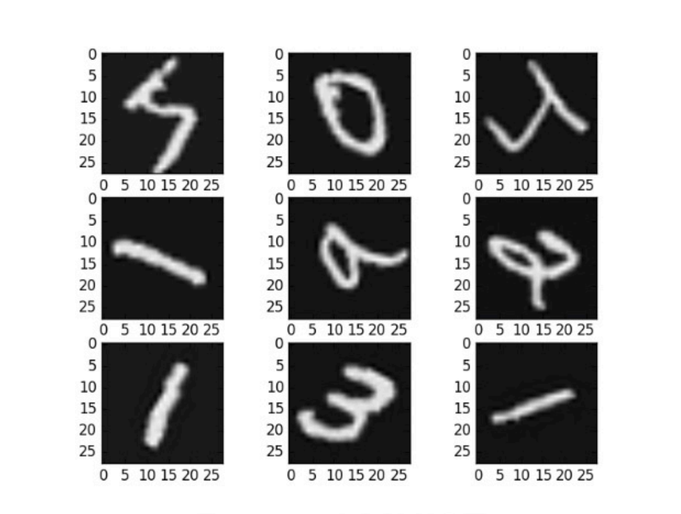
# Data augmentation: generate new data from the data



**MNIST**



**Shifted MNIST**



**Rotated MNIST**

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
- This guarantees that the prediction is invariant to $\tau$
- 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

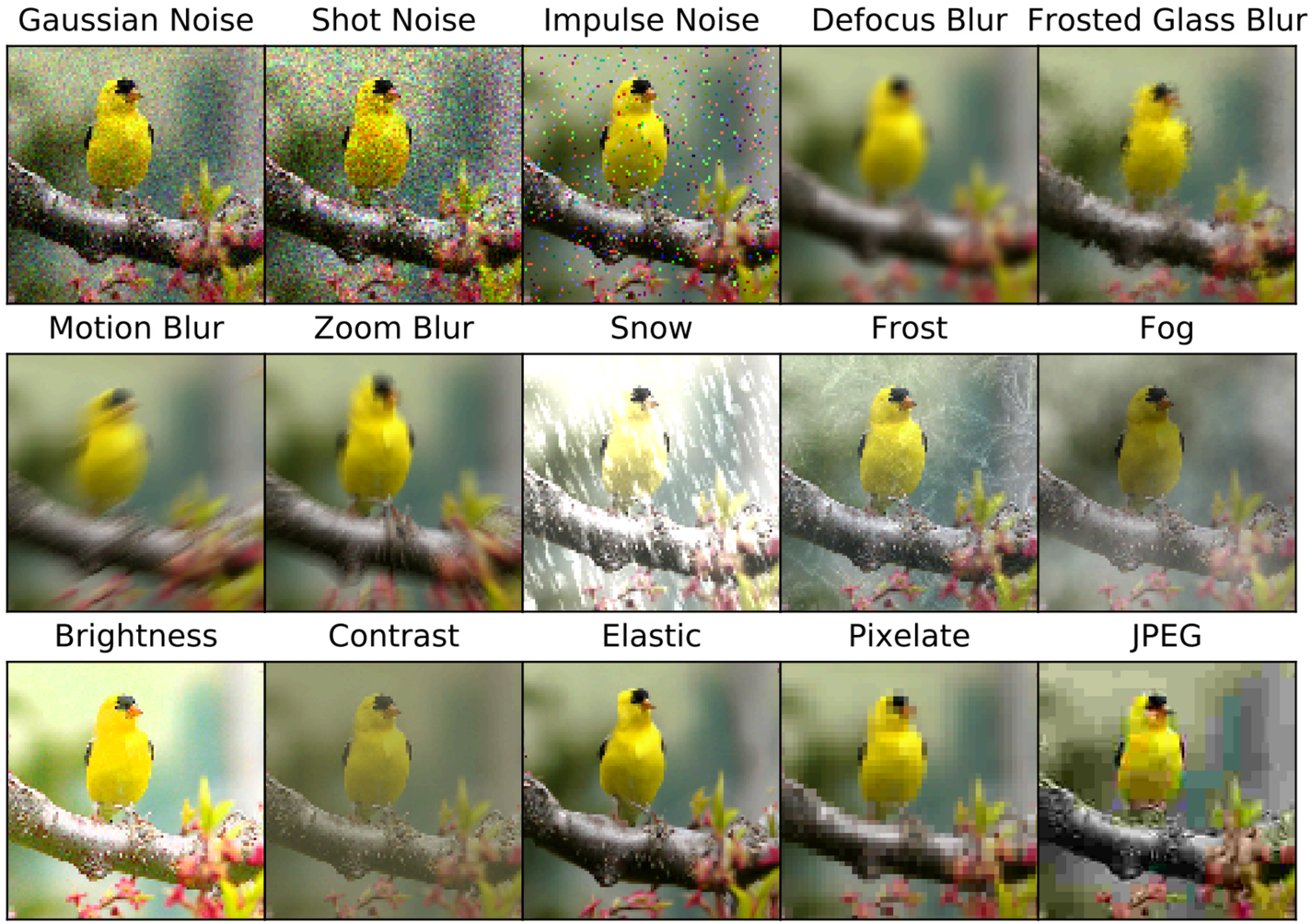
(h) Gaussian noise

(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

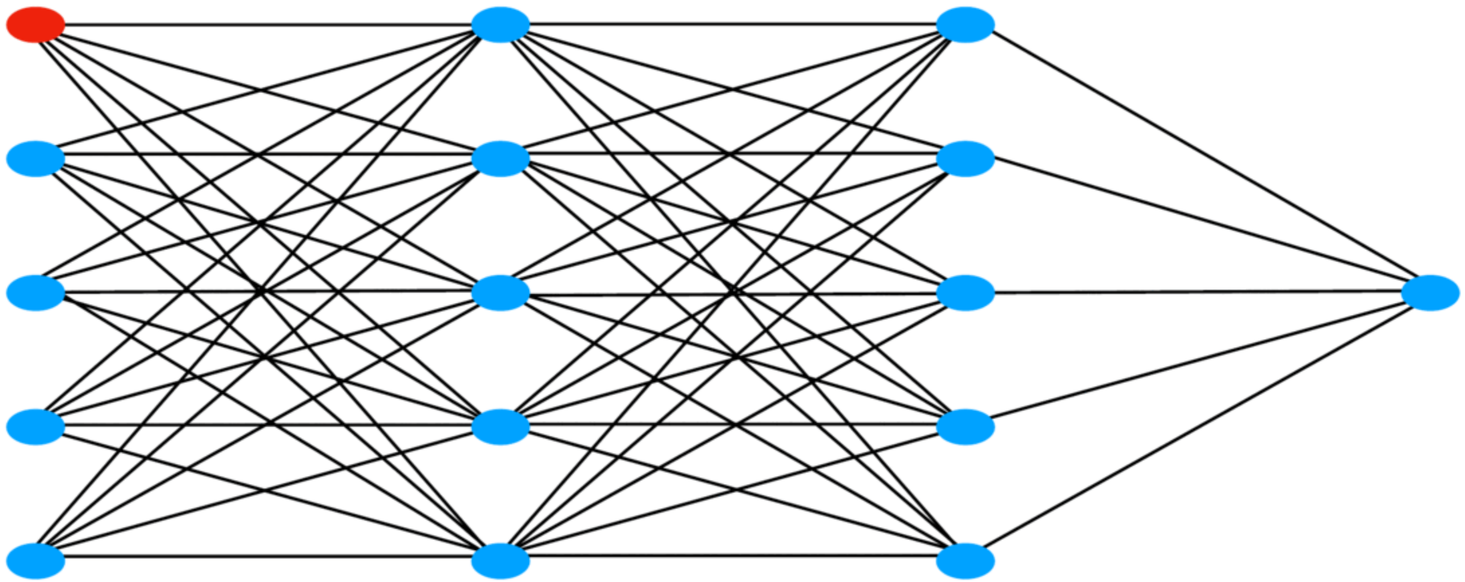



D. Hendrycks and T. Dietterich. Benchmarking Neural Network Robustness to Common Corruptions and Perturbations. ICLR, 2019

# Dropout

# Dropout: randomly drop nodes

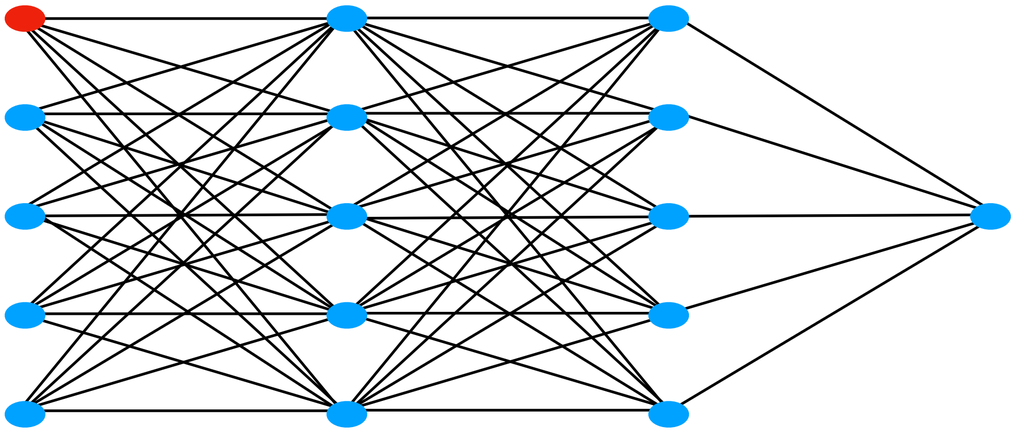
Def: At each training step, retain with probability $p^{(l)}$ the nodes in layer $(l)$ (typically $p^{(0)} = 0.8$ and $p^{(l)} = 0.5$ for $l \geq 1$)



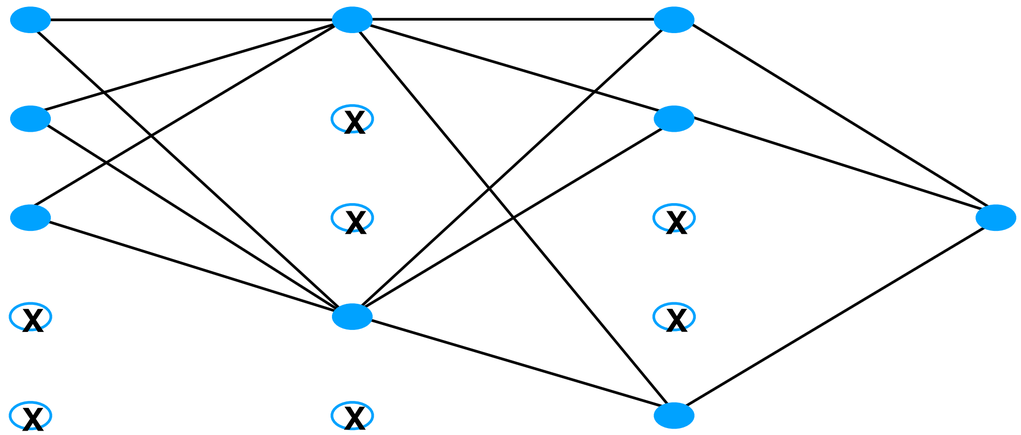
N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting, JMLR, 2014

# Dropout: training phase

Def: At each training step, retain with probability $p^{(l)}$ the nodes in layer $(l)$ (typically $p^{(0)} = 0.8$ and $p^{(l)} = 0.5$ for $l \geq 1$)



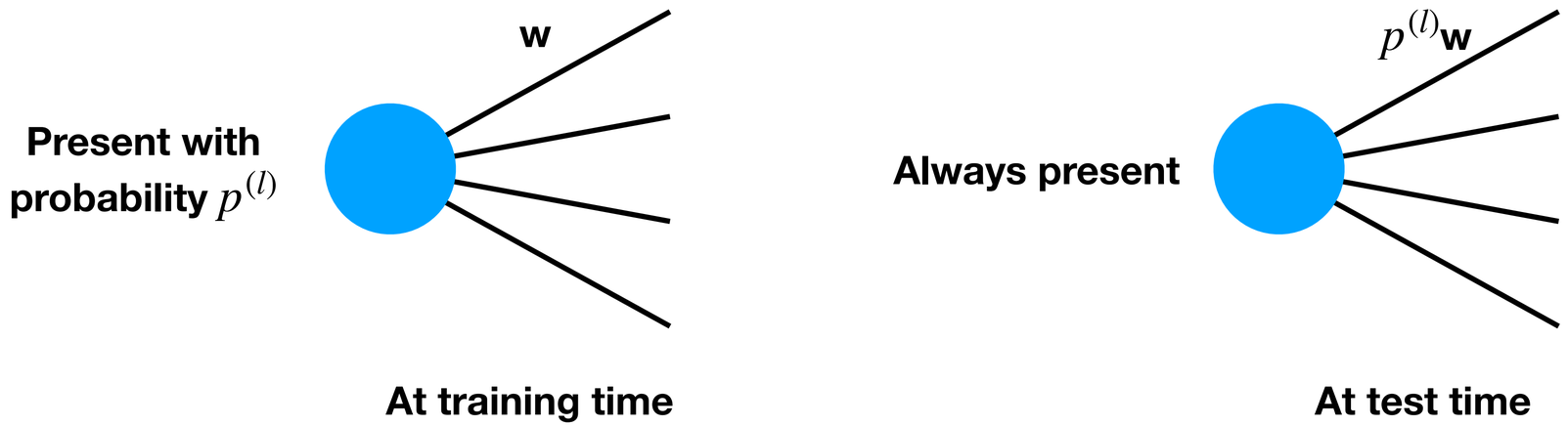
**Original network**



**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

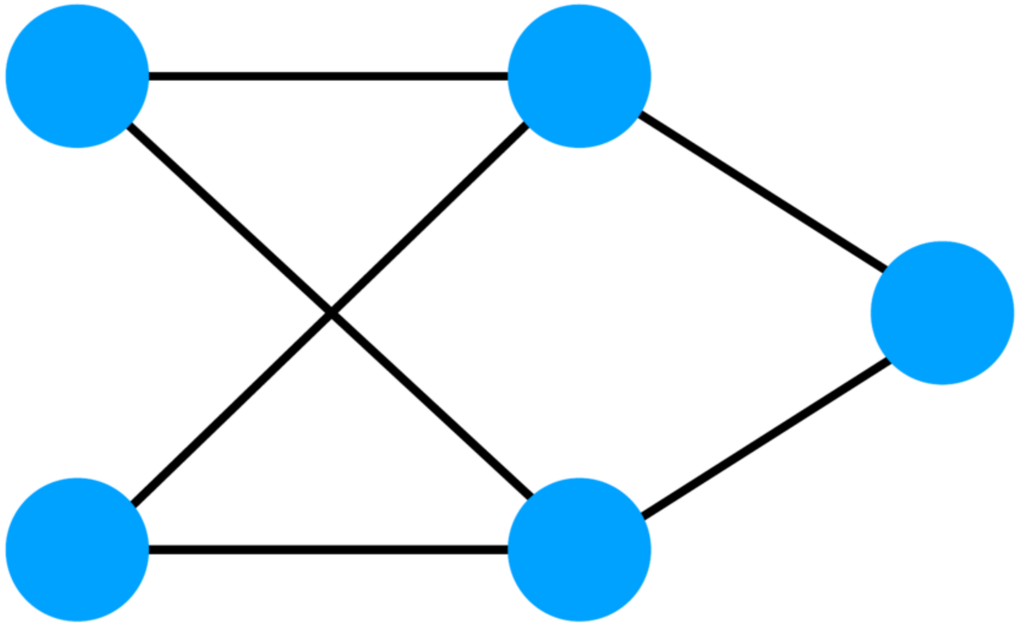
Rmk: the weights rescaling can be implemented at training time: after each weight update, scale the weight by $1/p^{(l)}$ - way it is implemented in practice

# 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**

# Batch Normalization

# Batch normalization makes training fast and stable

Train: normalize each layer's input using its mean and its variance over a batch:

$$\bar{x}_n^{(l)} = \frac{x_n^{(l)} - \mu_B^{(k)}}{\sqrt{(\sigma_B^{(l)})^2 + \varepsilon}}$$

where $\mu_B^{(l)} = \frac{1}{m}\sum_{n=1}^{m} x_n^{(l)}$ and $(\sigma_B^{(l)})^2 = \frac{1}{m}\sum_{n=1}^{m}(x_n^{(l)} - \mu_B)^2$ for minibatch $B$ of size $m$

Then output $$\hat{x}_n^{(l)} = \gamma^{(l)}\bar{x}_n^{(l)} + \beta^{(l)}$$

where $\gamma^{(l)}$ and $\beta^{(l)}$ are parameters which are also learnt using SGD

Test: Use the above formula with $\mu^{(l)}$ and $\sigma^{(l)}$ computed during the training (which approximate the true population means and variance)

Why: BN makes the training faster and allow the use of larger learning rate

S, Ioffe, C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. ICML 2015

# 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 | Top-1 train | Top-1 test |
|---|---|---|---|---|---|
| Inception | Yes | Yes | Yes | 92.18 | 77.84 |
| | Yes | No | No | 92.33 | 72.95 |
| | No | No | Yes | 90.60 | 67.18(72.57) |
| | No | No | No | 99.53 | 59.80(63.16) |

**( ) : 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