

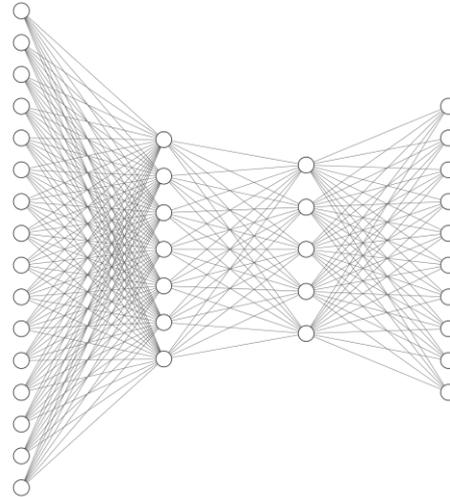
Video 1

*Recap Neural
Networks*

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.

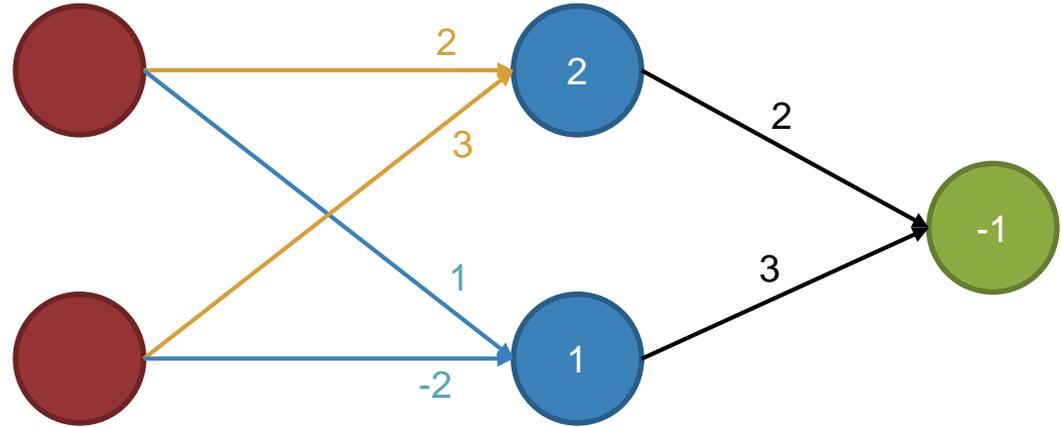


slido

Group 

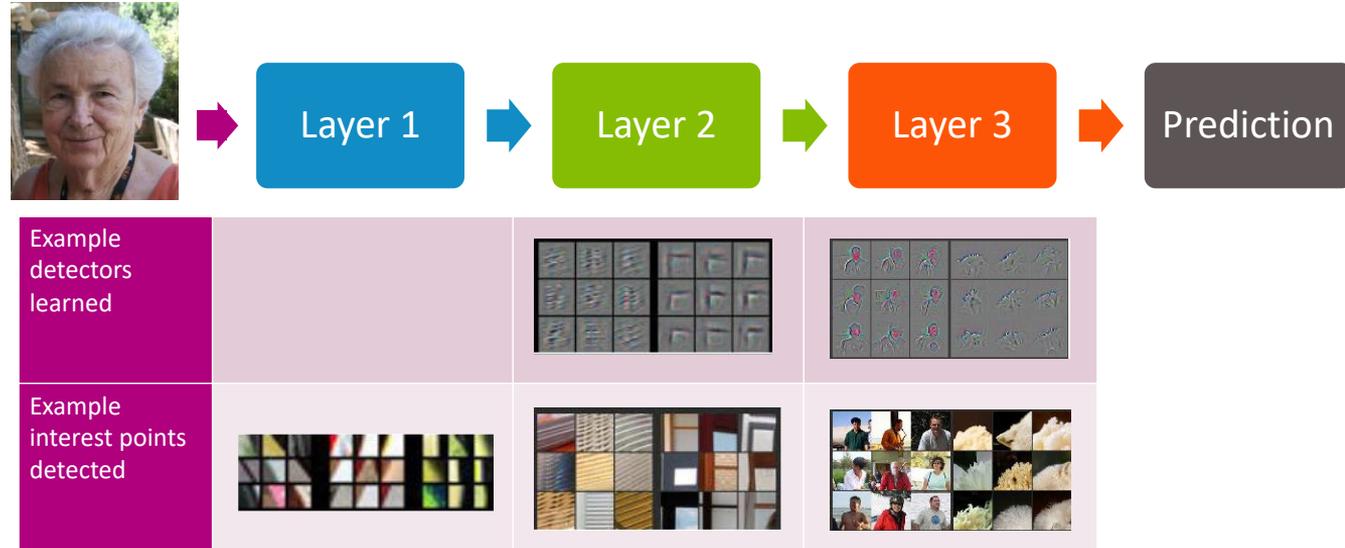
2 mins

- Compute the output for input (0, 1). There is a sign activation function on the hidden layers and output layer.



NNs to the Rescue

Neural Networks implicitly find these low level features for us!



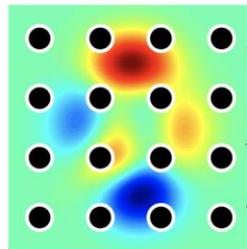
[Zeiler & Fergus '13]

Each layer learns more and more complex features

Hyperparameter Optimization

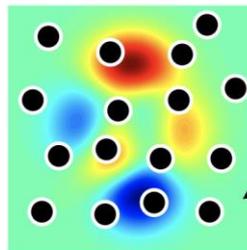
How do we choose hyperparameters to train and evaluate?

Grid search:



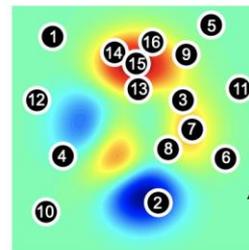
Hyperparameters
on 2d uniform grid

Random search:



Hyperparameters
randomly chosen

Bayesian Optimization:



Hyperparameters
adaptively chosen

Video 2

Convolutions

Image Challenges

Images are extremely high dimensional

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

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

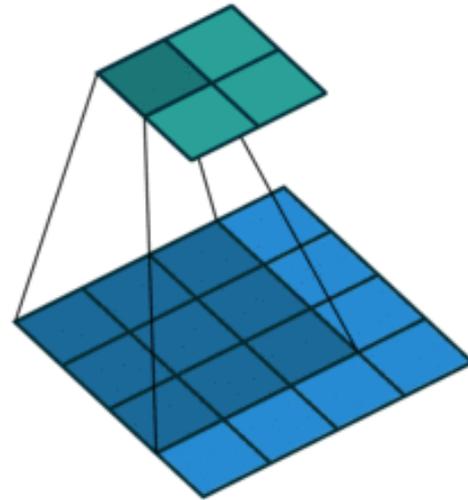
Images are structured, we should leverage this



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 combing 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



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

Convolution

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

3	3_0	2_1	1_2	0
0	0_2	1_2	3_0	1
3	1_0	2_1	2_2	3
2	0	0	2	2
2	0	0	0	1

12	12	17
10	17	19
9	6	14

Convolution

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

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

12	12	17
10	17	19
9	6	14

Convolution

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

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

12	12	17
10	17	19
9	6	14

Convolution

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

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

12	12	17
10	17	19
9	6	14

Convolution

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

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

12	12	17
10	17	19
9	6	14

Convolution

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

3	3	2	1	0
0	0	1	3	1
3_0	1_1	2_2	2	3
2_2	0_2	0_0	2	2
2_0	0_1	0_2	0	1

12	12	17
10	17	19
9	6	14



Convolution

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

3	3	2	1	0
0	0	1	3	1
3	1_0	2_1	2_2	3
2	0_2	0_2	2_0	2
2	0_0	0_1	0_2	1

12	12	17
10	17	19
9	6	14

Convolution

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

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

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

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



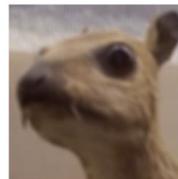
Sharpen

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



Box Blur

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



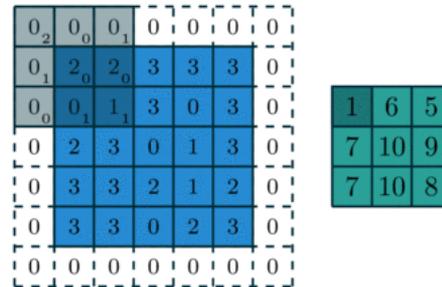
Convolutional Neural Networks (CNNs) learn the right weights for each kernel they use! Generally not interpretable!

Hyper-parameters of a Single Convolution

You can specify a few more things about a kernel

- Kernel dimensions
- 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



- Nobody
- Google Colab:



HW4
Walkthrough

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.

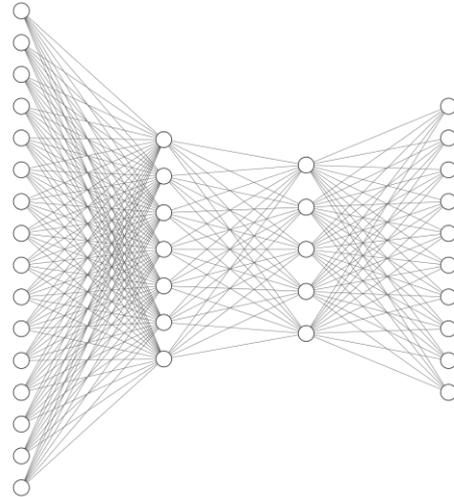
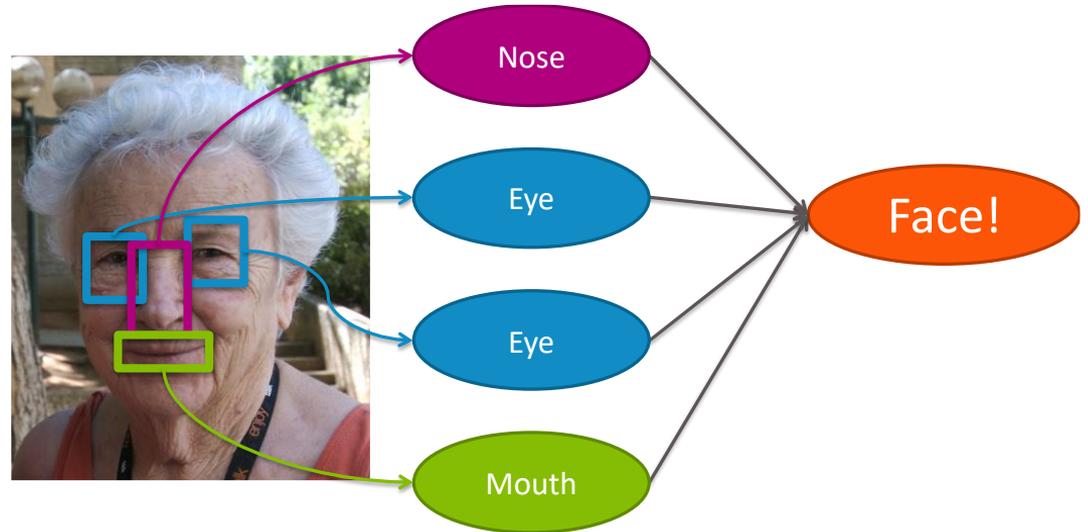


Image Features

Features in computer vision are local detectors

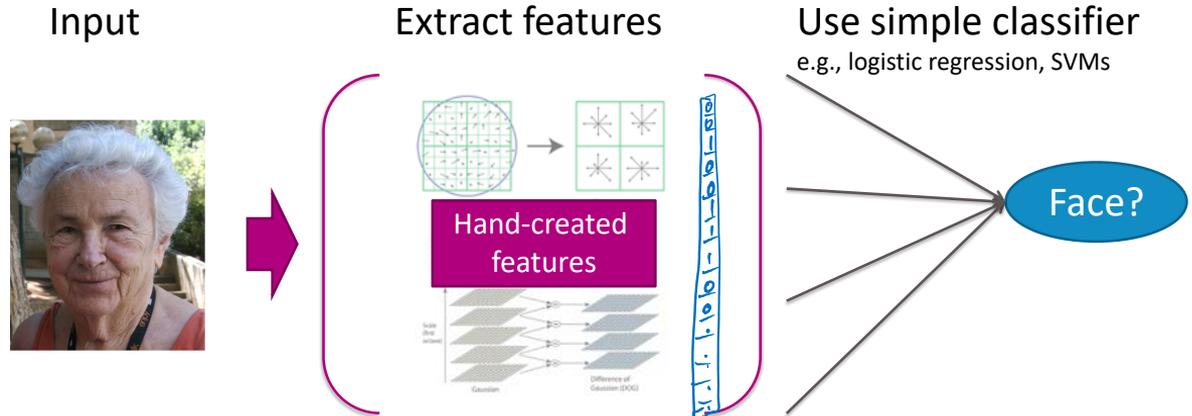
- Combine features to make prediction



In reality, these features are much more low level (e.g. Corner?)

The Past

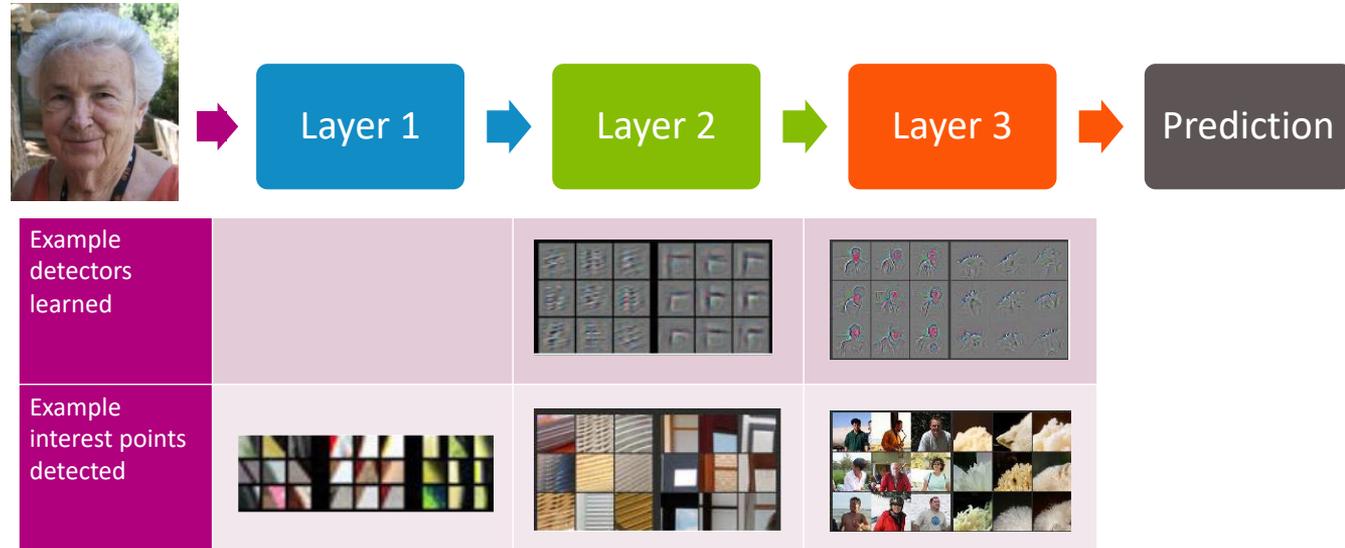
A popular approach to computer vision was to make hand-crafted features for object detection



Relies on coming up with these features by hand (yuck!)

NNs to the Rescue

Neural Networks implicitly find these low level features for us!



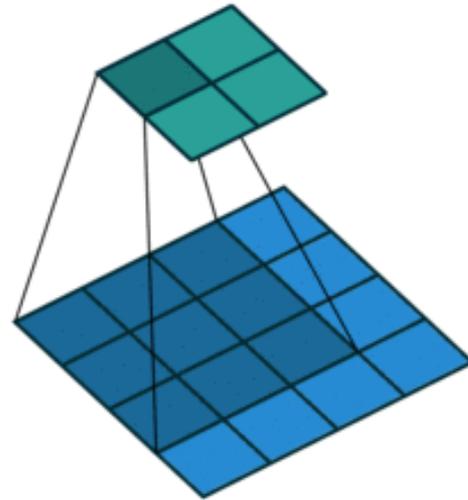
[Zeiler & Fergus '13]

Each layer learns more and more complex features

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 combing information about local pixels.

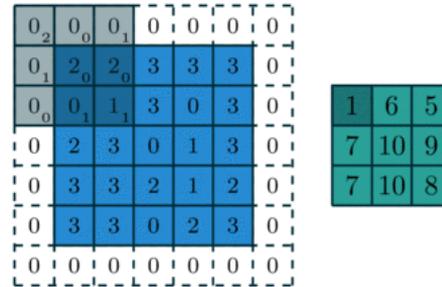


Hyper-parameters of a Single Convolution

You can specify a few more things about a kernel

- Kernel dimensions
- 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



slido

Group 

3 min

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

Convolutional Neural Networks

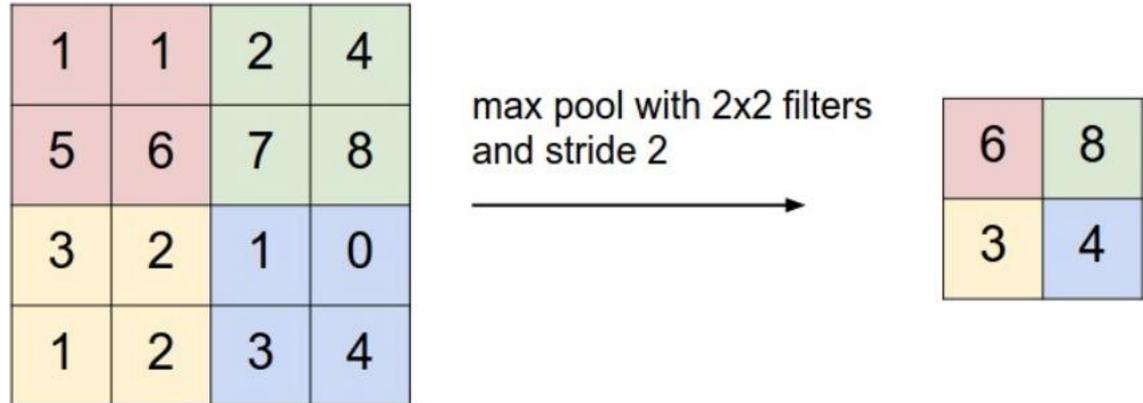
Pooling

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

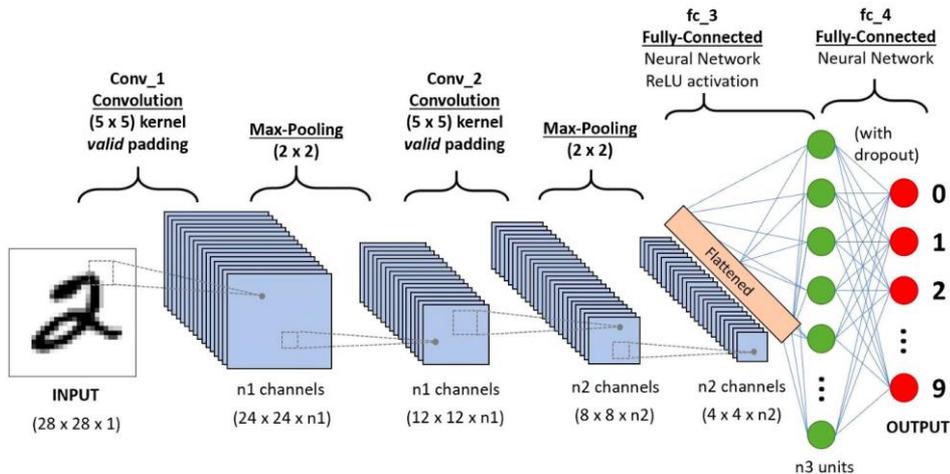


Convolutional Neural Network

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

Example architecture for hand-written digit recognition

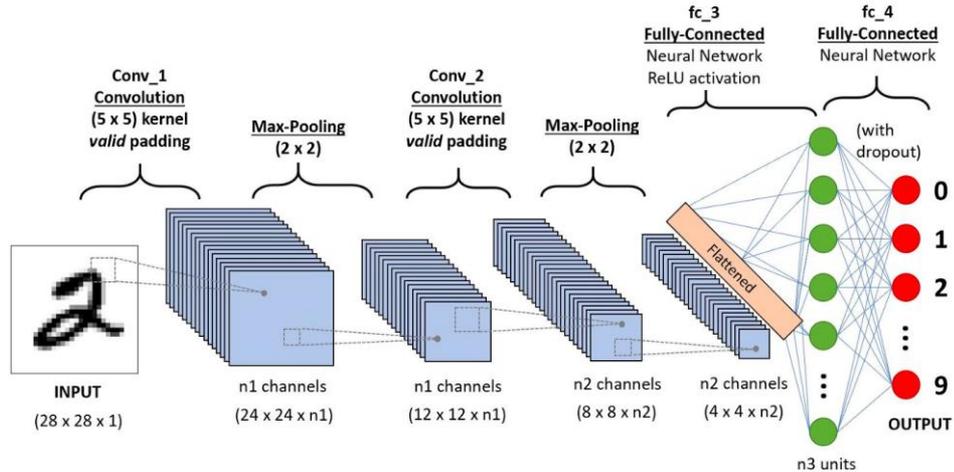
- Each convolution section uses many different kernels (increasing depth of channels)
- Pooling layers downsample each channel separately
- Usually ends with fully connected neural network



Convolutional Neural Network

Why does this help?

- Only need to learn a small number of values (kernel weights) that get applied to the entire image region by region
 - This is called weight-sharing
 - Gives efficiency + shift invariance
- Pooling lets us focus on features from larger and larger regions of the original image.

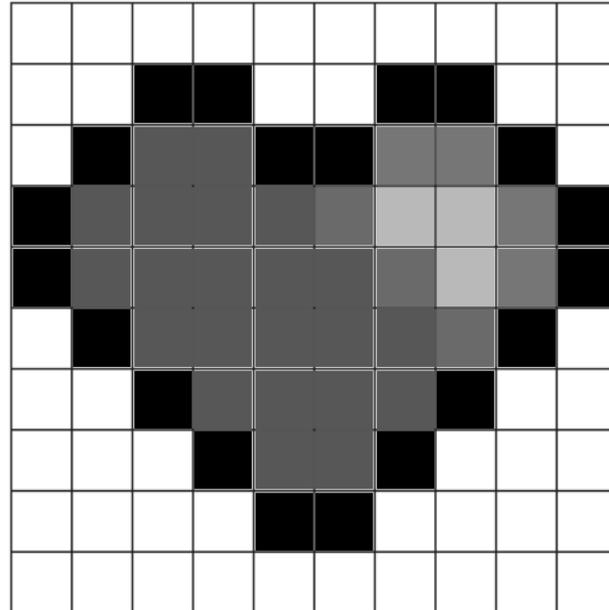


slido

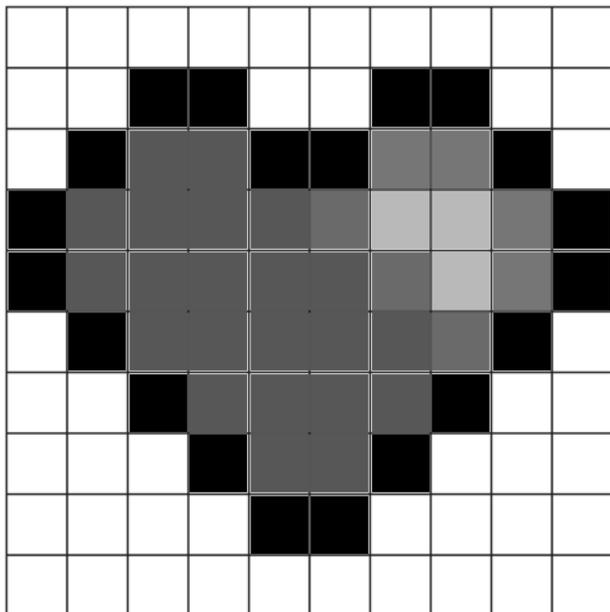
Group 

2 min

- **Input:** 10x10x1 image (grayscale image of 10x10 pixels)
- **Convolution:** 5x5 kernel, stride 1
- **MaxPool:** 2x2, stride 2
- What is the size of the resulting image?



- **Input:** 10x10x1 image (grayscale image of 10x10 pixels)
- **Convolution:** 5x5 kernel, stride 1
- **MaxPool:** 2x2, stride 2
- What is the size of the resulting image?





Brain Break

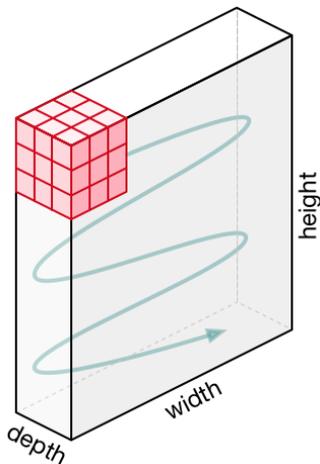


Number of
Weights /
Parameters

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



0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...

Input Channel #3 (Blue)

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

Kernel Channel #1

308

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

Kernel Channel #2

-498

0	1	1
0	1	0
1	-1	1

Kernel Channel #3

164

+

+

+ 1 = -25

Bias = 1

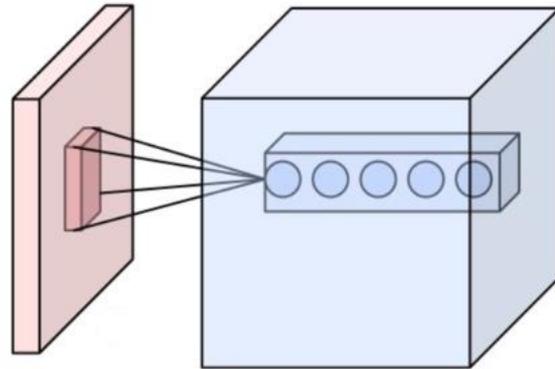
-25			...
			...
			...
			...
			...

CNN with Color Images

Another way of thinking about this process is each kernel is a (hidden-layer) 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.



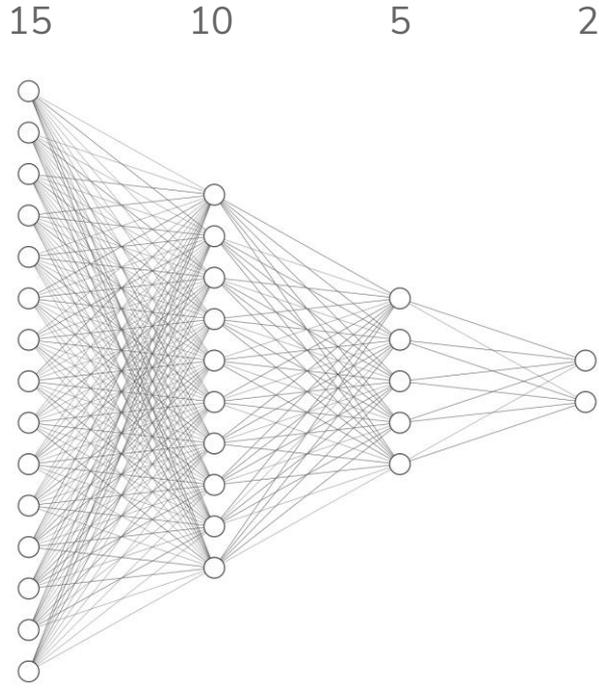
slido

Group 

2 min

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

Completely ignore intercept terms



Weight Sharing

Consider solving a digit recognition task on 28x28 grayscale images. Suppose I wanted to use a fully connected hidden layer with 84 neurons

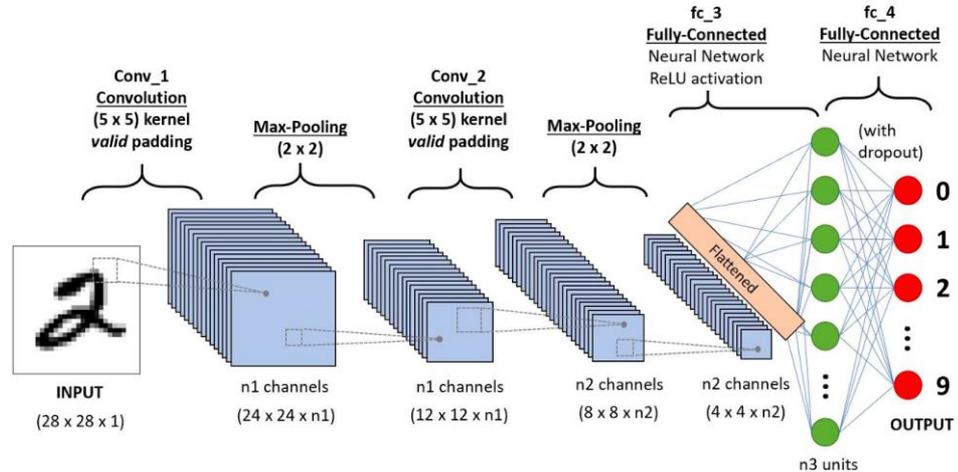
Without Convolutions:



Weight Sharing

Consider solving a digit recognition task on 28×28 images. Suppose I wanted to use a fully connected hidden layer with 84 neurons

With Convolutions (assume $n_1=10$, $n_2=20$) (not counting intercepts)

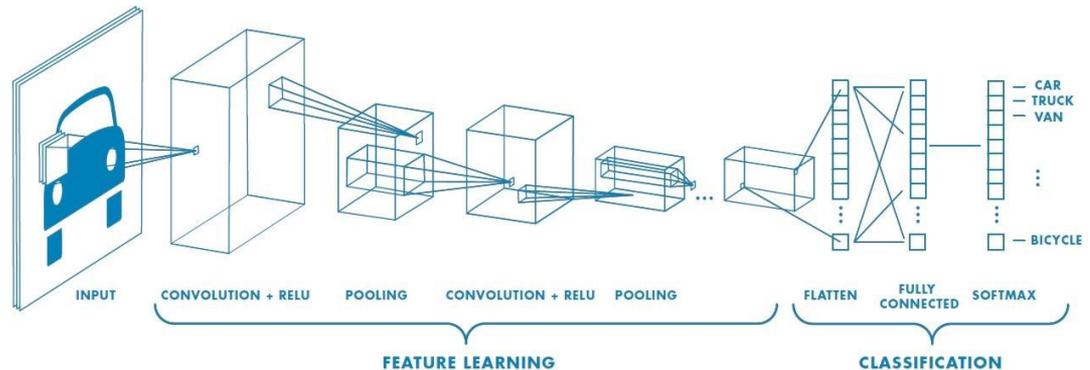


CNN Applications & Transfer Learning

General CNN Architecture

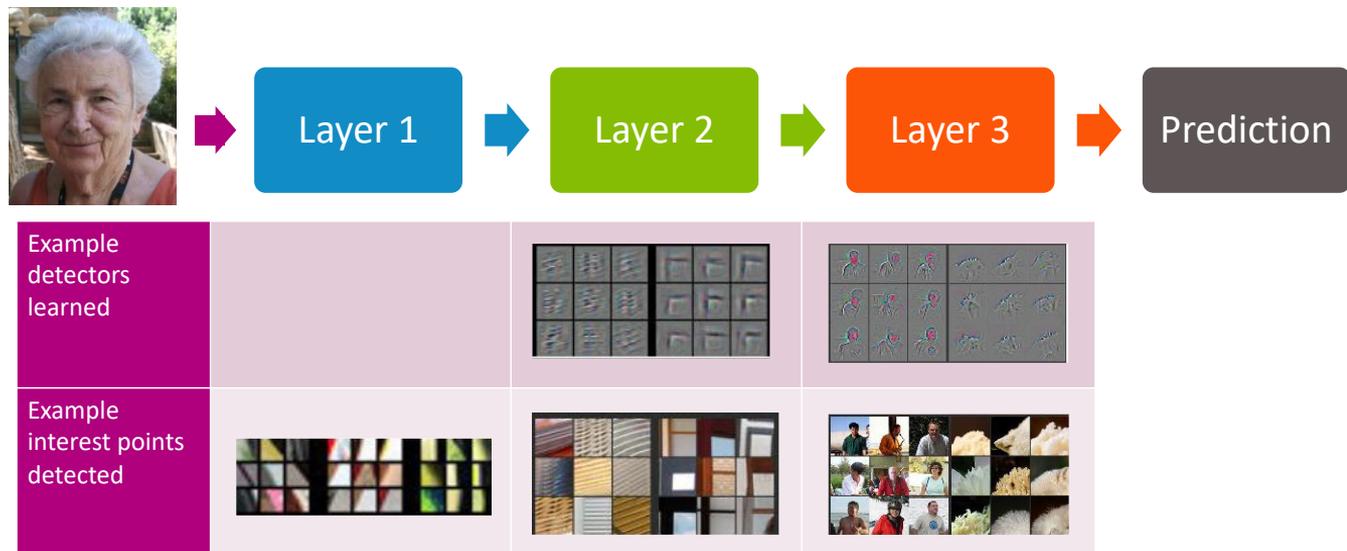
CNNs usually* have architectures that look like the following

- A series of Convolution + Activation Functions and Pooling layers. It's very common to do a pool after each convolution.
- Each set of operations lowers the size of the image but increases the number of features.
- 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]



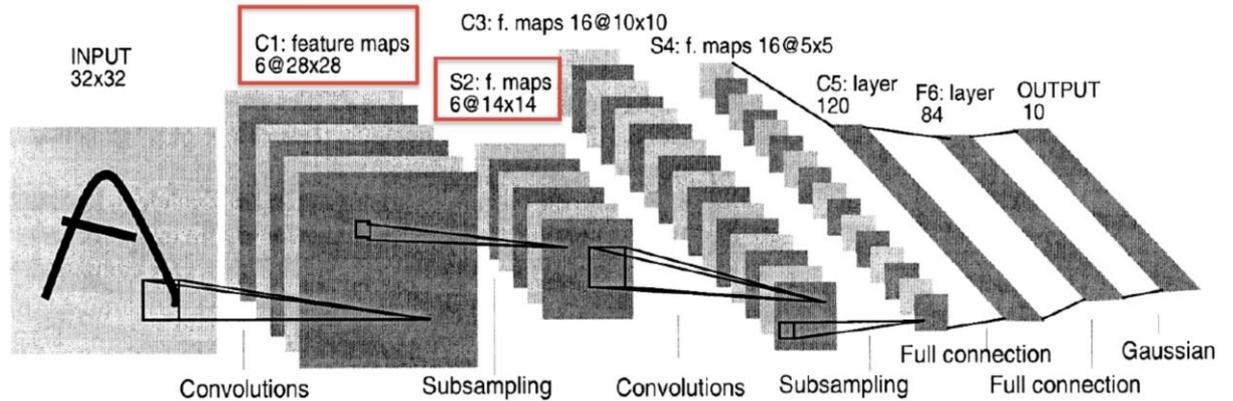
Brain Break



CNN Success

CNNs have had remarkable success in practice

LeNet, 1990s



CNN Success

LeNet made 82 errors on MNIST (popular hand-written digit dataset of size 60K). 99.86% accuracy



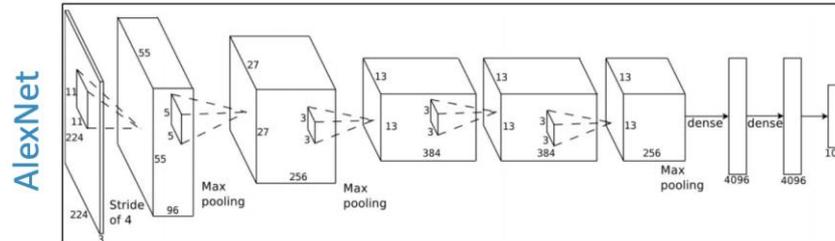
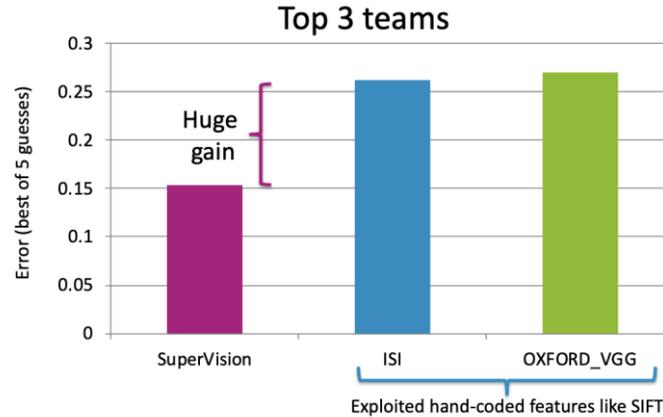
CNN Success

ImageNet 2012 competition:

- 1.2M training images
- 1000 categories

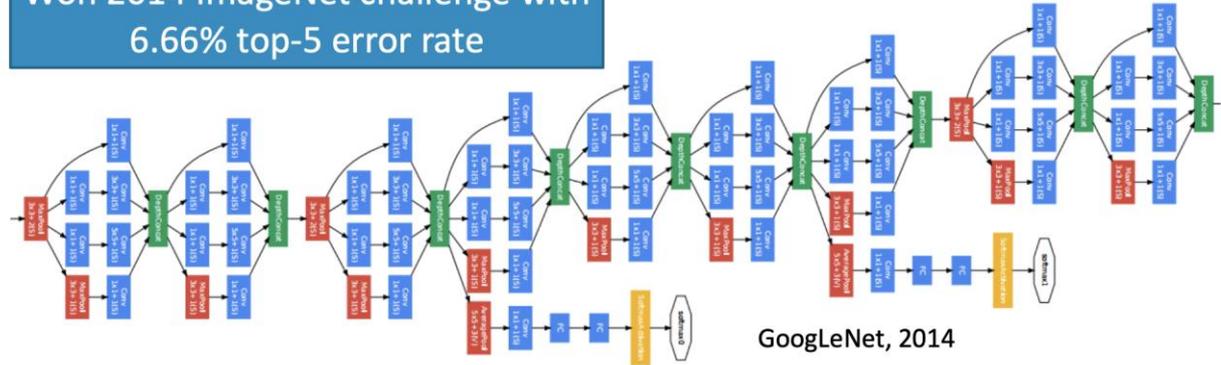
Winner: SuperVision

- 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



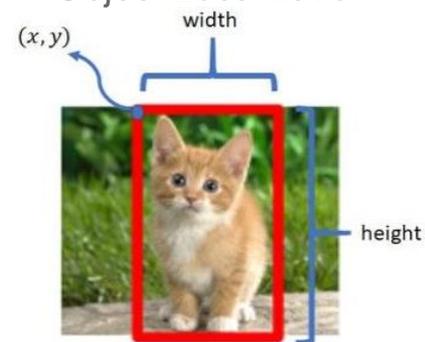
Input: x
Image pixels



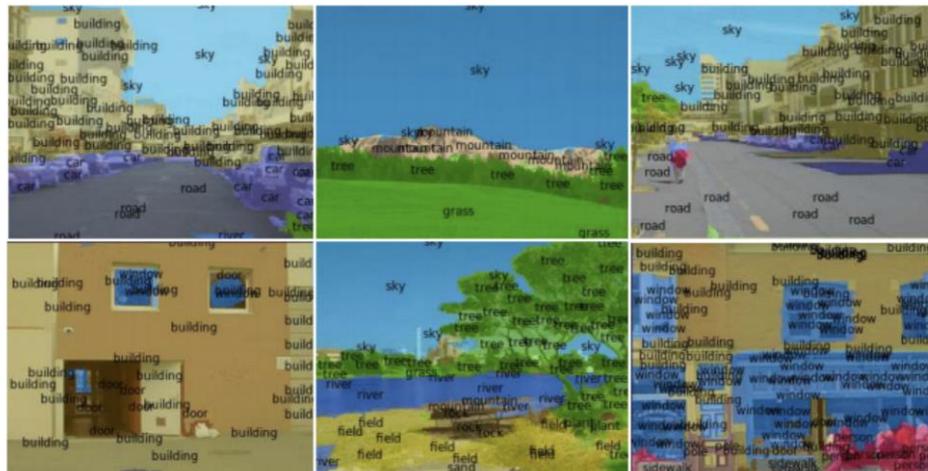
- Top Predictions
- Labrador retriever
 - golden retriever
 - redbone
 - bloodhound
 - Rhodesian ridgeback

Output: y
Predicted object

Object Localization



Scene Parsing [Farabet et al. '13]



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

Applications



Product Recommendation

Input Image



Nearest neighbors



slido

Group 

3 min

For each of the Computer Vision Tasks below, what do you think the output layer of the neural network would look like? What would each output neuron represent?

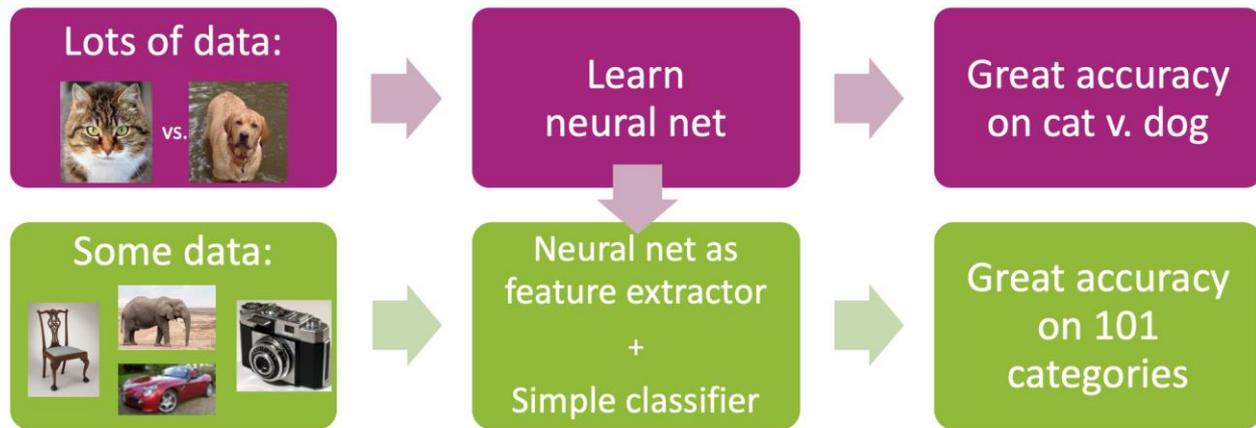
- **Image Classification:** Given an image with a single object, output the class of the object.
- **Object Localization:** Given an image with a single object, output the class **and** bounding box (x,y,w,h) of the object.
- **Object Detection:** Given an image with possibly multiple objects, output the bounding box **and** class for each object.

Image Classification: Given an image with a single object, output the class of the object.

Object Localization: Given an image with a single object, output the class **and** bounding box (x,y,w,h) of the object.

Object Detection: Given an image with possibly multiple objects, output the bounding box **and** class for each object.

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

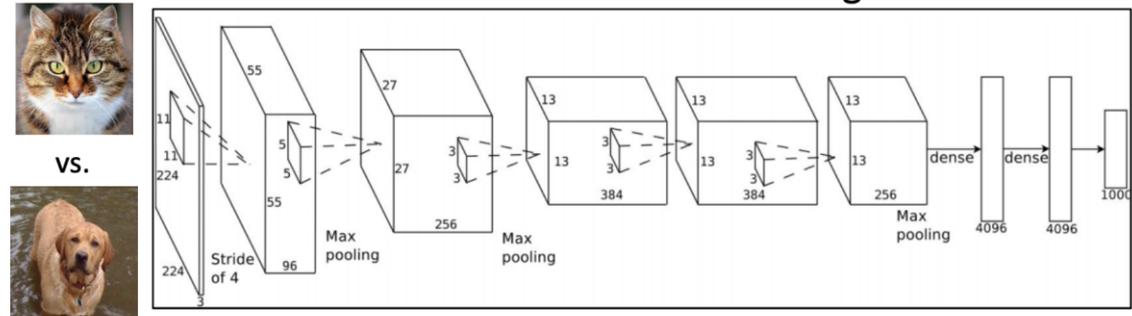
CNNs

What is learned in a neural network?

Initial layers are low-level and very general.

- Usually not sensitive/specific to the task at hand

Neural net trained for Task 1: cat vs. dog

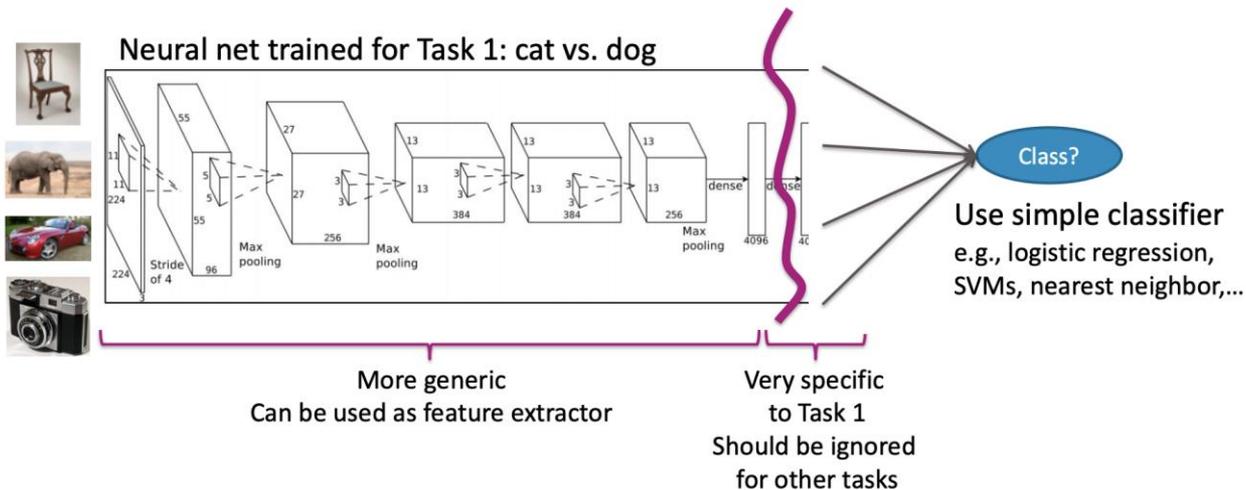


More generic
Can be used as feature extractor

Very specific
to Task 1
Should be ignored
for other tasks

Transfer Learning

Share the weights for the general part of the network



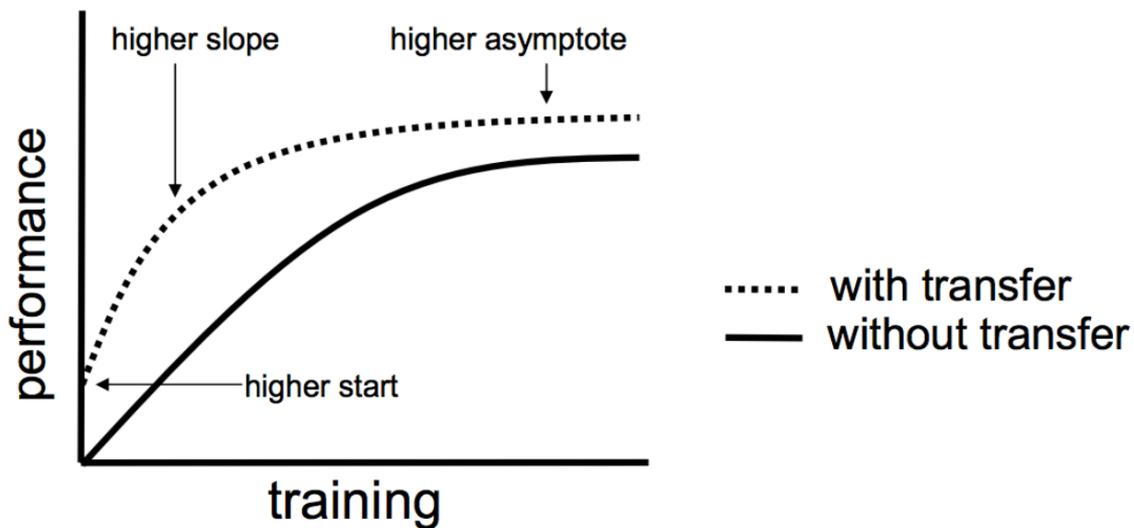
Keep weights fixed!

Re-train

Transfer Learning

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

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



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

- Environmentally, extremely expensive ([Green AI](#))

Hard to tune hyper-parameters

- Choice of architecture (we've added even more hyper-parameters)
 - Size of kernels, stride, 0 padding, number of conv layers, depth of outputs of conv layers,
- Learning algorithm

Still not very interpretable



NN Failures



"panda"
57.7% confidence

"No one adds noise to things in real applications"

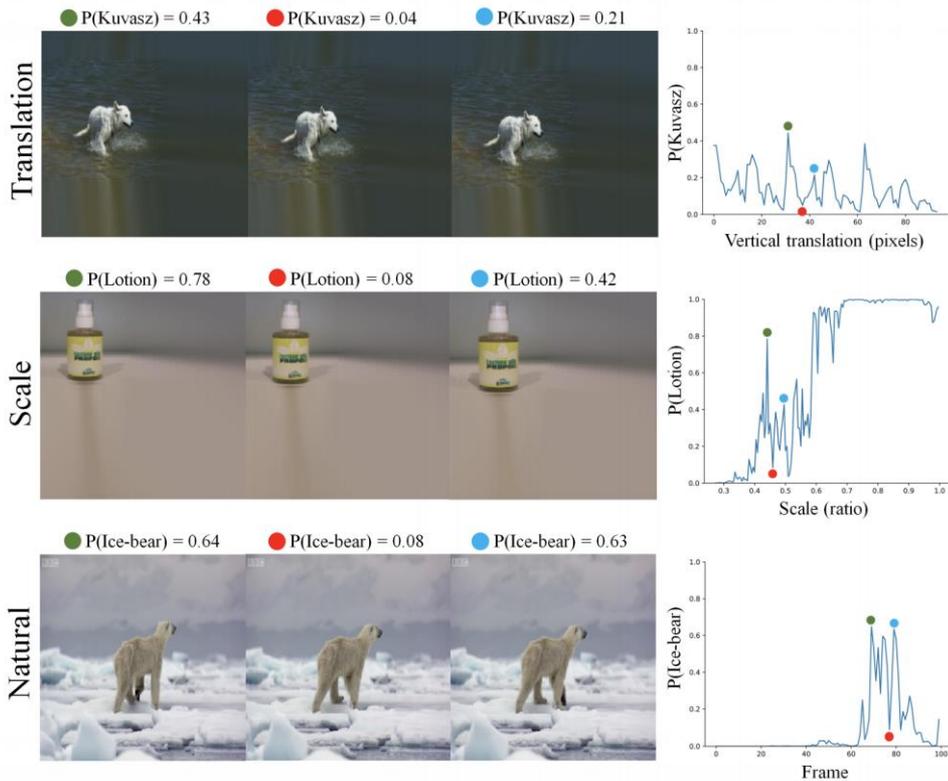
Not true!

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

NN Failures

They even fail with “natural” transformations of images

[Azulay, Weiss <https://arxiv.org/abs/1805.12177>]



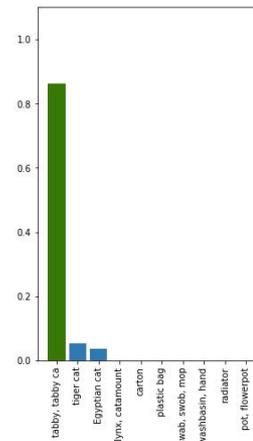
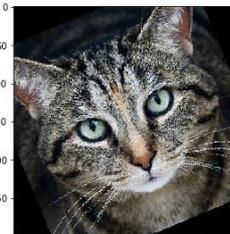
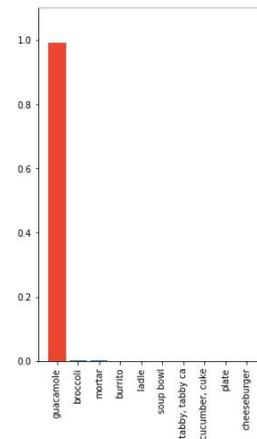
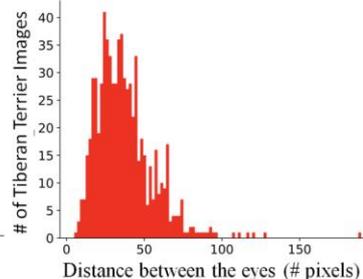
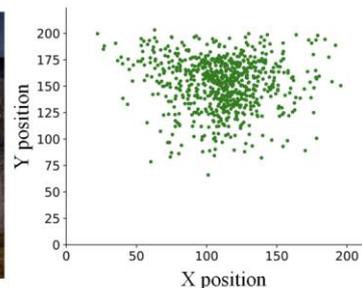
NN Failures

Objects can be created to trick neural networks!



Dataset Bias

Datasets, like ImageNet, are generally biased



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

Demo: Adversarial Neural Networks to Promote Fairness

<https://godatadriven.com/blog/towards-fairness-in-ml-with-adversarial-networks/>

Dataset: [Adult UCI](#)

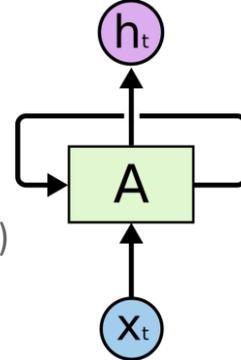
- Predict whether a person's income will be $> \$50K$ or $\leq \$50K$ based on factors like:
 - Age
 - Education level
 - Marital status
 - Served in Armed Services?
 - Hours per week worked
 - Occupation sector
 - Etc.



Further Readings on Deep Learning

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](#)

[Green AI](#)

