본 수업자료는 2025년도 과학기술 정보통신부 및 정보통신기획평가원의 ‘SW중심대학사업’ 지원을 받아 제작 되었습니다.

ComputerVision

Week2

2025-2

Mobile Systems Engineering

Dankook University

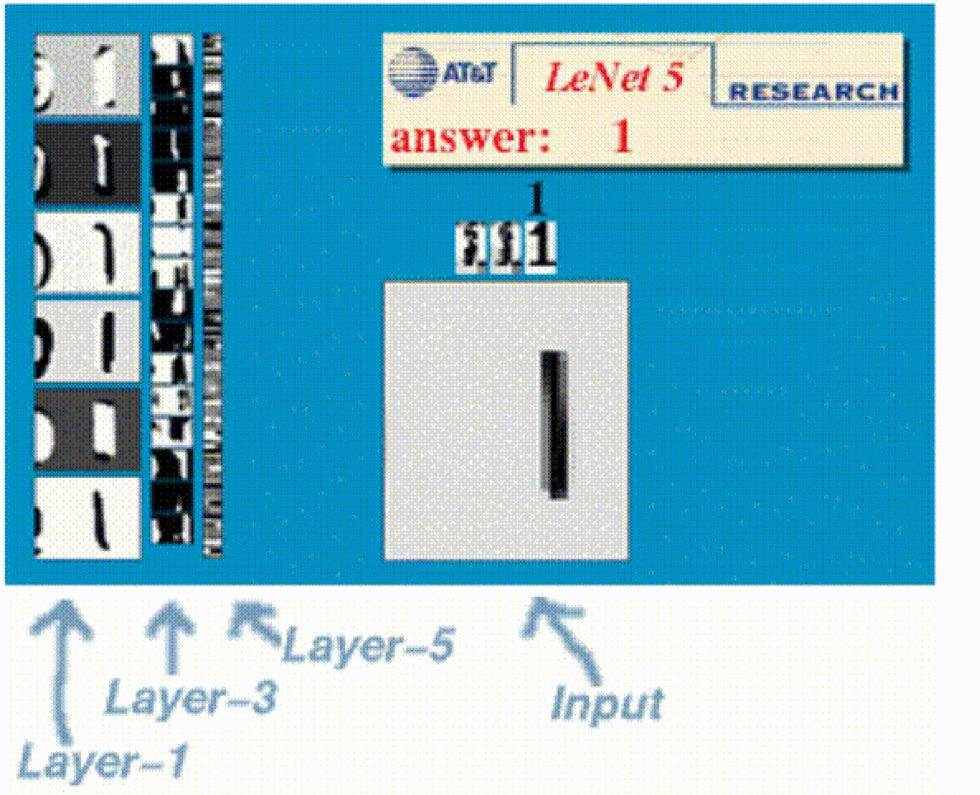
Learning Objectives

- **In this lecture, you will learn**
  - **(1)** What **LeNet-5** is and why it is historically important
  - **(2)** How a CNN processes images step-by-step.
  - **(3)** The roles of
    - **Convolutional Layers**: learn patterns like edges and textures.
    - **Subsampling (Pooling) Layers**: reduce the size of feature maps.
    - **Fully Connected Layers**: make the final decision.
  - **(4)** How to calculate the **number of trainable parameters** in each layer.
  - **(5)** Why LeNet-5 is considered the **foundation** of modern deep learning models.

LeNet-5: A Landmark in CNN History

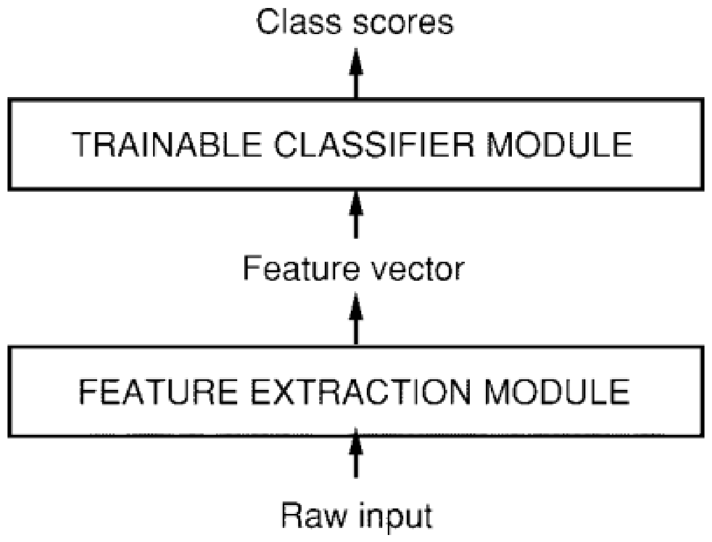
- **LeNet-5**
  - Developed by **Yann LeCun's team** in **1998**
  - One of the **first convolutional neural networks**
  - Designed to **recognize handwritten digits** (e.g., MNIST)
  - **Introduced in the paper**
    "Gradient-Based Learning Applied to Document Recognition"
  - Paved the way for modern deep learning models

LeNet and MNIST handwritten digit recognition



LeNet-5: A Landmark in CNN History

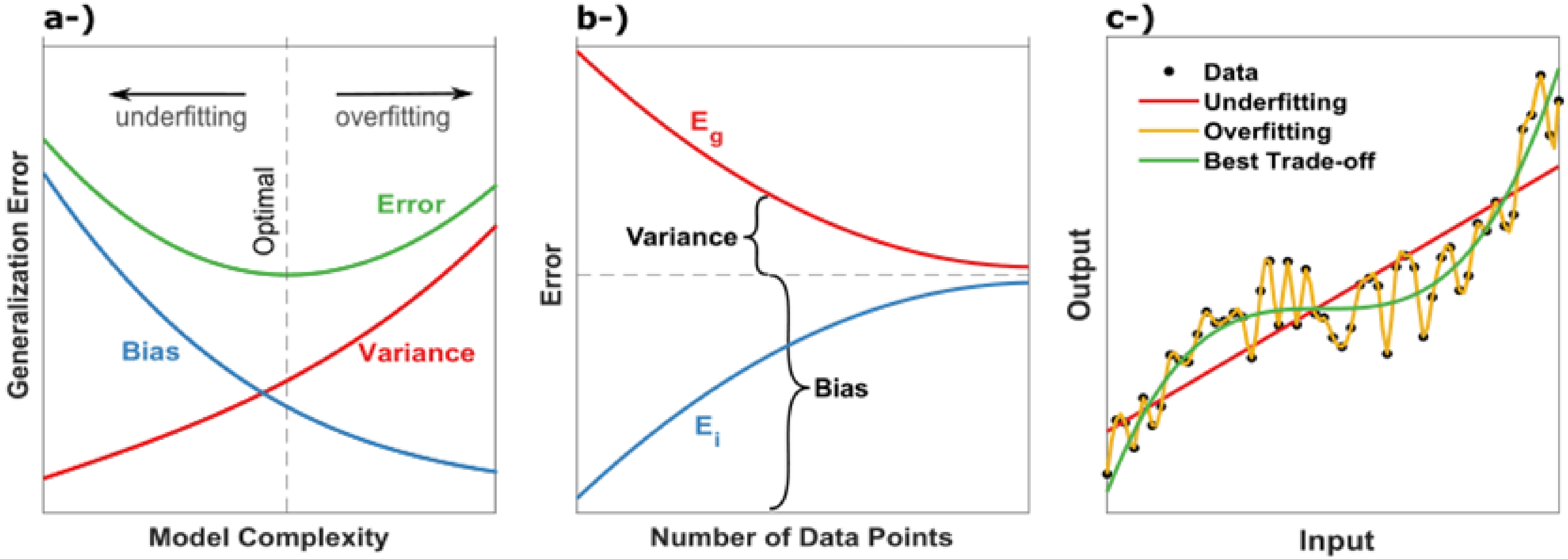
- **Historical Importance of LeNet-5**
  - **1. Real-world Application**
    - LeNet-5 was originally developed for digit recognition tasks in real documents such as **postal codes** and **bank checks**, addressing real industrial needs in the 1990s.
  - **2. Automatic Feature Extraction**
    - It replaced manually engineered features with a **trainable, end-to-end pipeline**, enabling the model to learn **edge, shape, and texture representations** directly from raw pixel data.
  - **3. Efficient Design Philosophy**
    - LeNet-5 used **shared weights**, **local receptive fields**, and **subsampling (average pooling)** to significantly **reduce the number of parameters**, allowing it to generalize well with limited computational resources and data.



Traditional pattern recognition

LeNet-5: A Landmark in CNN History

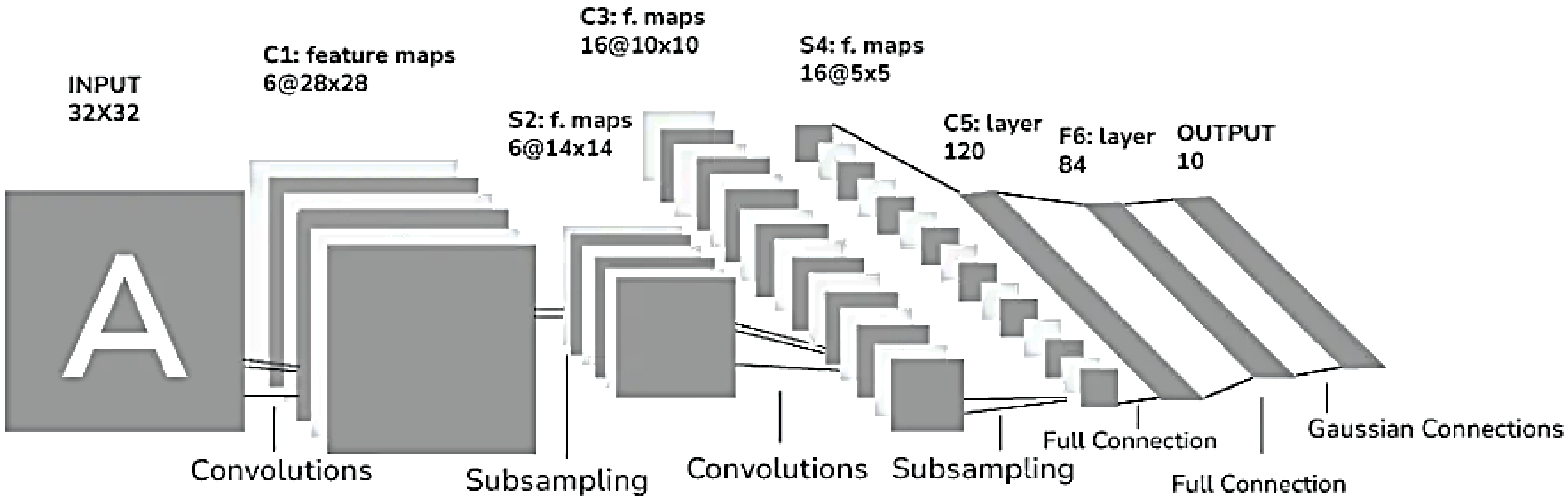
- **Historical Importance of LeNet-5**
  - **4. Designing for Generalization**
    - LeNet-5 demonstrated the importance of keeping model complexity low when training data is limited.
    - $E_{test} - E_{train} = k\left(\frac{h}{P}\right)^{\alpha}$; $\boldsymbol{k}, \boldsymbol{\alpha}$: constants (empirically determined)
      - ✓This equation explains how the **generalization gap** ($\boldsymbol{E_{test} - E_{train}}$) grows with model complexity $\boldsymbol{h}$ and shrinks with more data $\boldsymbol{P}$.
      - ✓LeNet-5 was carefully designed to keep $\boldsymbol{h}$ small by using **shared weights**, **local connections**, and **subsampling**.



Architecture of LeNet-5

▪ Architecture Overview

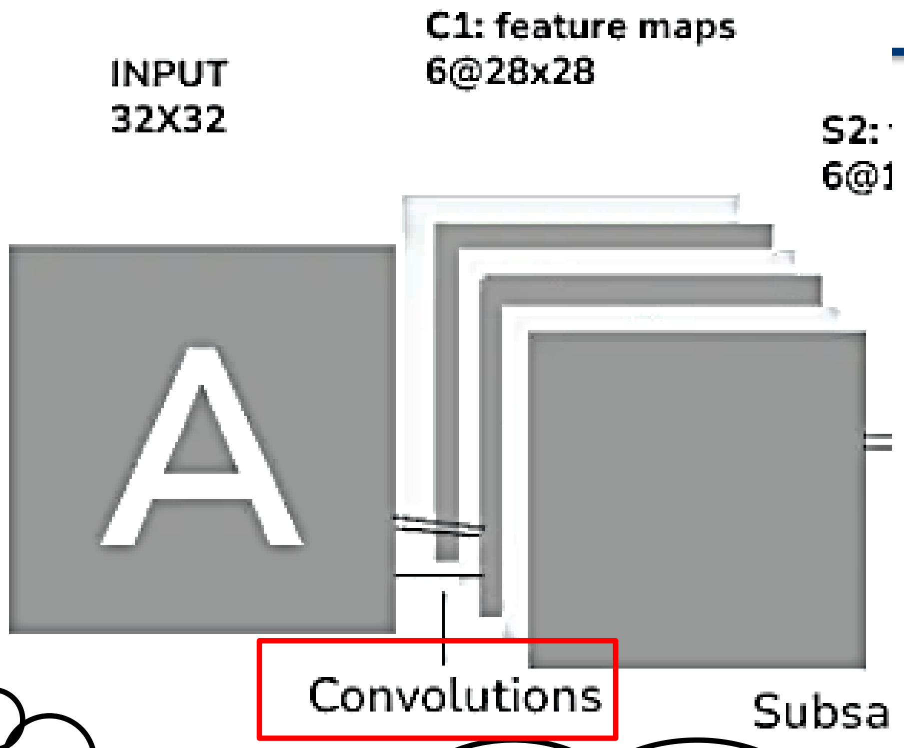
- **LeNet-5 is made of the following layers**
  - **Input layer:** Receives a **32×32 grayscale image** (1 channel)
  - **Three convolutional layers:** Extract patterns like **edges and textures** → **C1, C3, C5**
  - **Two subsampling layers** (average pooling): Reduce the size of feature maps to make the model faster and simpler → **S2, S4**
  - **One fully connected layer**: Combines all features and makes a decision → **F6**
  - **Output layer**: Gives the final prediction using **10 class scores** (digits 0 to 9)



Architecture of LeNet-5

- **C1 – First Convolution Layer**
  - **Input**: 32×32 grayscale image
  - **6 filters of size 5×5**
    → produces **6 feature maps of size 28×28 (i.e., 6@28x28)**
  - **No padding, stride = 1**
  - **Activation**: tanh
  - **Number of Parameters in C1**
    - = parameters

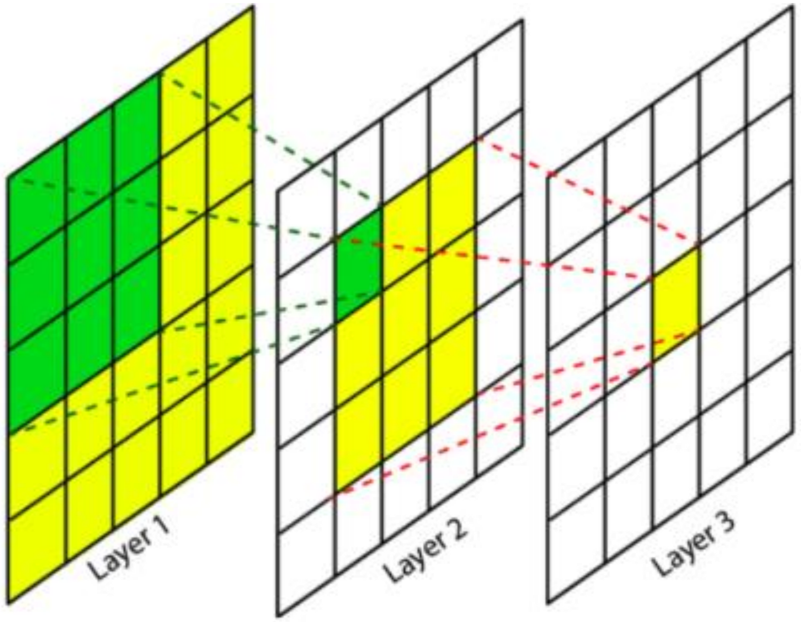
      ✓The size of each filter is 5×5.
      ✓Input has 1 channel
      ✓Plus 1 bias, which is also a trainable parameter
      ✓6 filters (i.e., 6 output channels)



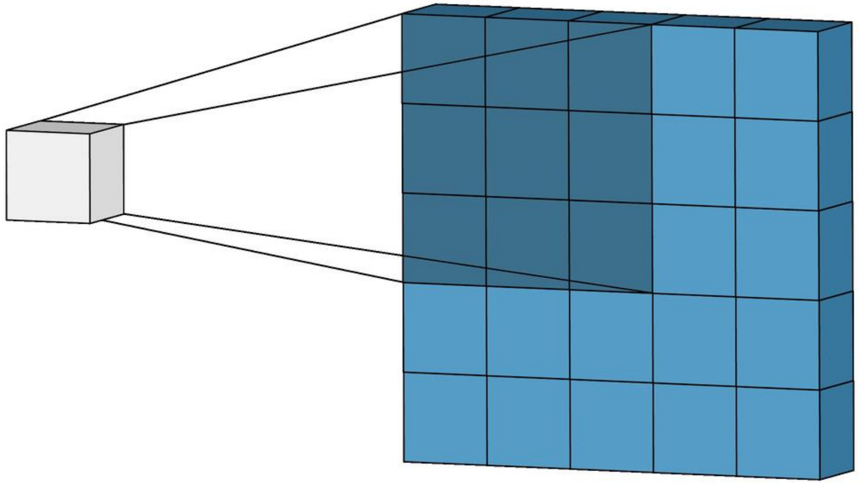
But… how exactly does a convolution work?
what's really happening during convolution?

Architecture of LeNet-5

- Convolution in Deep Learning
  - **What does a Convolution Layer do in CNNs?**
    - In deep learning, a **convolution layer** is used to
      - ✓Extract **local features** from an input image
      - ✓Slide a small filter (kernel) across the input
      - ✓At each location, **multiply** and **sum** the overlapping values
      - ✓Create a **feature map** that highlights patterns (e.g., edges, textures)



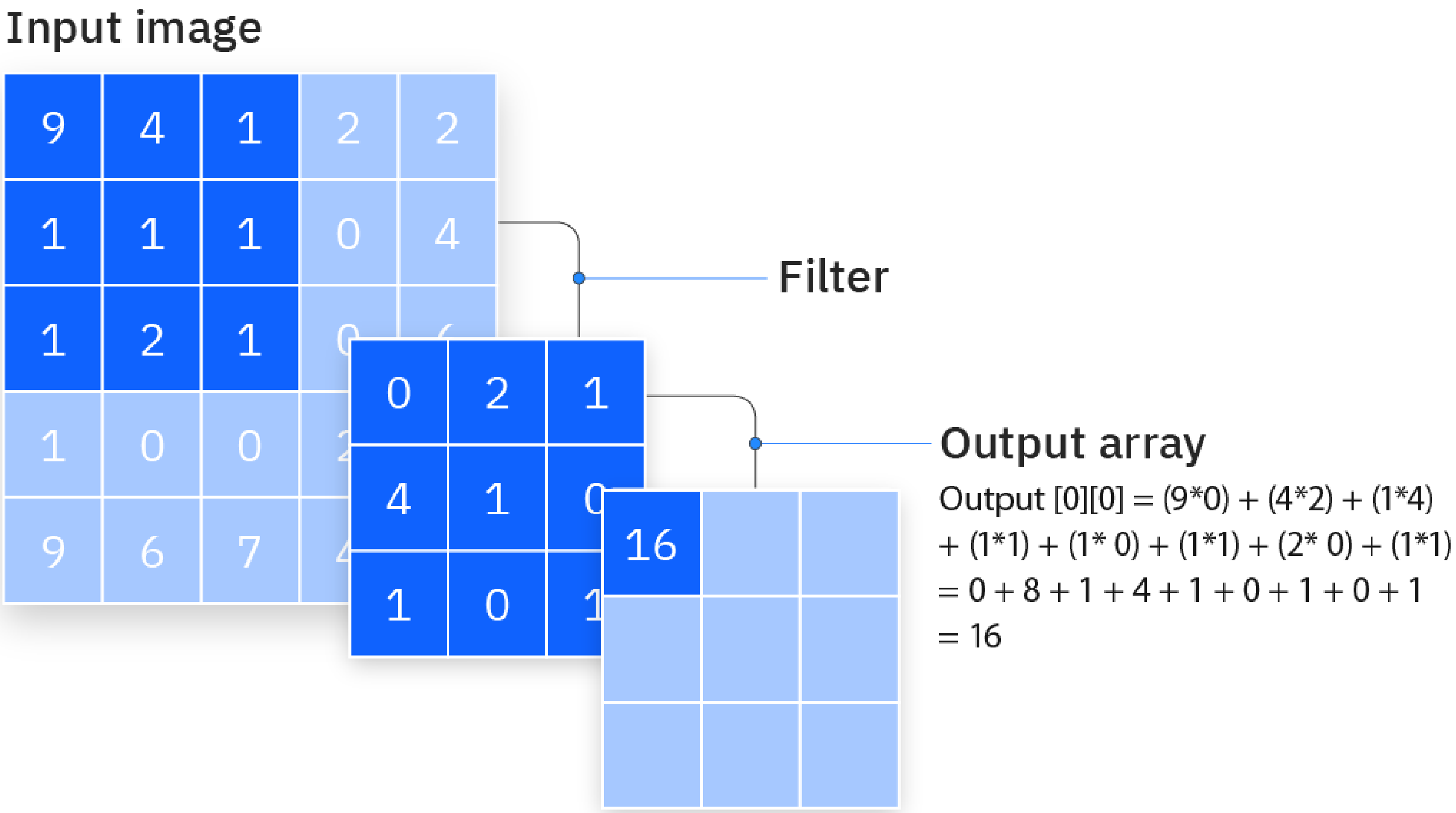
  - "Unlike traditional signal processing,
    the filters are **learned** from data during training."

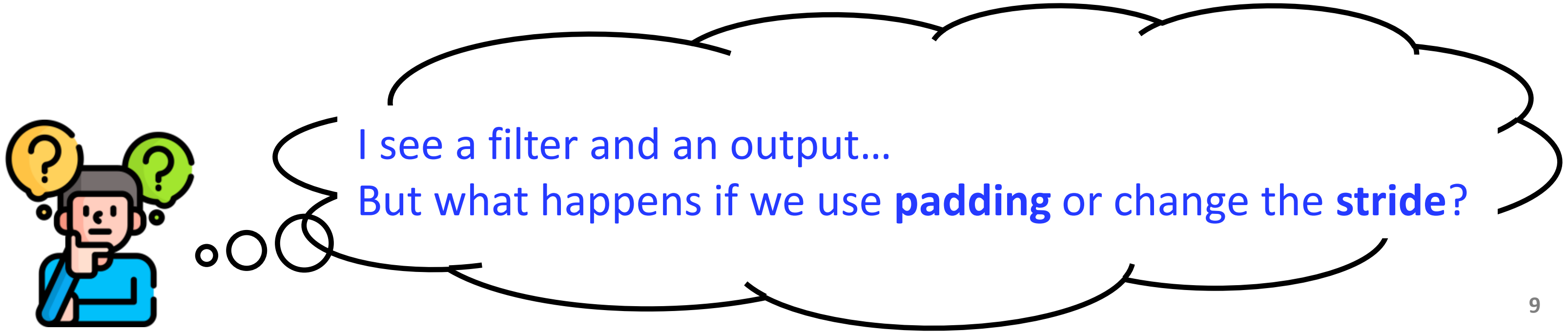


Example of convolution layer with 3x3 filter and 1 of stride

Architecture of LeNet-5

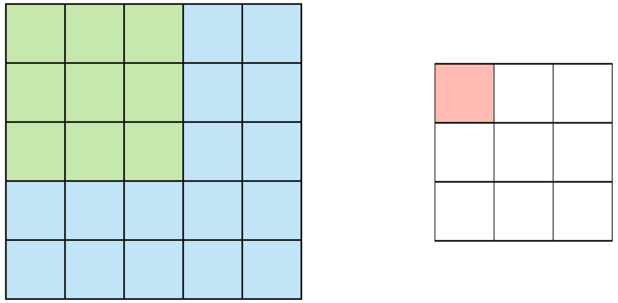
- **Convolution in Deep Learning**
  - **How to Compute Output (Size)?**
    - Let's define
      - ✓ W_{in}, H_{in}: Input width and height
      - ✓ K: Kernel (filter) size
      - ✓ P: Padding
      - ✓ S: Stride
    - Then the output size W_{out}, H_{out} is
      - ✓ $W_{out} =$
      - ✓ $H_{out} =$



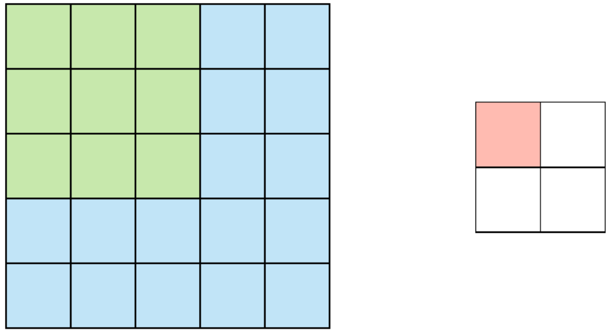


Architecture of LeNet-5

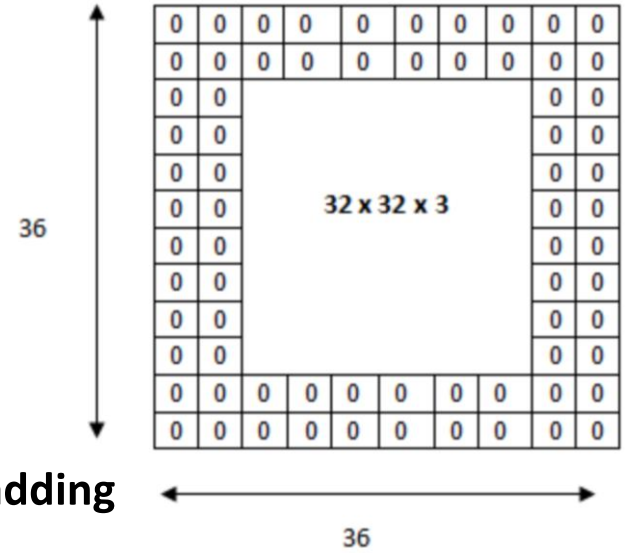
- **Convolution in Deep Learning**
  - **What is Stride?**
    - **Stride** is the number of steps the filter moves each time.
    - Default: stride = 1 (slide the filter **1 pixel** at a time)
    - If stride > 1 → output becomes **smaller** (you skip some positions)
  - **What is Padding?**
    - **Padding** adds extra pixels (usually zeros) around the input image.
    - Purpose
      - ✓Keep output size the same as input (called **"same" padding**)
      - ✓Prevent information loss at the edges



Stride 1 Feature Map



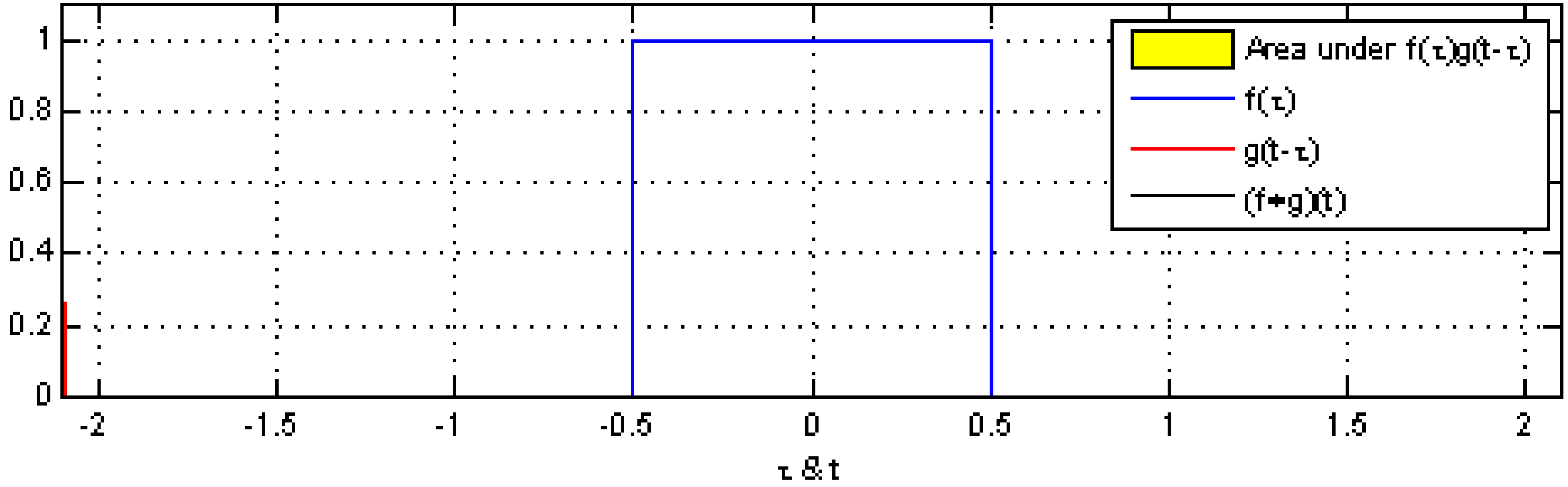
Stride 2 Feature Map



Add 2 of padding

Architecture of LeNet-5

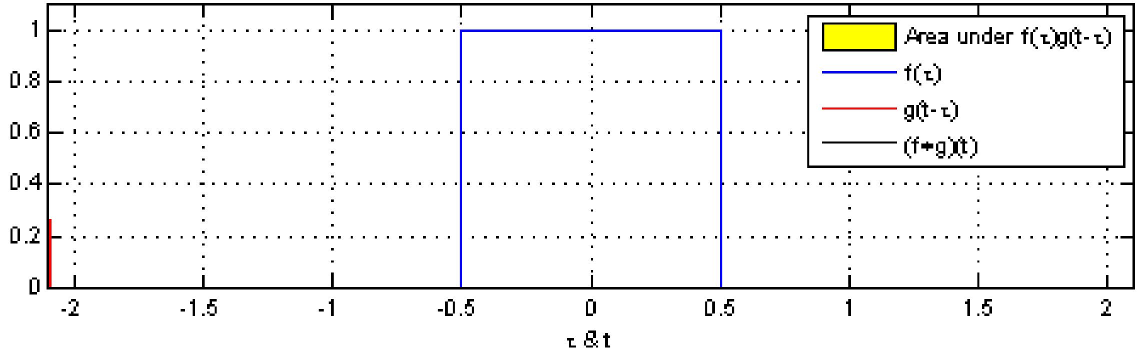
- **What is Convolution?**
  - "**Convolution** is a mathematical operation used **to combine two functions into a third one**."



→ "It **captures how one function modifies or responds to another function** as it shifts across time or space."
→ **"**Convolution is used to measure **how similar two functions are** at different alignments."

Architecture of LeNet-5

- **What is Convolution?**
  - **Let's define two continuous functions for convolution.**
    - $(f * g)(t) =$



  - **Step-by-step Explanation**
    - **1. Reverse one function**
      - ✓To compute the convolution, we first **flip** one of the two functions.
        - ➢**Original function:** $g(\tau)$
        - ➢**Flipped version:** $g(-\tau)$
      - ✓**Why do we flip the function?**
        - ➢A. To properly measure this similarity, one function (usually g) must be **reversed** so that it aligns correctly with the other.
        - ➢B. This is especially important when **matching a known pattern** (like in signal processing).
        - ➢C. Convolution is used to measure **how similar two functions are** at different alignments.

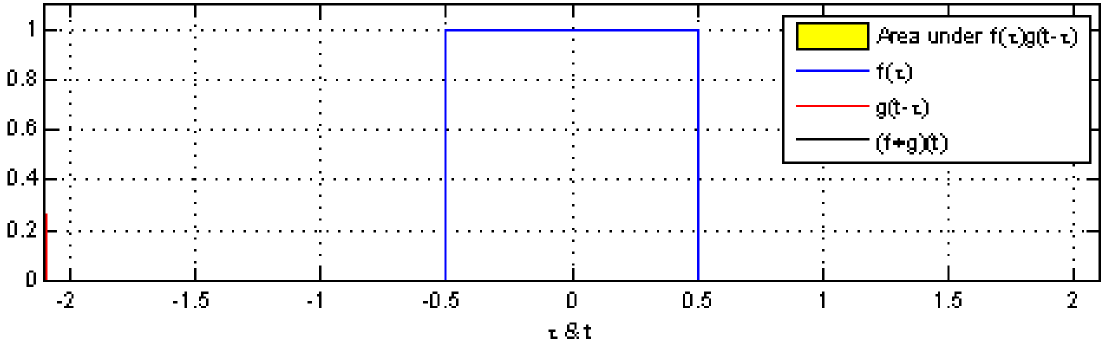
Architecture of LeNet-5

- **What is Convolution?**
  - **Step-by-step Explanation**
    - **1. Reverse one function**
      - ✓ **Why do we flip the function?**
        - ➢ $\boldsymbol{f}$: the **signal** (i.e., input data)
        - ➢ $\boldsymbol{g}$: the **pattern** or **template** (i.e., filter)
        - ➢ What we want is "To find where f and g match best — i.e., how similar they are at different positions."
          - However, if we don't flip $\boldsymbol{g}$, they might **slide in opposite directions**, and the result can be misleading.
      - ✓ Let's Take a Simple Example
        - ➢ $Let\ \boldsymbol{f} = [\mathbf{1}, \mathbf{2}, \mathbf{3}], \boldsymbol{g} = [\mathbf{3}, \mathbf{2}, \mathbf{1}]$
        - ➢ If we **don't flip**, and just multiply and sum:
          -
        - ➢ Now, flip $\boldsymbol{g}$ to get $\boldsymbol{g}' = [\mathbf{1}, \mathbf{2}, \mathbf{3}]$
          -

The result is **higher and more meaningful** when we flip.

Architecture of LeNet-5

- **What is Convolution?**
  - **Let's define two continuous functions for convolution.**
    - $(\boldsymbol{f} * \boldsymbol{g})(\boldsymbol{t}) =$
  - **Step-by-step Explanation**
    - **2. Shift the flipped function**
      - ✓The function g is then **shifted** by t.
        - ➢**Flipped version:**
        - ➢**Shifted version**:
      - ✓**Why Do We Shift by $\boldsymbol{t}$?**
        - ➢The variable $\boldsymbol{t}$ tells us **where** we are checking for similarity
        - ➢For each position $\boldsymbol{t}$, we compare $\boldsymbol{g}$ to a **different part** of $\boldsymbol{f}$.
        - ➢We are **sliding** $\boldsymbol{g}$ across the entire input $\boldsymbol{f}$.



Think of It Like This

At $\boldsymbol{t} = \mathbf{0}$: $\boldsymbol{g}$ starts at the beginning of $\boldsymbol{f}$

At $\boldsymbol{t} = \mathbf{1}$: $\boldsymbol{g}$ moves one step right

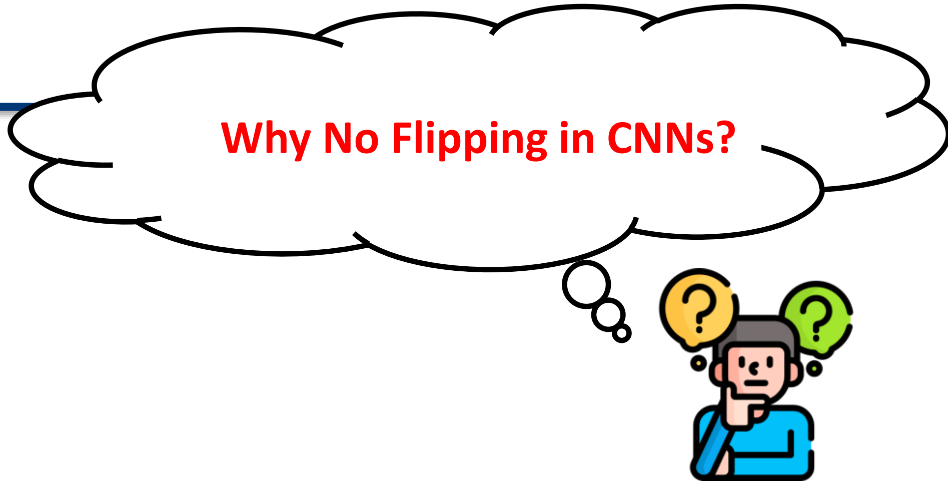
At $\boldsymbol{t} = \mathbf{2}$: move again, and so on...

We compute a value at each $\boldsymbol{t}$ that tells us **how well $\boldsymbol{g}$ matches $\boldsymbol{f}$ at that position**.

Architecture of LeNet-5

- **What is Convolution?**
  - **Let's define two continuous functions for convolution.**
    - $(\boldsymbol{f} * \boldsymbol{g})(\boldsymbol{t}) =$
  - **Step-by-step Explanation**
    - **3.** Multiply and accumulate
      - ✓At each shift, we compute the **product** $\boldsymbol{f(\tau)g(t-\tau)}$ and then **integrate** $\int_{-\infty}^{\infty} \dots \boldsymbol{d\tau}$ (i.e., sum) over all $\boldsymbol{\tau}$ to produce a single output value.
      - ✓This is like measuring **how well the two functions overlap** at each position.
        → ***"The Similarity of Two Functions"***
    - ***Intuitive Interpretation***
      - ✓"How similar is the input $\boldsymbol{f}$ to the flipped and shifted version of $\boldsymbol{g}$ (i.e., filter)?"
        - ➢Sometimes the overlap is large → high output.
          Sometimes there's no overlap → output is zero.

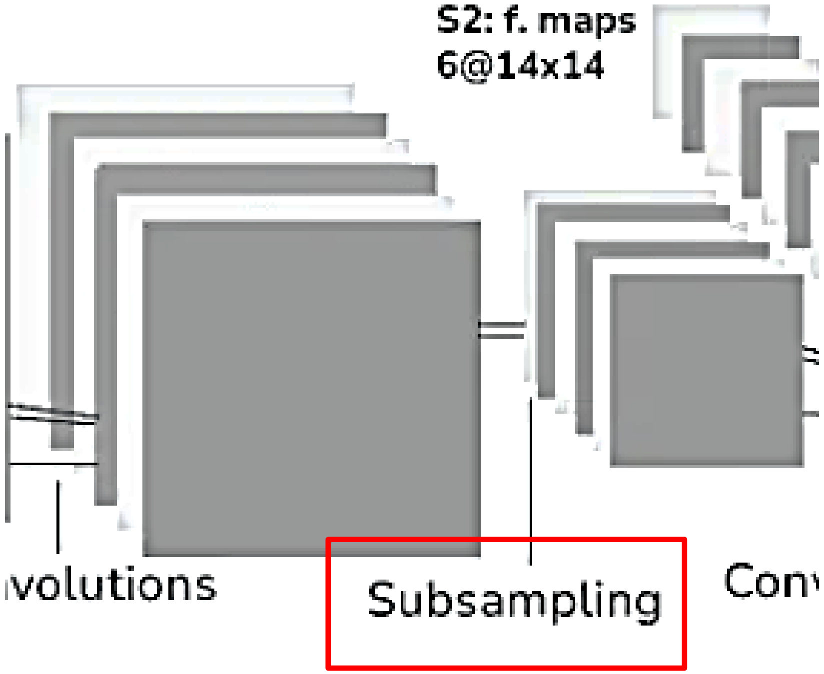
Architecture of LeNet-5



- **What is Convolution?**
  - In deep learning, we use the **discrete version.**
    - $(f * g)(t) =$
    - **Why Is Convolution Discrete in CNNs?**
      - ✓ **1.** Input Data Is Discrete – CNNs process **digital images**, which are made of **pixels.** So, a digital image is a **2D array of values**, not a continuous function.
      - ✓ **2.** Efficient for Computation – **Continuous convolution** involves integration, which is slow and hard to compute.
      - ✓ **3.** Learnable Filters in Deep Learning – CNN filters are **learnable parameters.** Furthermore, continuous filters would be too complex to learn efficiently
    - **Why No Flipping in CNNs?**
      - ✓ In signal processing, convolution flips g to match known patterns.
      - ✓ But in CNNs, we don't have a known pattern — we **learn the filter weights** g!
      - ✓ So **flipping isn't needed**, and cross-correlation is used instead.

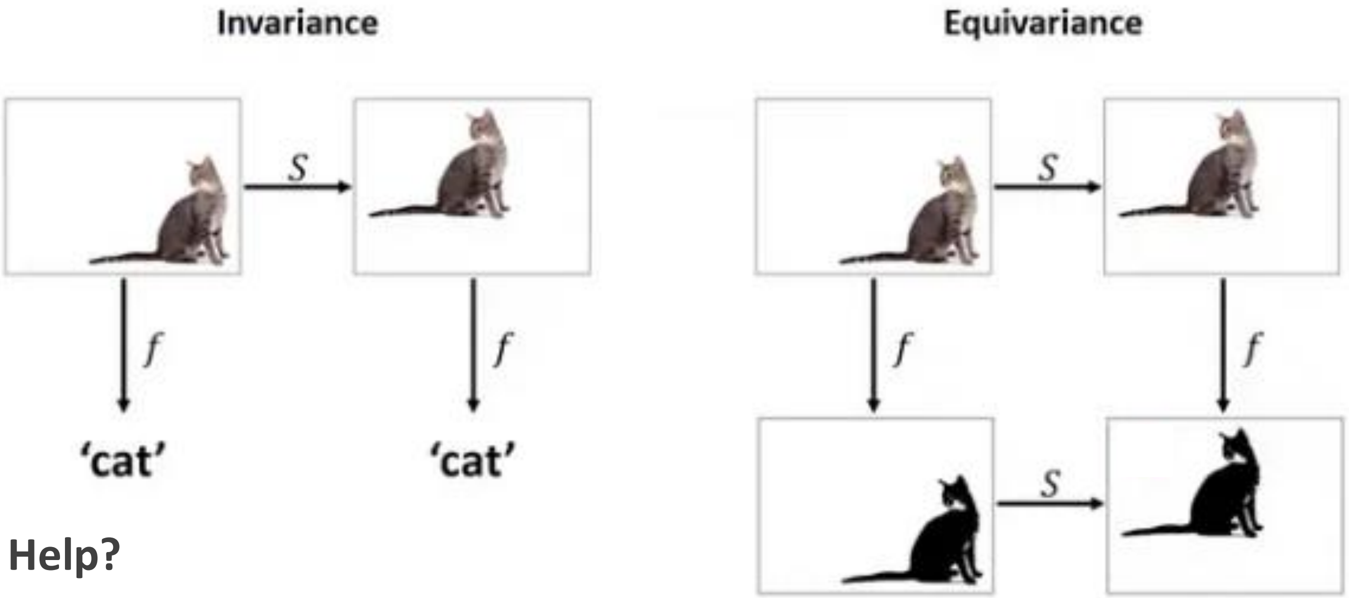
Architecture of LeNet-5

- **S2 – First Subsampling (Average Pooling) Layer**
  - **Input**: **6 feature maps of size 28×28 (i.e., 6@28x28)**
  - **Pooling operation**:
    → Uses **2×2 average pooling**
    → **Stride** = 2 → reduces each map to **14×14**
  - **Activation function**: tanh after pooling
  - **Number of Parameters in S2**
    - = parameters
  - **Purpose of Pooling**
    - Reduces spatial size
    - Introduces ***translation invariance***
    - Keeps computational cost low
    - Retains important structure before next convolution
    - Provides ***non-linearity*** based on activation functions



Architecture of LeNet-5

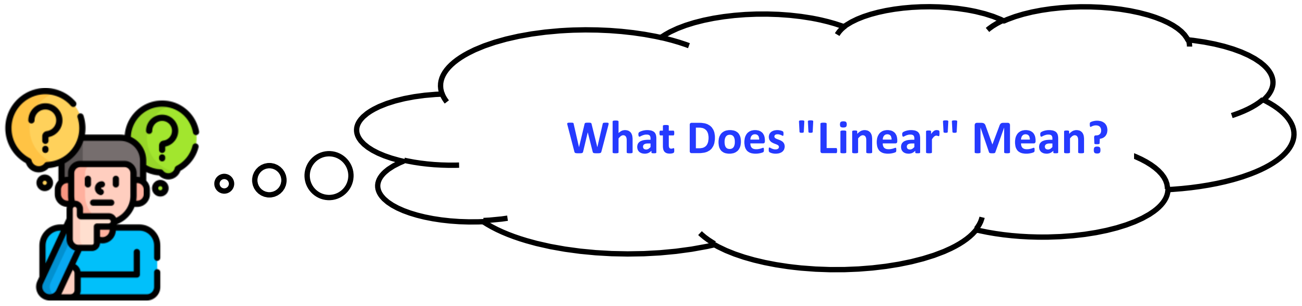
- **S2 – First Subsampling (Average Pooling) Layer**
  - **What Is Translation Invariance?**
    - A feature is still recognized even if it slightly moves in the input image.



    - **How Does Pooling Help?**
      - ✓Pooling layers help build **shift invariance** in convolutional networks.
      - ✓**Shift invariance** means that ***<u>the same maximum value will be found under the pooling kernel even if the image is shifted slightly</u>***.
      - ✓Focuses on **"what exists"**, not exactly **"where"**
      - ✓***<u>However, this shift invariance is only locally true and may not hold if the image is shifted too much.</u>***

Architecture of LeNet-5

- **S2 – First Subsampling (Average Pooling) Layer**
  - **Activation Functions Add Non-Linearity After Pooling**
    - **Why Is Non-Linearity Important?**
      - ✓ Without non-linearity, a neural network would just be a **linear function**
      - ✓ Linear layers stacked together still behave like a single linear layer
      - ✓ We need **non-linear functions** to model **complex patterns**
    - **Activation After Pooling**
      - ✓ In CNNs, pooling layers are typically followed by an **activation function** (like ReLU, tanh, etc.)
        - ➢ This introduces **non-linearity** into the model.

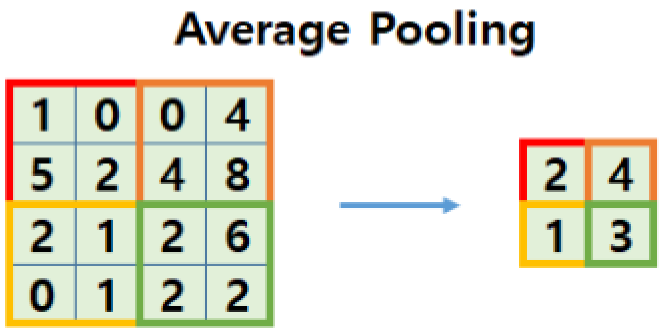


Architecture of LeNet-5

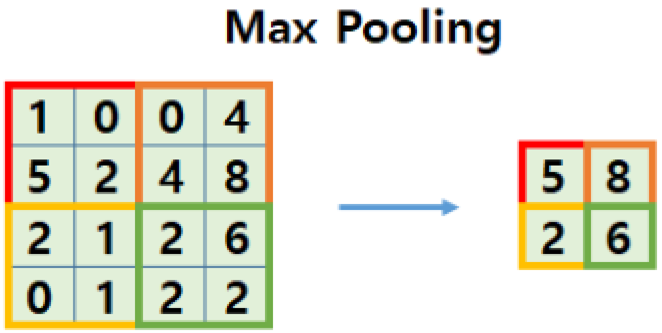
- **S2 – First Subsampling (Average Pooling) Layer**
  - **Activation Functions Add Non-Linearity After Pooling**
    - **What Does "Linear" Mean?**
      - ✓ In mathematics, a function is **linear** if it satisfies
        - 1. Additivity – $\boldsymbol{f(x+y)} =$
        - 2. Homogeneity (scaling) – $\boldsymbol{f(a \cdot x)} =$
      - ✓ **Why Is That a Problem?**
        - If a neural network uses only **linear layers**, stacking multiple layers is still just one big linear transformation.
        - "It **cannot model complex patterns**, curves, or decisions."
    - **How Do Activation Functions Add Non-Linearity?**
      - ✓ Activation functions like: ReLu $\boldsymbol{f(x) = \max(0, x)}$, Sigmoid $\boldsymbol{f(x) = \frac{1}{1+e^{-z}}}$, tanh $\boldsymbol{f(x) = \tanh(x)}$
      - ✓ These functions break linearity: $\boldsymbol{f(a \cdot x) \neq a \cdot f(x)}$
        - → "This allows the network to **learn complex decision boundaries and features**."

Architecture of LeNet-5

- **S2 – First Subsampling (Average Pooling) Layer**
  - **Types of Pooling in CNNs**
    - Average Pooling – **(1)** Computes the **mean value** in each window and **(2)** smooths out feature maps

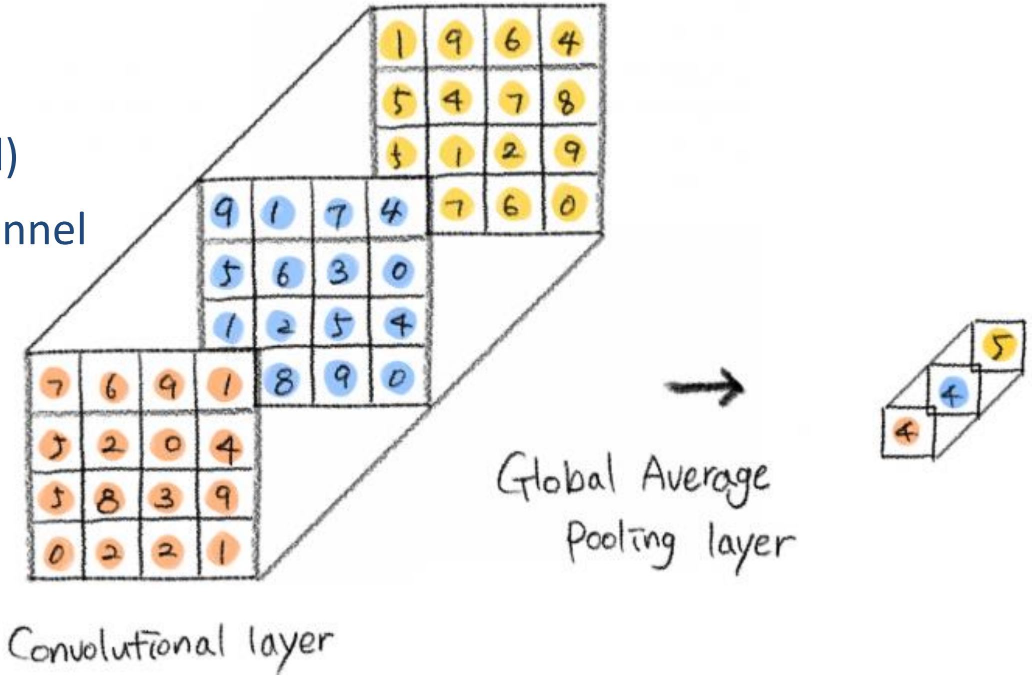


    - Max Pooling – **(1)** Selects the **maximum value** in the window and **(2)** Preserves strong activations, highlights key features



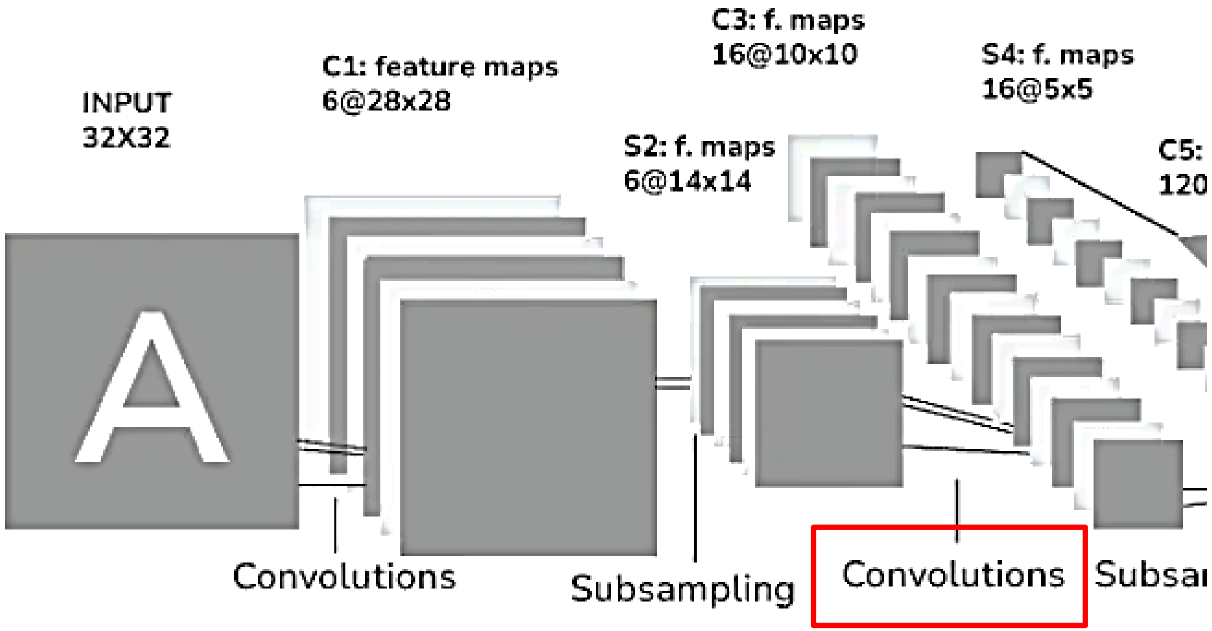
Architecture of LeNet-5

- **S2 – First Subsampling (Average Pooling) Layer**
  - **Types of Pooling in CNNs**
    - Global Average (or Max) Pooling
      - ✓GAP computes the **average value** of each feature map (channel)
      - ✓Converts a feature map of size $H \times W$ to a **single value** per channel
      - ✓Output becomes a **1 × 1 × C** tensor (C = number of channels)
      - ✓For each channel, simply compute
        - ➢$GAP(X_c) =$
      - ✓**Why Was GAP Introduced?**
        - ➢To replace the **fully connected (FC) layer** at the end of CNNs
        - ➢FC layers have: **(1) too many parameters** → high computational cost, **(2) High risk of overfitting, (3) Fixed input size** → limits flexibility
        - ➢GAP was proposed to solve these problems by: **(1)** Reducing the number of parameters to **zero, (2)** Making the model **lighter and more flexible, (3)** Still allowing for **end-to-end learning**



Architecture of LeNet-5

- **C3 – Second Convolution Layer**
  - **Overview of C3 Layer**
    - **Input**: 6 feature maps from S2 layer (size 14×14)
    - **Filter**: 5×5 size
    - **Output**: 16 feature maps (size 10×10)
    - **Stride**: 1, **No padding**
    - ***“But not all input maps are connected to all output maps!”***



  - **Why Not Full Connection?**
    - The original LeNet-5 paper gives two reasons
      - ✓**(1) Reduce the number of connections** (parameter efficiency)
      - ✓**(2) Encourage diversity**: Each output feature map extracts **different combinations of features**, avoiding redundancy

Architecture of LeNet-5

- **C3 – Second Convolution Layer**
  - **Why Not Full Connection?**
    - Grouped Connections

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

TABLE I

EACH COLUMN INDICATES WHICH FEATURE MAP IN S2 ARE COMBINED BY THE UNITS IN A PARTICULAR FEATURE MAP OF C3.

Output Maps (C3)	Input Maps (S2) Used	Number of Outputs
Group 1		
Group 2		
Group 3		
Group 4		

✓These groupings are **manually chosen** hyperparameters

Architecture of LeNet-5

- **C3 – Second Convolution Layer**
  - **Parameter Calculation**
    - Each convolution filter has: $\boldsymbol{Filter\ size} \times \boldsymbol{input\ maps} + \boldsymbol{bias}$

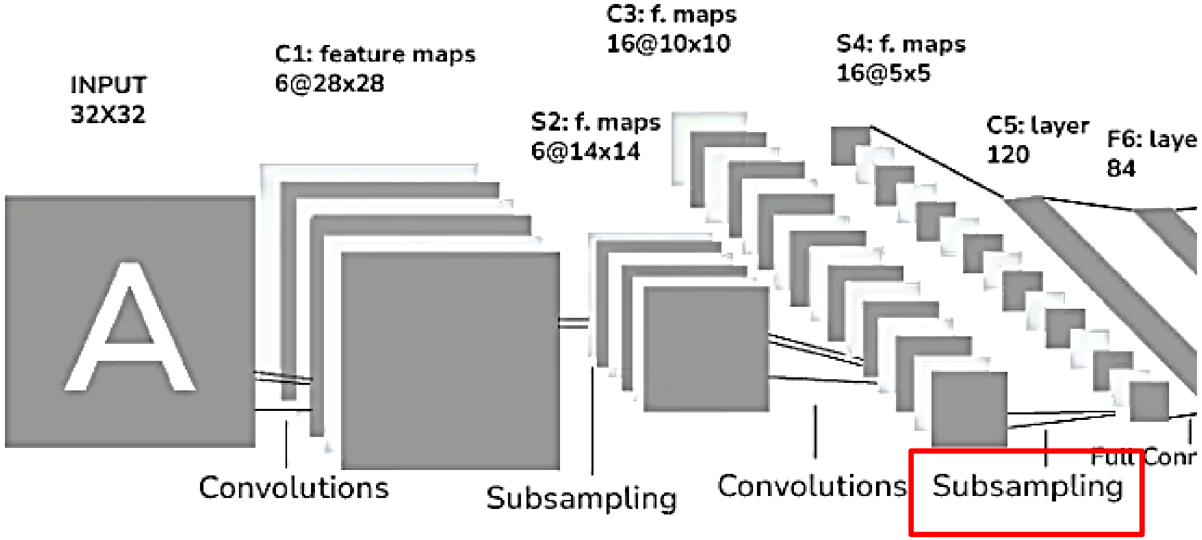
Group	Formula	Parameters
Group 1		
Group 2		
Group 3		
Group 4		

✓Total trainable parameters:

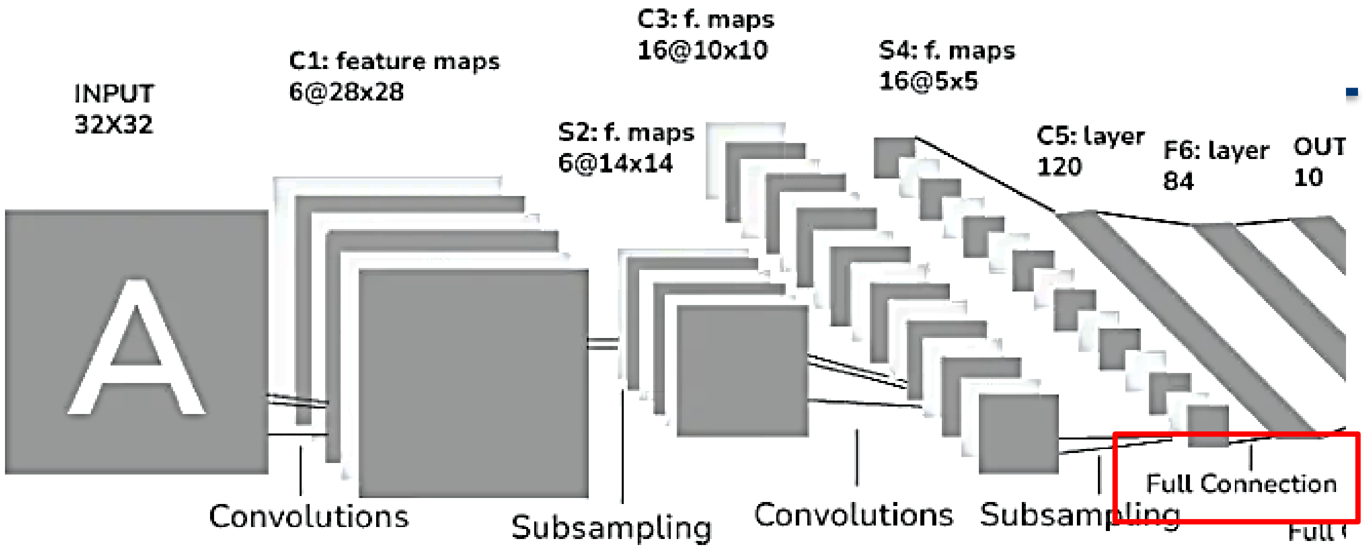
  - **Why This Design Is Smart**
    - Reduces parameter count without sacrificing performance
    - Prevents overfitting by reducing redundancy
    - Promotes specialization in different output maps

Architecture of LeNet-5

- **S4 – Second Subsampling (Average Pooling) Layer**
  - **Overview of S4 Layer**
    - **Input:** 16 feature maps of size **10×10** (from C3)
    - **Pooling**
      - ✓Average Pooling
      - ✓**Filter size: 2×2**, Stride: **2**
      - ✓**Output:** 16 feature maps of size **5×5**
    - **Activation Function:** tanh
    - **Learnable Parameters**
      - ✓Each feature map uses **1 learnable weight** and **1 bias**
      - ✓ $= \quad parameters$



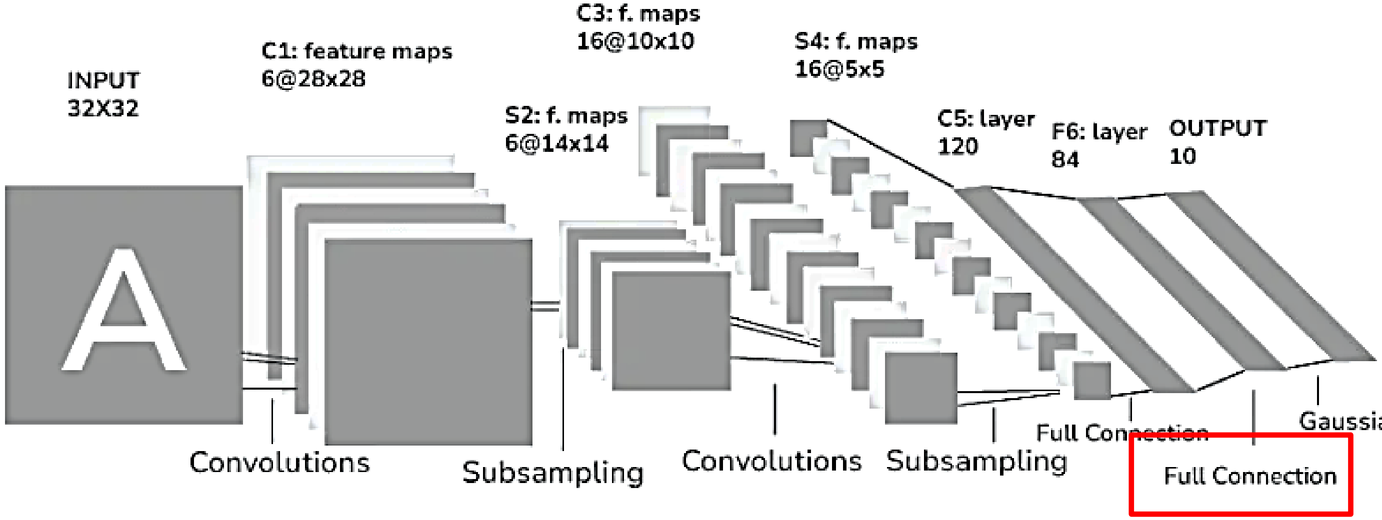
Architecture of LeNet-5



- **C5 – Third Convolution Layer**
  - **Overview of C5 Layer**
    - **Input:** 16 feature maps of size **5×5** (from S4)
    - **Output:** 120 feature maps of size **1×1**
    - **How?**
      - ✓ Each filter covers **all 16 input maps**
      - ✓ Filter size: **5×5×16** (i.e., 5×5 per input map, across 16 maps)
    - **Activation Function:** tanh
    - **Learnable Parameters**
      - ✓ Each output feature map has = weights
      - ✓ Plus **1 bias**
      - ✓ = *parameters*

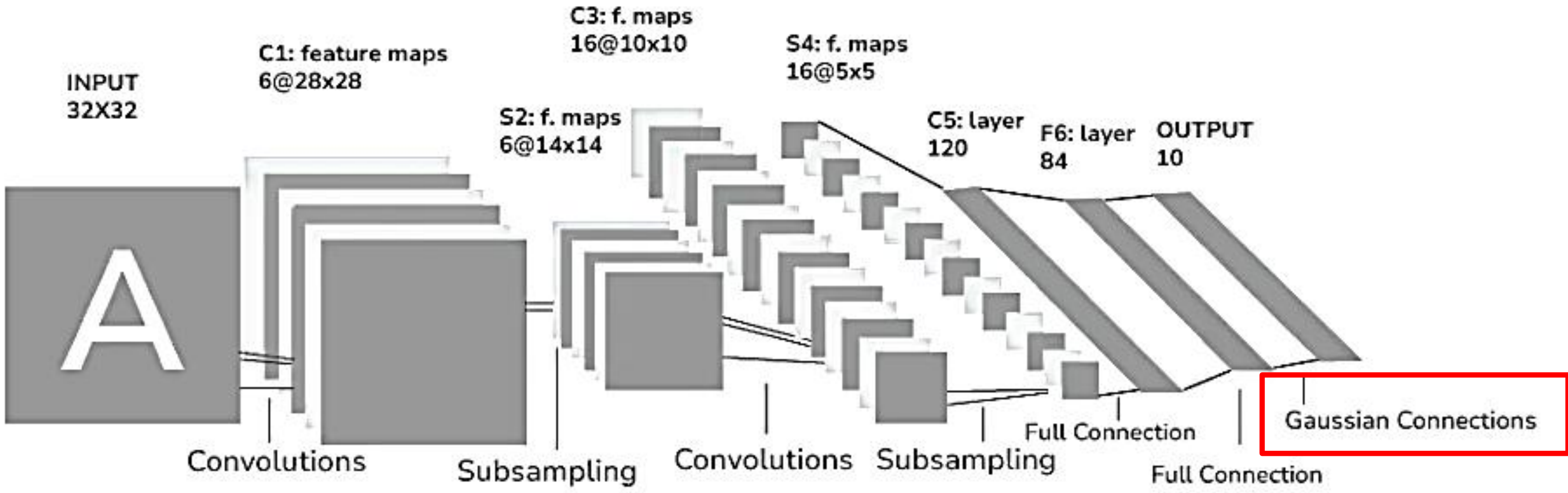
Architecture of LeNet-5

- **F6 – Fully Connected Layer**
  - **Overview of F6 Layer**
    - ○ **Input:** 120 values from previous C5 layer (1×1 feature maps)
    - ○ **Output:** 84 neurons
    - ○ **Connection**
      - ✓Fully connected to all 120 inputs
      - ✓Each output neuron also has a **bias**
    - ○ **Activation Function:** tanh
    - ○ **Parameters**
      - ✓Each output neuron → 120 weights + 1 bias



✓ $=$ $\boldsymbol{parameters}$

Architecture of LeNet-5



- Output Layer
  - Overview of Output Layer
    - **Output:** 10 neurons
      (for digit classification: 0 through 9)
    - **Key Characteristics**
      - ✓Uses **Radial Basis Function (RBF)** units instead of softmax
      - ✓Each neuron is connected to **all 84 units** from F6
    - **Learning**
      - ✓**Backpropagation** is used to update the weights
      - ✓The output neuron with the **highest activation** is selected as the predicted class
    - ***Why RBF?***
      - ✓The original LeNet-5 paper used **Euclidean distance-based RBF units.**
      - ✓These act like **prototypes** for each digit class and measure **similarity** to input.

Summary – What We Learned about LeNet-5

- **What is LeNet-5?**
  - One of the **first CNNs**, developed by **Yann LeCun** in 1998 for **digit recognition** (MNIST).
  - Pioneered **end-to-end learning** from raw pixels using learned filters.
  - Introduced key design ideas still used in modern CNNs.
  - Architecture Breakdown

Layer	Type	Output Shape	Key Details
C1	Convolution	6@28×28	5×5 filters, stride=1
S2	Avg Pooling	6@14×14	2×2 pooling, tanh activation
C3	Convolution	16@10×10	Not fully connected – grouped connections
S4	Avg Pooling	16@5×5	2×2 pooling, learnable weights
C5	Convolution	120@1×1	5×5×16 filters (full connection)
F6	Fully Connected	84	Each neuron gets all 120 inputs
Output	RBF Layer	10	Radial Basis Function units

Assignment 1 – Dive into LeNet-5

- **What you will explore**
  - **Please revisit the following questions based on your reading and implementation**
    - **Q1.** Why did LeNet-5 use RBF in the output layer instead of softmax?
    - **Q2.** How is LeNet-5 different from modern CNN architectures?
    - **Q3.** Why is C3 only partially connected to S2?
    - **Q4.** Why is LeNet-5 still important today?
  - **Your Task**
    - **Assignment 1**
      - ✓**Read** the original paper *"Gradient-Based Learning Applied to Document Recognition"*
      - ✓**Understand** the full design and reasoning behind LeNet-5
      - ✓**Answer** the above four questions
      - ✓**Implement** LeNet-5 and **Train** LeNet-5 on **MNIST** Dataset