CS 461: Machine Learning Principles

Class 17: Oct. 31

Deep CNN Architecture

AlexNet

Instructor: Diana Kim

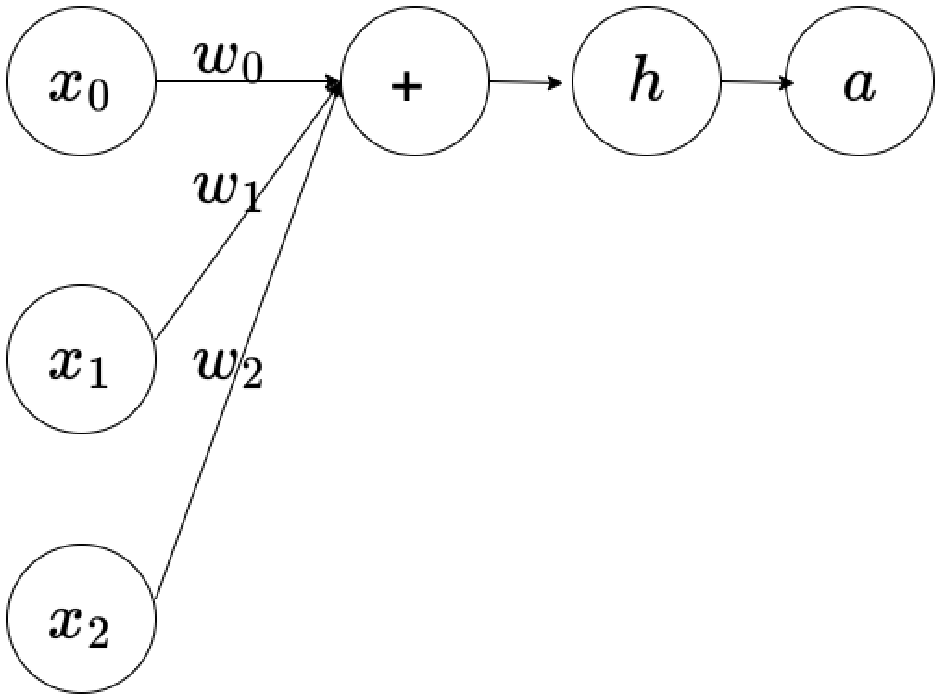
In the last class,
we learned that

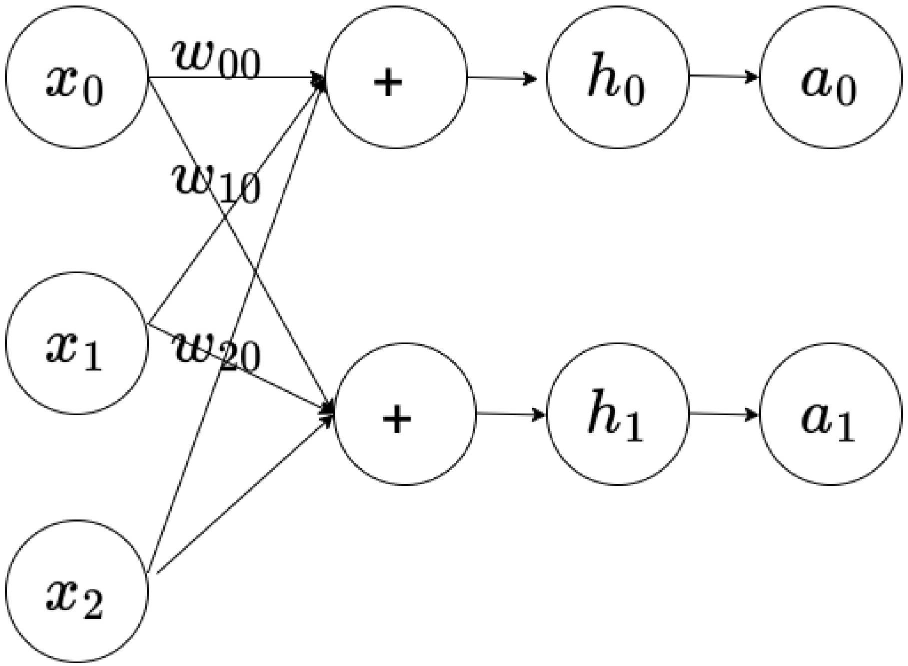
- DNN is a large composite function, consisting of many layers
- DNN non-convexity
- Backpropagation algorithm to compute gradients.

Today, we will study DNN architecture.

- Connectivity of Units and Its Internal Operations
- The Design Philosophy
- Possible problems as training a DNN

The Functional Units of Neural Net

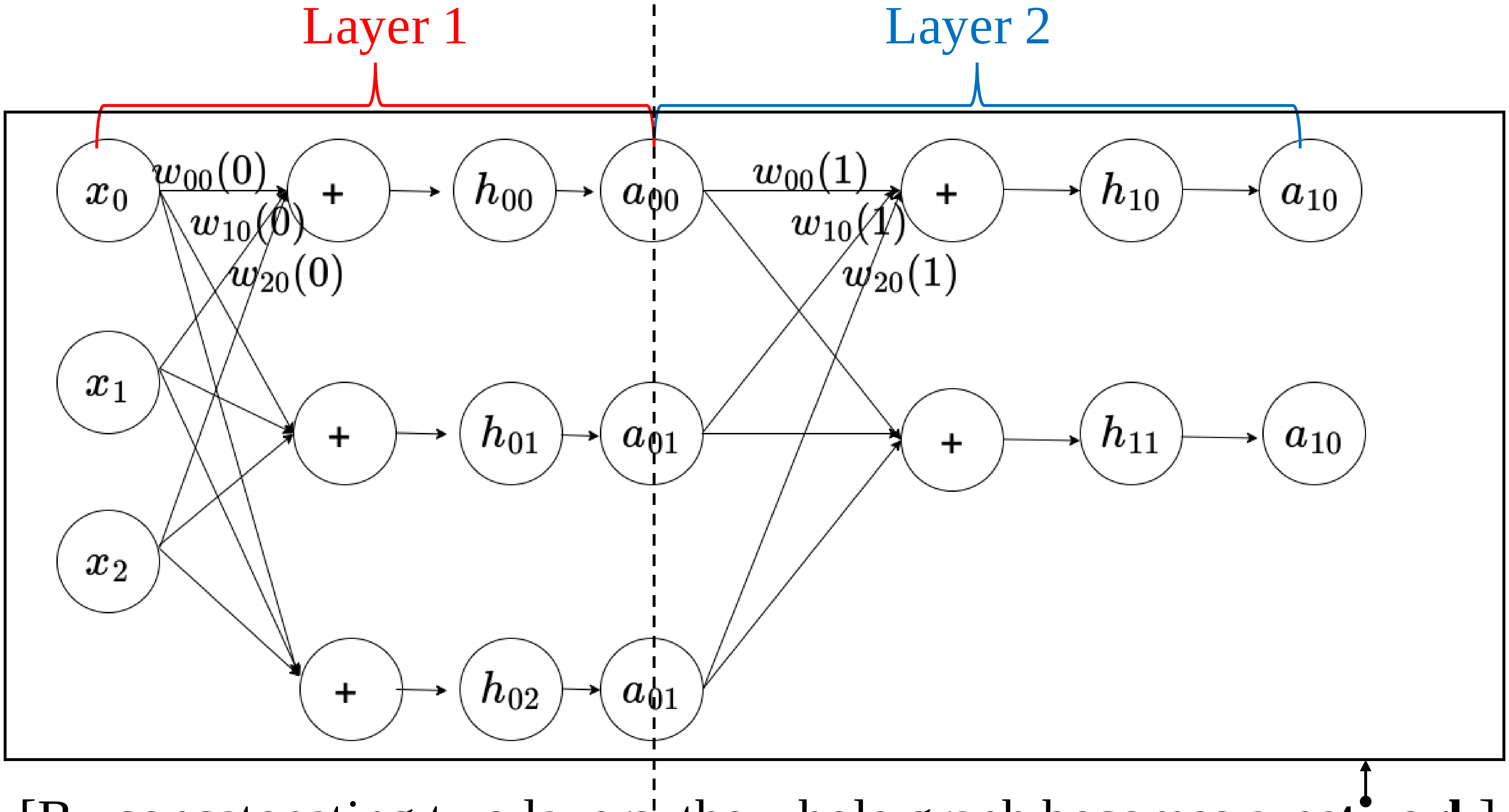




[A Single Unit] → [A single Layer]

two output units that share inputs.

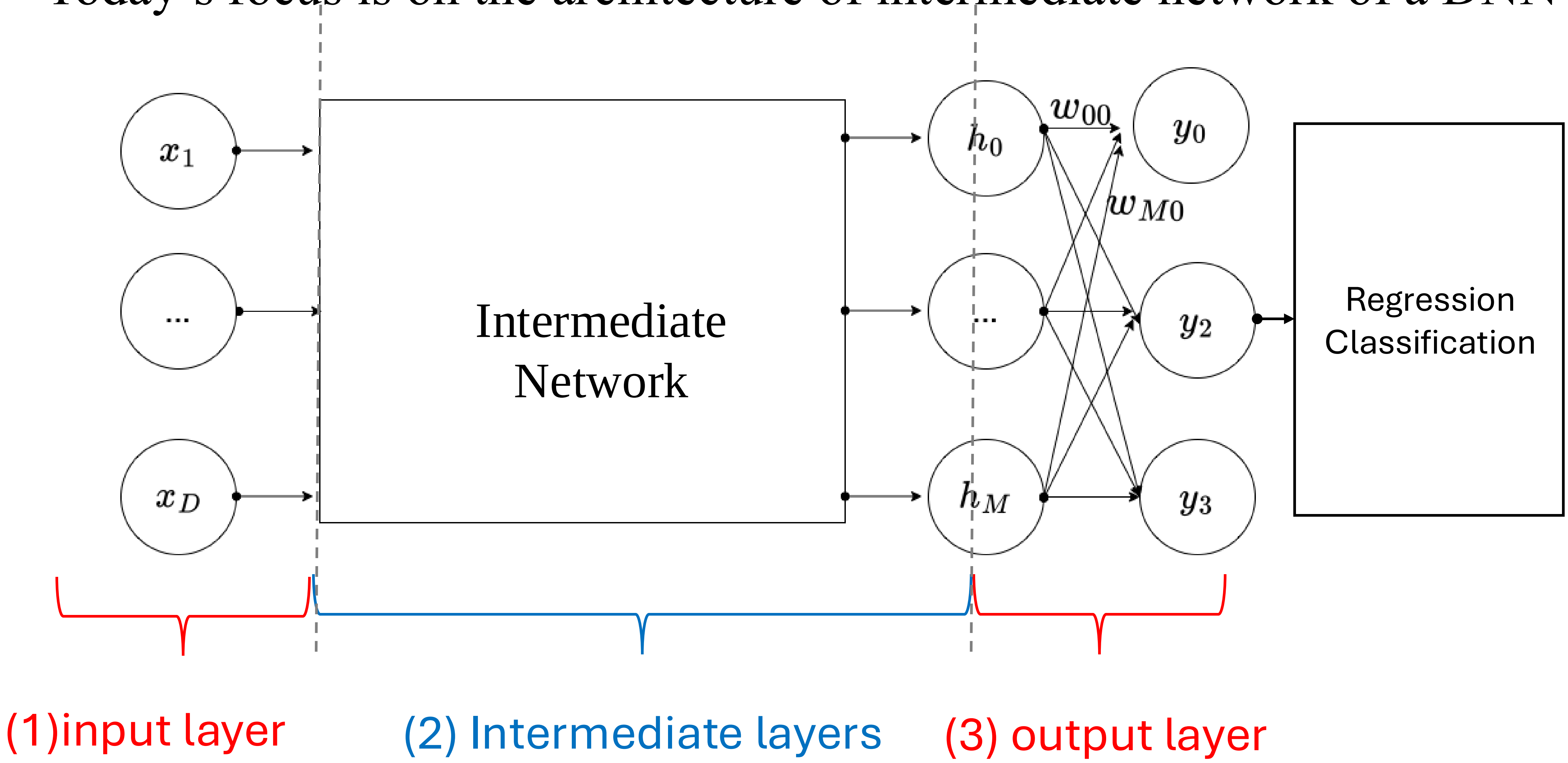
Deep Neural net is a large composite function



[By concatenating two layers, the whole graph becomes a **network**]

[Many Composite Layers: a **deep** network]

- Three Parts of DNN:
- Today's focus is on the architecture of intermediate network of a DNN



Analysis of AlexNet Architecture

- The first deep neural net that developed for ImageNet classification in 2012
- ImageNet(1.2 M, 1000 object classification)
- Developed by Alex Krizhevsky, Ilya Sutskever , and Geoffrey E. Hinton

By the 1980s, however, researcher had developed algorithms for modifying neural nets' weights and threshold that were efficient enough for networks with more than one layer, removing many of the limitations addressed in the one layer perceptron.

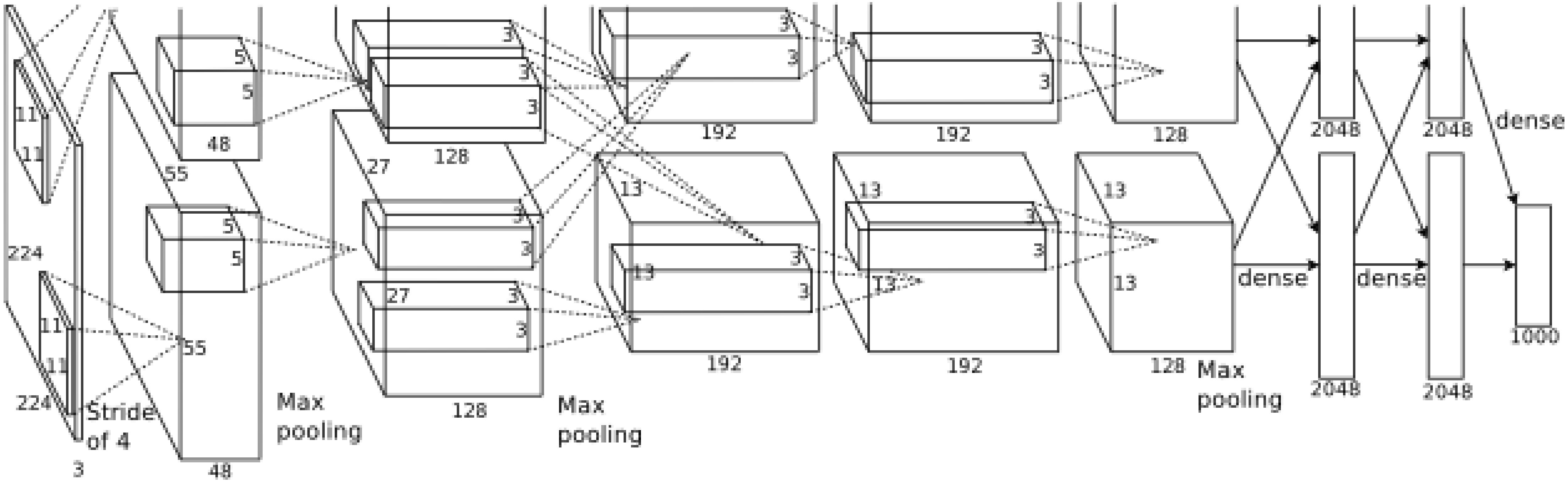
(https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414)

- **Neocognitron** (Fukushima 1980) : very similar to the architecture of modern deep neural net, alternating convolutional and down sampling layers. However, it did not use backpropagation.
- **LetNet Series** (Yann LeCun 1989 – 1998): Neocognitron like network and backpropagation is applied.

(https://www.rctn.org/bruno/public/papers/Fukushima1980.pdf)

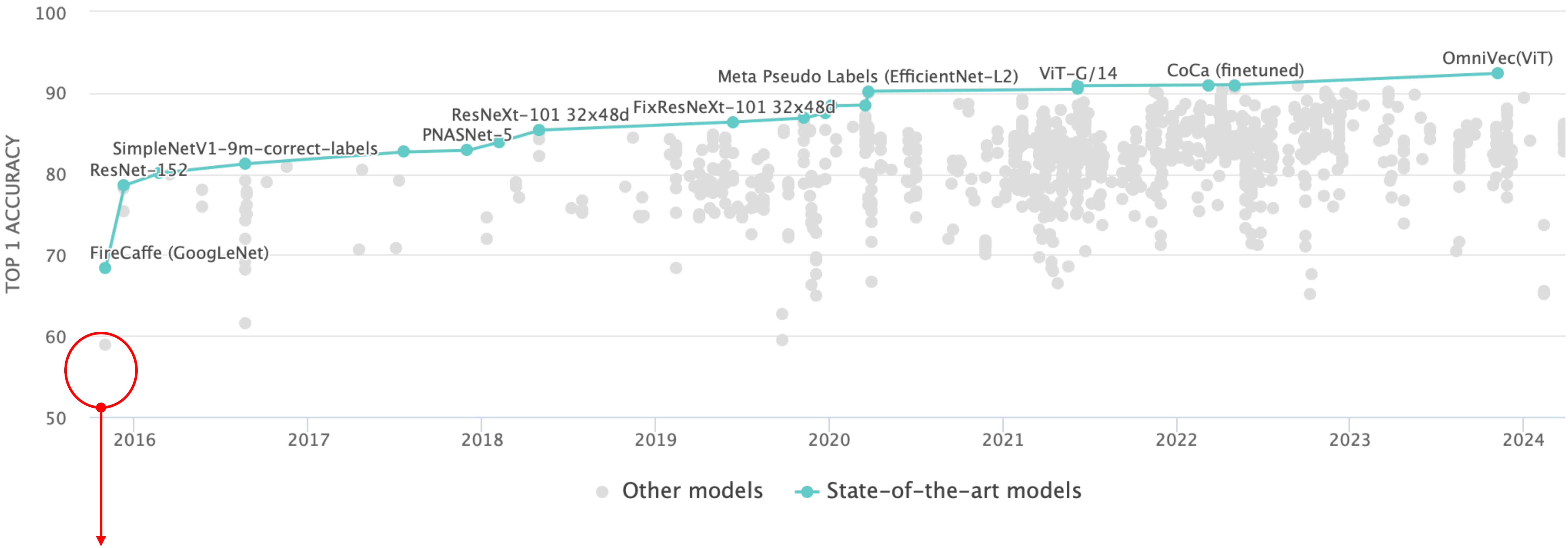
(http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf)

The Architecture of AlexNet



From the original paper:
https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

AlexNet is the first deep neural implemented for ImageNet.
The previous attempts were not based on deep learning.
For example) the paper "ImageNet: A Large-Scale Hierarchical Image Database"



AlexNet (Top 1 Accuracy: 58.9%)
(paper report: 62.5%)

Feature Layers of AlexNet

- Convolutional block
- Activation
- Max pooling

[Pretrained AlexNet Pytorch]

```
Downloading: "https://github.com/pytorch/vision/zipball/v0.10.0" to /Users/dianakim/.cache/torch/hub/v0.10.0.zip
Downloading: "https://download.pytorch.org/models/alexnet-owt-7be5be79.pth" to /Users/dianakim/.cache/torch/hub/c
100%|
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

Repeat!

[Features]

[Classifier]

[1] Convolution Block (Filter Bank)

- Convolutional Layers:

In Computer Vision,
Linear filtering is a technique
that uses a filter to modify an image's signal frequency spectrum.

[Convolution Operation]

$$\underbrace{C(i,j)}_{\text{Feature Map}} = \sum_m \sum_n \overbrace{I(i+m, j+n)}^{\text{image}} \underbrace{K(m.n)}_{\text{Kernel / Filter}}$$

Figure 3.14 from Computer Vision, Algorithms and Applications by Richard Szeliski

$$\frac{1}{K^2}\begin{array}{|c|c|c|c|}\hline 1 & 1 & \cdots & 1 \\ \hline 1 & 1 & \cdots & 1 \\ \hline \vdots & \vdots & 1 & \vdots \\ \hline 1 & 1 & \cdots & 1 \\ \hline\end{array} \quad \frac{1}{16}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline\end{array} \quad \frac{1}{256}\begin{array}{|c|c|c|c|c|}\hline 1 & 4 & 6 & 4 & 1 \\ \hline 2 & 8 & 12 & 8 & 2 \\ \hline 6 & 24 & 36 & 24 & 6 \\ \hline 2 & 8 & 12 & 8 & 2 \\ \hline 1 & 4 & 6 & 4 & 1 \\ \hline\end{array} \quad \frac{1}{6}\begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline\end{array} \quad \frac{1}{4}\begin{array}{|c|c|c|}\hline 1 & -2 & 1 \\ \hline -2 & 4 & -2 \\ \hline 1 & -2 & 1 \\ \hline\end{array}$$

$$\frac{1}{K}\begin{array}{|c|c|c|c|}\hline 1 & 1 & \cdots & 1 \\ \hline\end{array} \quad \frac{1}{4}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline\end{array} \quad \frac{1}{16}\begin{array}{|c|c|c|c|c|}\hline 1 & 4 & 6 & 4 & 1 \\ \hline\end{array} \quad \frac{1}{2}\begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline\end{array} \quad \frac{1}{2}\begin{array}{|c|c|c|}\hline 1 & -2 & 1 \\ \hline\end{array}$$

Gaussian : Low Pass Filter

Laplacian : Edge Detector

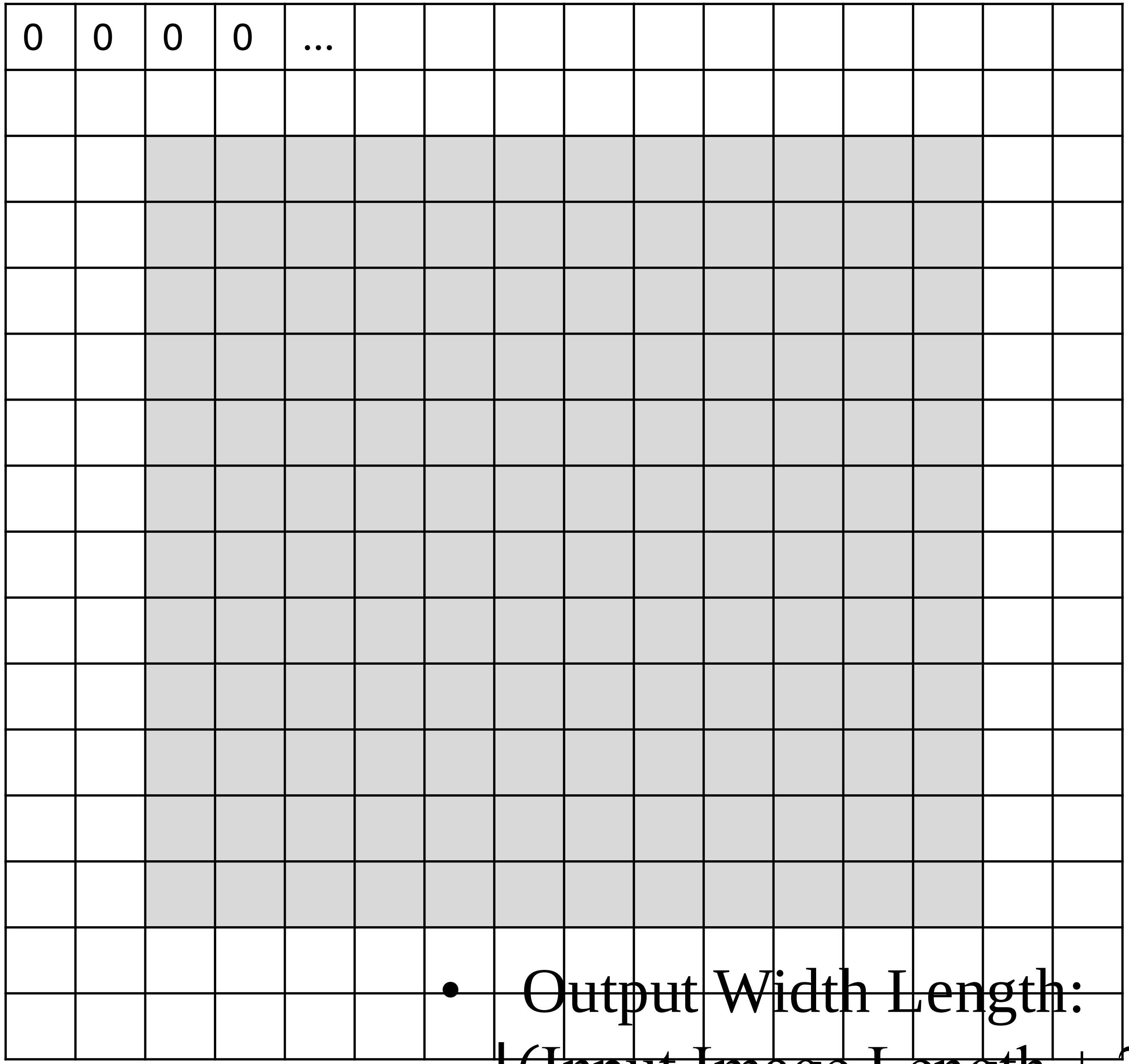
(a) box, $k = 5$ (b) bilinear (c) "Gaussian" (d) Sobel (e) "Laplacian"

Linear Filters: Extract features/ by using collection of pixel values in the vicinity of a given pixel to determine its final output value.

Convolution block aims to learn those filters. (Learnable Filters)
What filters would be necessary to perform a targeted task like 1000 object recognition?

- The first convolution block of AlexNet:

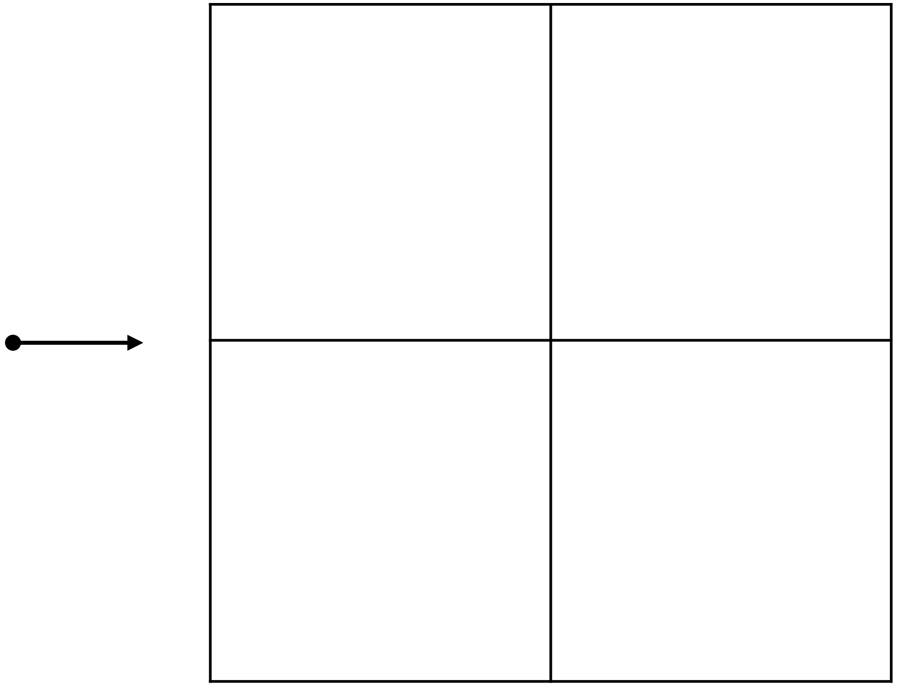
Conv2d (3, 64, Kernel Size=(11,11), Stride=4, padding=(2,2)) input image (16 ×16)



Convolution

- filter size: (11,11)
- stride = (4,4)
- padding=(2,2)

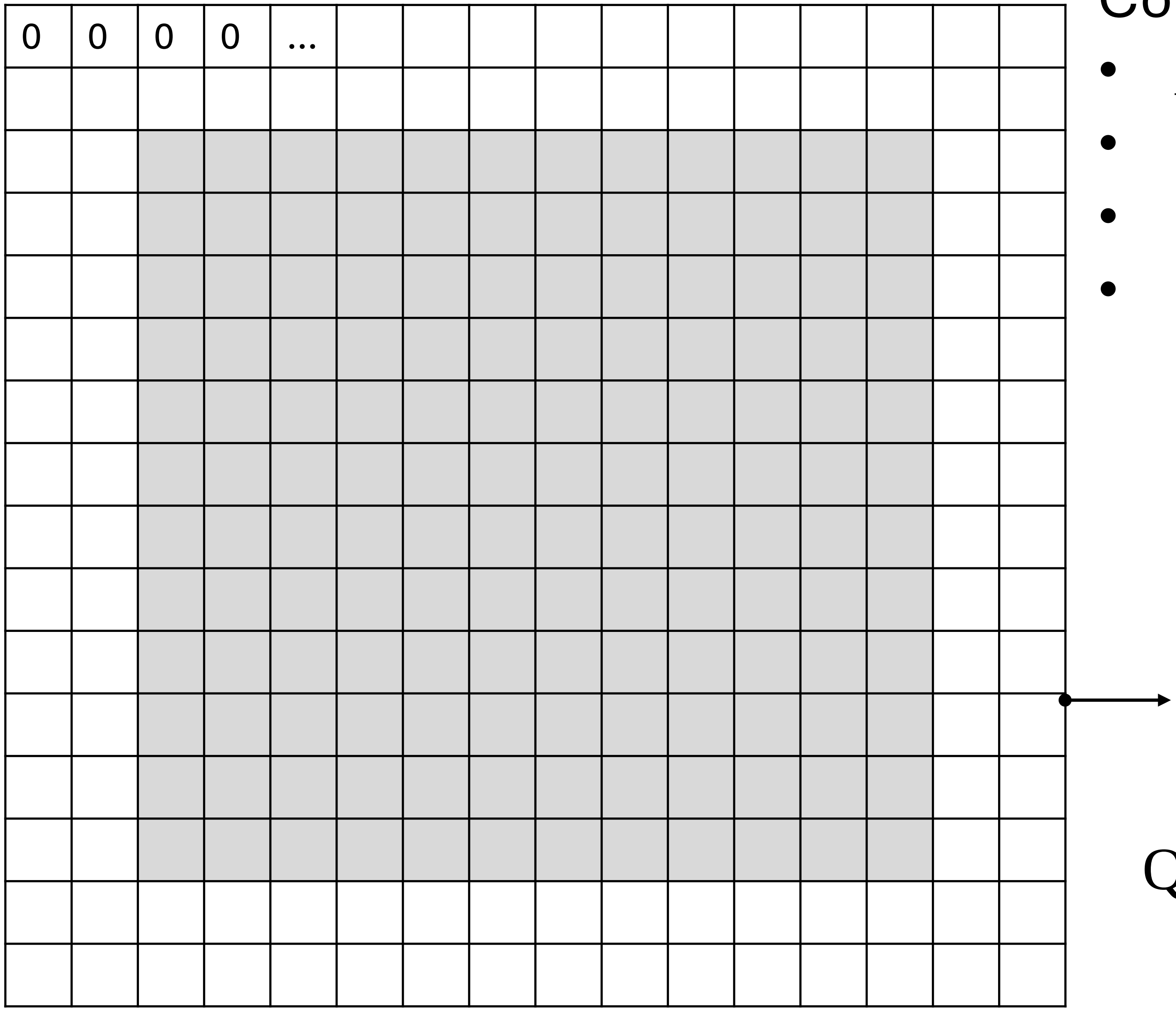
[output feature map/plane]



- Output Width Length:

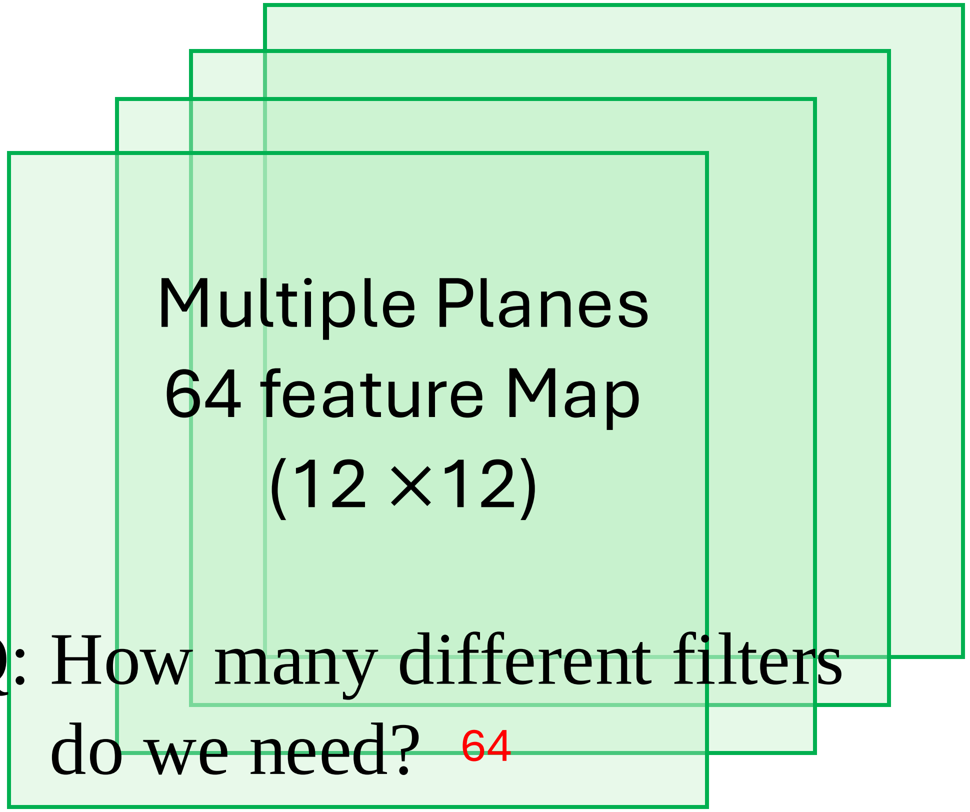
$\lfloor$(Input Image Length + 2 × padding−Filter size) / stride + 1$\rfloor$

input image



Convolution 2D

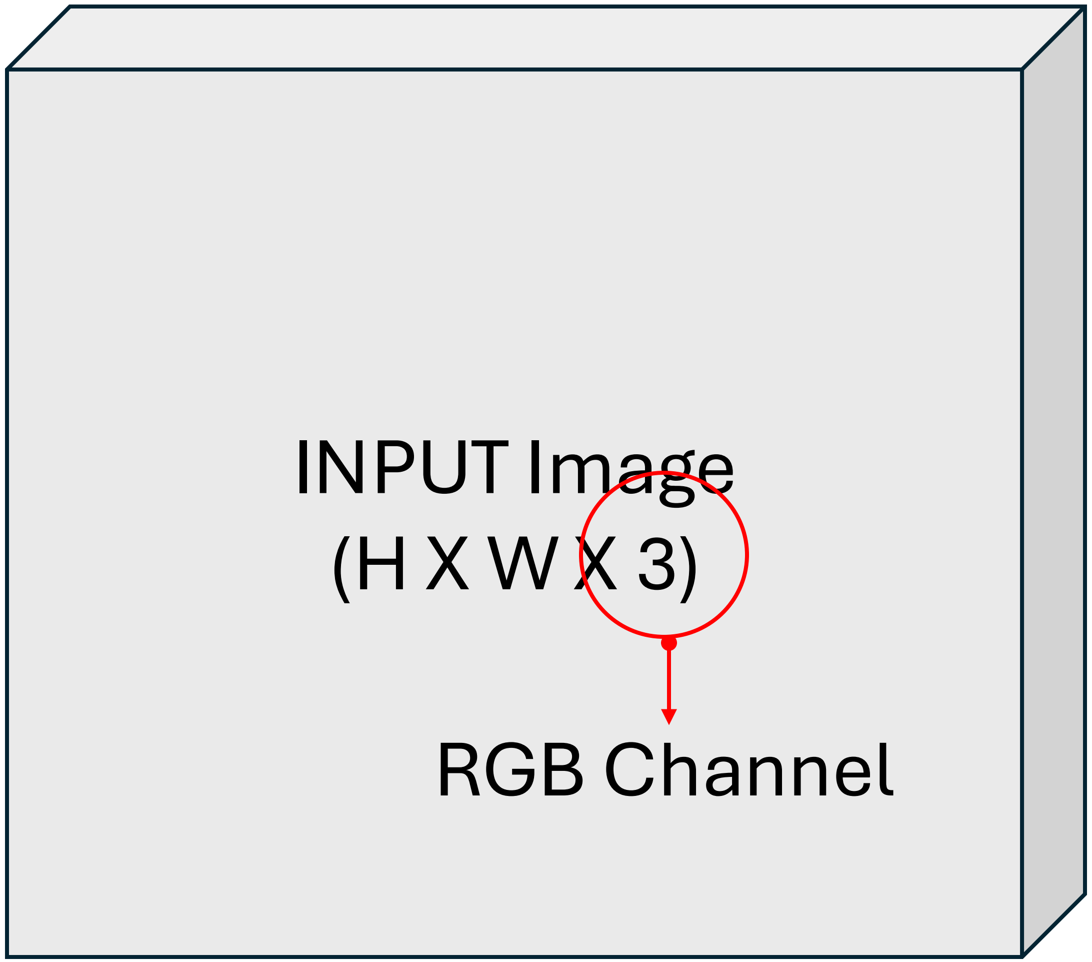
- Output Channel: 64
- Filter size: (11,11)
- Stride = (4,4)
- padding=(2,2)

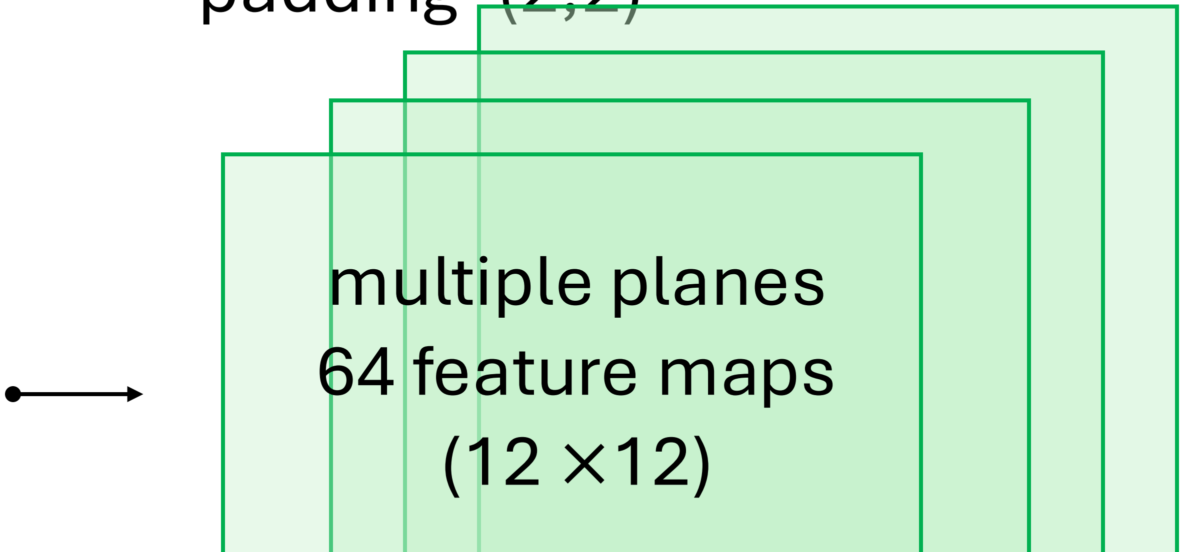


Q: How many different filters do we need? 64

Convolution 2D

- <u>Input Channel: 3</u>
- <u>Output Channel: 64</u>
- Filter size: (11,11)
- Stride = (4,4)
- padding=(2,2)



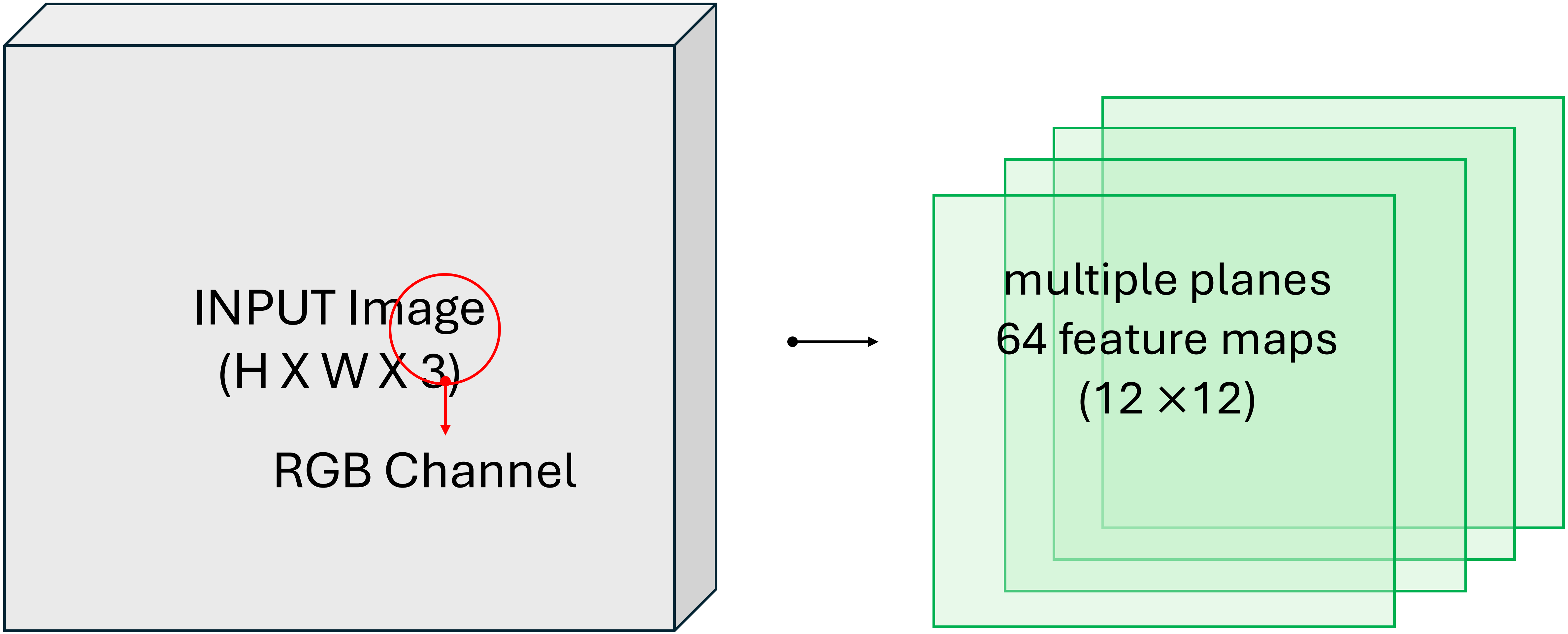


Q: What is the filter dimension? 3 x 11 x 11

Q: How many different filters do we need? 64

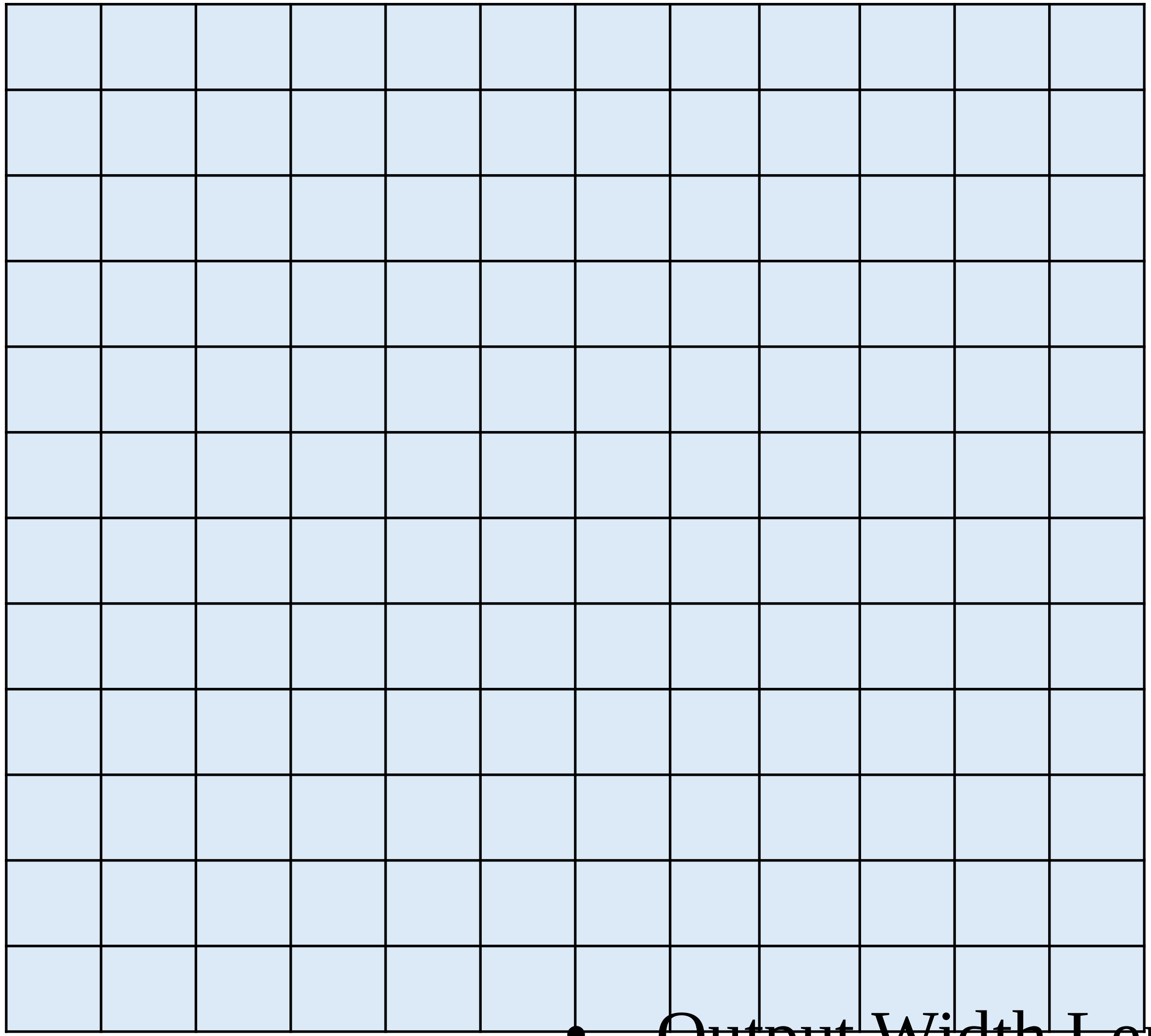
Q: How the **bias term** incorporated into the filter?

How many bias terms are in a conv2 block? 64



[2] Max Pooling Block
promotes the invariance to
small translations/ scaling / rotation of
input images

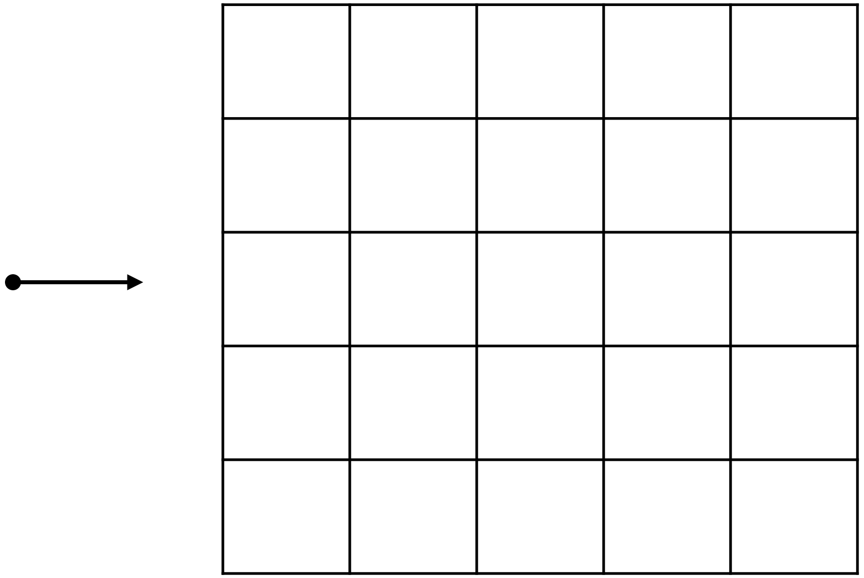
Feature input



Max Pooling (down sampling)

- Filter size: (3)
- Stride = (2)

max pooling output



- Output Width Length:
  ⌊Input Image Length + 2 × padding − Filter size)/stride + 1

What is the philosophy underlying the architecture?
In the feature layer, why do we need
the convolutional blocks and max pooling layers?

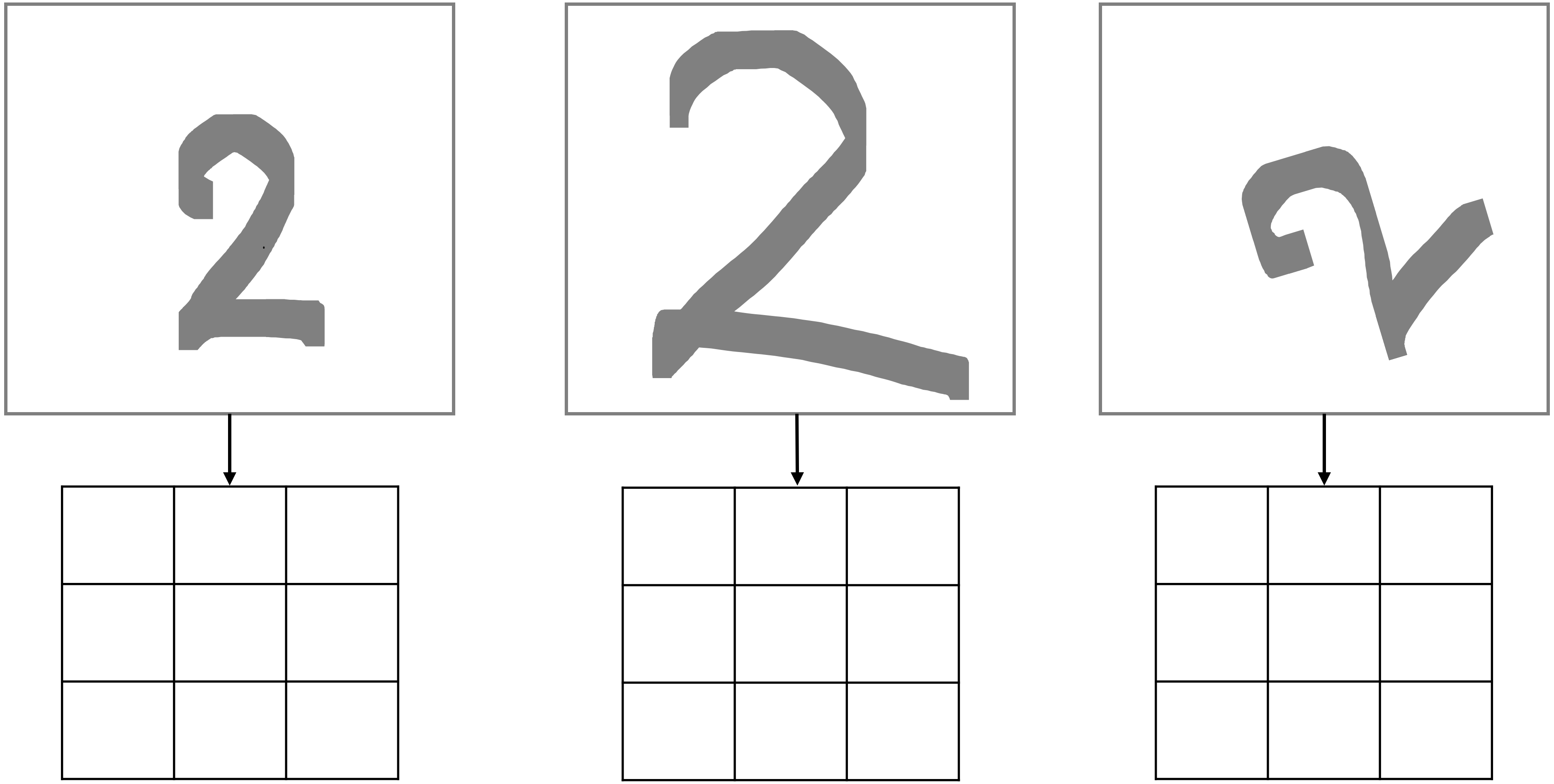
+ local receptive field and parameter sharing enhances equivariance.
+ max pooing enhances invariance.

In the feature layer, why do we need
the convolutional blocks and max pooling layers?
The constrained architecture helps to learn the visual features
invariant to the small translations/ scaling / rotation of input images.

Q: Then, why this invariance property is desirable?

+ invariance enhances generalization.

Data images have variations in terms of location, scales, and rotation.



Q: what will be a desirable feature map for the three images?

Equivariant vs. Invariant Feature Representation

- **Invariant** representation: (achieved by max pooling)
  the transformation of the input will result in the same output.

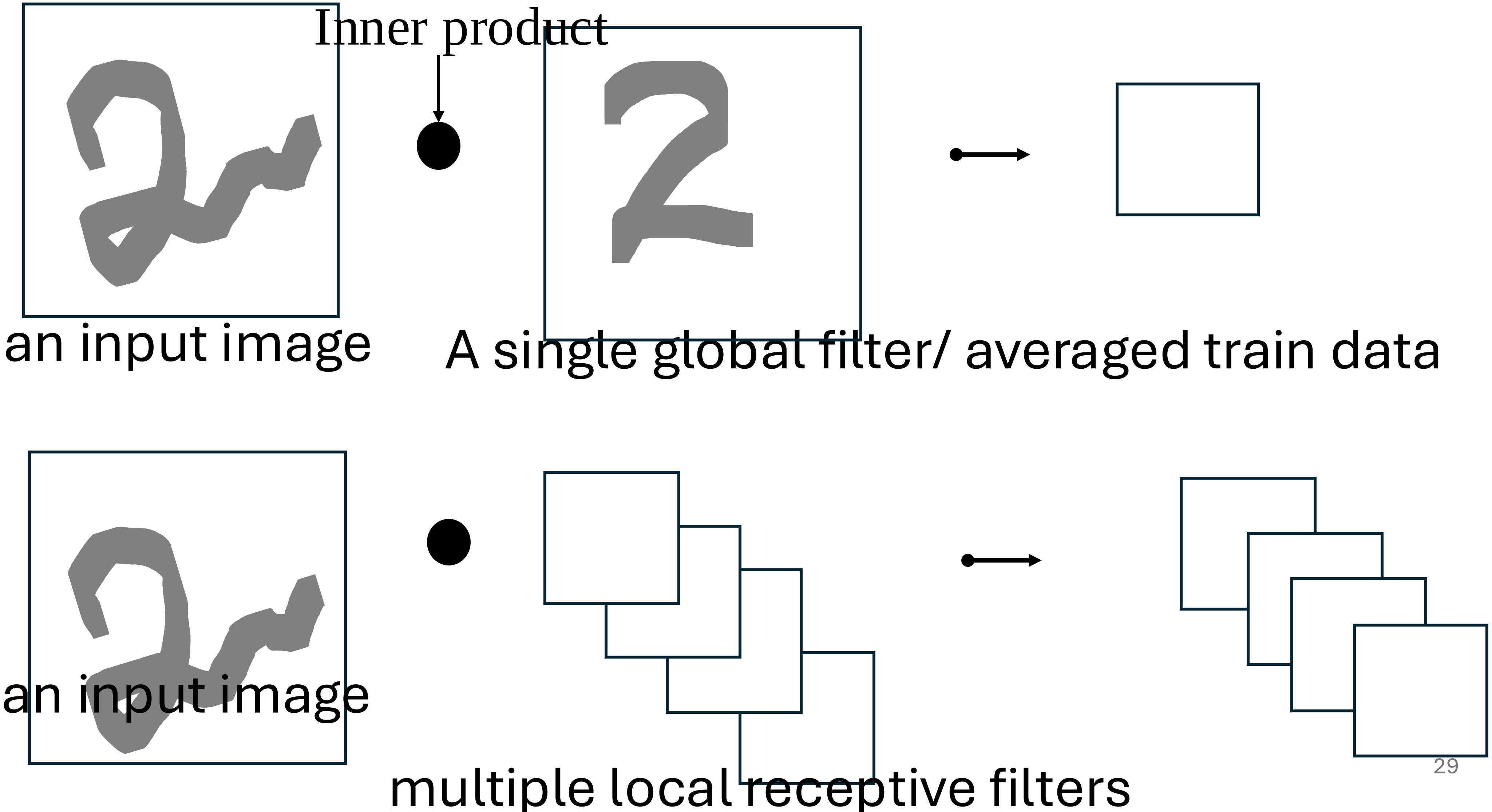
- **Equivariant** representation: (achieved by conv block)
  the transformation of the input
  will result in the same transformation in the feature space.

- **Equivariant** representation:
  the transformation of the input
  will result in the same transformation in the feature space.

Equivariance to geometric transformations in DNN improves

- parameter efficiency,
- data efficiency,
- and robustness to unseen data.

Q: Which one will be an ideal filter to improve generalization?

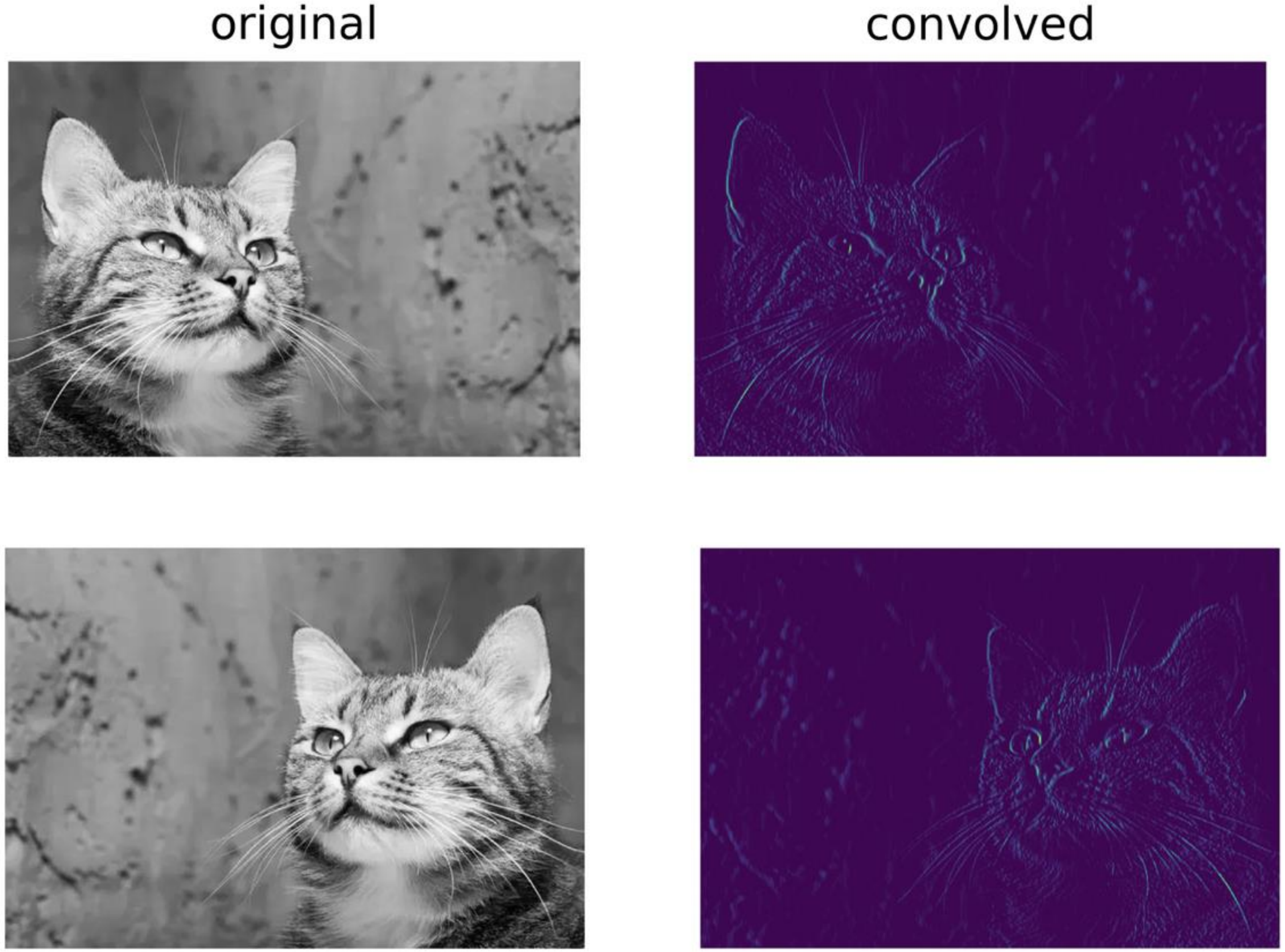


Equivariance Example:

https://blog.paperspace.com/pooling-and-translation-invariance-in-convolutional-neural-networks/

- When the image is convoluted with the Sober filter below.

-1	0	1
-1	0	1
-1	0	1



The Mechanisms of Convolution for Equivariance and Invariance.

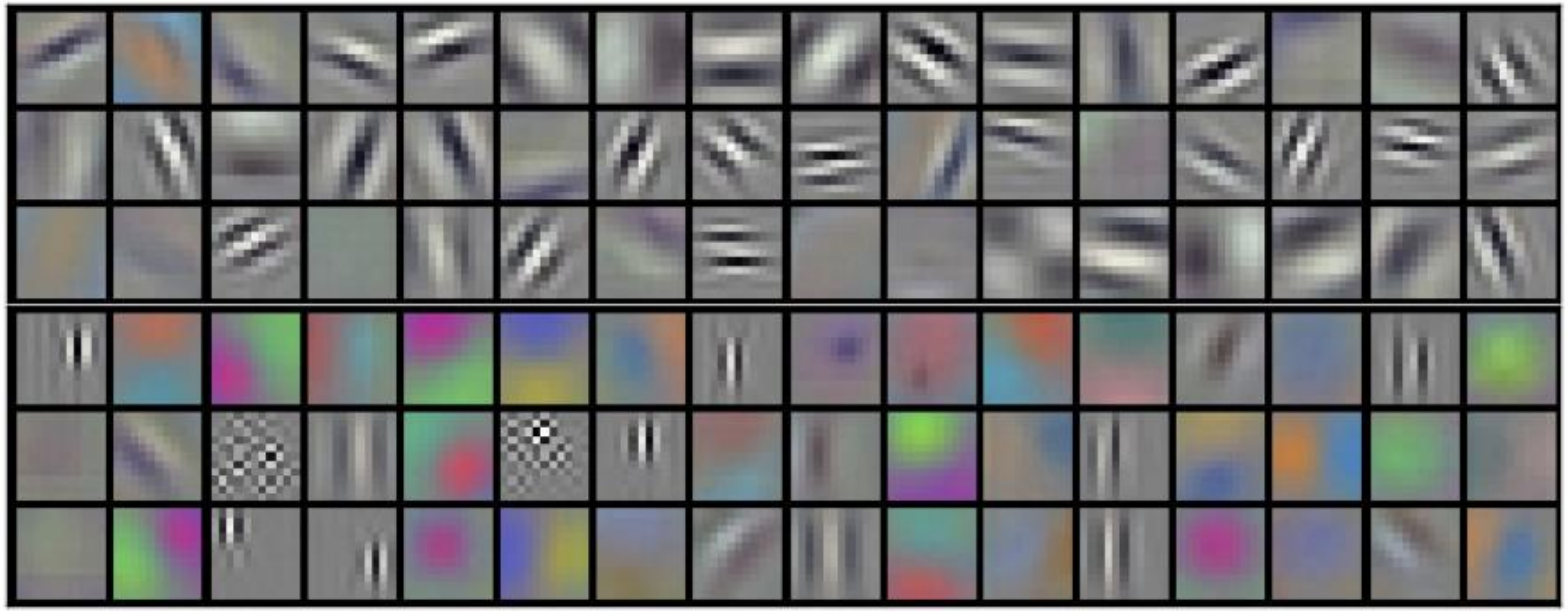
- Local Receptive Field (Sparse Interaction): Equivariance
- Parameter Sharing: Equivariance
- Subsampling: Invariances

Q: How can we achieve the ideal property without using convolution block?
What if we implement the DNN only using fully connected layers?

+ if we have a dataset includes all possible variation and a sufficiently large number of data samples, we can build a network that is robust to the geometric variations.

Visualization of Convolutional Filters

Visualization of the First Layer Convolutional Filters ($11 \times 11 \times 3$)



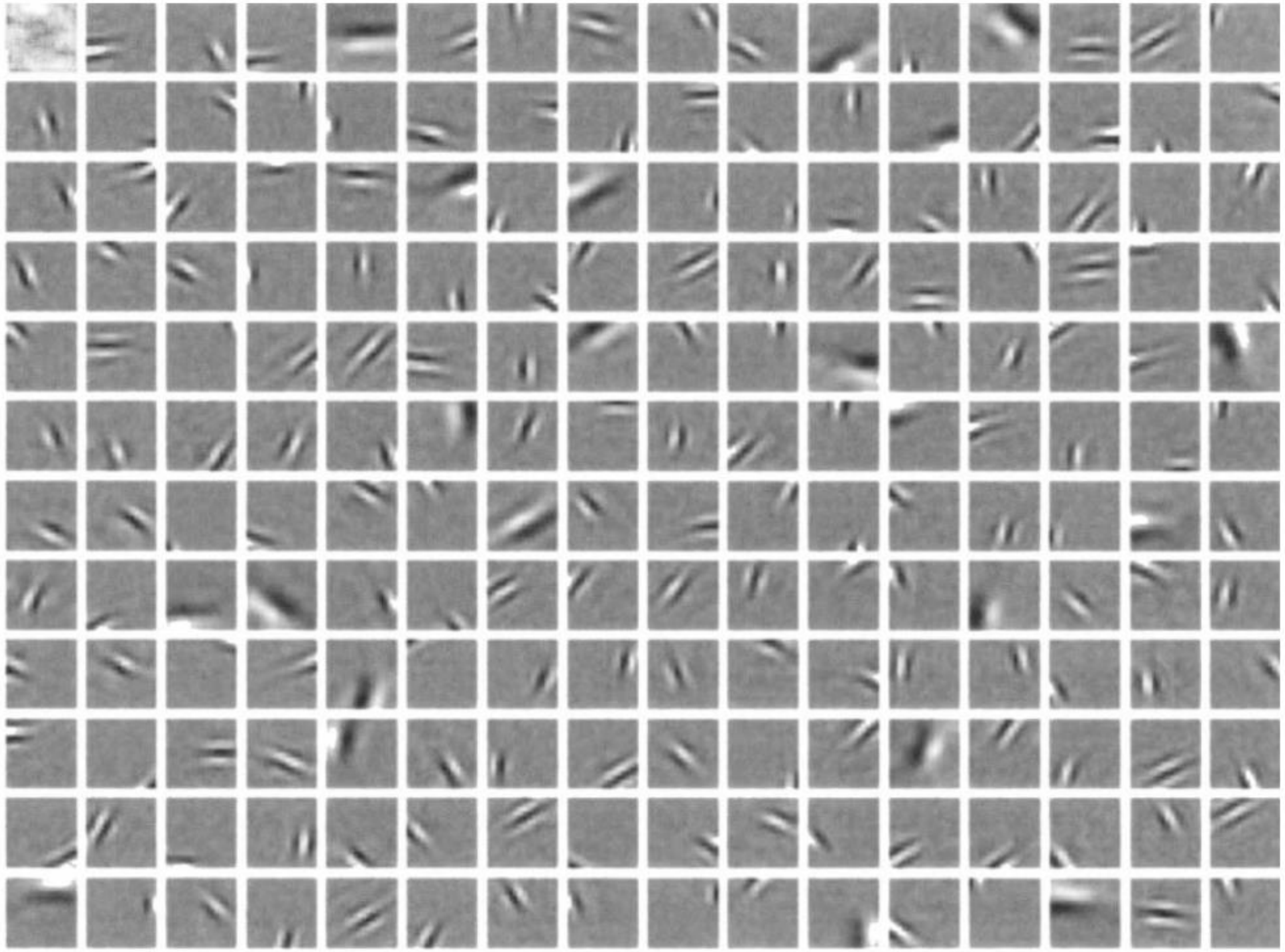
From the paper "ImageNet Classification with Deep Convolutional Neural Networks"

How can we visualize the Filters?

The first convolution filters have three channels.
The three channels can be thought of as RGB.

We can see that many Gabor-like Functions.

Some kernels synthesized by the network are remarkably similar to those found to exist in biological vision systems.

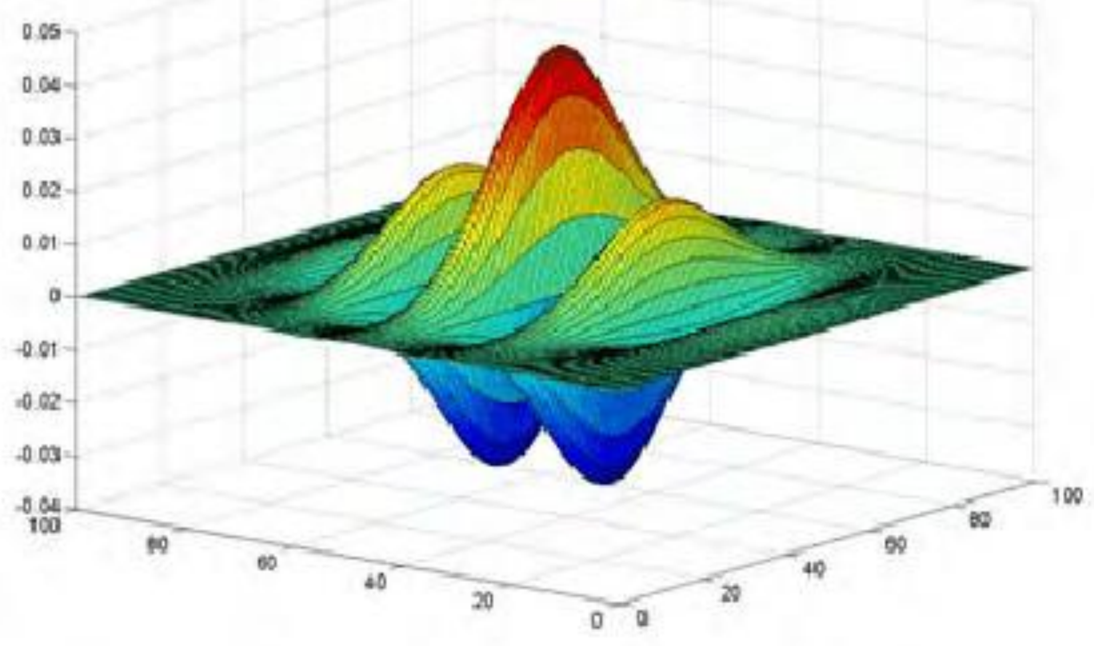


From the paper ``Emergence of simple-cell receptive field properties by learning a sparse code for natural Images" by Bruno A. Olshausen* & David J. Field.

Impulse function modeling between visual stimuli and brain impulses in primary visual cortex (V1).

Garbor Filter

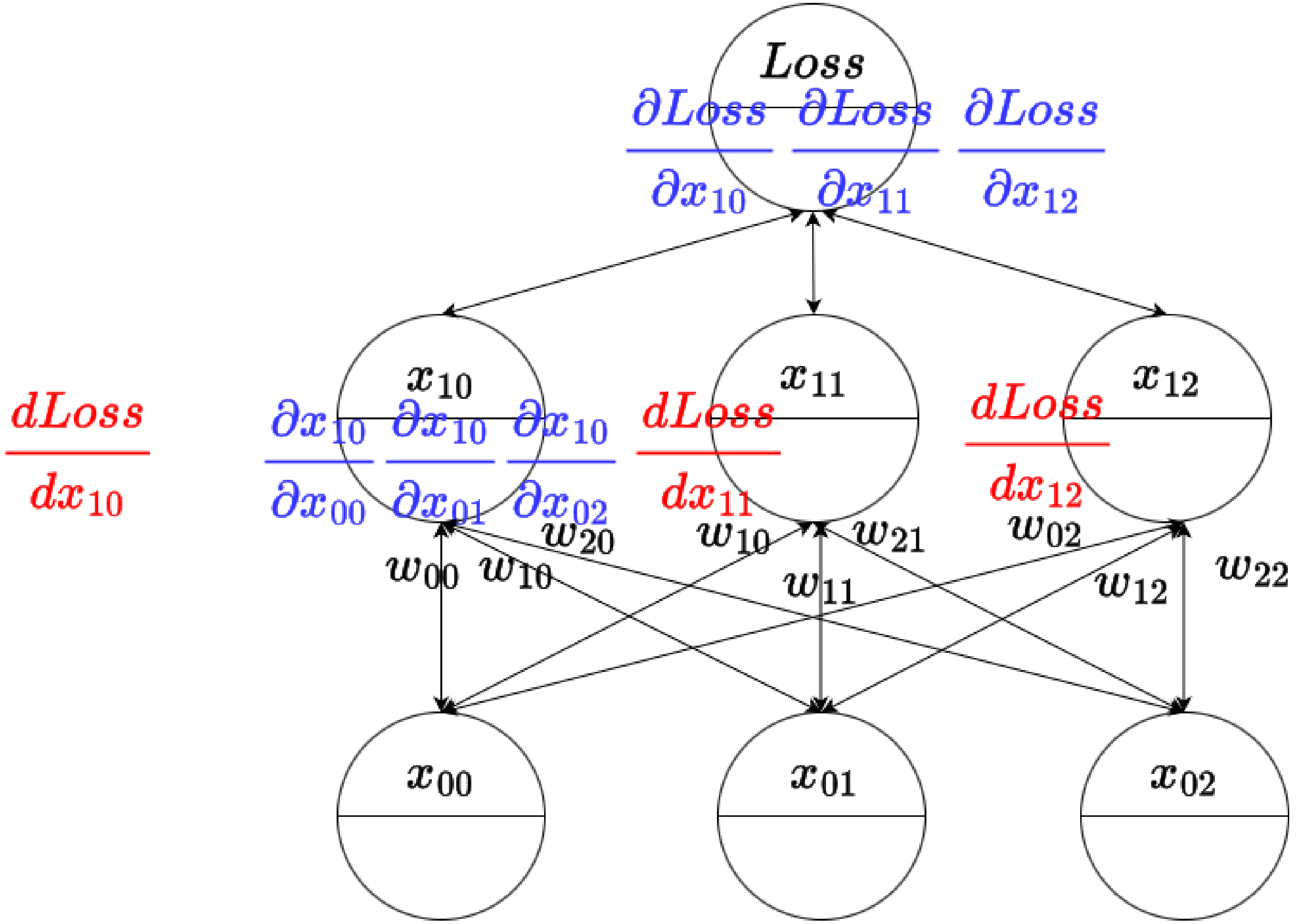
- different spatial frequencies
- different orientations in visual stimuli
- local receptive field.



Convolution net is the example of artificial visual systems replicates the biological vison.

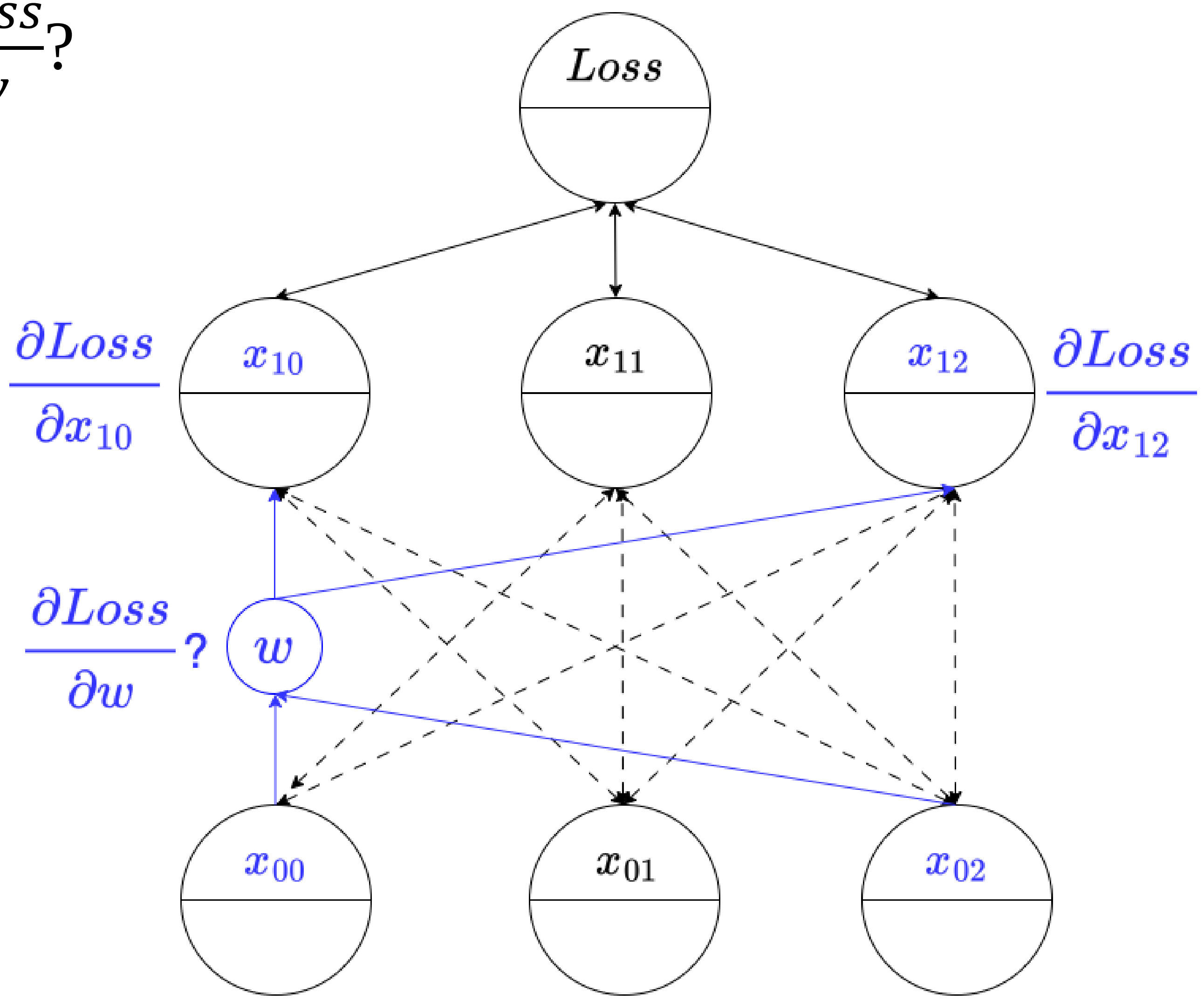
Backpropagation for Convolution Block

- no sharing case: in the last class



$$\frac{dLoss}{dx_{00}} = \frac{dx_{10}}{dx_{00}}\frac{\partial Loss}{\partial x_{10}} + \frac{dx_{11}}{dx_{00}}\frac{\partial Loss}{\partial x_{11}} + \frac{dx_{12}}{dx_{00}}\frac{\partial Loss}{\partial x_{12}}$$

- as w_{00} is shared?
- how can we compute $\frac{\partial Loss}{\partial w}$?



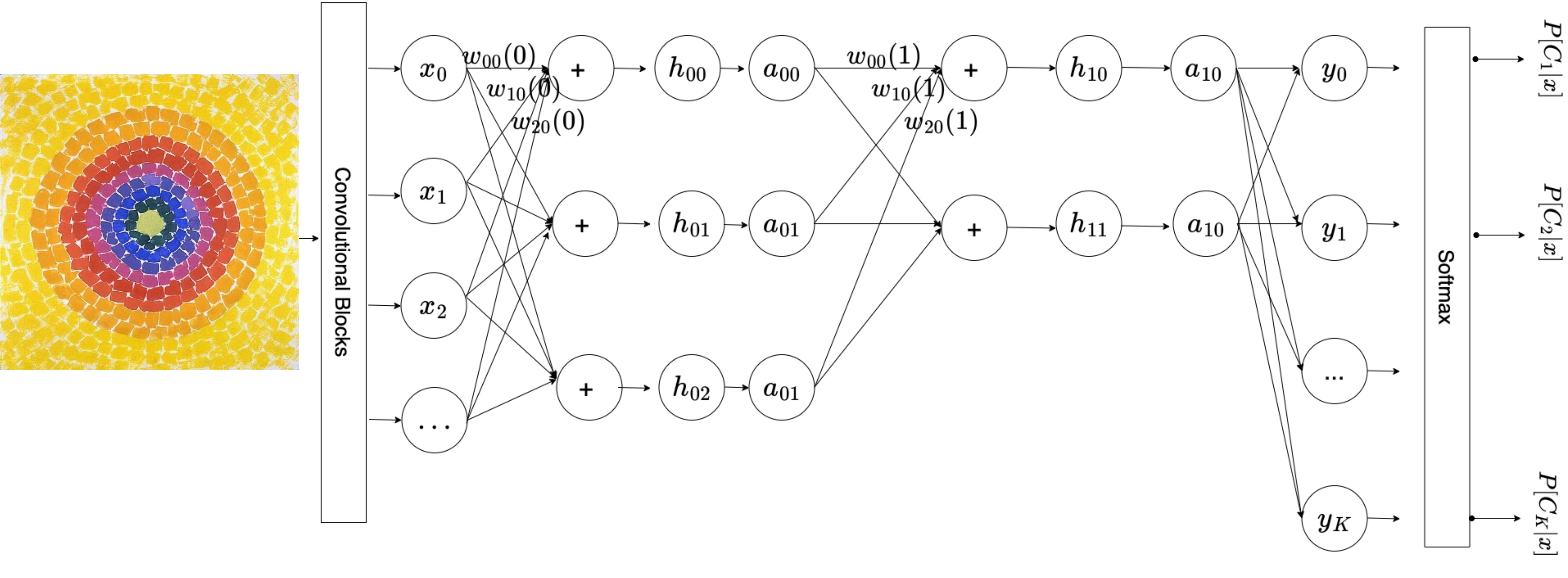
Classifier Layers of AlexNet

- Fully Connected layers
- Drop Out
- Activation

```
Downloading: "https://github.com/pytorch/vision/zipball/v0.10.0" to /Users/dianakim/.cache/torch/hub/v0.10.0.zip
Downloading: "https://download.pytorch.org/models/alexnet-owt-7be5be79.pth" to /Users/dianakim/.cache/torch/hub/c
100%|
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

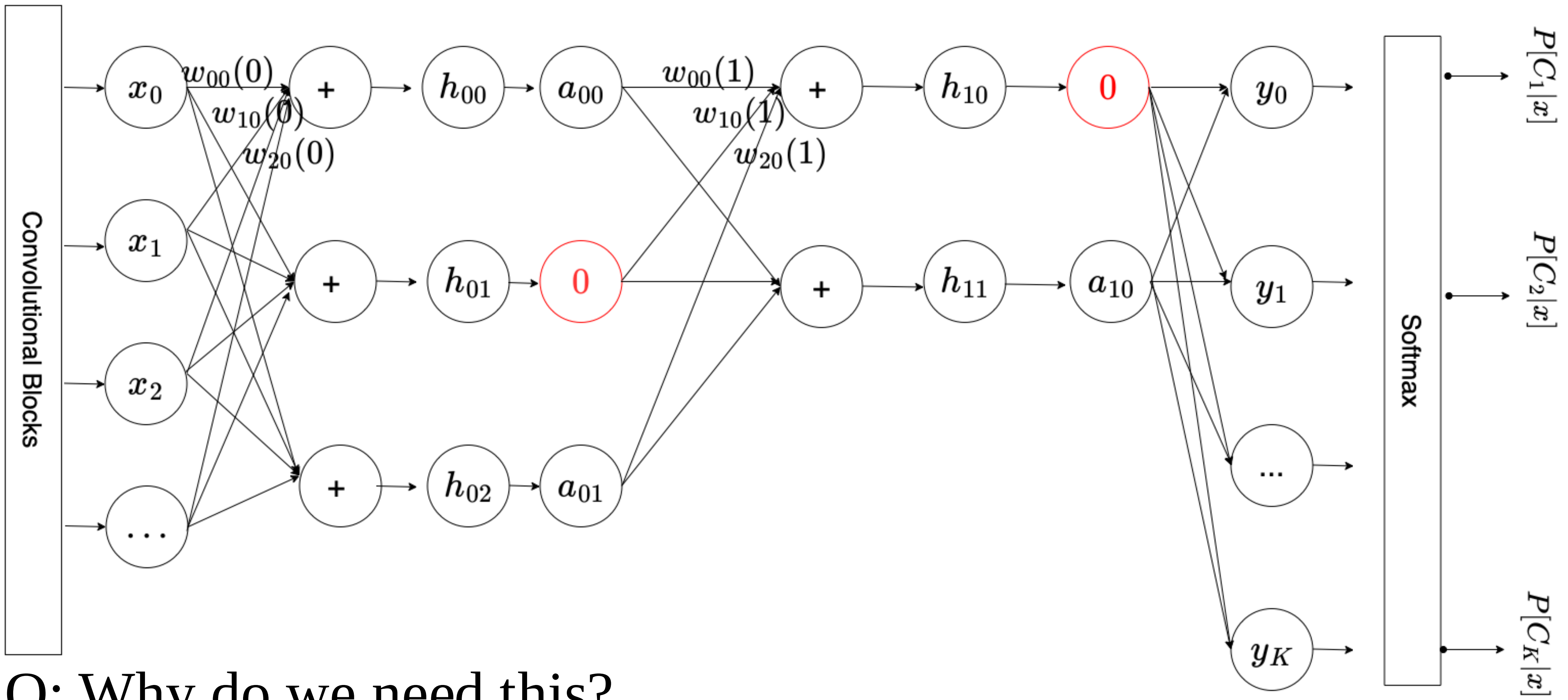
Pytorch Pretrained AlexNet

The last few layers of AlexNet
alternate between fully connected layers, activation layers, and dropout.



[1] Drop Out Layer (0.5)

: 50% of units are set as zero in training process.

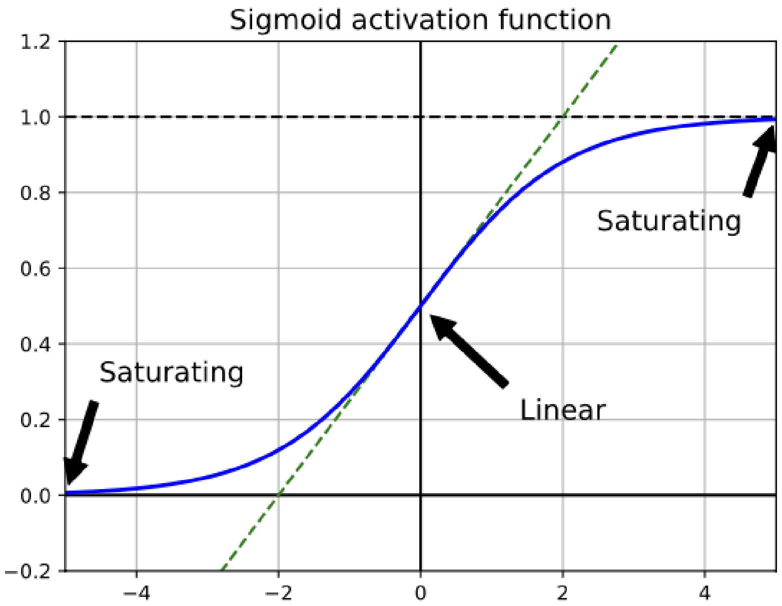


Q: Why do we need this?

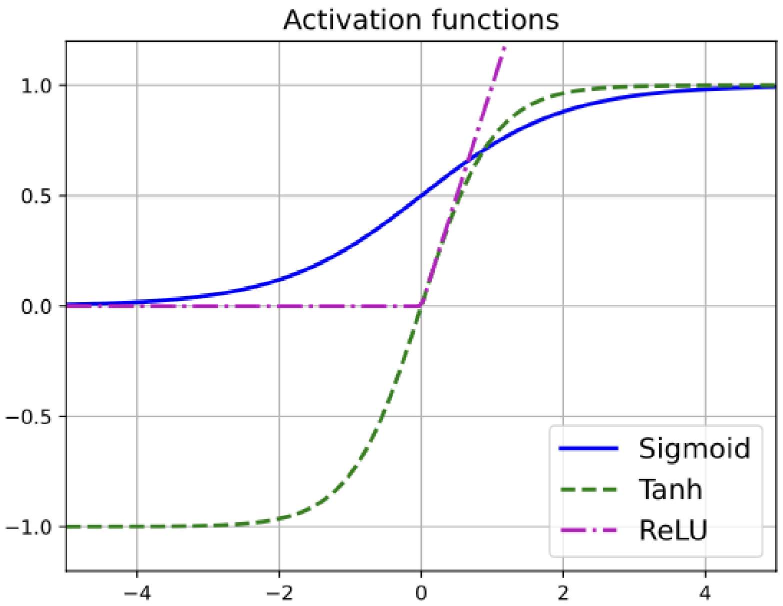
How drop out operation is different in training and inference stage?

[2] Activation Functions: gives non linearity to neural net

Fig 13.2 Textbook Murphy



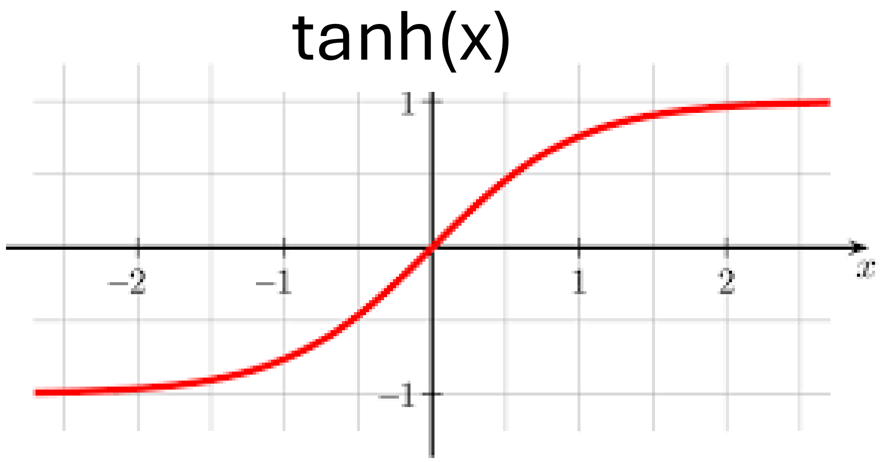
Sigmoid Activation

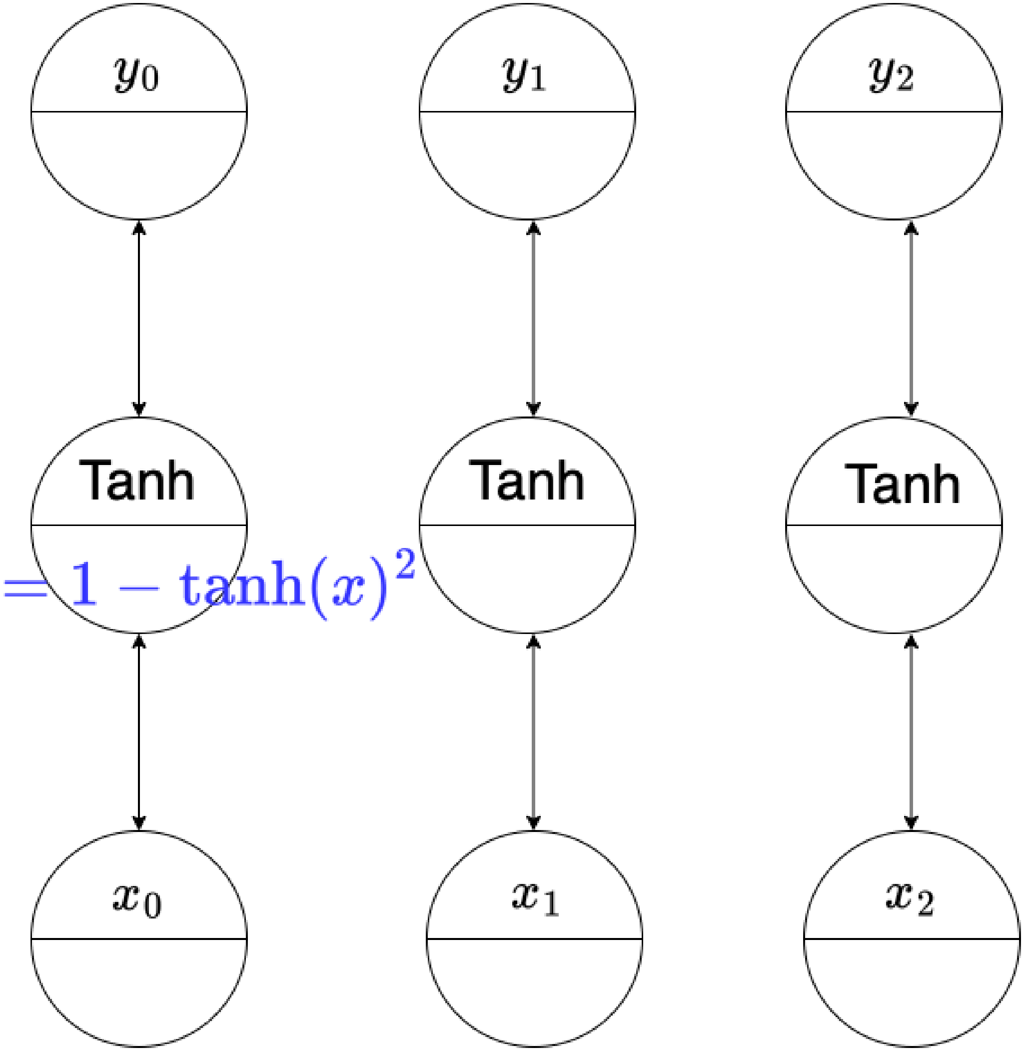


Activation Functions of Neural Nets

- Sigmoid
- Tanh
- ReLU

(2) Activation Function:
Tanh (Hyperbolic Tangent Unit)





$$\frac{\partial J}{\partial y_0}$$

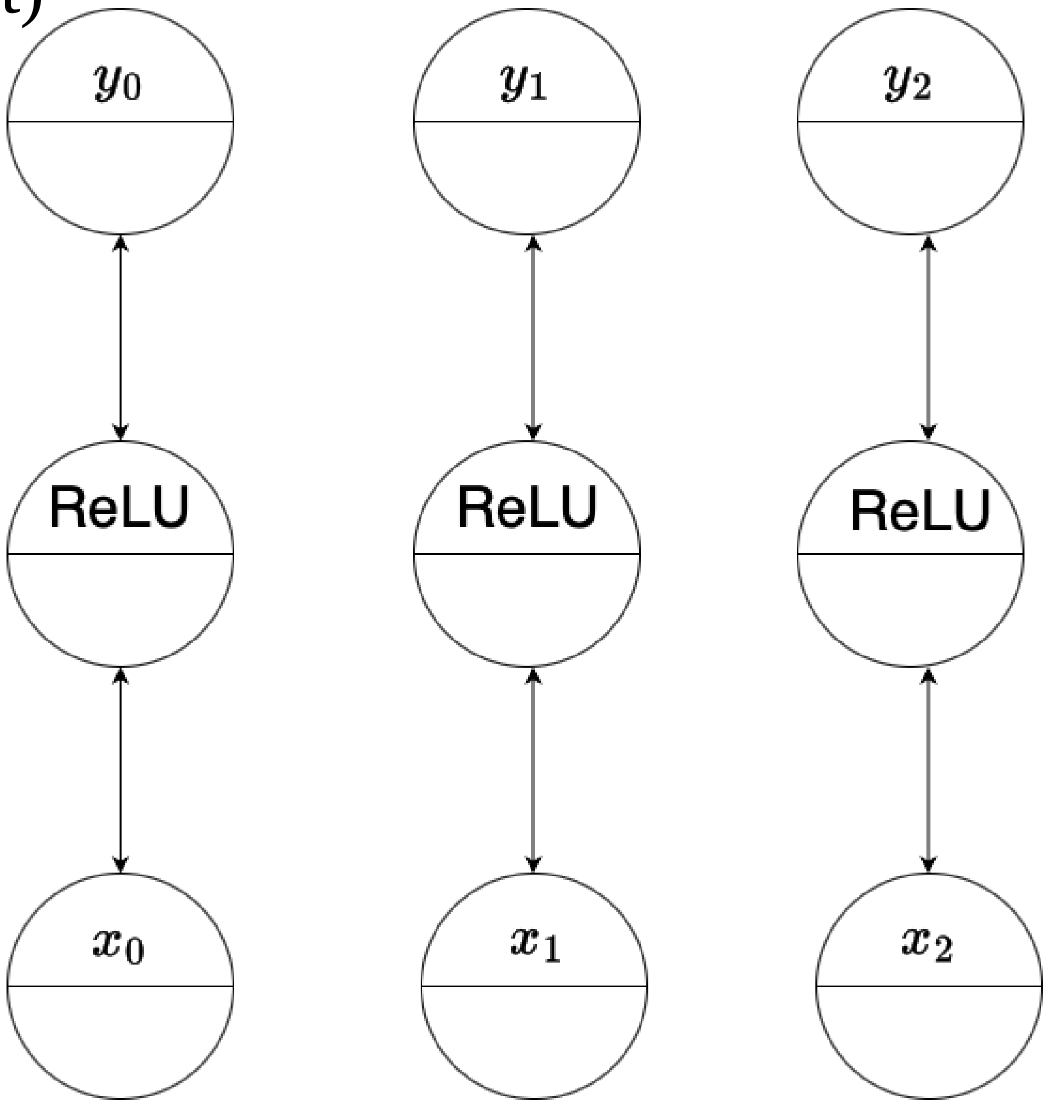
$$\frac{dy_0}{dx_0} = 1 - \tanh(x)^2$$

$$\frac{dJ}{dx_0} = \frac{dJ}{dy_0} \cdot \frac{dy_0}{dx_0}$$

(1) Activation Function:

ReLU (**R**ectified **L**inear **U**nit)

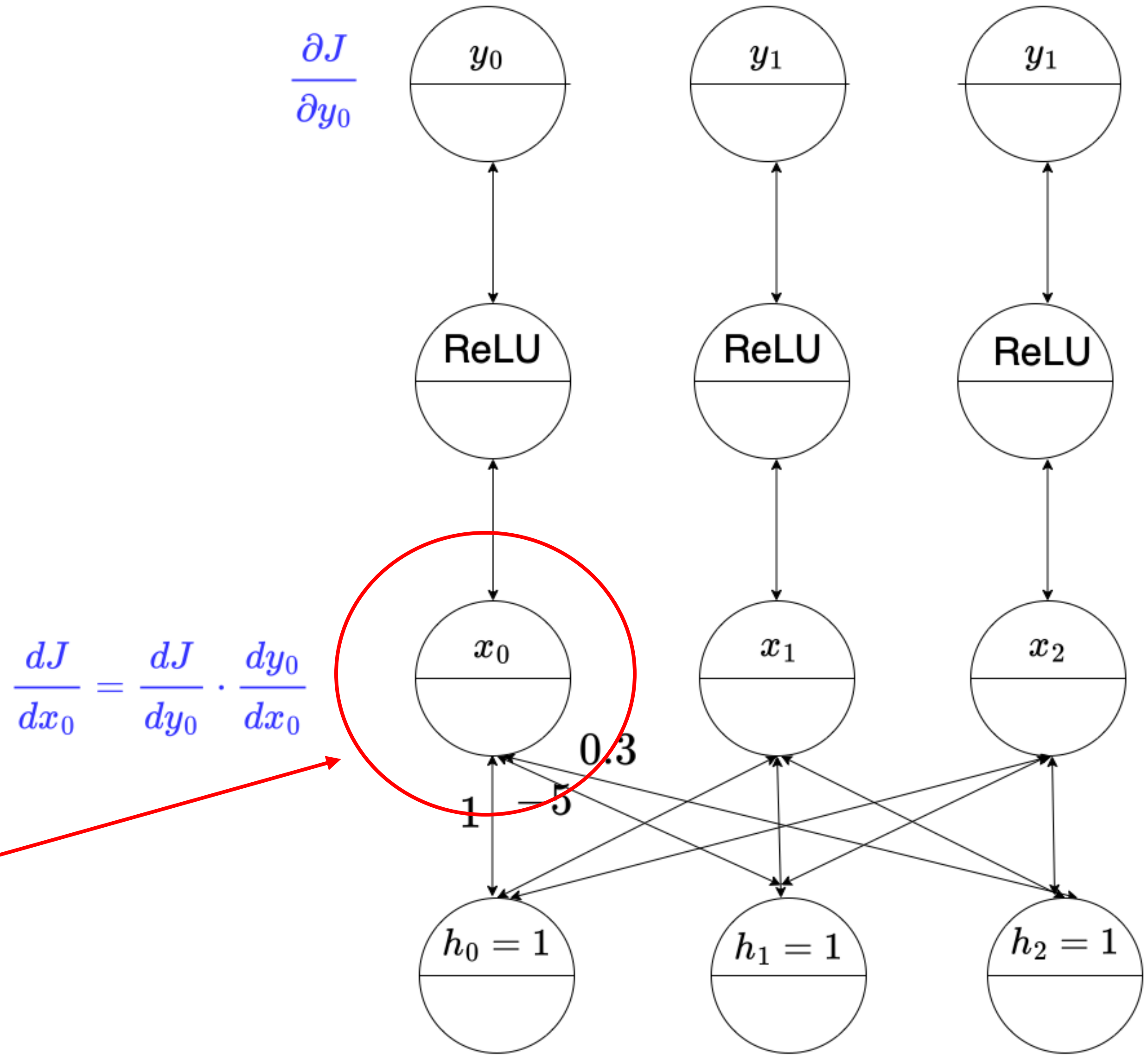
$\frac{\partial J}{\partial y_0}$



?

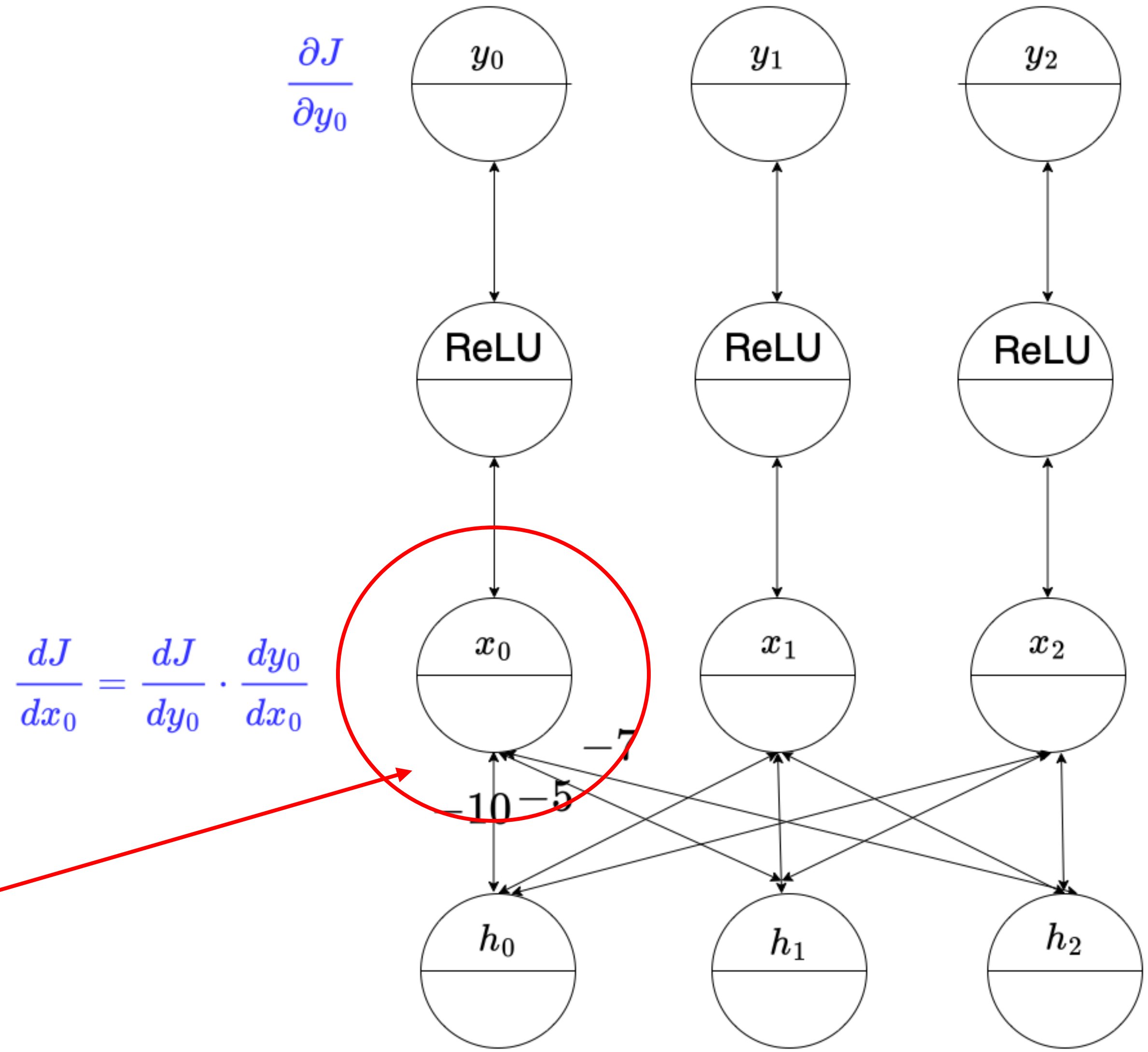
$$\frac{dJ}{dx_0} = \frac{dJ}{dy_0} \cdot \boxed{\frac{dy_0}{dx_0}}$$

(1) Dead Neuron Case for ReLU



Q: can the unit x_0 *revive*?

(2) Dead Neuron Case for ReLU



Q: can the unit x_0 $revive$?

The possible reasons for dead neurons:

- Large step size: making all parameters as big negatives.

$$W_{t+1} = W_t - \eta \nabla J(W_t)$$

- Large negative bias.

The possible solutions for dead neurons:

- Lowering the step size
- Careful weight initialization
- What if ReLU leak small gradients for the negative values?

In the next class…

- Training of Deep CNN
  - data preprocessing
  - stochastic gradient descent hyper parameter selection
    : momentum and step size
  - regularization methods
- Generalization: performance of deep CNN and adversarial examples.

Q: What does a deep CNN classify this cup as?

[“Object” by Meret Oppenheim , 1936]