Introduction to Deep Learning (I2DL)

Tutorial 10: Semantic Segmentation

Today's Outline

- **Exercise 9: Results**
  - the facial-keypoint leaderboard
  - how the top submission was built
- **Semantic Segmentation**
  - the task, labels, loss and accuracy
  - architectures and upsampling

Exercise 9: Leaderboard

Leaderboard

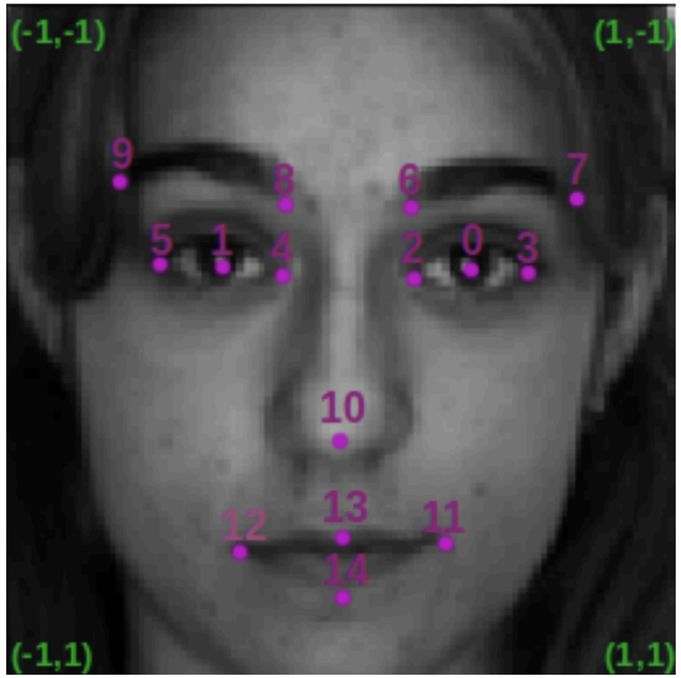
The leaderboard shows for each exercise the highest scoring submission from each user. Only valid submissions are displayed.

Exercise 1 | Exercise 3 | Exercise 4 | Exercise 5 | Exercise 6 | Exercise 7 | Exercise 8 | Exercise 9 | Exercise 10 | Exercise 11 | Exercise 12

#	User	Score
1	u0994	2309.67
2	u1969	1787.42
3	u1503	1588.51
4	u0456	1498.46
5	u1258	1481.95
6	u1619	1352.68
7	u1636	1221.95
8	u1626	1216.56
9	u0362	1206.27
10	u1557	1047.45

Recap: Facial Keypoint Detection

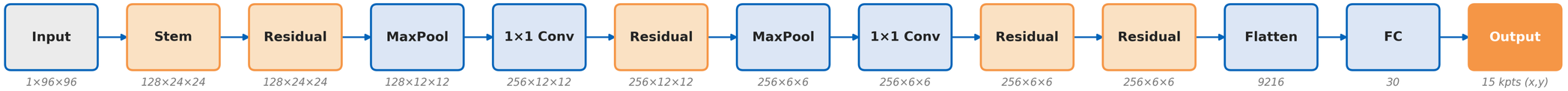
- **Input:** a 1×96×96 grayscale image
- **Output:** 15 facial keypoints, each an (x, y)
- **Score** $= \frac{1}{2\,\mathrm{MSE}}$ — smaller error, higher score
- **Threshold:** score of 100 ⇔ MSE below 0.005



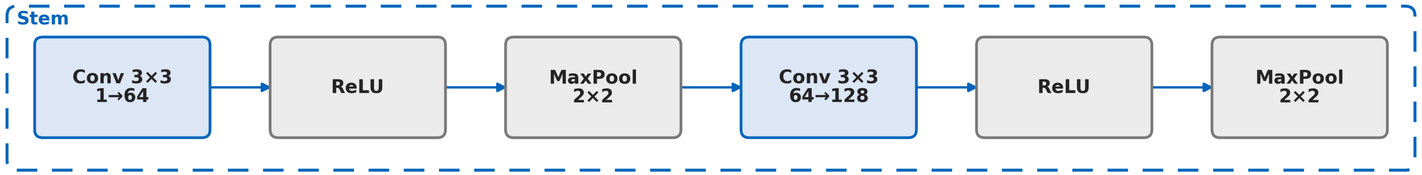
How the #1 Submission Scored 2309

The Model — a Residual CNN

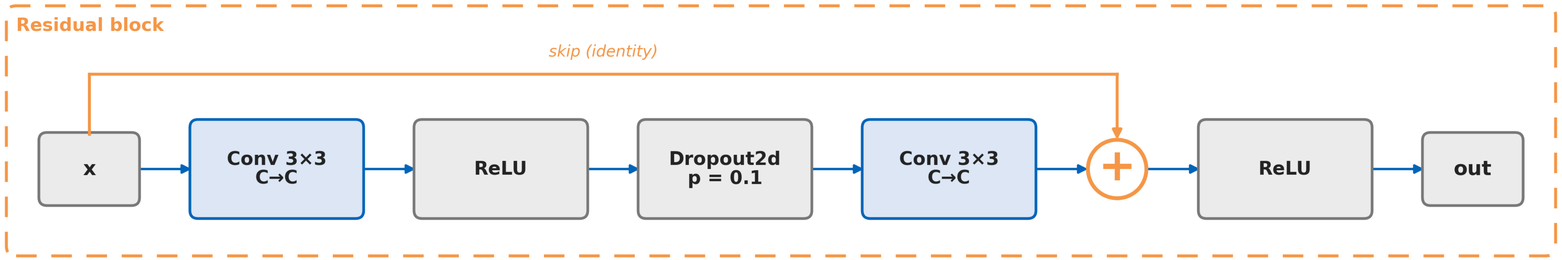
1 · Overall pipeline



2 · Stem



3 · Residual block

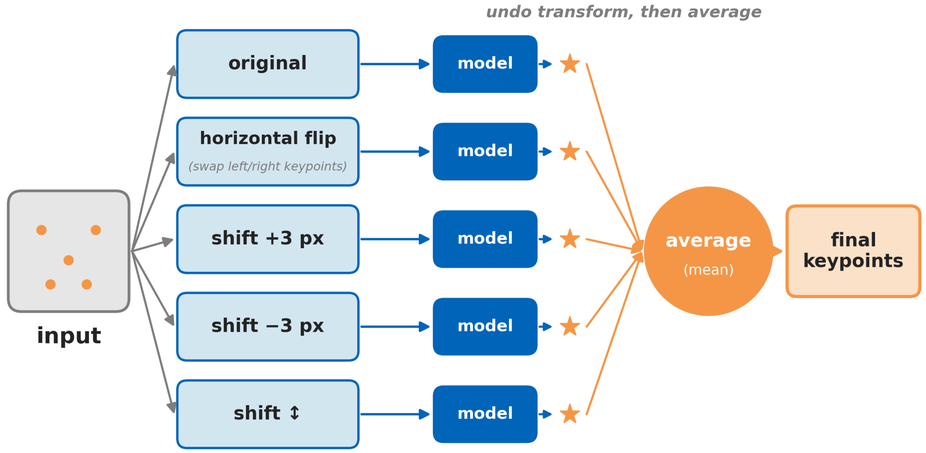


Training Recipe

- **Data:** the facial-keypoint images — 1×96×96 grayscale
- **Optimizer:** adam — learning rate 3e-4, batch size 4
- **Loss:** mean squared error (MSE)
- **Schedule:** train to convergence, early stopping (patience 100)
- **Augmentation:** horizontal flip (swap left/right) + small rotations & shifts

The Winning Trick — Test-Time Augmentation

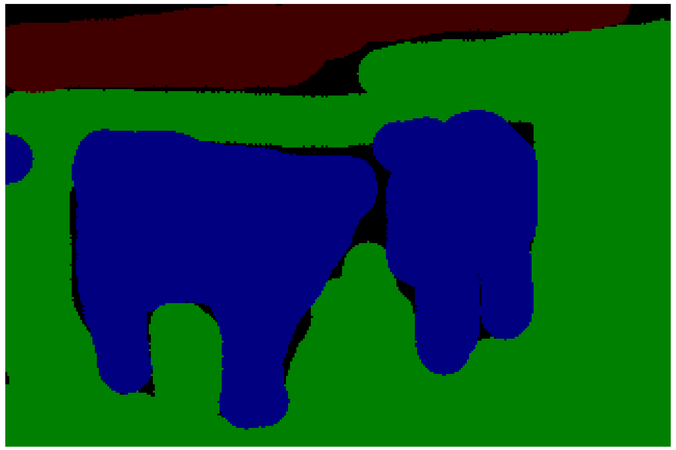
- Predict on the **horizontal flip and small shifts**, undo each, then **average**
- Averaging cancels the jitter in any single prediction, the score jumps



Exercise 10: Semantic Segmentation

Semantic Segmentation

- Not one label for the whole image — a label for **every pixel**
- **Input:** a 3×W×H RGB image
- **Output:** a 23×W×H map — class scores at every pixel



Segmentation Labels

- Every class has a fixed **RGB color**
- 23 classes: building, grass, tree, cow, sky, ...
- **void** (black) marks unlabelled pixels — we ignore them

"void" = unlabelled pixels

object class	*R*	*G*	*B*	*Colour*
void	0	0	0	
building	128	0	0	
grass	0	128	0	
tree	128	128	0	
cow	0	0	128	
horse	128	0	128	
sheep	0	128	128	
sky	128	128	128	
mountain	64	0	0	

Metric — Loss Function

- **Averaged per-pixel cross-entropy** loss

```
for (inputs, targets) in train_data[0:4]:
    inputs, targets = inputs, targets
    outputs = dummy_model(inputs.unsqueeze(0))
    loss = torch.nn.CrossEntropyLoss(ignore_index=-1, reduction='mean')
    losses = loss(outputs,targets.unsqueeze(0))
    print(losses)
```

- Set **ignore_index = −1** so the void pixels never contribute

Metric — Accuracy

- Fraction of **correctly classified pixels**, void pixels excluded

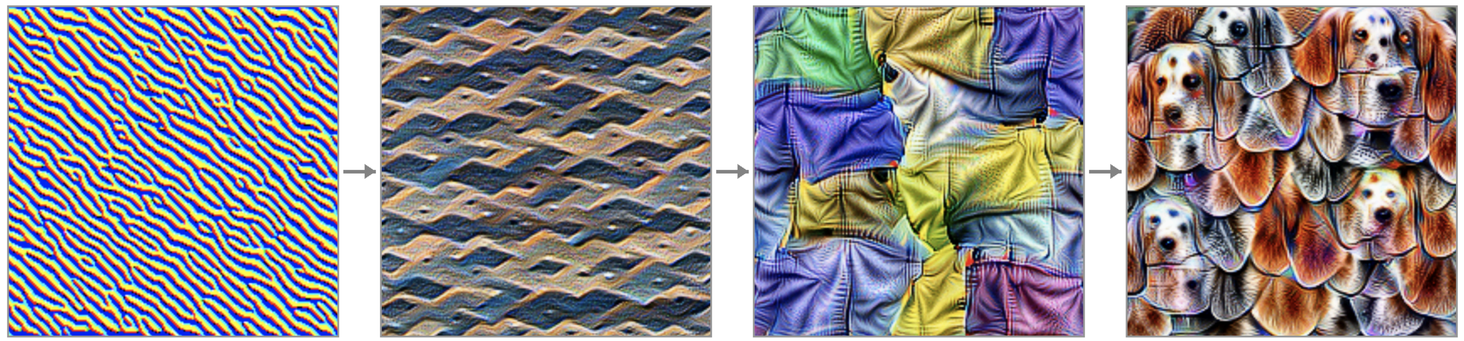
```
def evaluate_model(model):
    test_scores = []
    model.eval()
    for inputs, targets in test_loader:
        inputs, targets = inputs.to(device), targets.to(device)

        outputs = model.forward(inputs)
        _, preds = torch.max(outputs, 1)
        targets_mask = targets >= 0
        test_scores.append(np.mean((preds == targets)[targets_mask].data.cpu().numpy()))


    return np.mean(test_scores)
print("Test accuracy: {:.3f}".format(evaluate_model(dummy_model)))
```

- **To pass:** reach an accuracy of at least 64%

Model Architecture

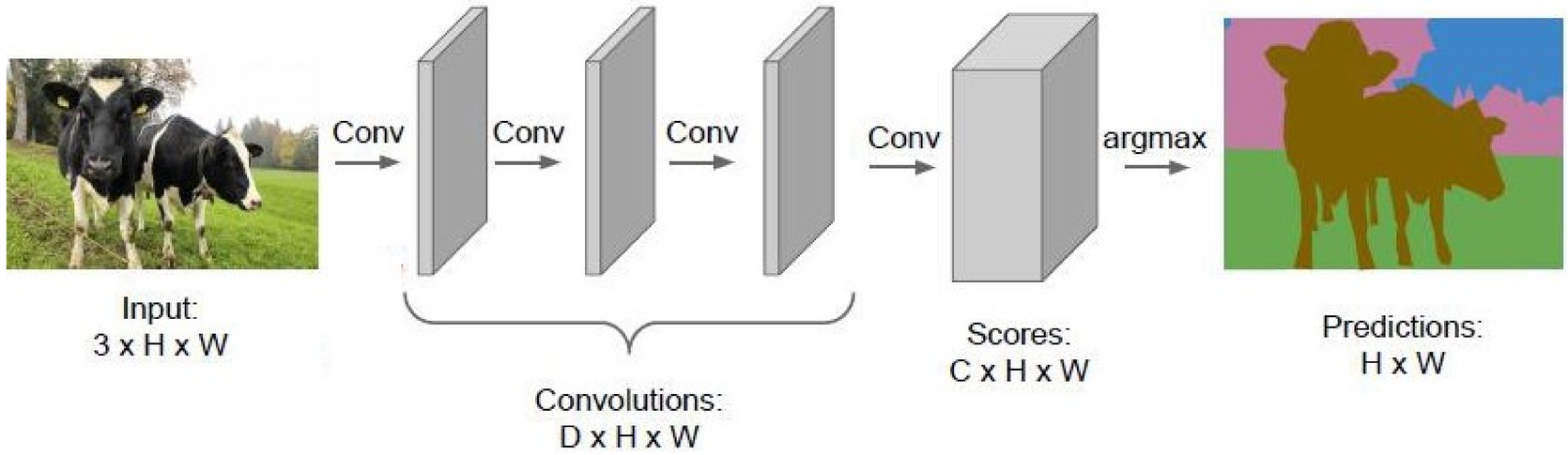
Maintaining Resolution

- **Input:** (N, C, H, W) → **Output:** (N, classes, H, W)
- We must **keep the height and width** of the input
- ...while capturing features from edges and textures up to parts and whole objects



Naive Solution

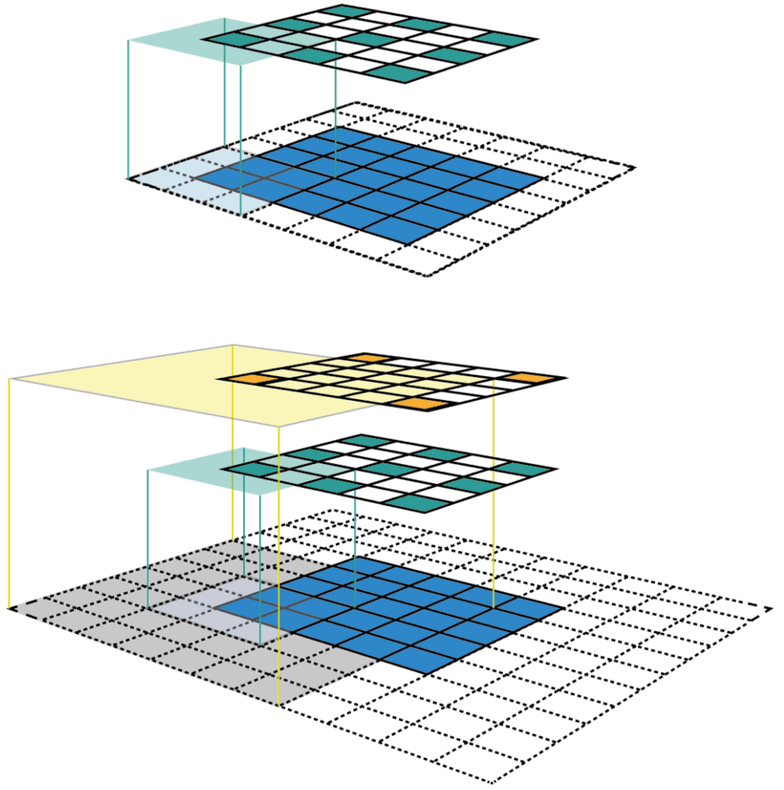
- Keep the resolution **constant** through the whole network



- **Problem:** every layer stores a full-resolution map — memory blows up to millions

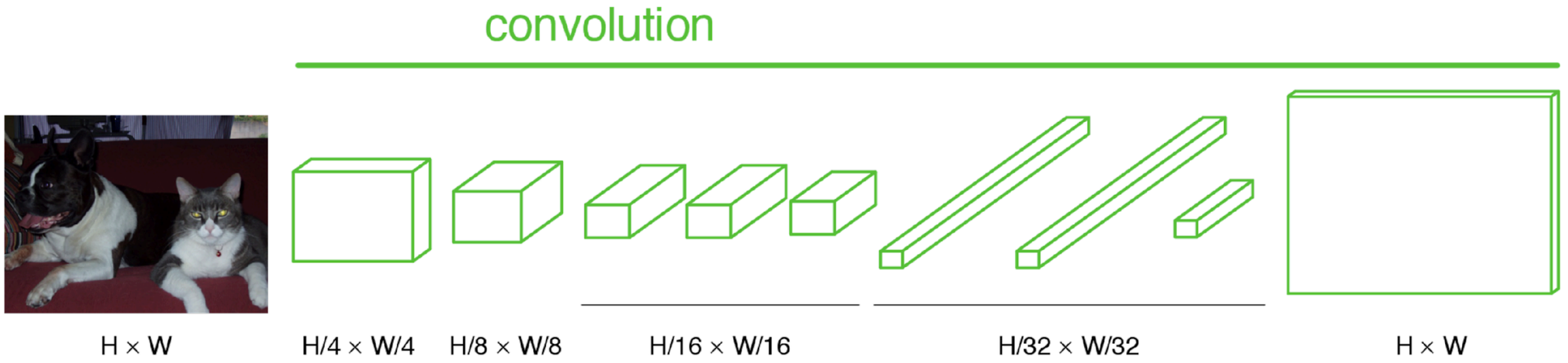
Excursion: Receptive Field

- **Receptive field:** the region of the input one feature looks at
- Convolutions and pooling **grow** the receptive field…
- …but **shrink** the resolution — a bigger field means a coarser map



Coming from Classification

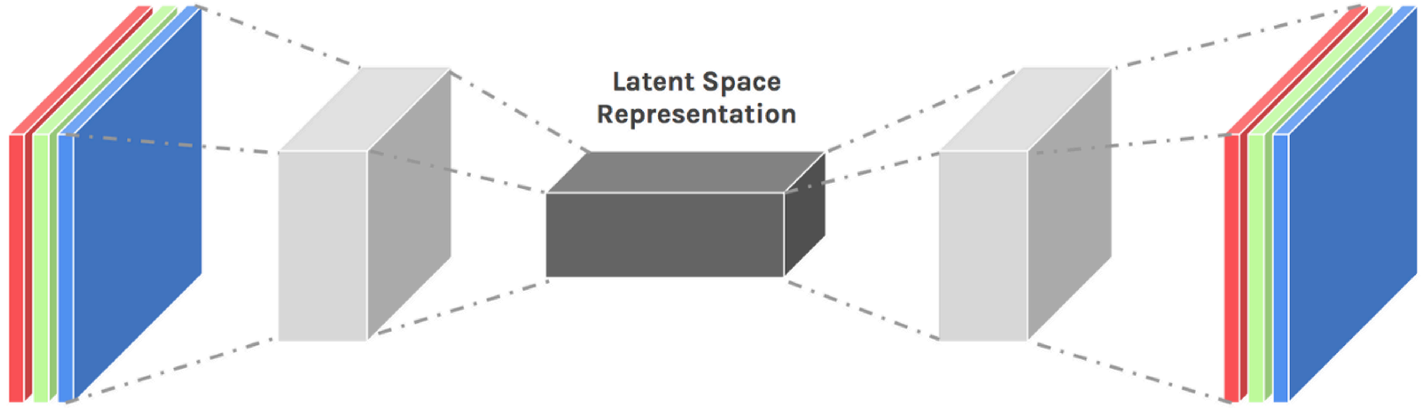
- Strided convolutions and pooling **grow the receptive field**



- Then **upsample** the result back to the input resolution

Better Solution: Encoder–Decoder

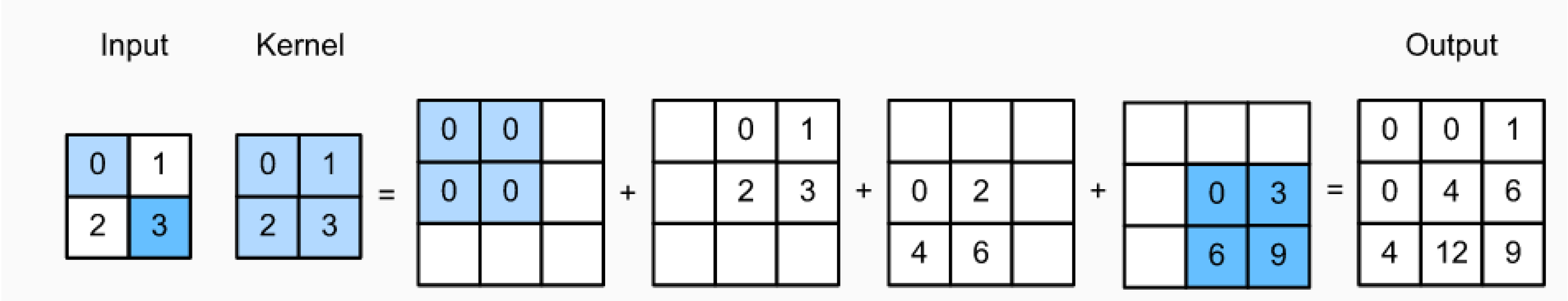
- Slowly **reduce** the size, then slowly **increase** it again



- Pooling ↔ **upsampling**, strided conv ↔ **transposed conv**

Transposed Convolutions

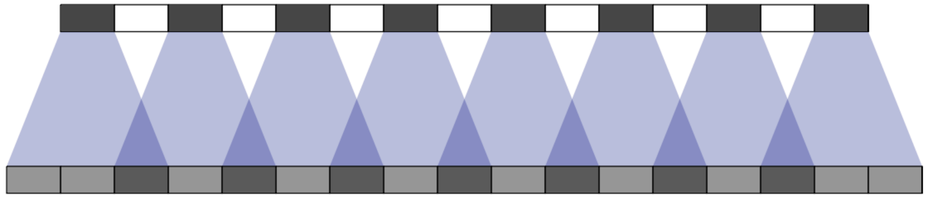
- Roughly the **inverse** of a normal convolution



- Each input value scales the kernel into a larger output, and overlaps are summed

Are Transposed Convolutions Superior?

- **Short answer:** no, not always
- They can produce **checkerboard artifacts** in the output
- **A safer go-to:** plain upsampling, followed by a convolution



Getting Results Quickly: Transfer Learning

- The fastest path to a good score is **transfer learning**
- Take a **pretrained encoder** (alexnet, mobilenet) and attach a fresh decoder
- Reuse the features it already learned — a strong result with far less training

Good luck, and see you next week!