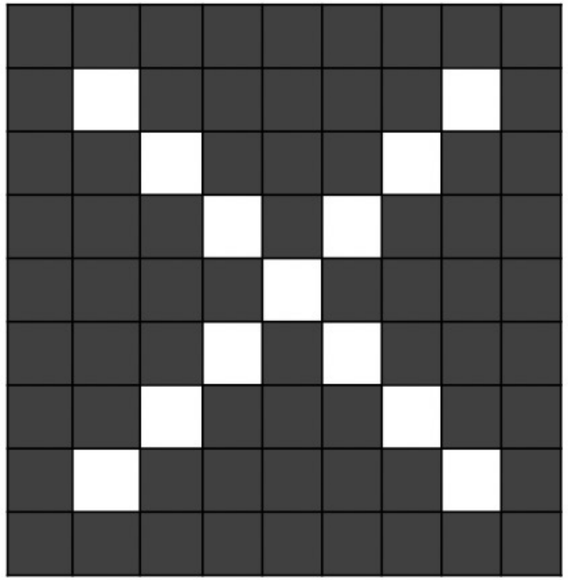
THE UNIVERSITY OF MELBOURNE

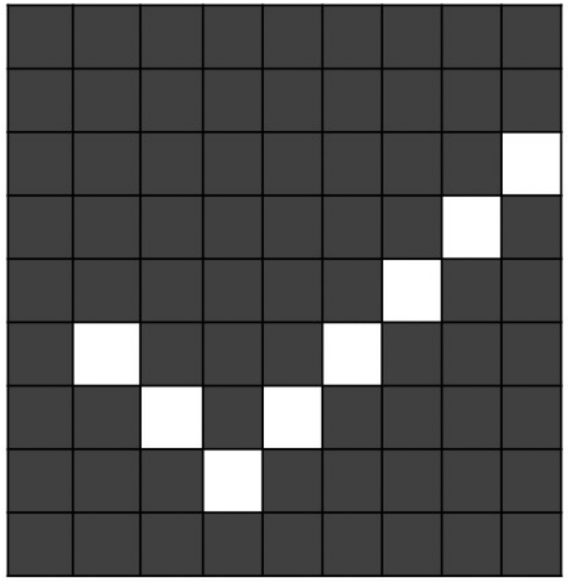
Convolutional Neural Networks

COMP90051 Statistical Machine Learning

Qiuhong Ke

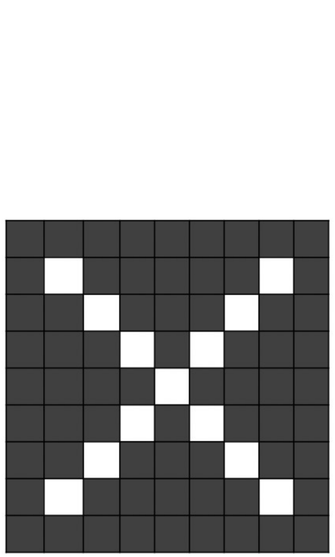
✗ vs ✓





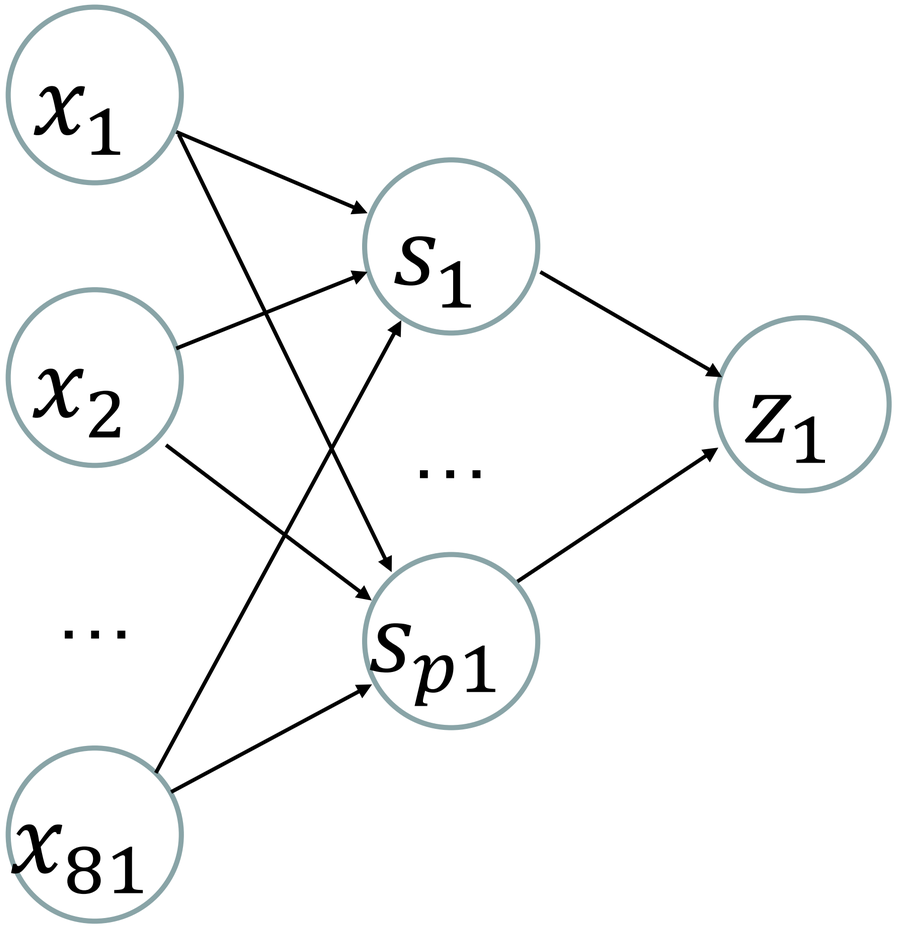
Multi-layer perceptron: A fully connected network

Consists of only fully connected (FC) layers



9x9

81x1



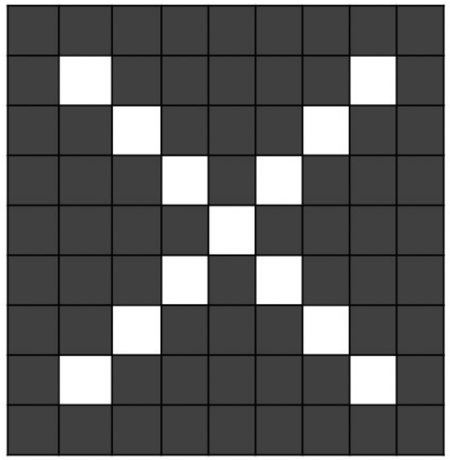
Input layer

Hidden layer

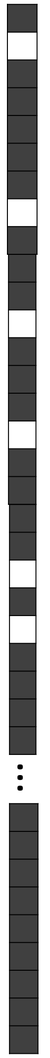
Output layer

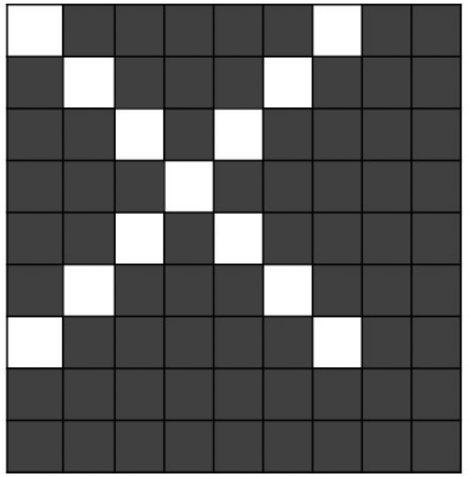
Multi-layer perceptron: A fully connected network

Disadvantage: Not spatial invariant



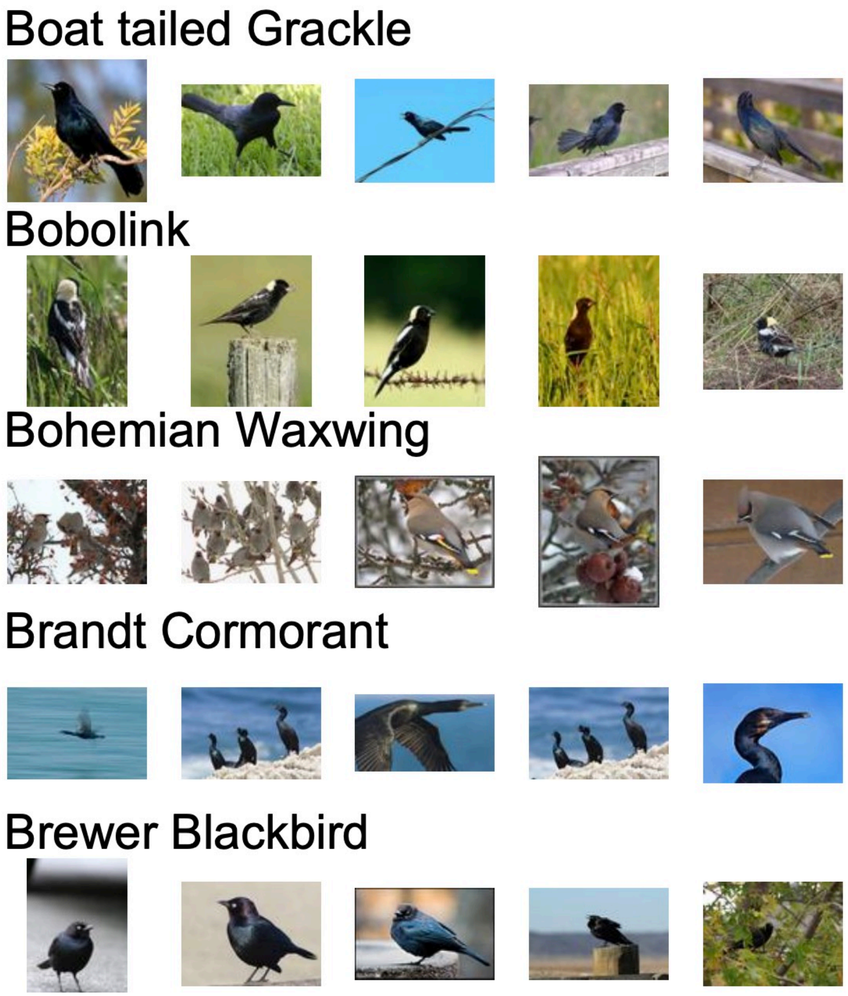
≠



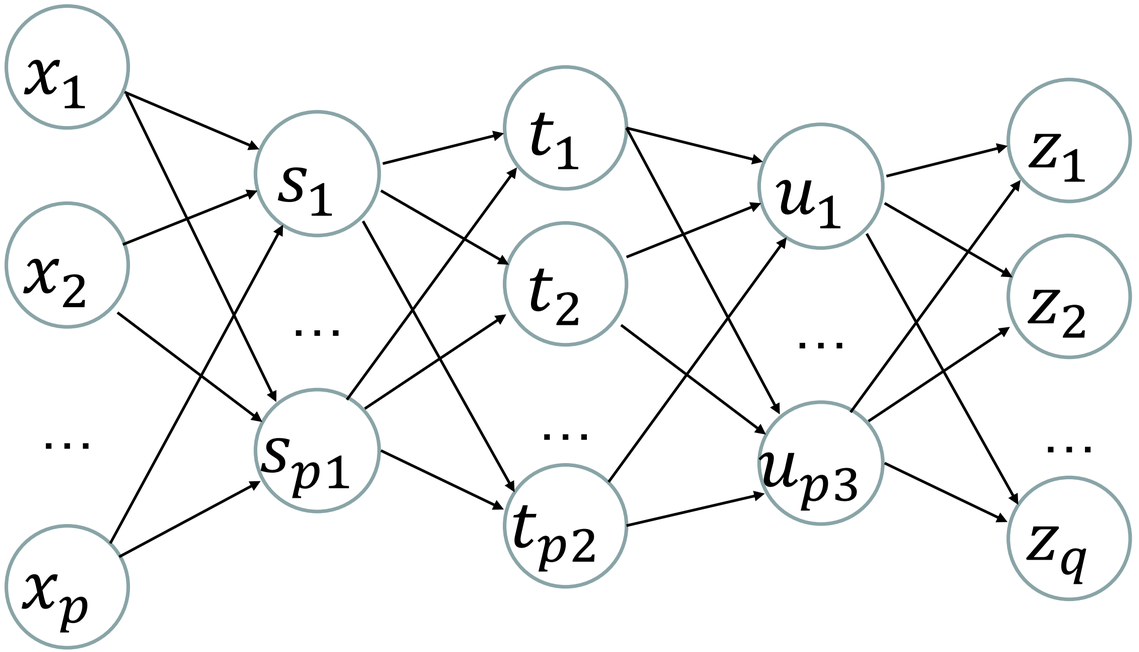


Multi-layer perceptron: A fully connected network

Disadvantage: more parameters with more hidden layers

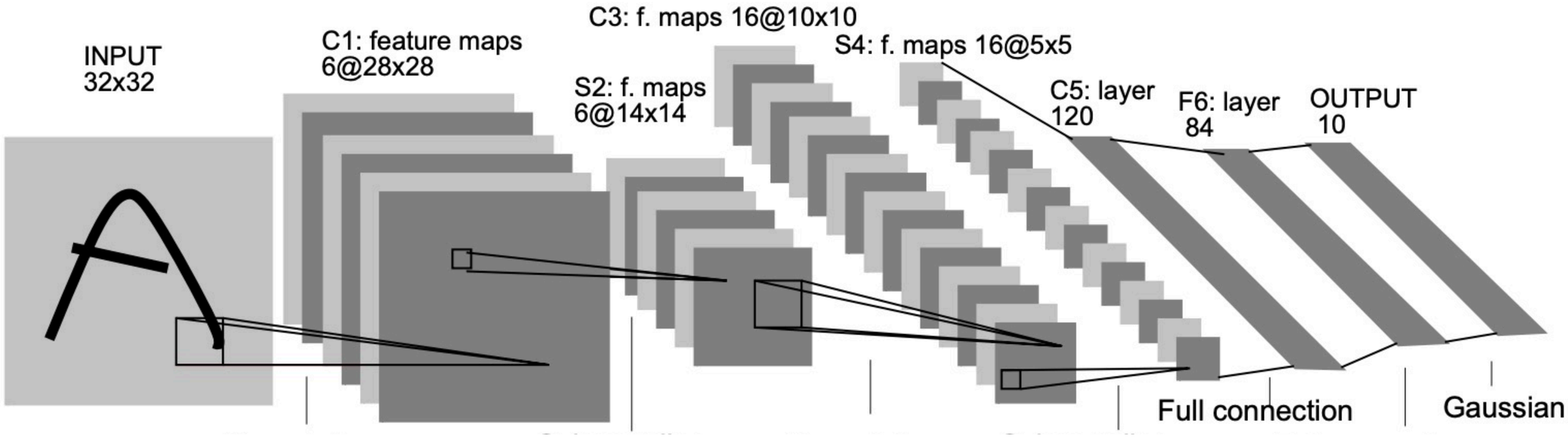


Source: Welinder, Peter, et al. "Caltech-UCSD birds 200." (2010).



Convolutional Neural Network (CNN)

Convolution, Max-Pooling, and Fully Connected (FC) layers



LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998): 2278-2324.

Outline

- Convolutional layer
- Max-Pooling layer
- Additional notes in training neural network
  - Batch size
  - Optimisation algorithms
  - Activation function
  - How to prevent overfitting

Tool: Keras

Easy, simple and powerful

- Build the architecture (add layers from input to output. eg. FC layer, convolution layer…)

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)
```

Tool: Keras

Easy, simple and powerful

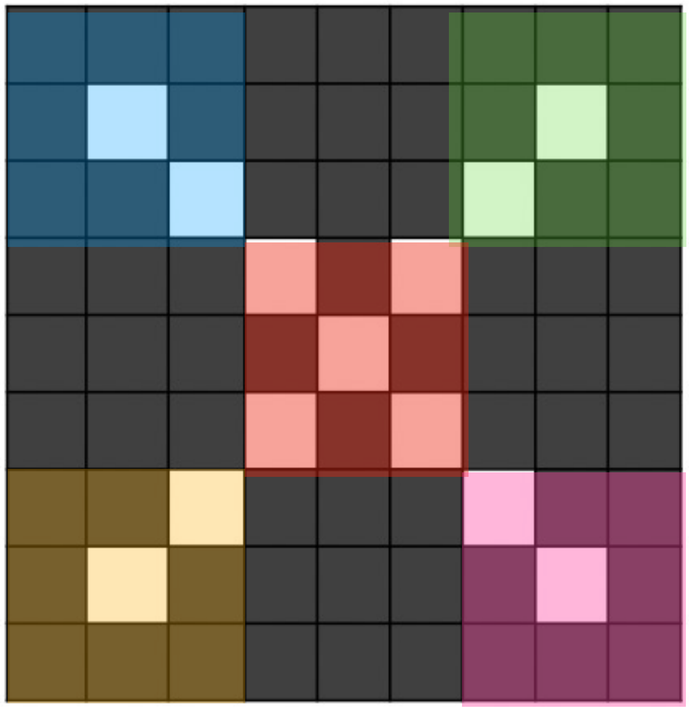
- Select an optimisation algorithm (eg. SGD, more in this lecture)
- Select the loss function
- Compile the model and train the model

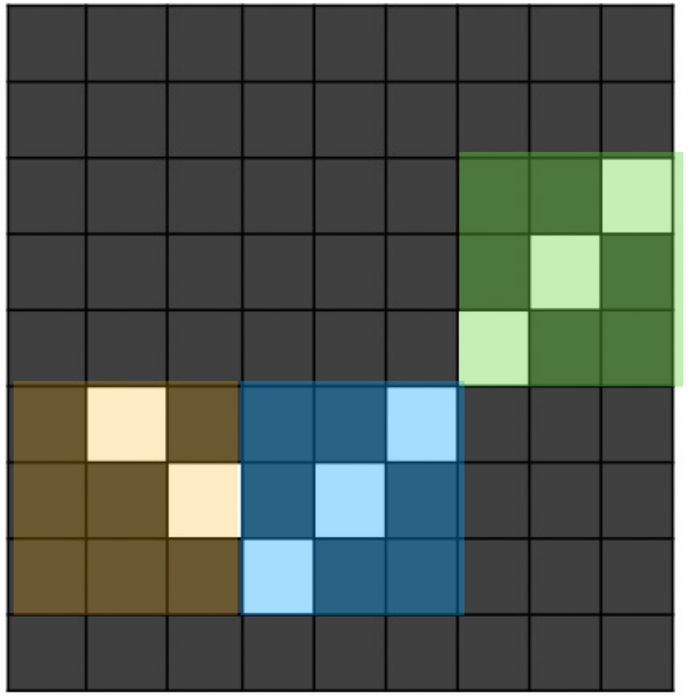
```
opt = tf.keras.optimizers.SGD(learning_rate=0.1)
model.compile(loss="categorical_crossentropy",
              optimizer=opt, metrics=["accuracy"])

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs)
```

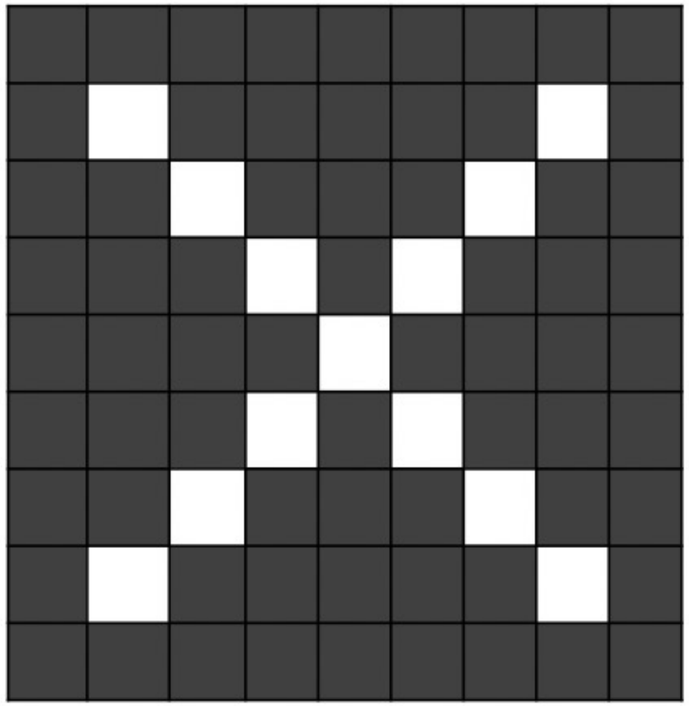
An image can be decomposed into local patches

- Different local patches could have different patterns
- To do classification, we can first extract local features(: Identify local patterns) and then combine the local features for classification



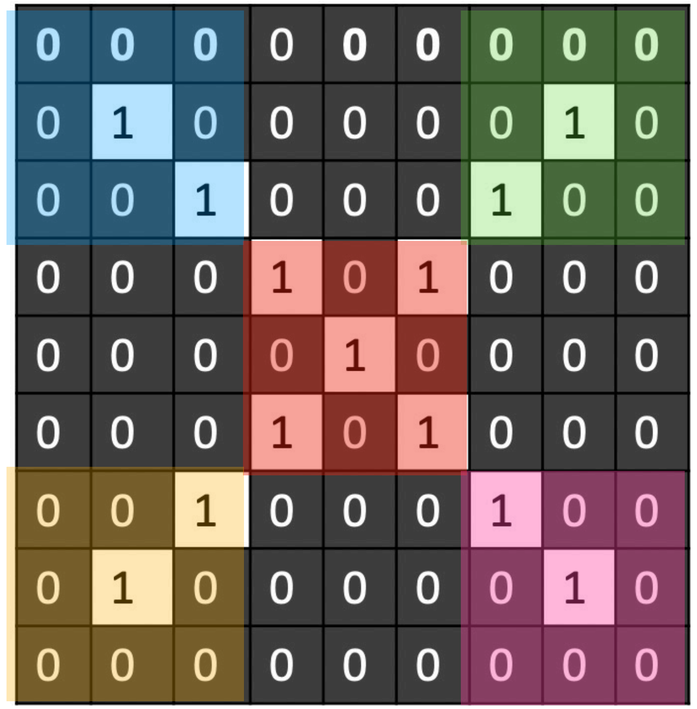


Identify different patterns at local patches

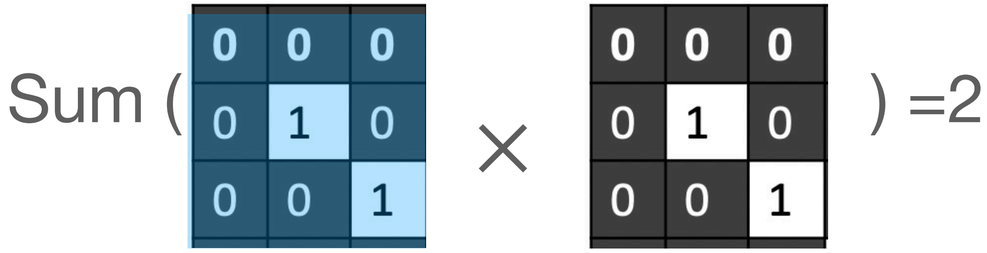


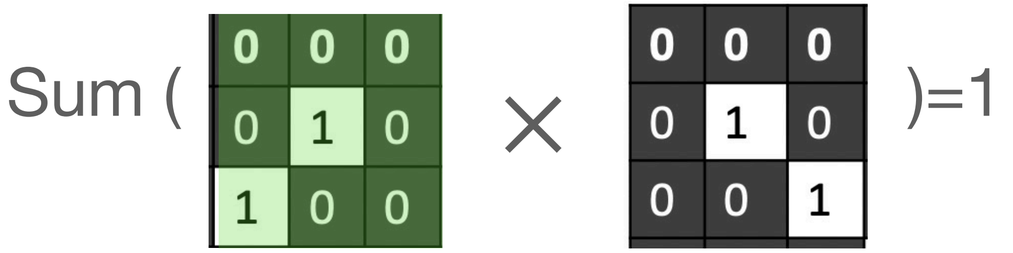
0	**0**	**0**	0	**0**	**0**	**0**	**0**	**0**
0	1	0	0	0	0	0	1	0
0	0	1	0	0	0	1	0	0
0	0	0	1	0	1	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	1	0	1	0	0	0
0	0	1	0	0	0	1	0	0
0	1	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0

Identify different patterns



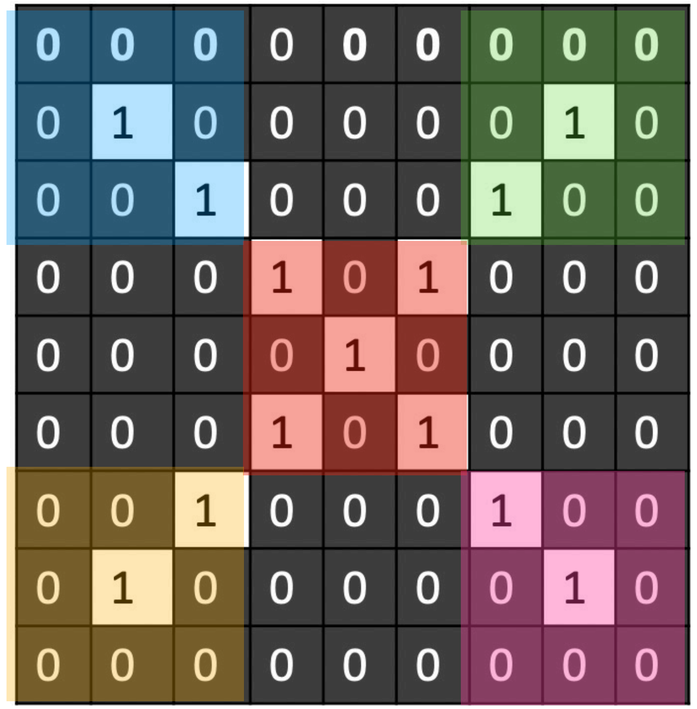
Element-wise multiplication





Filter (kernel)

Input and kernel have the same pattern: high response



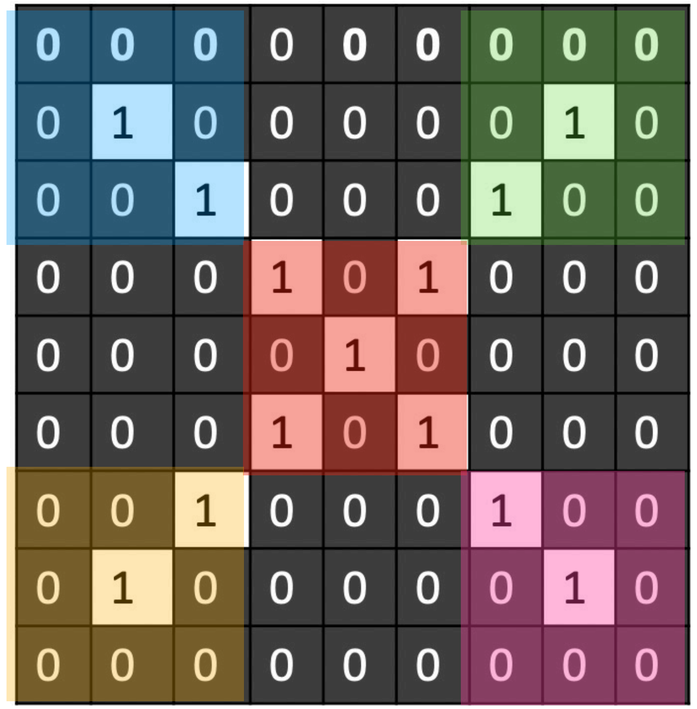
Element-wise multiplication

Sum ($\begin{bmatrix}0&0&0\\0&1&0\\0&0&1\end{bmatrix}$ $\times$ $\begin{bmatrix}0&0&0\\0&1&0\\1&0&0\end{bmatrix}$) =1

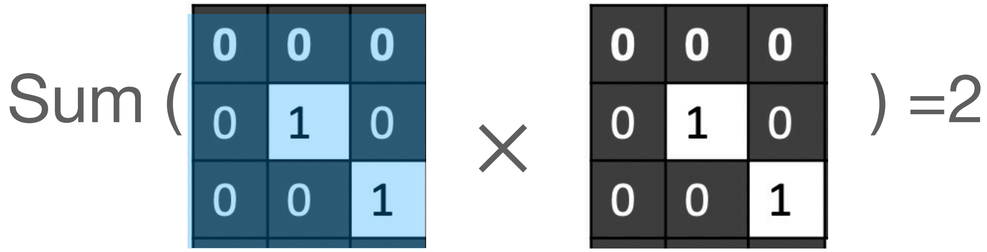
Sum ($\begin{bmatrix}0&0&0\\0&1&0\\1&0&0\end{bmatrix}$ $\times$ $\begin{bmatrix}0&0&0\\0&1&0\\1&0&0\end{bmatrix}$)=2

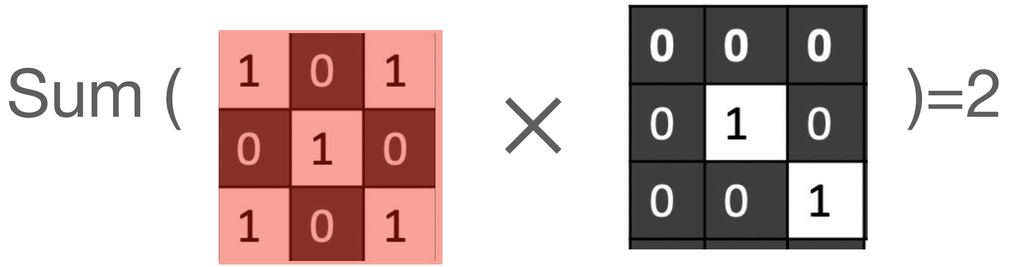
Filter (kernel)

Identify different patterns



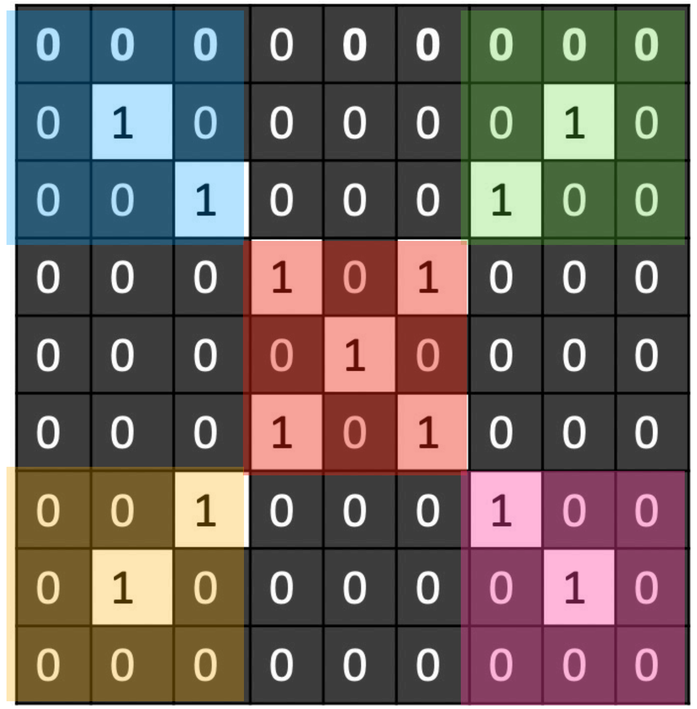
Element-wise multiplication





Filter (kernel)

Different kernels identify different patterns



Element-wise multiplication

Sum (0 0 0 / 0 1 0 / 0 0 1) × (1 0 1 / 0 1 0 / 1 0 1) =2

Sum (1 0 1 / 0 1 0 / 1 0 1) × (1 0 1 / 0 1 0 / 1 0 1)=5

Filter (kernel)

Convolution on 2D

Use kernel to perform element-wise multiplication and sum for every local patch

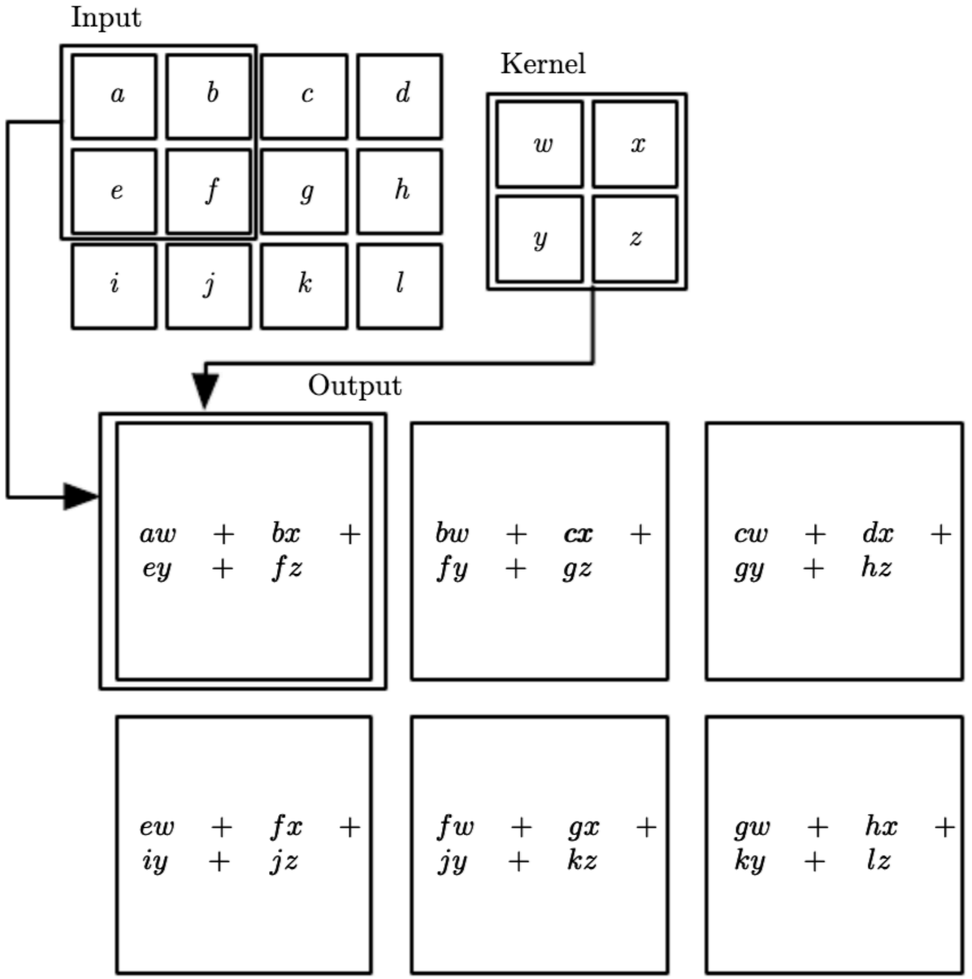
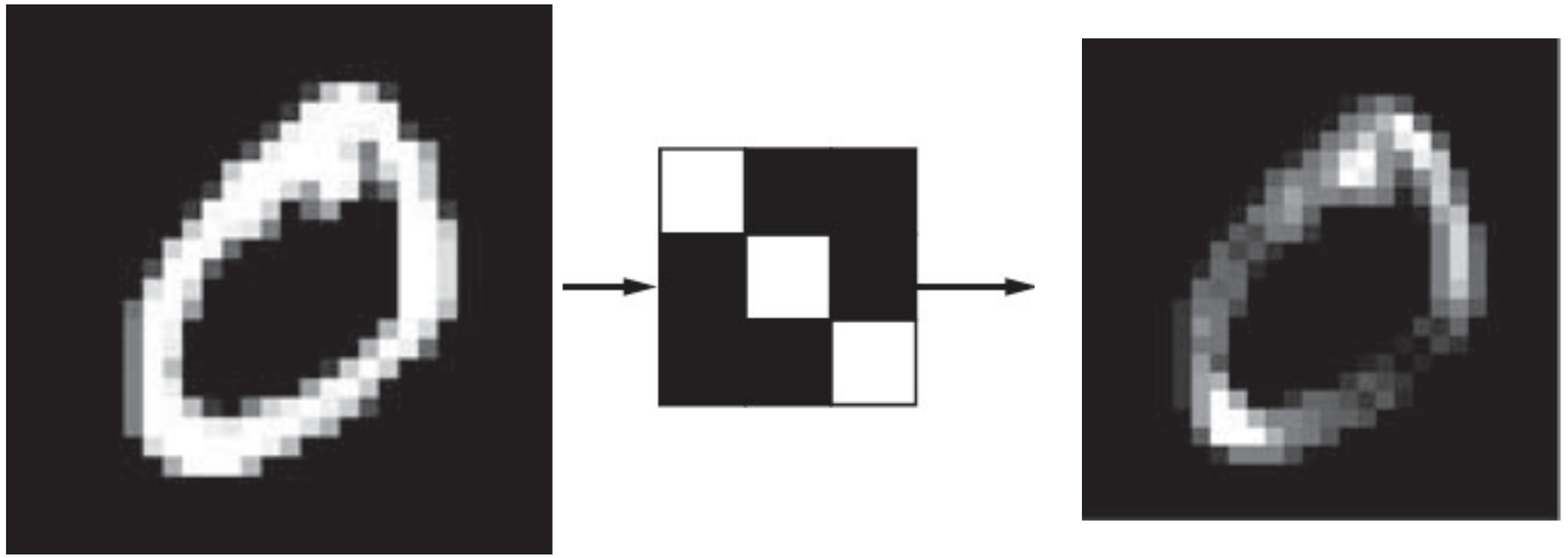


Figure 9.1 in Deep learning by Ian Goodfellow and Yoshua Bengio and Aaron Courville

Response map (Feature map)

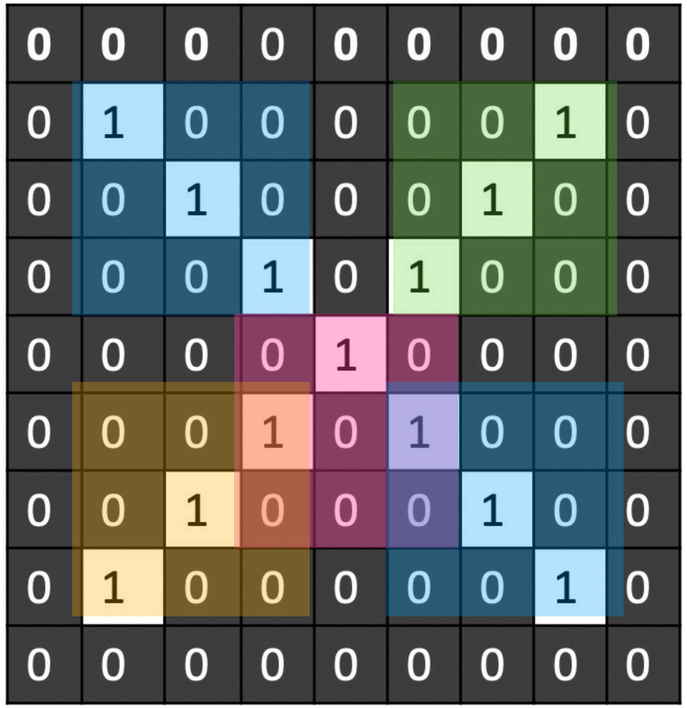


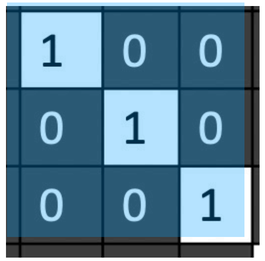
Input kernel

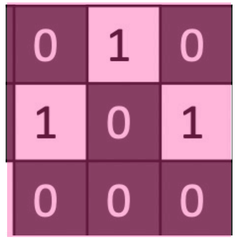
Feature map: 2D map of the presence of a pattern at different locations in an input

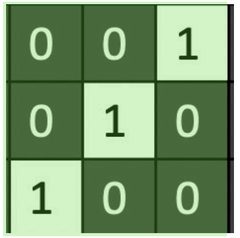
Figure 5.3 in Deep learning with python by Francois Chollet

Different kernels identify different patterns: use multiple filters in each layer





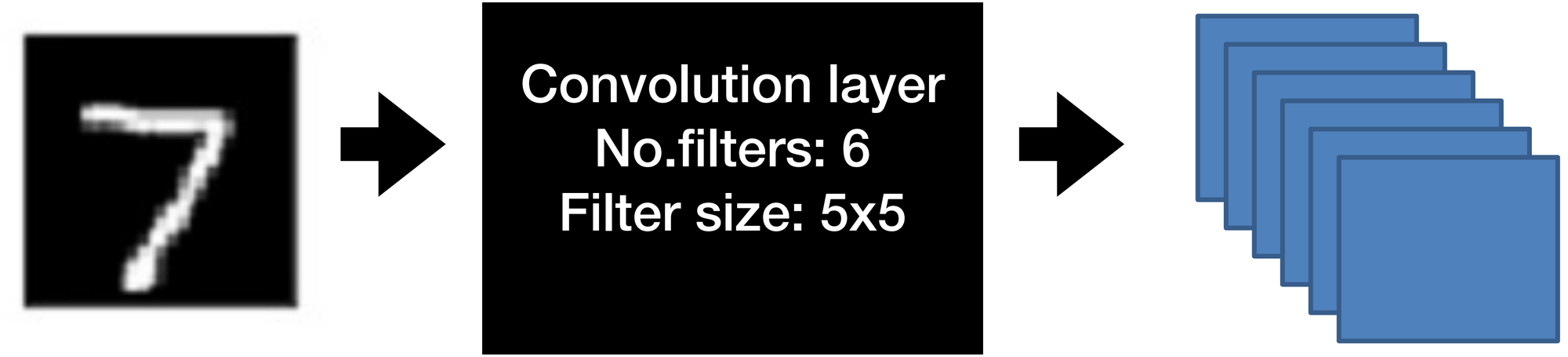




The number of filters decides the number of output feature maps

Two key parameters in Convolution

Filter (kernel) size: Size of the patches extracted from the inputs
Number of filters: Depth (channel) of the output feature map



32x32x1

Input: 1 channel

output: 6 channel

Convolution on Multiple-channel input

R

$* K$(Channel 1)

G

$* K$(Channel 2)

B

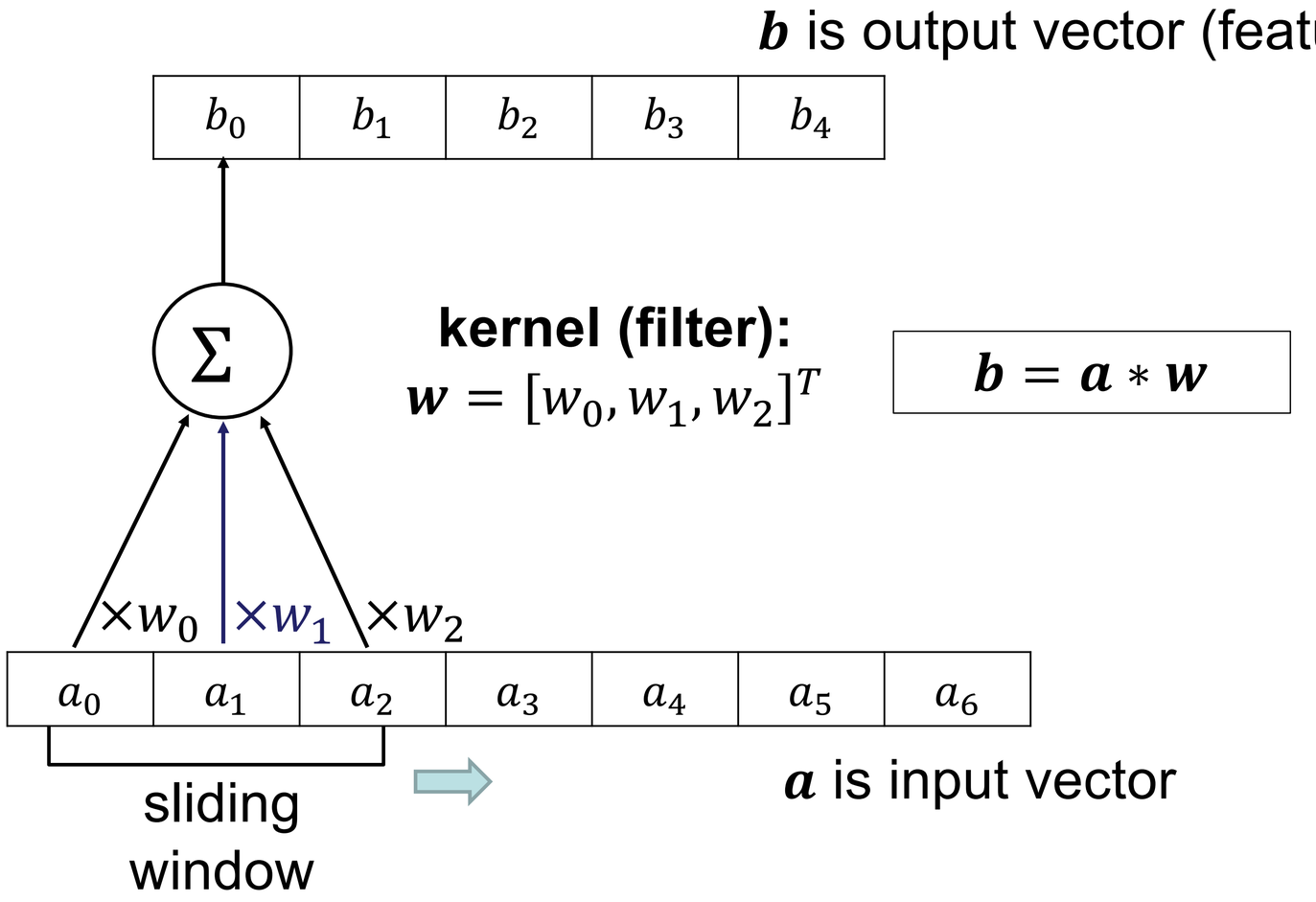
$* K$(Channel 3)

Kernel: same channel (depth)

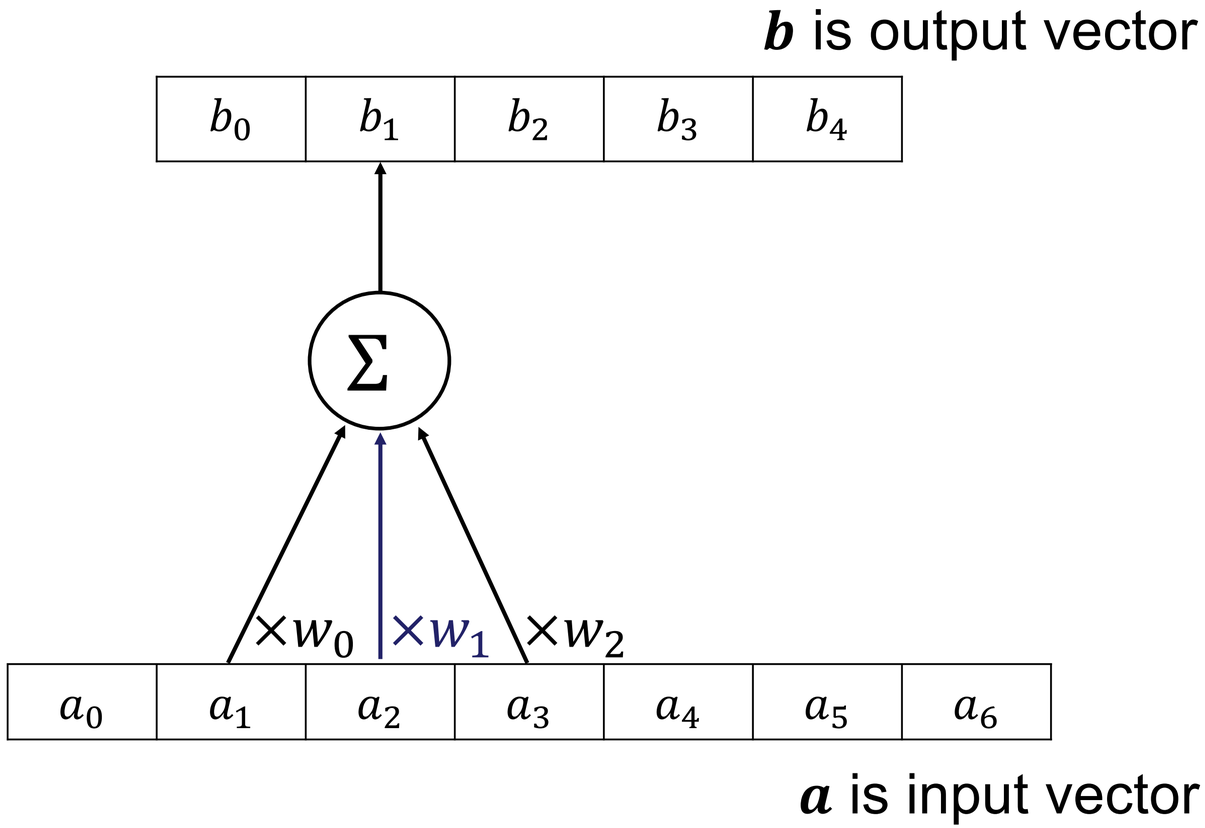
Element-wise sum

One channel

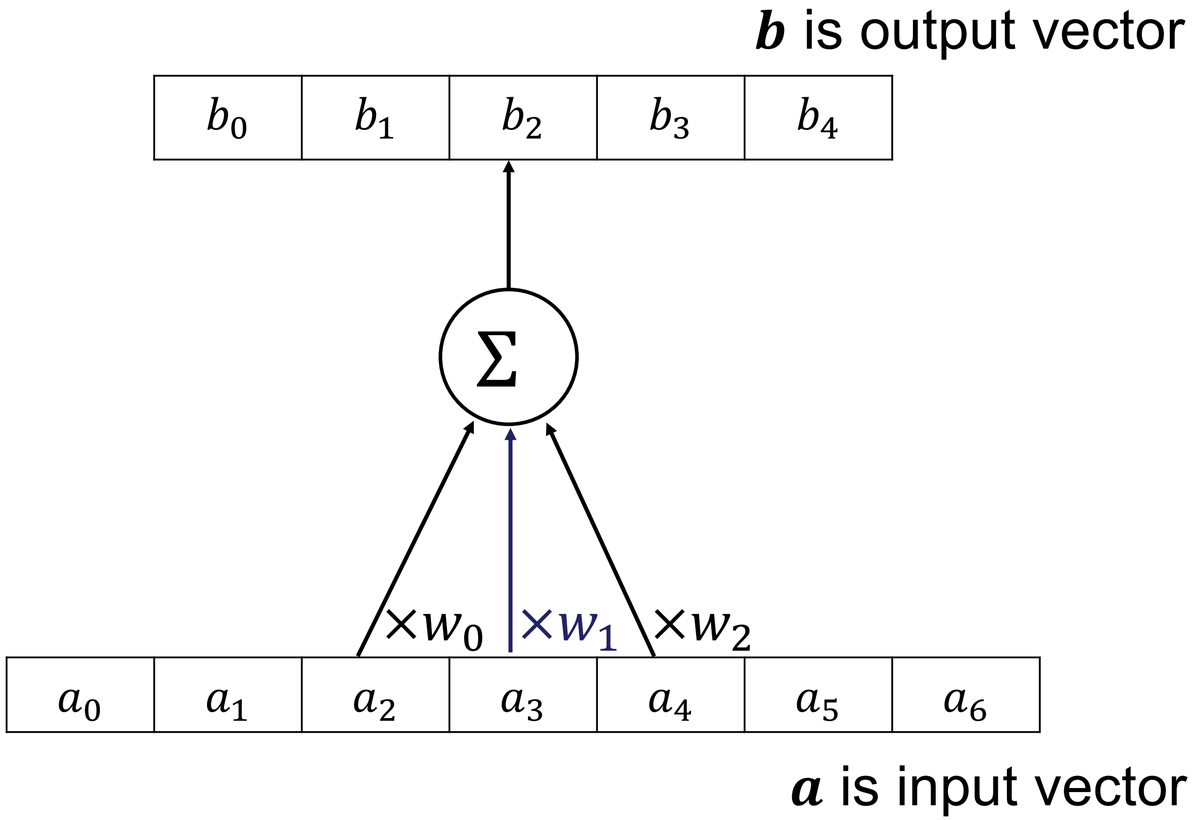
Convolution on 1D



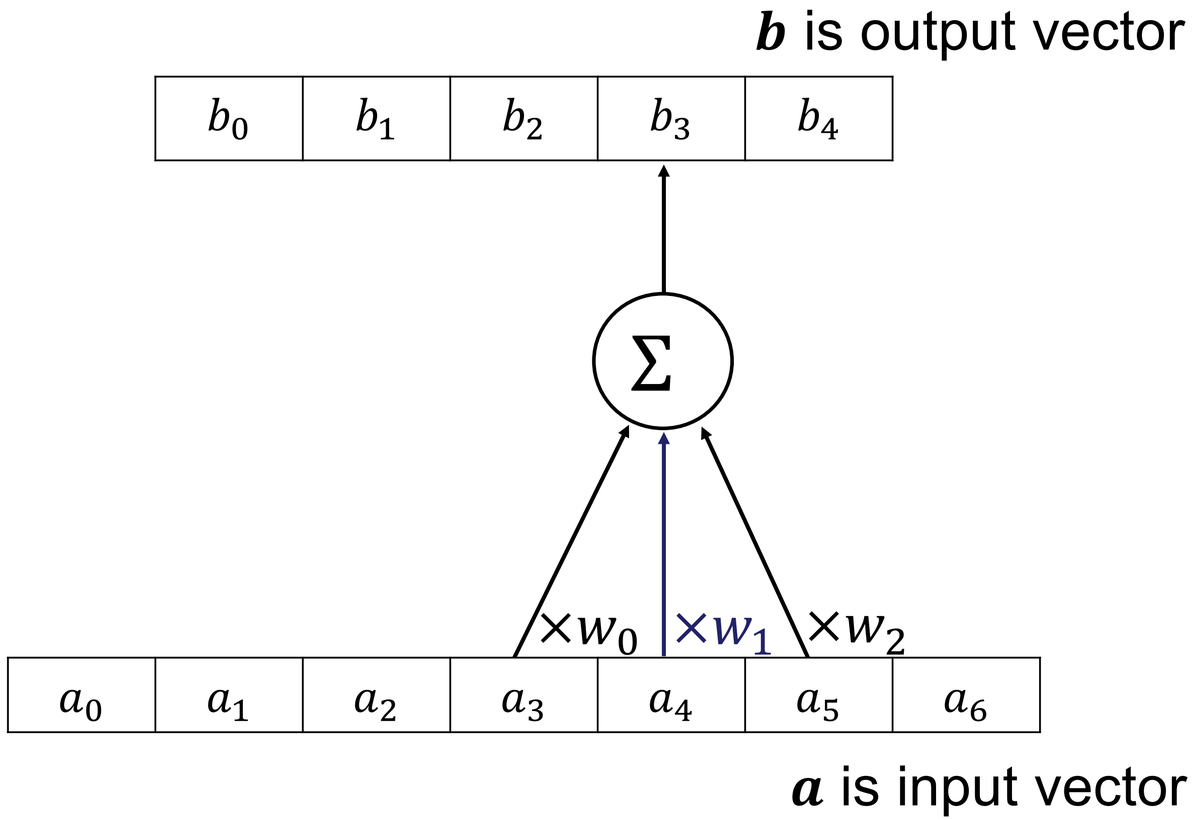
Convolution on 1D



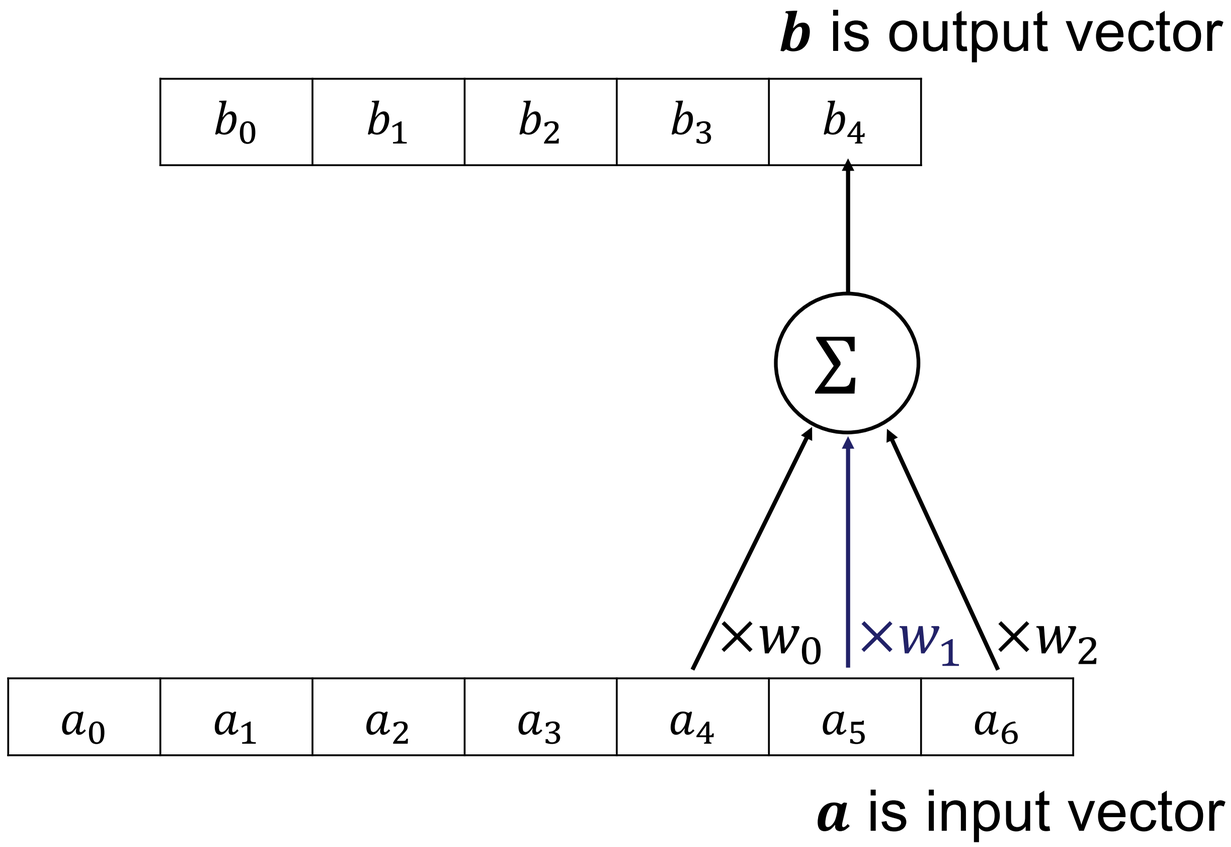
Convolution on 1D



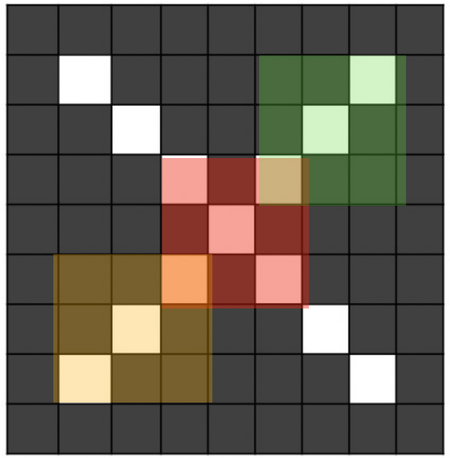
Convolution on 1D



Convolution on 1D

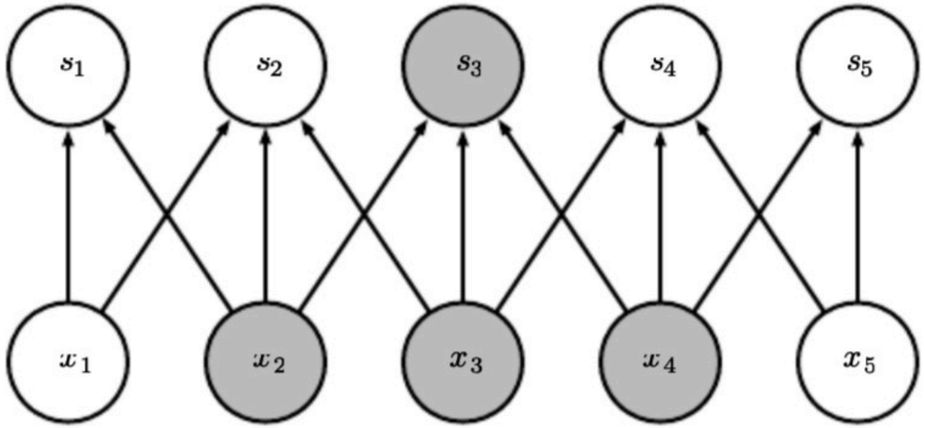


Advantage: learn translation-invariant pattern



Advantage: weight sharing and sparse connection

Convolutional layer:



Fully connected layer:
Each arrow is a weight (no sharing)

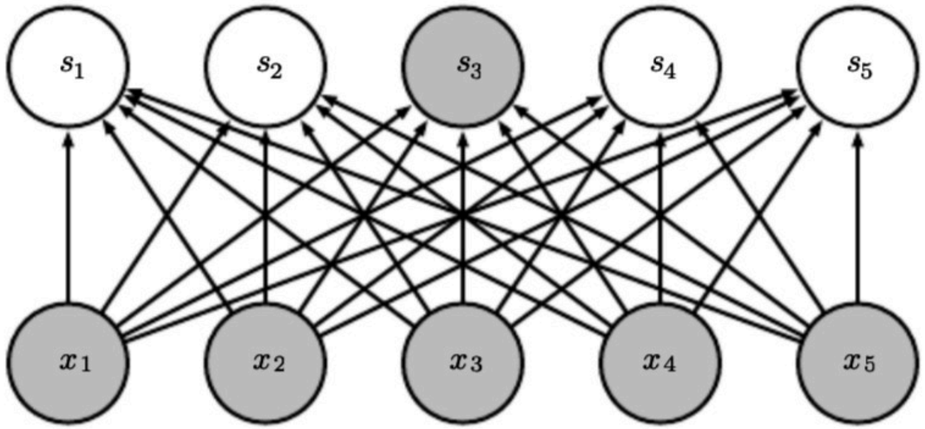


Figure 9.3 in Deep learning by Ian Goodfellow and Yoshua Bengio and Aaron Courville

Advantage: learn hierarchical pattern

More layers: larger size of receptive field (larger window of the input is seen)

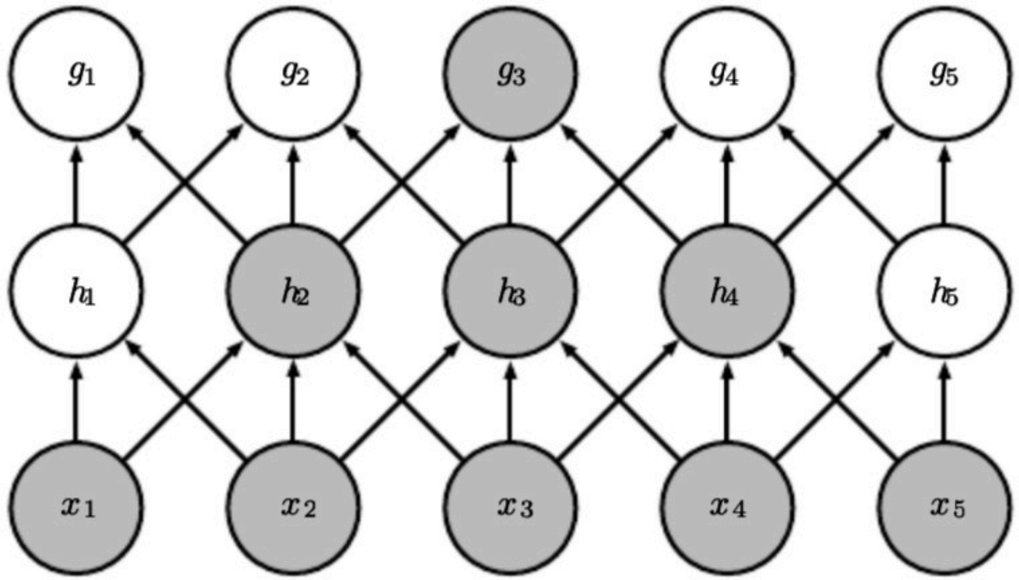


Figure 9.4 in Deep learning by Ian Goodfellow and Yoshua Bengio and Aaron Courville

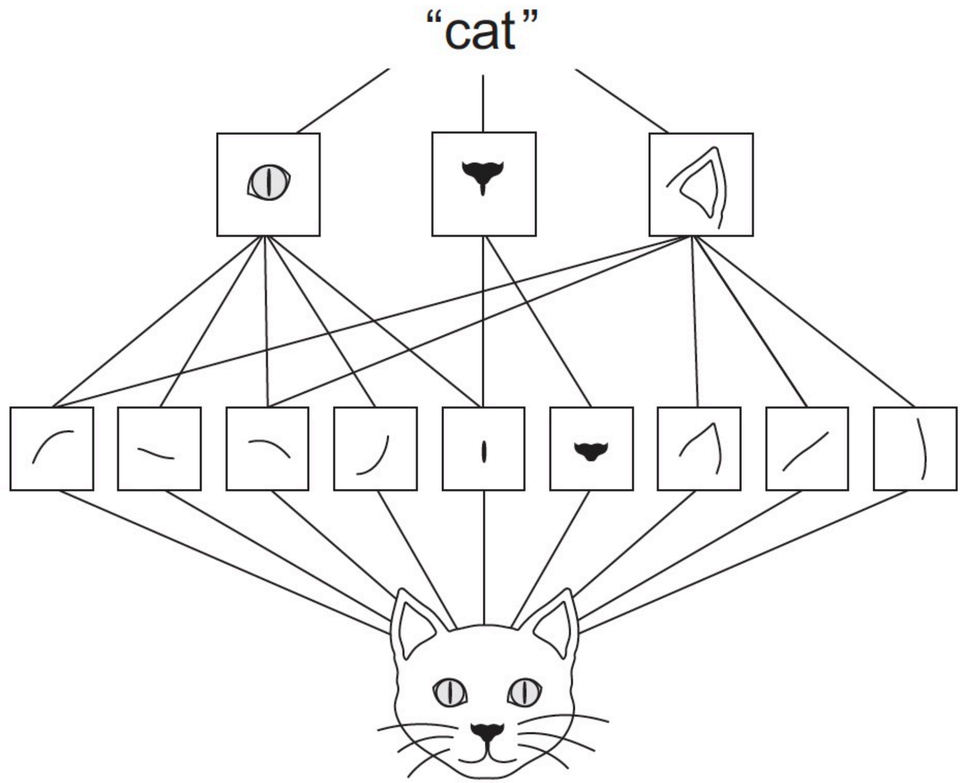
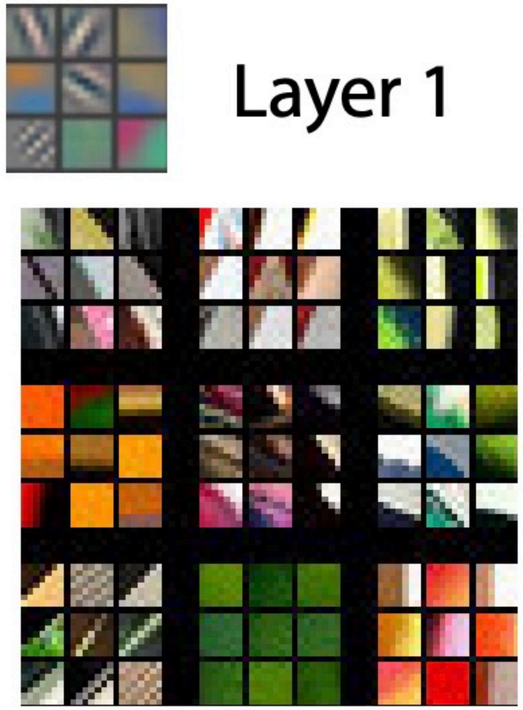
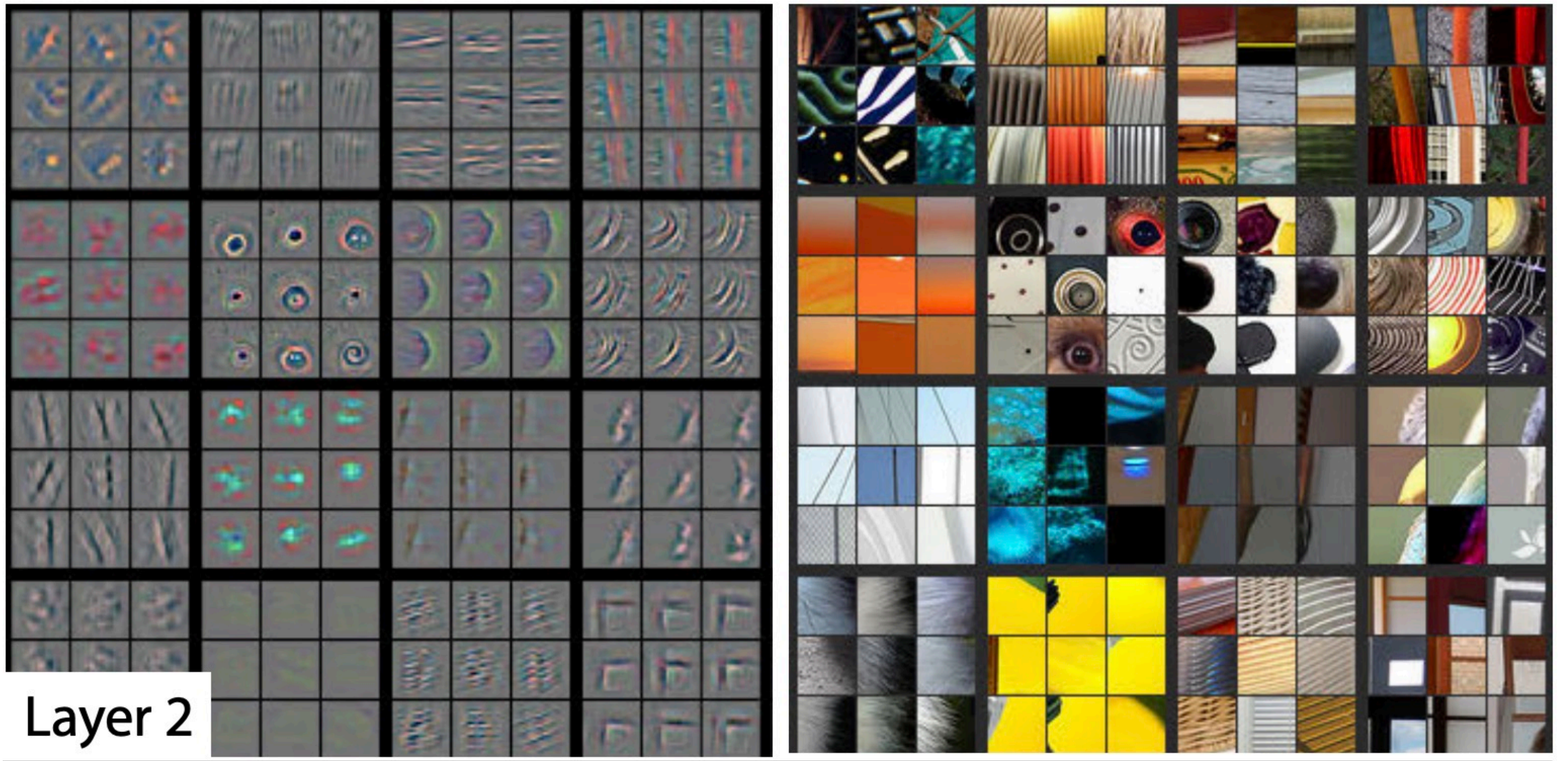
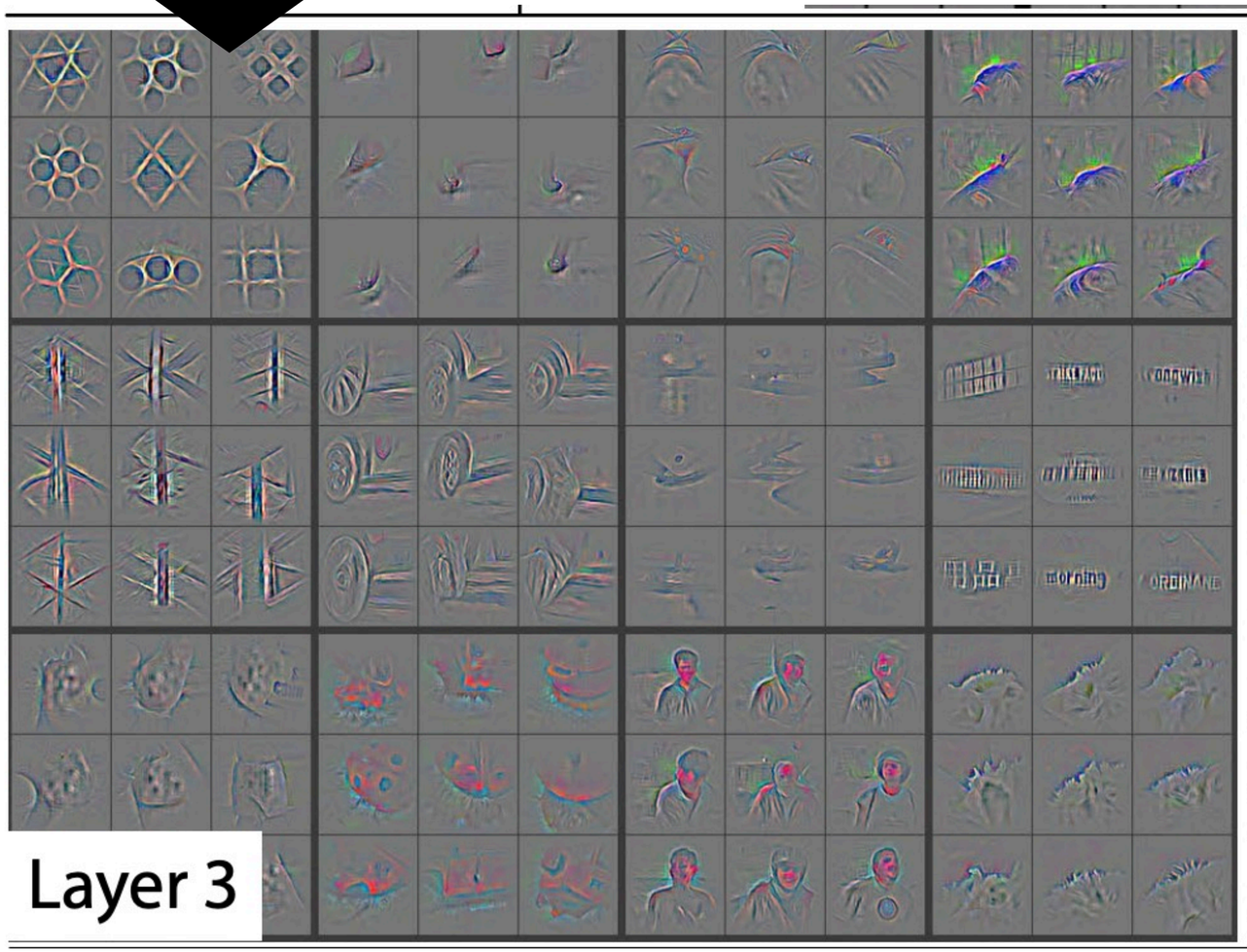


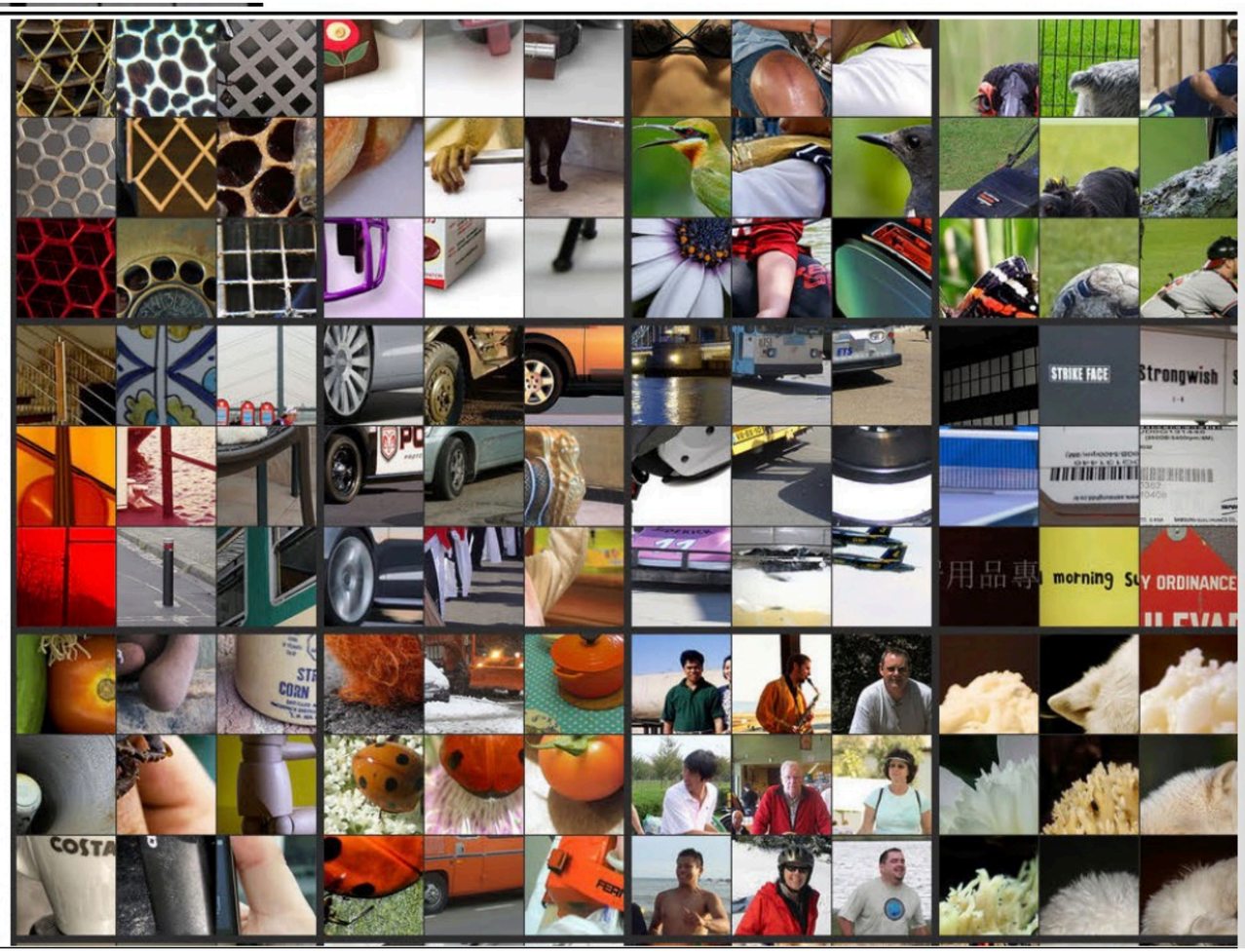
Figure 5.2 in Deep learning with python by Francois Chollet



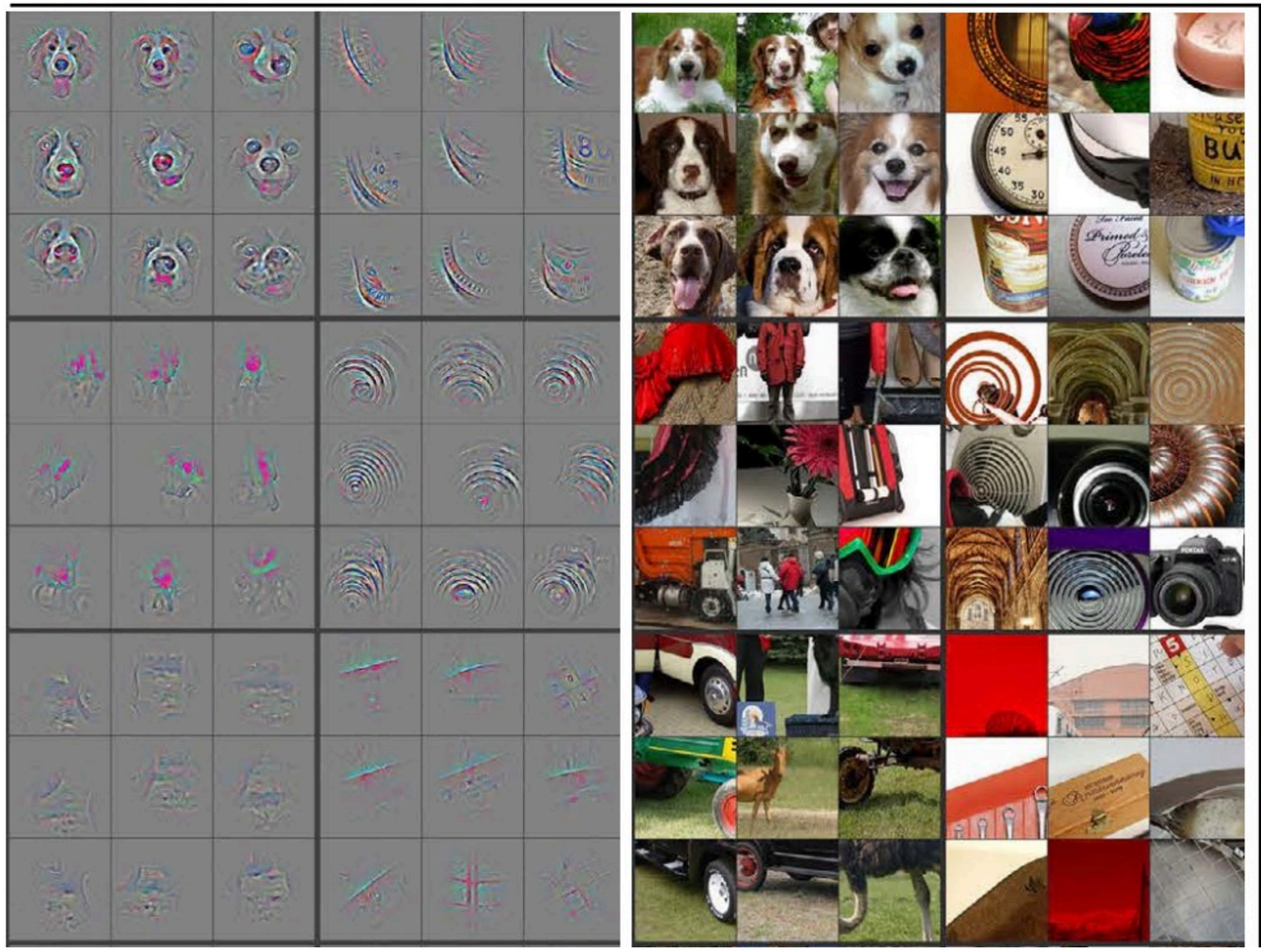


Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." *European conference on computer vision*. Springer, Cham, 2014.

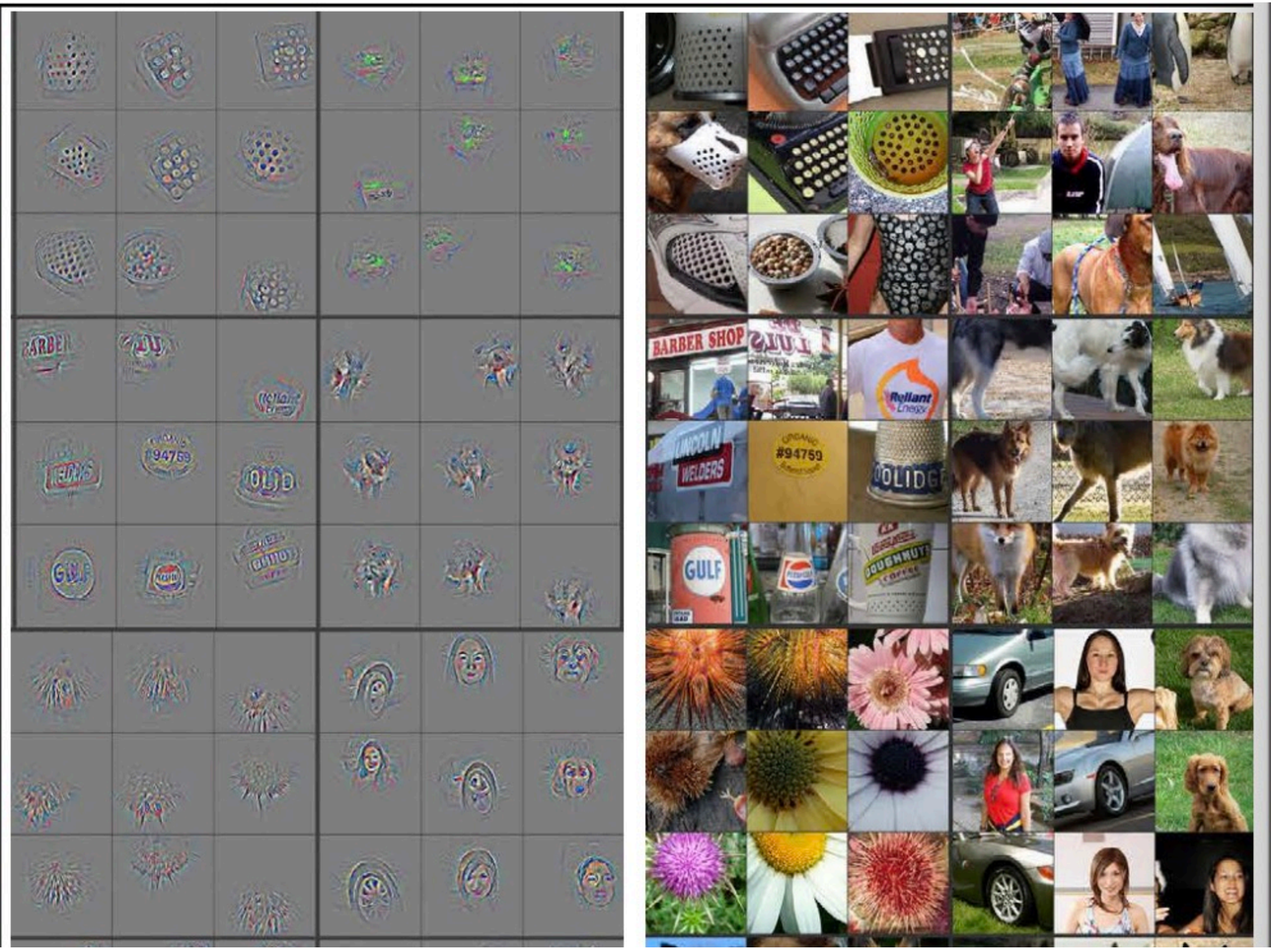




Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." *European conference on computer vision*. Springer, Cham, 2014.



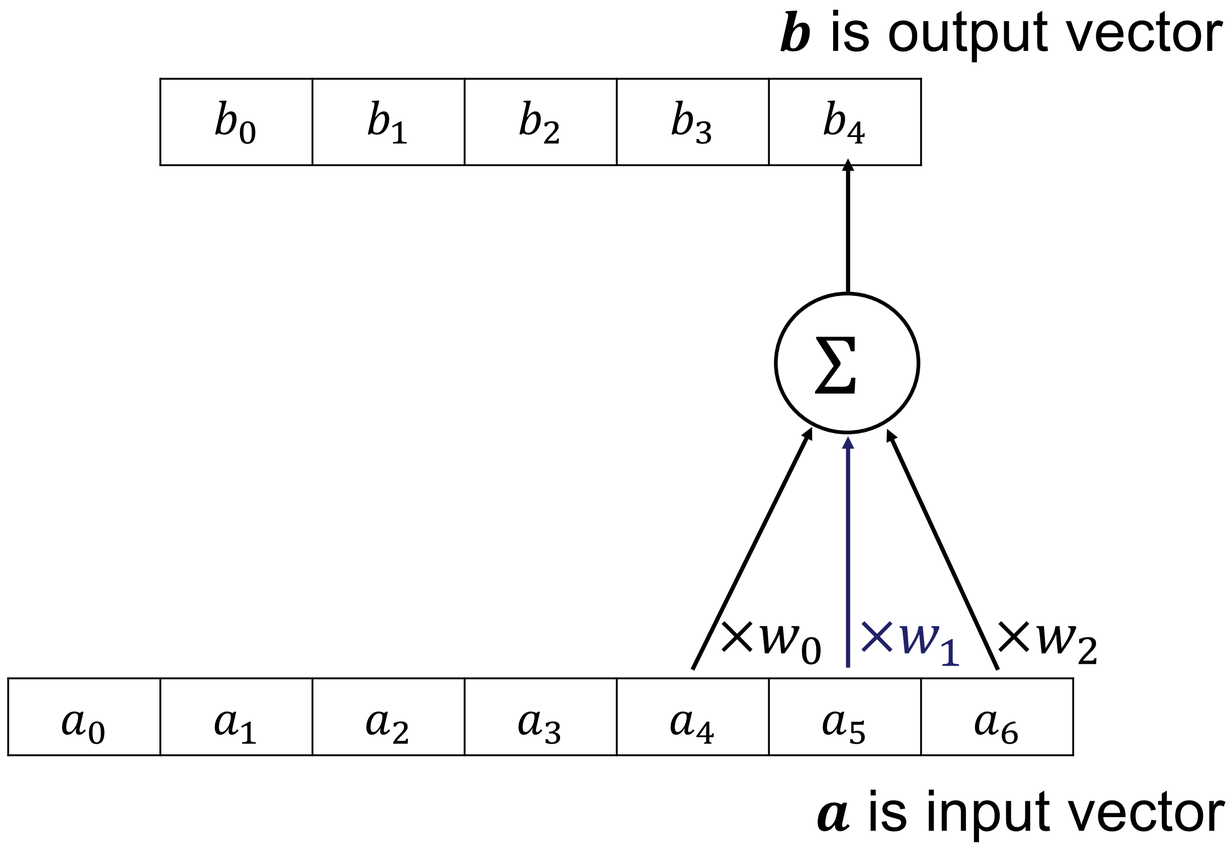
Layer 4



Layer 5

Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." *European conference on computer vision*. Springer, Cham, 2014.

output size $\neq$ input size



output size $\neq$ input size

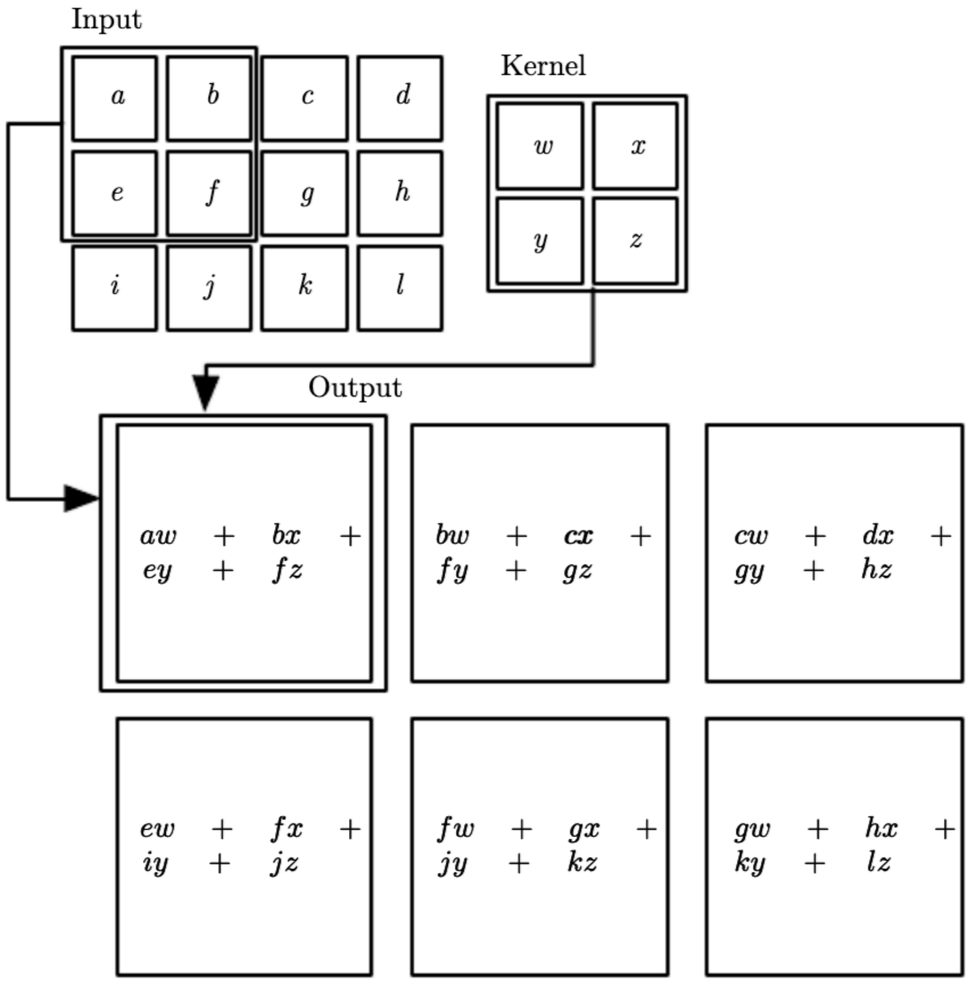


Figure 9.1 in Deep learning by Ian Goodfellow and Yoshua Bengio and Aaron Courville

Padding

adding an appropriate number of rows and columns on each side of the input feature map

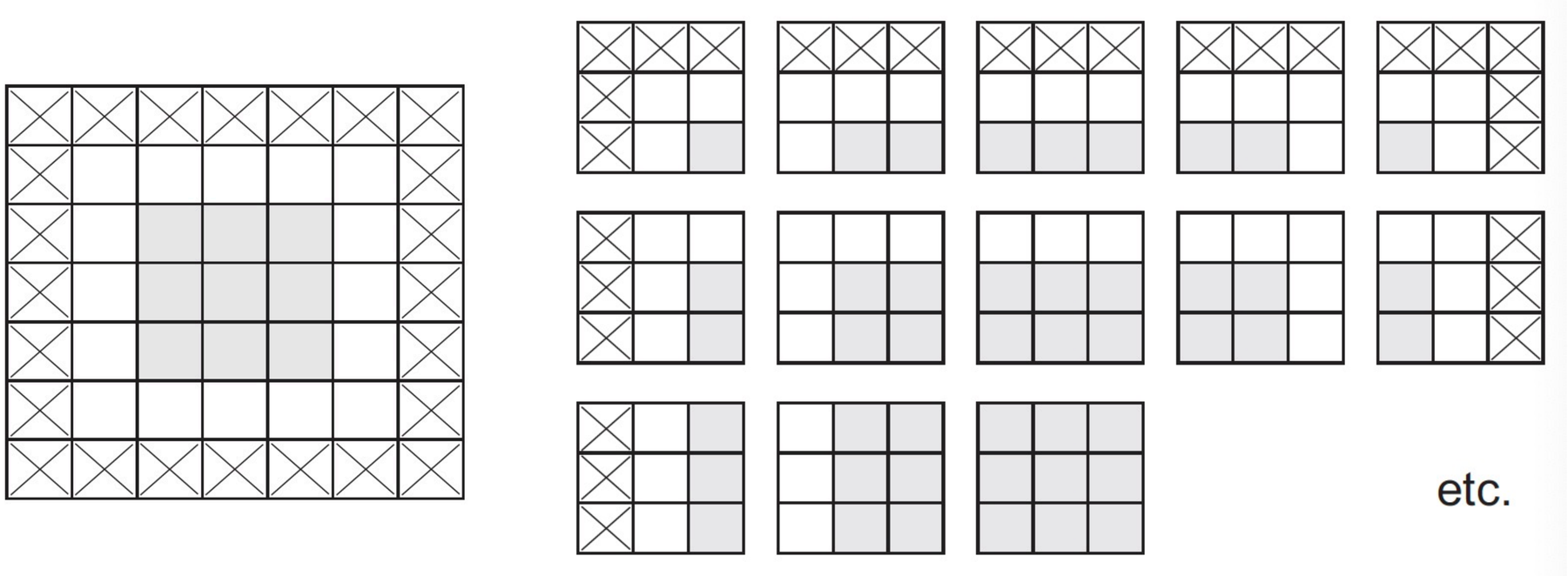


Figure 5.6 in Deep learning with python by Francois Chollet

Stride

the distance between two successive windows

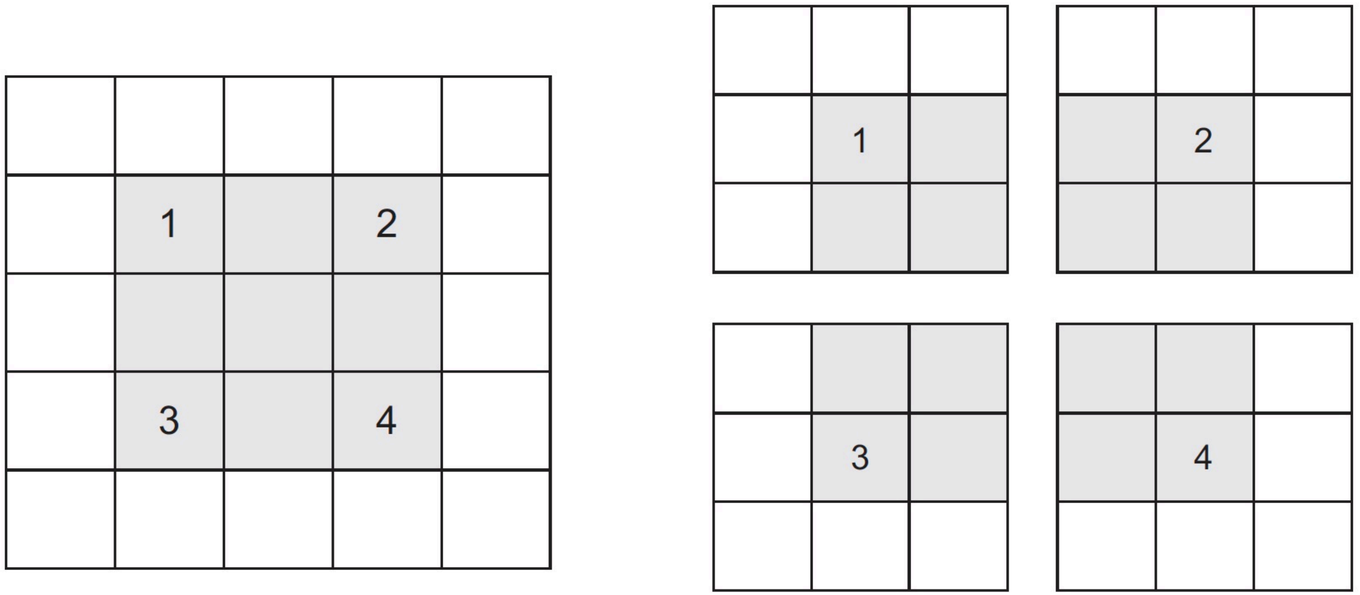


Figure 5.5 in Deep learning with python by Francois Chollet

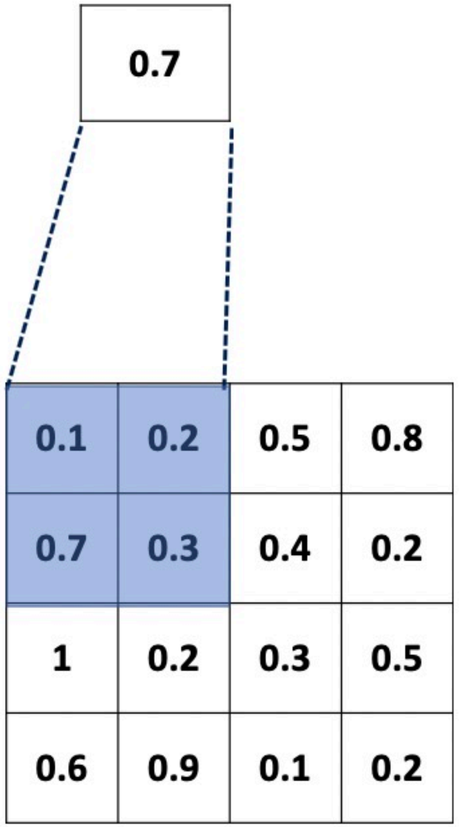
No padding: output_size=ceiling((input_size-kernel_size+1)/stride)

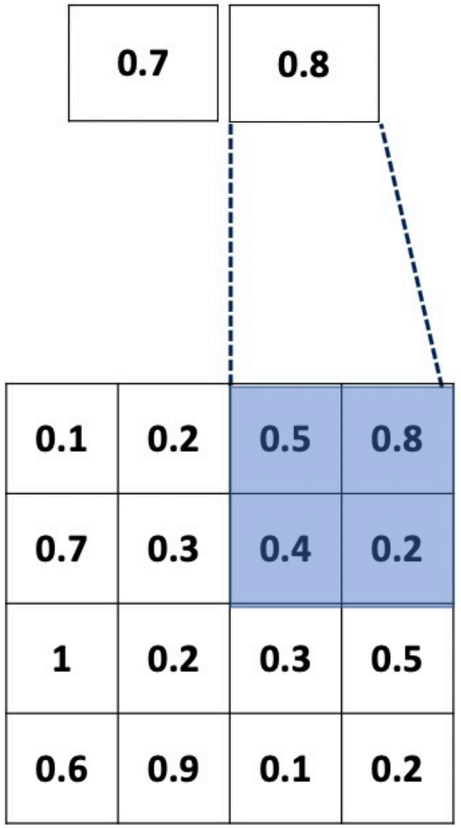
padding: output_size=ceiling(input_size /stride)

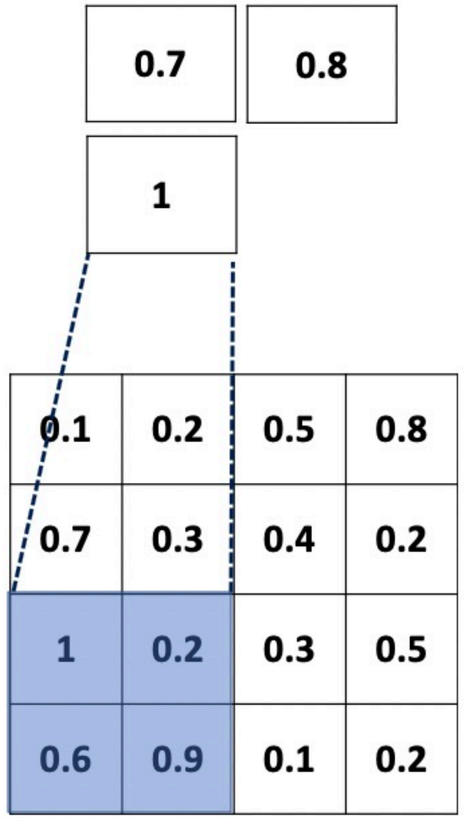
Convoltutional layer

filters: the number of filters in the convolution
kernel_size: the height and width of the 2D convolution window
padding: one of "valid" or “same”
stride: the strides of the convolution along the height and width

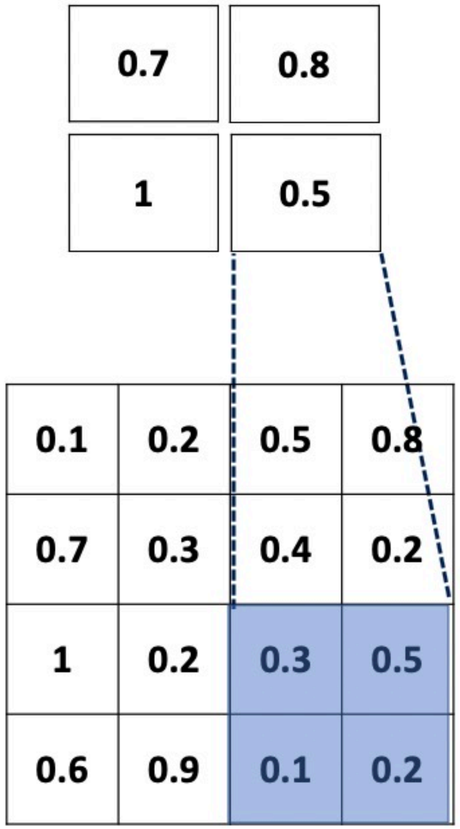
```
layer=keras.layers.Conv2D(
    32,
    (3, 3),
    padding='same',
    strides=1,
    activation='relu')
```


0.7
0.1
0.2
0.5
0.8
0.7
0.3
0.4
0.2
1
0.2
0.3
0.5
0.6
0.9
0.1
0.2


0.7
0.8
0.1
0.2
0.5
0.8
0.7
0.3
0.4
0.2
1
0.2
0.3
0.5
0.6
0.9
0.1
0.2

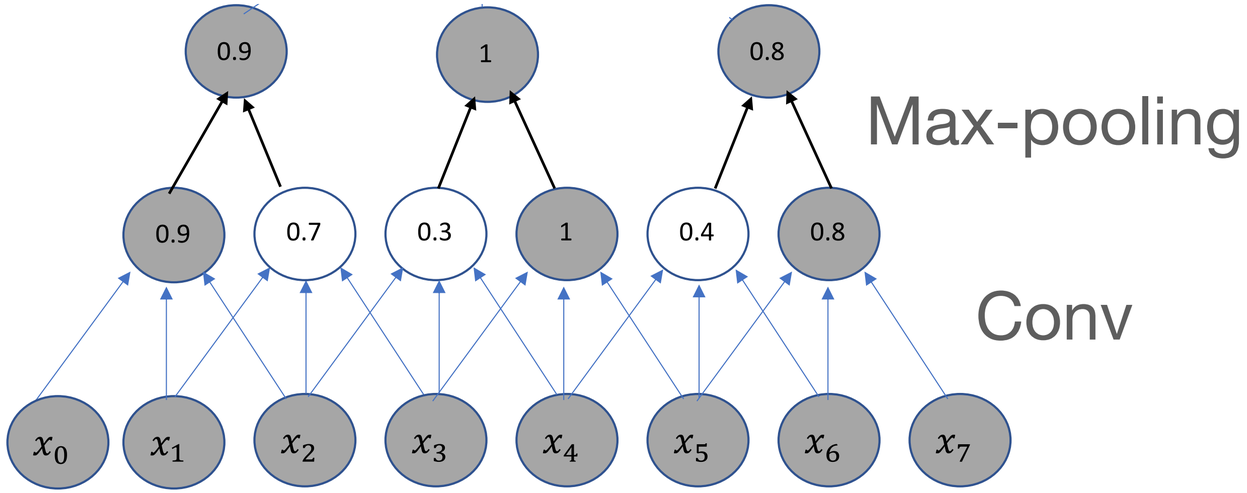

0.7
0.8
1
0.1
0.2
0.5
0.8
0.7
0.3
0.4
0.2
1
0.2
0.3
0.5
0.6
0.9
0.1
0.2

Max-pooling layer

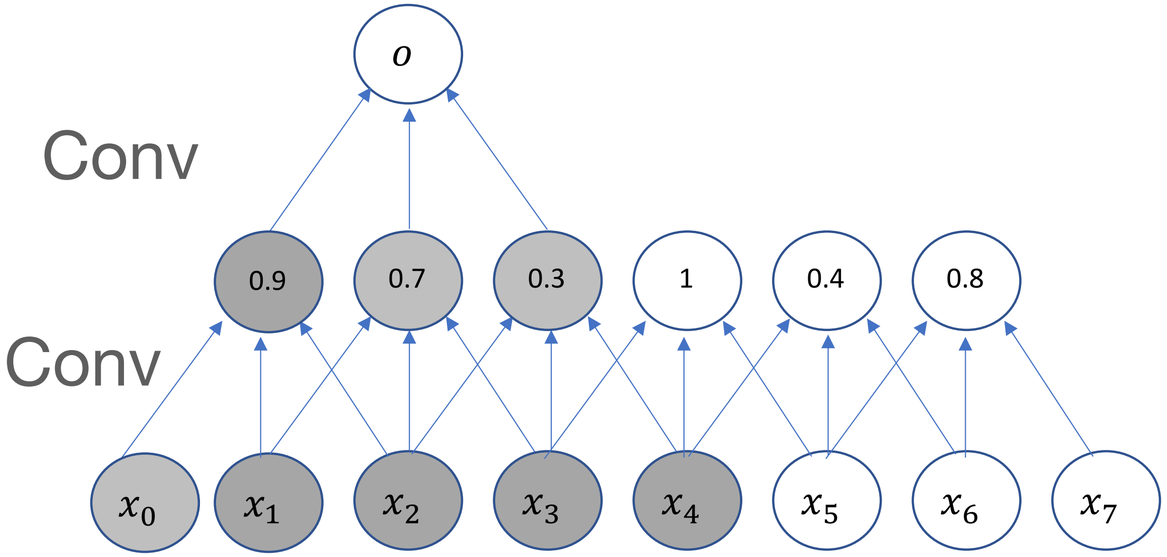


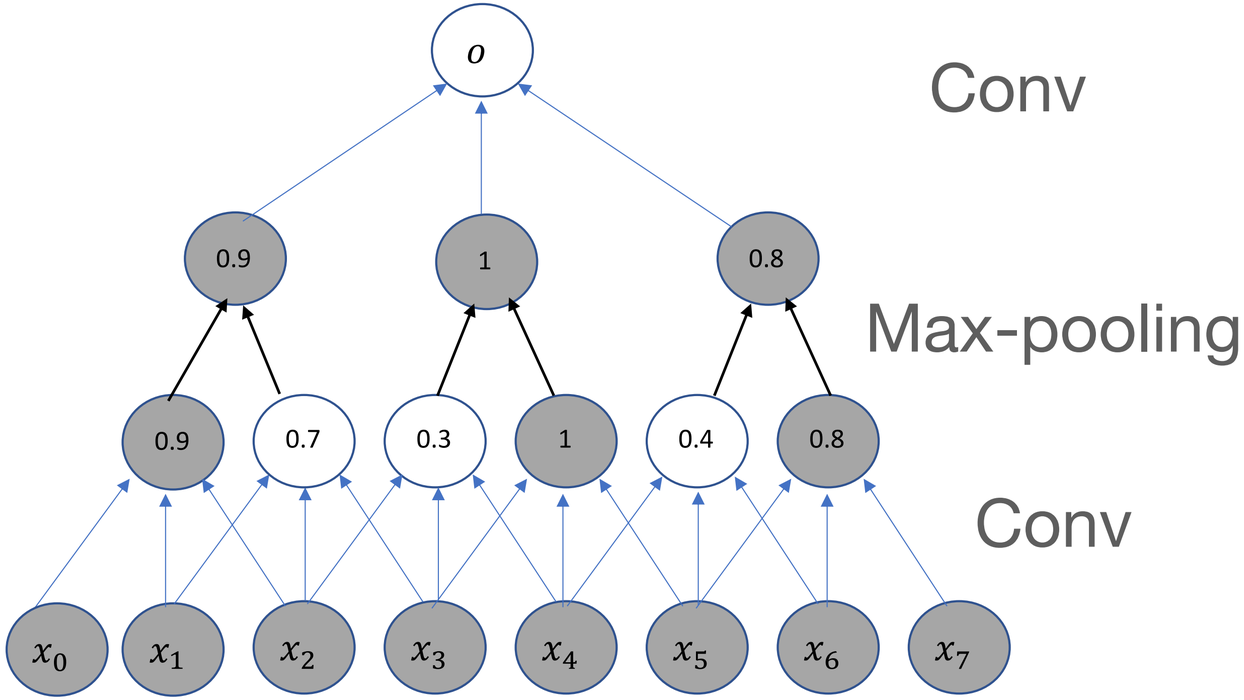

```
tf.keras.layers.MaxPooling2D(
    pool_size=(2, 2), strides=None, padding="valid")
```

Advantage: downsample feature map, reduce computational burden



Advantage: increase size of receptive field (window of the input is seen)





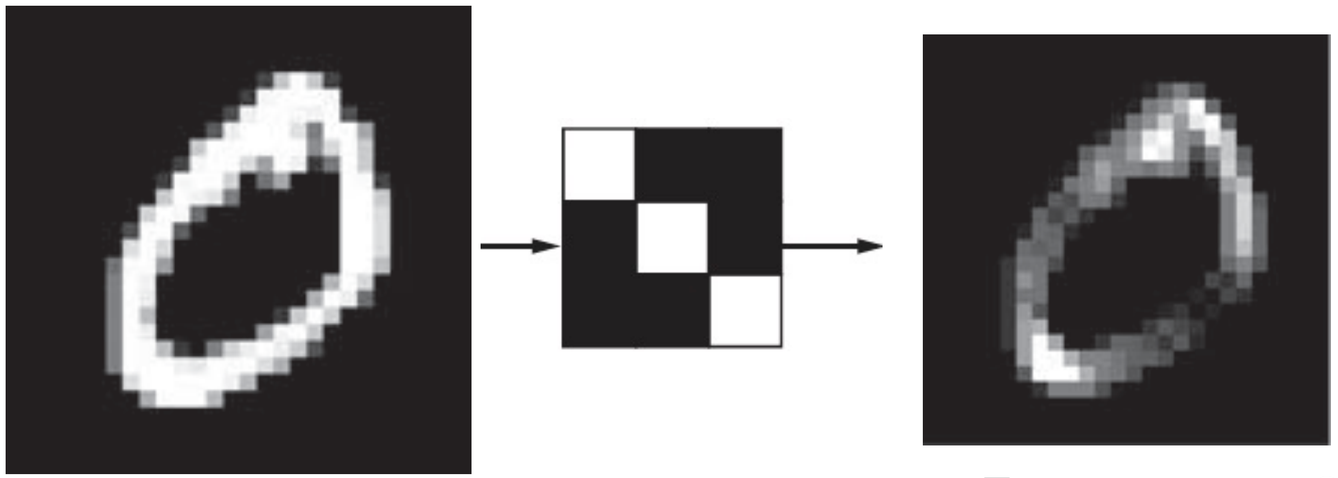
Other pooling method: Average pooling

taking the average value over the patch

Why max-pooling to downsample feature map?

Average pooling: dilute feature-presence information

Convolution with stride>1: miss feature-presence information



Input kernel

Feature map: 2D map of the presence of a pattern at different locations in an input

Figure 5.3 in Deep learning with python by Francois Chollet

Outline

- Batch size
- Othe optimisation methods (optimiser)
  - Momentum
  - Adaptive gradient (AdaGrad)
  - Root Mean Square Propagation (Rmsprop)
  - Adaptive moment estimation (Adam)
- Activation function
- How to prevent overfitting

Gradient descent Algorithm

- Randomly shuffle/split all training examples in B batches
- Choose initial $\boldsymbol{\theta}^{(1)}$
- For i from 1 to T
- For j from 1 to B
- Do gradient descent update using data from batch j

Iterations over the entire dataset are called *epochs*

Stochastic gradient descent: B=N

Choose initial guess $\boldsymbol{\theta}^{(0)}$, $k = 0$

Here $\boldsymbol{\theta}$ is a set of all weights form all layers

For i from 1 to T (epochs)

For j from 1 to N (training examples)

Consider example $\{\boldsymbol{x}_j, y_j\}$

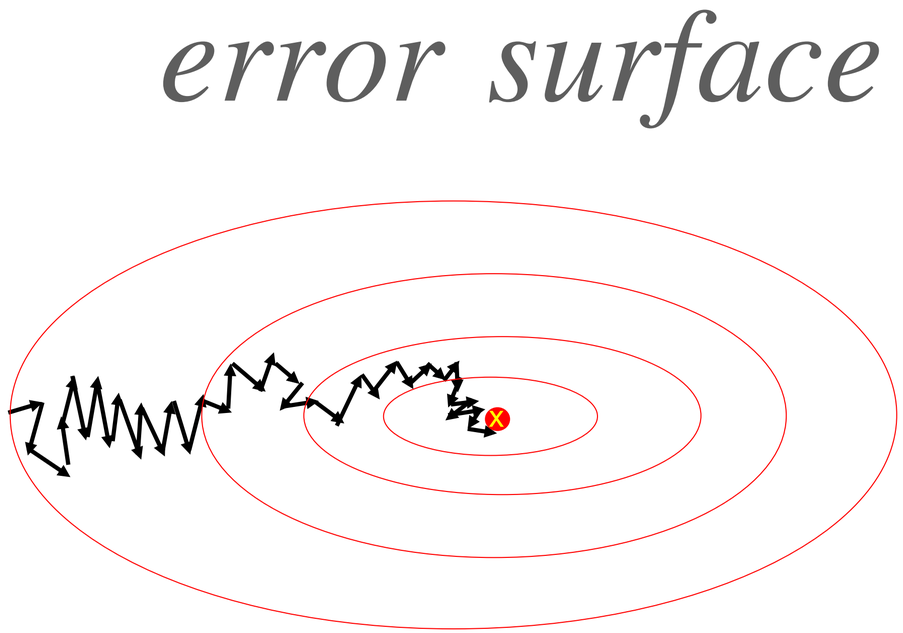
Update: $\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \eta \nabla L(\boldsymbol{\theta}^{(k)})$; $k \leftarrow k+1$

Stochastic gradient descent (SGD)

Batch number==N (Batch size==1)

Quick update each step, but

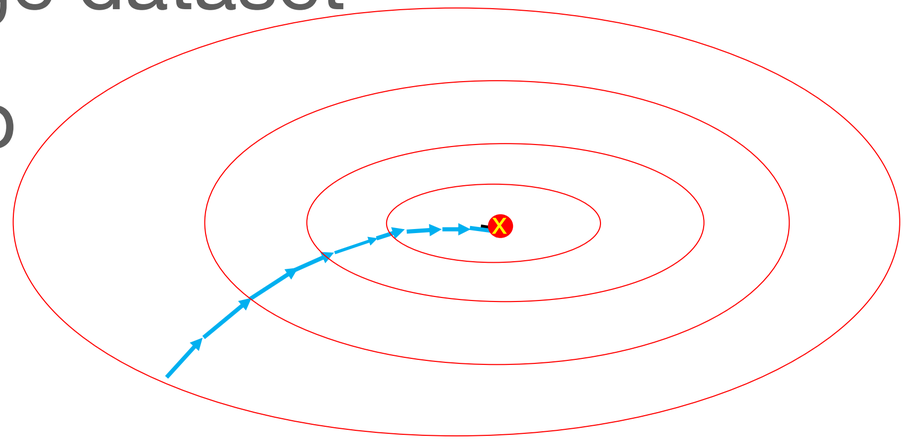
- high variance in gradient
- update model too often



Batch SGD: Batch number==1 (Batch size==N)

Stable update, but

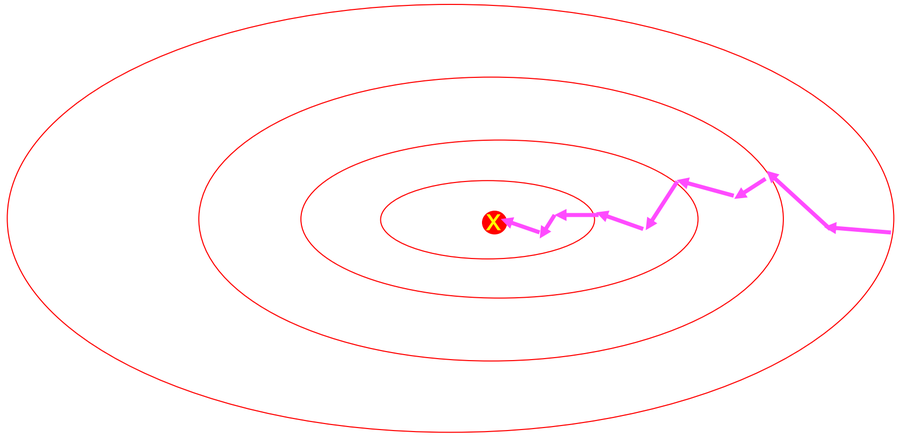
- not computational feasible for large dataset
- takes long time to move each step



Mini-batch SGD

mini-batch: 1<batch size < N

- compared to SGD (batch size==1), more stable update
- compared to batch SGD (batch size==N), more computational feasible



Gradient descent: carefully choose learning rate

To reduce loss, update weights:

$$w_{t+1} = w_t - \eta * \Delta L(w_t)$$

$$\Delta L(w_t) = \frac{\partial L}{\partial w_t}$$

$$\eta : \ leanring \ rate$$

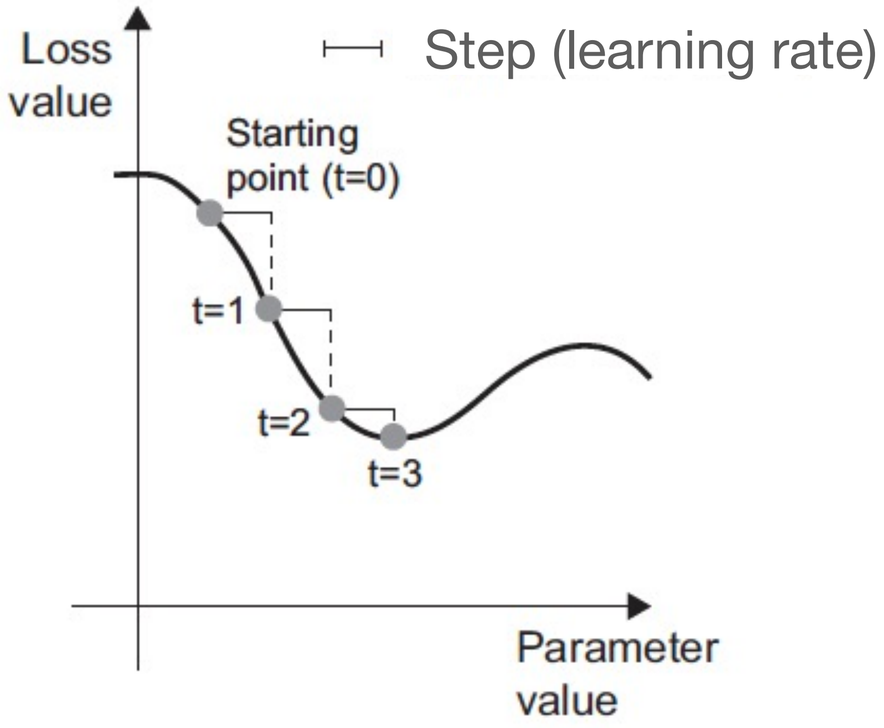


Figure 2.11 in Deep learning with python by Francois Chollet

Gradient descent: carefully choose learning rate

η : *leanring rate*

Large η: random location

Small η: local optimal value

$$w_{t+1} = w_t - \eta * \Delta L(w_t)$$

$$\Delta L(w_t) = 0 : \ no\ update$$

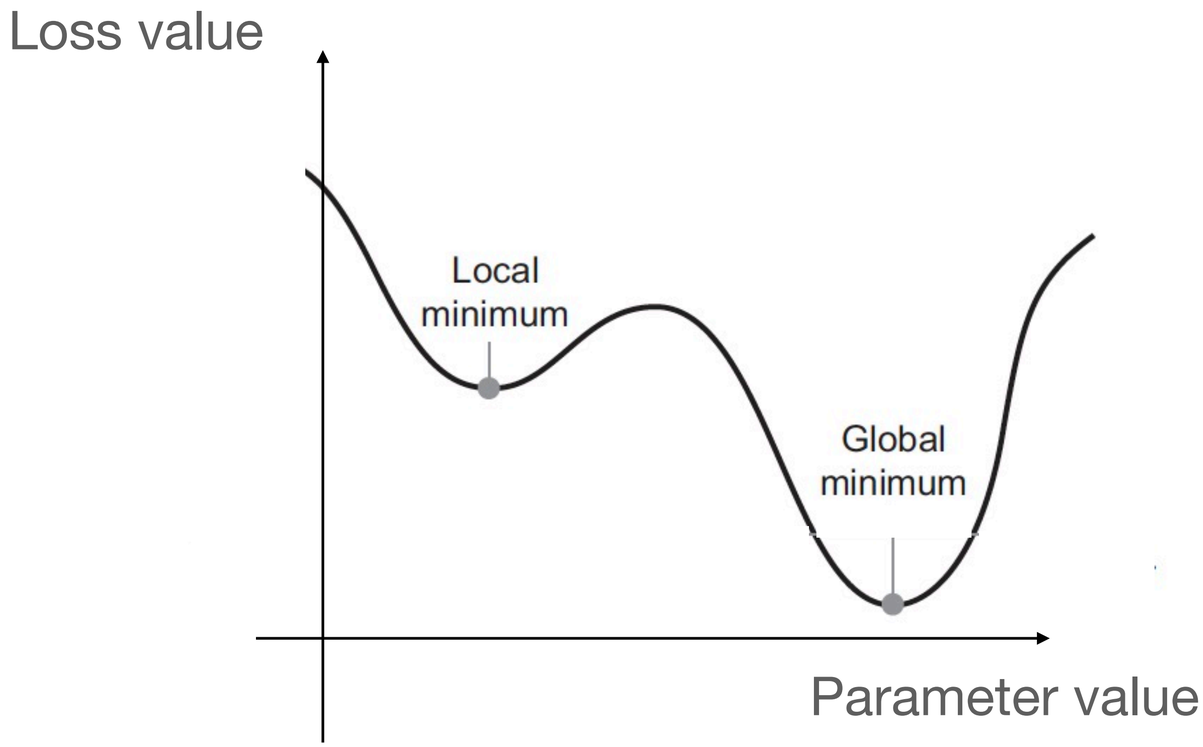


Figure 2.13 in Deep learning with python by Francois Chollet

Gradient descent: carefully choose learning rate

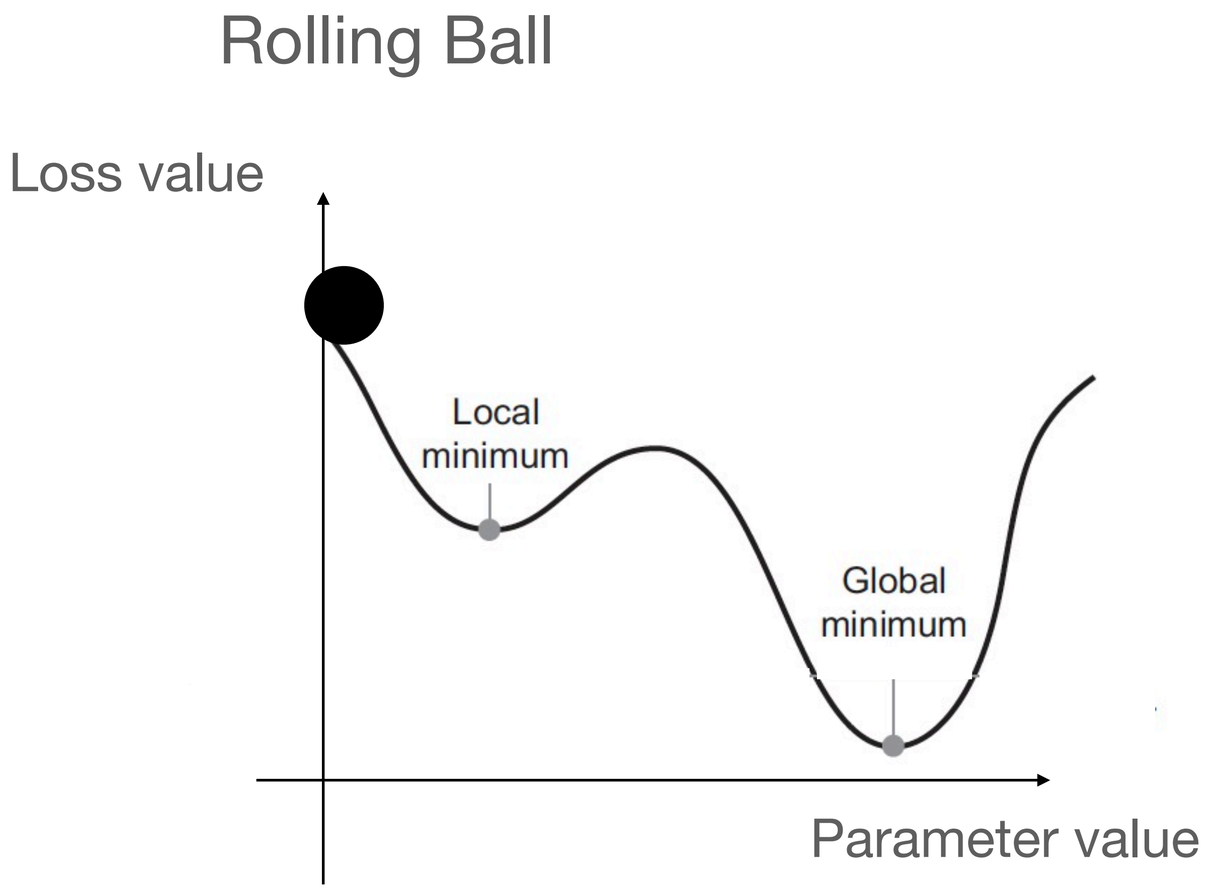


Figure 2.13 in Deep learning with python by Francois Chollet

Gradient descent: carefully choose learning rate

Rolling Ball has velocity (momentum) to move forward: avoid the local minimum location

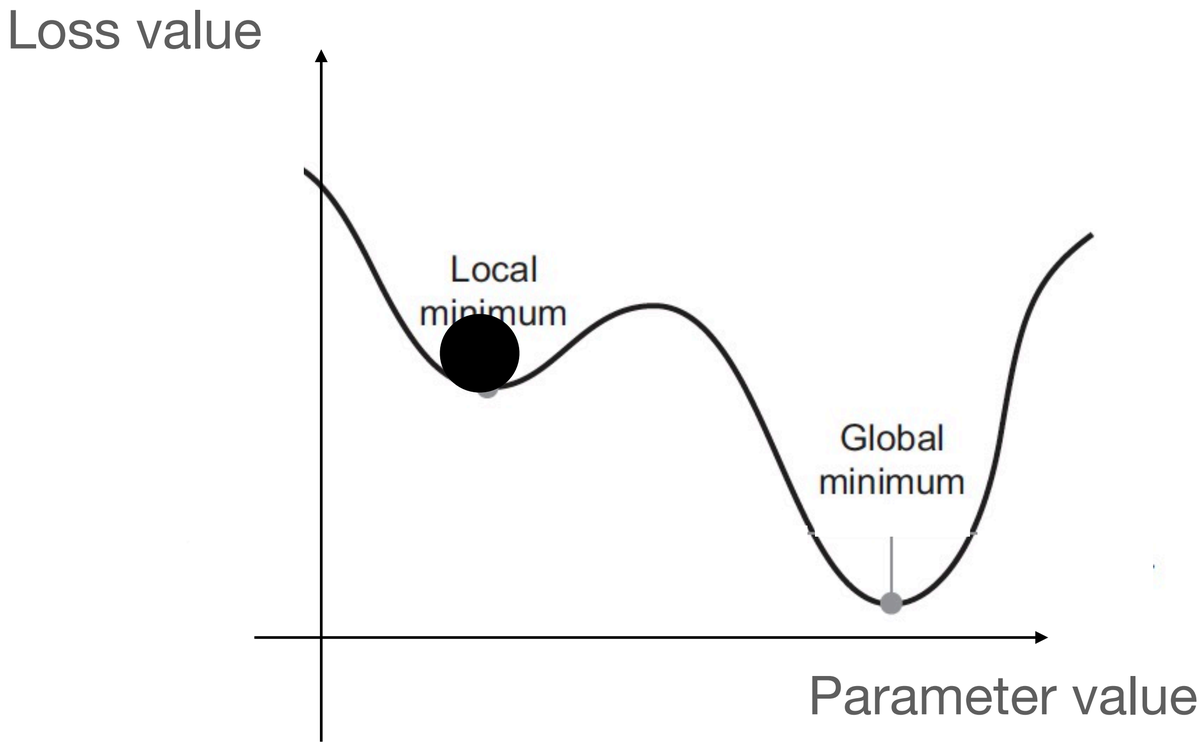


Figure 2.13 in Deep learning with python by Francois Chollet

Gradient descent: carefully choose learning rate

Rolling Ball has velocity (momentum) to move forward:
avoid the local minimum location

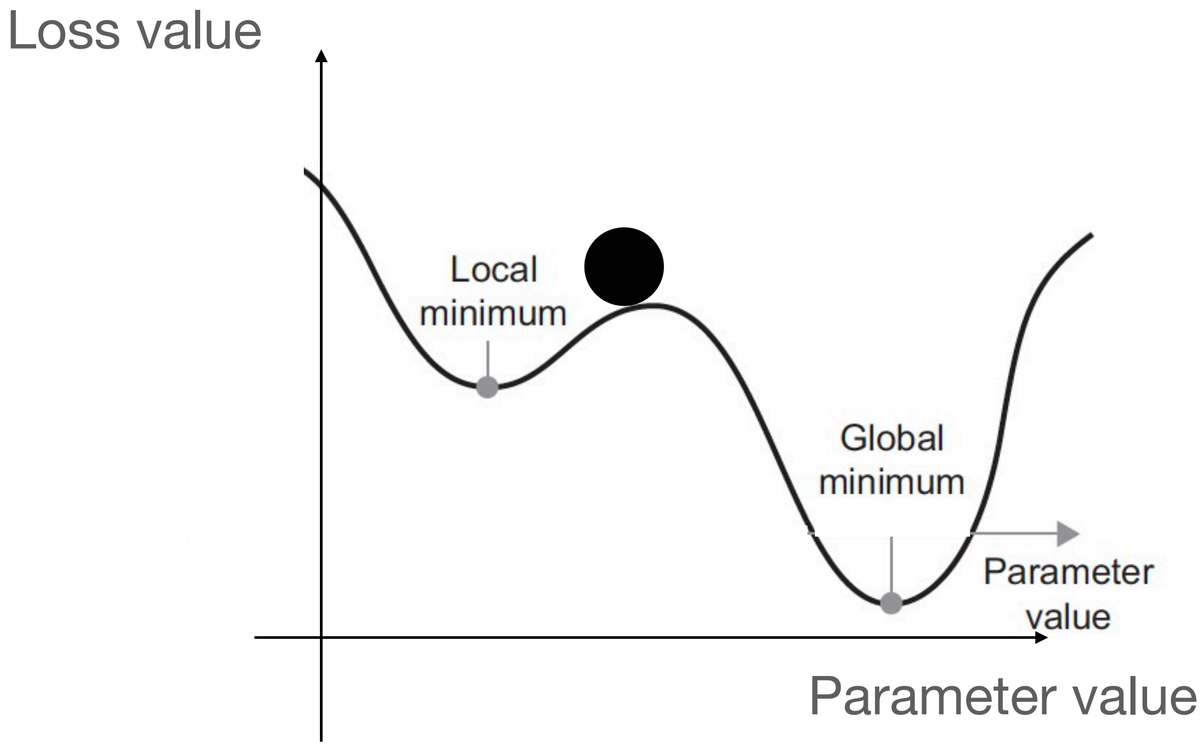


Figure 2.13 in Deep learning with python by Francois Chollet

Update weights with momentum

```
opt = tf.keras.optimizers.SGD(learning_rate=0.01,momentum=0.9)
```

$$w_{t+1} = w_t - \eta \Delta L(w_t) + v_t$$

$$v_t = \alpha * v_{t-1} + (-\eta \Delta L(w_t))$$

α：momentum (decay the previous velocity. e.g. 0.9)

Combine previous weight update with current gradient to calculate the next weight update

Advantages:

- fast convergence speed
- avoid local minimum

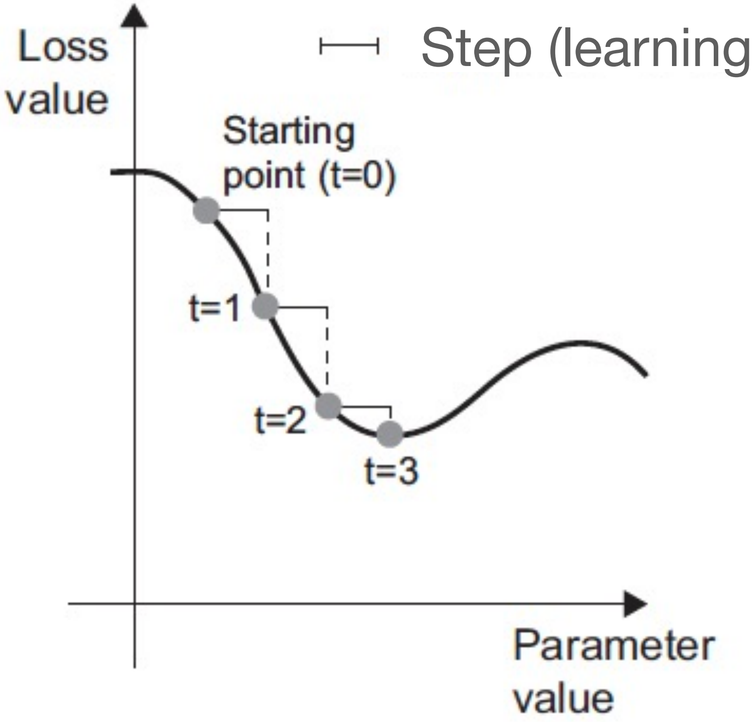


Figure 2.11 in Deep learning with python by Francois Chollet

A single fix learning rate?

SGD maintains a single learning rate, and does not change, but

Magnitudes of gradient

- Different for different weights
- change during training

Problem: difficult to find a single global learning rate

Other optimisation algorithm: Adaptive gradient (AdaGrad)

- Keep a accumulative sum of the squared gradient

$$r_t = r_{t-1} + (\Delta L(w_t))^2 = \sum_{i=1}^{t} \Delta(L(w_i))^2$$

- Divide the gradient with squared root of r_t

$$w_{t+1} = w_t - \eta \frac{\Delta L(w_t)}{\sqrt{r_t + \epsilon}}$$

ϵ : A small constant for numerical stability (1e-7)

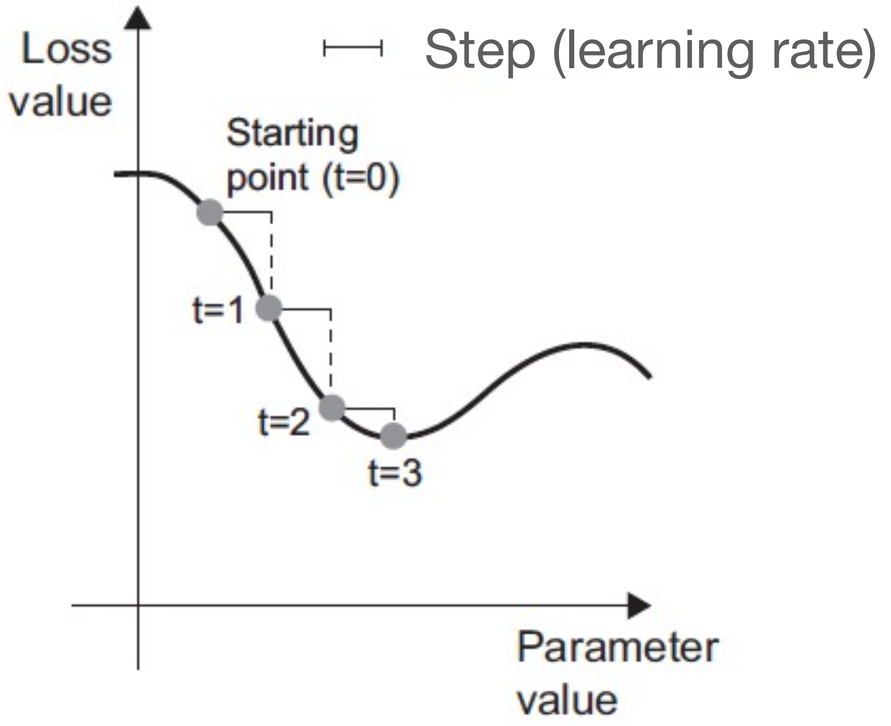


Figure 2.11 in Deep learning with python by Francois Chollet

Other optimisation algorithm: Root Mean Square Propagation (Rmsprop)

- Keep a moving average of the squared gradient

$$r_t = \rho r_{t-1} + (1-\rho)(\Delta L(w_t))^2$$

- Divide the gradient with squared root of r_t

$$w_{t+1} = w_t - \eta \frac{\Delta L(w_t)}{\sqrt{r_t + \epsilon}}$$

ρ : exponential decay parameter (0.9)

ϵ : A small constant for numerical stability (1e-7)

```
opt = tf.keras.optimizers.RMSprop(learning_rate=0.001,
    rho=0.9,
    momentum=0.0,
    epsilon=1e-07)
```

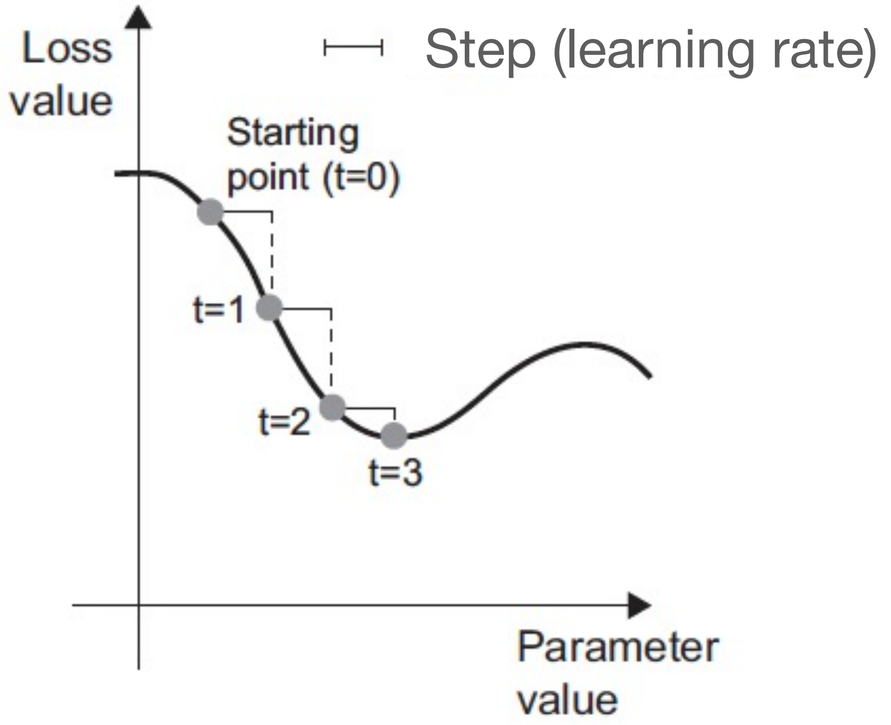


Figure 2.11 in Deep learning with python by Francois Chollet

Other optimisation algorithm: Adaptive moment estimation (Adam)

- Keep a moving average of the gradient and squared gradient

$$m_t = \frac{\beta_1 m_{t-1} + (1-\beta_1)(\Delta L(w_t))}{1-\beta_1^{t+1}}$$

$$r_t = \frac{\beta_2 r_{t-1} + (1-\beta_2)(\Delta L(w_t))^2}{1-\beta_2^{t+1}}$$

Bias correction

- Divide the m_t with squared root of r_t

$$w_{t+1} = w_t - \eta \frac{m_t}{\sqrt{r_t + \epsilon}}$$

β_1, β_2 : exponential decay parameter (0.9 and 0.999)

ϵ : A small constant for numerical stability (1e-7)

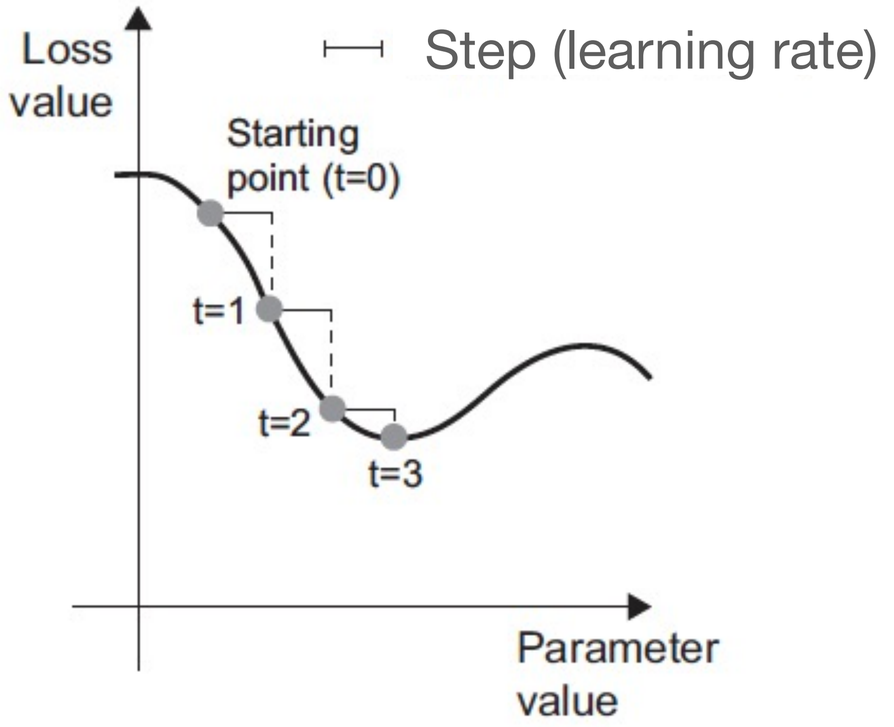
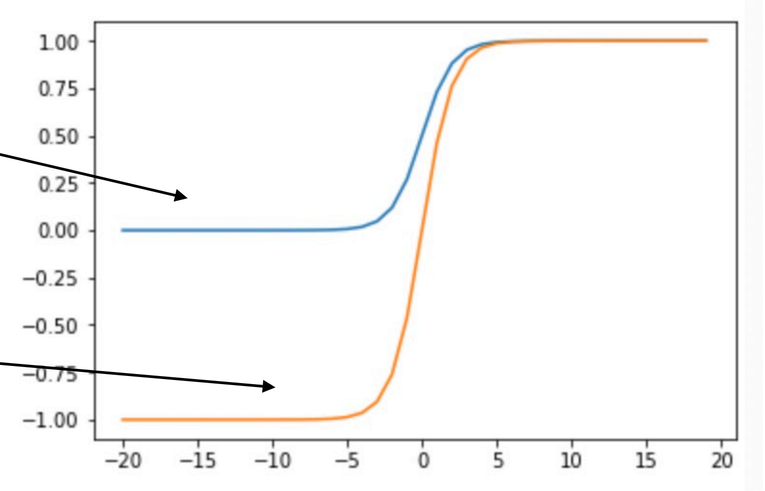


Figure 2.11 in Deep learning with python by Francois Chollet

Activation function

Sigmoid: $f(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$

TanH $f(x) = \frac{2}{1+e^{-2x}} - 1 = \frac{2e^{2x}}{e^{x2}+1} - 1$



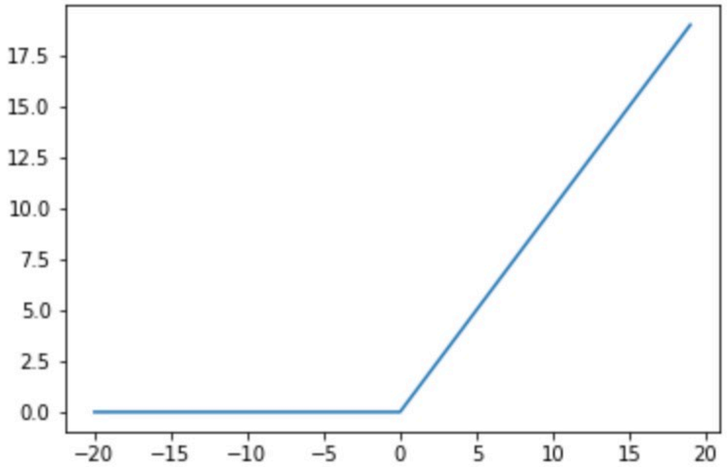
Problem: output saturates if x is very large or very small

➡ Gradient is killed (vanishing gradient problem)

Activation function

Use Relu (Rectified Linear Unit)
to handle vanishing gradient problem :

$$f(x) = \max(0,x)$$



Problem: no gradient for inactive unit
(Dying Relu)

Use Leaky Relu to allow a small gradient for inactive unit:

$$f(x) = x,\ if\ x \geq 0$$
$$f(x) = \alpha x,\ if\ x < 0$$

Prevent overfitting

Early stopping: Stop the training if generalisation error (validation dataset) increases

Explicit regularisation:

- Instead of minimising the loss L, minimise regularised function $L + \lambda\left(\sum_{i=0}^{m}\sum_{j=1}^{p} v_{ij}^2 + \sum_{j=0}^{p} w_j^2\right)$
- This will simply add $2\lambda v_{ij}$ and $2\lambda w_j$ terms to the partial derivatives

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Randomly dropping out (setting to zero) a number of output features of the layer during training

e.g., normal output: [1.3, 0.4, 0.5, 1.5, 3.2, 0.9]
After dropout: [1.3, 0, 0, 1.5, 3.2, 0]

Dropout rate: the fraction of the units that are zeroed out (0.2-0.5).
No units are dropped out during testing

Summary

- How to perform convolution? How to compute output size after convolution? Advantages & comparison between convolutional layer and fully connected layer?
- How to perform max-pooling? Advantages of max-pooling?
- What problem does momentum solve in SGD? How do RMSprop and Adam update weights?
- What’s the advantage of (Leaky) ReLU compared to sigmoid and tanh?
- How to prevent overfitting problem in network training?

Next：CNN Architectures

References

- Hinton: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf
- Deep learning with python by Francois Chollet, chapter 2.4.3, 5.1
- Deep Learning, by Ian Goodfellow and Yoshua Bengio and Aaron Courville, chapter 9.2-9.3
- Pattern Recognition and Machine Learning by Chris Bishop, chapter 5.5.6