DEEP LEARNING 2: CONVOLUTIONS

Keith Butler

CONVOLUTIONAL NEURAL NETS: THE POWER OF INDUCTIVE BIAS

The Need for Biases in Learning Generalizations

Tom M. Mitchell

The **inductive bias** (also known as **learning bias**) of a learning algorithm is the set of assumptions that the learner uses to predict outputs of given inputs that it has not encountered.

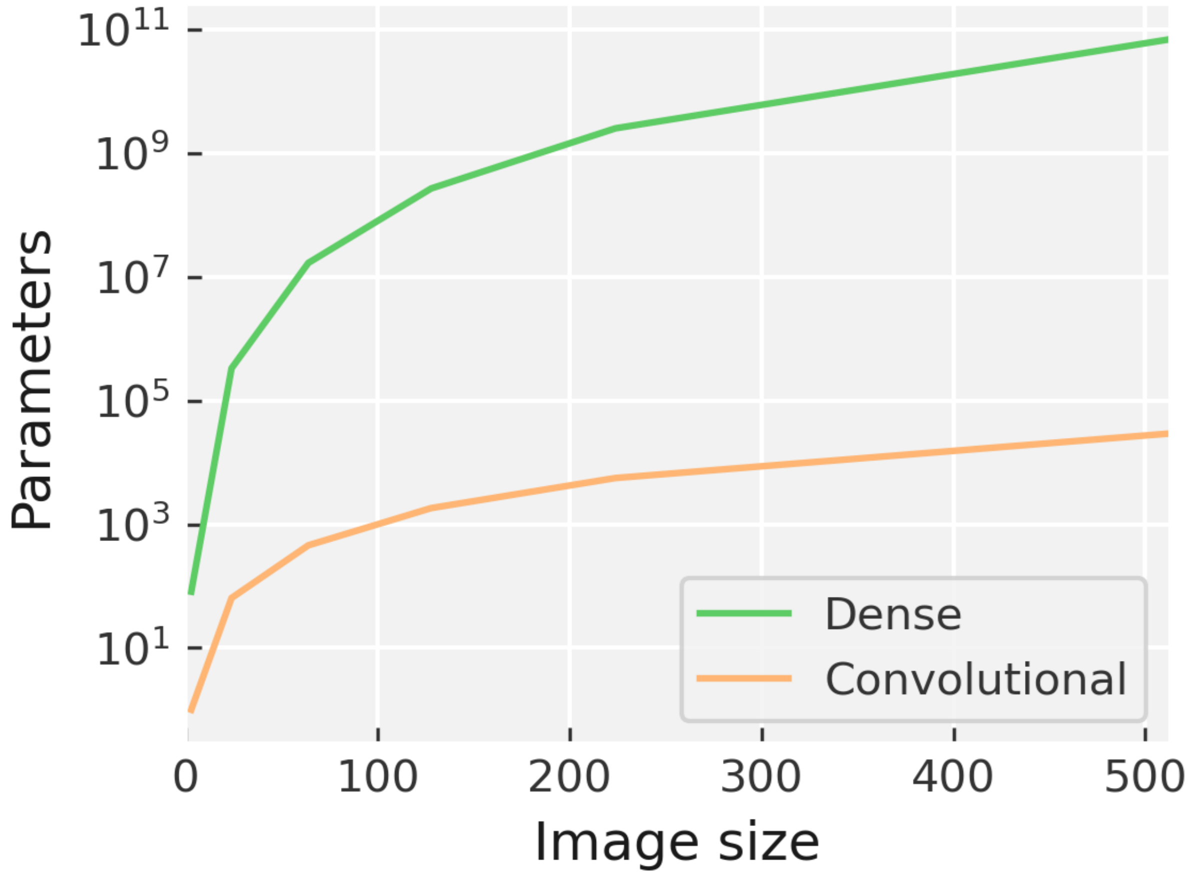
The need for biases in learning generalizations, CBM-TR 5-110, New Brunswick, New Jersey, USA: Rutgers University

OVERVIEW

- Intro to convolutional neural networks
- Building blocks of CNNs
- Deep CNNs
- Advanced CNNs – Residual blocks

DRAWBACKS OF MLPS

MLPs have **no spatial awareness** and also suffer from **parametric explosions** as the input gets larger



EARLY CNNS

LeCun – restricting the number of parameters in a NN leads to **better generalisation**

Generalization and Network Design Strategies

Y. le Cun
Department of Computer Science
University of Toronto

Technical Report CRG-TR-89-4
June 1989

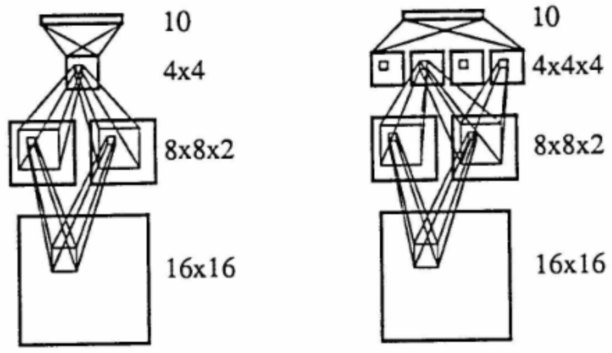
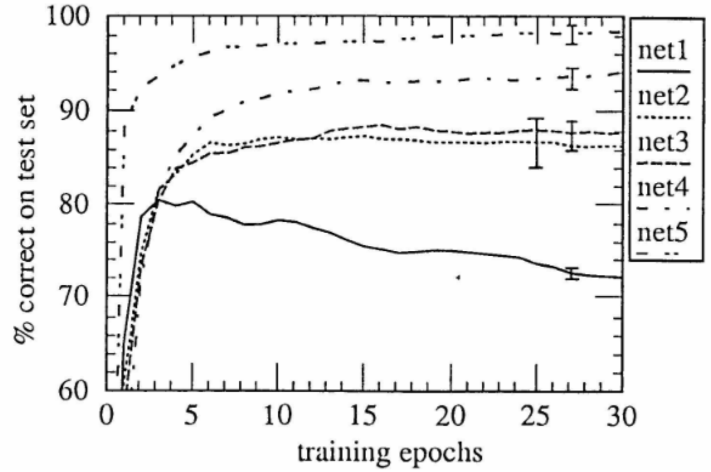


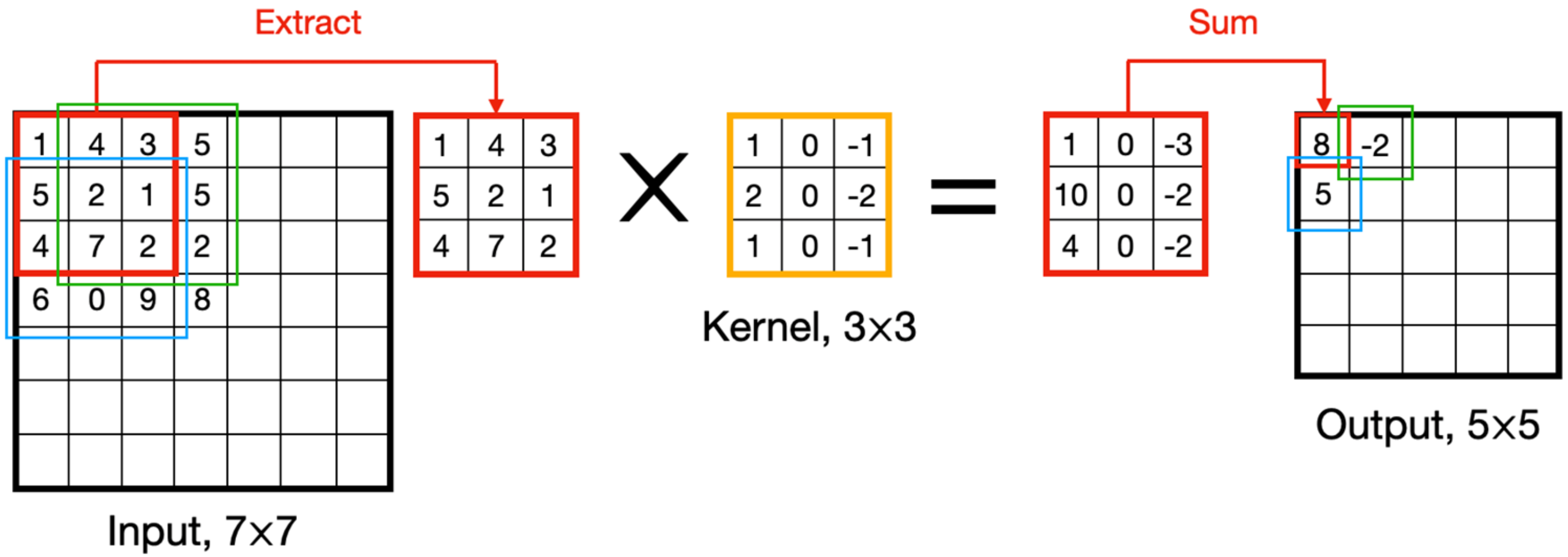
Figure 5 two network architectures with shared weights: Net-4 and Net-5



STRUCTURE OF A CONVOLUTIONAL LAYER

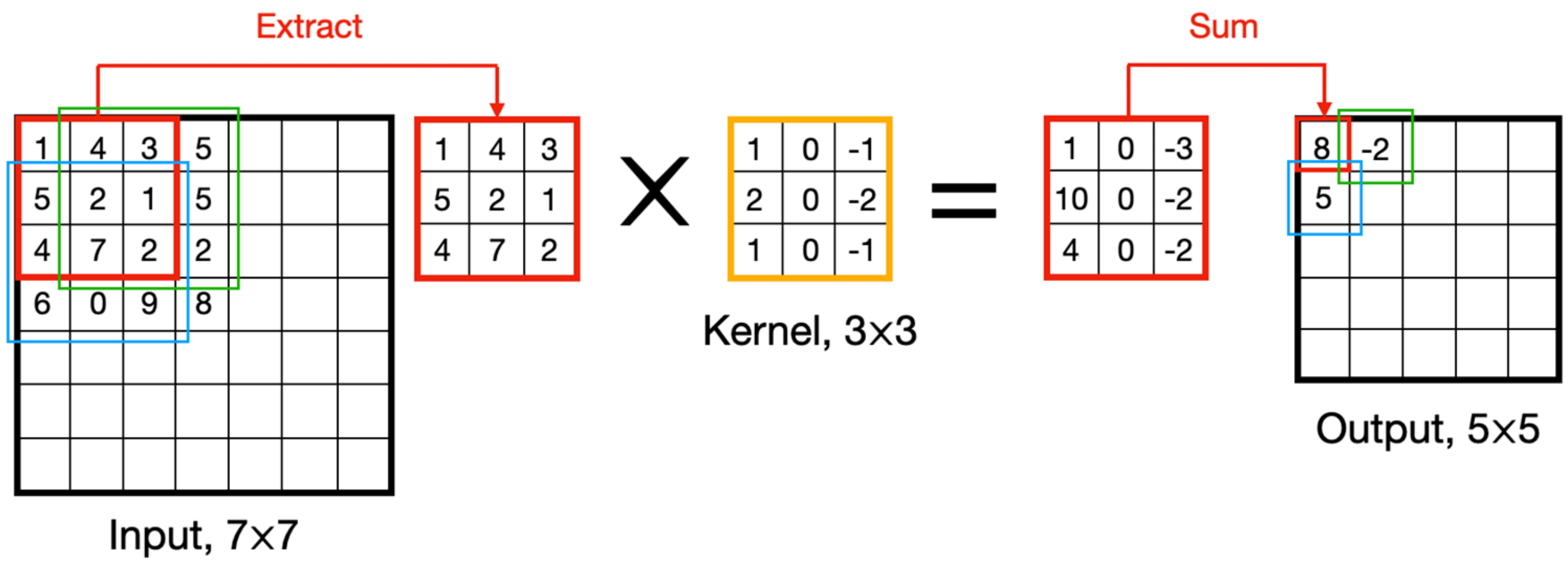
Typical convolutional layers have three main ingredients:

- Kernel
- Pooling
- Activation



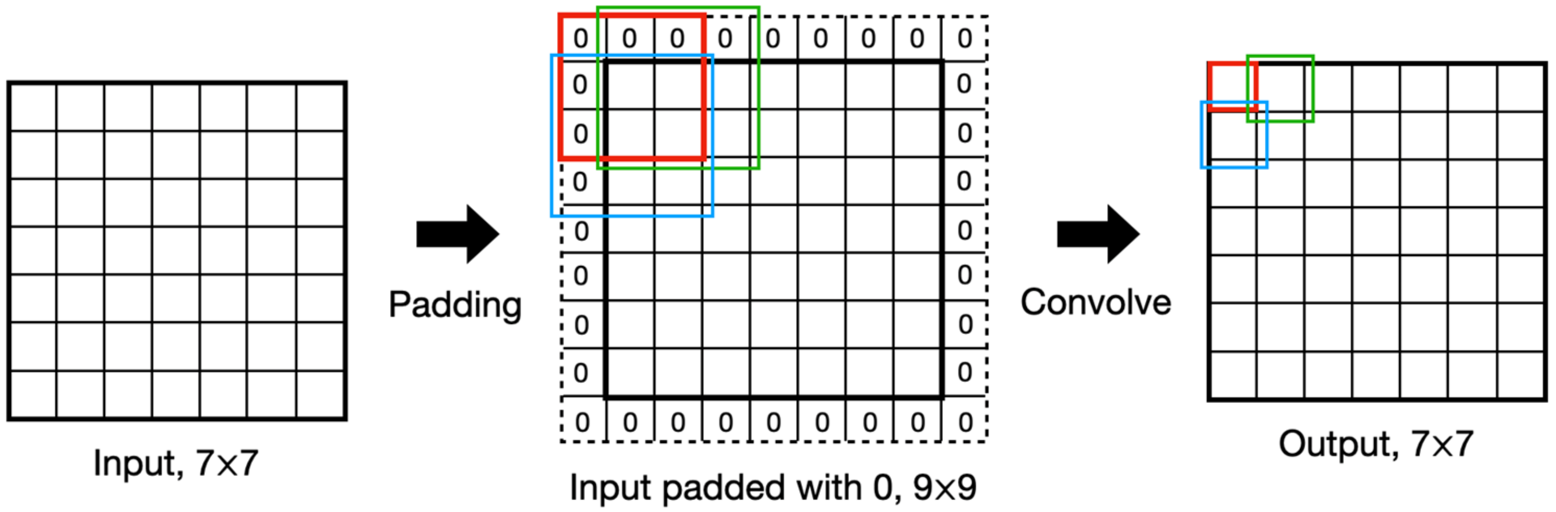
CONVOLUTION IN ACTION: KERNEL

- Input + kernel -> activation map



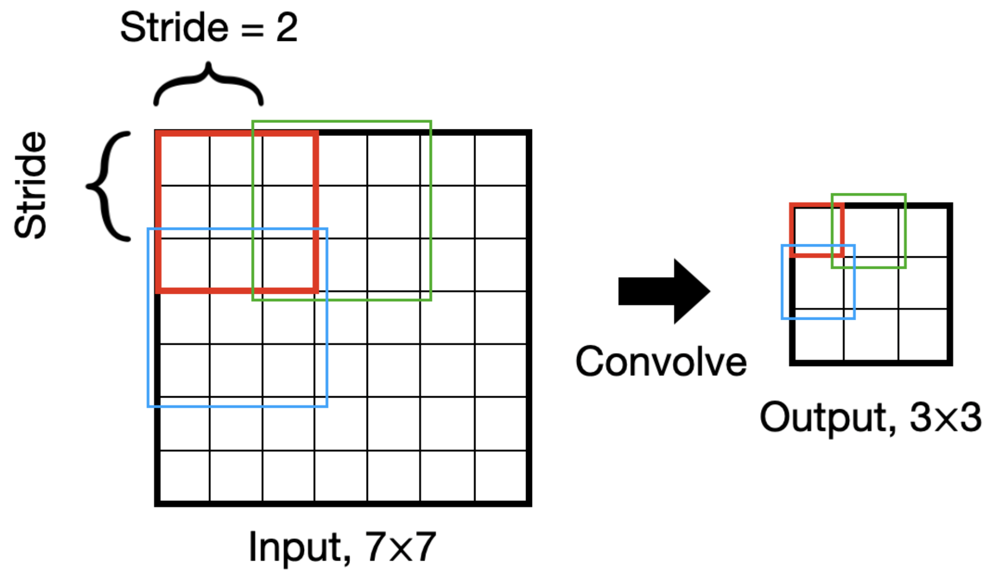
CONVOLUTION IN ACTION: PADDING

- Padding around the outside of images
  - Zero pad: pad with zeros to make `torch.nn.ZeroPad2d(padding)`
  - No padding `output.shape < input.shape`



CONVOLUTION IN ACTION: STRIDING

Controls how the filter slides across the image



$$\text{output width} = \frac{W - F_w + 2P}{S_w} + 1$$

$$\text{output height} = \frac{H - F_h + 2P}{S_h} + 1$$

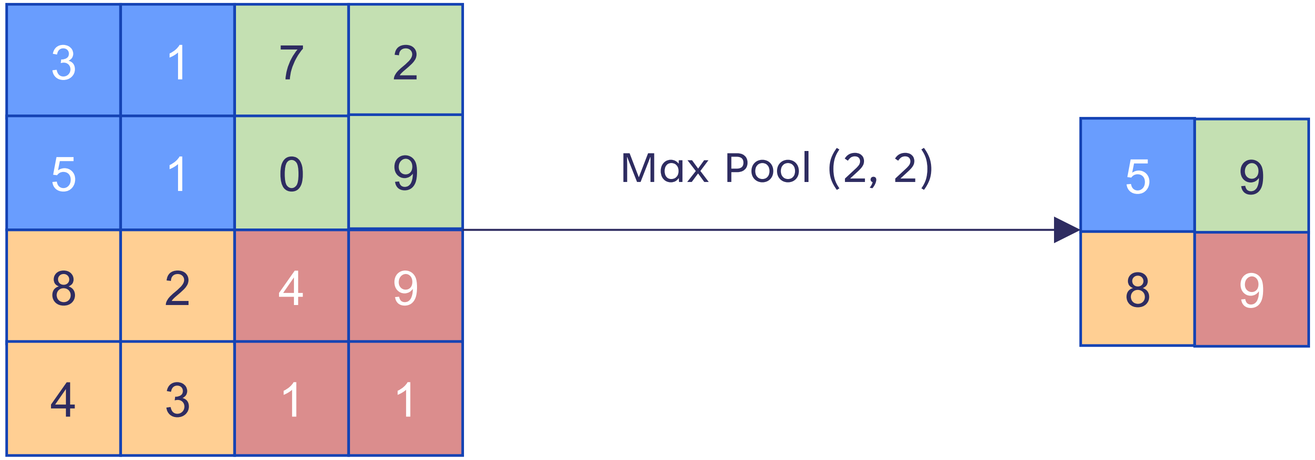
GO TO NOTEBOOK

Let's try building and understanding some filters

```python
# a 2D convolutonal filter
def convolve2D(input_image, kernel, padding=1, stride=1):
    # padding
    nx = input_image.shape[0]
    ny = input_image.shape[1]
    nchannel = input_image.shape[2]
    if padding > 0:
        padded_image = np.zeros((nx + padding * 2, ny + padding * 2, nchannel))
        padded_image[padding:-padding, padding:-padding, :] = input_image
    else:
        padded_image = input_image

    # allocate output
    k = kernel.shape[0]
    nx_out = (nx + padding * 2 - k) // stride + 1 # must use // instead of /
    ny_out = (ny + padding * 2 - k) // stride + 1
    output_image = np.zeros((nx_out, ny_out, nchannel))

    # compute output pixel by pixel
    for ix_out in np.arange(nx_out):
        for iy_out in np.arange(ny_out):
            ix_in = ix_out * stride
            iy_in = iy_out * stride
            # the inner product
            output_image[ix_out, iy_out, :] = \
            np.tensordot(kernel, padded_image[ix_in:(ix_in + k), iy_in:(iy_in + k), :], axes=2)

    # truncate to [0, 1]
    output_image = np.maximum(output_image, 0)
    output_image = np.minimum(output_image, 1)
    return output_image
```

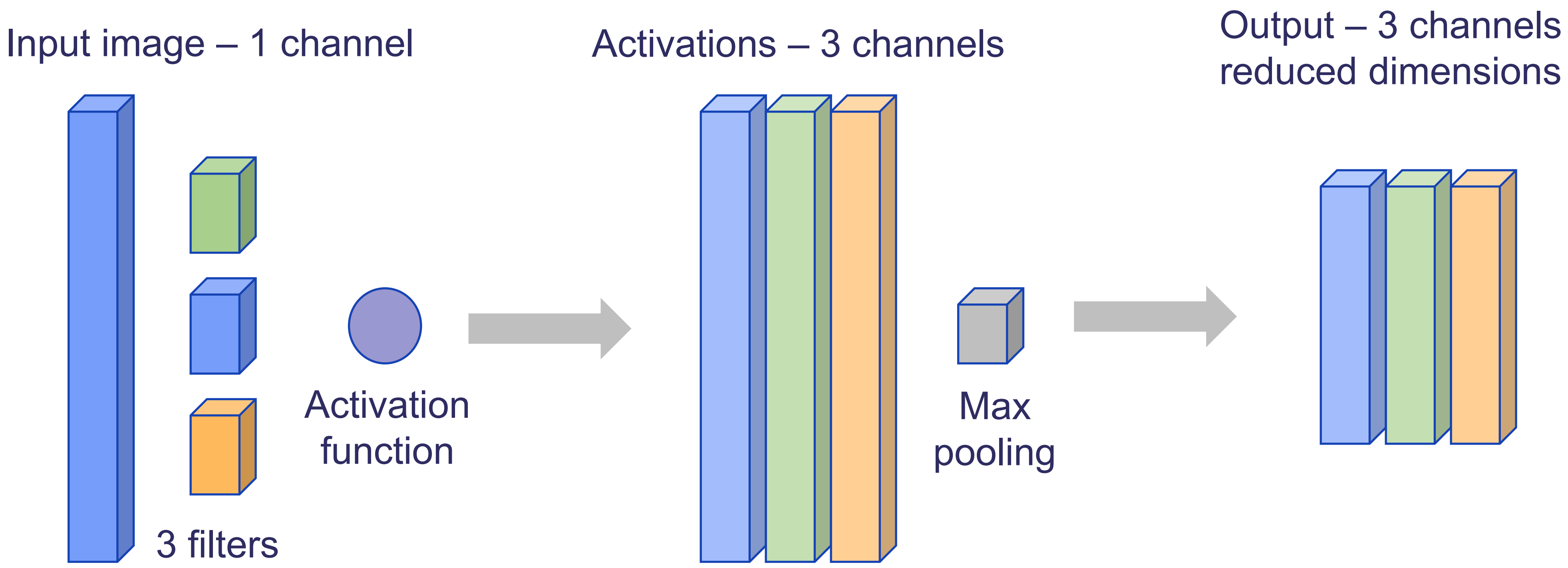
CONVOLUTION IN ACTION: POOLING

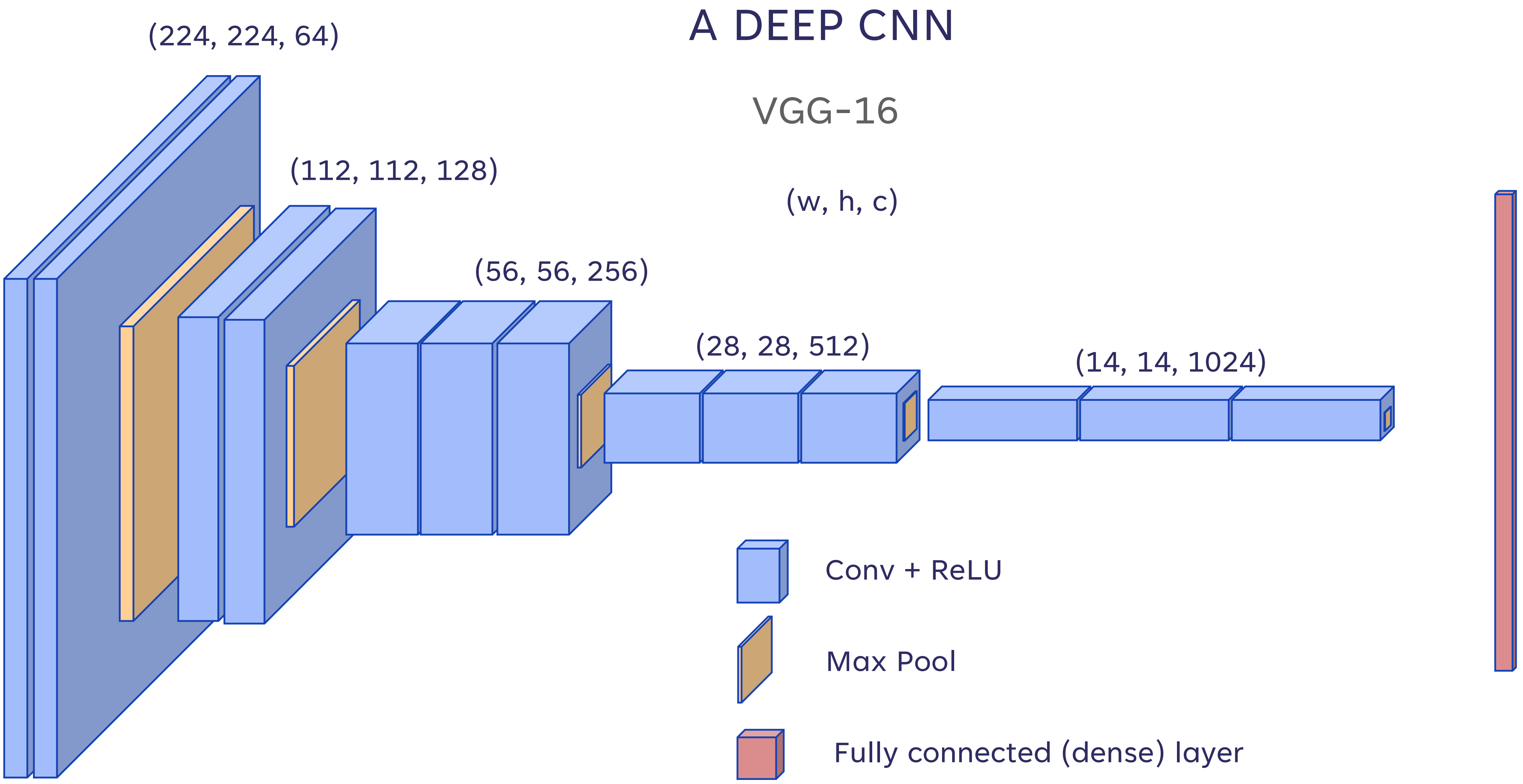
Pooling **compresses information** content between layers



The most commonly used pooling is choosing the maximum value patchwise; **max pooling**

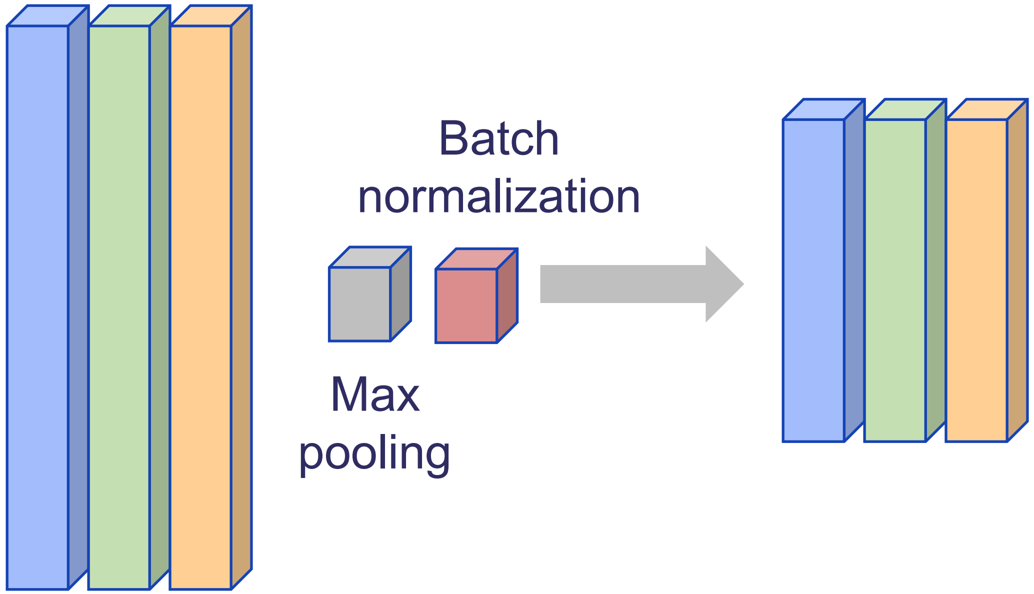
CONVOLUTION IN ACTION: PUTTING IT TOGETHER




A DEEP CNN
VGG-16
(224, 224, 64)
(112, 112, 128)
(w, h, c)
(56, 56, 256)
(28, 28, 512)
(14, 14, 1024)
Conv + ReLU
Max Pool
Fully connected (dense) layer

BATCH NORMALISATION

Normalise the outputs from intermediate layers



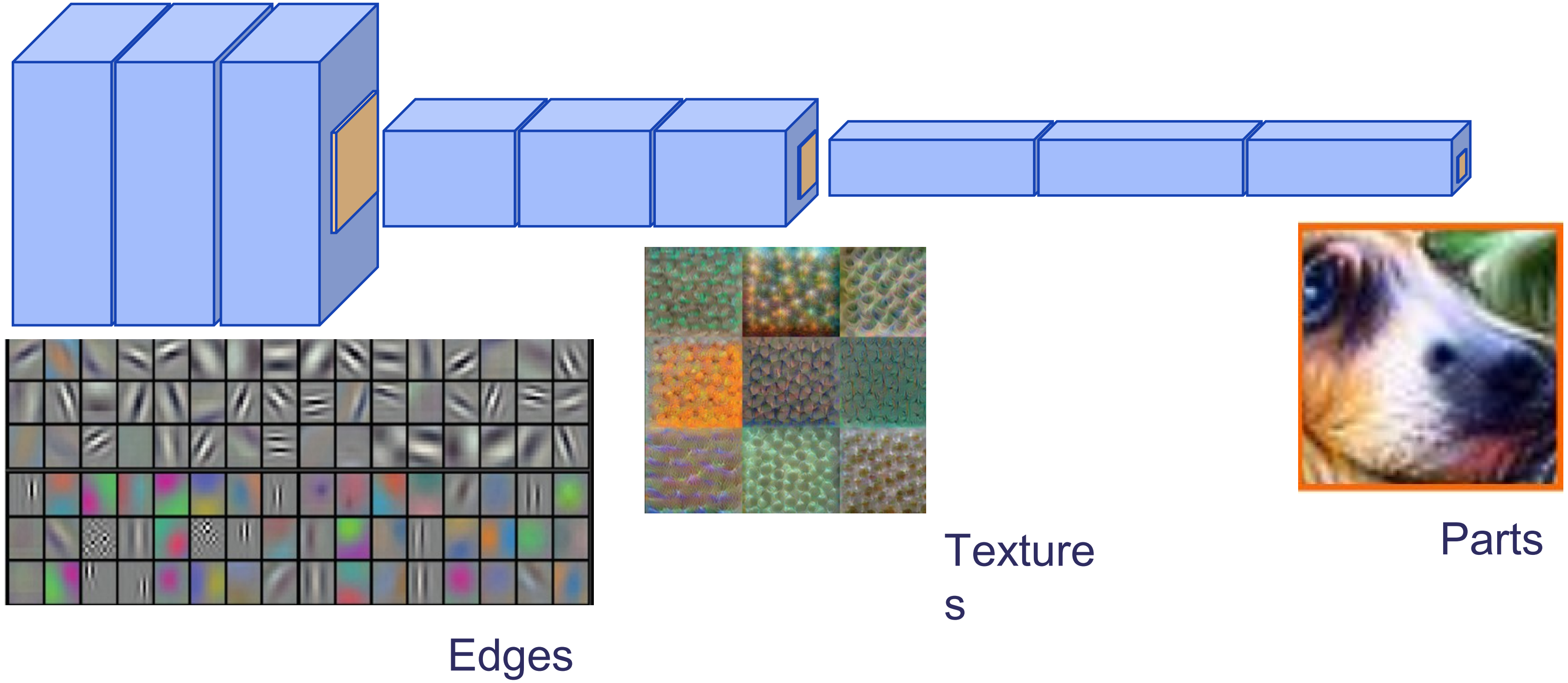
Makes weights deep in the NN more robust to changes early in the NN

BUILDING BLOCKS: CONVOLUTION BLOCK

```
import torch
import torch.nn as nn
import torch.nn.functional as F

nn.Conv2d(in_channels=1, out_channels=6,kernel_size=5)
F.max_pool2d(x, kernel_size=2)
```

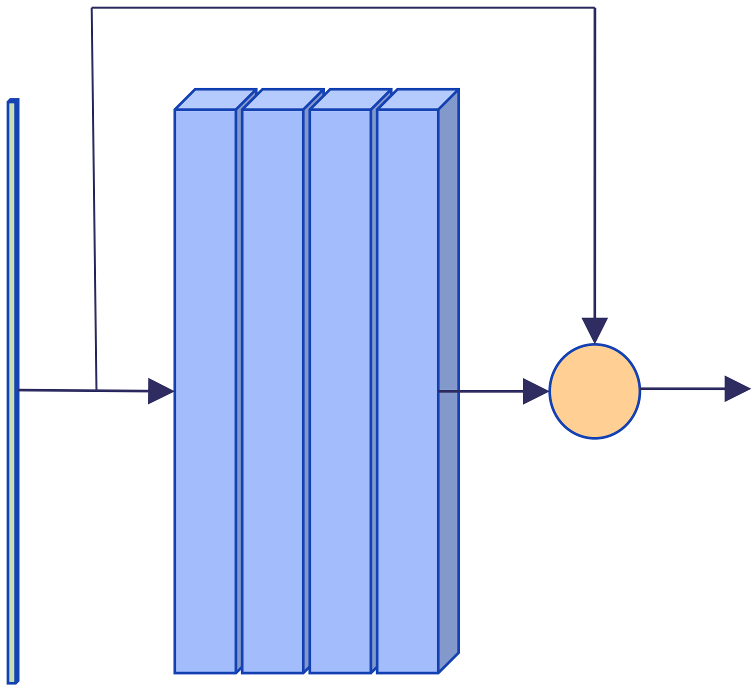
Hierarchy of filters

- Stacking deep networks means that different levels of features are learned at different depths



Advanced CNNs: Residual blocks

- A connection that passes the input over a block of convolutions
- Useful in very deep architectures
- Allows network to learn to skip blocks
- Allows gradient to pass back through the network more effectively in backprop



CONCEPT CHECKLIST

Origins of convolutional neural networks

Building blocks of CNNs – kernel, padding, stride

Max pooling

Deep CNNs

Batch normalisation

Feature detection in different layers

Residual blocks

THANK YOU

mdi-group.github.com