

# Feature Extraction

---

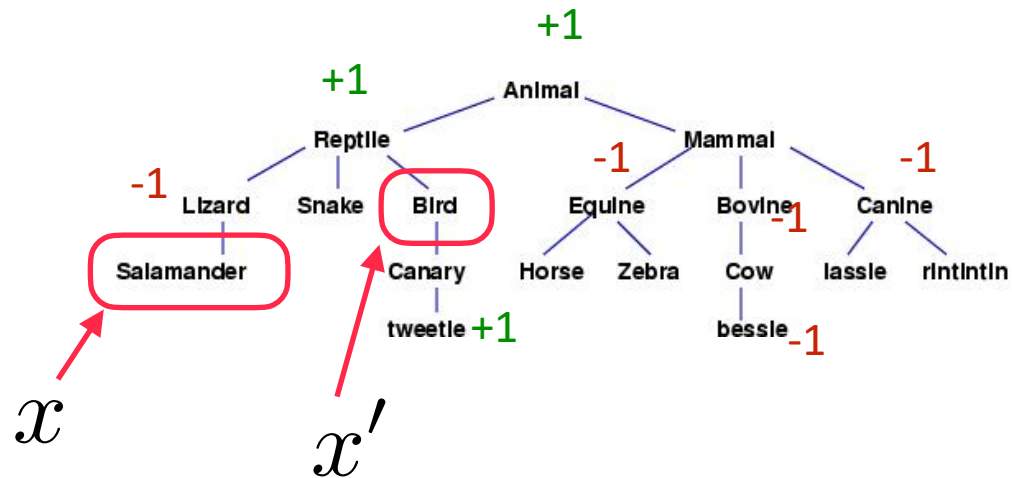
# Feature extraction

## Data comes in all forms:

Real, continuous features  $x \in \mathbb{R}^d$   $x = [0.1, 4.0, 4.3, \dots, 2.5]^\top$

Categorical data  $x = [\text{Red}, 98105, \text{Finished basement}, \dots, 2.5]^\top$

Structured data

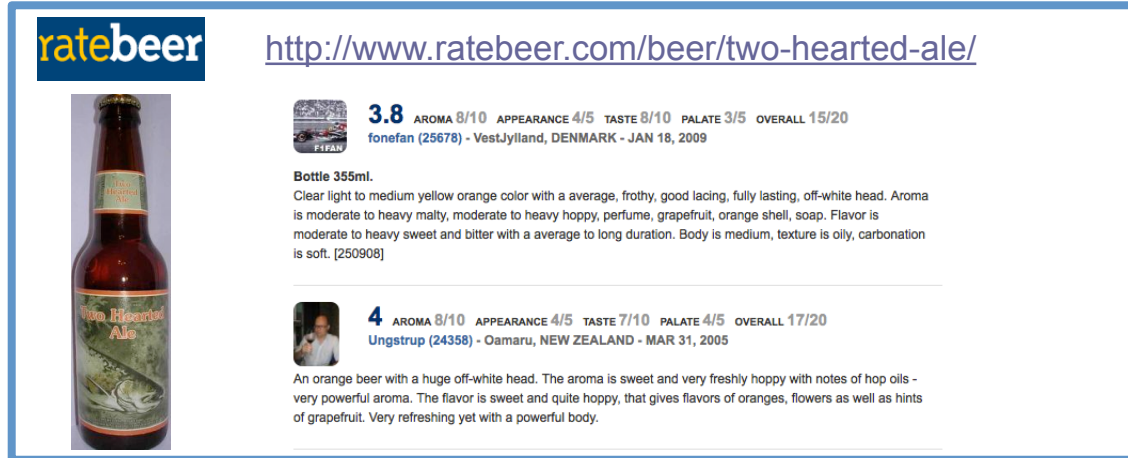


Given tree and labels are known at some nodes, how do we predict unknown labels?

# Feature extraction

Data comes in all forms:

Text data



The screenshot shows a RateBeer page for 'Two-Hearted Ale'. It features a bottle image on the left and two user reviews on the right. The first review is from 'fonefan' (25678) in Vestjylland, Denmark, dated Jan 18, 2009, with a 3.8 overall rating. The second review is from 'Ungstrup' (24358) in Oamaru, New Zealand, dated Mar 31, 2005, with a 4 overall rating. The page includes the RateBeer logo and a URL: <http://www.ratebeer.com/beer/two-hearted-ale/>

Image data



Audio data



Time-series data



# Feature Extraction given real-valued data

---

# Feature extraction - real vectors

---

Real, continuous features       $x \in \mathbb{R}^d$        $x = [0.1, 4.0, 4.3, \dots, 2.5]^\top$

Strategies if many features are **uninformative**?

# Feature extraction - real vectors

---

Real, continuous features       $x \in \mathbb{R}^d$        $x = [0.1, 4.0, 4.3, \dots, 2.5]^\top$

Strategies if many features are **superfluous** or correlated with each other?

# Feature extraction - real vectors

---

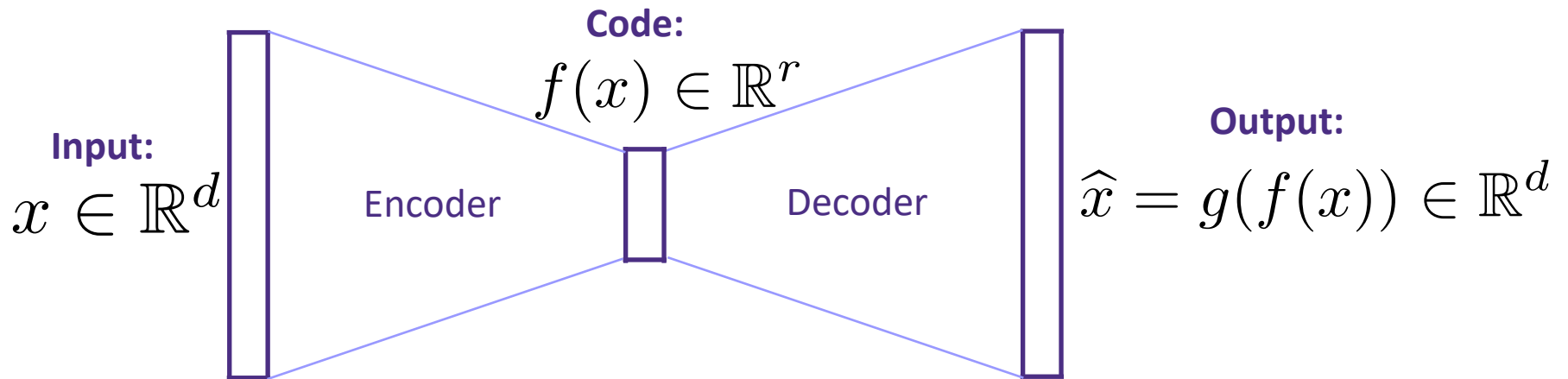
Real, continuous features       $x \in \mathbb{R}^d$        $x = [0.1, 4.0, 4.3, \dots, 2.5]^\top$

Pre-processing pipeline:

1. Standardize data (de-mean, divide by standard deviation)
2. Project down to lower dimensional representation using PCA
3. Apply exact transformation to Train and Test. How?

# Autoencoders

Find a low dimensional representation for your data by predicting your data



$$\underset{f, g}{\text{minimize}} \sum_{i=1}^n \|x_i - g(f(x_i))\|_2^2$$

# More



Self-supervised learning / Contrastive learning (later)  
Pre-training / fine-tuning (later)

# Feature Extraction given categorical data

---

# Feature extraction - categorical

---

Categorical data  $x = [\text{Red}, 98105, \text{Finished basement}, , \dots, 2.5]^T$

Many machine learning algorithms (e.g., linear predictors) require **real valued-vectors** to make predictions.

And we want those real-valued numbers to be **correlated with the label**.

# Feature extraction - categorical

---

Categorical data  $x = [\text{Red}, 98105, \text{Finished basement}, \dots, 2.5]^\top$

Many machine learning algorithms (e.g., linear predictors) require **real valued-vectors** to make predictions.

And we want those real-valued numbers to be **correlated with the label**.

One-hot encoding: Assign canonical vector to each categorical variable

$\text{color} \in \{\text{red}, \text{green}, \text{blue}\}$

# Feature extraction - categorical

---

Categorical data  $x = [\text{Red}, 98105, \text{Finished basement}, , \dots, 2.5]^\top$

Many machine learning algorithms (e.g., linear predictors) require **real valued-vectors** to make predictions.

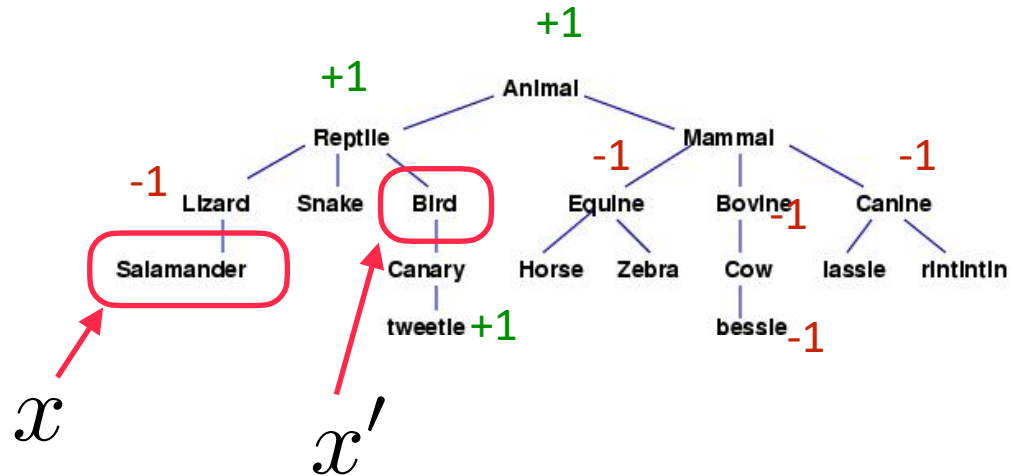
And we want those real-valued numbers to be **correlated with the label**.

Zip codes are also categorical. Is one-hot encoding appropriate?

zip code = 98105

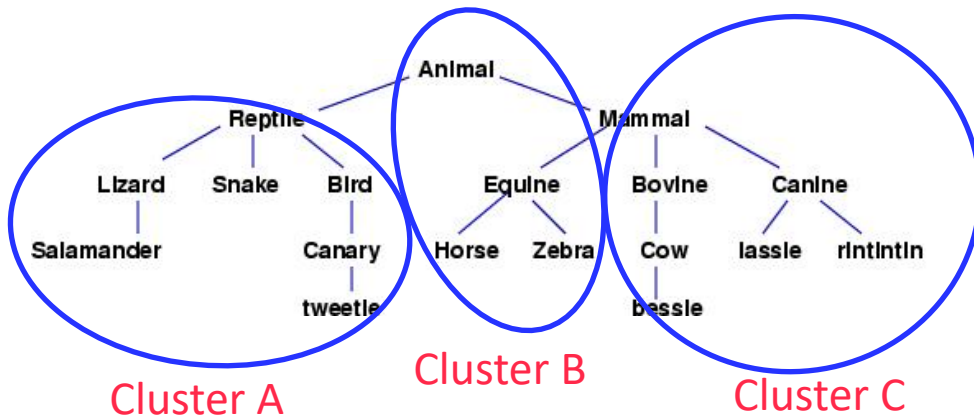
# Feature extraction - structured

Structured data



Trees define a distance between any two nodes (length of path connecting them)

Given this graph, you can compute the laplacian to compute features, or cluster



Then one-hot encode:

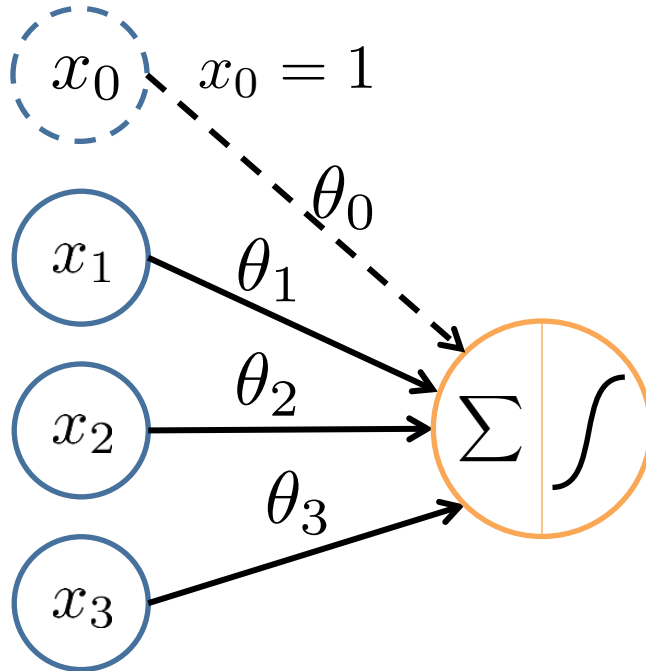
$$\text{cluster} \in \{A, B, C\}$$

# Feature extraction given Image data

---

# Single Node

“bias unit”



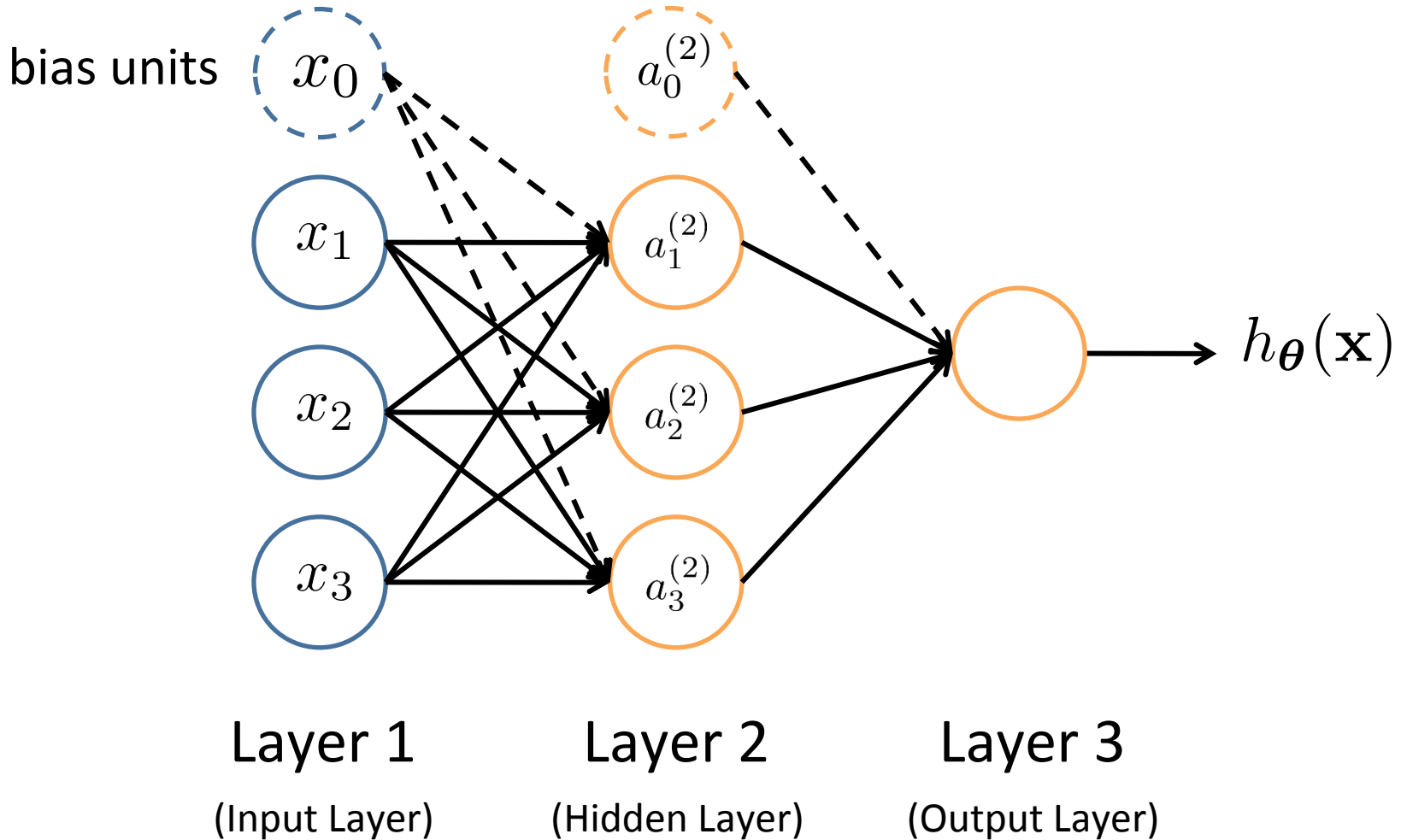
$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

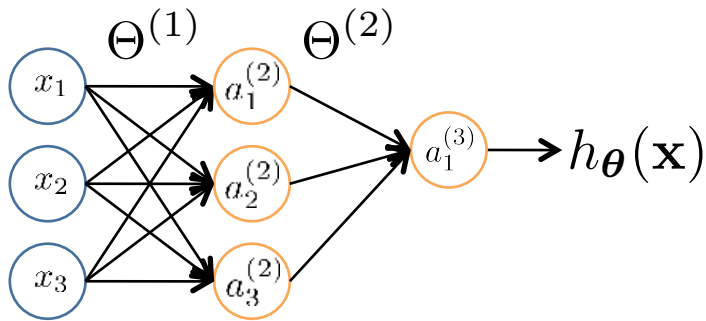
$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g(\boldsymbol{\theta}^T \mathbf{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^T \mathbf{x}}}$$

Binary  
Logistic  
Regression

Sigmoid (logistic) activation function:  $g(z) = \frac{1}{1 + e^{-z}}$

# Neural Network





$a_i^{(j)}$  = “activation” of unit  $i$  in layer  $j$   
 $\Theta^{(j)}$  = weight matrix stores parameters from layer  $j$  to layer  $j + 1$

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$

$$h_{\Theta}(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

If network has  $s_j$  units in layer  $j$  and  $s_{j+1}$  units in layer  $j+1$ , then  $\Theta^{(j)}$  has dimension  $s_{j+1} \times (s_j+1)$ .

$$\Theta^{(1)} \in \mathbb{R}^{3 \times 4} \quad \Theta^{(2)} \in \mathbb{R}^{1 \times 4}$$

# Multi-layer Neural Network

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g(z^{(2)})$$

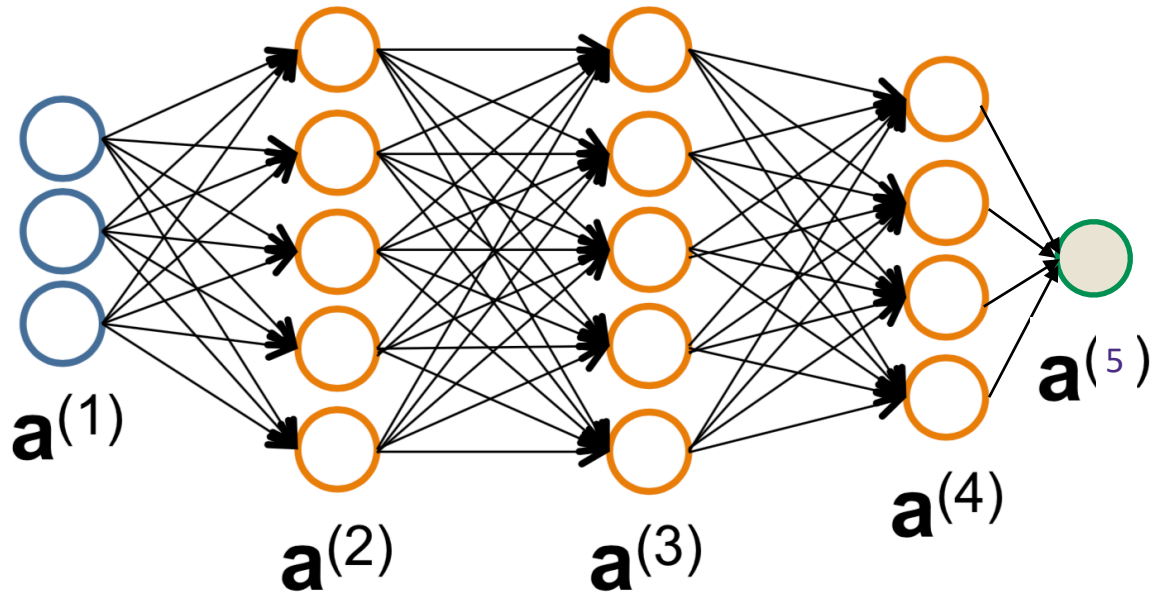
⋮

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g(z^{(l+1)})$$

⋮

$$\hat{y} = a^{(L+1)}$$



$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

Binary  
Logistic  
Regression

# Multiple Output Units: One-vs-Rest



Pedestrian



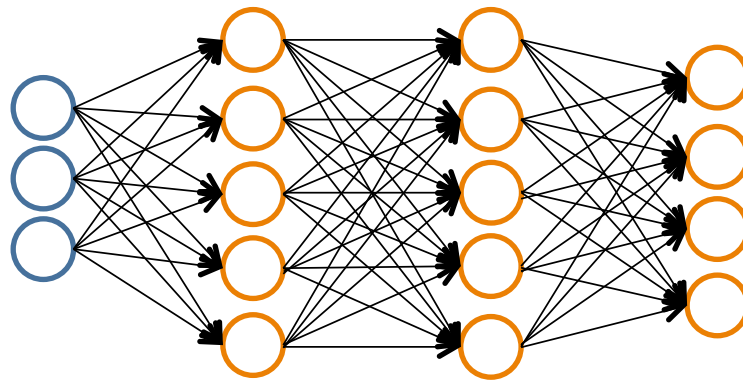
Car



Motorcycle



Truck



$$h_{\Theta}(\mathbf{x}) \in \mathbb{R}^K$$

Multi-class  
Logistic  
Regression

We want:

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

when pedestrian

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

when car

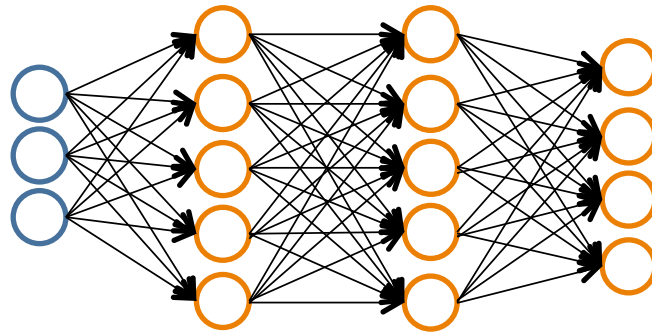
$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

when motorcycle

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

when truck

# Multiple Output Units: One-vs-Rest



$$h_{\Theta}(\mathbf{x}) \in \mathbb{R}^K$$

We want:

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

when pedestrian

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

when car

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

when motorcycle

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

when truck

- Given  $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$
- Must convert labels to 1-of- $K$  representation

– e.g.,  $y_i = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$  when motorcycle,  $y_i = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$  when car, etc.

# Neural Networks are arbitrary function approximators

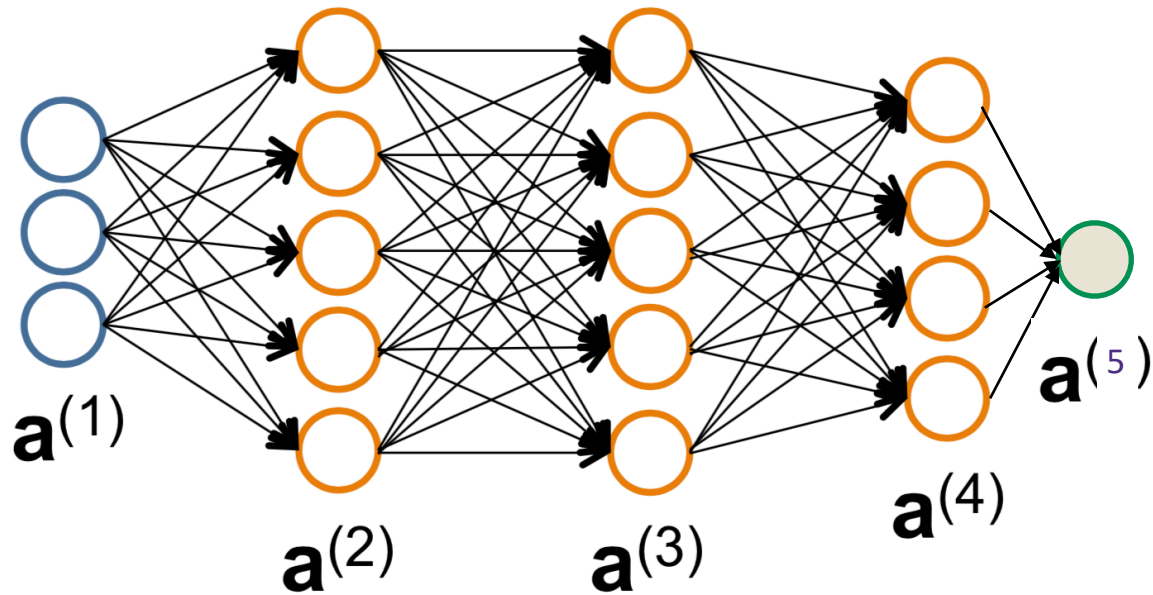
---

**Theorem 10** (Two-Layer Networks are Universal Function Approximators). *Let  $F$  be a continuous function on a bounded subset of  $D$ -dimensional space. Then there exists a two-layer neural network  $\hat{F}$  with a finite number of hidden units that approximate  $F$  arbitrarily well. Namely, for all  $\mathbf{x}$  in the domain of  $F$ ,  $|F(\mathbf{x}) - \hat{F}(\mathbf{x})| < \epsilon$ .*

Cybenko, Hornik (theorem reproduced from CIML, Ch. 10)

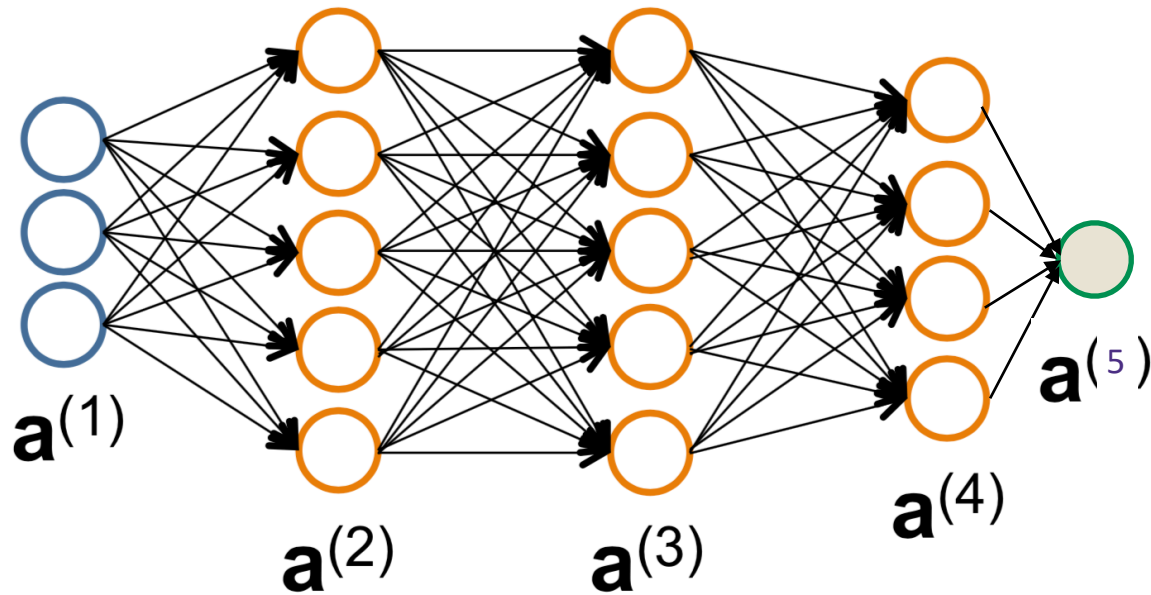
# Neural Network Architecture

The neural network architecture is defined by the number of layers, and the number of nodes in each layer, but also by **allowable edges**.



# Neural Network Architecture

The neural network architecture is defined by the number of layers, and the number of nodes in each layer, but also by **allowable edges**.



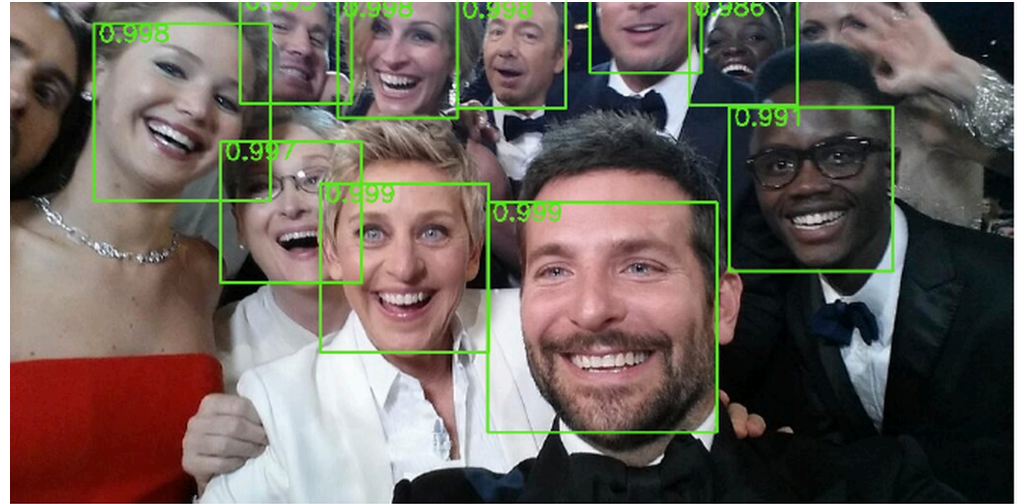
We say a layer is **Fully Connected (FC)** if all linear mappings from the current layer to the next layer are permissible.

$$\mathbf{a}^{(k+1)} = g(\Theta \mathbf{a}^{(k)}) \quad \text{for any } \Theta \in \mathbb{R}^{n_{k+1} \times n_k}$$

A lot of parameters!!  $n_1 n_2 + n_2 n_3 + \cdots + n_L n_{L+1}$

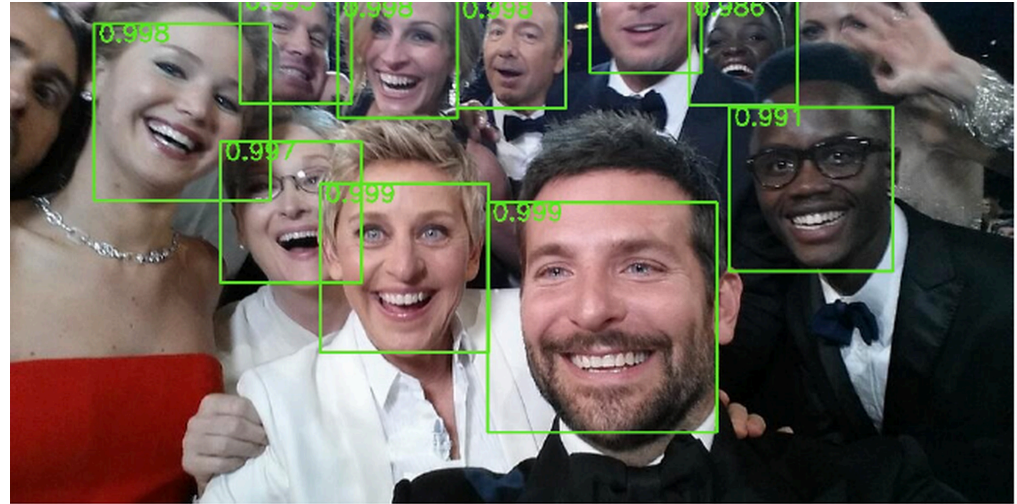
# Neural Network Architecture

Objects are often **localized in space** so to find the faces in an image, not every pixel is important for classification—makes sense to drag a window across an image.

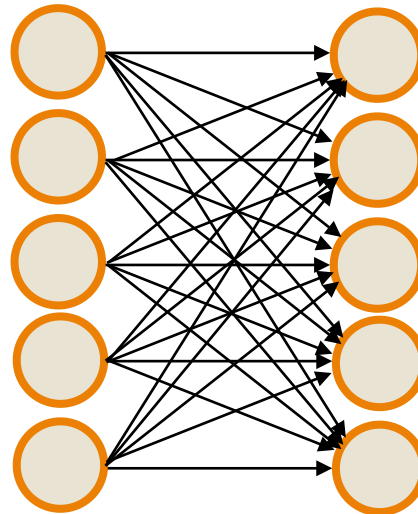


# Neural Network Architecture

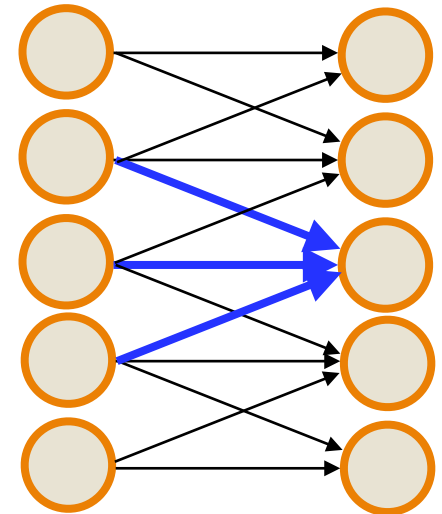
Objects are often **localized in space** so to find the faces in an image, not every pixel is important for classification—makes sense to drag a window across an image.



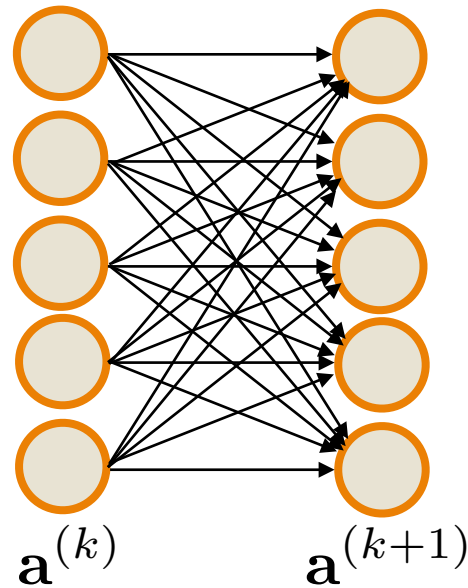
Similarly, to identify edges or other local structure, it makes sense to only look at **local information**



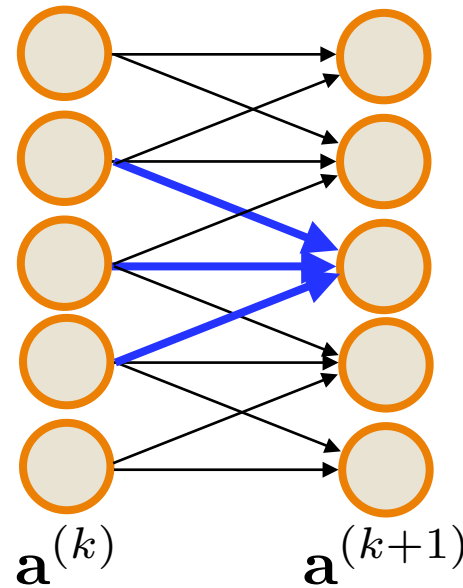
vs.



# Neural Network Architecture



vs.



$$\begin{bmatrix} \Theta_{0,0} & \Theta_{0,1} & \Theta_{0,2} & \Theta_{0,3} & \Theta_{0,4} \\ \Theta_{1,0} & \Theta_{1,1} & \Theta_{1,2} & \Theta_{1,3} & \Theta_{1,4} \\ \Theta_{2,0} & \Theta_{2,1} & \Theta_{2,2} & \Theta_{2,3} & \Theta_{2,4} \\ \Theta_{3,0} & \Theta_{3,1} & \Theta_{3,2} & \Theta_{3,3} & \Theta_{3,4} \\ \Theta_{4,0} & \Theta_{4,1} & \Theta_{4,2} & \Theta_{4,3} & \Theta_{4,4} \end{bmatrix}$$

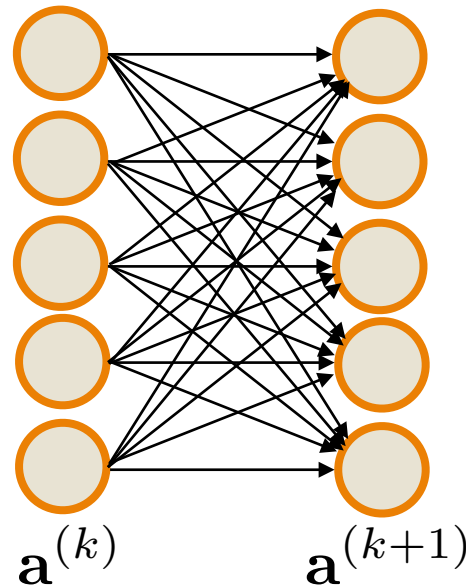
$$\begin{bmatrix} \Theta_{0,0} & \Theta_{0,1} & 0 & 0 & 0 \\ \Theta_{1,0} & \Theta_{1,1} & \Theta_{1,2} & 0 & 0 \\ 0 & \Theta_{2,1} & \Theta_{2,2} & \Theta_{2,3} & 0 \\ 0 & 0 & \Theta_{3,2} & \Theta_{3,3} & \Theta_{3,4} \\ 0 & 0 & 0 & \Theta_{4,3} & \Theta_{4,4} \end{bmatrix}$$

Parameters:  $n^2$

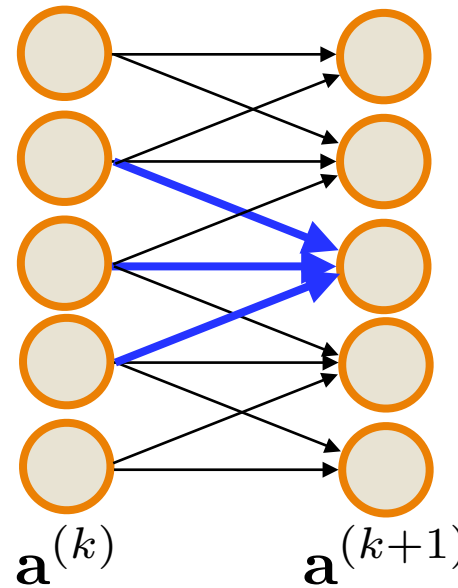
$3n - 2$

$$\mathbf{a}_i^{(k+1)} = g \left( \sum_{j=0}^{n-1} \Theta_{i,j} \mathbf{a}_j^{(k)} \right)$$

# Neural Network Architecture



vs.



Mirror/share local weights everywhere (e.g., structure equally likely to be anywhere in image)

$$\begin{bmatrix} \Theta_{0,0} & \Theta_{0,1} & \Theta_{0,2} & \Theta_{0,3} & \Theta_{0,4} \\ \Theta_{1,0} & \Theta_{1,1} & \Theta_{1,2} & \Theta_{1,3} & \Theta_{1,4} \\ \Theta_{2,0} & \Theta_{2,1} & \Theta_{2,2} & \Theta_{2,3} & \Theta_{2,4} \\ \Theta_{3,0} & \Theta_{3,1} & \Theta_{3,2} & \Theta_{3,3} & \Theta_{3,4} \\ \Theta_{4,0} & \Theta_{4,1} & \Theta_{4,2} & \Theta_{4,3} & \Theta_{4,4} \end{bmatrix}$$

Parameters:  $n^2$

$$\begin{bmatrix} \Theta_{0,0} & \Theta_{0,1} & 0 & 0 & 0 \\ \Theta_{1,0} & \Theta_{1,1} & \Theta_{1,2} & 0 & 0 \\ 0 & \Theta_{2,1} & \Theta_{2,2} & \Theta_{2,3} & 0 \\ 0 & 0 & \Theta_{3,2} & \Theta_{3,3} & \Theta_{3,4} \\ 0 & 0 & 0 & \Theta_{4,3} & \Theta_{4,4} \end{bmatrix}$$

$3n - 2$

$$\begin{bmatrix} \theta_1 & \theta_2 & 0 & 0 & 0 \\ \theta_0 & \theta_1 & \theta_2 & 0 & 0 \\ 0 & \theta_0 & \theta_1 & \theta_2 & 0 \\ 0 & 0 & \theta_0 & \theta_1 & \theta_2 \\ 0 & 0 & 0 & \theta_0 & \theta_1 \end{bmatrix}$$

3

$$\mathbf{a}_i^{(k+1)} = g \left( \sum_{j=0}^{n-1} \Theta_{i,j} \mathbf{a}_j^{(k)} \right)$$

$$\mathbf{a}_i^{(k+1)} = g \left( \sum_{j=0}^{m-1} \theta_j \mathbf{a}_{i+j}^{(k)} \right)$$

# Neural Network Architecture

## Fully Connected (FC) Layer

$$\begin{bmatrix} \Theta_{0,0} & \Theta_{0,1} & \Theta_{0,2} & \Theta_{0,3} & \Theta_{0,4} \\ \Theta_{1,0} & \Theta_{1,1} & \Theta_{1,2} & \Theta_{1,3} & \Theta_{1,4} \\ \Theta_{2,0} & \Theta_{2,1} & \Theta_{2,2} & \Theta_{2,3} & \Theta_{2,4} \\ \Theta_{3,0} & \Theta_{3,1} & \Theta_{3,2} & \Theta_{3,3} & \Theta_{3,4} \\ \Theta_{4,0} & \Theta_{4,1} & \Theta_{4,2} & \Theta_{4,3} & \Theta_{4,4} \end{bmatrix}$$

## Convolutional (CONV) Layer (1 filter)

$$\begin{bmatrix} \theta_1 & \theta_2 & 0 & 0 & 0 \\ \theta_0 & \theta_1 & \theta_2 & 0 & 0 \\ 0 & \theta_0 & \theta_1 & \theta_2 & 0 \\ 0 & 0 & \theta_0 & \theta_1 & \theta_2 \\ 0 & 0 & 0 & \theta_0 & \theta_1 \end{bmatrix} \quad m=3$$

$$\mathbf{a}_i^{(k+1)} = g \left( \sum_{j=0}^{n-1} \Theta_{i,j} \mathbf{a}_j^{(k)} \right)$$

$$\mathbf{a}_i^{(k+1)} = g \left( \sum_{j=0}^{m-1} \theta_j \mathbf{a}_{i+j}^{(k)} \right) = g([\theta * \mathbf{a}^{(k)}]_i)$$

Convolution\*

$\theta = (\theta_0, \dots, \theta_{m-1}) \in \mathbb{R}^m$  is referred to as a “filter”

\* Actually defined as the closely related quantity of “cross-correlation” but the deep learning literature just calls this “convolution”

# Example (1d convolution)

$$(\theta * x)_i = \sum_{j=0}^{m-1} \theta_j x_{i+j}$$

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

Input  $x \in \mathbb{R}^n$

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

Filter  $\theta \in \mathbb{R}^m$

--	--	--

Output  $\theta * x$

# Example (1d convolution)

$$(\theta * x)_i = \sum_{j=0}^{m-1} \theta_j x_{i+j}$$

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

Input  $x \in \mathbb{R}^n$

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

Filter  $\theta \in \mathbb{R}^m$

1 <sub>x1</sub>	1 <sub>x0</sub>	1 <sub>x1</sub>	0	0
-----------------	-----------------	-----------------	---	---

2		
---	--	--

Output  $\theta * x$

# Example (1d convolution)

$$(\theta * x)_i = \sum_{j=0}^{m-1} \theta_j x_{i+j}$$

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

Input  $x \in \mathbb{R}^n$

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

Filter  $\theta \in \mathbb{R}^m$

1	1 <sub>x1</sub>	1 <sub>x0</sub>	0 <sub>x1</sub>	0
---	-----------------	-----------------	-----------------	---

2	1	
---	---	--

Output  $\theta * x$

# Example (1d convolution)

$$(\theta * x)_i = \sum_{j=0}^{m-1} \theta_j x_{i+j}$$

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

Input  $x \in \mathbb{R}^n$

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

Filter  $\theta \in \mathbb{R}^m$

1	1	1 <sub>x1</sub>	0 <sub>x0</sub>	0 <sub>x1</sub>
---	---	-----------------	-----------------	-----------------

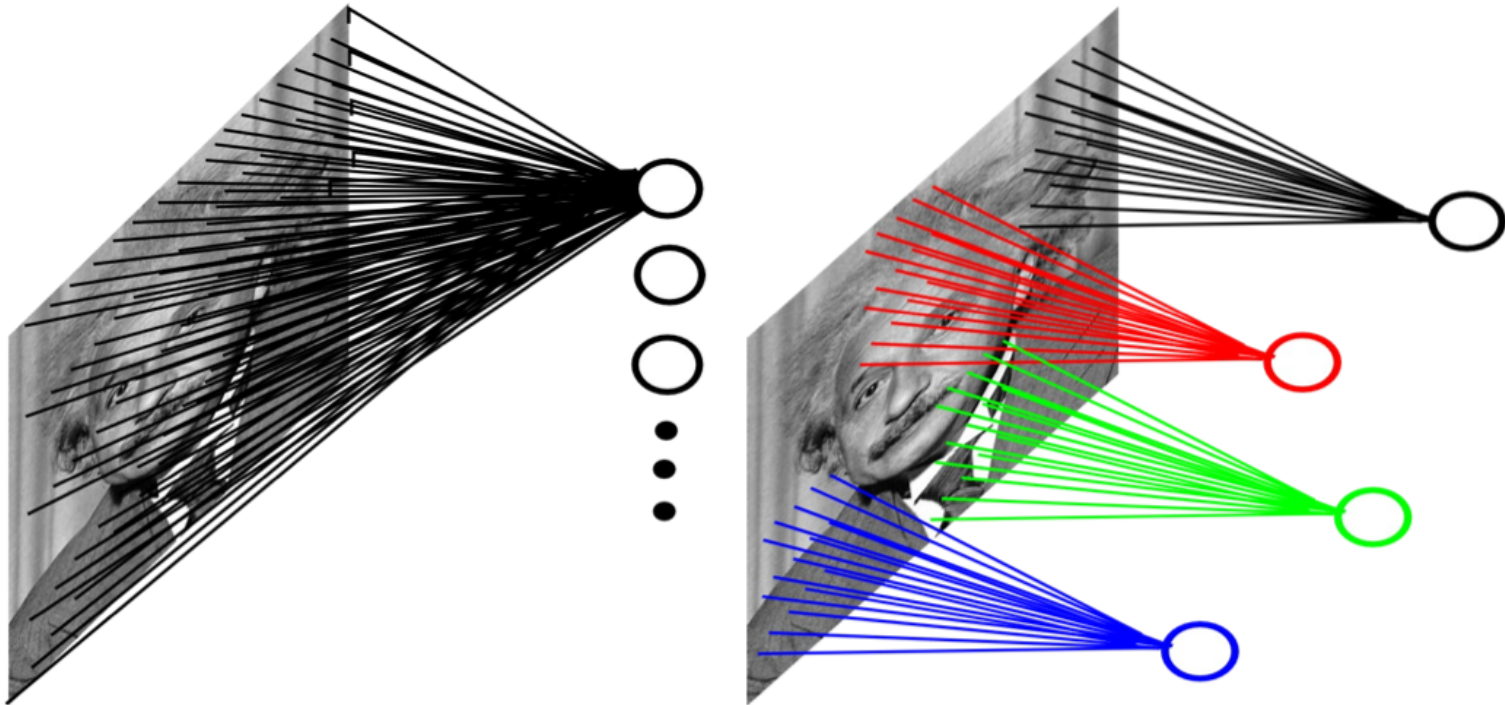
2	1	1
---	---	---

Output  $\theta * x$

# 2d Convolution Layer

## ■ Example: 200x200 image

- ▶ Fully-connected, 400,000 hidden units = 16 billion parameters
- ▶ Locally-connected, 400,000 hidden units 10x10 fields = 40 million params
- ▶ Local connections capture local dependencies



# Convolution of images (2d convolution)

$$(I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n)$$

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

Image  $I$

1	0	1
0	1	0
1	0	1

Filter  $K$

1 <sub>x1</sub>	1 <sub>x0</sub>	1 <sub>x1</sub>	0	0
0 <sub>x0</sub>	1 <sub>x1</sub>	1 <sub>x0</sub>	1	0
0 <sub>x1</sub>	0 <sub>x0</sub>	1 <sub>x1</sub>	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved  
Feature

$$I * K$$

# Convolution of images

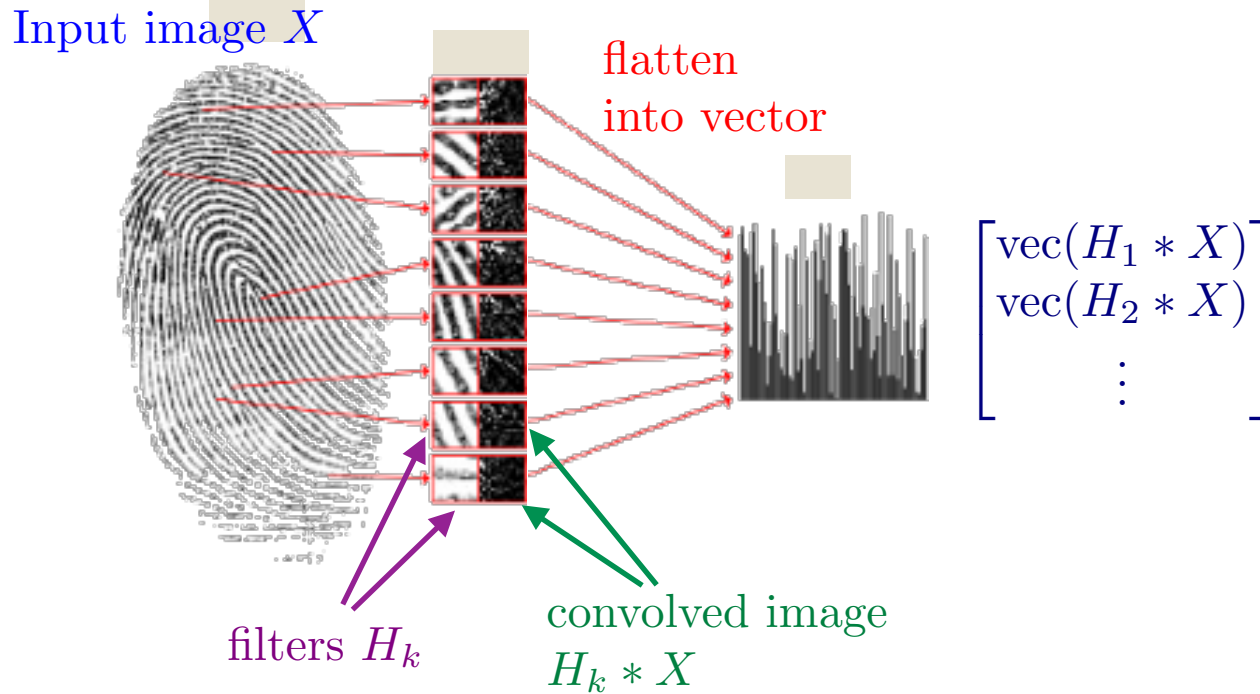
$$(I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n)$$

Image  $I$

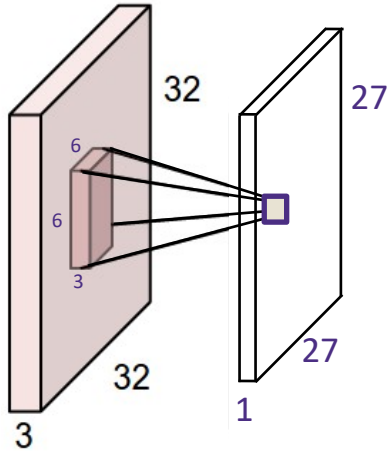


Operation	Filter $K$	Convolved Image $I * K$
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

# Convolution of images



# 3d Convolution

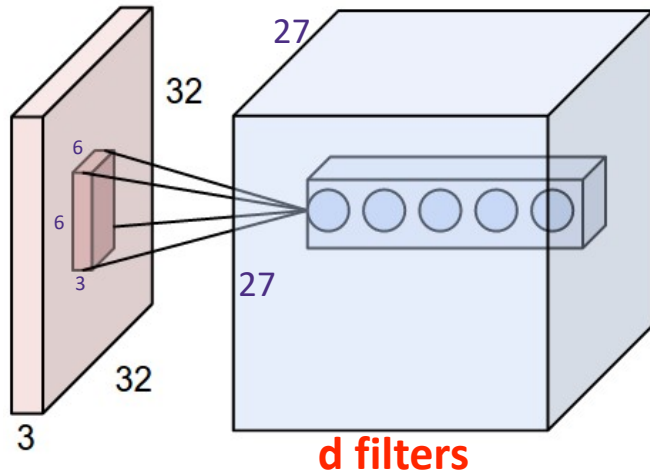


$$\Theta \in \mathbb{R}^{m \times m \times r}$$

$$x \in \mathbb{R}^{n \times n \times r}$$

$$(\Theta * x)_{s,t} = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} \sum_{k=0}^{r-1} \Theta_{i,j,k} x_{s+i,t+j}$$

# Stacking convolved images



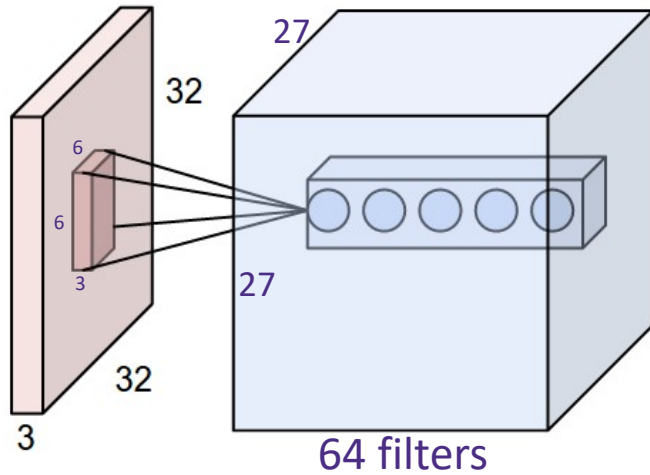
**Repeat with d filters!**

$$\Theta \in \mathbb{R}^{m \times m \times r}$$

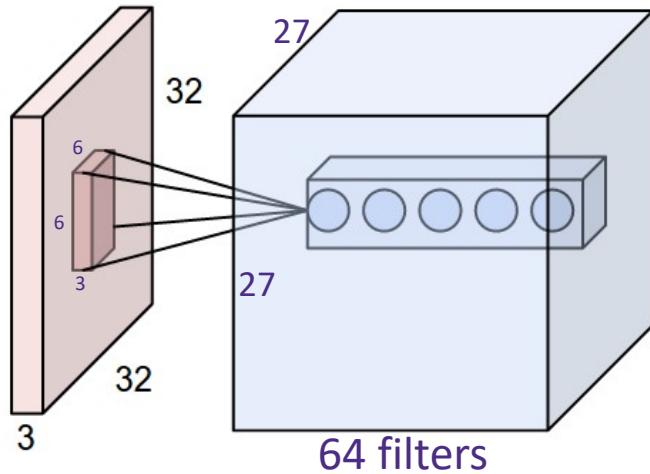
$$x \in \mathbb{R}^{n \times n \times r}$$

$$(\Theta * x)_{s,t} = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} \sum_{k=0}^{r-1} \Theta_{i,j,k} x_{s+i,t+j}$$

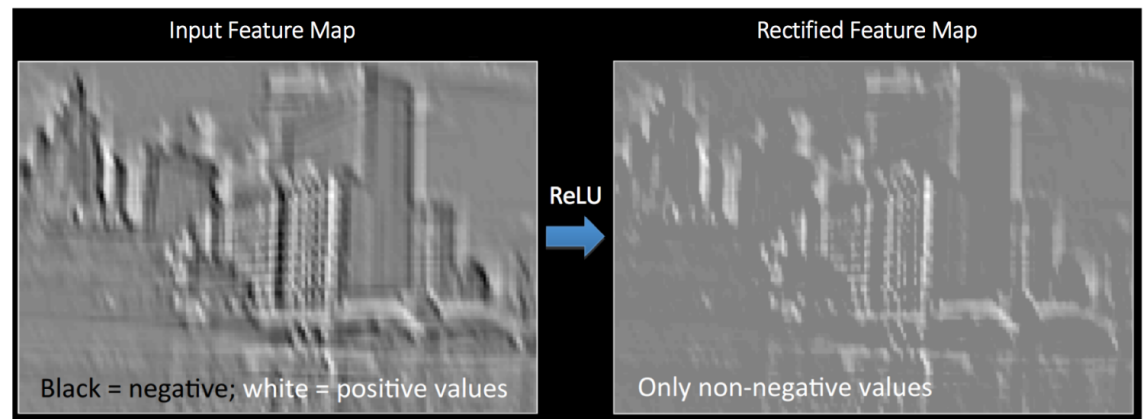
# Stacking convolved images



# Stacking convolved images



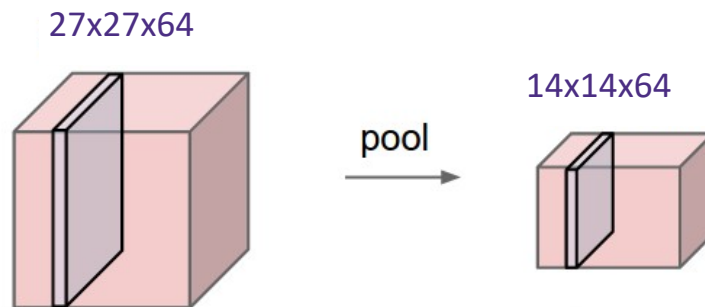
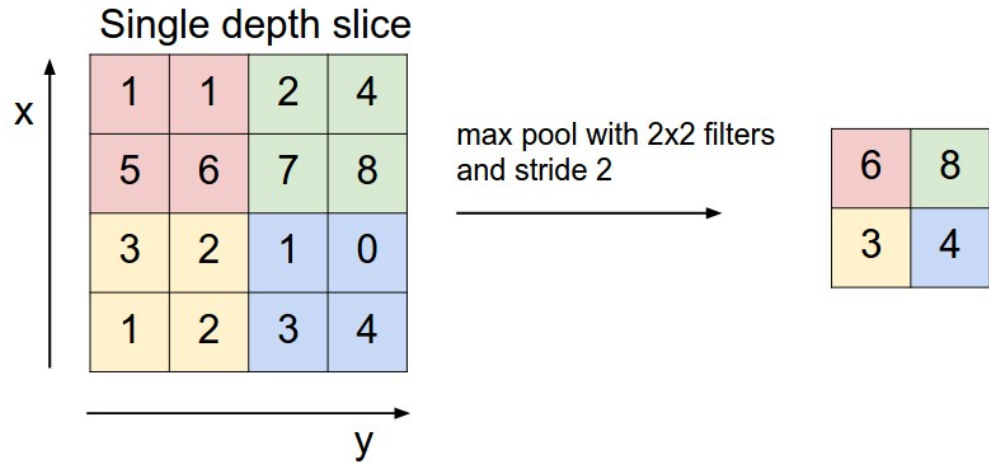
Apply Non-linearity to the output of each layer, Here: ReLu (rectified linear unit)



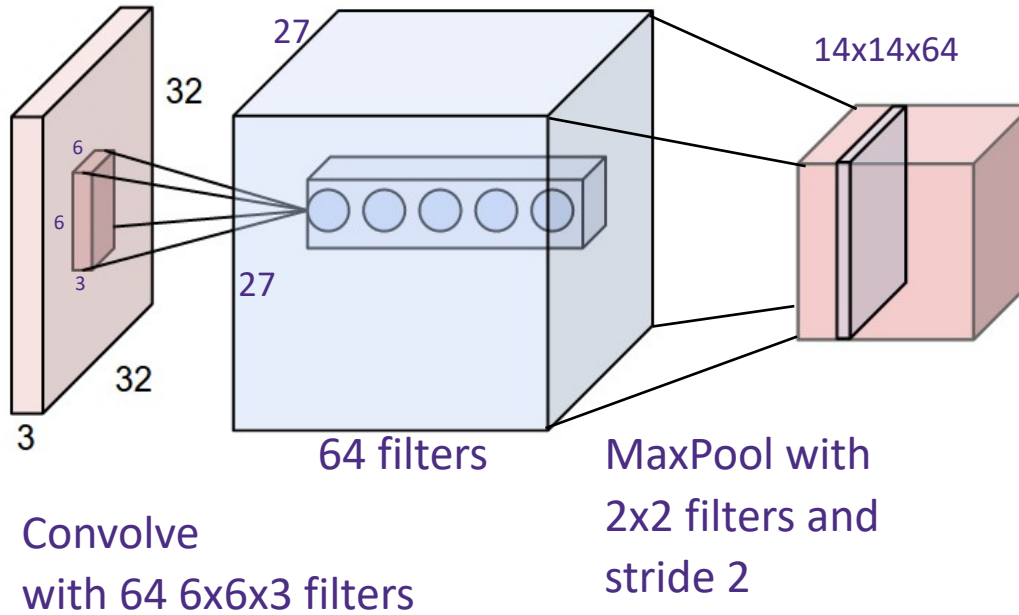
Other choices: sigmoid, arctan

# Pooling

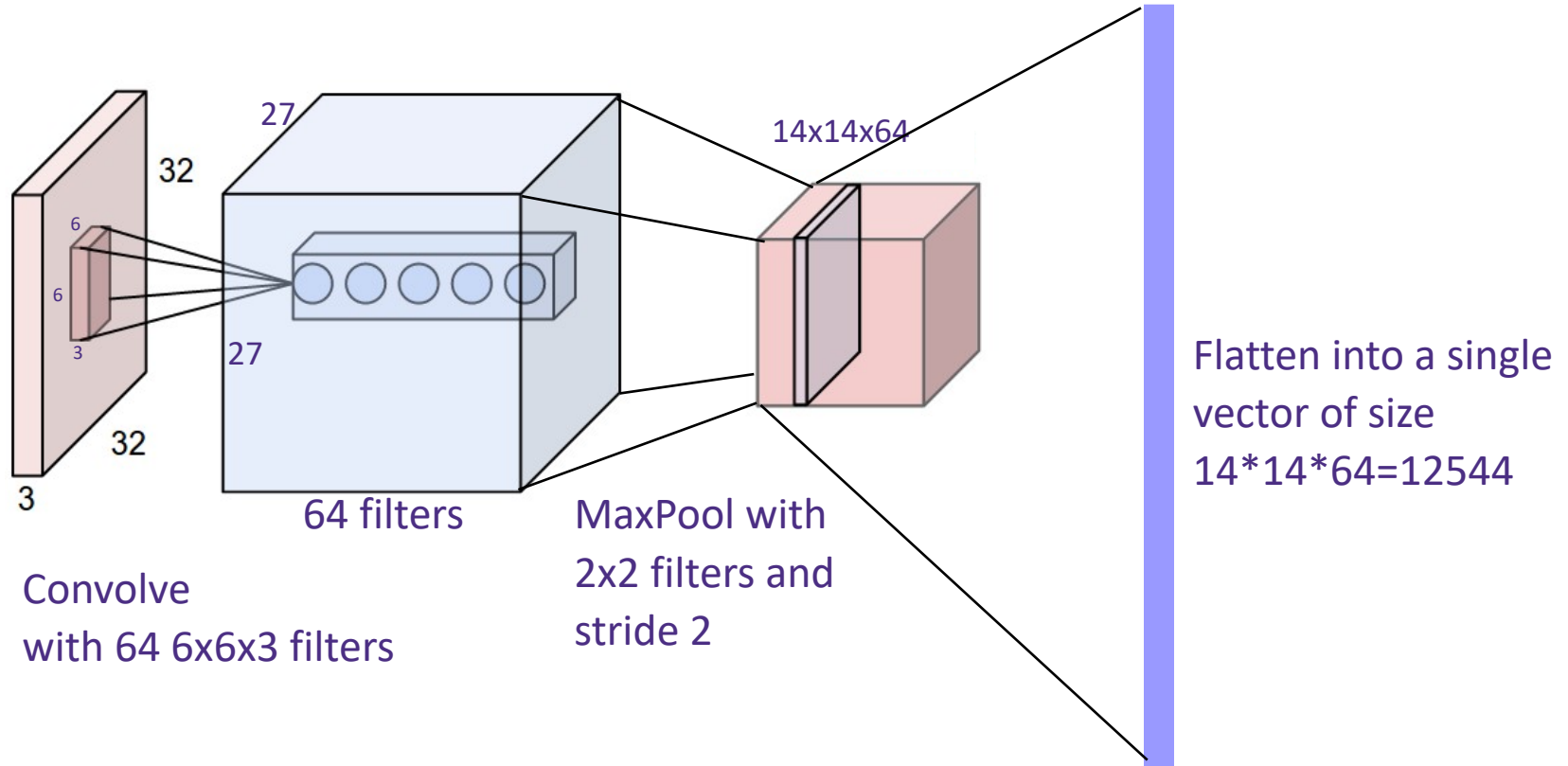
Pooling reduces the dimension and can be interpreted as “This filter had a high response in this general region”



# Pooling Convolution layer



# Simplest feature pipeline

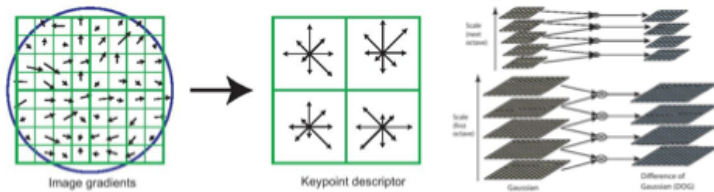


How do we choose all the hyperparameters?

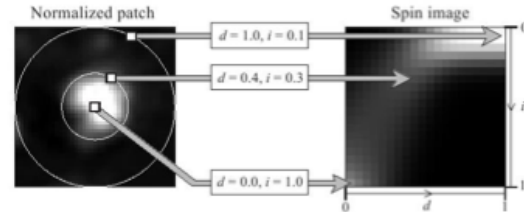
How do we choose the filters?

- Hand crafted (digital signal processing, c.f. wavelets)
- Learn them (deep learning)

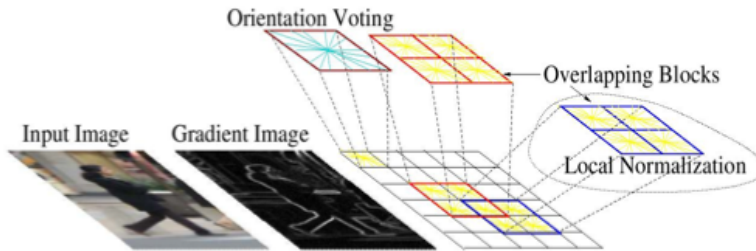
# Some hand-created image features



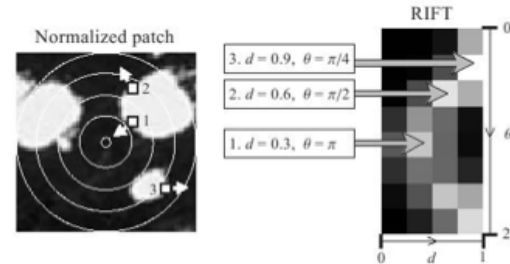
SIFT



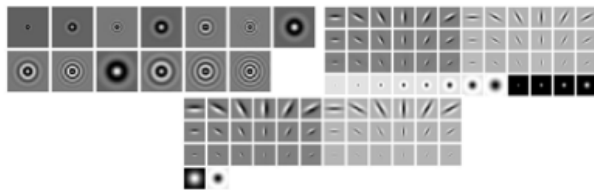
Spin Image



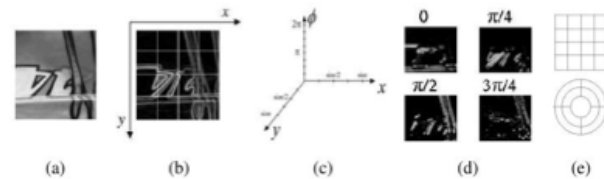
HoG



RIFT

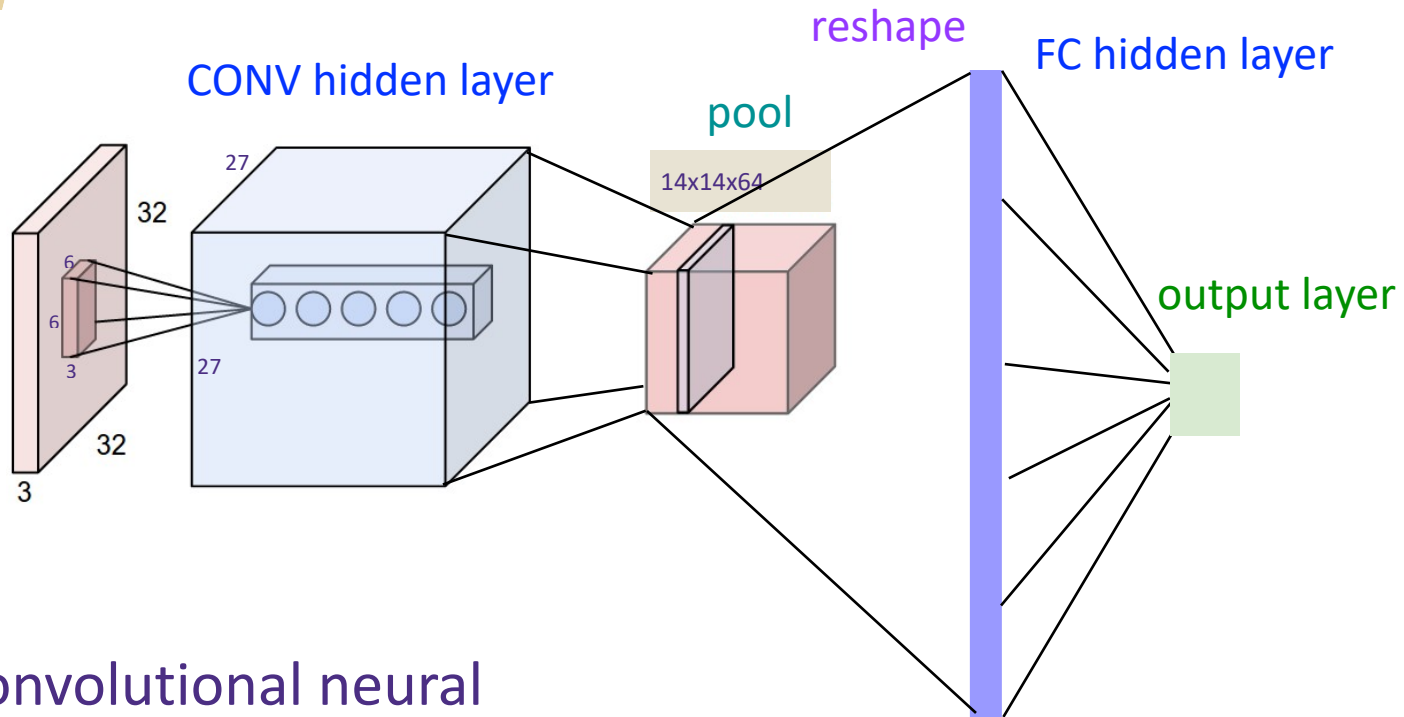


Texton



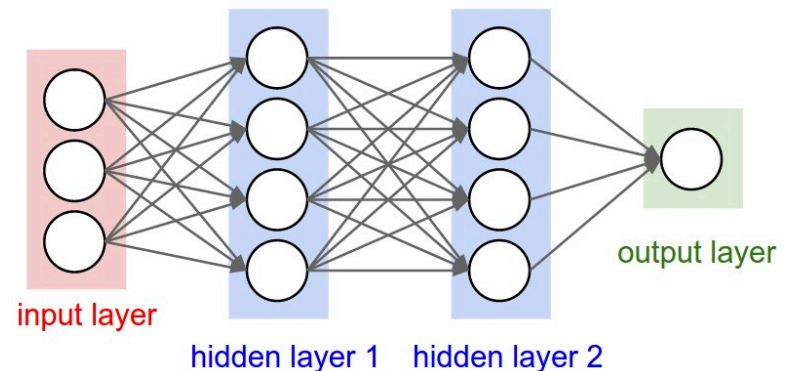
GLOH

# Learning Features with Convolutional Networks

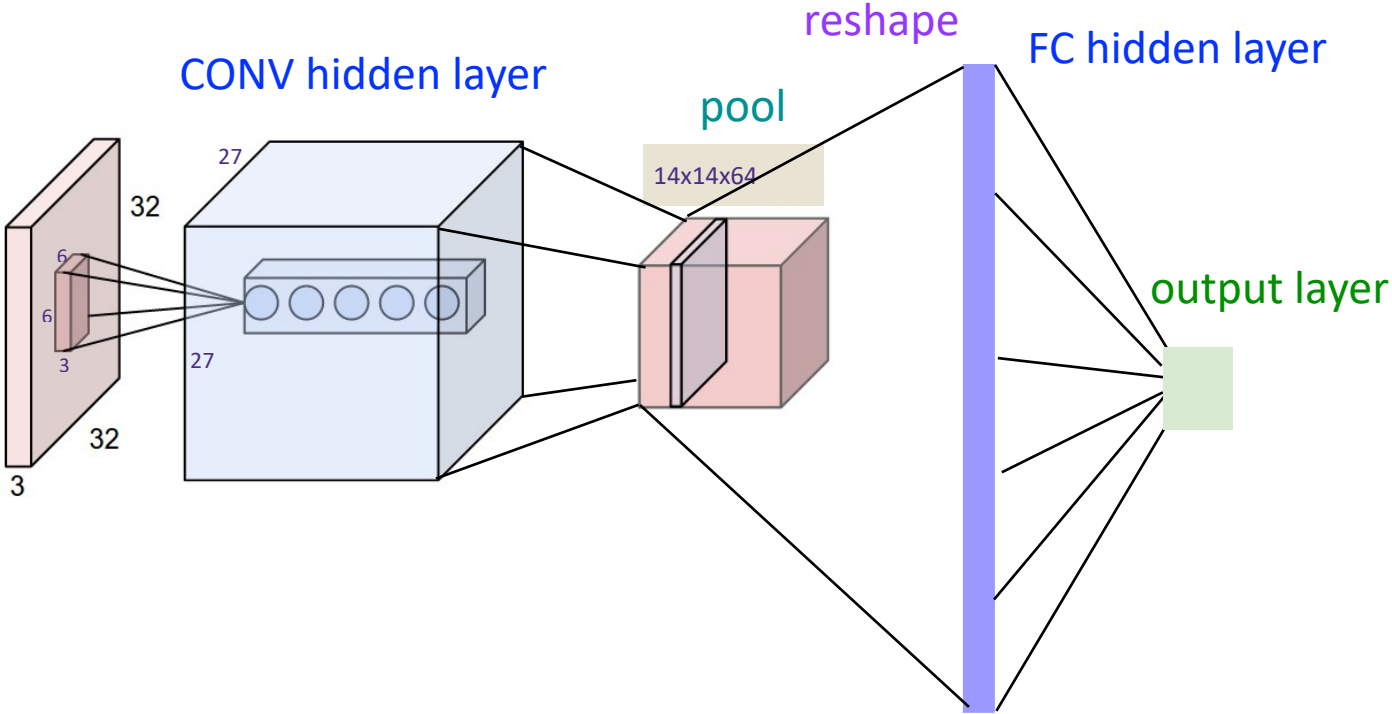


Recall: Convolutional neural networks (CNN) are just regular fully connected (FC) neural networks with some connections removed.

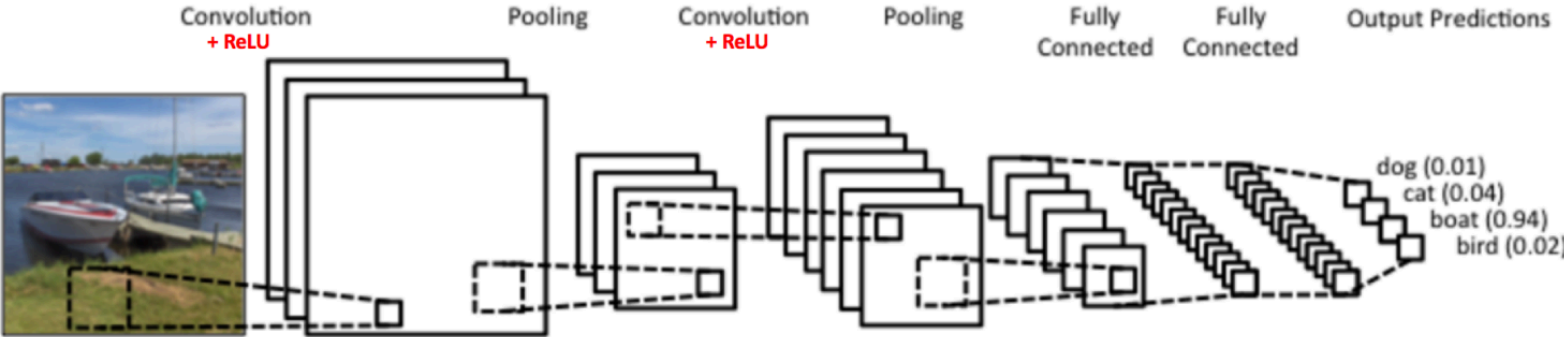
**Train with SGD!**

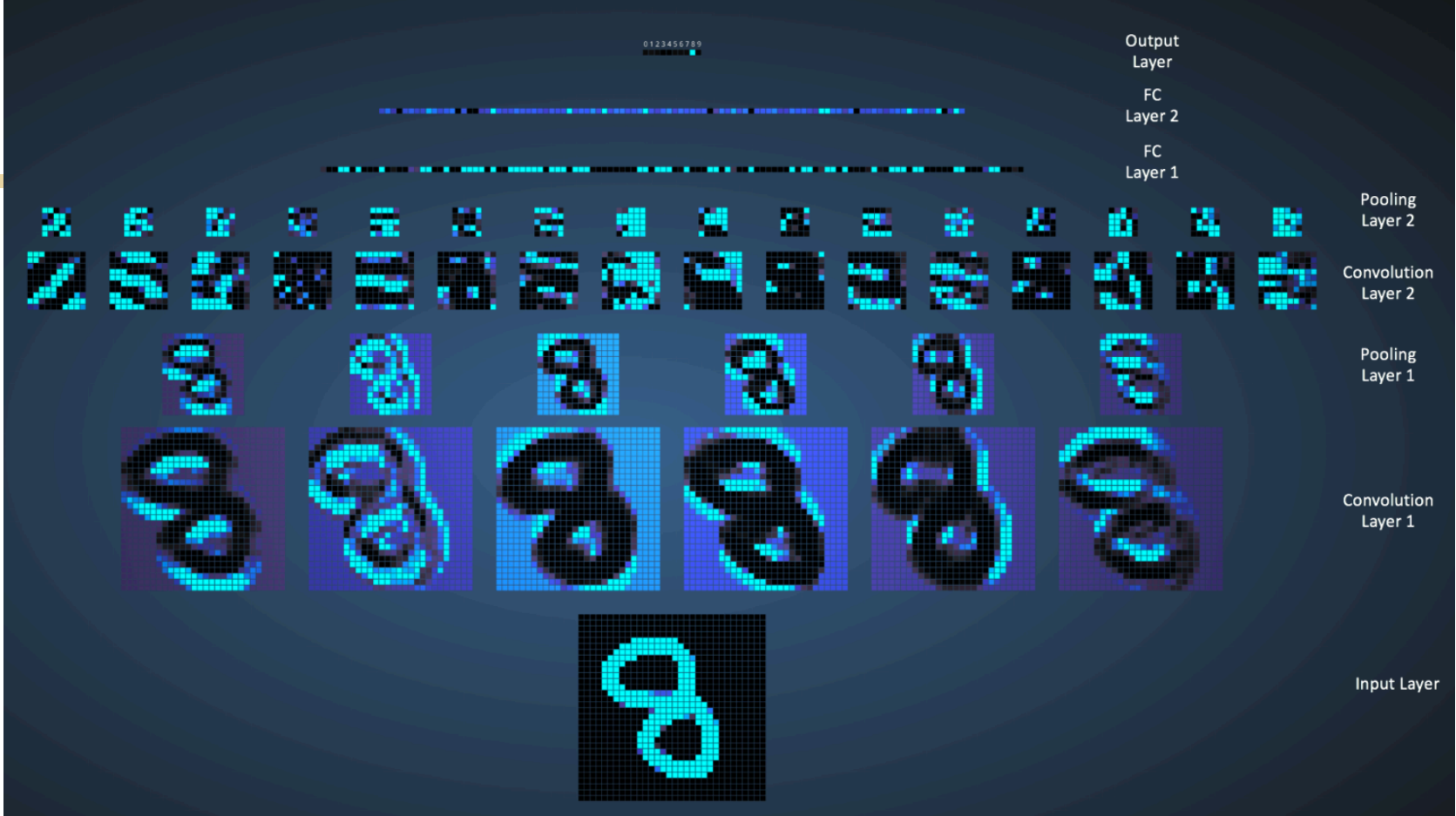


# Training Convolutional Networks

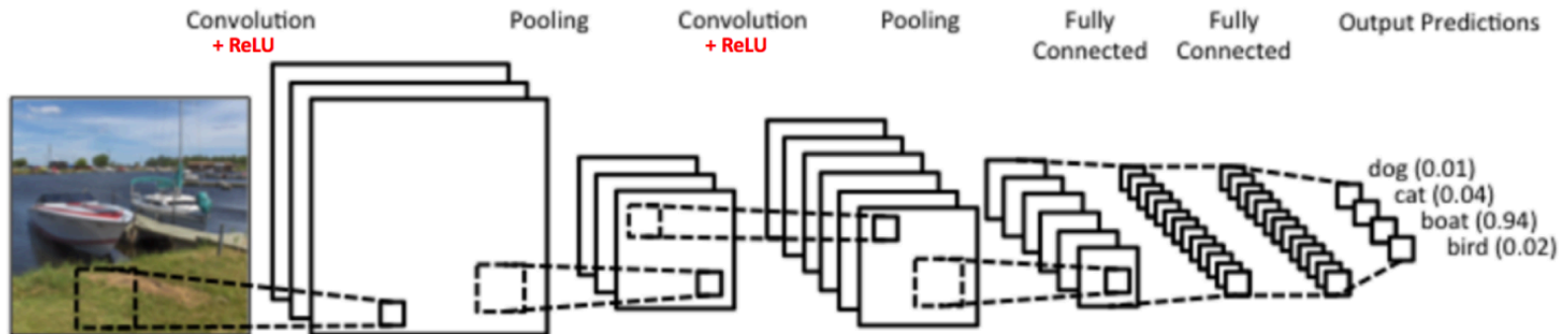


## Real example network: LeNet





Real example network: LeNet

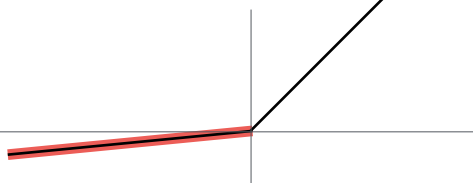


# Real networks

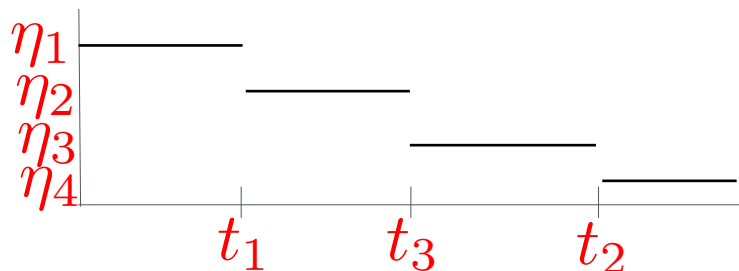
Modern networks have dozens of parameters to tune.

Data augmentation?  
Batch norm?

RELU leakiness  
slope



Learning rate schedule



# Remarks

---

- Convolution is a fundamental operation in signal processing. Instead of hand-engineering the filters (e.g., Fourier, Wavelets, etc.) **Deep Learning *learns* the filters and CONV layers with back-propagation**, replacing fully connected (FC) layers with convolutional (CONV) layers
- **Pooling** is a dimensionality reduction operation that summarizes the output of convolving the input with a filter
- Typically the last few layers are **Fully Connected (FC)**, with the interpretation that the CONV layers are feature extractors, preparing input for the final FC layers. Can replace last layers and retrain on different dataset+task.
- Just as hard to train as regular neural networks.
- More exotic network architectures for specific tasks

# More tricks

- **Data augmentation:** Given an image, its label (“dog and cat”) does not change whether the image is mirrored left to right, or zoomed in. But these additional transformations provide information to the model that it may not know



*Figure 19.1: Illustration of random crops and zooms of a image images. Generated by code at [figures.probml.ai/book1/19.1](https://figures.probml.ai/book1/19.1).*

# Transfer Learning

- **Fine tuning:** Suppose you have a relatively small labeled dataset of rare birds. Given an existing model trained on a large *related* dataset (like popular birds, or animals) use the trained model as an initialization for training the small dataset of rare birds:

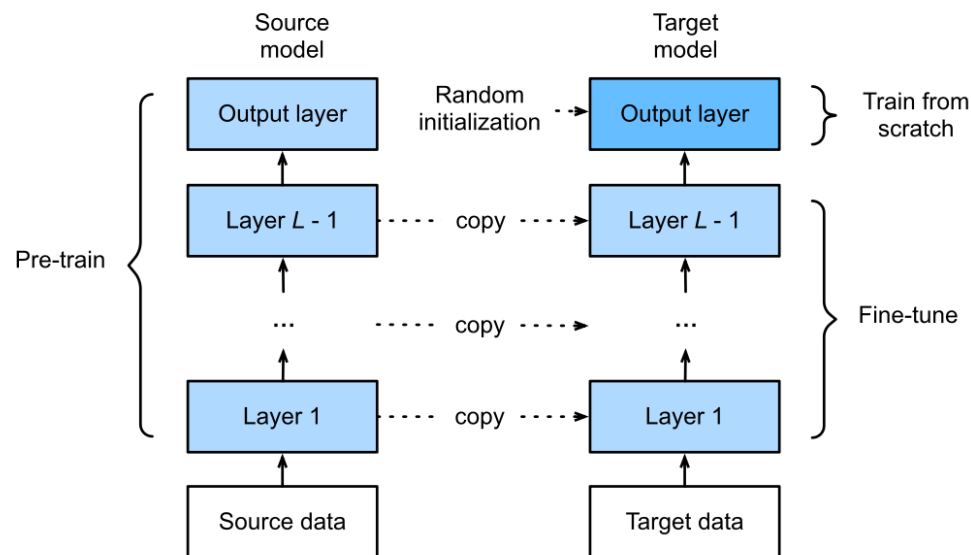


Figure 19.2: Illustration of fine-tuning a model on a new dataset. The final output layer is trained from scratch, since it might correspond to a different label set. The other layers are initialized at their previous parameters, and then optionally updated using a small learning rate. From Figure 13.2.1 of [Zha+20]. Used with kind permission of Aston Zhang.

# Contrastive Learning

- **Contrastive learning:** Select pairs of data points in your dataset, apply augmentations to each and train a model to predict which augmented copy is more similar to which original of the pairs. Example: SimCLR

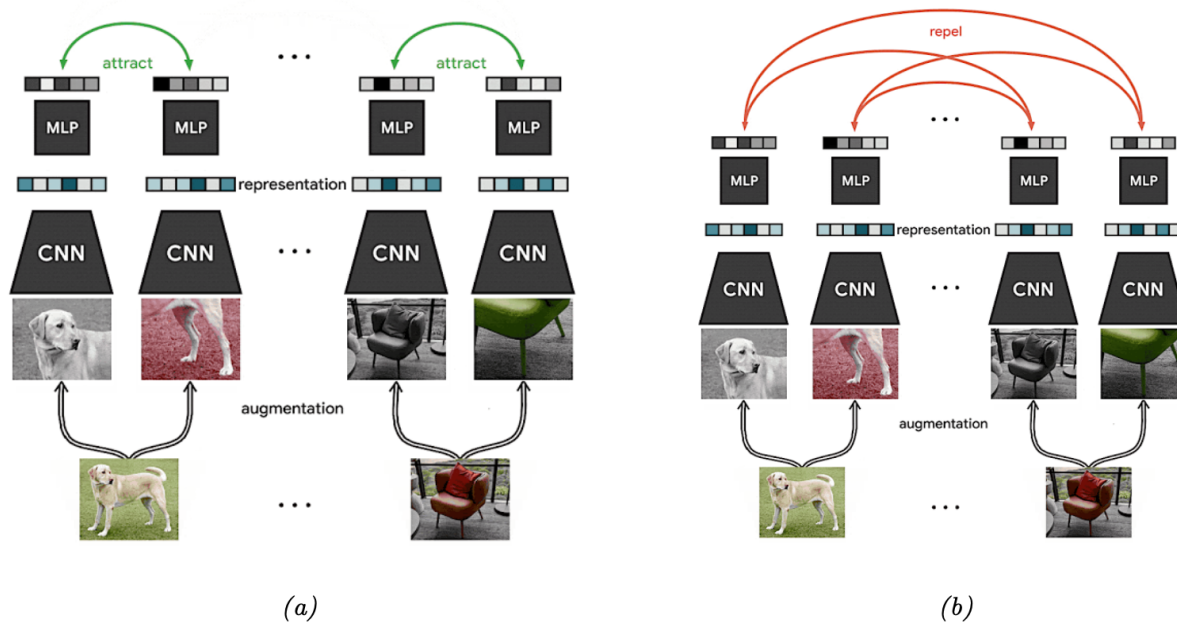
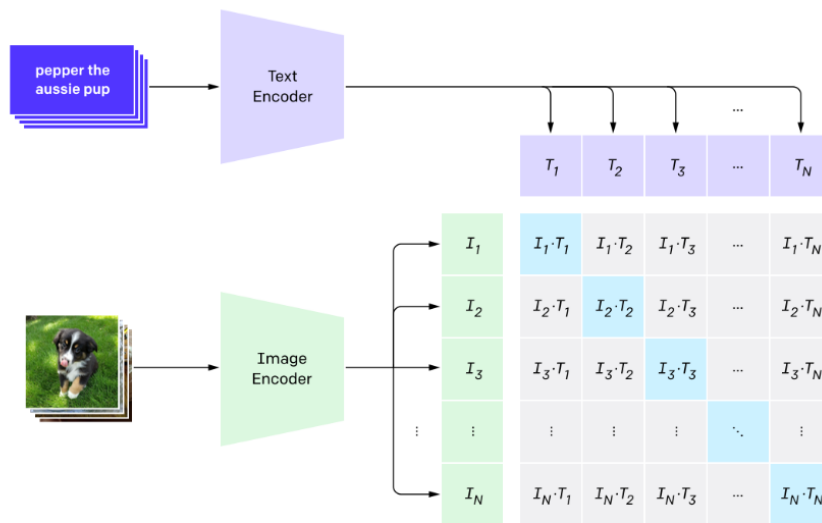


Figure 19.6: Visualization of SimCLR training. Each input image in the minibatch is randomly modified in two different ways (using cropping (followed by resize), flipping, and color distortion), and then fed into a Siamese network. The embeddings (final layer) for each pair derived from the same image is forced to be close, whereas the embeddings for all other pairs are forced to be far. From <https://ai.googleblog.com/2020/04/advancing-self-supervised-and-semi.html>. Used with kind permission of Ting Chen.

# Example: CLIP

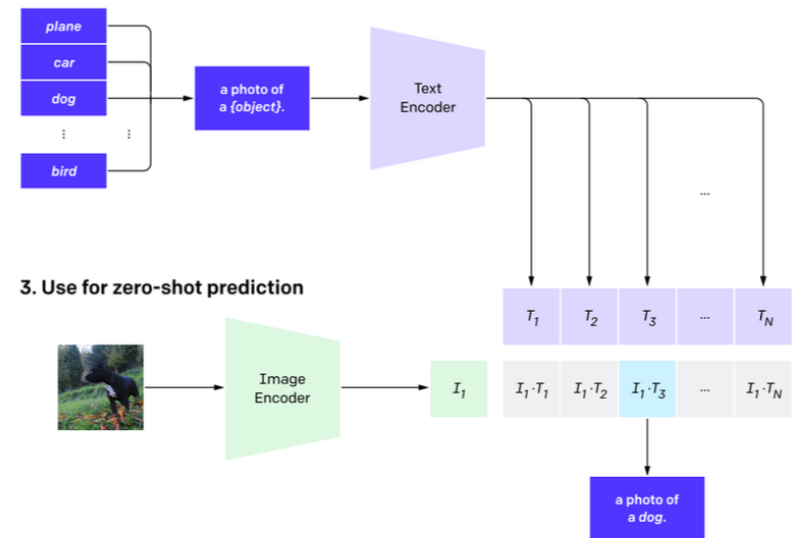
- **CLIP**, from OpenAI, considers a mini batch of image-caption pairs from the internet (e.g., Wikipedia), and trains a model to match each caption to the correct image. This enables *zero-shot prediction*: predicting on a new test set without ever seeing a training image.

## 1. Contrastive pre-training



(a)

## 2. Create dataset classifier from label text



## 3. Use for zero-shot prediction

(b)

Figure 19.7: Illustration of the CLIP model. From Figure 1 of [Rad+]. Used with kind permission of Alec Radford.

Any questions?

