
Convolutions and Image Architectures

ECE 57000 — October 21, 2026

David I. Inouye

| Convolution combines **local connections** and **shared parameters**

- A filter spans all input channels; each learned filter produces one output channel.
- Kernel size, padding, and stride determine spatial output size; filter count determines channel count.
- Stack convolutions with nonlinearities to expand the receptive field; reuse the same training loop.

Today: finish and evaluate the simple CNN, then read ResNet and U-Net architectures.

| A convolution is a sparse matrix with shared entries

Input image

x_{11}	x_{12}	x_{13}
x_{21}	x_{22}	x_{23}

Shared kernel

a	b
c	d

One row per output position

a	b	0	c	d	0
0	a	b	0	c	d

2 outputs $\times$ 6 inputs

Vectorize the image $\rightarrow$ multiply by the matrix $\rightarrow$ reshape

Zeros = local connections • repeated letters = shared parameters

Four learned kernel weights; the matrix entries are constrained.

$$\text{vec}(Y) = A_K \text{vec}(X), \quad A_K = \begin{bmatrix} a & b & 0 & c & d & 0 \\ 0 & a & b & 0 & c & d \end{bmatrix}.$$

A small CNN changes the model but reuses the training loop

```
1 cnn = nn.Sequential(  
2     nn.Conv2d(1, 8, 3, padding=1),  
3     nn.ReLU(),  
4     nn.Conv2d(8, 16, 3, stride=2, padding=1),  
5     nn.ReLU(),  
6     nn.Flatten(),  
7     nn.Linear(16 * 14 * 14, 10))  
8 a = torch.zeros(64, 1, 28, 28)  
9 for layer in cnn:  
10     a = layer(a)  
11     print(type(layer).__name__, tuple(a.shape))
```

Output:

```
1 Conv2d (64, 8, 28, 28)  
2 ReLU (64, 8, 28, 28)  
3 Conv2d (64, 16, 14, 14)  
4 ReLU (64, 16, 14, 14)  
5 Flatten (64, 3136)  
6 Linear (64, 10)
```

| Keep the pixels, **change their arrangement**

Randomly permute pixel positions. Use the **same permutation for every image**, then retrain both models.

Which model's architectural advantage would you expect to shrink?

A. MLP

B. CNN

C. Both equally

B. CNN. The information remains, but nearby entries no longer form local image patterns.

| Convolution provides **structure, not a guarantee of accuracy**

- **Locality:** each output sees a patch, expanded by stacking layers.
- **Sharing:** the same filters are reused across positions.
- **Shape control:** filters choose channels; kernel, padding, and stride choose spatial sizes.

Next, compare CNN and MLP using the same data splits and a documented training/selection protocol.

The design is now explicit. Its benefit is an empirical question.

Testing Architectural Assumptions

| Architectural assumptions need **empirical evidence**

Arc 5: CNN fundamentals → **empirical evidence** → ResNet classification → U-Net segmentation.

1. **Compare:** what changes under the same training recipe?
2. **Stress:** what happens when image position changes?
3. **Decide:** which conclusions does the evidence support?

Locality and sharing motivate the CNN. Now test what they buy us.

| Compare architectures under **one stated recipe**

Keep fixed	Change
Same 12,000 training / 3,000 validation images	MLP: 784 → 64 → 10
Same batches, five epochs, Adam at 10^{-3}	CNN: the two-convolution model just shown
Three seeds; use the final epoch of every run	Connections, parameter count, and computation

Equal epochs give equal data passes—not equal computation.

No augmentation, tuning, or test-set access in this comparison.

| This CNN improves validation accuracy **under this recipe**

Model	Mean accuracy	Range across three seeds
MLP	83.4%	83.0–83.7%
CNN	86.0%	85.7–86.3%

Same 3,000 validation images; five training epochs per run.

Supported: this CNN recipe performed better on this split.

Not established: every CNN beats every MLP, or the gap persists under equal compute.

| Fewer parameters can still require **more computation**

Complete model	Learned parameters	Multiply-accumulates per image
MLP	50,890	50,816
CNN	32,618	313,600

A convolution **stores a filter once but applies it at many positions.**

This CNN has about **36% fewer parameters** and **6.2× as many multiply-accumulates.**

| A better average can hide **different mistakes**

Label: T-shirt/top



MLP: Trouser
CNN: T-shirt/top

Label: Sneaker



MLP: Sneaker
CNN: Ankle boot

The CNN fixes one MLP error; the MLP gets another example right.

Inspect disagreements against the label before deciding which errors matter.

Stress the assumption about **position**

Architecture evaluation · 2/3 Stress · Compare → **Stress** → Decide

| Predict what happens when the image moves

A CNN uses convolution, then flattening and a linear classifier.

Must shifting an image preserve its predicted class?

A. Yes, filters are shared

B. Only for one-pixel shifts

C. No, the classifier can use position

C. A pattern can move in the feature map; the classifier can weight positions differently.

| A shared filter can feed a position-sensitive decision

Use a size-one filter with weight 1 and bias 0. It copies each input value.

	Original	Shift right
Input and feature map	[0, 1, 0]	[0, 0, 1]
Class A score: middle entry	1	0
Class B score: right entry	0	1
Prediction	A	B

Same filter at every position; different class after a shift.

| Define the shift **before** interpreting its score

Original



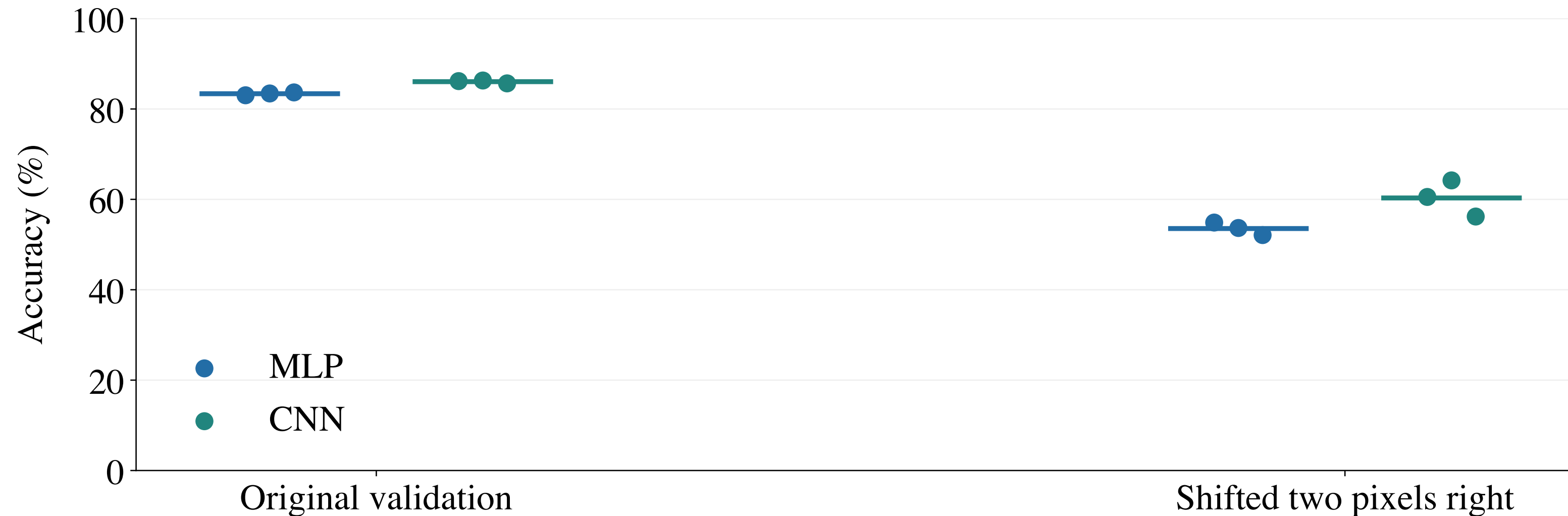
Two pixels right; zero fill



Move each validation image **two pixels right**. Fill the left edge with zeros; discard the rightmost two columns. No wrapping.

Keep the original labels. **This assumes the remaining image still supports the same category.**

| Both models lose accuracy on shifted images



Mean accuracy: MLP 83.4% → 53.5%; CNN 86.0% → 60.3%.

The CNN does better here, but its original validation accuracy **does not establish shift robustness**.

| Choose a model for the conditions it must handle

Architecture evaluation · 3/3 Decide

Evidence so far	Decision it informs
Higher original validation accuracy	Candidate for similarly positioned images
Fewer parameters, more arithmetic	Check storage and runtime budgets separately
Large shifted-image accuracy loss	Investigate position variation before deployment

Compare → **Stress** → **Decide**: a single accuracy number cannot settle all three.

| Architectural bias helps **when its assumptions fit**

- **Assumption:** local patterns recur; sharing is useful when that is true.
- **Evidence:** these two recipes favor the CNN on one split; both degrade under one shift.
- **Next test:** fix the recipe and selection rule, then evaluate on held-out data matching the intended conditions.

More expressive features, efficient gradients, and a working loop are necessary steps. **They do not replace evidence about where the model will be used.**

Image Classification with ResNet

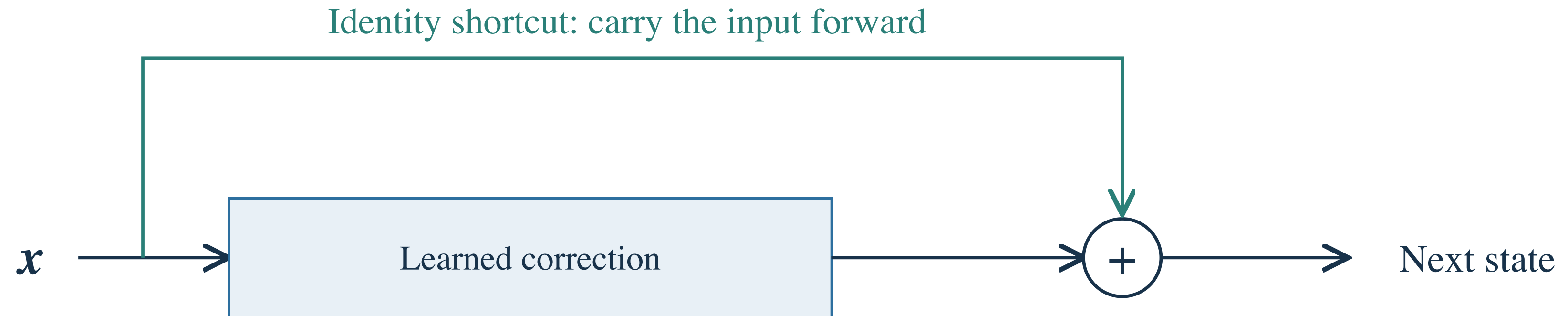
| A realistic classifier combines **updates, normalization, and aggregation**

Arc 5: CNN fundamentals → empirical evidence → **ResNet classification** → U-Net segmentation.

1. **Update:** refine a representation with residual blocks.
2. **Normalize:** control feature scales within the network.
3. **Assemble:** read a complete ResNet-18.

The simple comparison is complete. Now recognize how these components fit into a widely used architecture.

| A residual block learns a change to the representation



If the correction is zero, the representation passes through.

Addition requires matching shapes; use a projection when they differ.

$$\boldsymbol{x}_{\text{next}} = \boldsymbol{x} + f(\boldsymbol{x})$$

| Even a linear residual block starts from a different reference point

Replace the representation

$$x_{\text{next}} = Wx$$

Preserve the input: **learn** $W = I$.

Near $W = 0$, the output is near zero.

Update the representation

$$x_{\text{next}} = (I + W)x$$

Preserve the input: **learn** $W = 0$.

Near $W = 0$, the update is small.

Stacking linear residual blocks still gives a linear map. The change is how we parameterize learning, not the linear model's expressivity.

| The shortcut supplies a **direct gradient contribution**

For the additive block $y = x + f(x)$:

$$\frac{\partial y}{\partial x} = I + \frac{\partial f(x)}{\partial x}.$$

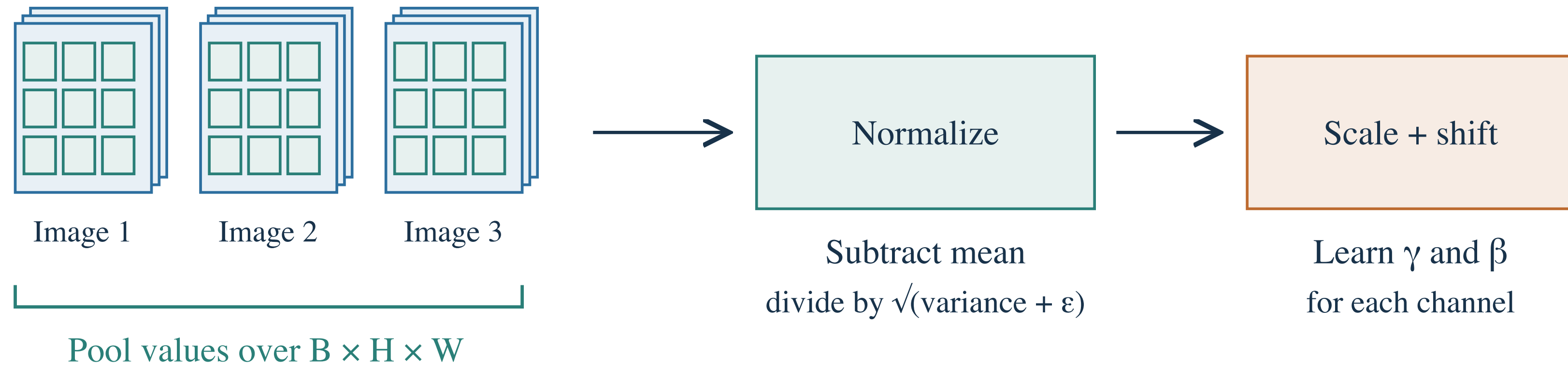
$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} + \left(\frac{\partial f(x)}{\partial x} \right)^T \frac{\partial L}{\partial y}.$$

The gradient has a path that **does not multiply through the learned branch**.

This helps explain deep residual networks; it does not guarantee stable gradients or better generalization.

| Batch normalization uses **one set of statistics per channel**

One channel across a batch



Training: batch statistics • **Evaluation:** stored running statistics

$$y_{bchw} = \gamma_c \frac{x_{bchw} - \mu_c}{\sqrt{v_c + \epsilon}} + \beta_c$$

| Normalization layers differ in **which values share statistics**

Layer	Values grouped for each normalization
BatchNorm	One channel across examples and spatial positions
InstanceNorm	One channel's spatial positions within one example
GroupNorm	A group of channels and spatial positions within one example
LayerNorm	Chosen trailing feature dimensions within one example

ResNet-18 uses BatchNorm. The U-Net example uses InstanceNorm.

The other three need no cross-example statistics. LayerNorm will return when we discuss transformers.