

CSE/STAT 416

Convolutional Neural Networks

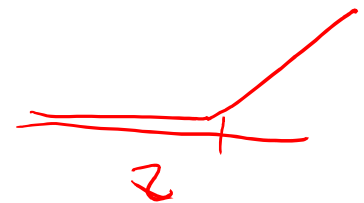
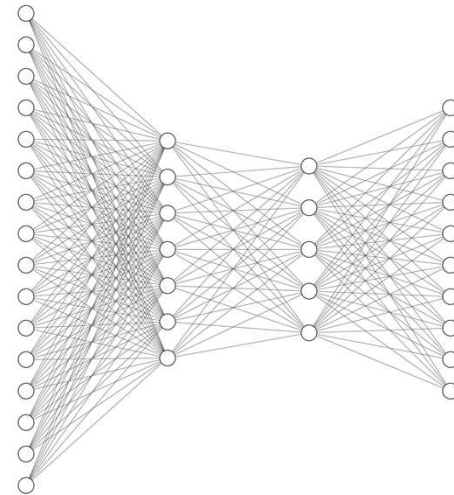
Hunter Schafer
University of Washington
Aug 7, 2019



Deep Learning

A lot of the buzz about ML recently has come from recent advancements in **deep learning**.

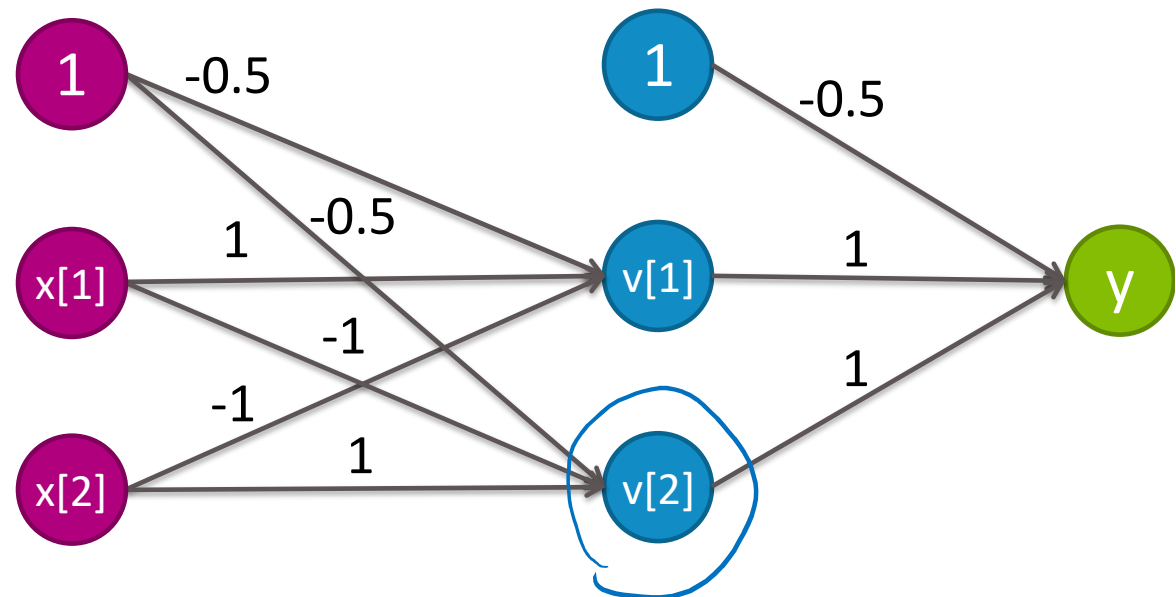
When people talk about “deep learning” they are generally talking about a class of models called **neural networks** that are a loose approximation of how our brains work.



XOR

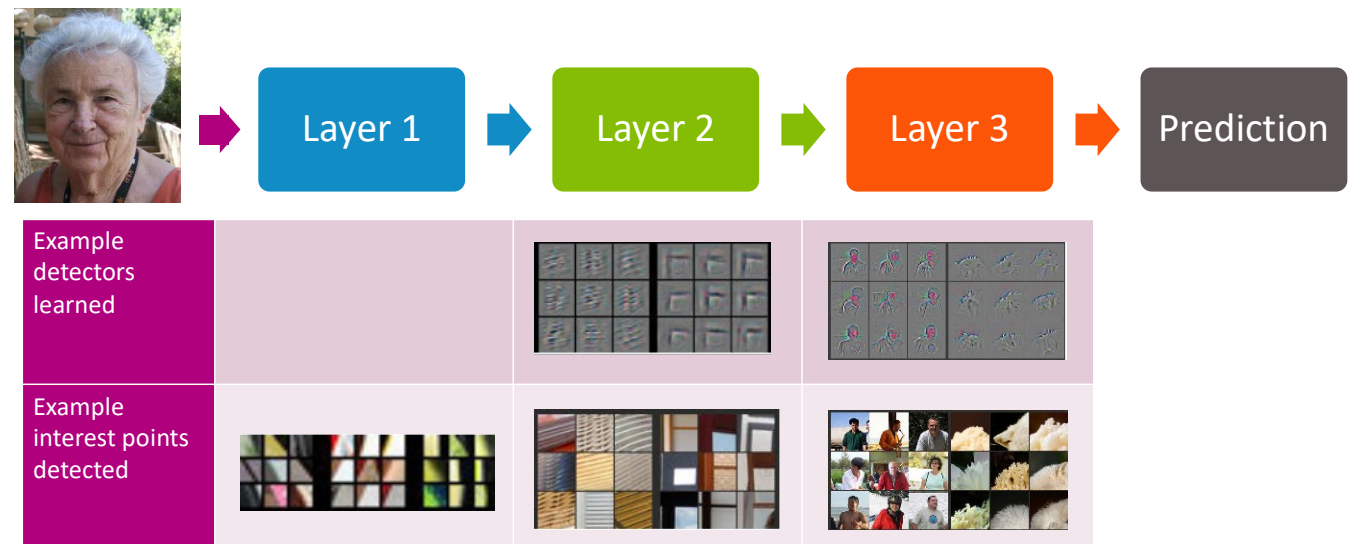
Notice that we can represent

$$x[1] \text{ XOR } x[2] = (x[1] \text{ AND } \neg x[2]) \text{ OR } (\neg x[1] \text{ AND } x[2])$$



NN to the Rescue

Neural Networks implicitly find these low level features for us!



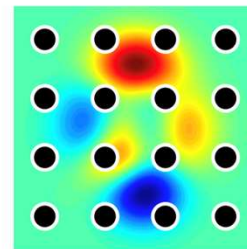
[Zeiler & Fergus '13]

Each layer learns more and more complex features

Hyperparameter Optimization

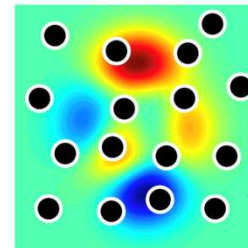
How do we choose hyperparameters to train and evaluate?

Grid search:



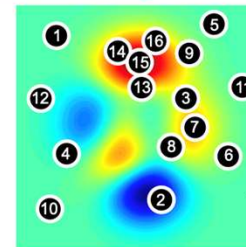
Hyperparameters on 2d uniform grid

Random search:



Hyperparameters randomly chosen

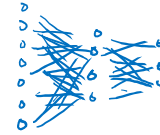
Bayesian Optimization:



Hyperparameters **adaptively** chosen

Image Challenges

Approach #1 : Flatten image



Images are extremely high dimensional

- CIFAR-10 dataset are very small: 3@32x32
 - # inputs:

$$3 \cdot 32 \cdot 32 = 3072$$

- For moderate sized images: 3@200x200
 - # inputs:

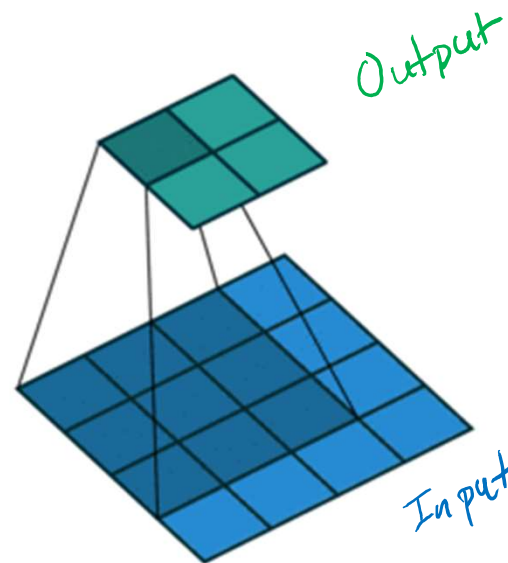
$$3 \cdot 200 \cdot 200 = 120,000$$

Images are structured, we should leverage this

Convolutional Neural Networks

Idea: Reduce the number of weights that need to be learned by looking at local neighborhoods of image.

Use the idea of a **convolution** to reduce the number of inputs by combining information about local pixels.



Convolution

Use a **kernel** that slides across the image, computing the sum of the element-wise product between the kernel and the overlapping part of the image

Image

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

Kernel

0	1	2
2	2	0
0	1	2

$$\begin{pmatrix} 3 & 3 & 2 \\ 0 & 0 & 1 \\ 3 & 1 & 2 \end{pmatrix} \odot \begin{pmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 3 & 4 \\ 0 & 0 & 0 \\ 0 & 1 & 4 \end{pmatrix}$$

sum
→ 12

Convolution

The input image (blue), the kernel (dark blue, numbers lower right) slide over the image to produce a result (green)

3 ₀	3 ₁	2 ₂	1	0
0 ₂	0 ₂	1 ₀	3	1
3 ₀	1 ₁	2 ₂	2	3
2	0	0	2	2
2	0	0	0	1

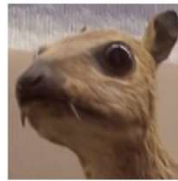
12	12	17
10	17	19
9	6	14

Special Kernels

The numbers in the kernels determine special properties

Identity

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



Edge Detection

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$



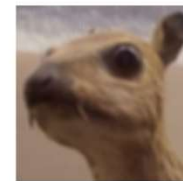
Sharpen

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



Box Blur

$$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

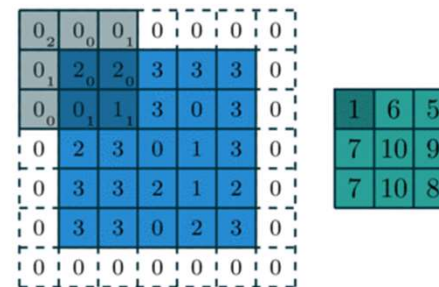


More Convolutions

You can specify a few more things about a kernel

- Kernel dimensions and values
- Padding size and padding values
- Stride (how far to jump) values

For example, a 3x3 kernel applied to a 5x5 image with 1x1 zero padding and a 2x2 stride



Poll Everywhere

Think 

1.5 min

pollev.com/cse416

What is the result of applying a convolution using this kernel on this input image?

Use 1x1 zero padding and a 2x2 stride

Image

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Kernel

1	1
0	2

1:30

Poll Everywhere

Pair 

3 min

pollev.com/cse416

What is the result of applying a convolution using this kernel on this input image?

Use 1x1 zero padding and a 2x2 stride

Image

0	0	0	0	0	0
0	1	2	3	4	0
0	5	6	7	8	0
0	9	10	11	12	0
0	13	14	15	16	0
0	0	0	0	0	0

Kernel

1	1
0	2

Result size: 3x3

$$\begin{pmatrix} 2 & 6 & 0 \\ 23 & 35 & 8 \\ 13 & 29 & 16 \end{pmatrix}$$

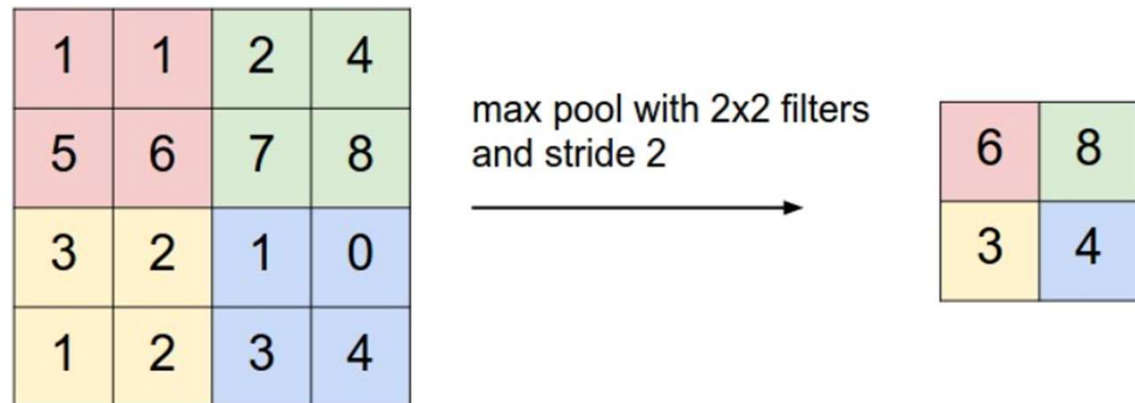

Pooling

Another core operation that is similar to a convolution is a **pool**.

- Idea is to down sample an image using some operation
- Combine local pixels using some operation (e.g. max, min, average, median, etc.)

Typical to use **max pool** with 2x2 filter and stride 2

- Tends to work better than average pool

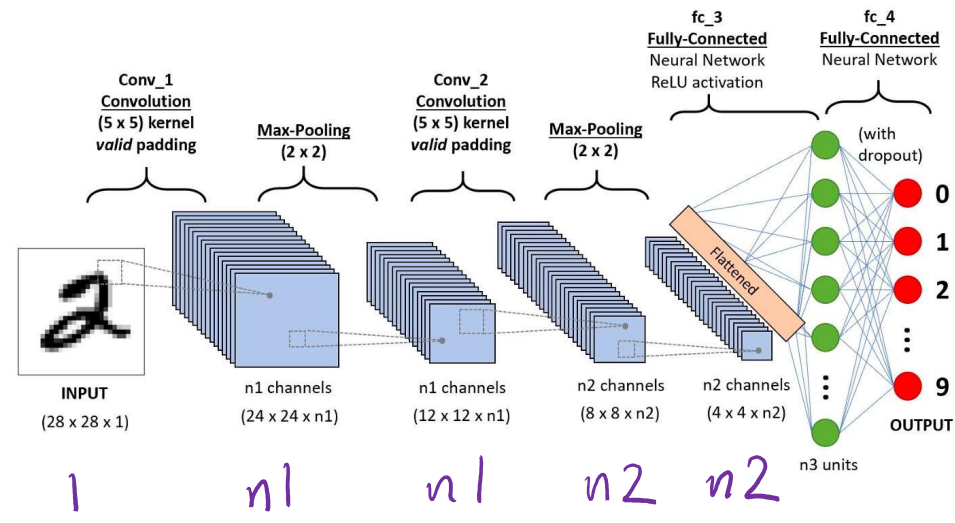


Convolutional Neural Network

Combine convolutions and pools into pre-processing layers on image to learn a smaller, information dense representation.

Example architecture for hand-written digit recognition

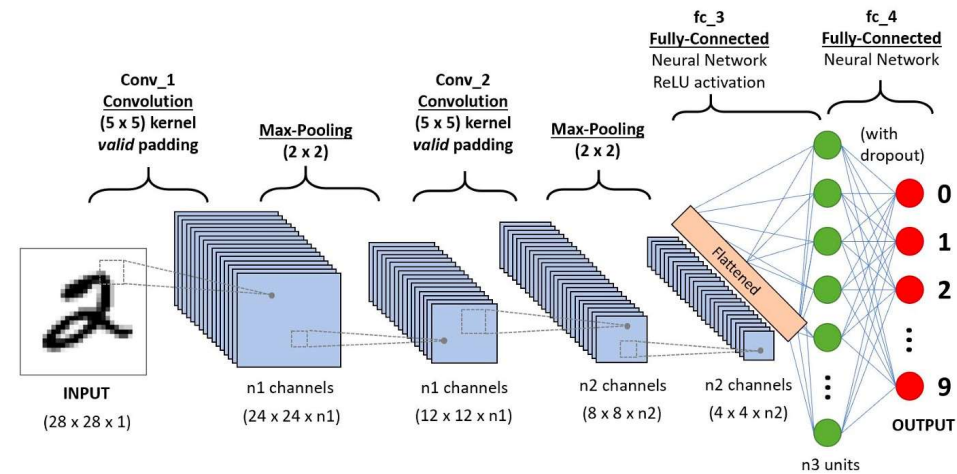
- Each convolution section uses many different kernels (increasing depth of channels)
- Pooling layers down sample each channel separately
- Usually ends with fully connected neural network



Convolutional Neural Network

Why does this help?

- For each channel in a convolution layer (Conv_1), it uses the same kernel (with same values) to all sub-regions
 - This is called weight-sharing
 - Gives efficiency + shift invariance
- Pooling helps reduce the number of inputs by “blurring” the image without losing too much info.





Brain Break

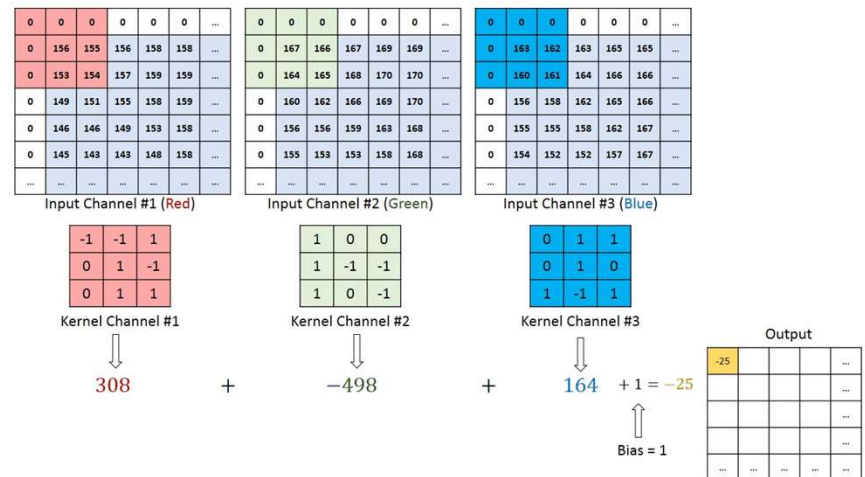
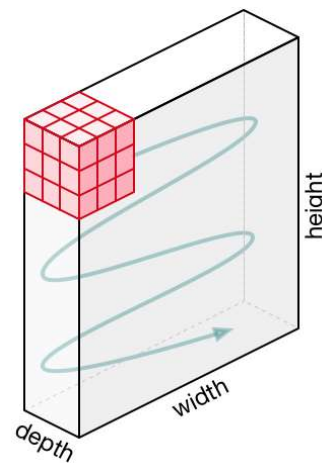
10:36



CNN with Color Images

How does this work if there is more than one input channel?

- Usually, use a 3 dimensional **tensor** as the kernel to combine information from each input channel

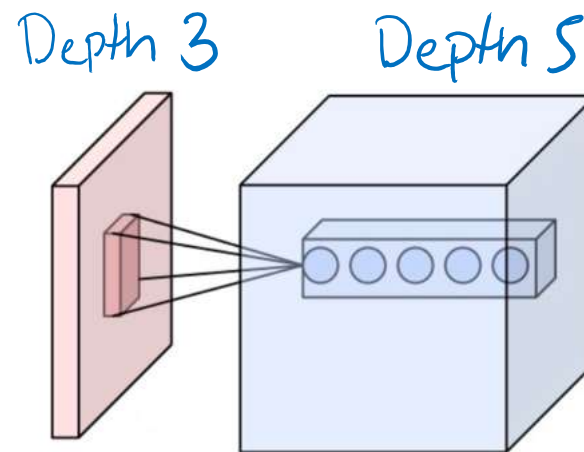


CNN with Color Images

Another way of thinking about this process is each kernel is a neuron that looks at the kernel-size pixels in a neighborhood

If there are 5 output channels in a conv layer, only need to learn the weights for the 5 neurons

- These neurons are a bit different since they look at the pixels that overlap with the window at each position.



Poll Everywhere

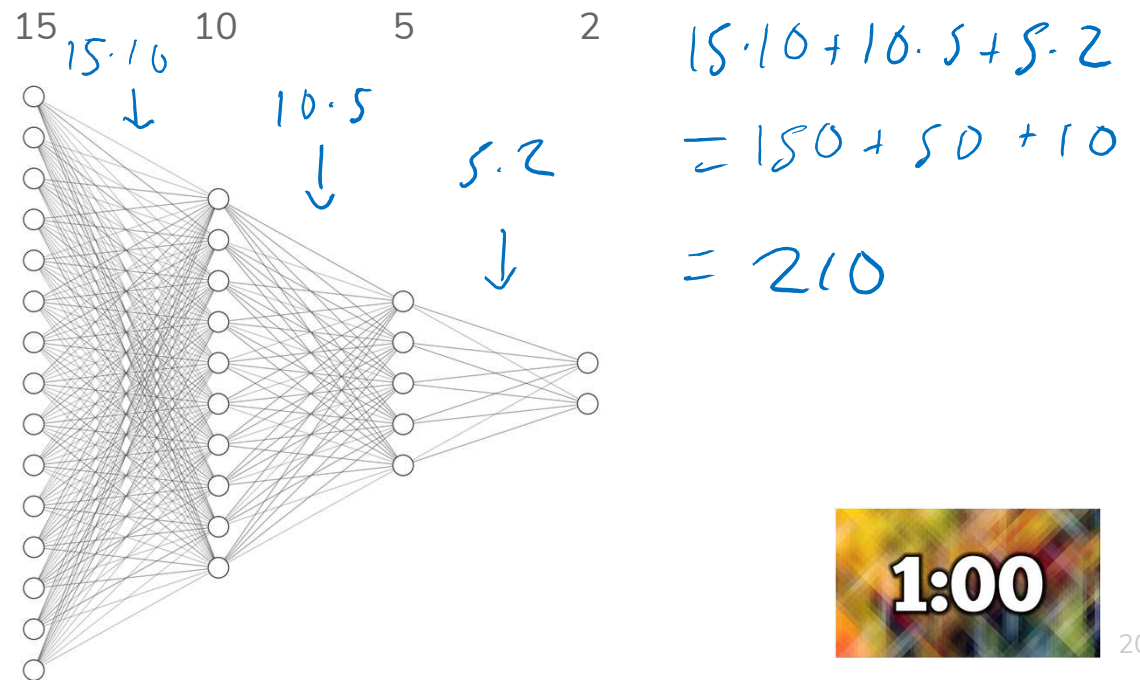
Think 

1 min

pollev.com/cse416

Consider a plain neural network below, how many weights need to be learned?

Assume the image already includes the intercept units



1:00

Poll Everywhere

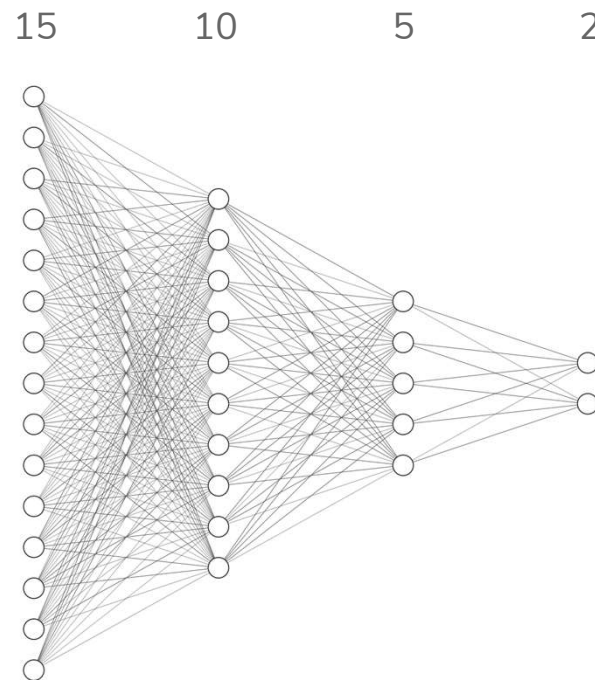
Pair 

2 min

pollev.com/cse416

Consider a plain neural network below, how many weights need to be learned?

Assume the image already includes the intercept units



Weight Sharing

Want to use NN w/ 84 hidden neurons, 10 outputs
 Consider solving this digit recognition task. Suppose I wanted to use a hidden layer with 84 neurons

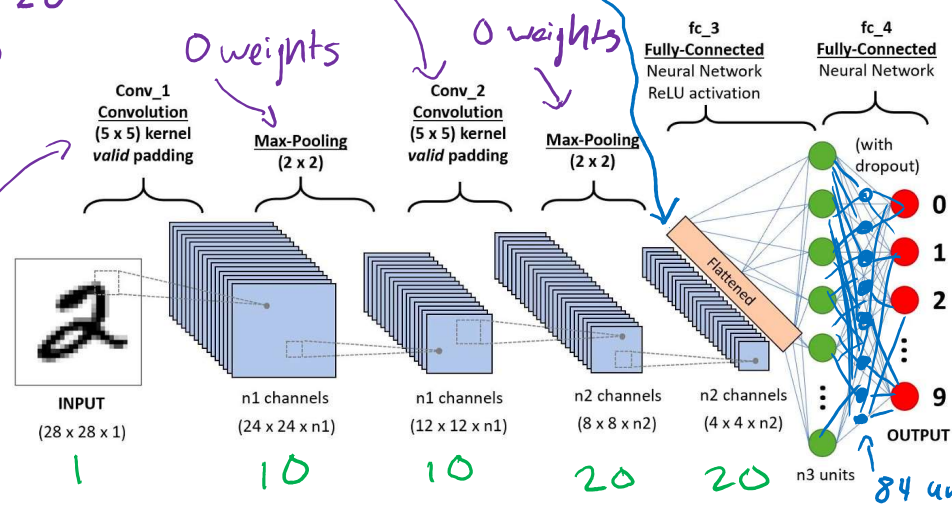
Without Convolutions: # inputs: $28 \cdot 28 \cdot 1 = 784$
 Number Weights: $28 \cdot 28 \cdot 1 \cdot 84 + 84 \cdot 10 = 66,696$
 (That's a lot of weights) # Fully Connected Weights

With Convolutions (assume $n1=10, n2=20$)
 # Flattened Features: $4 \cdot 4 \cdot 20 = 320$
 $4 \cdot 4 \cdot 20 \cdot 84 + 84 \cdot 10 = 27,720$

Conv-2
 $5 \cdot 5 \cdot 10 \cdot 20 = 5000$

Conv-1
 $5 \cdot 5 \cdot 1 \cdot 10 = 250$

Depth

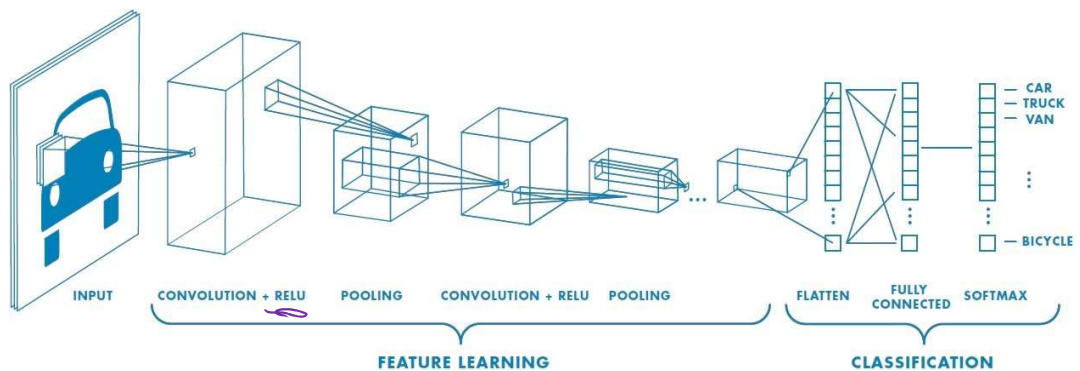


Total weights
 $250 + 5000 + 27,720 = 32,970 << 66k$

General CNN Architecture

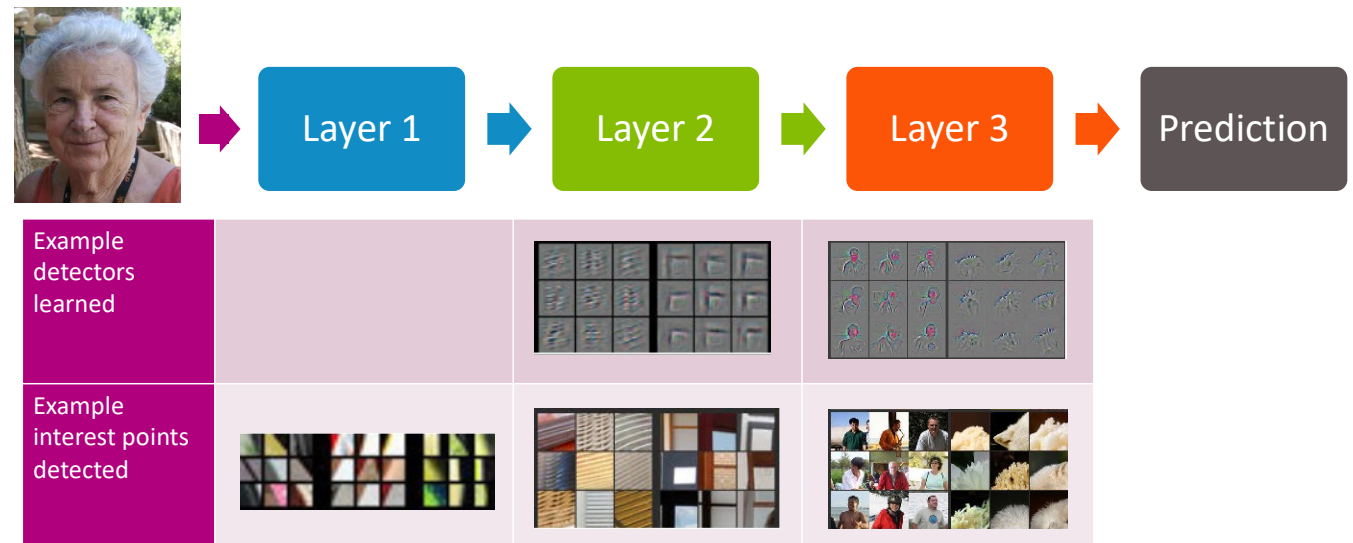
CNNs generally (not always) have architectures that look like the following

- A series of Convolution + Activation Functions and Pooling layers. It's very common to do a pool after each convolution.
- Then after some number of these operations, flatten the image to work with the final neural network



Features

The learned kernels are exactly the “features” for computer vision!
They start simple (corners, edges) and get more complex after more layers



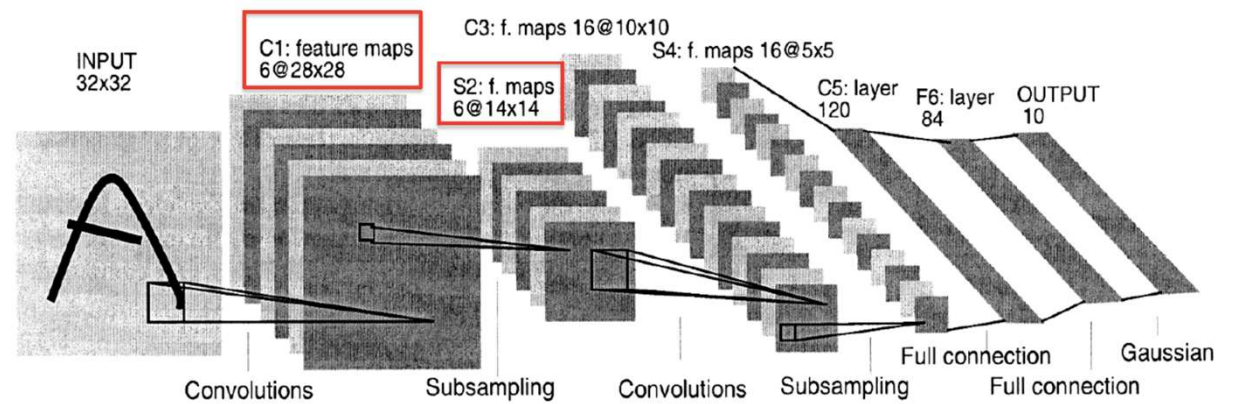
[Zeiler & Fergus '13]

CNN Success

MNIST

CNNs have had remarkable success in practice

LeNet, 1990s



CNN Success

LeNet made 82 errors on MNIST (popular hand-written digit dataset).



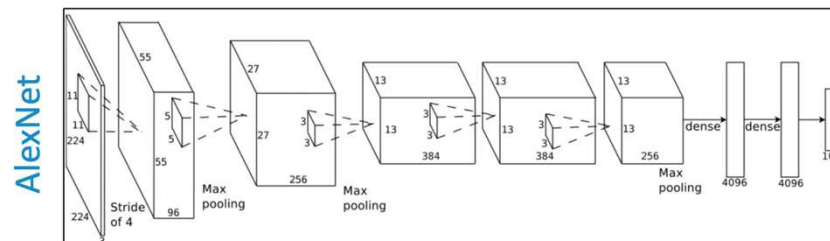
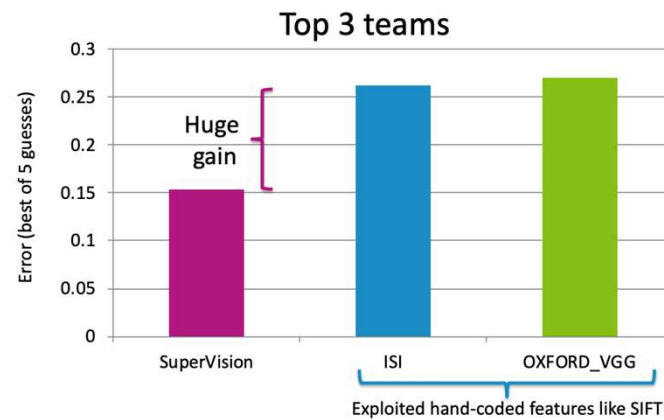
CNN Success

ImageNet 2012 competition:

- 1.2M training images
- 1000 categories

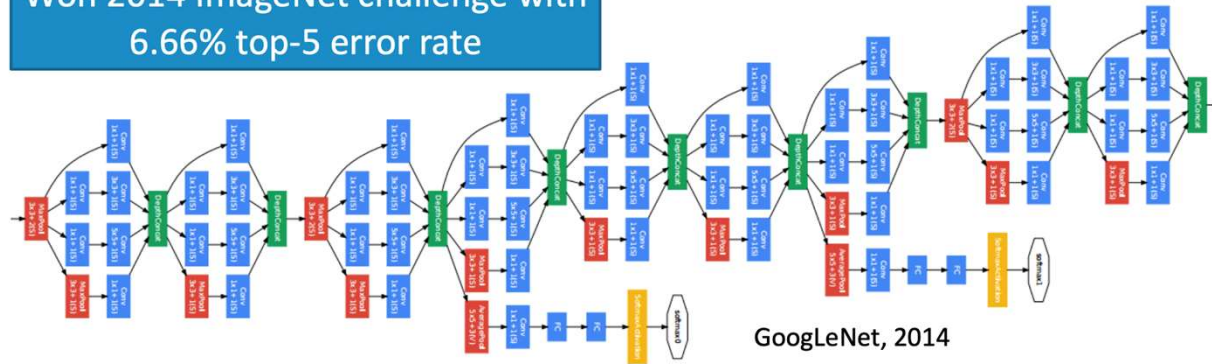
Winner: SuperVision

- 8 layers, 60M parameters [Krizhevsky et al. '12]
- Top-5 Error: 17%



CNN Success

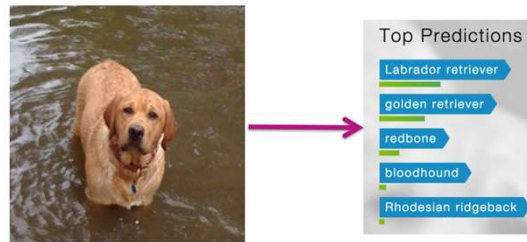
Won 2014 ImageNet challenge with 6.66% top-5 error rate



Huge CNN depth has proven helpful in recognition systems... Maybe because images contain hierarchical structure (faces contain eyes contain edges, etc.)

Applications

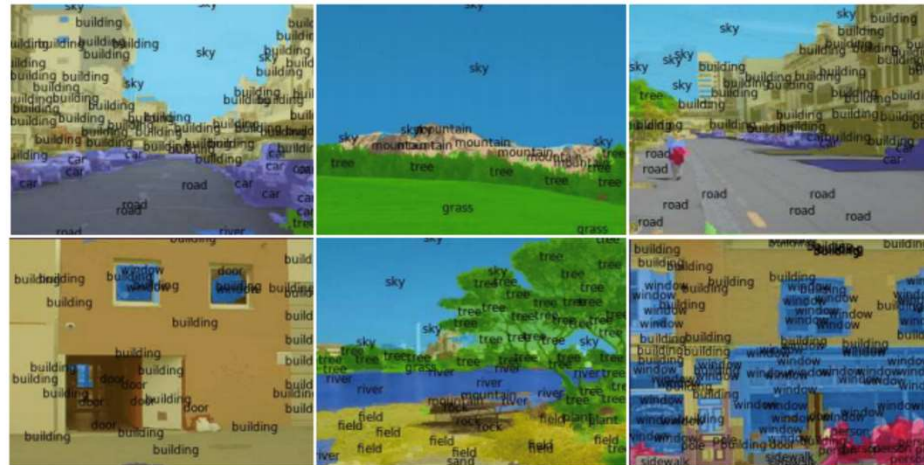
Image Classification



Input: x
Image pixels

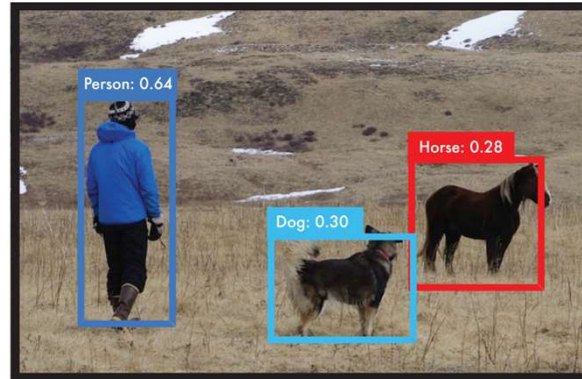
Output: y
Predicted object

Scene Parsing [Farabet et al. '13]



Applications

Object Detection [Redmon et al. 2015] (<http://pjreddie.com/yolo/>)



~~Product Recommendation [Hebert et al. 2012]~~ Product Recommendation





Brain Break



Deep Learning in Practice

Pros

No need to manually engineer features, enable automated learning of features

Impressive performance gains

- Image processing
- Natural Language Processing
- Speech recognition

Making huge impacts in most fields

Cons

Requires a LOT of data

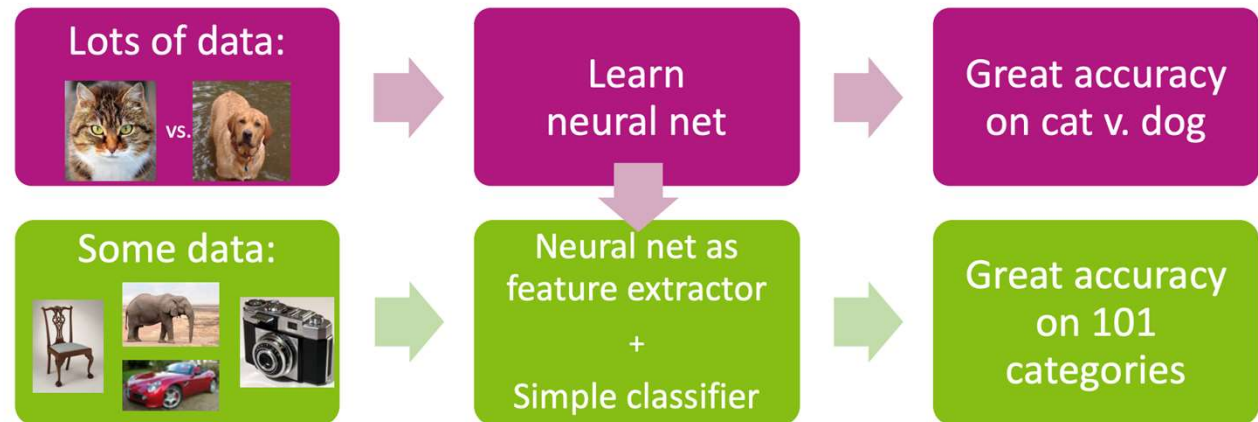
Computationally really expensive

Hard to tune hyper-parameters

- Choice of architecture (we've added even more hyper-parameters)
- Learning algorithm

Still not very interpretable

A Tale of 2 Tasks



If we don't have a lot of data for Task 2, what can we do?

Idea: Use a model that was trained for one task to help learn another task.

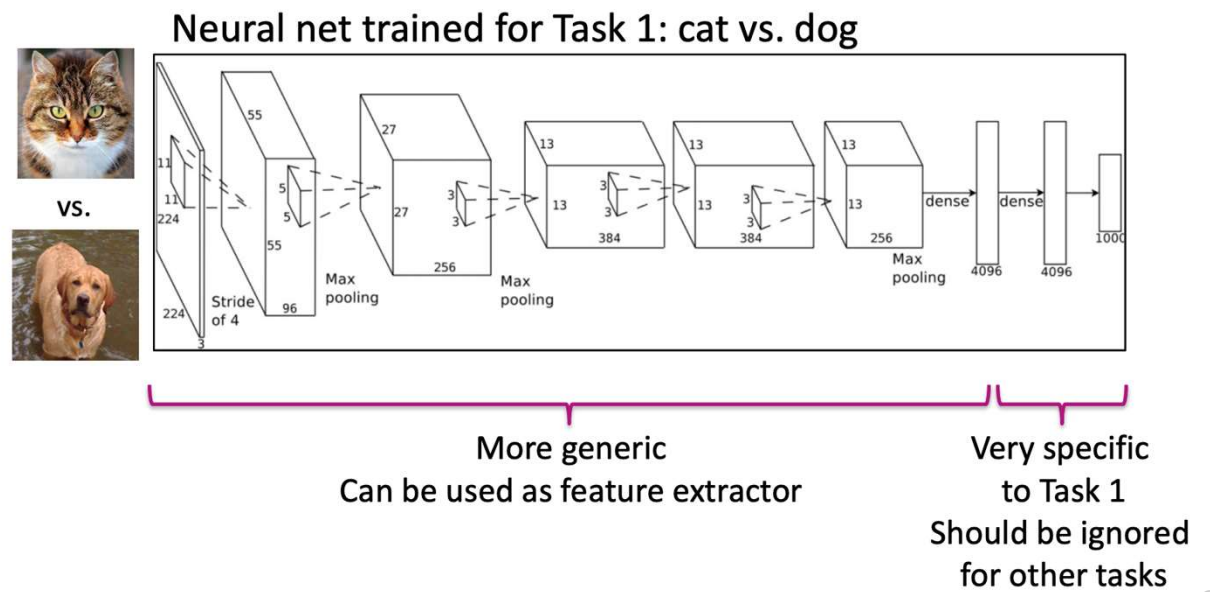
- An old idea, explored for deep learning by Donahue et al. '14 & others

CNNs

What is learned in a neural network?

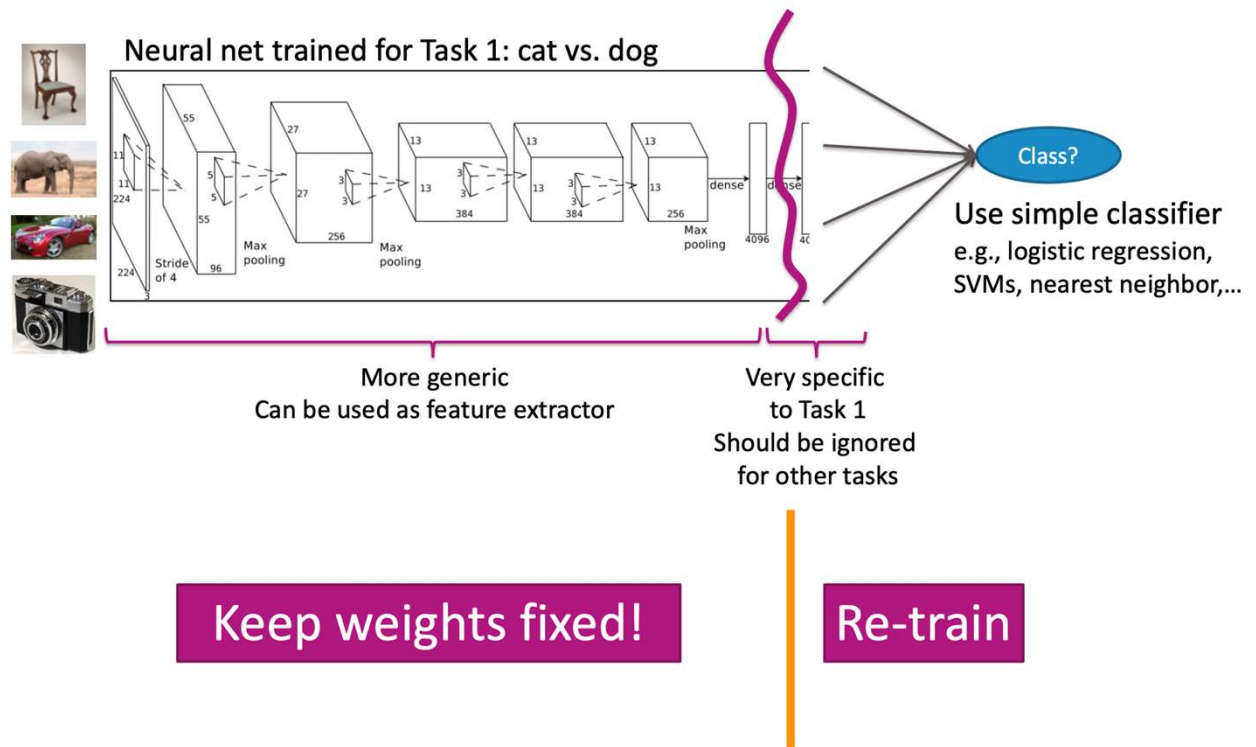
Initial layers are low-level and very general.

- Usually not sensitive/specific to the task at hand



Transfer Learning

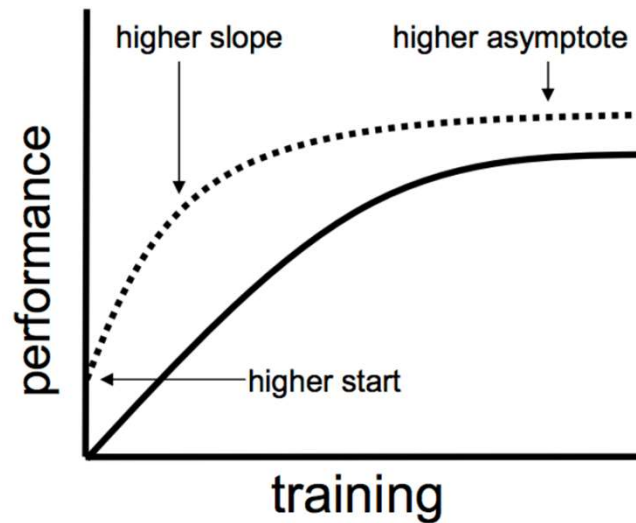
Share the weights for the general part of the network



Transfer Learning

If done successfully, transfer learning can really help. Can give you

- A higher **start**
- A higher **slope**
- A higher **asymptote**



..... with transfer
— without transfer

NN Failures

While NNs have had amazing success, they also have some baffling failures.



“panda”

57.7% confidence

“No one adds noise to things in real applications”

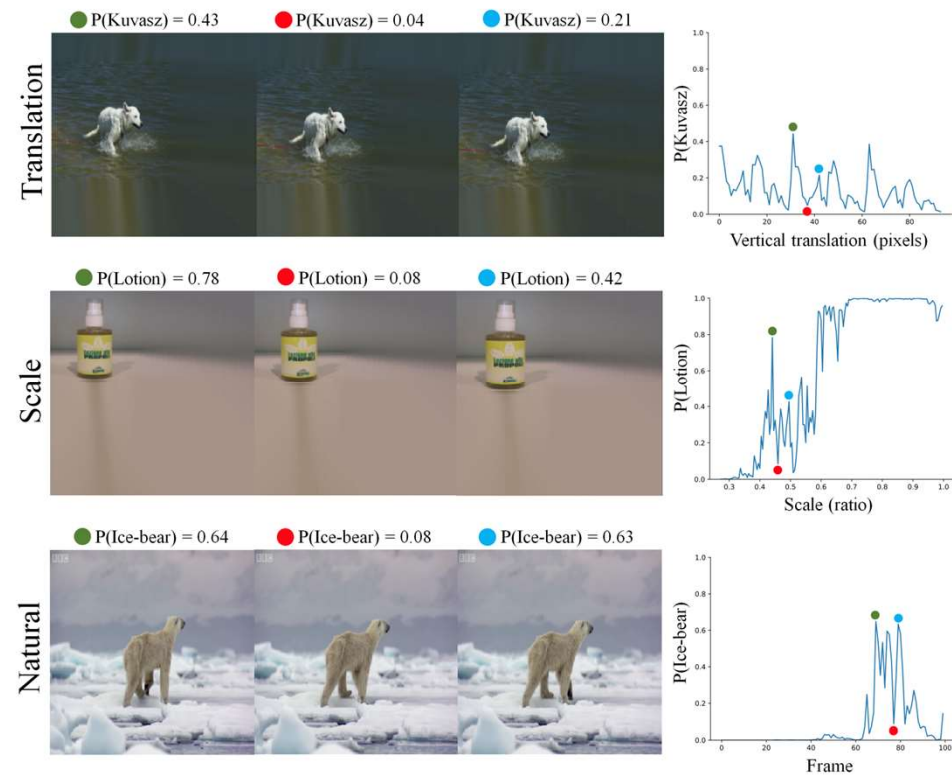
Not true!

- Hackers will hack
- Sensors (cameras) are noisy!

NN Failures

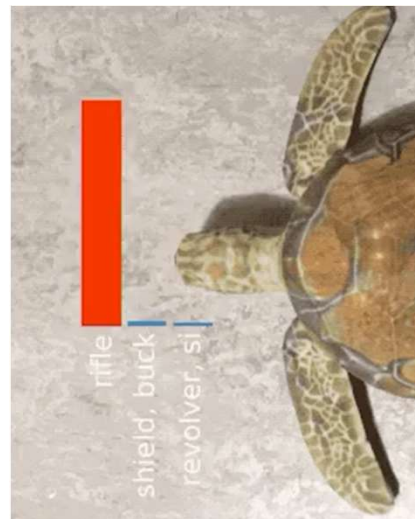
They even fail with “natural” transformations of images

[Azulay, Weiss preprint]



NN Failures

Objects can be created to trick neural networks!



 **TAHKION**
@takhion

i made a breakthrough. it turns out juggalo makeup defeats facial recognition successfully. if you want to avoid surveillance, become a juggalo i guess

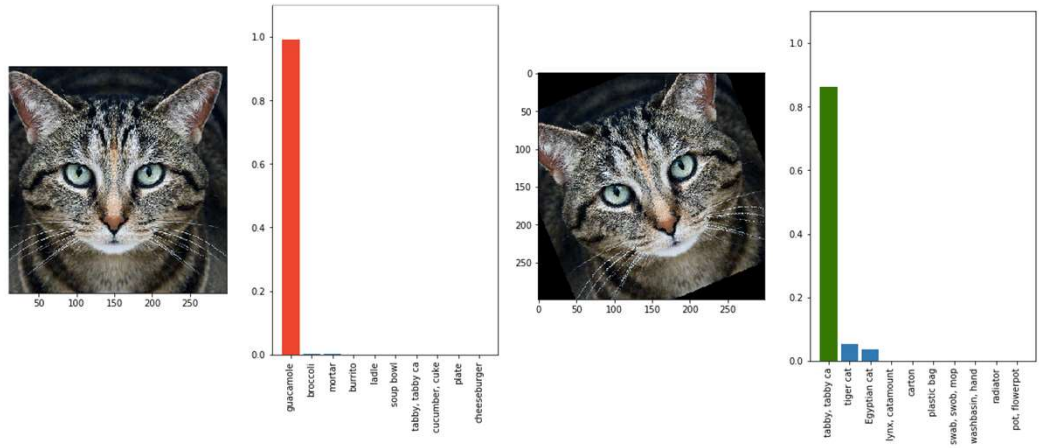
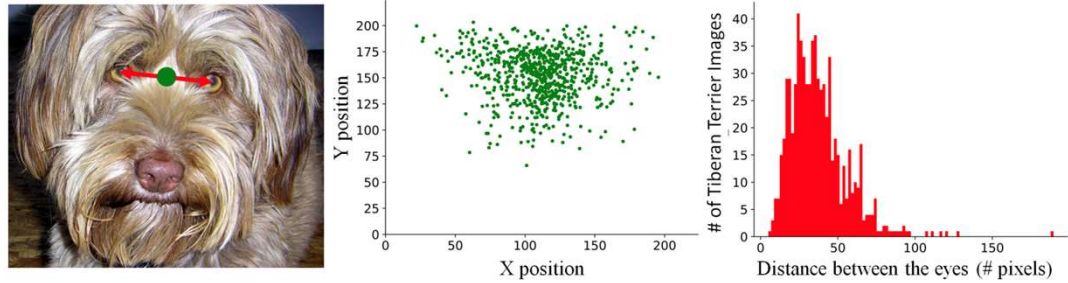


♥ 9,509 11:13 PM - Jun 30, 2018

🗨 3,618 people are talking about this

Dataset Bias

Datasets, like ImageNet, are generally biased



One approach is to augment your dataset to add random permutations of data to avoid bias.

Further Reading

Dealing with Variable Length Sequences (e.g. language)

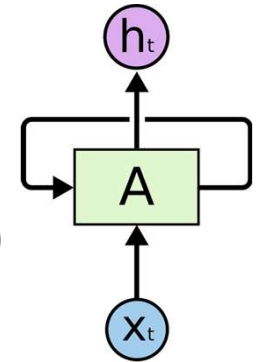
- Recurrent Neural Networks (RNNs)
- Long Short Term Memory Nets (LSTMs)
- <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Reinforcement Learning

- [Google DeepMind AlphaGo Zero](#)

Generative Adversarial Networks

- [How to learn synthetic data](#)



HW7

Your last assignment involves using a modern neural network library to make predictions using the CIFAR-10 dataset.

We recommend you use Google Colab for this assignment so that you can use their free GPU

Your first task is to read through the PyTorch tutorial to learn how to use their library

- Section tomorrow will introduce some stuff, but reading tutorial and documentation is critical

- Nobody

- Google Colab:

