
Image Classification with ResNet (continued)

ECE 57000 — October 23, 2026

David I. Inouye

| Architectural choices need evidence and a matching output

- CNN locality and sharing helped under our measured recipe, but did not guarantee shift robustness.
- Residual blocks add learned updates and a direct gradient path; normalization groups selected values.
- BatchNorm uses current-batch statistics during training and stored running statistics in default evaluation mode.

Today: finish ResNet-18 and U-Net, then define the prediction task for sequence models.

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

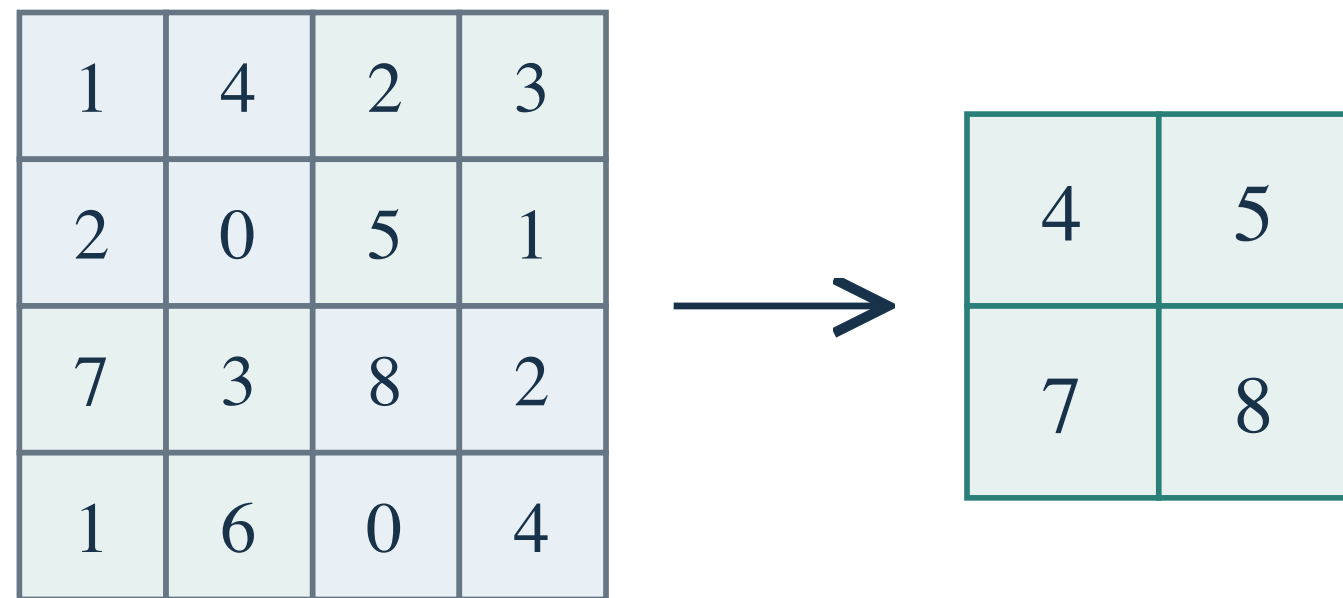
Layer	Values grouped for each normalization
BatchNorm	One channel across examples and spatial positions
InstanceNorm	One channel's spatial positions within one example
GroupNorm	A group of channels and spatial positions within one example
LayerNorm	Chosen trailing feature dimensions within one example

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.

| Pooling either reduces a map or summarizes each channel

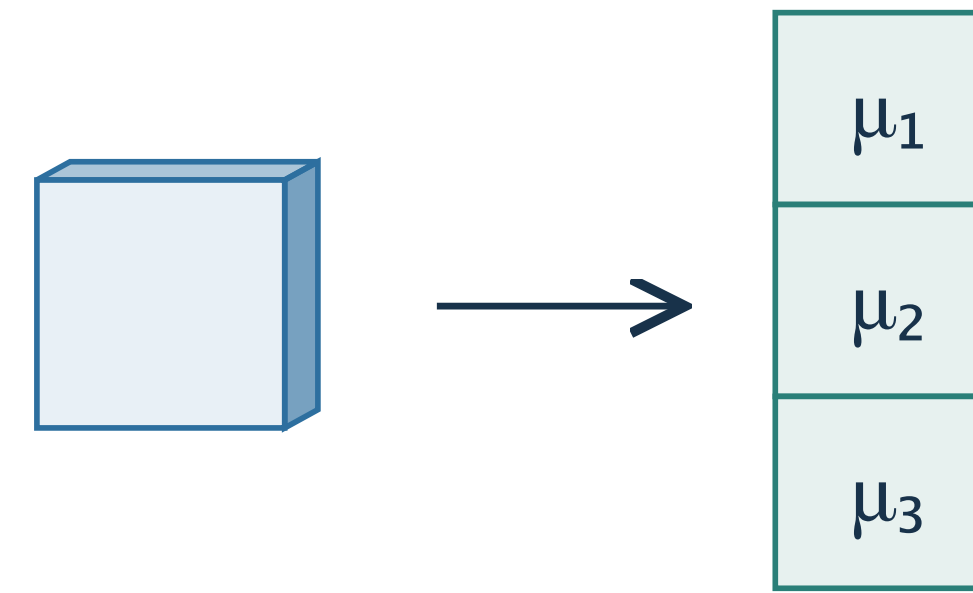
Local max pooling



Keep the largest value in each 2×2 patch

$4 \times 4 \rightarrow 2 \times 2$, separately per channel

Global average pooling



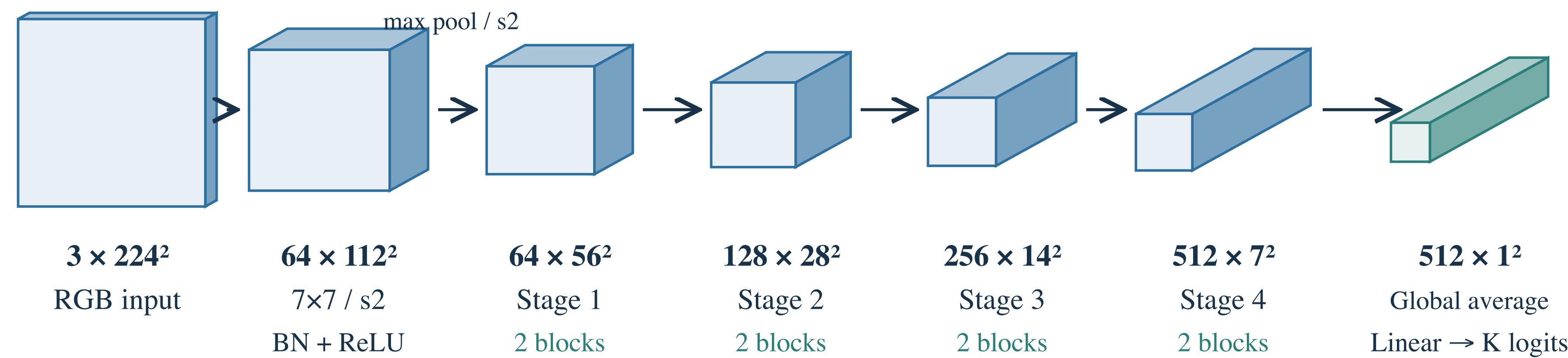
Average each channel over all locations

$C \times H \times W \rightarrow C$ values

ResNet uses both: local pooling near the input; global averaging before the classification head.

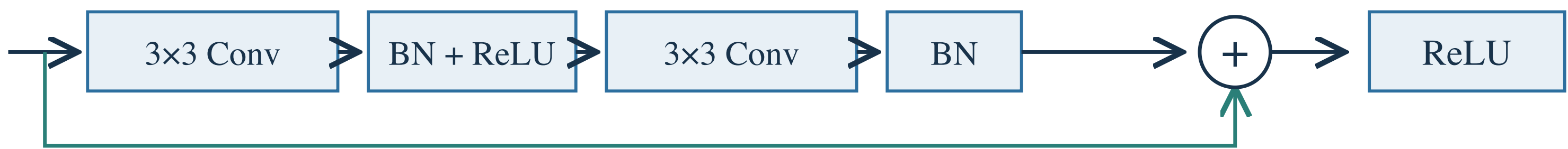
| ResNet-18 uses smaller spatial maps and more channels

ResNet-18 • RGB image classification



Front face: height $\times$ width • Box depth: channels • Sizes compressed for readability

Inside each basic block



The shortcut preserves shape or projects to match it.

At stage changes:

shortcut: 1×1 / s2
Conv + BN

Original illustration of torchvision ResNet-18; dimensions omit the batch axis.

| A prediction should not depend on **unrelated batch companions**

A trained BatchNorm classifier gives different scores for one image when its batch companions change.

What should you check first?

A. Whether `model.eval()` was called

B. Whether `optimizer.zero_grad()` was called

C. Whether the model needs more filters

A. Default BatchNorm uses stored running statistics in evaluation mode; training mode uses the current batch.

Image Segmentation with U-Net

| Segmentation needs a prediction at every image location

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

Task → Upsampling → Architecture

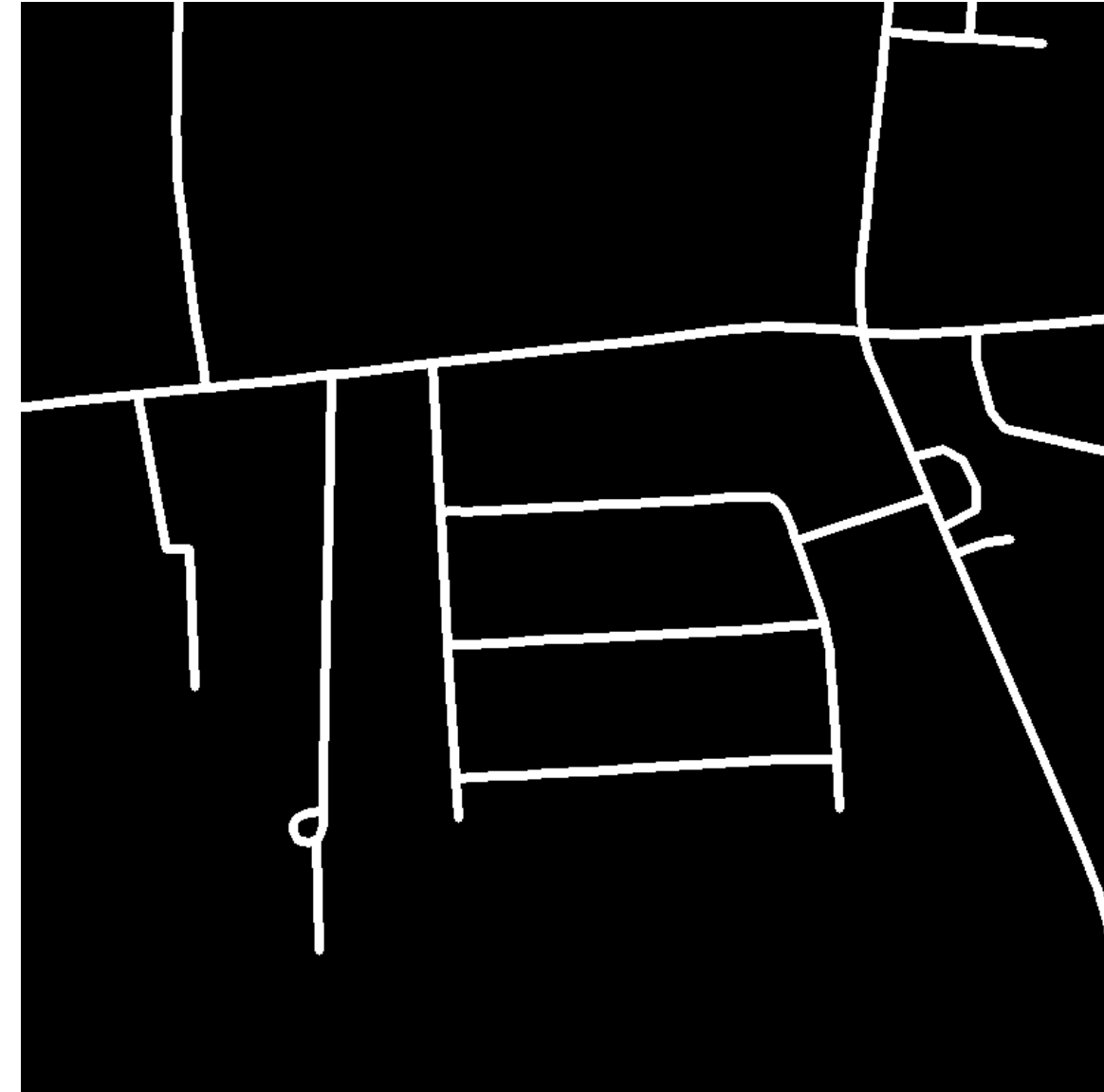
A classifier summarizes an image. A segmentation model must retain where each class occurs.

| A road map needs a **label** at every pixel

Input: aerial RGB image



Target: road / background



Provided annotation, not a model prediction • Massachusetts Roads, Mnih (2013)

Mnih, Massachusetts Roads (2013), validation tile 10228690_15; aligned crop.

| The input can stay the same while **the output becomes spatial**

	Image classification	Semantic segmentation
RGB input	$[B, 3, H, W]$	$[B, 3, H, W]$
Model logits	$[B, K]$	$[B, K, H, W]$
Integer targets	$[B]$	$[B, H, W]$

For roads: **two logits per pixel**, road or background.

Reuse cross-entropy, averaged over labeled pixels. **Keep the spatial grid instead of globally pooling it away.**

| Transposed convolution uses **the transpose of the linear operator**

Convolution: 6 inputs $\rightarrow$ 2 outputs

Transpose: 2 inputs $\rightarrow$ 6 outputs

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

A_K

rows become columns



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

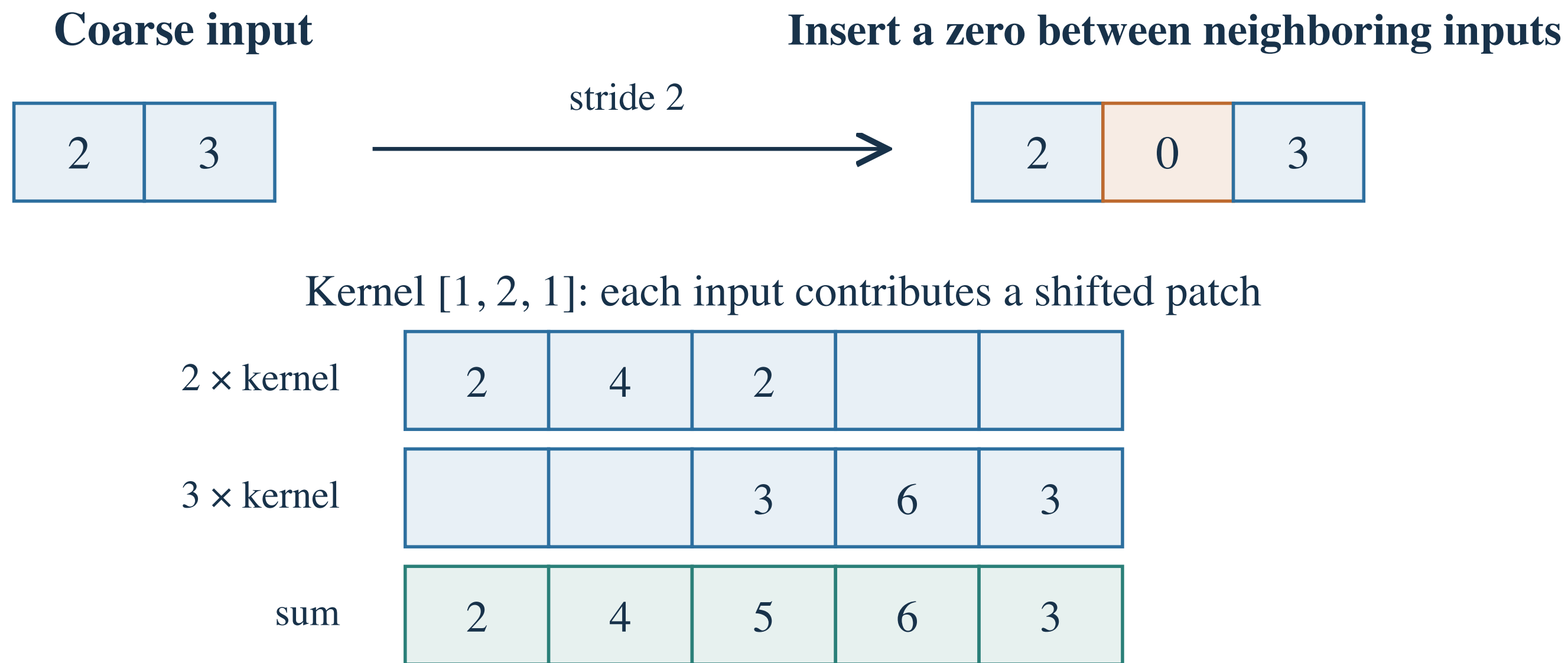
A_K^T

Collect weighted sums $\rightarrow$ scatter weighted patches; overlaps add.

Transpose is not inverse: increasing size does not undo information loss.

$$y = A_K x \quad \longrightarrow \quad u = A_K^T z$$

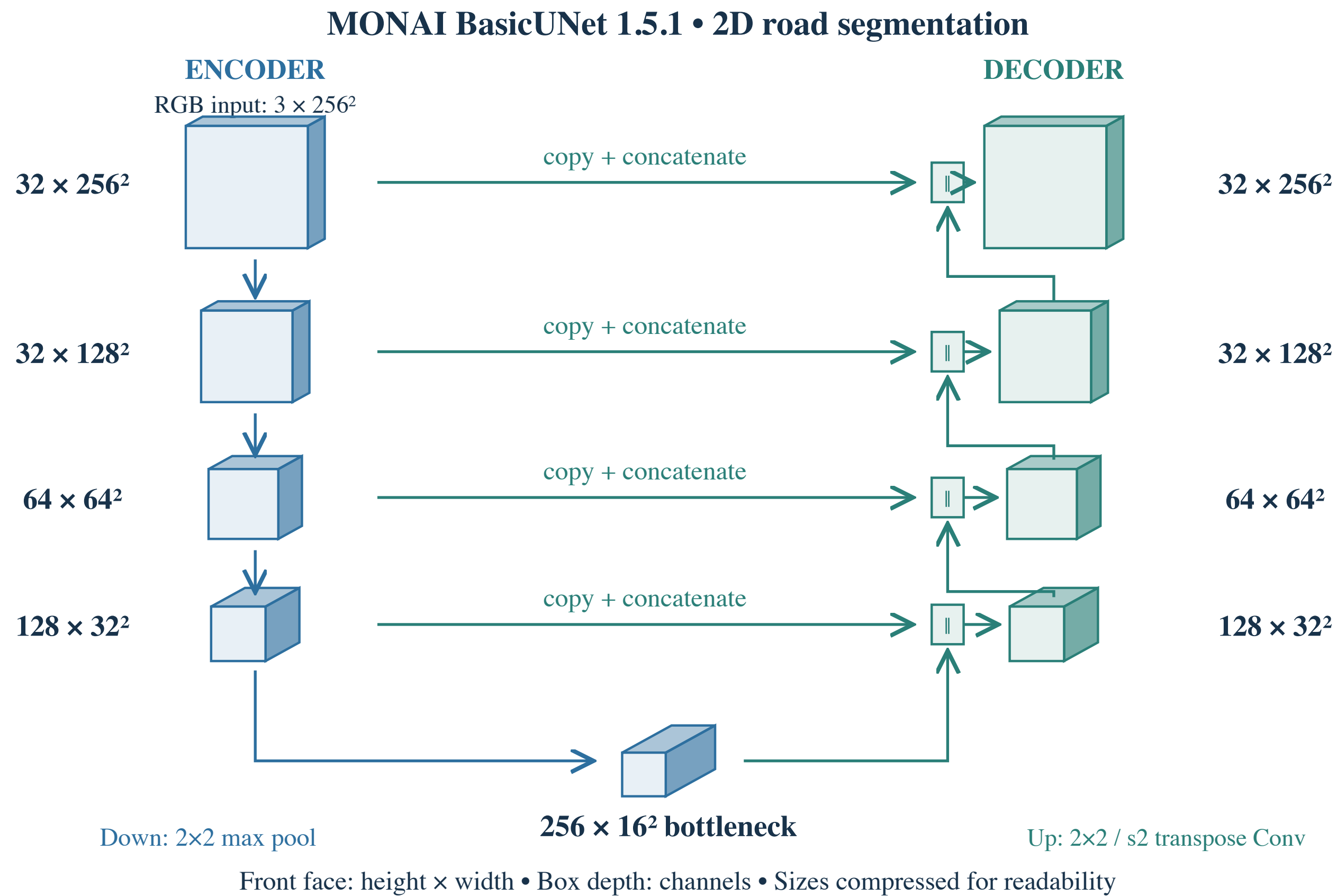
Fractional-stride intuition explains **learned spatial expansion**



The overlap adds: $2 + 3 = 5$. The fine-grid values are learned combinations.

In 2D, `ConvTranspose2d(16, 8, 2, stride=2)` maps $[B, 16, 16, 16]$ to $[B, 8, 32, 32]$.

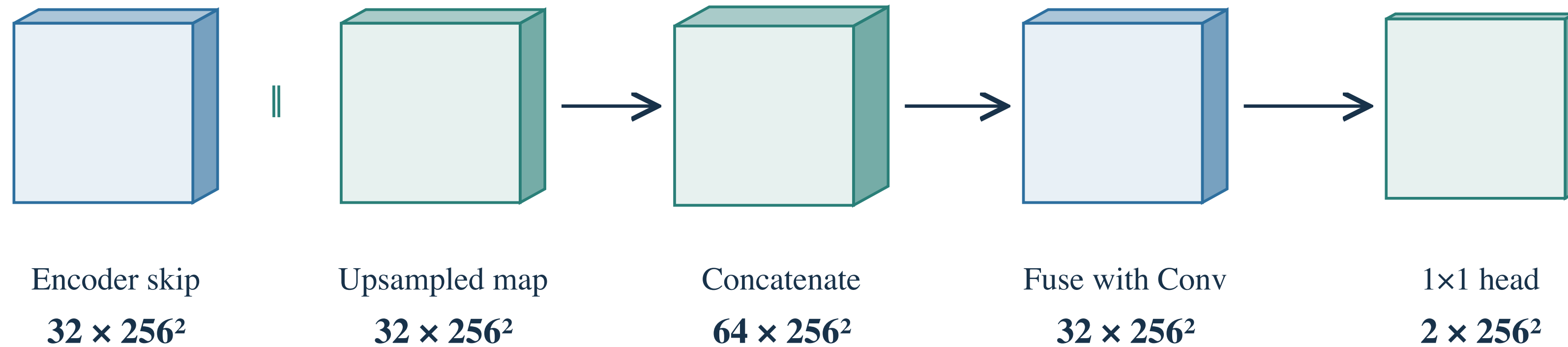
| U-Net combines coarse context with fine-resolution features



Original illustration of MONAI BasicUNet 1.5.1, 2D configuration; shapes are channels $\times$ spatial size.

| The U-Net head keeps **one score vector at every pixel**

Last decoder stage: preserve locations, combine channels



Concatenation doubles channels; the height and width stay fixed.

Front face: height $\times$ width • Box depth: channels • Sizes compressed for readability

| The task determines **what the architecture must preserve**

- **Classification:** aggregate spatial features into an image-level decision.
- **Segmentation:** combine broad context with location-specific detail.
- **Both:** architectural choices need evidence on the intended data and task.

We can now read complete architectures from familiar operations. **Knowing the pieces does not show that a model solves the application.**

Sequence Prediction and Representation

David I. Inouye

Dynamic Content · Last Updated: October 23, 2026

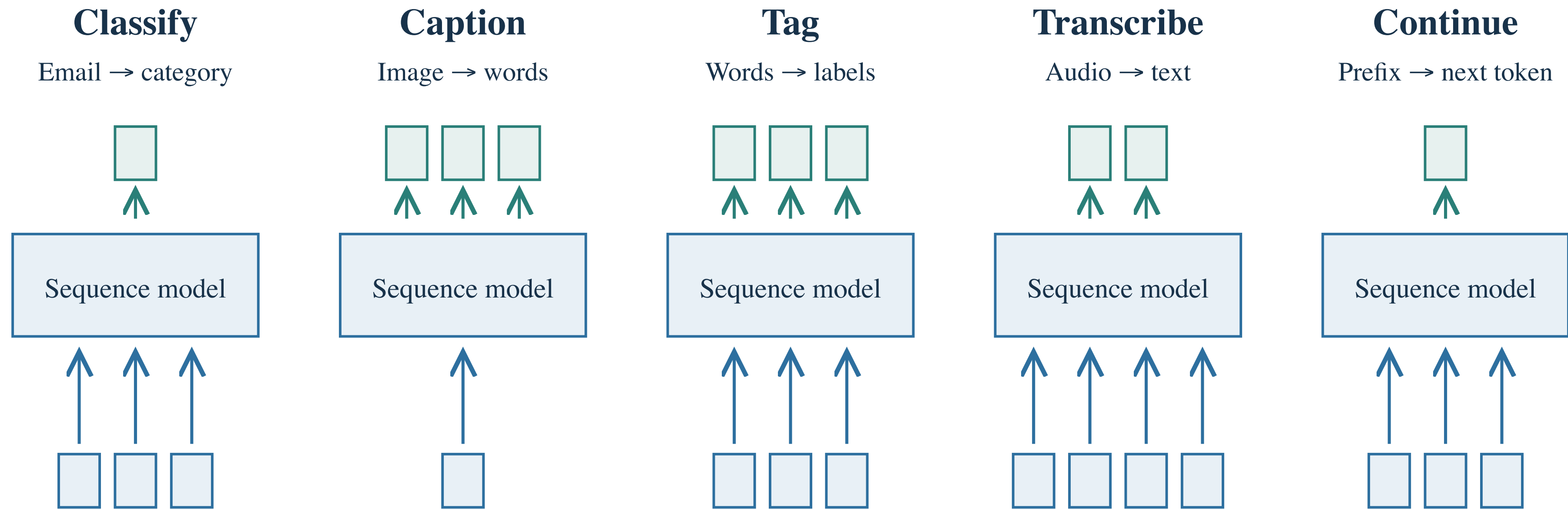
| A completion assistant needs a task, examples and a way to use context

Arc 6: Sequence-model architectures

1. **Define the task:** what is given, and what should come next?
2. **Build training examples:** tokens, learned features and shifted targets.
3. **Choose the context model:** fixed window, recurrent state, then attention.

We will follow one email-completion example through each choice.

| Sequence tasks differ in **what goes in and what comes out**



Inputs below → model → outputs above. Each small box is one item.

Our focus: next-token completion. Repeating it generates a continuation.

Original illustration informed by Karpathy's sequence-task examples.

| Email autocomplete suggests **the next few words while you write**

Typed prefix: “please send the”

Possible continuations: “report” · “slides” · “agenda”

- The writer supplies the message and accepts or ignores a suggestion.
- We predict the next token, then repeat to suggest a short continuation.
- Suggestions should fit the prefix and arrive quickly.

First: turn text into choices the model can predict.

Real application: [Gmail Smart Compose \(Chen et al., 2019\)](#). The email examples here are synthetic.

| Token IDs identify choices **without ordering their meanings**

Synthetic vocabulary for the worked examples:

Token	<BOS>	please	send	the	report	<EOS>
ID	0	1	2	3	4	5

`please send the report` becomes the ID sequence (1, 2, 3, 4).

- IDs select categories; subtracting two IDs does not measure similarity.
- A token can be a word, a word piece, punctuation, or a special marker.

Next: turn each ID into learned features.

| An embedding table learns **features** for each token

Synthetic table $E \in \mathbb{R}^{6 \times 2}$:

Token	Feature 1	Feature 2
<BOS>	0	0
please	1	0
send	0	1
the	1	1
report	1	-1
<EOS>	-1	0

For token ID $w_t = 2$ (**send**):

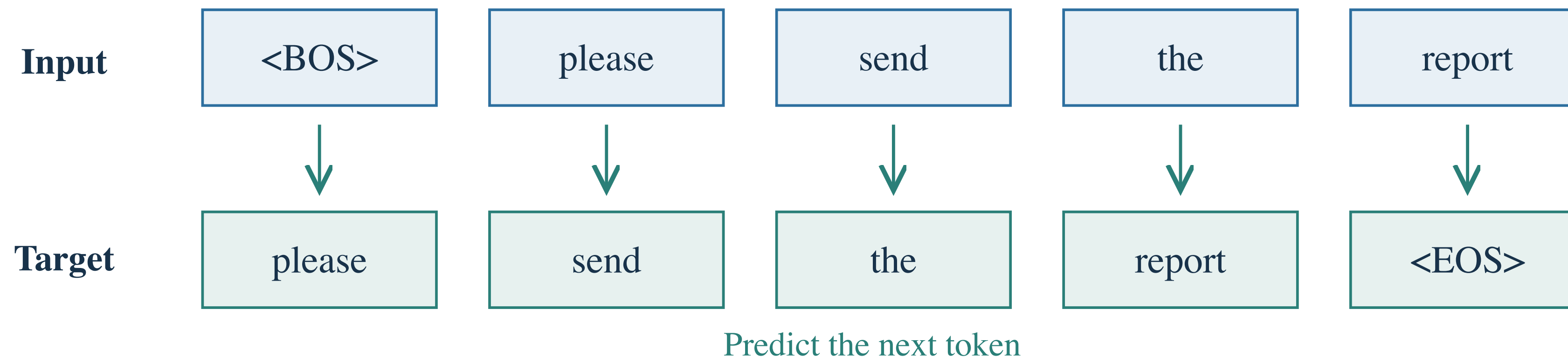
$$\mathbf{e}_t = E^T \mathbf{o}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$\mathbf{o}_2 = [0, 0, 1, 0, 0, 0]^T$ selects row 2.

- Lookup selects the same learned row wherever a token occurs.
- Training updates that row through the prediction loss.

Next: specify the target that teaches these features.

| Each input position predicts **the token immediately after it**



$$p_t = \text{softmax}(z_t), \quad \ell_t = -\log p_{t,w_{t+1}}$$

Allowed context for z_t : tokens through w_t only.

Putting it together: one next-token prediction at each position.