SCALER School of Technology × Priyansh Saxena

Convolutions: The Core Operation

Local Connectivity • Filters • Padding & Stride • Parameter Sharing • Receptive Fields • Pooling

Priyansh Saxena

Scaler School of Technology | Deep Learning I: Neural Networks | Week 4 — Session 1/2

26 May 2026

Recap & Today's Questions

Week 3: Making Training Work

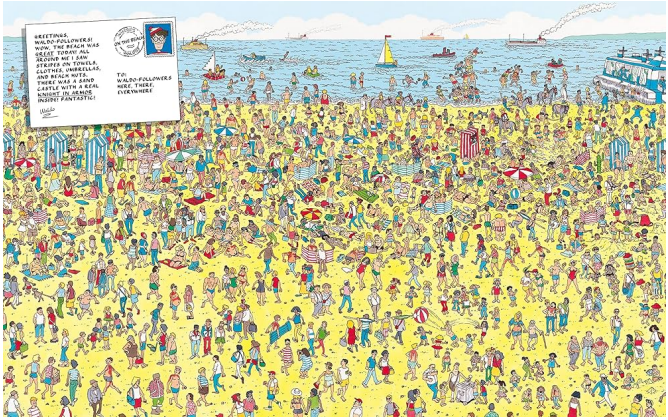
- **Initialization:** Xavier (tanh) / He (ReLU) prevent vanishing & exploding gradients
- **BatchNorm / LayerNorm:** stabilize activations, allow higher learning rates
- **Debugging:** overfit one batch first, verify loss at init, check shapes

Week 4, Session 1: A New Architecture

- **Why** do fully-connected layers fail on images?
- **What** is a convolution and how does it work?
- **How** do filters, padding, stride, and pooling build up a CNN?

We know how to train networks. Now we design one built for images.

How Do You Find Waldo?



How do you find him?

You don't examine every pixel simultaneously.

Your eye **scans local patches** — looking for the red-and-white stripes.

Once found in one patch, you recognise him **anywhere** in the image.

CNNs do exactly this:
scan a small filter across the image, detecting the same pattern wherever it appears.

The Problem with Fully-Connected Layers

A typical image: $224 \times 224 \times 3 = \mathbf{150{,}528}$ input values

FC layer to 1000 neurons:

$$150{,}528 \times 1000 = \mathbf{150.5 \text{ million weights}}$$

- That's ONE layer
- A network has dozens of layers
- Massively wasteful, overfits instantly

The Core Issue

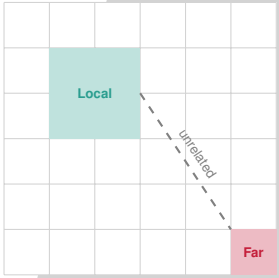
FC layers treat every pixel as equally important to every neuron.

They completely **ignore spatial structure**.

Images Have Spatial Structure

Key Observation

- Nearby pixels are correlated (sky is blue in patches, not individual pixels)
- Distant pixels are usually unrelated
- A pixel in the top-left says nothing about the bottom-right



Nearby = related. Far = unrelated.

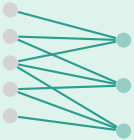
Solution: connect each neuron to only a **small local patch** of the input.

Local Connectivity

Fully-Connected

15 connections

Convolutional (local)



9 connections

Same job, drastically fewer parameters. Each neuron sees only a small region.

Quiz 1

1. A 64×64 grayscale image is connected to an FC layer with 100 neurons. How many weights does this layer have?

2. Why is this problematic for training?

Quiz 1

1. A 64×64 grayscale image is connected to an FC layer with 100 neurons. How many weights does this layer have?

2. Why is this problematic for training?

Answers

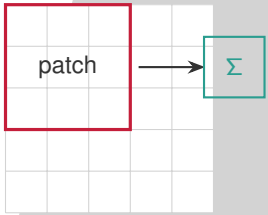
1. $64 \times 64 \times 100 = \mathbf{409{,}600}$ weights
2. Too many parameters $\rightarrow$ overfits easily, slow to train, memory-hungry

What Is a Filter (Kernel)?

- A small grid of **learnable weights** (e.g., 3×3)
- Slides across the image, position by position
- At each position: **element-wise multiply** $\rightarrow$ **sum**
- Produces one output value per position

Analogy

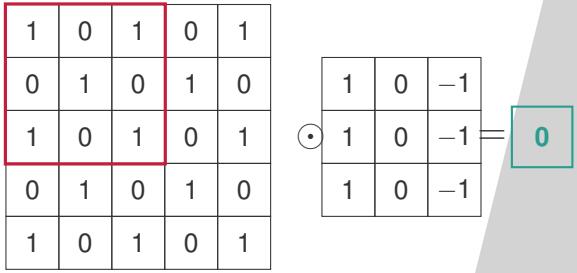
A magnifying glass scanning a document — same lens, different locations.



Slide, multiply, sum, repeat.

Convolution Step-by-Step

Input (5×5) $*$ **Filter** (3×3) $=$ **Output** (3×3)



Position $(0, 0)$: $(1 \cdot 1) + (0 \cdot 0) + (1 \cdot -1) + (0 \cdot 1) + (1 \cdot 0) + (0 \cdot -1) + (1 \cdot 1) + (0 \cdot 0) + (1 \cdot -1) = \mathbf{0}$

Then the filter slides one position to the right and repeats.

What Filters Detect

Vertical Edge Filter

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

Responds strongly to vertical edges

Horizontal Edge Filter

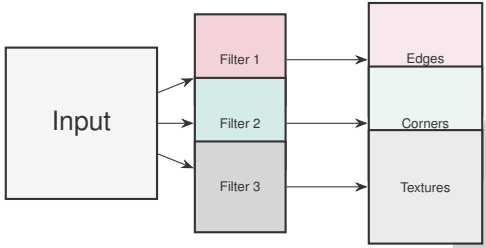
$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Responds strongly to horizontal edges

Key Insight

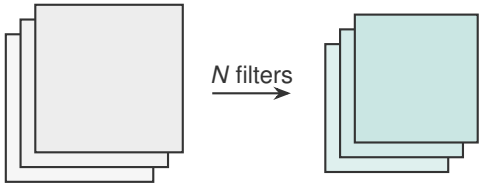
The filter **responds** (outputs a large value) when it finds the pattern it's looking for.
On uniform regions $\rightarrow$ output ≈ 0. On matching edges $\rightarrow$ output is large.

Multiple Filters = Multiple Feature Maps



- 1 filter $\rightarrow$ 1 feature map (detects one pattern)
- *N* filters $\rightarrow$ *N* feature maps (each detects something different)
- Typical: 32, 64, 128, 256 filters per layer

The Full Dimensional Picture



$H \times W \times C$
(e.g., $224 \times 224 \times 3$)

$H' \times W' \times N$
(e.g., $224 \times 224 \times 64$)

Component	Dimensions
Input	$H \times W \times C$
One filter	$F \times F \times C$ (spans all input channels)
N filters	Output: $H' \times W' \times N$

Filter depth always equals input channels.

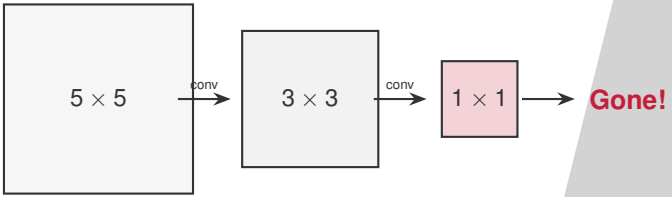
Coming Up Next

Part 2: Padding, Stride, Parameter Sharing, Receptive Fields & Pooling

Take 5 minutes. Stretch, grab water.

Think about this: if the output keeps shrinking with every layer, what happens after 10 layers?

The Shrinking Problem



Problem

Every conv layer **shrinks** the spatial dimensions.
After a few layers, your feature maps disappear entirely.

Solution: Padding.

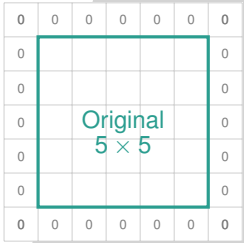
Padding

Valid Padding (no pad)

- No zeros added
- Output shrinks: $5 \times 5 \xrightarrow{3\times3} 3 \times 3$

Same Padding (pad = 1)

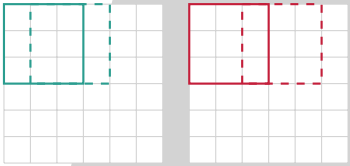
- Ring of zeros around border
- Output = input size: $5 \times 5 \xrightarrow{3\times3} 5 \times 5$
- **Most common** in practice



Padded to 7×7

Stride

- **Stride** = how far the filter jumps between positions
- Stride 1: examine every position (default)
- Stride 2: skip every other position $\rightarrow$ output halves



Use Case

Stride > 1 is used for **downsampling** — an alternative to pooling in modern architectures.

The Output Size Formula

$$\text{Output} = \frac{W - F + 2P}{S} + 1$$

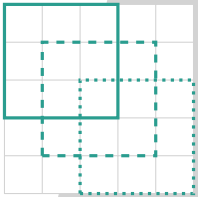
W	F	P	S	**Output**	
7	3	0	1	$(7-3+0)/1+1 = \mathbf{5}$	
7	3	1	1	$(7-3+2)/1+1 = \mathbf{7}$	(same!)
7	3	1	2	$(7-3+2)/2+1 = \mathbf{4}$	
32	5	2	1	$(32-5+4)/1+1 = \mathbf{32}$	(same!)

Rule of thumb: For “same” padding with stride 1, set $P = \lfloor F/2 \rfloor$.

Parameter Sharing — The Key Insight

The Idea

The **same filter weights** are used at every spatial location.



Why?

- A vertical edge looks the same whether it's in the top-left or bottom-right
- No need to learn separate detectors for each position

Result:

- One 3×3 filter = **9 parameters**
- Reused at all $224 \times 224 = 50{,}176$ positions
- FC equivalent: $\sim$150 million parameters

Quiz 2

1. Input: 7×7, Filter: 3×3, Padding: 0, Stride: 1.
   What is the output size?

2. Same as above, but with Padding = 1.
   What is the output size now?

Quiz 2

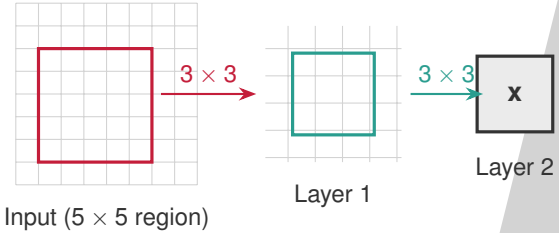
1. Input: 7×7, Filter: 3×3, Padding: 0, Stride: 1.
   What is the output size?

2. Same as above, but with Padding = 1.
   What is the output size now?

Answers

1. $(7 - 3 + 0)/1 + 1 = \mathbf{5} \times \mathbf{5}$
2. $(7 - 3 + 2)/1 + 1 = \mathbf{7} \times \mathbf{7}$ (same padding preserves size!)

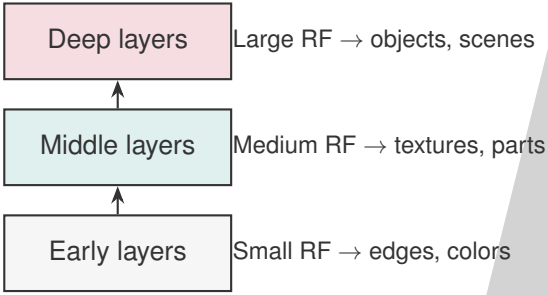
Receptive Field — What Does a Neuron “See”?



Layer	Receptive Field on Input
Layer 1 (one 3×3 conv)	3×3
Layer 2 (two 3×3 convs)	5×5
Layer 3 (three 3×3 convs)	7×7

Deeper layers see more of the input image.

Why Receptive Field Matters



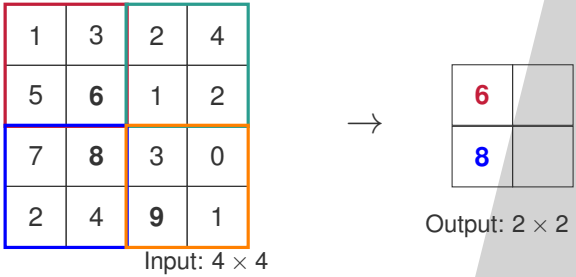
Key Takeaway

Stacking layers builds a **hierarchy of features**:
simple patterns → complex structures → full objects.

This is WHY we stack conv layers — each layer adds context.

Pooling — Downsample and Summarize

Max Pooling (2×2, stride 2)



Take the maximum value in each 2×2 window.

"Is there a strong feature somewhere in this region? I don't care exactly where."

Why Pool?

1. **Reduces spatial size**
   Less computation downstream
   $224 \times 224 \xrightarrow{\text{pool}} 112 \times 112 \xrightarrow{\text{pool}} 56 \times 56$
2. **Translation invariance**
   Small shifts in input $\rightarrow$ same pooled output
3. **Increases receptive field**
   Each subsequent layer covers more of the input

Tradeoff

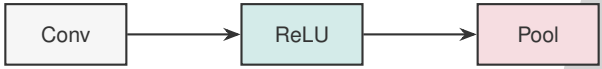
Pooling discards spatial precision.

You know a feature exists, but you lose *exactly where*.

Types

- Max pooling (classic)
- Average pooling
- Global Avg Pool (end of network)

The Standard Conv Block



Shape trace (MNIST, 28×28 input):

After	Shape
Input	$28 \times 28 \times 1$
Conv1 (16 filters, 3×3, pad 1)	$28 \times 28 \times 16$
MaxPool (2×2)	$14 \times 14 \times 16$
Conv2 (32 filters, 3×3, pad 1)	$14 \times 14 \times 32$
MaxPool (2×2)	$7 \times 7 \times 32$
Flatten	1568
FC $\rightarrow$ 10 classes	10

Total parameters: $\sim$20K. FC-only equivalent: $\sim$100K.

Recap — 5 Key Takeaways

1. **Local connectivity** exploits spatial structure — don't connect everything
2. **Filters slide** across the image, detecting patterns via dot products
3. **Padding** preserves spatial size; **stride** controls downsampling
4. **Parameter sharing** = same filter everywhere = massive efficiency
5. **Deeper layers see bigger regions** $\rightarrow$ hierarchy of features

Final Quiz

1. How many parameters in a 3×3 conv filter (with 1 bias)?
2. True or False: A conv layer uses different weights at each spatial position.
3. Two stacked 3×3 conv layers — what is the effective receptive field on the input?

Final Quiz

1. How many parameters in a 3×3 conv filter (with 1 bias)?
2. True or False: A conv layer uses different weights at each spatial position.
3. Two stacked 3×3 conv layers — what is the effective receptive field on the input?

Answers

1. $3 \times 3 + 1 = \mathbf{10}$
2. **False** — parameter sharing means same weights everywhere
3. 5×5 — each layer adds 2 pixels of context

Next Session

Week 4, Session 2

CNN Architectures: From LeNet to ResNet

- LeNet-5 (1998) — the pioneer
- AlexNet (2012) — the ImageNet breakthrough
- VGGNet — depth matters
- GoogLeNet/Inception — efficiency through design
- ResNet — skip connections solve the depth problem

You know the building block. Next: how people assembled them into systems that beat humans at image recognition.

Thank You!

Questions?

Priyansh Saxena

All lecture materials available at:

www.priyanshsaxena.com/sst-deep-learning-nn