Lecture 11: Convolutional neural networks

Announcements:

- HW #4 is due **Monday, Feb 17**, uploaded to Gradescope. To submit your Jupyter Notebook, print the notebook to a pdf with your solutions and plots filled in. You must also submit your .py files as pdfs.

- Midterm review session: Young Hall CS76 on February 14, from 4pm-6pm.

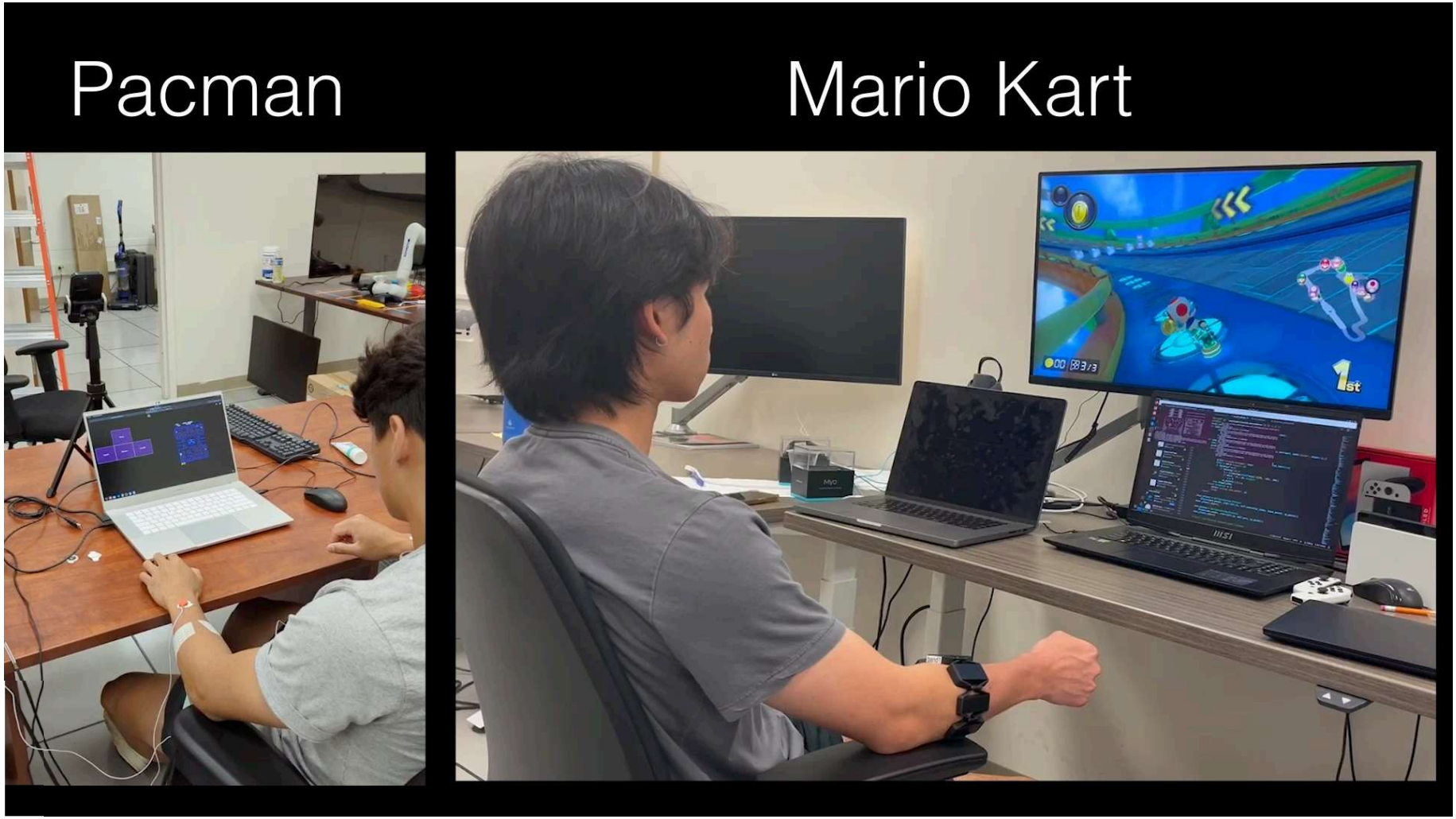
- The midterm is in 9 days on February 19, 2025.

  - You are allowed 4 cheat sheets (each an 8.5 x 11 inch paper). You can fill out both sides (8 sides total). You can put whatever you want on these cheat sheets.
  - It will cover material up to and including lecture on February 12, 2025.
  - Past exams are uploaded to Bruin Learn (under “Modules” —> “past exams”).
  - You may bring a calculator to the exam.
  - You **must do the exam in pen.**
  - MSOL students will take the exam remotely on Saturday, February 22, from 2-3:50pm. We will send out details to the MSOL students closer to the date.

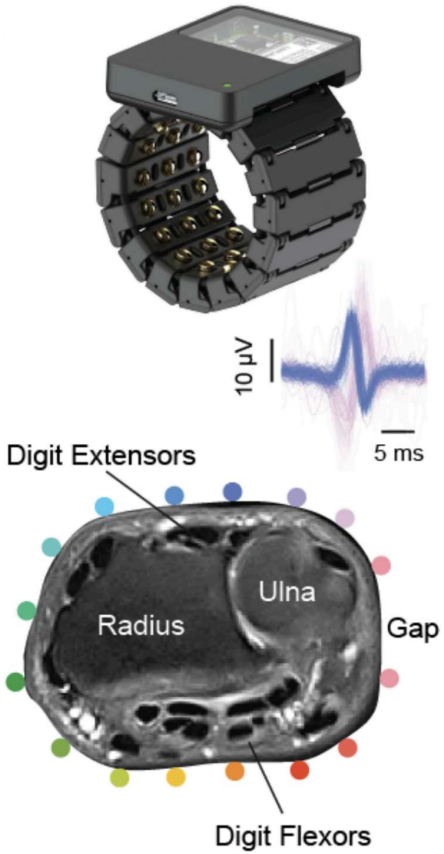
- If you have CAE accommodations, you must request CAE to provide them.

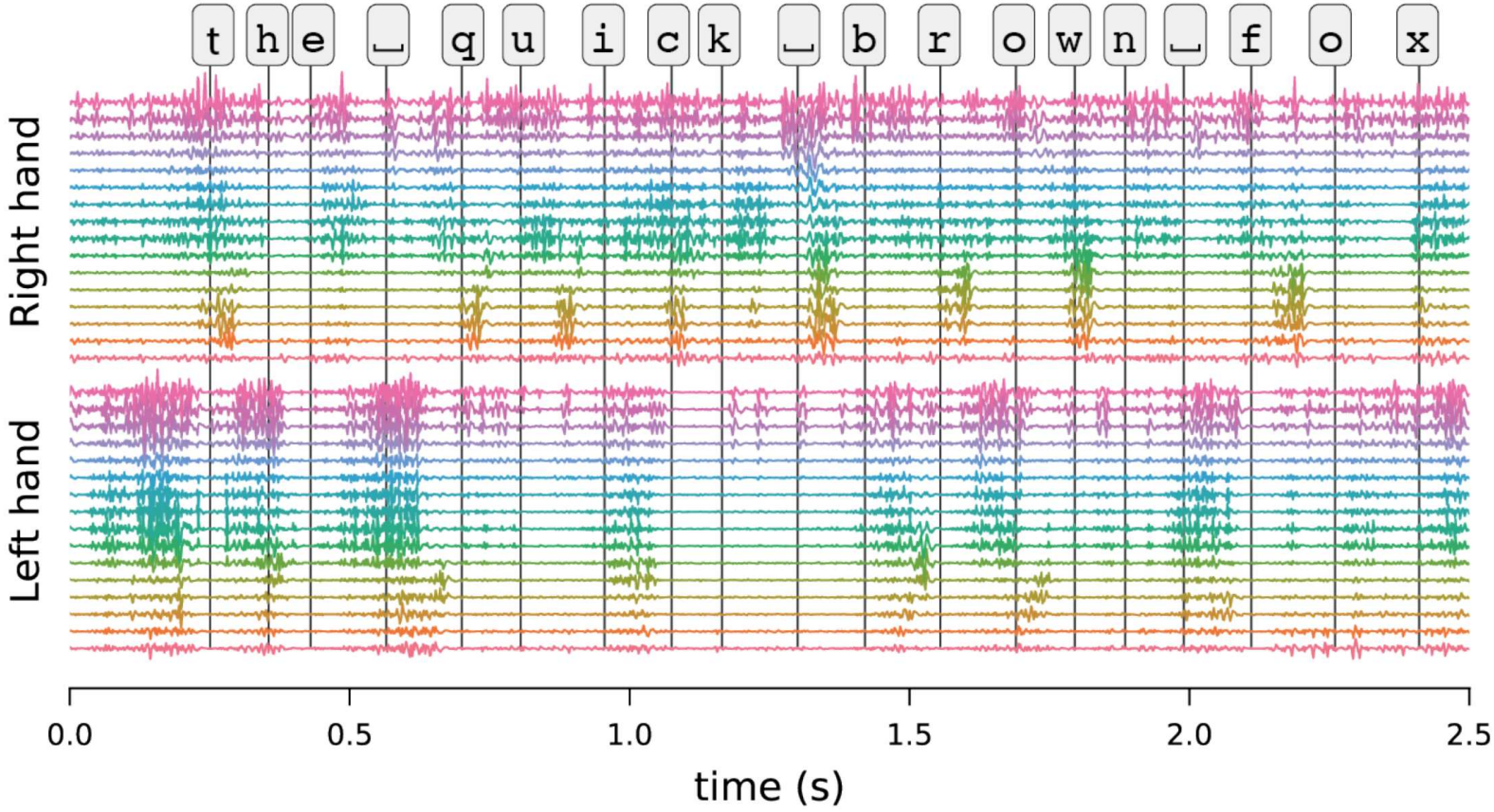
ECE C147/247 project

Project motivation:



ECE C147/247 project





This data is less than a year old — released Oct 26, 2024.

ECE C147/247 project

Project:

- The goal will be to develop deep learning architectures for EMG decoding.
  - Key thing to be tested is to explore post-CNN architectures.
- You can also submit your own project proposal. This must be submitted by Feb 24, 2025.
- You can use ANY software packages out there. We will have an inverted lecture on PyTorch + the project on Monday, February 24, 2025. This will replace our scheduled "Deep Learning Libraries" lecture in the syllabus. We've released a PyTorch notebook and video to teach you how to use PyTorch.
  - You can also use PyTorch Lightning, JAX, etc. any software package is fine.
- Caveat: **We are still uploading some of the data**, but they will be uploaded to the box links provided in the project document. The single user data for the "baseline" project is uploaded. You can always download all the user data following the instructions in the emg2qwerty github repo. We will upload the data in easier-to-download and smaller chunks. Finally, Prof. Kao will upload some custom collected EMG data for a potential project.

ECE C147/247 project

Project:

- The “baseline” project.
    - We picked out one subject, #89335547. You can train an sEMG typing decoder using Meta’s codebase on this subject in ~1 hr on Google Colab, so this dataset should not be computationally burdensome.

ECE C147/247 project

"Baseline" project suggested directions (using single or few user data)

Below, we detail suggested directions you should pursue for the "baseline" project. This project is what we expect most students will do, and it will certainly be possible to score the maximum points on the project following these baseline project guidelines. We suggest you to explore the following directions:

1. Train, evaluate, and subsequently compare different architectures to reduce the test CER on the provided single subject, ID #89335547. You must experiment with at least 1 recurrent architecture (e.g. RNN, LSTM, GRU). Projects that evaluate more architectures, such as RNN+CNN hybrids, transformers, or other architectures, will receive more creativity points in the rubric below.

   Please note that we have already split the data for subject #89335547 into train/val/test splits at `https://github.com/joe-lin-tech/emg2qwerty/blob/main/config/user/single_user.yaml`. Please use these splits for the project.

2. Experiment with various data pre-processing or augmentation techniques.

3. How many channels are needed to achieve good decoding performance? Investigate relationship between the number of electrode channels and CER.

4. How much data is needed to achieve good decoding performance? Investigate relationship between the amount of training data and CER.

5. How fast does sEMG data need to be sampled for good performance? Investigate the relationship between sampling rate and CER.

Feel free to innovate beyond the components above to earn points for creativity and insight. Extra insight points may also be rewarded for explaining how these different approaches result in better or worse performance.

ECE C147/247 project

"State of the art" (SOTA) project:

- The SOTA project is to develop new state-of-the-art performance on this dataset, which is more feasible because this dataset is new. This will require working with the entire emg2qwerty dataset and making progress on impactful questions.

- If you are interested in potentially turning this project into a conference paper, you should attempt the SOTA project.

- To support this, a PhD student in my lab (Nima) working on these directions is happy to help provide input. If you would like our help advising a project, and we feel it is promising, impactful, and would meet the threshold of a conference paper, we will be happy to directly advise and support some projects (not all — we have limited bandwidth).

    - This may include doing further research or real-time experiments within my lab.
    - The submitted project for this course doesn't have to be all the way there — we understand there's just a month to do this project.
    - This is our first time piloting this, and so be patient with us.

- You can also of course do a SOTA project without our input and try to submit it on your own.

- In the project doc, if you do SOTA direction 4, this project would have the greatest chance of turning into experiments within our lab.

ECE C147/247 project

Final project report:

- The output of the project is ultimately a project report, written using the NeurIPS LaTeX template. (Yes, this project must be written in LaTeX. A common user-friendly LaTeX editor is overleaf.com — overleaf is free to use).

- The rubric for grading is part of the project report.

Adam

Adam (cont.)

Initialize $\mathbf{v} = 0$ as the "first moment", and $\mathbf{a} = 0$ as the "second moment." Set β_1 and β_2 to be between 0 and 1. (Suggested defaults are $\beta_1 = 0.9$ and $\beta_2 = 0.999$.) Initialize ν to be sufficiently small. Initialize $t = 0$. Until stopping criterion is met:

- Compute gradient: $\mathbf{g}$
- Time update: $t \leftarrow t + 1$
- First moment update (momentum-like):

$$\mathbf{v} \leftarrow \beta_1 \mathbf{v} + (1 - \beta_1)\mathbf{g}$$

- Second moment update (gradient normalization):

$$\mathbf{a} \leftarrow \beta_2 \mathbf{a} + (1 - \beta_2)\mathbf{g} \odot \mathbf{g}$$

- Bias correction in moments:

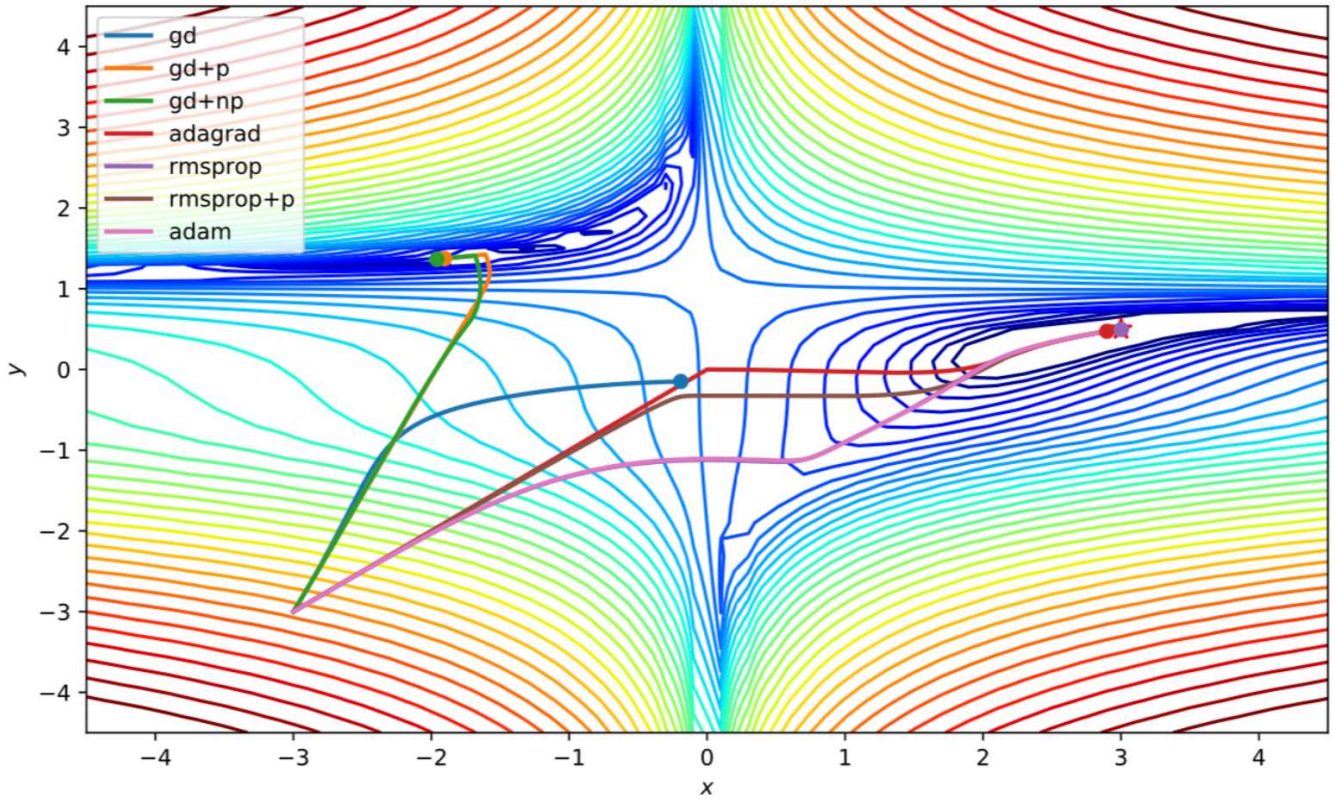
$$\begin{aligned} \tilde{\mathbf{v}} &= \frac{1}{1 - \beta_1^t}\mathbf{v} \\ \tilde{\mathbf{a}} &= \frac{1}{1 - \beta_2^t}\mathbf{a} \end{aligned}$$

- Gradient step:

$$\theta \leftarrow \theta - \frac{\varepsilon}{\sqrt{\tilde{\mathbf{a}}} + \nu} \odot \tilde{\mathbf{v}}$$

https://www.coursera.org/lecture/deep-neural-network/bias-correction-in-exponentially-weighted-averages-XjuhD

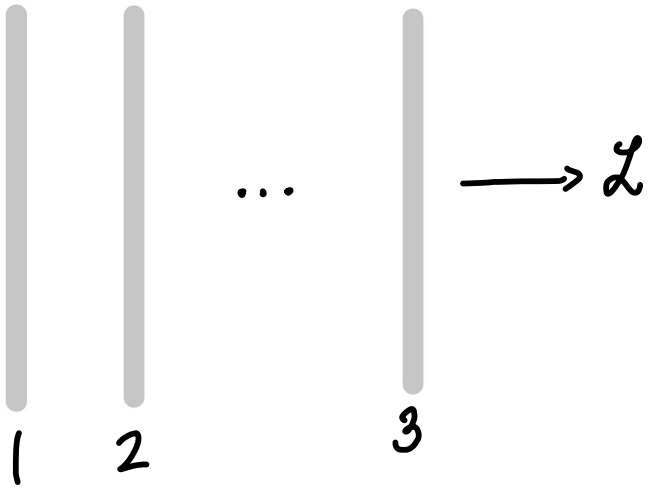
Adam (opt 2)



Video: `http://seas.ucla.edu/~kao/opt_anim/2gd_gd+p_gd+np_adagrad_rmsprop+p_adam.mp4`

Challenges in gradient descent

- **Exploding gradients.**
  Sometimes the cost function can have "cliffs" whereby small changes in the parameters can drastically change the cost function. (This usually happens if parameters are repeatedly multiplied together, as in recurrent neural networks.) Because the gradient at a cliff is large, an update can result in going to a completely different parameter space. This can be ameliorated via gradient clipping, which upper bounds the maximum gradient norm.



amplification of your gradients as you backpropagate more and more.

measure $\|g\|$
↳ if they become too big, scale down to be smaller.

$\|g\| > \text{clip}$
$g \leftarrow g$ (complete)

Challenges in gradient descent

- **Vanishing gradients.**
  Like in exploding gradients, repeated multiplication of a matrix $\mathbf{W}$ can cause vanishing gradients. Say that each time step can be thought of as a layer of a feedforward network where each layer has connectivity $\mathbf{W}$ to the next layer. By layer t, there have been $\mathbf{W}^t$ multiplications. If $\mathbf{W} = \mathbf{U}\Lambda\mathbf{U}^{-1}$ is its eigendecomposition, then $\mathbf{W}^t = \mathbf{U}\Lambda^t\mathbf{U}^{-1}$, and hence the gradient along eigenvector $\mathbf{u}_i$ is shrunk (or grown) by the factor λ_i^t. Architectural decisions, as well as appropriate regularization, can deal with vanishing gradients.

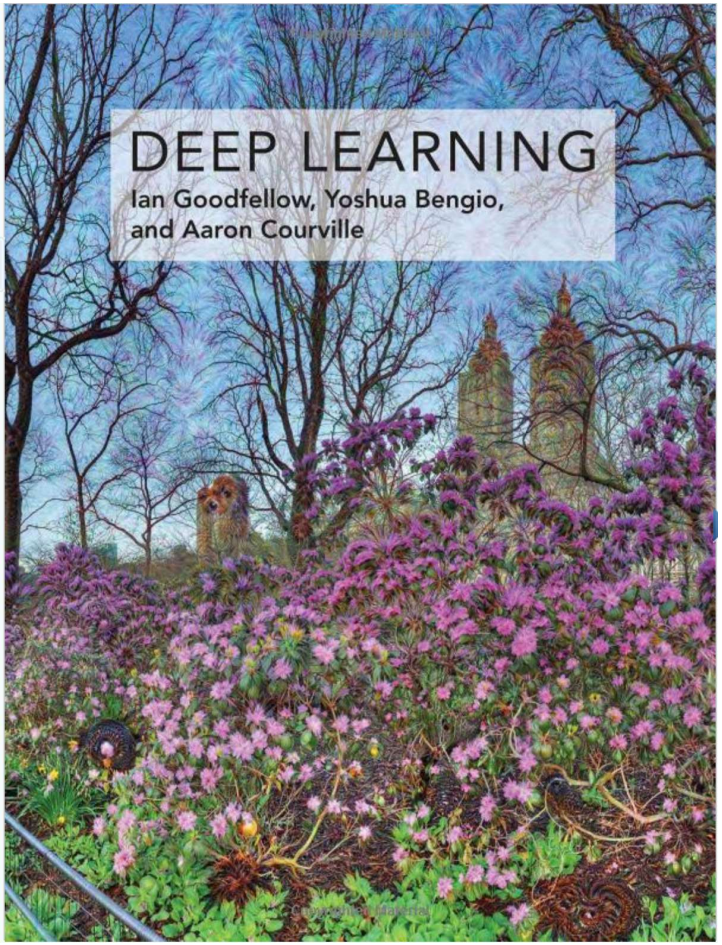
"small" – spectral norm < 1

primary method to avoid exploding / vanishing gradients
↳ use diff. architecture (ResNet)

Convolutional neural networks

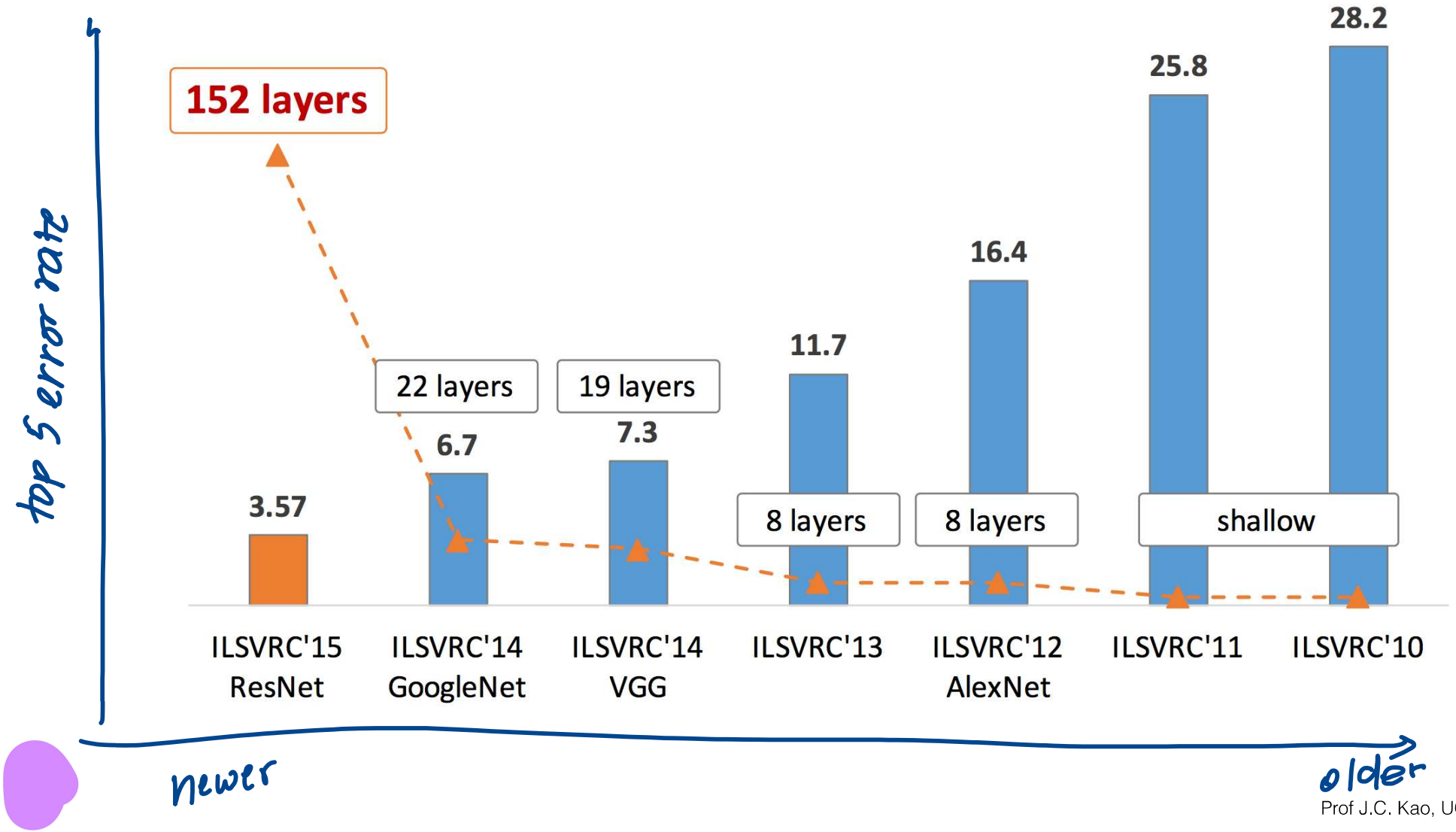
Reading:

Deep Learning, Chapter 9



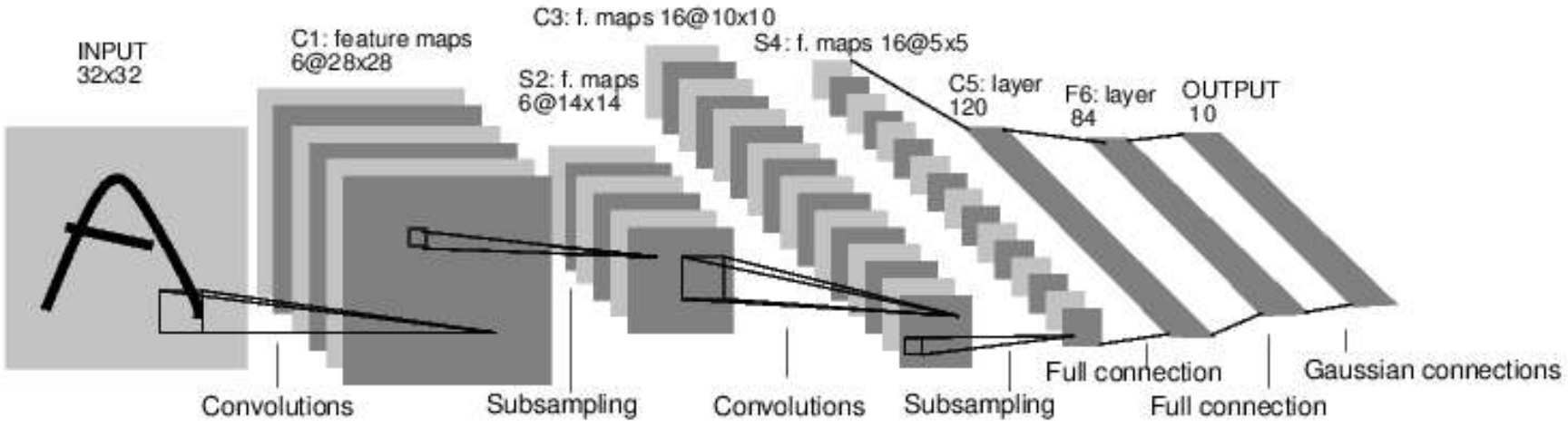
Background: convolutional neural networks

The CNN played a large role in this last "revival" of neural networks (i.e., the past 5 years).



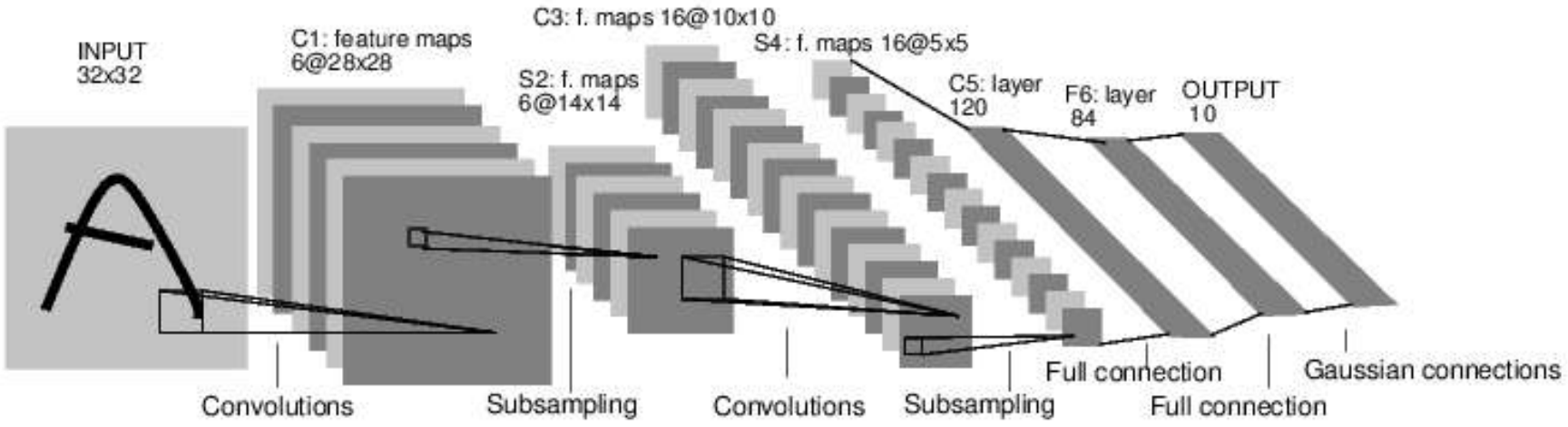
Background: convolutional neural networks

CNNs have been around since the 1990's. In 1998, Yann LeCun introduced LeNet, which is the modern convolutional neural network.



Background: convolutional neural networks

How do we arrive at this architecture? What are the principles that lead us to this?



Biological inspiration for the CNN

Biological inspiration

Principles of the convolutional neural network are inspired from neuroscience. Seminal work by Hubel and Wiesel in cat visual cortex in the 1960's (Nobel prize awarded in the 1980's) found that V1 neurons were tuned to the movement of oriented bars. Hubel and Wiesel also defined "simple" and "complex" cells, the computational properties of which were later well-described by linear models and rectifiers (e.g., Movshon and colleagues, 1978). Three key principles of neural coding in V1 are incorporated in convolutional neural networks (CNNs):

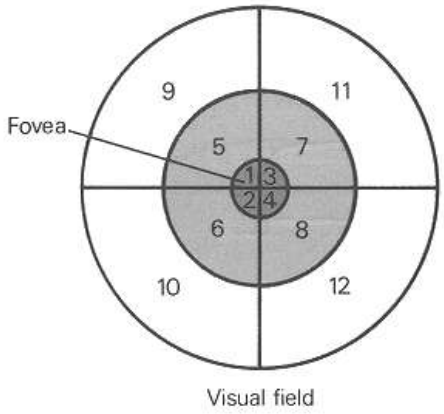
- V1 has a retinotopic map.
- V1 is composed of simple cells that can be adequately described by a linear model in a spatially localzed receptive field.
- V1 is composed of complex cells that respond to similar features as simple cells, but importantly are largely invariant to the position of the feature.

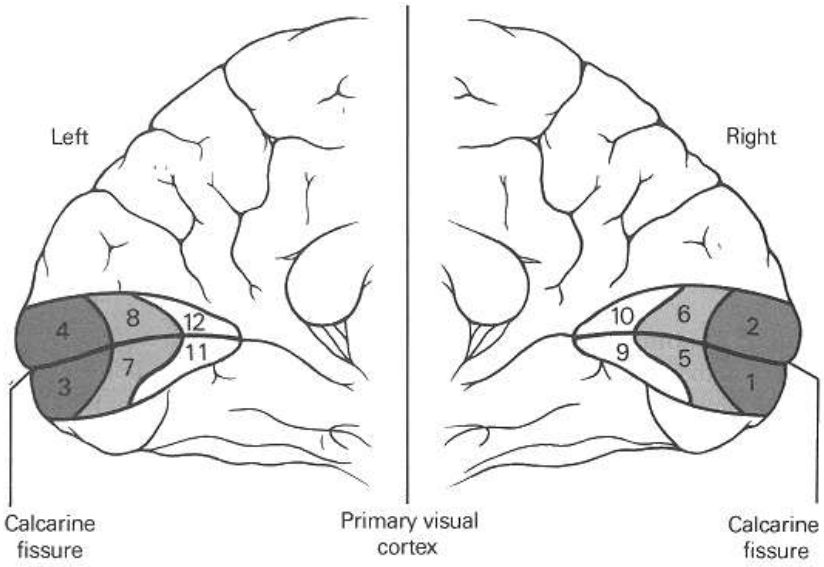
If interested, you can YouTube some videos of their experiments. Warning: they do show cats in an experimental apparatus.

Biological inspiration for the CNN

- Retinotopic map: CNNs are spatially organized, so that nearby cells act on nearby parts of the input image.

objects that are close together in visual field fire neurons close together in the brain
↓
"spatially local"





Biological inspiration for the CNN

- Simple cells: CNNs use spatially localized linear filters, which are followed by thresholding.
- Complex cells: CNNs use pooling units to incorporate invariance to shifts of the position of the feature.

Biological inspiration for the CNN

Do CNN's compute like the visual system?

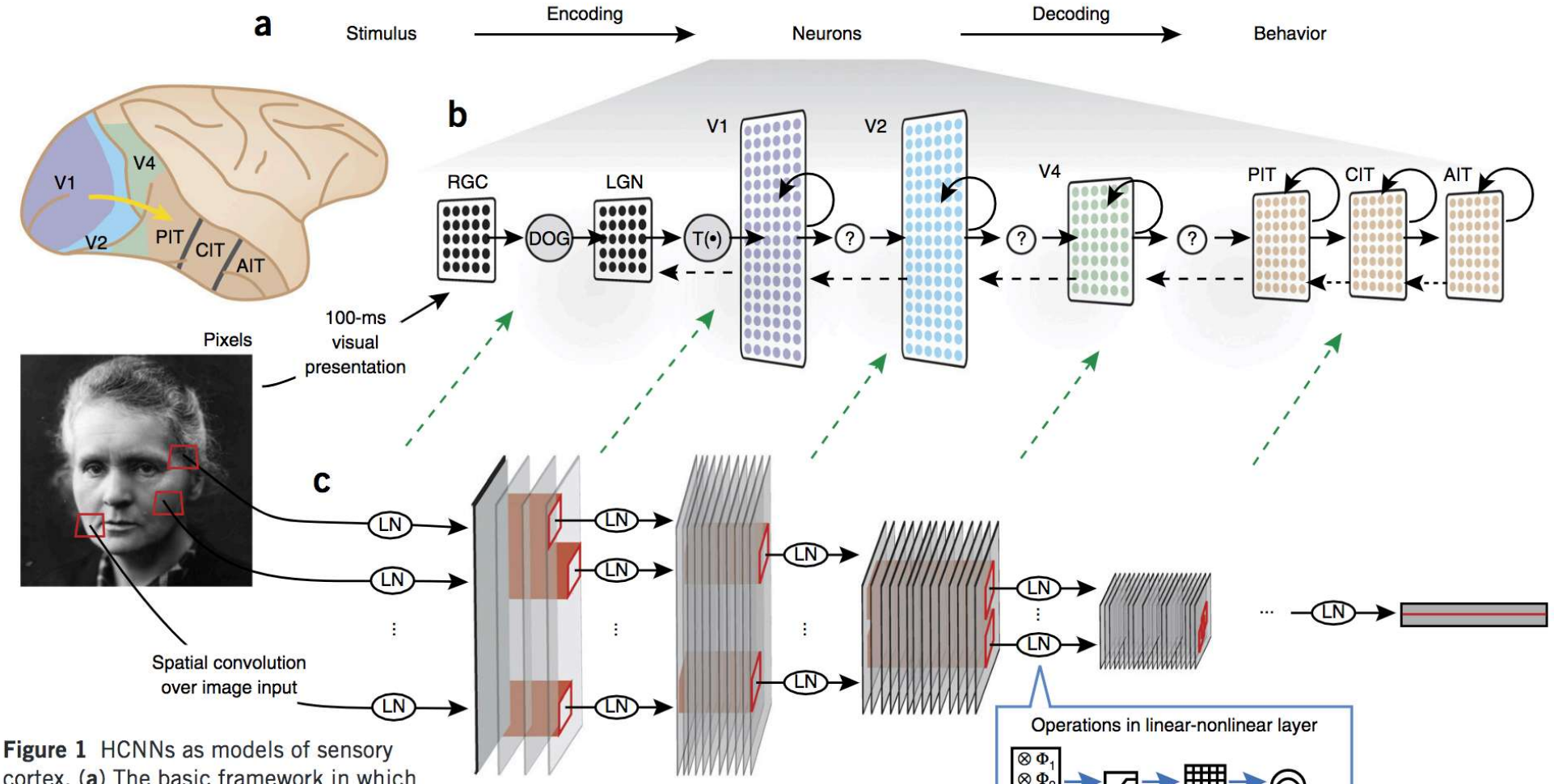
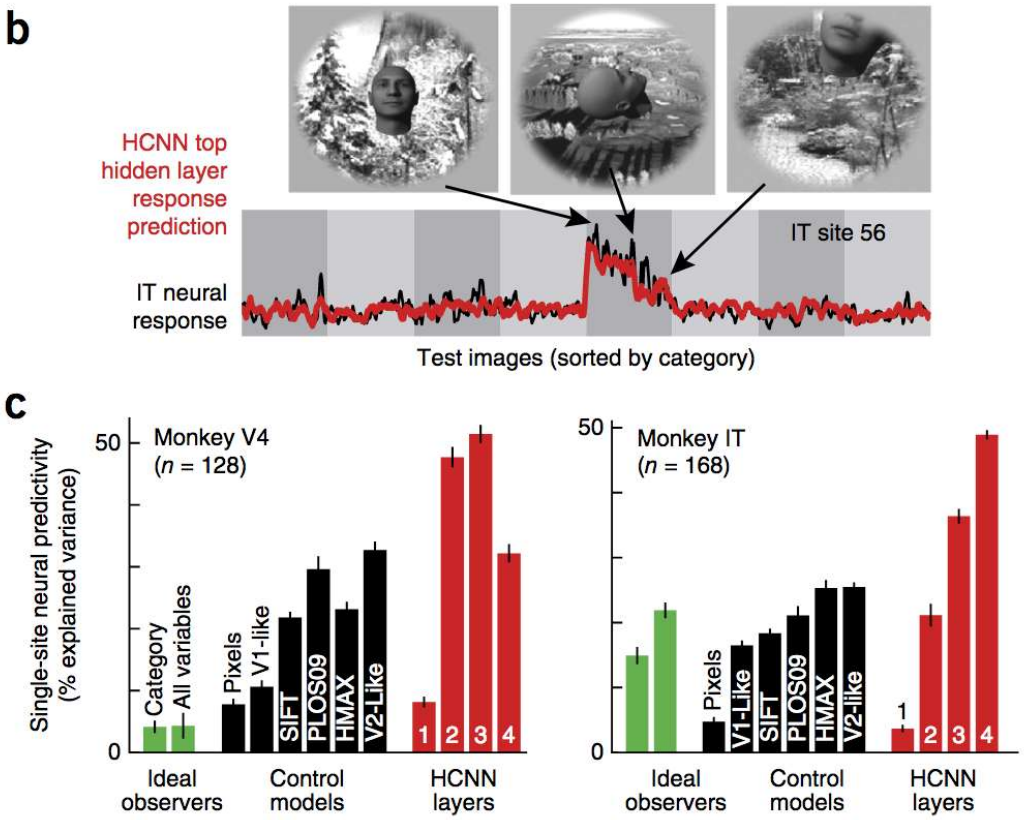


Figure 1 HCNNs as models of sensory cortex. (**a**) The basic framework in which sensory cortex is studied is one of encoding—the process by which stimuli are transformed into patterns of neural activity—and decoding, the process by which neural activity generates behavior. HCNNs have been used to make models of the encoding step; that is, they describe

Biological inspiration for the CNN

Do CNN's compute like the visual system?



Biological inspiration for the CNN

Do CNN's compute like the visual system?

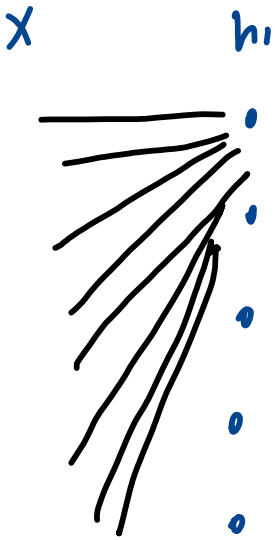
Limitations of biological analogies

There are several limitations in these analogies. For example,

- CNNs have no feedback connections; neural populations are recurrently connected.
- The brain foveates a small region of the visual space, using saccades to focus on different areas.
- The output of the brain is obviously more than just image category classification.
- Pooling done by complex cells is an approximate way to incorporate invariance.

Motivation for convolutional neural networks

- For images, fully connected networks require many parameters. In CIFAR-10, the input size is $32 \times 32 \times 3 = 3072$, requiring 3072 weights for each neuron. For a more normal sized image that is 200×200, each neuron in the *first layer* would require 120000 parameters. This quickly gets out of hand. This network, with a huge number of parameters, would not only take long to train, but may also be prone to overfitting.



3072 · #artificial neurons in layer 1

CNNs help us avoid large parameter explosion that comes from fully connecting neural networks

Convolution

The convolution operation

The convolution of two functions, $f(t)$ and $g(t)$, is given by:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

In discrete time, this is given by:

$$(f * g)(n) = \sum_{m=-\infty}^{\infty} f(m)g(n-m)$$

in practice, this is a + sign in a CNN.

convolution is commutative and associative

Correlation and convolution

Note, however, that in general CNNs don't use *convolution*, but instead use *cross-correlation*. Colloquially, instead of "flip-and-drag," CNNs just "drag." For real-valued functions, cross-correlation is defined by:

$$(f \star g)(n) = \sum_{m=-\infty}^{\infty} f(m)g(n+m)$$

We'll follow the field's convention and call this operation convolution.

Convolution in 2D

The 2D convolution (formally: cross-correlation) is given by:

$$(f * g)(i, j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f(m, n) g(i + m, j + n)$$

This generalizes to higher dimensions as well. Note also: these "convolutions" are not commutative.

Convolution in 2D example

1	1	1
1	1	1
1	1	1

$*$

w	x
y	z

***** All convolutions in this class will be VALID convolutions (the filter and inputs must be totally overlapping). *****

Convolution in 2D example

A - "input"

1	2	3
4	5	6
7	8	9

$*$

B - "filter"

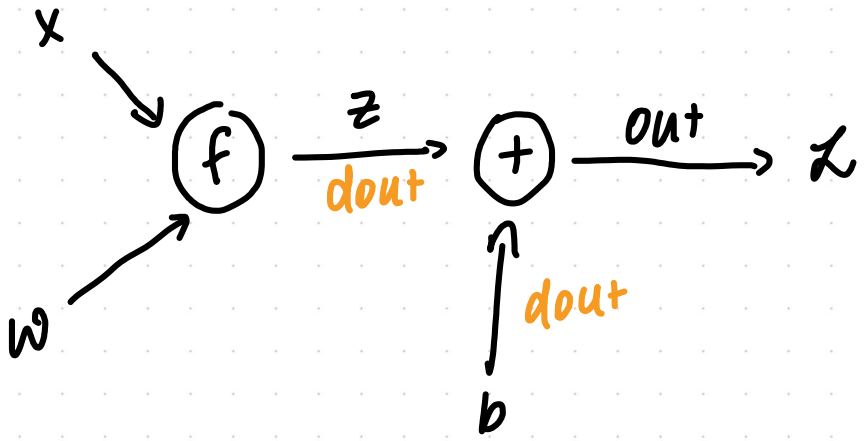
w	x
y	z

$=$

$1 \cdot w + 2 \cdot x + 4 \cdot y + 5 \cdot z$	$2 \cdot w + 3 \cdot x + 5 \cdot y + 6 \cdot z$
$4 \cdot w + 5 \cdot x + 7 \cdot y + 8 \cdot z$	$5 \cdot w + 6 \cdot x + 8 \cdot y + 9 \cdot z$

output.

"valid convolution"
↳ filter entirely overlaps the input.



$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial dout}$$

x_{11}	x_{12}	x_{13}
x_{21}	x_{22}	x_{23}
x_{31}	x_{32}	x_{33}

$\otimes$

F_{11}	F_{12}
F_{21}	F_{22}

$=$

O_{11}	O_{12}
O_{21}	O_{22}

$$O_{11} = x_{11}F_{11} + x_{12}F_{12} + x_{21}F_{21} + x_{22}F_{22}$$
$$O_{12} = x_{12}F_{11} + x_{13}F_{12} + x_{22}F_{21} + x_{23}F_{22}$$
$$O_{21} = x_{21}F_{11} + x_{22}F_{12} + x_{31}F_{21} + x_{32}F_{22}$$
$$O_{22} = x_{22}F_{11} + x_{23}F_{12} + x_{32}F_{21} + x_{33}F_{22}$$

$O \in \mathbb{R}^{2\times 2}$

$W = F \in \mathbb{R}^{2\times 2}$

$X \in \mathbb{R}^{3\times 3}$

$$\frac{\partial Z}{\partial W} \in \mathbb{R}^{2\times 2\times 2\times 2}$$

↑ tensor

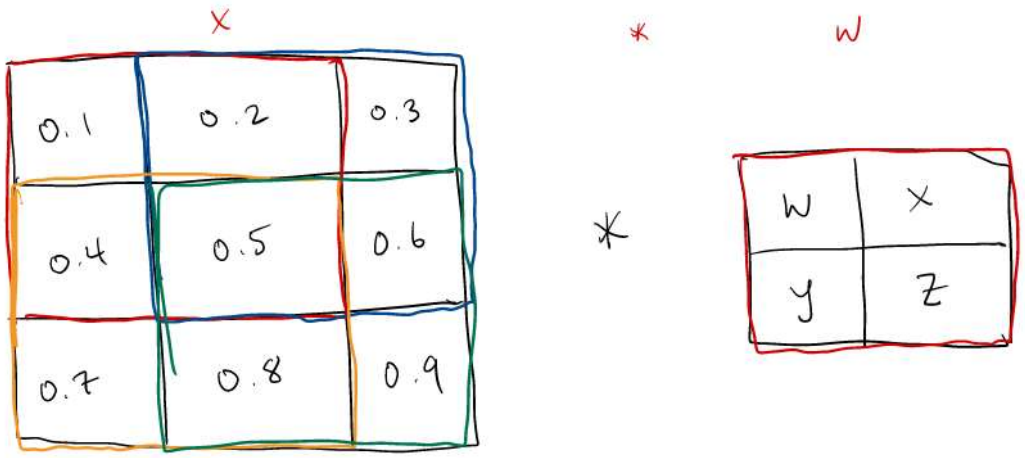
$$O_{11} = X_{11}F_{11} + X_{12}F_{12} + X_{21}F_{21} + X_{22}F_{22}$$

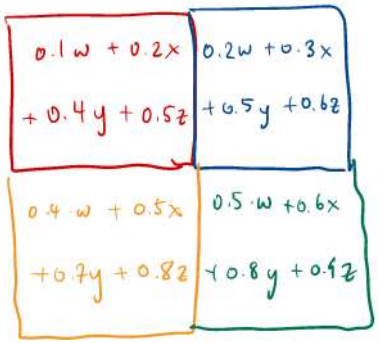
$$O_{12} = X_{12}F_{11} + X_{13}F_{12} + X_{22}F_{21} + X_{23}F_{22}$$

$$O_{21} = X_{21}F_{11} + X_{22}F_{12} + X_{31}F_{21} + X_{32}F_{22}$$

$$O_{22} = X_{22}F_{11} + X_{23}F_{12} + X_{32}F_{21} + X_{33}F_{22}$$

Convolution in 2D

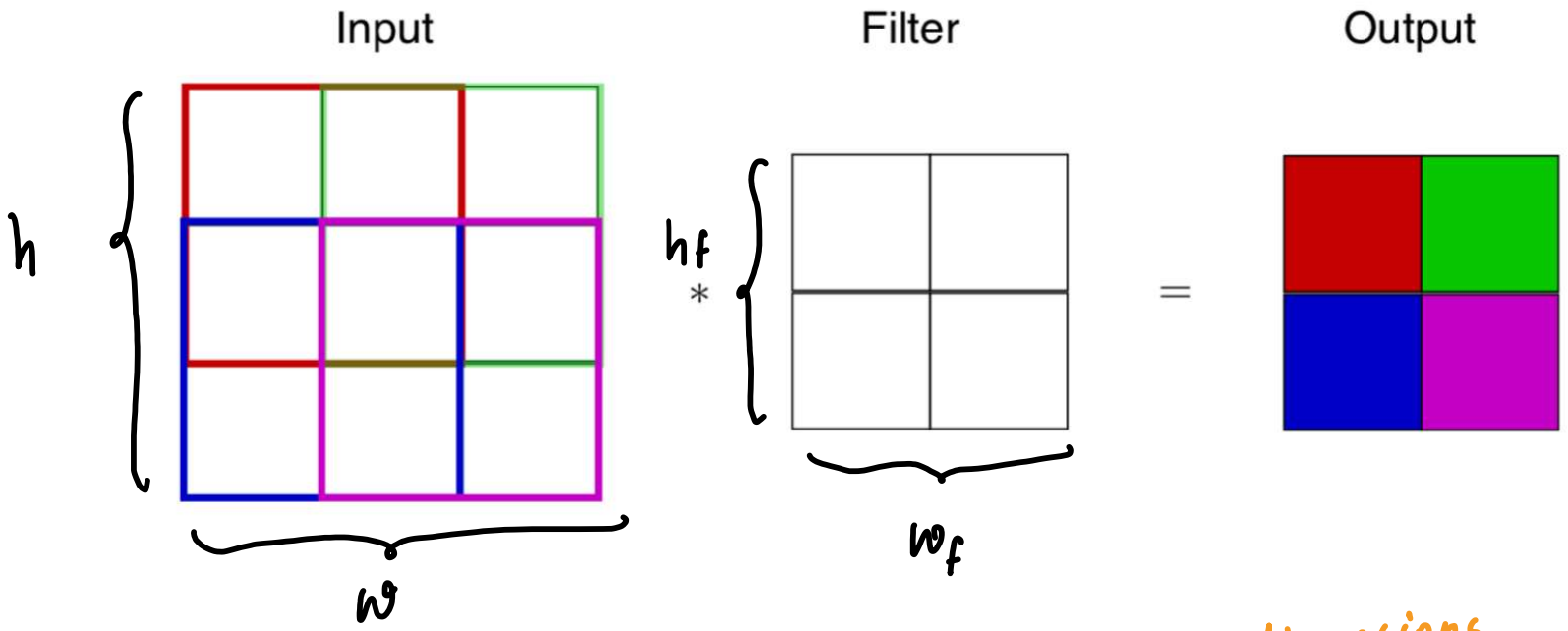




"VALID CONVOLUTION"

Convolution in 2D

***** All convolutions in this class will be VALID convolutions (the filter and inputs must be totally overlapping). *****



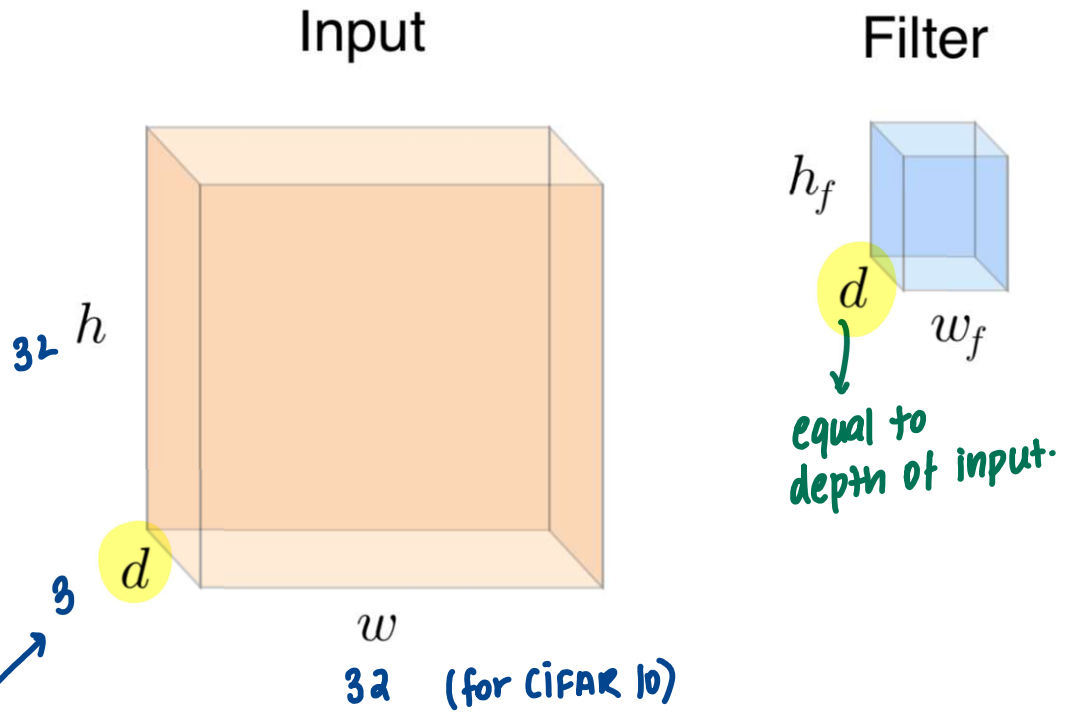
$$(w - w_f + 1, h - h_f + 1)$$ dimensions of output.

The convolutional layer of a CNN

Convolutional layer

This convolution operation (typically in 3D, since images often come with a width and height as well as *depth* for the R, G, B channels) defines the "convolutional layer" of a CNN. The convolutional layer defines a collection of filters (or activation maps), each with the same ~~dimension~~ depth as the input.

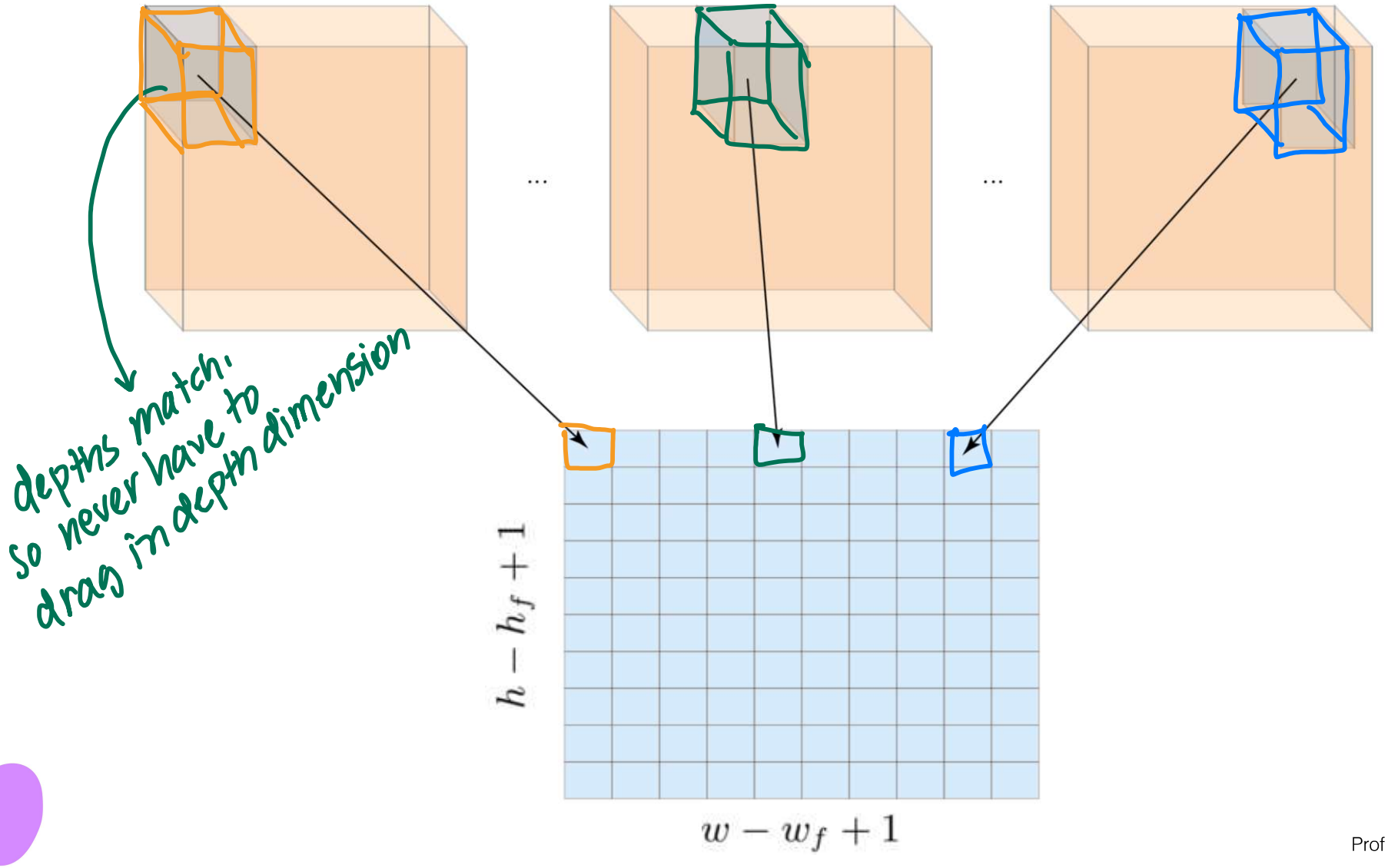
- Say the input was of dimensionality (w, h, d).
- Say the filter dimensionality is (w_f, h_f, d). So that the filter operates on a small region of the input, typically $w_f < w$.
- The depths being equal means that the output of this convolution operation is 2D.



The convolutional layer of a CNN

Convolutional layer (cont.)

After performing the convolution, the output is $(w - w_f + 1, h - h_f + 1)$

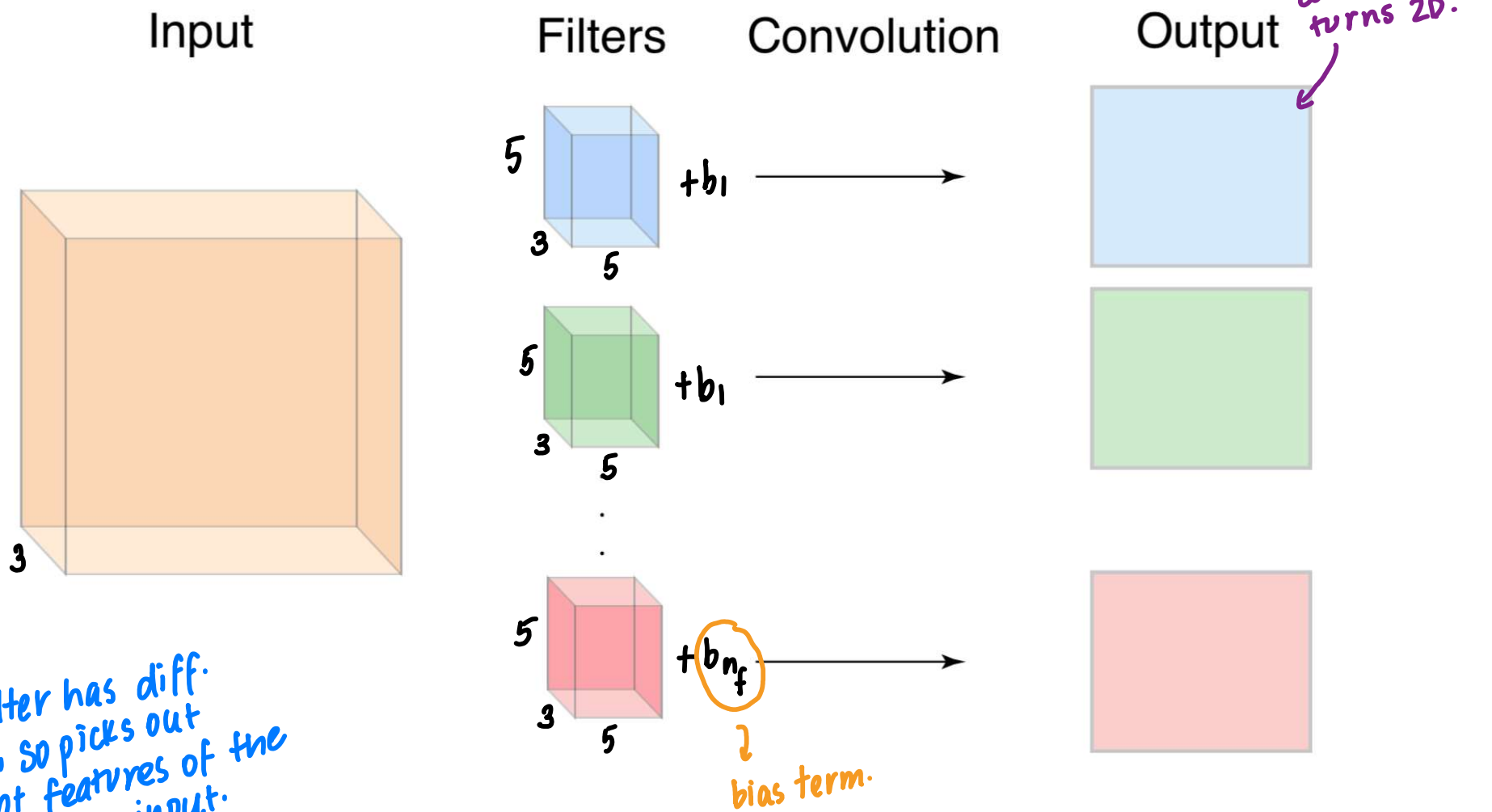


The convolutional layer of a CNN

Convolutional layer (cont.)

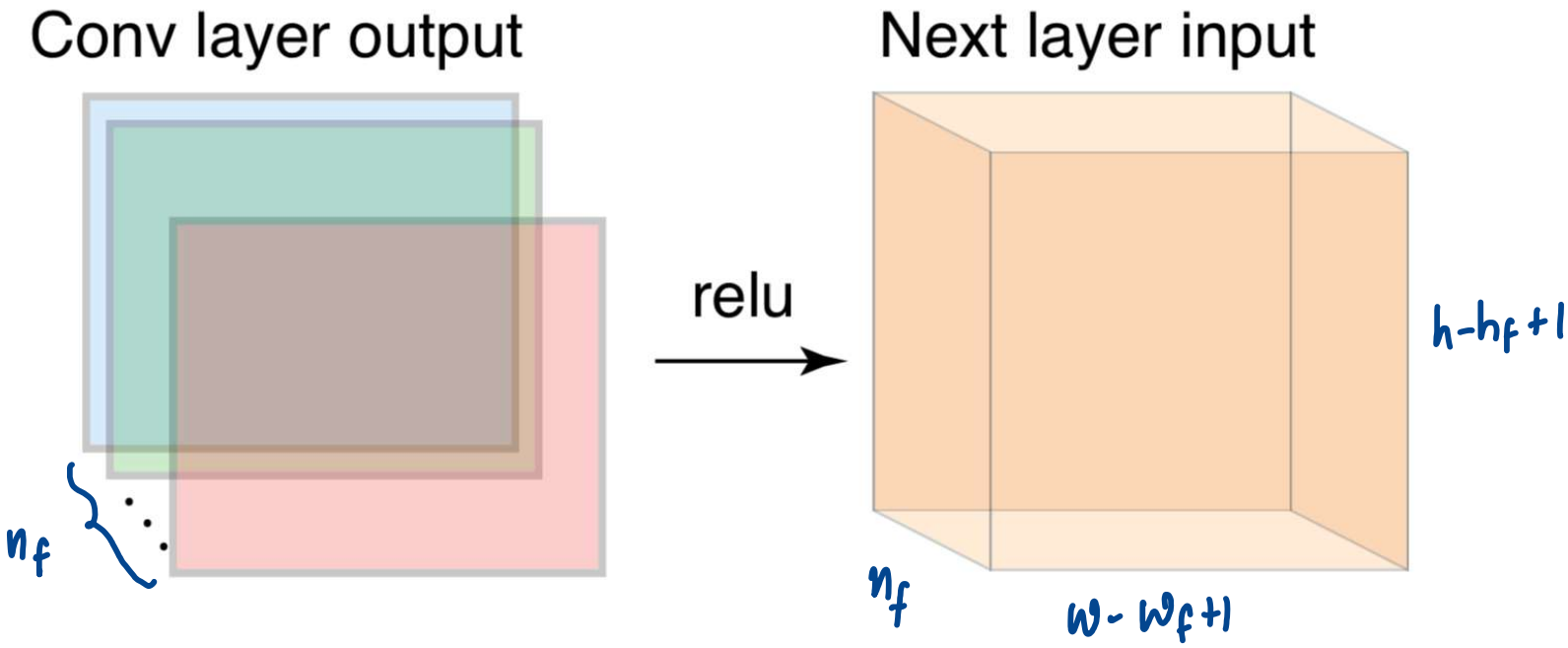
filters = n_f

Now, we don't have just one filter in a convolutional layer, but multiple filters.
We call each output (matrix) a slice.



Convolutional layer (cont.)

The output slices of the convolution operations with each filter are composed together to form a $(w - w_f + 1, h - h_f + 1, n_f)$ tensor, where n_f is the number of filters. The output is then passed through an activation nonlinearity, such as $\mathrm{ReLU}(\cdot)$. This then acts as the input to the next layer.

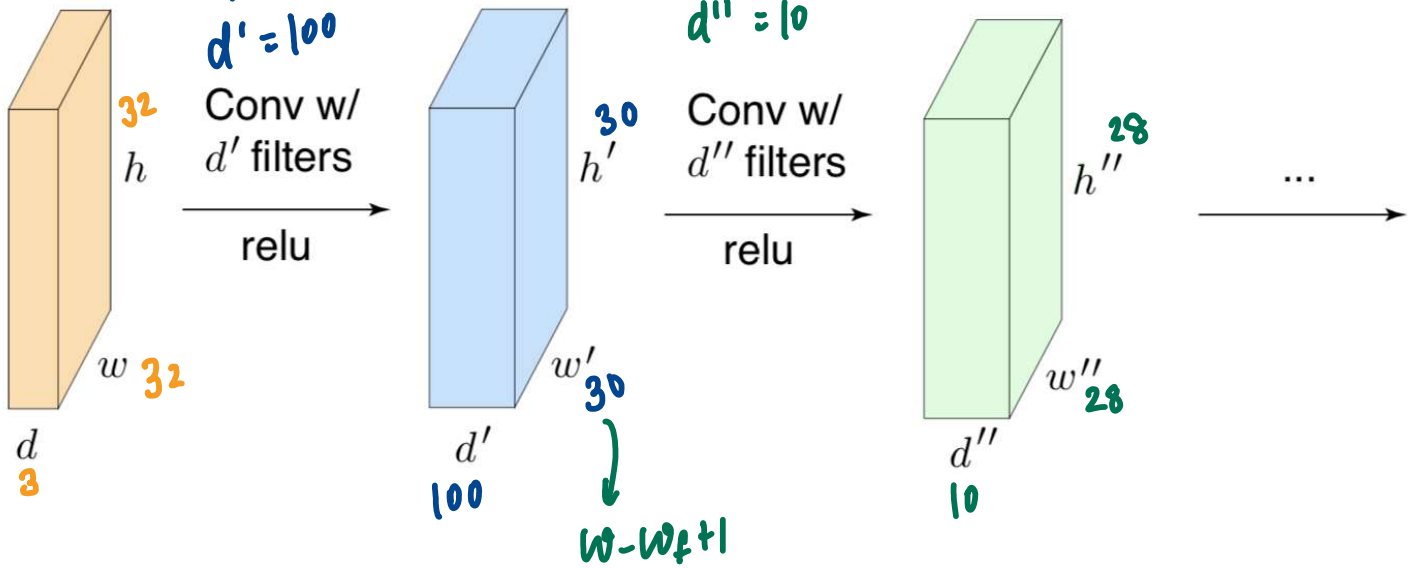


OH: output dim with tensor?

The convolutional layer of a CNN

Composition of convolutional layers

These layers can then be composed, which comprise a large component of the CNN.



d' = #filters.

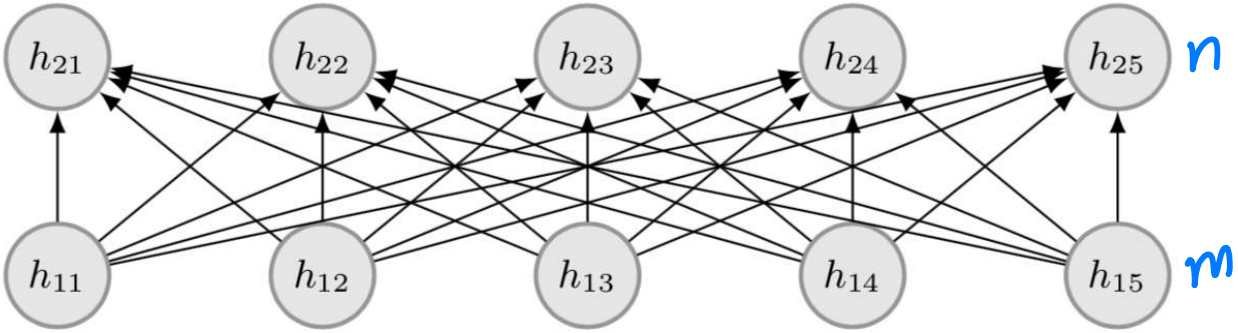
Convolutional layers have sparse interactions

Convolutional layers have *sparse interactions* or *sparse connectivity*. The idea of sparse connectivity is illustrated below, where each output is connected to a small number of inputs.

Fully connected layers:

Layer 2

Layer 1



n

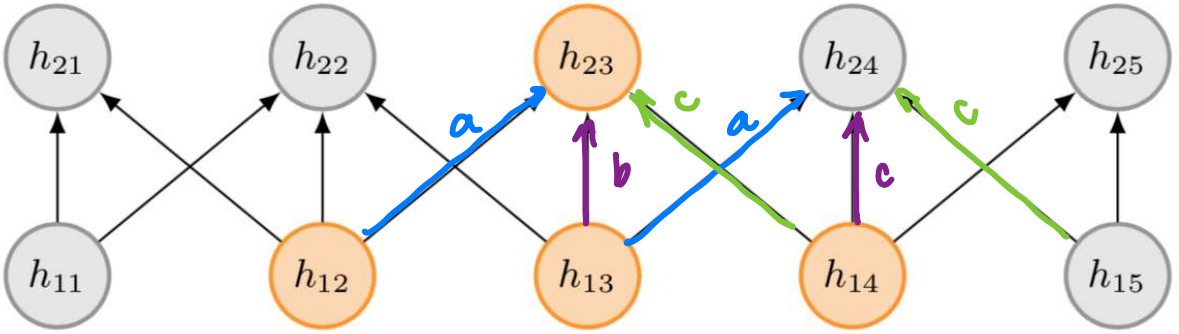
m

COMP - 3:19 PM

Sparsely connected layers:

Layer 2

Layer 1



Regularization of the structure of the NN

Convolutional layers have sparse interactions (cont.)

Sparse interactions aid with computation.

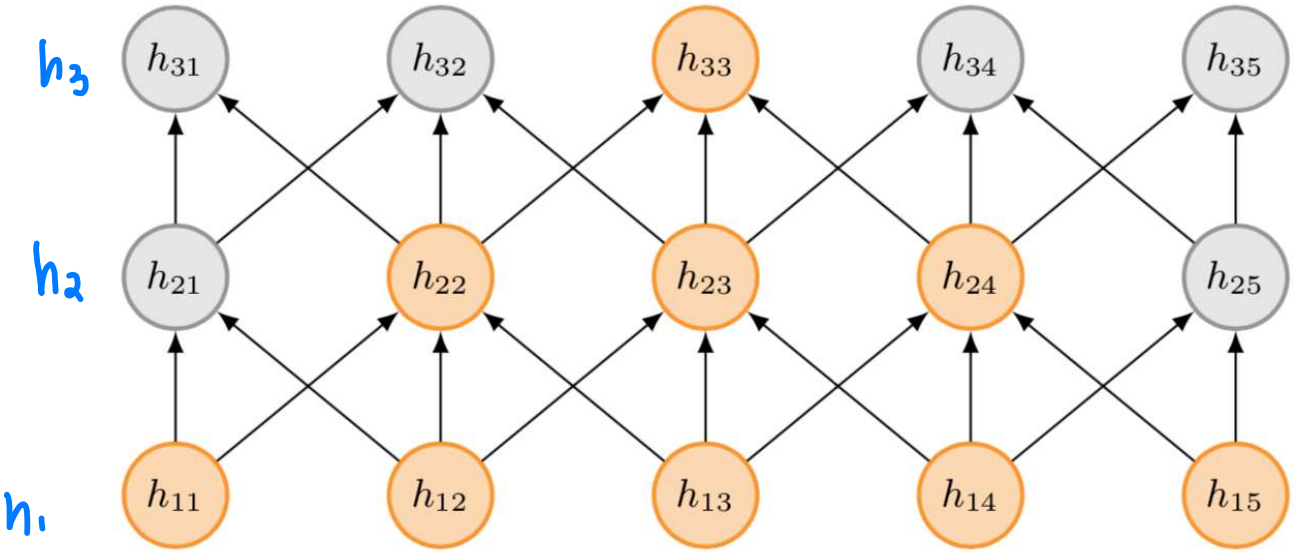
- **Sparse interactions reduce computational memory.** In a fully connected layer, each neuron has $w \cdot h \cdot d$ weights corresponding to the input. In a convolutional layer, each neuron has $w_f \cdot h_f \cdot d$ weights corresponding to the input. Moreover, in a convolutional layer, every output neuron in a slice has the *same* $w_f \cdot h_f \cdot d$ weights (more on this later). As there are far fewer parameters, this reduces the memory requirements of the model.
- **Sparse interactions reduce computation time.** If there are m inputs and n outputs in a hidden layer, a fully connected layer would require $\mathcal{O}(mn)$ operations to compute the output. If each output is connected to only k inputs (e.g., where $k = w_f \cdot h_f \cdot d$) then the layer would require $\mathcal{O}(kn)$ operations to compute the output.

cons: self driving car application (want model to see pedestrian on left and car on right).

Convolutional layers have sparse interactions (cont.)

A concern of sparse interactions is that information information from different parts of the input may not interact. For example, a self-driving car should know where obstacles are from all over the image to avoid them.

This argues that networks should be composed of more layers, since units in deeper layers indirectly interact with larger portions of the input.



"receptive field"

Convolutional layers share parameters

Convolutional layers have shared parameters (or *tied weights*), in every output neuron in a given slice uses the same set of parameters in the filter.

Example: Consider an input that is $(32 \times 32 \times 3)$. We have two architectures; in the fully connected architecture, there are 500 output neurons at the first layer. In the convolutional neural net there are 4 filters that are all $4 \times 4 \times 3$. (1) How many output neurons are there in the convolutional neural network, assuming that the convolution is only applied in regions where the filter fully overlaps the kernel? (2) How many parameters are in each model?

(1) input = (32x32x32) convolved with filters (4x4x3)

(w-wf+1, h-hf+1) ⟶ output of 1 filter (29,29)

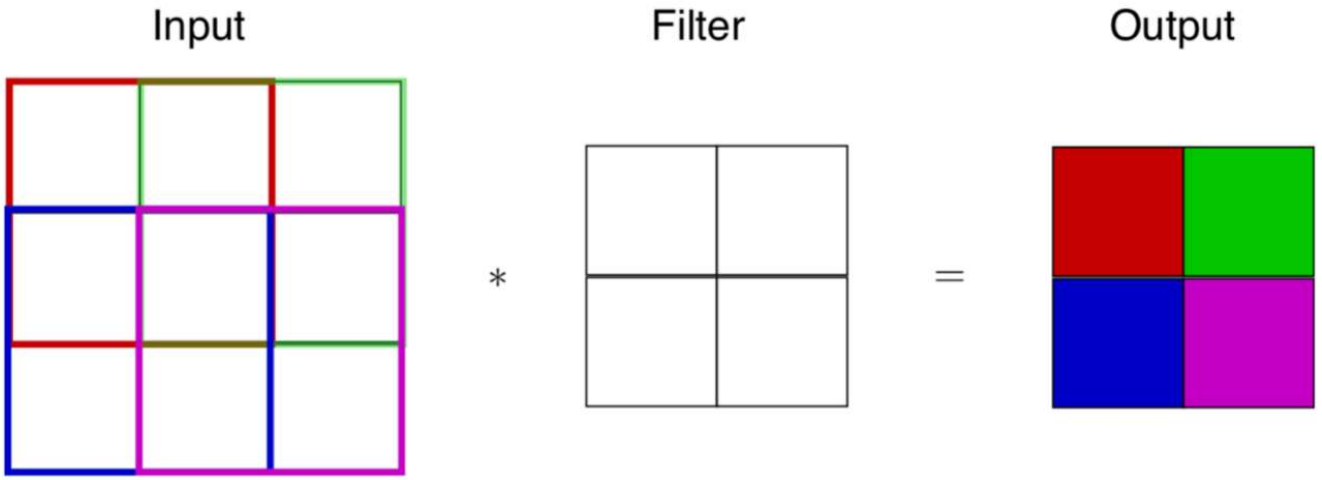
#neurons = 29x29x4

(2) FC: (32x32x3 +1) x 500 = 1.5 million parameters

conv: (4x4x3+1) x4 = 1960 0 params

All convolutions in this class are valid convolutions

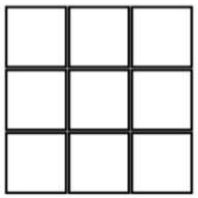
In this class, all convolutions will be valid convolutions. We explicitly specify the amount of zero padding when we need it.



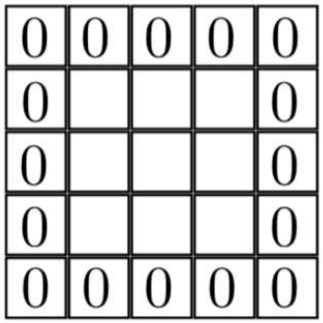
Output size: $(w - w_f + 1, h - h_f + 1)$

Convolutional padding

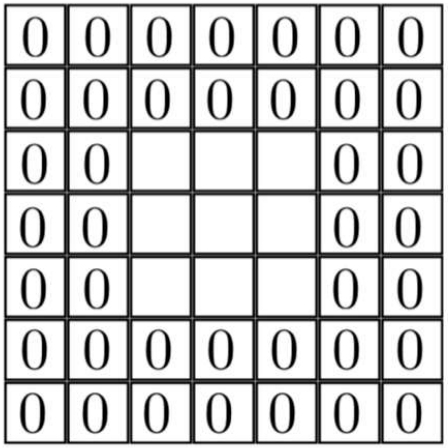
$\text{pad} = 0$



$\text{pad} = 1$



$\text{pad} = 2$



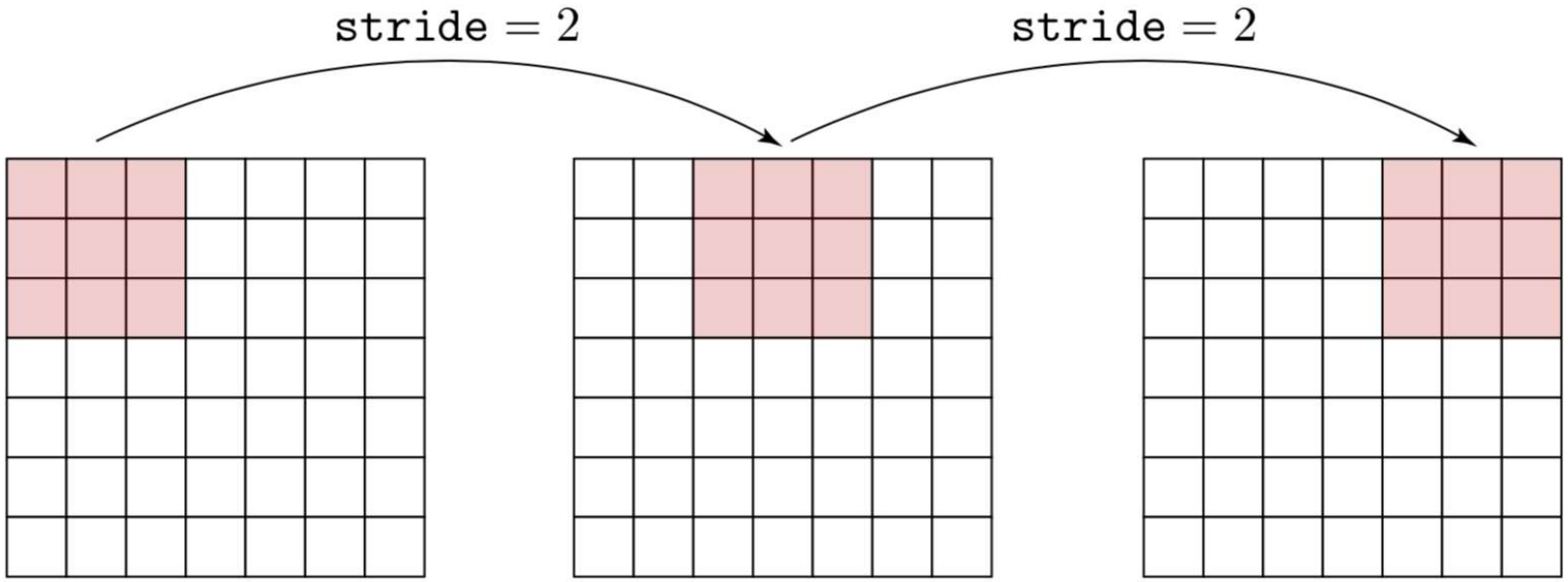
The output of the convolution is now $(w - w_f + 1 + 2\text{pad}, h - h_f + 1 + 2\text{pad})$.

Convolution padding (cont.)

It is worth noting that Goodfellow et al. report that the optimal amount of zero padding (in terms of test accuracy) is somewhere between $\text{pad} = 0$ and the pad that causes the output and input to have the same width and height.

Convolution stride

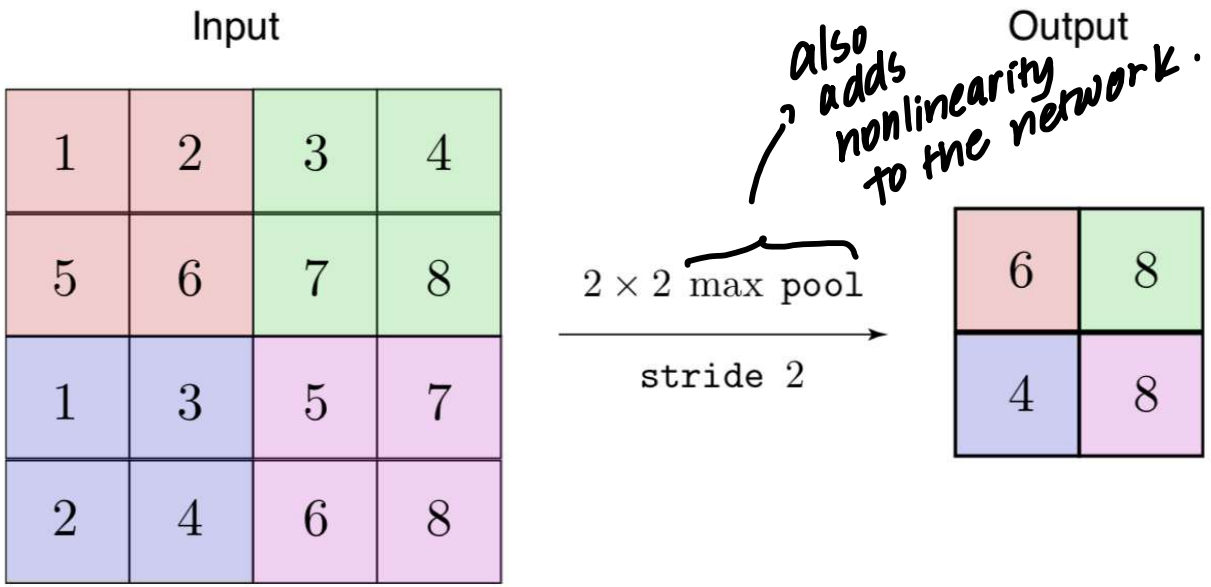
Another variable to control is the stride, which defines how much the filter moves in the convolution. For normal convolution, $\texttt{stride} = 1$, which denotes that the filter is dragged across every part of the input. Consider a 7×7 input with a 3×3 filter. If convolution is only applied where the kernel overlaps the input, the output is 5×5. With $\texttt{stride} = 2$, the output is 3×3.



Pooling layer

CNNs also incorporate pooling layers, where an operation is applied to all elements within the filtering extent. This corresponds, effectively, to downsampling.

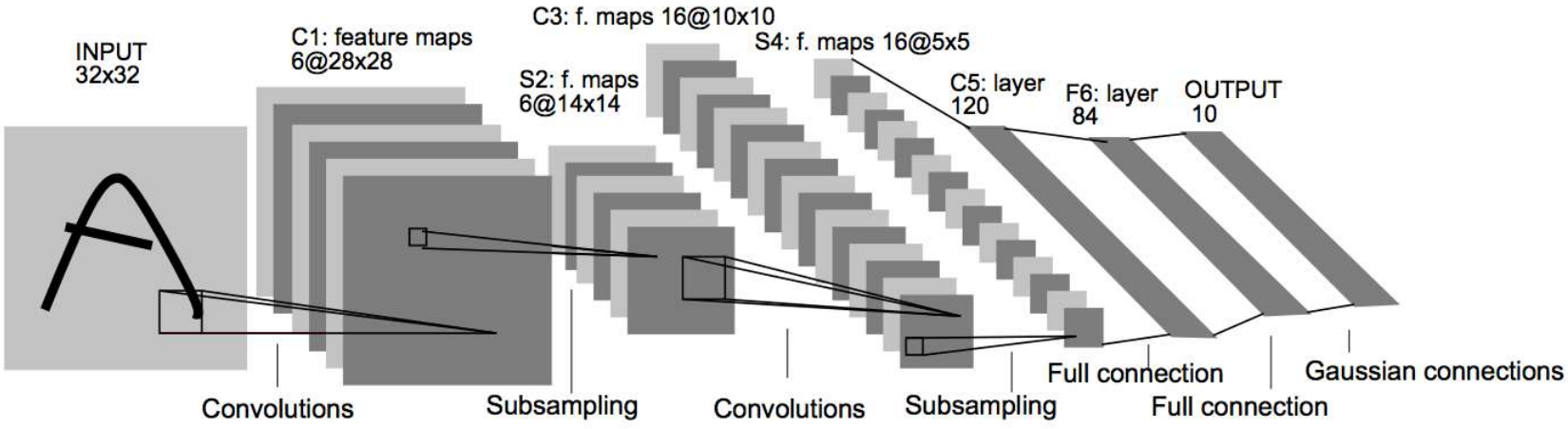
The pooling filter has width and height (w_p, h_p) and is applied with a given stride. It is most common to use the $\max()$ operation as the pooling operation.



also adds nonlinearity to the network.

MIDTERM.

Sizing examples



C1 contains six 5x5 conv filters.
Size of feature maps at C1?
Number of parameters in C1 layer?

S2 is a 2x2 pooling layer applied at stride 2.
Size of feature maps at S2?
Number of parameters in S2 layer?

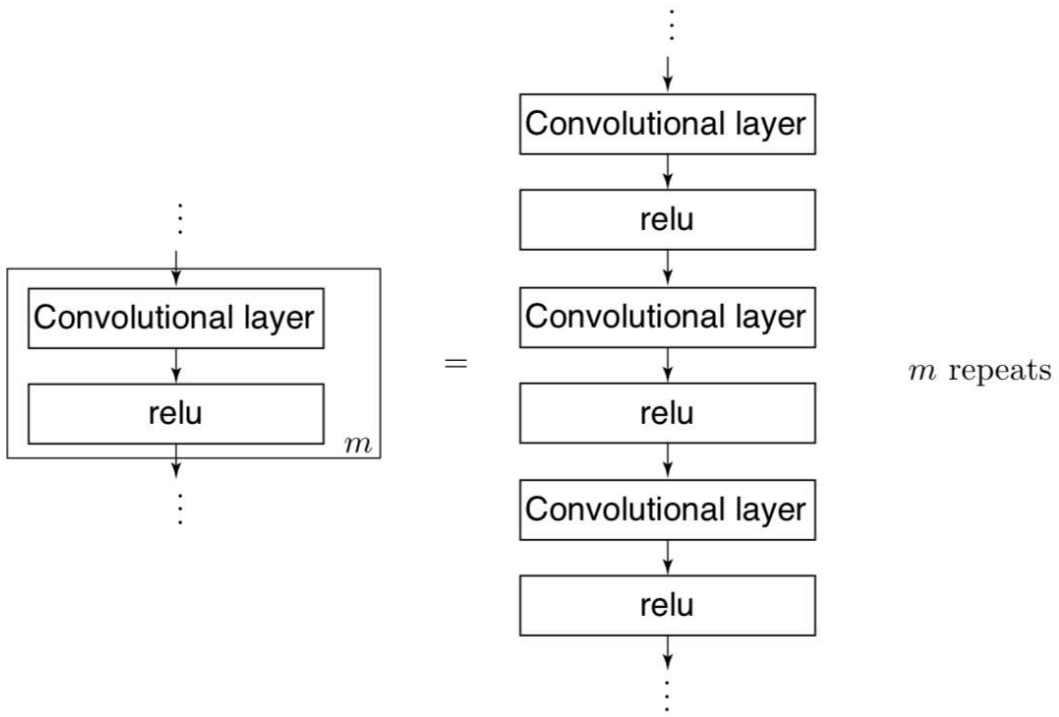
C3 contains sixteen 5x5 conv filters.
Number of parameters in C3 layer?
Size of feature maps at C3?

Convolutional neural network architectures

Typically, convolutional neural networks are comprised of units that are composed of:

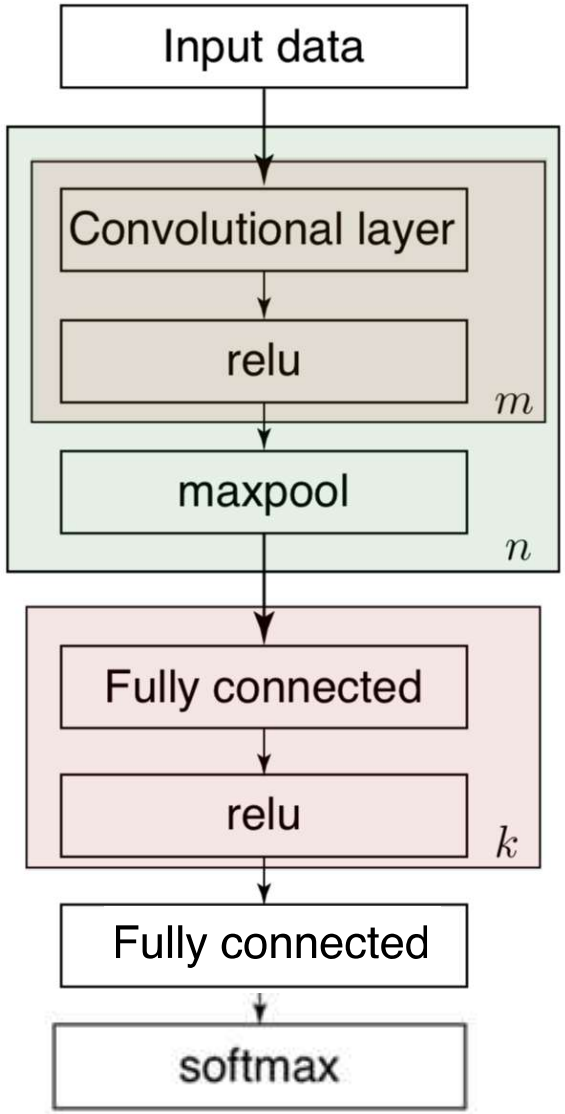
- Several paired convolutional-relu layers.
- Potentially one max pooling layer.

These units are cascaded. Finally, a CNN may end with a fully connected-relu layer, followed by a softmax classifier. To illustrate this, we borrow the plate notation from graphical models, where:



Convolutional neural network architectures (cont.)

With this in mind, the following describes a fairly typical CNN architecture.



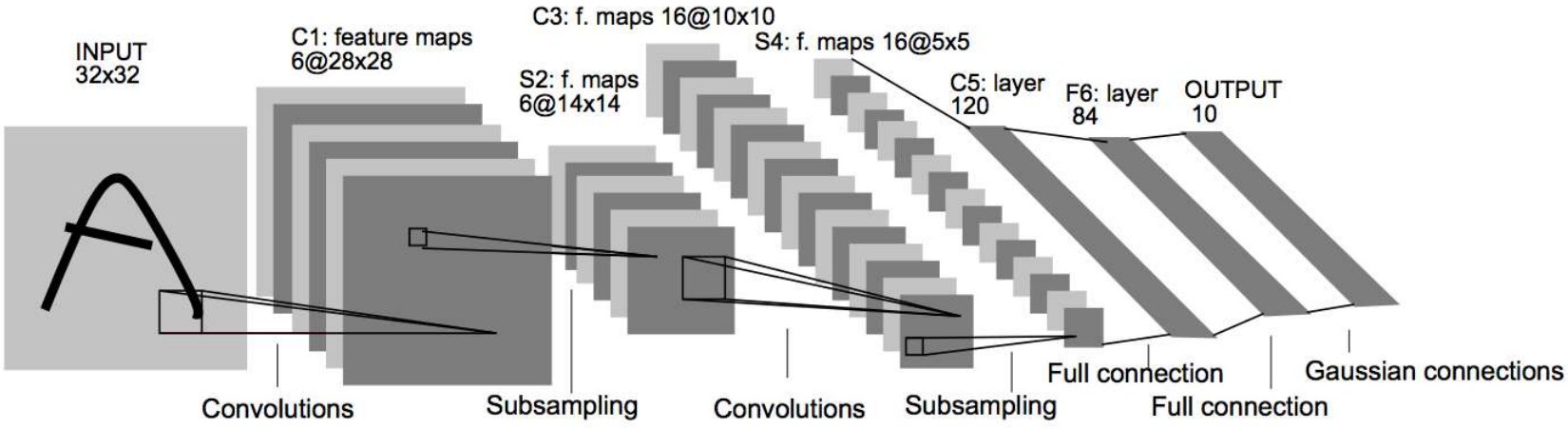
Case studies

To help get an intuition behind CNN's, we'll go over a few architectures that have been influential in recent years.

Case studies:

- LeNet (1998)
- AlexNet (2012)
- VGG (2013)
- GoogLeNet (2014)
- ResNet (2015)

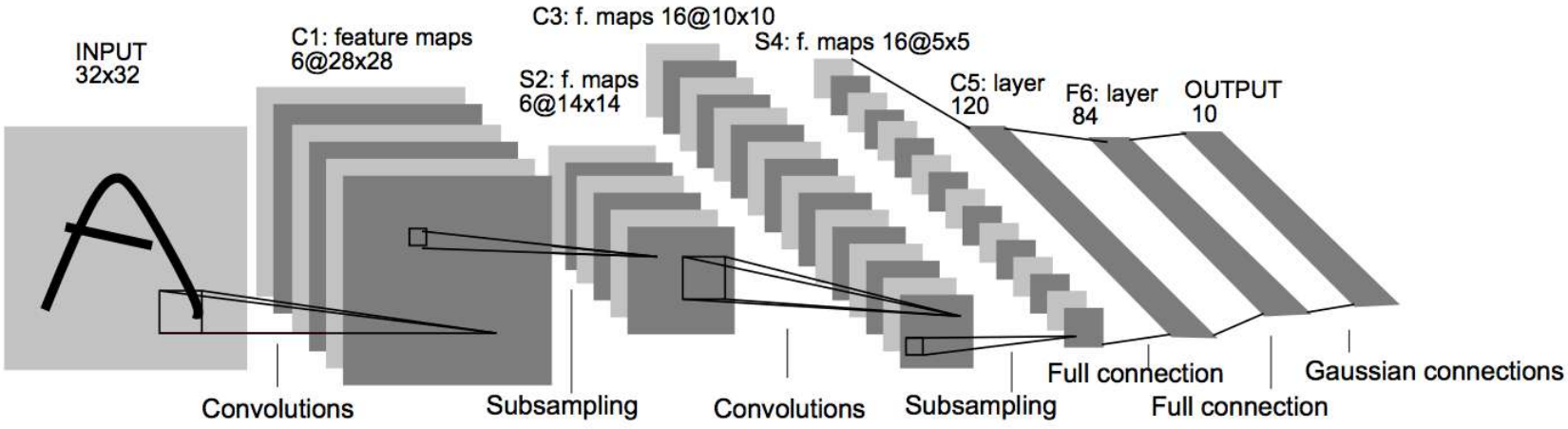
LeNet-5



LeCun et al., 1998.

Applied to handwriting recognition.

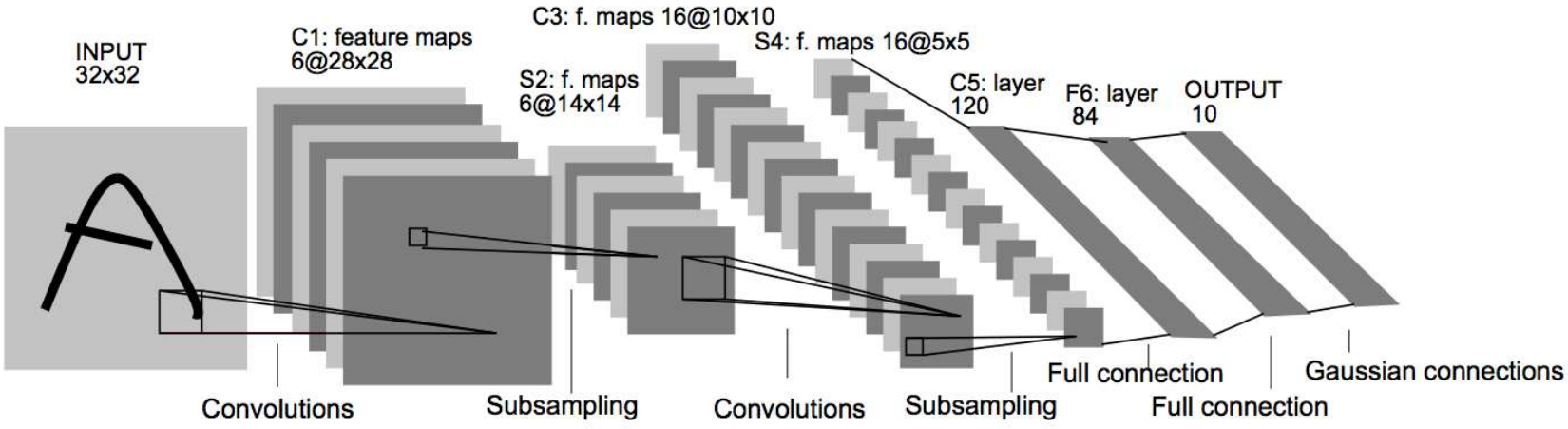
LeNet-5



LeCun et al., 1998.

Question: How many connections are there in the first convolutional layer?

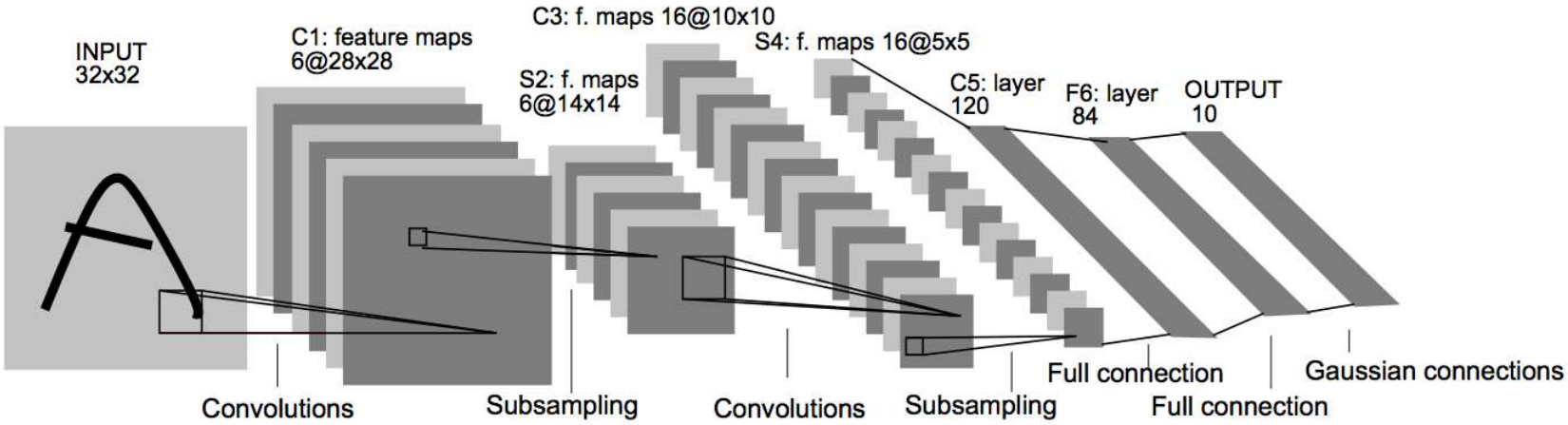
LeNet-5



4 total layers. Input is 32x32.

1. [28x28x6] **CONV**: 6 convolutional filters, 5x5 filter size, applied at stride 1.
2. [14x14x6] **POOL**: 2x2 pool with stride 2. (Adds all elems, multiplies them by trainable coefficient, then passes through sigmoid.)
3. [10x10x16] **CONV:** 16 convolutional filters, 5x5.
4. [5x5x16] **POOL**: 2x2 pool with stride 2.
5. [120] **CONV**: 120 5x5 convolutional filters.
6. [84] **FC**: FC layer: 84 x 120.
7. [10] **OUT:** MSE against a template for each digit.

LeNet-5



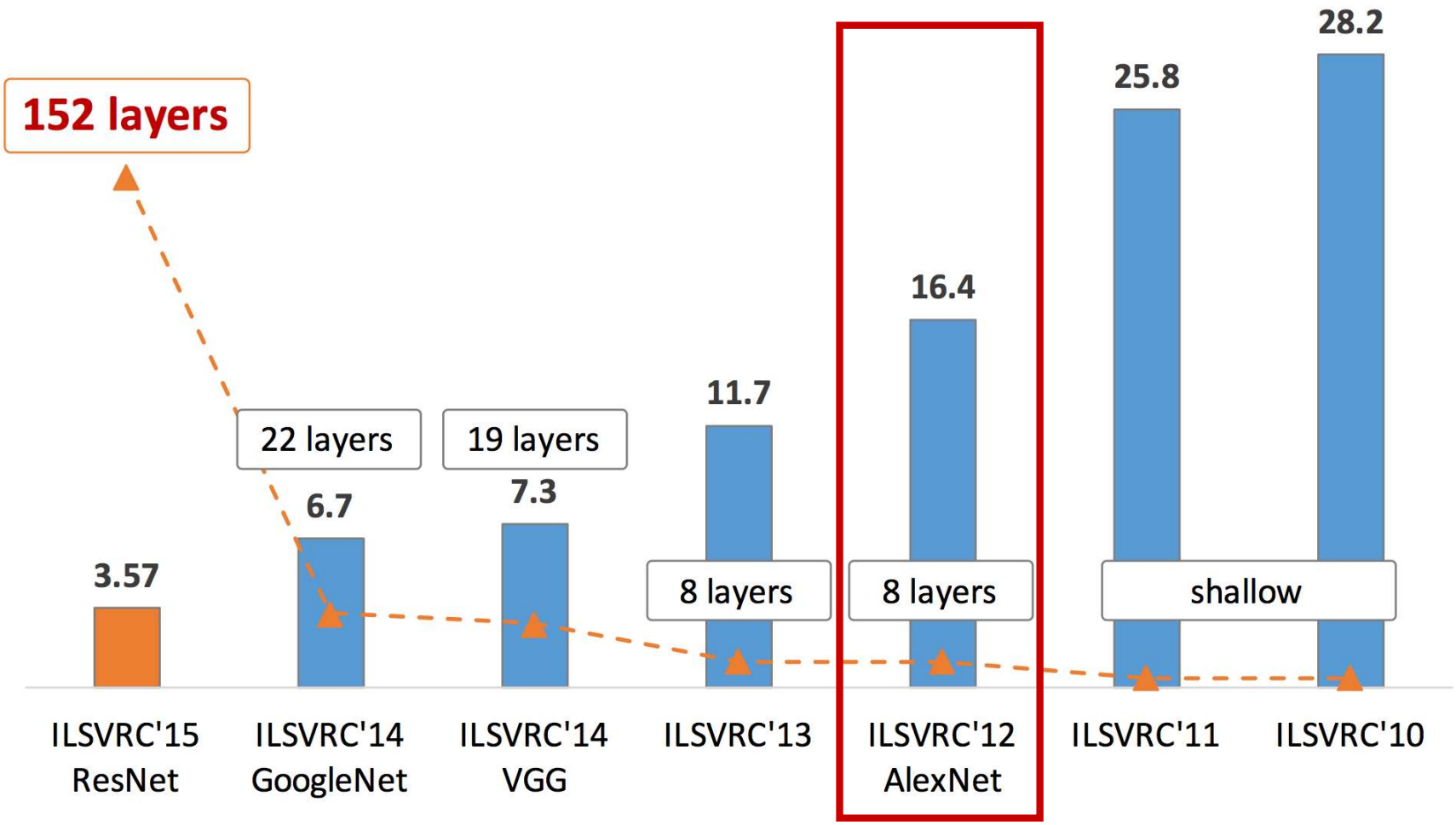
LeCun et al., 1998.

Overall architecture:

[CONV-POOL]x2 - CONV - FC - OUT

AlexNet in context

The number of layers refers to the number of convolutional or FC layers.



http://kaiminghe.com/icml16tutorial/icml2016_tutorial_deep_residual_networks_kaiminghe.pdf