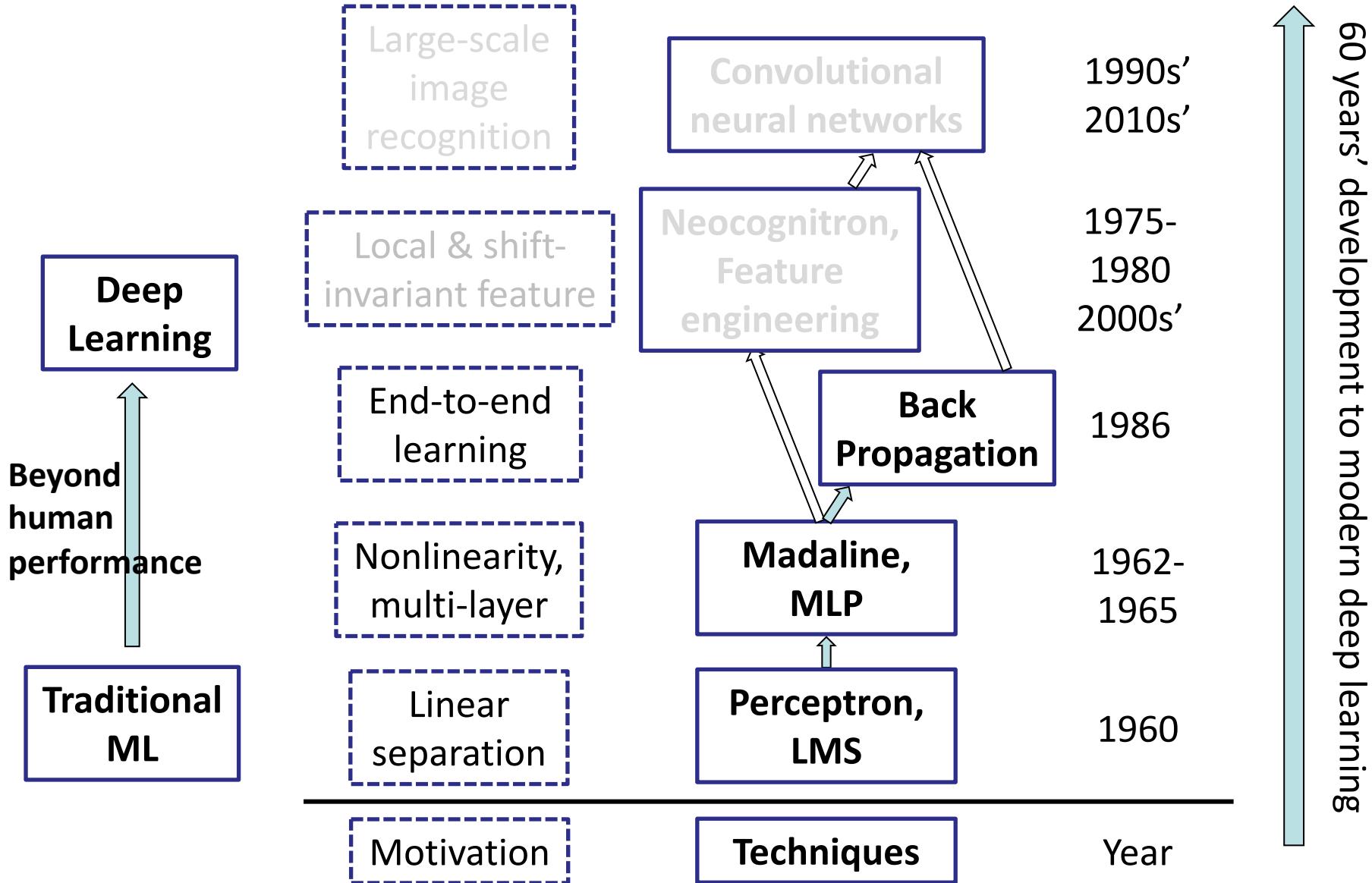




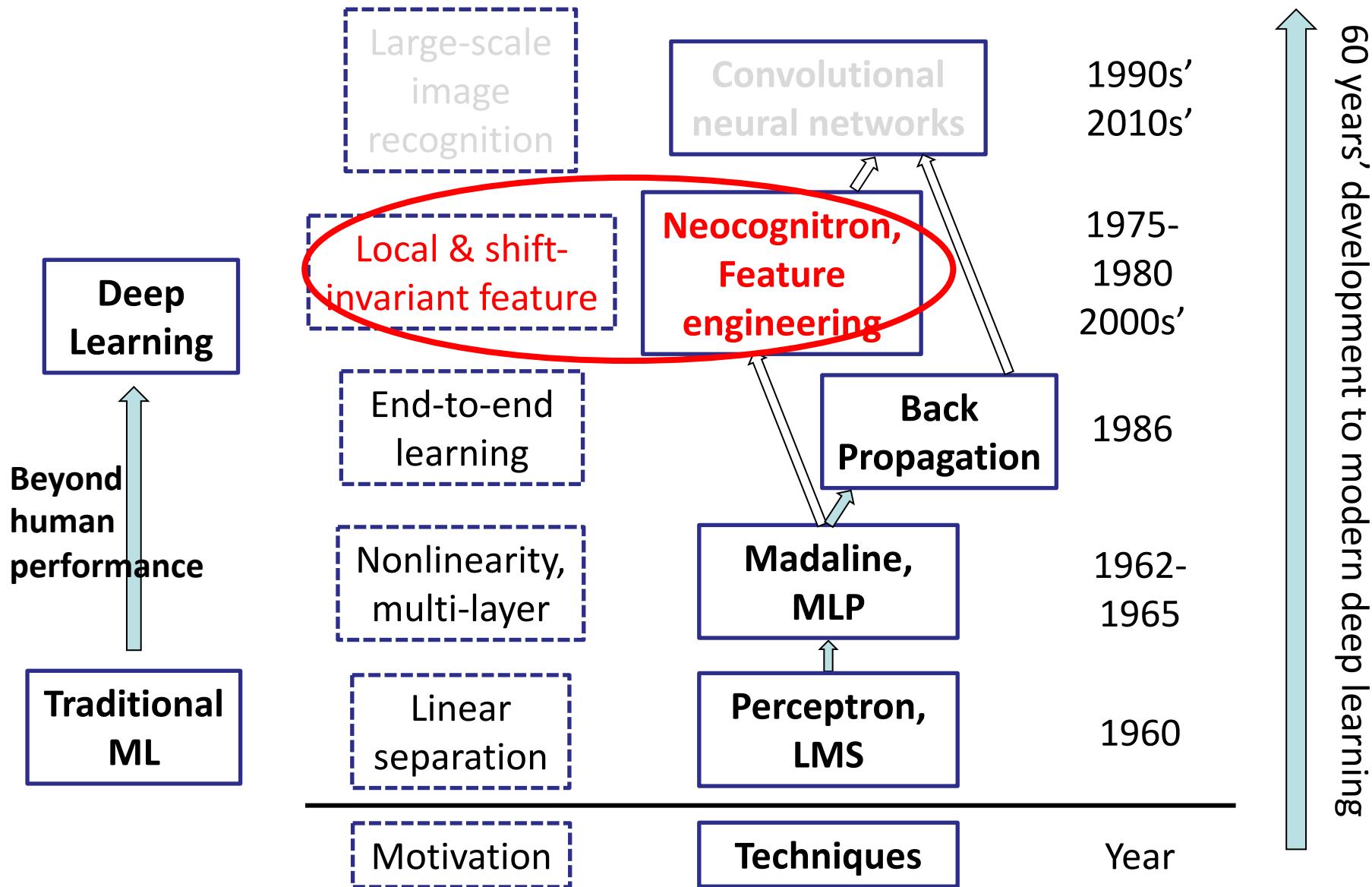
ECE 661 COMP ENG ML & DEEP NEURAL NETS

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

Previously



This lecture



Outline

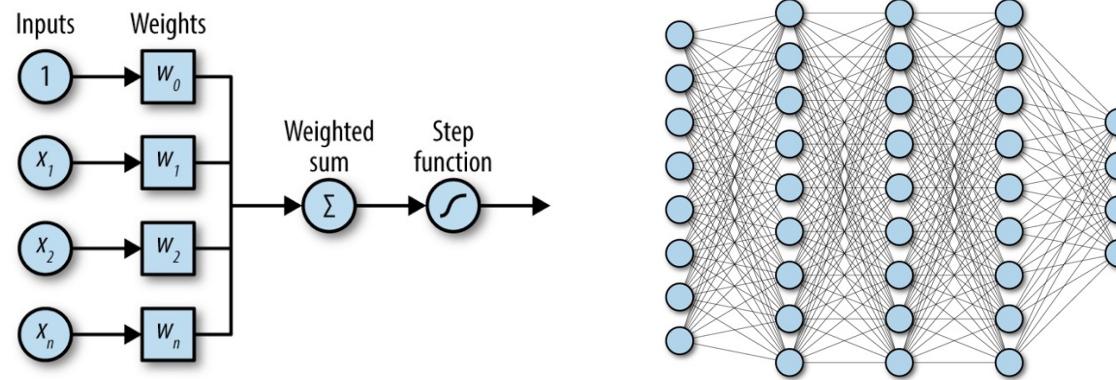
- Lecture 2: NN and Back Propagation
- Lecture 3: Emergence of convolution
 - Useful features for image classification
 - Convolution and feature engineering
 - Hierarchical structure
 - Putting things together: CNN
- Lecture 4: Building a CNN model

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

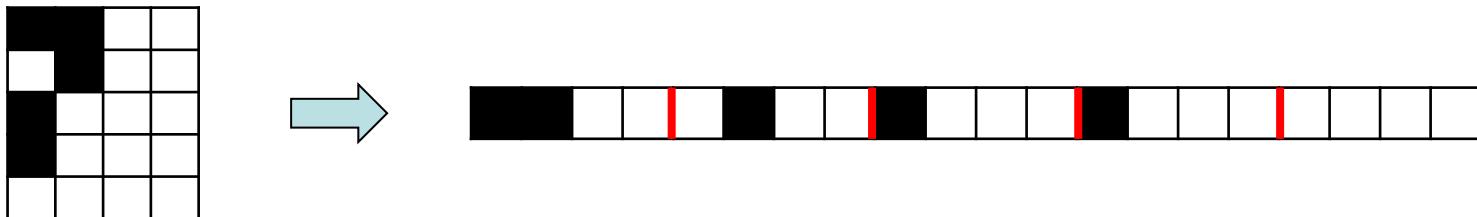
- Fully connected (FC) networks
 - Takes an $N \times 1$ vector as the input, a scalar or a vector as the output
 - **Universal approximation theorem:** A 2-layer arbitrary width FC can fit any smooth function that maps one vector to another



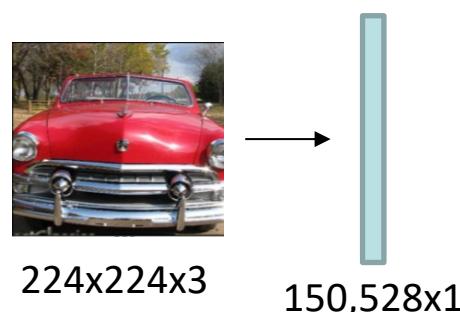
- FC models are theoretically powerful, but has practical limitations
 - What about having 2-D matrix as the input, like an image?
 - Reshape it into a vector? Let's give a try.

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

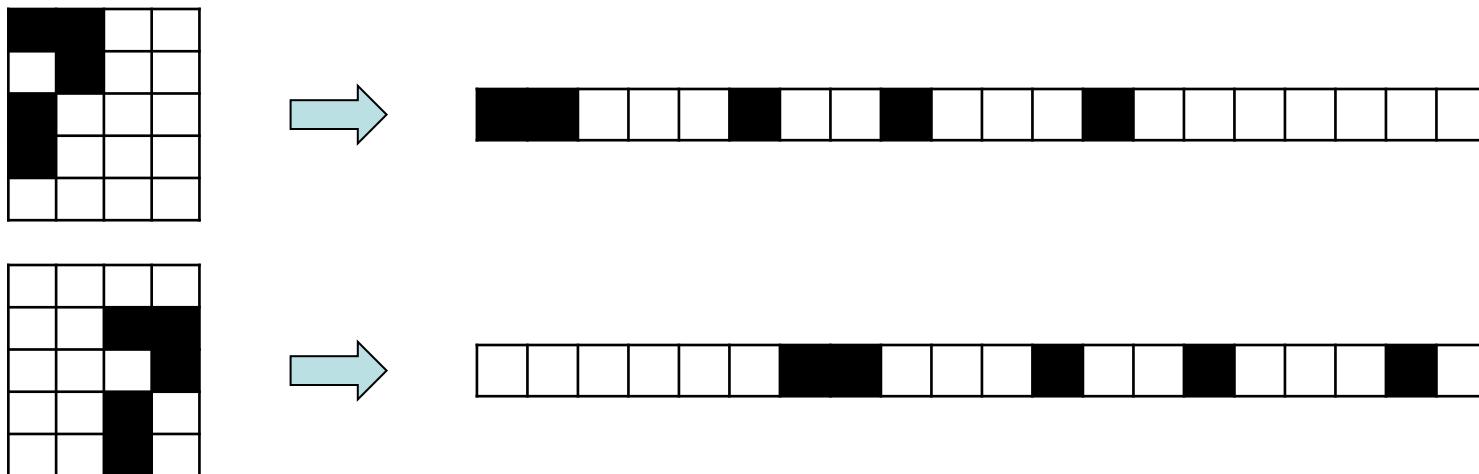
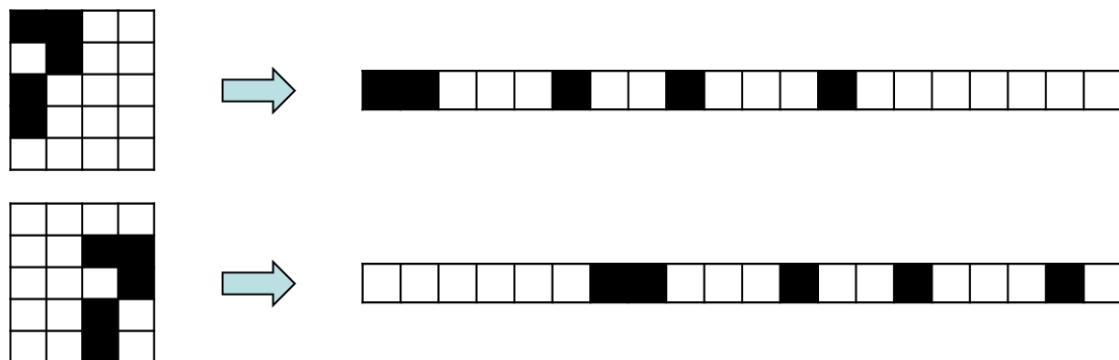
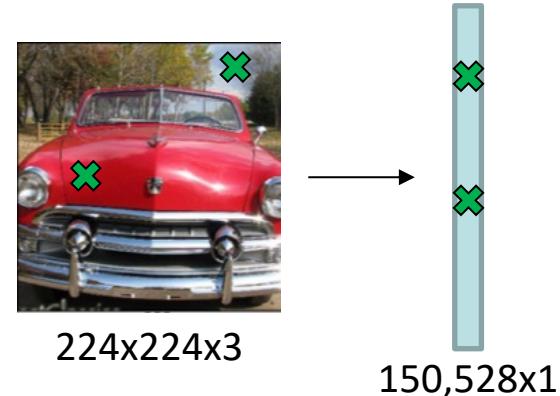


Image features that FC hardly captures

- 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



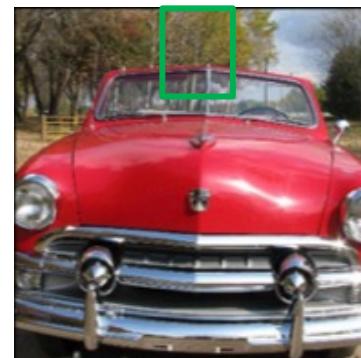
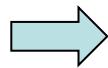
Local, shift-invariant
features are important
in image processing

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**

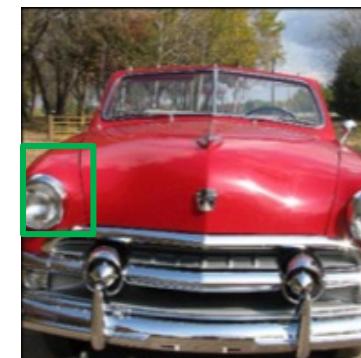


Not a headlight



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

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

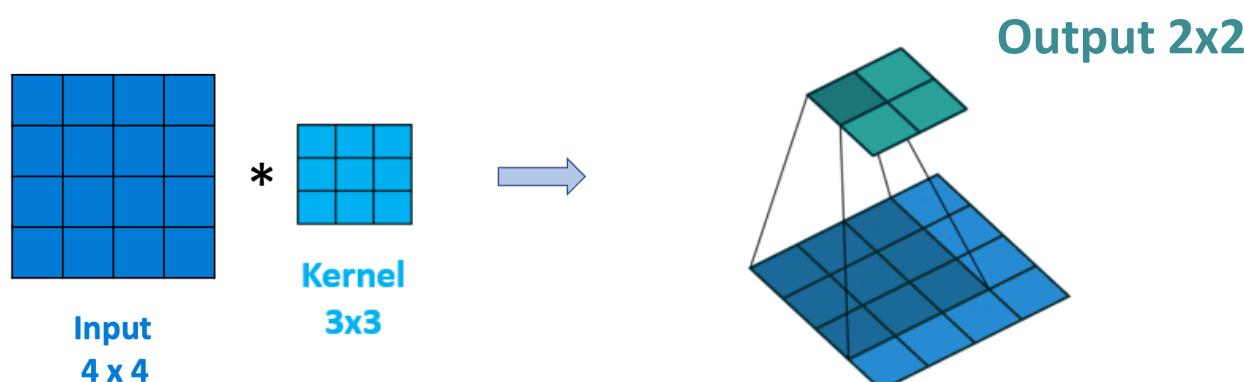
- Example

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

Convolution operation

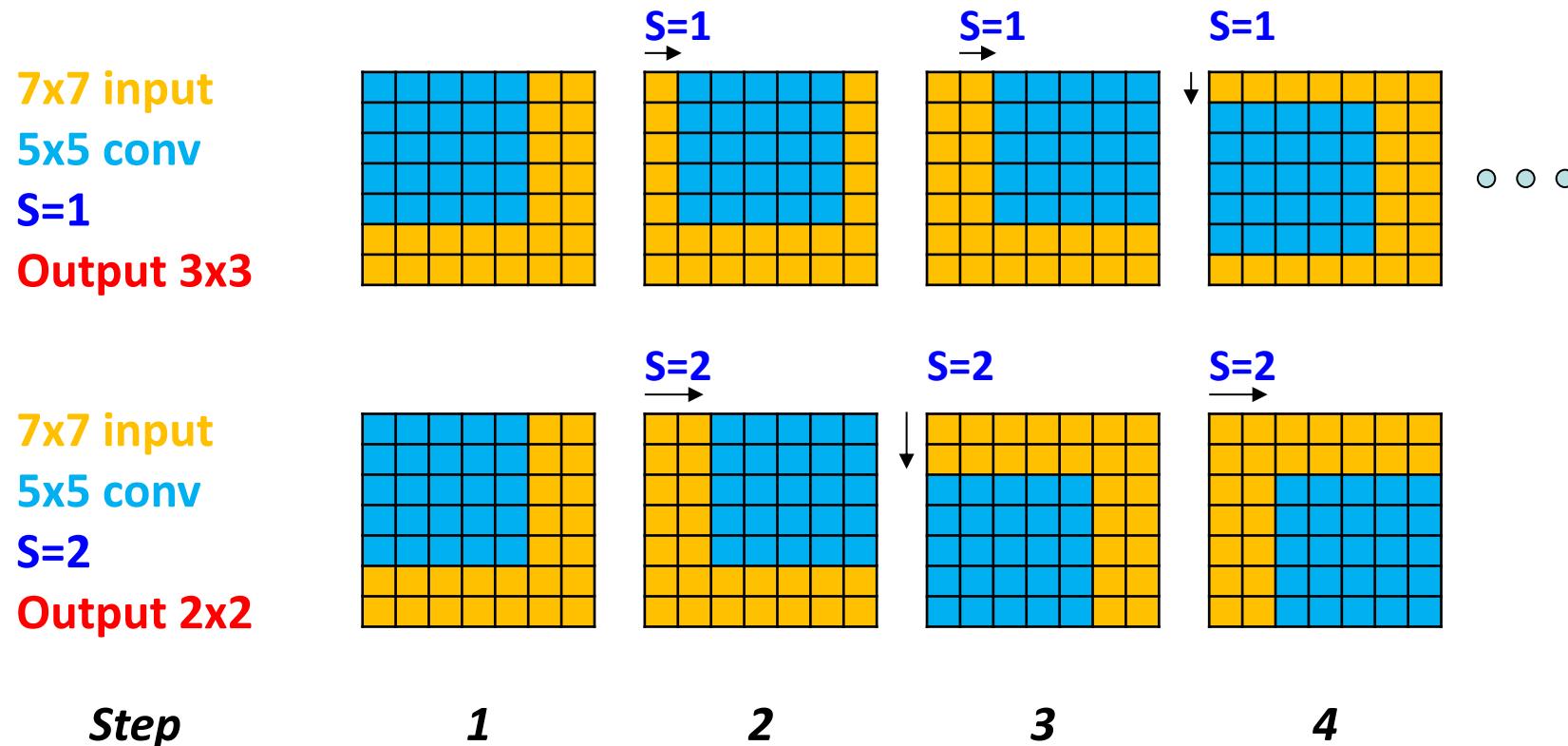
- Shifting and applying the linear filter
 - Start with applying the kernel at the top-left corner, the output goes to the top-left pixel of the **output feature map**
 - Shift kernel over the image to get other output pixels
 - Linear operation (weighted sum) is applied on the input region overlapped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

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



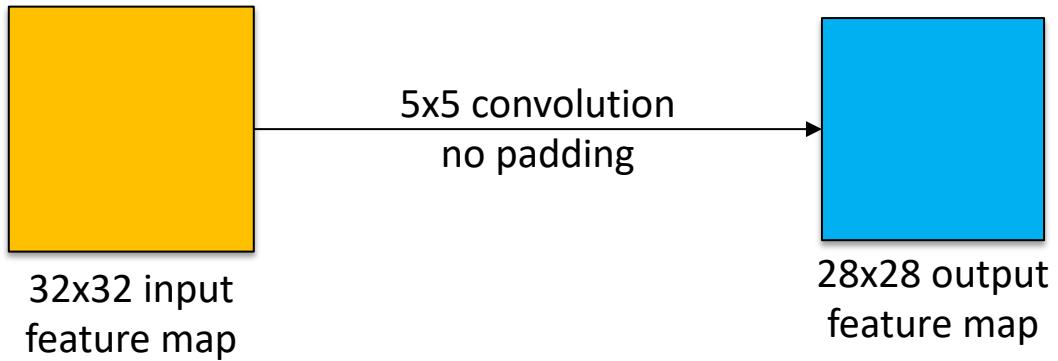
How filter moves: strides

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

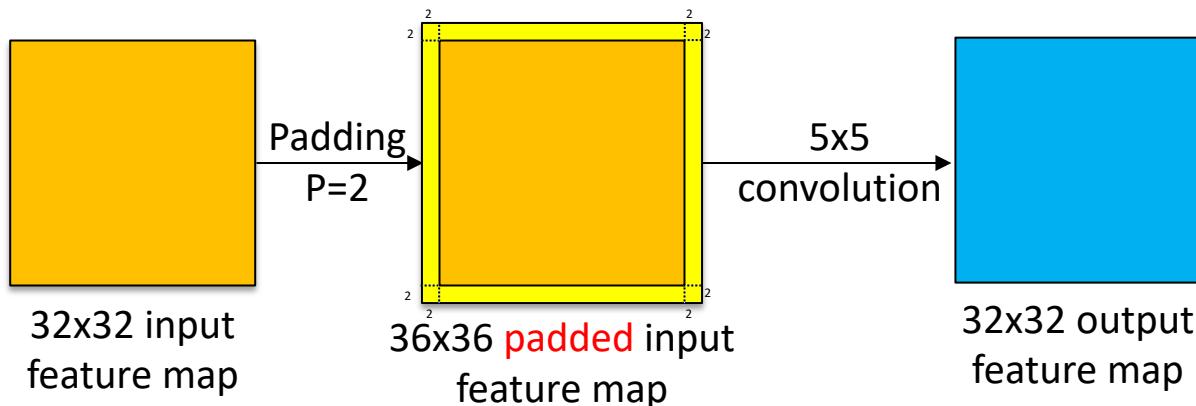


Prevent shape change: padding

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



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 $H_1 \times W_1$.
- Convolution configuration:
 - Convolution kernel size K
 - Stride for convolution S
 - Padding for each border P
- The output feature map has the following shape $H_2 \times W_2$, where

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

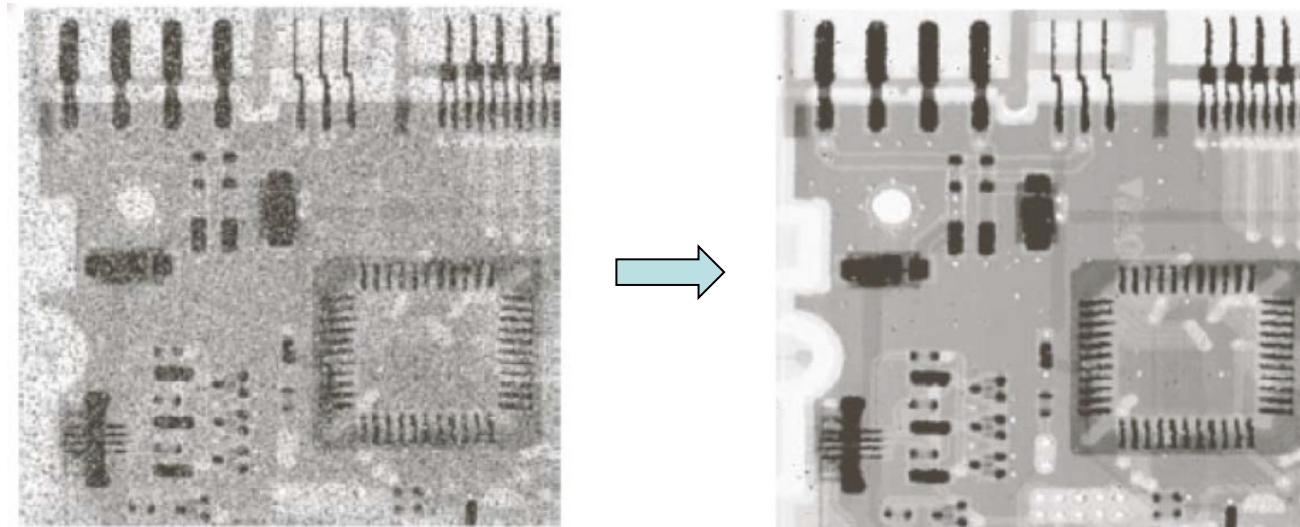
$$H_2 = \left\lfloor \frac{H_1 - K + 2P}{S} \right\rfloor + 1$$

- 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)

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



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

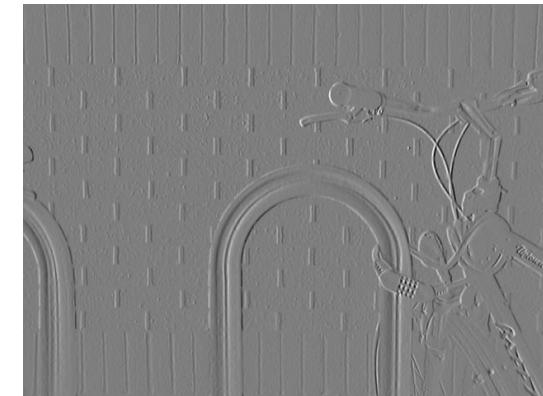
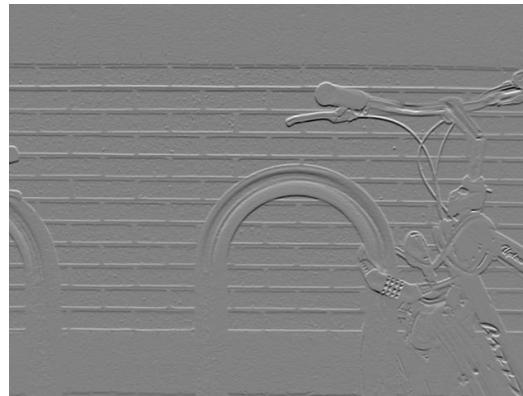
Horizontal edge

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



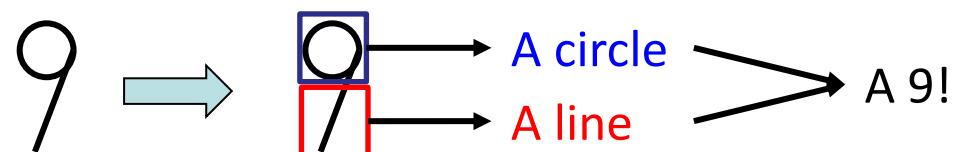
Vertical edge

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

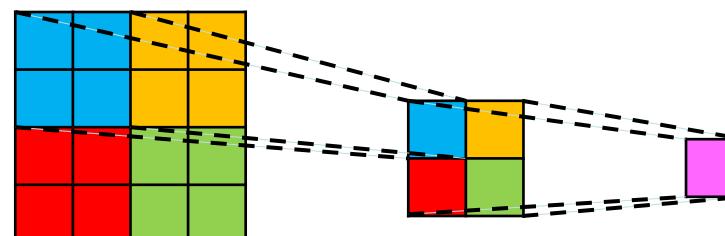


What's still missing

- 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)



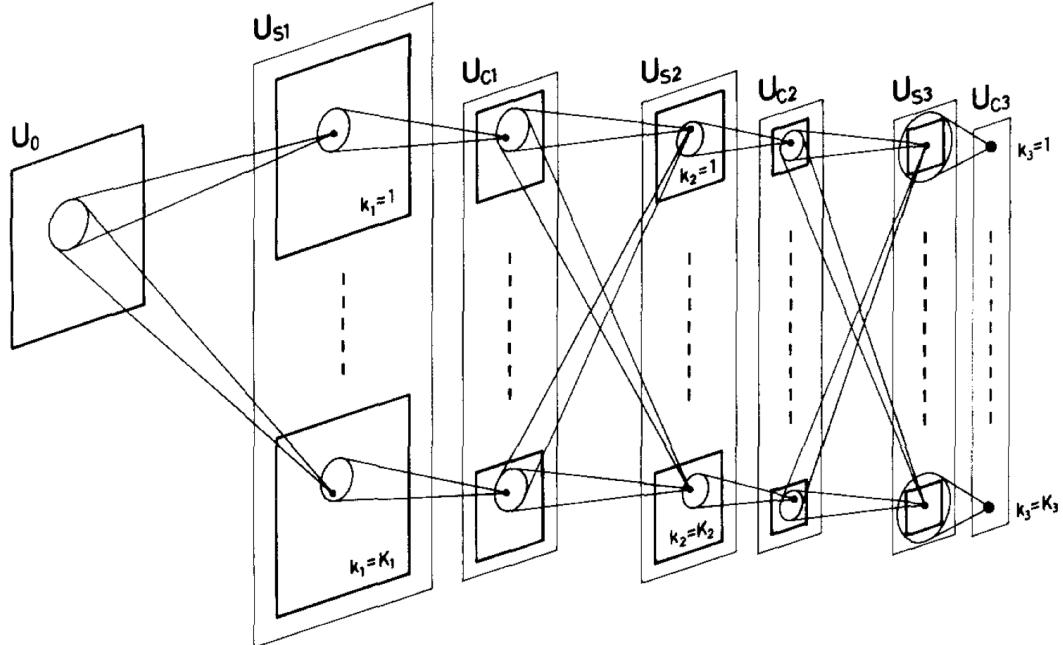
- Higher-layer filters take in the information from a larger range of input image pixels



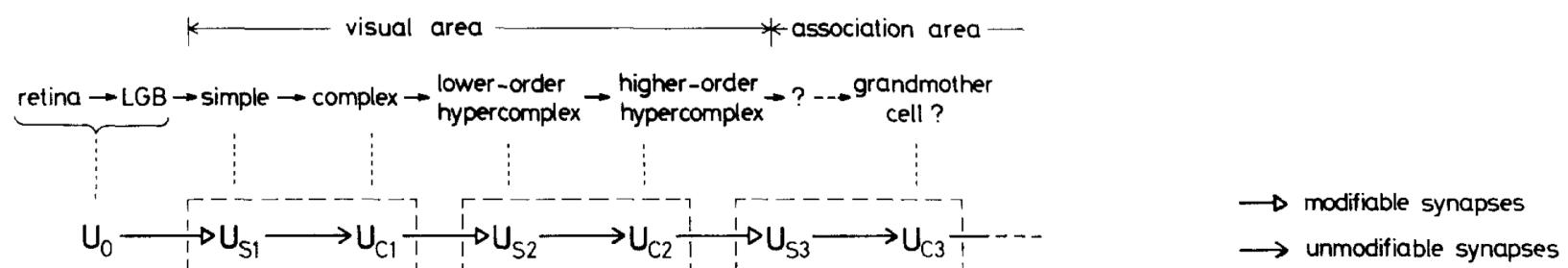
Outline

- 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

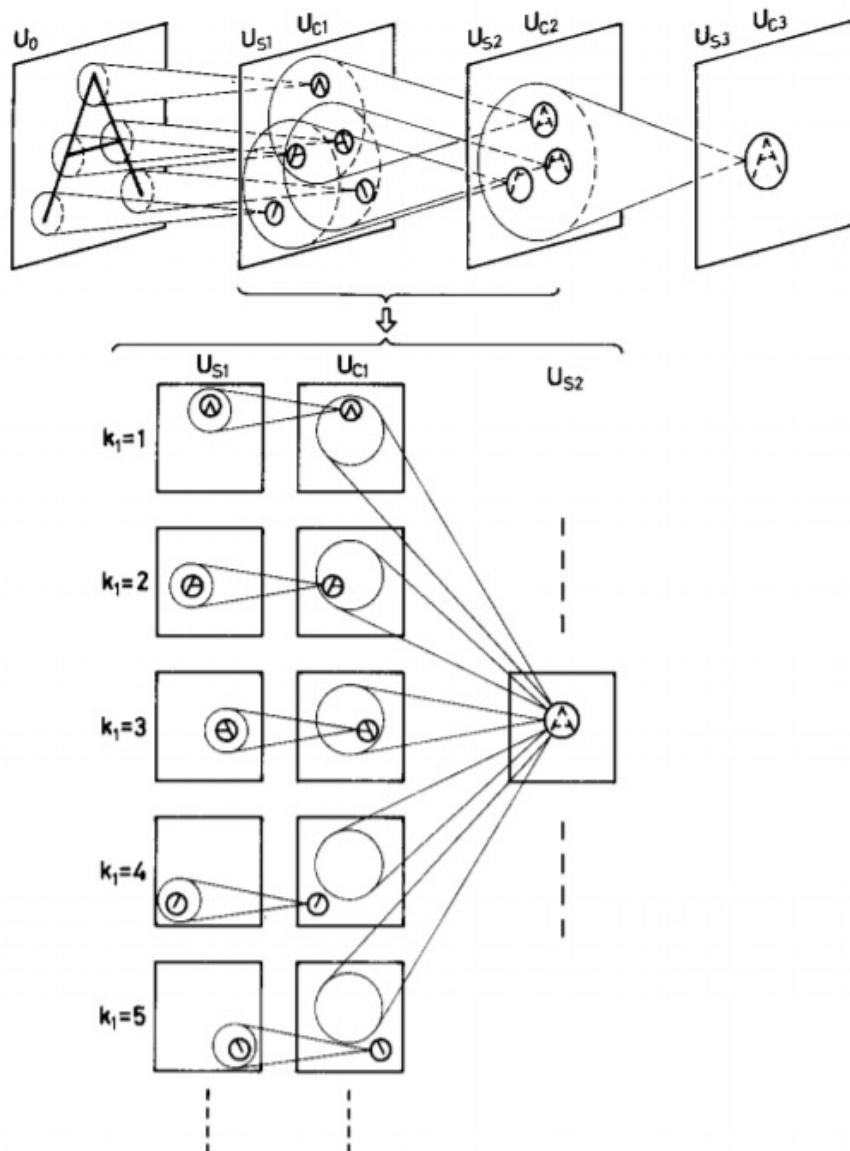


- Fukushima, Kunihiko. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position." (1980)
- S-C-S-C-S-C structure
 - Simple cells: cells with trainable parameters -> **convolution**
 - Complex cells: Perform down-sampling which extracts 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

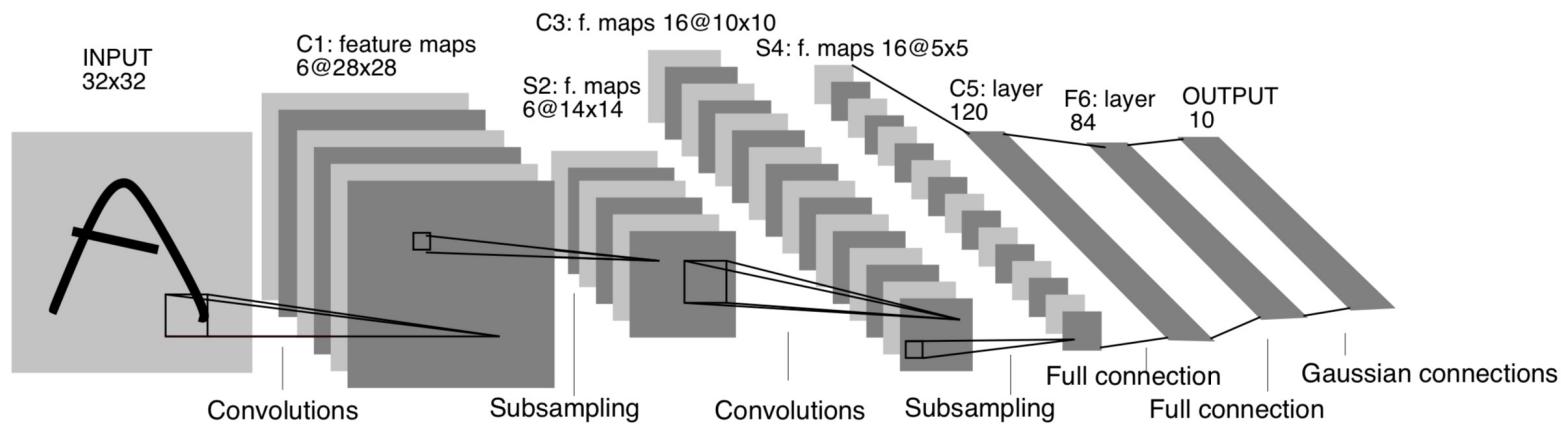
Components of a modern CNN model

- 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

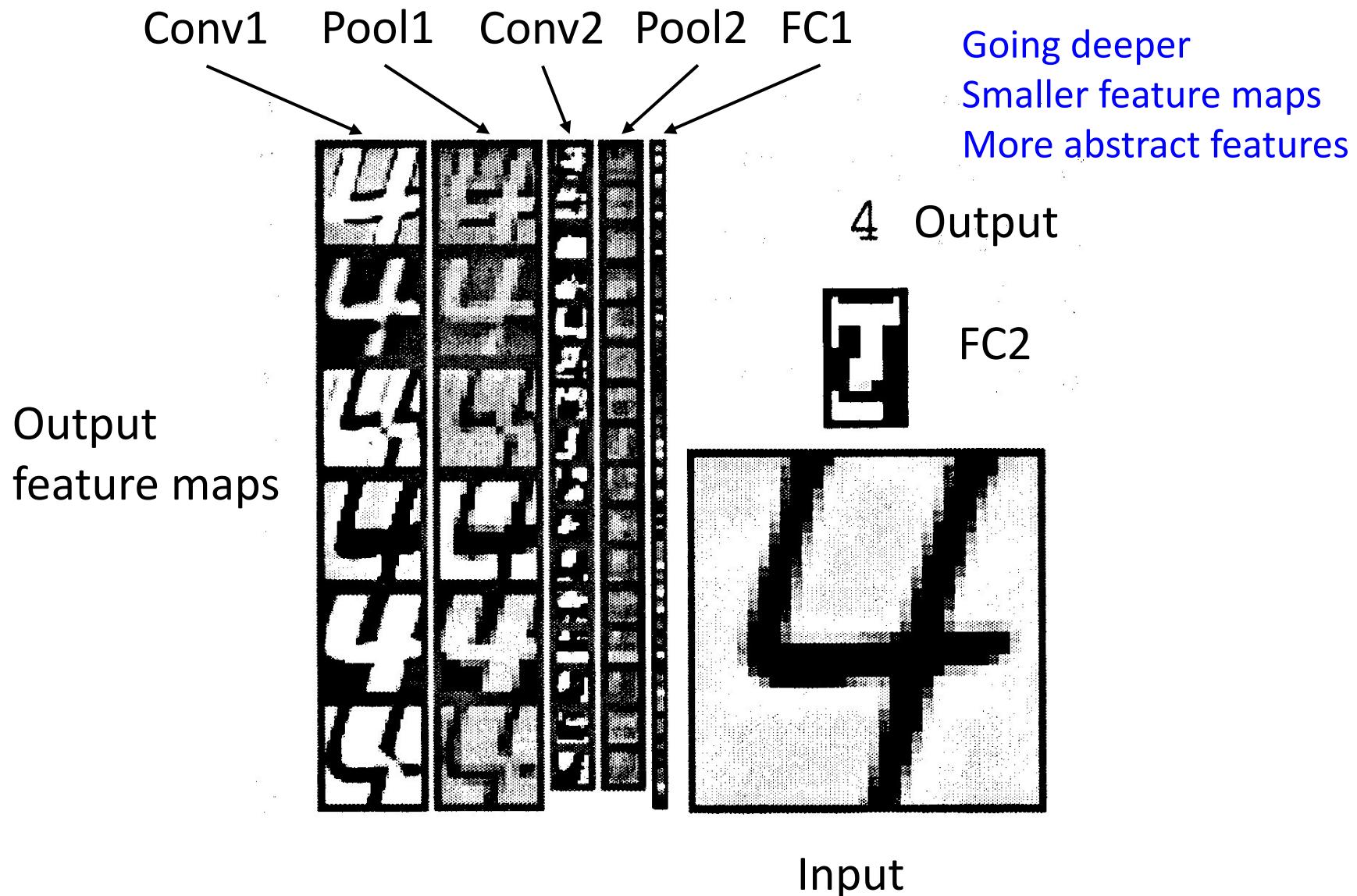
- Case-study: LeNet-5 for MNIST handwritten digits classification
LeCun, Yann et al. "Gradient-based learning applied to document recognition." (1998)



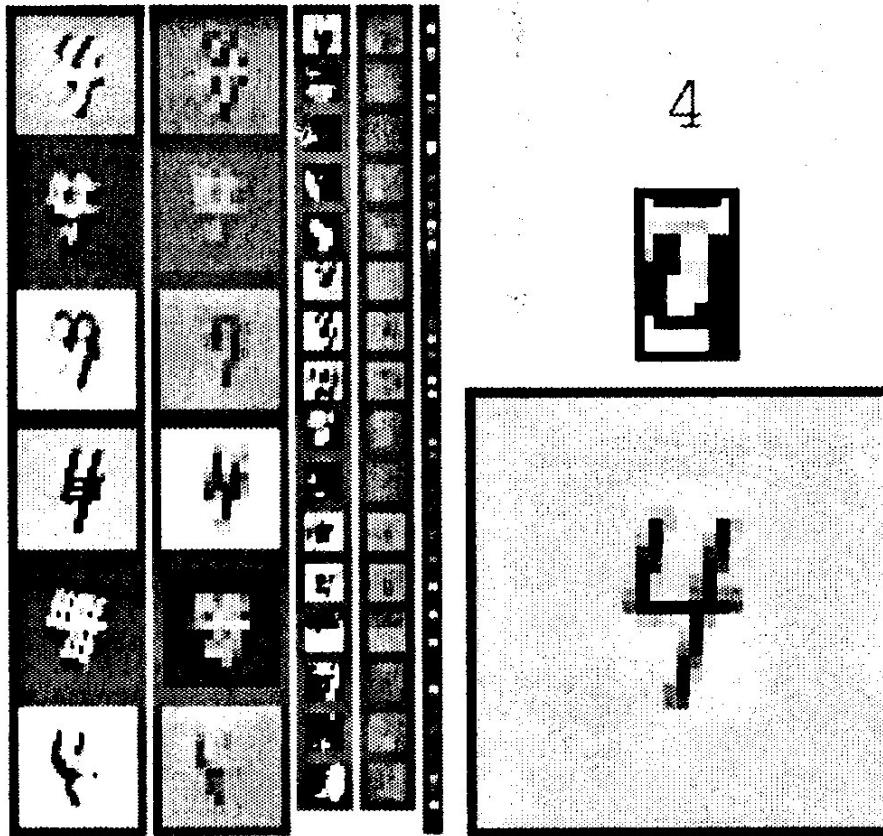
LeNet Structure:

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

LeNet-5: recognition results

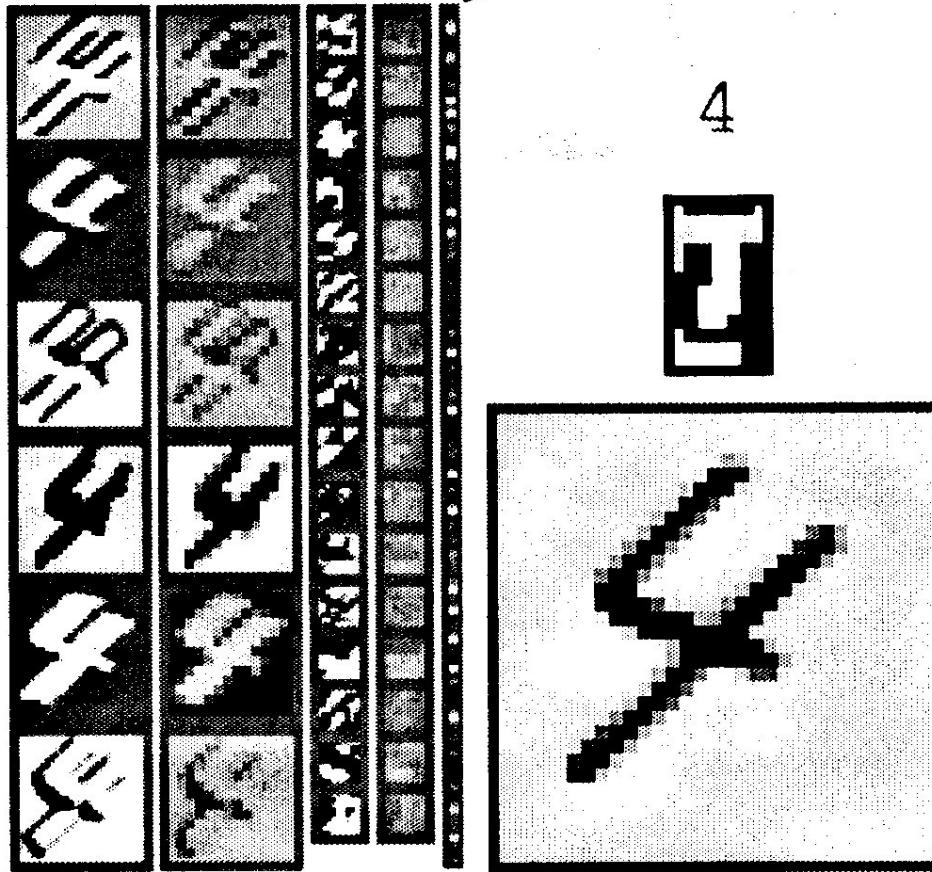


LeNet-5: recognition results



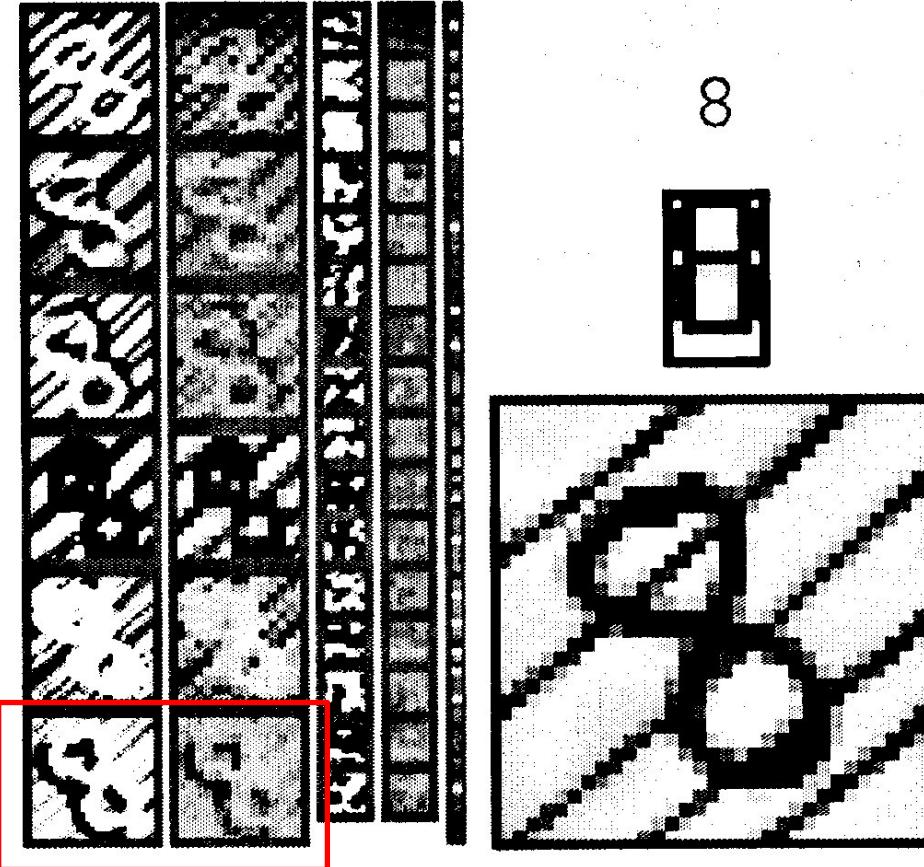
Input size scaling
Stable representation
at FC layer
Output is still correct

LeNet-5: recognition results



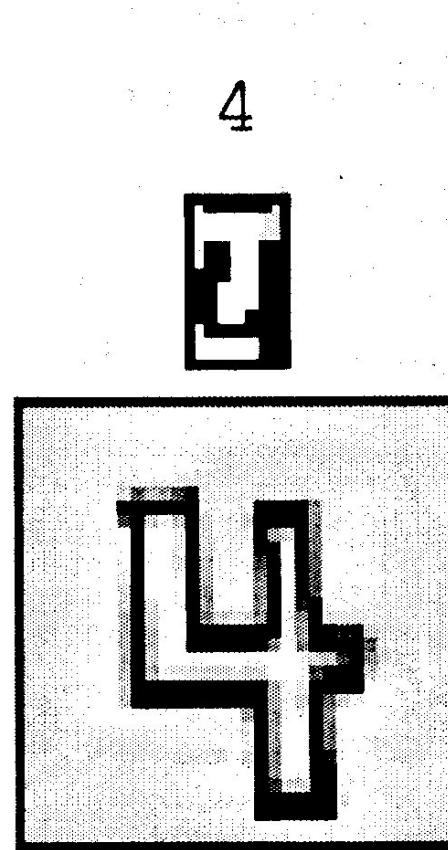
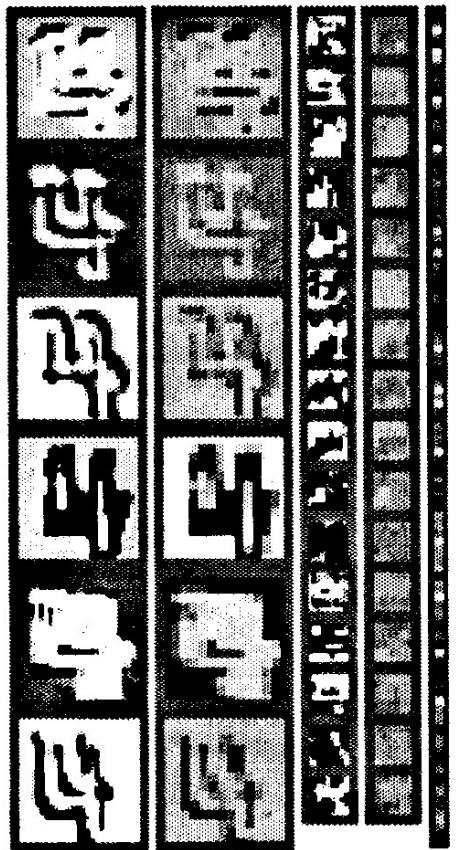
Slight rotation
Stable representation
at FC layer
Output is still correct

LeNet-5: recognition results



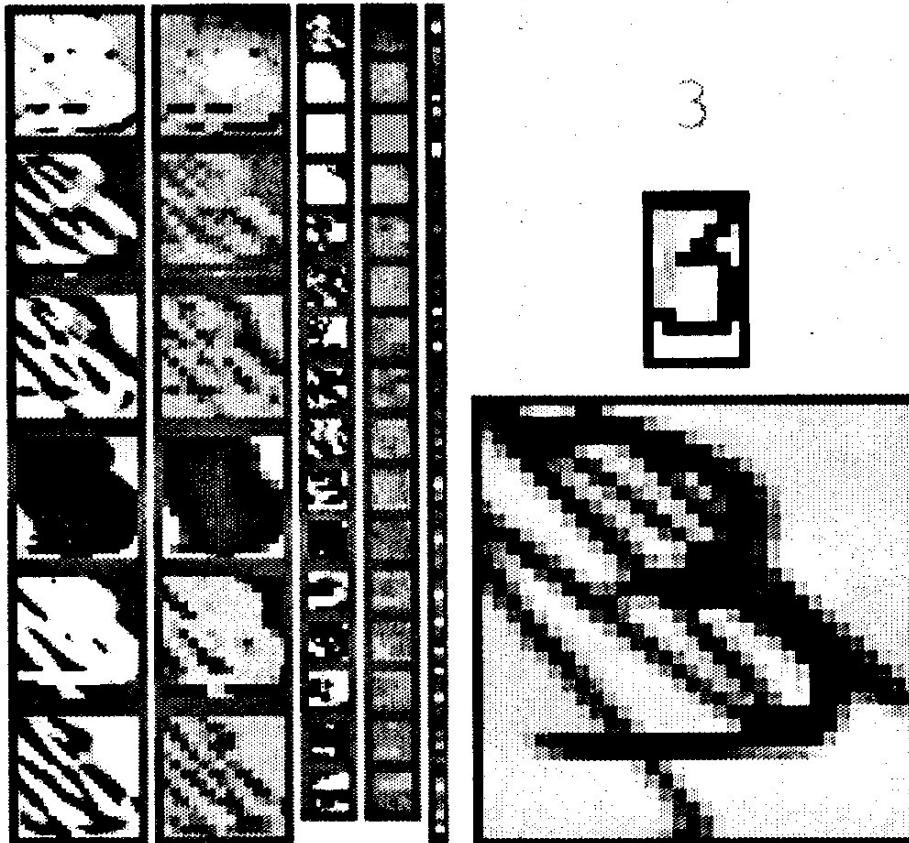
Noise in input
Cleaned up by part of
the filters
Noise reduced through
pooling layers
Output is still correct

LeNet-5: recognition results

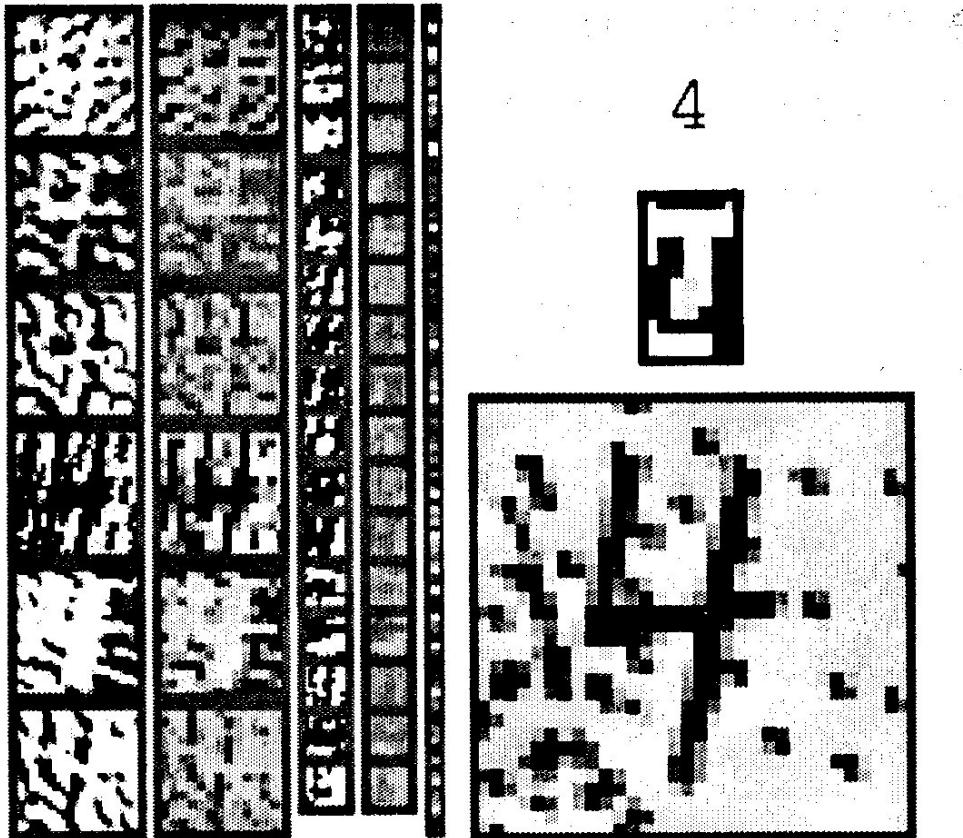


Different input style
Also classified correctly

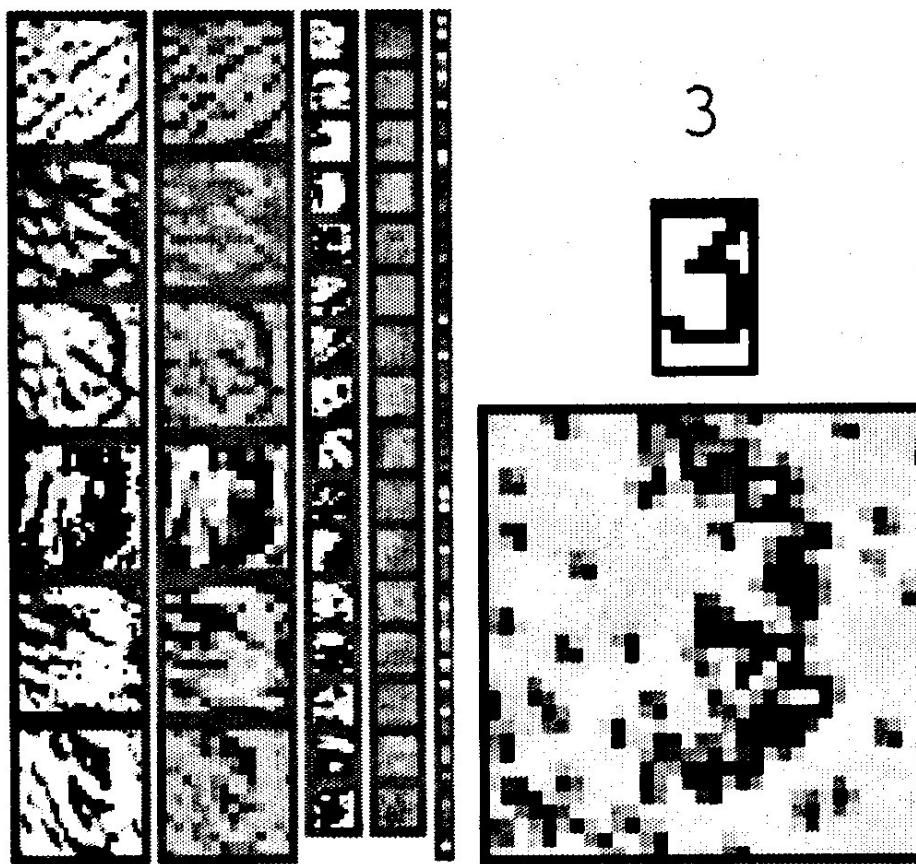
LeNet-5: recognition results



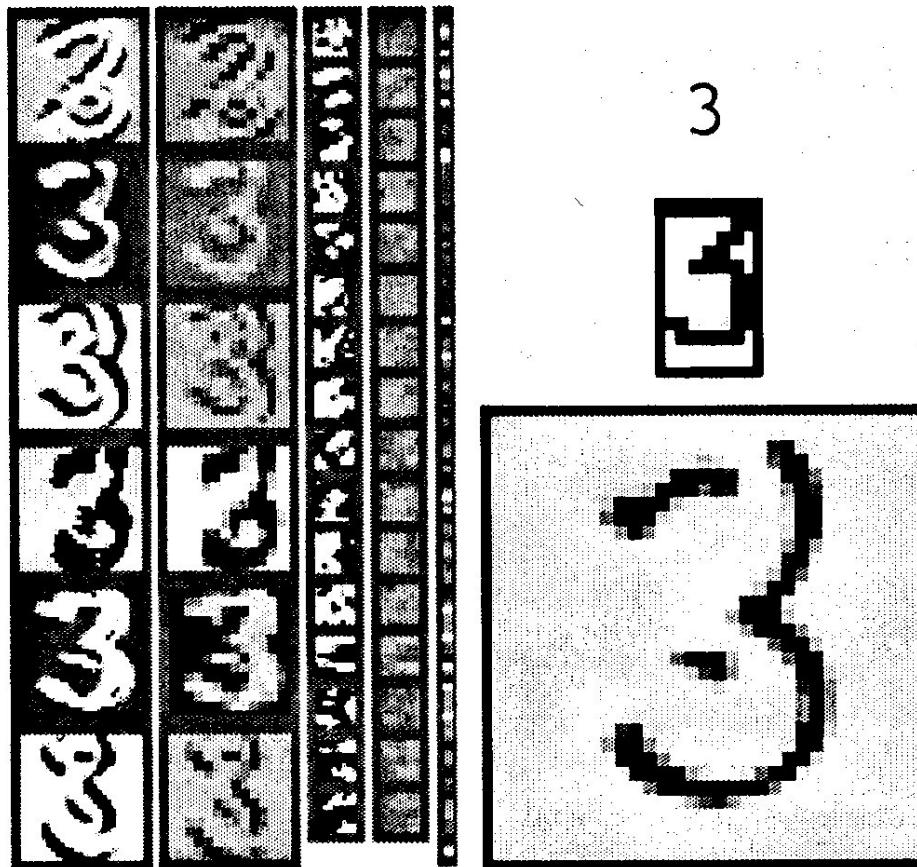
LeNet-5: recognition results



LeNet-5: recognition results



LeNet-5: recognition results

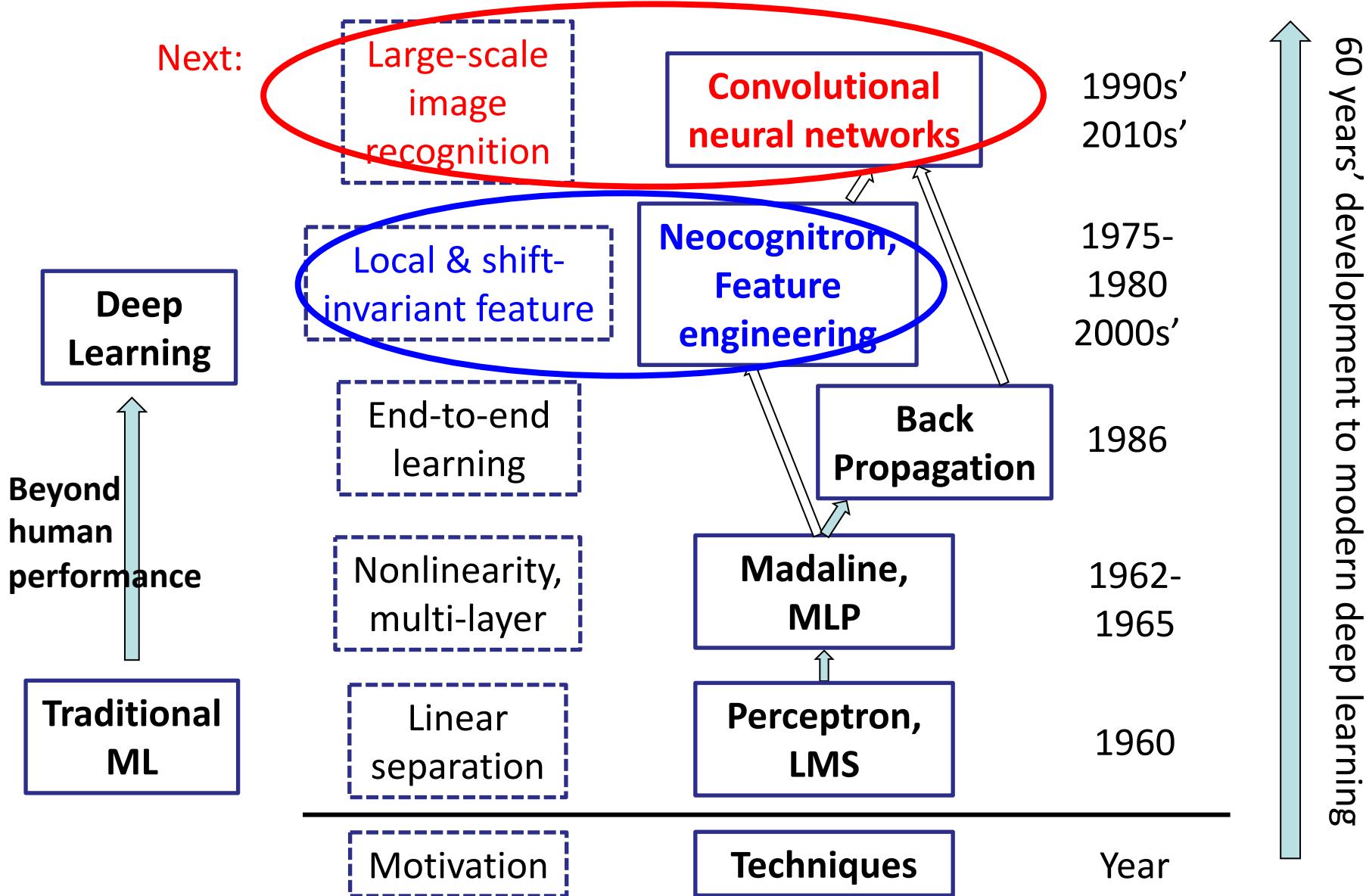


Is CNN perfect?

- 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:



Office Hours

- The office hours are planned as follows (Eastern time):
 - **Time changes compared to prior announcements.**
 - **Tuesday, 5:00 pm-6:30 pm**
 - Location: Wilkinson 126
 - Junyao, Mark
 - **Thursday, 3:00 pm-4:30 pm**
 - Location: Wilkinson 126
 - Haoxuan, Martin
- You may join and leave anytime during the meeting period.
- If you specific debug needs, please email TA with the following information: Email Address, Name, NetID, Description, Steps to Reproduce, source files (Snack, Screenshot, or Link to a Repository).

Homework 1 Release

ECE 661: Homework #1 Linear Model, Back Propagation and Building a CNN

Yiran Chen

ECE Department, Duke University — January 22, 2024

Objectives

Homework #1 covers the contents of Lectures 02 ~ 04. This assignment includes conceptual questions on the linear model, back propagation and convolutions, as well as lab questions involving trying out LMS algorithm and building and observing a CNN model.

The lab questions in this assignment don't involve the training of deep learning models. So you are welcome to use your own computer to finish the labs. Please refer to the [NumPy/PyTorch tutorial](#) slides on Sakai for the environment setup on your computer.

HW1 Release Time: 8:00:00pm, January 22

HW1 Due Date: **11:55:00pm, February 5**

Platform: Gradescope Best Practice: Start Early!

Homework Submission

- Submission requirements
 - a self-contained report in PDF
 - **Include all necessary code screenshots and explanations in your PDF.**
- Gradescope
 - access it through Sakai
 - When uploading your report, match pages with questions (Gradescope provide this function).
 - <https://learninginnovation.duke.edu/technology/gradescope/>

Server Setup

- You may refer to <https://cmgr.oit.duke.edu/containers> to see the amount of available resources. Due to the limited availability, you may start your work early on our Jupyter Lab server.
- You may also refer to Google Colaboratory <https://colab.research.google.com/> for additional free computing resources. With a google account, you are good to go. The environment is similar to our Jupyter Lab environment, except that Google Colab is using PyTorch 2.0 and will be consistently upgrading the packages.
- We will keep you updated on alternative available resources to complete the coursework on Slack channel.

Server Setup

ECE661 Computer Engineering Machine Learning and Deep Neural Nets

[reserve ECE661](#)

GPUscavenger Scavenger JupyterLab with CUDA GPU

[reserve GPUscavenger](#)

CEE690 Environmental Spatial Data Analysis

[reserve CEE690](#)

BIOS823 notebooks for BioStat 823 Biomedical Big Data

[reserve BIOS823](#)

PHY164L Introduction to Physics 164L

[reserve PHY164L](#)