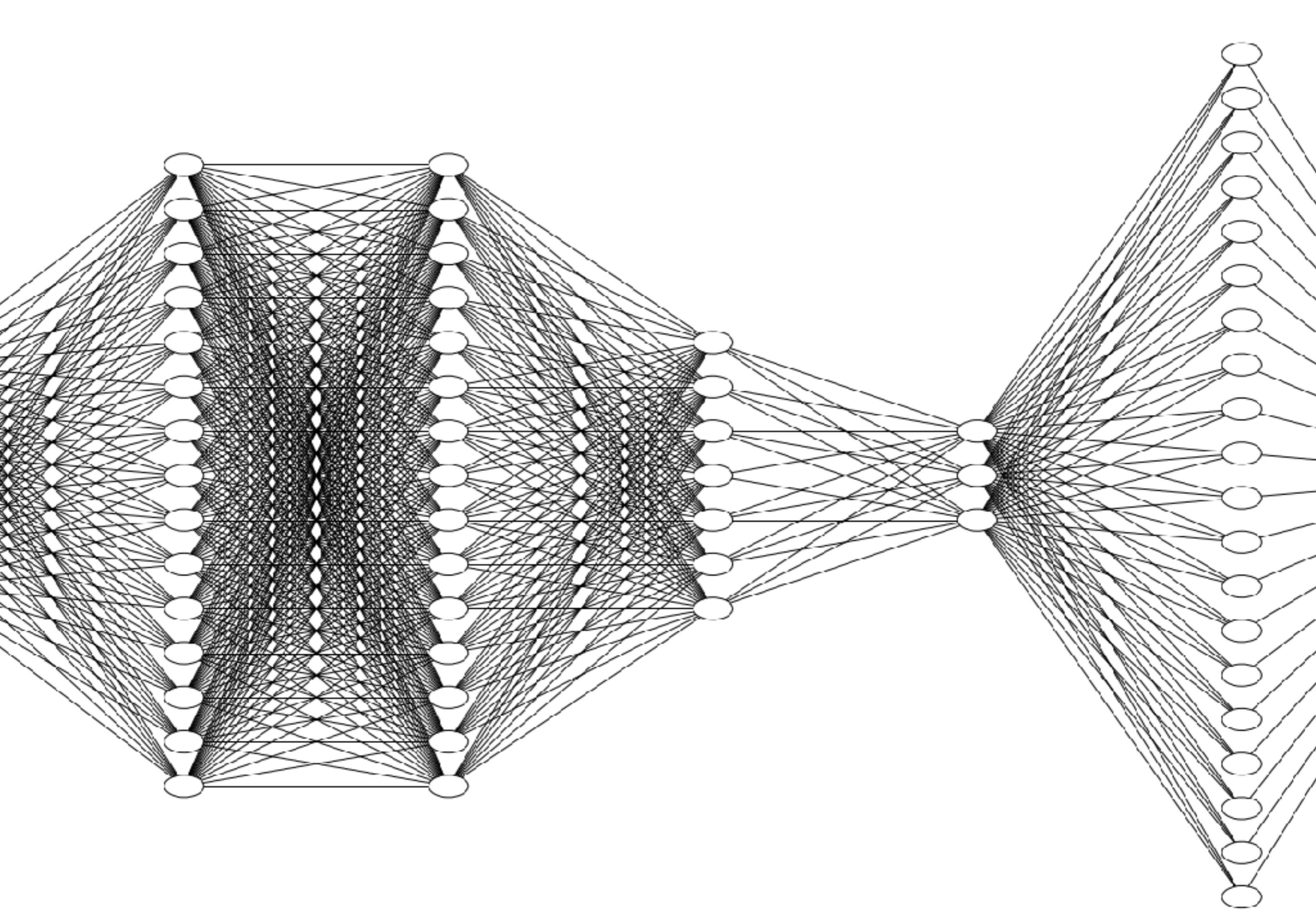


Deep Learning II

Introduction to Computer Vision:
Convolutional Neural Networks

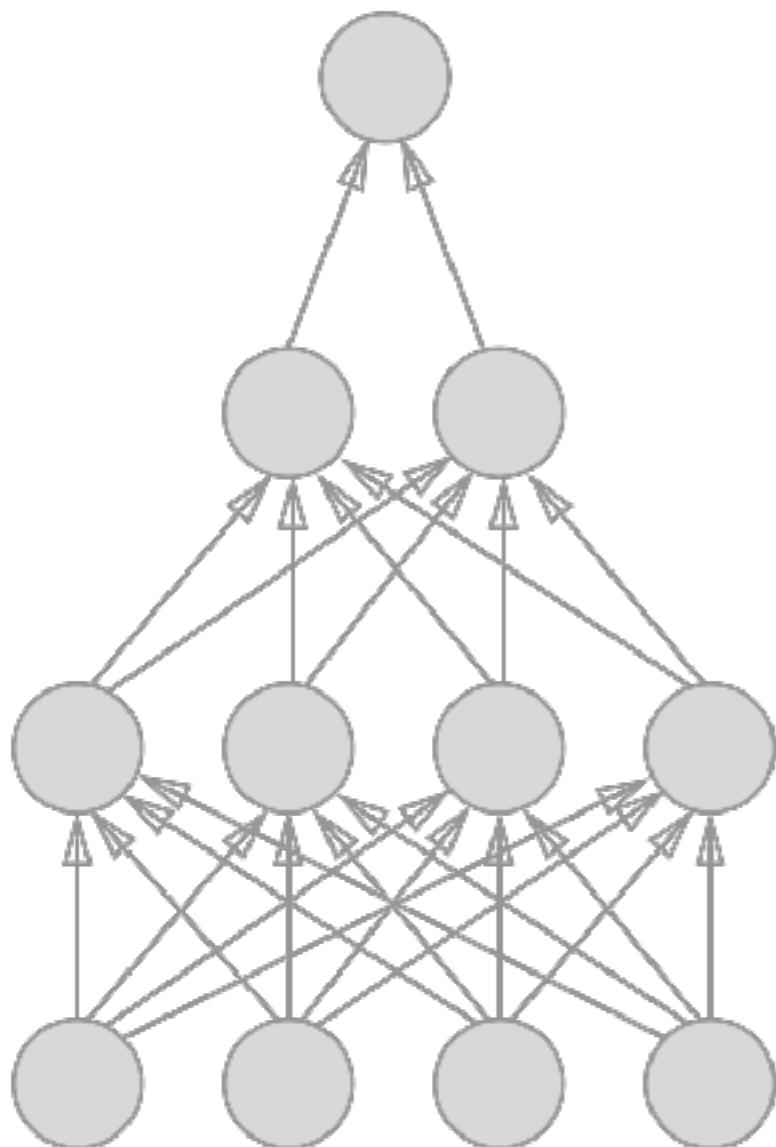
Yesterday, we managed to
make a fully connected
neural network recognize
handwritten digits



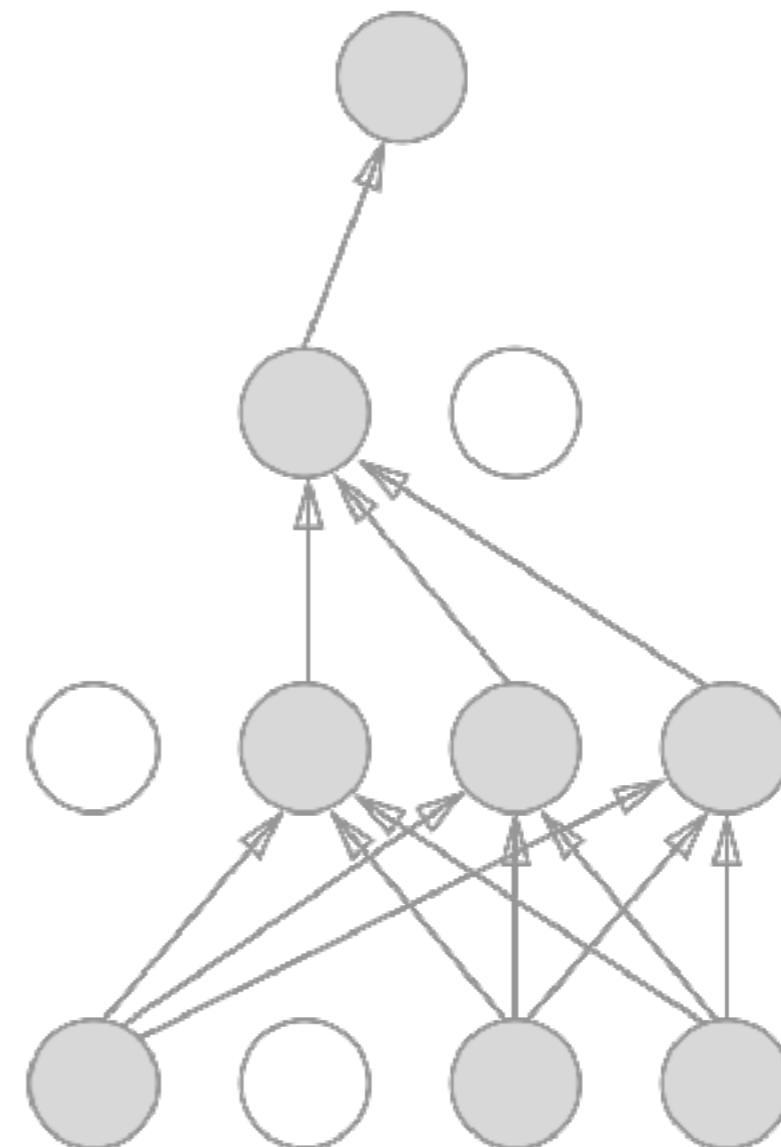
Does going deeper help?

- Intuition suggests so
- In practice, that's not easy
 - Overfitting
 - Optimization

Dropout



Before Dropout



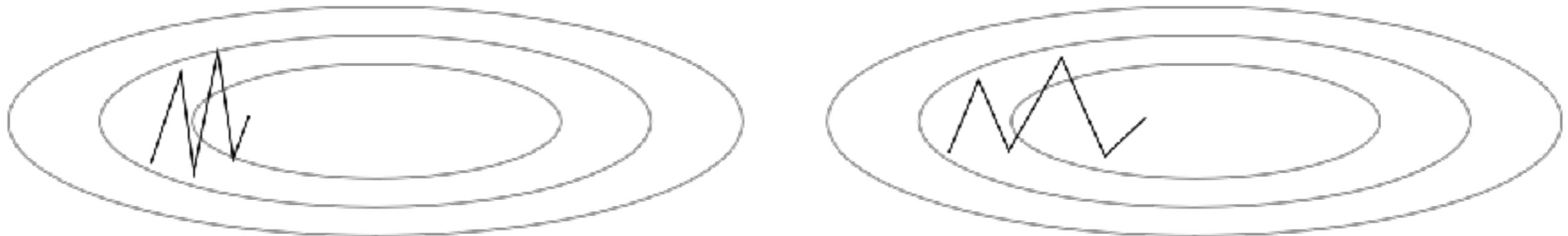
After Dropout

Momentum

Momentum helps accelerating SGD in the relevant direction by adding a fraction of the update vector of the past time step to the current update vector

$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta} J(\theta)$$

$$\theta = \theta - v_t$$



Adam

Adam (Adaptive Moment Estimation) keeps exponentially decaying estimates of the first and second moments (mean and uncentered variance)

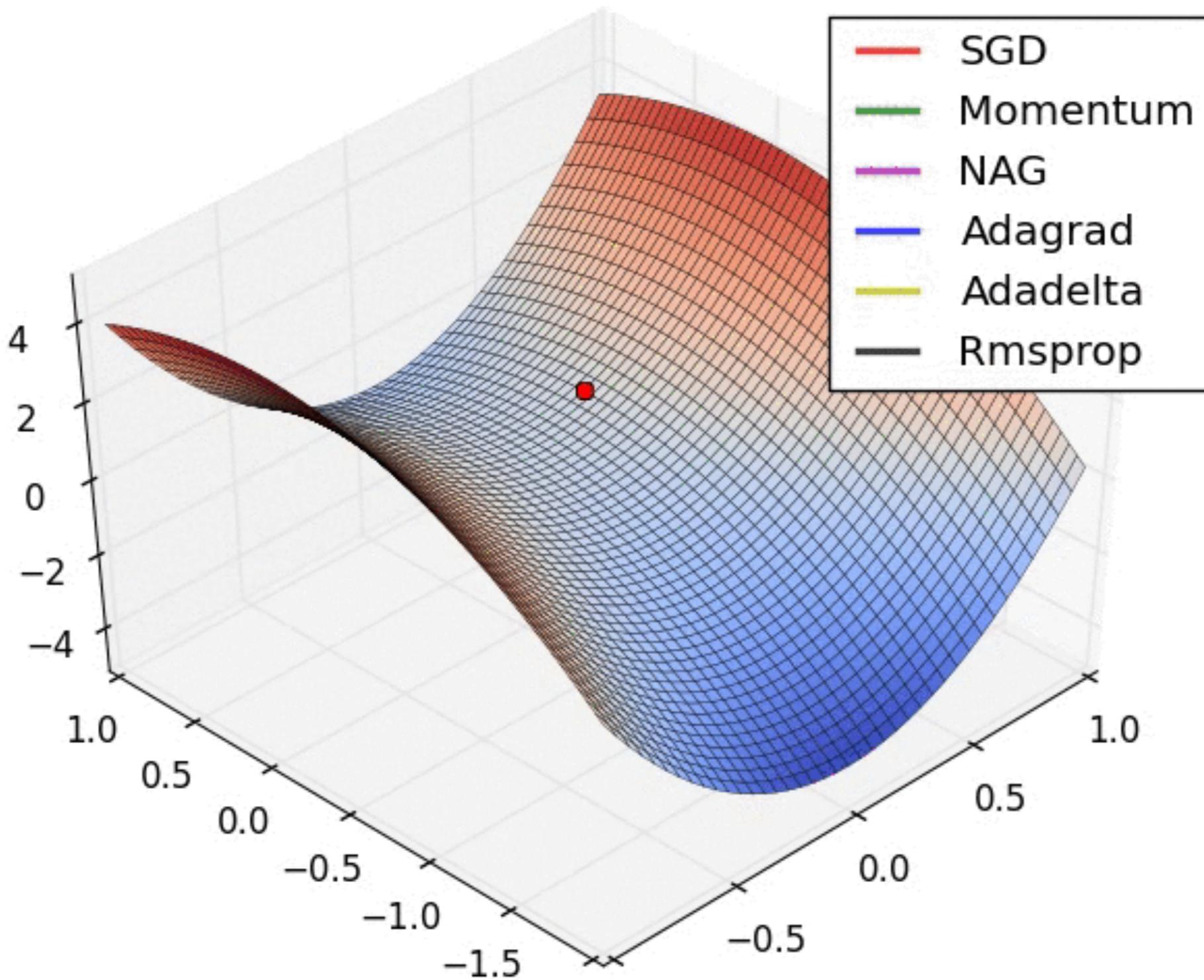
$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$



Animation by Sebastian Ruder
<http://sebastianruder.com/optimizing-gradient-descent/>

Fully connected neural
network get increasingly **hard**
to train when they scale

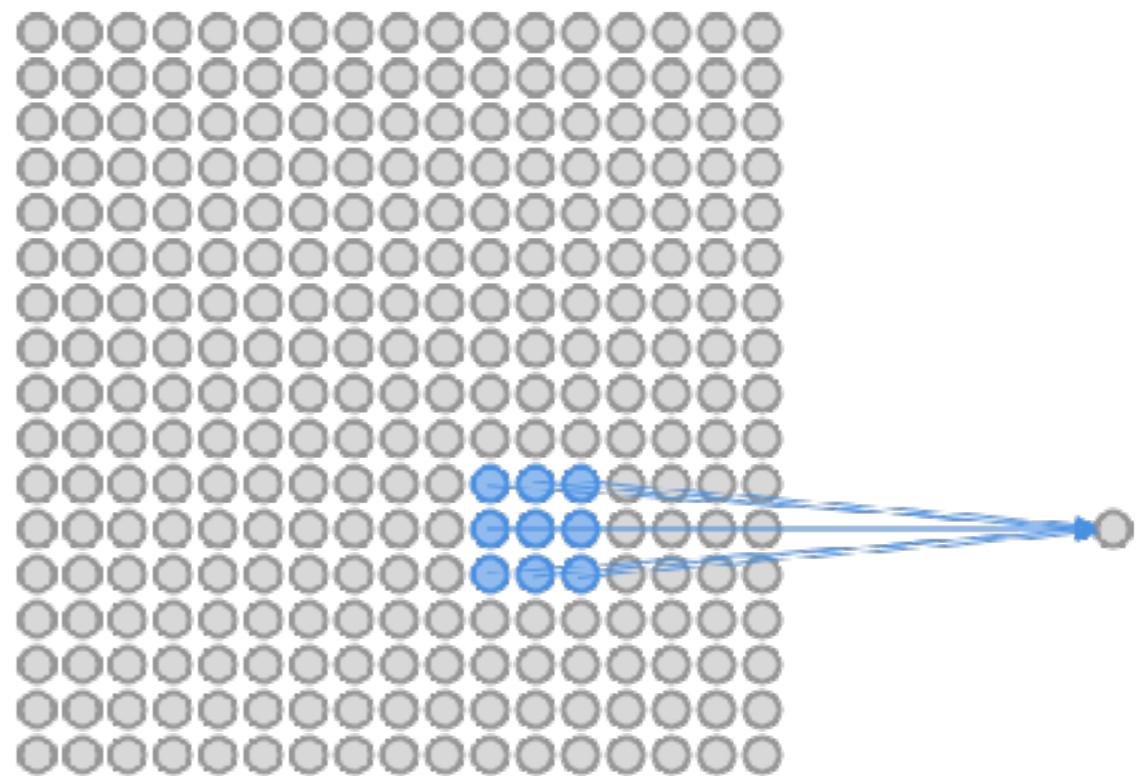
Does it even *make sense*
to use a fully connected
network on input images?

We'd rather design an architecture
which takes advantage of the
spatial structure of the input

We'd also like our model
to be **lighter** and **more**
scalable

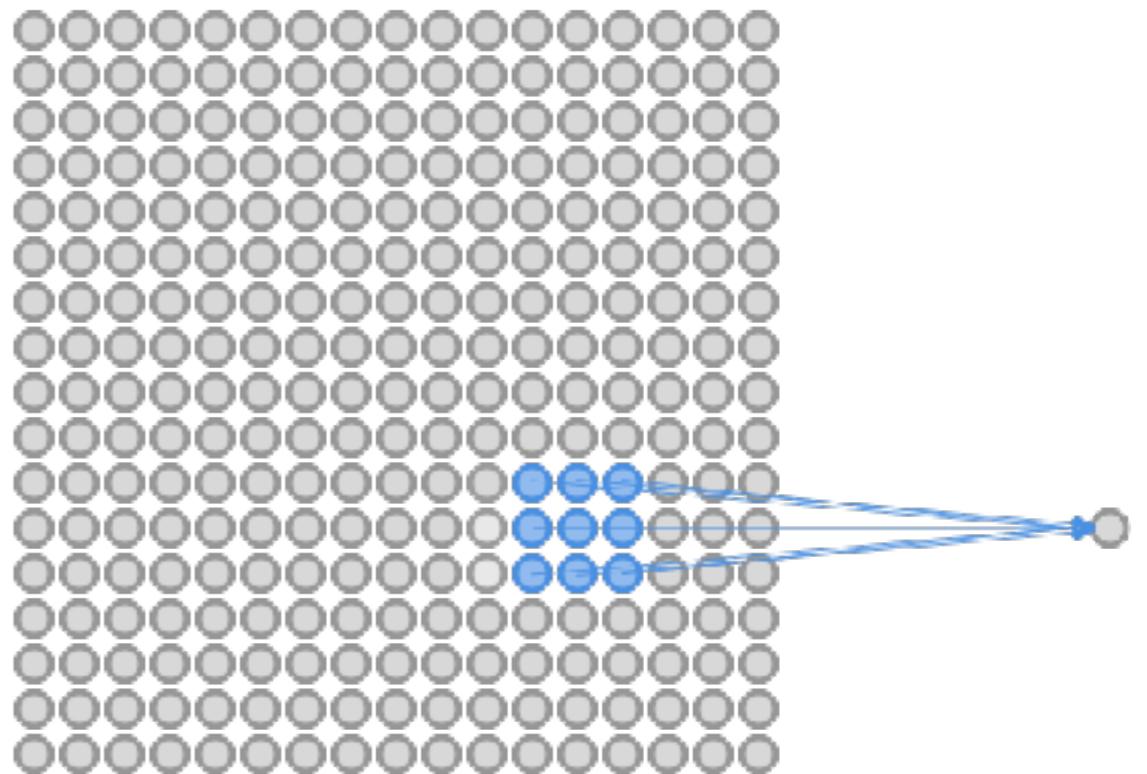
Convolutional Nets

- The input neurons can now be arranged according to the pixels position
- We won't connect each input pixel to every hidden neuron, we'd rather make localized connections
- The input region that maps to the hidden neuron is called **local receptive field** of the hidden neuron

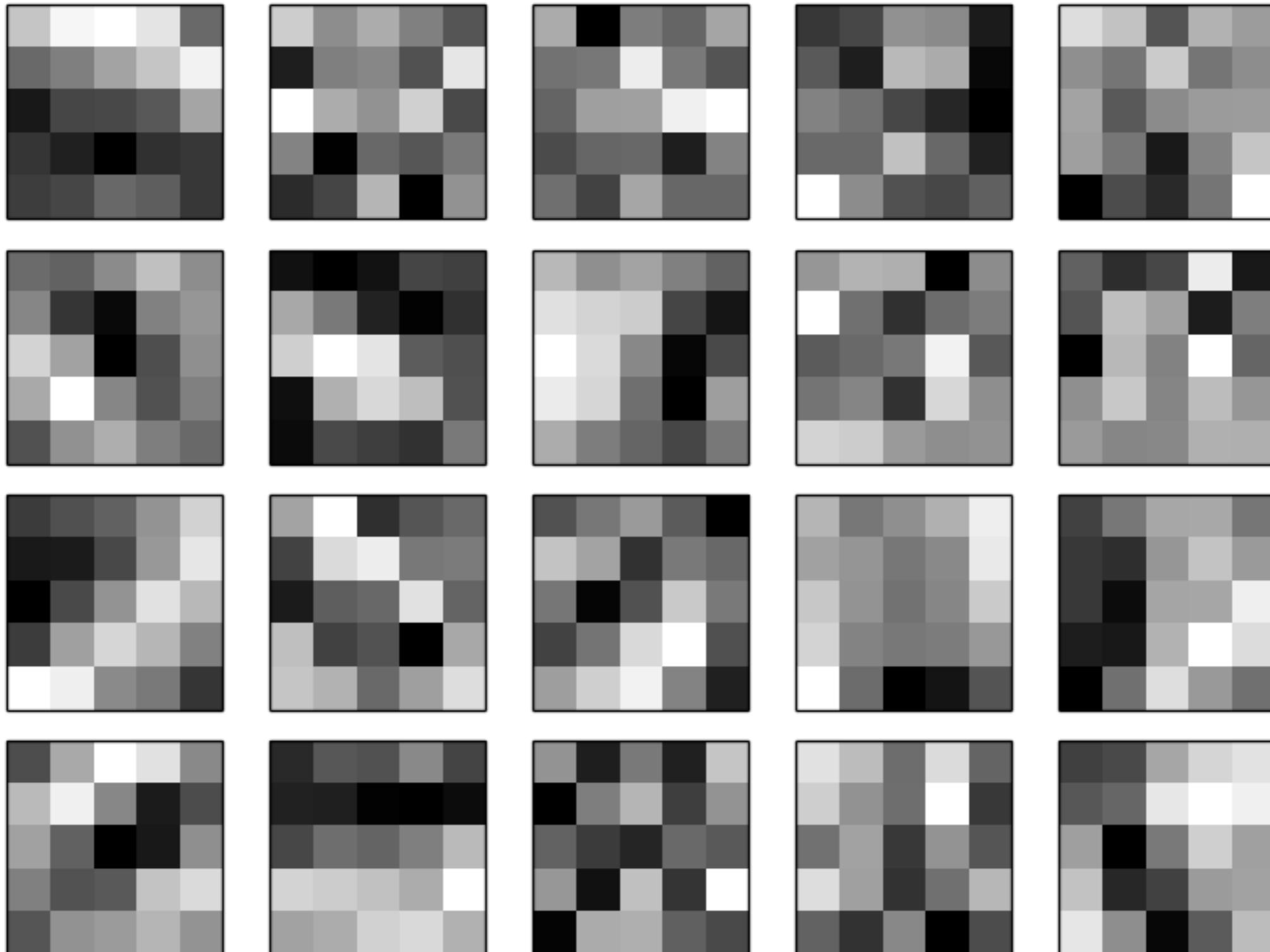


Convolutional Nets

- Receptive fields slide through the image until they cover it all
- The step size of this slide is called **stride**
- The map from the input layer to the hidden layer is called **feature map**
- Multiple feature maps share weights (called **kernels** or **filters**) and biases



Filters, kernels, shared weights



We can think of a convolution as the complete **mapping** between the input space and the hidden space

The term *convolution* is used because of the mathematical expression of this mapping

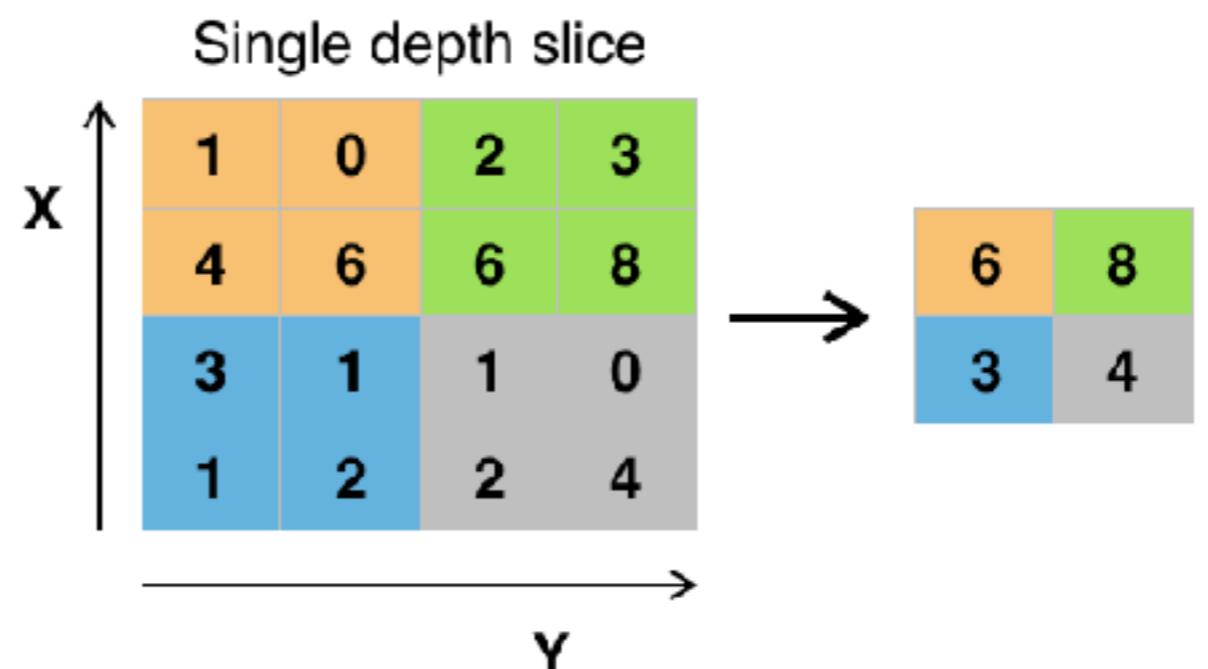
$$\sigma \left(b + \sum_{l=0}^f \sum_{m=0}^f w_{l,m} a_{j+l, k+m} \right)$$

Convolutional neural networks also contain **pooling layers**, usually immediately after convolutional layers

Pooling layers map *feature map outputs* to a ***condensed feature map***, depending on a particular pooling function

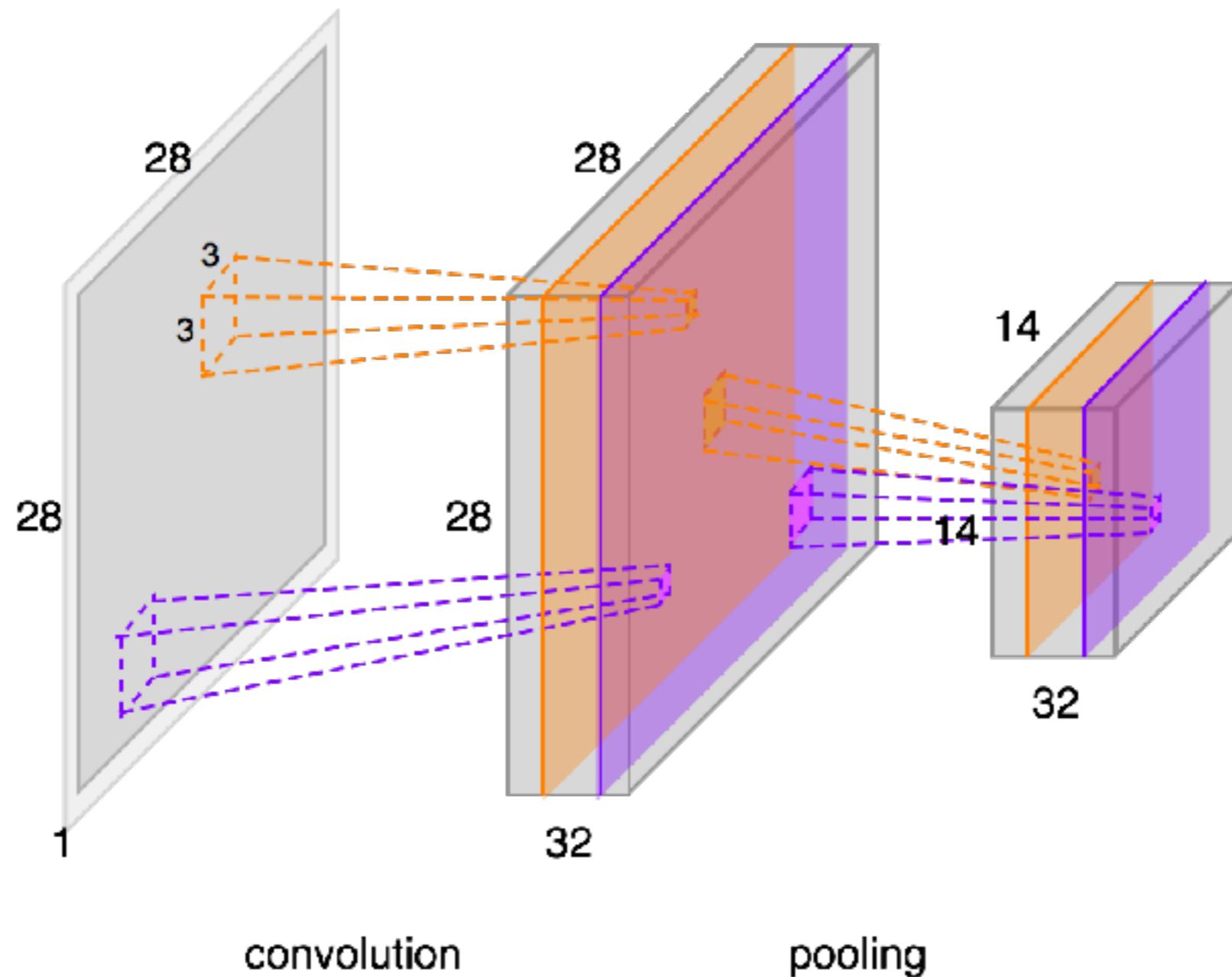
Pooling

- **Max-pooling** extracts the maximum activation among a pooling region
- **Average-pooling** calculates the average activation among a pooling region
- More complex pooling layers perform different operations. Any kind of (reasonable) pooling layer can be designed



All pooling layers share one characteristic: they **lose spatial information** while **reducing the parameters** in the network

Putting it all together



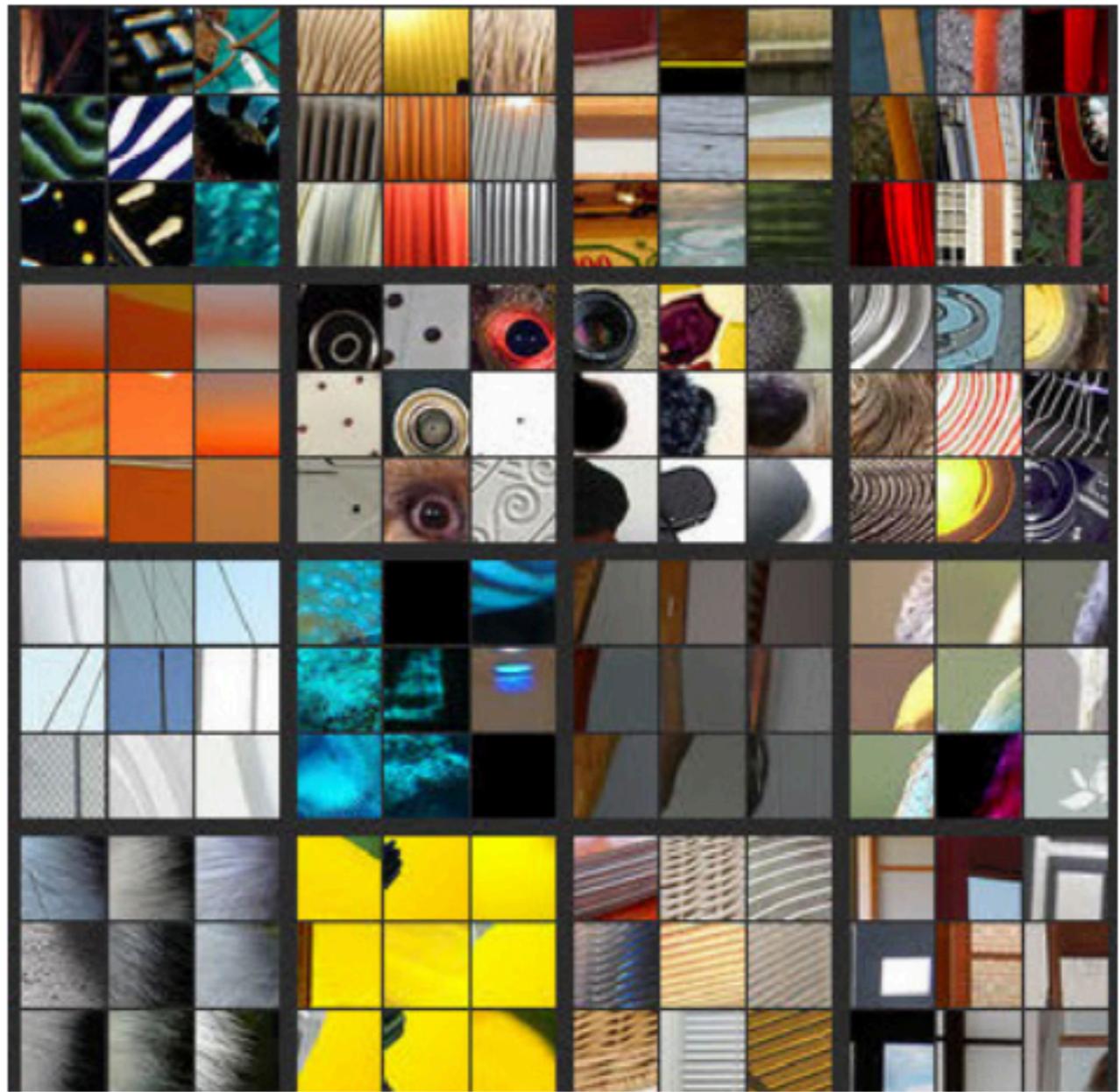
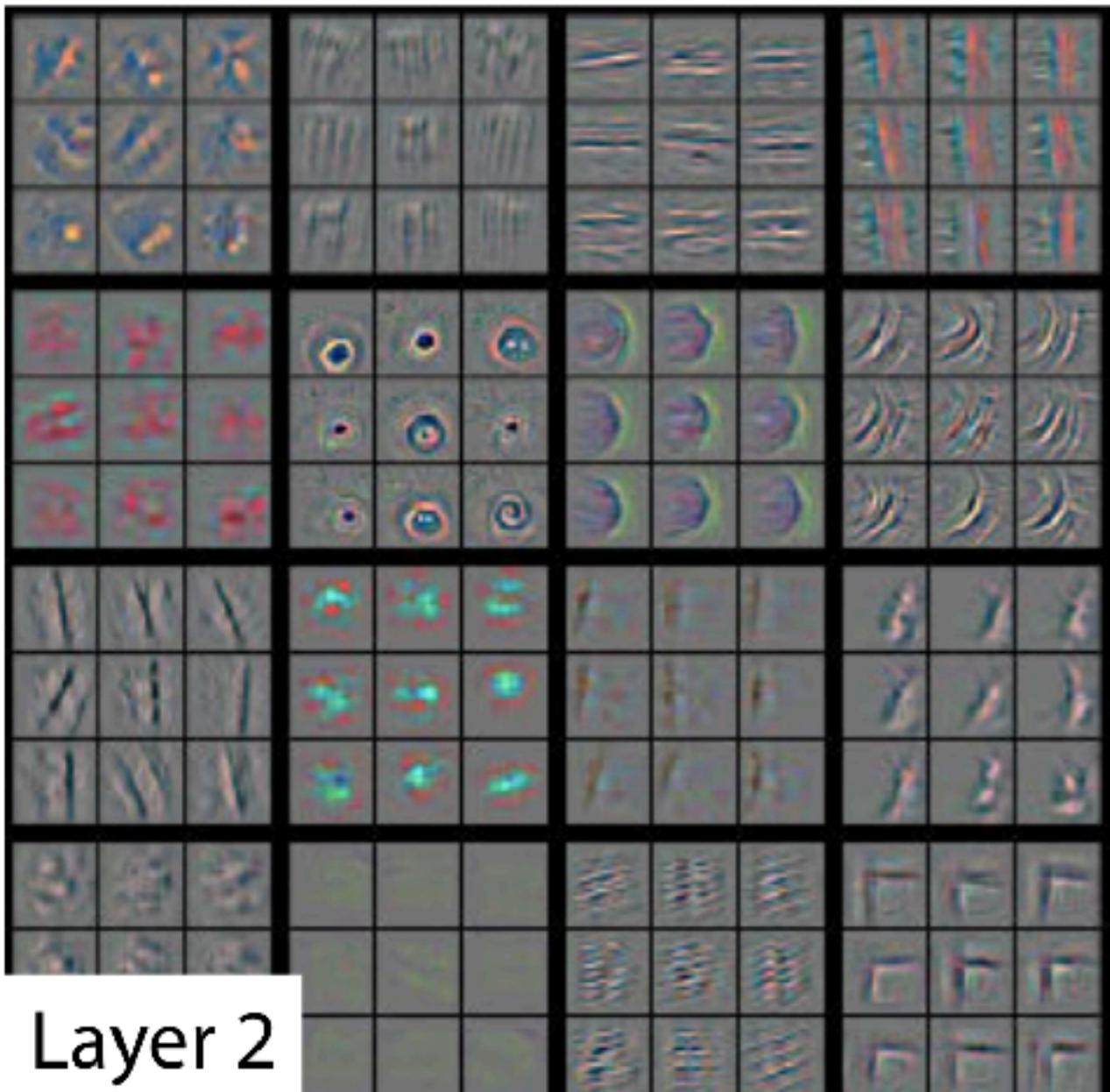
We can still use fully connected
layers, dropout, softmax
activations within CNNs

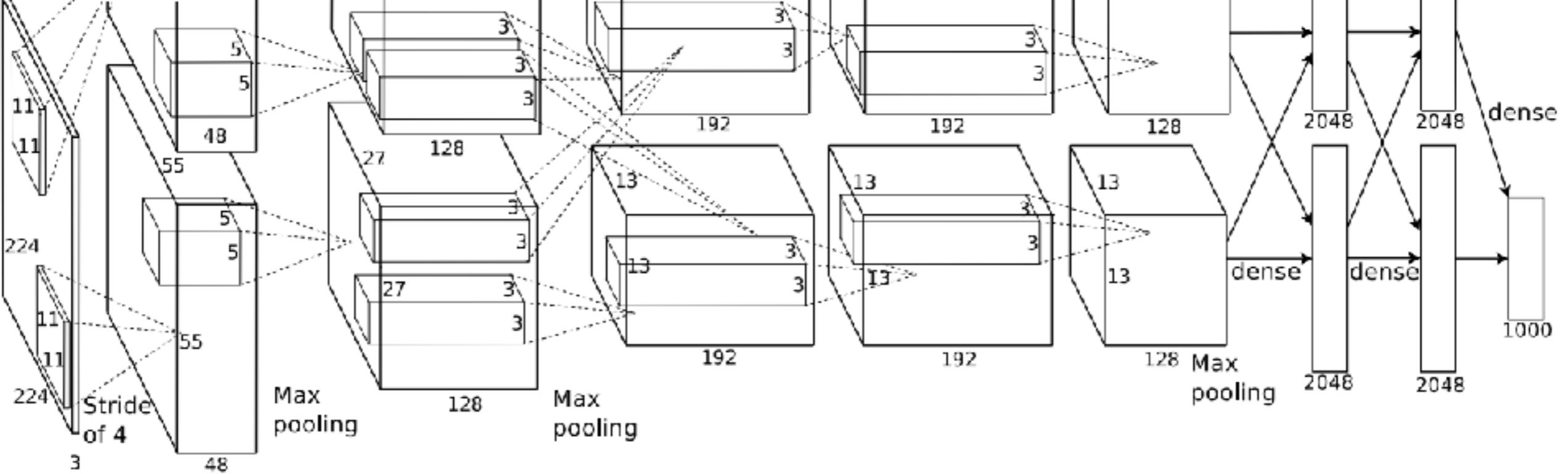
You still train a convolutional neural network in the same way we trained fully connected ones:
backpropagation

Visualize learning



Visualize learning

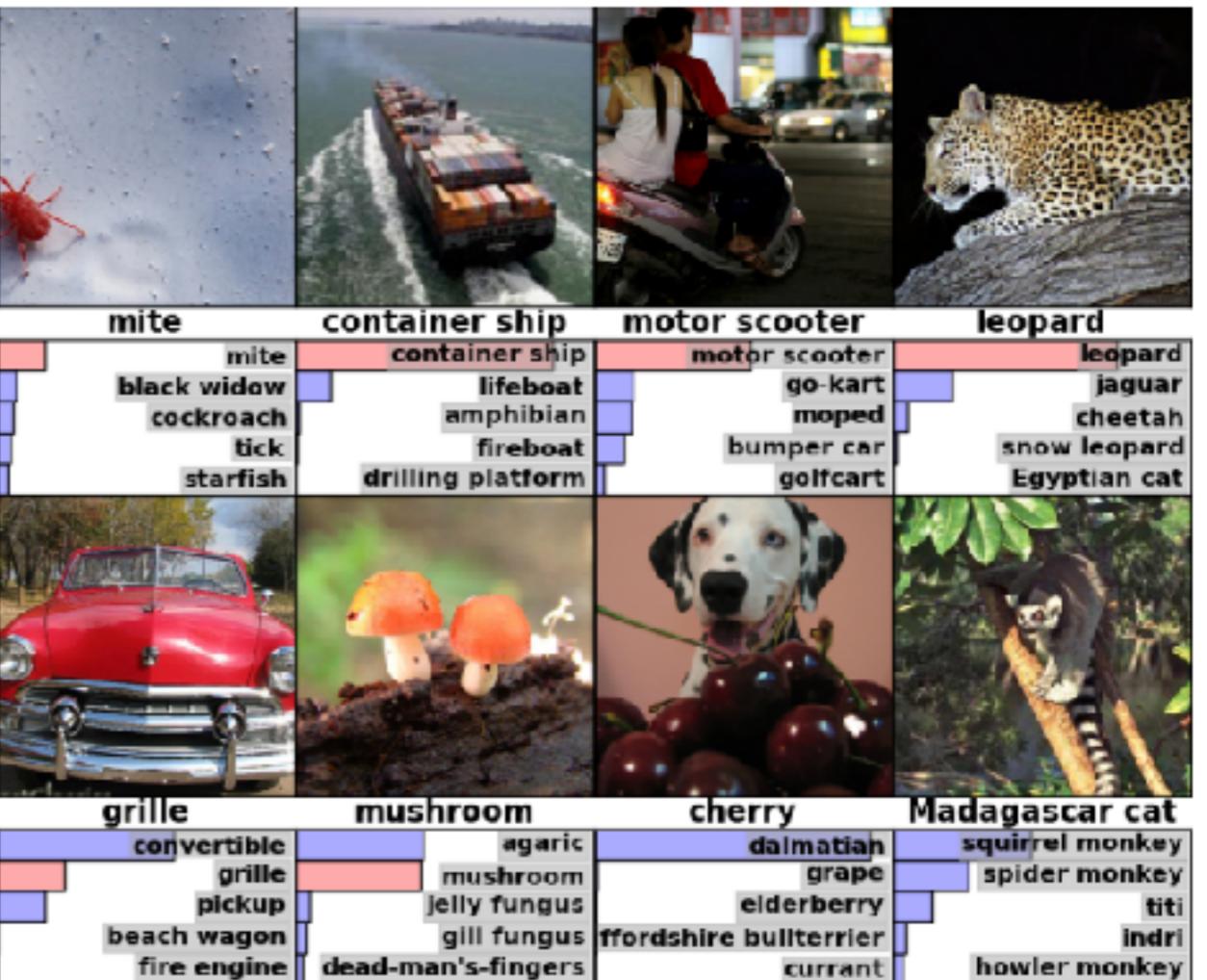




Krizhevsky et al. (2012)

Alexnet is able to give the right answer 63% of the time

One of the top 5 answers it gives is right 85% of the time

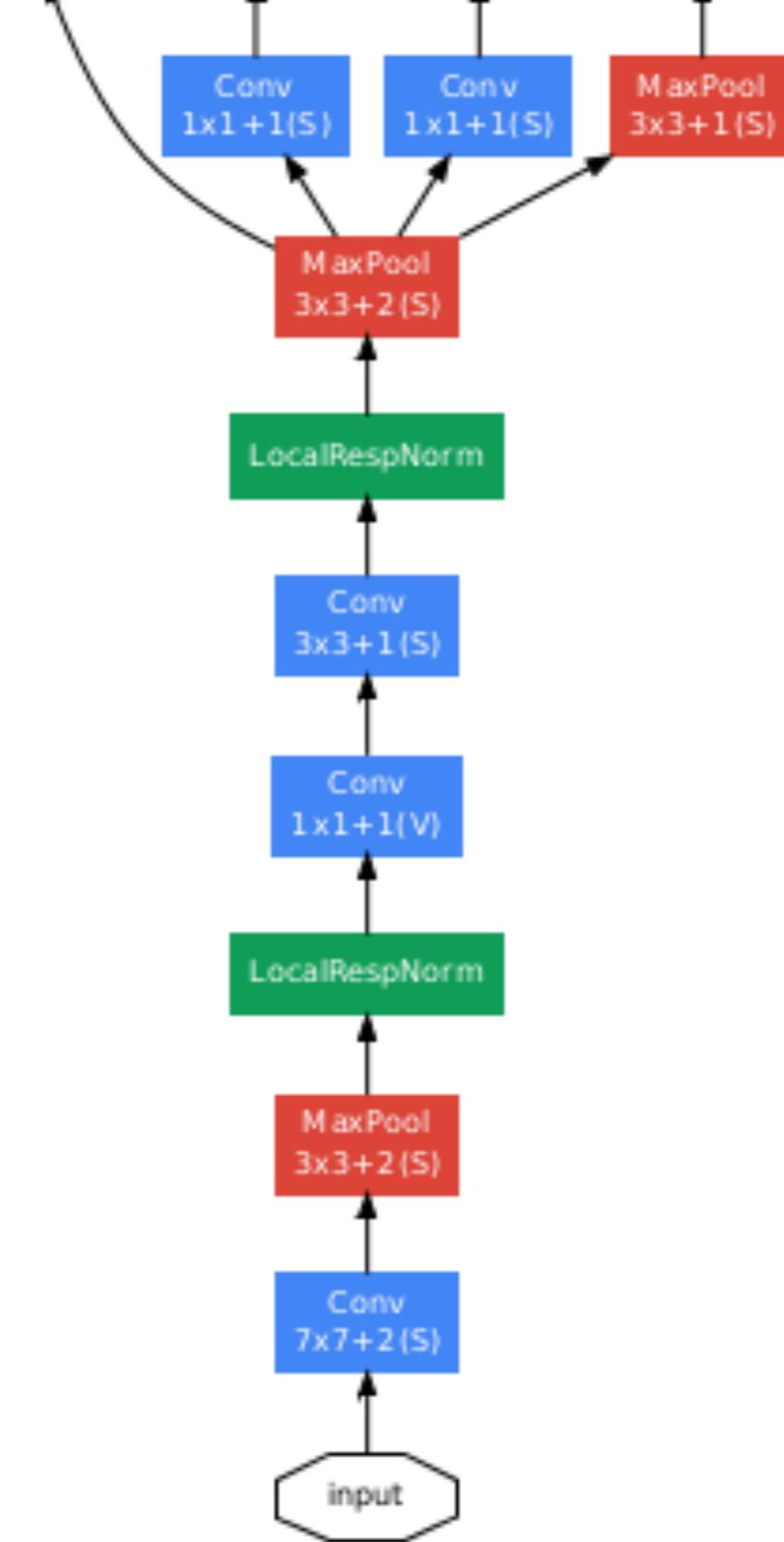


Why 2012?

The origins of convolutional neural networks go back to the 70s, and developed as we know them today during the late 90s. So *why* did we need to wait until 2012 to see their potential?

- Large datasets (e.g. ImageNet)
- Cheap parallel computation (e.g. GPUs)

GoogLeNet (2014)



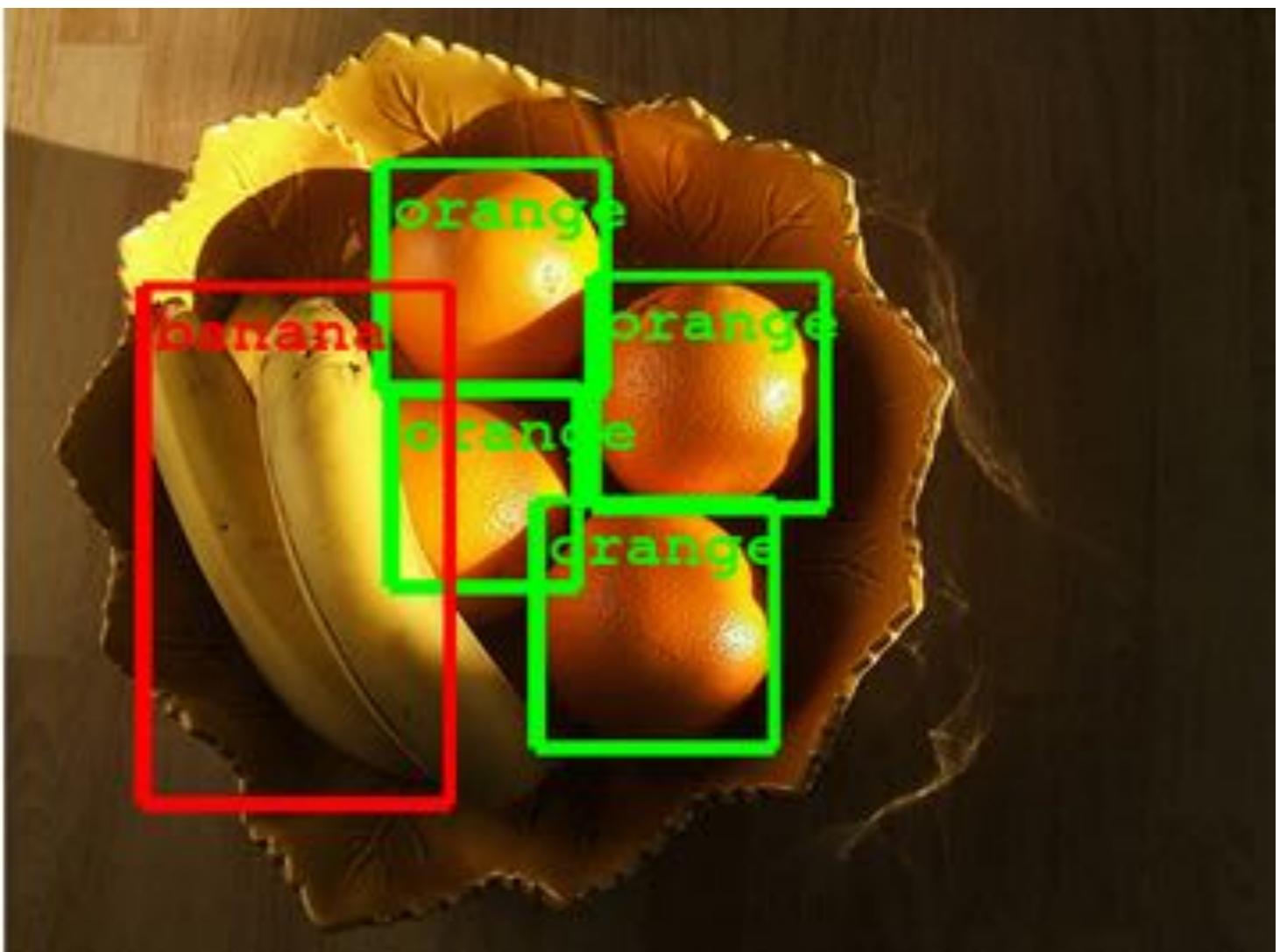
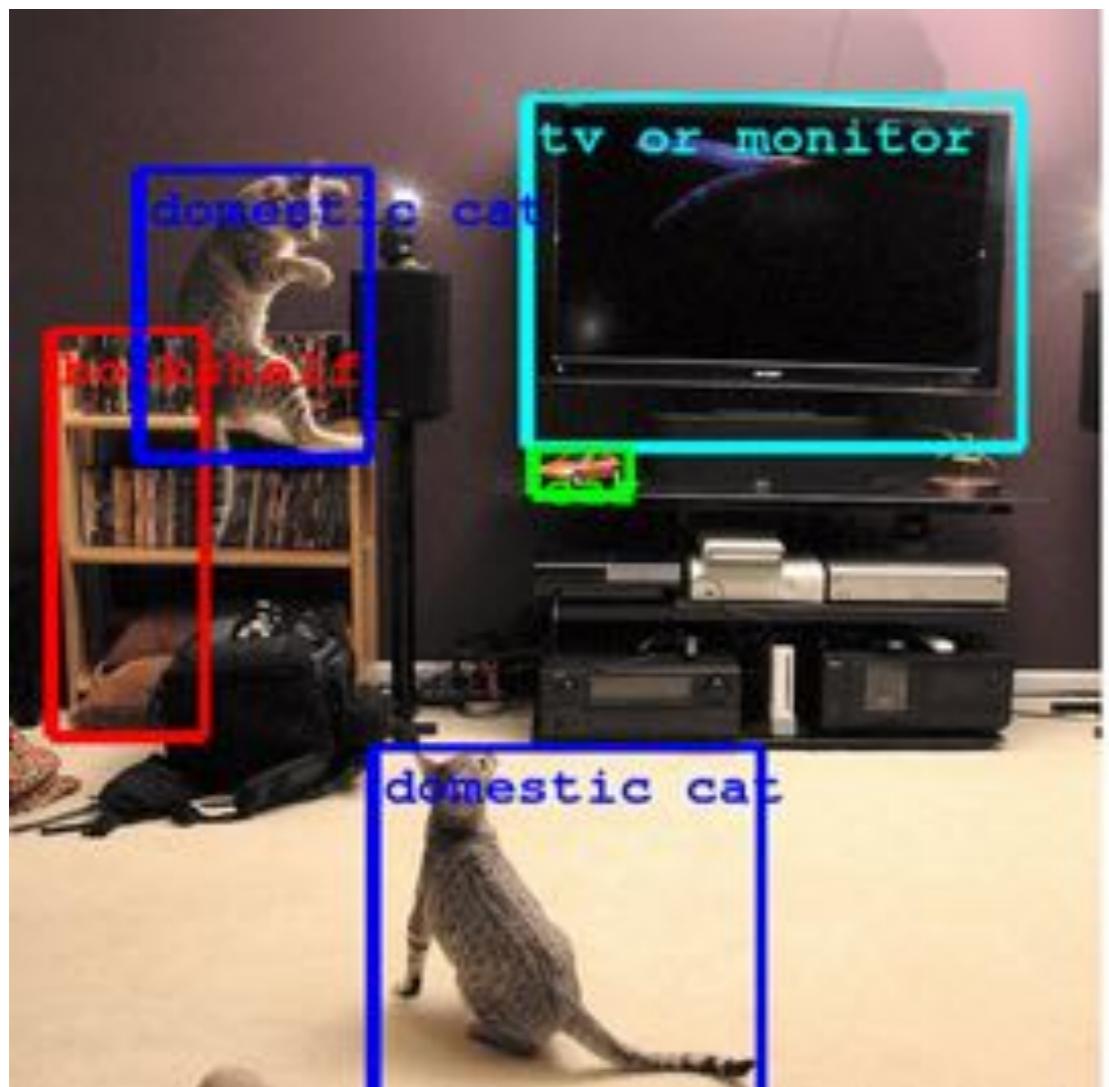


CNNs are everywhere now

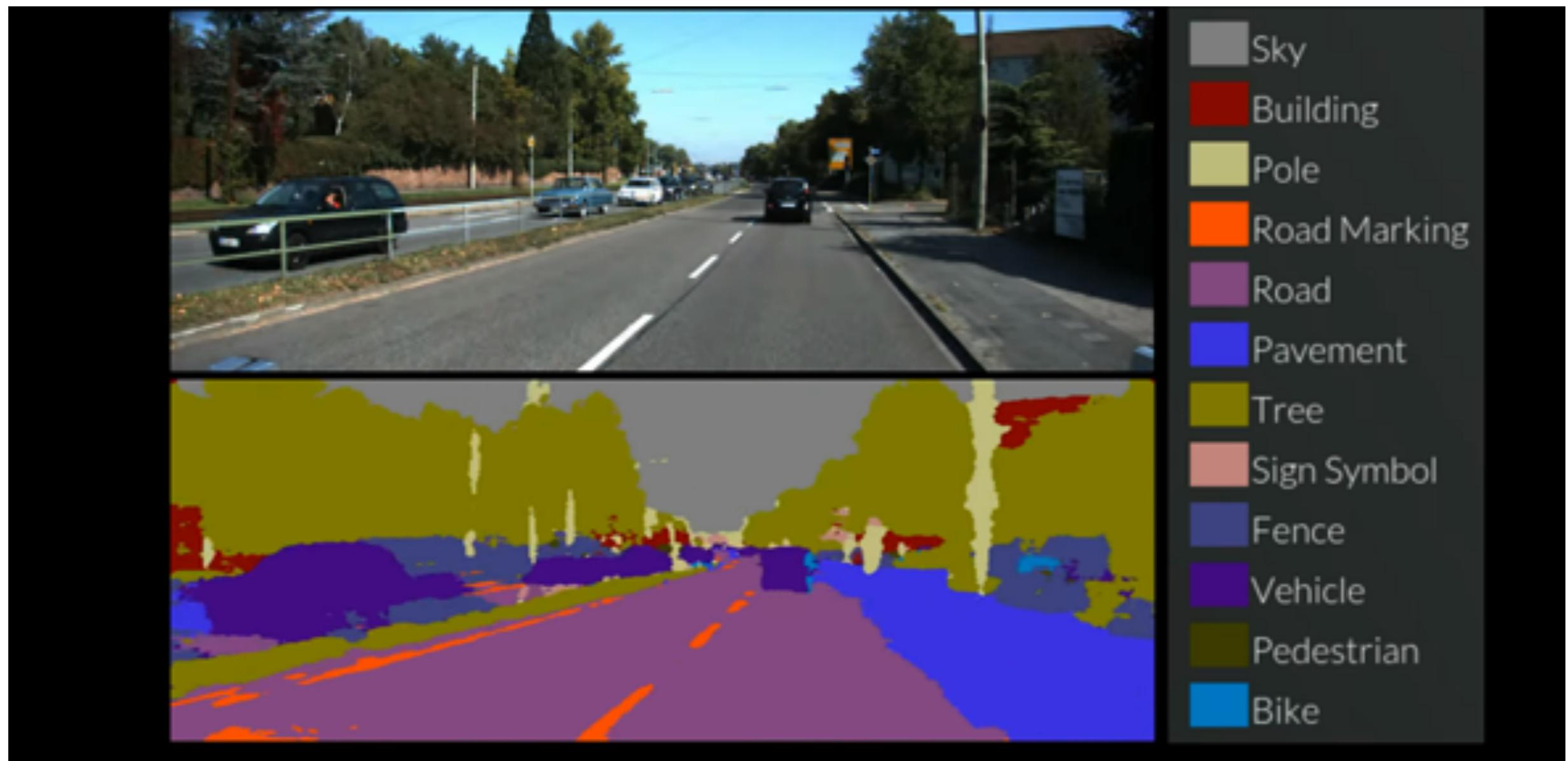
Image classification

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 8 9 9 9 9 9 9 9

Object detection



Semantic segmentation



Let's get our hands dirty
with some **TensorFlow**