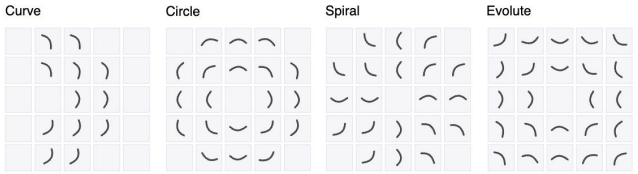
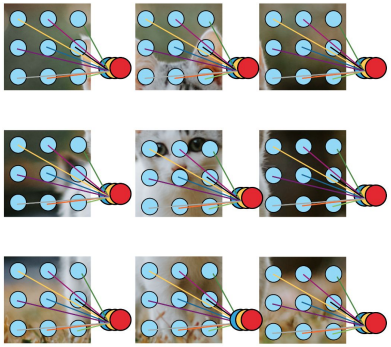
NYU CS-GY 6923 Machine Learning

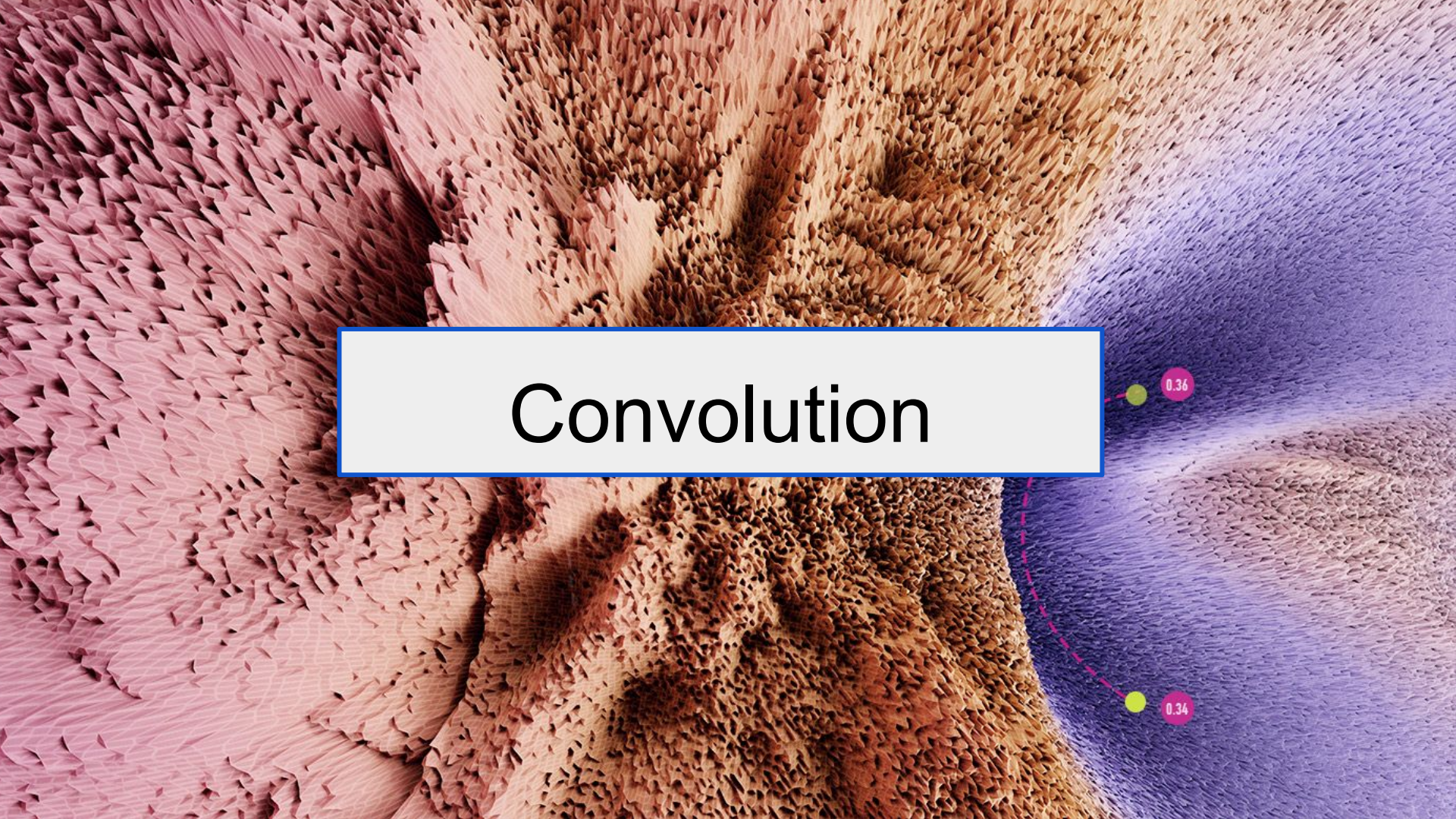
Prof. Pavel Izmailov

Today

- Convolution
- Interpretability
- Convolution beyond images
- Sequence Modeling



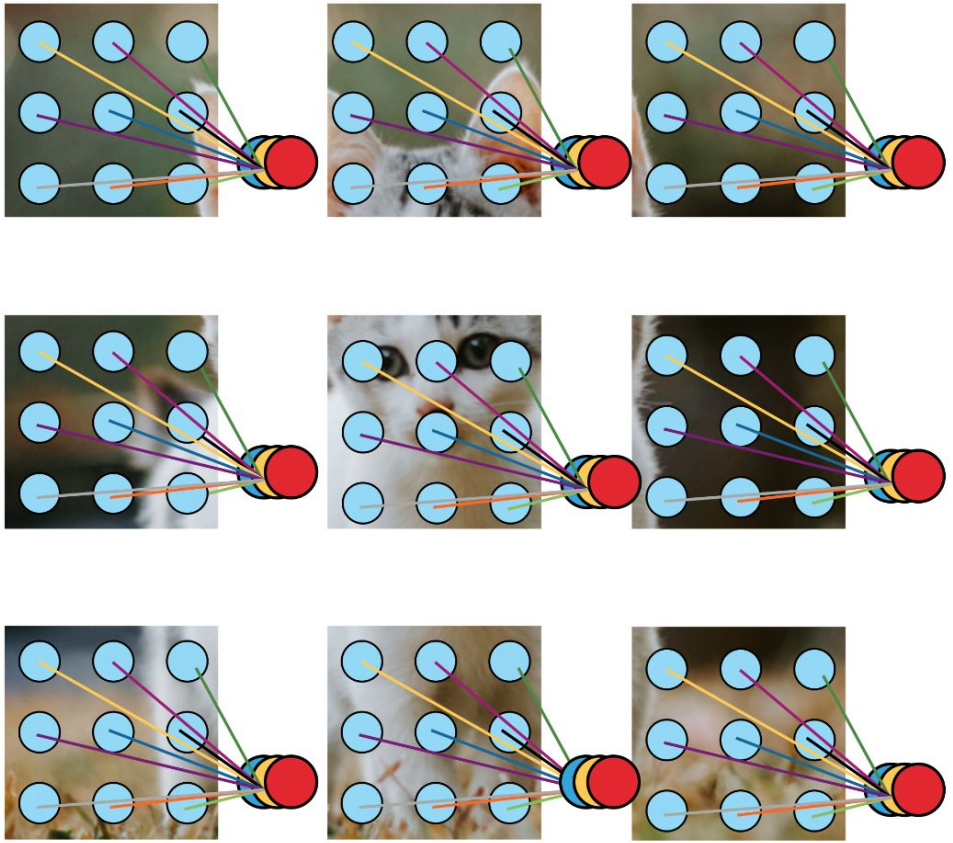



Convolution
0.36
0.34

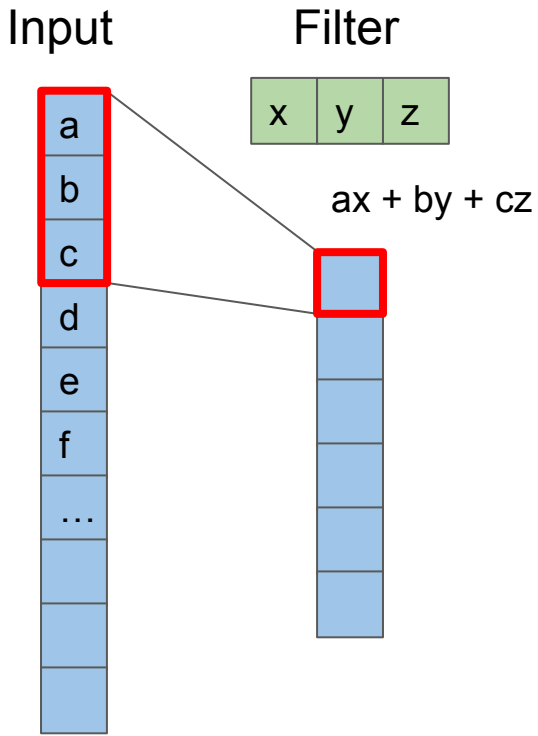
Convolutions

Reminder:

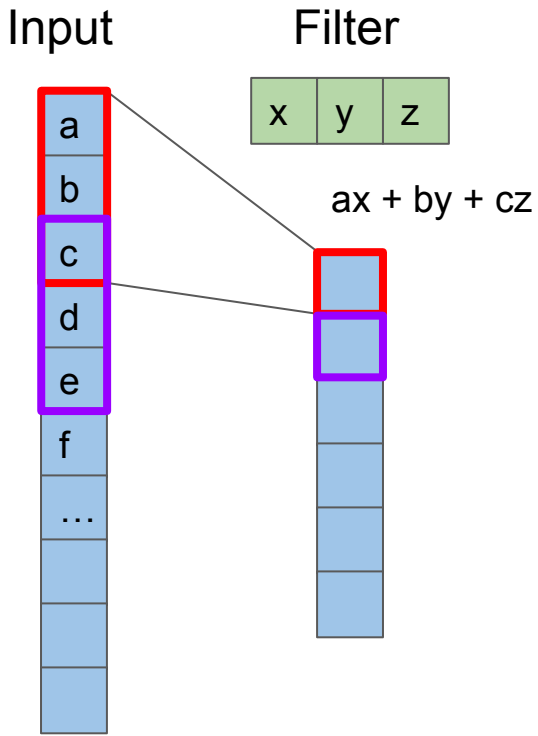
- Same fully-connected layer applied at each patch
- Convolution is local
- Convolution introduces shared parameters



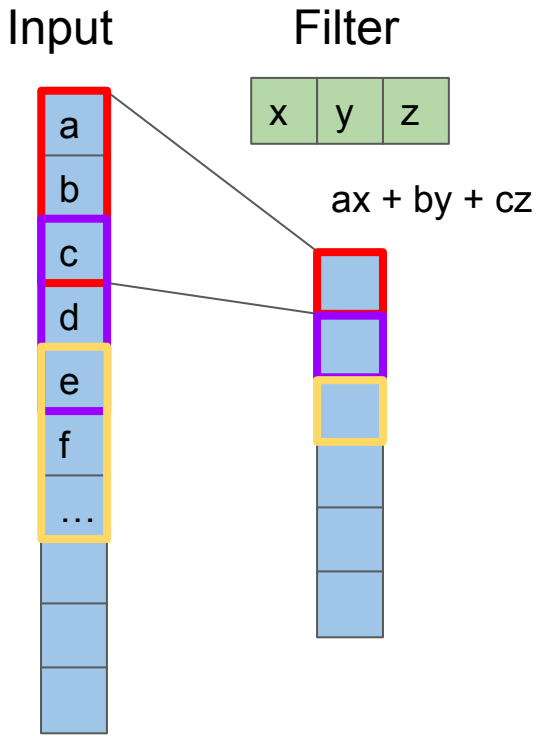
Beyond image classification



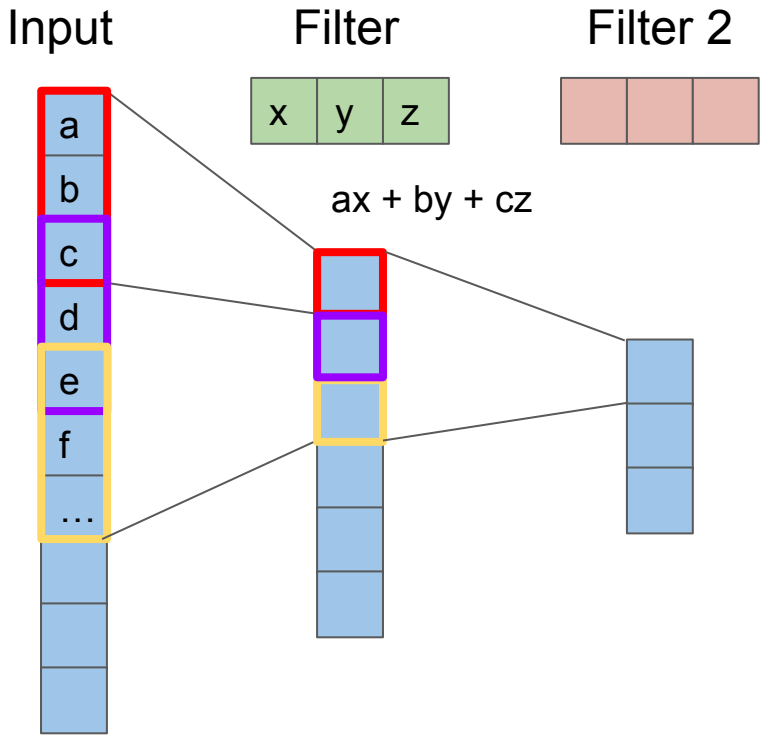
Beyond image classification



Beyond image classification

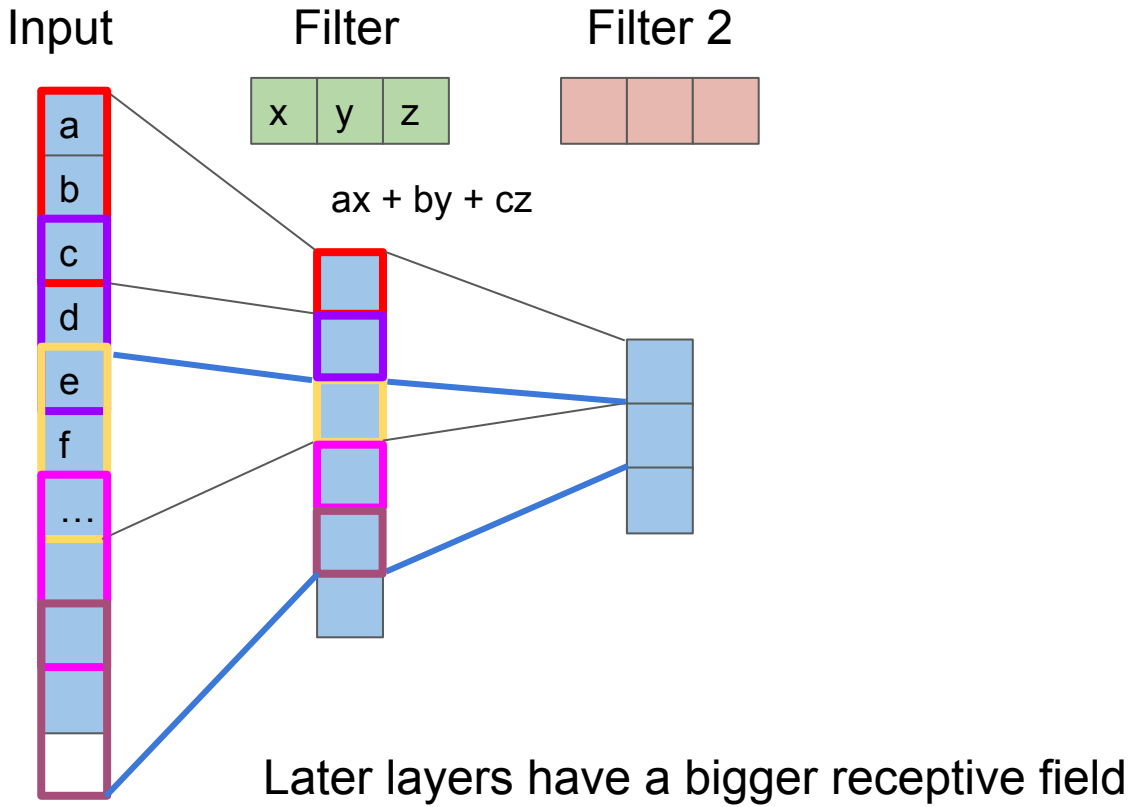


Beyond image classification



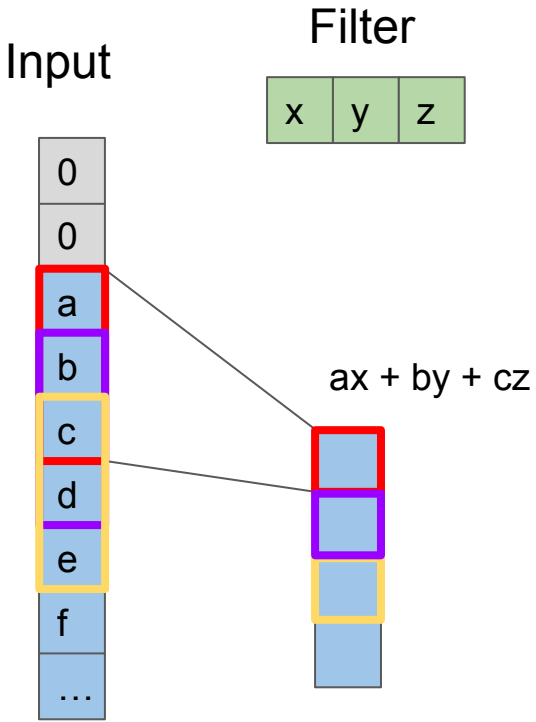
Later layers have a bigger receptive field

Beyond image classification



Later layers have a bigger receptive field

Beyond image classification

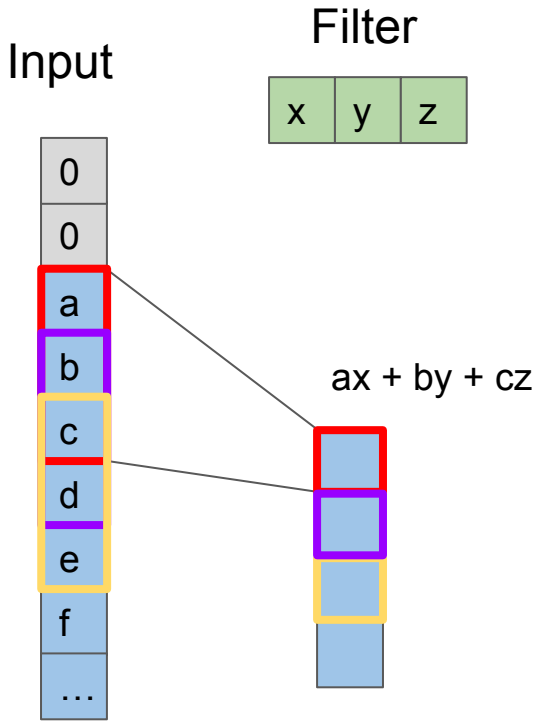


Stride 1, zero padding

$$\mathbf{H} = \begin{bmatrix} x & 0 & 0 & 0 & 0 \\ y & x & 0 & 0 & 0 \\ z & y & x & 0 & 0 \\ 0 & z & y & x & 0 \\ 0 & 0 & z & y & x \\ 0 & 0 & 0 & z & y \\ 0 & 0 & 0 & 0 & z \end{bmatrix}^T \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix}$$

Convolution is a special case of a linear layer, where the weight matrix is *Toeplitz*

Beyond image classification



Stride 1, zero padding

$$\mathbf{H} = \begin{bmatrix} x & 0 & 0 & 0 & 0 \\ y & x & 0 & 0 & 0 \\ z & y & x & 0 & 0 \\ 0 & z & y & x & 0 \\ 0 & 0 & z & y & x \\ 0 & 0 & 0 & z & y \\ 0 & 0 & 0 & 0 & z \end{bmatrix}$$

Two main features:

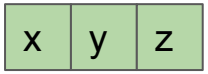
- Locality
- Weight sharing

Convolution is a special case of a linear layer, where the weight matrix is *Toeplitz*

Beyond image classification

Input

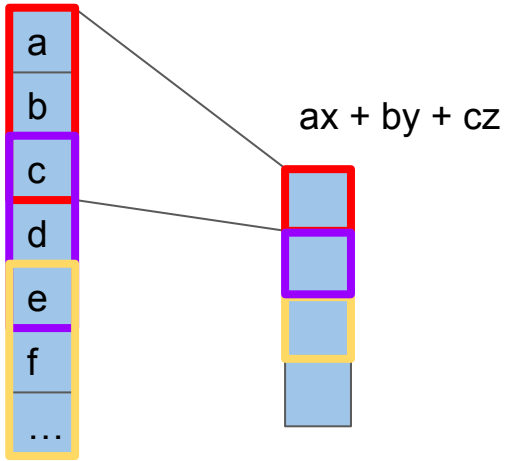
Filter



Stride 2, no padding

Two main features:

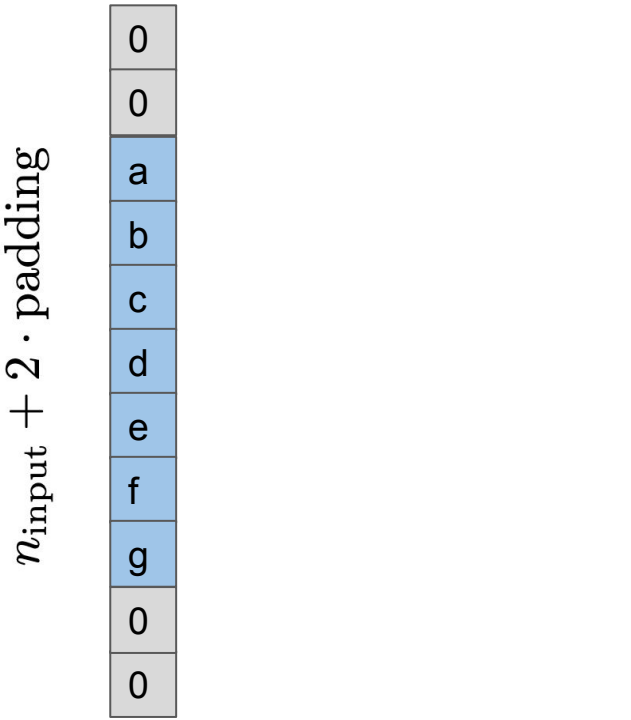
- Locality
- Weight sharing



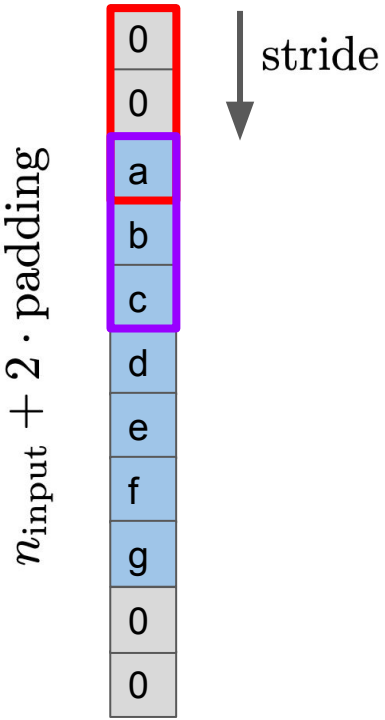
$$\mathbf{H}_{\text{valid, stride=2}} = \begin{bmatrix} z & y & x & 0 & 0 & 0 & 0 \\ 0 & 0 & z & y & x & 0 & 0 \\ 0 & 0 & 0 & 0 & z & y & x \end{bmatrix}$$

Convolution is a special case of a linear layer, where the weight matrix is *Toeplitz*

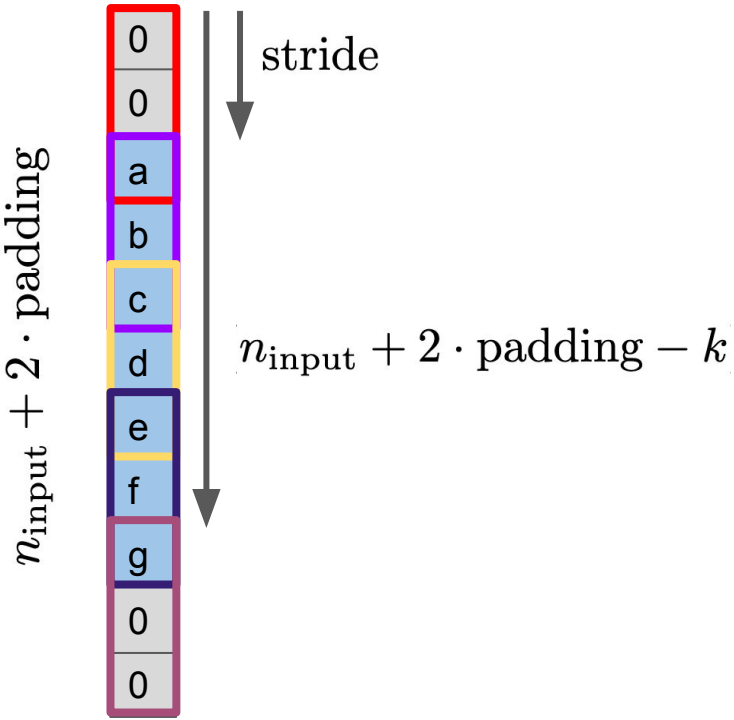
Convolutions: output size



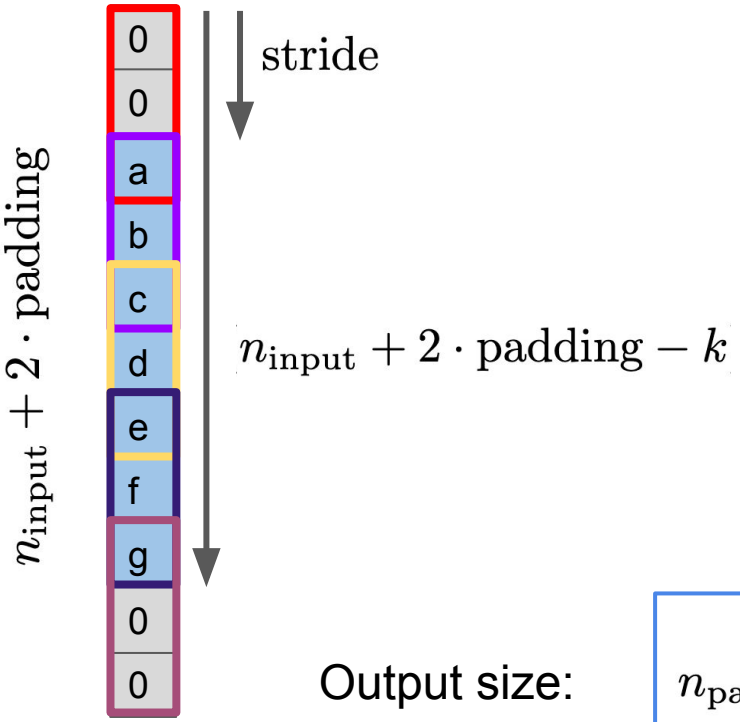
Convolutions: output size



Convolutions: output size

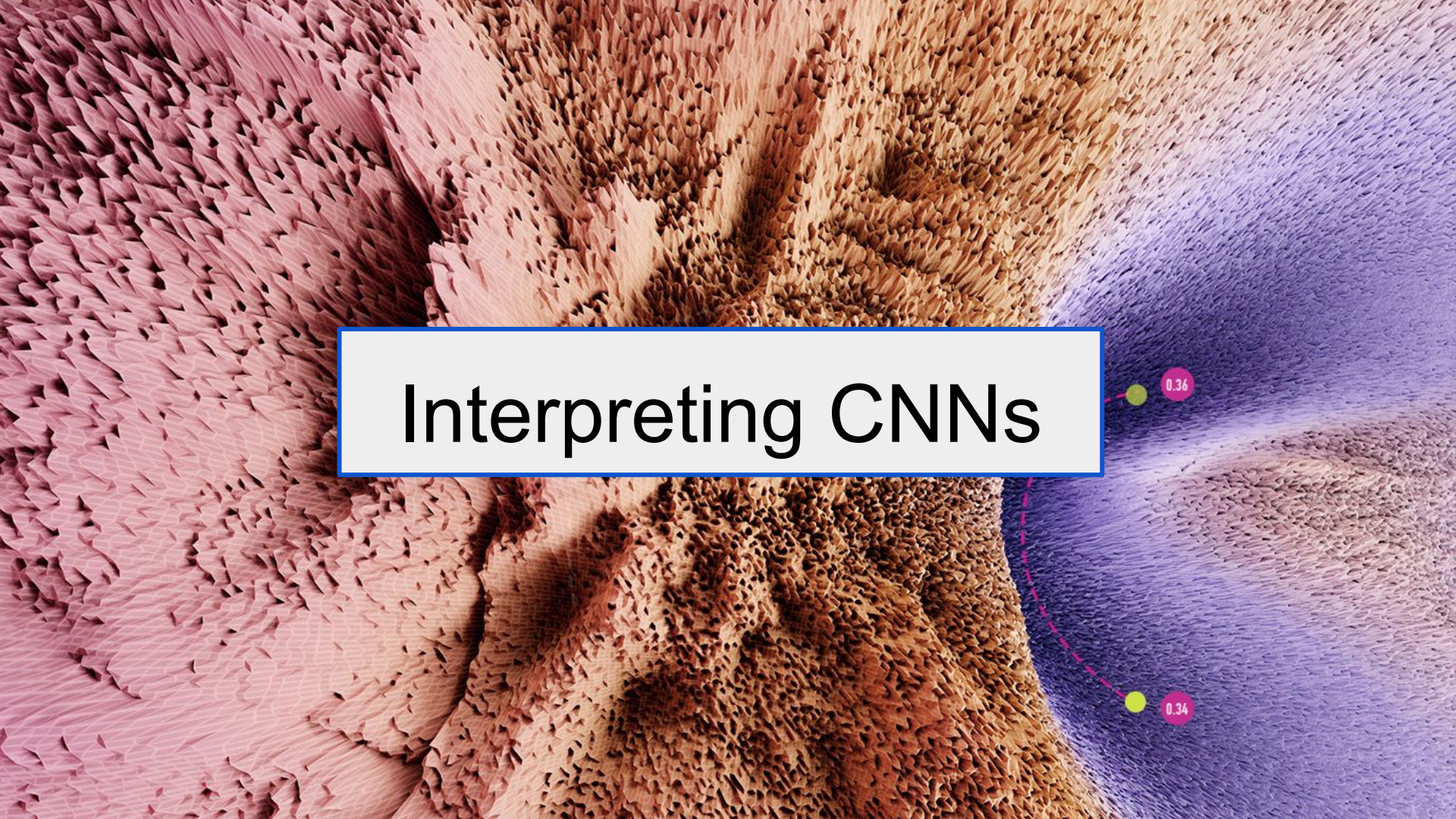


Convolutions: output size



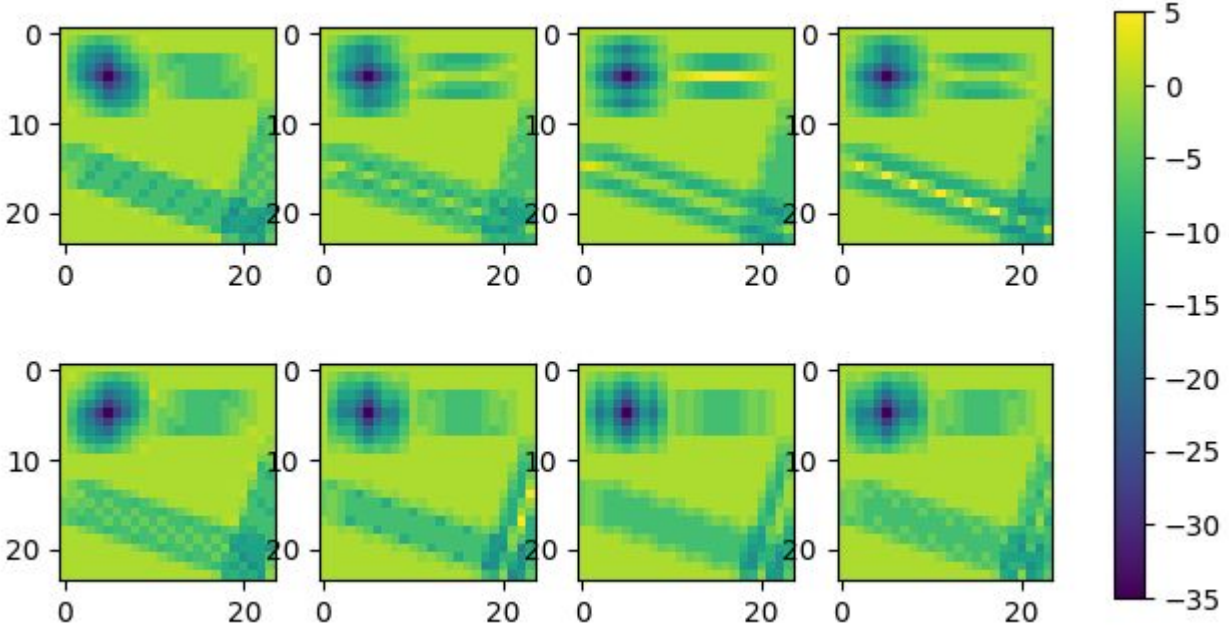
Output size:

$$n_{\text{patches}} = \frac{(n_{\text{input}} + 2 \cdot \text{padding} - k)}{\text{stride}} + 1$$

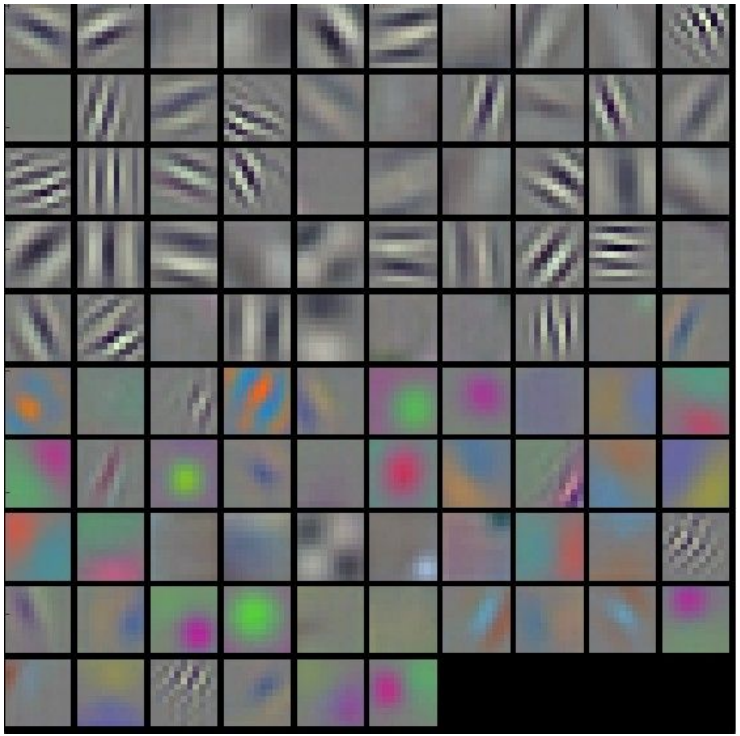

Interpreting CNNs
0.36
0.34

Interpretability

Let’s go through a demo: [cnn-interp-demo.ipynb]



Interpretability



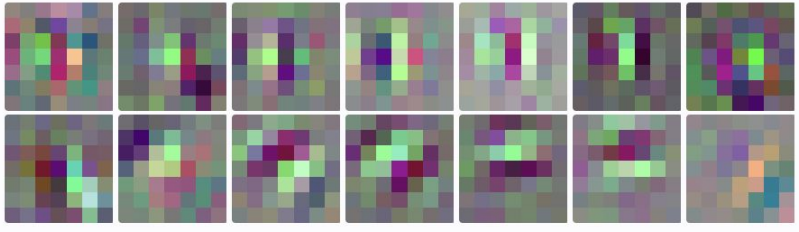
We can visualize and interpret the filters in the first layer:

- Edge detectors at different slopes
- Color blob detectors

https://cs231n.github.io/understanding-cnn/

Interpretability

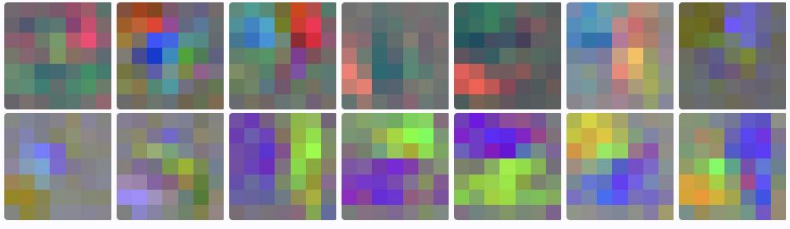
Gabor Filters 44%



Show all 28 neurons.

Gabor filters are a simple edge detector, highly sensitive to the alignment of the edge. They're almost universally found in the fist layer of vision models. Note that Gabor filters almost always come in pairs of negative reciprocals.

Color Contrast 42%



Show all 27 neurons.

These units detect a color one side of their receptive field, and the opposite color on the other side. Compare to later color contrast (`conv2d1`, `conv2d2`, `mixed3a`, `mixed3b`).

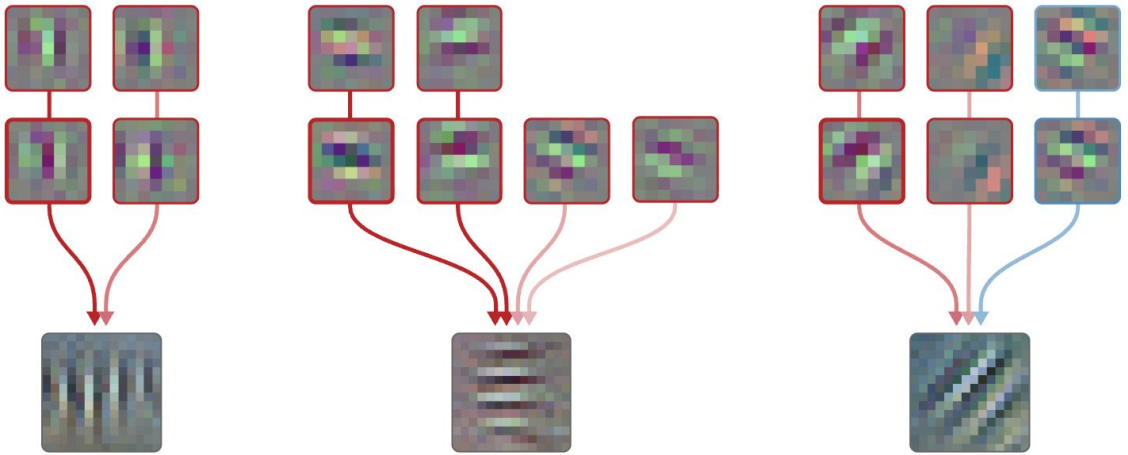
We will be looking at patches that maximally activate different filters.

https://distill.pub/2020/circuits/early-vision/

Other Units 14%

Units that don't fit in another category.

Interpretability



All neurons in the previous layer with at least 30% of the max weight magnitude are shown, both **positive (excitation)** and **negative (inhibition)**. Click on a neuron to see its forwards and backwards weights.

Later layers can put together multiple filters to detect more complex features. The second layer is a 1x1 conv.

https://distill.pub/2020/circuits/early-vision/

Interpretability

Low Frequency 27%

Show all 17 neurons.

These units seem to respond to lower-frequency edge patterns, but we haven't studied them very carefully.

Gabor Like 17%

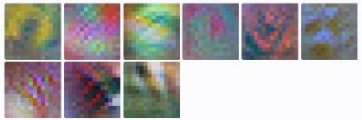
Show all 11 neurons.

These units respond to edges stimuli, but seem to respond to a wider range of orientations, and also respond to color contrasts that align with the edge. We haven't studied them very carefully.

Color Contrast 16%

These units detect a color on one side of the receptive field, and a different color on the opposite side. Composed of lower-level color contrast detectors, they often respond to color transitions in a range of translation and orientation variations. Compare to earlier color contrast (`conv2d0`) and later color contrast (`conv2d2`, `mixed3a`, `mixed3b`).

Multicolor 14%

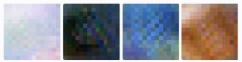


These units respond to mixtures of colors without an obvious strong spatial structure preference.

Complex Gabor 14%

Like Gabor Filters, but fairly invariant to the exact position, formed by adding together multiple Gabor detectors in the same orientation but different phases. We call these 'Complex' after complex cells in neuroscience.

Color 6%



Two of these units seem to track brightness (bright vs dark), while the other two units seem to mostly track hue, dividing the space of hues between them. One responds to red/orange/yellow, while the other responds to purple/blue/turqoise. Unfortunately, their circuits seem to heavily rely on the existence of a Local Response Normalization layer after `conv2d0`, which makes it hard to reason about.

Other Units 5%

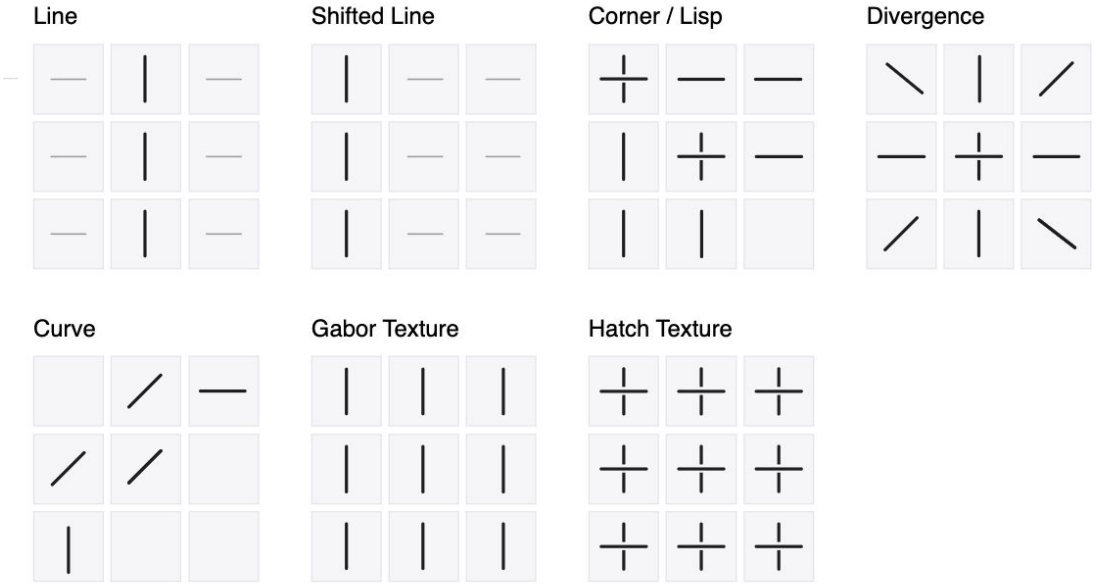
Units that don't fit in another category.

hatch 2%

This unit detects Gabor patterns in two orthogonal directions, selecting for a "hatch" pattern.

https://distill.pub/2020/circuits/early-vision/

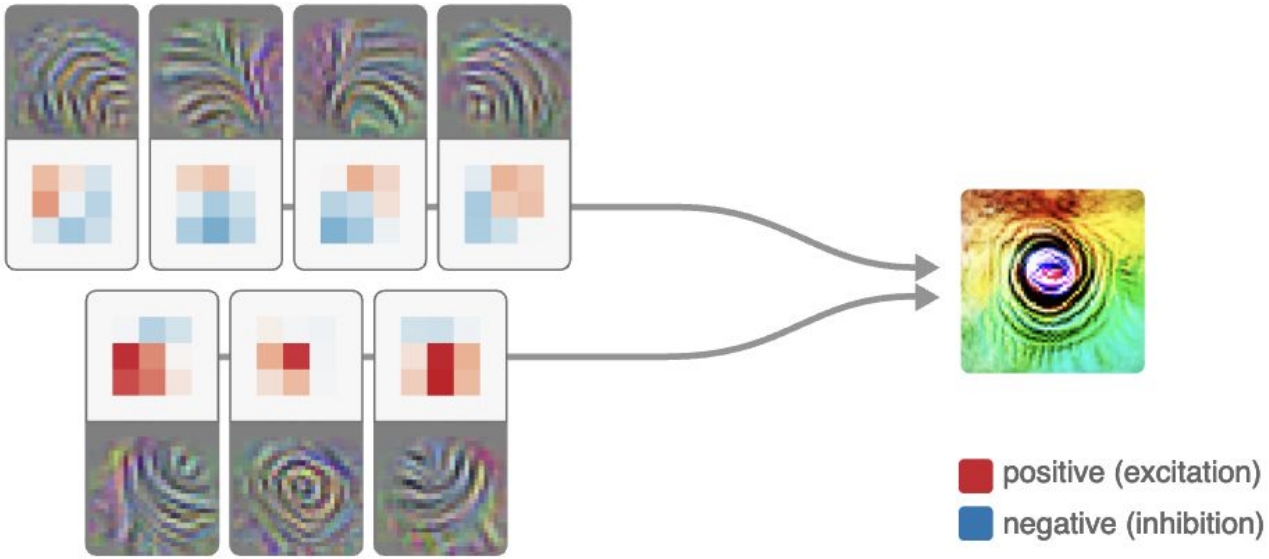
Interpretability



Layer 3 is a 3x3 convolution, and it can put together multiple line detectors to detect more complex shapes.

https://distill.pub/2020/circuits/early-vision/

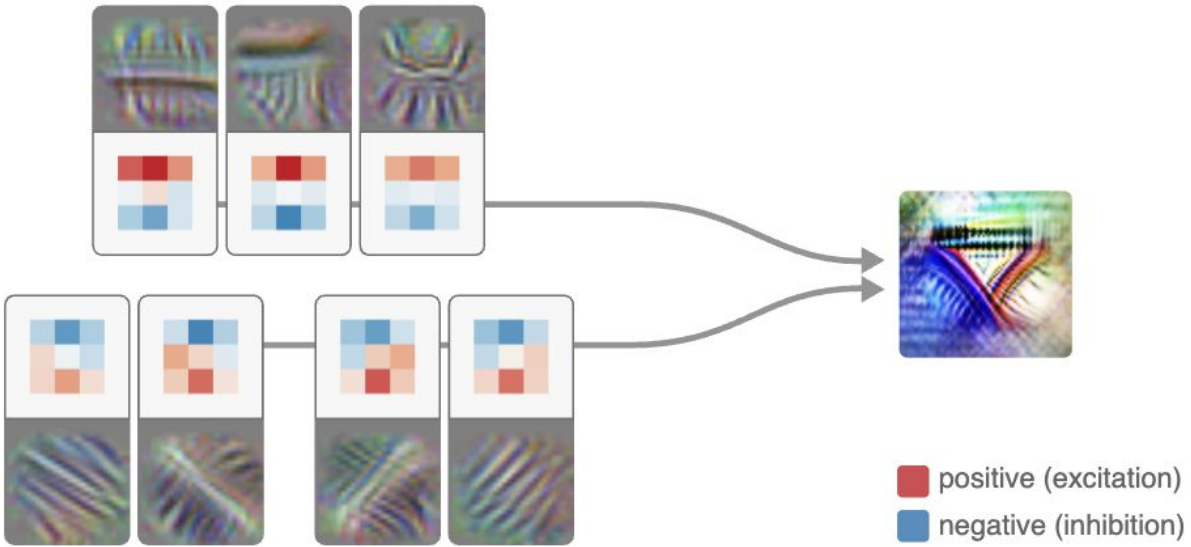
Interpretability



Example of weights for different filters that detect a more complex shape.

https://distill.pub/2020/circuits/early-vision/

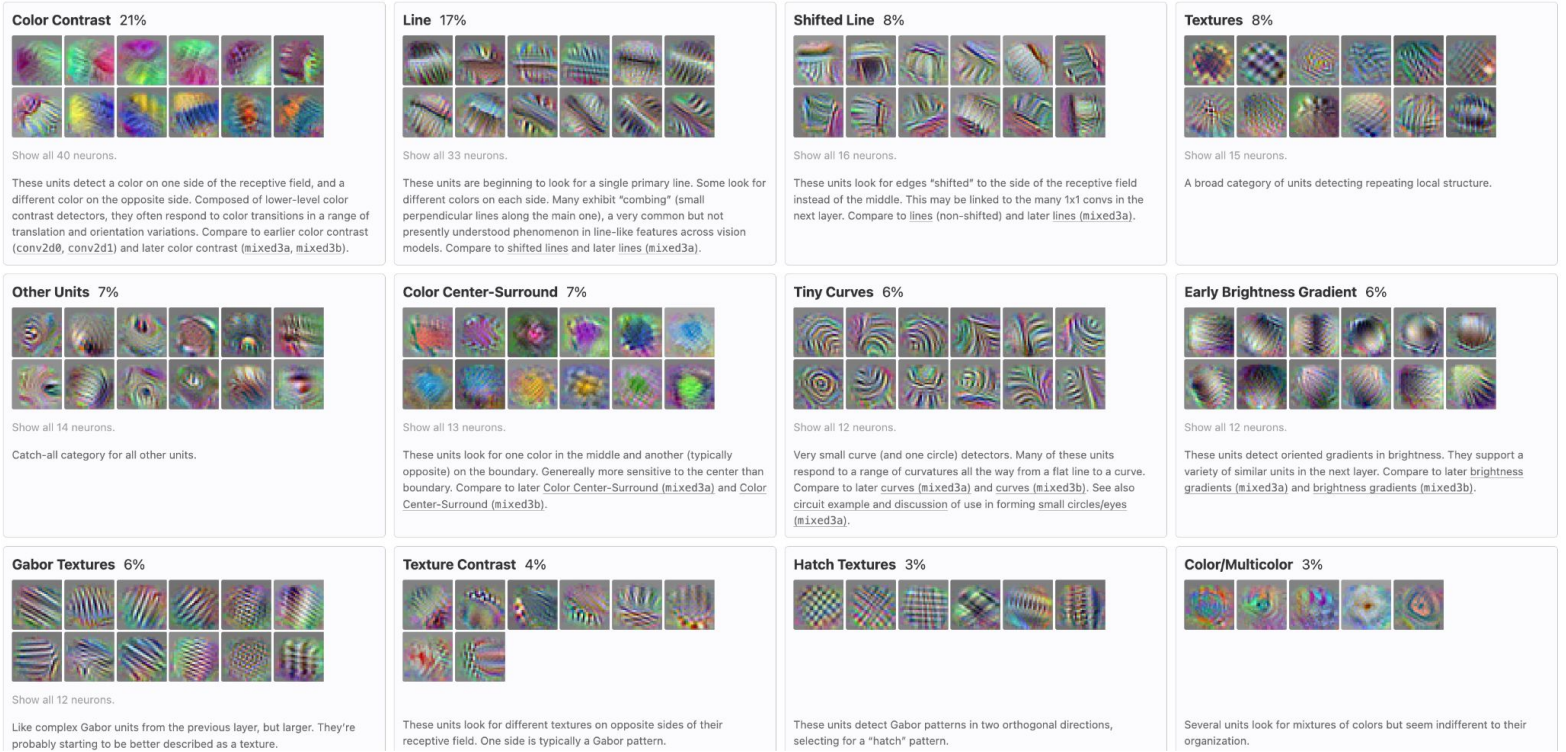
Interpretability



Example of weights for different filters that detect a more complex shape.

https://distill.pub/2020/circuits/early-vision/

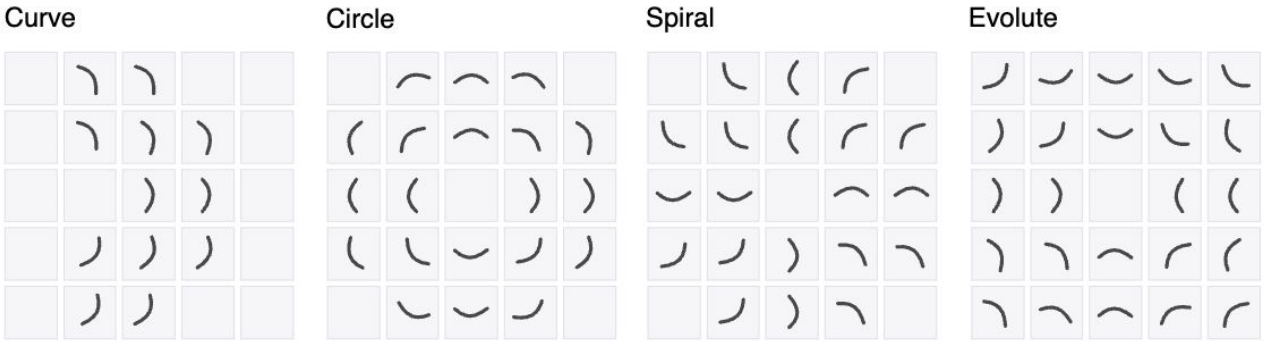
Interpretability



Layer 3.

https://distill.pub/2020/circuits/early-vision/

Interpretability



Later layers continue to put together more complex features.

https://distill.pub/2020/circuits/early-vision/

Interpretability

Boundary 8%

Show all 36 neurons.

These units use multiple cues to detect the boundaries of objects. They vary in orientation, detecting convex/concave/straight boundaries, and detecting artificial vs fur foregrounds. Cues they rely on include line detectors, high-low frequency detectors, and color contrast.

Proto-Head 3%

Show all 12 neurons.

The tiny eye detectors, along with texture detectors for fur, hair and skin developed at the previous layer enable these early head detectors, which will continue to be refined in the next layer.

Generic, Oriented Fur 2%

We don't typically think of fur as an oriented feature, but it is. These units detect fur parting in various ways, much like how hair on your head parts.

Curves 2%

The third iteration of curve detectors. They detect larger radii curves than their predecessors, and are the first to not slightly fire for curves rotated 180 degrees. Compare to the earlier curves (conv2d2) and curves (mixed3a).

See the full paper on curve detectors.

Divots 2%

Curve-like detectors for sharp corners or bumps.

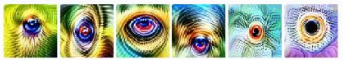
Square / Grid 2%

Units detecting grid patterns.

Brightness Gradients 1%

These units detect brightness gradients. This is their third iteration; compare to earlier brightness gradients (conv2d2) and brightness gradients (mixed3a).

Eyes 1%



Again, we continue to see eye detectors quite early in vision. Note that several of these detect larger eyes than the earlier eye detectors (mixed3a). In the next layer, we see much larger scale eye detectors again.

Shallow Curves 1%

Detectors for curves with wider radii than regular curve detectors.

Curve Shapes 1%

Simple shapes created by composing curves, such as spirals and S-curves.

Circles / Loops 1%

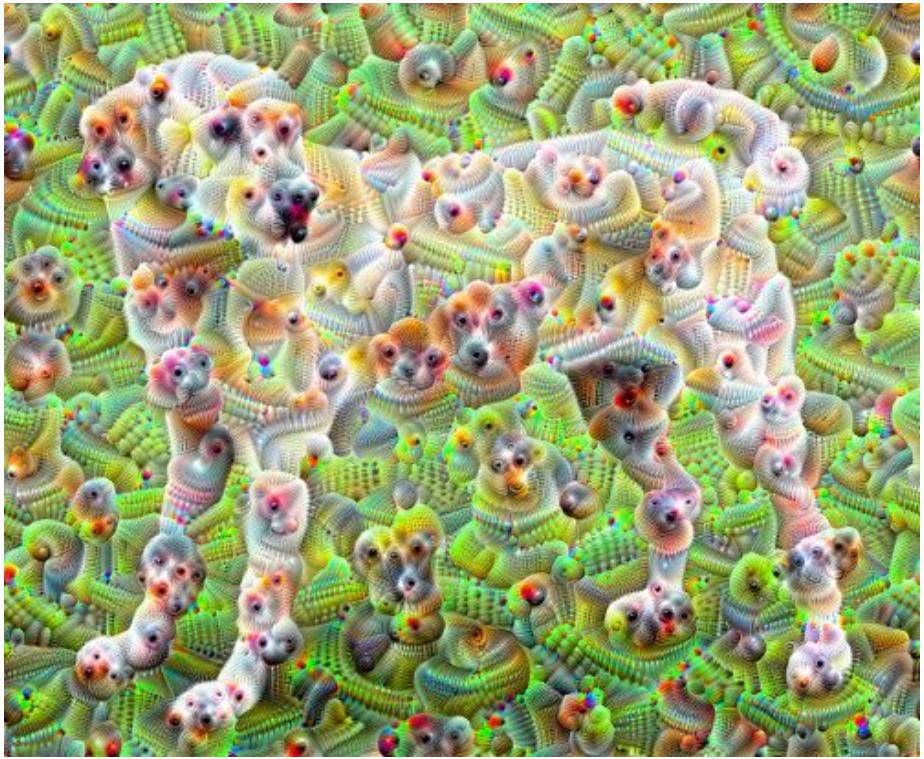
Piece together curves in a circle or partial circle. Opposite of evolute.

Circle Cluster 1%

Units detecting circles and curves without necesarily requiring spatial coherrence.

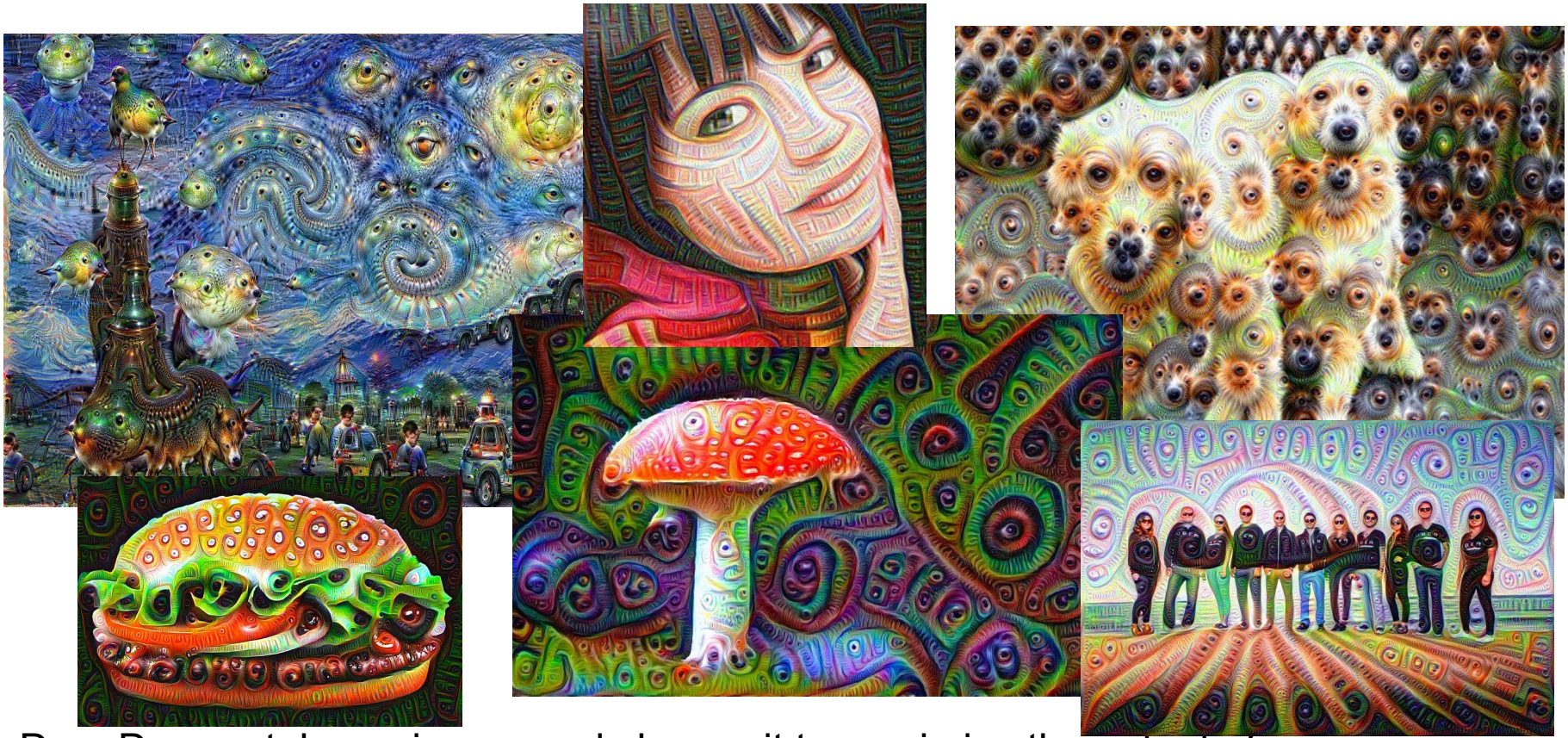
https://distill.pub/2020/circuits/early-vision/

Interpretability



DeepDream: take an image and change it to maximize the output of a neuron.

Interpretability



DeepDream: take an image and change it to maximize the output of a neuron.

Bigger vs Smaller Kernels

Smaller kernels (1x1, 3x3):

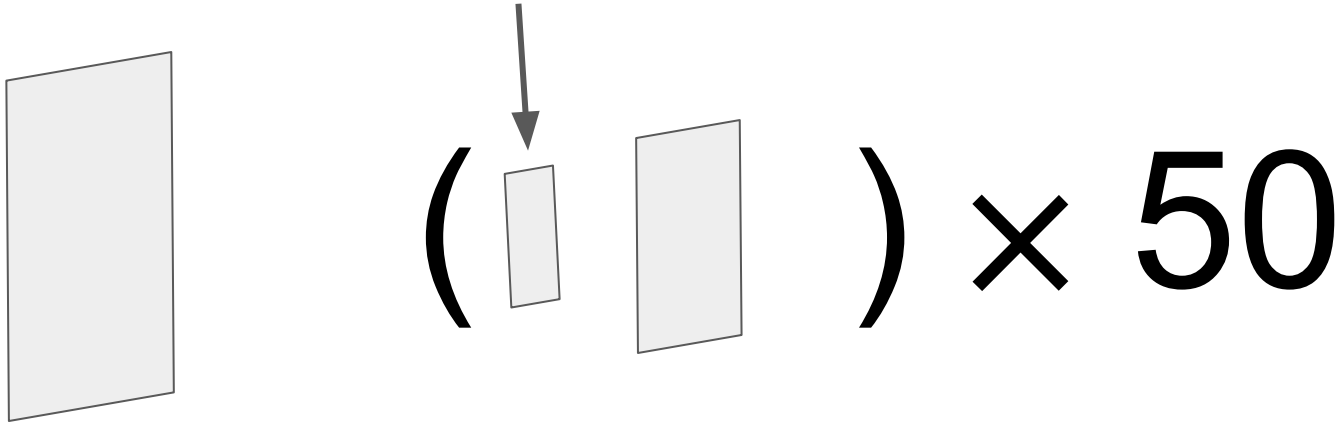
- Capture local, fine-grained features (edges, textures, small patterns)
- More computationally efficient
- Need to stack more layers to build up a large receptive field
- Better at learning complex, hierarchical representations through depth

Larger kernels (5x5, 7x7, 11x11):

- Capture broader spatial context in a single layer
- Can detect larger patterns/structures immediately
- More parameters and computation per layer
- Each filter can look at more of the image at once

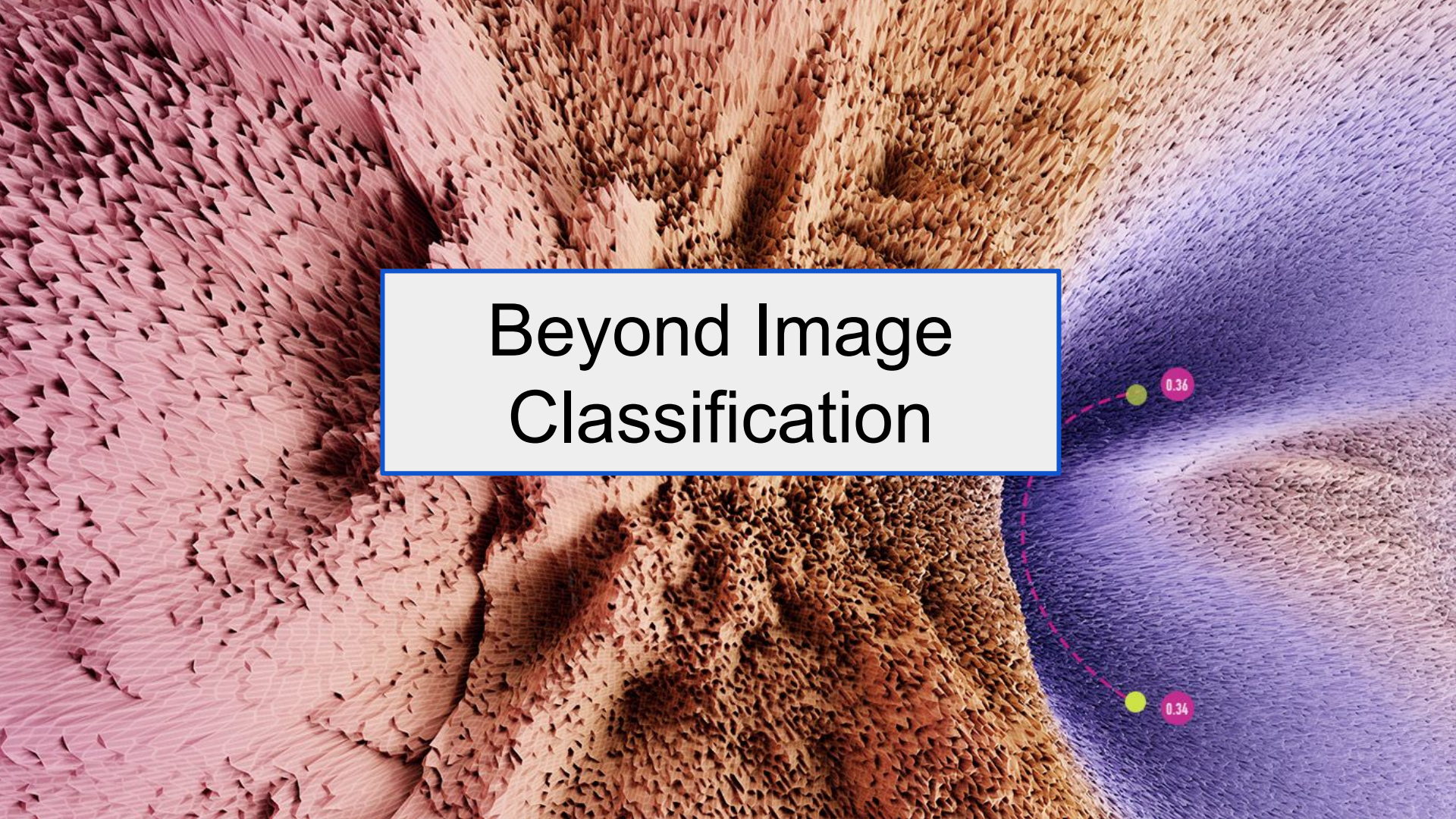
Bigger vs Smaller Kernels

1x1 are applied independently to each pixel, just operate on channels



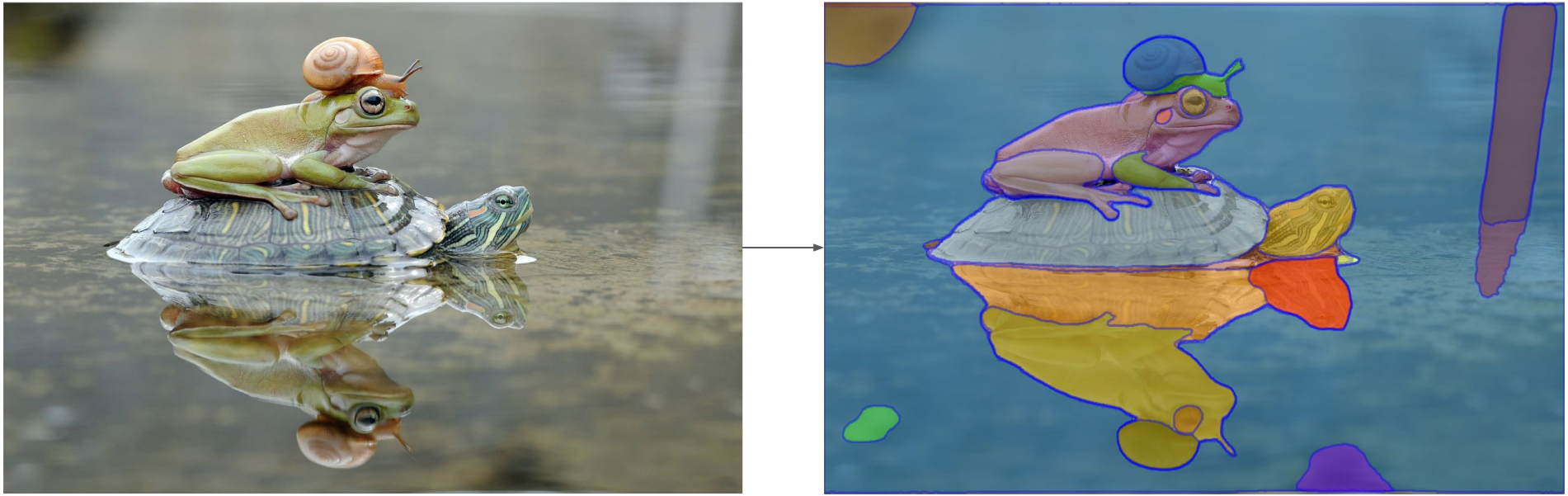
7x7 early to downsample

1x1 and 3x3 for the rest of the net


Beyond Image Classification
0.36
0.34

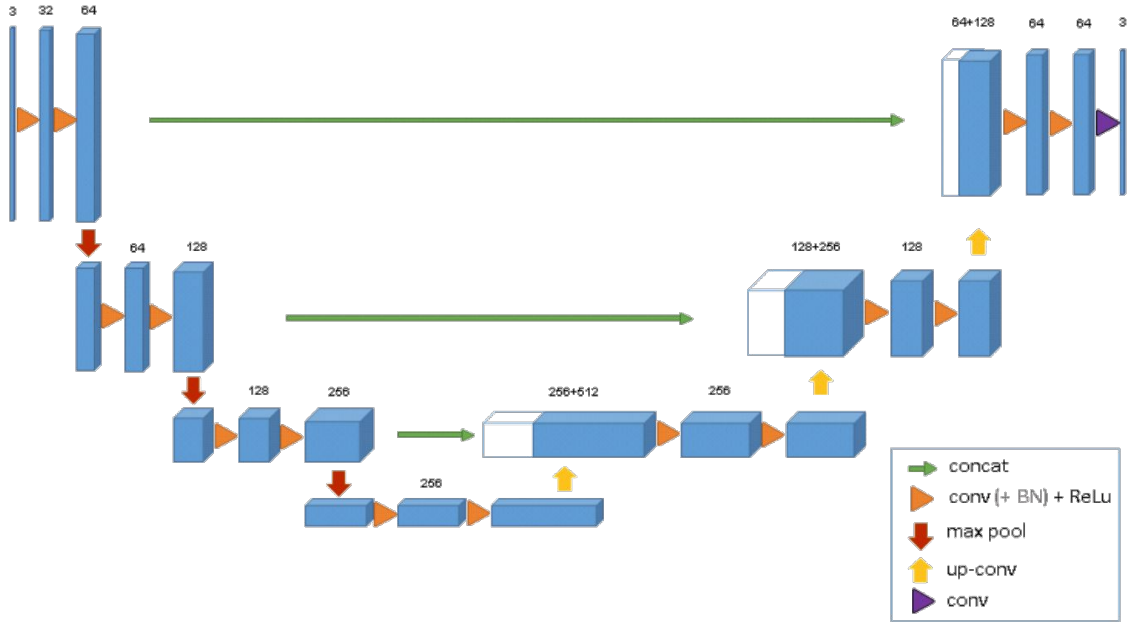
Beyond image classification

Segmentation: identify locations of objects



https://segment-anything.com/demo

Beyond image classification



Reduce spatial dimension, increase channels; then, go back

Beyond image classification

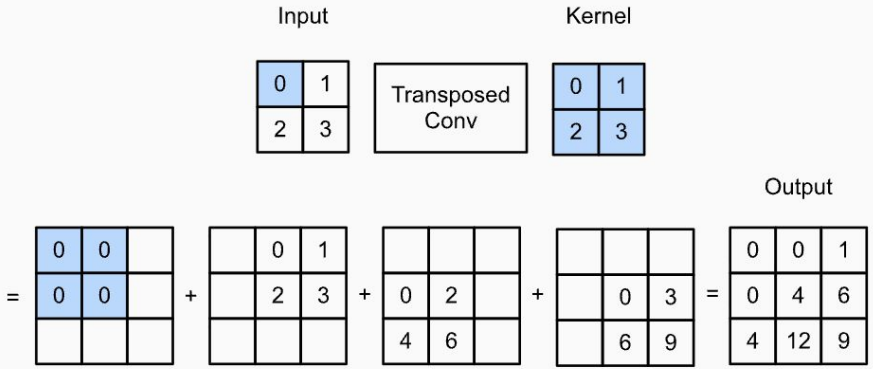
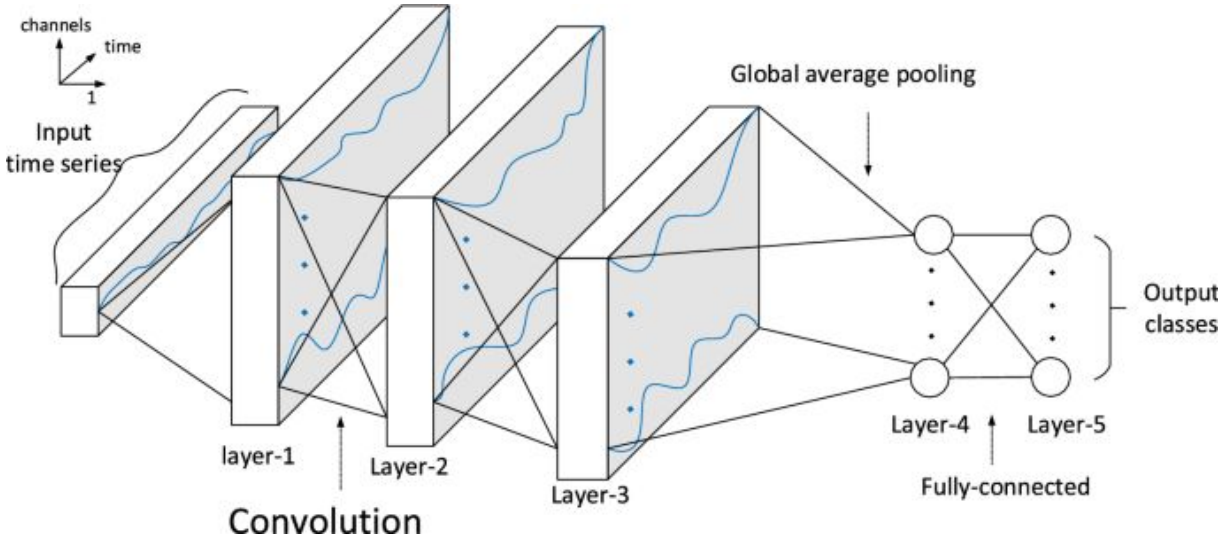


Fig. 14.10.1 Transposed convolution with a 2×2 kernel. The shaded portions are a portion of an intermediate tensor as well as the input and kernel tensor elements used for the computation.

Transposed convolution can be used to increase the spatial dimension of the input.

https://d2l.ai/chapter_computer-vision/transposed-conv.html

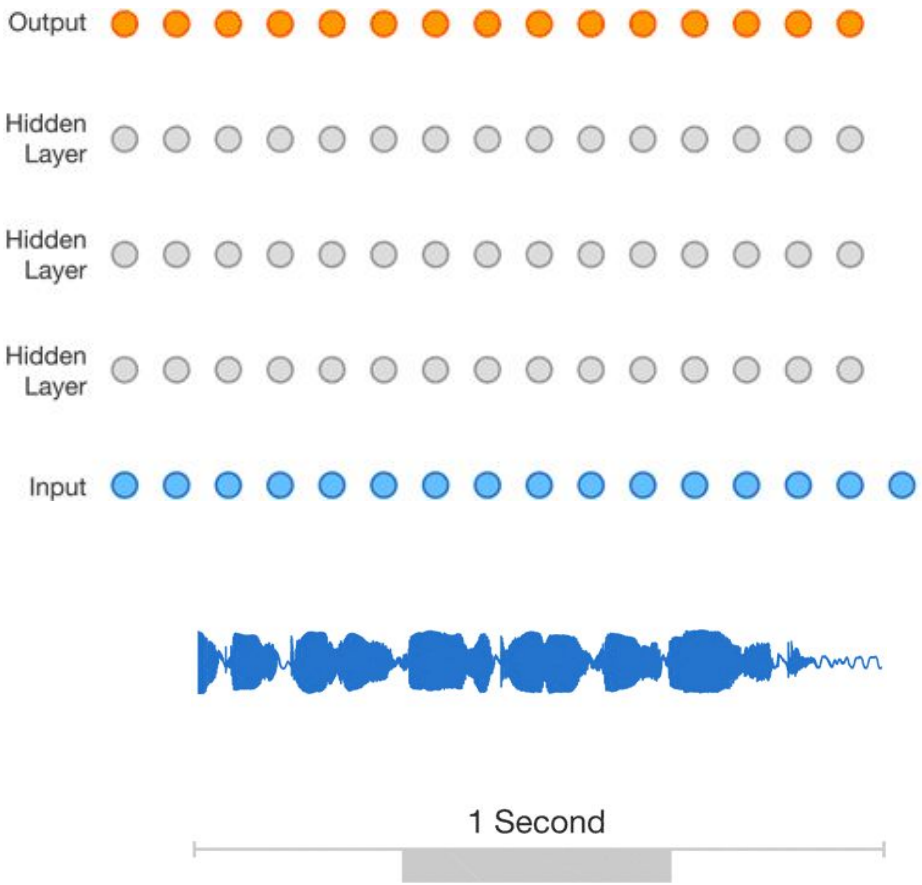
Beyond image classification



1-D CNN can be used for time series processing. There, time is the only spatial dimension.

https://link.springer.com/article/10.1007/s11227-022-04431-5

Beyond image classification



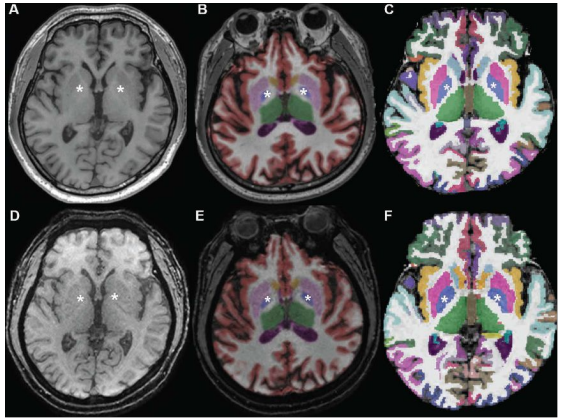
DeepMind's wavenet is a fully-convolutional network for audio generation

https://deepmind.google/discover/blog/wavenet-a-generative-model-for-raw-audio/

Beyond image classification

Regular grid data

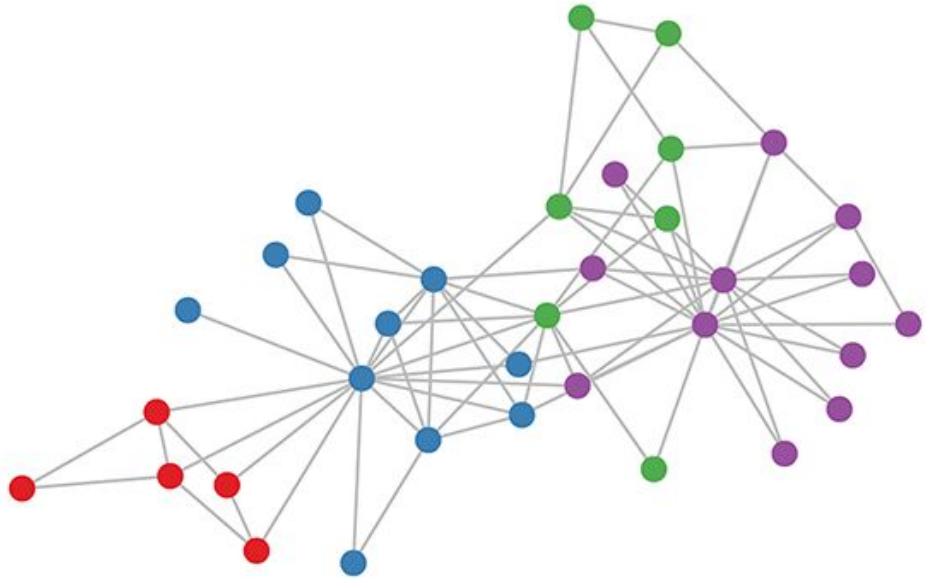
- 1-d
  - Time Series
  - Audio
- 2-d
  - Images
- 3-d
  - MRI
  - Videos



Beyond image classification

Problems on graphs:

- Community detection
- Node classification
- …

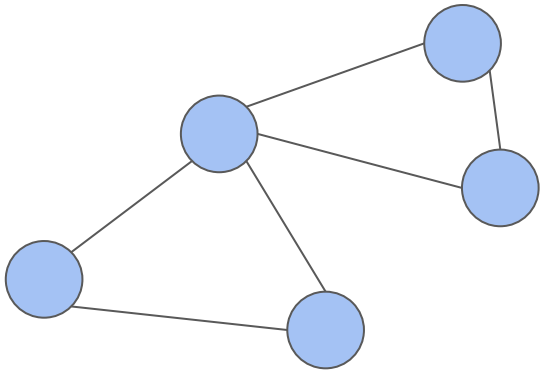


https://tkipf.github.io/graph-convolutional-networks/

Beyond image classification

Graph *G* = *(V, E)*:

- *N* Nodes *V*
- *M* Edges *E*, pairs of nodes

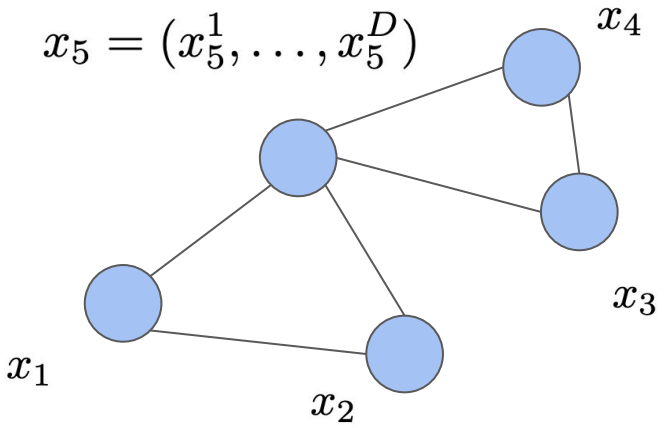


https://tkipf.github.io/graph-convolutional-networks/

Beyond image classification

Graph *G* = *(V, E)*:

- *N* Nodes *V*
- *M* Edges *E*, pairs of nodes



For GNNs:

- Each node has *D* features

https://tkipf.github.io/graph-convolutional-networks/

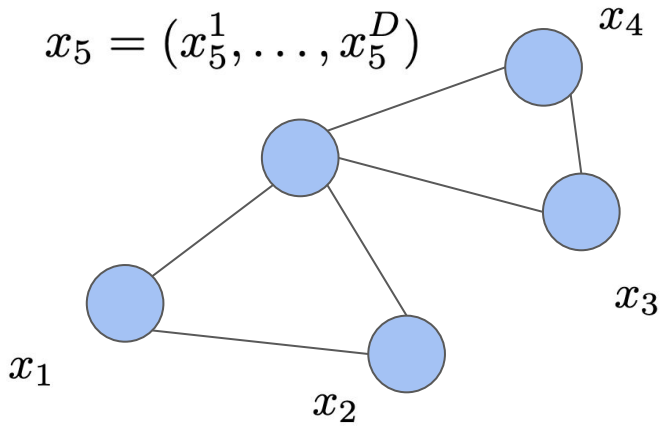
Beyond image classification

Graph *G* = *(V, E)*:

- *N* Nodes *V*
- *M* Edges *E*, pairs of nodes

For GNNs:

- Each node has *D* features
- Adjacency matrix *A*



$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{bmatrix} \in \mathbb{R}^{N \times N}$$

https://tkipf.github.io/graph-convolutional-networks/

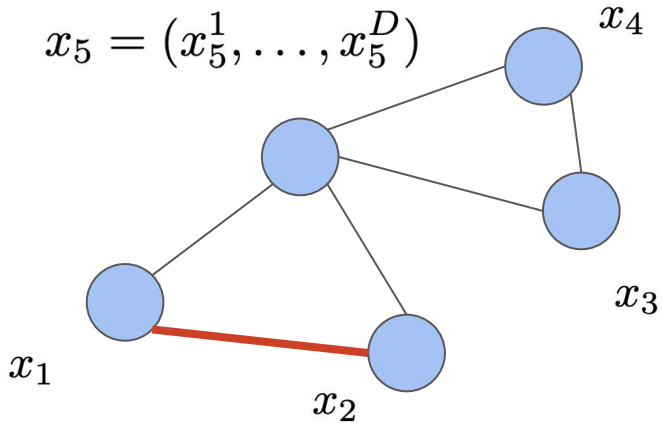
Beyond image classification

Graph *G* = *(V, E)*:

- *N* Nodes *V*
- *M* Edges *E*, pairs of nodes

For GNNs:

- Each node has *D* features
- Adjacency matrix *A*



$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{matrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{matrix}$$

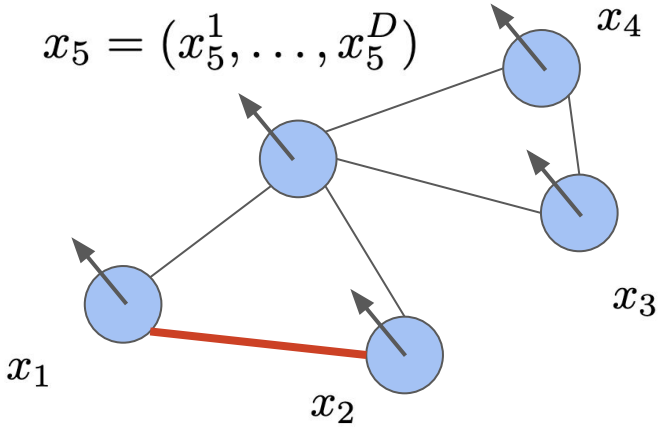
$$\begin{matrix} x_1 & x_2 & x_3 & x_4 & x_5 \end{matrix}$$

https://tkipf.github.io/graph-convolutional-networks/

Beyond image classification

Graph *G* = *(V, E)*:

- *N* Nodes *V*
- *M* Edges *E*, pairs of nodes



For GNNs:

- Each node has *D* features
- Adjacency matrix *A*
- Produce a node-level output

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{matrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{matrix}$$

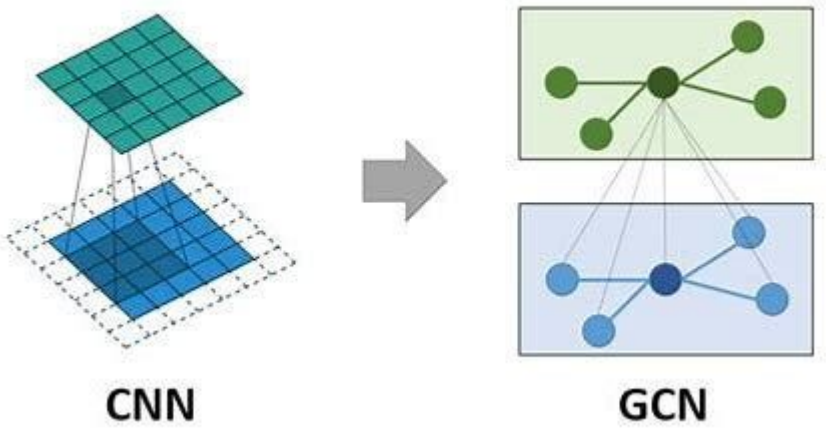
$$\begin{matrix} x_1 & x_2 & x_3 & x_4 & x_5 \end{matrix}$$

https://tkipf.github.io/graph-convolutional-networks/

Beyond image classification

We can generalize convolution to graphs:

- Aggregate features across neighbors
- Apply a linear layer to each node
- Apply non-linearity



$$h_v^{(l+1)} = \sigma\left(W^{(l)} \cdot \text{AGGREGATE}(\{h_u^{(l)} : u \in N(v)\})\right)$$

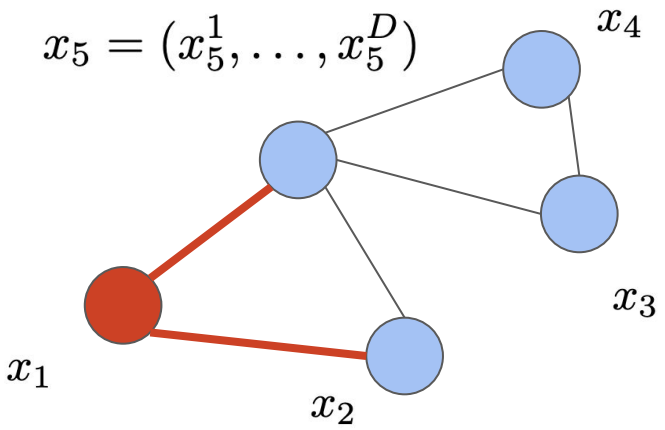
$$W^{(l)} \in \mathbb{R}^{F_{in} \times F_{out}}$$

No spatial arrangement, so the "filter weights" are the same for all neighbors

https://tkipf.github.io/graph-convolutional-networks/

Beyond image classification

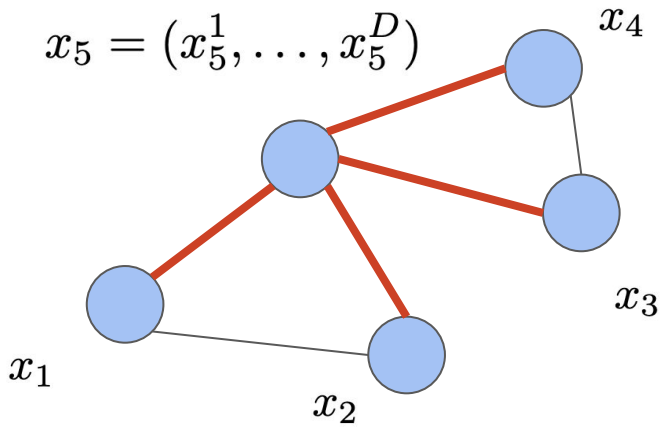
$$h_1^{(1)} = W^{(1)}x_1 + W^{(1)}x_2 + W^{(1)}x_5$$



https://tkipf.github.io/graph-convolutional-networks/

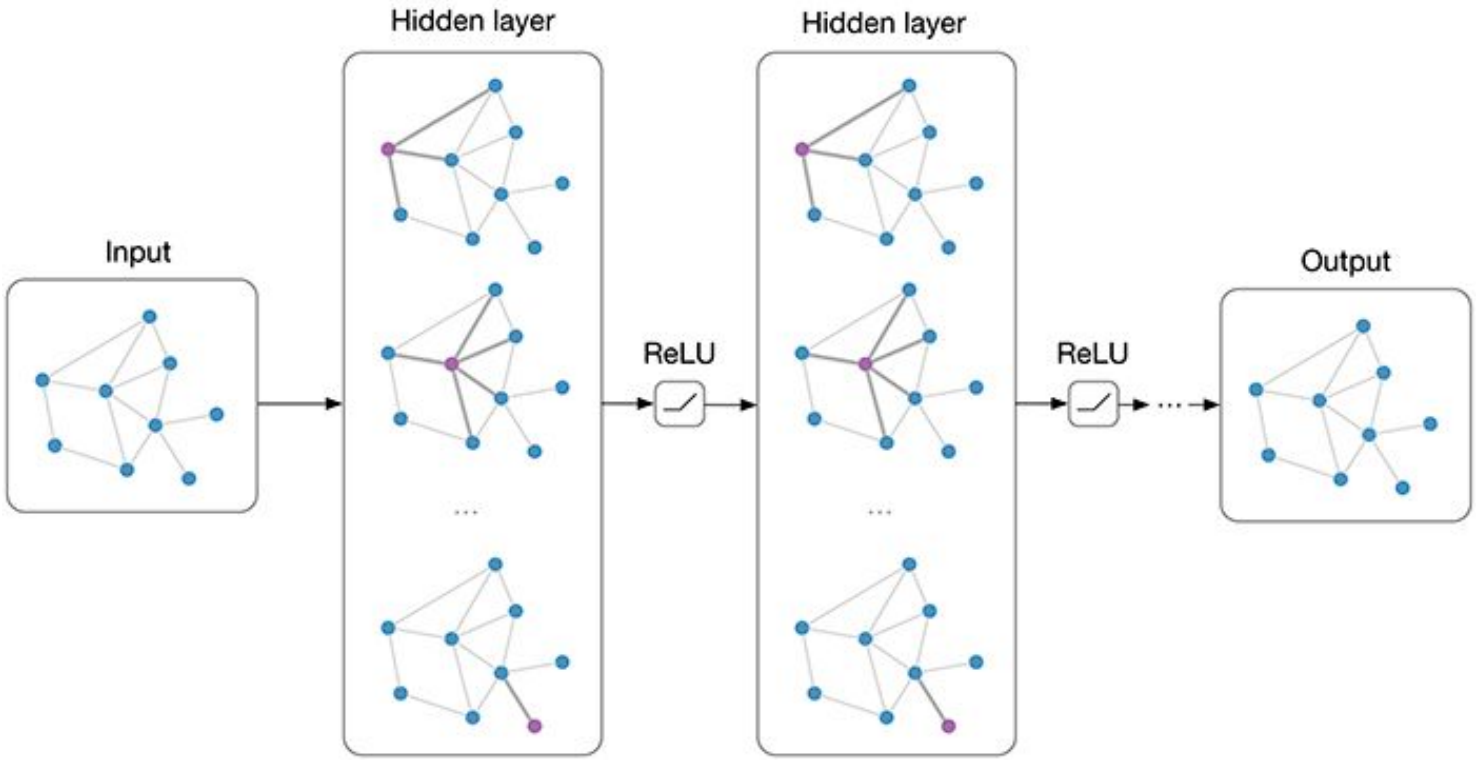
Beyond image classification

$$h_5^{(1)} = W^{(1)}x_1 + W^{(1)}x_2 + W^{(1)}x_3 + W^{(1)}x_4 + W^{(1)}x_5$$



https://tkipf.github.io/graph-convolutional-networks/

Beyond image classification



https://tkipf.github.io/graph-convolutional-networks/

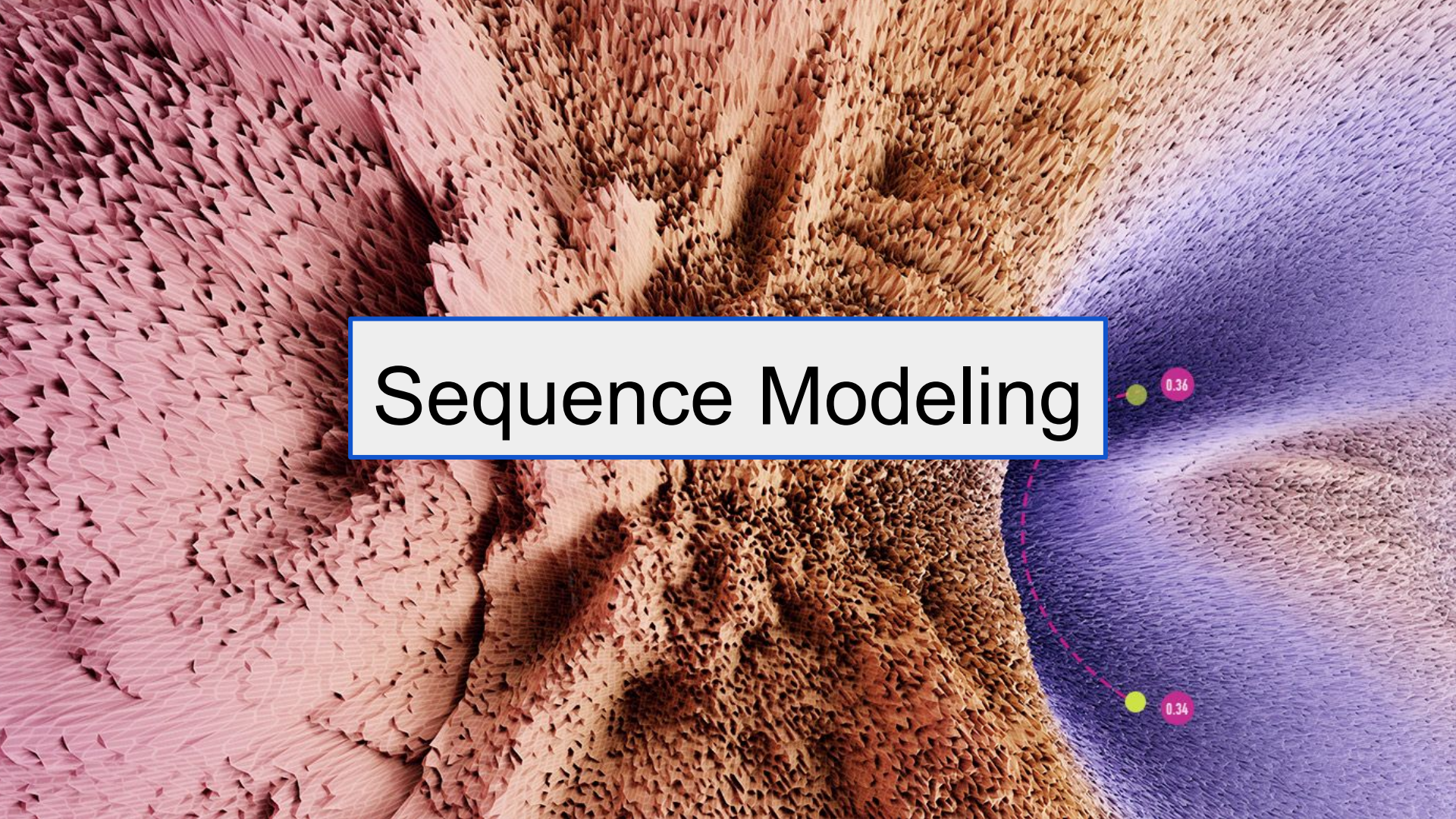
Beyond image classification

Regular grid data

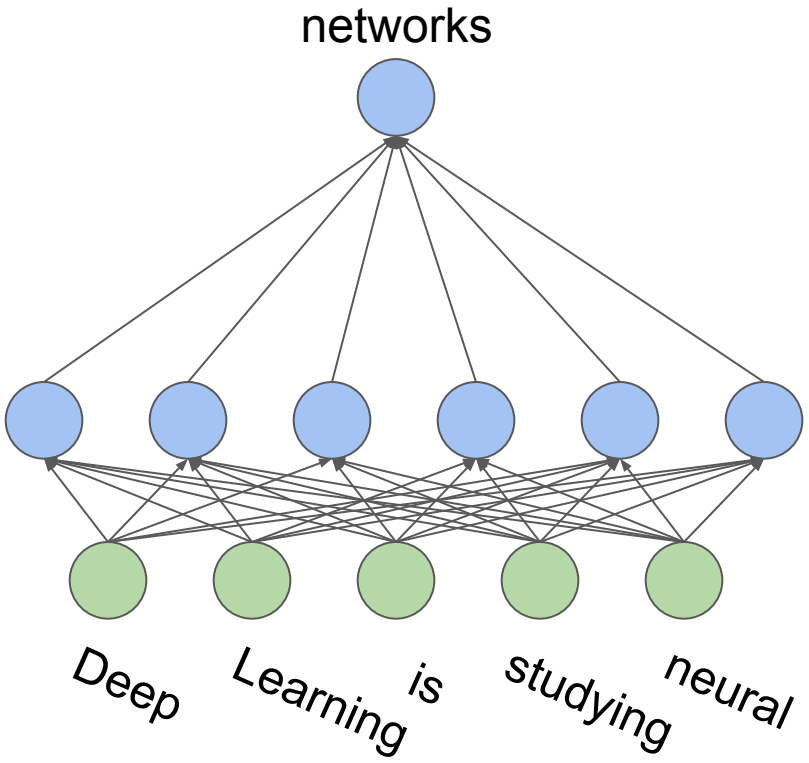
- 1-d
  - Time Series
  - Audio
- 2-d
  - Images
- 3-d
  - MRI
  - Videos

Irregular data:

- Graphs
- Point clouds
- …


Sequence Modeling
0.36
0.34

Sequence Modeling



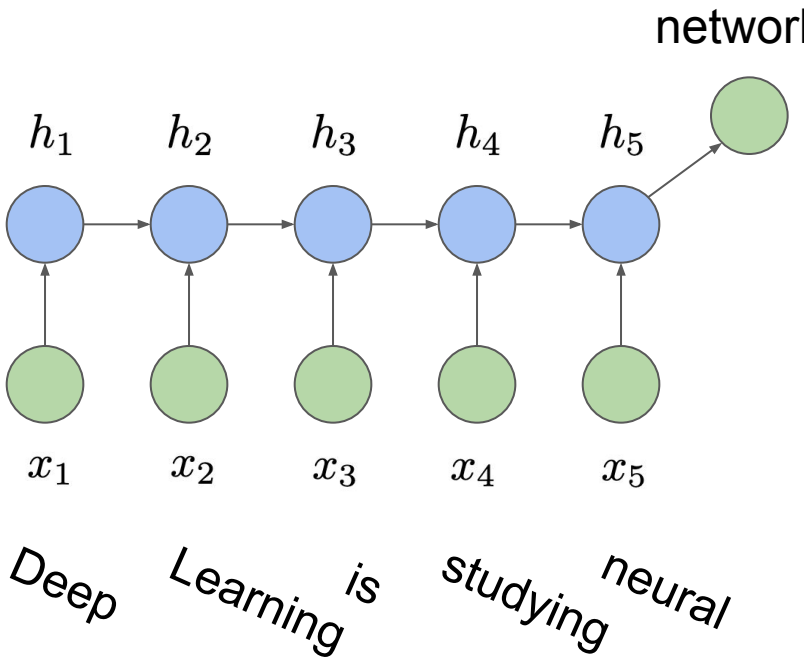
Imagine we want to predict the next word in a sentence.

We could make a fully connected network, but

- What if different length?
- No parameter sharing, relearn the same thing many times (remember motivation for convolutions)

Sequence Modeling

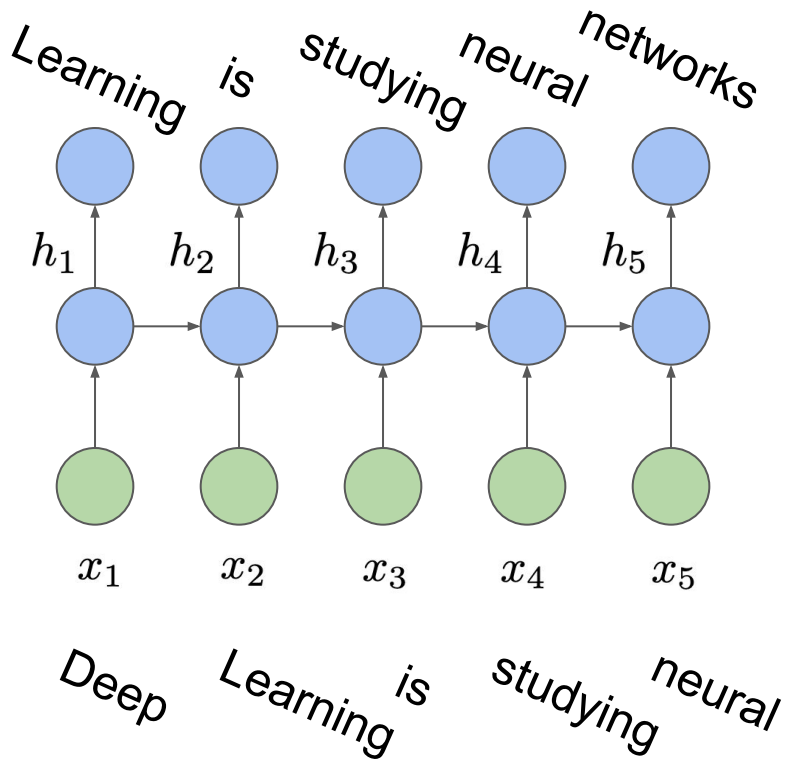
$$h^t = g(x_t, x_{t-1}, \ldots, x_2, x_1)$$
$$= f(x_t, h_{t-1}, \theta)$$



Recurrent Neural Nets (RNNs)

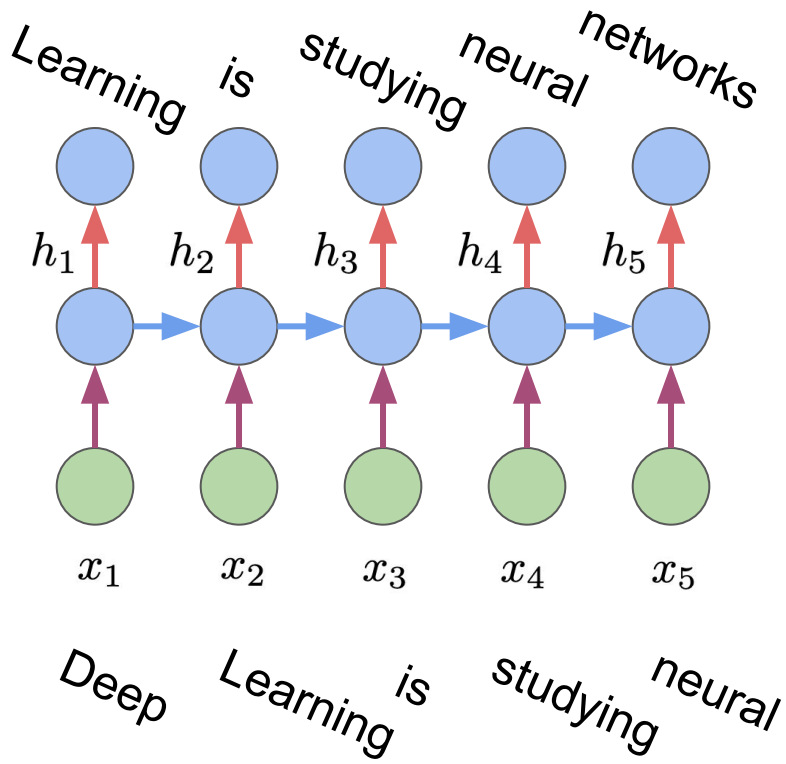
- Introduce time axis
- Hidden layer depends on itself in the previous timesteps
- Information from previous timesteps is only passed via a hidden layer

Sequence Modeling



During training, we can predict the next word at each position.

Sequence Modeling



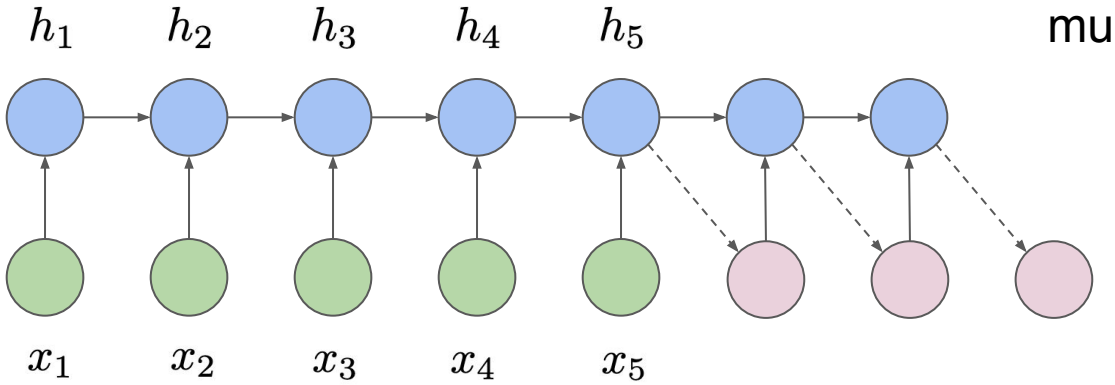
During training, we can predict the next word at each position.

The arrows of the same color correspond to shared layers, i.e. same layer applied to different inputs

Sequence Modeling

$$h^t = g(x_t, x_{t-1}, \ldots, x_2, x_1)$$
$$= f(x_t, h_{t-1}, \theta)$$

networks



At test time, we can feed the predictions back into the model to generate multiple steps ahead.