Deep Convolutional Neural Network

Jerry Xing

2023/04/19

Today's Lecture in Ten Seconds

GO

DEEPER

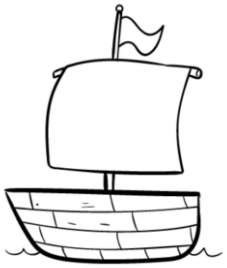
LARGER

SMARTER

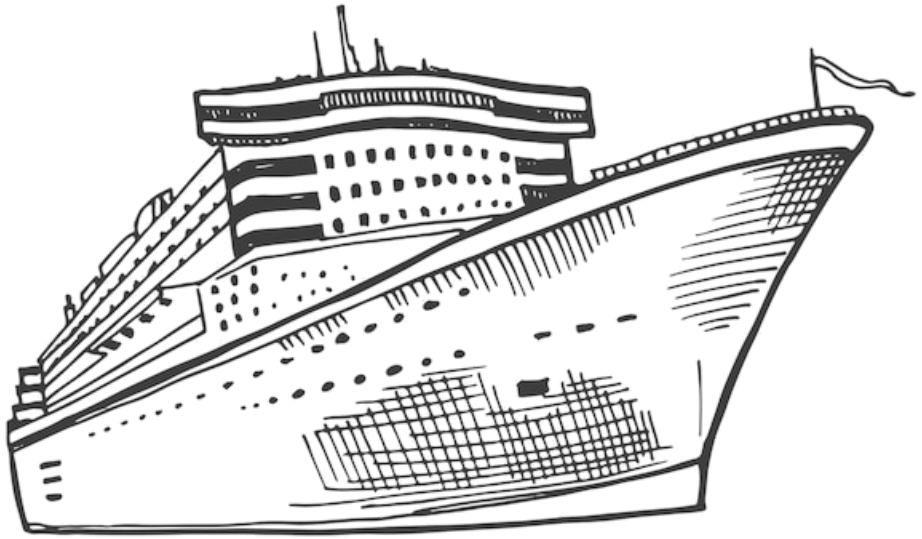
Today's Lecture in One Minute

- Deep neural network = great approximator
  - Complex and efficient
- Fully connected network (FCN)
  - A simple neural network structure
  - Stack of perceptrons
- Convolutional neural network (CNN)
  - Stack of "local perceptrons"
  - More suited for images
- Other common neural network components
  - Pooling

The Content of This Lecture is Like



Models we'll study today



Models in real world

Different Power
Shared Design

Computation: Input -> Output

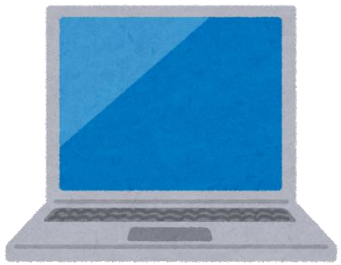
Input:
image

Output:
objects in the image

Computation

dog, laptop, table

Human with related knowledge



Program
or, a function f_T

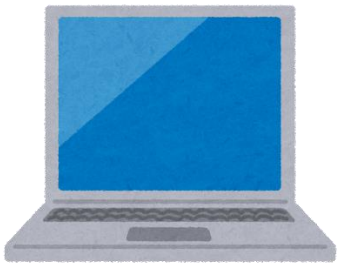
Computation: Input ->f_T-> Output

Input:
image

$\longrightarrow$ f_T $\longrightarrow$

Output:
objects in the image

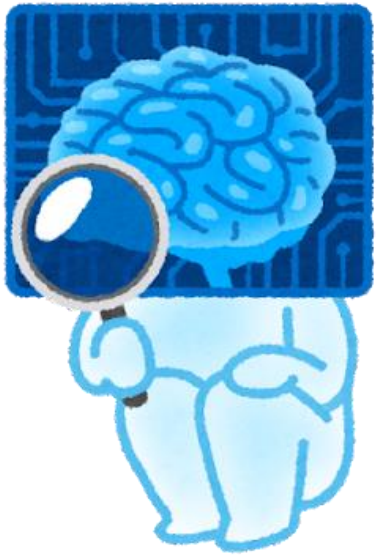
dog, laptop, table



Getting f_T: coding or searching

- How to we get the desired function f_T?

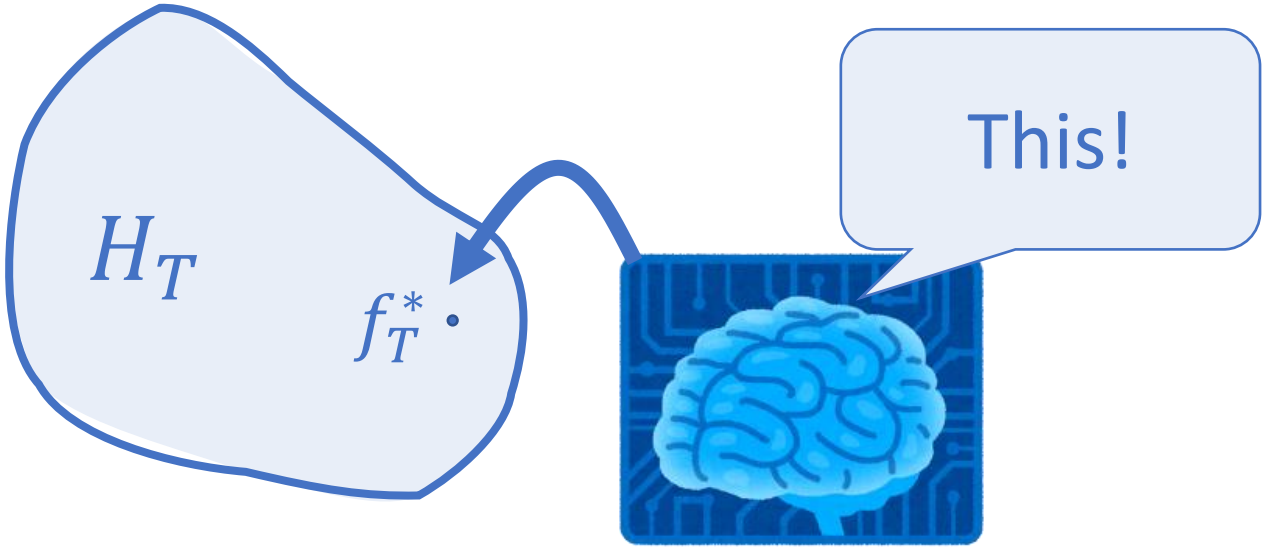
Manual coding



Automatic Searching

Searching for f_T

- Searching range H_T: $f_T \in H_T$
  - E.g., f should have the form $f_T(x) = a_0 + a_1x + a_2x^2 + \cdots$ and H_T the set of all polynomials
- Badness metric L
  - If $L(f_{T1}) < L(f_{T2})$, then we know f_1 is better
- Task: find the $f_T^* \in H_T$ that gives us the minimal L



Approximated instead of exact f_T

- But wait, how do we know H_T?
- f_T can have an extremely complex form when the task is hard!
- We search for the approximation $f \approx f_T$ instead!
- Often, all we need is a powerful model, and we don't care whether f is really the "ground truth" f_T
- Example: polynomial approximation

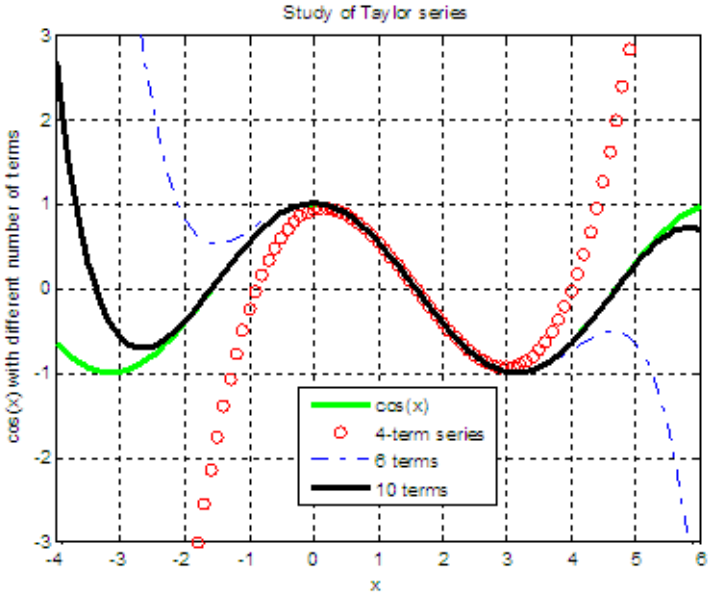
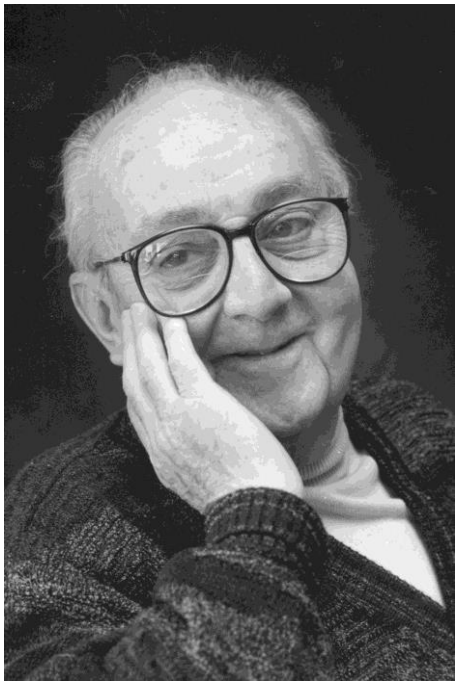


Image credit: https://www.quora.com/Its-well-known-that-e-x-sin-x-and-cos-x-can-be-represented-by-their-infinite-Taylor-series-Is-there-a-geometric-interpretation-to-these-series-I

Approximated instead of exact f_T

- But wait, how do we know H_T?
- f_T can have an extremely complex form when the task is hard!
- We search for the approximation $f \approx f_T$ instead!
- Often, all we need is a powerful model, and we don't care whether f is really the f_T
- Example: polynomial approximation



> *All models are wrong, but some are useful*
>
> ——*George Box*

Approximated instead of exact f_T

- But wait, how do we know H_T?
- f_T can have an extremely complex form when the task is hard!
- We search for the approximation $f \approx f_T$ instead!
- Often, all we need is a powerful model, and we don't care whether f is really the f_T
- Example: polynomial approximation

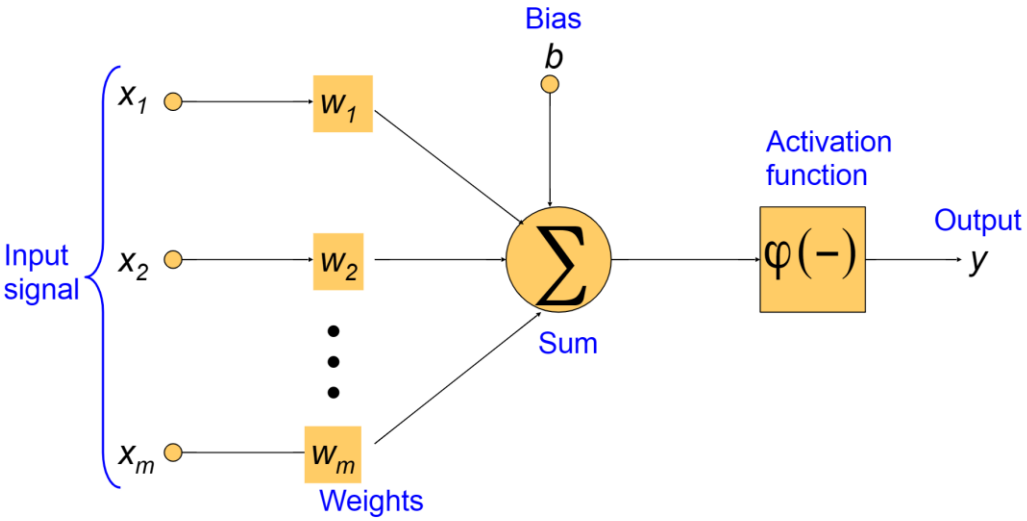
> *if it looks like a duck, swims like a duck, and quacks like a duck, then it probably is a duck*

What makes a good H?

- Let the approximation $f \in H$
- What makes a good H?
  - Complexity
    - $f \in H$ should be complex enough to approximate f_T
  - Efficiency
    - f^* can be found (learned) efficiently
- Current most popular choice: $H = \{\text{Deep nerual networks}\}$
- Why?

Simple + Composition = Complex = Powerful

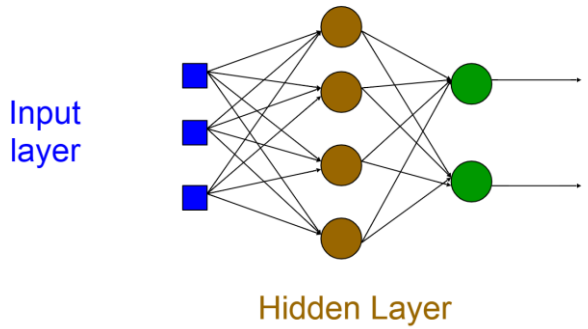
- Recap: perceptron = linear transformation + activation function



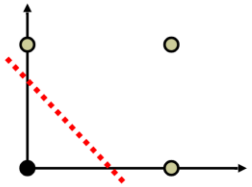
$$f(x) = \varphi\left(b + \sum_{i=1}^{m} w_i \cdot x_i\right)$$

Simple + Composition = Complex = Powerful

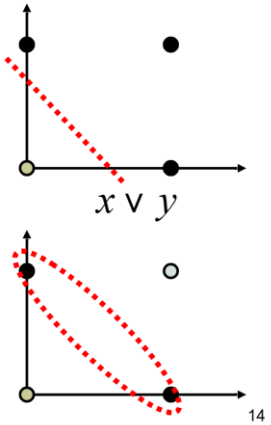
- Recap: multi-layer perceptron = composition of two perceptrons
  - Solve the problem that a single perceptron cannot solve
  - Idea: composition -> complexity?



- Linear separable:



- Linear inseparable:
- Solution?



$$f(x) = f_2 \circ f_1(x)$$
$$= f_2\big(f_1(x)\big)$$

where f_1 and f_2 are two perceptrons

Simple + Composition = Complex = Powerful

- Inspiration: build a more complex and powerful function by composing simple functions
- And we can continue composing!

$$f(x) = \cdots \circ f_{100} \circ \cdots \circ f_2 \circ f_1(x)$$

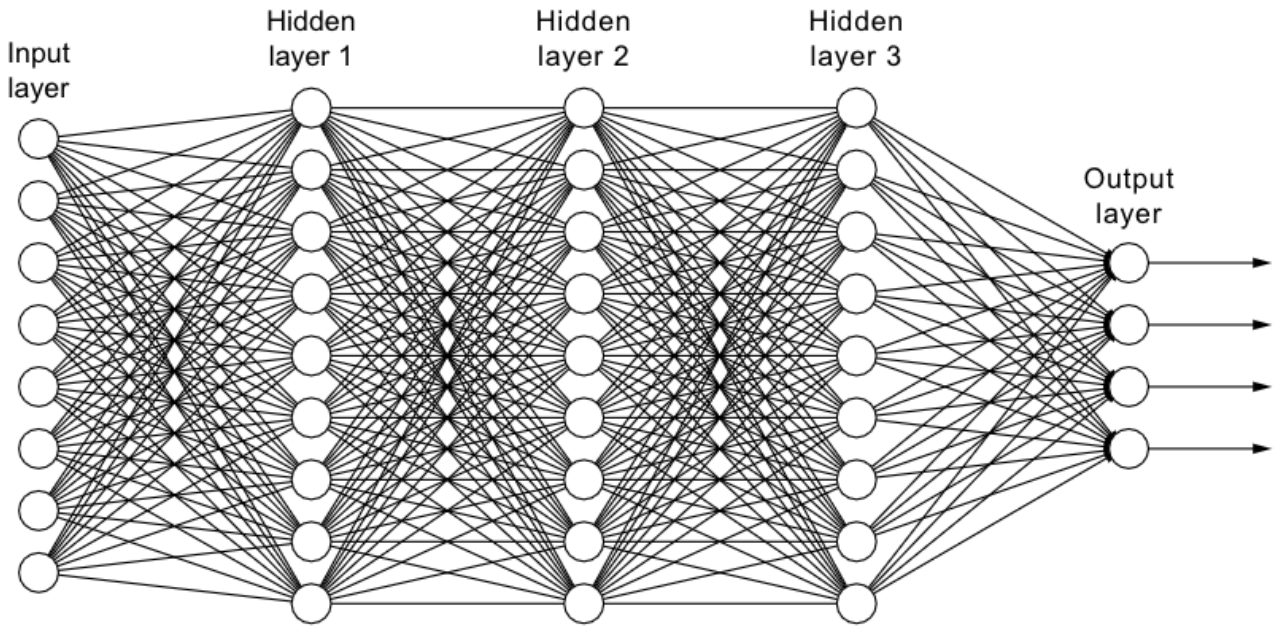


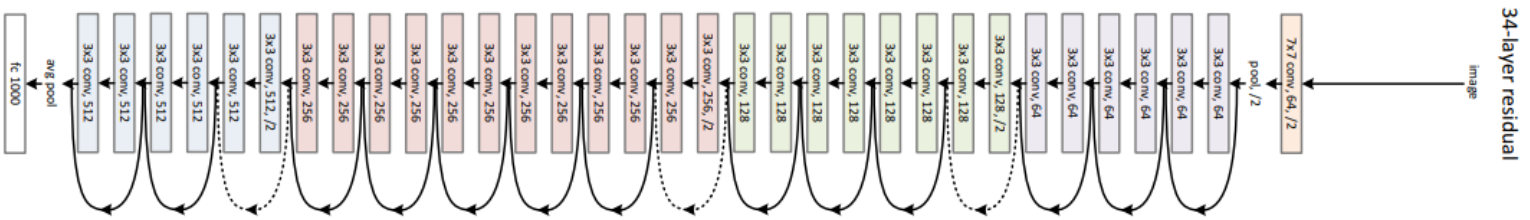
Image credit: https://freecontent.manning.com/neural-network-architectures/

Simple + Composition = Complex = Powerful

- Inspiration: build a more complex and powerful function by composing simple functions
- And we can continue composing!

$$f(x) = \cdots \circ f_{100} \circ \cdots \circ f_2 \circ f_1(x)$$

- Example: ResNet



(a shallow version that has only 34 layers)

Simple + Composition = Complex = Powerful

- Inspiration: build a more complex and powerful function by composing simple functions
- And we can continue composing!

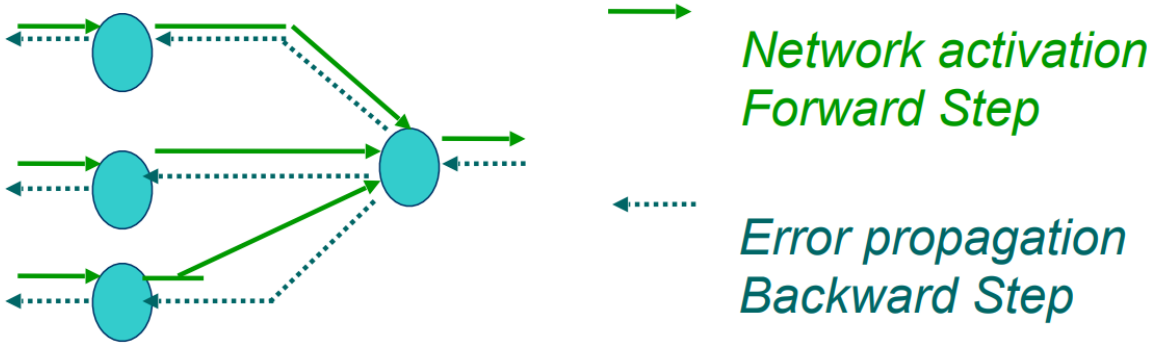
$$f(x) = \cdots \circ f_{100} \circ \cdots \circ f_2 \circ f_1(x)$$

- Example: ResNet
- Analogy: telephone game

Image credit: https://blog.prototypr.io/winning-the-telephone-game-5bf5457e16b4

Backpropagation: train NN efficiently

- In general, computing the gradient for such giant model is nothing easy!
- Backpropagation gives us an efficient way
- So widely used that it's hard to realize its importance



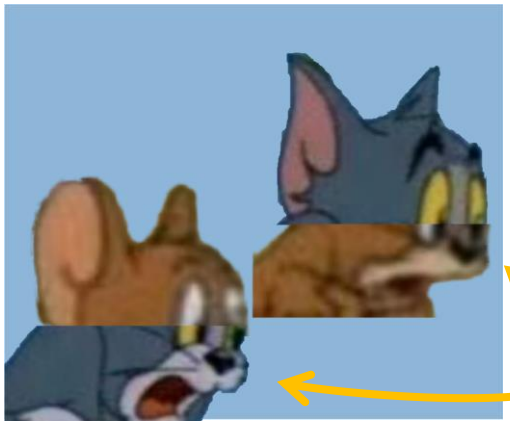
Quick recap: deep neural network

- Solving problem by searching for a good approximator
- Deep neural network = great approximator
  - Complex enough
  - Efficient training by backpropagation
- Fully connected network (FCN)
  - A simple neural network structure
  - Stack of perceptrons

FCN for images? Probably no

- FCN does NOT care the order of input features
- Reasonable for some tasks
  - E.g., in the house price prediction task, organizing data like (area, location, age) or (area, age, location) doesn't make any difference
- Not true for image-related tasks!
  - The pixel neighbor information matters!

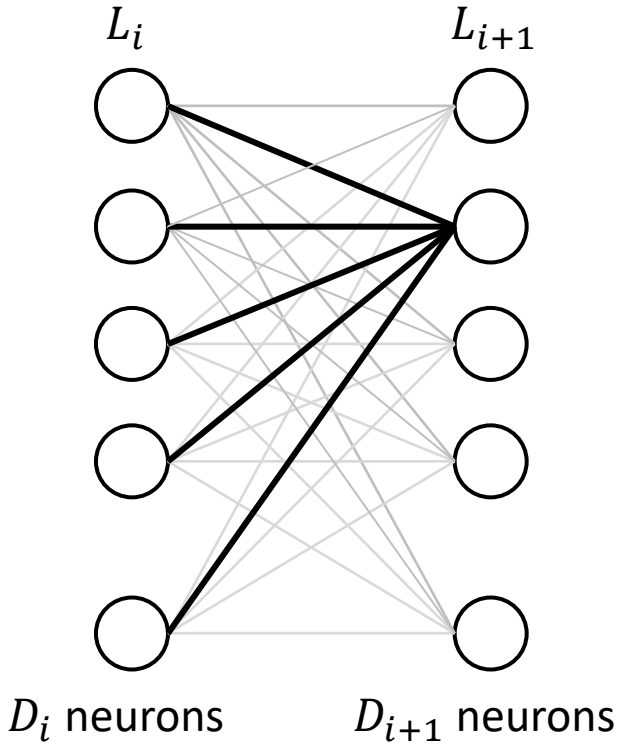
Content: cat and mouse



Content: ???

FCN for images? Probably no

- FCN is computationally expensive
- lots of weight parameters to train
- Extremely unfriendly for high-dimensional data like images



# of weight parameters:

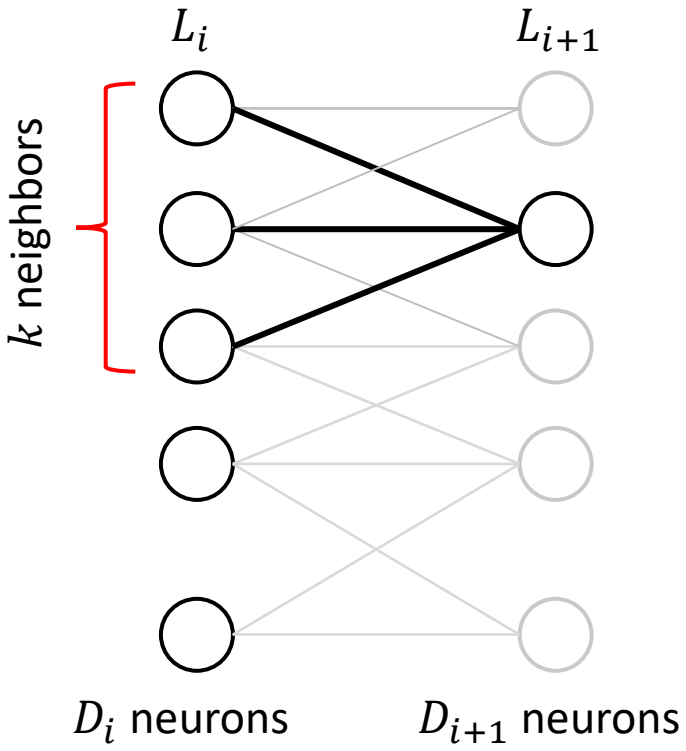
$$D_i \times D_{i+1}$$

For two 256x256 images:

$$D_i \times D_{i+1} = 256^2 \times 256^2 = 4.295 \times 10^9$$

“local” FCN

- Idea: let each neuron only look at few neurons
- Force the network to respect pixel neighbor relationship
- Dramatically decrease # of weight parameters



# of weight parameters:

$$k \times D_{i+1}$$

For two 256x256 images, when $k = 3$:

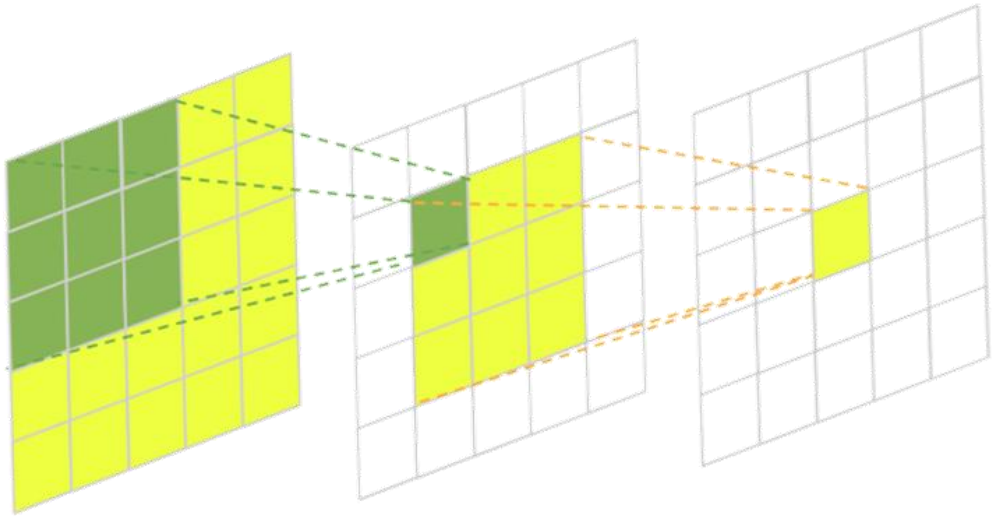
$$k \times D_{i+1} = 3 \times 256^2 = 1.966 \times 10^5$$

Before:

$$D_i \times D_{i+1} = 256^2 \times 256^2 = 4.295 \times 10^9$$

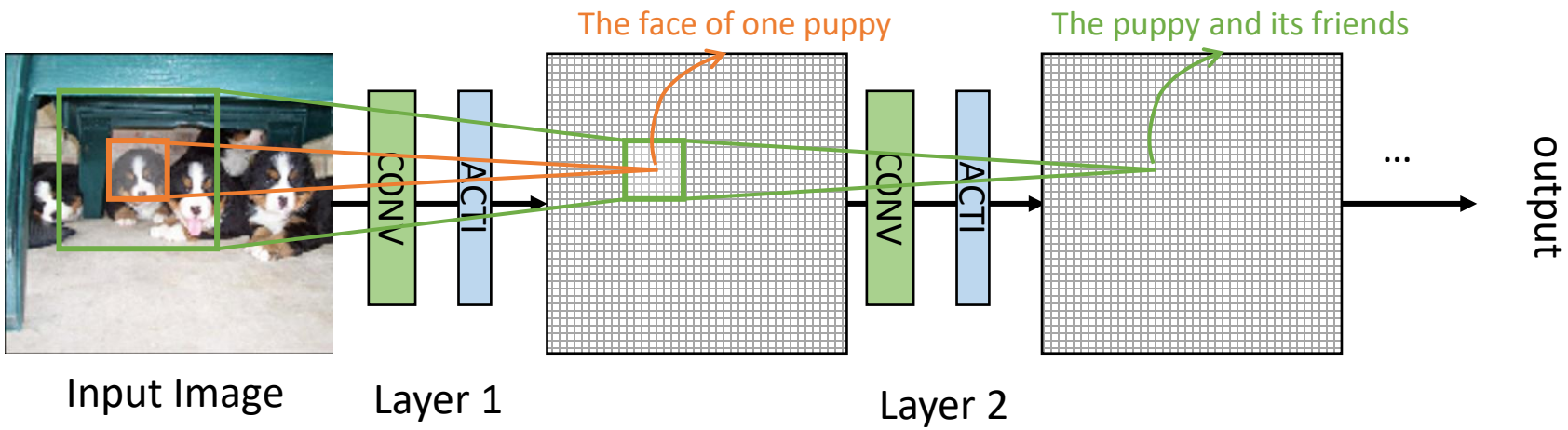
“local” FCN

- Idea: let each neuron only look at few neurons
- Force the network to respect pixel neighbor relationship
- Dramatically decrease # of weight parameters



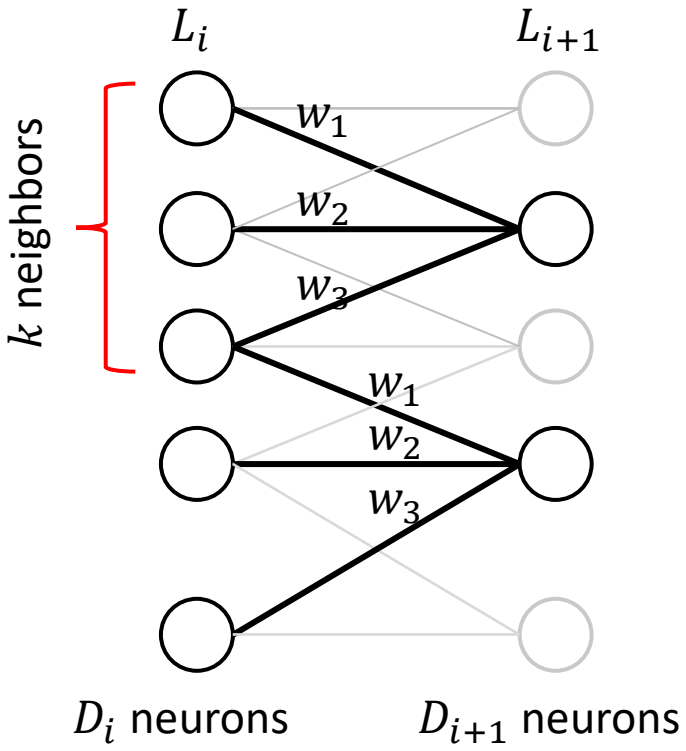
“local” FCN

- Idea: let each neuron only look at few neurons
- Force the network to respect pixel neighbor relationship
- Dramatically decrease # of weight parameters
- Deeper layers look at larger regions (receptive fields)
  - Capture information in all scales



“local” FCN with shared weights

- Share the weights to further reduce weight amount
- Can thinks this group of weights as “local feature extractor” that slides over the whole input image
- This is also called convolution!



# of weight parameters:

k

For two 256x256 images, when $k = 3$:

$k = 3$

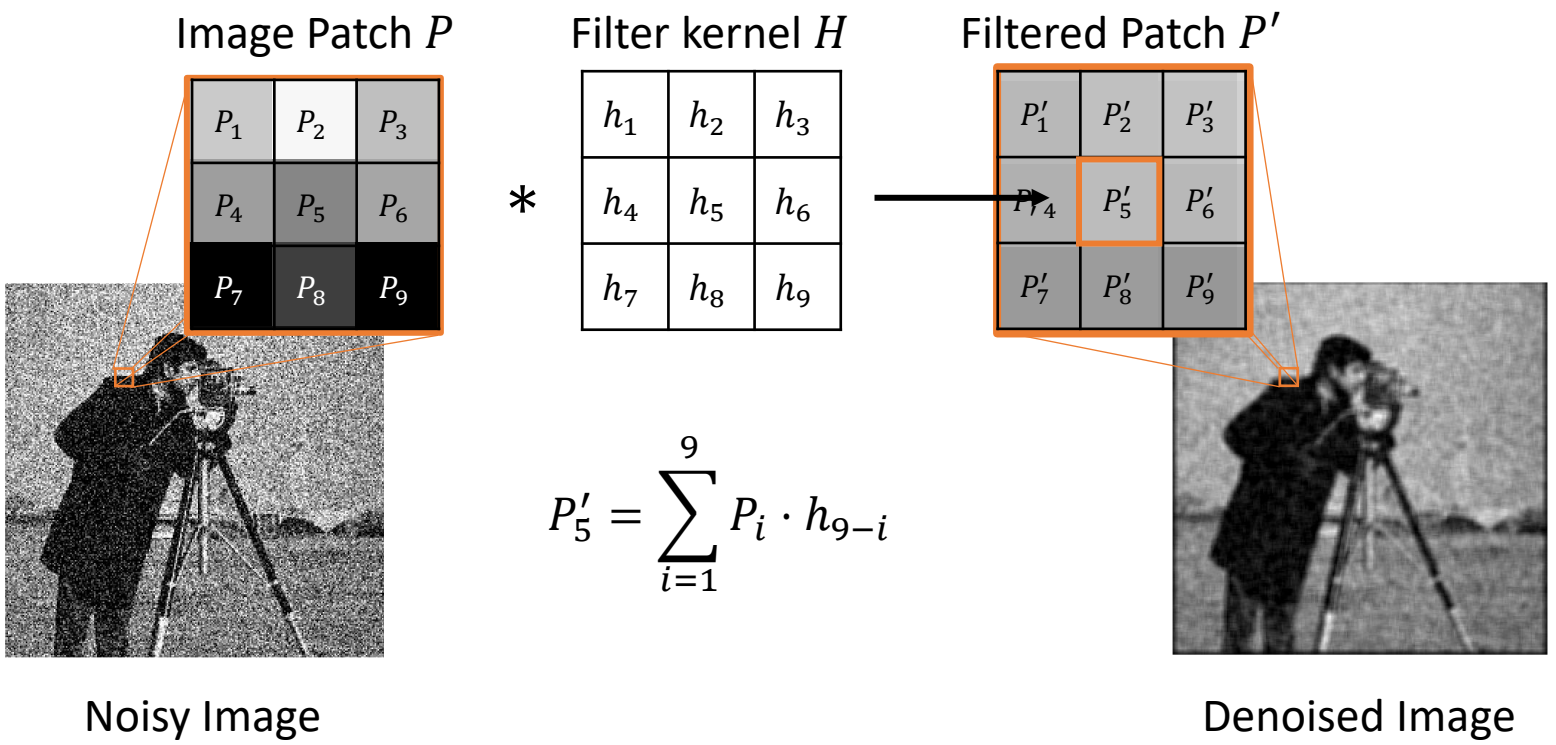
Before:

$k \times D_{i+1}$

$= 3 \times 256^2 = 1.966 \times 10^5$

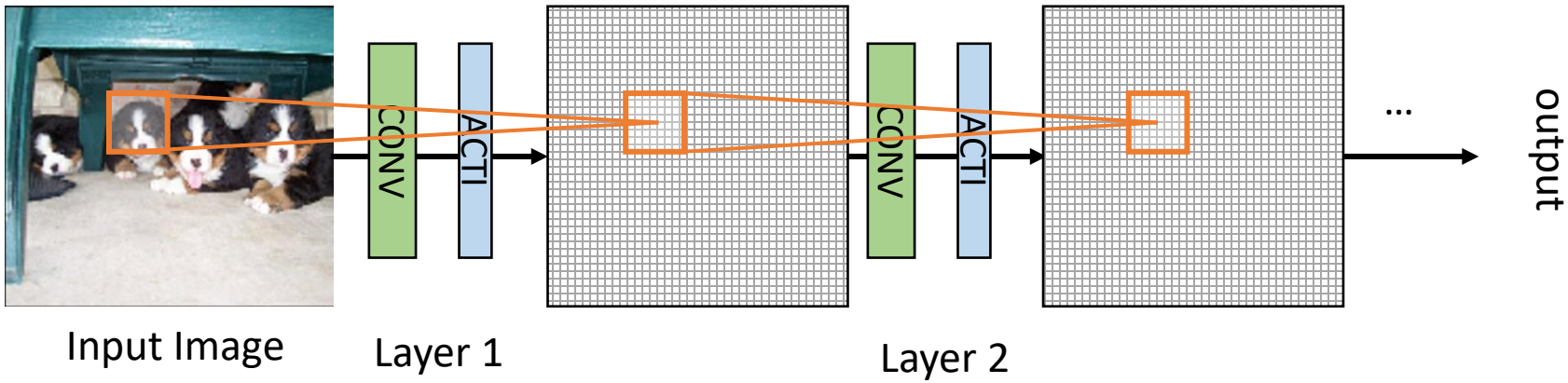
Convolutional Neural Network

- Recall: convolution with filter kernel for image processing
  - Use manually designed filters to process each patch of the image
- Examples
  - Average filter for image denoising ($h_1 = h_2 = \cdots = \frac{1}{9}$)
  - Edge detector for edge extraction



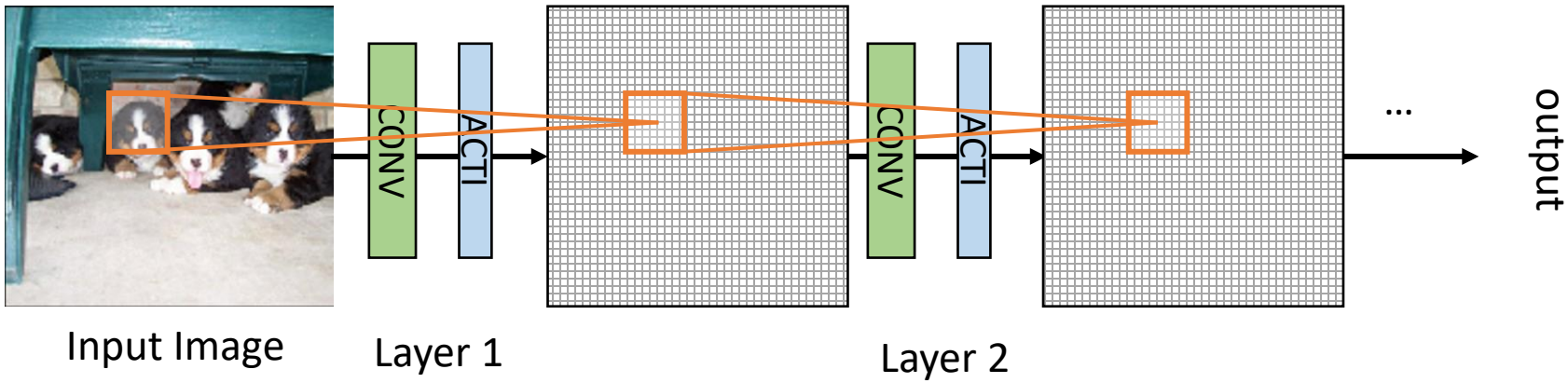
Convolutional Neural Network

- Convolutional neural network:
  N * [convolutional layer + activation function]
- Learn kernel values from data



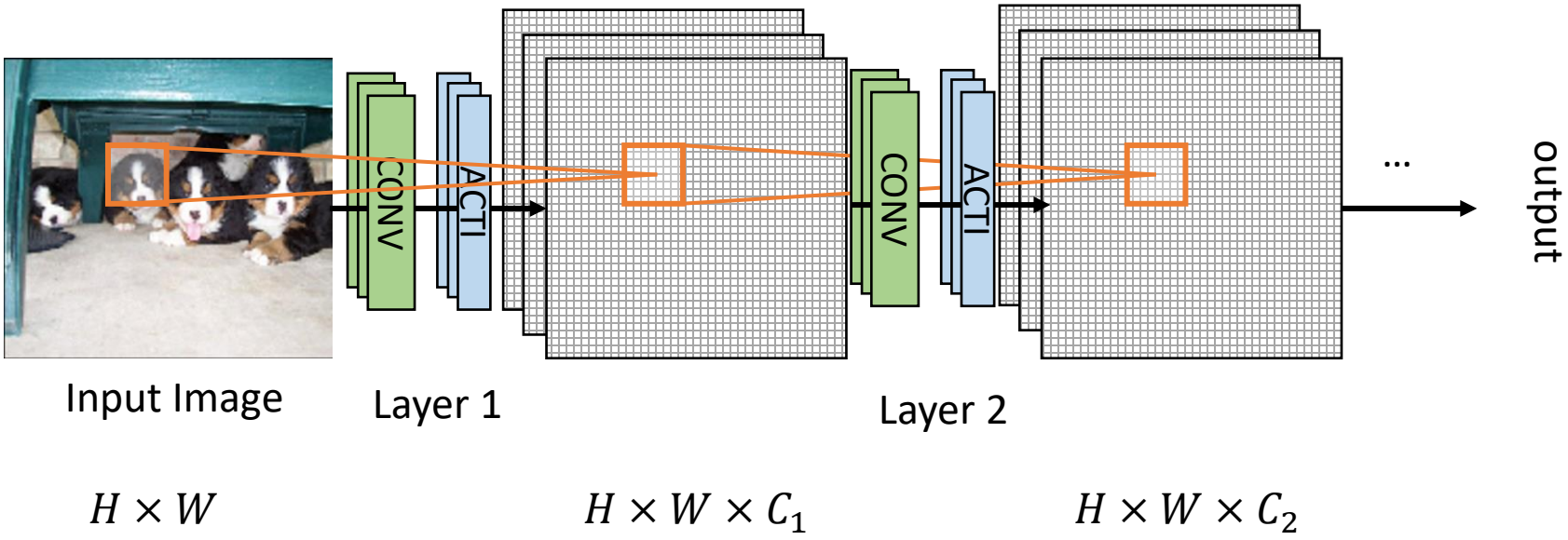
Convolutional Neural Network

- Many different useful feature for prediction
  - Animal classification example: clues of color, shape, pattern, etc.
- One kernel cannot capture them all
- Solution: using multiple kernels parallelly



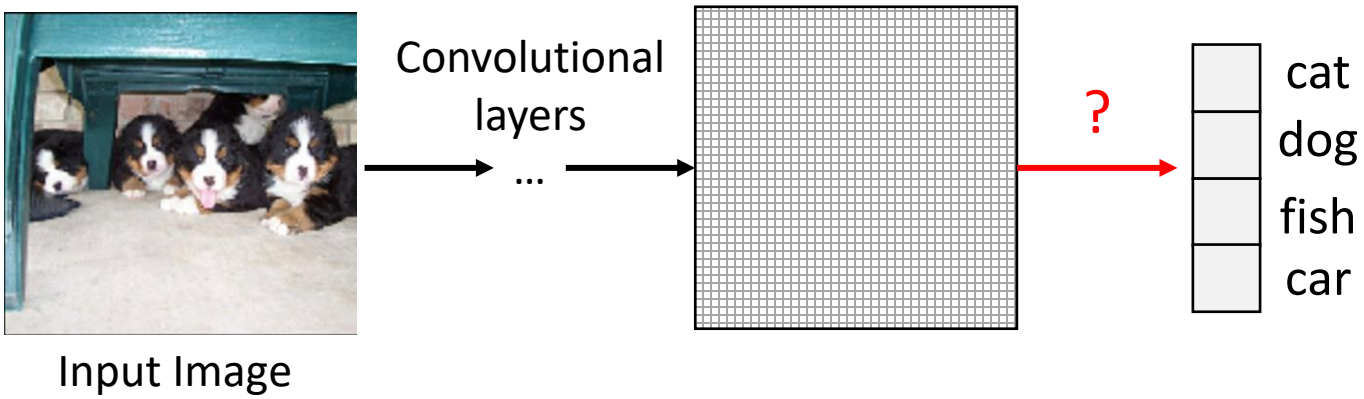
Multi-channel Convolutional Network

- We use multiple kernels for each layer, and call each layer a feature "channel"



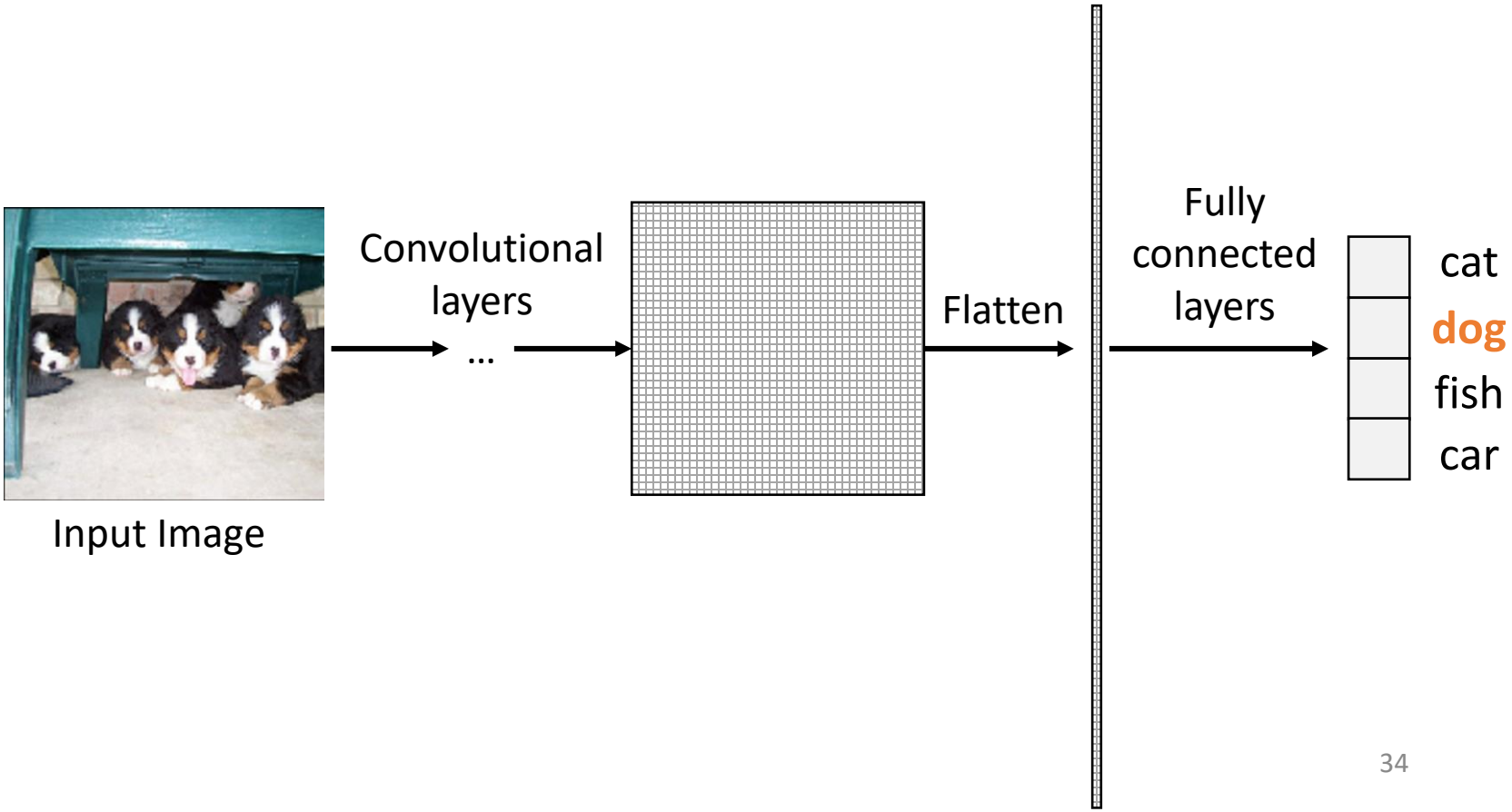
Use Fully Connected Layers for Classification

- The final output for classification task is usually a low-dimensional vector



Use Fully Connected Layers for Classification

- The final output for classification task is usually a low-dimensional vector
- Common practice: Vectorization (“flatten”) + fully connected layers

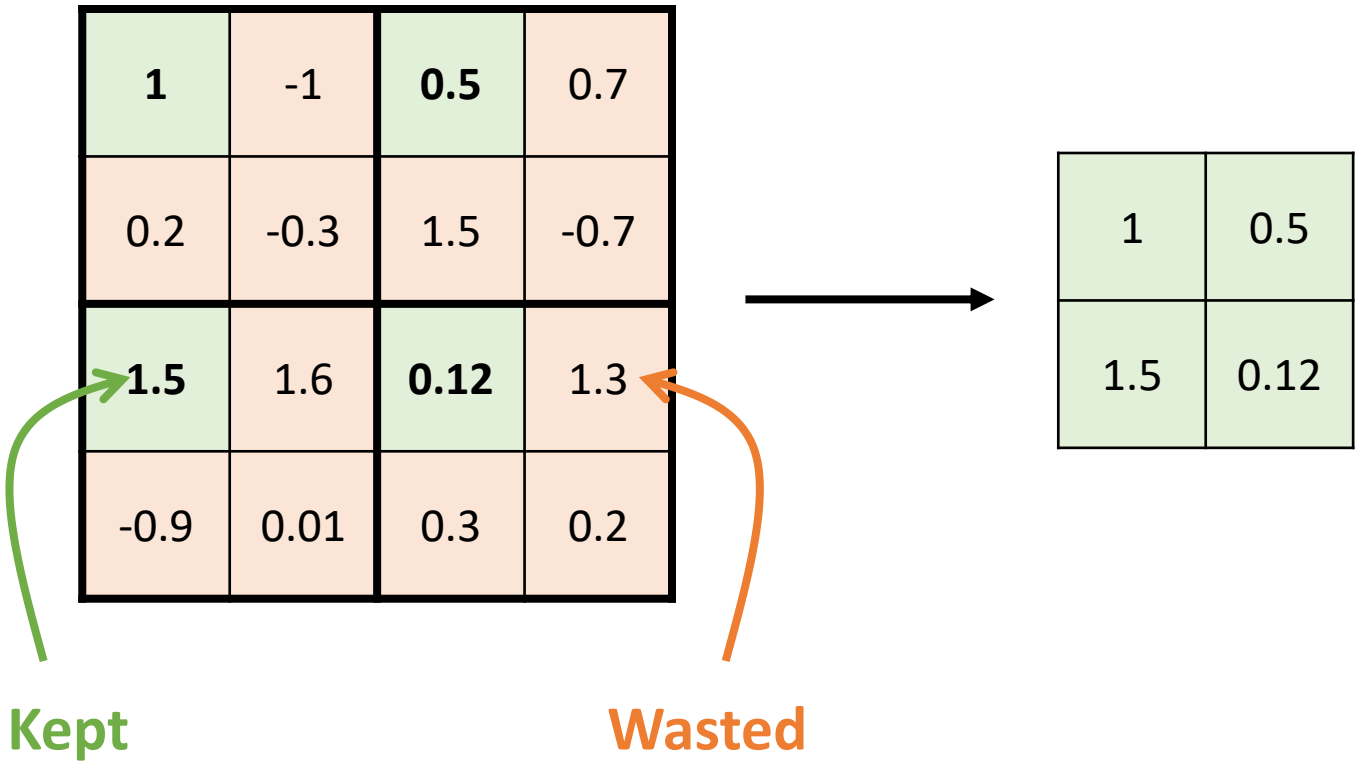


Quick Recap: Convolutional Neural Network

- Fully connected network: not suitable for image data
  - Discards pixel neighbor information
  - Too many parameters for high-dimensional data
  - low training efficiency
- Convolutional neural network
  - Allow only local connection between layers
  - Preserves pixel neighbor information
  - Enables the network to capture information at all scales
  - Much lower parameter size => higher training efficiency
- Next: two more common components in deep neural networks
  - Pooling

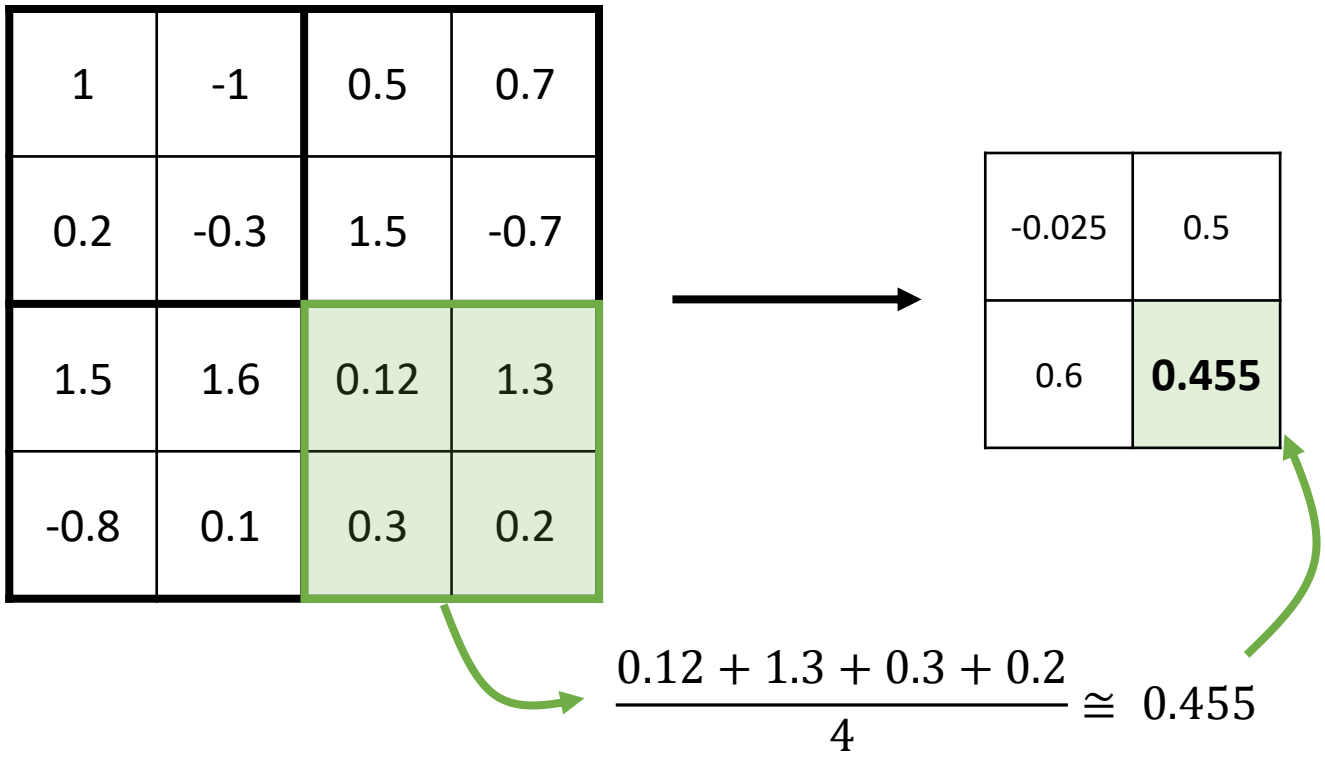
Reduce Data Dimension by Subsampling

- Straight-forward subsampling reduces data dimension
- too much information are untouched and wasted



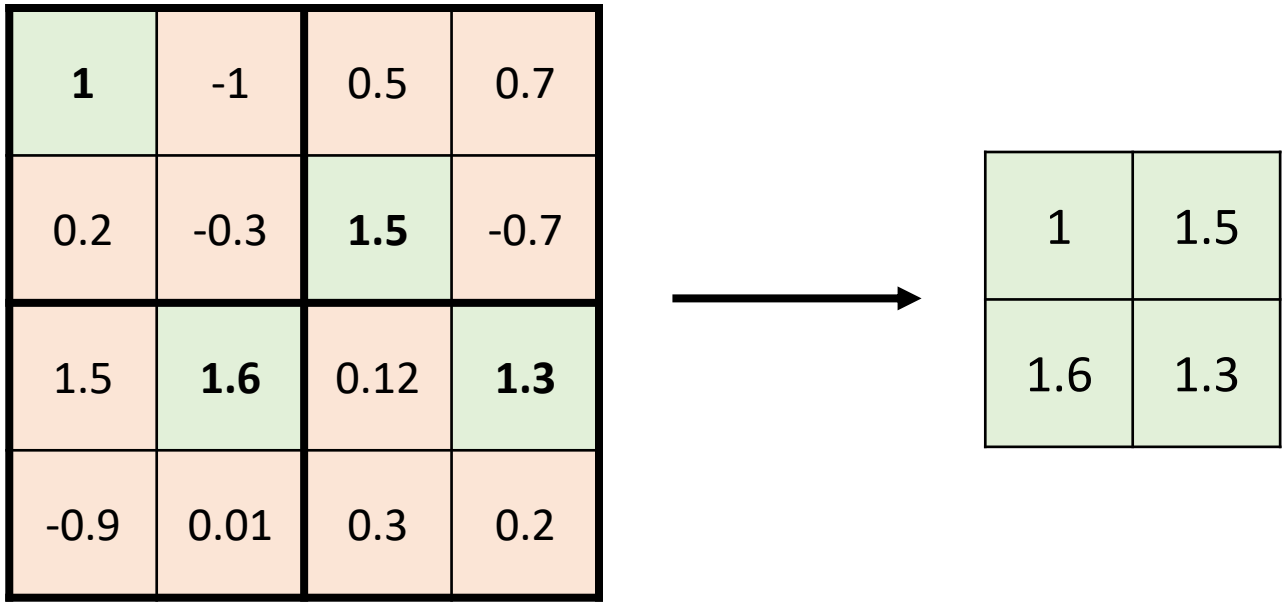
Pooling Layers

- "Wisely" subsample the data
  - Average pooling: take the average value in local region



Pooling Layers

- "Wisely" subsample the data
  - Average pooling: take the average value in local region
  - Max pooling: keep only the largest value in local region

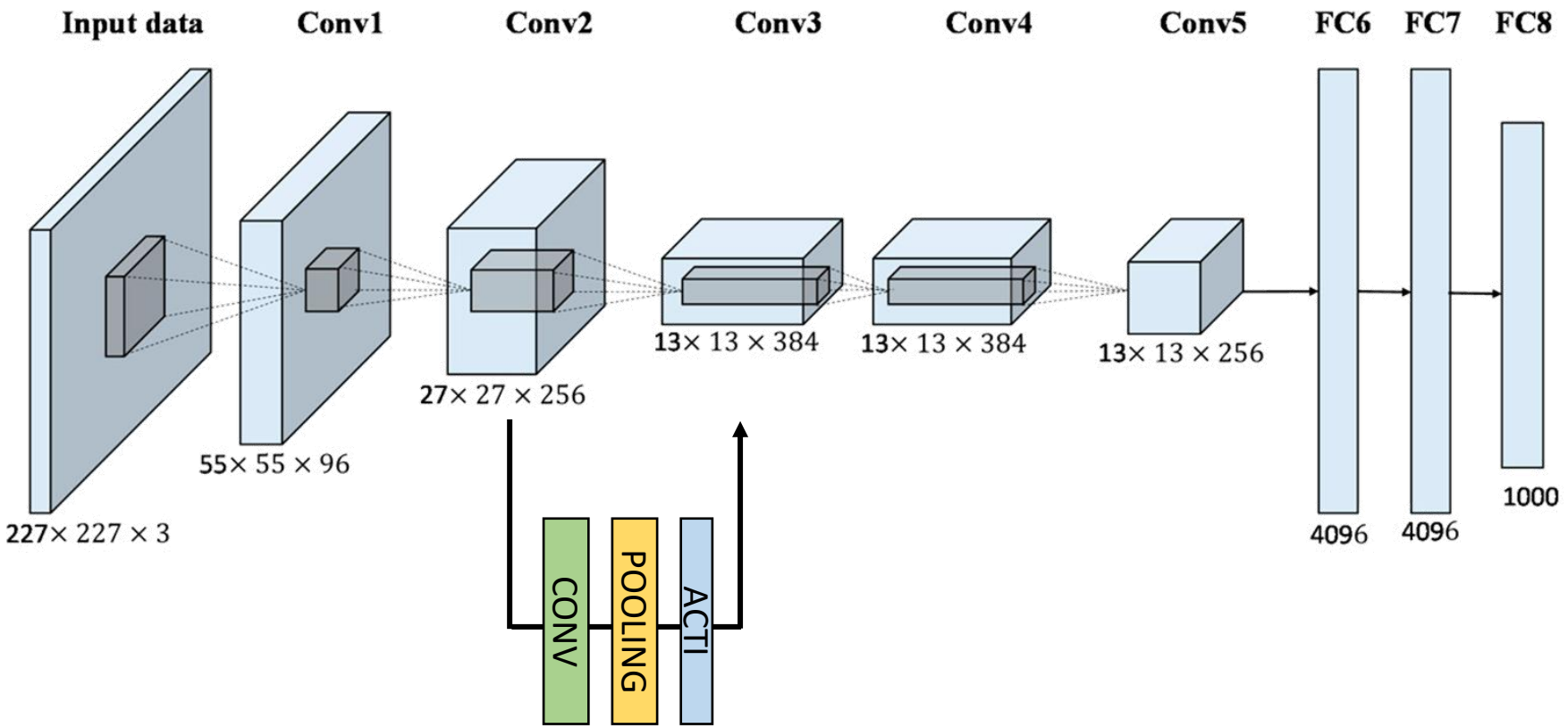


Pooling Layers

- “Wisely” subsample the activation maps
  - Average pooling: take the average value in local region
  - Max pooling: keep only the largest value in local region
- Efficiently activation map size
- Avoid overfitting
- Additional nonlinearity

Combining All Pieces Together

- Example: AlexNet



Conclusion

- Solving problem by searching a good approximator
- Deep neural network = great approximator
  - Complex enough
  - Efficient training by backpropagation
- Fully connected network (FCN)
  - A simple neural network structure
  - Stack of perceptrons
  - Discards pixel neighbor information
  - Too many parameters for high-dimensional data
  - low training efficiency
- Convolutional neural network (CNN)
  - Allow only local connection between layers
  - Preserves pixel neighbor information
  - Enables the network to capture information at all scales
  - Much lower parameter size => higher training efficiency
- Pooling