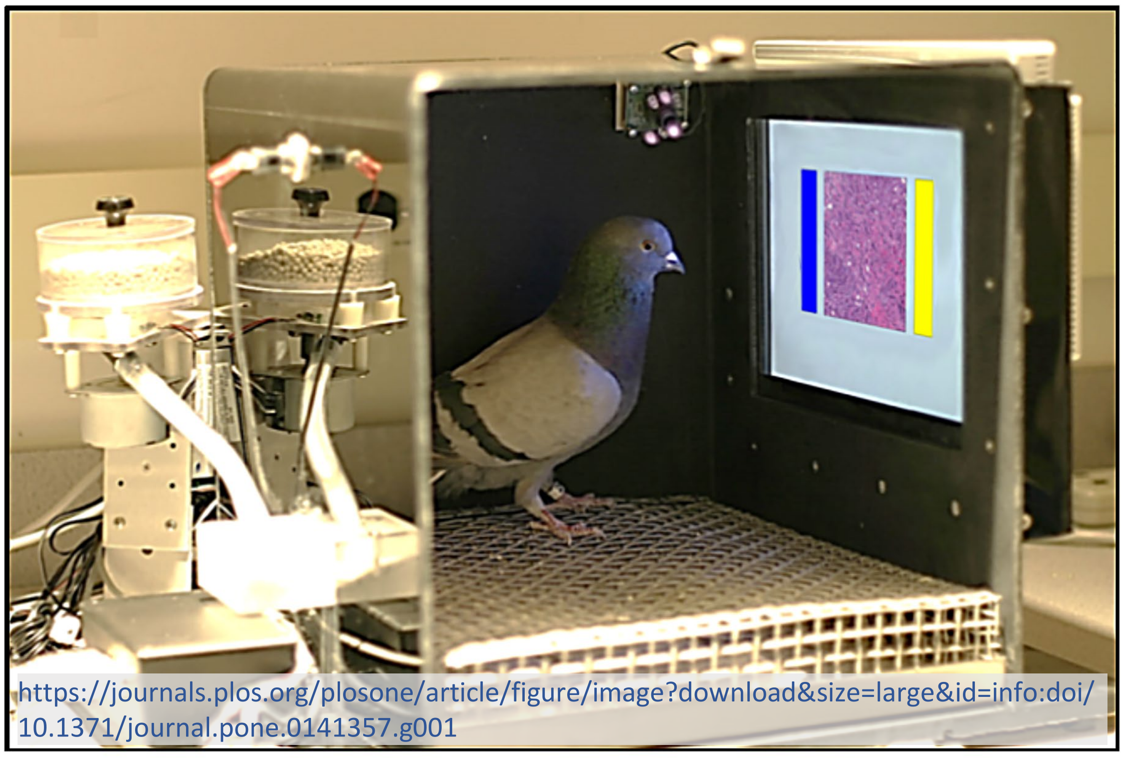
# Lecture 7: Neural Networks



https://journals.plos.org/plosone/article/figure/image?download&size=large&id=info:doi/10.1371/journal.pone.0141357.g001

Haiping Lu

YouTube Playlist: https://www.youtube.com/c/HaipingLu/

COM4059/6059: MLAI20@The University of Sheffield

# Week 7 Contents / Objectives

- Learning with Neurons
- Neural Networks (NNs)
- NN Decision Boundary & Features
- Convolutional NN Basics
- Convolutional NN Unboxing

# Week 7 Contents / Objectives

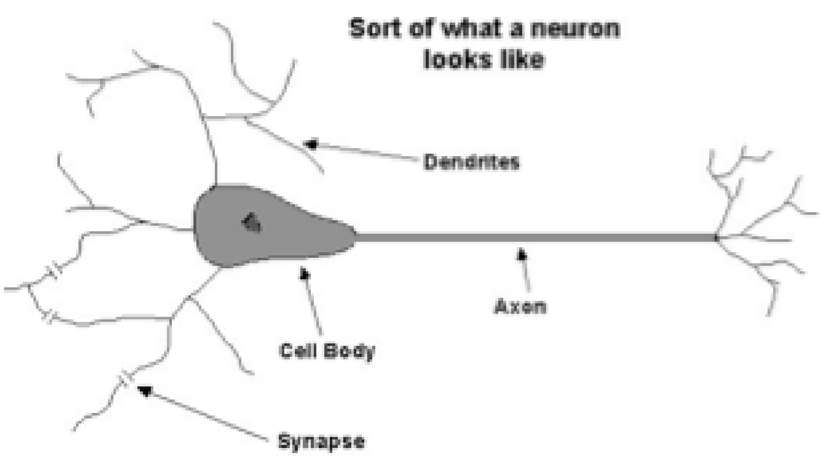
- **Learning with Neurons**
- Neural Networks (NNs)
- NN Decision Boundary & Features
- Convolutional NN Basics
- Convolutional NN Unboxing

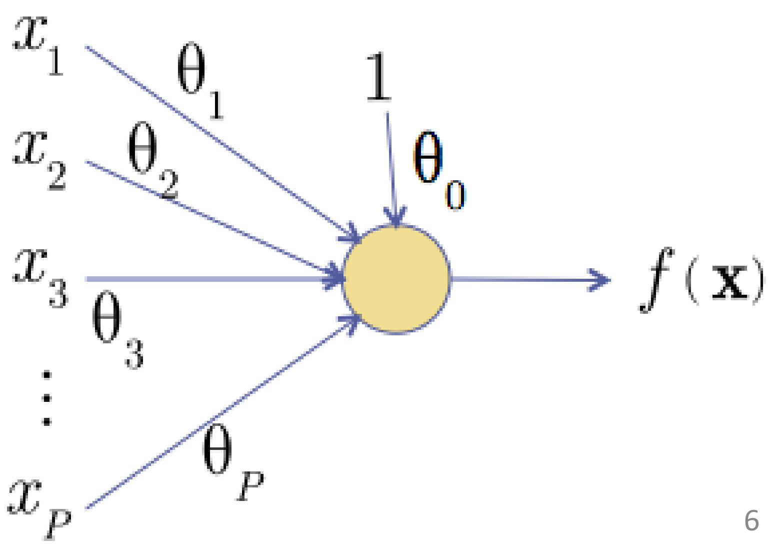

https://pbs.twimg.com/media/DqczsWhX4AErP0h.jpg:large


https://www.youtube.com/watch?v=flzGjnJLyS0

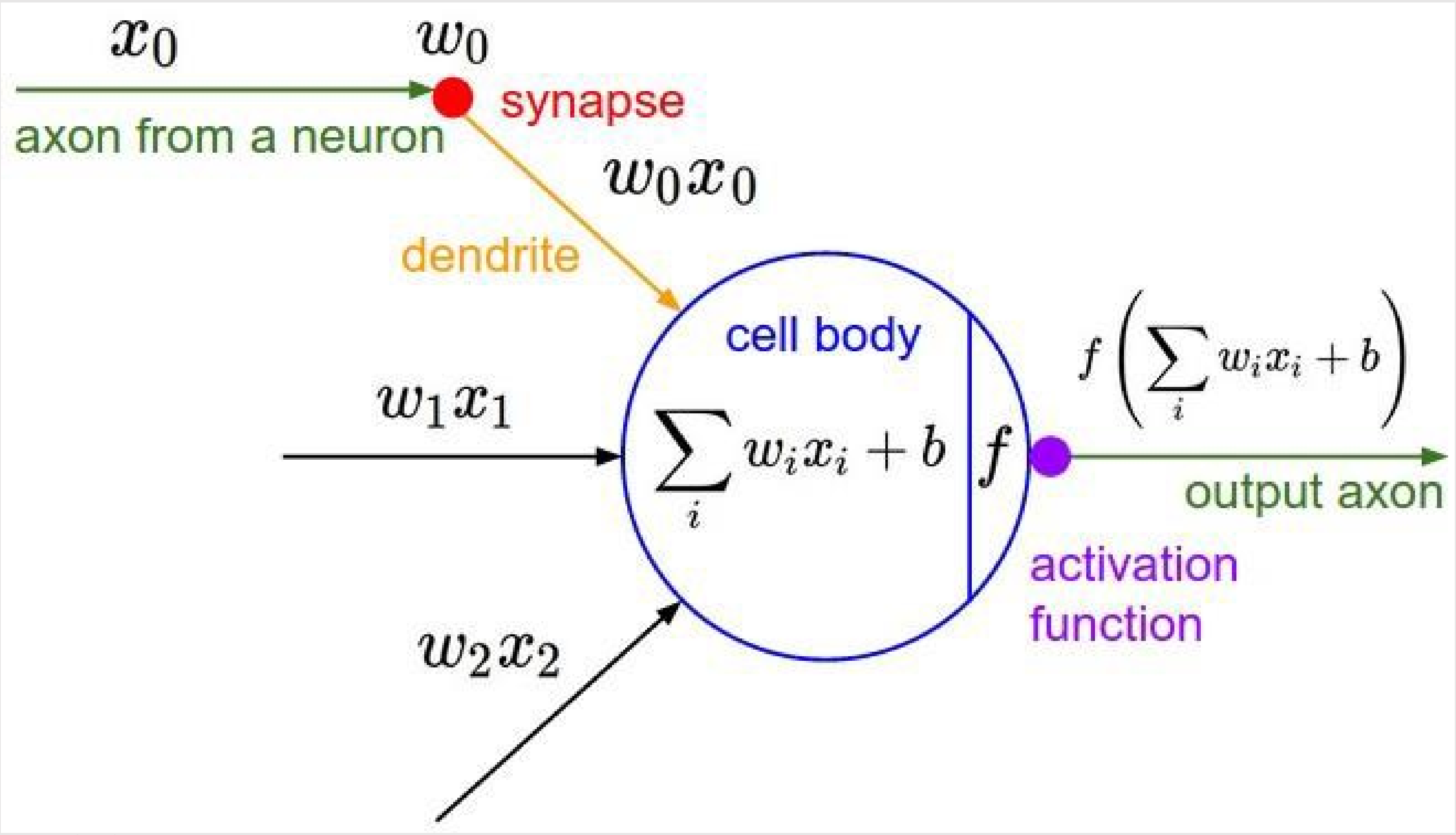
# The Neuron Metaphor

- Neurons
  - Accept information from multiple inputs
  - Transmit information to other neurons
- Multiply inputs by weights along edges
- Apply some function to the inputs at each node



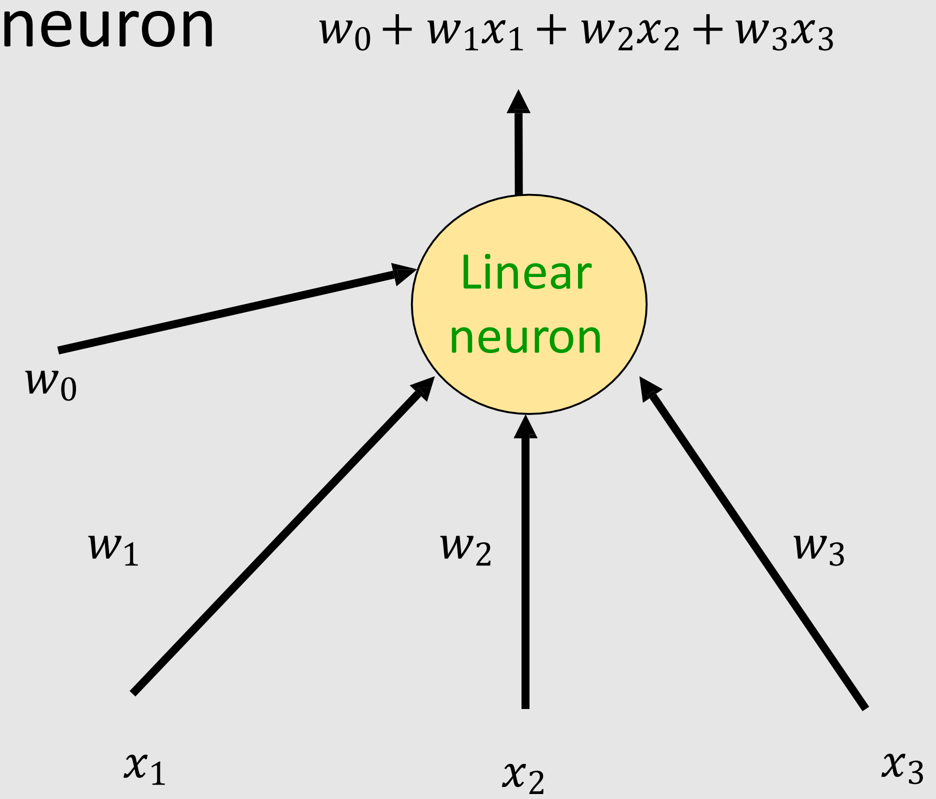



# A Neuron Analogous to the Brain



# Neural Network

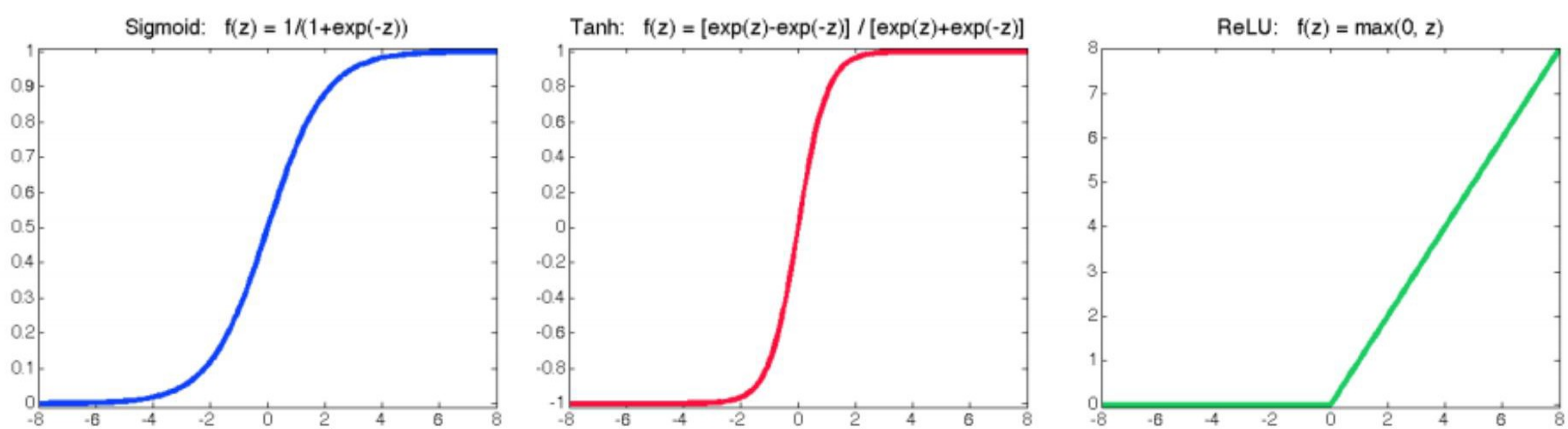
- Network of neurons
- Linear neuron



# Activation Functions

- Commonly used activation functions

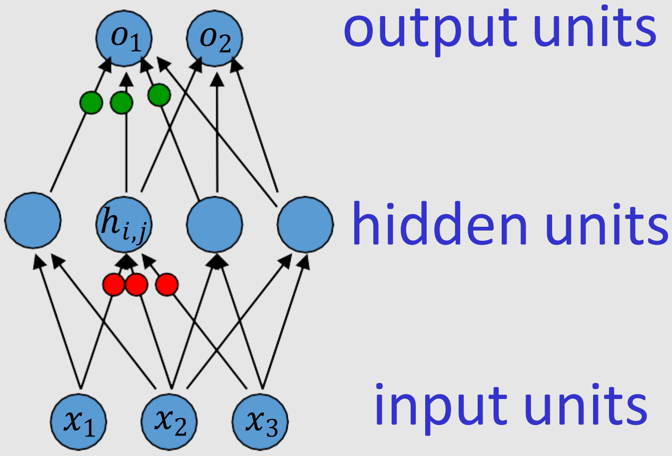
- Sigmoid: $\sigma(z) = \frac{1}{1+\exp(-z)}$
- Tanh: $\tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$
- ReLU (Rectified Linear Unit): $\mathrm{ReLU}(z) = \max(0, z)$



# Computation in Neural Networks

- Forward pass
  - Making **prediction**s (**decision**s)
  - Plug in the input $x$, get the output $y$

$$\mathbf{o} = g\left(\left(W^{(2)}\right)^T \boldsymbol{h} + b^{(2)}\right)$$
$$\mathbf{h} = g\left(\left(W^{(1)}\right)^T \boldsymbol{x} + b^{(1)}\right)$$




- Backward pass (backpropagation for optimisation)
  - Compute the gradient of the **cost** (**loss/error**) function with respect  to the weights to find good values for weights

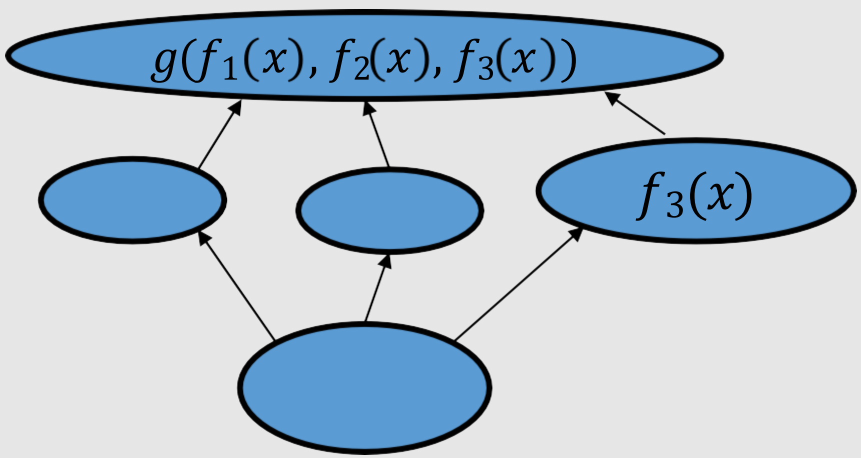
# Autograd: Chain Rule

- Univariate chain rule

$$\frac{d}{dt} g\big(f(t)\big) = \frac{dg}{df} \cdot \frac{df}{dt}$$

- Multivariate chain rule

$$\frac{\partial g}{\partial x} = \Sigma \, \frac{\partial g}{\partial f_i} \frac{\partial f_i}{\partial x}$$



# Week 7 Contents / Objectives

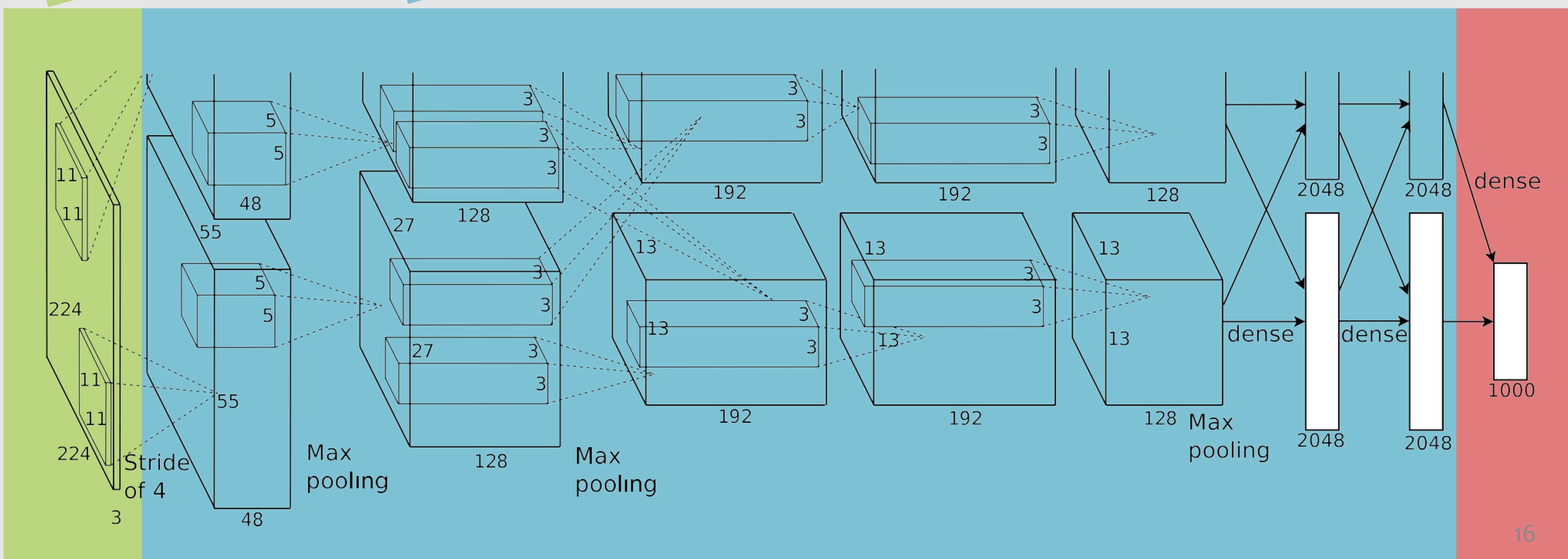
- Learning with Neurons
- **Neural Networks (NNs)**
- NN Decision Boundary & Features
- Convolutional NN Basics
- Convolutional NN Unboxing

# AlexNet for ImageNet LSVRC-2010

Input image (pixels)

- Five convolutional layers (w/max-pooling)
- Three fully connected layers

1000-way softmax



# Data, Model, Metric for Learning

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{N}$$

*Face*

*Face*

*Not a face*

2. Choose each of these:
   – Decision function/model

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

   – Loss function/metric

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

**Examples:** Linear regression, Logistic regression, Neural Network

**Examples:** Mean-squared error, Cross Entropy

# Data, Model, Metric, Optimisation

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{N}$$

2. Choose each of these:

– Decision function/model

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

– Loss function/metric

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

3. Define goal:

– Objective function

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

4. Train/optimize with SGD: (take small steps opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

# Data, Model, Metric, Optimisation

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

2. Choose each of these:

– Decision function/mod

$$\hat{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

– Loss function/metric

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

3. Define goal:

– Objective function

$$(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

*Compute* ***gradients***
*via backpropagation*
*Using automatic*
*differentiation*

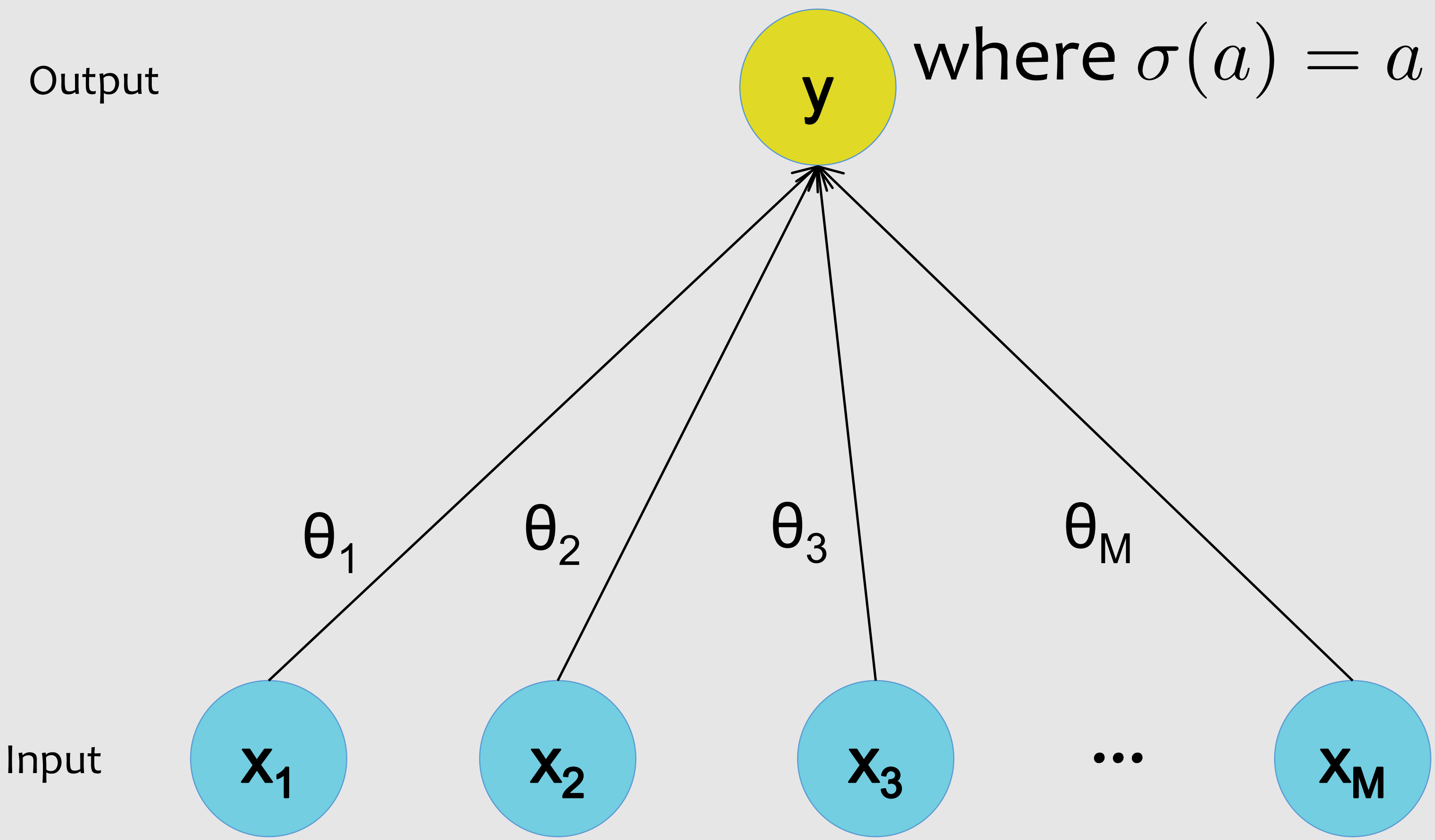
opp site the gradient)

$$\boldsymbol{\theta}^{(t} \quad {}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

# Linear Regression Model (x → y)

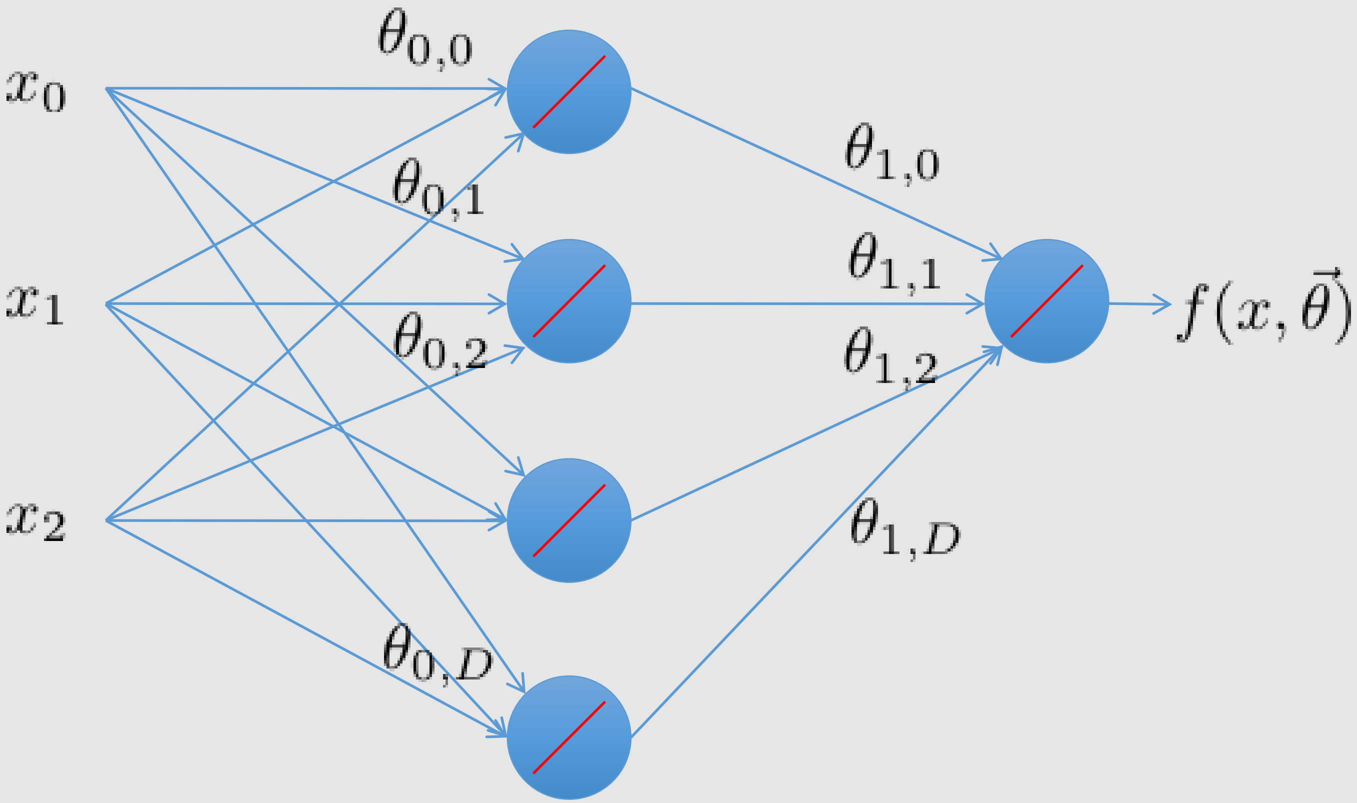
$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

where $\sigma(a) = a$



# Linear Regression Neural Networks

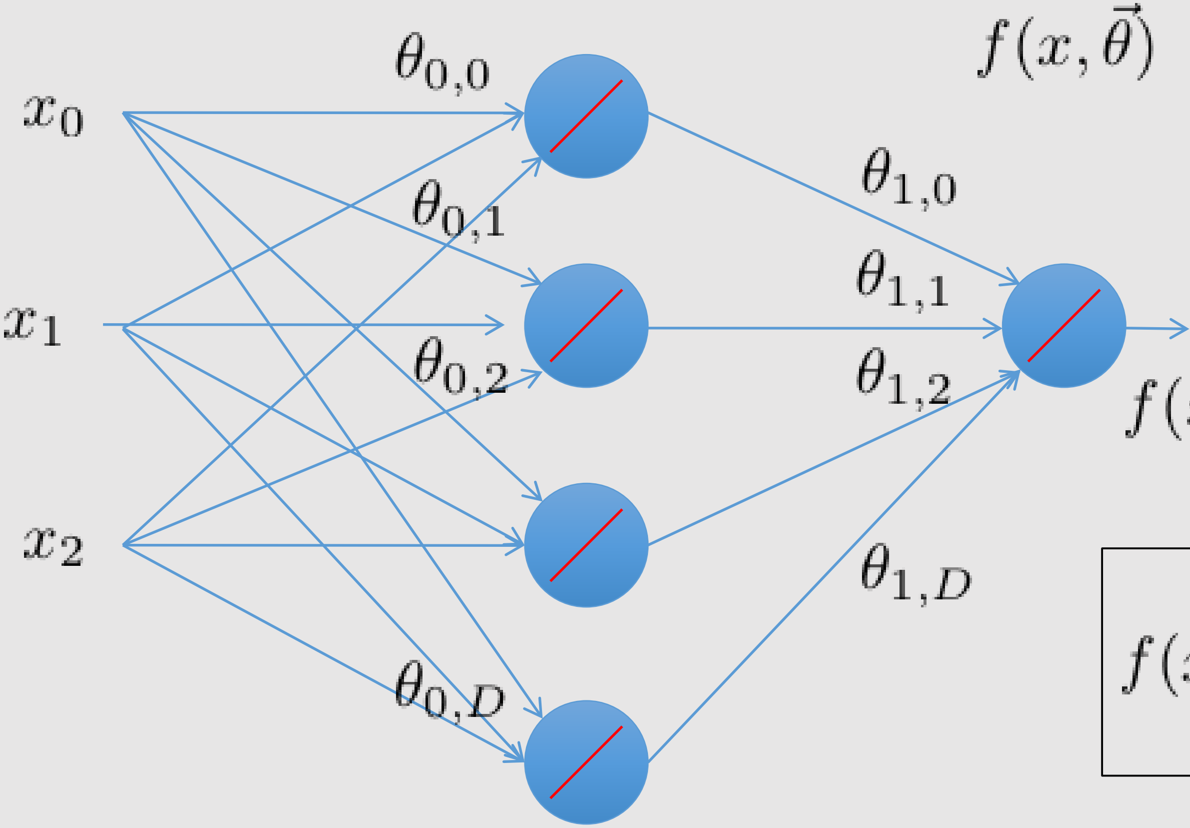
- **Question**: What happens when we arrange **linear neurons** in a multilayer network?



# Linear Regression Neural Networks

- Nothing special happens.
  - The product of two linear transformations is itself a linear transformation.

$$f(x,\vec{\theta}) = \sum_{i=0}^{D}\theta_{1,i}\sum_{n=0}^{N-1}\theta_{0,i,n}x_n$$



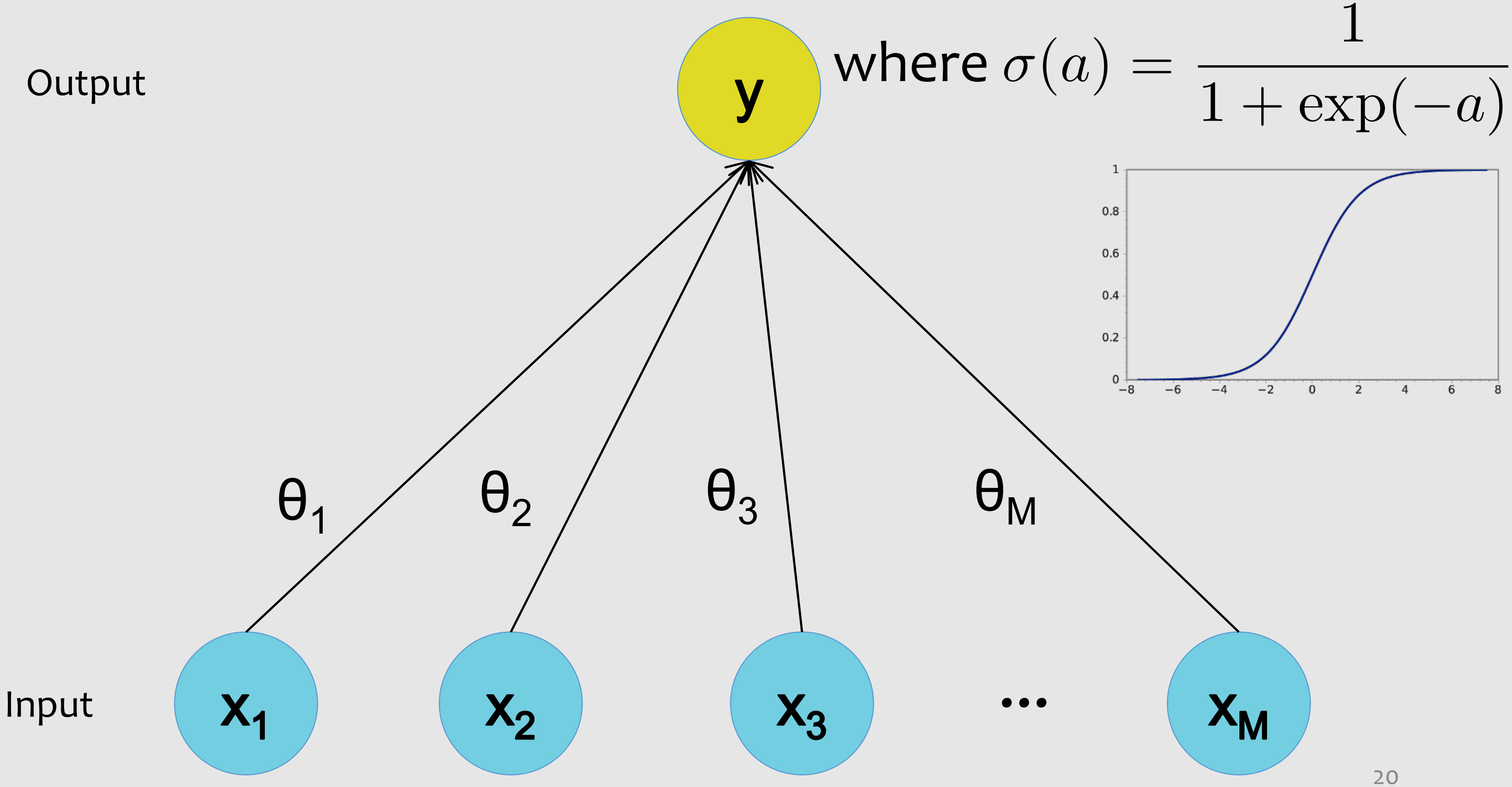

$$f(x,\vec{\theta}) = \sum_{i=0}^{D}\theta_{1,i}[\theta_{0,i}^{T}\vec{x}]$$

$$\boxed{f(x,\vec{\theta}) = \sum_{i=0}^{D}[\hat{\theta}_i^{T}\vec{x}]}$$

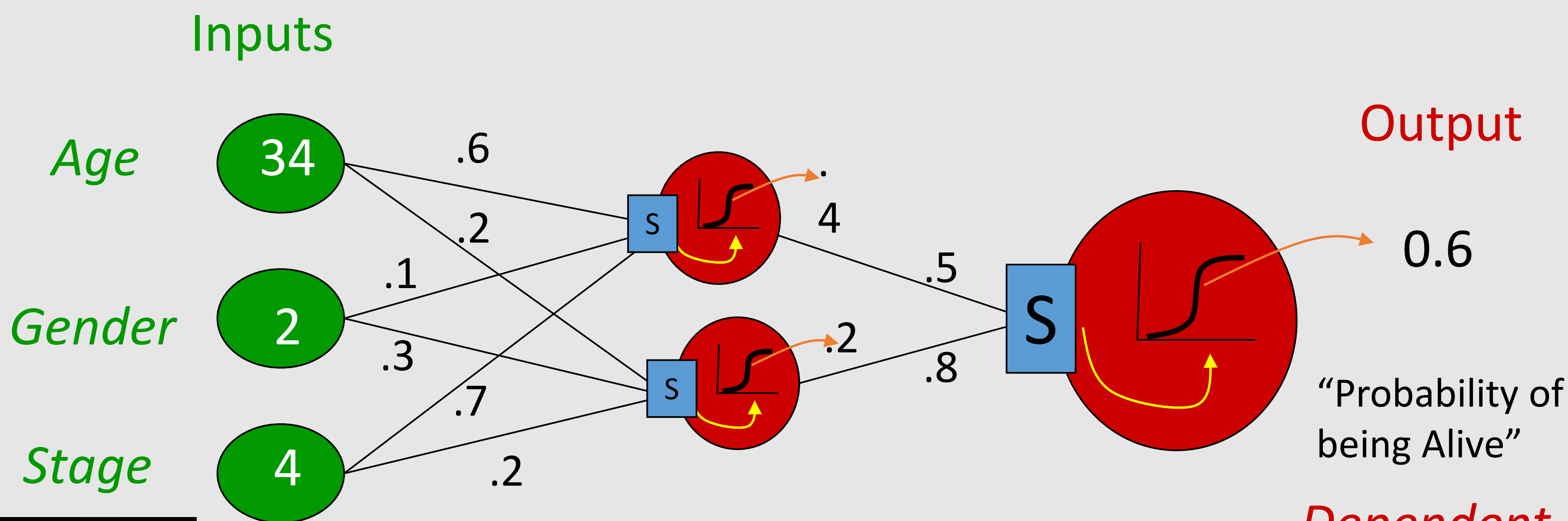
# Logistic Regression Model (x → y)

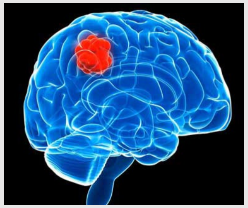
$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

where $\sigma(a) = \dfrac{1}{1 + \exp(-a)}$



# Logistic Regression Neural Networks



*Independent (input) variables*

Weights (Coefficients )

Hidden layer

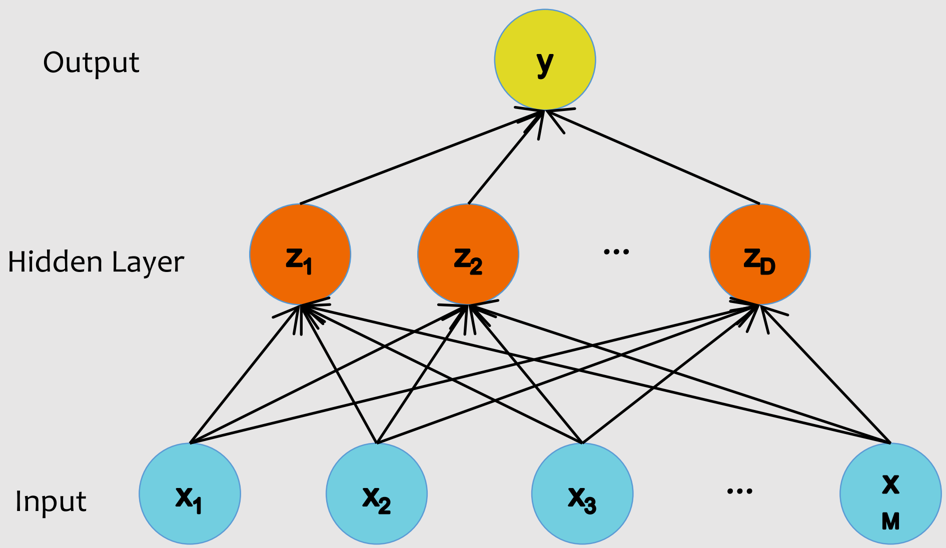
Weights (Coefficients )

*Dependent (output) variable (Prediction /estimate/ target)*

# Week 7 Contents / Objectives

- Learning with Neurons
- Neural Networks (NNs)
- **NN Decision Boundary & Features**
- Convolutional NN Basics
- Convolutional NN Unboxing

# Hidden Layers → Decisions




**(E) Output (sigmoid)**

$y = \frac{1}{1+\exp(-b)}$

**(D) Output (linear)**

$b = \sum_{j=0}^{D} \beta_j z_j$

**(C) Hidden (sigmoid)**

$z_j = \frac{1}{1+\exp(-a_j)}, \ \forall j$

**(B) Hidden (linear)**

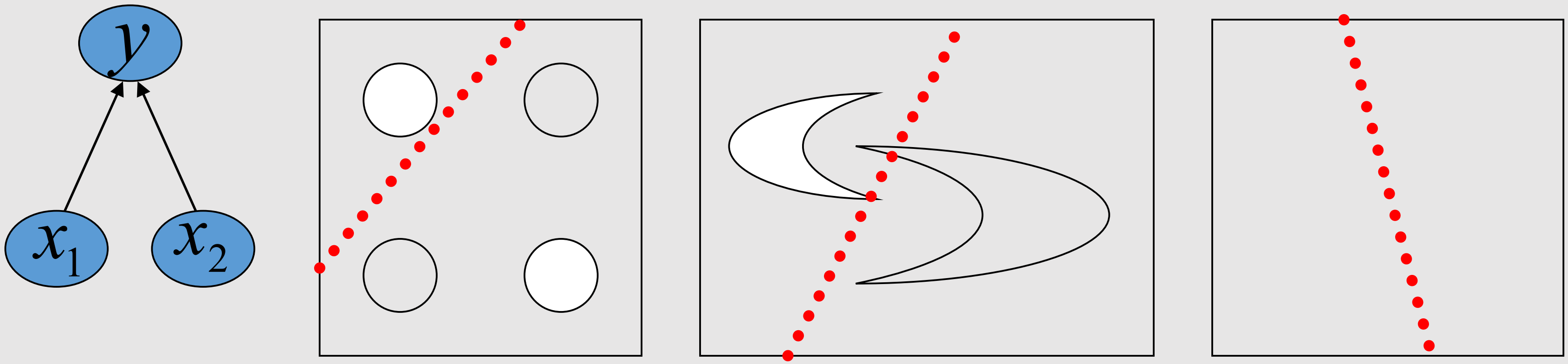
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$

**(A) Input**

Given $x_i, \ \forall i$

# Decision Boundary

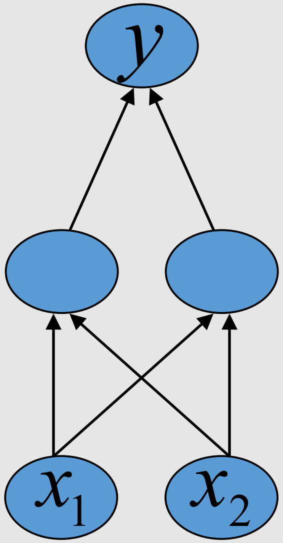
- 0 hidden layers: linear classifier
  - Hyperplanes

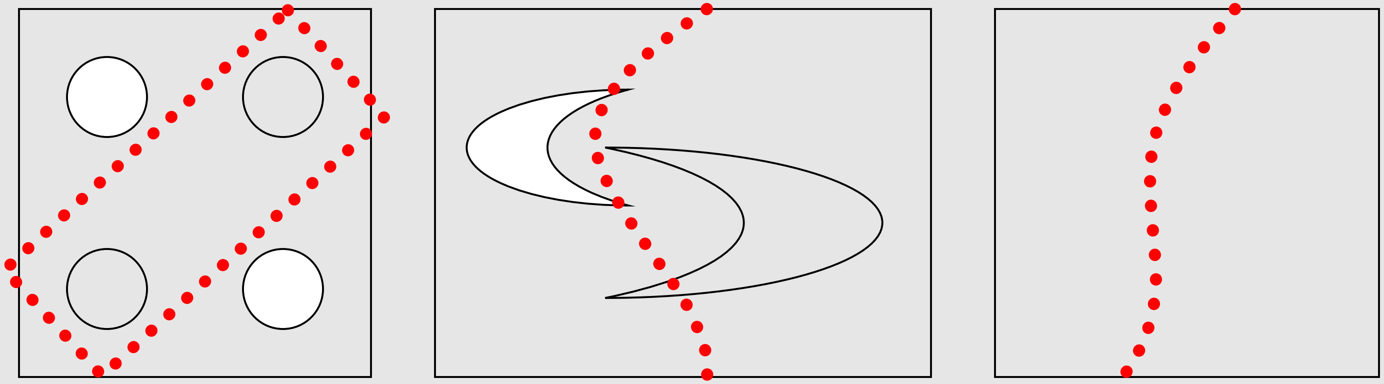



Example from to Eric Postma via Jason Eisner

# Decision Boundary

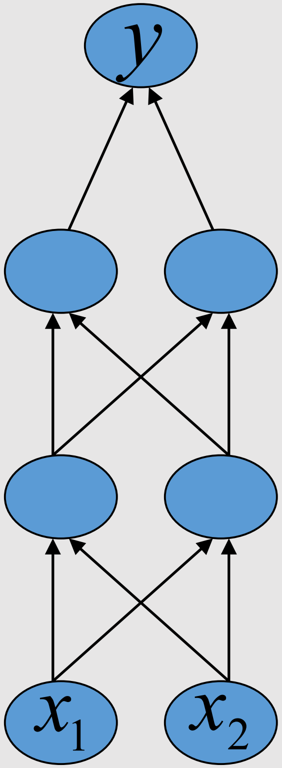
- 1 hidden layer
  - Boundary of **convex** region (open or closed)



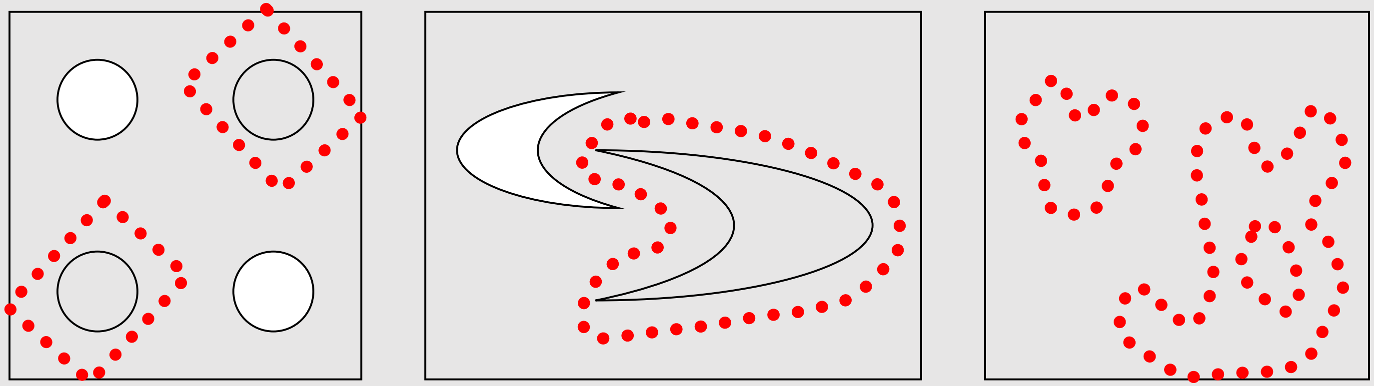



Example from to Eric Postma via Jason Eisner

# Decision Boundary

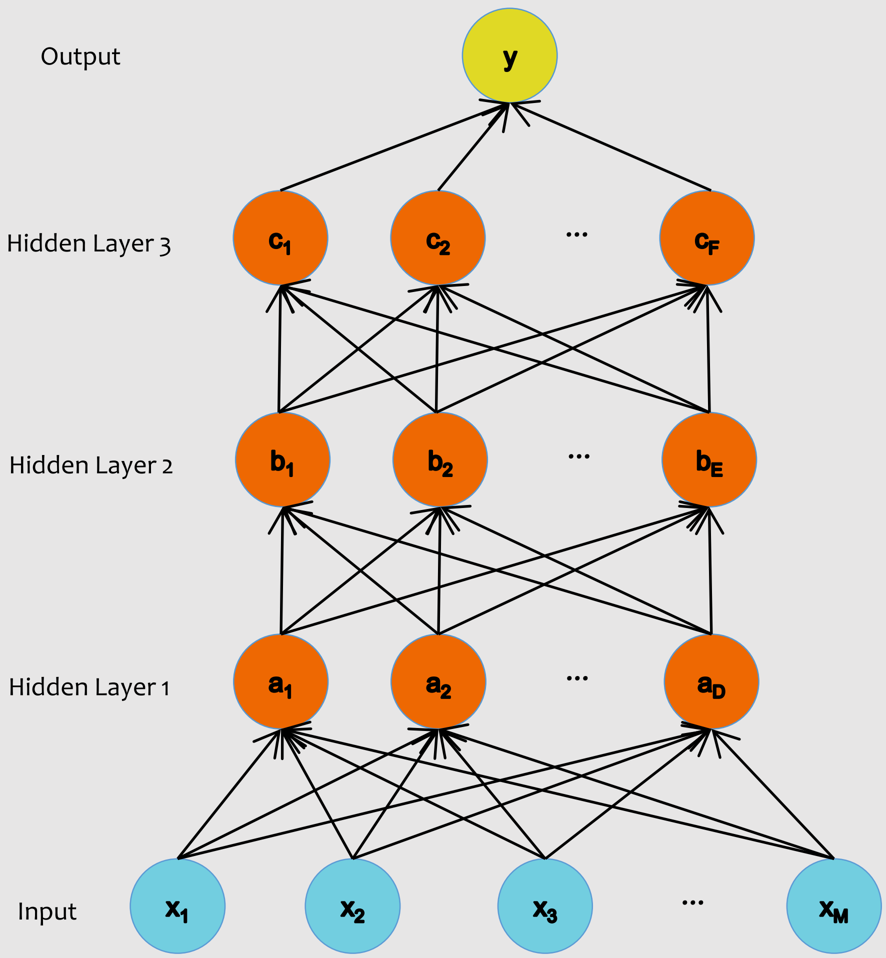



- 2 hidden layers
  - Combinations of convex regions



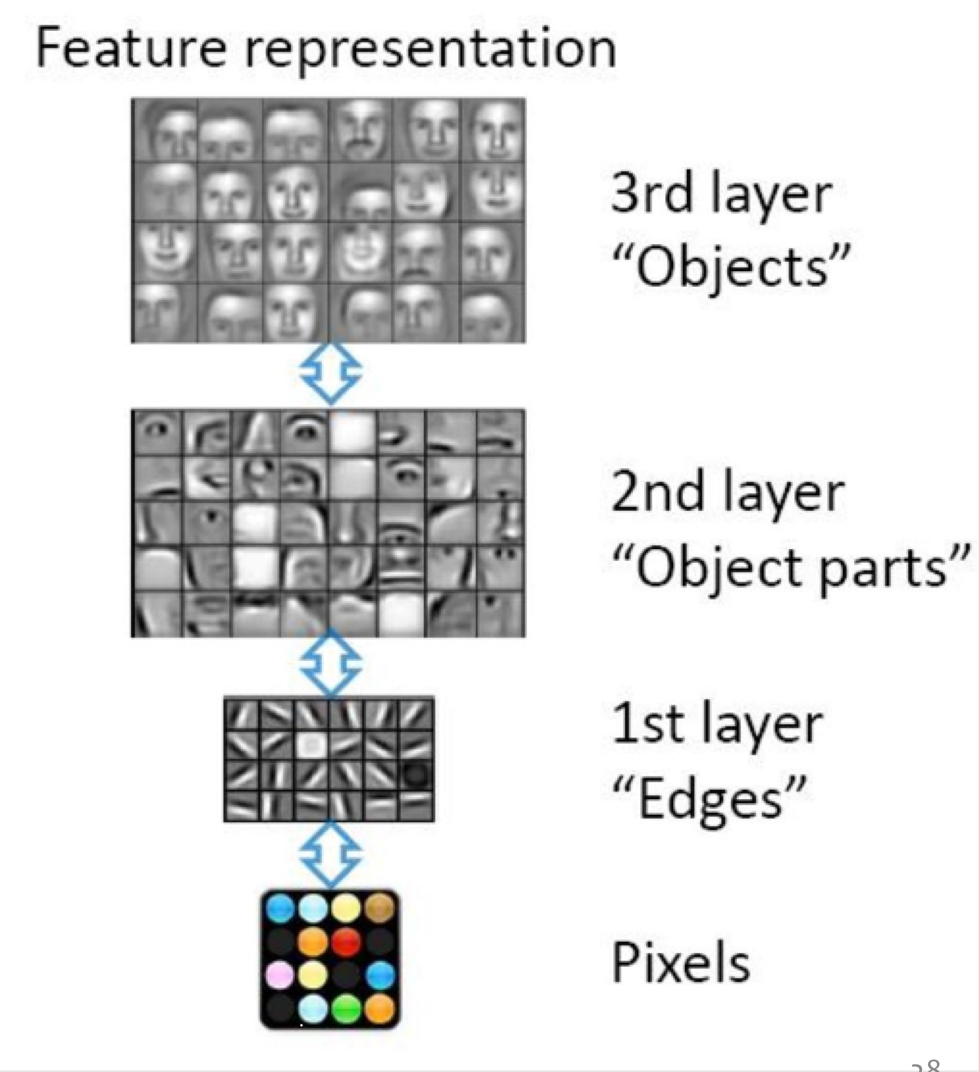
Example from to Eric Postma via Jason Eisner

# Deeper Networks



# Different Levels of Abstraction

- We don’t know the “right” levels of abstraction
- So let the model figure it out!

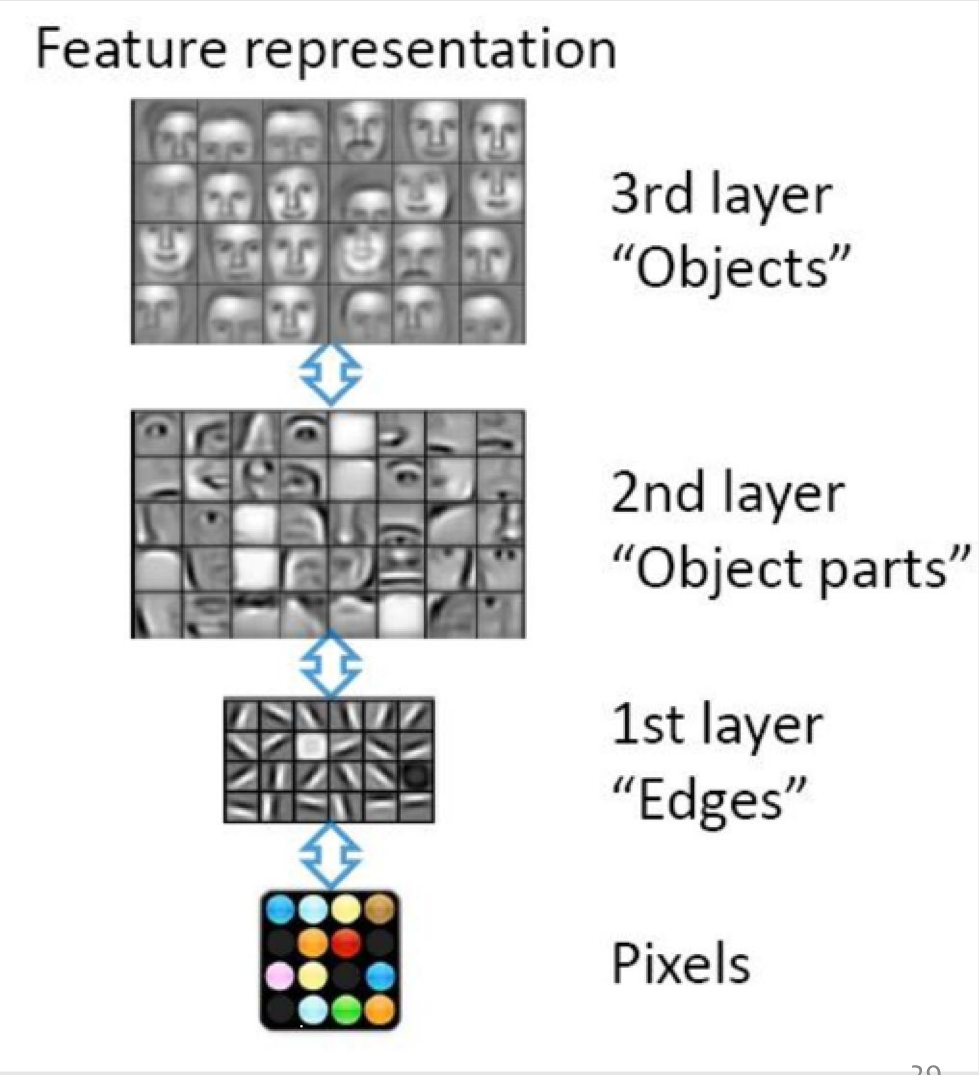



Example from Honglak Lee (NeurIPS 2010)

# Different Levels of Abstraction

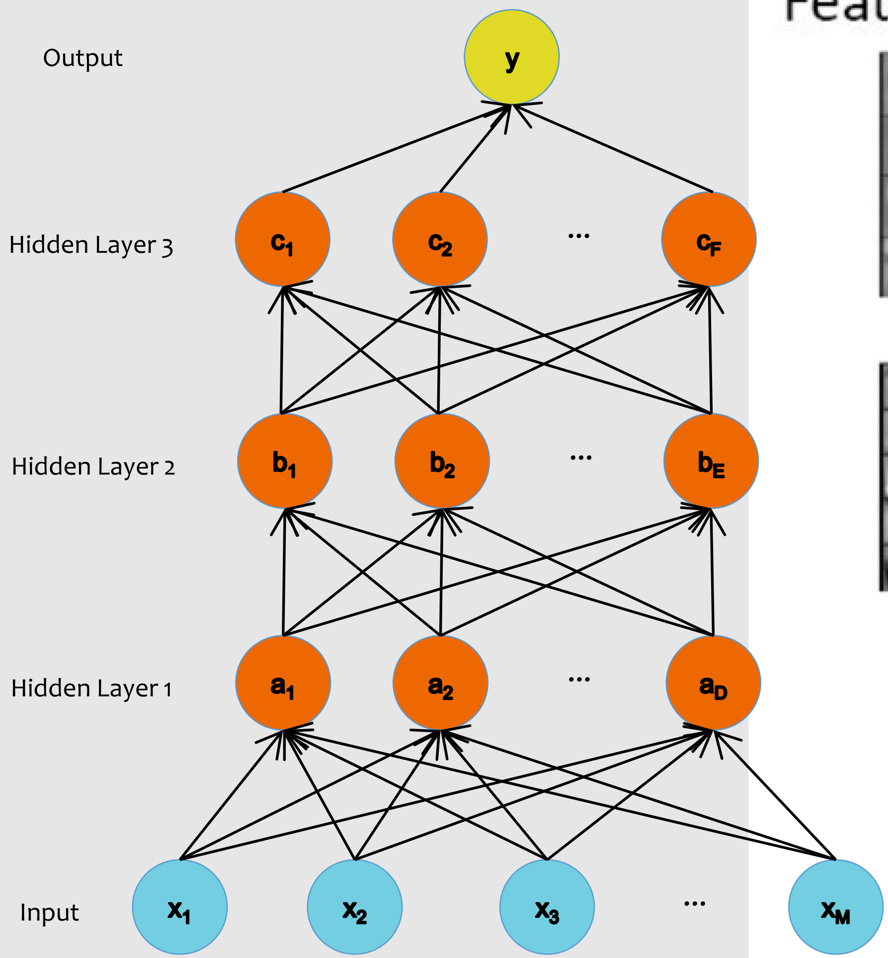
**Face Recognition:**

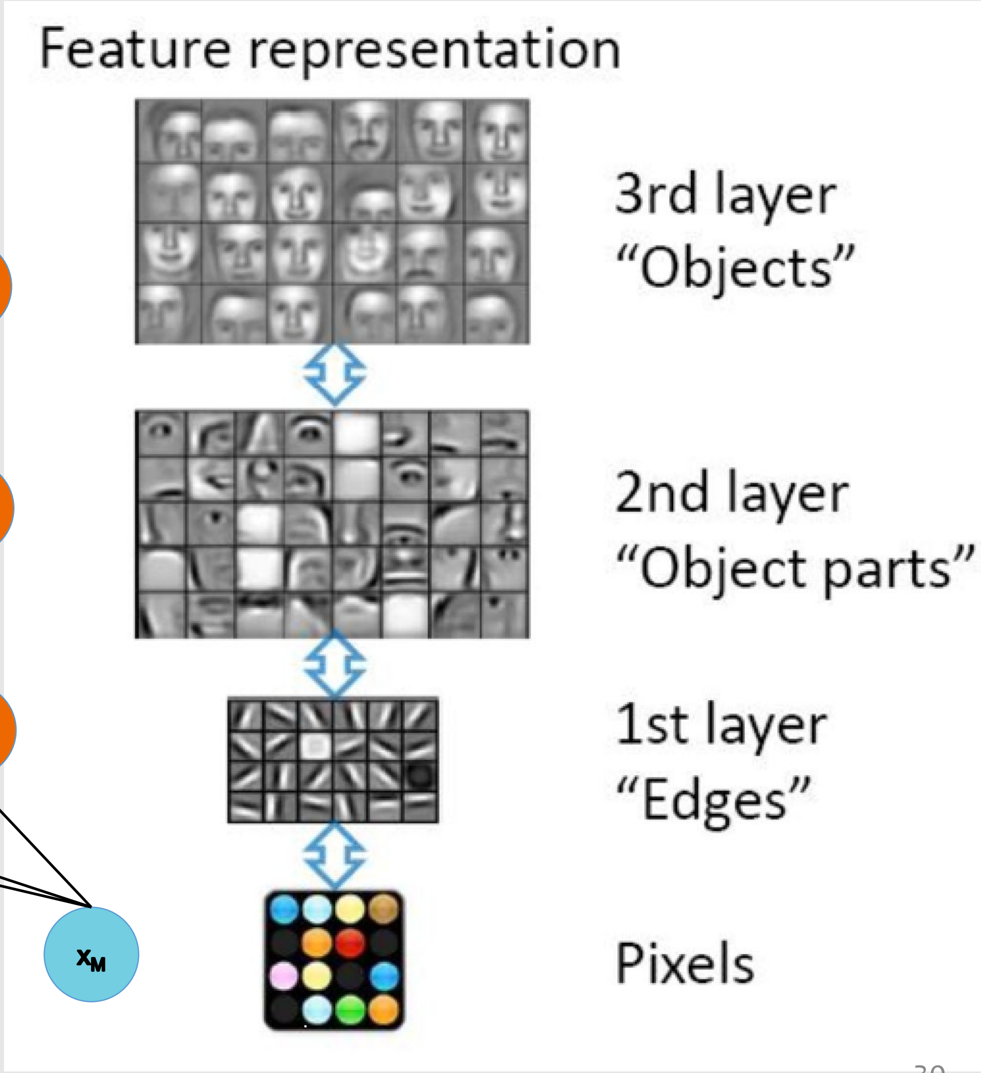
- Deep Network can build up increasingly higher levels of abstraction
- Lines, parts, regions




Example from Honglak Lee (NeurIPS 2010)

# Different Levels of Abstraction




Example from Honglak Lee (NeurIPS 2010)

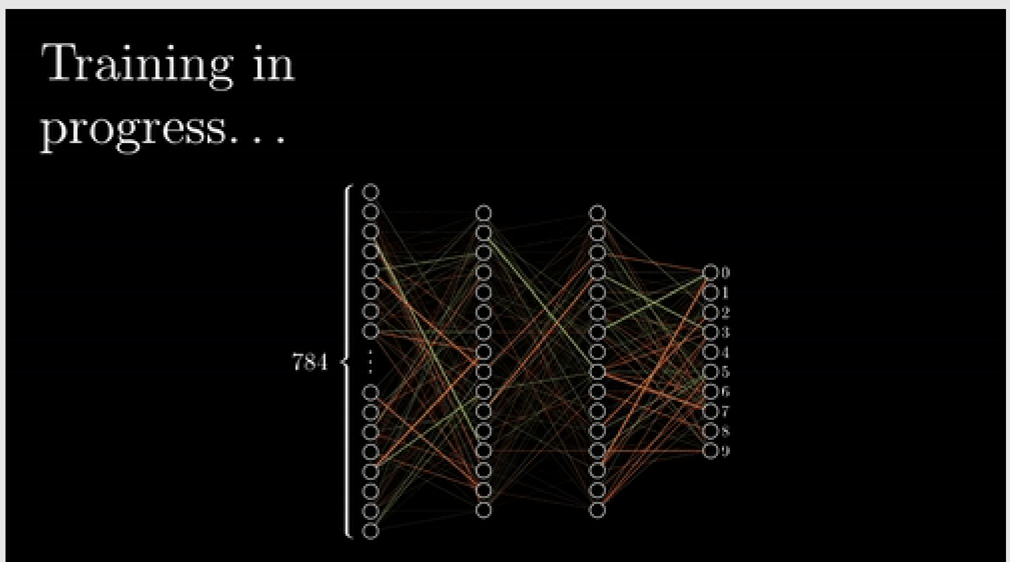
# Machine Learning Ingredients

- **Data**: + pre-processing (& visualisation), e.g., $\mathcal{N}(0,1)$
- **Model**
  - Structure ~ Architecture ← expert knowledge
    - Must **specify** before ML, can optimise via cross validation (CV)
  - **Hyper-parameter**, e.g., prior, #degree, layer ← knowledge
    - Must **specify** (choices) and can optimise via CV (***tuning***)
  - Parameters (theta)
    - Compute/learn parameter, e.g., **weights**, bias ← optimisation alg.
- Evaluation **metric** (what's best): loss/error function
- **Optimisation**: (how to find the best) learnable parameters

# Neural Network Training

Sample labeled data (**batch**) → **Forward** it through the network, get predictions → **Back-propagate** the errors → **Update** the network weights

Data → Model → Metric → Optimisation



https://slazebni.cs.illinois.edu/fall18/assignment2/nn.gif

# Neural Network Ingredients

- **Data**: + pre-processing, e.g., $\mathcal{N}(0,1)$
- **Model**
  - Structure/Architecture: layered network
  - **Hyper-parameter**: layer specs, e.g. #layers, #neurons/units, activation function
  - Parameters (theta): layer weights & biases
- Evaluation metric (loss): max likelihood (min NLL), cross-entropy, etc.
- Optimisation: backpropagation (gradient-based)

# Week 7 Contents / Objectives

- Learning with Neurons
- Neural Networks (NNs)
- NN Decision Boundary & Features
- **Convolutional NN Basics**
- Convolutional NN Unboxing

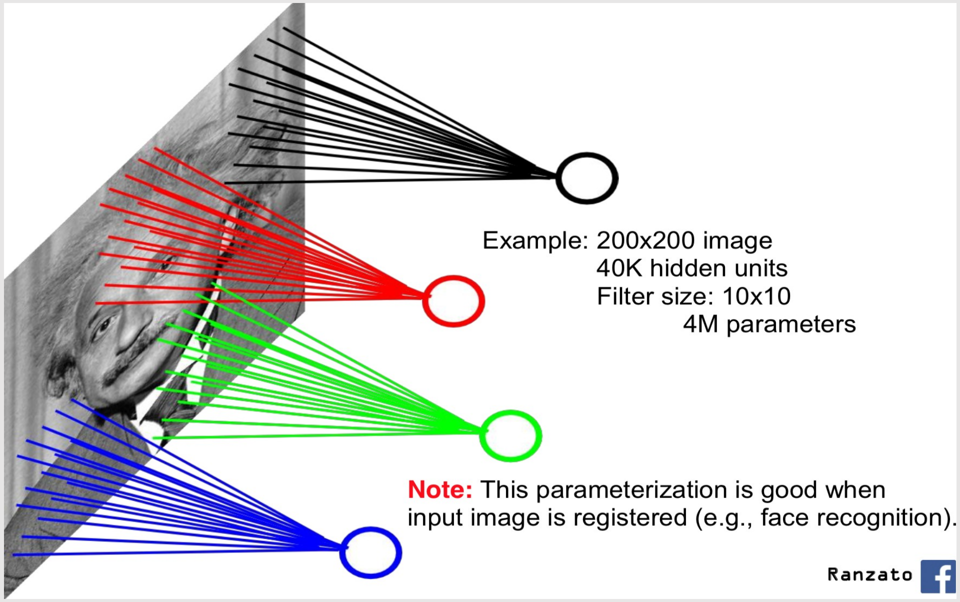
# Fully Connected (FC) Layer

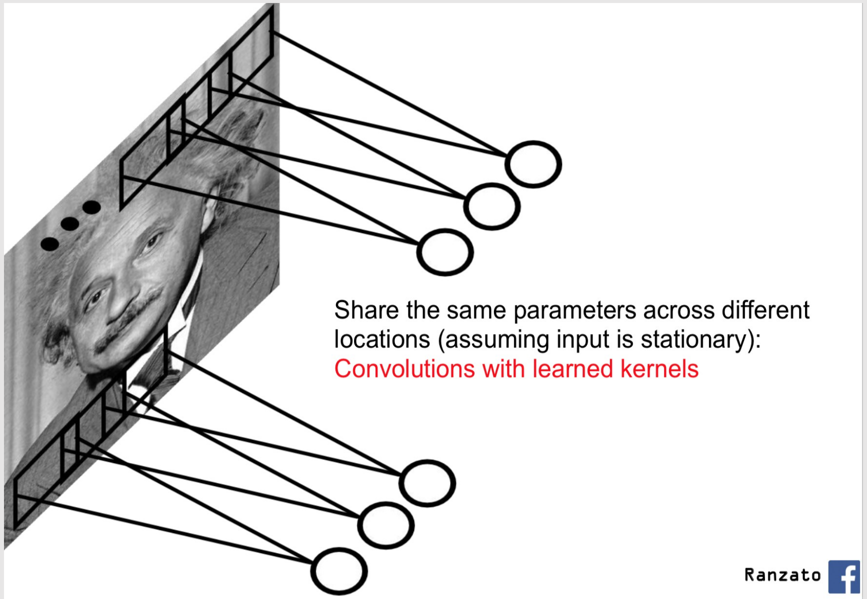
- Linear layers such as linear/logistic regression
- What if our network is bigger?
  - Input image: 200 × 200 pixels, first hidden layer: 500 units
  - **Question**: How many weights for input→1st hidden?

    **20 million**
  - **Q**: Why is using an FC layer problematic for images?
    - Computing predictions (forward pass) will take a long time
    - A large number of weights requires a lot of training data to avoid overfitting
    - Small shift in image can result in large change in prediction
    - Not making use of the image geometry

# Convolutional Neural Network

- Key ideas:
  - Locally-connected layers: look for local features in small regions of the image
  - Weight-sharing: detect the same local features across the  entire image

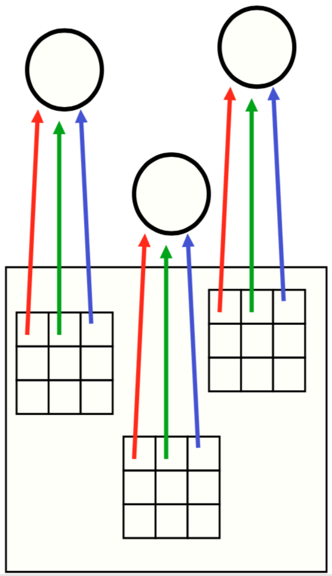


# Weight Sharing

- Each neuron on the higher layer detects the same feature, but in different locations on the lower layer

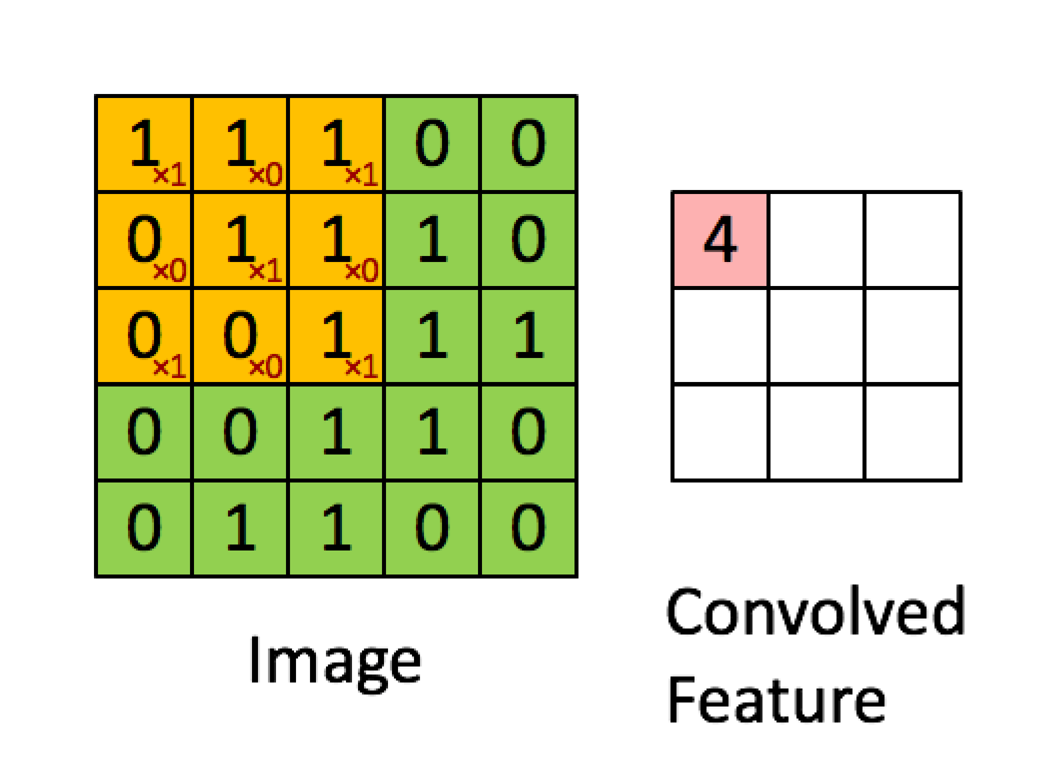
The red connections all have the same weight.



“Detecting” = the output (activation) is high if the feature is present

“Feature” = something in the image, like an edge, blob or shape

# Forward Pass Example (Single Channel)




https://developer.nvidia.com/sites/default/files/pictures/2018/convolution-2.gif

- The kernel/filter (yellow) contains the trainable weights. In the above, the kernel *size* is 3 ×3.
- The "*convolved features*" is another term for "convolution output"

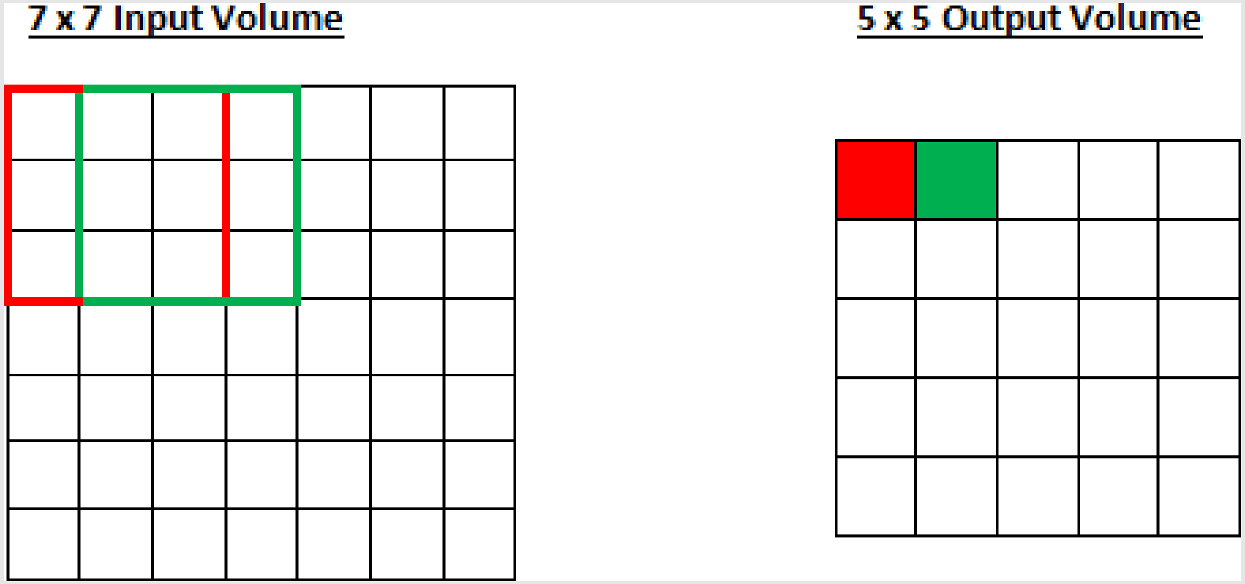
# Example of convolution

Greyscale input image: $7 \times 7$

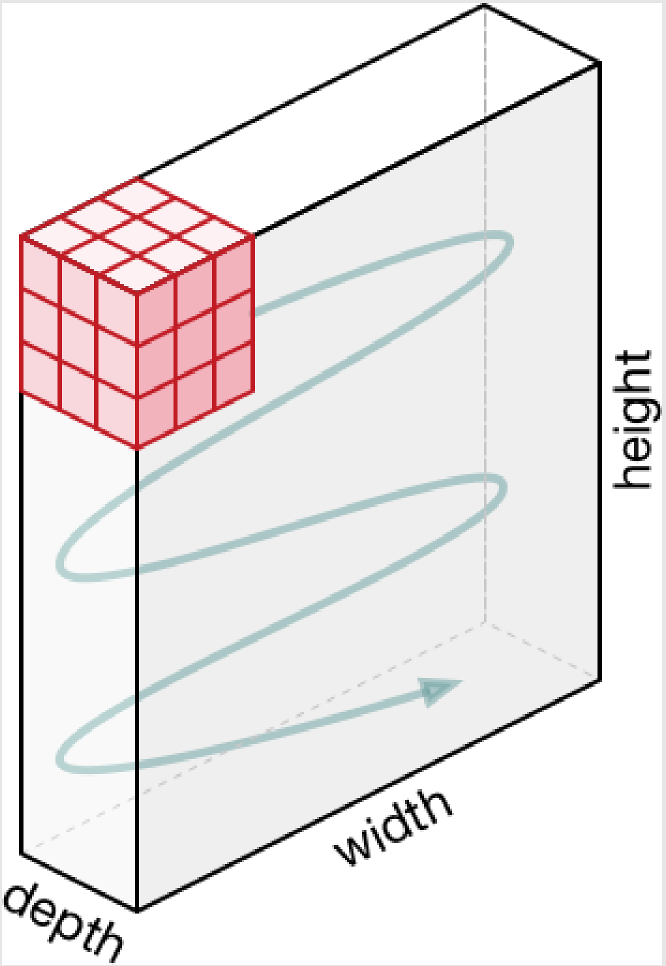
Convolution **kernel**: $3 \times 3$

**Questions**:

- How many units are in the output?
- How many trainable weights are there?



# Convolution in RGB for colour images

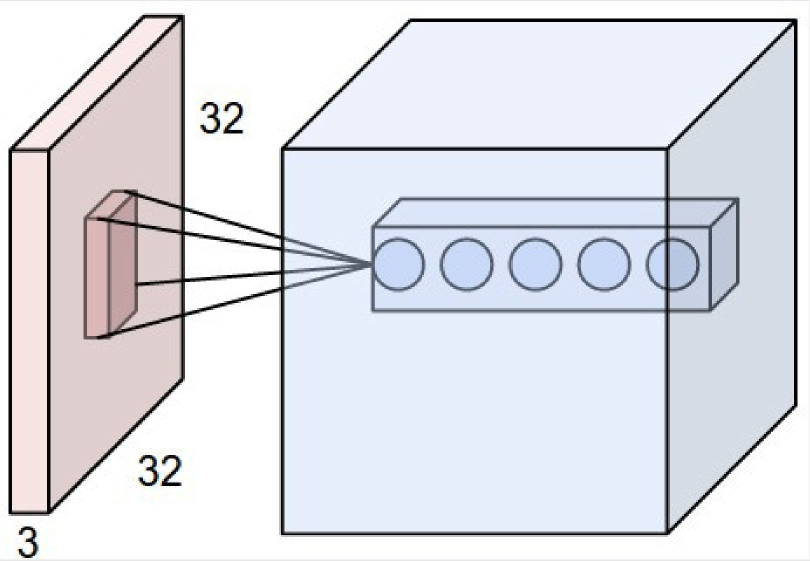



The kernel: a 3-D tensor! In this example, the kernel has size **3** ×3 × 3.

The first number 3: the number of **input channels** or **input feature maps**

# Detecting Multiple Features

- **Q**: What if we want to detect many features of the input? (e.g. **both** horizontal edges and vertical edges, and maybe even other features?)
- **A**: Have many convolutional filters!

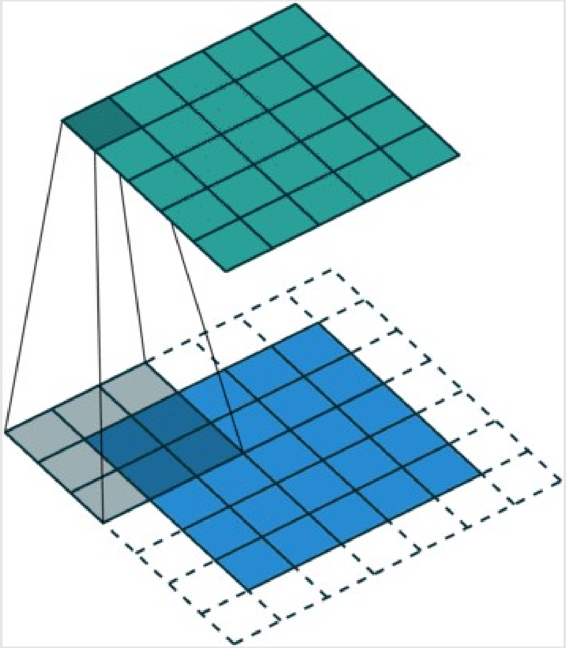



Input image size: $3 \times 32 \times 32$

Convolution kernel (4D): $\underline{\mathbf{3}} \times 3 \times 3 \times \underline{\mathbf{5}}$

- The number **3** is the number of **input channels** or **input feature maps**
- The number **5** is the number of **output channels** or **output feature maps**

# Zero Padding

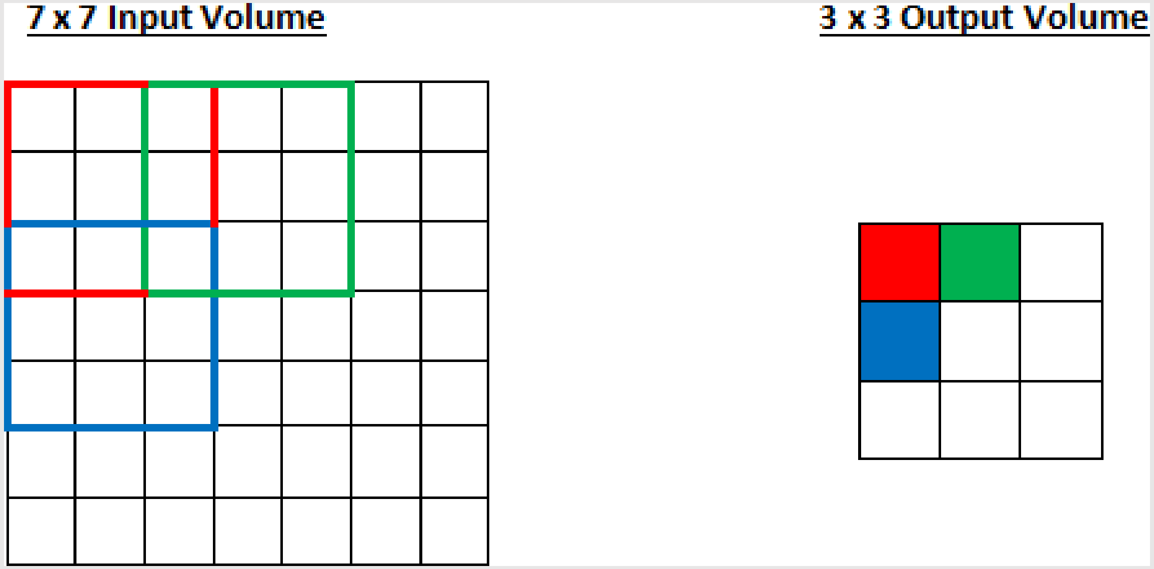


Add zeros around the border of the image (can add more than one pixel of zeros)

Q**uestion**: Why might we want to add zero padding?

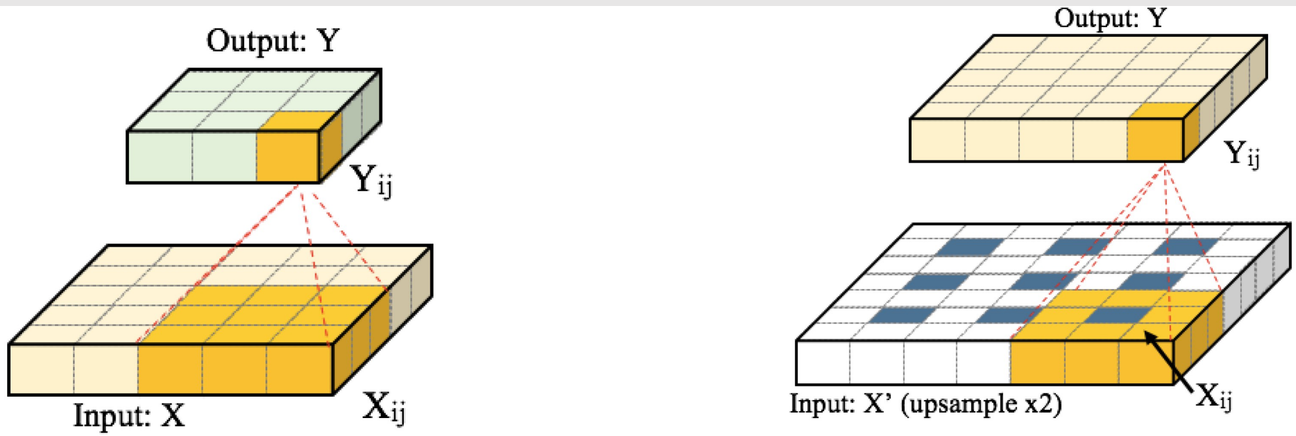
- Keep the next layer's width and height consistent with the previous
- Keep the information around the border of the image

# Strided Convolution




Shift the kernel by **2** (stride=2) when computing the next output feature.
**Objective**: to consolidate (summarise) information

# Transpose Convolution Layer




(**a**) Convolutional layer: the input size is $W_1 = H_1 = 5$; the receptive field $F = 3$; the convolution is performed with stride $S = 1$ and no padding ($P = 0$). The output Yis of size $W_2 = H_2 = 3$.

(**b**) Transposed convolutional layer: input size $W_1 = H_1 = 3$; transposed convolution with stride $S = 2$; padding with $P = 1$; and a receptive field of $F = 3$. The output Yis of size $W_2 = H_2 = 5$.

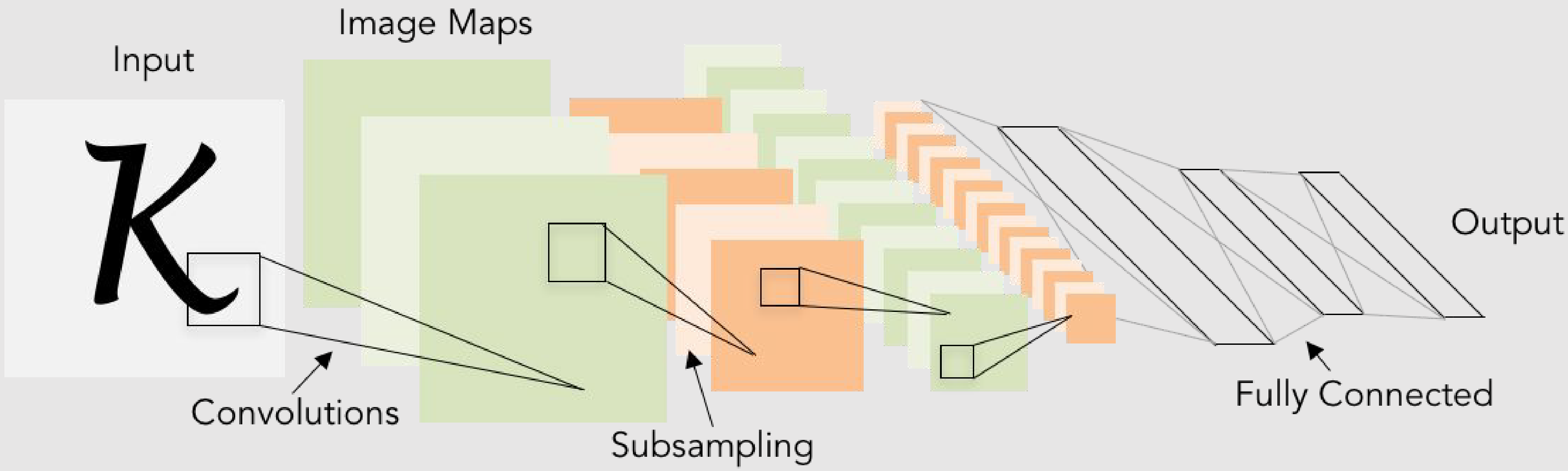
https://www.mdpi.com/2072-4292/9/6/522/htm

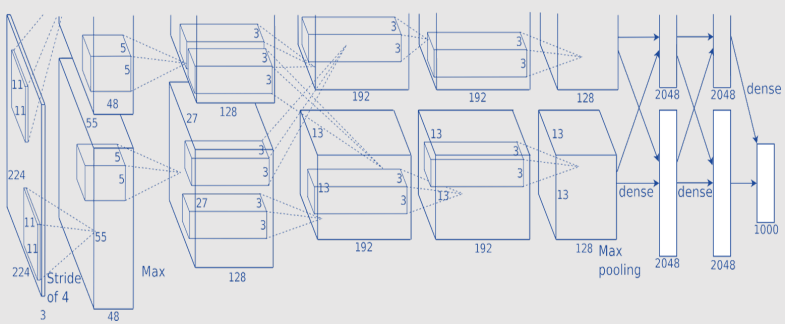
More at https://github.com/vdumoulin/conv_arithmetic

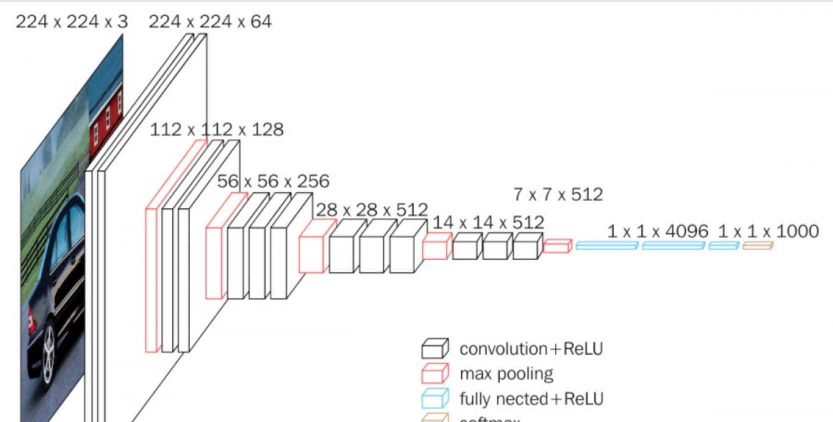
# Week 7 Contents / Objectives

- Learning with Neurons
- Neural Networks (NNs)
- NN Decision Boundary & Features
- Convolutional NN Basics
- **Convolutional NN Unboxing**

# Convolutional Neural Networks

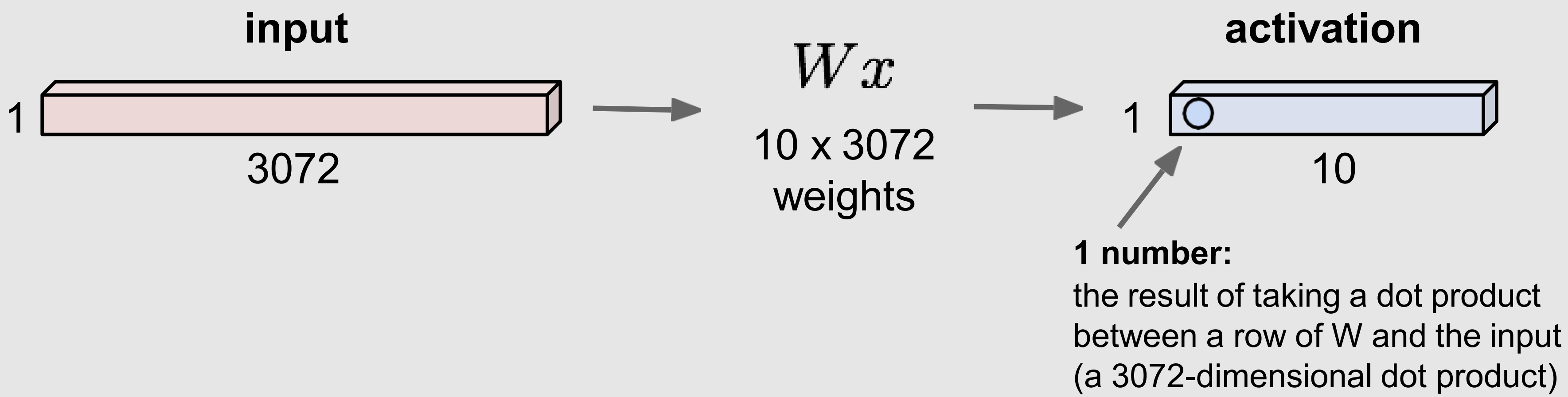


# Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

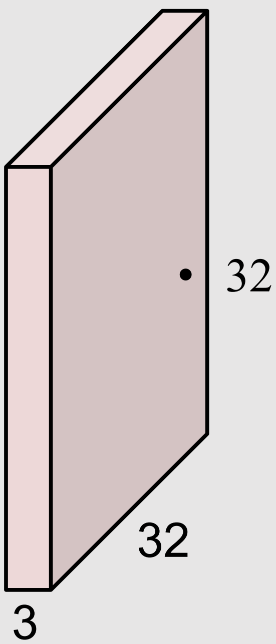


# Convolution Layer

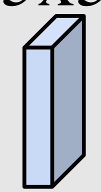
## Tensor: Preserve spatial structure

Filters always extend the full depth of the input volume

- 32x32x3 image




- 5x5x3 filter

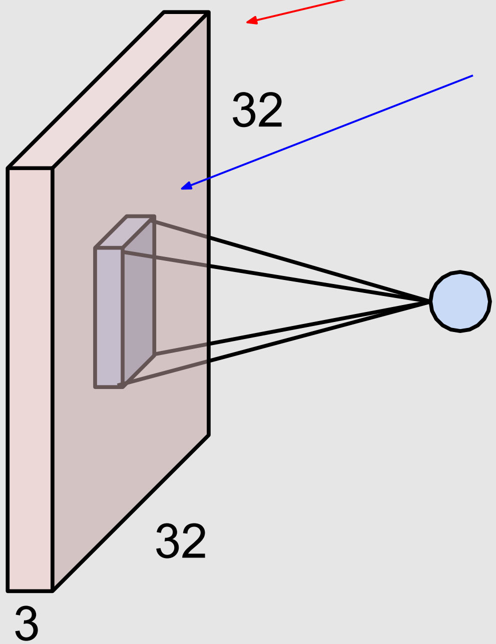


- **Convolve** the filter with the image
- i.e. "slide over the image spatially, computing dot products"

# Convolution Layer

32x32x3 image

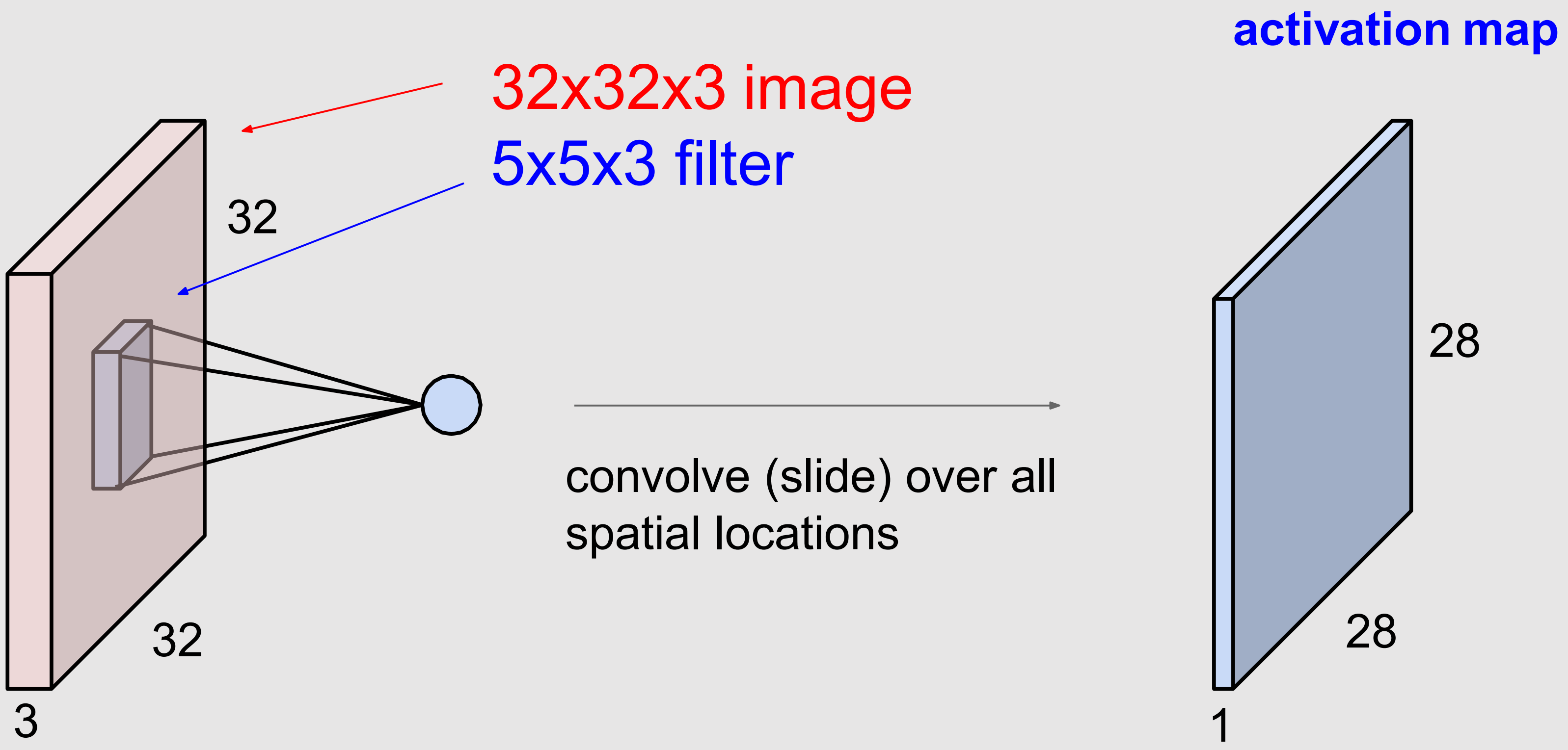
5x5x3 filter $w$




**1 number:**
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

# Convolution Layer

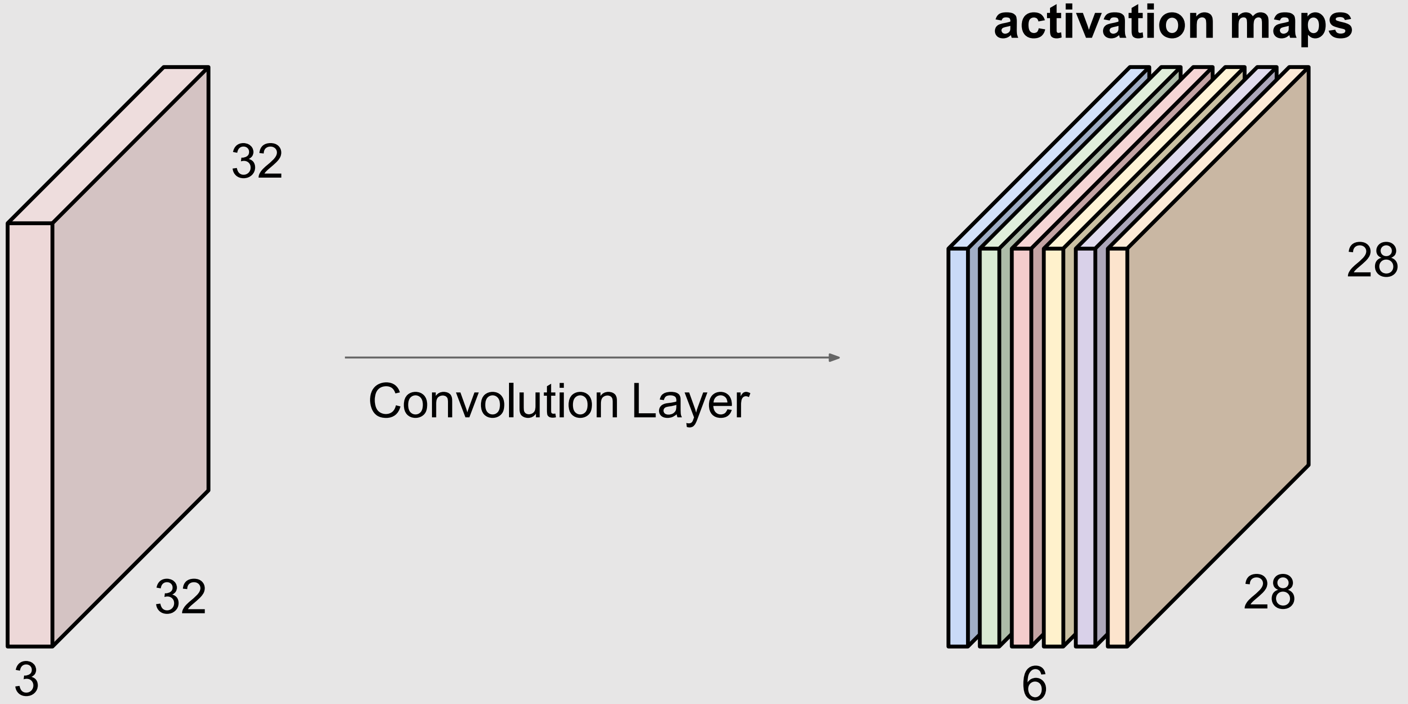


# Convolution Layer

consider a second, green filter

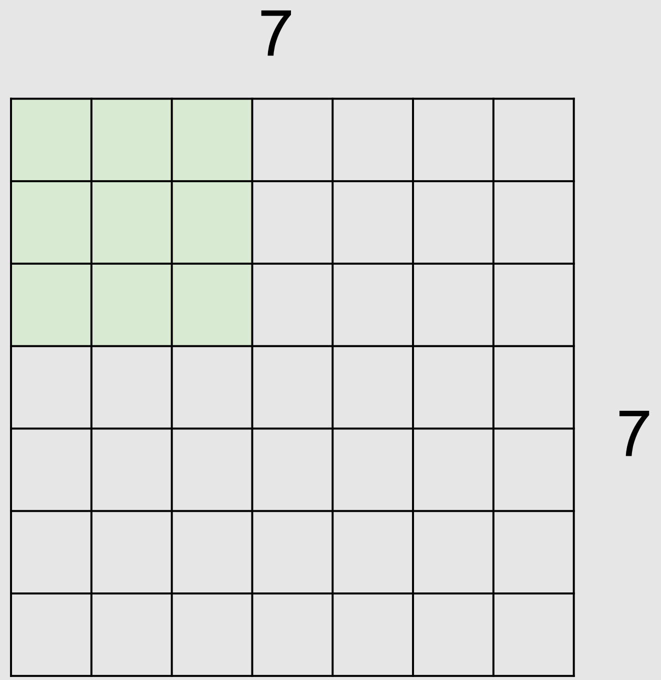
# Convolution Layer

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



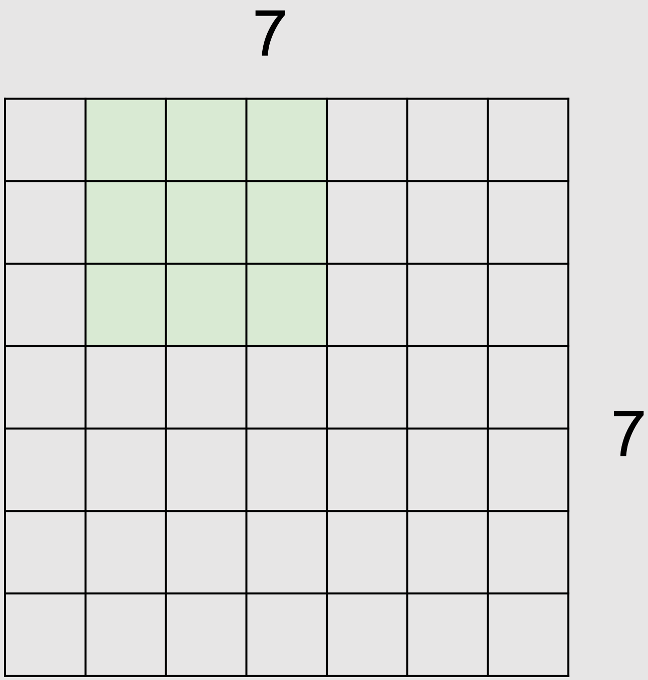

We stack these up to get a "new image" of size 28x28x6!

# Convolution Operation




7x7 input (spatially)
assume 3x3 filter

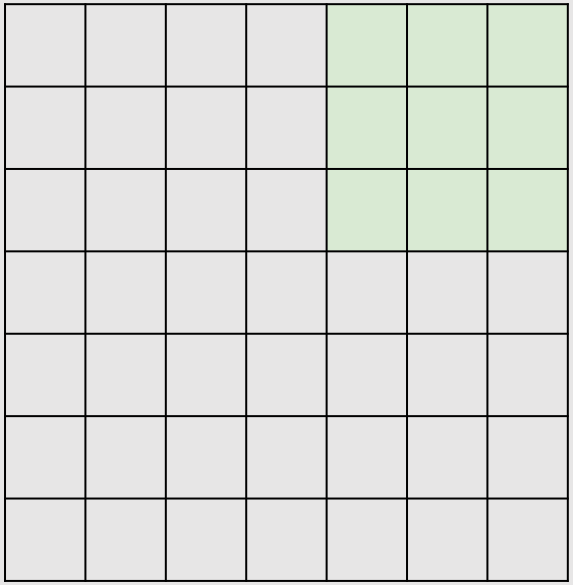
# Convolution Operation




7x7 input (spatially)
assume 3x3 filter

# Convolution Operation

7



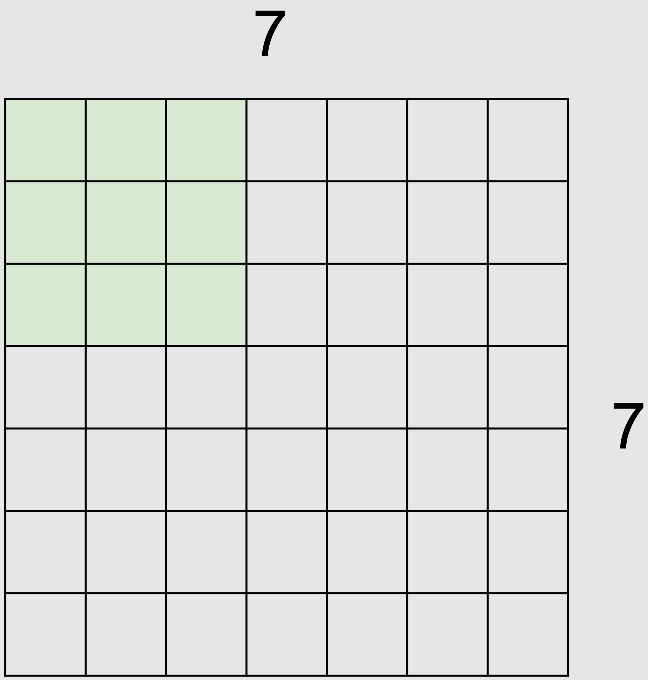
7

7x7 input (spatially)
assume 3x3 filter

→ **5x5 output**

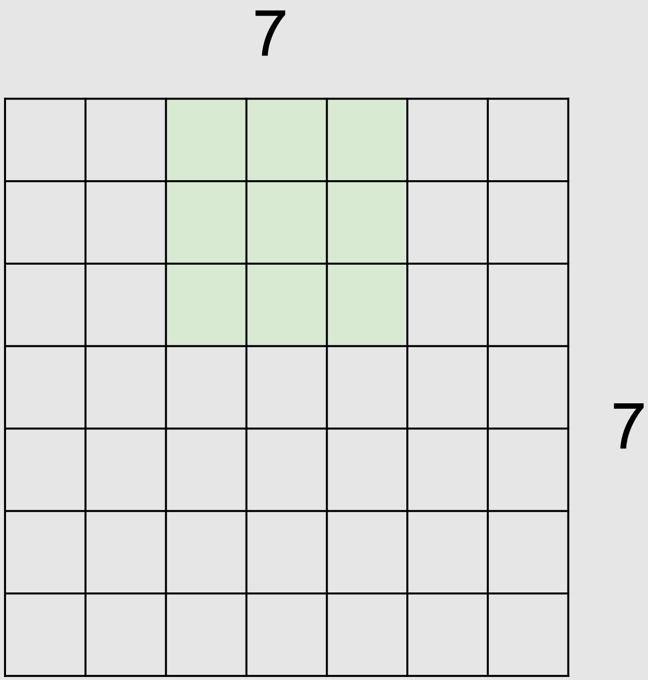
After three more sliding and dot products

# Convolution with **Stride**



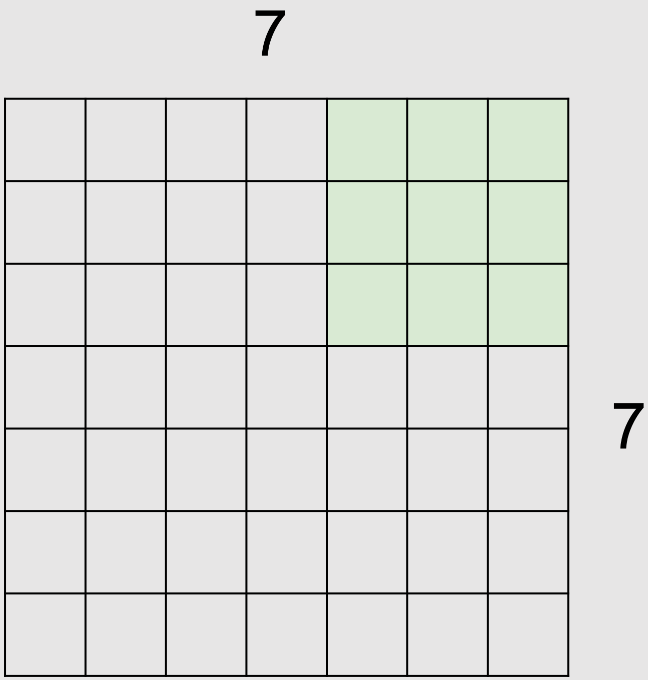

7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

# Convolution with **Stride**



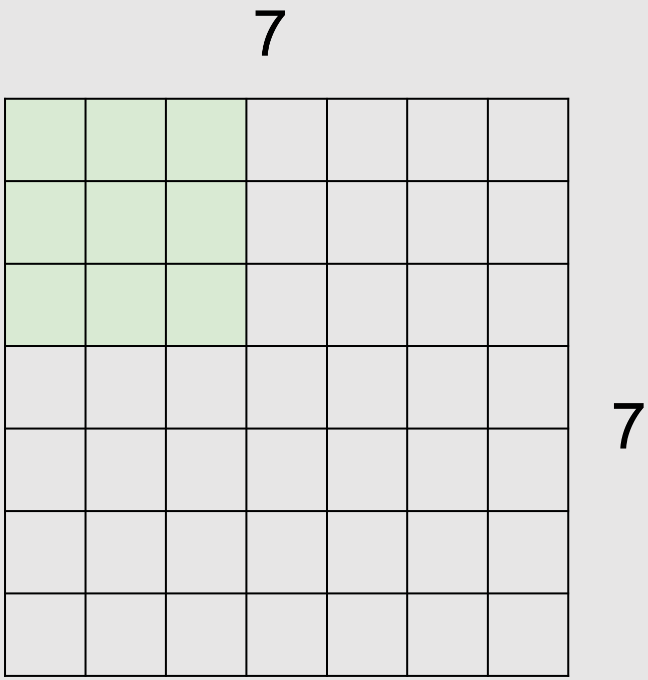

7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

# Convolution with **Stride**




7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
**=> 3x3 output!**

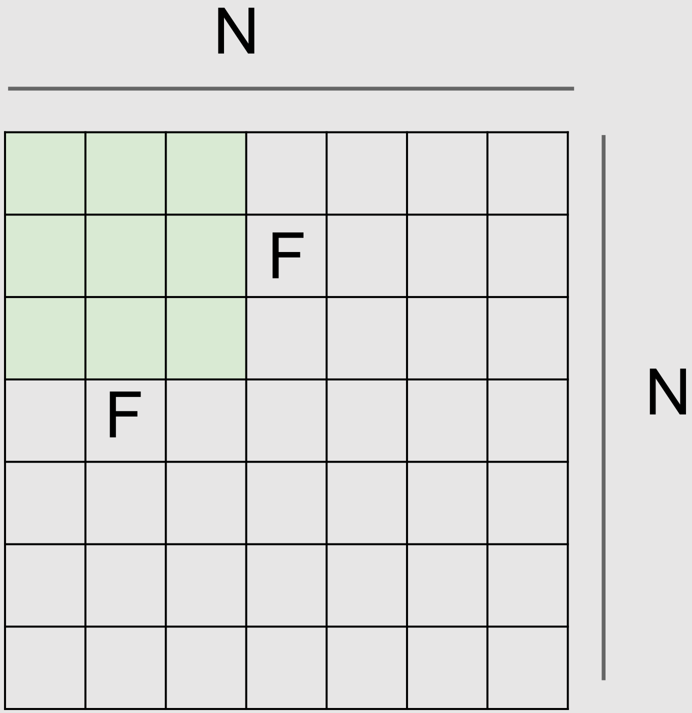
# Convolution with **Stride**




**Question**: 7x7 input (spatially) assume 3x3 filter applied **with stride 3?**

**doesn't fit!**
cannot apply 3x3 filter on 7x7 input with stride 3 (unless ignoring parts).

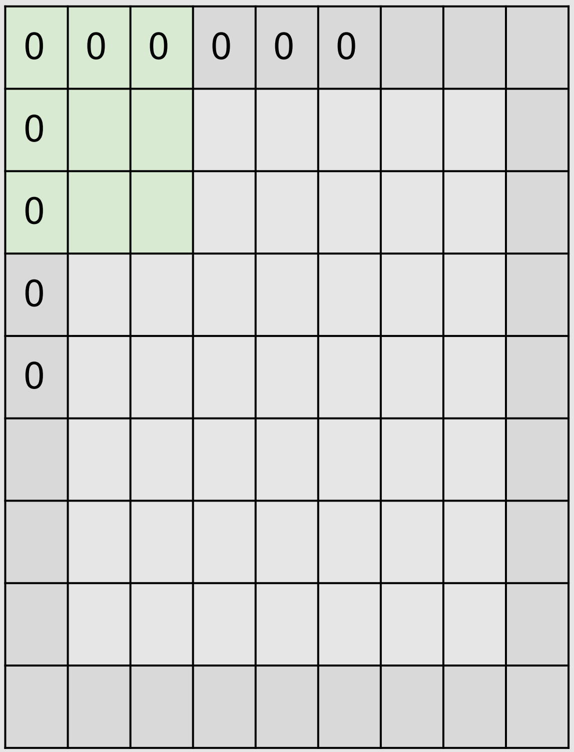
# Convolution – Size of output




Output size:
**(N - F) / stride + 1**

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33

# Zero Padding

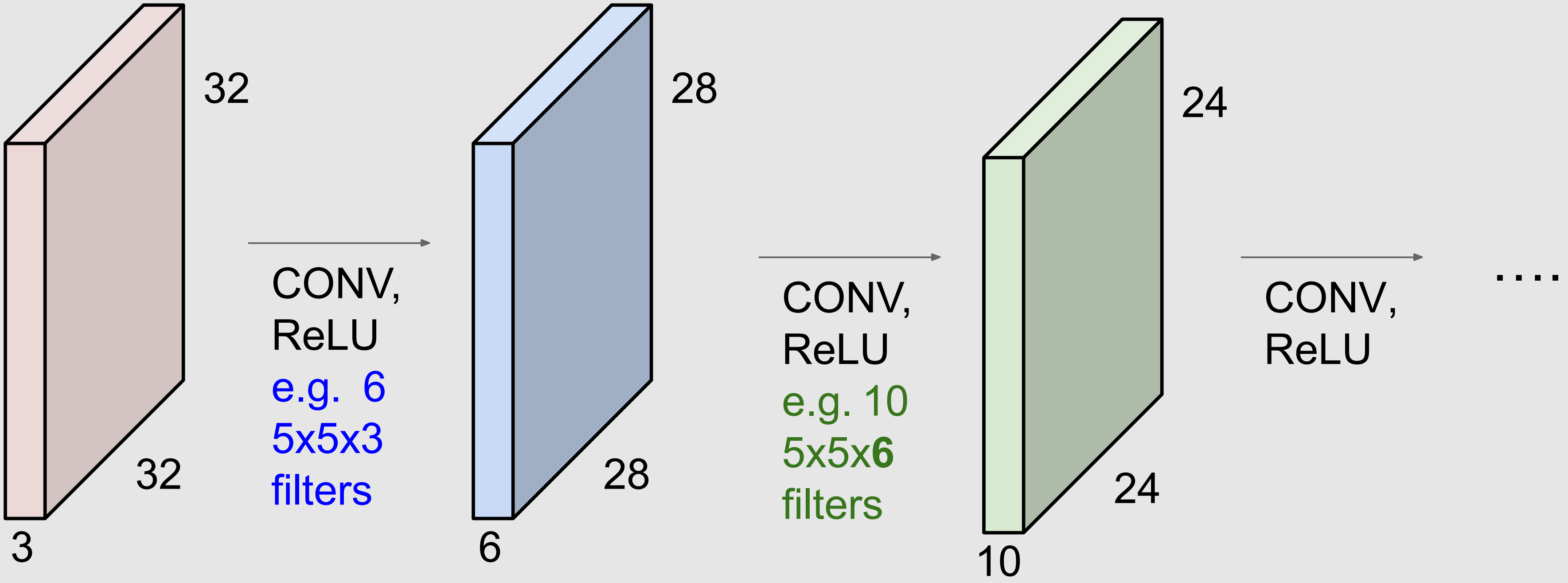



e.g. input 7x7
**3x3** filter, applied with **stride 1**
**pad with 1 pixel** border => what is the output?

**7x7 output!**
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

# Convolution Shrinks

Example: 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.

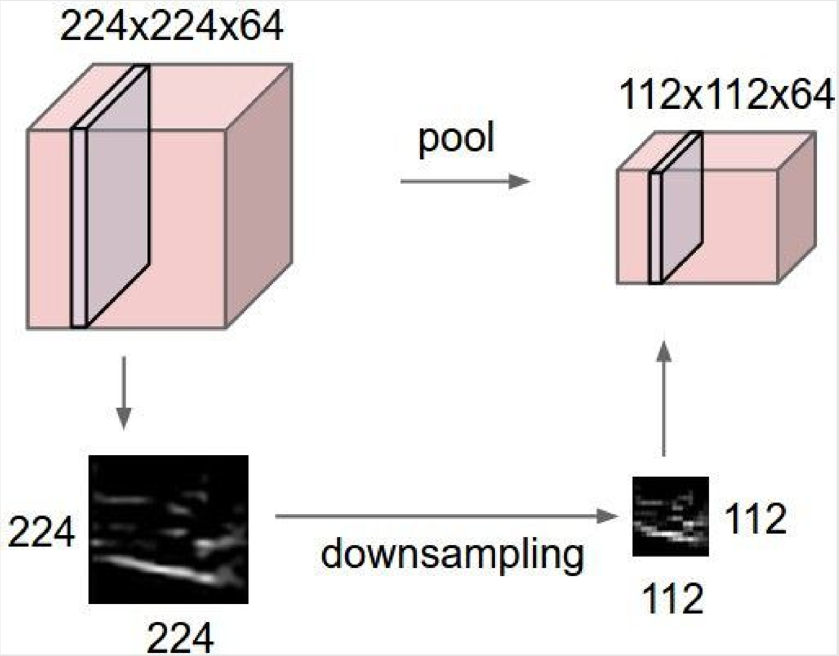


# Exercises

- Input volume: **32x32x3;** 10 5x5 filters with stride 1, pad 2
  - Output volume size?
  - 
  - Number of parameters for this layer?
  - 

# Pooling Layer: Downsampling

- Make the representations smaller and more manageable → dimensionality reduction
- Operate over each activation map independently:



# Max Pooling

Single depth slice

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

x

y

max pool with 2x2 filters
and stride 2

# Lab 7 CNN (See notebook for details)

```
__init__(self):
super(CNN, self).__init__()
self.conv1 = nn.Conv2d(3, 6, 5) #3: #
self.pool = nn.MaxPool2d(2, 2)
self.conv2 = nn.Conv2d(6, 16, 5)
self.fc1 = nn.Linear(16 * 5 * 5, 120)
self.fc2 = nn.Linear(120, 84)
self.fc3 = nn.Linear(84, 10)

forward(self, x):
x = self.pool(F.relu(self.conv1(x)))
x = self.pool(F.relu(self.conv2(x)))
x = x.view(-1, 16 * 5 * 5)
x = F.relu(self.fc1(x))
x = F.relu(self.fc2(x))
x = self.fc3(x)
return x
```

- Initial Image Size: $3 \times 32 \times 32$
- After `conv1`: $6 \times 28 \times 28$ (32
- After Pooling: $6 \times 14 \times 14$ (ima
- After `conv2`: $16 \times 10 \times 10$ (14
- After Pooling: $16 \times 5 \times 5$ (halve
- After `fc1`: $120$
- After `fc2`: $84$
- After `fc3`: $10$ (**= number of cla**

## Acknowledgement

- The slides used materials from: *Matt Gormley, Eric Xing, Nina Balcan, Fei-Fei Li & Justin Johnson & Serena Yeung, Lisa Zhang, Michael Guerzhoy, Svetlana Lazebnik, Ismini Lourentzou, Dekai Wu*

## Recommended Reading

- CS231n: Convolutional Neural Networks for Visual Recognition from Stanford (Fei-Fei Li et al.)
- The Deep Learning Book with a free official **html** version provided by the authors (Ian Goodfellow et al.)
- Convolution arithmetic
- PyTorch documentations
- The lab notebook and references

# Next

**Lab notebooks**

**Feedback (if any)**

@end of the week