Neural Networks

Some of these slides based on content from Seyong Kim and Nando de Freitas' youtube lectures (which are are really great!) See last slide for link
Some of these slides are from my lecture at the Neuromatch Deep Learning school.
https://deeplearning.neuromatch.io/tutorials/intro.html

Administrivia

- Virtual office hours
  - (**Oct.3, 17, Nov. 14, Dec. 5 only**):
  - https://meet.google.com/ptm-znnz-kbb
  - Meet link is also on eclass

- **There are still in person office hours on these days!!**

Administrivia

- Changes to assignment mentioned last time are now online

Administrivia

- Ask questions on forum today if possible.
  - Priority given to forum questions
- TA coverage over the weekend is limited
  - Urgent questions only over the weekend

Today

- Start with a bit of a review
  - Linear Regression
    - Canonical Equations
  - Logistic Regression
- Neural Networks / CNNs

Recall…. Gradient Descent

Gradient Descent Algorithm

Initialize at some $\mathbf{w}_0$

For t=0,1,2,…do

Compute the gradient $$\nabla_E(\mathbf{w}_t) = -\frac{1}{n}\sum_{k=1}^{n}\mathbf{x}^k\left(y^k - \mathbf{w}_t'\mathbf{x}^k\right)$$

Update the weights $$\mathbf{w}_{t+1} = \mathbf{w}_t - \kappa\nabla_E(\mathbf{w}_t) = \mathbf{w}_t + \kappa\frac{1}{n}\sum_{k=1}^{n}\mathbf{x}^k\left(y^k - \mathbf{w}_t'\mathbf{x}^k\right)$$

Iterate with the next step until **w** doesn't change too much

(or for a fixed number of iterations)

Return final **w**.

Variants of Grad Descent

- **Batch**
  - Use the **whole dataset** to compute the gradient

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \kappa \nabla_E(\mathbf{w}_t) = \mathbf{w}_t + \kappa \ \frac{1}{n}\sum_{k=1}^{N} \mathbf{x}^k \left(y^k - \mathbf{w}_t' \mathbf{x}^k \right)$$

- **Mini-Batch**
  - Use **subsets** of the data (e.g. 1…b, b+1...2*b, etc) to compute gradient

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \kappa \nabla_E(\mathbf{w}_t) = \mathbf{w}_t + \kappa \ \frac{1}{b}\sum_{k=1}^{b} \mathbf{x}^k \left(y^k - \mathbf{w}_t' \mathbf{x}^k \right)$$

- **Stochastic Gradient Descent**
  - Calculate the gradient using **one** training example at a time

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \kappa \nabla_E(\mathbf{w}_t) = \ \mathbf{w}_t + \kappa \ \left(\mathbf{x}^k \left(y^k - \mathbf{w}_t' \mathbf{x}^k \right) \right)$$

The Matrix way:
Canonical Equations for Linear Regression

Matrix **X** and vector **y**

x_0	GPA	YearsOfExperience	Salary
1	90	1	50
1	80	3	60
1	90	2	55
1	70	8	70

$$\mathbf{X} = \begin{bmatrix} 1 & 90 & 1 \\ 1 & 80 & 3 \\ 1 & 90 & 2 \\ 1 & 70 & 8 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} 50 \\ 60 \\ 55 \\ 70 \end{bmatrix}$$

The Matrix Way: Canonical Equations

- *E* will have the smallest value when the gradient is equal to zero.

$$\nabla_E(\mathbf{w}) = -\frac{1}{n}\sum_{k=1}^{n}\mathbf{x}^k\left(y^k - \mathbf{w}'\mathbf{x}^k\right) = \mathbf{0}$$

$$\sum_{k=0}^{n}\mathbf{x}^k\left(y^k - \mathbf{w}'\mathbf{x}^k\right) = \mathbf{0}$$

$$\sum_{k=1}^{n}\left(\mathbf{x}^k y^k - \mathbf{x}^k\mathbf{w}'\mathbf{x}^k\right) = \mathbf{0}$$

$$\sum_{k=1}^{n}\mathbf{x}^k y^k = \sum_{k=1}^{n}\mathbf{x}^k\mathbf{w}'\mathbf{x}^k$$

$$\mathbf{X}'\mathbf{y} = \left(\mathbf{X}'\mathbf{X}\right)\mathbf{w}$$

$$\left(\mathbf{X}'\mathbf{X}\right)^{-1}\mathbf{X}'\mathbf{y} = \mathbf{w}$$

- **X** is the $n \times (m+1)$ data matrix
  - one row of $m+1$ elements for each data instance
    - without the y attribute
- **y** is the n-vector of class values
- **X'X** is $(m+1) \times (m+1)$ matrix
  - Good if number m of attributes is not too big.
- **w** is m-vector, i.e. $(m+1) \times 1$

Python

```
# Training Data
X = np.matrix([[1, 1],
[1, 2]])
# These are the values we want to predict as a column vector
y = np.matrix([1.5, 2]).T

W = np.linalg.inv(X.T*X)*X.T*y
```

```
Result (as expected)
W = [[1. ] [0.5]]
```

https://colab.research.google.com/drive/1lY1BuzQl8G1HtFnCAGVk8zB8_ryfEwYI?usp=sharing

n training tuples *m* attributes

Discussion: GD vs. canonical equations

GD

Needs to iterate, sometimes a lot.

Need to play with kappa.

Method of choice when *m* is big (>10,000).

Canonical equations

No need to iterate, adjust kappa, or scale attributes.

However, the challenge is to compute:

$$\left(\mathbf{X}'\mathbf{X}\right)^{-1}$$

X'X not a problem;

result is $(m+1) \times (m+1)$

Inverting takes $\sim O(m^3)$ time.

Fine for smallish m, difficult after that.

What if X^TX is non-invertible

- Causes:
  - Some columns are linearly dependent
    - E.g. salary is given in two columns; both in CAD and USD
  - Too many attributes, few tuples

- Solution:
  - Delete some attributes
  - Use regularization
  - Use pseudo inverse

Regularization

- Let's say the input data X has dimension n x p
- If there are many parameters to learn, this can lead to overfitting
  - E.g. when $n < p$
- We need a simpler model
- How can we make our regression model simpler?
  - Is a model with $q << p$ parameters simpler?
  - (yes)
- How can we learn a regression model with fewer parameters?

L2 Regularization

$$E(\mathbf{w}) = \frac{1}{2n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)^2 + \frac{\lambda}{2}\sum_{i=1}^{p}\left(\mathrm{w}_i\right)^2$$

$$= \frac{1}{2n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)^2 + \frac{\lambda}{2}\left\|w\right\|_2^2$$

- We get to take the derivative again!!!!

Derivative of the Regularizer

$$\frac{\partial}{\partial \mathrm{w}_j}\left(\frac{\lambda}{2}\sum_{i=1}^{p}\left(\mathrm{w}_i\right)^2\right) = ?$$

Exercises:

1. What's the update for L2 regression using Gradient Descent?
2. Can you set the derivative to zero and solve for w using this version of the error function?

Recall: logistic regression

Expressing Conditional Log Likelihood

$$\mathcal{L}(W) \equiv ln\left(\prod_i P(Y^i|X^i,W)\right)$$

$$= \sum_i ln\ \underline{P(Y^i|X^i,W)}$$

$$p(y=0\,|\,\mathbf{x}) = \frac{1}{1+e^{\mathbf{w}'\mathbf{x}}}$$

$$p(y=1\,|\,\mathbf{x}) = \frac{e^{\mathbf{w}'\mathbf{x}}}{1+e^{\mathbf{w}'\mathbf{x}}}$$

$$\mathcal{L}(W) = \sum_i \underline{Y^i\ ln\ P(Y^i=1|X^i,W)} + \sum_i \underline{(1-Y^i)\ ln\ P(Y^i=0|X^i,W)}$$

For the samples with Y^l=1

For the samples with Y^l=0

Expressing Conditional Log Likelihood

$$\mathcal{L}(W) = \sum_i Y^i \; ln \; P(Y^i = 1|X^i, W) + \sum_i (1 - Y^i) \; ln \; P(Y^i = 0|X^i, W)$$

$$= \sum_i Y^i \; ln \; \frac{P(Y^i = 1|X^i, W)}{P(Y^i = 0|X^i, W)} + \sum_i ln \; P(Y^i = 0|X^i, W)$$

Expressing Conditional Log Likelihood

$$\mathcal{L}(W) = \sum_i Y^i \; ln \; P(Y^i = 1|X^i, W) + \sum_i (1 - Y^i) \; ln \; P(Y^i = 0|X^i, W)$$

$$= \sum_i Y^i \; ln \; \frac{P(Y^i = 1|X^i, W)}{P(Y^i = 0|X^i, W)} + \sum_i ln \; P(Y^i = 0|X^i, W)$$

$$= \sum_i Y^i \; (w'x^i) - \sum_i ln(1 + e^{w'x^i})$$

$$p(y = 0 \mid \mathbf{x}) = \frac{1}{1 + e^{\mathbf{w}'\mathbf{x}}}$$

$$p(y = 1 \mid \mathbf{x}) = \frac{e^{\mathbf{w}'\mathbf{x}}}{1 + e^{\mathbf{w}'\mathbf{x}}}$$

Maximizing Conditional Log Likelihood

$$\mathcal{L}(W) = \sum_i Y^i \ (w' x^i) - \sum_i ln(1 + e^{w' x^i})$$

$$\frac{d}{dw}\mathcal{L}(W) = \sum_i Y^i x^i - \sum_i \frac{x^i e^{w' x^i}}{1 + e^{w' x^i}}$$

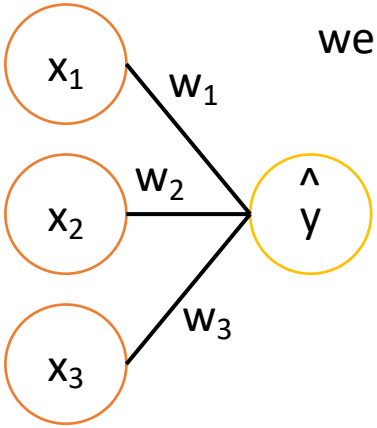
$$= \sum_i x^i (Y^i - P(Y = 1 | W, x^i))$$

- Good news: *L(W)* is concave function of *W*
- **Bad news: no closed-form solution to maximize *L(W) [i.e., no canonical equation]***

Logistic regression is actually a simple neural net

Neural Nets

input features (x)



weights (w) associated with each edge

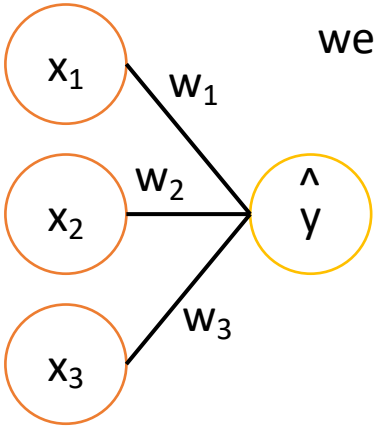
prediction $\hat{y}$

$$\hat{y} = \frac{1}{1 + e^{-\sum_i x_i w_i}}$$

output
(in the range [0…1])

Neural Nets

input features (x)



weights (w) associated with each edge

prediction $\hat{y}$

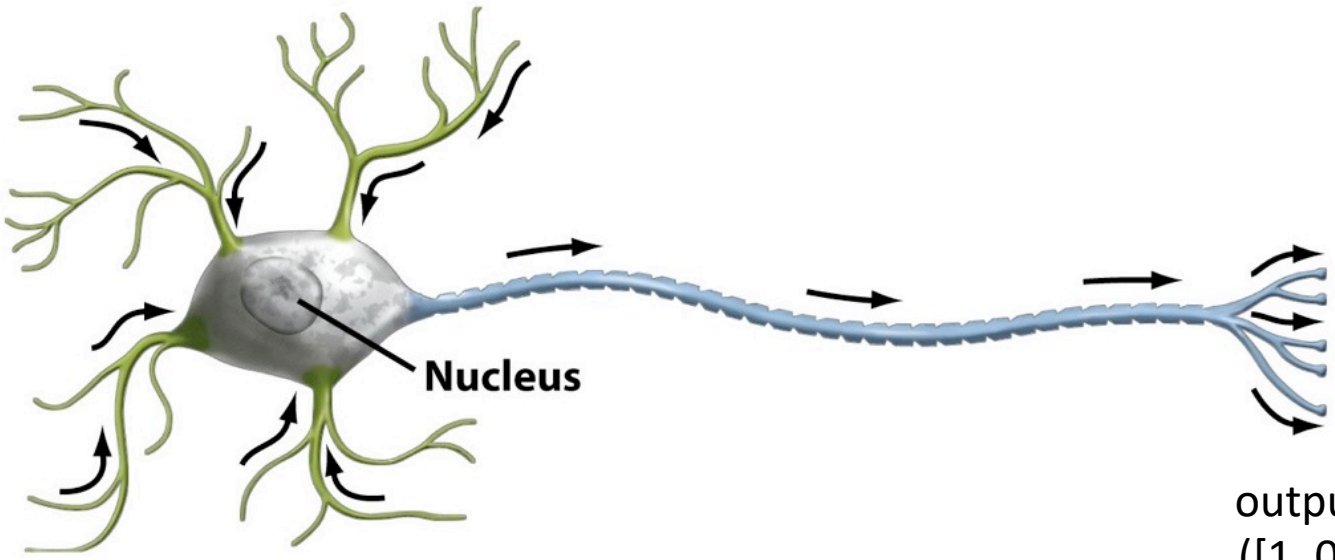
$$\hat{y} = \frac{1}{1 + e^{-\sum_i x_i w_i}}$$

output
(in the range [0…1])

Neural Nets

- Have been around for decades
  - See http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/ for a brief history lesson
- Recently re-popularized due to
  - ability to process more data more quickly
    - GPUs
  - availability of large datasets (particularly in computer vision)

Inspiration

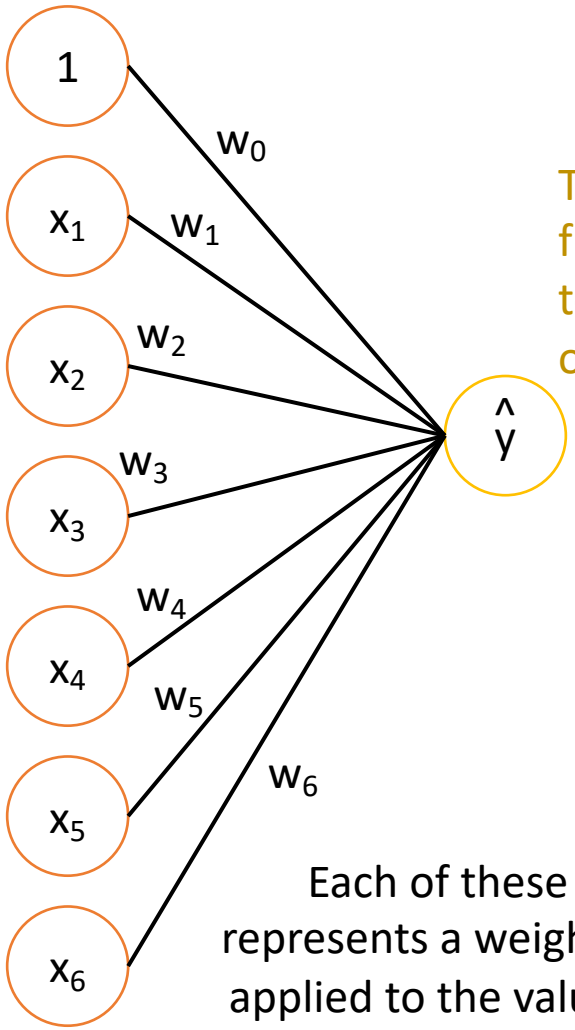


input
(many inputs
from many cells,
weighted with
different
importance)

output
([1, 0]
fire, don’t fire)

Logistic Regression Models are Neural Networks

x is our input feature vector



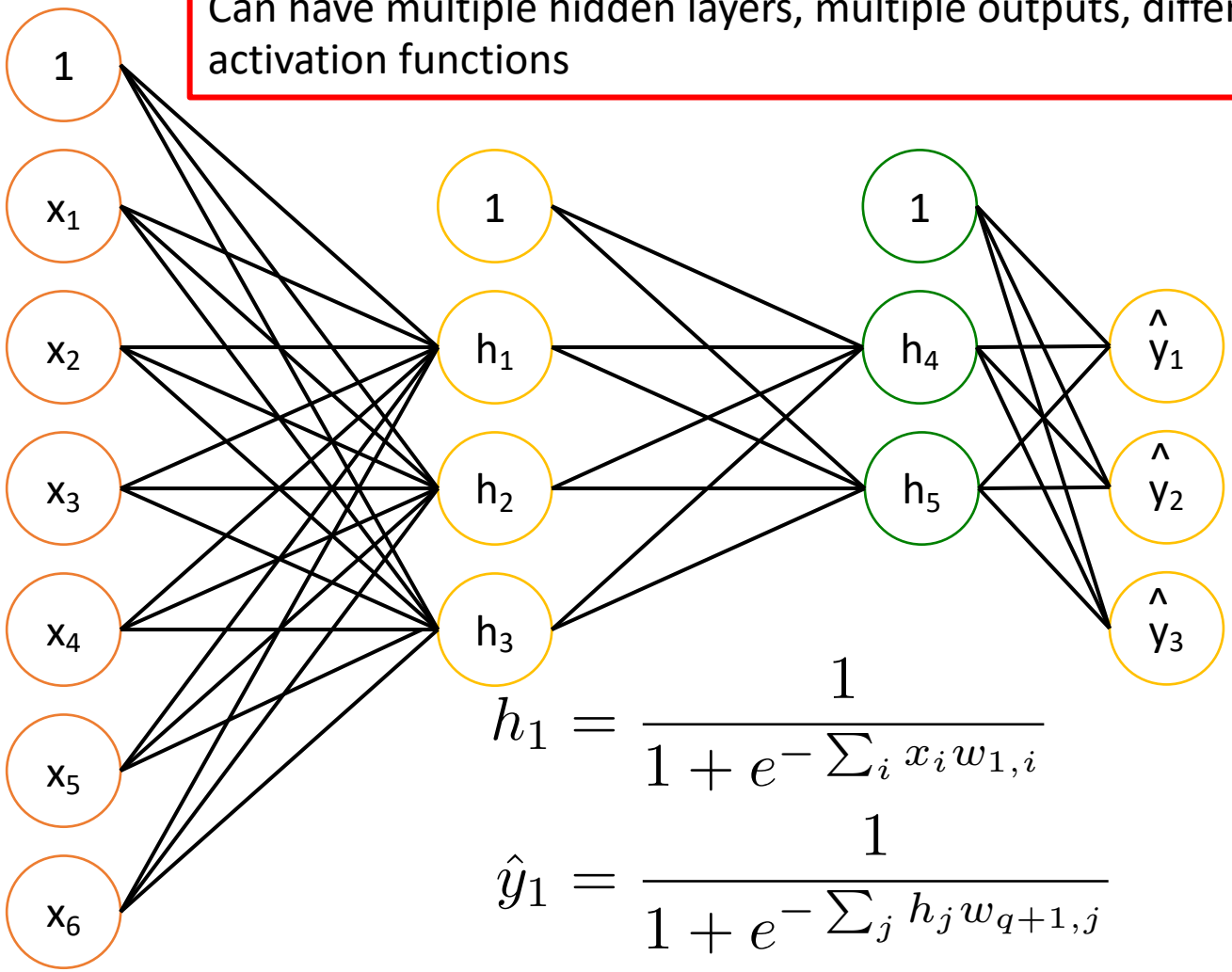
This circle (node) represents a function of the edges connected to it from the left, and the nodes on the other end of that edge

$$\hat{y} = \frac{1}{1 + e^{-\sum_i x_i w_i}}$$

Each of these lines (edges) represents a weight (*w*) that will be applied to the value in the node of the left

Neural Nets

Can have multiple hidden layers, multiple outputs, different activation functions



$$h_1 = \frac{1}{1 + e^{-\sum_i x_i w_{1,i}}}$$

$$\hat{y}_1 = \frac{1}{1 + e^{-\sum_j h_j w_{q+1,j}}}$$

q is total # of hidden nodes across all layers

Deep Learning for Computer Vision

- Some of these slides are from my lecture at the Neuromatch Deep Learning school.
  - https://deeplearning.neuromatch.io/tutorials/intro.html

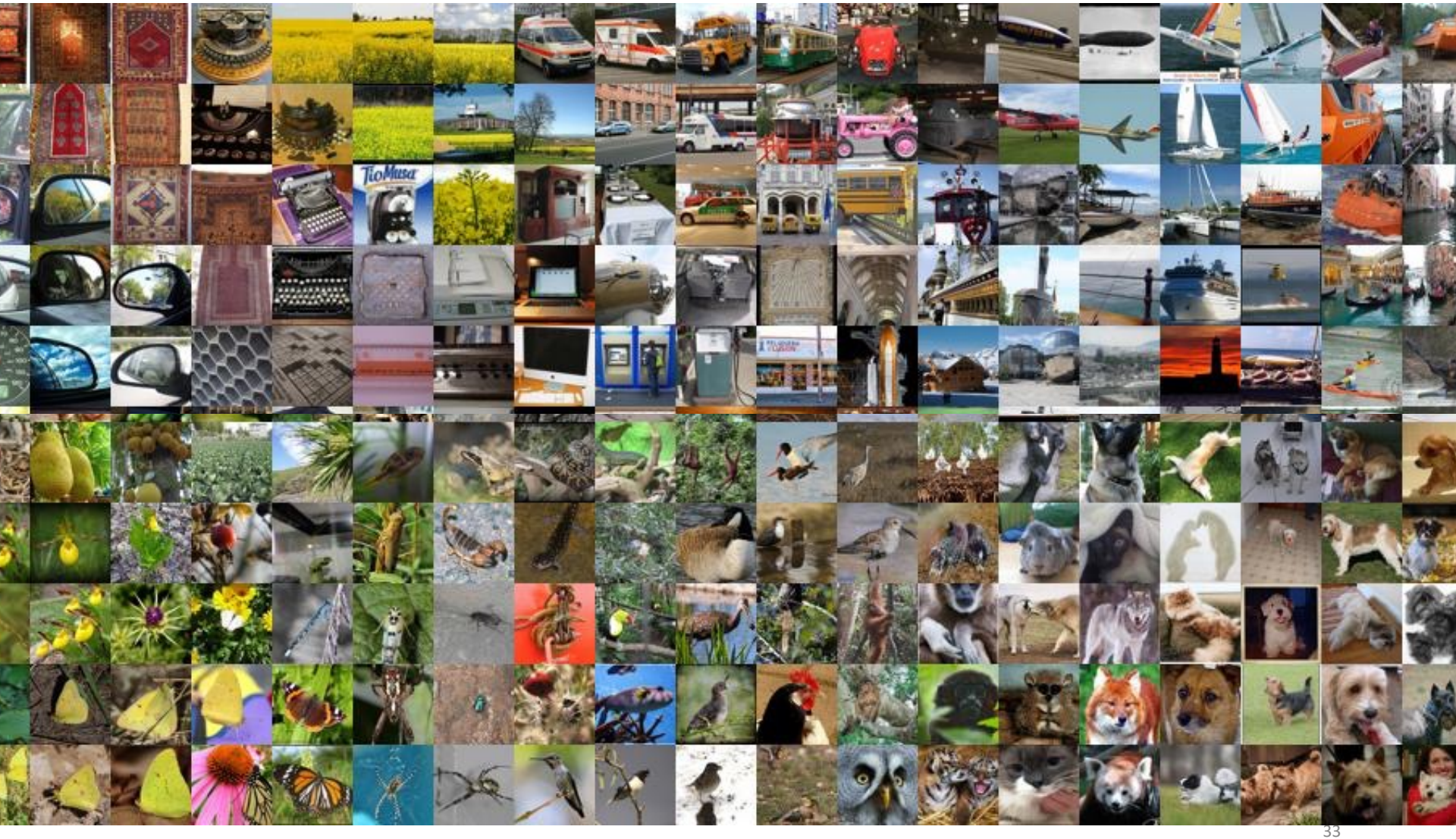
- Notebook
  - https://colab.research.google.com/drive/1_YP5SicR49OrlWpct0IrrAI5r4LsTDVz?usp=sharing

CNNs

- Amazing advances in computer vision
  - ImageNet
    - First released 2009
    - 1.2 million images
    - more than 1000 concepts
      - e.g. cup, oil filter, ptarmigan
  - Deep learning
    - CNNs

http://image-net.org/

ImageNet



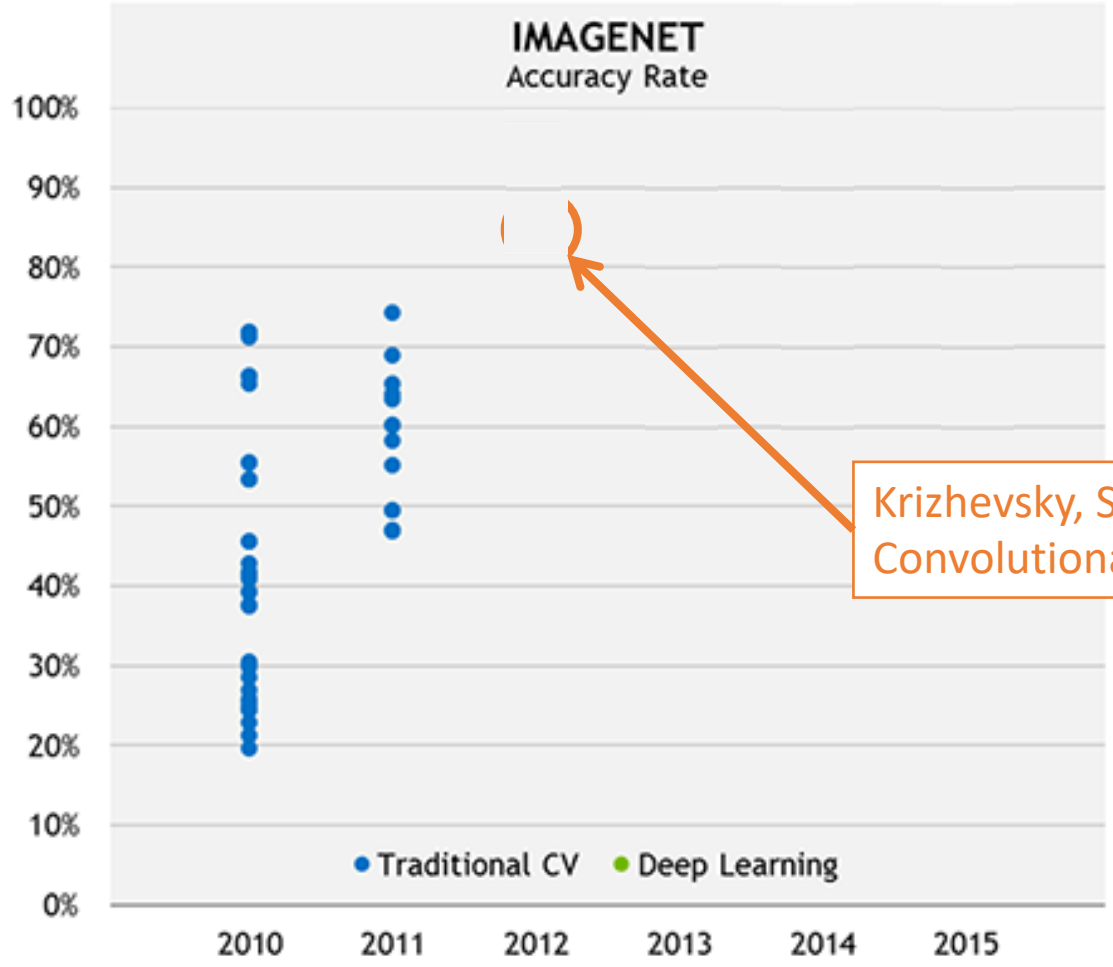
http://cs.stanford.edu/people/karpathy/cnnembed/

ImageNet

french fries mashed potato black olive face powder crab apple Granny Smith strawberry blueberry cranberry currant blackberry raspberry persimmon mulberry orange kumquat lemon grapefruit plum fig pineapple banana jackfruit cherry grape custard apple durian mango elderberry guava litchi pomegranate quince kidney bean soy green pea chickpea chard lettuce cress spinach bell pepper pimento jalapeno cherry tomato parsnip turnip mustard bok choy head cabbage broccoli cauliflower brussels sprouts zucchini spaghetti squash acorn squash butternut squash cucumber artichoke asparagus green onion shallot leek cardoon celery mushroom pumpkin cliff lunar crater valley alp volcano promontory sandbar dune coral reef lakeside seashore geyser bakery juniper berry gourd acorn olive hip ear pumpkin seed sunflower seed coffee bean rapeseed corn buckeye bean peanut walnut cashew chestnut hazelnut coconut pecan pistachio lentil pea peanut okra sunflower lesser celandine wood anemone blue columbine delphinium nigella calla lily sandwort pink baby's breath ice plant globe amaranth four o'clock Virginia spring beauty wallflower damask violet candytuft Iceland poppy prickly poppy oriental poppy celandine blue poppy Welsh poppy celandine poppy corydalis pearly everlasting strawflower yellow chamomile dusty miller tansy daisy common marigold China aster cornflower chrysanthemum mistflower cosmos dahlia coneflower blue daisy gazania African daisy male orchis butterfly orchid aerides brassavola spider orchid grass pink calypso cattleya red helleborine coelogyne cymbid lady's slipper marsh orchid dendrobium disa helleborine fragrant orchid fringed orchis lizard orchid laelia masdevallia odontoglossum oncidium bee orchid fly orchid spider orchid phaius moth orchid ladies' tresses stanhopea stelis vanda cyclamen centaury gentian begonia commelina scabious achimenes African violet streptocarpus scorpionweed calceolaria toadflax veronica bonsai star anise wattle huisache silk tree rain tree dita pandanus linden American beech New Zealand beech live oak shingle oak pin oak cork oak yellow birch American white birch downy birch alder fringe tree European ash fig witch elm Dutch elm cabbage tree golden shower tree honey locust Kentucky coffee tree Brazilian rosewood logwood coral tree Japanese pagoda tree kowhai palm Arabian coffee cork tree weeping willow pussy willow goat willow China tree pepper tree balata teak ginkgo pine ilang-ilang laurel magnolia tulip tree baobab kapok red beech cacao sorrel tree iron tree mangrove paper mulberry Judas tree redbud mountain ash ailanthus silver maple Oregon maple sycamore box elder Japanese maple holly dogwood truffle shiitake lichen hen-of-the-woods jelly fungus dead-man's-fingers earthstar coral fungus stinkhorn puffball gyromitra bolete polypore gill fungus morel agaric trilobite harvestman scorpion black and gold garden spider barn spider garden spider black widow tarantula wolf spider tick mite centipede millipede horseshoe crab

http://image-net.org/challenges/LSVRC/2010/browse-synsets

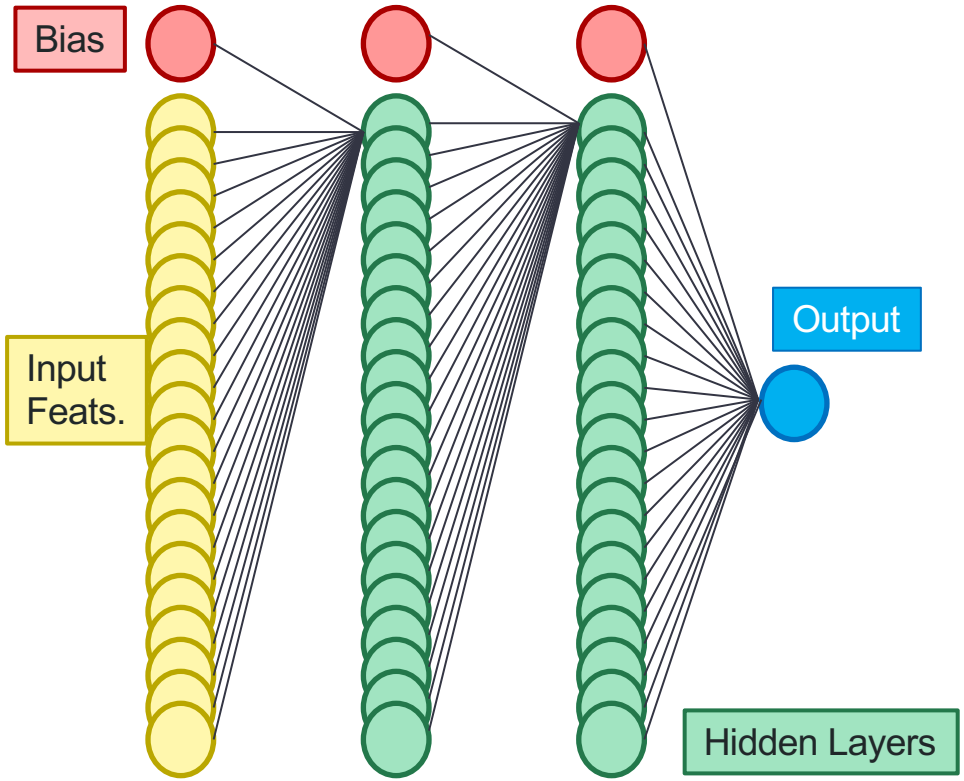
http://image-net.org/challenges/LSVRC/2012/analysis/



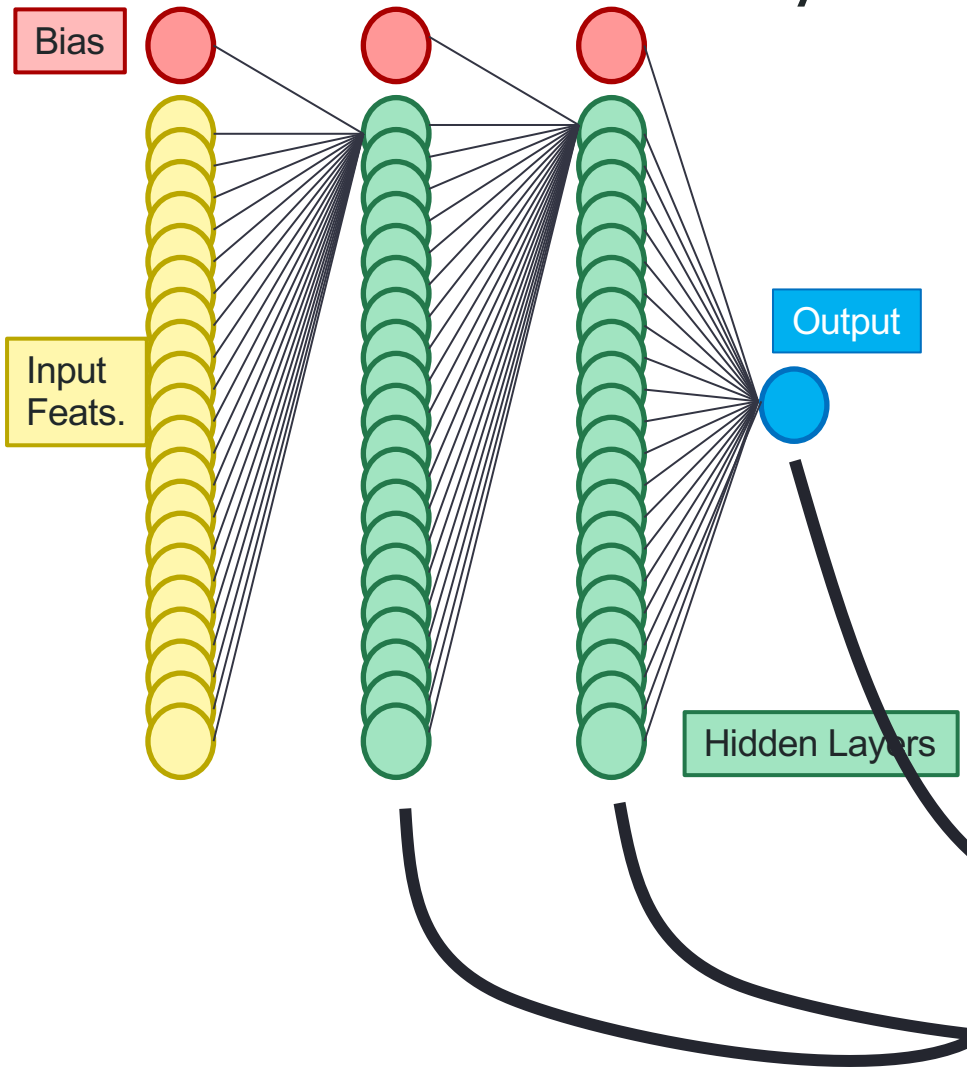
Krizhevsky, Sutskev & Hinton
Convolutional Neural Net (CNN)

https://blogs.nvidia.com/blog/2016/01/12/accelerating-ai-artificial-intelligence-gpus/

Recall: Multi-layer perceptrons (MLPs)



An MLP can have many many params



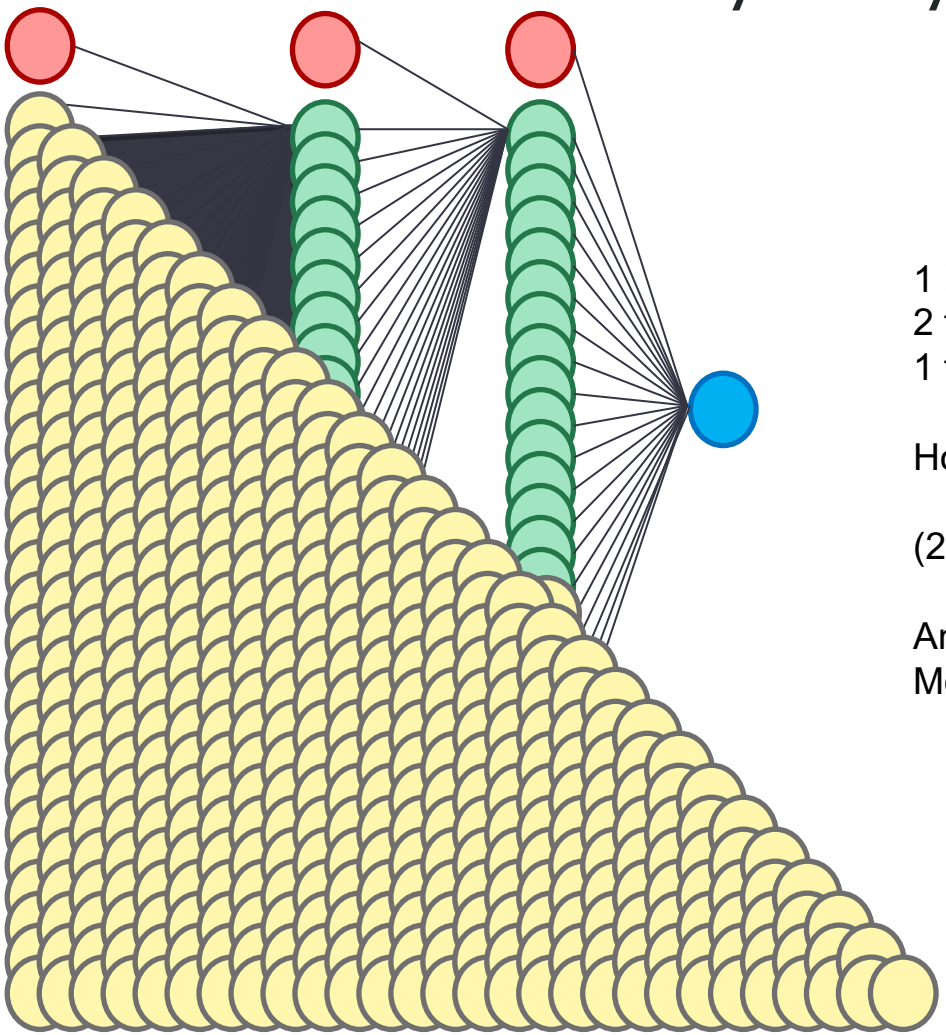
Data: 20 input features, single binary label

- 1 input layer with 20 nodes,
- 2 fully connected layers, 20 nodes each
- 1 final prediction node

How many weights is that?

(20+1)*20+(20+1)*20+(20+1)*1=861 params

An MLP can have many many params



1 input layer with 256*256 nodes,
2 fully connected layers, 20 nodes each
1 final prediction node

How many weights is that?

(256*256+1)*20+(20+1)*20+(20+1)*1 = ~1.3M params

And that's with only two fairly small hidden layers!
Modern CNNs have many more layers! 50-100 or more!

Translation invariance in images

- At the same time that the number of parameters grows very quickly, we are missing an opportunity

JOSH HAROLDSON VIA FLICKR // CC BY-NC 2.0

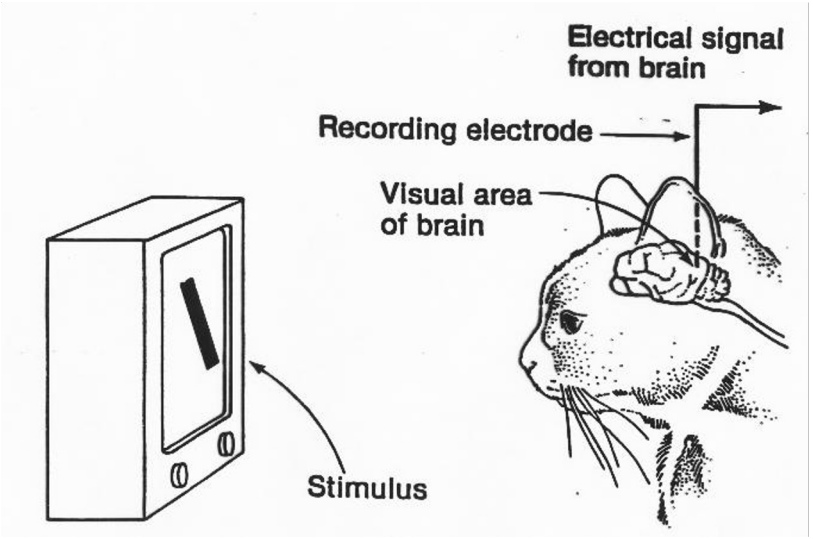
Invariances

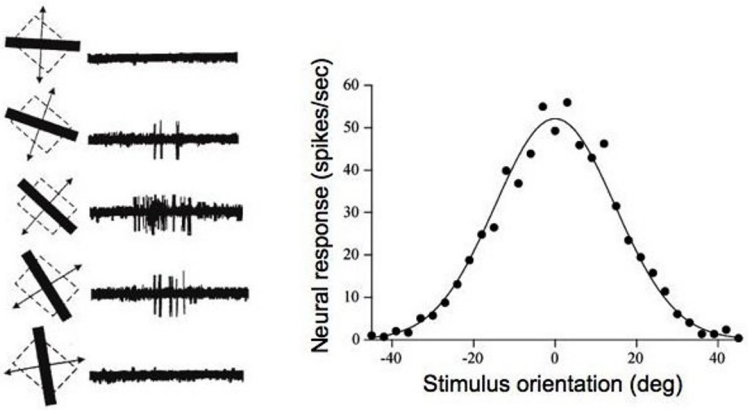
- Would like to capture the invariances we see in images
  - objects can appear at any place in the image

Visual Processing in the Brain

- The brain is an efficient machine
- How does it “solve” vision?

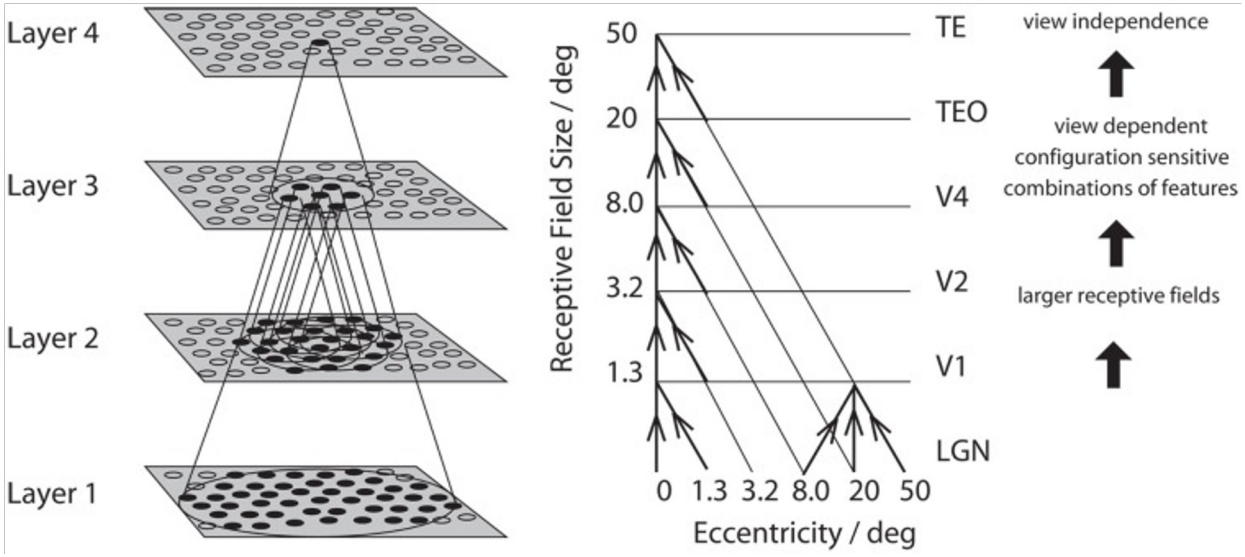
Hubel and Wiesel





Hubel & Wiesel, 1968

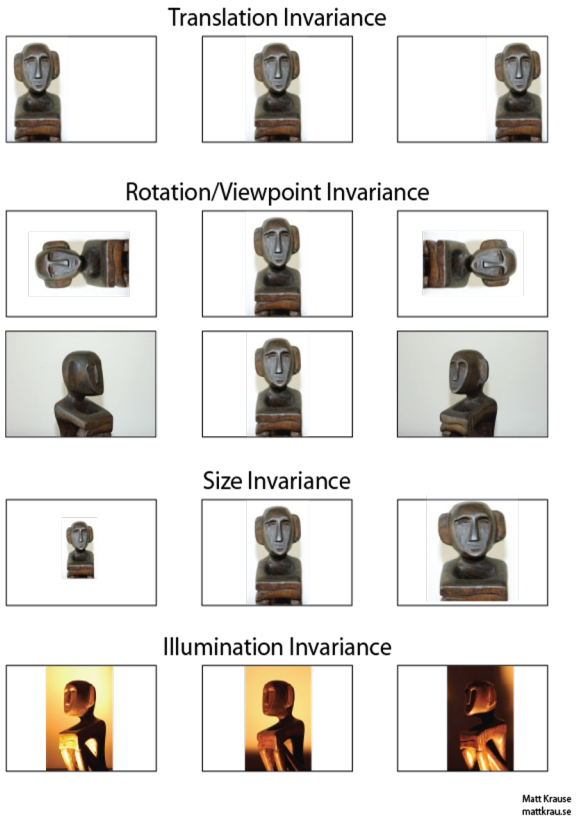
A hierarchy of processing



Rolls 2012

Invariances

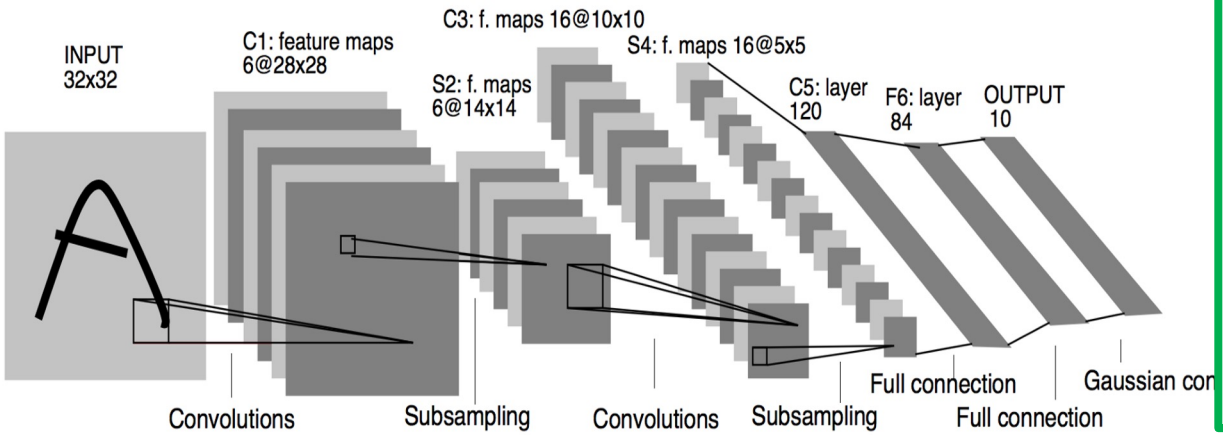
JOSH HAROLDSON VIA FLICKR // CC BY-NC 2.0

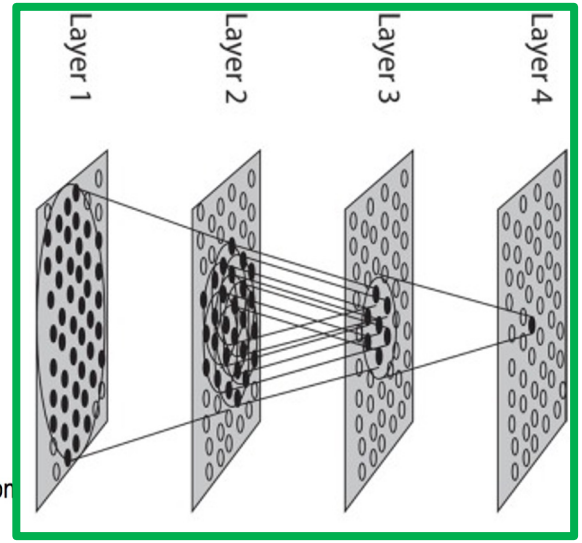


Early history of CNNs

LeNet (1998) -- Architecture

- Developed by Yann LeCun @ Bell labs



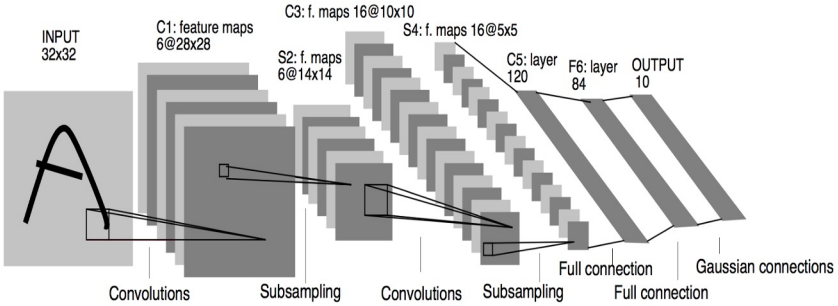


LeNet (1998) -- Results

- Successfully trained a 60K parameter neural network without GPU acceleration!
- Automatically reading zip codes on mail
- 0.8% error on MNIST; near state-of-the-art at the time.

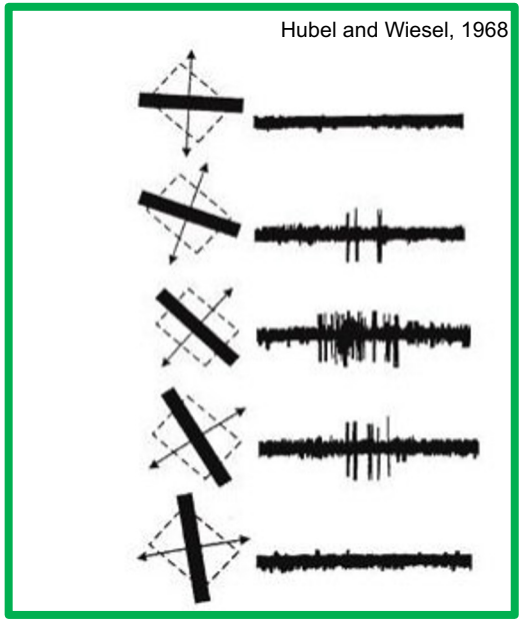
LeNet: http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

CNNs: Convolution



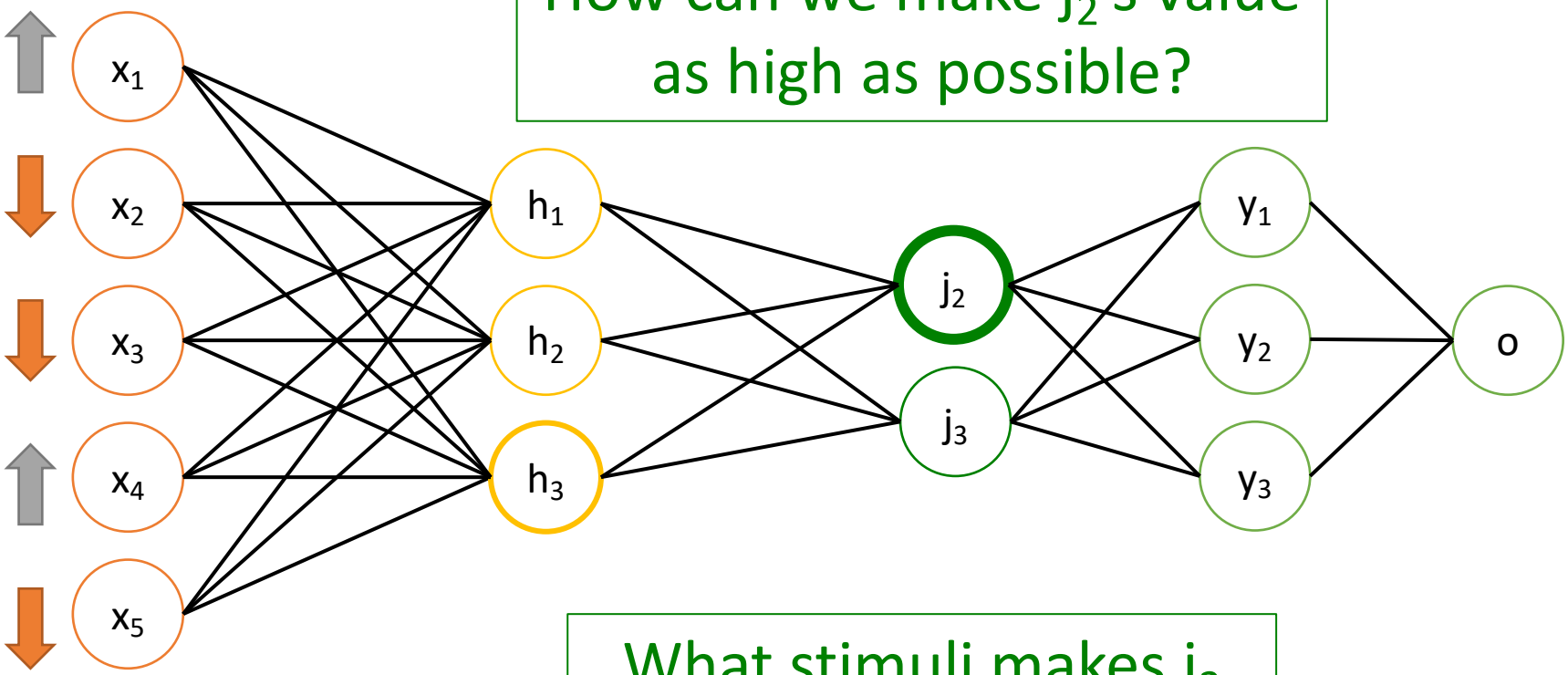
- Typical Learned Filters (C1 feature maps)

Krizhevsky et al., 2012

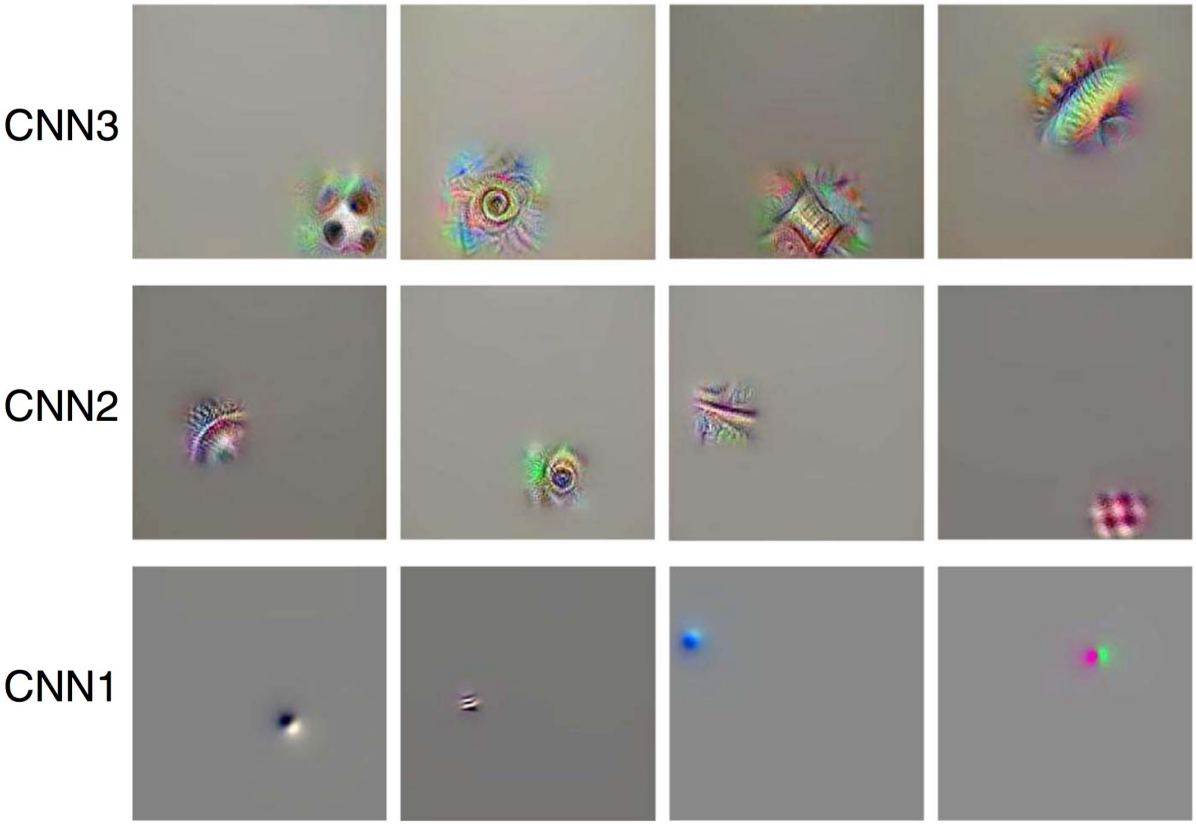


What do the deeper neurons represent?

How can we make j_2's value as high as possible?



What stimuli makes j_2 fire maximally?



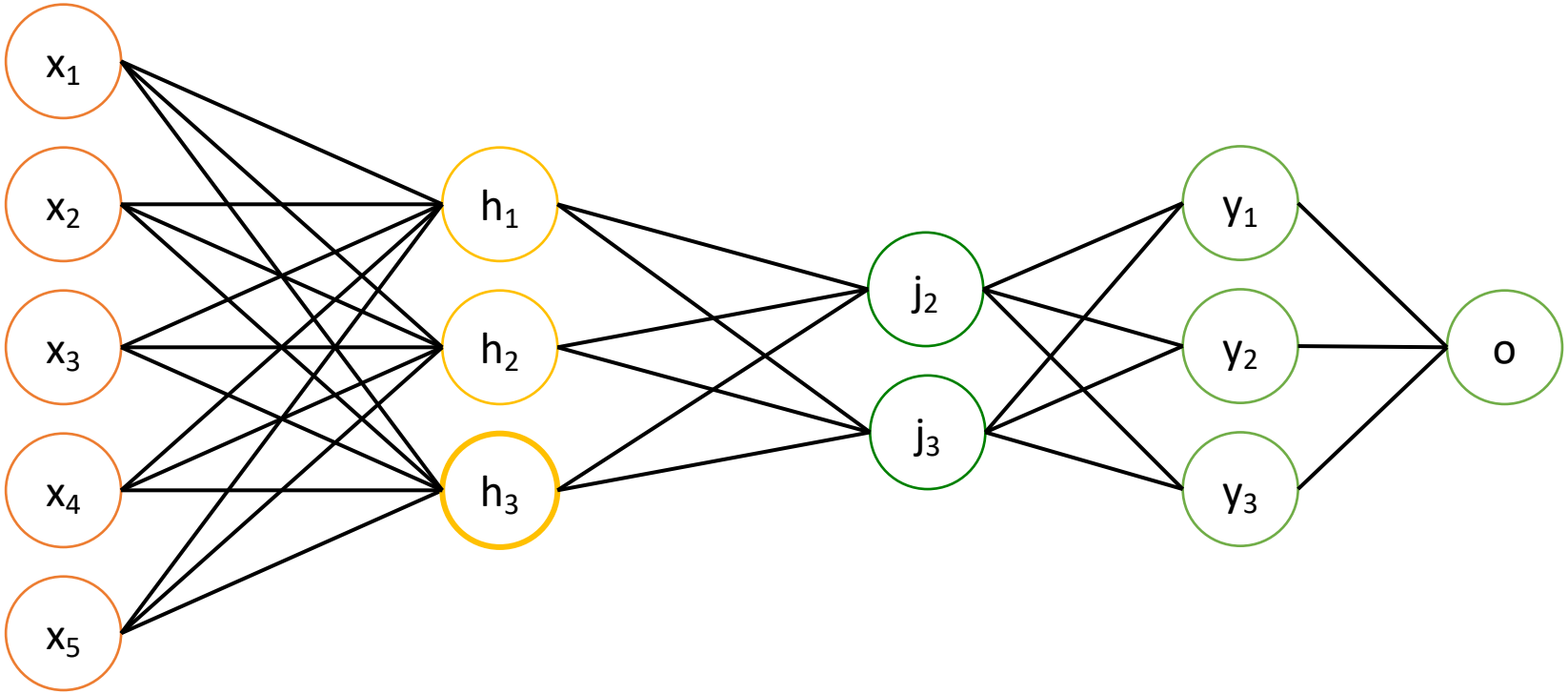
Horikawa & Kamitani (2017)

Horikawa & Kamitani (2017)

DL is a story of representation learning

Hence ICLR

What is a representation?



DL is a story of representation learning

What is a representation?

A symbol or construct that approximates the entities and/or relations in the real world

Your brain has representations of the real world because it cannot actually *contain* the real world

DL is a story of representation learning

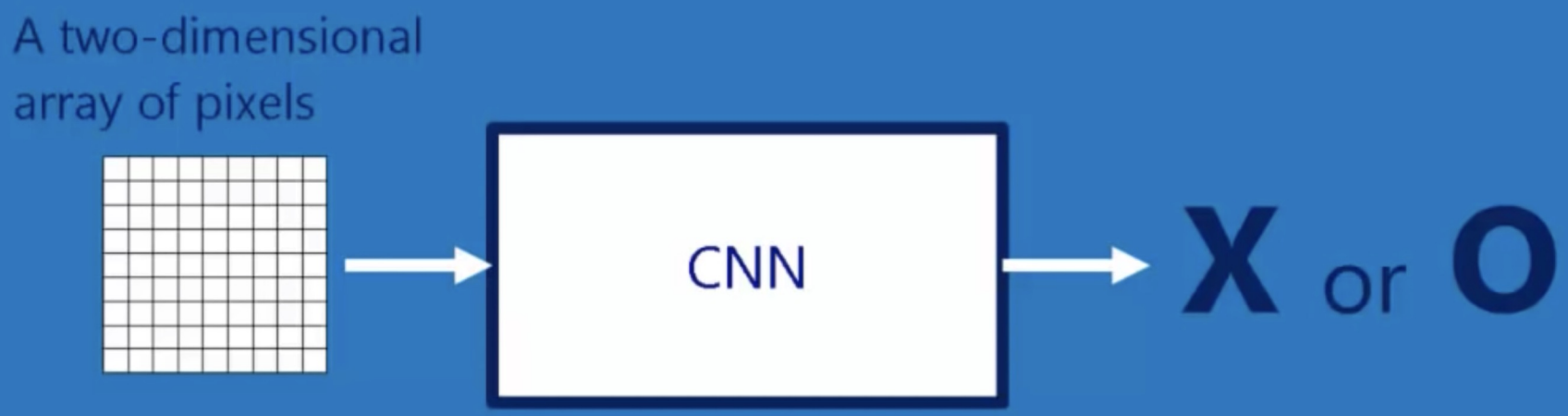
Your brain's representations allow it to approximate the real world in very useful ways.

E.g. you understand that some objects are more similar than others, you understand that objects can work together, objects can be parts of other objects, etc…

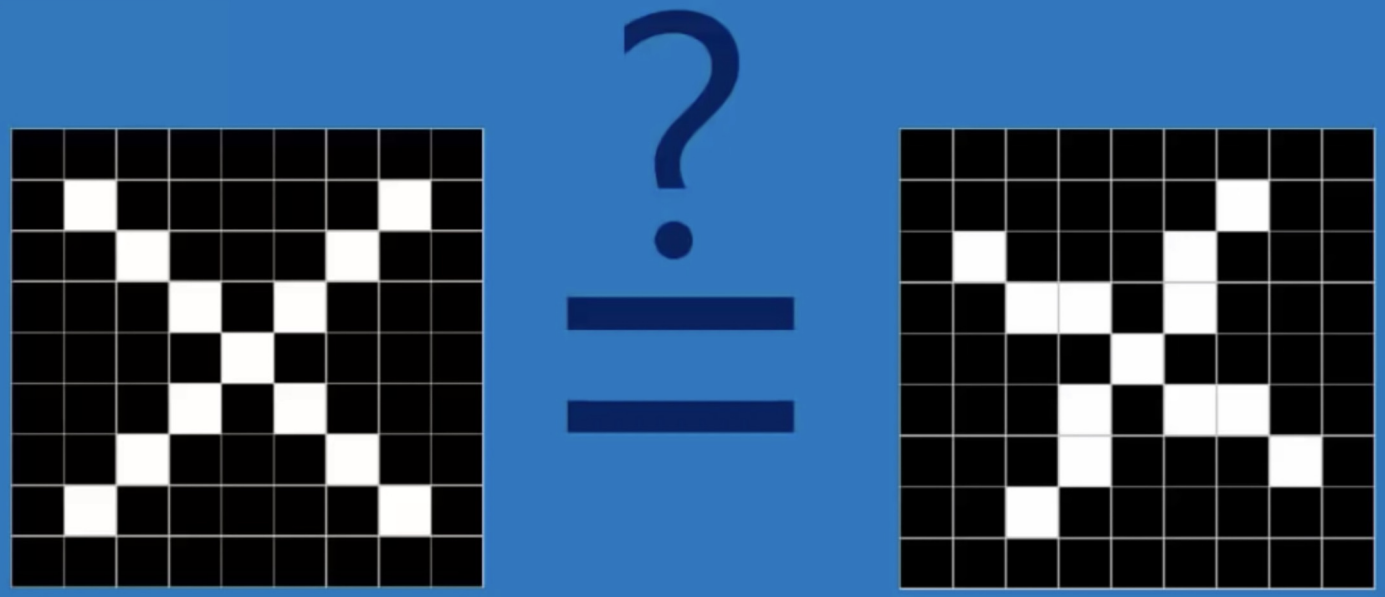
What do we want a good representation to do for us?

What is a CNN?

All the (blue) slides adapted from Brandon Rohrer with permission



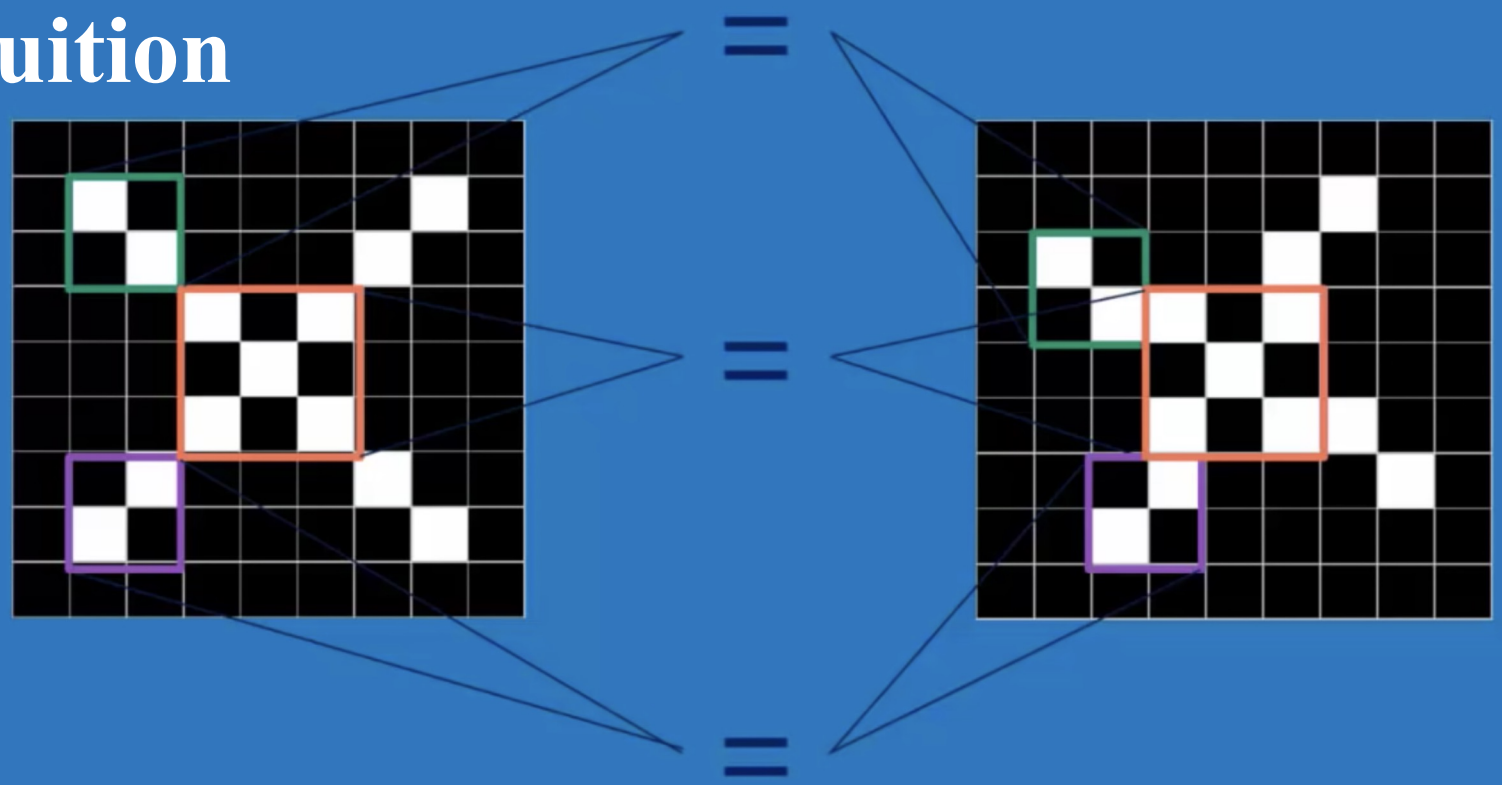
Nontrivial



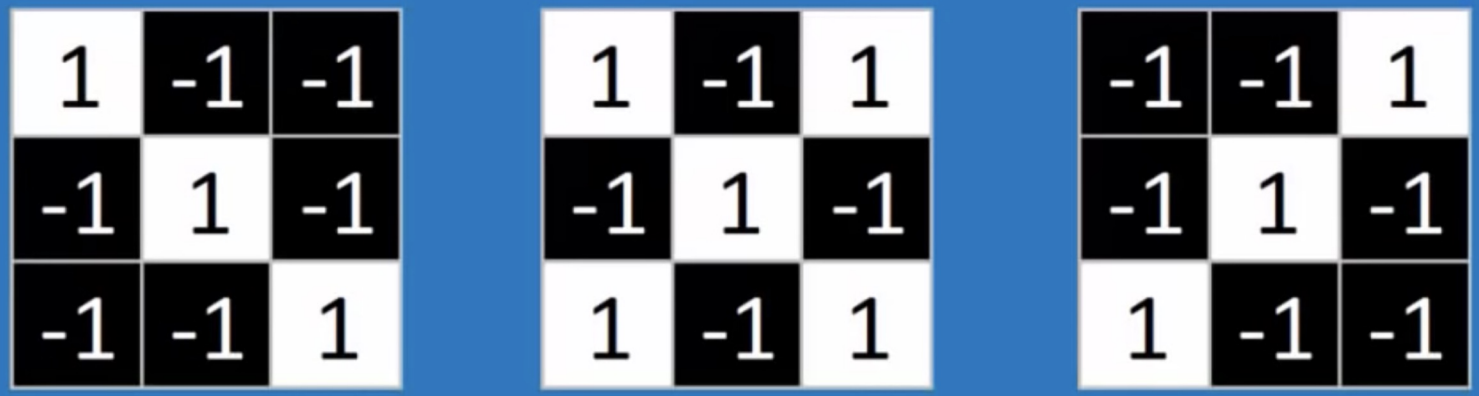
Naive comparison

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	X	-1	-1	-1	-1	X	X	-1
-1	X	X	-1	-1	X	X	-1	-1
-1	-1	X	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	X	-1	-1
-1	-1	X	X	-1	-1	X	X	-1
-1	X	X	-1	-1	-1	-1	X	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

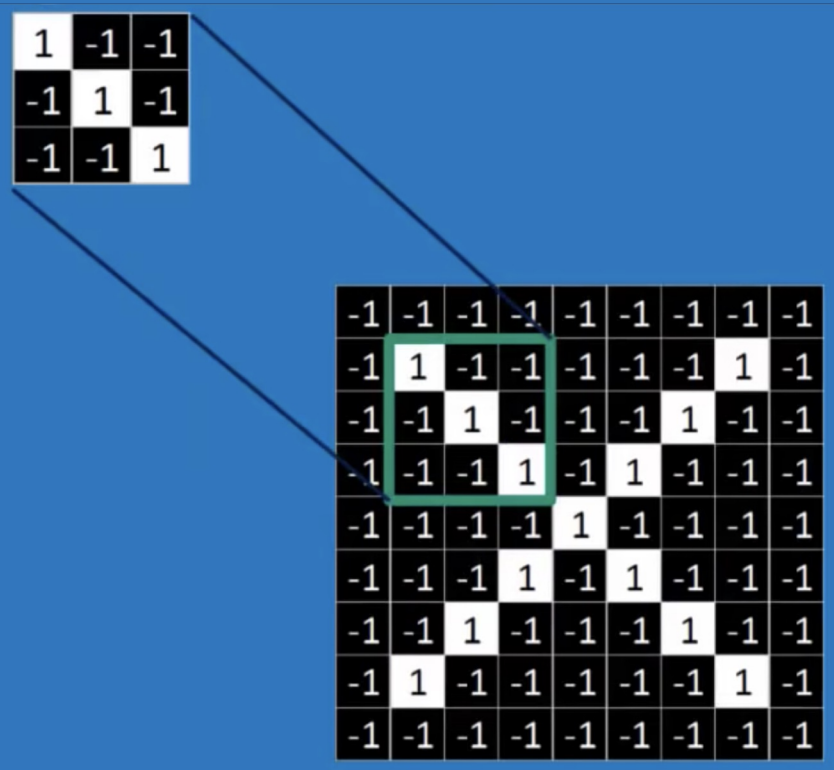
Intuition



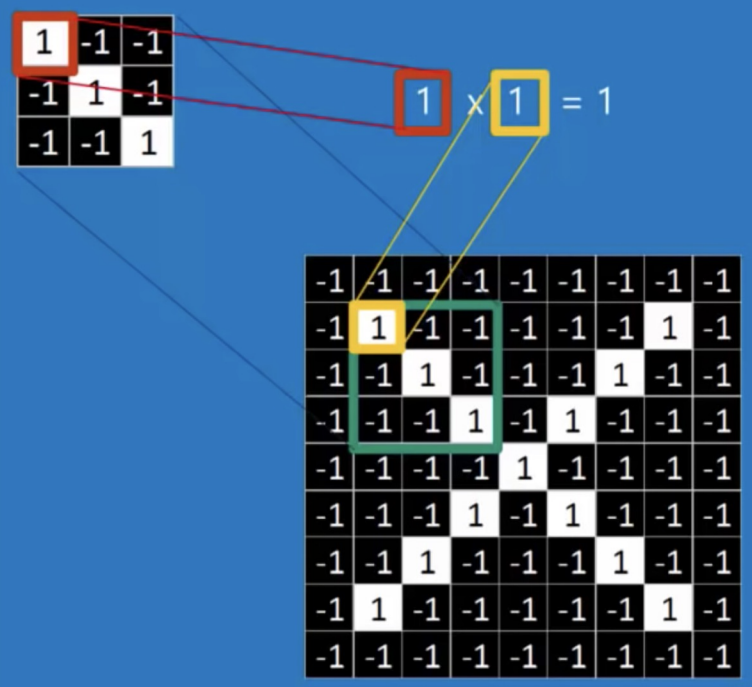
Potential local features



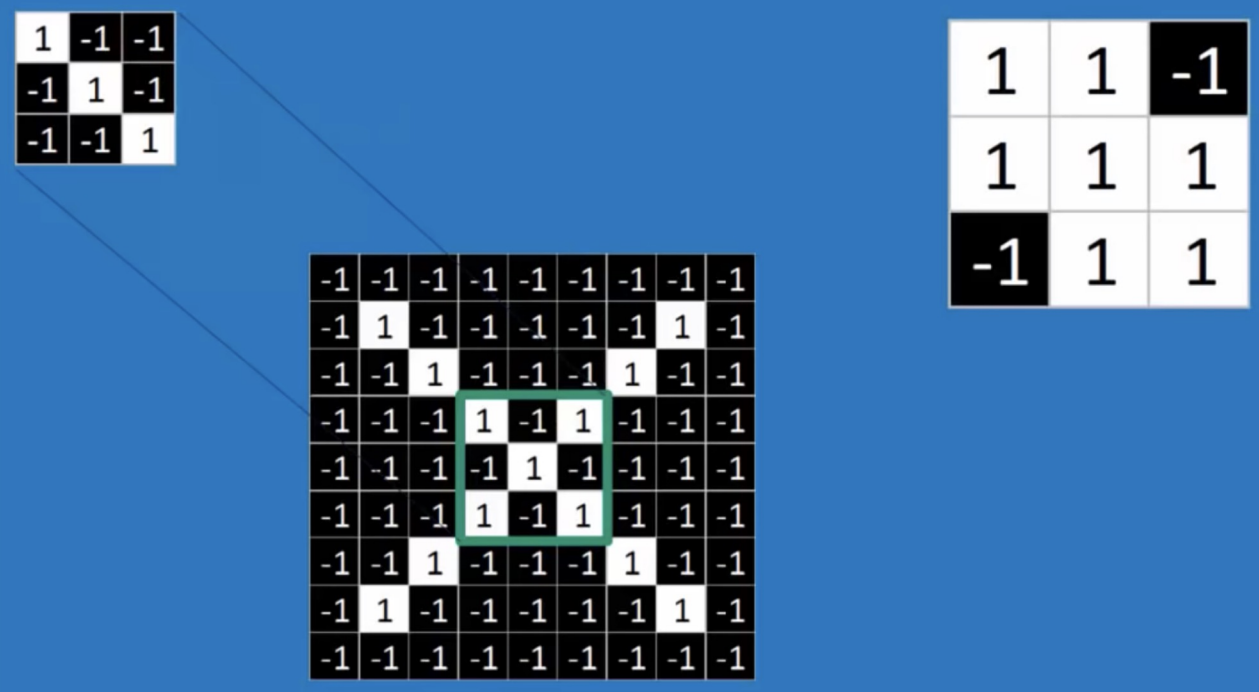
Setting


1 -1 -1
-1 1 -1
-1 -1 1
-1 -1 -1 -1 -1 -1 -1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 1 -1 -1 -1 1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 -1 -1 1 -1 -1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 1 -1 -1 -1 1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1

Local filtering



And at different locations



Convolution

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

$\otimes$

1	-1	-1
-1	1	-1
-1	-1	1

$=$

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

This is 2D convolution

- Convolution can also be 1D and 3D
- Convolutions can be used in other application areas
  - Whenever there’s predictable correlation over time/space
  - Language, protein or DNA sequences

A convolution exercise

$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & -2 & 0 & -2 \end{bmatrix}$$

Here's our "image"

$$\begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix}$$

Here's our "filter"

A convolution exercise

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix}$$

$$\begin{bmatrix} 2 & \cdot \\ \cdot & \cdot \end{bmatrix}$$

$$\left(0 \times \frac{-1}{2}\right) + (1 \times 1) + \left(0 \times \frac{-1}{2}\right)$$

$$\left(0 \times \frac{-1}{2}\right) + (1 \times 1) + \left(0 \times \frac{-1}{2}\right)$$

$$\left(1 \times \frac{-1}{2}\right) + (1 \times 1) + \left(1 \times \frac{-1}{2}\right) = 2$$

A convolution exercise

$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & \cdot & \cdot & \cdot \end{bmatrix} \quad \begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} 2 & -2 \\ \cdot & \cdot \end{bmatrix}$$

A convolution exercise

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & -2 & 0 \end{bmatrix} \qquad \begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} 2 & -2 \\ -1 & \cdot \end{bmatrix}$$

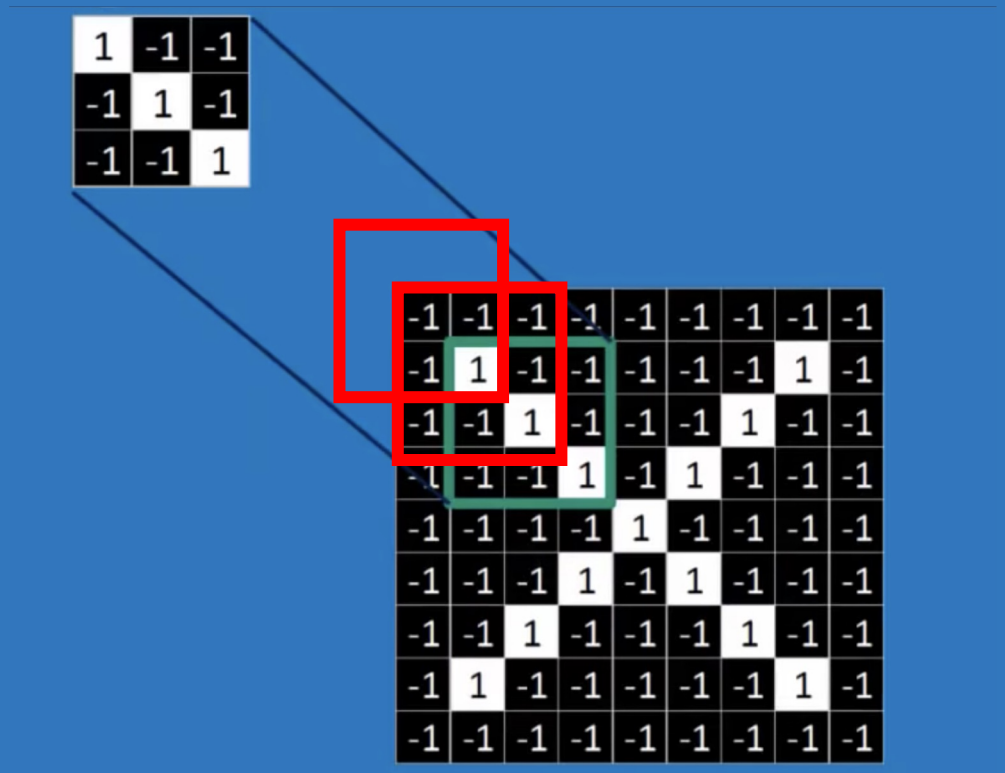
A convolution exercise

$$\begin{bmatrix} 0 & & & \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & -2 & 0 & -2 \end{bmatrix} \quad \begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} 2 & -2 \\ -1 & 1 \end{bmatrix}$$

What about the edges of an image?

JOSH HAROLDSON VIA FLICKR // CC BY-NC 2.0

Padding


1 -1 -1
-1 1 -1
-1 -1 1
-1 -1 -1 -1 -1 -1 -1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 1 -1 -1 -1 1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 -1 -1 1 -1 -1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 1 -1 -1 -1 1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1

Padding

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

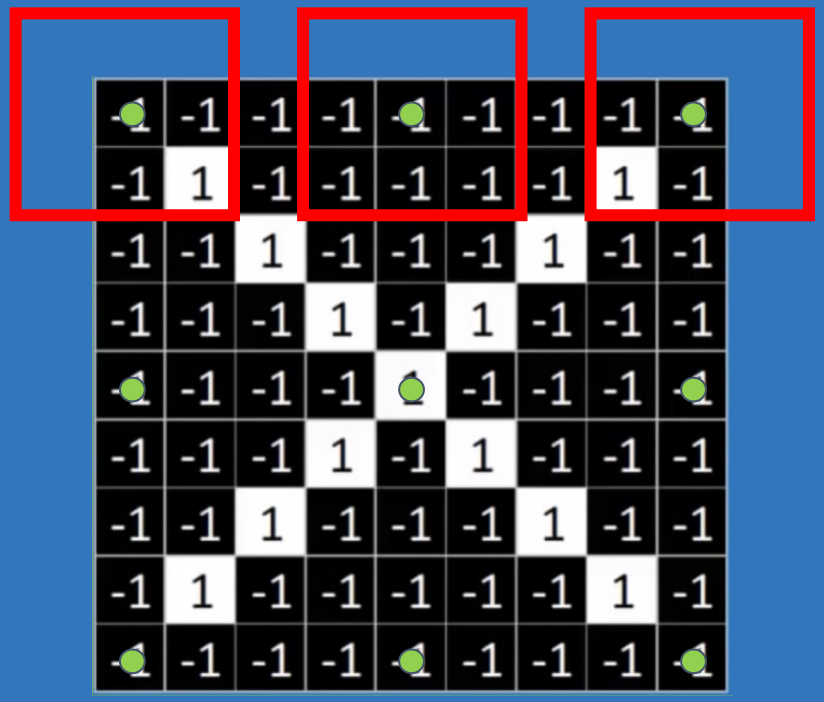
Stride 2

-1 -1 -1 -1 -1 -1 -1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1

Stride 3

-1 -1 -1 -1 -1 -1 -1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 1 -1 -1 -1 1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 -1 -1 1 -1 -1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 1 -1 -1 -1 -1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1

Stride 4


-1 -1 -1 -1 -1 -1 -1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 1 -1 -1 -1 1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 -1 -1 1 -1 -1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 1 -1 -1 -1 1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1

Motivation: conv for edge detection

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

Horizontal lines

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

Vertical lines

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

45 degree lines

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

135 degree lines

Horizontal lines:

Feature detection - edges

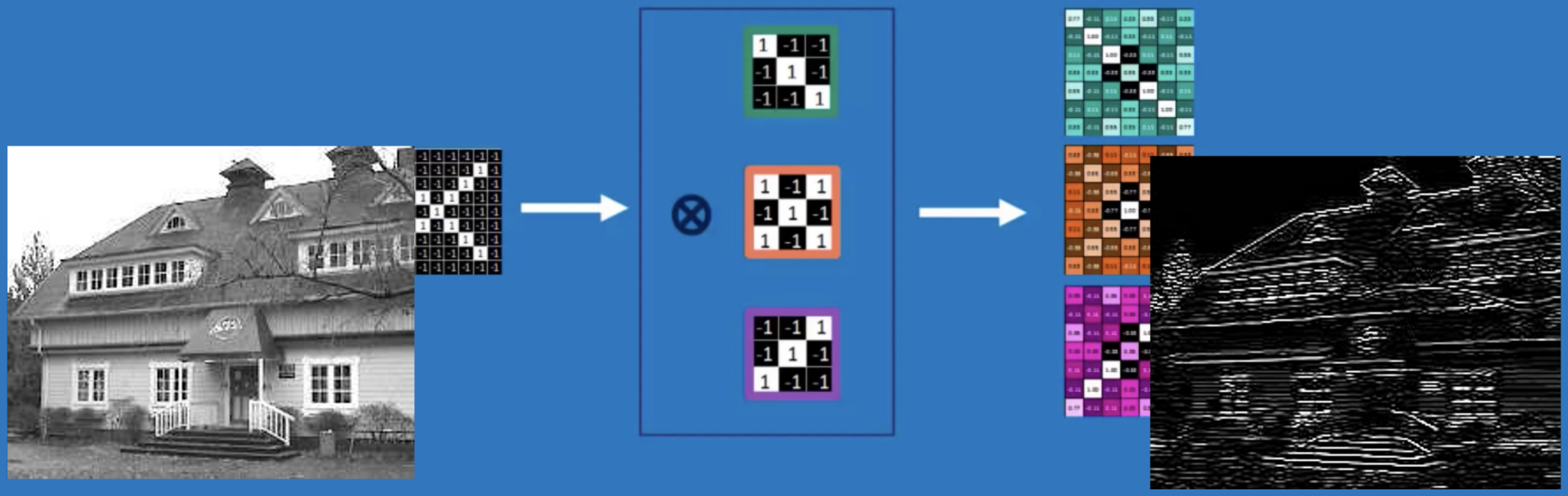
Filters give us **global** translation invariance

Run a filter across an image

How could that be useful for image recognition?

Multiple filters

A convolution layer



Potential to add a ReLU

ReLU(x) = max(0,x)

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	0.77

Pooling = Local invariance

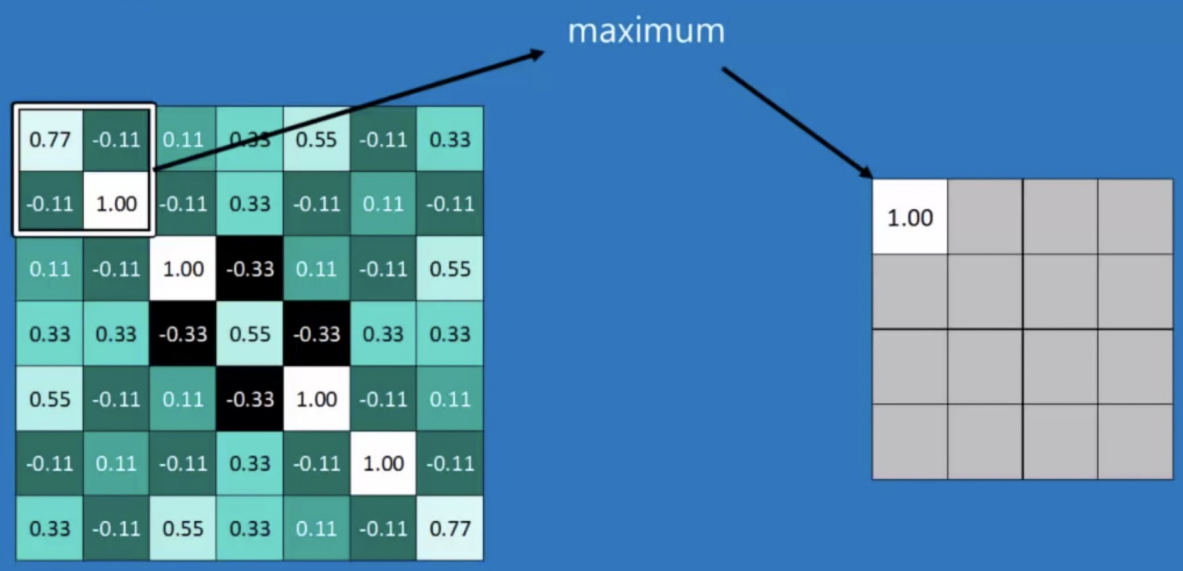
- Recall: filters give us global invariance
- Pooling gives local invariance

The need for some invariance

Features may appear in (slightly) different places

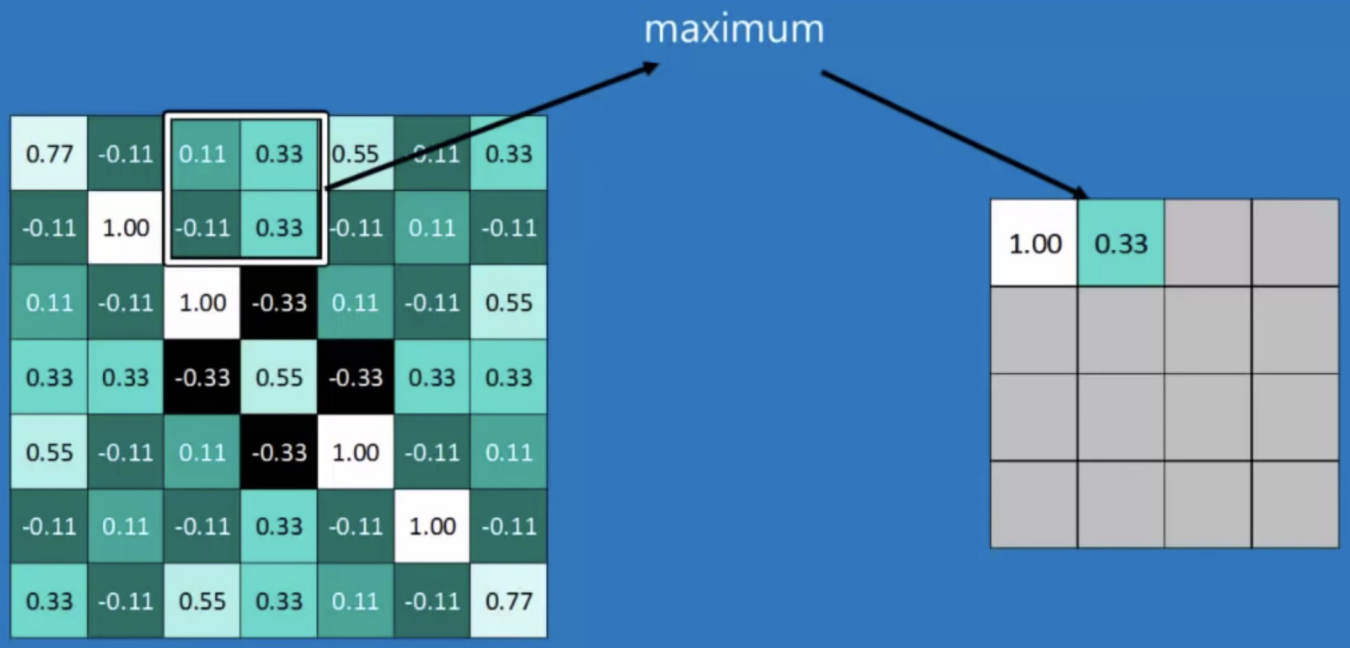
JOSH HAROLDSON VIA FLICKR // CC BY-NC 2.0

Max-pooling

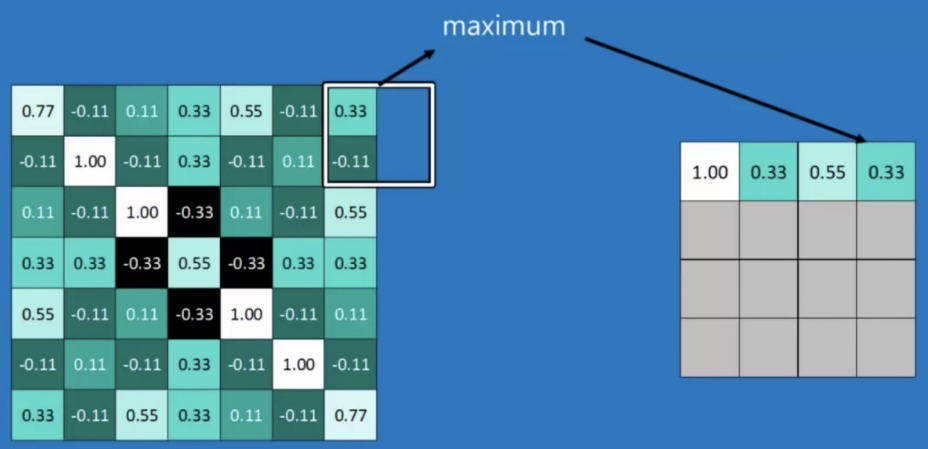


2*2, Stride 2

Across space



Again, Padding problem!



And a whole max pooling operation

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

max pooling

1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

How many parameters?

Hold up, how does this reduce the # of params?

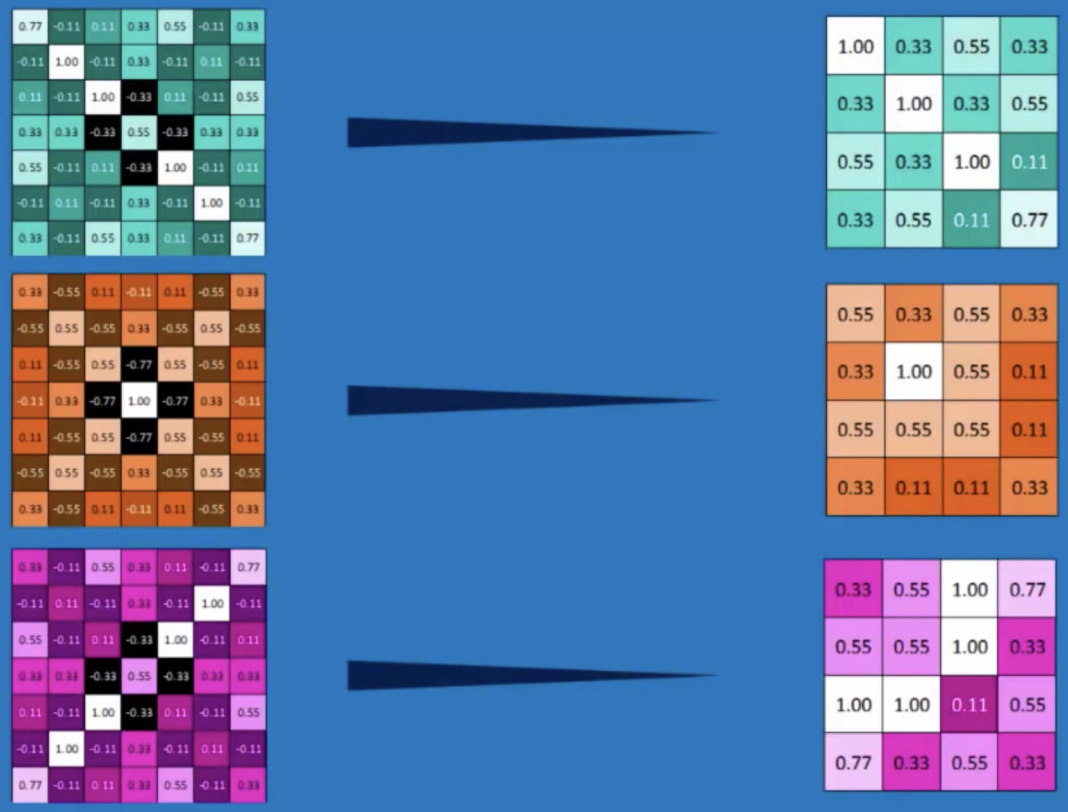
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

max pooling

1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

Next layer has fewer connections!

Bunch of Filters

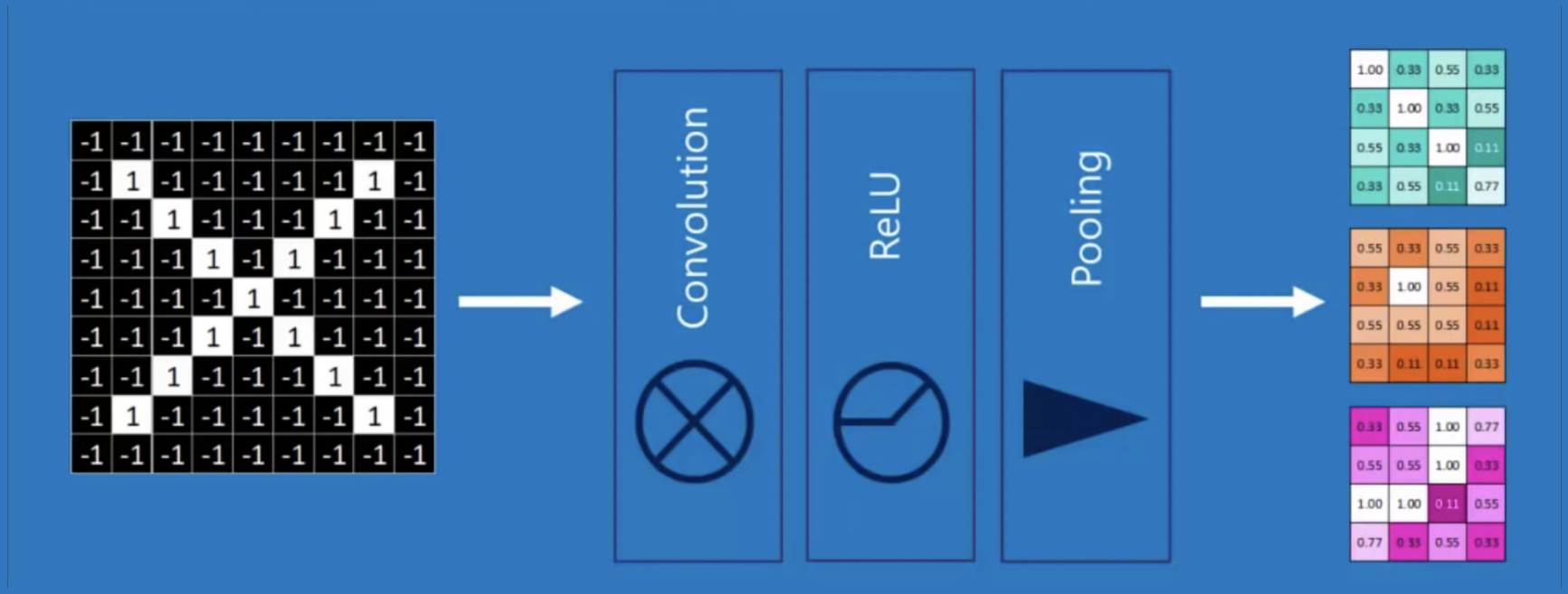


Pooling layers

- Want to:
  - Reduce dimensions of data, without additional parameters
  - Introduce some ***local*** translation invariance
- Pooling operation can be max(0,x) or average(x)

Section 4: Putting it all together

So far: not much of a hierarchy



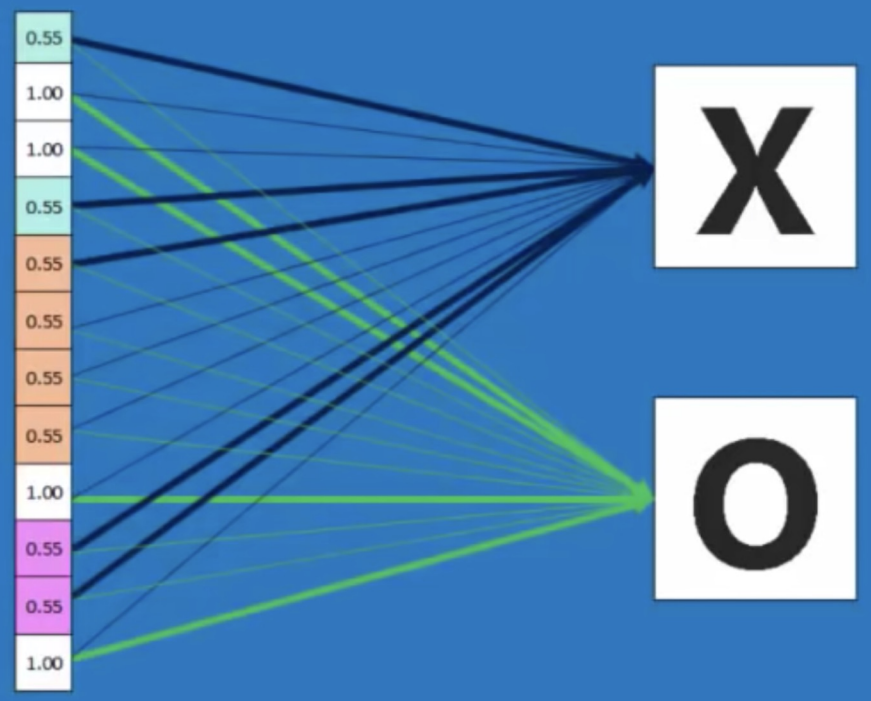
Chain it

-1 -1 -1 -1 -1 -1 -1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 1 -1 -1 -1 1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 -1 -1 1 -1 -1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 1 -1 -1 -1 1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1
Convolution
ReLU
Convolution
ReLU
Pooling
Convolution
ReLU
Pooling
1.00 0.55
0.55 1.00
1.00 0.55
0.55 0.55
0.55 1.00
1.00 0.55

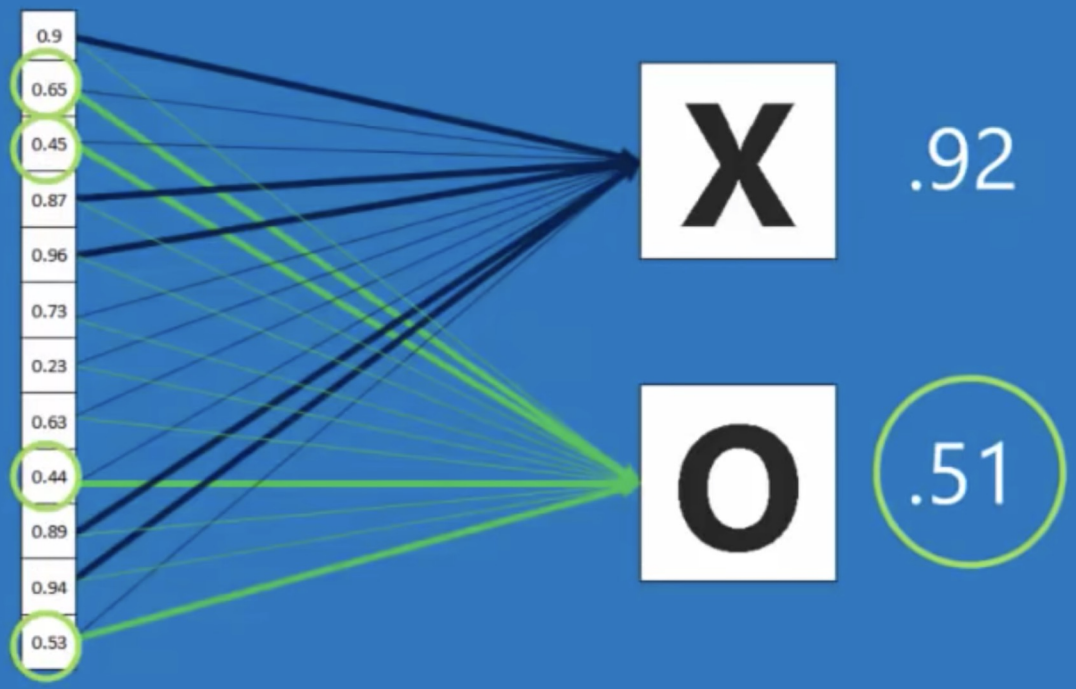
How to transition to fully connected

Fully connected read out

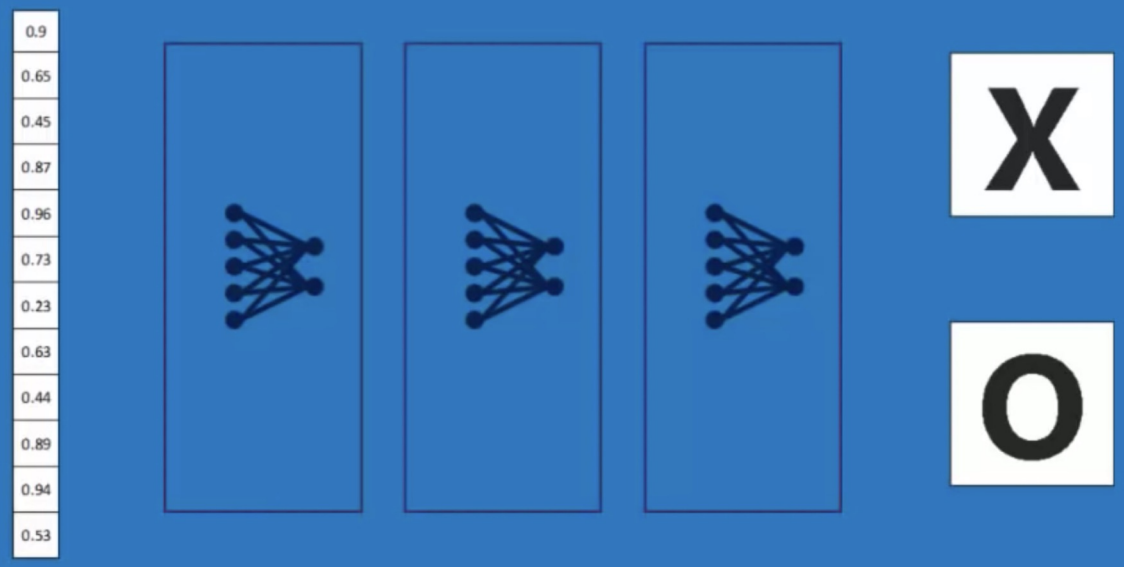
Predict both



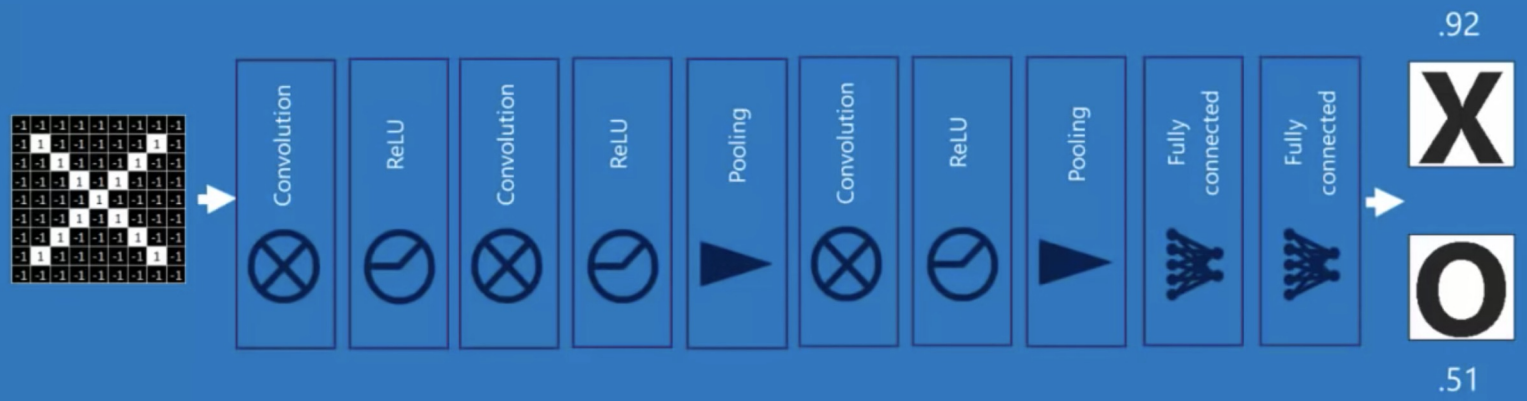
Get activations



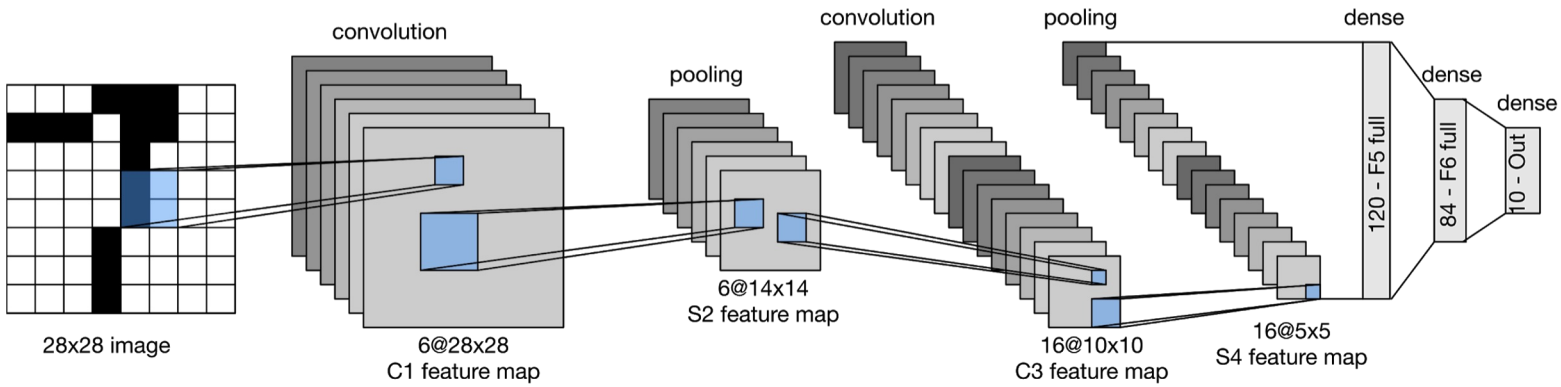
Multiple layers of fully connected



Now stack it all together



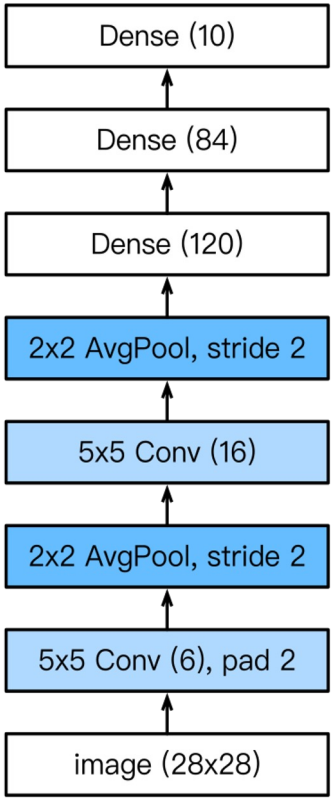
Putting things together



Input→(Conv→ReLU→Pooling)→...→Fully connected

From ‘Dive into deep learning’

Another way of visualizing the same network



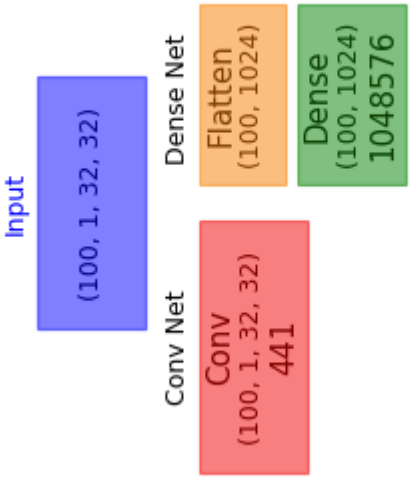
Let us calculate that: Convolution and pooling

- 1*1 image with 2 padding is 5*5
- 5*5 image, 5*5 filter => 1 output
- 300*400 image with 2 padding is 304*404
- 304*404 image, 5*5 filter =>300*400
- Maxpool 2x2 pooling window 300*400, stride 2 =150*200

Dense vs conv layers

- Let's compare the number of parameters in networks that use dense (fully connected) vs convolutional layers

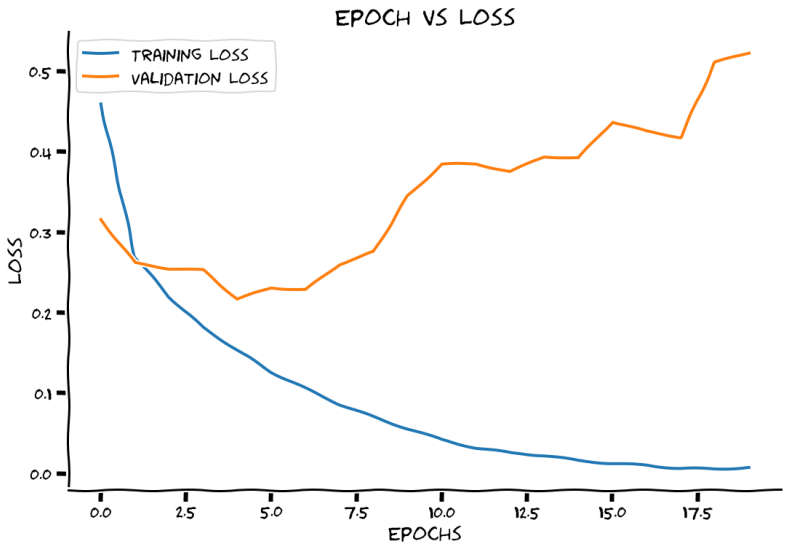
Total Parameters in Dense Layer 1,048,576
Total Parameters in Conv Layer 441

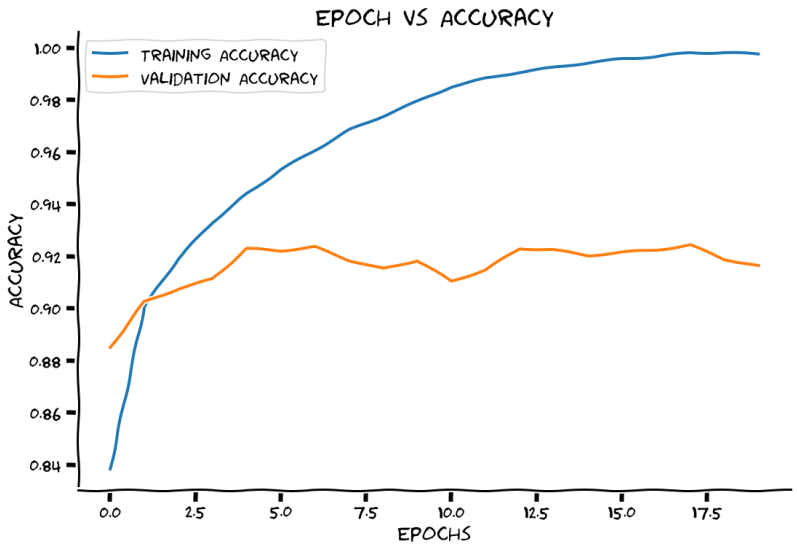


https://colab.research.google.com/drive/1_YP5SicR49OrlWpct0IrrAI5r4LsTDVz?usp=sharing

Overfitting

Below are strong signs of overfitting.





What can we do? Dropout

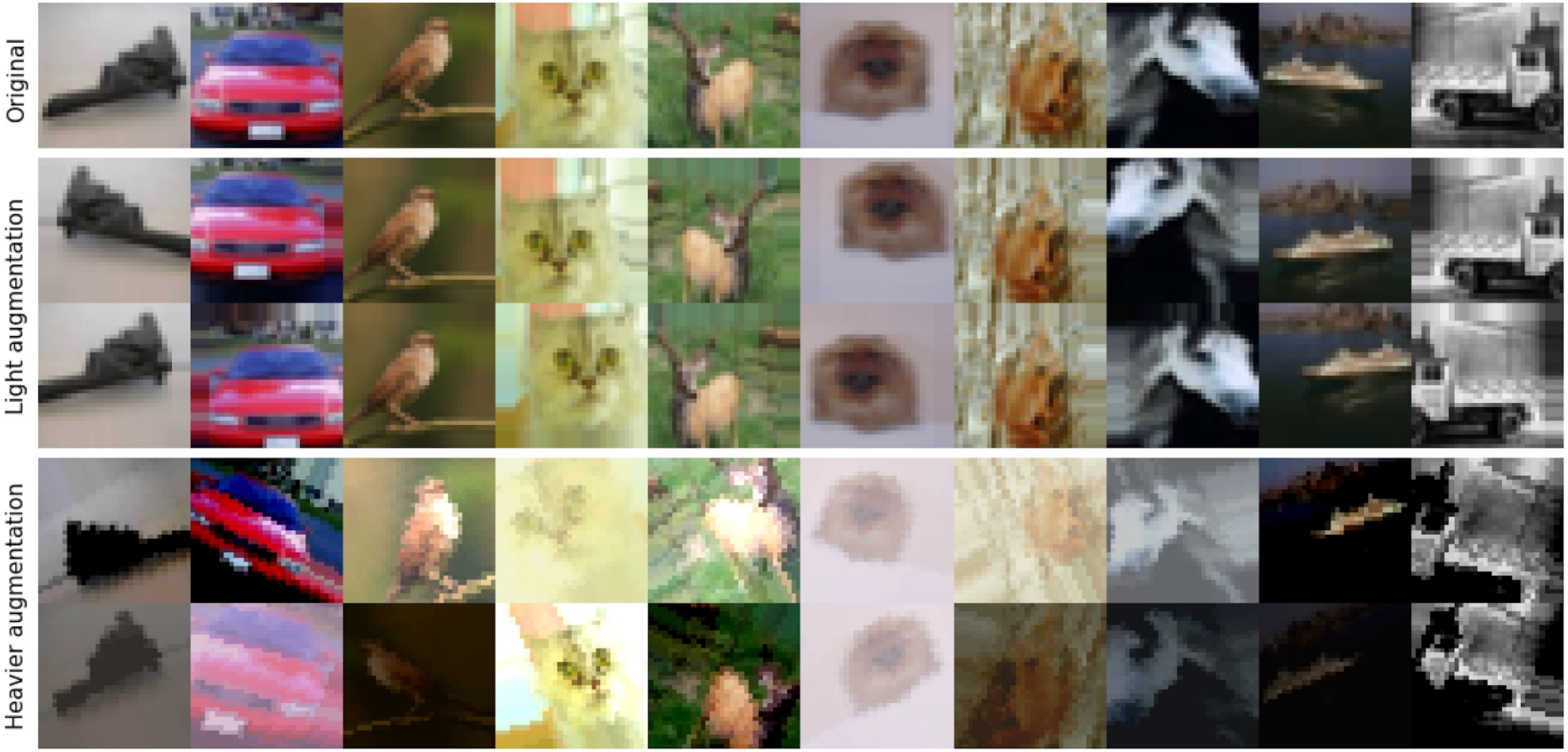
Intuition: no one neuron should be *too* important
Just like the brain, there should be redundancy in connections

Dropout: randomly set some of the hidden activations to 0
Parameter: % of neurons to set to 0

What can we do? Data augmentation (DA)

source: Hernandez Garcia Thesis

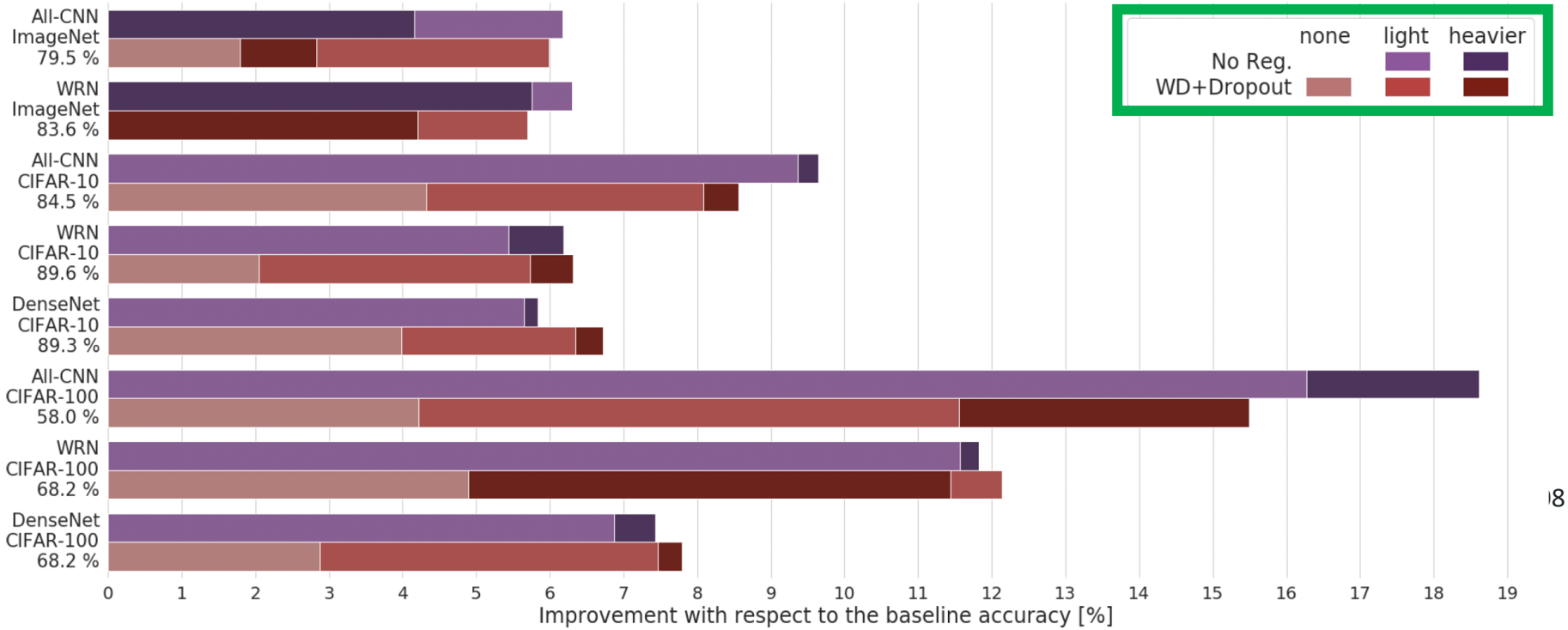
Different strengths of DA



.07

source: Hernandez Garcia, arxiv
https://arxiv.org/pdf/1806.03852.pdf

Data augmentation is often more important than direct regularization



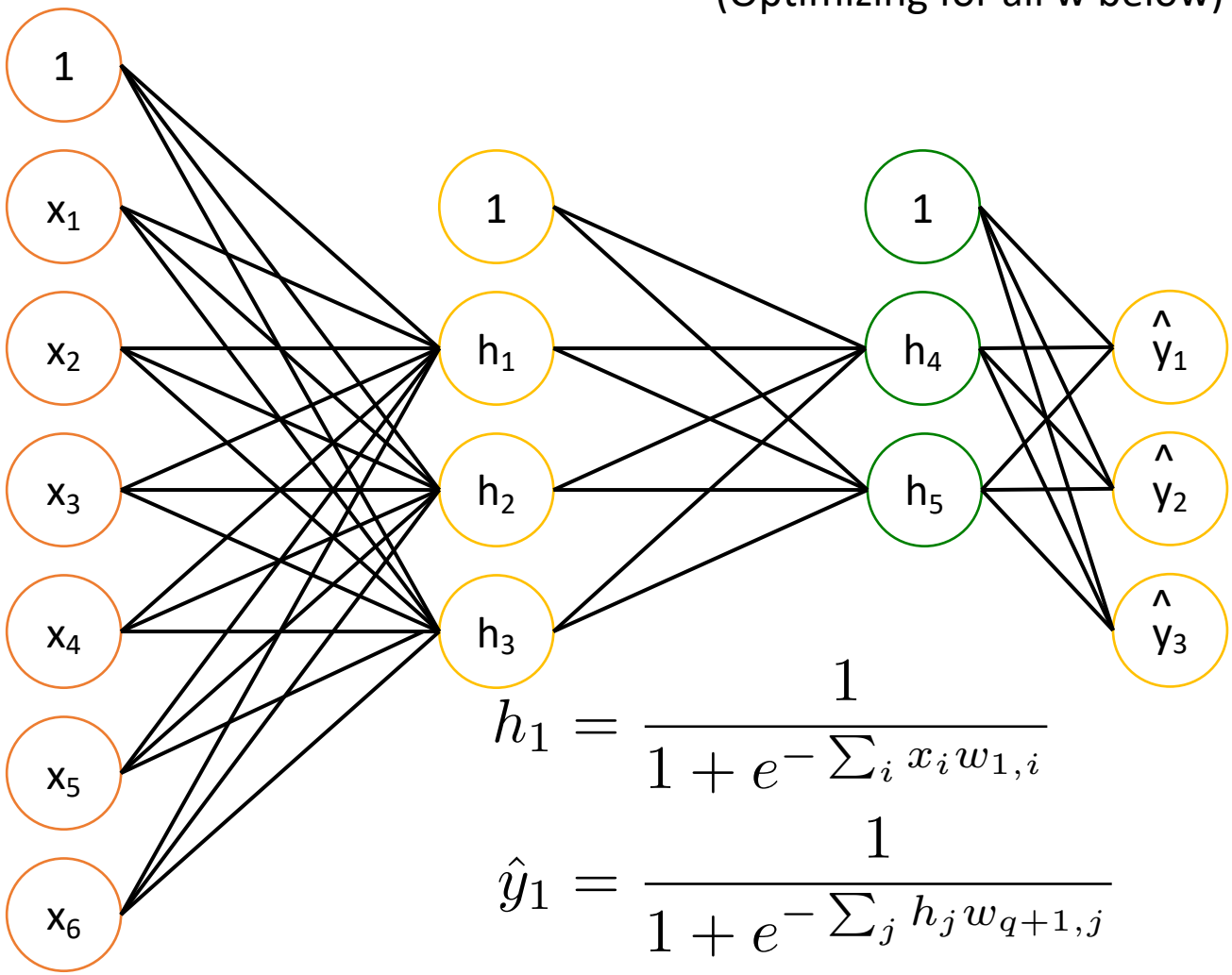
source: Hernandez Garcia, arxiv

Interpretability

- Chris Olah has a great series on interpretability (the distill ones are particularly good)
  - https://colah.github.io/

Next time: Back Prop

(Optimizing for all w below)



$$h_1 = \frac{1}{1 + e^{-\sum_i x_i w_{1,i}}}$$

$$\hat{y}_1 = \frac{1}{1 + e^{-\sum_j h_j w_{q+1,j}}}$$

q is total # of hidden nodes across all layers

Resources

- Programming resources for training your own NNs
  - Tensorflow https://www.tensorflow.org/
  - Keras https://keras.io/
  - Pytorch https://pytorch.org/
  - For intuition: http://playground.tensorflow.org/
- Short course on deep learning (Nando De Freitas)
  - https://www.youtube.com/playlist?list=PLjK8ddCbDMphIMSXn-w1IjyYpHU3DaUYw
- Commentary on AlphaGo
  - https://www.youtube.com/watch?v=UMm0XaCFTJQ
  - https://www.youtube.com/watch?v=g-dKXOlsf98
- Other fun videos
  - Geoff Hinton is in this one! Neural Net stuff is towards the end
    - https://www.youtube.com/watch?v=yxxRAHVtafI
  - Fei Fei Li's Ted Talk (Creator of ImageNet)
    - https://www.ted.com/talks/fei_fei_li_how_we_re_teaching_computers_to_understand_pictures?language=en