

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

Machine Learning Course - CS-433

Nov 16, 2021

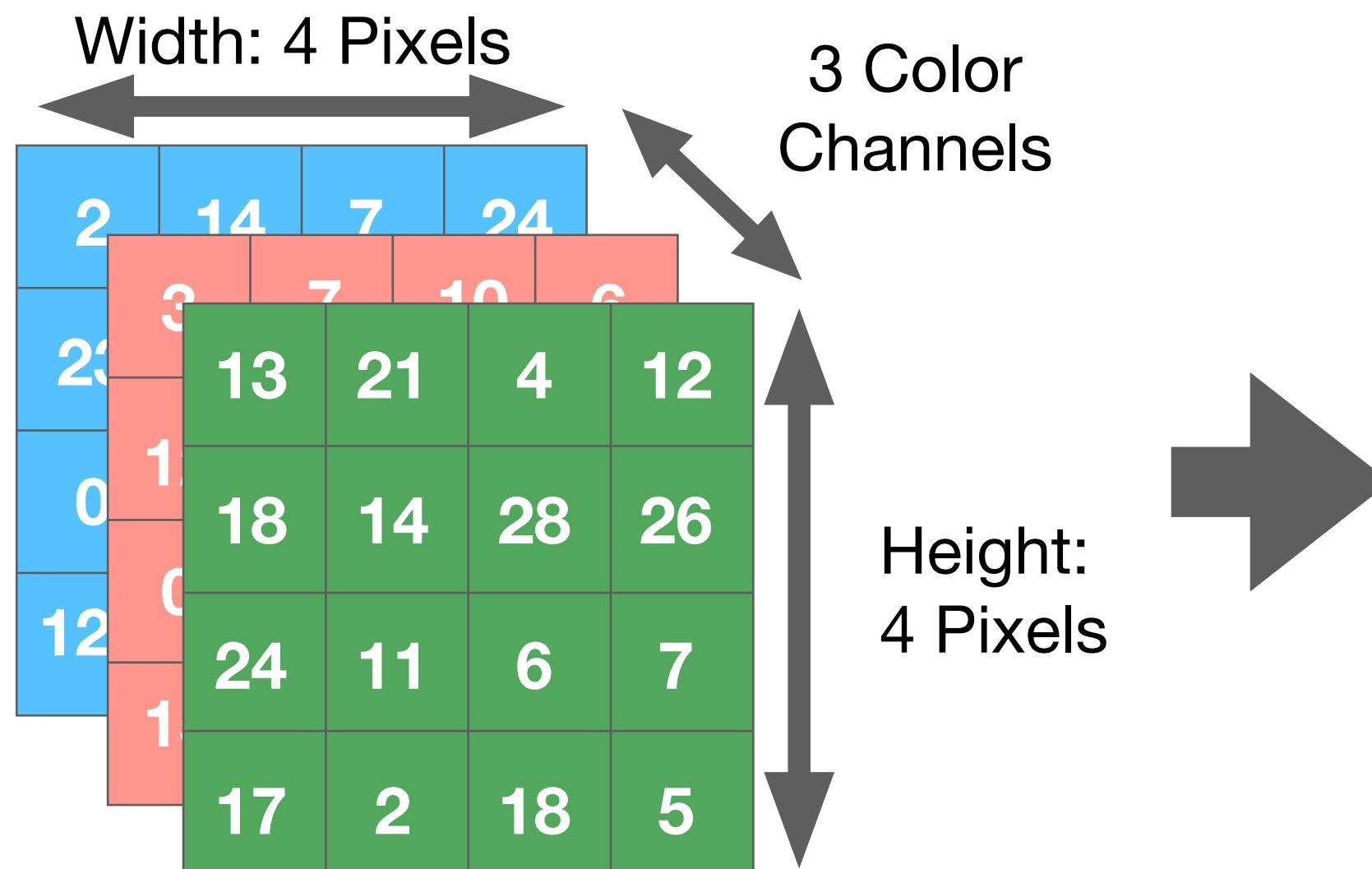
Nicolas Flammarion



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



ImageNet Dimension:
 $256 \times 256 \times 3 \sim 2 \cdot 10^5$

- 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

Convolutional layer

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

- We consider local filter: $f_{k,l} \neq 0$ for small values of $|k|$ and $|l|$
→ $x_{n,m}^{(1)}$ 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

1 <small>×0</small>	0 <small>×1</small>	1 <small>×0</small>	1	0
1 <small>×1</small>	1	0 <small>×0</small>	1	1
0 <small>×0</small>	0 <small>×0</small>	1 <small>×1</small>	1	0
0	1	1	0	0
1	1	1	0	1

0	1	0
1	1	0
0	0	1

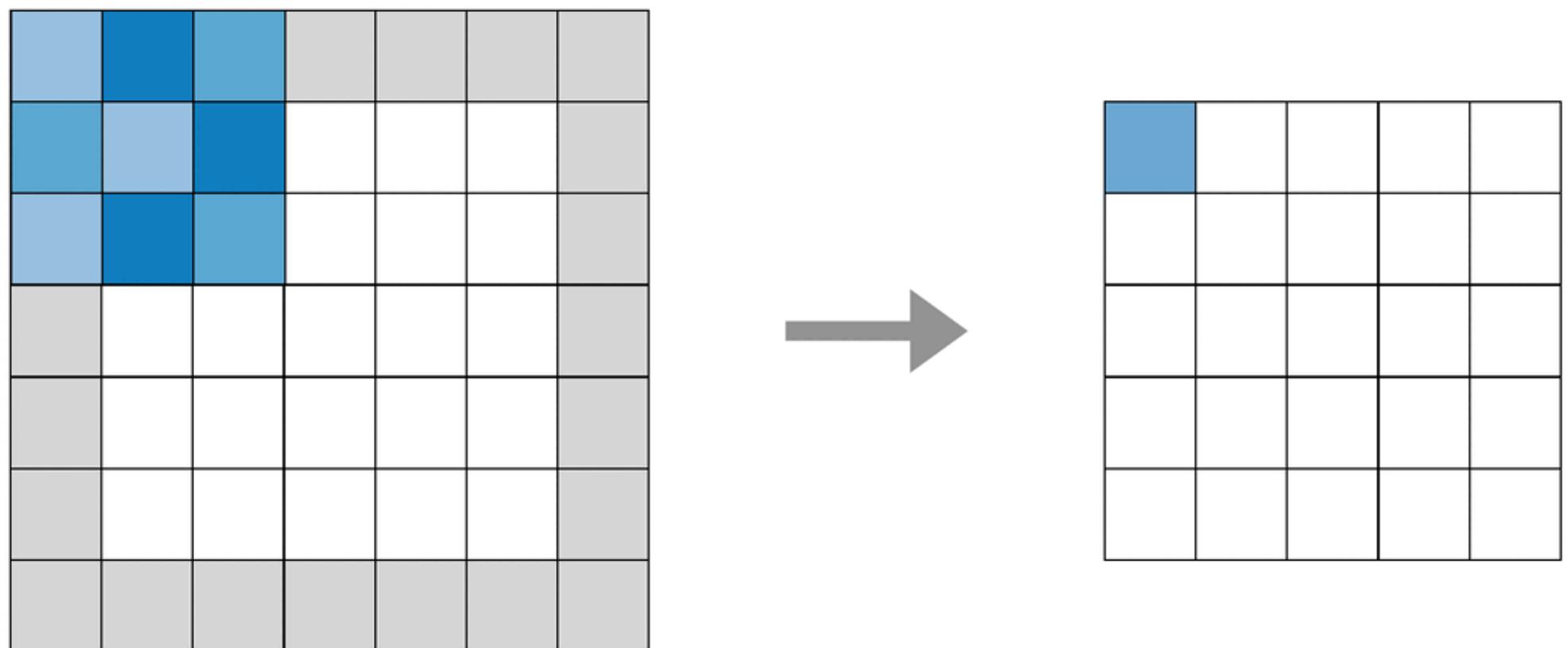
Filter f

3		

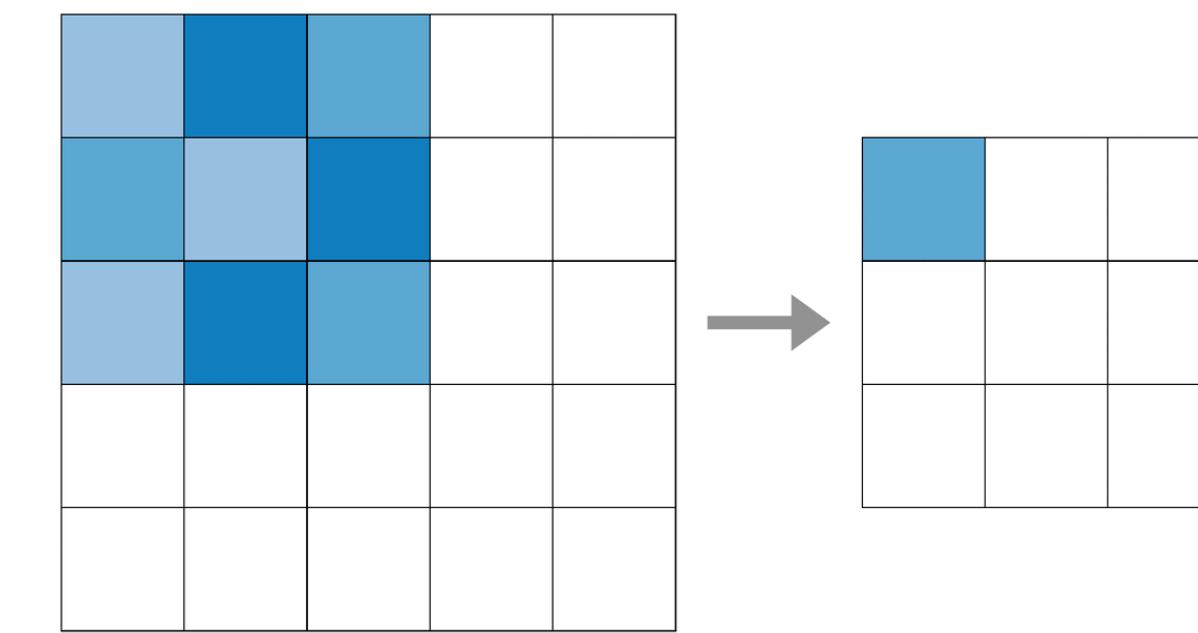
Convolved Feature

Handling of borders

Zero padding:



Valid padding:



Add zeros to each side of the boundaries of the input

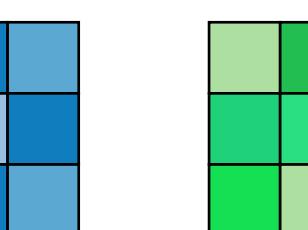
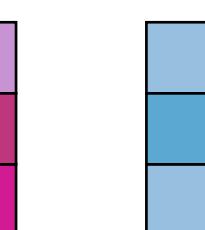
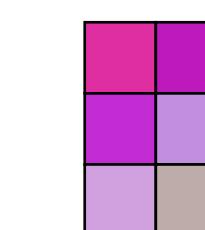
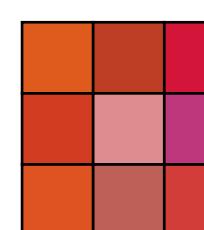
- Convolved feature is of the same dimensionality as the input

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

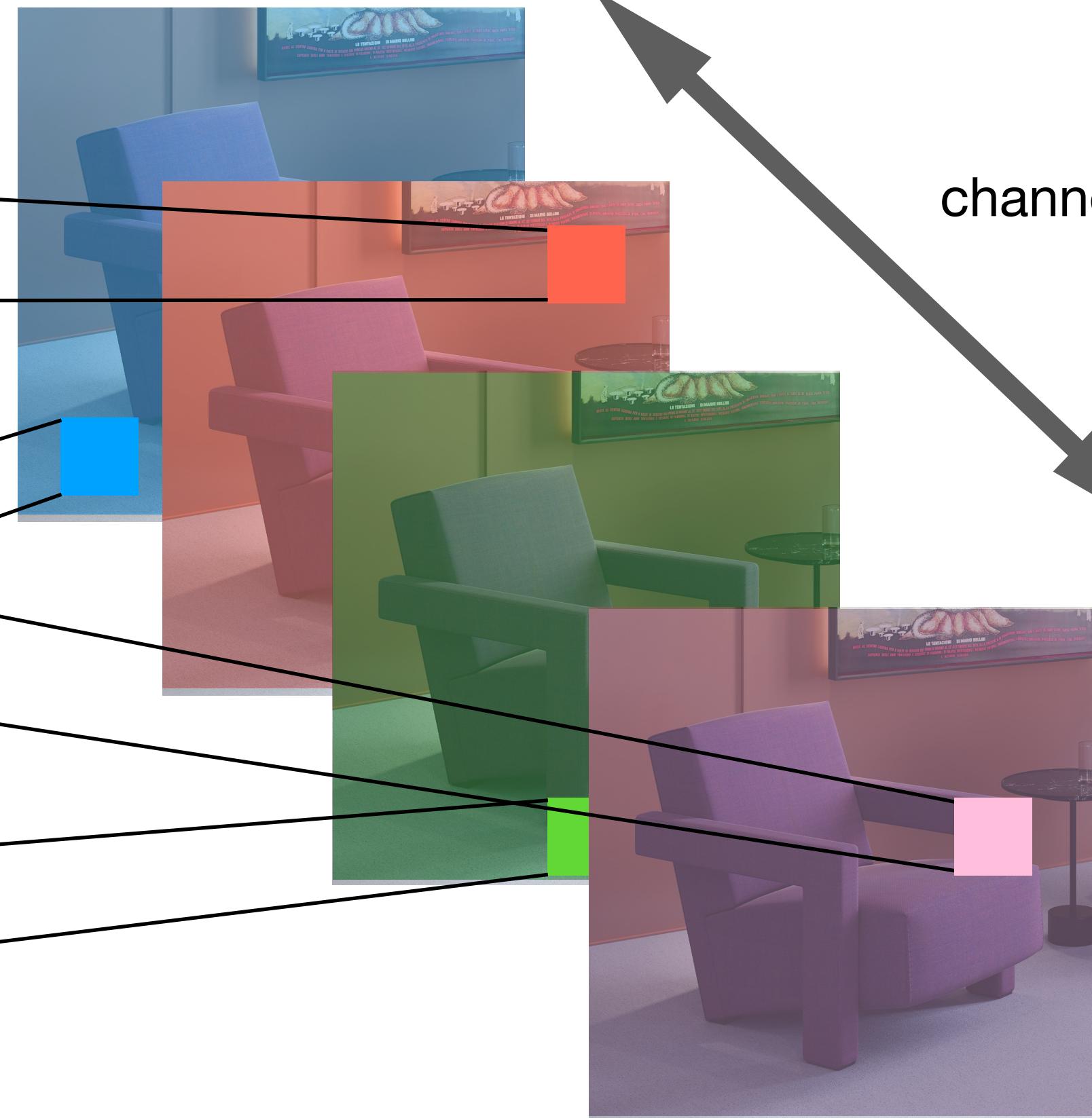
- Dimension of the convolved feature is smaller than the input's one

Multiple Channels

It is common to use multiple filters.

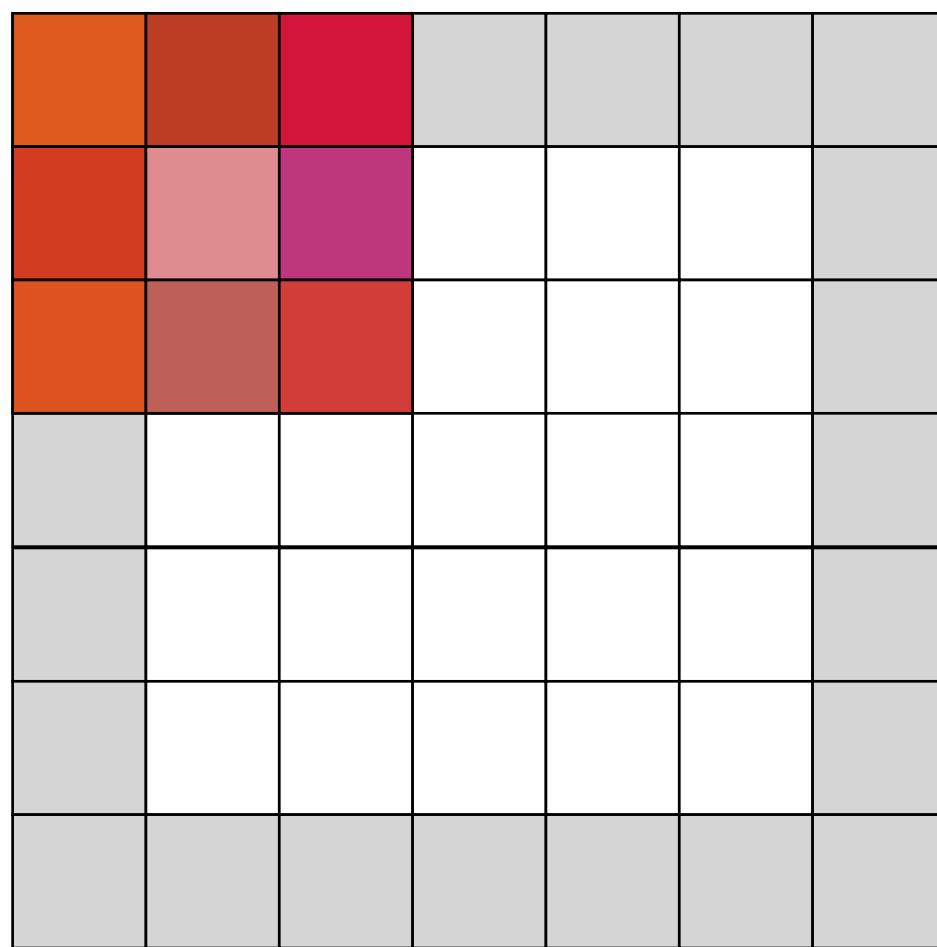


Different filters

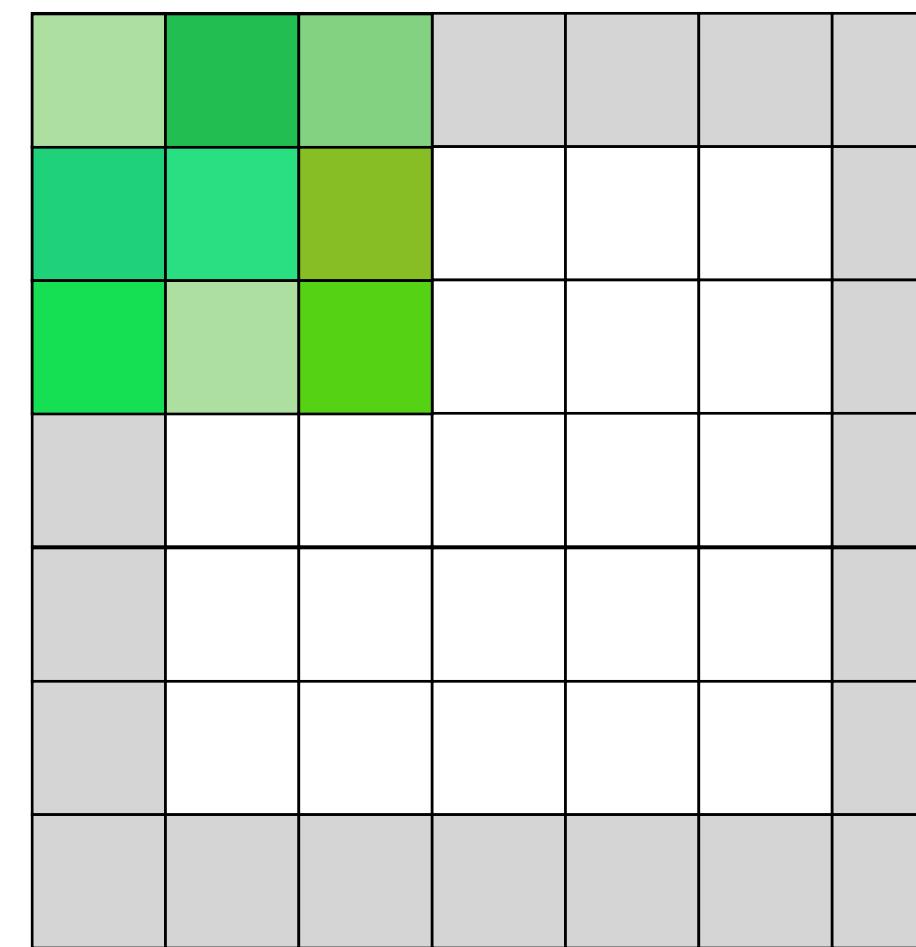


The various outputs are called *channels*

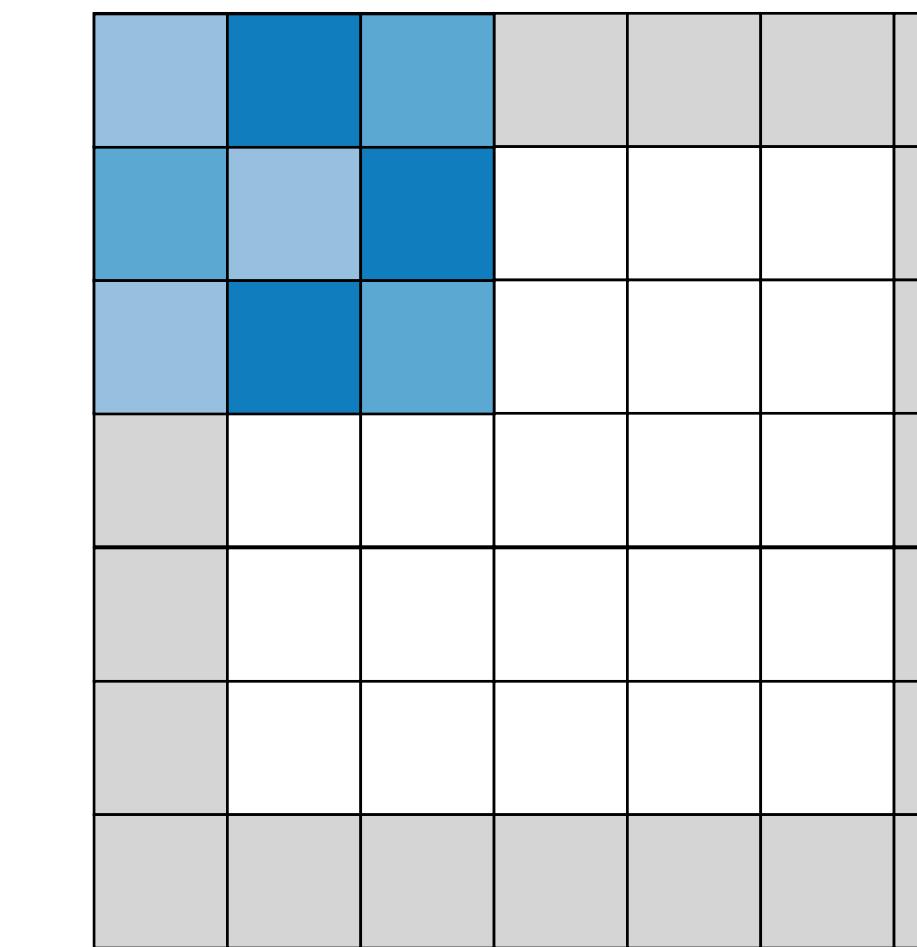
Convolution on input with multiple channels



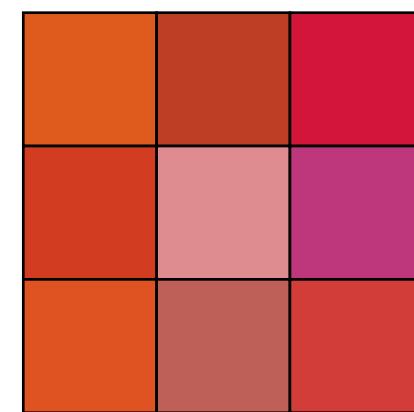
Input Channel #1



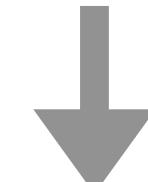
Input Channel #2



Input Channel #3



Filter Channel #1



120

+

-75

+

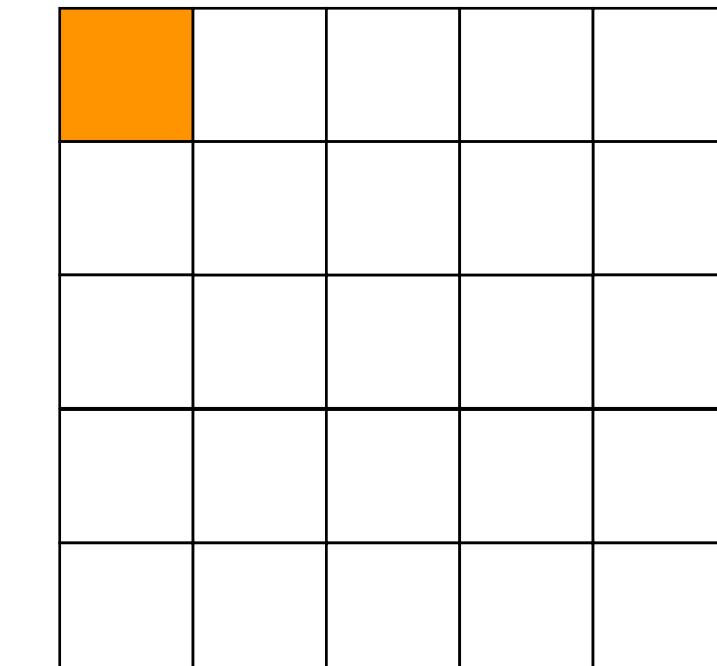
205

+

10

= 260

Bias

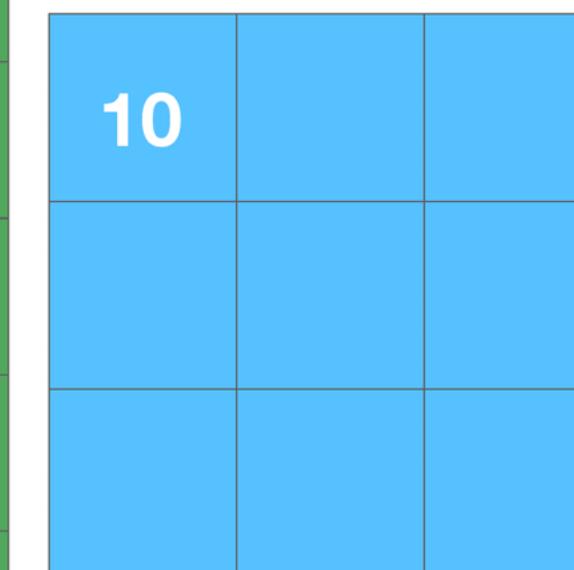


Output

Pooling: often applied after the convolutional layer

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

3	5	8	10	9	4
7	10	2	6	2	8
7	3	5	1	7	0
1	2	6	1	9	2
3	1	2	2	5	4
4	8	2	1	4	3

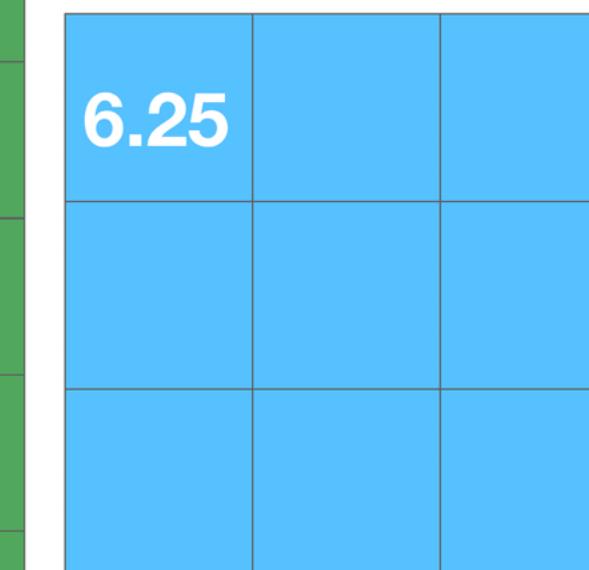


Convolved feature

2 × 2 max pooling

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

3	5	8	10	9	4
7	10	2	6	2	8
7	3	5	1	7	0
1	2	6	1	9	2
3	1	2	2	5	4
4	8	2	1	4	3

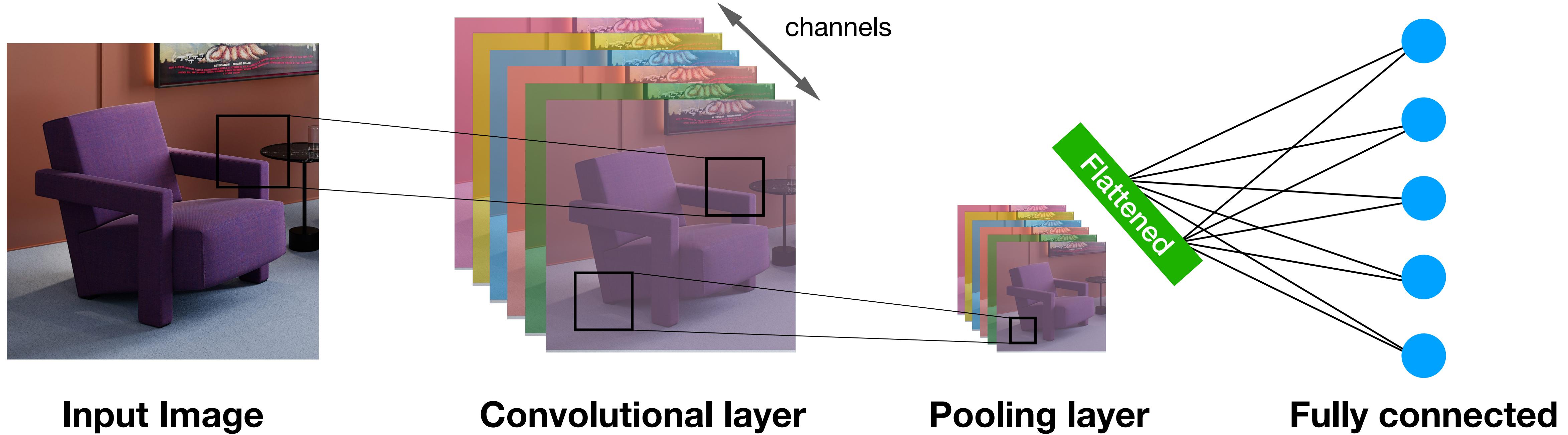


Convolved feature

2 × 2 average pooling

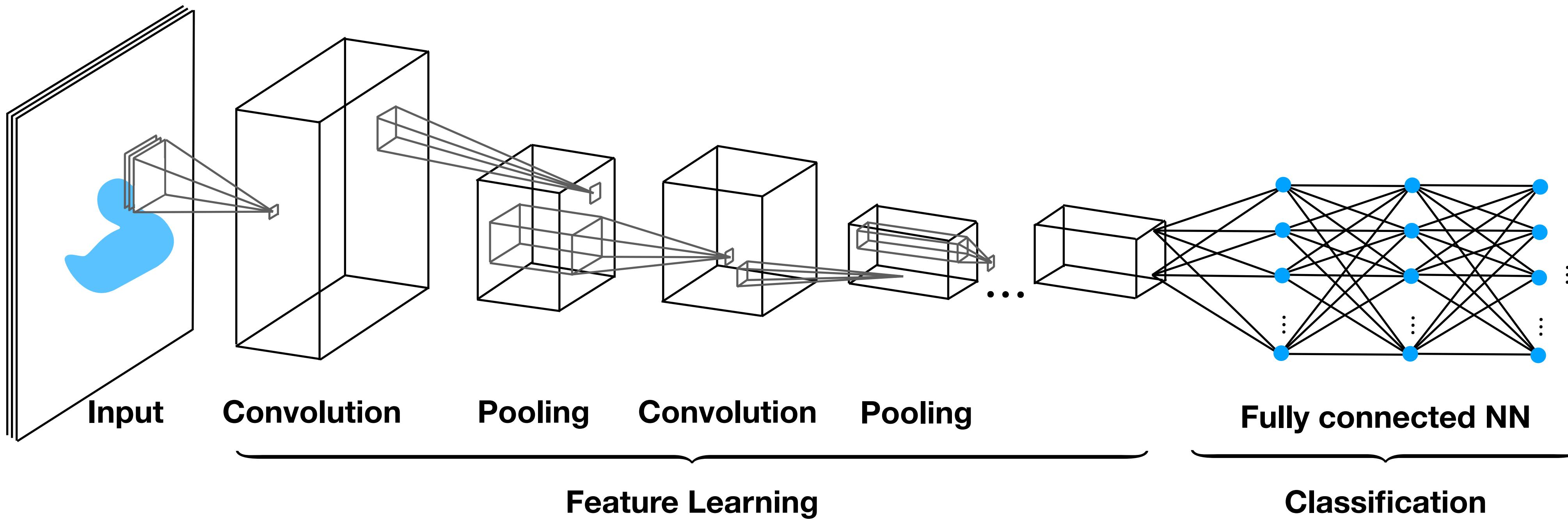
Pooling is a downsampling operation which reduces the spatial size of the convolved feature

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

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


Chain rule

Regularization

Weight decay: ℓ_2 -regularization for NNs

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

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

- It favors small weights and help generalization and optimization

Optimization (GD):

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

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: ℓ_2 -regularization for NNs

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

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

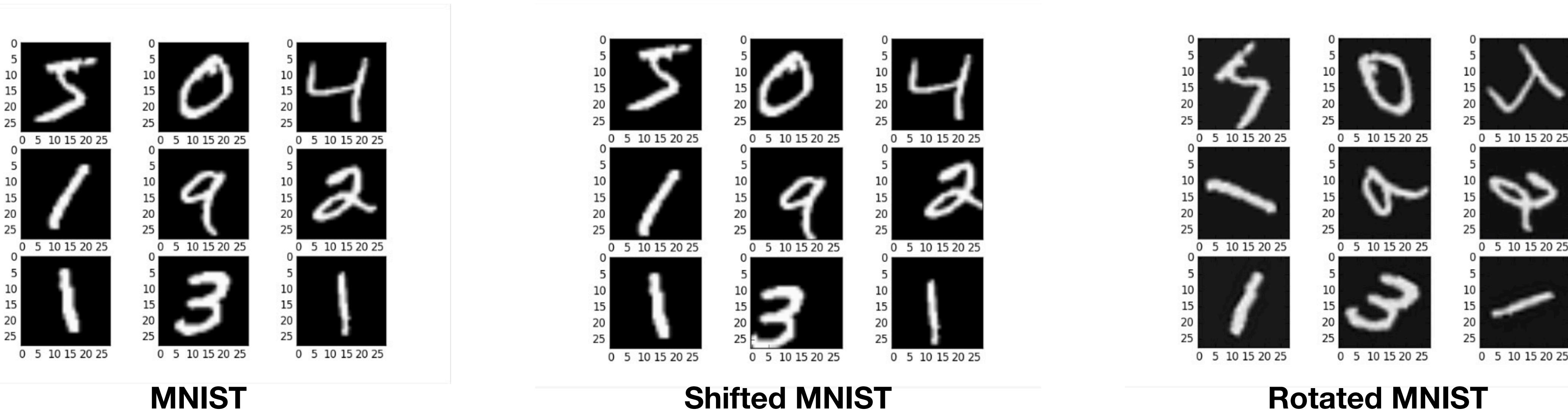
It can be minimized with projected gradient descent

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



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 τ
- It can be seen as regularization
- These transformations are task and dataset specific

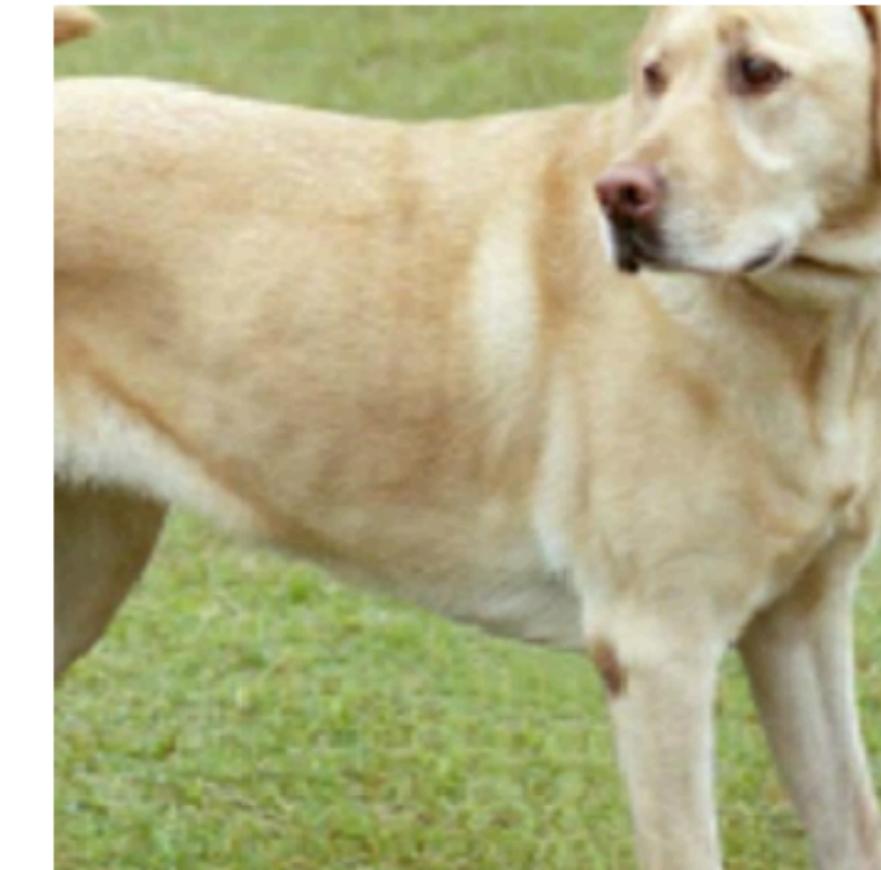
Data augmentation: pictures can also be cropped, resized, or perturbed by a small amount of noise



(a) Original



(b) Crop and resize



(c) Crop, resize (and flip)



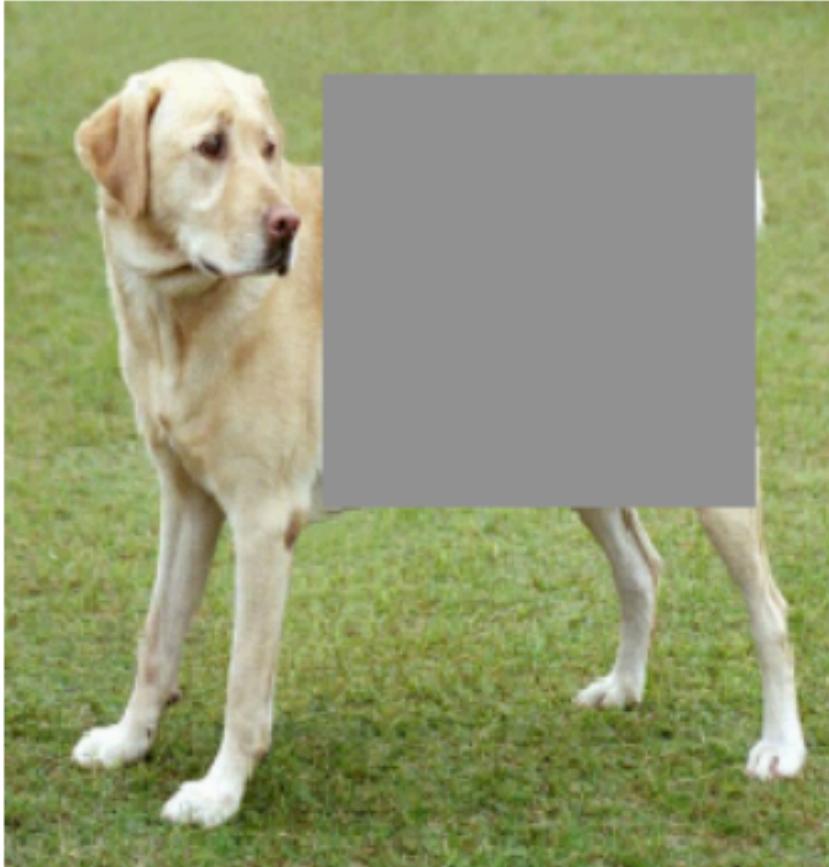
(d) Color distort. (drop)



(e) Color distort. (jitter)



(f) Rotate $\{90^\circ, 180^\circ, 270^\circ\}$



(g) Cutout



(h) Gaussian noise

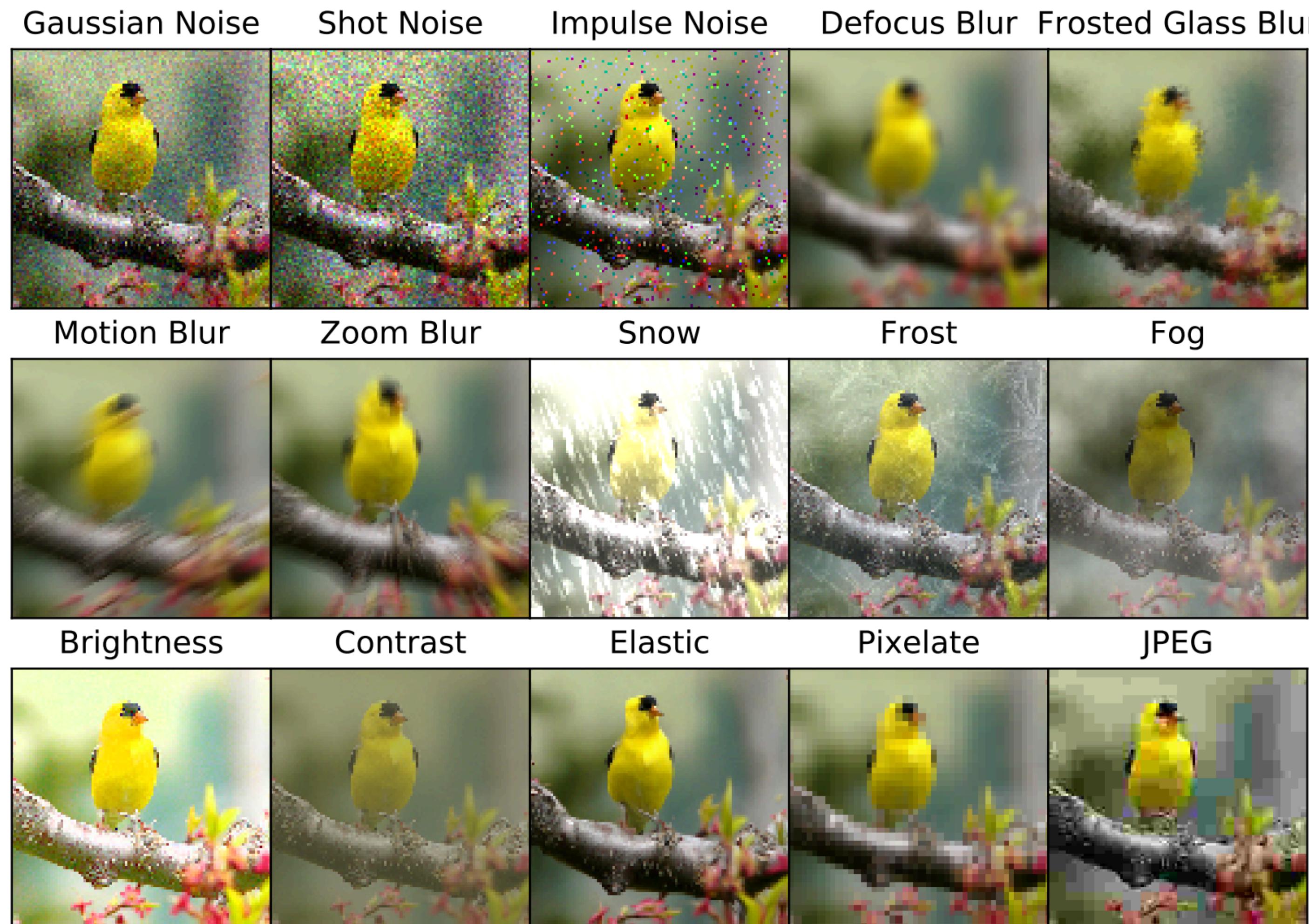


(i) Gaussian blur



(j) Sobel filtering

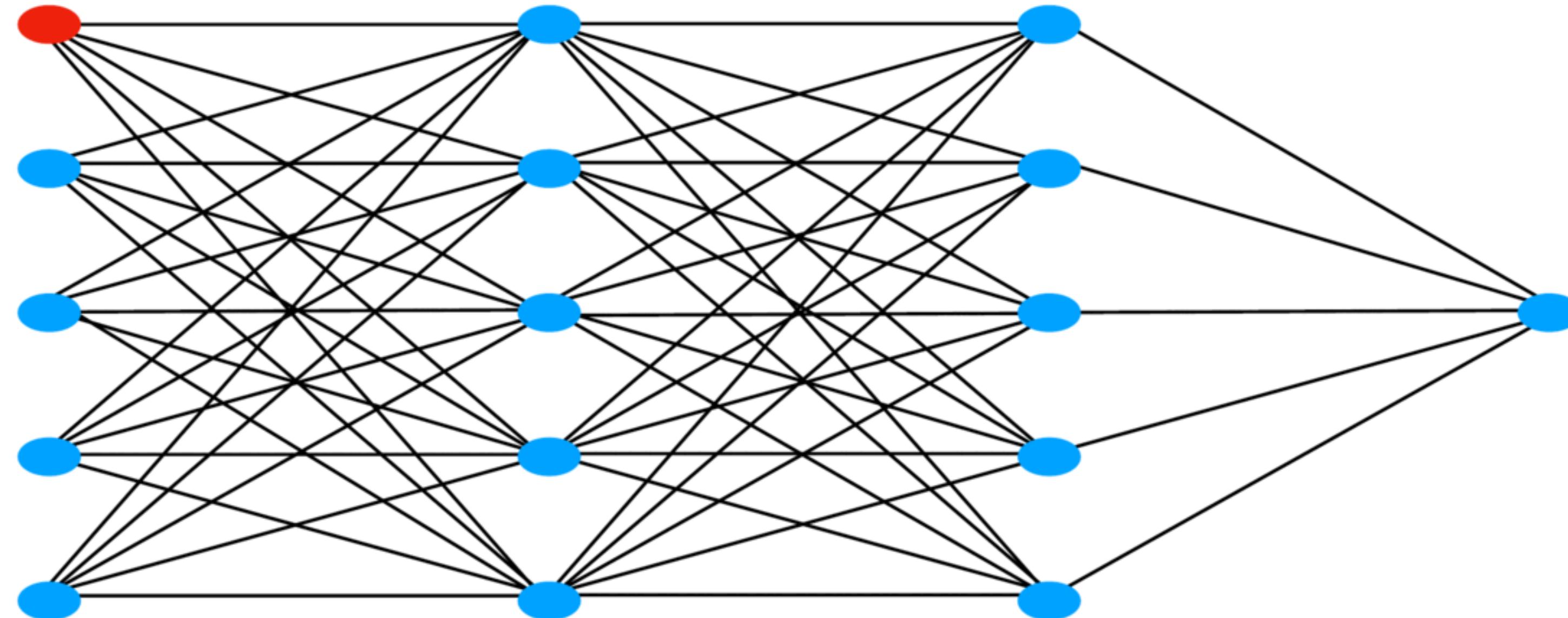
Data augmentation: generated corruptions



Dropout

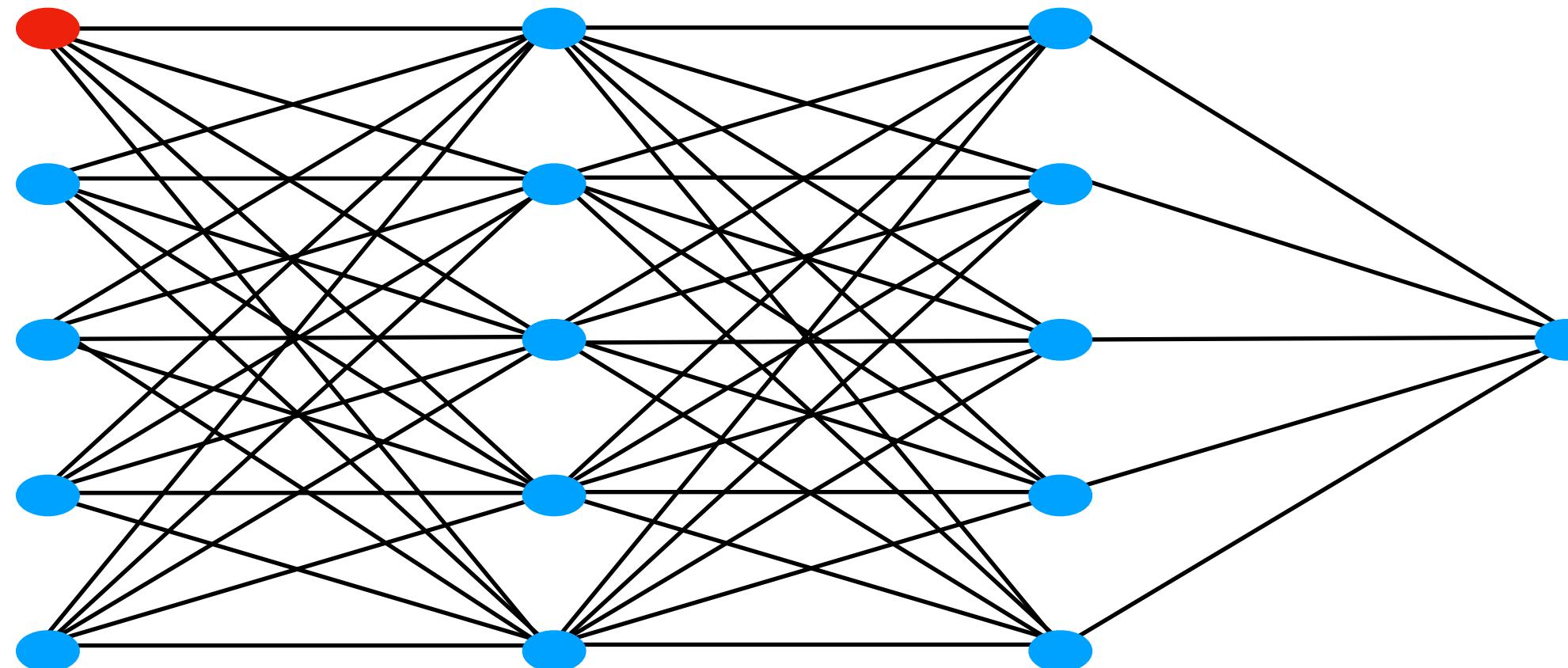
Dropout: randomly drop nodes

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

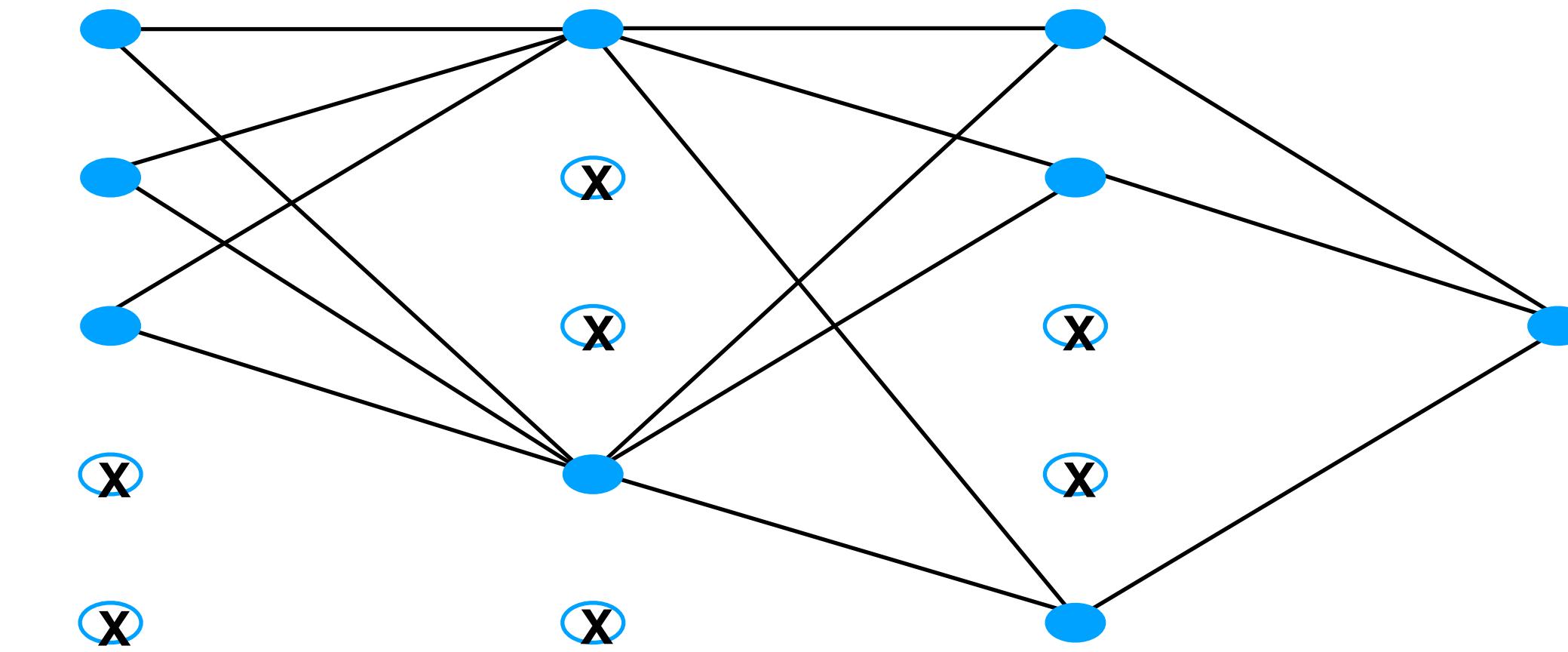


Dropout: training phase

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 $l \geq 1$)



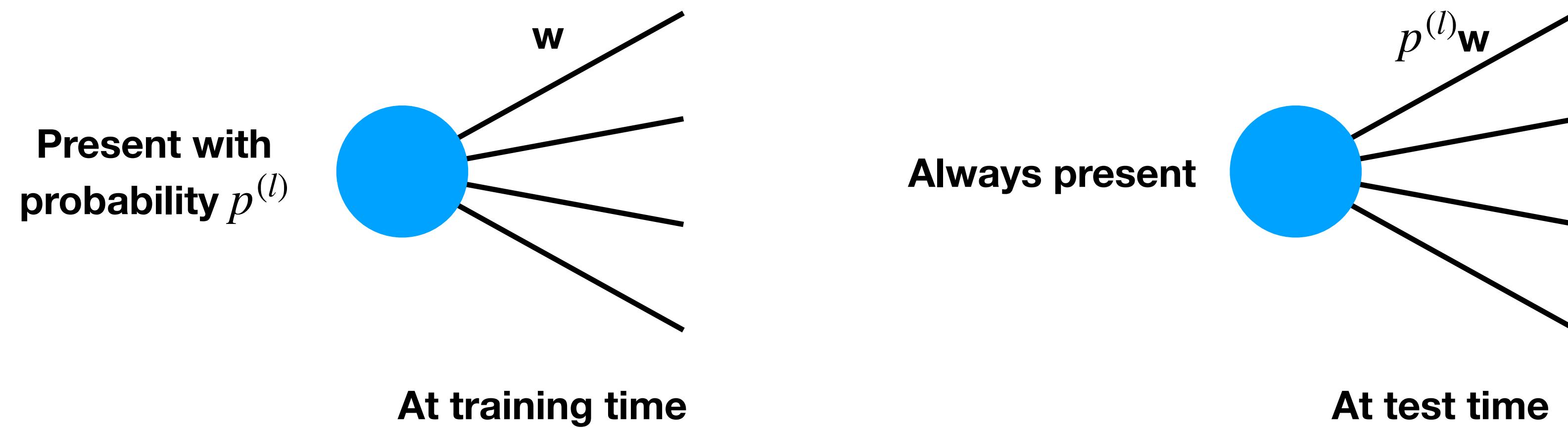
Original network



Random subnetwork

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

Dropout: testing phase



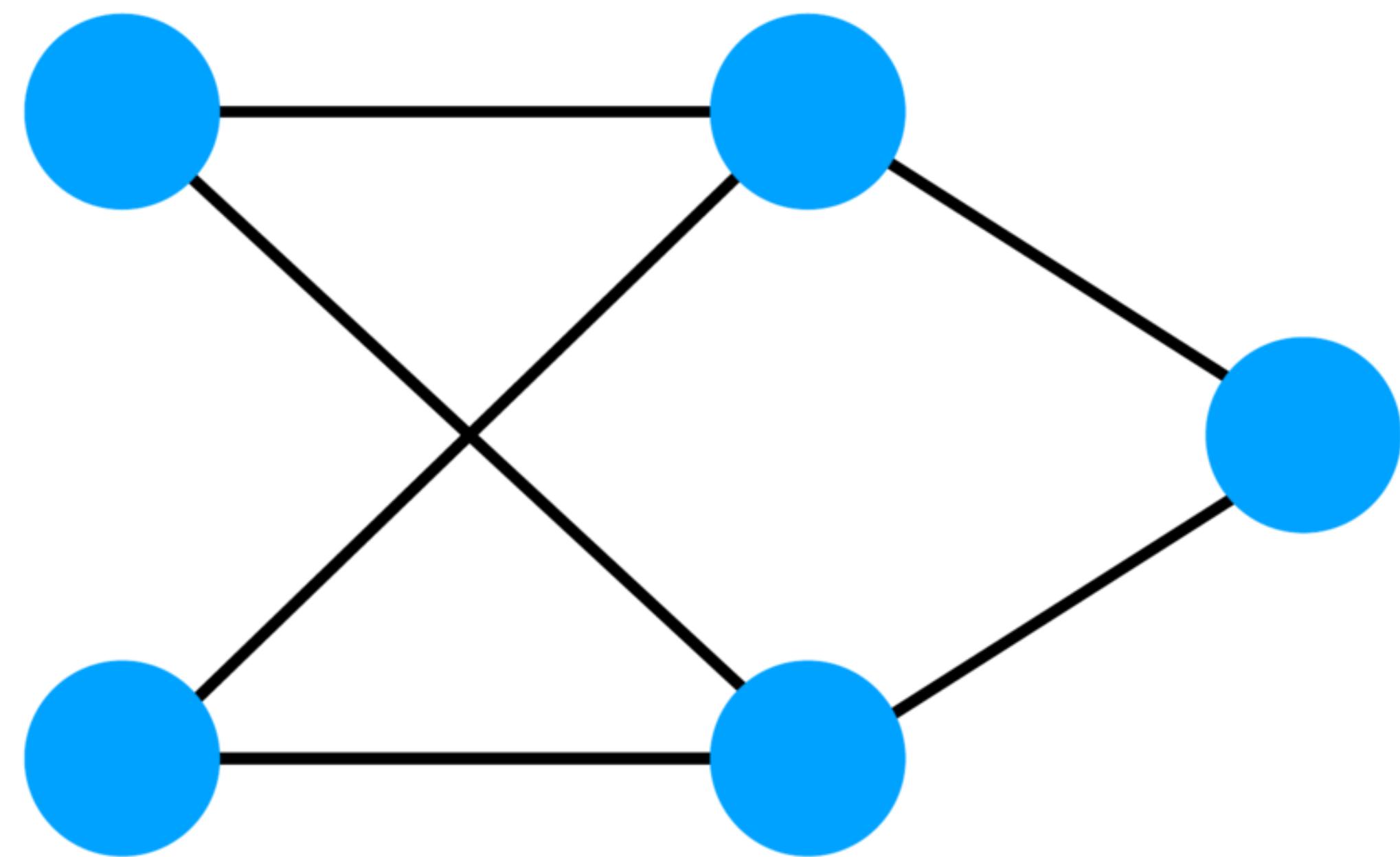
When testing:

- 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:

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

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

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/o 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

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