Machine Learning and Artificial Intelligence

Korbinian Kottmann
ICFO – Institute for Photonic Science
Barcelona, Spain

/Qottmann

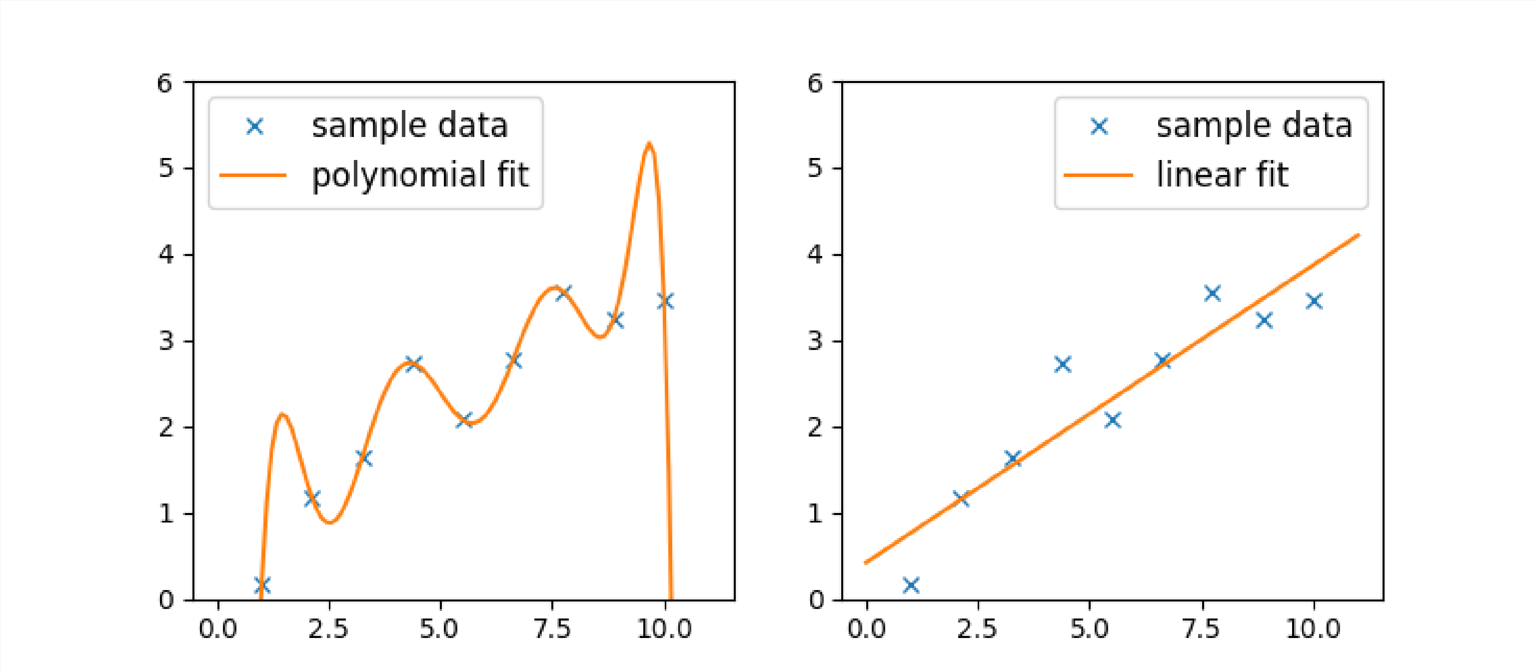
@Qottmann

ICFO

UNIÓ EUROPEA
Fons Europeu
de Desenvolupament Regional

GOBIERNO DE ESPAÑA
MINISTERIO DE ECONOMÍA Y COMPETITIVIDAD

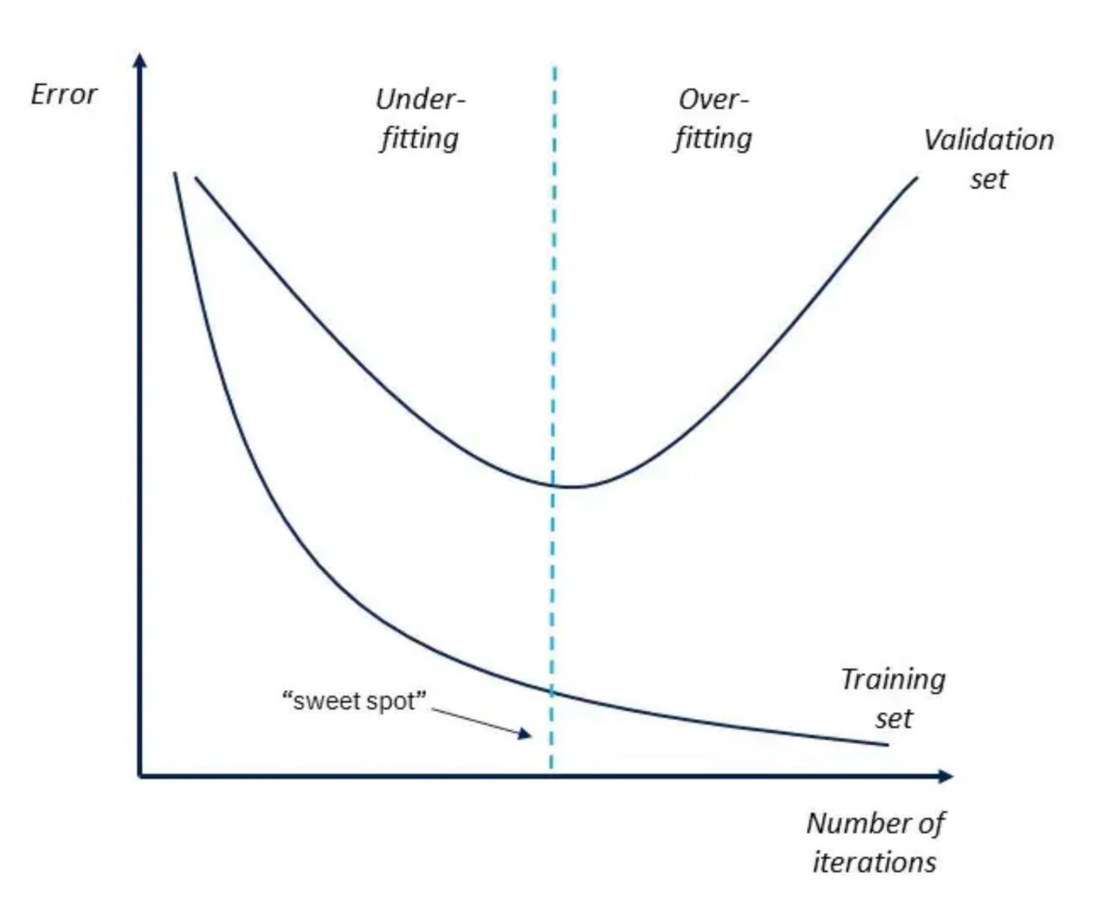
Overfitting



0 loss, poor generalization

Good generalization

Overfitting



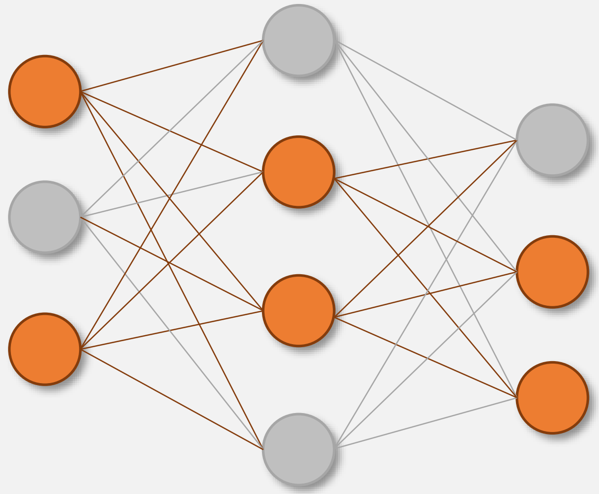
Overfitting

Remedies

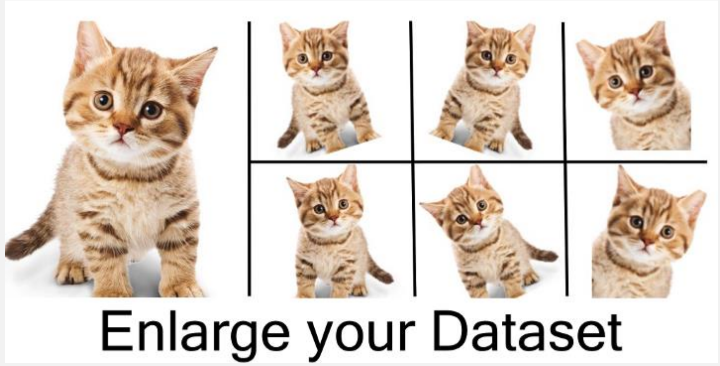
Regularization

$$L \mapsto L + \lambda \sum_{ij\ell} |\omega_{ij}^{\ell}|$$

Dropout



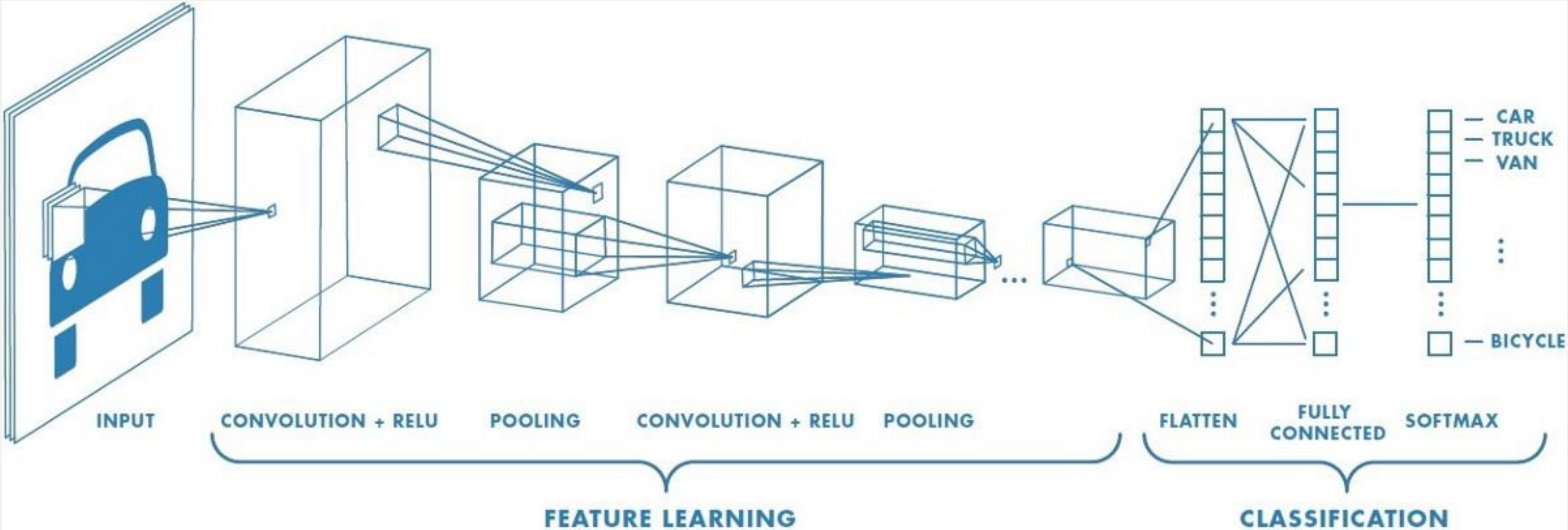
Data augmentation



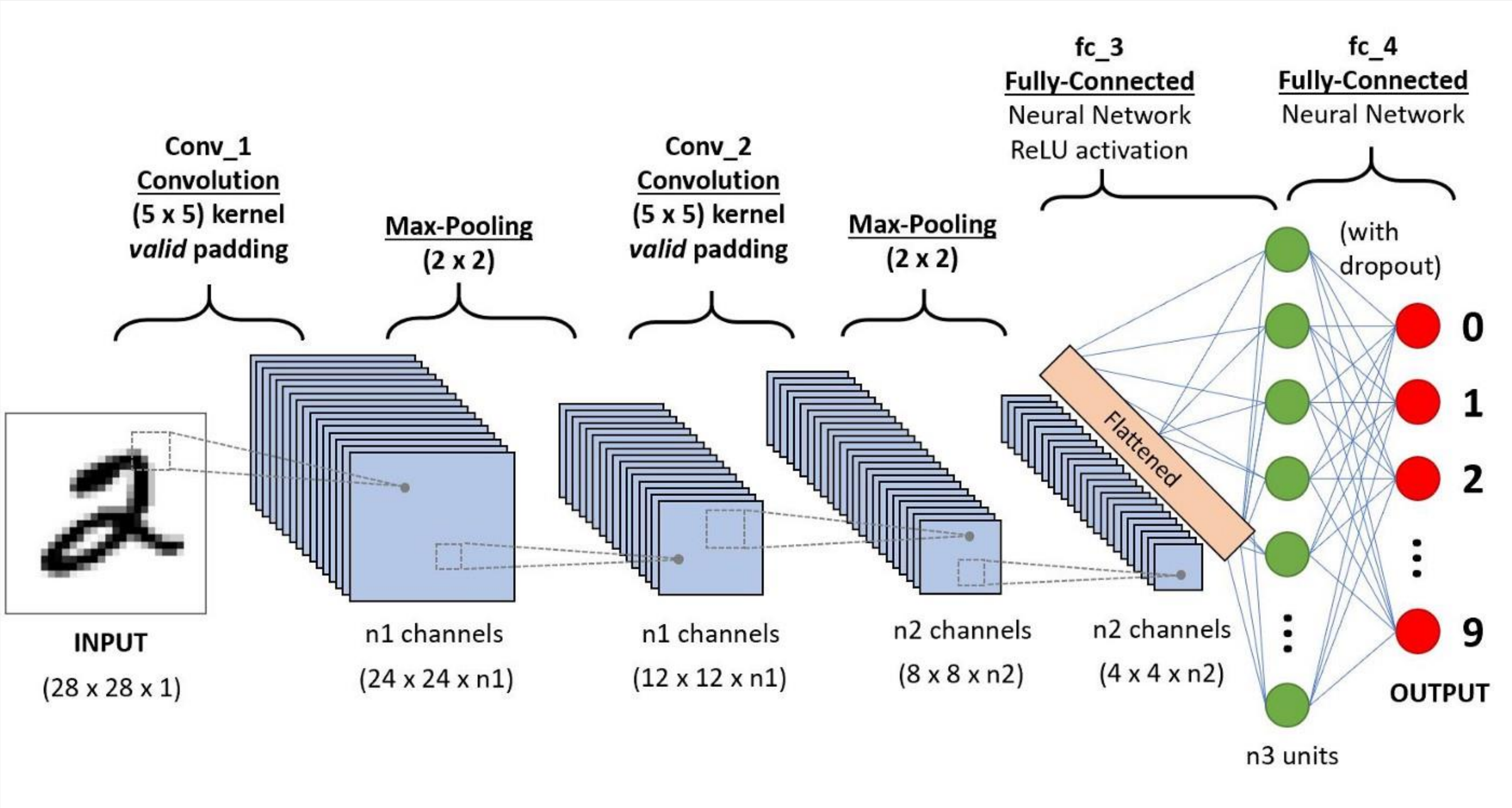
Continue with notebook 14_overfitting.ipynb

Part II: Convolutional Layers

Convolutional Layers

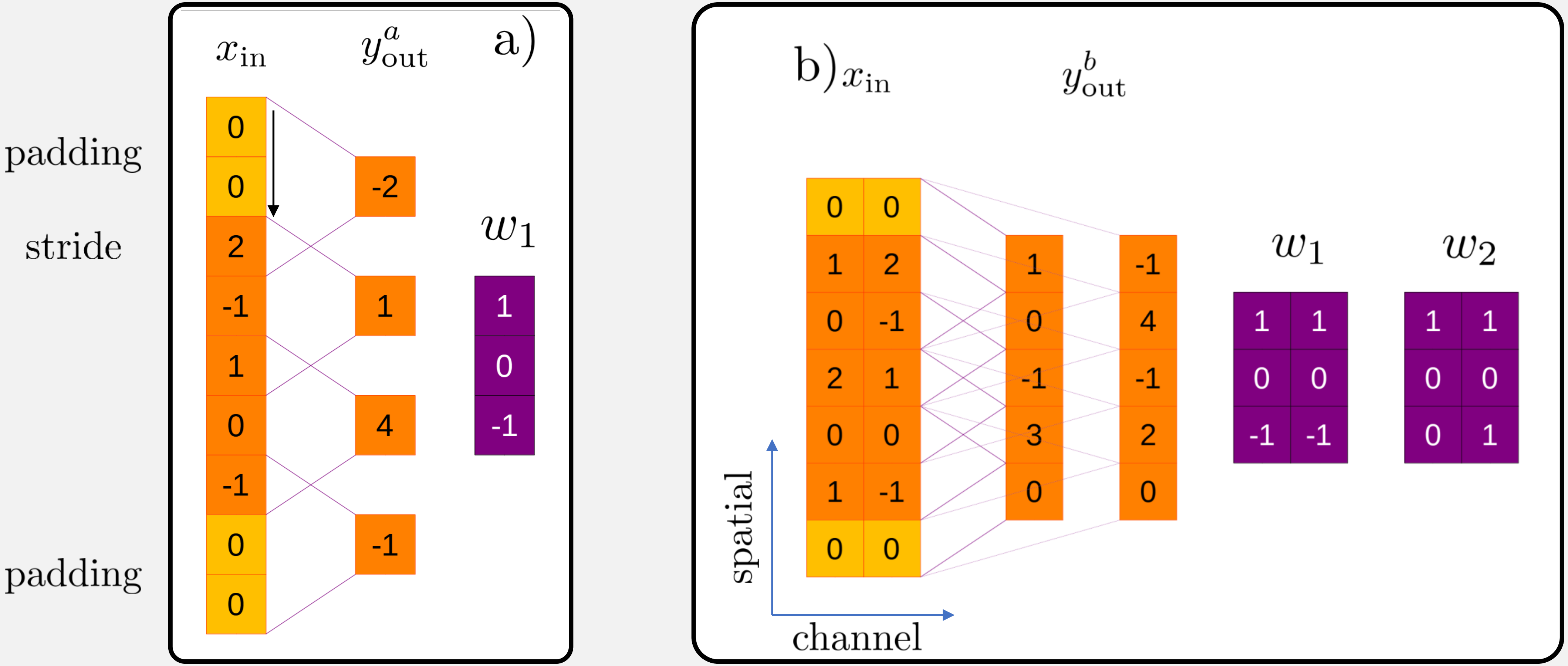


Convolutional Layers



Sumit Saha

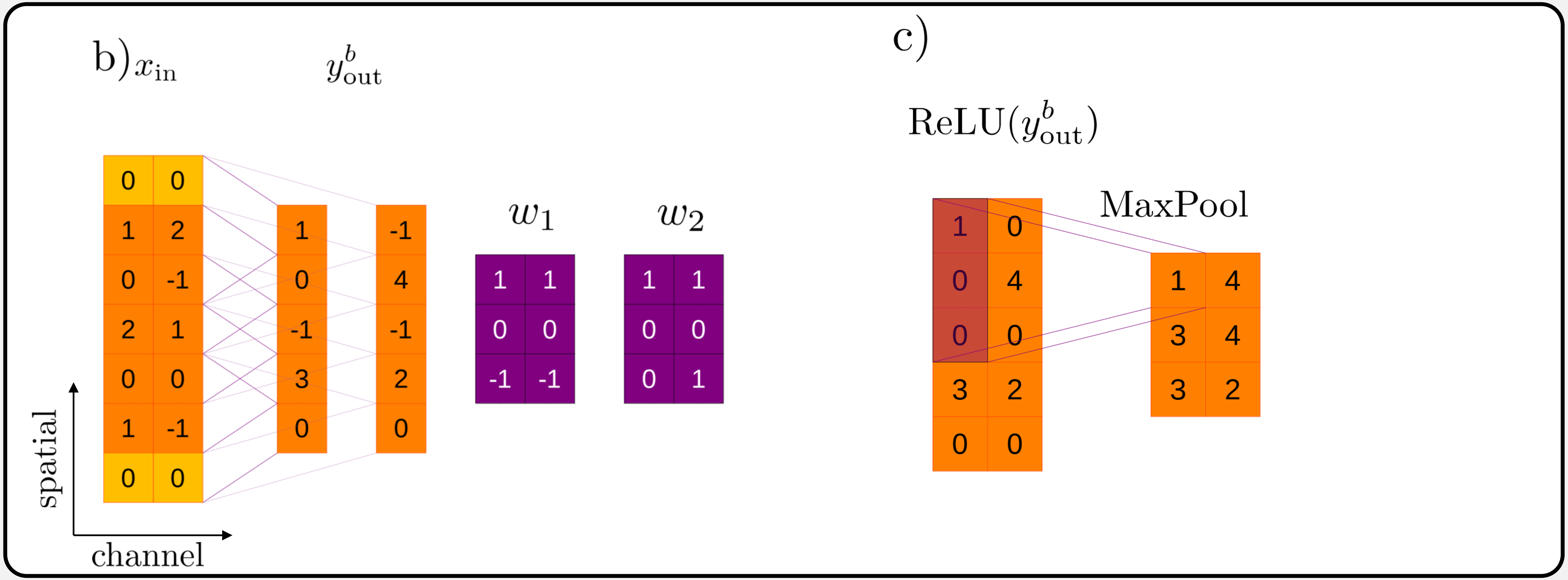
Convolutional Layers



$$|y| = (|x| - |w| + 2\text{pad})/\text{stride} + 1$$

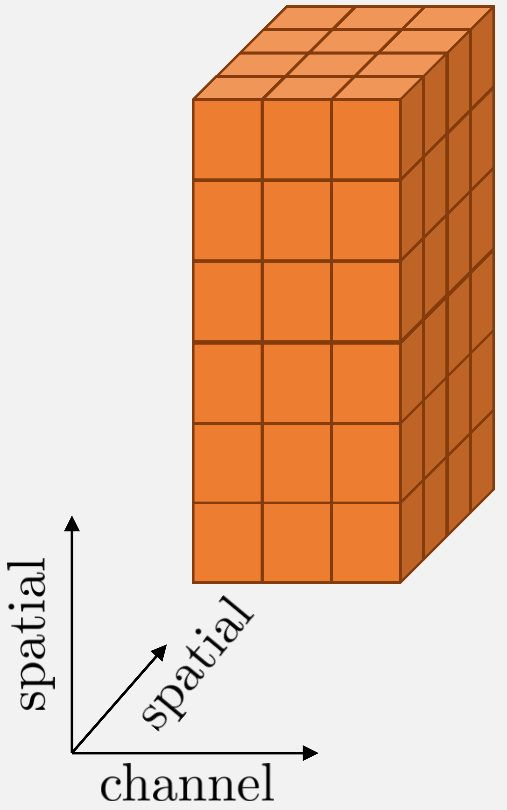
$$(5 - 3 + 2*2)/2 + 1 = 4$$

Convolutional Layers

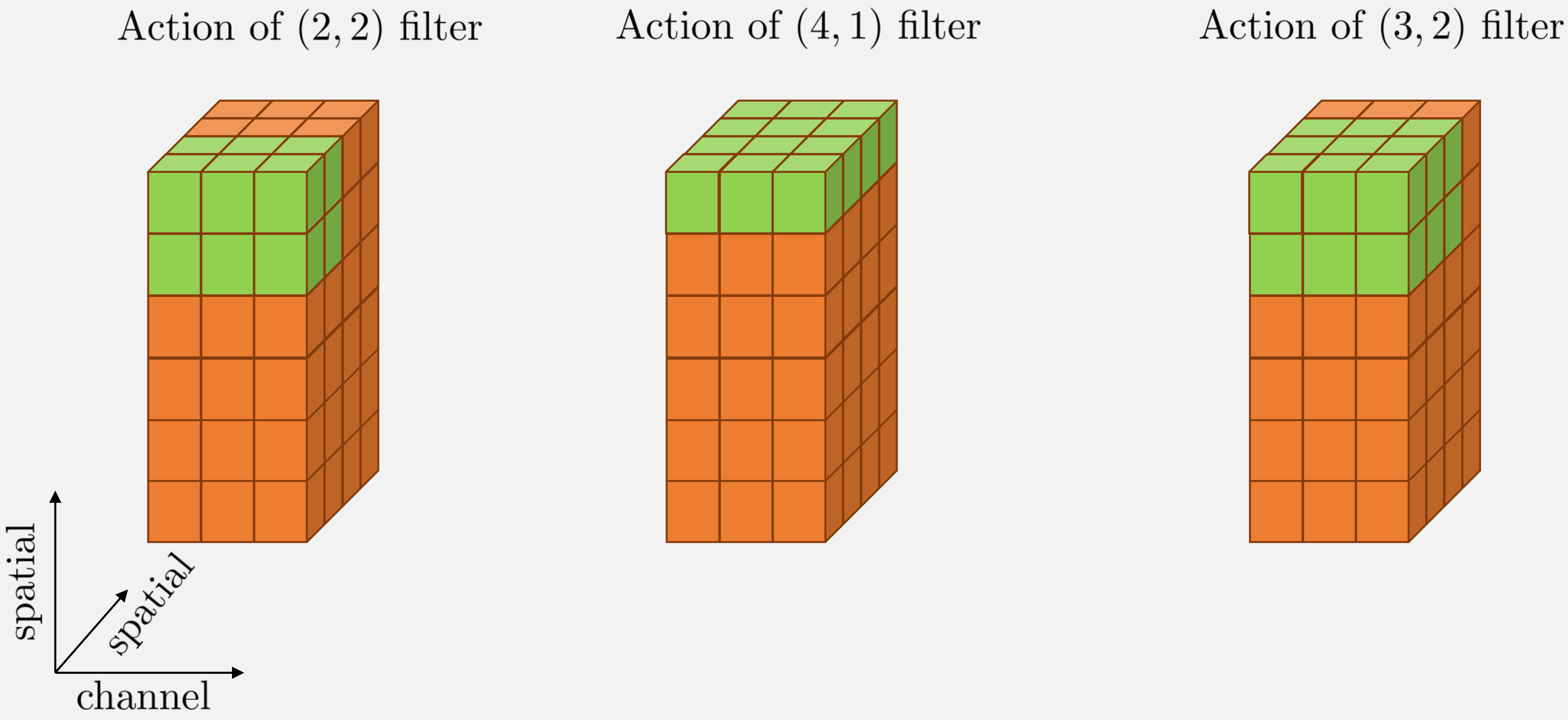


2D Convolutional Layers

Define (d_1, d_2) and n_{filters}



2D Convolutional Layers



Further nice illustrations on padding, strides etc. https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md

Ex: Given color images of 32x32 (|x| = (32, 32, 3)) and the following convolutional neural network, what is are the input and output dimensions for each layer?

```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.flatten = nn.Flatten()
        self.conv_block = nn.Sequential(
            nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3, padding="same"), # (28 x 28 x 1) -> (28 x 28 x 16)
            nn.ReLU(),
            nn.MaxPool2d(2),                         # (28 x 28 x 16) -> (14 x 14 x 16)
            nn.Conv2d(16, 16, 3, padding="same"),    # (14 x 14 x 16) -> (14 x 14 x 16)
            nn.ReLU(),
            nn.MaxPool2d(2),                         # (14 x 14 x 16) -> (7  x 7  x 16)
            nn.Conv2d(16, 16, 3, padding="same"),    # (7  x 7  x 16) -> (7  x 7  x 16)
            nn.ReLU(),
            nn.Flatten())                            # (7  x 7  x 16) ->  (7*7*16)
        self.dense_block = nn.Sequential(
            nn.Linear(7*7*16, 512),  # input dimension, hidden1 dimension
            nn.ReLU(),               # Non-linear activation function
            nn.Linear(512, 512),     # hidden1 dimension, hidden2 dimension
            nn.ReLU(),               # Non-linear activation function
            nn.Linear(512, out_dim), # hidden2 dimension, output dimension
        )

    def forward(self, x):
```

Exercises

- Ex: Write (by hand using e.g. numpy) a fully connected layer
- Ex: Connect multiple fully connected layers by hand.
- Ex: Write (by hand using e.g. numpy) a 1D convolution function conv1D(x, filter, stride, padding) to process an input vector x (of arbitrary size).
  The filter can be arbitrary but fixed. Try writing it such that it works for any stride, padding and filter size.
- Extra (difficult): Write all the above functions in such a way that they can process batches of inputs (so instead of a single input x, a collection of batch_size inputs x)