Convolutional Neural Networks (CNNs)

AI with Deep Learning

EE4016

Prof. Lai-Man Po

Department of Electrical Engineering

City University of Hong Kong

Message 1: Submission of Project Proposal

- **Just a friendly reminder:** The deadline to submit your group project proposal is **Feb 14, 2026, at 11pm**. Please submit a PDF file with the project title, list of group members, and other necessary details to the CANVAS group project proposal assignment.
- You can find more information about the group project on the course website:
  - https://www.ee.cityu.edu.hk/~lmpo/ee4016/projects.html
- Remember, **each group should assign a team leader** who will be responsible for submitting the proposal on CANVAS.
- The file name should follow this format:
  - Filename format : Proposal_GroupNumber_ProjectName.pdf
  - Filename example: Proposal_Group01_Audio_Classification.pdf

Message 2 : Canvas Quiz on Week 6

The Canvas quiz consisting of **30 to 40 multiple choice questions** will be released on **Feb 23, 2026, at 1:45 PM and the quiz will be end at 2:45PM**

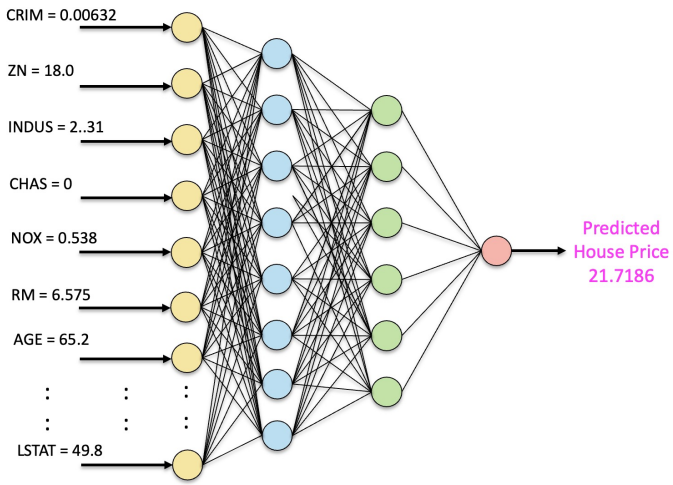
- **Students are required to take the quiz in the LI-2614:** Attendance will be recorded using student IDs.
- The quiz will cover course content from Weeks 1 to 4
  - Math, Perceptron, ADALINE, MLPs, Gradient Descent, Backpropagation, Mastering Deep Neural Network Trainings
- **This quiz is Semi-Open Book:**
  - The first 45 minutes will be closed book. Students are not allowed to use any materials.
  - The last 15 minutes will be open book. You are permitted to use your devices to access the PPT lecture notes of the class.
    - You may also view all course handouts during this time in hardcopy or on your devices.
    - However, you are not allowed to communicate with others and use of ChatGPT or any other LLM services during the quiz.
    - Investigators will conduct periodic checks to ensure that students comply with these regulations during the quiz.

Content

- **Introduction of Convolutional Neural Network (CNN)**
  - CNN vs MLP for Image Classification Applications
  - Intuition of CNN
  - Convolution Operation
  - Basic Architecture of CNNs
- **Evolution of CNN Architectures**
  - Variants of Convolution
  - LeNet, AlexNet, VGGNet, GoogleNet, ResNet, DenseNet, MobileNet, etc
  - Transfer Learning: Pre-training vs Fine-tuning
  - Applications of CNNs: Object Detection, Image Segmentation, etc.

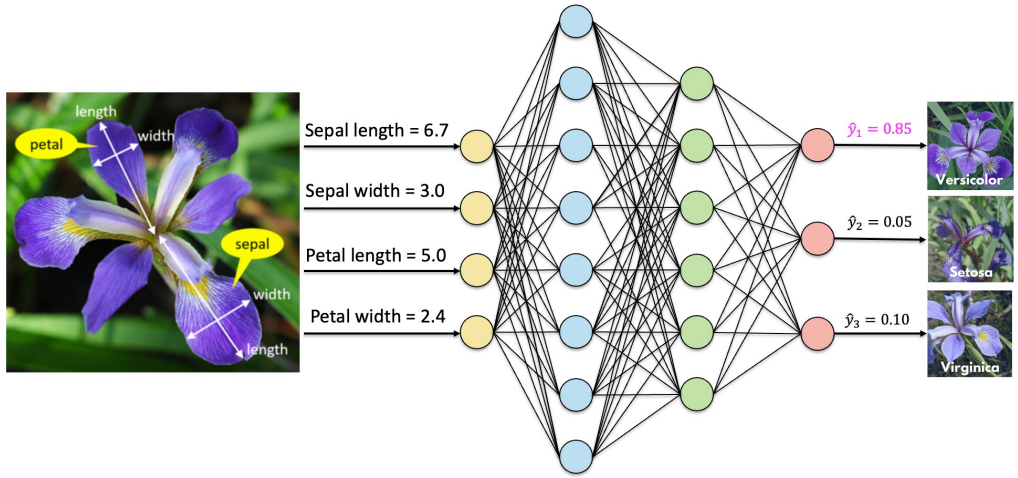
MLPs for Simple Regression and Classification

Regression



- **Boston Housing Dataset**
  - 13 features and **506** records
  - A 3-Layer MLP (13-8-6-1)
  - **No. of Parameters: 173**
    - 13*8+8*6+6*1+8+6+1
  - **Performance: RMSE = 3.97**

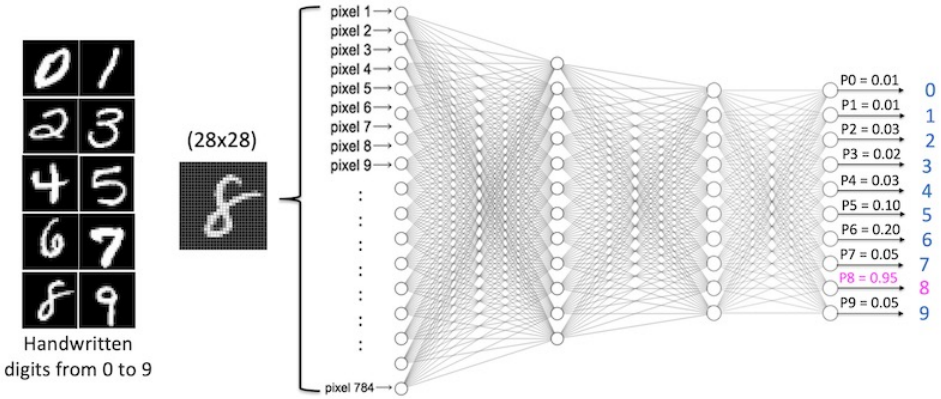
Classification



- **Iris Flower Dataset**
  - 4 features and **150** records
  - A 3-Layer MLP (4-8-6-3)
  - **No. of Parameters: 187**
    - 13*8+8*6+6*3+8+6+3
  - **Performance: 98% Accuracy**

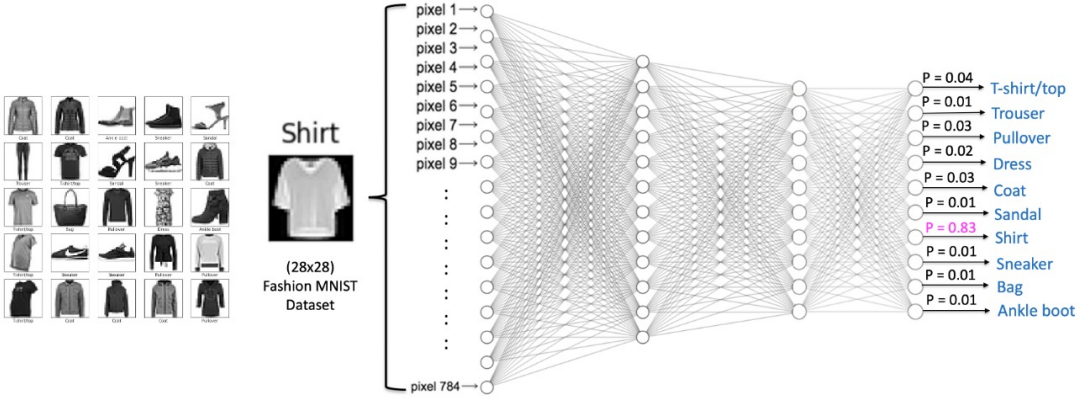
MLPs for Grayscale Image Classifications

Handwritten Digits Recognition



- MNIST Dataset
  - **70,000** 28x28 Grayscale Images
  - 3-Layer MLP (784-128-64-10)
  - **No. of Parameters**: 109.386K
    - 784*128+128*64+64*10+128+64+10
  - **Performance**: **98.36% Accuracy**

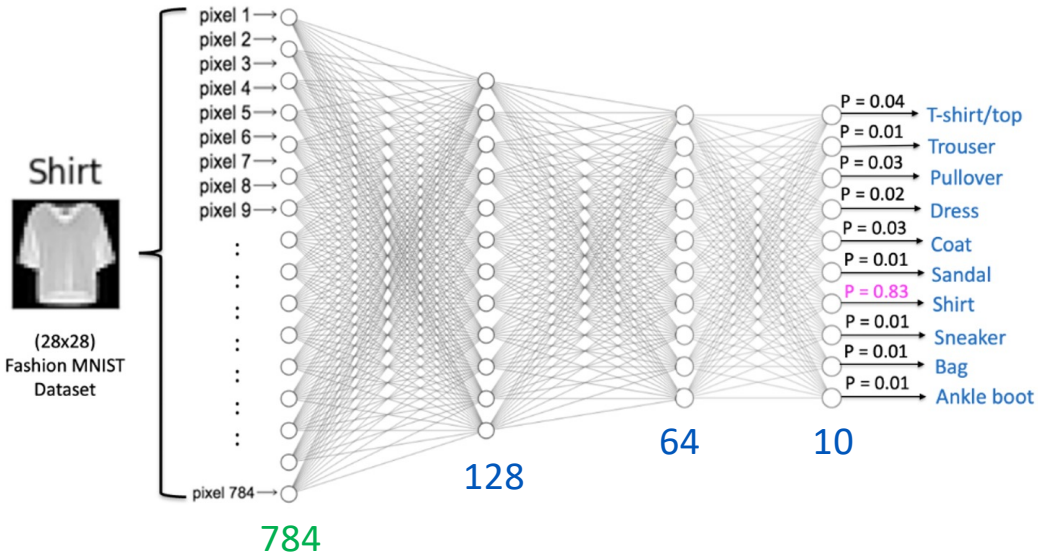
Fashion Image Classification



- Fashion MNIST Dataset
  - **70,000** 28x28 Grayscale Images
  - 3-Layer MLP (784-128-64-10)
  - **No. of Parameters**: 109.386K
    - 784*128+128*64+64*10+128+64+10
  - **Performance**: **84.18% Accuracy**

The Shortcomings of MLPs for Image Data

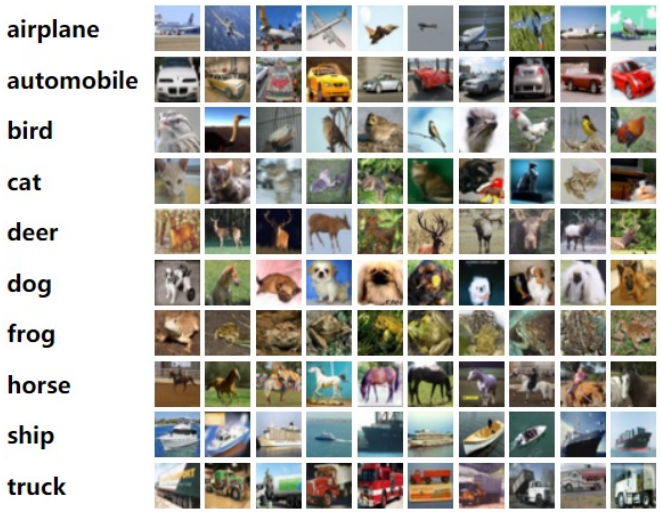
- 2D grayscale or 3D color images need to be **flattened into one-dimensional vectors** as input to the MLP model, which removes spatial image data structure
- The flatten image **input vector dimension is huge**, which results in a large number of model parameters (weights and biases). This leads to longer training and inference times.

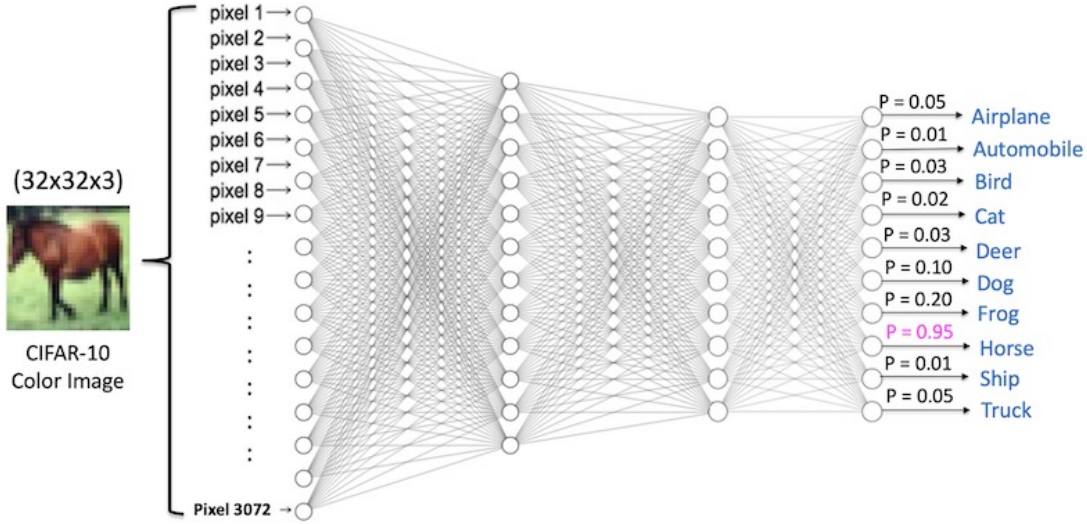


The number of weights
= 784x128 + 128x64 + 64x10
= 109184
The number of biases
= 128 + 64 + 10 = 202

Total number of parameter
= 109184 + 202
= **109,386 (about 109K)**

MLPs for CIFAR-10 Color Image Classification

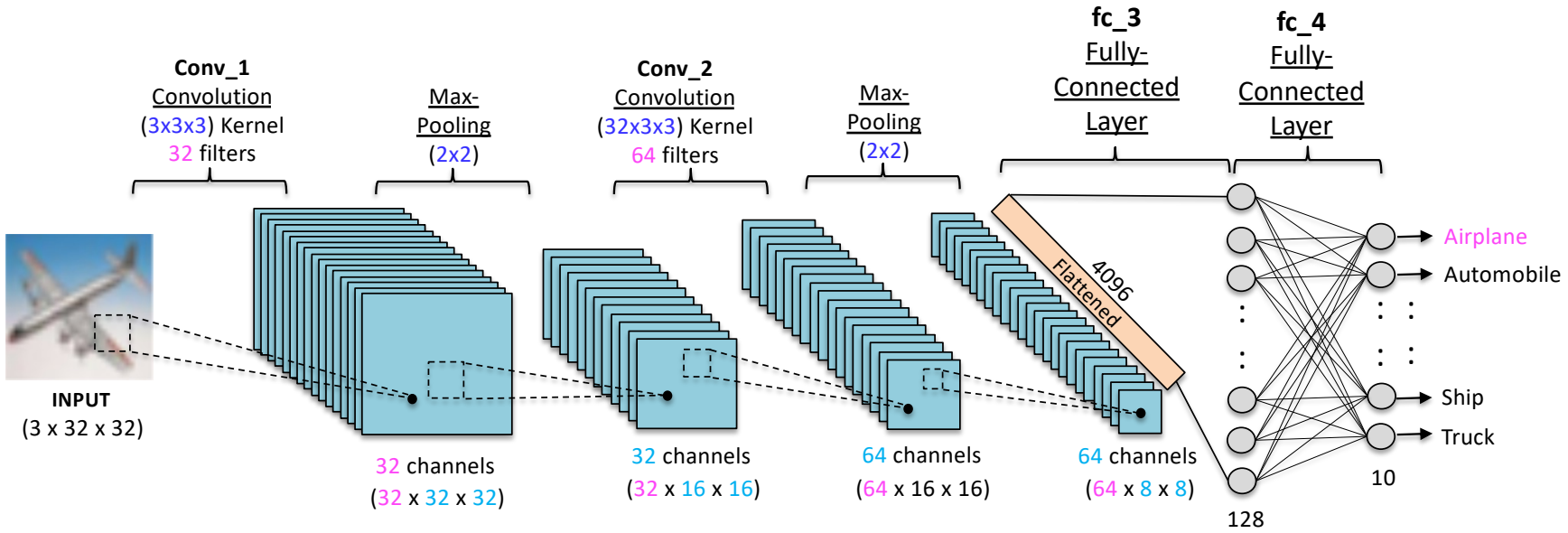




- **CIFAR-10 Color Image Dataset**
  - **60,000** 32x32x3 RGB Color Images
  - 5-Layer MLP (3072-1536-768-384-128-10)
  - **No. of Parameters**: 6,246,410 **(6.2M)**
  - **Performance**: **55% Accuracy**

An MLP with 3-5 hidden layers and 500+ units per layer should achieve **65%+ accuracy** on CIFAR-10 with proper tuning and regularization.

Convolutional Neural Network for CIFAR-10 Image Classification



https://colab.research.google.com/drive/1W-CpyU3mwWr_ueSm_86C-do6pm361VMe?usp=sharing

- **CIFAR-10 Color Image Dataset**
  - 60,000 32x32x3 RGB-Color Images
  - **4-Layer CNN** (3x3-32, 3x3-64, 4096-128-10)
  - **No. of Parameters**: 545,098
  - **Performance**: **78% Accuracy**

A simple Convolutional Neural Network (CNN) can achieve **70-80% accuracy**. State-of-the-art is **above 97%.**

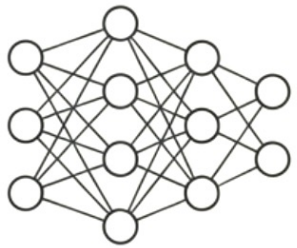
Comparing MLP and CNN for Image Classification

- MLPs are simple models that can work reasonably well only on simple image classification problems such as MNIST dataset.
- In practice, **Convolutional Neural Networks (CNNs)** tend to work better than MLPs for complex image classification tasks:
  - **Local connectivity** - Captures spatial structure
  - **Weight sharing** - Reduces parameters and provides translation invariance
  - **Hierarchical feature learning** - Learns from low-level to high-level
  - **Translation invariance** - Robust to feature location
  - **Efficient computation** - Fewer overall parameters, GPU parallel processing
  - **Inductive bias** - **Architecture suited for images**

Recap: Network Architecture & Inductive Bias

Selecting the right hypothesis space for the data

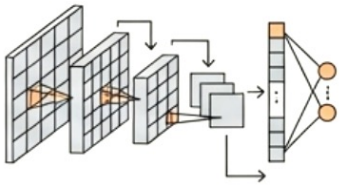
MLP (Multi-Layer Perceptron)



Inductive Bias:
Independence

Use Case:
Tabular Data

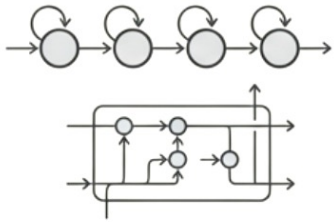
CNN (Convolutional Network)



Inductive Bias:
Spatial Locality & Invariance

Use Case:
Image Data

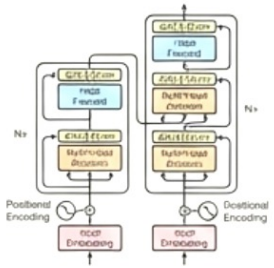
RNN / LSTM



Inductive Bias:
Sequentiality

Use Case:
Time-series, Sequence Data

Transformer



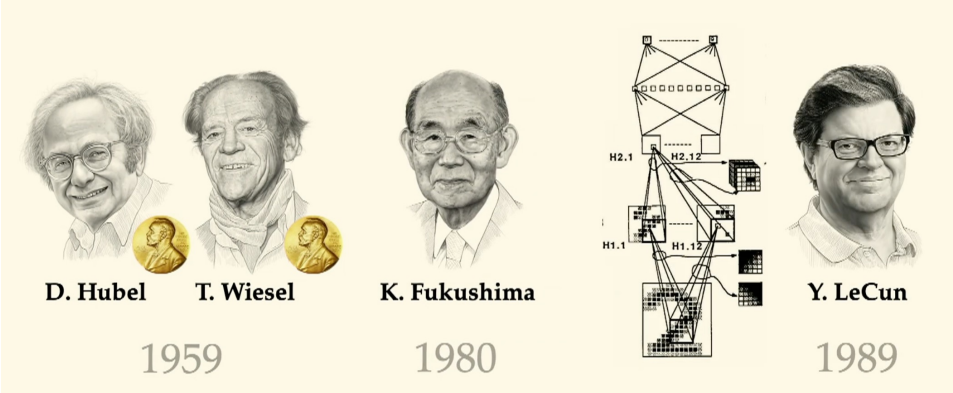
Inductive Bias:
Global Context (Self-Attention)

Use Case:
NLP, Seq2Seq, Vision

Data Types and Neural Network Architectures

- **Tabular Data**: Initially, the focus was on utilizing **Multilayer Perceptrons (MLPs)** to process tabular data. This approach evolved into deep learning, increasing the model's capacity to capture complex patterns by adding more layers.
- **Image Data**: **Convolutional Neural Networks (CNNs)** emerged to interpret and analyze visual information in grid formats, outperforming MLPs.
- **Sequential Data**: Sequences with meaningful order (e.g., textual or time-series data) require specialized models, which led to the development of Recurrent Neural Networks (RNNs), which can model and learn from sequential patterns.
- **Seq2Seq Data**: Specialized architectures were created to handle sequence-to-sequence data, such as machine translation tasks, due to the complexities involved in aligning variable-length input and output sequences.

Intuition of Convolutional Neural Networks



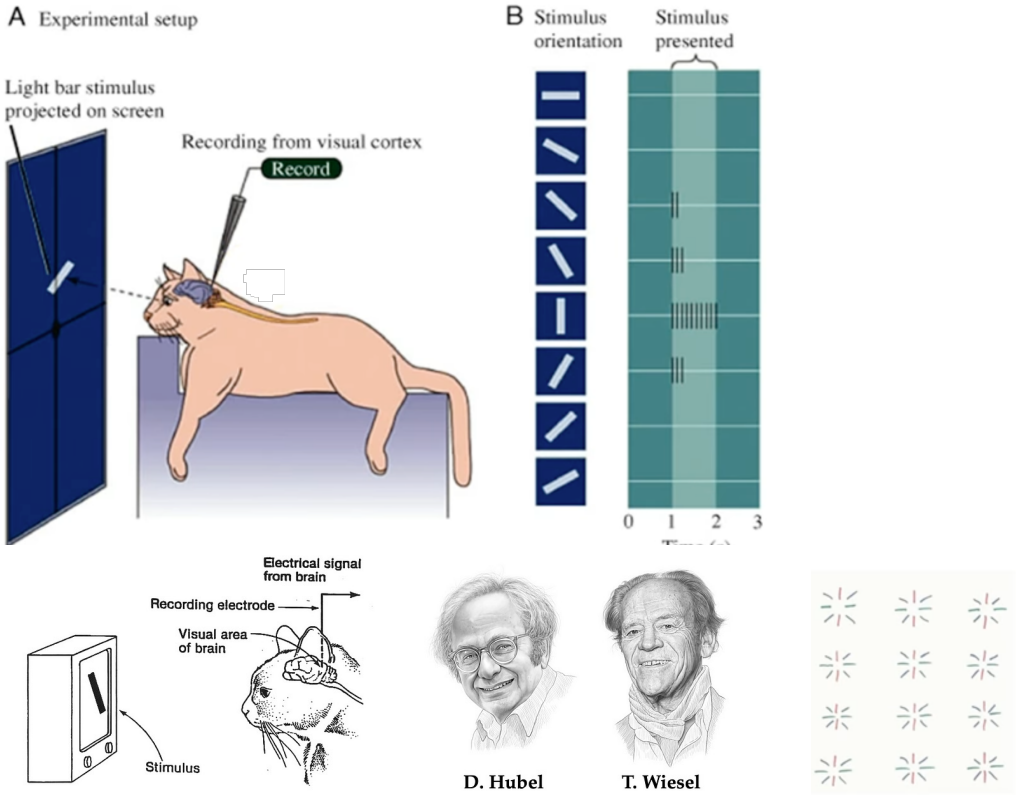
Hubel and Wiesel's Experiments (1950s)

Discovery: Simple Cells (orientation) and Complex Cells (motion).

Hubel and Wiesel's experiments was focused on studying the responses of neurons in the visual cortex to various visual stimuli, such as lines, edges, and motion.

- They discovered that the brain does not process images all at once.
- Instead, individual neurons fire only for specific patterns — lines, edges, and corners—located in a specific part of the visual field.
  - **Simple Cells (orientation)**
  - **Complex Cells (motion)**

https://www.youtube.com/watch?v=QsikPDDxy4g



The Nobel-winning physiologists David Hubel and Torsten Wiesel and the depiction of their classical experiment revealing the structure of the visual cortex. Portraits: Ihor Gorsky. (Source)

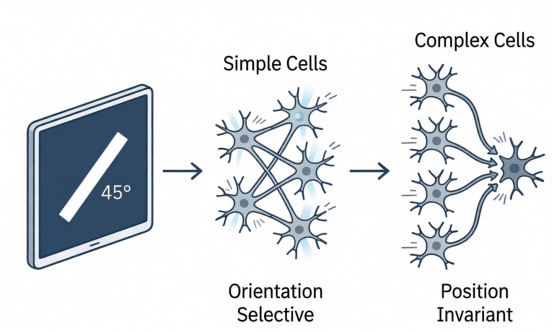
Hubel and Wiesel Cat Experiment

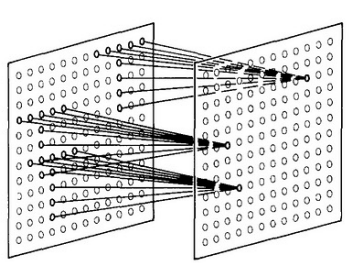
https://www.youtube.com/watch?v=IOHayh06LJ4

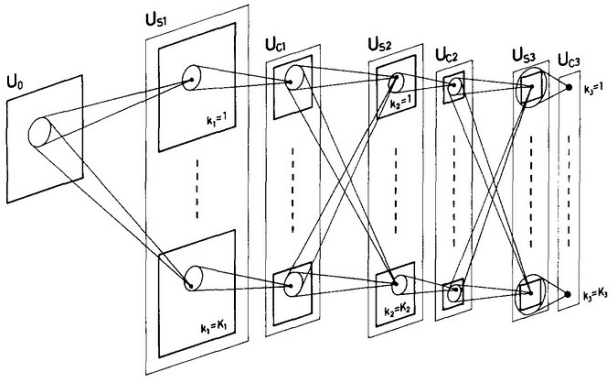
The cat experiment showed that cells are sensitive to orientation of edges.

The First Blueprint: The Neocognitron (1980)

Kunihiko Fukushima's alternating S-Cell/C-Cell structure. (Source)





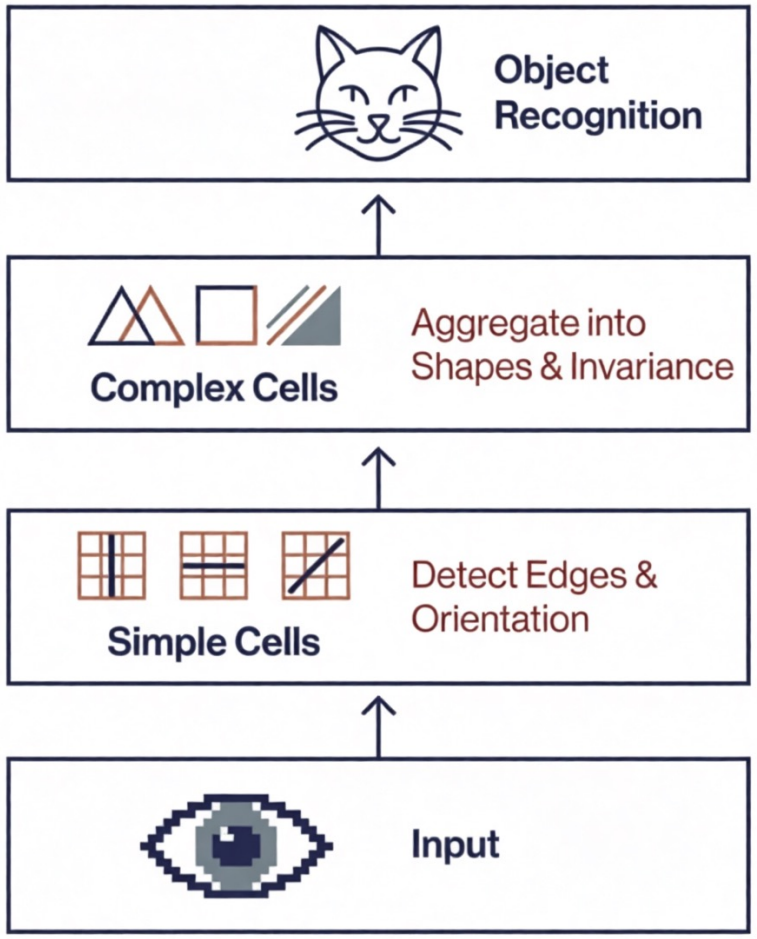


K. Fukushima

- **Context:** Developed by Kunihiko Fukushima, this was the first neural network directly inspired by Hubel and Wiesel's findings.
- **The Architecture:**
  - **S-Cells** (Simple): Performed feature extraction (template matching).
  - **C-Cells** (Complex): Provided invariant pattern recognition (recognizing a shape even if it shifted slightly).
- **Legacy:** While limited by the hardware of 1980, the Neocognitron demonstrated that hierarchical feature extraction could be simulated mathematically.

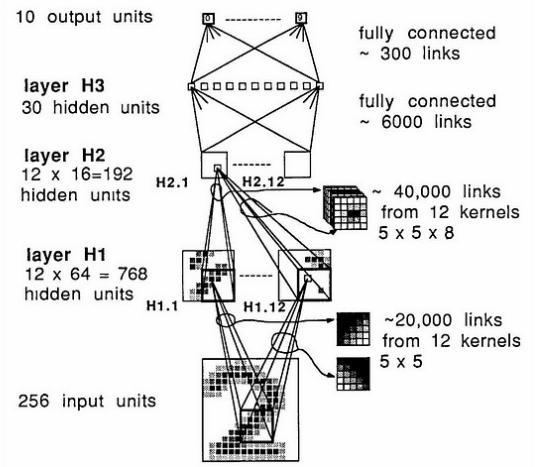
The Hierarchy of Sight

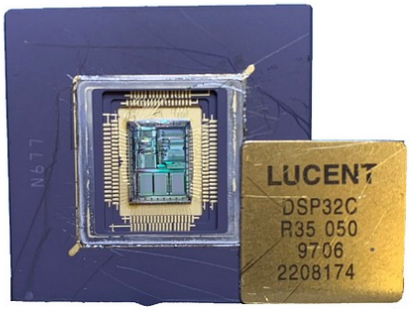
- Vision is a bottom-up process.
- We build complex understanding from simple



Introduction of CNNs (1989)

- In 1989, Yann LeCun's team introduced CNNs, revolutionizing image processing by preserving images' 2D nature and processing information spatially.
- **CNNs used spatial filtering to extract spatial patterns, allowing them to learn hierarchical representations of features.**



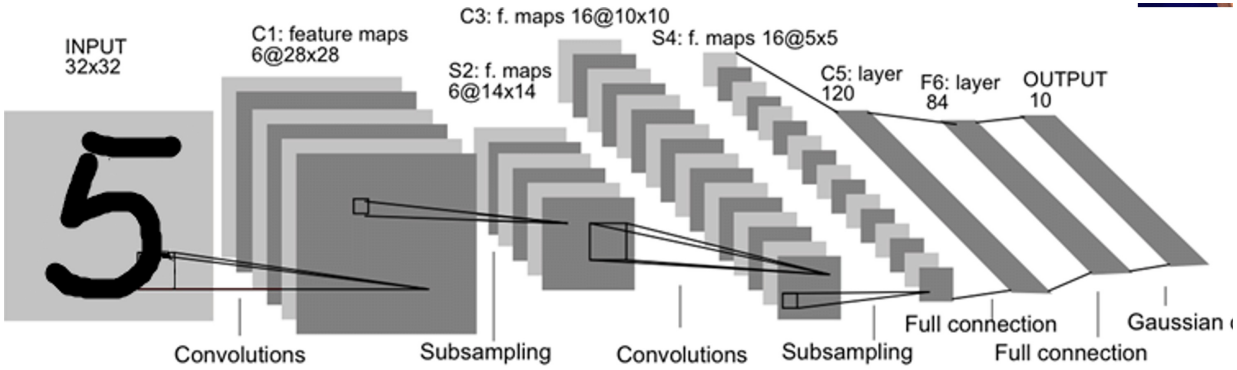


Y. LeCun

LeCun's paper demonstrated how to train nonlinear CNNs from scratch using backpropagation, paving the way for efficient image classification tasks. (Source)

LeNet-5 (1998)

- In 1998, Yan LeCun and his team made a significant contribution to the field of deep learning with the publication of the **LeNet-5 model, a pioneering CNN architecture that consisted of multiple convolutional and pooling layers**.



LeCun, Bottou, Bengio, Haffner: Gradient-based learning applied to document recognition. Proceedings of the IEEE, 1998.

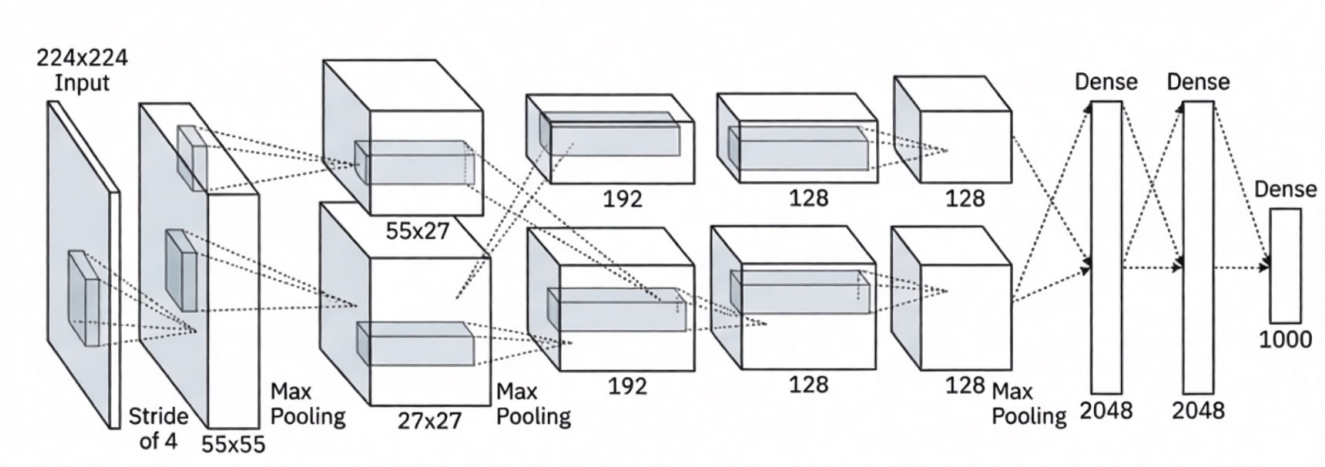
The Convergence: Igniting the Golden Era (2010s)

- **The Catalyst:** The 2010s marked a resurgence sparked by three converging factors:
  1. **Big Data**: The release of ImageNet (2009) and the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) challenge provided 14 million hand-annotated images needed for deep learning.
  2. **Hardware**: The arrival of powerful GPUs capable of parallel matrix operations made training feasible.
  3. **Algorithms**: Deeper architectures and better activation functions (ReLU) solved the vanishing gradient problem.

The 2010s marked the "Golden Era" of CNNs, sparked by the release of the large-scale **ImageNet dataset (2009)** and the **ImageNet Large Scale Visual Recognition Challenge (ILSVRC)** from 2010 to 2017.

GPUs

The Game Changer: AlexNet (2012)



16.4%
Top-5 Error Rate (Winner)

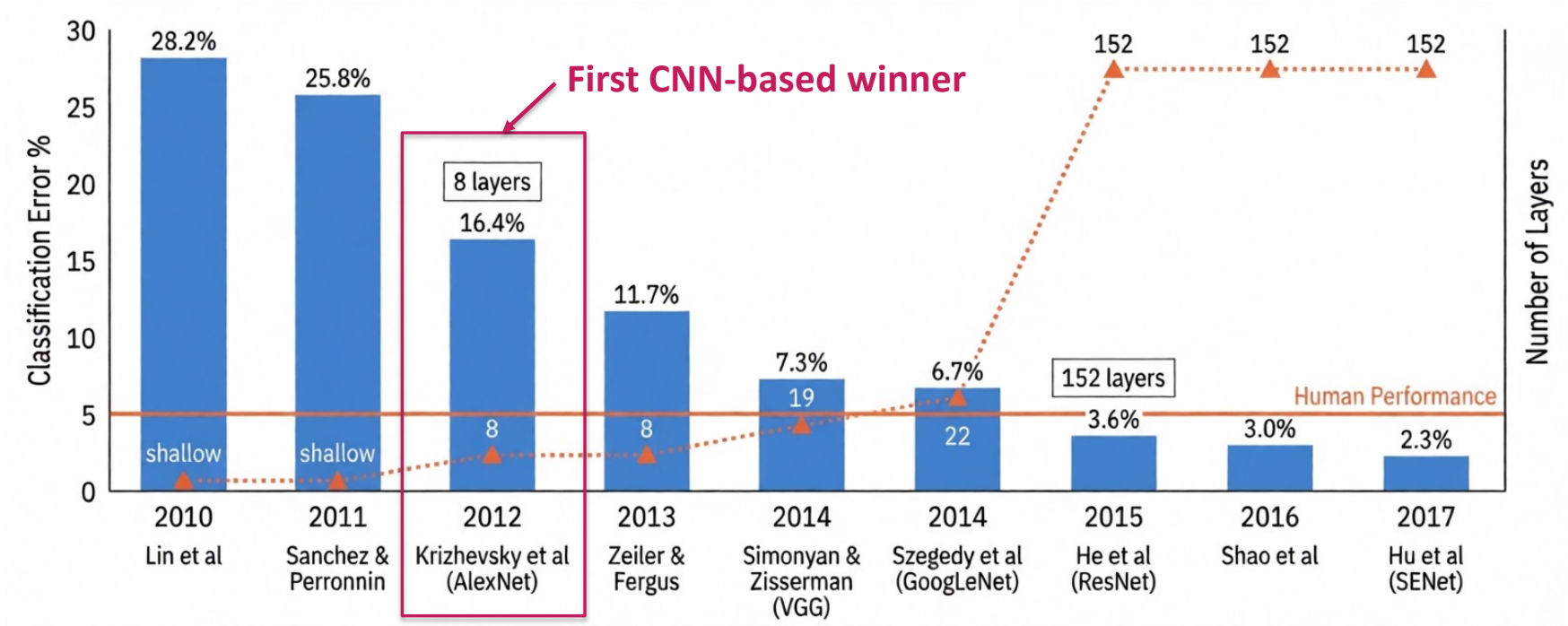
26.2%
Runner-up Error Rate

8 Layer
CNN Depth

Legacy: AlexNet (Krizhevsky et al.) didn't just **win ILSVRC 2012**; it decimated the competition, nearly halving the error rate. It introduced Rectified Linear Units (ReLU) and Dropout, launching the modern Deep Learning revolution.

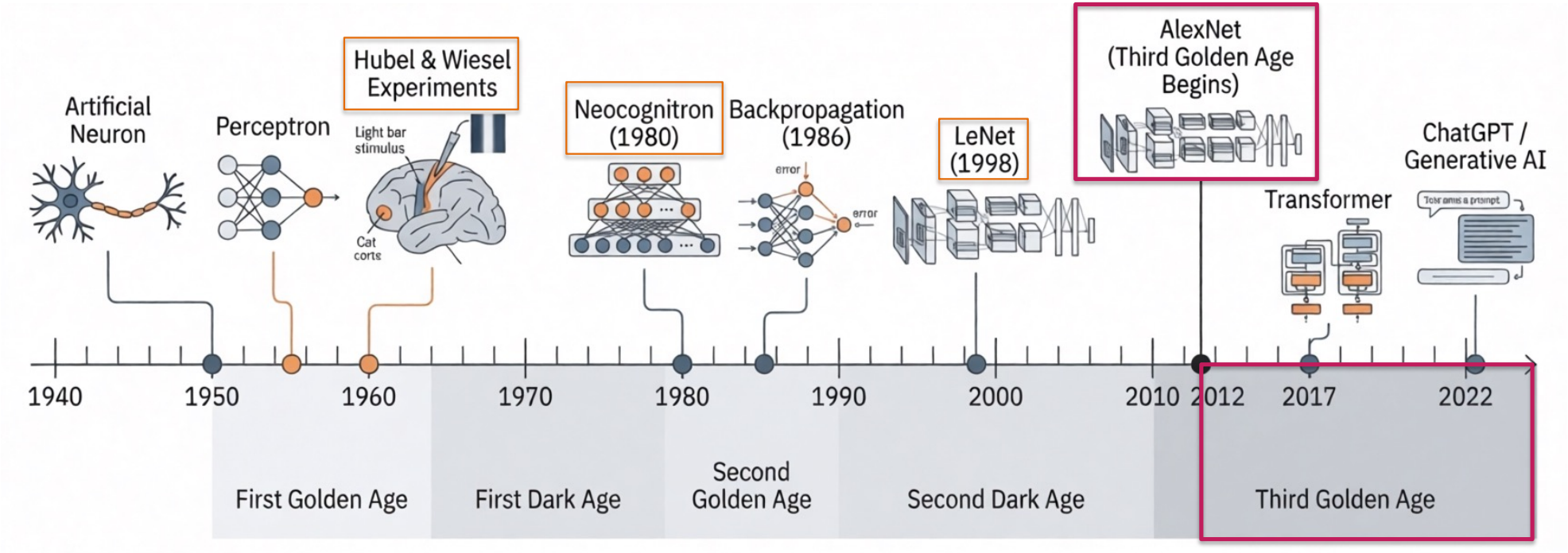
The Explosion of Depth

Winners of ILSVR Challenge (2010 - 2017)



Following AlexNet, models became exponentially deeper to capture more complex features.
By 2015, ResNet (152 layers) surpassed human-level performance.

AlexNet Sparks the Third Golden Age of Neural Networks



The current golden age (2012 - present) is marked by the convergence of deep learning, big data, and powerful computing platforms. This era has seen remarkable breakthroughs in image recognition, natural language processing, and robotics. Ongoing research continues to push the boundaries of AI capabilities.

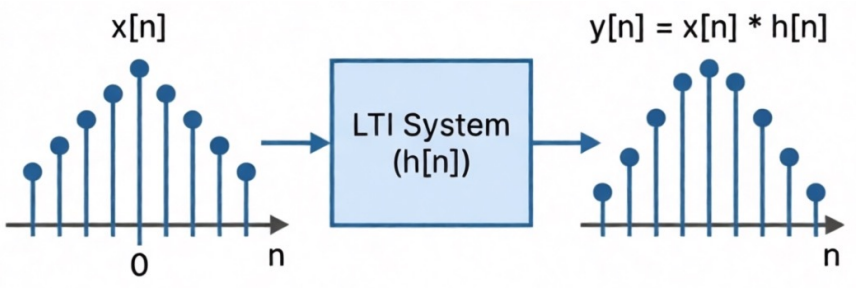
Convolution Operations

Foundations in Signal Processing

The Roots: LTI Systems

In discrete Linear-Time Invariant (LTI) systems, the output sequence $y[n]$ is determined by the convolution of input $x[n]$ and unit impulse response $h[n]$.

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[n-k]h[k]$$

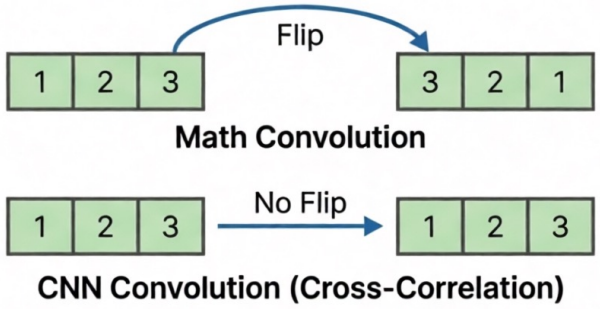


The Crucial Difference: Flipping

In traditional mathematics, the kernel is "flipped" (mirrored) to preserve commutative properties.

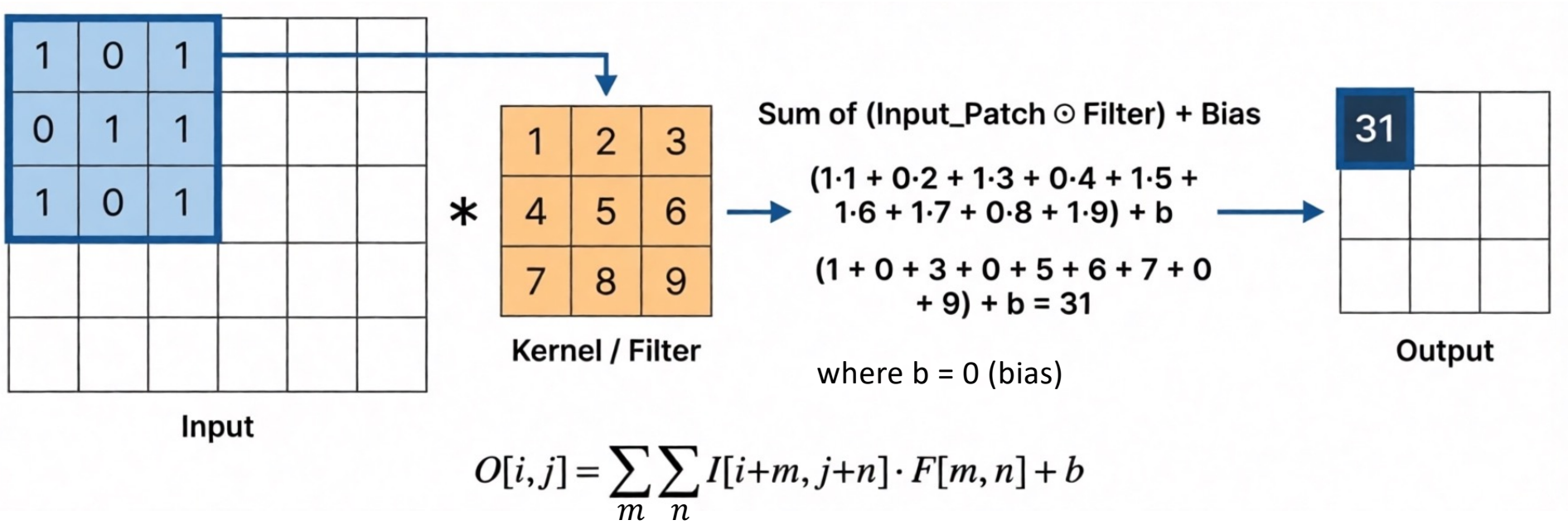
In Deep Learning, we do NOT flip the kernel.

Why? Images lack temporal causality (no "past" or "future"). Since weights are learned end-to-end, the network learns the kernel in whichever orientation is required.



The 2D Convolution Operation

The Sliding Window Mechanism



$$O[i,j] = \sum_m \sum_n I[i+m, j+n] \cdot F[m,n] + b$$

The filter slides over the input image (local receptive field), performing an element-wise multiplication and sum at every position to generate the feature map.

2D Filter Convolution

1	0	1
0	1	0
1	0	1

Weights of the
3x3 Filter (or Kernel)

$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	0	0
$0_{\times 0}$	$1_{\times 1}$	$1_{\times 0}$	1	0
$0_{\times 1}$	$0_{\times 0}$	$1_{\times 1}$	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved
Feature

2D Filter Convolution

0	1	0
1	0	1
0	1	0

Weights of the
3x3 Filter (or Kernel)

1	1	1	1	0
0	0	1	1	0
0	0	1	0	1
0	0	1	1	1
1	1	0	1	0

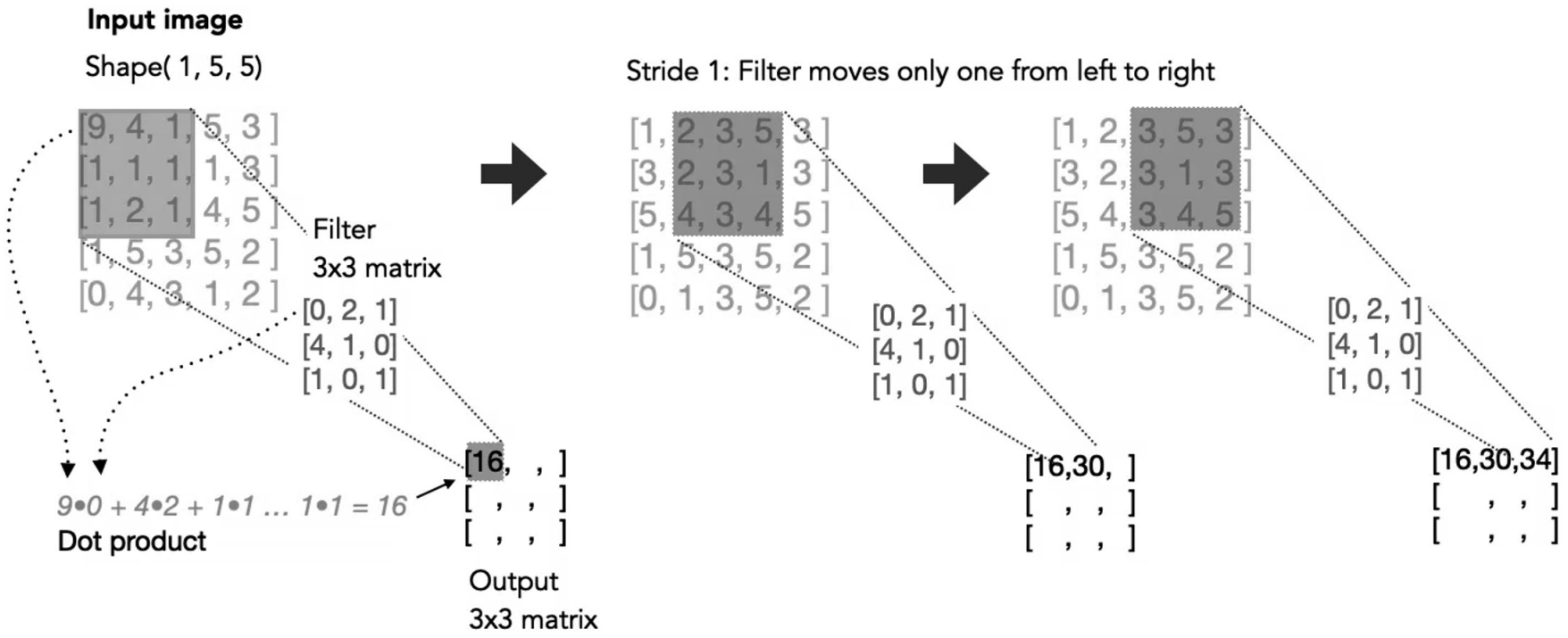
Image

2	3	2
1	2	3
3	2	4

Convolved
Feature

2-D Kernels (Filters)

- In case of 2D data (grayscale images), the convolution operation between an input 5x5 image and a 3x3 kernel (or fitter):



The Role of Filters: Feature Detection

Input Image (Vertical Edge)

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

Represents an image with a sharp vertical boundary.

Vertical Edge Filter

1	0	-1
1	0	-1
1	0	-1

*

=

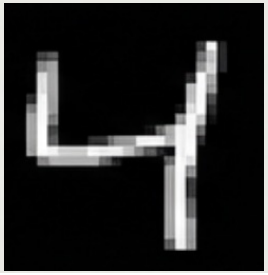
Feature Map (Edge Detected)

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

Why does this work?

The filter responds positively when there is a contrast transition from left to right.

If we rotated this filter 90 degrees (*Horizontal Edge Detector*), the result on this specific image would be all zeros.



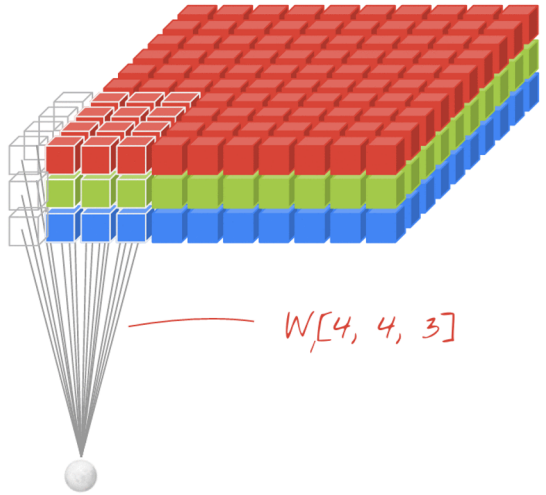
Filters Define the Features

- Changing the weights inside the kernel changes what the network sees.

Identity	Sharpen	Edge Detection
0 0 0 0 1 0 0 0 0	0 -1 0 -1 5 -1 0 -1 0	-1 -1 -1 -1 8 -1 -1 -1 -1
		

Convolution with 3D Kernel

- In the 2D case, we slide a two-dimensional filter over a two-dimensional input (grayscale image)
- What would happen **in the 3D case** where **the input images are in color (RGB)**?
- **We need to use 3D Filter Convolution**
- Steps:
  - Compute the dot product for each channel (same as 2D)
  - Sum over each channel
- Note: The depth of the filter is always the same as the depth of the input image



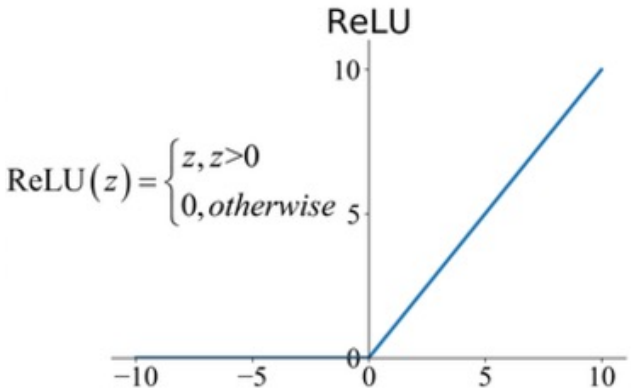
⚠ W_1 and W_2 are **distinct** 4 x 4 x 3 filters

Activation Layer: ReLU

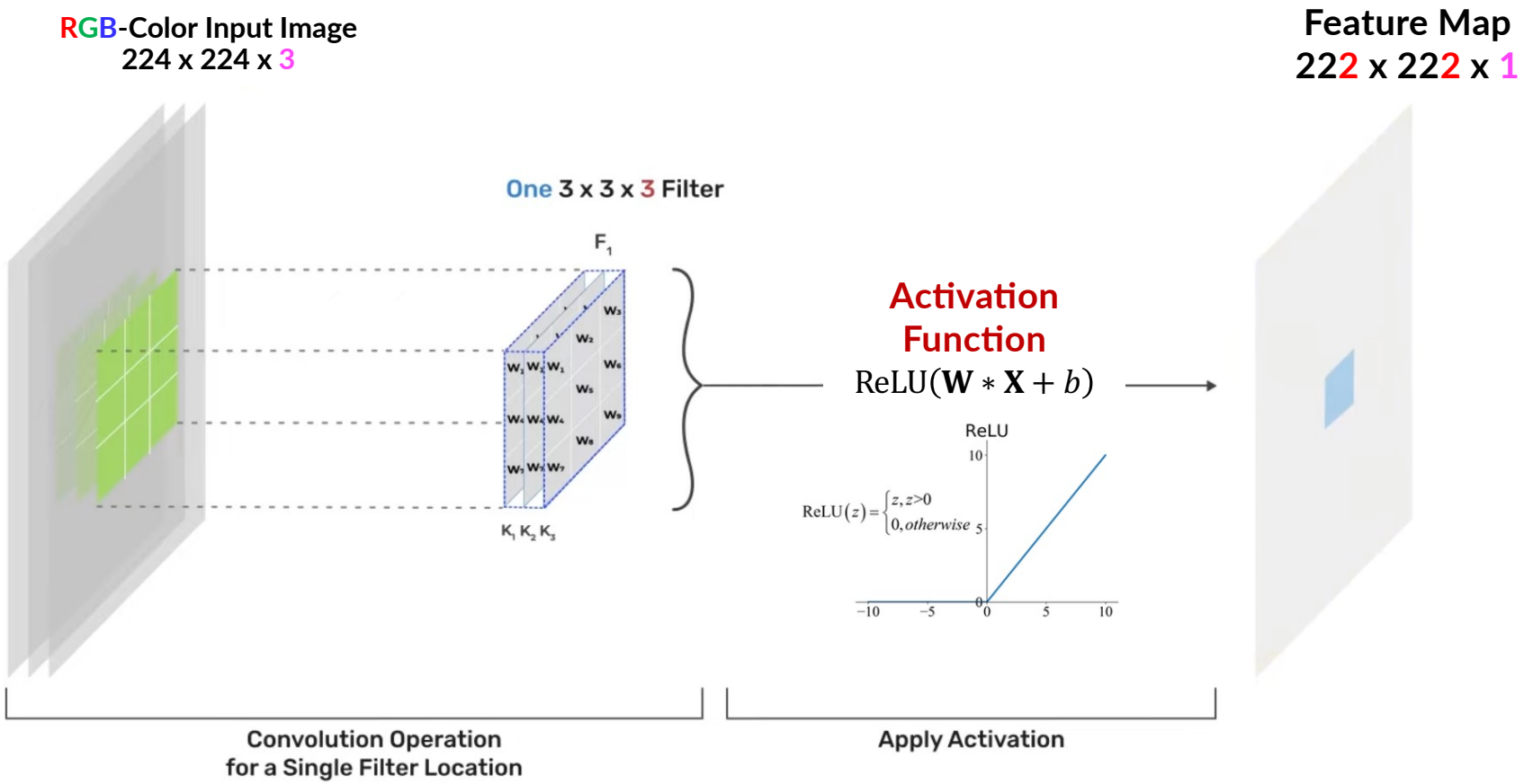
- After the convolution operation, **bias** and **an activation function**, often Rectified Linear Unit (ReLU), is applied element-wise to introduce non-linearity into the model.

$$O[i,j] = \text{ReLU}\left(\sum_{m=0}^{K-1}\sum_{n=0}^{K-1}\sum_{c=0}^{C-1} I[i+m, j+n, c] \cdot F[m,n,c] + b\right)$$

  - ReLU helps the network learn complex relationships and makes the model more expressive.
  - It completely depends upon your use case which activation you will use, in most cases researchers use ReLU, there some activations which can also be used, for example: Leaky ReLU, ELU.



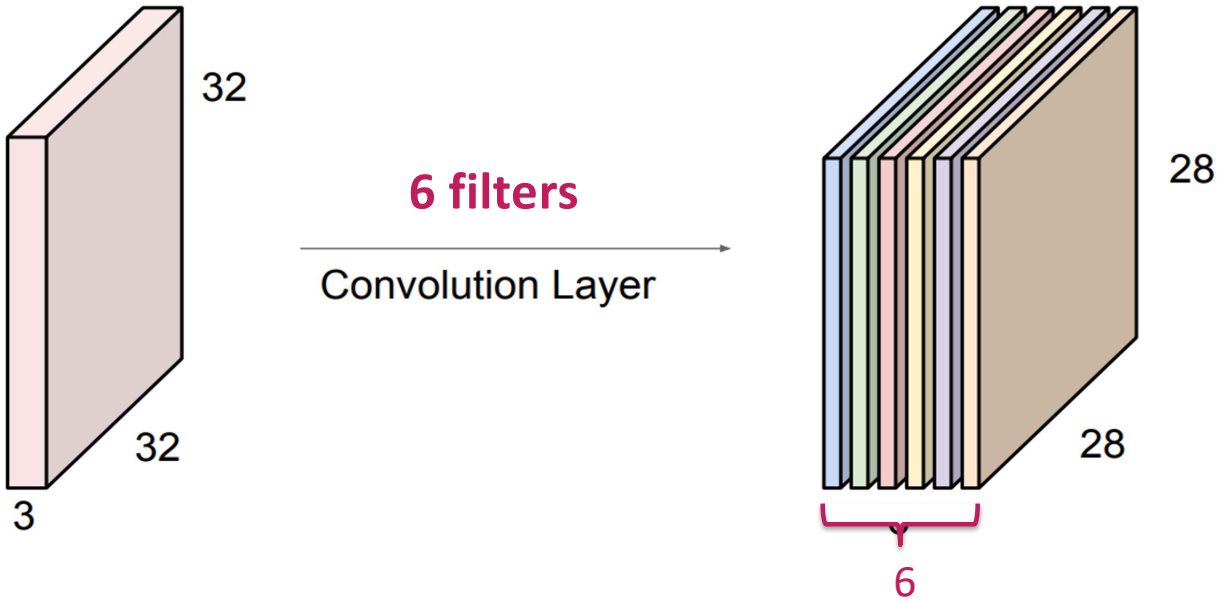
Convolutional Layers



https://www.youtube.com/watch?v=N15mjfAEPqw

Multiple Filters: Stacking the Feature Maps

- For example, if we had six 5x5x3 filters, we would get **6 separate feature maps**:



A layer doesn't produce one map; it produces a volume. If you apply 6 filters, you get a stack of 6 feature maps. Each channel in this volume represents a different learned feature (e.g., Channel 1: Vertical Edges, Channel 2: Color Gradients).

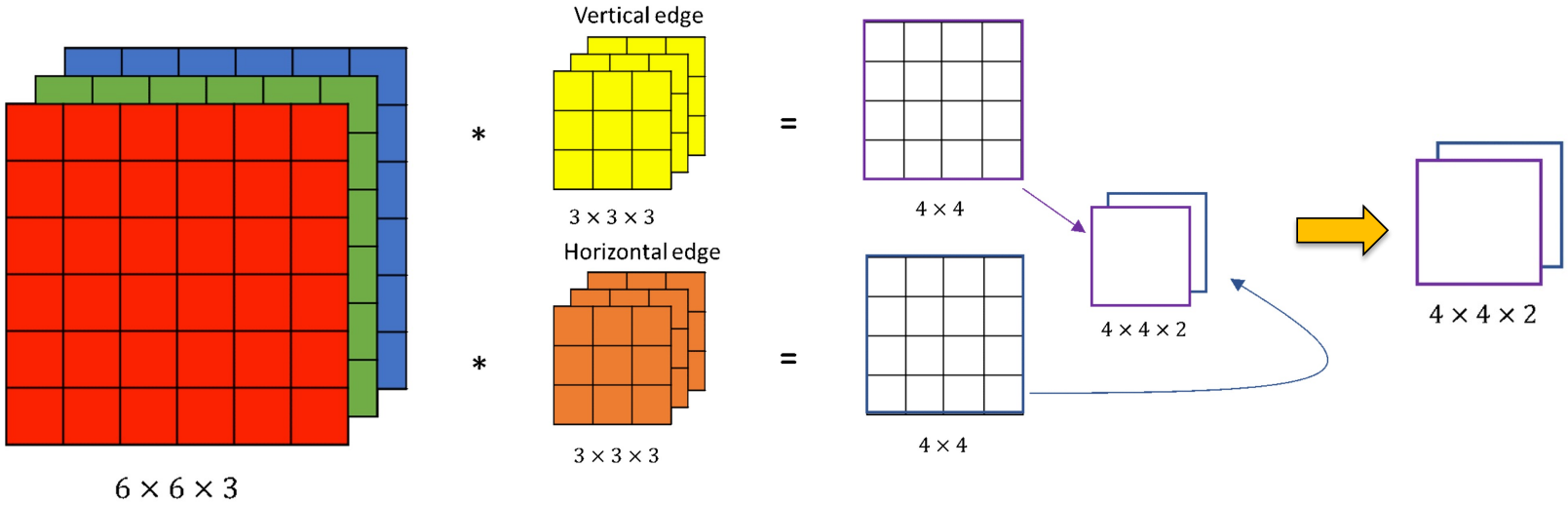
Example 3-D Convolution for RGB Image

Input
RGB image 6x6x3

Convolutional Layer with
2 Filters of size 3x3x3

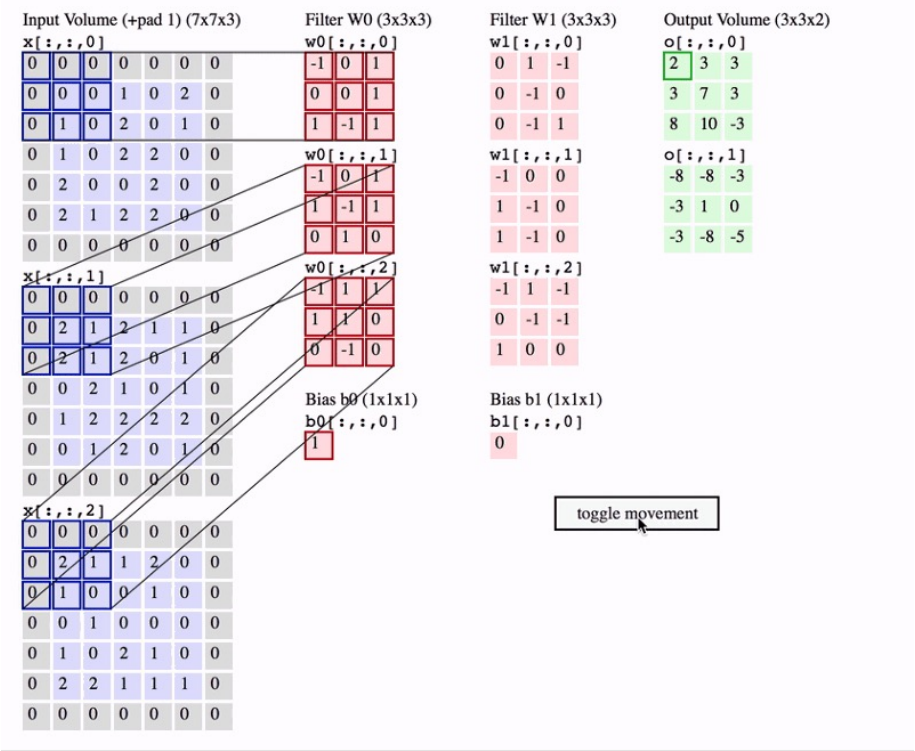
ReLU

Output
Feature map 4x4x2

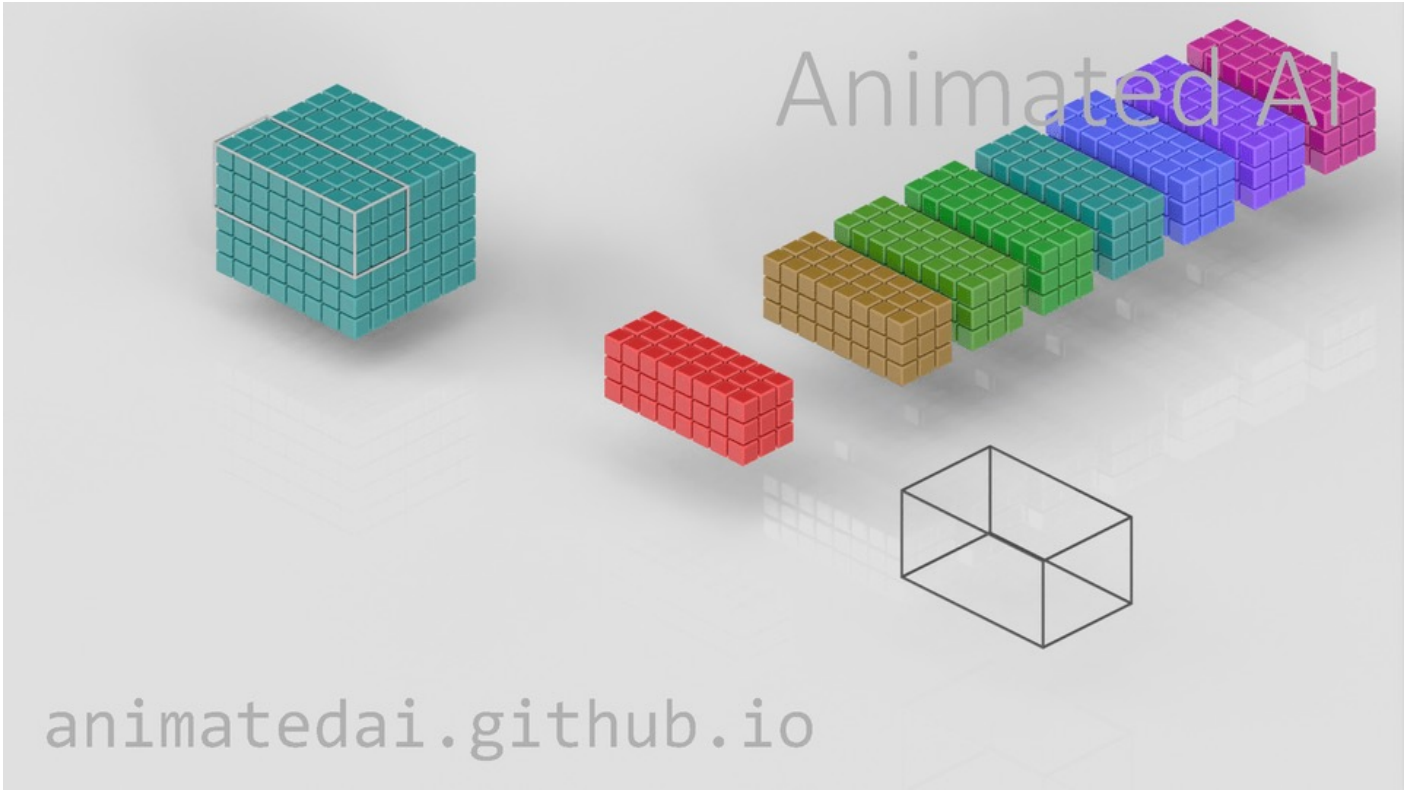


Convolution Operation Animation

- In practice, a convolution layer can consist of multiple filters that operate on multiple channels of input data. For example, as depicted below with an input of size 7x7x3, two filters can be applied, each extracting distinct feature maps by convolving across the three input channels.



Convolution Operation Animation



Convolution Operation (Source)

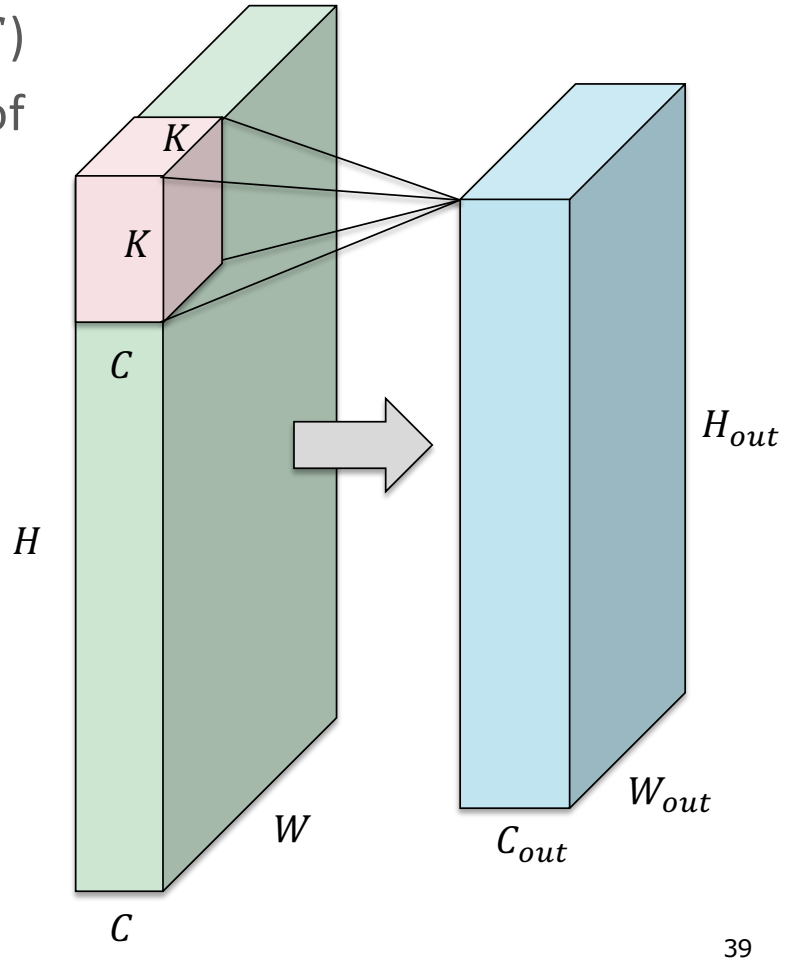
Convolution: Hyperparameters

- **Input dimension**: Width (W) x Height (H) x Depth (C)
- **Spatial extent** (K) of each filter's kernel (the depth of each kernel is same as the channels of input)
  - Kernel dimension = $K \times K \times C$
- **Output dimensions** is $W_{out} \times H_{out} \times C_{out}$
- **Stride** (S)
- **Number of Filers** $F = C_{out}$
- **Padding** (P)

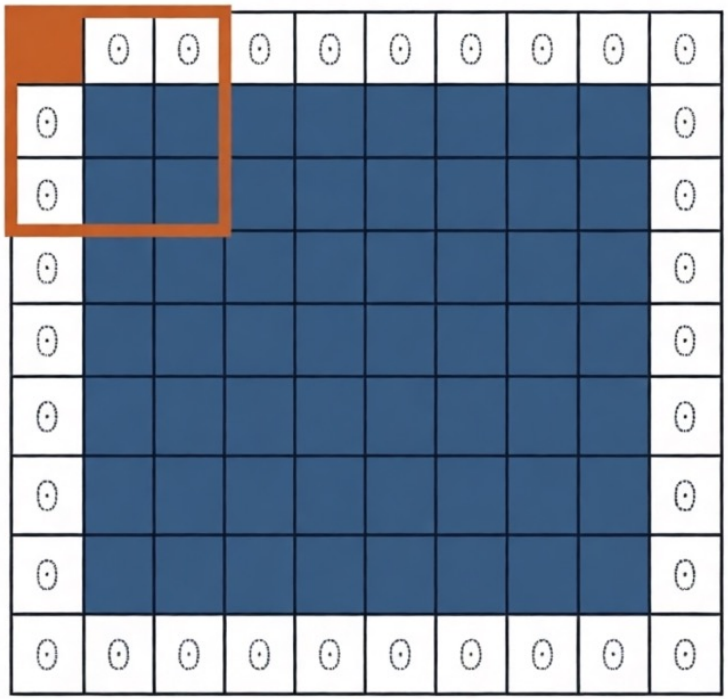
$$H_{out} = \left\lfloor \frac{H - K + 2P}{S} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W - K + 2P}{S} + 1 \right\rfloor$$

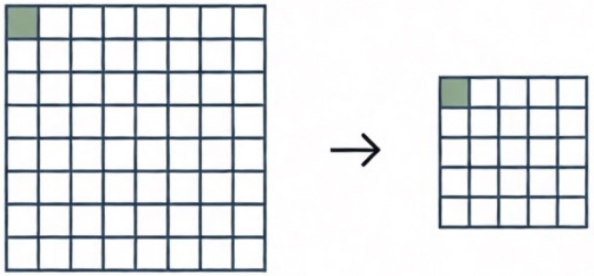
$$C_{out} = F$$



Preserving Geometry with Padding (P)



Zero Padding ($P = 1$)



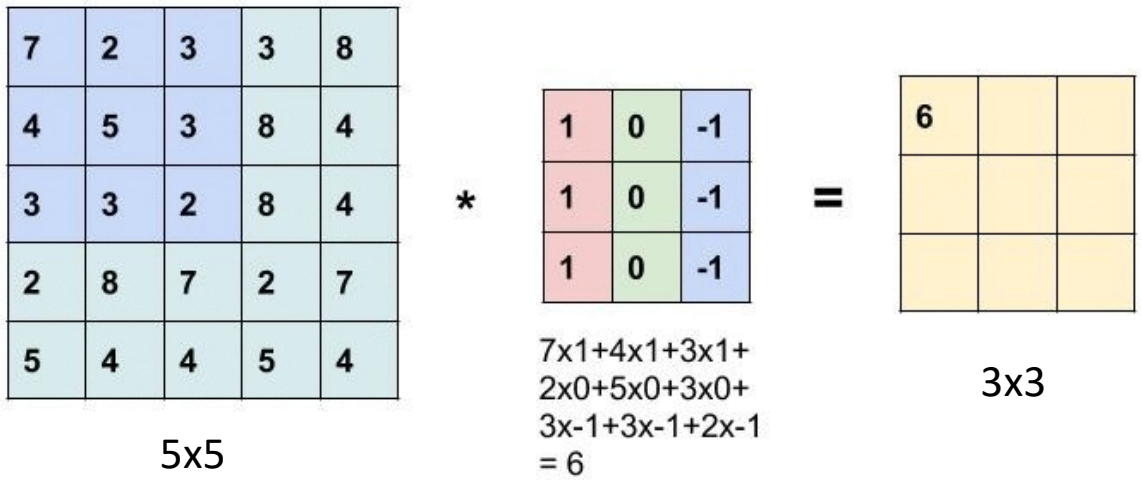
Without Padding Shrinking Output

Valid Padding: No padding.
Image shrinks with every layer.

Same Padding: Pad to ensure
Output Size = Input Size.

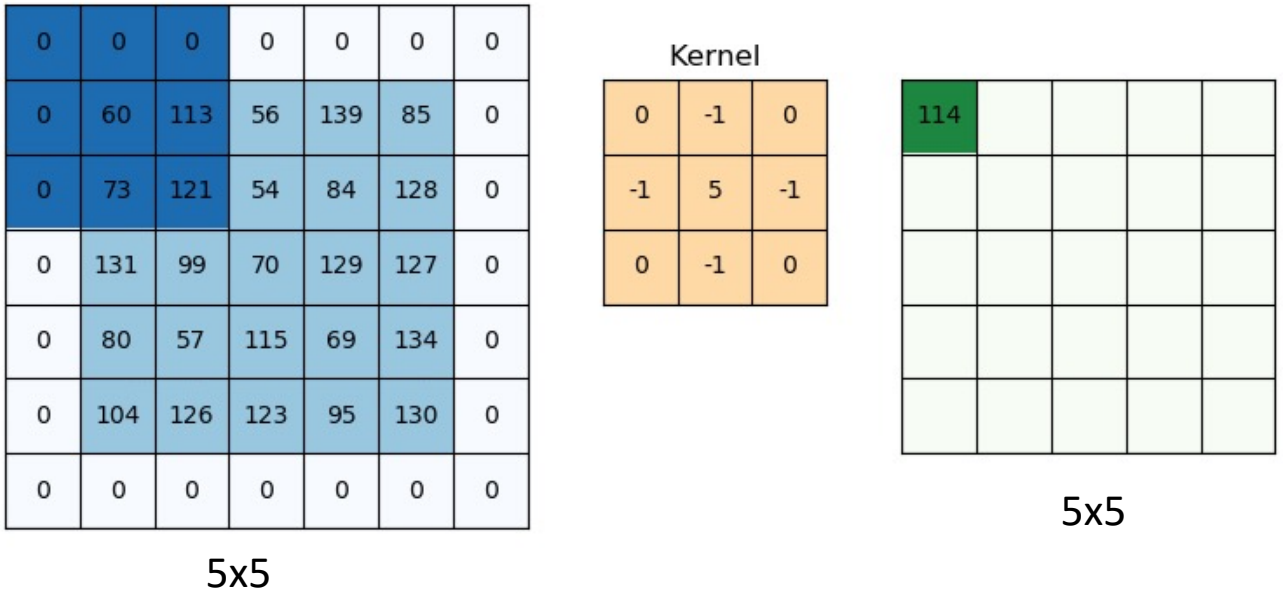
Valid Padding Example

- No padding is applied, resulting in a smaller output feature map.
  - Example: Convolution of a filter over a 2D image without padding



Same Padding Example

- Padding is added so that the output feature map has the same spatial dimensions as the input.
  - The convolution of a filter over a 2D image with 1-pixel padding



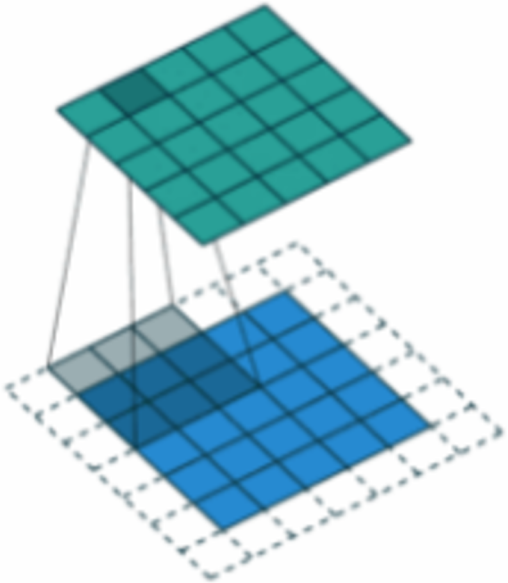
Padding Examples

- Convolving an image with a filter results in a block with a smaller height and width — what if we want the height and width as before?

Output size: 2x2
smaller

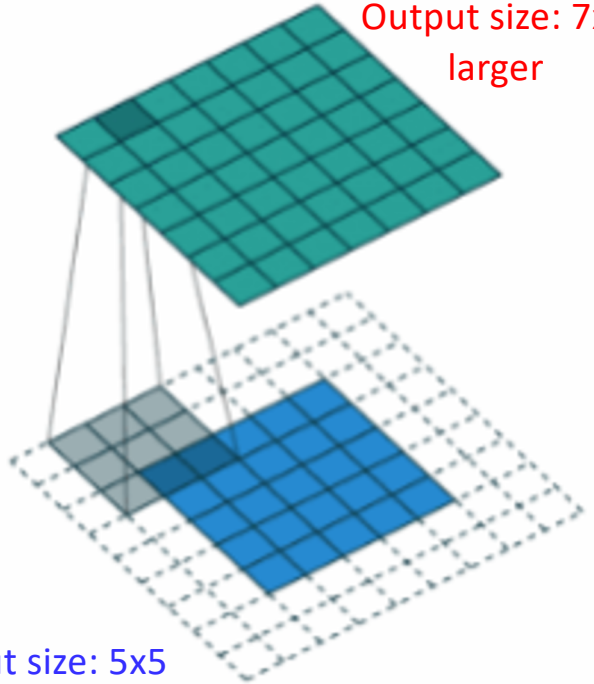
Input size: 4x4

Output size: 5x5



Input size: 5x5
Padding with 1 pixel

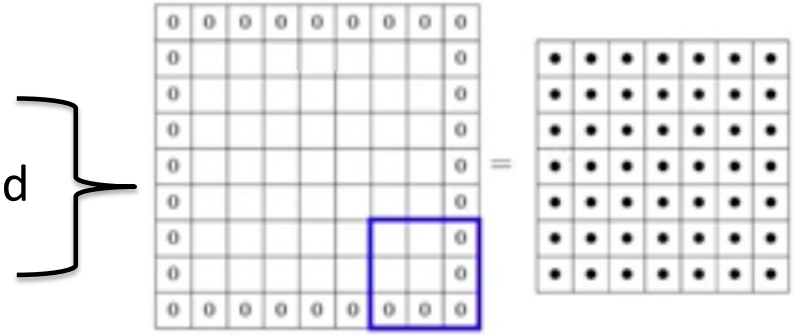
Output size: 7x7
larger



Input size: 5x5
Padding with 2 pixels

Padding: Preserving Spatial Information

- Convolving an image with a filter results in a block with a smaller height and width — what if **we want the height and width as before**?
- **Pad inputs with appropriate number of inputs** so **you can now apply kernel at corners**
- Let us use pad $P = 1$ with a 3x3 kernel
  - This means we will add one row and one column of 0 inputs at the top, bottom, left and right.
- The (H_{out}, W_{out}) formula can be modified as

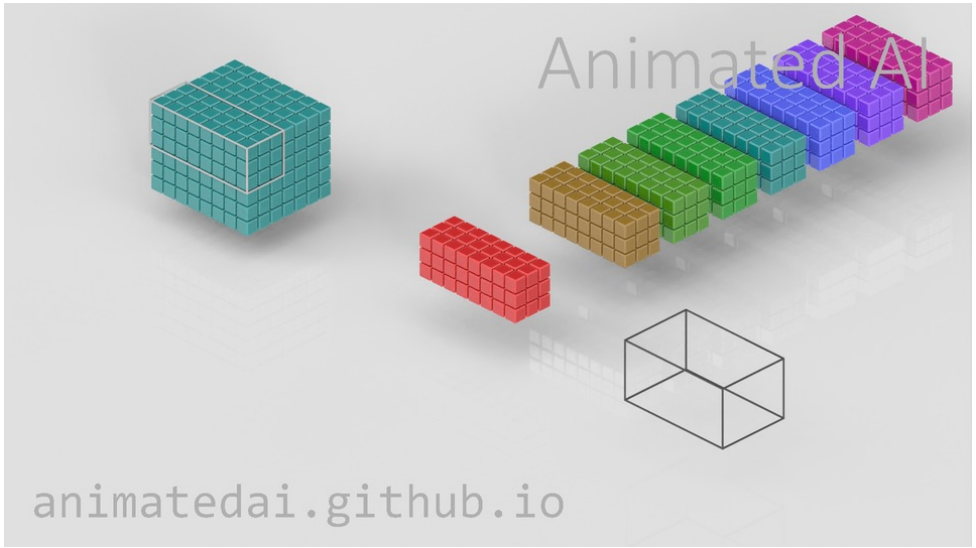


$$H_{out} = H - K + 2P + 1$$

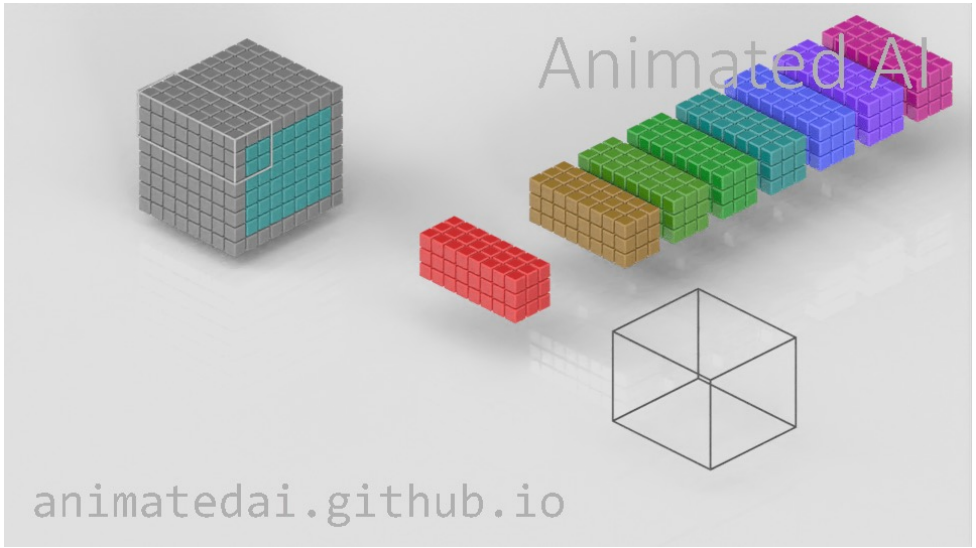
$$W_{out} = W - K + 2P + 1$$

Padding Animations

No Padding (aka Valid)



Same Padding



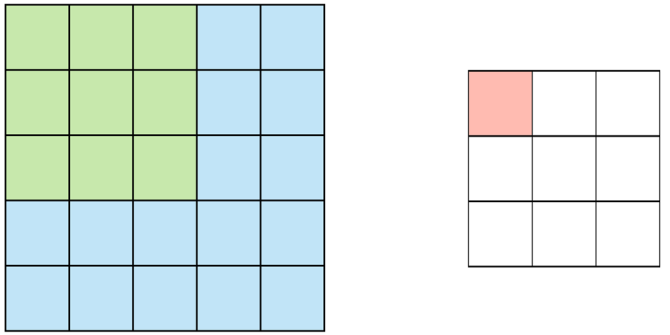
https://animatedai.github.io/

Stride (S): Controlling Output Size

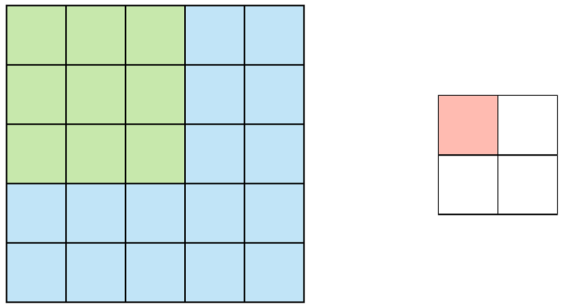
- **The number of pixels to slide the kernel** by (both horizontally and vertically):
  - A stride of 1 will shift the filter every pixel
  - A stride of 2 will shift the filter every 2 pixels

$$H_{out} = \left\lfloor \frac{H - K + 2P}{S} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W - K + 2P}{S} + 1 \right\rfloor$$



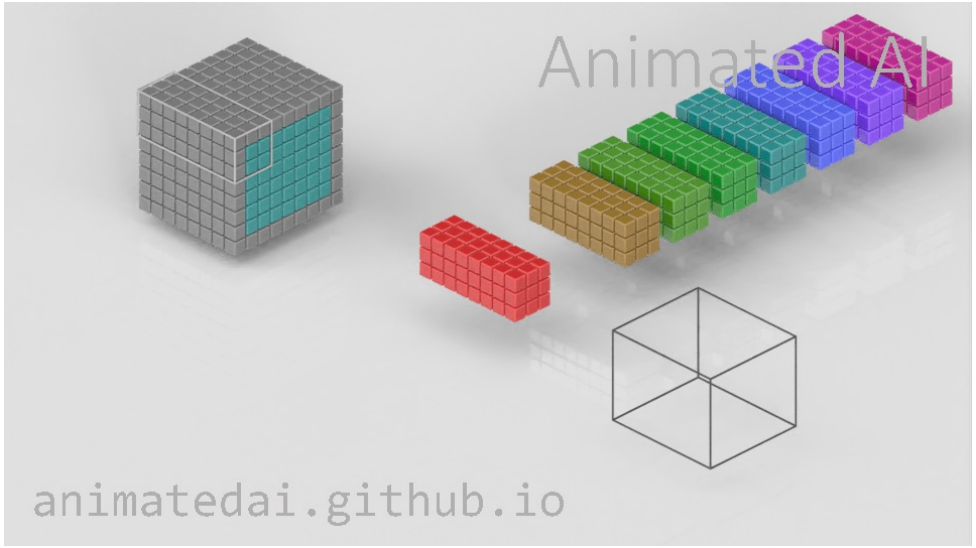
Stride 1 Feature Map



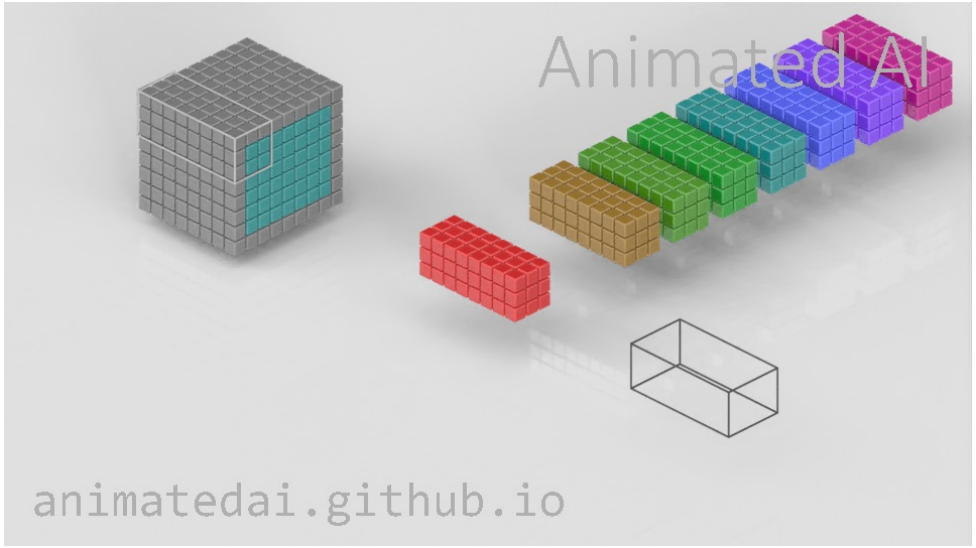
Stride 2 Feature Map

Stride Animations

Same Padding with Stride of 1



Same Padding with Stride of **2**



https://animatedai.github.io/

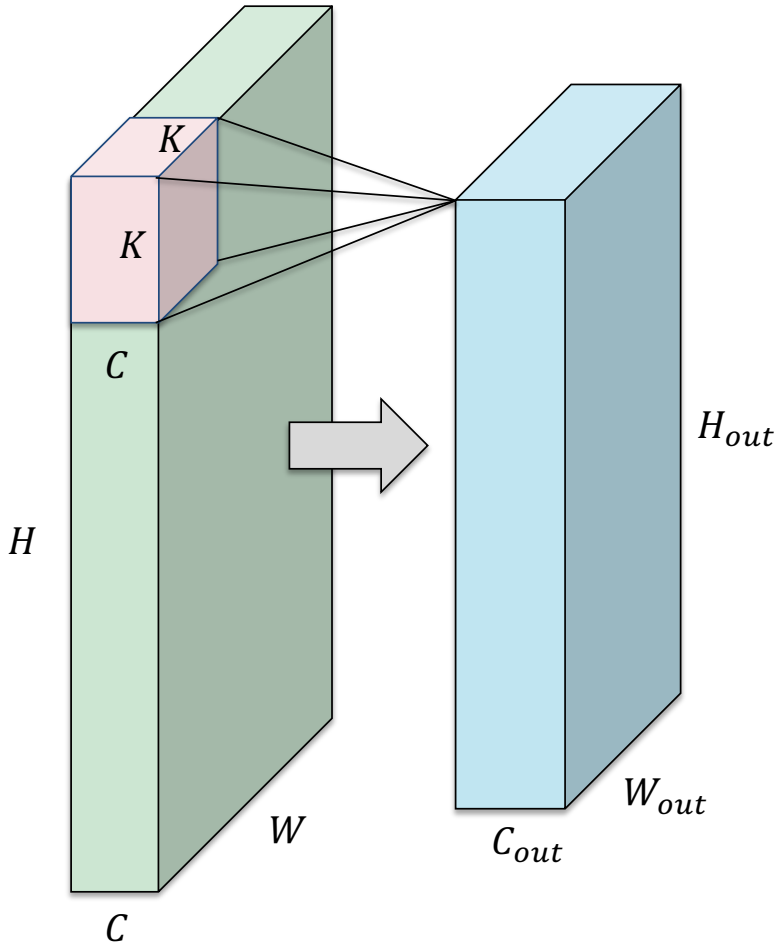
Depth of the Output

- Finally, coming to depth of output
- Each filter gives us one 2D output
- F filters will give us F such 2D outputs
- We can think of resulting outputs as $K \times W_{out} \times H_{out}$ volume
- Thus, $C_{out} = F$

$$H_{out} = \left\lfloor \frac{H - K + 2P}{S} + 1 \right\rfloor$$

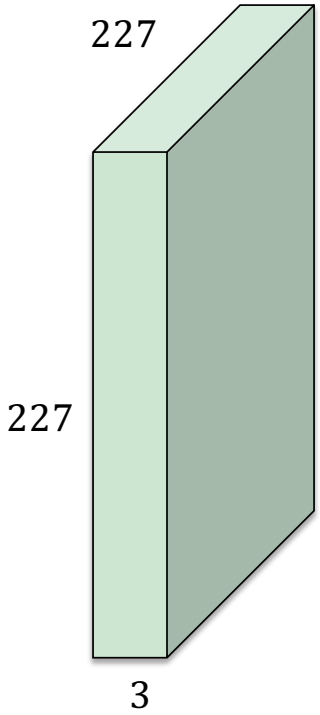
$$W_{out} = \left\lfloor \frac{W - K + 2P}{S} + 1 \right\rfloor$$

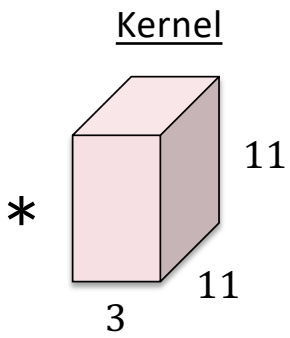
$$C_{out} = F$$



Example

- Work out output dimensions for the following setting:

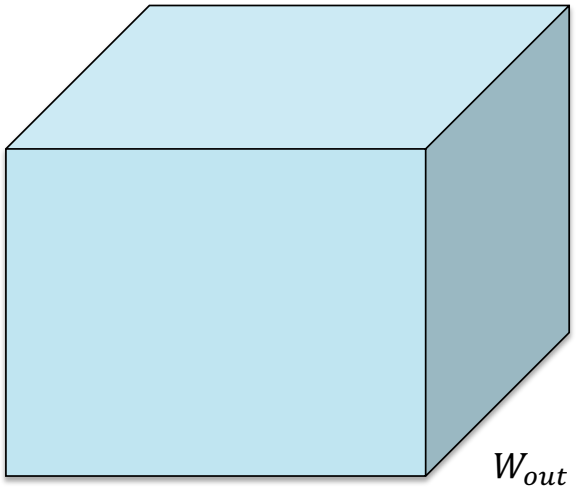




96 filters
Stride = 4
Padding = 0

$$H_{out} = \left\lfloor \frac{H - K + 2P}{S} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W - K + 2P}{S} + 1 \right\rfloor$$



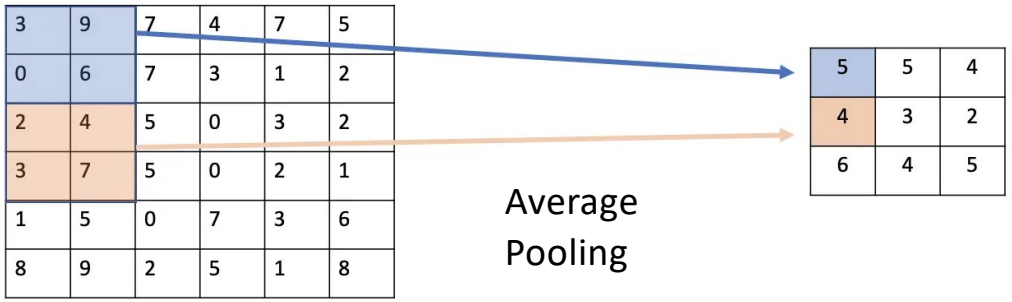
$H_{out} = 55$

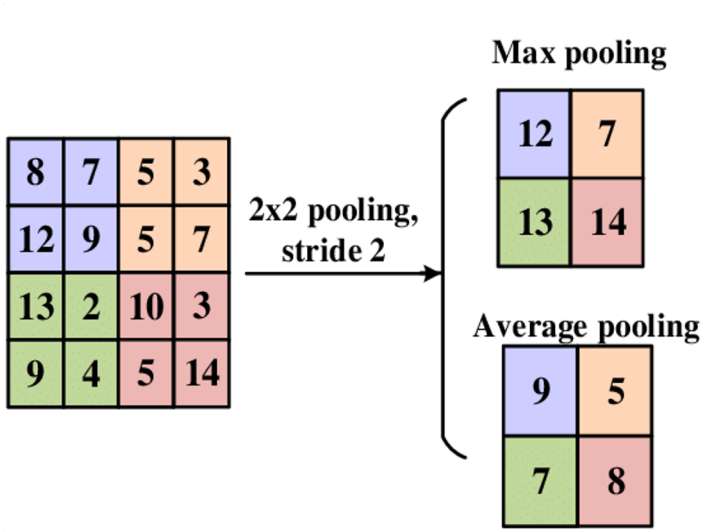
$C_{out} = 96$

$$W_{out} = \left\lfloor \frac{227 - 11}{4} + 1 \right\rfloor = 55$$

Pooling Layer: Downsampling for Efficiency

- Pooling is a **parameter-free** down sampling operation
  - **Max Pooling** and **Average Pooling**



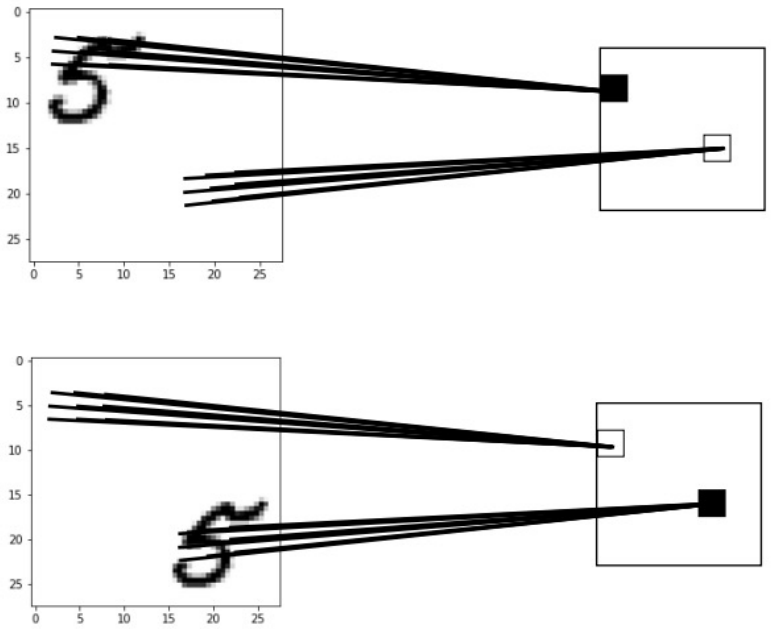


- **Reduces output size**
  - Extract local information
  - Neighboring features may be similar

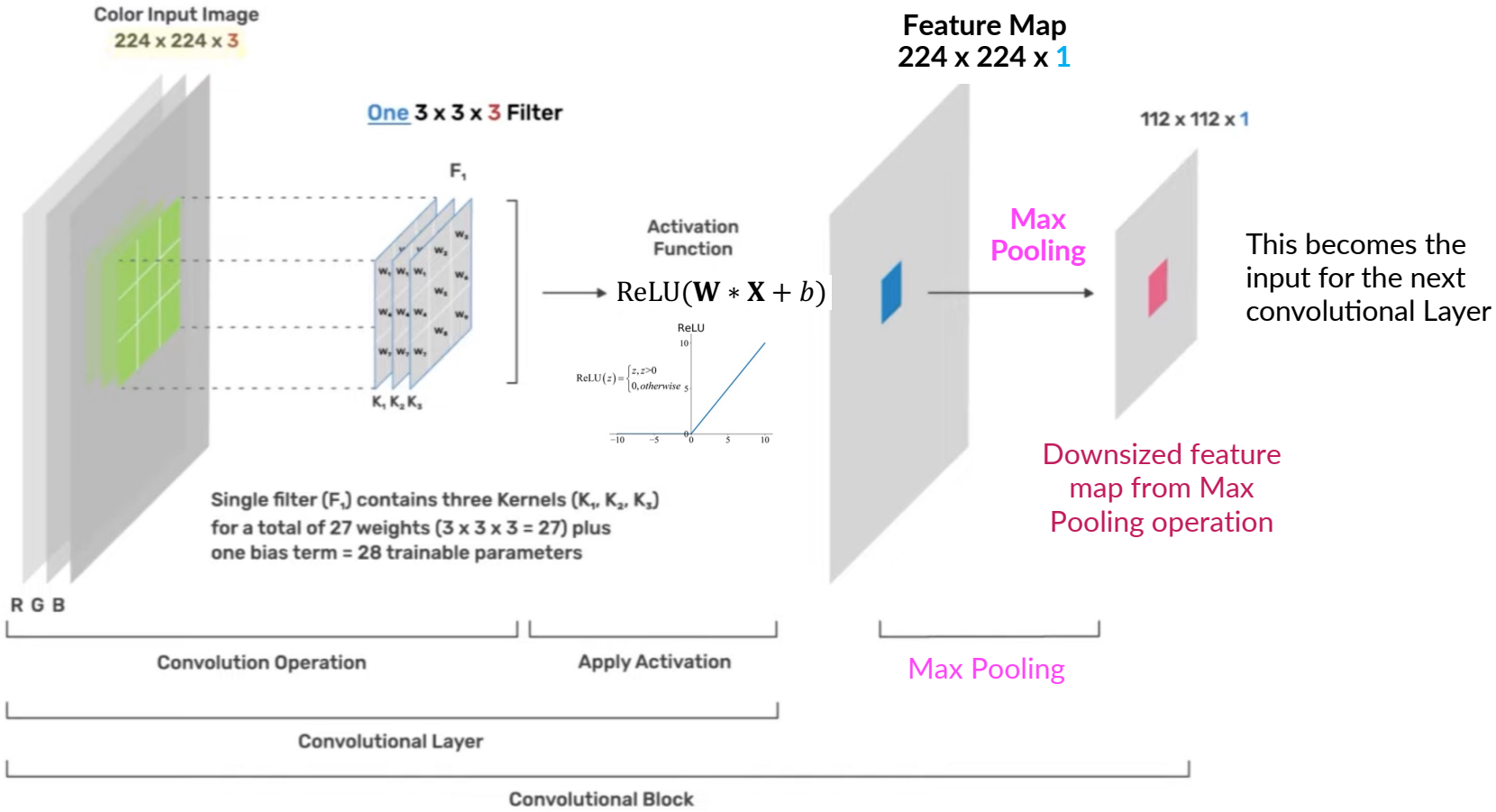
CNN Pooling => Translation Invariant?

- **Note that CNNs are not inherently invariant to scale, rotation, translation, and other transformations.**
- However, pooling layers within CNN architectures can only enhance their resilience to these operations.

The activations are still dependent on the location, etc.

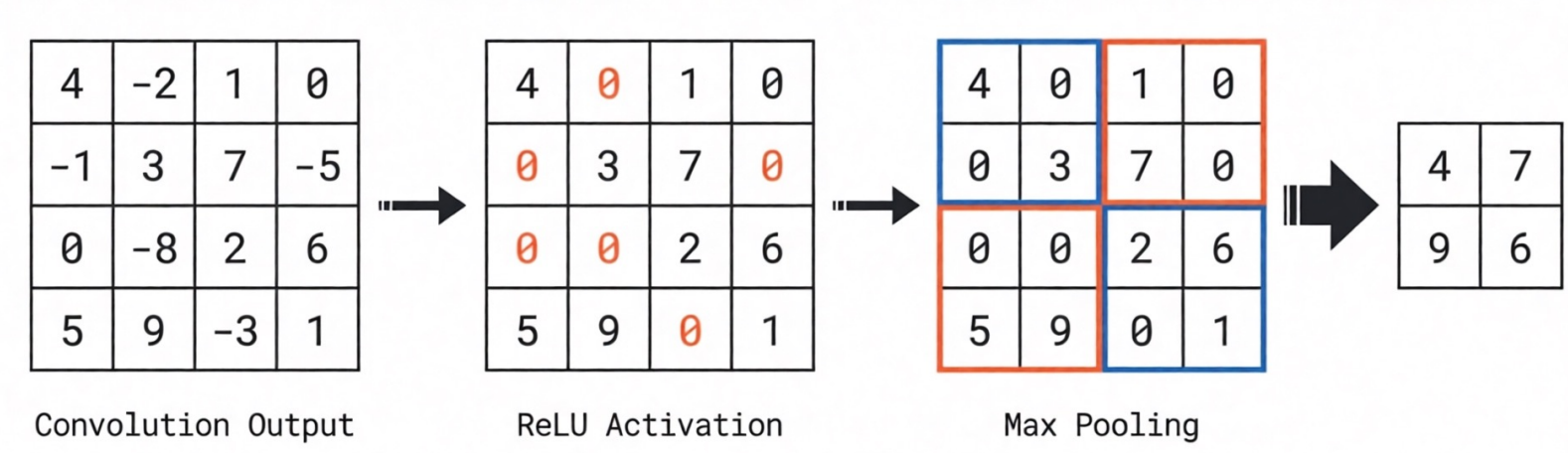


Pooling Layer



Convolution => ReLU => Pooling

- ReLU removes linearity of Convolution
- Pooling reduces complexity and achieve **Spatial Invariance (Translation Robustness)**

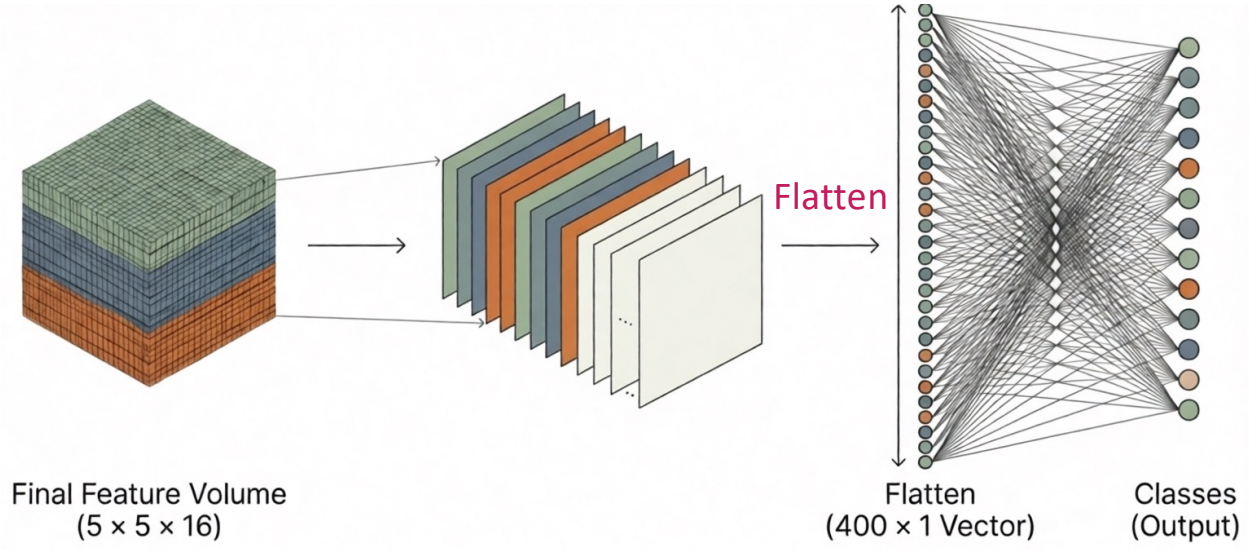


Summary: How Convolutional Layers Work

- **Filter Operation:** A filter (e.g., a 3×3 matrix) slides over the input image, computing the dot product between its weights and the corresponding region of the input. This produces a feature map that emphasizes specific features, such as horizontal edges or corners.
- **Weight Sharing:** Unlike fully connected layers, filters reuse the same weights across all spatial locations, reducing the number of parameters and enabling efficient computation.
- **Multiple Filters:** Each convolutional layer applies multiple filters (e.g., 32 or 64), each producing a unique feature map. These maps collectively capture diverse patterns, from low-level features (e.g., edges) in early layers to high-level features (e.g., objects) in deeper layers.

The MLP Head: From Features to Decisions

- Convolutional layers extract features. **Fully Connected (FC) layers** classify them.
- The 'Flatten' operation bridges these two worlds, unrolling the 3D volume into a 1D vector for the final decision.



Implementation of CNNs

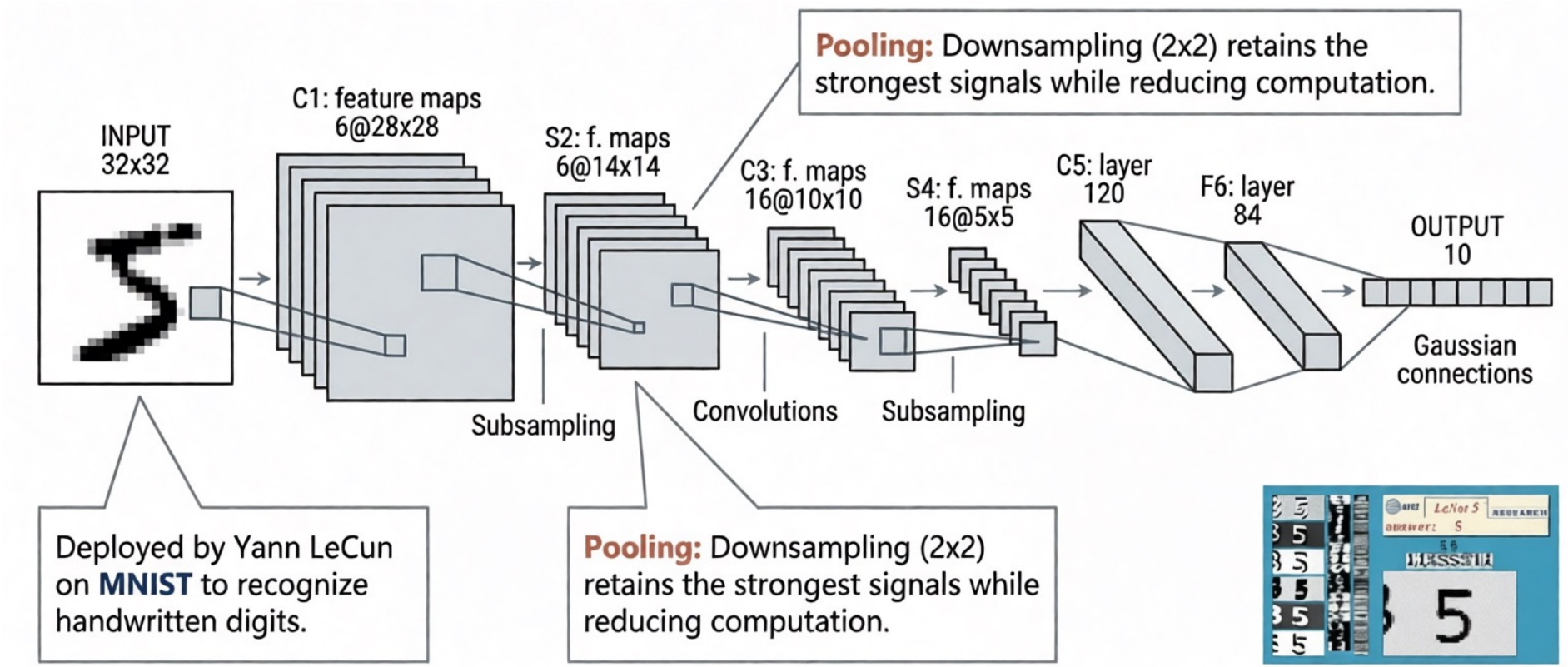
- **Convolutional Layer (CONV)**
  - A way to avoid needing millions of parameters with image data
  - Each layer is “local” and “shared weights” to generate output feature map
  - Each layer produces “feature maps” with (roughly) the same width & height of input but output channel number is equal to number of kernels (filers)
- **Pooling Layer (POOL)**
  - If we ever want to get down to a single output, we must reduce resolution as we go
  - Max pooling: downsample the “feature maps” at each layer, taking the max in each region
  - This makes it robust to small translation changes

9

- **MLP Head: Flatten it up and Full-Connected Layer (FC)**
  - At the end, we get something small enough that we can “flatten” it (turn it into a vector), and feed into a standard fully connected layers for classification or regression application
- **Overall Architecture**: CONV-POOL-CONV-POOL-FC-FC-FC

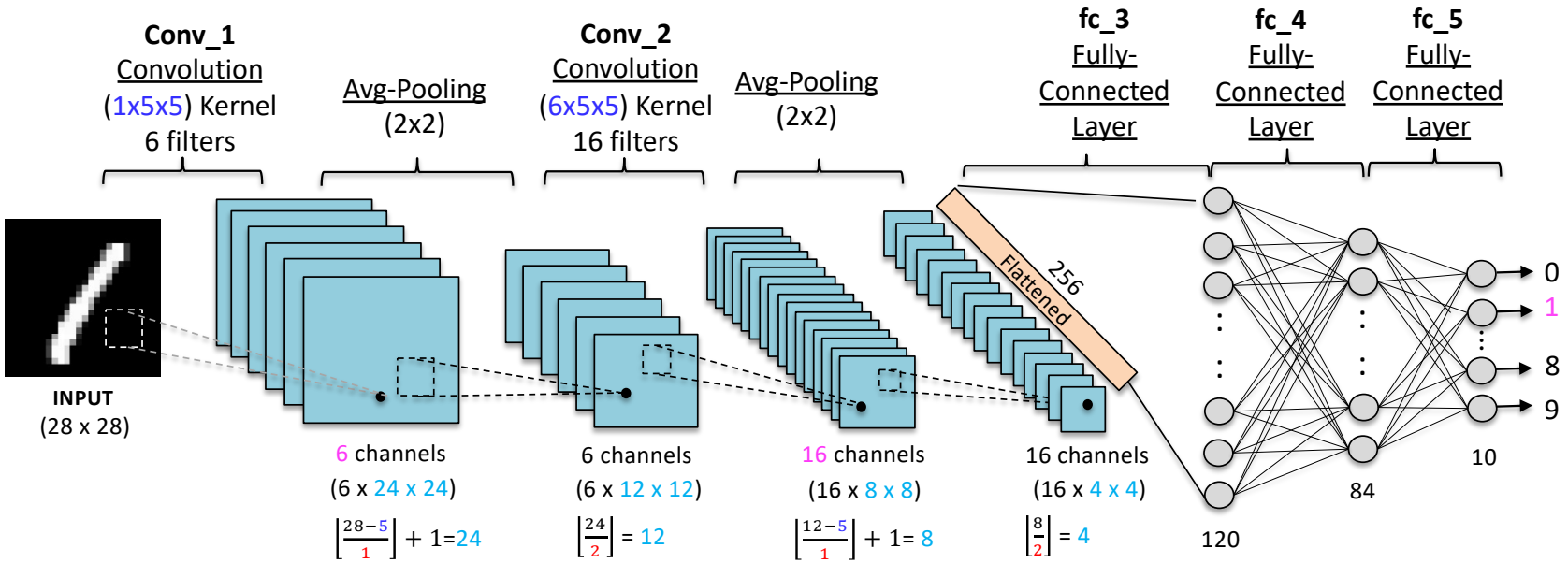
Case Study: The LeNet-5 Architecture

Yann LeCun et al. (1998)



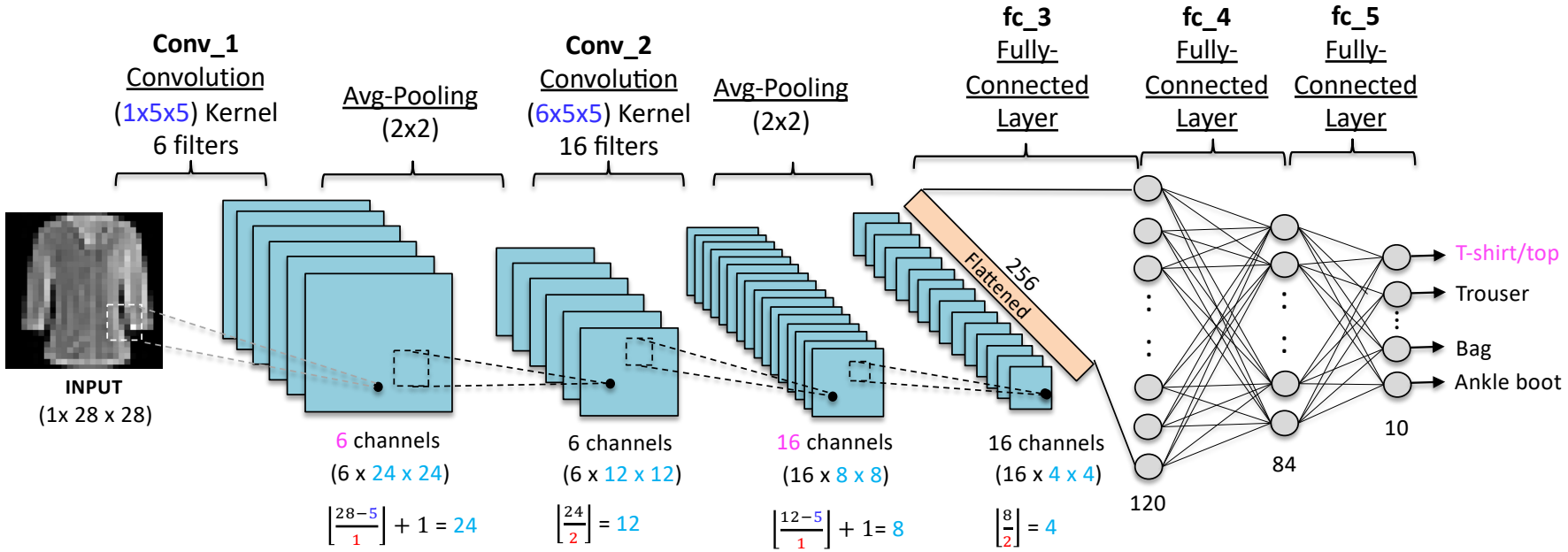
Simplified LeNet-5 for 28×28 MNIST Dataset

Many modern implementations skip padding to 32×32 and apply LeNet-5 directly to 28×28 inputs.



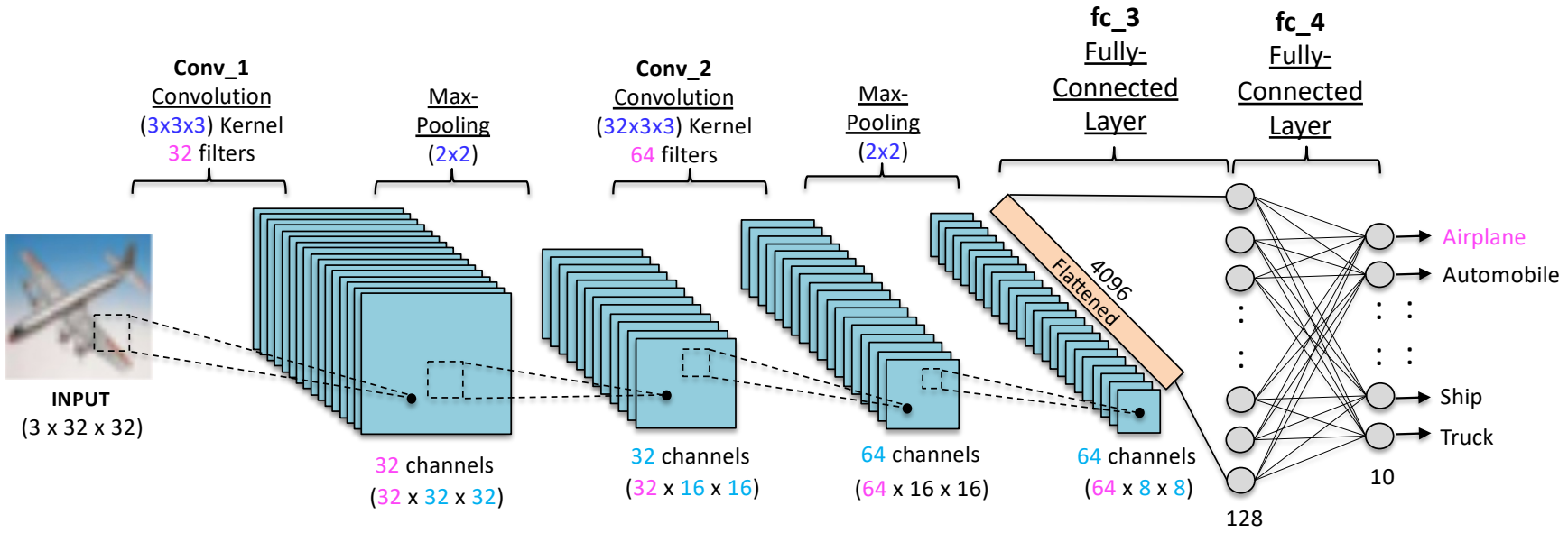
Number of parameters for conv layers = (1x5x5+1)x6 + (6x5x5+1)x16 = 2572
Number of parameters for fully connected layers = (256+1)x120 + (120+1)x84 + (84+1)x10 = 41,854
Total Number of parameters = 2572 + 41,854 = **44,426**
This simplified LeNet-5 for MNIST can achieve more **99.36% accuracy with only 44,426K parameters.**

LeNet-5 for Fashion MNIST Dataset



Similarly, a 5-layer CNN can significantly perform better than MLP model for image classification which can easily achieve accuracy higher than 91% accuracy.

CNN for CIFAR-10 Color Image Classification

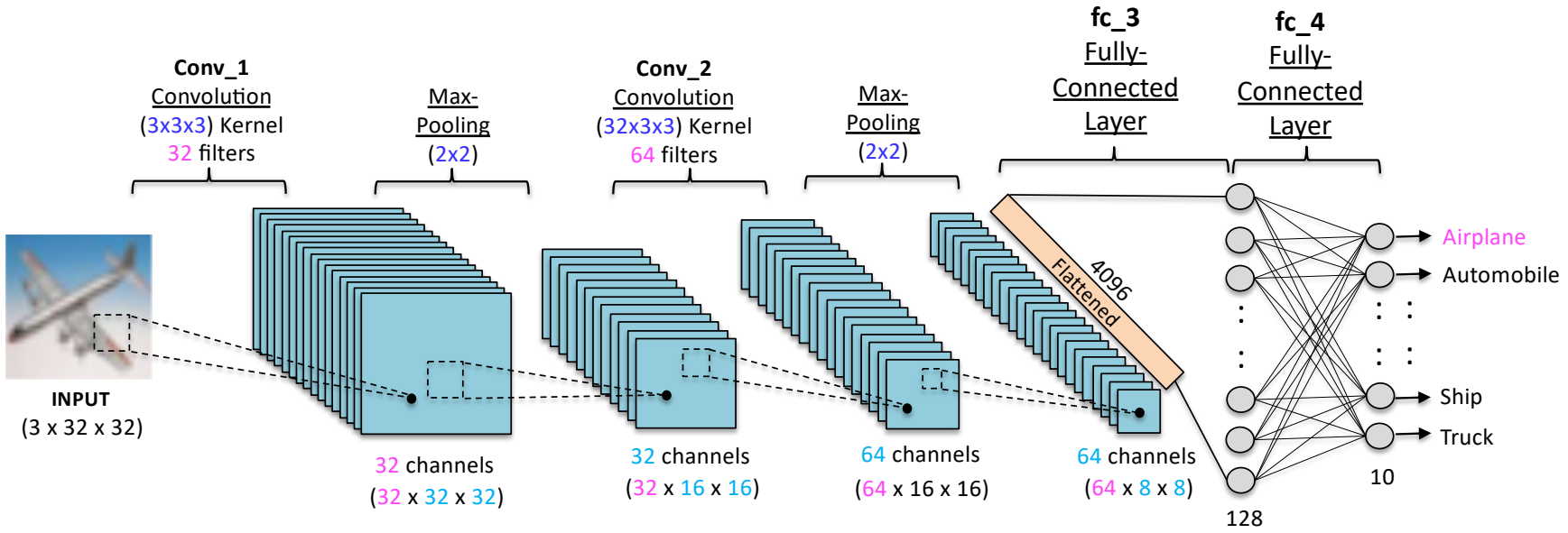


Number of parameters for conv layers = (3x3x3+1)x32 + (32x3x3+1)x64 = 19,392
Number of parameters for fully connected layers = (4096+1)x(128) + (128+1)x(10) = 525,706
Total Number of parameters = 19,392 + 525,706 = **545,098**

A simple Convolutional Neural Network (CNN) can achieve **78% accuracy**.
State-of-the-art is **above 97%.**

https://colab.research.google.com/drive/1W-CpyU3mwWr_ueSm_86C-do6pm361VMe?usp=sharing

CNN for CIFAR-10 Color Image Classification

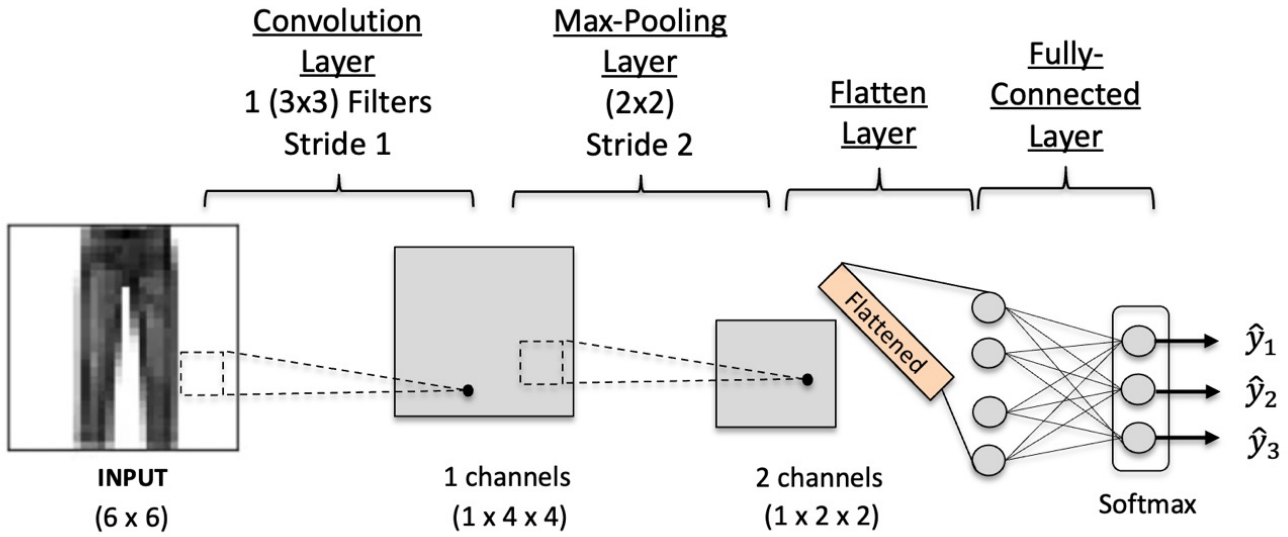


Number of parameters for conv layers = (3x3x3+1)x32 + (32x3x3+1)x64 = 19,392
Number of parameters for fully connected layers = (4096+1)x(128) + (128+1)x(10) = 525,706
Total Number of parameters = 19,392 + 525,706 = **545,098**

https://colab.research.google.com/drive/1W-CpyU3mwWr_ueSm_86C-do6pm361VMe?usp=sharing

CNN Exercise

Consider a Convolutional Neural Network (CNN) that takes a 6×6 grayscale image as input. The network applies a single 3×3 convolutional filter **with bias** to produce one feature map, which is then passed through a ReLU activation function. A max pooling operation is subsequently applied to reduce the spatial dimensions of the feature map. The pooled output is flattened and fed into a fully connected layer (also **with bias**), and a Softmax function is used to produce a single-class probability prediction.



Question (a)

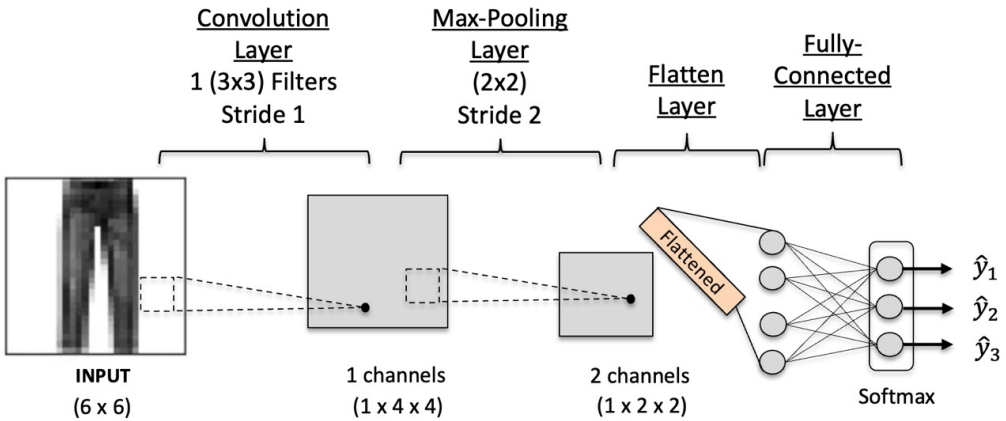
(a) How many learnable parameters (weights and biases) are there in this CNN?

Convolutional layer: $(3\times3+1)\times1=10$

Max-Pool layer: 0

FC layer: $(4+1)\times3=15$

Total: $10+15=\mathbf{25}$



Question (b)

(b) Given the 6×6 input image and the 3×3 convolutional filter kernel **with bias** *b* = **-5** shown below, compute the complete feature map produced by the convolution layer (using valid padding), and then determine the resulting output after applying a 2×2 max-pooling operation with stride 2. Show all intermediate values clearly.

0	5	5	5	5	0
0	5	5	5	5	0
0	5	0	5	5	0
0	5	0	5	5	0
0	5	0	5	5	0
0	5	0	5	5	0

Input image

1	0	-1
1	0	-1
1	0	-1

3x3 filter

Solution (b)

Input image: $\begin{bmatrix} 0 & 5 & 5 & 5 & 5 & 0 \\ 0 & 5 & 5 & 5 & 5 & 0 \\ 0 & 5 & 0 & 5 & 5 & 0 \\ 0 & 5 & 0 & 5 & 5 & 0 \\ 0 & 5 & 0 & 5 & 5 & 0 \\ 0 & 5 & 0 & 5 & 5 & 0 \end{bmatrix}$

Image filter: $\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$

The convolution output without bias :

$$\begin{bmatrix} -10 & 0 & -5 & 15 \\ -5 & 0 & -10 & 15 \\ 0 & 0 & -15 & 15 \\ 0 & 0 & -15 & 15 \end{bmatrix}$$

The convolution output with bias *b*=-5 :

$$\begin{bmatrix} -10-5 & 0-5 & -5-5 & 15-5 \\ -5-5 & 0-5 & -10-5 & 15-5 \\ 0-5 & 0-5 & -15-5 & 15-5 \\ 0-5 & 0-5 & -15-5 & 15-5 \end{bmatrix} = \begin{bmatrix} -15 & -5 & -10 & 10 \\ -10 & -5 & -15 & 10 \\ -5 & -5 & -20 & 10 \\ -5 & -5 & -20 & 10 \end{bmatrix}$$

Output of ReLU operation (Output of convolution layer):

$$\mathrm{ReLU}\left(\begin{bmatrix} -15 & -5 & -10 & 10 \\ -10 & -5 & -15 & 10 \\ -5 & -5 & -20 & 10 \\ -5 & -5 & -20 & 10 \end{bmatrix}\right) = \begin{bmatrix} 0 & 0 & 0 & 10 \\ 0 & 0 & 0 & 10 \\ 0 & 0 & 0 & 10 \\ 0 & 0 & 0 & 10 \end{bmatrix}$$

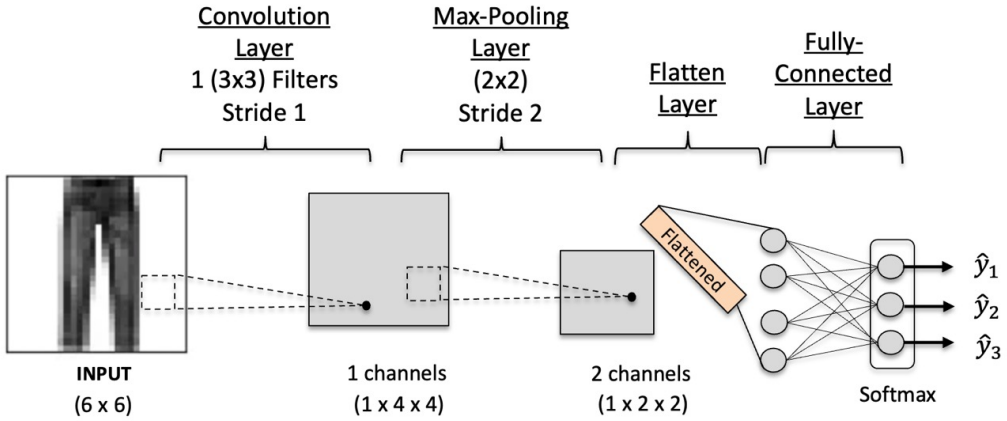
Output of Max-pooling layer

$$\begin{bmatrix} 0 & 10 \\ 0 & 10 \end{bmatrix}$$

Question (c)

(c) Using the feature map obtained from part (b), flatten it and pass it through the fully connected layer with weight matrix $\mathbf{W}$ with bias $\mathbf{b}$. Compute the final output vector $[\hat{y}_1 \quad \hat{y}_2 \quad \hat{y}_3]^T$, apply the Softmax function, and determine the predicted class label for the given 6×6 input image.

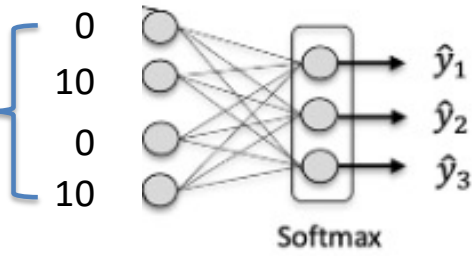
$$\mathbf{W} = \begin{bmatrix} 0.8 & -0.2 & 0.1 & -0.2 \\ 0.2 & -0.6 & 0.5 & -0.3 \\ 0.5 & 0.3 & -0.8 & 0.9 \end{bmatrix} \text{ and } \mathbf{b} = \begin{bmatrix} 0.1 \\ -0.3 \\ 0.6 \end{bmatrix}$$



Solution (c)

After flatten process:

$$\mathbf{x}_{feature} = [0 \quad 10 \quad 0 \quad 10]^T$$



Output of Fully-connected layer 1:

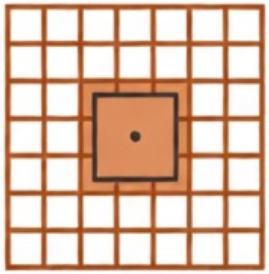
$$\hat{\mathbf{y}} = [\hat{y}_1 \quad \hat{y}_2 \quad \hat{y}_2]^T = \text{Softmax}(\mathbf{W}\mathbf{x}_{feature} + \mathbf{b})$$

$$= \text{Softmax}\left(\begin{bmatrix} 0.8 & -0.2 & 0.1 & -0.2 \\ 0.2 & -0.6 & 0.5 & -0.3 \\ 0.5 & 0.3 & -0.8 & 0.9 \end{bmatrix}\begin{bmatrix} 0 \\ 10 \\ 0 \\ 10 \end{bmatrix} + \begin{bmatrix} 0.1 \\ -0.3 \\ 0.6 \end{bmatrix}\right) = \text{Softmax}\left(\begin{bmatrix} -4.0 \\ -9.0 \\ 12.0 \end{bmatrix} + \begin{bmatrix} 0.1 \\ -0.3 \\ 0.6 \end{bmatrix}\right) = \text{Softmax}\left(\begin{bmatrix} -3.9 \\ -9.3 \\ 12.6 \end{bmatrix}\right) = \begin{bmatrix} \frac{e^{-3.9}}{e^{-3.9} + e^{-9.3} + e^{12.6}} \\ \frac{e^{-9.3}}{e^{-3.9} + e^{-9.3} + e^{12.6}} \\ \frac{e^{12.6}}{e^{-3.9} + e^{-9.3} + e^{12.6}} \end{bmatrix} = \begin{bmatrix} 6.8256\times10^{-8} \\ 3.0828\times10^{-10} \\ 0.9999 \end{bmatrix}$$

The predicted class is the index with the highest probability.

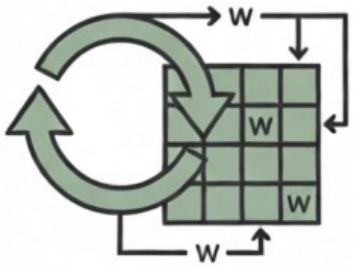
- Class 1: 6.8256×10^{-8}
- Class 2: 3.0828×10^{-10}
- Class 3: 0.9999 ← highest
- Predicted class label = 3

Why CNN Architecture Work



Local Connectivity

Filters focus on small, local regions, mimicking the biological eye's receptive fields.



Parameter Sharing

Features are useful everywhere. Reusing weights makes the model efficient and translation invariant.

Parameter Sharing

Simple patterns (edges) combine to form complex objects (faces) as the network deepens.

Complex understanding arises from the layering of simple operations.

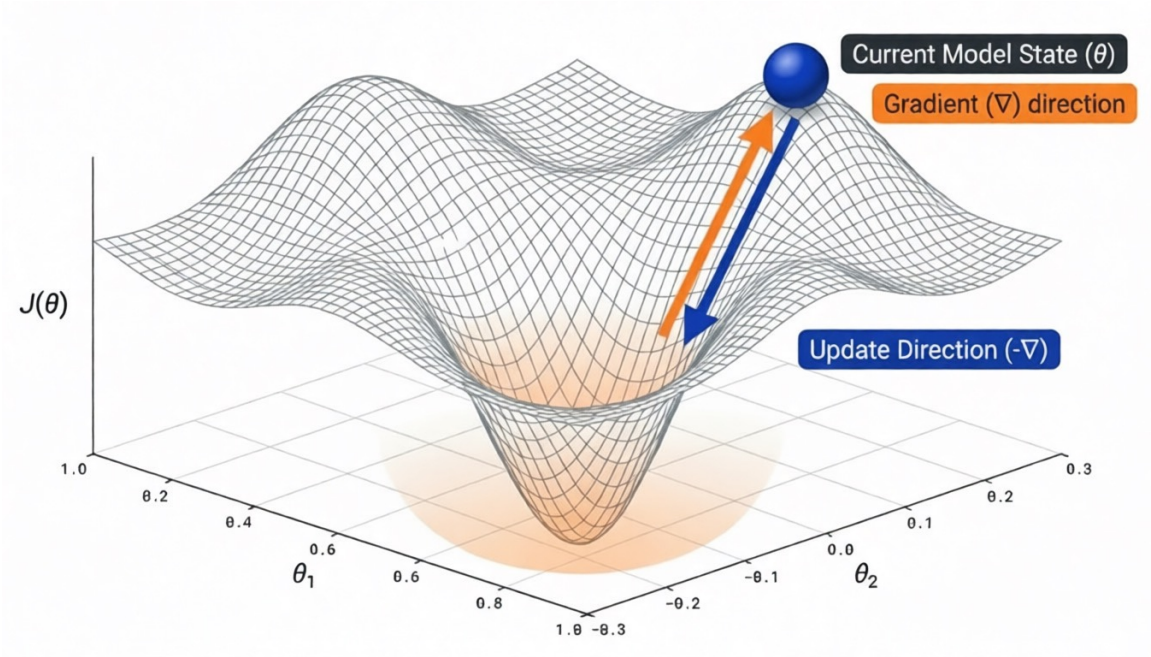
How to Train CNNs?

Backpropagation in CNNs

- Backpropagation trains CNNs by adjusting weights to minimize a cost function.
- During the forward pass, inputs are processed through convolutional, activation, pooling, and fully connected layers.
- In the backward pass, gradients of the loss with respect to the weights are computed using the chain rule. Key steps include:
  1. **Convolutional Layers** : Gradients for filter weights are calculated via convolutions of the error signal with input feature maps.
  2. **Pooling Layers** : Gradients are passed based on max-pooling or average-pooling rules.
  3. **Weight Updates** : Optimizers like SGD or Adam update weights using computed gradients.
- This process efficiently trains CNNs to learn spatial hierarchies, making them ideal for tasks like image recognition.

Quantifying Error: The Loss Function

Loss Landscape



The Objective

- Find weights e that minimize the cost function $\mathcal{L}(\theta)$.

The Trigger

- Training begins by measuring how 'wrong' the prediction is.

Common Functions

- Binary Cross-Entropy: For 2-class tasks (e.g., Healthy vs. Sick).
- Softmax Loss: For multi-class tasks (e.g., CIFAR-10)

Propagating Error: The Chain Rule

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial y} \times \frac{\partial y}{\partial z} \times \frac{\partial z}{\partial w}$$

How much the Loss changes as Output changes.

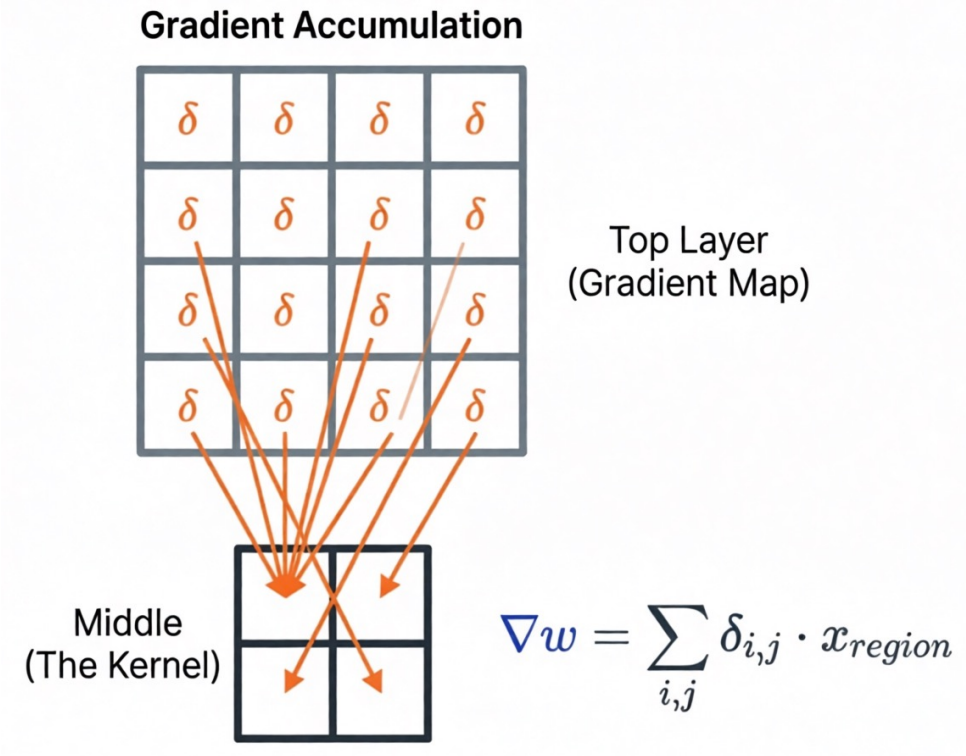
Derivative of Activation (e.g., ReLU).

Input from previous layer.

This recursive mechanism allows the network to assign "blame" to specific weights for the total error, cascading from the final layer back to the first.

Gradients in Convolutional Layers

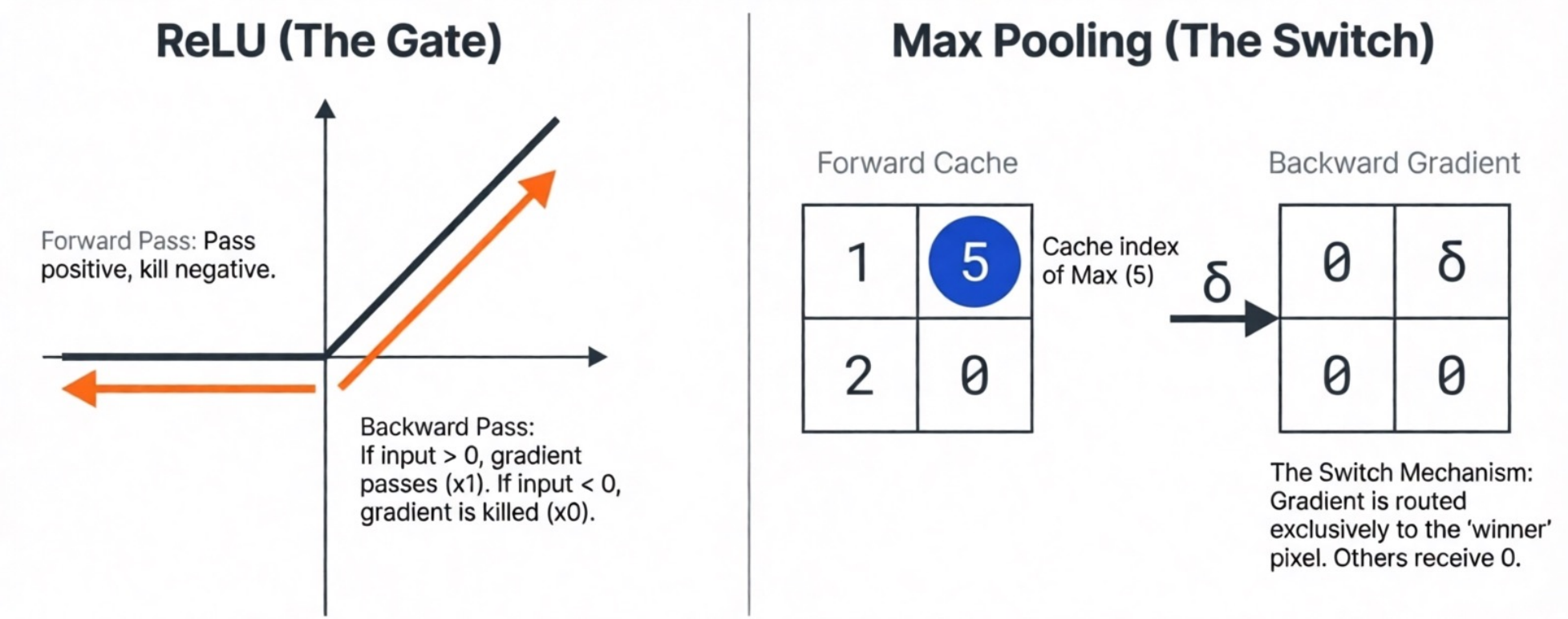
Handling Shared Parameters (Weights)



- **The Challenge**: In a CNN, a single kernel weight is used at every spatial position (i, j) of the image.
- **The Solution**: We cannot update the weight based on just one pixel's error. We must SUM the gradients from every position where that weight was applied.
- **Implication**: This enforces Translation Invariance-the filter learns features that are useful anywhere in the image.

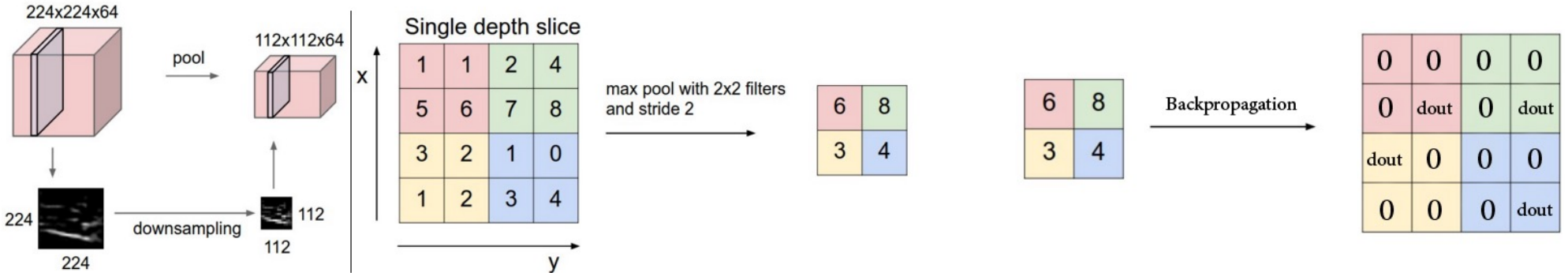
Routing Gradients: Non-Linearities

Pooling and ReLU have no weights, but they guide the flow.



Backpropagation Pooling Layers

- **There are no weights to learn**, only have to propagate gradients through
- In **Max-Pooling**, backpropagated gradient is assigned only to the winning pixel i.e., the one which had maximum value in the pooling block; this can be kept track of in the forward pass
- In **Average Pooling**, the backpropagated gradient is divided by the area of the pooling block ($K_1 \times K_2$) and equally assigned to all pixels in the block.



https://leonardoaraujosantos.gitbook.io/artificial-inteligence/machine_learning/deep_learning/pooling_layer

The Weight Update

Gradient Descent

The Weight
(Kernel Parameter)

Learning Rate (Step Size)
Controls speed of convergence.

Subtracting the gradient
(Moving downhill)

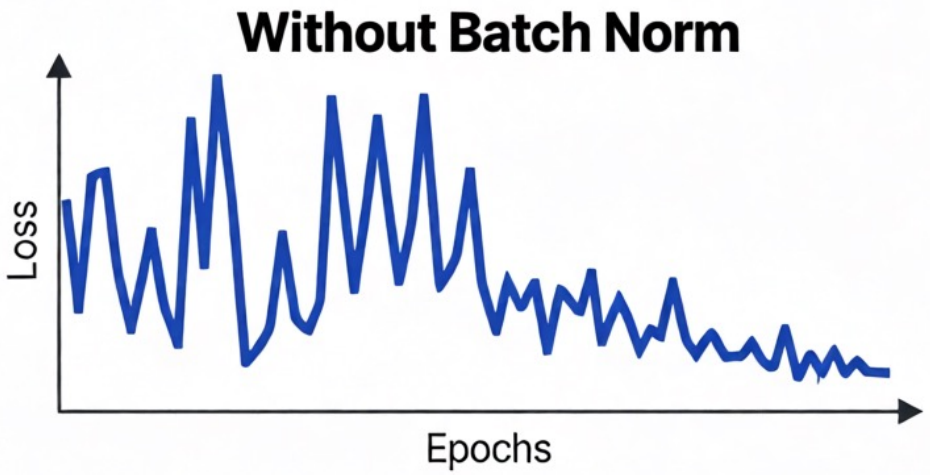
$$\theta_{new} = \theta_{old} - \eta \cdot \nabla \mathcal{L}(\theta)$$

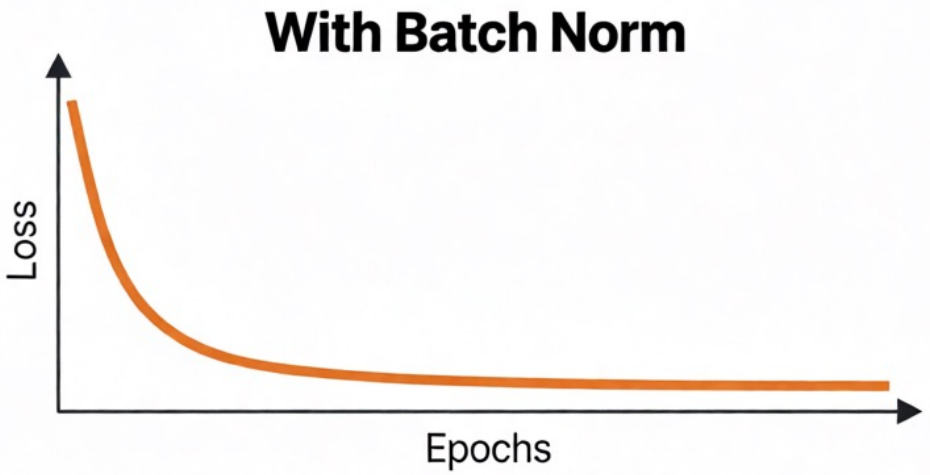
The Calculated Gradient
(Direction of steepest ascent)

- Optimizers like **Adam** or **AdamW** dynamically adjust the learning rate (η) during training to speed up convergence and avoid getting stuck in local minima.

Stabilizing the Gradient

Batch Normalization & Dropout





1. Batch Normalization

- **Problem:** Internal Covariate Shift. Input distributions change as layers update.
- **Solution:** Normalize layer inputs to Mean=0, Variance=1.
- **Result:** Allows higher learning rates, prevents vanishing gradients.

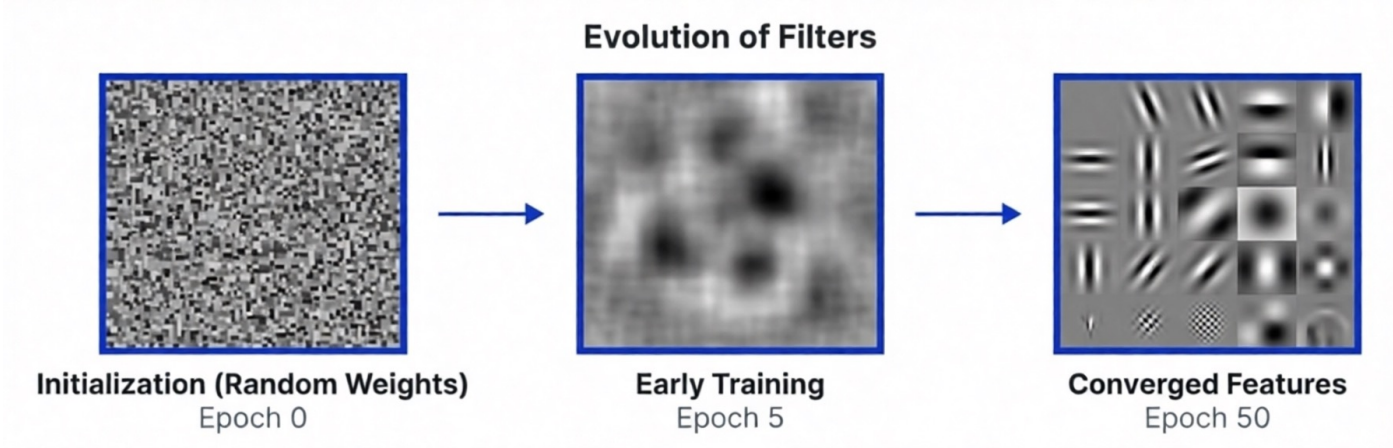
2. Dropout

- **Problem:** Overfitting (memorization).
- **Solution:** Randomly "kill" 50% of neurons during training.
- **Result:** Forces network to learn robust, redundant features.

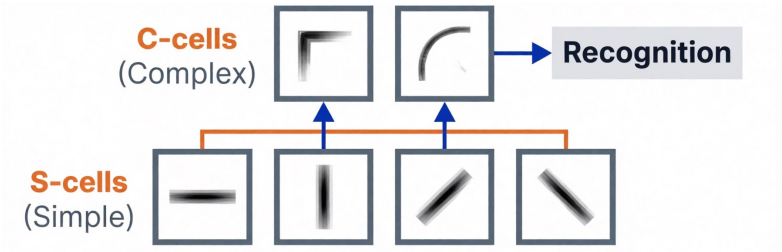
Tricks for improving CNN Trainings

- **Better weight initialization**:
  - Glorot/He initialization: Empirically shown to give good results
  - **Hand-designed weight initialization:** Using domain knowledge, come up with features like edges (with certain orientations), shapes etc.
- **Regularization methods**:
  - L2-weight decay, L1-weight decay
  - BatchNorm, Input/Gradient Noise
  - Dropout (Not commonly used today)
  - Data augmentation

From Randomness to Recognition



- Through millions of iterations of the Forward → Backward → Update loop, the network automatically organizes itself. Random noise evolves into structured feature detectors capable of perceiving edges, shapes, and objects.



Interesting Property of CNNs

- CNN layers learn features in a **hierarchical manner.**
- **Initial layers** learn simple and generic features like edges and color blobs, which are consistent across different models trained on various datasets.
- **Later layers** capture more abstract and specialized features that are specific to the dataset being trained.
- **Exploit the hierarchical nature of CNN features for tasks such as transfer learning**, where lower layers can be reused for different tasks or datasets.