Convolutional Neural Networks

(i.e. almost magic)

We’re moving into 2000s

Yann LeCun published first convolutional network in 1998 (handwriting recognition)

Still in the drought of funding for neural network research

Everyone (including me) is doing semantic reasoning and symbolic computation

Everyone thinks neural networks are toys

(I built a toy stock trading neural network and lost $2000 in a week)

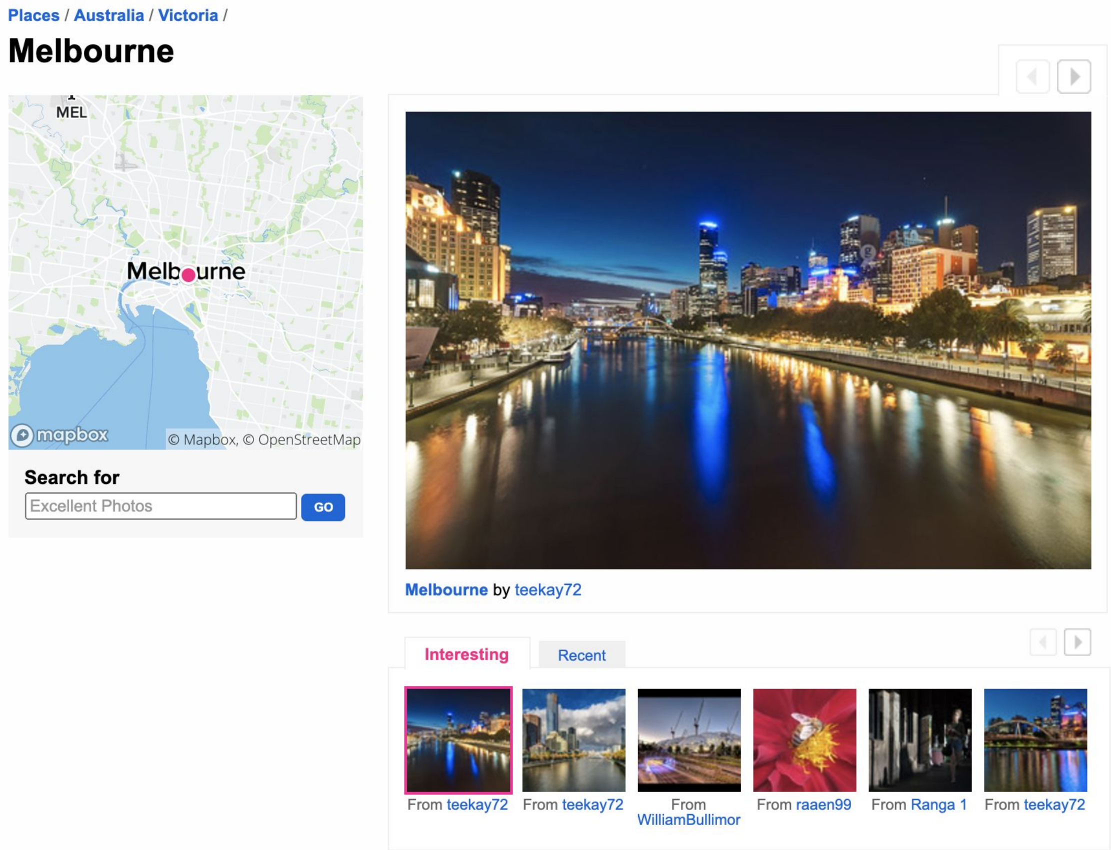
Meanwhile we have Google and (very important) Flickr

The Flickr effect

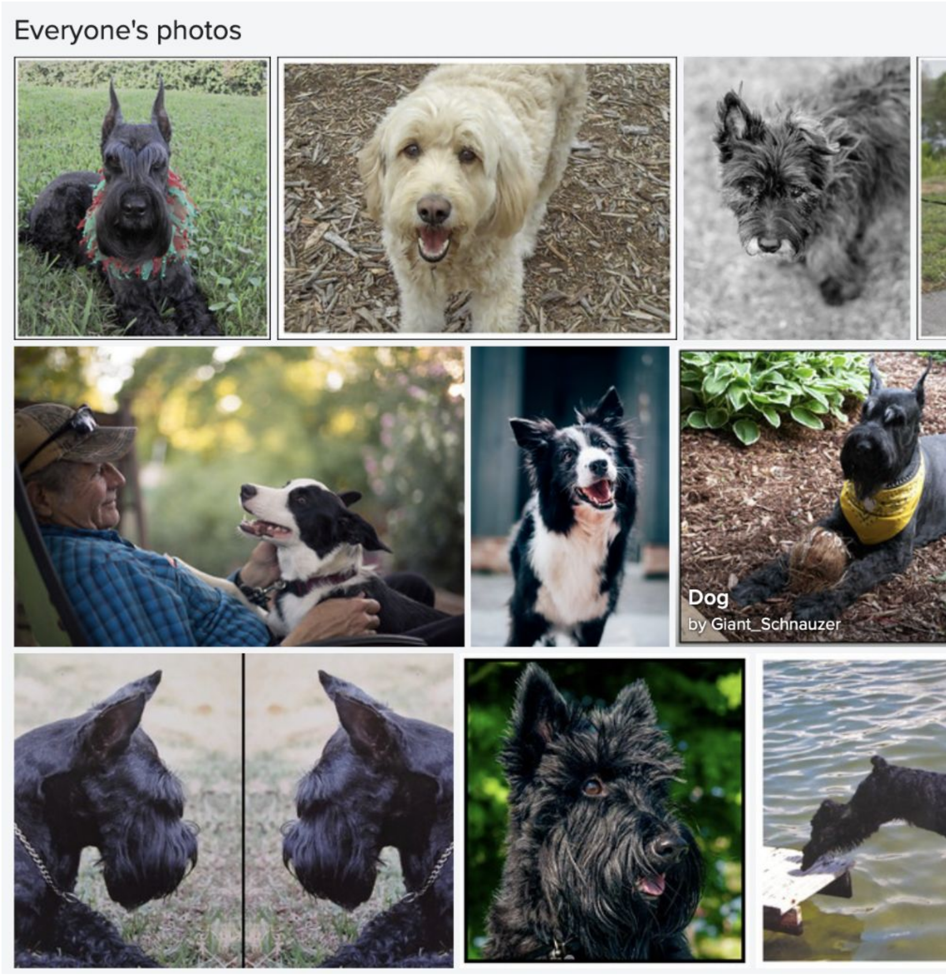
Flickr allows images to be tagged with their content, objects or location

So we now have a giant dataset of public images tagged by humans

Best training data for object recognition



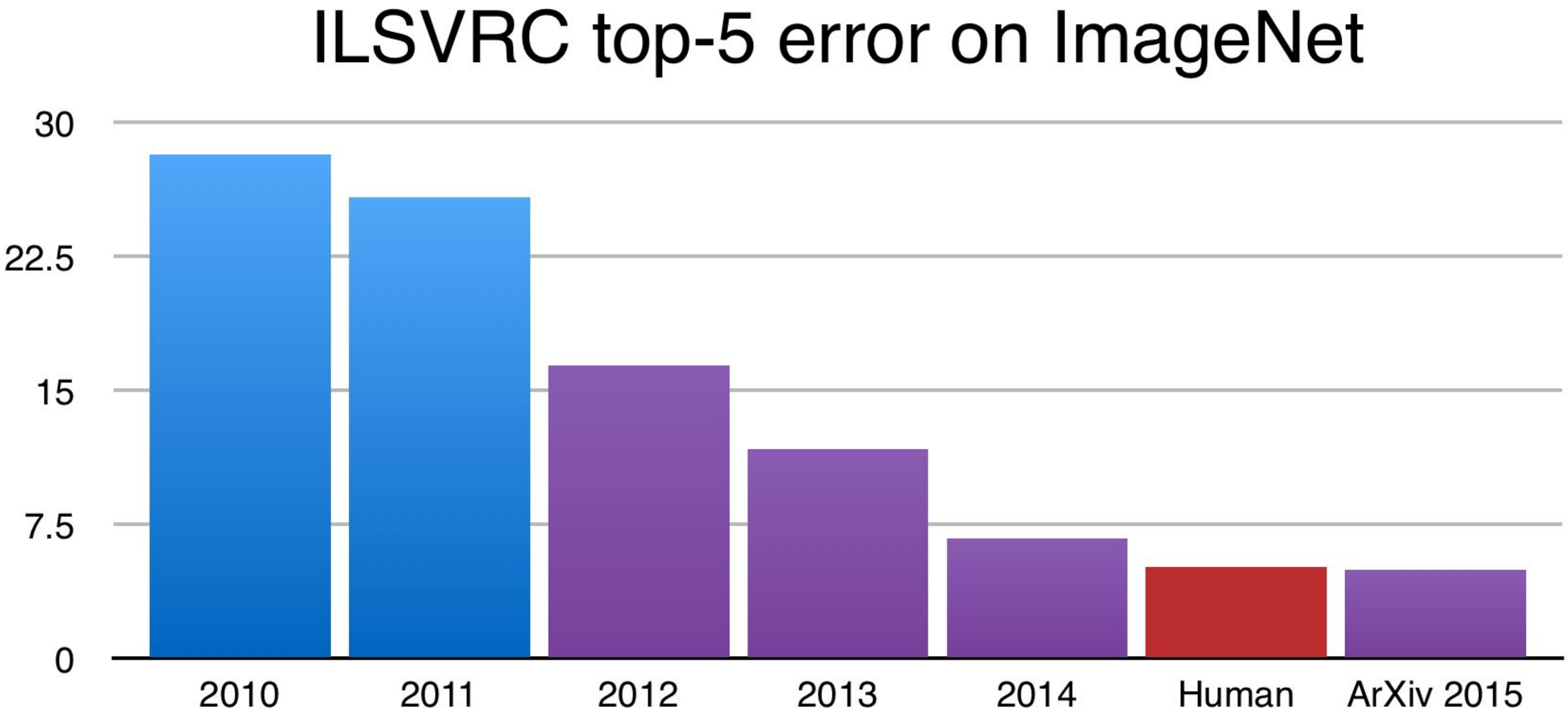

Everyone's photos
Cat
by Roddie


Everyone's photos
Dog
by Giant_Schnauzer

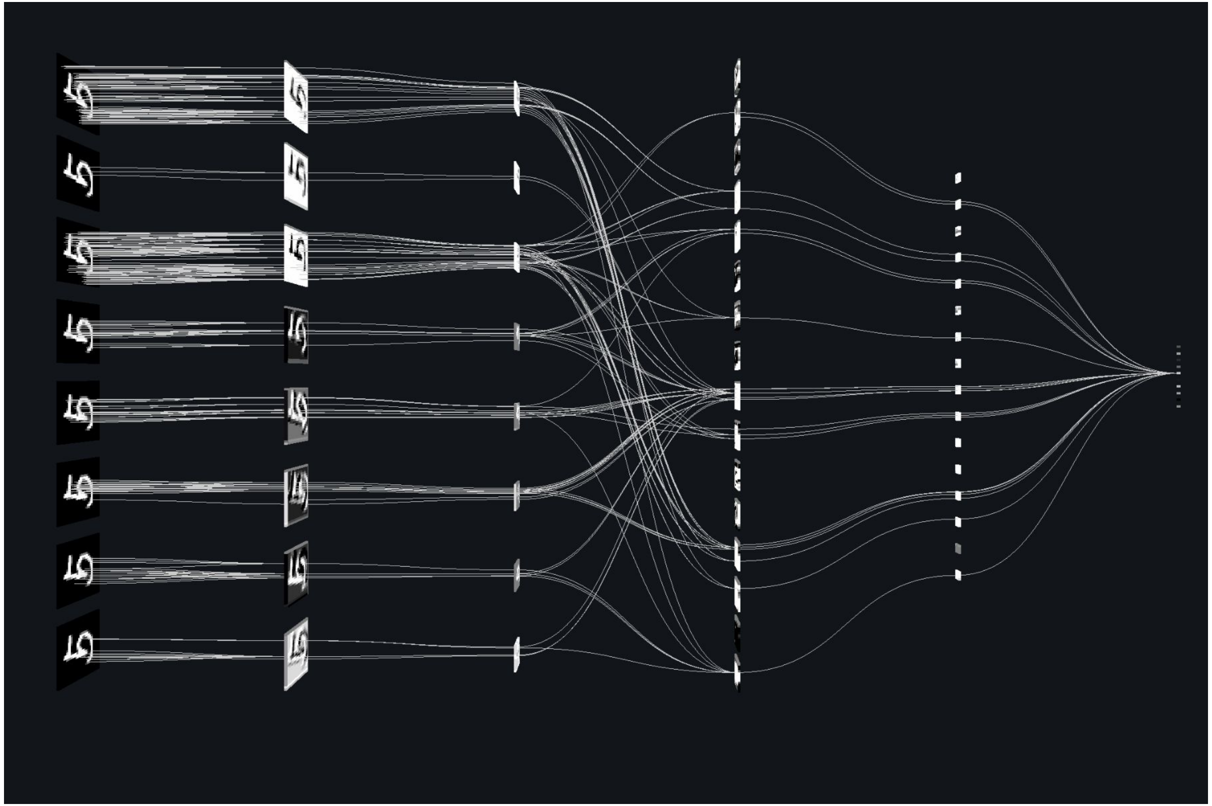
What else changed?

1. Massive training data
2. Ability do matrix multiplication FAST
    a. Graphics cards are designed to do this to render images
    b. They were hacked to just multiply any matrices using a language called CUDA
3.

As a result…. Robots are taking over



So what created this magic? Actually pretty simple math



Feature detection

We did feature engineering before and it sucks.

Or, rather....

Feature X’= f(X) such that it…

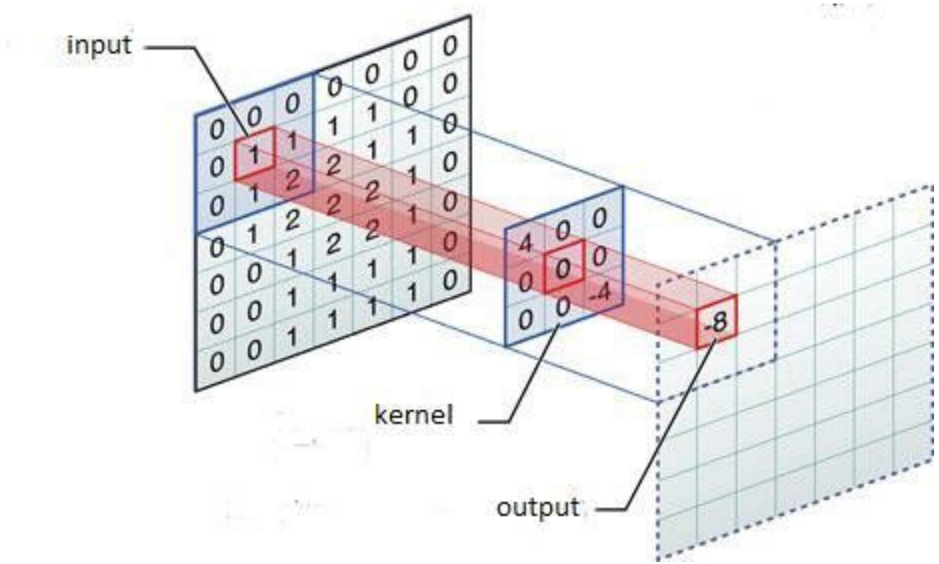
- Amplifies what we care about
- Suppresses what we don’t care about (noise)
- Reduces size of X’ compared to X (so computations down the road are easier)

Can we automate it?

Image filters/kernels

Filter = a small matrix of weights

Slide the filter across the image and perform a **CONVOLUTION**



$$g(x,y) = \omega * f(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega(dx,dy) f(x+dx, y+dy),$$

Filter matrix is such that its value is large when desired feature is present and small if not

Detecting a curve going right

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

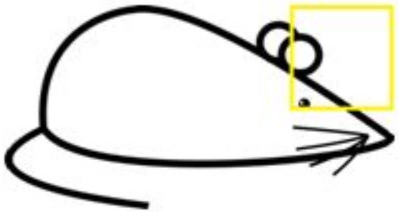
$*$

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

Here a curve is going left



Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

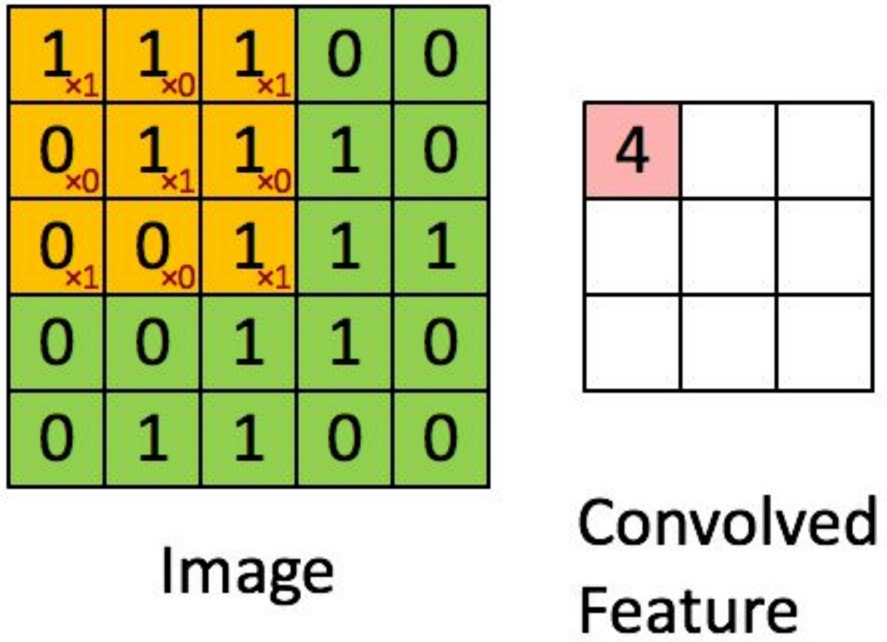
Pixel representation of receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = 0



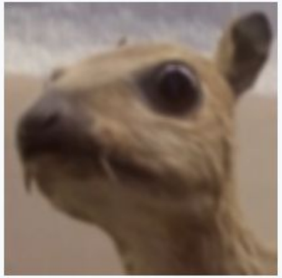
https://ujwlkarn.files.wordpress.com/2016/07/convolution_schematic.gif

Some fun filters:

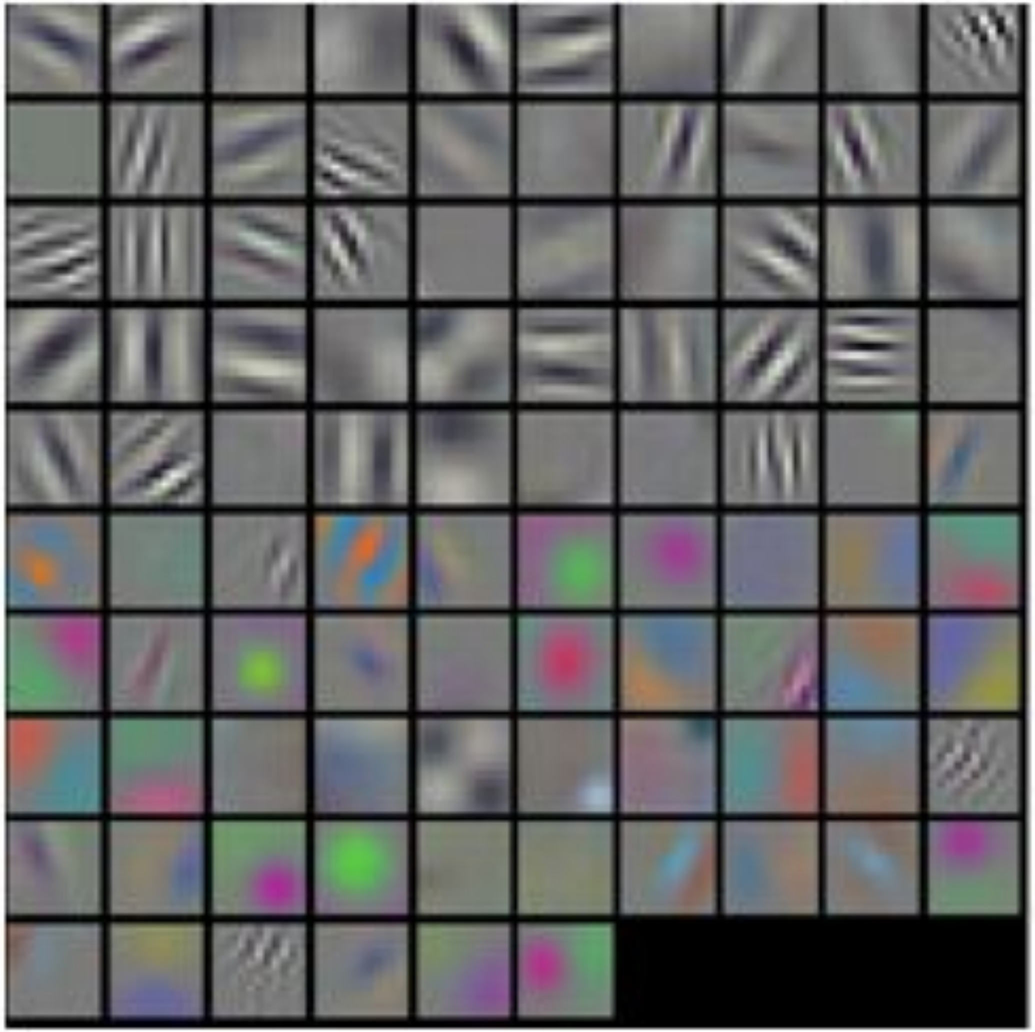
$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$



Some filters



Input

How do we train this thing?

We can start with a set of pre-made filters… but where's the fun in that?

A filter is… just like the weight in a fully connected layer (convolution instead of matrix multiplication)

$$\frac{d}{dx}(f(x) * g(x)) = \left(\frac{d}{dx} f(x)\right) * g(x)$$

It can be trained using gradients!!!

(so we can start with random filters!!!)

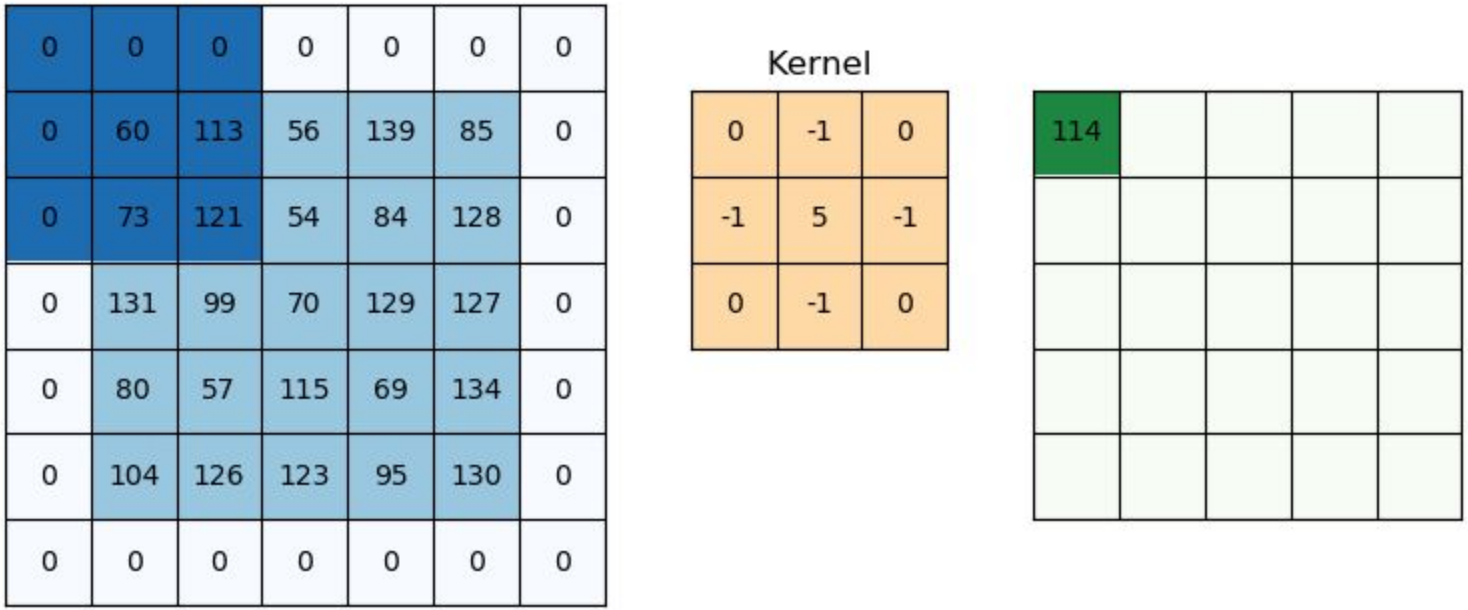
```
tensorflow.keras.layers.Conv2D(filters, kernel_size, strides=(1, 1),
 padding='valid', data_format=None, dilation_rate=(1, 1),
 activation=None, use_bias=True, kernel_initializer='glorot_uniform',
 bias_initializer='zeros', kernel_regularizer=None,
 bias_regularizer=None, activity_regularizer=None,
 kernel_constraint=None, bias_constraint=None)
```

3x3

0.91	0.32	0.07
0.73	0.26	0.81
0.53	0.68	0.14

5x5

0.27	0.64	0.44	0.84	0.29
0.28	0.06	0.89	0.99	0.33
0.64	0.67	0.08	0.38	0.03
0.04	0.31	0.16	0.57	0.08
0.87	0.85	0.97	0.71	0.96


0 0 0 0 0 0 0
0 60 113 56 139 85 0
0 73 121 54 84 128 0
0 131 99 70 129 127 0
0 80 57 115 69 134 0
0 104 126 123 95 130 0
0 0 0 0 0 0 0
Kernel
0 -1 0
-1 5 -1
0 -1 0
114

Striiiiiide

Stride value is speed with which the filter moves across an image (default= 1)

(how many pixels do we skip?)

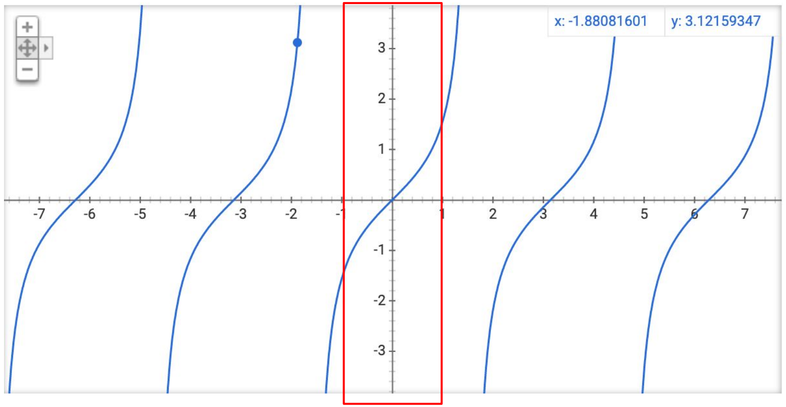
Dimensionality reduction

Filtered image is smaller

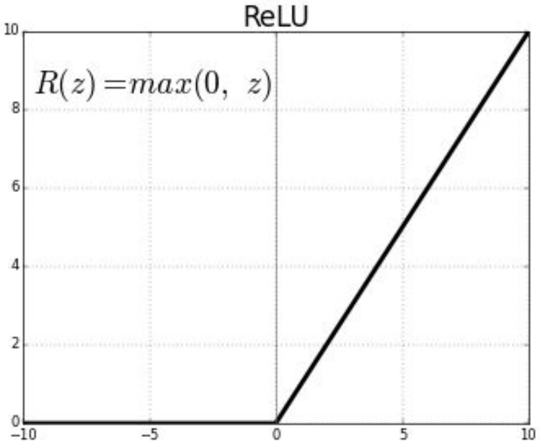
$$n_{out} = floor(\frac{n_{in} - f}{s}) + 1$$

Activation function

Last time we used y = tan(x)



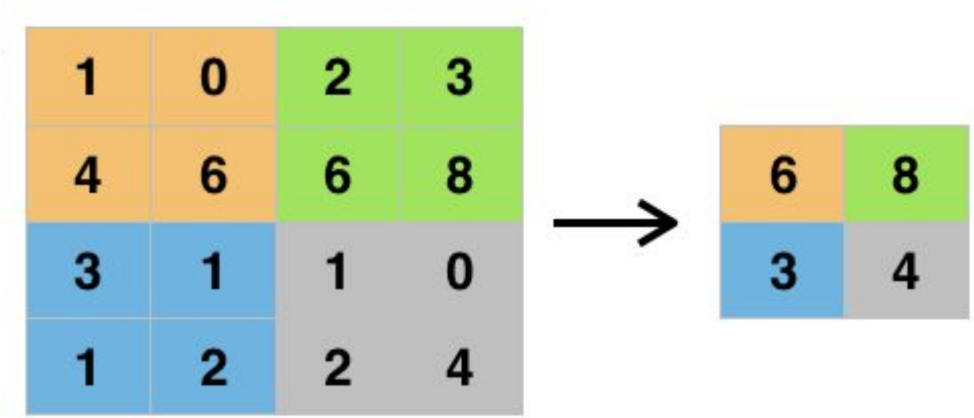
In CNN's we use ReLU



Original Image Feature Map Non-Linear

Pooling

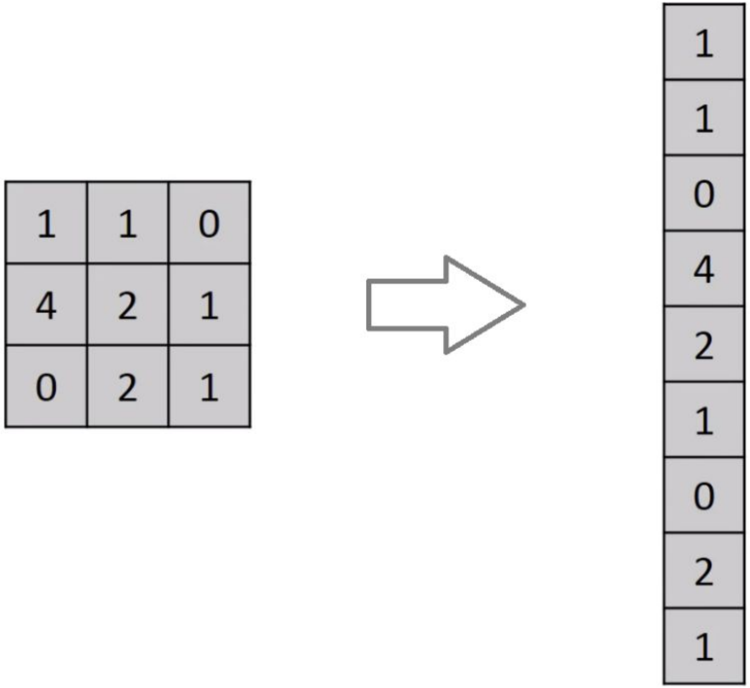
Simple = 2x2 Max-Pooling



Other methods: average(mean) pooling, L2 Norm pooling

((((pooling layers are dying out and are not used on modern CNNs))))

Flattening -- fully connected layer



Flattening data

SoftMax

SoftMax activation function maps the outputs of a dense layer to a vector whose elements sum up to 1

Output of SoftMax is probability of belonging to a class

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$

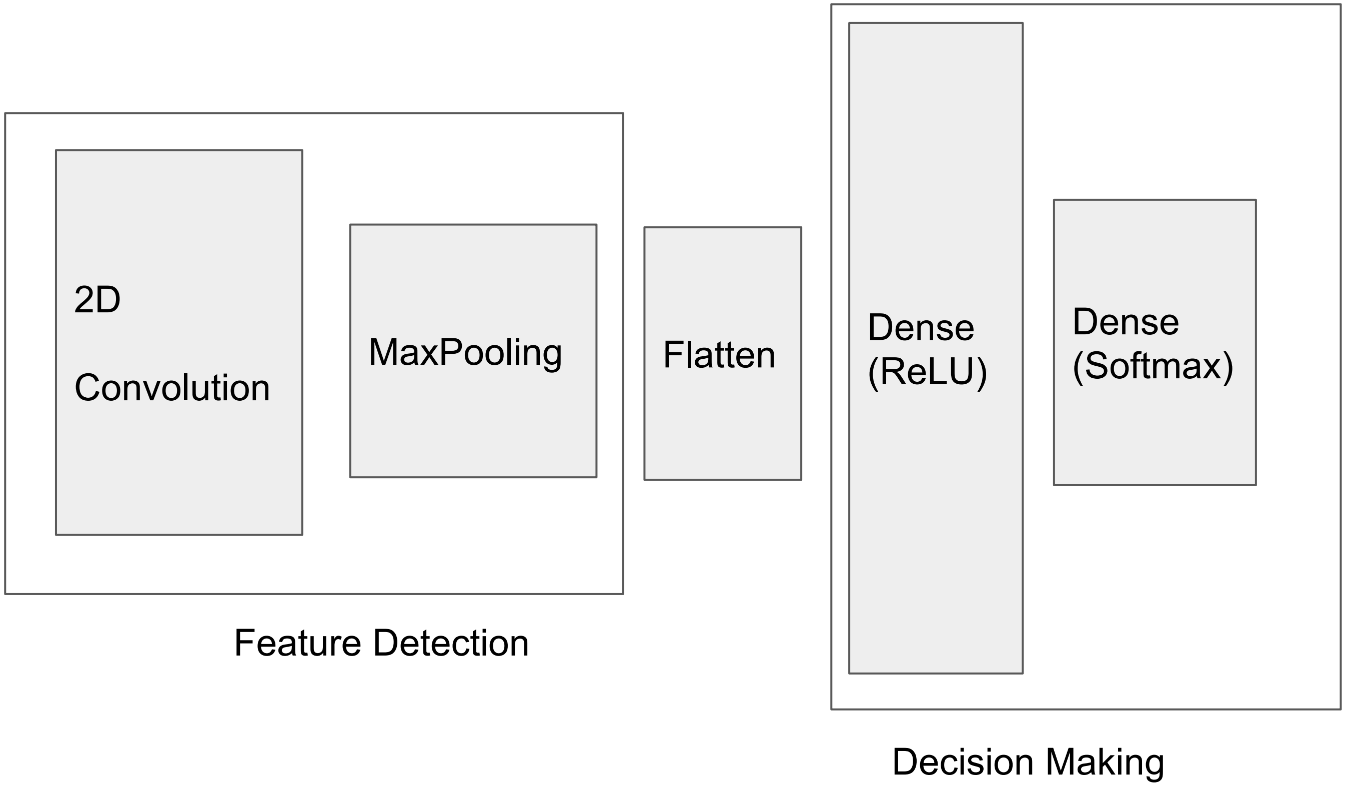
Loss Function -- Categorical Cross-Entropy

$$H(y, \hat{y}) = \sum_i y_i \log \frac{1}{\hat{y}_i} = -\sum_i y_i \log \hat{y}_i$$

0<=y^<=1

y(i) = 0 or 1

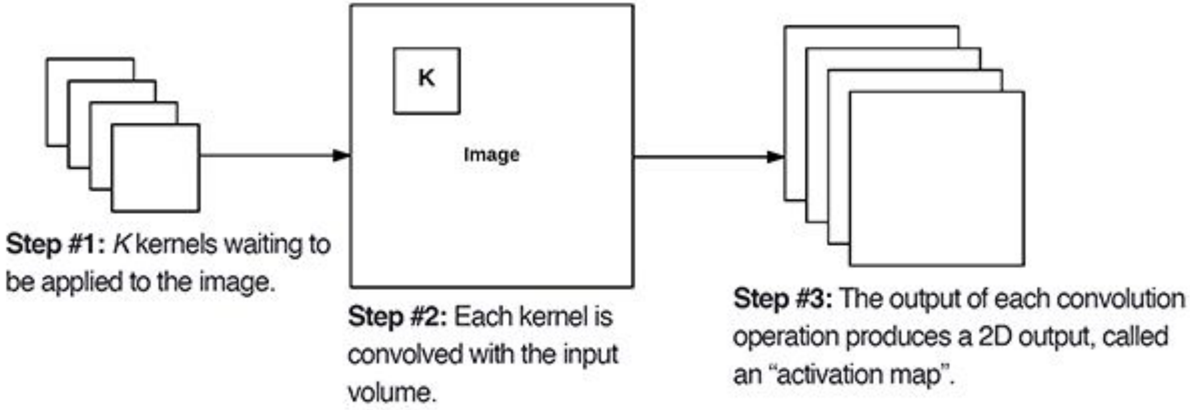
Finally…. Network architecture



Keras library / Tensorflow

Yay we’re using libraries like civilized people

So we don’t have to write everything from scratch



Activa