CNN:
CONVOLUTIONAL NEURAL NETWORKS

Deep Learning for Computer Vision

Arthur Douillard

https://arthurdouillard.com/deepcourse

Convolutional Neural Networks

Images

Large dimensionality $W \times H \times C$

→ Classifying a 224x224x3 images to 1000 classes with a single layer means more than 150M parameters!

→ **Need some dimension reduction tool**

Fewly structured data

→ MLPs have no prior on spatiality

→ **need more data**

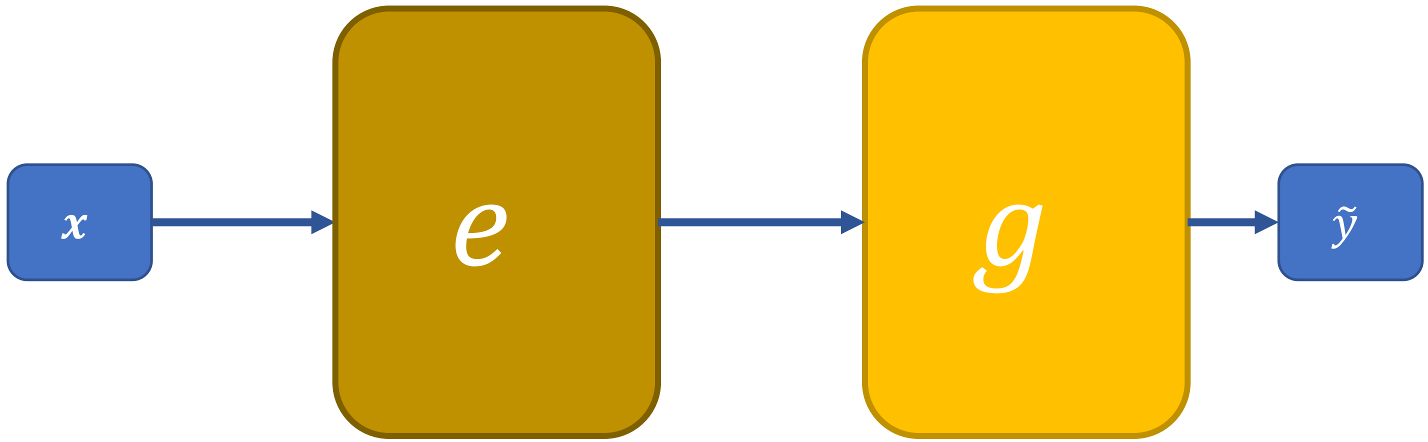
Before

Lot of work on how to extract interesting features

But with "hand-crafted" priors

Hand-crafted Features

Model

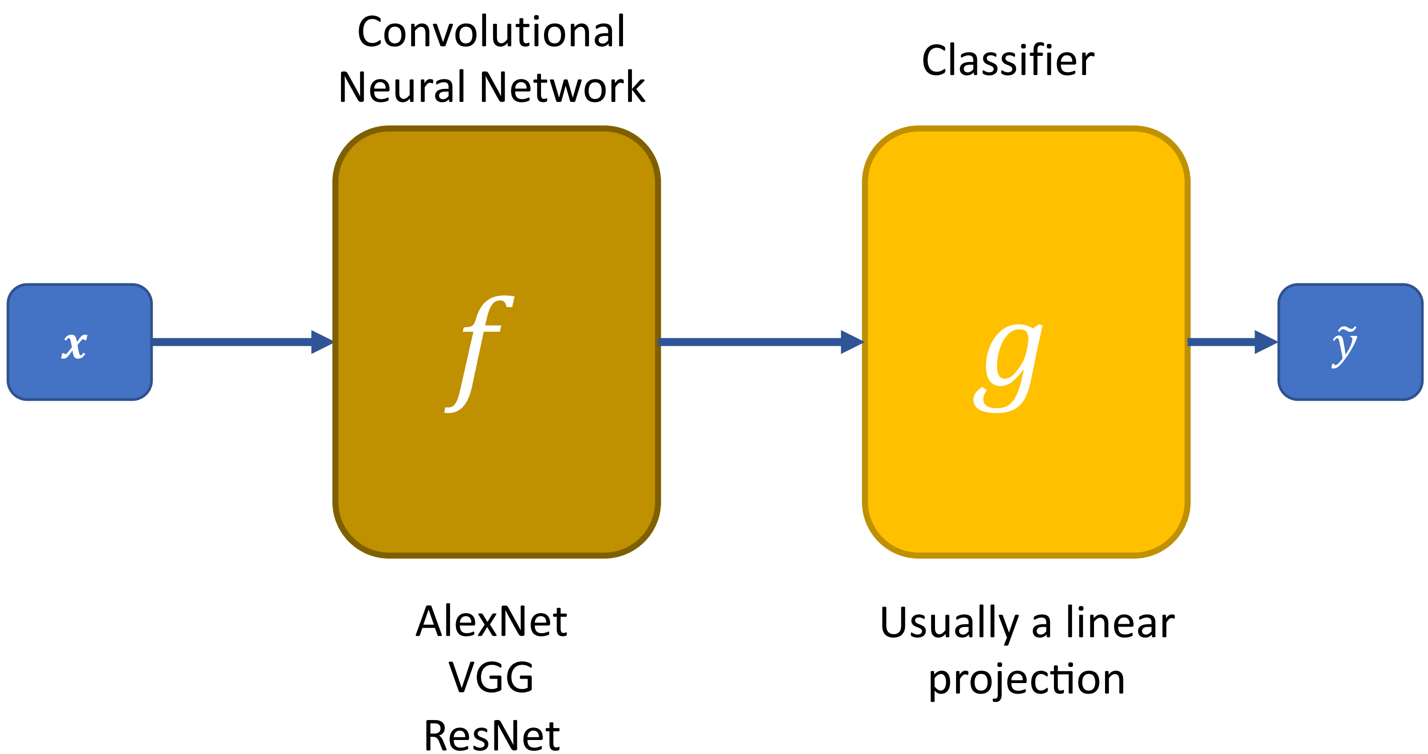


SIFT
HOG
Fisher
etc.

SVM
Gradient Boosting
Random Forest
etc.

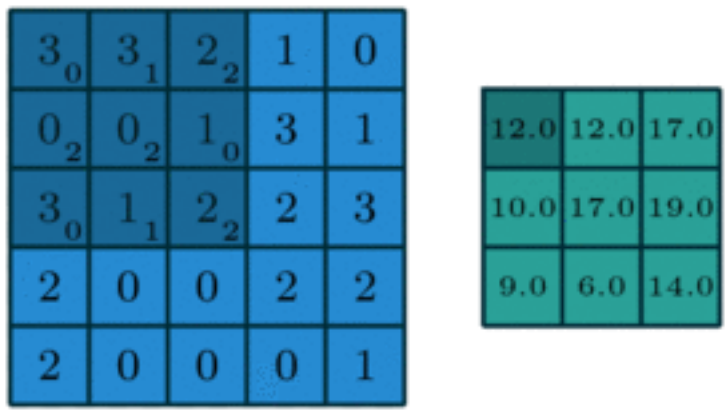
Now

Use gradient descent and neural network to learn how to extract the right features



Convolution kernel

$$\left(\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}\right)[2,2] = (a \cdot 1) + (b \cdot 2) + (c \cdot 3) + (d \cdot 4) + (e \cdot 5) + (f \cdot 6) + (g \cdot 7) + (h \cdot 8) + (i \cdot 9)$$



Translation invariance!

Light blue: input image
Dark blue: convolution kernel
Green: output features

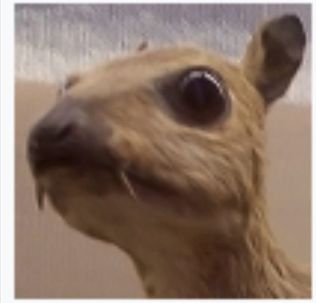
[GIF source]

NB: I'm talking about convolutions, but it's actually cross-correlation. The only difference is a transpose on the image.

Convolution Neural Network

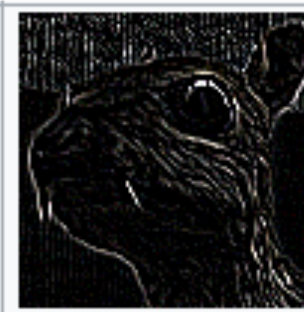
Identity

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$



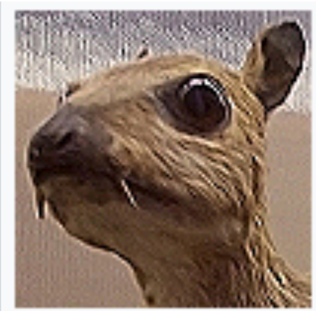
Edge detection

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$



Sharpen

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



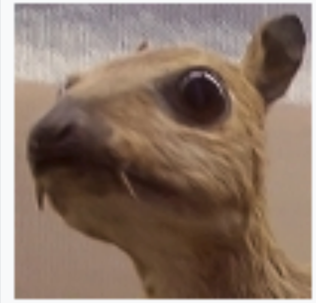
[Wikipedia](https://en.wikipedia.org)

Convolution Neural Network

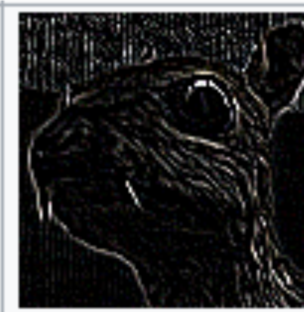
A convolution kernel is a learned weight of the network

$$W = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

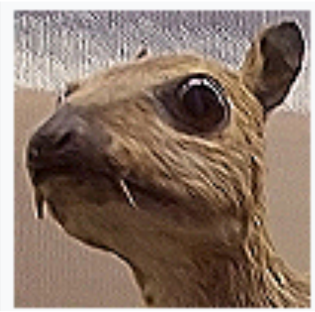
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$



$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

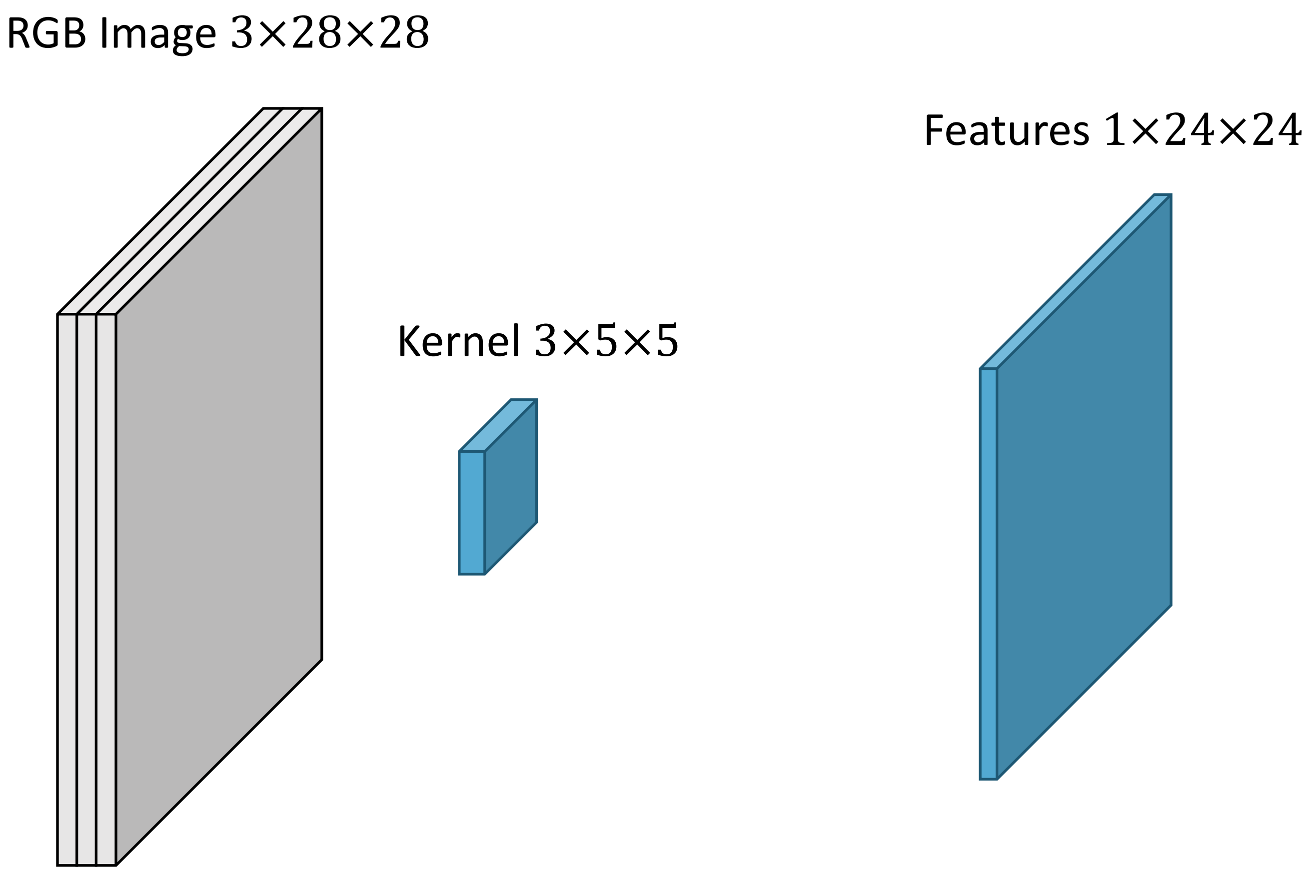


$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



Wikipedia

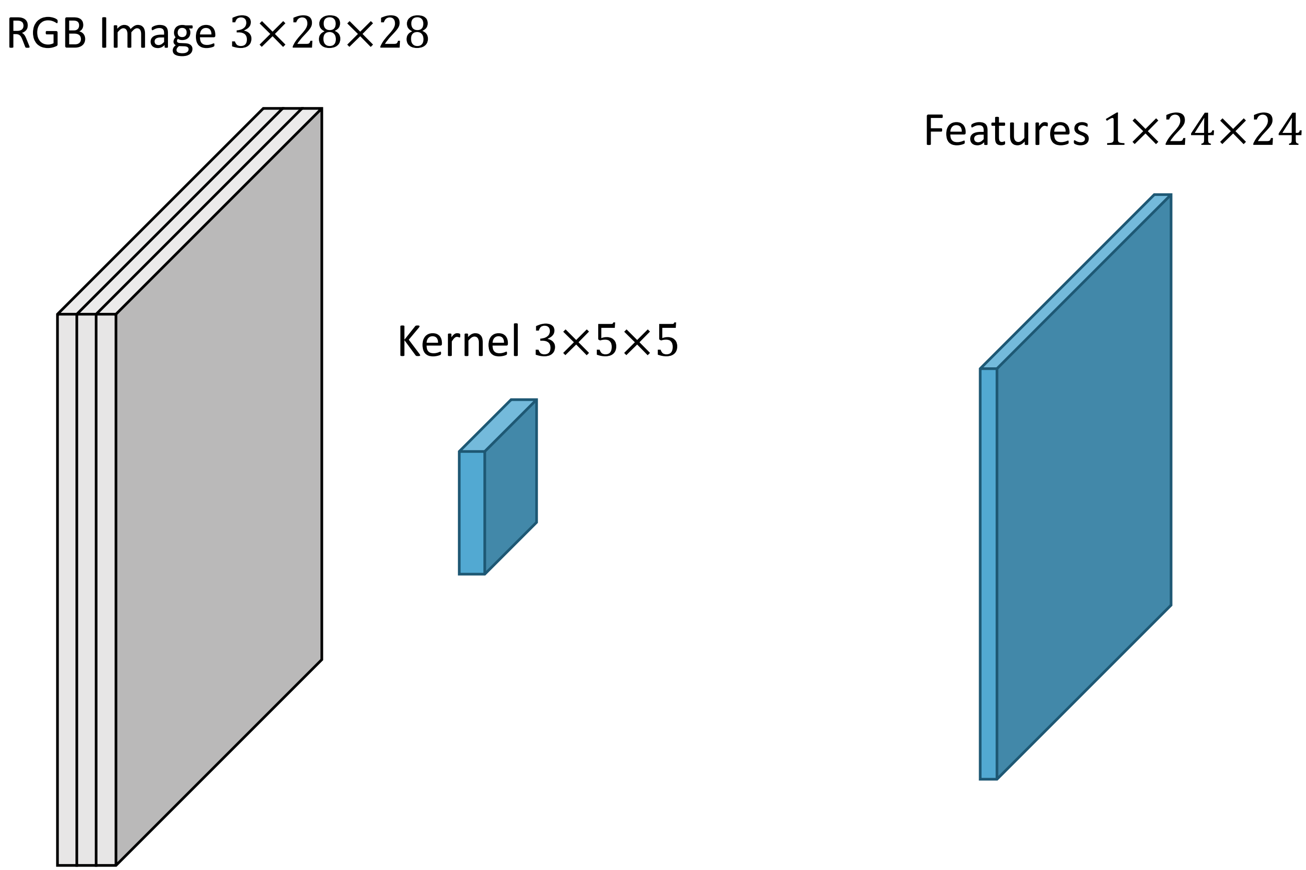
On RGB images



One kernel 5×5 is applied on Red, one on Green, and one on Blue.
The three results are summed together pixel-wise.

$$X * W = \sum_{c_i}^{3} X_{c_i} * W_{c_i}$$

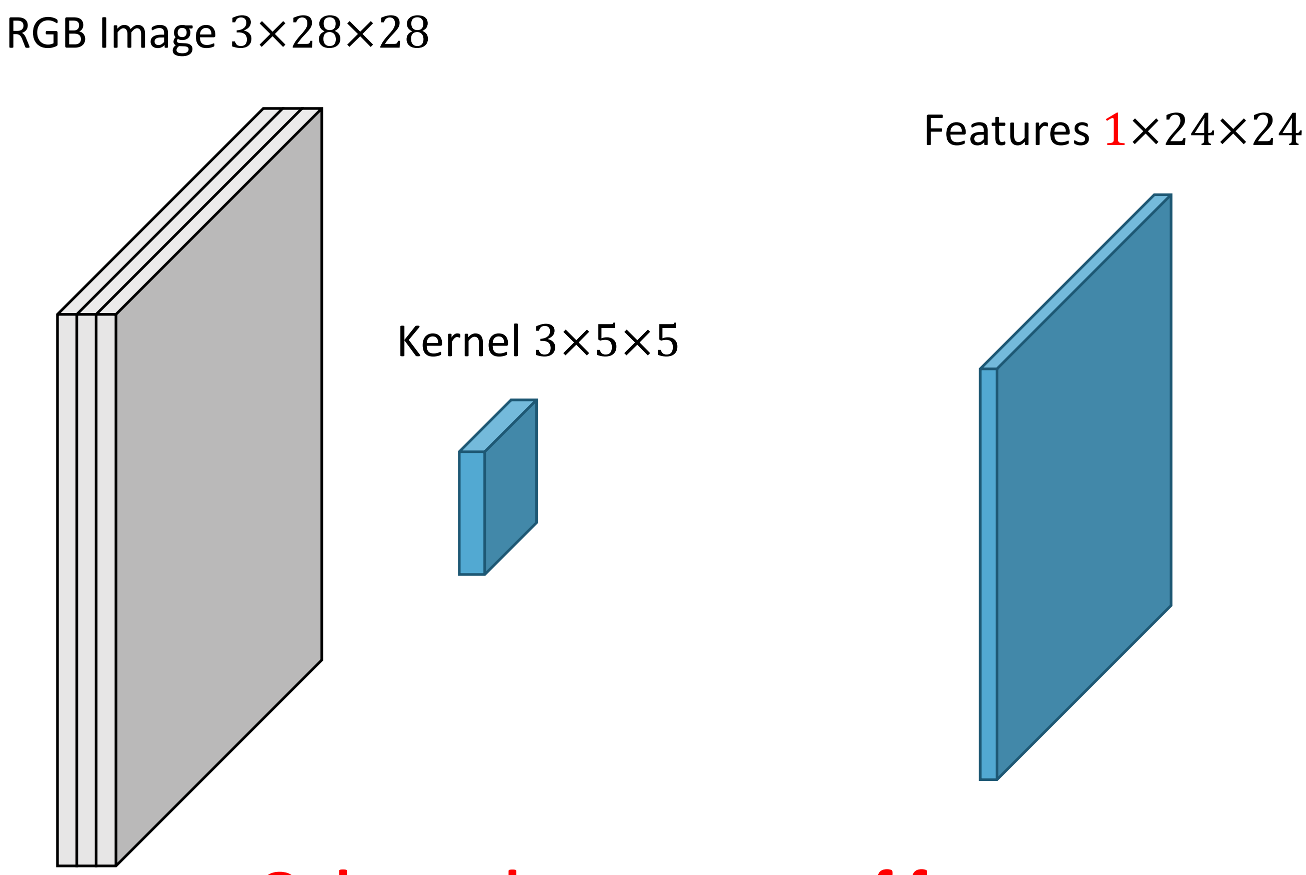
On RGB images



One kernel 5×5 per input channel!

$$X * W = \sum_{c_i}^{3} X_{c_i} * W_{c_i}$$

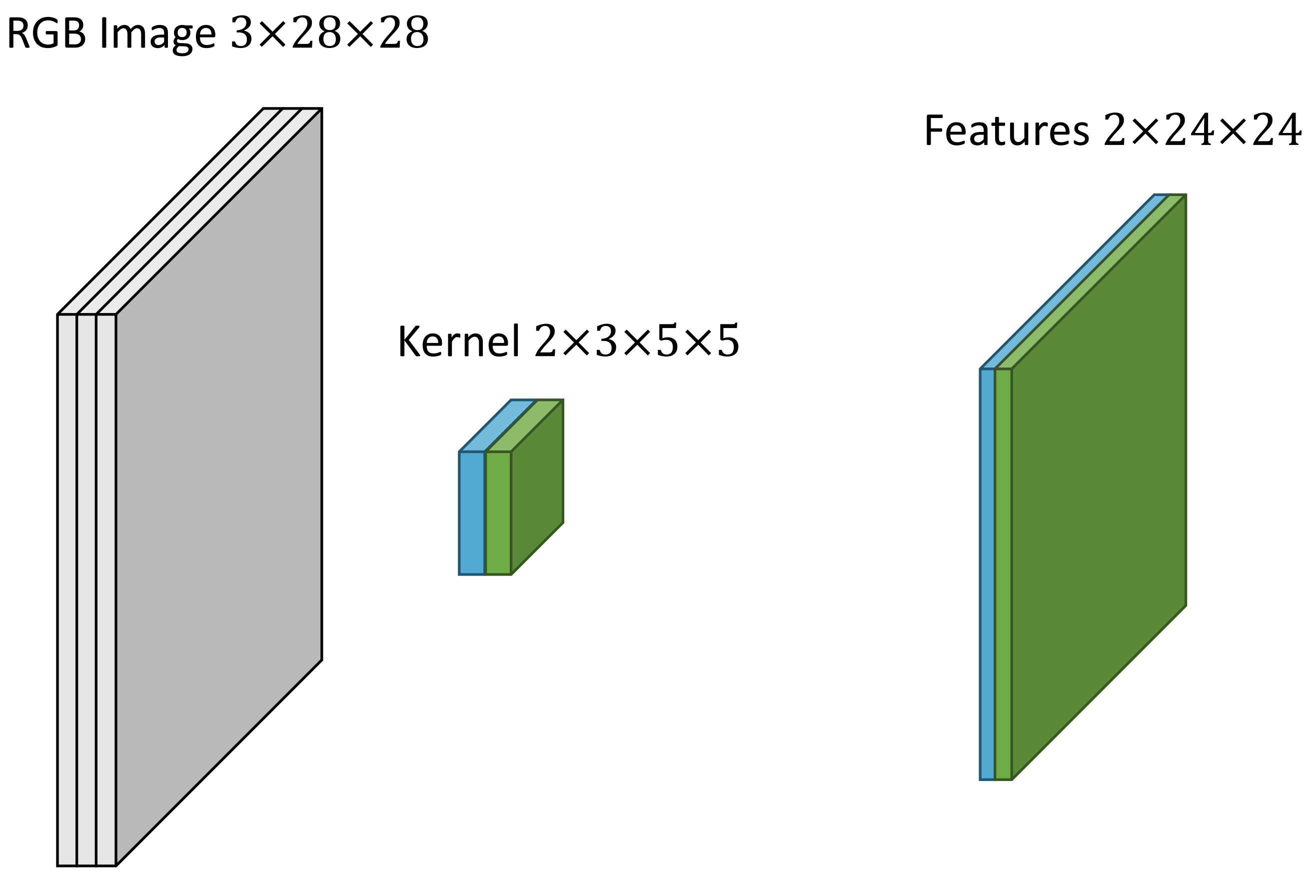
On RGB images



Only produce one set of features…

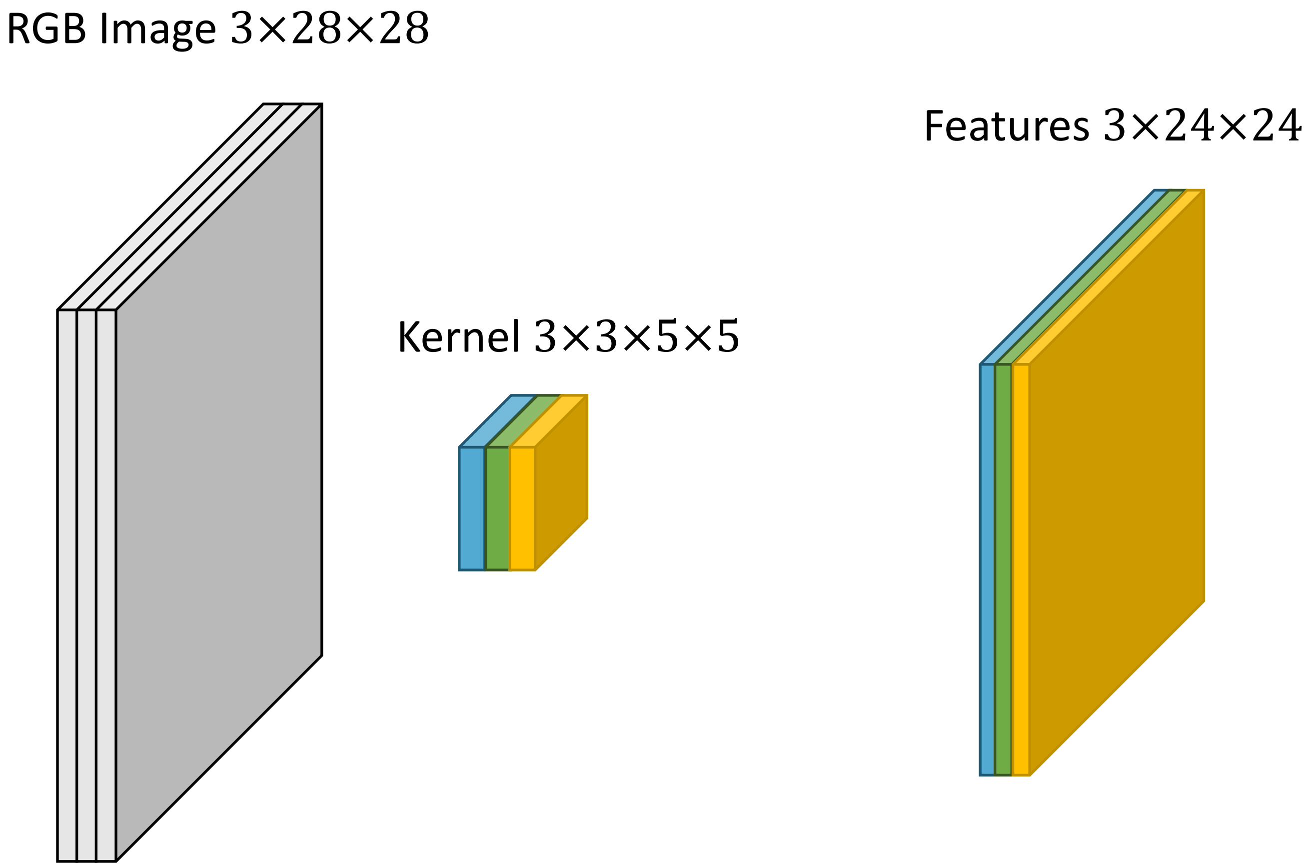
$$X * W = \sum_{c_i}^{3} X_{c_i} * W_{c_i}$$

On RGB images



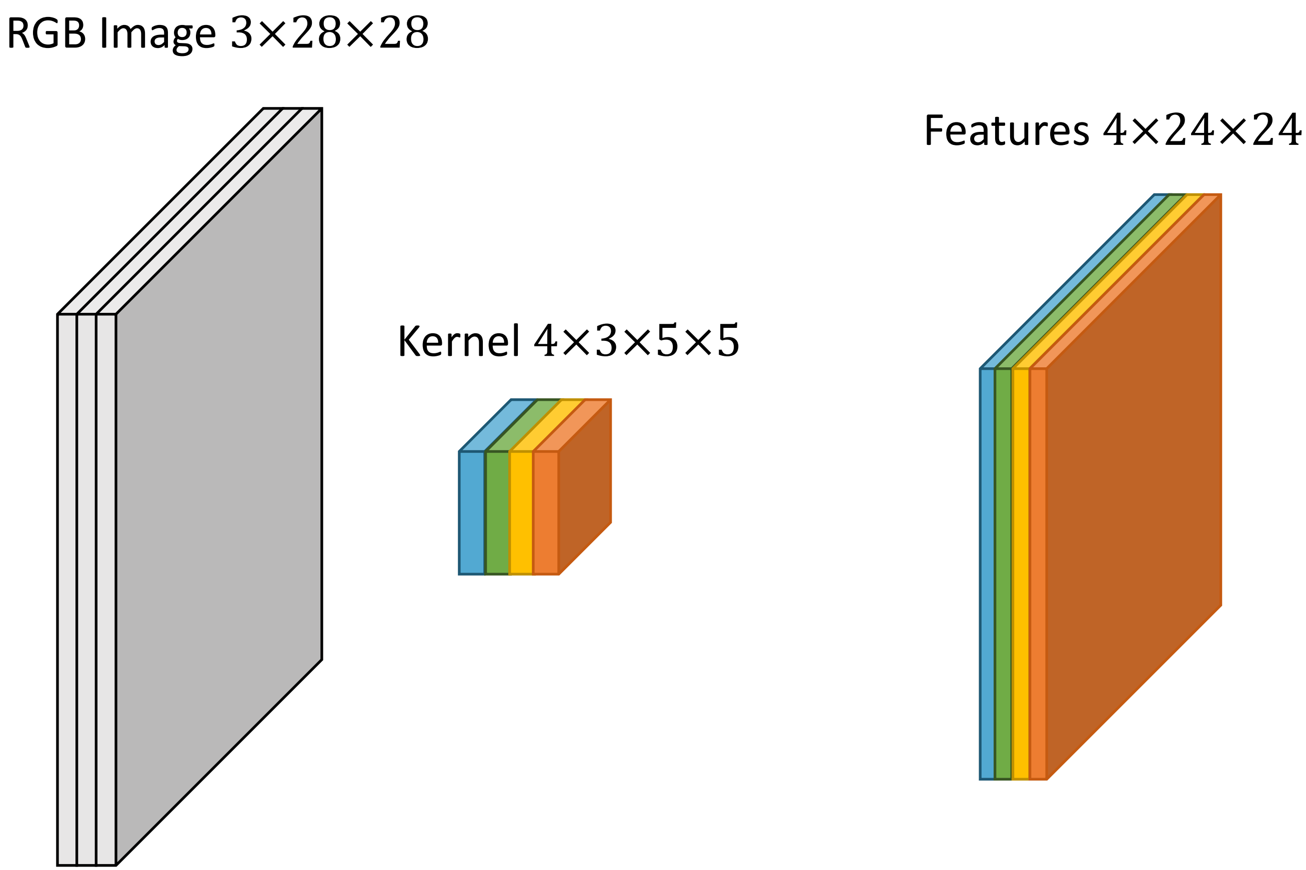
$$(X * W)_{c_o} = \sum_{c_i}^{3} X_{c_i} * W_{c_o,c_i}$$

On RGB images



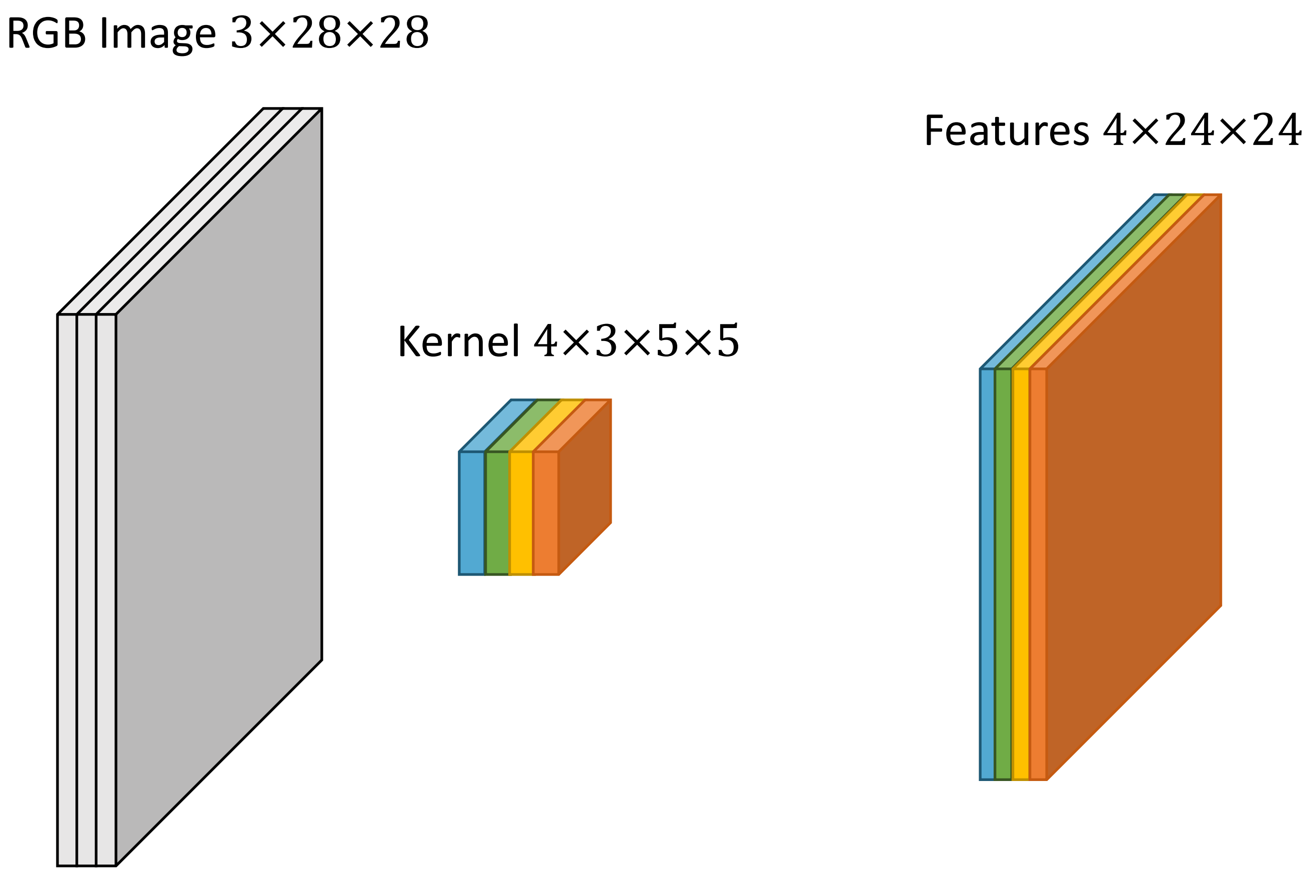
$$(X * W)_{c_o} = \sum_{c_i}^{3} X_{c_i} * W_{c_o, c_i}$$

On RGB images



$$(X * W)_{c_o} = \sum_{c_i}^{3} X_{c_i} * W_{c_o,c_i}$$

On RGB images



Likewise, these 4-channels features will be input to another kernel

$$(X * W)_{c_o} = \sum_{c_i}^{3} X_{c_i} * W_{c_o,c_i}$$

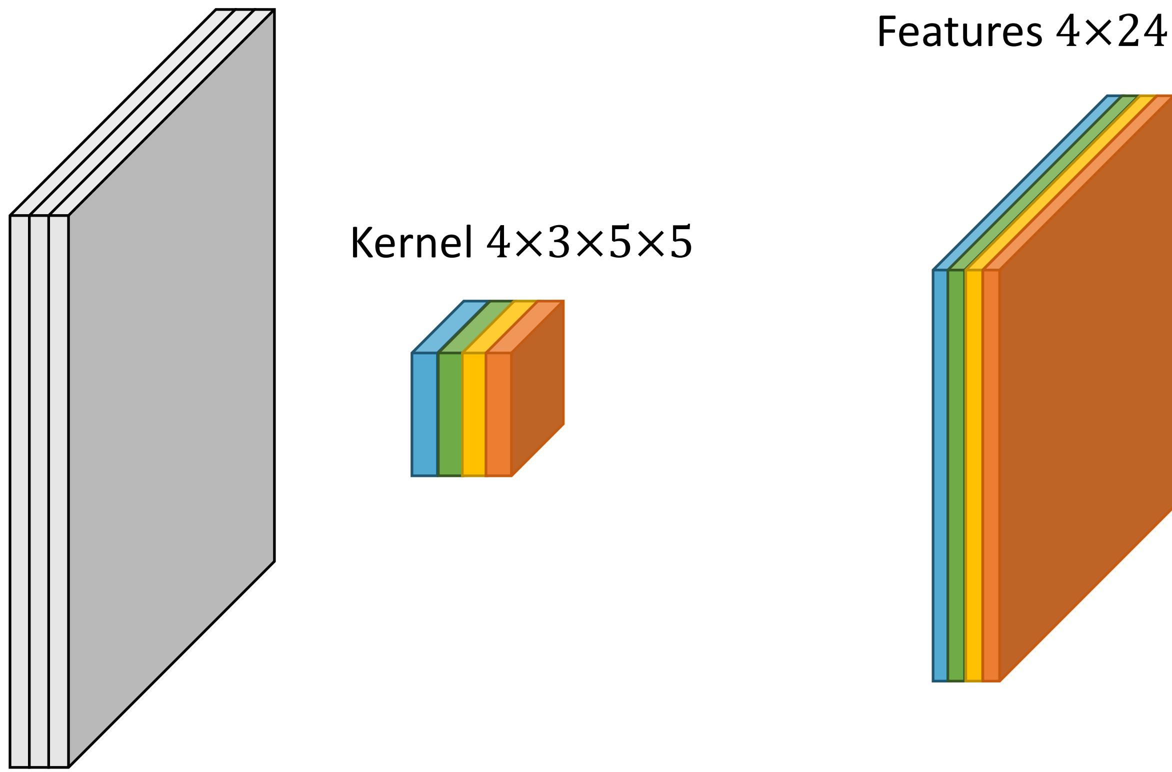
On RGB images

RGB Image 3×28×28

Notice features spatial dimension changed?

Features 4×24×24

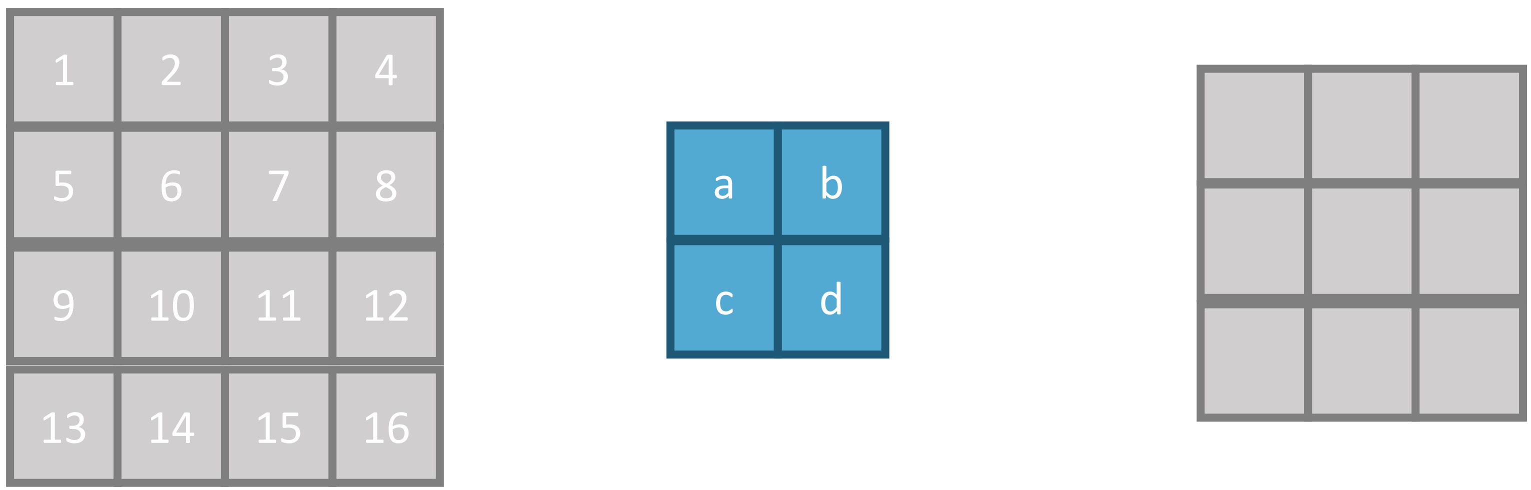
Kernel 4×3×5×5



$$(X * W)_{c_o} = \sum_{c_i}^{3} X_{c_i} * W_{c_o,c_i}$$

Kernel size, stride, and padding

Input 4×4 Kernel 2×2 Output 3×3

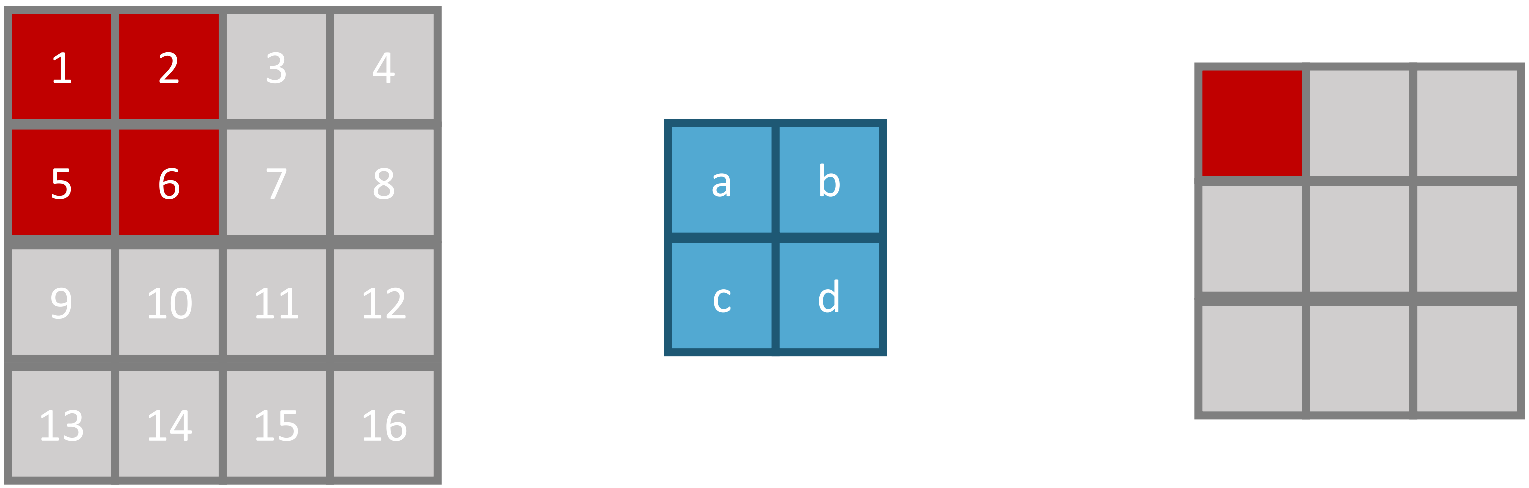


$w = 4, k = 2, s = 1, p = 0$

Output volume: $$\frac{w - k + 2p}{s} + 1$$

Kernel size, stride, and padding

Input 4×4 Kernel 2×2 Output 3×3



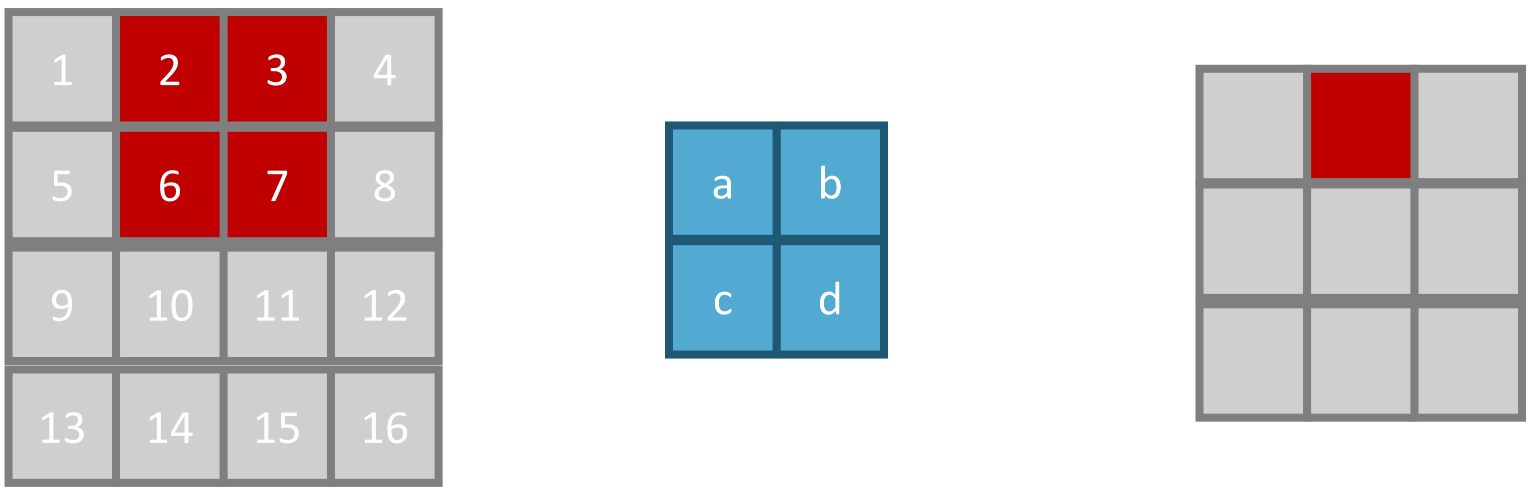
$$1a + 2b + 5c + 6d$$

$w = 4, k = 2, s = 1, p = 0$

Output volume: $\dfrac{w - k + 2p}{s} + 1$

Kernel size, stride, and padding

Input 4×4 Kernel 2×2 Output 3×3



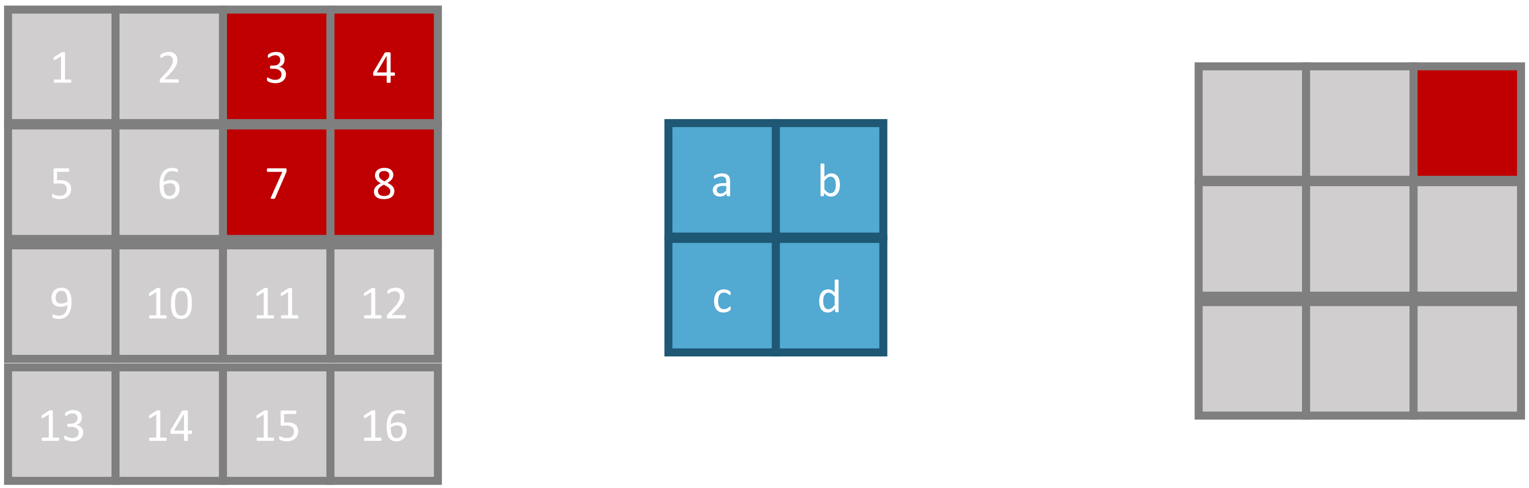
$$2a + 3b + 6c + 7d$$

$w = 4, k = 2, s = 1, p = 0$

Output volume: $\dfrac{w - k + 2p}{s} + 1$

Kernel size, stride, and padding

Input 4×4 | Kernel 2×2 | Output 3×3

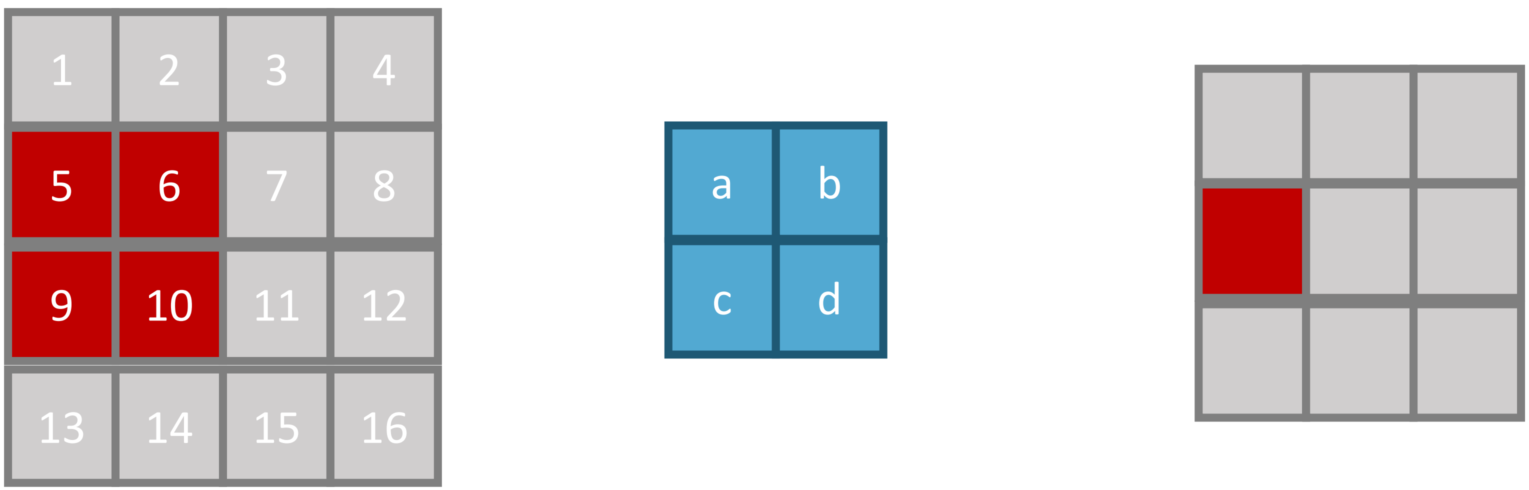


$w = 4, k = 2, s = 1, p = 0$

Output volume: $\dfrac{w - k + 2p}{s} + 1$

Kernel size, stride, and padding

Input 4×4 Kernel 2×2 Output 3×3

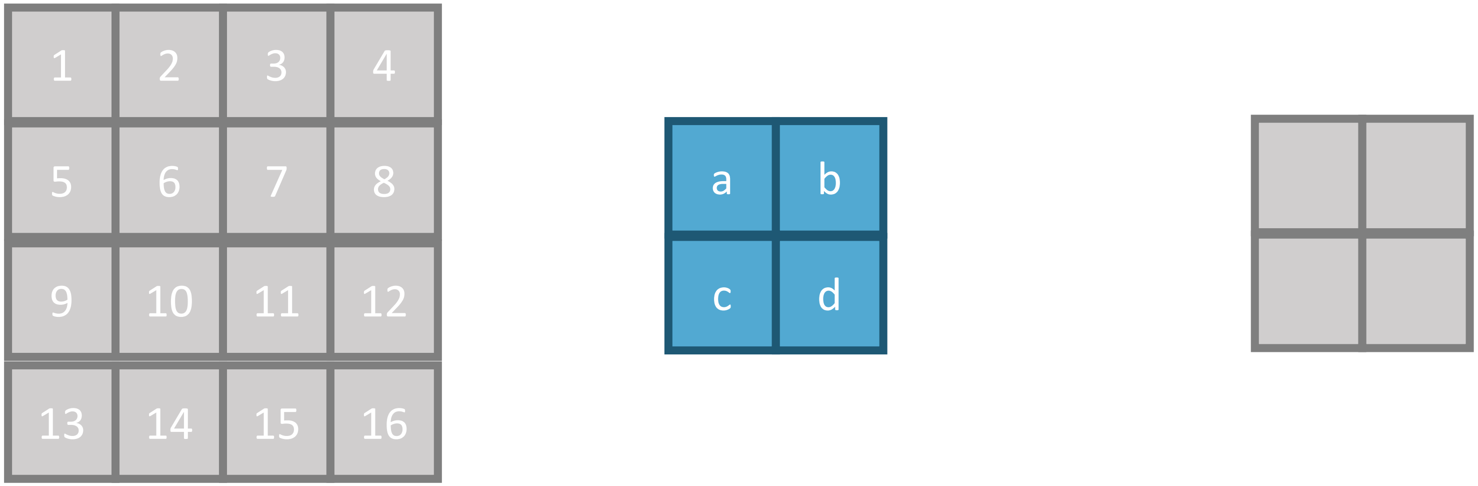


$w = 4, k = 2, s = 1, p = 0$

Output volume: $\dfrac{w - k + 2p}{s} + 1$

Kernel size, stride, and padding

Input 4×4 | Kernel 2×2 | Output 2×2



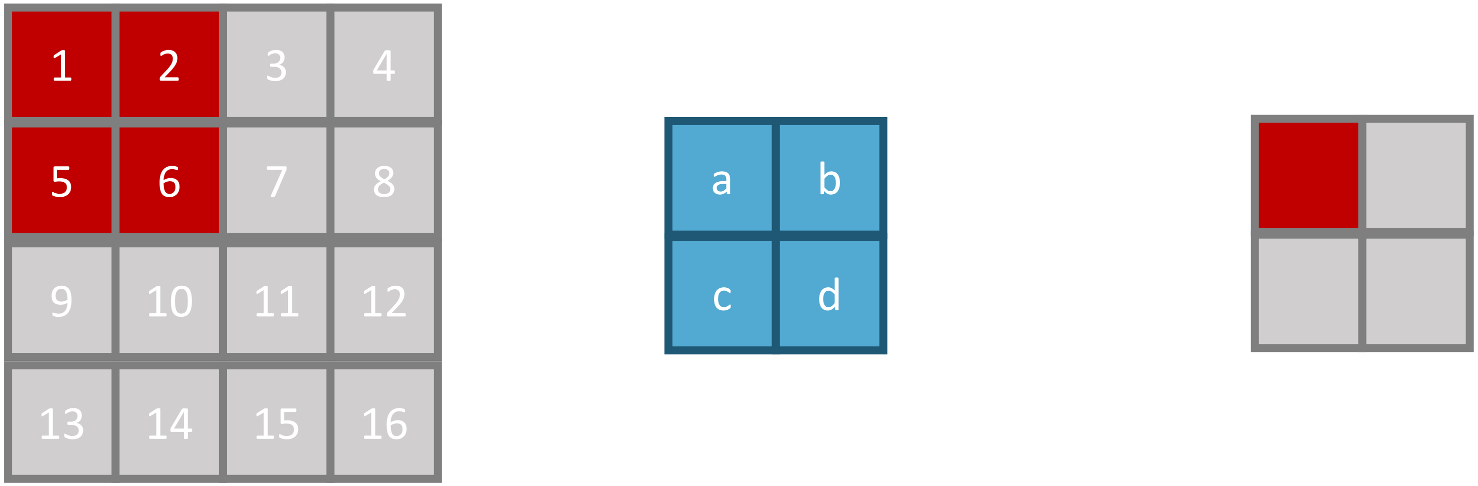
Stride equal to 2

$w = 4, k = 2, \boldsymbol{s} = \mathbf{2}, p = 0$

Output volume: $\frac{w - k + 2p}{s} + 1$

Kernel size, stride, and padding

Input 4×4 Kernel 2×2 Output 2×2



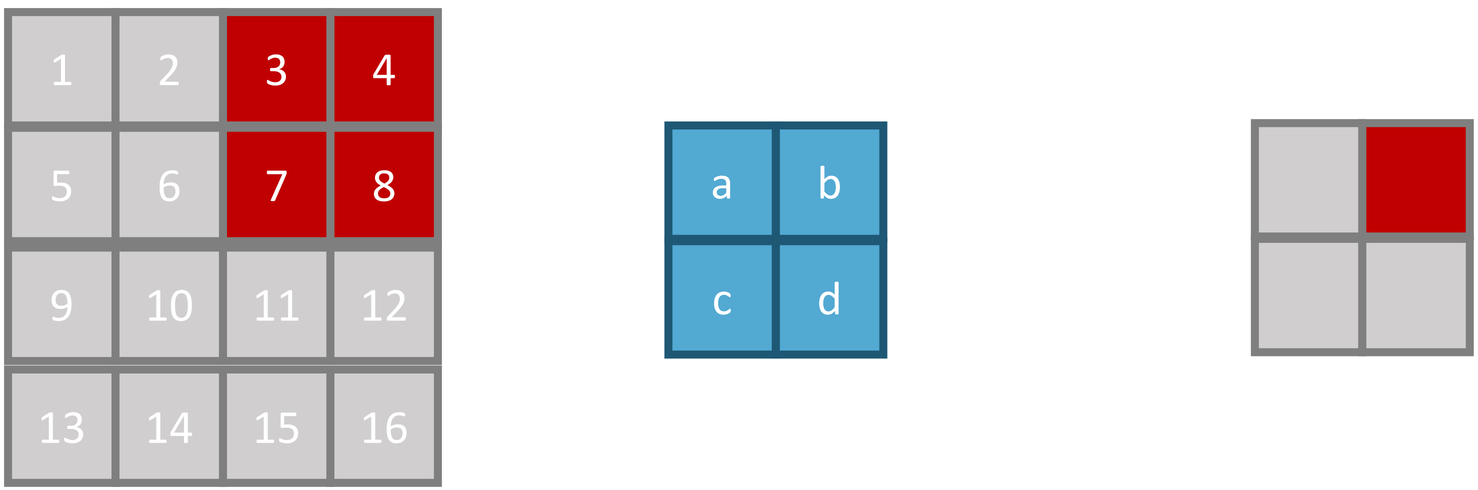
Stride equal to 2

$w = 4, k = 2, \boldsymbol{s} = \mathbf{2}, p = 0$

Output volume: $\dfrac{w - k + 2p}{s} + 1$

Kernel size, stride, and padding

Input 4×4 | Kernel 2×2 | Output 2×2



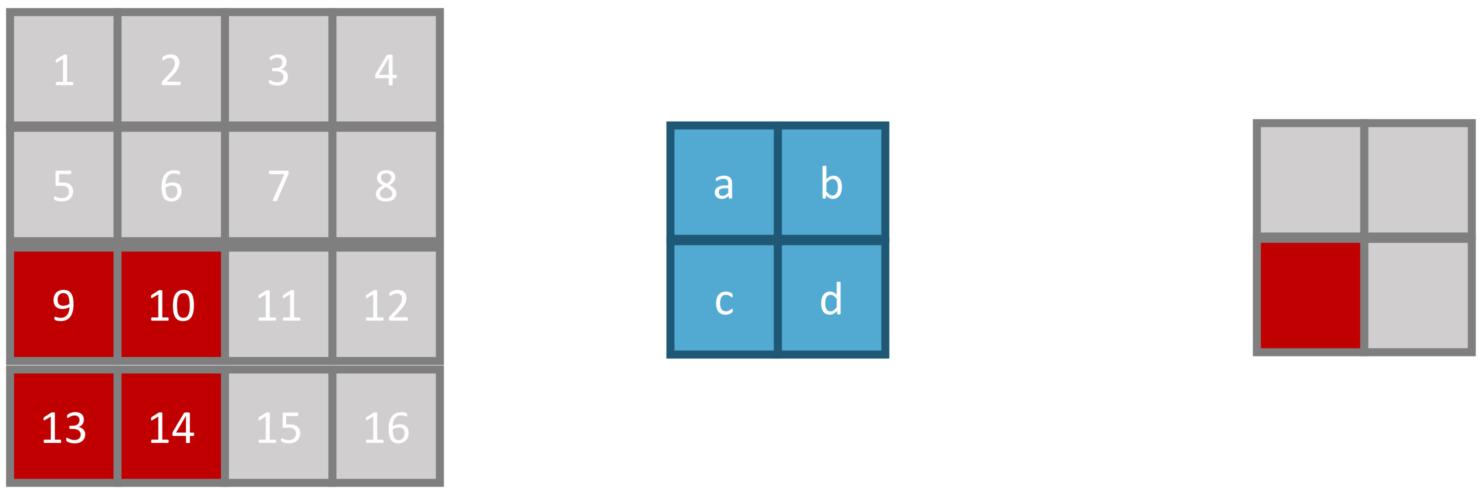
Stride equal to 2

$w = 4, k = 2, \boldsymbol{s} = \mathbf{2}, p = 0$

Output volume: $\dfrac{w - k + 2p}{s} + 1$

Kernel size, stride, and padding

Input 4×4 Kernel 2×2 Output 2×2



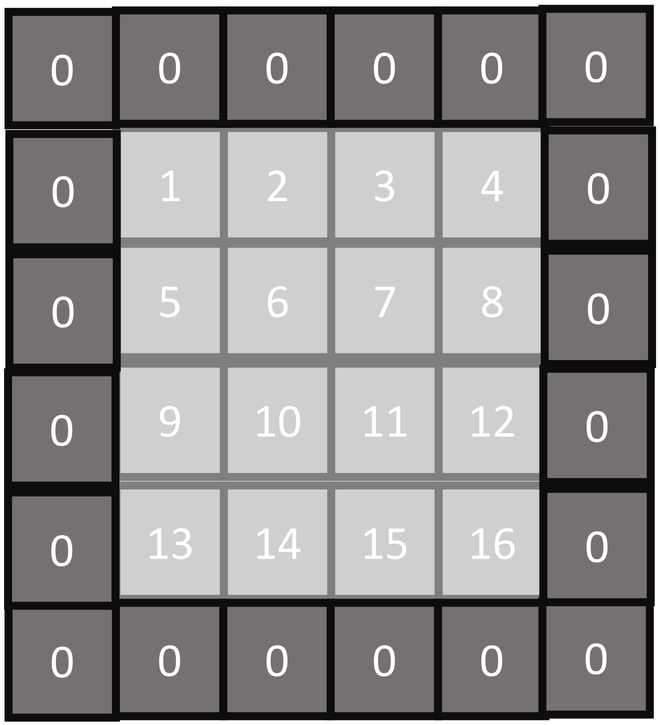
Stride equal to 2

$w = 4, k = 2, \boldsymbol{s} = \mathbf{2}, p = 0$

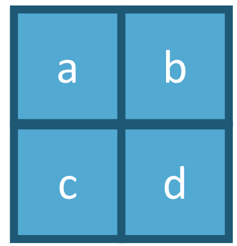
Output volume: $$\frac{w - k + 2p}{s} + 1$$

Kernel size, stride, and padding

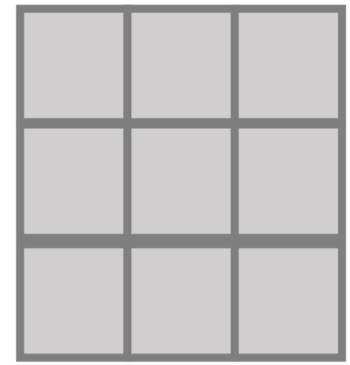
Input 4×4



Kernel 2×2



Output 2×2



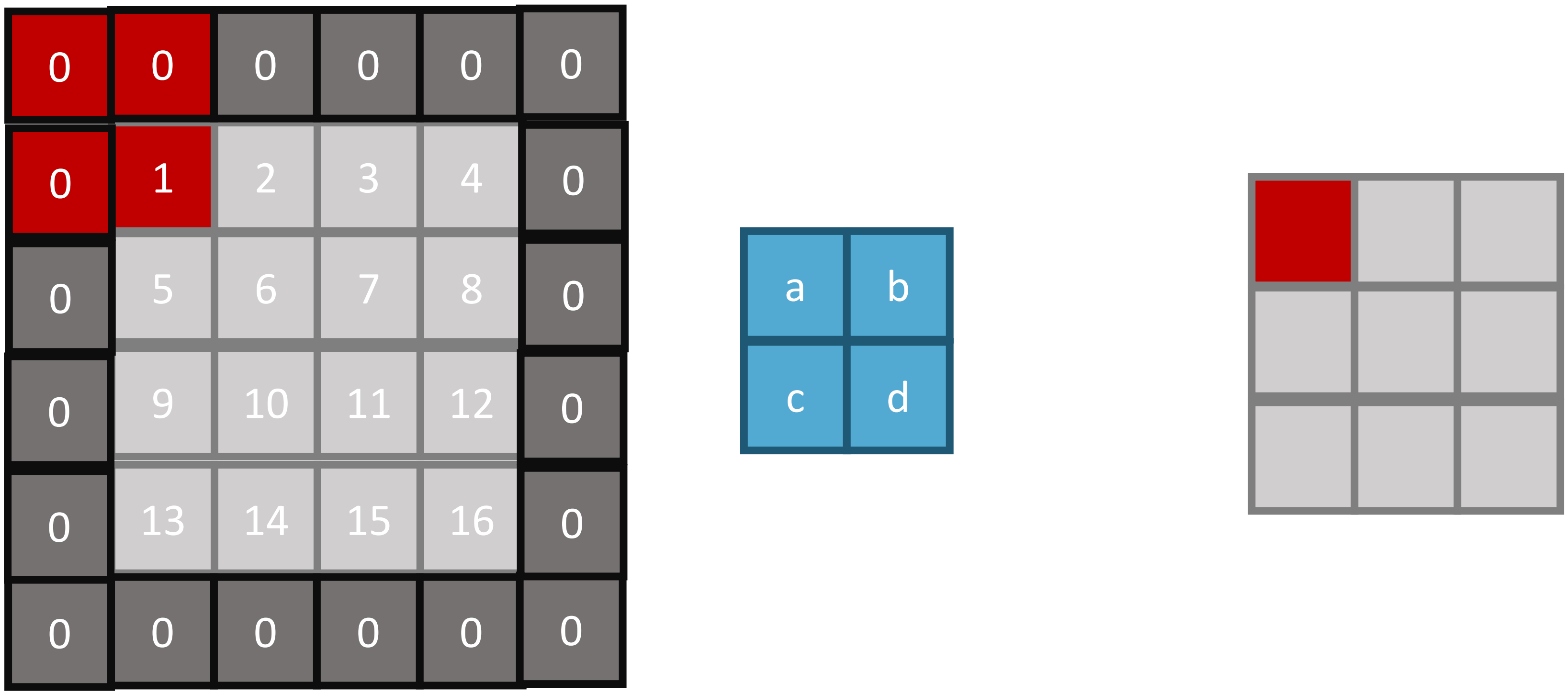
Padding equal to 1, useful to keep spatial dimension constant

$w = 4, k = 2, s = 2, \boldsymbol{p} = \mathbf{1}$

Output volume: $\dfrac{w - k + 2p}{s} + 1$

Kernel size, stride, and padding

Input 4×4 Kernel 2×2 Output 2×2



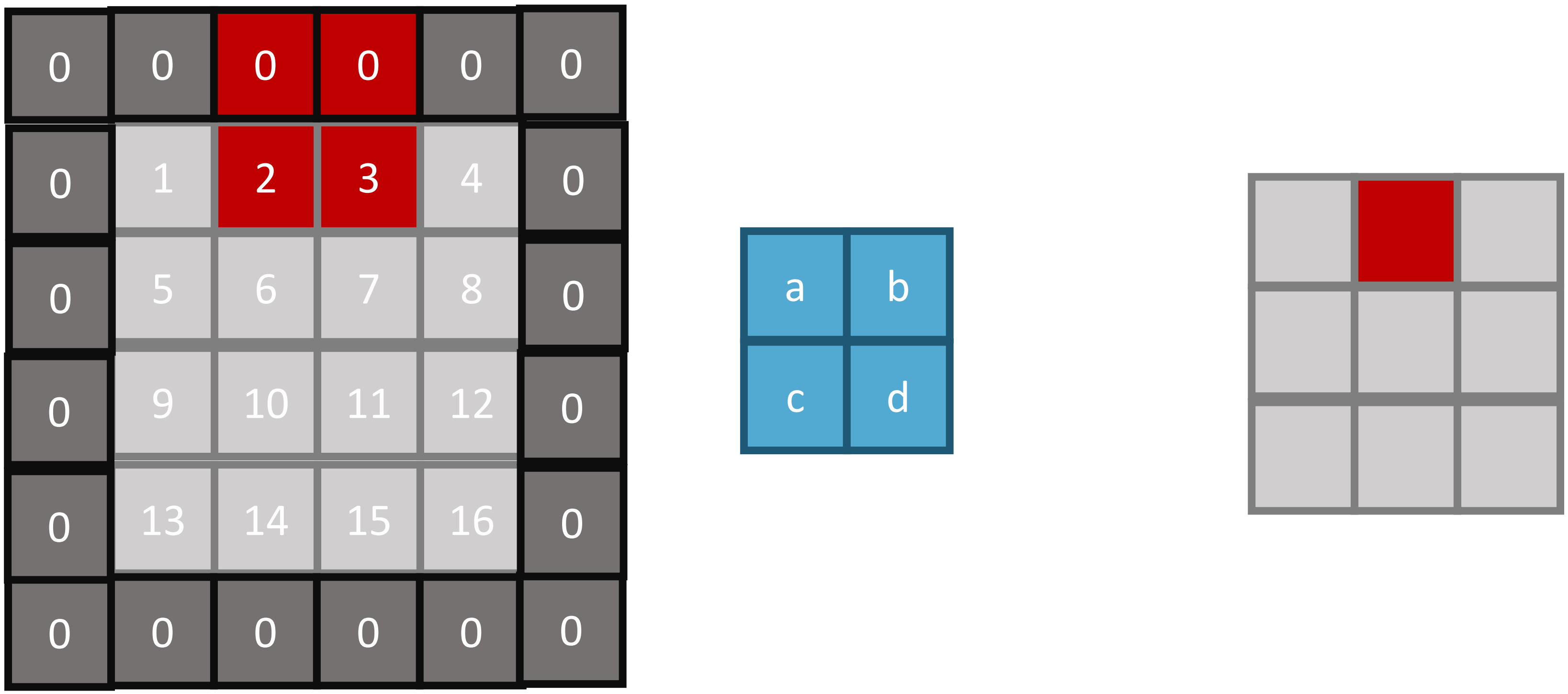
Padding equal to 1, useful to keep spatial dimension constant

$w = 4, k = 2, s = 2, \boldsymbol{p} = \mathbf{1}$

Output volume: $$\frac{w - k + 2p}{s} + 1$$

Kernel size, stride, and padding

Input 4×4 Kernel 2×2 Output 2×2



Padding equal to 1, useful to keep spatial dimension constant

$w = 4, k = 2, s = 2, \boldsymbol{p} = \mathbf{1}$

Output volume: $\dfrac{w - k + 2p}{s} + 1$

Complexity & Space

The larger the input features, and the number of output channels
→ More compute, **slower**
→ More intermediary activations to store, **heavier**

Pooling with a 2×2 kernel and stride 2

Input 4×4

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Average Pooling

3.5	5.5
11.5	13.5

Blur the edges

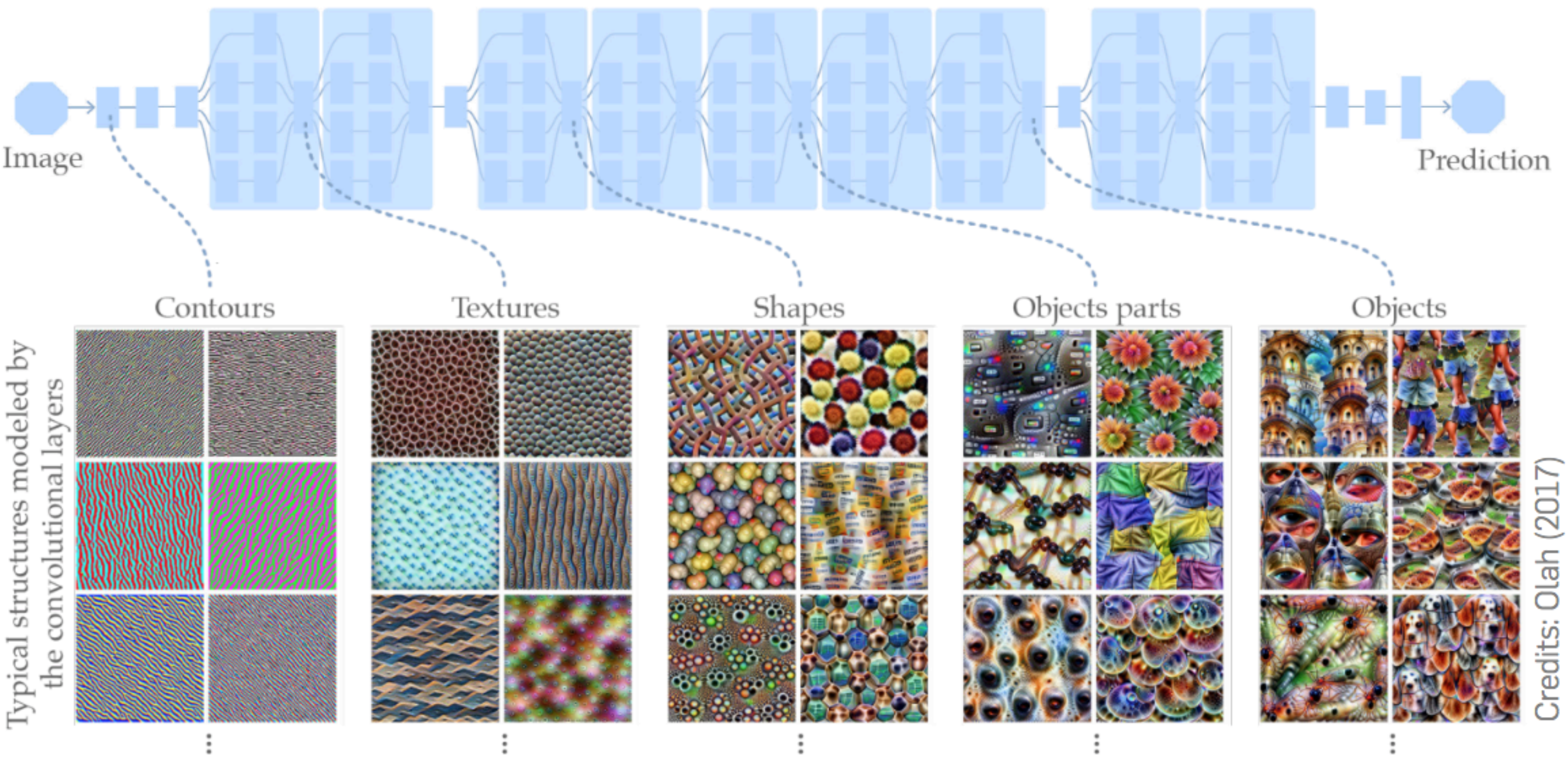
Max Pooling

6	8
14	16

Sharpen the edges

No learned parameters!

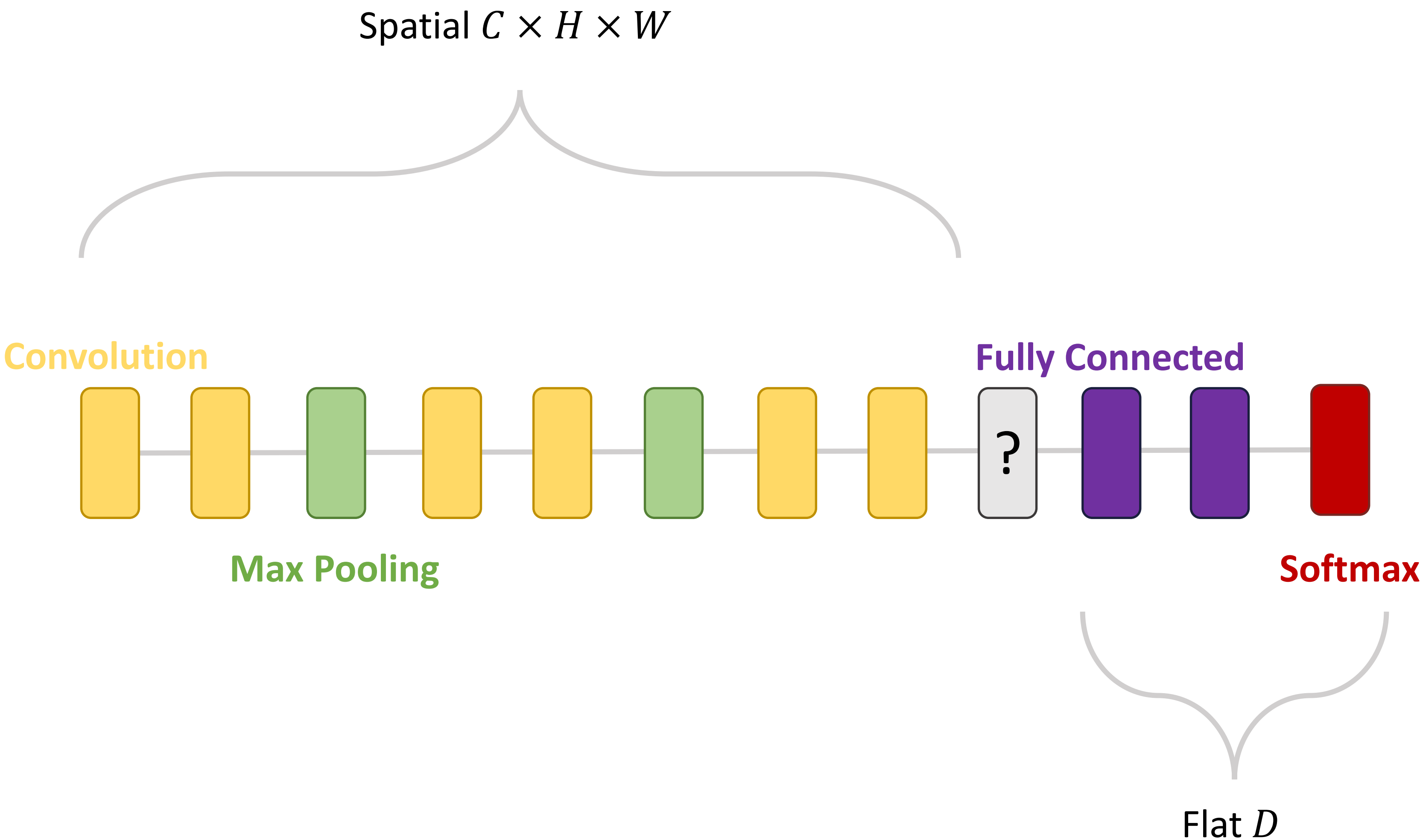
From crude to fine-grained patterns



Semantic structure a posteriori (not really usable)

CNN
Architectures

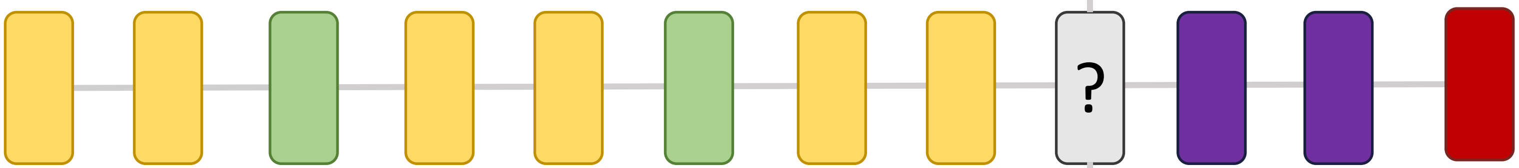
Basic CNN



Spatial $C \times H \times W$
Convolution
Fully Connected
?
Max Pooling
Softmax
Flat D

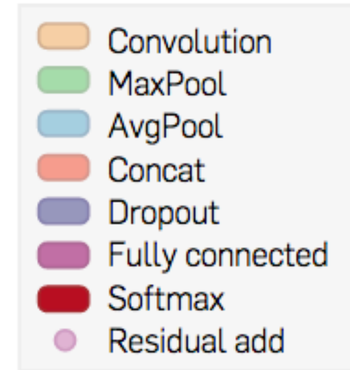
Basic CNN

Flatten: merge all dimensions into one
→ no loss of information
→ huge dimensionality
→ Dependent on the image size



Global Average Pooling: pooling with full kernel size
→ lose a lot of information
→ Dimensionality of the number of channels
→ No dependent on the image size

First ConvNet

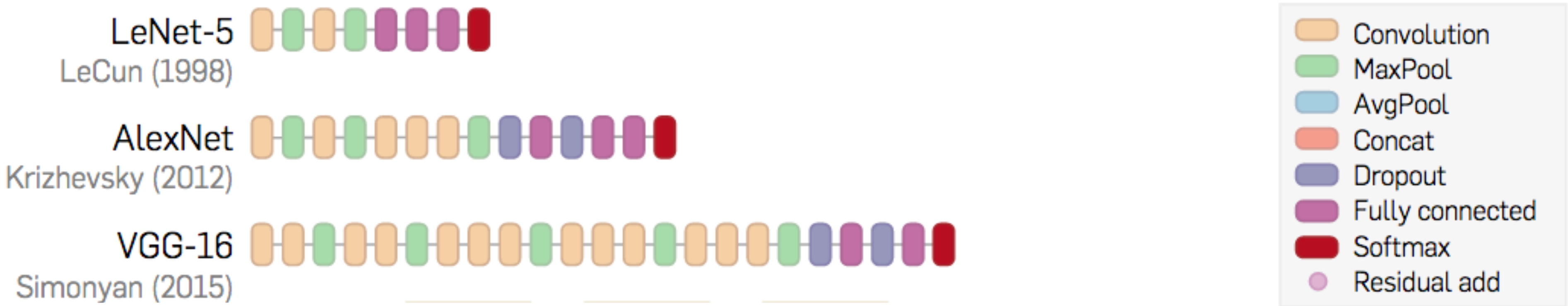


ConvNet that initiated the Deep Learning revolution

- Trained on ImageNet
- One of the first to use GPUs
- Model parallelism on 2 GPUs

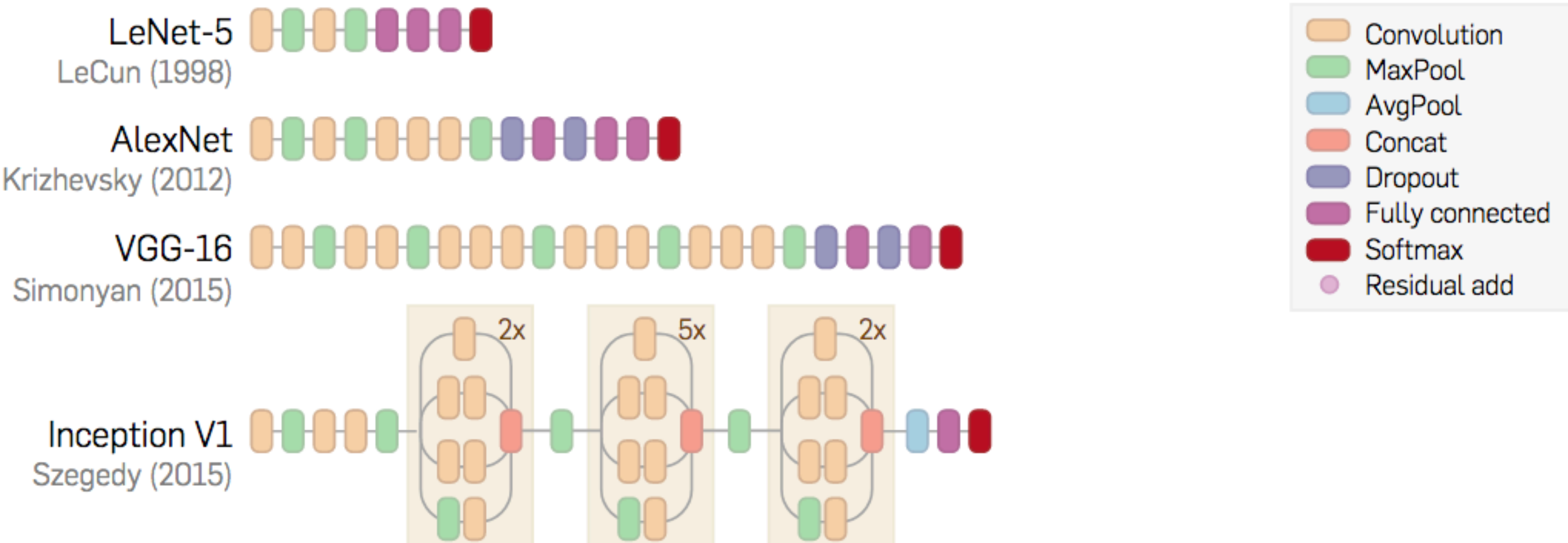
[Thomas Robert's Thesis]

Big ConvNet

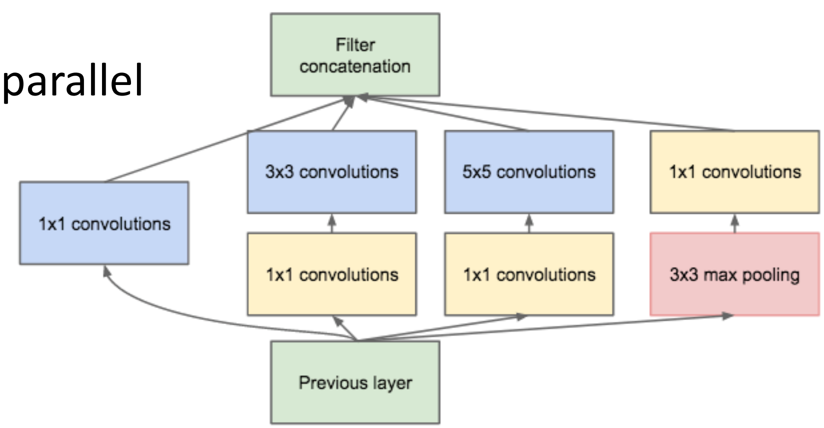


- Super large (134M parameters), mainly because of the flatten + fully-connected layers
- Similar to AlexNet in bigger
- Large kernel sizes (7x7 and 5x5)

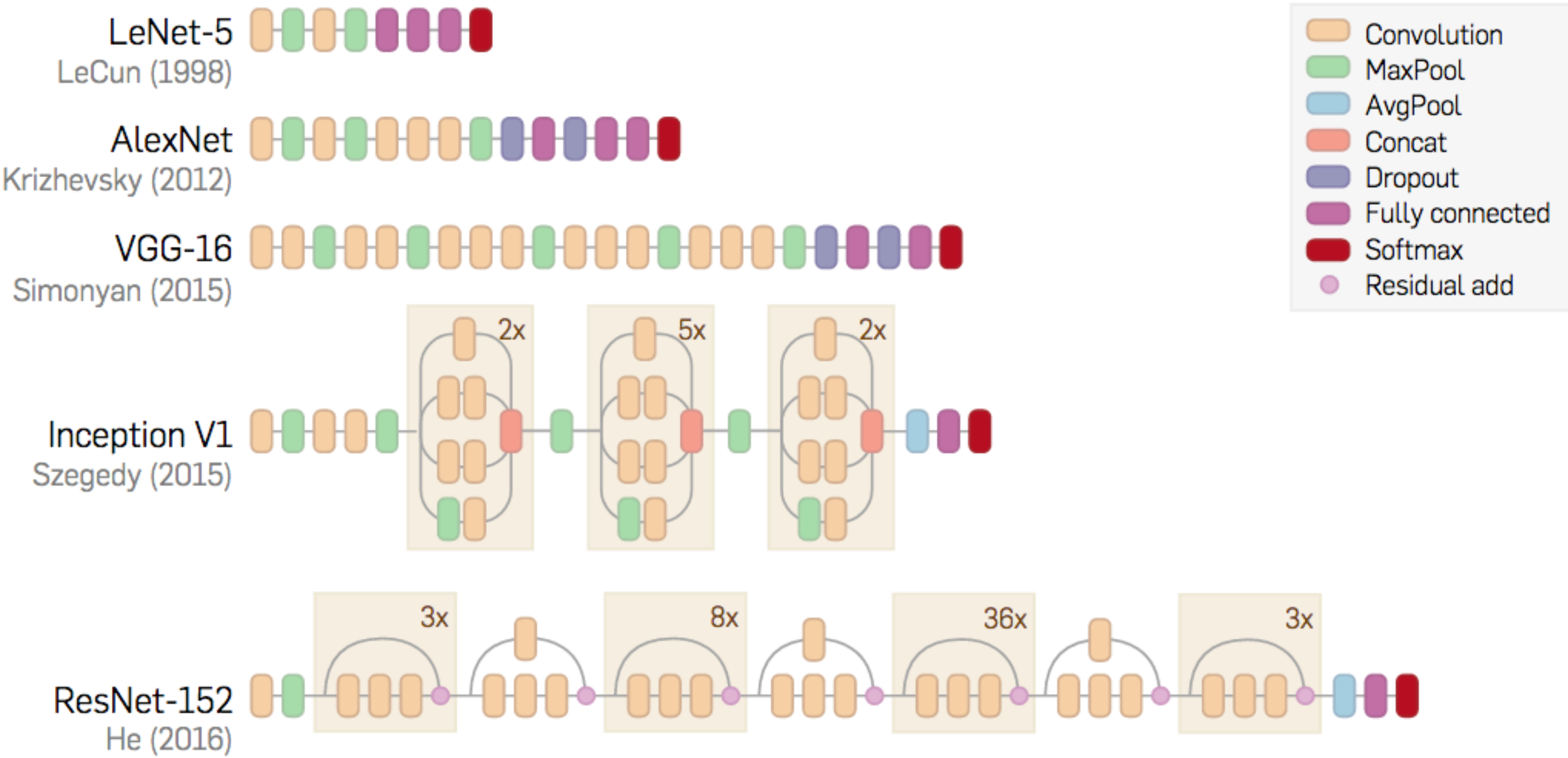
Multiple towers in parallel



- Convolutions with different kernel sizes in parallel
  - Some relation to neuroscience
  - Multi-scale view



Residual-based network



- **Super strong architecture, still important today**

ResNet

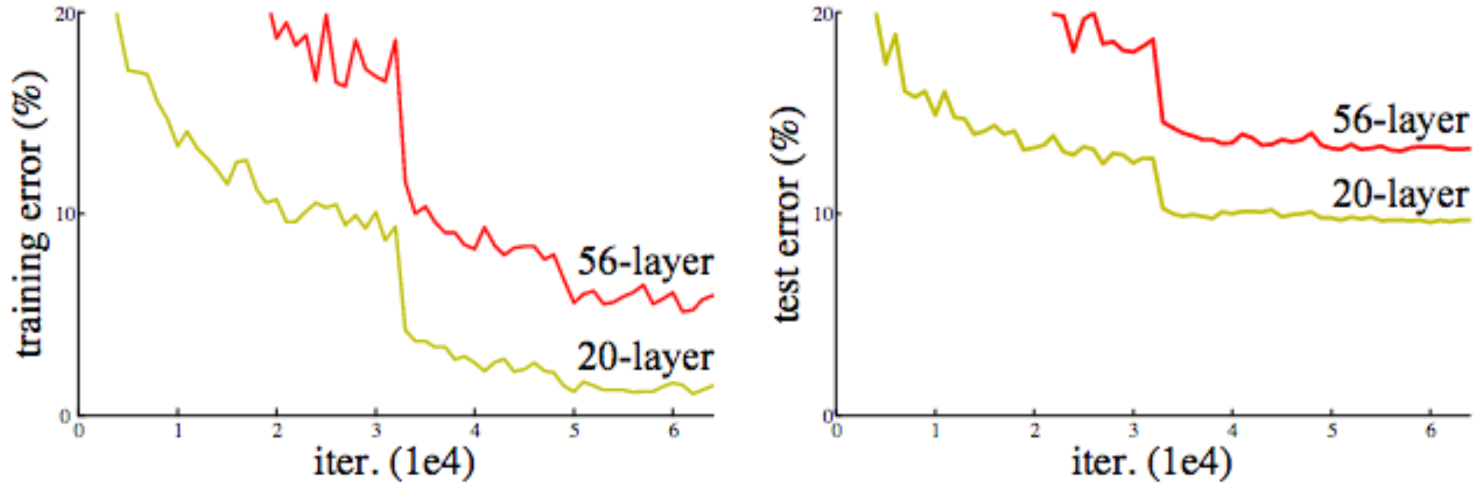
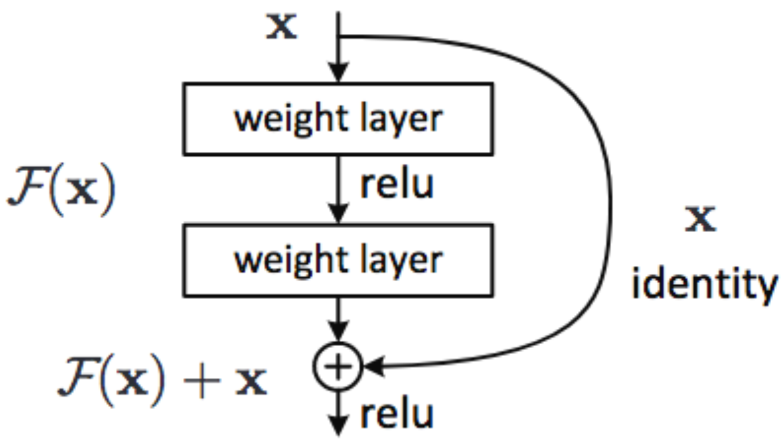


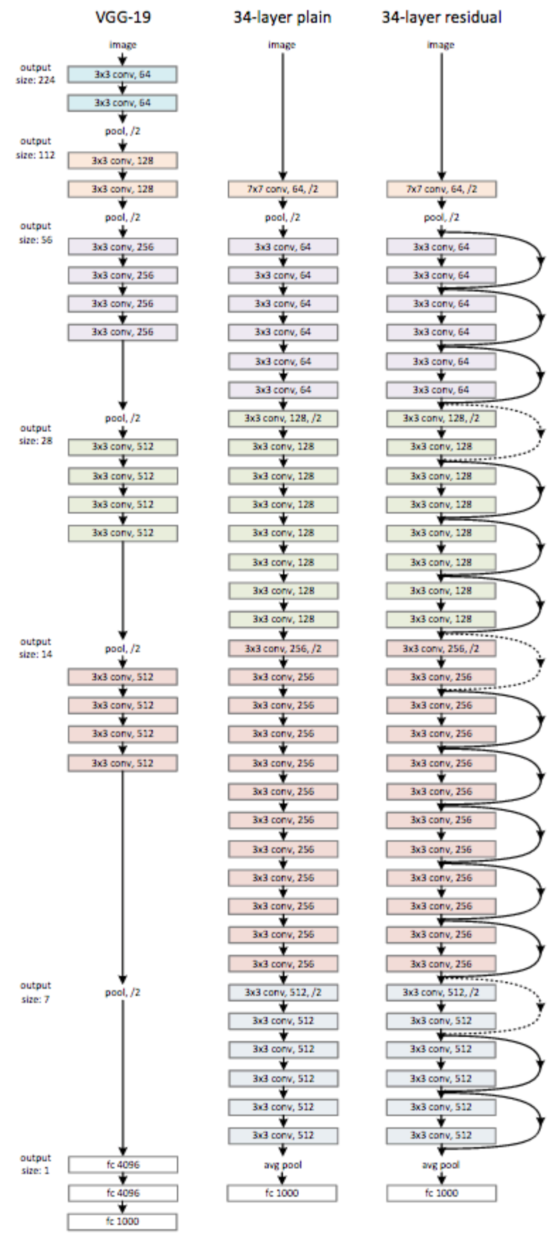
Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

Hard to train very deep network
→Gradient struggles to each earlier layers

ResNet (2015)



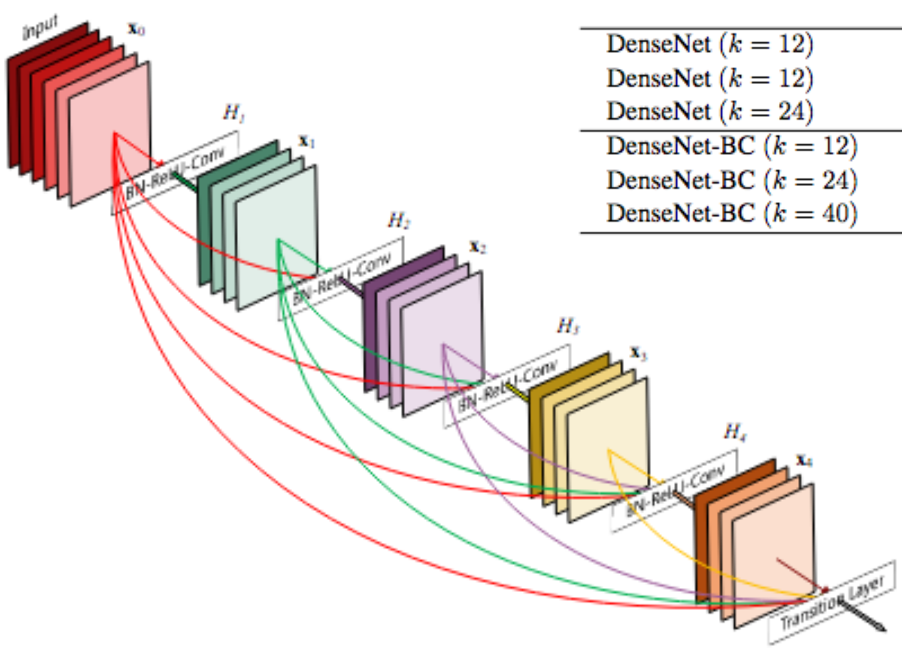
x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu



VGG-19
34-layer plain
34-layer residual

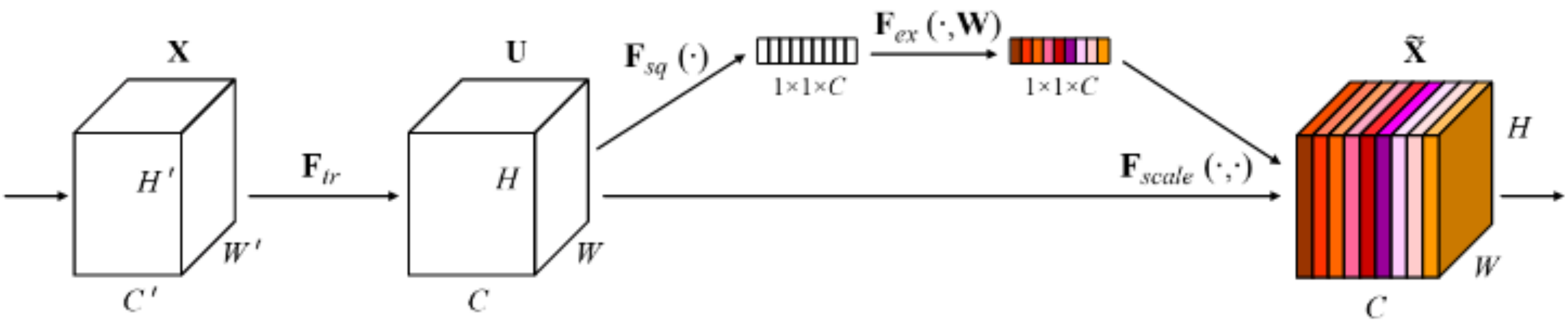
DenseNet (2016)

Method	Depth	Params	C10	C10+	C100	C100+	SVHN
Network in Network [22]	-	-	10.41	8.81	35.68	-	2.35
All-CNN [32]	-	-	9.08	7.25	-	33.71	-
Deeply Supervised Net [20]	-	-	9.69	7.97	-	34.57	1.92
Highway Network [34]	-	-	-	7.72	-	32.39	-
FractalNet [17]	21	38.6M	10.18	5.22	35.34	23.30	2.01
with Dropout/Drop-path	21	38.6M	7.33	4.60	28.20	23.73	1.87
ResNet [11]	110	1.7M	-	6.61	-	-	-
ResNet (reported by [13])	110	1.7M	13.63	6.41	44.74	27.22	2.01
ResNet with Stochastic Depth [13]	110	1.7M	11.66	5.23	37.80	24.58	1.75
	1202	10.2M	-	4.91	-	-	-
Wide ResNet [42]	16	11.0M	-	4.81	-	22.07	-
	28	36.5M	-	4.17	-	20.50	-
with Dropout	16	2.7M	-	-	-	-	1.64
ResNet (pre-activation) [12]	164	1.7M	11.26*	5.46	35.58*	24.33	-
	1001	10.2M	10.56*	4.62	33.47*	22.71	-
DenseNet ($k = 12$)	40	1.0M	**7.00**	5.24	**27.55**	24.42	1.79
DenseNet ($k = 12$)	100	7.0M	**5.77**	**4.10**	**23.79**	**20.20**	1.67
DenseNet ($k = 24$)	100	27.2M	**5.83**	**3.74**	**23.42**	**19.25**	**1.59**
DenseNet-BC ($k = 12$)	100	0.8M	**5.92**	4.51	**24.15**	22.27	1.76
DenseNet-BC ($k = 24$)	250	15.3M	**5.19**	**3.62**	**19.64**	**17.60**	1.74
DenseNet-BC ($k = 40$)	190	25.6M	-	**3.46**	-	**17.18**	-



More residuals!

[Huang et al. CVPR 2017]

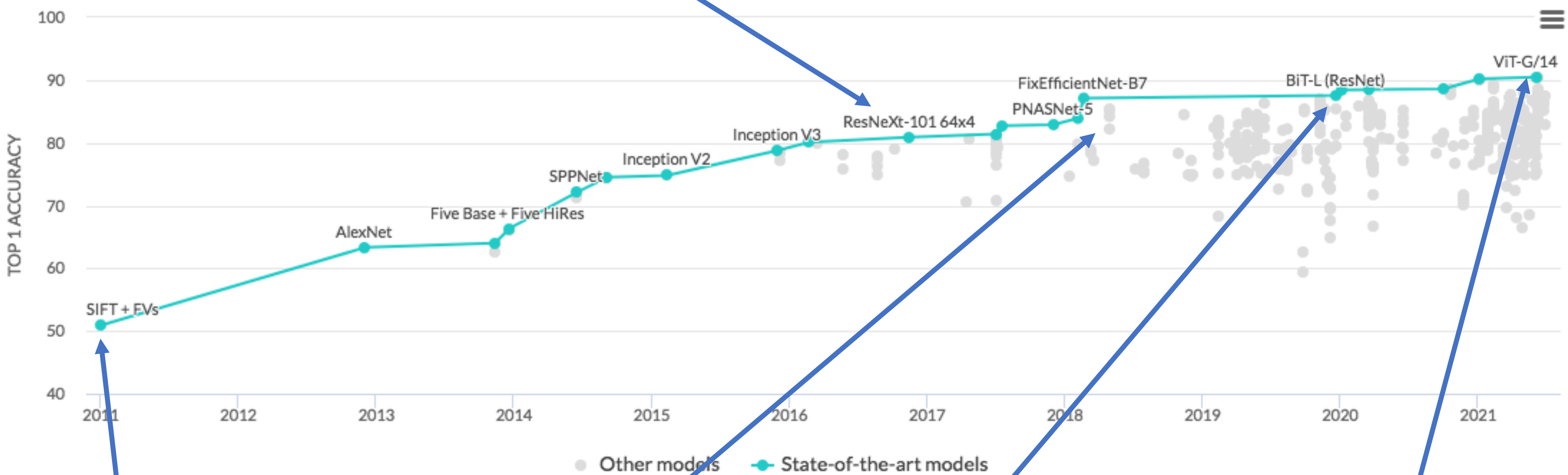


Attention per channels

[Hu et al. CVPR 2018]

State-of-the-Art

ResNet+Inception



Without deep

ResNet-based + training on larger image resolution

ResNet-based + data augmentation

NOT a CNN! Covered in a later lecture

PS: Most of the State-of-the-Art papers now use external data

[Paperswithcode.com](https://paperswithcode.com)

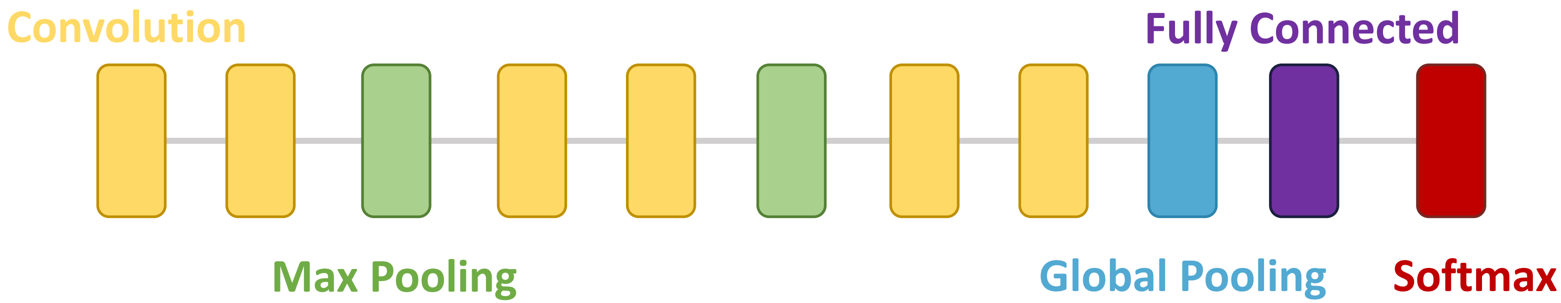
Transfer Learning

Data

Need a lot of data to train a modern CNN.

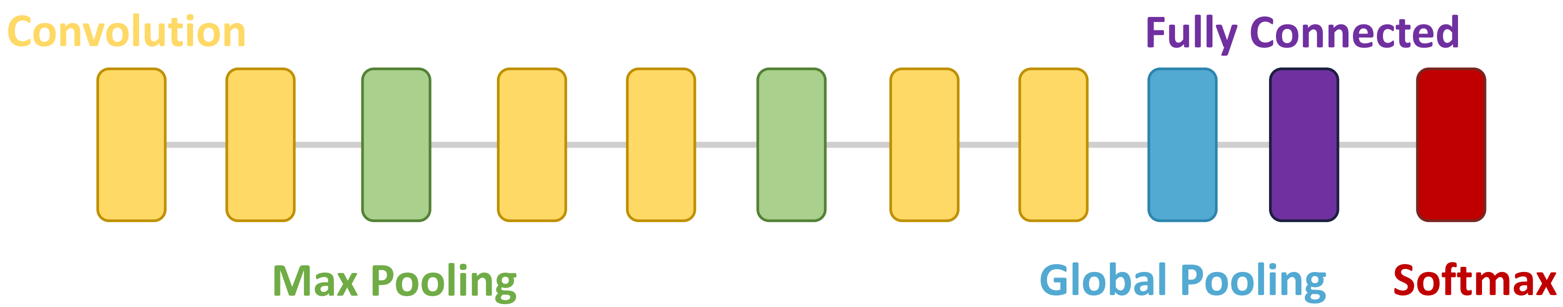
But what to do when dataset is not big enough?

Transfer Learning



1. Train model on ImageNet with 1000 classes
2. Remove last fully connected layer
3. Add new fully connected with the number of classes of the target dataseet
4. Fine-tune model

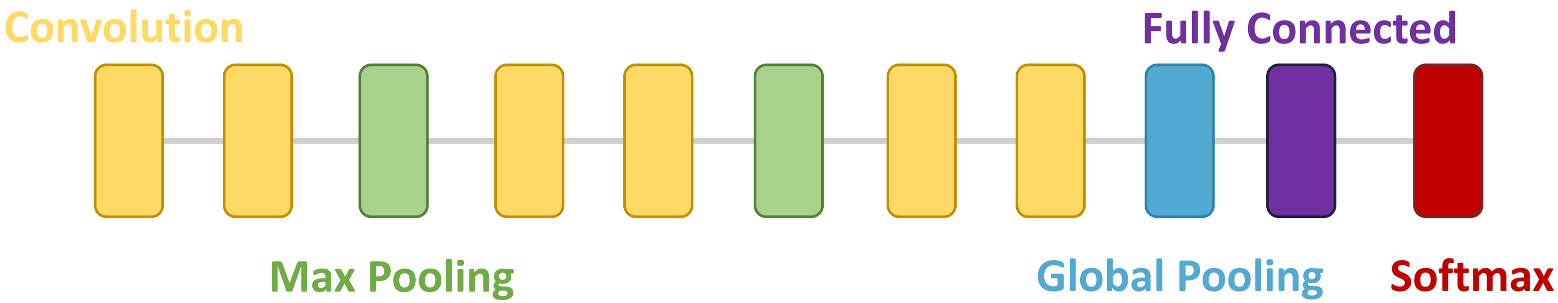
Transfer Learning



We can finetune only the new FC layer, or also the whole ConvNet.

Usually starts with a learning rate 10x lower.

Transfer Learning

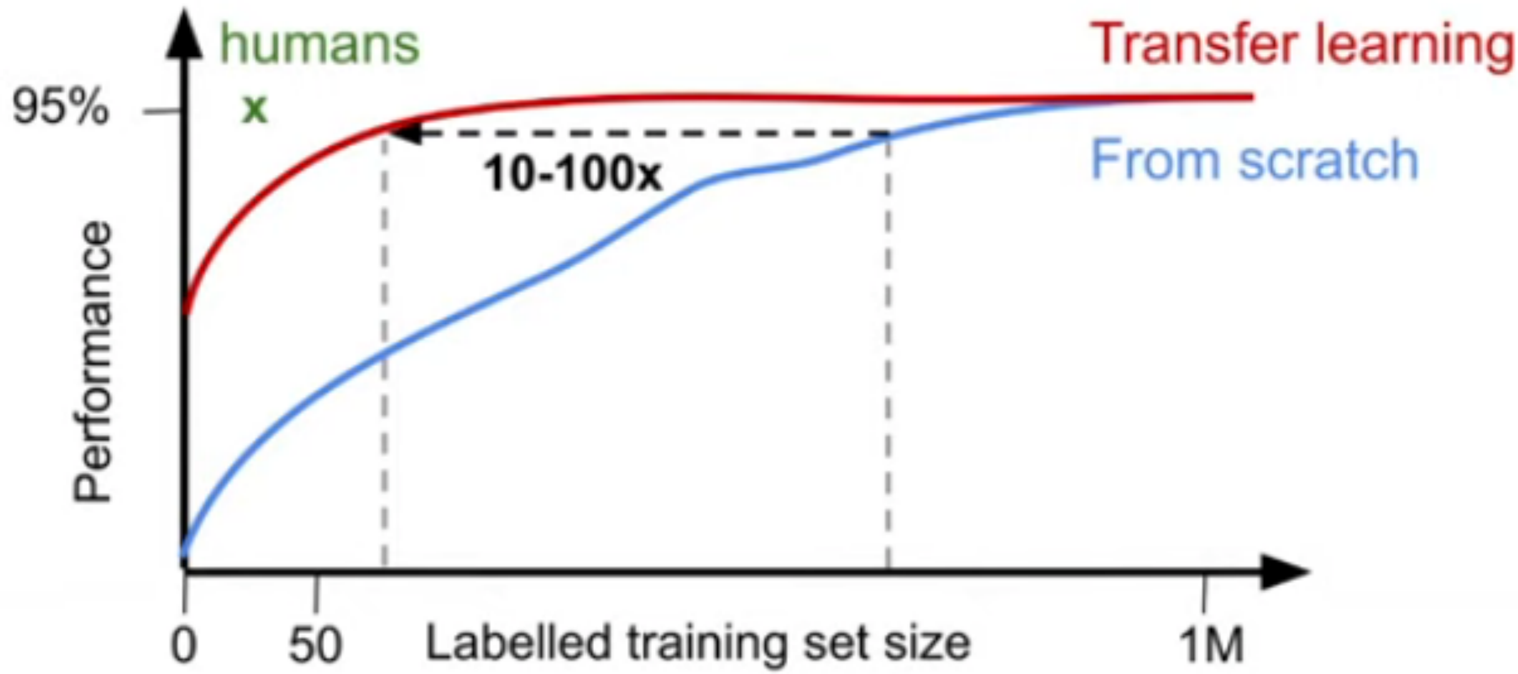


Transfer works better if the source domain is close to the target domain.

Learning cat vs dog after imagenet: easy

Learning to spot cancers on radiography after imagenet: harder

Transfer Learning

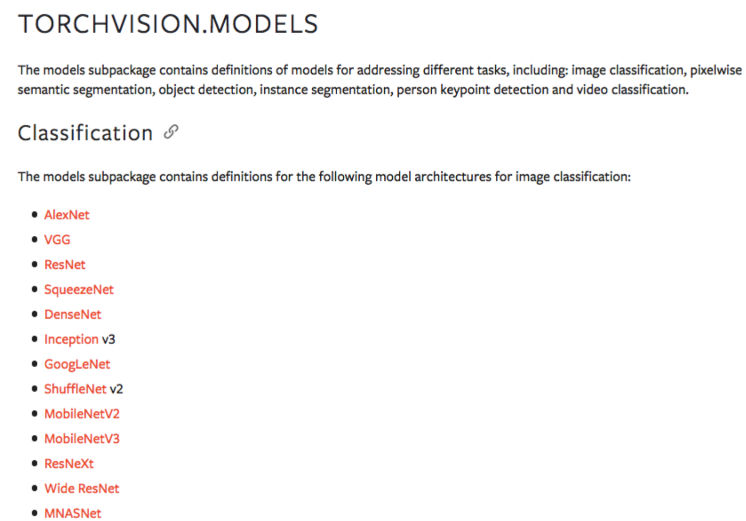


Graph from Neil Houlsby

Pretrained models

Plenty of pretrained models in PyTorch on

Torchvision zoo:


TORCHVISION.MODELS

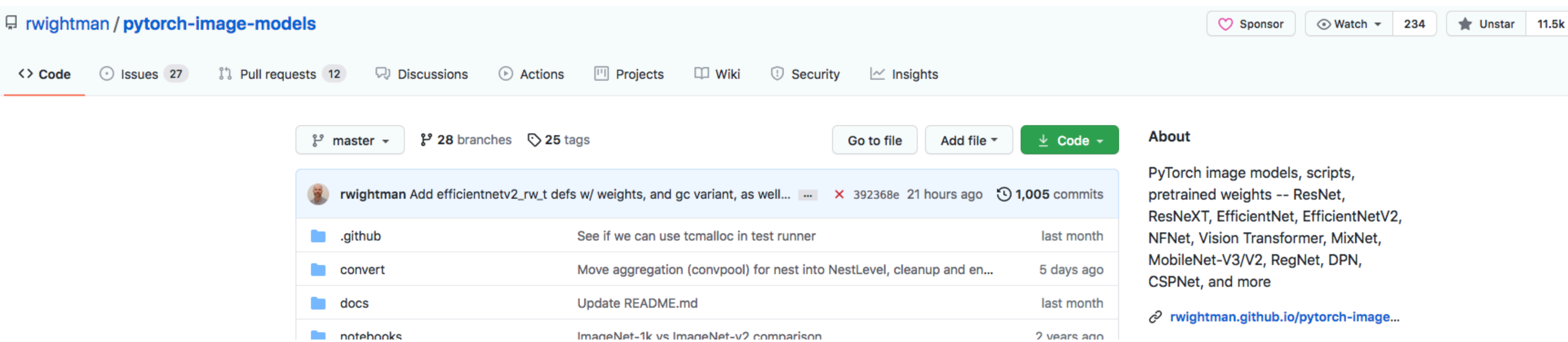
The models subpackage contains definitions of models for addressing different tasks, including: image classification, pixelwise semantic segmentation, object detection, instance segmentation, person keypoint detection and video classification.

Classification

The models subpackage contains definitions for the following model architectures for image classification:

- AlexNet
- VGG
- ResNet
- SqueezeNet
- DenseNet
- Inception v3
- GoogLeNet
- ShuffleNet v2
- MobileNetV2
- MobileNetV3
- ResNeXt
- Wide ResNet
- MNASNet

Timm library:


rwightman / pytorch-image-models

Sponsor | Watch 234 | Unstar 11.5k

Code | Issues 27 | Pull requests 12 | Discussions | Actions | Projects | Wiki | Security | Insights

master | 28 branches | 25 tags | Go to file | Add file | Code

rwightman Add efficientnetv2_rw_t defs w/ weights, and gc variant, as well... 392368e 21 hours ago 1,005 commits

.github	See if we can use tcmalloc in test runner	last month
convert	Move aggregation (convpool) for nest into NestLevel, cleanup and en...	5 days ago
docs	Update README.md	last month
notebooks	ImageNet-1k vs ImageNet-v2 comparison	2 years ago

About

PyTorch image models, scripts, pretrained weights -- ResNet, ResNeXT, EfficientNet, EfficientNetV2, NFNet, Vision Transformer, MixNet, MobileNet-V3/V2, RegNet, DPN, CSPNet, and more

rwightman.github.io/pytorch-image...

Tricks

Data Augmentation

Combine multiple image alterations to produce a “new” image.

These augmentations are done on-the-fly with different randomness each time.

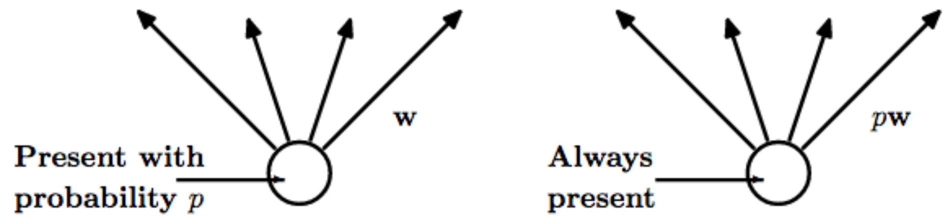
Results:
- Increase artificially a small dataset, less overfit!
- Make the model more robust to image corruption

Dropout

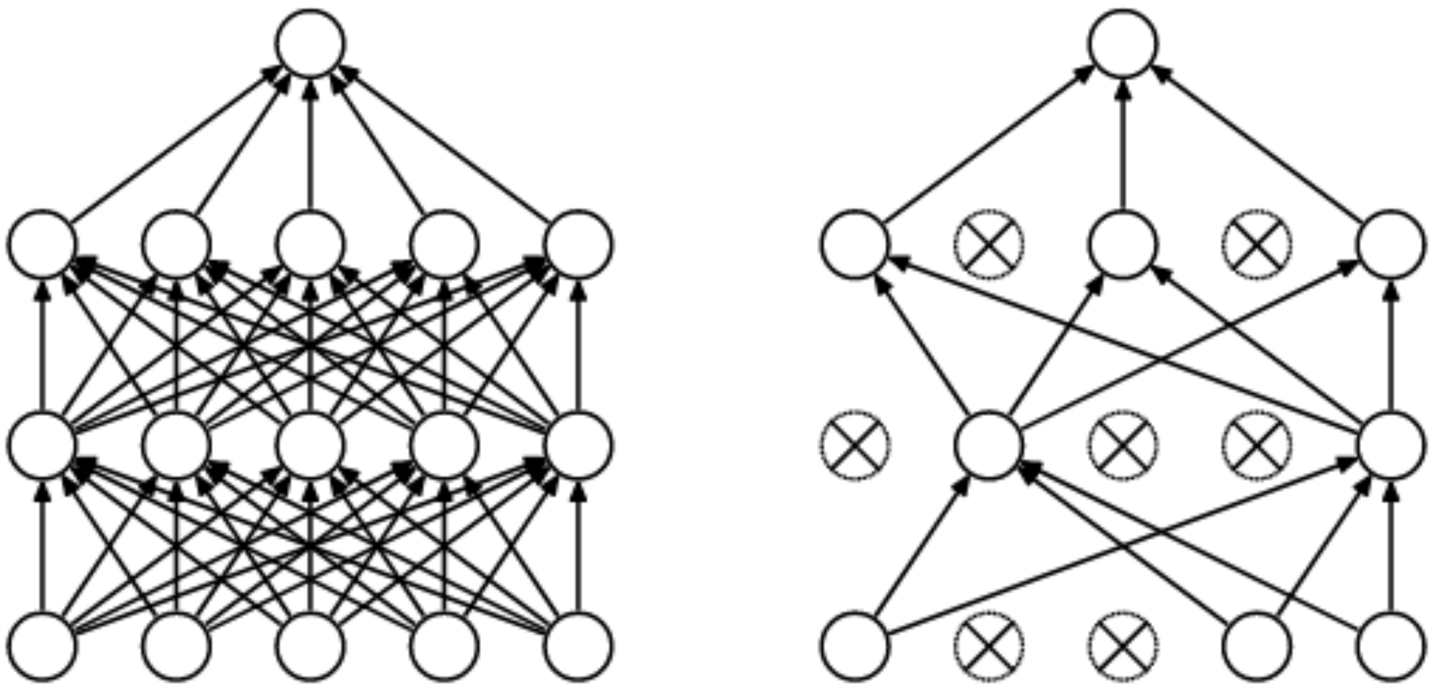
Randomly drop unit during a forward pass.

Drastically reduce overfitting:
- Sort of ensemble of networks
- Force all units to contribute

Usually only for fully connected layers.



(a) At training time (b) At test time

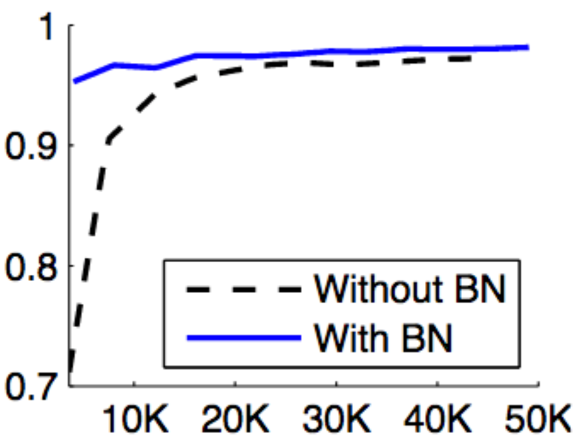

(a) Standard Neural Net (b) After applying dropout.

[Sritastava et al. JMLR 2014]

Batch Normalization

Normalize intermediary features.

During training with batch Statistics. During testing with running mean and std.



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;
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.

[Ioffe and Szegedy ICML 2015]

Small break,
then coding session!