Yale University

Convolutional Neural Networks

S&DS 265 · Introductory Machine Learning · Lecture 13

Zhuoran Yang

Fall 2026

AI usage: figures were generated, and these slides polished, with the help of AI tools.

Outline

Learning Objectives

In this lecture you will learn:

1. Why images call for locality, weight sharing, and translation equivariance;
2. How convolution, padding, stride, channels, and pooling determine every tensor shape;
3. How gradients flow backward through shared filters and pooling windows;
4. How initialization, residual paths, and batch normalization stabilize training;
5. How to implement and diagnose a small convolutional network in PyTorch.

Outline

Motivating Example: Recognizing a Handwritten Digit

A digit can move a few pixels without changing its identity. How should a model use this spatial regularity instead of relearning the same pattern at every location?

Example data: MNIST, LeCun et al.; experiment data: scikit-learn handwritten digits.

The Handwritten Digits Data

object	value
example	grayscale image
feature	pixel intensity
target	digit 0, … , 9

MNIST has 60,000 training and 10,000 test images at 28×28. The companion experiment uses 1,797 scikit-learn images at 8×8.

Source: MNIST: LeCun, Cortes, and Burges. Experiment: scikit-learn digits data.

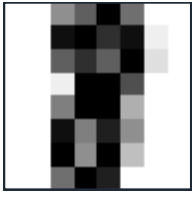
From LeNet-5 to ResNet

From AlexNet to ResNet, ImageNet top-5 test error fell from 15.3% to 3.57%, while the architecture grew from 8 to 152 learned layers. The ResNet number is from an ensemble.

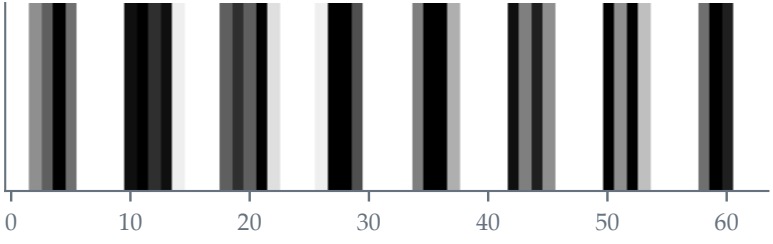
Source: LeCun et al. (1998); Krizhevsky, Sutskever, and Hinton (2012); He et al. (2016).

Reading One Image in Two Representations

Image: 8×8



Flattened input: 64 coordinates



One held-out scikit-learn digit. The vector retains every intensity but discards the explicit neighborhood structure.

Problem Setup

For $X_i \in [0,1]^{C \times H \times W}$ and $y_i \in \{0, \dots, 9\}$, let a model $f_\theta(X) \in \mathbb{R}^{10}$ produce the logits. Fit it by multiclass cross-entropy:

$$\widehat{R}_n(\theta) = \frac{1}{n} \sum_{i=1}^{n} -\log \frac{\exp([f_\theta(X_i)]_{y_i})}{\sum_{k=0}^{9} \exp([f_\theta(X_i)]_k)}.$$

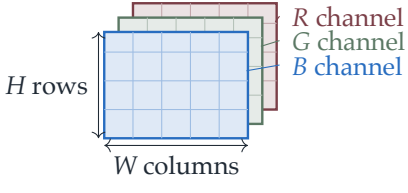
The remaining modeling choice is the **function class** $\mathcal{F} = \{f_\theta : \theta \in \Theta\}$. Ideally, it should leverage the spatial structure of images.

An Image Is a Three-Dimensional Tensor

A color image stores one intensity for every channel c, row i, and column j:

$$X = (X_{c,i,j}) \in [0,1]^{C \times H \times W}.$$

- C: channels; $C = 1$ for grayscale and $C = 3$ for RGB
- H: number of rows
- W: number of columns



A grayscale 8×8 digit has 64 values; an RGB 224×224 image has 150,528 values before modeling begins.

Why a Naïve MLP Is a Poor Fit for Images

Flattening treats every pixel-channel value as a separate input:

$$X \in \mathbb{R}^{C\times H\times W} \longrightarrow x \in \mathbb{R}^{CHW}.$$

A dense layer with m hidden units needs $mCHW$ weights. Thus

$$\underbrace{3 \times 224 \times 224}_{150{,}528 \text{ inputs}} \xrightarrow{m=1{,}000} \text{150,528,000 first-layer weights.}$$

With inputs this large, a naïve MLP scales poorly. It also relearns the same edge separately at every location.

We need to exploit locality and repeated patterns to reduce the network's parameter count.

Classical Image Filters Already Use These Two Ideas

Long before CNNs, image processing applied a small, hand-designed filter at every image location:

$$\underbrace{\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}}_{\text{blur: low-pass}} \qquad \underbrace{\begin{bmatrix}-1 & 0 & 1\\ -1 & 0 & 1\\ -1 & 0 & 1\end{bmatrix}}_{\text{vertical edge}} \qquad \underbrace{\begin{bmatrix}0 & -1 & 0\\ -1 & 5 & -1\\ 0 & -1 & 0\end{bmatrix}}_{\text{sharpen}}.$$

- Locality: each output uses only a nearby 3×3 patch.
- Pattern reuse: the same nine filter entries are applied at every location.

A Blur Uses Only Nearby Pixels

Input image

Fixed filter, multiplied by 1/9

1	1	1
1	1	1
1	1	1

Blurred output

Each output averages one 3 × 3 neighborhood. Reusing the same average everywhere smooths local variation across the image.

Public-domain U.S. Navy photograph of Grace Hopper, distributed with Matplotlib.

An Edge Filter Finds the Same Pattern Everywhere

Input image

Fixed 3×3 filter

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

Vertical-edge response

The filter contrasts the left and right sides of each local patch. Large responses appear wherever that vertical-edge pattern occurs.

Public-domain U.S. Navy photograph of Grace Hopper, distributed with Matplotlib.

A Sharpening Filter Emphasizes Local Contrast

Input image

Fixed 3×3 filter

0	-1	0
-1	5	-1
0	-1	0

Sharpened output

The center pixel receives positive weight and its four neighbors negative weight. The same local contrast rule is reused across the image.

Public-domain U.S. Navy photograph of Grace Hopper, distributed with Matplotlib.

Outline

A Convolutional Layer Learns and Reuses Filters

A classical filter fixes its numbers by hand. A convolutional layer makes those numbers **learnable parameters**.

At image location (i, j), a 3×3 filter K meets one local patch:

$$Y_{i,j} = b + \left\langle K,\ \text{patch}_{i,j}(X) \right\rangle .$$

- The 3×3 patch makes the calculation local.
- Sliding the same learned K across the image reuses the pattern.
- All responses together form the output **feature map** Y.

Reference Formula: One Local Patch Inner Product

Let $K \in \mathbb{R}^{k_h \times k_w}$ be one filter. At output location (i, j), the corresponding image patch is

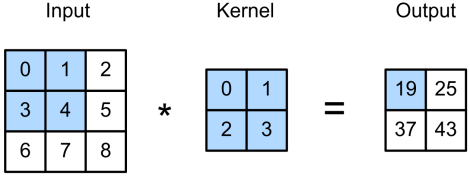
$$\text{patch}_{i,j}(X) = \Big[X_{i+u,j+v}\Big]_{\substack{u=0,\dots,k_h-1 \\ v=0,\dots,k_w-1}} \in \mathbb{R}^{k_h \times k_w}.$$

Its inner product with K is the concrete double sum

$$\boxed{Y_{i,j} = b + \Big\langle K, \text{patch}_{i,j}(X) \Big\rangle = b + \sum_{u=0}^{k_h-1} \sum_{v=0}^{k_w-1} K_{u,v} X_{i+u,j+v}.}$$

- (i, j) selects the output location; (u, v) moves inside the filter.
- The same K and bias b are used for every location (i, j).

One Patch Produces One Output Value



The upper-left output is

$$0(0) + 1(1) + 3(2) + 4(3) = 19.$$

A 3×3 input and a 2×2 kernel admit two legal starts in each direction, hence a 2×2 output.

Moving the same kernel one column gives 25; moving it one row gives 37.

Source: D2L, “Convolutions for Images”; CC BY-SA 4.0.

One Local Calculation

derive

With

$$\text{patch} = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \qquad K = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix},$$

the output at this location is

$$\langle K, \text{patch} \rangle = (0 + 1 + 1) - (0 + 0 + 0) = \boxed{2}.$$

Sliding the same K across the image produces one number per location. Large positive or negative responses identify oppositely oriented horizontal edges.

Sliding the Filter Fills the Entire Feature Map

Apply the same 3×3 filter at each valid location:

$$X = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}, \qquad K = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}.$$

$$Y = \begin{bmatrix} \langle K, X_{1:3,1:3} \rangle & \langle K, X_{1:3,2:4} \rangle & \langle K, X_{1:3,3:5} \rangle \\ \langle K, X_{2:4,1:3} \rangle & \langle K, X_{2:4,2:4} \rangle & \langle K, X_{2:4,3:5} \rangle \\ \langle K, X_{3:5,1:3} \rangle & \langle K, X_{3:5,2:4} \rangle & \langle K, X_{3:5,3:5} \rangle \end{bmatrix} = \begin{bmatrix} -2 & 2 & 2 \\ -2 & 1 & 1 \\ -1 & -1 & -1 \end{bmatrix}.$$

The input has $5 - 3 + 1 = 3$ valid vertical starts and the same number of horizontal starts, so the response map is 3×3.

The Learned Weights Are Small Spatial Patterns

Each 3 × 3 matrix is one fitted filter. Its nine entries are learned once, then the same matrix is reused at every image location.

Course experiment: first layer of the seeded digit CNN trained later in this lecture.

Property: Translation Equivariance

A convolutional filter searches for the same local pattern at every image location.

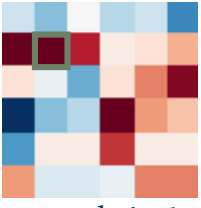
- Shift the pattern to a new location in the input image.
- Its response shifts to the corresponding location in the feature map.

The output is not unchanged: it moves in the same way as the input.

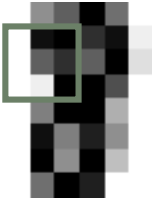
The Feature Map Moves with the Pattern

Input

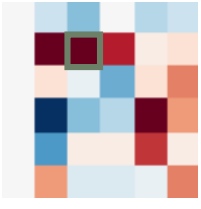
Response



Input shifted right



Response shifted right



Course calculation using the same fixed edge filter. Boundary padding determines the exact behavior at the image edge.

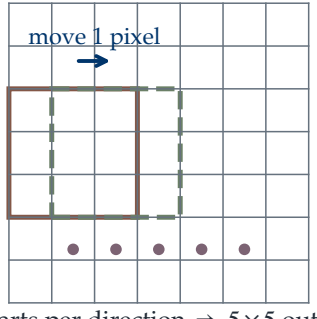
Outline

How Large Is the Output of a Convolutional Layer?

The filter size determines what each response sees. Two more choices determine how many responses we obtain: how far the filter moves, and whether it may extend beyond the original image border.

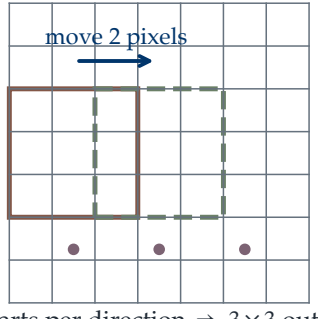
Stride Controls How Far the Filter Moves

Stride 1



5 starts per direction $\Rightarrow$ 5×5 output

Stride 2



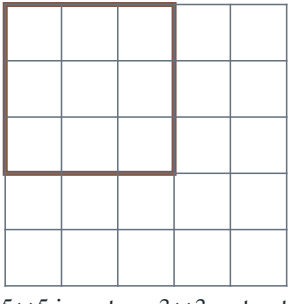
3 starts per direction $\Rightarrow$ 3×3 output

Stride is the number of pixels between successive filter starts. Larger stride evaluates fewer patches, producing a smaller feature map.

A 3×3 filter moving across a 7×7 input.

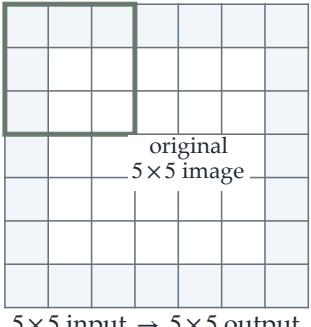
Padding Lets the Filter Reach the Border

No padding



5×5 input → 3×3 output

One-cell zero padding



5×5 input → 5×5 output

Padding adds a border, usually filled with zeros. Without it the output shrinks; with a one-cell border, a 3 × 3 filter can preserve size.

Stride one and a 3 × 3 filter.

Count the Legal Filter Starts

After padding, the input height is $H + 2p_h$. A filter of height k_h can start at

$$0,\ s_h,\ 2s_h,\ \ldots \quad \text{while the start is at most} \quad H + 2p_h - k_h.$$

Therefore the number of vertical starts is

$$\boxed{H_{\text{out}} = \left\lfloor \frac{H + 2p_h - k_h}{s_h} \right\rfloor + 1}.$$

The same count gives W_{out}.

For $H = 7$ and $k_h = 3$: stride one gives starts $0, 1, 2, 3, 4$; stride two gives starts $0, 2, 4$. Thus the output heights are 5 and 3.

A Convolution Uses Far Fewer Parameters

A layer with C_{in} input channels, C_{out} output channels, and $k_h \times k_w$ filters has

$$\boxed{C_{\text{out}}(C_{\text{in}}k_hk_w + 1)}$$

trainable parameters, independent of H and W.

Example: $C_{\text{in}} = 3$, $C_{\text{out}} = 64$, and 3×3 filters:

$$64(3 \cdot 3 \cdot 3 + 1) = 1{,}792.$$

A dense layer from a $224 \times 224 \times 3$ image to only 64 hidden units would require more than 9.6 million parameters.

Outline

A Filter Spans All Input Channels

For a minibatch tensor $X \in \mathbb{R}^{N \times C_{\text{in}} \times H \times W}$,

$$Y_{n,r,i,j} = b_r + \sum_{c=1}^{C_{\text{in}}} \sum_{a,b} K_{r,c,a,b} X_{n,c,i+a,j+b}.$$

r: output channel c: input channel

Each output channel r has its own bank of filters across all input channels c. The input-channel responses are summed before applying the activation.

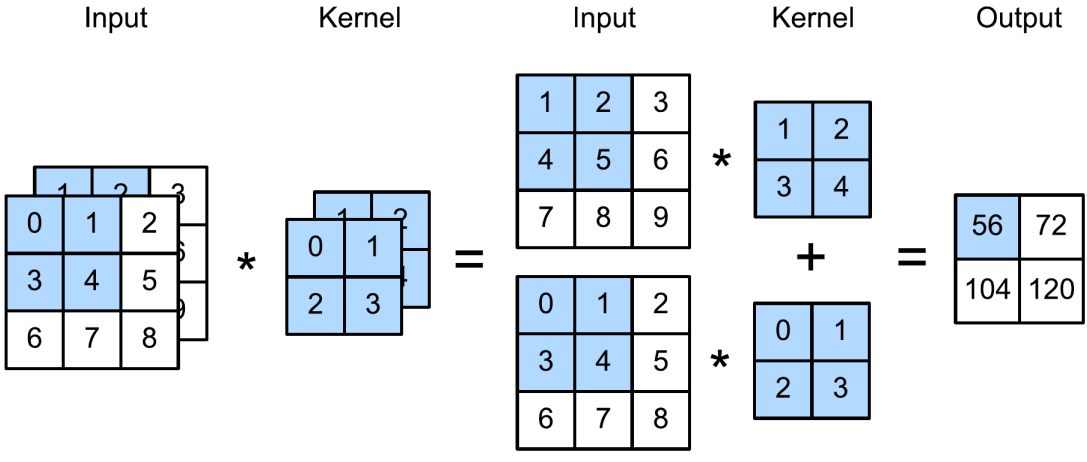
RGB is only the input interpretation. In a hidden layer, a channel is a learned response map, not necessarily a human-named color or concept.

Every Convolutional Tensor Has Four Named Axes

object	shape	meaning
input X	$N \times C_{\text{in}} \times H \times W$	examples, channels, height, width
filters K	$C_{\text{out}} \times C_{\text{in}} \times k_h \times k_w$	one filter bank per output channel
bias b	C_{out}	one scalar per output channel
output Y	$N \times C_{\text{out}} \times H_{\text{out}} \times W_{\text{out}}$	one feature map per output channel

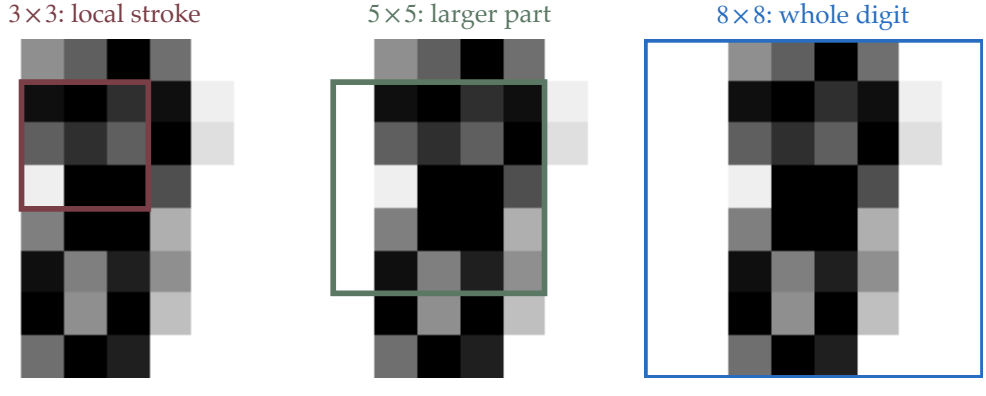
A convolution changes the channel count and possibly the spatial size; it never changes the minibatch size N.

Input Channels Are Summed into Each Output Channel



D2L, "Multiple Input and Multiple Output Channels"; CC BY-SA 4.0.

Later Layers Need Information at Larger Scales



One 3×3 filter sees a local stroke. Later layers must combine evidence over larger parts and eventually the whole digit. Pooling builds a coarser map so a later filter covers more of the original image.

Course illustration using one held-out scikit-learn digit.

Pooling Builds a Coarser Feature Map

Pooling replaces each small block of nearby responses by one summary. The most common choice, **max pooling**, keeps the largest response in the block.

- It has no learned weights.
- It reduces spatial resolution and computation.
- A later 3×3 filter covers a larger region of the original image.
- The tradeoff is loss of precise location and amplitude information.

Stacking convolutions also enlarges the receptive field; pooling makes the enlargement faster while explicitly changing to a coarser spatial scale.

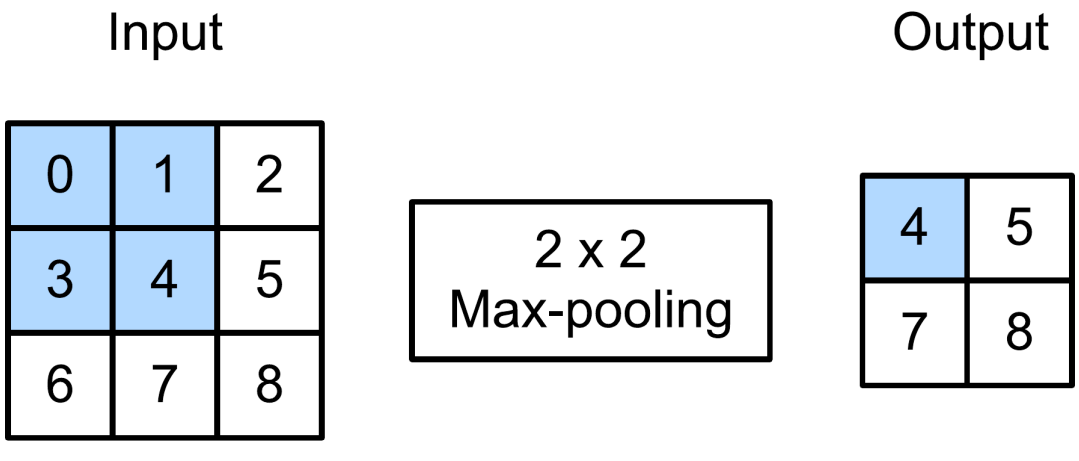
One 2×2 Max-Pooling Step Is Just Four Maxima

With nonoverlapping windows and stride two,

$$\begin{bmatrix} 1 & 3 & 2 & 0 \\ 4 & 6 & 5 & 1 \\ 0 & 2 & 7 & 8 \\ 1 & 3 & 4 & 9 \end{bmatrix} \longmapsto \begin{bmatrix} \max(1,3,4,6) & \max(2,0,5,1) \\ \max(0,2,1,3) & \max(7,8,4,9) \end{bmatrix} = \boxed{\begin{bmatrix} 6 & 5 \\ 3 & 9 \end{bmatrix}}.$$

Pooling operates independently in every channel. It changes spatial size but not the number of channels and introduces no trainable parameters.

Maximum Pooling Keeps the Strongest Local Response



D2L, "Pooling"; CC BY-SA 4.0. Blue windows show the pooling neighborhoods.

Average Pooling Keeps the Local Mean

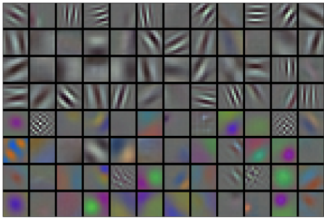
Use the same nonoverlapping 2×2 windows and stride two, but replace each block by its average:

$$\begin{bmatrix} 1 & 3 & 2 & 0 \\ 4 & 6 & 5 & 1 \\ 0 & 2 & 7 & 8 \\ 1 & 3 & 4 & 9 \end{bmatrix} \longmapsto \begin{bmatrix} (1+3+4+6)/4 & (2+0+5+1)/4 \\ (0+2+1+3)/4 & (7+8+4+9)/4 \end{bmatrix} = \boxed{\begin{bmatrix} 3.5 & 2 \\ 1.5 & 7 \end{bmatrix}}.$$

- **Max pooling** keeps the strongest response in each block.
- **Average pooling** keeps the mean response in each block.

Both have no trainable parameters and reduce the spatial resolution.

What First-Layer Filters Learn



Source: D2L reproduction of AlexNet filters; Krizhevsky et al. (2012).

- Many kernels are localized and oriented: they look like edge-like patterns at different orientations.
- Others respond to color contrasts or lower-frequency blobs.
- These were learned for classification; no theorem forces this form.

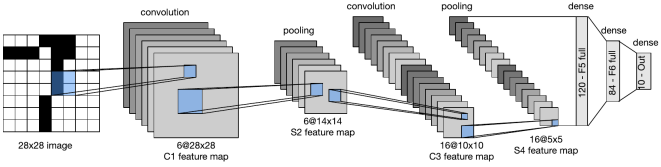
A Deep ResNet Contains Concept-Selective Units

Each small row shows the image regions that most strongly activate one hidden unit. Some ResNet-152 units repeatedly respond to recognizable concepts such as houses, dogs, trains, plants, and airplanes.

This alignment is empirical: not every unit has a simple human label.

Source: David Bau, Network Dissection examples; Bau et al. (CVPR 2017).

LeNet Alternates Feature Extraction and Downsampling



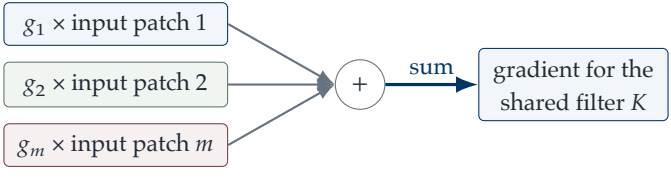
Convolution preserves spatial structure; pooling reduces it; the final dense head converts the learned representation to class scores.

Source: D2L, “LeNet”; CC BY-SA 4.0. Architecture after LeCun et al.

Outline

Convolution Uses Ordinary Backpropagation

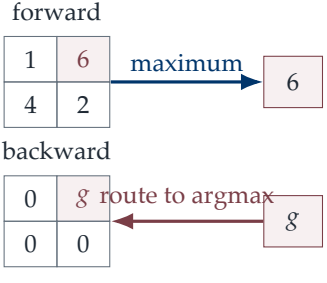
A convolution is a **linear layer**. The only extra bookkeeping comes from reusing one filter on many input patches.



Add one contribution from every patch where K was used. Automatic differentiation performs this sum.

Max-Pooling Backpropagation Follows the Argmax

During the forward pass, max pooling records which input attained the maximum. The backward pass sends the arriving gradient only to that cell.



The argmax receives g; every other entry receives zero.

Initialize a Convolution by Its Fan-In, Not Image Size

Each pre-activation sums $\text{fan}_{\text{in}} = C_{\text{in}} k_h k_w$ inputs. For ReLU-family activations, He initialization uses

$$K_{r,c,a,b} \sim \mathcal{N}\left(0, \frac{2}{C_{\text{in}} k_h k_w}\right), \qquad b_r = 0.$$

The scale depends on channels and kernel area, not H or W, because one output location never sums all image pixels.

Xavier balances fan-in and fan-out for roughly symmetric activations; He compensates for the part of a symmetric signal removed by ReLU.

Source: Glorot and Bengio (2010); He et al. (2015).

Very Deep Plain CNNs Became Harder to Optimize

A deeper network can represent a shallower one by making its extra layers the identity, so depth does not force a higher minimum training error.

Yet on CIFAR-10, a plain 56-layer network had higher training error than a plain 20-layer network. This is not overfitting: optimization failed to find the available identity-like solution.

Residual parameterization changes which solution is easy to reach.

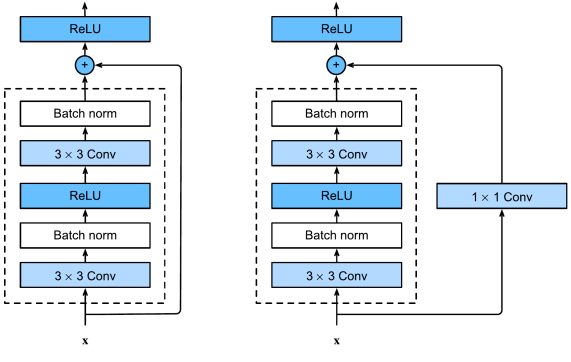
Source: He, Zhang, Ren, and Sun, “Deep Residual Learning,” 2016, Figure 1.

A Residual Block Makes the Identity Path Explicit

A residual block adds a **shortcut** around the main path.

The main path learns $F(x)$; the shortcut carries x. Their sum is $x + F(x)$. If $F(x)$ is near zero, the block preserves its input.

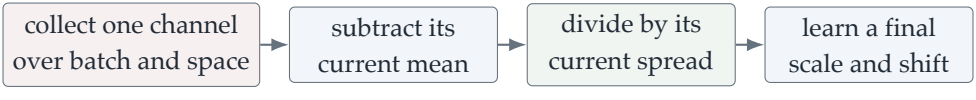
Left: identity shortcut. Right: a 1×1 projection when shapes change.



Source: D2L, "Residual Networks"; CC BY-SA 4.0. Architecture after He et al. (2016).

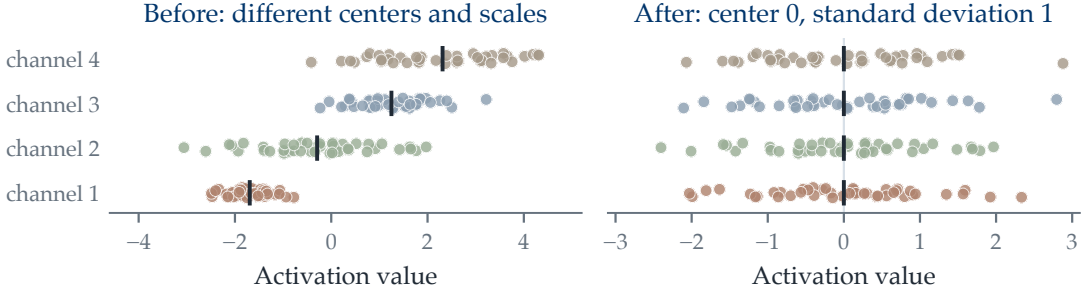
Batch Normalization Rescales One Channel at a Time

During training, different feature channels can develop very different centers and scales. Batch normalization applies the same four-step operation to each channel separately.



The first three steps create a common numerical scale; the last step lets the network choose the scale it actually needs.

Batch Normalization Brings Channels to a Common Scale



Course illustration. Each row is one channel; the dark tick marks its mean. The learned final scale and shift are omitted to isolate standardization.

Batch Normalization Has Two Modes

training: `model.train()`

use the current minibatch
update running averages

prediction: `model.eval()`

use stored running averages
produce deterministic predictions

Call `model.eval()` before validation or prediction. The normalization rule is part of the fitted network, not a preprocessing step that can be changed afterward.

Source: Ioffe and Szegedy (2015); D2L, "Batch Normalization."

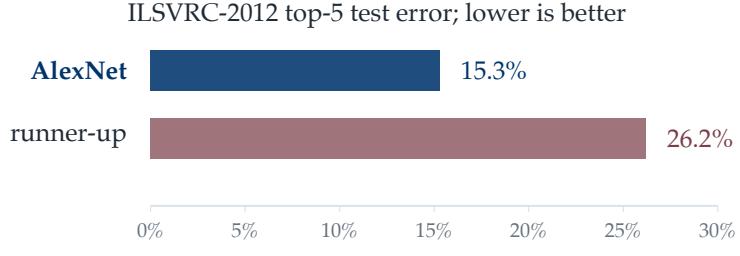
The Stabilizers Solve Different Problems

mechanism	when it acts	primary role
He/Xavier initialization	before step 1	sensible signal scale
batch normalization	every forward pass	control channel scale
residual connection	every block	short identity and gradient path
augmentation + weight decay	across training	reduce overfitting
learning-rate schedule	across epochs	control optimization progress

These tools are complementary. A normalization layer does not replace a good optimizer; a residual path does not replace regularization; augmentation does not repair an incorrect tensor shape.

Outline

ImageNet 2012: AlexNet versus the Runner-Up



AlexNet combined five convolutional layers, ReLUs, max pooling, GPU training, augmentation, and dropout. Its success joined architecture, data, computation, and regularization.

Source: Krizhevsky, Sutskever, and Hinton (2012).

A Small CNN in PyTorch

```
model = nn.Sequential(
    nn.Conv2d(1, 8, kernel_size=3, padding=1),
    nn.BatchNorm2d(8),
    nn.ReLU(),
    nn.MaxPool2d(2),
    nn.Conv2d(8, 16, kernel_size=3, padding=1),
    nn.BatchNorm2d(16),
    nn.ReLU(),
    nn.Flatten(),
    nn.Linear(16 * 4 * 4, 10),
)
```

PyTorch stores images in `(batch, channels, height, width)` order. The next slide tracks how every operation changes this shape.

Make the Initialization Choice Explicit

```
def initialize_cnn(module):
    if isinstance(module, nn.Conv2d):
        nn.init.kaiming_normal_(
            module.weight, mode="fan_in", nonlinearity="relu"
        )
        if module.bias is not None:
            nn.init.zeros_(module.bias)

model.apply(initialize_cnn)
```

PyTorch can initialize layers automatically, but an explicit rule records the intended activation and makes experiments reproducible. Batch-normalization scale and shift are conventionally initialized to 1 and 0.

Derive the First CNN Block before Running It

Begin with $X : [N, 1, 8, 8]$. The first convolution has $k = 3$, $p = 1$, $s = 1$, and $C_{\text{out}} = 8$:

$$H_{\text{out}} = W_{\text{out}} = \left\lfloor \frac{8 + 2(1) - 3}{1} \right\rfloor + 1 = 8,$$

$$\text{shape}(Y) = [N, 8, 8, 8], \qquad \#\theta = 8(1 \cdot 3 \cdot 3 + 1) = 80.$$

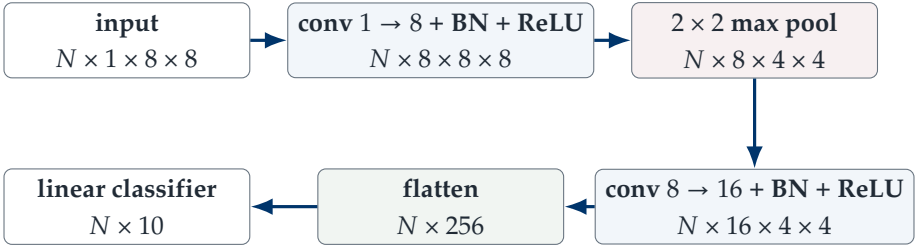
Batch normalization and ReLU preserve this shape. A 2×2 max pool with stride two then gives

$$[N, 8, 8, 8] \longrightarrow [N, 8, 4, 4].$$

The same three checks—spatial formula, output channels, parameter count—apply to every later block.

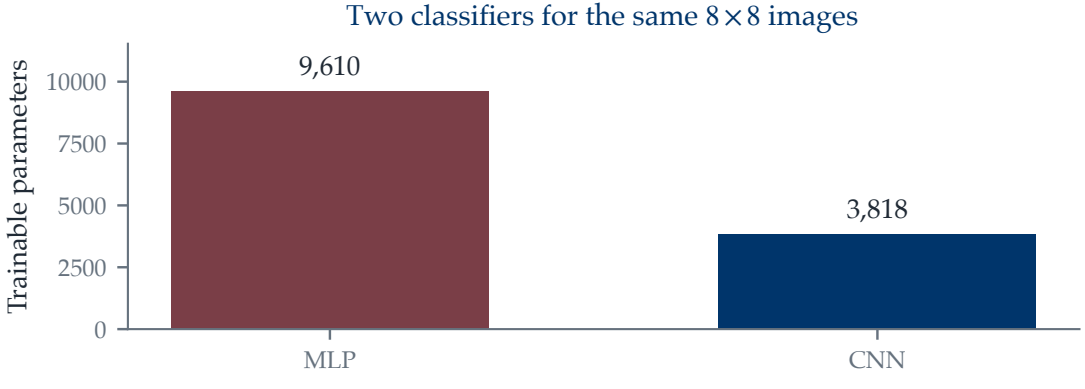
Track Tensor Shapes Through the Network

PyTorch image order: batch × channels × height × width



BN and ReLU preserve shape; pooling halves spatial size; the second convolution changes the channel count from 8 to 16.

Weight Sharing Reduces the Parameter Count



Course models: MLP 64 → 128 → 10; CNN 1 → 8 → 16 with one pooling layer and a dense head. The plotted CNN omits batch norm for a direct weight-sharing comparison.

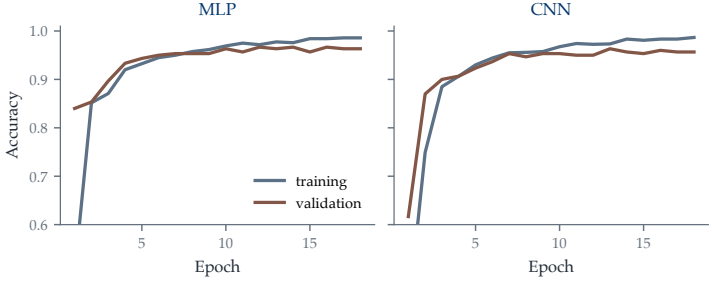
The Training Loop Is the Same Interface

```
for images, labels in train_loader:
    model.train()
    optimizer.zero_grad()
    logits = model(images)
    loss = loss_fn(logits, labels)
    loss.backward()
    optimizer.step()

model.eval()
with torch.no_grad():
    validation_logits = model(validation_images)
```

The model architecture changes the forward graph. The optimizer still reads gradients produced by automatic differentiation, as in Lectures 11–12.

MLP and CNN Training on the Same Split



Course experiment: common split, 18 AdamW epochs, batch size 64; the CNN omits batch normalization.

A Tie on This Small Test Set Is Still Informative

On the untouched 297-image test split, both specified models obtain

$$\text{accuracy} = 97.98\%.$$

This does not prove the architectures are equivalent. It says this small, centered, low-resolution data set is not enough evidence for a performance claim.

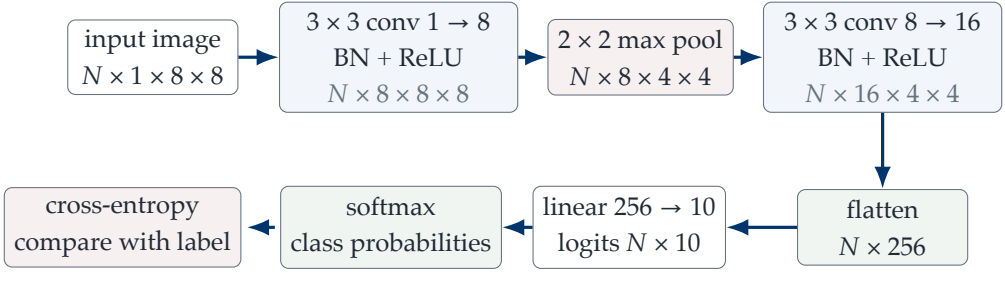
A stronger comparison would vary training size, translations, label noise, architecture size, tuning budget, and random seed while reporting uncertainty and wall-clock cost.

Common CNN Failure Modes

1. **Shape mismatch:** print every intermediate tensor shape.
2. **Data mismatch:** verify channel order, scaling, labels, and augmentation.
3. **No learning:** inspect loss, gradient norms, and learning rate.
4. **Train–validation gap:** inspect capacity, leakage, augmentation, and decay.
5. **Unstable evaluation:** switch dropout and batch normalization to evaluation mode.

Inspect predictions and feature responses in addition to a single accuracy number.

The Complete CNN: Input to Prediction



Backpropagation follows the arrows in reverse and updates every convolutional filter, normalization parameter, and classifier weight.

CNN Training: A Practical Recipe

1. **Initialize:** use He initialization for ReLU convolutional filters, zero biases, and batch-normalization scale/shift $(\gamma, \beta) = (1, 0)$.
2. **Build each block:** use `Conv--BN--ReLU`; add residual paths when depth makes identity mappings difficult to optimize.
3. **Feed data:** shuffle each epoch, train with minibatches, and use label-preserving augmentation such as crops or flips when appropriate.
4. **Update parameters:** backpropagate cross-entropy; use SGD or AdamW with a tuned learning rate, schedule, and weight decay.
5. **Validate correctly:** track training and validation curves, inspect shapes and gradients, and call `model.eval()` so batch normalization and dropout use prediction-time behavior.

Summary: What Convolution Assumes

$$(f * w)(i,j) = \sum_u \sum_v f(i+u,\ j+v) \cdot w(u,v)$$

1. A pixel is best explained by its neighbours, so a unit reads a small window rather than the whole image.
2. The same pattern is worth detecting anywhere, so one kernel is shared across every position and the parameter count stops growing with image size.
3. Stacking layers widens the receptive field, so later units see structure that no single kernel could reach.

Summary: What Makes the Stack Trainable

1. **Stride and padding** fix the output shape, and the shape arithmetic has to close before anything trains.
2. **Channels** carry the learned feature vocabulary; depth turns edges into parts and parts into objects.
3. **Batch normalization** and **residual paths** keep gradients usable at depth, and `model.eval()` switches them to prediction-time behaviour.

Summary: What Makes the Stack Trainable

1. **Stride and padding** fix the output shape, and the shape arithmetic has to close before anything trains.
2. **Channels** carry the learned feature vocabulary; depth turns edges into parts and parts into objects.
3. **Batch normalization** and **residual paths** keep gradients usable at depth, and `model.eval()` switches them to prediction-time behaviour.

An architecture that trains is not an architecture that generalizes. The evidence is the validation curve, not the loss.

Next Lecture

Convolutional networks learn supervised representations for images. If labels are absent, we need a different criterion for deciding which directions and low-dimensional structure in the data are worth keeping.

Lecture 14 — PCA, SVD, and Low-Rank Approximation connects maximum variance to minimum reconstruction error, derives principal directions, and expresses the solution through the singular value decomposition.

Reading for this lecture: D2L Chapter 7, Convolutional Neural Networks; DL Chapter 9.