ECE 661 COMP ENG ML & DEEP NEURAL NETS

3. 60 YEARS OF NEURAL NETWORK (2/3): EMERGENCE OF CONVOLUTIONAL LAYERS

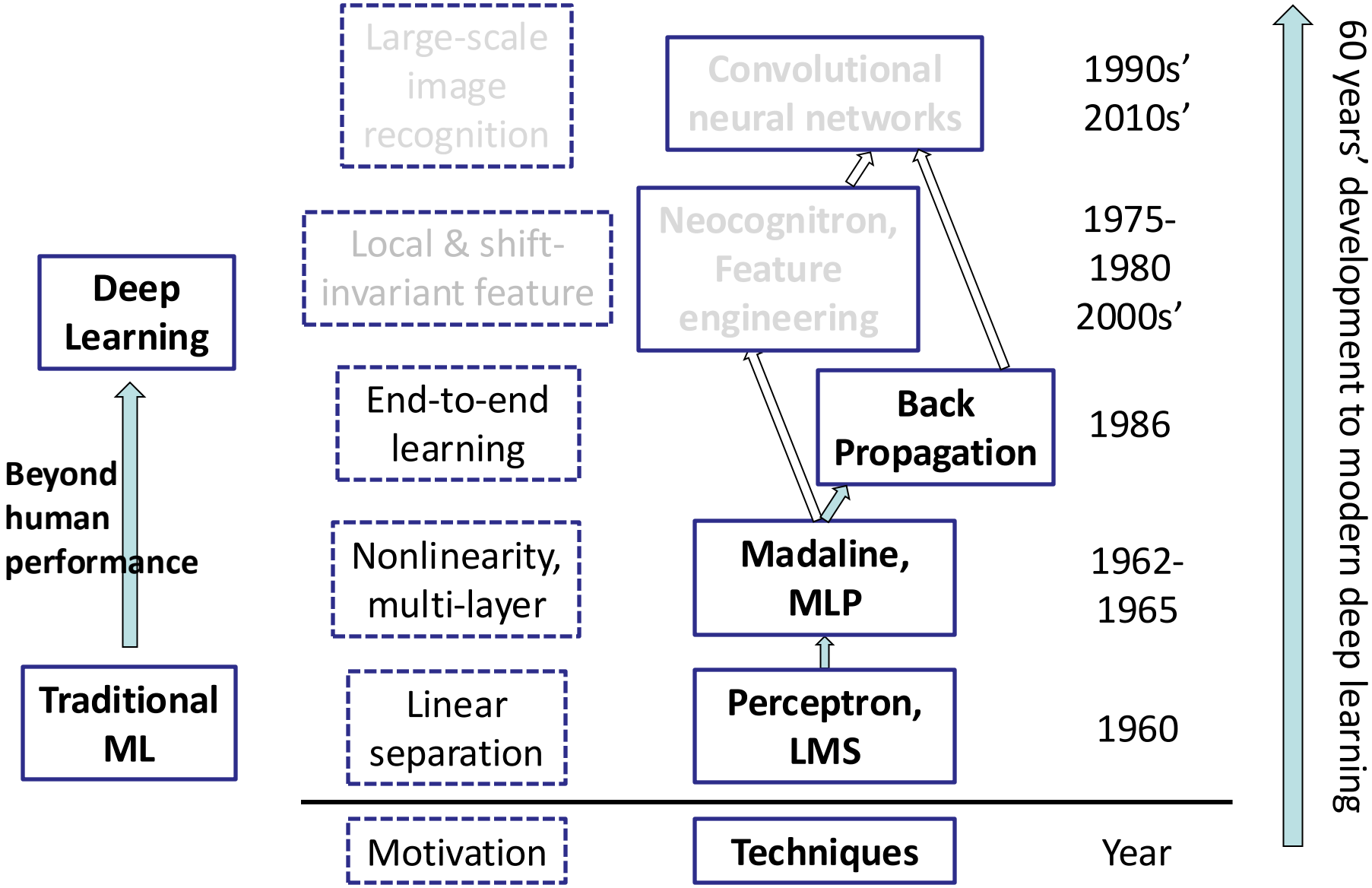
HAI LI, SPRING 2025

Logistics – Office hours Update

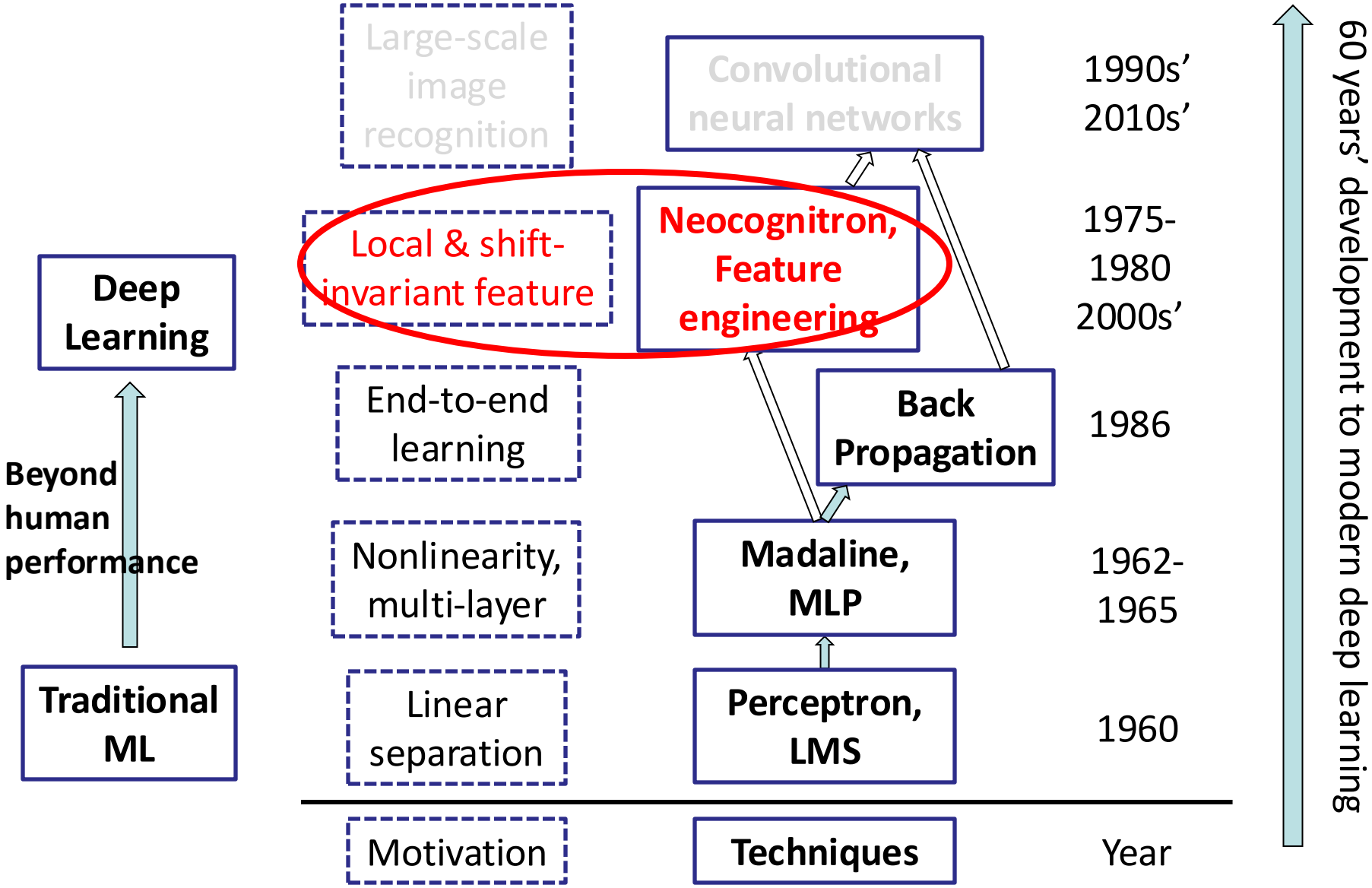
ECE 661 COMP ENG ML & DEEP NEURAL NETS		
Faculty:	Dr. Hai "Helen" Li	Hai.li@duke.edu
Lectures:	Mondays/Wednesdays 10:05 AM – 11:20 AM FITZPATRICK SCHICIANO B 1466 In person only. No recording	
Office Hours:	By Appointment (please send email to Dr. Li)	
Teaching Assistants:	Junyao Zhang Ben Morris Easop Lee James Kiessling Mark Horton Zhixu Du	jz420@duke.edu ben.morris@duke.edu easop.lee@duke.edu james.kiessling@duke.edu mark.horton@duke.edu zhixu.du@duke.edu
Office Hours:	Junyao & Easop: Monday 4:00 PM-5:00 PM Wilkinson 420 Ben & Mark: Wednesday 4:00 PM-5:00 PM Wilkinson 419 Zhixu & James: Friday 4:00 PM-5:00 PM Wilkinson 420	

Office hour starts from Friday of the second week (January 17).

Previously


Large-scale image recognition
Convolutional neural networks
1990s'
2010s'
Local & shift-invariant feature
Neocognitron, Feature engineering
1975-1980
2000s'
Deep Learning
End-to-end learning
Back Propagation
1986
Beyond human performance
Nonlinearity, multi-layer
Madaline, MLP
1962-1965
Traditional ML
Linear separation
Perceptron, LMS
1960
Motivation
Techniques
Year
60 years' development to modern deep learning

This lecture



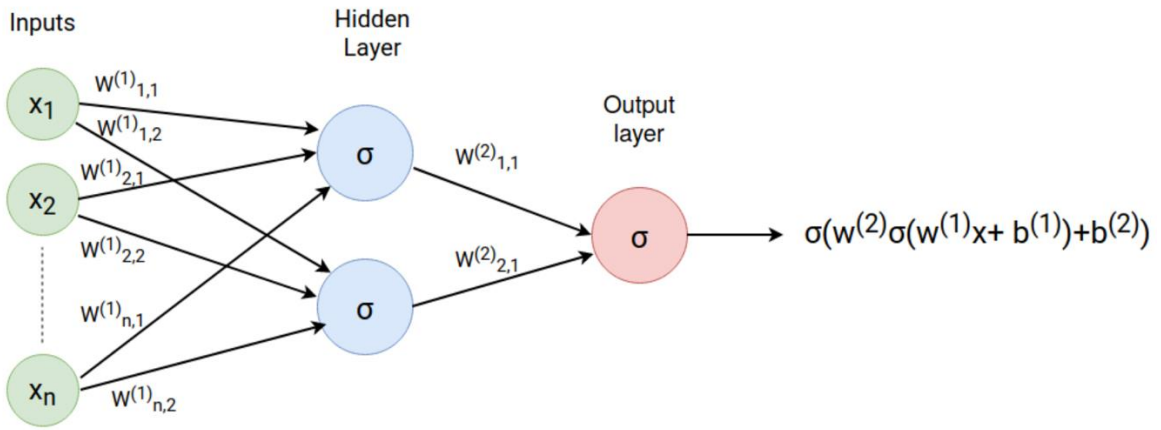
Outline

- Lecture 2: NN and Back Propagation
- Lecture 3: Emergence of convolution
  - Useful features for image classification
  - Convolution and feature engineering
  - Hierarchical structure
  - Putting things together: CNN

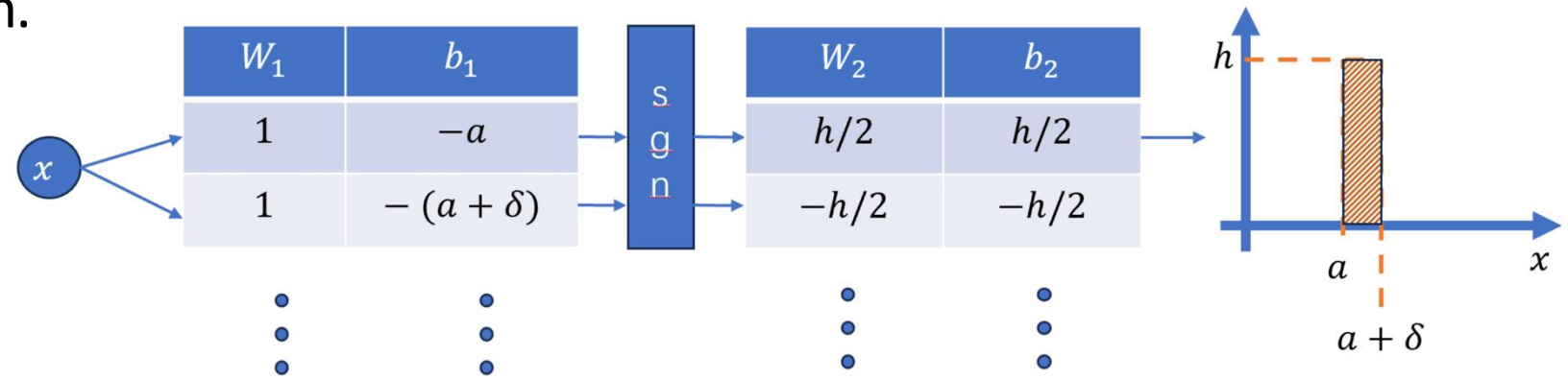
- Lecture 4: Building a CNN model

Where we left

- Multilayer Perceptron's Non-linearity
  - A Perceptron with a single hidden layer, even with a non-linear activation function, remains a linear classifier and cannot model an XOR function.
  - To model complex relationships, a Multilayer Perceptron requires non-linear activation functions (such as ReLU or sigmoid) between its layers.

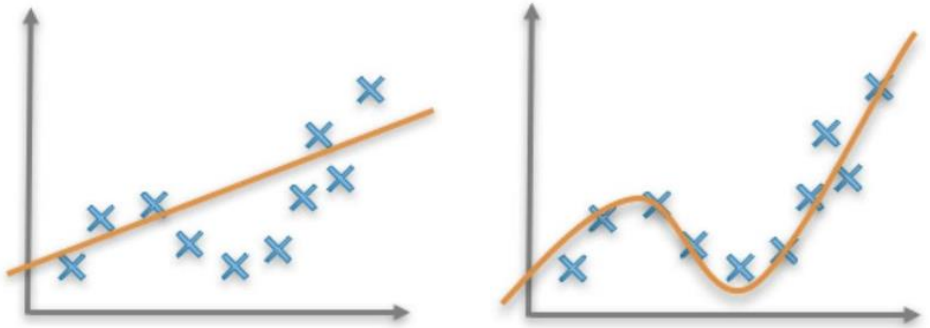


- Construct arbitrary function (with specific precision) with sgn activation for non-linear classification.



Where we left

- Multilayer Perceptron's Non-linearity
  - A Perceptron with a single hidden layer, even with a non-linear activation function, remains a linear classifier and cannot model an XOR function.
  - To model complex relationships, a Multilayer Perceptron requires non-linear activation functions (such as ReLU or sigmoid) between its layers.
  - Without non-linear activation functions, linear networks are equivalent to a single layer.



Linear function

Non-linear function

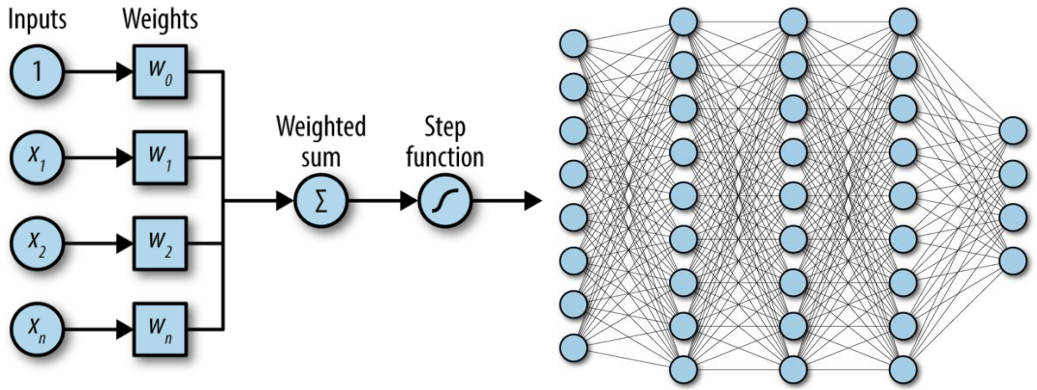
$$y = (\dots(w_3(w_2(w_1 x + b_1) + b_2) + b_3)\dots)$$

When no non-linear activation,
linear networks are equivalent to a single layer!

$$y = Wx + B$$

Where we left

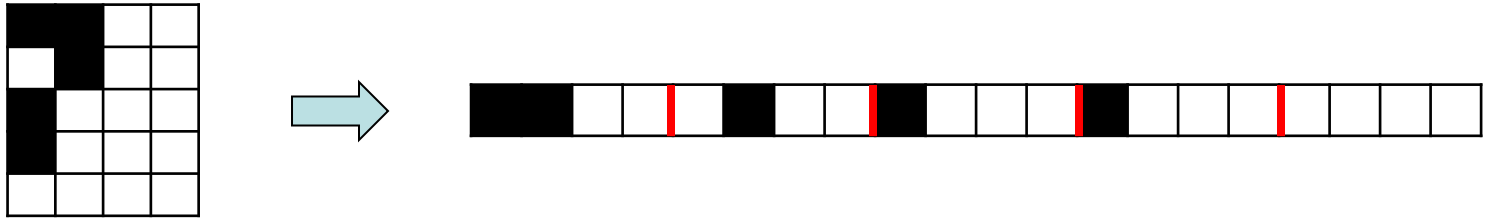
- Fully connected (FC) networks
  - Takes an $N \times 1$ vector as the input, a scalar or a vector as the output
  - **Universal approximation theorem**: A 2-layer FC with arbitrarily wide hidden layer and a suitable non-polynomial activation function can approximate any smooth function that maps one vector to another



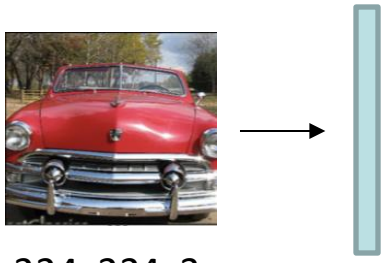
- FC models are theoretically powerful, but has practical limitations
  - What about having 2-D matrix as the input, like an image?
  - Reshape it into a vector? Let's give a try.

Processing images with FC

- Step 1: Reshape an image into a 1-D vector
  - Not human-meaningful anymore. All spatial information is lost
  - Not a big deal for FC though, since it "learns everything"



- Step 2: Design a model
  - Recall: $N \times 1$ vector as input, $N \times M$ weight in layer 1 (M is the number of the output neurons)
  - To avoid dramatic loss of information, $M \sim N/3$ to $N/2$
  - For a typical color image:



Potentially 10+ Billion parameters in layer 1 alone, without considering the remaining layers

Processing images with FC

- Step 3: Shifted input
  - We expect the model to recognize the object in the image wherever it is
  - Yet the reshaped vector changes dramatically after shifting, hard to make a FC model to consider the two inputs identical

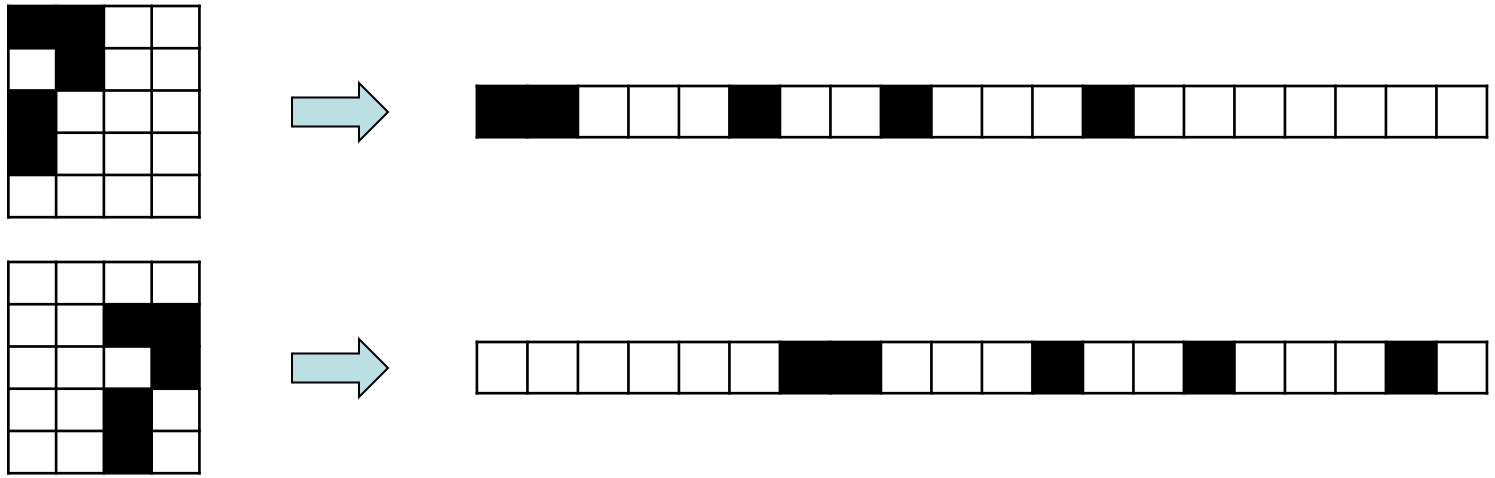
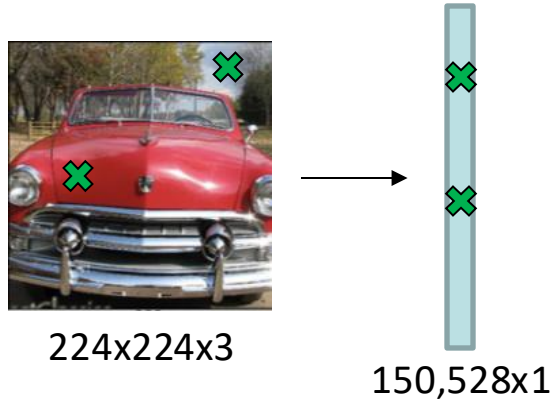
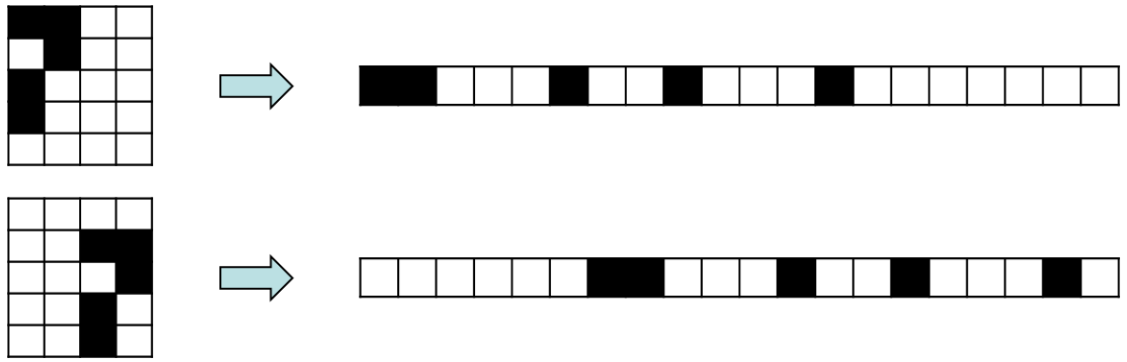


Image features that FC hardly captures

- Loss of spatial information/locality
  - Not all pairs of pixels have useful connections. Fully connecting all the pixels is a waste of effort
  - Humans recognize images largely based on local information

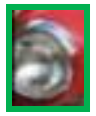


- Shift invariant
  - FC behaves poorly when input is shifted



Local, **shift-invariant** features are important in image processing

How should the feature be captured?

- Suppose we are asked to find out if a “headlight” exists 
- Local feature: Design a function to check if the pixels within a small rectangle region is a headlight 
  - This function is called as a **filter**
  - Designing this filter manually is called **feature engineering**
- Shift invariant: Gradually shift the filter and apply it on every rectangle region in the image
  - Shifting and applying the filter leads to **convolution**

Not a headlight | Still not | **There it is!**

Outline

- Lecture 2: NN and Back Propagation
- Lecture 3: Emergence of convolution
  - Useful features for image classification
  - Convolution and feature engineering
  - Hierarchical structure
  - Putting things together: CNN
- Lecture 4: Building a CNN model

Linear filter function

- Suppose the filter is applied to a $d_1 \times d_2$ region f, then the filter function is a function of all $d_1 d_2$ pixels within this region

- The easiest case: Linear function
  - Define $d_1 \times d_2$ weights g, named **kernel**
  - Output the weighted sum of all pixels within the region

$$f * g = \sum_{d_1, d_2} f[i,j] g[i,j]$$

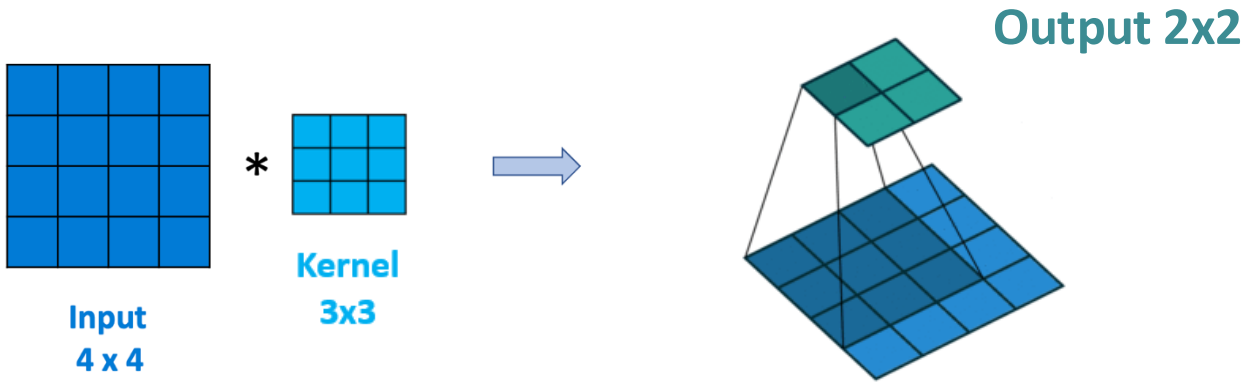
  - Example

$$\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = 1 + 4 + 6 + 4 = 15$$

Convolution operation

- Shifting and applying the linear filter
  - Start with applying the kernel at the top-left corner, the output goes to the top-left pixel of the **output feature map**
  - Shift kernel over the image to get other output pixels
  - Linear operation (weighted sum) is applied on the input region overlapped by the filter

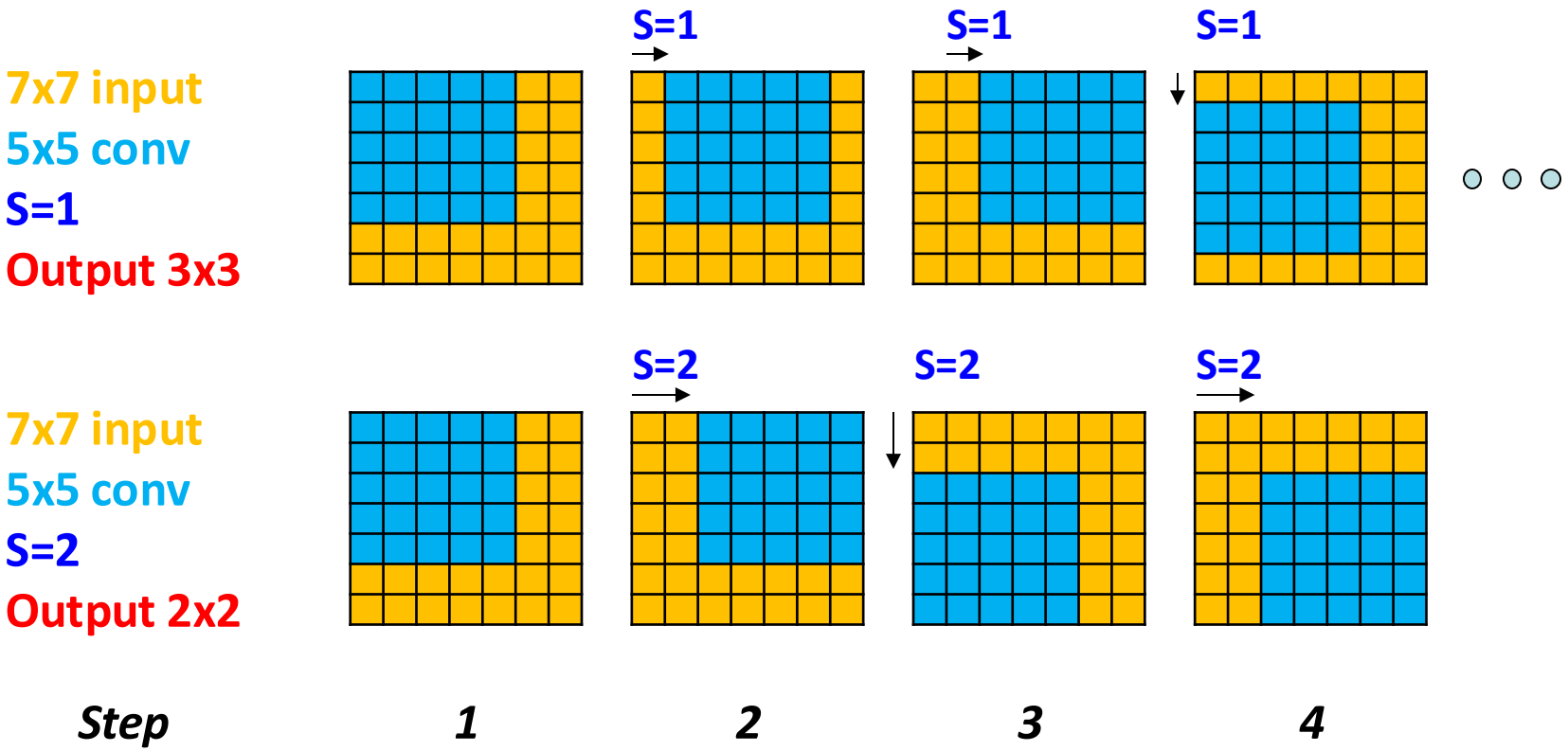
$$(f * g)[x, y] = \sum_{d_1, d_2} f[x + i, y + j] g[i, j]$$



https://towardsdatascience.com/types-of-convolution-kernels-simplified-f040cb307c37

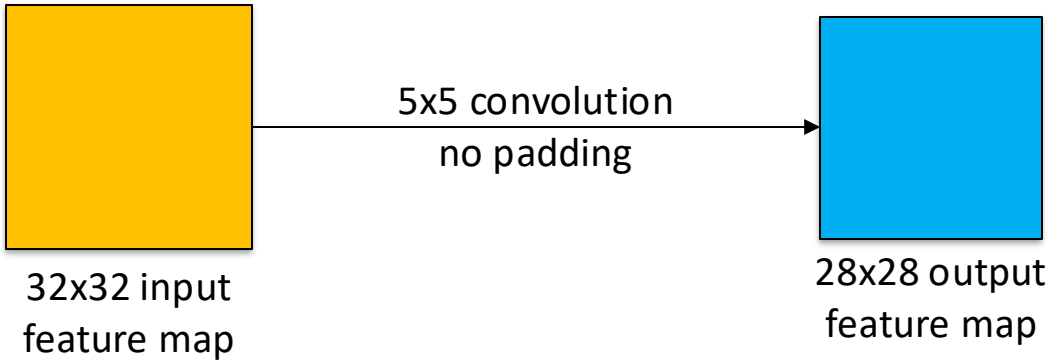
How filter moves: strides

- **Strides** control the amount by which the filter slides over the input image. Use *S* to denote the strides of one convolution operation.

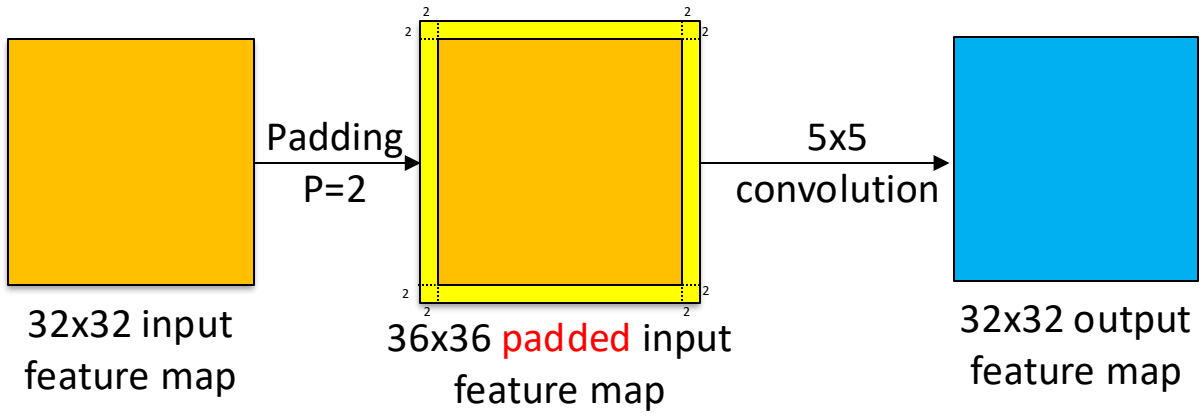


Prevent shape change: padding

- **Padding** prevents the change of the feature map size after the convolutional operation. Use *P* to denote the number of paddings on each boarder.



Zero padding is the most common way to pad the input feature map.



For stride S=1, to maintain the input shape, padding is set to

$$P = \left\lfloor \frac{K-1}{2} \right\rfloor$$

Convolution shape rule

- Assume the input feature map has a shape of $\mathrm{H}_1 \times \mathrm{W}_1$.
- Convolution configuration:
  - Convolution kernel size *K*
  - Stride for convolution *S*
  - Padding for each boarder *P*

- The output feature map has the following shape $H_2 \times W_2$, where

$$W_2 = \left\lfloor \frac{W_1 - K + 2P}{S} \right\rfloor + 1$$

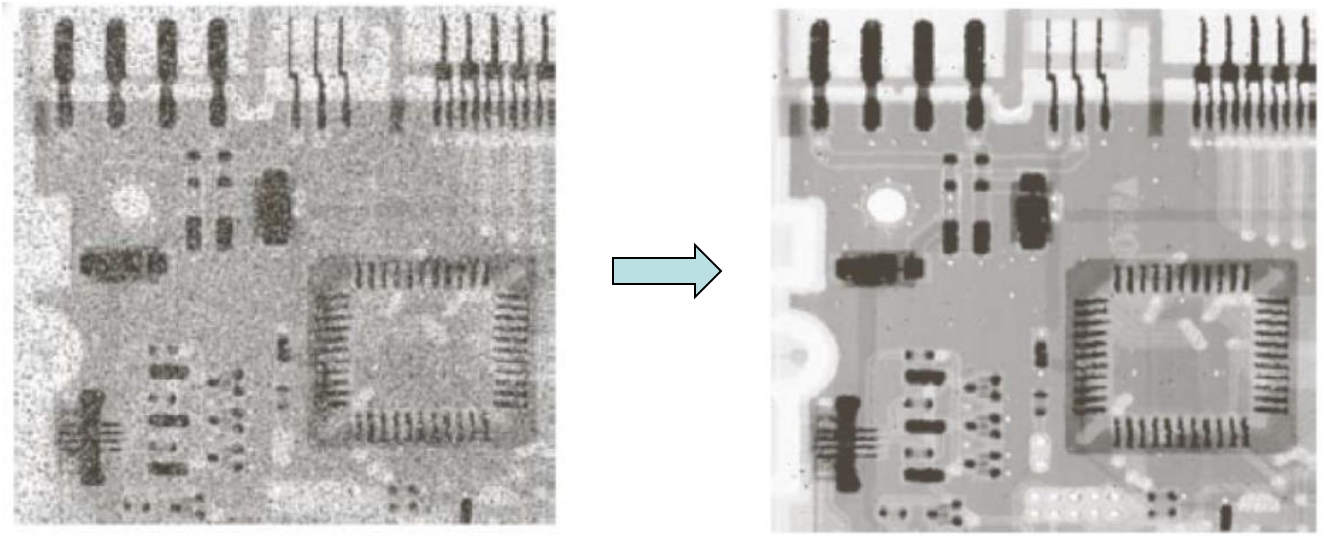
$$H_2 = \left\lfloor \frac{H_1 - K + 2P}{S} \right\rfloor + 1$$

- If confused, draw some blocks to help understanding ☺

Case study: Selected famous kernels

- Before CNN gets popular, researchers manually designed a lot of convolutional kernels to fulfill image process or feature extraction task, namely **feature engineering**
- **Example 1: Smoothing operator**
  - Smoothing the image
  - Can take form of a moving average
  - Or a 3x3 median filter (return median value within 3x3 region)

$$\begin{bmatrix} 0 & 1/4 & 0 \\ 1/4 & 0 & 1/4 \\ 0 & 1/4 & 0 \end{bmatrix}$$



Figures credited to Dr. M. K. Mandal, University of Alberta

Case study: Selected famous kernels

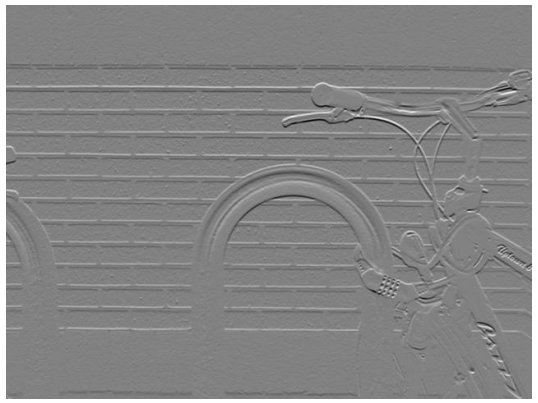
- **Example 2: Sobel operator (edge detector)**
  - Approximate of first-order gradient of an image, famous in early computer vision works

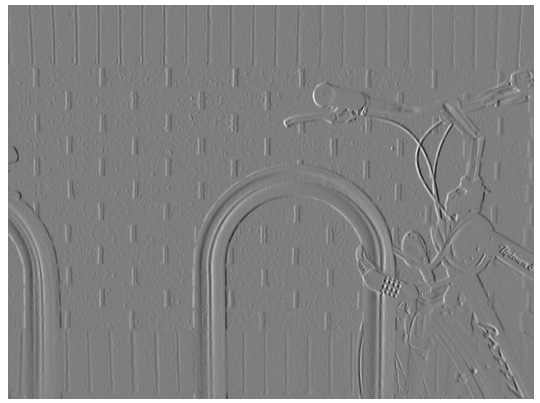
Horizontal edge

$$G_x = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Vertical edge

$$G_y = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

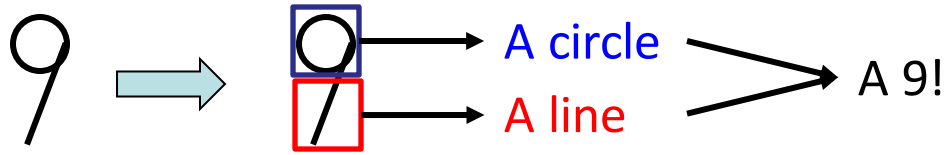




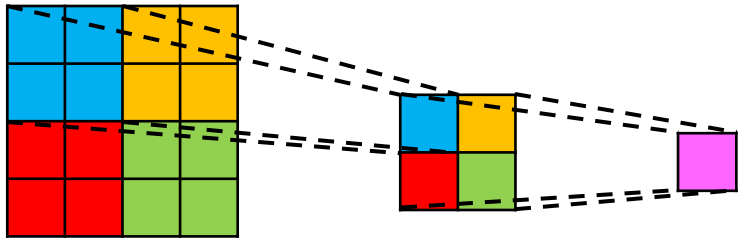
Figures credited to Wikipedia https://en.wikipedia.org/wiki/Sobel_operator

What's still missing

- As a linear function, a single convolution operation can effectively extract a few small local features, but the capacity is limited. It can't directly represent complex feature/objects
- Typical 3x3 or 5x5 convolution kernels can't capture larger features
- Need **hierarchical structure**!
  - Combine the features of multiple filters towards more completed features (like MLP)



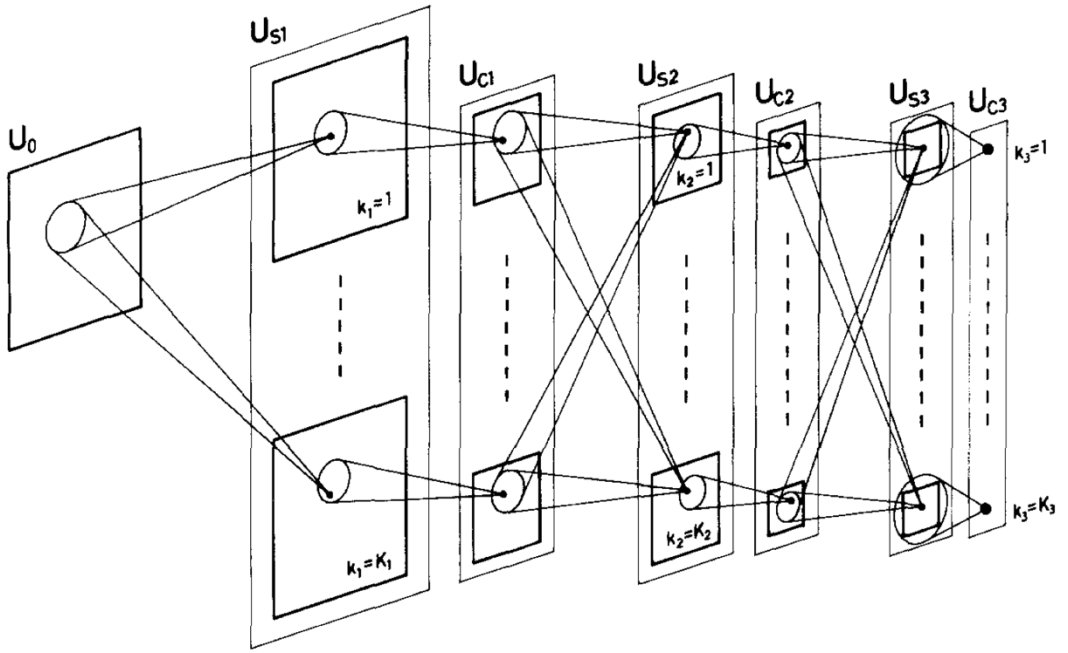
  - Higher-layer filters take in the information from a larger range of input image pixels

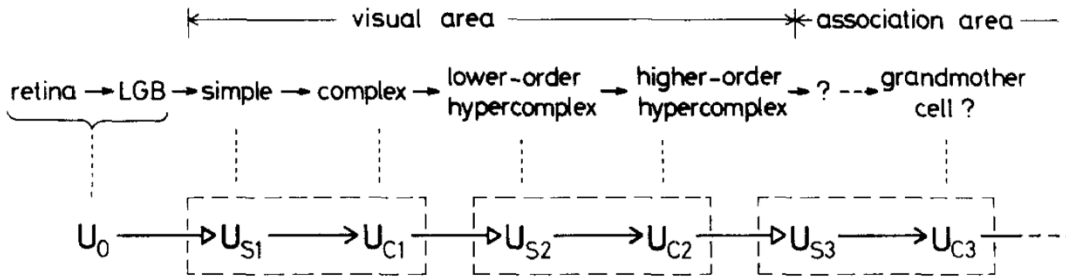


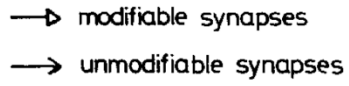
Outline

- Lecture 2: NN and Back Propagation
- Lecture 3: Emergence of convolution
  - Useful features for image classification
  - Convolution and feature engineering
  - Hierarchical structure
  - Putting things together: CNN
- Lecture 4: Building a CNN model

Brain inspired structure: Neocognitron



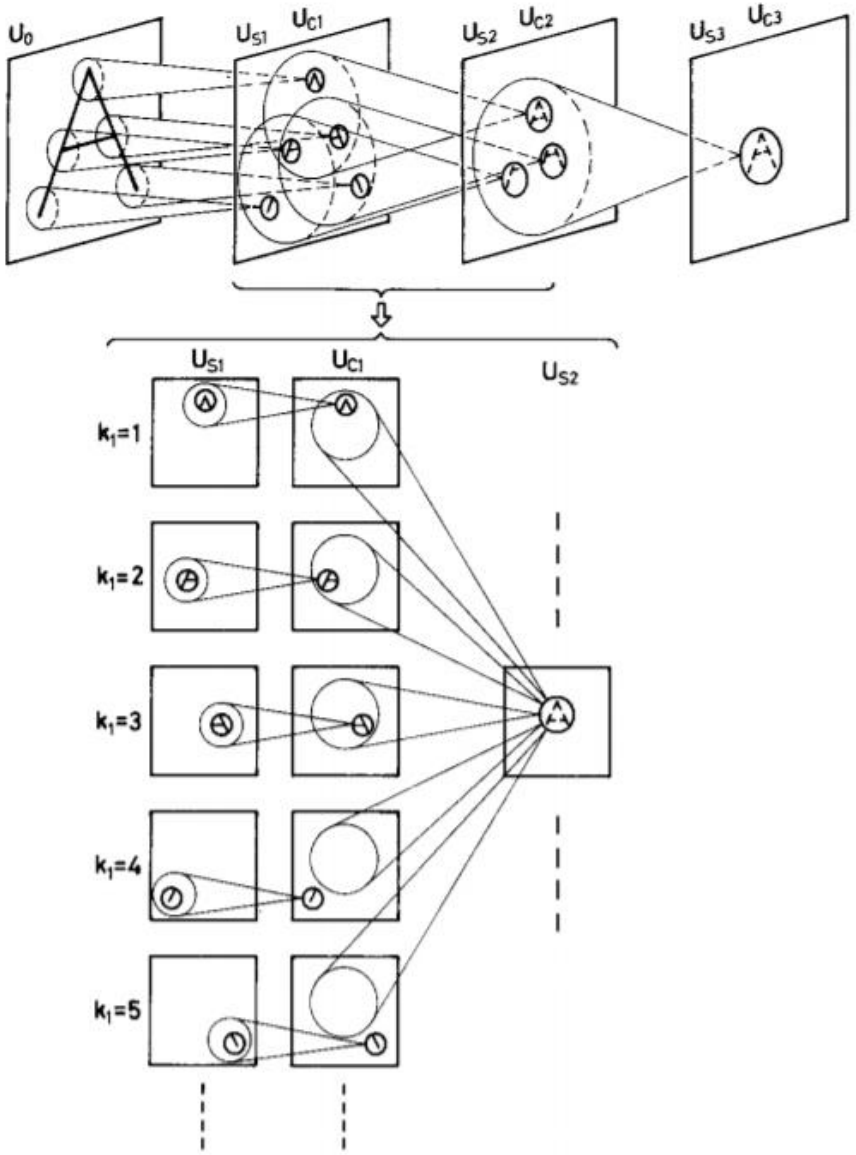




- Fukushima, Kunihiko. “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position.” (1980)
- S-C-S-C-S-C structure
  - Simple cells: cells with trainable parameters -> **convolution**
  - Complex cells: Perform down-sampling which extracts features and cannot be modified. -> **pooling** (will be discussed in Lec 4)

S->C: unmodifiable layers for transformation; C->S: modifiable trainable layers to extract feature

Neocognitron at work



- S -> C: Down sampling/pooling
  - Down sample output features to fit more filter outputs into the **receptive field** (draw as circles on feature map) of a later filter
- C -> S: Convolution
  - Gather the extracted local feature from previous layer's output feature map to form a more complex feature
- Gradually captures more abstract features towards later layers

Outline

- Lecture 2: NN and Back Propagation
- Lecture 3: Emergence of convolution
  - Useful features for image classification
  - Convolution and feature engineering
  - Hierarchical structure
  - Putting things together: CNN
- Lecture 4: Building a CNN model

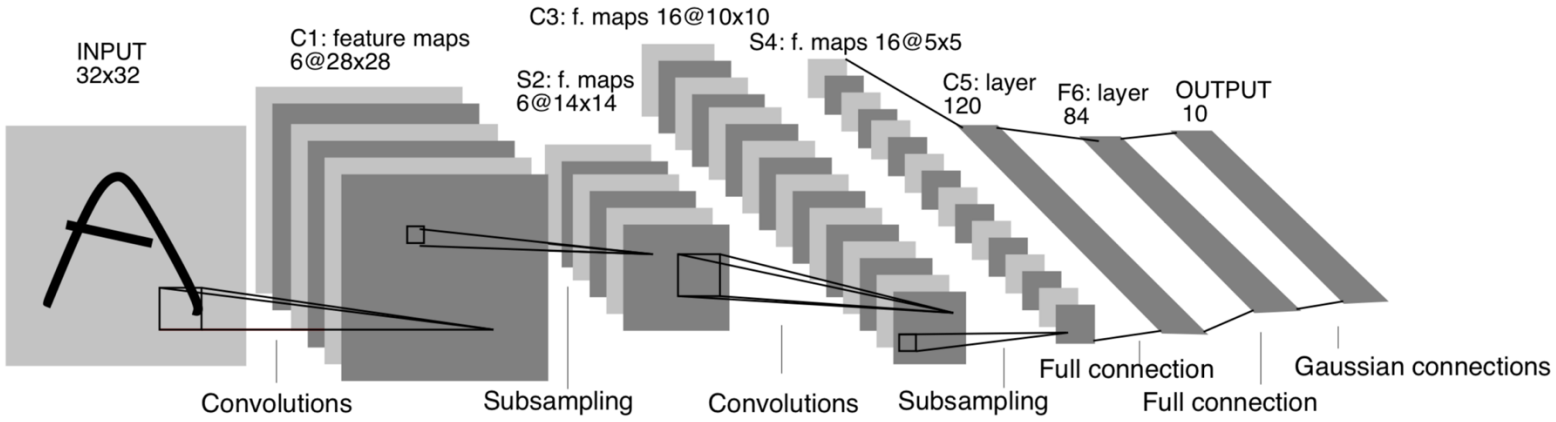
Components of a modern CNN model

- Convolutional layers
  - Perform convolution operation on input feature maps with multiple filters
- Pooling layers
  - Down sampling the feature map, gathering more information into the receptive field of later layer
- FC layer (classifier)
  - With high-level features extracted in later CONV layers, use those features to perform learning task
- Everything learns end-to-end with back propagation

Technical and implementation details coming in Lecture 4

LeNet-5

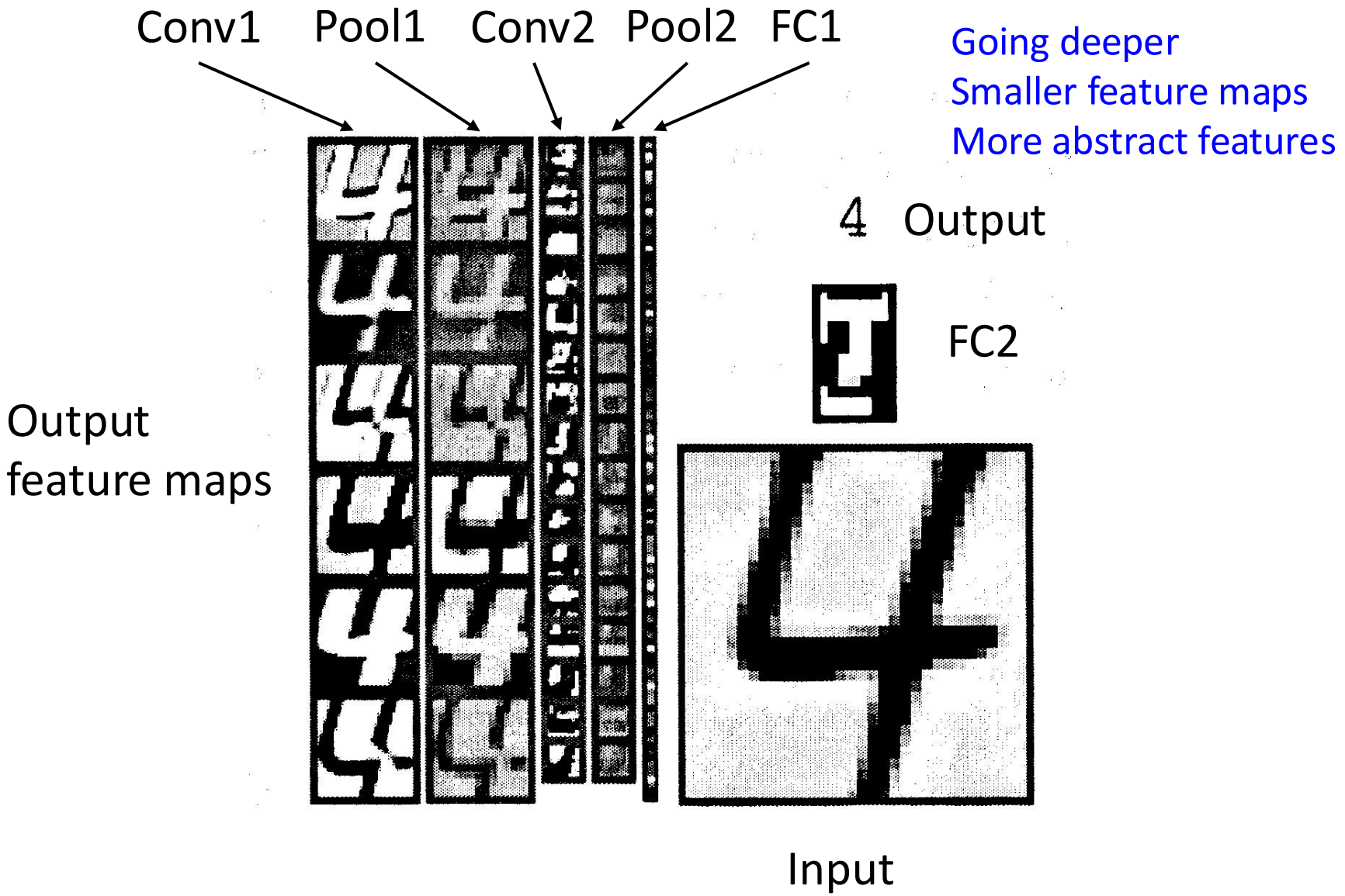
- Case-study: LeNet-5 for MNIST handwritten digits classification
  LeCun, Yann et al. "Gradient-based learning applied to document recognition." (1998)



LeNet Structure:

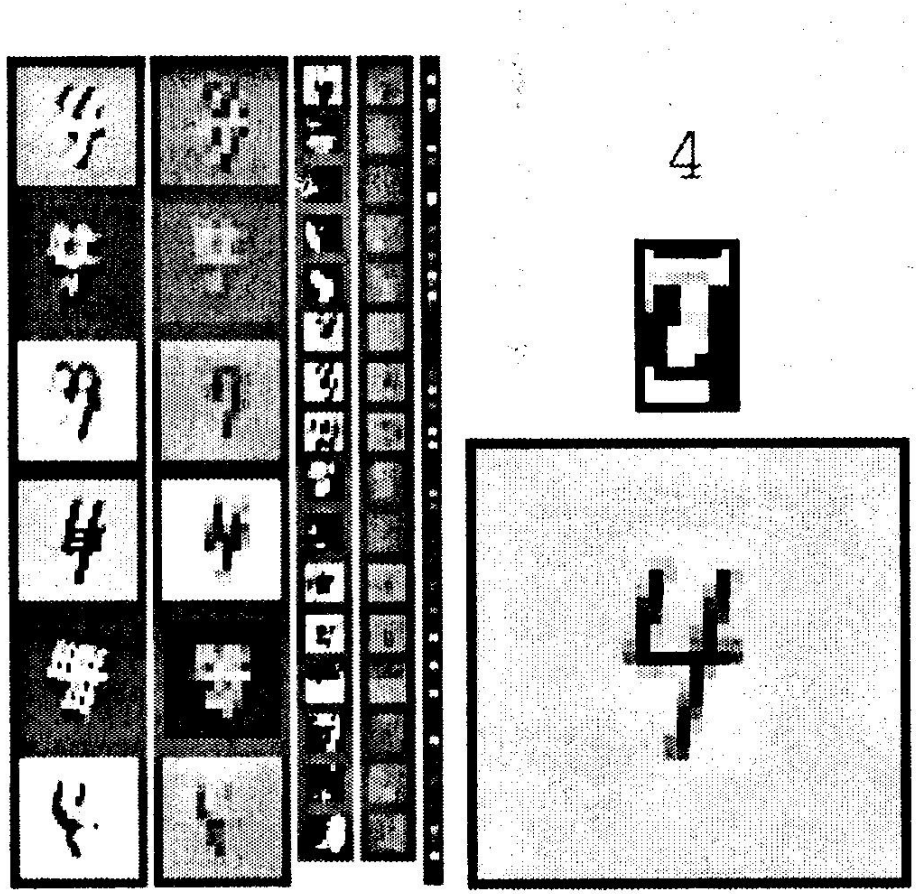
CONV-**POOL**-**CONV**-**POOL**-FC-FC-FC

LeNet-5: recognition results



Source of Images: Adapted from *Prof. Geoffrey E. Hinton's course material*

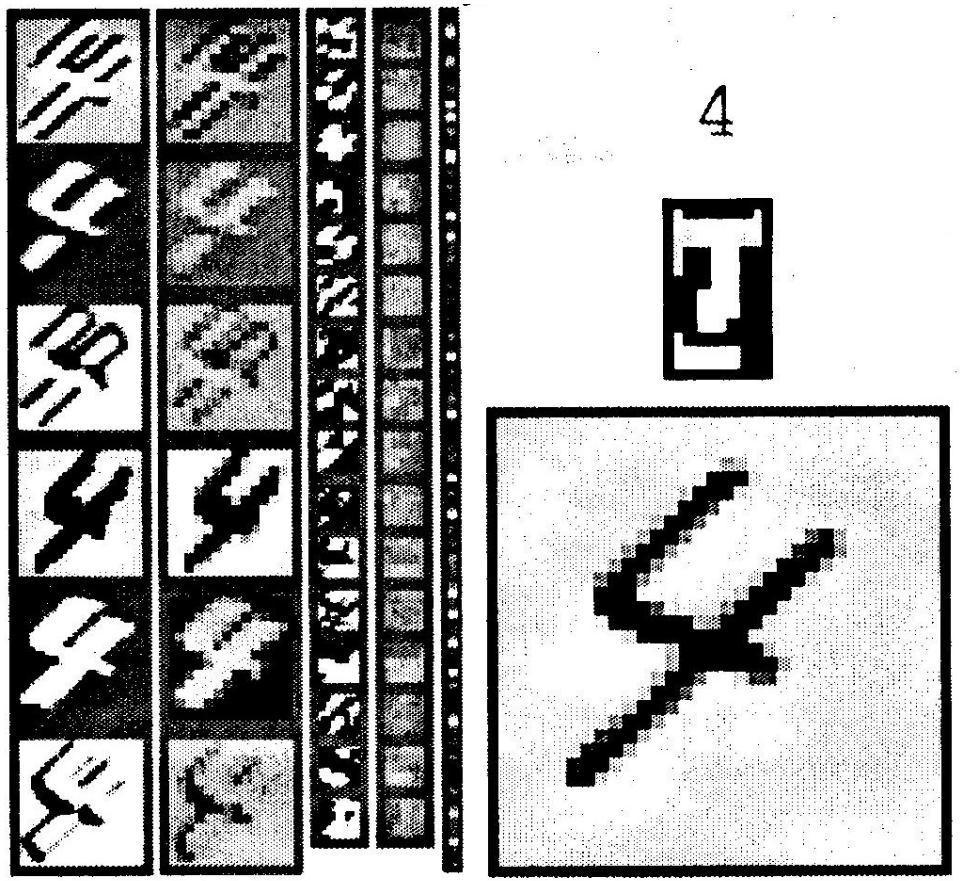
LeNet-5: recognition results



Input size scaling
Stable representation at FC layer
Output is still correct

Source of Images: Adapted from *Prof. Geoffrey E. Hinton's course material*

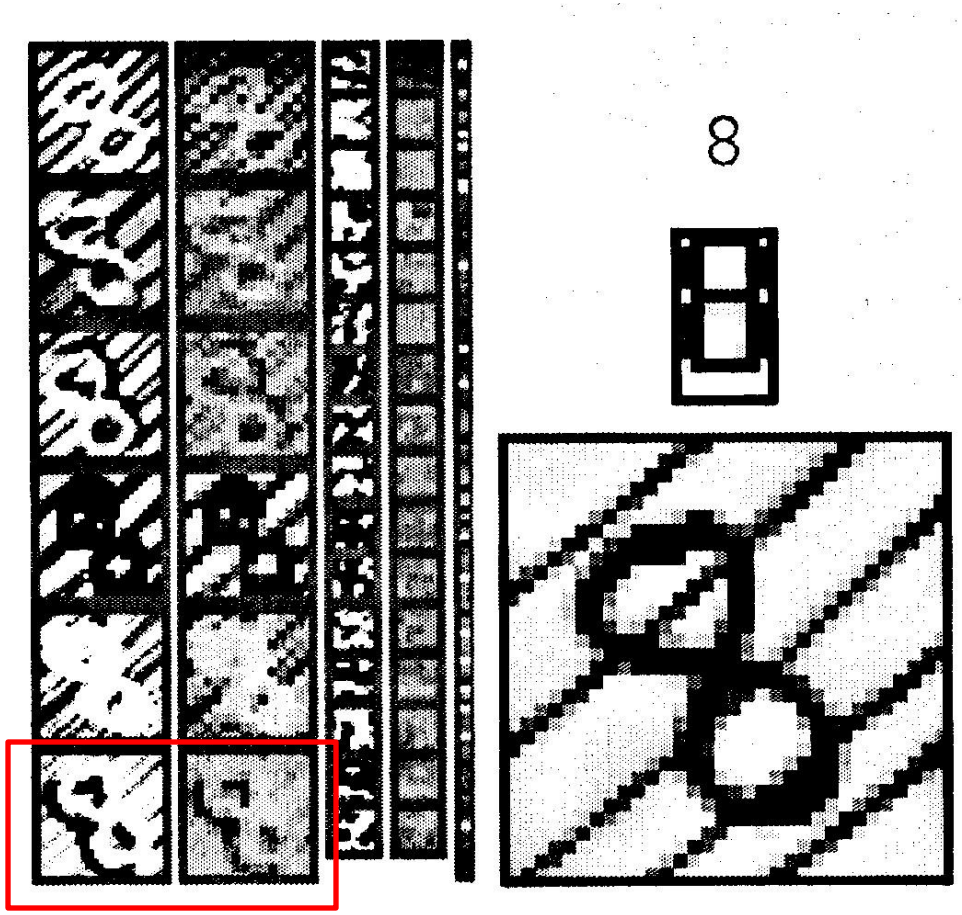
LeNet-5: recognition results



Slight rotation
Stable representation at FC layer
Output is still correct

Source of Images: Adapted from *Prof. Geoffrey E. Hinton's course material*

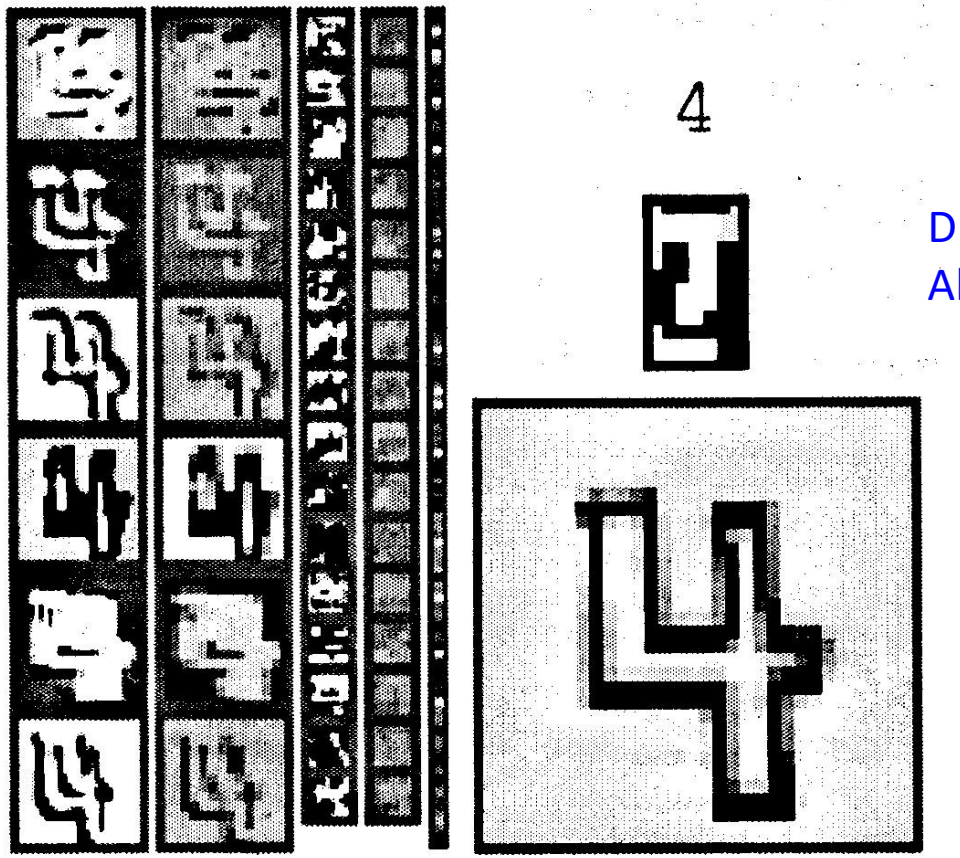
LeNet-5: recognition results



Noise in input
Cleaned up by part of the filters
Noise reduced through pooing layers
Output is still correct

Source of Images: Adapted from *Prof. Geoffrey E. Hinton's course material*

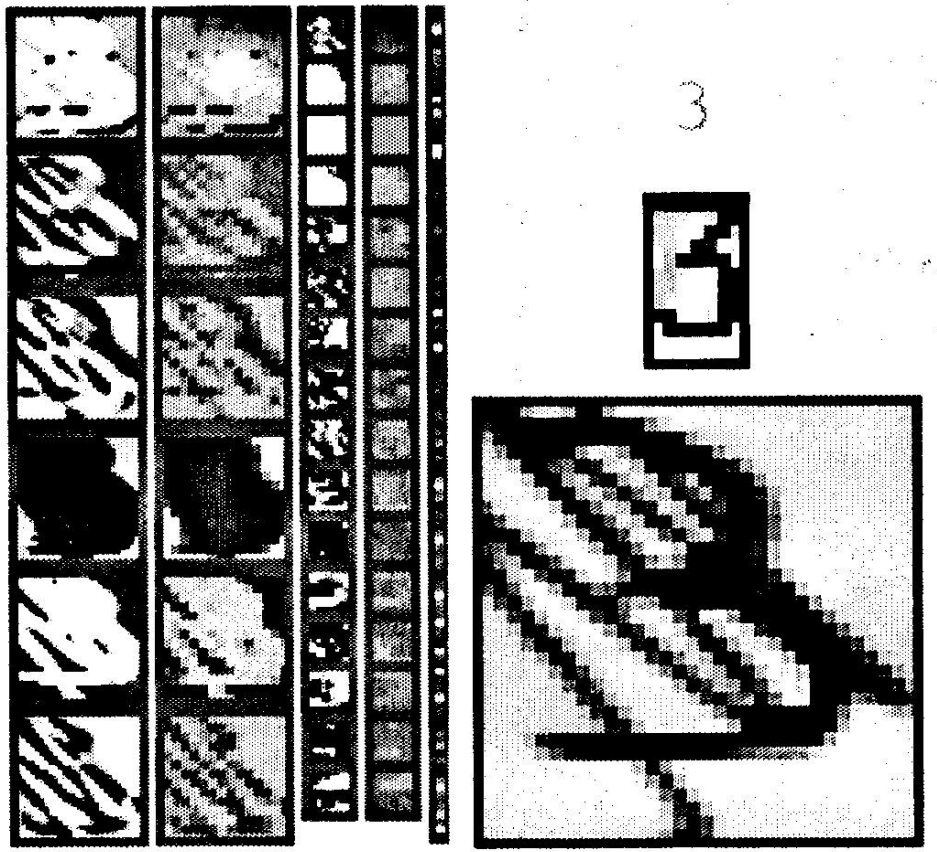
LeNet-5: recognition results



Different input style
Also classified correctly

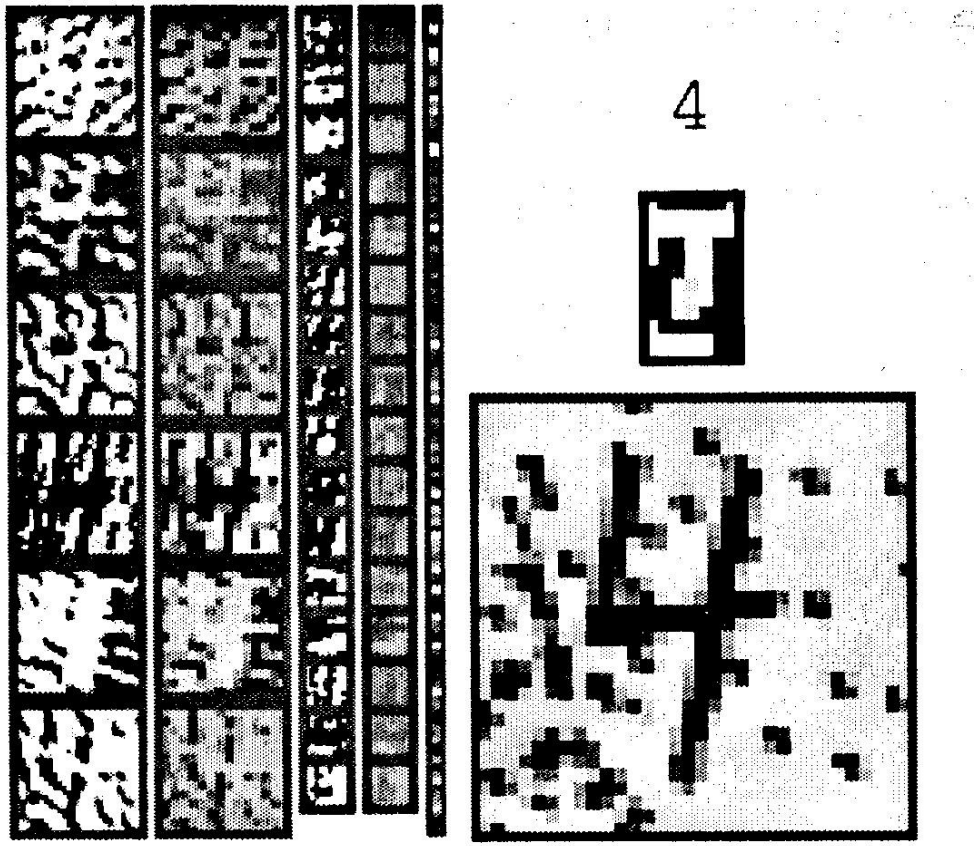
Source of Images: Adapted from *Prof. Geoffrey E. Hinton's course material*

LeNet-5: recognition results



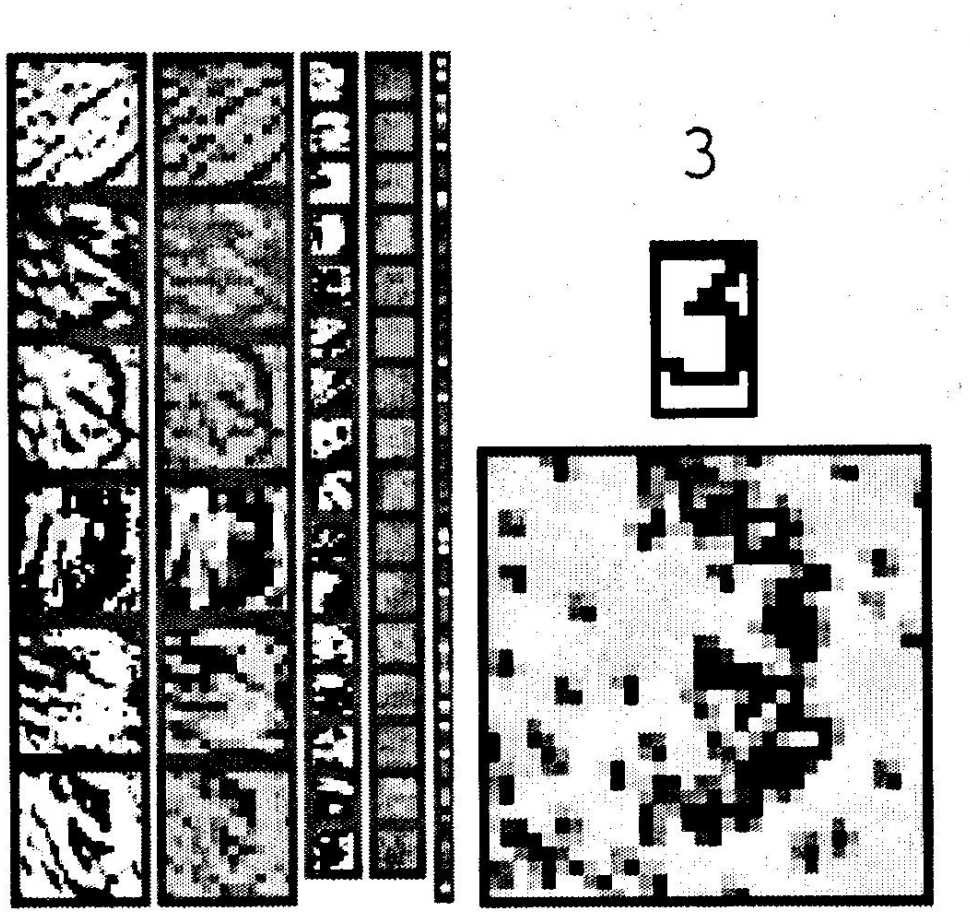
Source of Images: Adapted from *Prof. Geoffrey E. Hinton's course material*

LeNet-5: recognition results



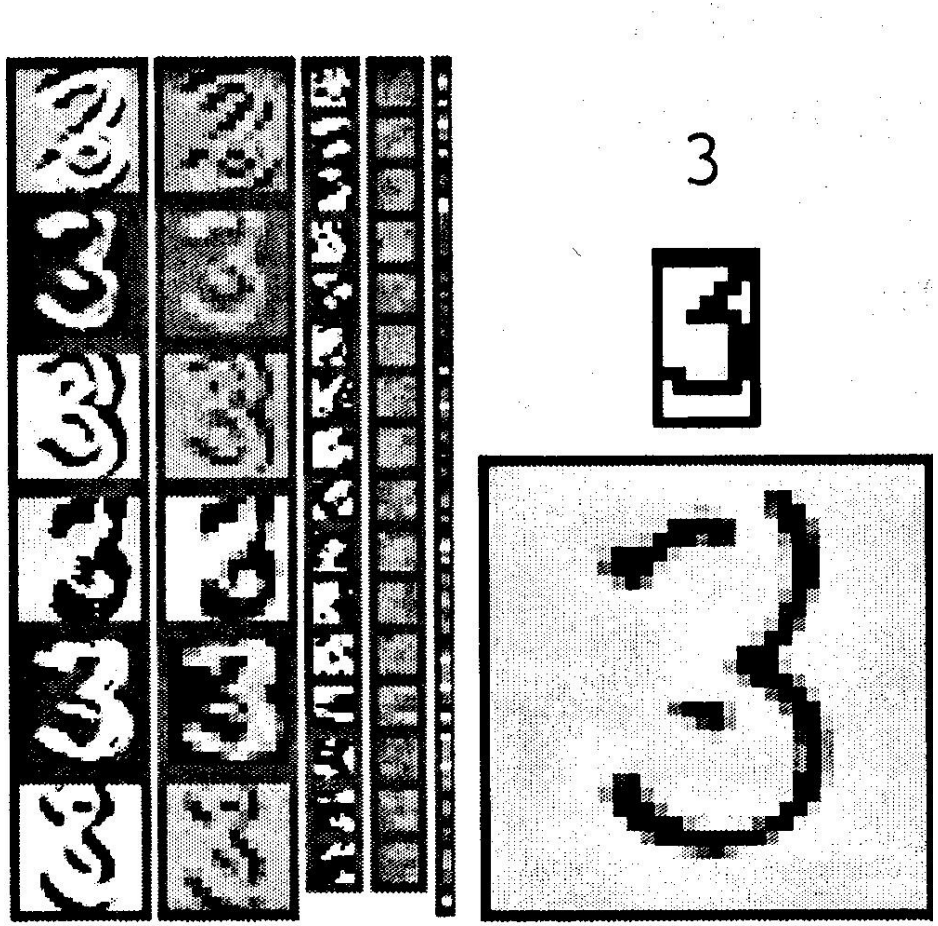
Source of Images: Adapted from *Prof. Geoffrey E. Hinton's course material*

LeNet-5: recognition results



Source of Images: Adapted from *Prof. Geoffrey E. Hinton's course material*

LeNet-5: recognition results



Source of Images: Adapted from *Prof. Geoffrey E. Hinton's course material*

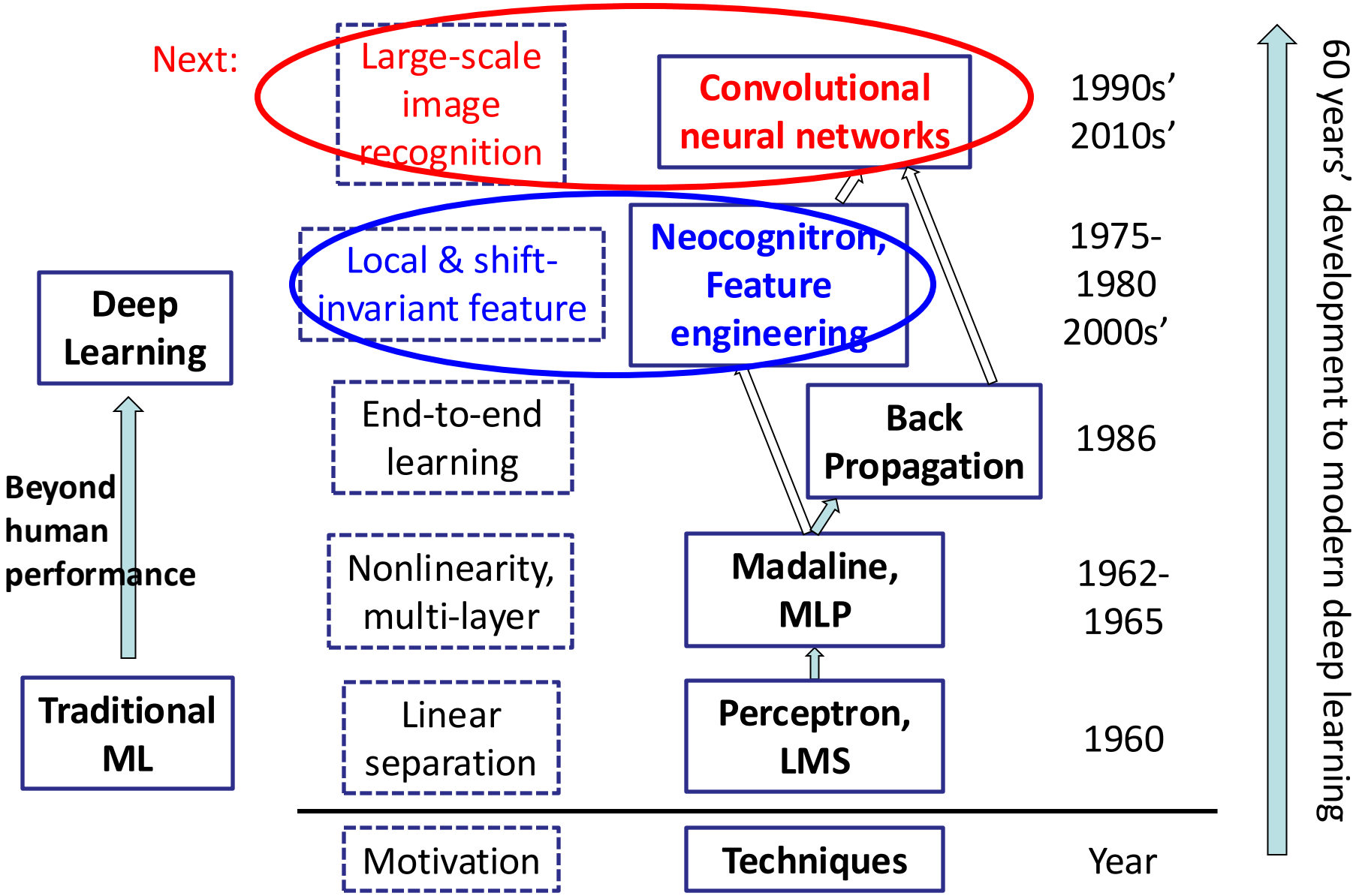
Is CNN perfect?

- Design of CNN resolves the problems of local feature and shift invariance, but there are still issues left, especially for **harder learning tasks**
- Other invariance: scaling, rotation etc.
  - Partially resolved by augmenting the training set
  - Ongoing studies on new kernel design/learning scheme
- Stability/Generalizability
  - Partially occluded objects, noisy input (adversarial attack)
- Interpretability/context awareness etc.
  - Better theoretical understanding needed

These topics are still active research fields.

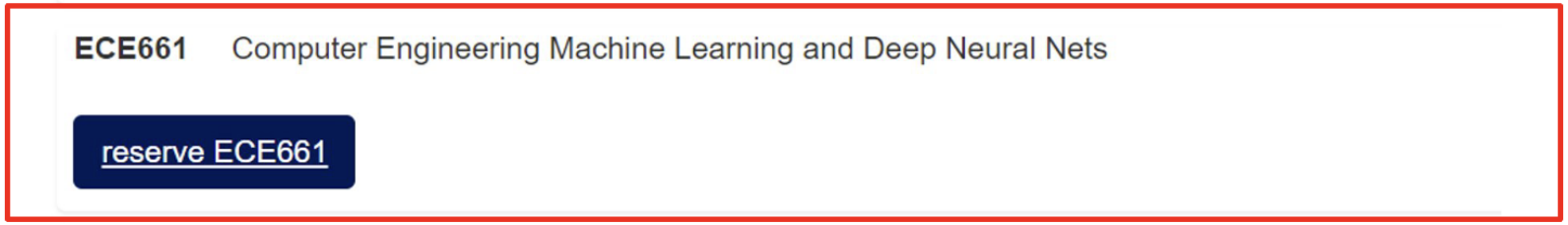
We will come across some of these issues in later lectures of this course

In this lecture, we learned:



Server Setup

- Google Colab is the top recommendation. You can refer to Google Colaboratory https://colab.research.google.com/ for free computing resources. With a google account, you are good to go. The Google Colab uses PyTorch 2.4 and will consistently upgrade the packages. More details can be found in the tutorial under Files/Tutorials on Canvas.
- You may also refer to https://cmgr.oit.duke.edu/containers to see the amount of available resources. Due to the limited availability, access is not guaranteed, so you may start your work early on our Jupyter Lab server. The environment is similar to Google Colab environment, except for the older PyTorch version.



- We will keep you updated on alternative available resources to complete the coursework on Slack channel.