Lecture 6: Convolutional Neural Networks (CNN) Part 1

Olexandr Isayev

Department of Chemistry, CMU

olexandr@cmu.edu

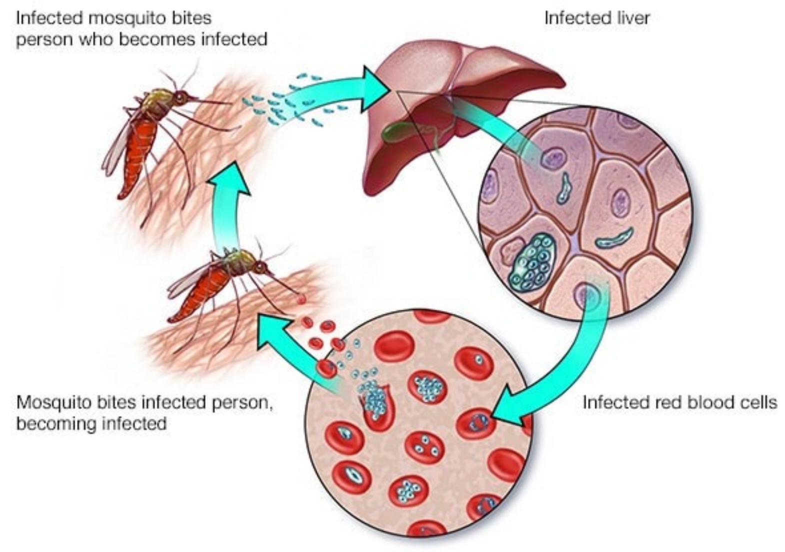
In Class Projects

Please select a problem and dataset

- Upload title and one paragraph summary to canvas.
- I encourage you to use your **domain-specific dataset**
- If not, I am happy to give you one. Just send email or see after class.
- Deadline: March 3

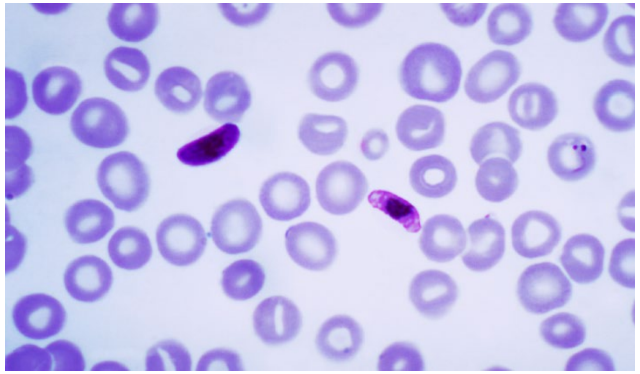
HW2: Fighting Malaria with CNNs!

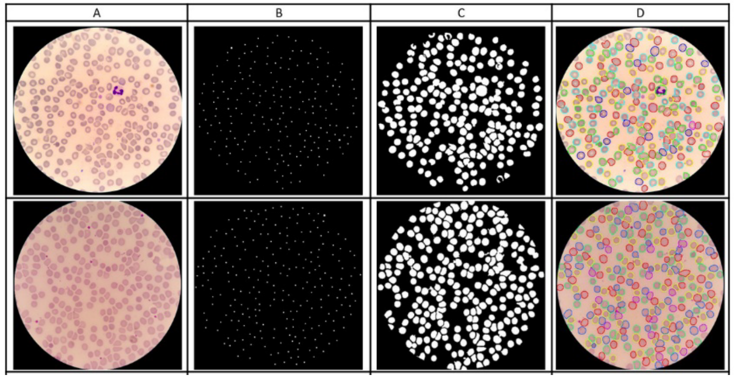
Malaria, a disease caused by protozoan parasites of the genus Plasmodium, is not only an acute life threat in many developing countries but also a significant burden on the healthcare system worldwide.



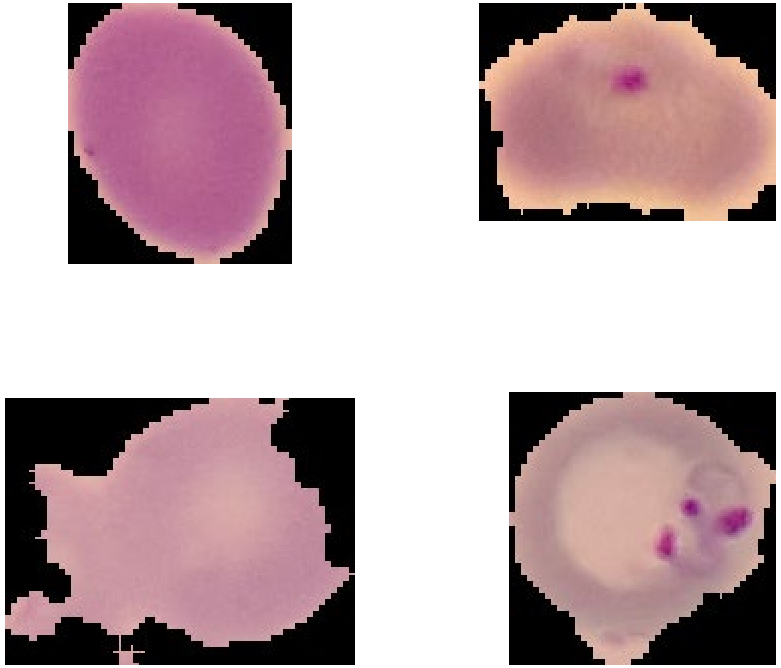
In this homework, you are required to <u>implement a convolution neural network-based machine learning model</u> to predict whether the cell in the given picture is parasitized by the genus Plasmodium or not.

Training and Test Data





Initial blood sample data processing



~25,000 images

Your Mission

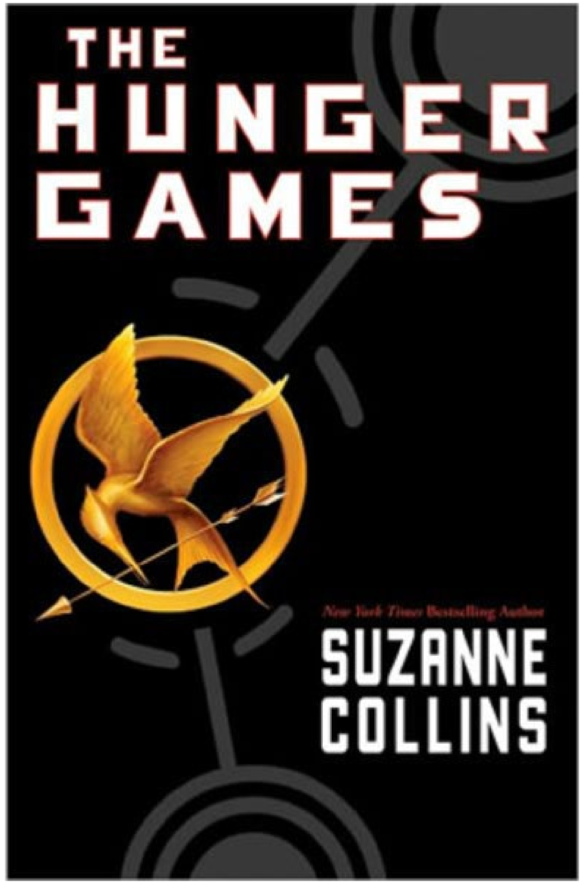
Kaggle competition

Get data from kaggle.com
Solve the problem
Submit solution for autograding to Kaggle
Submit IPYNB file to canvas

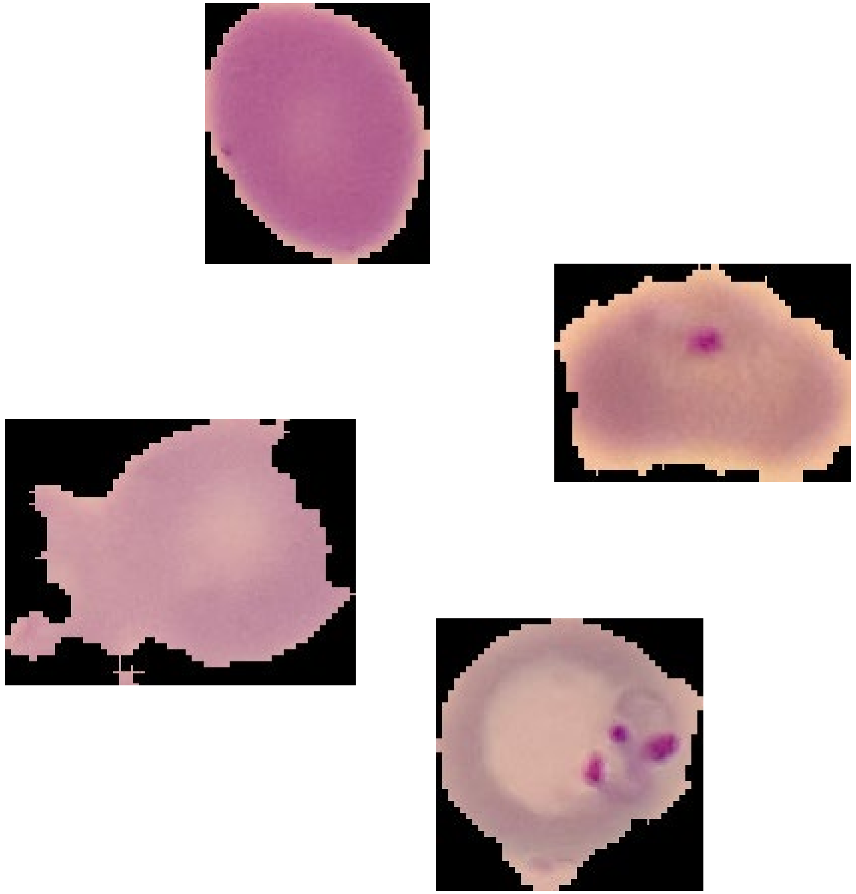
Deadline: **March 11**

Sign-up Link:

https://www.kaggle.com/t/b7af9038607d45afadb8d22bbdfc0c5d



Data



X

0
1
0
1
0
0
1
1
1
0
0
1
1
.
.
.
.

y

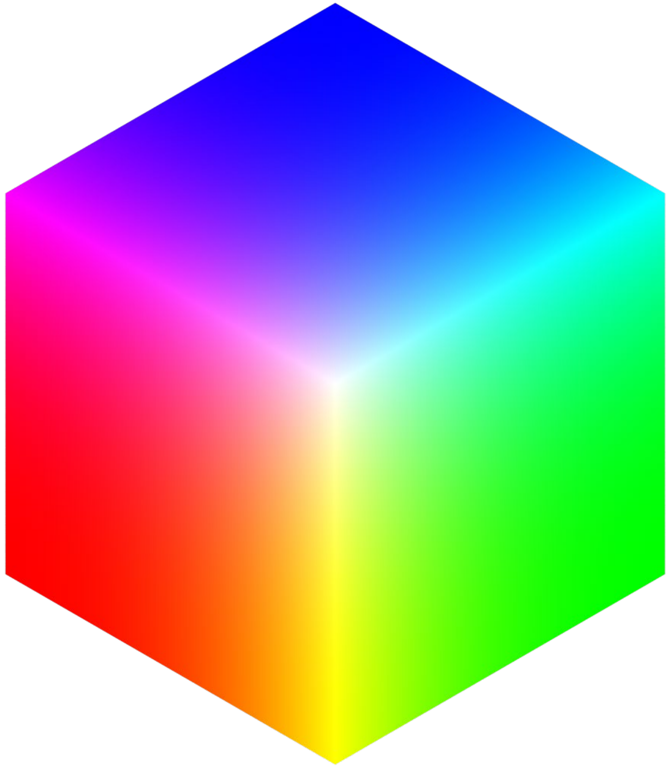
What you need to do:

- Given the training data: **train_data.zip** and **train_labels.csv**
- You are allowed to build any type of CNN classification model using PyTorch!
- You are allowed to use any type of data processing (data augmentation)
- Please explore training from scratch or finetuning/pertaining existing model .
- Find the best performing model
- Use your model to score **test_data.zip**

- In this work you will predict the binarized disease state for classification between sick (class 1) and healthy (class 0) blood samples.
- Performance will be measured using accuracy score (sklearn.metrics.accuracy_score)
- Overfitting is prevented by using public and private leaderboards (50%-50%).

Kaggle demo

Standard RGB image



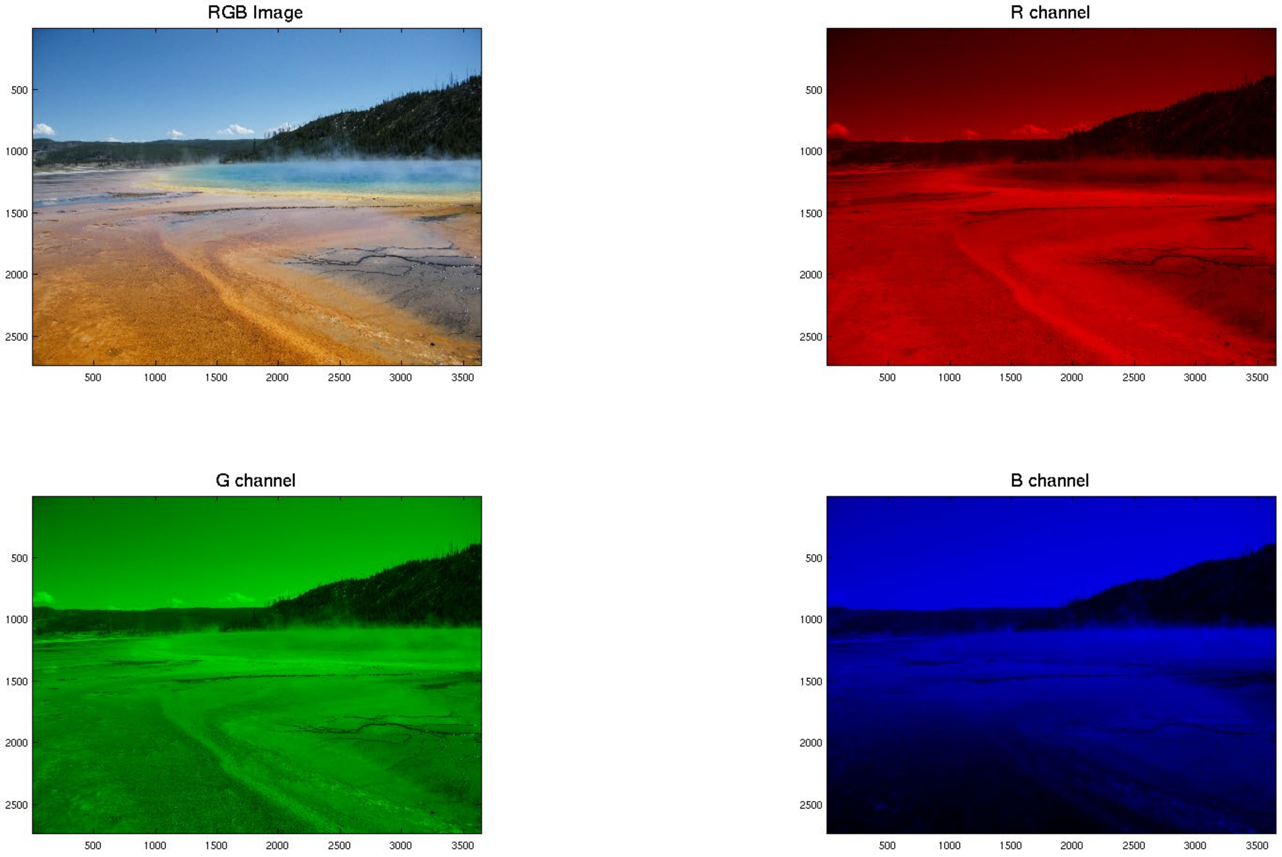
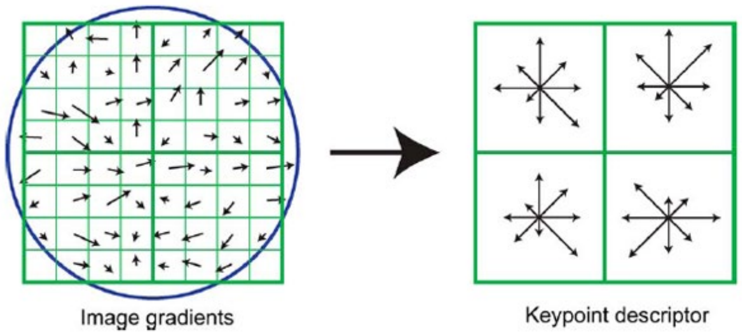
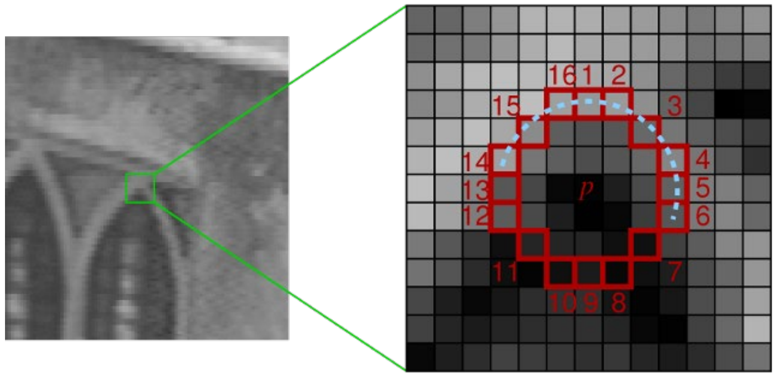


Image features

- We’ve been basically talking about detecting features in images, in a very naïve way.
- Researchers built multiple computer vision techniques to deal with these issues: SIFT, FAST, SURF, BRIEF, etc.
- However, similar problems arose: the detectors where either too general or too over-engineered. Humans were designing these feature detectors, and that made them either too simple or hard to generalize.

SIFT feature descriptor





FAST corner detection algorithm

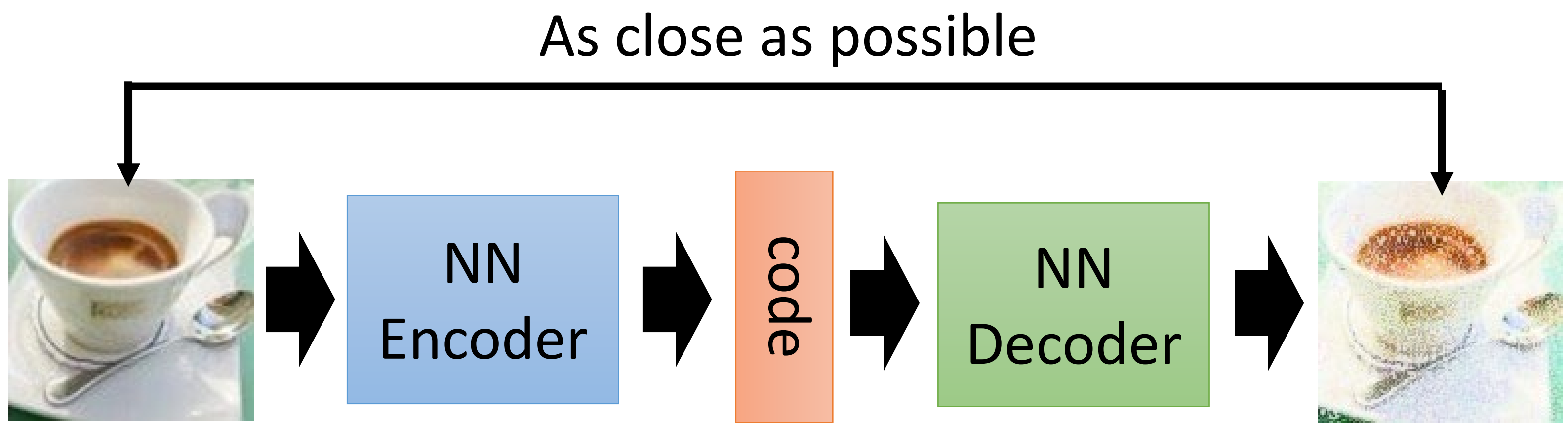
Image features (cont)

- What if we learned the features to detect?
- We need a system that can do Representation Learning (or Feature Learning).

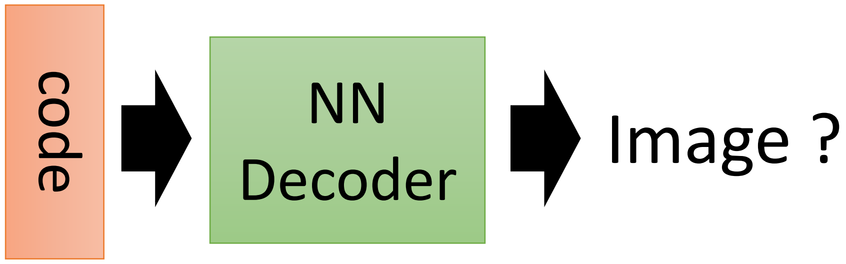
Representation Learning: technique that allows a system to automatically find relevant features for a given task. Replaces manual feature engineering.

- Multiple techniques for this:
  - Unsupervised (K-means, PCA, …).
  - Supervised (Sup. Dictionary learning, Neural Networks!)

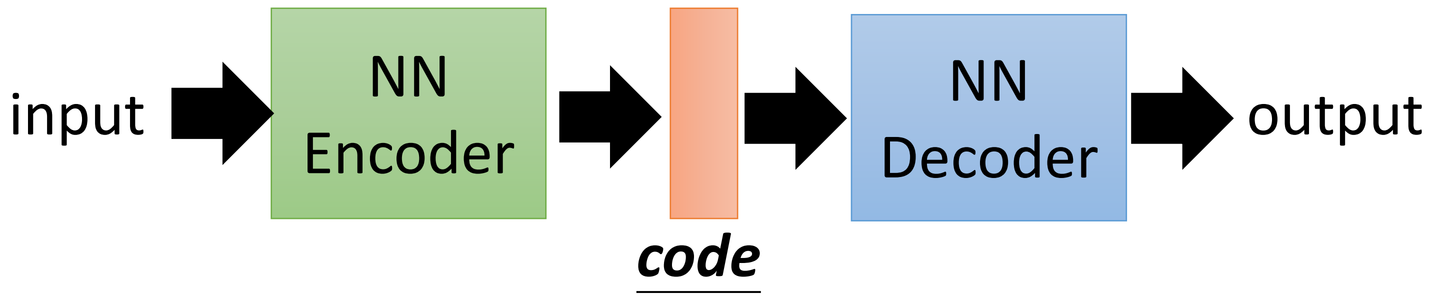
Auto-encoder



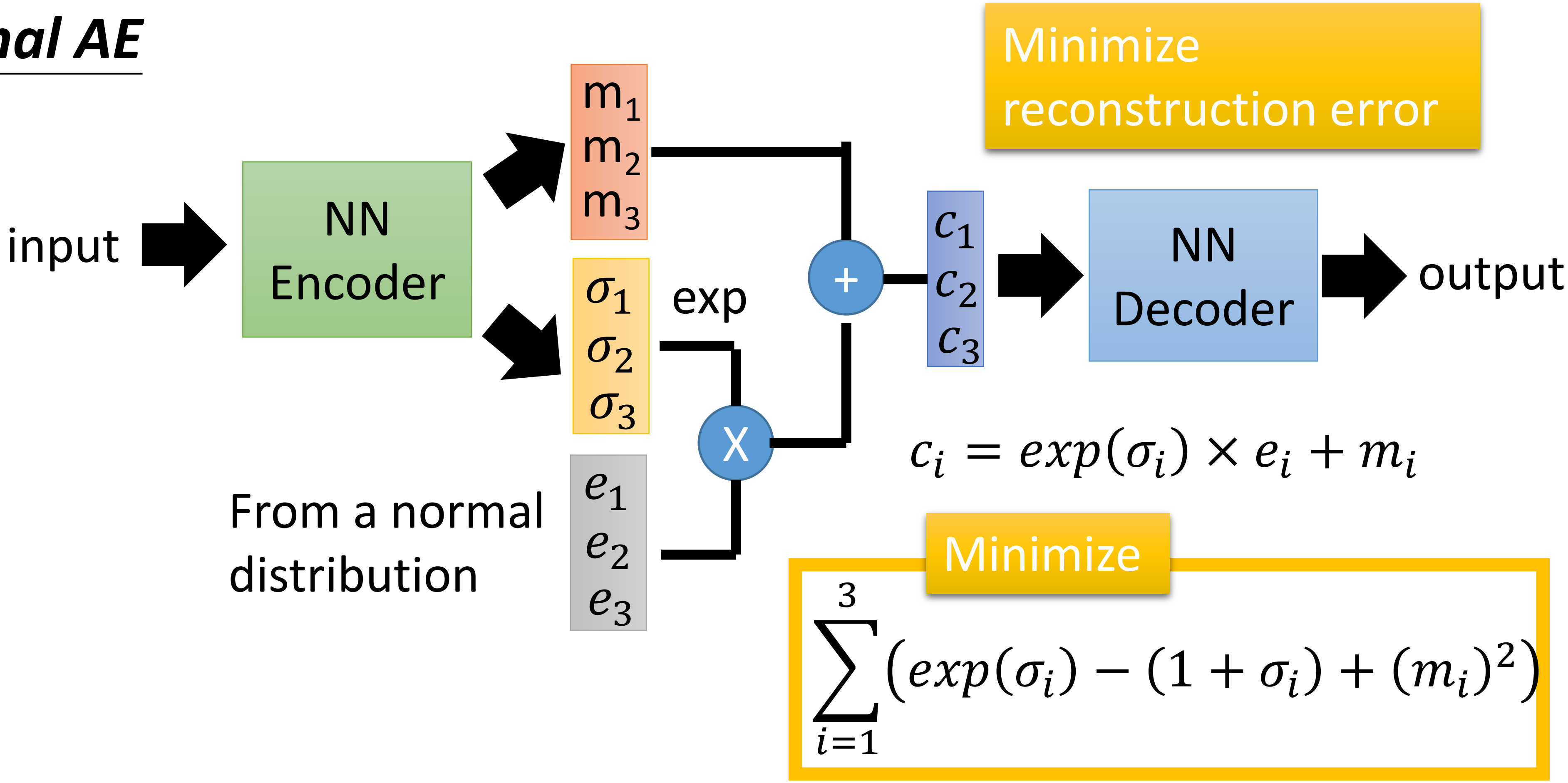
Randomly generate
a vector as code



Auto-encoder

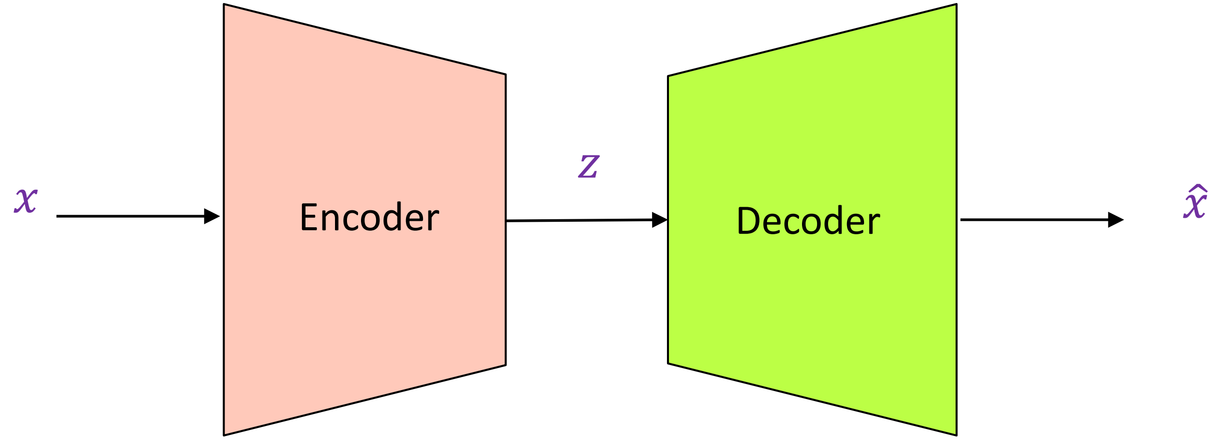


Variational AE



Variational autoencoders: Overview

- Probabilistic formulation based on *variational Bayes* framework
- At training time, jointly learn *encoder* and *decoder* by maximizing (a bound on) the data likelihood
- At test time, discard encoder and use decoder to sample from the learned distribution



D. Kingma and M. Welling, [Auto-Encoding Variational Bayes](), ICLR 2014

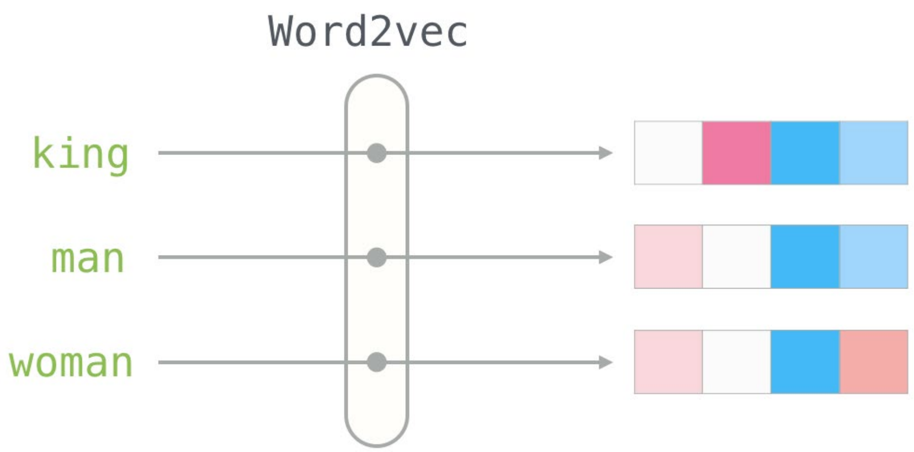
Embeddings

Neural network embeddings have 3 primary purposes:

- Finding nearest neighbors in the embedding space. These can be used to make recommendations based on user interests or cluster categories.
- As input to a machine learning model for a supervised task.
- For visualization of concepts and relations between categories.

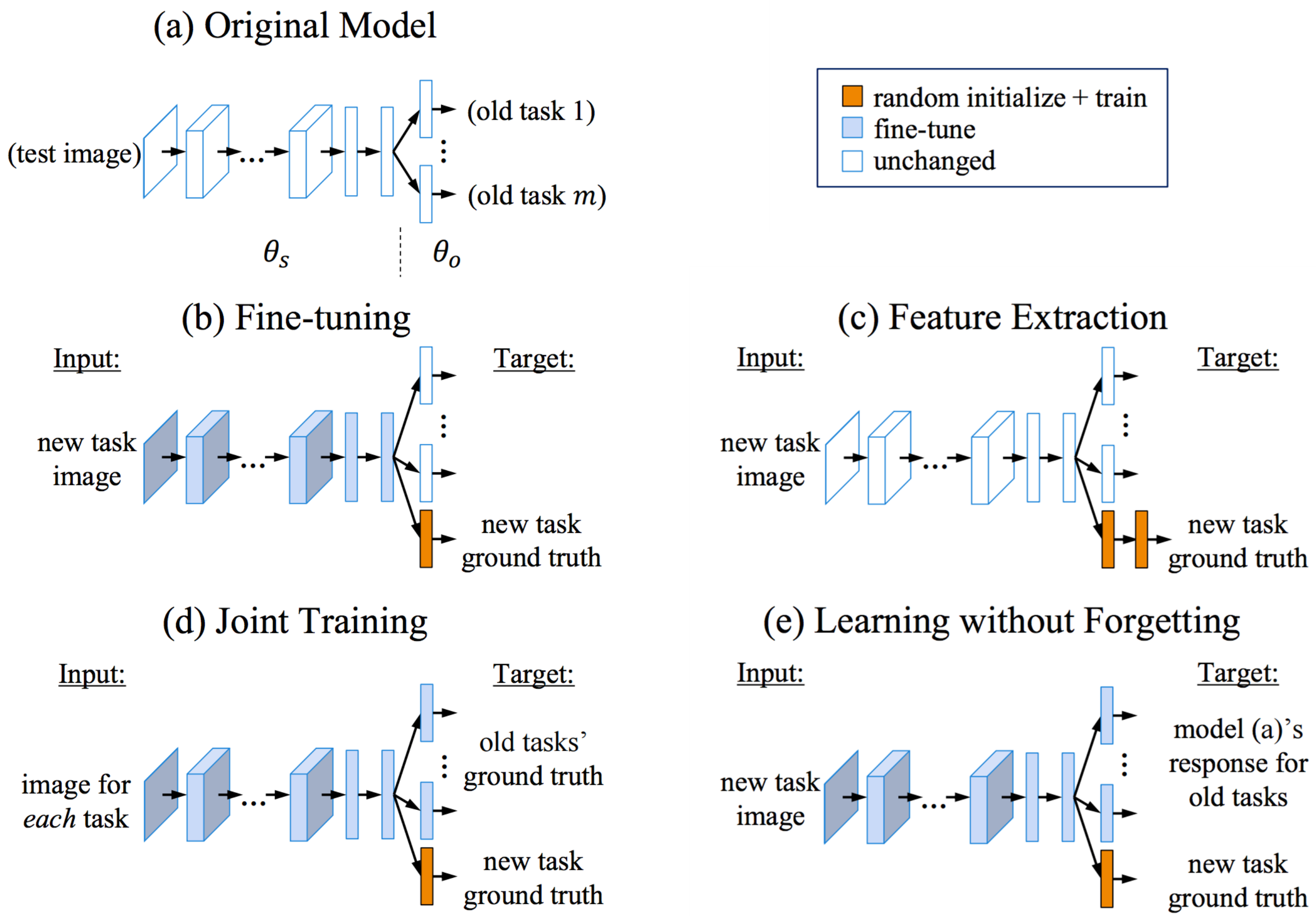
Word2Vec

Distributed vector representation for words



Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. "Distributed representations of words and phrases and their compositionality." In *Advances in neural information processing systems*, pp. 3111-3119. 2013.

Mixing Representations - LwF



Li & Hoiem ECCV'16

Li & Hoiem. 2016.

Main **drawbacks** of ANNs (MLPs)

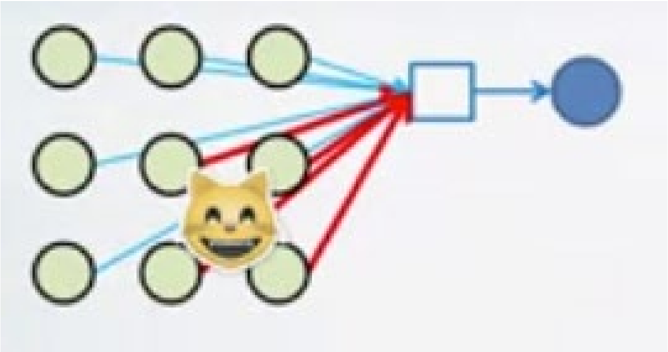
MLPs use one perceptron for each input (e.g. pixel in an image, multiplied by 3 in RGB case). The amount of weights rapidly becomes unmanageable for large images.

Training difficulties arise, overfitting can appear.

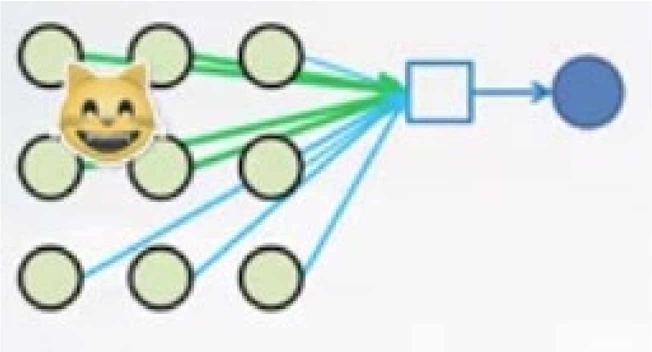
MLPs react differently to an input (images) and its shifted version – they are not translation invariant.

Drawbacks of ANNs and MLPs

Imagine we want to build a cat detector with an MLP.



In this case, the red weights will be modified to better recognize cats



In this case, the green weights will be modified.

We are learning redundant features. Approach is not robust, as cats could appear in yet another position.

Drawbacks

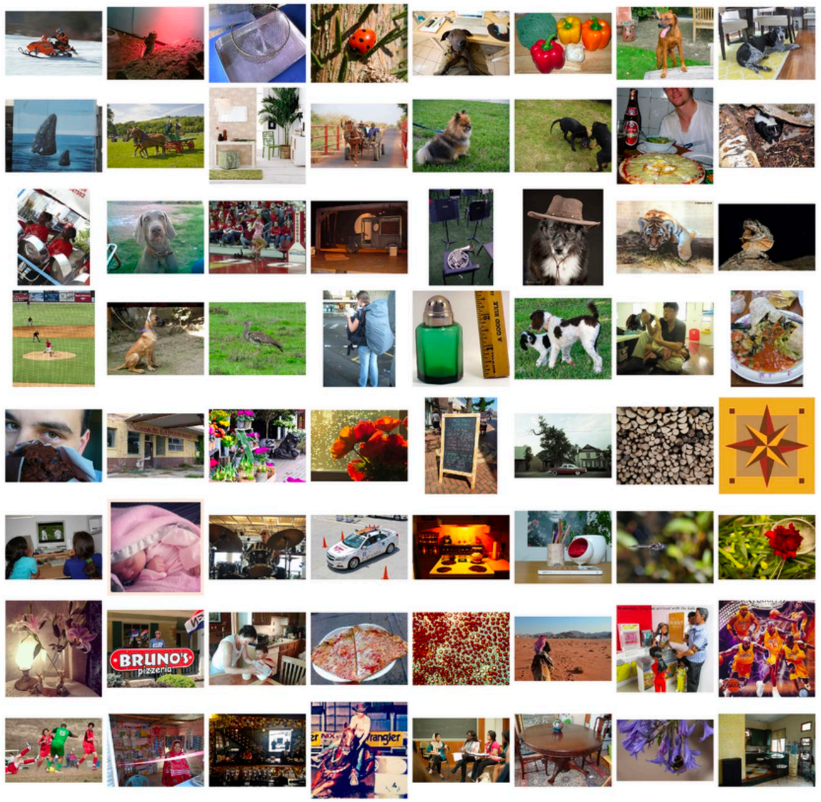
- Example: CIFAR10
- Simple 32x32 color images (3 channels)
- Each pixel is a feature: an MLP would have 32x32x3+1 = 3073 weights per neuron!

Drawbacks

- Example: ImageNet

- Images are usually 224x224x3: an MLP would have 150129 weights per neuron. If the first layer of the MLP is around 128 nodes, which is small, this already becomes very heavy to calculate.

- Model complexity is extremely high: overfitting.



Images are Local and Hierarchical

Nearby pixels are more strongly related than distant ones.

Objects are built up out of smaller parts.

Images are Invariant





Basics of CNNs

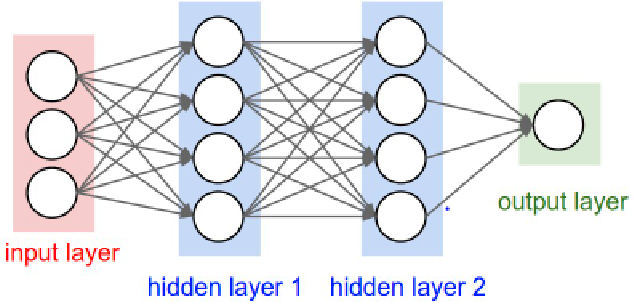
We know that MLPs:

- Do not scale well for images
- Ignore the information brought by pixel position and correlation with neighbors
- Cannot handle translations

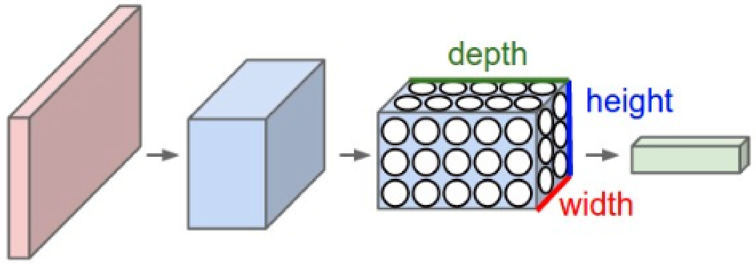
The general idea of CNNs is to intelligently adapt to properties of images:

- Pixel position and neighborhood have semantic meanings.
- Elements of interest can appear anywhere in the image.

Basics of CNNs



MLP



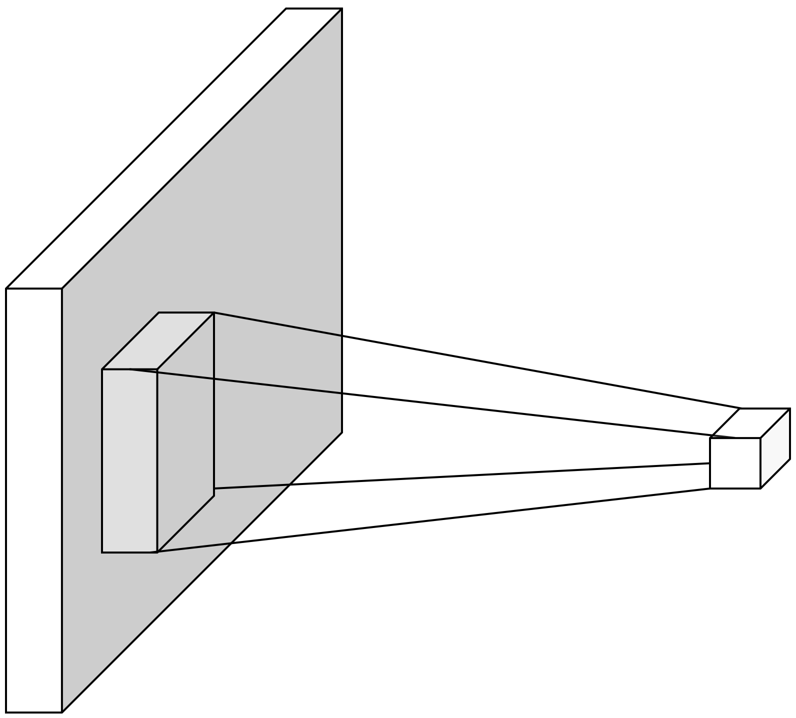
CNN

CNNs are also composed of layers, but those layers are not fully connected: they have filters, sets of cube-shaped weights that are applied throughout the image. Each 2D slice of the filters are called kernels.

These filters introduce translation invariance and parameter sharing.

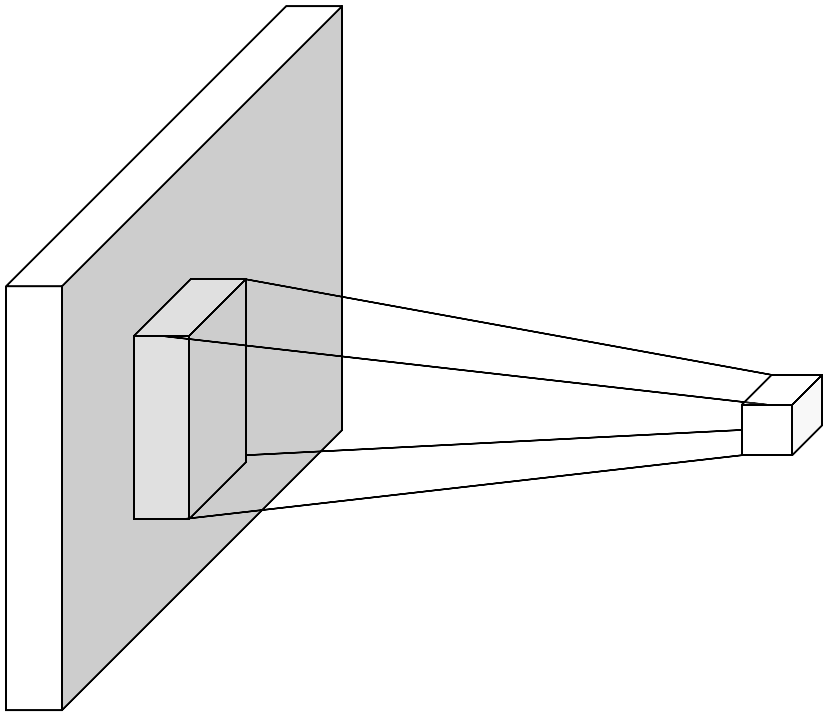
How are they applied? Convolutions!

Convolutional architecture



- Let's limit the *receptive fields* of units, tile them over the input image, and share their weights

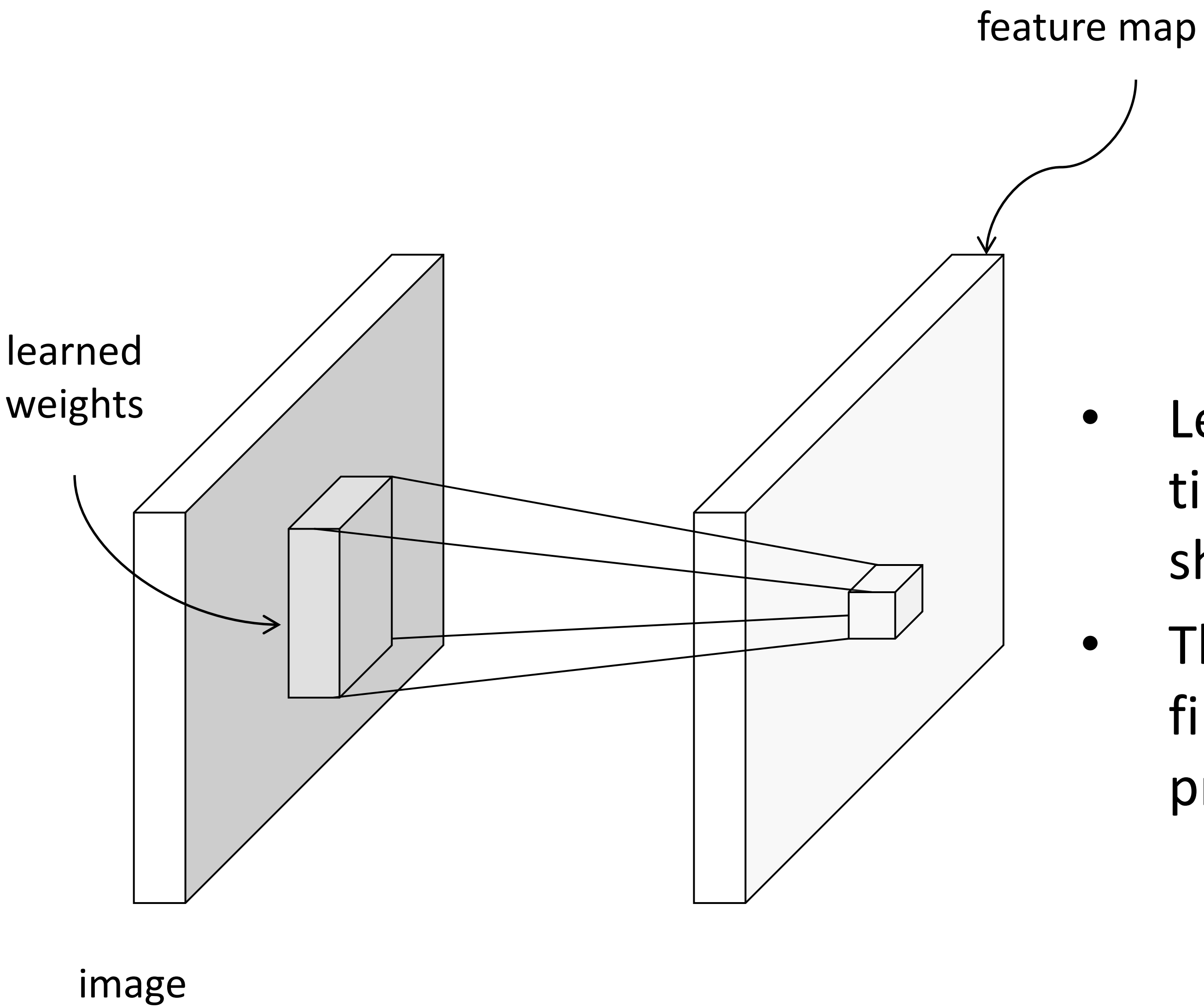
Convolutional architecture



image

- Let's limit the *receptive fields* of units, tile them over the input image, and share their weights

Convolutional architecture



- Let's limit the *receptive fields* of units, tile them over the input image, and share their weights
- This is equivalent to sliding the learned filter over the image, computing dot products at every location

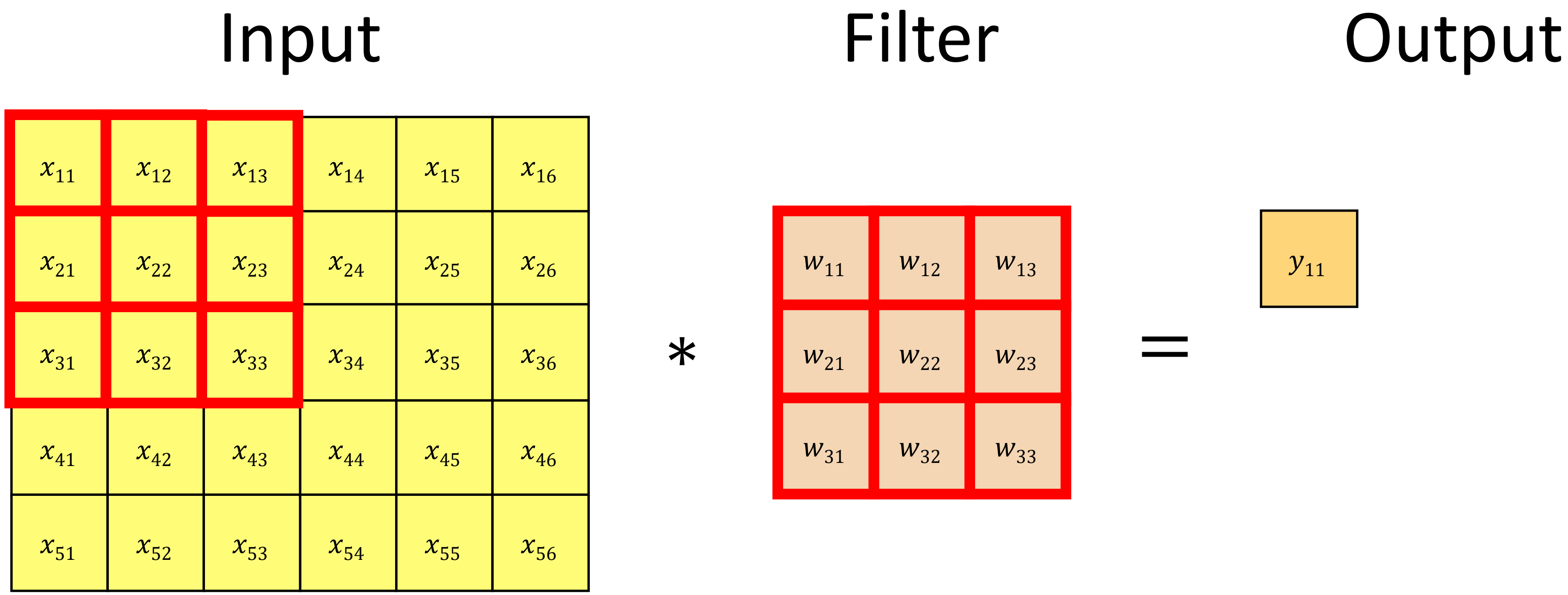
Convolution example

Input | Filter | Output

$$
\begin{bmatrix}
x_{11} & x_{12} & x_{13} & x_{14} & x_{15} & x_{16} \\
x_{21} & x_{22} & x_{23} & x_{24} & x_{25} & x_{26} \\
x_{31} & x_{32} & x_{33} & x_{34} & x_{35} & x_{36} \\
x_{41} & x_{42} & x_{43} & x_{44} & x_{45} & x_{46} \\
x_{51} & x_{52} & x_{53} & x_{54} & x_{55} & x_{56}
\end{bmatrix}
*
\begin{bmatrix}
w_{11} & w_{12} & w_{13} \\
w_{21} & w_{22} & w_{23} \\
w_{31} & w_{32} & w_{33}
\end{bmatrix}
=
$$

Adapted from [D. Fouhey and J. Johnson]

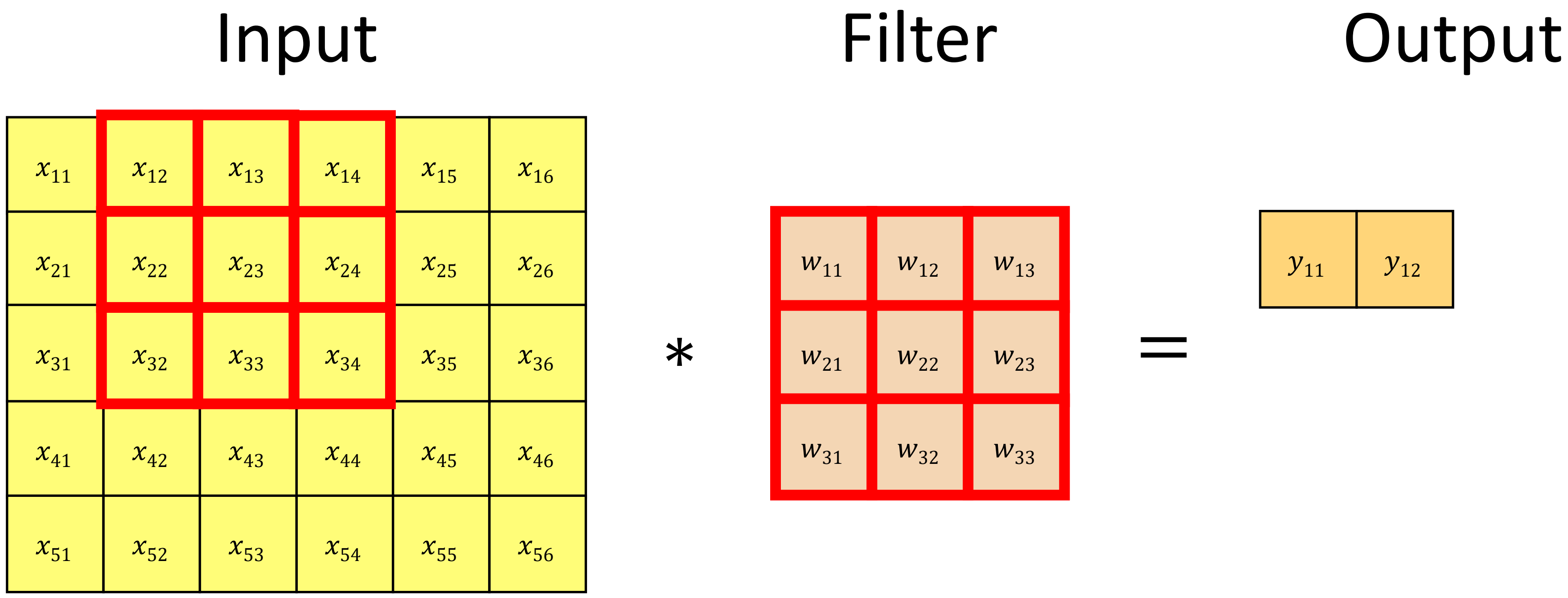
Convolution example



$$y_{11} = x_{11} \cdot w_{11} + x_{12} \cdot w_{12} + x_{13} \cdot w_{13} + \ldots + x_{33} \cdot w_{33}$$

Adapted from [D. Fouhey and J. Johnson]

Convolution example



$$y_{12} = x_{12} \cdot w_{11} + x_{13} \cdot w_{12} + x_{14} \cdot w_{13} + \ldots + x_{34} \cdot w_{33}$$

Adapted from [D. Fouhey and J. Johnson]

Convolution example

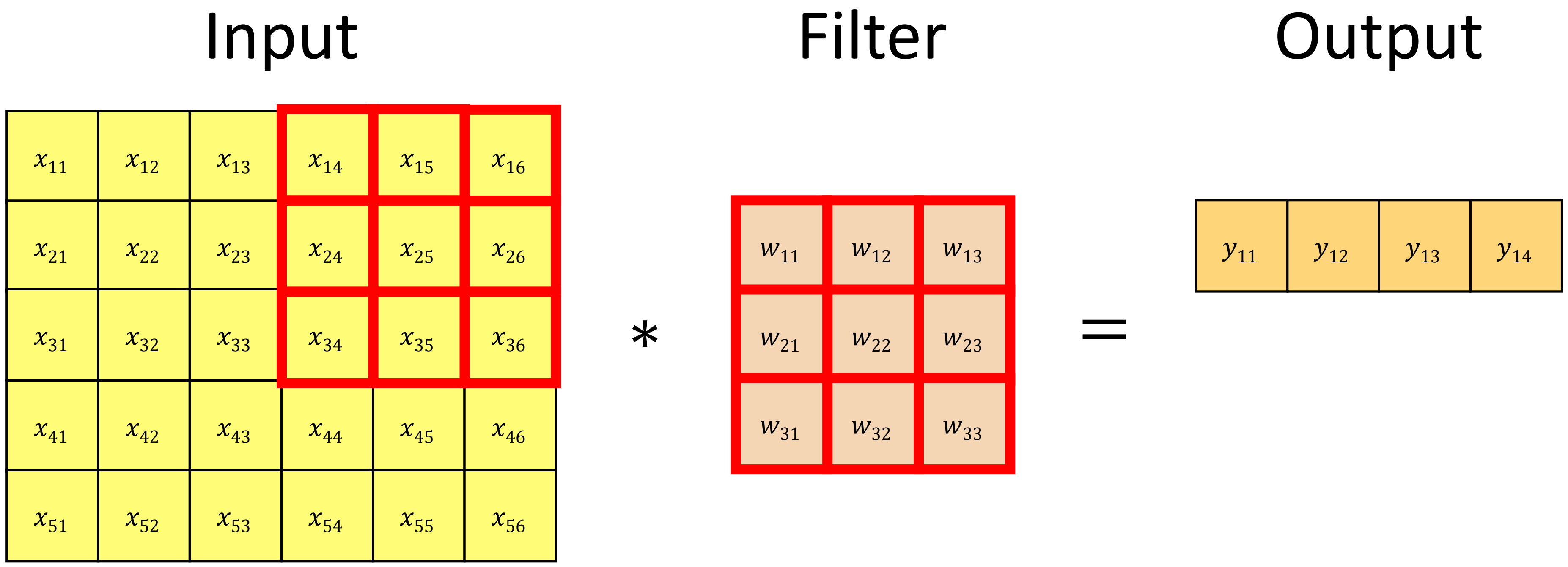
Input / Filter / Output

$$
\begin{bmatrix}
x_{11} & x_{12} & x_{13} & x_{14} & x_{15} & x_{16} \\
x_{21} & x_{22} & x_{23} & x_{24} & x_{25} & x_{26} \\
x_{31} & x_{32} & x_{33} & x_{34} & x_{35} & x_{36} \\
x_{41} & x_{42} & x_{43} & x_{44} & x_{45} & x_{46} \\
x_{51} & x_{52} & x_{53} & x_{54} & x_{55} & x_{56}
\end{bmatrix}
*
\begin{bmatrix}
w_{11} & w_{12} & w_{13} \\
w_{21} & w_{22} & w_{23} \\
w_{31} & w_{32} & w_{33}
\end{bmatrix}
=
\begin{bmatrix}
y_{11} & y_{12} & y_{13}
\end{bmatrix}
$$

$$y_{13} = x_{13} \cdot w_{11} + x_{14} \cdot w_{12} + x_{15} \cdot w_{13} + \ldots + x_{35} \cdot w_{33}$$

Adapted from [D. Fouhey and J. Johnson]

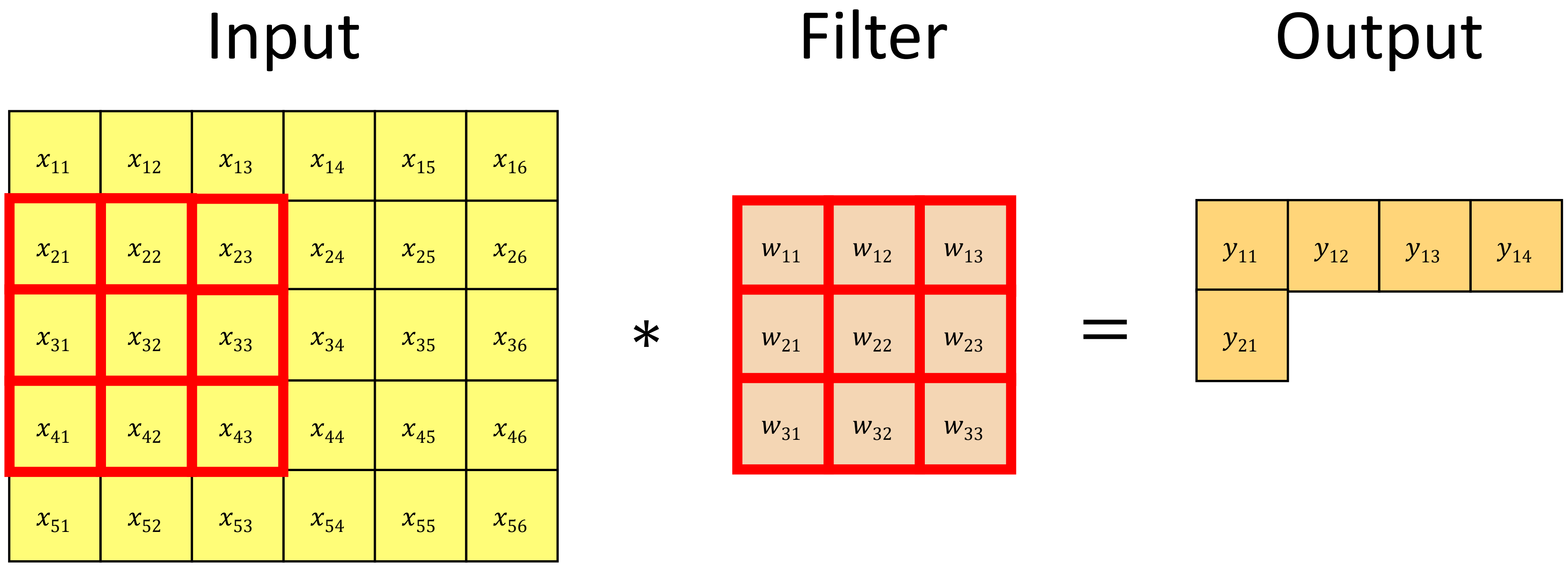
Convolution example



$$y_{14} = x_{14} \cdot w_{11} + x_{15} \cdot w_{12} + x_{16} \cdot w_{13} + \ldots + x_{36} \cdot w_{33}$$

Adapted from [D. Fouhey and J. Johnson]

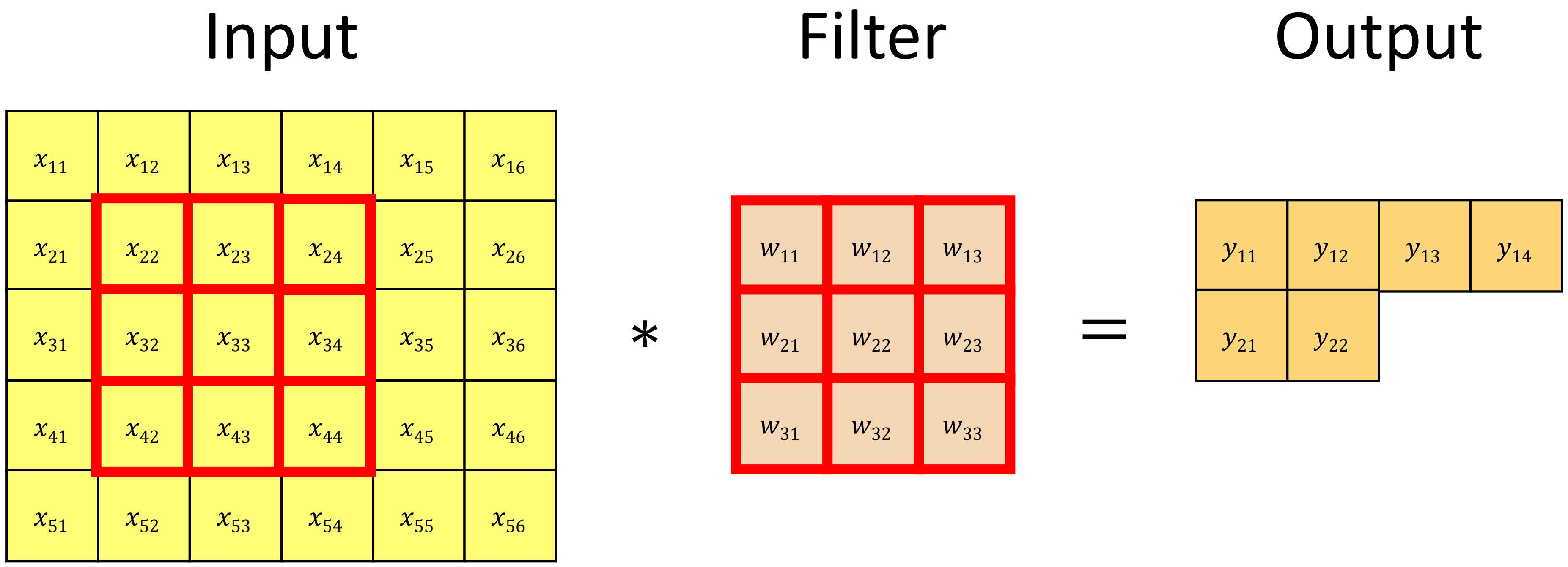
Convolution example



$$y_{21} = x_{21} \cdot w_{11} + x_{22} \cdot w_{12} + x_{23} \cdot w_{13} + \ldots + x_{43} \cdot w_{33}$$

Adapted from [D. Fouhey and J. Johnson]

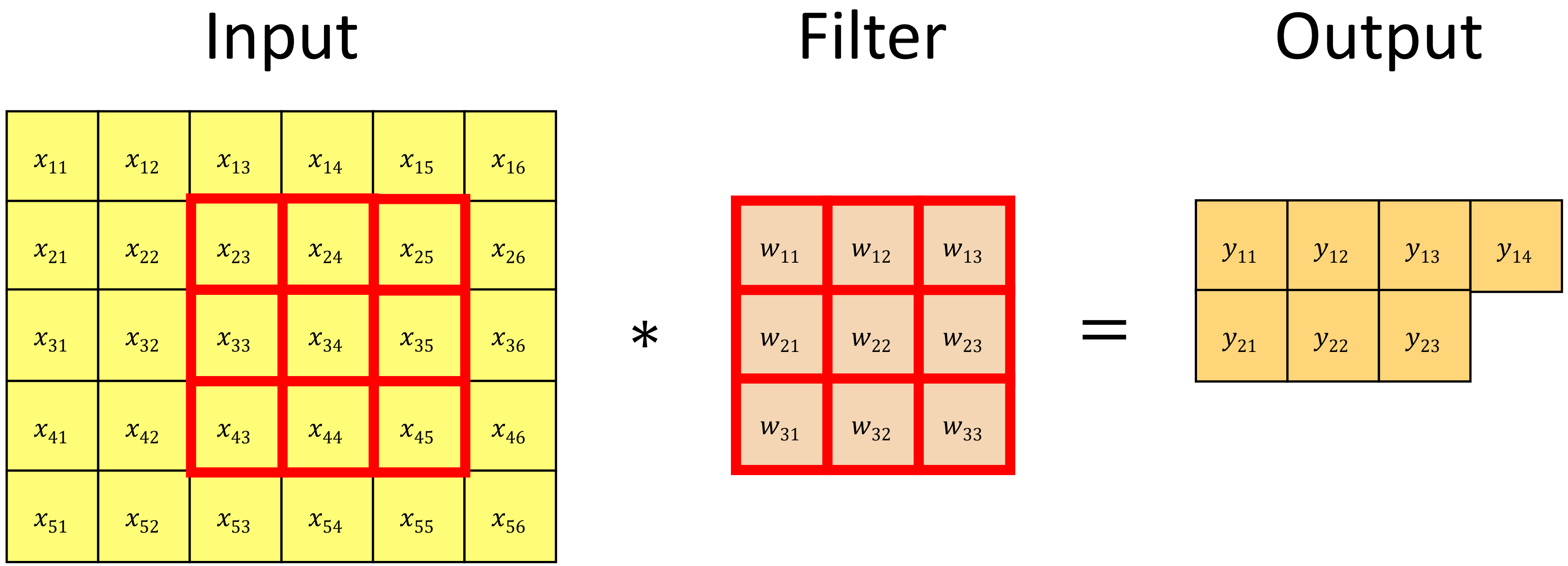
Convolution example



$$y_{22} = x_{22} \cdot w_{11} + x_{23} \cdot w_{12} + x_{24} \cdot w_{13} + \ldots + x_{44} \cdot w_{33}$$

Adapted from [D. Fouhey and J. Johnson]

Convolution example



$$y_{23} = x_{23} \cdot w_{11} + x_{24} \cdot w_{12} + x_{25} \cdot w_{13} + \ \ldots + x_{45} \cdot w_{33}$$

Adapted from [D. Fouhey and J. Johnson]

Convolution example

$$\overset{\text{Input}}{\begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{15} & x_{16} \\ x_{21} & x_{22} & x_{23} & x_{24} & x_{25} & x_{26} \\ x_{31} & x_{32} & x_{33} & x_{34} & x_{35} & x_{36} \\ x_{41} & x_{42} & x_{43} & x_{44} & x_{45} & x_{46} \\ x_{51} & x_{52} & x_{53} & x_{54} & x_{55} & x_{56} \end{bmatrix}} * \overset{\text{Filter}}{\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}} = \overset{\text{Output}}{\begin{bmatrix} y_{11} & y_{12} & y_{13} & y_{14} \\ y_{21} & y_{22} & y_{23} & y_{24} \\ y_{31} & y_{32} & y_{33} & y_{34} \end{bmatrix}}$$

Adapted from [D. Fouhey and J. Johnson]

Convolution and cross-correlation

- A **convolution** of f and g $(f * g)$ is defined as the integral of the product, having one of the functions inverted and shifted:

$$(f * g)(t) = \int_a f(a)g(t - a)da$$

Function is inverted and shifted left by t

- Discrete convolution:

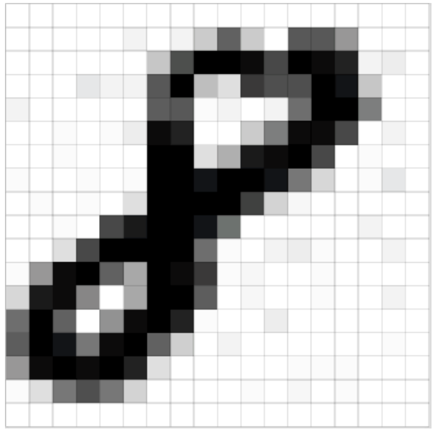
$$(f * g)(t) = \sum_{a=-\infty}^{\infty} f(a)g(t - a)$$

- Discrete cross-correlation:

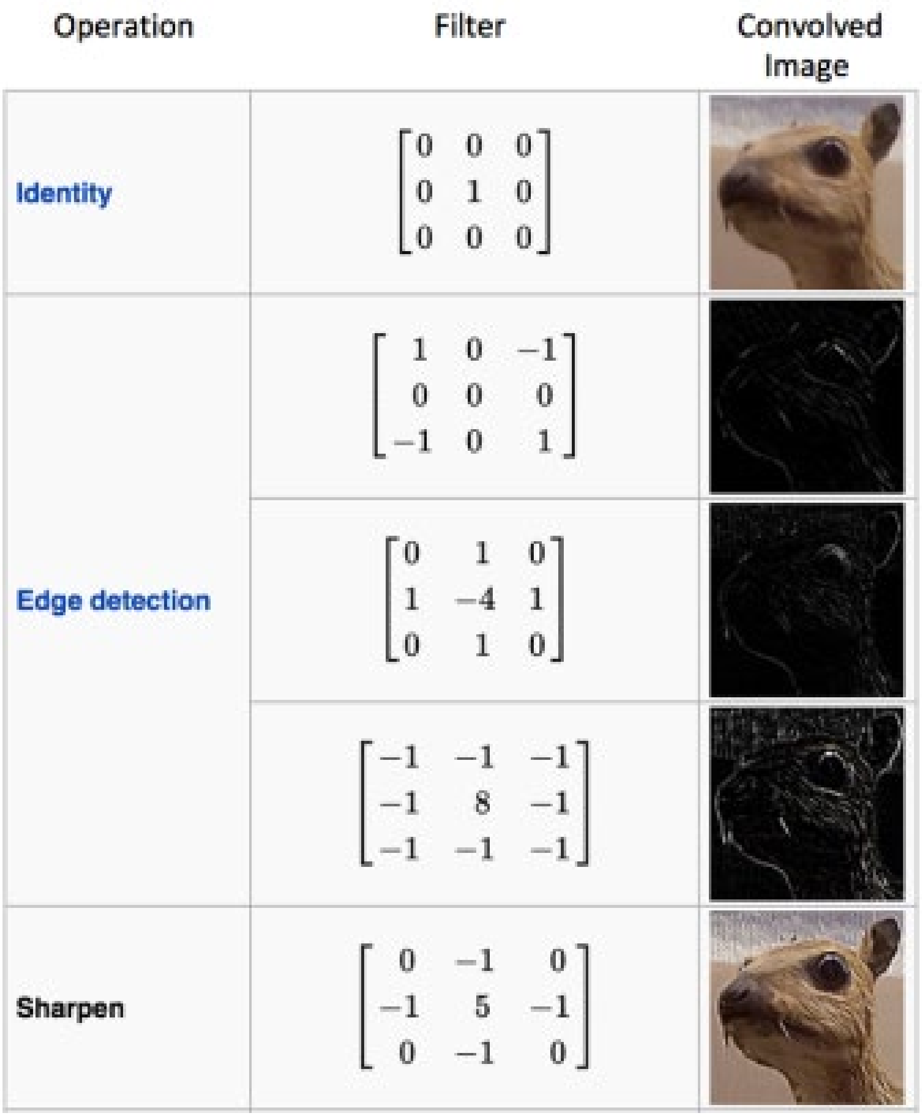
$$(f \star g)(t) = \sum_{a=-\infty}^{\infty} f(a)g(t + a)$$

Why does this make sense?

- In image is just a matrix of pixels.

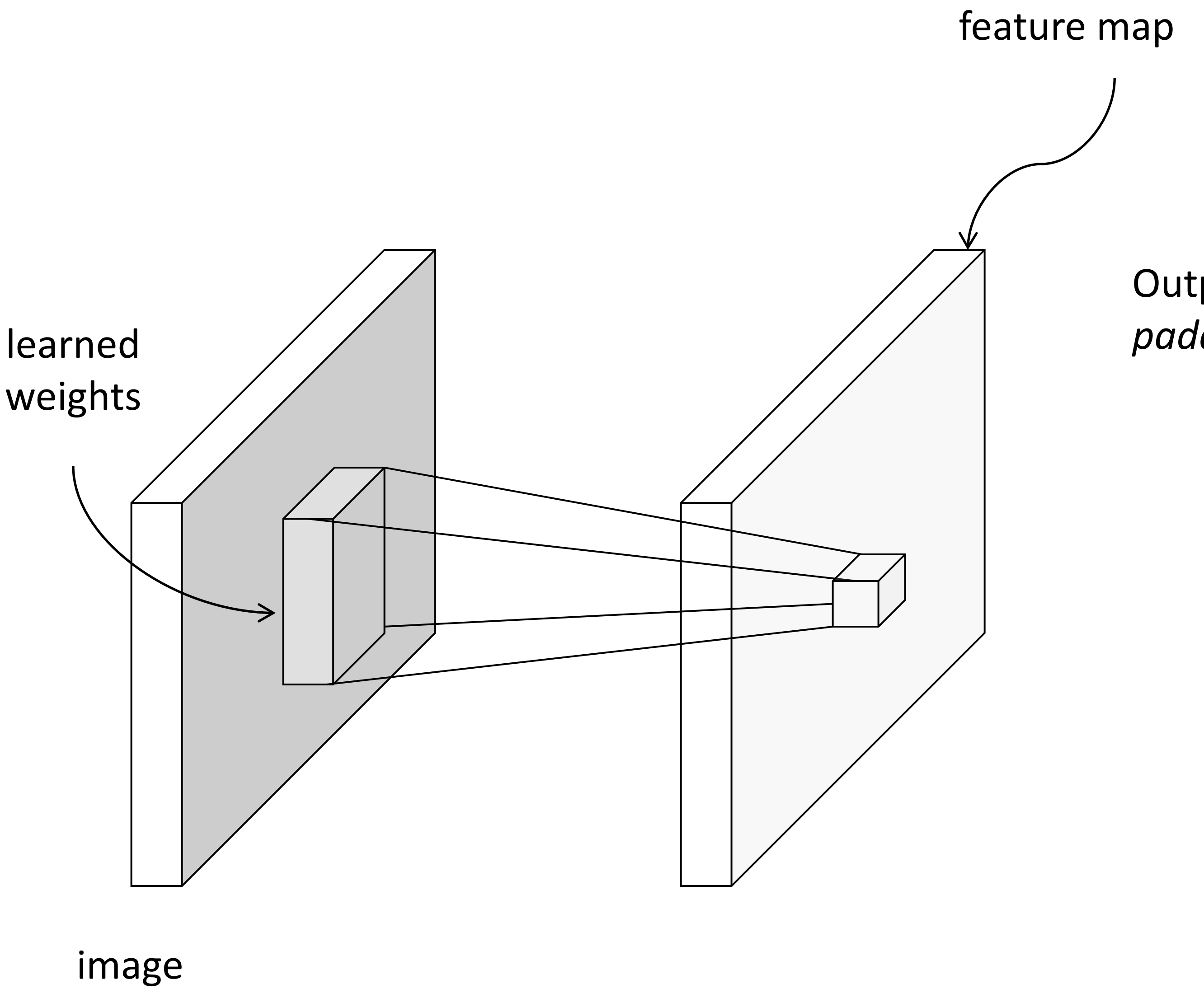


- Convolving the image with a filter produces a feature map that highlights the presence of a given feature in the image.

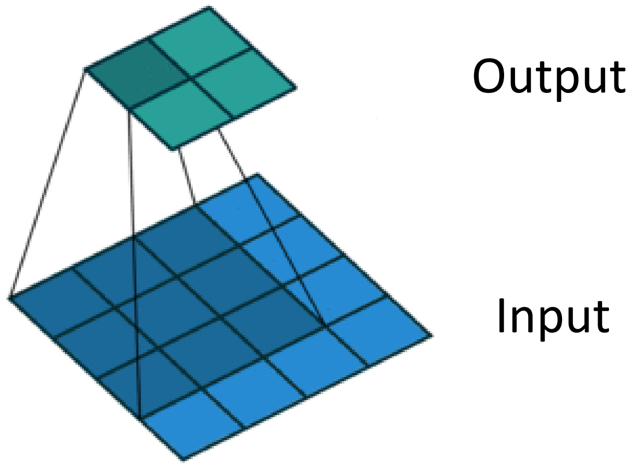


Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

Convolutional architecture



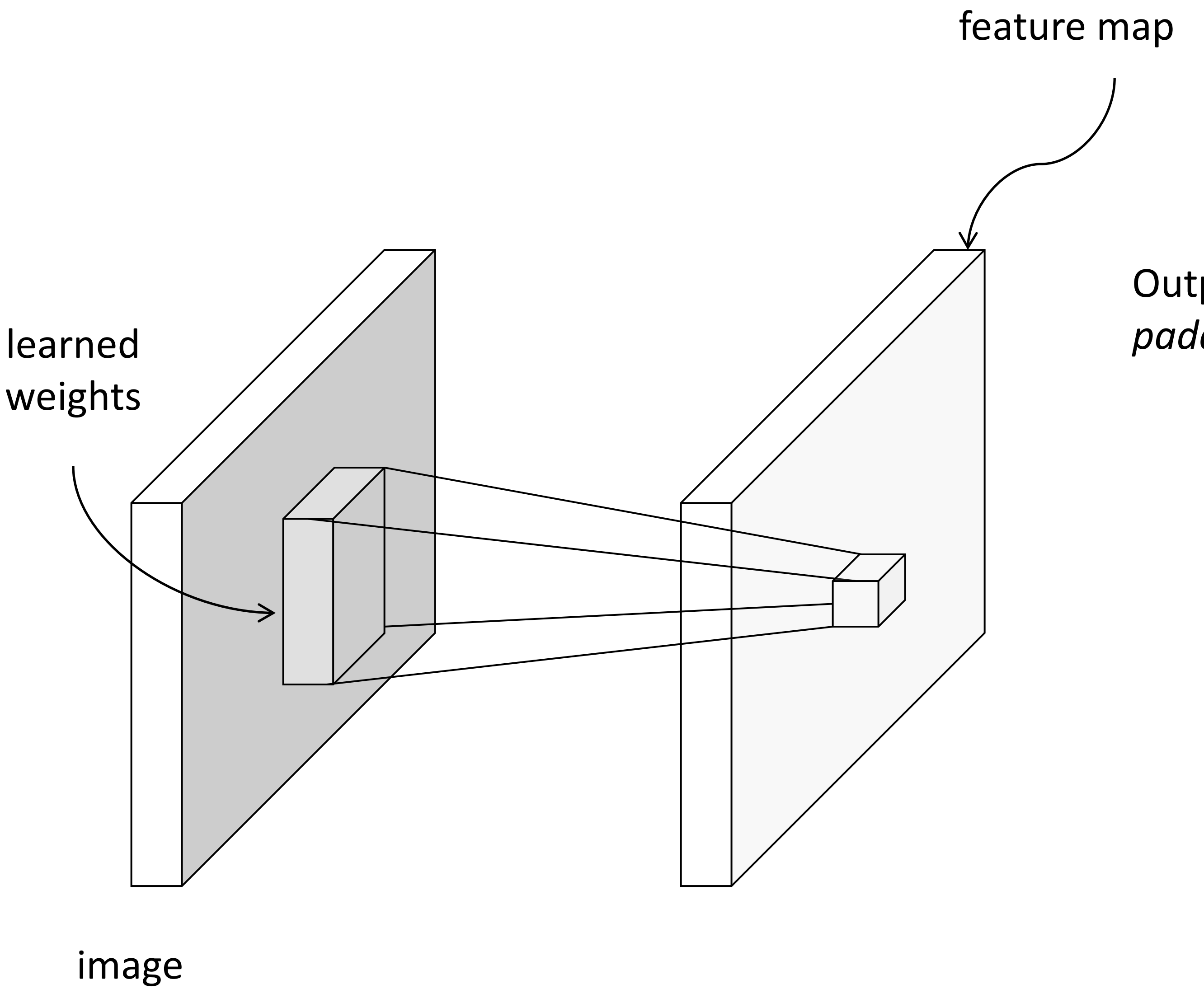
Output feature map resolution depends on *padding* and *stride*



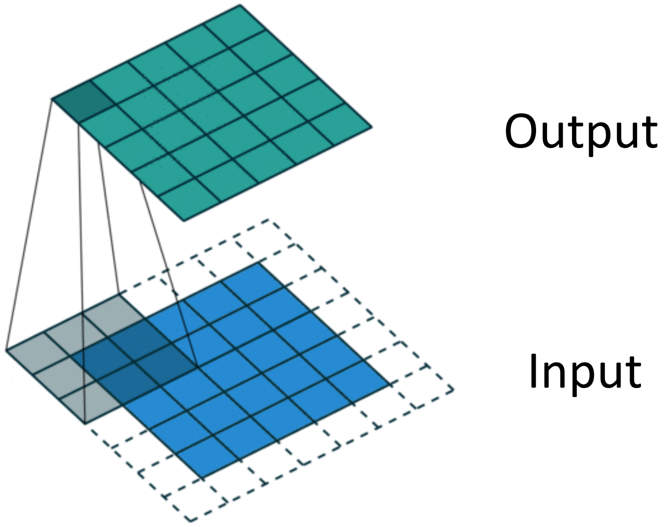
No padding, stride 1

Animation source

Convolutional architecture



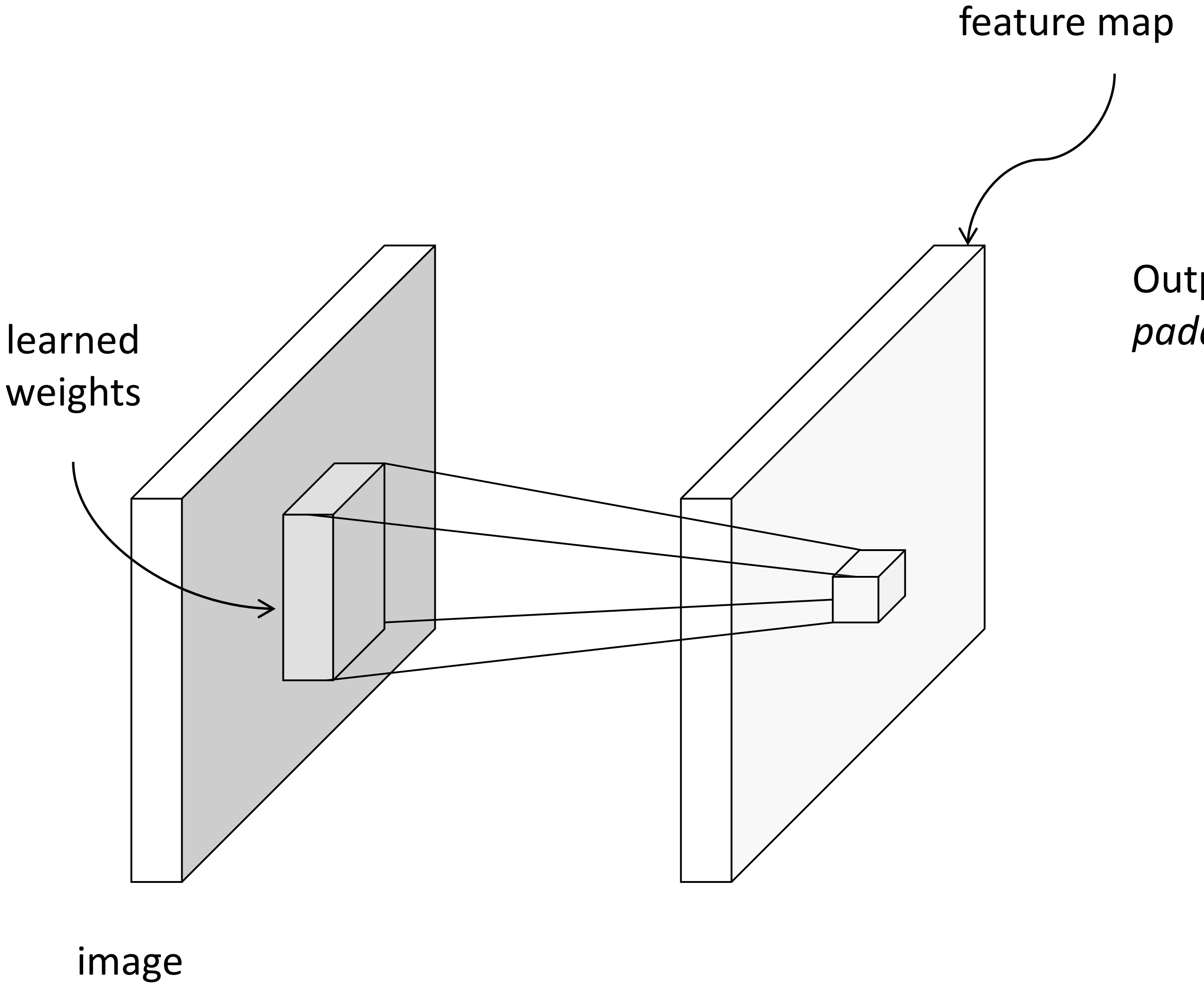
Output feature map resolution depends on *padding* and *stride*



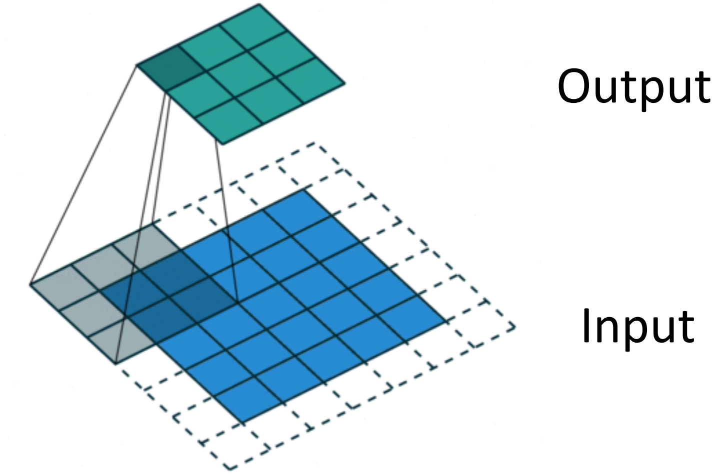
With padding, stride 1

[Animation source]

Convolutional architecture



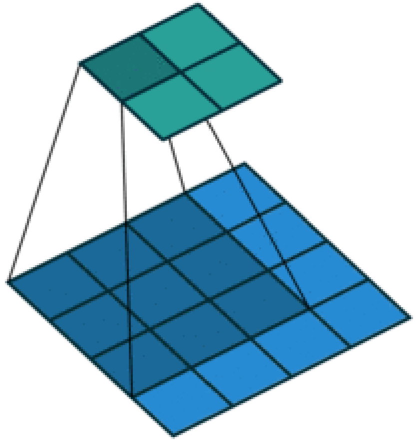
Output feature map resolution depends on *padding* and *stride*



With padding, stride 2

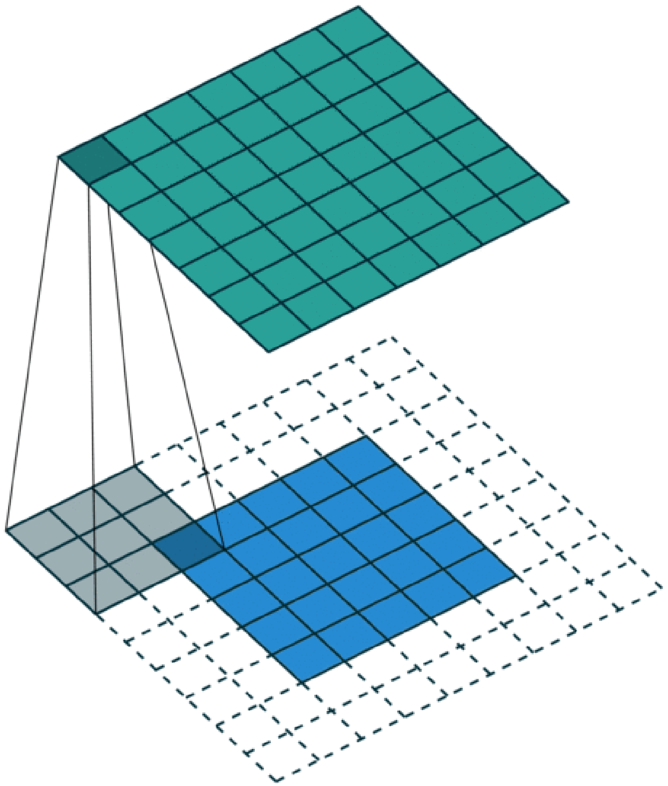
Convolutions – what happens at the edges?

If we apply convolutions on a normal image, the result will be down-sampled by an amount depending on the size of the filter.

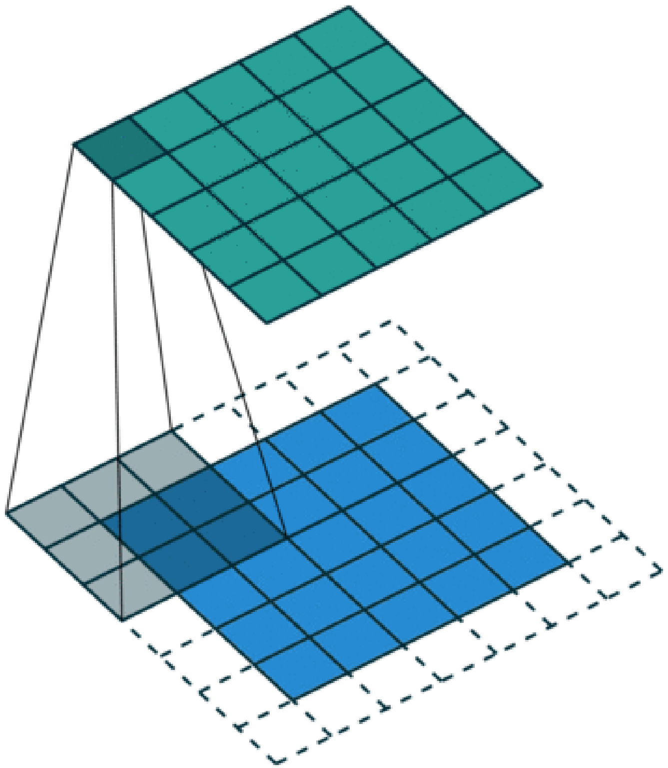


We can avoid this by padding the edges in different ways.

Padding

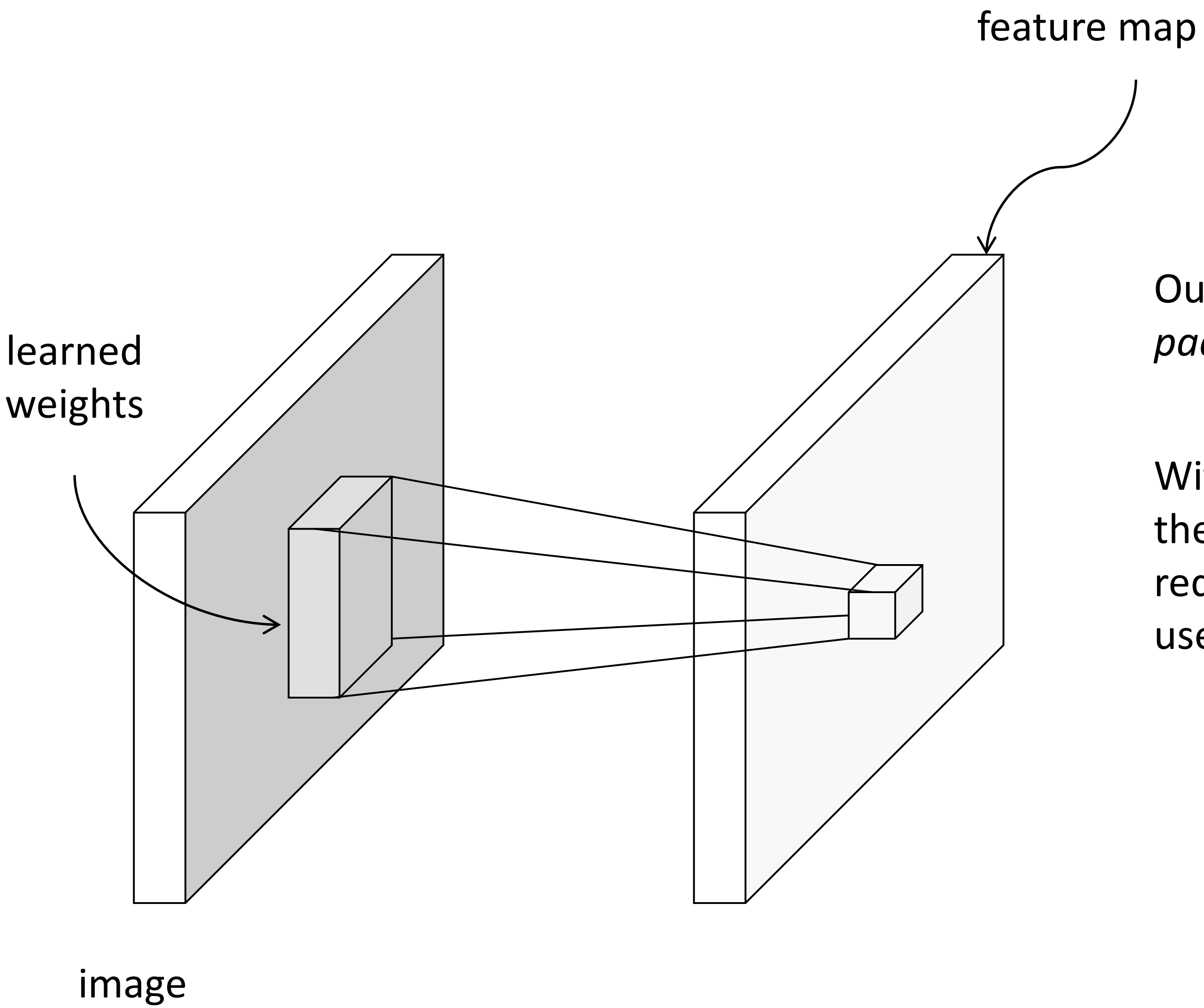


Full padding. Introduces zeros such that all pixels are visited the same amount of times by the filter. Increases size of output.



Same padding. Ensures that the output has the same size as the input.

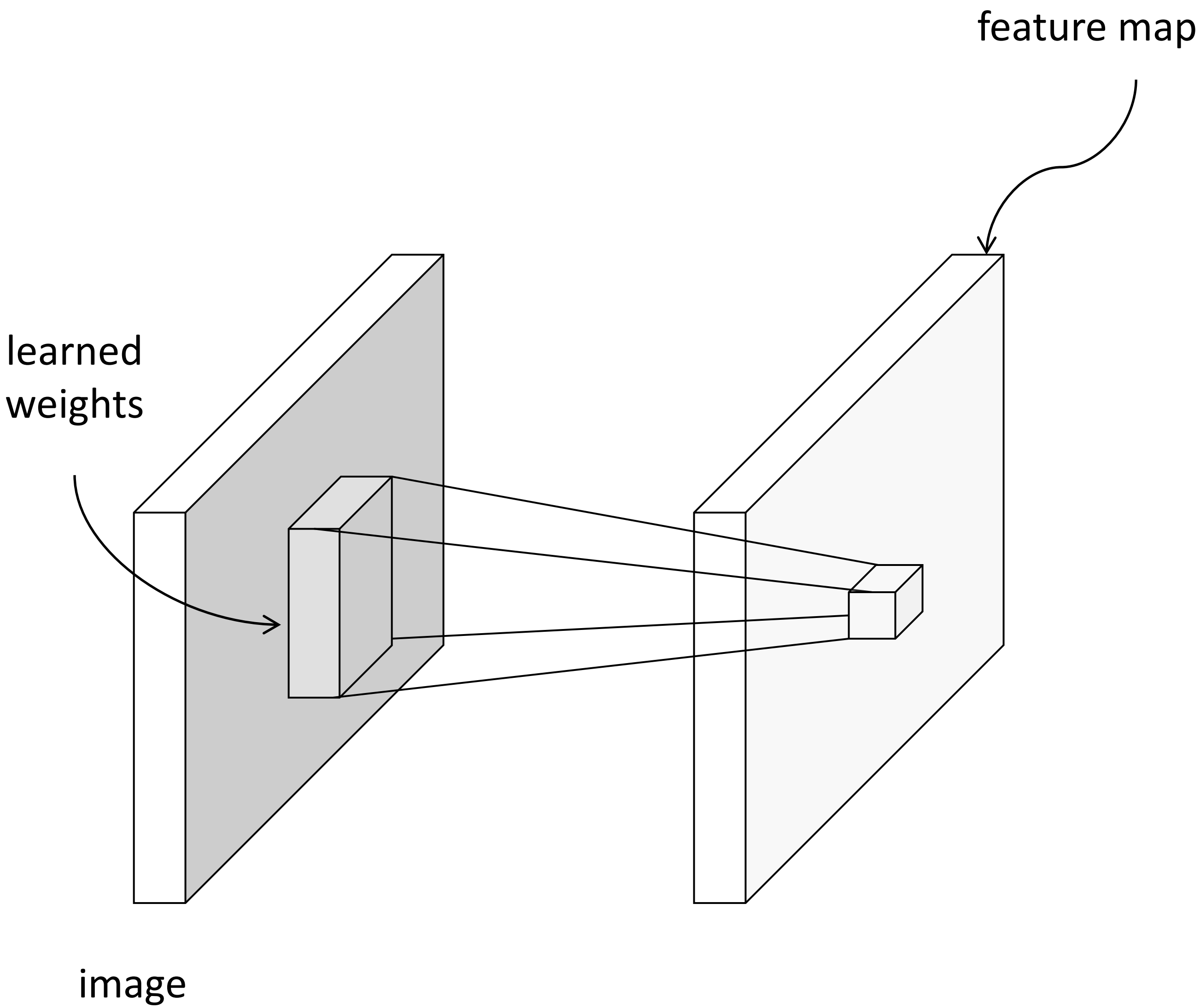
Convolutional architecture



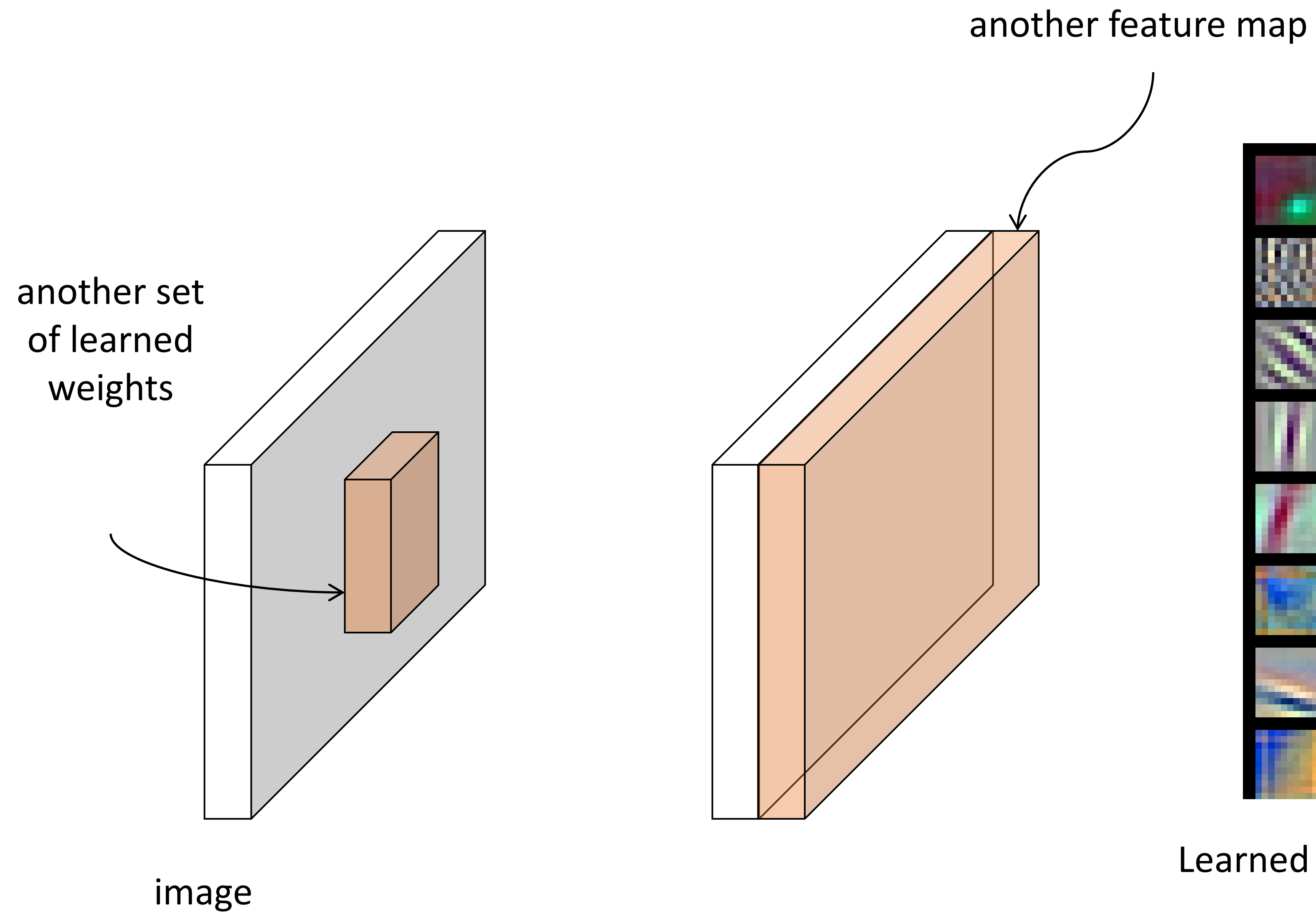
Output feature map resolution depends on *padding* and *stride*

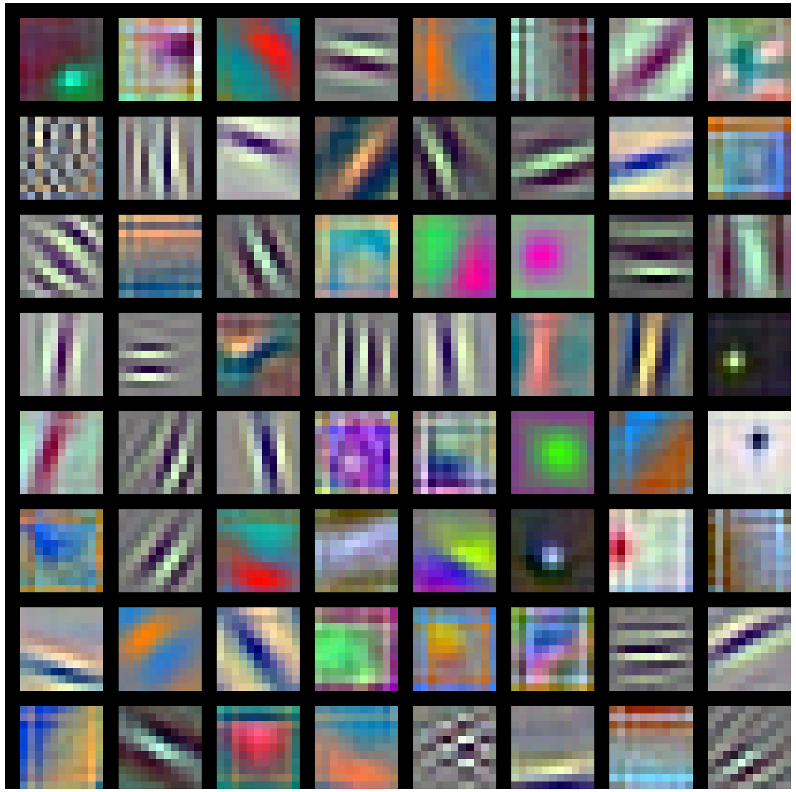
With padding, spatial resolution remains the same if stride of 1 is used, is reduced by factor of $1/S$ if stride of S is used

Convolutional architecture



Convolutional architecture

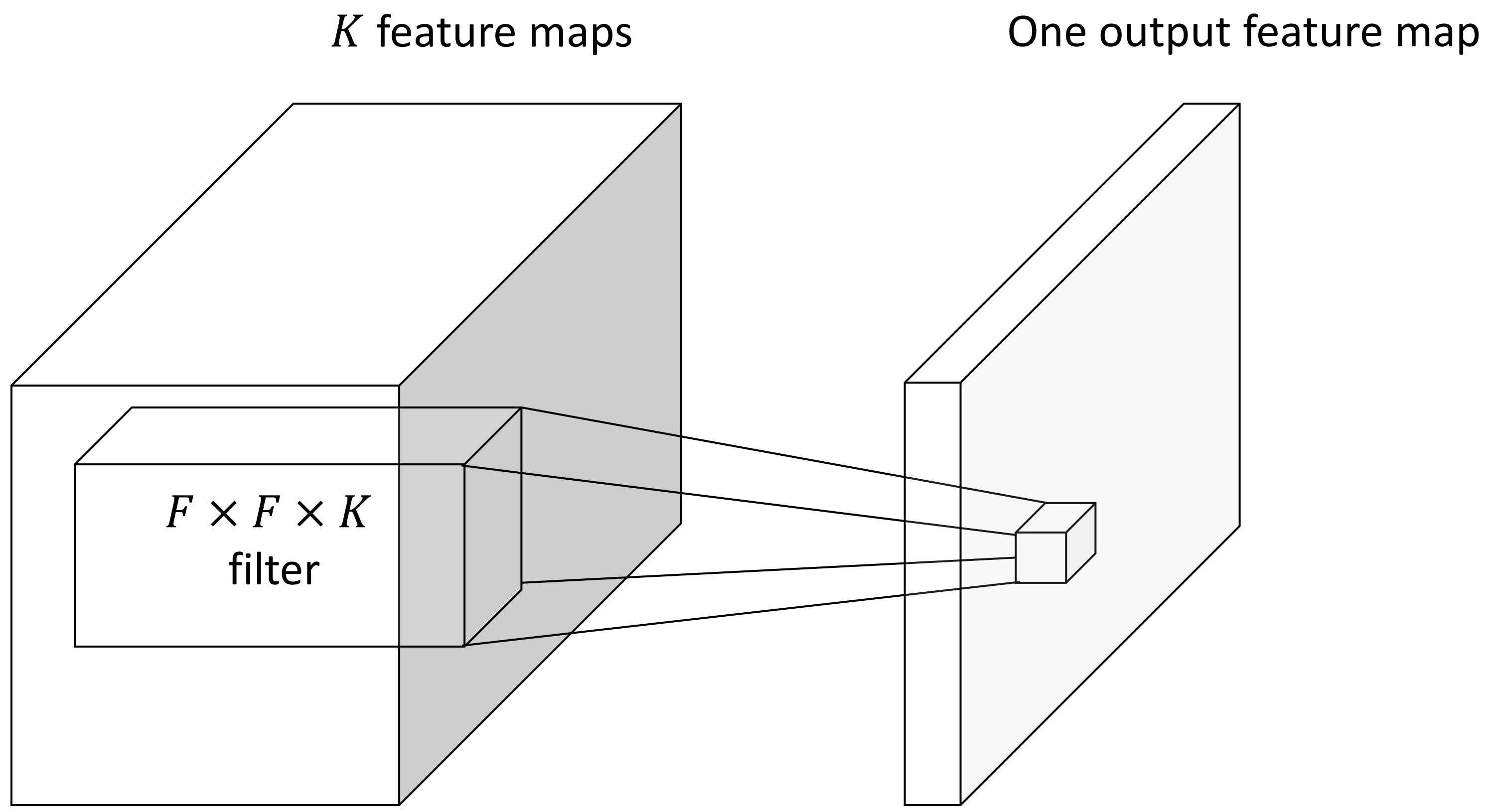




Learned weights can be thought of as *local* templates

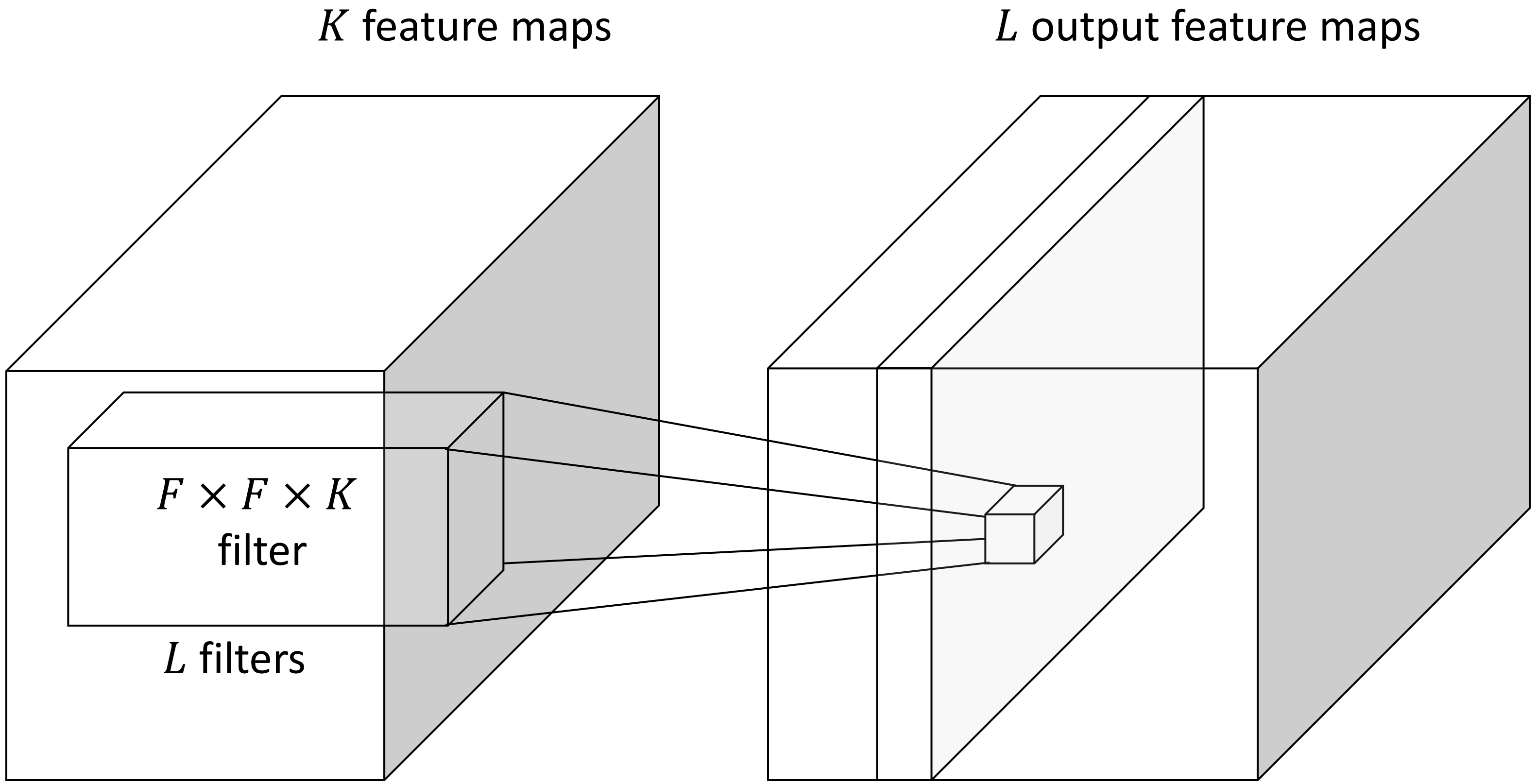
Three-dimensional convolutions

What if the *input* to a convolutional layer is a stack of K feature maps?

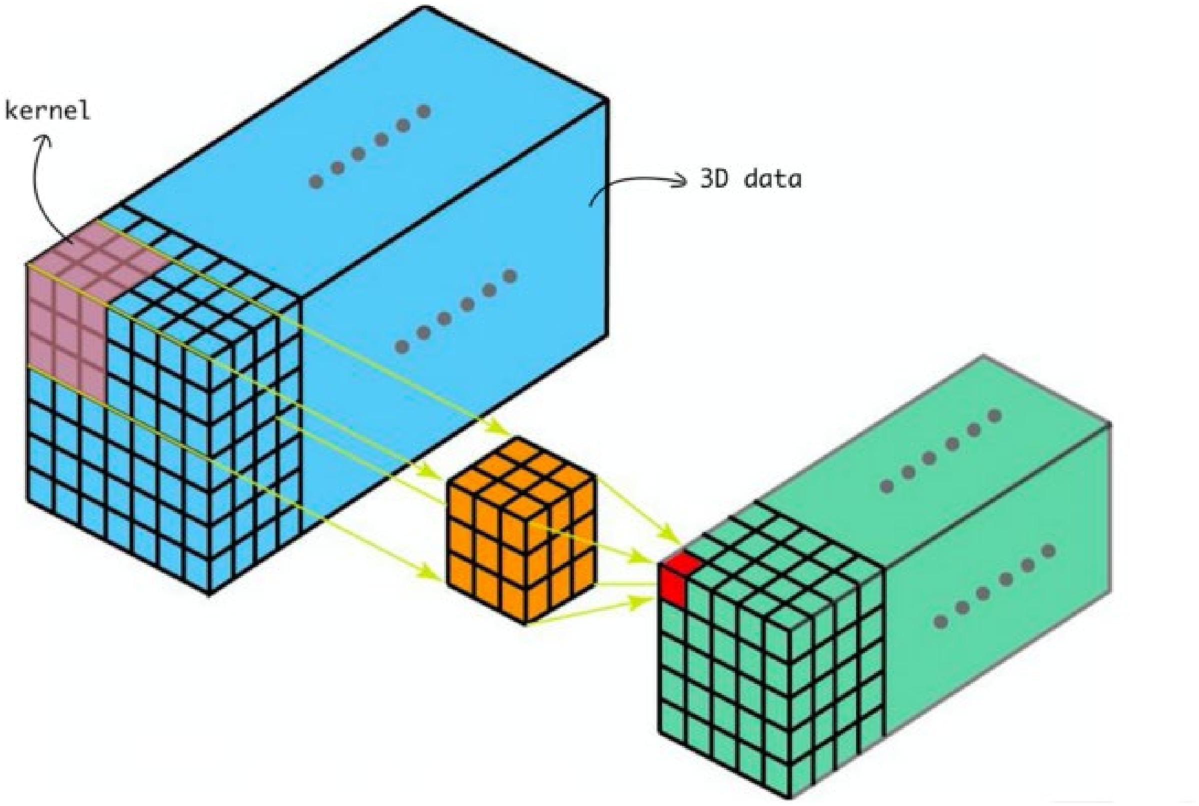


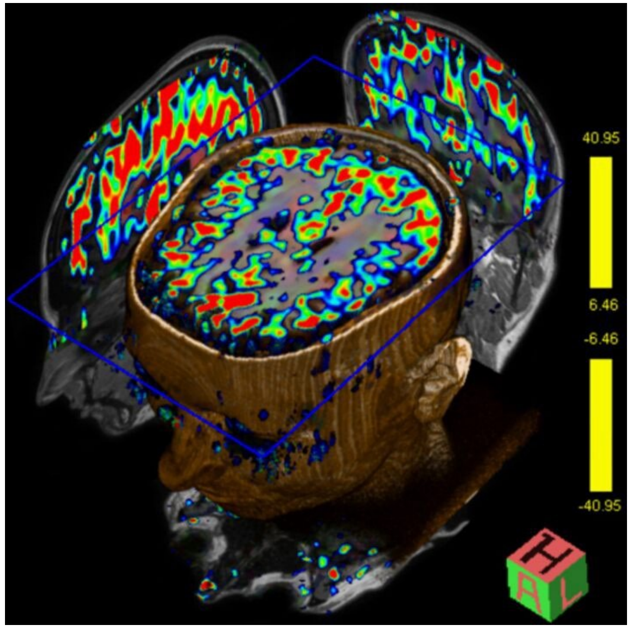
Three-dimensional convolutions

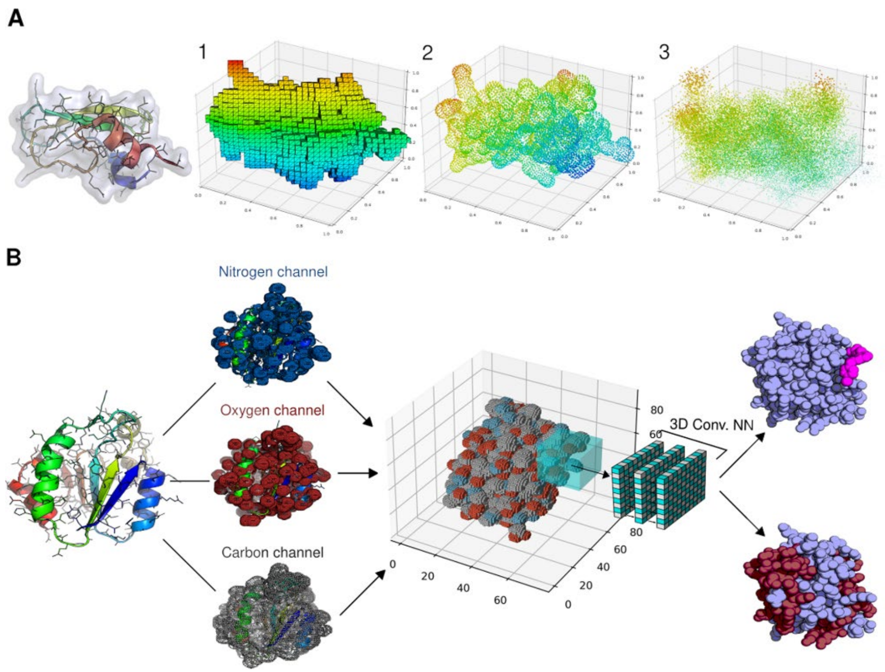
What if the *input* to a convolutional layer is a stack of K feature maps?



3-dimensional CNN aka Conv3D





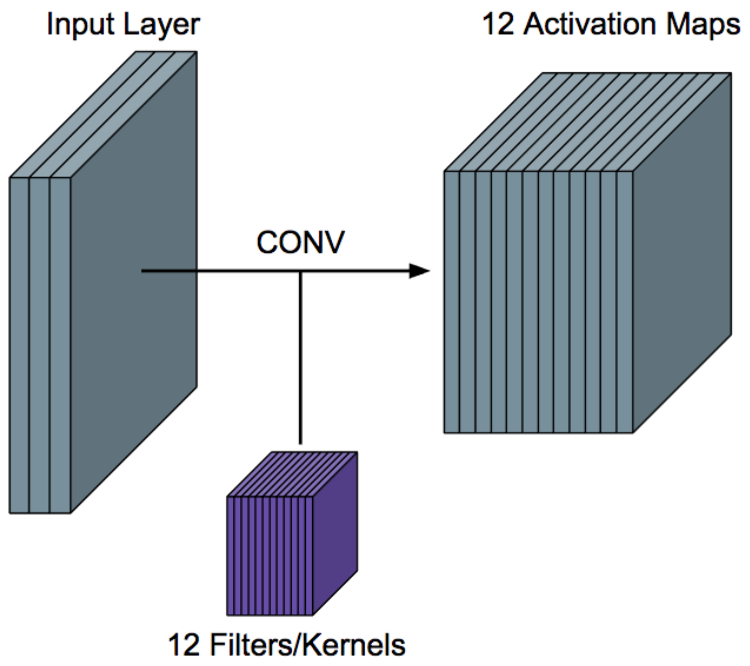


Convolutional layers (cont)

To be clear: each filter is convolved with the entirety of the 3D input cube, but generates a 2D feature map.

Because we have multiple filters, we end up with a 3D output: one 2D feature map per filter.

The feature map dimension can change drastically from one conv layer to the next: we can enter a layer with a 32x32x16 input and exit with a 32x32x128 output if that layer has 128 filters.



Learning CNN

In a convolutional layer, we are basically applying multiple filters at over the image to extract different features.

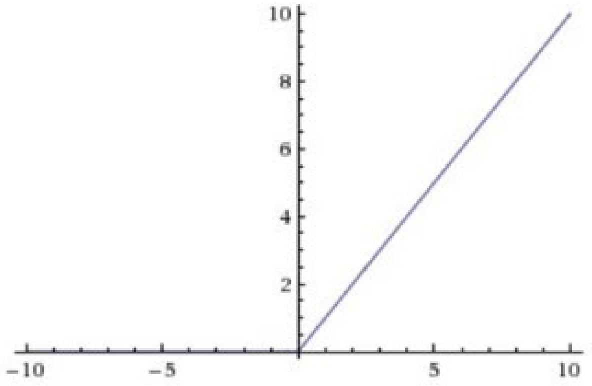
But most importantly, we are learning those filters!

One thing we're missing: non-linearity.

ReLU

The most successful non-linearity for CNNs is the Rectified Non-Linear unit (ReLU):

Output = Max(zero, Input)

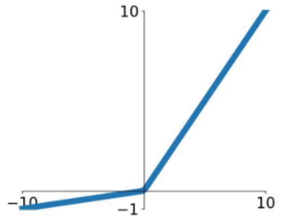


Combats the vanishing gradient problem occurring in sigmoids, is easier to compute, generates sparsity (not always beneficial)

Some alternatives to ReLU:

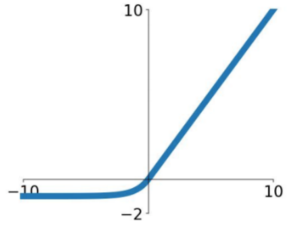
Leaky ReLU

$$\max(0.1x, x)$$

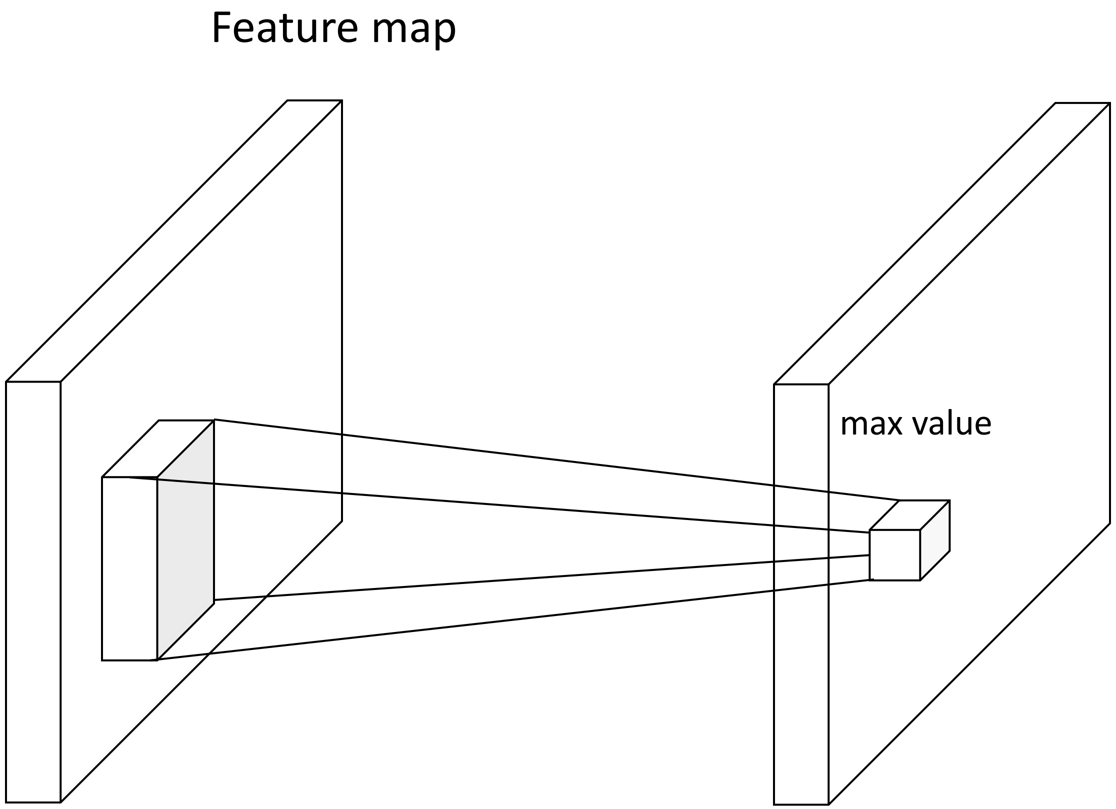


ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Max pooling layer



$F \times F$ pooling window,
stride S

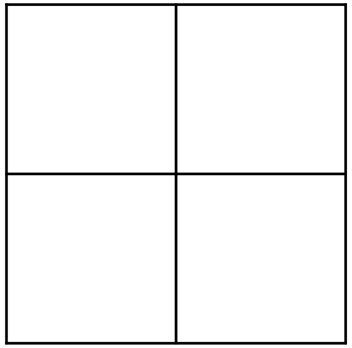
Usually: $F = 2$ or 3, $S = 2$

Max pooling: Example

Single channel

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

Max pooling with 2×2 kernel size and stride 2



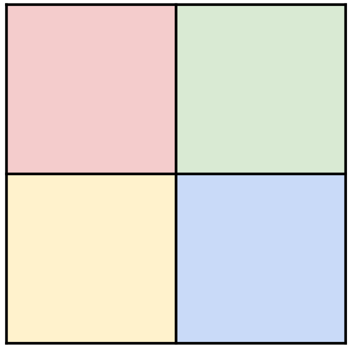
Source: [J. Johnson]

Max pooling: Example

Single channel

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

Max pooling with 2×2 kernel size and stride 2



Source: J. Johnson

Max pooling: Example

Single channel

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

Max pooling with 2×2 kernel size and stride 2

6	8
3	4

Source: [J. Johnson]

Convolutional layers so far

- A convolutional layer convolves each of its filters with the input.
- Input: a 3D tensor, where the dimensions are Width, Height and Channels (or Feature Maps)
- Output: a 3D tensor, with dimensions Width, Height and Feature Maps (one for each filter)
- Applies non-linear activation function (usually ReLU) over each value of the output.
- Multiple parameters to define: number of filters, size of filters, stride, padding, activation function to use, regularization.

Building a CNN

A convolutional neural network is built by stacking layers, typically of 3 types:

- Convolutional Layers
- Pooling Layers
- Fully connected Layers

Building a CNN

• A
3

Convolutional Layers

Action

- Apply filters to extract features
- Filters are composed of small kernels, learned.
- One bias per filter.
- Apply activation function on every value of feature map

Parameters

- Number of kernels
- Size of kernels (W and H only, D is defined by input cube)
- Activation function
- Stride
- Padding
- Regularization type and value

I/O

- Input: 3D cube, previous set of feature maps
- Output: 3D cube, one 2D map per filter

Building a CNN

- A convolutional neural network is built by stacking layers, typically of 3 types:

Convolutional Layers

Pooling Layers

Fully connected Layers

Building a CNN

• A … of
3

Pooling Layers

Action

- Reduce dimensionality
- Extract maximum of average of a region
- Sliding window approach

Parameters

- Stride
- Size of window

I/O

- Input: 3D cube, previous set of feature maps
- Output: 3D cube, one 2D map per filte, reduced spatial dimensions

Building a CNN

- A convolutional neural network is built by stacking layers, typically of 3 types:

Convolutional Layers

Pooling Layers

Fully connected Layers

Building a CNN

Fully connected Layers

Action

- Aggregate information from final feature maps
- Generate final classification

Parameters

- Number of nodes
- Activation function: usually changes depending on role of layer. If aggregating info, use ReLU. If producing final classification, use Softmax.

I/O

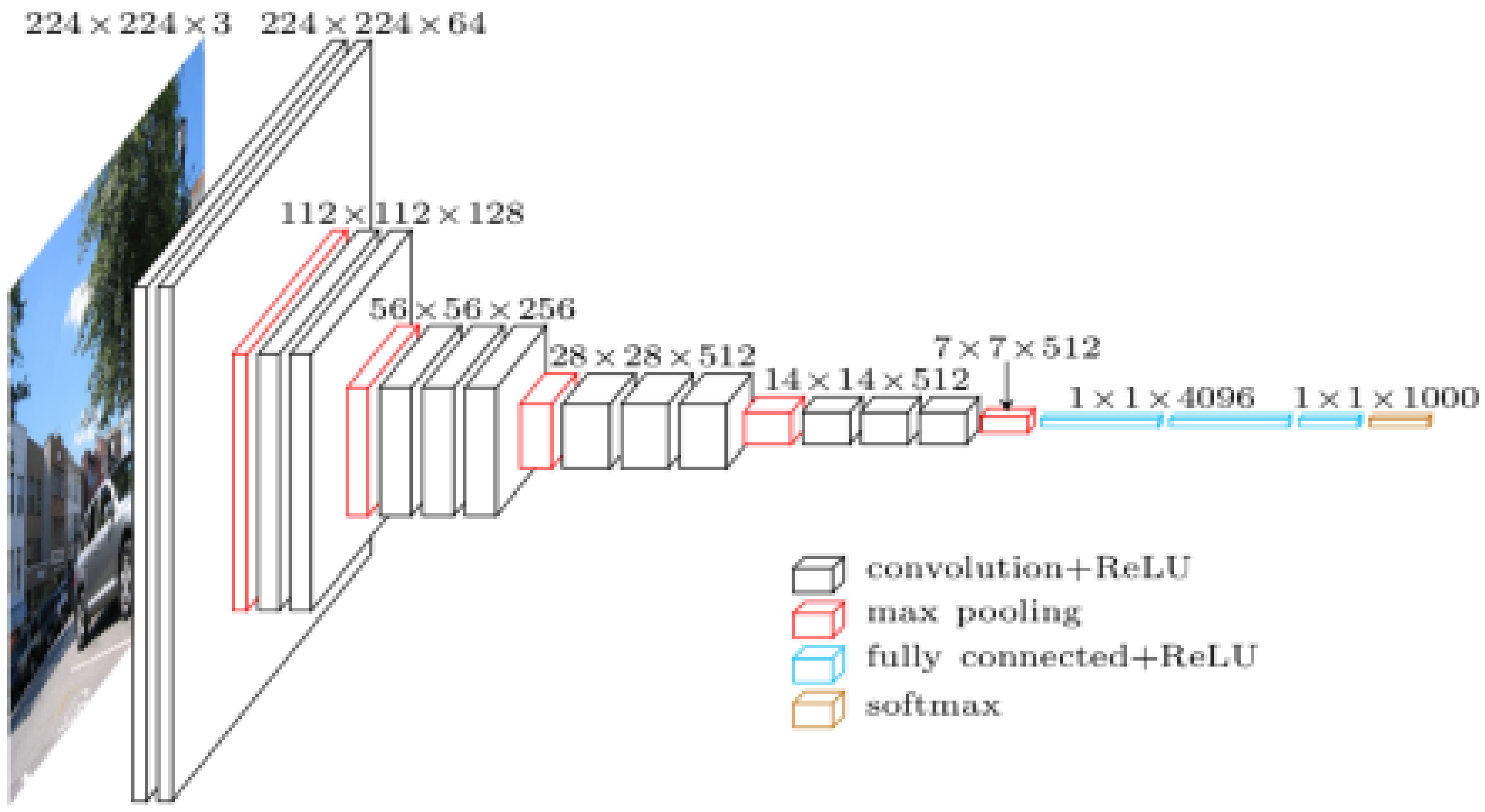
- Input: FLATTENED 3D cube, previous set of feature maps
- Output: 3D cube, one 2D map per filter

Examples

- I have a convolutional layer with 16 3x3 filters that takes an RGB image as input.
  - What else can we define about this layer?
    - Activation function
    - Stride
    - Padding type
  - How many parameters does the layer have?

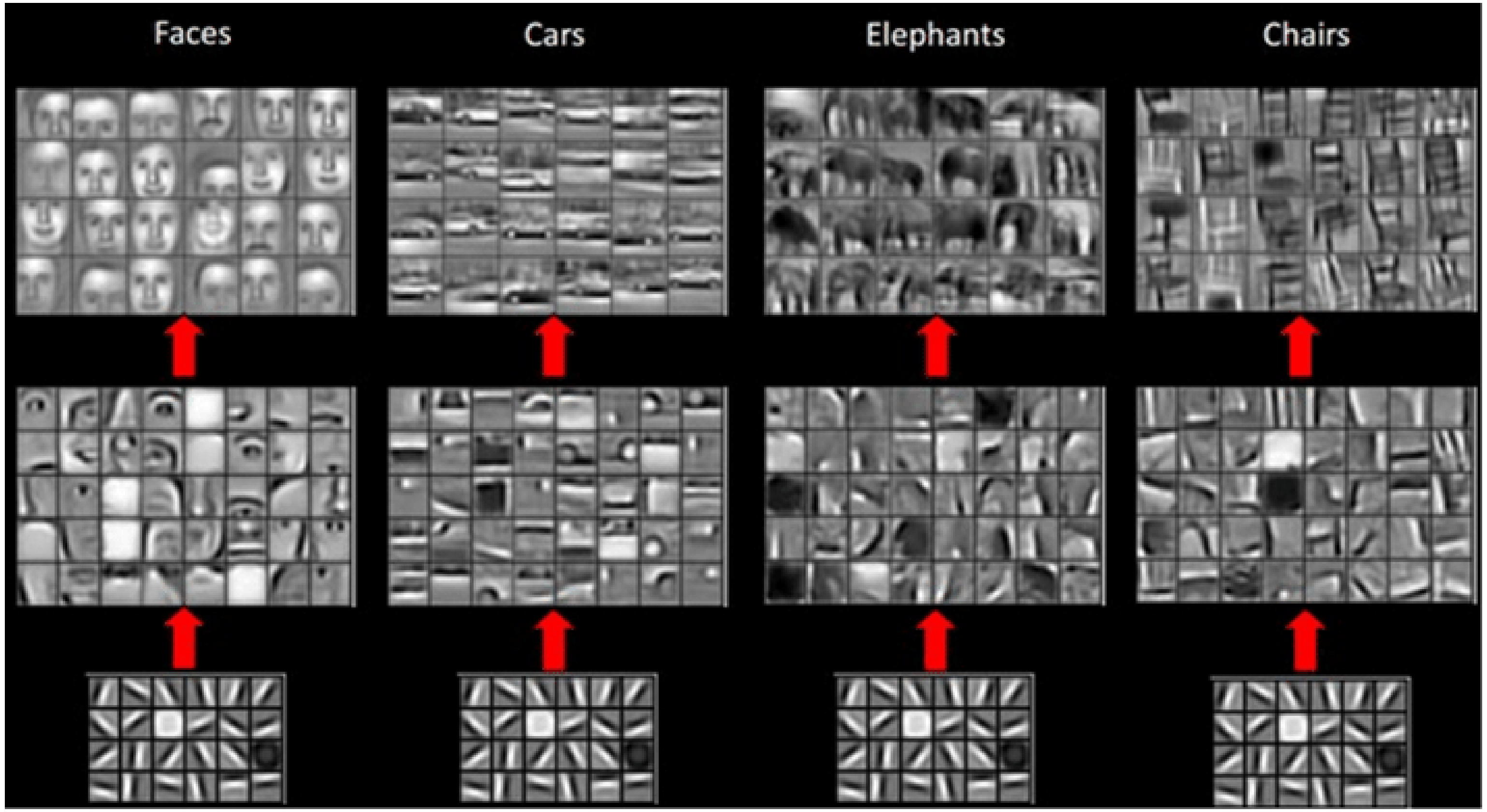
$$\underbrace{16}_{\text{Number of filters}} \times \underbrace{3 \times 3}_{\text{Size of Filters}} \times \underbrace{3}_{\text{Number of channels of prev layer}} + \underbrace{16}_{\text{Biases (one per filter)}} = 448$$

Fully built CNN (VGG)



What do CNN layers learn?

- Each CNN layer learns filters of increasing complexity.
- The first layers learn basic feature detection filters: edges, corners, etc.
- The middle layers learn filters that detect parts of objects. For faces, they might learn to respond to eyes, noses, etc.
- The last layers have higher representations: they learn to recognize full objects, in different shapes and positions.


Faces
Cars
Elephants
Chairs

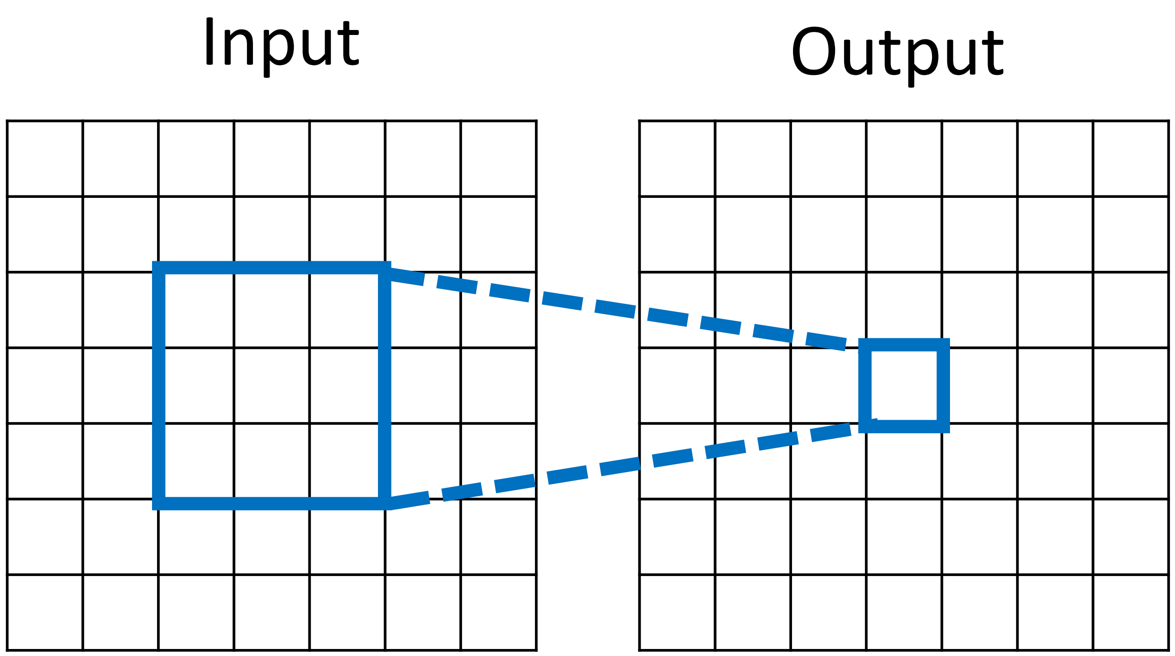
- 3D visualization of networks in action

- http://scs.ryerson.ca/~aharley/vis/conv/
- https://www.youtube.com/watch?v=3JQ3hYko51Y

Receptive field

The *receptive field* of a unit is the region of the input feature map whose values contribute to the response of that unit (either in the previous layer or in the initial image)

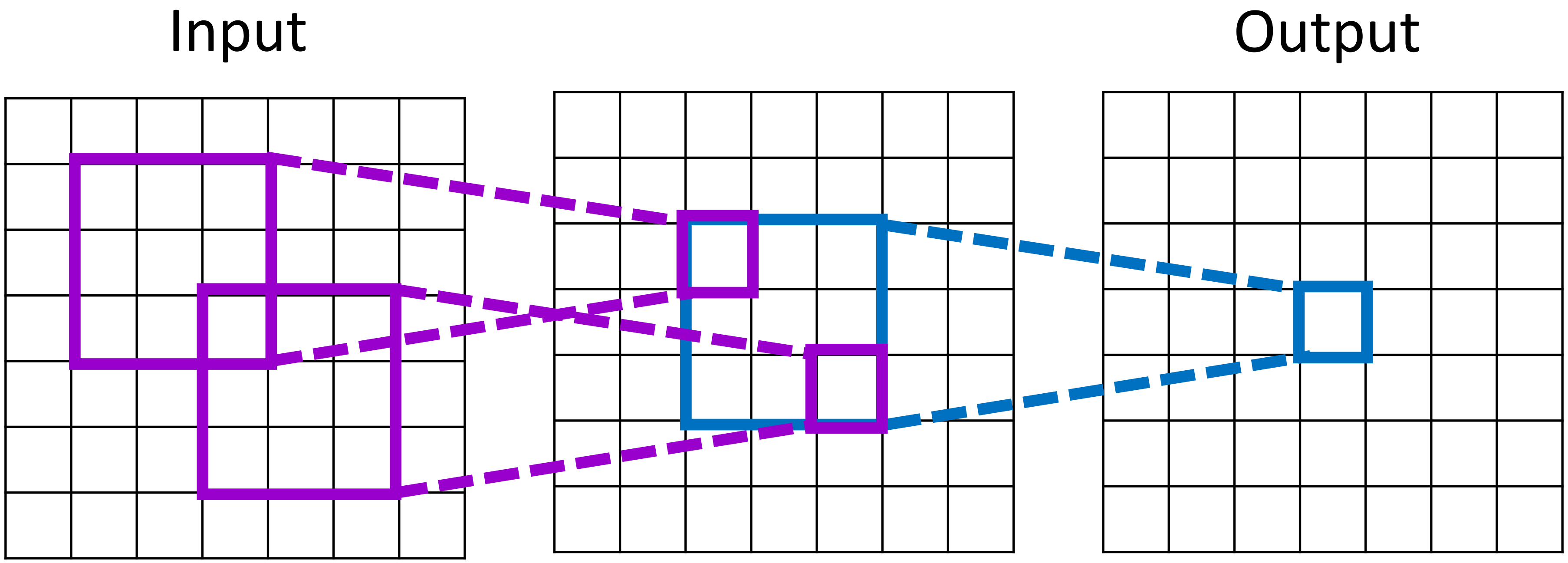
3x3 convolutions, stride 1



Receptive field size: 3

Receptive field

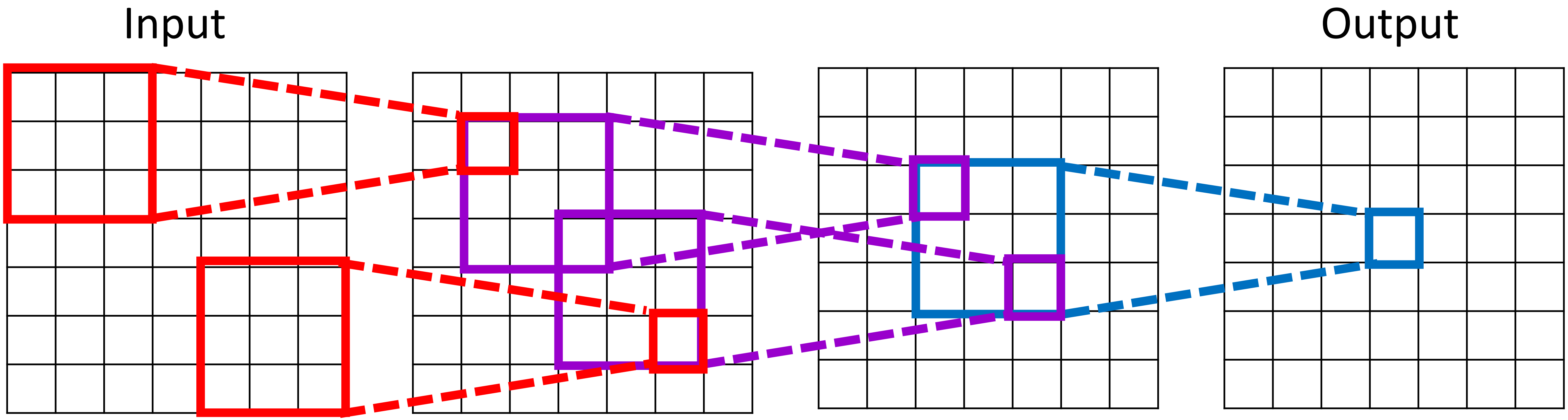
3x3 convolutions, stride 1



Receptive field size: 5

Receptive field

3x3 convolutions, stride 1



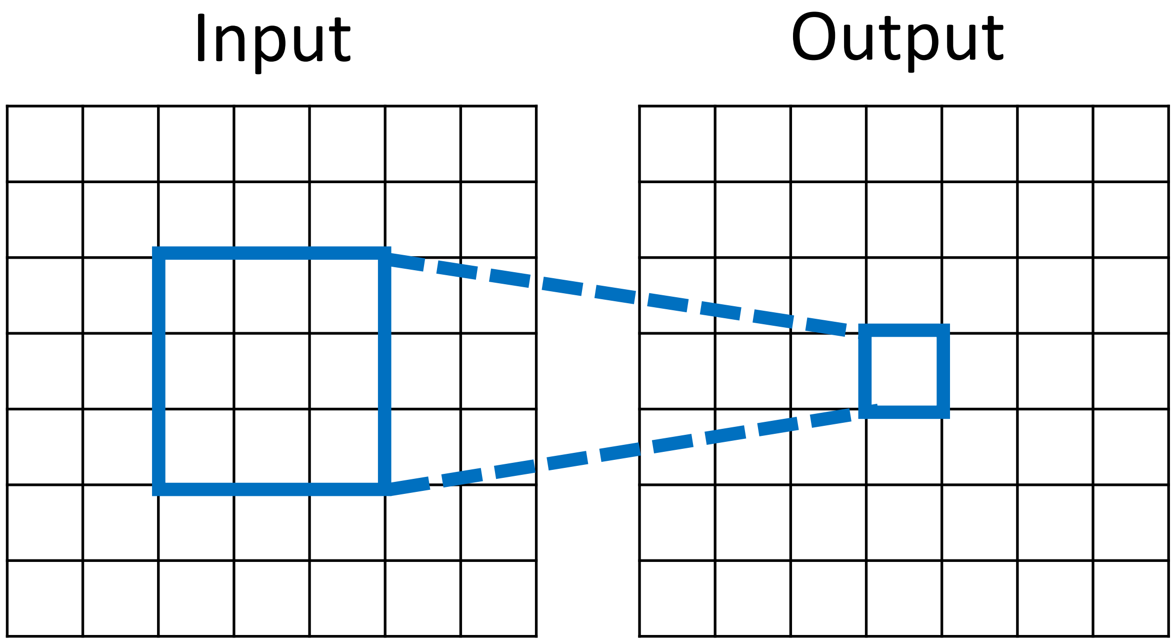
Receptive field size: 7

Each successive convolution adds $F - 1$ to the receptive field size

With L layers the receptive field size is $1 + L * (F - 1)$

Receptive field

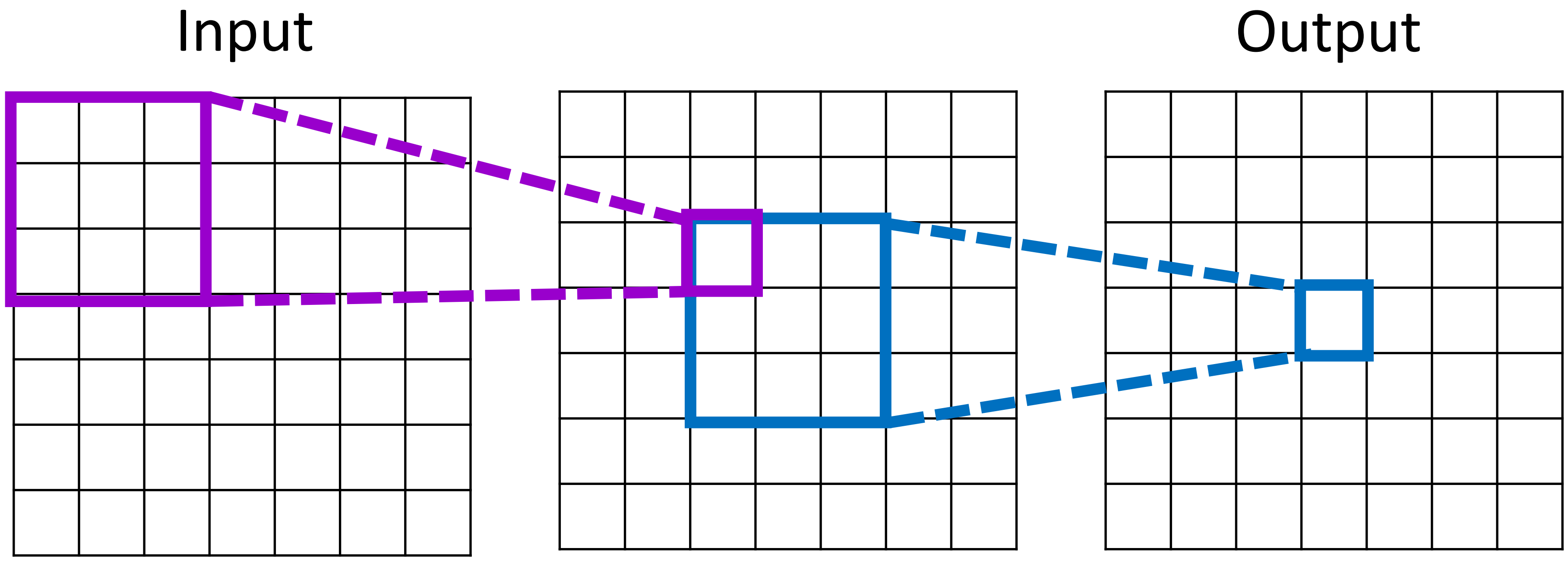
3x3 convolutions, stride 2



Receptive field size: 3

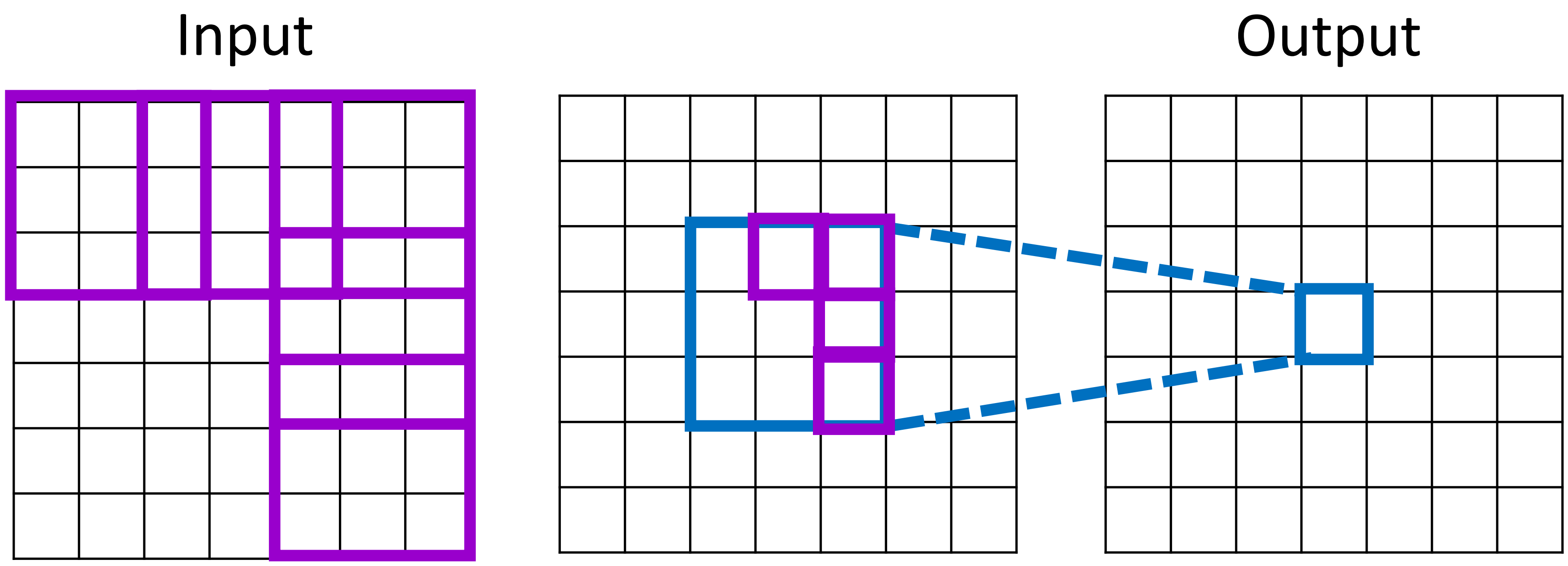
Receptive field

3x3 convolutions, stride 2



Receptive field

3x3 convolutions, stride 2



Receptive field size: 7

With a stride of 2, receptive field size is given by $2^{L+1} - 1$, i.e., it grows exponentially (though spatial resolution decreases exponentially)

Dropout, Overfitting & Normalization

Overfitting in Deep Neural Nets

Deep nets have many non-linear hidden layers.

- Making them very expressive to learn complicated relationships between inputs and outputs
- But with limited training data, many complicated relationships will be the result of training noise
- So they will exist in the training set and not in test set even if drawn from same distribution

Many methods developed to reduce overfitting

- Early stopping with a validation set
- Weight penalties (L1 and L2 regularization)

Batch Normalization

- Training time:
  - Mini-batch of activations for layer to normalize

$$H' = \frac{H - \mu}{\sigma}$$

where

$$\mu = \frac{1}{m}\sum_i H_{i,:} \qquad \sigma = \sqrt{\frac{1}{m}\sum_i (H-\mu)_i^2 + \delta}$$

Vector of mean activations across mini-batch

Vector of SD of each unit across mini-batch

Batch Normalization

Training time:

- Normalization can reduce expressive power
- Instead use:

$$\gamma H' + \beta$$

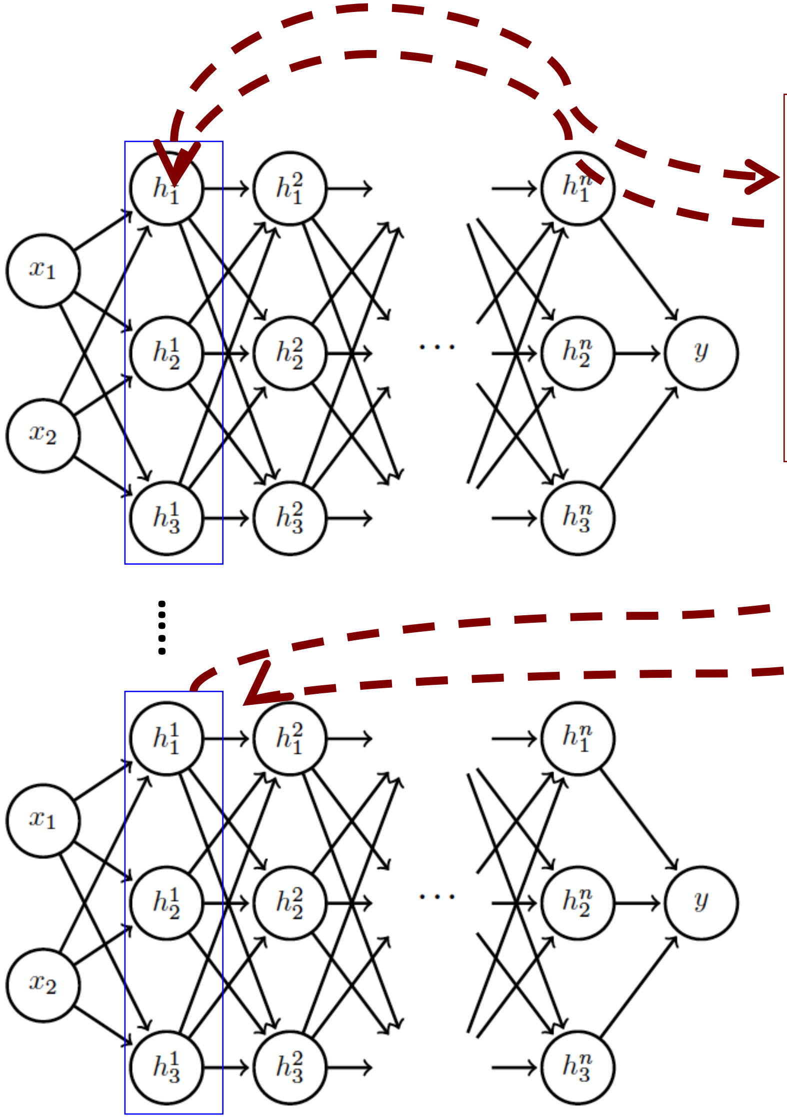
Learnable parameters

Allows network to control range of normalization

Batch Normalization

Batch 1

Batch N



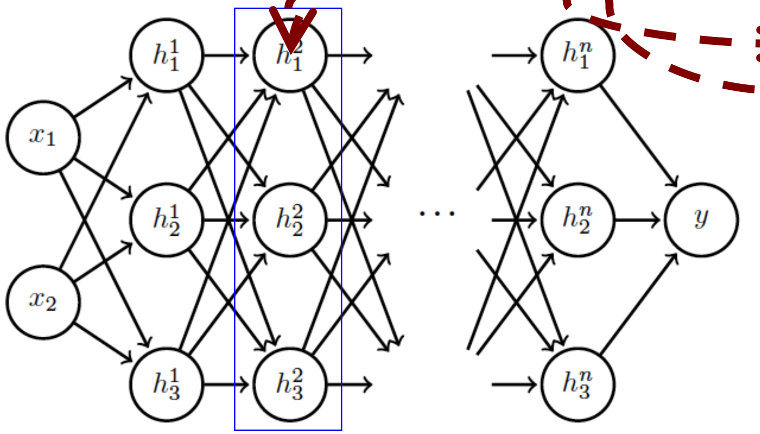
$$\mu^1 = \frac{1}{m}\sum_i H_{i,:}$$

$$\sigma^1 = \sqrt{\frac{1}{m}\sum_i (H-\mu)_i^2 + \delta}$$

Add normalization operations for layer 1

Batch Normalization

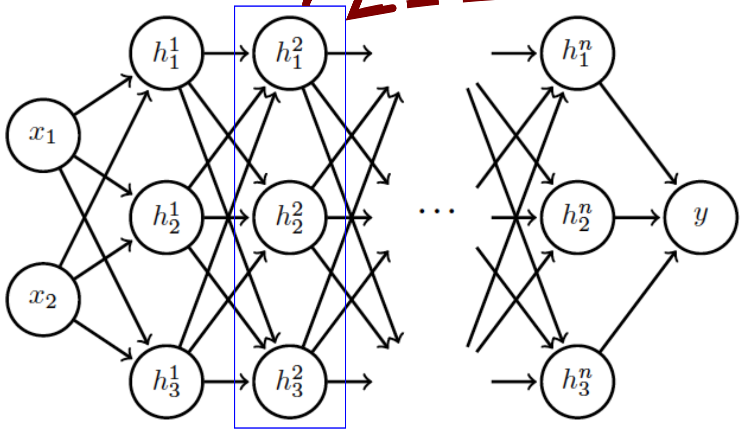
Batch 1



$$\mu^2 = \frac{1}{m}\sum_i H_{i,:}$$

$$\sigma^2 = \sqrt{\frac{1}{m}\sum_i (H-\mu)_i^2 + \delta}$$

Batch *N*



Add normalization operations for layer 2 and so on …

Regularization with unlimited computation

Best way to regularize a fixed size model is:

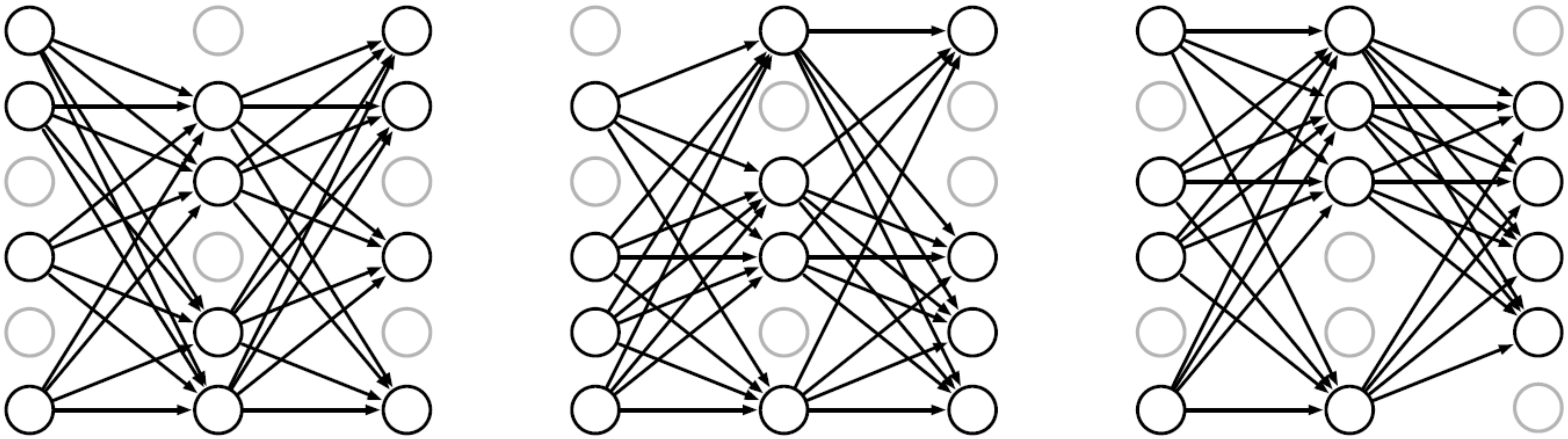
- Average the predictions of all possible settings of the parameters
- Weighting each setting with the posterior probability given the training data (Bayesian approach)

Dropout

Dropout does this using considerably less computation!

By approximating an equally weighted geometric mean of the predictions of an exponential number of learned models that share parameters

Dropout



- For each batch, different random set of nodes is removed
- Their values are set to 0 and their weights are not updated
- 10%, 20% or even 50% of all the nodes

Dropout is a bagging method

Bagging is a method of averaging over several models to improve generalization

Impractical to train many neural networks since it is expensive in time and memory.

Dropout makes it practical to apply bagging to very many large neural networks

It is a method of bagging applied to neural networks

Dropout is an inexpensive but powerful method of regularizing a broad family of models

Shortcuts and Highways

Deep learning: many layers of processing. Error propagation has to travel farther

All parameters in processing change have to be adjusted

Instead of always passing through all layers, add connections from first to last

Jargon alert:

- Shortcuts
- Residual connections
- Skip connections

Shortcuts and Highways