

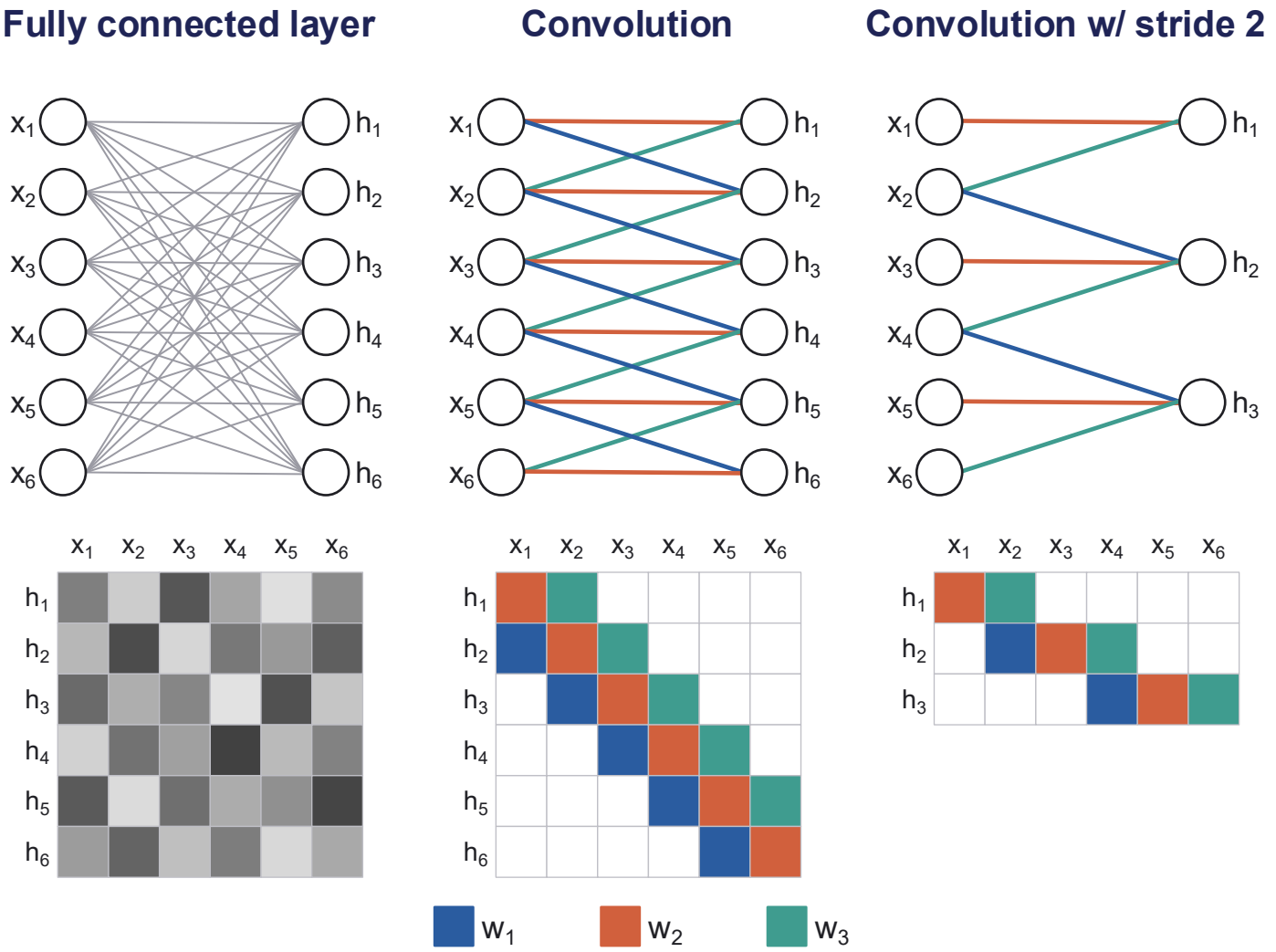
ML FOR SCIENCE · LECTURE 6

CNN and U-Net Architectures for Spatial Fields

Aparna Chandramowlishwaran · EECS 195/221 · Fall 2026

Convolution as a fully connected layer with shared weights

- **Fully connected layer**: every input connects to every output, one weight per pair (a, b)
- **Convolution**, kernel size 3: each output uses its 3 neighbors (**locality**) with the same 3 weights for every output (**weight sharing**) (c, d)
  - The weight matrix is a band, constant along each diagonal
- **Convolution w/ stride 2** keeps every other output: every other row of the matrix (e, f)
- A matrix that is constant along its diagonals commutes with shifts



x: inputs, h: outputs, zero padding at the ends. Matrix: row i holds the weights of output h_i.

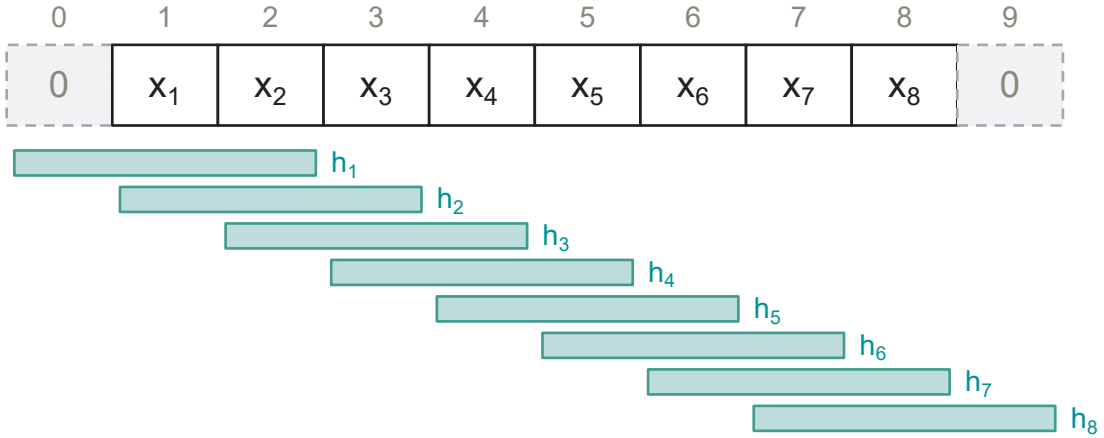
Output size of a convolution

- **Pad** the N input cells with p zeros on each side: N + 2p cells
- Each kernel placement gives one output. Its first cell can be at positions 0 to N + 2p − k
- With **stride** s, only every s-th position:

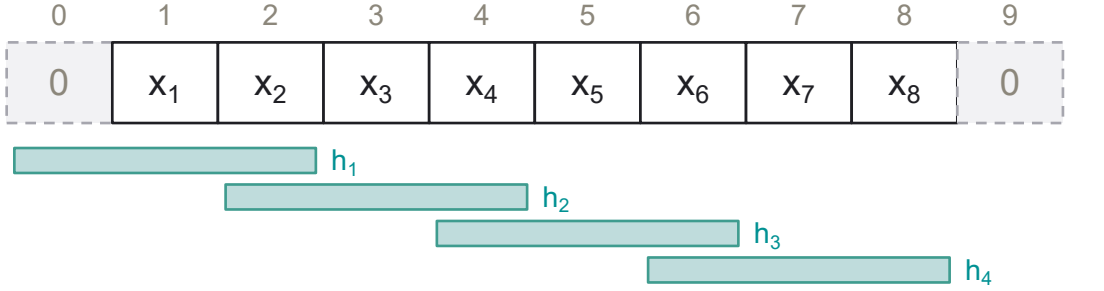
$$N_{\text{out}} = \left\lfloor \frac{N + 2p - k}{s} \right\rfloor + 1$$

- **Example**: N = 8, k = 3, p = 1
  - s = 1: positions 0 to 7, 8 outputs
  - s = 2: positions 0, 2, 4, 6: ⌊7 / 2⌋ + 1 = 4 outputs
  - SmallCNN: N = 48, k = 3, p = 1, s = 1: 48 outputs

Stride 1: 8 outputs, the same size as the input



Stride 2: 4 outputs, half the size



Padding p = (k − 1)/2 keeps the grid size

N: input cells per side, k: kernel size, p: padding, s: stride. Gray numbers: position of the kernel's first cell in the padded row. In 2D, the same count holds along each axis

Parameter count of a convolution layer

- Each output channel has one k x k kernel per input channel; it sums their results and adds a bias:

$$\text{parameters} = k^2 \, C_{\text{in}} \, C_{\text{out}} + C_{\text{out}}$$

- Example, layer 1 of SmallCNN: 32 output channels, each with 3 kernels of 3 x 3 (one for θ, u, v) and a bias: $32 \cdot (27 + 1) = 896$
- One fully connected layer from a 48 x 48 field to a 48 x 48 field: $2{,}304^2 \approx$ **5.3 million** weights
- Parameters do not depend on the grid size N

SmallCNN on BubbleML

layer	channels	parameters
1	3 → 32	9·3·32 + 32 = 896
2	32 → 32	9·32·32 + 32 = 9248
3	32 → 32	9,248
4	32 → 1	9·32 + 1 = 289
total		**19681**

The number of parameters depends only on the kernel size and the channels

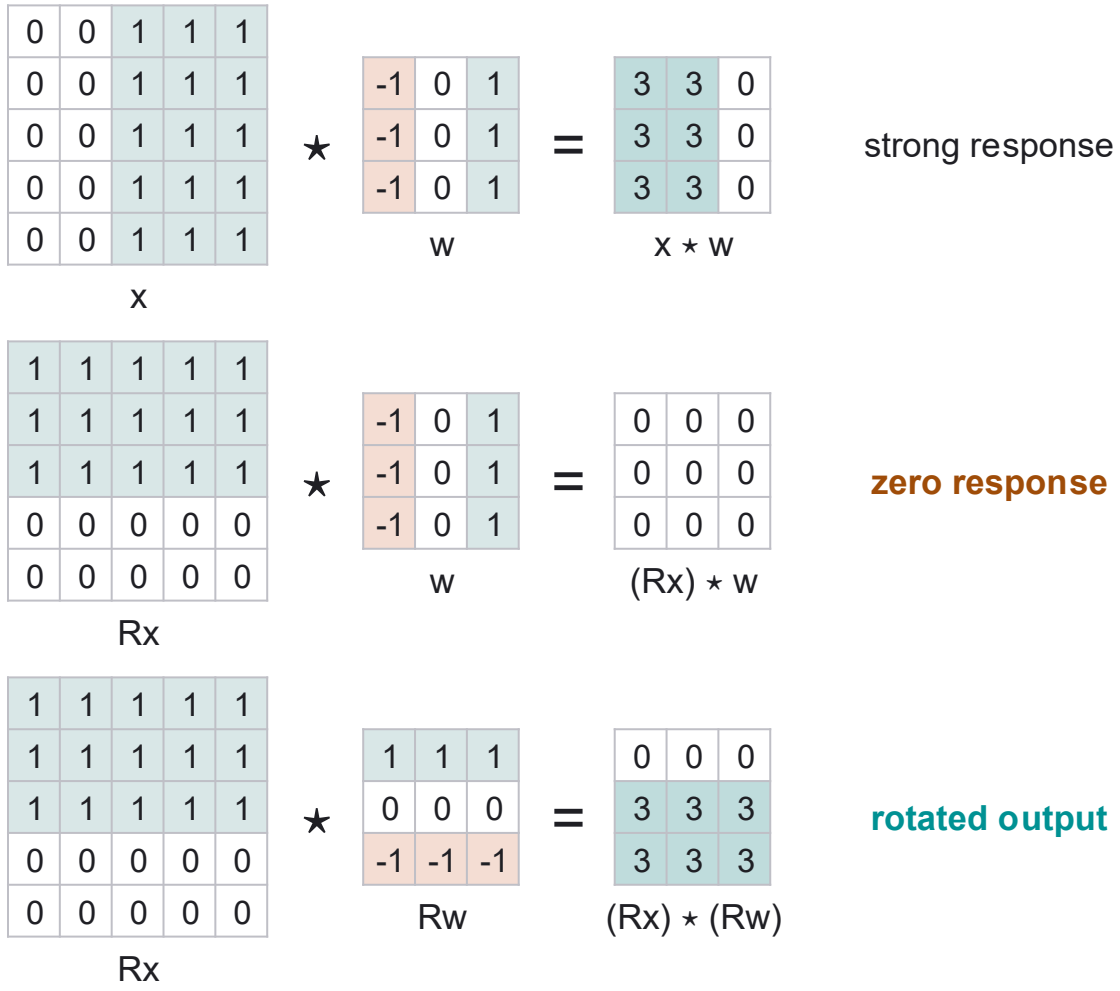
k: kernel size, C_{in}, C_{out}: input and output channels. SmallCNN on BubbleML: inputs θ, u, v (3 channels), 32 hidden channels, output: change in θ

Convolution does not commute with rotation

- **Top**: a vertical-edge kernel w on a vertical edge outputs a strong response
- **Middle**: rotate the input. The edge is horizontal; the same kernel outputs zero everywhere
- **Bottom**: rotate the kernel too. The output is the top output, rotated:

$$(R\,x) \star (R\,w) \;=\; R\,(x \star w)$$

- A standard CNN applies the same kernels to a rotated input.



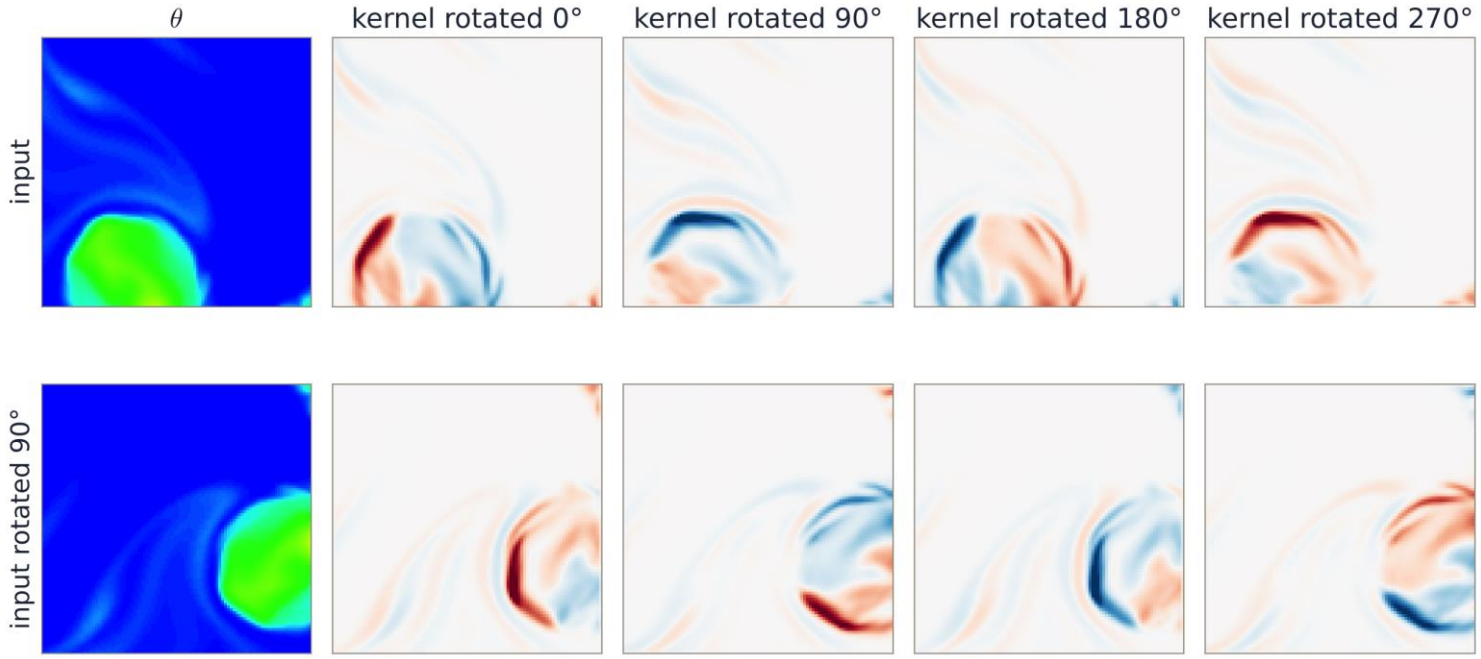
⋆: convolution as in PyTorch, no padding. R: rotation by 90° counter-clockwise. Kernel w: each row (−1, 0, 1), responds to vertical edges

Group convolution

- Use all 4 rotations of the kernel, $R^m w$ for m = 0, 1, 2, 3, as 4 output channels: the **lifting layer**
- Rotating the input rotates each response map and moves it to the next channel:

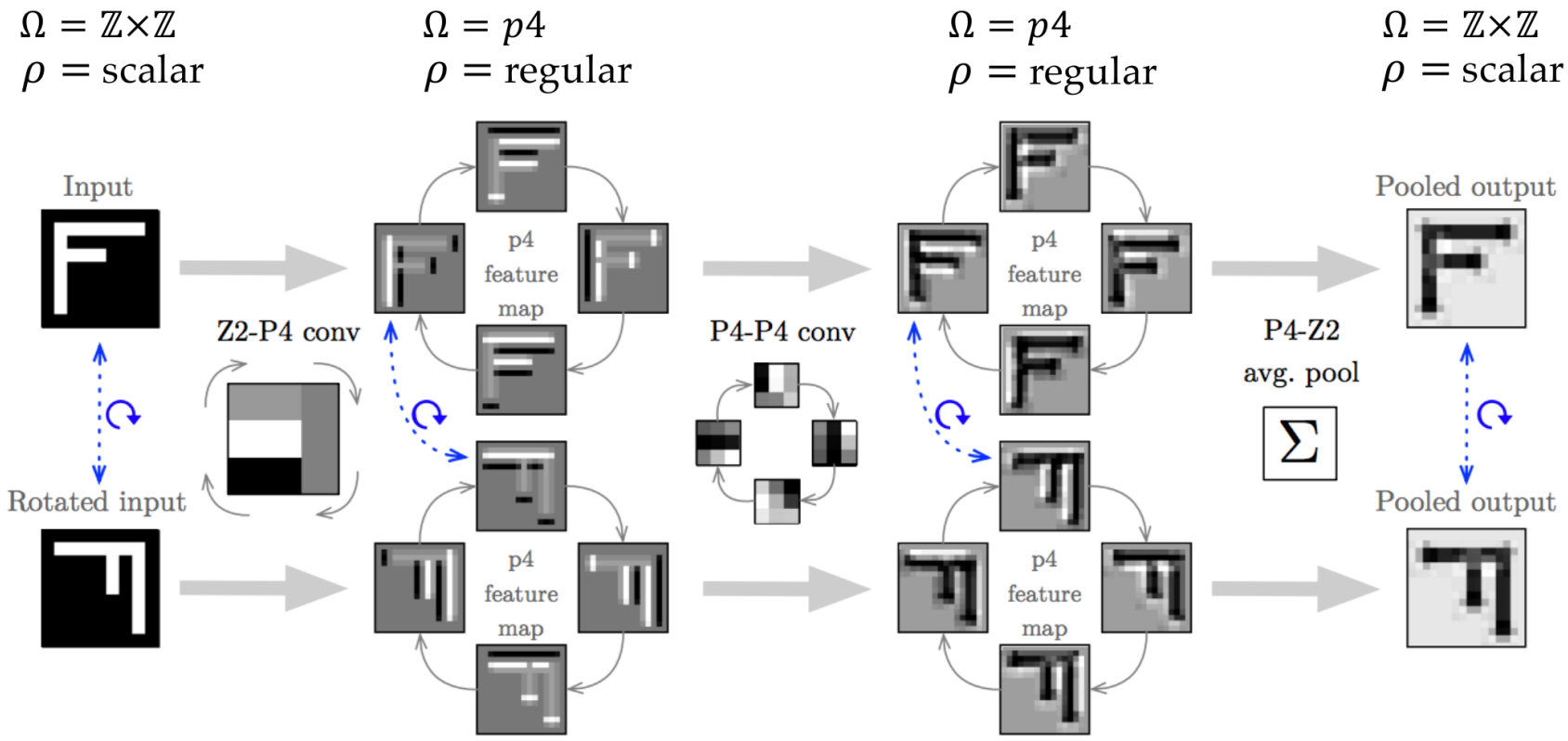
$$(R\,x) \star (R^m w) \;=\; R\left(x \star R^{m-1} w\right)$$

- Figure: the 0° channel (top) reappears, rotated, as the 90° channel (bottom). Edge direction is kept
- Lecture 5 alternative: a kernel equal to its own rotations has 3 weights and **responds the same to every edge direction**
- **Cost**: 4x the channels for the same weights



Recall: Group convolutions

- Apply one kernel in all 4 rotations to get 4 **orientation channels** (a lifting layer). Rotating the input rotates each channel and moves it to the next one, so direction information is kept
- Later layers share weights across rotations, so the network is D4-equivariant. Cost: 4 times the activations for the same number of weights



Veeling et al. 2018

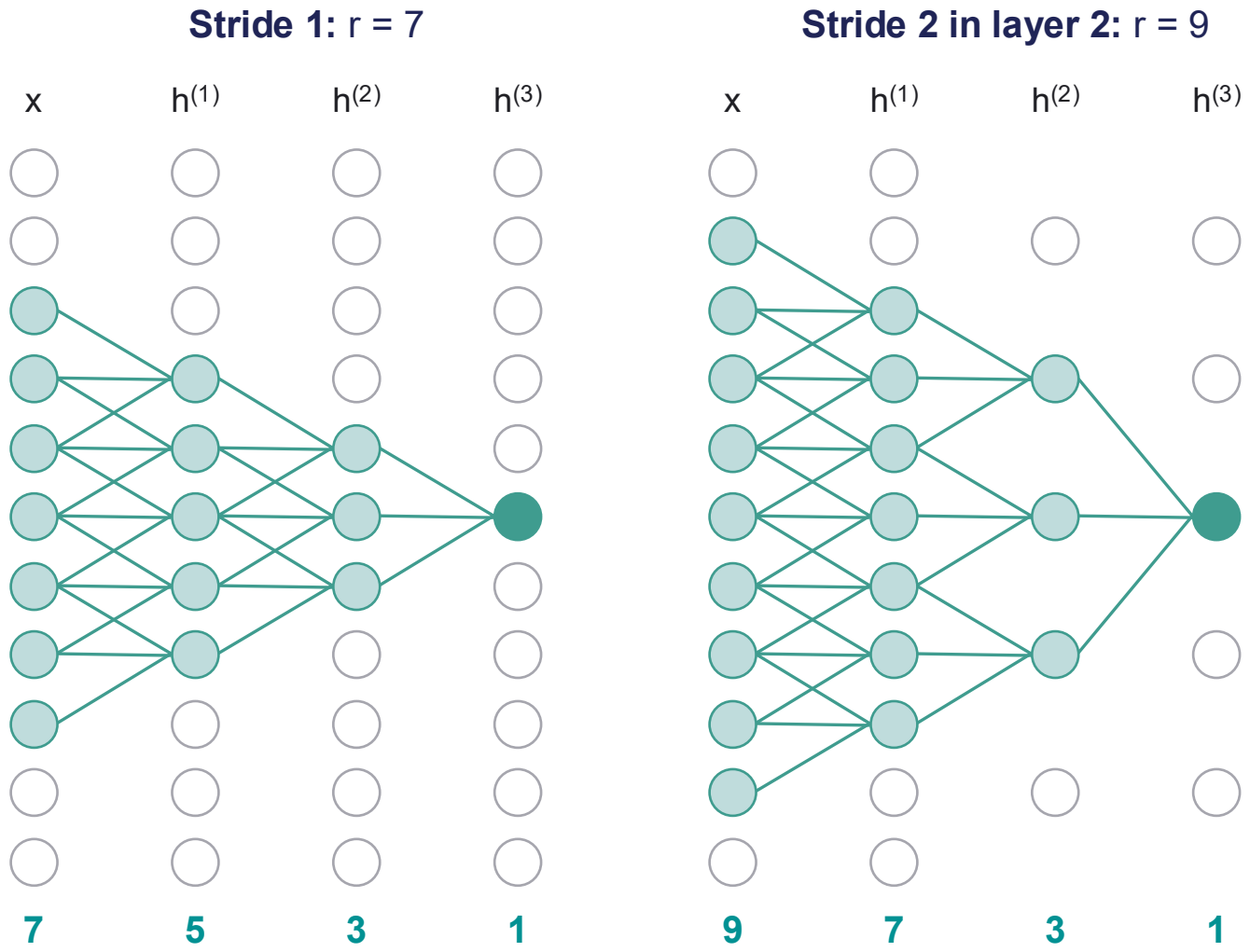
Figure: Veeling et al., MICCAI 2018. Cohen & Welling, ICML 2016 · steerable CNNs: Weiler & Cesa, NeurIPS 2019

Receptive field

- The **receptive field** of a neuron: the input cells that can change its value

- Compute the output at $h^{(3)}$ (**left**): 3 neurons of $h^{(2)}$, which use 5 of $h^{(1)}$, which use 7 input cells: r = 7
- Each layer with kernel size 3 adds one cell on each side
- **Right**: stride 2 in the second layer leaves a gap between $h^{(2)}$ neurons. The third layer then adds **2 cells** on each side: r = 9

- In **2D**, the receptive field is an r x r square



x: input cells, $h^{(\ell)}$: neurons of layer ℓ, kernel size 3 in every layer. Teal numbers: how many neurons in each layer it depends on

Computing the receptive field

- Starting from the input layer, track two quantities:
  - **r**: receptive field, in input cells
  - **j**: spacing between neighboring outputs, in input cells (the product of the strides so far)

$$r_\ell = r_{\ell-1} + (k_\ell - 1)\, j_{\ell-1}, \qquad j_\ell = j_{\ell-1}\, s_\ell, \qquad r_0 = j_0 = 1$$

- Right network from the previous slide:

layer	k	s	j before	r
$h^{(1)}$	3	1	1	1 + 2·1 = 3
$h^{(2)}$	3	2	1	3 + 2·1 = 5
$h^{(3)}$	3	1	**2**	**5 + 2·2 = 9**

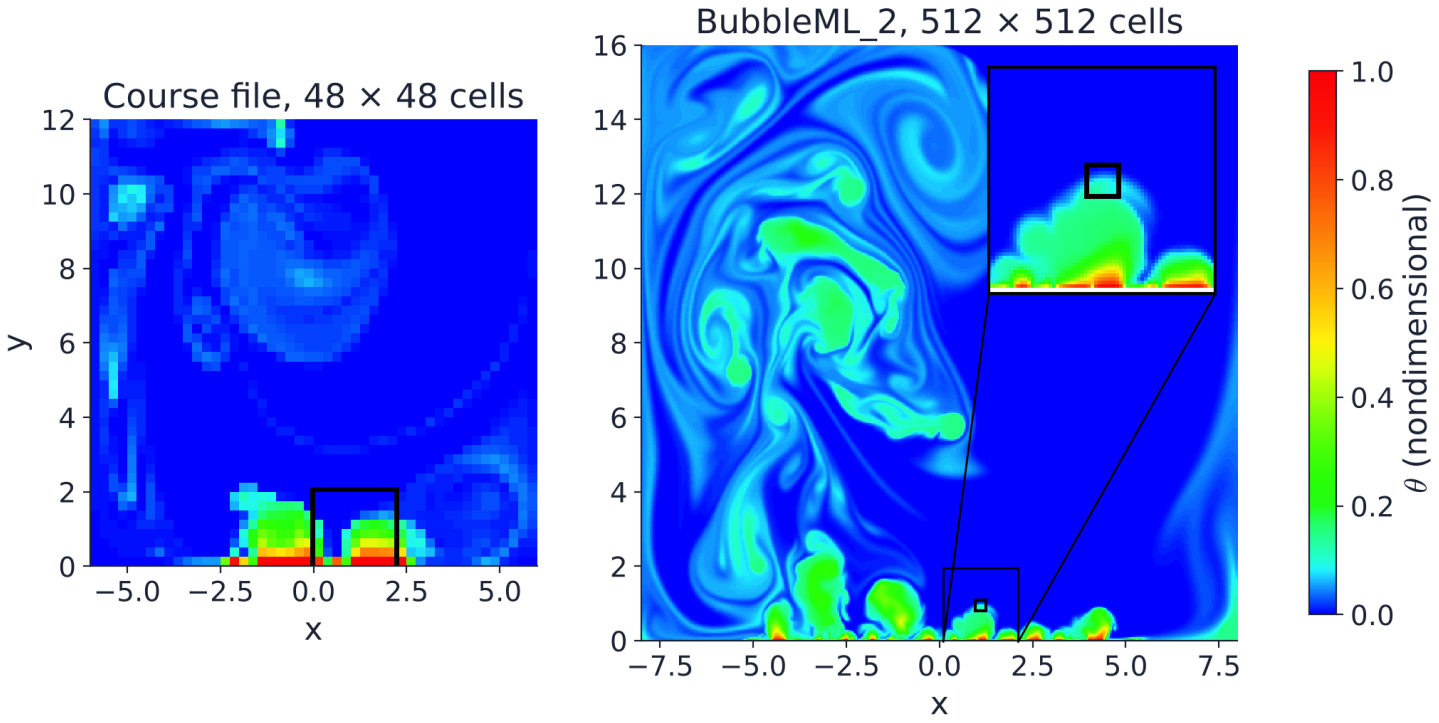
- SmallCNN: four 3 x 3 layers, stride 1, so j = 1:
  - r = 3, 5, 7, 9
- A stride multiplies the growth of every layer after it

Each layer adds (k − 1) · j cells to the receptive field

Layer ℓ: kernel size k_ℓ, stride s_ℓ.

Receptive field at two resolutions

- SmallCNN: four 3 x 3 layers, stride 1:
  r = 1 + 2 + 2 + 2 + 2 = 9
  - j stays 1, so each layer adds k − 1 = 2 cells
  - Each output cell depends on a **9 x 9** block of input cells
- Lecture 1 notebook, cell size 0.25: the block spans 2.25 length units, about the size of a bubble
- Full-resolution BubbleML_2, cell size 1/32: the same 9 cells span 0.28 units, a small part of one bubble interface
- At high resolution, the model needs information from **many more cells** to predict the next frame



Left: Lecture 1 notebook (48 x 48, cell size 0.25). Right: BubbleML_2 subcooled pool boiling, FC-72, Twall = 106 ° C (512 x 512, cell size 1/32). Black square: 9 x 9 cells.

How far information travels in one time step

- Lecture 3: in one time step Δt, information travels
  - by **advection** at speed U: a distance of about $U\,\Delta t$
  - by **diffusion** with diffusivity α: a few $\sqrt{(\alpha \Delta t)}$
  - through the **pressure**: the **entire domain**, since pressure solves an elliptic (Poisson) equation at every step
- Model's time step is the frame interval. Its receptive field reaches $(r - 1)/2$ cells on each side

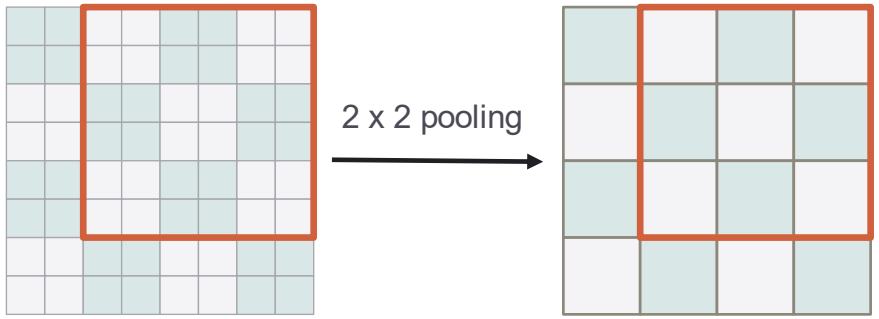
- Example: BubbleML
  - **Near bubbles**, velocity reaches about 10. In one frame interval, fluid moves up to 2 length units
    - 64 cells on the 512 x 512 grid; 16 cells on the 128 x 128 grid
  - **Away from bubbles**, speeds are around 1: under 2 cells per frame on the 128 x 128 grid
  - SmallCNN reaches 4 cells in each direction from the output cell

Receptive field needed (in cells) = distance information travels per frame ÷ cell size

Velocity: BubbleML_2, Twall = 106 ° C, frames 400 to 2000, averaged to 256 x 256. Our models use frames every 0.2 time units on a 128 x 128 grid (cell size 0.125)

Receptive field with pooling

- **Pooling** replaces each 2 x 2 block with its *average* or its *maximum*. No weights; the grid halves
- A 3 x 3 kernel on the pooled grid covers 6 x 6 input cells (orange). In the rule: **k = 2, s = 2**, so j doubles



- Two 3 x 3 convolutions per level, pooling between levels (first half of U-Net): 8 convolutions and 3 poolings reach **68 cells**; same 8 convolutions with stride 1 reach 1 + 8 · 2 = **17 cells**
- On the 128 x 128 grid, it reaches 33 cells on each side, beyond the 16 cells the fastest fluid moves per frame

layer	k	s	j before	r
conv	3	1	1	3
conv	3	1	1	5
pool	2	2	1	6
conv	3	1	2	10
conv	3	1	2	14
pool	2	2	2	16
conv	3	1	4	24
conv	3	1	4	32
pool	2	2	4	36
conv	3	1	8	52
conv	3	1	8	**68**

$$r_\ell = r_{\ell-1} + (k_\ell - 1)\, j_{\ell-1}, \qquad j_\ell = j_{\ell-1}\, s_\ell, \qquad r_0 = j_0 = 1$$

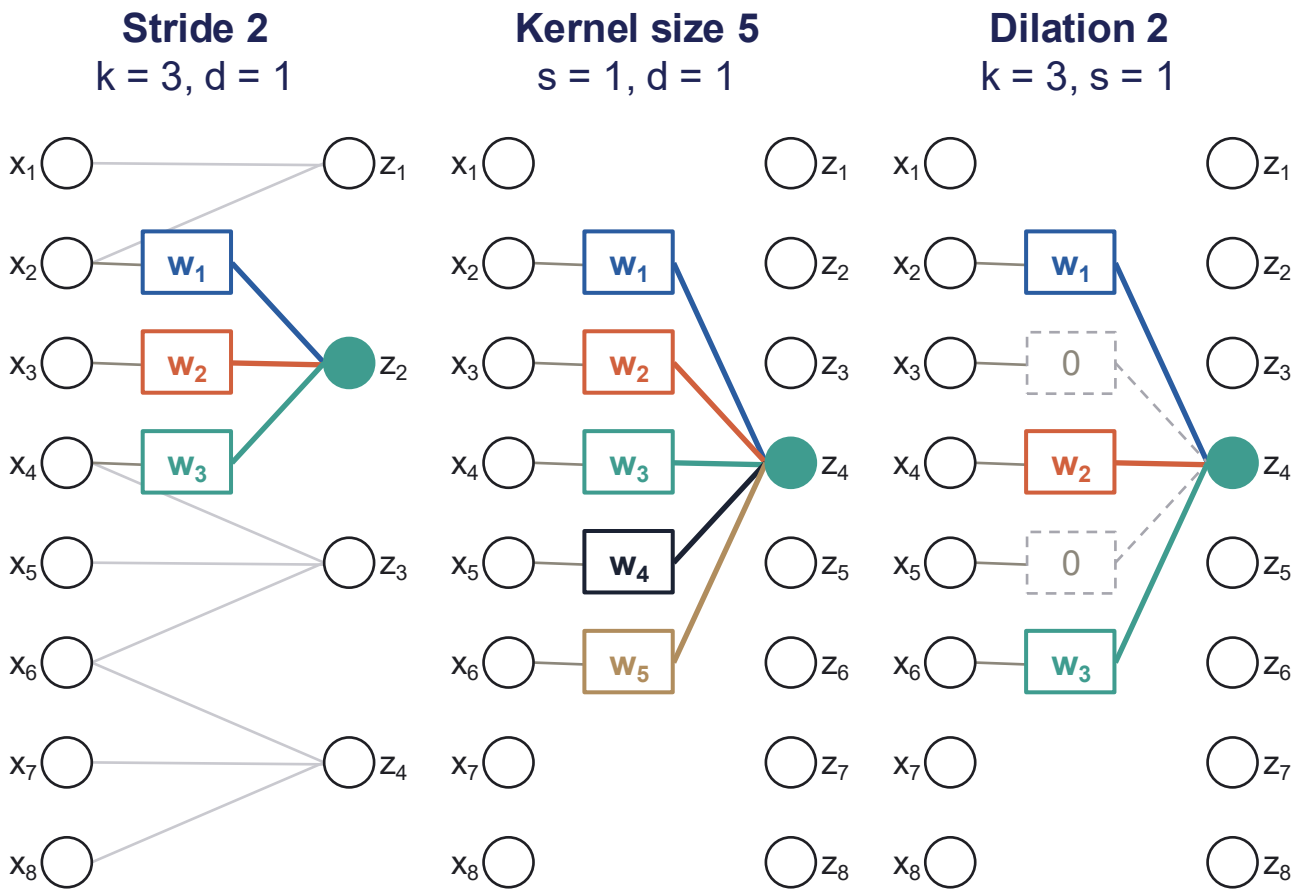
Table: notebook U-Net encoder, 3 levels of two 3 x 3 convolutions, 2 x 2 average pooling between levels, two 3 x 3 convolutions at the bottleneck

Kernel size, stride, and dilation

- **Stride** s: the kernel moves s cells between outputs, so there are fewer outputs
- **Kernel size** k: more inputs per output, and more weights
- **Dilation** d: the k weights are d cells apart. The kernel spans k_{eff} cells:

$$k_{\text{eff}} = d\,(k-1) + 1$$

  - with only k weights. Use k_{eff} in place of k in the receptive field rule
- Example: four 3 x 3 layers
  - with d = 1, 2, 4, 8: r = 1 + 2 (1 + 2 + 4 + 8) = **31**
  - with the weights of SmallCNN: r = **9**
- Dilated kernels skip cells and can leave gaps in the network (**gridding**; Yu & Koltun, ICLR 2016)



Deep networks without skip connections

- More layers result in **larger receptive field**. Networks without skip connections, where each layer feeds only the next, get harder to train as they get deeper
- 56 layers: **higher training error** than 20 layers. Overfitting would lower the training error, so the problem is **optimization**
  - The 56-layer network could copy the 20-layer network and set the extra layers to the identity; gradient descent does not find that solution
- One difficulty is **vanishing gradients**: the gradient is a product of one factor per layer (chain rule). If each factor is about 0.5:

$$\frac{\partial a^{(20)}}{\partial a^{(0)}} = \prod_{l=1}^{20} \frac{\partial a^{(l)}}{\partial a^{(l-1)}} \approx 0.5^{20} \approx 10^{-6}$$

- He et al. used batch normalization, so their gradients did not vanish, and the 56-layer network still trained worse: the **degradation problem**. Skip connections fix both

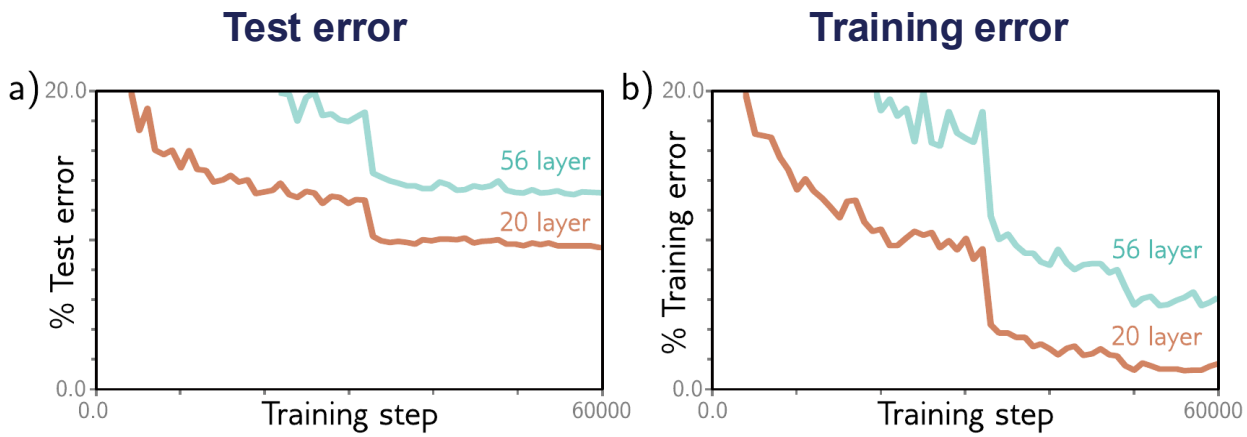


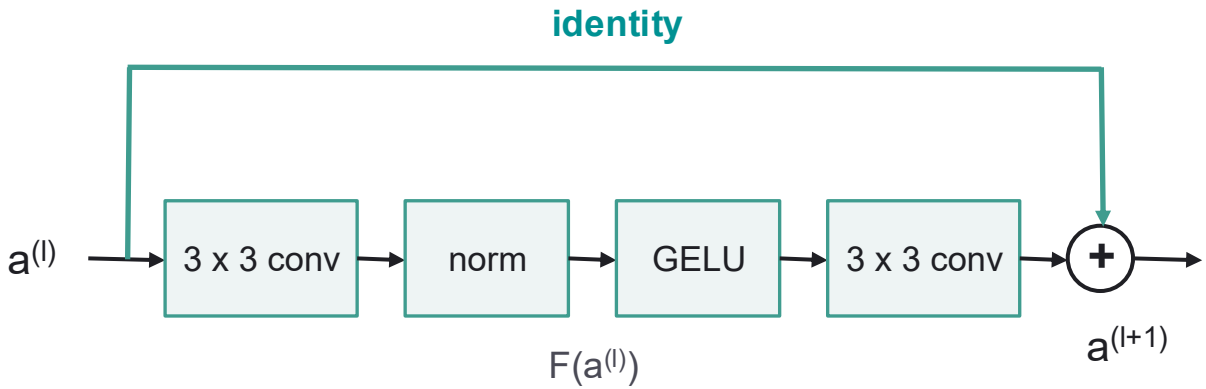
Figure: S. Prince, Understanding Deep Learning (MIT Press), 2023, after He et al., CVPR 2016. CIFAR-10 image classification, CNNs without skip connections

Residual connections

- A **residual block** adds its input to its output:

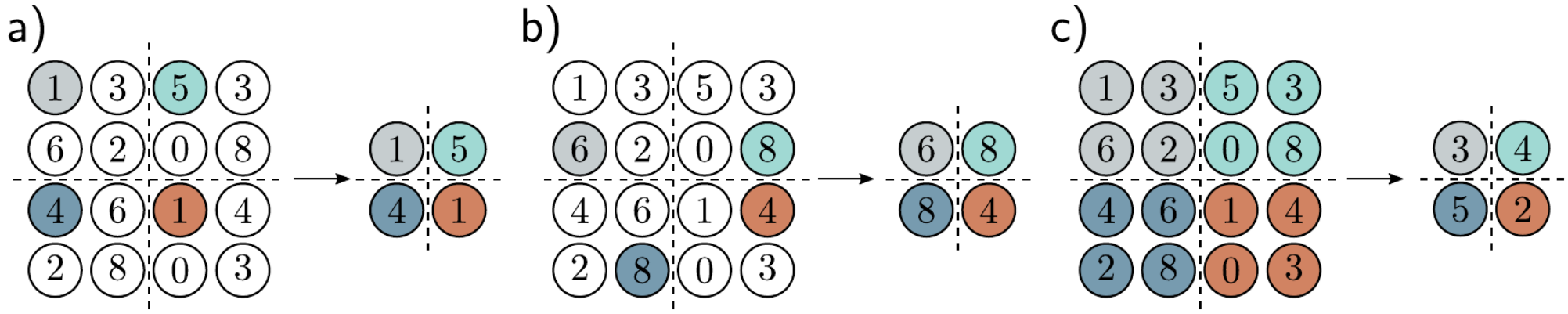
$$a^{(l+1)} = a^{(l)} + F\big(a^{(l)}\big), \qquad \frac{\partial a^{(l+1)}}{\partial a^{(l)}} = I + \frac{\partial F}{\partial a^{(l)}}$$

- The **identity I** carries the gradient through every block, irrespective of ∂F / ∂a
- Each block **learns a correction F**; F = 0 gives the identity
- SmallCNN predicts the change: new field = current field + predicted change, the **forward Euler** from Lecture 3
- **norm**: a **normalization** layer rescales activations to mean 0 and variance 1, then applies a learned scale and shift.
- **Batch normalization** mixes samples of different physical states and behaves differently at test time. PDE models often use **group normalization**, within one sample, so training and testing match



a(l): activations after block l; F: the block's layers (e.g. conv, norm, GELU, conv). He et al., CVPR 2016. Group normalization: Wu & He, ECCV 2018

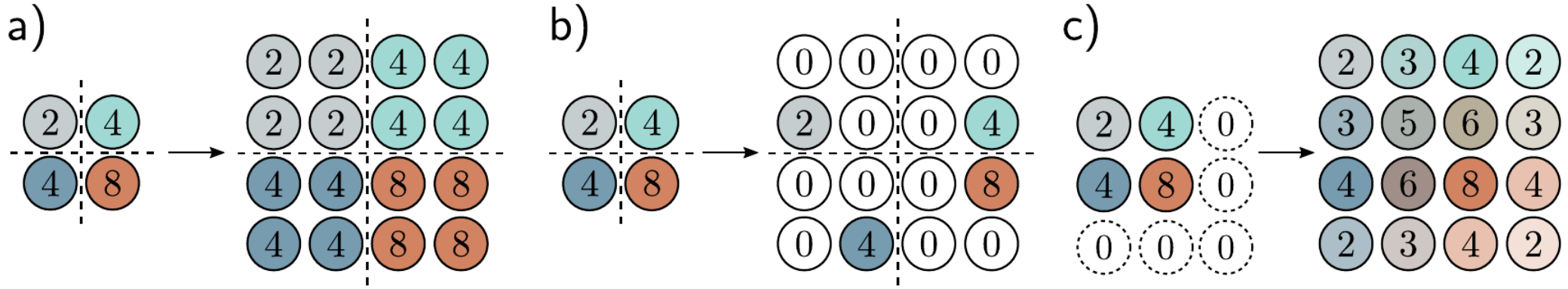
Downsampling



- Downsampling halves the grid: each 2 x 2 block of cells becomes 1 cell. (a) keep every other cell (stride 2), (b) max pooling, (c) mean (average) pooling
- After one downsampling, a 3 x 3 kernel covers 6 x 6 fine cells. After n downsamplings, **$3 \cdot 2^n$**
- Lecture 4: average before downsampling to limit aliasing. For physical fields, use **average pooling**
- Lecture 5: downsampling **breaks shift equivariance**; shifts by 2^n cells stay exact

Figure: S. Prince, Understanding Deep Learning (MIT Press, 2023). Each output cell combines a 2 x 2 block of input cells

Upsampling and transposed convolution



- Upsampling doubles the grid: (a) duplicate, (b) max-upsampling, (c) bilinear interpolation
- A **transposed convolution** learns it: each input cell adds a copy of the kernel, scaled by its value; with stride 2, the copies are 2 cells apart
- A constant input gives an alternating output: **checkerboard artifacts**
- Common fix: bilinear upsampling, then a 3 x 3 convolution

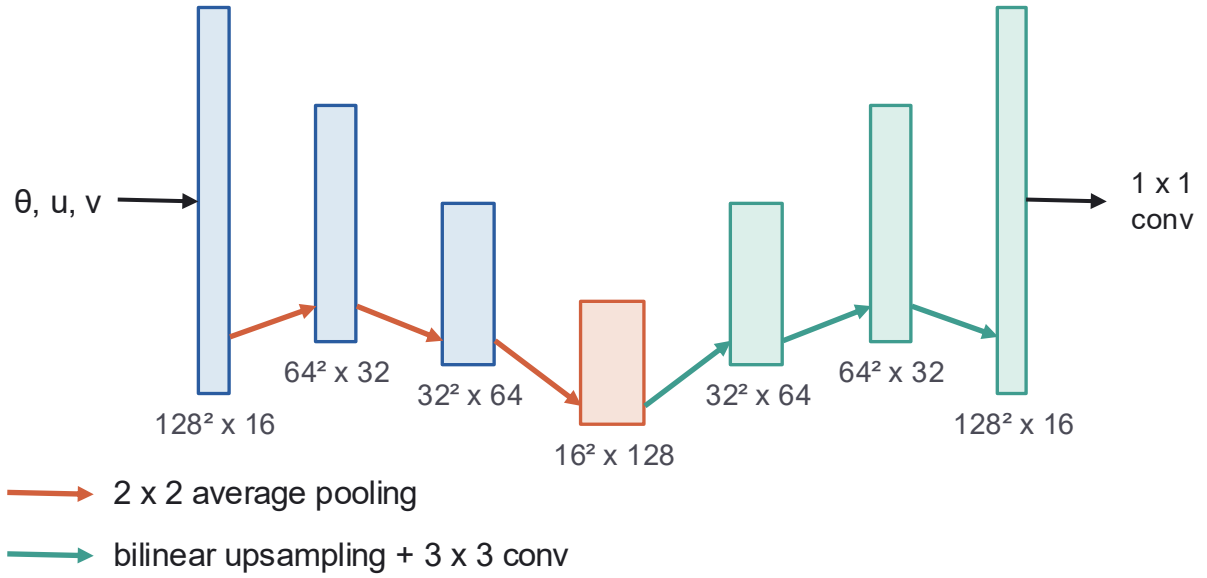
input (1, 1, 1), kernel (1, 1, 1), stride 2

	out 1	2	3	4	5	6	7
copy from x_1	1	1	1	0	0	0	0
copy from x_2	0	0	1	1	1	0	0
copy from x_3	0	0	0	0	1	1	1
sum	**1**	**1**	**2**	**1**	**2**	**1**	**1**

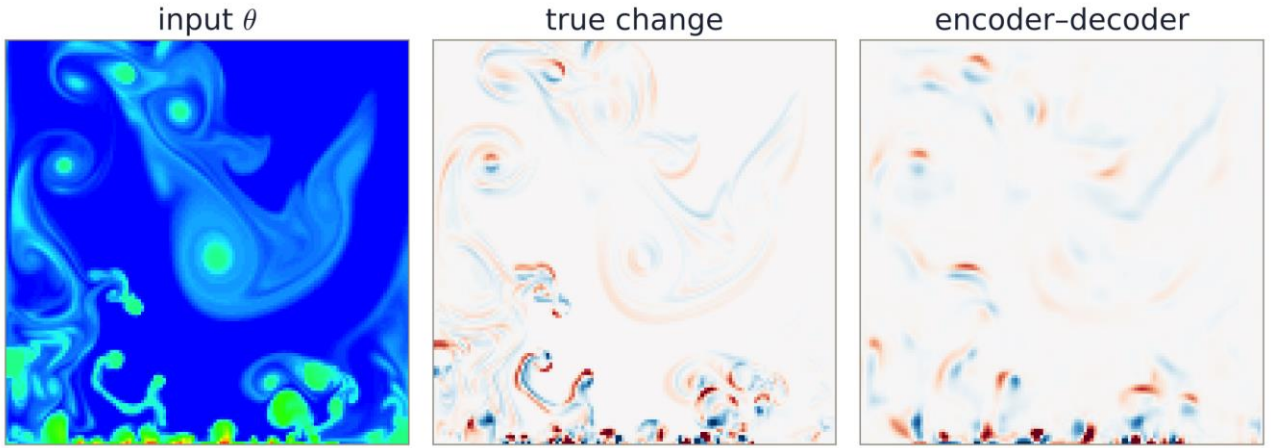
Figure: S. Prince, Understanding Deep Learning (MIT Press, 2023). Checkerboard artifacts: Odena, Dumoulin & Olah, Distill 2016

Encoder-decoder networks

- **Encoder**: convolutions and pooling. The grid shrinks, the channels grow, down to a 16 x 16 **bottleneck**
- **Decoder**: upsampling and convolutions back to 128 x 128
- Large receptive field (68 cells at the bottleneck) at low cost: most layers run on coarse grids



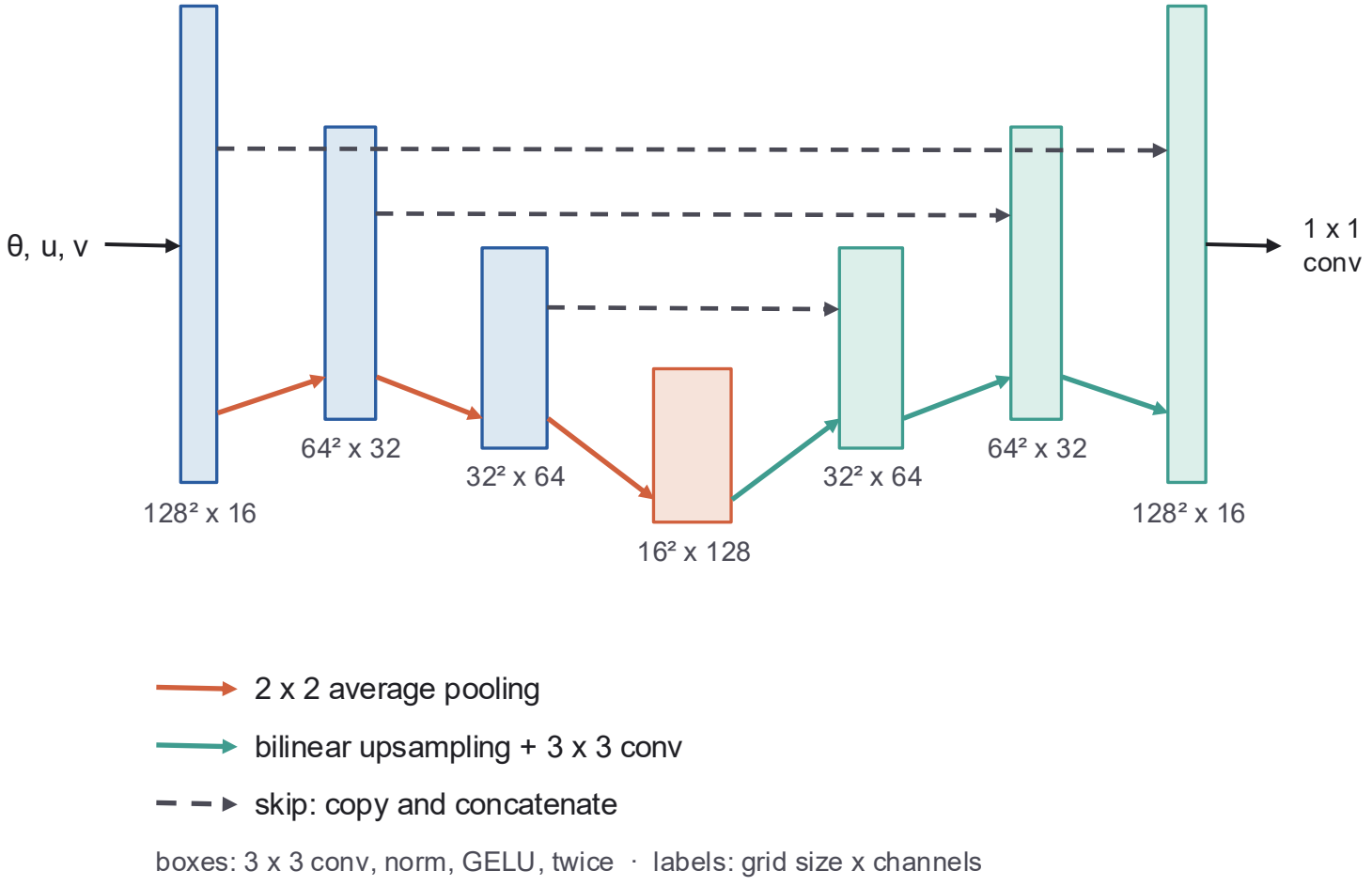
- Every value in the output passes through the 16 x 16 bottleneck. **Fine detail lost there cannot be recovered**



Encoder-decoder: the notebook U-Net without skip connections, 488,817 parameters. Trained on BubbleML_2 (Twall = 106 ° C)

U-Net

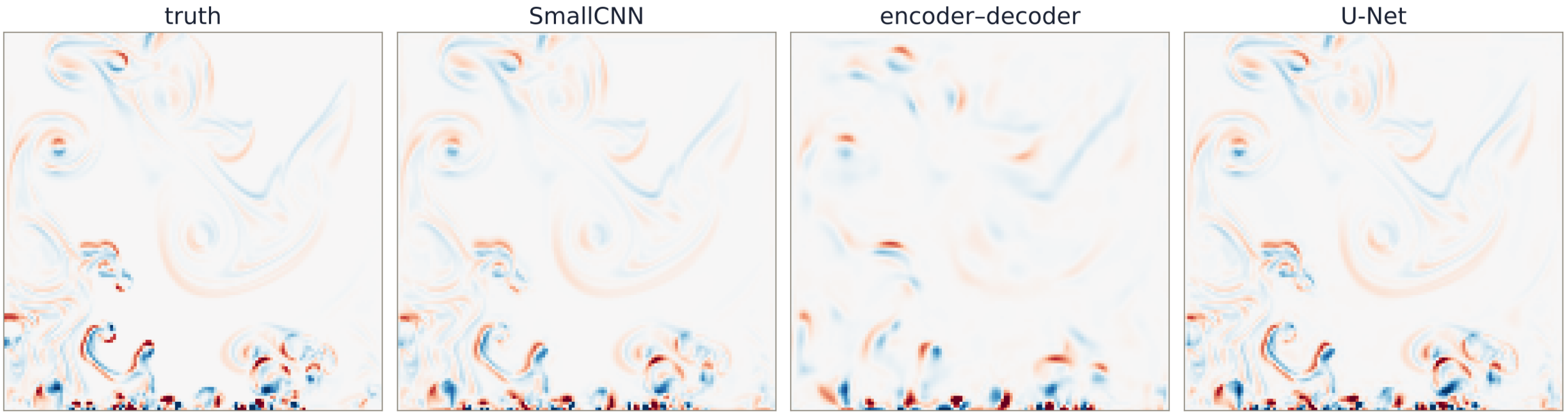
- An encoder-decoder with **skip connections**: each encoder level is copied to the decoder level of the same size and concatenated along the channels
- The decoder combines coarse information (large receptive field) with fine information (from the skips)
- Output: a 1 x 1 convolution, the same linear map at every cell, from 16 channels to the change in θ
- Skip connection here is a concatenation; by addition in a residual block



Notebook U-Net, 537,201 parameters. Ronneberger, Fischer & Brox, MICCAI 2015 (microscopy segmentation.

U-Net and encoder-decoder predictions on BubbleML

- Encoder-decoder: captures large-scale patterns but **blurred interfaces**
- U-Net: **sharp interfaces** at the correct locations
- SmallCNN: close to the U-Net, since most of the flow moves less than 4 cells per frame
- U-Net: much lower training loss, only slightly lower test error (600 training frames from one run)

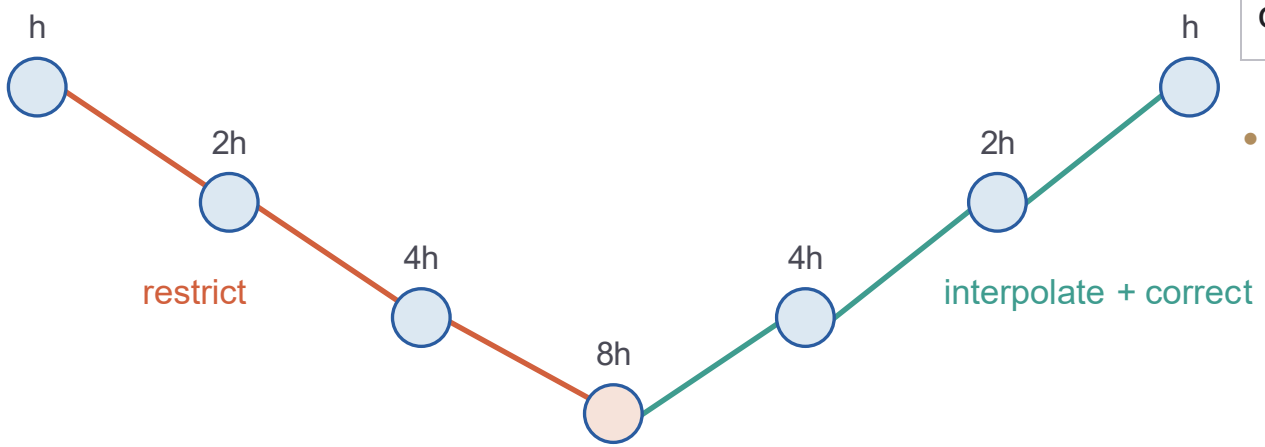


The skip connections restore the bubble interfaces lost in the bottleneck

Change in θ over one frame interval (0.2), one-step predictions on test frames $t \in [160, 184]$, each from the true previous frame.

U-Net and multigrid

- Multigrid solves elliptic equations, like the pressure Poisson equation
- Smoothing (e.g. Jacobi iterations) removes **short-wavelength** error fast and long-wavelength error slowly
- **V-cycle**: smooth, move the residual to a coarser grid, where long wavelengths look short, repeat; then interpolate the corrections back up



grid spacing h, 2h, … ; each circle: smoothing steps

multigrid	U-Net
smoothing steps (local stencil)	3 x 3 convolutions
restriction to a coarser grid	pooling
interpolation to a finer grid	upsampling
fine-grid solution, corrected	skip connection
operators from the PDE	**learned weights**

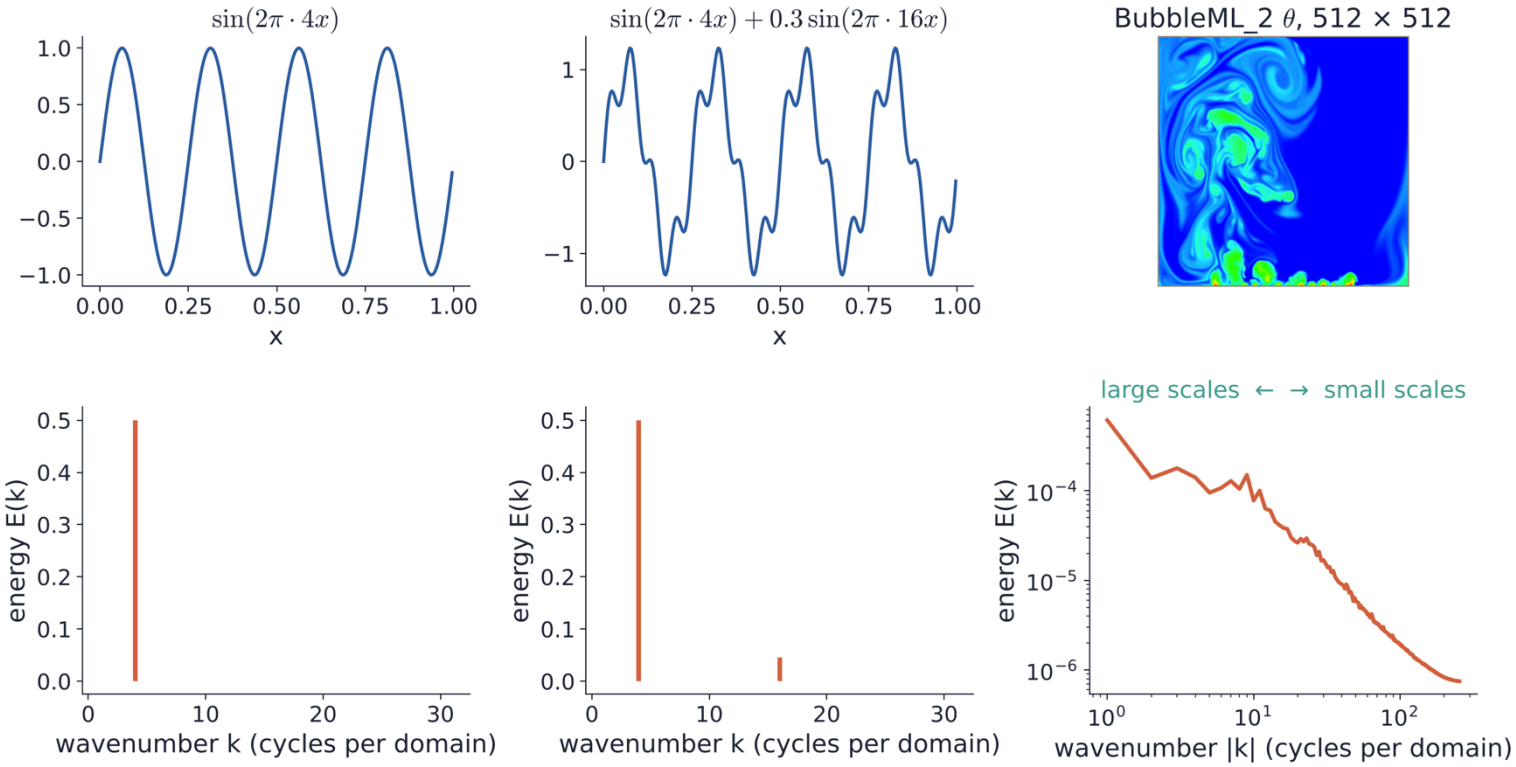
- A U-Net has the structure of a V-cycle, with learned operators

Brandt, Math. Comp. 1977; Briggs, Henson & McCormick, A Multigrid Tutorial (SIAM, 2000); He & Xu, MgNet, Sci. China Math. 2019

Energy spectrum

- The **energy spectrum** E(k): how much of a field is at wavenumber k (Fourier modes, Lecture 3)

$$u(x) = \sum_k \hat{u}_k \, e^{2\pi i k x}, \qquad E(k) = |\hat{u}_k|^2$$

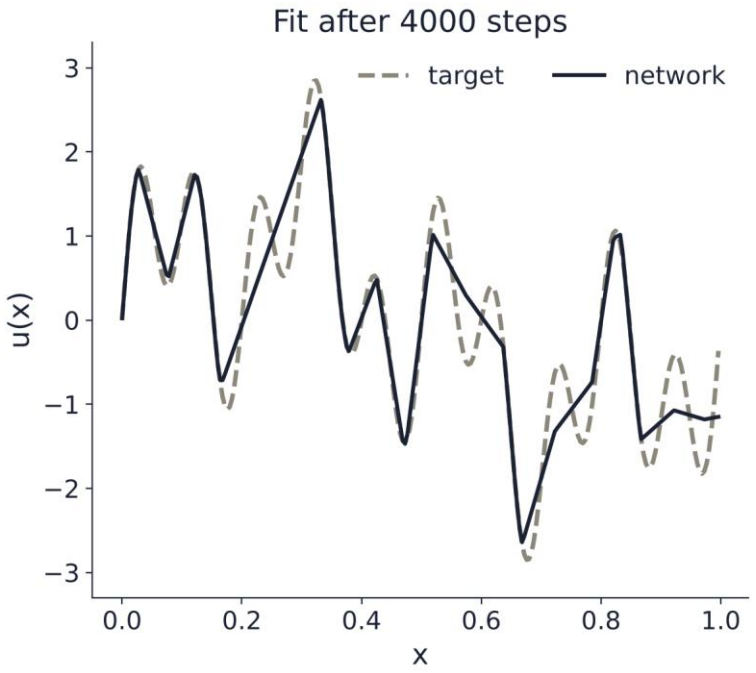


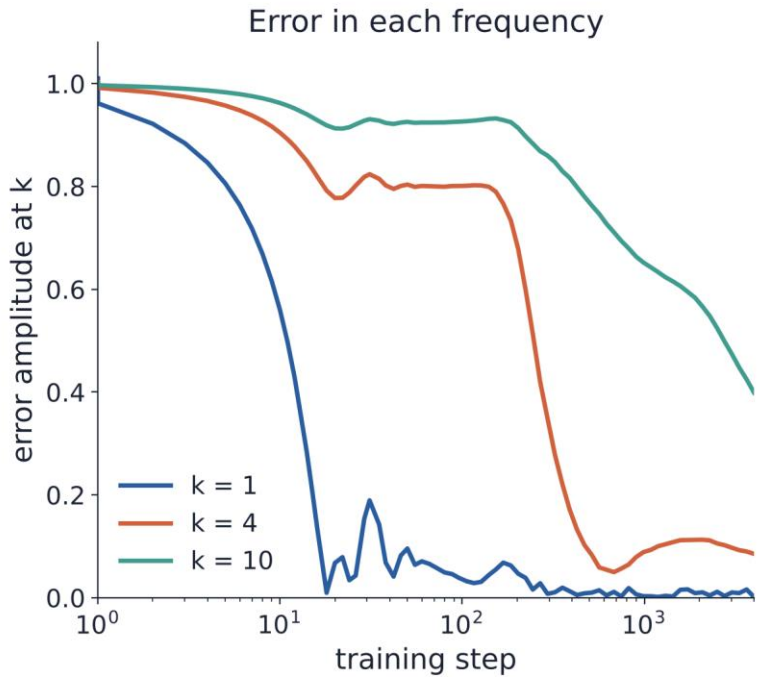
- One sine: one spike.
- Two sines: two spikes, heights ∝ amplitude² (0.3² = 0.09)
- Large scale = long wavelength = low k

For a 2D field, E sums $|\hat{u}|^2$ over all modes with the same |k|

Spectral bias

- Fit an MLP to $u(x) = \sin(2\pi x) + \sin(2\pi \cdot 4x) + \sin(2\pi \cdot 10x)$
- The error at k = 1 drops first, then at k = 4. At k = 10 it drops last and slowest
- Networks trained by gradient descent fit low frequencies first: **spectral bias**
- At a fixed training budget, **the small scales are the least accurate**

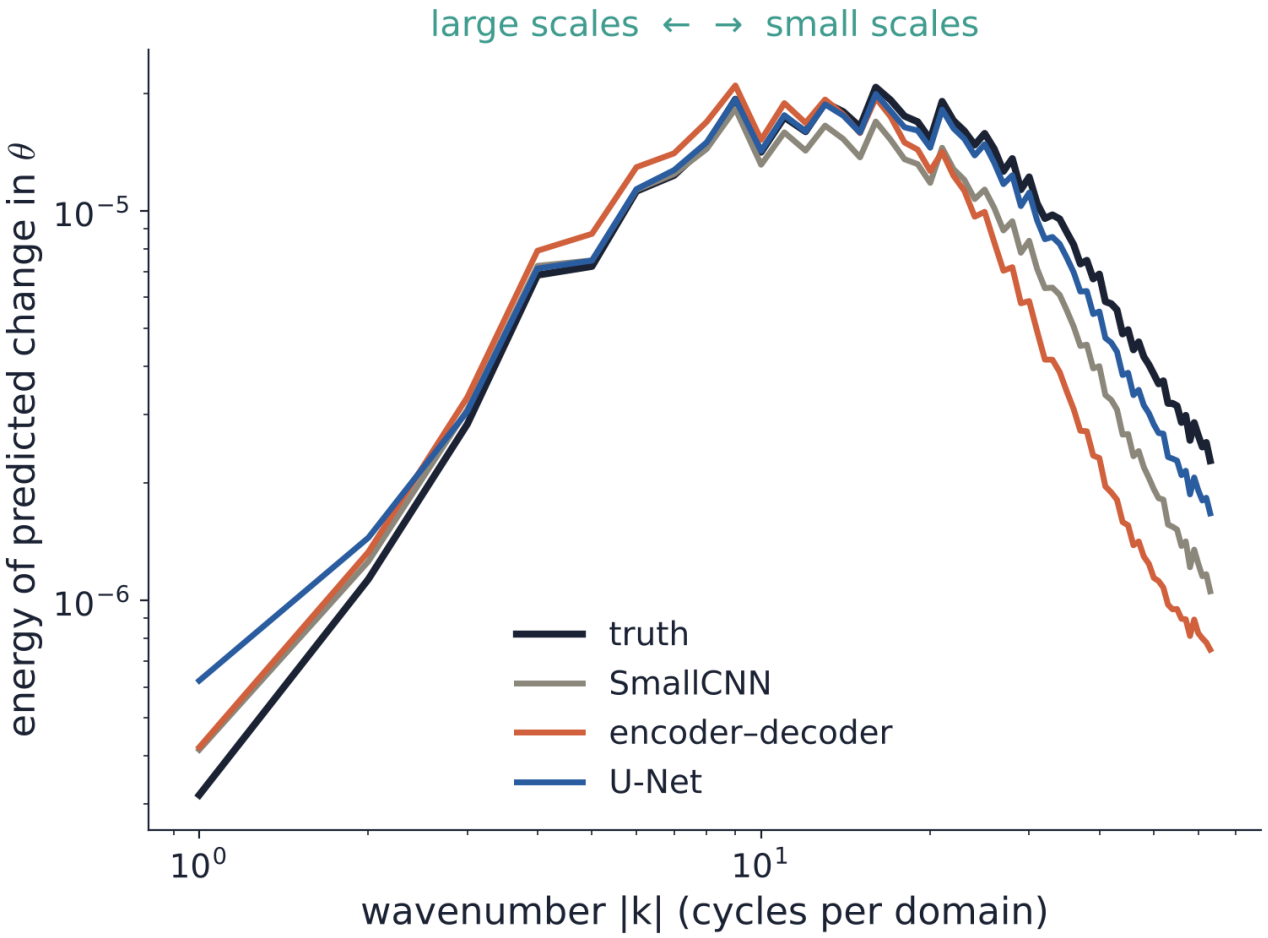




Error amplitude: magnitude of the Fourier coefficient of the residual at k. Rahaman et al., ICML 2019; Xu et al., 2019

Energy spectra of CNN predictions on BubbleML

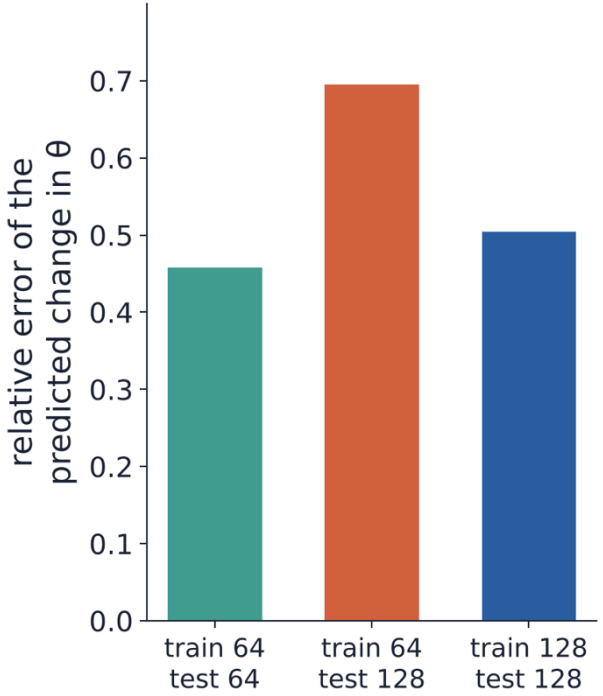
- Large scales (low k): all three models match the truth
- Small scales (high k): the predictions have **too little energy** (i.e., blurred prediction)
  - Encoder-decoder loses the most; U-Net stays closest to the truth
- The U-Net's error is only slightly below SmallCNN's, but its spectrum is much closer to the truth: compare spectra as well as the error
- For coordinate networks (neural fields, Lecture 4): **Fourier features** of the coordinates help fit high frequencies

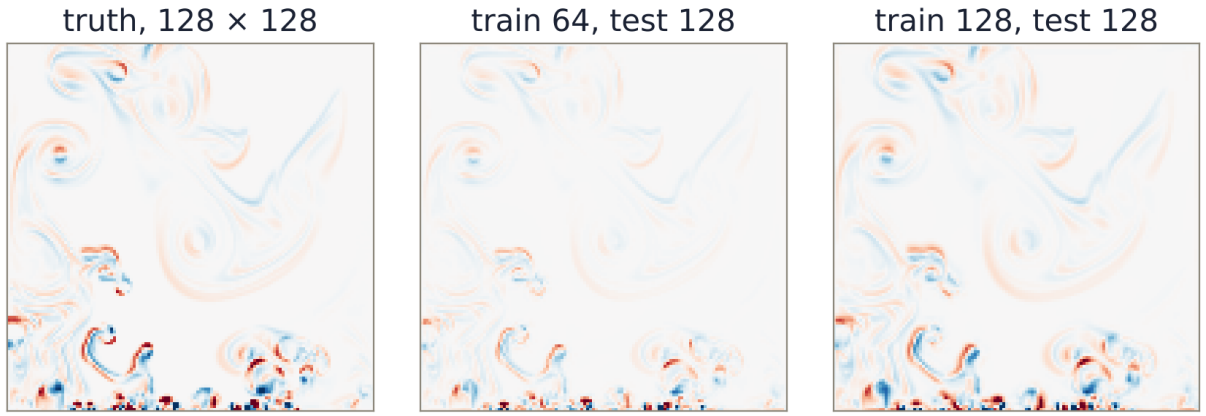


Fourier features: Tancik et al., NeurIPS 2020

Resolution dependence of CNNs

- Train SmallCNN at 64 x 64 and test it at 128 x 128 on the same frames: the **error grows**
- Its 3 x 3 kernels learned stencils for one cell size, like a finite-difference stencil with fixed grid spacing. On a finer grid, the receptive field covers half the physical distance
- A **CNN learns a map between arrays at one resolution**. **Operator learning** builds models that work across resolutions





PDEArena: Gupta & Brandstetter, TMLR 2023

Summary

- Receptive field: $r \leftarrow r + (k - 1)\, j$ and $j \leftarrow j\, s$, layer by layer. It must cover the distance information travels per frame, in cells
- Rotating the input is equivalent to rotating the kernel. Group convolutions keep all rotations as channels
- Dilation and downsampling increase r with few weights
- Residual connections enable deep networks: the identity path carries the gradient
- Encoder-decoders lose fine detail in the bottleneck; U-Net skip connections restore it. A U-Net has the structure of a multigrid V-cycle
- Spectral bias: networks fit large scales first. Compare energy spectra as well as the error
- CNN weights are tied to the grid spacing

After class

- Prince, *Understanding Deep Learning* (udlbook.github.io)
  - Ch. 10: convolutional networks · Ch. 11: residual networks and U-Nets
- Ronneberger, Fischer & Brox, U-Net: Convolutional Networks for Biomedical Image Segmentation, MICCAI 2015
- Luo, Li, Urtasun & Zemel, Understanding the Effective Receptive Field in Deep Convolutional Neural Networks, NeurIPS 2016
- Briggs, Henson & McCormick, *A Multigrid Tutorial*, 2nd ed., SIAM 2000, Ch. 1 to 3
- Rahaman et al., On the Spectral Bias of Neural Networks, ICML 2019
- Gupta & Brandstetter, Towards Multi-spatiotemporal-scale Generalized PDE Modeling (PDEArena), TMLR 2023

Next, Lecture 7 (Thu, Oct 15): Autoregressive rollout models

NEXT: HANDS-ON

From concepts to code

→ **Open the Lecture 6 notebook**

1. Compute the receptive field of SmallCNN and the U-Net encoder with the r, j rule; check it by backpropagating from one output cell

2. Build the U-Net from the diagram and count its parameters

3. Train SmallCNN and the U-Net on BubbleML at 128 x 128; compare one-step error and energy spectra

4. Remove the skip connections, retrain, and compare the spectra again