A brief introduction to Computer Vision

Oxford Machine Learning Summer School, 2022

Ishan Misra

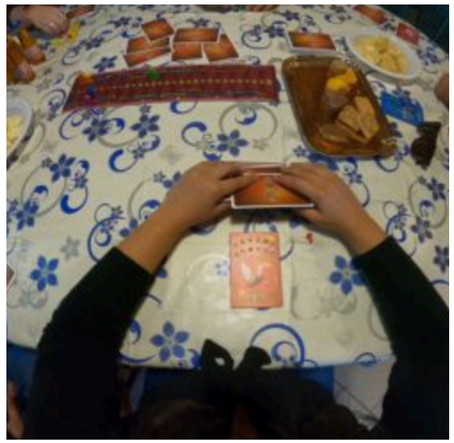
FAIR, Meta

Twitter: @imisra_
https://imisra.github.io/

Computer Vision

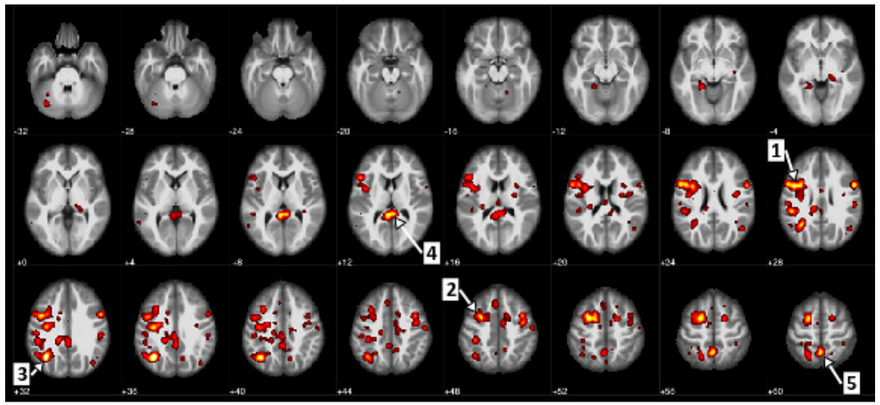
Creating models that can “understand” visual inputs

“Natural” Images

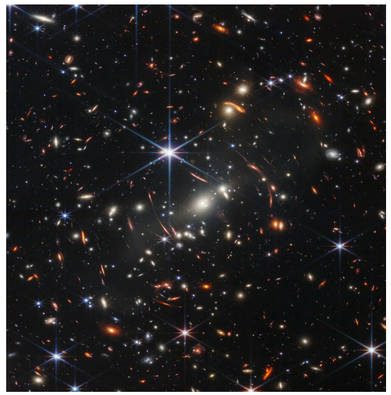


“Natural” Videos

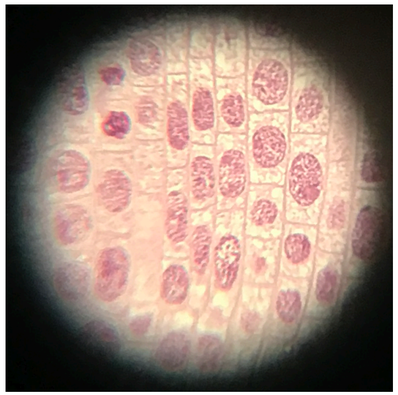
3D data



Medical domain



Astronomy



Biology

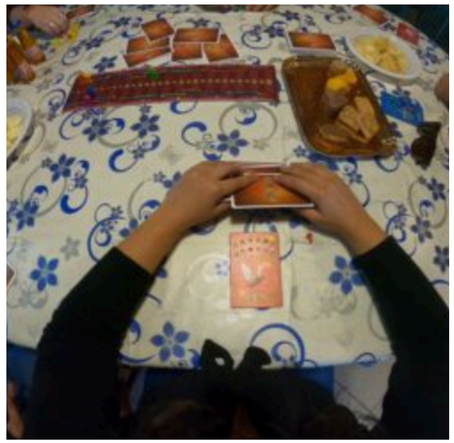
Aerial

Art

Types of Visual Data

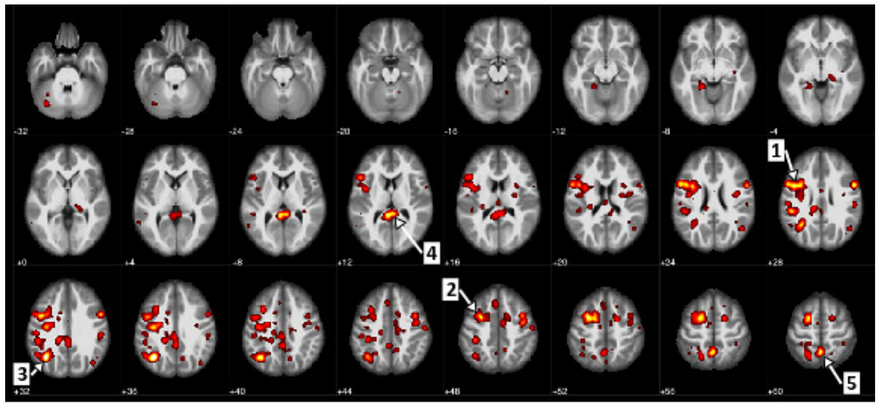
Visual data is diverse

"Natural" Images

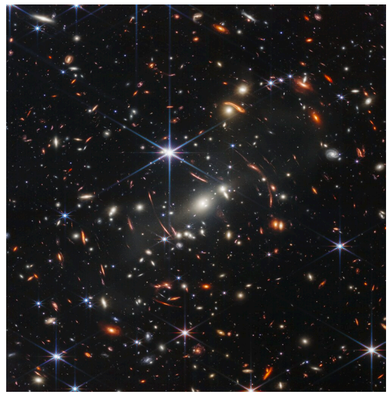


"Natural" Videos

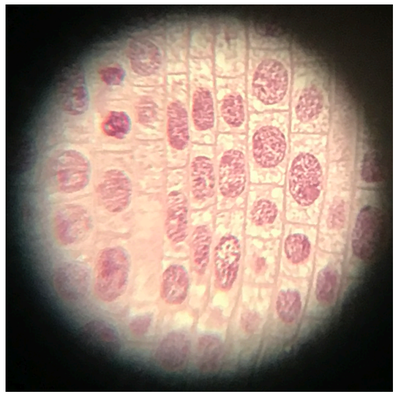
3D data



Medical domain



Astronomy



Biology

Aerial

Art

Types of Visual Tasks

Broadly

1. Recognition: Recognize visual concepts in an image
2. Geometric: Predict geometric properties (shape, depth)
3. Generation: Generate visual data

Recognition: Image Classification

Input: Image

Output: Predict one of “K” classes

Dog

Tree

Table

Cat

Horse

One of the most popular tasks.

ImageNet dataset involves classifying 1 million images into 1000 classes

Recognition: Object Detection

Input: Image

Output: Predict one of “K” classes AND draw a box around it

Recognition: Image Segmentation

Input: Image

Output: For every pixel, predict which one of “K” classes does it belong to

Recognition: Dense tasks

Image credit: Detectron2

Recognition: Dense tasks

Image credit: DensePose - Neverova et al., 2019

Image Classification — Still very popular!

Input: Image

Output: Predict one of “K” classes

Dog

Tree

Table

Cat

Horse

Performance on Image Classification translates well to different recognition tasks

Challenges in Recognition

Different lighting, viewing angle, seasons




Challenges in Recognition

Within-concept variance




Challenges in Recognition

Clutter/Multiple concepts

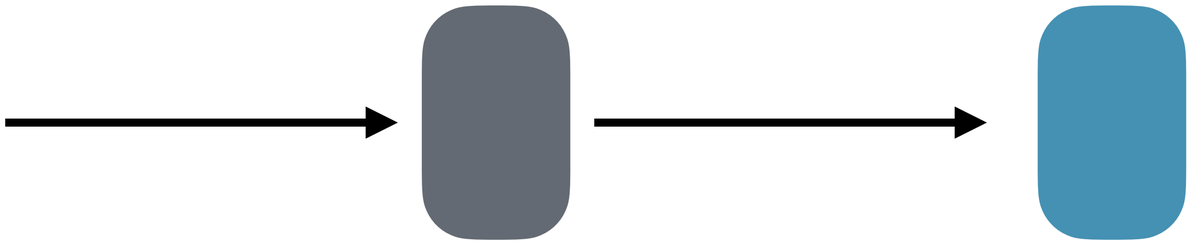
How do we address this?

Raw pixel values are not reliable

Need some reliable features from images

Recognize the features

Image —> Image Features —> Classifier

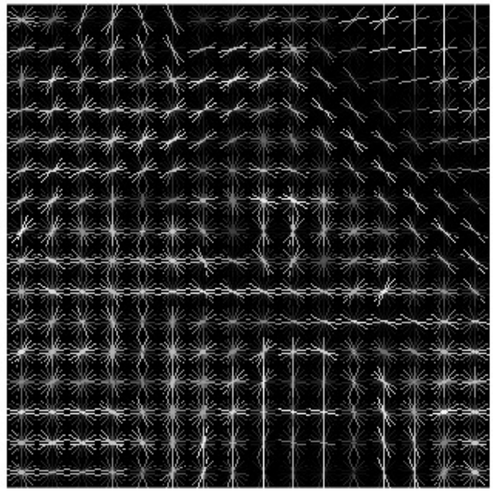


Image

Feature

Classifier

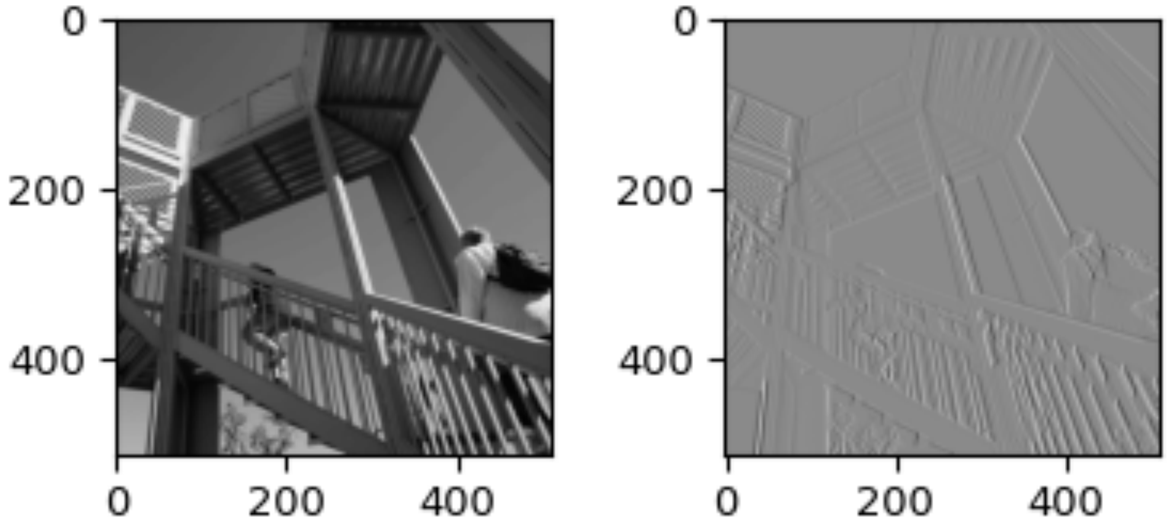
Hand-designed Image Features



HOG Features

Dalal & Triggs

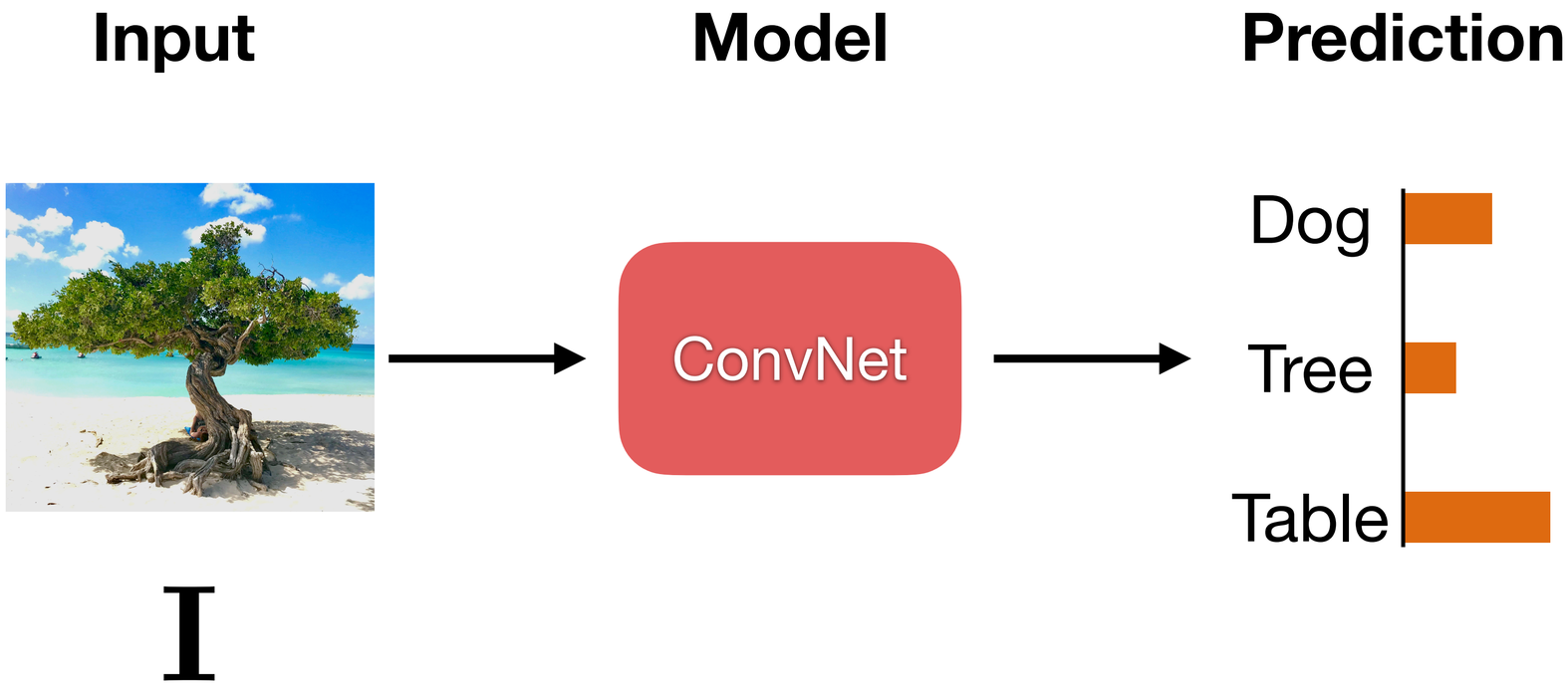
Image credit: VLFeat



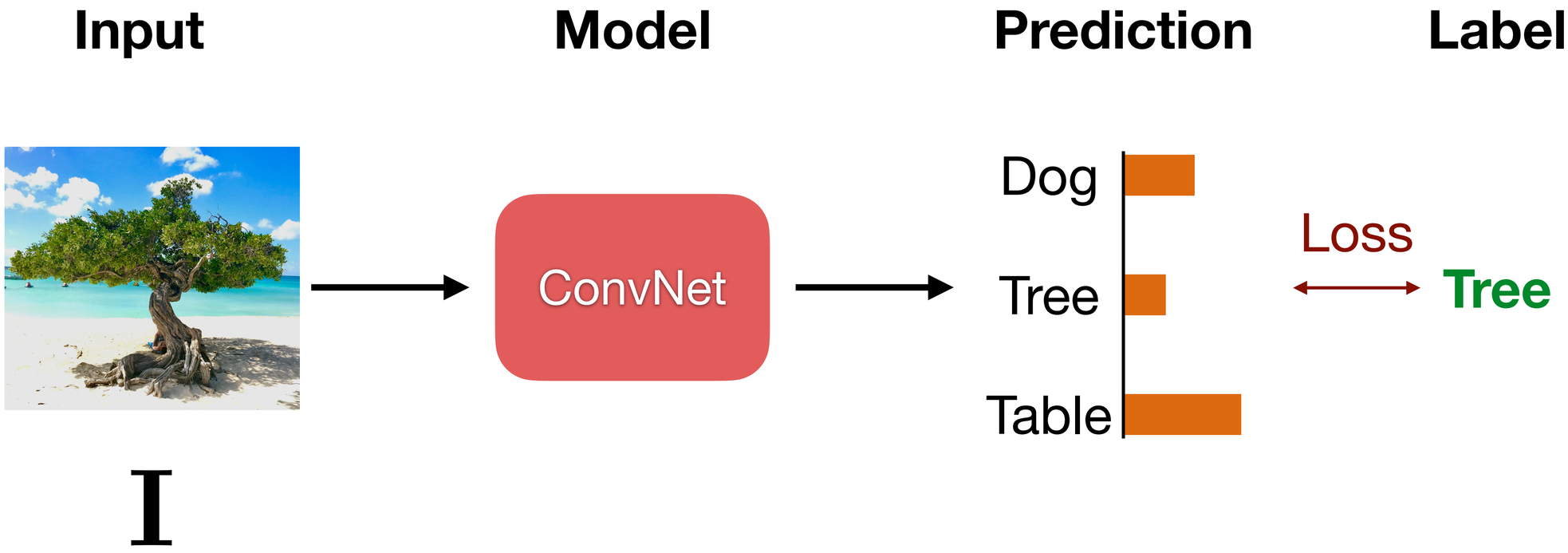
Edge Features

Image credit: Scipy

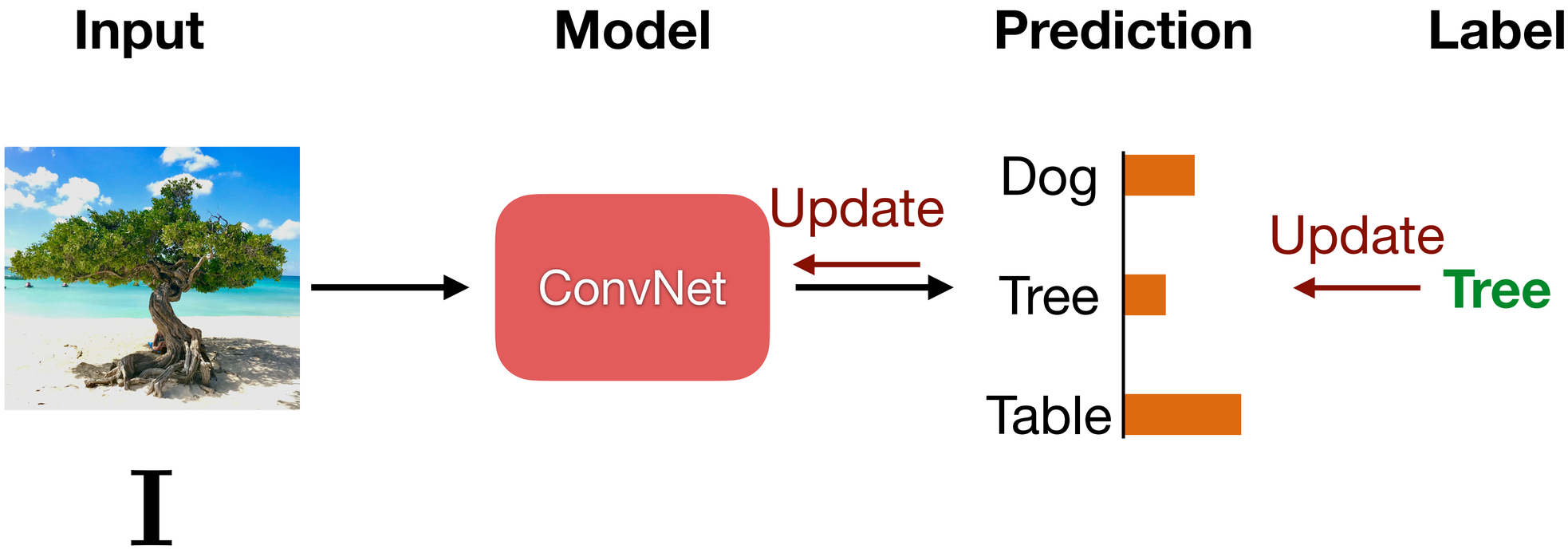
Deep Learning: Learning features and classifiers jointly



Deep Learning: Learning features and classifiers jointly

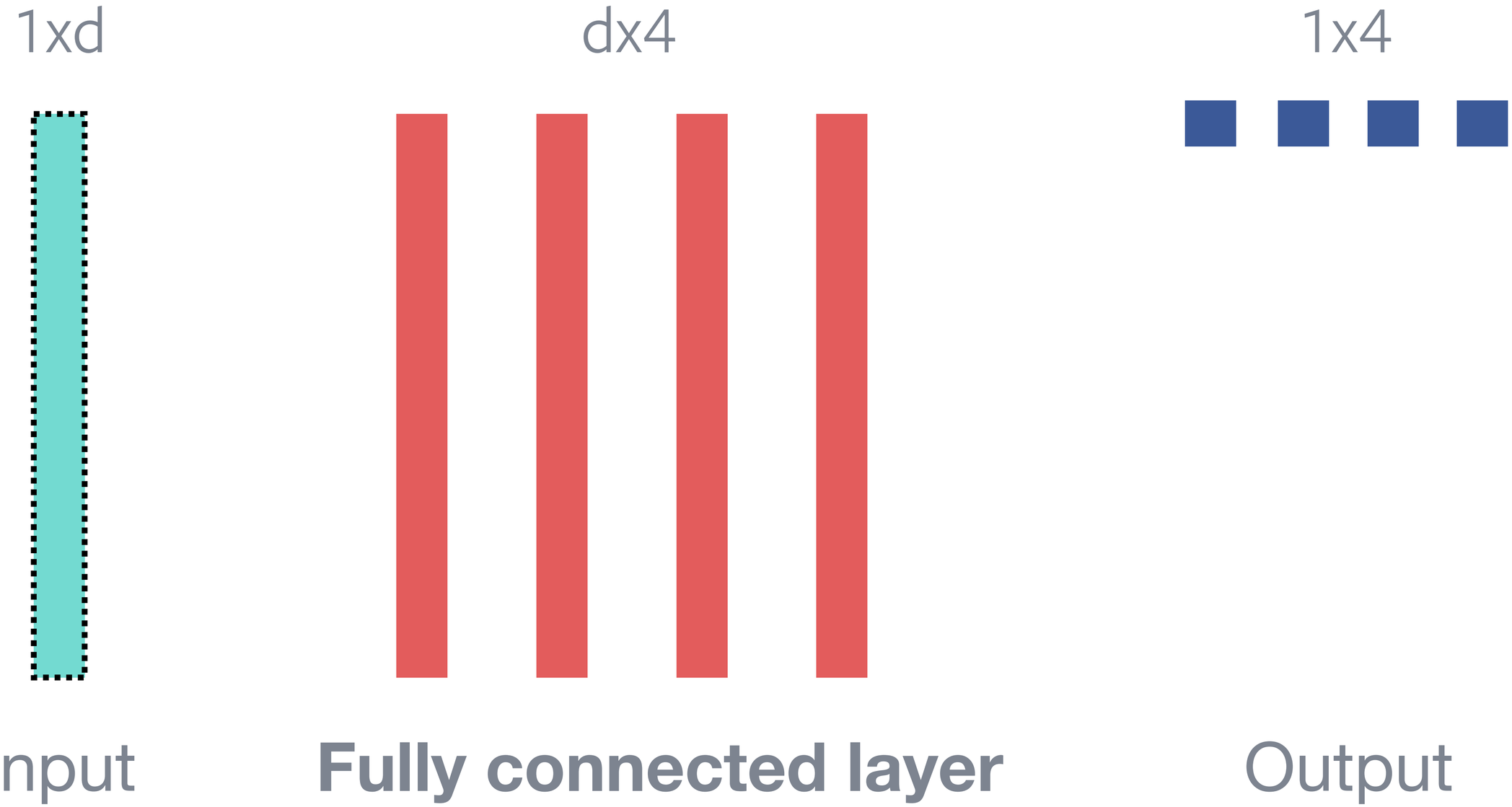


Deep Learning: Learning features and classifiers jointly

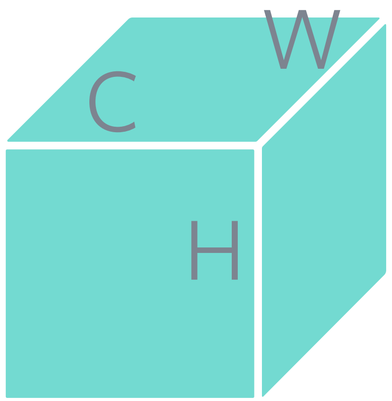


Convolutional Networks

Recap: Fully connected layer



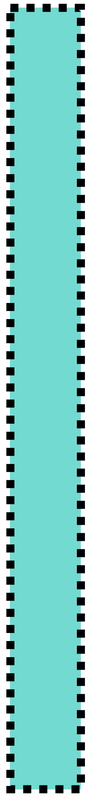
Flatten Images



Image

Flatten

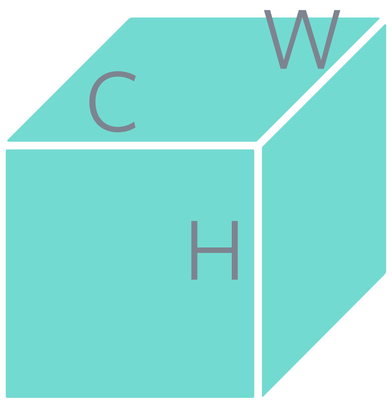
1xd



Input

d=C*H*W

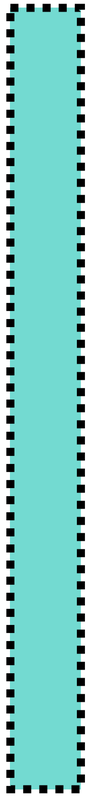
So flatten images?



Image

Flatten

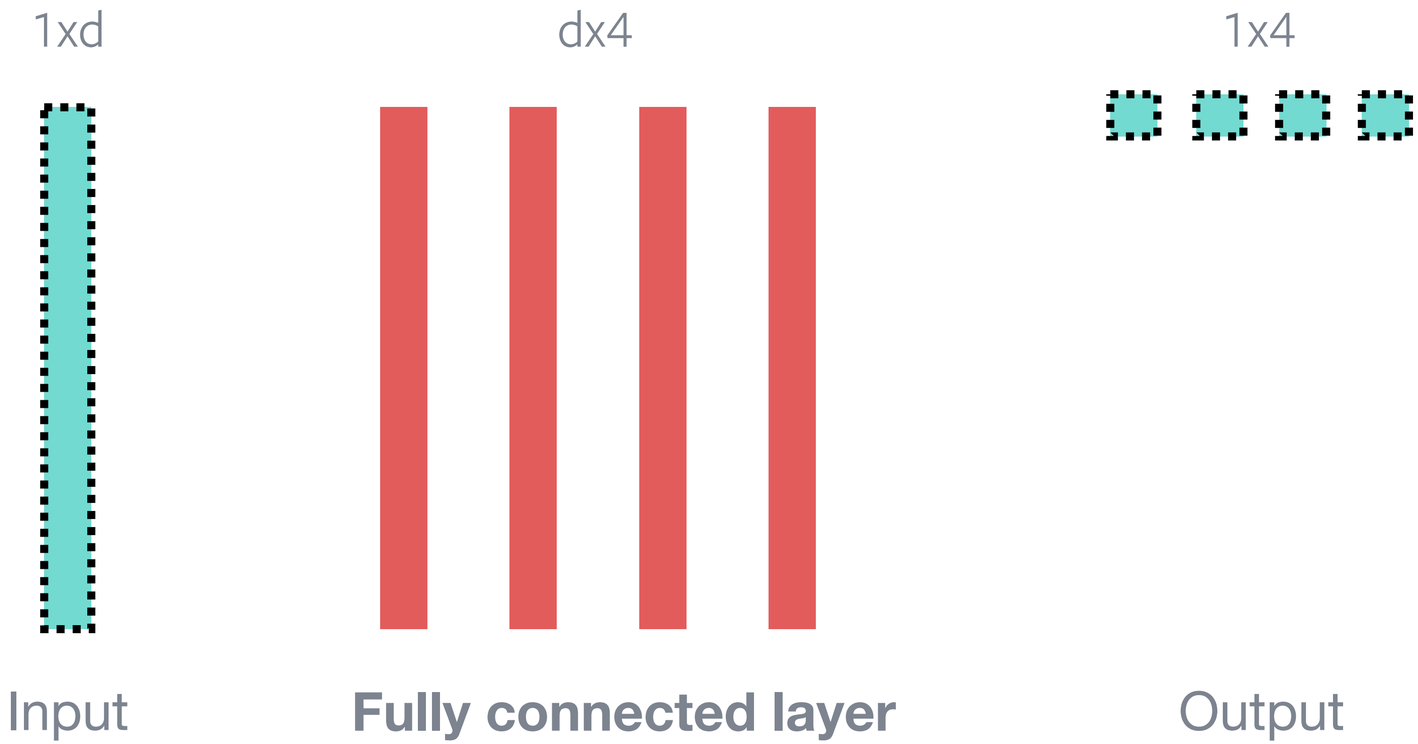
1xd



Input

d=C*H*W

Flatten Images —> Full connected layer



$$d = C*H*W$$

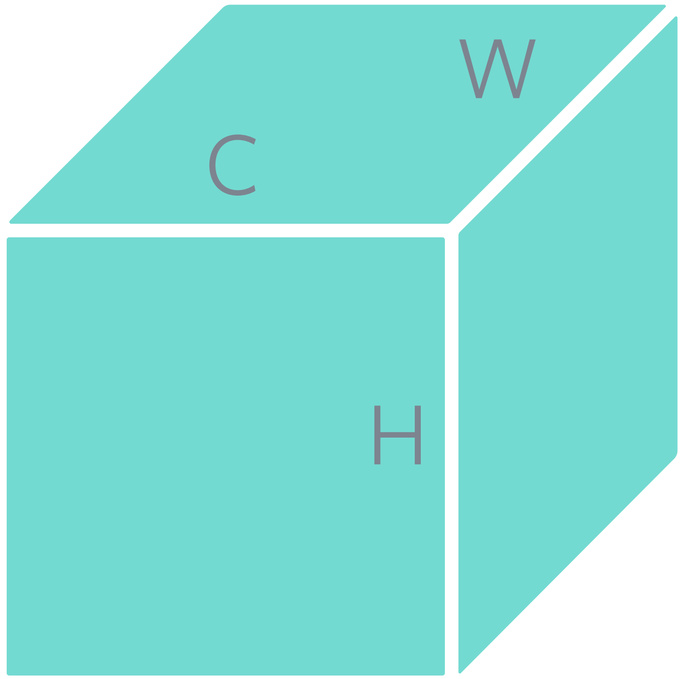
$$= 3*12*10^6$$

$$= 36 \text{ million}$$

Images are a 2D signal

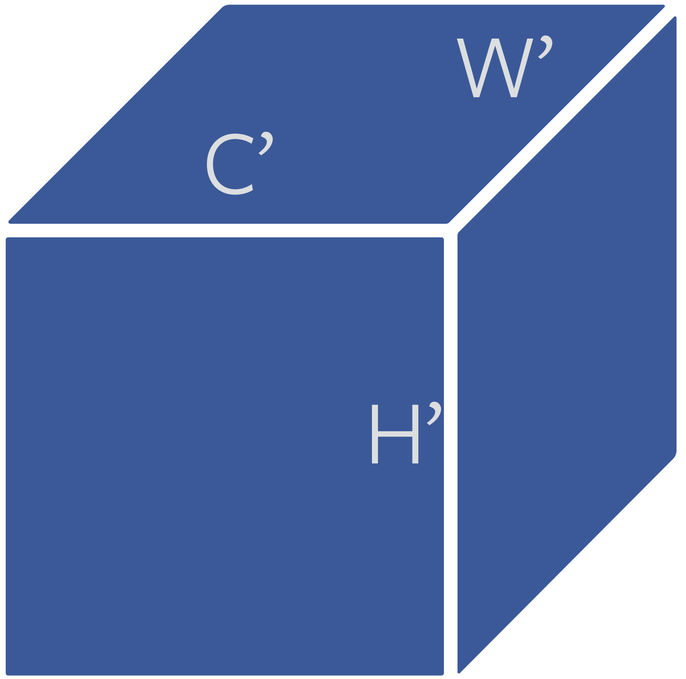
Nearby patches from an image have similar information

Convolution: basic operation in a ConvNet



Input Image

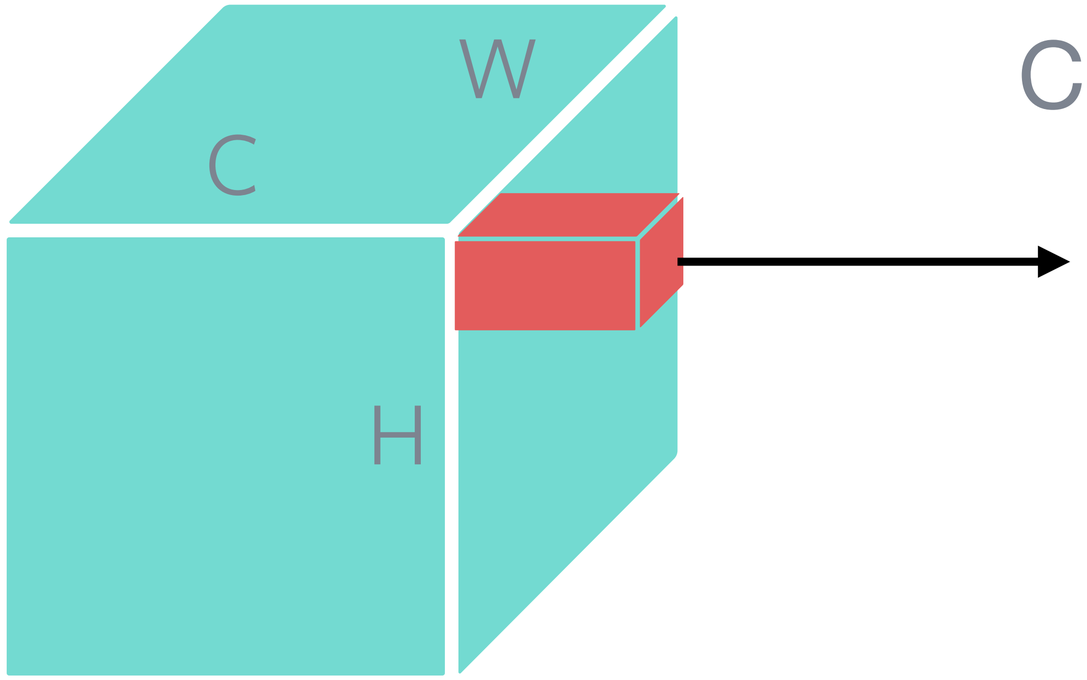
Kernel



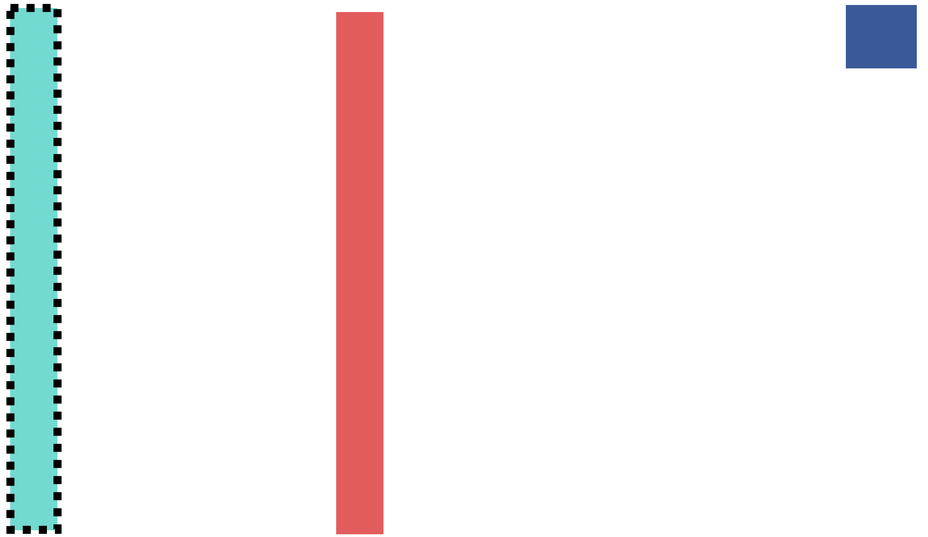
Output

Convolution: basic operation in a ConvNet

“Place” the kernel on the image
Flatten that portion of the image
Compute a single number



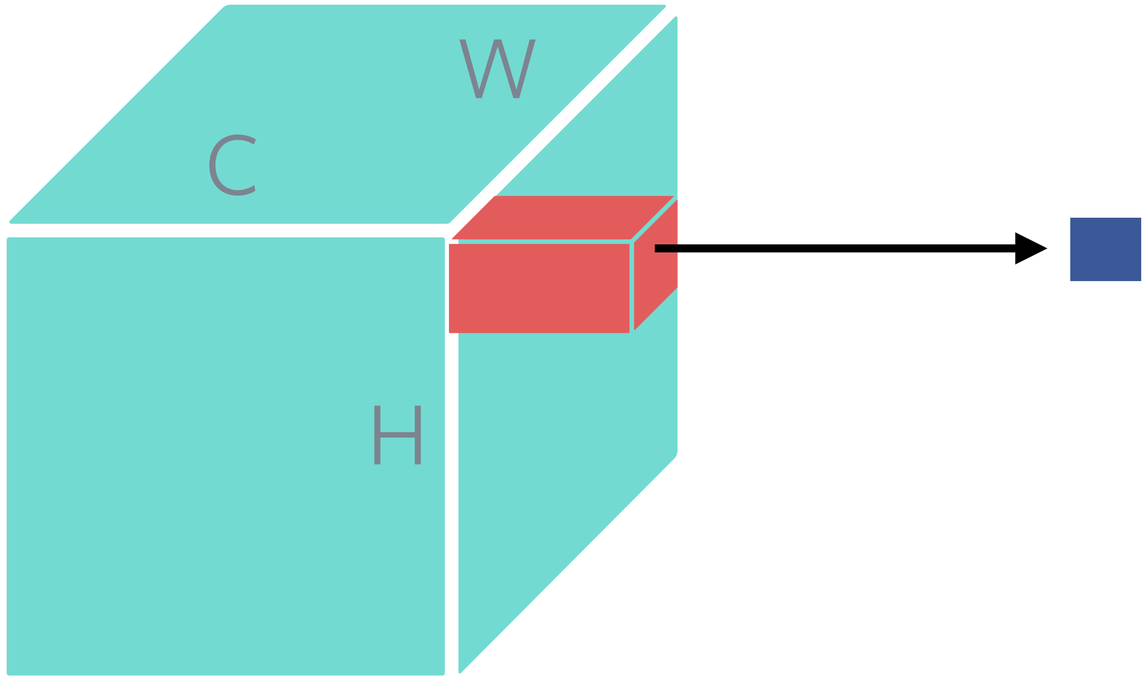
Input Image



Patch Kernel Output

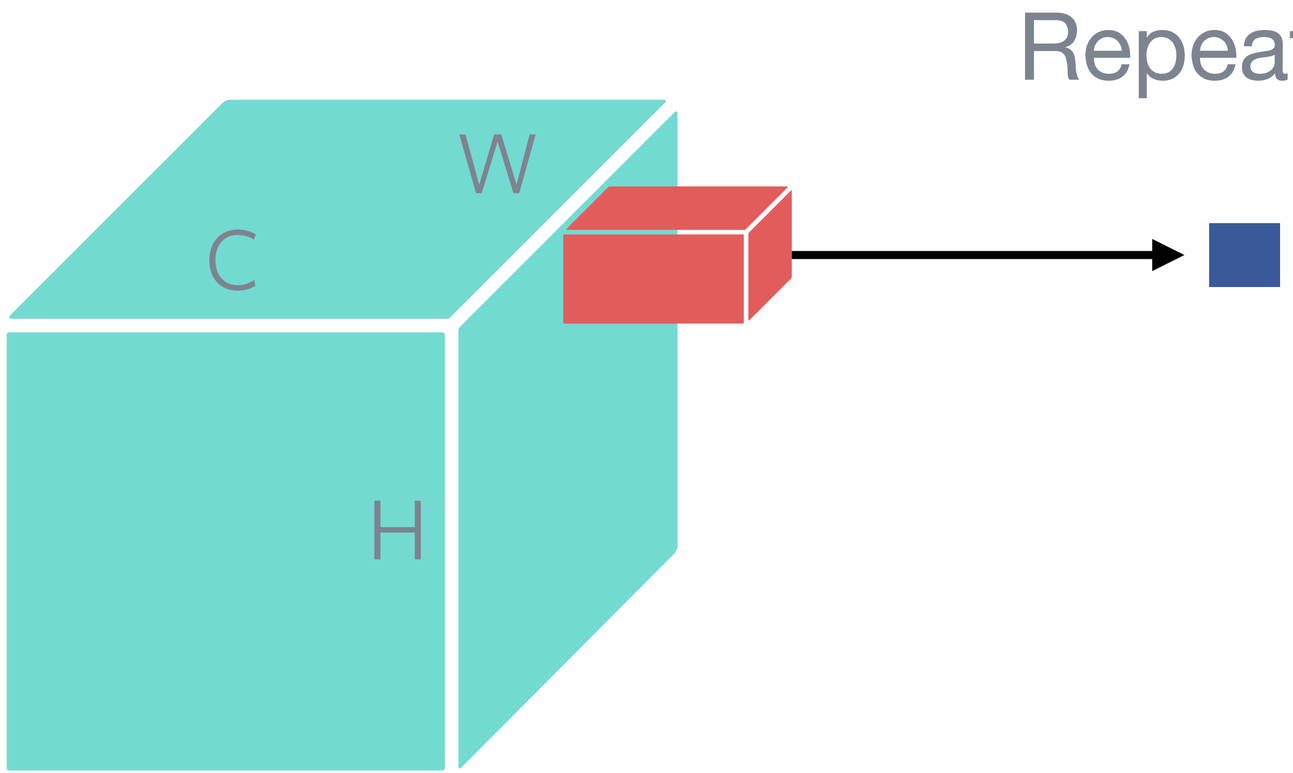
Convolution: basic operation in a ConvNet

Each placement gives one output



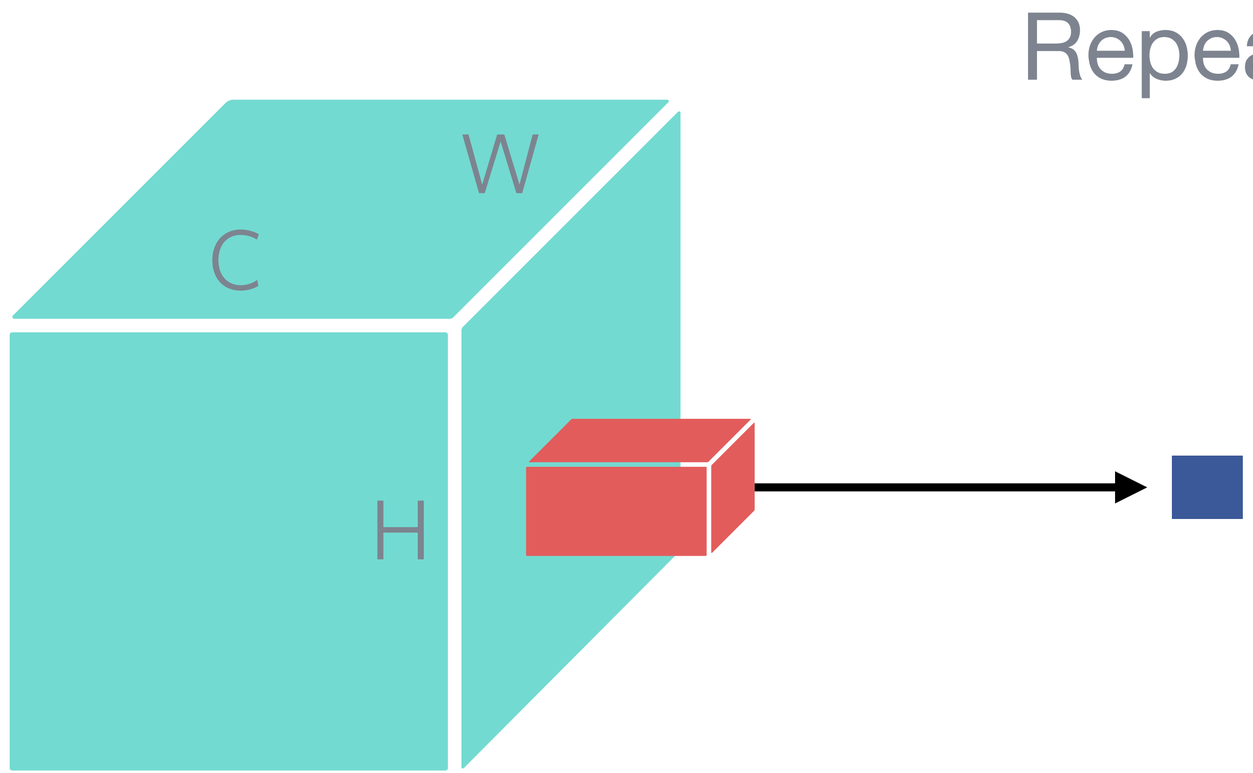
Input Image

Convolution: basic operation in a ConvNet

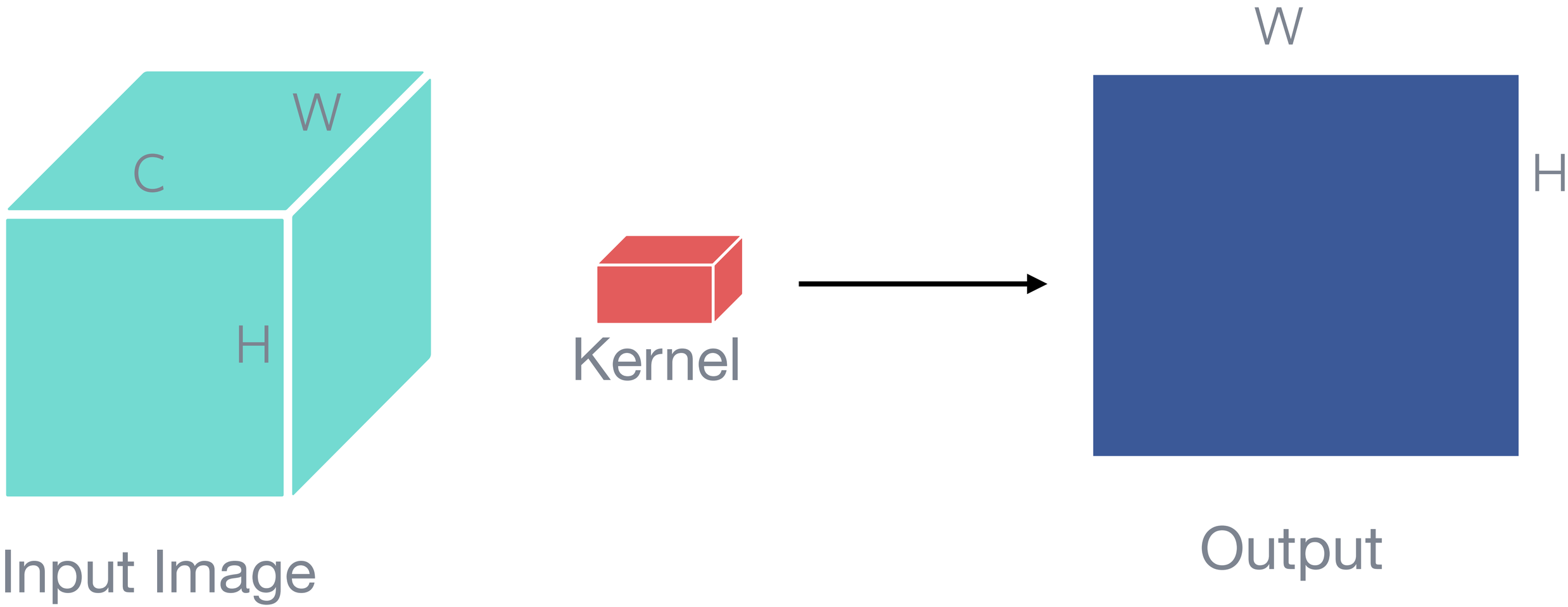


Input Image

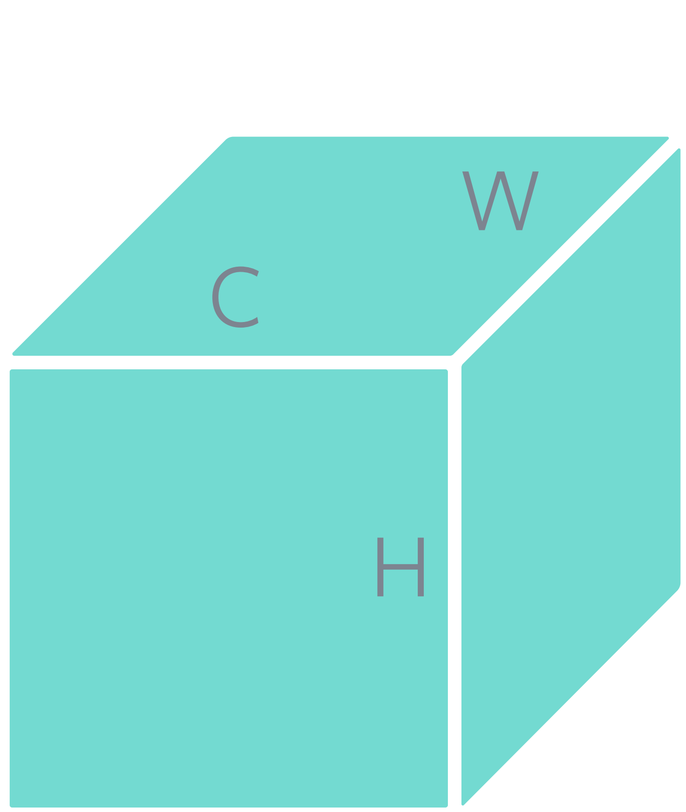
Convolution: basic operation in a ConvNet



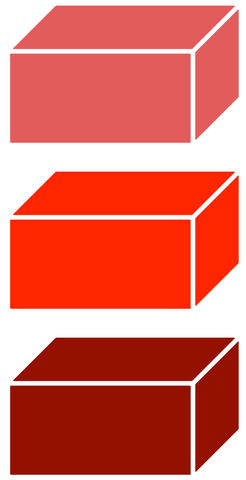
Convolution: single kernel



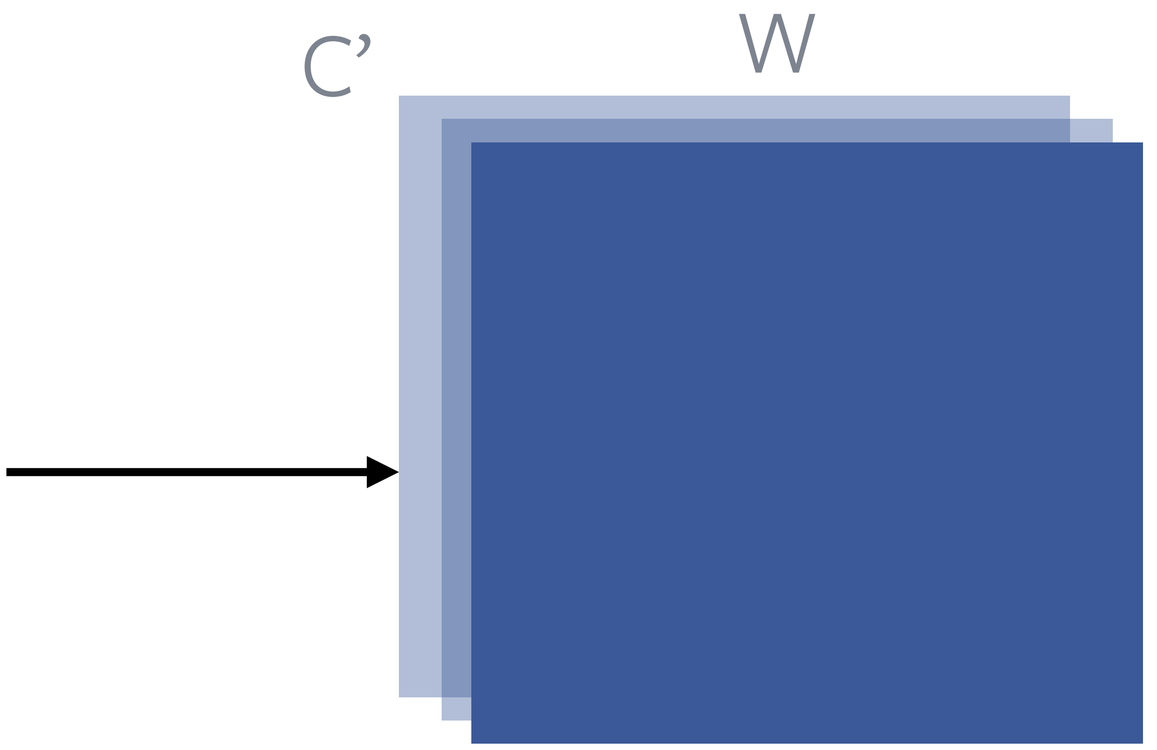
Convolution: multiple kernels



Input Image

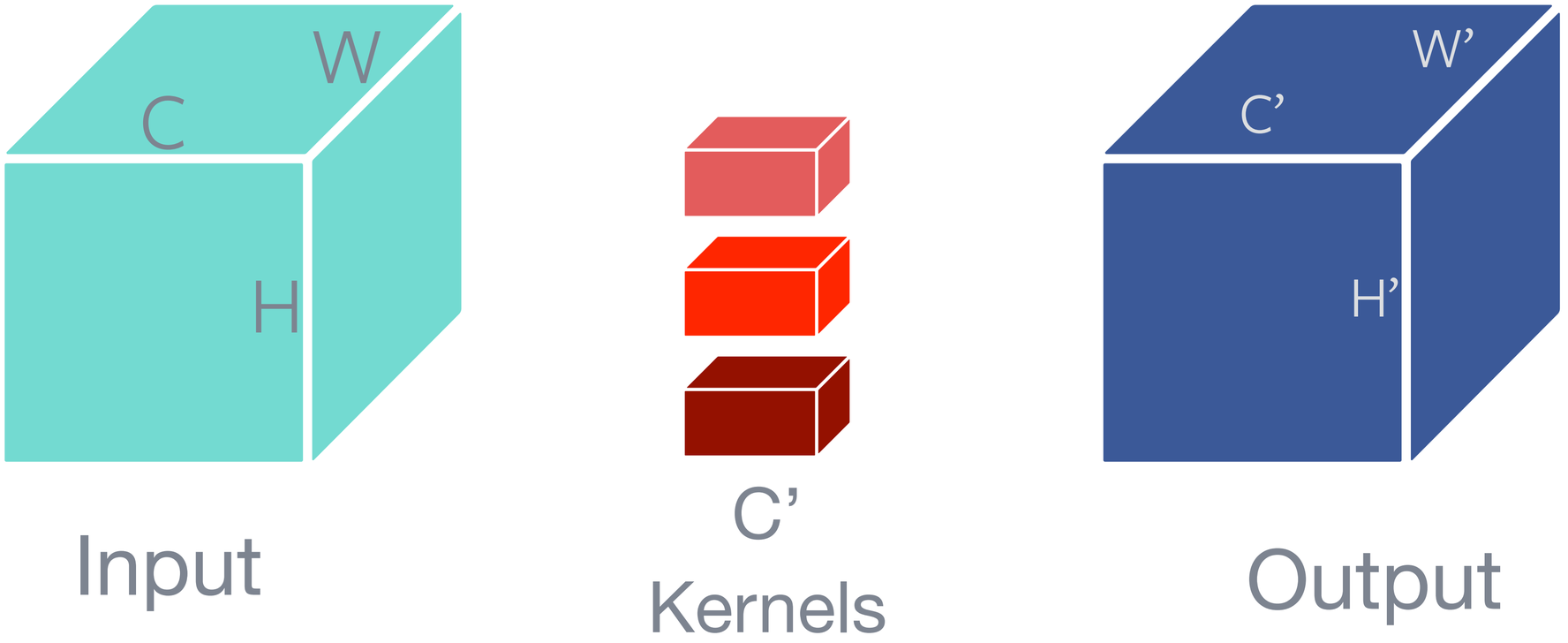


C’ Kernels



Output

Convolution: Preserves spatial structure of the input



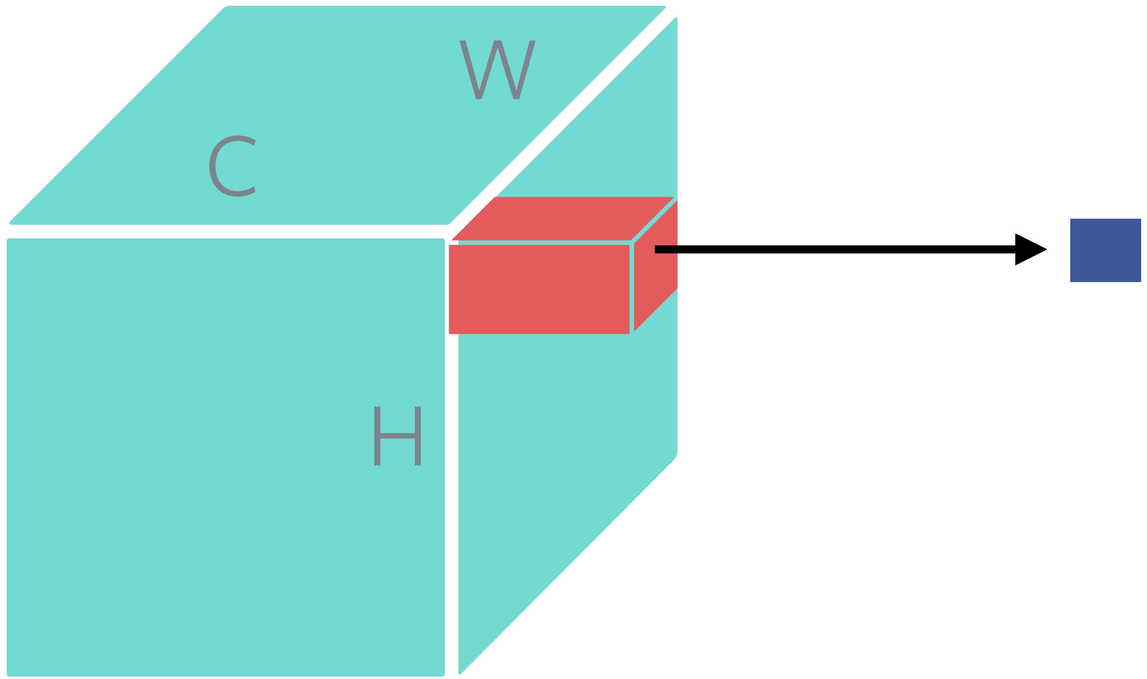
We “share” the parameters across different spatial locations

Each kernel is of the shape: C x h x w

Reduces parameters vs. fully connected layers

Stride: a way to change spatial dimensions

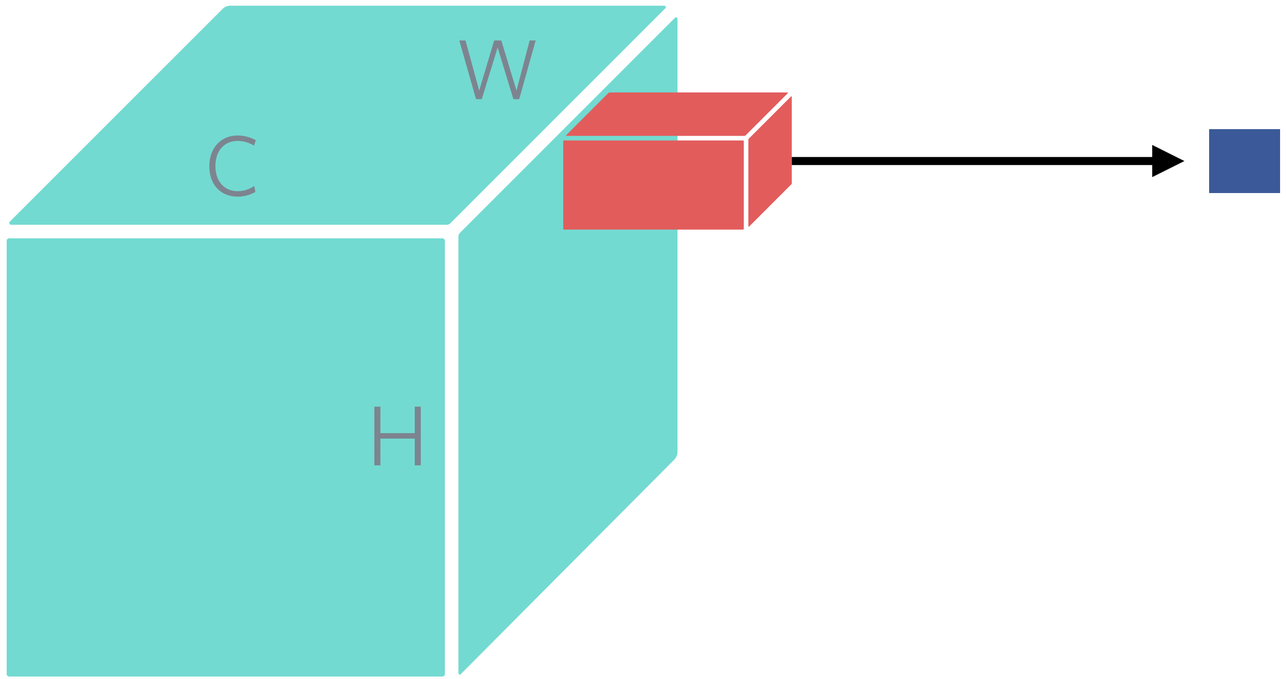
Each placement gives one output



Input Image

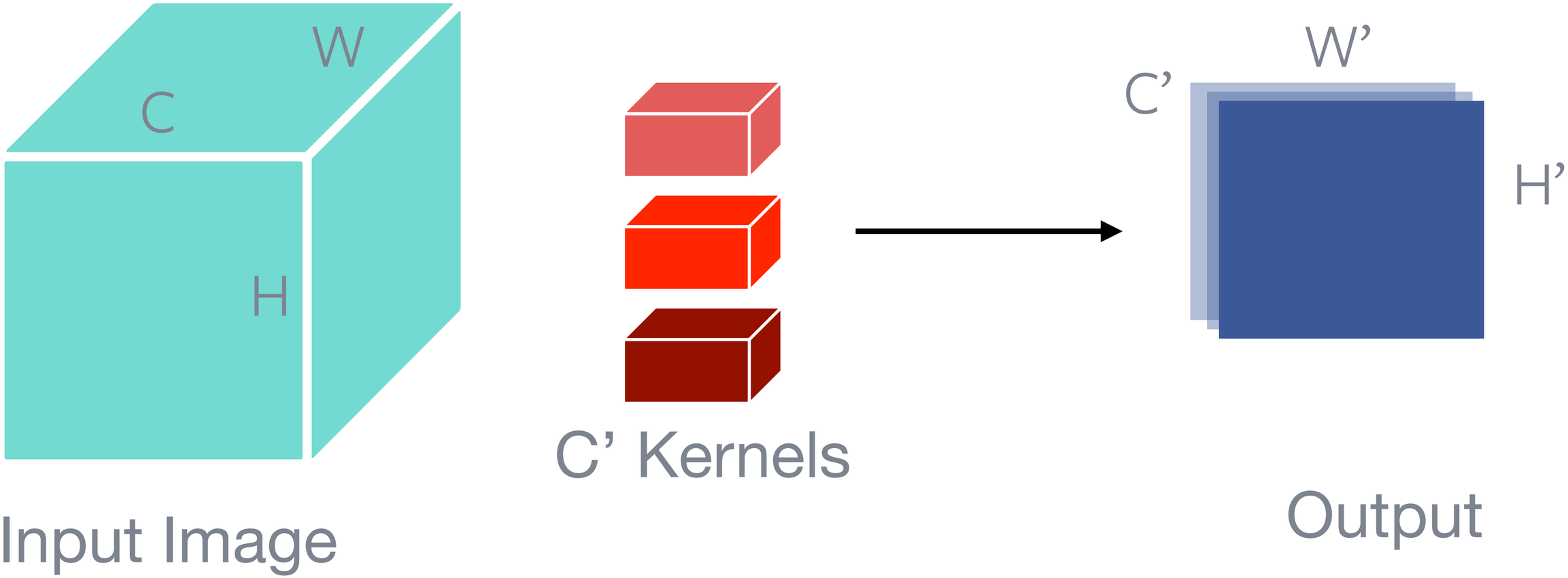
Placing the filter differently

Move filter by more than one pixel



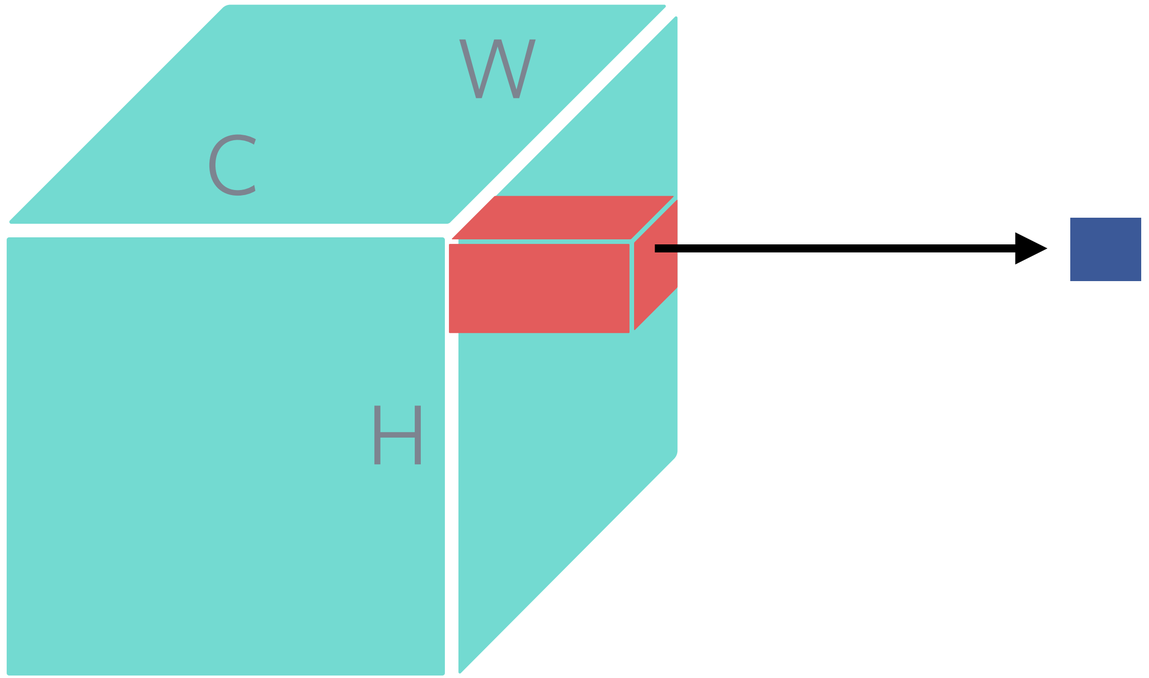
Input Image

Changing the spatial dimensions

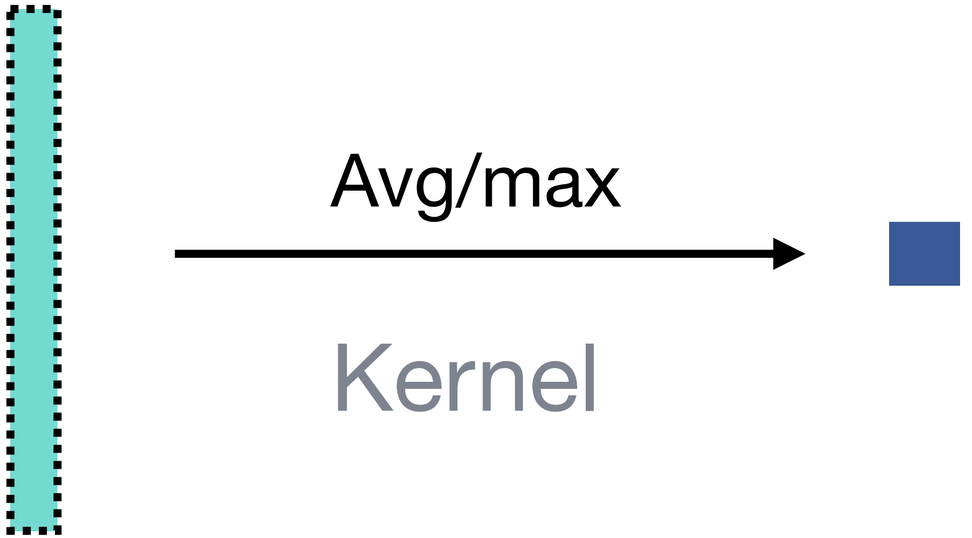


Pooling: A special kernel

Kernel just computes average or max



Input Image



Patch

Output

ConvNet

Stacks of convolutional layers + non-linearities + pooling

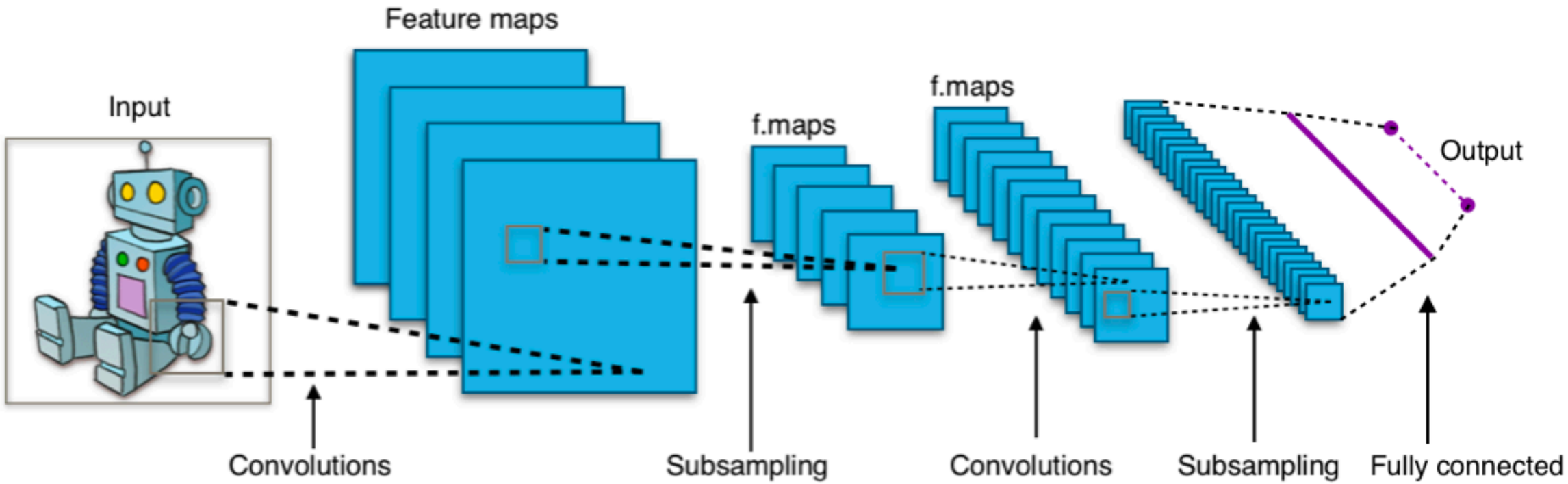
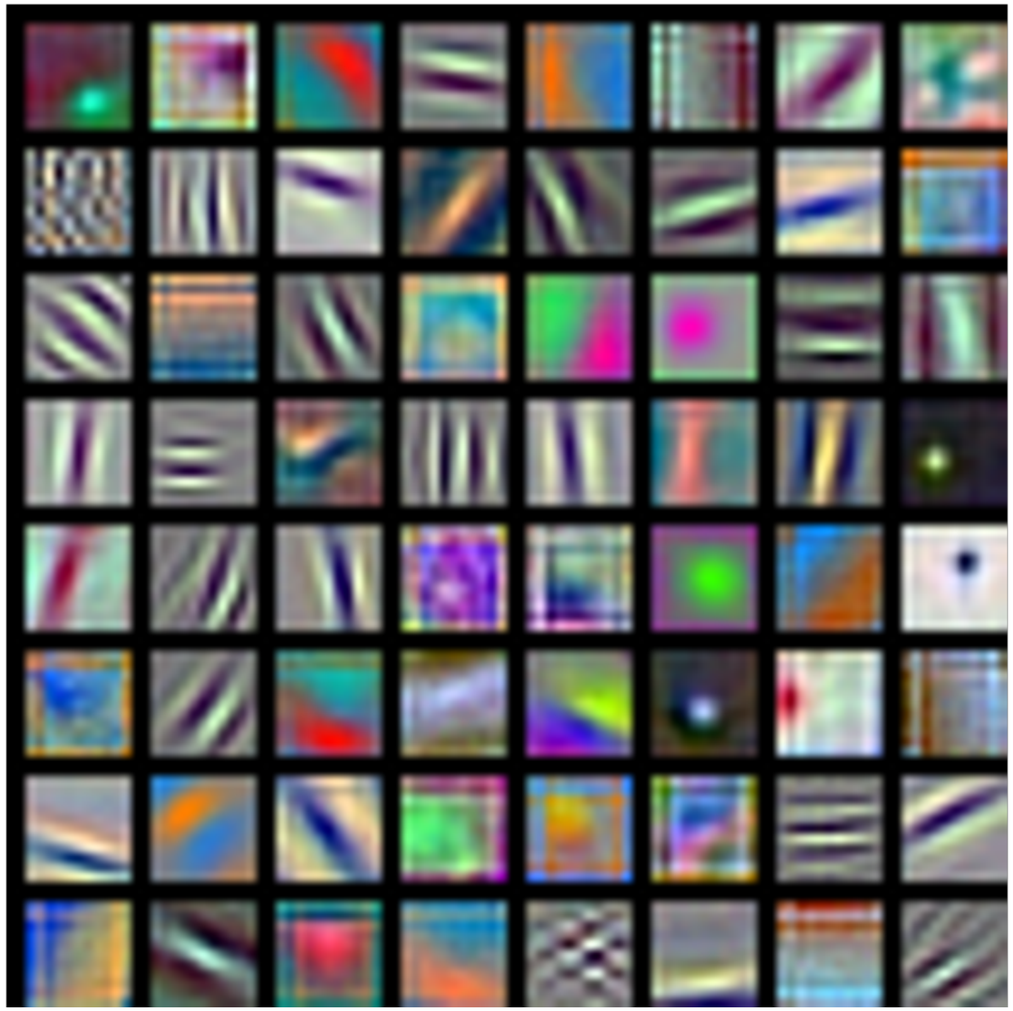


Image credit: Wikimedia

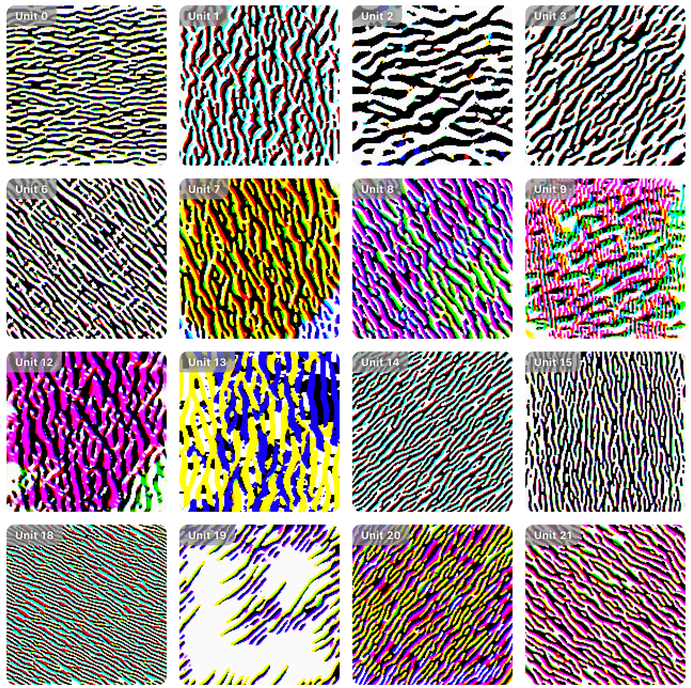
What do convolutional filters “look” like?



First few layers capture color and texture information

Image credit: AlexNet

Visualizing what ConvNets learn

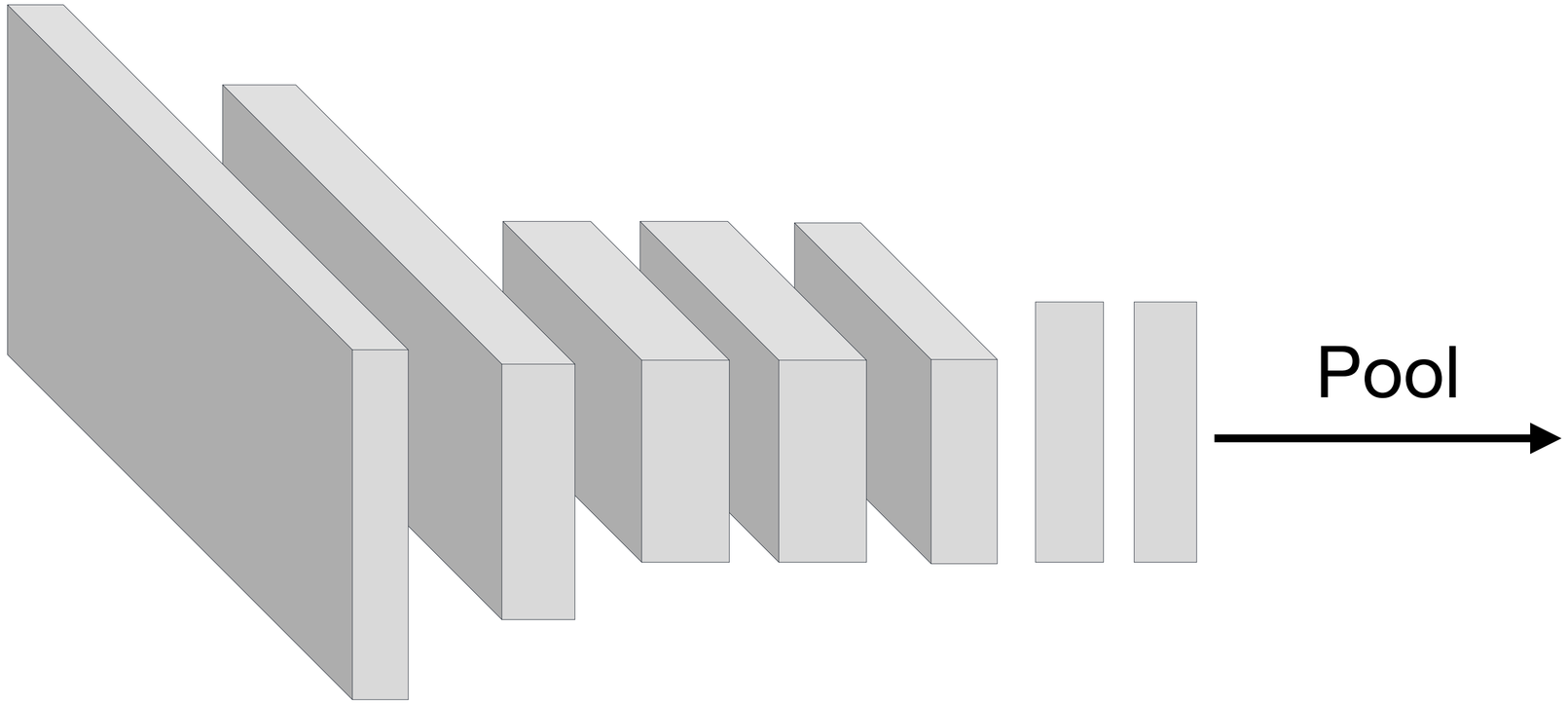


Layer closest to the input

Layer closest to the output

Visualizations:https://microscope.openai.com/models/inceptionv1?models.technique=deep_dream

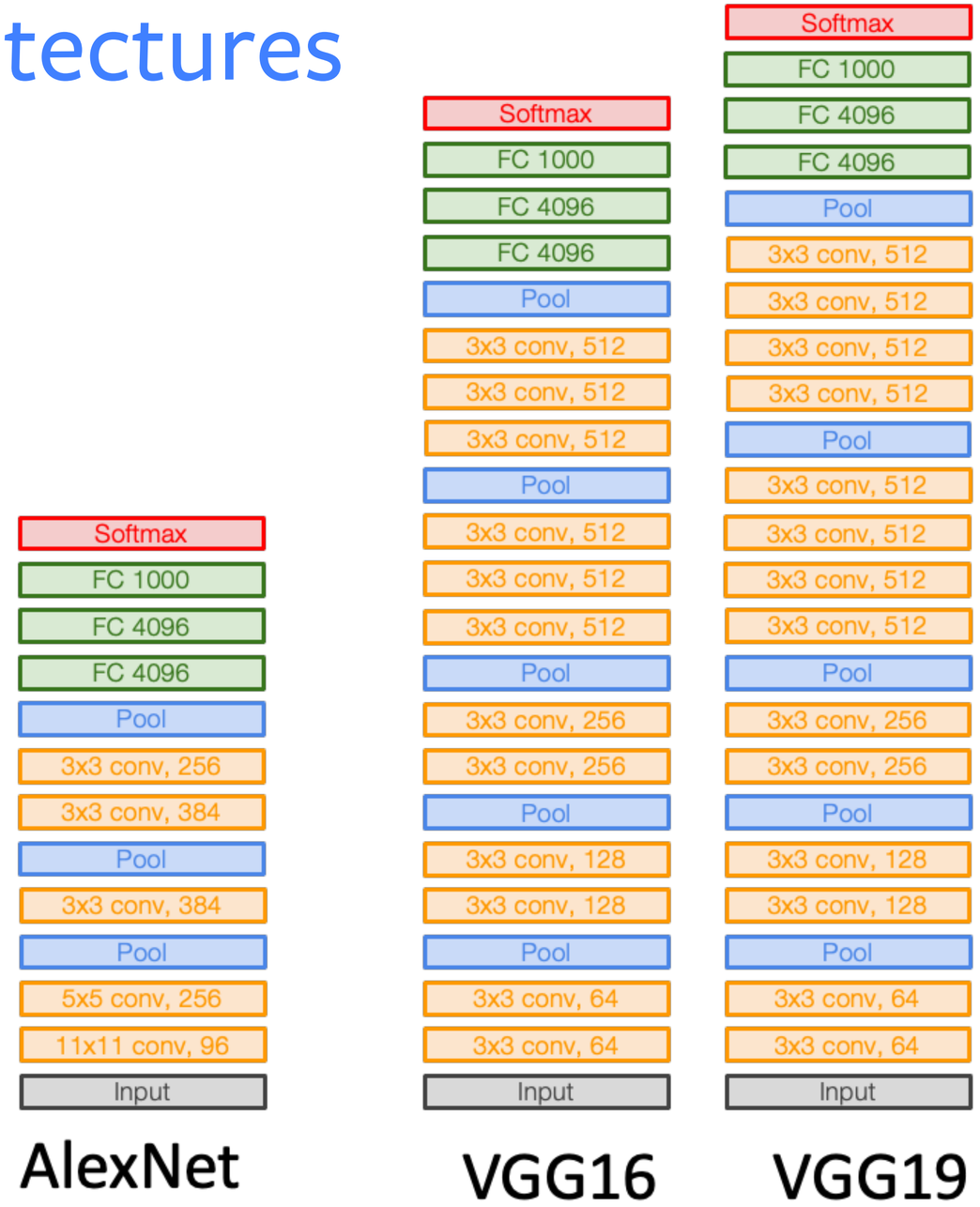
Global Image Features from ConvNets



Nearest Neighbors

Devlin et al, “Exploring Nearest Neighbor Approaches for Image Captioning”, 2015

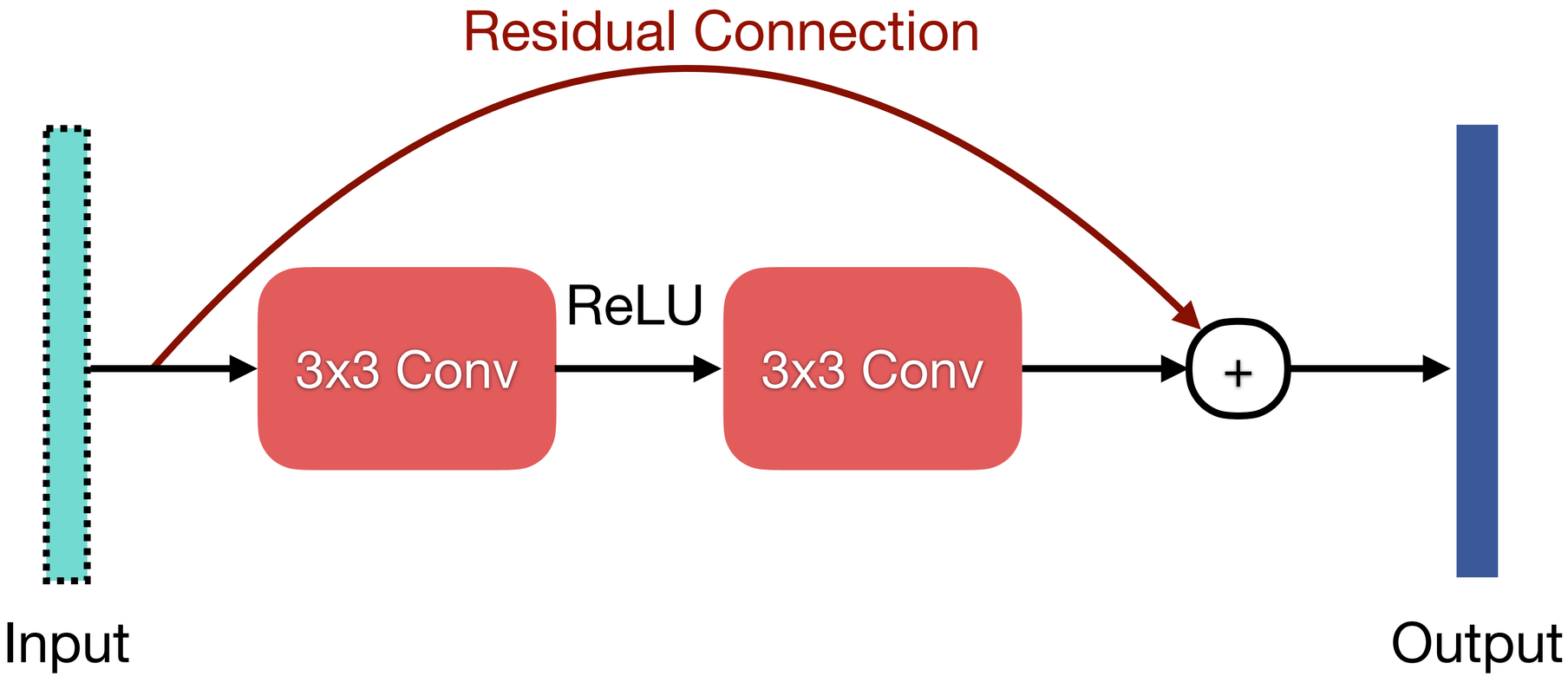
ConvNet architectures



AlexNet - Krizhevsky et al., 2012 ; VGG - Simonyan & Zisserman, 2016

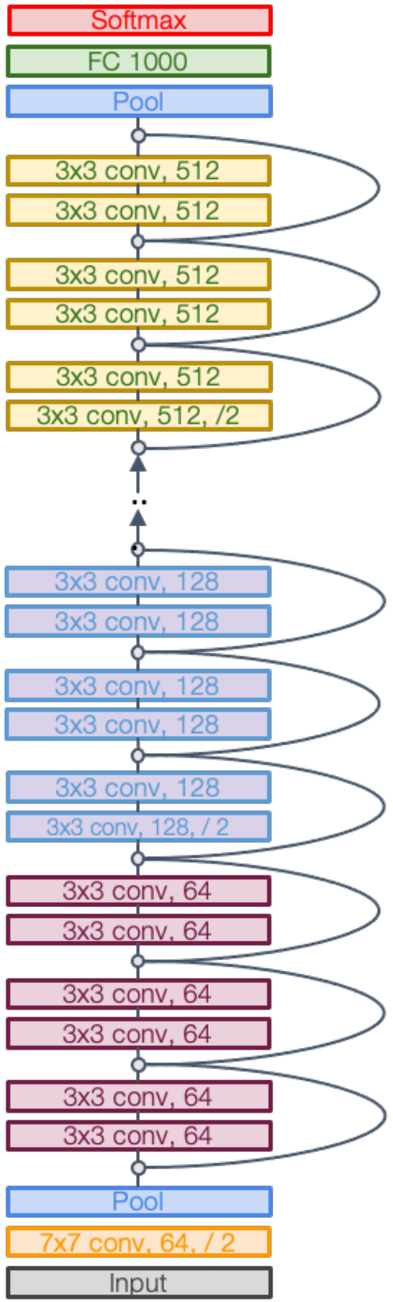
Figure credit: Justin Johnson

Residual Networks



ResNets - He et al., 2016

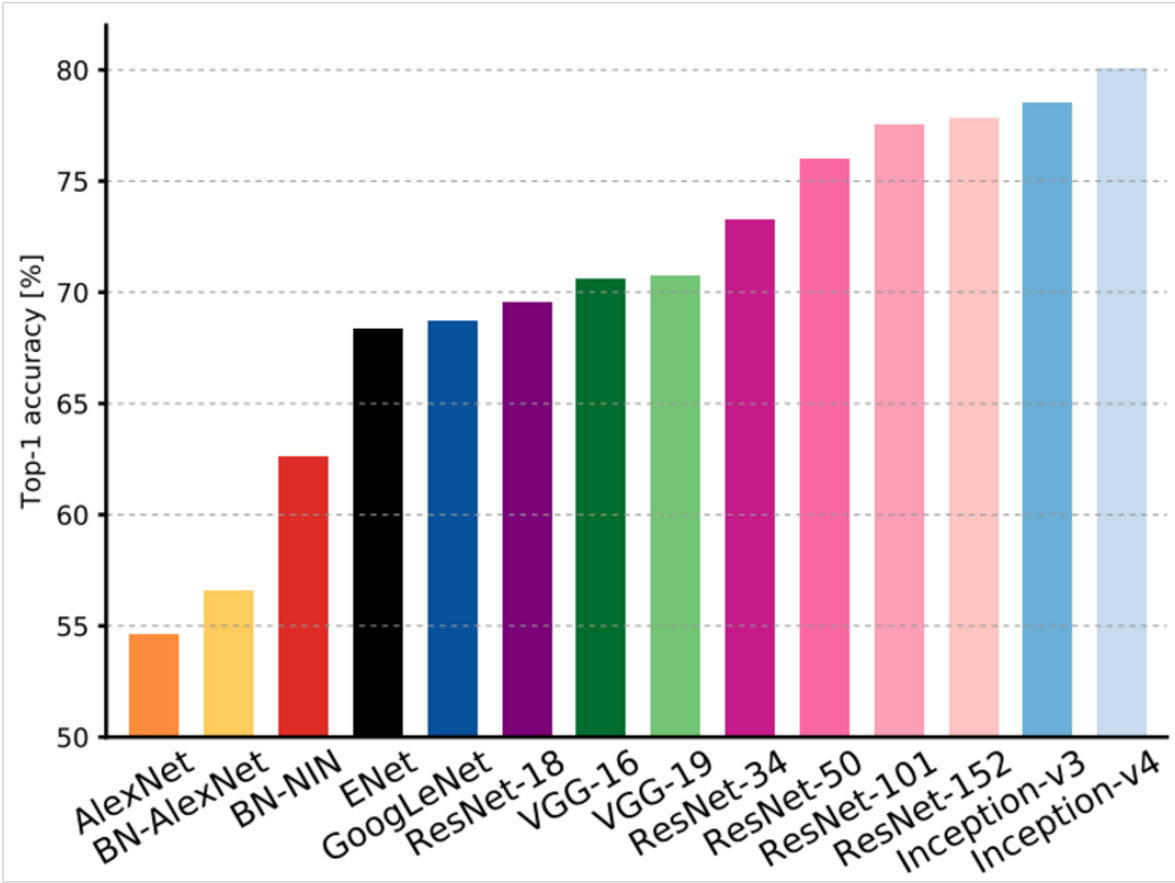
Residual Networks

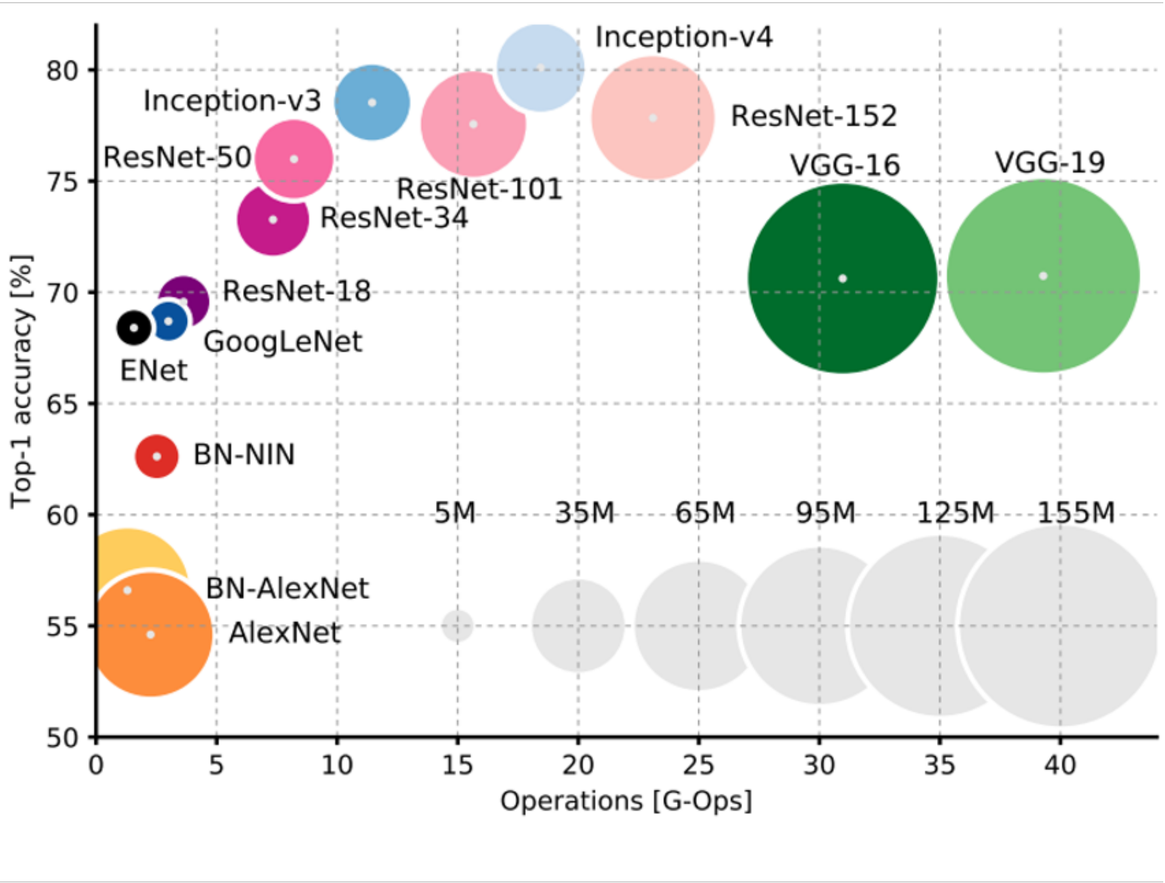


ResNets - He et al., 2016

Figure credit: Justin Johnson

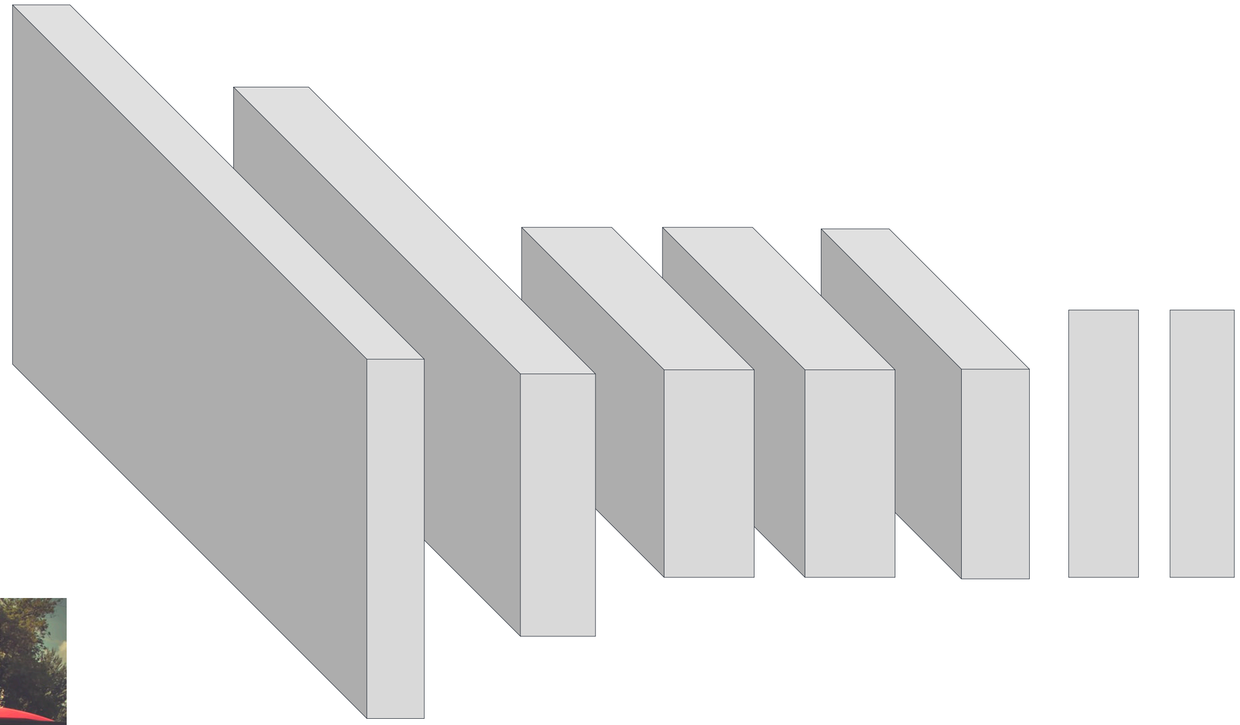
Image Classification Performance



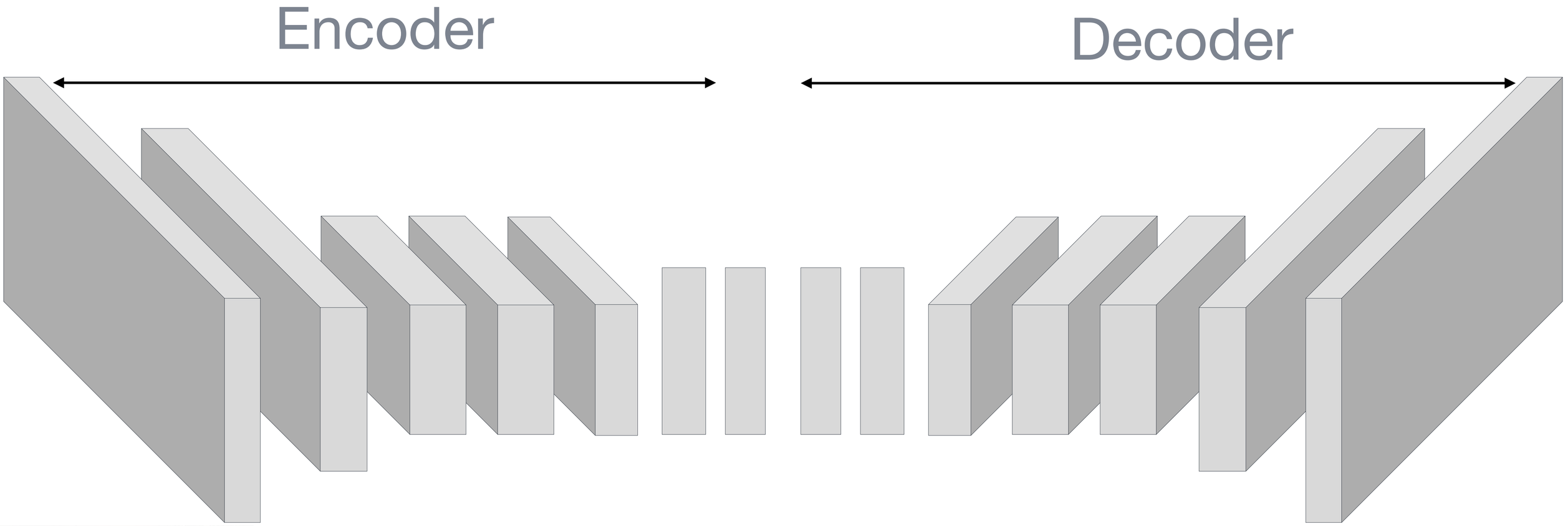


Canziani et al, “An analysis of deep neural network models for practical applications”, 2017

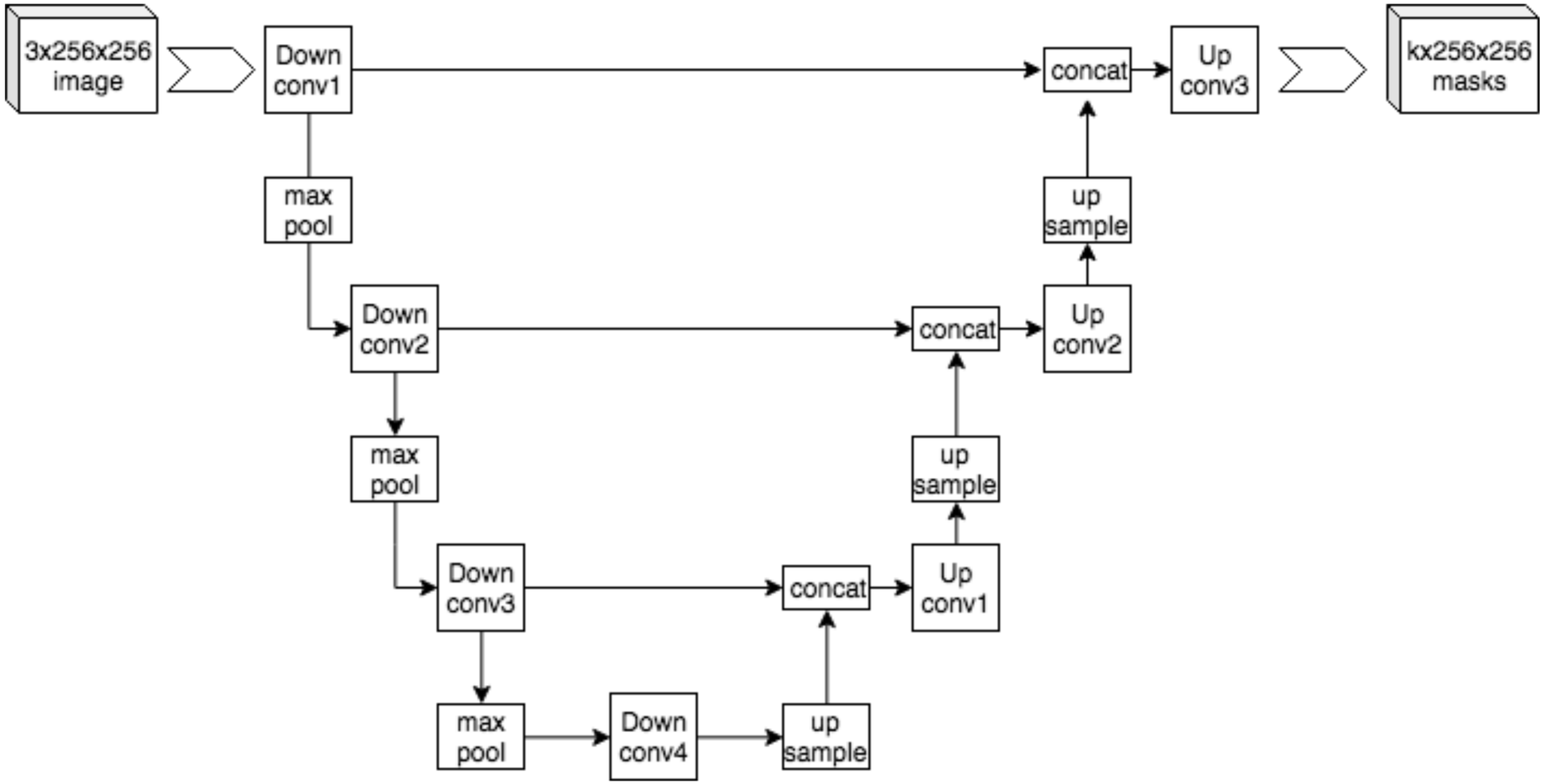
Image Classification: Progressively decrease spatial resolution



“Dense” Tasks



UNet

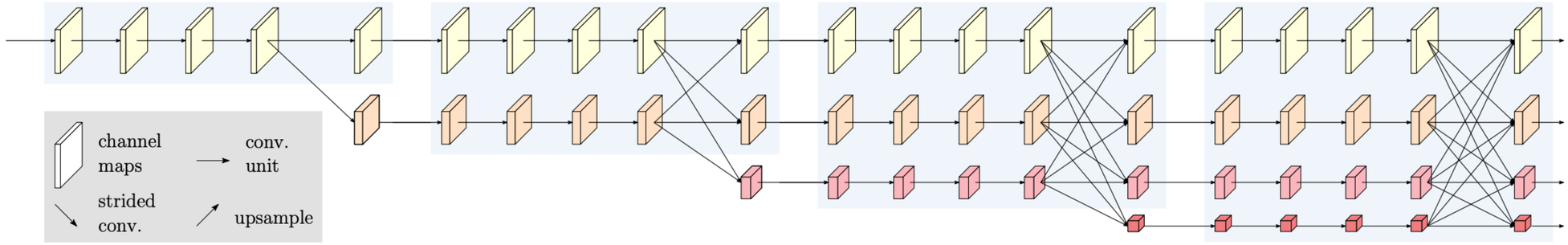


UNets - Ronneberger et al., 2015

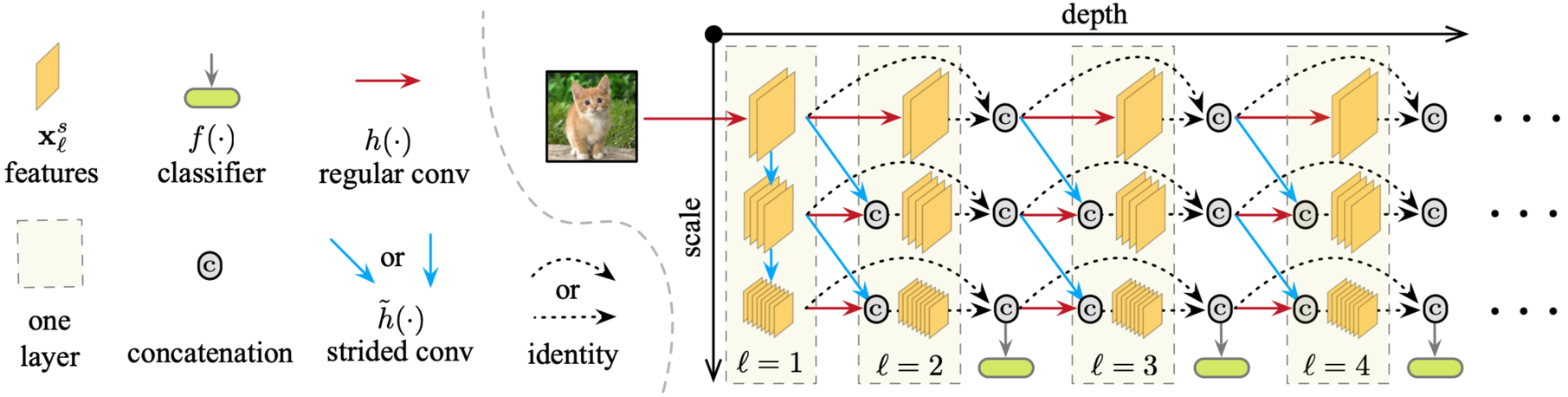
Image Credit: Wikipedia

Multi-resolution feature maps in a single network

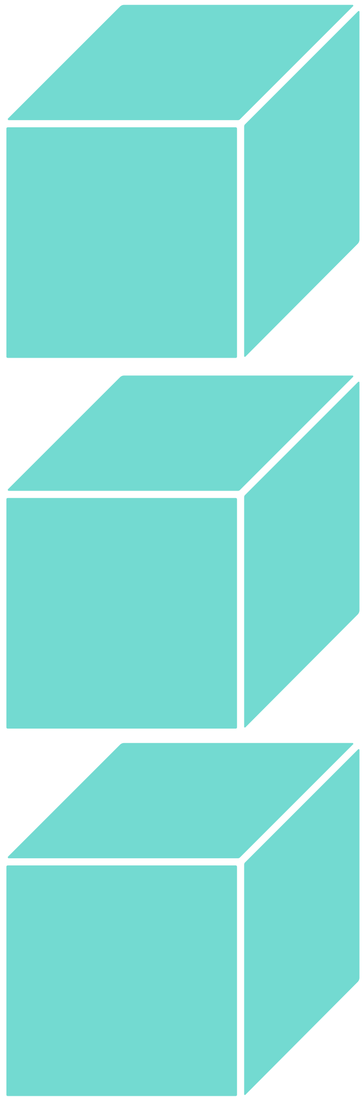
HRNet - Wang et al., 2020



MS DenseNet - Huang et al., 2018



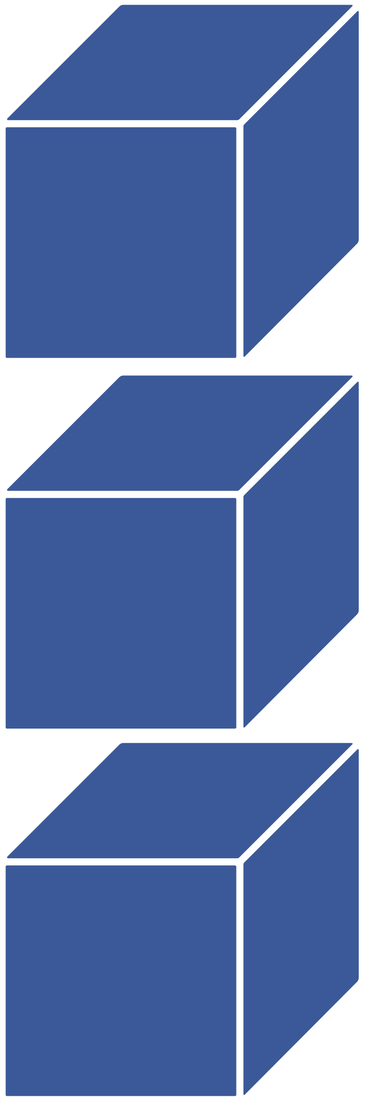
Videos ? —> Temporal Convolution



Input Video

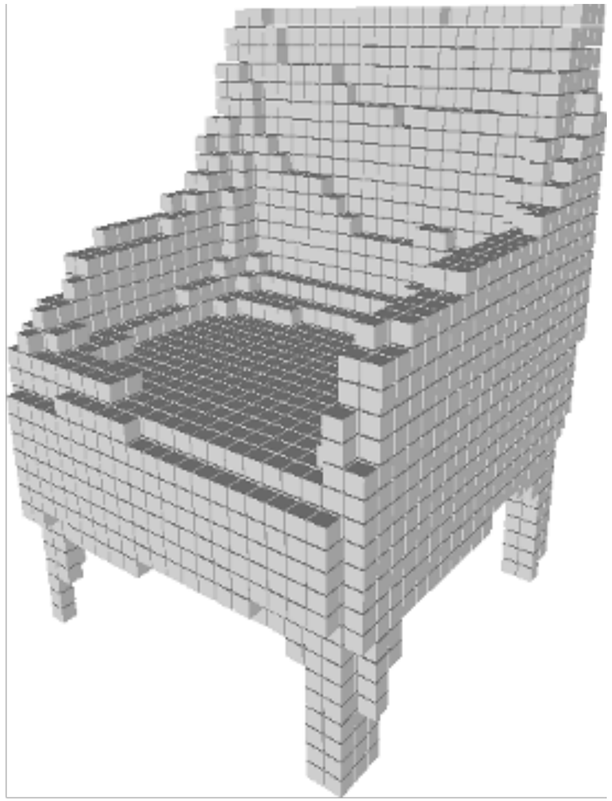
Kernel

Kernel operates on both
spatial and temporal dimensions



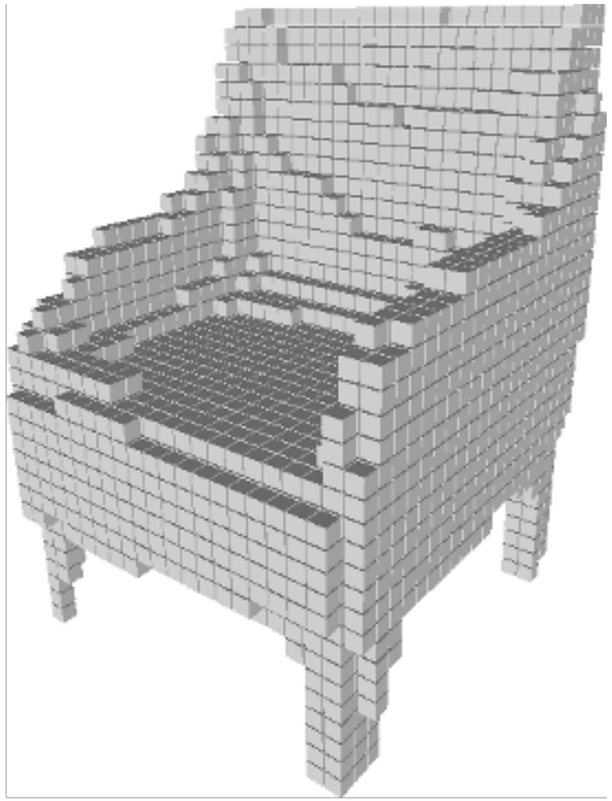
Output

3D data



Input is a “voxel” grid

3D data — convolution can work!



Kernel

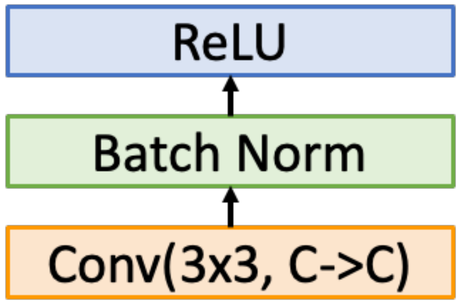
Input is a “voxel” grid

Incomplete story …

Depthwise Separable Convolutions

Standard Convolution Block

Total cost: $9C^2HW$



$9C^2HW$

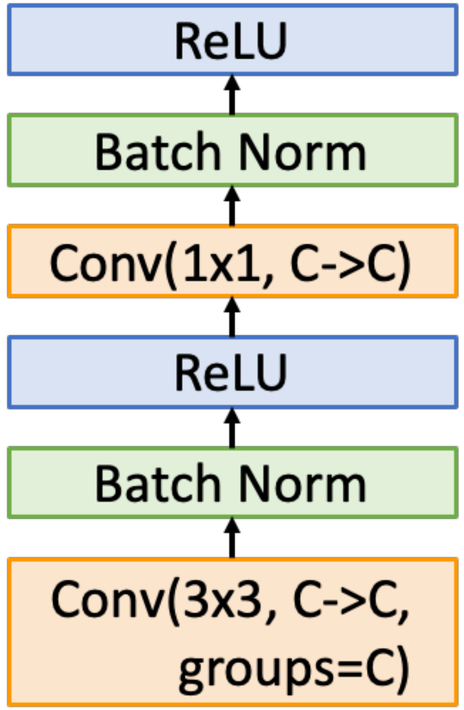
Speedup = $9C^2/(9C+C^2)$

= $9C/(9+C)$

=> 9 (as C->inf)

Depthwise Separable Convolution

Total cost: $(9C + C^2)HW$



C^2HW

"Pointwise Convolution"

$9CHW$

"Depthwise Convolution"

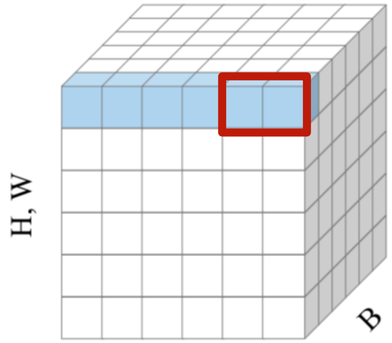
Slide credit: Justin Johnson

Normalization in Deep Networks

- Activations are N (batch size) x C (channels) x H (height) x W (width)

Layer Response Normalization

Local Response **Normalization**



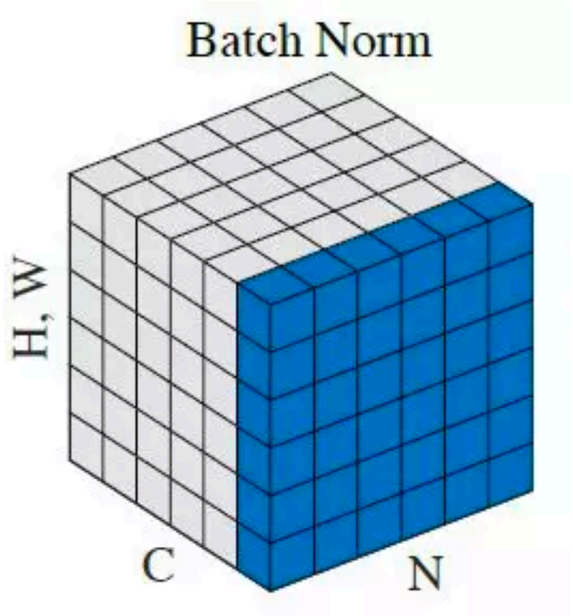
$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}$$

"j" indexes over channels

This sort of response normalization implements a form of lateral inhibition inspired by the type found in real neurons, creating competition for big activities amongst neuron outputs computed using different kernels.

ImageNet Classification with Deep Convolutional Neural Networks - Krizhevsky et al., 2012
See also - "Local Contrast Normalization"
- What is the best multi-stage architecture for object recognition? - Jarret et al., 2009

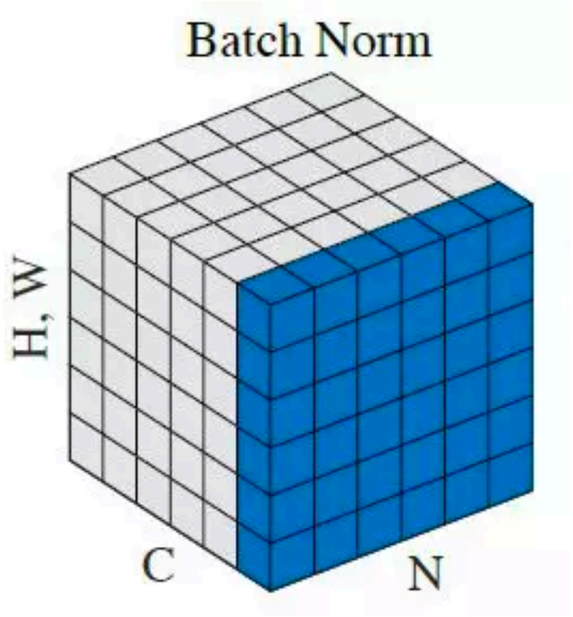
BatchNorm



- Standardize the activations
- Compute first order and second order batch statistics
  - mean (Estimated Moving Average)
  - variance
- Normalize activations
- Apply a learned affine transform
- Reduces need for "Dropout", LRN, weight decay, photometric augmentations

Image from Wu & He, 2018
Batch normalization - Ioffe and Szegedy
See also - Efficient backprop: LeCun et al., 1998

BatchNorm



- Activations are N (batch size) x C (channels) x H (height) x W (width)
- Compute mean/variance along N, H, W
- Pixels in blue normalized by same mean/variance

Image from Wu & He, 2018
Batch normalization - Ioffe and Szegedy

Fusing layers at test time

- At test time, BatchNorm behaves as a 1x1 conv filter

$$\hat{\mathbf{f}}_{i,j} = \mathbf{W}_{BN} \cdot (\mathbf{W}_{conv} \cdot \mathbf{f}_{i,j} + \mathbf{b}_{conv}) + \mathbf{b}_{BN}$$

- This filter can be “fused” with previous conv layer

- filter weights: $\mathbf{W} = \mathbf{W}_{BN} \cdot \mathbf{W}_{conv}$
- bias: $\mathbf{b} = \mathbf{W}_{BN} \cdot \mathbf{b}_{conv} + \mathbf{b}_{BN}$

Math from: https://learnml.today/speeding-up-model-with-fusing-batch-normalization-and-convolution-3

BatchNorm - what does it do?

- Out of syllabus question :)

- Makes the optimization landscape smooth -> gradients are stable and predictive
  - effective smoothness of gradients
  - Lipschitz smoothness of loss

How Does Batch Normalization Help Optimization? - Santurkar et al., 2018

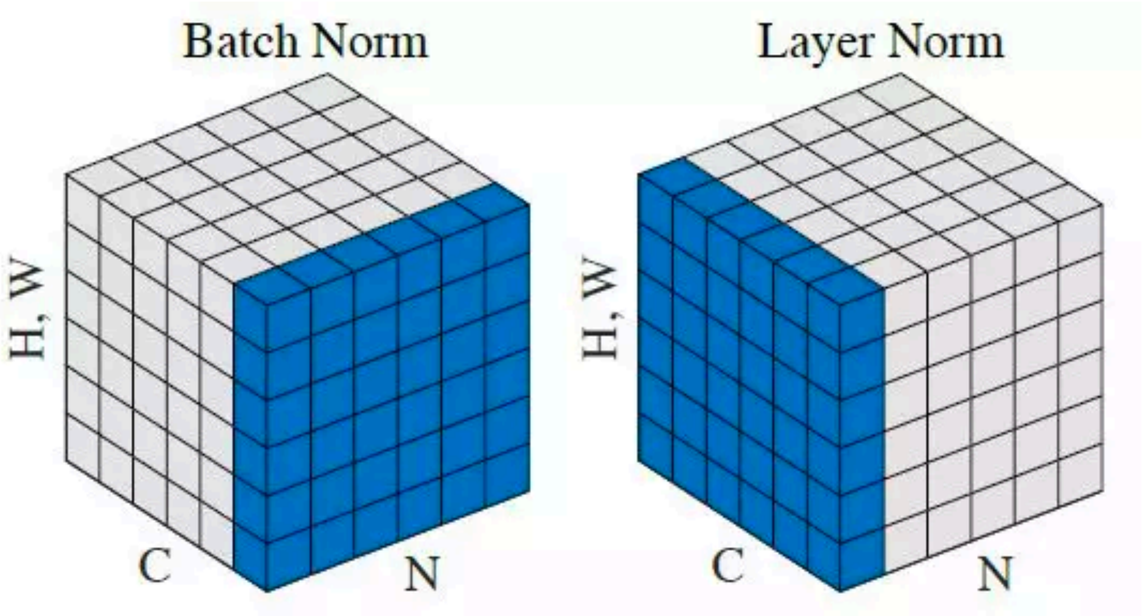
BatchNorm - what does it not let us do?

- Sleep peacefully :)

- Train with correlated samples
- Train with small mini-batches, i.e., micro batches
- Train with varying sized mini-batches

- Not widely used in training recurrent networks
- Kept ″frozen″ for object detection methods

Layer Norm



- Compute mean/variance along C, H, W
- Each sample in a batch is normalized independently
- Popular for Transformer Architectures

Same behavior in train/test

Image from Wu & He, 2018
Layer Norm - Ba, Kiros, Hinton

Layer Norm

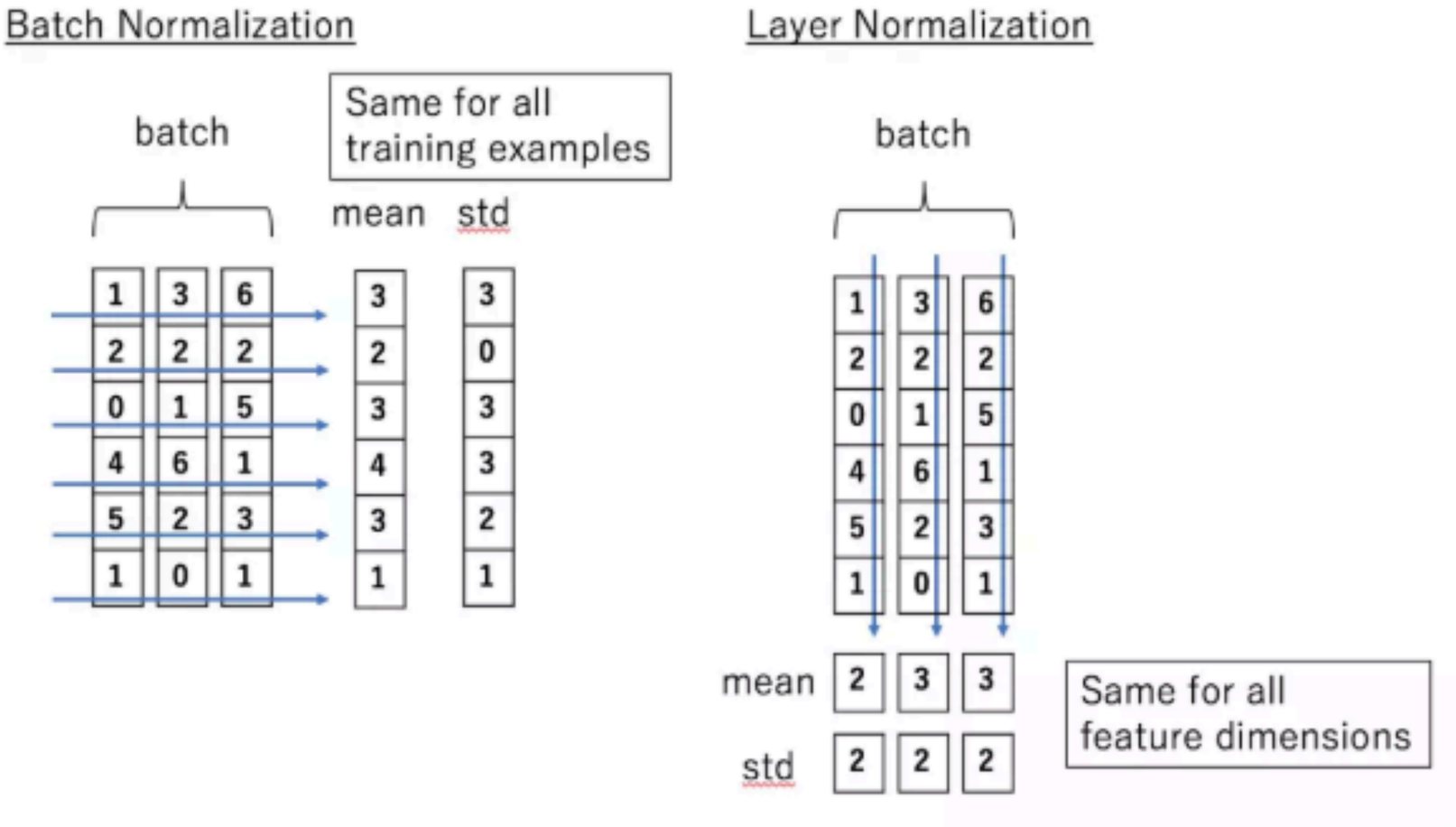


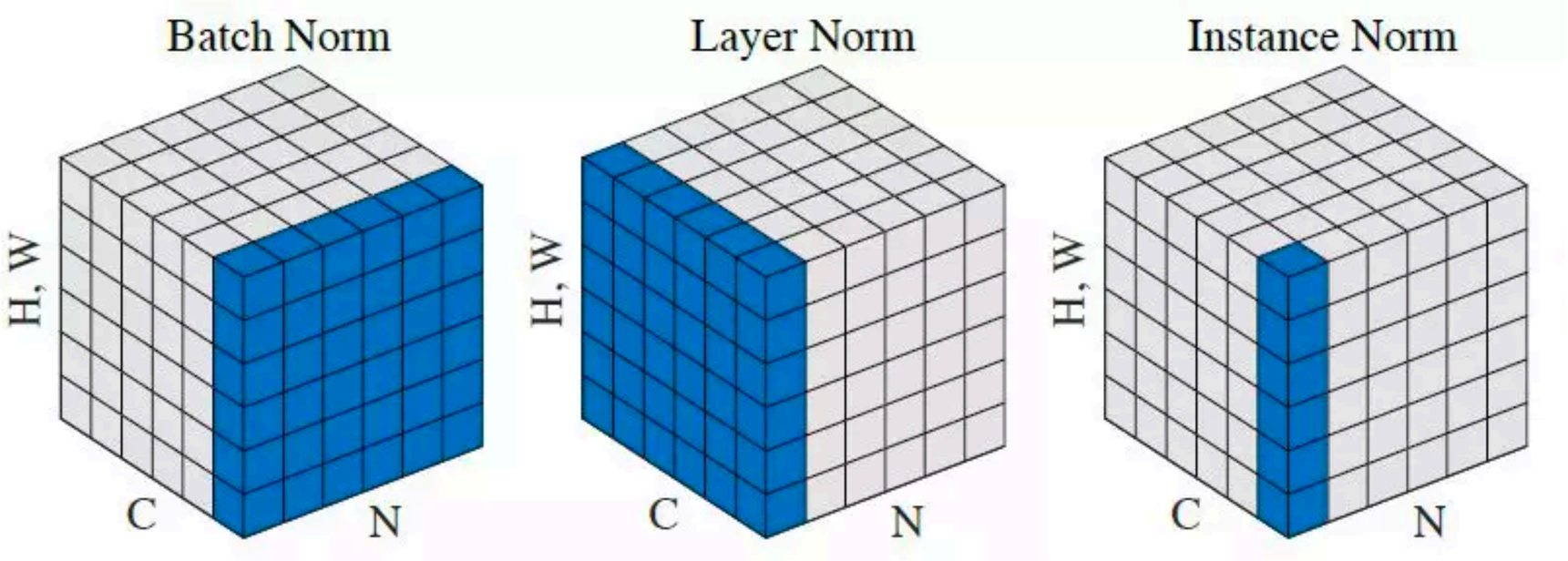
Image from keitakara

Layer Norm

- Shows promise in micro-batch training and recurrent networks

- Not so much for ConvNets (Sec 6.7 of the LayerNorm paper)
  - LayerNorm assumes all channels make similar contributions
- Keeps no running stats => test samples should be very similar to train

Instance Norm



- Mean/var along H, W
- Each sample normalized independently

Instance Norm - Ulyanov, Vedaldi, Lempitsky

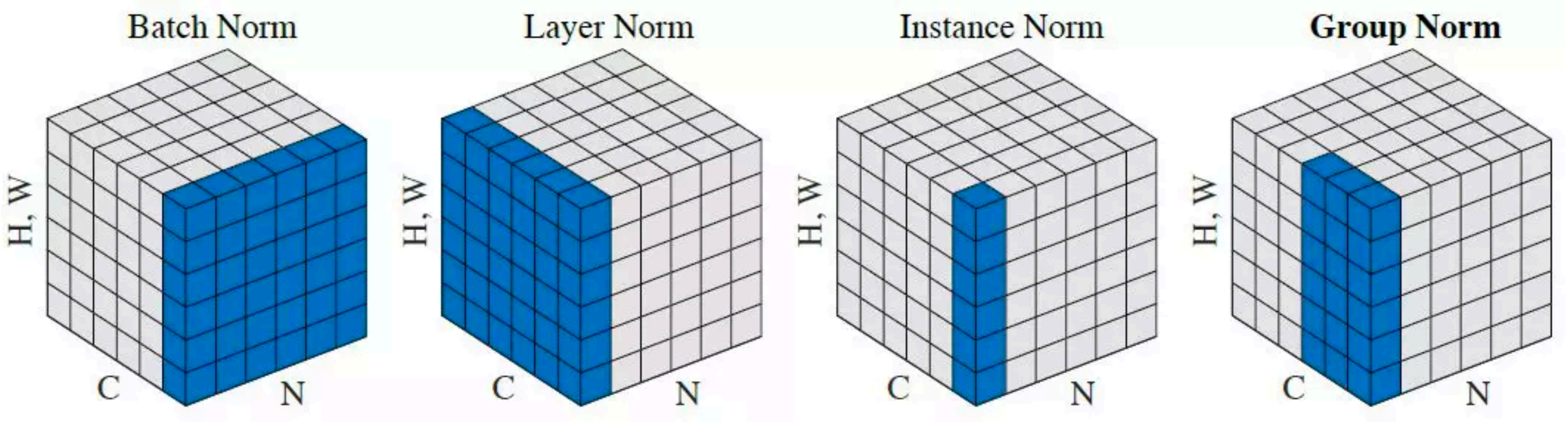
Instance Norm

Figure 2: Comparison of normalization techniques in image stylization. From left to right: BN, cross-channel LRN at the first layer, IN at the first layer, IN throughout.

Instance Norm

All channels are independent => Channel dependence is not captured

Group Norm



- Mean/var along "groups" of channels
- Each sample in a batch is normalized independently

Group Normalization Wu & He, 2018

Group Norm

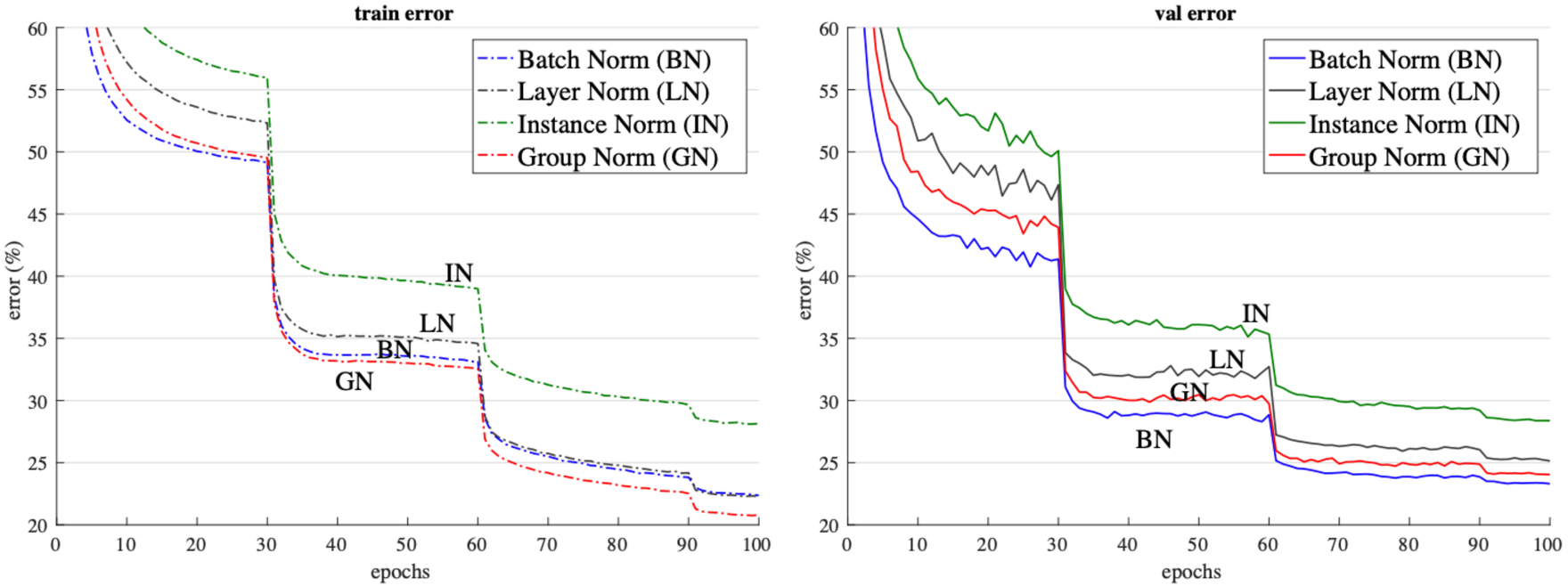


Figure 4. **Comparison of error curves** with a batch size of **32 images**/GPU. We show the ImageNet training error (left) and validation error (right) *vs*. numbers of training epochs. The model is ResNet-50.

At large batch sizes, performance is only slightly worse than BN

Group Norm

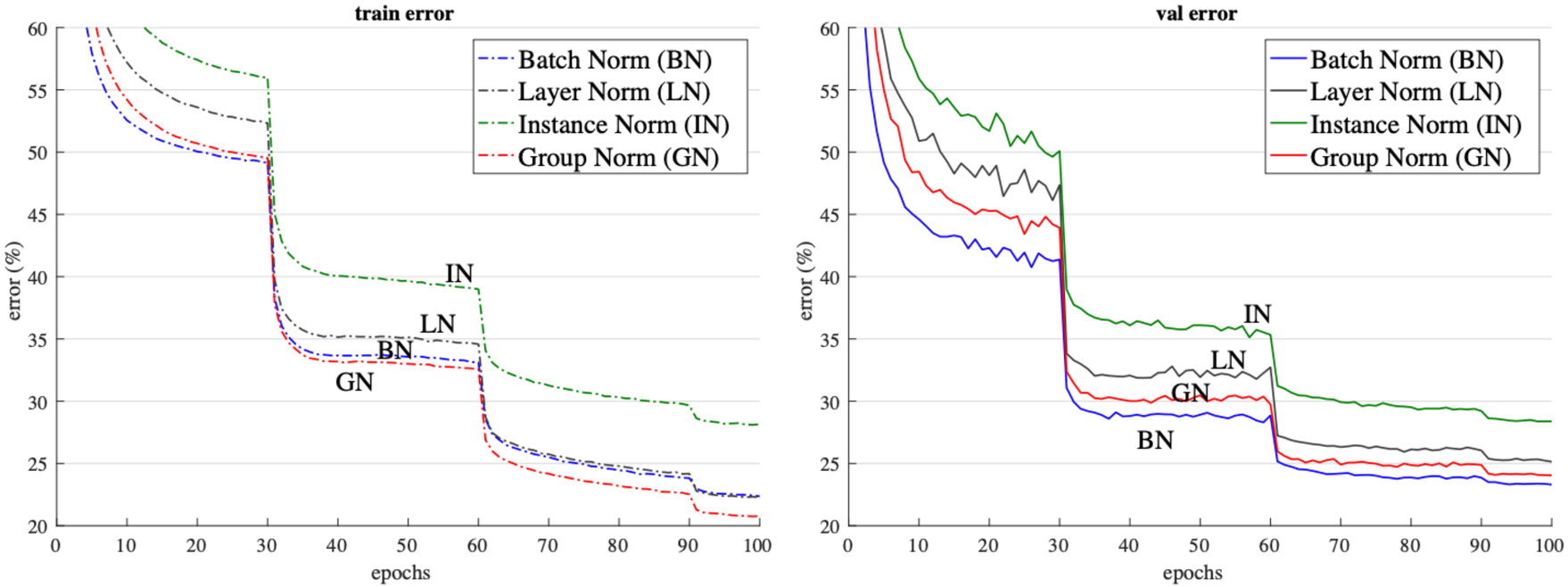
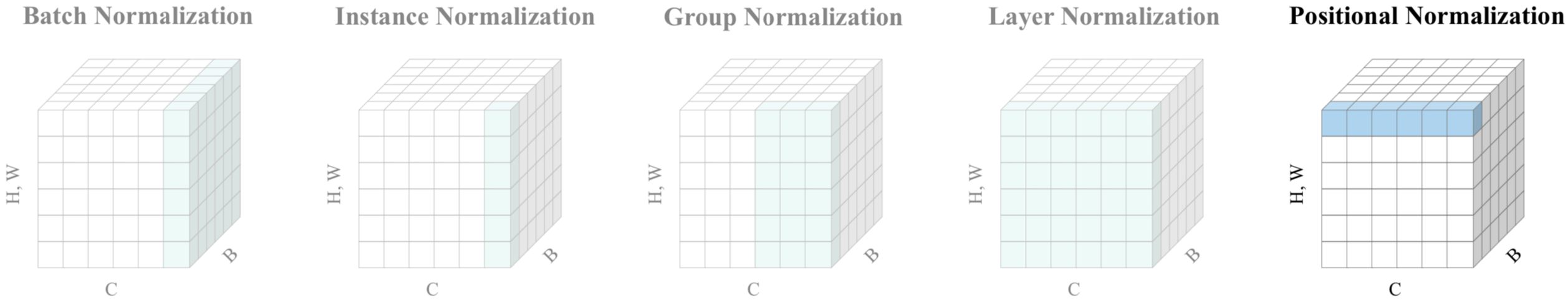


Figure 4. **Comparison of error curves** with a batch size of **32 images**/GPU. We show the ImageNet training error (left) and validation error (right) *vs*. numbers of training epochs. The model is ResNet-50.

This is understandable, because BN's mean and variance computation introduces uncertainty caused by the stochastic batch sampling, which helps regularization [26]. This uncertainty is missing in GN (and LN/IN). But it is possible that GN combined with a suitable regularizer will improve results. This can be a future research topic.

Position Norm

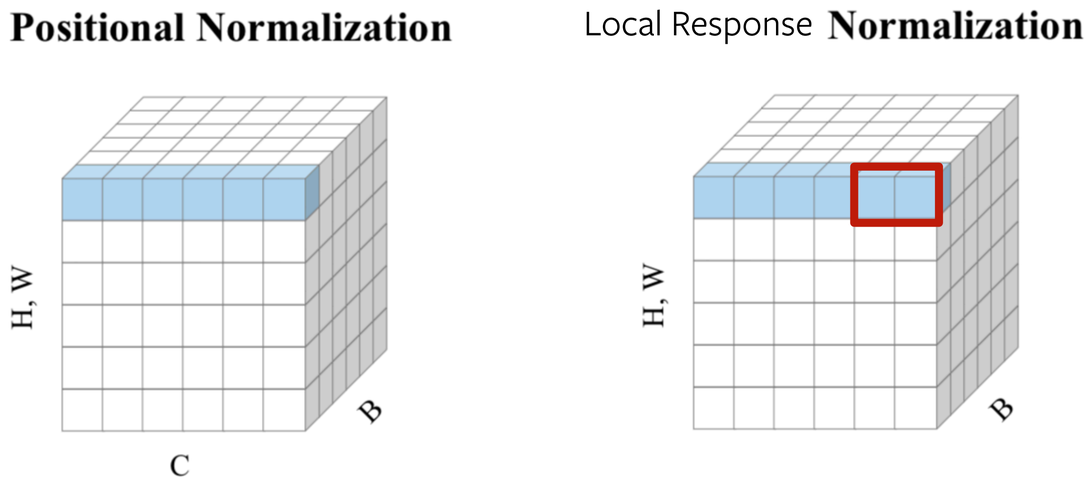


Normalize each location (height, width) in a feature map across channels

Designed mainly for image generation/labeling tasks (GANs)
No experiments on classification/detection tasks

Position Normalization - Li et al., 2019

Position Norm -- Related to LRN



I interpret the normalization scheme to be similar to LRN

Position Normalization - Li et al., 2019

Position Norm

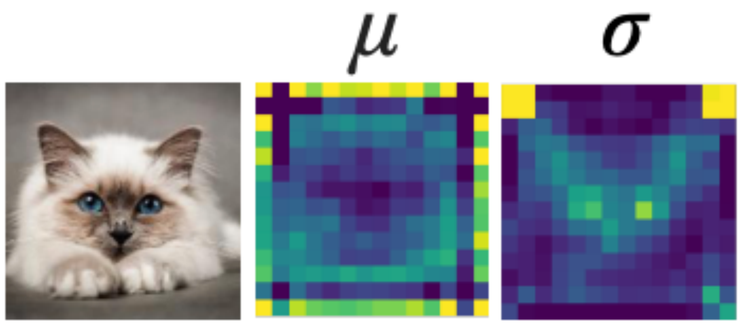


Figure 3: PONO statistics of DenseBlock-3 of a pretrained DenseNet-161.

Important difference from BN/GN/IN/LN

- Invariant under geometric transforms of the input
- Re-uses the mean/std to modulate features later
  - Called "moment" shortcut.

Weight Norm

- Normalize the weights
- Decouple length and direction of weights
- Parameters g, v are updated during learning

- Mean of activations still depends on v

$$\mathbf{w} = \frac{g}{||\mathbf{v}||}\mathbf{v}$$

Weight Normalization - Salimans & Kingma, 2016

Weight Norm

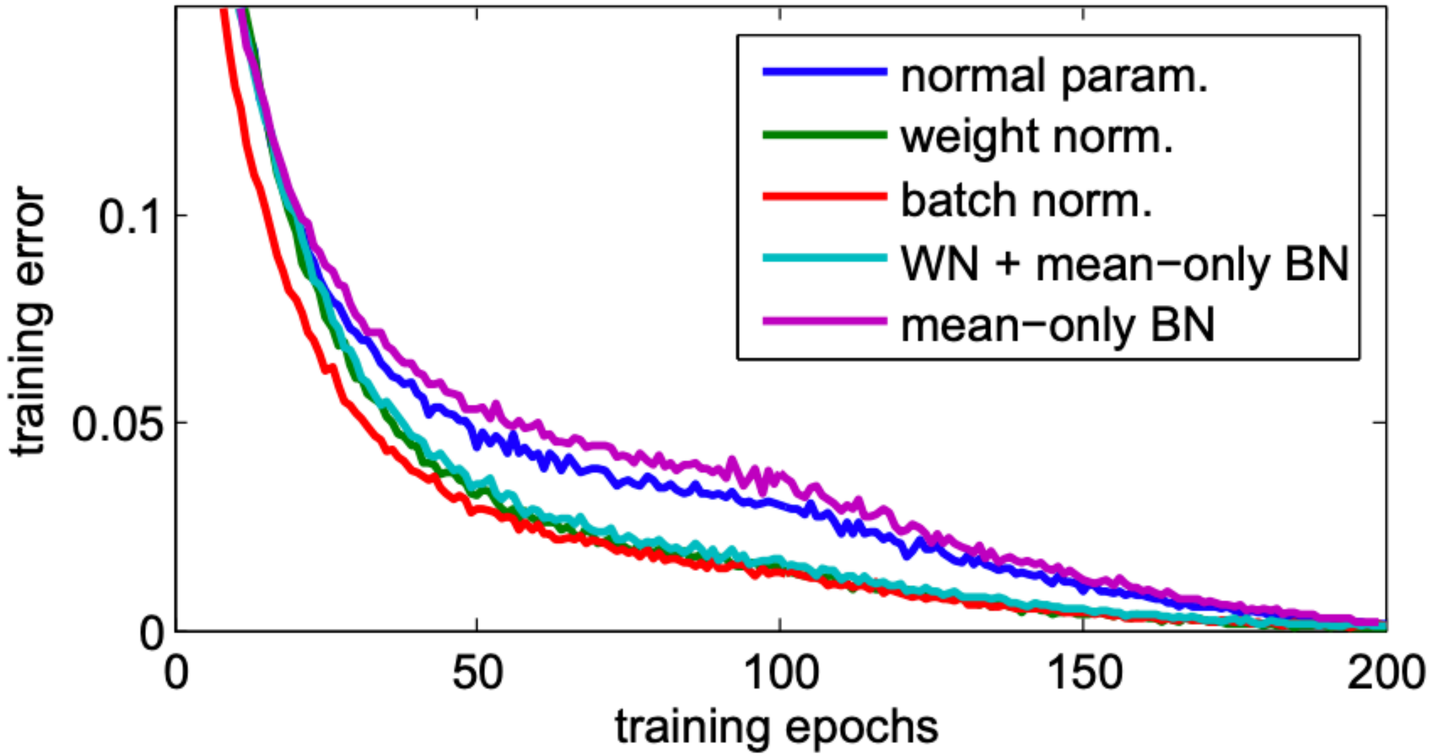


Figure 1: Training error for CIFAR-10

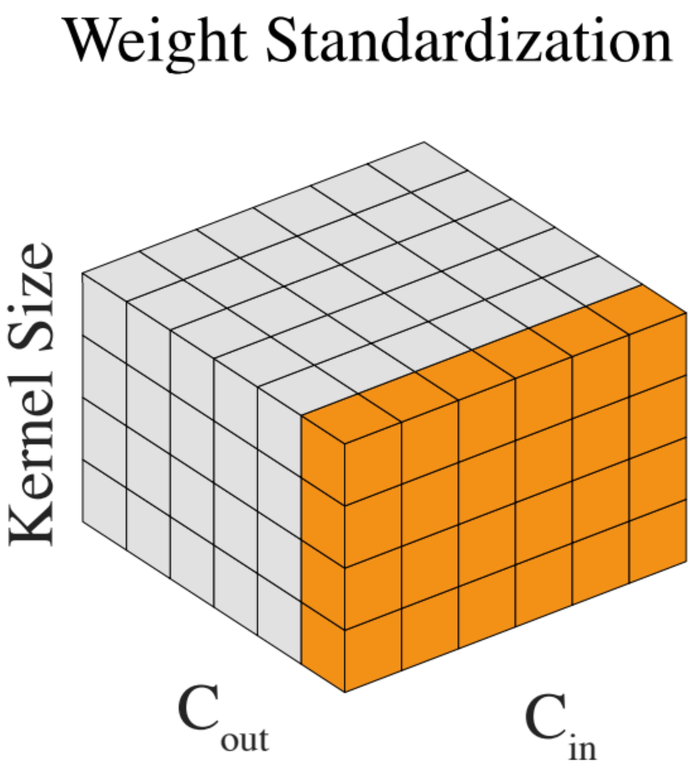
Weight Norm

	BN	LN	IN	GN	Weight Norm
val error	**23.6**	25.3	28.4	24.1	28.2
△ (*vs.* BN)	-	*1.7*	*4.8*	***0.5***	4.6

Table 1. **Comparison of error rates** (%) of ResNet-50 in the ImageNet validation set, trained with a batch size of **32 images**/GPU.

Numbers from Wu & He, 2018

Weight Standardization



- Standardize the weights (in addition to activations)
- Normalize each output channel independent of each other

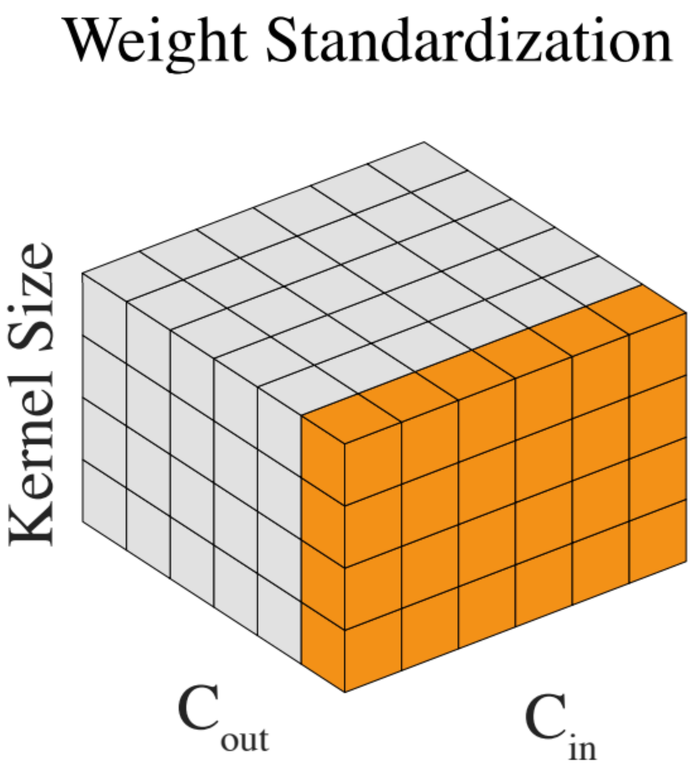
$$\text{dimension}(\mathbf{W}) = C_{\text{out}} \times (C_{\text{in}} \times \text{kernel\_size})$$

$$\boldsymbol{y} = \hat{\boldsymbol{W}} * \boldsymbol{x}$$

$$\hat{\boldsymbol{W}}_{i,j} = \frac{\boldsymbol{W}_{i,j} - \mu_{\boldsymbol{W}_{i,\cdot}}}{\sigma_{\boldsymbol{W}_{i,\cdot}} + \epsilon}$$

Weight Standardizaton - Qiao et al., 2019

Weight Standardization



- Standardize the weights (in addition to activations)
- Normalize each output channel independent of each other
- No affine transform (activation normalization will have it)

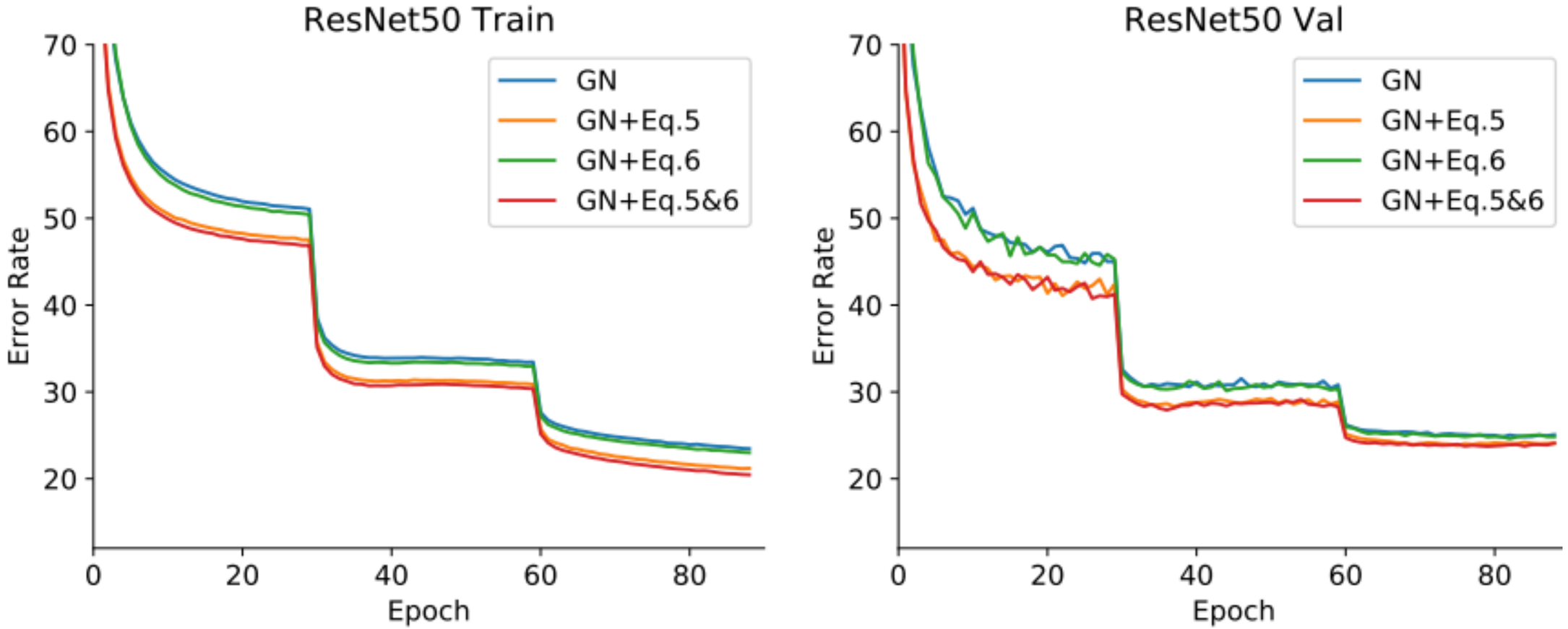
$$\text{dimension}(\mathbf{W}) = C_{\text{out}} \times (C_{\text{in}} \times \text{kernel\_size})$$

$$\boldsymbol{y} = \hat{\boldsymbol{W}} * \boldsymbol{x}$$

$$\hat{\boldsymbol{W}}_{i,j} = \frac{\boldsymbol{W}_{i,j} - \mu_{\boldsymbol{W}_{i,\cdot}}}{\sigma_{\boldsymbol{W}_{i,\cdot}} + \epsilon}$$

Normalization vs. Centering?

Eq 5 => Centering (subtract mean)
Eq 6 => Normalize (divide by std)



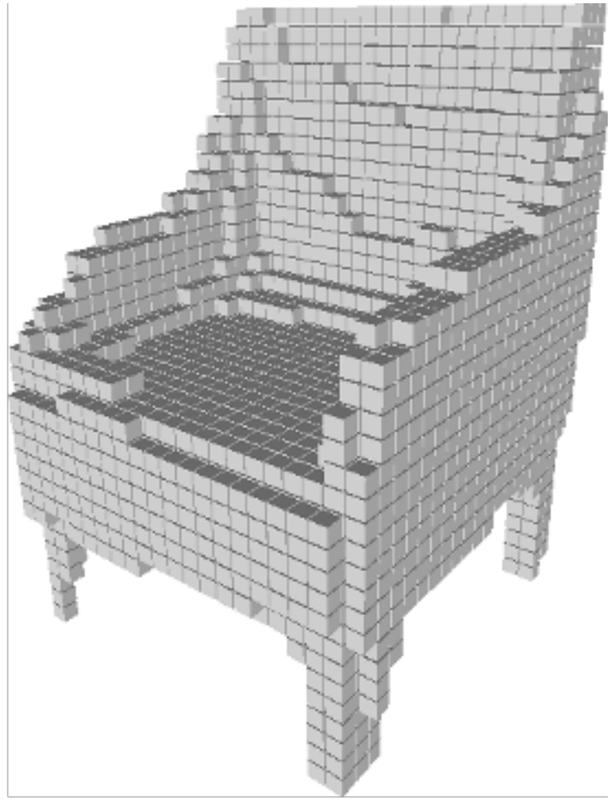
Major improvements seem to come from centering

Non-Convolutional Architectures

Assumption in convolution

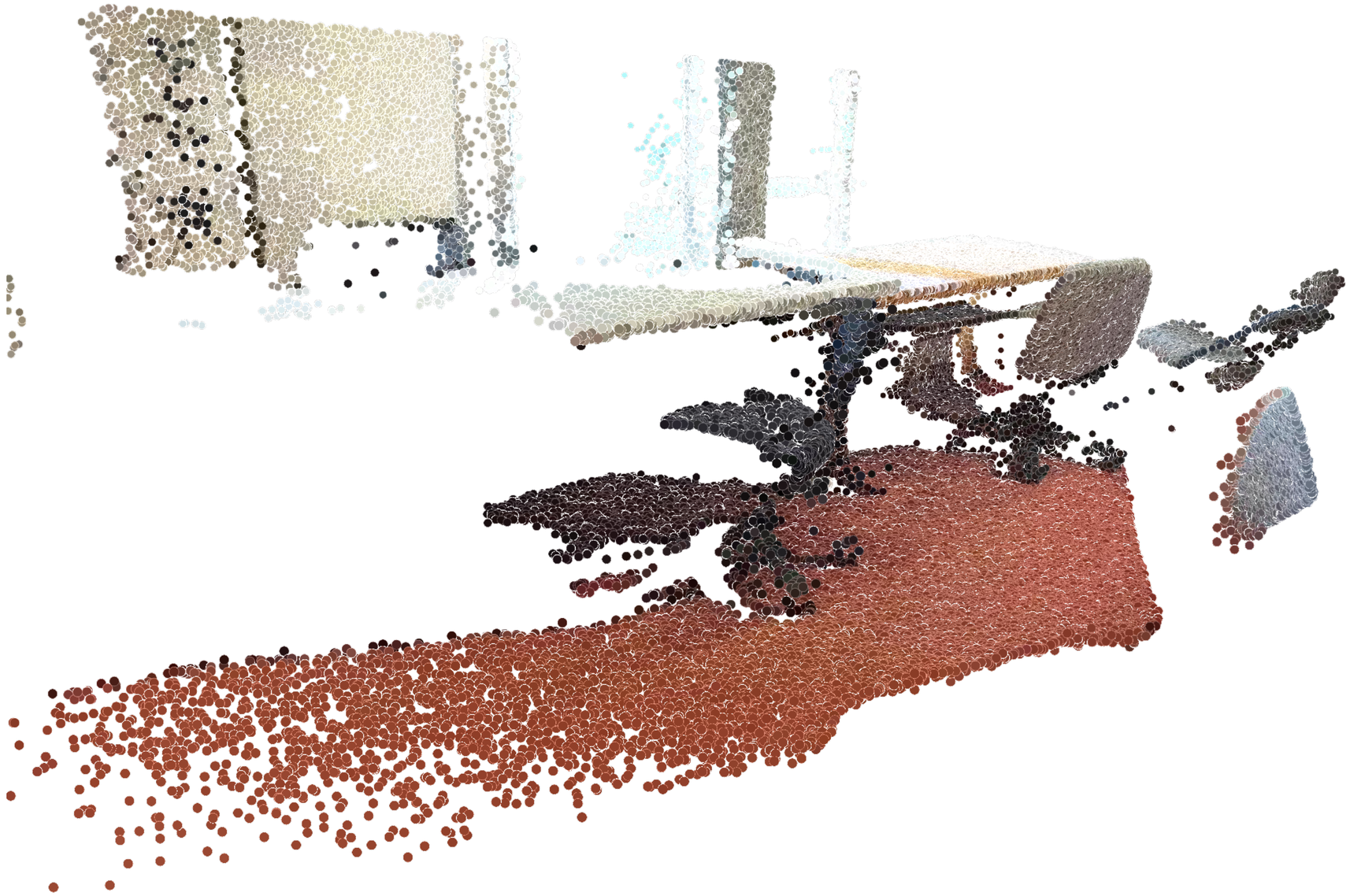
Input is a regularly spaced grid

Array of pixels



Array of voxels

Non-regular data



Point Clouds

- Varying Density
- Unordered points

Permutation invariant operators?

k x d kernel

Convolution?

N x d’ weights

Fully Connected?

Permutation invariant operators

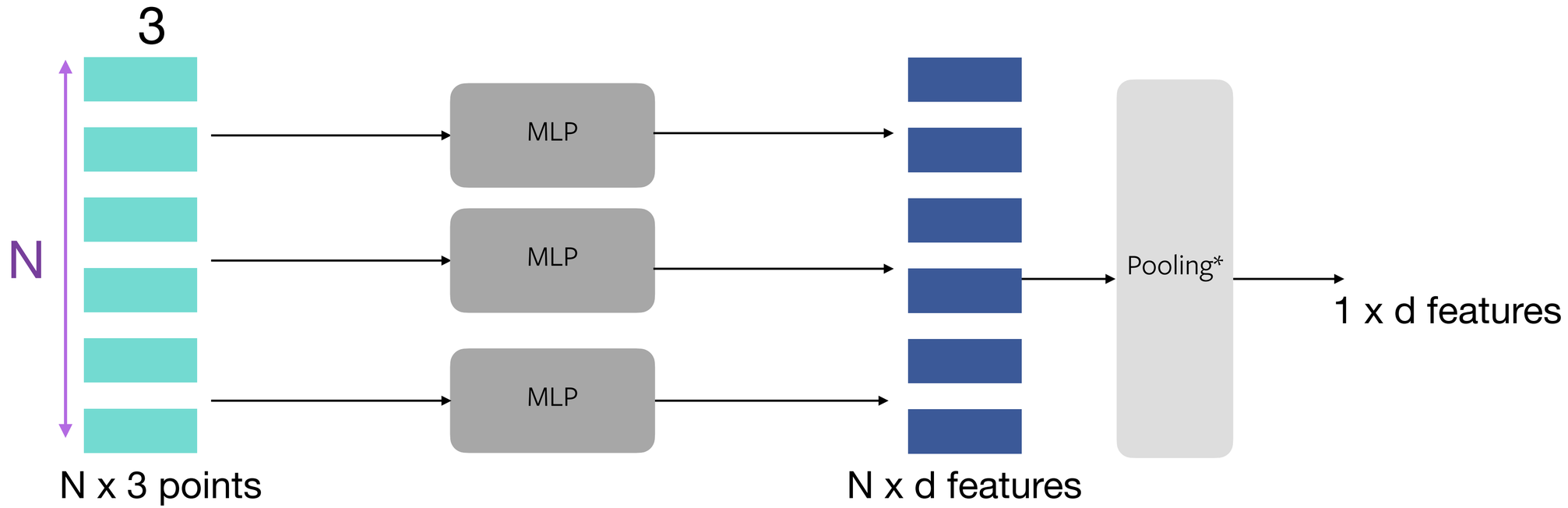
1 x d’ weights

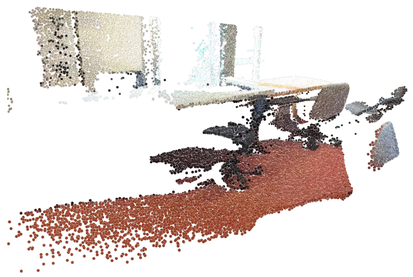
Per-element operation

Average/max

Global operation

Point Clouds: PointNet



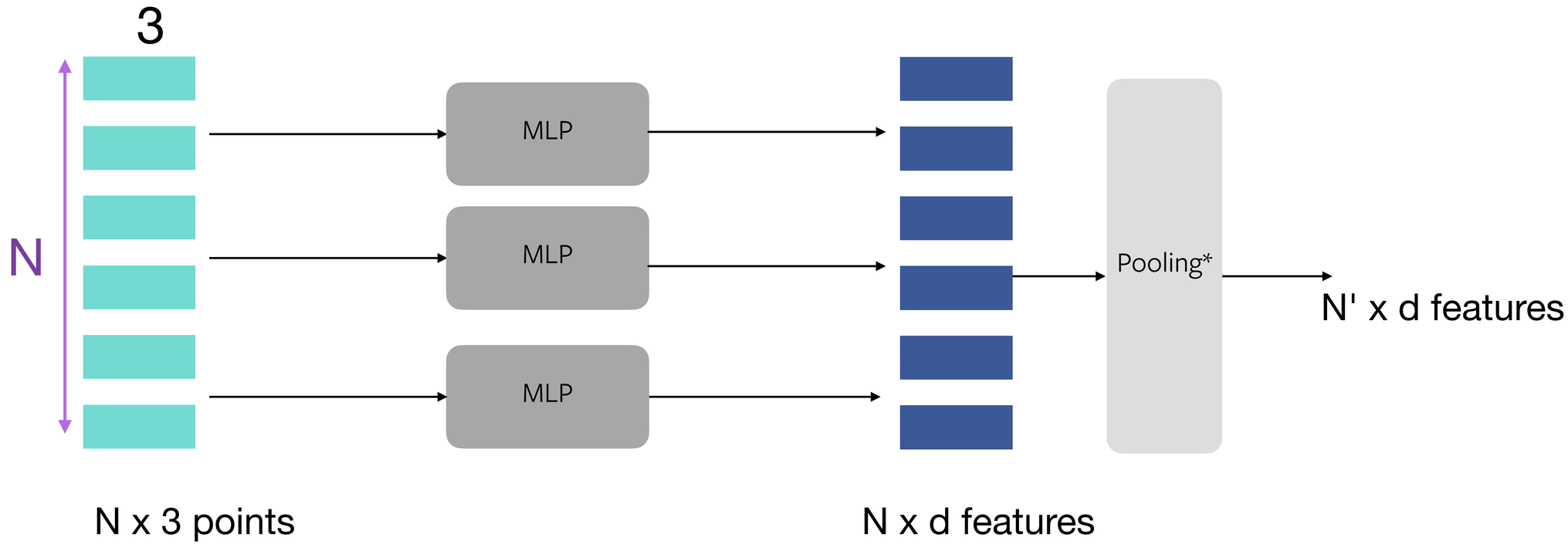


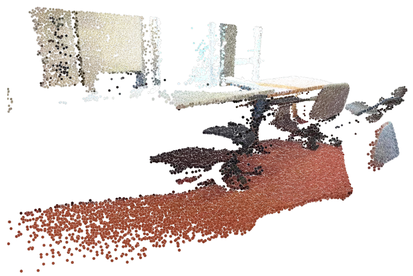
Per-element operation

Global operation

PointNet - Qi et al., 2016

Point Clouds: PointNet



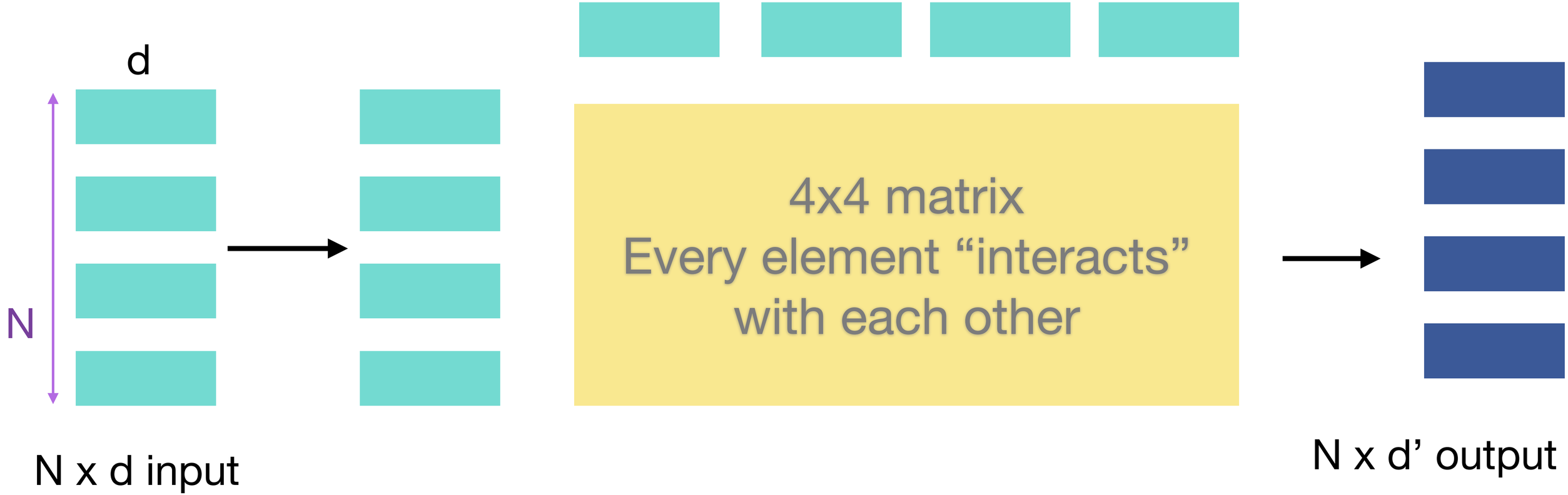


Apply permutation to the input —> Output is unchanged!

PointNet - Qi et al., 2016

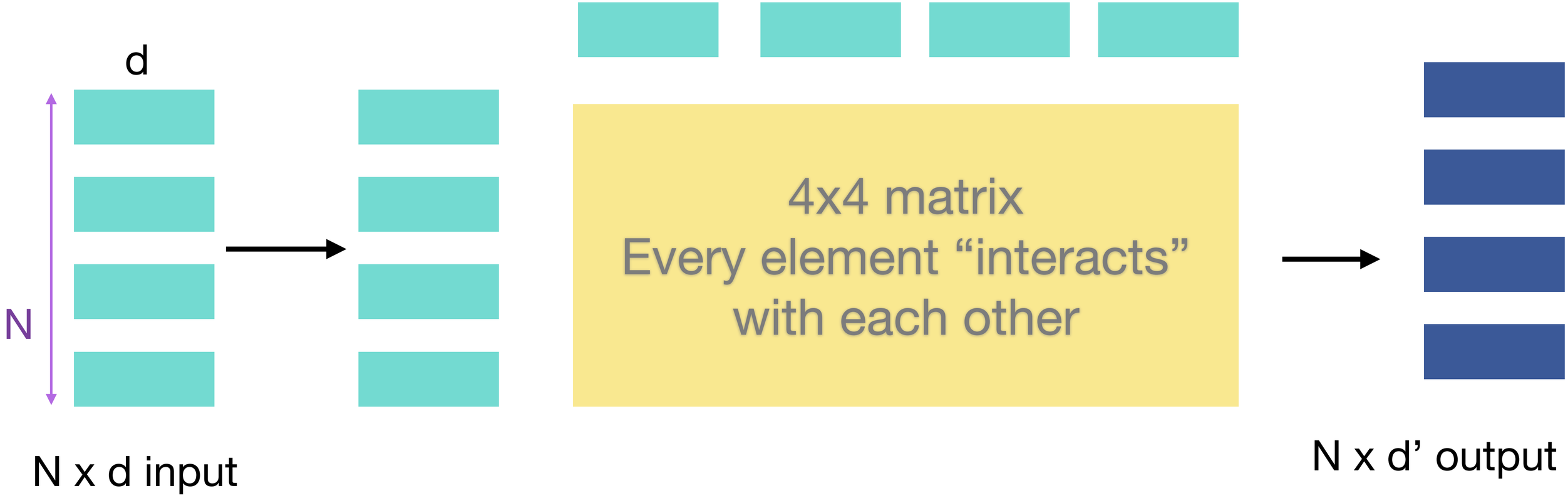
Self-Attention and Transformers

Self-Attention



Attention is all you need - Vaswani et al,, 2017

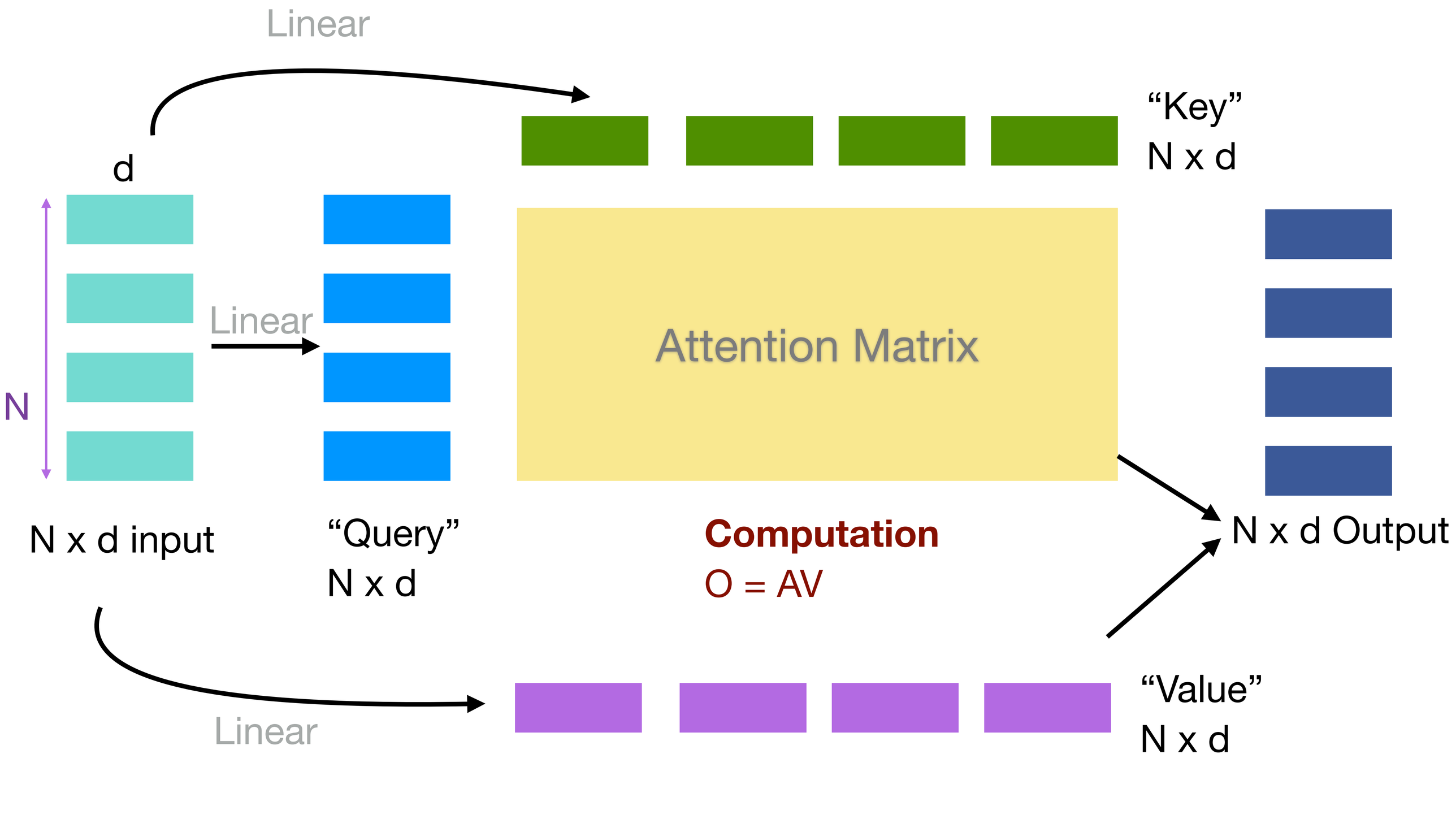
Self-Attention



Apply permutation to the input —> Output is unchanged!

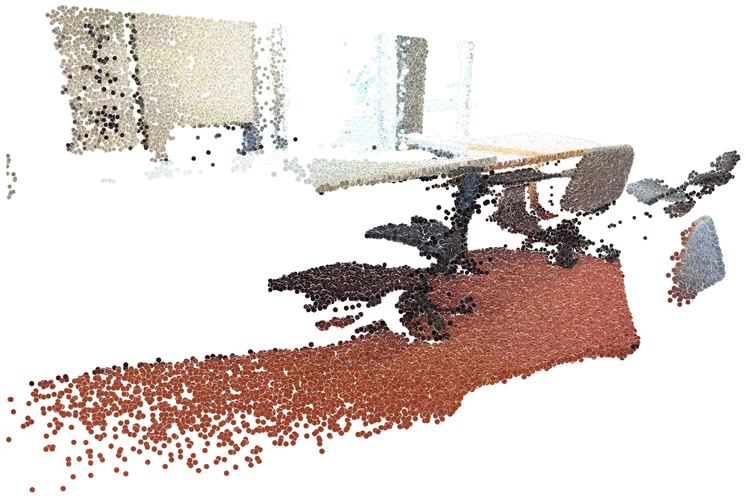
Computation

$A = \text{softmax}(QK^T/s)$

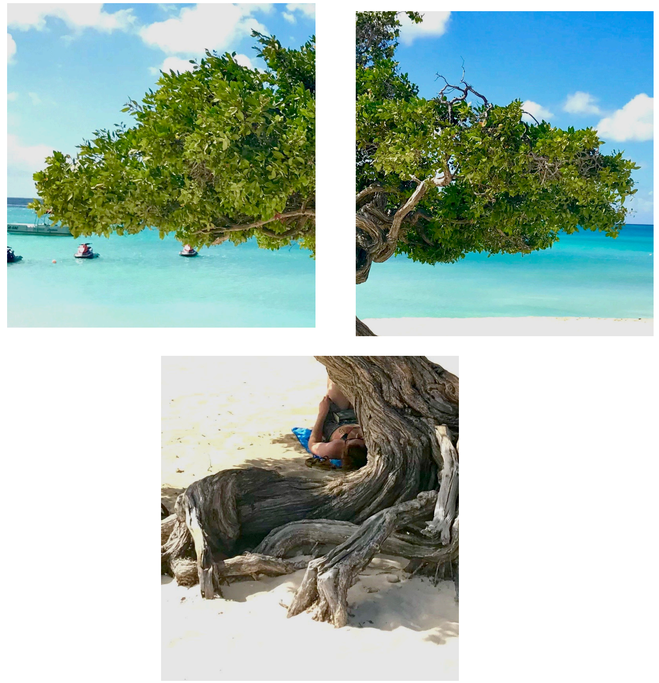

Linear
d
"Key"
N x d
N
Linear
Attention Matrix
N x d input
"Query"
N x d
Computation
O = AV
N x d Output
Linear
"Value"
N x d

Self-Attention: Versatile operation!

N x d input



N = number of points

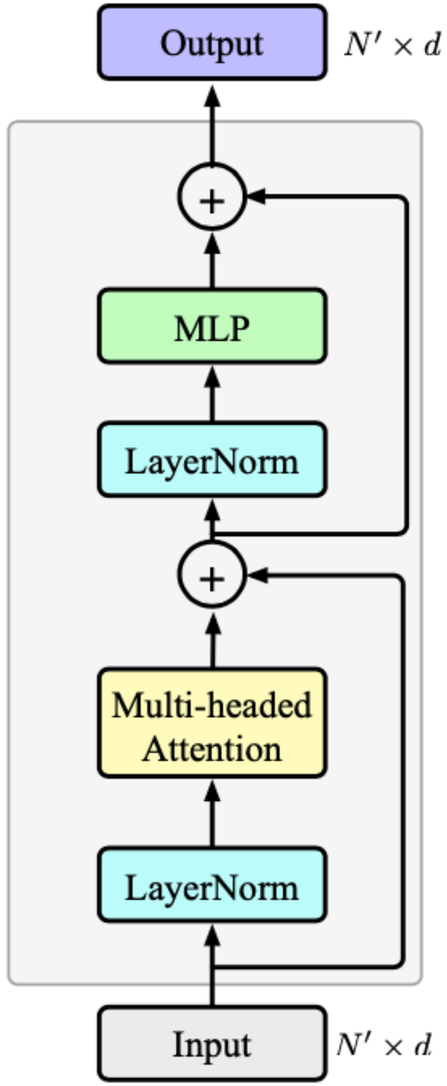


N = number of image patches

A sunny day

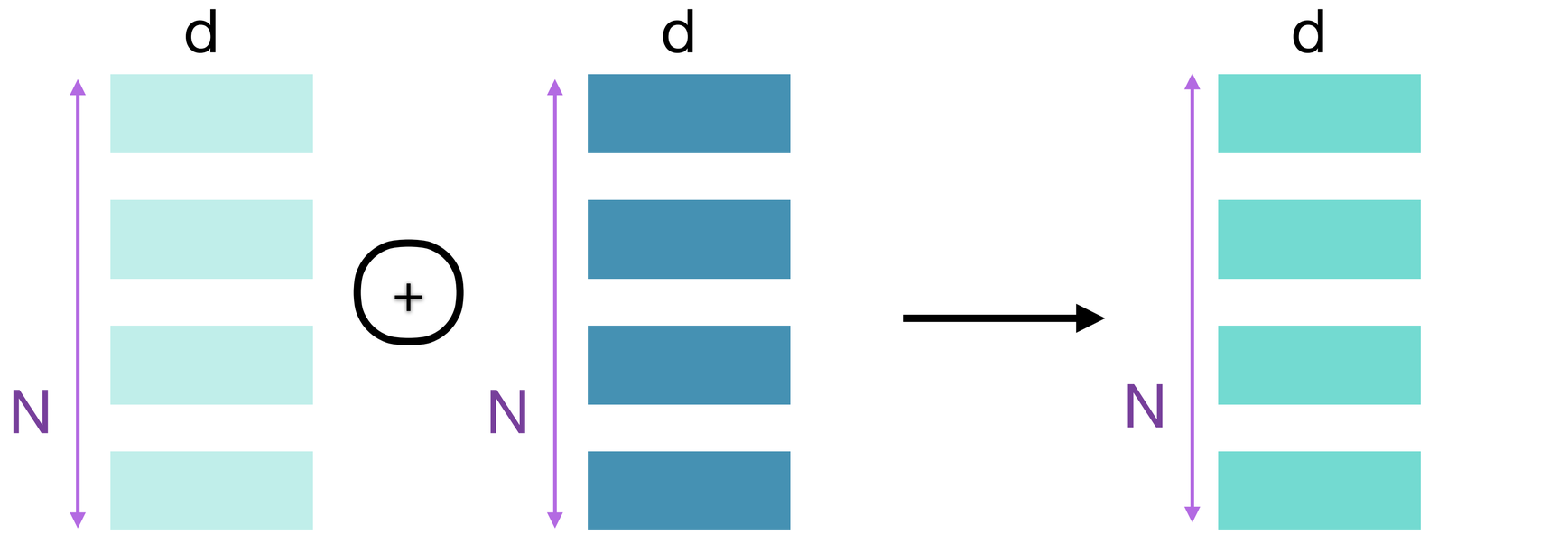
N = number of words

Transformer: Sequence of Self-Attention “blocks”



One Layer of Transformer Encoder

Transformer: Ordered inputs?

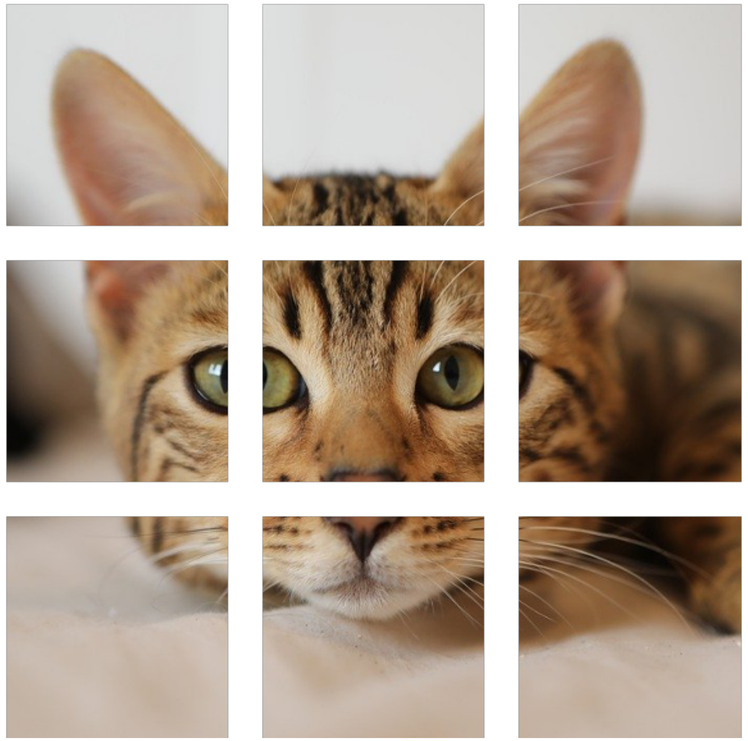


N x d input

N x d positional encoding

N x d input with positions

Vision Transformers



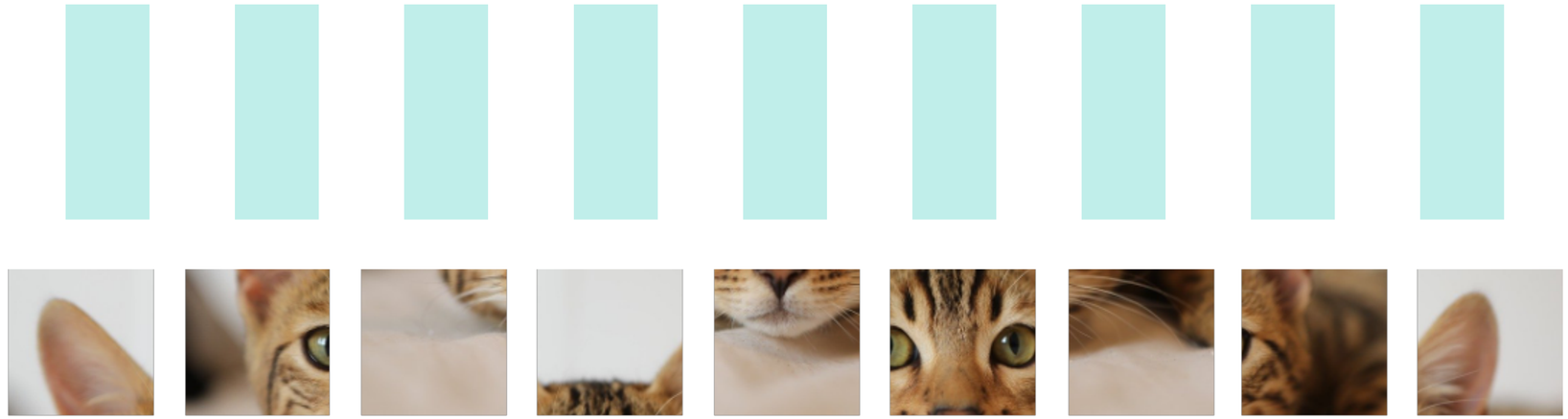
Dosovitskiy et al, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, ICLR 2021

Figure credit: Justin Johnson

Vision Transformers

Linear Projections

Create 16x16 patches

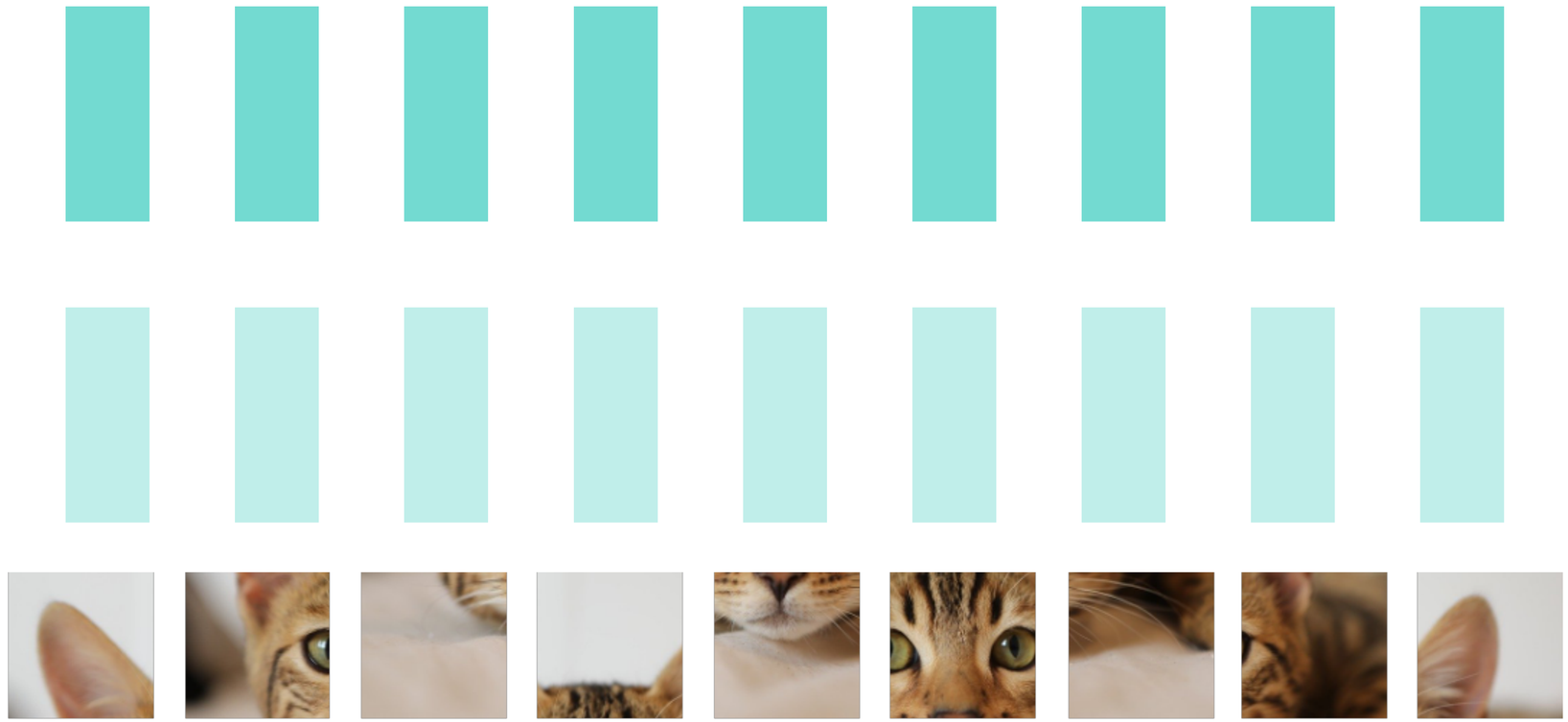


Vision Transformers

Add Pos. Encoding

Linear Projections

Create 16x16 patches



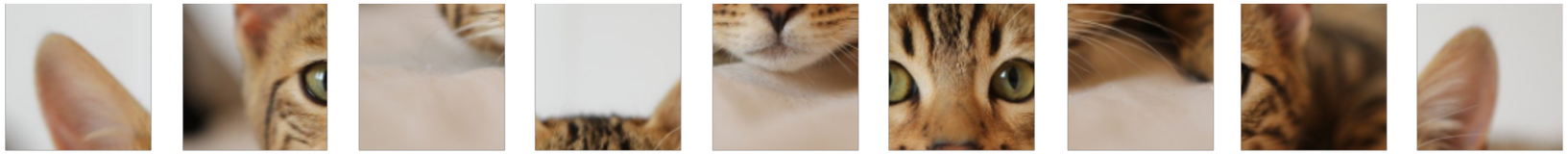
Output

Transformer

Add Pos. Encoding

Linear Projections

Create 16x16 patches



Output

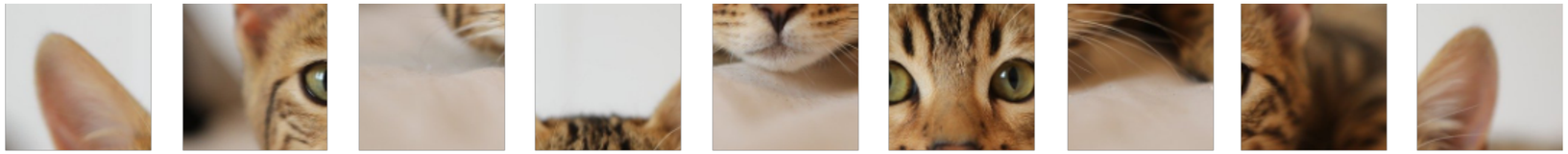
Transformer

Add Pos. Encoding

[CLS]

Linear Projections

Create 16x16 patches



General Models

Modality specific

Image

VGG,
Inception,
ResNet,
ViT,
Swin

Video

I3D,
R3D,
R(2+1)D,
ViViT,
VideoSwin

(Single-view) 3D

ShapeConv
DF^{2}Net,
TRecgNet,
PointTx

Need supervision per modality

Image

VGG,
Inception,
ResNet,
ViT,
Swin

Video

I3D,
R3D,
R(2+1)D,
ViViT,
VideoSwin

(Single-view) 3D

ShapeConv
DF²Net,
TRecgNet,
PointTx

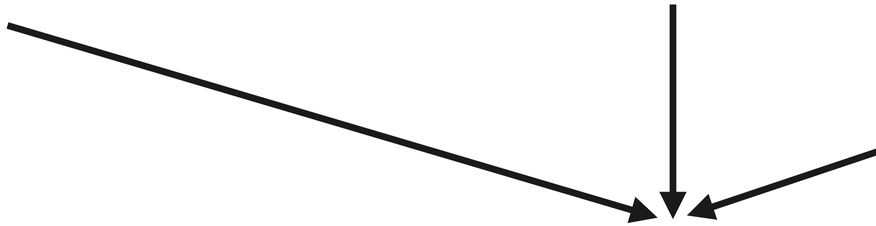
Recognize a pineapple in 3D => Need 3D supervision

Goal: Design a **single** “omnivorous” model

Image

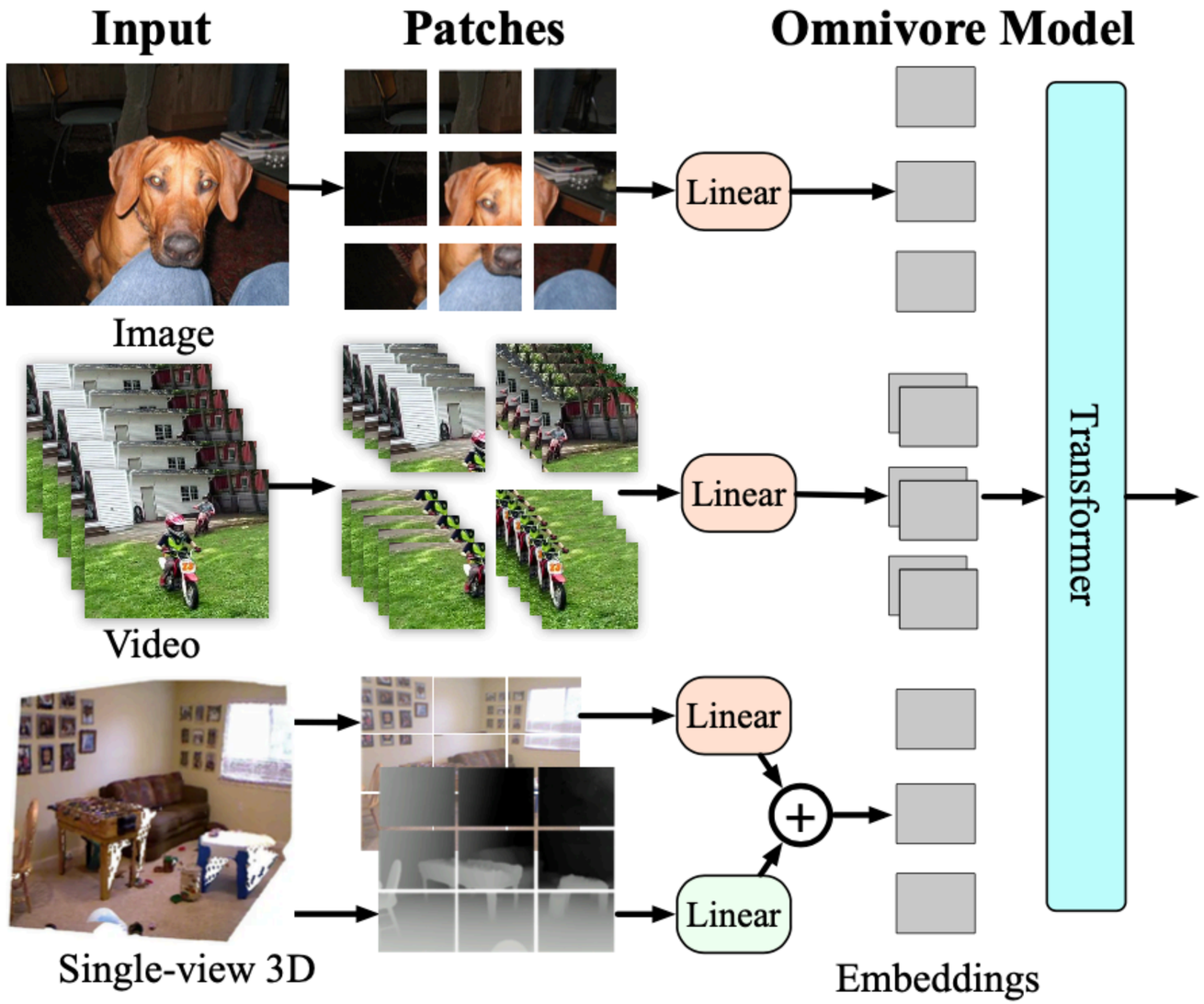
Video

(Single-view) 3D



Omnivore

Can we design a “universal” omnivorous model?

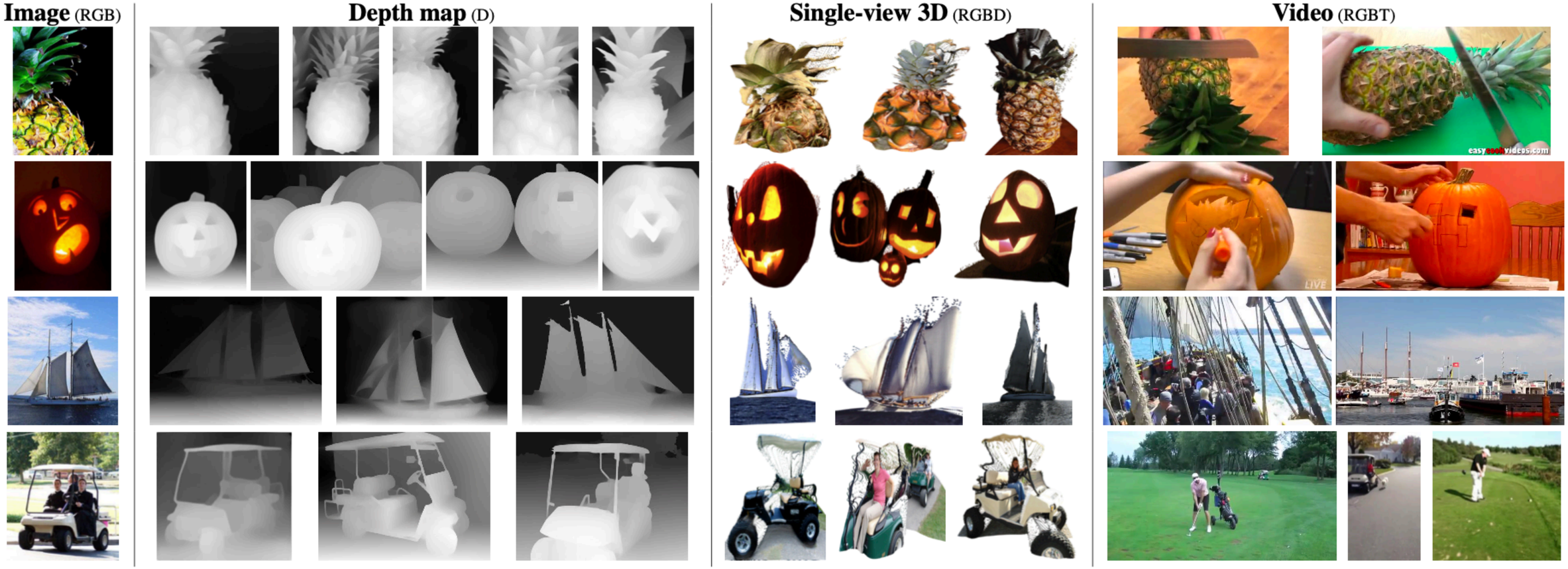


Omnivore: A Single Model for Many Visual Modalities: Girdhar et al., 2022

One model - works across datasets/tasks

Method	ImageNet-1K		Kinetics-400		SUN
	top-1	top-5	top-1	top-5	top-1
ImageSwin-T [49]	**81.2**	95.5	✗	✗	✗
VideoSwin-T [50]	✗	✗	78.8	93.6	✗
DepthSwin-T	✗	✗	✗	✗	**63.1**
OMNIVORE (Swin-T)	80.9	**95.5**	**78.9**	**93.8**	62.3
ImageSwin-S [49]	83.2	96.2	✗	✗	✗
VideoSwin-S [50]	✗	✗	80.6	94.5	✗
DepthSwin-S	✗	✗	✗	✗	**64.9**
OMNIVORE (Swin-S)	**83.4**	**96.6**	**82.2**	**95.4**	64.6
ImageSwin-B [49]	83.5	96.5	✗	✗	✗
VideoSwin-B [50]	✗	✗	80.6	94.6	✗
DepthSwin-B	✗	✗	✗	✗	64.8
OMNIVORE (Swin-B)	**84.0**	**96.8**	**83.3**	**95.8**	**65.4**

Emergent property - Shared visual representation



Thanks

Next: Self-supervised Learning in Vision