MIDDLE EAST TECHNICAL UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

CENG 403
Introduction to Deep Learning

Week 10a

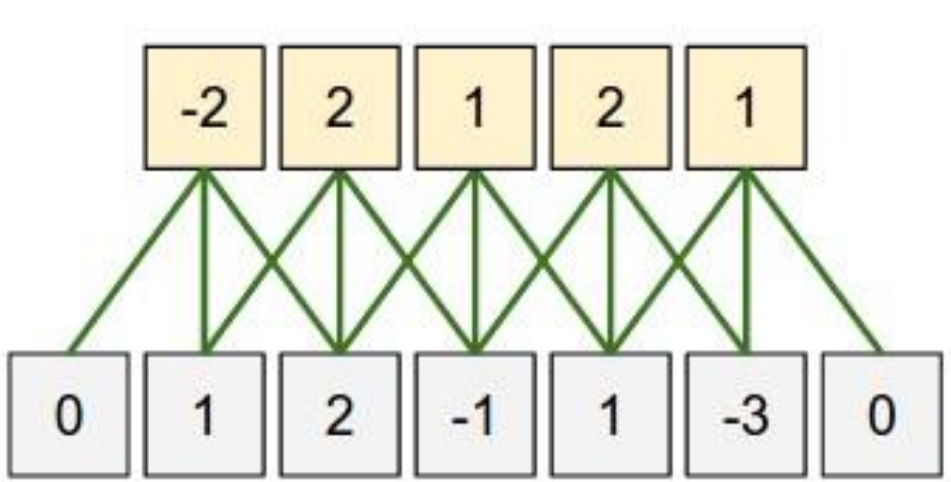
Sinan Kalkan

Previously on CENG403

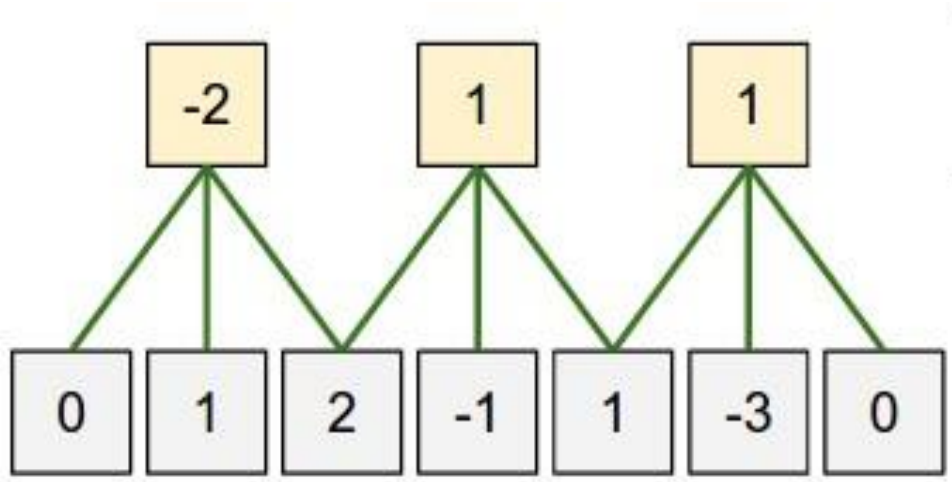
Important parameters

Stride

- The amount of space between neighboring receptive fields
- If it is small, RFs overlap more
- It it is big, RFs overlap less

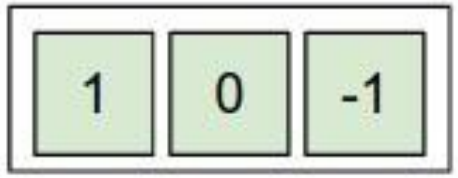


Stride = 1



Stride = 2

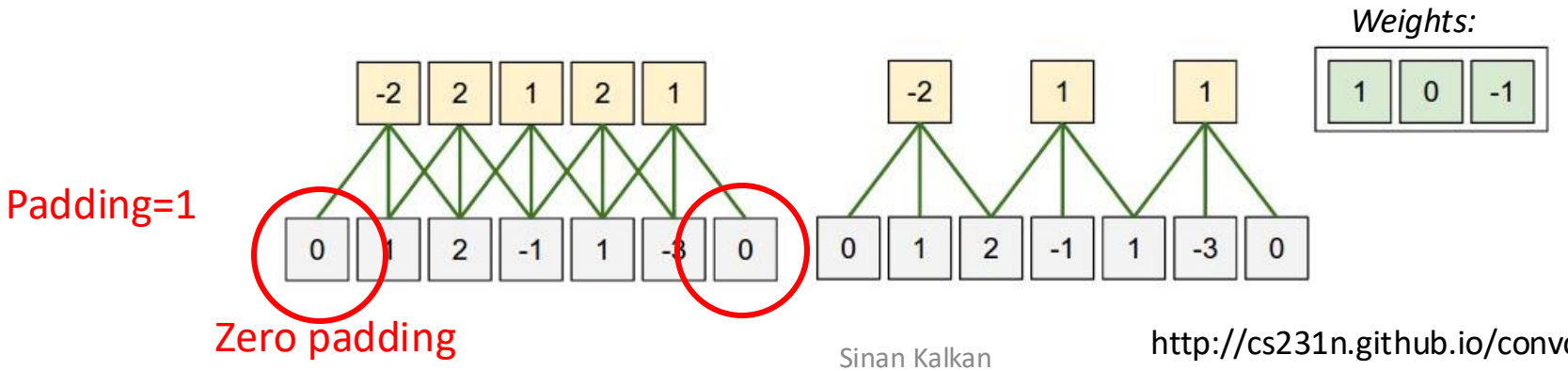
Weights:



Previously on CENG403

Important parameters

- Depth (number of channels)
  - We will have more neurons getting input from the same receptive field
  - This is similar to the hidden neurons with connections to the same input
  - These neurons learn to become selective to the presence of different signals in the same receptive field
- How to handle the boundaries?
  i. Option 1: Don't process the boundaries. Only process pixels on which convolution window can be placed fully.
  ii. Option 2: Zero-pad the input so that convolution can be performed at the boundary pixels.



http://cs231n.github.io/convolutional-networks/

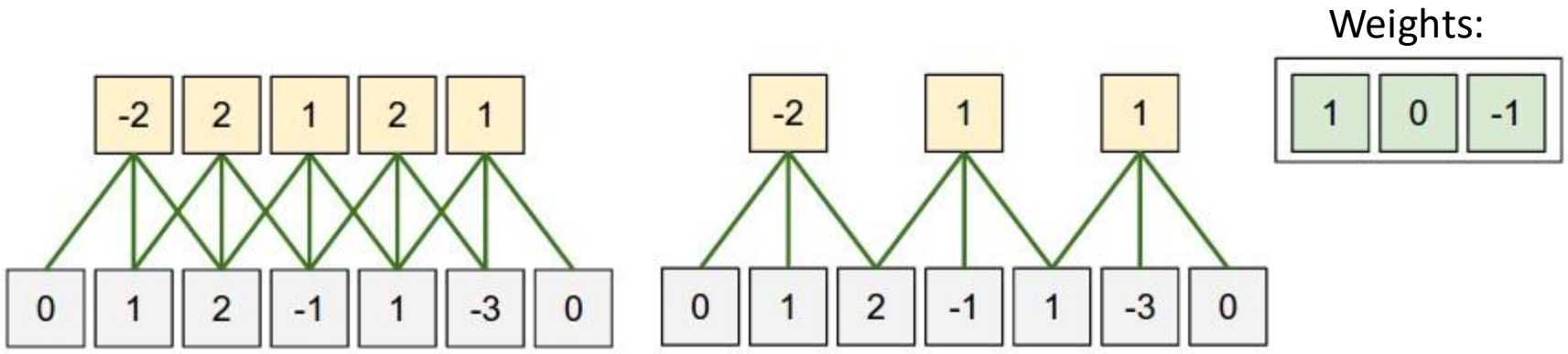
Previously on CENG403

Size of the next layer

- Along a dimension:
  - W: Size of the input
  - F: Size of the receptive field
  - S: Stride
  - P: Amount of zero-padding
- Then: the number of neurons as the output of a convolution layer:

$$\frac{W - F + 2P}{S} + 1$$

- If this number is not an integer, your strides are incorrect and your neurons cannot tile to cover the input volume



http://cs231n.github.io/convolutional-networks/

Previously on CENG403

Types of Convolution: Unshared convolution

- In some cases, sharing the weights does not make sense
  - When?
- Different parts of the input might require different types of processing/features
- In such a case, we just have a network with local connectivity
- E.g., a face.
  - Features are not repeated across the space.

Types of Convolution:
Dilated (Atrous) Convolution

Previously on CENG783

Purpose: Increase effective receptive field size without increasing parameters.

Published as a conference paper at ICLR 2016

MULTI-SCALE CONTEXT AGGREGATION BY DILATED CONVOLUTIONS

Fisher Yu
Princeton University

Vladlen Koltun
Intel Labs

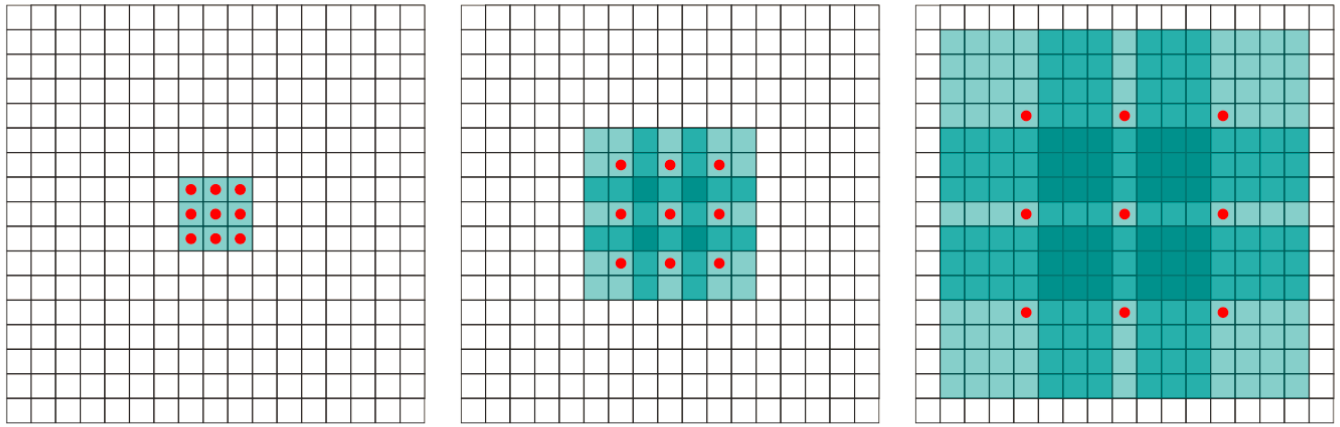
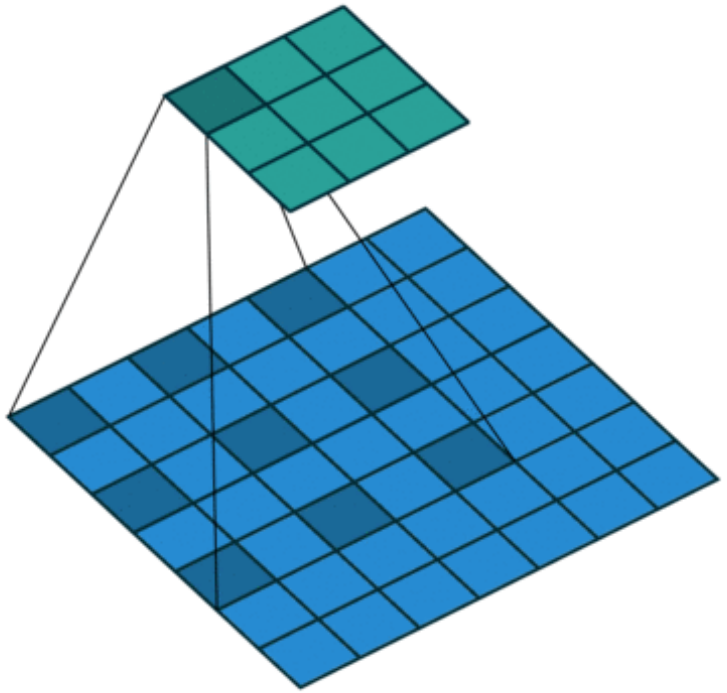


Figure 1: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) F_1 is produced from F_0 by a 1-dilated convolution; each element in F_1 has a receptive field of 3×3. (b) F_2 is produced from F_1 by a 2-dilated convolution; each element in F_2 has a receptive field of 7×7. (c) F_3 is produced from F_2 by a 4-dilated convolution; each element in F_3 has a receptive field of 15×15. The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly.



https://github.com/vdumoulin/conv_arithmetic

Previously on CENG403

Types of Convolution: Transposed Convolution

Purpose: Increasing layer width+height (upsampling).

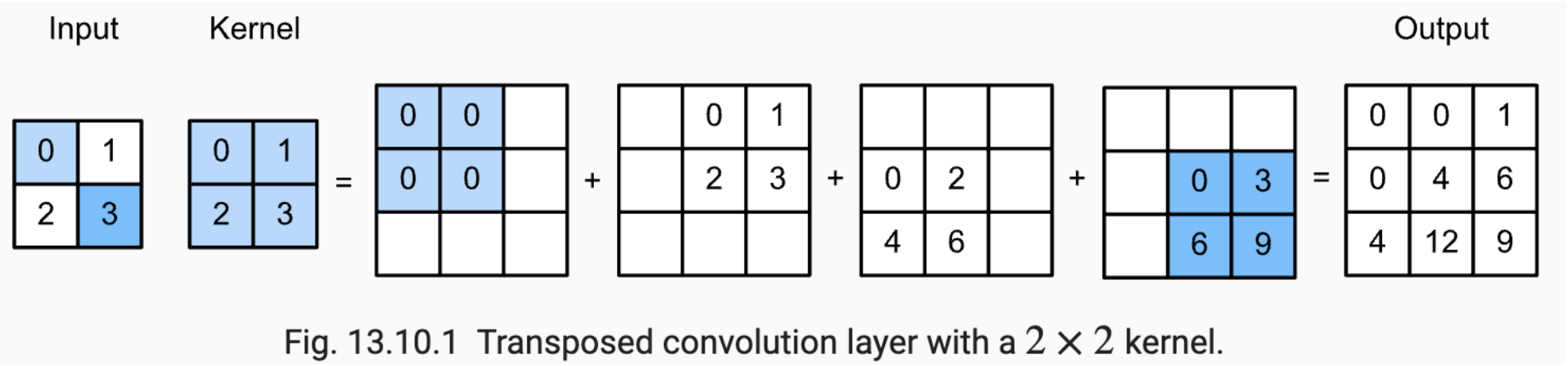


Fig. 13.10.1 Transposed convolution layer with a 2×2 kernel.

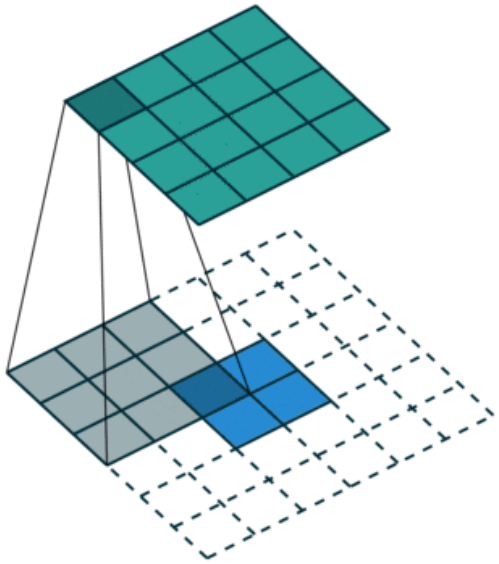
Figure: https://d2l.ai/chapter_computer-vision/transposed-conv.html

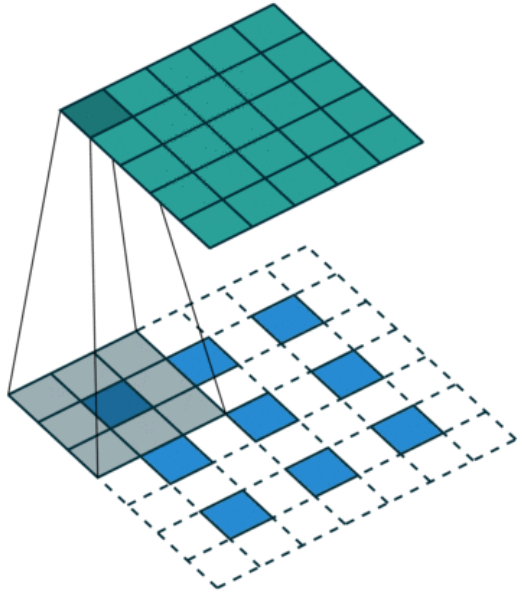
The size of the output:

- Regular convolution: $O = \frac{W-F+2\times P}{S} + 1$
- Transpose convolution: $W = (O-1) \times S + F - 2 \times P$

Previously on CENG403

Types of Convolution:
Upsampling with Padding or Dilation



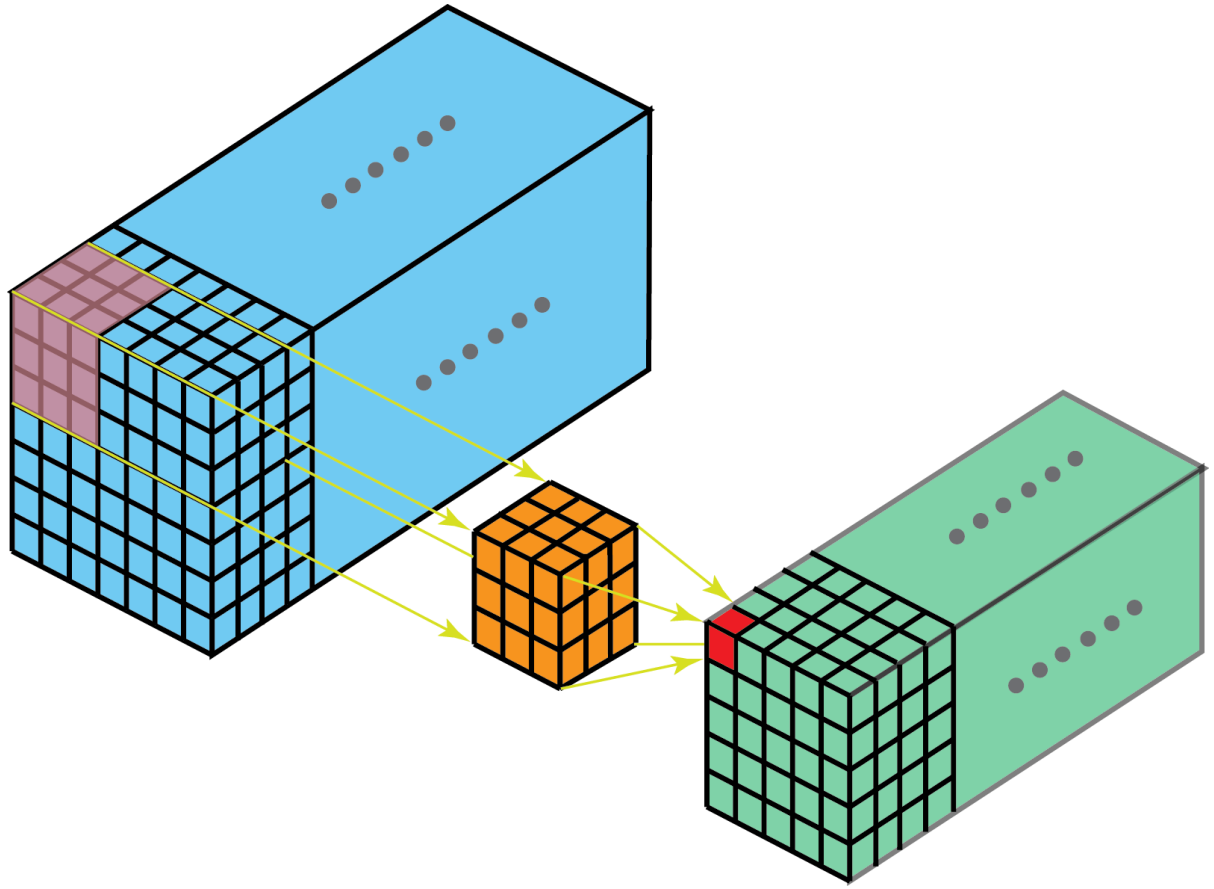


https://github.com/vdumoulin/conv_arithmetic

Previously on CENG403

Types of Convolution: 3D Convolution

Purpose: Work with 3D data, e.g. learn spatial + temporal representations for videos.

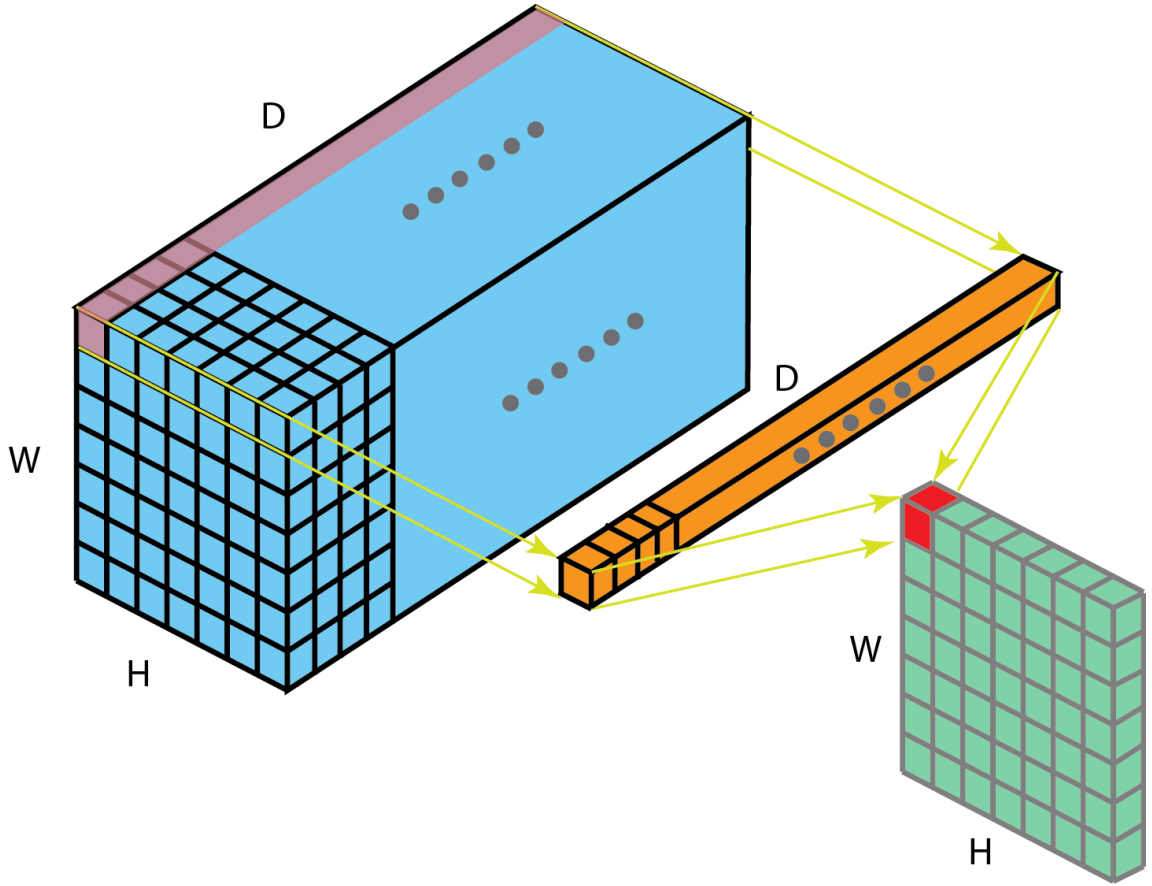


https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

Previously on CENG403

Types of Convolution: 1x1 Convolution

Purpose: Reduce number of channels.



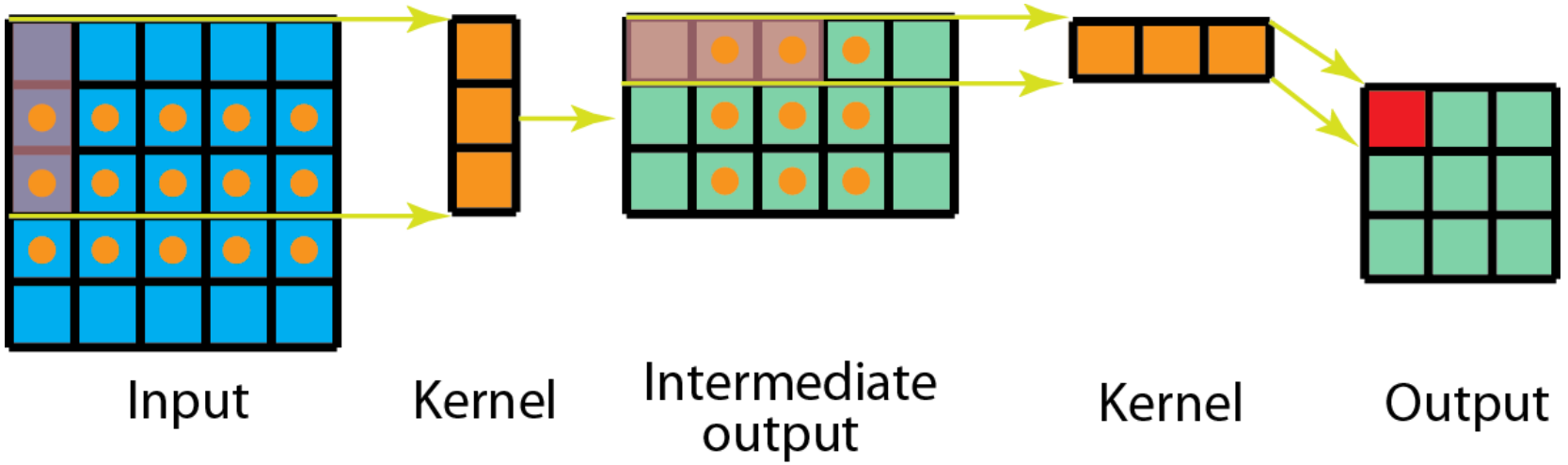
https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

Previously on CENG403

Types of Convolution: Separable Convolution

Purpose: Reduce number of parameters and multiplications.

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

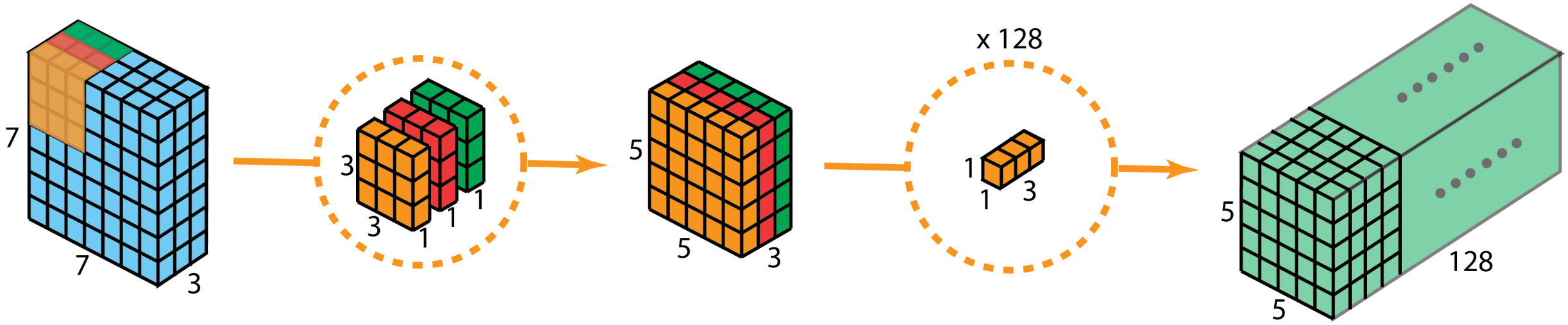


https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

Previously on CENG403

Types of Convolution:
Depth-wise Separable Convolution

Purpose: Reduce number of parameters and multiplications.

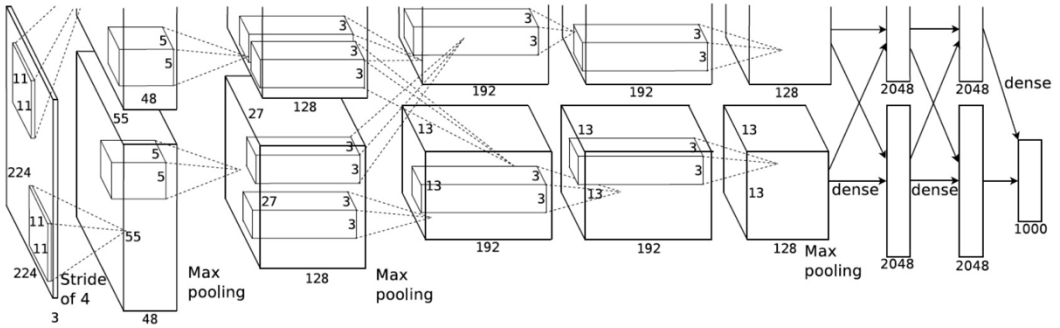


https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

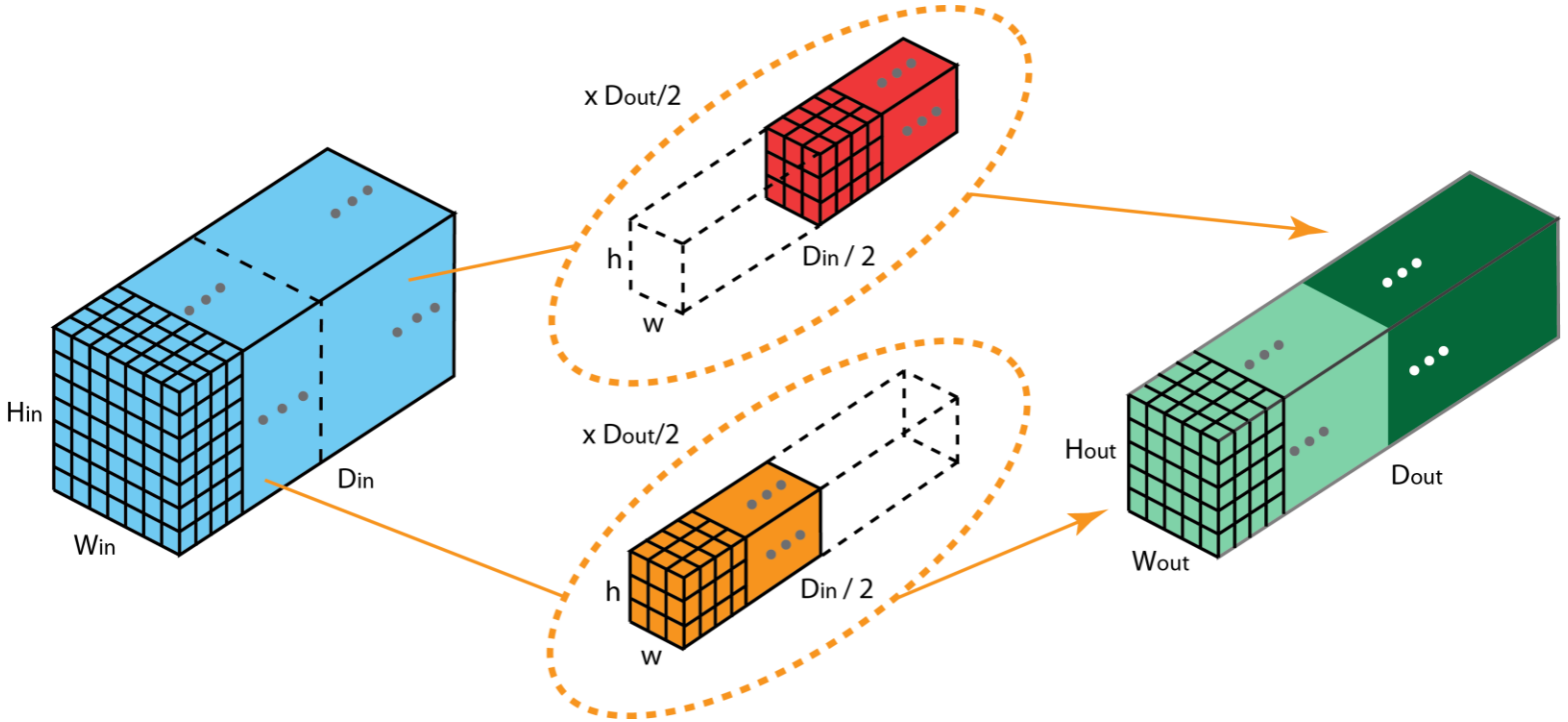
Types of Convolution: Group Convolution

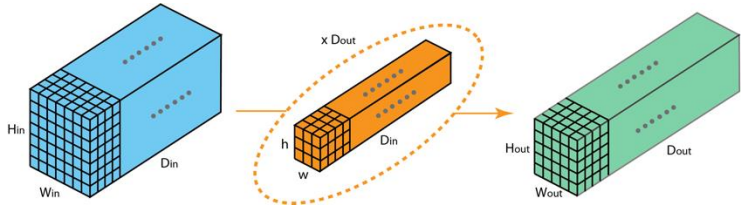
Previously on CENG403

Purpose: Reduce number of parameters and multiplications.



AlexNet





Normal Convolution

https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

Previously on CENG403

Types of Convolution: Deformable Convolution

Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H., & Wei, Y. (2017). Deformable convolutional networks. ICCV.

Purpose: Flexible receptive field.

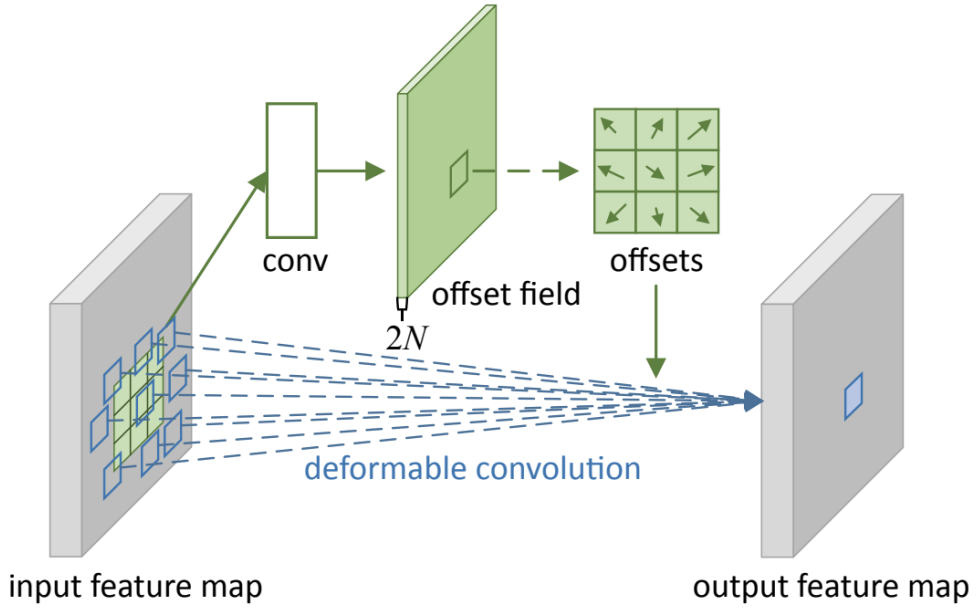


Figure 2: Illustration of 3×3 deformable convolution.

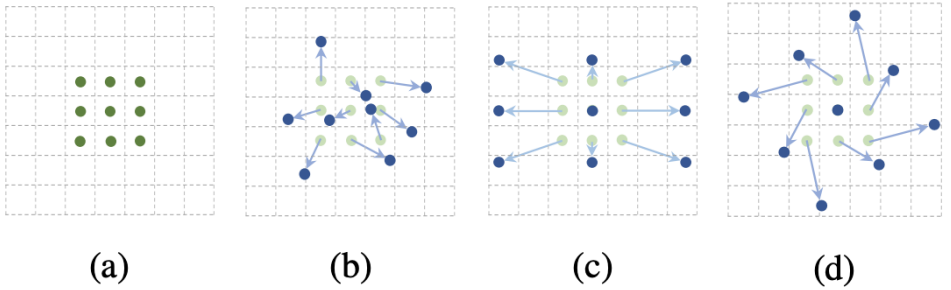
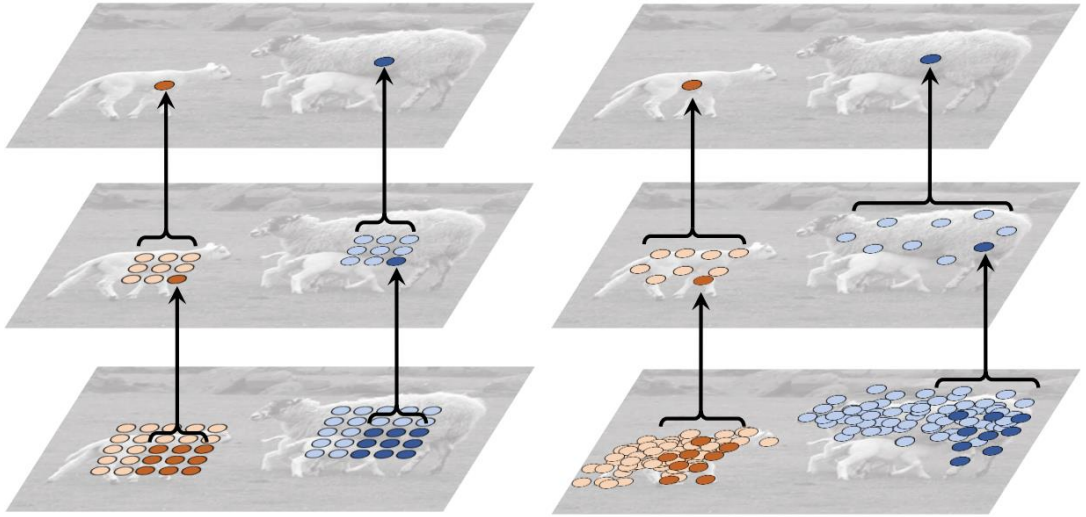


Figure 1: Illustration of the sampling locations in 3×3 standard and deformable convolutions. (a) regular sampling grid (green points) of standard convolution. (b) deformed sampling locations (dark blue points) with augmented offsets (light blue arrows) in deformable convolution. (c)(d) are special cases of (b), showing that the deformable convolution generalizes various transformations for scale, (anisotropic) aspect ratio and rotation.



(a) standard convolution (b) deformable convolution

Previously on CENG403

Types of Convolution: Deformable Convolution

Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H., & Wei, Y. (2017). Deformable convolutional networks. ICCV.

$$\mathcal{R} = \{(-1,-1),(-1,0),\ldots,(0,1),(1,1)\}$$

defines a 3×3 kernel with dilation 1.

For each location $\mathbf{p}_0$ on the output feature map $\mathbf{y}$, we have

$$\mathbf{y}(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n), \quad (1)$$

where $\mathbf{p}_n$ enumerates the locations in $\mathcal{R}$.

In deformable convolution, the regular grid $\mathcal{R}$ is augmented with offsets $\{\Delta\mathbf{p}_n | n = 1, ..., N\}$, where $N = |\mathcal{R}|$. Eq. (1) becomes

$$\mathbf{y}(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n). \quad (2)$$

Now, the sampling is on the irregular and offset locations $\mathbf{p}_n + \Delta\mathbf{p}_n$. As the offset $\Delta\mathbf{p}_n$ is typically fractional, Eq. (2) is implemented via bilinear interpolation as

$$\mathbf{x}(\mathbf{p}) = \sum_{\mathbf{q}} G(\mathbf{q}, \mathbf{p}) \cdot \mathbf{x}(\mathbf{q}), \quad (3)$$

where $\mathbf{p}$ denotes an arbitrary (fractional) location ($\mathbf{p} = \mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n$ for Eq. (2)), $\mathbf{q}$ enumerates all integral spatial locations in the feature map $\mathbf{x}$, and $G(\cdot,\cdot)$ is the bilinear interpolation kernel. Note that G is two dimensional. It is separated into two one dimensional kernels as

$$G(\mathbf{q}, \mathbf{p}) = g(q_x, p_x) \cdot g(q_y, p_y), \quad (4)$$

where $g(a,b) = max(0, 1 - |a - b|)$. Eq. (3) is fast to compute as $G(\mathbf{q}, \mathbf{p})$ is non-zero only for a few $\mathbf{q}$s.

In the deformable convolution Eq. (2), the gradient w.r.t. the offset $\Delta\mathbf{p}_n$ is computed as

$$\frac{\partial \mathbf{y}(\mathbf{p}_0)}{\partial \Delta\mathbf{p}_n} = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \frac{\partial \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n)}{\partial \Delta\mathbf{p}_n}$$

$$= \sum_{\mathbf{p}_n \in \mathcal{R}} \left[\mathbf{w}(\mathbf{p}_n) \cdot \sum_{\mathbf{q}} \frac{\partial G(\mathbf{q}, \mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n)}{\partial \Delta\mathbf{p}_n} \mathbf{x}(\mathbf{q}) \right], \quad (7)$$

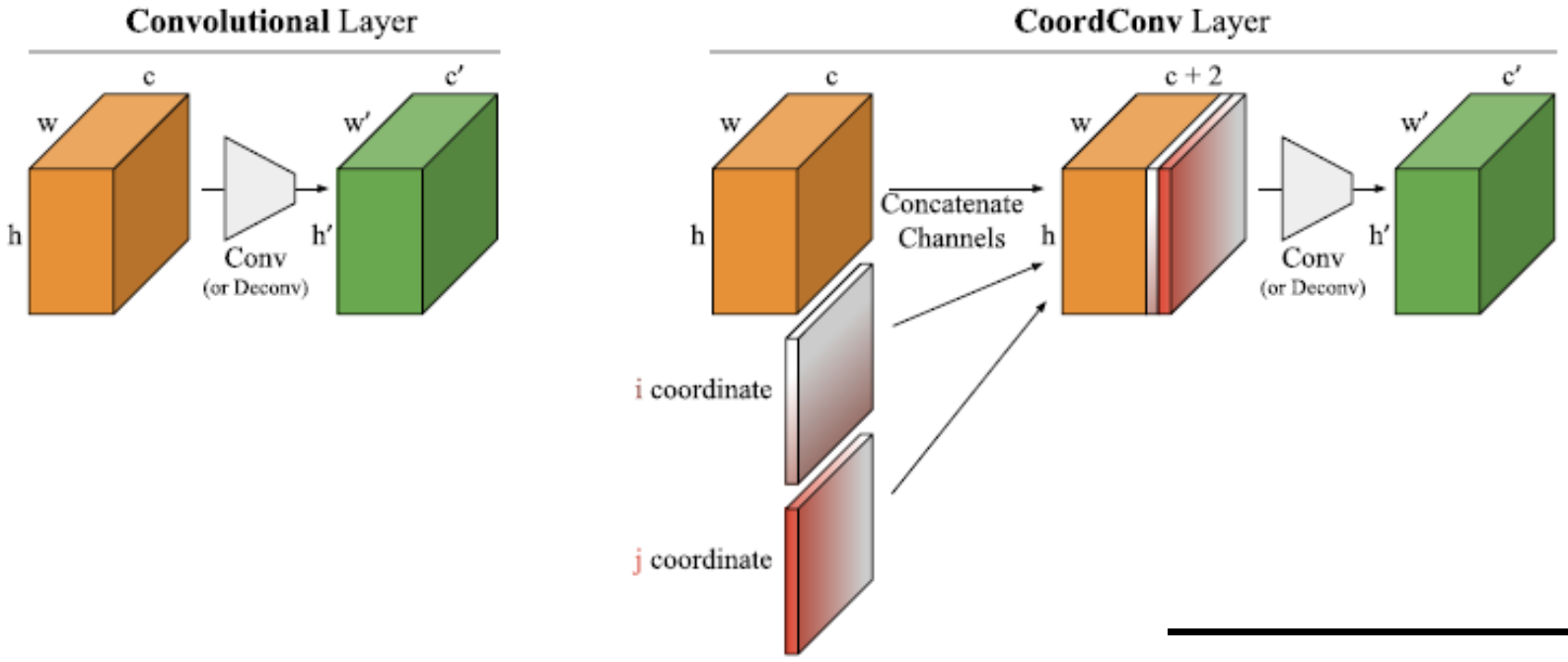
where the term $\frac{\partial G(\mathbf{q}, \mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n)}{\partial \Delta\mathbf{p}_n}$ can be derived from Eq. (4). Note that the offset $\Delta\mathbf{p}_n$ is 2D and we use $\partial\Delta\mathbf{p}_n$ to denote $\partial\Delta p_n^x$ and $\partial\Delta p_n^y$ for simplicity.

Today

- Convolutional Neural Networks (CNNs)
  - Finish types of convolution in CNNs
  - Pooling
  - FC layers
  - Backpropagation

Types of Convolution:
Position-sensitive convolution

- Learn to use position information when necessary



An intriguing failing of convolutional neural networks and the CoordConv solution 2018

Rosanne Liu[1] rosanne@uber.com
Joel Lehman[1] joel.lehman@uber.com
Piero Molino[1] piero@uber.com
Felipe Petroski Such[1] felipe.such@uber.com
Eric Frank[1] mysterefrank@uber.com
Alex Sergeev[2] asergeev@uber.com
Jason Yosinski[1] yosinski@uber.com

Convolution demos & tutorials

- https://github.com/vdumoulin/conv_arithmetic
- http://cs231n.github.io/assets/conv-demo/index.html
- https://ezyang.github.io/convolution-visualizer/index.html
- https://ikhlestov.github.io/pages/machine-learning/convolutions-types/
- https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

OPERATIONS IN A CNN:
Pooling

Pooling

Remember the motivation for CNNs:
S (simple) cells: local feature extraction.
C (complex) cells: provide tolerance to deformation, e.g. shift.

- Apply an operation on the "detector" results to combine or to summarize the answers of a set of units.
  - Applied to **each channel (depth slice) independently**
  - The operation has to be differentiable of course.
- Alternatives:
  - Maximum
  - Sum
  - Average
  - Weighted average with distance from the value of the center pixel
  - L2 norm
  - Second-order statistics?
  - …
- Different problems may perform better with different pooling methods
- Pooling can be overlapping or non-overlapping

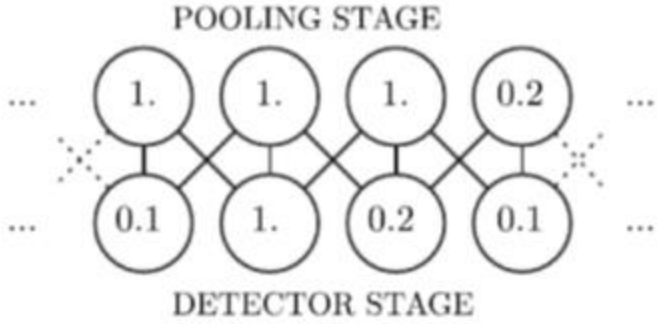


Figure: Goodfellow et al., "Deep Learning", MIT Press, 2016.

http://cs231n.github.io/convolutional-networks/

Pooling

- Example
  - Pooling layer with filters of size 2x2
  - With stride = 2
  - Discards 75% of the activations
  - Depth dimension remains unchanged
- Max pooling with F=3, S=2 or F=2, S=2 are quite common.
  - Pooling with bigger receptive field sizes can be destructive
- Avg pooling is an obsolete choice. Max pooling is shown to work better in practice.

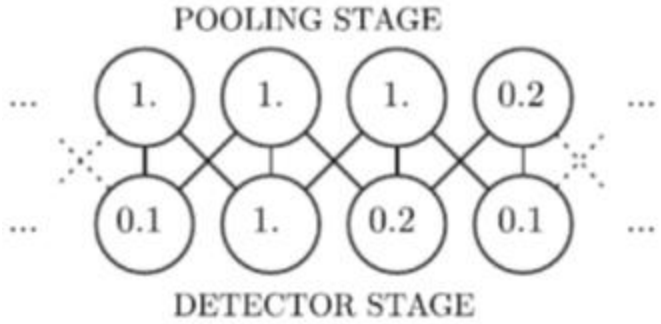
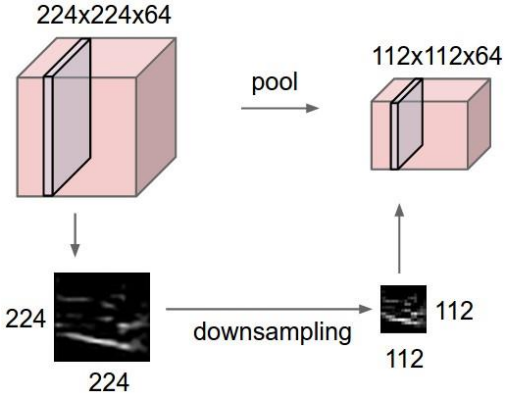
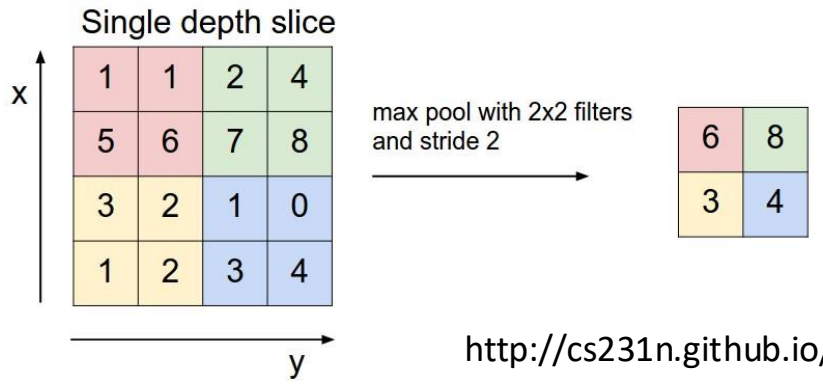


Figure: Goodfellow et al., "Deep Learning", MIT Press, 2016.

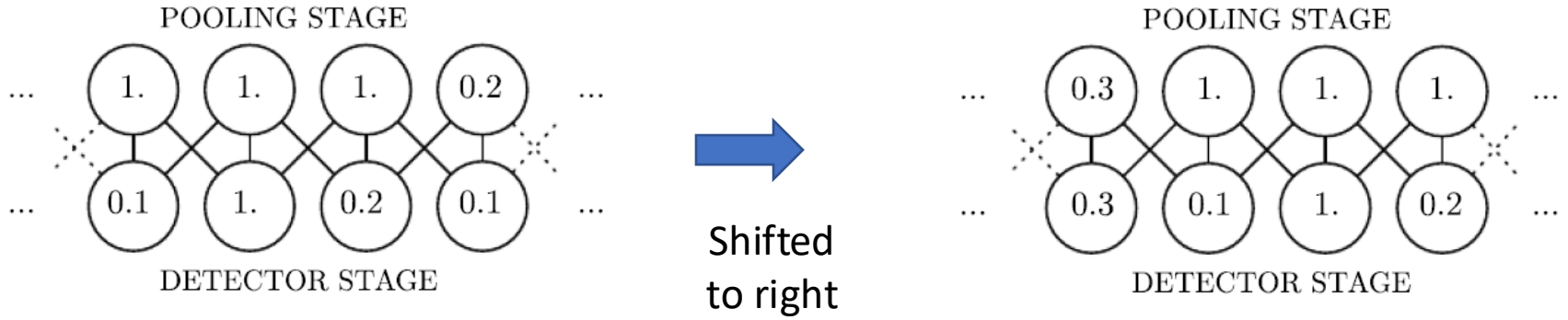




http://cs231n.github.io/convolutional-networks/

Pooling

- Pooling provides invariance to small translation.



Figures: Goodfellow et al., "Deep Learning", MIT Press, 2016.

- If you pool over different convolution operators, you can gain invariance to different transformations.

Pooling can downsample

- Especially needed when to produce an output with fixed-length on varying length input.

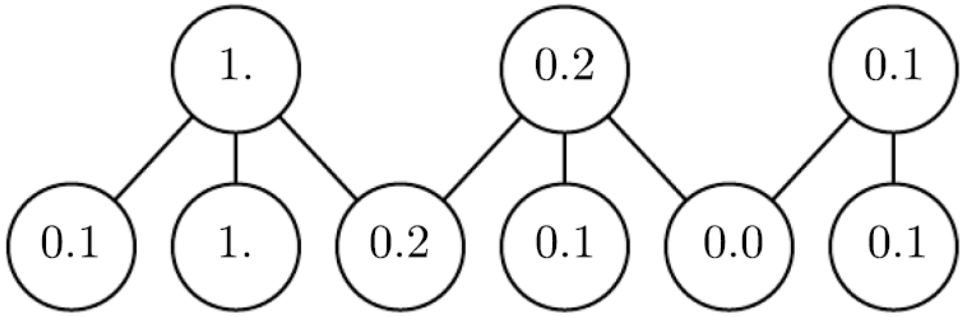


Figure: Goodfellow et al., “Deep Learning”, MIT Press, 2016.

- If you want to use the network on images of varying size, you can arrange this with pooling (with the help of convolutional layers)

CNNs without pooling

- *"Striving for Simplicity: The All Convolutional Net proposes to discard the pooling layer in favor of architecture that only consists of repeated CONV layers. To reduce the size of the representation they suggest using larger stride in CONV layer once in a while."*

CIFAR-10 classification error

Model	Error (%)	# parameters
without data augmentation		
Model A	12.47%	≈ 0.9 M
Strided-CNN-A	13.46%	≈ 0.9 M
ConvPool-CNN-A	**10.21%**	≈ 1.28 M
ALL-CNN-A	10.30%	≈ 1.28 M
Model B	10.20%	≈ 1 M
Strided-CNN-B	10.98%	≈ 1 M
ConvPool-CNN-B	9.33%	≈ 1.35 M
ALL-CNN-B	**9.10%**	≈ 1.35 M
Model C	9.74%	≈ 1.3 M
Strided-CNN-C	10.19%	≈ 1.3 M
ConvPool-CNN-C	9.31%	≈ 1.4 M
ALL-CNN-C	**9.08%**	≈ 1.4 M

(ALL-CNN: No pooling)

https://arxiv.org/pdf/1412.6806.pdf

http://cs231n.github.io/convolutional-networks/

Summary: Convolution & pooling

- Provide strong bias on the model and the solution
- They directly affect the overall performance of the system

OPERATIONS IN A CNN:
nonlinearity

Non-linearity

- Sigmoid
- Tanh
- ReLU and its variants
  - The common choice
  - Faster
  - Easier (in backpropagation etc.)
  - Avoids saturation issues
- …

OPERATIONS IN A CNN:
Normalization

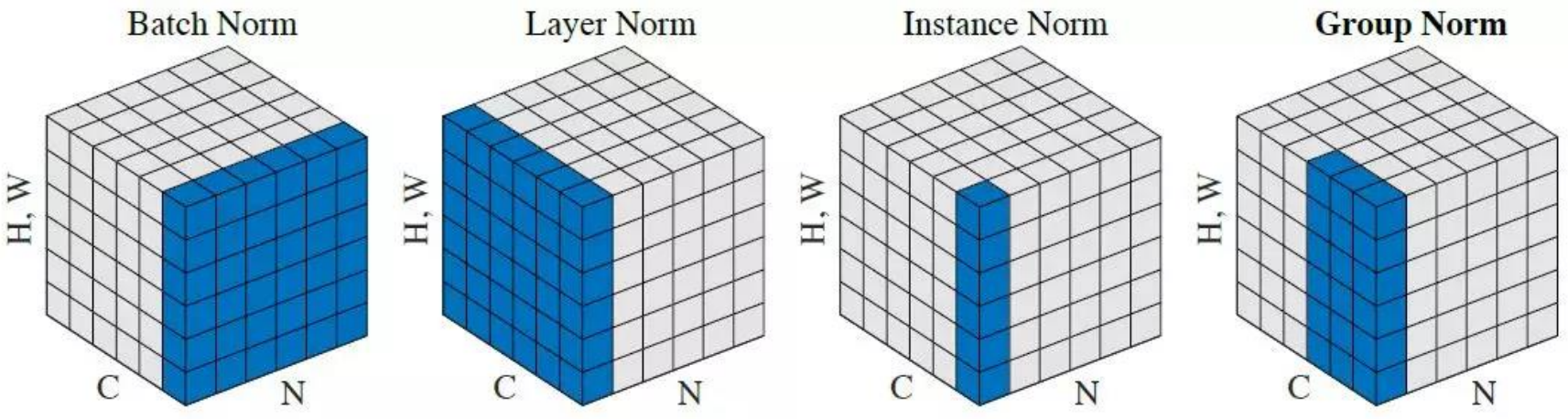
- From Krizhevsky et al. (2012):

generalization. Denoting by $a^i_{x,y}$ the activity of a neuron computed by applying kernel i at position (x, y) and then applying the ReLU nonlinearity, the response-normalized activity $b^i_{x,y}$ is given by the expression

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

where the sum runs over n "adjacent" kernel maps at the same spatial position, and N is the total number of kernels in the layer. The ordering of the kernel maps is of course arbitrary and determined before training begins. This sort of response normalization implements a form of lateral inhibition inspired by the type found in real neurons, creating competition for big activities amongst neuron outputs computed using different kernels. The constants k, n, α, and β are hyper-parameters whose values are determined using a validation set; we used $k = 2$, $n = 5$, $\alpha = 10^{-4}$, and $\beta = 0.75$. We

Normalization



$$\mu_c = \frac{1}{N \cdot H \cdot W} \sum_{n=1}^{N} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{nchw}$$

$$\sigma_c^2 = \frac{1}{N \cdot H \cdot W} \sum_{n=1}^{N} \sum_{h=1}^{H} \sum_{w=1}^{W} (x_{nchw} - \mu_c)^2$$

$$\hat{x}_{nchw} = \frac{x_{nchw} - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}}$$

$$\mu_n = \frac{1}{C \cdot H \cdot W} \sum_{c=1}^{C} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{nchw}$$

$$\sigma_n^2 = \frac{1}{C \cdot H \cdot W} \sum_{c=1}^{C} \sum_{h=1}^{H} \sum_{w=1}^{W} (x_{nchw} - \mu_n)^2$$

$$\hat{x}_{nchw} = \frac{x_{nchw} - \mu_n}{\sqrt{\sigma_n^2 + \epsilon}}$$

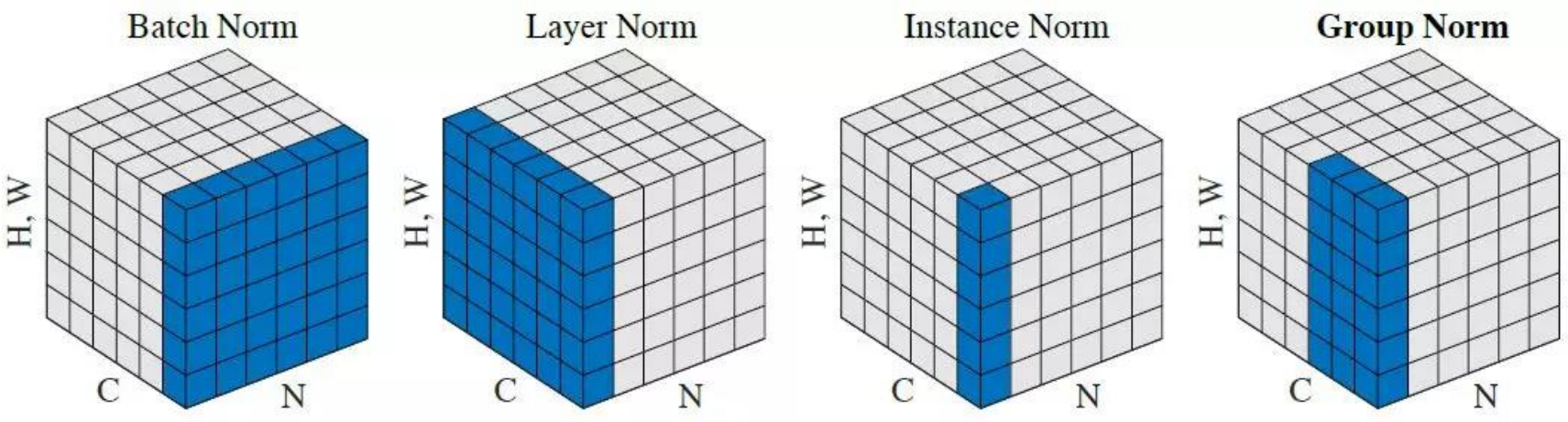
$$\mu_{nc} = \frac{1}{H \cdot W} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{nchw}$$

$$\sigma_{nc}^2 = \frac{1}{H \cdot W} \sum_{h=1}^{H} \sum_{w=1}^{W} (x_{nchw} - \mu_{nc})^2$$

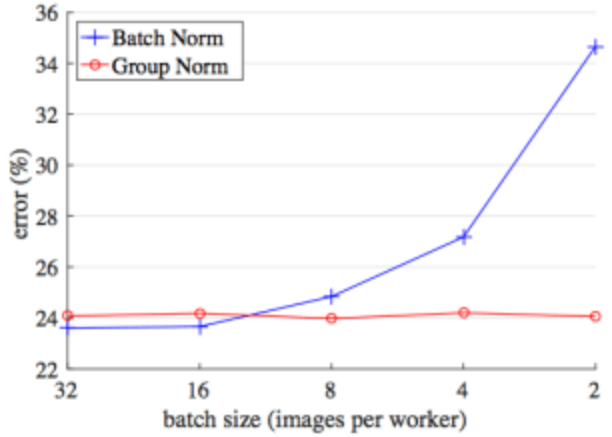
$$\hat{x}_{nchw} = \frac{x_{nchw} - \mu_{nc}}{\sqrt{\sigma_{nc}^2 + \epsilon}}$$

Figure: https://medium.com/syncedreview/facebook-ai-proposes-group-normalization-alternative-to-batch-normalization-fb0699bffae7

Normalization



Lower batch size may mean inaccurate calculation of batch stats, which can degrade performance:

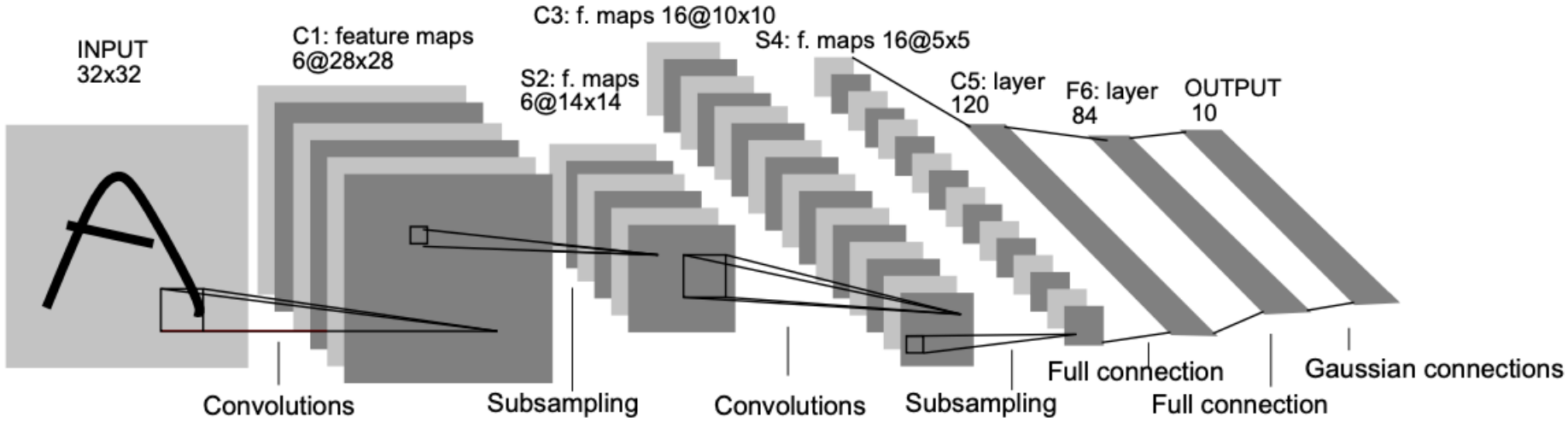


Both figures: https://medium.com/syncedreview/facebook-ai-proposes-group-normalization-alternative-to-batch-normalization-fb0699bffae7

OPERATIONS IN A CNN:
fully connected layer

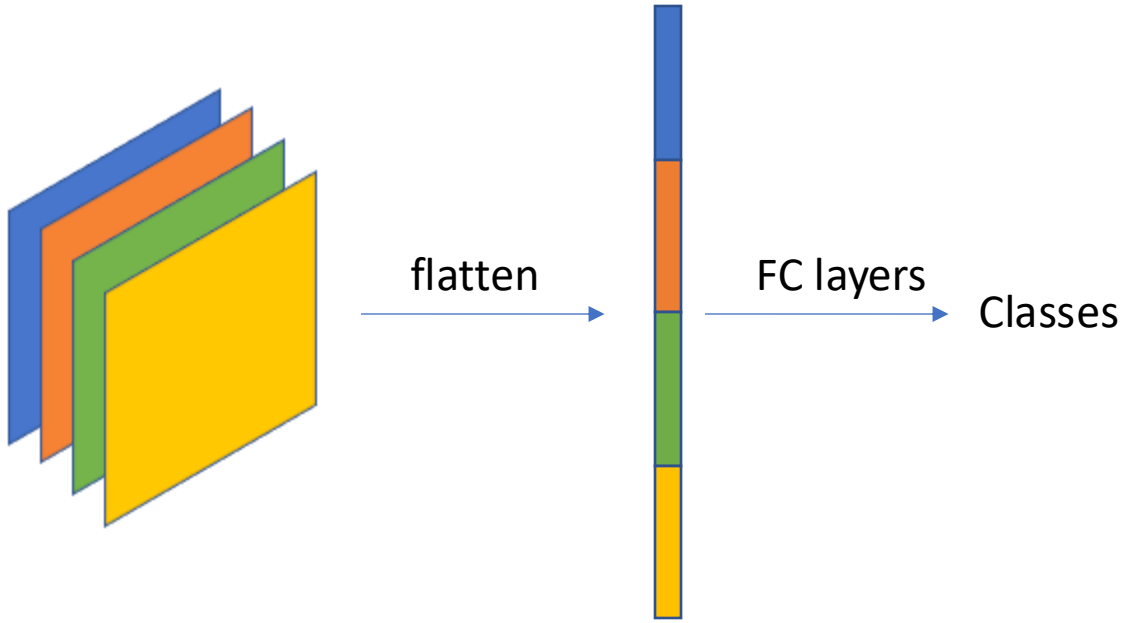
Fully-connected layer

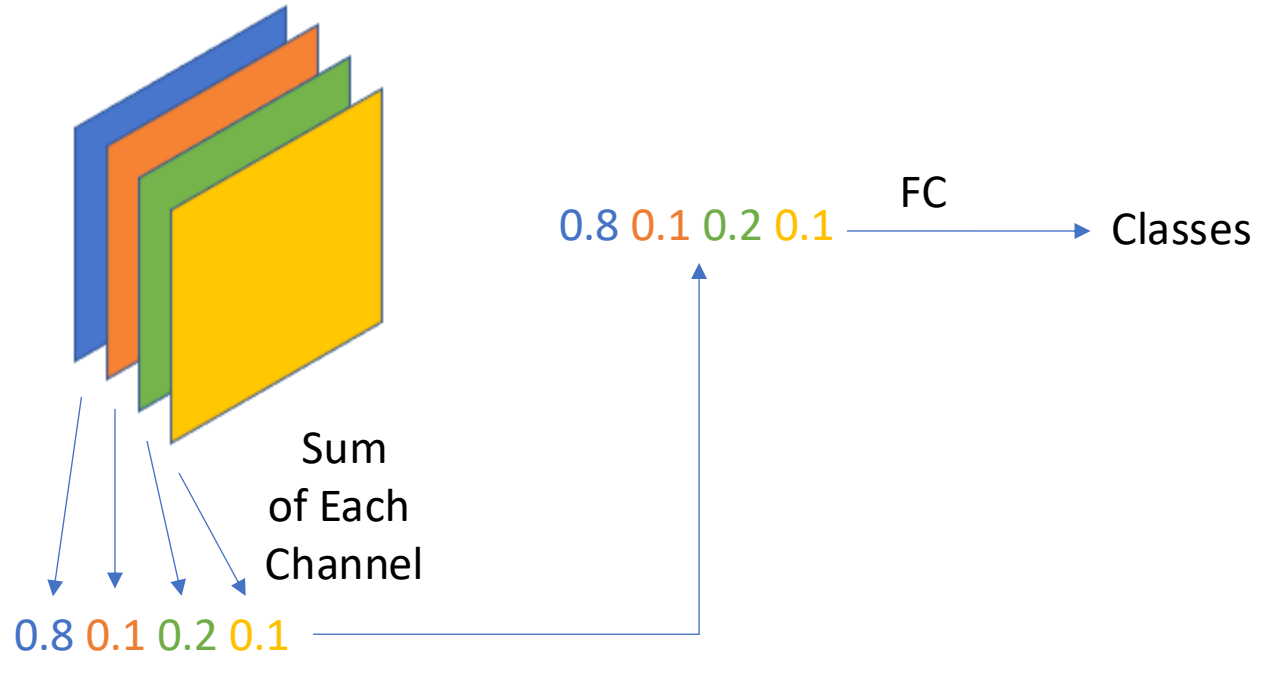
- At the top of the network for mapping the feature responses to output labels
- Full connectivity
- Can be many layers
- Various activation functions can be used



Alternative to FC: Global Average Pooling

"Network In Network", https://arxiv.org/pdf/1312.4400.pdf





Alternative to FC: Global Average Pooling

"Network In Network", https://arxiv.org/pdf/1312.4400.pdf

- We have n feature maps:

  $f_1, \ldots, f_n$.

- Global average pooling is then:

$$\bar{f}_i = \sum_{x,y} f_i(x, y)$$

- Classification scores are obtained by:

$$S_c = \sum_i w_i^c \bar{f}_i$$

- Advantages:
  - No parameters, hence significant improvement in terms of overfitting problem.
  - Forces the feature maps to capture confidence maps.
  - It is more suitable to the nature of CNNs.
  - Provides invariance to spatial transformations.

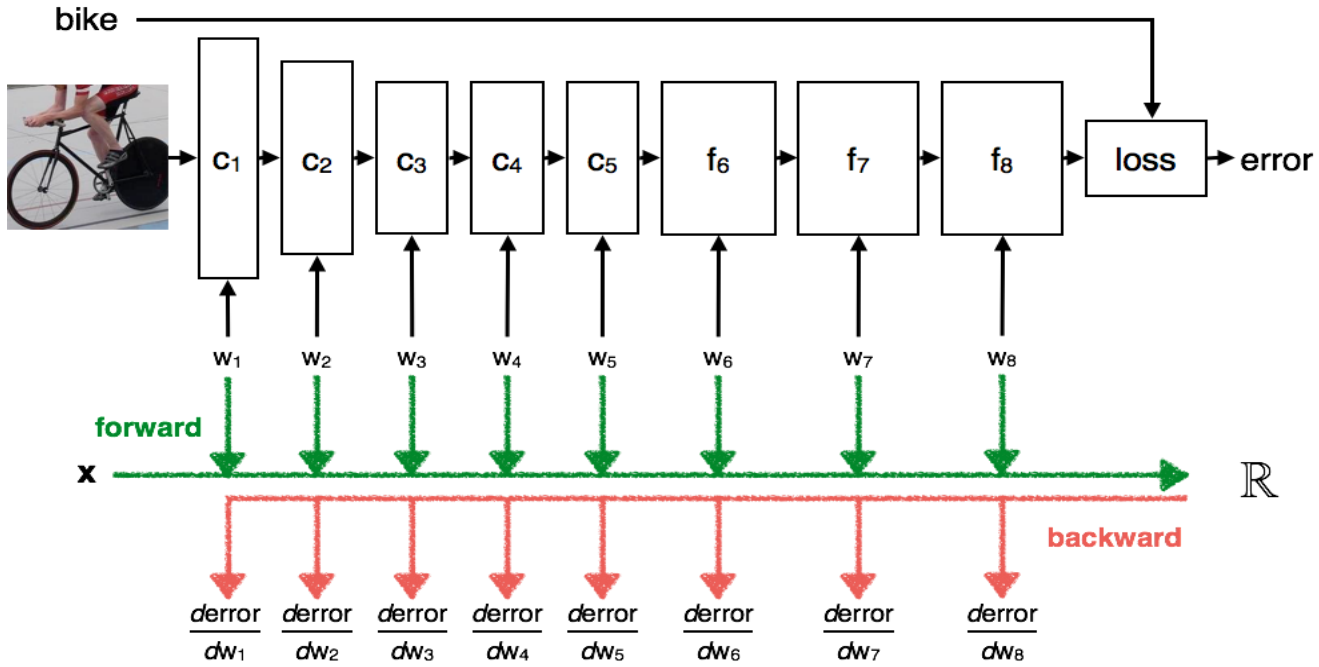
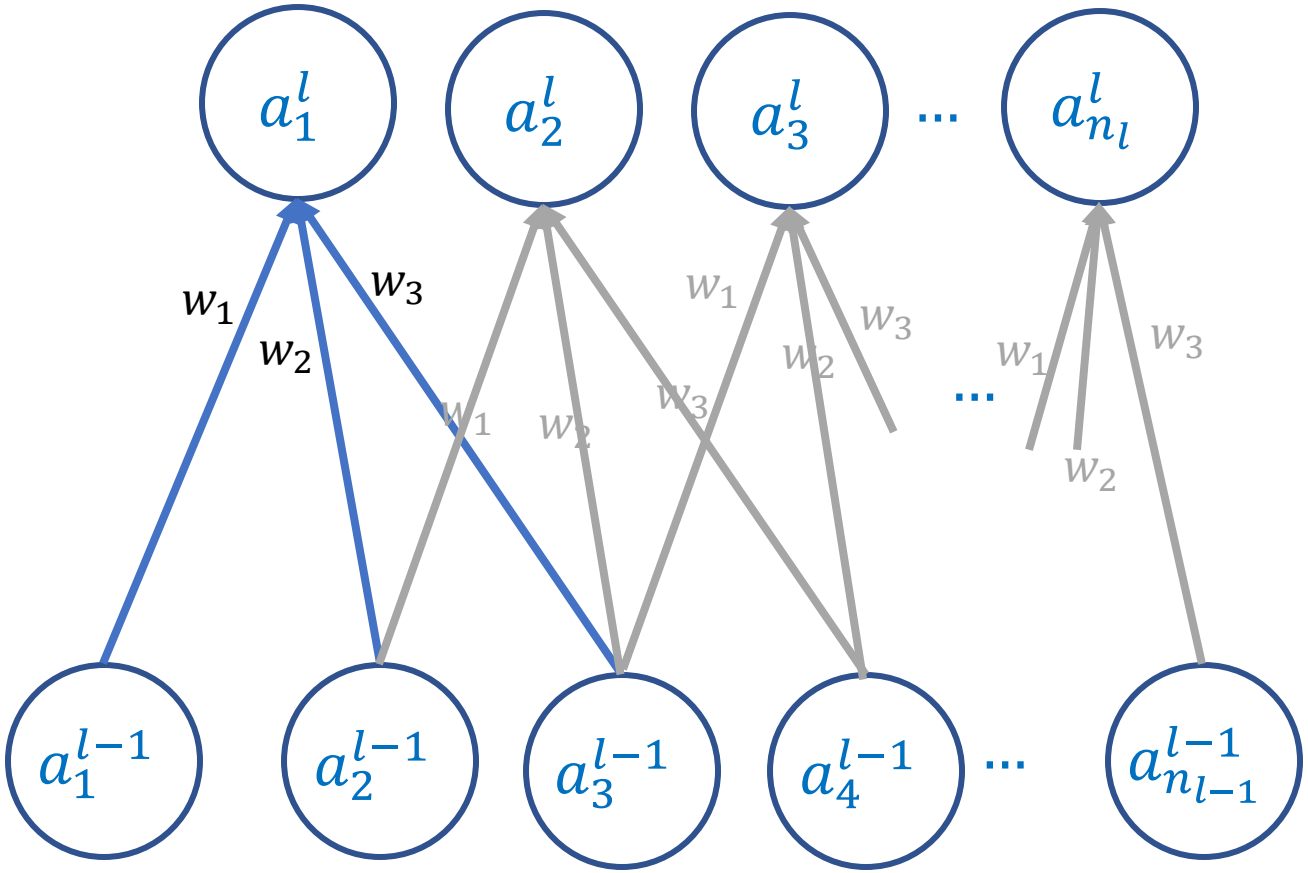


Fig: http://www.robots.ox.ac.uk/~vgg/practicals/cnn/

Training a CNN

Feed-forward through convolution



$$a_i^l = \sigma\left(t_i^l\right)$$

$$t_i^l = \sum_{j=1}^{F} w_j \cdot a_{i+j-1}^{l-1}$$

For example:

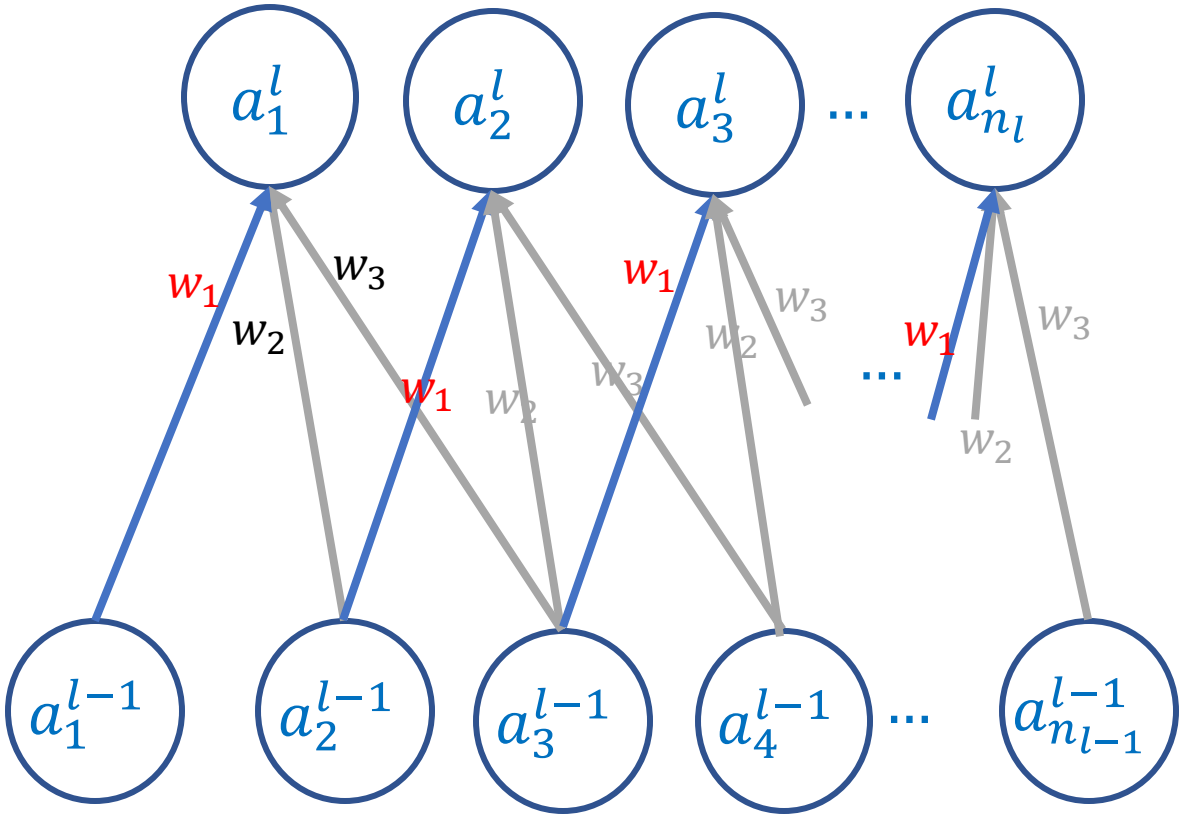
$$t_1^l = w_1 a_1^{l-1} + w_2 a_2^{l-1} + w_3 a_3^{l-1}$$

Backpropagation through convolution

Feedforward:

$$a_i^l = \sigma(t_i^l)$$

$$t_i^l = \sum_{j=1}^{F} w_j \cdot a_{i+j-1}^{l-1}$$



Gradient wrt. weights:

$$\frac{\partial L}{\partial w_k} = ?$$

$$= \frac{\partial L}{\partial a_1^l}\frac{\partial a_1^l}{\partial w_k} + \frac{\partial L}{\partial a_2^l}\frac{\partial a_2^l}{\partial w_k} \dots$$

$$= \sum_i \frac{\partial L}{\partial a_i^l}\frac{\partial a_i^l}{\partial w_k}$$

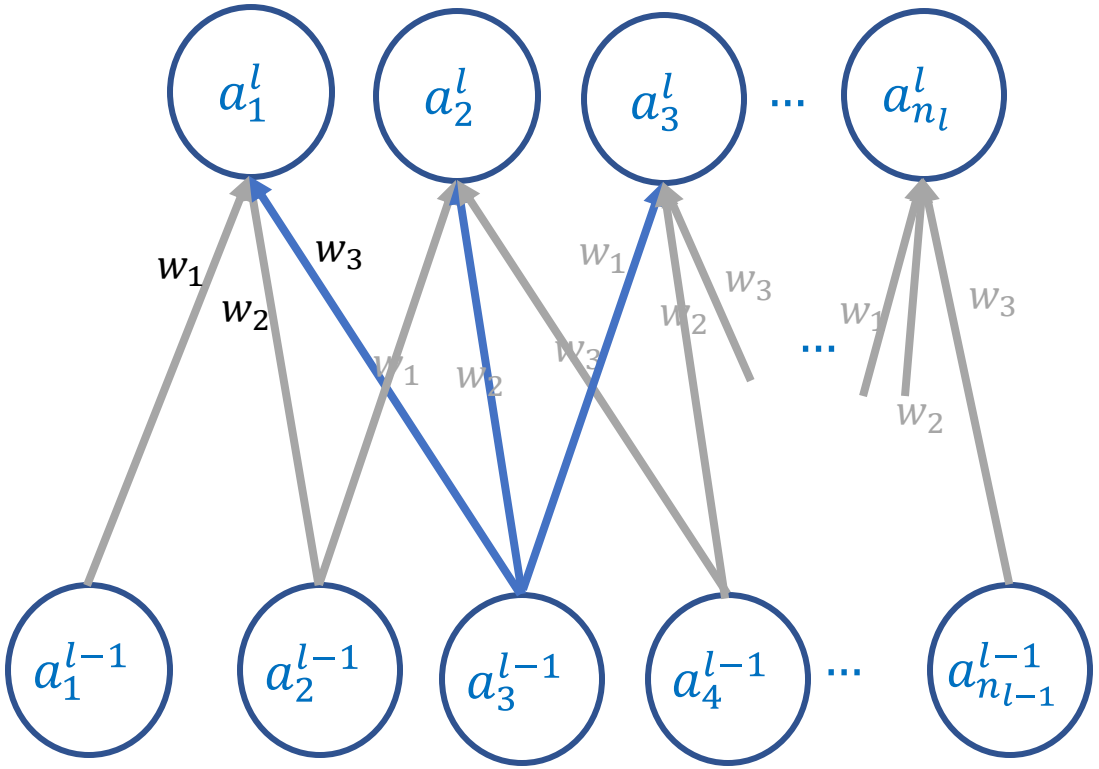
$$= \sum_i \frac{\partial L}{\partial a_i^l}\frac{\partial a_i^l}{\partial t_i^l}\frac{\partial t_i^l}{\partial w_k}$$

Backpropagation through convolution

Feedforward:

$$a_i^l = \sigma(t_i^l)$$

$$t_i^l = \sum_{j=1}^{F} w_j \cdot a_{i+j-1}^{l-1}$$



Gradient wrt. input layer:

$$\frac{\partial L}{\partial a_3^{l-1}} = ?$$

$$= \frac{\partial L}{\partial a_1^l}\frac{\partial a_1^l}{\partial t_1^l}\frac{\partial t_1^l}{\partial a_3^{l-1}} + \frac{\partial L}{\partial a_2^l}\frac{\partial a_2^l}{\partial t_2^l}\frac{\partial t_2^l}{\partial a_3^{l-1}} + \frac{\partial L}{\partial a_3^l}\frac{\partial a_3^l}{\partial t_3^l}\frac{\partial t_3^l}{\partial a_3^{l-1}}$$

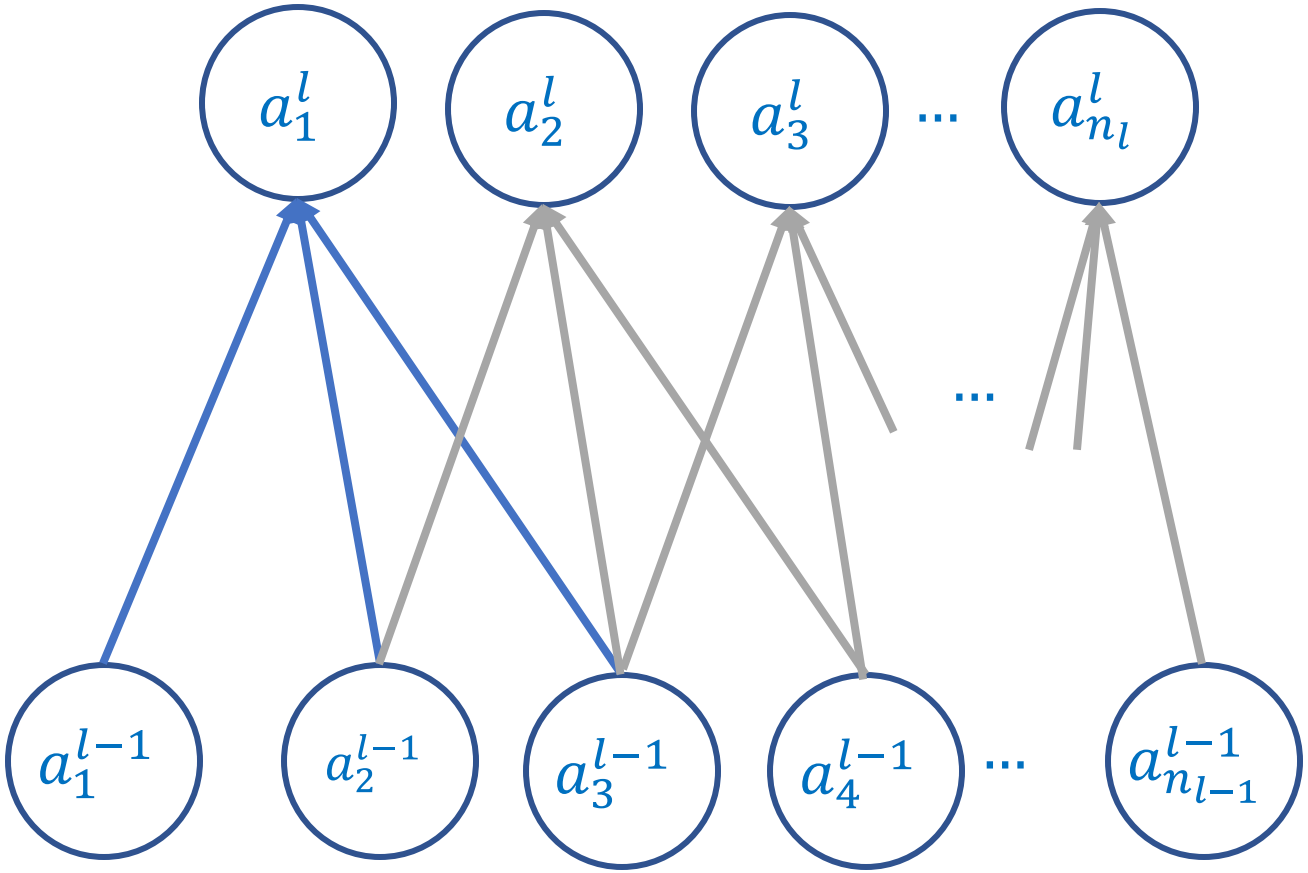
$$= \frac{\partial L}{\partial t_1^l} w_3 + \frac{\partial L}{\partial t_2^l} w_2 + \frac{\partial L}{\partial t_3^l} w_1$$

This is also convolution!

In general:

$$\frac{\partial L}{\partial a_i^{l-1}} = \sum_{j=1} \frac{\partial L}{\partial t_{i-j+1}^l} w_j$$

Feed-forward through pooling



$$a_i^l = \max \left\{ a_{i+j-1}^{l-1} \right\}_{j=1}^{F}$$

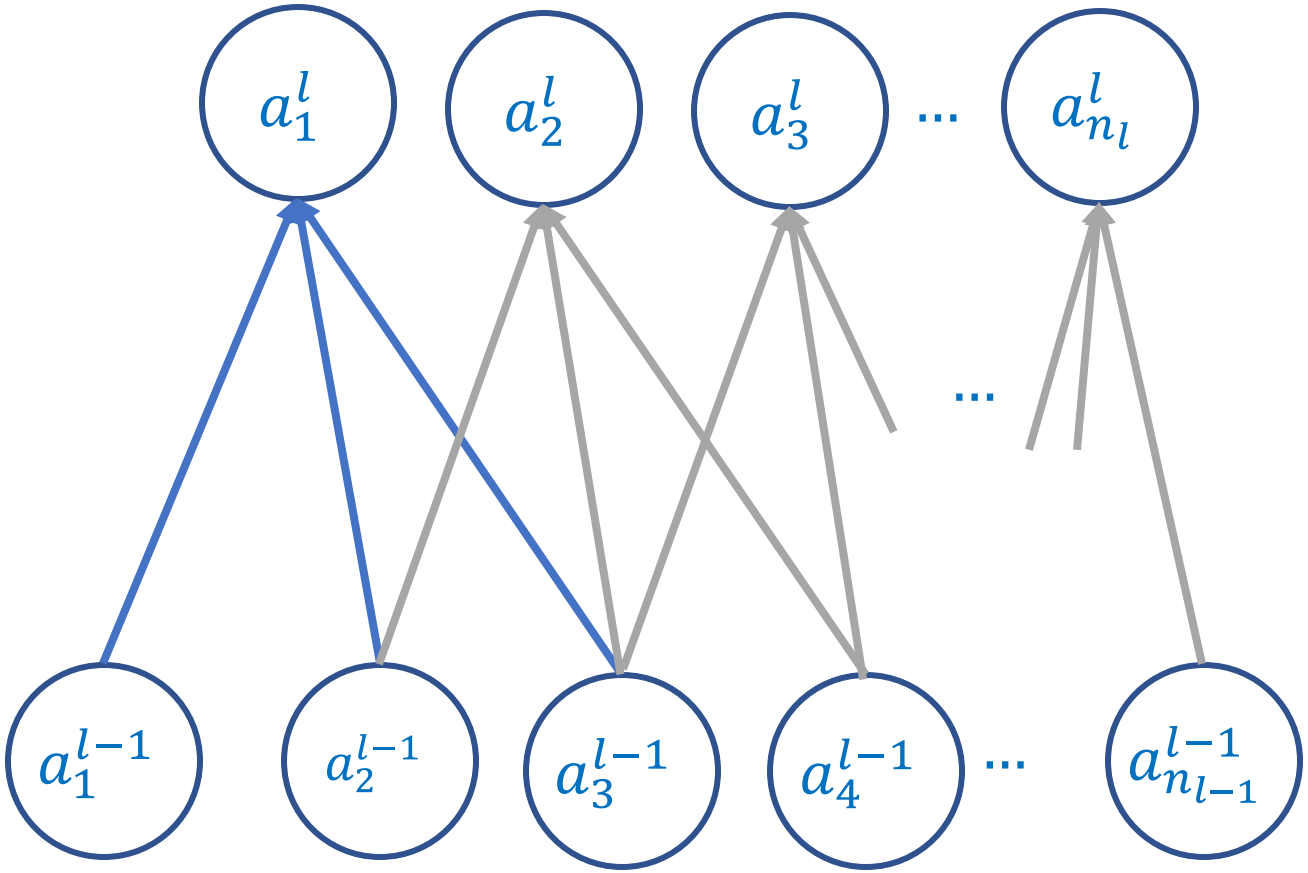
For example:

$$t_1^l = \max \left\{ a_1^{l-1}, a_2^{l-1}, a_3^{l-1} \right\}$$

Backpropagation through pooling

Feedforward:

$$a_i^l = \max\{a_{i+j-1}^{l-1}\}_{j=1}^{F}$$



Using derivative of max:

$$\frac{\partial L}{\partial a_i^{l-1}} = \frac{\partial L}{\partial t_k^l}\frac{\partial t_k^l}{\partial a_i^{l-1}}$$

$$= \begin{cases} \frac{\partial L}{\partial t_k^l}, & a_i^{l-1} \text{ is max} \\ 0, & \text{otherwise} \end{cases}$$

This requires that we save the index of the max activation (sometimes also called *the switches*) so that gradient "routing" is handled efficiently during backpropagation.

Backpropagation

- Backpropagation through non-linearity and fully-connected layers are straight-forward