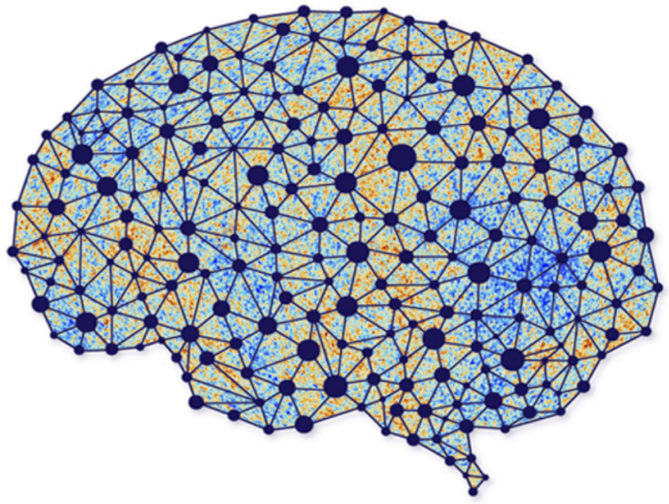
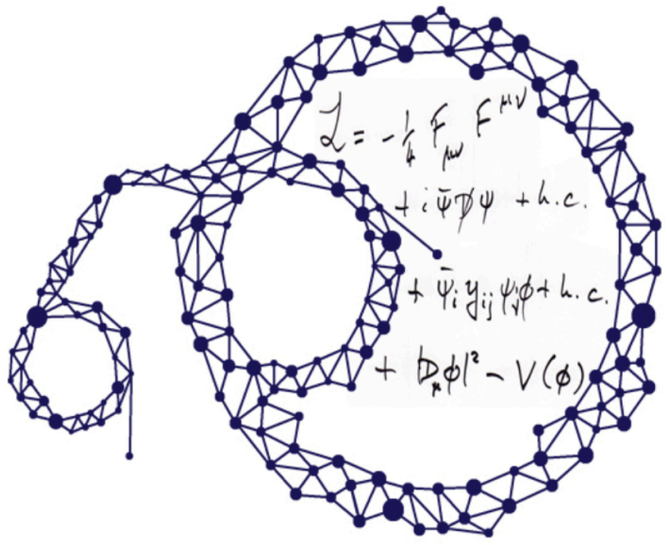
Physics 361 - Machine Learning in Physics

Lecture 10 – CNN and Field-to-Field

Feb. 20th 2025



AI
∩
Universe



Moritz Münchmeyer

Convolutional Neural Networks

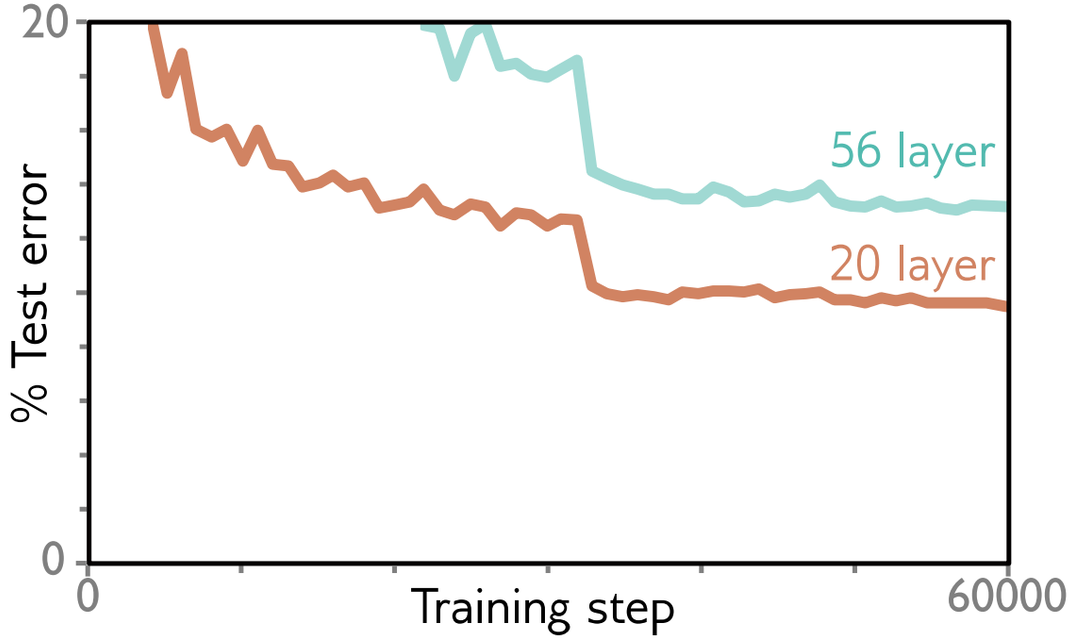
Residual networks (ResNets) Cont.

Going deeper

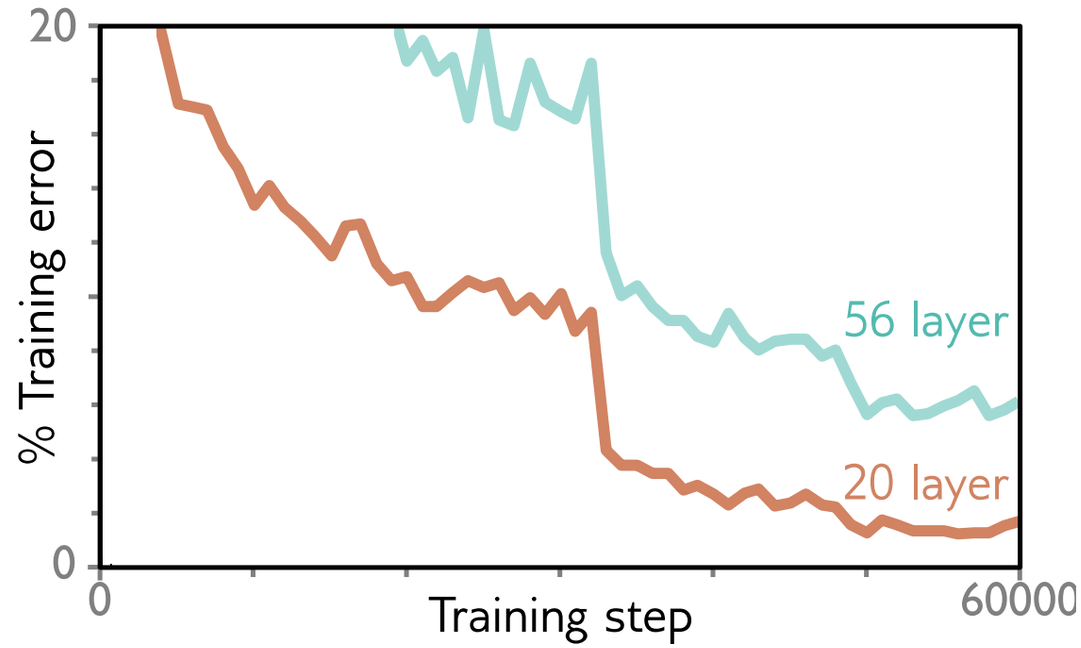
- Image classification performance improved as the depth of convolutional networks was extended from eight layers (AlexNet) to nineteen layers (VGG). This led to experimentation with even deeper networks. However, **performance decreased again when many more layers were added.**
- A novel idea to **overcome this problem are residual blocks**. Here, each network layer computes an additive change to the current representation instead of transforming it directly.
- **Residual blocks** allow much deeper networks to be trained, and these networks improve performance across a variety of tasks.

CIFAR Image classification for deeper networks

a)

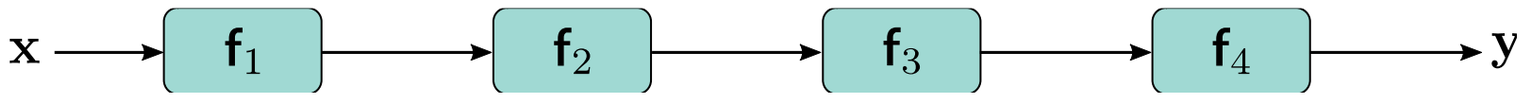


b)



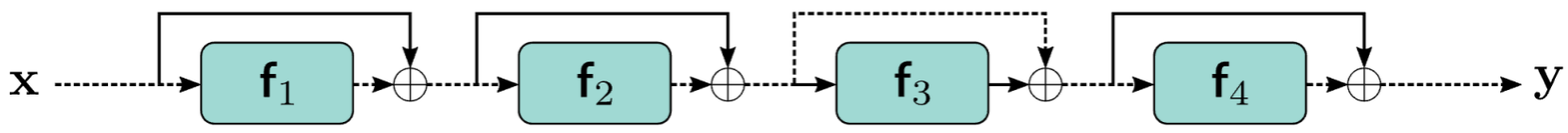
Regular network:

$$\mathbf{h}_1 = \mathbf{f}_1[\mathbf{x}, \boldsymbol{\phi}_1]$$
$$\mathbf{h}_2 = \mathbf{f}_2[\mathbf{h}_1, \boldsymbol{\phi}_2]$$
$$\mathbf{h}_3 = \mathbf{f}_3[\mathbf{h}_2, \boldsymbol{\phi}_3]$$
$$\mathbf{y} = \mathbf{f}_4[\mathbf{h}_3, \boldsymbol{\phi}_4]$$



Residual network (2016):

$$\mathbf{h}_1 = \mathbf{x} + \mathbf{f}_1[\mathbf{x}, \boldsymbol{\phi}_1]$$
$$\mathbf{h}_2 = \mathbf{h}_1 + \mathbf{f}_2[\mathbf{h}_1, \boldsymbol{\phi}_2]$$
$$\mathbf{h}_3 = \mathbf{h}_2 + \mathbf{f}_3[\mathbf{h}_2, \boldsymbol{\phi}_3]$$
$$\mathbf{y} = \mathbf{h}_3 + \mathbf{f}_4[\mathbf{h}_3, \boldsymbol{\phi}_4]$$



Order of operations is important

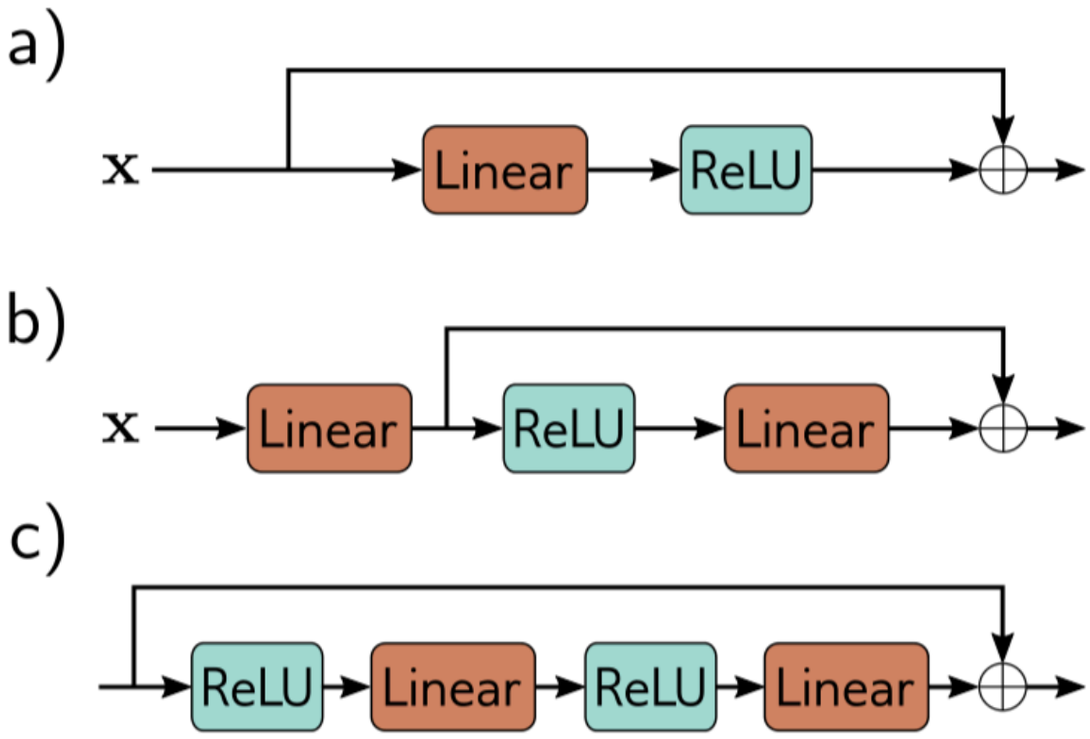


Figure 11.5 Order of operations in residual blocks. a) The usual order of linear transformation or convolution followed by a ReLU nonlinearity means that each residual block can only add non-negative quantities. b) With the reverse order, both positive and negative quantities can be added. However, we must add a linear transformation at the start of the network in case the input is all negative. c) In practice, it's common for a residual block to contain several network layers.

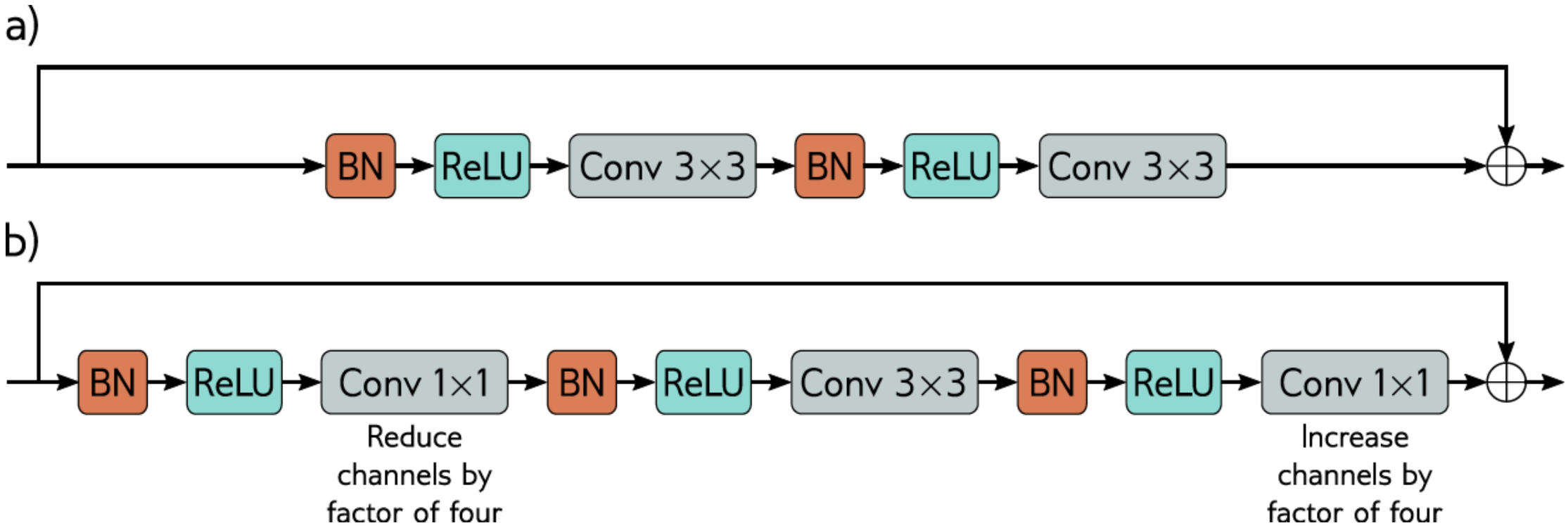
Batch normalization (in ResNets and other NN)

- Residual blocks employ **batch normalization**, which re-centers and rescales the activations (the output after applying the activation function) at each layer. This makes training more stable and solves the “exploding gradient problem”.
- **Batch normalization or BatchNorm** shifts and rescales each activation h so that its mean and variance across the batch B become values that are learned during training.
- During training, the batch statistics (mean and variance) are computed dynamically. During inference, a running estimate of mean and variance is maintained and used instead.

Why Use Batch Normalization?

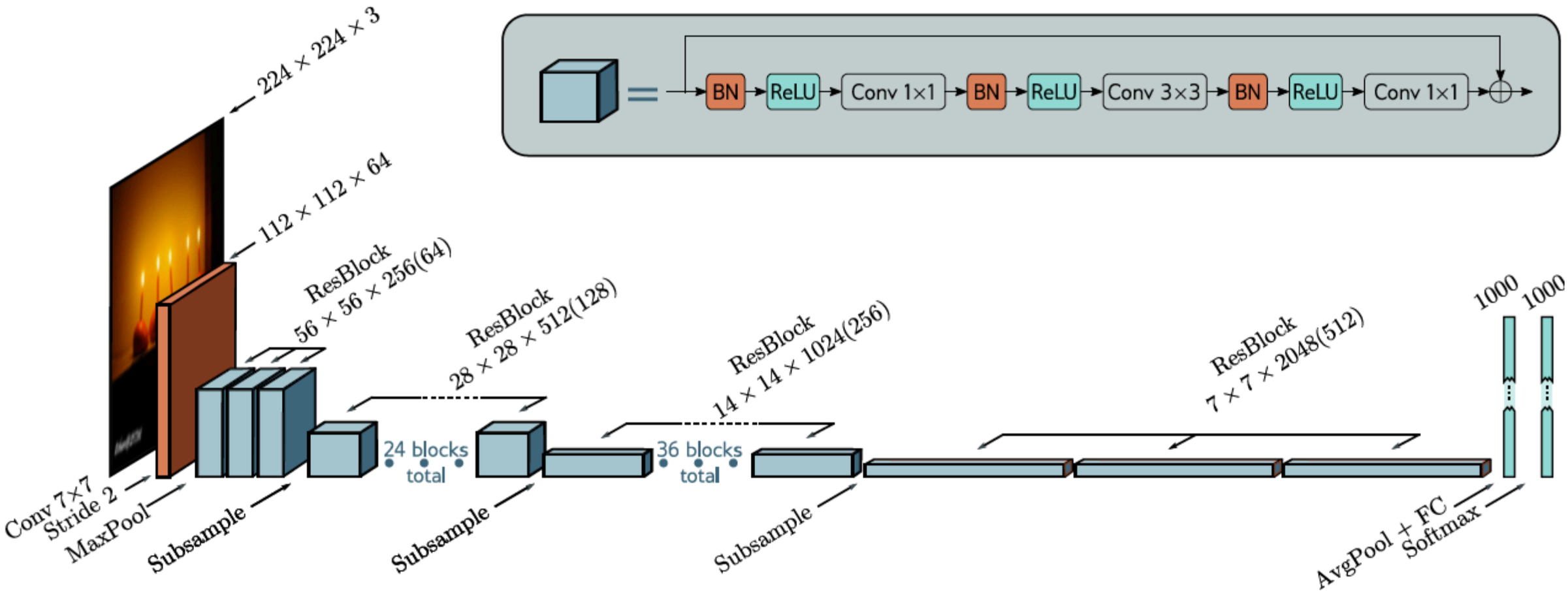
1. **Faster Convergence:** Reduces internal covariate shift, making training more stable and allowing higher learning rates.
2. **Regularization Effect:** Reduces reliance on dropout by adding noise to the activations, acting as a form of implicit regularization.
3. **Reduces Vanishing/Exploding Gradients:** Keeps activations in a stable range, improving gradient flow.
4. **Less Sensitivity to Initialization:** Helps mitigate issues from poorly initialized weights.

Resnet Block

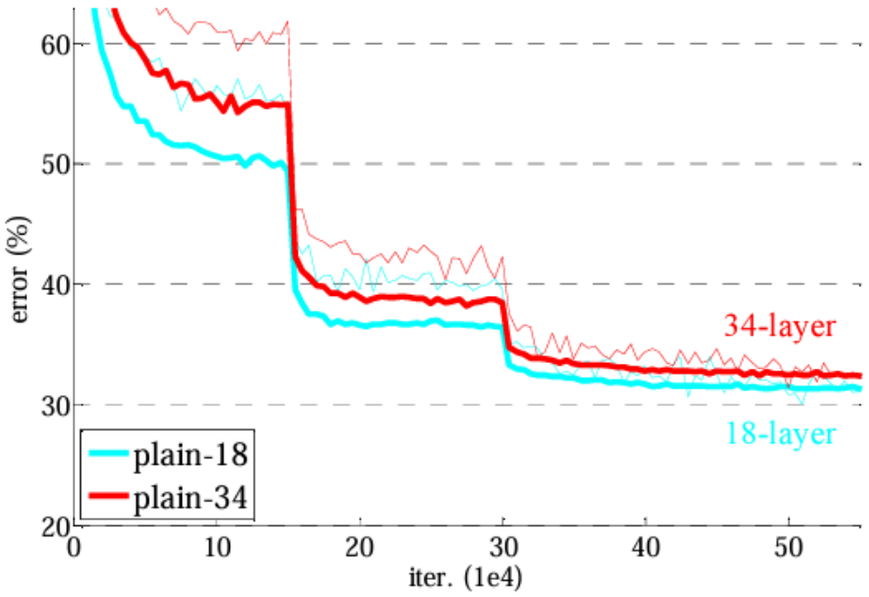


https://arxiv.org/abs/1512.03385 Deep Residual Learning for Image Recognition (250000 citations)

Resnet 200 (2016)



Resnet 200 (2016)



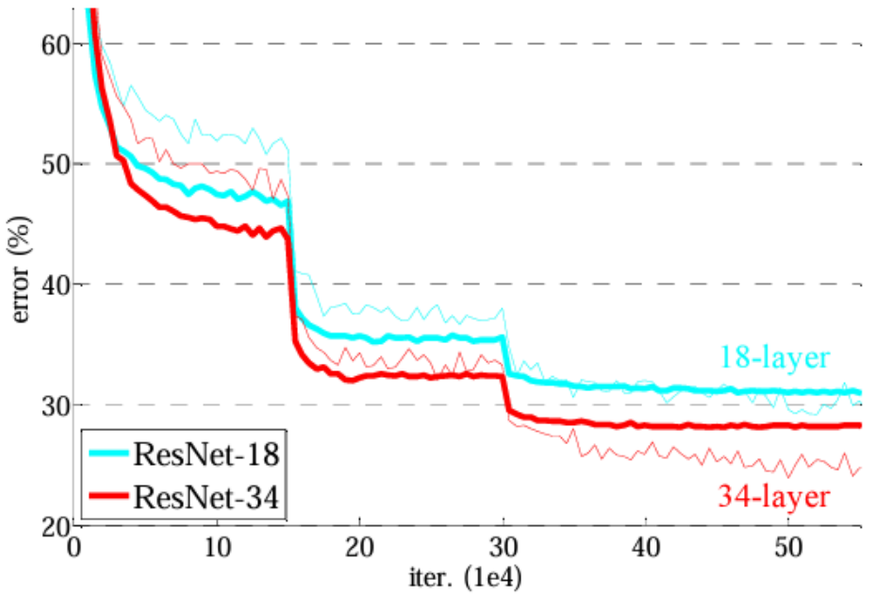
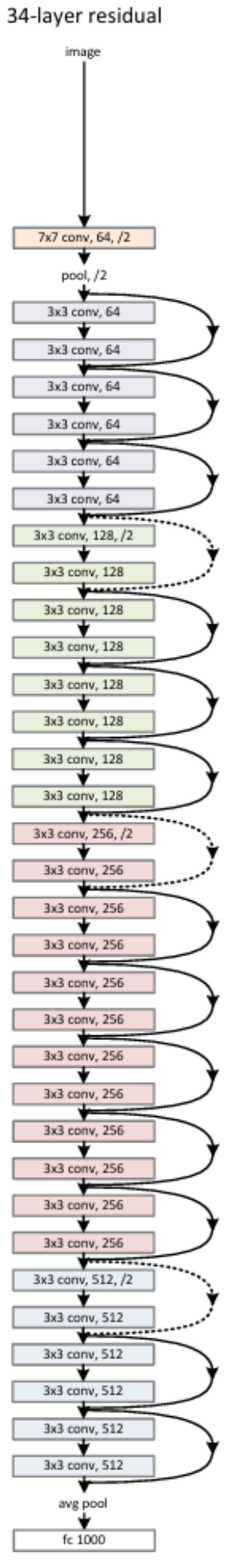


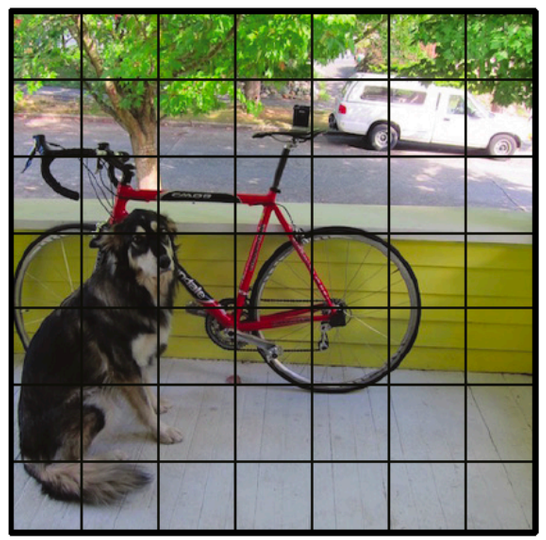
Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.



Convolutional Neural Networks

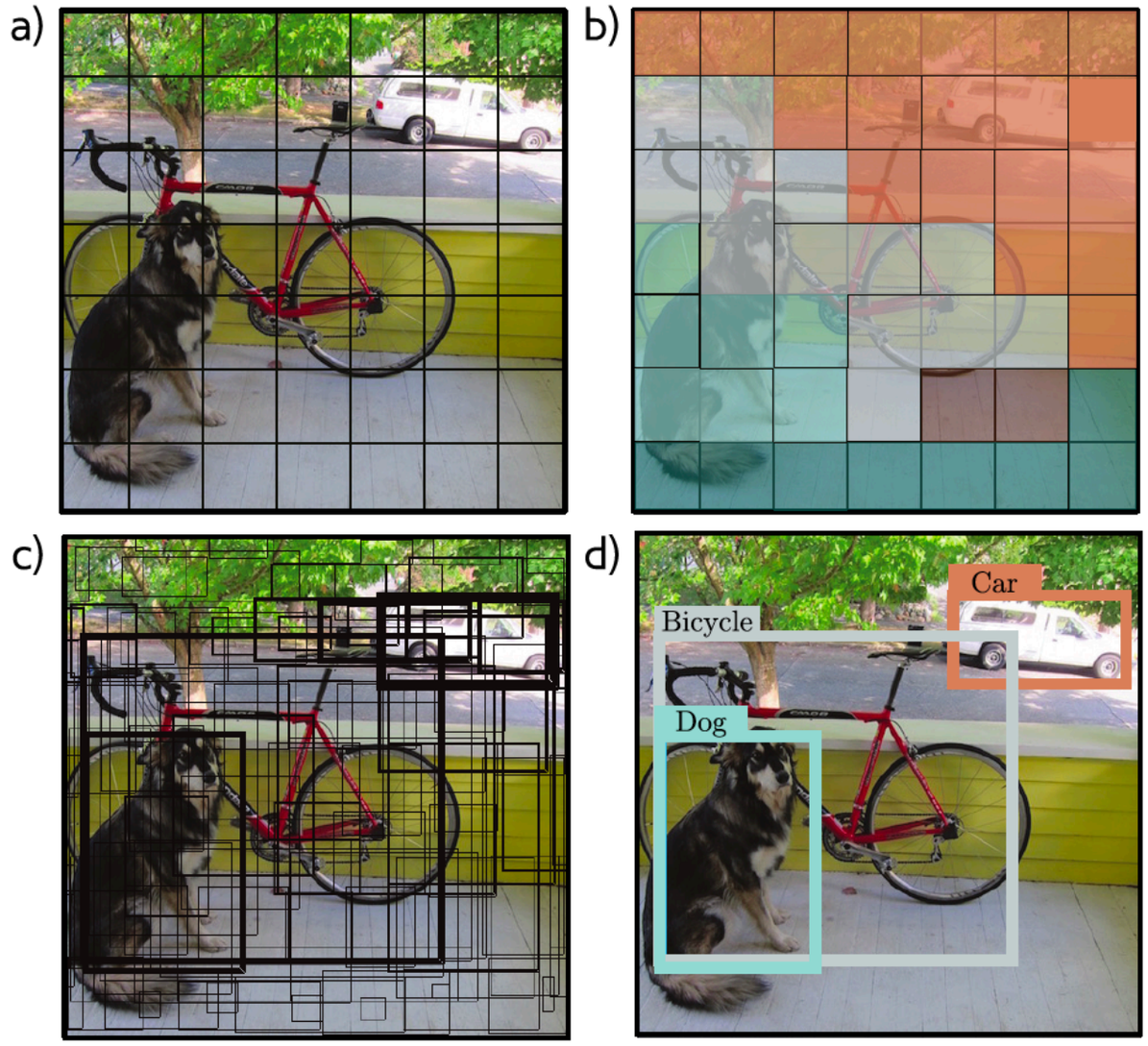
Object detection and semantic segmentation

Objects Detection: You Only Look Once (YOLO)

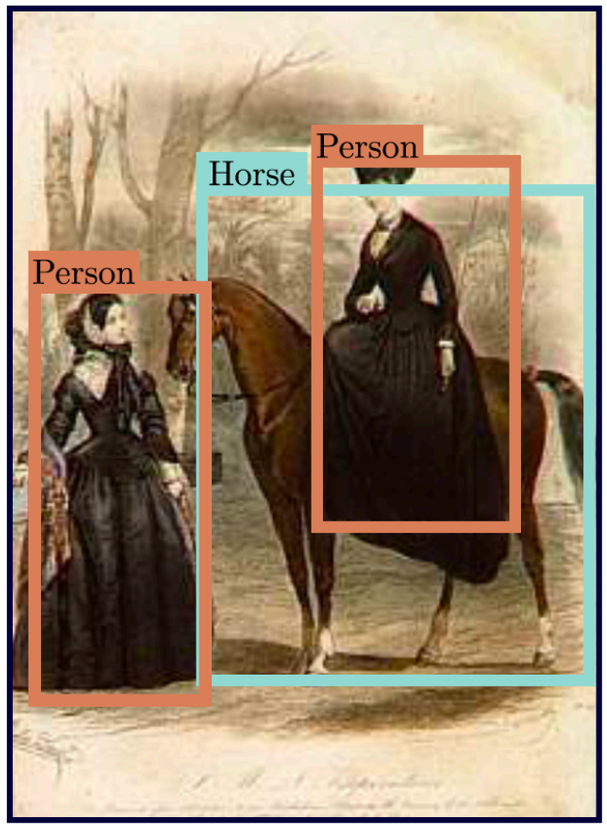


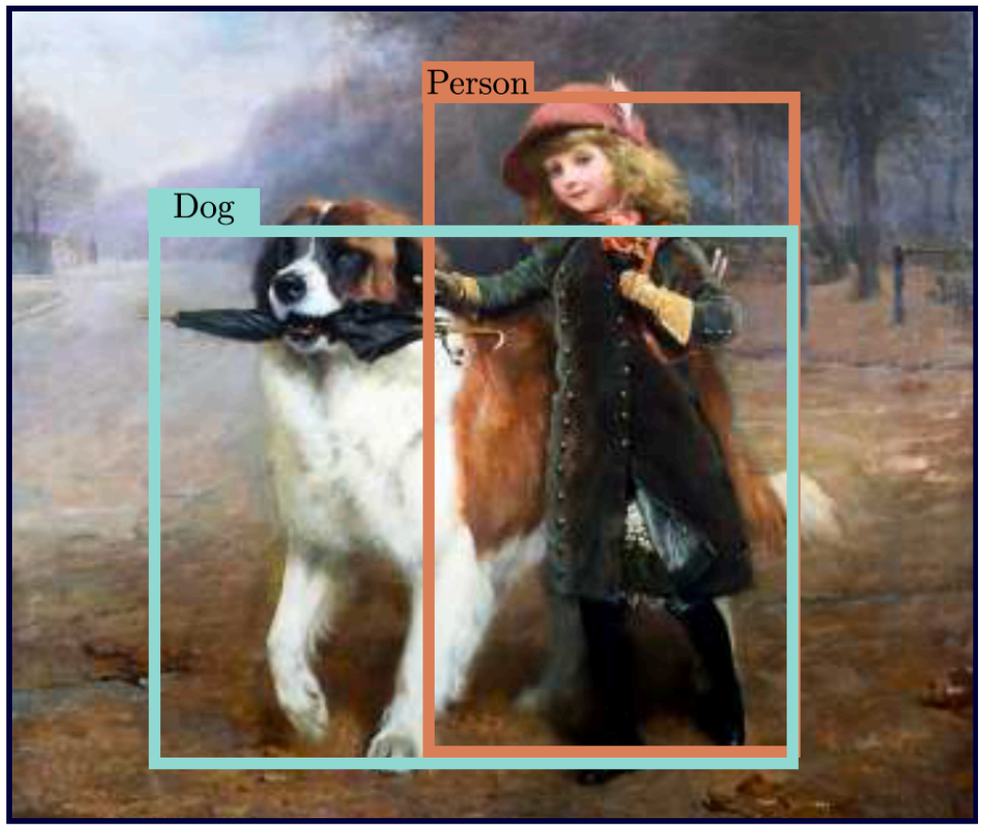
- Network similar to VGG (448x448 input)
- 7×7 grid of locations
- Predict class at each location
- Predict 2 bounding boxes at each location
  - Five parameters –x,y, height, width, and confidence
- Momentum, weight decay, dropout, and data augmentation
- Heuristic at the end to threshold and decide final boxes

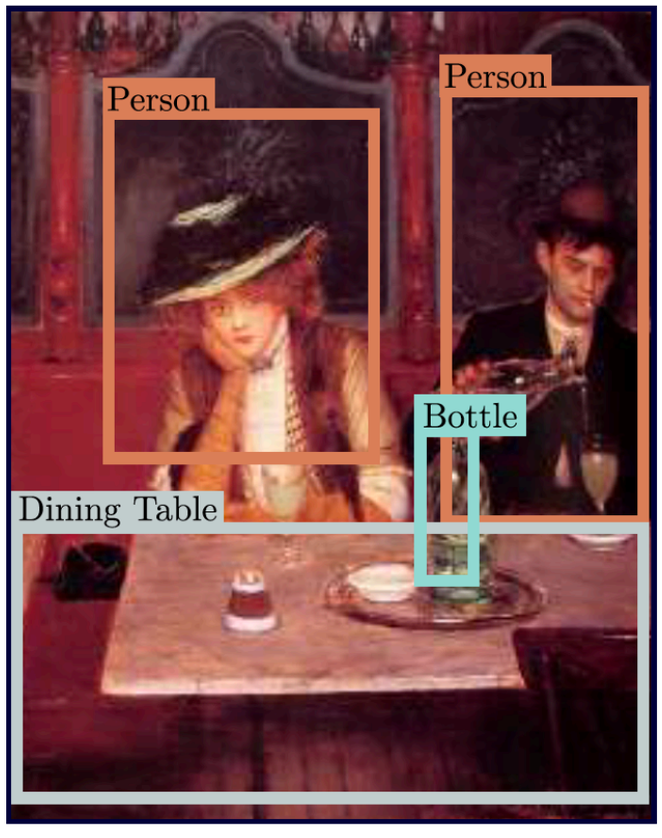
Object detection (YOLO)



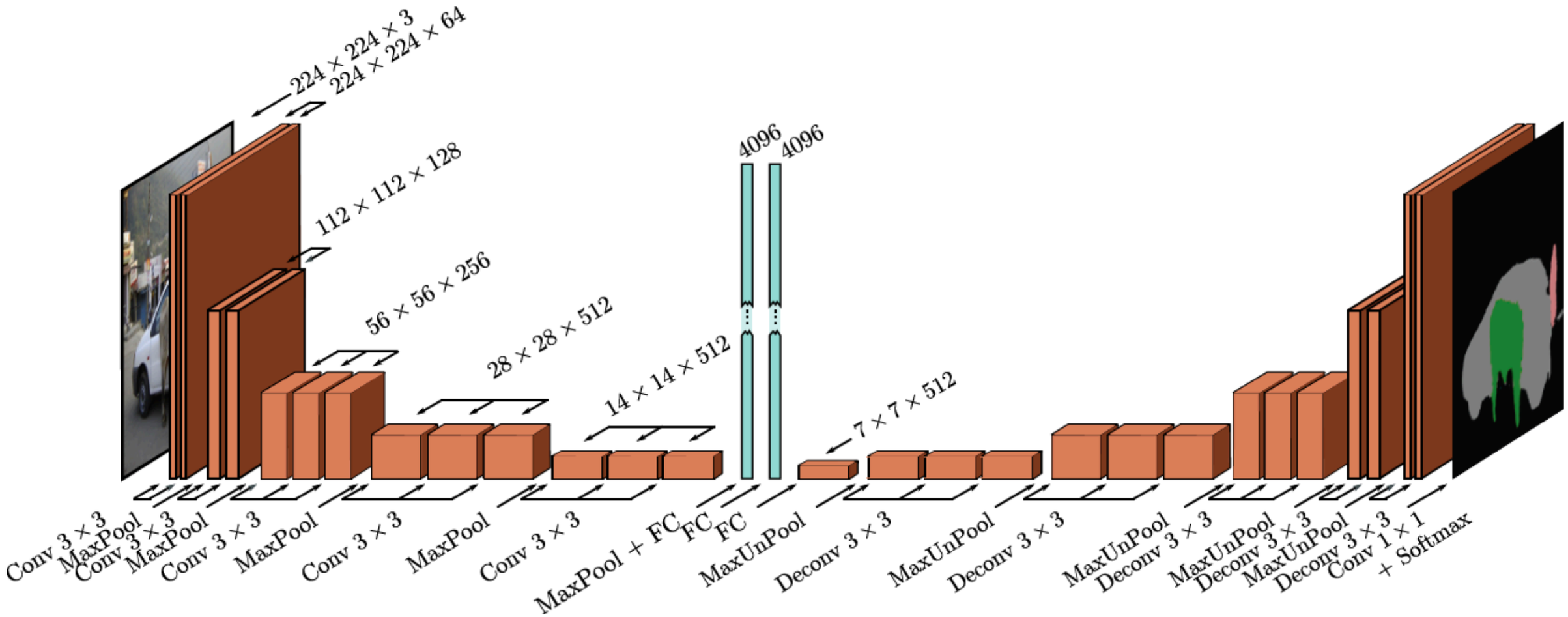
Results


Person
Horse
Person


Person
Dog


Person
Person
Bottle
Dining Table

Semantic Segmentation with Encoder-Decoder architecture (2015)

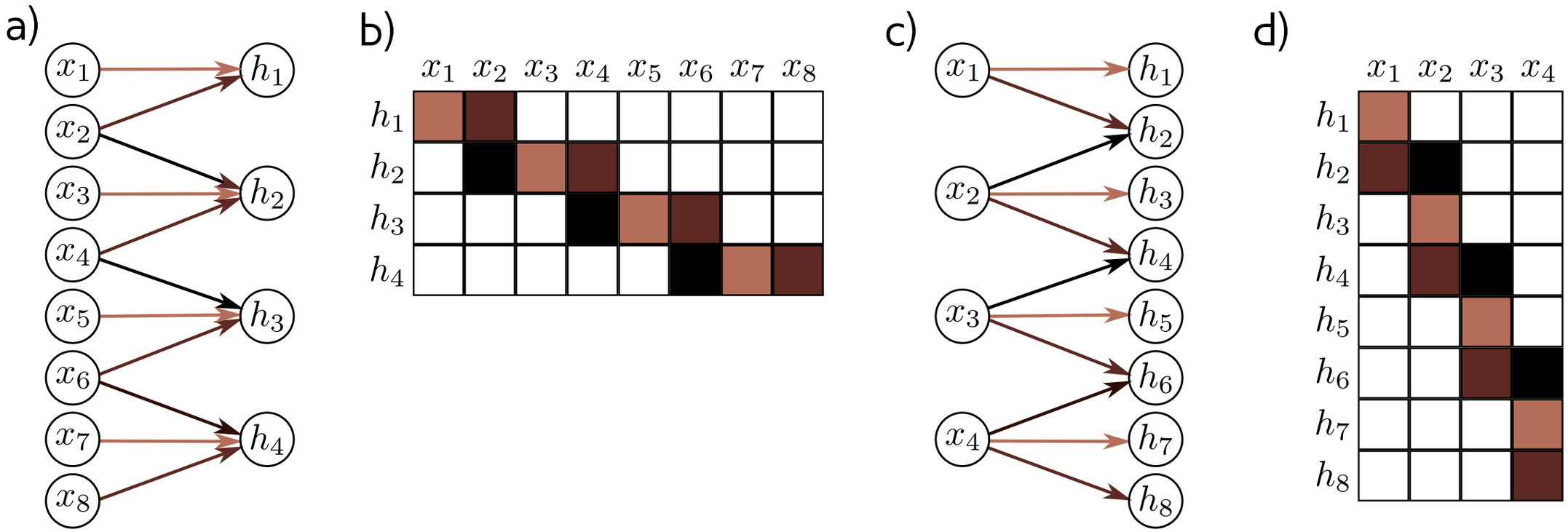


Encoder

Decoder

This is the first time we encounter an encoder-decoder architecture, which will become more common in later lectures.

Recall: Transposed convolutions (Deconvolutions)

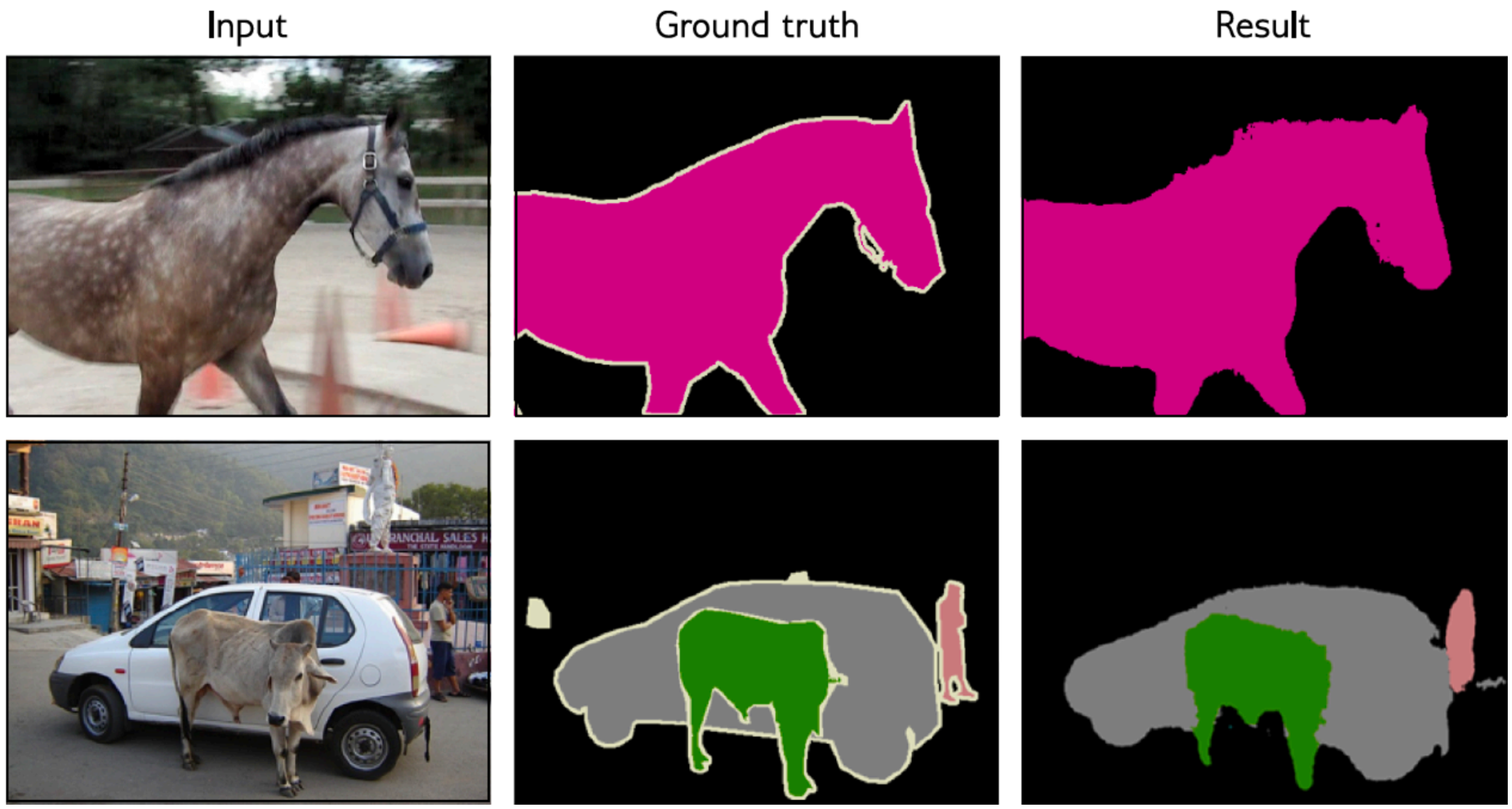


Normal Convolution

Transposed convolution

Transposed convolution in 1D. a) Downsampling with kernel size three, stride two, and zero-padding. Each output is a weighted sum of three inputs (arrows indicate weights). b) This can be expressed by a weight matrix (same color indicates shared weight). c) In transposed convolution, each input contributes three values to the output layer, which has twice as many outputs as inputs. d) The associated weight matrix is the transpose of that in panel (b).

Semantic segmentation results



Convolutional Neural Networks

Symmetries beyond translation invariance

Learning with Symmetries

- **Locality:** features that define a “cat” are local in the picture: whiskers, tail, paws, …

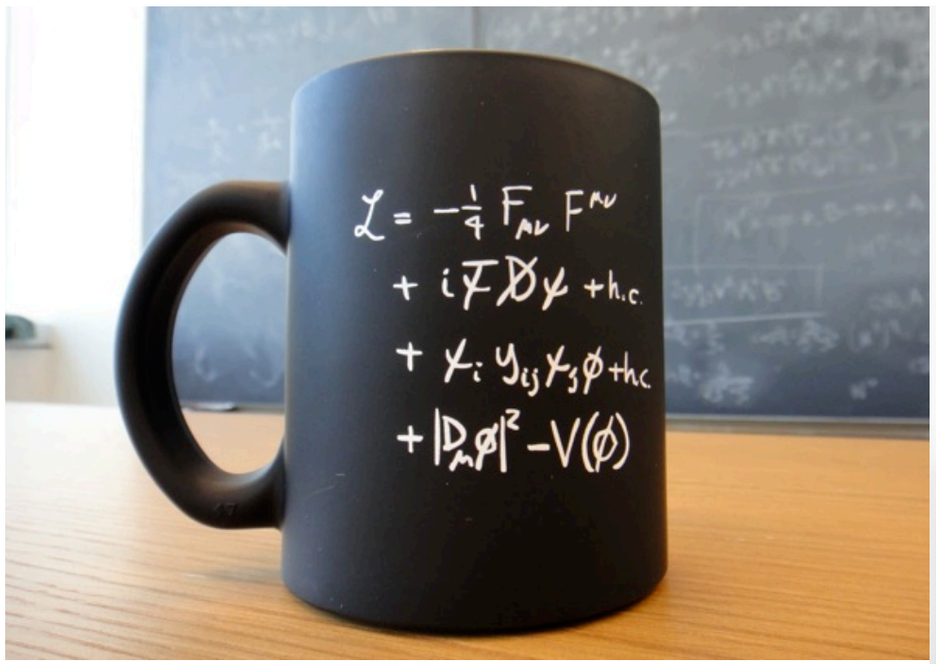
- **Translational invariance:** Cats can be anywhere in the image.

- **Rotational invariance:** Relative position of features must be respected (e.g. whiskers and tail should appear on opposite sides)

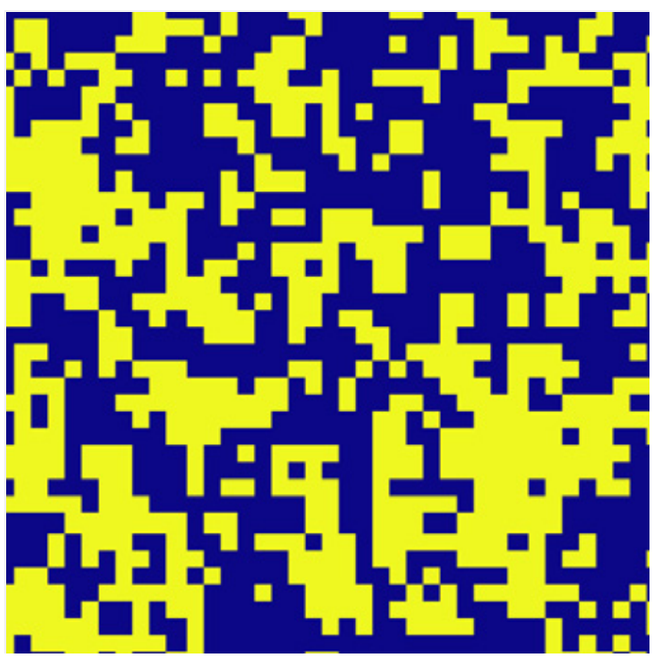
- Our classifier should exhibit all these high-level structures.

Locality and Symmetries

- **Locality** & **Symmetries**: basic principles underlying physical laws.
- Physics is governed by **local interactions**. Think about QFT, relativity, and statistical physics:



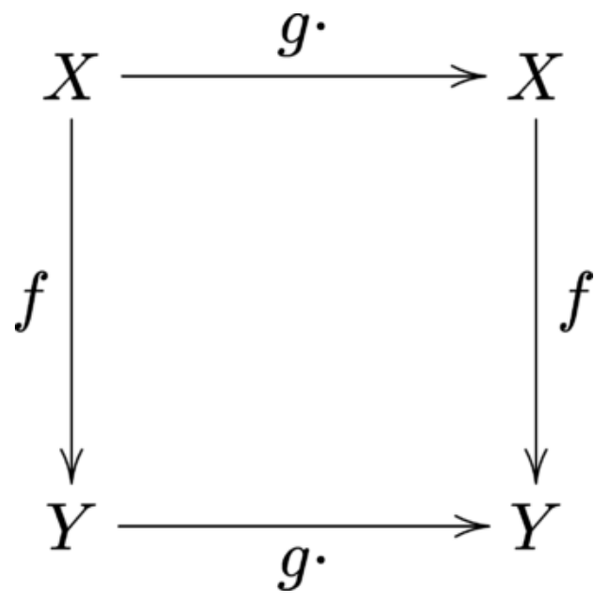
Vision tasks: local features matters, e.g.. whiskers, edge of a table, …



$$H = -J \sum_{\langle ij \rangle} S_i S_j$$

Locality and Symmetries

- **Symmetries** are at the heart of physics. For example, translation invariance allows to work in momentum space→less parameters
- In relativity and quantum field theory, Poincare-symmetry (translations, rotations, boosts) is essential.
- Gauge symmetries are ubiquitous in QFT and gravity. Equivariant CNNs (Cohen, Welling 2016).
- $f(x)$ is equivariant if we change the input in a particular way as $x' = g \cdot x$, the output changes in the same way: $f(g \cdot x) = g \cdot f(x)$:



How to make rotationally invariant CNN

1. Data Augmentation

- **Rotation Augmentation:** Rotating input images during training ensures the model learns to recognize patterns regardless of orientation.
- **Random Rotations:** Applying random rotations within a specific range can improve generalization.

2. Rotational Pooling

- **Max-Pooling Over Rotations:** Apply convolutional filters at different rotations and take the maximum response (or average) across all rotations.
- Example: **Oriented Response Networks (ORN)** use rotation-equivariant convolutional layers combined with rotation-invariant pooling.

3. Equivariant CNNs

- **Group Equivariant CNNs (G-CNNs):** Introduced by Cohen and Welling (2016), G-CNNs use convolutional layers designed to be equivariant to transformations from groups like rotations and translations.
- Example: G-CNNs with rotation groups (e.g., C_4, C_8) learn filters that rotate in discrete steps, maintaining equivariance and allowing for invariance through pooling.

4. Fourier Transform Approaches

- **Harmonic Networks (H-Nets):** These use the Fourier transform to capture rotational symmetries, making the network inherently rotation-invariant.
- By leveraging the Fourier domain, rotations become phase shifts, which are easier to handle with fewer parameters.

Convolutional Neural Networks

Fully Convolutional NN

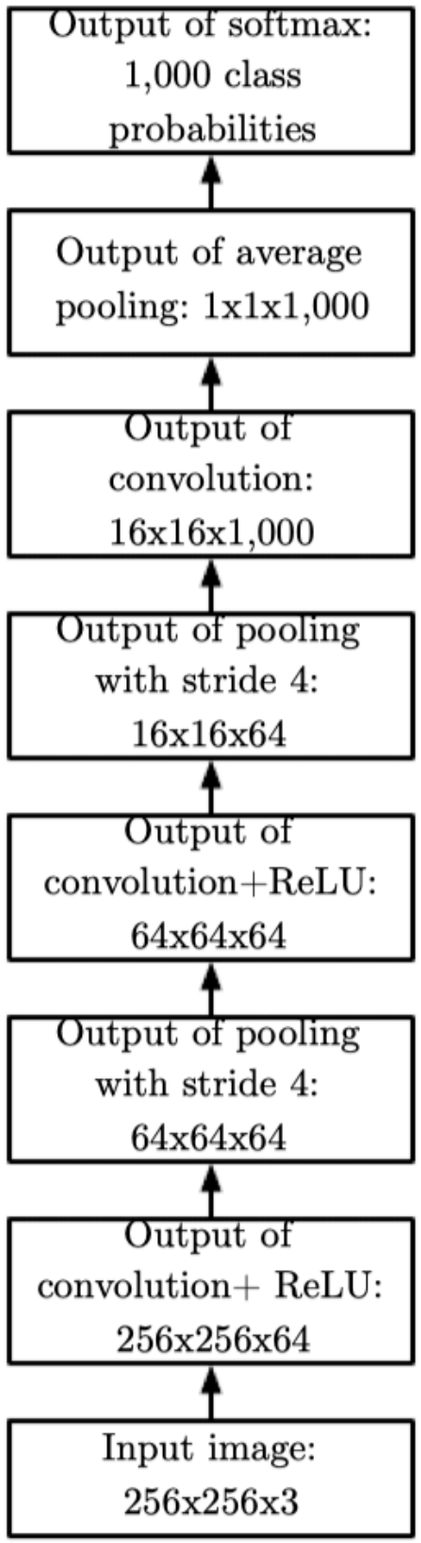
Dropping the fully connected layer

- A **fully convolutional neural network (FCNN)** is a type of neural network designed so that all layers are convolutional, without any fully connected layers. This architecture allows FCNNs to efficiently process inputs of varying sizes and is particularly well-suited for tasks like image segmentation and other field-to-field learning tasks.

- Features
  - **Convolutional Layers Only**: Unlike traditional CNNs, which often have fully connected layers at the end, FCNNs replace those layers with convolutional layers.
  - **Spatial Preservation**: FCNNs maintain spatial information throughout the network, which is crucial for pixel-level predictions.
  - **Efficient and Flexible**: They can take inputs of arbitrary sizes and produce outputs of corresponding dimensions.

- Famous paper: https://arxiv.org/abs/1411.4038 Fully Convolutional Networks for Semantic Segmentation (54000 citations)

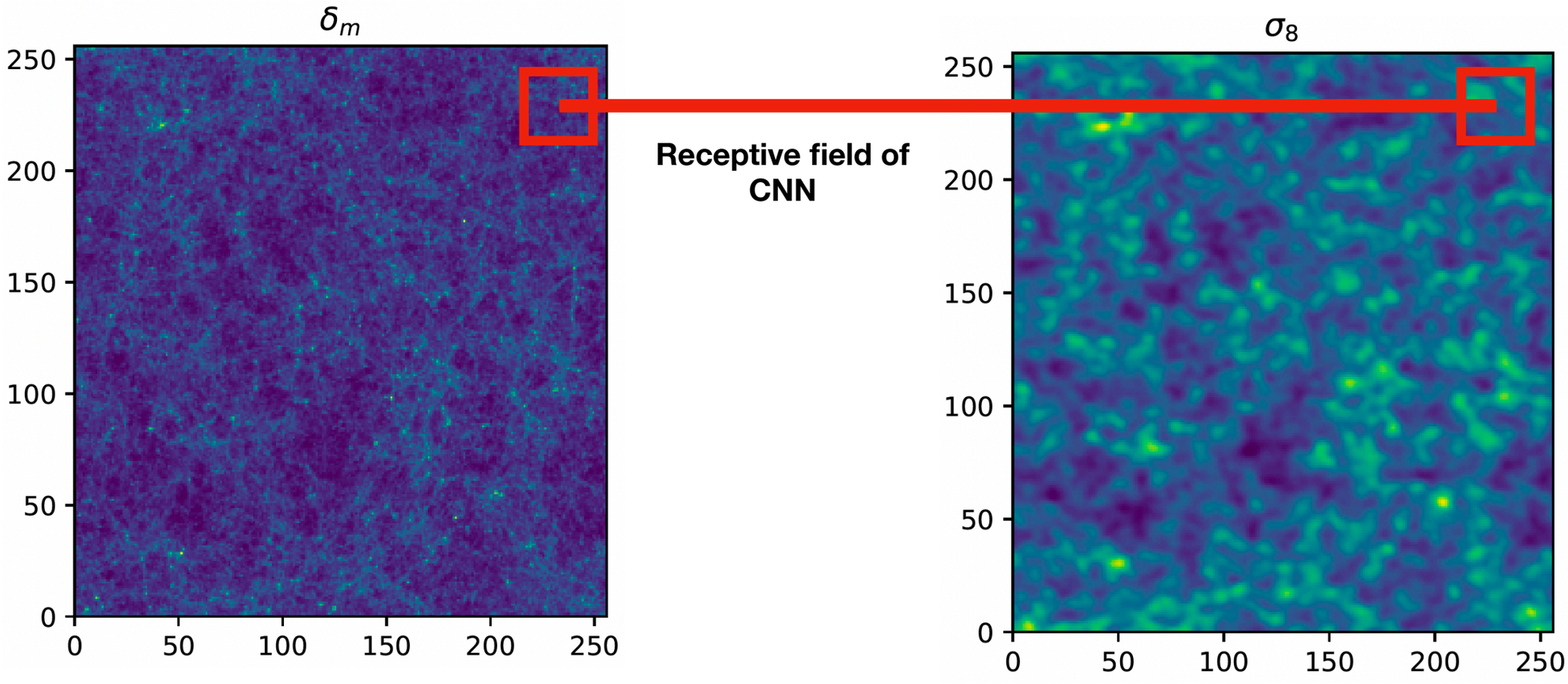
Example FCNN architecture

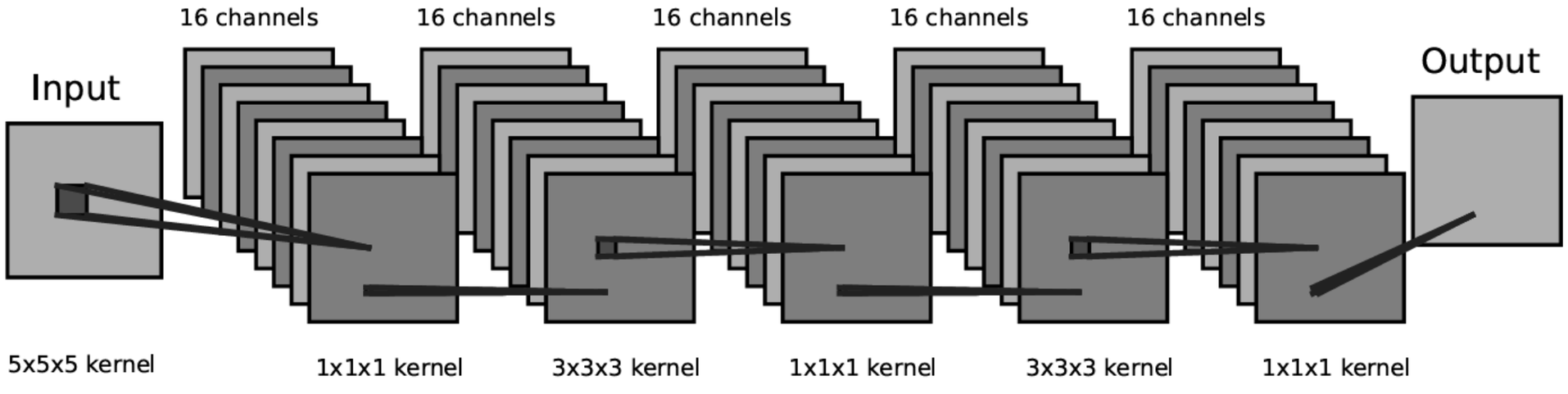


A convolutional network that does not have any fully connected weight layer. Instead, the last convolutional layer outputs one feature map per class. The model learns a map of how likely each class is to occur at each spatial location. Averaging a feature map down to a single value provides the argument to the softmax classifier at the top.

Image credit: https://towardsai.net/

Example from my research





Example use from my research

Robust Neural Network-Enhanced Estimation of Local Primordial Non-Gaussianity

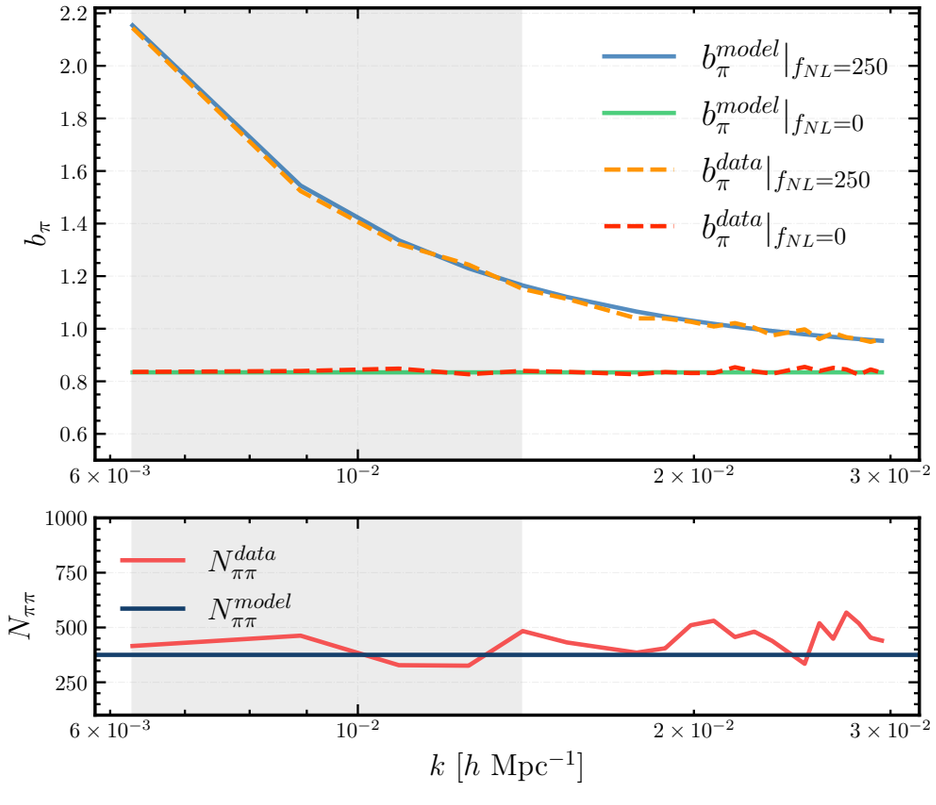
Utkarsh Giri* and Moritz Münchmeyer
University of Wisconsin-Madison, Madison, Wisconsin, USA

Kendrick M. Smith
Perimeter Institute for Theoretical Physics, Waterloo, Ontario, CA
(Dated: May 27, 2022)

When applied to the non-linear matter distribution of the universe, neural networks have been shown to be very statistically sensitive probes of cosmological parameters, such as the linear perturbation amplitude σ_8. However, when used as a "black box", neural networks are not robust to baryonic uncertainty. We propose a robust architecture for constraining primordial non-Gaussianity f_{NL}, by training a neural network to locally estimate σ_8, and correlating these local estimates with the large-scale density field. We apply our method to N-body simulations, and show that $\sigma(f_{NL})$ is 3.5 times better than the constraint obtained from a standard halo-based approach. We show that our method has the same robustness property as large-scale halo bias: baryonic physics can change the normalization of the estimated f_{NL}, but cannot change whether f_{NL} is detected.

Keywords: non-Gaussianity, neural networks, large-scale structure

https://arxiv.org/pdf/2205.12964



Boundary conditions and FCNNs

- The **FCNN output will have the same size as the input if we use zero padding or periodic (“wrap around”, “circular”) boundary conditions.**
- Periodic boundary conditions are common in physics simulations, but less common in every day life.
- If we want to use “valid” convolutions, and we want large receptive fields, the output image would be much smaller, e.g we could only segment the center of the image.

The U-Net architecture

- The U-Net is an **encoder-decoder architecture where the earlier representations are concatenated to the later ones**.
- The U-Net's encoder-decoder structure with skip connections is great for **capturing both global context and fine-grained local details**, making it ideal for segmentation or reconstruction tasks.
- Note that the U**-Net is completely convolutional,** so after training, it can be run on an image of any size.

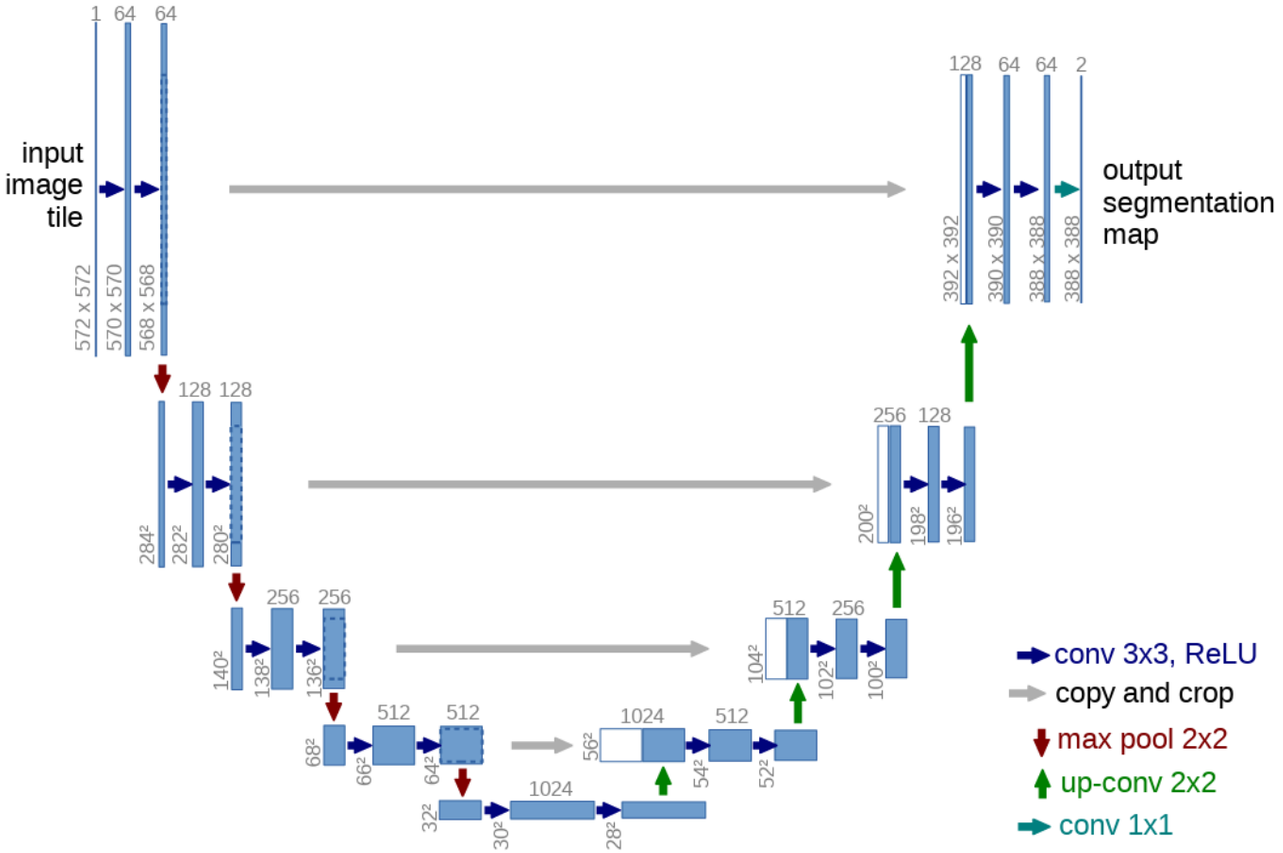


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

https://arxiv.org/pdf/1505.04597
U-Net: Convolutional Networks for Biomedical Image Segmentation (100000 citations)

The U-Net architecture

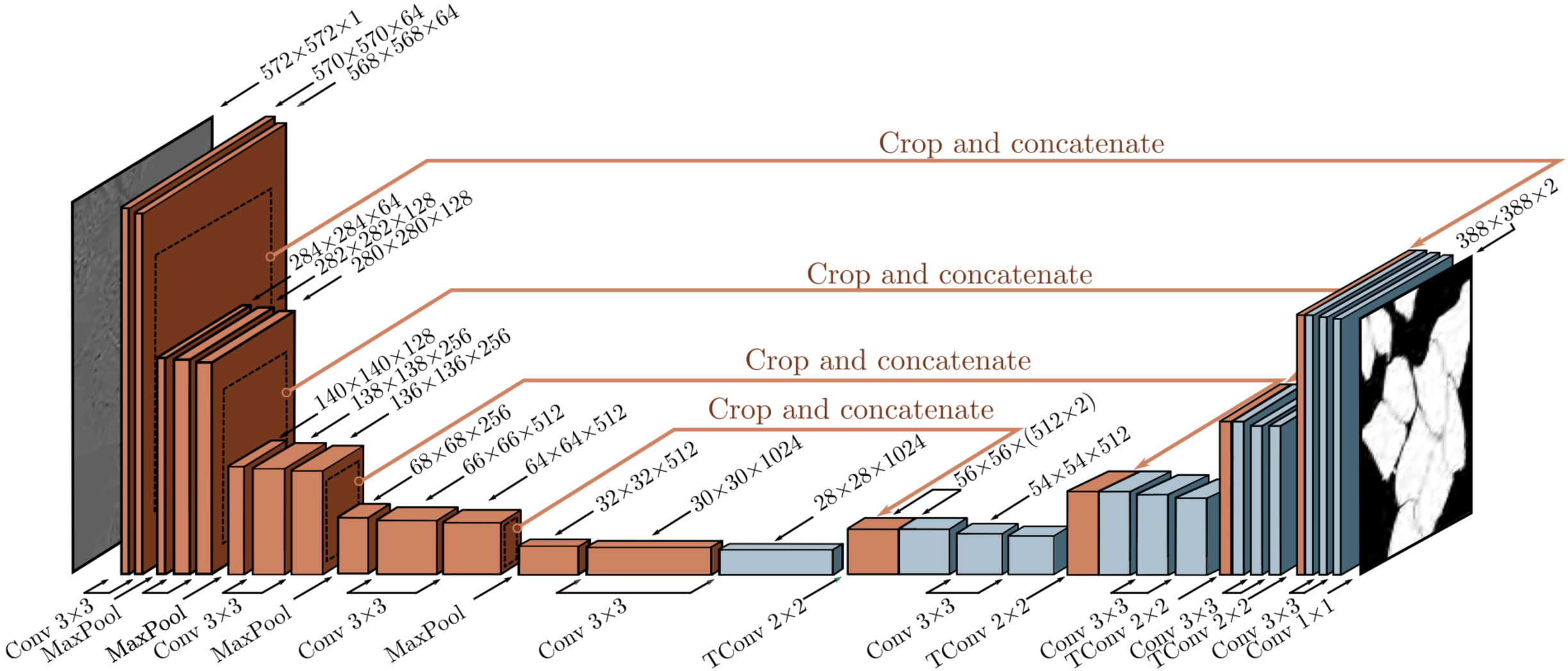
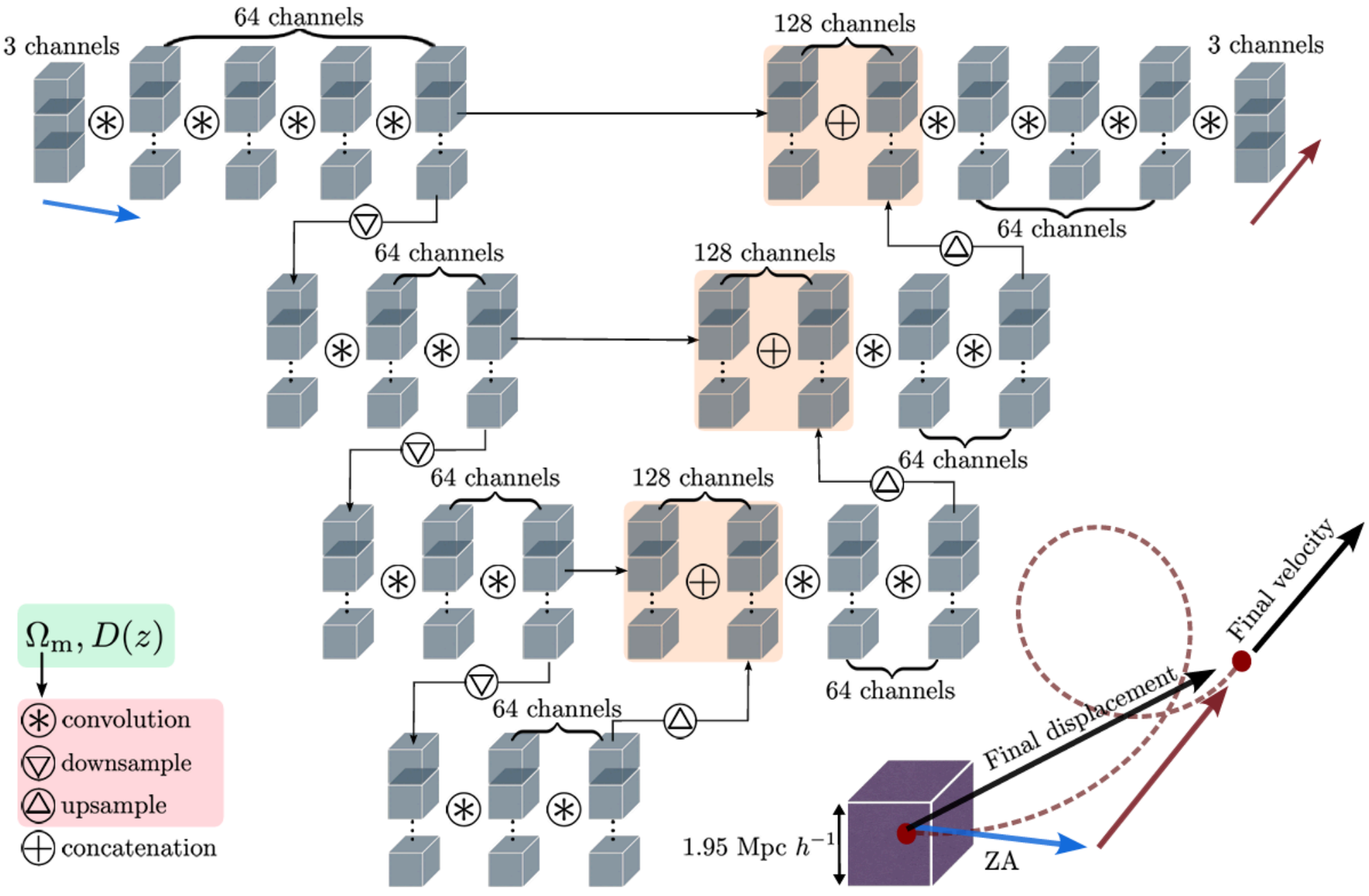


Figure 11.10 U-Net for segmenting HeLa cells. The U-Net has an encoder-decoder structure, in which the representation is downsampled (orange blocks) and then re-upsampled (blue blocks). The encoder uses regular convolutions, and the decoder uses transposed convolutions. Residual connections append the last representation at each scale in the encoder to the first representation at the same scale in the decoder (orange arrows). The original U-Net used "valid" convolutions, so the size decreased slightly with each layer, even without downsampling. Hence, the representations from the encoder were cropped (dashed squares) before appending to the decoder. Adapted from Ronneberger et al. (2015).

U-Nets and V-Nets in cosmology

Example: https://arxiv.org/abs/2408.07699 Field-level Emulation of Cosmic Structure Formation with Cosmology and Redshift Dependence



Convolutional Neural Networks

Data Normalization

Before training we normalize the data

Normalizing data for a machine learning task is essential to ensure faster and more stable training.

1. Channel-wise Normalization (Standard Approach)

This is the most common method where each channel is normalized independently:

$$x' = \frac{x - \mu_c}{\sigma_c}$$

- μ_c and σ_c are the mean and standard deviation of each channel computed across the entire dataset.
- This ensures that each channel has zero mean and unit variance.

2. Min-Max Scaling

If your regression targets are sensitive to the scale of input features, consider Min-Max scaling:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

- This scales values to a range $[0, 1]$ or $[-1, 1]$.
- For multi-channel data, apply this to each channel individually or to the entire dataset depending on the scenario.

When to Normalize

Always compute normalization statistics using the training dataset only.

Apply the same normalization to both training and test data.

- **Input:** Always normalize.
- **Target:** Normalize if it's large or has a wide range, but remember to invert the normalization during inference.

Convolutional Neural Networks

Using pre-trained models

Fine-tuning pre-trained models

- For many tasks it is better not to train a CNN from scratch, but to fine-tune parameters from a pertained model.
- Finetuning a pre-trained CNN is a powerful technique to leverage existing knowledge from large datasets (like ImageNet) and apply it to your specific task.
- This involves the following steps:
  - Choose a Pre-trained Mode

```
from torchvision import models
model = models.resnet18(pretrained=True)  # Example: ResNet18 pretrained
```

  - Modify the Output Layer for your target (e.g. regression)
  - Freeze Some Layers (optional, reduces computation time)
    - Usually we freeze lower level layers and fine tune higher level ones
  - Train the model parameters on your data set

Convolutional Neural Networks

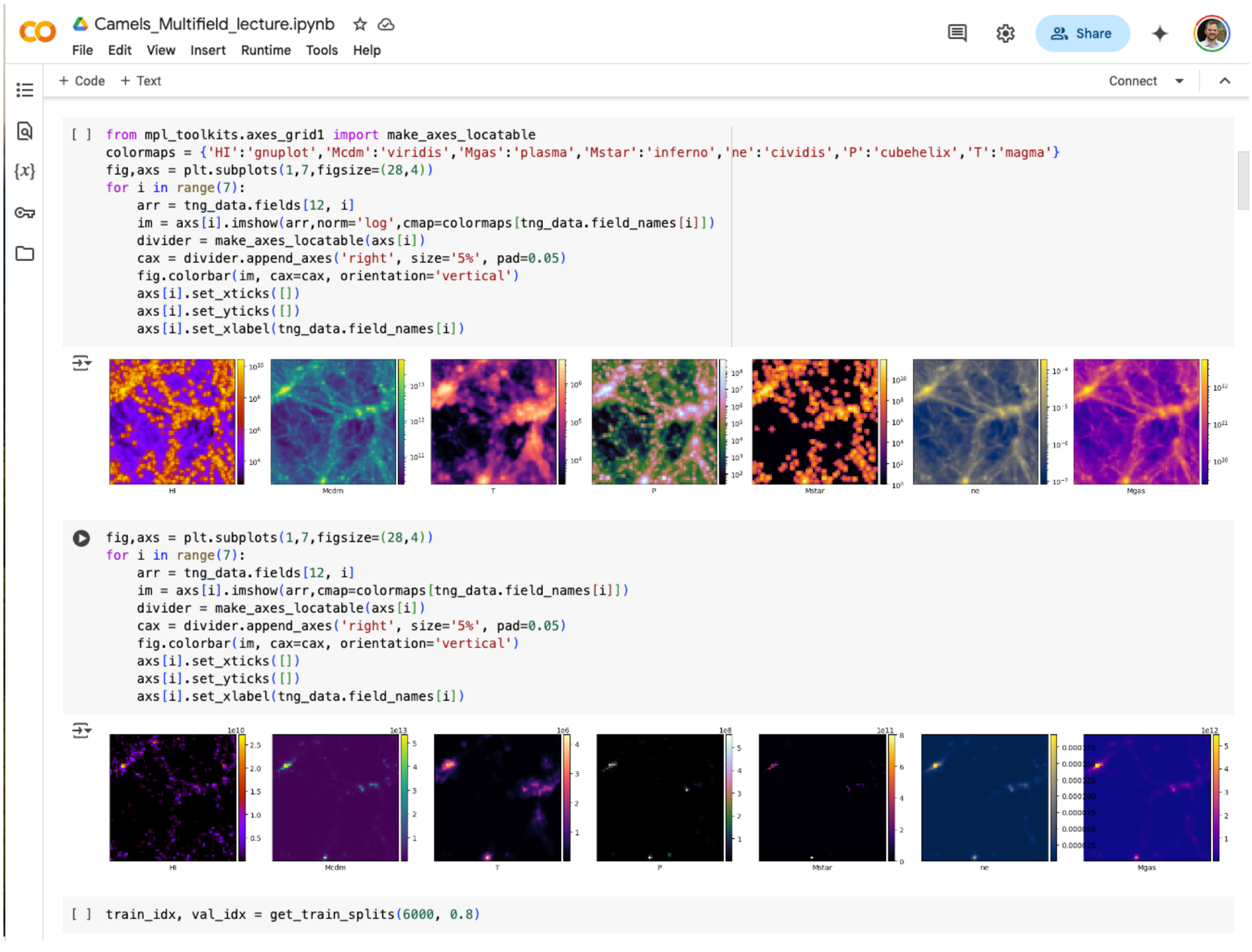
Our CNN project: CAMELS simulations

CAMELS Multi-field dataset

- We will use a real research dataset to explore both regression and field-to-field learning with CNNs.
- The dataset comes from here:
  - https://www.camel-simulations.org/
  - https://arxiv.org/abs/2201.01300 The CAMELS project: public data release
    - The Cosmology and Astrophysics with MachinE Learning Simulations (CAMELS) project was developed to combine cosmology with astrophysics through thousands of cosmological hydrodynamic simulations and machine learning. CAMELS contains 4,233 cosmological simulations, 2,049 N-body and 2,184 state-of-the-art hydrodynamic simulations that sample a vast volume in parameter space. In this paper we present the CAMELS public data release, describing the characteristics of the CAMELS simulations and a variety of data products generated from them, including halo, subhalo, galaxy, and void catalogues, power spectra, bispectra, Lyman-α spectra, probability distribution functions, halo radial profiles, and X-rays photon lists. We also release over one thousand catalogues that contain billions of galaxies from CAMELS-SAM: a large collection of N-body simulations that have been combined with the Santa Cruz Semi-Analytic Model. We release all the data, comprising more than 350 terabytes and containing 143,922 snapshots, millions of halos, galaxies and summary statistics.
- We will use the CAMELS multi field dataset
  - https://arxiv.org/abs/2109.10915 The CAMELS Multifield Dataset: Learning the Universe's Fundamental Parameters with Artificial Intelligence
    - We present the Cosmology and Astrophysics with MachinE Learning Simulations (CAMELS) Multifield Dataset, CMD, a collection of hundreds of thousands of 2D maps and 3D grids containing many different properties of cosmic gas, dark matter, and stars from 2,000 distinct simulated universes at several cosmic times. The 2D maps and 3D grids represent cosmic regions that span ∼100 million light years and have been generated from thousands of state-of-the-art hydrodynamic and gravity-only N-body simulations from the CAMELS project. Designed to train machine learning models, CMD is the largest dataset of its kind containing more than 70 Terabytes of data. In this paper we describe CMD in detail and outline a few of its applications. We focus our attention on one such task, parameter inference, formulating the problems we face as a challenge to the community.

Colab

- The rest of this lecture will be using a Colab notebook, prepared by our TA Yurii Kvasiuk.



Course logistics

- **Reading for this lecture:**
  - https://udlbook.github.io/udlbook/ (Simon Prince - Understanding Deep Learning)
  - deeplearningbook.com