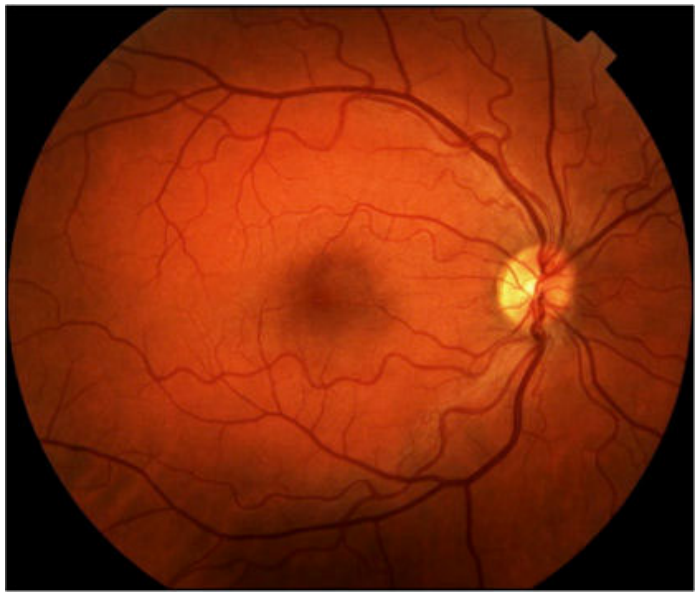
Deep Convolutional Neural Nets
Part I

Tim Dunn

Duke MLSS 2019

Deep Learning for Image Analysis

Diabetic Retinopathy Classification

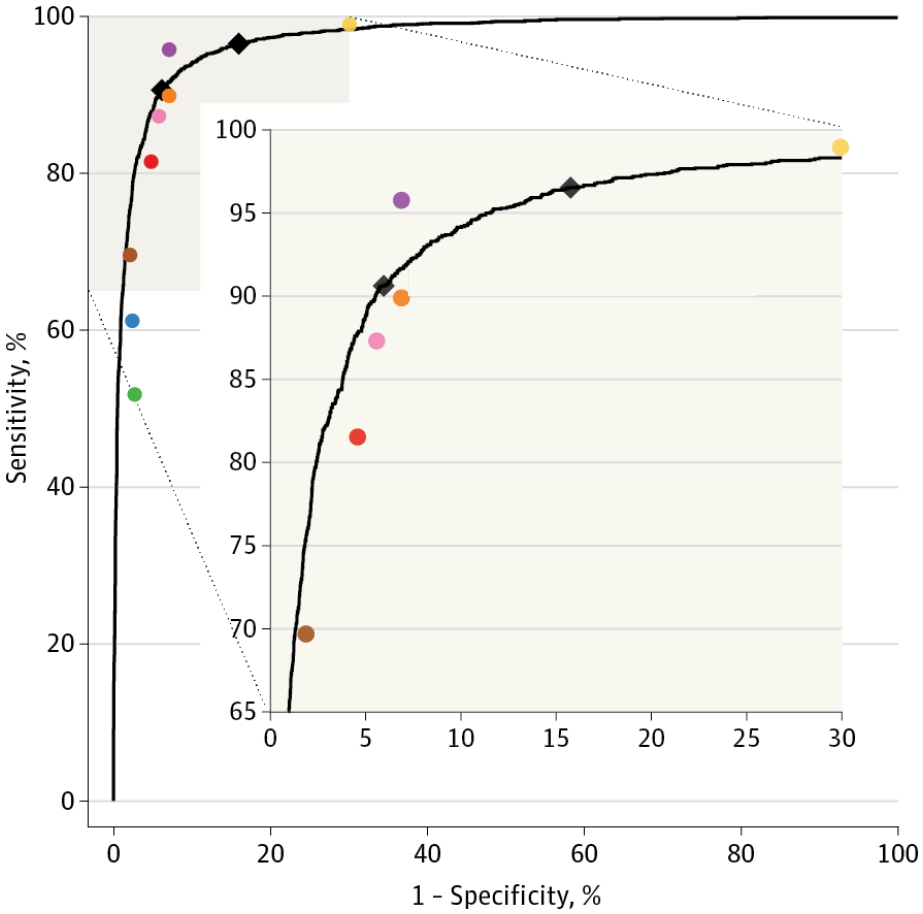


Healthy Retina

Unhealthy Retina

Deep Learning for Image Analysis

Diabetic Retinopathy Classification



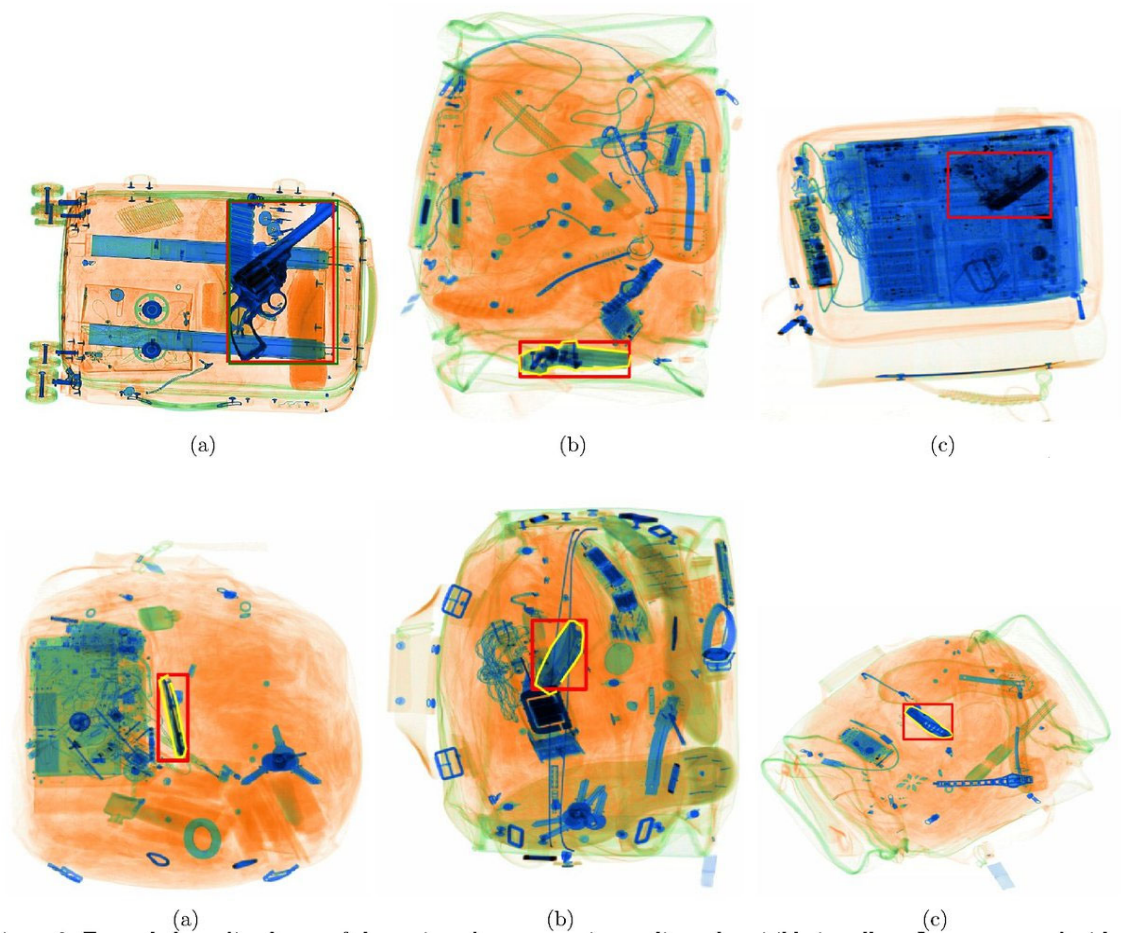
$$\text{sensitivity} = \frac{\text{number of true positives}}{\text{total number of positives in the dataset}}$$

$$\text{specificity} = \frac{\text{number of true negatives}}{\text{total number of negatives in the dataset}}$$

Gulshan et al. *JAMA* (2016)

Deep Learning for Image Analysis

TSA Screening



Liang et al. *SPIE* (2018)

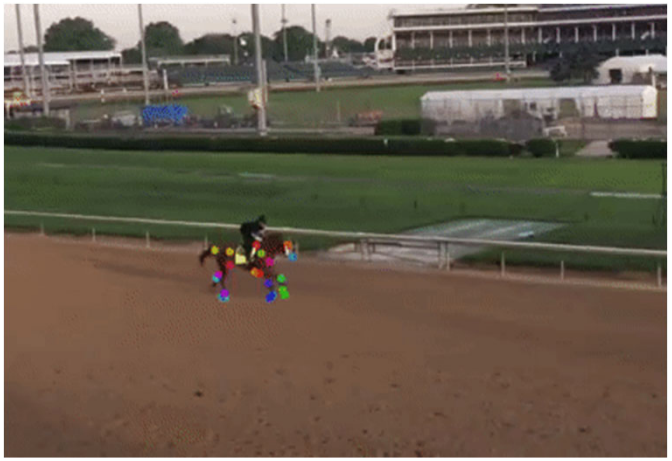
Deep Learning for Image Analysis

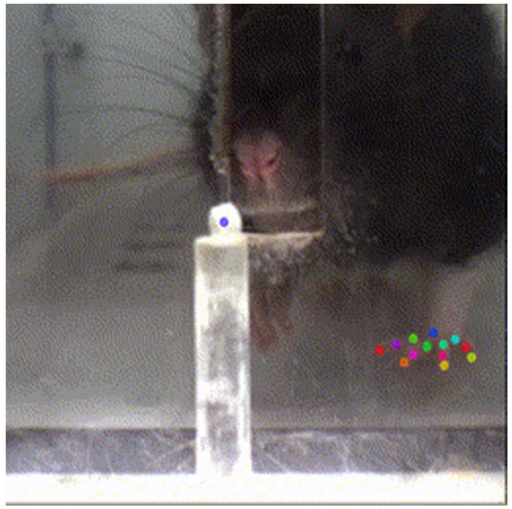
Markerless Motion Capture: Automatic 3D Surface Meshes from Video

DensePose (Facebook)

Deep Learning for Image Analysis

Markerless Motion Capture – Body Landmarks

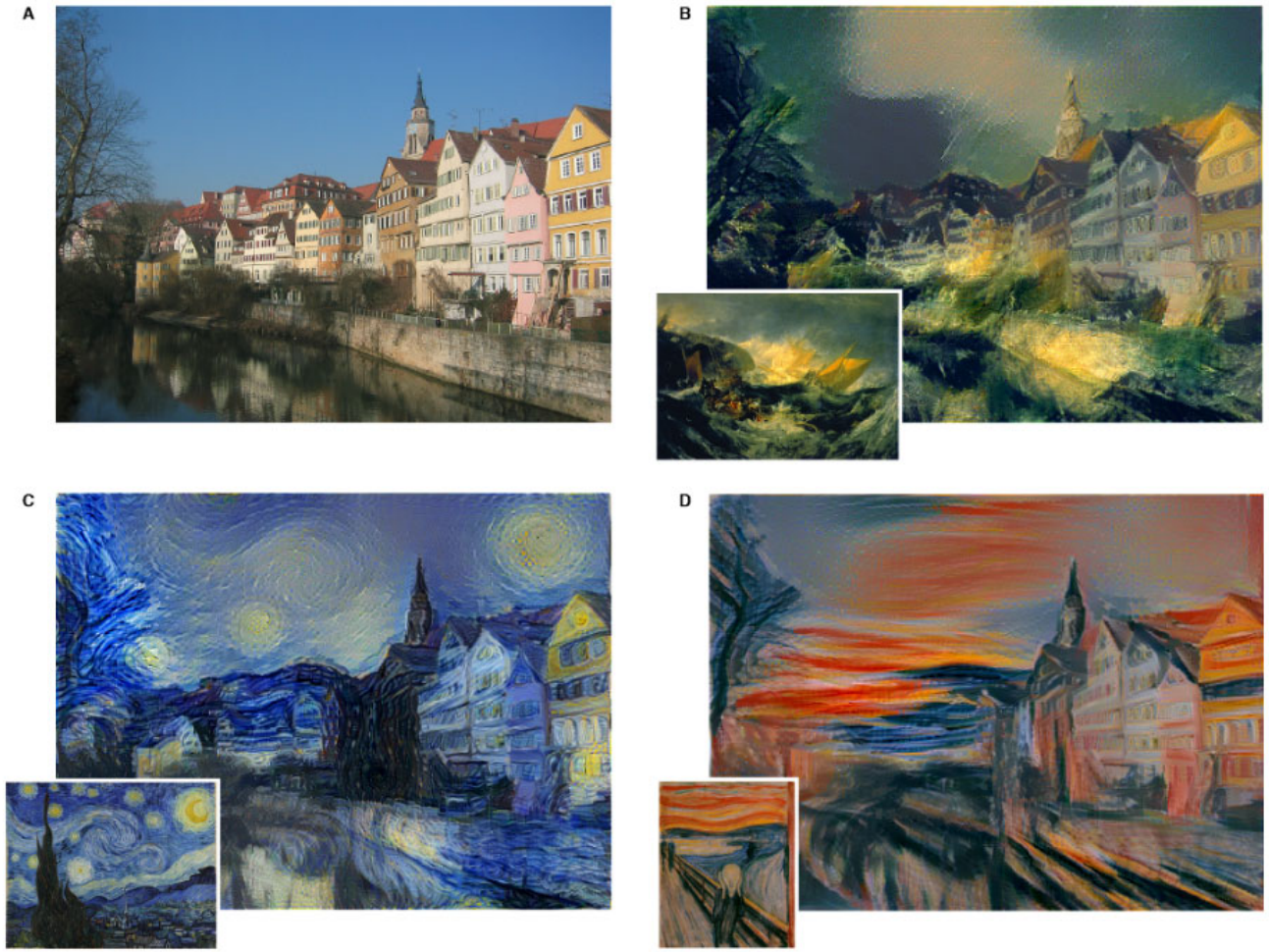




Mathis et al. *Nature Neuroscience* (2018)

Deep Learning for Image Analysis

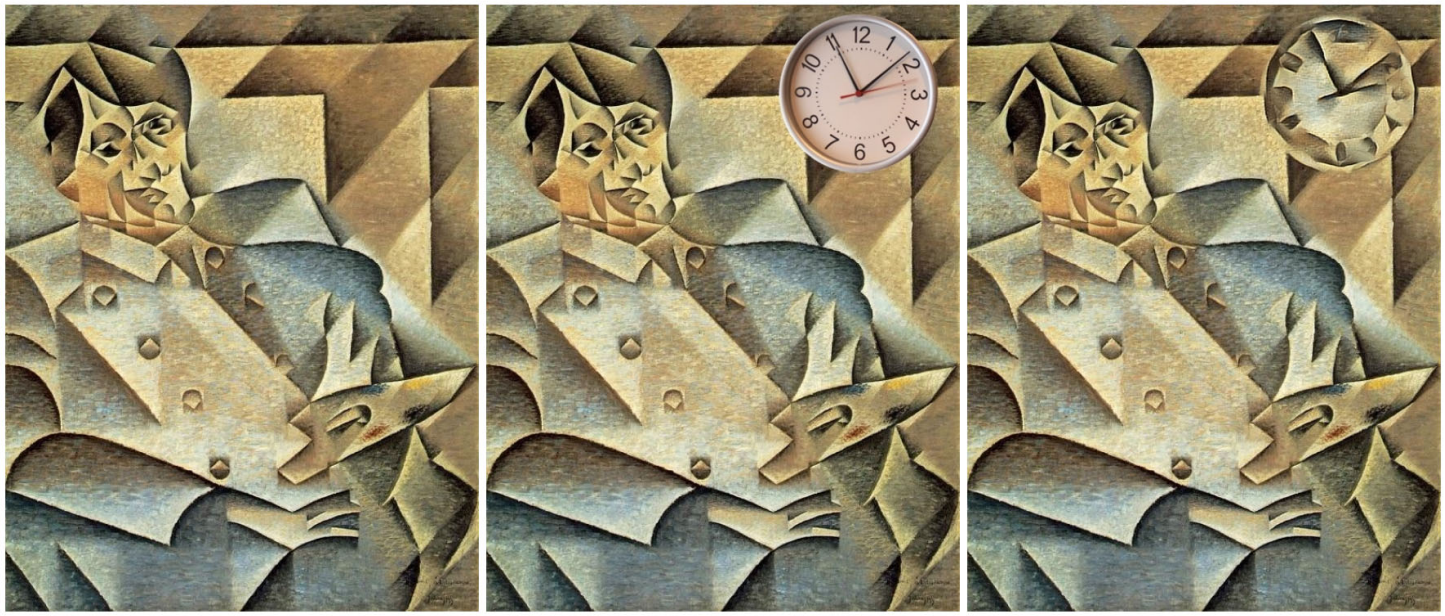
Style Transfer and Harmonization



Gatys et al. A Neural Algorithm of Artistic Style. *arXiv* (2015)

Deep Learning for Image Analysis

Style Transfer and Harmonization



Luan et al. Deep Painterly Harmonization. *arXiv* (2018)

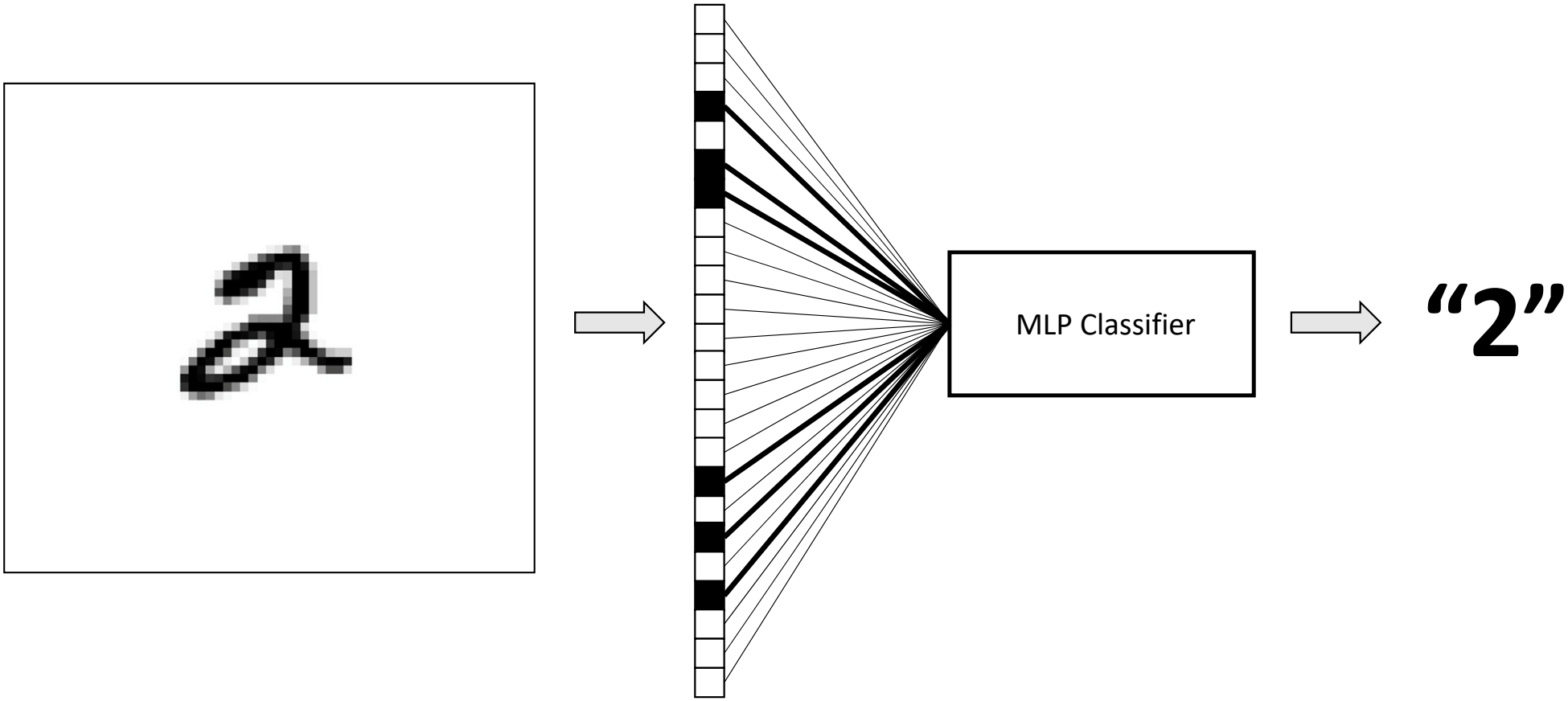
Deep Learning for Image Analysis

Style Transfer and Harmonization

Luan et al. Deep Painterly Harmonization. *arXiv* (2018)

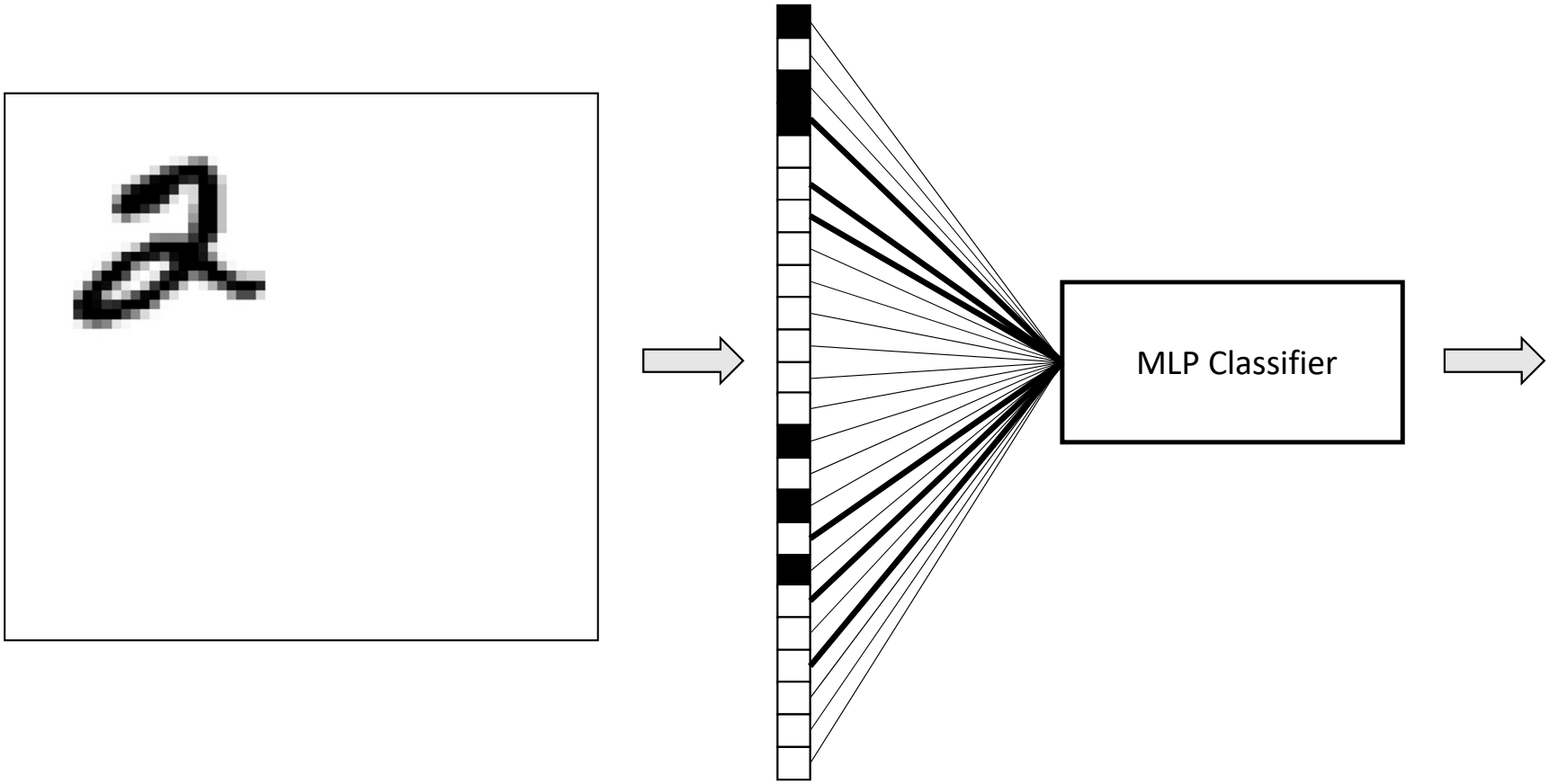
Convolutional Neural Networks Find Structure Anywhere In Images

Consider the multi-layer perceptron for digit recognition:



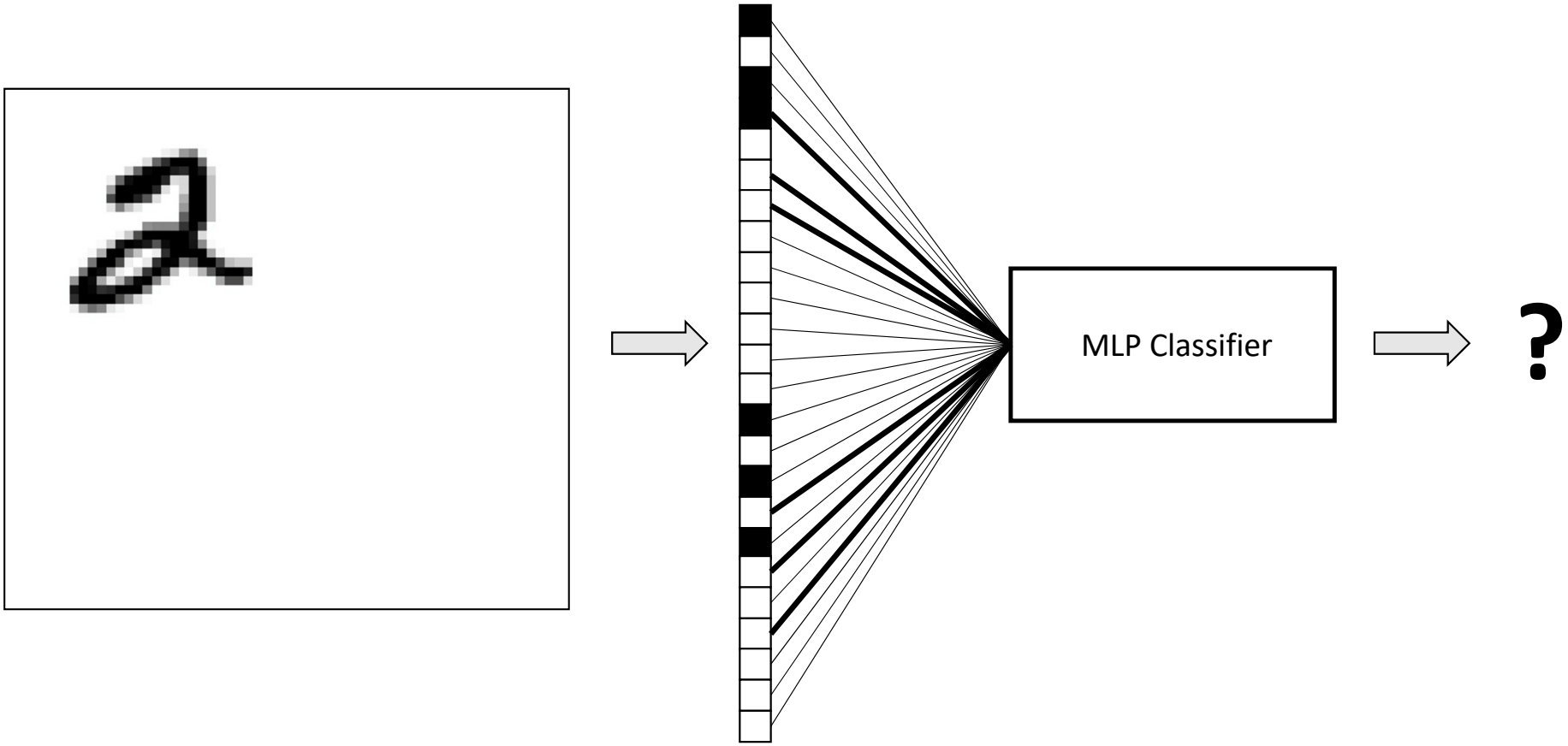
Convolutional Neural Networks Find Structure Anywhere In Images

Consider the multi-layer perceptron for digit recognition:



Convolutional Neural Networks Find Structure Anywhere In Images

Consider the multi-layer perceptron for digit recognition:



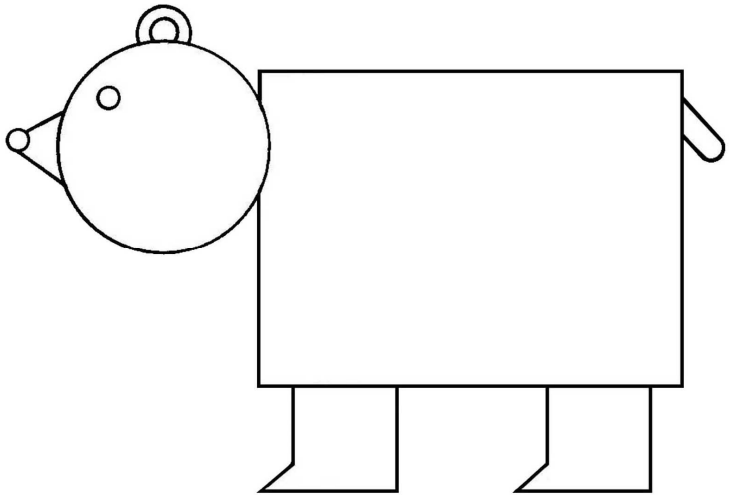
Convolutional Neural Networks Find Structure Anywhere In Images

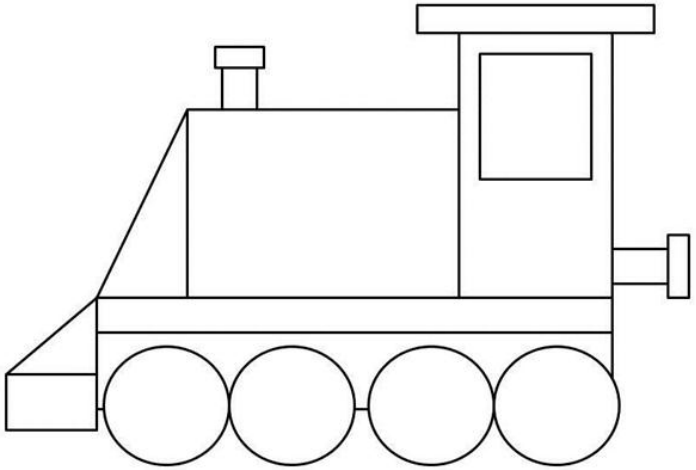
CNN Classifier

“2”

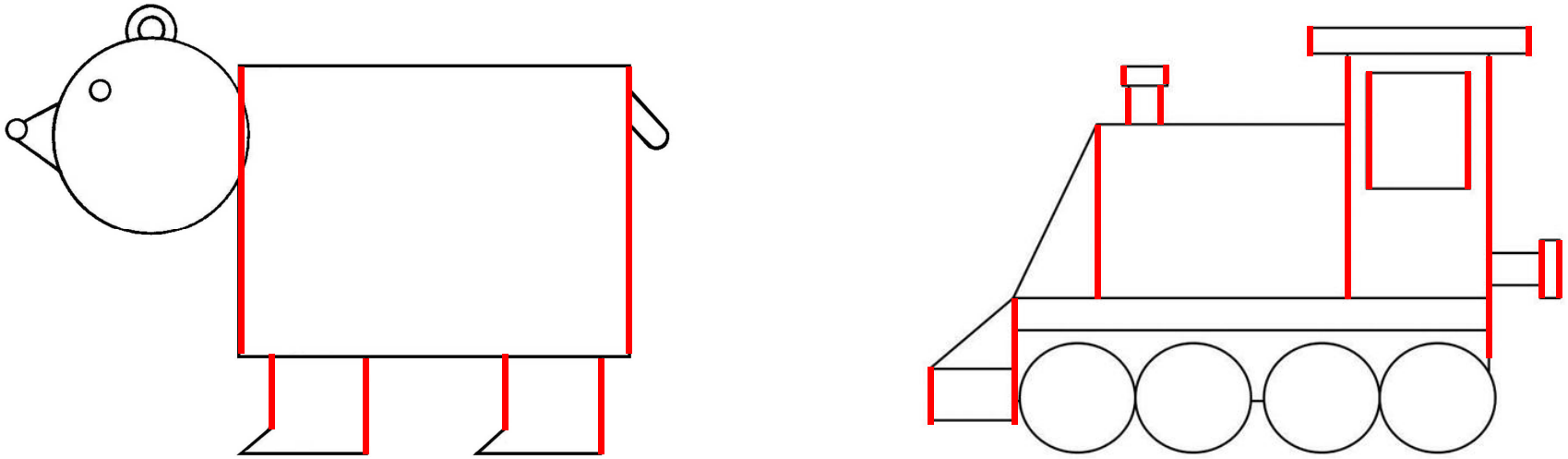
CNNs Take Advantage of Repeated, Hierarchical Structure in Images

CNNs Take Advantage of Repeated, Hierarchical Structure in Images



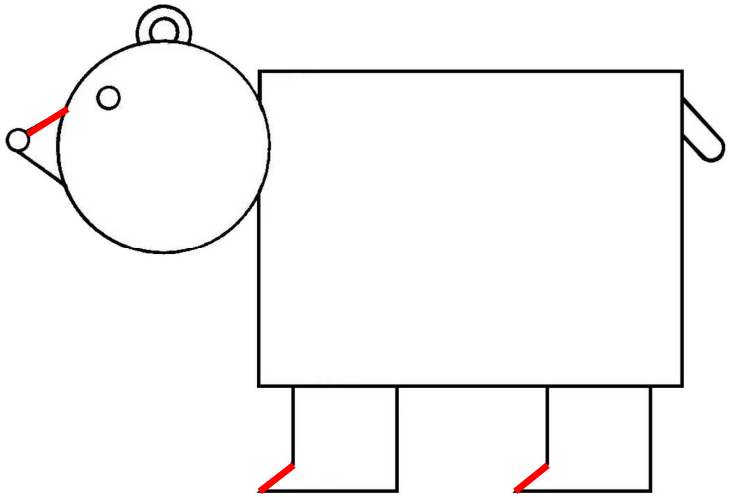


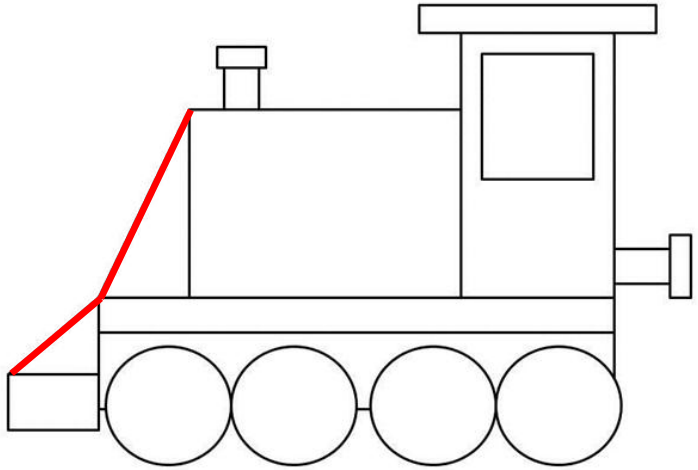
CNNs Take Advantage of Repeated, Hierarchical Structure in Images



Low-level structure: lines, curves

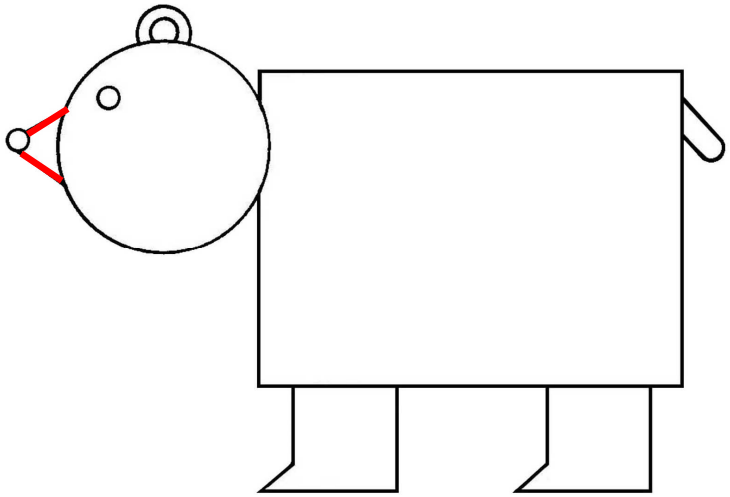
CNNs Take Advantage of Repeated, Hierarchical Structure in Images

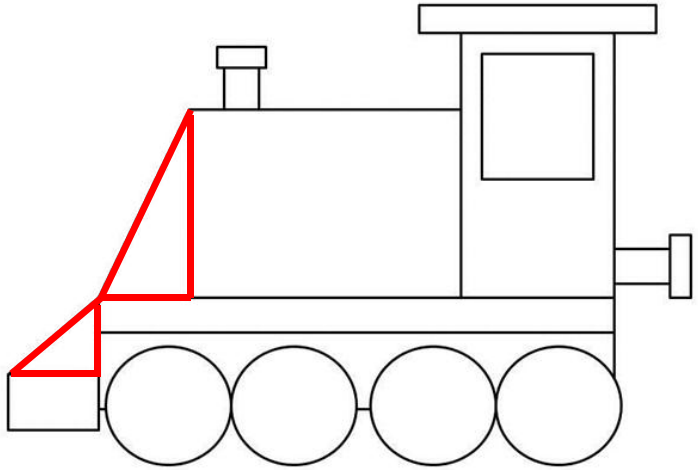




Low-level structure: lines, curves

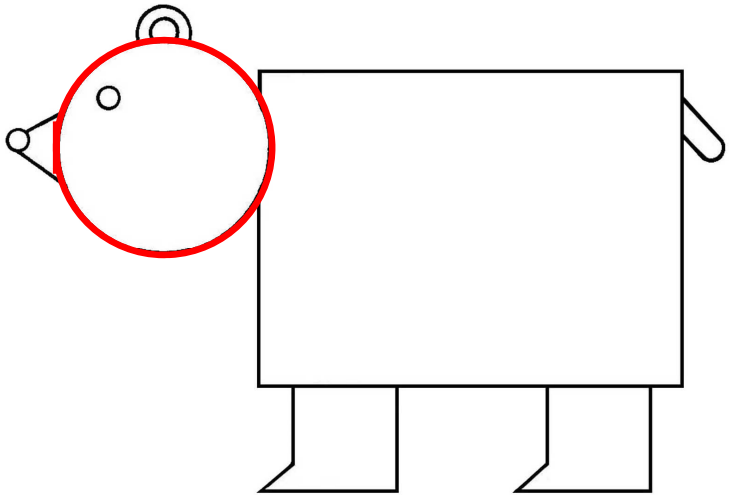
CNNs Take Advantage of Repeated, Hierarchical Structure in Images

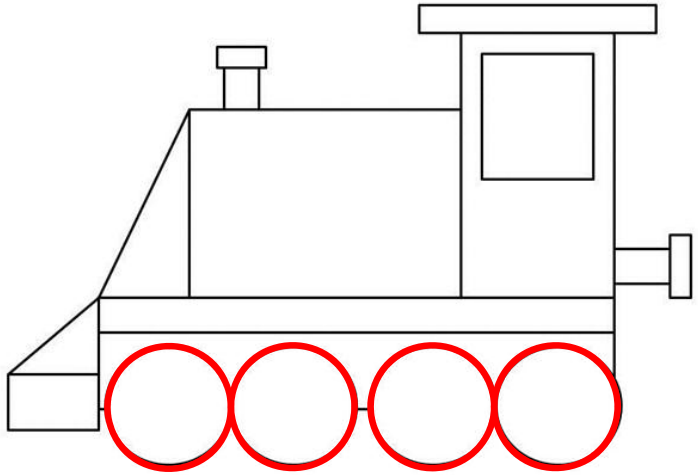




Mid-level structure: shapes

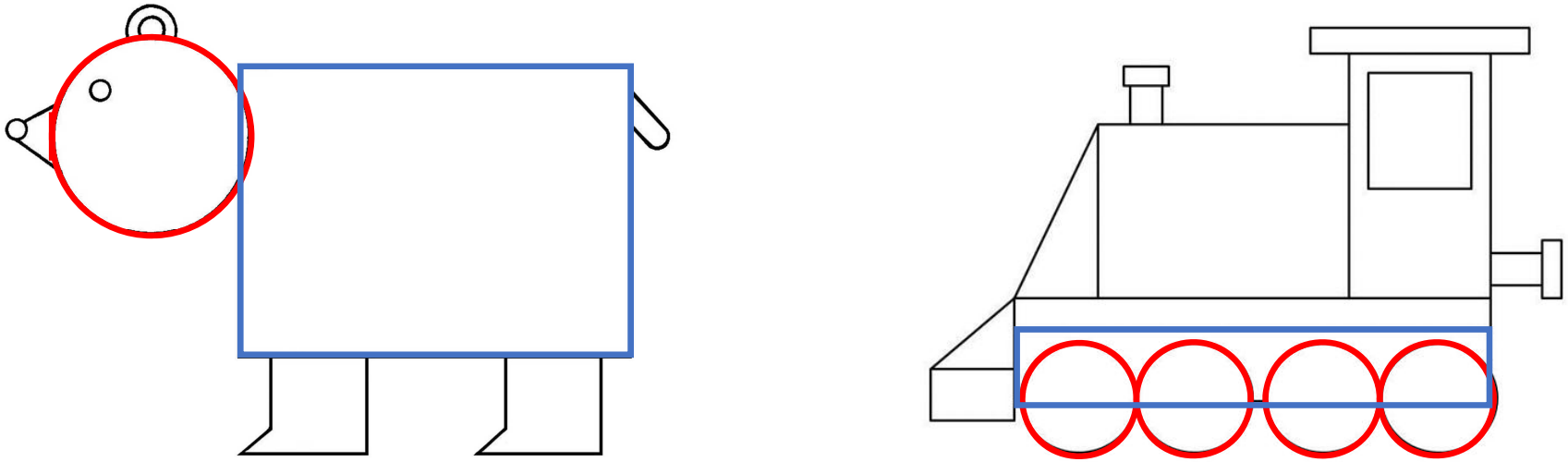
CNNs Take Advantage of Repeated, Hierarchical Structure in Images





Mid-level structure: shapes

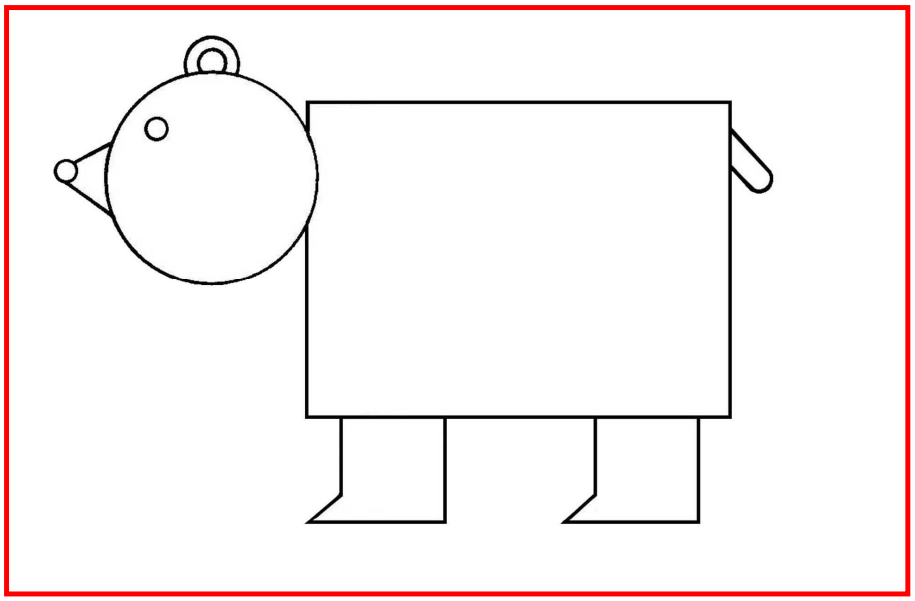
CNNs Take Advantage of Repeated, Hierarchical Structure in Images



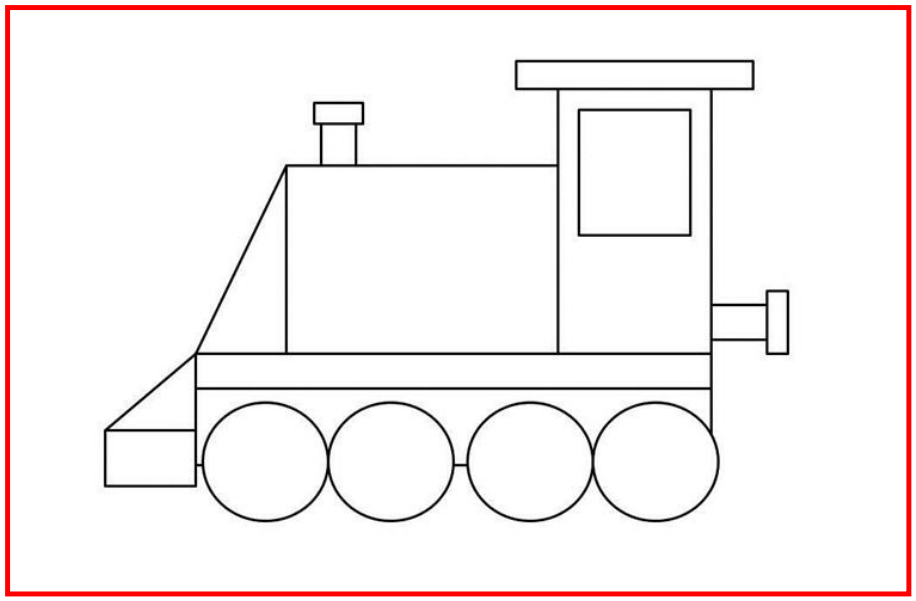
High-level structure: groups of shapes

CNNs Take Advantage of Repeated, Hierarchical Structure in Images

Bear

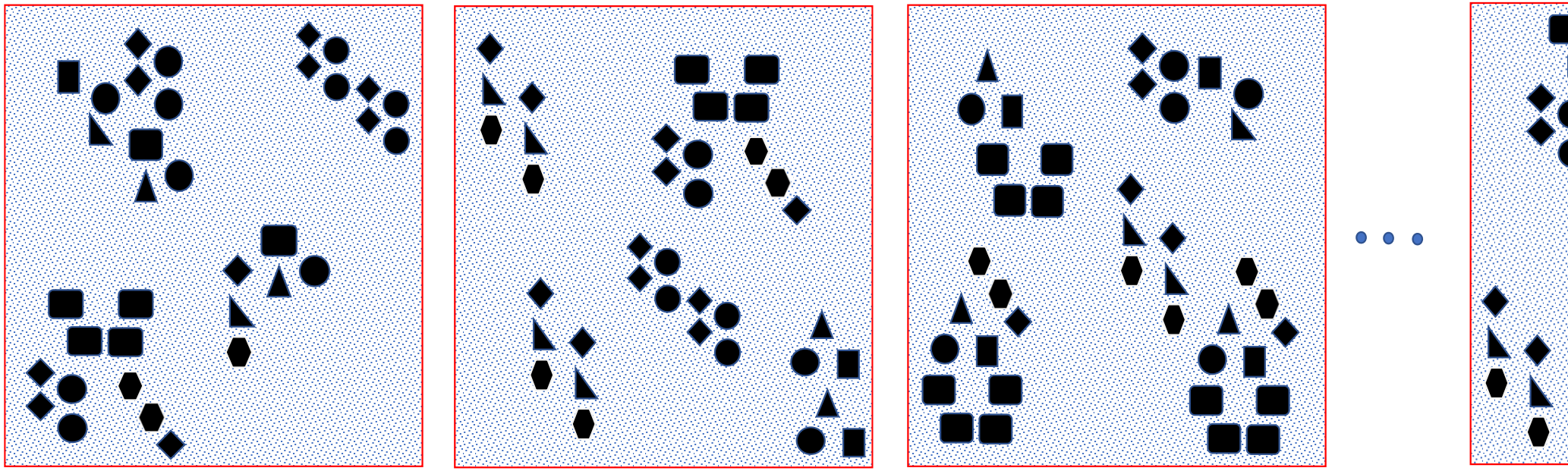


Train

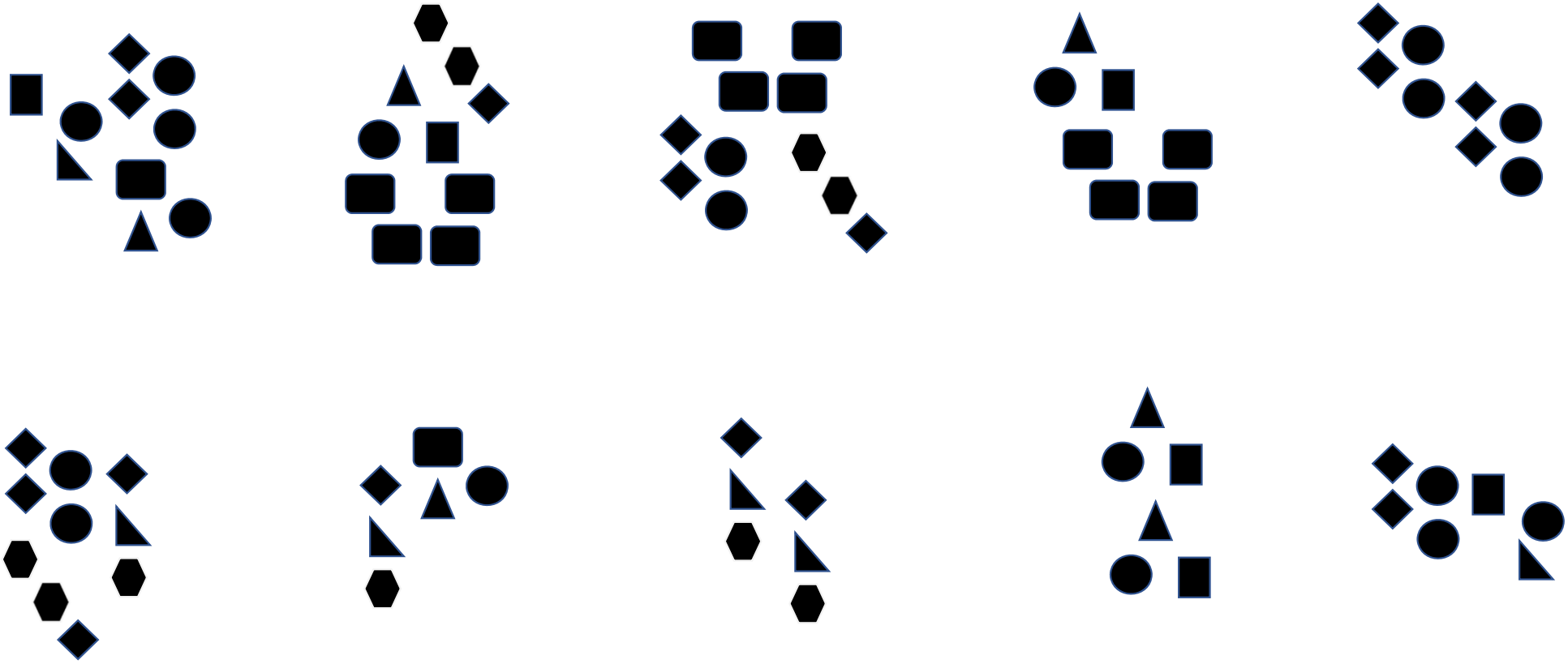


High-level structure: groups of shapes → objects

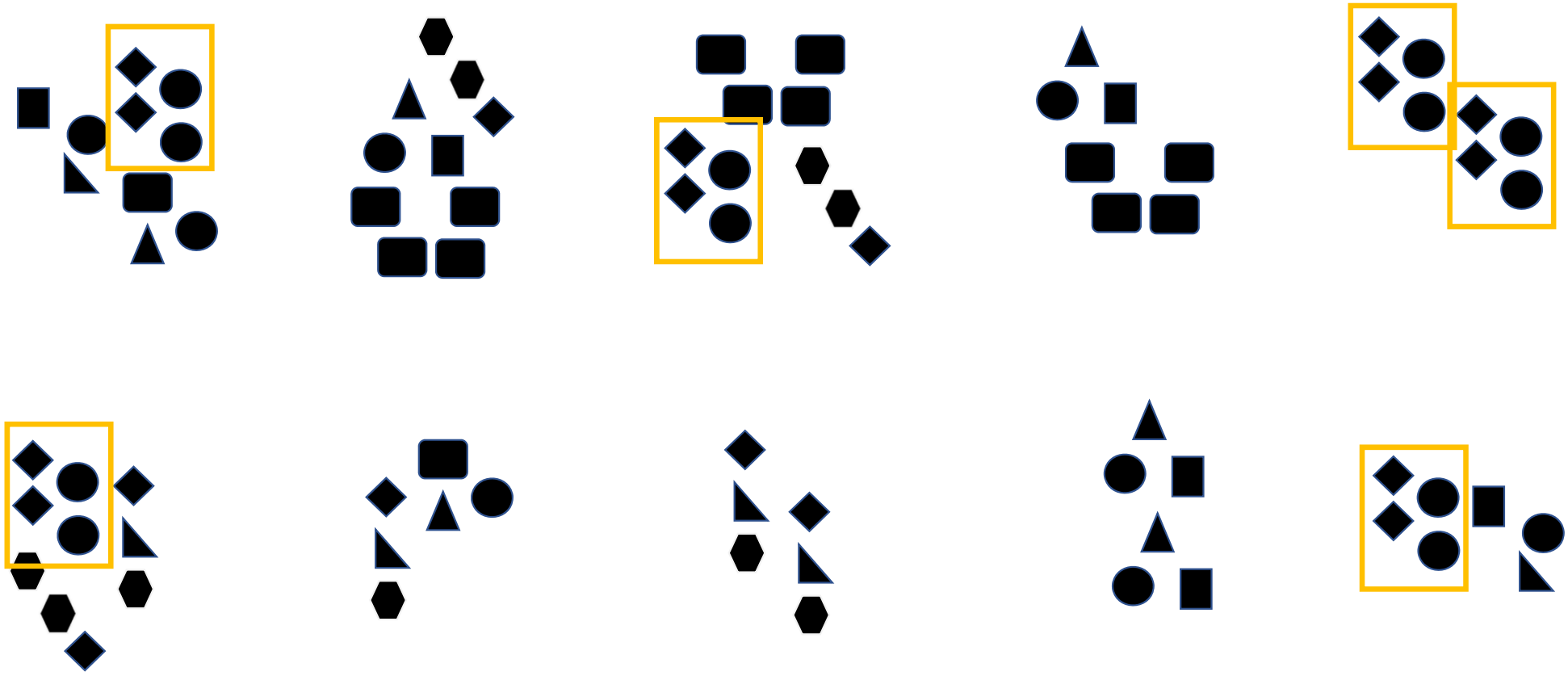
Consider a Set of “Toy” Images,
for illustration of how this structure can be extracted by an algorithm



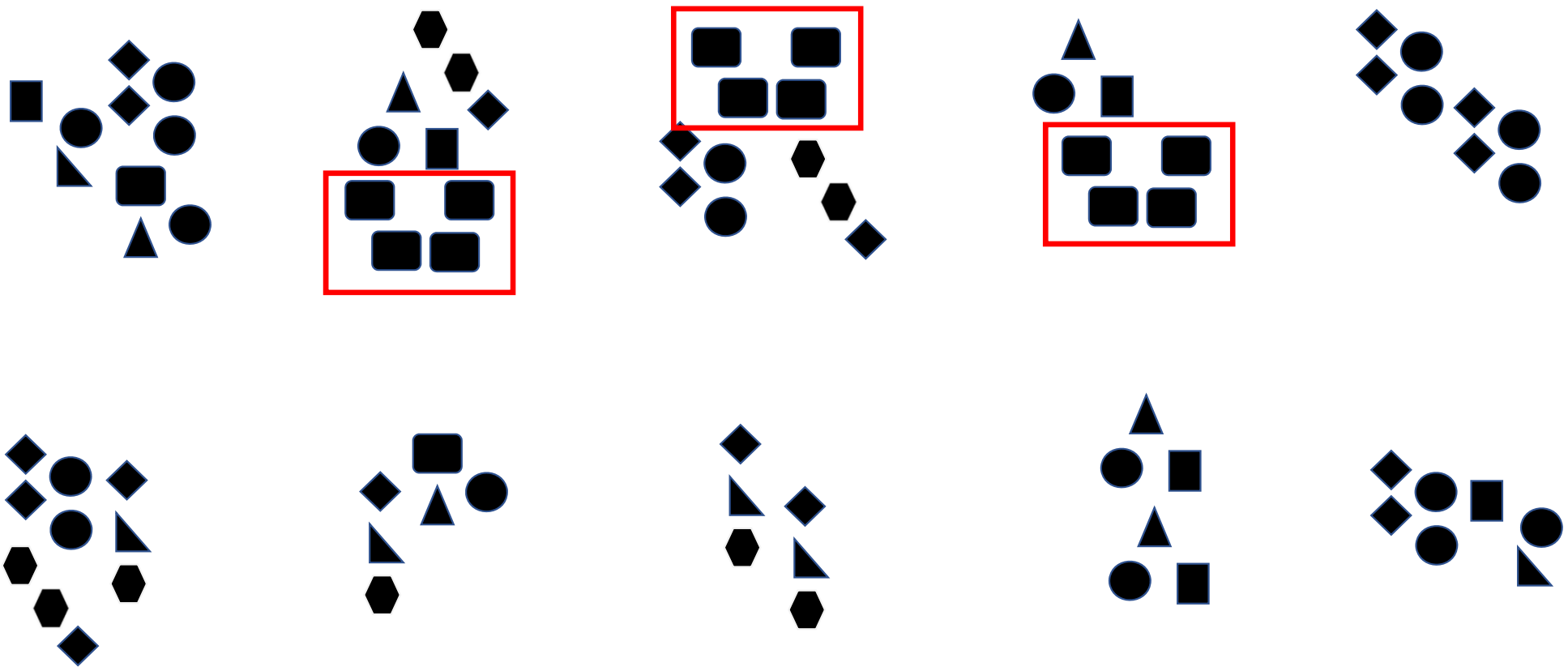
High-Level Motifs/Structure



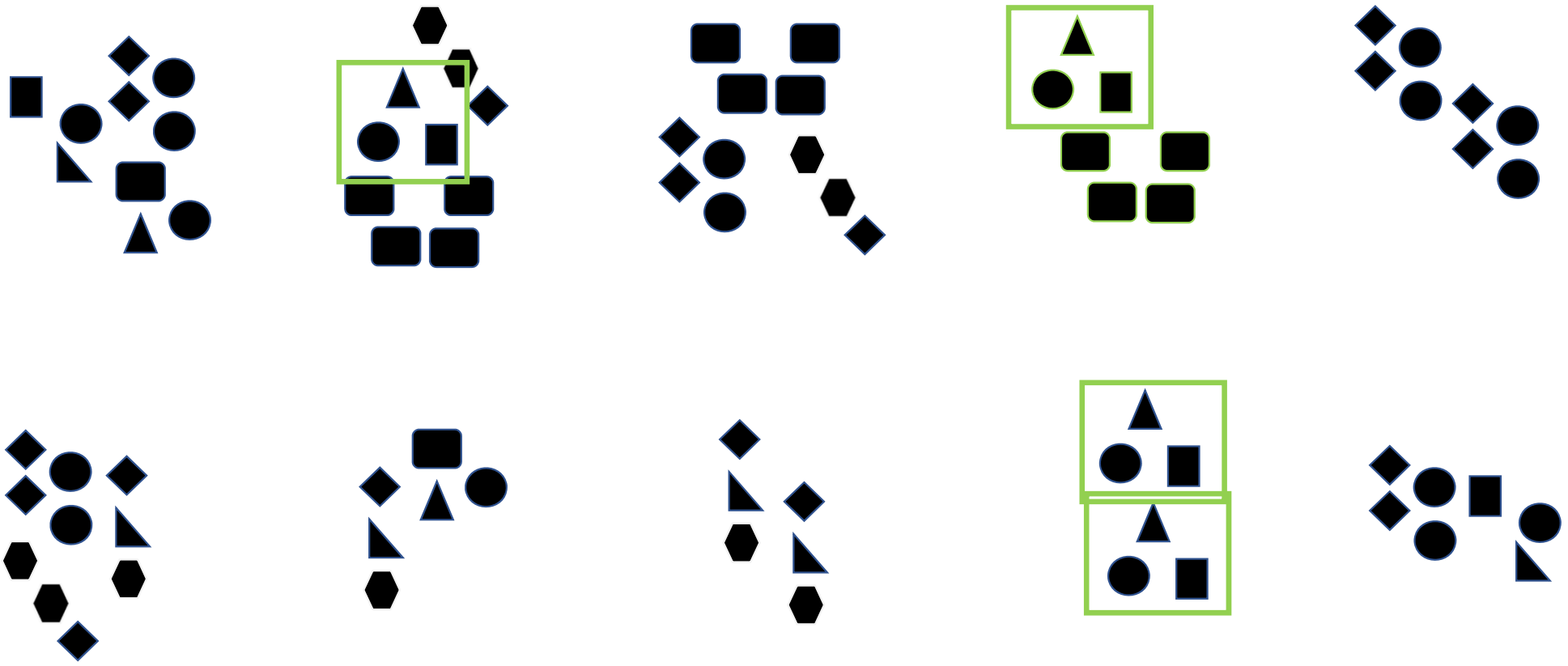
Shared Substructure Within Motifs



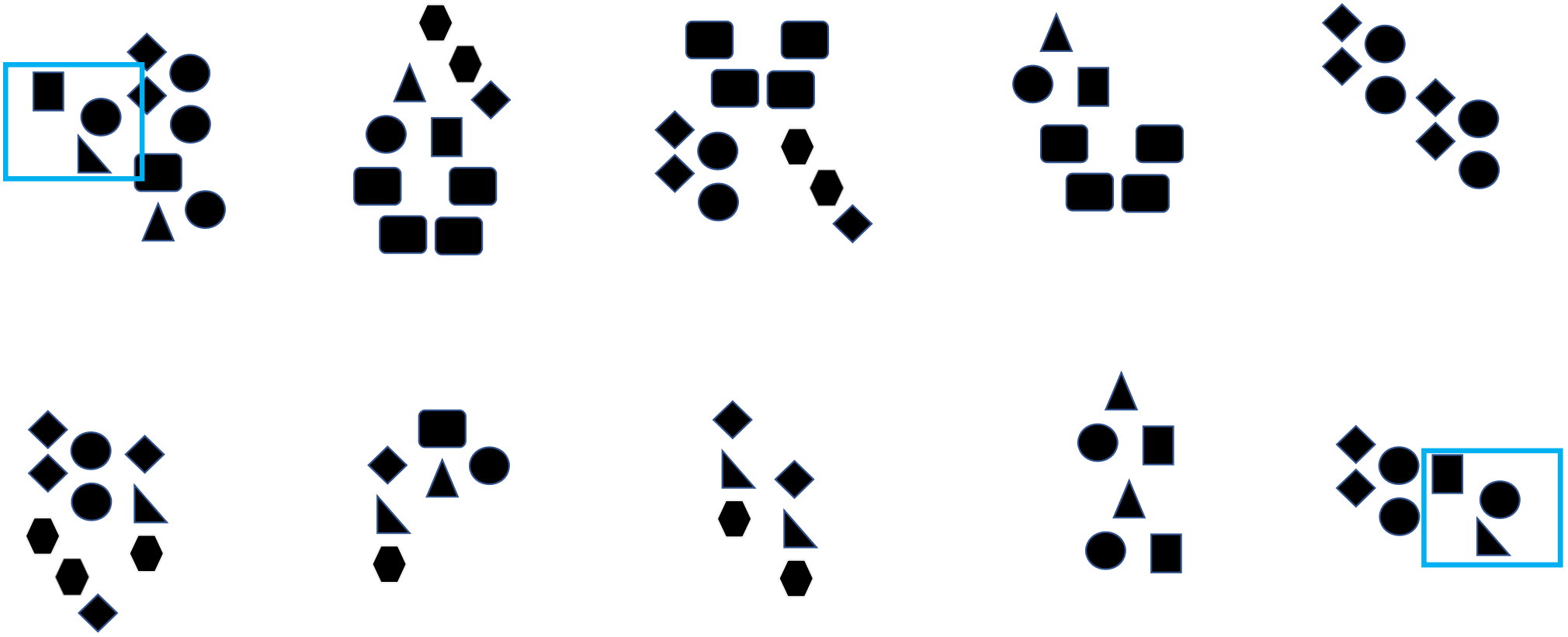
Shared Substructure Within Motifs



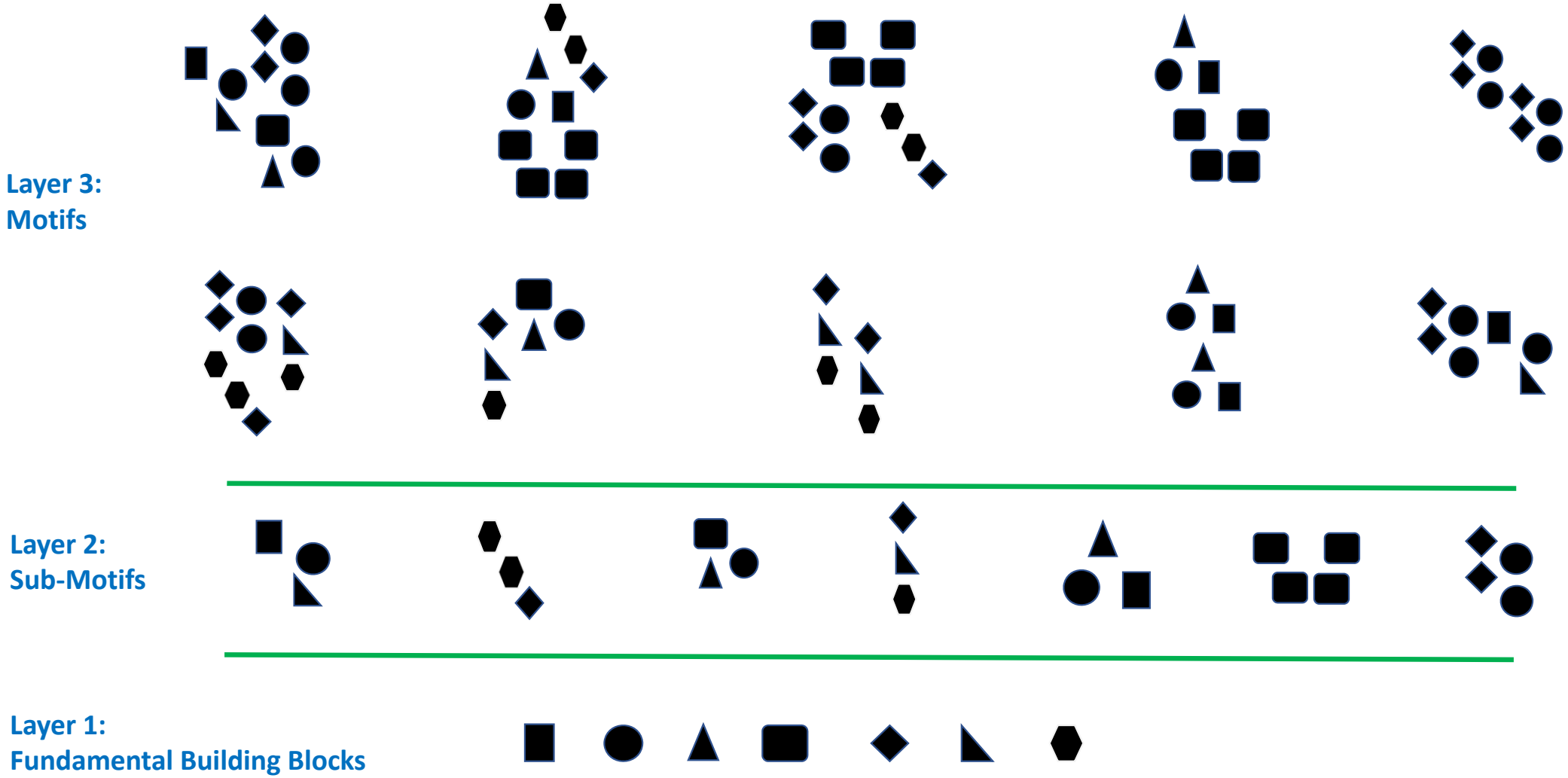
Shared Substructure Within Motifs



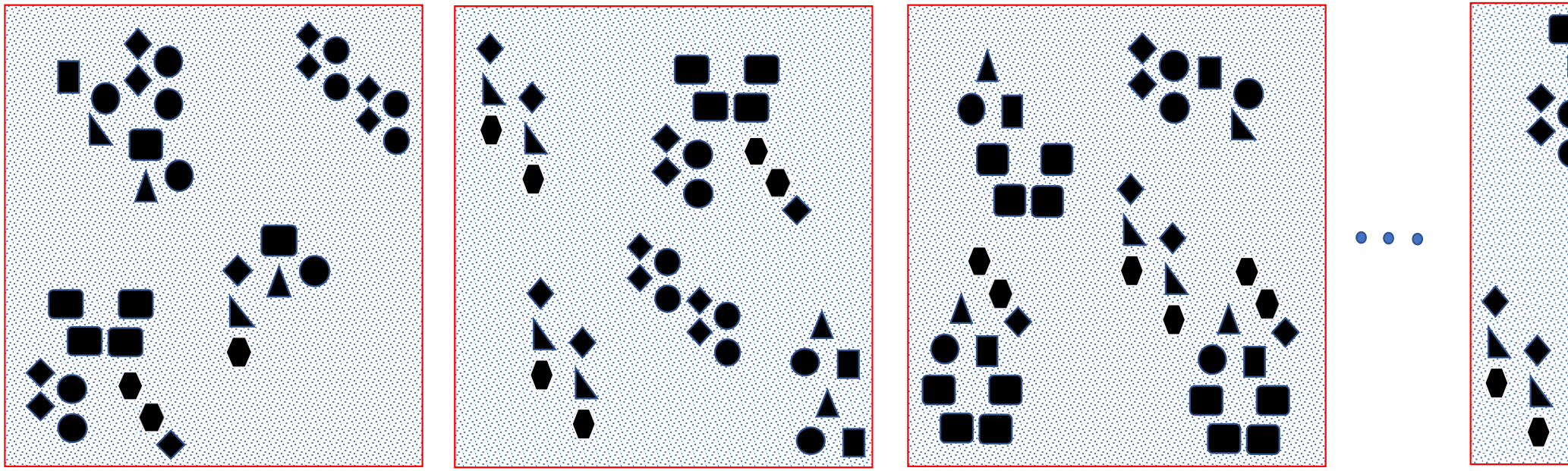
Shared Substructure Within Motifs



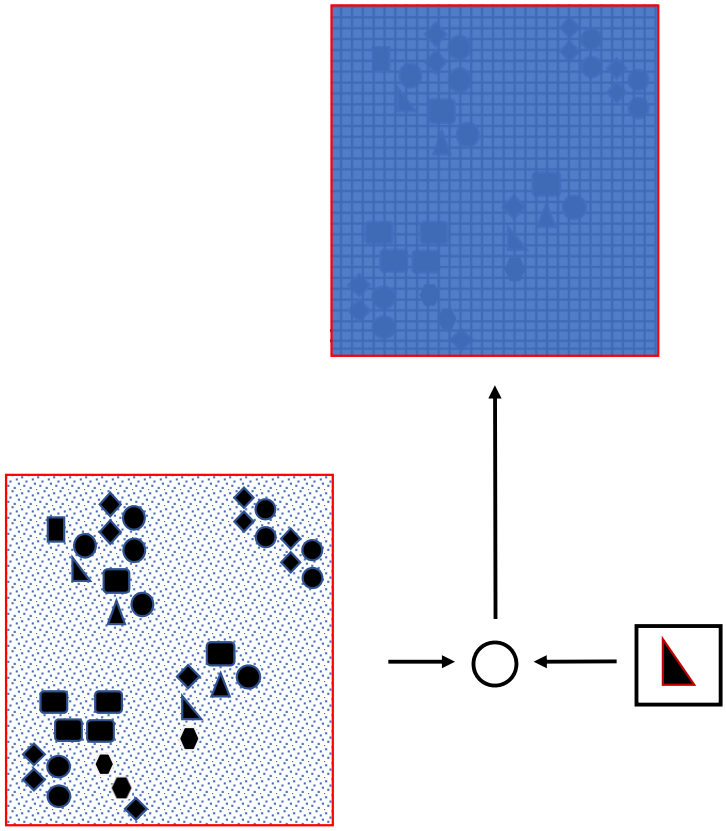
Hierarchical Representation of Images



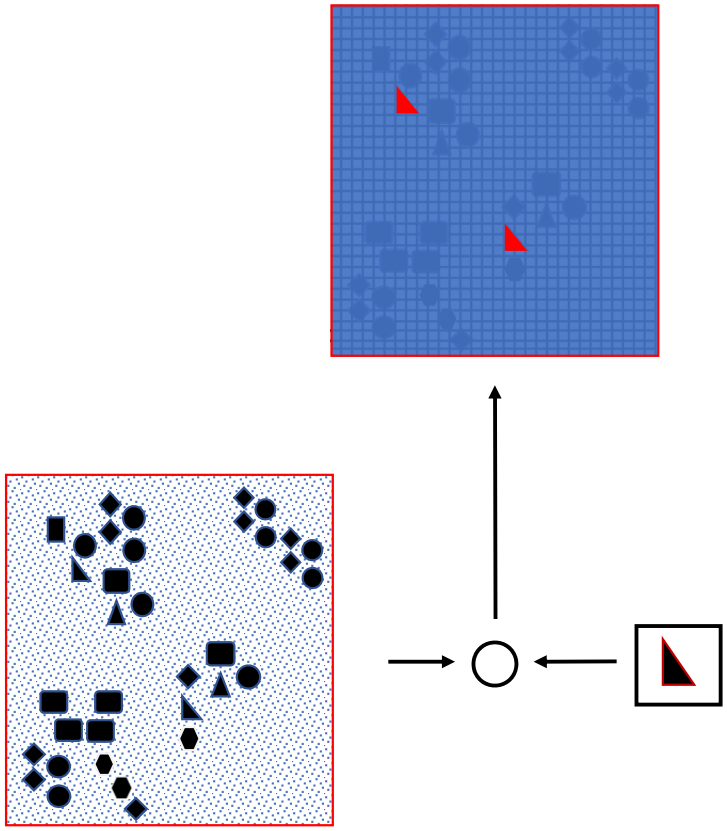
Recall the Data/Images



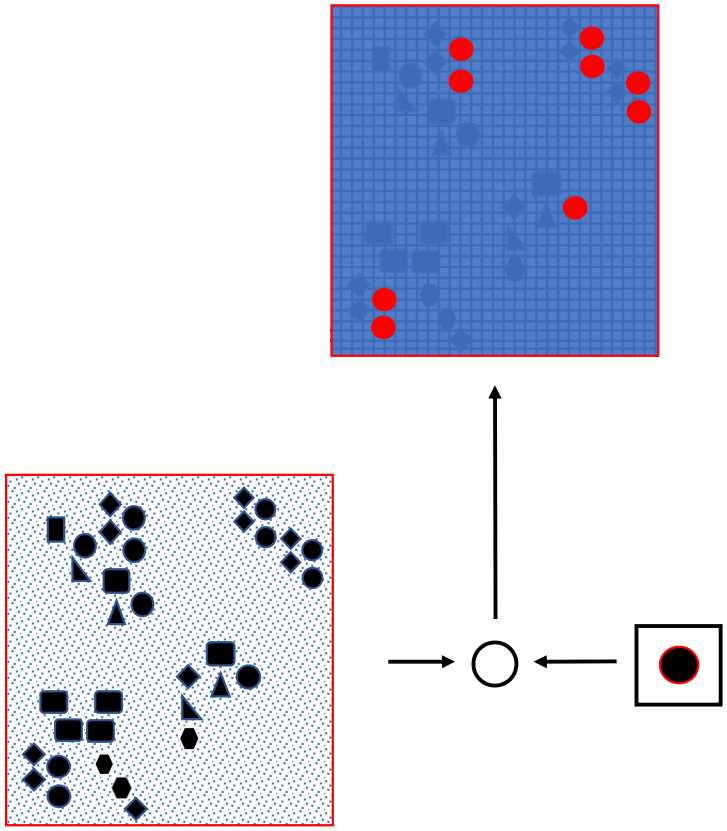
Convolutional Filter



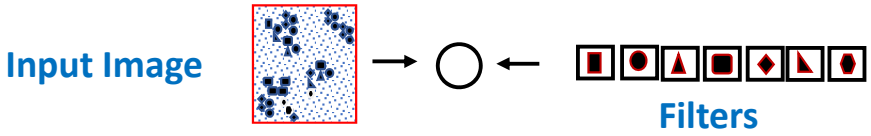
Convolutional Filter

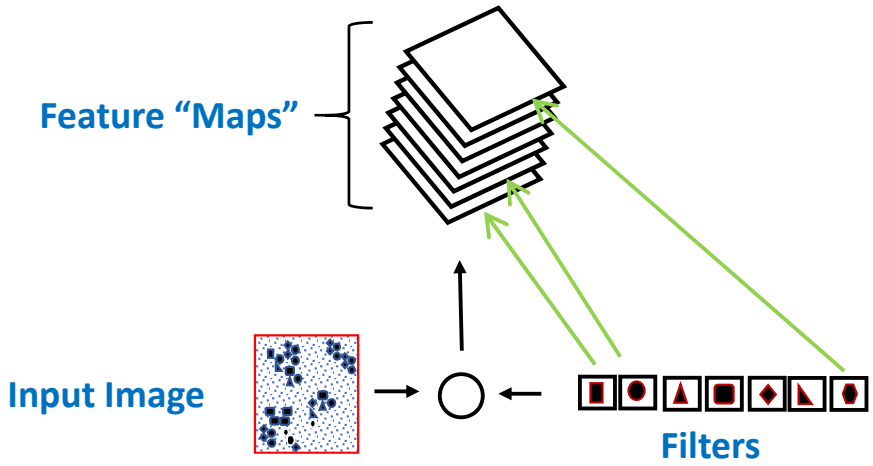


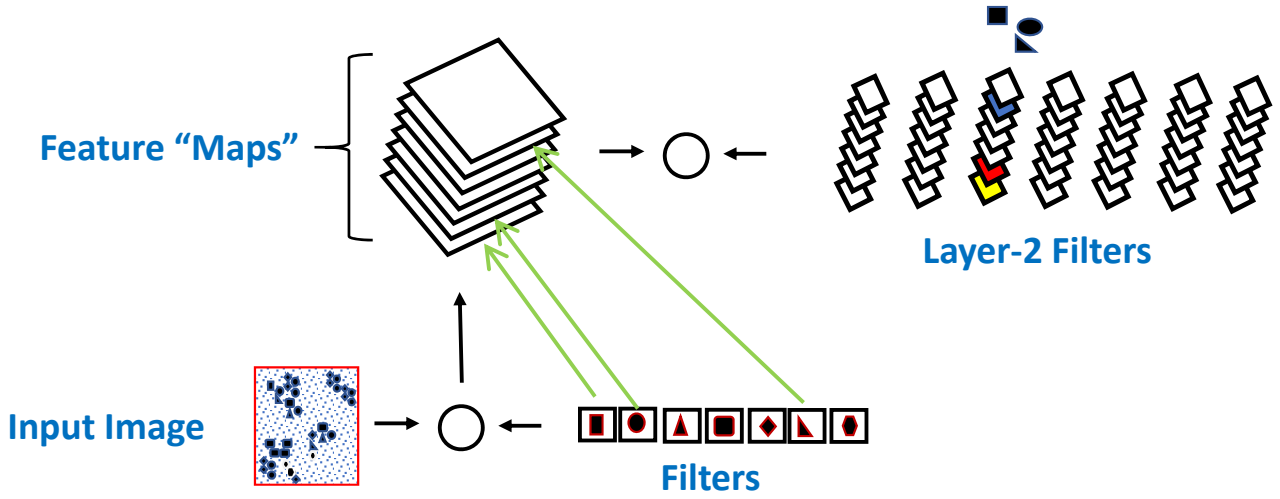
Convolutional Filter

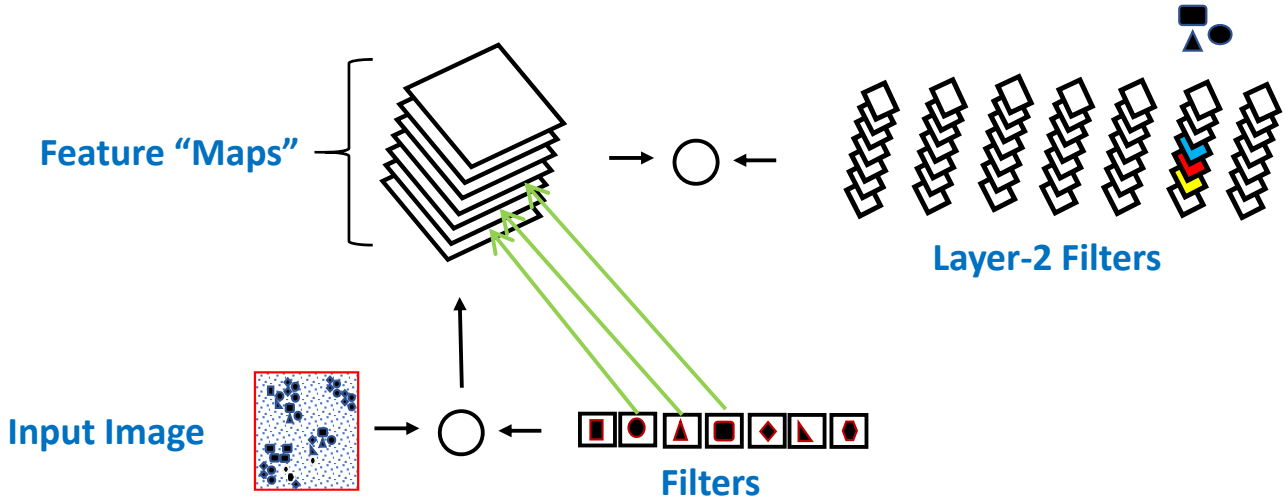


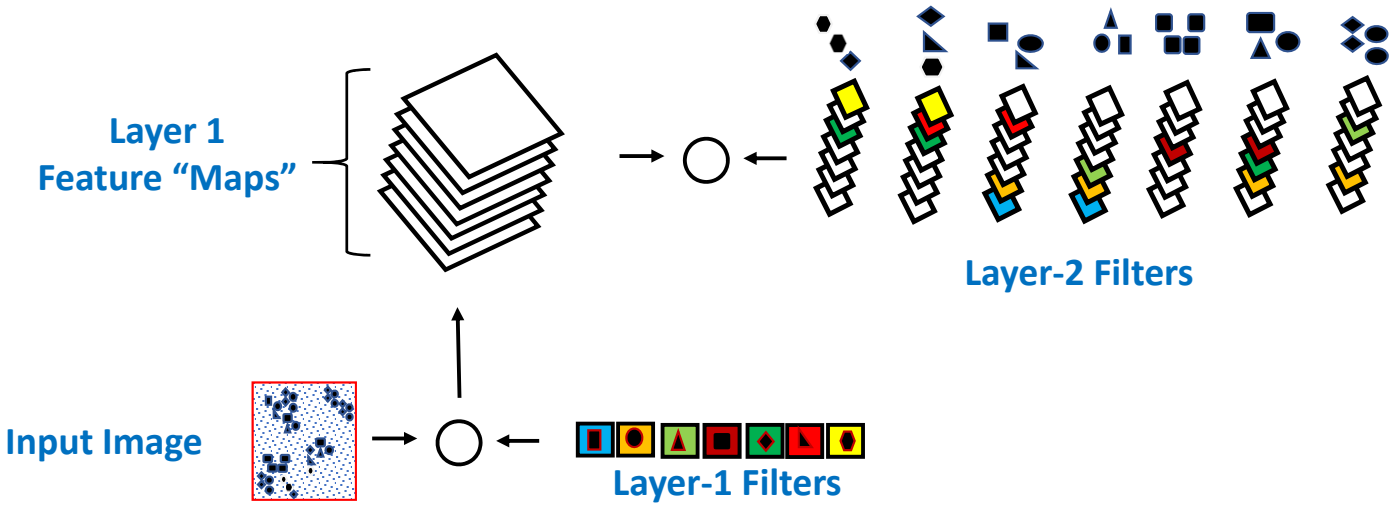
Multiple Filters, One for Each Building Block

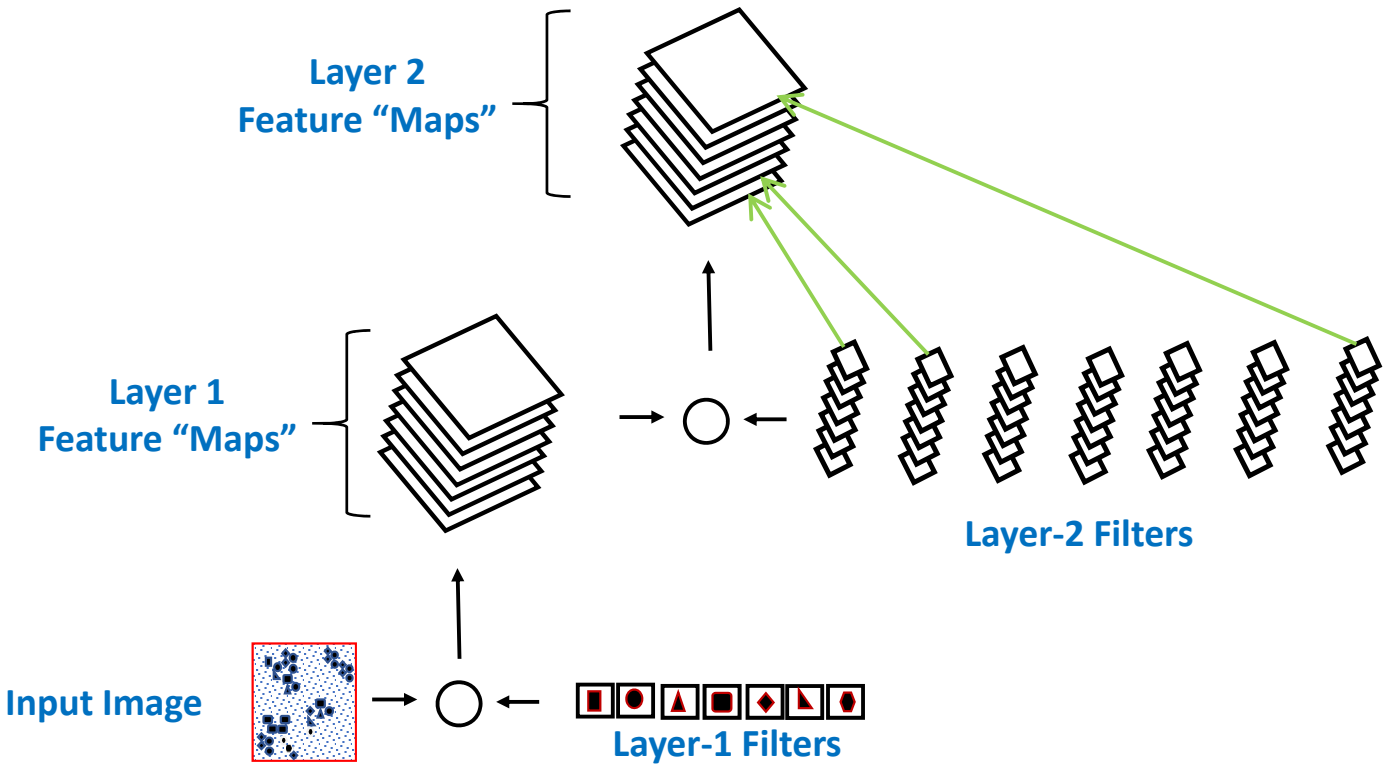


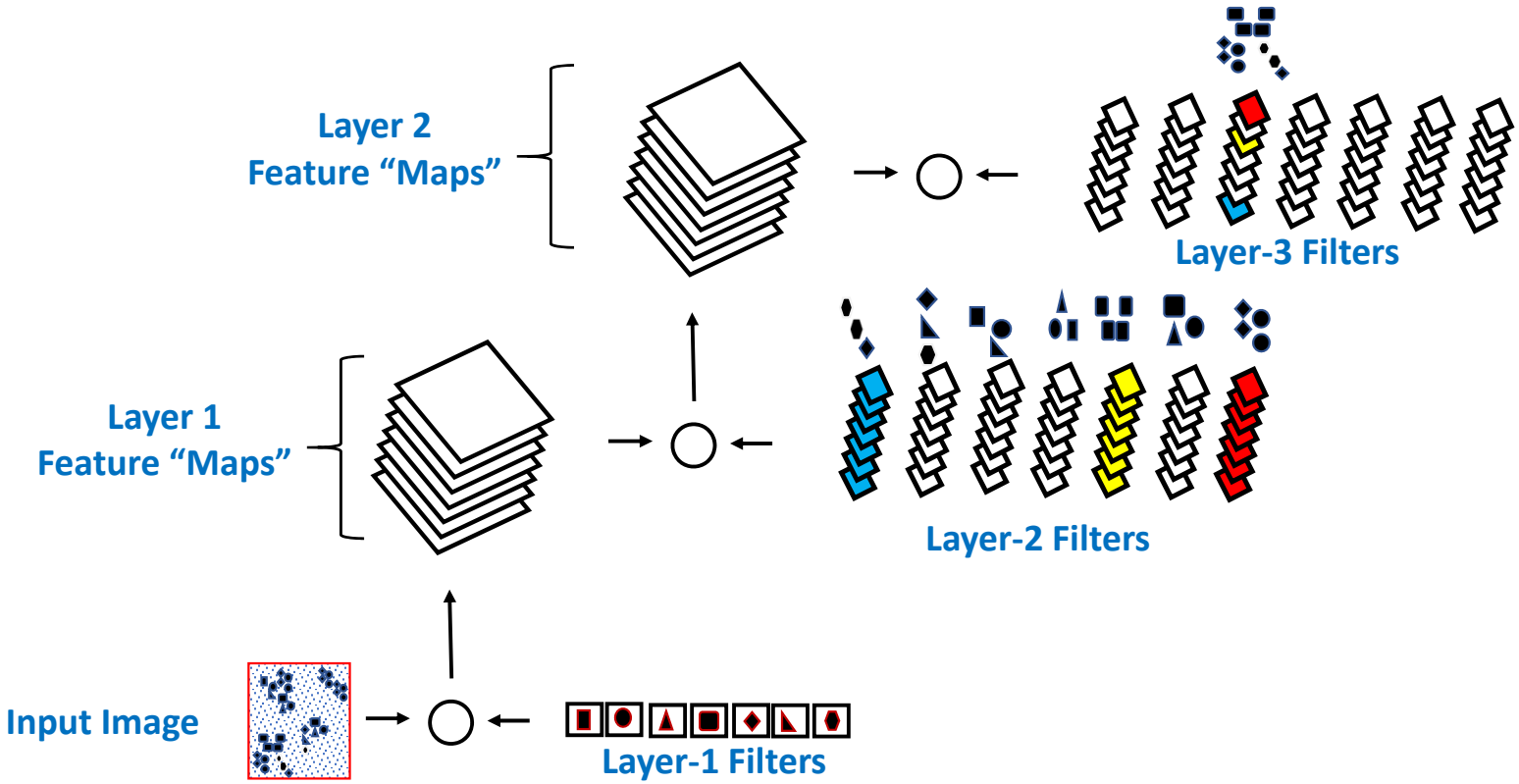

Feature “Maps”
Input Image
Filters

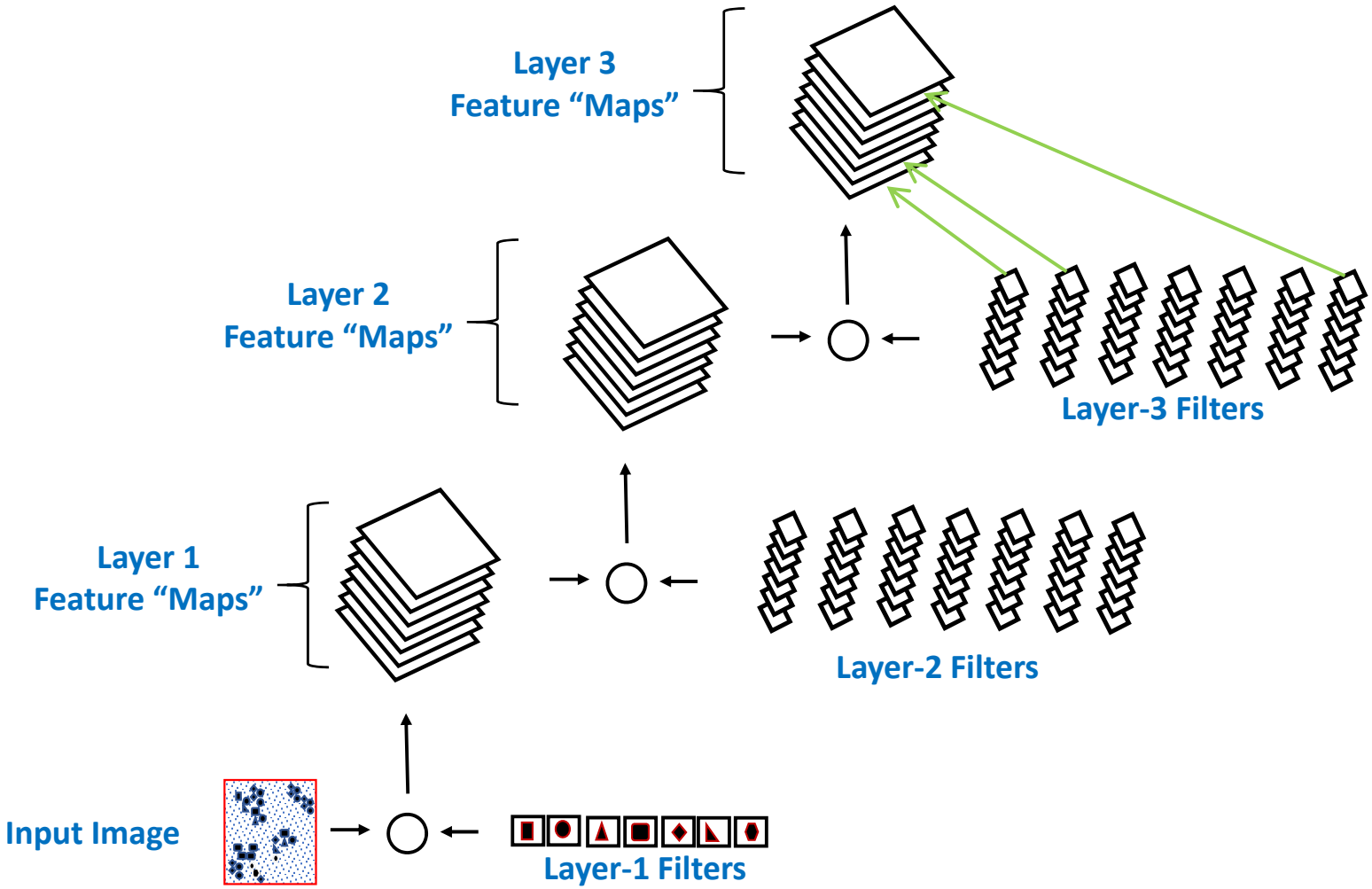

Feature “Maps”
Layer-2 Filters
Input Image
Filters


Feature “Maps”
Layer-2 Filters
Input Image
Filters

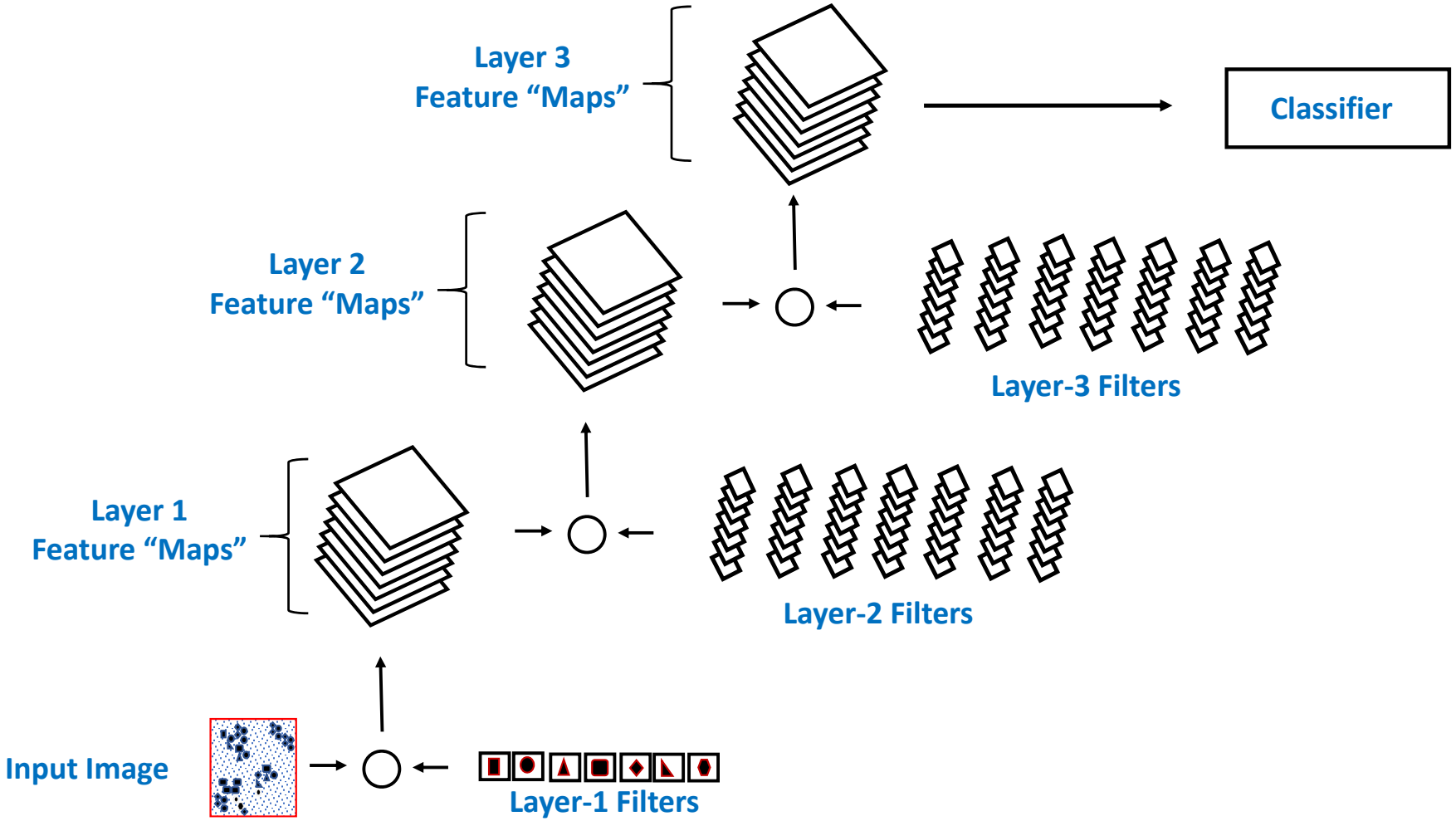

Layer 1
Feature "Maps"
Layer-2 Filters
Input Image
Layer-1 Filters


Layer 2
Feature "Maps"
Layer 1
Feature "Maps"
Layer-2 Filters
Input Image
Layer-1 Filters


Layer 2
Feature "Maps"
Layer-3 Filters
Layer 1
Feature "Maps"
Layer-2 Filters
Input Image
Layer-1 Filters

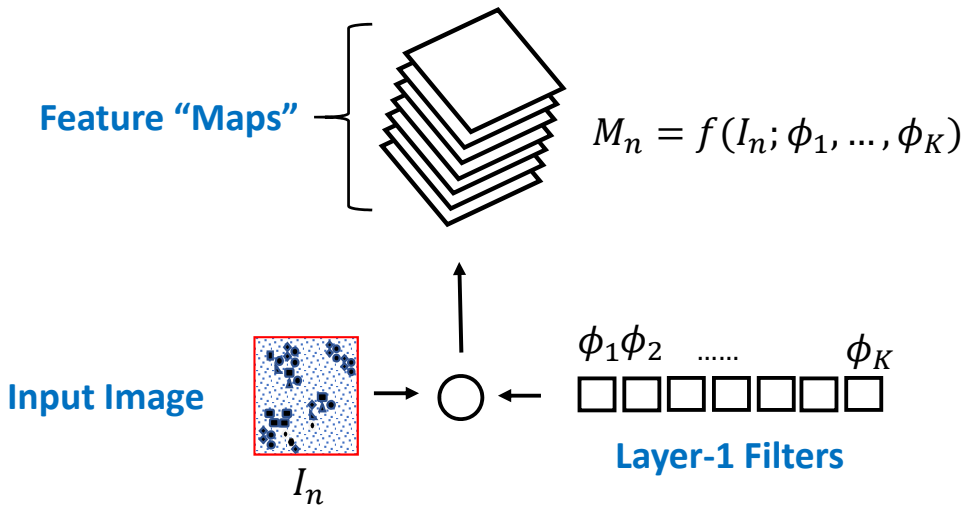

Layer 3
Feature "Maps"
Layer 2
Feature "Maps"
Layer-3 Filters
Layer 1
Feature "Maps"
Layer-2 Filters
Input Image
Layer-1 Filters

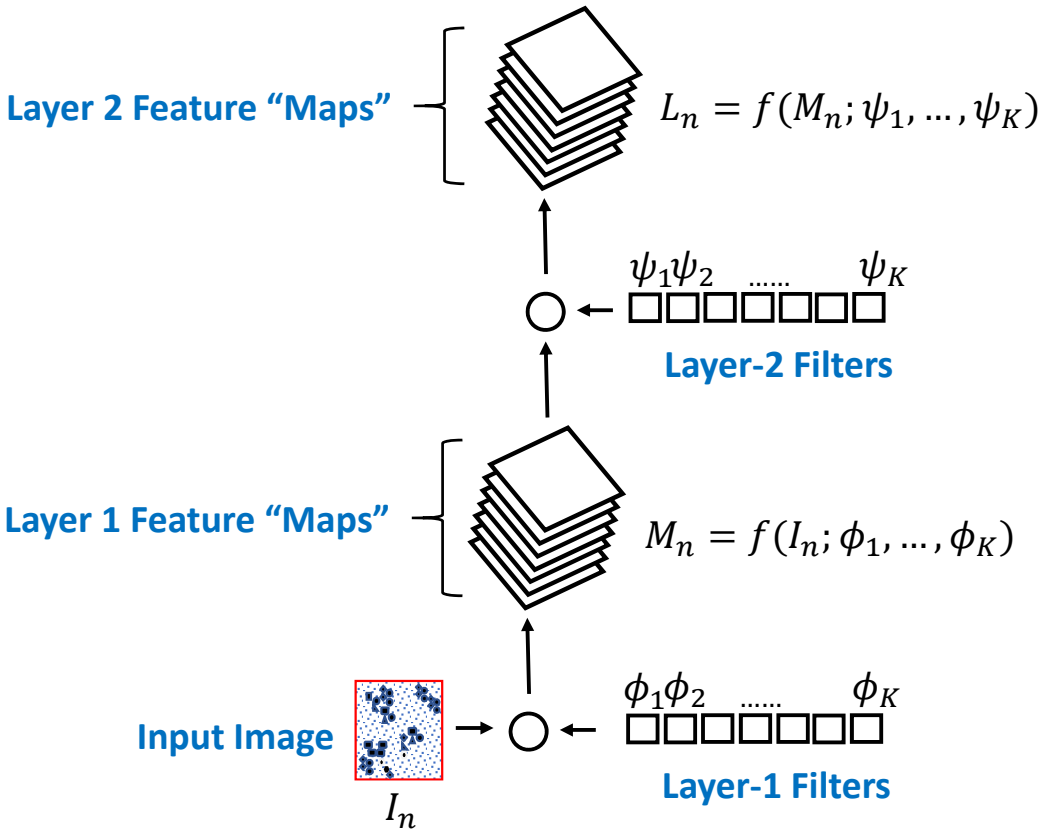
Deep Analysis Architecture

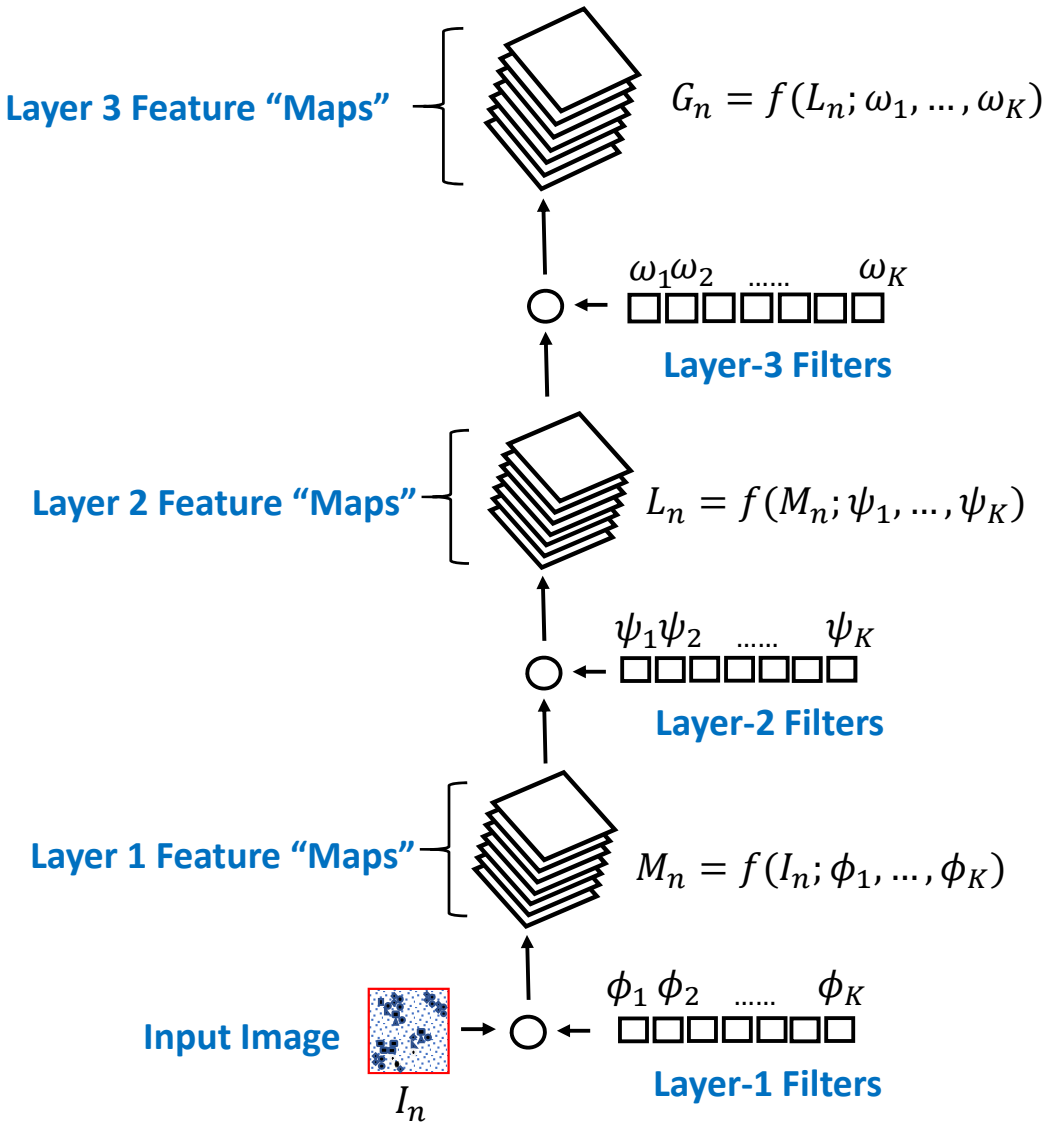


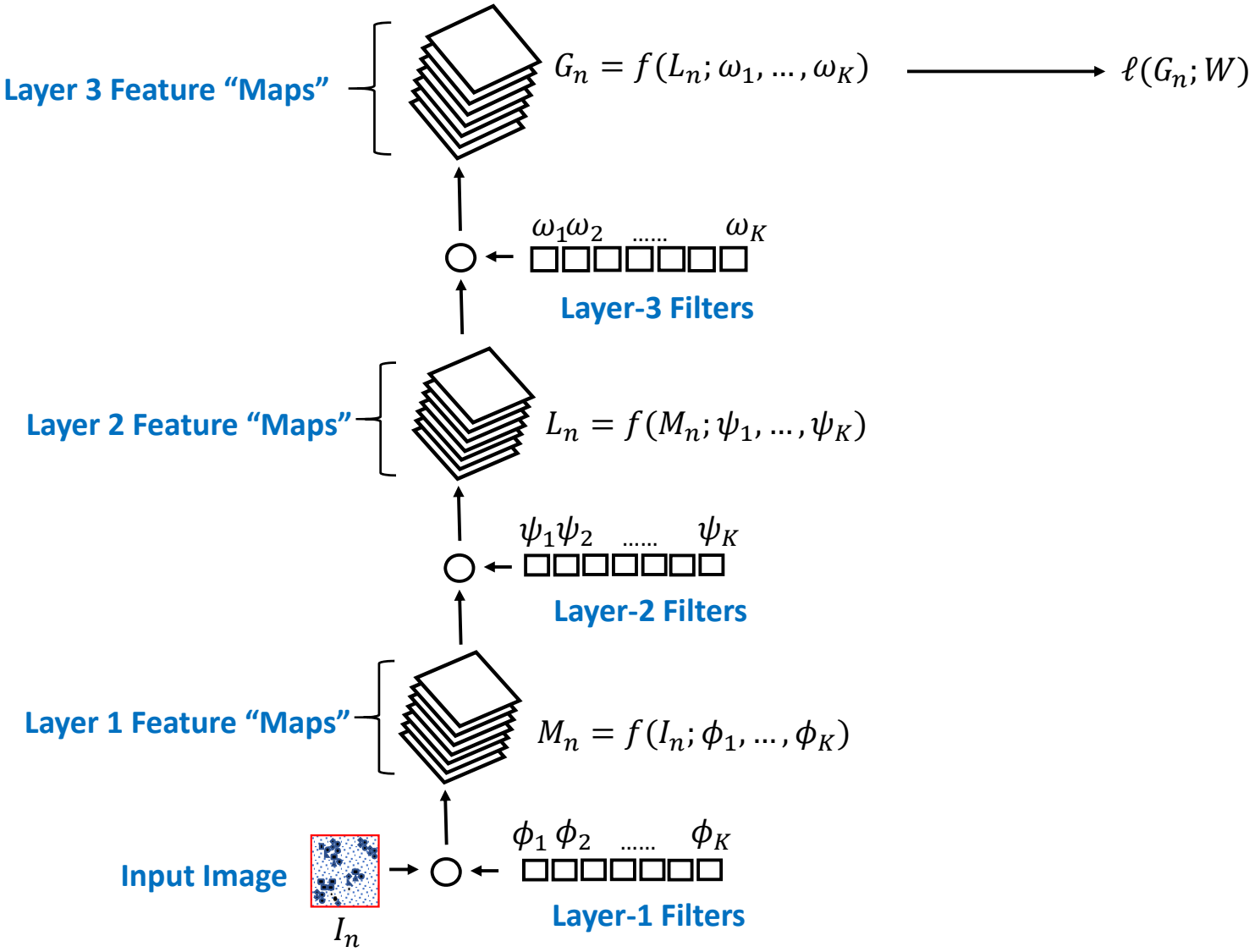
Given Images, How Do We Learn Model Parameters?

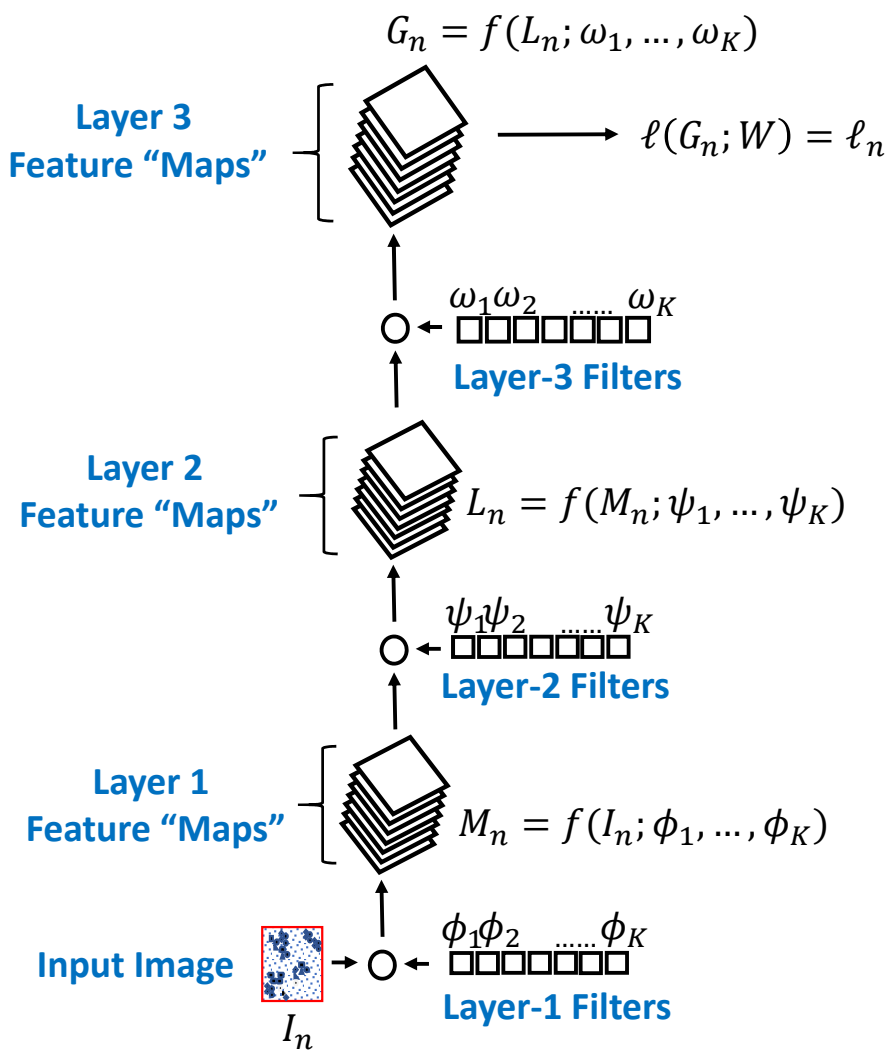
- The previous discussion was an illustration for motivating the “deep” algorithm concept
- Demonstrated using “toy” images
- How do we build such an algorithm in practice, given a large set of training images?


Feature "Maps"
$M_n = f(I_n; \phi_1, \ldots, \phi_K)$
Input Image
I_n
$\phi_1 \phi_2$ ..... ϕ_K
Layer-1 Filters


Layer 2 Feature "Maps"
$L_n = f(M_n; \psi_1, \ldots, \psi_K)$
$\psi_1 \psi_2$ ..... ψ_K
Layer-2 Filters
Layer 1 Feature "Maps"
$M_n = f(I_n; \phi_1, \ldots, \phi_K)$
Input Image
I_n
$\phi_1 \phi_2$ ..... ϕ_K
Layer-1 Filters


Layer 3 Feature "Maps"
$G_n = f(L_n; \omega_1, \ldots, \omega_K)$
$\omega_1 \omega_2$ ..... ω_K
Layer-3 Filters
Layer 2 Feature "Maps"
$L_n = f(M_n; \psi_1, \ldots, \psi_K)$
$\psi_1 \psi_2$ ..... ψ_K
Layer-2 Filters
Layer 1 Feature "Maps"
$M_n = f(I_n; \phi_1, \ldots, \phi_K)$
$\phi_1 \phi_2$ ..... ϕ_K
Input Image
I_n
Layer-1 Filters


Layer 3 Feature "Maps"
$G_n = f(L_n; \omega_1, \ldots, \omega_K)$
$\ell(G_n; W)$
$\omega_1 \omega_2$ ..... ω_K
Layer-3 Filters
Layer 2 Feature "Maps"
$L_n = f(M_n; \psi_1, \ldots, \psi_K)$
$\psi_1 \psi_2$ ..... ψ_K
Layer-2 Filters
Layer 1 Feature "Maps"
$M_n = f(I_n; \phi_1, \ldots, \phi_K)$
Input Image
I_n
ϕ_1 ϕ_2 ..... ϕ_K
Layer-1 Filters

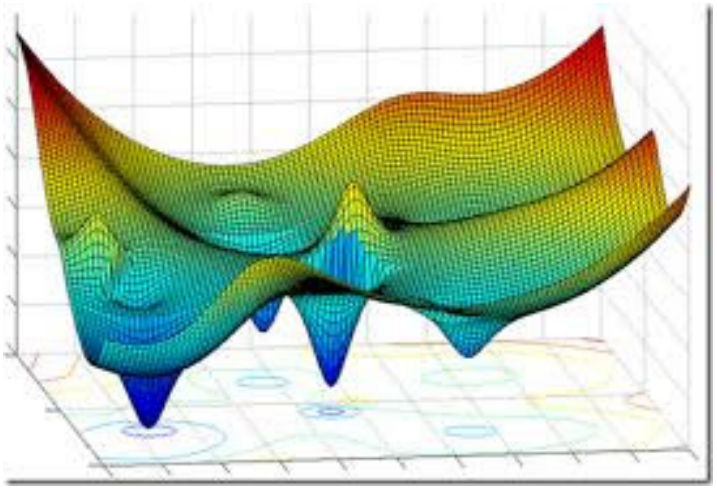


- Assume we have labeled images $\{I_n, y_n\}_{n=1,N}$
- I_n is image n, $y_n \in \{+1, -1\}$ is associated label
- Risk function of model parameters:

$$E(\Phi, \Psi, \Omega, W) = 1/N \sum_{n=1}^{N} loss(y_n, \ell_n)$$

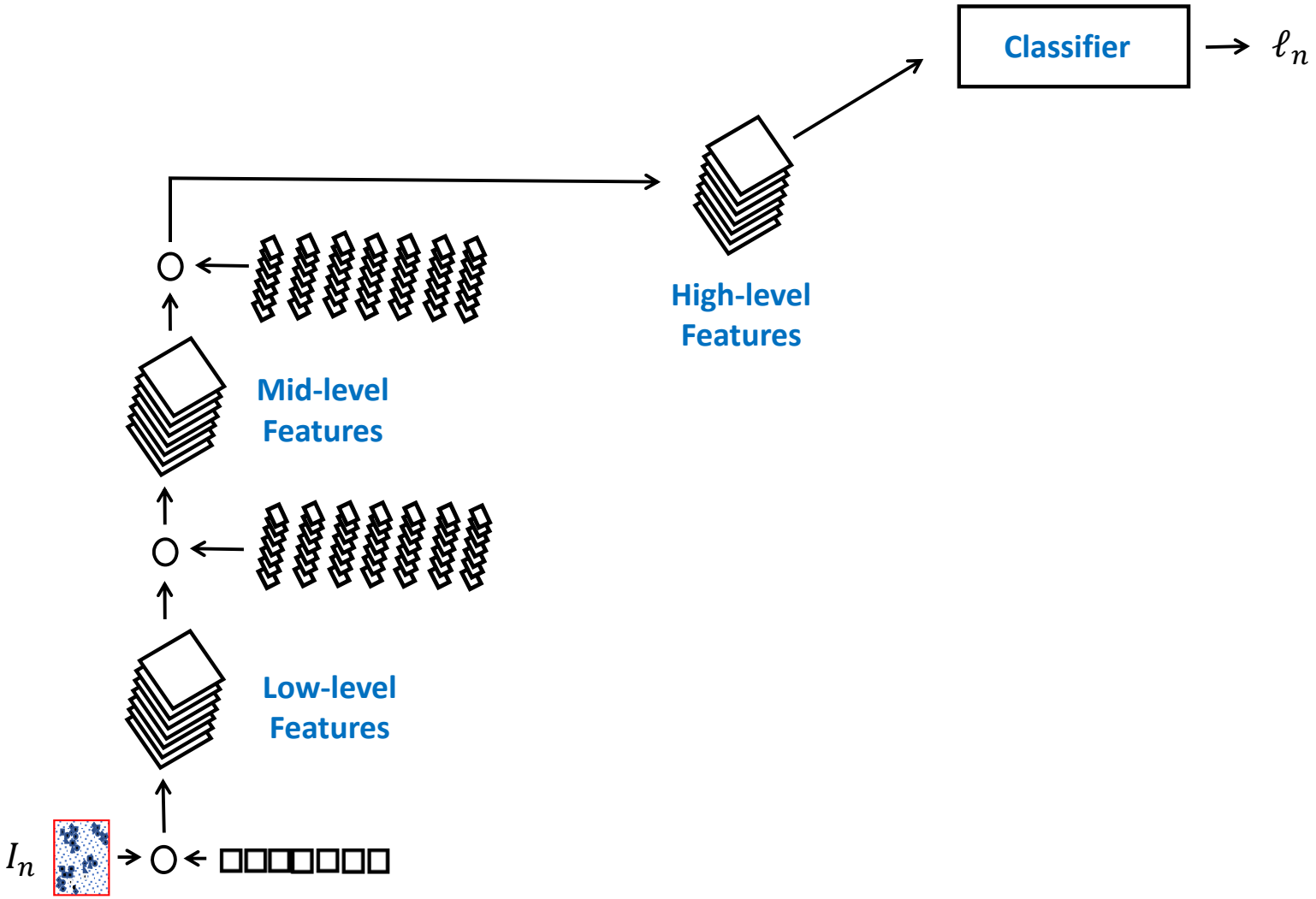
- Find model parameters $\hat{\Phi}, \hat{\Psi}, \hat{\Omega}, \hat{W}$ that minimize $E(\Phi, \Psi, \Omega, W)$

Cost Function vs. Model Parameters



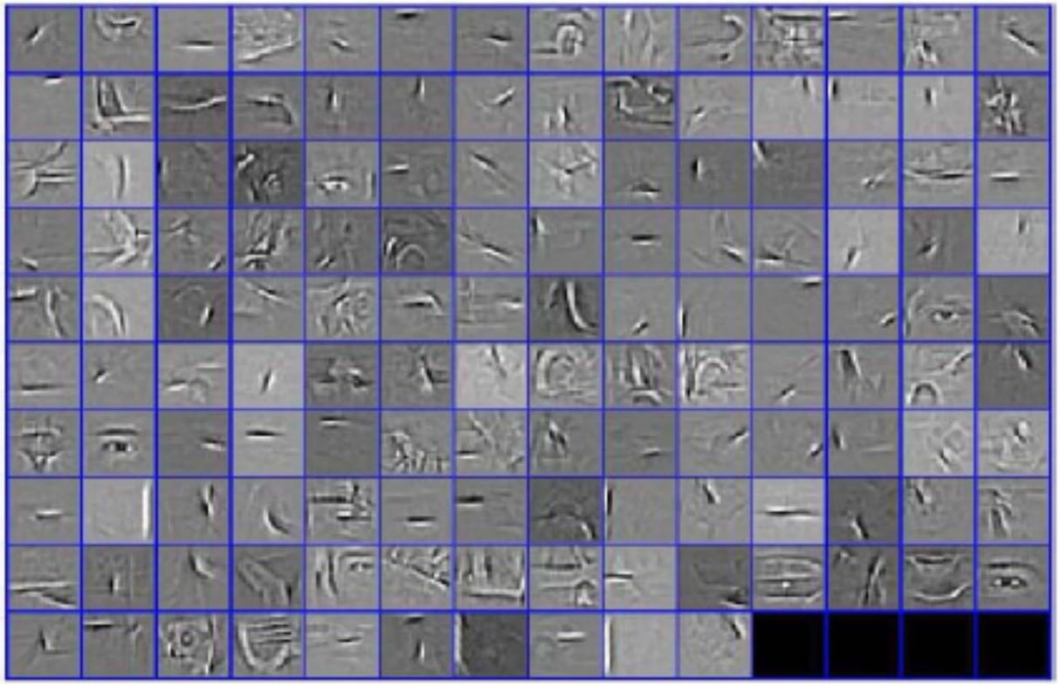
- High-dimensional function, as a consequence of a large number of model parameters
- Typically many local minima
- May be expensive to compute, for sophisticated models & large quantity of training images

Deep Architecture

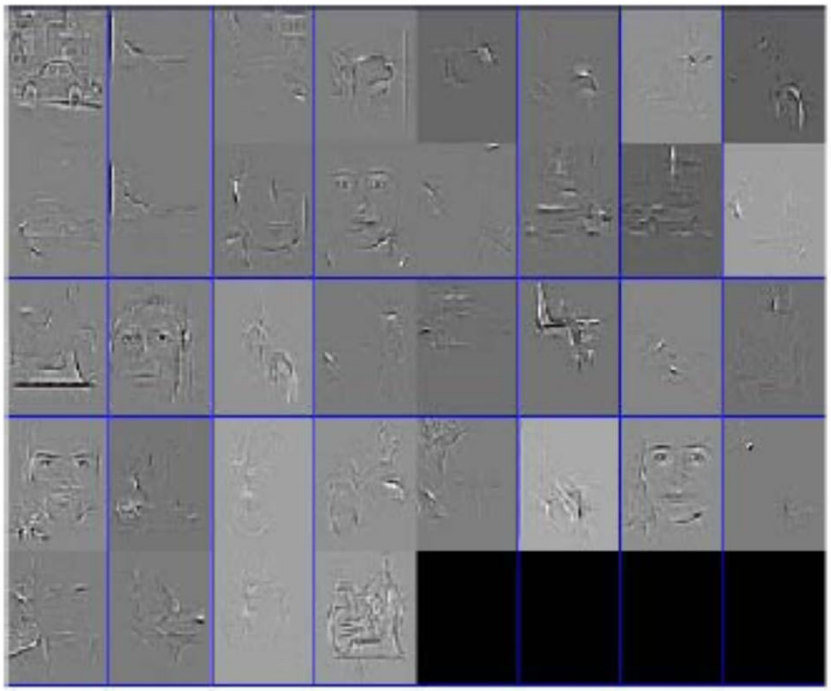


Layer 1

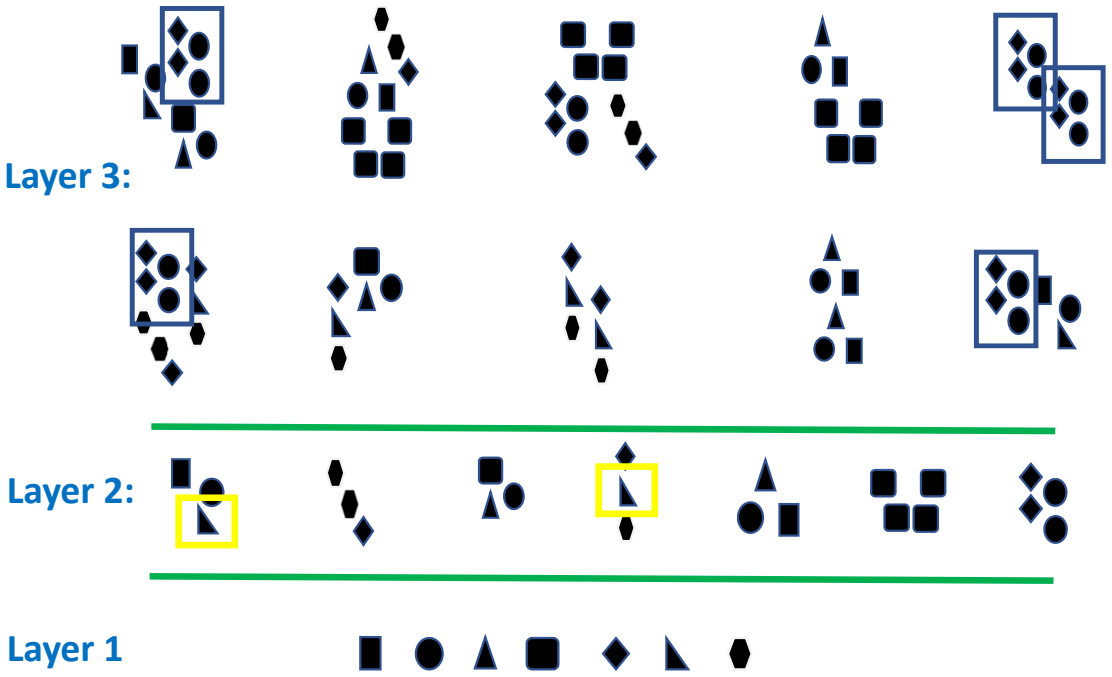
Layer 2



Layer 3

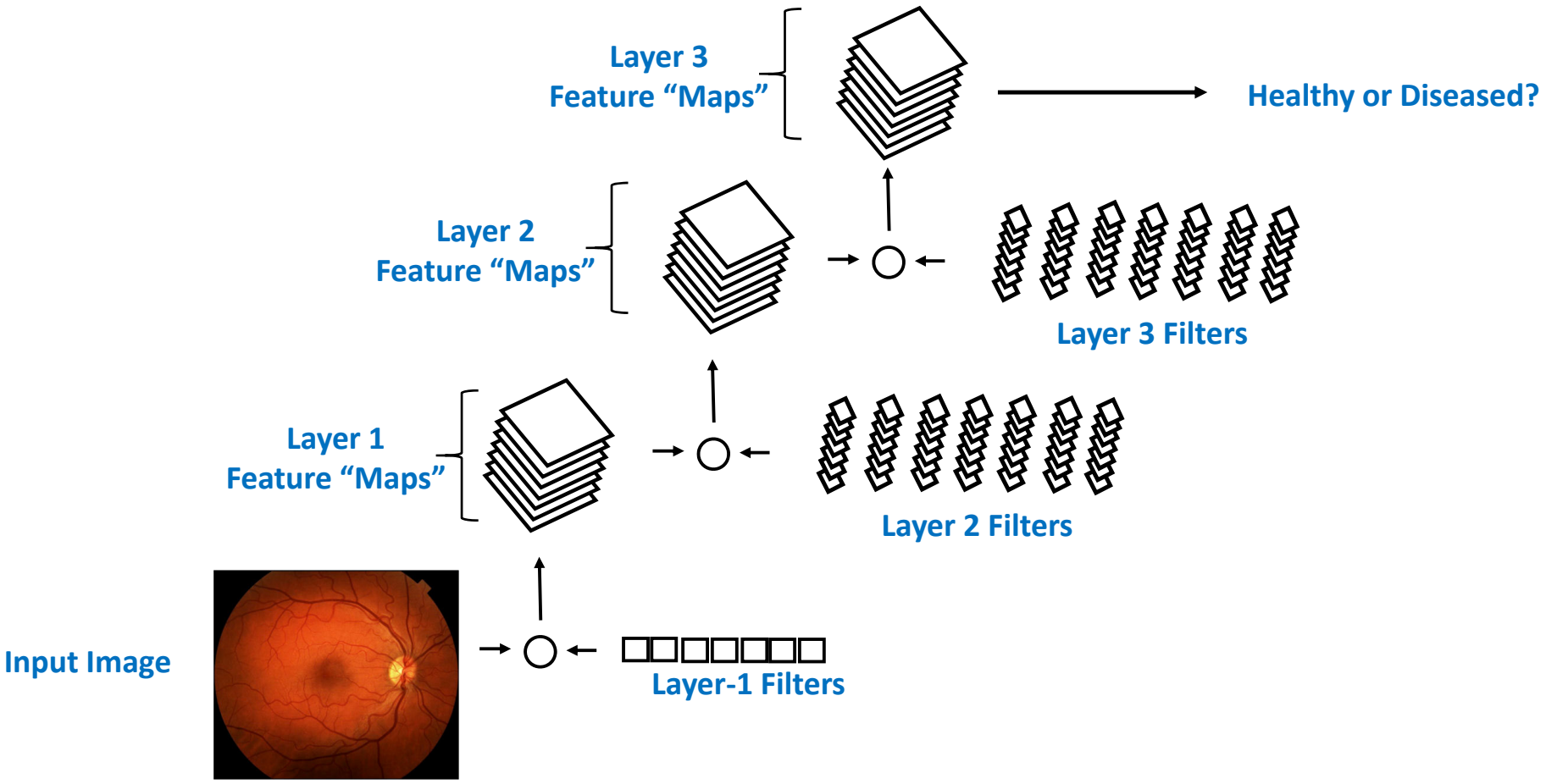


Advantage of Hierarchical Features?



- By learning and sharing statistical similarities within high-level motifs, we better leverage all training data
- If we do not use such a hierarchy, top-level motifs would be learned in isolation of each other

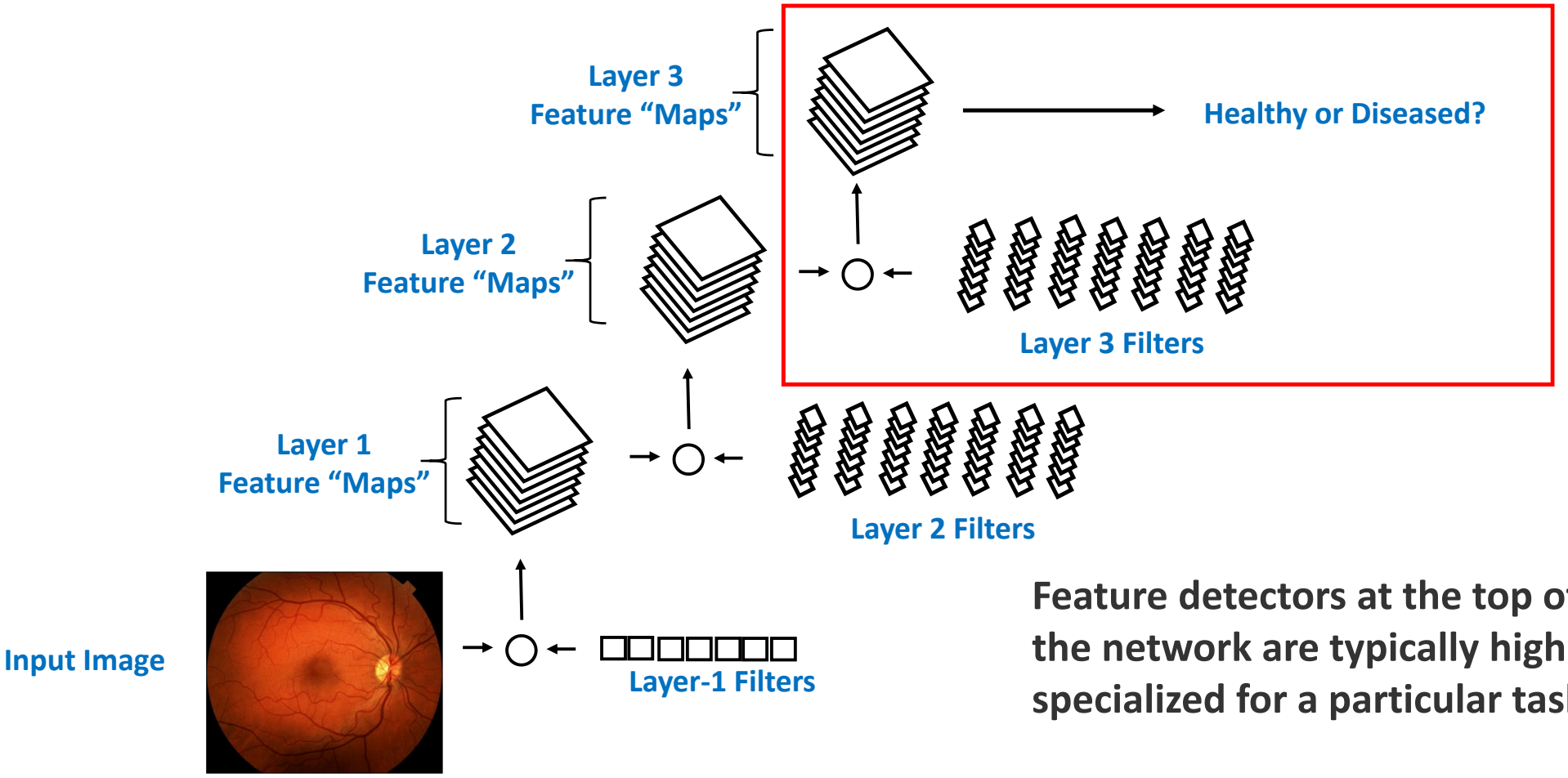
Transfer Learning



Transfer Learning

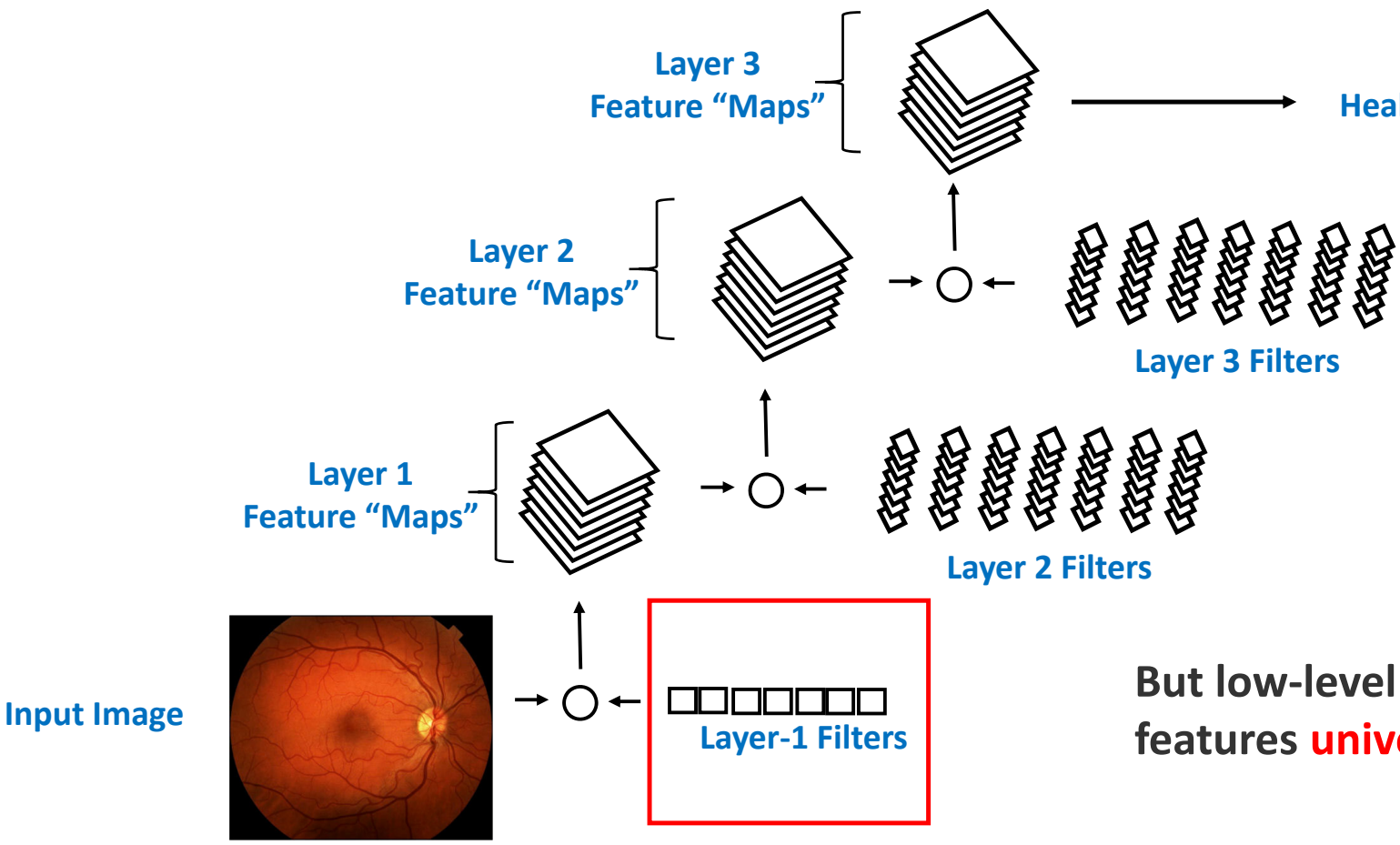
“To speed up the training, batch normalization as well as preinitialization using weights from the same network trained to classify objects in the ImageNet data set were used. Preinitialization also improved performance.”

Transfer Learning



Feature detectors at the top of the network are typically highly specialized for a particular task

Transfer Learning

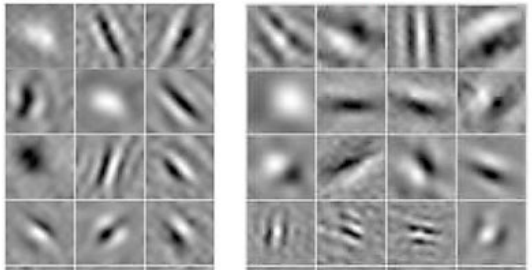


But low-level detectors extract features universal to all images

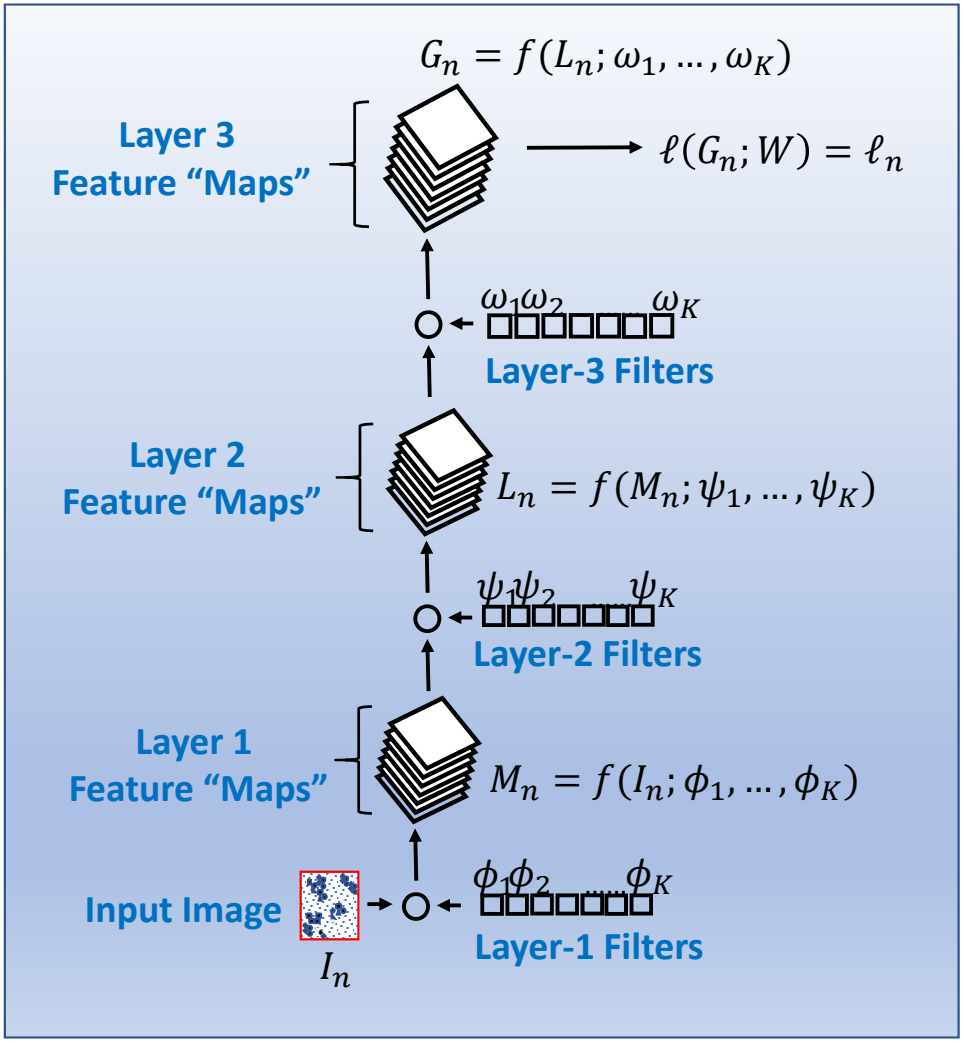
Transfer Learning

Layer 1 Filters,
Convolutional Neural Network

Neuron Receptive Fields,
Macaque Visual Cortex



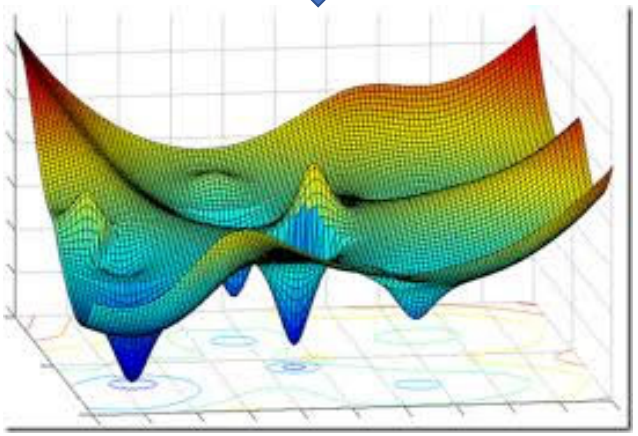
Big Picture



- Assume we have labeled images $\{I_n, y_n\}_{n=1,N}$
- I_n is image n, $y_n \in \{+1, -1\}$ is associated label
- Risk function of model parameters:

$$E(\Phi, \Psi, \Omega, W) = 1/N \sum_{n=1}^{N} loss(y_n, \ell_n)$$

- Find model parameters $\hat{\Phi}, \hat{\Psi}, \hat{\Omega}, \hat{W}$ that minimize $E(\Phi, \Psi, \Omega, W)$

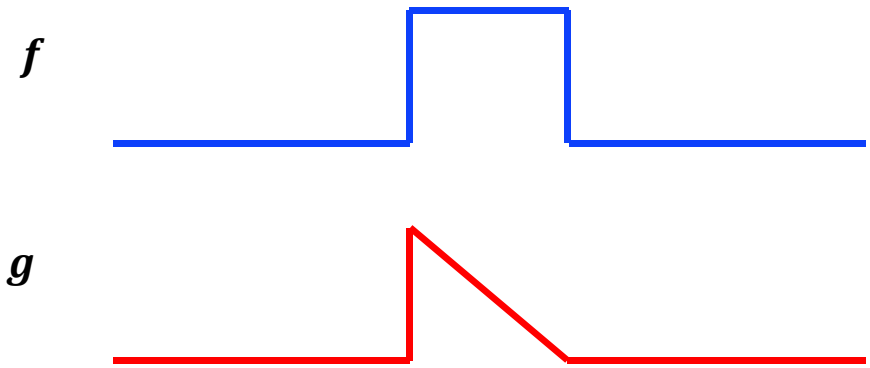


Deep Convolutional Neural Nets
Part II

Tim Dunn

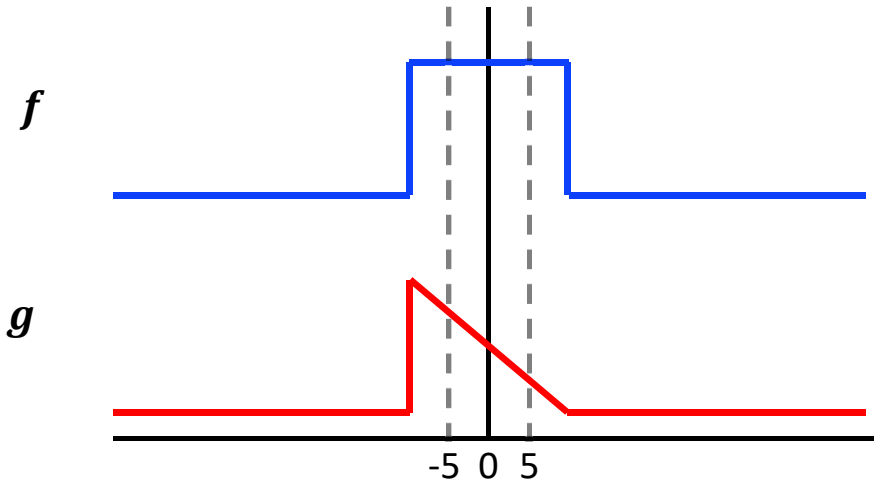
Duke MLWS 2019

Intro to the Convolution (1D)



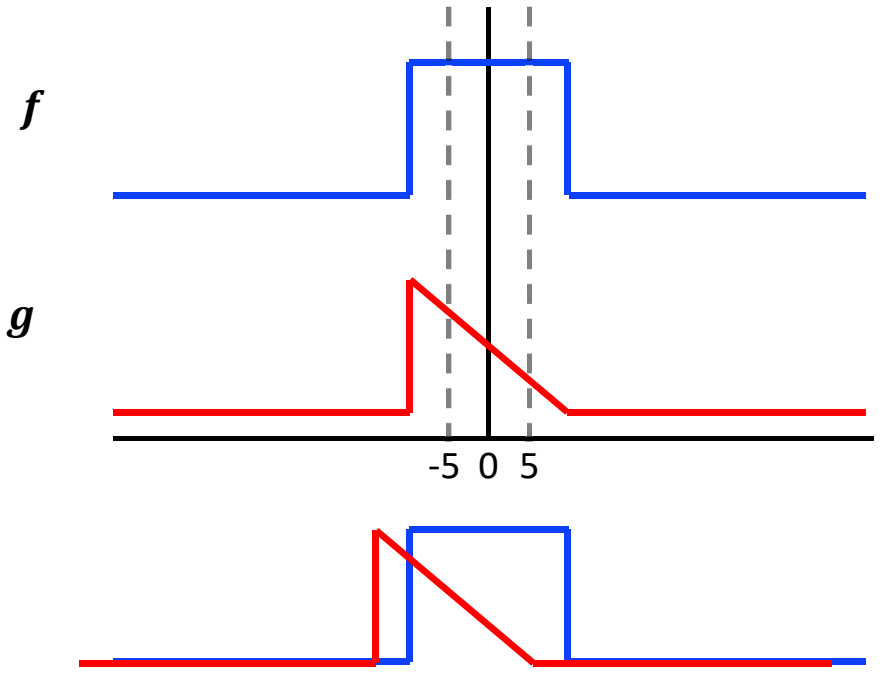
$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

Intro to the Convolution (1D)



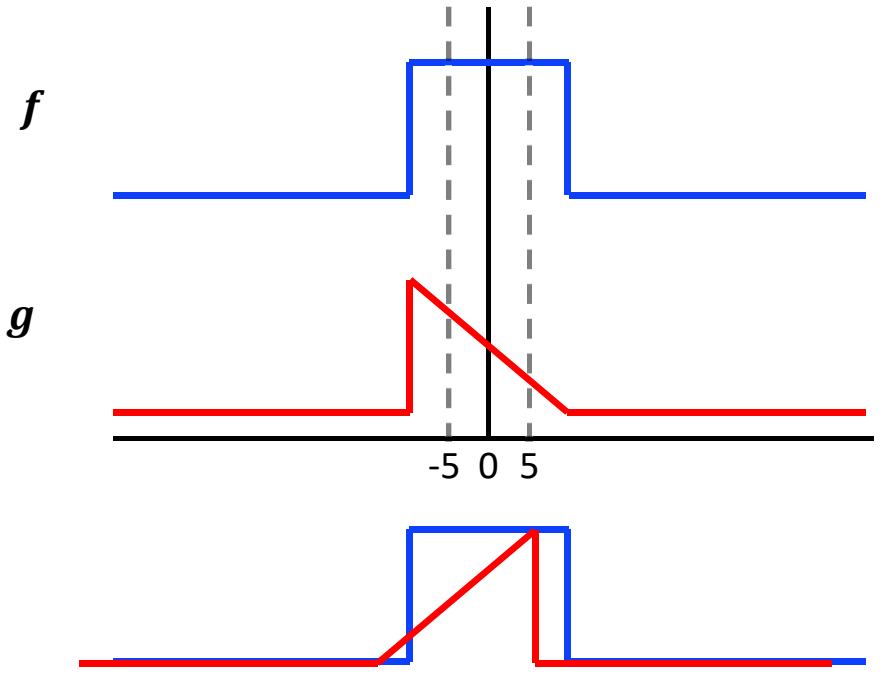
$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

Intro to the Convolution (1D)



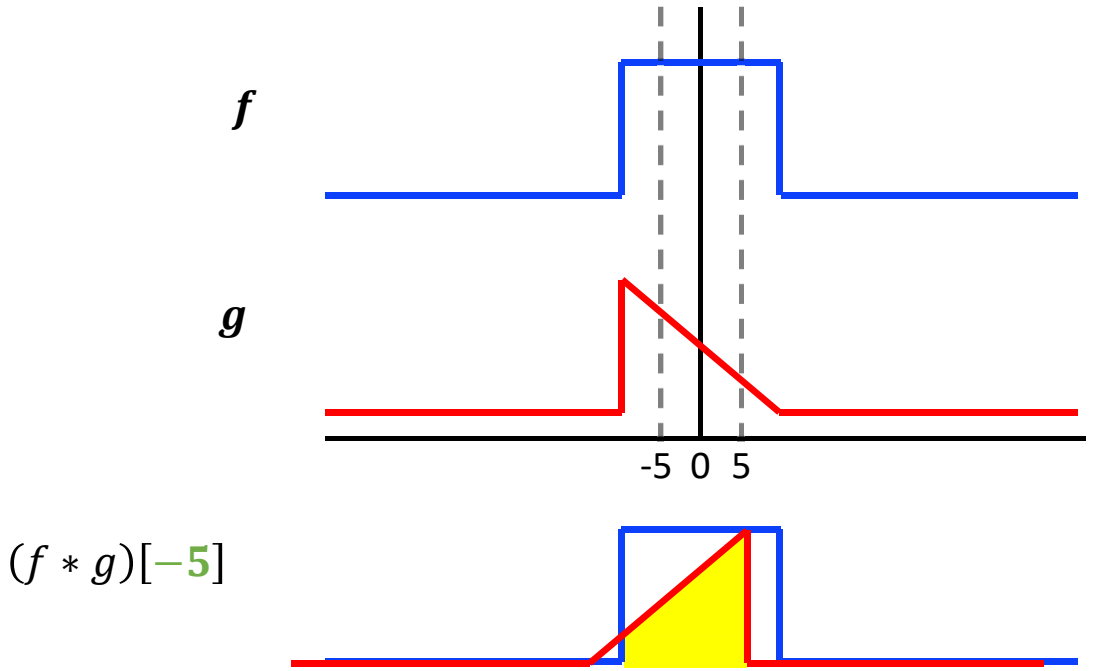
$$(f * g)[-5] = \sum_{m=-\infty}^{\infty} f[m]g[-(5+m)]$$

Intro to the Convolution (1D)



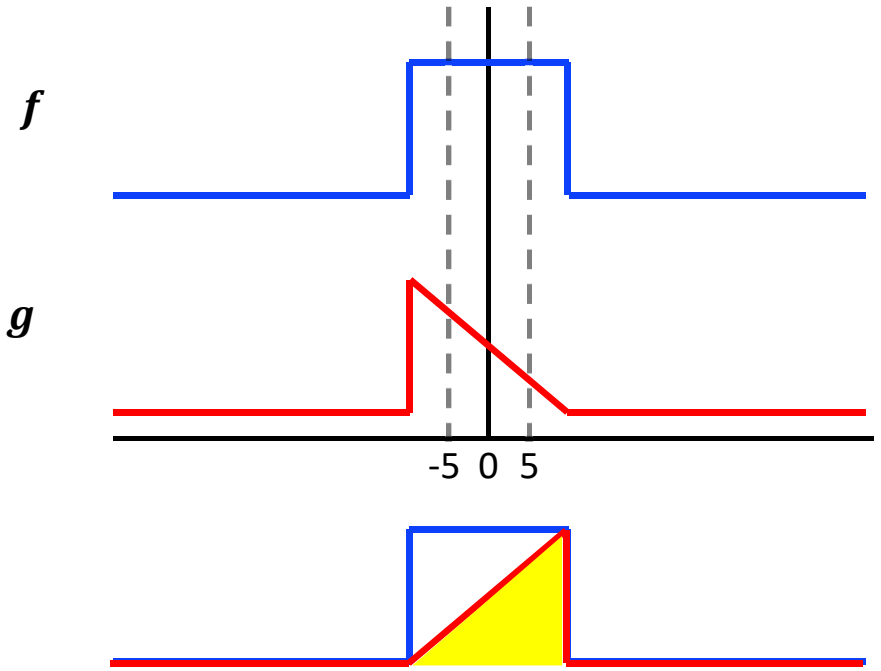
$$(f * g)[-5] = \sum_{m=-\infty}^{\infty} f[m]g[-(5+m)]$$

Intro to the Convolution (1D)



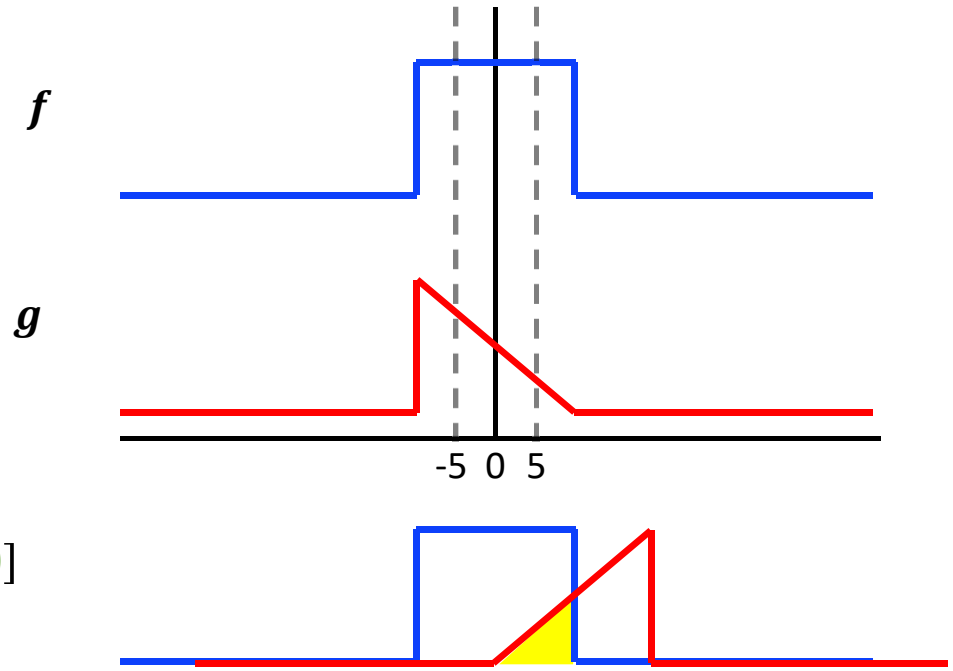
$$(f * g)[-5] = \sum_{m=-\infty}^{\infty} f[m]g[-(5+m)] = 15$$

Intro to the Convolution (1D)



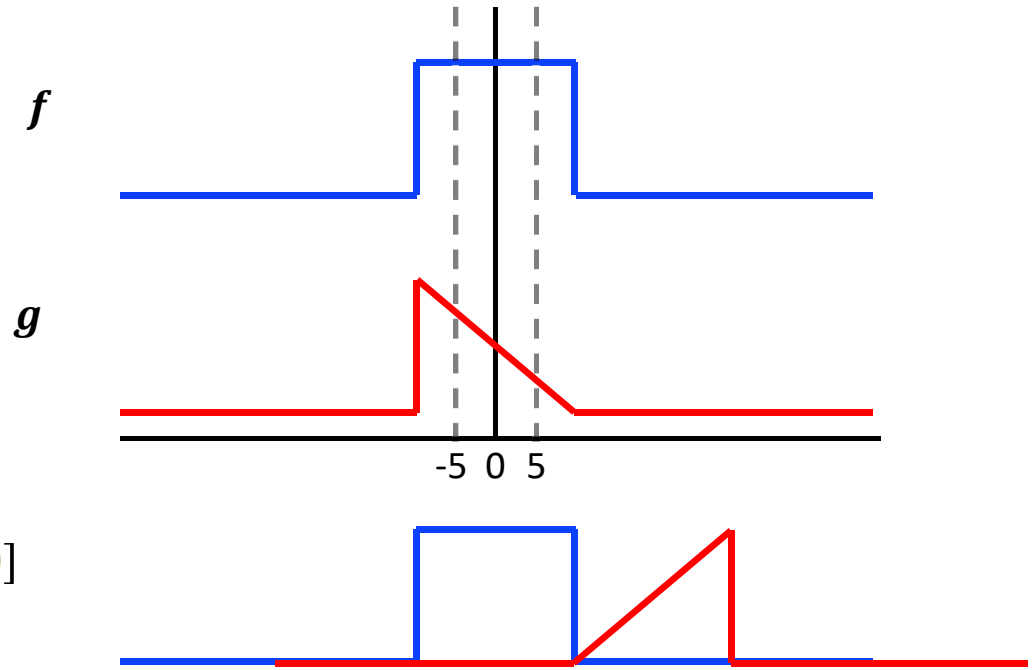
$$(f * g)[\mathbf{0}] = \sum_{m=-\infty}^{\infty} f[m]g[-(m)] = 20$$

Intro to the Convolution (1D)



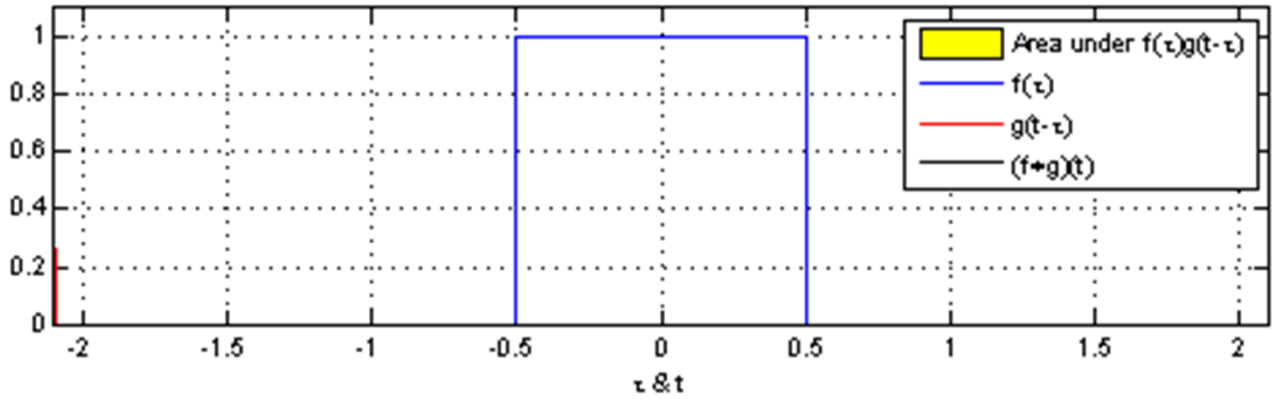
$$(f * g)[\mathbf{10}] = \sum_{m=-\infty}^{\infty} f[m]g[-(\mathbf{-10} + m)] = 10$$

Intro to the Convolution (1D)



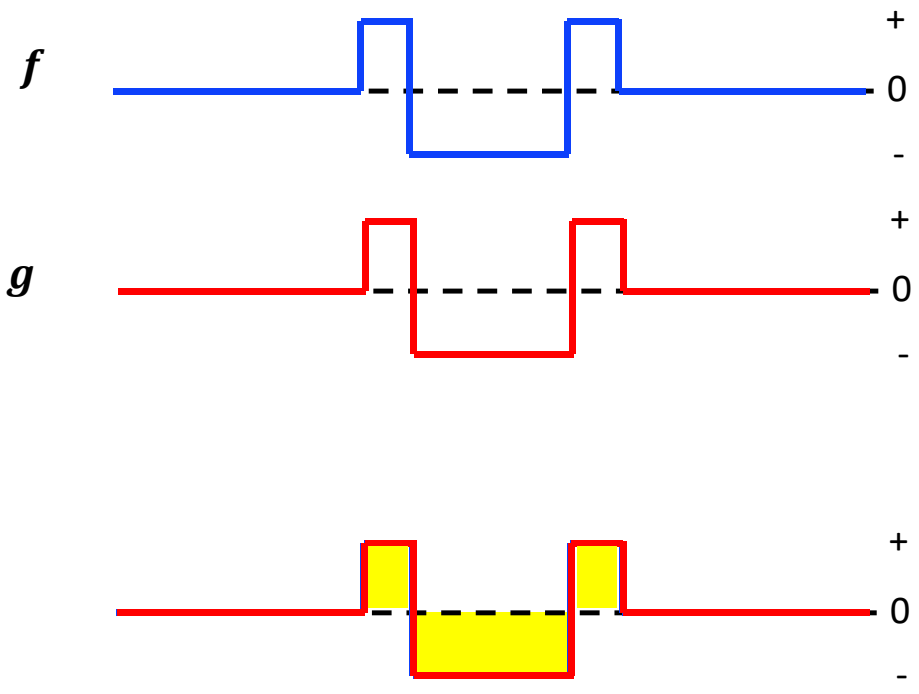
$$(f * g)[\mathbf{20}] = \sum_{m=-\infty}^{\infty} f[m]g[-(-\mathbf{20} + m)] = 0$$

Intro to the Convolution (1D)

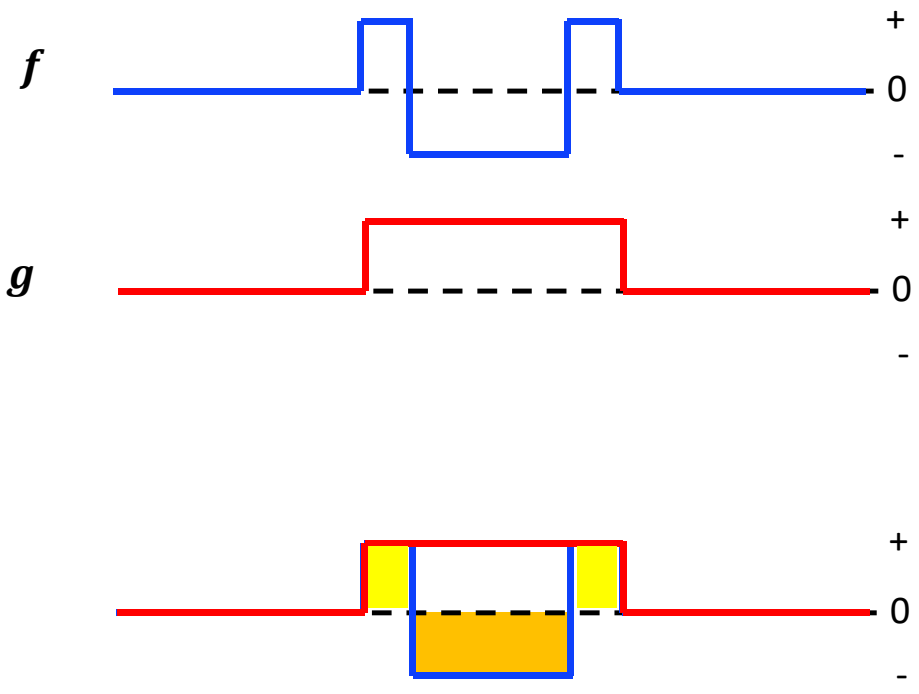


Wikipedia

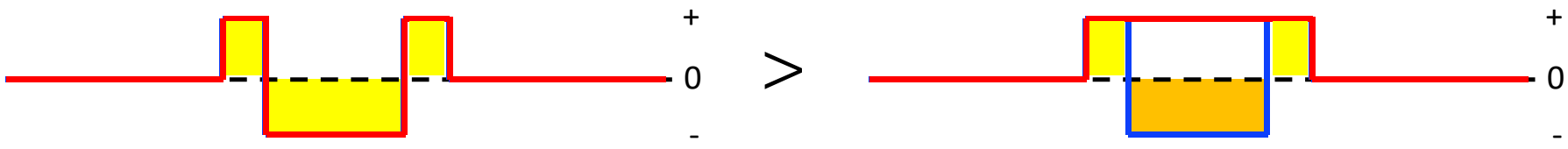
Convolutional Filters Are Feature Detectors



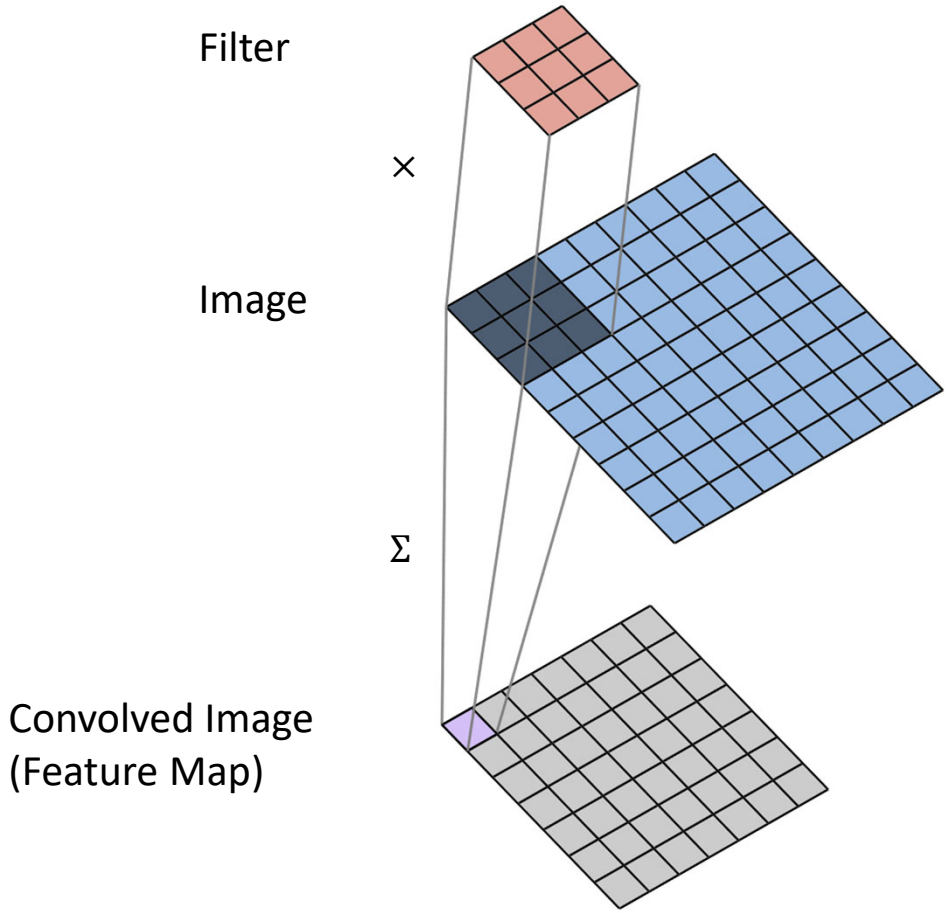
Convolutional Filters Are Feature Detectors



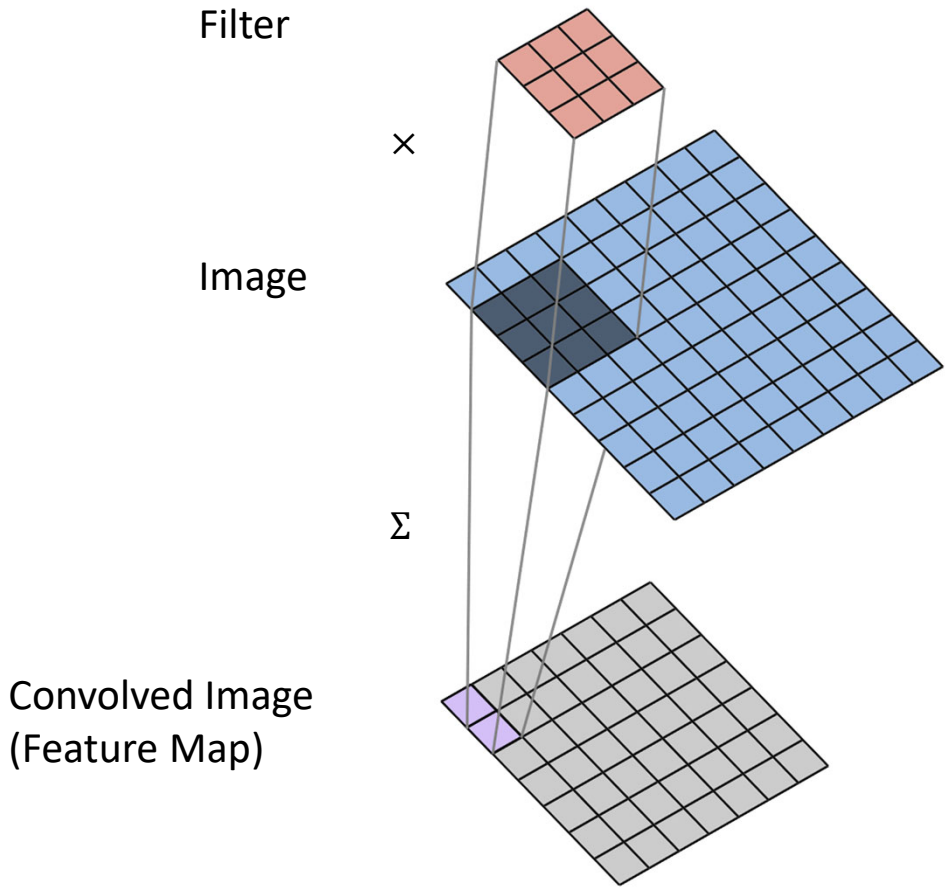
Convolutional Filters Are Feature Detectors



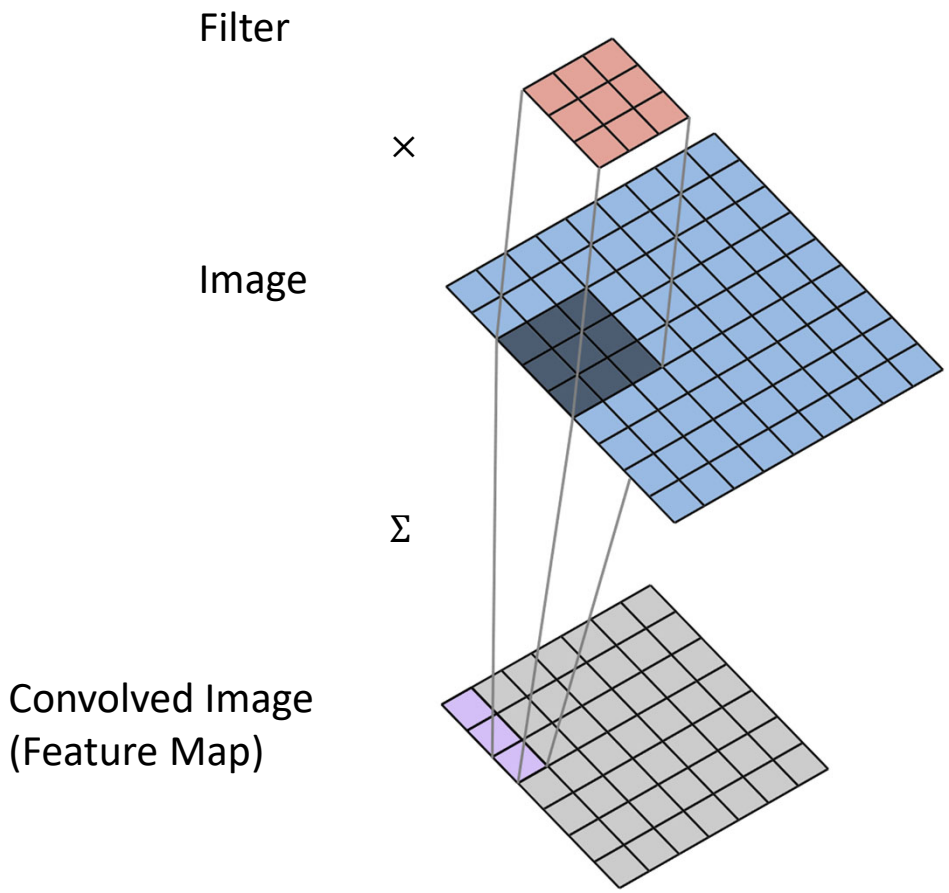
2D Spatial Convolution



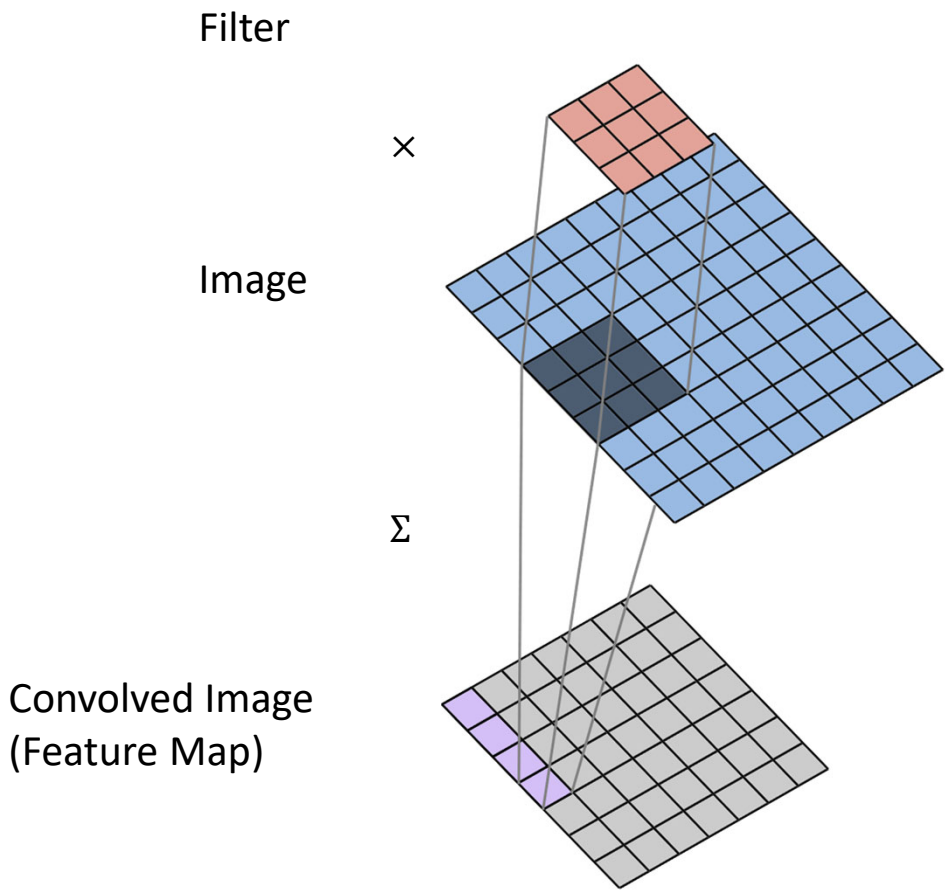
2D Spatial Convolution



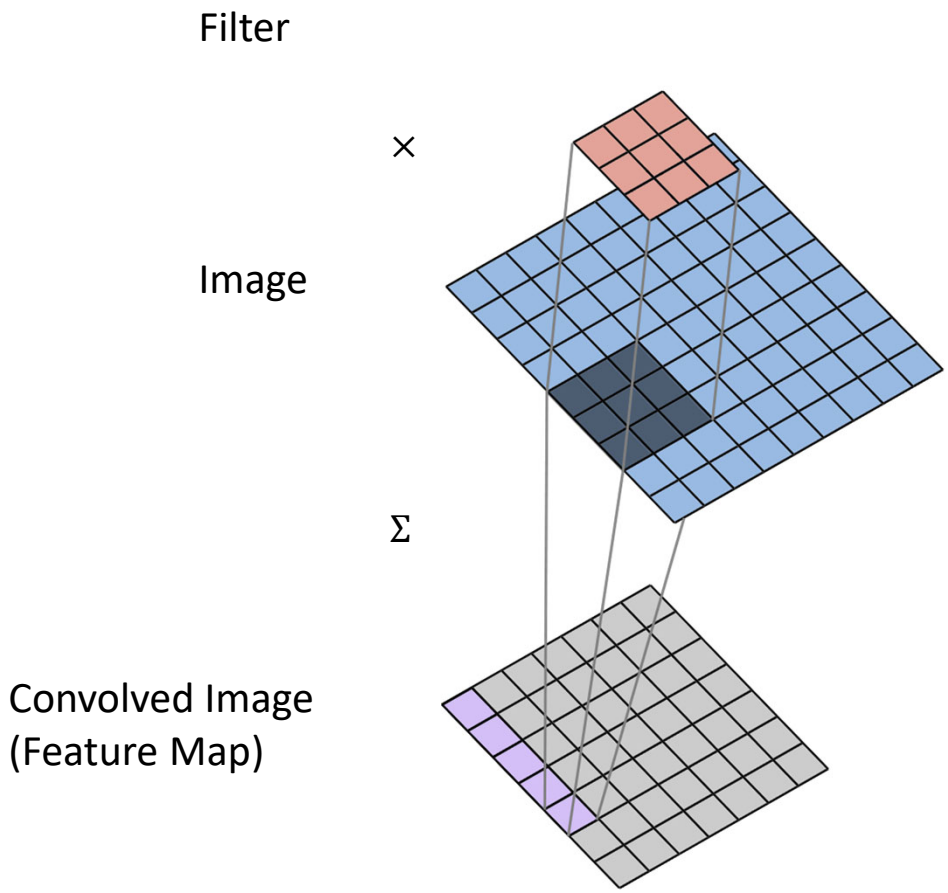
2D Spatial Convolution



2D Spatial Convolution



2D Spatial Convolution



2D Spatial Convolution

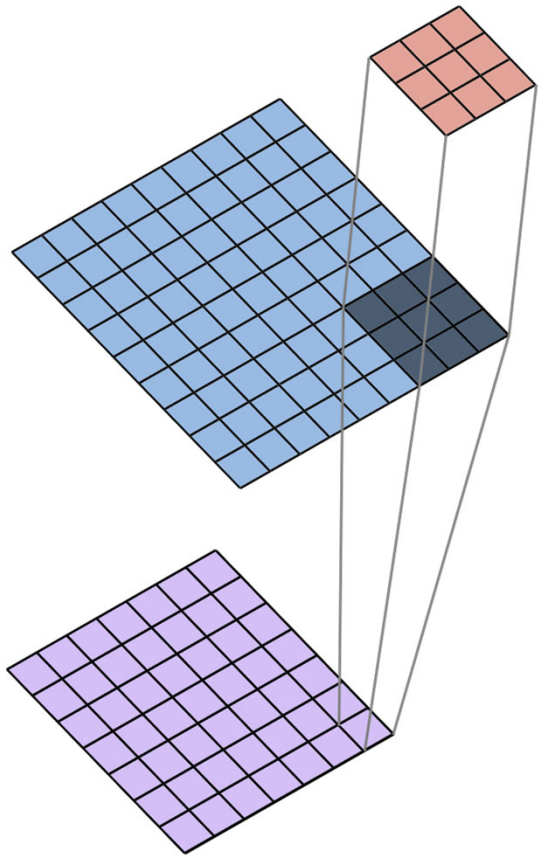
Filter

×

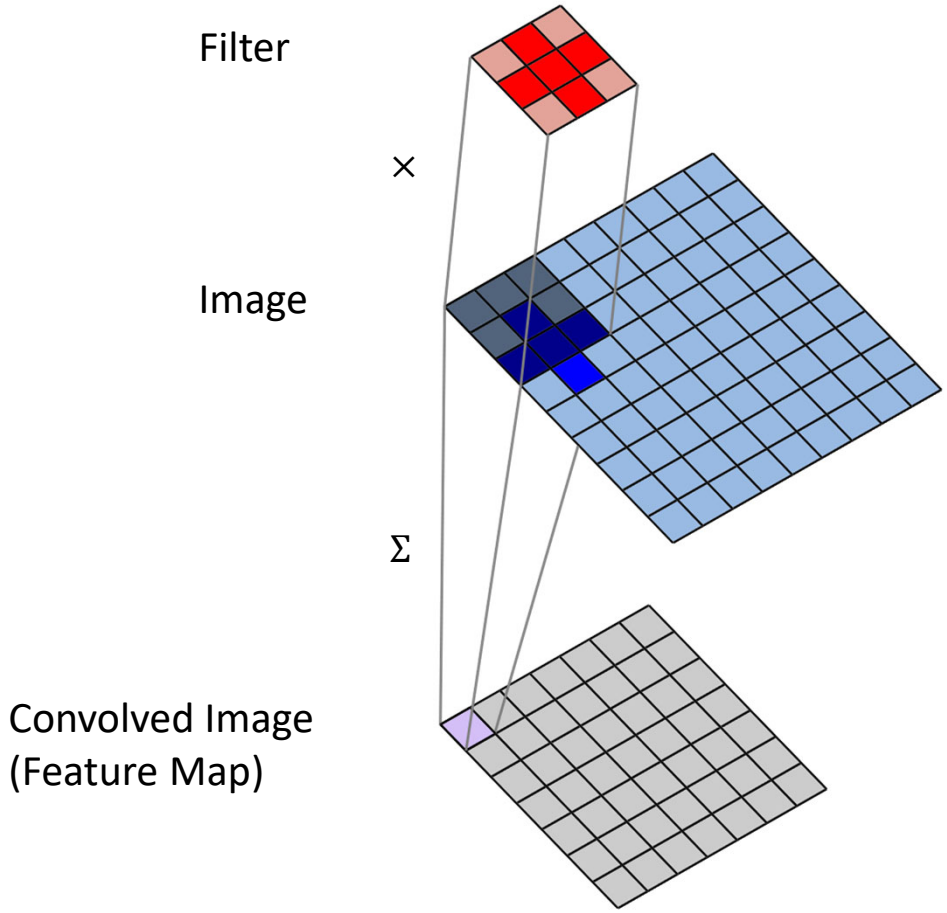
Image

Σ

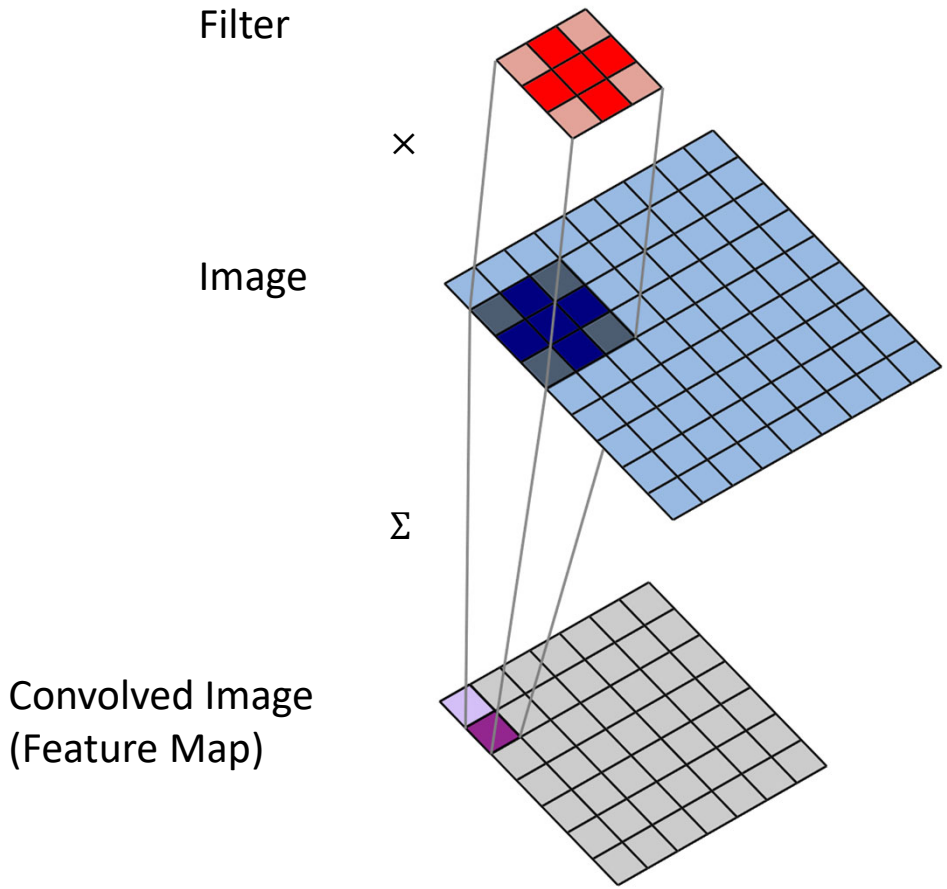
Convolved Image
(Feature Map)



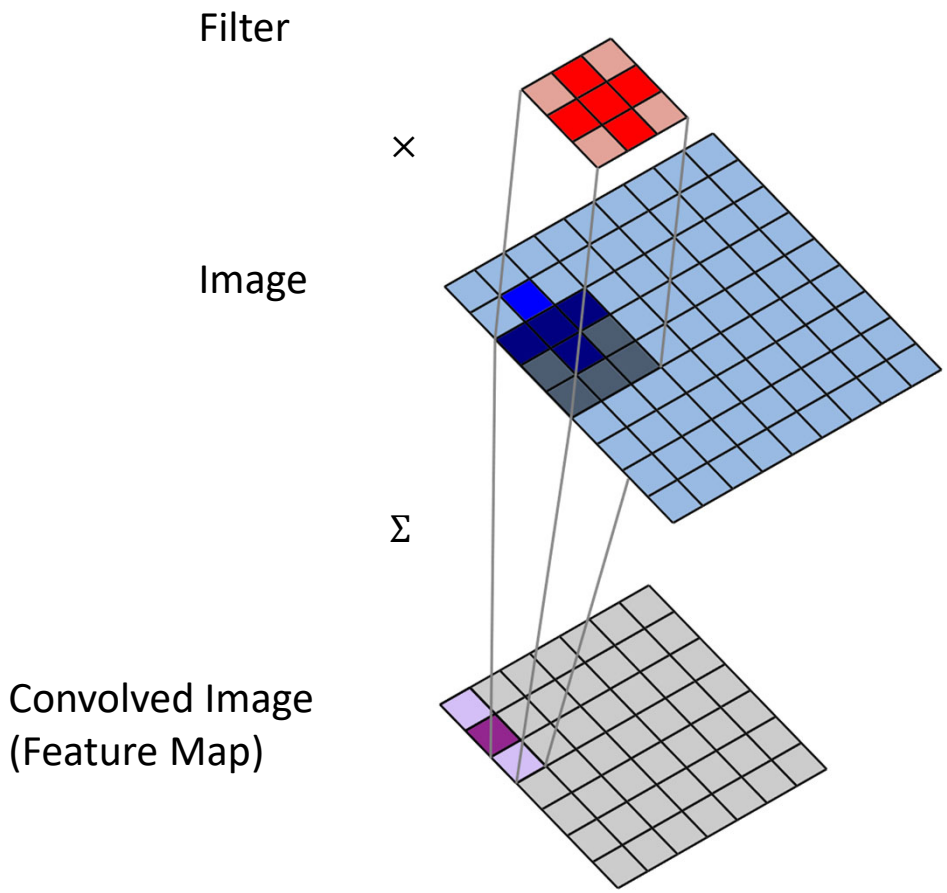
2D Spatial Convolution



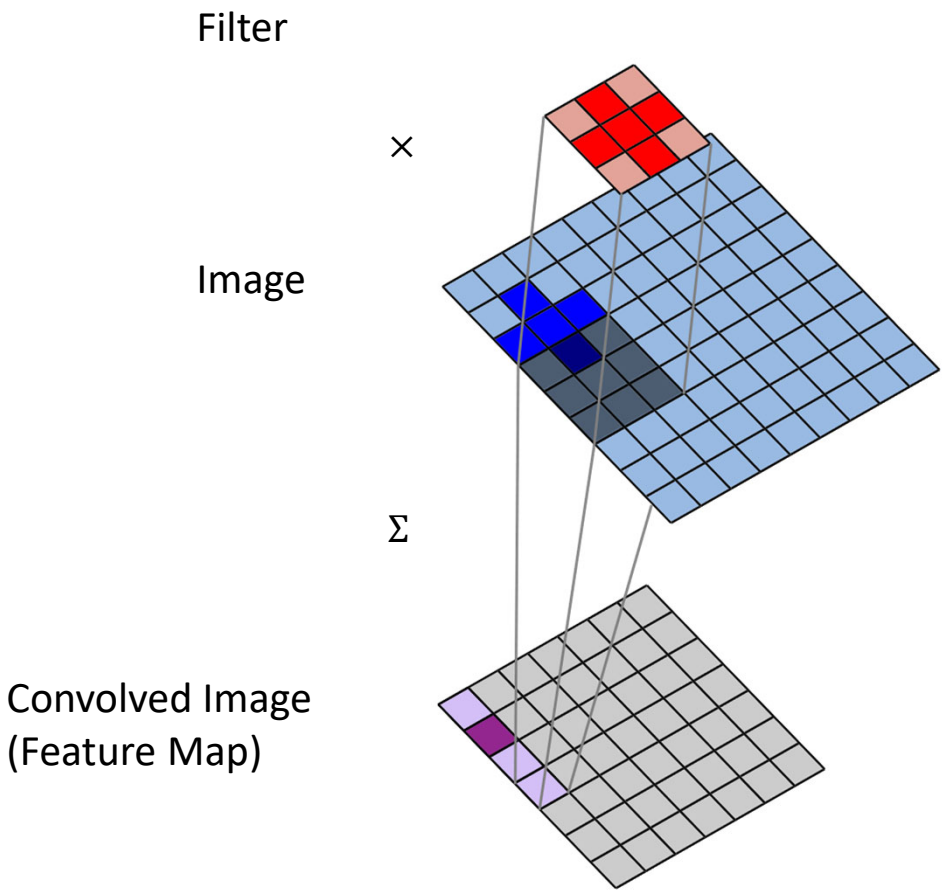
2D Spatial Convolution



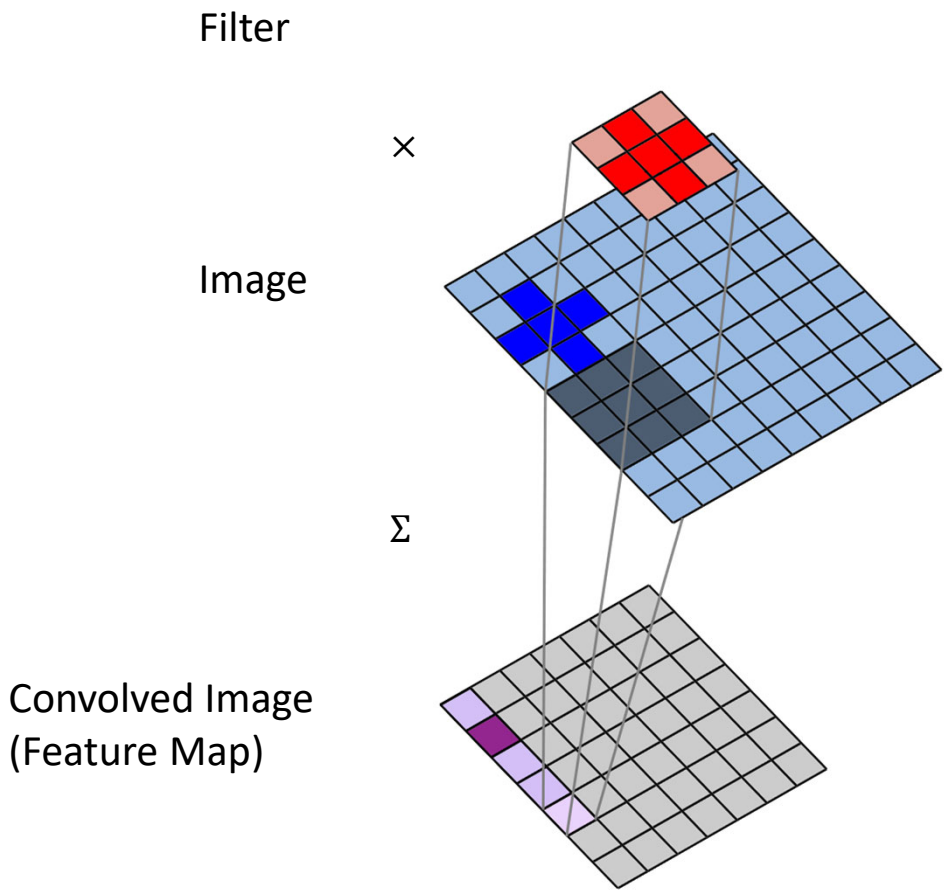
2D Spatial Convolution



2D Spatial Convolution



2D Spatial Convolution



2D Spatial Convolution

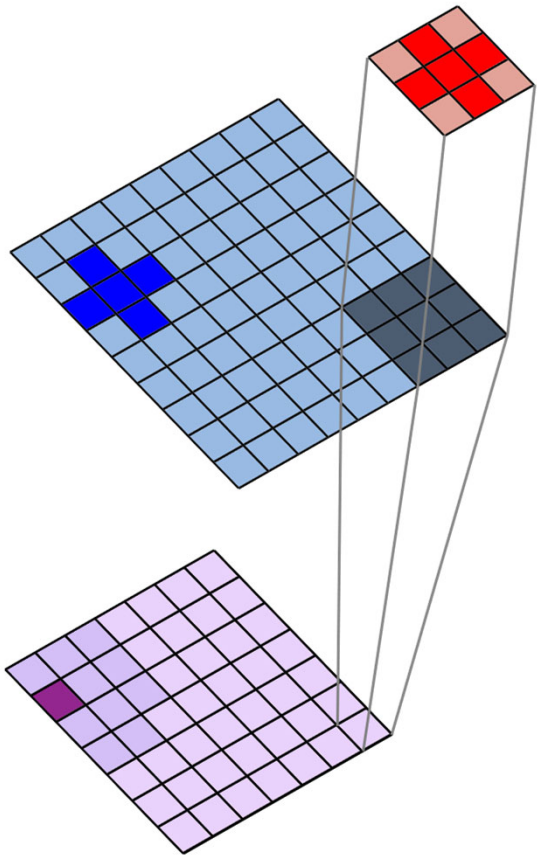
Filter

×

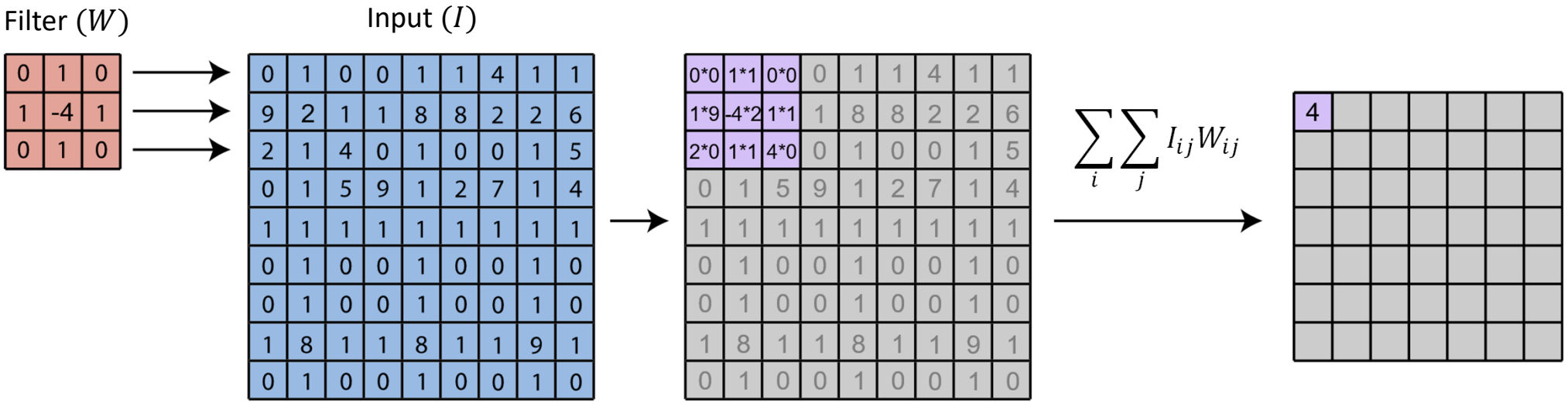
Image

Σ

Convolved Image
(Feature Map)



2D Spatial Convolution



2D Spatial Convolution

Filter (W)

0	1	0
1	-4	1
0	1	0

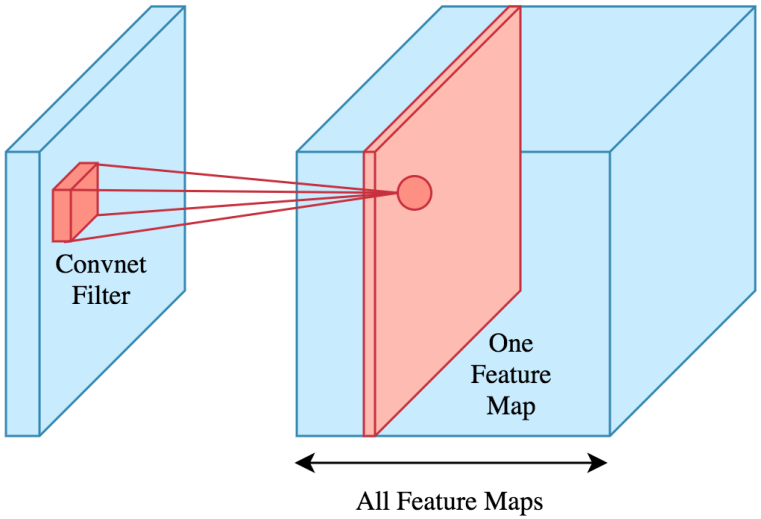
Input (I)

Convolutional Filters Are Feature Detectors

Core Elements of the Convolutional Neural Network

- **Convolutional Layers**
- **Activation Functions**
- **Pooling Layers**
- **Fully Connected Layers**

Convolutional Layer

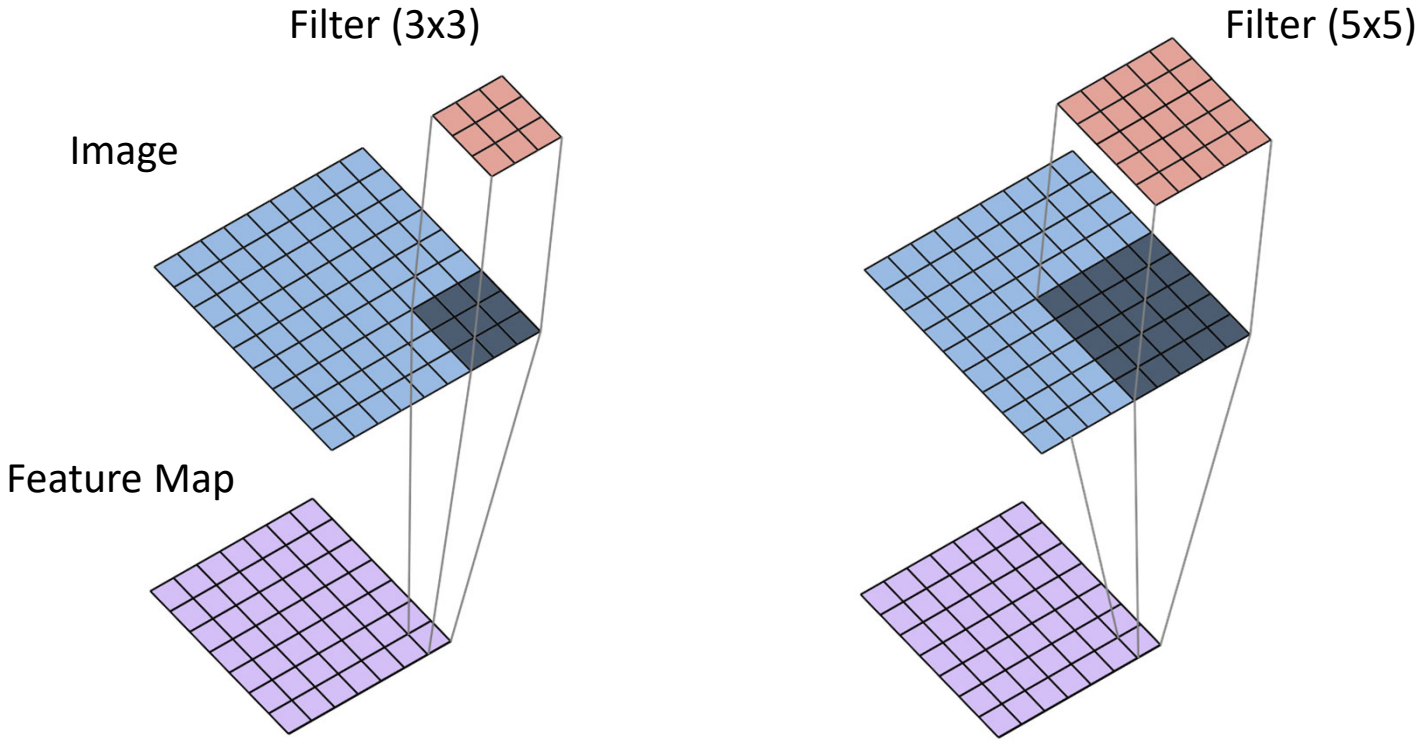


Elements of a convolutional layer:

Filter Size
Filter Stride
Filter (Feature) Number

Convolutional Layer

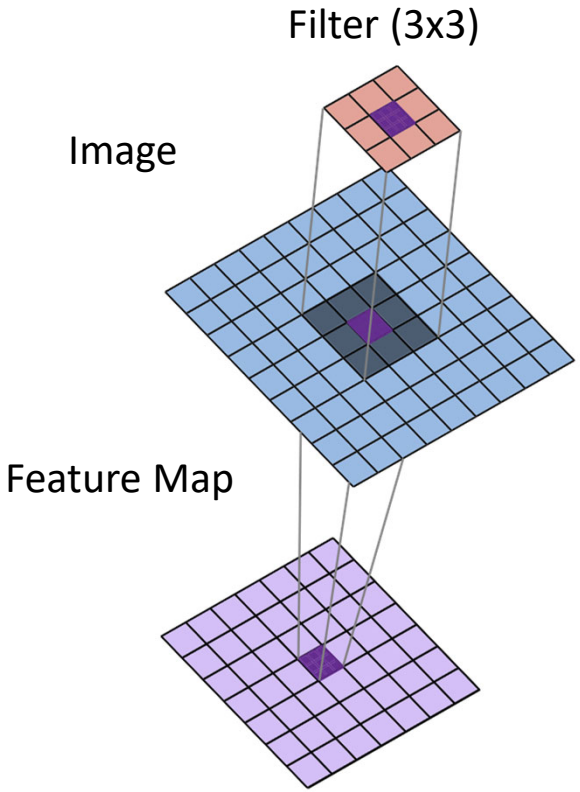
Filter Size



Filters should be just large enough to capture small local features (e.g. edges) in space

Convolutional Layer

Filter Size

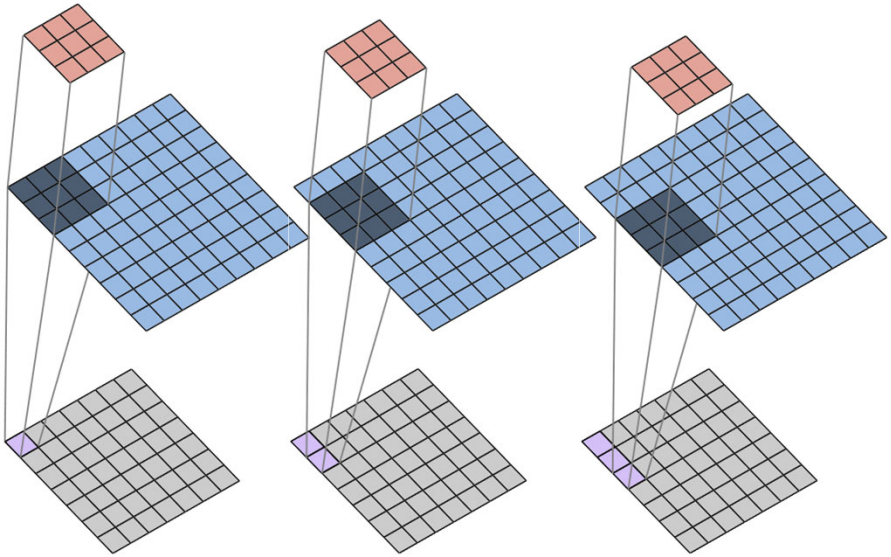


Filter size is typically odd to enforce symmetry

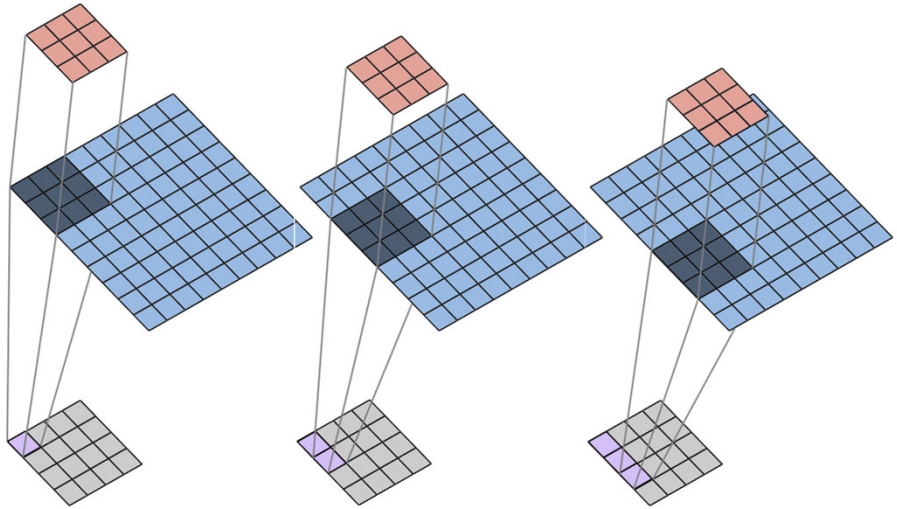
Convolutional Layer

Filter Stride

Stride = 1



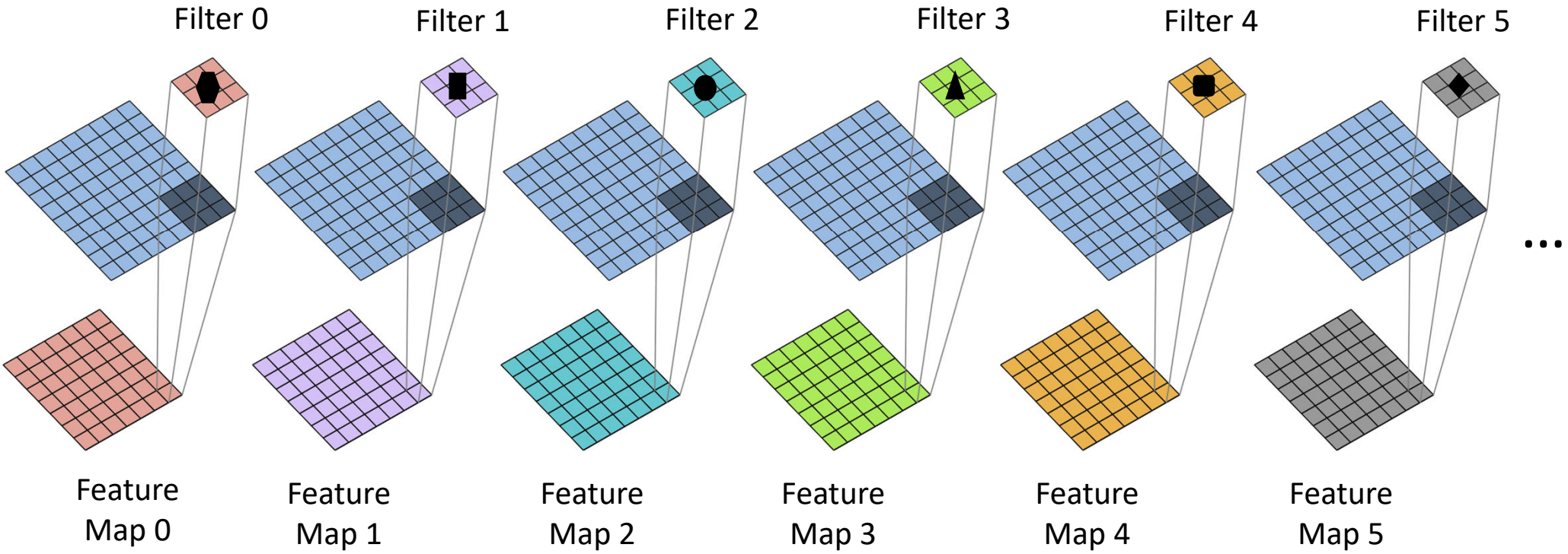
Stride = 2



Filter stride > 1 reduces computational load by downsampling the input

Convolutional Layer

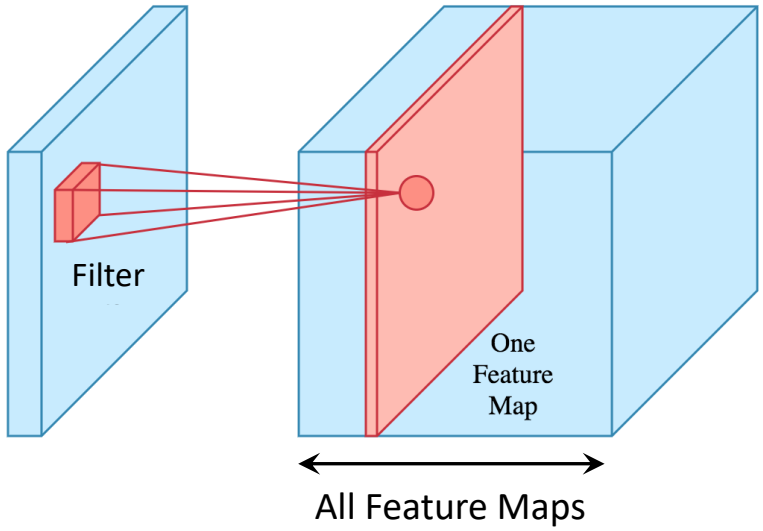
Filter (Feature) Number



Filter number determines the number of unique feature detectors that operate on inputs

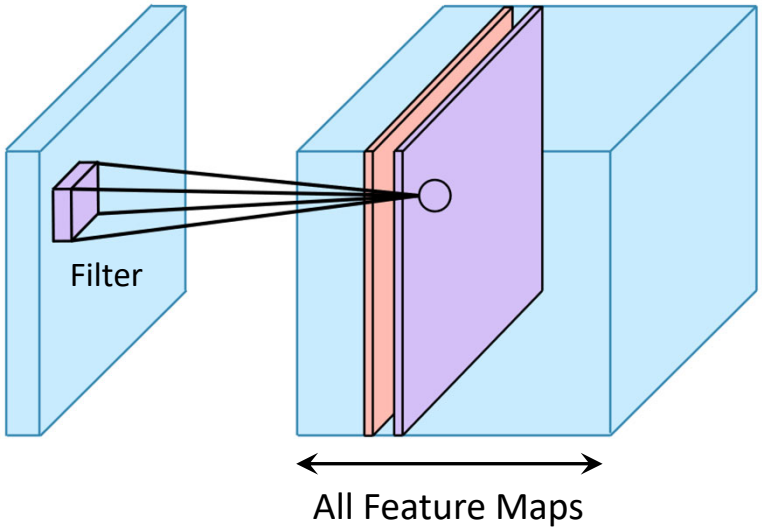
Convolutional Layer

Filter (Feature) Number



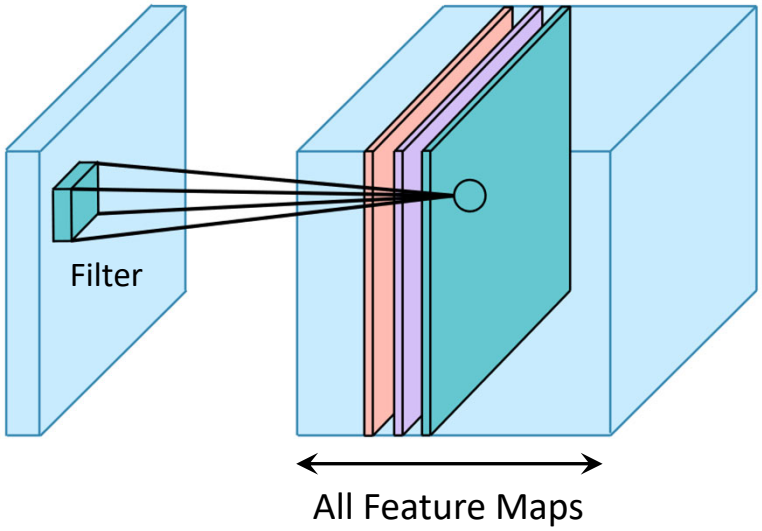
Convolutional Layer

Filter (Feature) Number



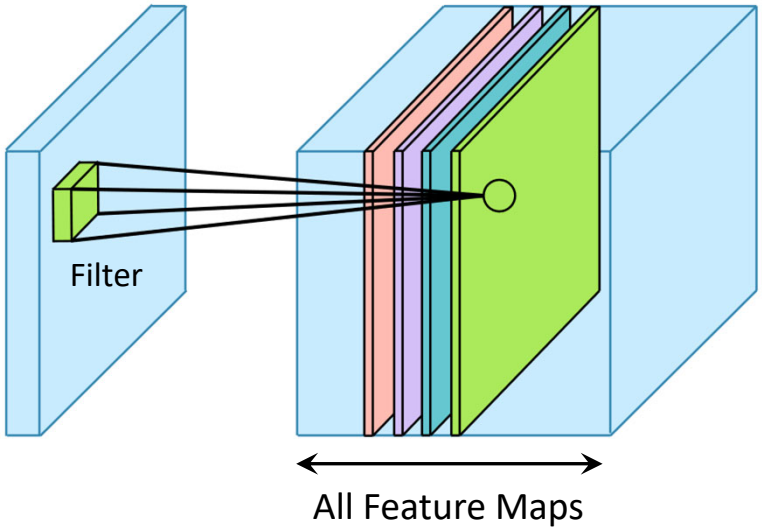
Convolutional Layer

Filter (Feature) Number



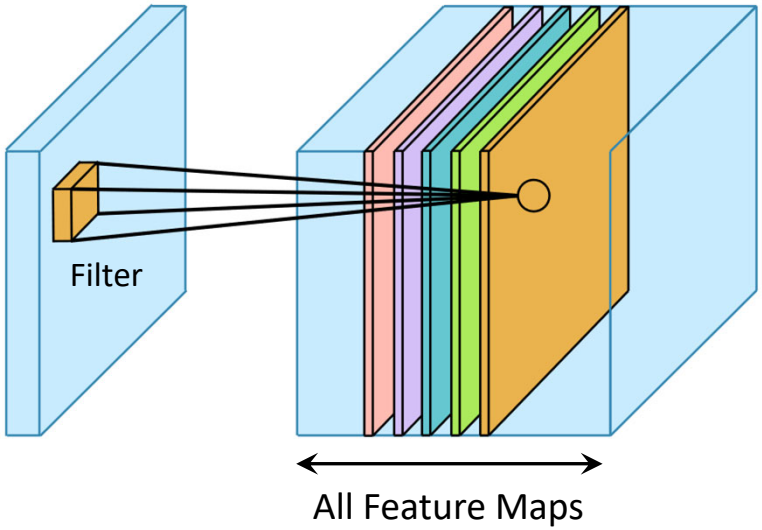
Convolutional Layer

Filter (Feature) Number



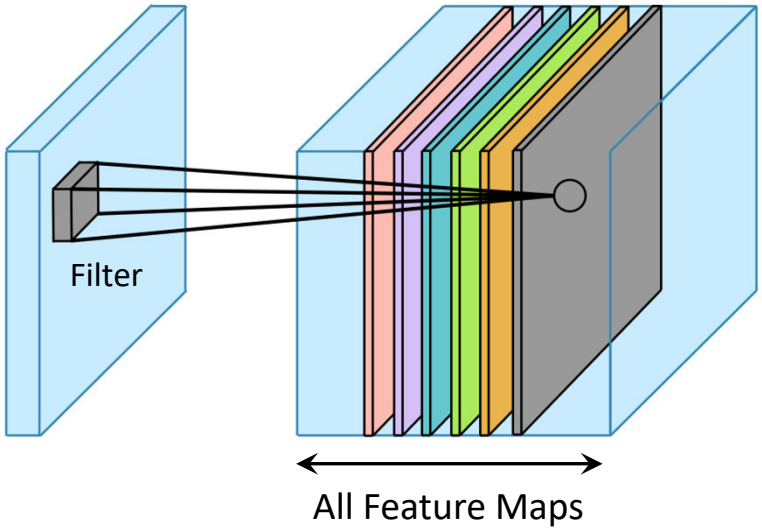
Convolutional Layer

Filter (Feature) Number

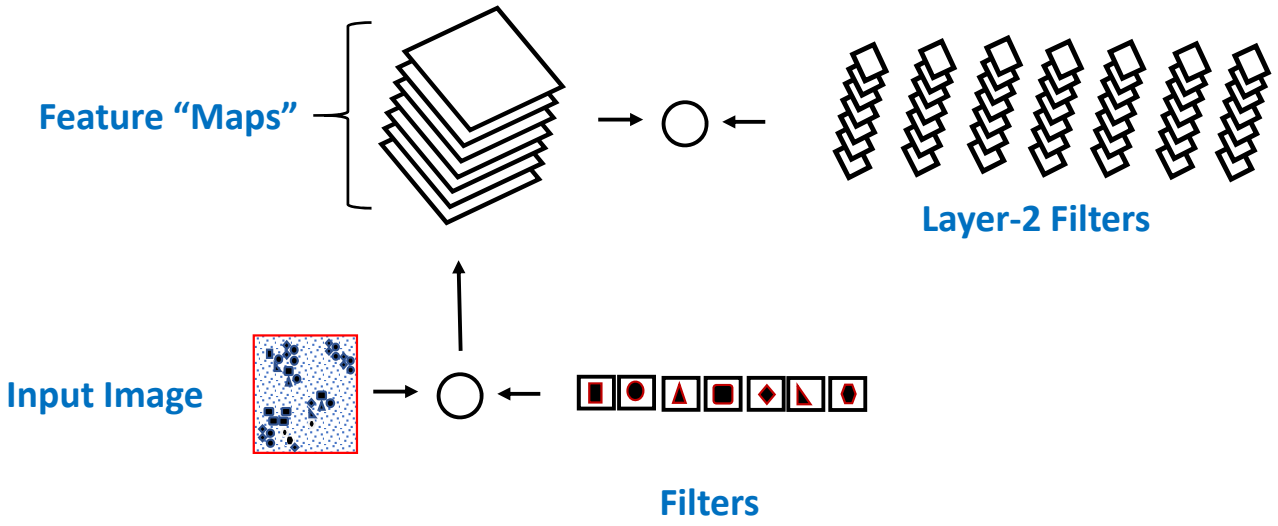


Convolutional Layer

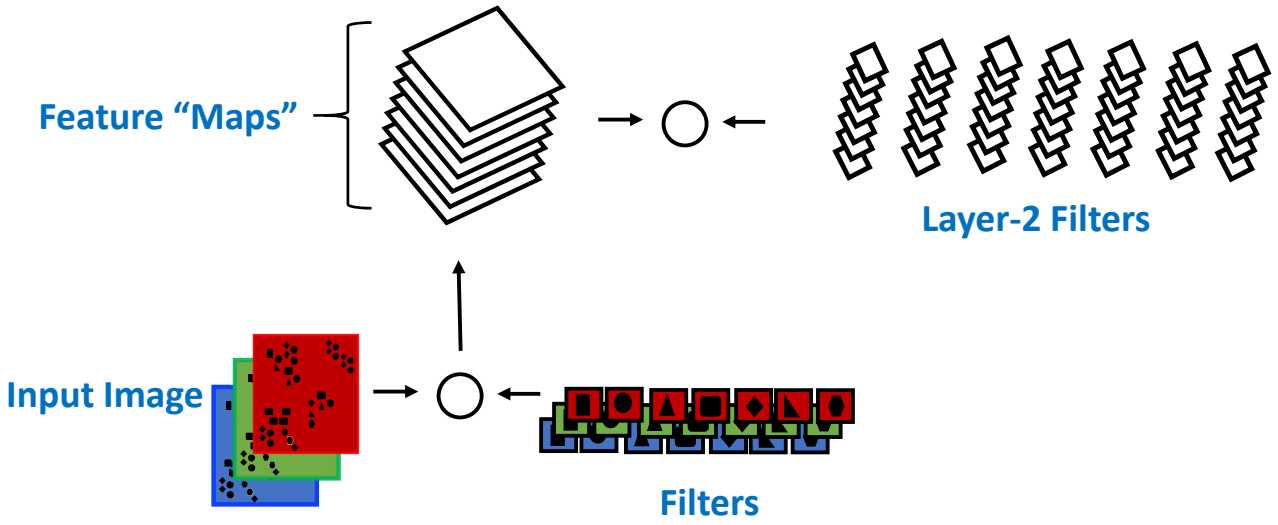
Filter (Feature) Number



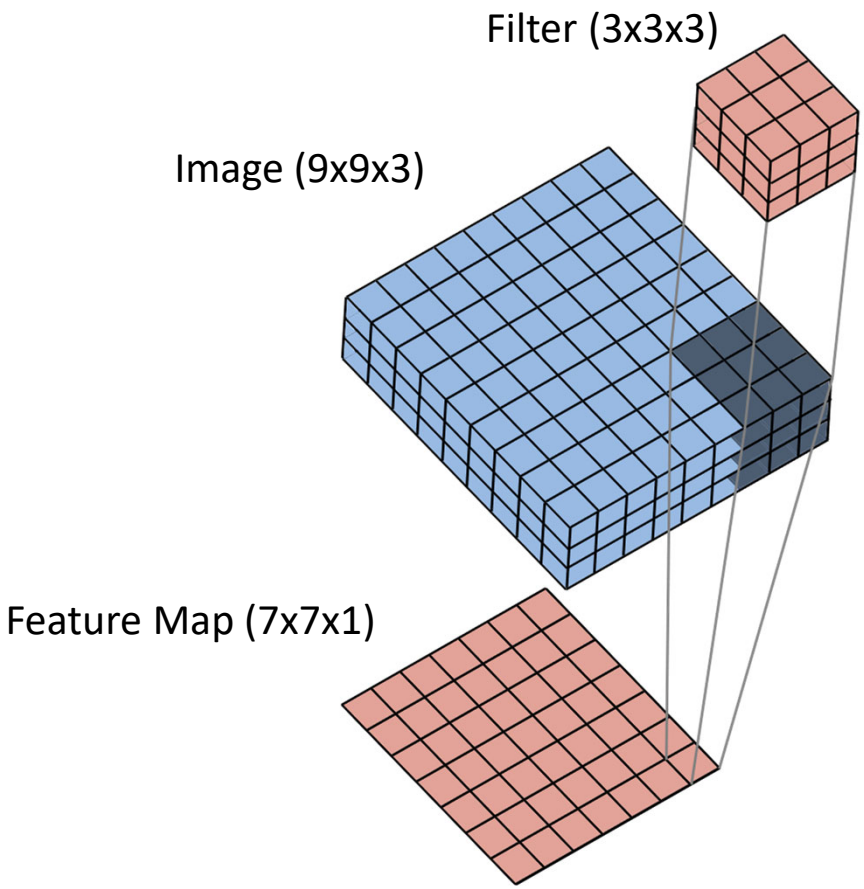
Filters Operate Over Input Volumes



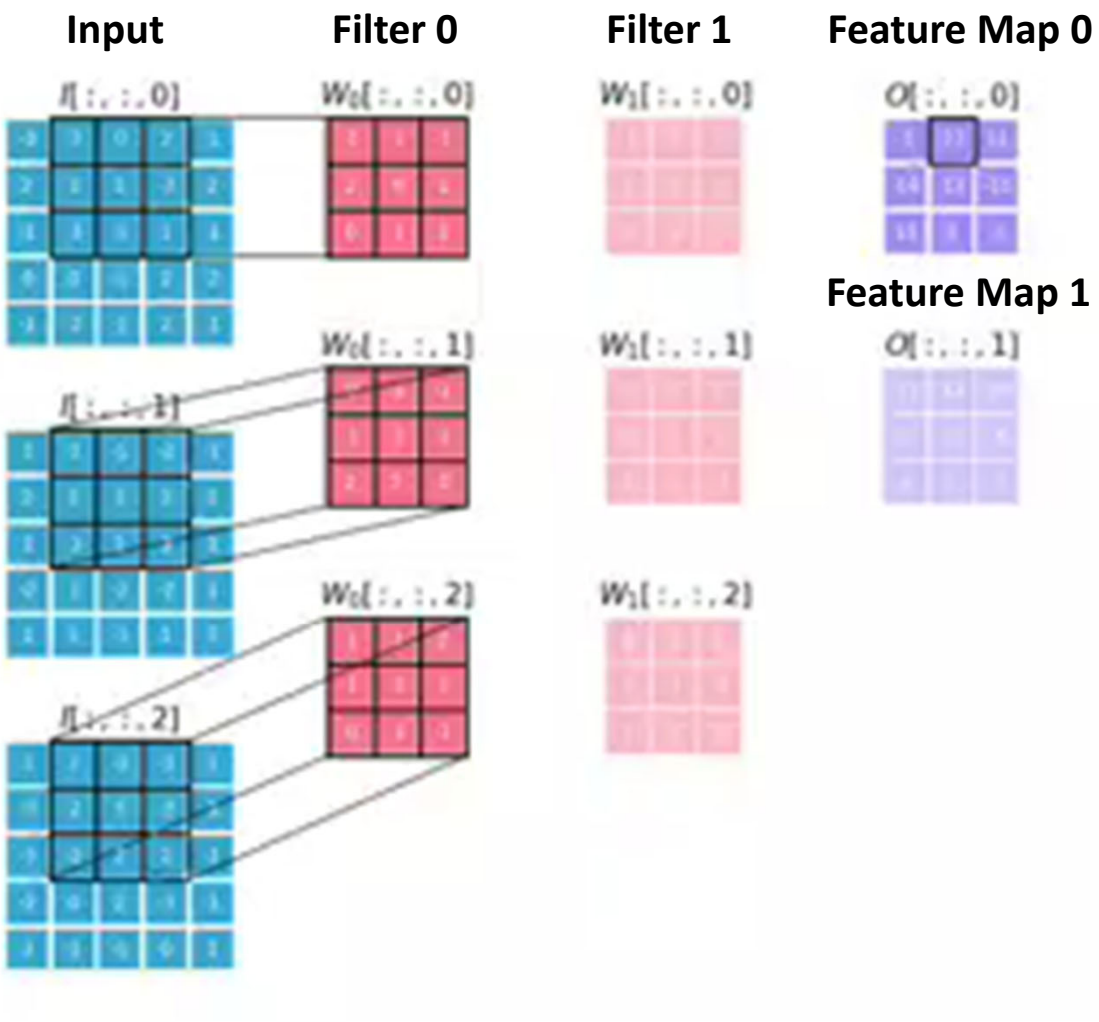
Filters Operate Over Input Volumes



Filters Operate Over Input Volumes

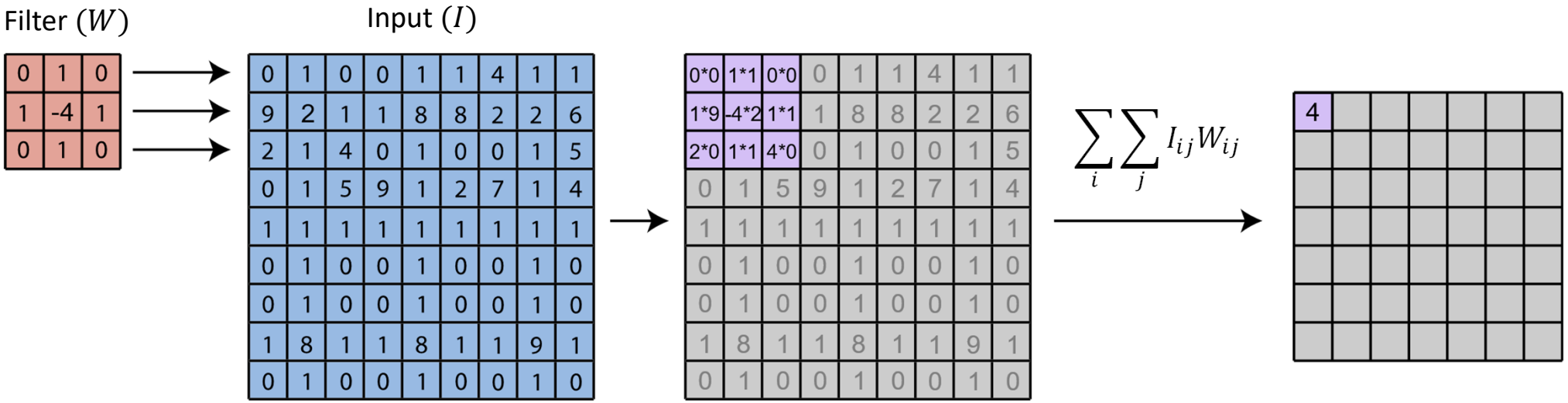


Convolutional Layer

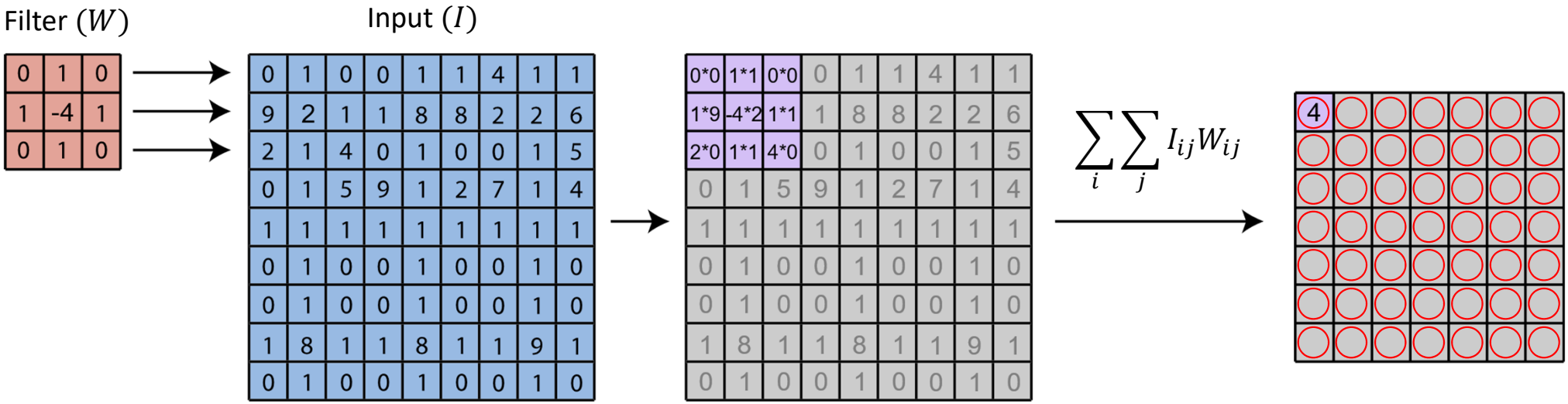


Inspired by Stanford 231n

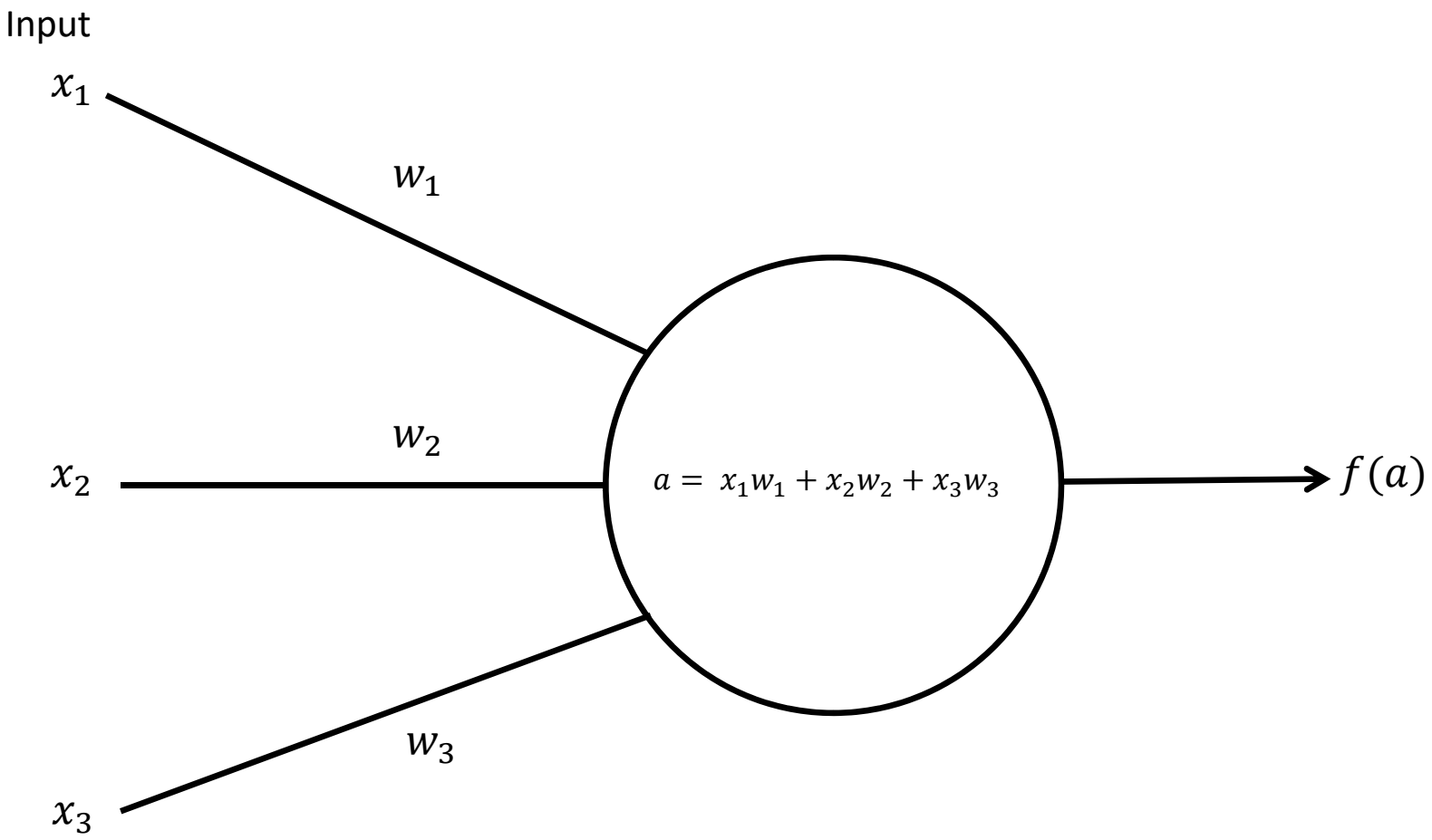
Activation Functions



Activation Functions

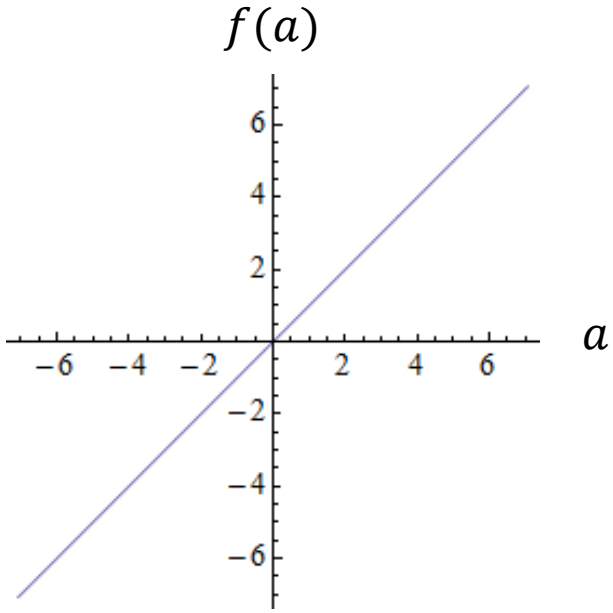


Activation Functions



Activation Functions

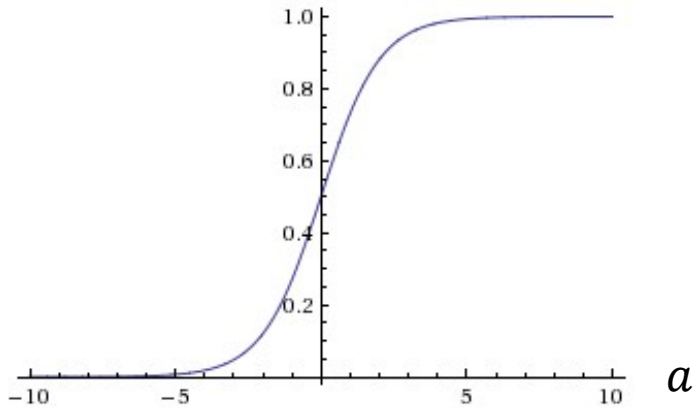
Linear Activation



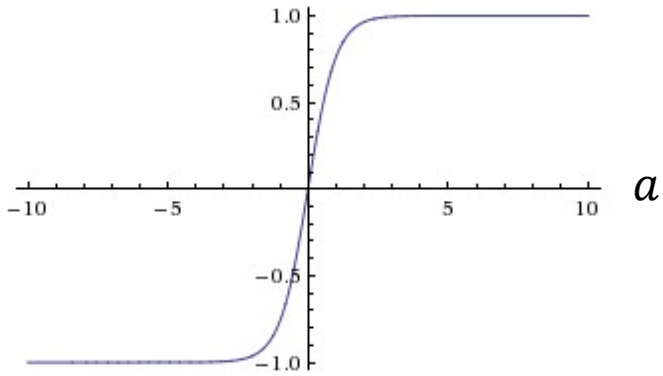
Activation Functions

Non-Linear Activations

$f(a) = \sigma(a)$



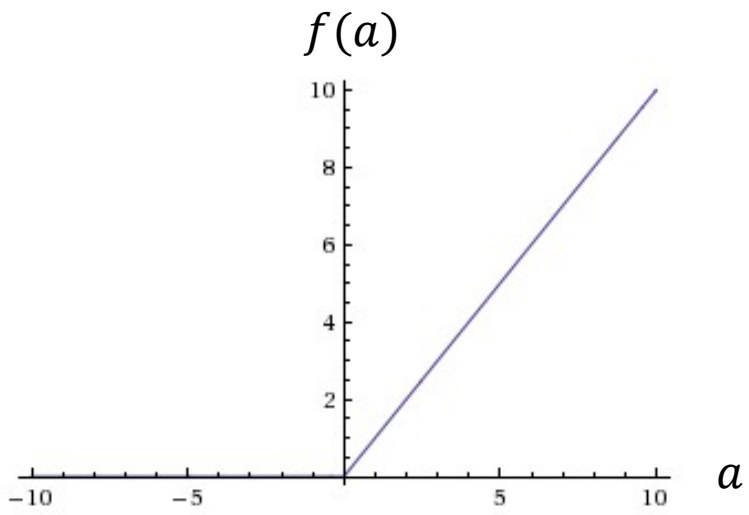
$f(a) = \tanh(a)$



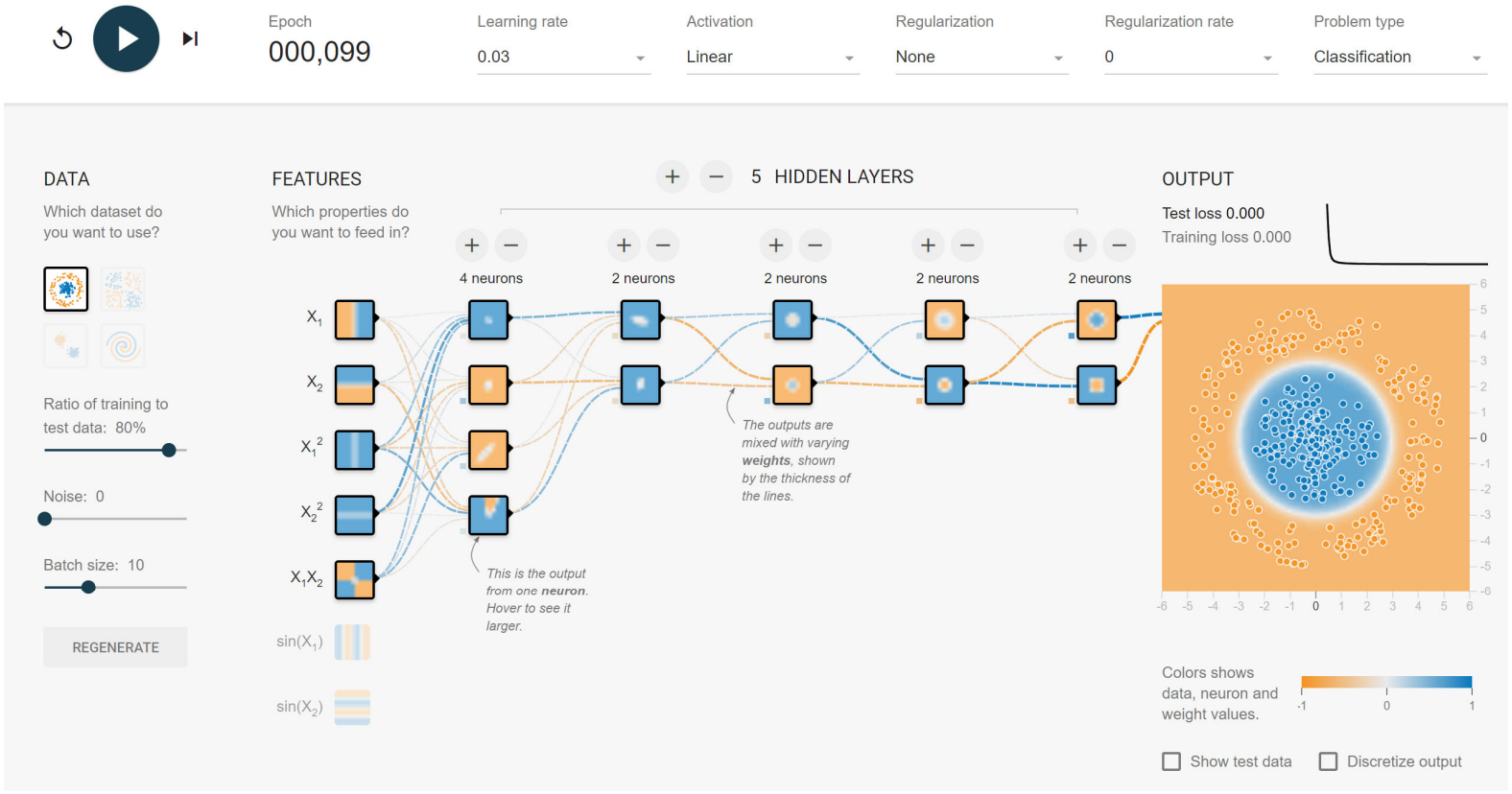
Non-linear activations increase the functional capacity of the neural network

Activation Functions

Non-Linear Activation:
Rectified Linear Unit (ReLU)

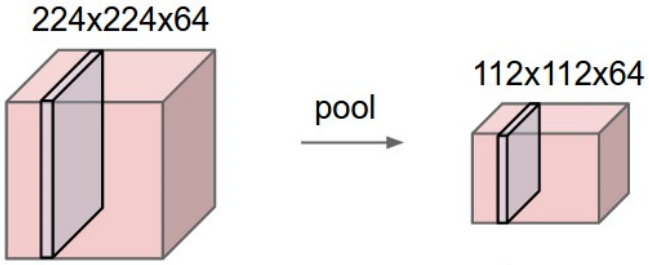


Activation Functions



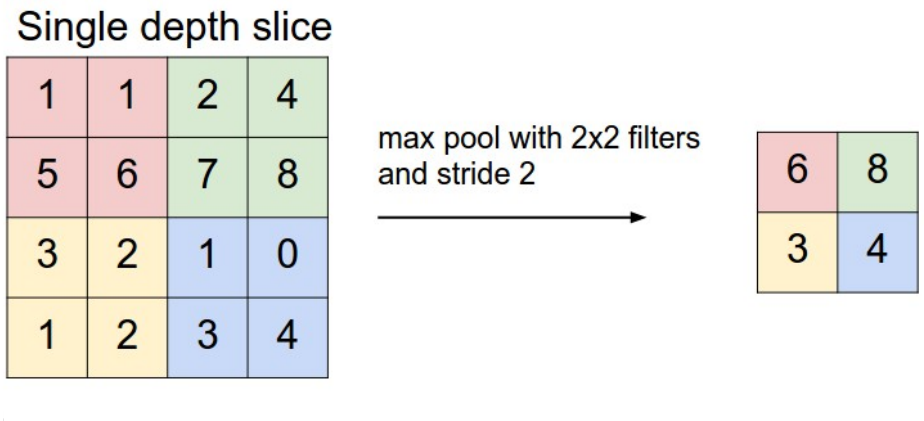
playground.tensorflow.org

Pooling Layer



- Reduces computational complexity
- Combats overfitting
- Encourages translational invariance

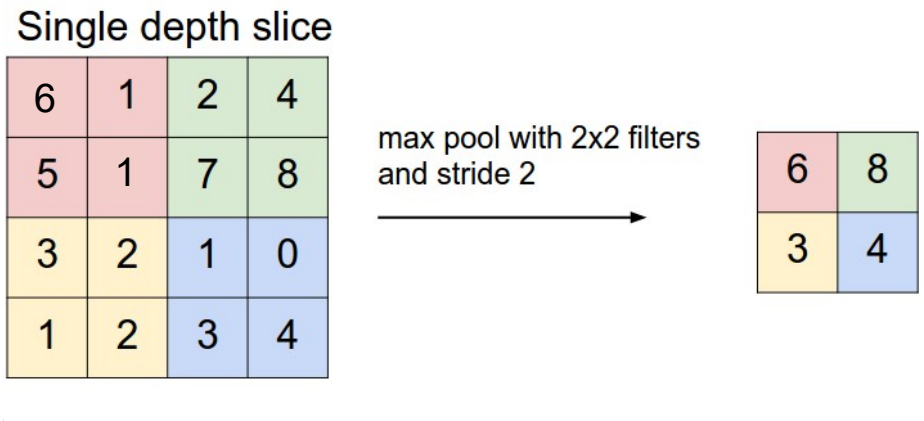
Pooling Layer



Pooling layers also have **width** and **stride**.

Pooling is typically done by taking the **maximum** across the pooling area

Pooling Layer

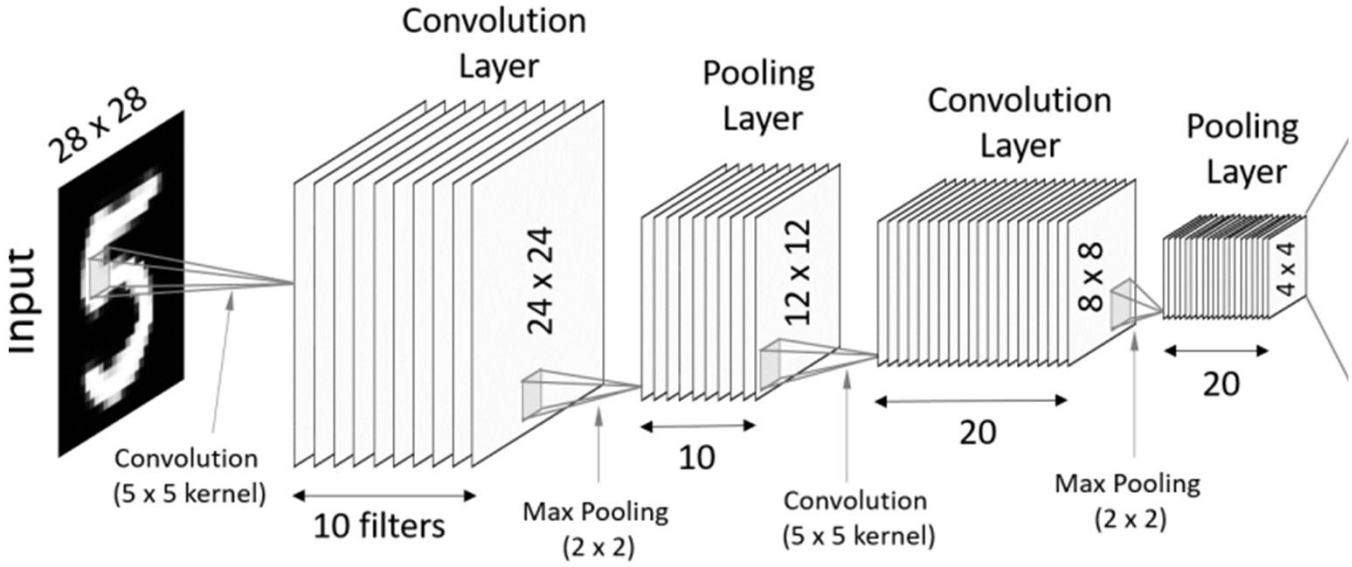


Max Pooling picks out strong activations with some position independence

Fully Connected Layer

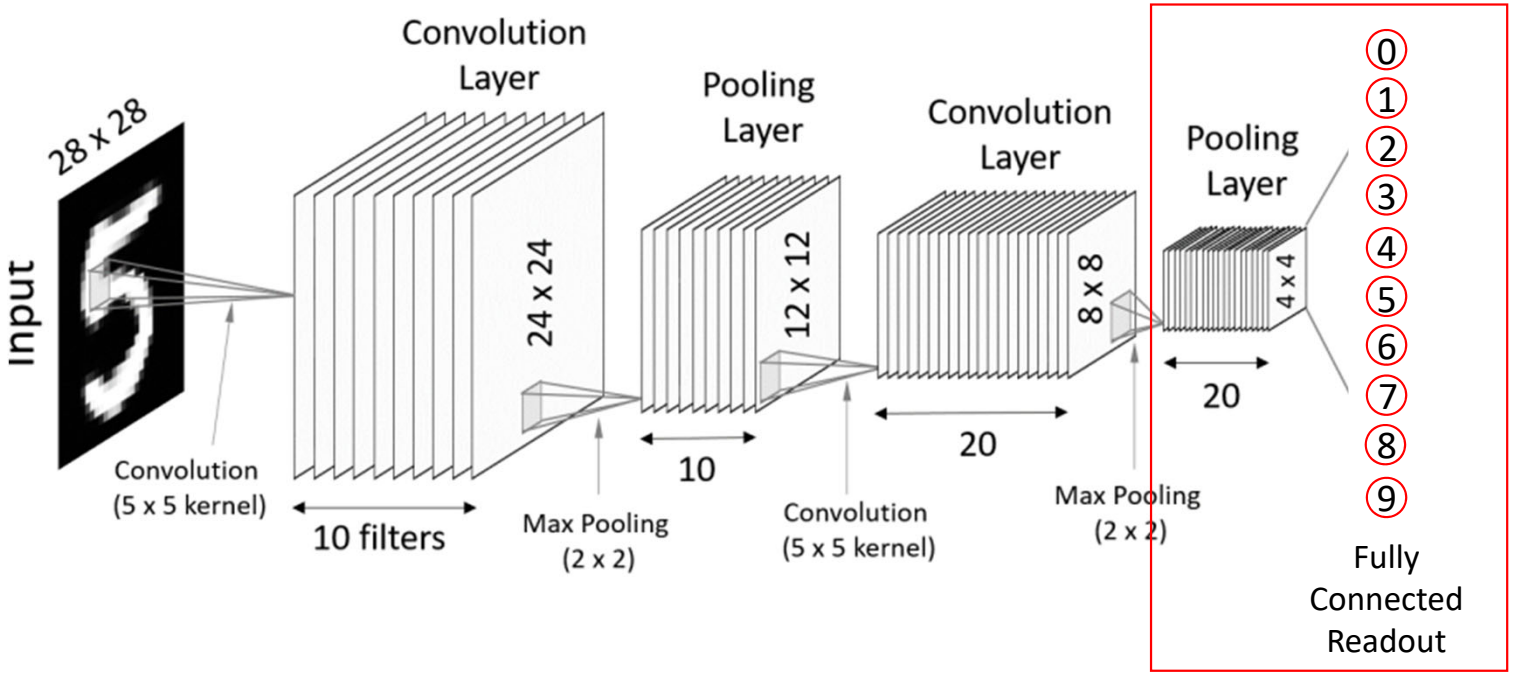
Convolutional and pooling layers are stacked to build up high-level feature representations

How are these high-level features processed to arrive at a final classification?



Fully Connected Layer

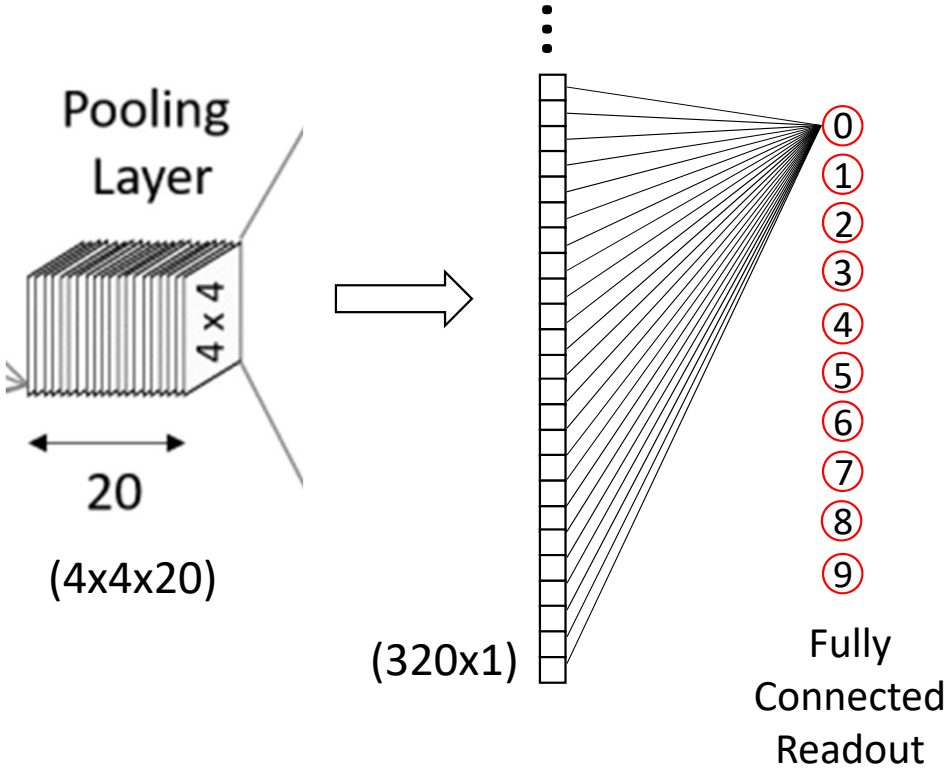
The most basic way is to have a final **fully connected** readout layer with as many neurons as there are classes



Fully Connected Layer

Fully connected means each neuron takes input from all neurons in the final set of feature maps

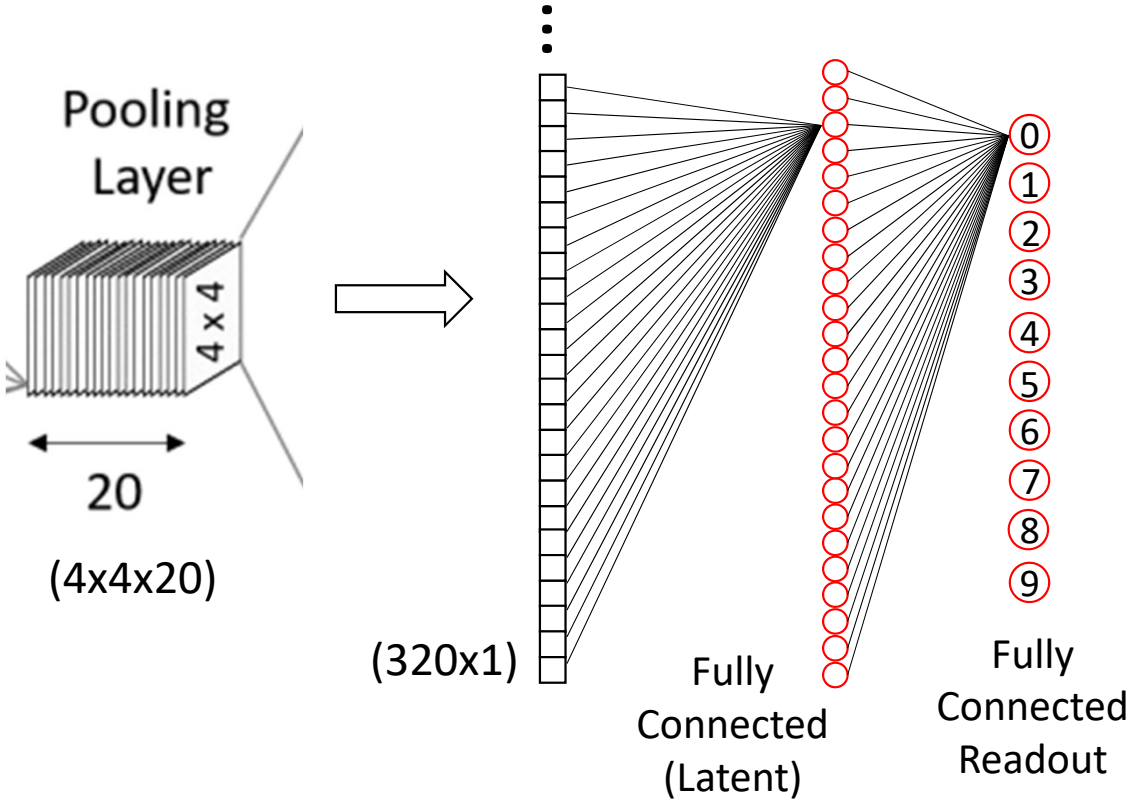
The final set of feature maps are vectorized to create an MLP-like configuration



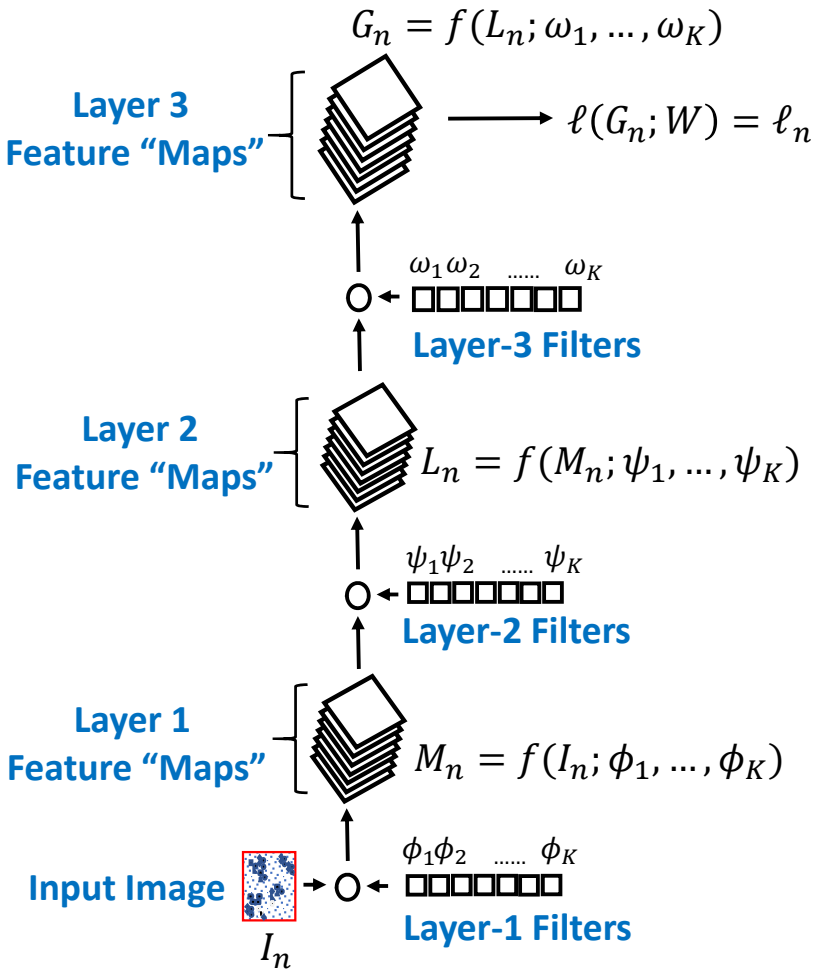
Fully Connected Layer

But it is also common to stack multiple fully connected layers before the final readout layer

Neurons in these intermediate layers can be considered **latent** classes



Review: Training A Deep Convolutional Neural Network

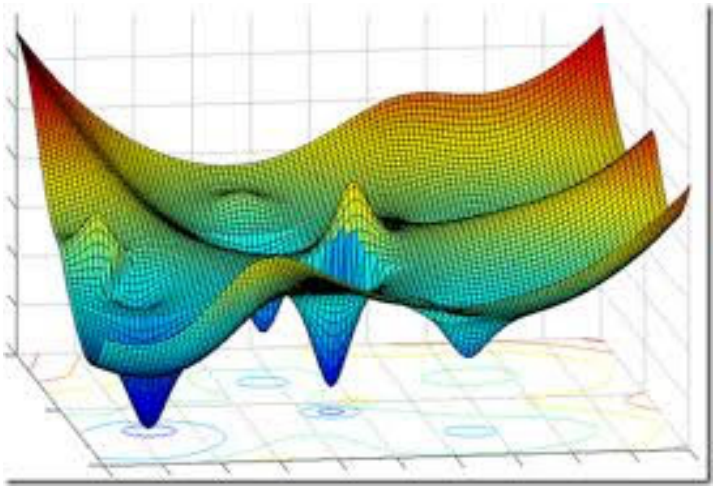


- Assume we have labeled images $\{I_n, y_n\}_{n=1,N}$
- I_n is image n, $y_n \in \{+1, -1\}$ is associated label
- Risk function of model parameters:

$$E(\Phi, \Psi, \Omega, W) = 1/N \sum_{n=1}^{N} loss(y_n, \ell_n)$$

- Find model parameters $\hat{\Phi}, \hat{\Psi}, \hat{\Omega}, \hat{W}$ that minimize $E(\Phi, \Psi, \Omega, W)$

Cost Function vs. Model Parameters



- High-dimensional function, as a consequence of a large number of model parameters
- Typically many local minima
- May be expensive to compute, for sophisticated models & large quantity of training images

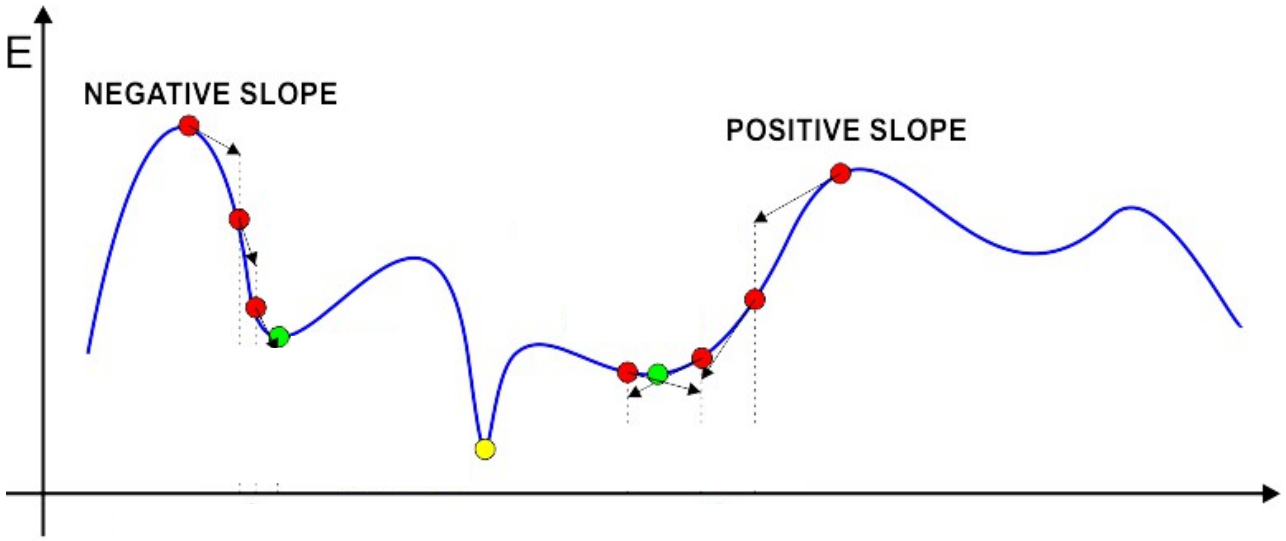
Gradient Descent

$$E(\Phi, \Psi, \Omega, W) = 1/N \sum_{n=1}^{N} loss(y_n, \ell_n)$$

- For large number of training pairs *N*, computation of risk $E(\cdot)$ could be prohibitive

- How do we optimize for Φ, Ψ, Ω, W?

Optimization for $\Theta = \{\Phi, \Psi, \Omega, \mathbf{W}\}$?



Gradient Descent

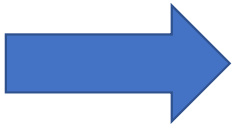
$$\Theta_{t+1} = \Theta_t - \alpha \underbrace{\nabla_\Theta E(\Theta_t)}$$

Multi-dimensional "slope"

Massive N ?

Gradient Descent

$$\Theta_{t+1} = \Theta_t - \alpha \nabla_\Theta E(\Theta_t)$$



Stochastic Gradient Descent

$$\Theta_{t+1} = \Theta_t - \alpha \nabla_\Theta \hat{E}_t(\Theta_t)$$

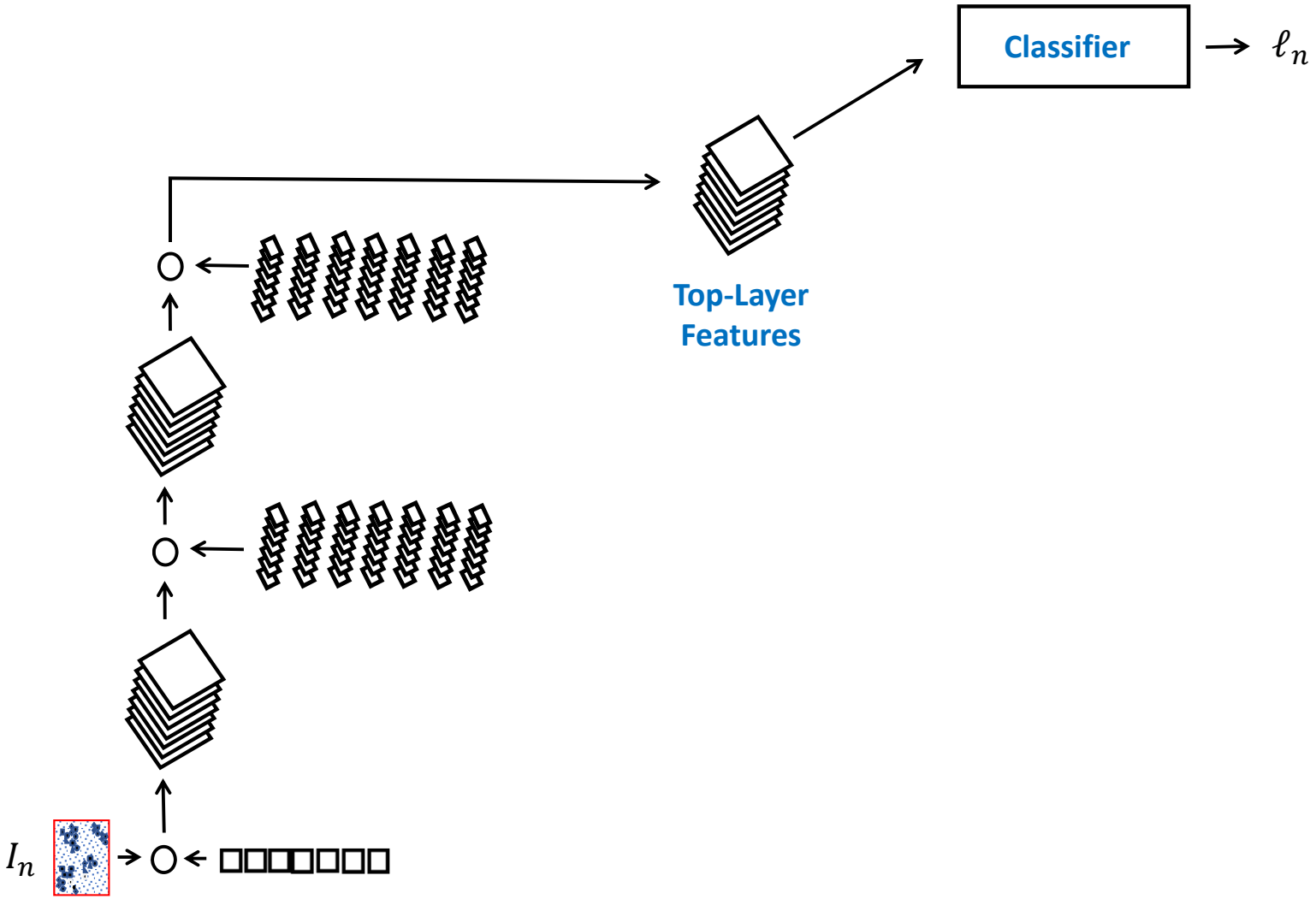
$$\hat{E}_t(\Phi, \Psi, \Omega, W) = 1/|S_t| \sum_{n \in S_t} loss(y_n, \ell_n)$$

S_t a *random* subset of data, at iteration *t*

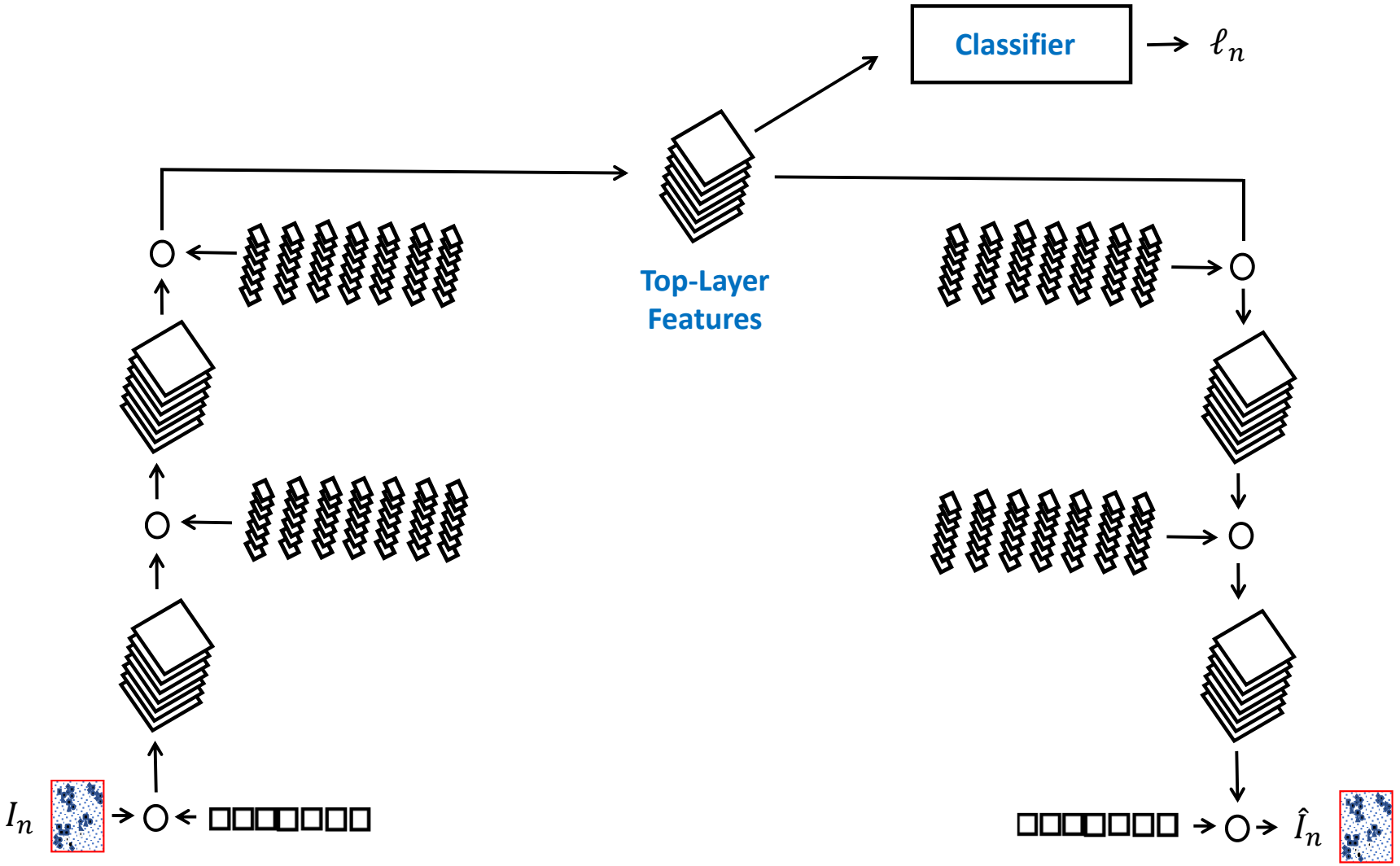
Summary

- Convolutional neural networks learn to recognize **high-level structure** in images by building **hierarchical representations of features**
- Features are extracted via spatial convolutions with **filters**
- Filters are learned via iterative minimization of a risk function
- Convolutional neural networks have shown capabilities beyond human performance for image analysis

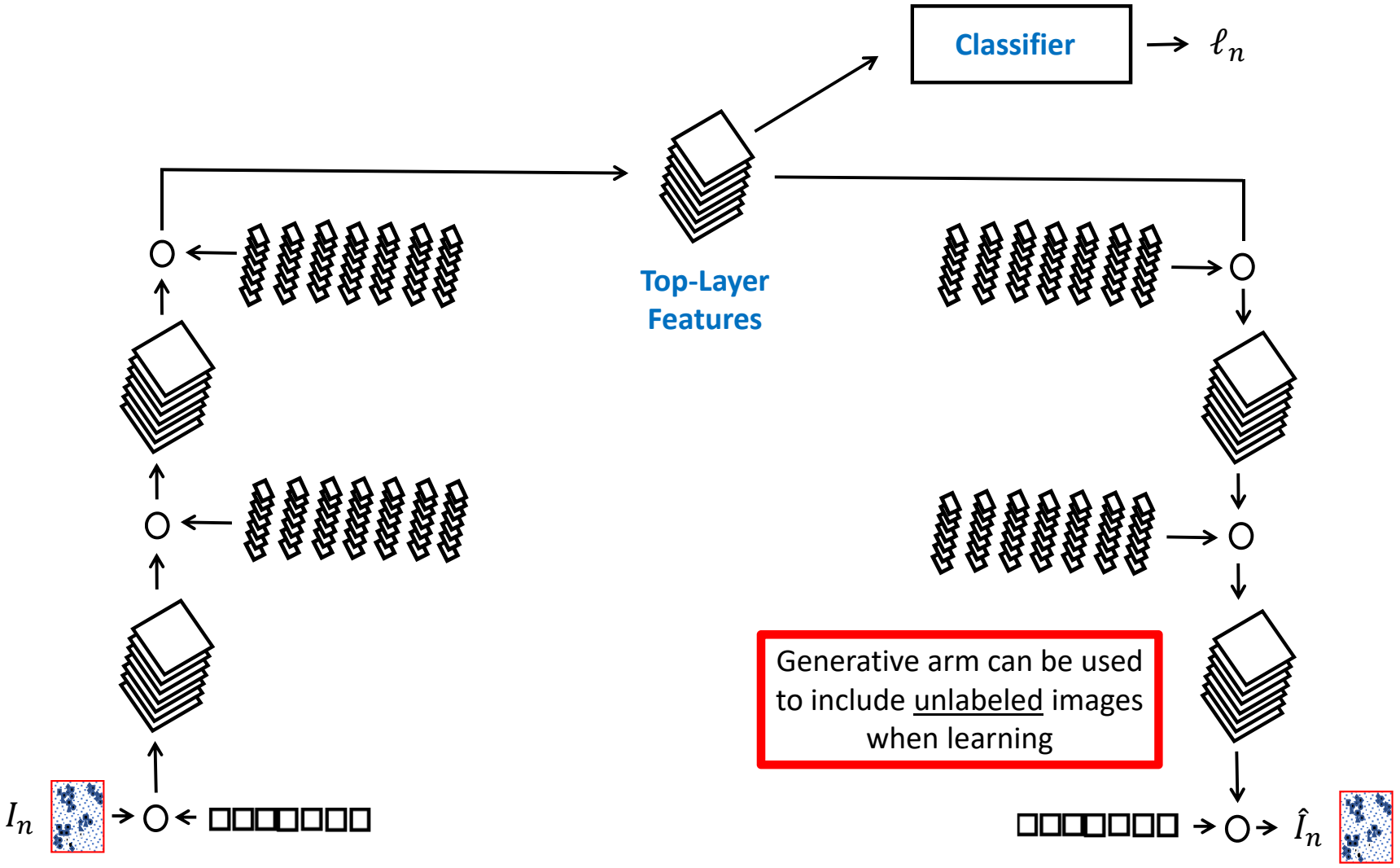
Deep Architecture



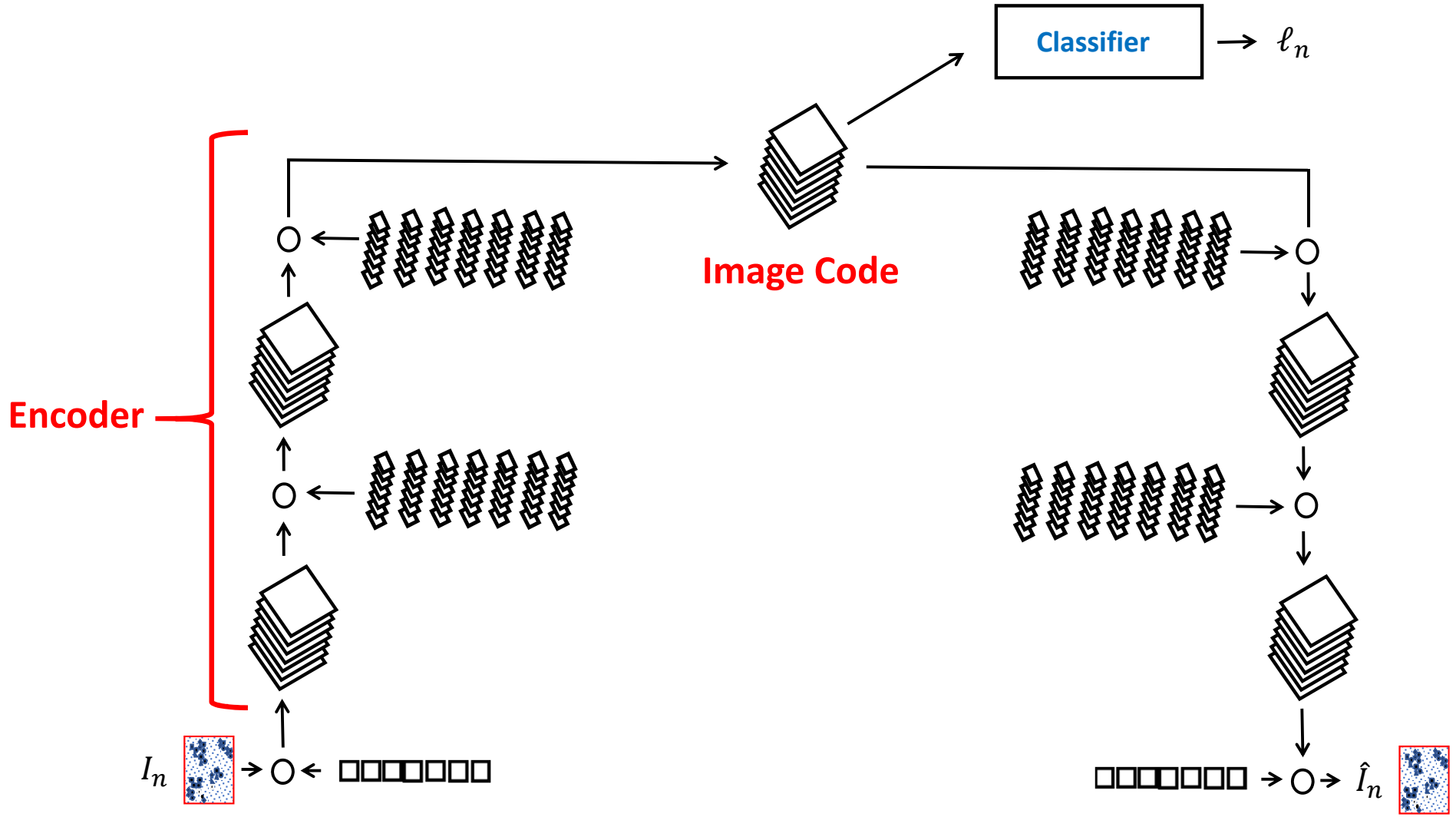
Add Generative Arm



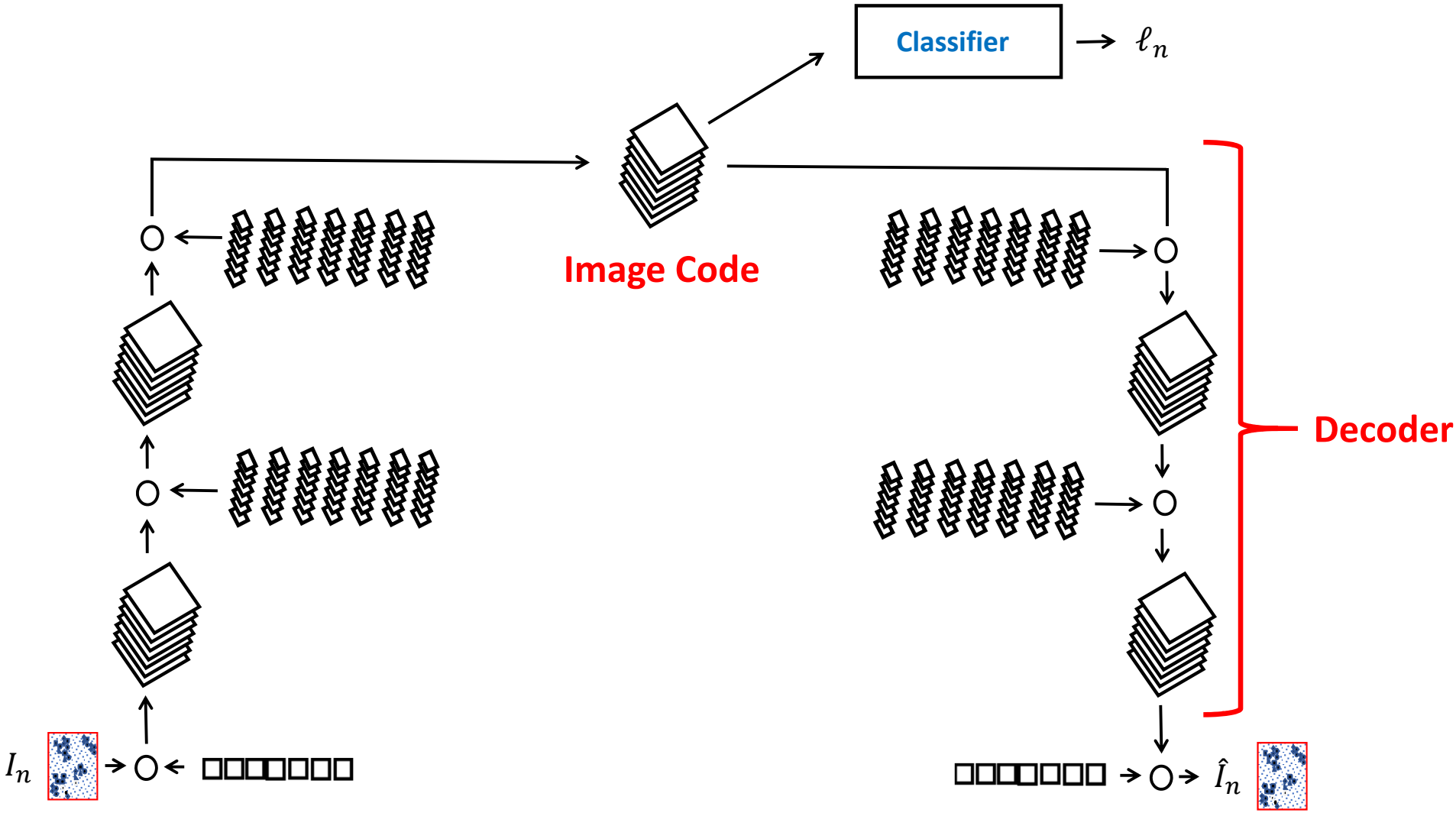
Add Generative Arm



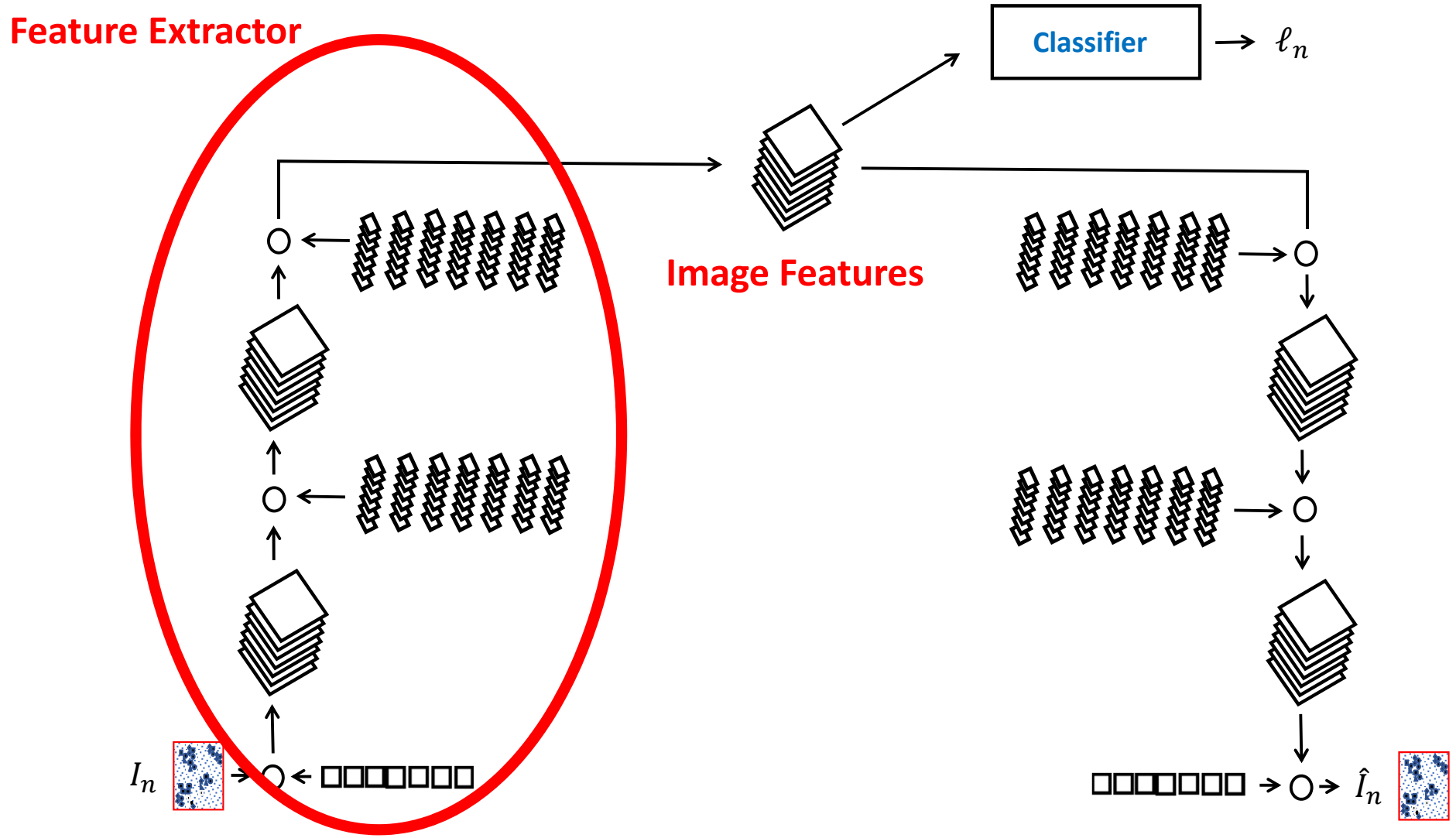
Deep Analysis Architecture



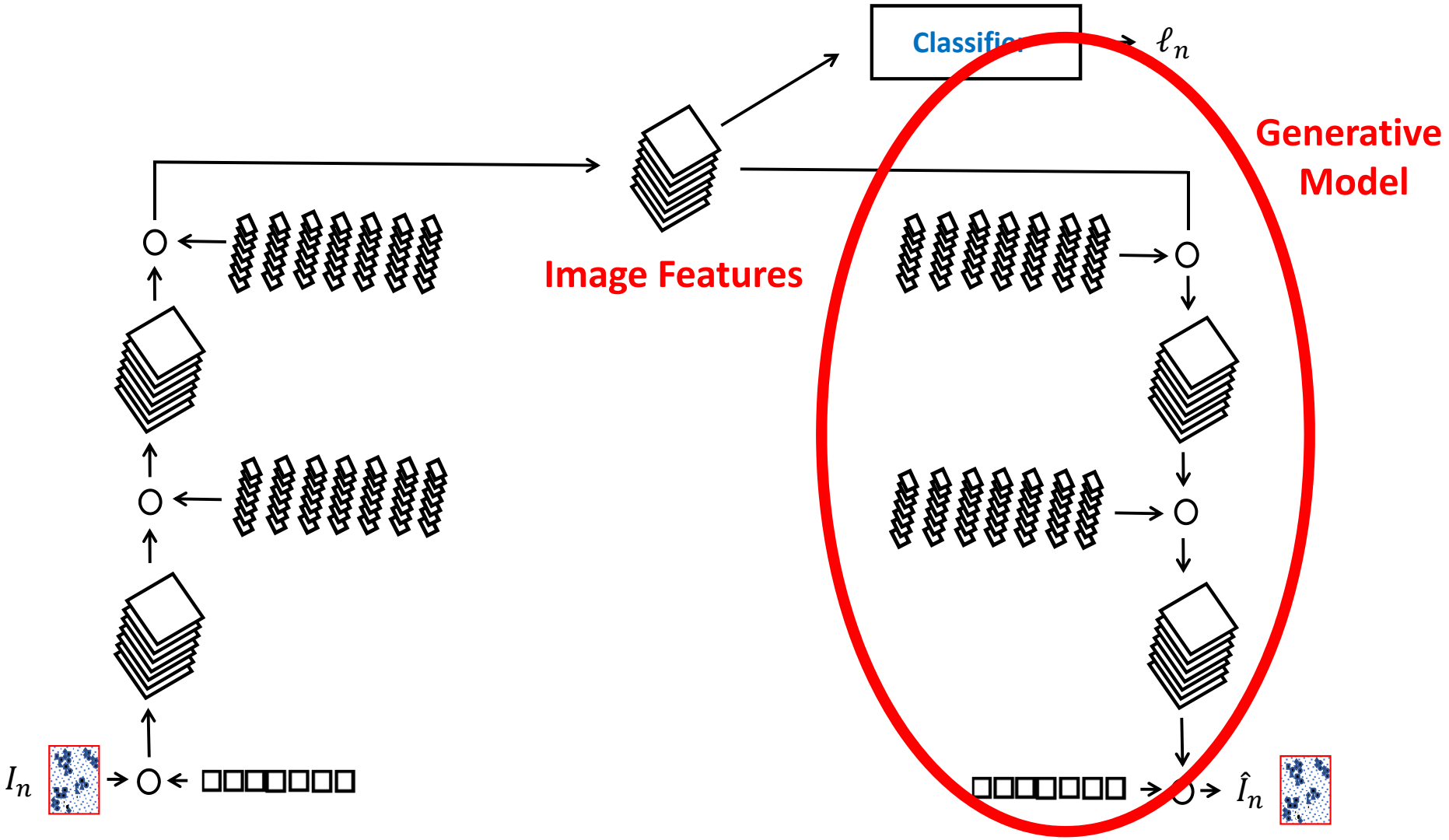
Deep Analysis Architecture



Deep Analysis Architecture

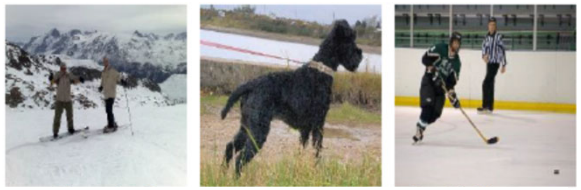


Deep Analysis Architecture

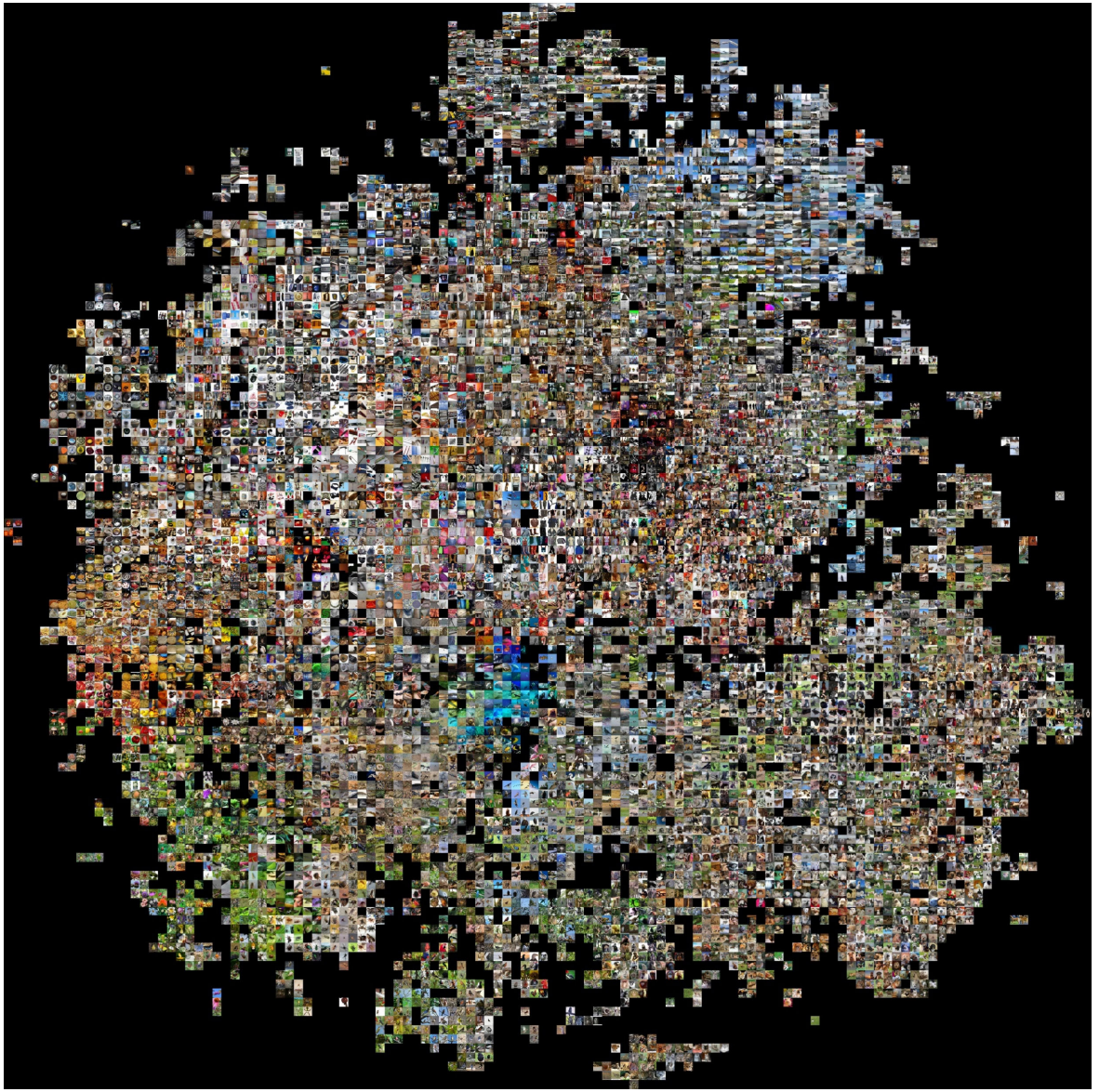


ImageNet Challenge

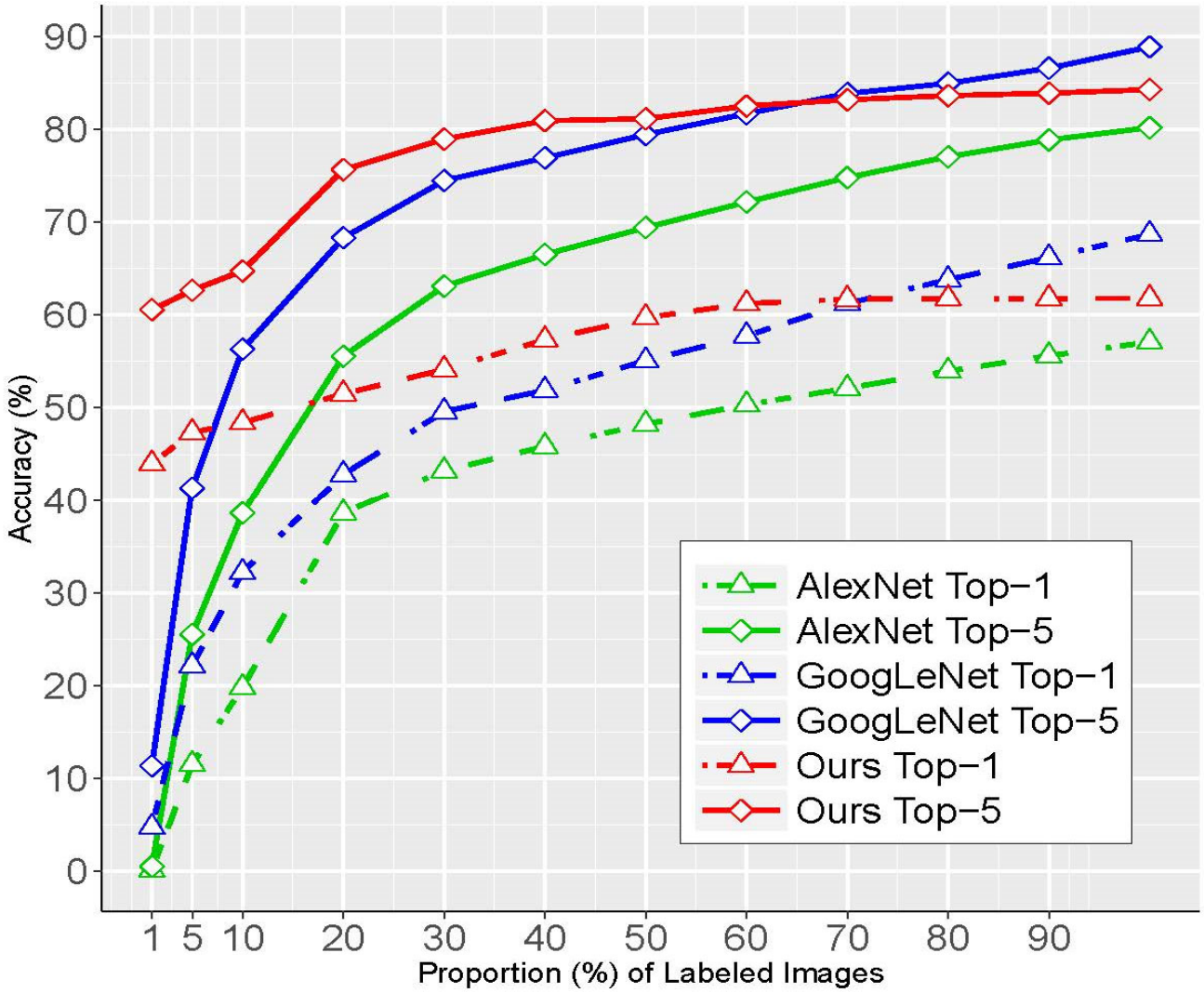
- 1000 image classes
- 1000 training images per class
- 1 million training images
- Real-life RGB images



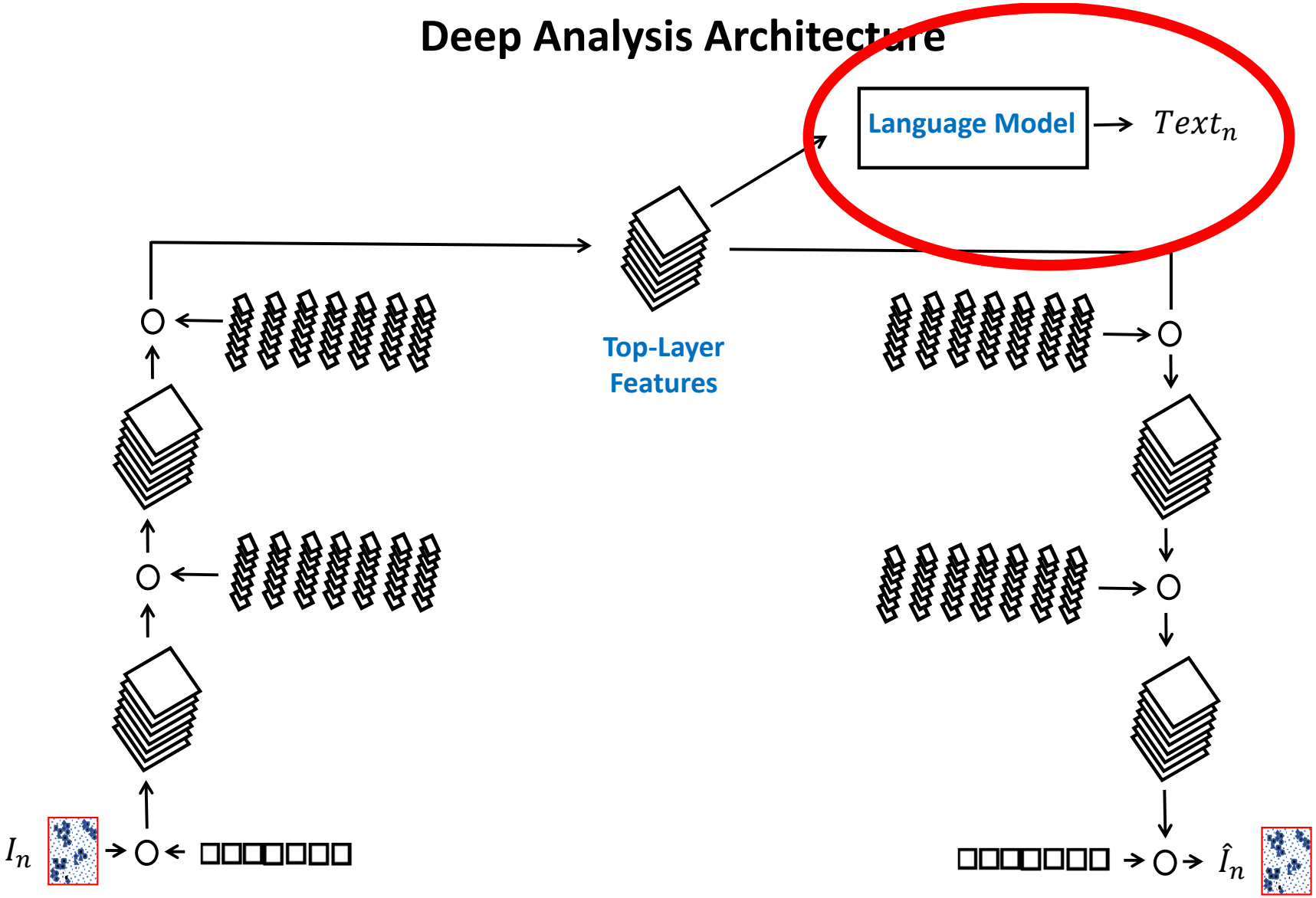
Example Images



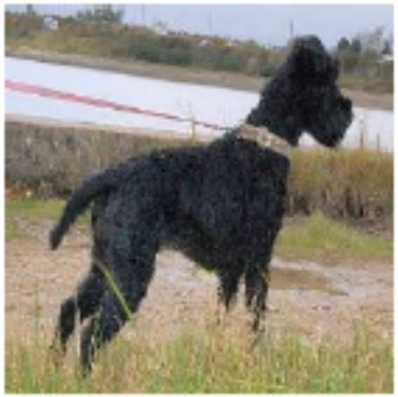
ImageNet Challenge



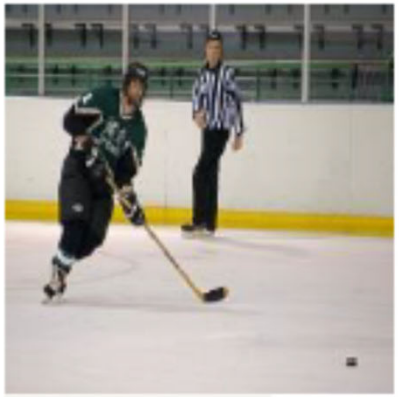
Deep Analysis Architecture



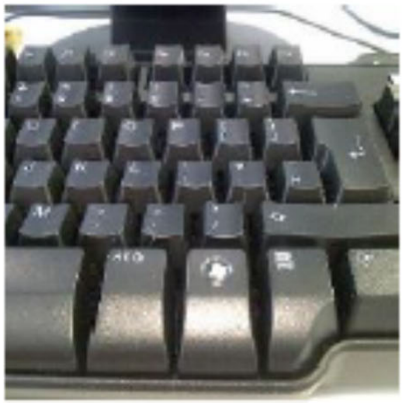
a man with a snowboard next to a man with glasses



a big black dog standing on the grass



a player is holding a hockey stick



a desk with a keyboard

a man is standing next to a brown horse

a box full of apples and oranges