Image classification

Lecture 3 • Sept 19th, 2019

Today's agenda

Administrative stuff

- Cloud credits
- Office hours updated
- A2 will be posted to CW tomorrow night

Topics

- CNNs
- Transfer learning
- Data augmentation
- Visualizing activatations

Cloud credits

Please fill out this form if you'd like some.

Notes

- Optional. You can do the course project for free in Colab. Credits may be helpful if you'd like to use Cloud Storage (instead of Google Drive), or if you're working in a different environment.
- Google Cloud offers a $300 free trial. These are on top of that.
- Keep an eye on billing / usage (be careful you're not accidentally charged).

Office hours update

Benedikt Schifferer	Monday 9:30am - 11am	Mudd CS TA room
Su Ji Park	Tues 1:30pm - 3pm	Mudd CS TA room
Pratik Dubal	Wed 10am - 11:30am	Mudd CS TA room
Kunyan Han	Thurs 1:30pm - 3pm	Mudd CS TA Room
Josh Gordon	Thurs 5:30pm - 6:30pm	Mudd 417
Pengyu Chen	Fri 11am - 12:30pm	Mudd CS TA room

Questions from last time

!nvidia-smi

Convolution

Not a Deep Learning concept

```
import scipy
from skimage import color, data
import matplotlib.pyplot as plt
img = data.astronaut()
img = color.rgb2gray(img)
plt.axis('off')
plt.imshow(img, cmap=plt.cm.gray)
```

Convolution example

Quick discussion: does anyone know who this is?

-1	-1	-1
-1	8	-1
-1	-1	-1

Notes

Edge detection intuition: dot product of the filter with a region of the image will be zero if all the pixels around the border have the same value as the center.

Convolution example

Eileen Collins

-1	-1	-1
-1	8	-1
-1	-1	-1

Notes

Edge detection intuition: dot product of the filter with a region of the image will be zero if all the pixels around the border have the same value as the center.

Launched the Chandra X-ray Observatory, in 1999

A simple edge detector

```
kernel = np.array([[-1,-1,-1],
                   [-1,8,-1],
                   [-1,-1,-1]])
result = scipy.signal.convolve2d(img, kernel, 'same')
plt.axis('off')
plt.imshow(result, cmap=plt.cm.gray)
```

Convolution example

Worked example of a dot product shortly, in case you're new to it.

Eileen Collins

-1	-1	-1
-1	8	-1
-1	-1	-1

Notes

Edge detection intuition: dot product of the filter with a region of the image will be zero if all the pixels around the border have the same value as the center.

Easier to see with seismic

Worked example of a dot product shortly, in case you're new to it

Eileen Collins

-1	-1	-1
-1	8	-1
-1	-1	-1

Notes

Edge detection intuition: dot product of the filter with a region of the image will be zero if all the pixels around the border have the same value as the center.

Key ideas

Efficiency: you found edges **everywhere** in the image with just a 3x3 filter!

- Contrast this to a dense layer (which would need to learn to detect edges **separately** at each location in the image).

Unlike hand-designed filters in Photoshop, ours will be learned.

- **Not limited to 2d**

Example

2	0	1	1
0	1	0	0
0	0	1	0
0	3	0	0

An input image
(no padding)

1	0	1
0	0	0
0	1	0

A filter
(3x3)

Output image
(after convolving with stride 1)

Example

2	0	1	1
0	1	0	0
0	0	1	0
0	3	0	0

An input image
(no padding)

1	0	1
0	0	0
0	1	0

A filter
(3x3)

3	

Output image
(after convolving with stride 1)

2*1 + 0*0 + 1*1 + 0*0 + 1*0 + 0*0 + 0*0 + 0*1 + 1*0

Example

2	0	1	1
0	1	0	0
0	0	1	0
0	3	0	0

An input image
(no padding)

1	0	1
0	0	0
0	1	0

A filter
(3x3)

3	2

Output image
(after convolving with stride 1)

Example

2	0	1	1
0	1	0	0
0	0	1	0
0	3	0	0

An input image
(no padding)

1	0	1
0	0	0
0	1	0

A filter
(3x3)

3	2
3	

Output image
(after convolving with stride 1)

Example

2	0	1	1
0	1	0	0
0	0	1	0
0	3	0	0

An input image
(no padding)

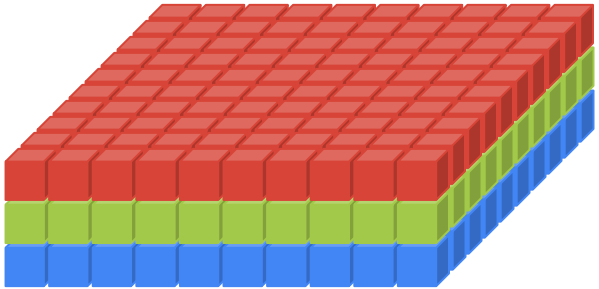
1	0	1
0	0	0
0	1	0

A filter
(3x3)

3	2
3	1

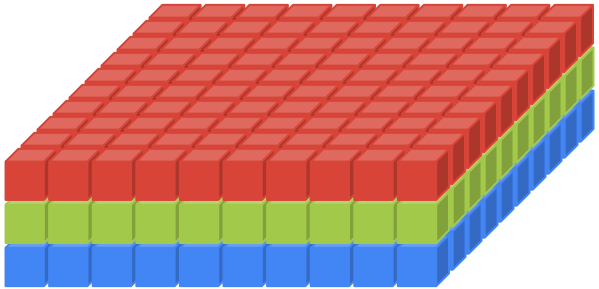
Output image
(after convolving with stride 1)

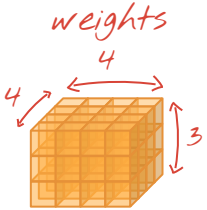
```
model = Sequential()

model.add(Conv2D(filters=4,
                 kernel_size=(4,4),
                 input_shape=(10,10,3))
```



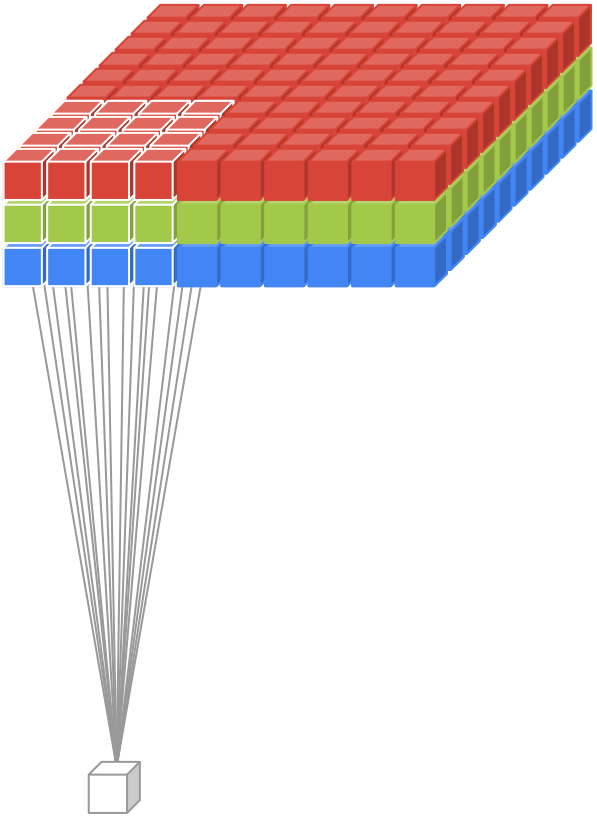
A RGB image as a 3d **volume**. Each color (or channel) is a layer.

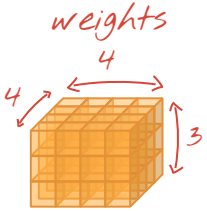
Beautiful diagrams based on slides from Martin Gorner (thank you!)



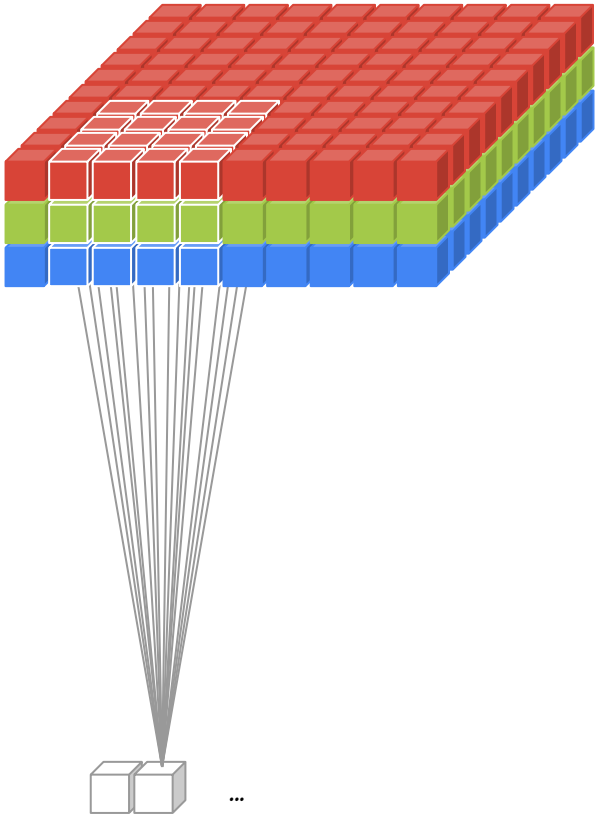


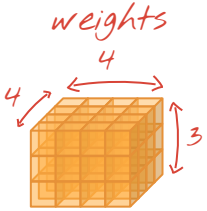
In 3d, our filters have width, height, and depth.



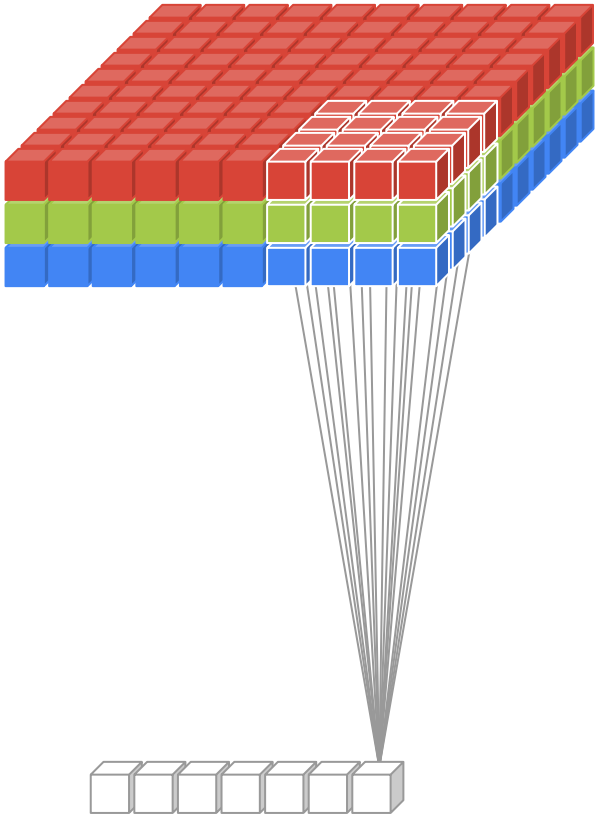


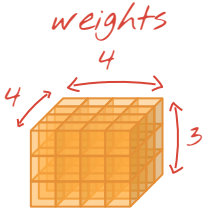
Applied in the same way as 2d (sum of weight * pixel value as they slide across the image).



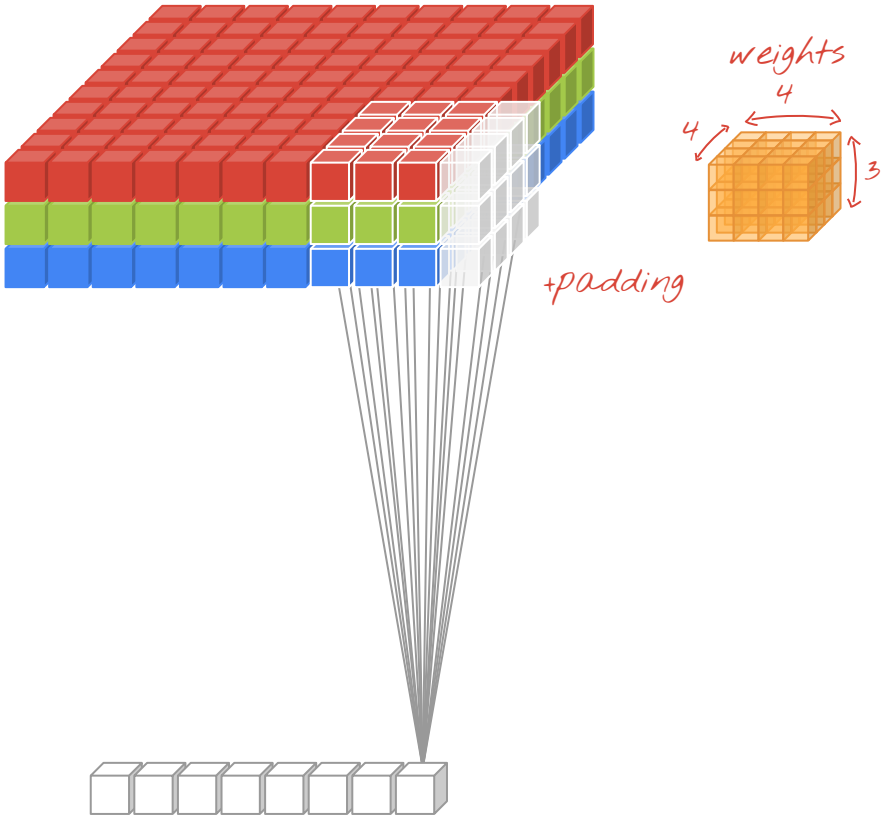


Applied in the same way as 2d (sum of weight * pixel value as they slide across the image).

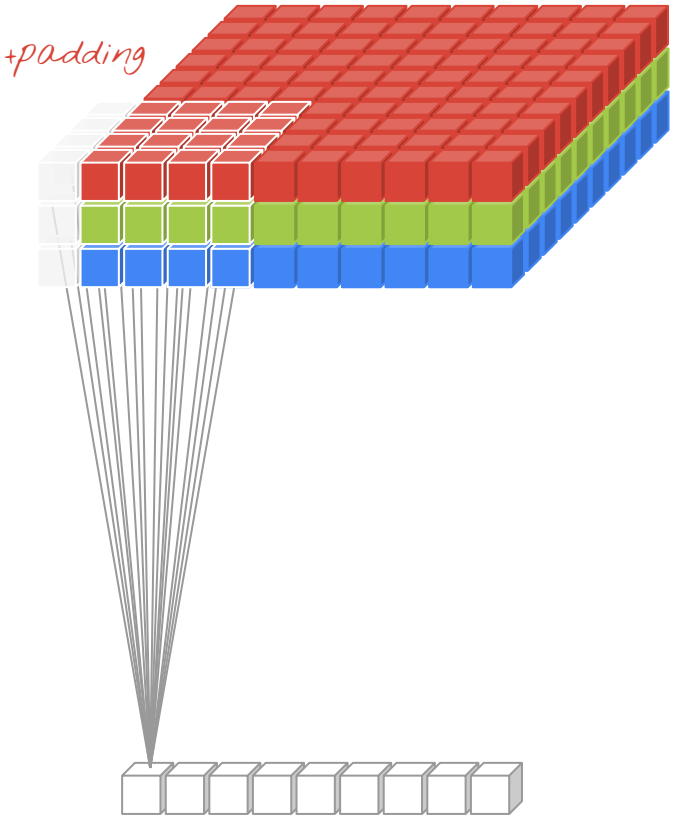


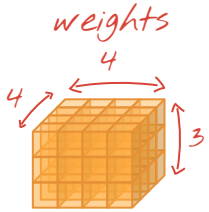


Notice the width of the output volume is smaller.



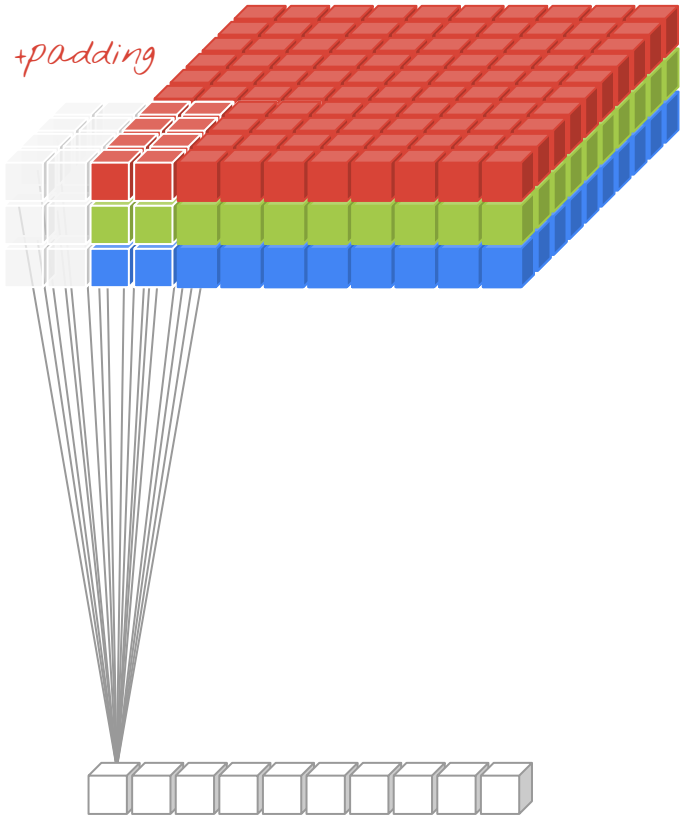
We can apply padding (zeros along the borders) to the original image to maintain the same output dimensions.

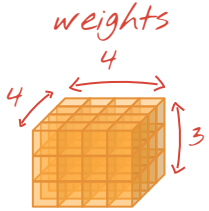




Adding more padding so the dimensions match (most libraries provide helper functions for this).

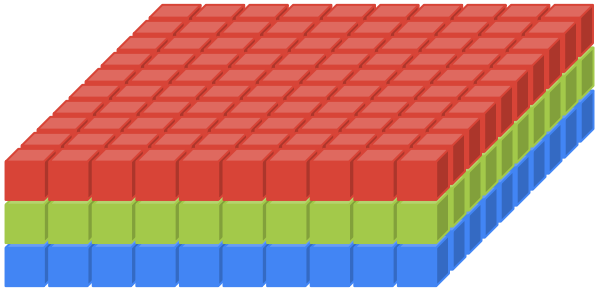
Various strategies (hint: same can be helpful if you don't want to worry about your filters perfectly dividing images / it'll add zeros around the border if necessary to keep the output dimensions the same).

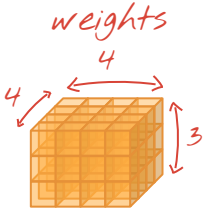




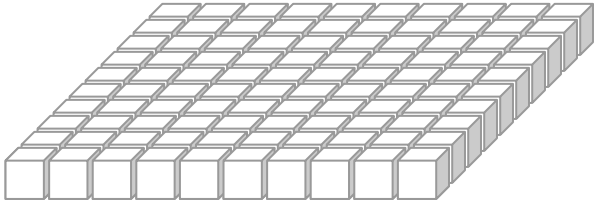
Adding more padding so the dimensions match (most libraries provide helper functions for this).

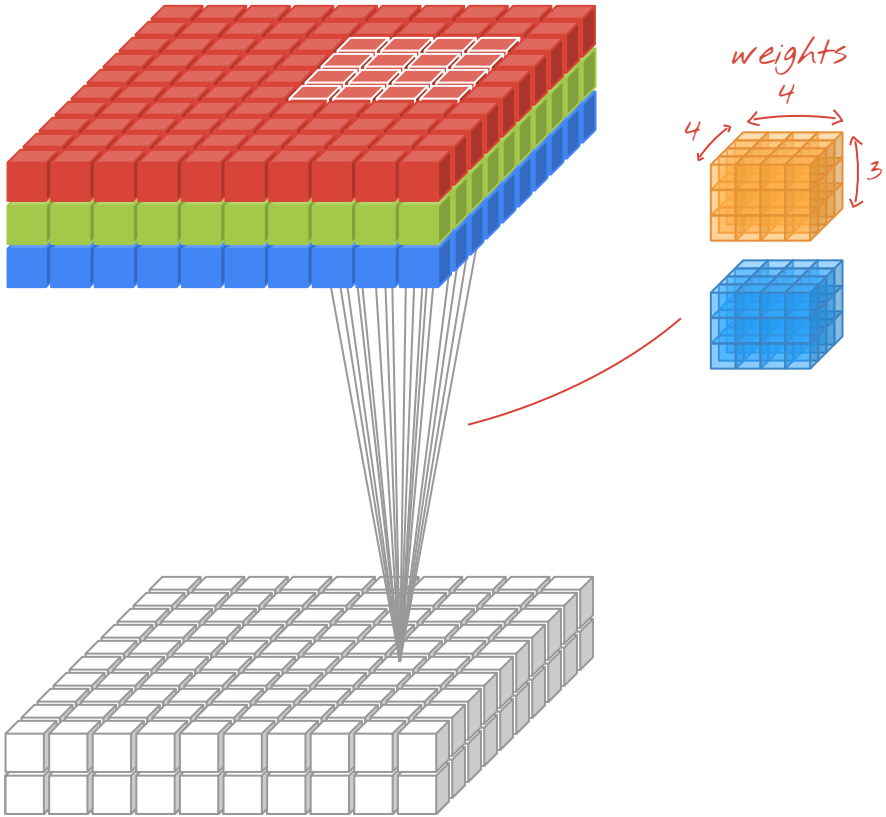
Various strategies (hint: “padding=same” can be helpful if you don’t want to worry about your filters perfectly dividing images / it’ll add zeros around the border if necessary to keep the output dimensions the same as the input).



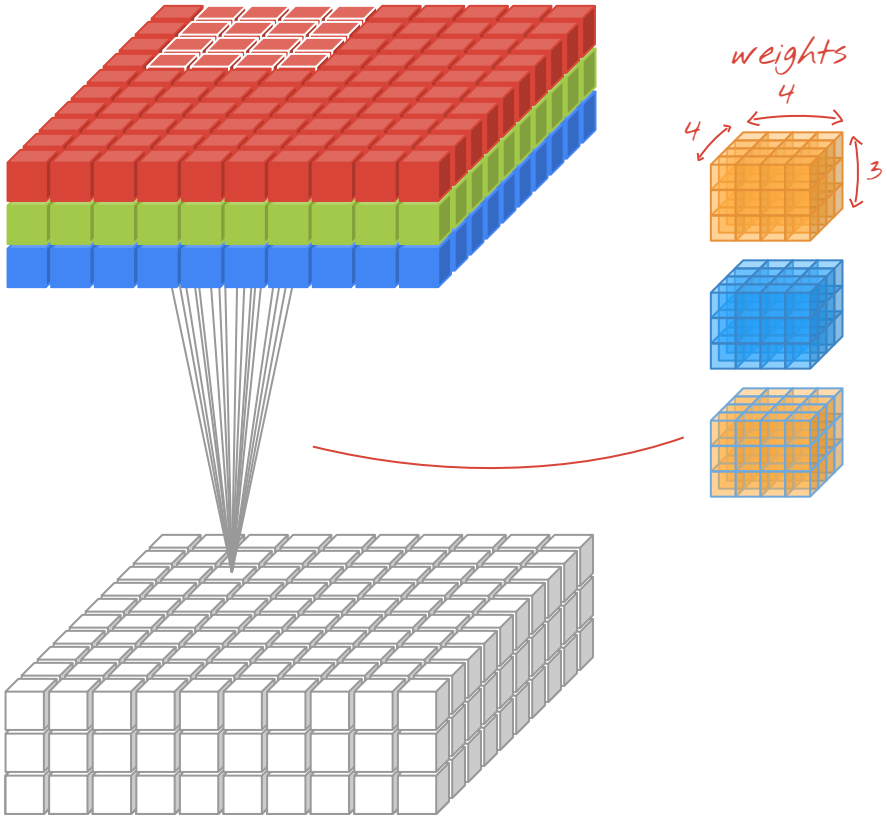


Applying the convolution over the rest of the input image.

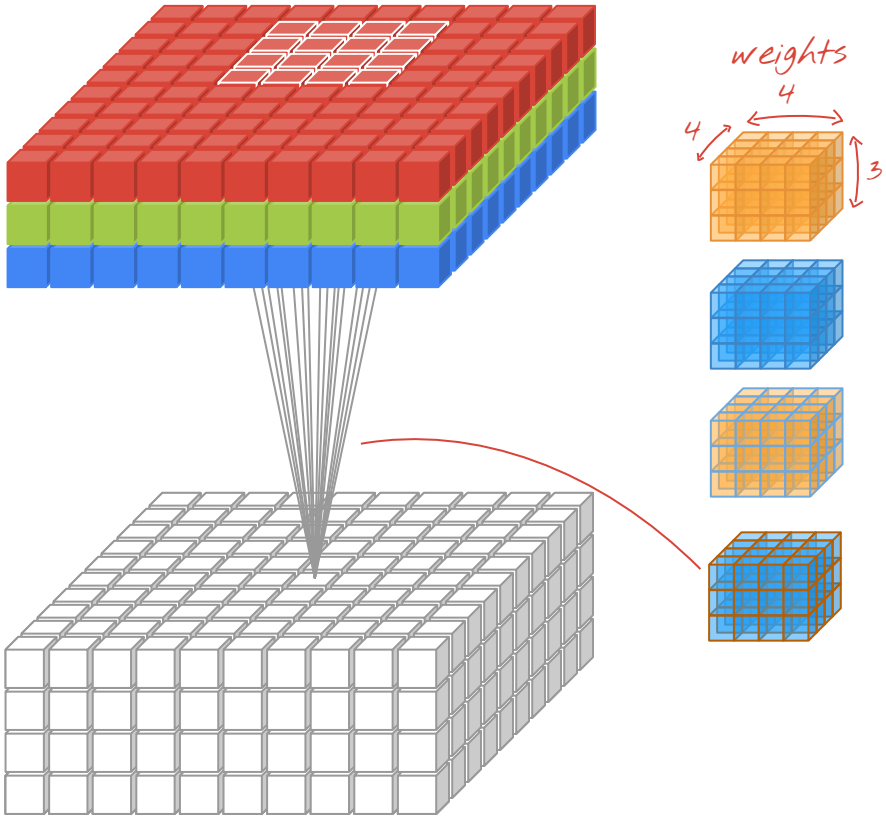




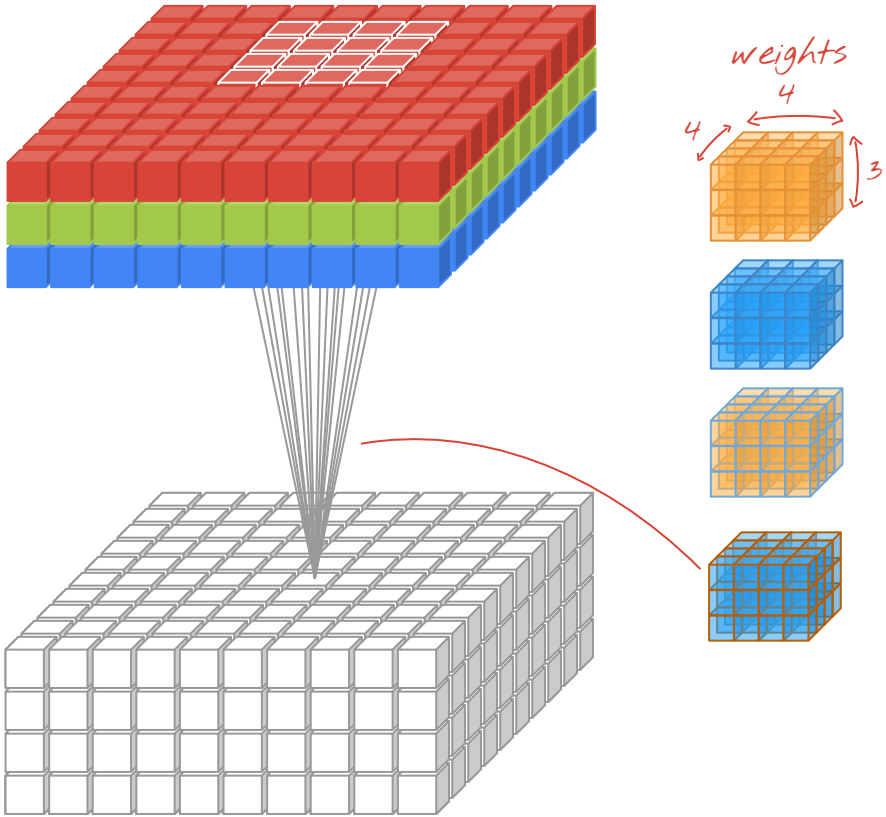
More filters, more output channels.



More filters, more output channels.



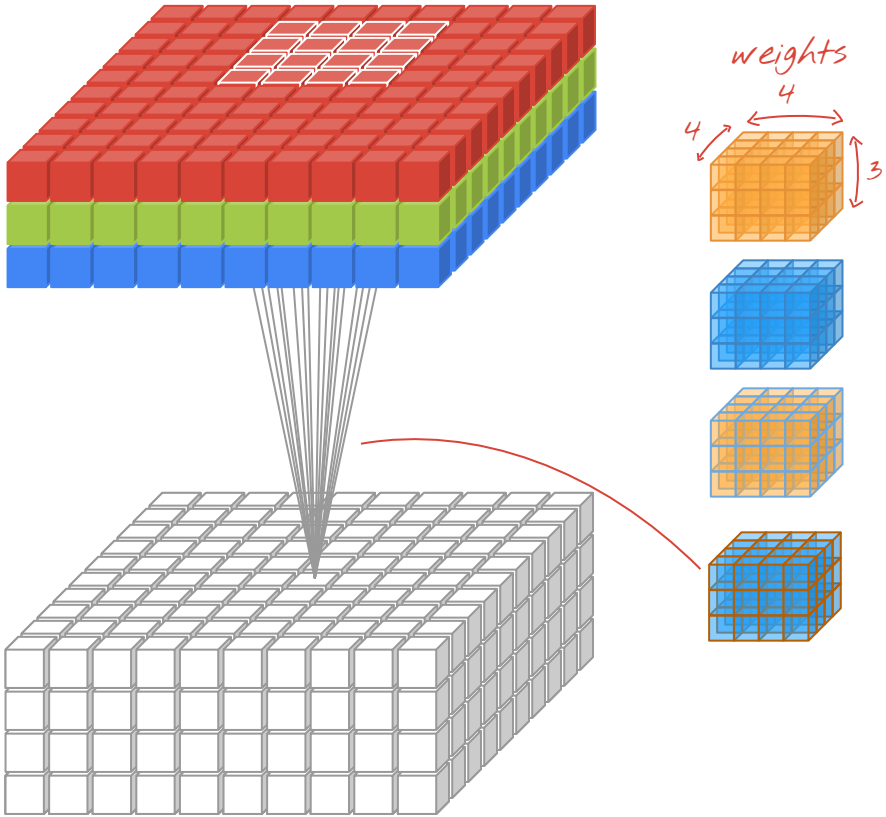
More filters, more output channels.



More filters, more output channels.

Notes:

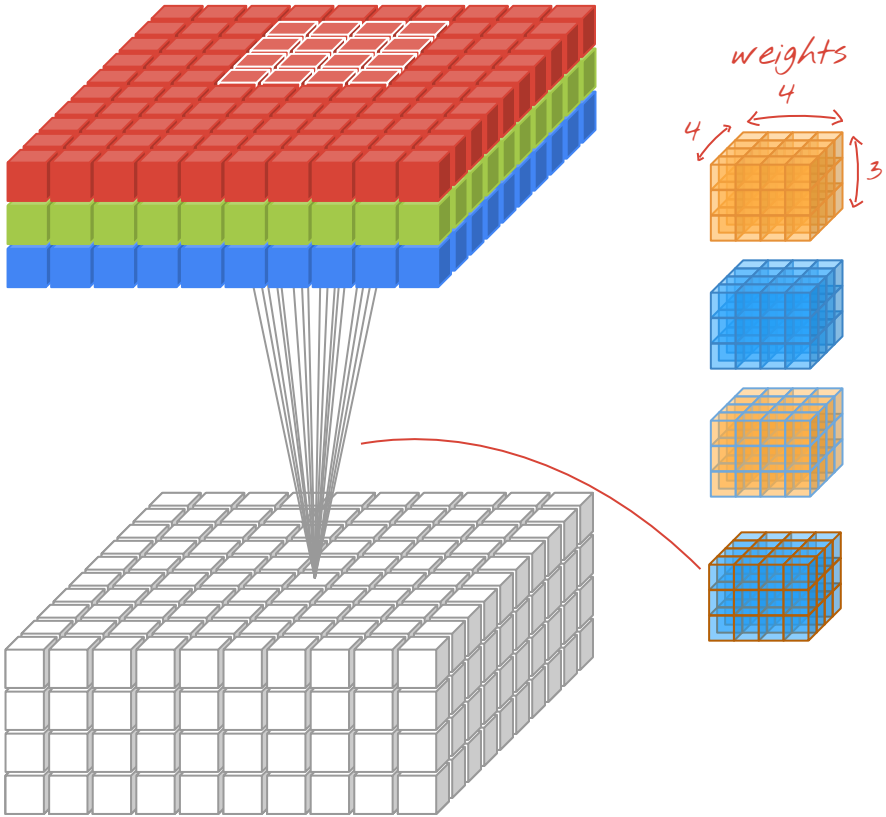
- Each filter learns one set of weights (it does not change between steps as it slides across the image)
- At this stage, we have four output channels. We can begin another layer of convolution using these as our input! The new bank of filters would each need depth 4.



More filters, more output channels.

Notes:

- Unlike the 2d filters from before, notice each filter connects to *every* input channel.
- This means they can compute sophisticated features. Initially, by looking at R,G,B - but later, by looking at combinations of learned features - like various edges, and later shapes / textures / and semantic features!



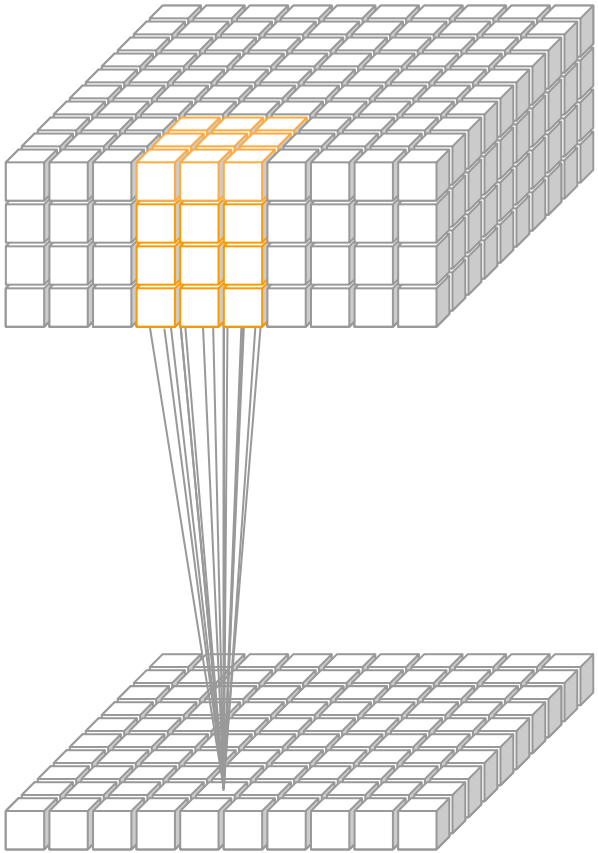
More filters, more output channels.

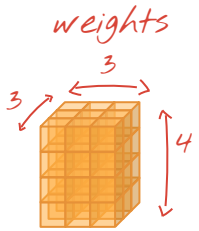
Notes:

- An output channel is also called a “feature map”. It encodes the presence or absence (and degree of presence) of the feature it detects.
- CNNs are somewhat resistant to translation (an image shifted a bit will have a similar activation map, unless of course parts of it fall “off screen”).

How many parameters are in the second layer?

```
model = Sequential()

model.add(Conv2D(filters=4,
                 kernel_size=(4,4),
                 input_shape=(10,10,3))

model.add(Conv2D(filters=8,
                 kernel_size=(3,3))
```

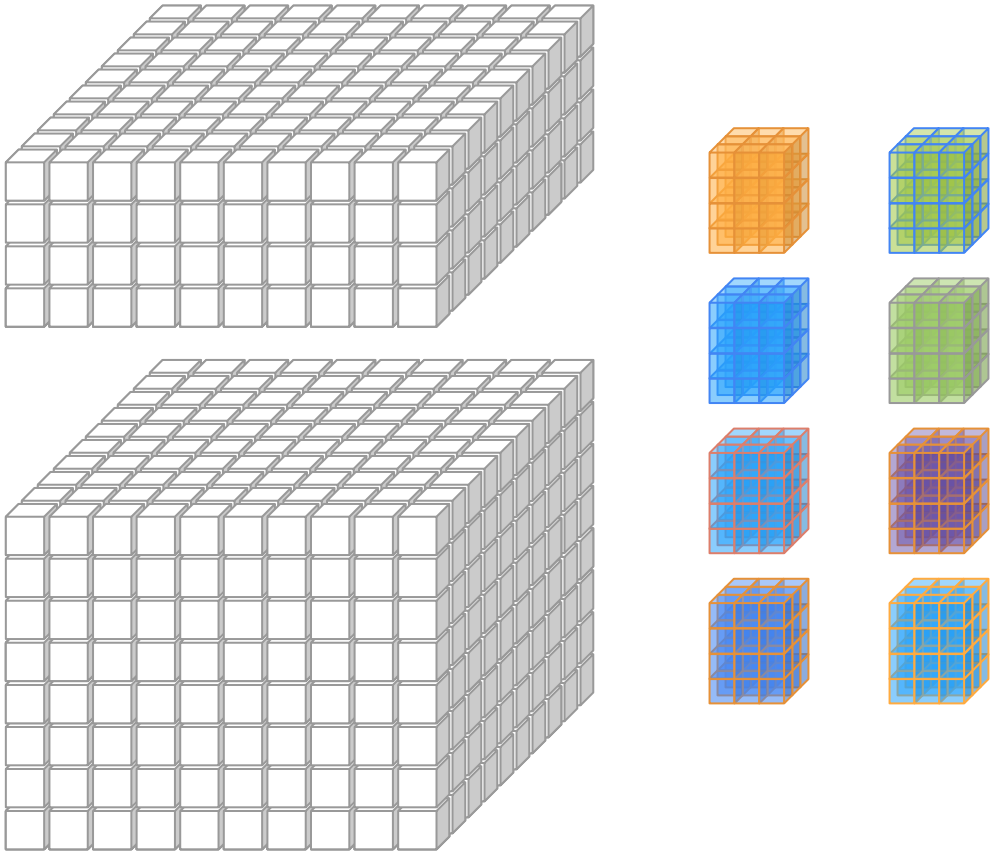




Second layer

Notes:

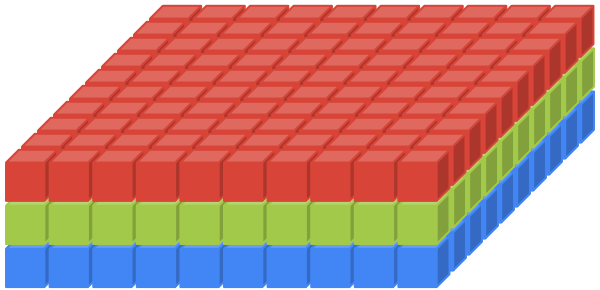
- Filters pass through entire volume of image depthwise.
- Filters would become increasingly difficult to manually design as the layer and depth increase.
- Here, the filter is taking a dot product of the activation maps produced by the previous layer.



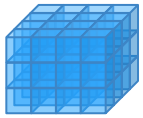
Second layer

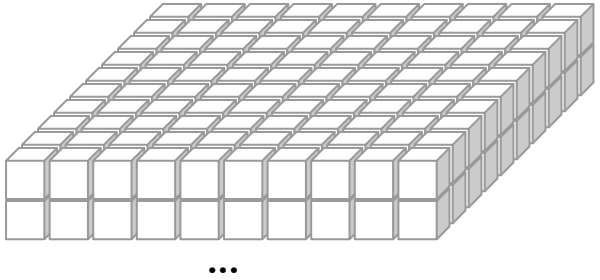
Notes cont'd:

- As before, increasing the number of filters increases the number of activation maps produced.
- Each of these can be visualized as an image.
- Even better: later we'll see how to "find" images that maximally excite these filters (Deep Dream).



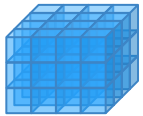
Edges

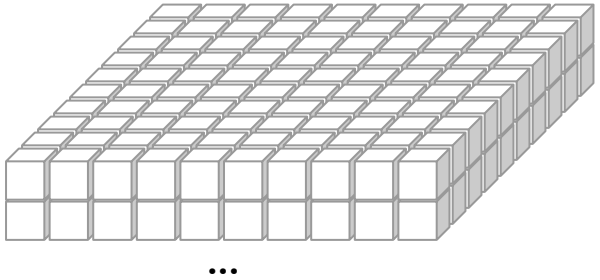




...

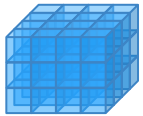
Shapes





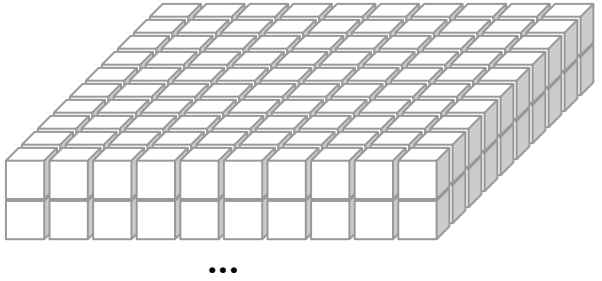
...

Textures



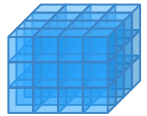
Subsequent layers

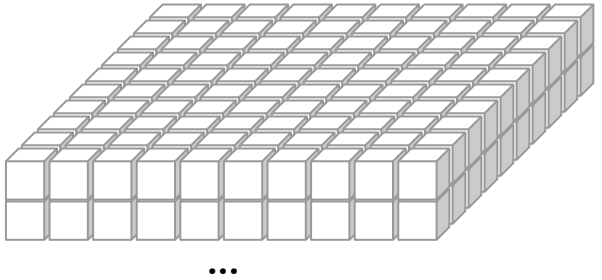
- Hierarchical representation!



...

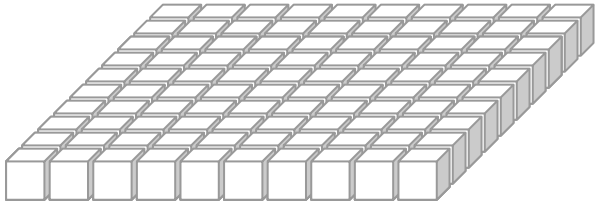
???

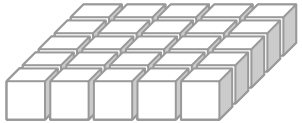




...

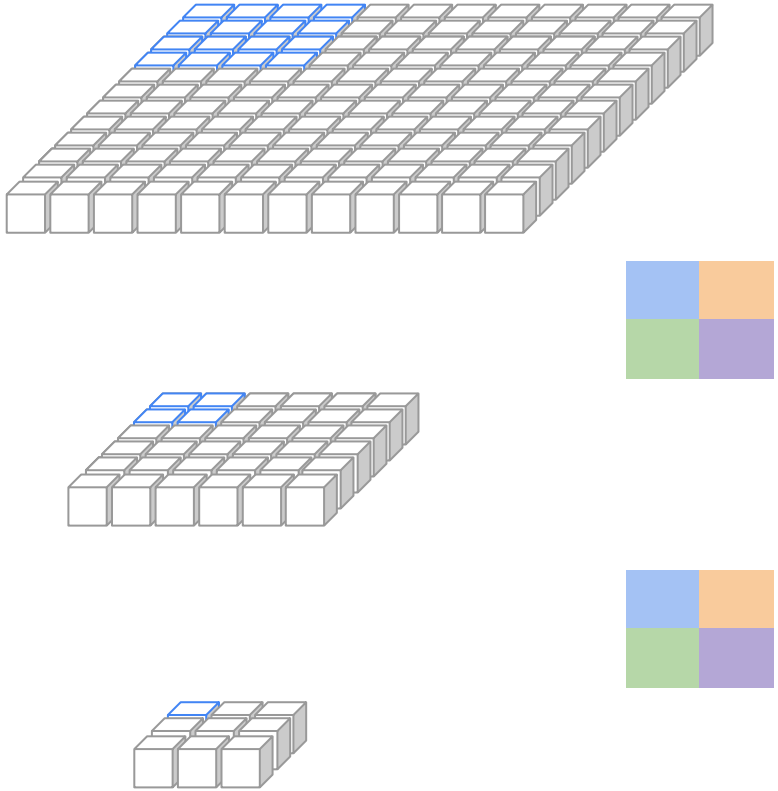
```
model = Sequential()

model.add(Conv2D(filters=4,
                 kernel_size=(3,3),
                 input_shape=(10,10,3))

model.add(Conv2D(filters=8,
                 kernel_size=(3,3))

model.add(MaxPooling2D(pool_size=(2, 2)))
```





2	0	1	1
0	1	0	0
0	0	1	0
0	3	0	0

2	1
3	1



Notes

- Pooling is lossy (intuition: retain only the most prominent features)
- Reducing image size means we can afford more filters at subsequent layers.
- Receptive field. Imagine sliding a filter over the lowest layer. Each activation there represents a region several times larger on the input image.

```
model = Sequential()

model.add(Conv2D(filters=4,
                 kernel_size=(3,3),
                 input_shape=(10,10,3))

model.add(Conv2D(filters=8,
                 kernel_size=(3,3))

model.add(MaxPooling2D(pool_size=(2, 2)))


model.add(Flatten())

model.add(Dense(10)
```

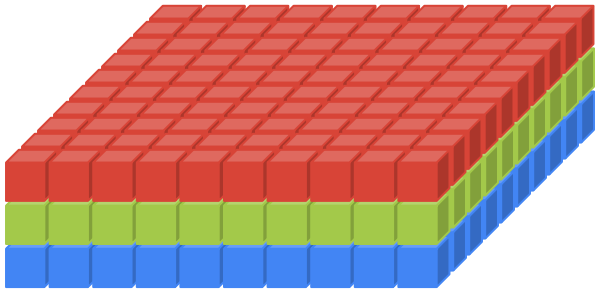
Convolutional base produces a feature vector (or an embedding).

```
model = Sequential()

model.add(Conv2D(filters=4,
                 kernel_size=(3,3),
                 input_shape=(10,10,3))

model.add(Conv2D(filters=8,
                 kernel_size=(3,3))

model.add(MaxPooling2D(pool_size=(2, 2)))
```

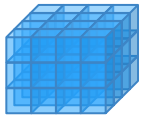
Convolutional base produces a feature vector (or an embedding).

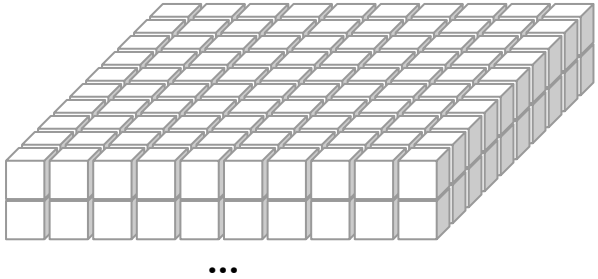
```
model.add(Flatten())

model.add(Dense(10)
```

Top (or head) classifies the image.

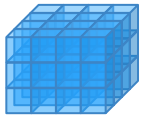


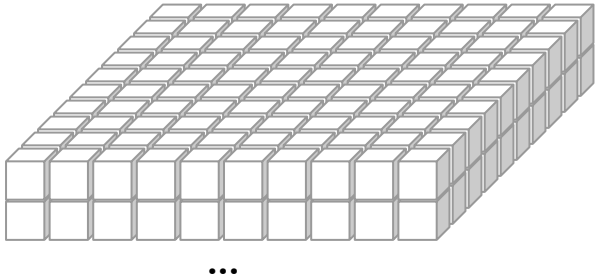
Edges



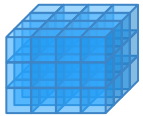


Shapes

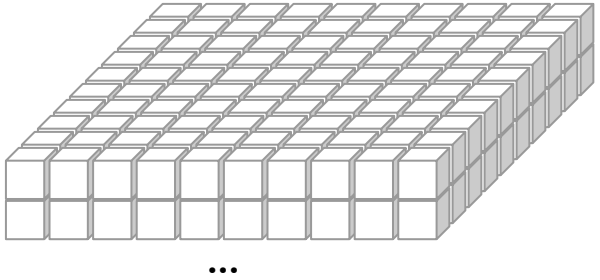


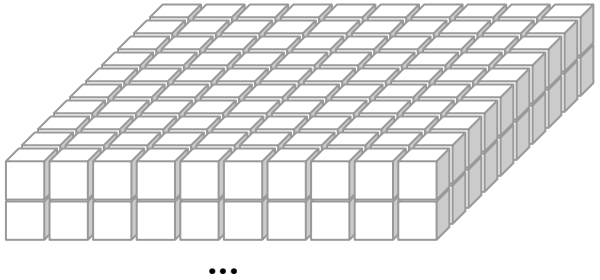


Textures

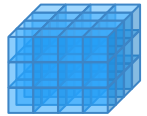


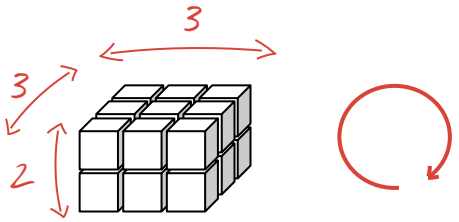
Deep learning = information distilling pipeline.





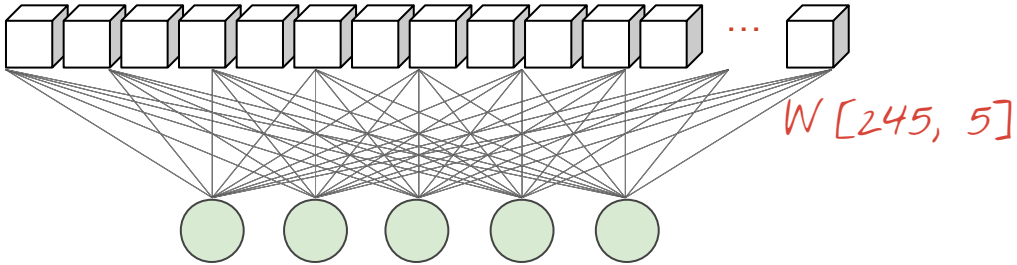
???





flatten

245



1	2	3
4	5	6
7	8	9

1	2	3	4	5	6	7	8	9

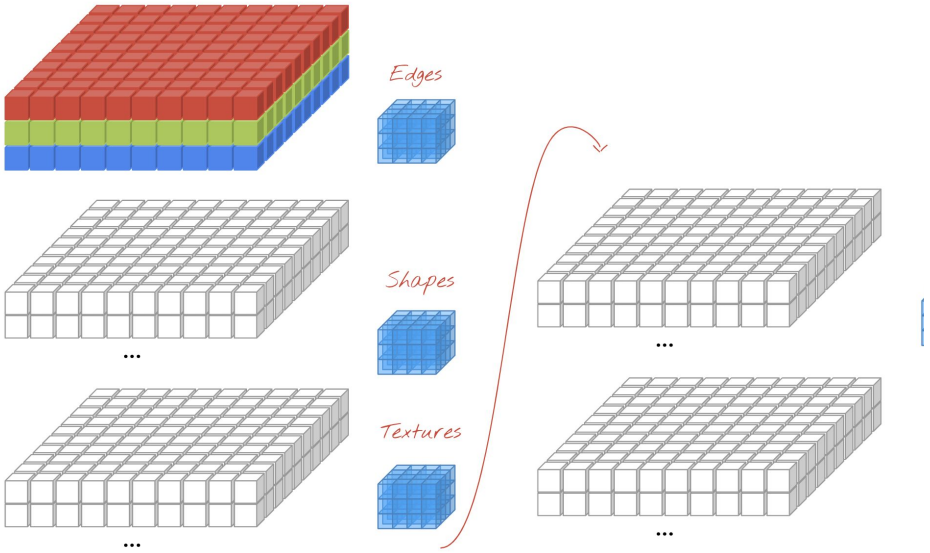
1	2	3
4	5	6
7	8	9

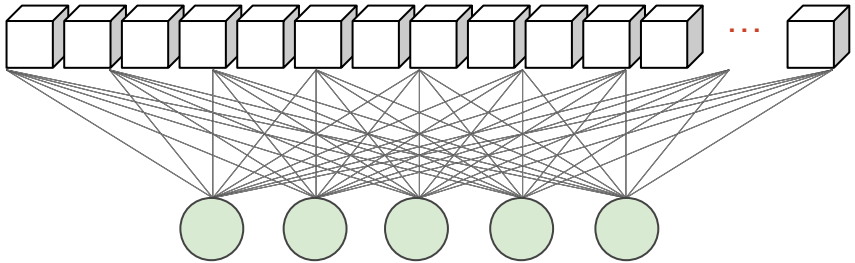
10	11	12
13	14	15
16	17	18

1	2	3	4	5	6	7	8	9	10	11	12	...

Notes

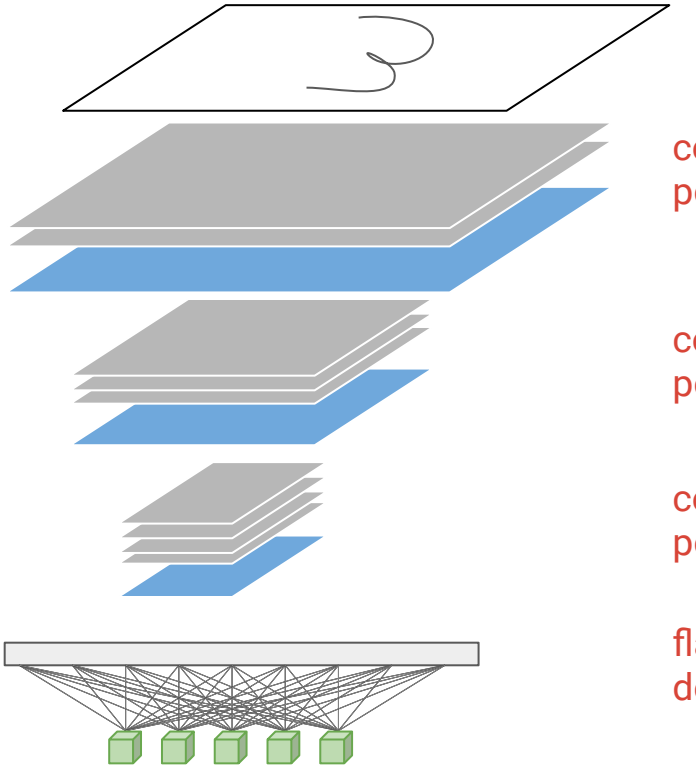
- Multiple channels can be flattened in the same way.


Edges
Shapes
Textures
...
...
...
...


...

Common setup

- One or more stacks of conv / pool layers with relu activation, followed by a flatten then one or two dense layers.
- As we move through the network, feature maps become **smaller spatially**, and **increase in depth**.
- Features become increasingly abstract (but lose spatial information). E.g., “image contains a eye, but not sure exactly it was”).



Input image (here, one color channel)

convolutional
pooling

Each filter in a convolutional layer products a feature (or activation map)

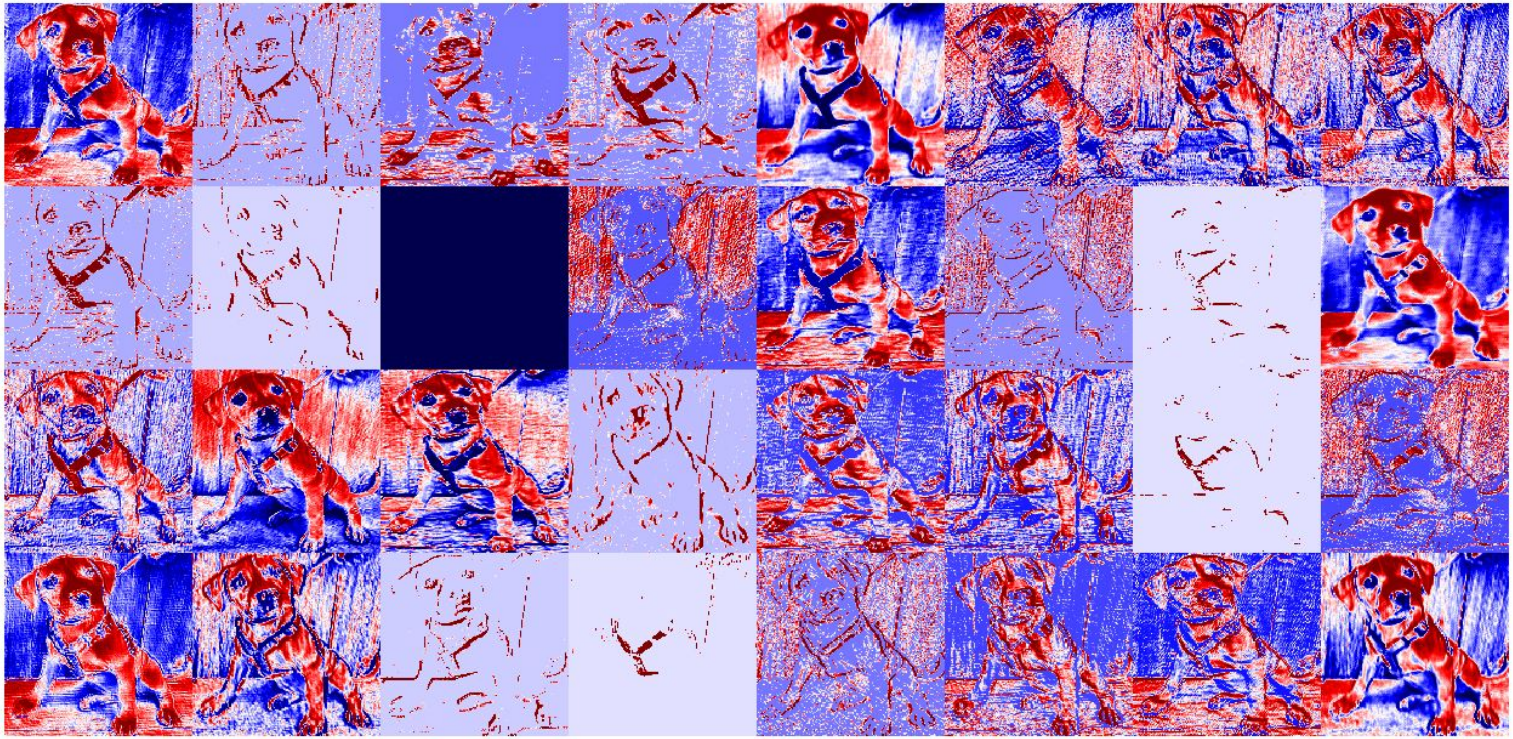
convolutional
pooling

convolutional
pooling

flatten
dense

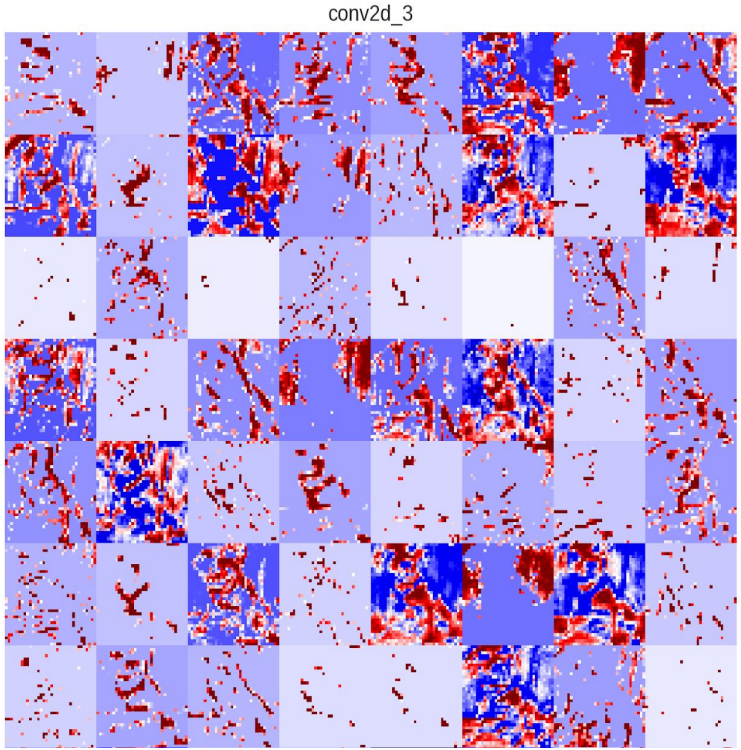
Visualizing activations

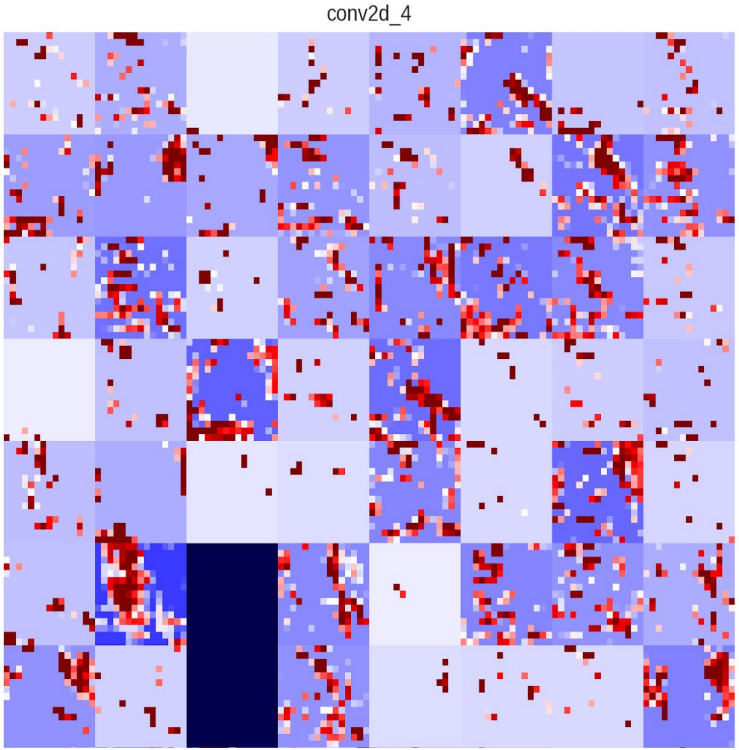
conv2d_1



Forward an image through the network, capture the output of each convolutional filter.

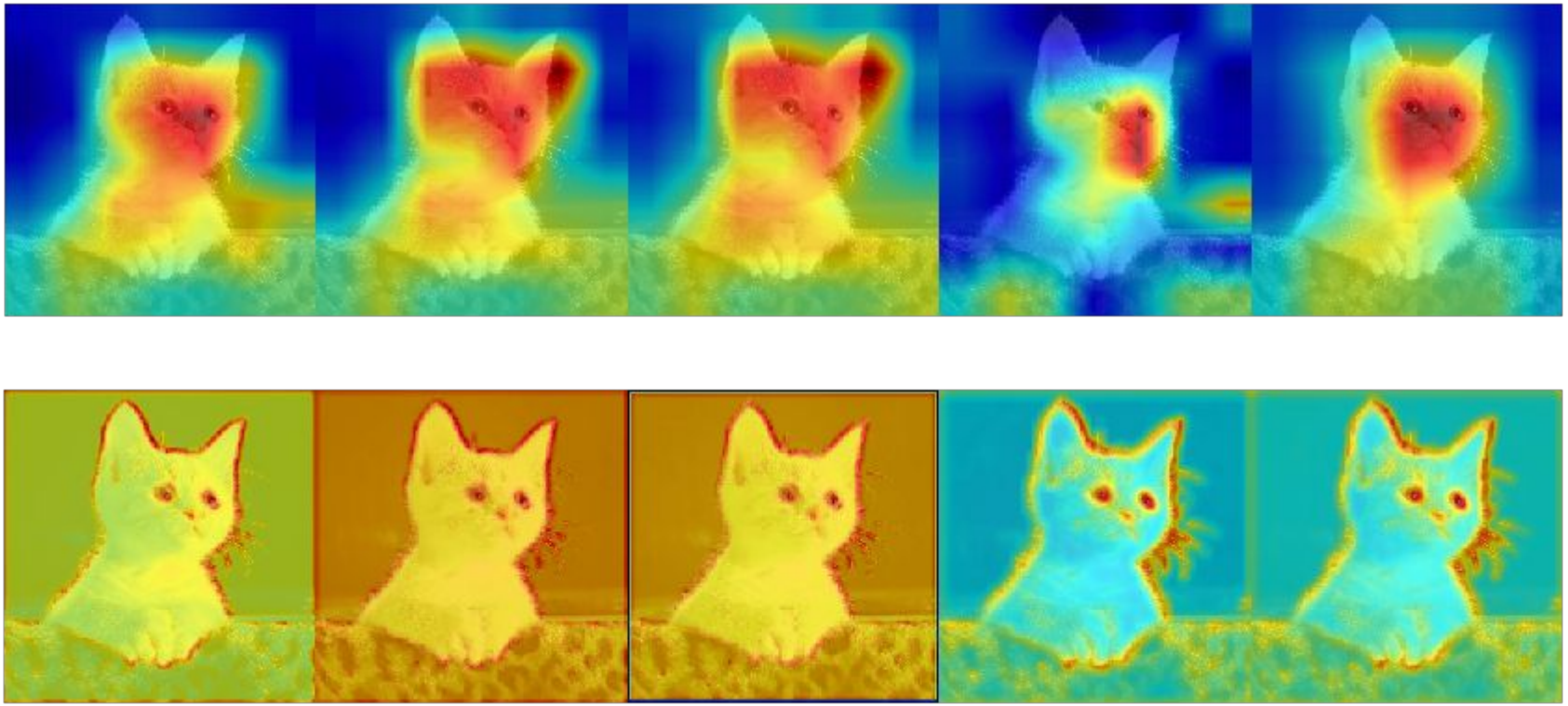
https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/5.4-visualizing-what-convnets-learn.ipynb





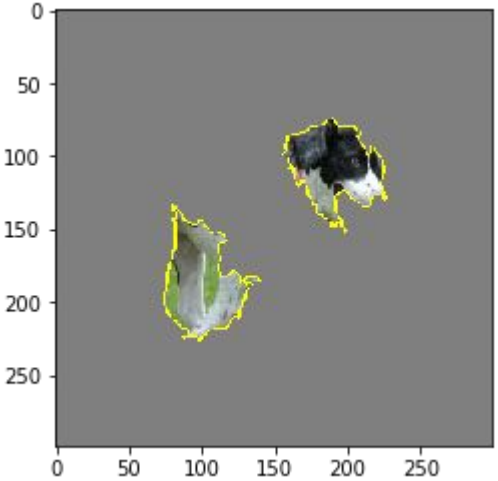
Resolution decreases with depth. Activations tell us more about “what” was in the image, rather than “where”.

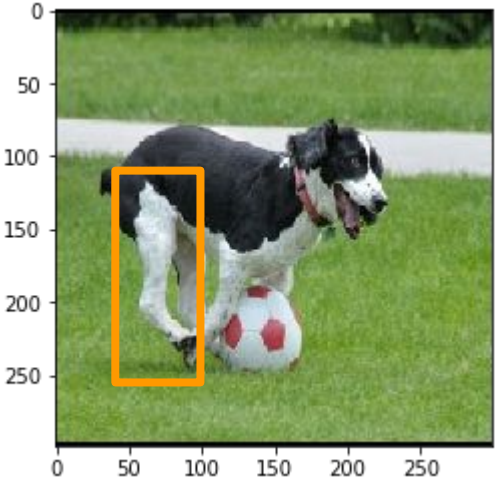
https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/5.4-visualizing-what-convnets-learn.ipynb

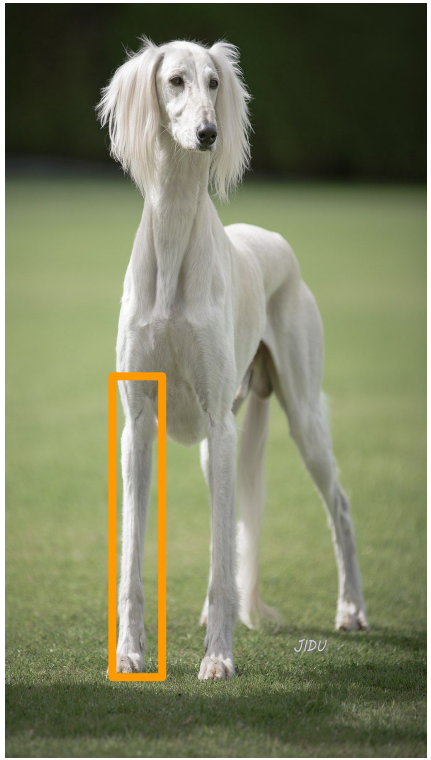


https://colab.research.google.com/github/zaidalyafeai/AttentioNN/blob/master/Attention_Maps.ipynb
Quick walkthrough - this is lightweight and works relatively well

LIME, FYI

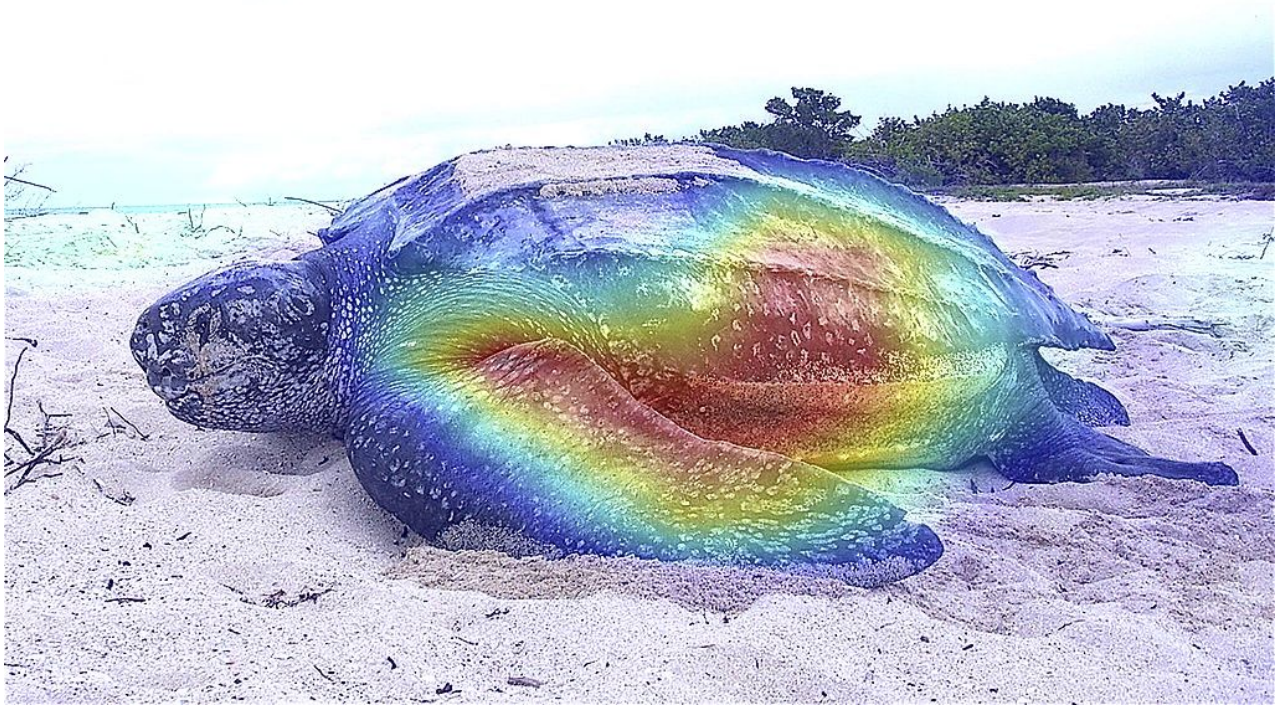






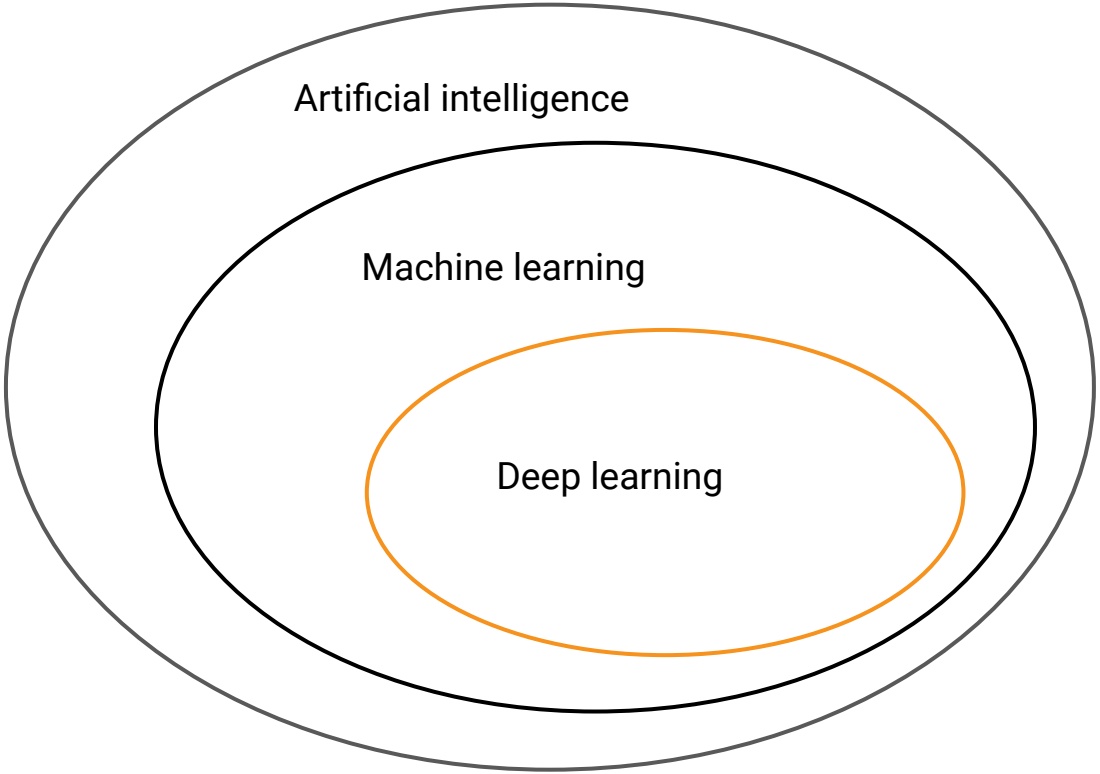
[“Why Should I Trust You?” Explaining the Predictions of Any Classifier Quick walkthrough]

Gradient-based approaches also possible



Leatherback turtle

Representation learning



CNNs are a great way to see the representations learned.

Deep Dream and Style Transfer

Just a few quick notes.

Idea in a nutshell

We know: early layers capture edges / shapes / textures (**style)**. Later layers capture eyes / buildings / etc (**content**).

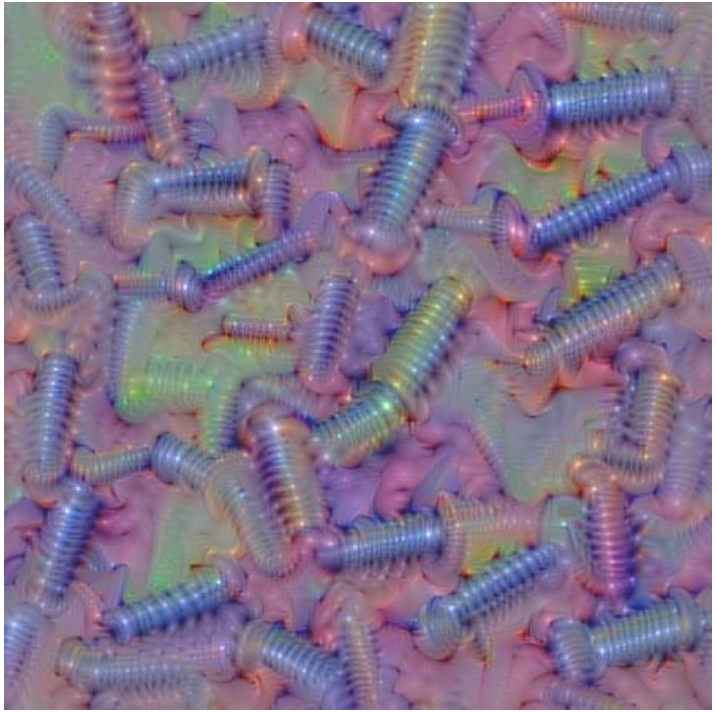
Start with three images.

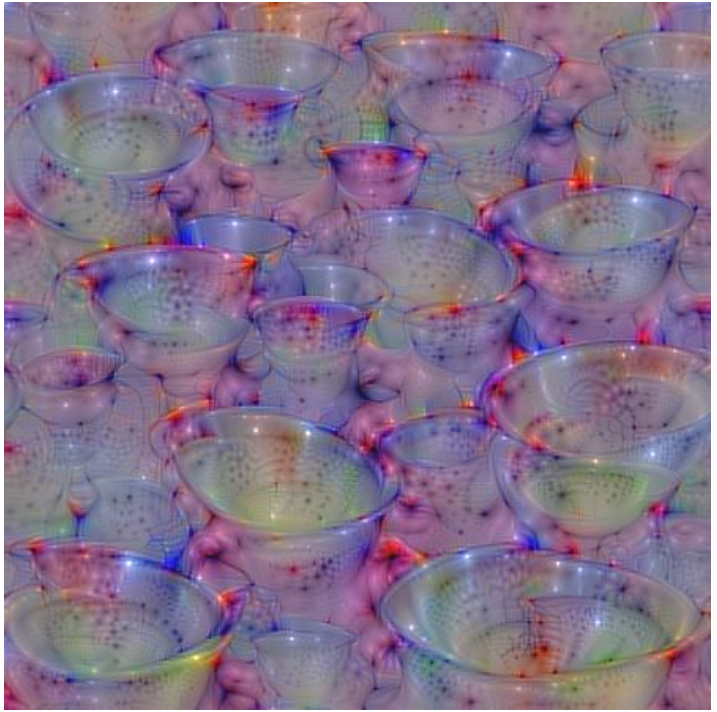
- Random noise
- A content image (picture of Chicago)
- A style image (famous painting)

Create a loss function that nudges the activations of the noise image to activate a pretrained CNN similarly to the style image (for early layers) and similarly to the content image (for later layers). Optimize with gradient ascent.

https://www.tensorflow.org/beta/tutorials/generative/style_transfer
https://github.com/lengstrom/fast-style-transfer

https://www.tensorflow.org/beta/tutorials/generative/deepdream
https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html



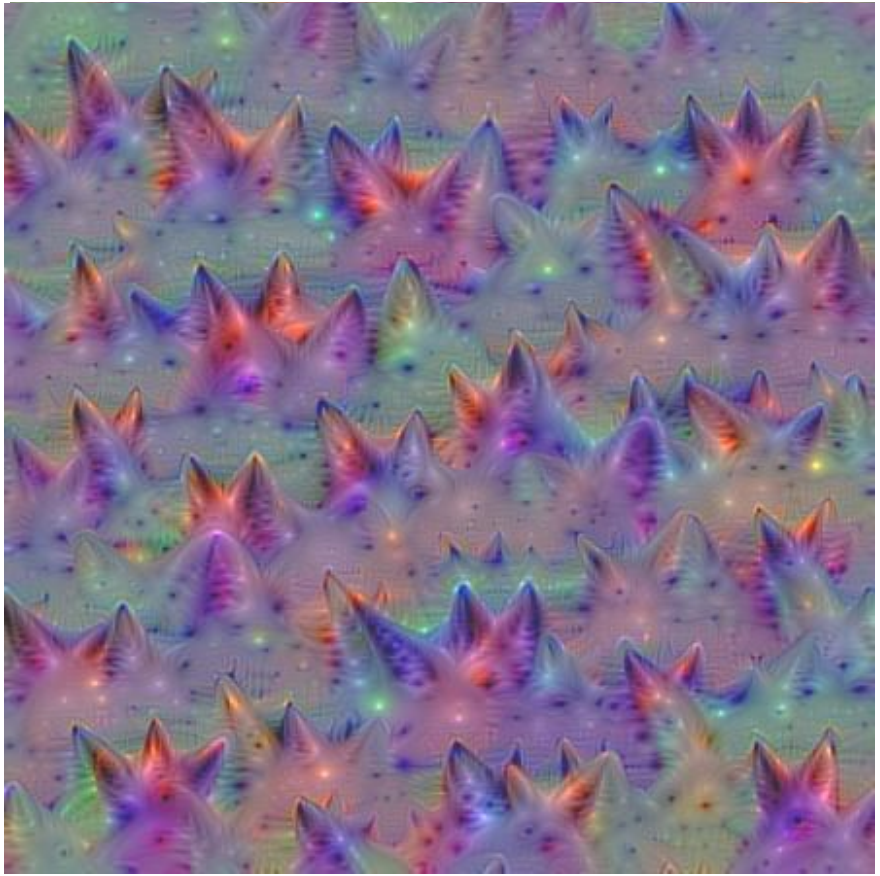


Idea: Create a loss function that's the mean activation of a feature map! Starting from a random noise image, calculate gradients of loss w.r.t pixels, optimize per usual.

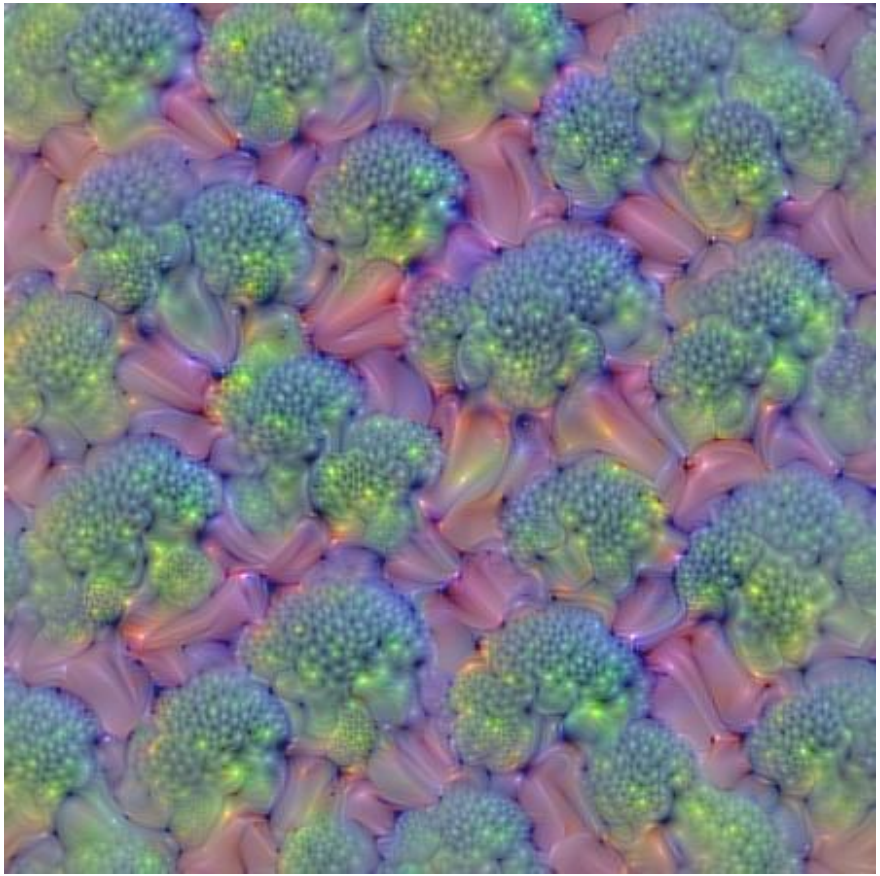
All filters

Screws / Corkscrews?

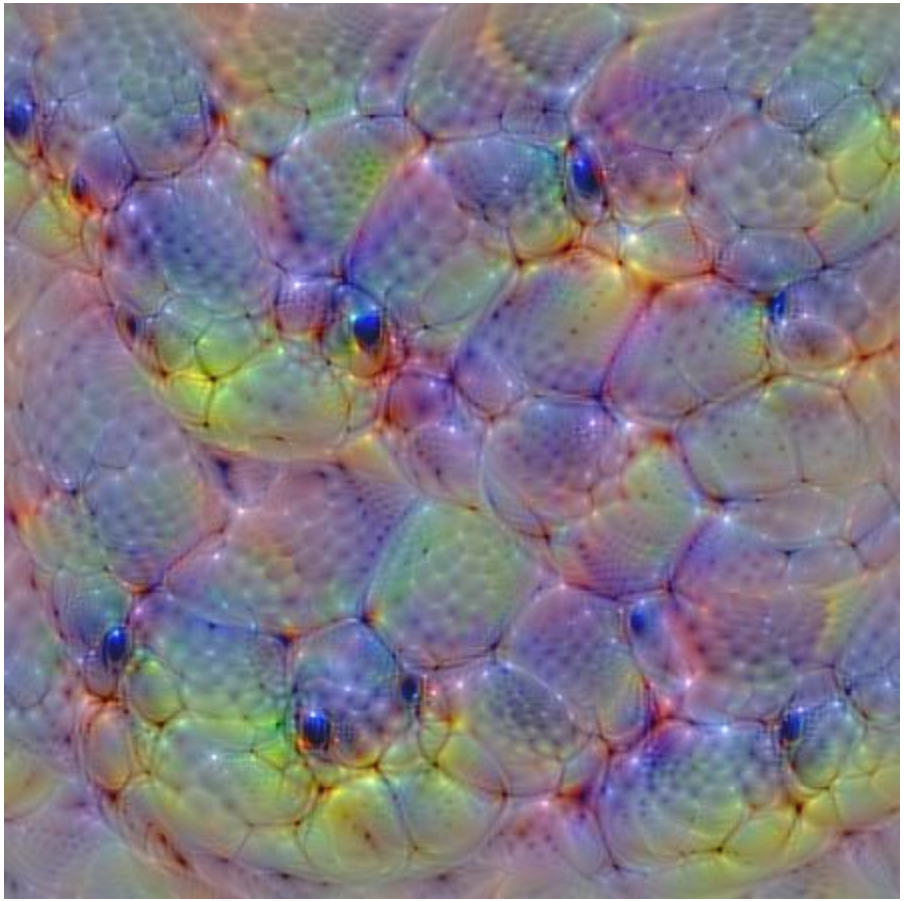
Wine glass?



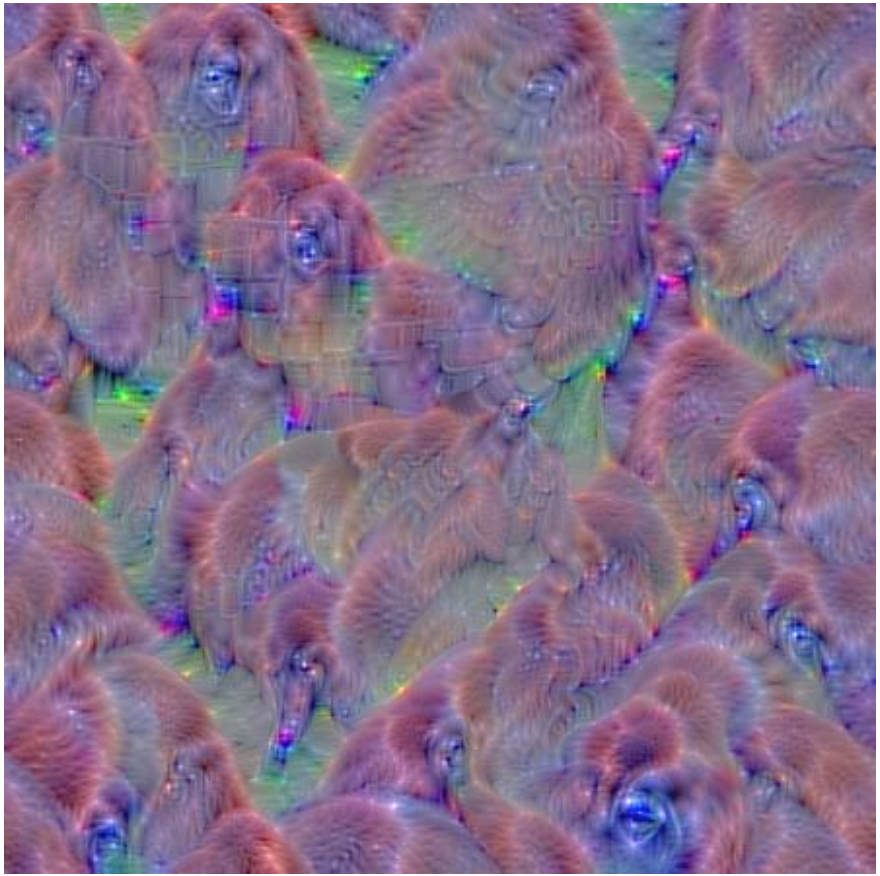
Ears?



Broccoli / Cauliflower?



Snakes?



Gorillas / chimpanzees?

Quick walkthrough
(Deep Dream / Style Transfer)

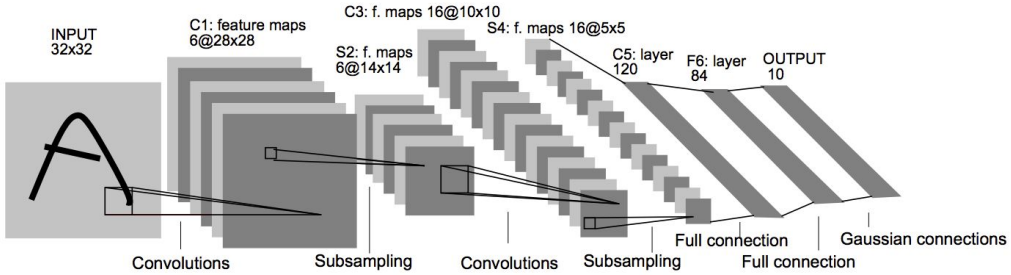
Break: work on the homework, or read notebooks 5.1 and 5.2

https://github.com/fchollet/deep-learning-with-python-notebooks

Various famous architectures

LeNet-5 (1998!)

This pattern (convolution, pool, convolution, pool, etc) became standard.



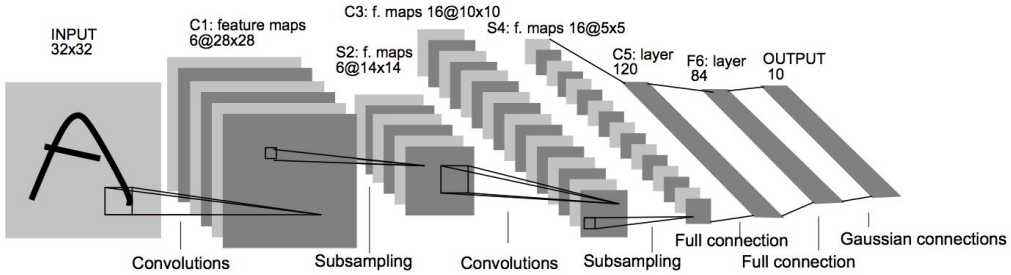
Introduced MNIST

Quick discussion: Disadvantages of using dense layers to recognize images?

http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

LeNet-5 (1998!)

This pattern (convolution, pool, convolution, pool, etc) became standard.



Introduced MNIST

Quick discussion: Why not just use Dense layers to recognize images?

1. **Efficiency**: Assume input image is 150x150x3 = 67500 pixels. Say 1K neurons in first layer = 67.5M parameters (just in layer 1!)

2. **Features must be detected separately at all locations:** Say detecting an edge is important. A Dense layer must learn to detect that edge at all positions (separately).

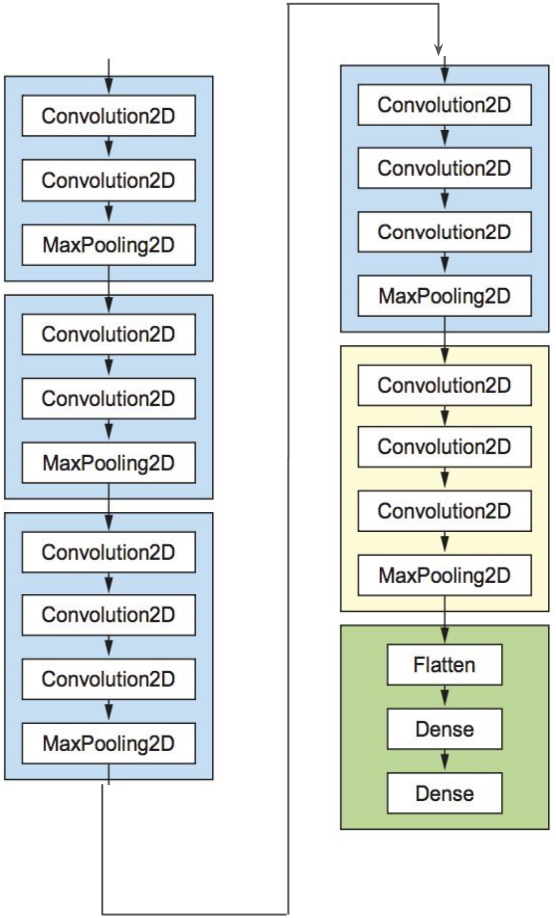
http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

A typical CNN

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=(28, 28, 1),
                 padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

rows, cols, color channels

VGG (2014)



[Very Deep Convolutional Networks for Large-Scale Image Recognition]

Importing

```
from keras.applications.vgg19 import VGG19
model = VGG19(weights='imagenet', include_top=True)
model.summary()
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv4 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv4 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv4 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000

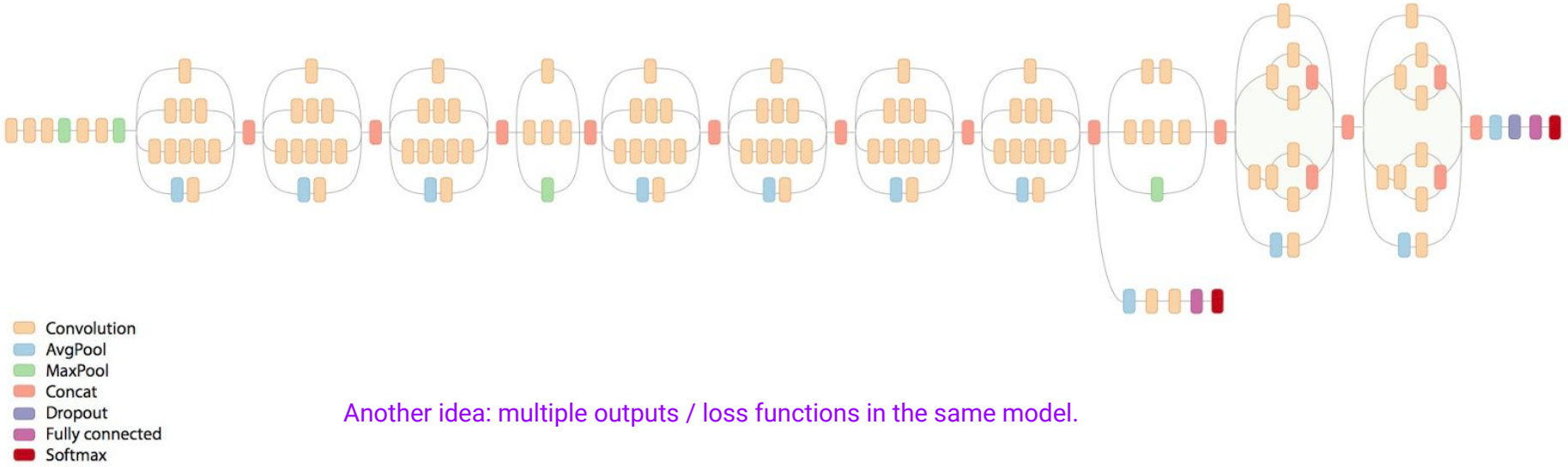
Total params: 143,667,240
Trainable params: 143,667,240
Non-trainable params: 0

Notice **~80%** of the parameters are in the last couple Dense connected layers...

Inception (2014)

- Emphasized efficiency (**10x** fewer parameters than AlexNet).
- A few layers you may not have seen yet (Global Average Pooling; 1d convolutions)
- Connections between layers (cannot implement with Sequential, you need Functional or Subclassing)

GoogLeNet



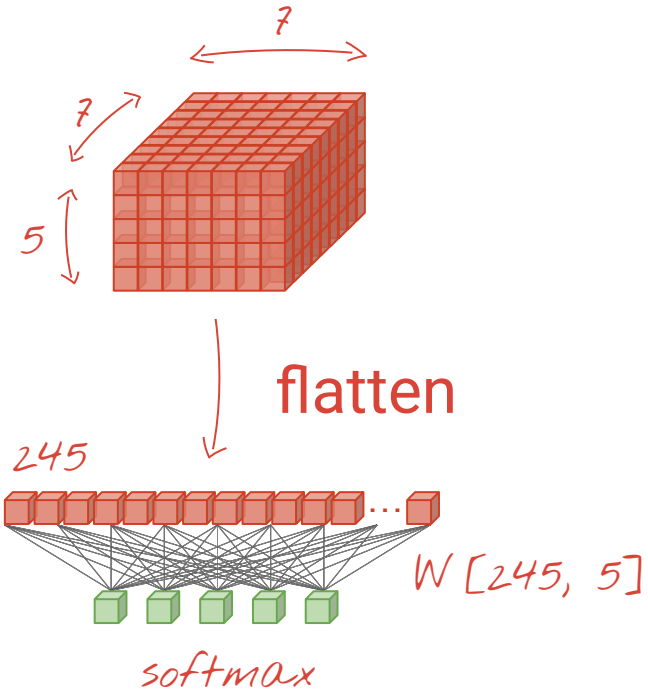
Another idea: multiple outputs / loss functions in the same model.

https://ai.googleblog.com/2016/08/improving-inception-and-image.html

```
from keras.applications.inception_v3 import InceptionV3
model = InceptionV3(weights='imagenet', include_top=True)
model.summary()
```

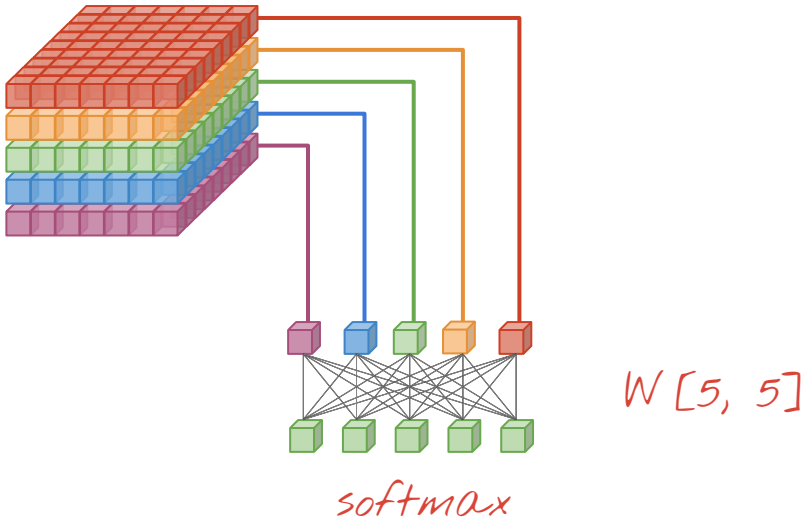
```
mixed10 (Concatenate)           (None, None, None, 2 0           activation_86[0][0]
                                                                 mixed9_1[0][0]
                                                                 concatenate_2[0][0]
                                                                 activation_94[0][0]
avg_pool (GlobalAveragePooling2 (None, 2048)         0           mixed10[0][0]
predictions (Dense)             (None, 1000)         2049000     avg_pool[0][0]
==================================================================================================
Total params: 23,851,784
Trainable params: 23,817,352
Non-trainable params: 34,432
```

Global average pooling

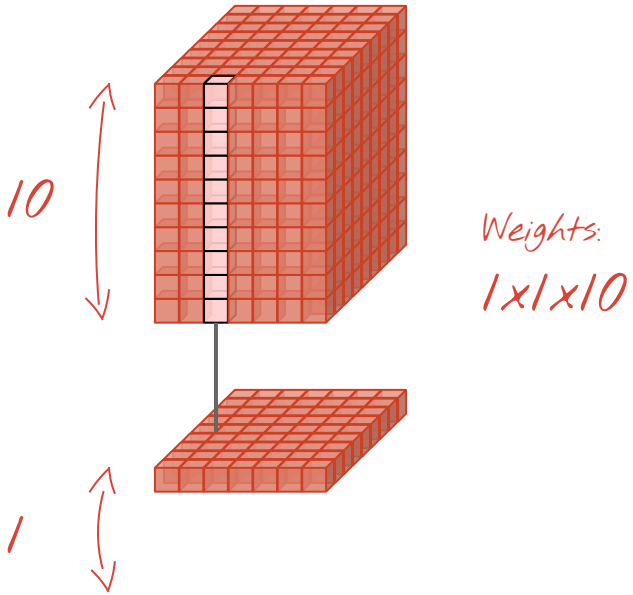


Notes: Although the last dense layers in CNNs contain most of the weights, they were found to be less helpful than thought in the Inception paper. Global average pooling prior to the dense layers greatly reduces the number of weights, without changing accuracy much.

Global average pooling



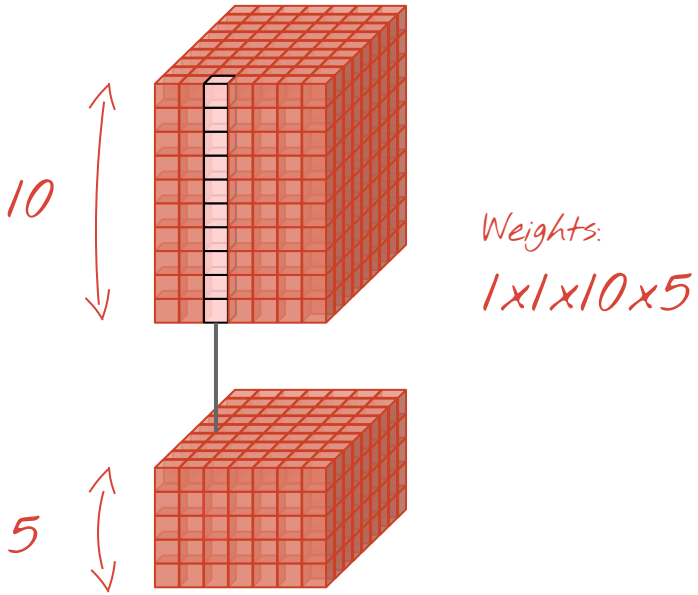
1d convolution



Why 1x1 convolutions?

- **Efficiency**: reduces the depth (number of channels). Width and height are unchanged. To reduce the horizontal dimensions, you would use pooling (or increase the stride of the conv).
- The 1x1 conv computes a weighted sum of input channels (or features). This allows it to "**select**" certain combinations of features that are useful downstream.

1d convolution



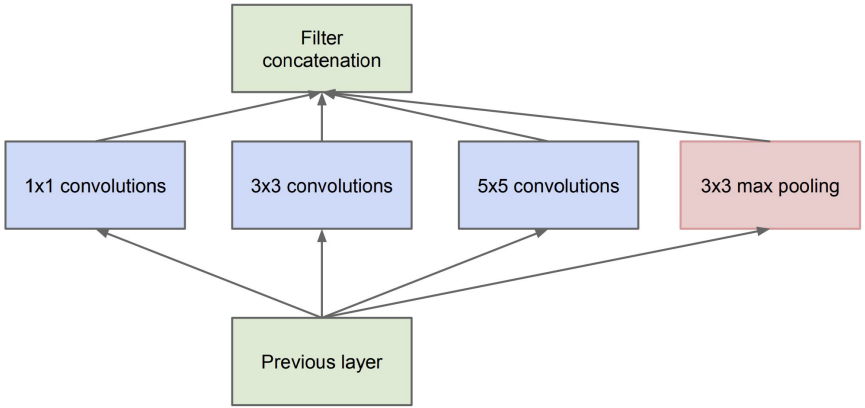
Why 1x1 convolutions?

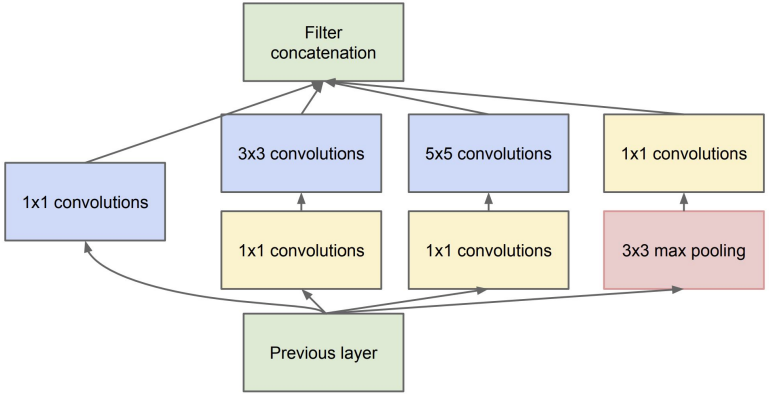
- **Efficiency**: reduces the depth (number of channels). Width and height are unchanged. To reduce the horizontal dimensions, you would use pooling (or increase the stride of the conv).
- The 1x1 conv computes a weighted sum of input channels (or features). This allows it to "**select**" certain combinations of features that are useful downstream.

Of course, you can use a bank of these as well.

Inception module

Basic idea: instead of choosing which filter size to use (3x3 or 5x5?), use a bunch in parallel, let the network sort out which are useful. Results merged by stacking depthwise.



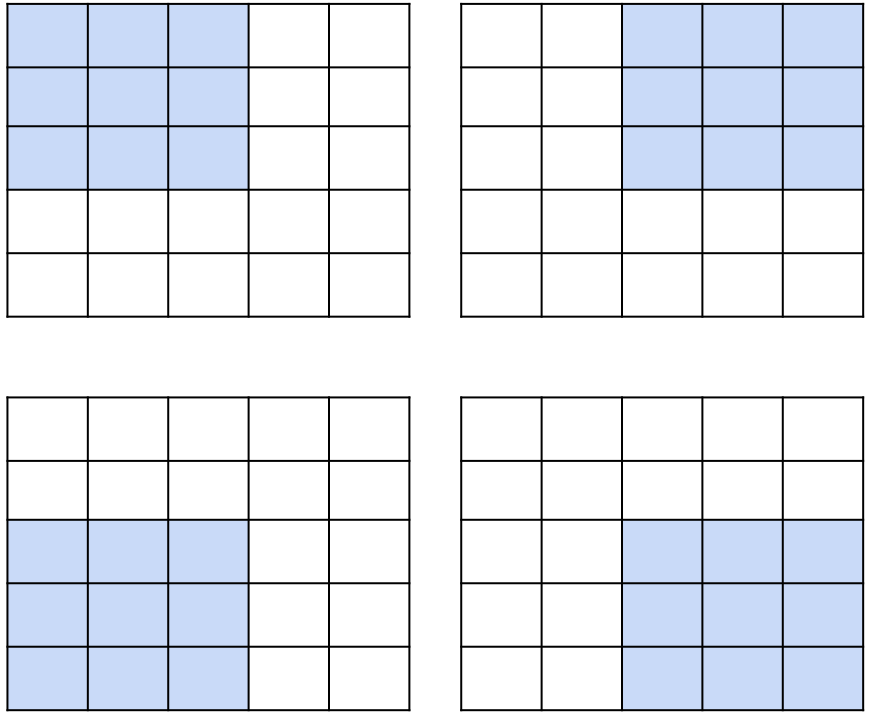


[GoogLeNet]

Left image: first idea (problem: output size too large). Right image: using 1d convs to reduce size.

Stride

- Stride (step size as the filter slides across the image).
- Using numbers >1 will downsize the image.

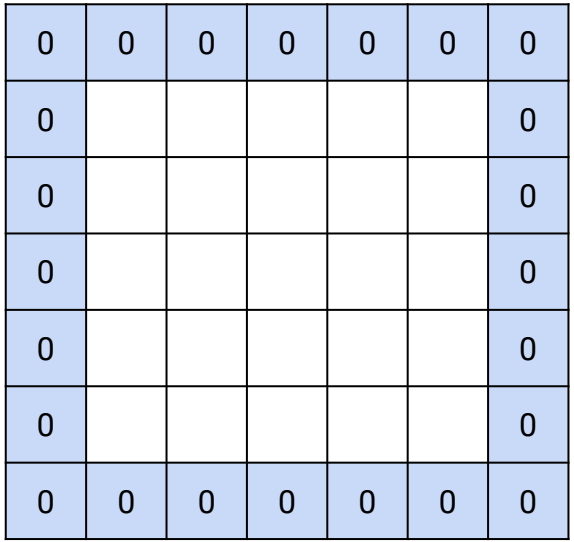


A 3x3 filter sliding over a 5x5 input with stride 2. Here, this results in a 2x2 output.

Padding

- **Valid** (no padding is applied; only valid locations will be used to compute the output of the conv)

- **Same** (library adds zero padding so the output has the same width and height as the input)

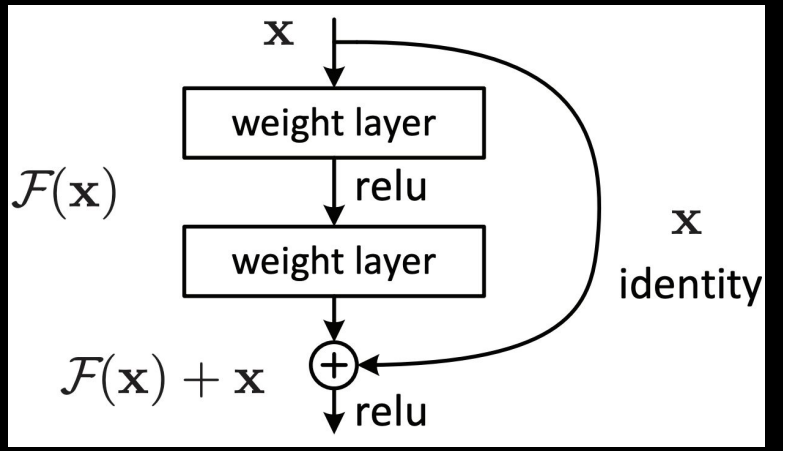


A 3x3 filter sliding over a 5x5 input with “same” padding produces a 5x5 output.

keras.io/layers/convolutional/#conv2d

```python
from tensorflow.keras.layers import Conv2D
from tensorflow.nn import relu

class ResExample(tf.keras.Model):
  def __init__(self):
    super(ResExample, self).__init__()
    self.c1 = Conv2D(32, (3, 3), padding='same')
    self.c2 = Conv2D(3, (3, 3), padding='same')

  def call(self, input_tensor):
    x = relu(self.c1(input_tensor))
    x = relu(self.c2(x))
    x += input_tensor
    return relu(x)

data = tf.random.normal((1,128,128,3))
block = ResExample()
print(block(data).shape)
```

Pratik is working on a notebook for you.



Deep Residual Learning for Image Recognition
https://www.tensorflow.org/beta/guide/keras/custom_layers_and_models

How many weights are in a dense layer?

flatten_1 (Flatten)	(None, 3136)	0
dense_1 (Dense)	(None, 128)	401536
dense_2 (Dense)	(None, 10)	1290

How many weights are in a dense layer?

flatten_1 (Flatten)	(None, 3136)	0
dense_1 (Dense)	(None, 128)	401536
dense_2 (Dense)	(None, 10)	1290

Fully connected, so...

input dimension * output dimension + one bias per output neuron

3136 * 128 + 128 = 401536

How many weights are in a dense layer?

flatten_1 (Flatten)	(None, 3136)	0
dense_1 (Dense)	(None, 128)	401536
dense_2 (Dense)	(None, 10)	1290

How many weights are in a dense layer?

flatten_1 (Flatten)	(None, 3136)	0
dense_1 (Dense)	(None, 128)	401536
dense_2 (Dense)	(None, 10)	1290

Fully connected, so...

input dimension * output dimension + one bias per output neuron

128 * 10 + 10 = 1290

How many weights are in a pooling layer?

conv2d_2 (Conv2D)	(None, 14, 14, 64)	18496
max_pooling2d_2 (MaxPooling2	(None, 7, 7, 64)	0
flatten_1 (Flatten)	(None, 3136)	0

How many weights are in a pooling layer?

conv2d_2 (Conv2D)	(None, 14, 14, 64)	18496
max_pooling2d_2 (MaxPooling2	(None, 7, 7, 64)	0
flatten_1 (Flatten)	(None, 3136)	0

Pooling layers have no parameters.

How many weights are in a conv layer?

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d_1 (MaxPooling2	(None, 14, 14, 32)	0
conv2d_2 (Conv2D)	(None, 14, 14, 64)	18496

Assume:

- Input shape: 28x28x1
- Filter size: 3x3

How many weights are in a conv layer?

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d_1 (MaxPooling2	(None, 14, 14, 32)	0
conv2d_2 (Conv2D)	(None, 14, 14, 64)	18496

Assume:

- Input shape: 28x28x1
- Filter size: 3x3

3*3 (filter size) * 1 (input depth) * 32 (# of filters) + 32 (bias, 1 per filter) = 320

How many weights are in a conv layer?

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d_1 (MaxPooling2	(None, 14, 14, 32)	0
conv2d_2 (Conv2D)	(None, 14, 14, 64)	18496

Assume:

- Filter size is 3x3

How many weights are in a conv layer?

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d_1 (MaxPooling2	(None, 14, 14, 32)	0
conv2d_2 (Conv2D)	(None, 14, 14, 64)	18496

Assume:

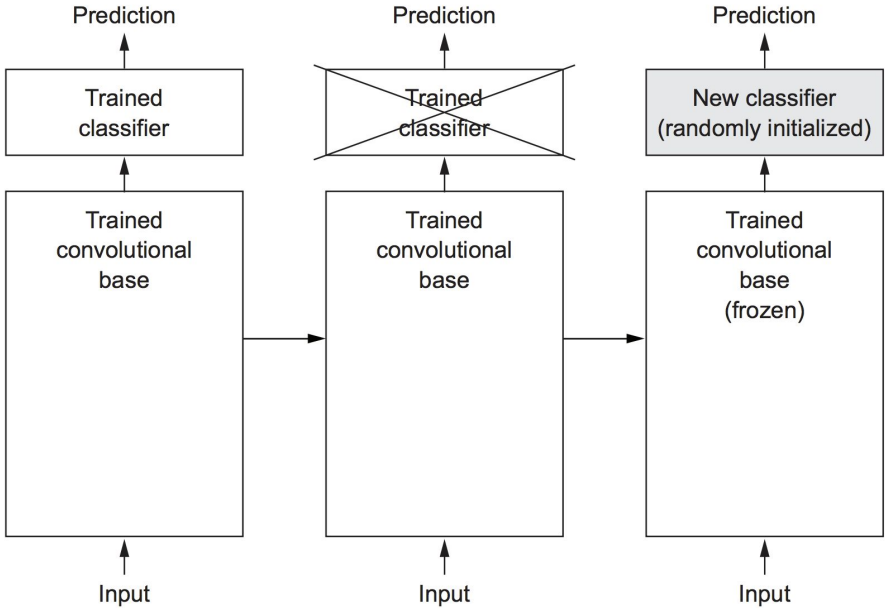
- Filter size is 3x3

3*3 (filter size) * 32 (input depth) * 64 (# of filters) + 64 (bias, 1 per filter) = 18496

Transfer learning

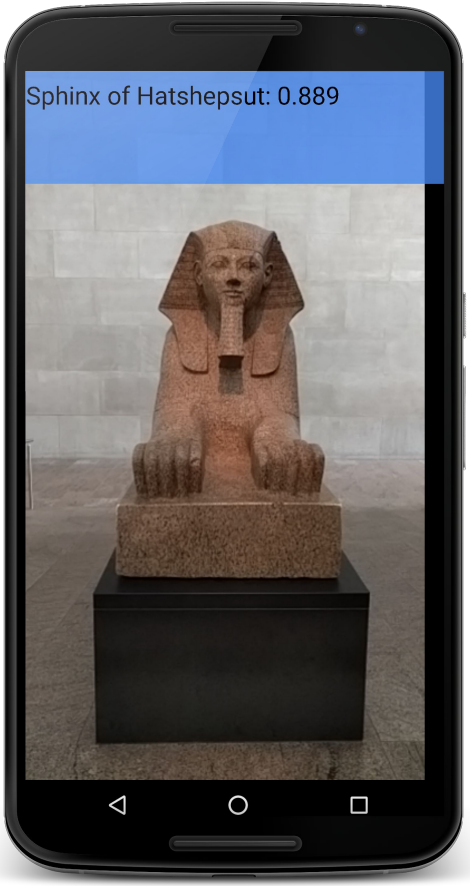
Idea: features learned on one task may be useful on another.

- Recall, the base of a CNN learns a feature hierarchy (edges -> shapes -> textures -> …. -> semantic features (eye detectors, ear detectors, etc).
- Earlier features may generalize to other tasks (especially if trained on a large amount of data, say, ImageNet).



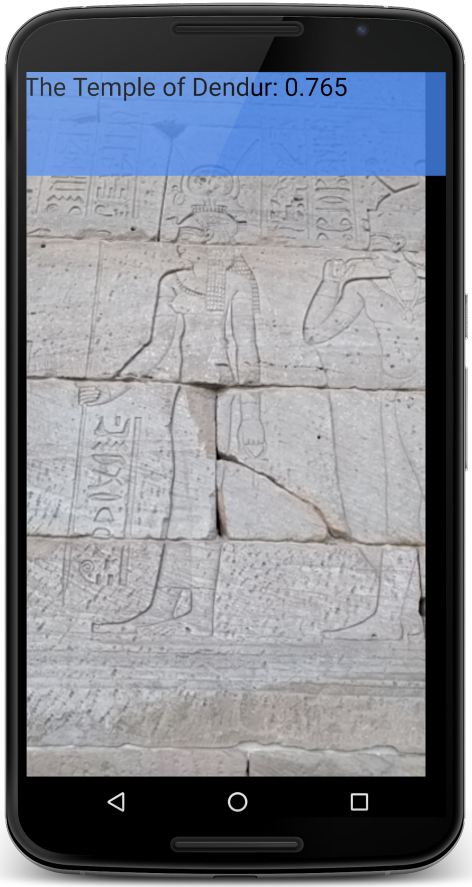
I have stolen this diagram.


Monet - Sunflowers: 0.992



Sphinx of Hatshepsut

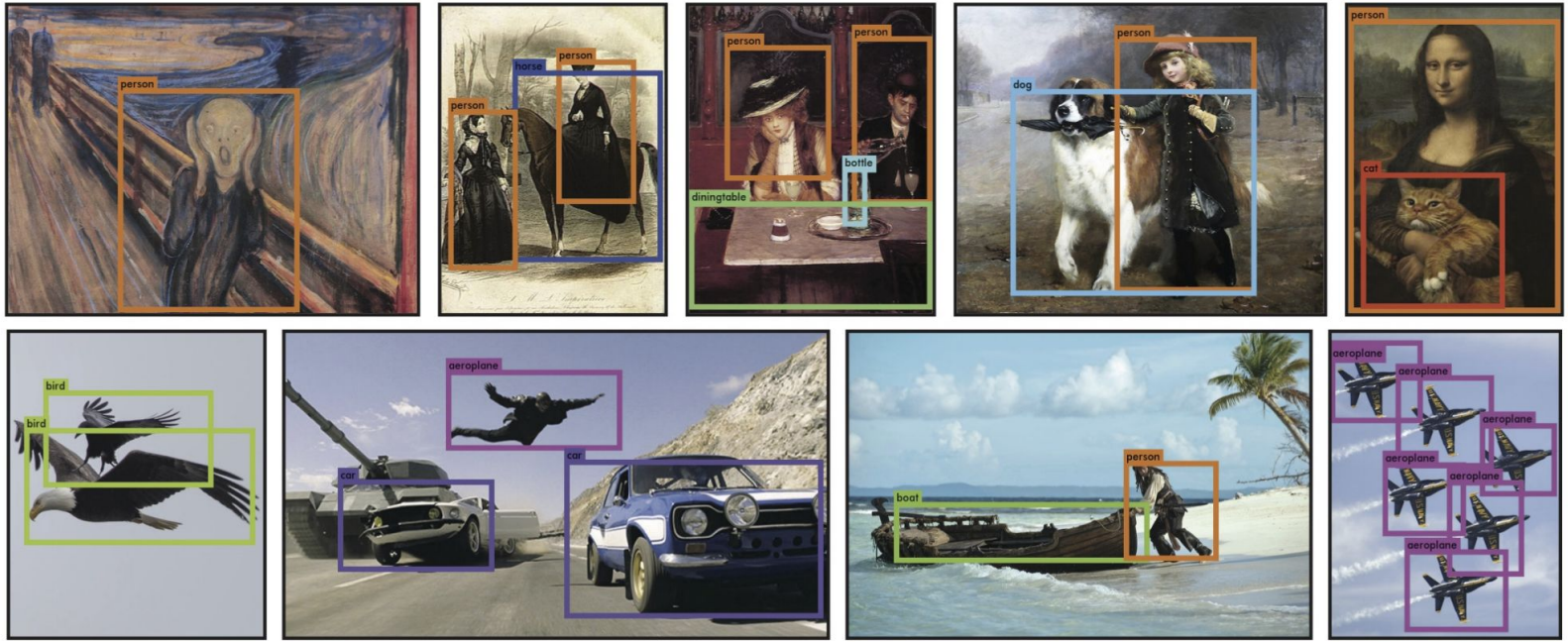
Bernini - Autumn in the Guise of Priapus



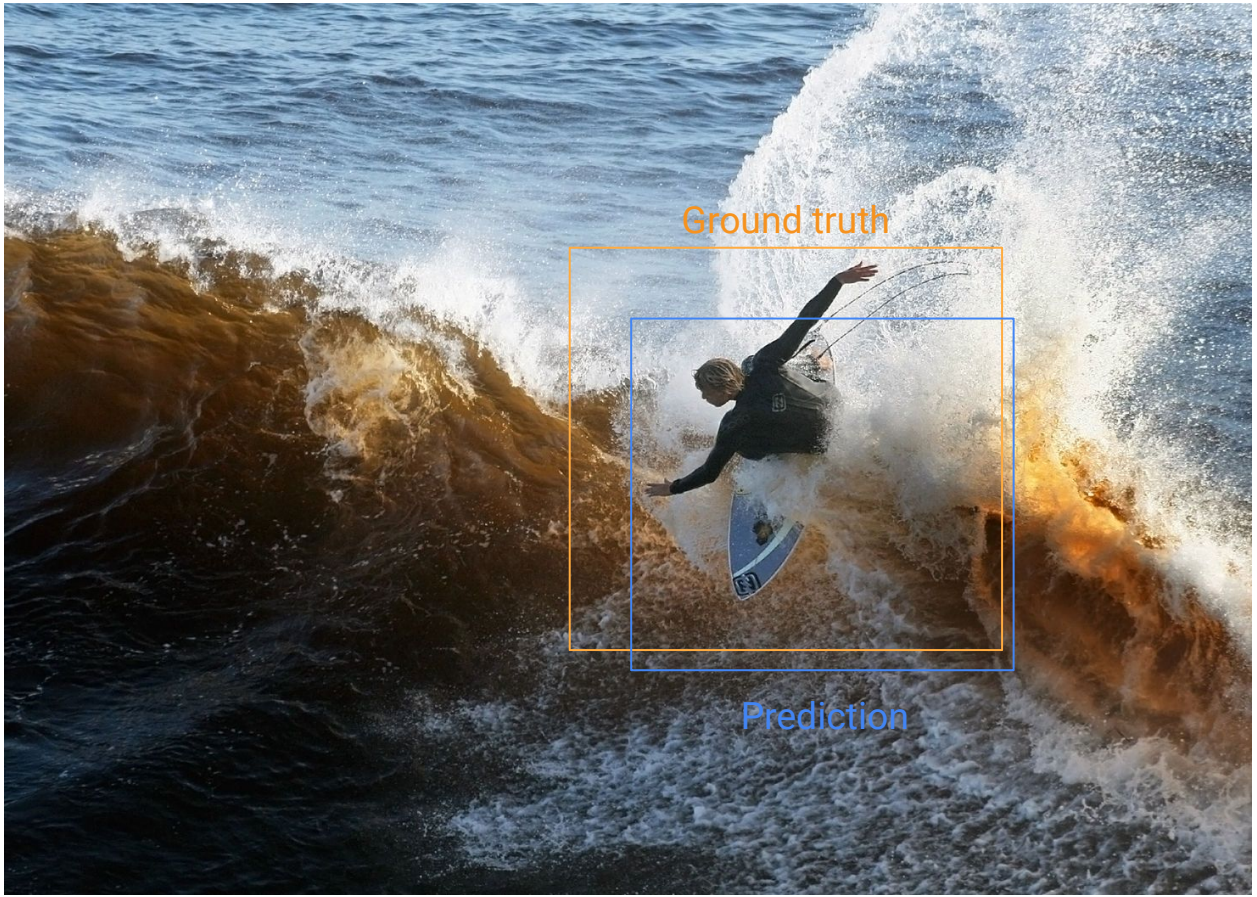
The Temple of Dendur

Beyond image classification

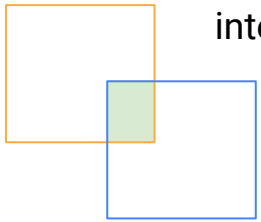
Object detection

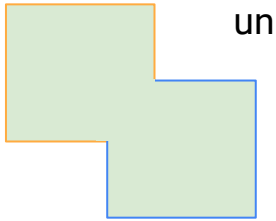


You only look once, 2015



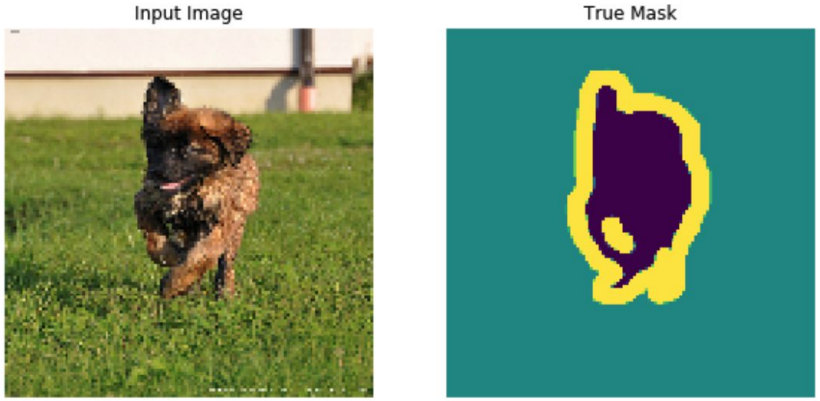
IOU (Intersection over union)

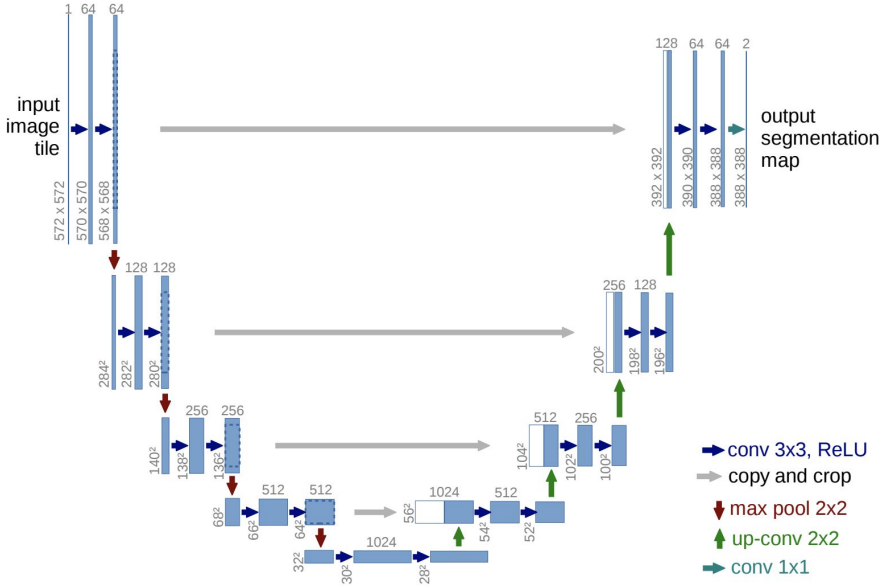




A surfer at the wave

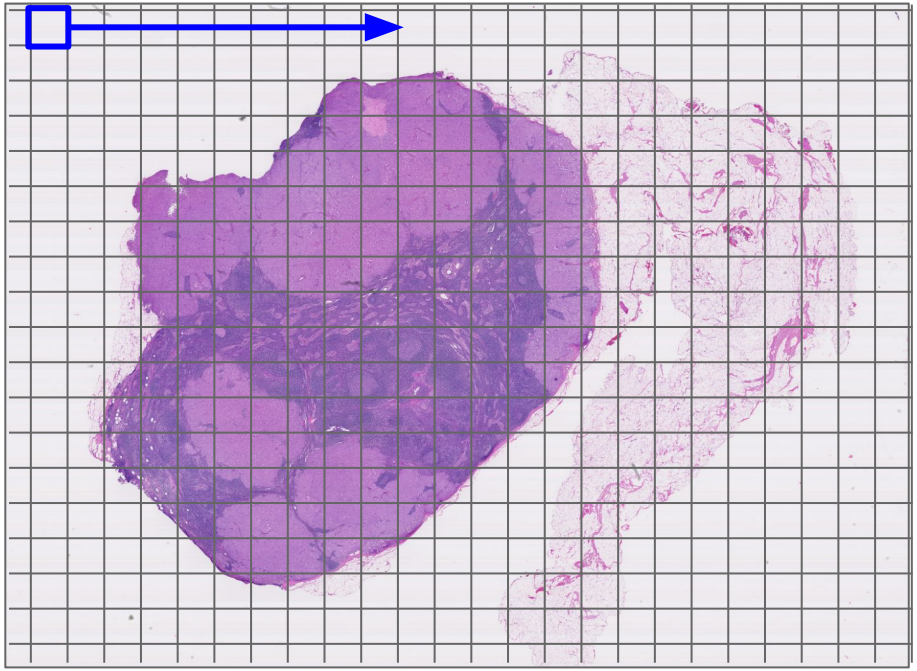
UNet (2015)





https://www.tensorflow.org/beta/tutorials/images/segmentation
U-Net: Convolutional Networks for Biomedical Image Segmentation

For large images, you can mimic segmentation with classification



Approach: Divide images into small patches; Train a model on to classify patches as malignant / benign. Slide window over new image, classifying each patch, produce heatmap.

Pros

- Works with images of any size; potentially more accurate. Simpler, easier to debug and **explain**.

Cons

- YOLO can “look” at the entire image all at once. Not obvious how to incorporate different views of the image..
- Super slow (must classify each grid, many forward passes).

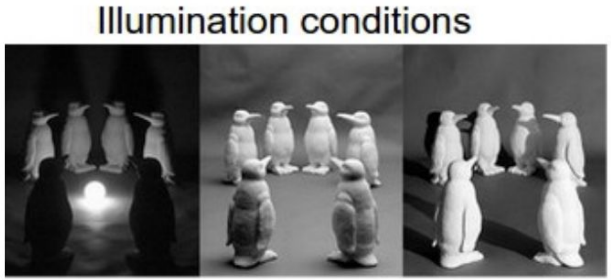
Classify images like **great data scientist you are**, not like the image classification researcher you aren't.

- 99% of the time, you are better off retraining or fine tuning an **existing well known architecture** for your task, rather than designing a novel architecture from scratch (unless your goal is to do research in image classification **itself**).

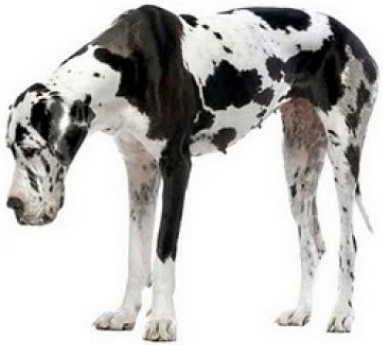
Another recent example:
https://arxiv.org/pdf/1909.05382.pdf (reused Inception)

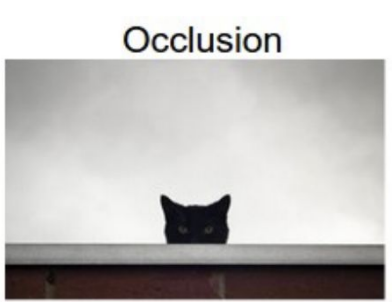
Data augmentation

Many sources of variation. Ideally, want to capture these in our training data to help our model generalize - but probably won't have enough training data to do so.



Scale variation





Parts of this slide borrowed from MIT (with permission).

Data augmentation

Etc

Data augmentation

Quick discussion: should you apply this to the validation set? How about test?

Less obvious transforms may be relevant for your domain

Data augmentation may be applied to the **training set only**.
Never to validation or test.

Computationally expensive (increasing the size of your training set by a factor of n_augmentations).

random_jpeg_quality

https://keras.io/preprocessing/image/

Next time

Reading

- ConvNets and ImageNet Beyond Accuracy: Understanding Mistakes and Uncovering Biases.
- A deep learning system for differential diagnosis of skin diseases.
- Deep Learning with Python: 4, 5.
- Deep Learning: 9

Assignments

- A2 will be posted tomorrow