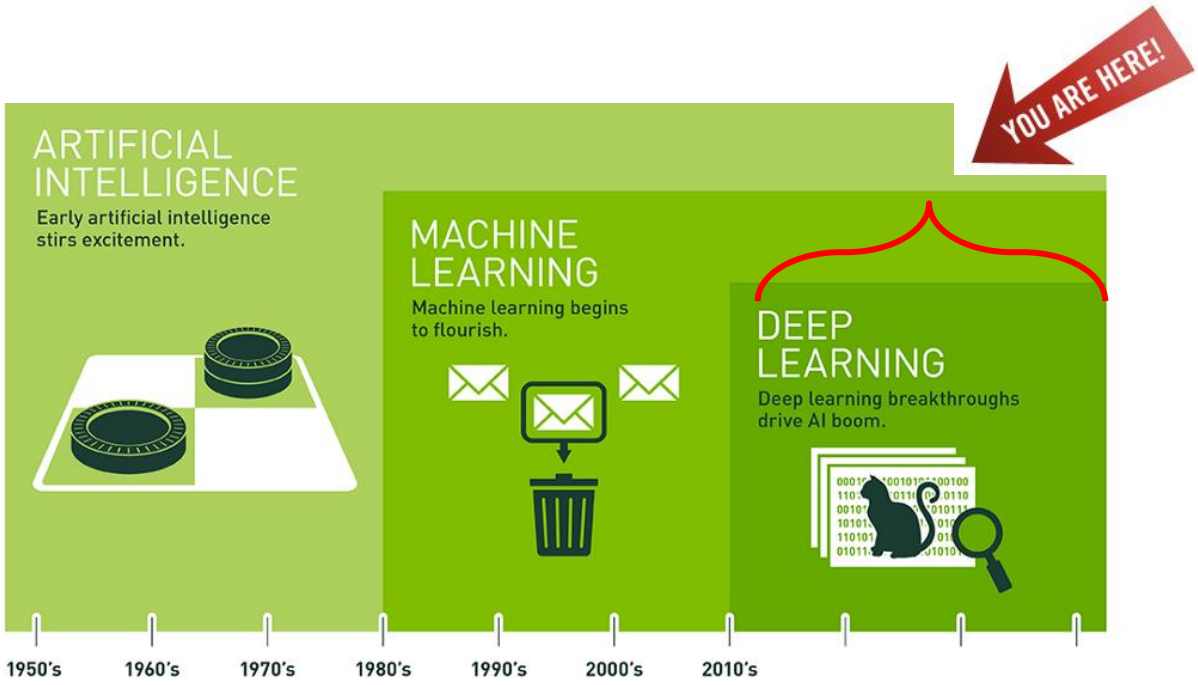
Artificial Intelligence: Deep Learning

many slides from: Y. Bengio, A. Ng and Y. LeCun

Today

1. Motivation
2. Feature Learning
3. Building Block Design of CNN
   - Convolutional Layer
   - Activation Function
   - Pooling Layer
   - Normalization Layer

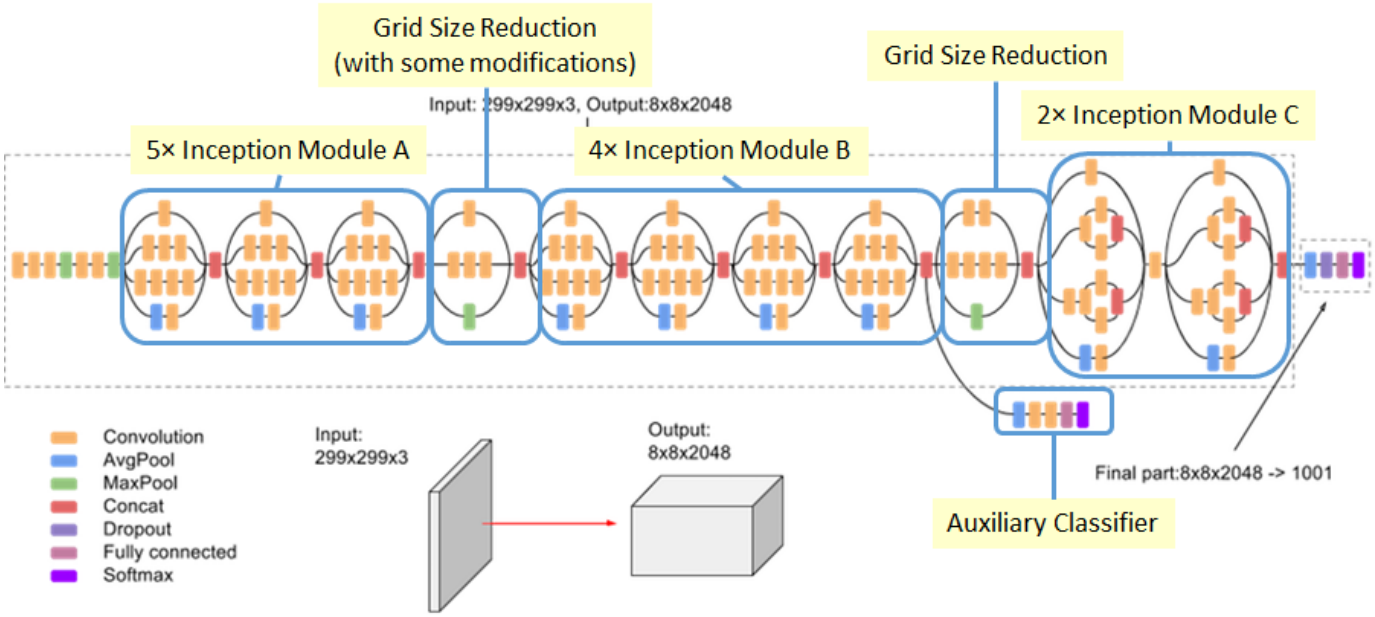
History of AI



https://www.linkedin.com/pulse/goedels-incompleteness-theorem-emergence-ai-eberhard-schoeneburg/

What is Really Deep Learning?

1. Good old Neural Networks, with more layers/modules
2. Non-linear, hierarchical, abstract representations of data
3. Flexible models with any input/output type and size
4. Differentiable Functional Programming



The Google "Inception" deep neural network architecture for image recognition (27 layers)

Why Deep Learning Now?

1. Better algorithms & understanding
2. Computing power (GPUs, TPUs, ...)
3. Data with labels
4. Open-source tools and models

PYTORCH

Microsoft CNTK

Caffe2

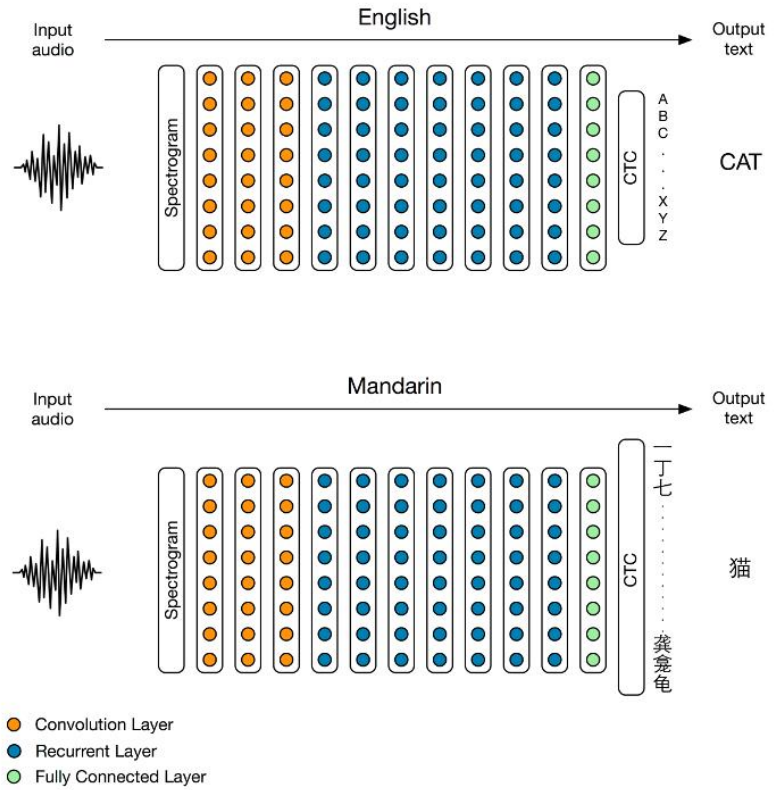
dmlc mxnet

gensim

spaCy

theano

DLToday: Speech-to-Text

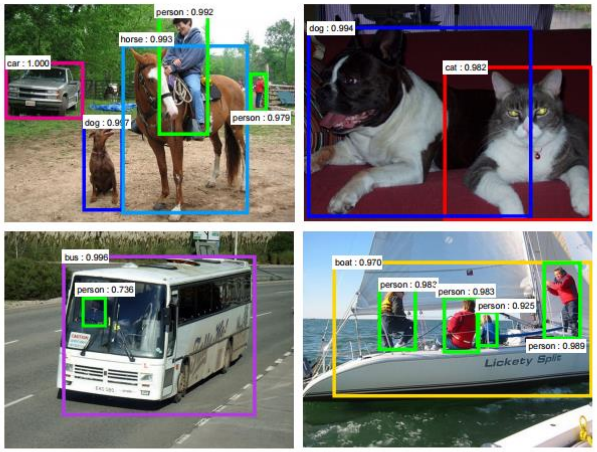


[Baidu 2014]

DLToday: Vision

[Krizhevsky 2012]

[Ciresan et al. 2013]

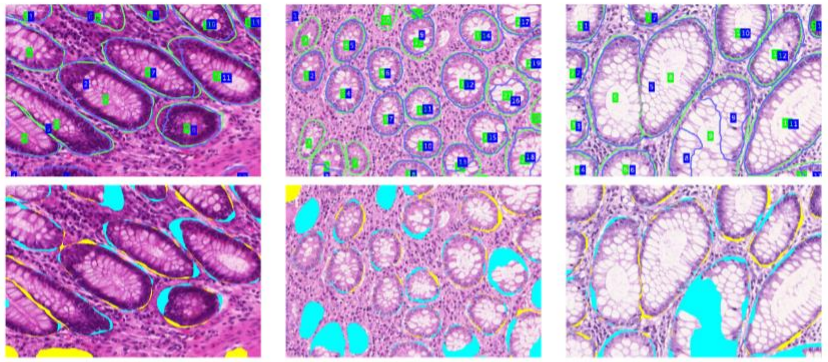


[Faster R-CNN - Ren 2015]

[NVIDIA dev blog]

DLToday: Vision

[Stanford 2017]



[Nvidia Dev Blog 2017]

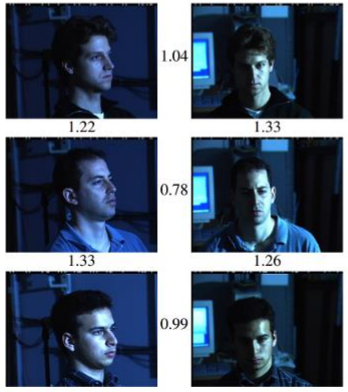
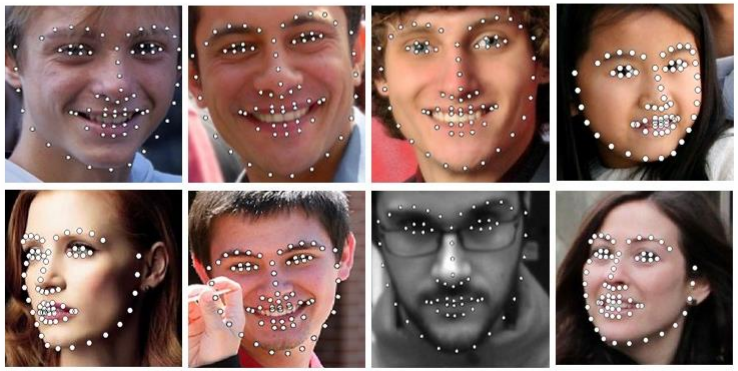


Figure 1. **Illumination and Pose invariance.**

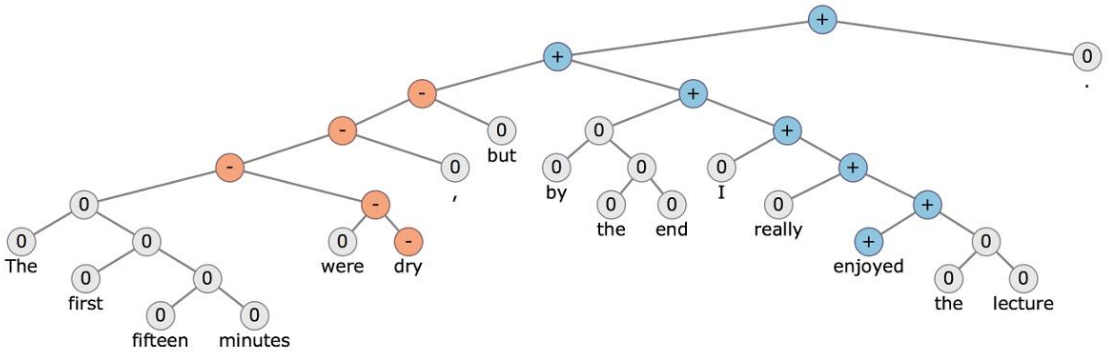
[FaceNet - Google 2015]



[Facial landmark detection CUHK 2014]

DLToday: NLP

[Google Translate System - 2016]



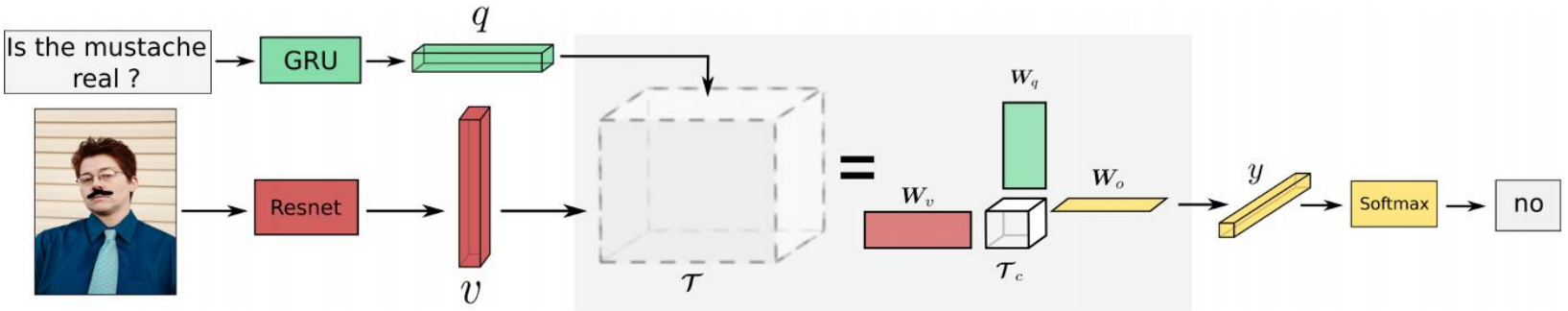
[Socher 2015]

DLToday: NLP

[Google Inbox Smart Reply]

[Amazon Echo / Alexa]

DLToday: Vision+NLP



[VQA - Mutan 2017]

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

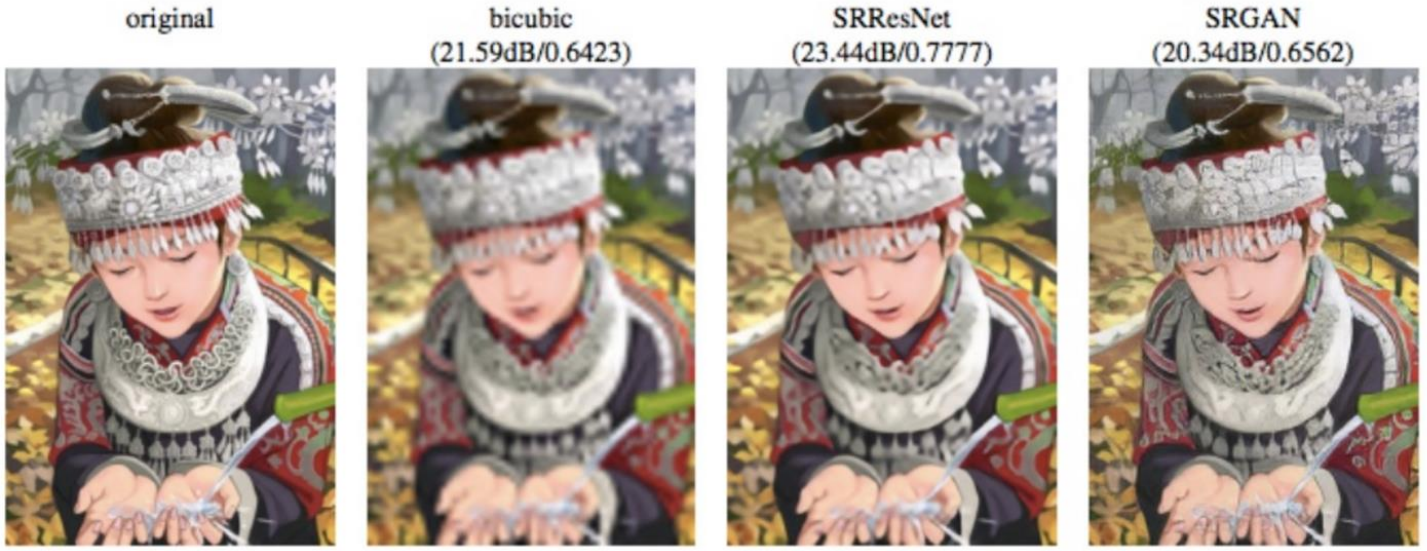
"boy is doing backflip on wakeboard."

[Karpathy 2015]

DLToday: Image Translation

[DeepDream 2015]

[Gatys 2015]



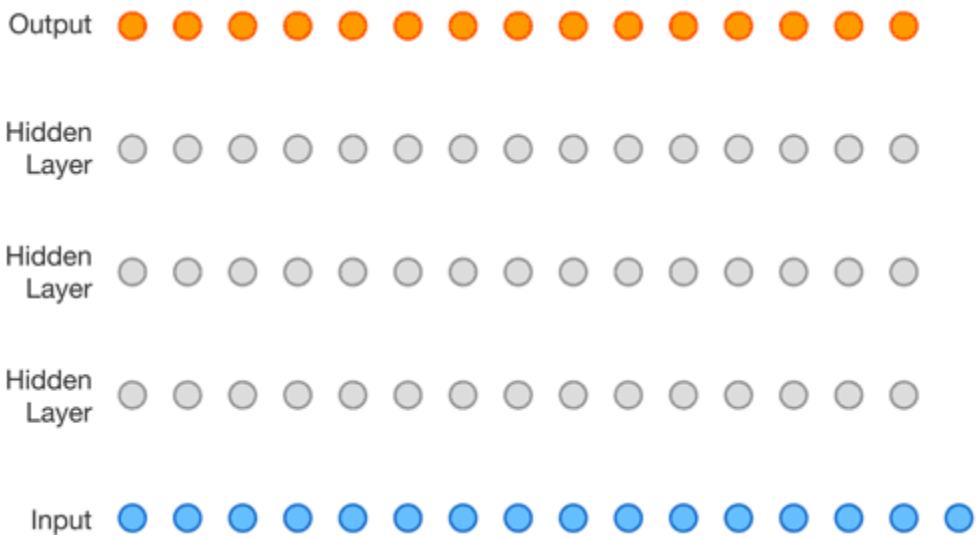
[Ledig 2016]

DLToday: Generative Models

Sampled celebrities [Nvidia 2017]

StackGAN v2 [Zhang 2017]

DLToday: Generative Models



Guess which one is generated?

Tacotron 2 Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions, 2017

DLToday: Language/Image Models

Open-AI GPT-3, or DALL-E: https://openai.com/blog/dall-e/

TEXT PROMPT

an armchair in the shape of an avocado [...]

AI-GENERATED IMAGES

View more or edit prompt

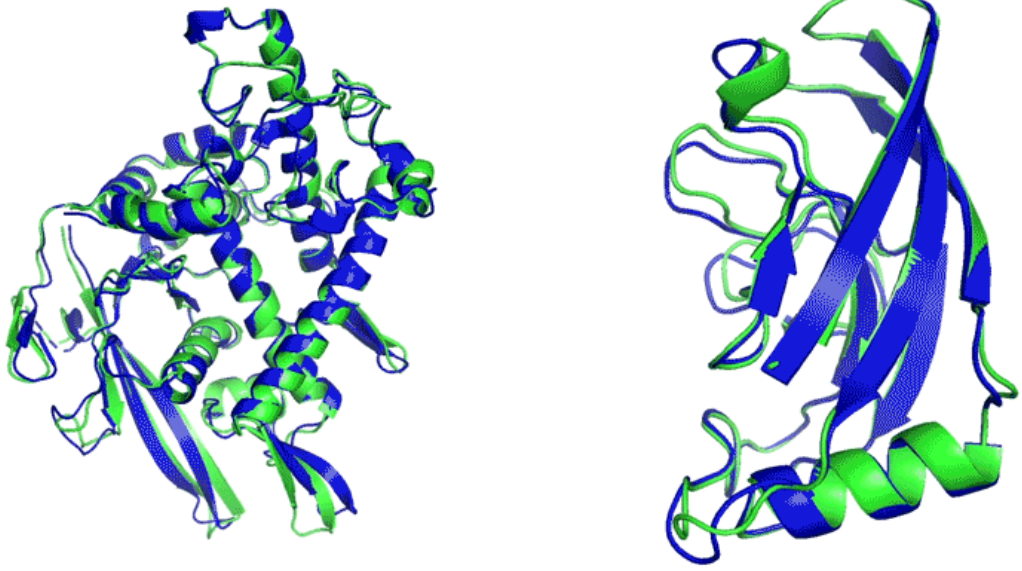
TEXT PROMPT

a store front that has the word 'openai' written on it [...]

AI-GENERATED IMAGES

View more or edit prompt

DLToday: Genomics



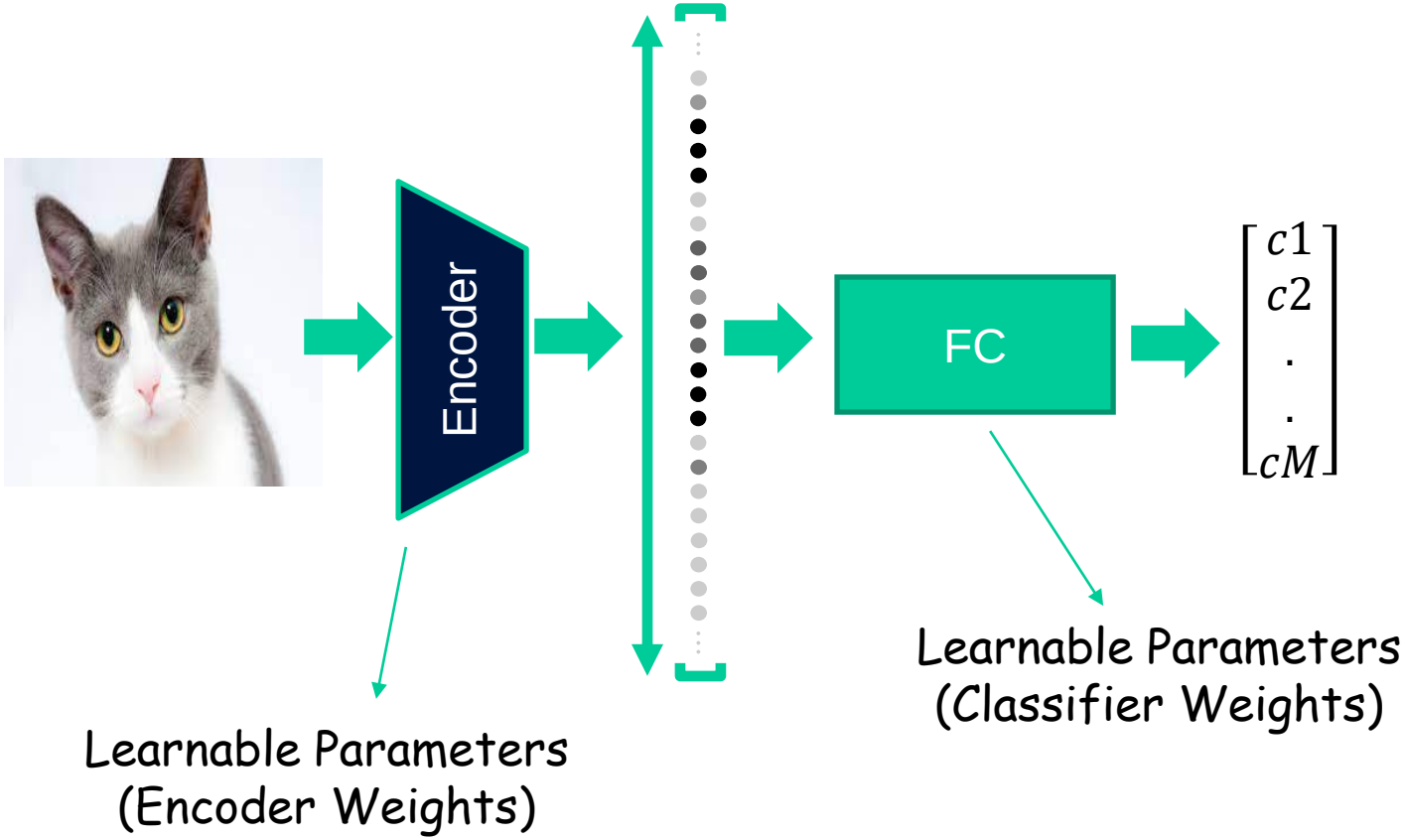
AlphaFold by DeepMind

Today

1. Motivation
2. Feature Learning 

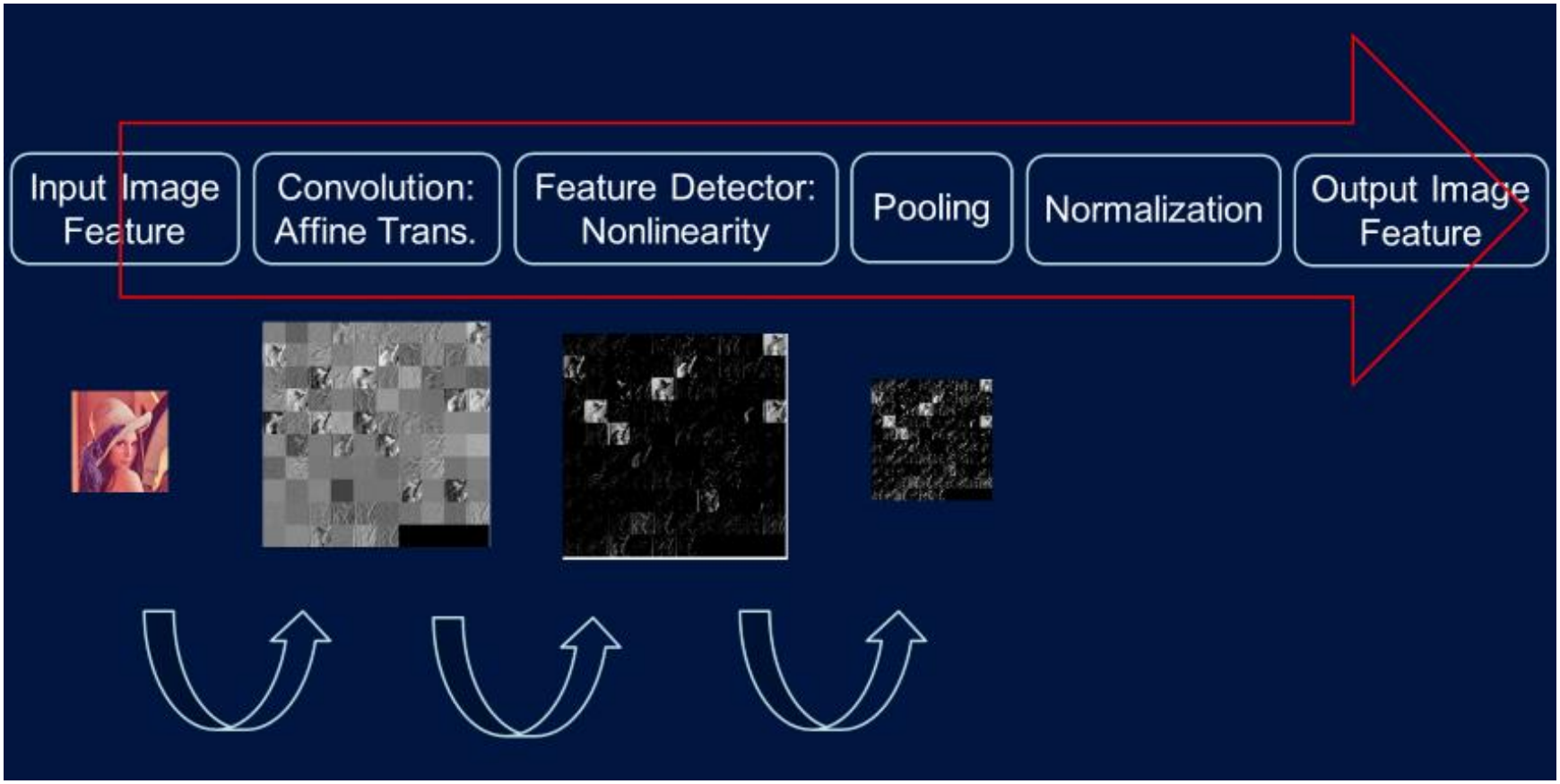
3. Building Block Design of CNN
    - Convolutional Layer
    - Activation Function
    - Pooling Layer
    - Normalization Layer

Feature (Encoder) Learning

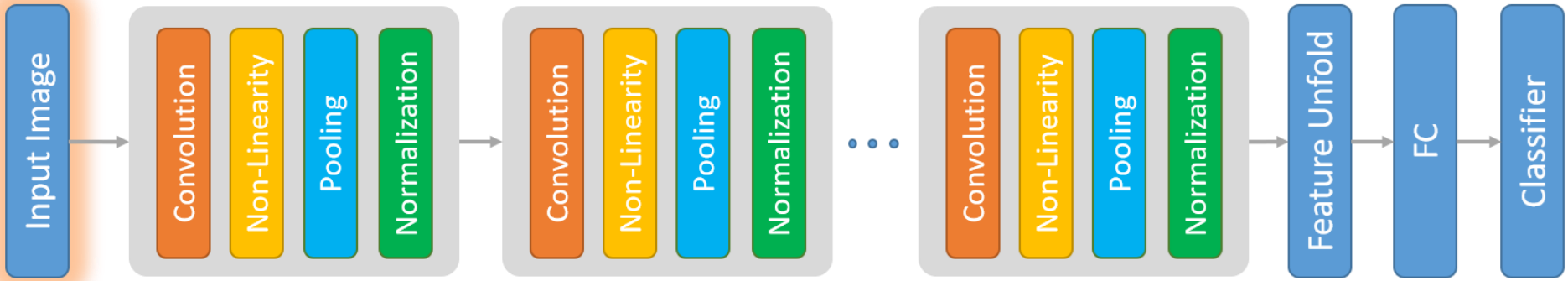


Encoder: Convolution Layer

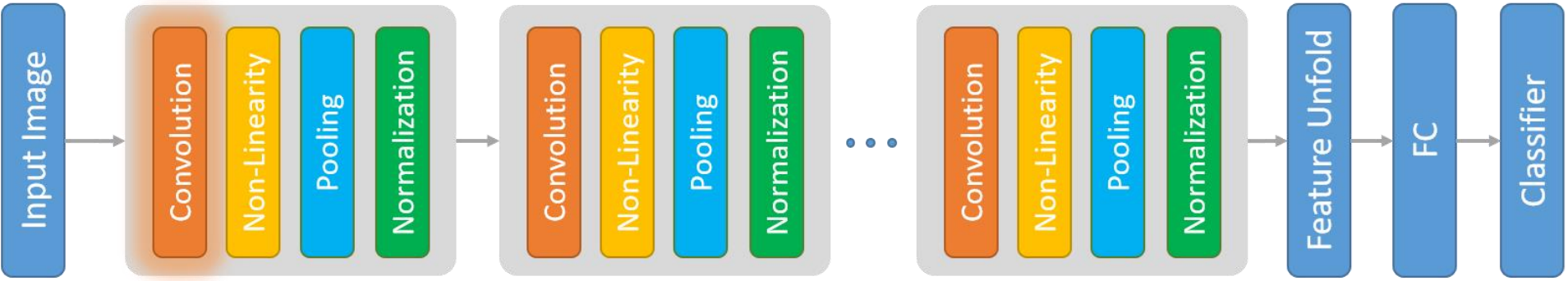
Higher level of feature abstraction

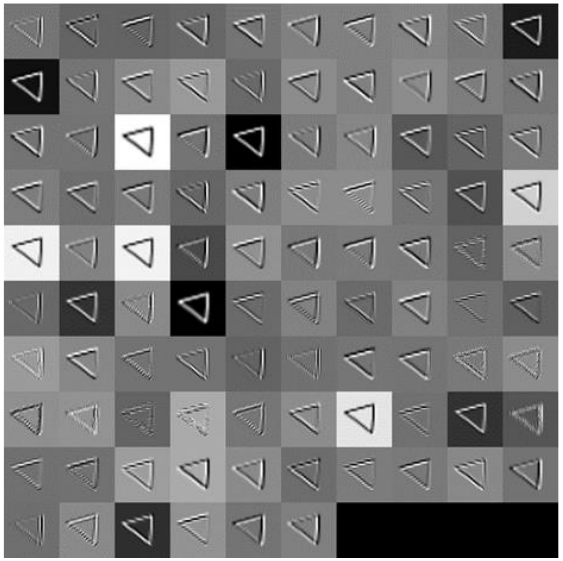


Encoder: Cascade Layer Design

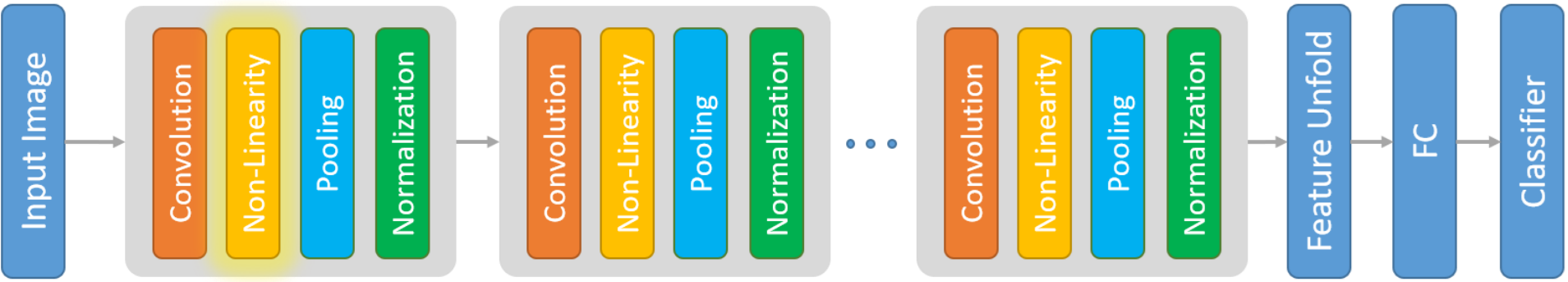


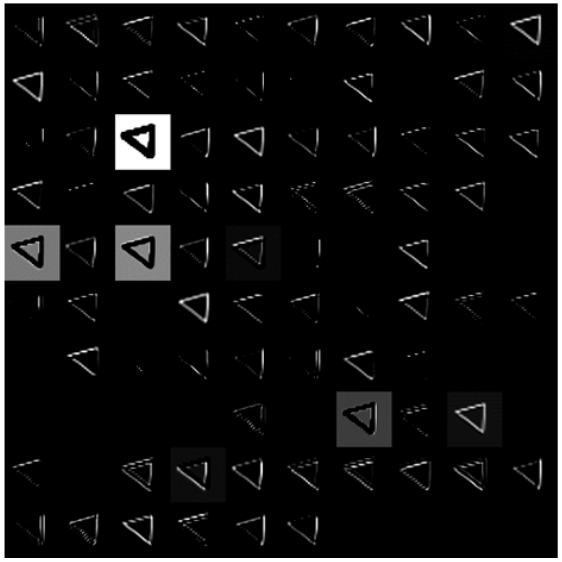
Encoder: Cascade Layer Design



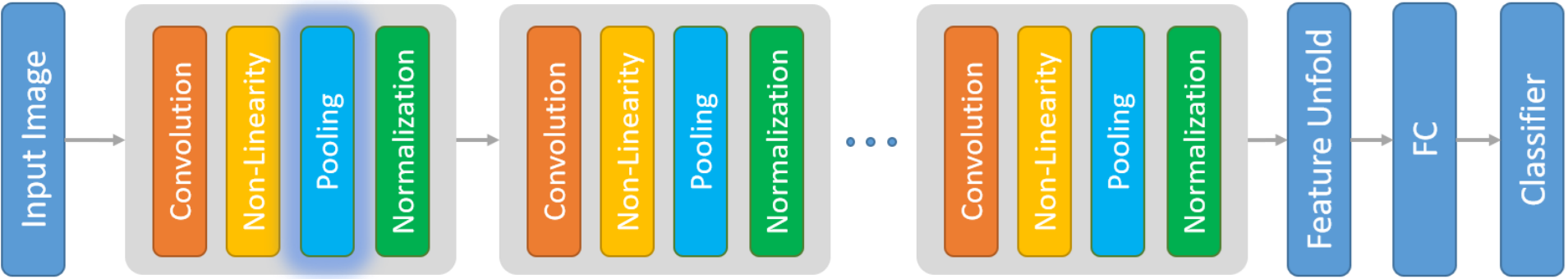


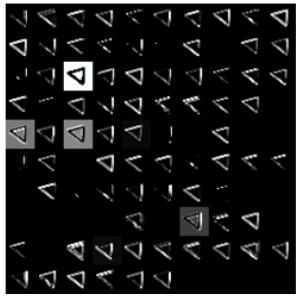
Encoder: Cascade Layer Design



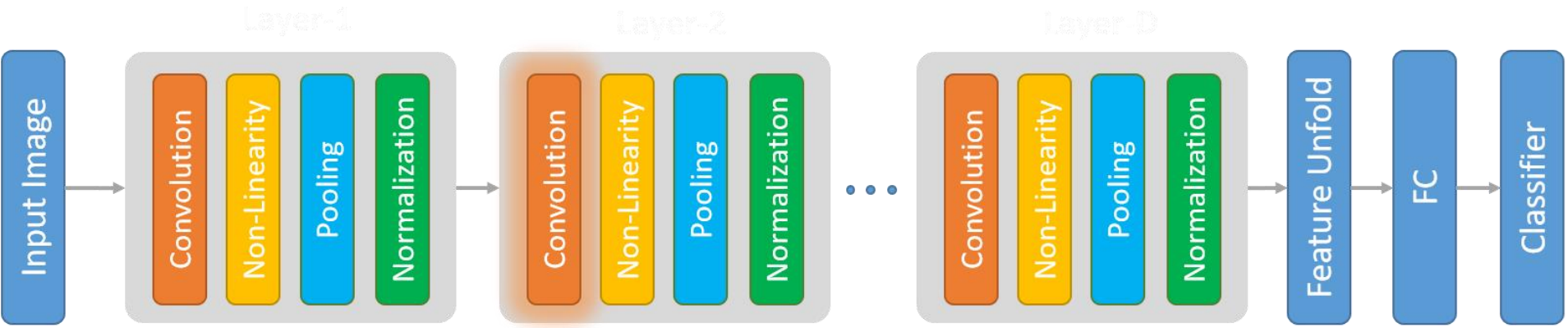


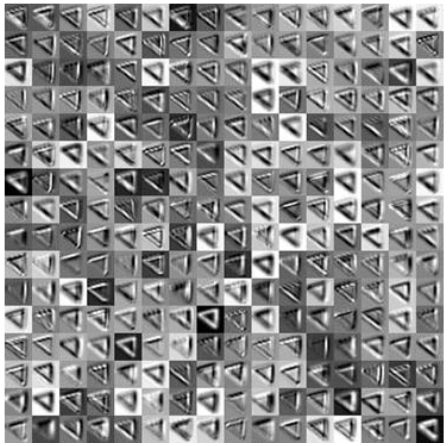
Encoder: Cascade Layer Design



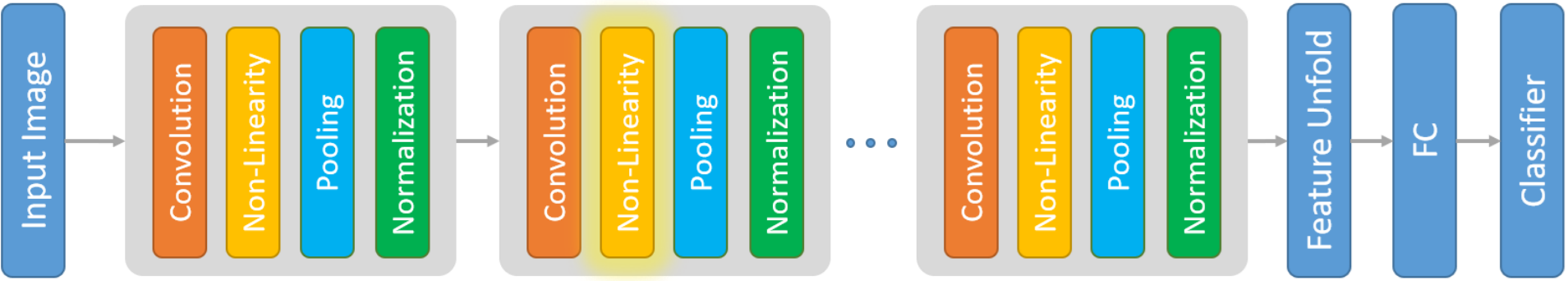


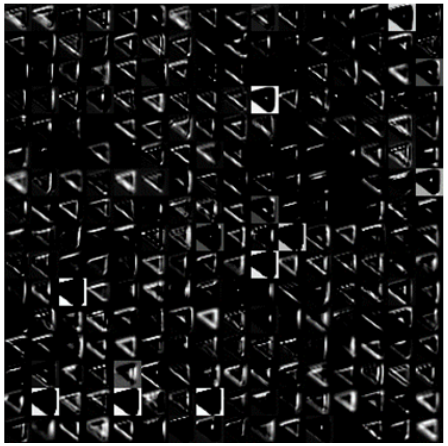
Encoder: Cascade Layer Design



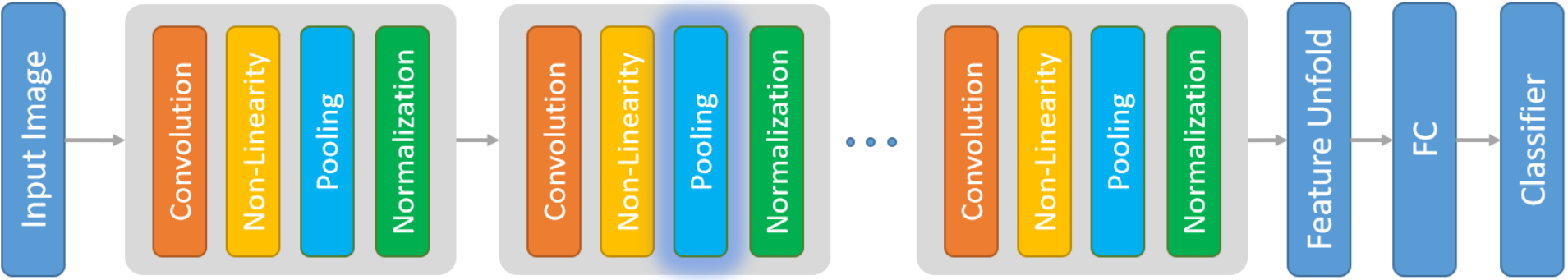


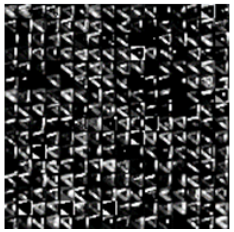
Encoder: Cascade Layer Design



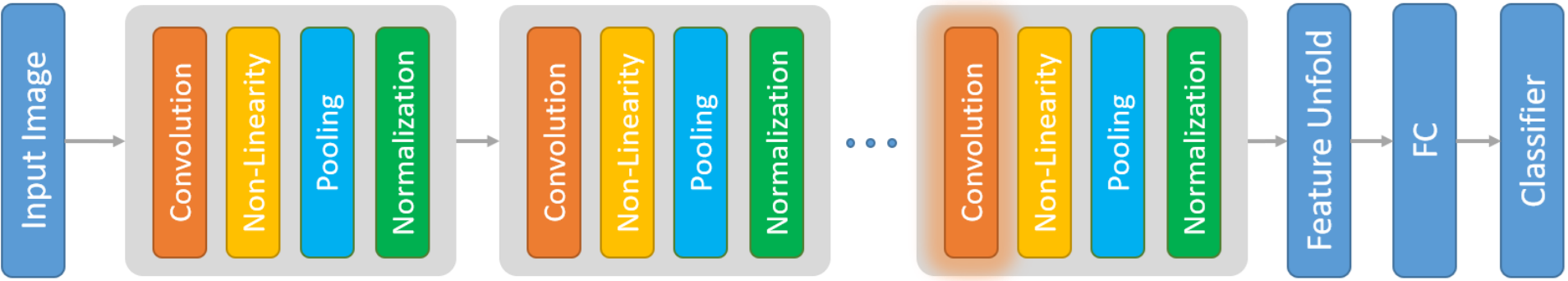


Encoder: Cascade Layer Design

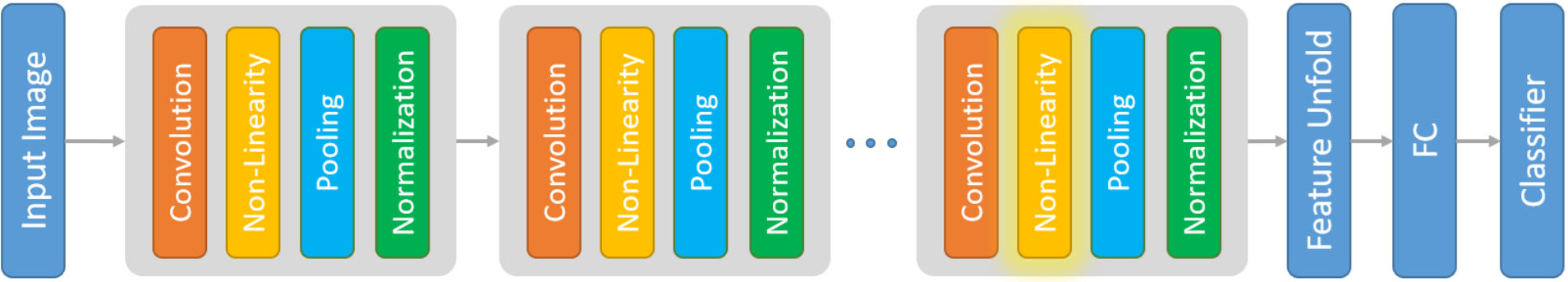


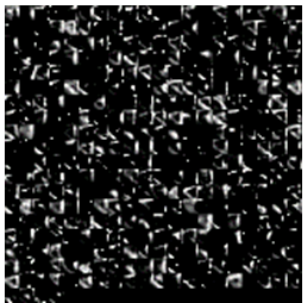


Encoder: Cascade Layer Design

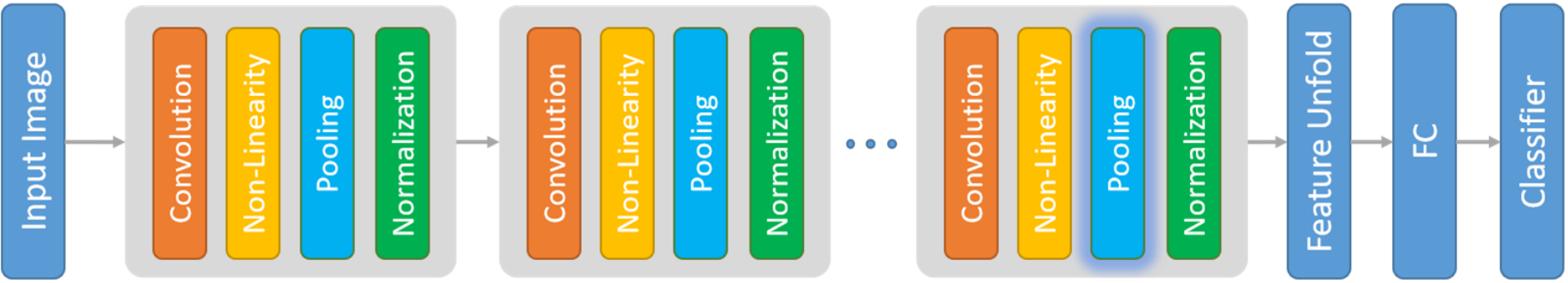


Encoder: Cascade Layer Design

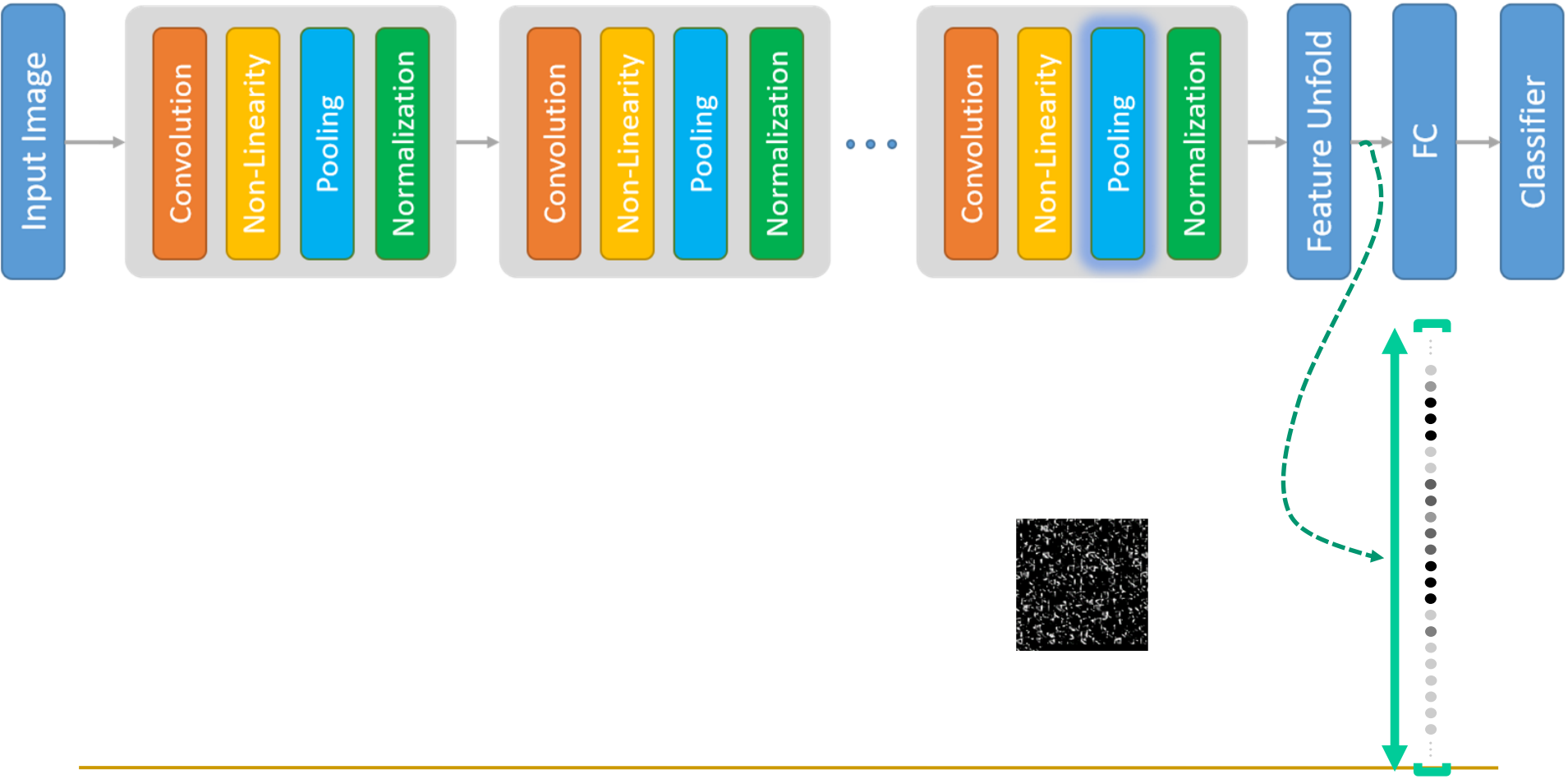




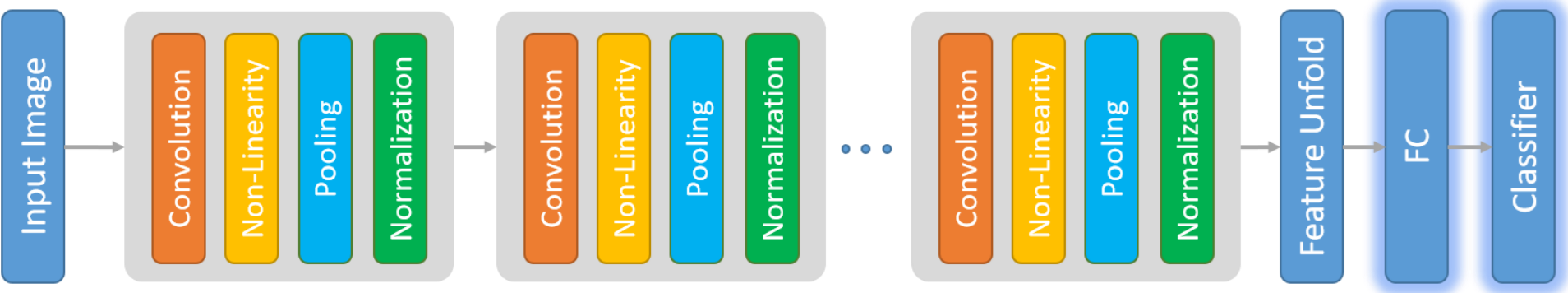
Encoder: Cascade Layer Design

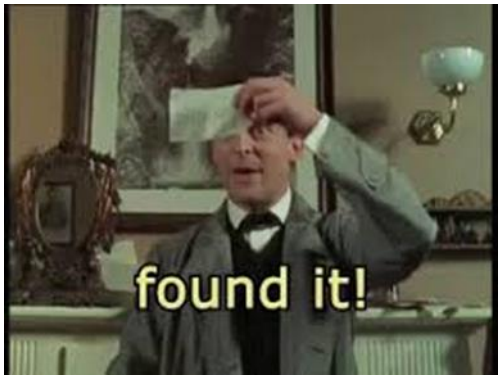


Encoder: Cascade Layer Design



Encoder: Cascade Layer Design

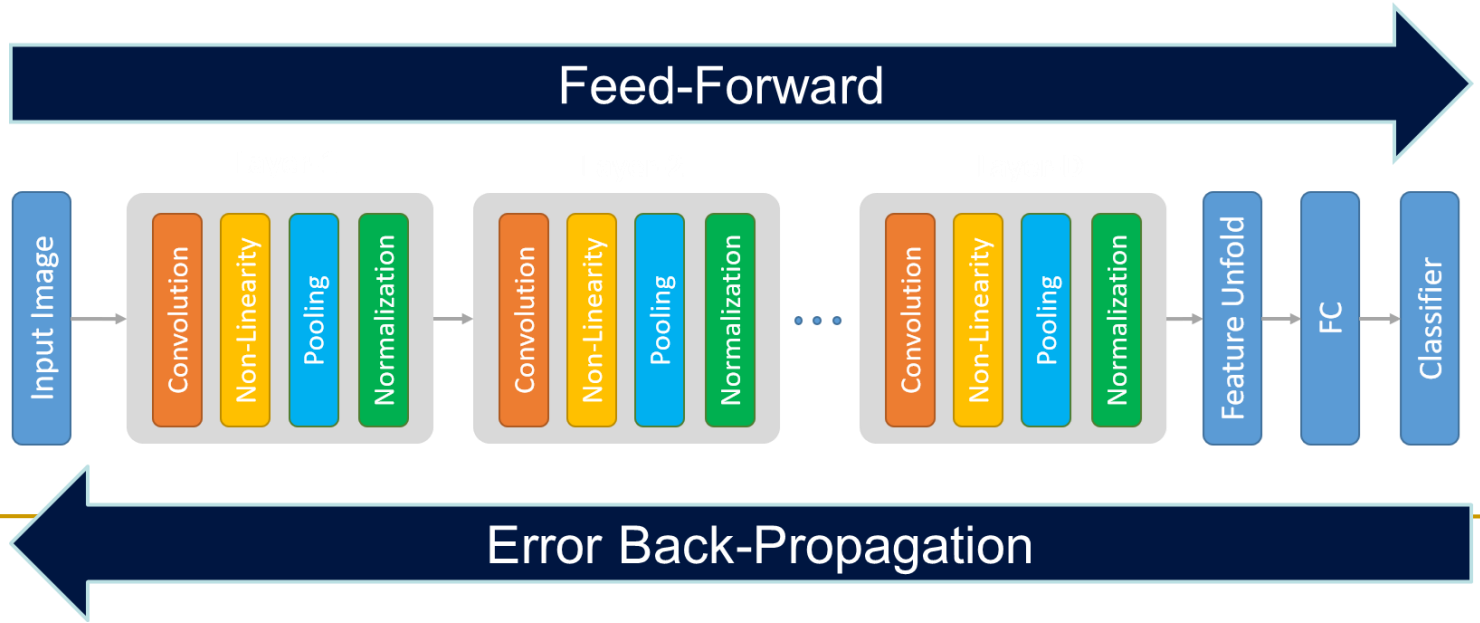




Y1: Square
Y2: Triangle
Y3: Circle
.
.
.
yN: Dimond

Encoder: Overall Training

1. Initialize all learnable parameters (e.g. random sampling)
2. Preprocess train image dataset and randomly shuffle them
3. Feed-forward images in mini-batches and compute "Target Predictions"
4. Calculate Error = Abs("Target Labels" – "Target Predictions")
5. Propagate back error gradients and adjust weights using an optimization method (e.g. stochastic gradient descent)
6. Repeat (1)-(4) to converge to a minimal error



Initial Drawbacks

1. Standard backpropagation with sigmoid activation function does not scale well with multiple layers
   - Weight of early layers change too slowly (no learning)
2. Overfitting
   - Large network -> lots of parameters -> increased capacity to "learn by heart"
3. Multilayered ANNs need lots of labeled data
   - Most data is not labeled 💻☎

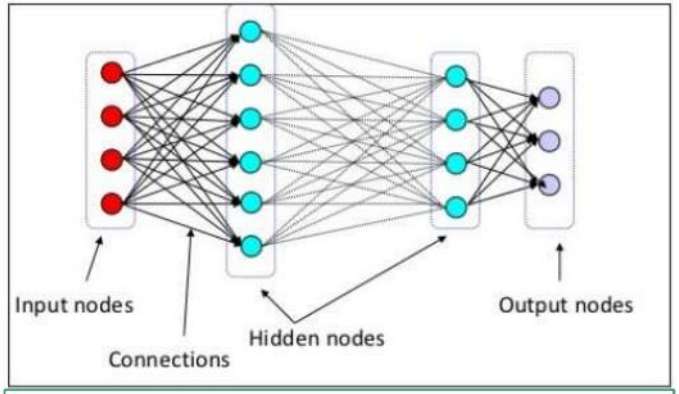
Initial Drawbacks (1)

1. Standard gradient-based backpropagation does not scale well with multiple layers…

When we multiply the gradients many times (for each layer), it can lead to …

a) Vanishing gradient problem:
   - gradients shrink exponentially with the number of layers
   - so weight updates get smaller and smaller
   - and weights of early layers change very slowly and network learns very very slowly

b) Exploding gradient problem:
   - multiplying gradients could also make them grow exponentially.
   - so weight updates get larger and larger
   - and the weights can become so large as to overflow and result in NaN values



$$\delta_h = g'(x_h) \times Err_h$$
$$= O_h\,(1 - O_h) \times \sum_k (w_{hk}\,\delta_k)$$

$$\delta_6 = O_6\,(1 - O_6) \times \sum\ (w_{6,7}\,\delta_7)$$

$$\delta_5 = O_5\,(1 - O_5) \times \sum\ (w_{5,6}\,\delta_6)$$

$$\delta_4 = O_4\,(1 - O_4) \times \sum\ (w_{4,5}\,\delta_5)$$

$$\delta_3 = O_3\,(1 - O_3) \times \sum\ (w_{3,4}\,\delta_4)$$

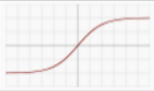
…

Initial Drawbacks (1)

To help, we can :

a) Use other activation functions…

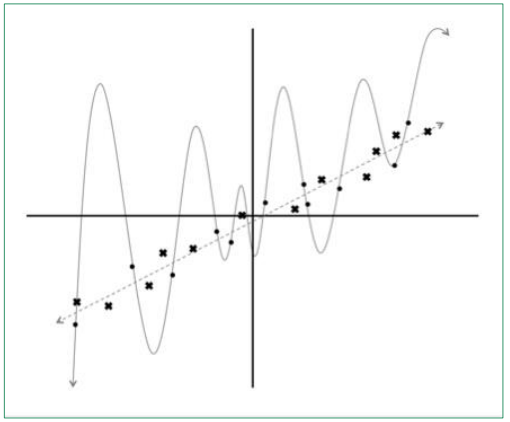
a) Do "gradient clipping" (i.e. set bounds on the gradients)

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

https://medium.com/towards-data-science/activation-functions-and-its-types-which-is-better-a9a5310cc8f

https://medium.com/towards-data-science/activation-functions-neural-networks-1cbd9f8d91d6

Initial Drawbacks (2)

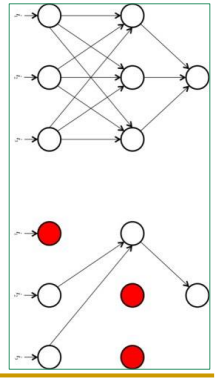


2. Overfitting
   - Large network -> lots of parameters -> increased capacity to "learn by heart"
   - *Solutions:*
     - *Regularization:*
       - modify the error function that we minimize to penalize large weights.

$$\frac{1}{2}\sum_{i=0}^{n}\frac{(T_i - O_i)^2}{n} + \lambda f(w)$$

       - where *f(w)* grows larger as the weights grow larger and λ is the regularization strength

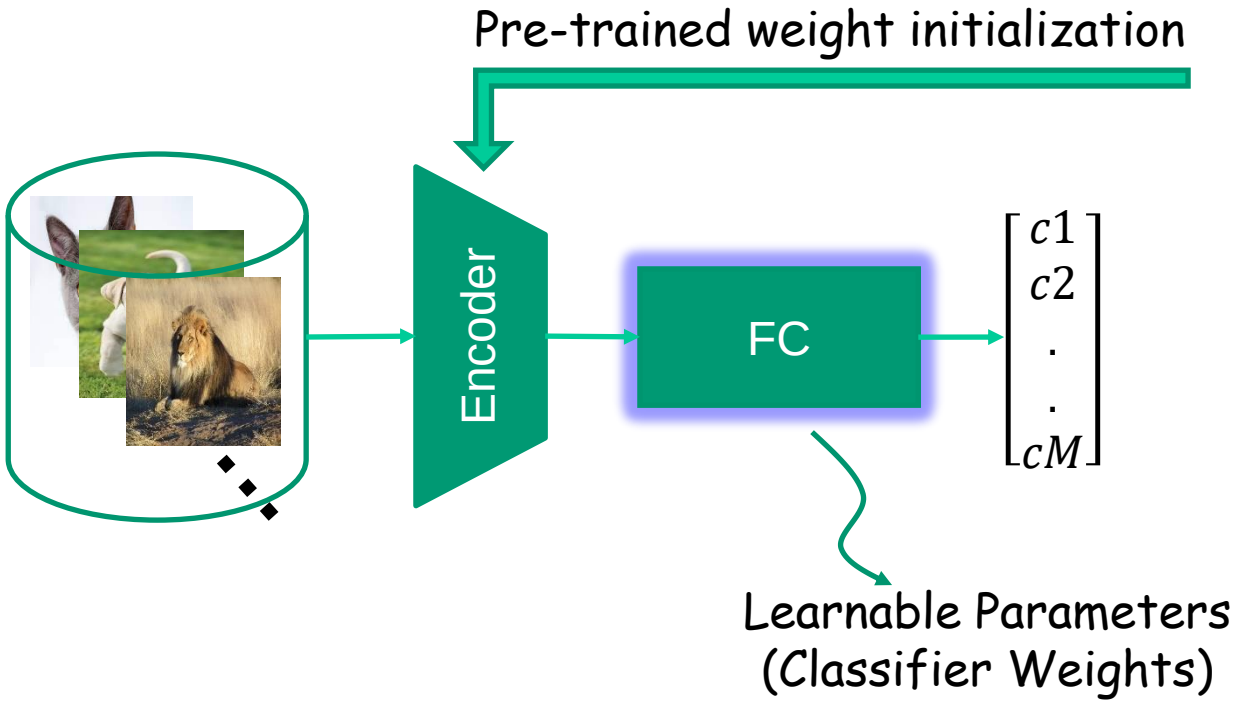
     - *Dropout:*
       - keep a neuron active with some probability *p* or setting it to zero otherwise.
       - prevents the network from becoming too dependent on any one neuron.



https://www.kdnuggets.com/2015/04/preventing-overfitting-neural-networks.html

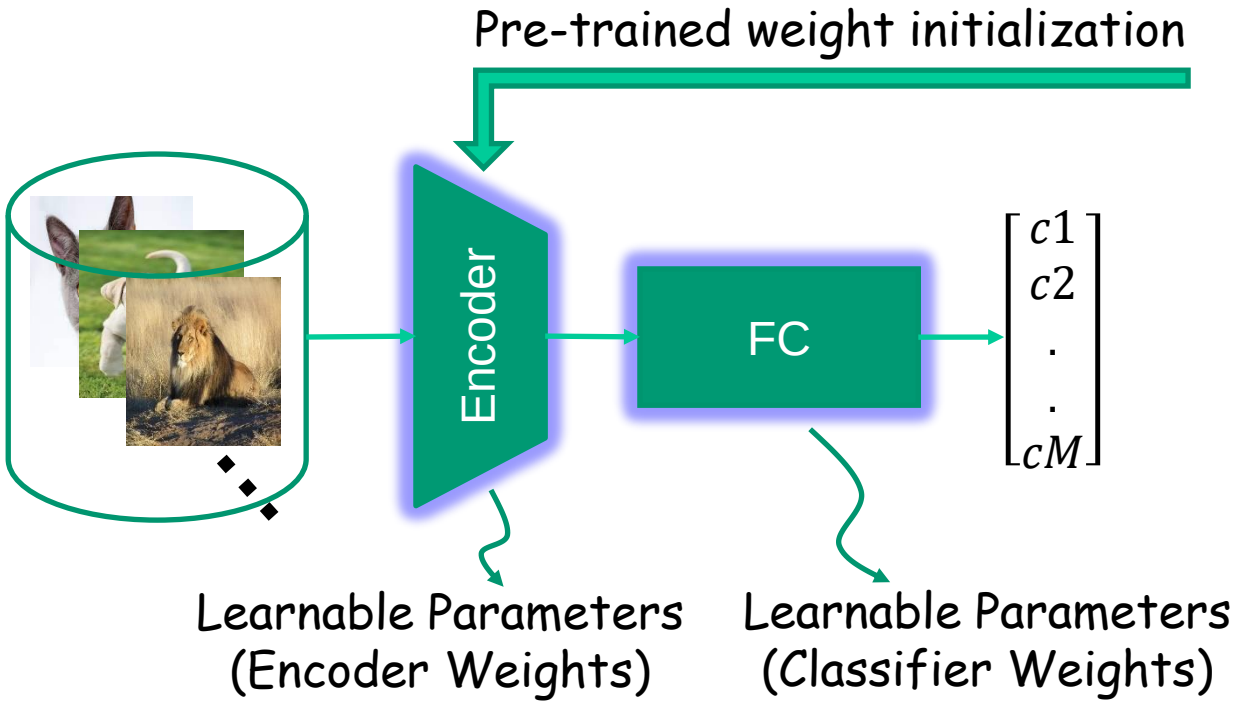
Initial Drawbacks (3)

3. Multilayered ANNs need lots of labeled data for training. To solve the problem:
   - **Transfer Learning:** use pre-trained encoder weights
     - Fine-Tuning



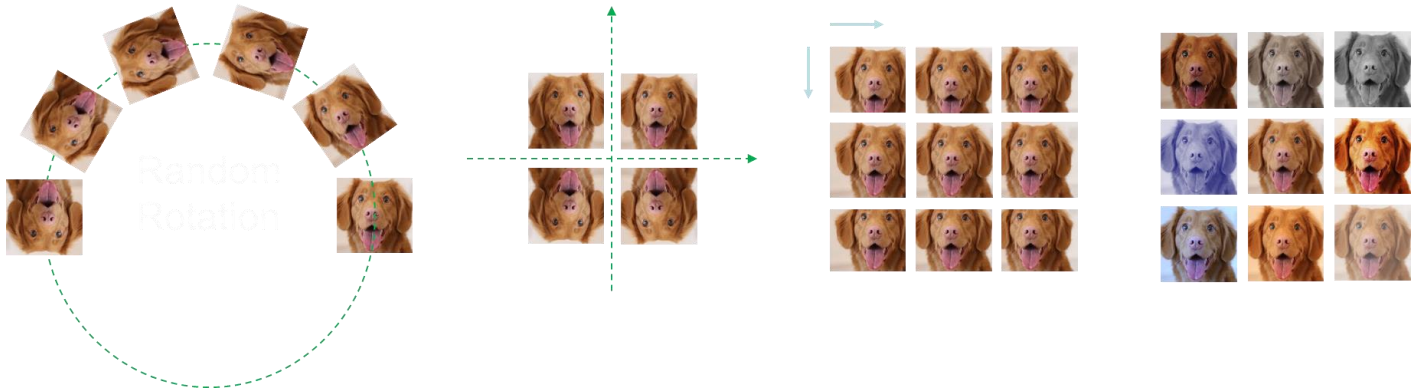
Initial Drawbacks (3)

3. Multilayered ANNs need lots of labeled data for training. To solve the problem:
   - **Transfer Learning:** use pre-trained encoder weights
     - Deep-Tuning



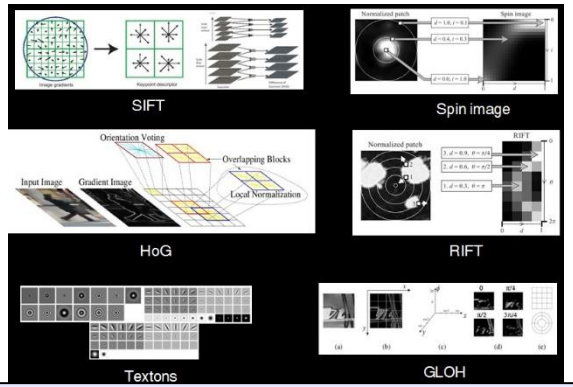
Initial Drawbacks (3)

3. Multilayered ANNs need lots of labeled data for training. To solve the problem:
   - **Image Augmentation:** randomly perturb representation
     - Shifting
     - Flipping
     - Rotation
     - Skewing
     - Color/illumination perturbation



Classic ML

Input

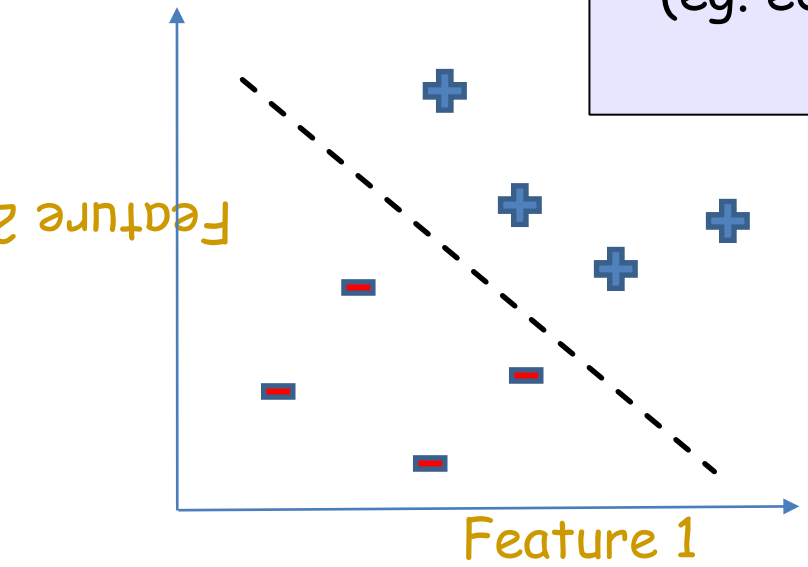


Manual Extraction of Features (eg. edge detection, colors, texture,…)

Learning algorithm

Classic ML, requires labeled data and hand-crafted features

1. Needs expert knowledge
2. Time-consuming and expensive
3. Does not generalize to other domains



Motorbikes
"Non"-Motorbikes

Slide from Y. LeCun

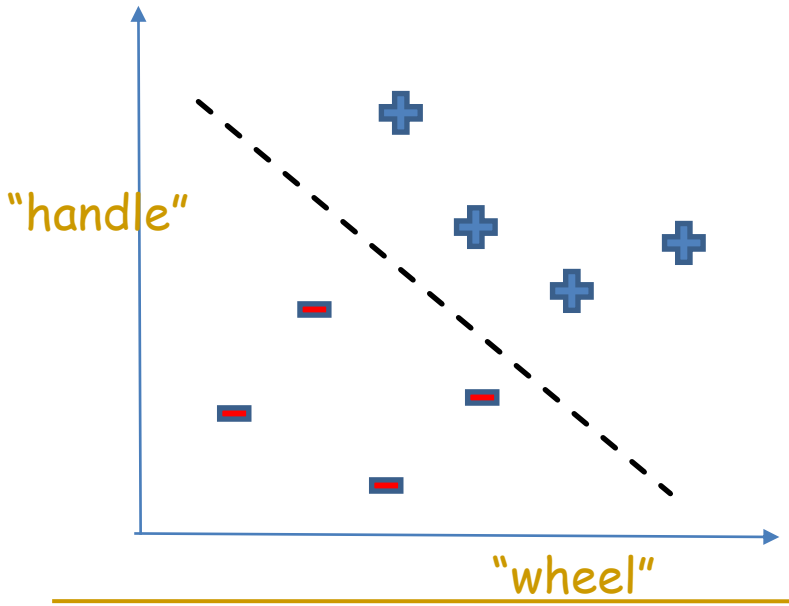
Automatic Feature Learning

Input

Automatic Feature Representation

eg. handle, wheel, …

Learning algorithm



Motorbikes

"Non"-Motorbikes

With Automatic Feature Learning:

1. We feed the network the raw data (not feature-curated)

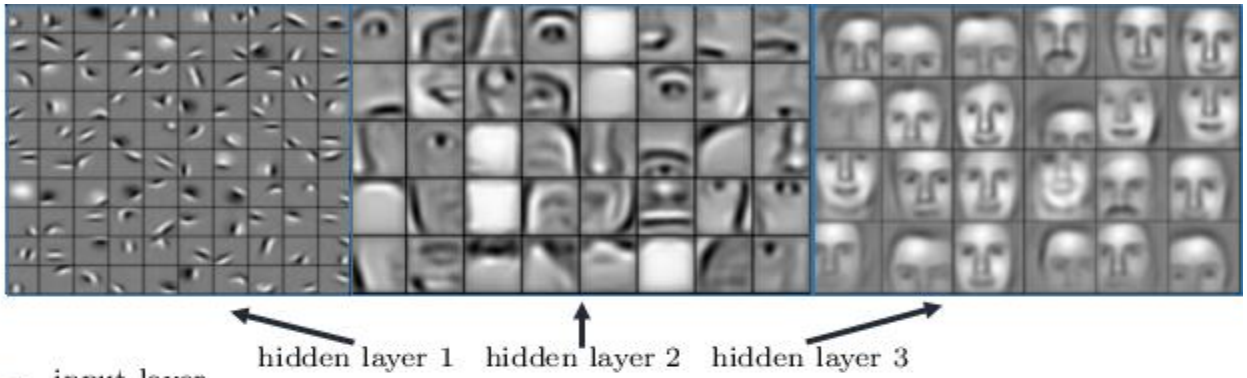
2. The features are learned by the network

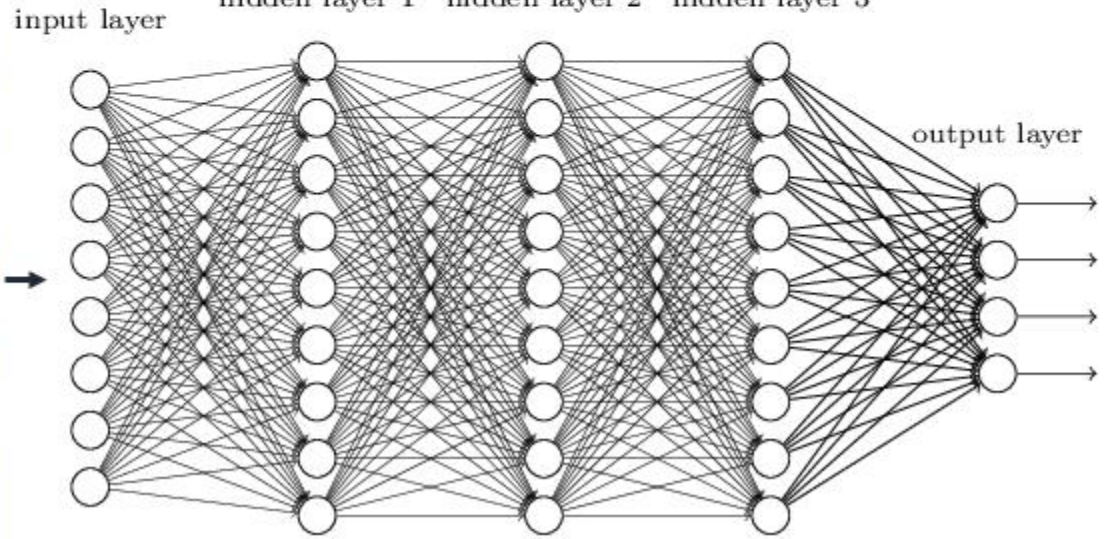
3. Features learned can be re-used in similar tasks.

Slide from Y. LeCun

Automatic Feature Learning

Deep neural networks learn hierarchical feature representations





https://www.strong.io/blog-images/movie-posters/Slide6.png

Automatic Feature Learning

Deep Learning = Machine learning algorithms based on learning multiple levels of representation / abstraction.
– Y. Bengio

- Each layer learns more abstract features that are then combined / composed into higher-level features automatically
- Like the human brain …
  - has many layers of neurons which act as feature detectors
  - detecting more and more abstract features as you go up
- E.g. to classify an image of a cat:
  - Bottom Layers: Edge detectors, curves, corners straight lines
  - Middle Layers: Fur patterns, eyes, ears
  - Higher Layers: Body, head, legs
  - Top Layer: Cat or Dog

Automatic Feature Learning

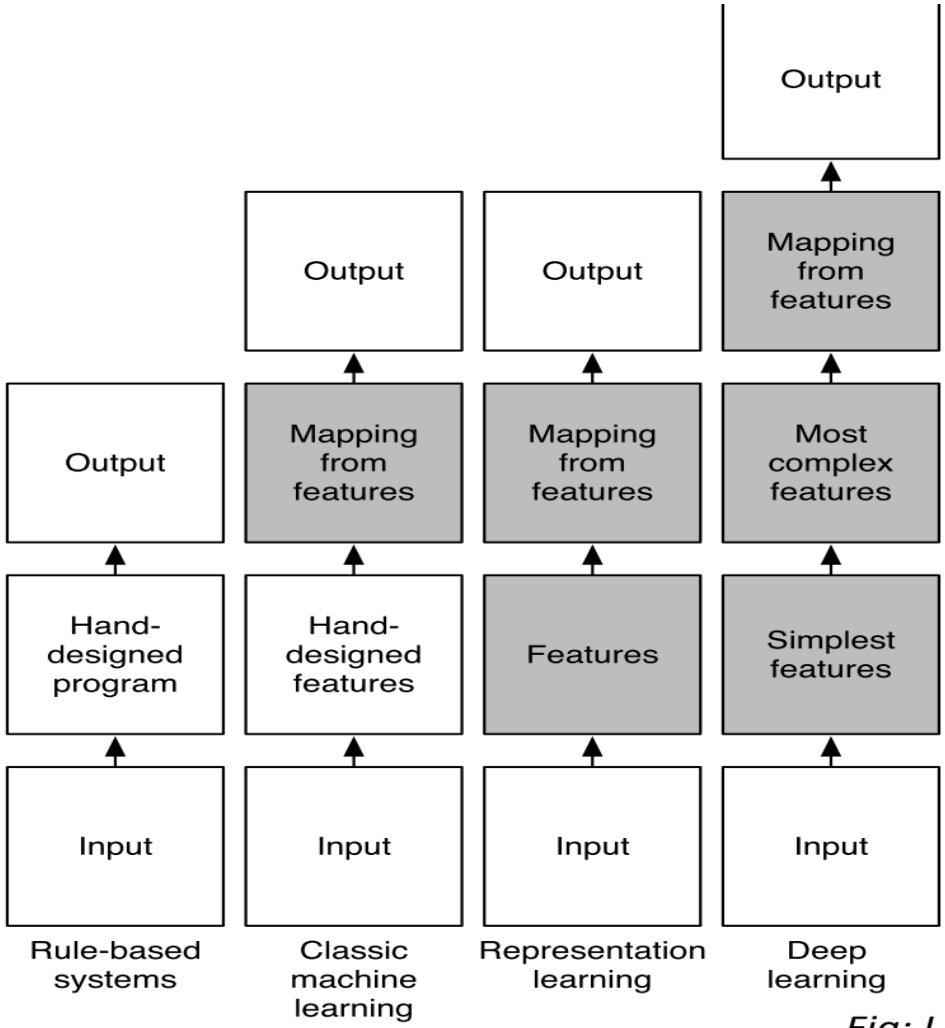
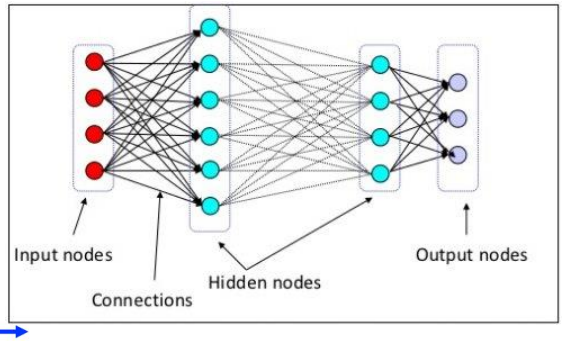
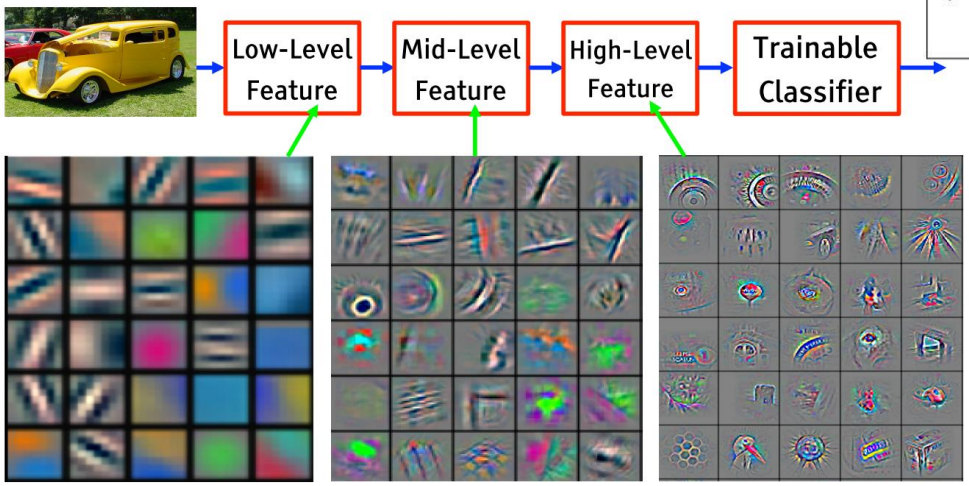


Fig: I. Goodfellow

What Types of Features?



- For image recognition
  - pixel -> edge -> texton -> motif -> part -> object

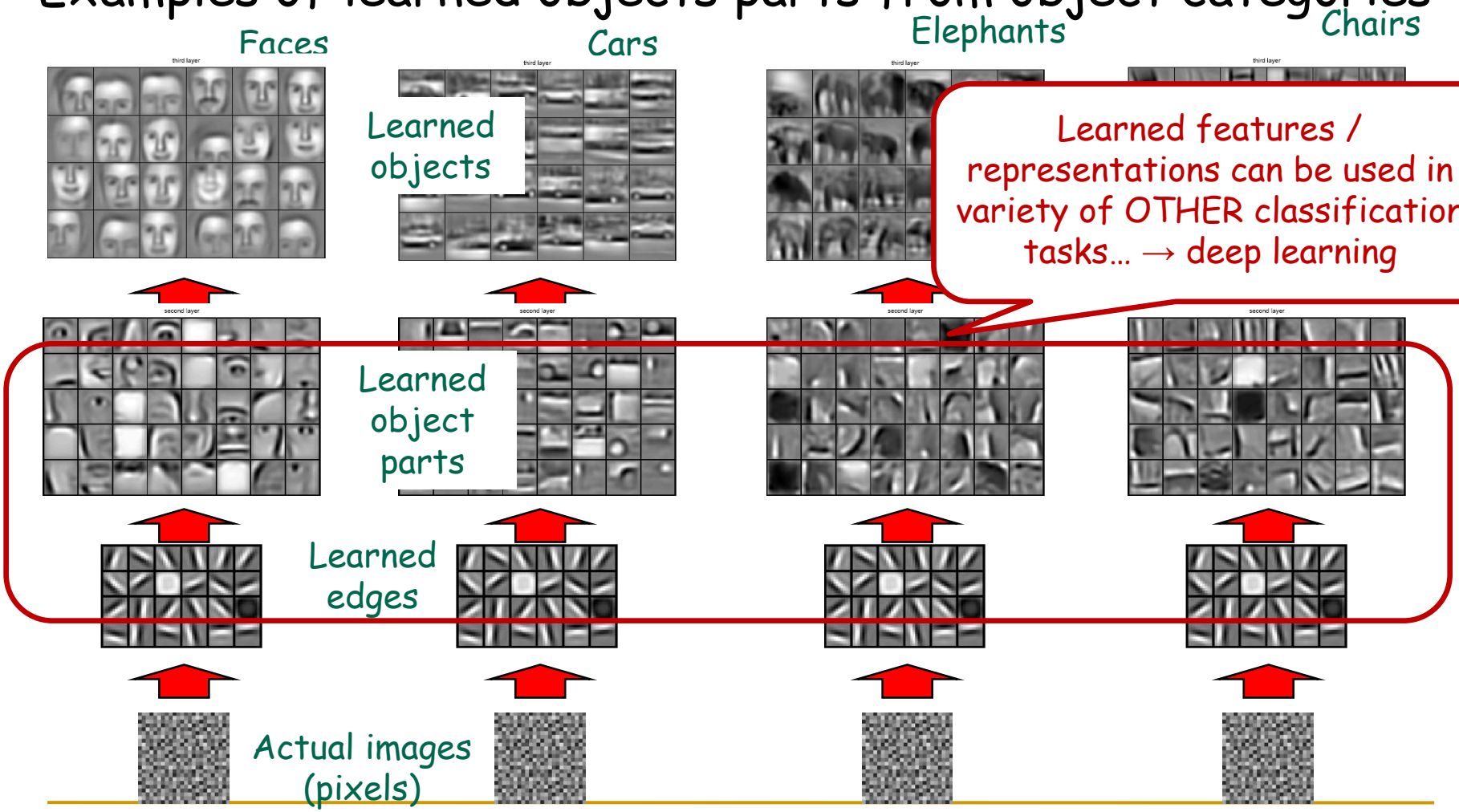


- For NLP
  - character -> word -> constituents -> clause -> sentence -> discourse
- For speech:
  - sample → spectral band -> sound -> … phone -> phoneme -> word

Figure from Y LeCun

Eg: Learning Image Features

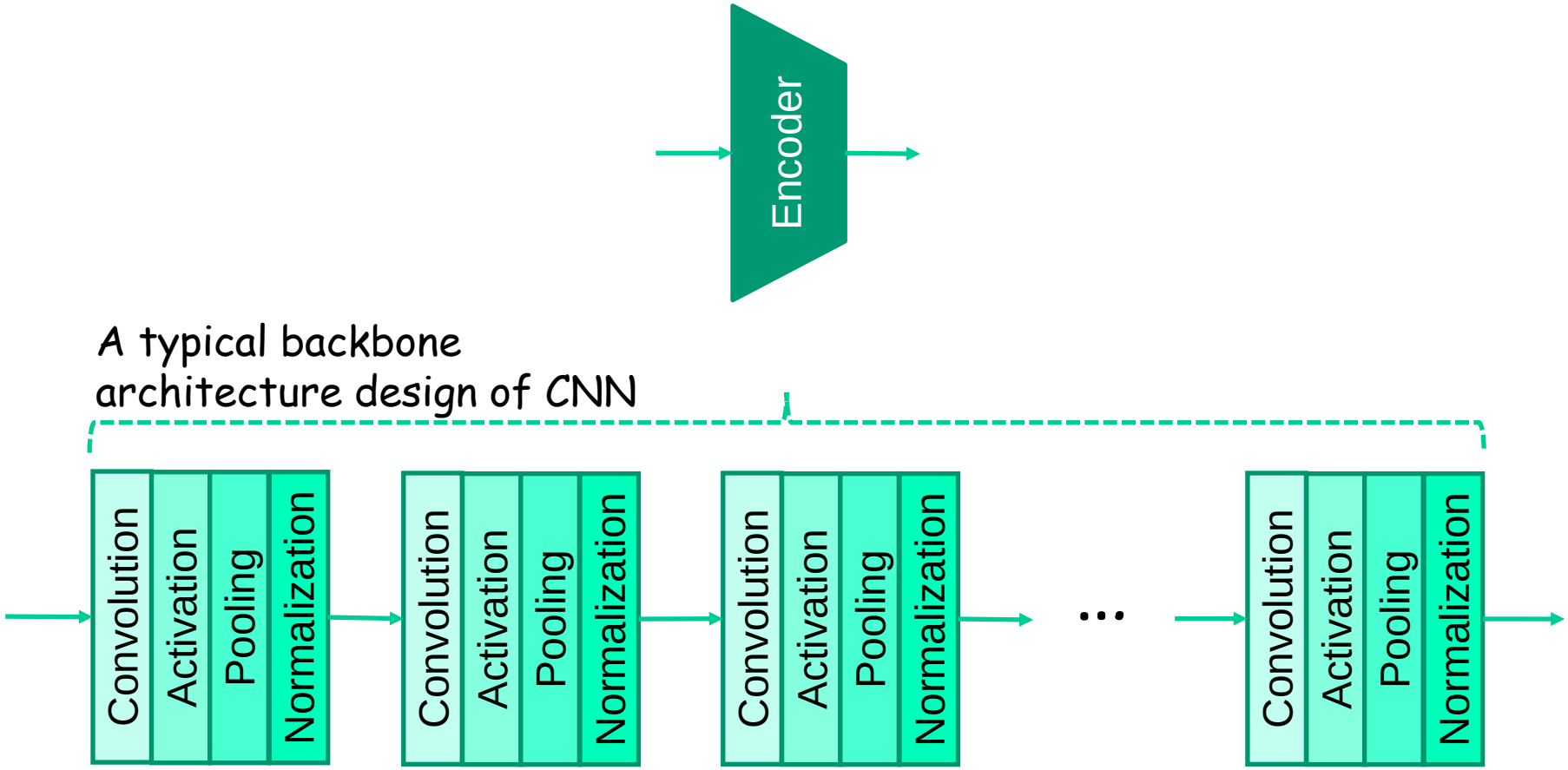
Examples of learned objects parts from object categories



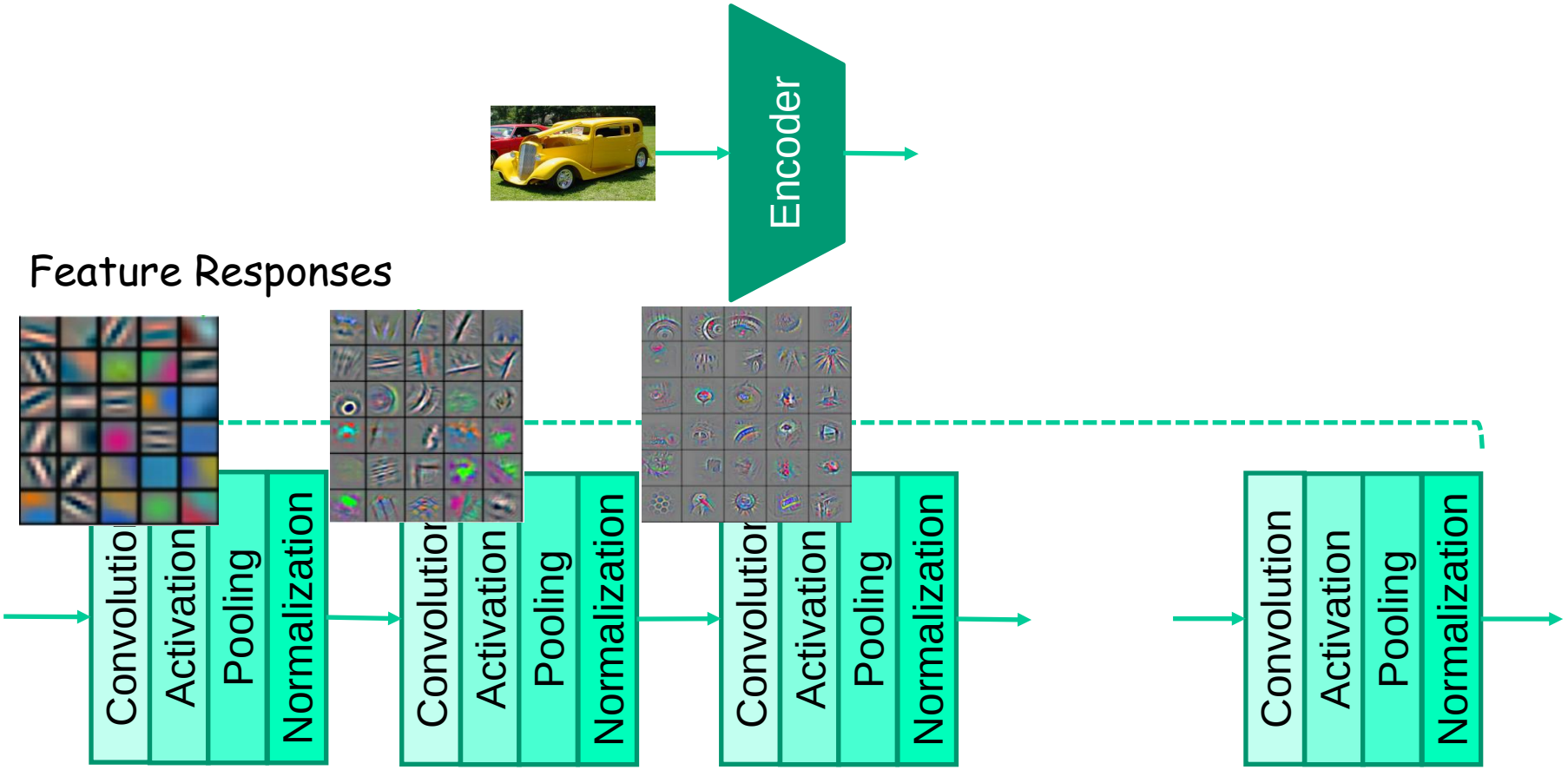
Today

1. Motivation
2. Feature Learning
3. Building Block Design of CNN
   - Convolutional Layer
   - Activation Function
   - Pooling Layer
   - Normalization Layer

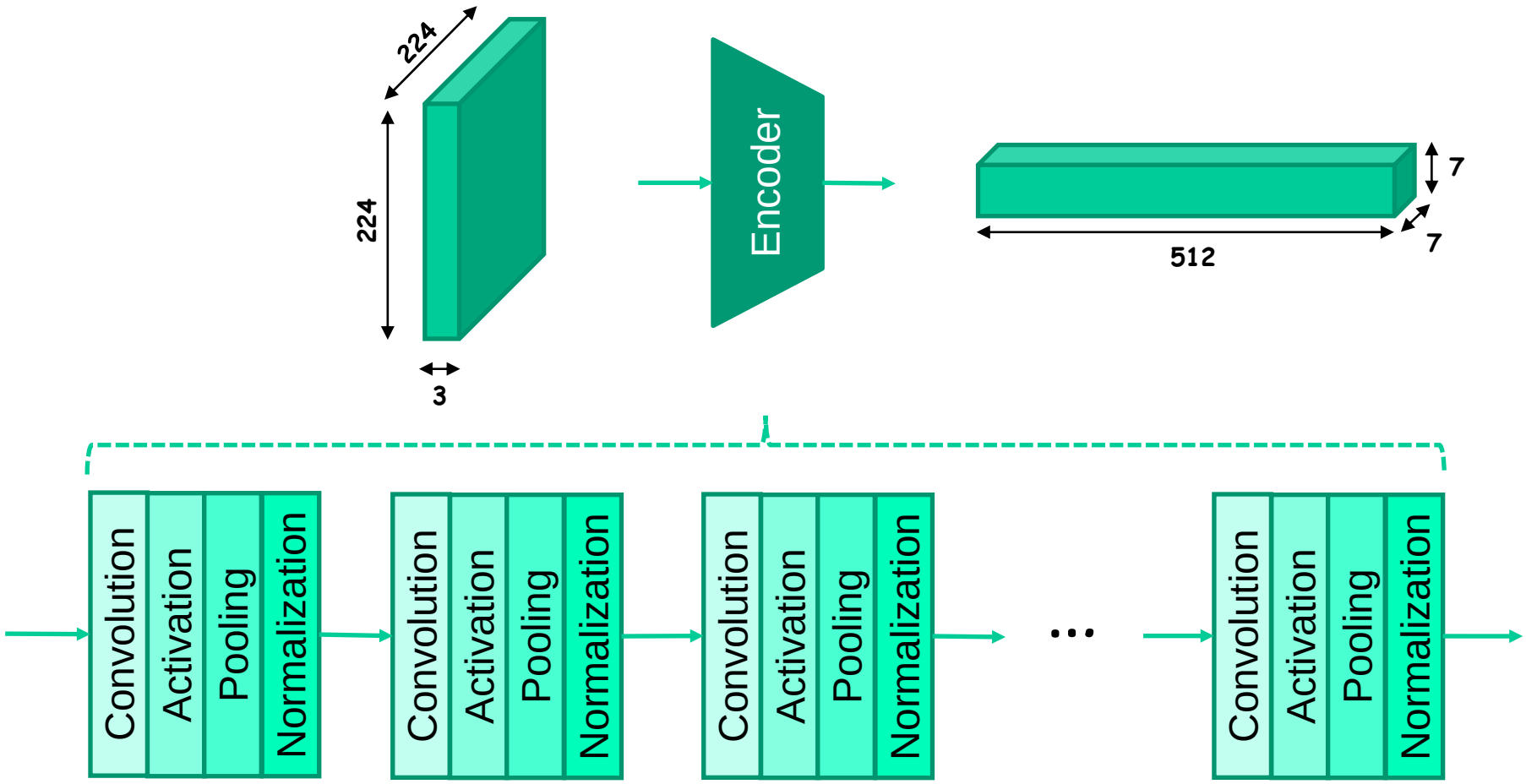
Building Block Design of CNN



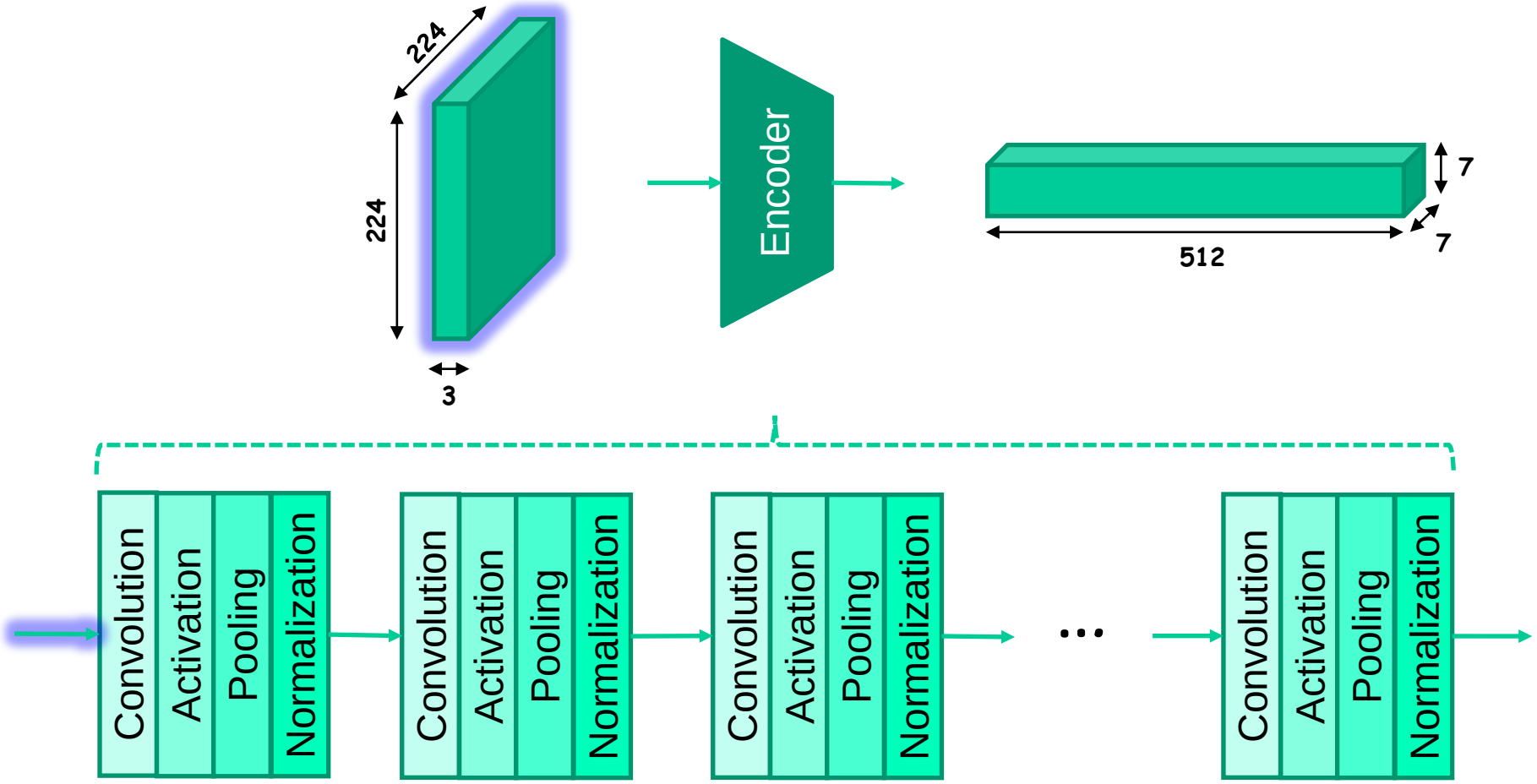
Building Block Design of CNN



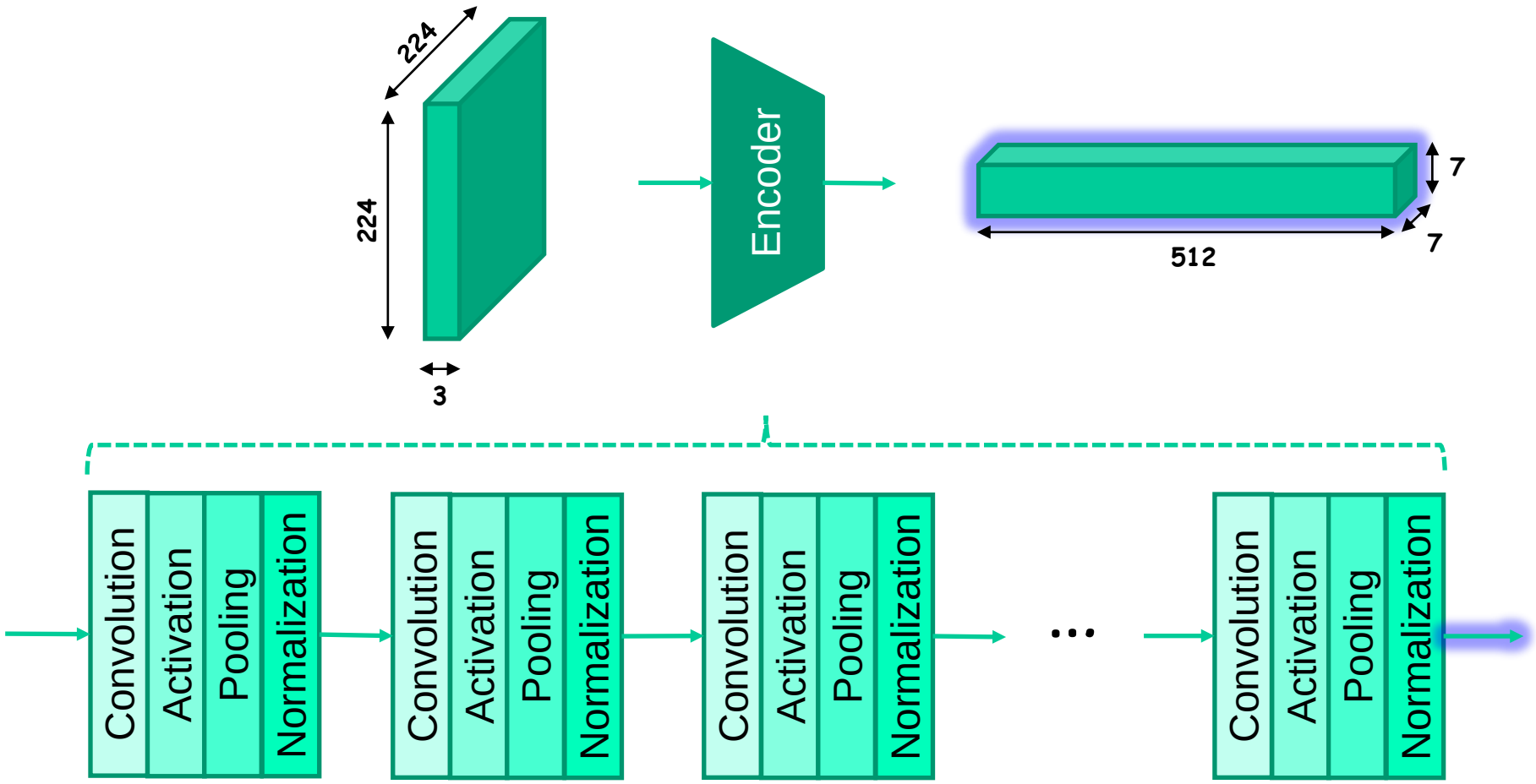
Building Block Design of CNN



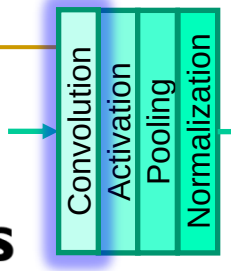
Building Block Design of CNN



Building Block Design of CNN

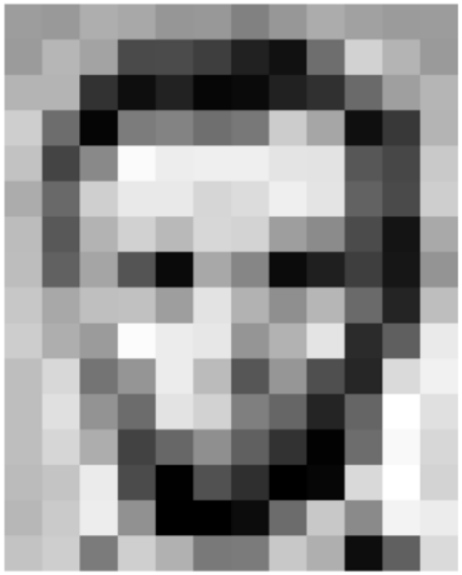


Conv Layer of CNN



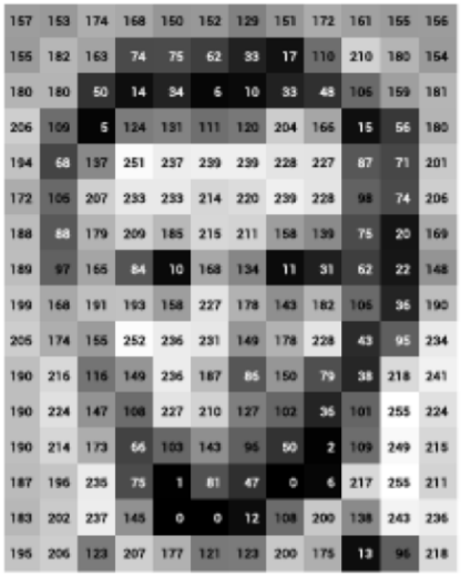
What computers 'see': Images as Numbers

What you see



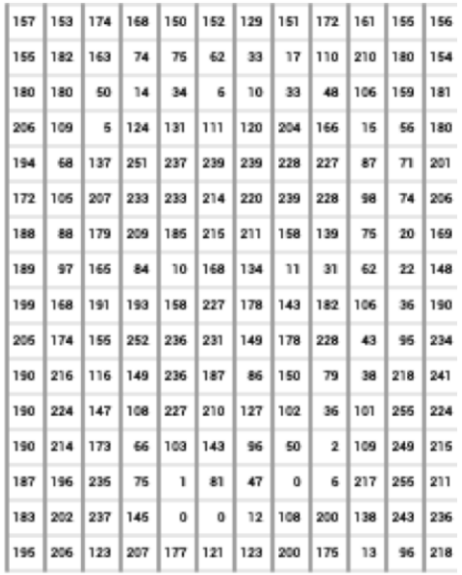
Input Image

What you both see



Input Image + values

What the computer "sees"



Pixel intensity values
("pix-el"=picture-element)

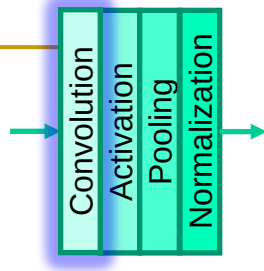
Levin *Image Processing & Computer Vision*

An image is just a matrix of numbers [0,255]. i.e., 1080x1080x3 for an RGB image.
Question: is this Lincoln? Washington? Jefferson? Obama?
How can the computer answer this question?

Can I just do classification on the 1,166400-long image vector directly?

No. Instead: exploit image spatial structure. Learn patches. Build them up

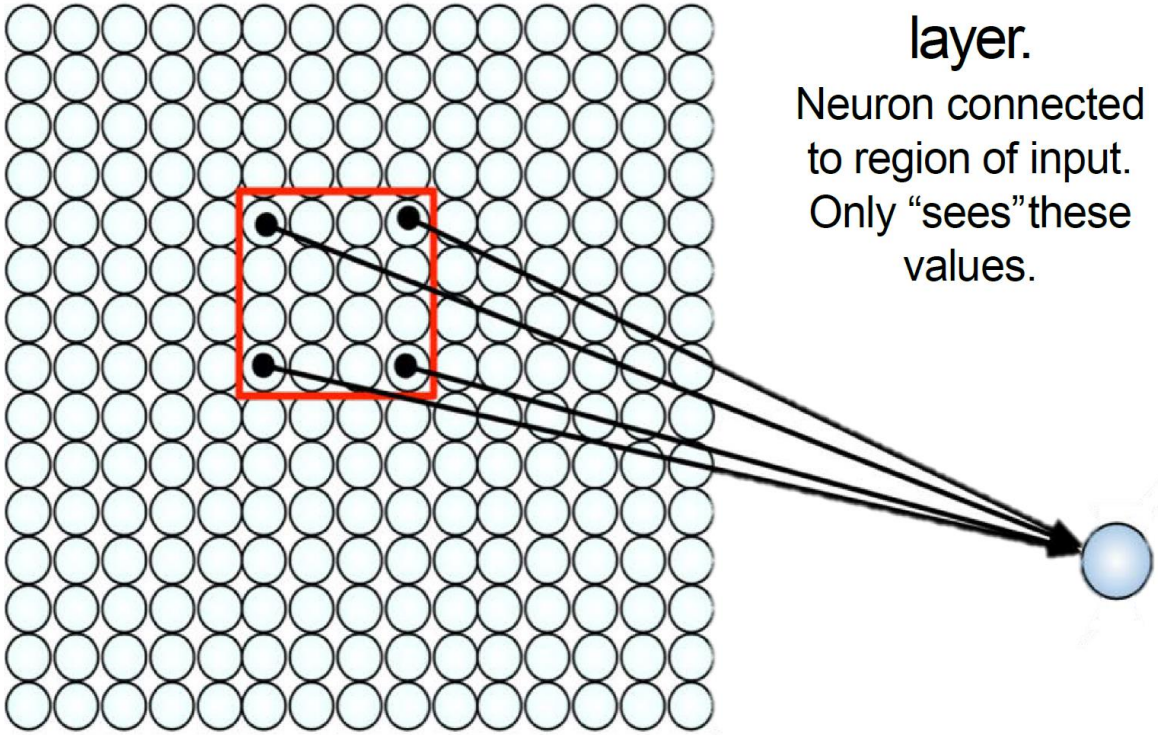
Using Spatial Structure



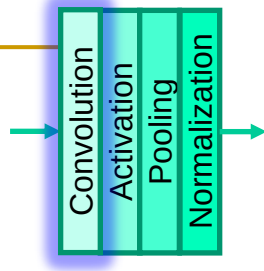
Input: 2D image. Array of pixel values

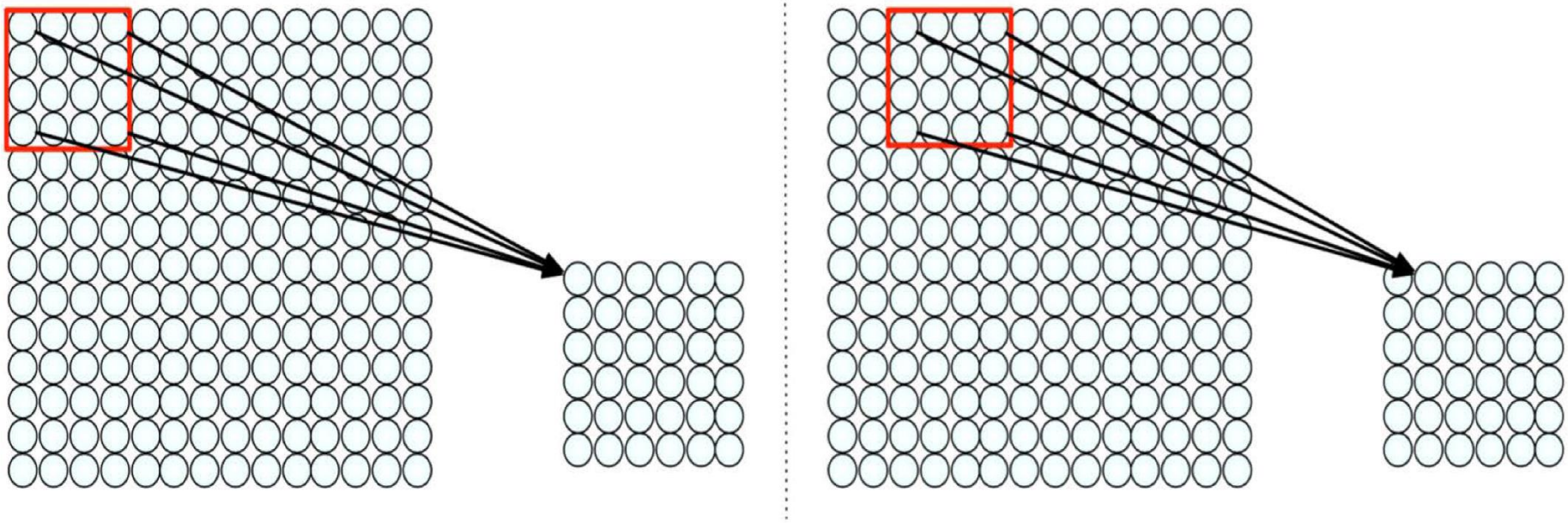
Idea: connect patches of input to neurons in hidden layer.

Neuron connected to region of input. Only "sees" these values.



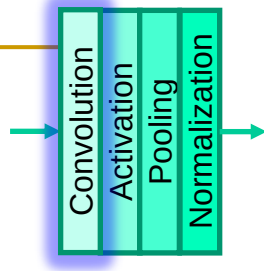
Using Spatial Structure

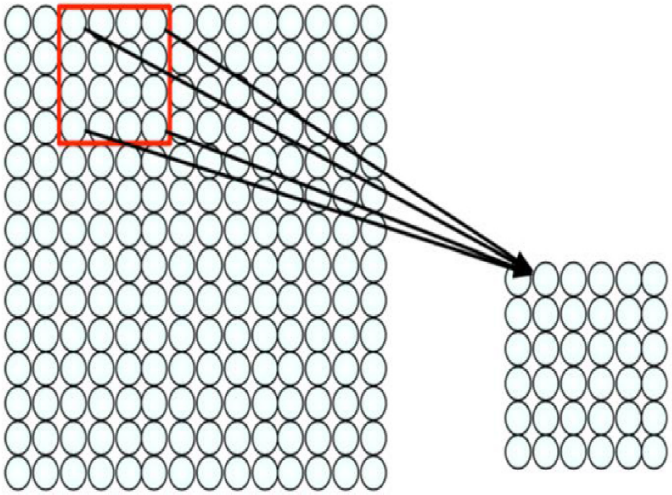




Connect patch in input layer to a single neuron in subsequent layer.
Use a sliding window to define connections.
How can we **weight** *the patch to detect particular features?*

Feature Extraction with Convolution





- Filter of size 4x4 : 16 different weights
- Apply this same filter to 4x4 patches in input
- Shift by 2 pixels for next patch

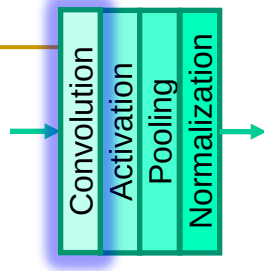
This "patchy" operation is **convolution**

1) Apply a set of weights – a filter – to extract **local features**

2) Use **multiple filters** to extract different features

3) Spatially share parameters of each filter

What is Convolution Operation?



- Elementwise **multiplication** and **addition**

Example 5x5 image with binary pixels

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Example 3x3 filter

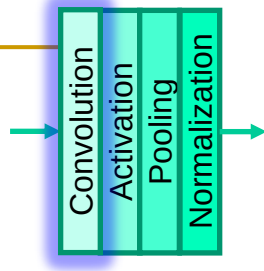
1	0	1
0	1	0
1	0	1

bias

0

- Scanning an image with a “filter”
  - Note: a filter is really just a perceptron, with weights and a bias

What is Convolution Operation?



- Elementwise **multiplication** and **addition**

1	0	1
0	1	0
1	0	1

Filter / Kernel

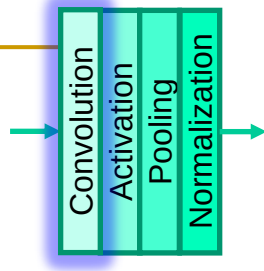
$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	0	0
$0_{\times 0}$	$1_{\times 1}$	$1_{\times 0}$	1	0
$0_{\times 1}$	$0_{\times 0}$	$1_{\times 1}$	1	1
0	0	1	1	0
0	1	1	0	0

Image

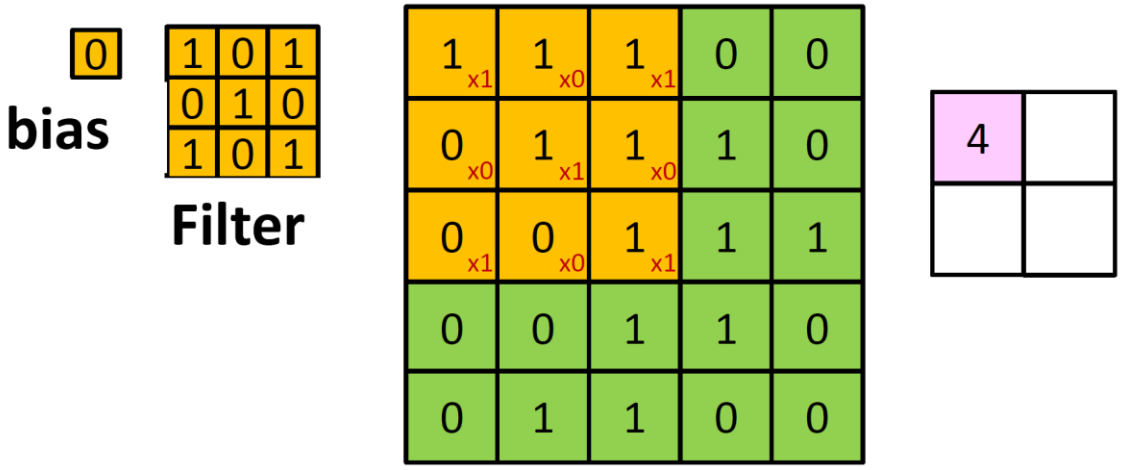
4		

Convolved Feature

Sliding Window for Convolution

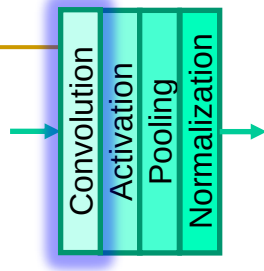


The "Stride" between adjacent scanned locations need not be 1

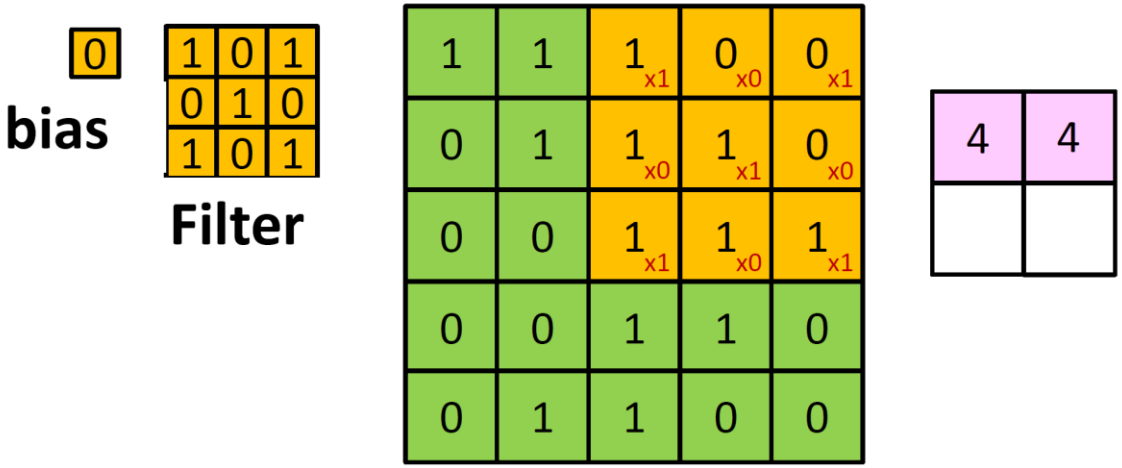


- Scanning an image with a "filter"
  - The filter may proceed by *more* than 1 pixel at a time
  - E.g. with a "stride" of *two* pixels per shift

Sliding Window for Convolution

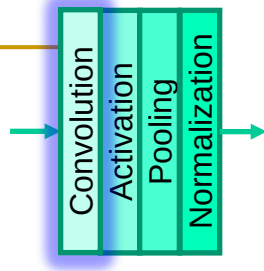


The "Stride" between adjacent scanned locations need not be 1

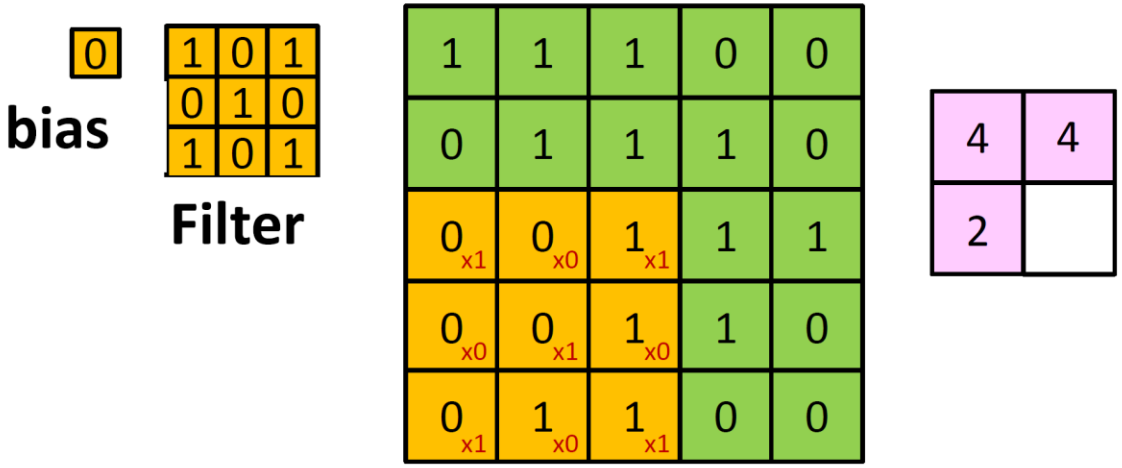


- Scanning an image with a "filter"
  - The filter may proceed by *more* than 1 pixel at a time
  - E.g. with a "hop" of *two* pixels per shift

Sliding Window for Convolution

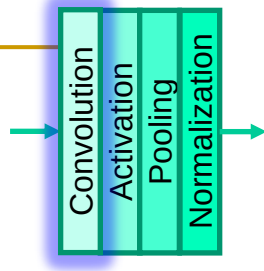


The "Stride" between adjacent scanned locations need not be 1

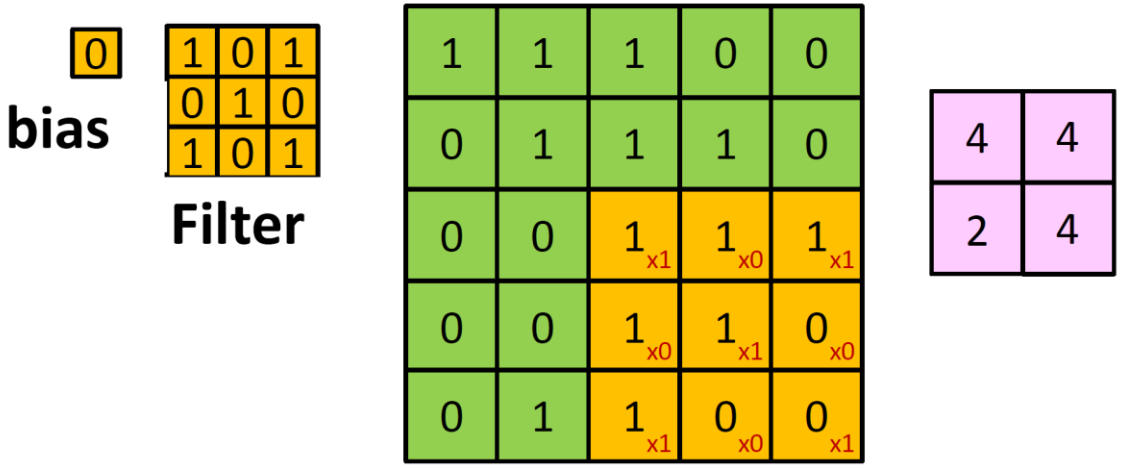


- Scanning an image with a "filter"
  - The filter may proceed by *more* than 1 pixel at a time
  - E.g. with a "hop" of *two* pixels per shift

Sliding Window for Convolution

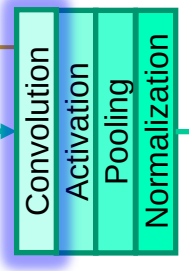


The "Stride" between adjacent scanned locations need not be 1

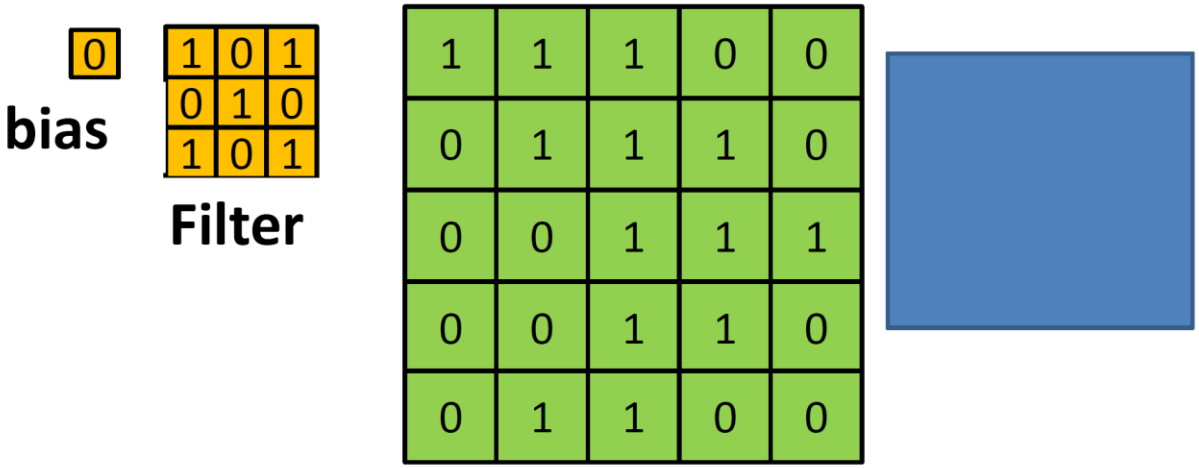


- Scanning an image with a "filter"
  - The filter may proceed by *more* than 1 pixel at a time
  - E.g. with a "hop" of *two* pixels per shift

Output Feature Map size in Convolution

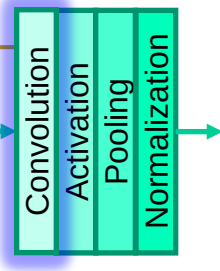


The size of the convolution

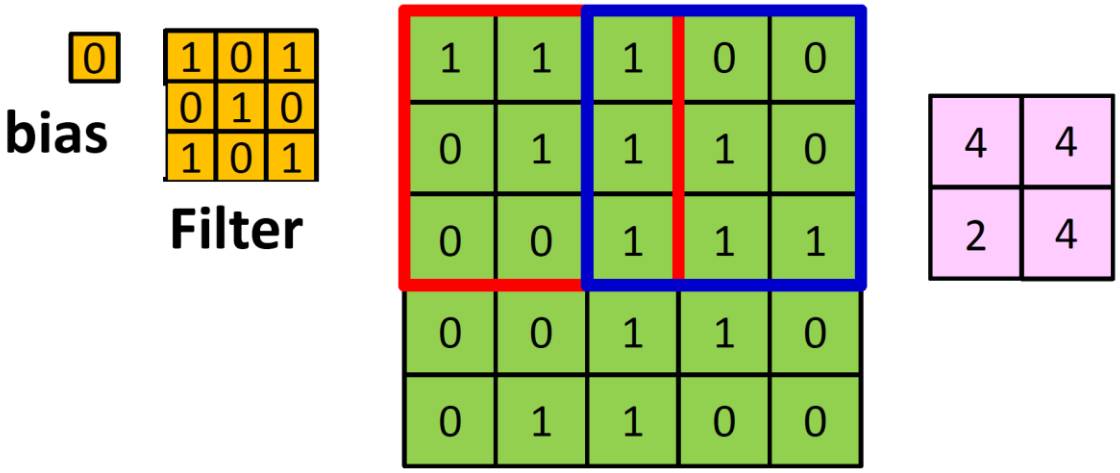


- Image size: 5x5
- Filter: 3x3
- Stride: 2
- Output size = ?

Output Feature Map size in Convolution

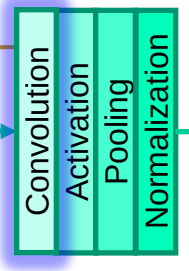


The size of the convolution

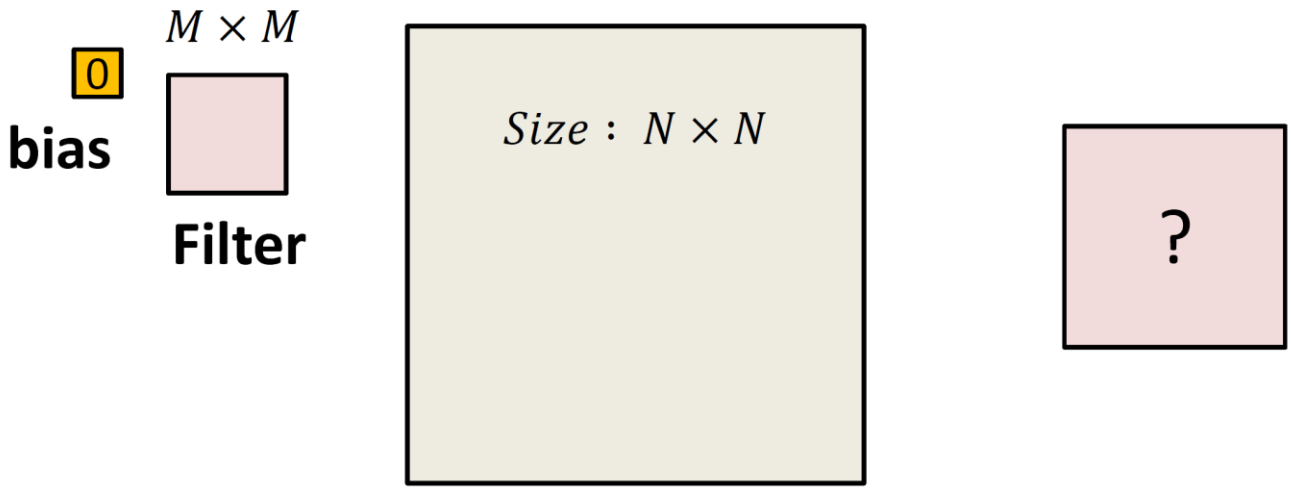


- Image size: 5x5
- Filter: 3x3
- Stride: 2
- Output size = ?

Output Feature Map size in Convolution

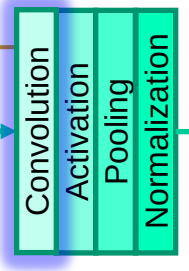


The size of the convolution

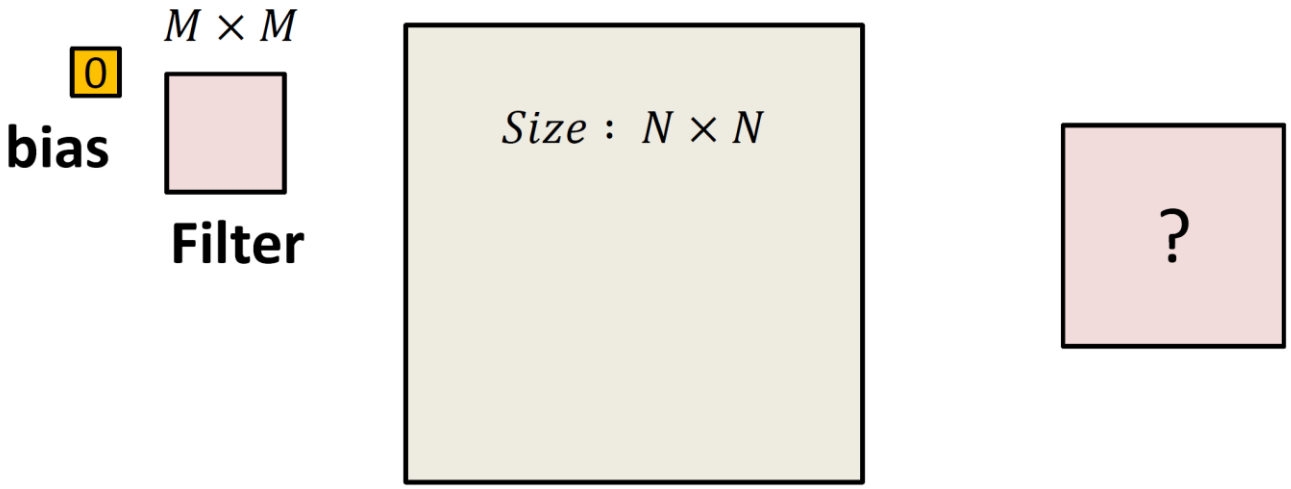


- Image size: $N \times N$
- Filter: $M \times M$
- Stride: 1
- Output size = ?

Output Feature Map size in Convolution

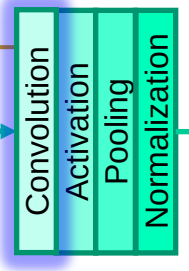


The size of the convolution

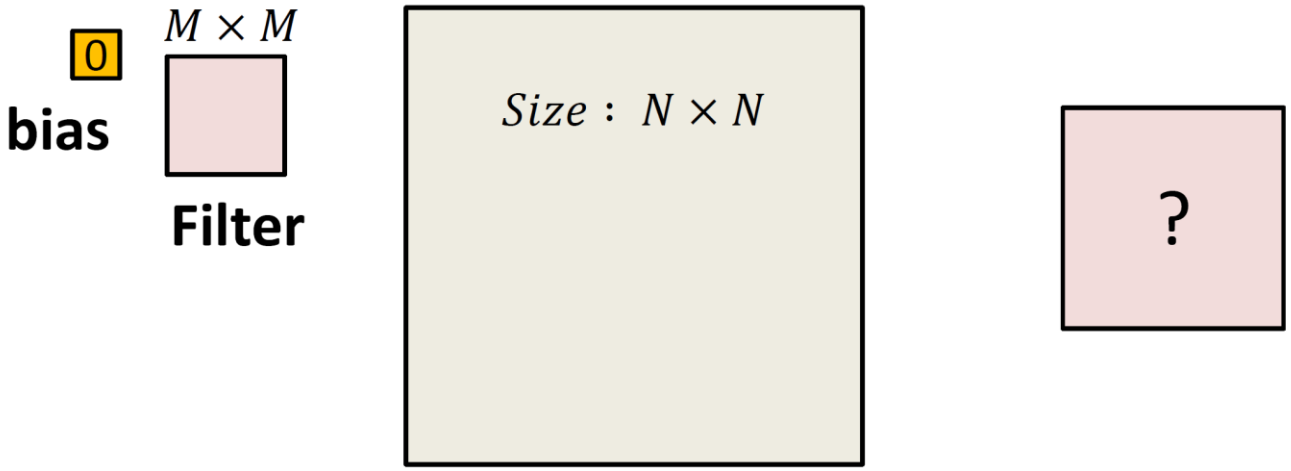


- Image size: $N \times N$
- Filter: $M \times M$
- Stride: S
- Output size = ?

Output Feature Map size in Convolution

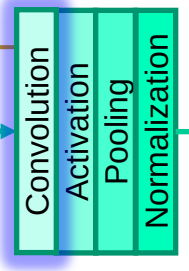


The size of the convolution



- Image size: $N \times N$
- Filter: $M \times M$
- Stride: S
- Output size (each side) = $\lfloor (N - M)/S \rfloor + 1$
  - Assuming you're not allowed to go beyond the edge of the input

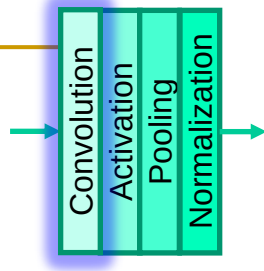
Output Feature Map size in Convolution



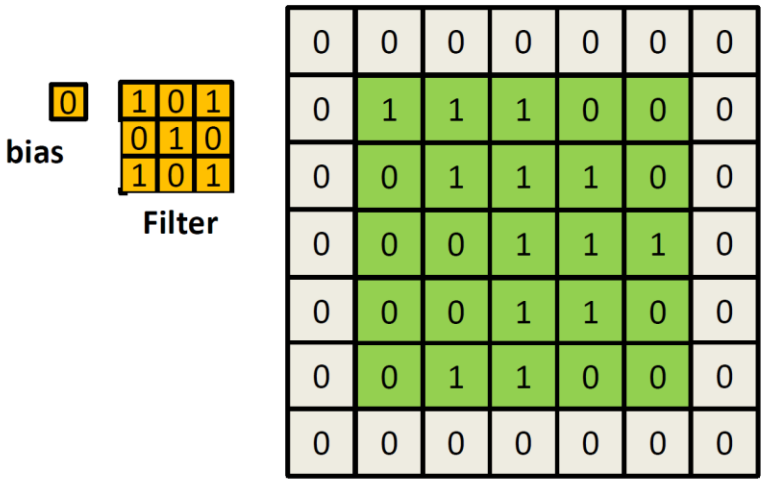
Convolution Size

- Simple convolution size pattern:
  - Image size: $N \times N$
  - Filter: $M \times M$
  - Stride: S
  - **Output size (each side)** = $\lfloor (N - M)/S \rfloor + 1$
    - Assuming you're not allowed to go beyond the edge of the input
- Results in a reduction in the output size
  - Even if $S = 1$
  - Sometimes not considered acceptable
    - If there's no active downsampling, through max pooling and/or $S > 1$, then the output map should ideally be the same size as the input

Padding for Convolution

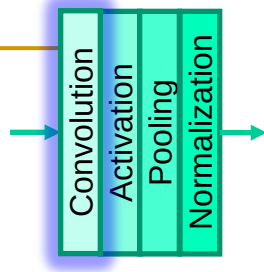


Solution



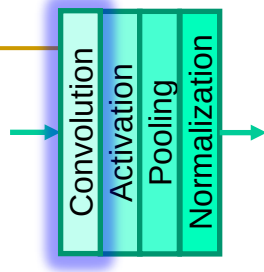
- Zero-pad the input
  - Pad the input image/map all around
  - Pad as symmetrically as possible, such that..
  - **For stride 1, the result of the convolution is the same size as the original image**

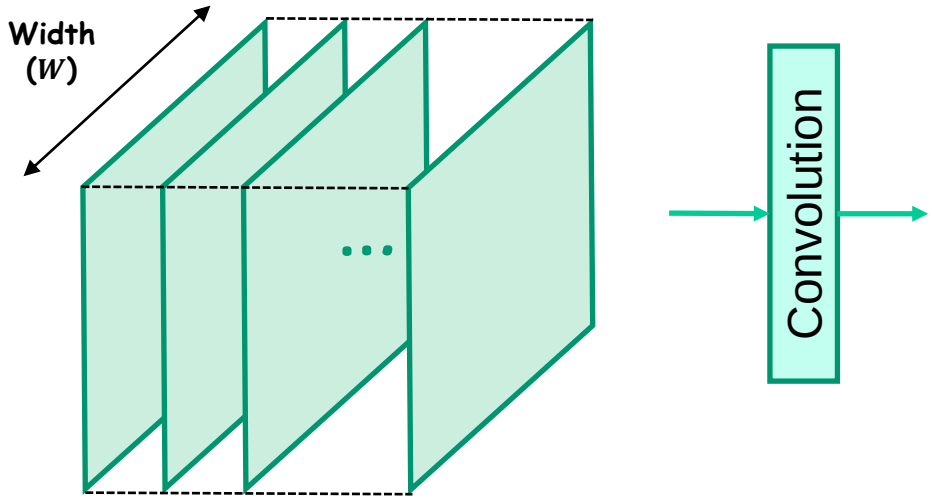
Conv Layer of CNN



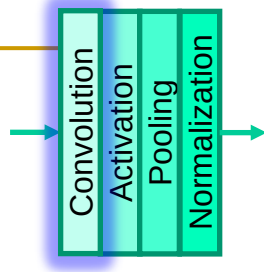
https://cs231n.github.io/convolutional-networks/

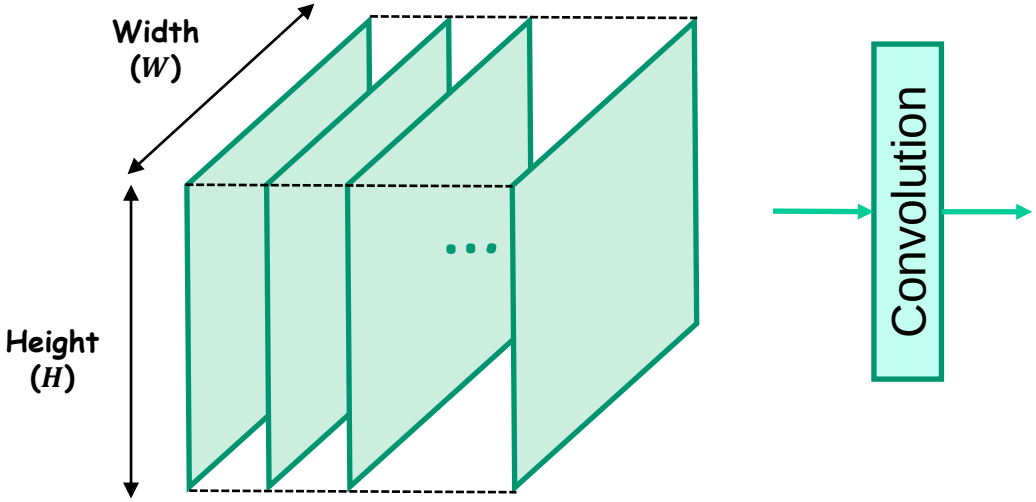
Convolutional Feature Mapping



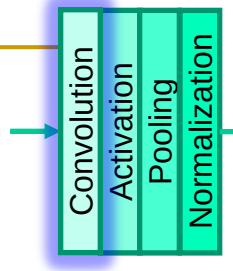


Convolutional Feature Mapping

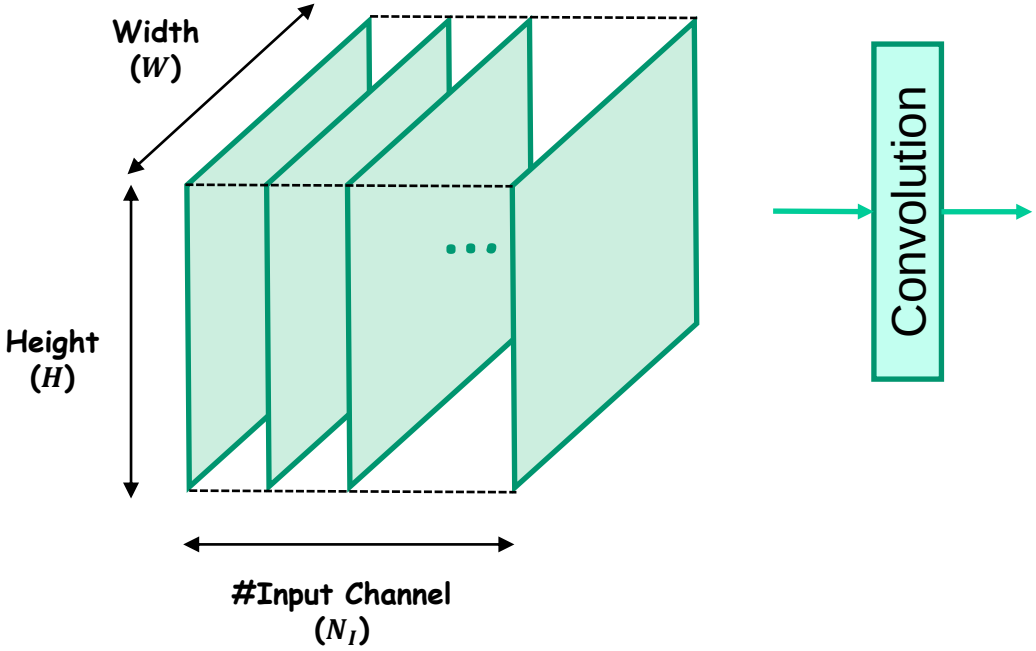




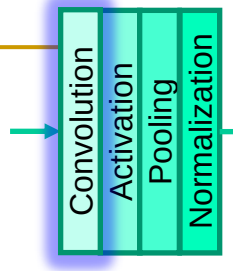
Convolutional Feature Mapping



$$F^I \in \mathbb{R}^{H \times W \times N_I}$$

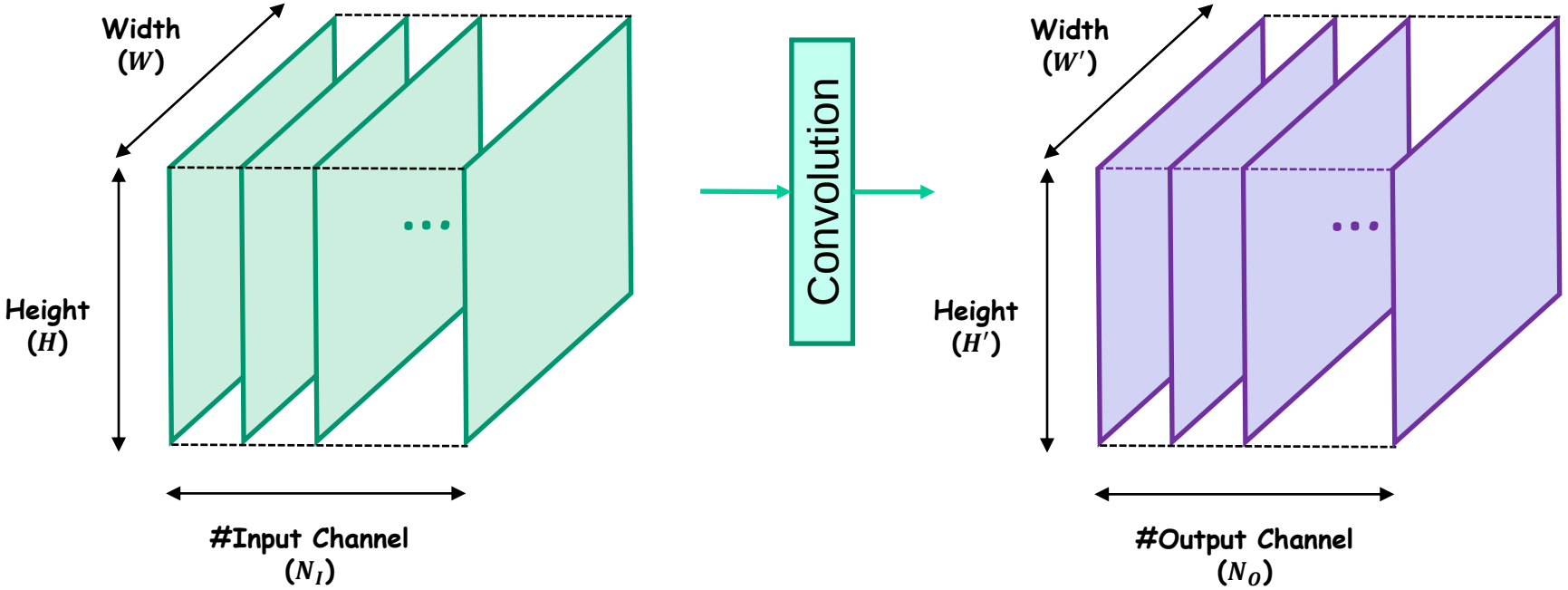


Convolutional Feature Mapping

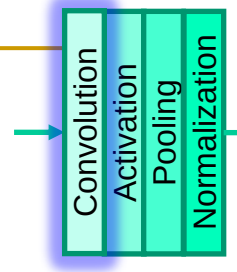


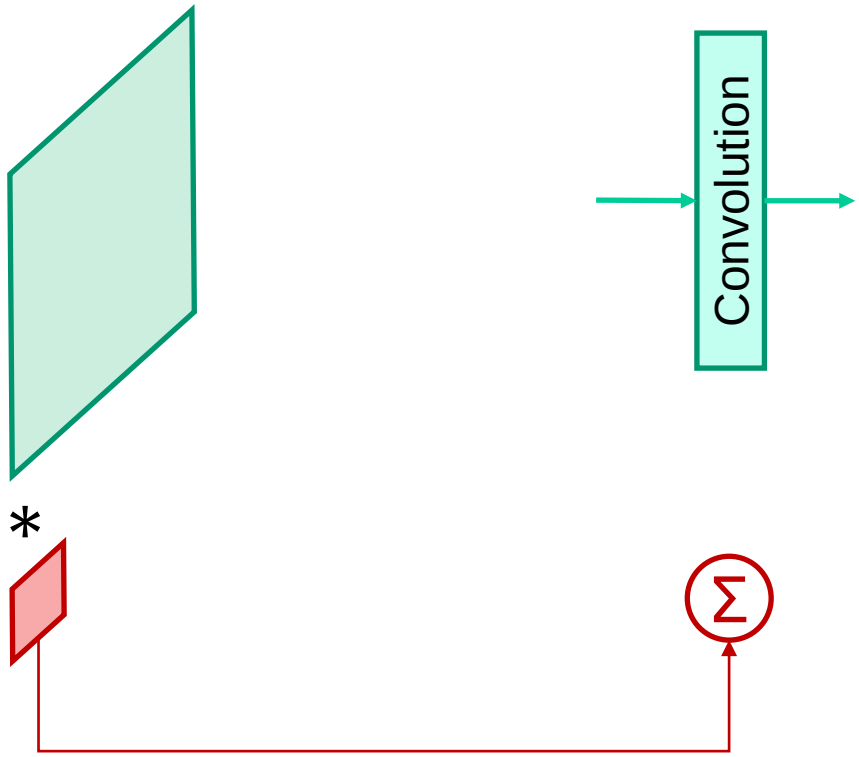
$F^I \in \mathbb{R}^{H \times W \times N_I}$

$F^O \in \mathbb{R}^{H' \times W' \times N_O}$

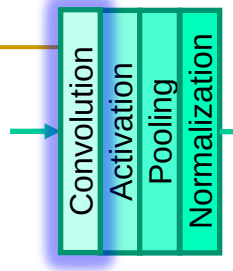


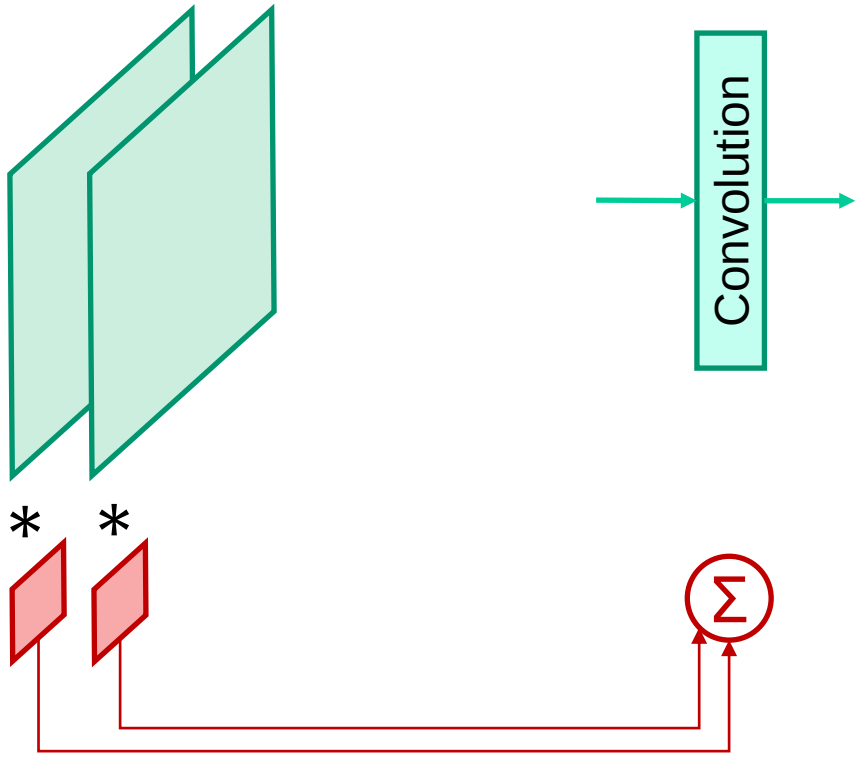
Convolutional Feature Mapping



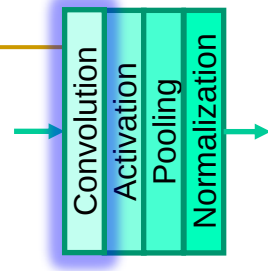


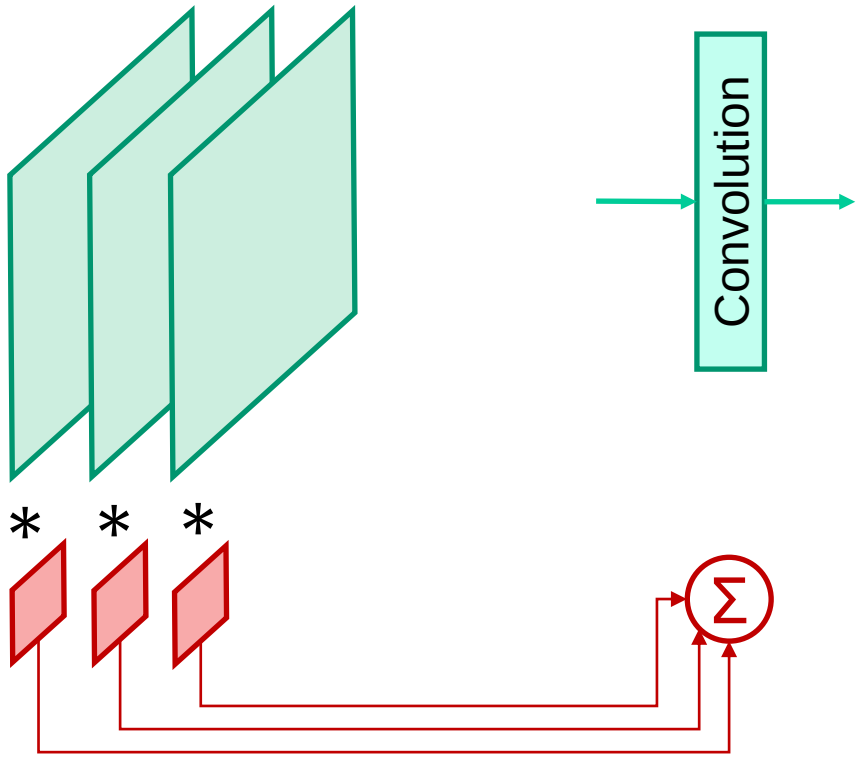
Convolutional Feature Mapping



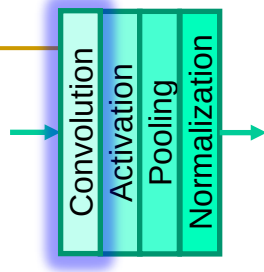


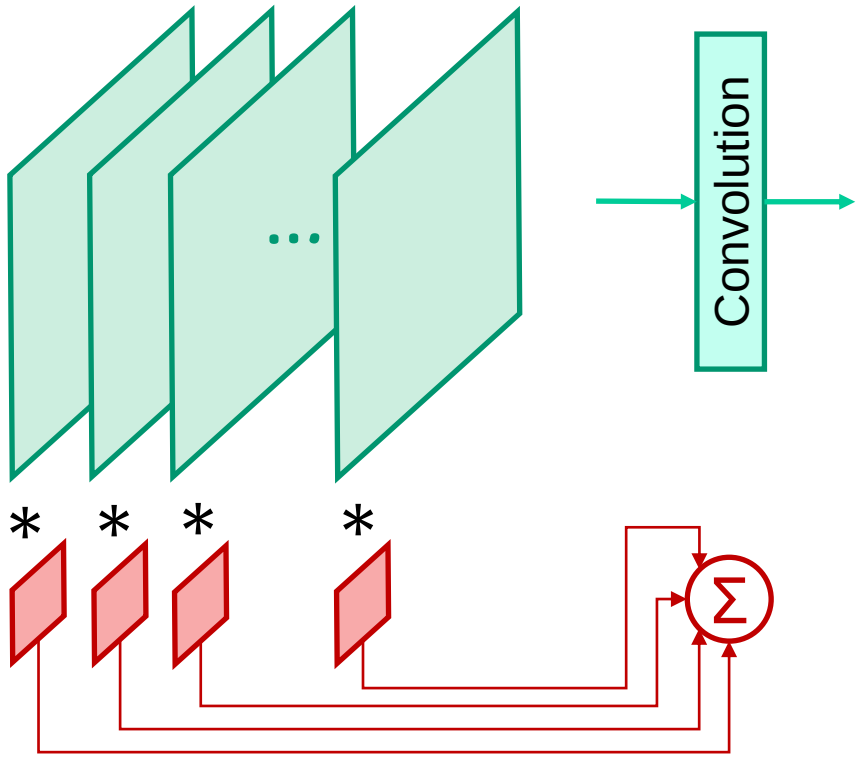
Convolutional Feature Mapping



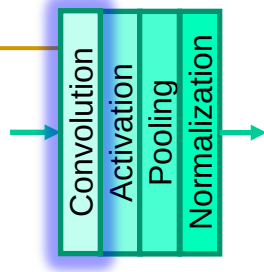


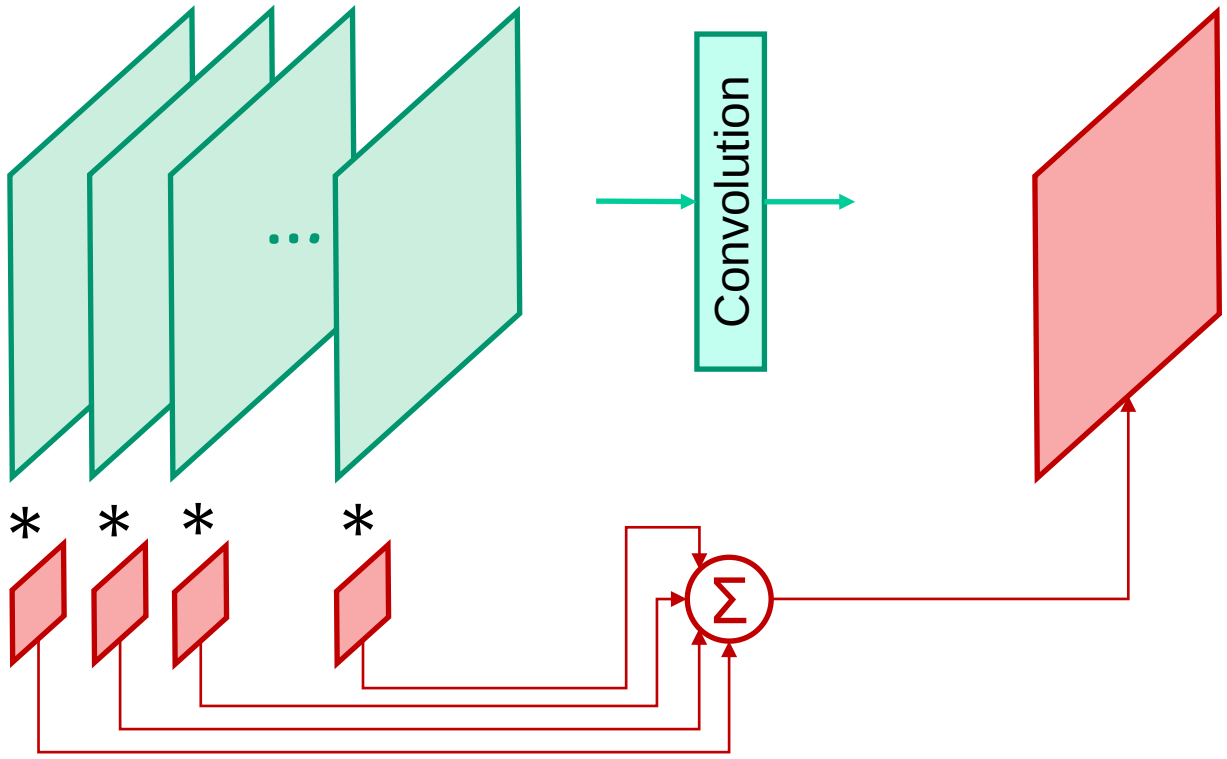
Convolutional Feature Mapping



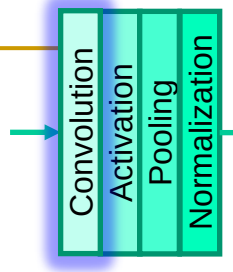


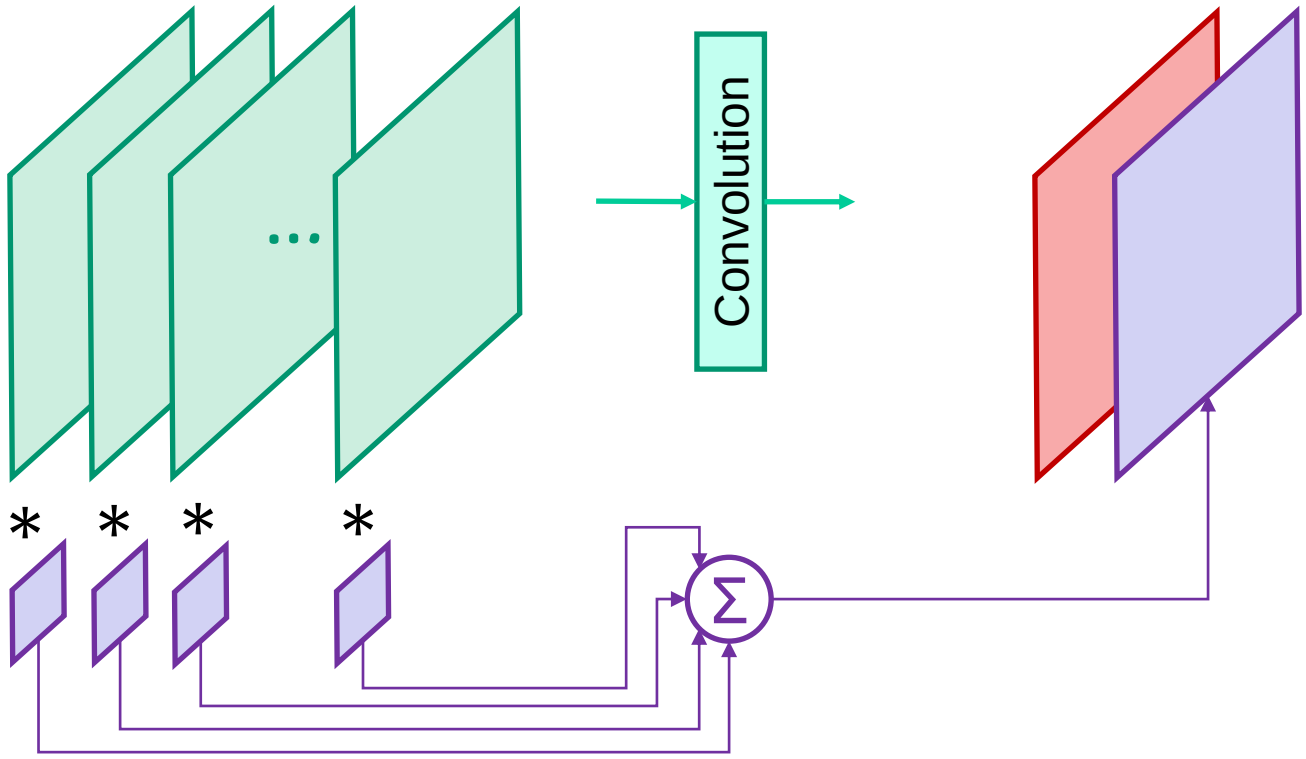
Convolutional Feature Mapping



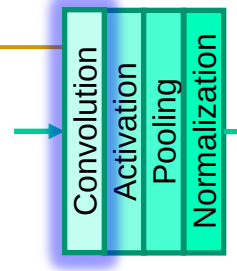


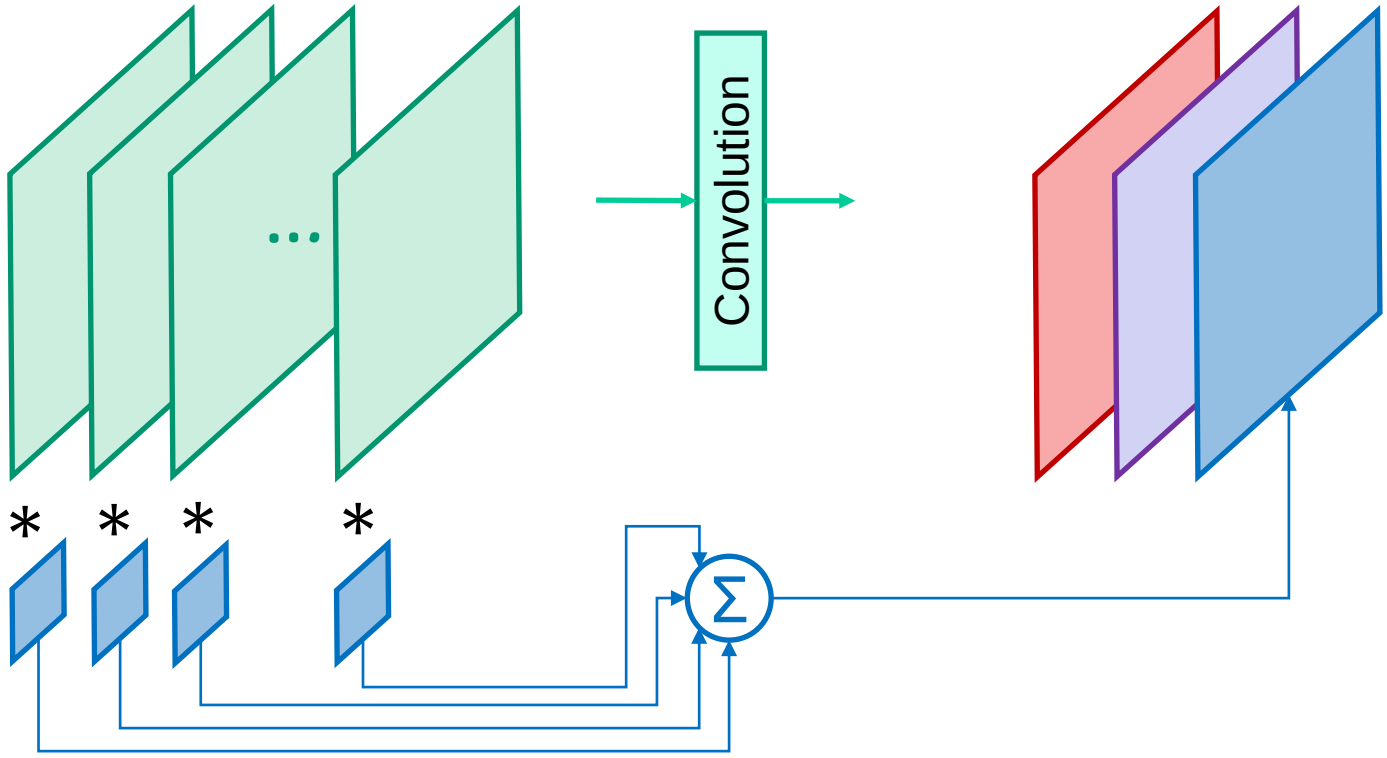
Convolutional Feature Mapping



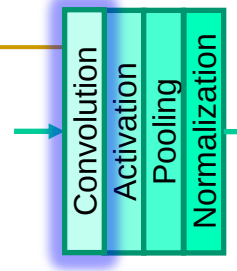


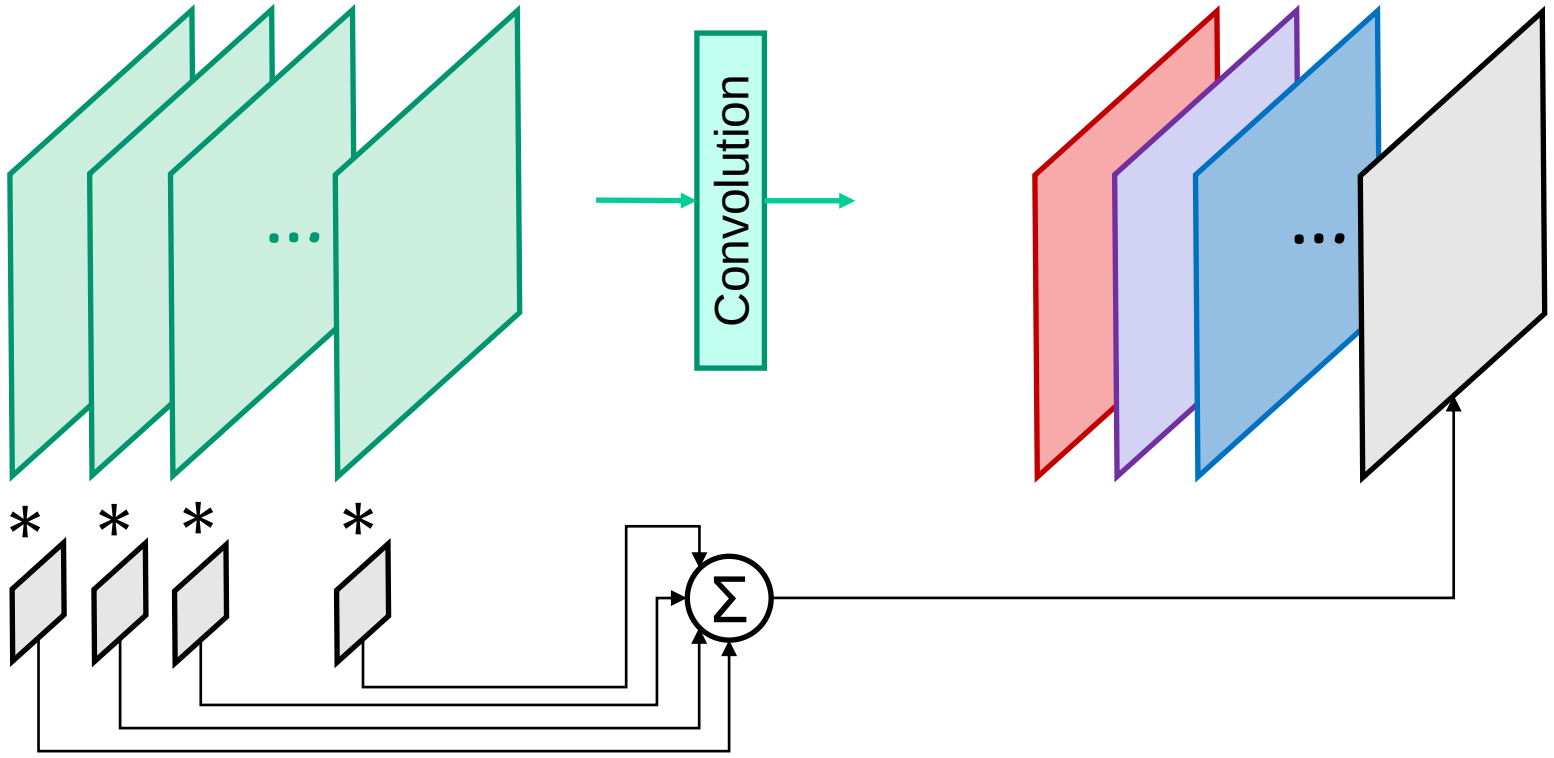
Convolutional Feature Mapping



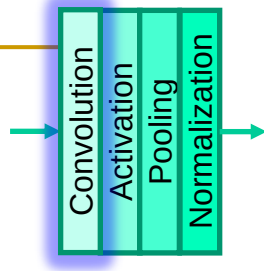


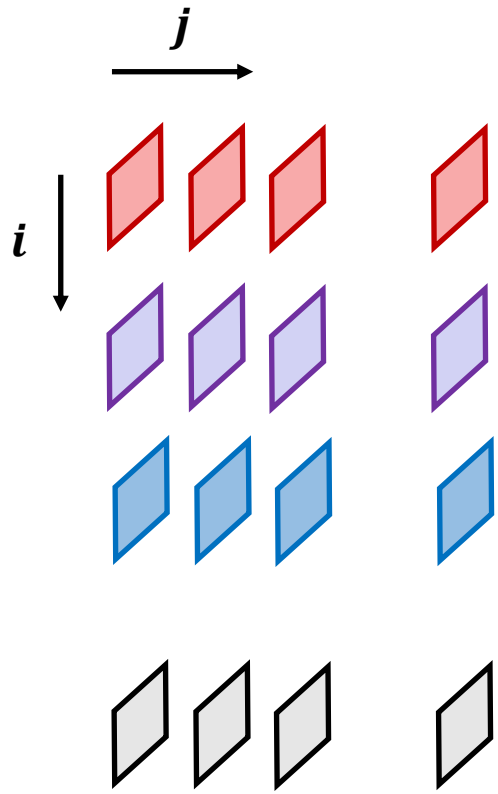
Convolutional Feature Mapping





Convolutional Feature Mapping

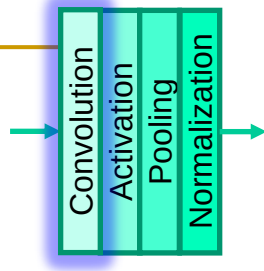


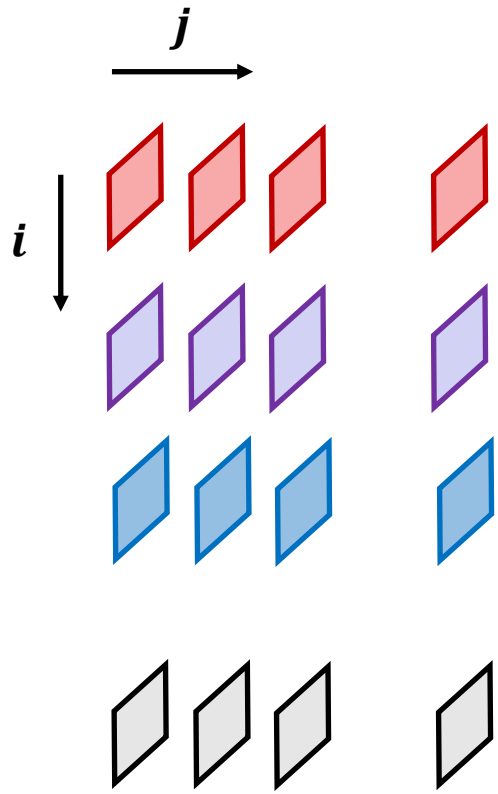


Convolutional filters (aka kernels) can be represented in Tensor format

$$\boldsymbol{K} \in \mathbb{R}^{\boldsymbol{h}\times\boldsymbol{w}\times\boldsymbol{N_I}\times\boldsymbol{N_O}}$$

Convolutional Feature Mapping



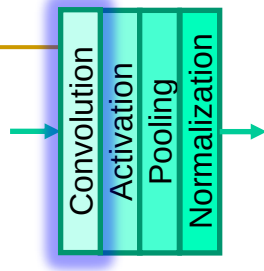


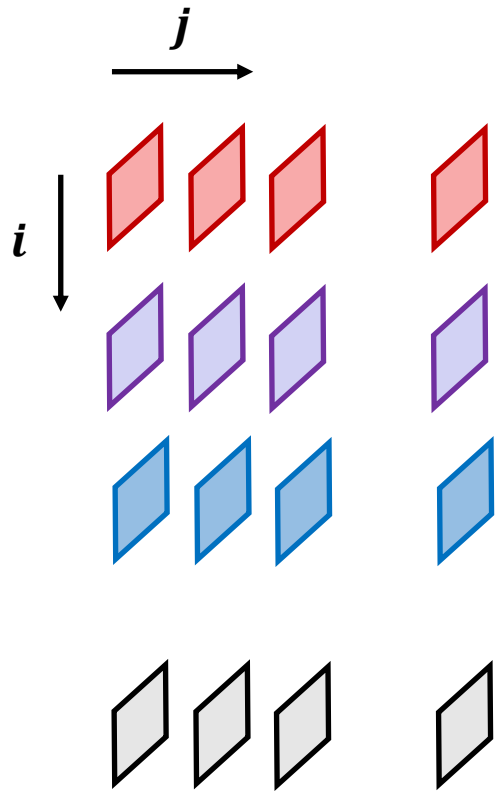
Convolutional filters (aka kernels) can be represented in Tensor format

$$\boldsymbol{K} \in \mathbb{R}^{\boldsymbol{h}\times\boldsymbol{w}\times\boldsymbol{N_I}\times\boldsymbol{N_O}}$$

Ex1. 3x3x64x128

Convolutional Feature Mapping





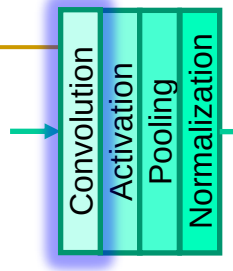
Convolutional filters (aka kernels) can be represented in Tensor format

$$\boldsymbol{K} \in \mathbb{R}^{\boldsymbol{h}\times\boldsymbol{w}\times\boldsymbol{N_I}\times\boldsymbol{N_O}}$$

Ex1. 3x3x64x128

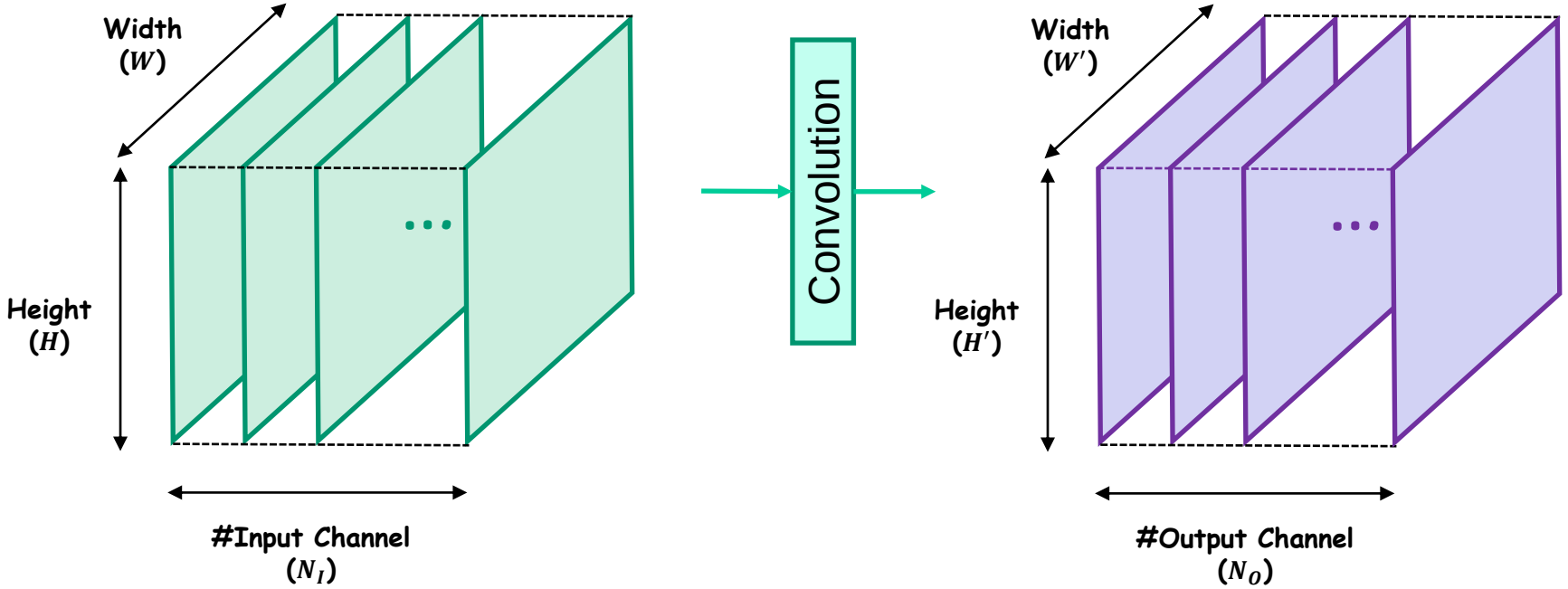
Ex2. 5x5x64x128

Convolutional Feature Mapping



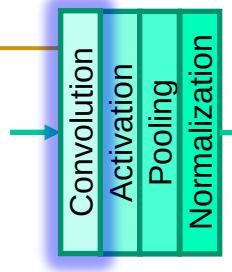
$$F^I \in \mathbb{R}^{H \times W \times N_I}$$

$$F^O \in \mathbb{R}^{H' \times W' \times N_O}$$



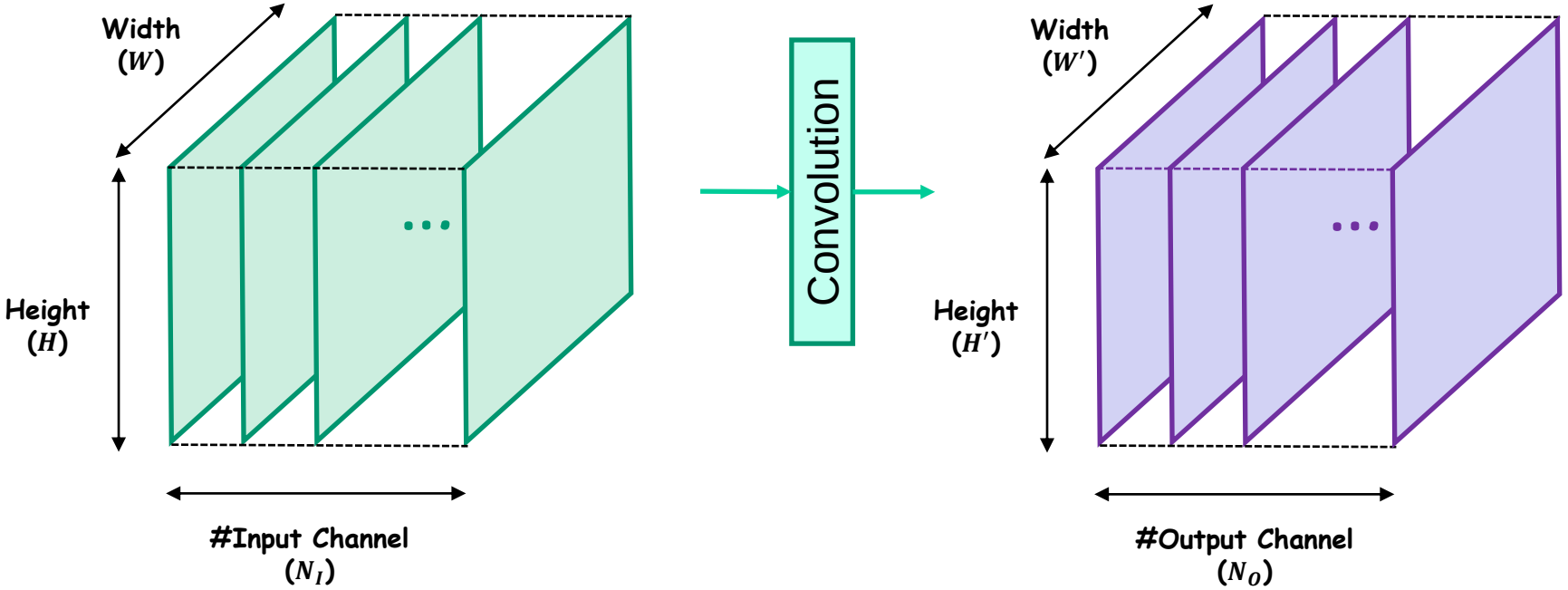
$$F^O(:,:,j) = \left[\sum_{i=1}^{N_I} F^I(:,:,i) * K(:,:,i,j)\right] + b_j$$

Convolutional Feature Mapping

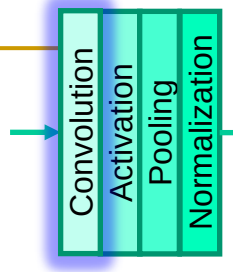


Ex1. Input Feature Map: 56x56x64 -->

Output Feature Map: 56x56x128 (Stride=1)

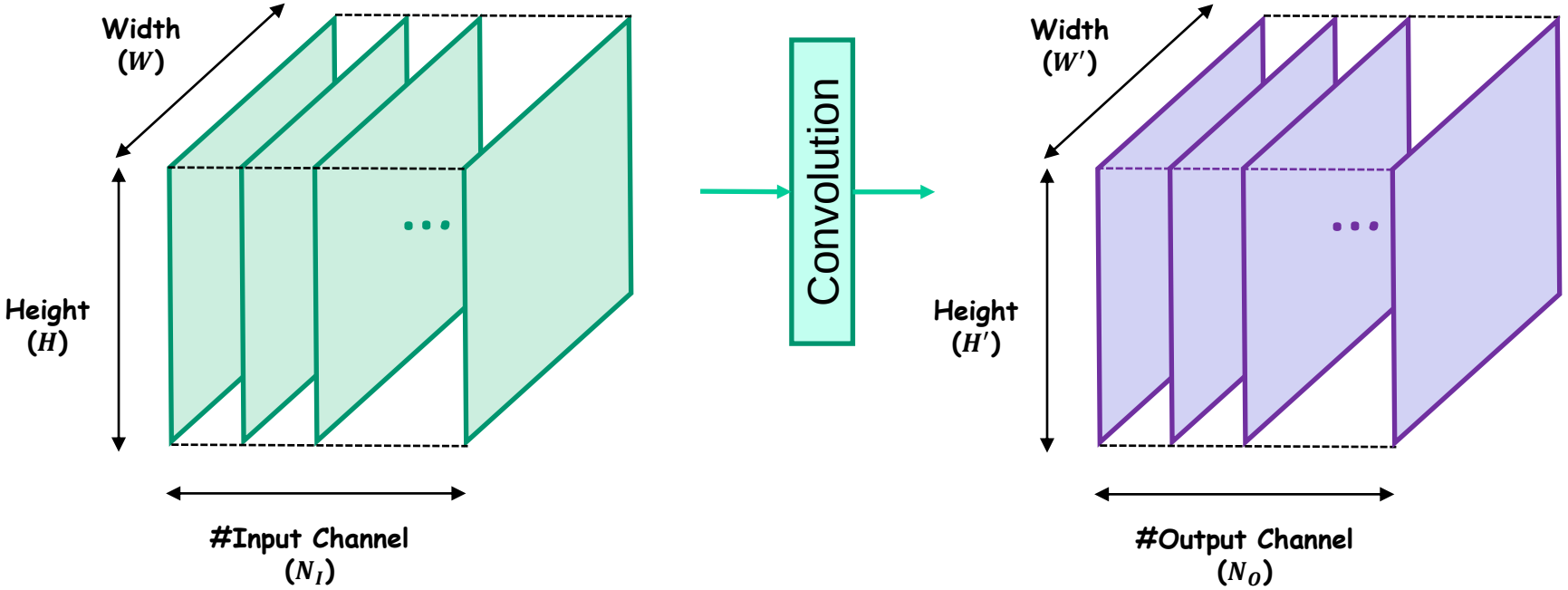


Convolutional Feature Mapping

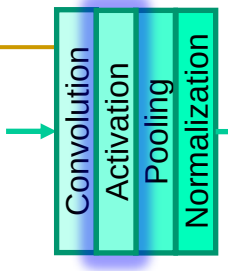


Ex1. Input Feature Map: 56x56x64 -->

Output Feature Map: 28x28x128 (Stride=2)

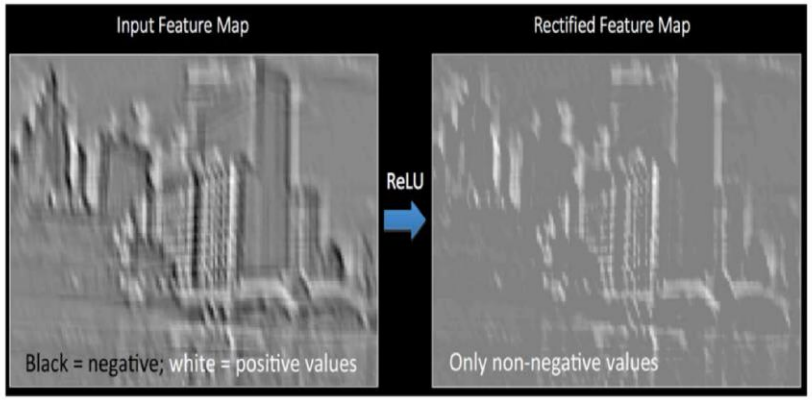


Non-Linear Activation

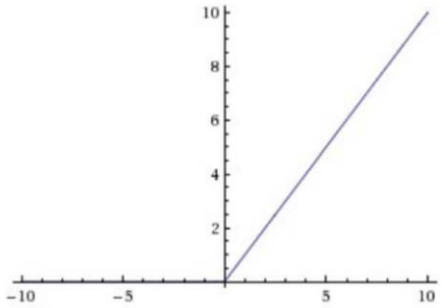


Introducing Non-Linearity

- Apply after every convolution operation (i.e., after convolutional layers)
- ReLU: pixel-by-pixel operation that replaces all negative values by zero.
- **Non-linear operation**



Rectified Linear Unit (ReLU)

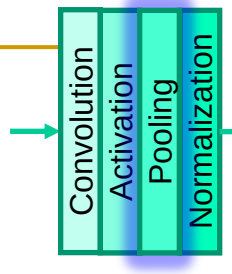


$$g(z) = \max(0, z)$$

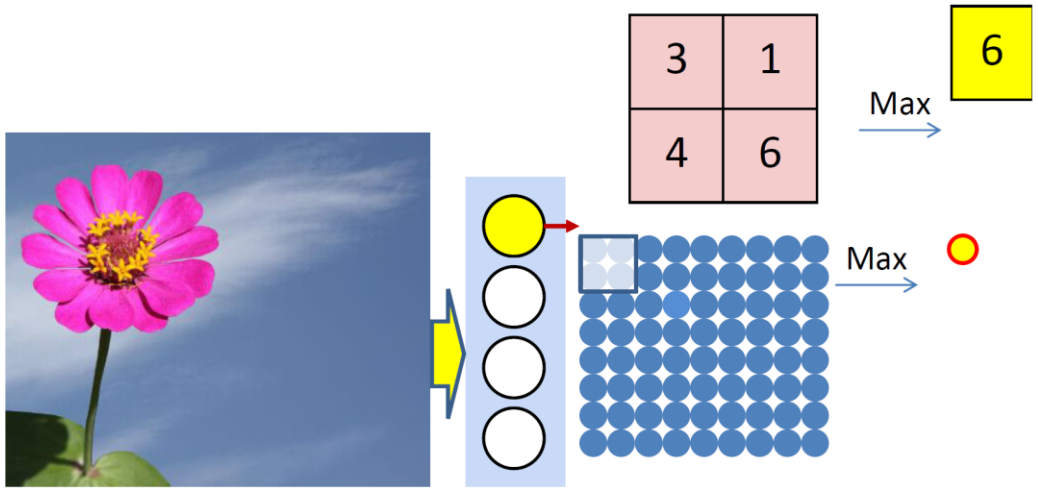
CLASS `torch.nn.ReLU(inplace=False)` [SOURCE]

Karn *Intuitive CNNs*

Pooling (i.e. down-sampling)

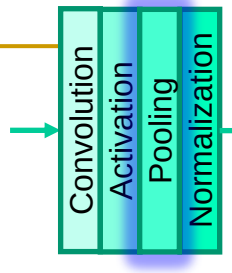


Recall: Max pooling

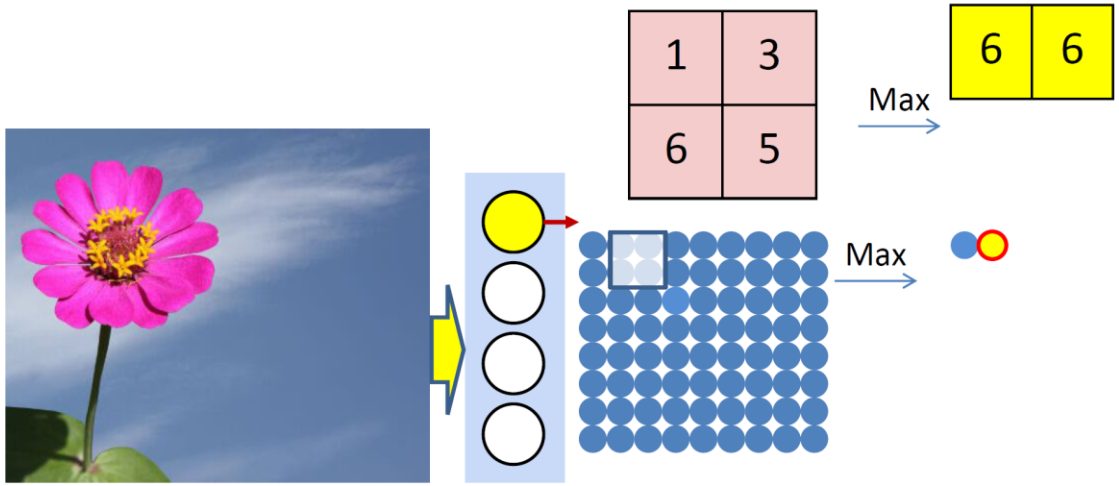


- Max pooling selects the largest from a pool of elements
- Pooling is performed by “scanning” the input

Pooling (i.e. down-sampling)

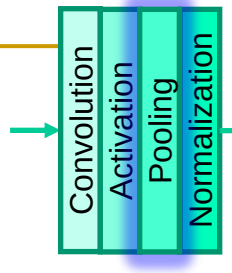


Recall: Max pooling

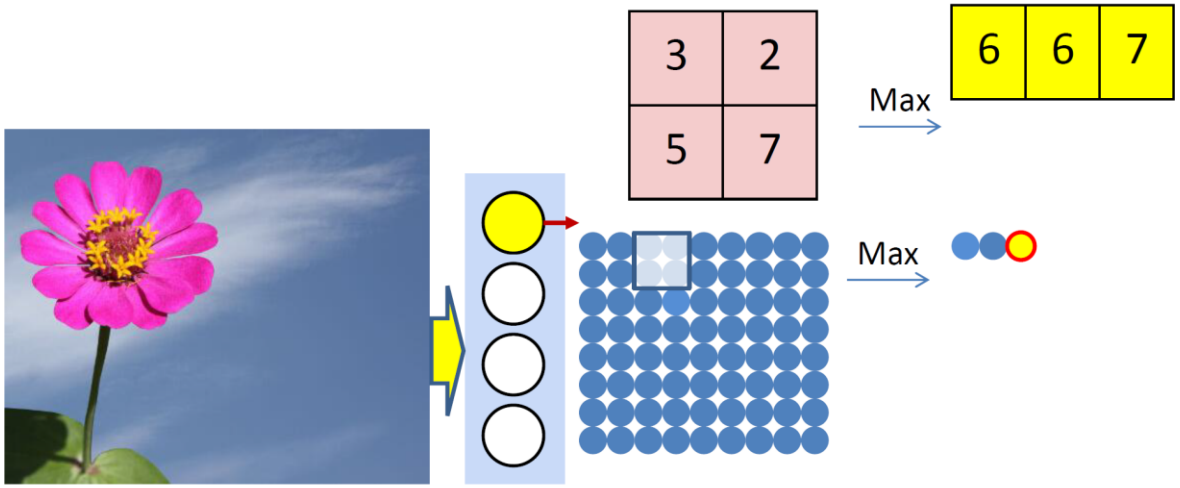


- Max pooling selects the largest from a pool of elements
- Pooling is performed by “scanning” the input

Pooling (i.e. down-sampling)

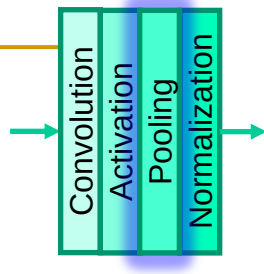


Recall: Max pooling

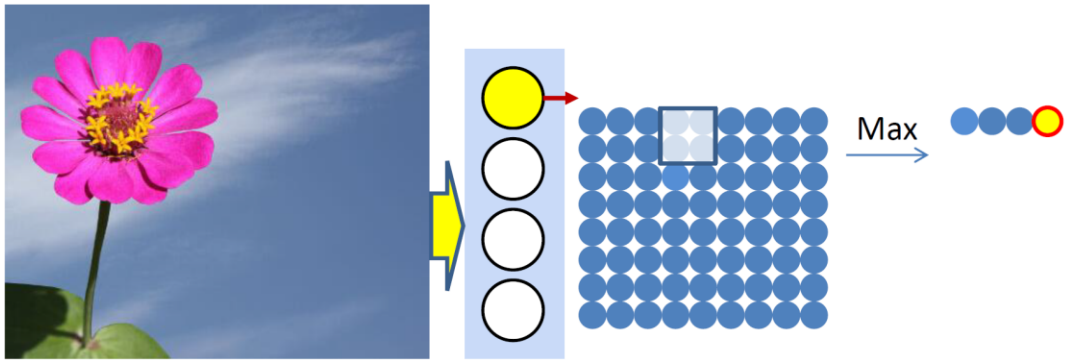


- Max pooling selects the largest from a pool of elements
- Pooling is performed by “scanning” the input

Pooling (i.e. down-sampling)

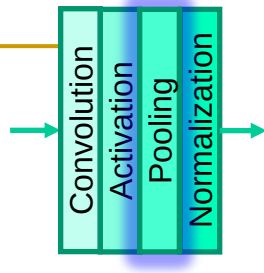


Recall: Max pooling

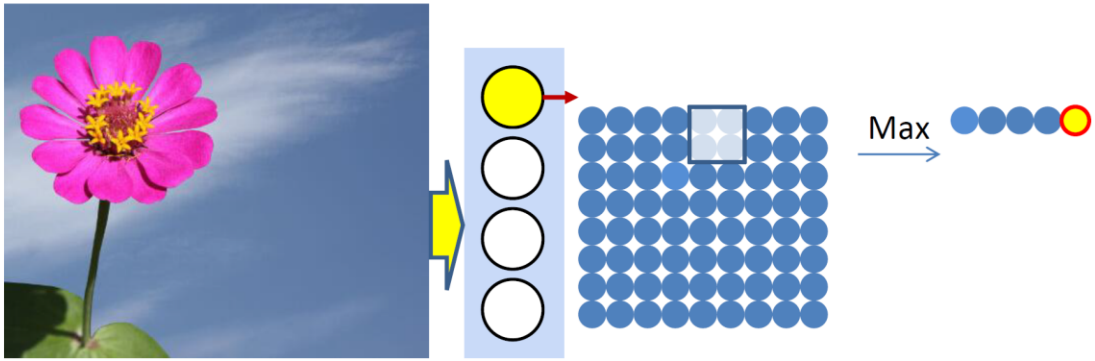


- Max pooling selects the largest from a pool of elements
- Pooling is performed by “scanning” the input

Pooling (i.e. down-sampling)

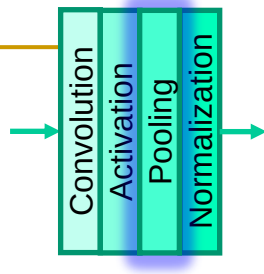


Recall: Max pooling

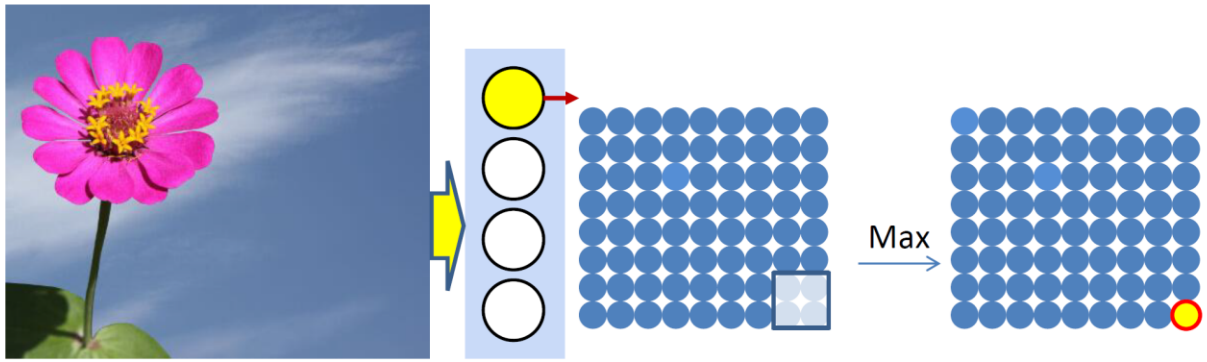


- Max pooling selects the largest from a pool of elements
- Pooling is performed by “scanning” the input

Pooling (i.e. down-sampling)

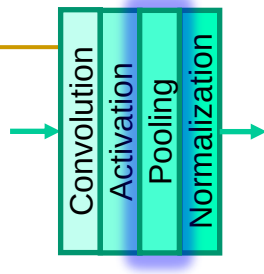


Recall: Max pooling

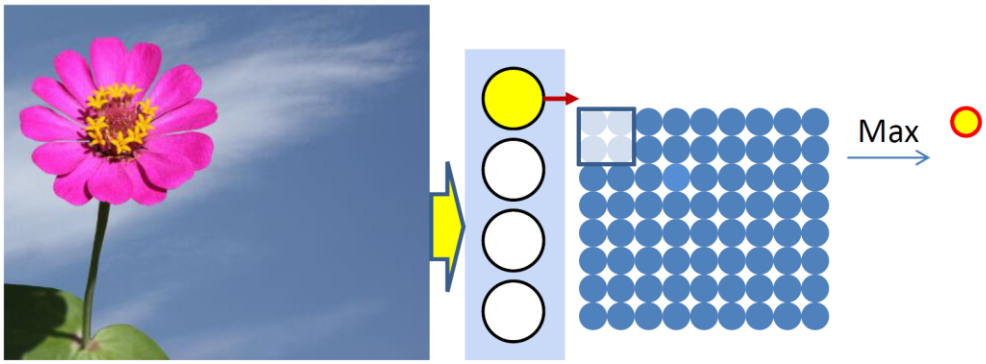


- Max pooling selects the largest from a pool of elements
- Pooling is performed by “scanning” the input

Pooling (i.e. down-sampling)

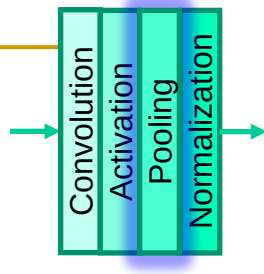


"Strides"

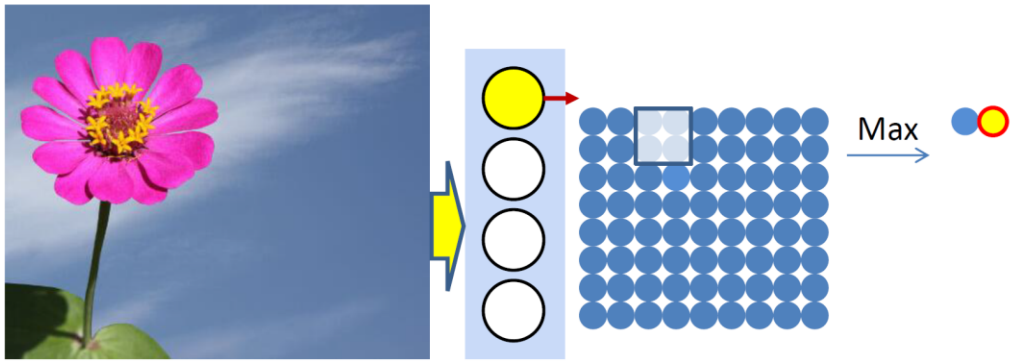


- The "max" operations may "stride" by more than one pixel

Pooling (i.e. down-sampling)

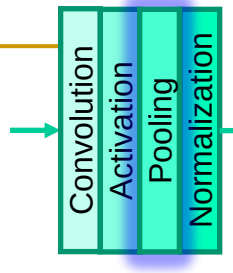


"Strides"

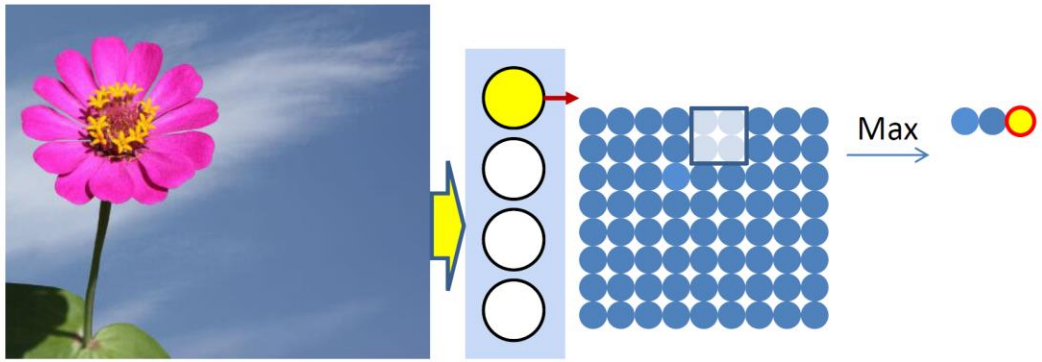


- The "max" operations may "stride" by more than one pixel

Pooling (i.e. down-sampling)

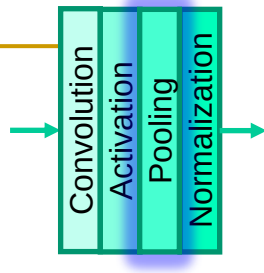


"Strides"

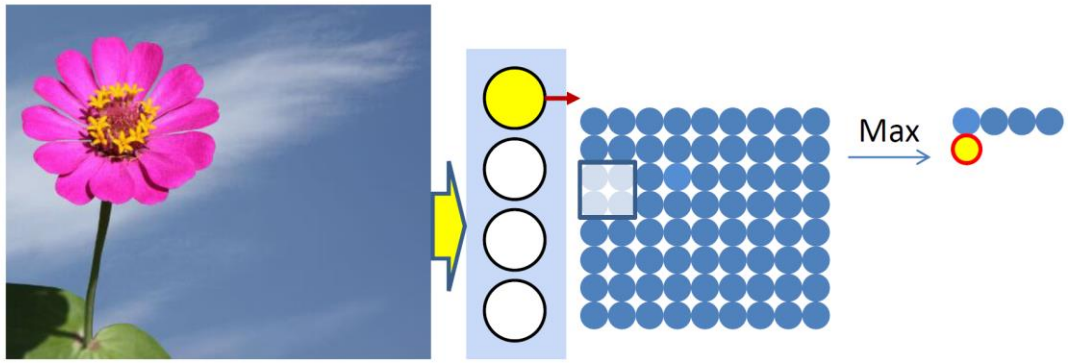


- The "max" operations may "stride" by more than one pixel

Pooling (i.e. down-sampling)

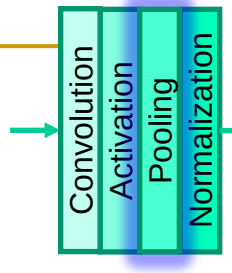


"Strides"

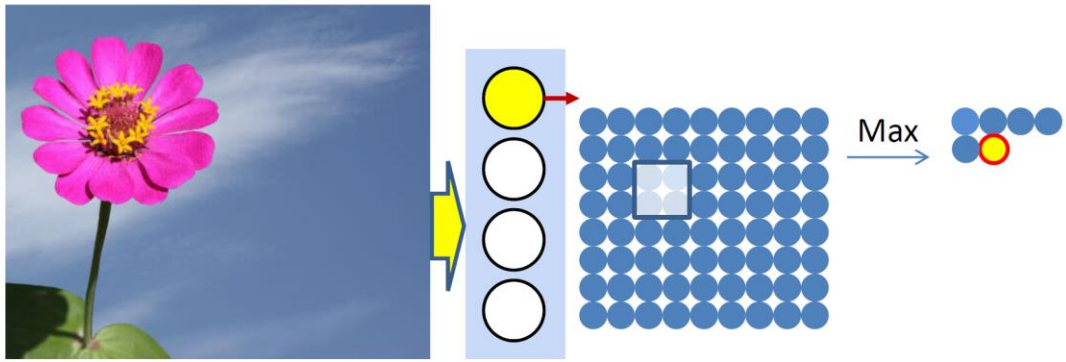


- The "max" operations may "stride" by more than one pixel

Pooling (i.e. down-sampling)

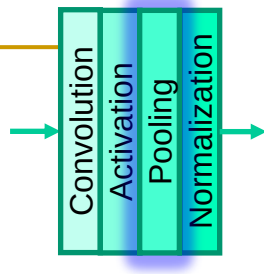


"Strides"

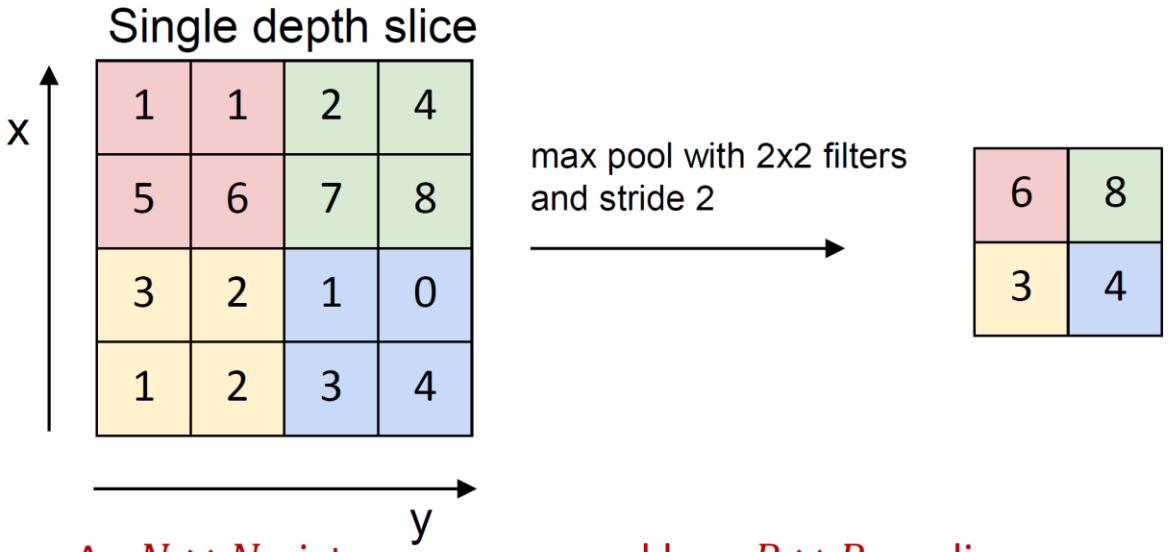


- The "max" operations may "stride" by more than one pixel

Pooling (i.e. down-sampling)

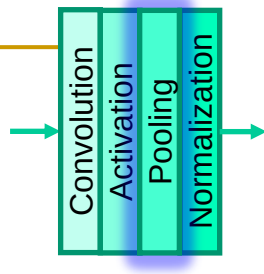


Pooling: Size of output

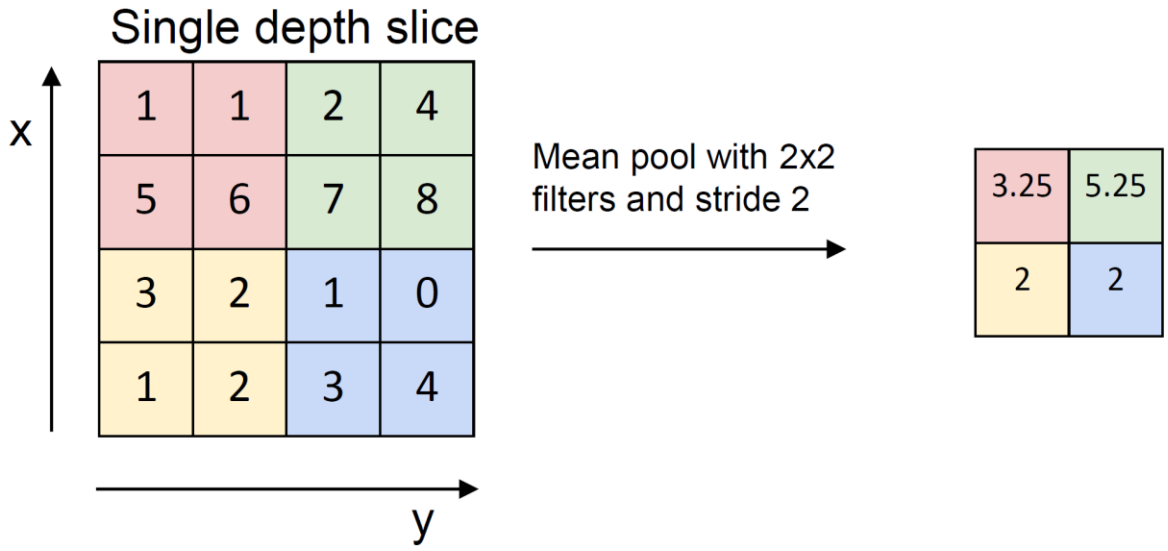


- An $N \times N$ picture compressed by a $P \times P$ pooling filter with stride D results in an output map of side $\lceil (N -$

Pooling (i.e. down-sampling)

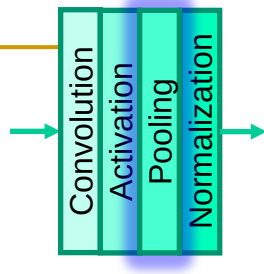


Alternative to Max pooling: Mean Pooling

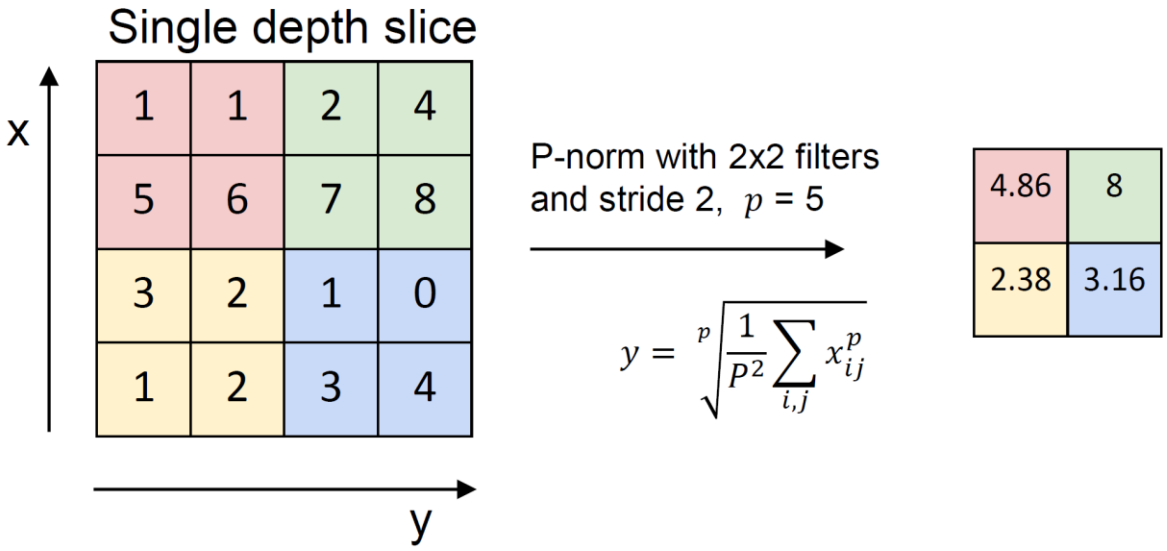


- Compute the mean of the pool, instead of the max

Pooling (i.e. down-sampling)

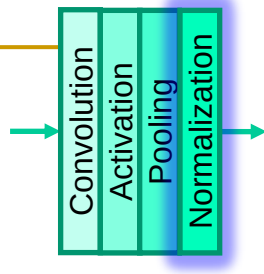


Alternative to Max pooling: P-norm



- Compute a p-norm of the pool

Batch Normalization



- BN is basically Whitening Transformation
- i.e. makes distribution close to standard normal

$$\hat{F} = \frac{F - \mu}{\sigma + \epsilon} \qquad \mu = E[F],\ \sigma = Var(F)$$

- To bring stochasticity, batch statistics are used

$$\hat{F} = \frac{F - \hat{\mu}}{\hat{\sigma} + \epsilon} \qquad \hat{\mu} = \frac{1}{B}\sum_{k=1}^{B} F_k\,, \quad \hat{\sigma} = \frac{1}{B}\sum_{k=1}^{B} (F_k - \hat{\mu})^2$$

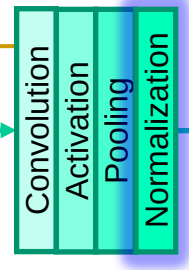
- We make BN layer to learn its scaling/bias adjusting factors

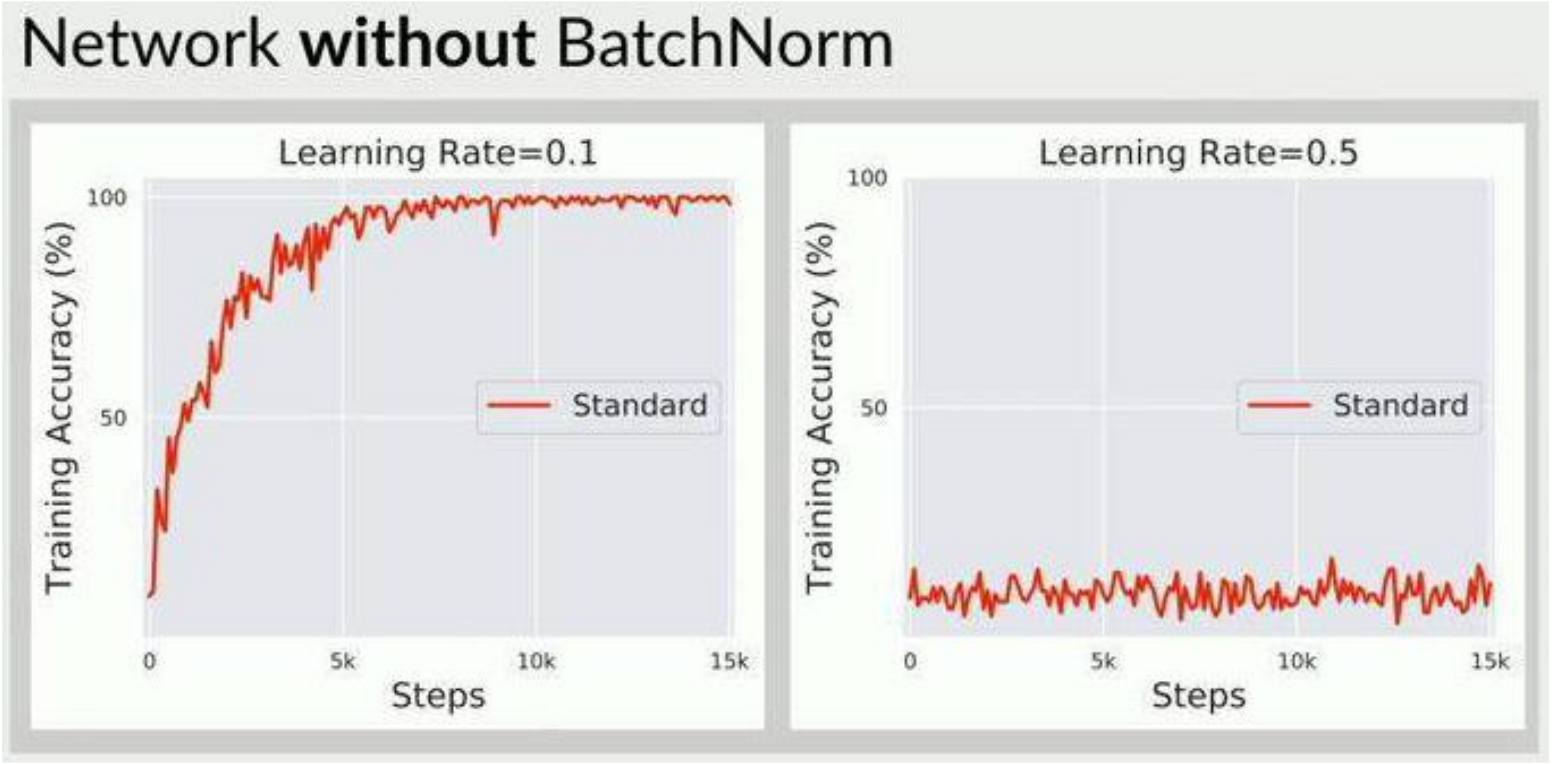
$$\hat{F} = \gamma \frac{F - \hat{\mu}}{\hat{\sigma} + \epsilon} + \beta$$

Learnable parameter: scaling

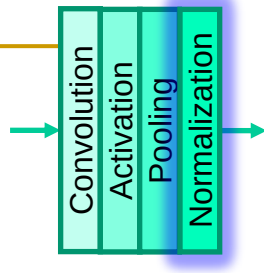
Learnable parameter: bias

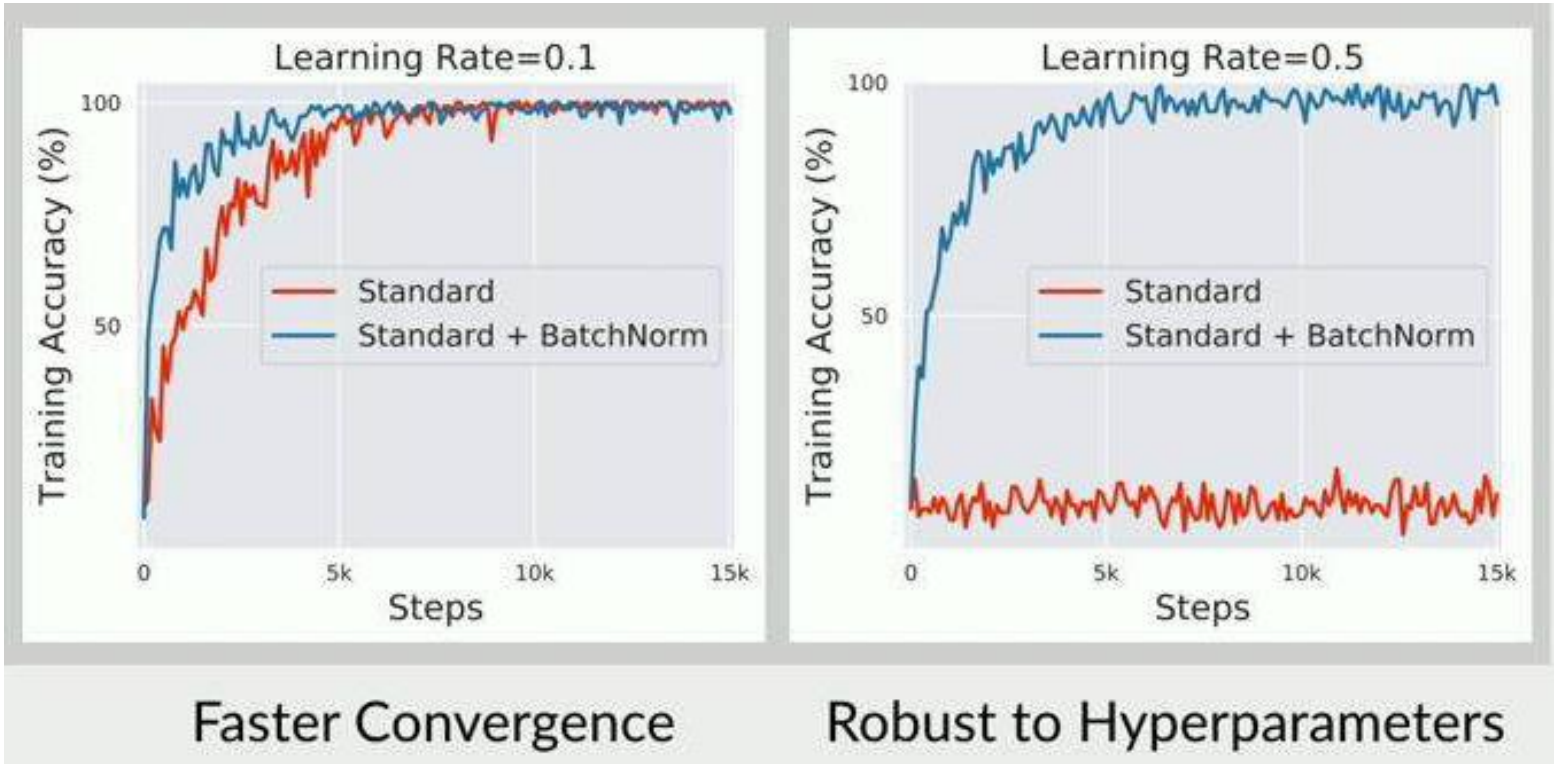
Why Do We Use BN?



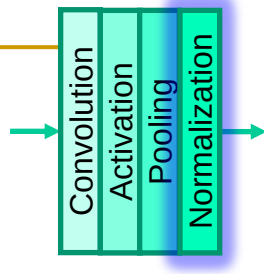


Why Do We Use BN?

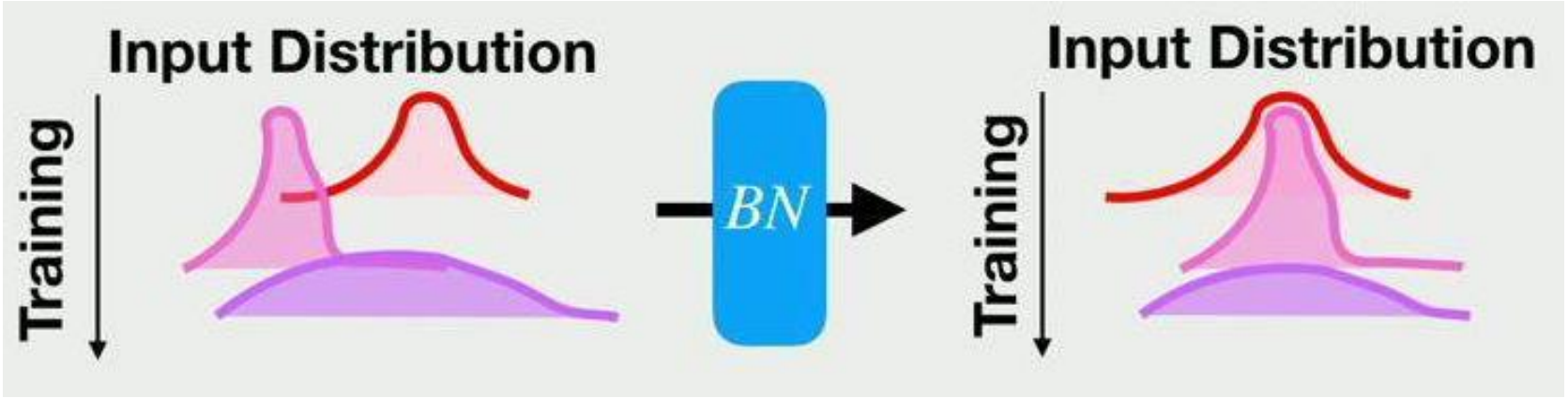




How BN Helps?



- BN reduces the internal covariate shift



Today

1. Motivation
2. Feature Learning
3. Building Block Design of CNN
   - Convolutional Layer
   - Activation Function
   - Pooling Layer
   - Normalization Layer