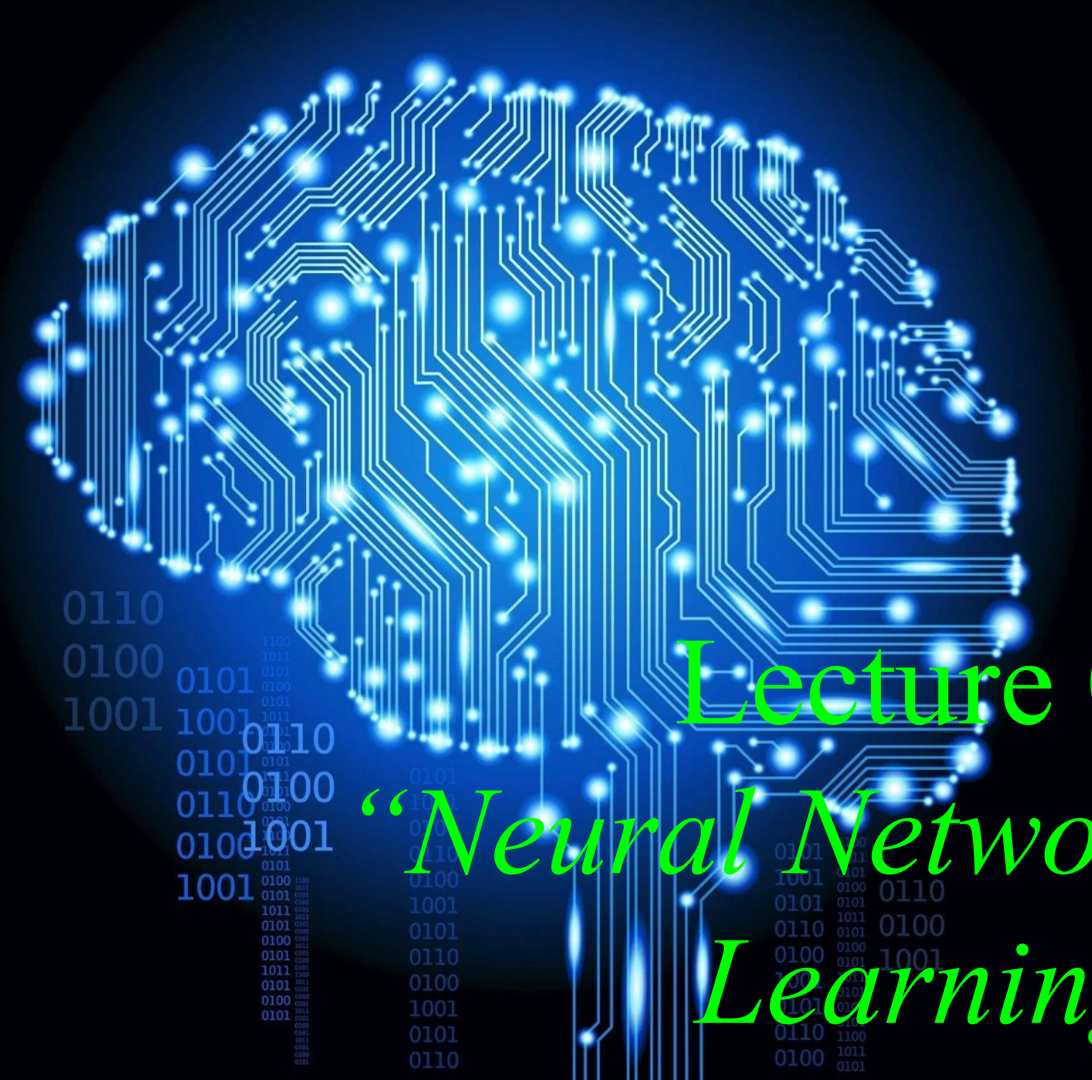
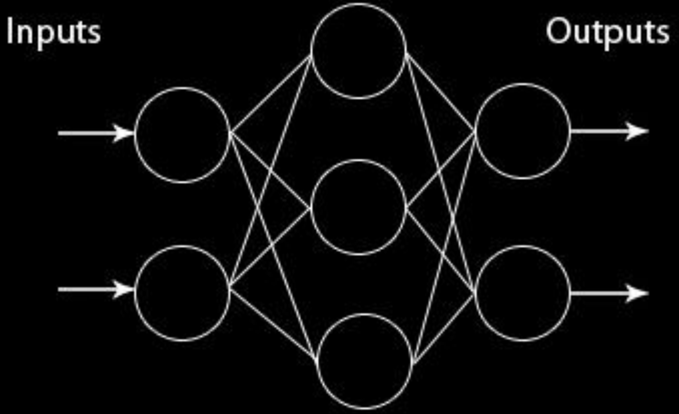


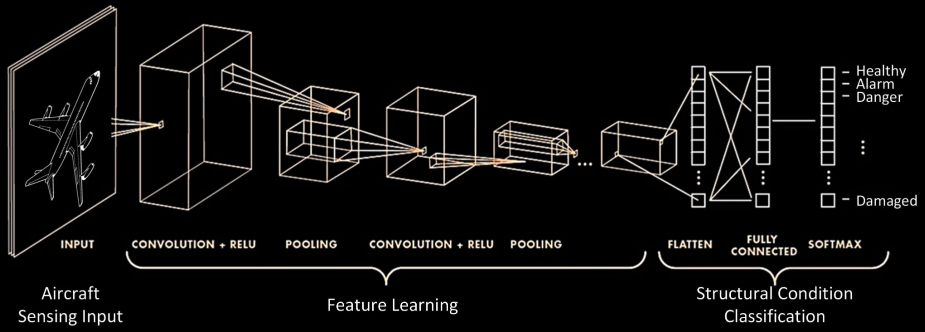
Lecture 09:

“Neural Networks, Deep Learning”

Two topics only for today...



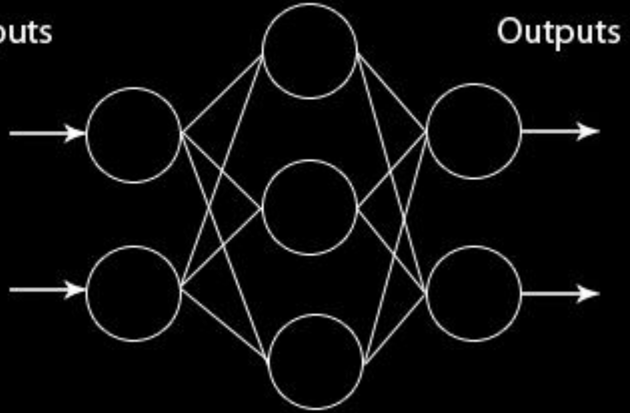
The Multilayer Perceptron (MLP)



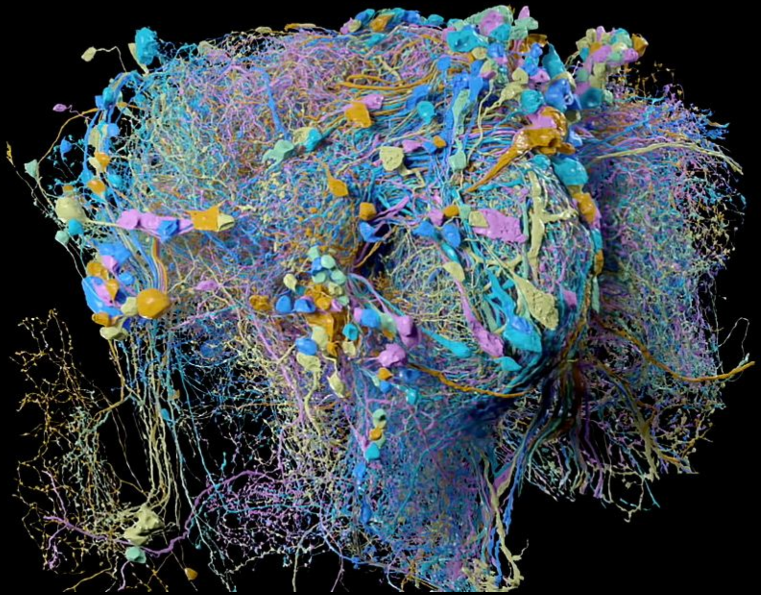
Deep Learning with CNNs

Part 1: The Multilayer Perceptron (MLP)

“Neural” Networks →Like the brain?



vs.



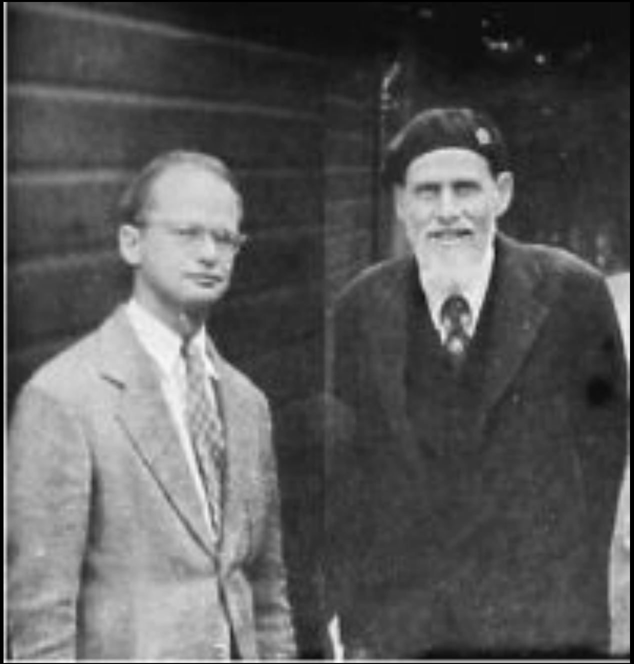
Advertising a Podcast

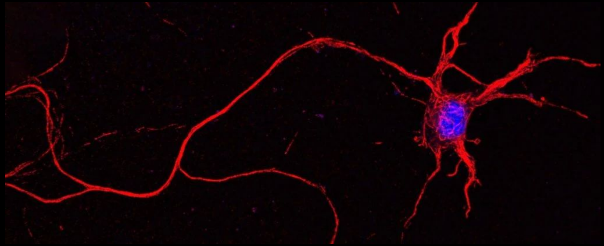
BRAIN INSPIRED

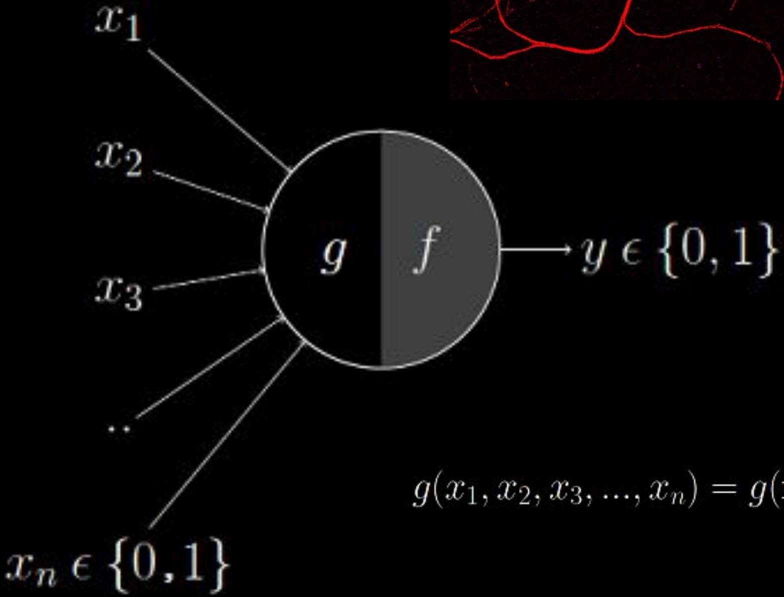
A podcast where neuroscience and AI converge.

https://braininspired.co/

Mccullough and Pitts (the 1943 MP-neuron)



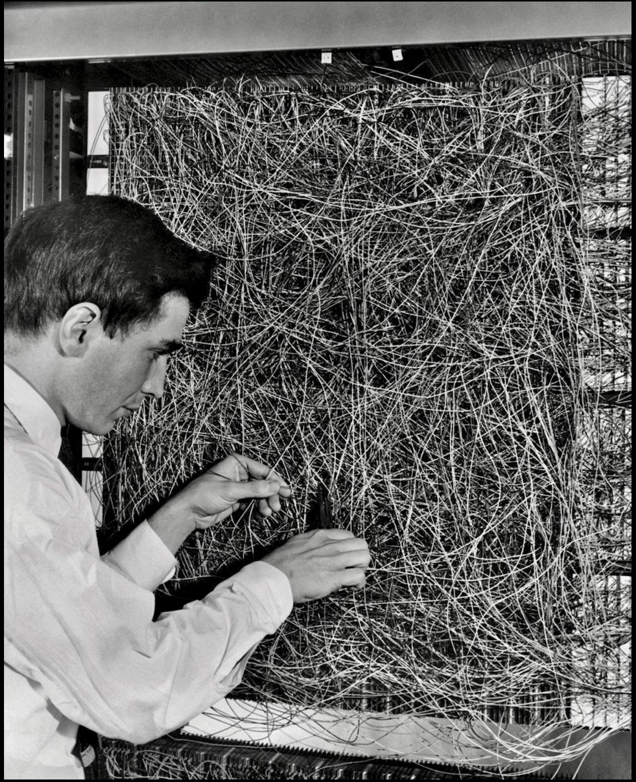


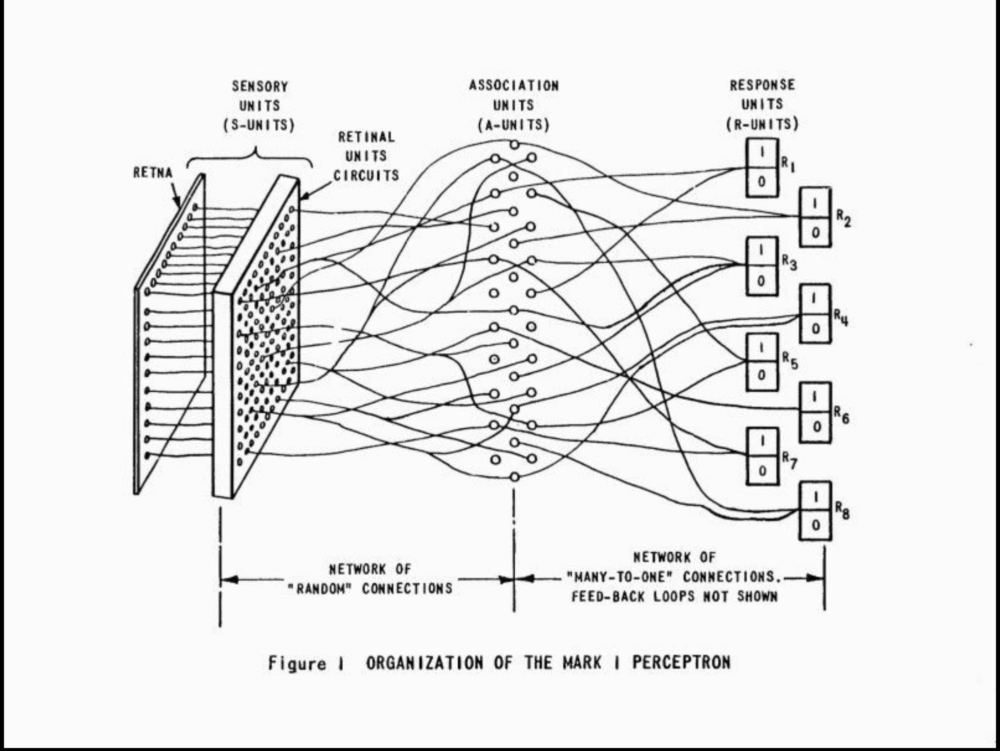


$$g(x_1, x_2, x_3, ..., x_n) = g(\mathbf{x}) = \sum_{i=1}^{n} x_i$$

$$\begin{aligned} y = f(g(\mathbf{x})) &= 1 \quad if \quad g(\mathbf{x}) \geq \theta \\ &= 0 \quad if \quad g(\mathbf{x}) < \theta \end{aligned}$$

Rosenblatt and the MLP (1960)



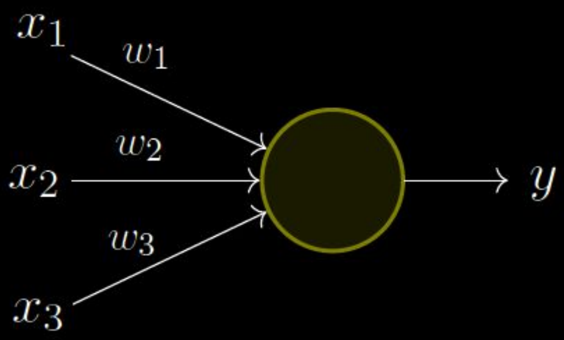


SENSORY UNITS (S-UNITS)
RETNA
RETINAL UNITS CIRCUITS
ASSOCIATION UNITS (A-UNITS)
RESPONSE UNITS (R-UNITS)
1
0
R_1
R_2
R_3
R_4
R_5
R_6
R_7
R_8
NETWORK OF "RANDOM" CONNECTIONS
NETWORK OF "MANY-TO-ONE" CONNECTIONS. FEED-BACK LOOPS NOT SHOWN

Figure 1 ORGANIZATION OF THE MARK I PERCEPTRON

Minsky-Papert (MP-neuron → the 1969 Perceptron)

- Added weights, removed binary requirement of input values



Perceptron Model (Minsky-Papert in 1969)

The Multilayer Perceptron (MLP)

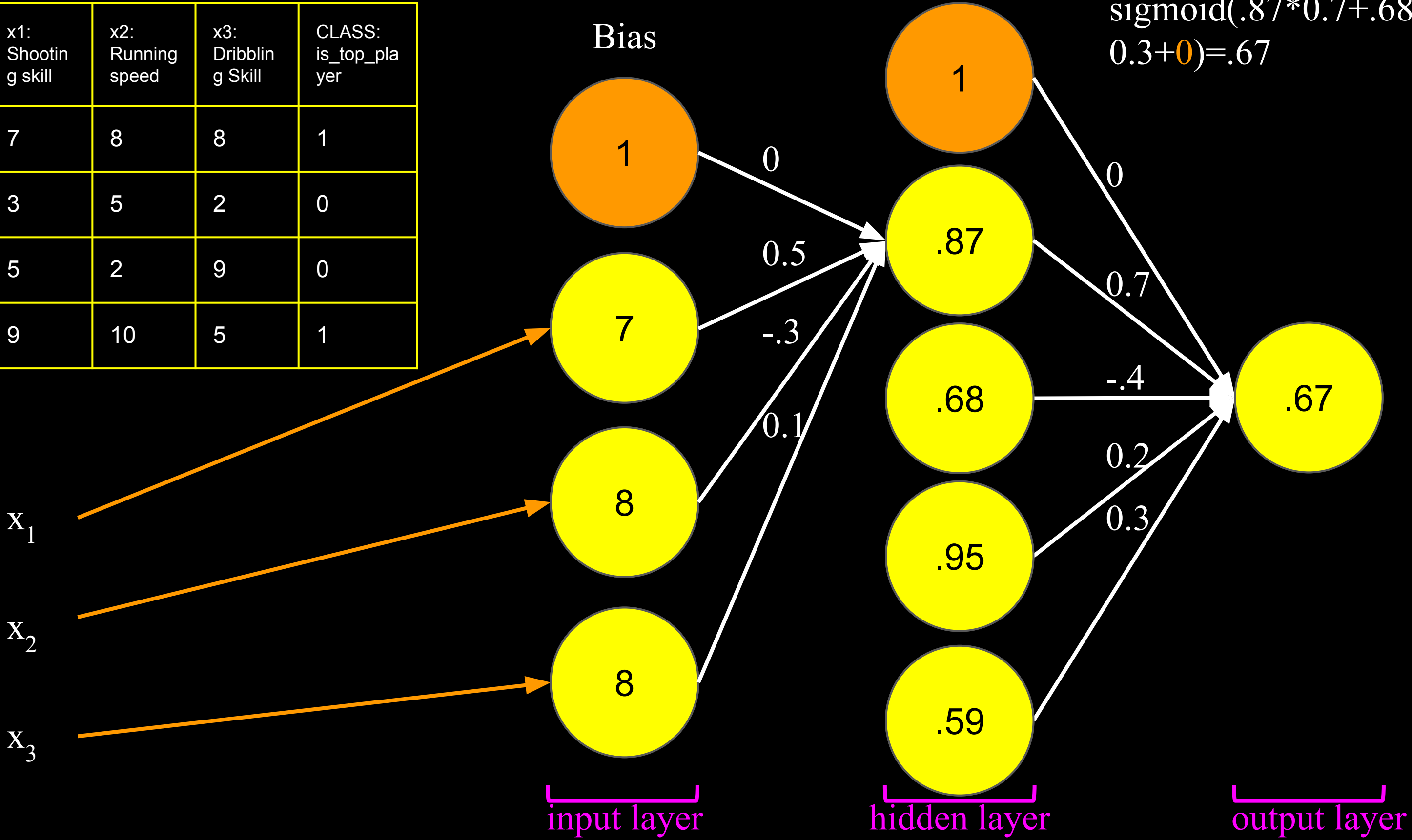
- Say that we have a dataset with 3 features and 1 class output

x1: Shooting skill	x2: Running speed	x3: Dribbling Skill	CLASS: is_top_player
7	8	8	1
3	5	2	0
5	2	9	0
9	10	5	1

- How does an MLP handle the inputs/outputs?

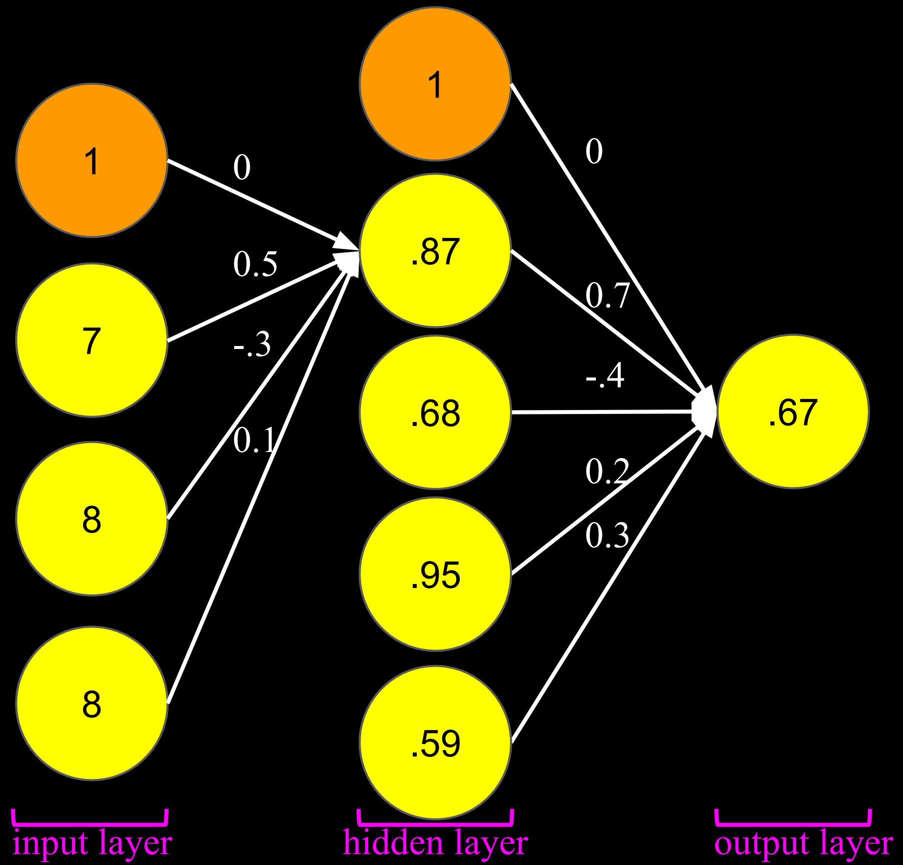
The Multilayer Perceptron (MLP)

sigmoid(7*0.5+8*-0.3+8*0.1+0)=.87
sigmoid(.87*0.7+.68*-.4+.95*0.2+.59*
0.3+0)=.67



x1: Shooting skill	x2: Running speed	x3: Dribbling Skill	CLASS: is_top_player
7	8	8	1
3	5	2	0
5	2	9	0
9	10	5	1

MLP - Backprop Intuition



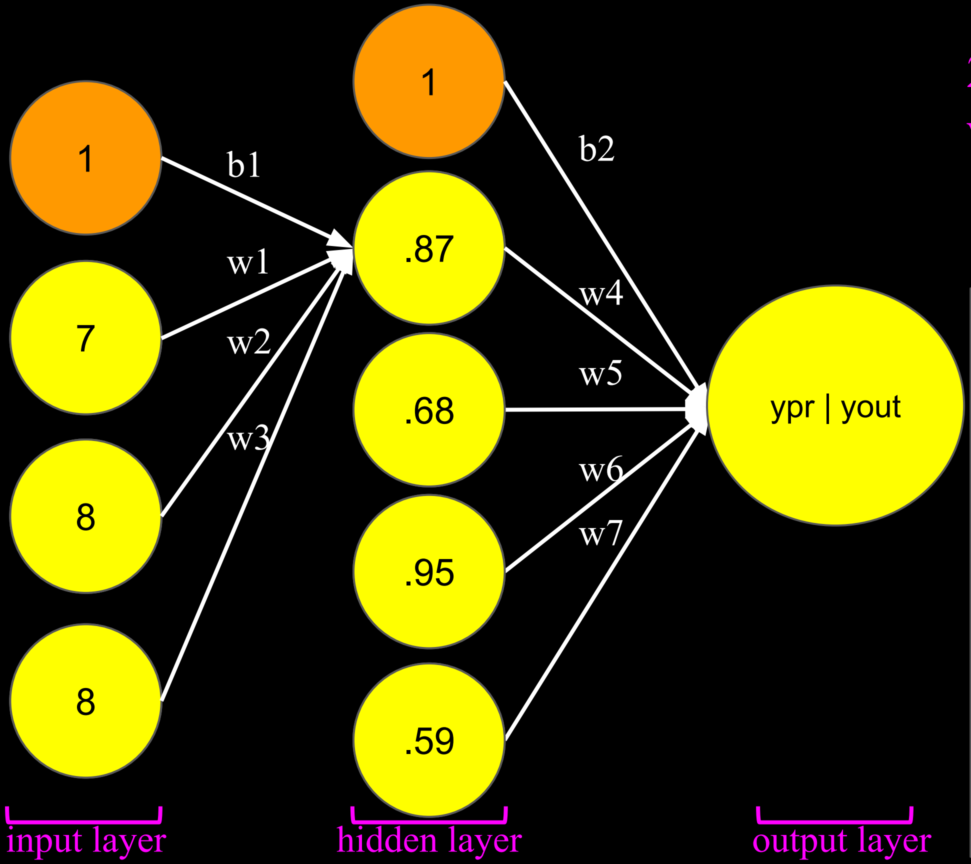
1) Calculate the loss: use Cross-Entropy
CE loss = -log(predicted probability)

For class 0 predictions, we would first need to correct the predicted probability (1 - predicted probability)

Loss = -log(0.67) = 0.44

As usual, we really mean "ln" here

MLP - Backprop Intuition



2) Determine the amount that the error would change if we changed each weight

For example, let's update w4:

$$\frac{\partial Loss}{\partial w4} = \boxed{\frac{\partial Loss}{\partial yout}} \cdot \boxed{\frac{\partial yout}{\partial ypr}} \cdot \boxed{\frac{\partial ypr}{\partial w4}}$$

$$\boxed{\frac{\partial Loss}{\partial yout}} = -\frac{1}{yout}$$

$$\boxed{\frac{\partial ypr}{\partial w4}} = 0.87$$

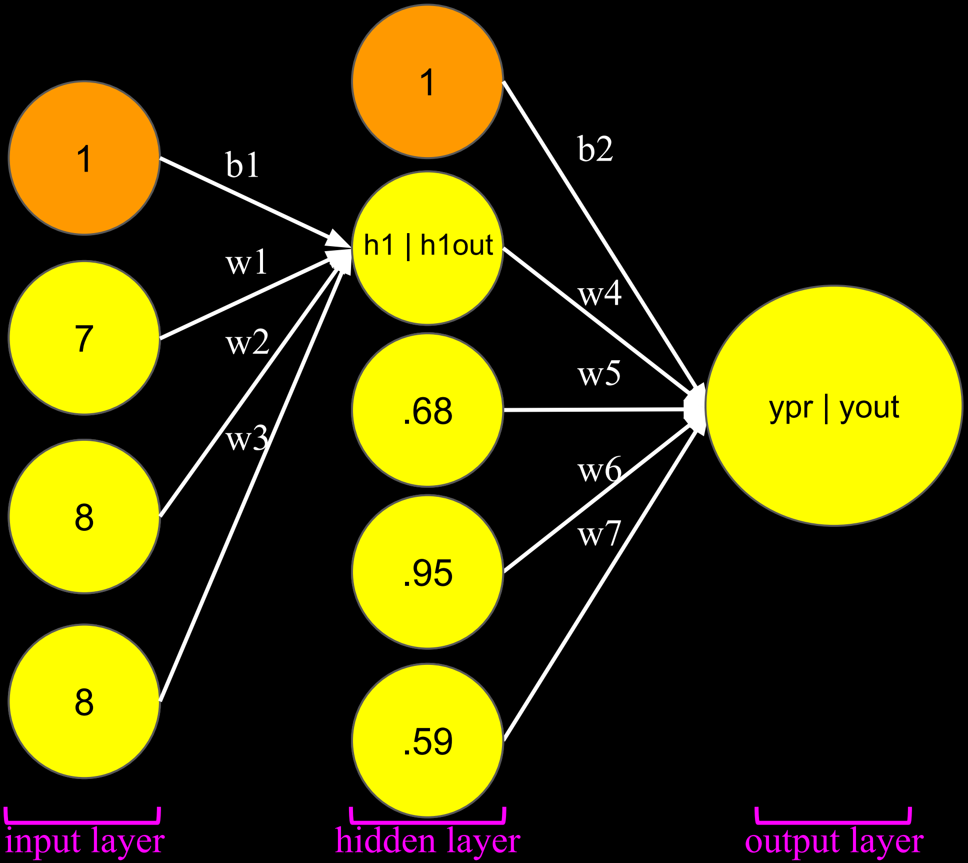
$$\boxed{\frac{\partial yout}{\partial ypr}} = \frac{e^{-ypr}}{\left(e^{-ypr}+1\right)^2}$$

$$\therefore \frac{\partial Loss}{\partial w4} = \left(-\frac{1}{.44}\right) \cdot \left(\frac{e^{-.67}}{\left(1+e^{-.67}\right)^2}\right) \cdot (0.87) = -.44$$

What about the hidden layer weights?

MLP - Backprop Intuition

For example, let's update w1:



$$\frac{\partial Loss}{\partial w1} = \boxed{\frac{\partial Loss}{\partial h1out}} \cdot \boxed{\frac{\partial h1out}{\partial h1}} \cdot \frac{\partial h1}{\partial w1}$$

$$\boxed{\frac{\partial Loss}{\partial h1out}} = \boxed{\frac{\partial Loss}{\partial ypr}} \cdot \frac{\partial ypr}{\partial h1out}$$

$$\boxed{\frac{\partial Loss}{\partial ypr}} = \frac{\partial Loss}{\partial yout} \cdot \frac{\partial yout}{\partial ypr}$$

$$\boxed{\frac{\partial h1out}{\partial h1}} = \frac{e^{-h1}}{\left(e^{-h1}+1\right)^2} \qquad \frac{\partial h1}{\partial w1} = 7$$

$$\frac{\partial Loss}{\partial w1} = \left(-\frac{1}{.44}\right)(.223917)(w4)\left(\frac{e^{-h1}}{\left(e^{-h1}+1\right)^2}\right)(7)$$

$$\cong -.282$$

Some finer points here...

1) Unspoken Guidelines
2) Activation Functions
3) Dropout regularization
4) Optimizers (Adam)

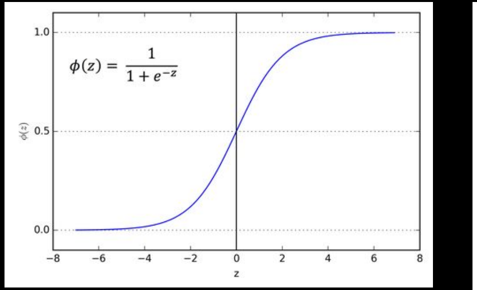
1) Unspoken guidelines:

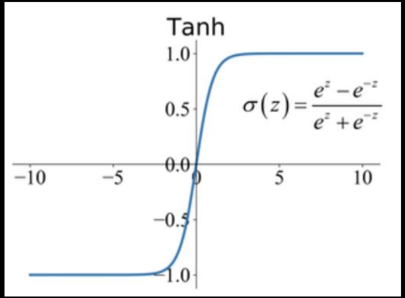
- For linearly separable features, no hidden layers required
- For most problems, 1 hidden layer is all you need

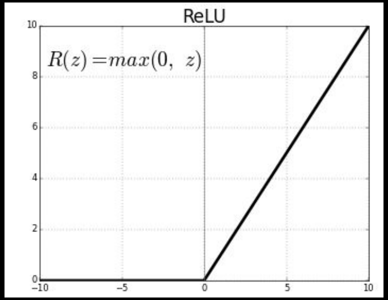
Some finer points here...

2) Activation functions (a few here)

- These **add nonlinearity** to your neural network → with these, we can fit more than a line





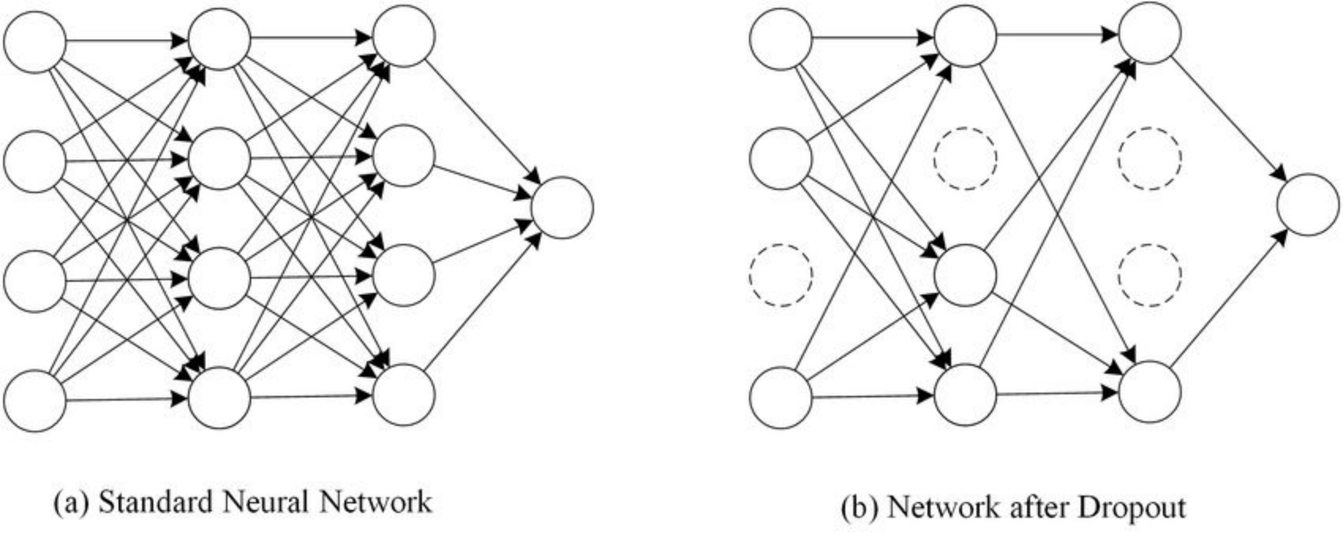


- Both sigmoid and tanh activation functions squeeze the outputs → leads to vanishing gradients

- Relu is a fix to this vanishing gradient issue.

Some finer points here...

3) Dropout



(a) Standard Neural Network

(b) Network after Dropout

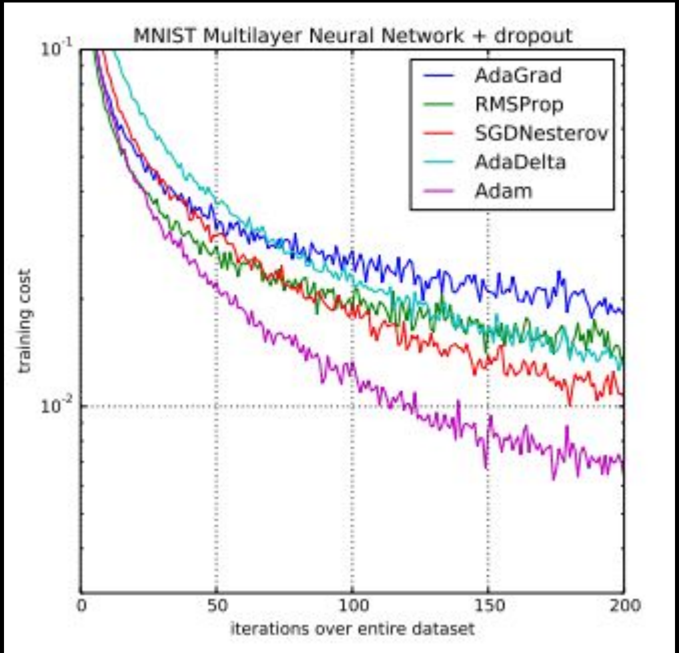
Extra notes (post-class):

- Dropout can be applied to the input layer or to any hidden layers (implemented per layer)
- It can, for example, be re-applied for each training sample in the sample set

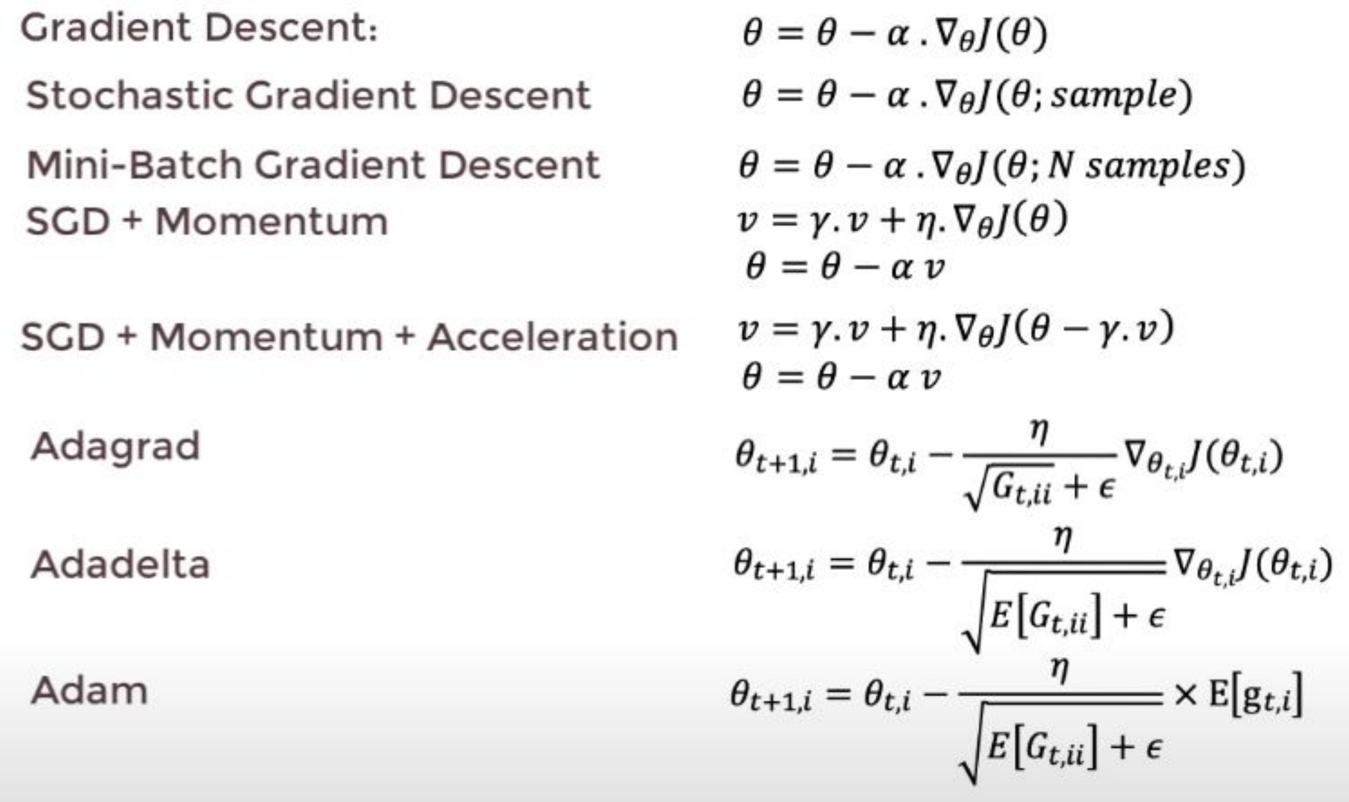
Some finer points here...

4) Optimizers

- Adam is a great optimizer, and we’ll use it by default for today
- Adam can take different-sized steps for different parameters; it also has momentum
- But, try to be skeptical whenever something is said to be “the best”. Often it’s task-dependent
- Here’s a good article on this topic: https://ruder.io/optimizing-gradient-descent/



Some Optimizer Equations (Post-class, **enrichment material)



Gradient Descent:	$\theta = \theta - \alpha \,.\, \nabla_{\theta} J(\theta)$
Stochastic Gradient Descent	$\theta = \theta - \alpha \,.\, \nabla_{\theta} J(\theta; sample)$
Mini-Batch Gradient Descent	$\theta = \theta - \alpha \,.\, \nabla_{\theta} J(\theta; N\ samples)$
SGD + Momentum	$v = \gamma . v + \eta . \nabla_{\theta} J(\theta)$ $\theta = \theta - \alpha\, v$
SGD + Momentum + Acceleration	$v = \gamma . v + \eta . \nabla_{\theta} J(\theta - \gamma . v)$ $\theta = \theta - \alpha\, v$
Adagrad	$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \nabla_{\theta_{t,i}} J(\theta_{t,i})$
Adadelta	$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{E\left[G_{t,ii}\right] + \epsilon}} \nabla_{\theta_{t,i}} J(\theta_{t,i})$
Adam	$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{E\left[G_{t,ii}\right] + \epsilon}} \times \mathrm{E}\left[g_{t,i}\right]$

What can a MLP do?

MLP-Mixer: An all-MLP Architecture for Vision

Ilya Tolstikhin*, Neil Houlsby*, Alexander Kolesnikov*, Lucas Beyer*,
Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner,
Daniel Keysers, Jakob Uszkoreit, Mario Lucic, Alexey Dosovitskiy
*equal contribution
Google Research, Brain Team
{tolstikhin, neilhoulsby, akolesnikov, lbeyer,
xzhai, unterthiner, jessicayung†, andstein,
keysers, usz, lucic, adosovitskiy}@google.com
†work done during Google AI Residency

Abstract

Convolutional Neural Networks (CNNs) are the go-to model for computer vision. Recently, attention-based networks, such as the Vision Transformer, have also become popular. In this paper we show that while convolutions and attention are both sufficient for good performance, neither of them are necessary. We present *MLP-Mixer*, an architecture based exclusively on multi-layer perceptrons (MLPs). MLP-Mixer contains two types of layers: one with MLPs applied independently to image patches (i.e. "mixing" the per-location features), and one with MLPs applied across patches (i.e. "mixing" spatial information). When trained on large datasets, or with modern regularization schemes, MLP-Mixer attains competitive scores on image classification benchmarks, with pre-training and inference cost comparable to state-of-the-art models. We hope that these results spark further research beyond the realms of well established CNNs and Transformers.[1]

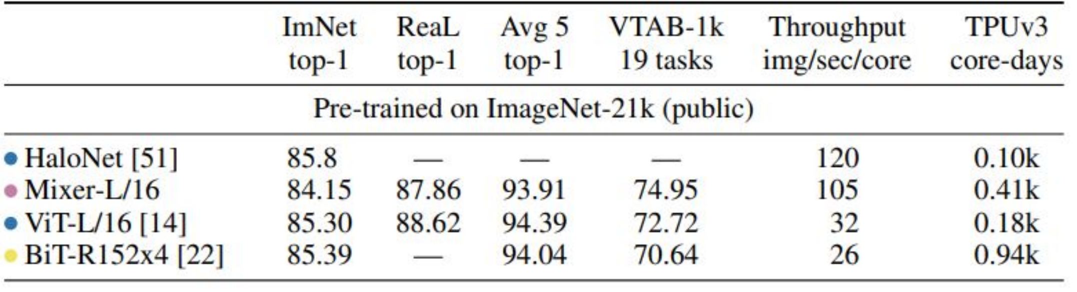
1 Introduction

As the history of computer vision demonstrates, the availability of larger datasets coupled with increased computational capacity often leads to a paradigm shift. While Convolutional Neural Networks (CNNs) have been the de-facto standard for computer vision, recently Vision Transformers [14] (ViT), an alternative based on self-attention layers, attained state-of-the-art performance. ViT continues the long-lasting trend of removing hand-crafted visual features and inductive biases from models and relies further on learning from raw data.

We propose the *MLP-Mixer* architecture (or "Mixer" for short), a competitive but conceptually and technically simple alternative, that does not use convolutions or self-attention. Instead, Mixer's architecture is based entirely on multi-layer perceptrons (MLPs) that are repeatedly applied across either spatial locations or feature channels. Mixer relies only on basic matrix multiplication routines, changes to data layout (reshapes and transpositions), and scalar nonlinearities.

Figure 1 depicts the macro-structure of Mixer. It accepts a sequence of linearly projected image patches (also referred to as *tokens*) shaped as a "patches × channels" table as an input, and maintains this dimensionality. Mixer makes use of two types of MLP layers: *channel-mixing MLPs* and *token-mixing MLPs*. The channel-mixing MLPs allow communication between different channels;

[1] MLP-Mixer code will be available at `https://github.com/google-research/vision_transformer`



	ImNet top-1	ReaL top-1	Avg 5 top-1	VTAB-1k 19 tasks	Throughput img/sec/core	TPUv3 core-days
	Pre-trained on ImageNet-21k (public)					
● HaloNet [51]	85.8	—	—	—	120	0.10k
● Mixer-L/16	84.15	87.86	93.91	74.95	105	0.41k
● ViT-L/16 [14]	85.30	88.62	94.39	72.72	32	0.18k
● BiT-R152x4 [22]	85.39	—	94.04	70.64	26	0.94k

- In general, it is a **Universal Function Approximator (but so are other ML models, i.e. Decision Trees)**. It can theoretically fit "any" function (some caveats). Thus, it is very powerful, but resource-hungry.

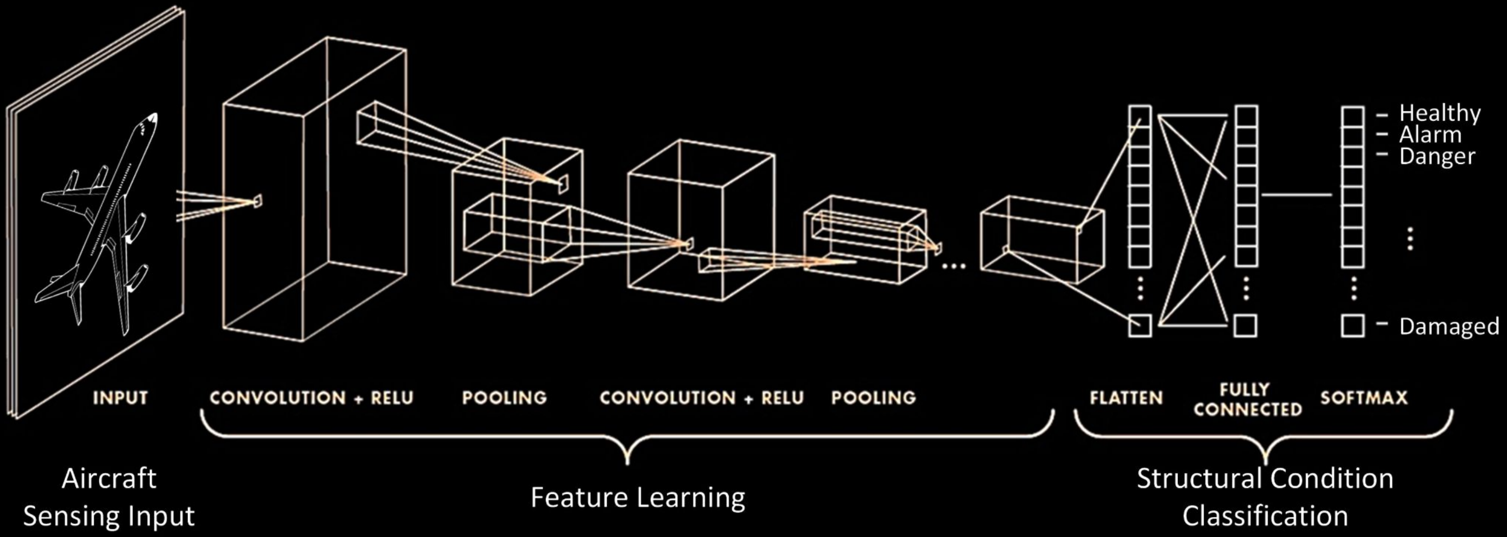
- Apparently, if one is creative, it can outperform State of the Art (SOTA) CNNs and Transformers

Let’s try it in Python...

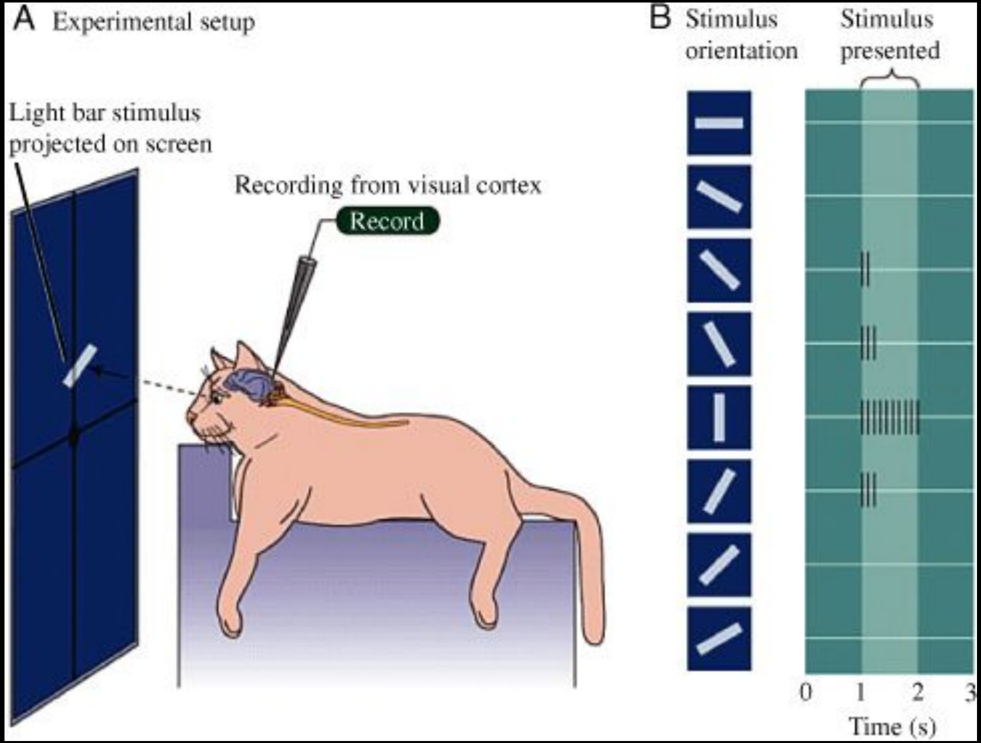
Part 2: Convolutional Neural Networks (CNNs)

Deep learning?

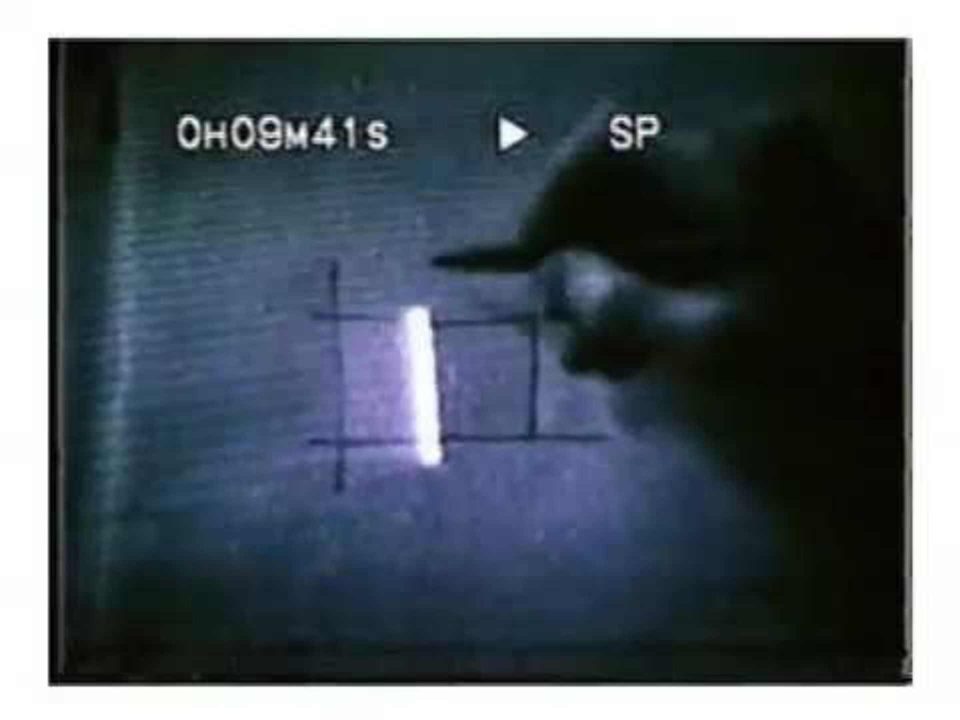
- Simply, this can be a MLP with 2 or more hidden layers
- But this isn't what you think about when I say "Deep Learning"
- I'll talk about Convolutional Neural Networks (CNNs) here



Hubel and Wiesel (1959)

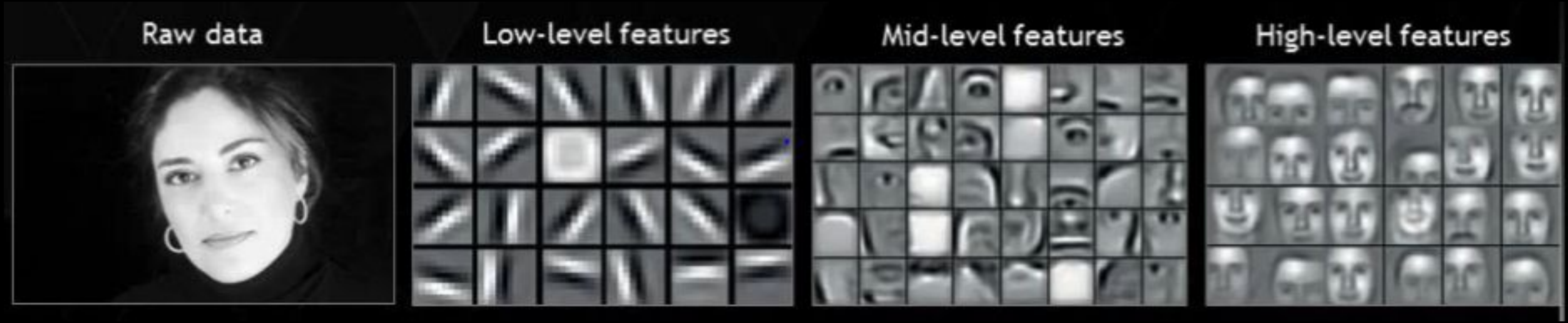


Hubel and Wiesel (1959)

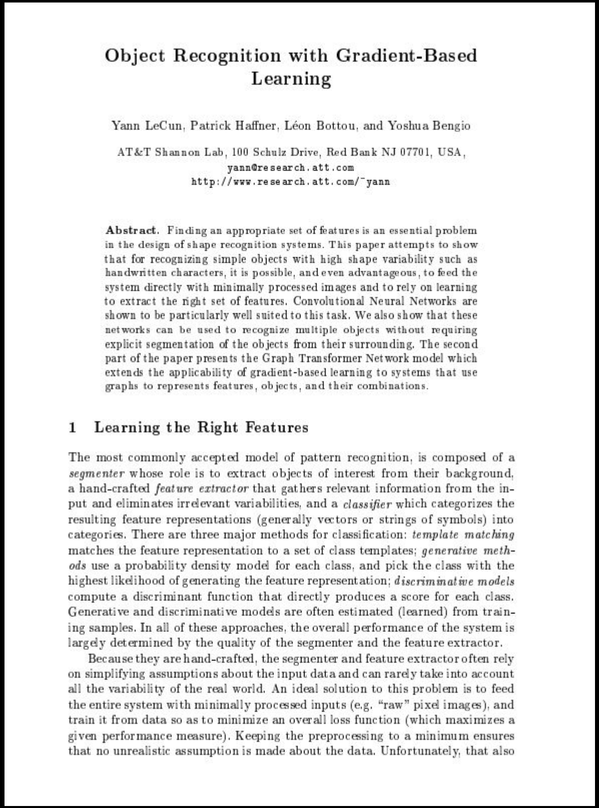


The idea

- Early stages of vision detect low-level features (motion detection)
- Later stages add complexity (i.e. shape of a face, then facial features)
- CNNs have actually been used here as a model of the primary visual cortex (V1).
    - Here's a recent paper on the topic by Grace Lindsay: **https://arxiv.org/pdf/2001.07092.pdf**



Yann LeCunn (1998)



Object Recognition with Gradient-Based Learning

Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio

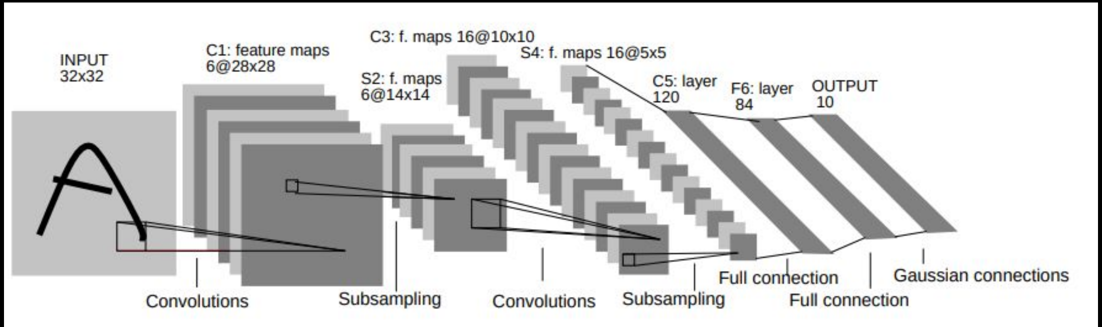
AT&T Shannon Lab, 100 Schulz Drive, Red Bank NJ 07701, USA,
yann@research.att.com
http://www.research.att.com/~yann

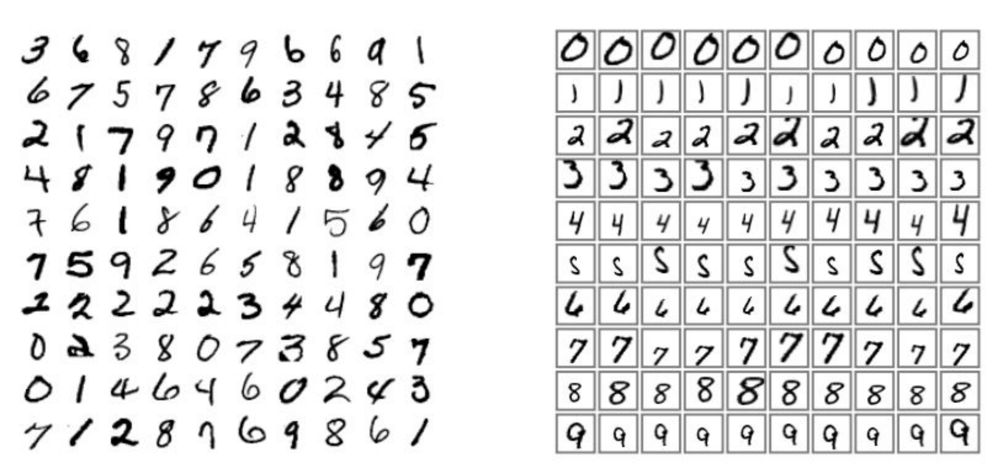
Abstract. Finding an appropriate set of features is an essential problem in the design of shape recognition systems. This paper attempts to show that for recognizing simple objects with high shape variability such as handwritten characters, it is possible, and even advantageous, to feed the system directly with minimally processed images and to rely on learning to extract the right set of features. Convolutional Neural Networks are shown to be particularly well suited to this task. We also show that these networks can be used to recognize multiple objects without requiring explicit segmentation of the objects from their surrounding. The second part of the paper presents the Graph Transformer Network model which extends the applicability of gradient-based learning to systems that use graphs to represents features, objects, and their combinations.

1 Learning the Right Features

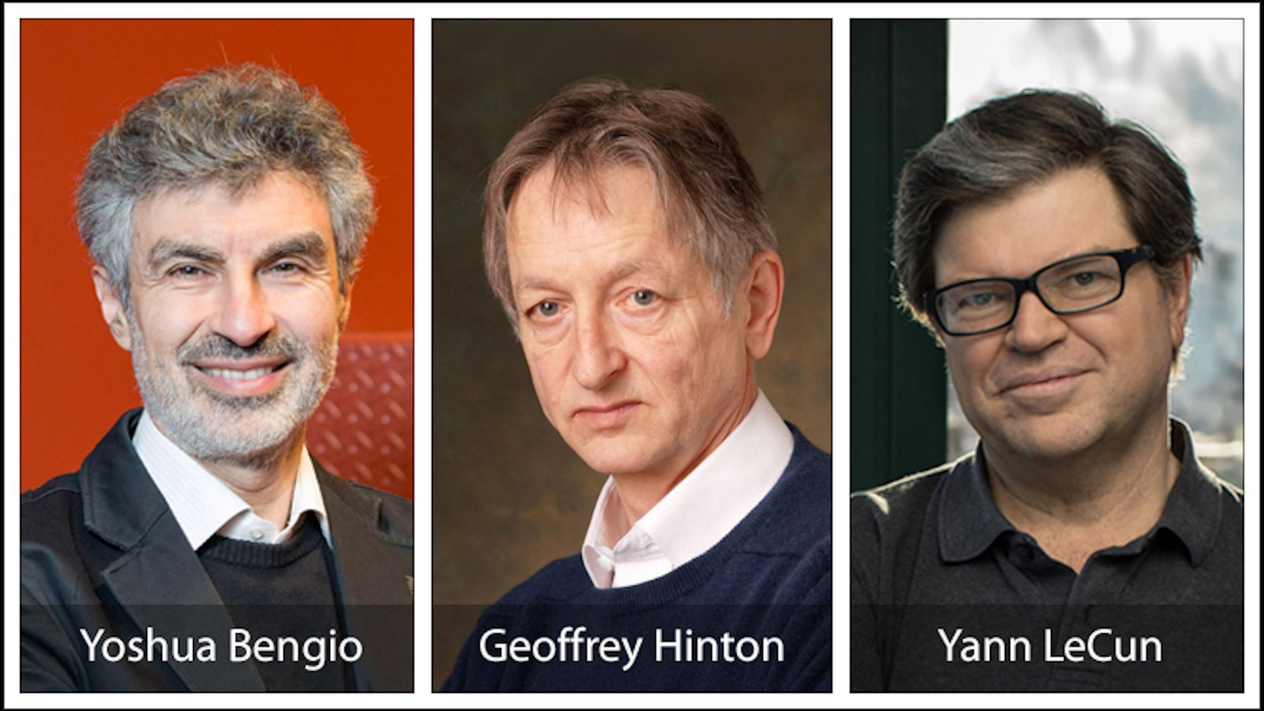
The most commonly accepted model of pattern recognition, is composed of a *segmenter* whose role is to extract objects of interest from their background, a hand-crafted *feature extractor* that gathers relevant information from the input and eliminates irrelevant variabilities, and a *classifier* which categorizes the resulting feature representations (generally vectors or strings of symbols) into categories. There are three major methods for classification: *template matching* matches the feature representation to a set of class templates; *generative methods* use a probability density model for each class, and pick the class with the highest likelihood of generating the feature representation; *discriminative models* compute a discriminant function that directly produces a score for each class. Generative and discriminative models are often estimated (learned) from training samples. In all of these approaches, the overall performance of the system is largely determined by the quality of the segmenter and the feature extractor.

Because they are hand-crafted, the segmenter and feature extractor often rely on simplifying assumptions about the input data and can rarely take into account all the variability of the real world. An ideal solution to this problem is to feed the entire system with minimally processed inputs (e.g. "raw" pixel images), and train it from data so as to minimize an overall loss function (which maximizes a given performance measure). Keeping the preprocessing to a minimum ensures that no unrealistic assumption is made about the data. Unfortunately, that also





The Godfathers of Deep Learning



Hinton: backprop, t-SNE, boltzmann machines

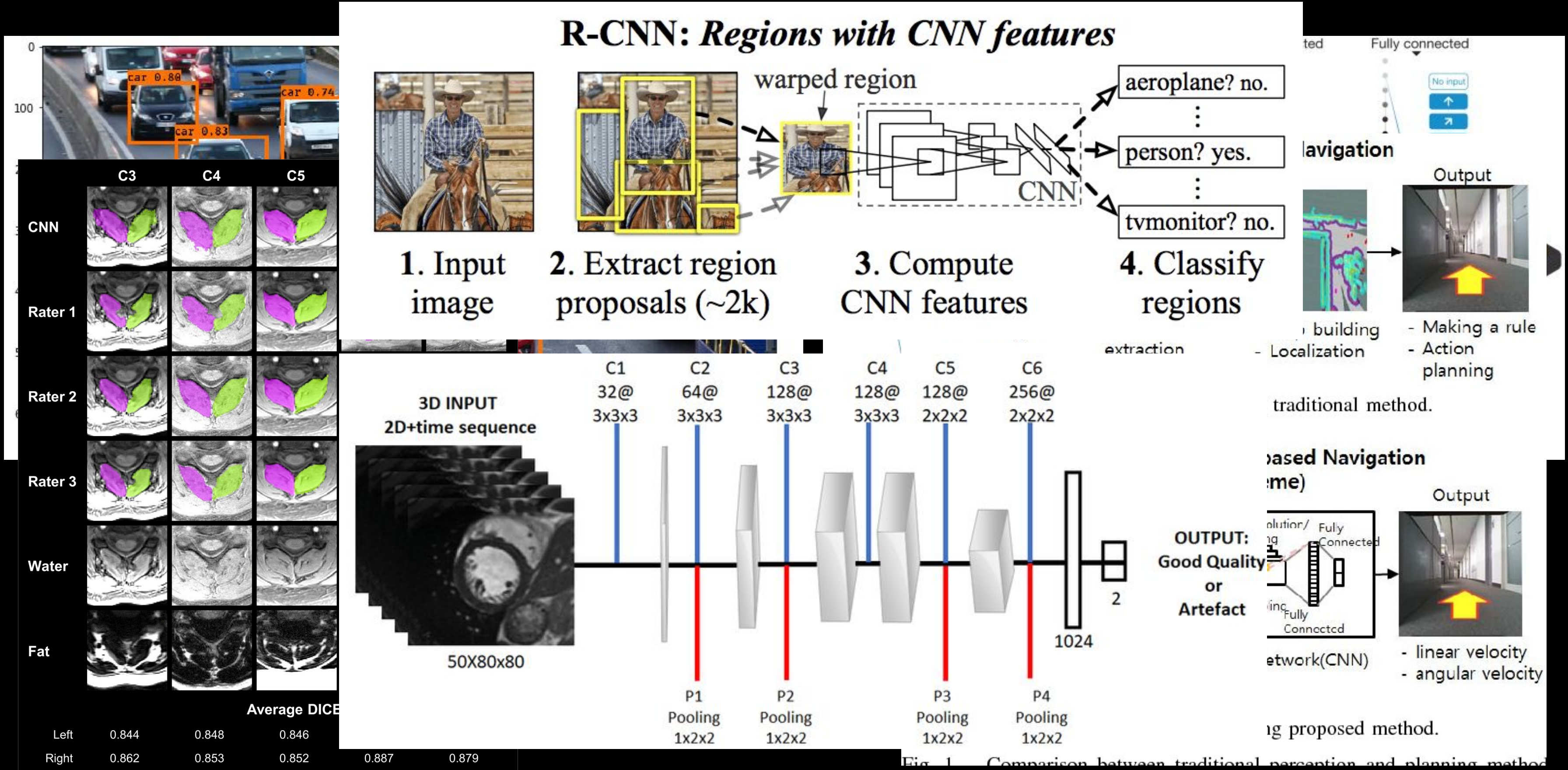
LeCun: CNNs

Bengio: document recognition, Attention (Transformers)

Other important figures:

- Andrew Ng (invented Google Brain)
- Ian Goodfellow (inventor of GANs)

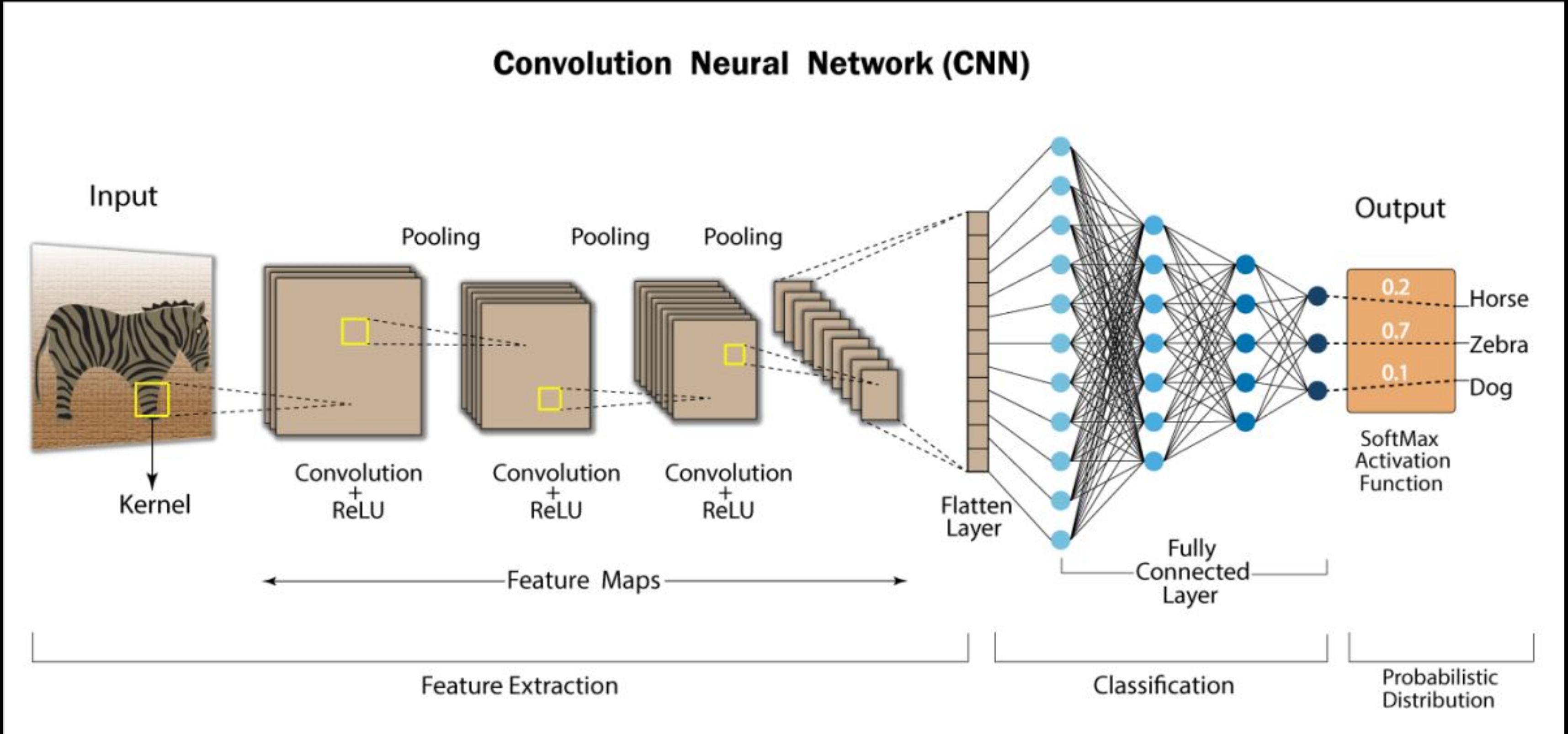
What can a CNN do?



Average DICE

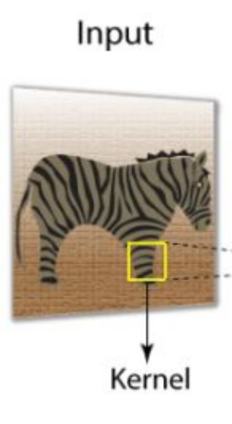
	C3	C4	C5		
Left	0.844	0.848	0.846		
Right	0.862	0.853	0.852	0.887	0.879

Your first CNN (CIFAR-10)



Break-down of Major steps

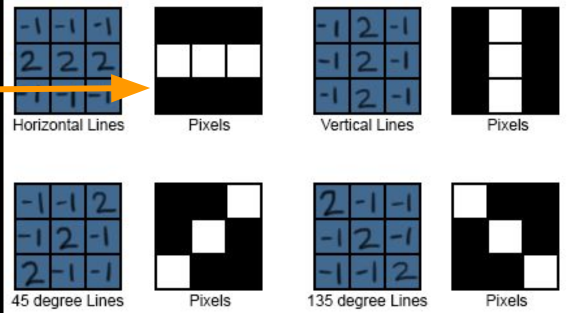
What is a kernel?



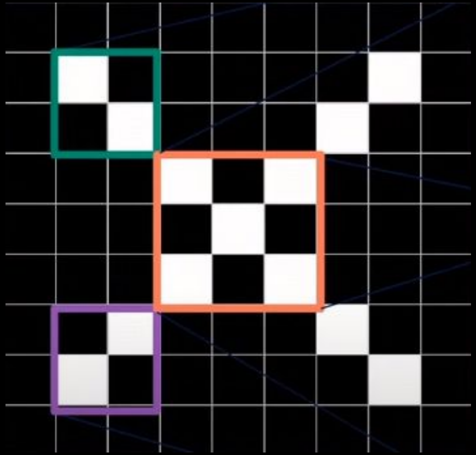
Answer: It’s a filter (2d-array) used to extract features from images

The kernel (which represents a particular feature) moves along the image. The process of moving this kernel across the whole image is called a **convolution**.

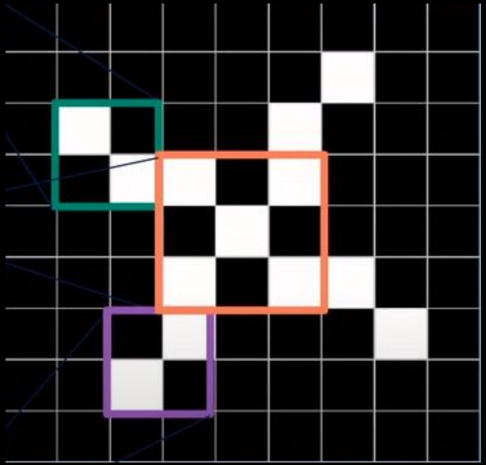
Terminology: feature



The point of filters



They let us match pieces (features) of an image on a particular image patch.



Filtering Example

Let’s convolve a filter on part of an image (this is called filtering, the whole thing is a 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

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

→

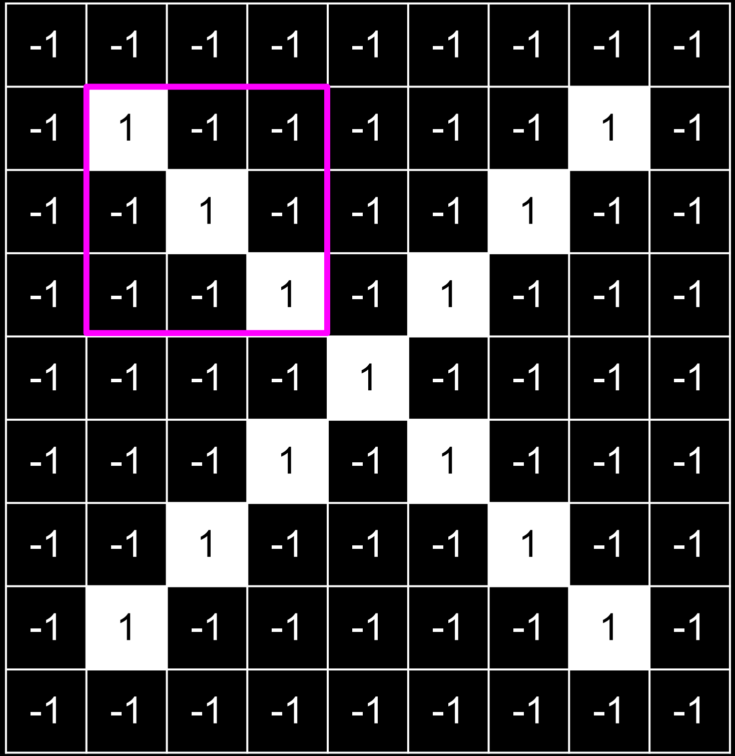
1	1	1
1	1	1
1	1	1

1*1=1
-1*-1=1

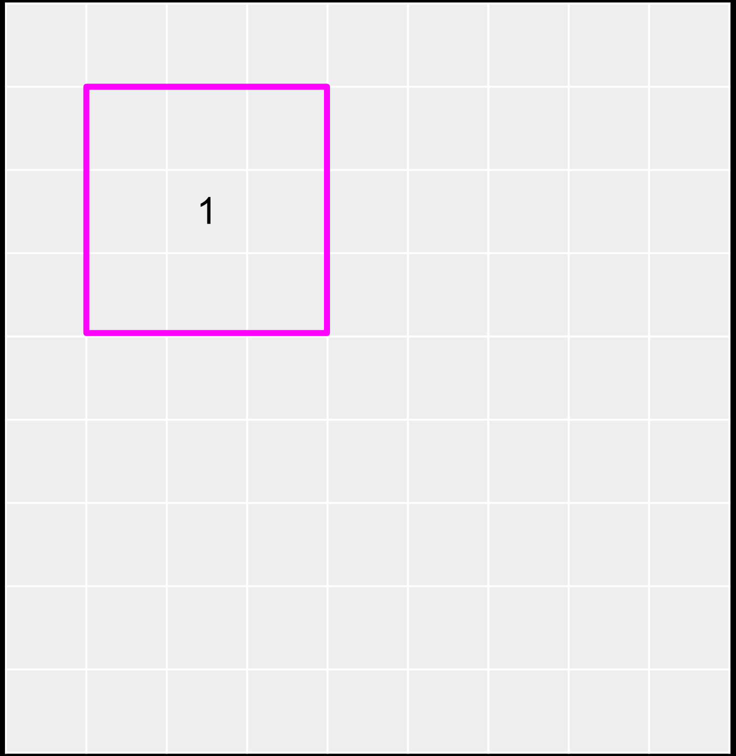
(1+1+1+1+1+1+1+1+1)/9=1

Filtering Example

Let's convolve a filter on part of an image (this is called filtering, the whole thing is a convolution)



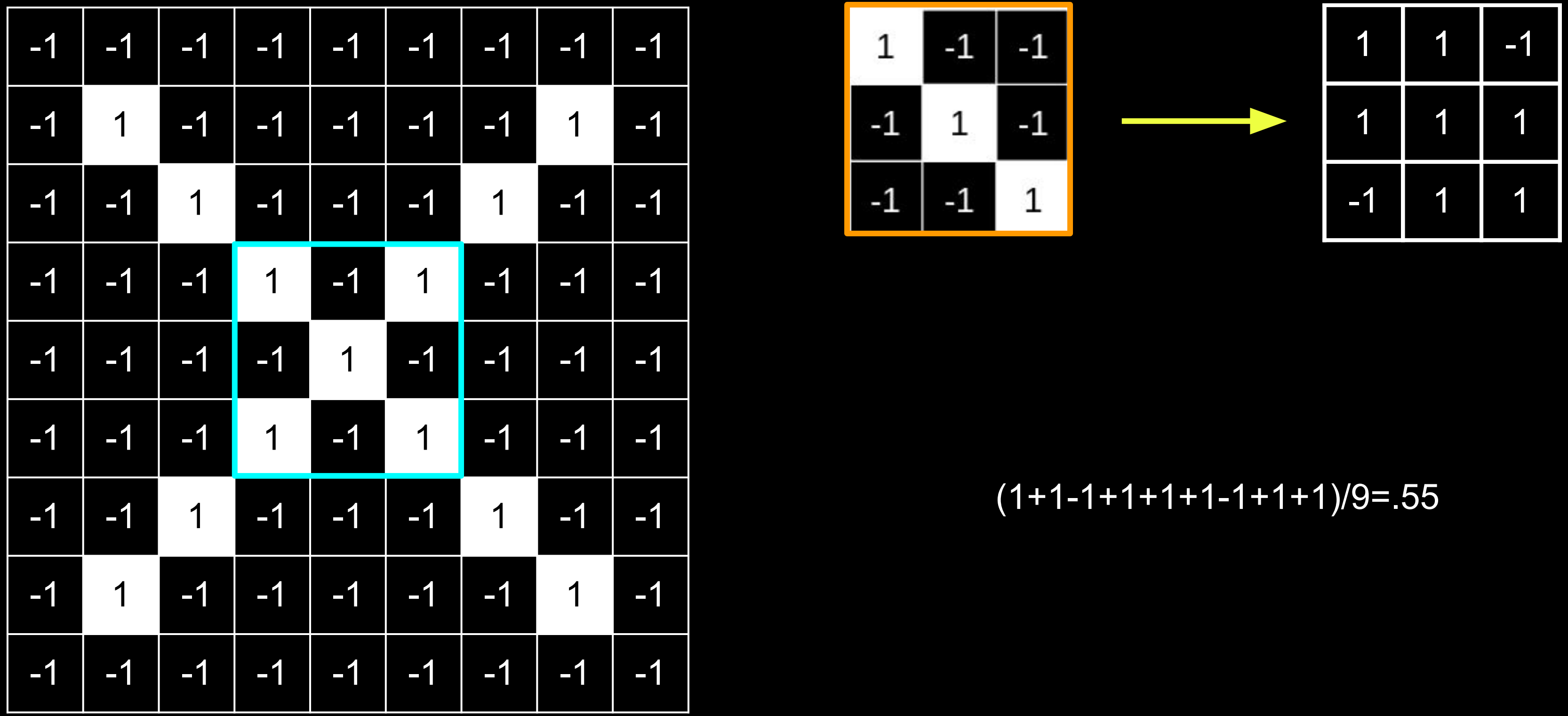
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1



		1						

Filtering Example

Let's convolve a filter on part of an image (this is called filtering, the whole thing is a 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

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

1	1	-1
1	1	1
-1	1	1

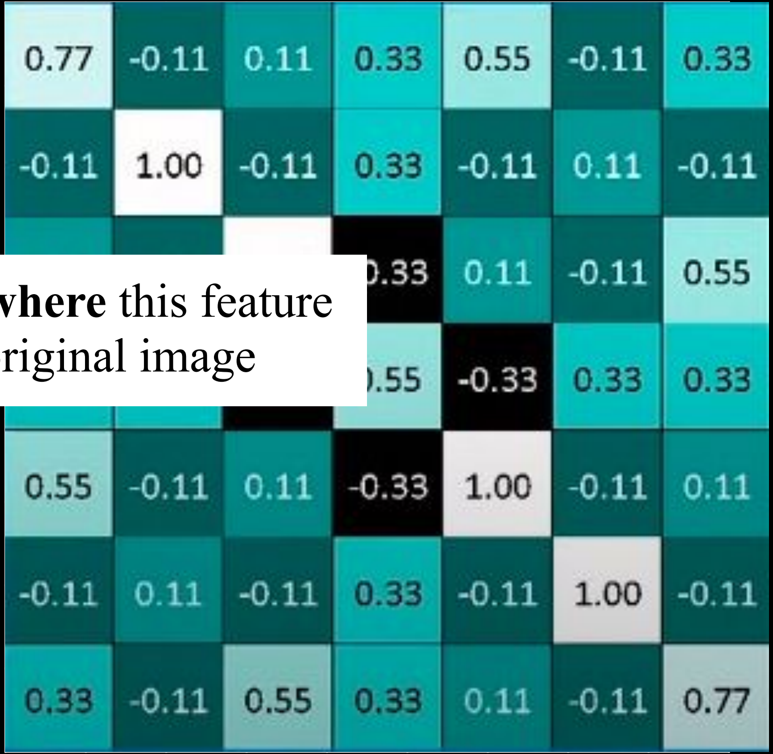
(1+1-1+1+1+1-1+1+1)/9=.55

Filtering Example

Let’s convolve a filter on part of an image (this is called filtering, the whole thing is a 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	[illegible]	[illegible]	[illegible]	[illegible]
-1	-1	-1	-1	1	[illegible]	[illegible]	[illegible]	[illegible]
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

The point: we now know **where** this feature is located throughout the original image

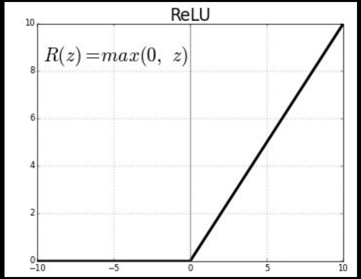


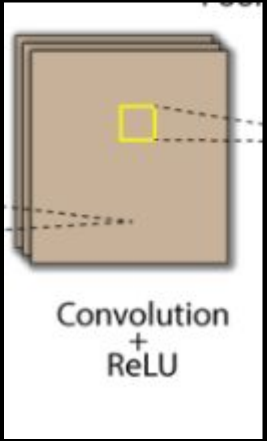
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
[illegible]	[illegible]	[illegible]	.33	0.11	-0.11	0.55
[illegible]	[illegible]	[illegible]	.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

For more on this: https://www.youtube.com/watch?v=JB8T_zN7ZC0

Break-down of Major steps

What is a Convolution + ReLU?

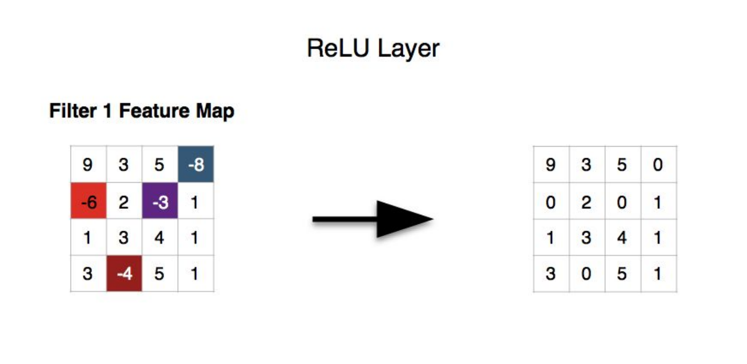




Convolution? After moving each filter across the entire image, we get a convolved image → Convolution

What is ReLU? Recall that this is an **activation function**. It converts all negative values to 0.

Notice that we have a stack of images. This one corresponds to 3 separate r,g,b images. Subsequent stacks correspond to the number of filters we try.



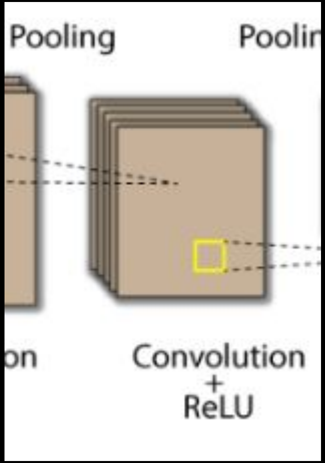
ReLU Layer

Filter 1 Feature Map

9	3	5	-8
-6	2	-3	1
1	3	4	1
3	-4	5	1

9	3	5	0
0	2	0	1
1	3	4	1
3	0	5	1

(Max) Pooling



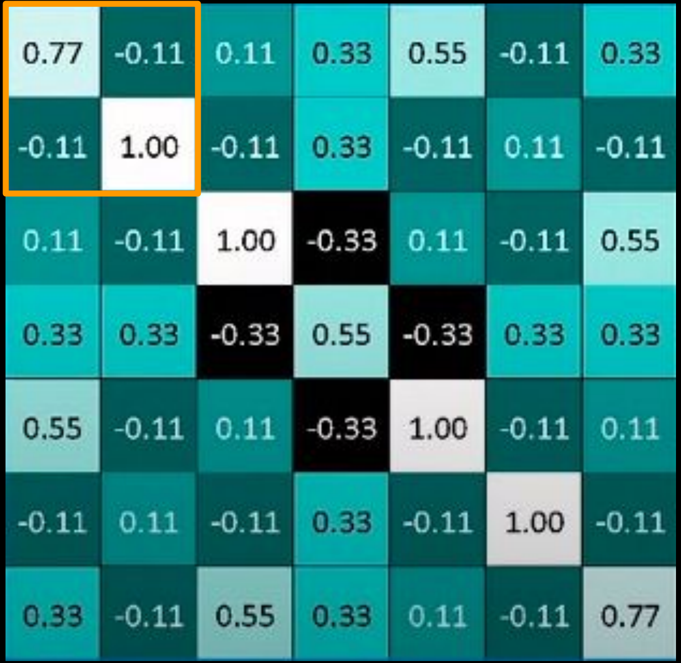
What is Max Pooling?

- For a given window and stride (step size), move the window across the convolved image, taking max values.
- Pad boundaries with zeros

Why do Max Pooling?

- Shrinks the image while retaining sharp features
- **Reduces model variance**, also reduces computation time. Note: Pooling is faster than convolution.

(Max) Pooling Example

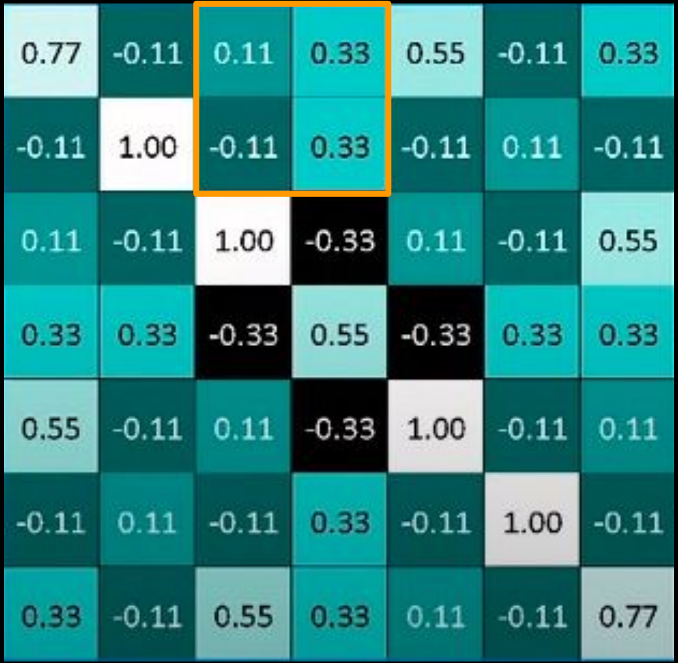


0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

*stride here is 2

1			

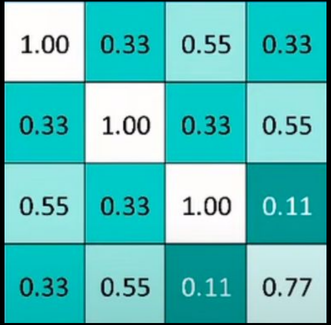
(Max) Pooling Example



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

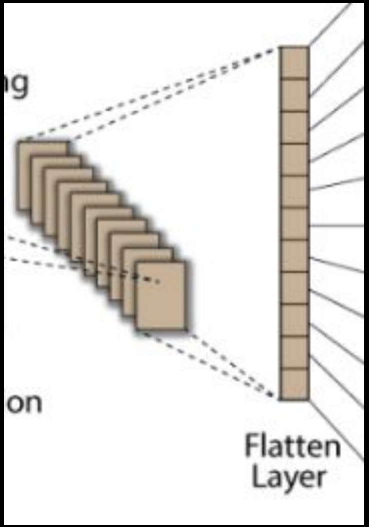
*stride here is 2

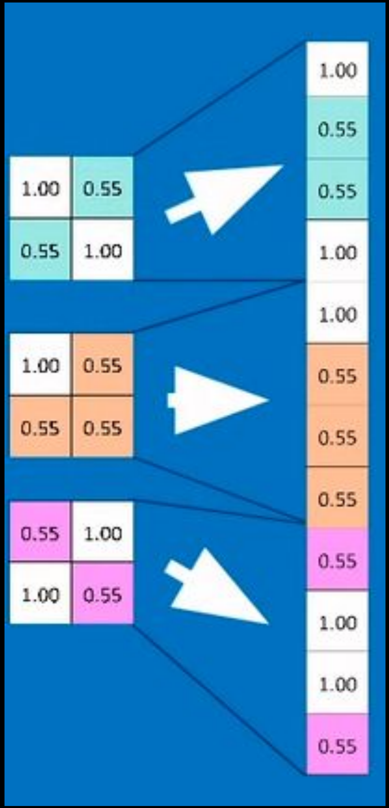
1	.33		



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

Flatten the filters, select output with MLP



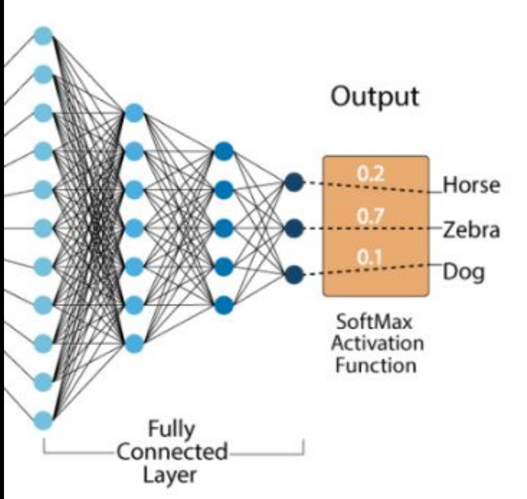


1.00	0.55
0.55	1.00

1.00	0.55
0.55	0.55

0.55	1.00
1.00	0.55

1.00
0.55
0.55
1.00
1.00
0.55
0.55
0.55
0.55
1.00
1.00
0.55



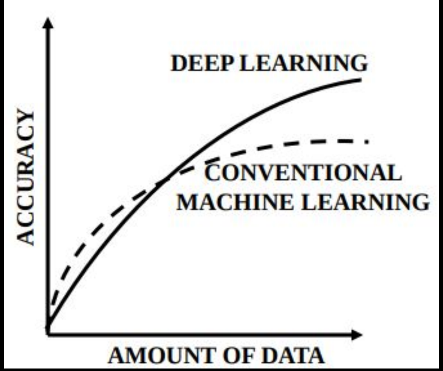
How do we train the weights? **Backprop. This applies to the MLP, and also the Convolutional and Pooling layers!**

More reading (optional):
https://cs231n.github.io/convolutional-networks/

Let’s try it in Python...

Deep Learning vs. other ML Techniques

- Deep Learning is **data-hungry and resource hungry**, but it is exactly this which makes it appear in so many SOTA implementations
- Well suited to unstructured data (i.e. images), but also works well with structured data
- Not a magical dragon to slay, just another (powerful) technique to have in your toolbox
- When in doubt (with structured data), start with a random forest. Why? It’s quicker to implement (my opinion).



Summary

- Neural Networks, MLP
  - Some history
  - How it executes
  - Backprop
  - Activation Functions
  - Regularization with Dropout
  - Optimizers
- Convolutional Neural Nets:
  - Some history
  - Implementation: step breakdown
  - Deep Learning vs. other ML