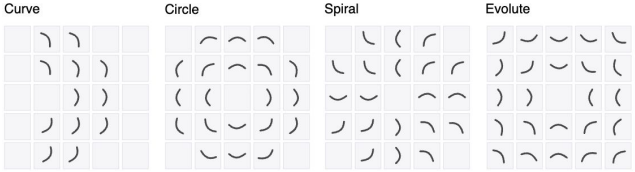
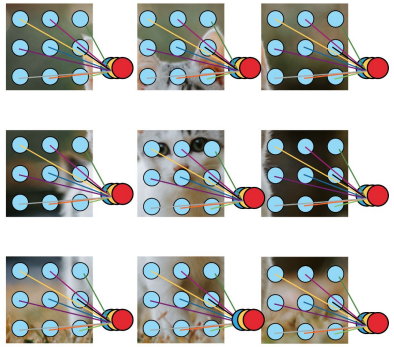
NYU CS-GY 6923 Machine Learning

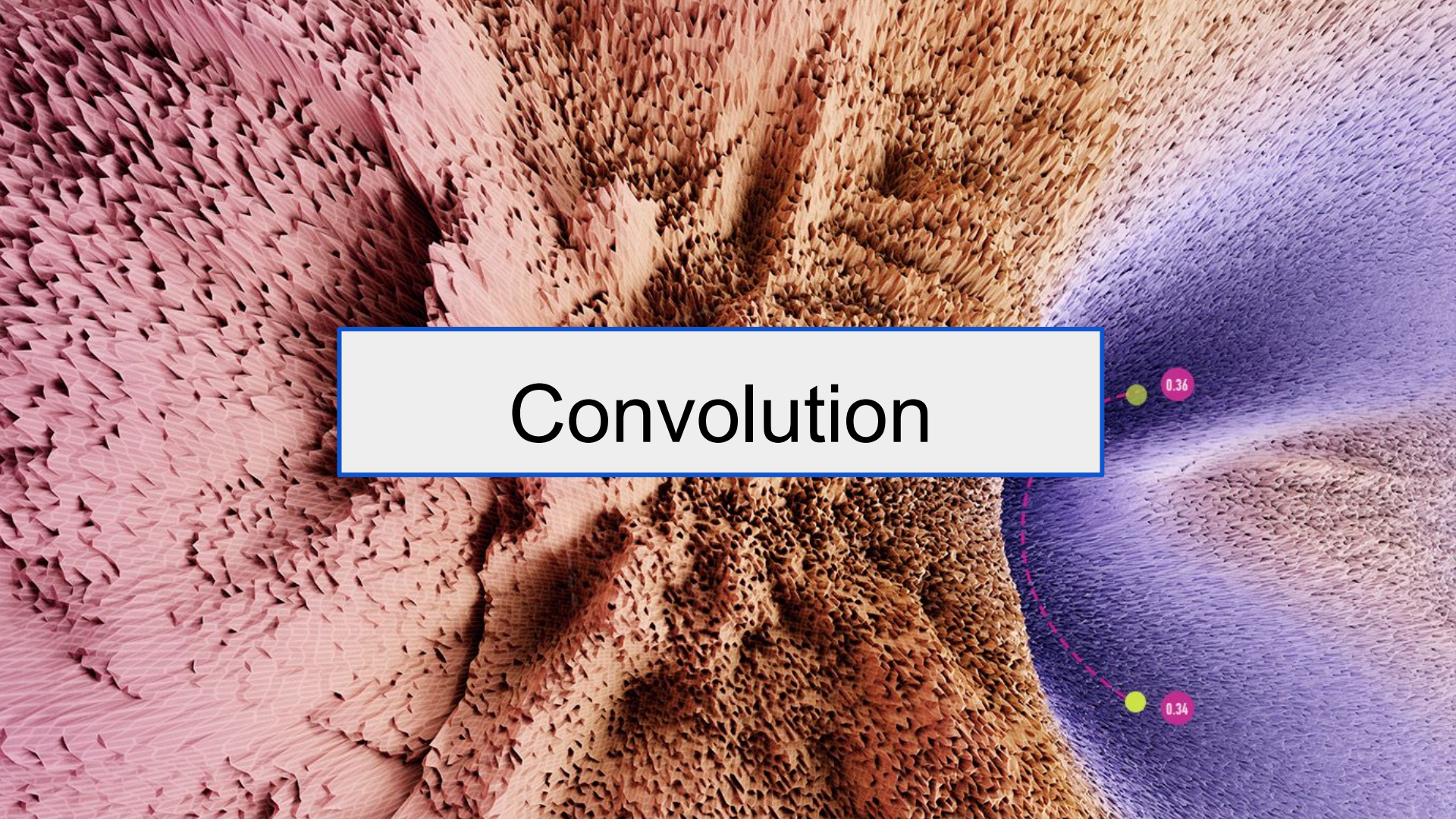
Prof. Pavel Izmailov

Today

- Midterm discussion
- Convolution
- Making a ResNet
- Interpretability
- Convolution beyond images



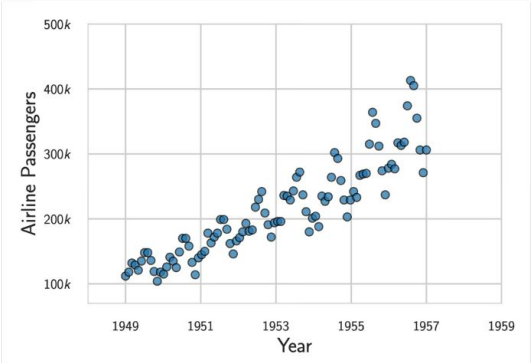



Convolution
0.36
0.34

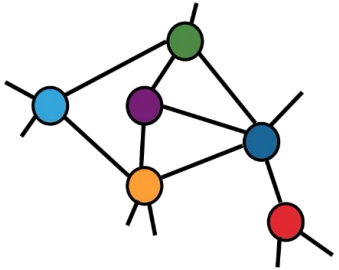
Convolutions

Images

Time-Series

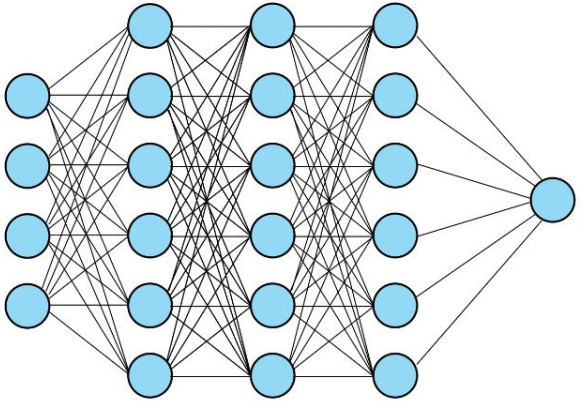


Graphs

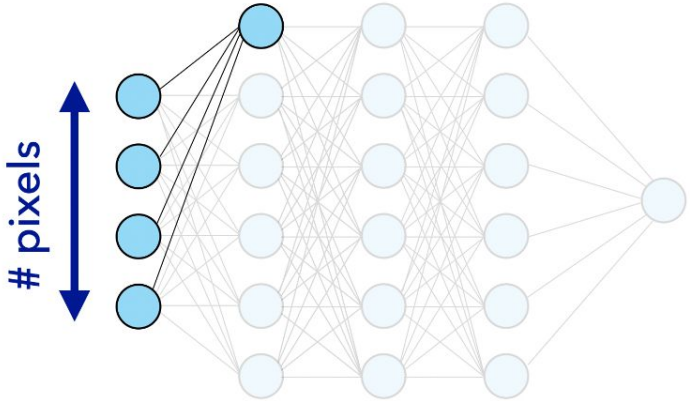


- Convolution is a fundamental idea applicable to many domains
- We will focus on images today

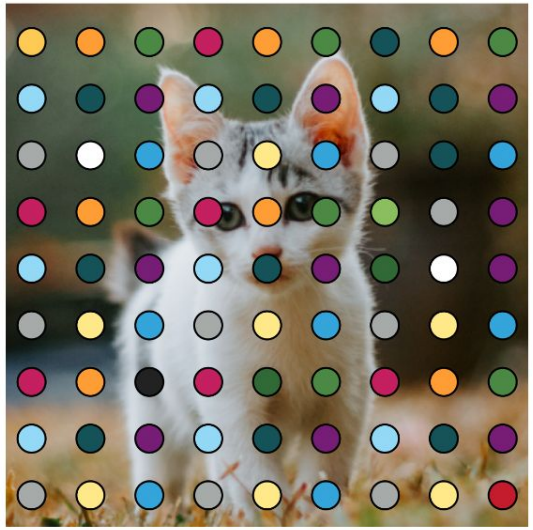
Convolutions

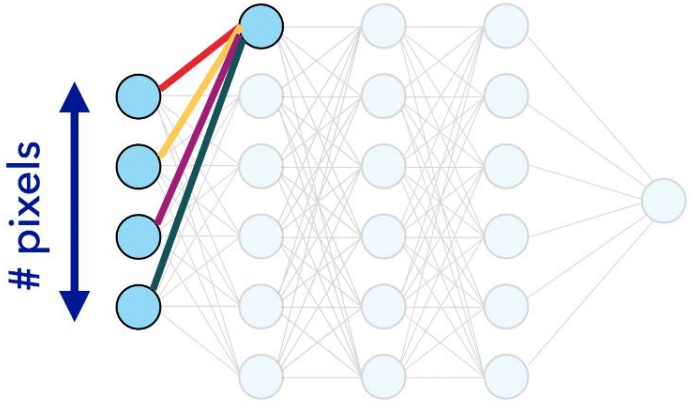


Convolutions

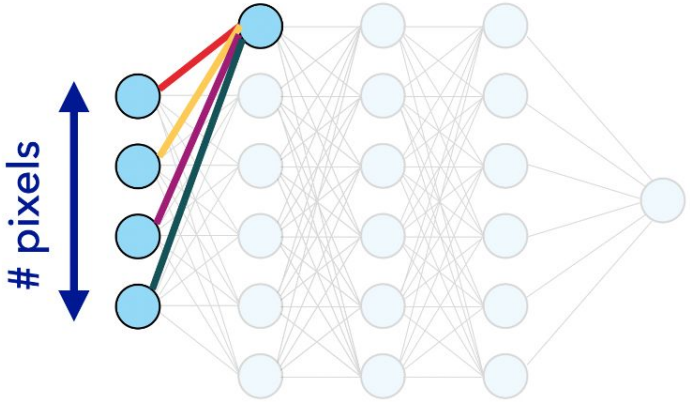


Convolutions

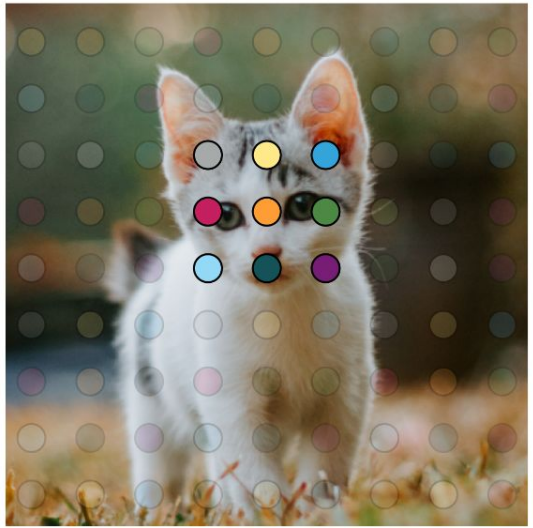




Convolutions

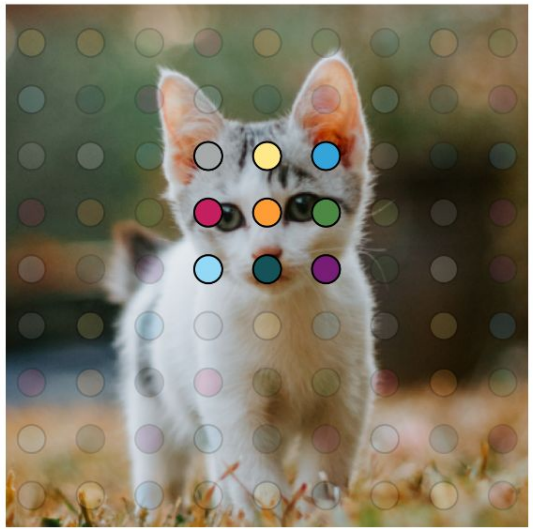


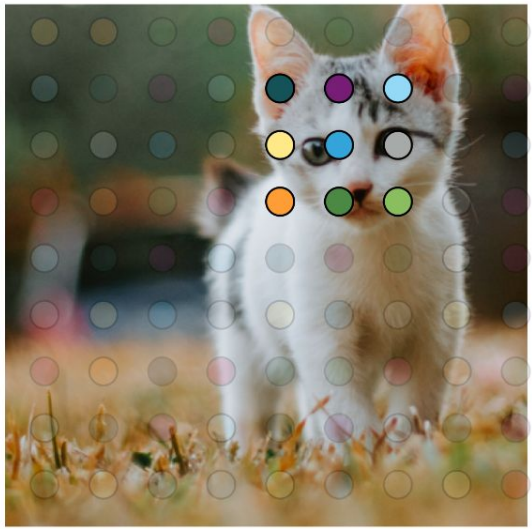
Convolutions



Convolutions

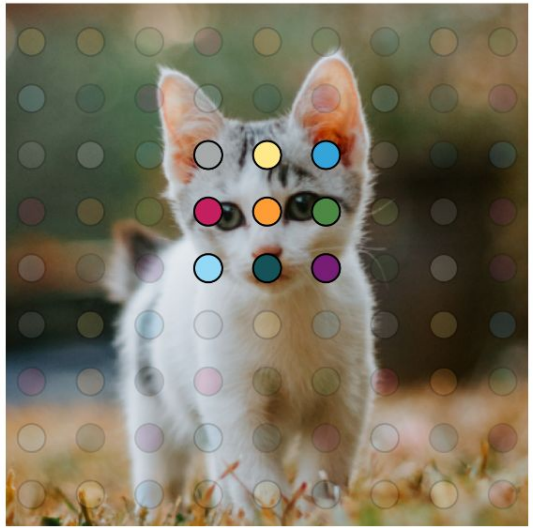
We have to learn the same  face detector in each location independently!





Convolutions

We have to learn the same face detector in each location independently!

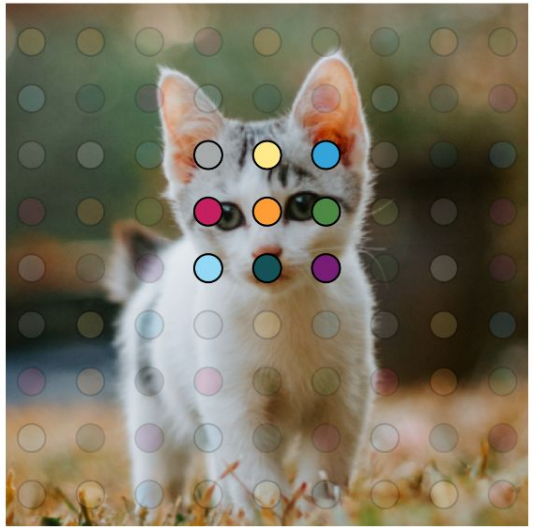


Data inefficient

Parameter inefficient

Convolutions

We have to learn the same face detector in each location independently!



Data inefficient

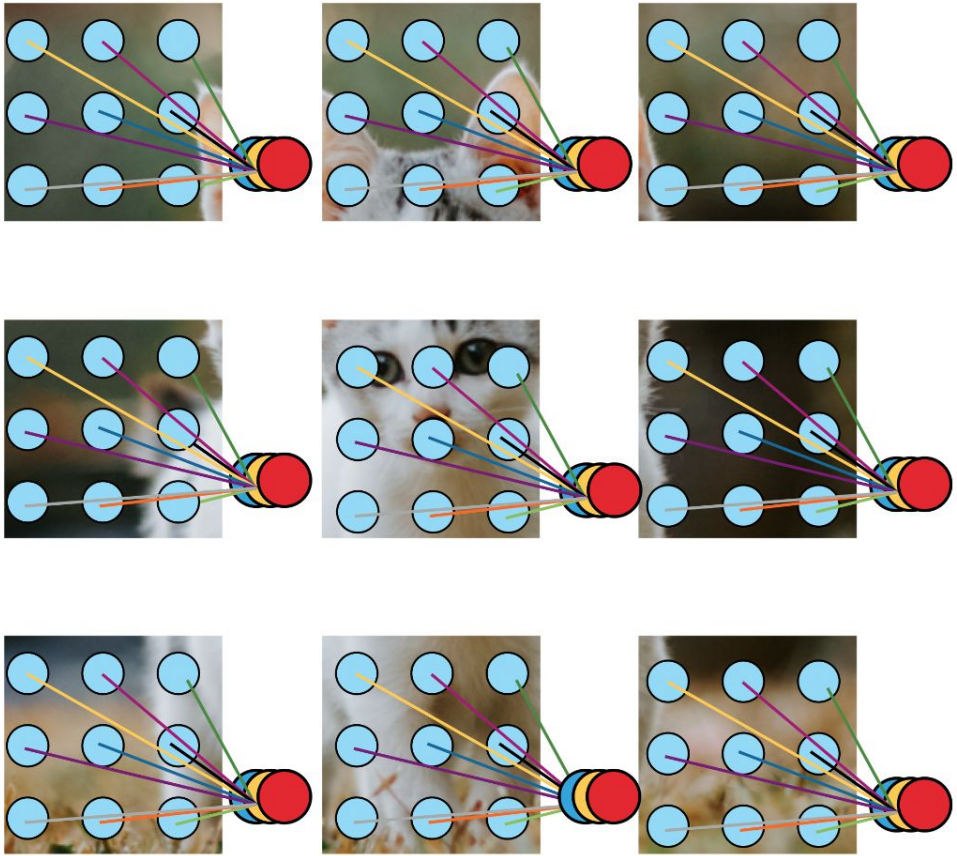
Parameter inefficient

Convolutions

Convolutions

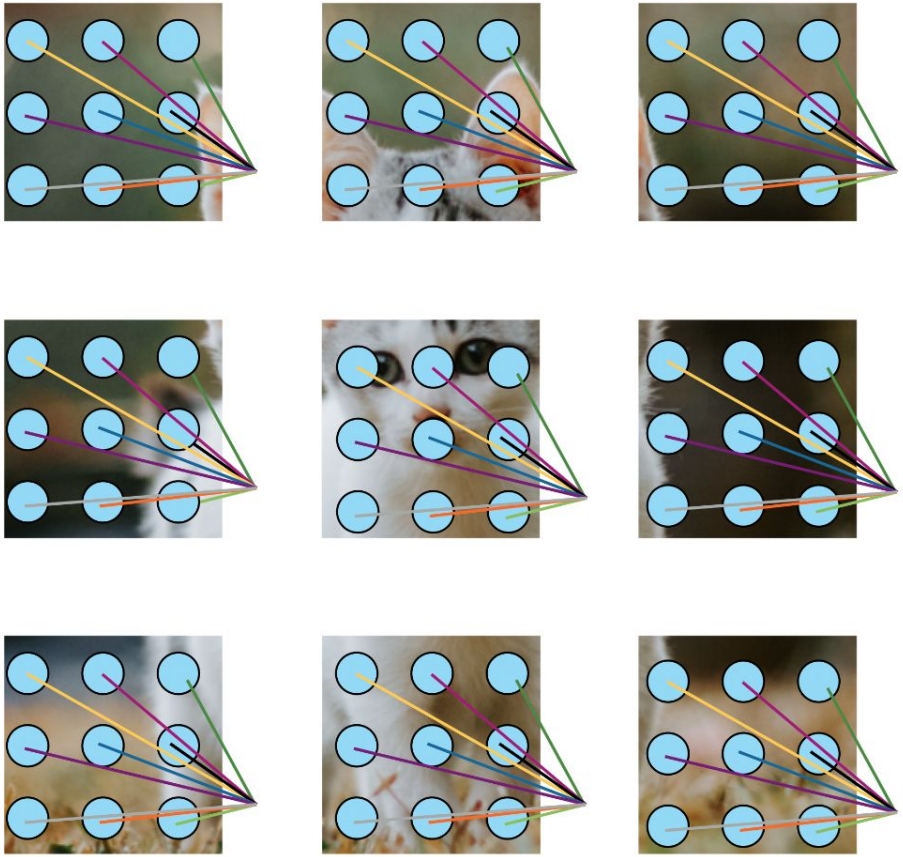
- Split the image in patches

Convolutions

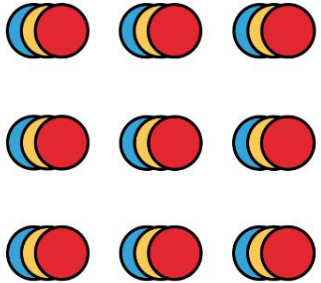


- Split the image in patches
- Apply the same fully-connected layer to each patch

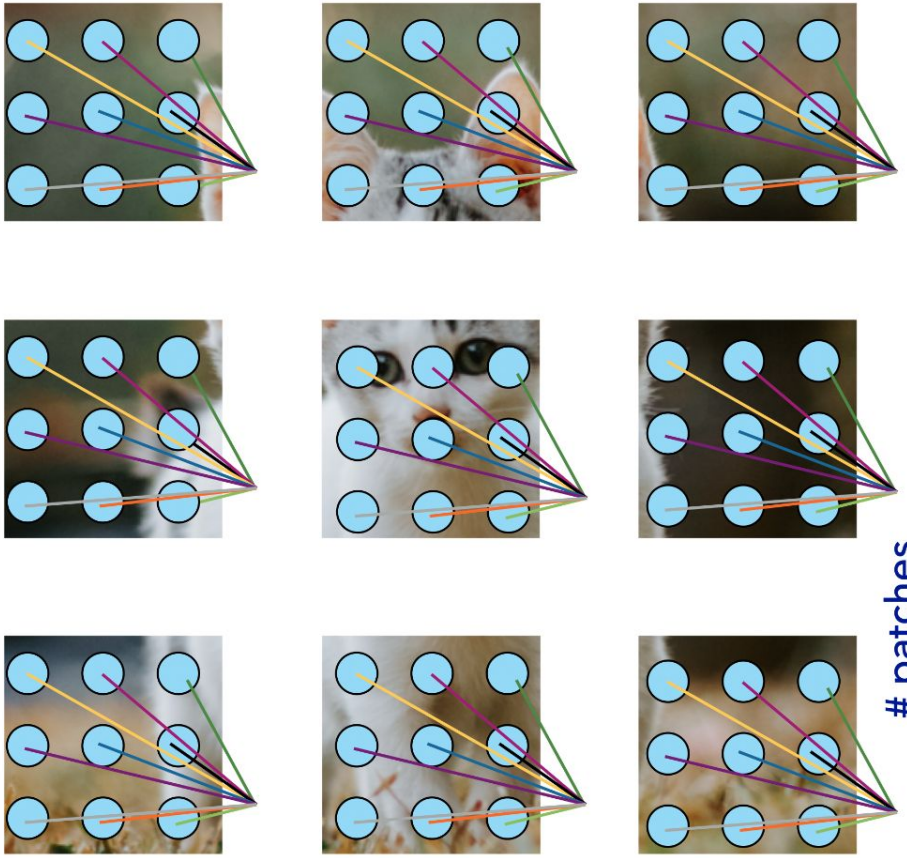
Convolutions



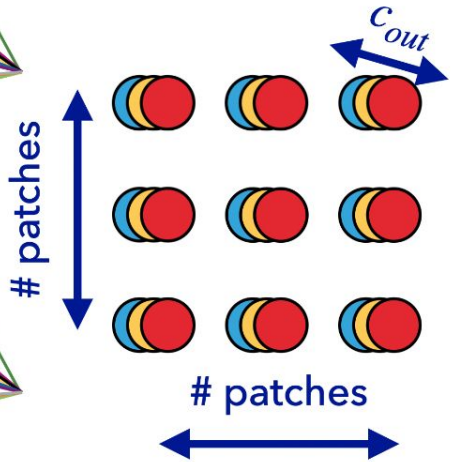
- Split the image in patches
- Apply the same fully-connected layer to each patch
- Collect the outputs



Convolutions



- Split the image in patches
- Apply the same fully-connected layer to each patch
- Collect the outputs

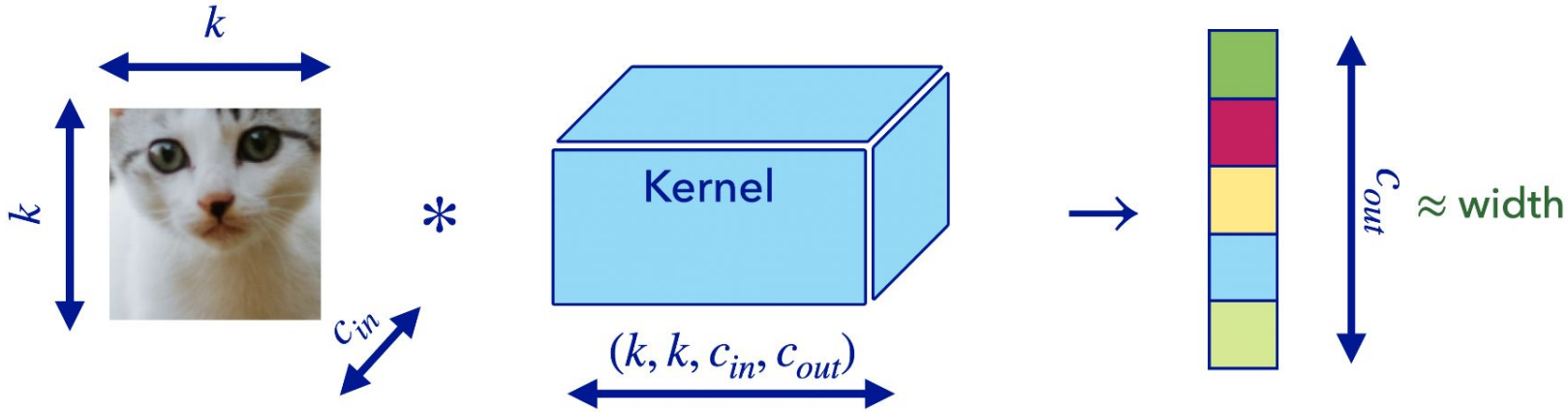


Output is an image

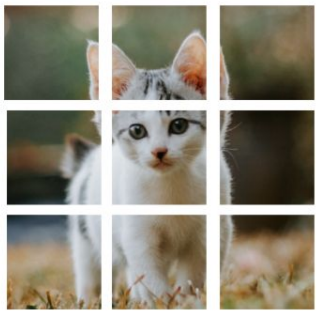
⇓

Can stack!

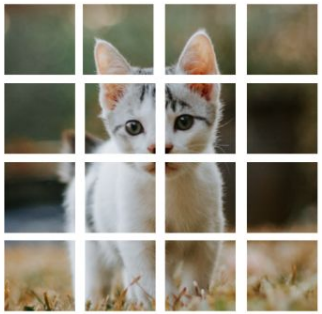
Convolutions



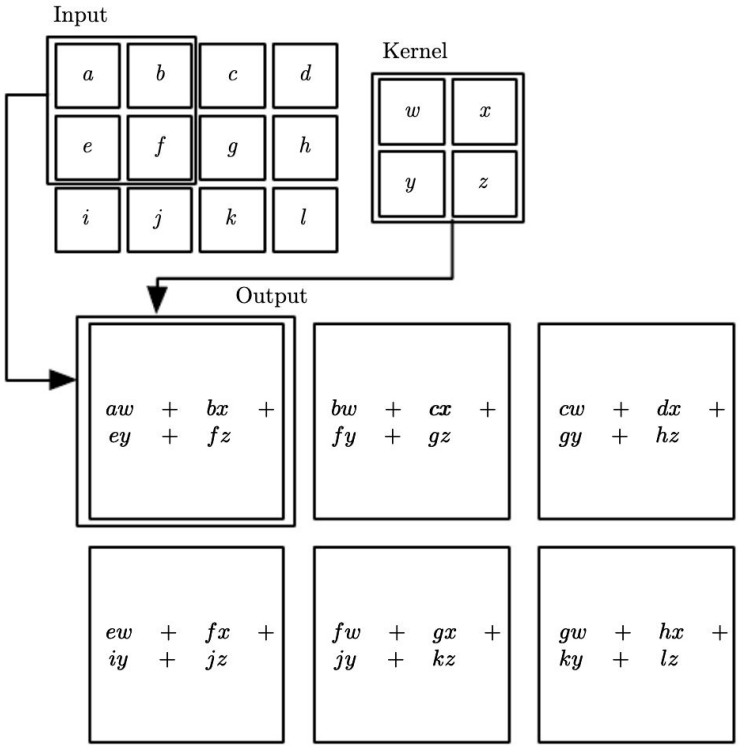
Larger k



Smaller k



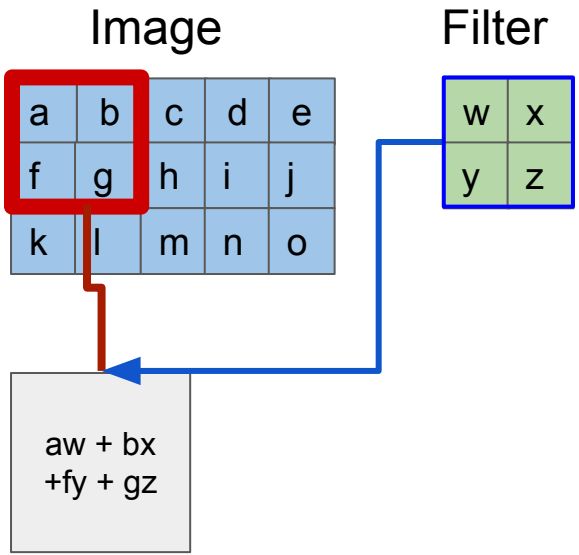
Convolutions



- Convolution with 1 input and 1 output channel
- Apply the kernel to each position in the input
- Get another image as output

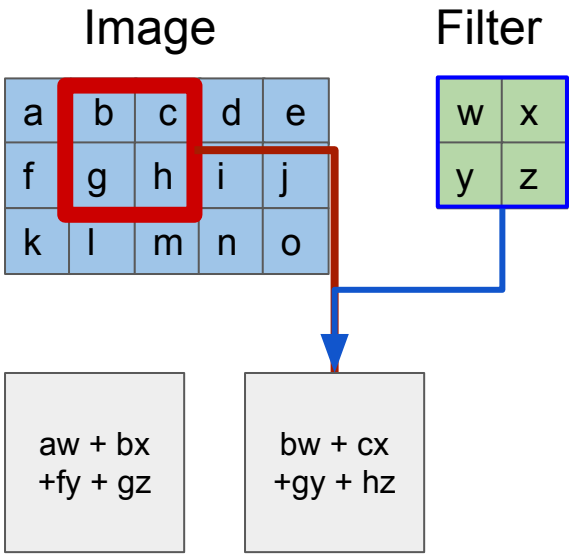
https://www.deeplearningbook.org/, Fig. 9.1

Convolutions



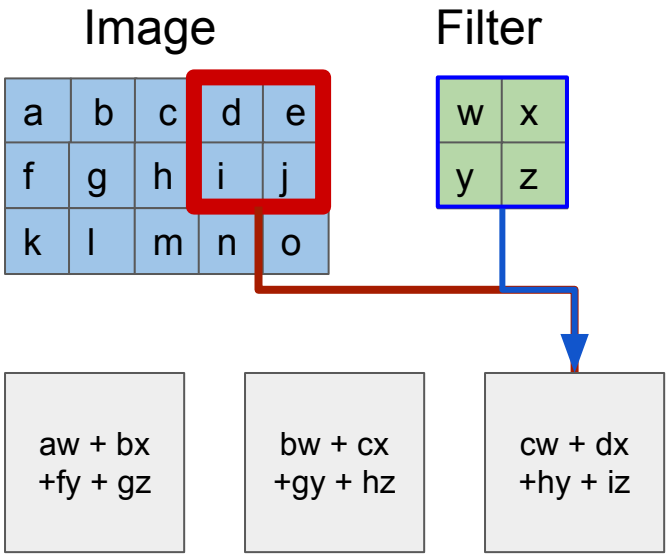
https://www.deeplearningbook.org/, Fig. 9.1

Convolutions



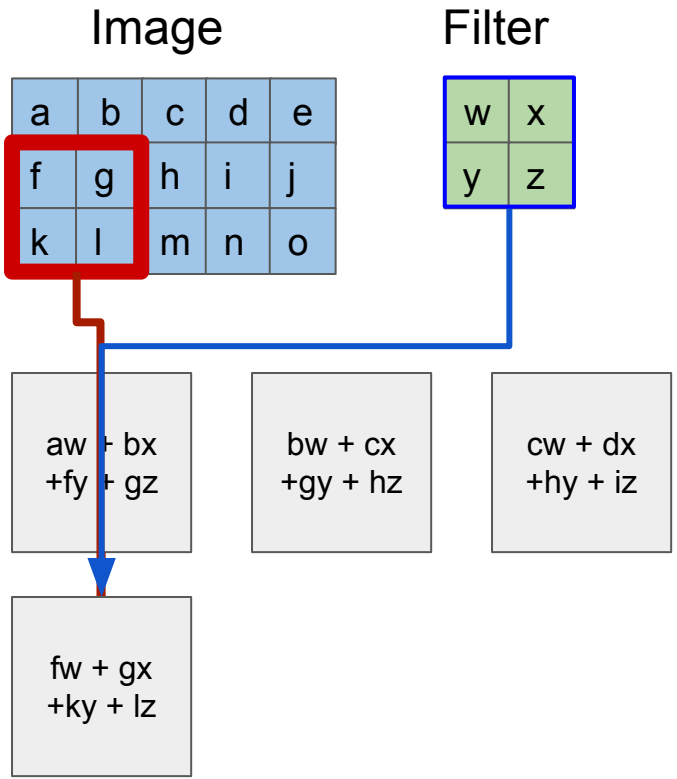
https://www.deeplearningbook.org/, Fig. 9.1

Convolutions



https://www.deeplearningbook.org/, Fig. 9.1

Convolutions



https://www.deeplearningbook.org/, Fig. 9.1

Convolutions

Image

a	b	c	d	e
f	g	h	i	j
k	l	m	n	o

Filter

w	x
y	z

aw + bx +fy + gz	bw + cx +gy + hz	cw + dx +hy + iz	dw + ex +iy + jz
fw + gx +ky + lz	gw + hx +ly + mz	hw + ix +my + nz	iw + jx +ny + oz

https://www.deeplearningbook.org/, Fig. 9.1

Convolutions

Image

a	b	c	d	e
f	g	h	i	j
k	l	m	n	o

Filter

w	x
y	z

aw + bx +fy + gz	bw + cx +gy + hz	cw + dx +hy + iz	dw + ex +iy + jz
fw + gx +ky + lz	gw + hx +ly + mz	hw + ix +my + nz	iw + jx +ny + oz

Image

*	a	b	c	d
#	f	g	h	i
&	k	l	m	n

Filter

w	x
y	z

*w + ax +#y + fz	aw + bx +fy + gz	bw + cx +gy + hz	cw + dx +hy + iz
#w + fx +&y + kz	fw + gx +ky + lz	gw + hx +ly + mz	hw + ix +my + nz

Convolutions

Image

a	b	c	d	e
f	g	h	i	j
k	l	m	n	o

Filter

w	x
y	z

aw + bx +fy + gz	bw + cx +gy + hz	cw + dx +hy + iz	dw + ex +iy + jz
fw + gx +ky + lz	gw + hx +ly + mz	hw + ix +my + nz	iw + jx +ny + oz

Image

*	a	b	c	d
#	f	g	h	i
&	k	l	m	n

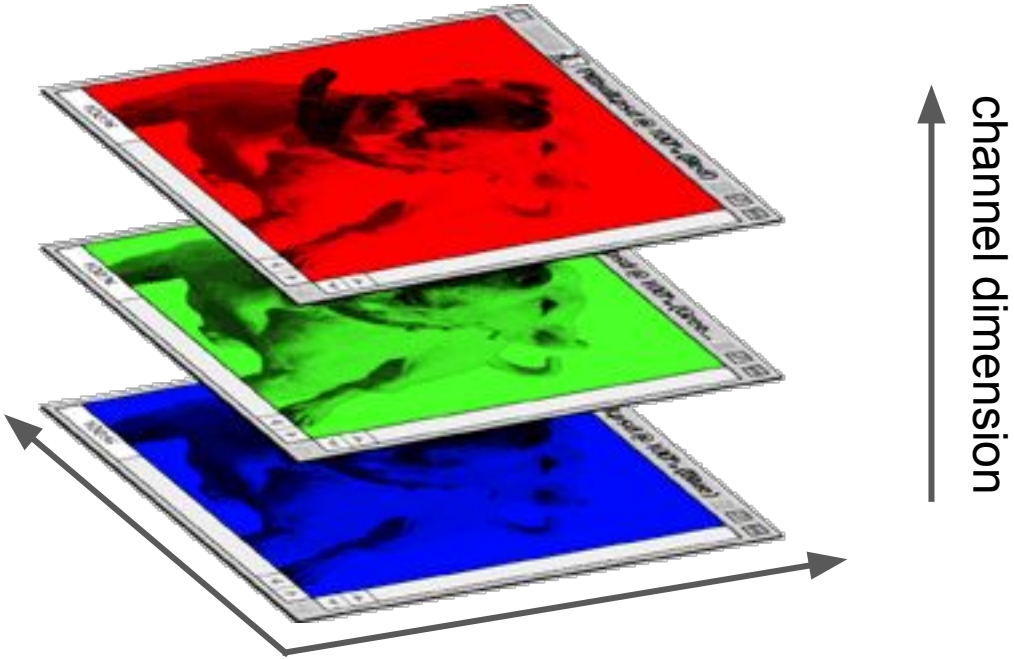
Filter

w	x
y	z

*w + ax +#y + fz	aw + bx +fy + gz	bw + cx +gy + hz	cw + dx +hy + iz
#w + fx +&y + kz	fw + gx +ky + lz	gw + hx +ly + mz	hw + ix +my + nz

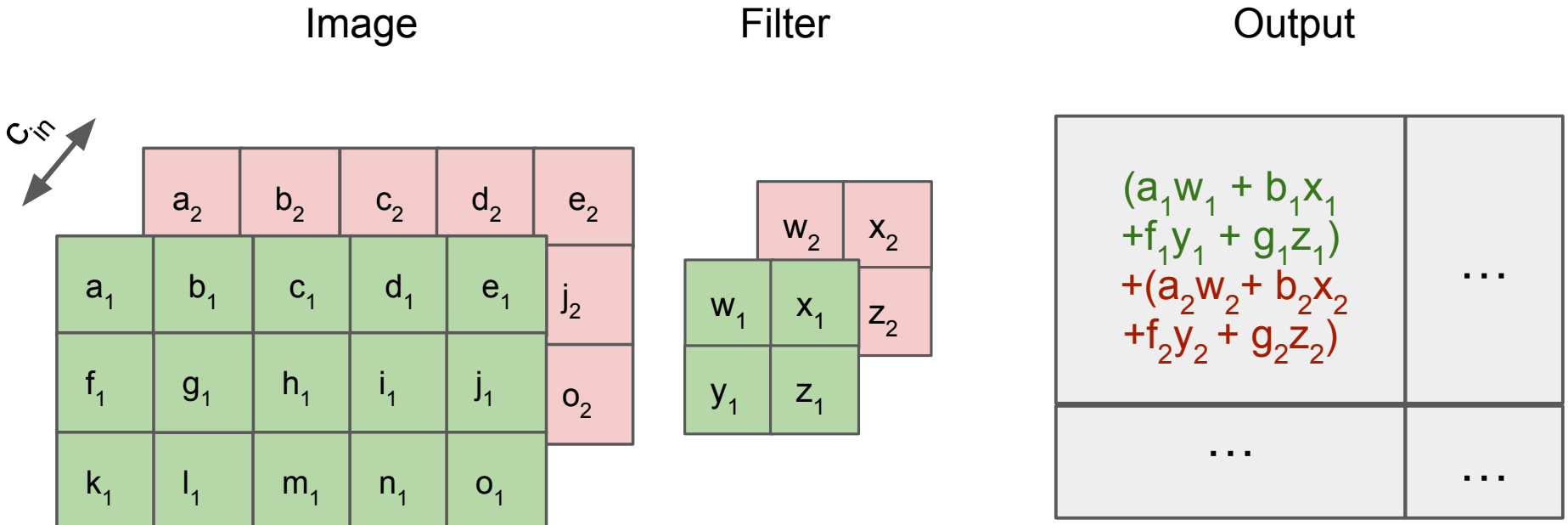
Convolution is locally *equivariant*

Convolutions



We will think about images in terms of 3 coordinates: (vertical, horizontal, channel).
Each channel is a matrix of numbers: pixel intensities.

Convolutions



We can have multiple input channels

Convolutions

Image

a	b	c
f	g	h
k	l	m

Filter 1

w	x
y	z

aw + bx +fy + gz	bw + cx +gy + hz
fw + gx +ky + lz	gw + hx +ly + mz

We can stack multiple filters

Convolutions

Image

a	b	c
f	g	h
k	l	m

Filter 1

w	x
y	z

Filter 2

t	u
v	w

aw + bx +fy + gz	bw + cx +gy + hz
fw + gx +ky + lz	gw + hx +ly + mz

at + bu +fv + gw	bt + cu +gv + hw
ft + gu +kv + lw	gt + hu +lv + mw

We can stack multiple filters

Convolutions

Image

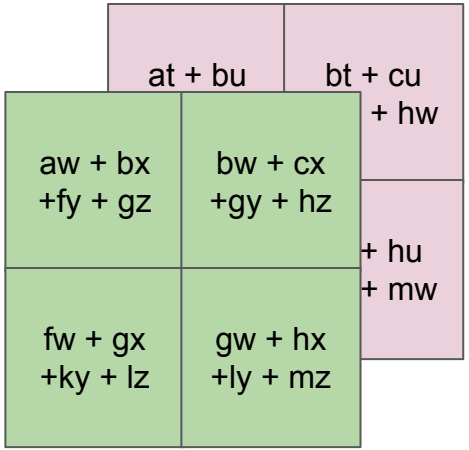
a	b	c
f	g	h
k	l	m

Filter 1

w	x
y	z

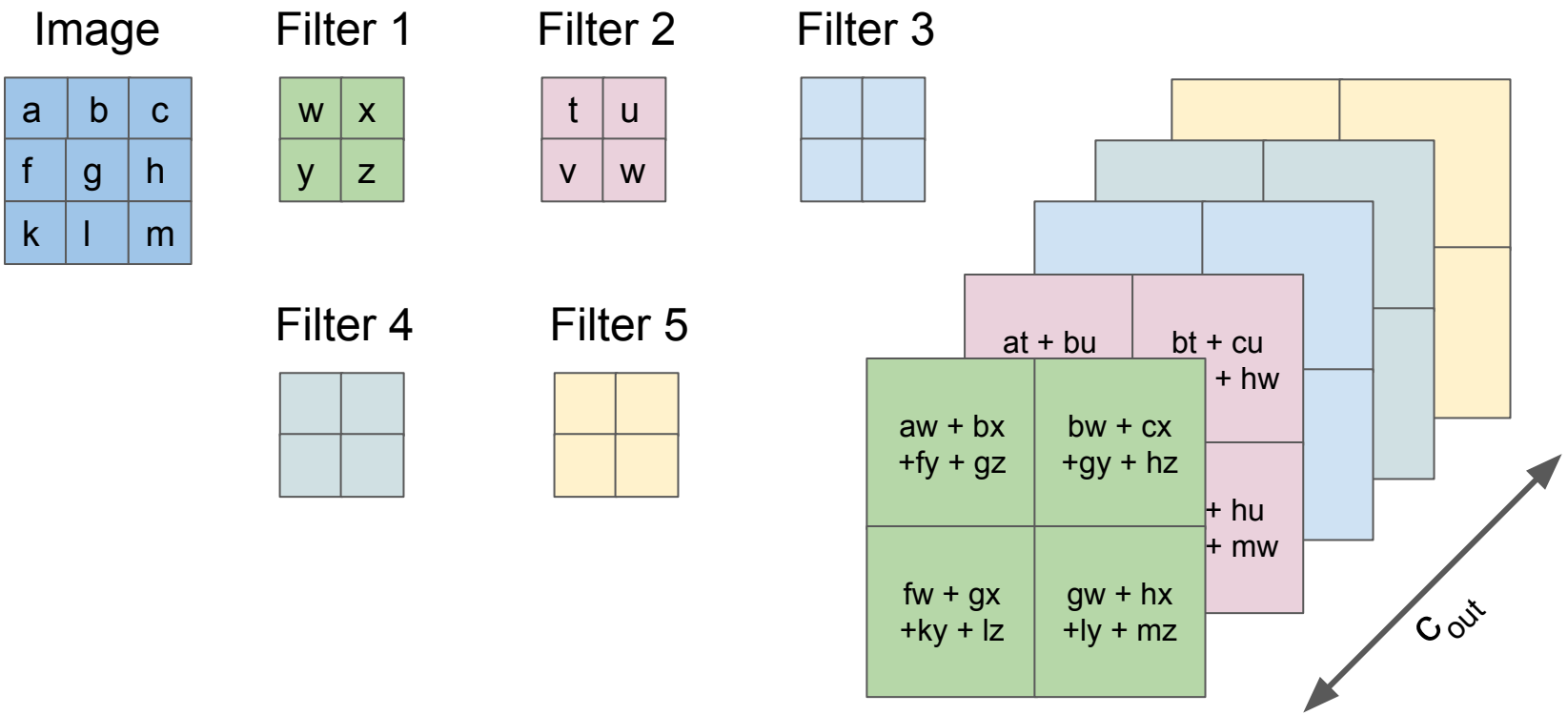
Filter 2

t	u
v	w



We can stack multiple filters

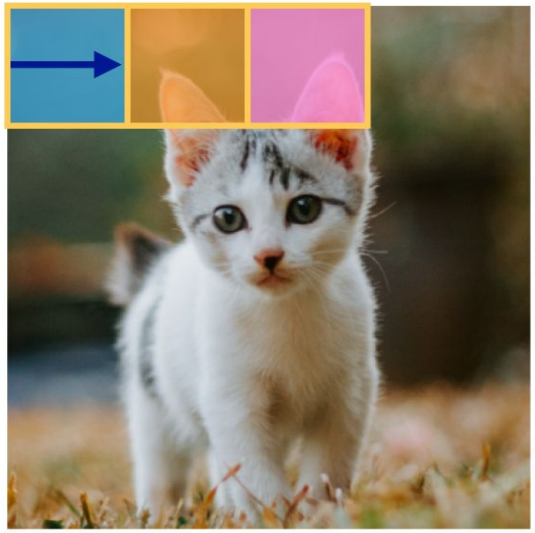
Convolutions



We can stack multiple filters

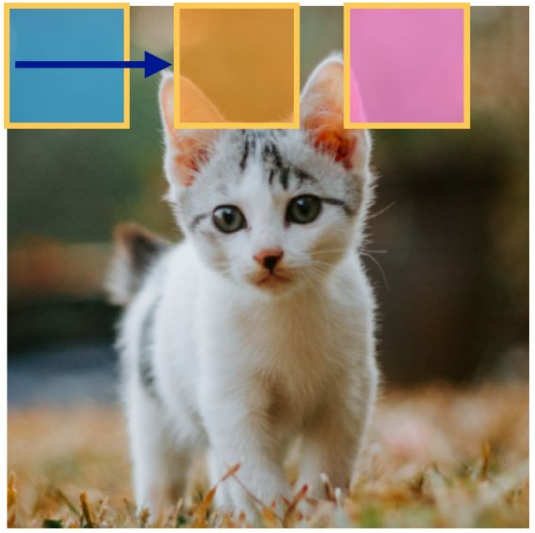
Convolutions: stride

Convolutions: stride



Smaller stride

Larger stride



Convolutions: stride

Fully-connected

```
fc_net = nn.Sequential(
    nn.Flatten(),
    nn.Linear(28**2, 100),
    nn.ReLU(),
    nn.Linear(100, 100),
    nn.ReLU(),
    nn.Linear(100, 100),
    nn.ReLU(),
    nn.Linear(100, 10)
)
```

Params: 100k, Acc: 94.5%

Let's go through part 1 of demo-cnn.ipynb.

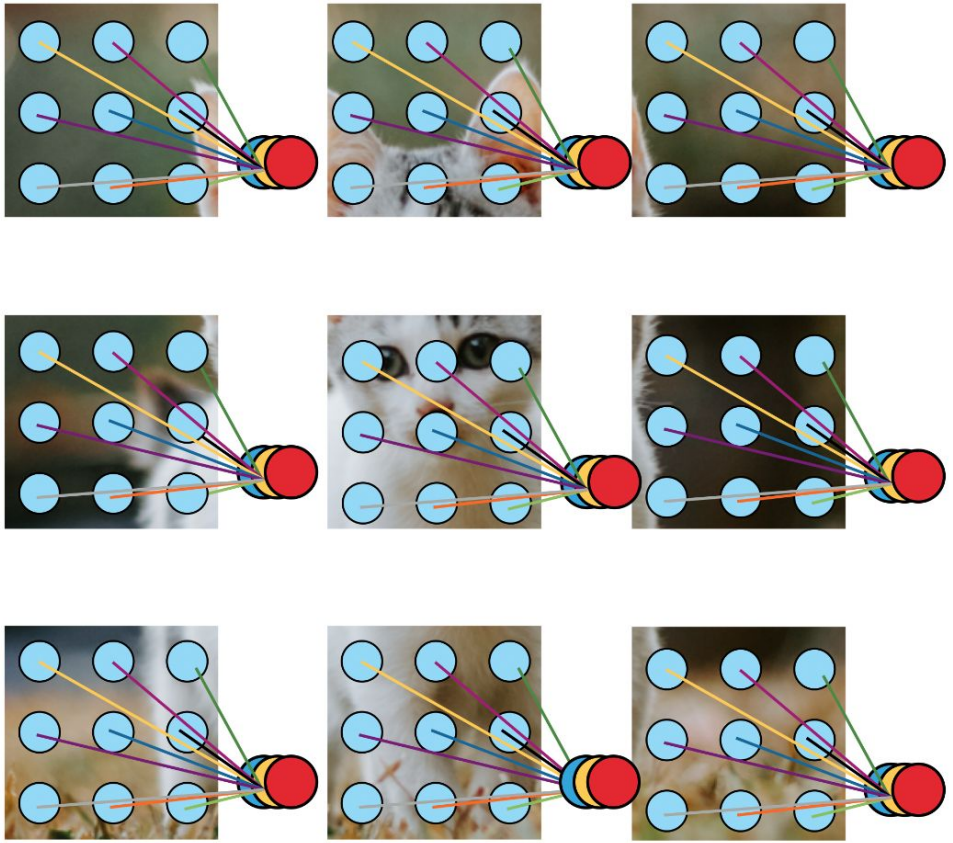
Convolutional

```
conv_net = nn.Sequential(
    nn.Conv2d(in_channels=1,
              out_channels=32,
              kernel_size=5,
              stride=2),
    nn.ReLU(),
    nn.Conv2d(in_channels=32,
              out_channels=32,
              kernel_size=5,
              stride=2),
    nn.ReLU()
    nn.Flatten(),
    nn.Linear(512, 10)
)
```

Params: 32k, Acc: 97.2%

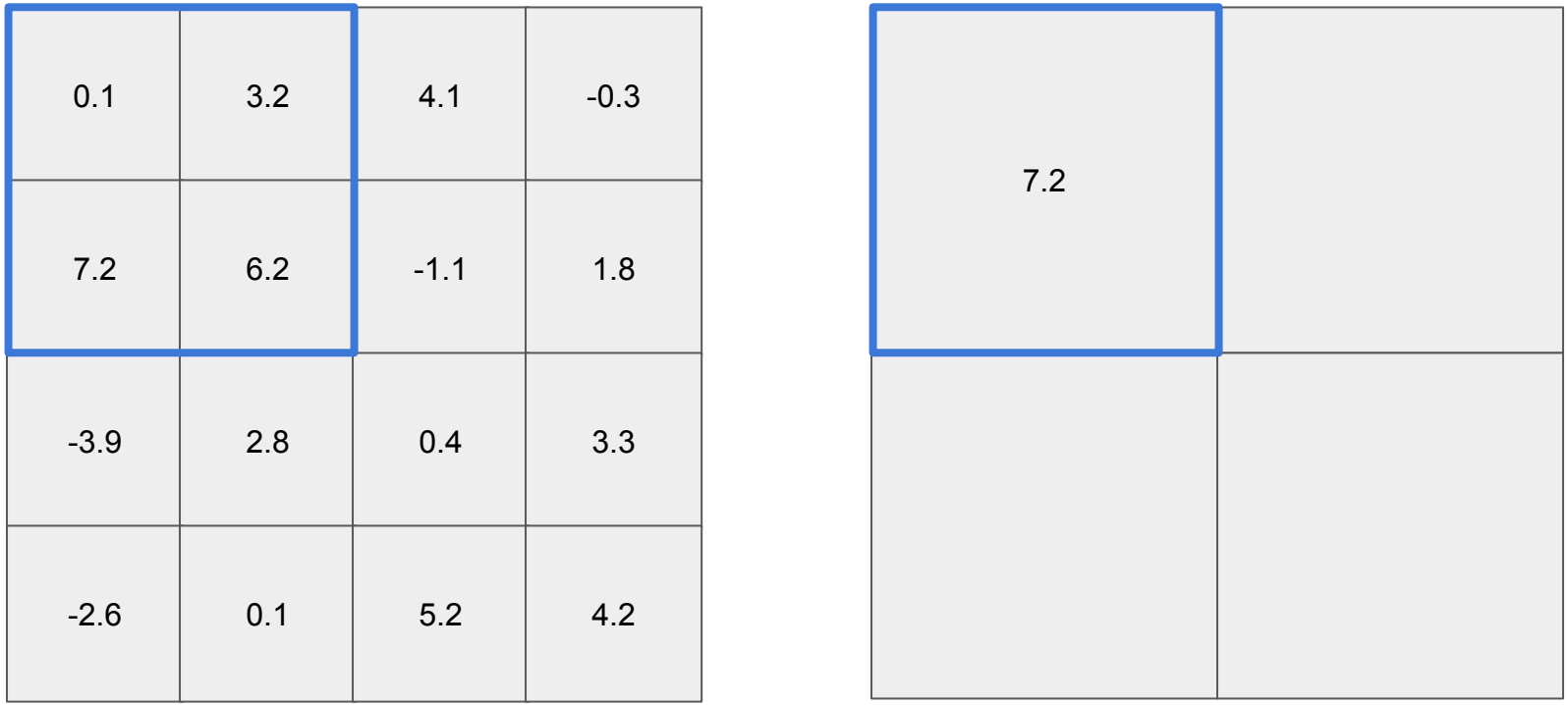
Convolutions: stride

- Shared fully-connected layer applied to patches
- Better parameter efficiency and data efficiency
- Better performance



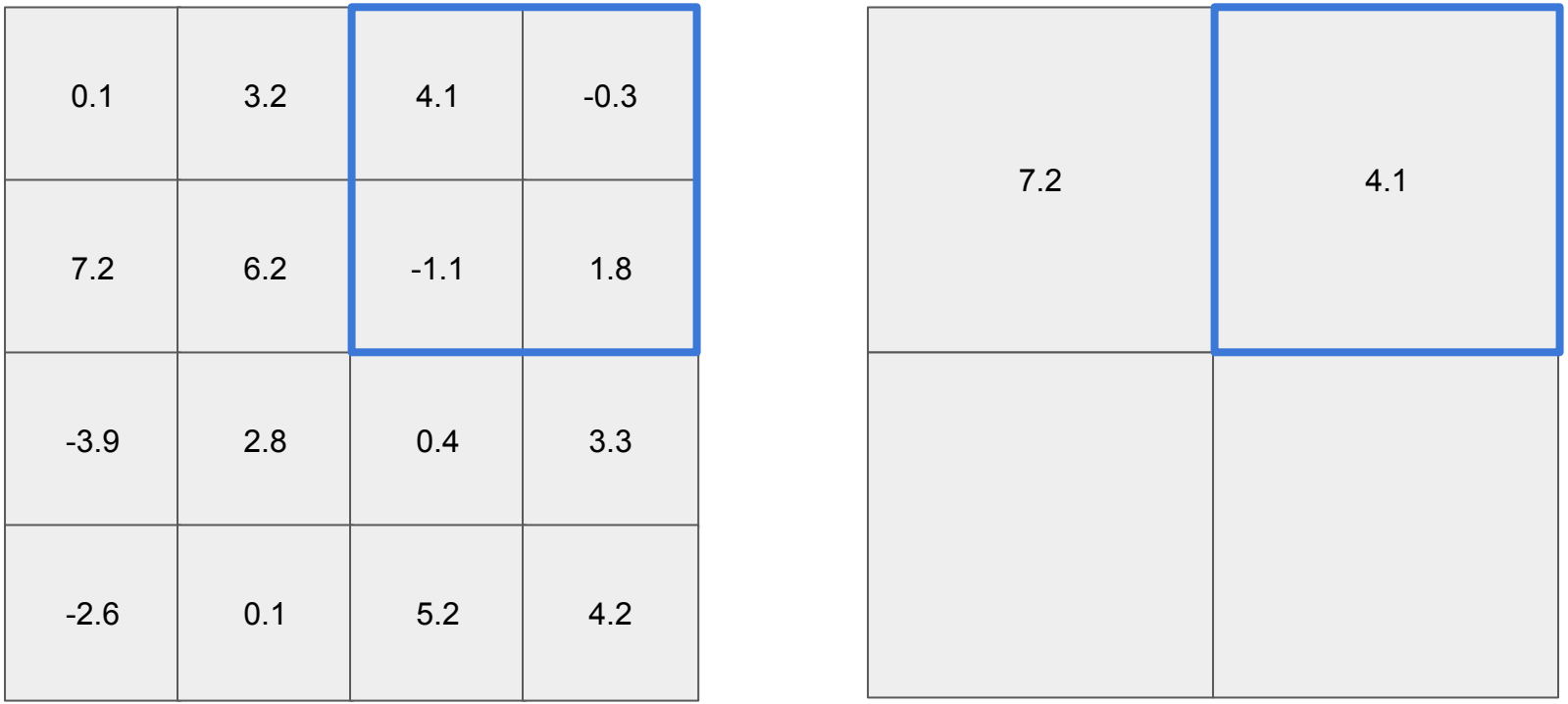

Pooling
0.36
0.34

Pooling



MaxPool: go over patches, take maximum

Pooling



MaxPool: go over patches, take maximum

Pooling

0.1	3.2	4.1	-0.3
7.2	6.2	-1.1	1.8
-3.9	2.8	0.4	3.3
-2.6	0.1	5.2	4.2

7.2	4.1
2.8	5.2

MaxPool: go over patches, take maximum

Pooling

0.1	3.2	4.1	-0.3
7.2	6.2	-1.1	1.8
-3.9	2.8	0.4	3.3
-2.6	0.1	5.2	4.2

7.2	4.1
2.8	5.2

We reduced the spatial dimension!

MaxPool: go over patches, take maximum

Pooling

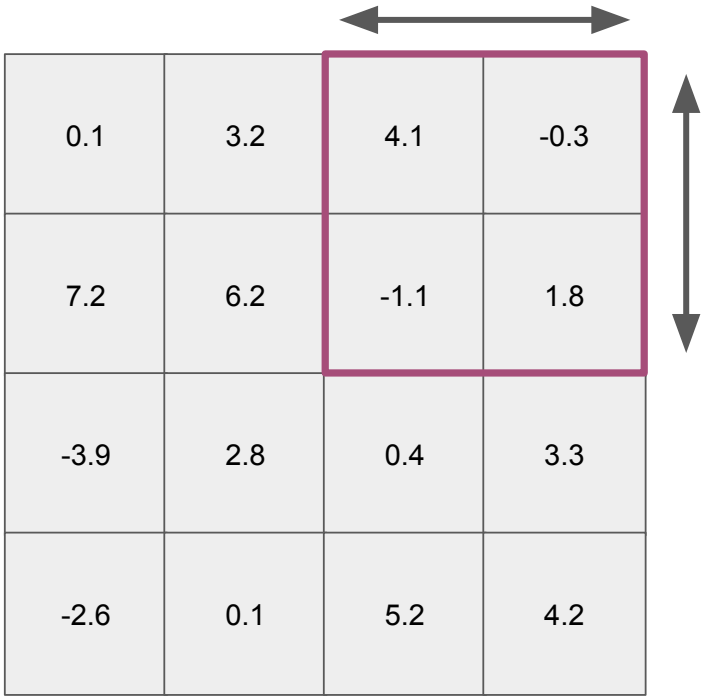
0.1	3.2	4.1	-0.3
7.2	6.2	-1.1	1.8
-3.9	2.8	0.4	3.3
-2.6	0.1	5.2	4.2

4.2	1.1
-0.9	3.3

AvgPool: go over patches, take average

Pooling

```
class torch.nn.MaxPool2d(kernel_size, stride=None, padding=0,
    dilation=1, return_indices=False, ceil_mode=False)
```

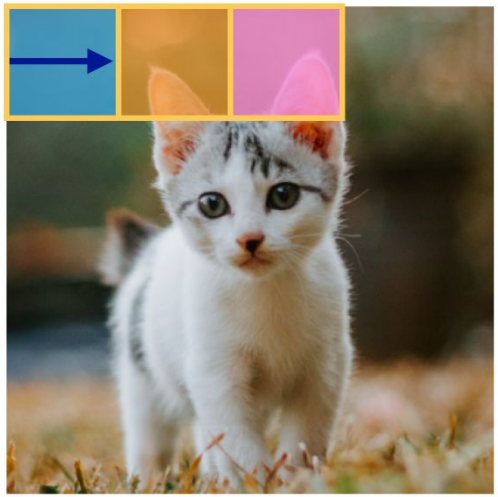


In our example, kernel size is (2, 2)

Pooling

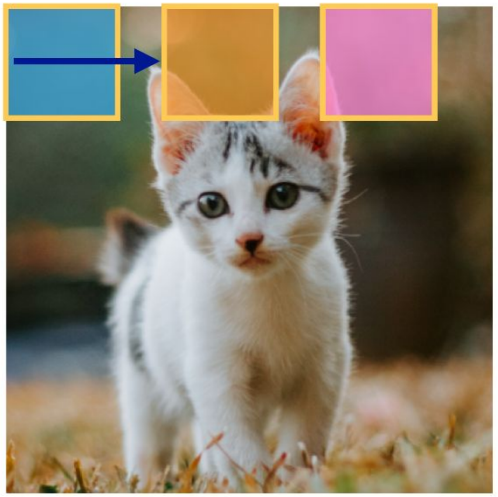
```
class torch.nn.MaxPool2d(kernel_size, stride=None, padding=0,
    dilation=1, return_indices=False, ceil_mode=False)
```

Default value is `kernel_size`



Smaller stride

Larger stride



Pooling

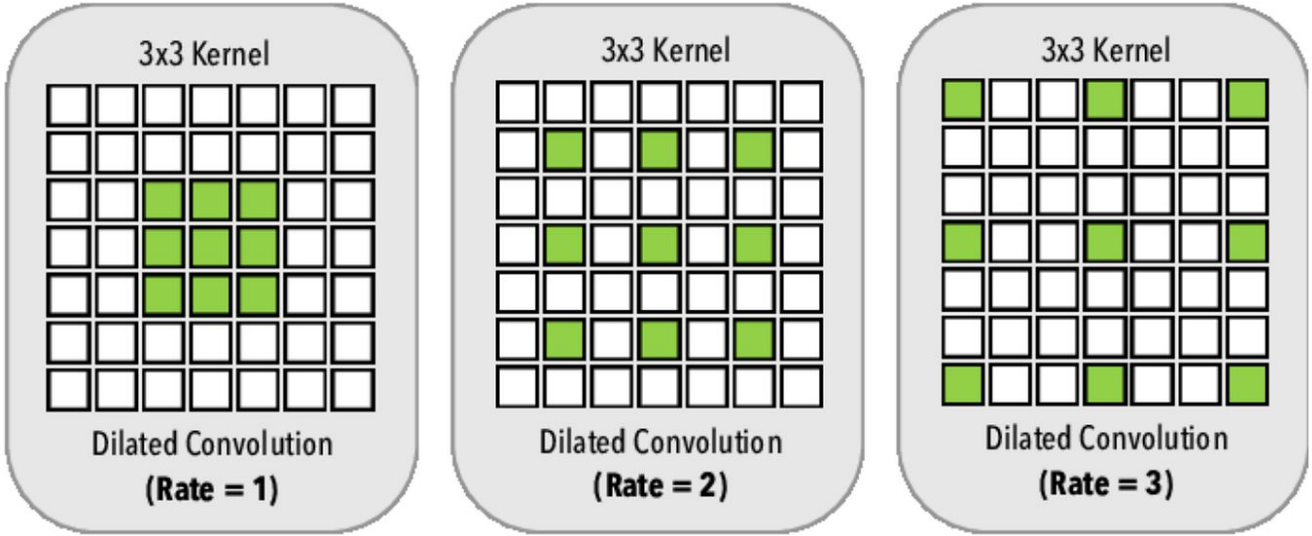
```
class torch.nn.MaxPool2d(kernel_size, stride=None, padding=0,
    dilation=1, return_indices=False, ceil_mode=False)
```

0	0	0	0	0	0
0	0.1	3.2	4.1	-0.3	0
0	7.2	6.2	-1.1	1.8	0
0	-3.9	2.8	0.4	3.3	0
0	-2.6	0.1	5.2	4.2	0
0	0	0	0	0	0

We can add a border of 0's around the image; sometimes useful for controlling output shape

Pooling

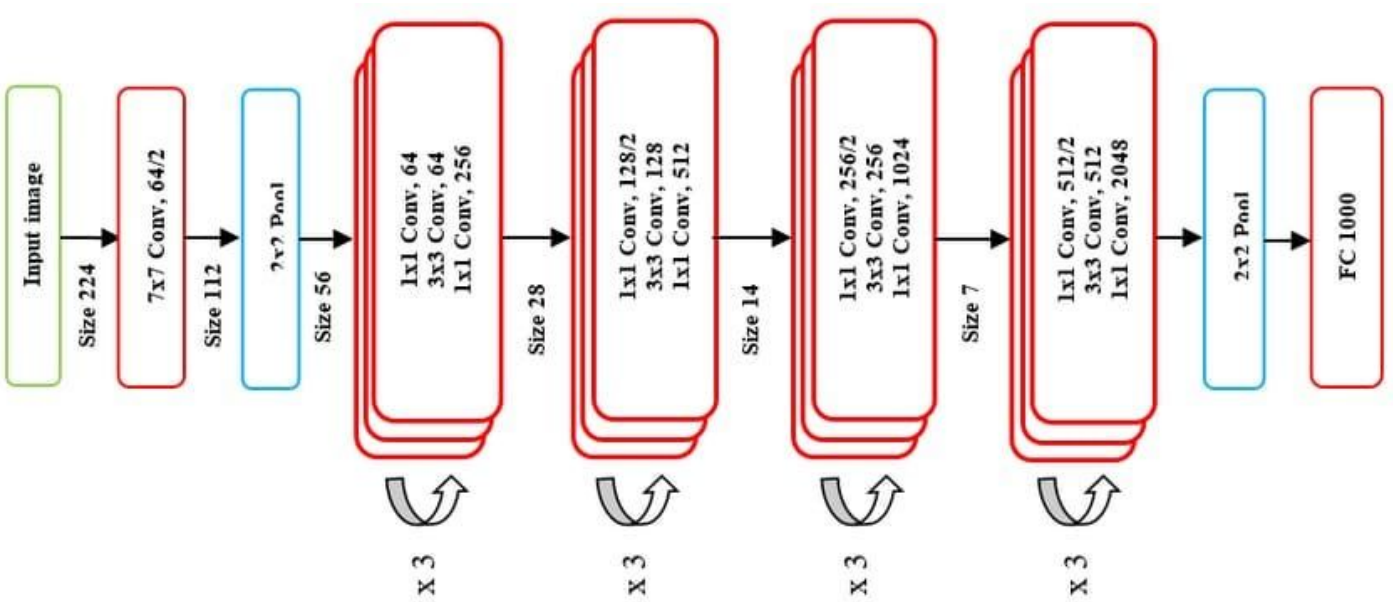
```
class torch.nn.MaxPool2d(kernel_size, stride=None, padding=0,
    dilation=1, return_indices=False, ceil_mode=False)
```



Dilation allows you to skip some pixels, covering more space; also for convolution

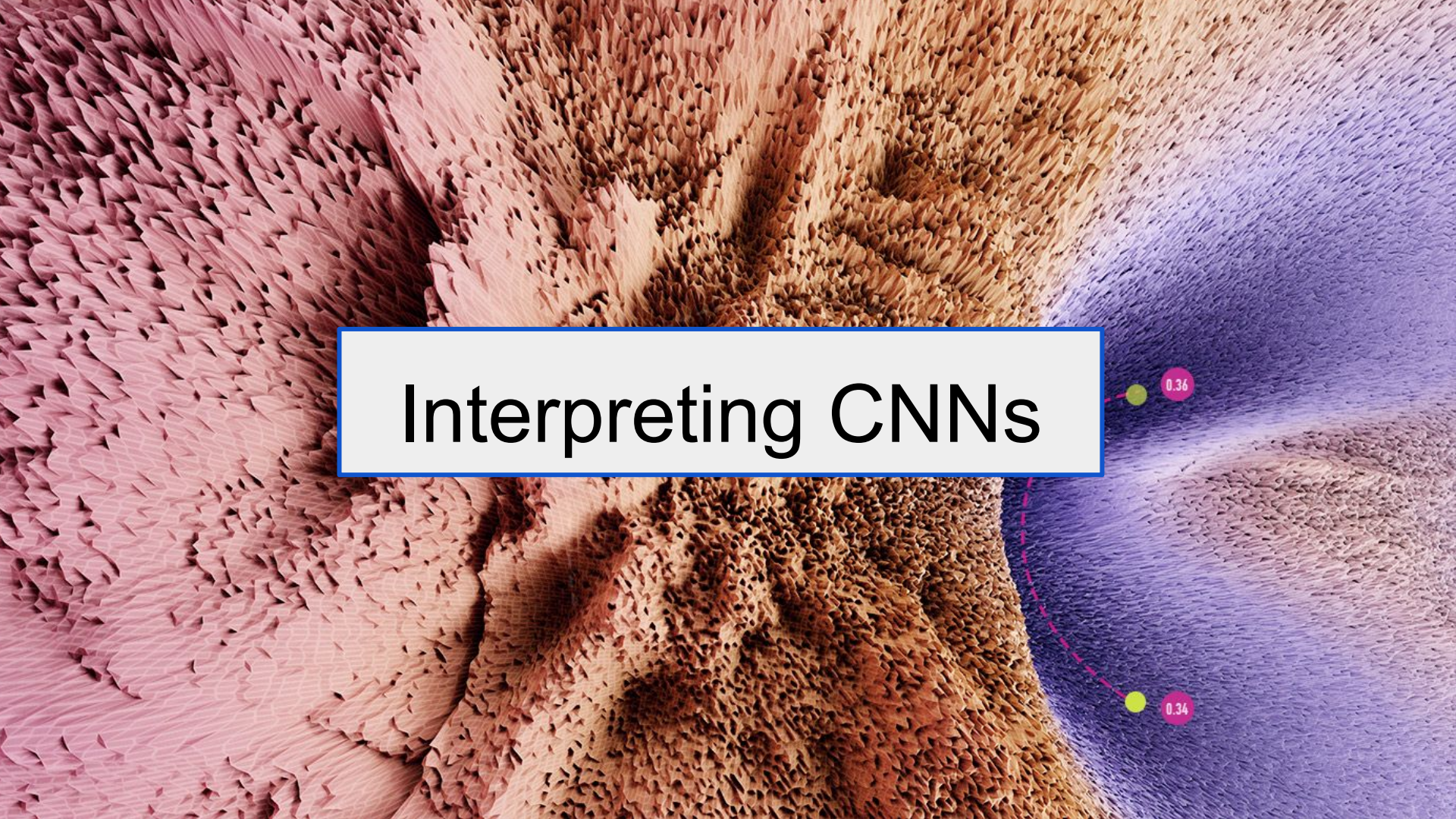

Making a ResNet
0.36
0.34

ResNet

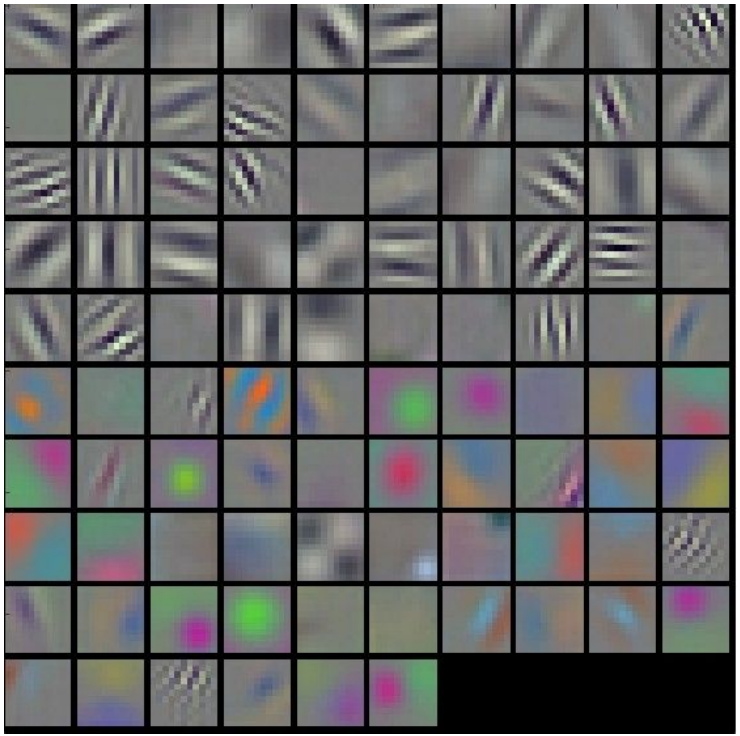


We now know all the components of a ResNet.

Let's go through part 2 of demo-cnn.ipynb.


Interpreting CNNs
0.36
0.34

Interpretability



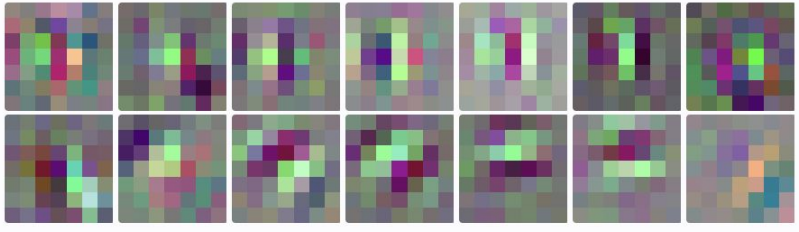
We can visualize and interpret the filters in the first layer:

- Edge detectors at different slopes
- Color blob detectors

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

Interpretability

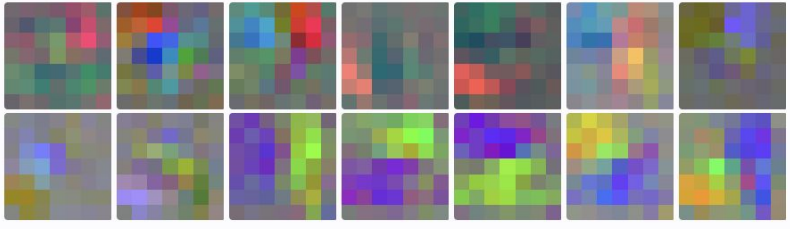
Gabor Filters 44%



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`).

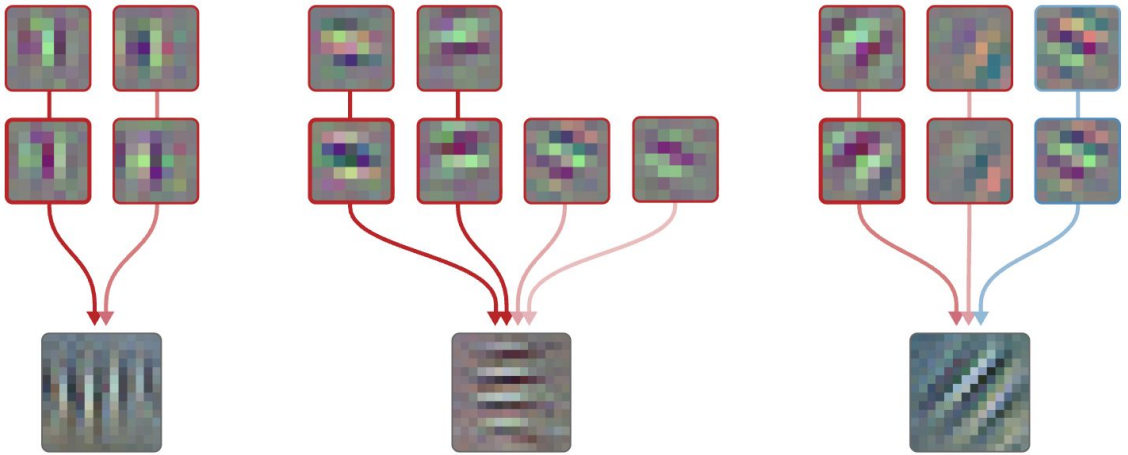
We will be looking at patches that maximally activate different filters.

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%

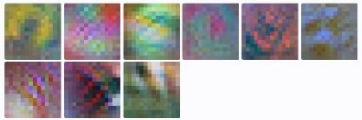
Show all 11 neurons.

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%

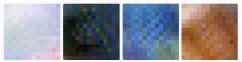


These units respond to mixtures of colors without an obvious strong spatial structure preference.

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%

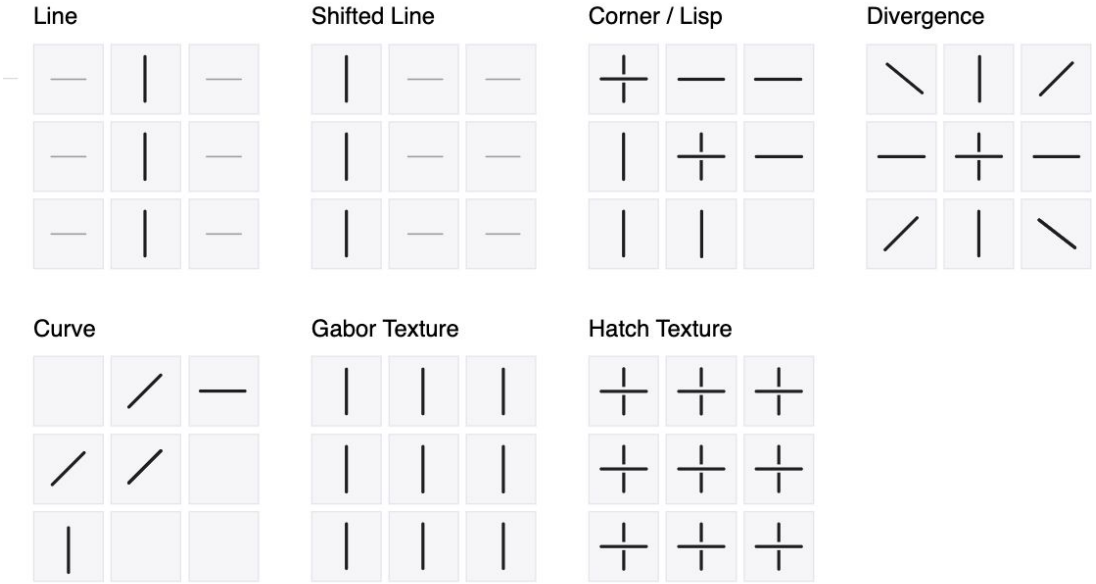
Units that don't fit in another category.

hatch 2%

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

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

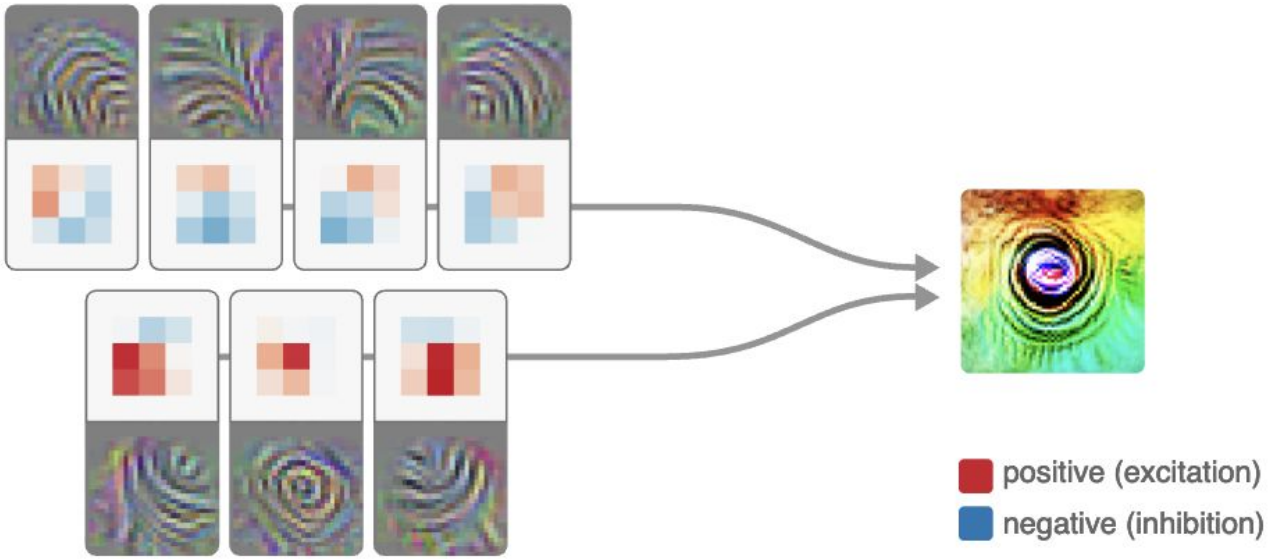
Interpretability



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/

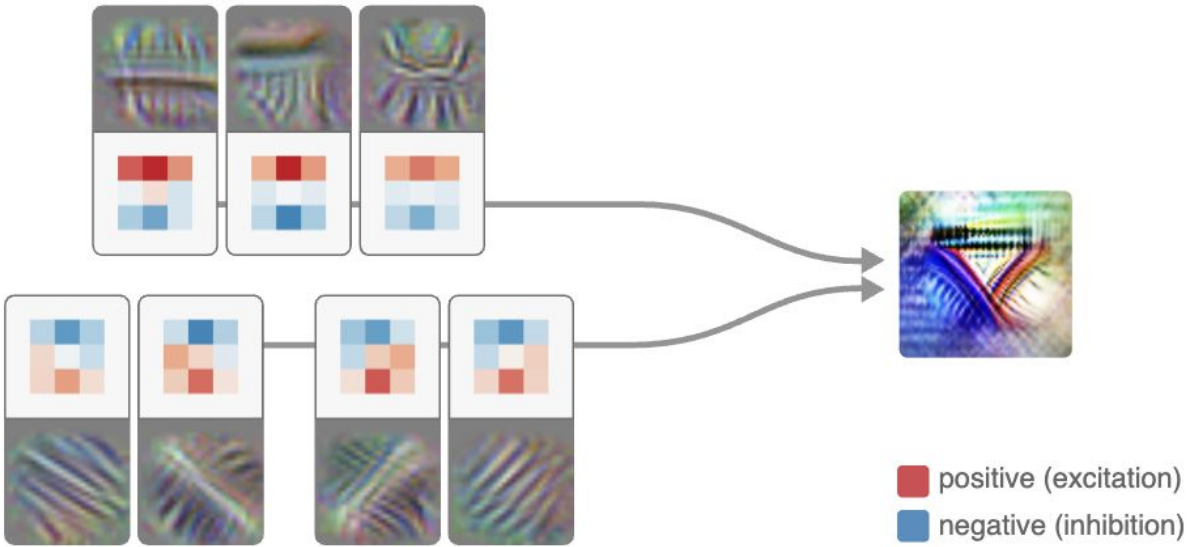
Interpretability



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

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

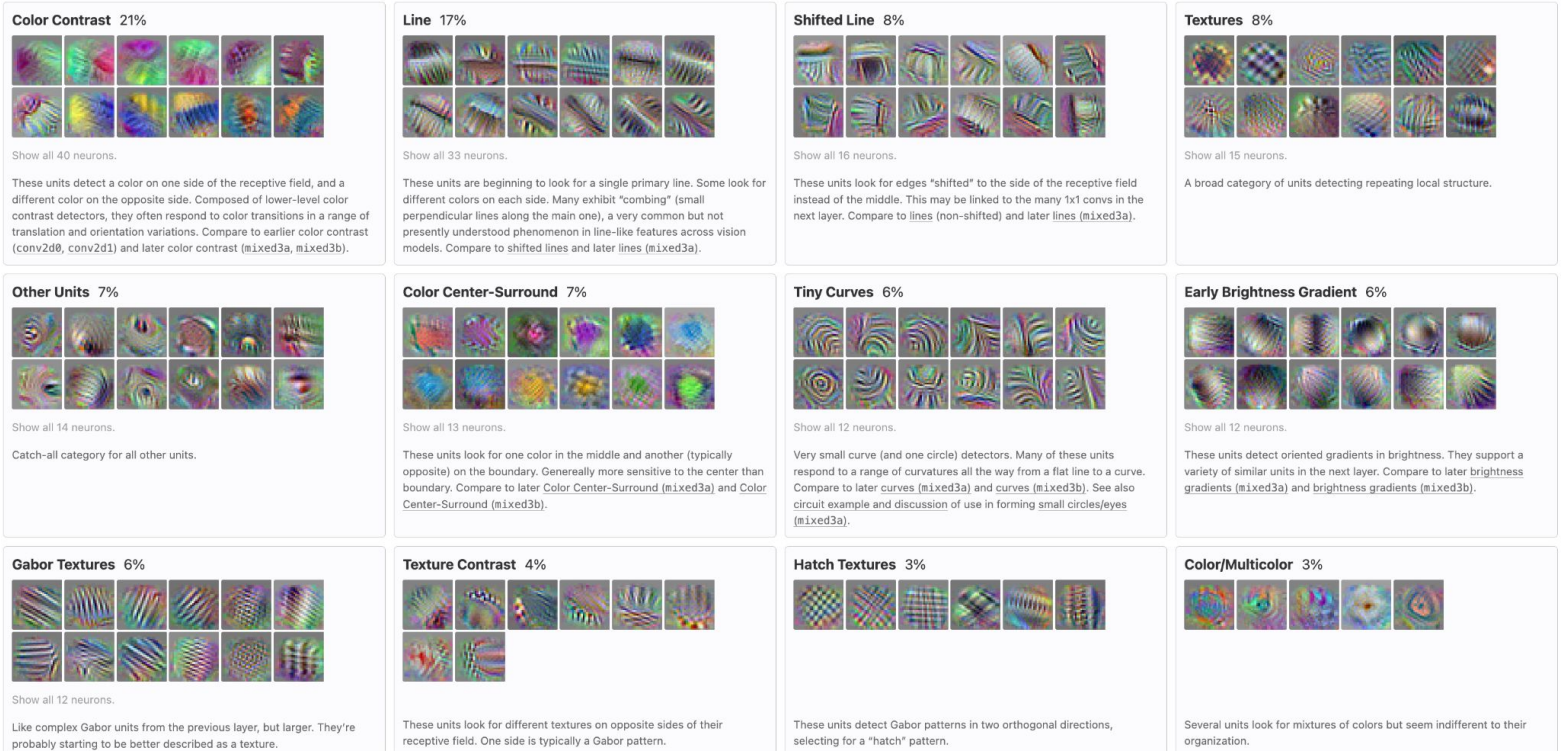
Interpretability



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

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

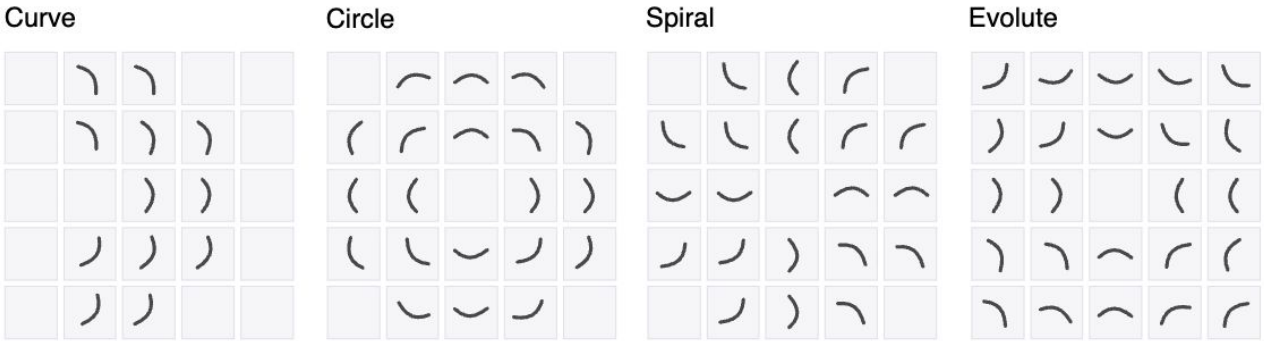
Interpretability



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.

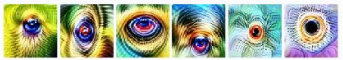
Square / Grid 2%

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%

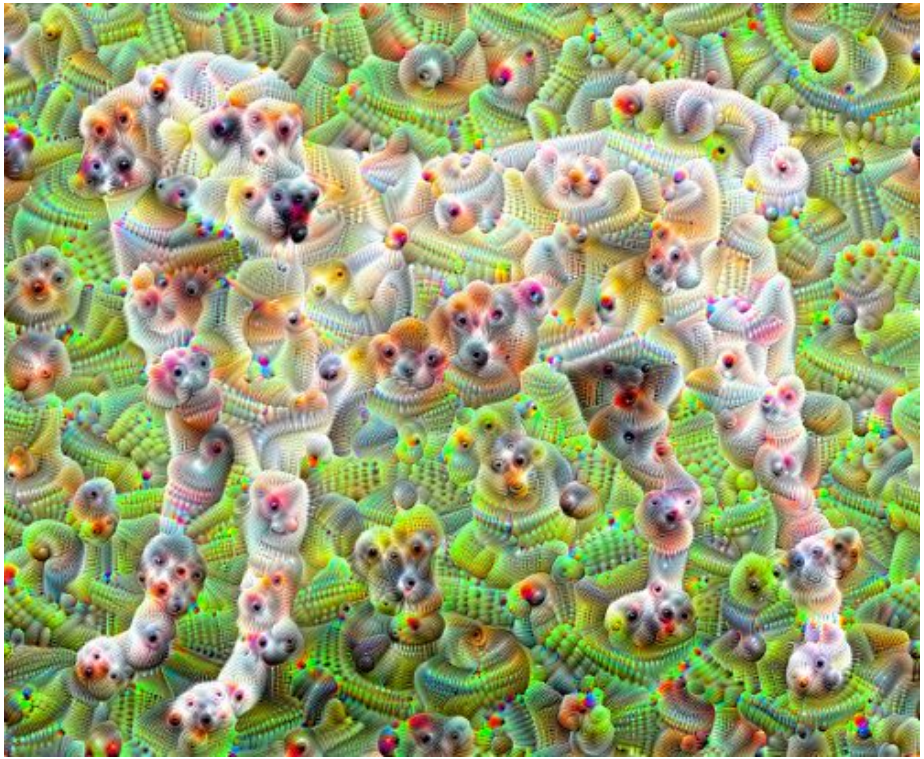
Piece together curves in a circle or partial circle. Opposite of evolute.

Circle Cluster 1%

Units detecting circles and curves without necesarily requiring spatial coherrence.

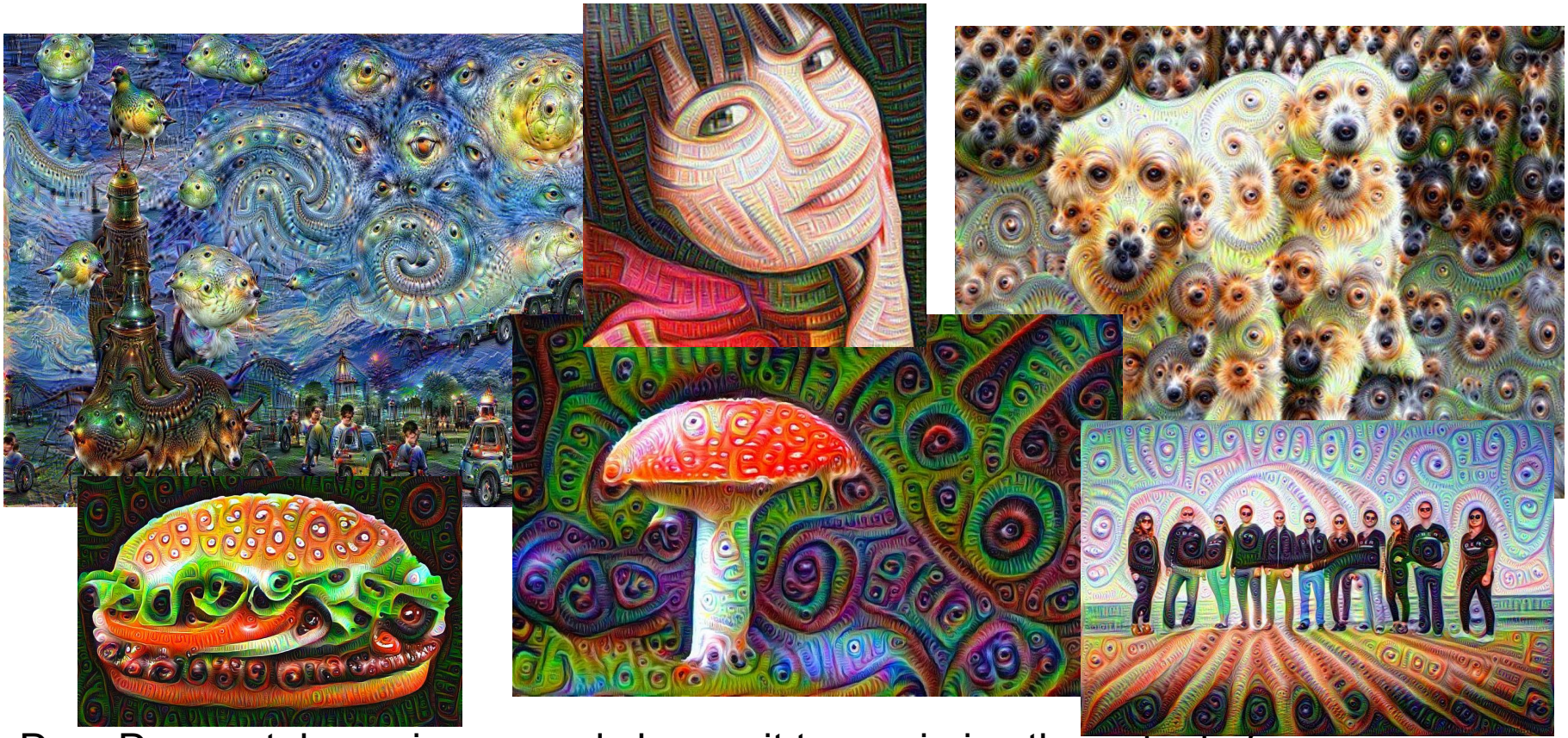
https://distill.pub/2020/circuits/early-vision/

Interpretability

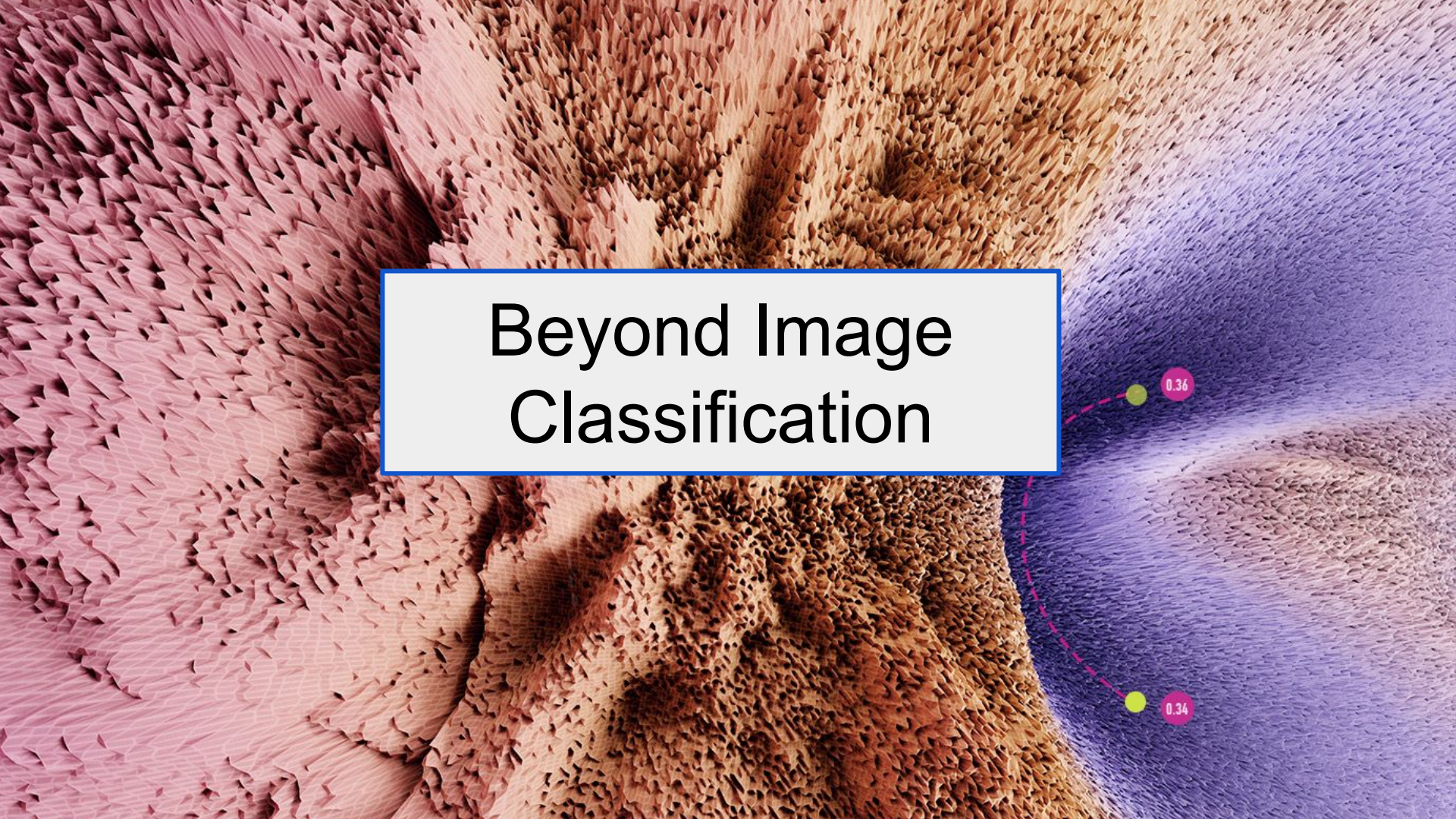


DeepDream: take an image and change it to maximize the output of a neuron.

Interpretability

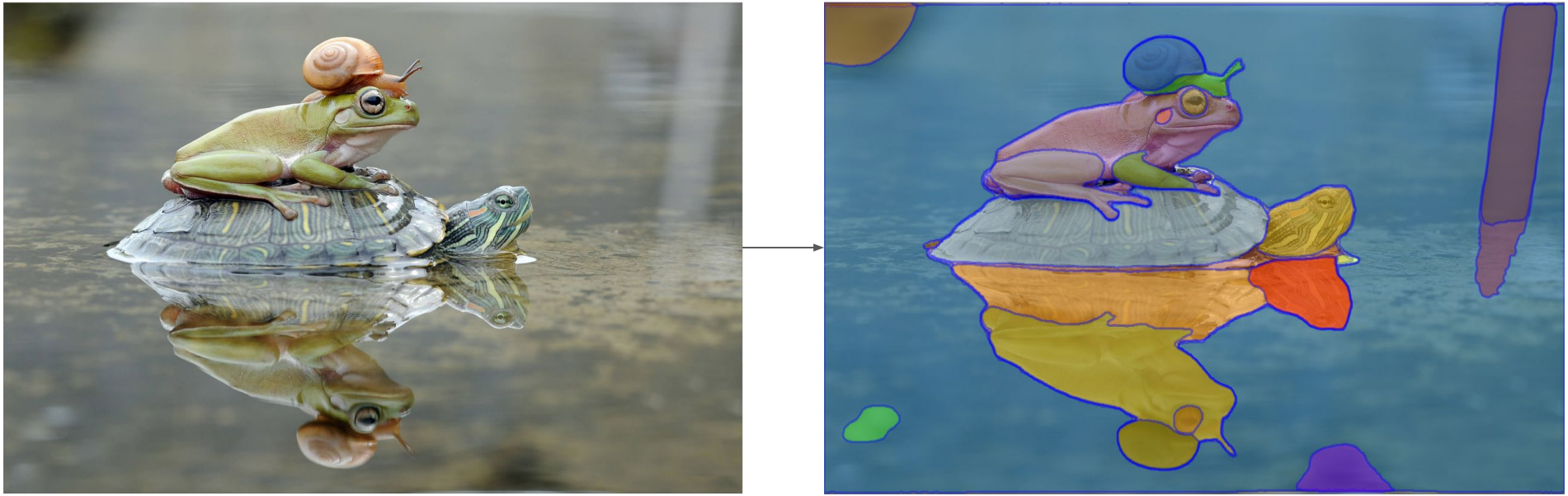


DeepDream: take an image and change it to maximize the output of a neuron.


Beyond Image Classification
0.36
0.34

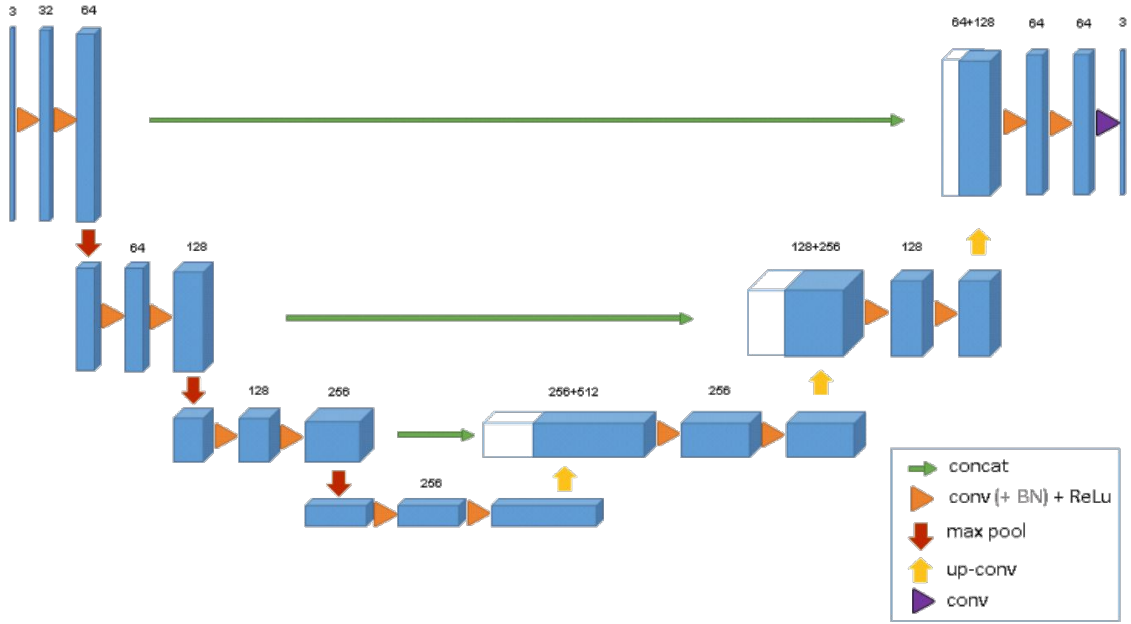
Beyond image classification

Segmentation: identify locations of objects



https://segment-anything.com/demo

Beyond image classification



Reduce spatial dimension, increase channels; then, go back

Beyond image classification

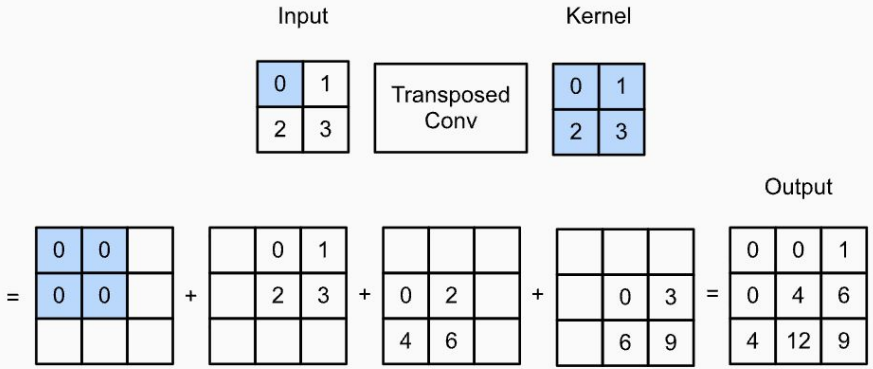
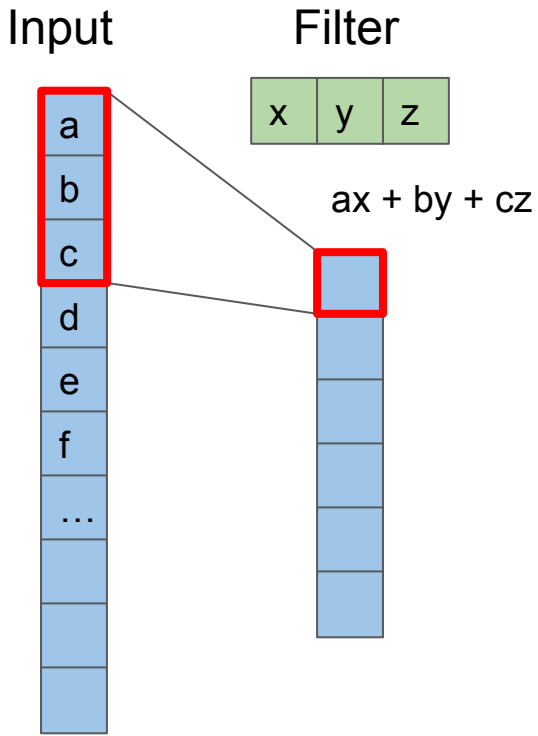


Fig. 14.10.1 Transposed convolution with a 2×2 kernel. The shaded portions are a portion of an intermediate tensor as well as the input and kernel tensor elements used for the computation.

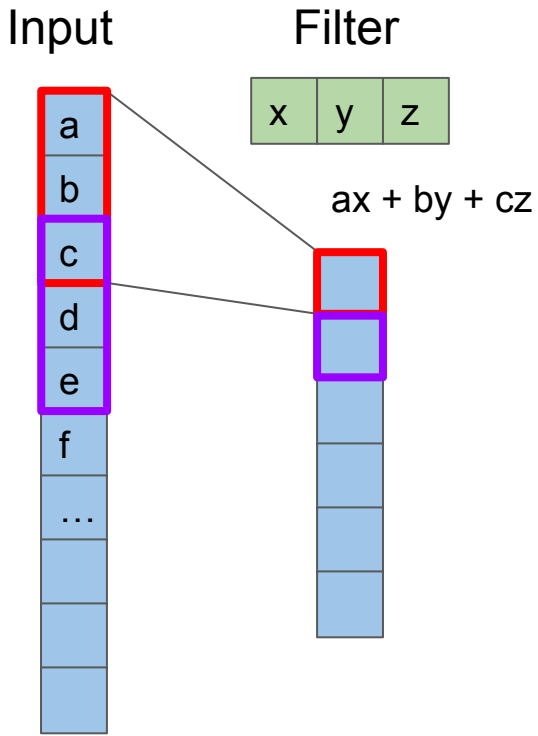
Transposed convolution can be used to increase the spatial dimension of the input.

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

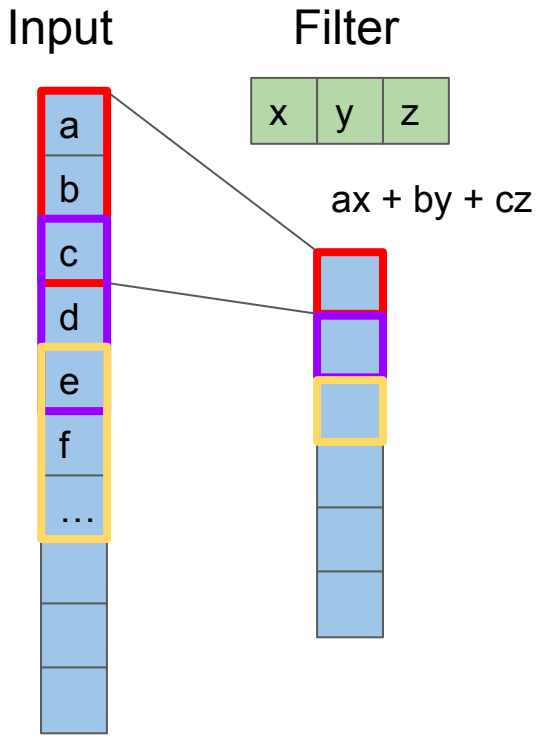
Beyond image classification



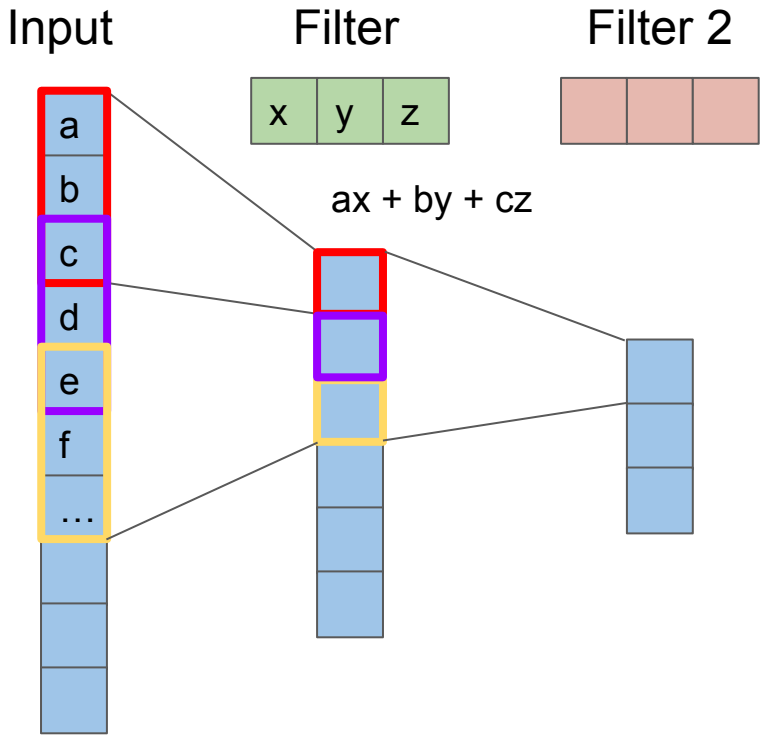
Beyond image classification



Beyond image classification

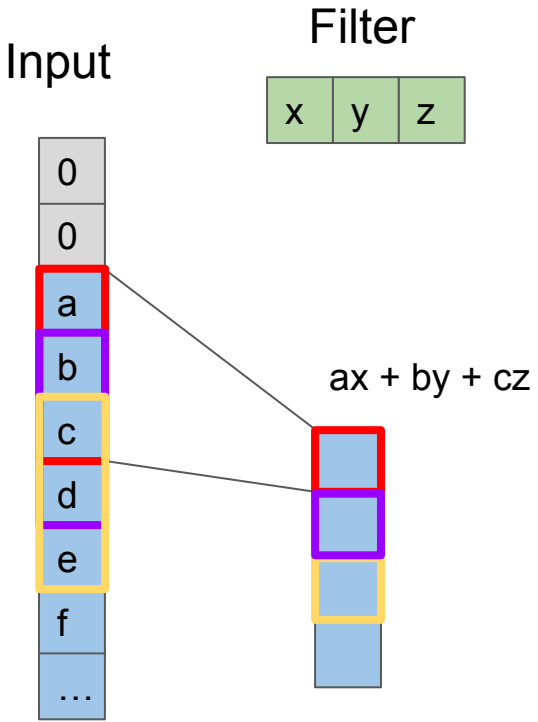


Beyond image classification



Later layers have a bigger receptive field

Beyond image classification

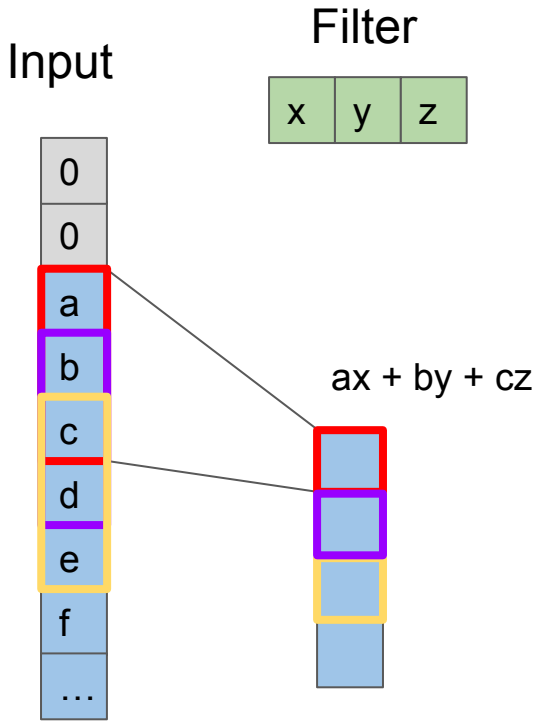


Stride 1, zero padding

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

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:

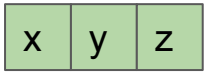
- Locality
- Weight sharing

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

Beyond image classification

Input

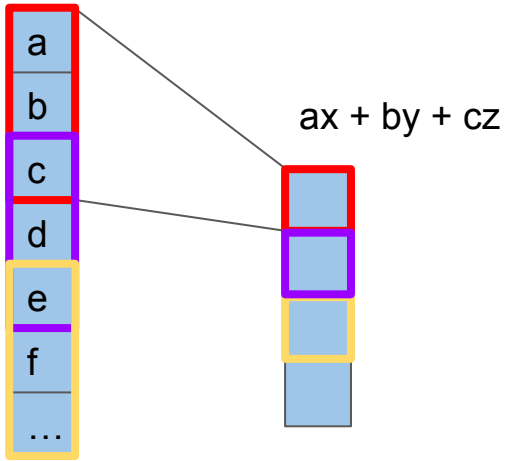
Filter



Stride 2, no padding

Two main features:

- Locality
- Weight sharing



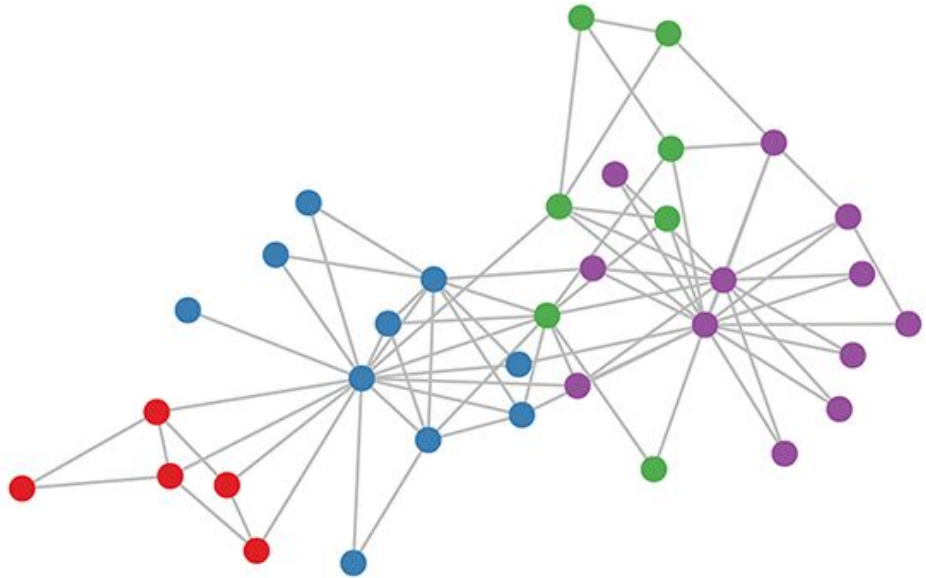
$$\mathbf{H}_{\text{valid, stride=2}} = \begin{bmatrix} z & y & x & 0 & 0 & 0 & 0 \\ 0 & 0 & z & y & x & 0 & 0 \\ 0 & 0 & 0 & 0 & z & y & x \end{bmatrix}$$

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

Beyond image classification

Problems on graphs:

- Community detection
- Node classification
- …

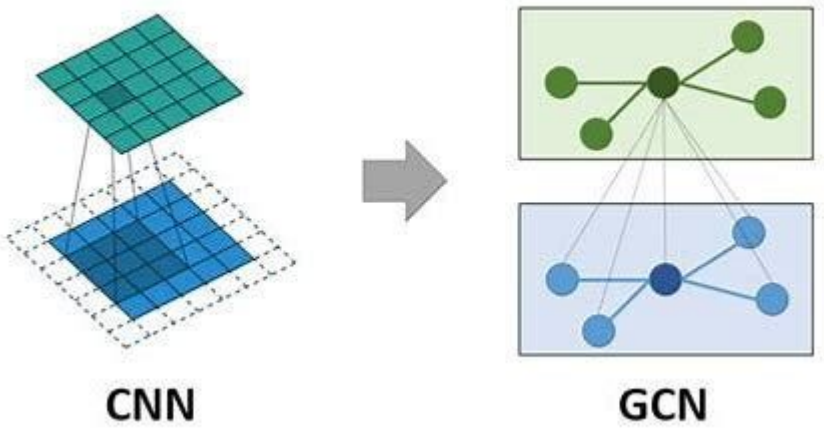


https://tkipf.github.io/graph-convolutional-networks/

Beyond image classification

We can generalize convolution to graphs:

- Aggregate across neighbors
- No spatial arrangement, so the “filters” do not depend on the neighbors

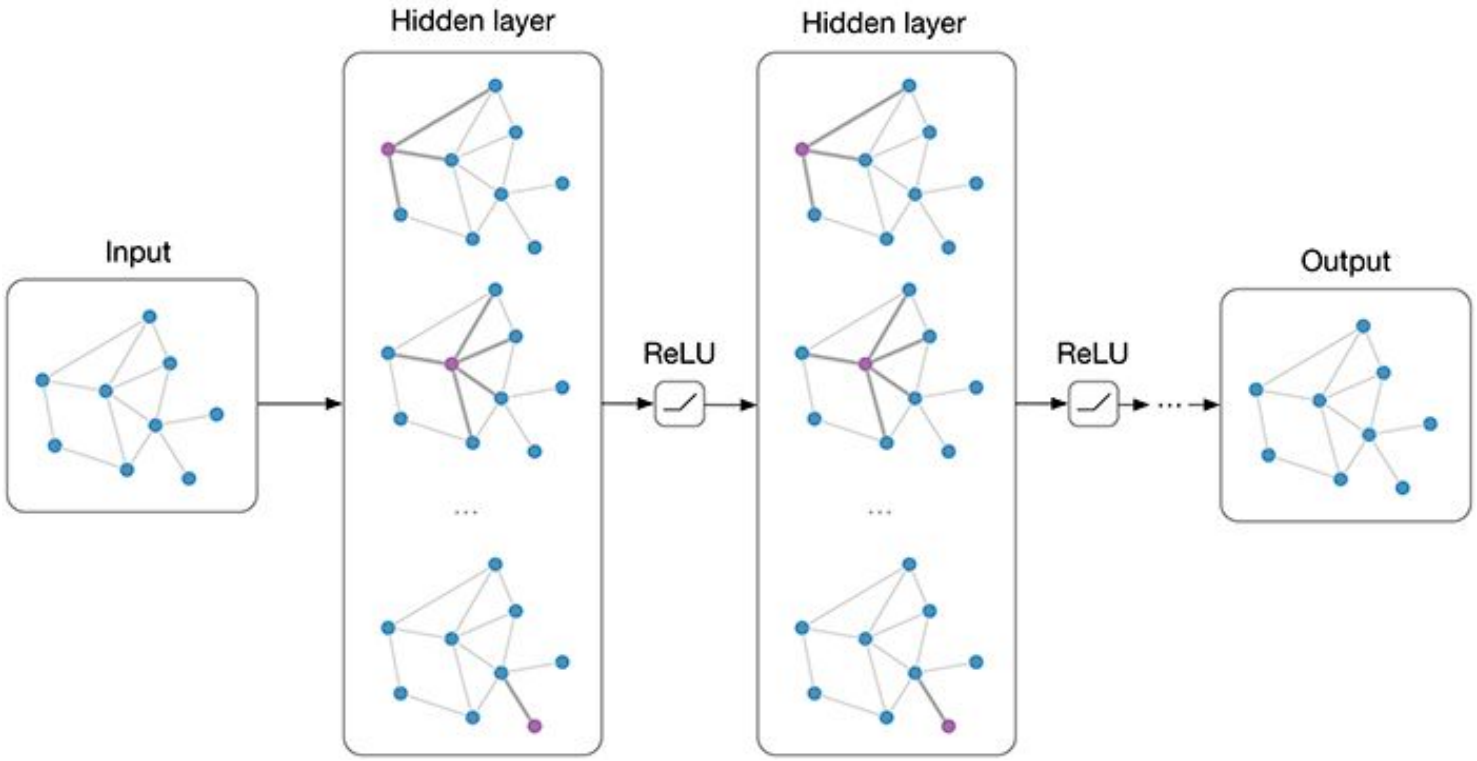


$$h_v^{(l+1)} = \sigma\left(W^{(l)} \cdot \mathrm{AGGREGATE}(\{h_u^{(l)} : u \in N(v)\})\right)$$

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

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

Beyond image classification



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