First steps to deep-learning

From the perceptron to the deep generative models

Nicolas Desassis

nicolas.desassis@minesparis.psl.eu

January 2022

Sources

- Deep Learning course for MINES ParisTech
  (E. Decenciere, T. Walter and S. Velasco-Forero)
- Machine Learning course for MINES ParisTech
  (C.A. Azincott)
- MOOC on Deep Learning specialization (Andrew Ng)
- Various blogs

Outline

- Metrics for classifications
- Prevent overfitting
- Convolutional network principle
- Some classical architectures
- Transfer learning
- Keras

Metrics for classification

Predicted \ Observed	Positive	Negative
Positive	TP	FP
Negative	FN	TN

Number of

- T: True
- F: False
- P: Positive (1)
- F: Negative (0)

Metrics for classification

Predicted \ Observed	Positive	Negative
Positive	TP	FP
Negative	FN	TN

Number of

- T: True
- F: False
- P: Positive (1)
- F: Negative (0)

Accuracy

$$\frac{TP + TN}{TP + TN + FP + FN}$$

Metrics for classification

Predicted \ Observed	Positive	Negative
Positive	TP	FP
Negative	FN	TN

Number of

- T: True
- F: False
- P: Positive (1)
- F: Negative (0)

Sensitivity (or recall)

$$\frac{TP}{FN + TP}$$

Metrics for classification

Predicted \ Observed	Positive	Negative
Positive	TP	FP
Negative	FN	TN

Number of

- T: True
- F: False
- P: Positive (1)
- F: Negative (0)

Precision (or Positive Predicted Value, PPV)

$$\frac{TP}{TP + FP}$$

Metrics for classification

Predicted \ Observed	Positive	Negative
Positive	TP	FP
Negative	FN	TN

Number of

- T: True
- F: False
- P: Positive (1)
- F: Negative (0)

F-Score (or F1-Score)

$$\frac{2\text{Precision.Recall}}{\text{Precision+Recall}} = \frac{2\,TP}{2\,TP + FP + FN}$$

Metrics for classification

Predicted \ Observed	Positive	Negative
Positive	TP	FP
Negative	FN	TN

Number of

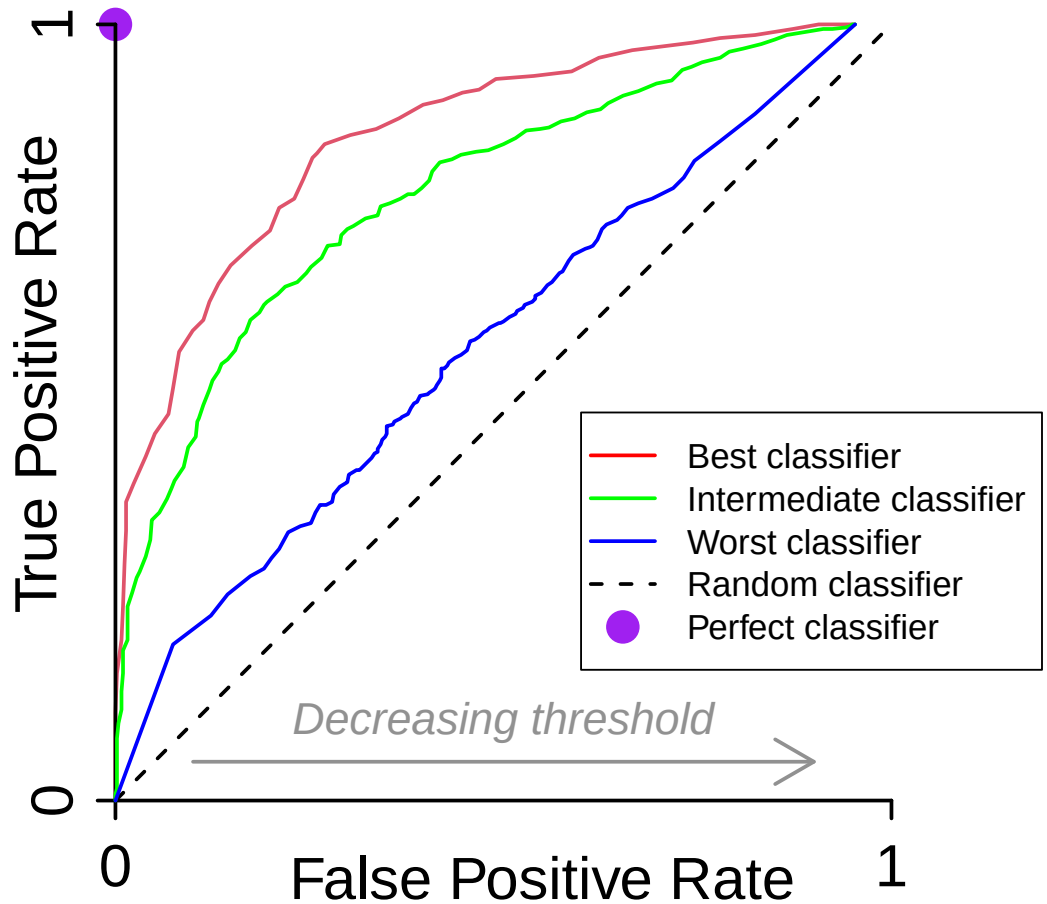
- T: True
- F: False
- P: Positive (1)
- F: Negative (0)

Specificity

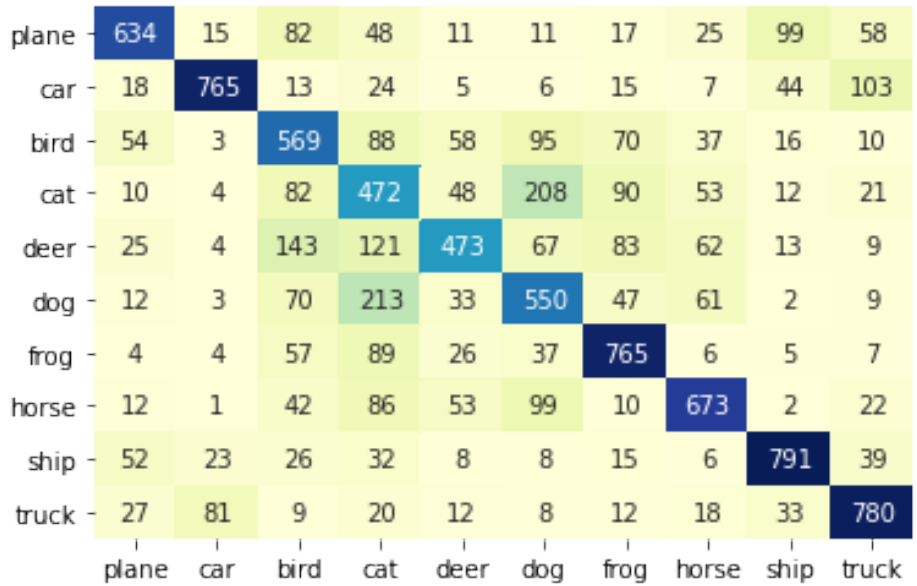
$$\frac{TN}{FP + TN}$$

ROC curve

Receiver Operating Characteristic



Confusion matrix



- For multi-class classification N_{ij} is the number of examples for which j is predicted and i is the true class.
- One can express the criterions for binary case for each class (one vs. all)

Machine learning: a piece of theory (Vapnik, 2013)

Machine learning: a piece of theory (Vapnik, 2013)

- We suppose that the examples (data)

$$(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)$$

are independant and identically distributed according to an unknown distribution $P(\mathbf{x}, \mathbf{y})$

Machine learning: a piece of theory (Vapnik, 2013)

- We suppose that the examples (data)

$$(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)$$

  are independant and identically distributed according to an unknown distribution $P(\mathbf{x}, \mathbf{y})$
- We will denote f_0 the optimal supervisor (classifior or predictor), the function which minimizes the risk to predict $\mathbf{y}$ with $f_0(\mathbf{x})$:

$$\mathcal{L}(f) = E_P[\Delta(\mathbf{y}, f(\mathbf{x}))] = \int \Delta(\mathbf{y}, f(\mathbf{x}))\, dP(\mathbf{x}, \mathbf{y})$$

Machine learning: a piece of theory (Vapnik, 2013)

- We suppose that the examples (data)

$$(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)$$

  are independant and identically distributed according to an unknown distribution $P(\mathbf{x}, \mathbf{y})$
- We will denote f_0 the optimal supervisor (classifior or predictor), the function which minimizes the risk to predict $\mathbf{y}$ with $f_0(\mathbf{x})$:

$$\mathcal{L}(f) = E_P[\Delta(\mathbf{y}, f(\mathbf{x}))] = \int \Delta(\mathbf{y}, f(\mathbf{x}))\, dP(\mathbf{x}, \mathbf{y})$$

- We try to find a function $f^\star$ which minimizes the risk over a class $\mathcal{H}$ (e.g linear functions, NN with 1 hidden layer,...)

Empirircal Risk Minimization

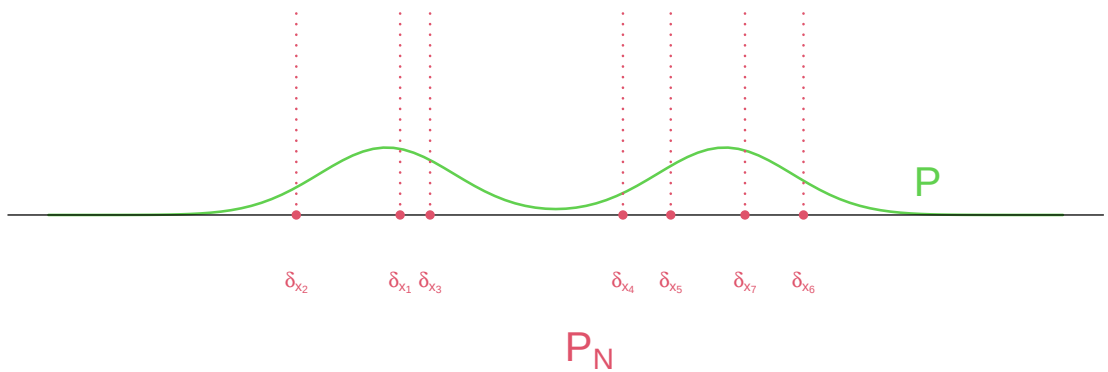
$$\mathcal{L}(f) = E_P[\Delta(\mathbf{y}, f(\mathbf{x}))] = \int \Delta(\mathbf{y}, f(\mathbf{x}))\, dP(\mathbf{x}, \mathbf{y})$$

Empirircal Risk Minimization

$$\mathcal{L}(f) = E_P[\Delta(\mathbf{y}, f(\mathbf{x}))] = \int \Delta(\mathbf{y}, f(\mathbf{x}))\, dP(\mathbf{x}, \mathbf{y})$$

- Since P is unknown, we replace it by the empirical distribution

$$P_N(\mathbf{x}, y) = \frac{1}{N} \sum_{i=1}^{N} \delta(\mathbf{x} = \mathbf{x}_i, \mathbf{y} = \mathbf{y}_i)$$



Empirircal Risk Minimization

$$\mathcal{L}(f) = E_P[\Delta(\mathbf{y}, f(\mathbf{x}))] = \int \Delta(\mathbf{y}, f(\mathbf{x}))\, dP(\mathbf{x}, \mathbf{y})$$

- Since P is unknown, we replace it by the empirical distribution

$$P_N(\mathbf{x}, y) = \frac{1}{N} \sum_{i=1}^{N} \delta(\mathbf{x} = \mathbf{x}_i, \mathbf{y} = \mathbf{y}_i)$$

and $\mathcal{L}$ by the empirical risk

$$\mathcal{L}_N(f) = E_{P_N}[\Delta(\mathbf{y}, f(\mathbf{x}))] = \int \Delta(\mathbf{y}, f(\mathbf{x}))\, dP_N(\mathbf{x}, \mathbf{y})$$

$$= \frac{1}{N} \sum_{i=1}^{N} \Delta(\mathbf{y}_i, f(\mathbf{x}_i))$$

which is minimized over $\mathcal{H}$ by f_N

Empirircal Risk Minimization

$$\mathcal{L}(f) = E_P[\Delta(\mathbf{y}, f(\mathbf{x}))] = \int \Delta(\mathbf{y}, f(\mathbf{x}))\, dP(\mathbf{x}, \mathbf{y})$$

- Since P is unknown, we replace it by the empirical distribution

$$P_N(\mathbf{x}, y) = \frac{1}{N} \sum_{i=1}^{N} \delta(\mathbf{x} = \mathbf{x}_i, \mathbf{y} = \mathbf{y}_i)$$

and $\mathcal{L}$ by the empirical risk

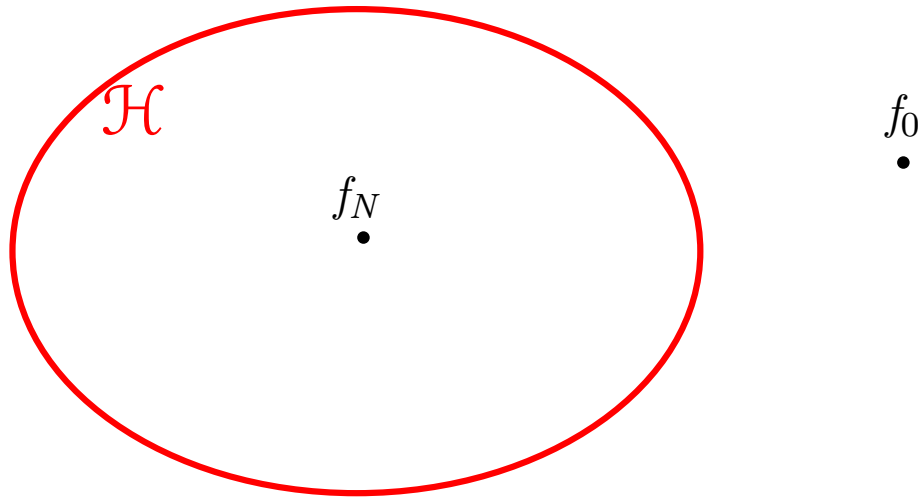
$$\mathcal{L}_N(f) = E_{P_N}[\Delta(\mathbf{y}, f(\mathbf{x}))] = \int \Delta(\mathbf{y}, f(\mathbf{x}))\, dP_N(\mathbf{x}, \mathbf{y})$$

$$= \frac{1}{N} \sum_{i=1}^{N} \Delta(\mathbf{y}_i, f(\mathbf{x}_i))$$

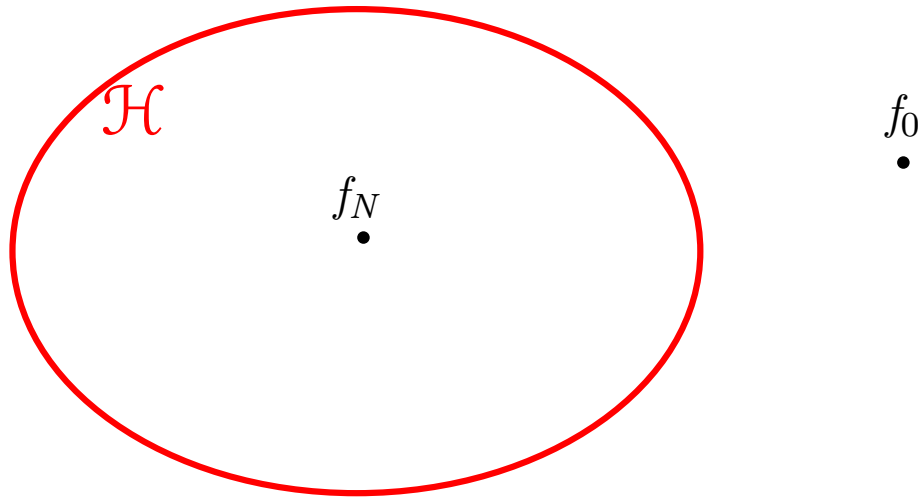
which is minimized over $\mathcal{H}$ by f_N

- This is the Empirical Risk Minimization (ERM)

Empirical Risk Minimization



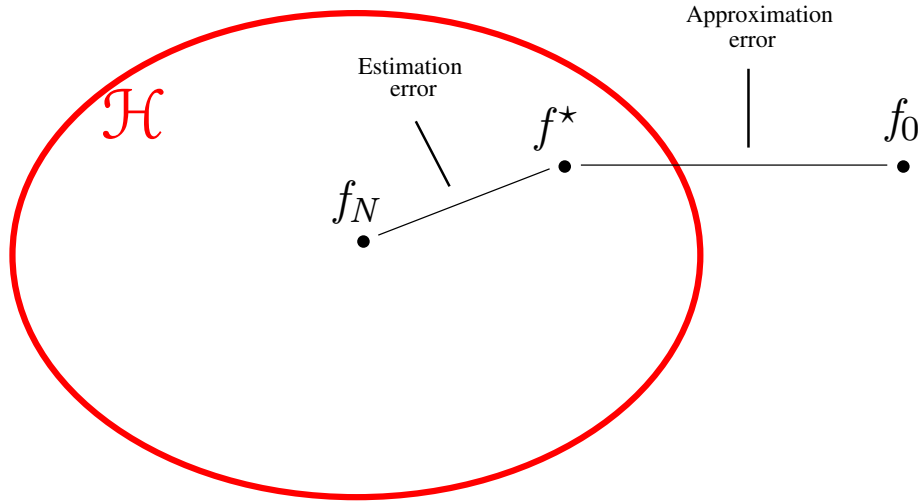
Empirical Risk Minimization



- The total error

$$\mathcal{L}(f_N) - \mathcal{L}(f_0)$$

Empirical Risk Minimization

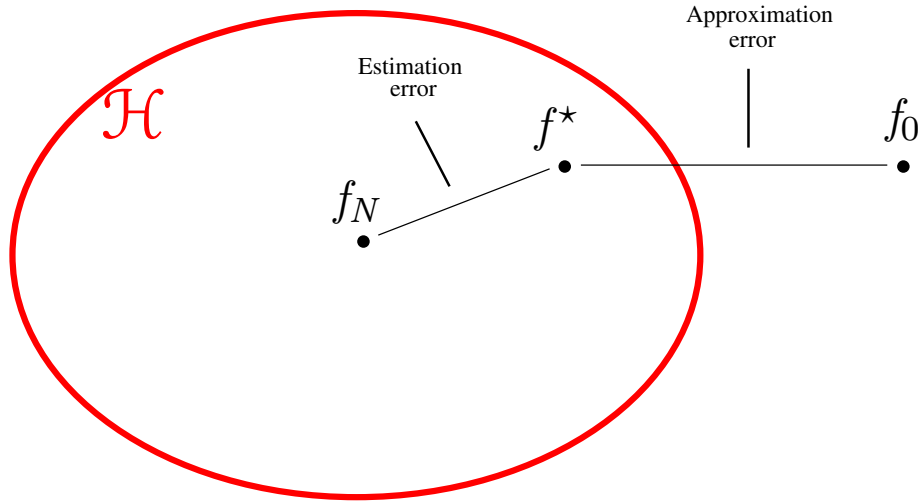


- The total error

$$\mathcal{L}(f_N) - \mathcal{L}(f_0) = \underbrace{[\mathcal{L}(f_N) - \mathcal{L}(f^\star)]}_{Variance \geq 0} + \underbrace{[\mathcal{L}(f^\star) - \mathcal{L}(f_0)]}_{Bias \geq 0}$$

where $f^\star$ is the minimizer of $\mathcal{L}$ over $\mathcal{H}$

Empirical Risk Minimization

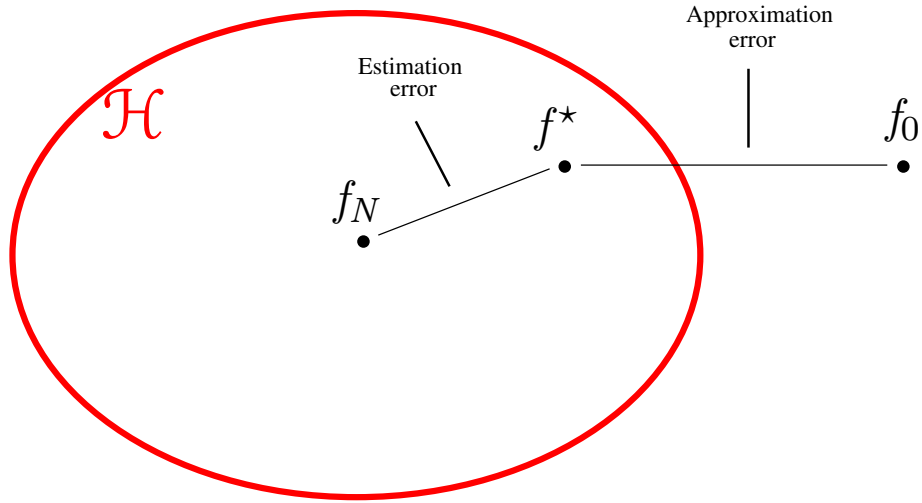


- The approximation error (or bias term)

$$[\mathcal{L}(f^\star) - \mathcal{L}(f_0)]$$

does not depend on the data but only on the model complexity (choice of $\mathcal{H}$)

Empirical Risk Minimization



- The approximation error (or bias term)

$$[\mathcal{L}(f^\star) - \mathcal{L}(f_0)]$$

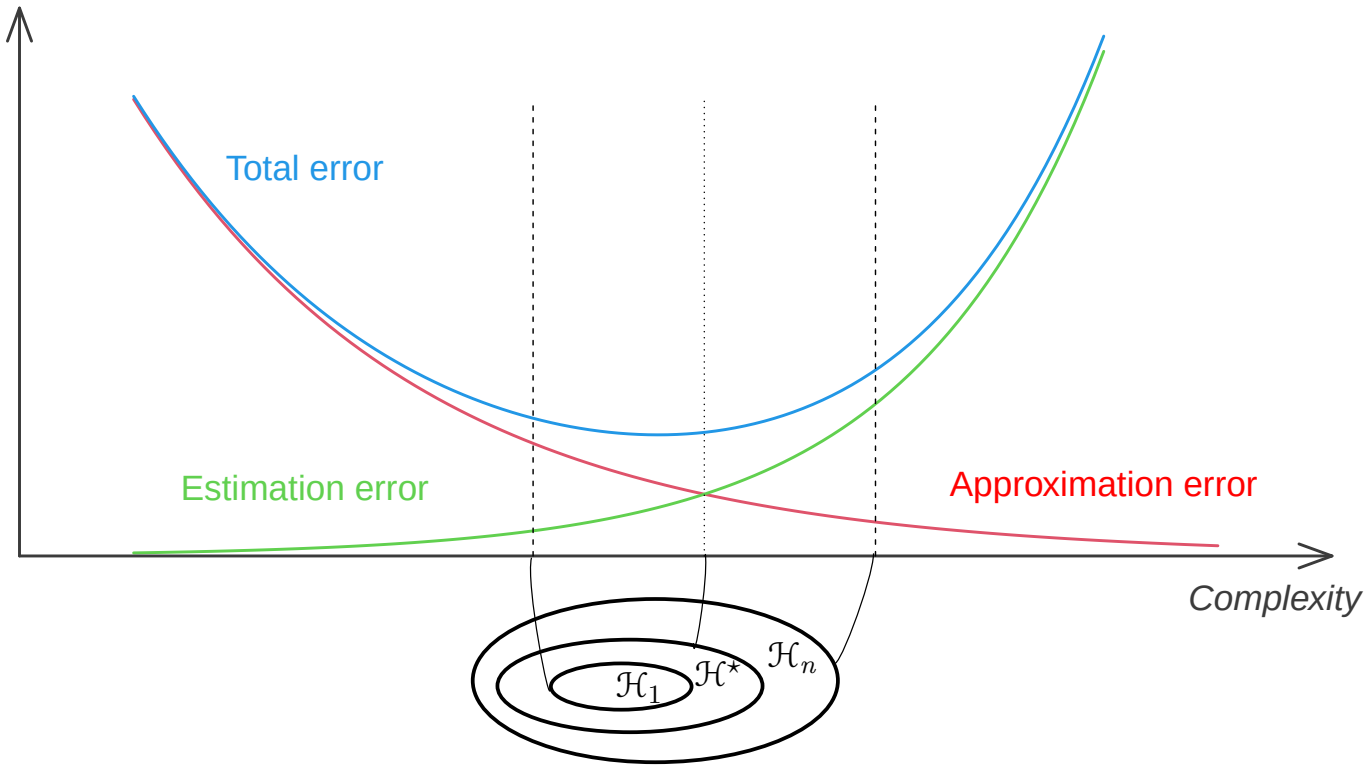
does not depend on the data but only on the model complexity (choice of $\mathcal{H}$)

- The estimation error (or variance term)

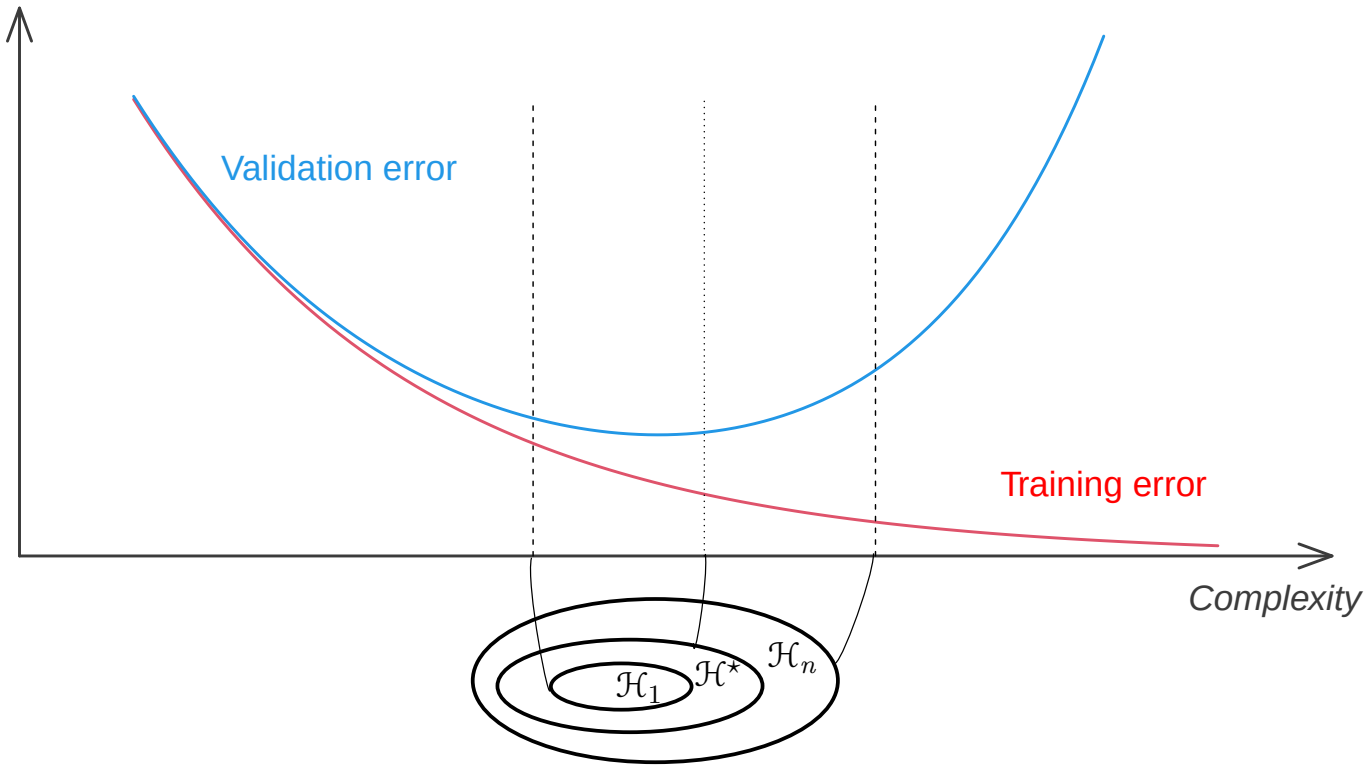
$$[\mathcal{L}(f_N) - \mathcal{L}(f^\star)]$$

increases with the model complexity and decreases when the data set size N increases

Bias-Variance trade-off

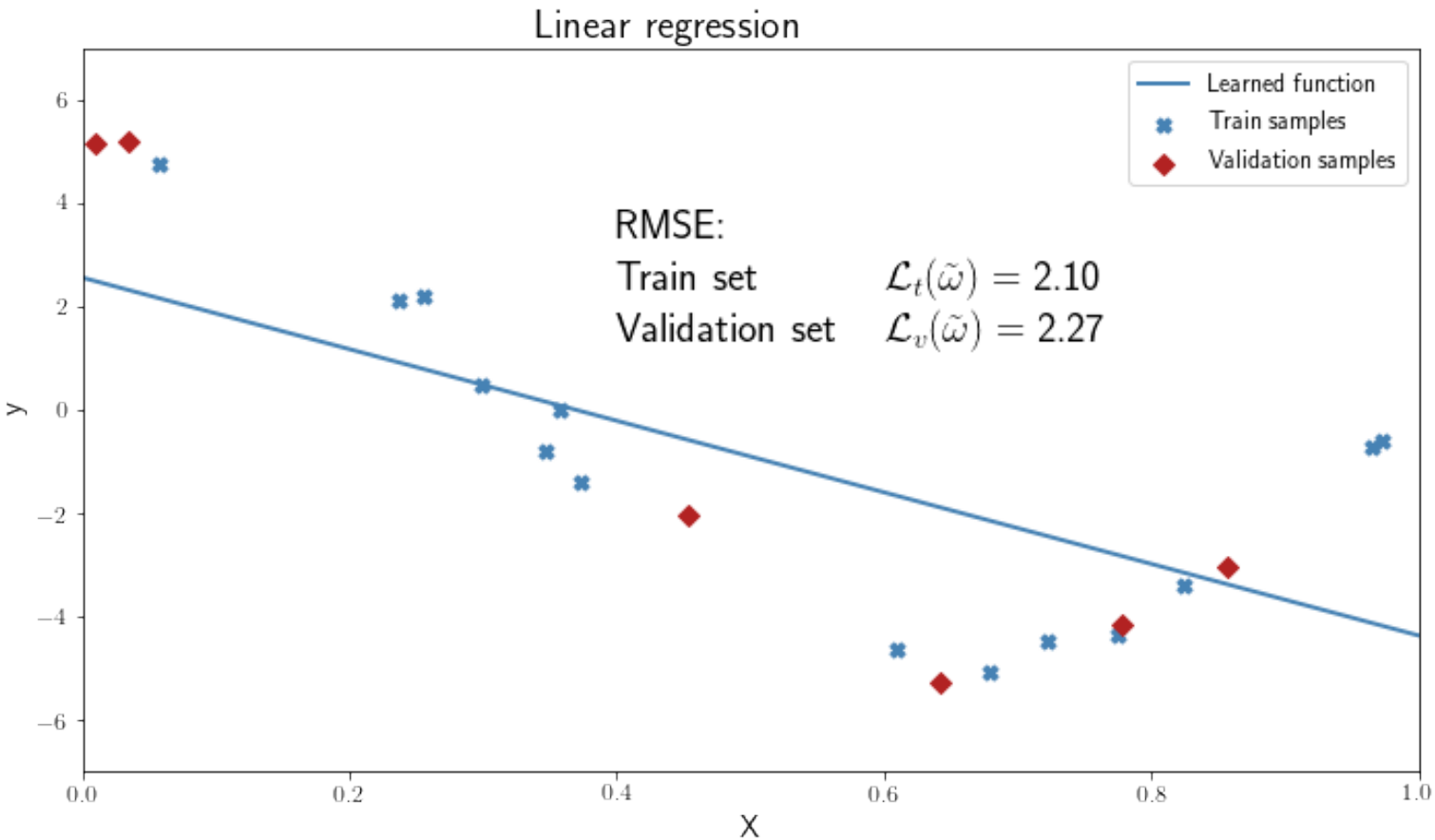


In practice


Validation error
Training error
Complexity
$\mathcal{H}_1$
$\mathcal{H}^\star$
$\mathcal{H}_n$

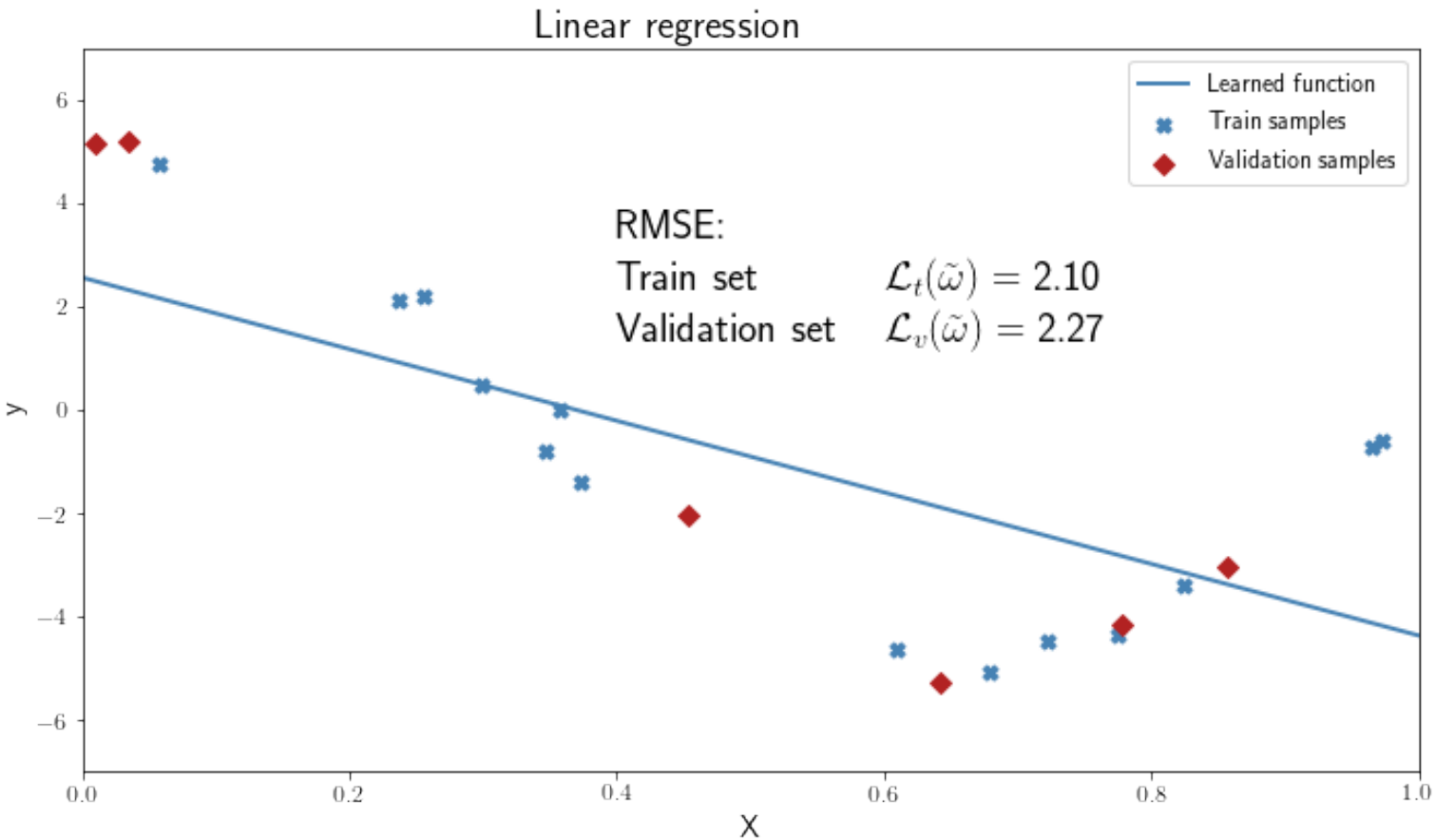
Regression

$y = \omega_0 + \omega_1 x$



Regression

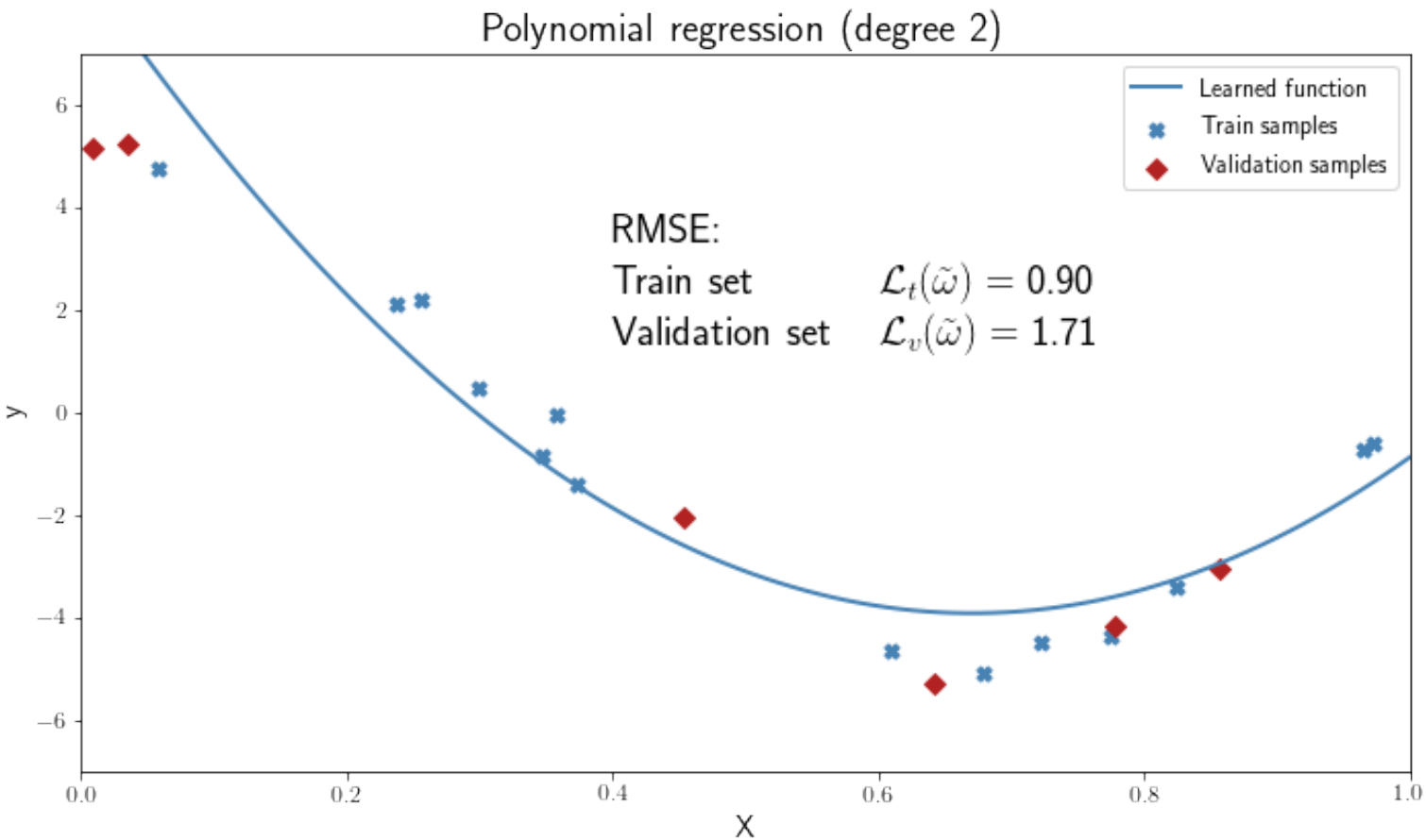
$y = \omega_0 + \omega_1 x$



Underfitting or lack of complexity of the model

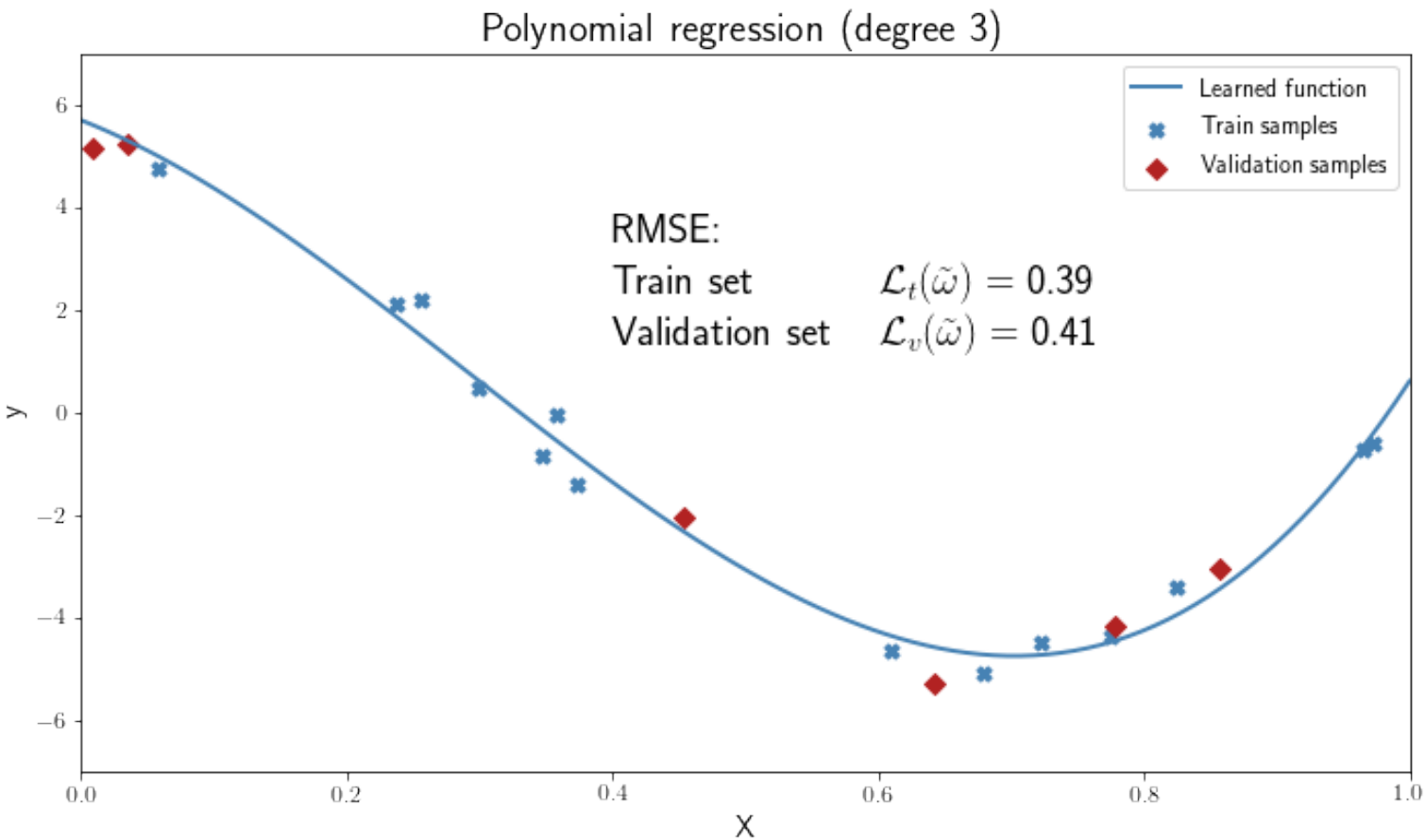
Polynomial regression

$$y = \omega_0 + \sum_{i=1}^{2} \omega_i x^i$$



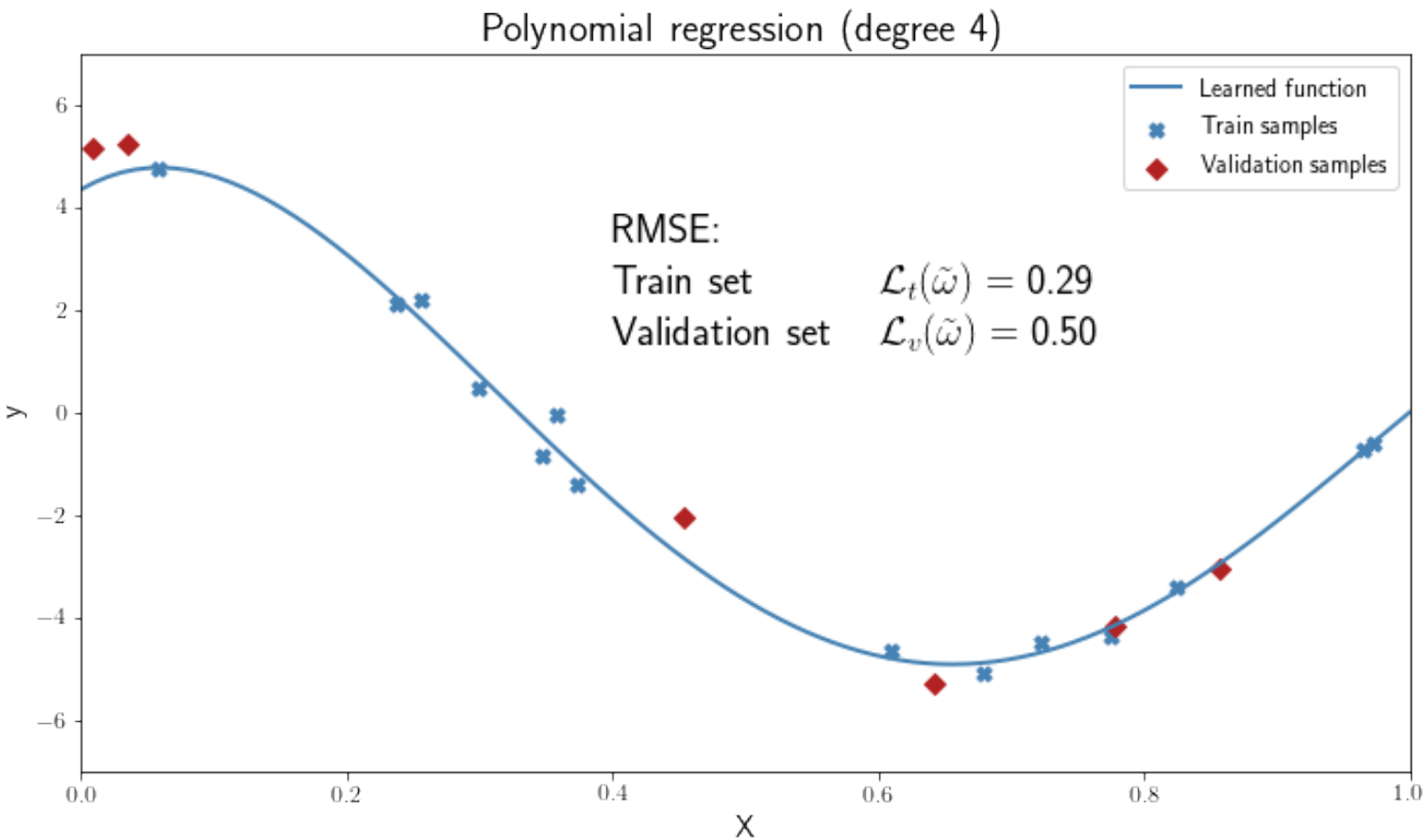
Polynomial regression

$$y = \omega_0 + \sum_{i=1}^{3} \omega_i x^i$$



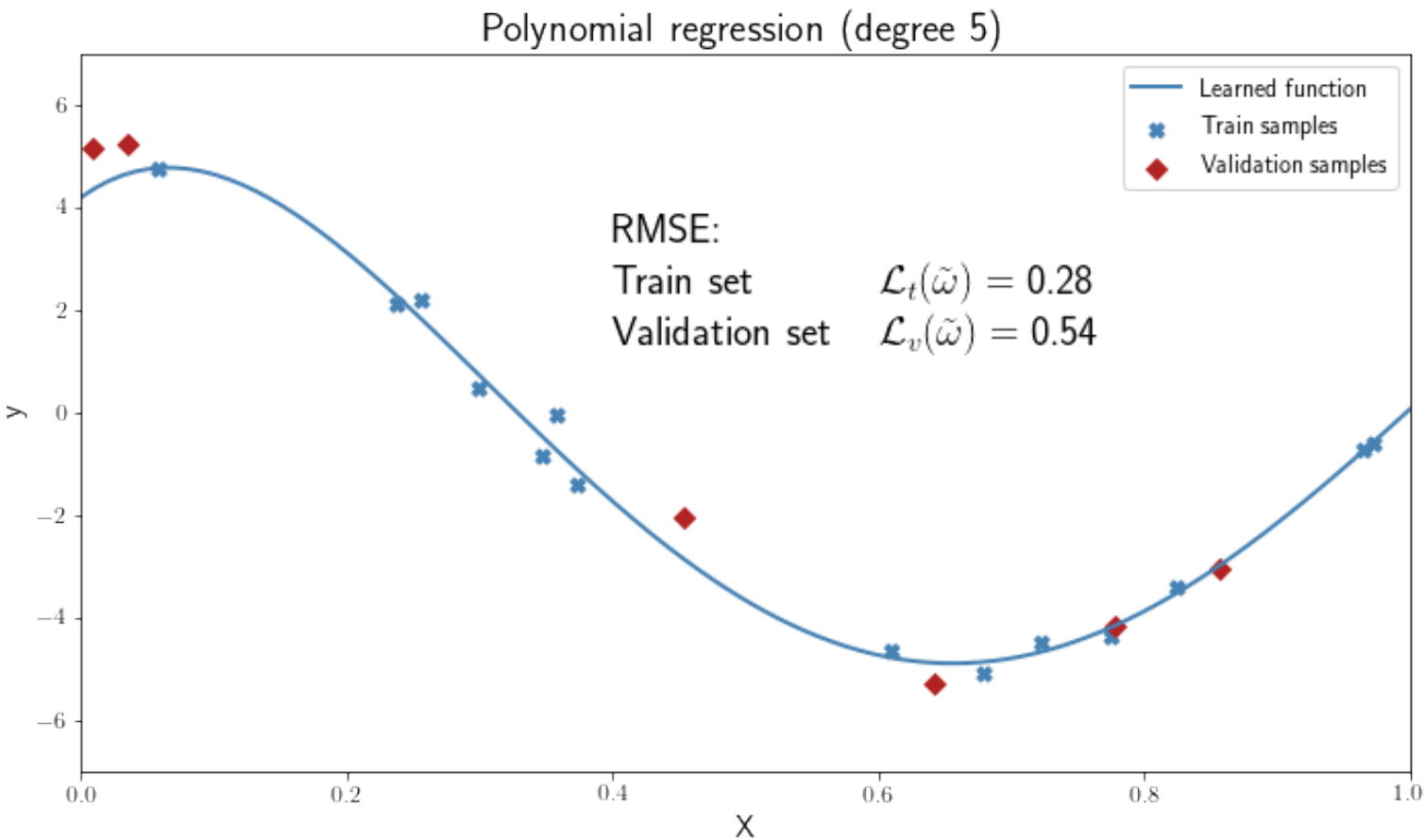
Polynomial regression

$$y = \omega_0 + \sum_{i=1}^{4} \omega_i x^i$$



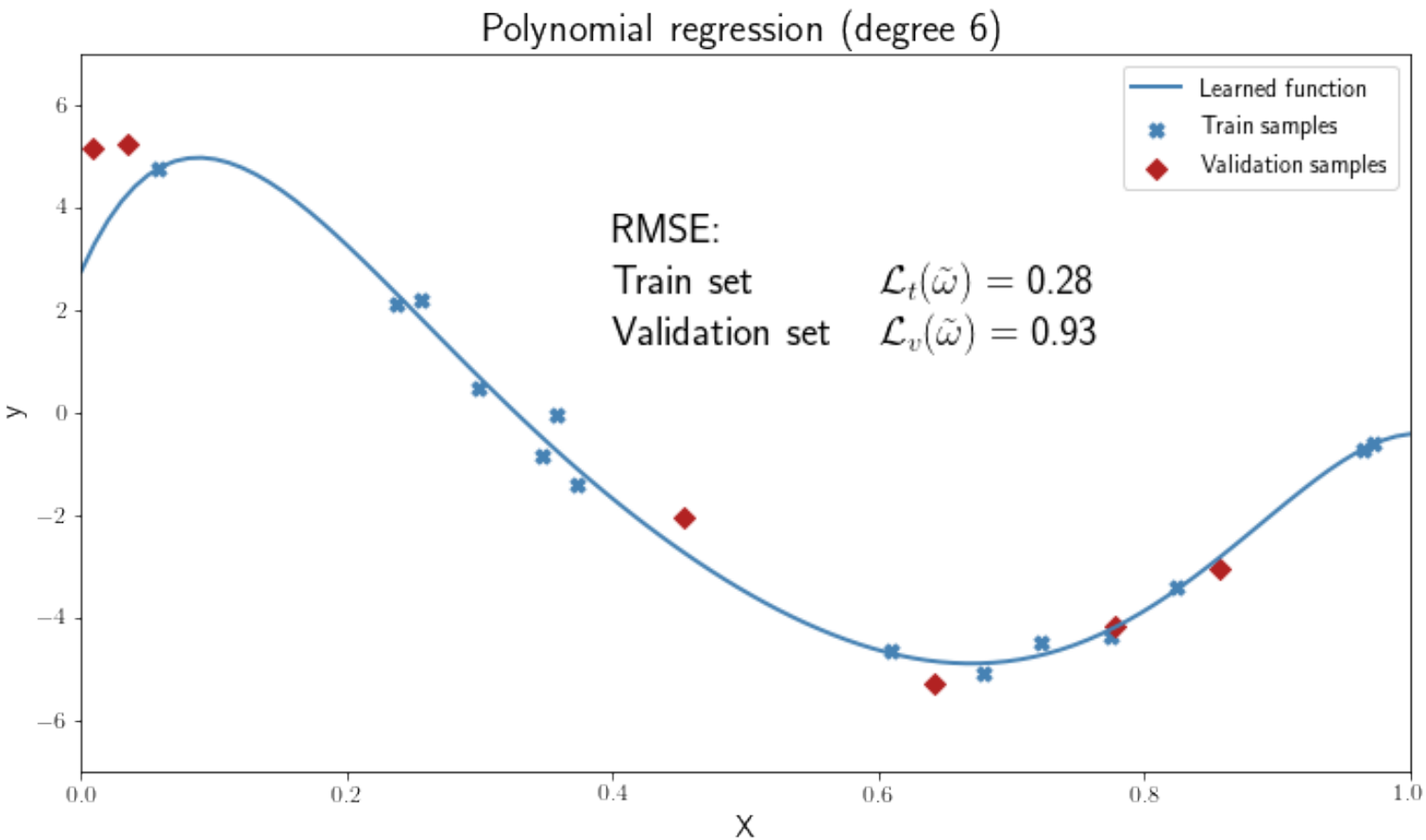
Polynomial regression

$$y = \omega_0 + \sum_{i=1}^{5} \omega_i x^i$$



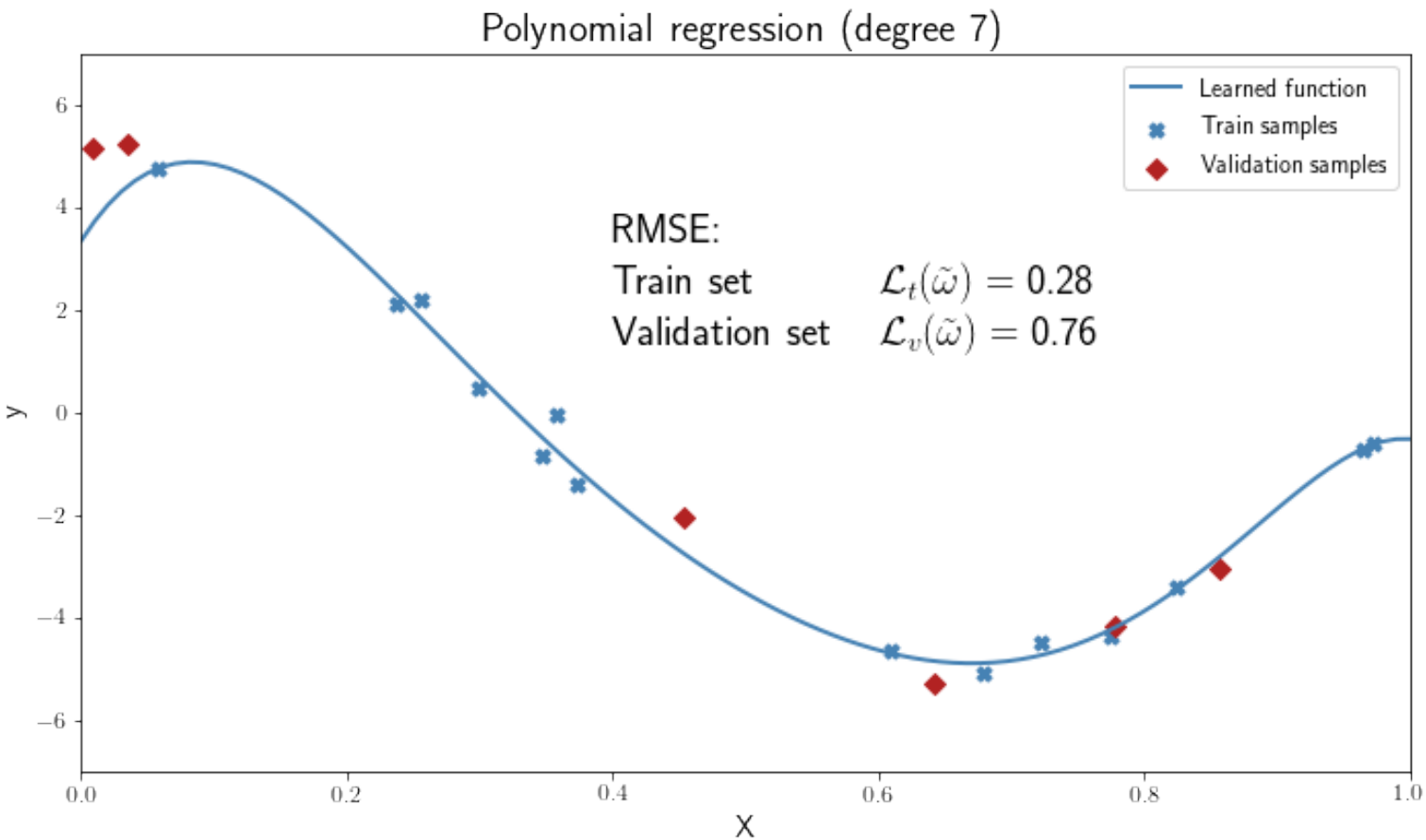
Polynomial regression

$$y = \omega_0 + \sum_{i=1}^{6} \omega_i x^i$$



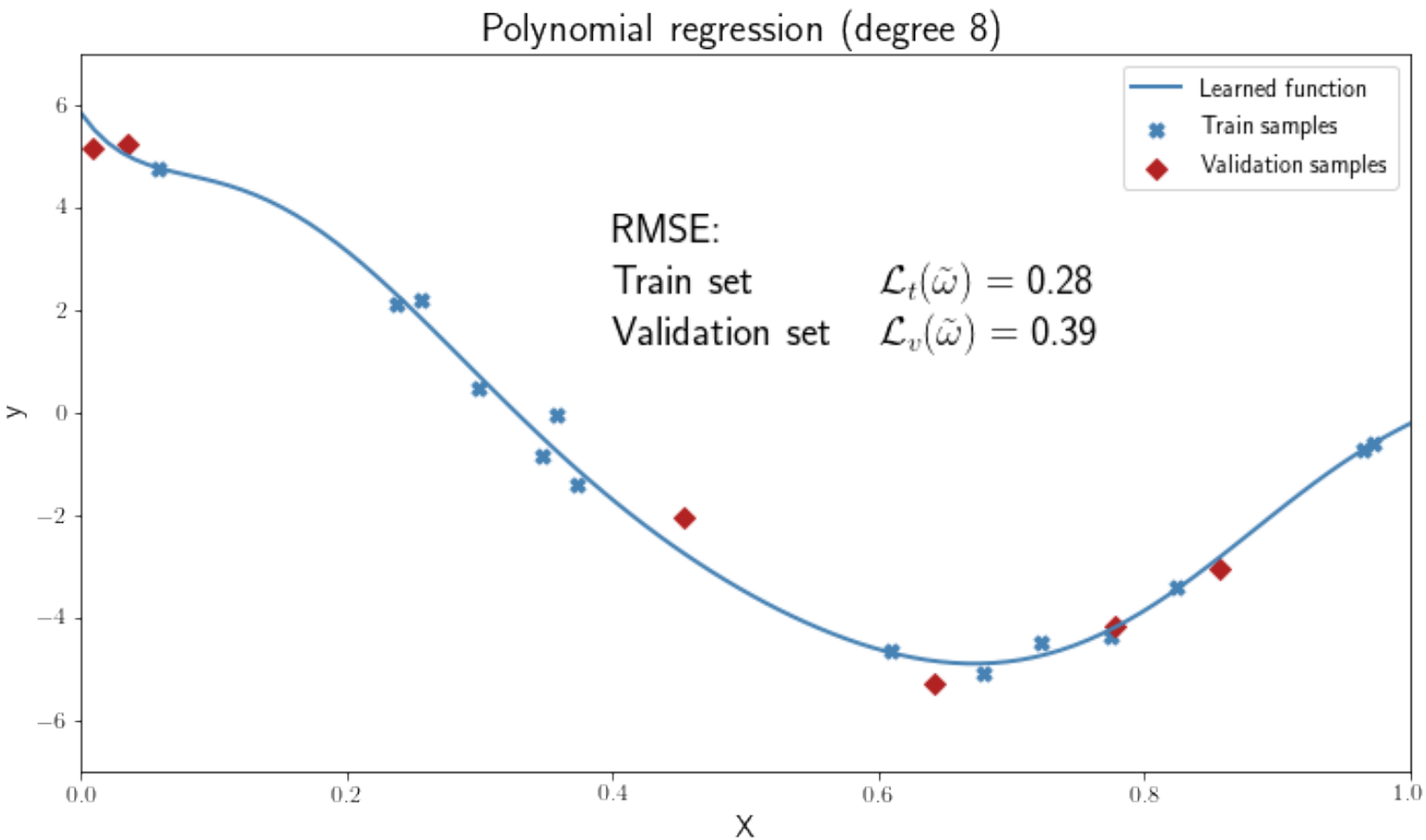
Polynomial regression

$$y = \omega_0 + \sum_{i=1}^{7} \omega_i x^i$$



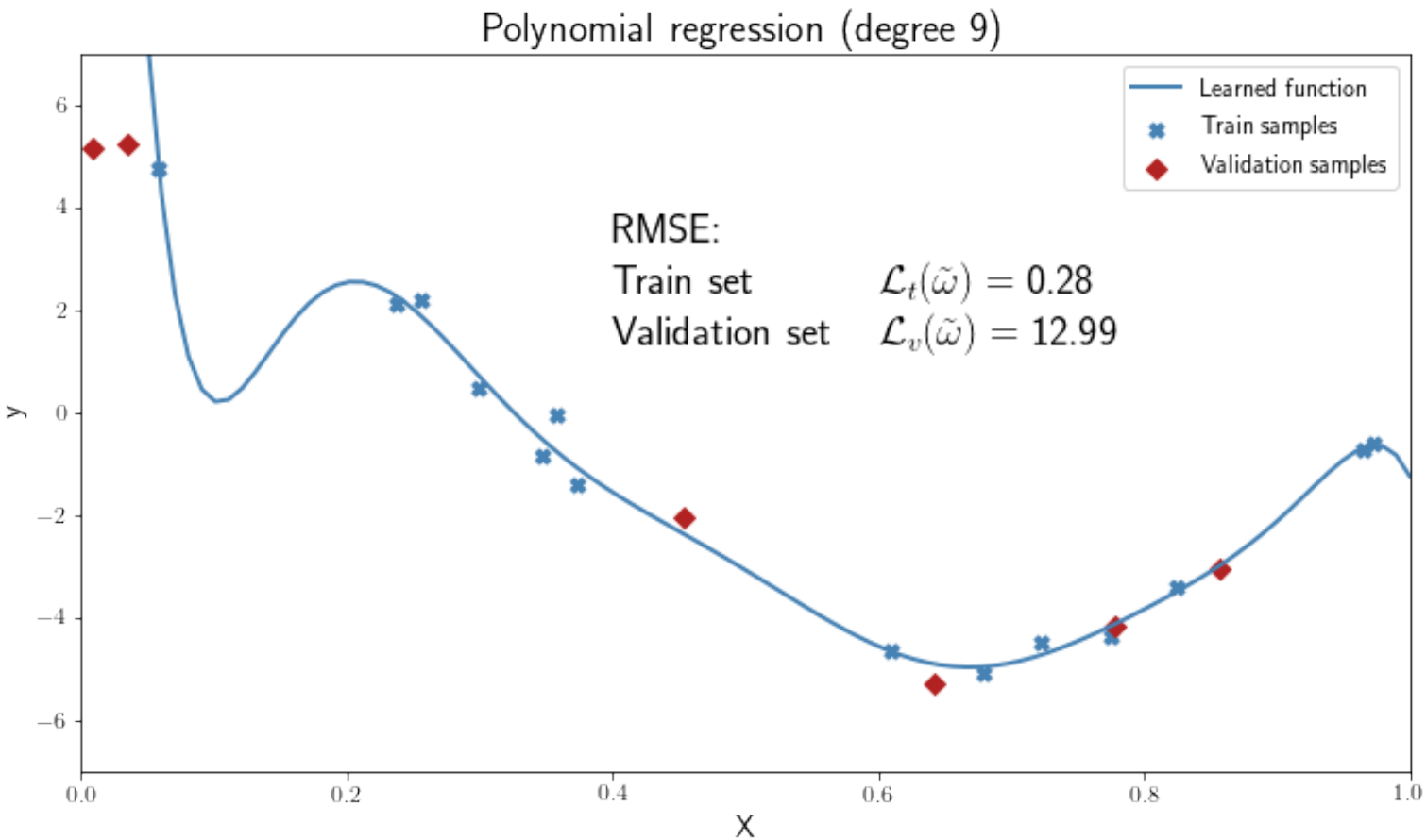
Polynomial regression

$$y = \omega_0 + \sum_{i=1}^{8} \omega_i x^i$$



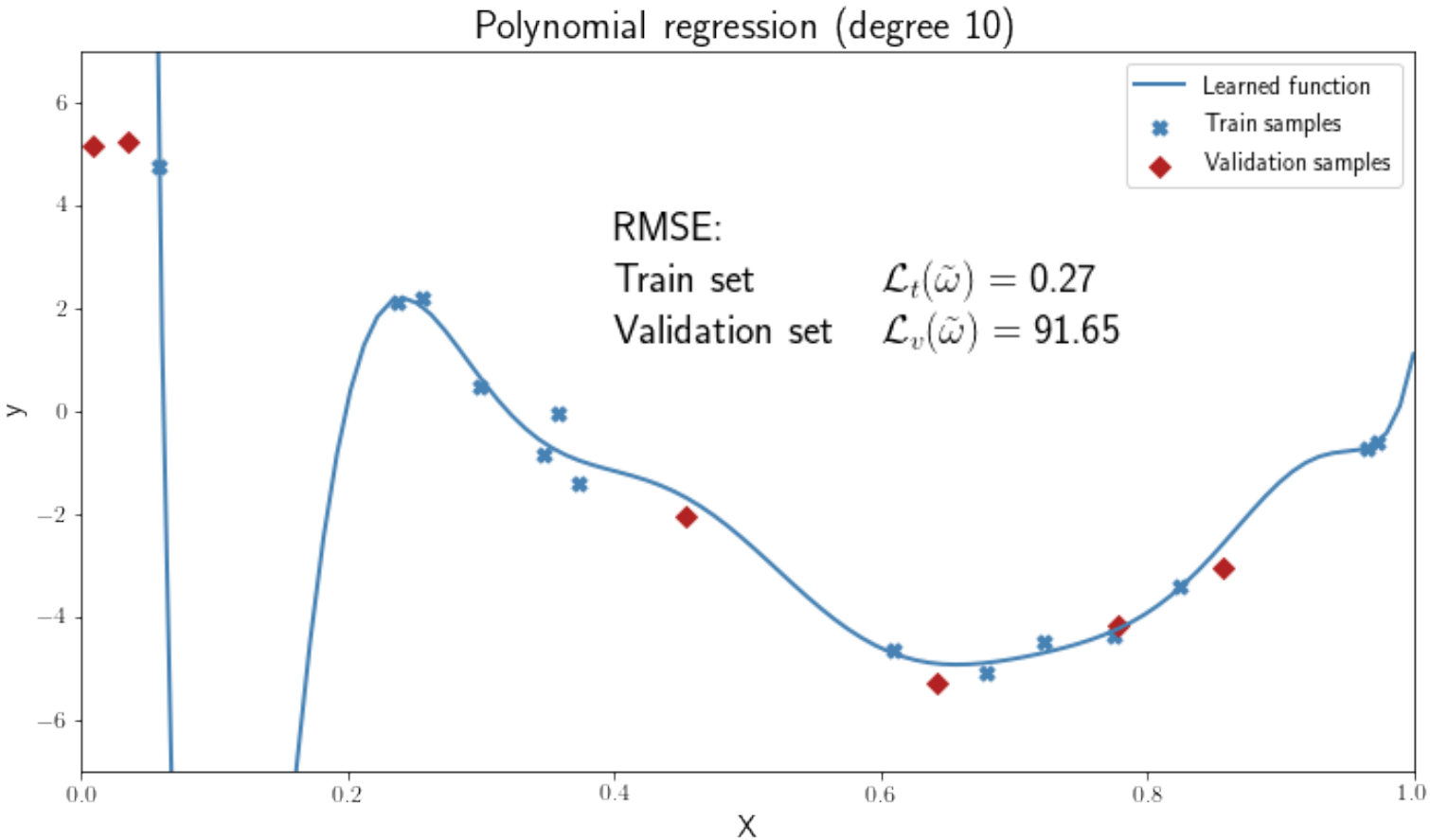
Polynomial regression

$$y = \omega_0 + \sum_{i=1}^{9} \omega_i x^i$$



Polynomial regression

$$y = \omega_0 + \sum_{i=1}^{10} \omega_i x^i$$



General methodology against overfitting

- With neural networks, it is more difficult to increase complexity with one single parameter (e.g polynomial degree)

General methodology against overfitting

- With neural networks, it is more difficult to increase complexity with one single parameter (e.g polynomial degree)
- With high dimensional inputs and large datasets, we have no prior idea about the necessary complexity order to "fit" our dataset and outliers are more difficult to detect

General methodology against overfitting

- With neural networks, it is more difficult to increase complexity with one single parameter (e.g polynomial degree)
- With high dimensional inputs and large datasets, we have no prior idea about the necessary complexity order to "fit" our dataset and outliers are more difficult to detect
- We want to allow high-level of complexity while preventing overfitting and without human effort

General methodology against overfitting

- With neural networks, it is more difficult to increase complexity with one single parameter (e.g polynomial degree)
- With high dimensional inputs and large datasets, we have no prior idea about the necessary complexity order to "fit" our dataset and outliers are more difficult to detect
- We want to allow high-level of complexity while preventing overfitting and without human effort

We want to prevent overfitting with few hyper-parameters and automatic procedures to obtain them (cross-validation on validation set with grid-searches)

Add a penalty term

- $\mathcal{L}$ is the loss-function (or loss)
- We consider the cost-function

$$\mathcal{C}(\mathbf{W}) = \mathcal{L}(\mathbf{W}) + \lambda \mathcal{R}(\mathbf{W})$$

where λ is an hyper-parameter, the regularization rate and $\mathcal{R}$ is a function which penalizes the complexity.

L_2 penalty

$$\mathcal{R}(\mathbf{W}) = \sum_{l=1}^{L} \sum_{i=1}^{n_l} \sum_{j=0}^{n_{l-1}} {\omega_{ij}^{(l)}}^2$$

- Example: Ridge regression

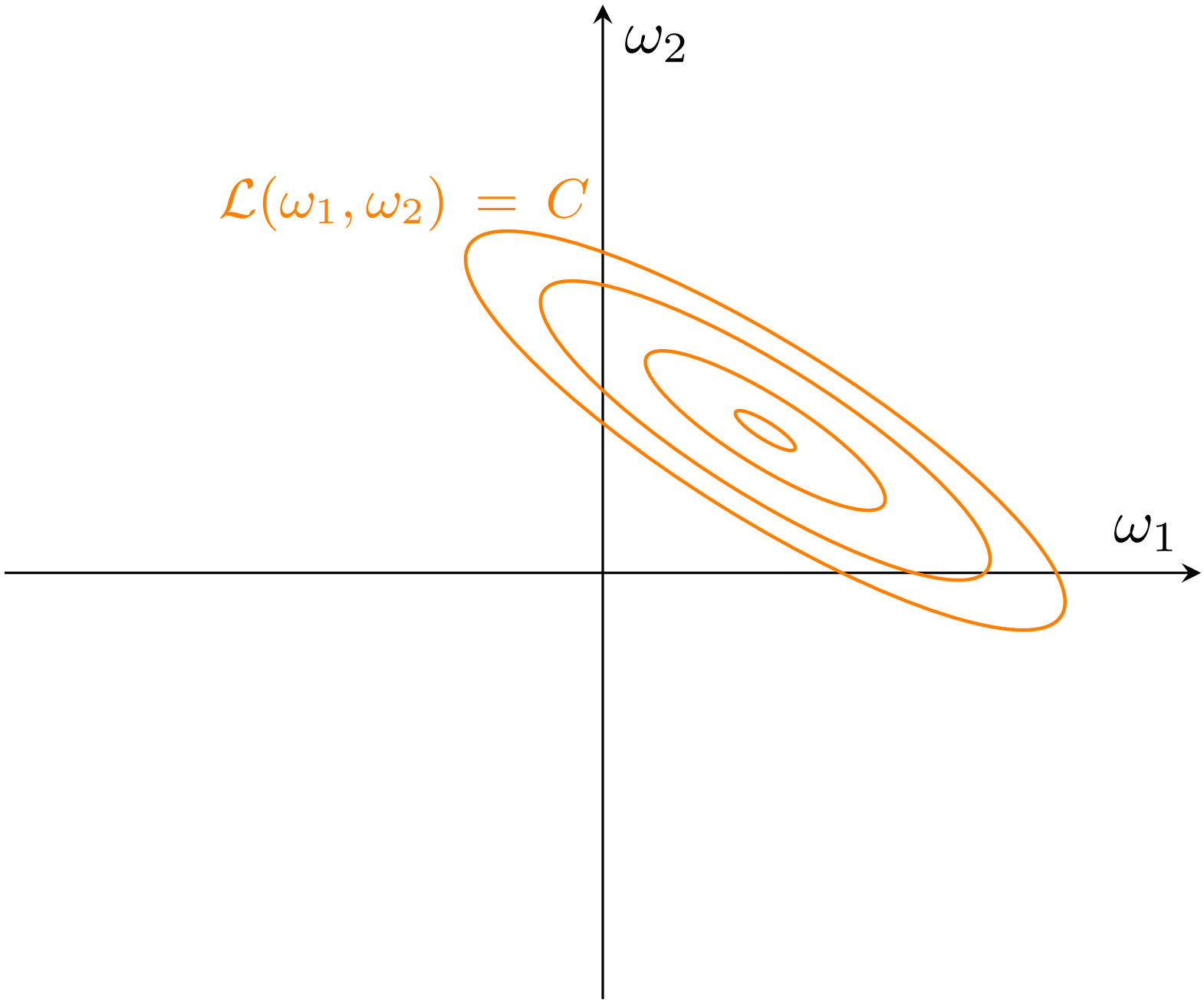
$$\tilde{\omega}^\star = \arg\min_{\tilde{\omega}} ||\mathbf{y} - \tilde{\omega}^t \tilde{X}||_2^2 + \lambda ||\tilde{\omega}||_2^2$$

- Solution

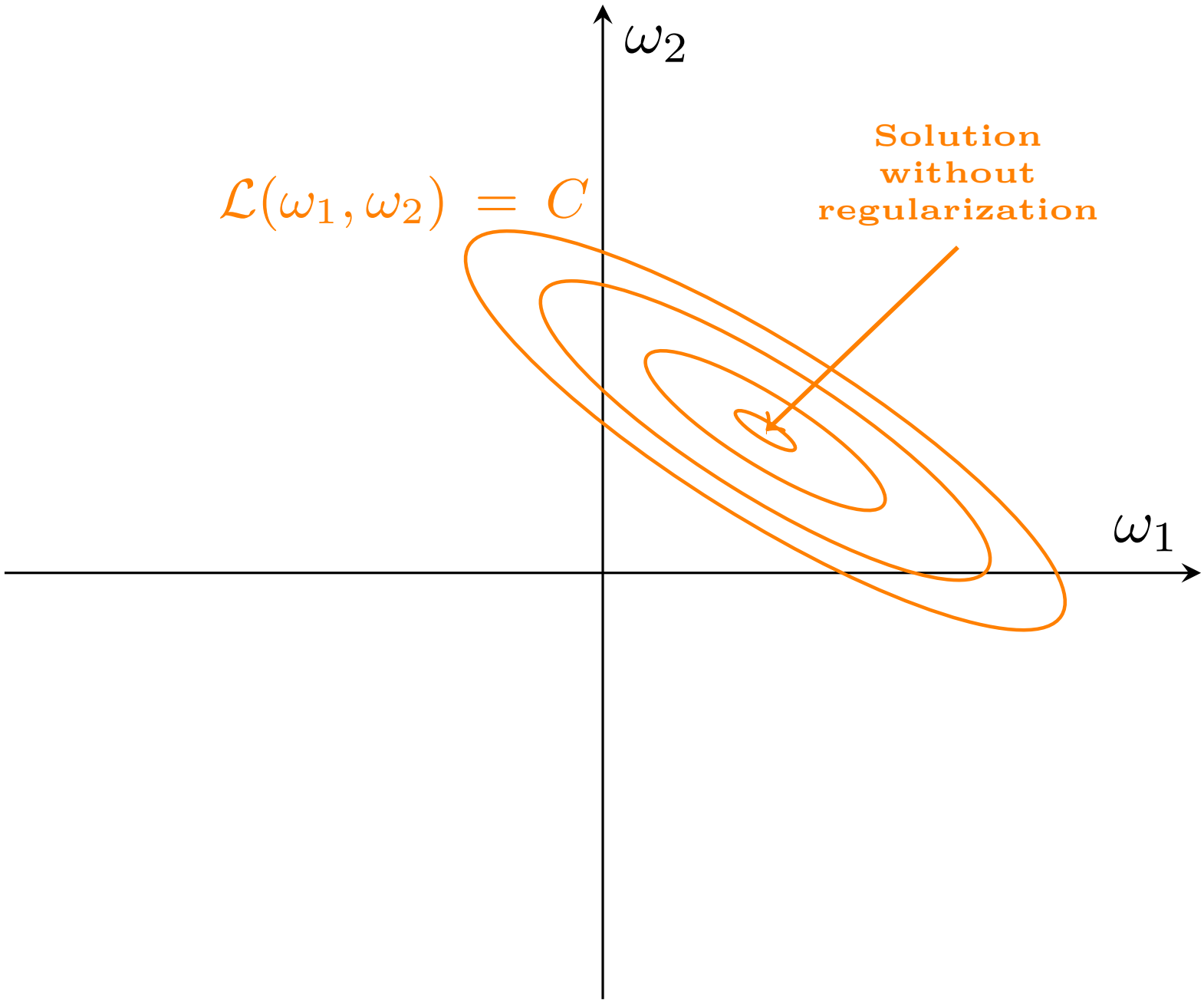
$$\tilde{\omega}^\star = (\tilde{X}^t \tilde{X} + \lambda I)^{-1} \tilde{X}^t \mathbf{y}$$

- Bayesian interpretation: $\tilde{\omega}^\star$ is the maximum a posteriori (MAP) when the residual is Gaussian and the weights have a Gaussian (with variance controlled by λ)

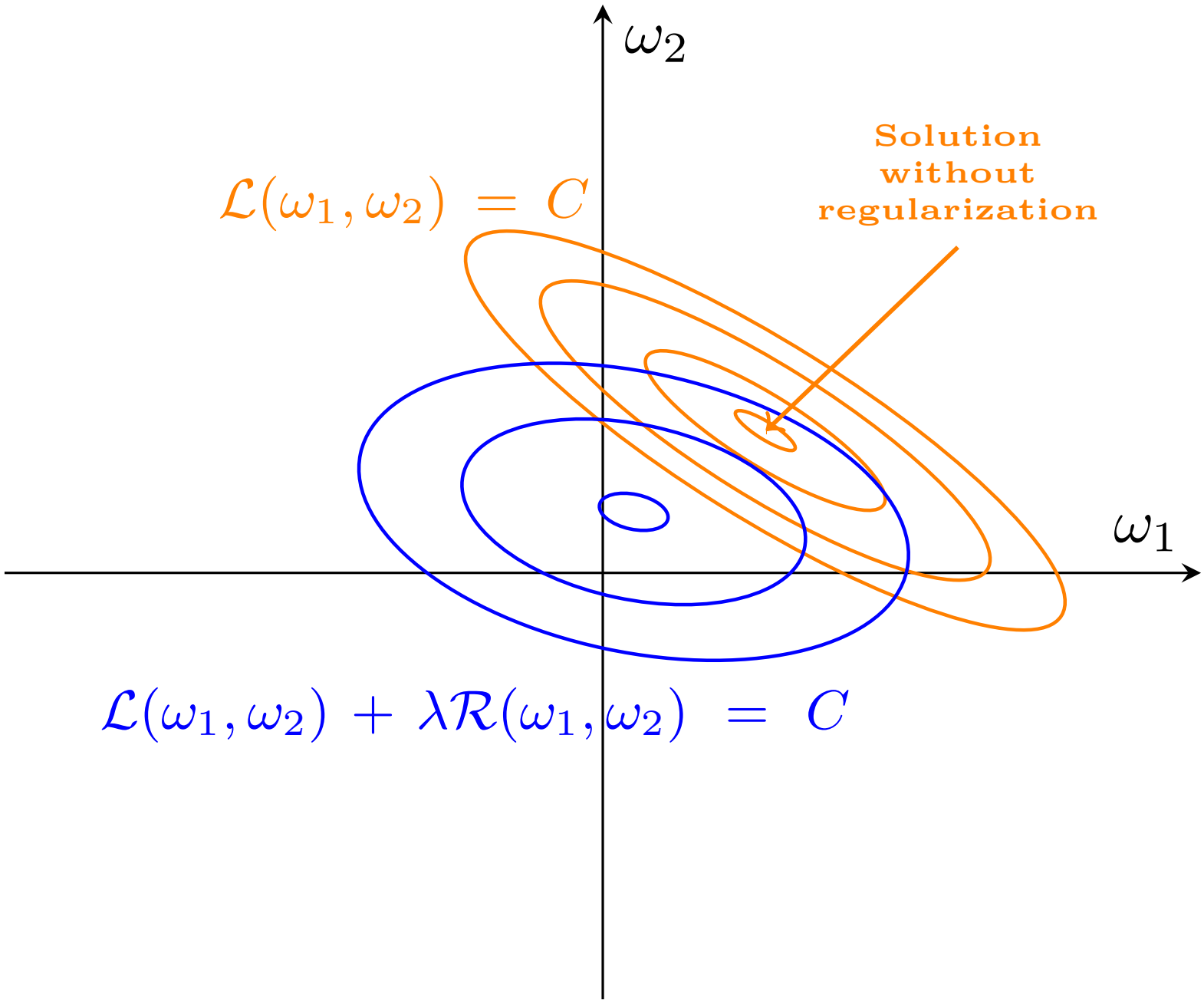
Illustration



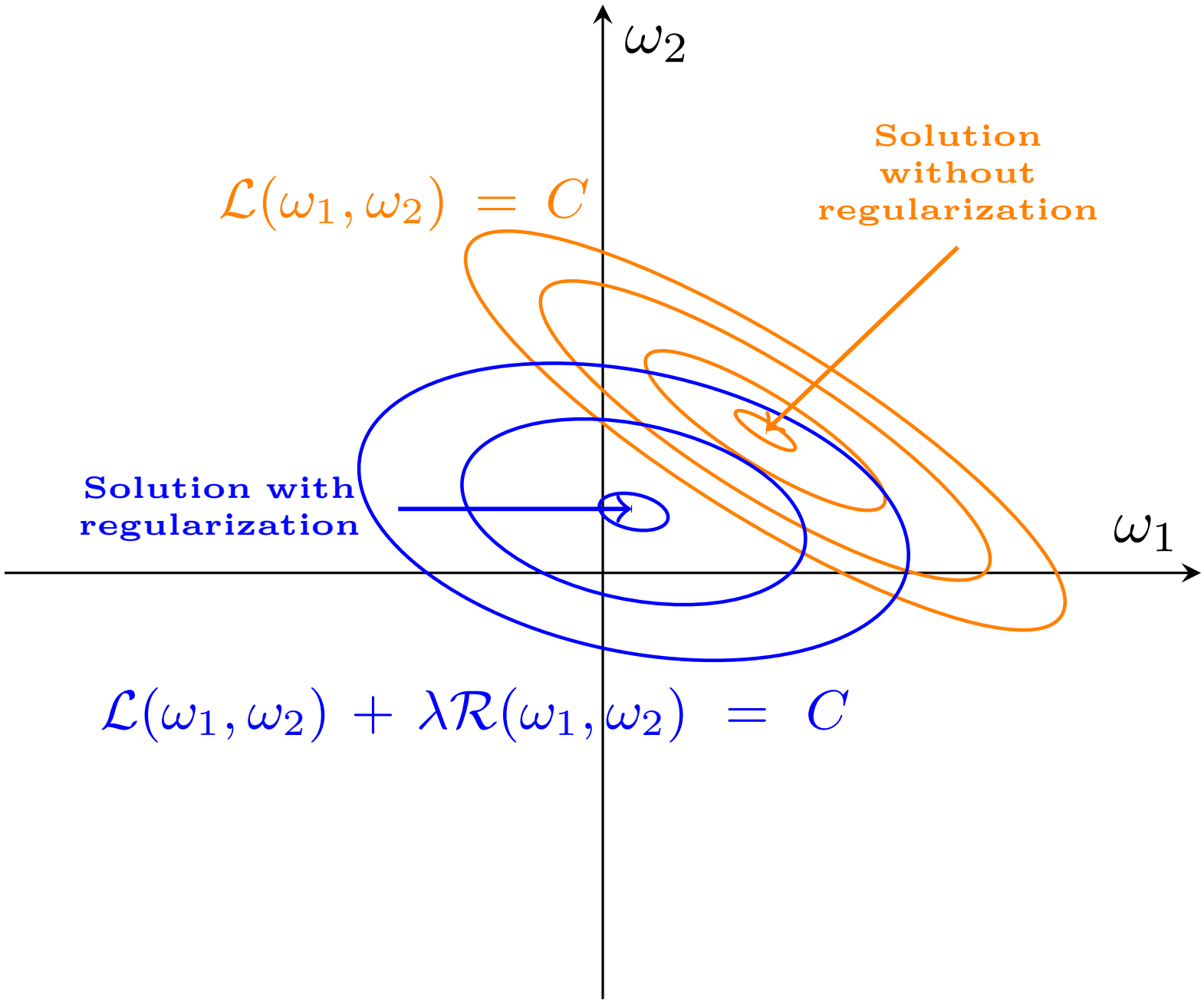
Illustration



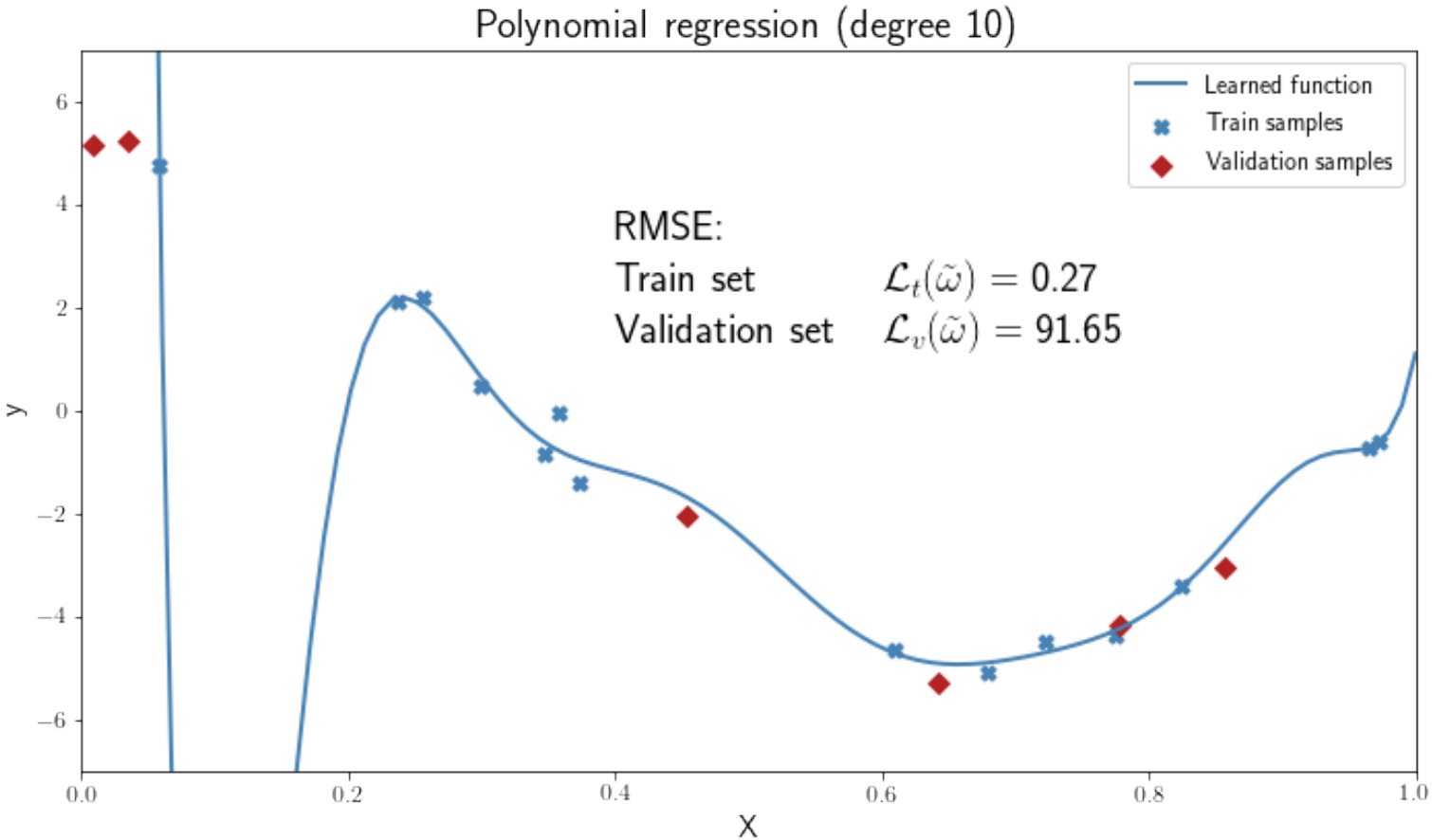
Illustration



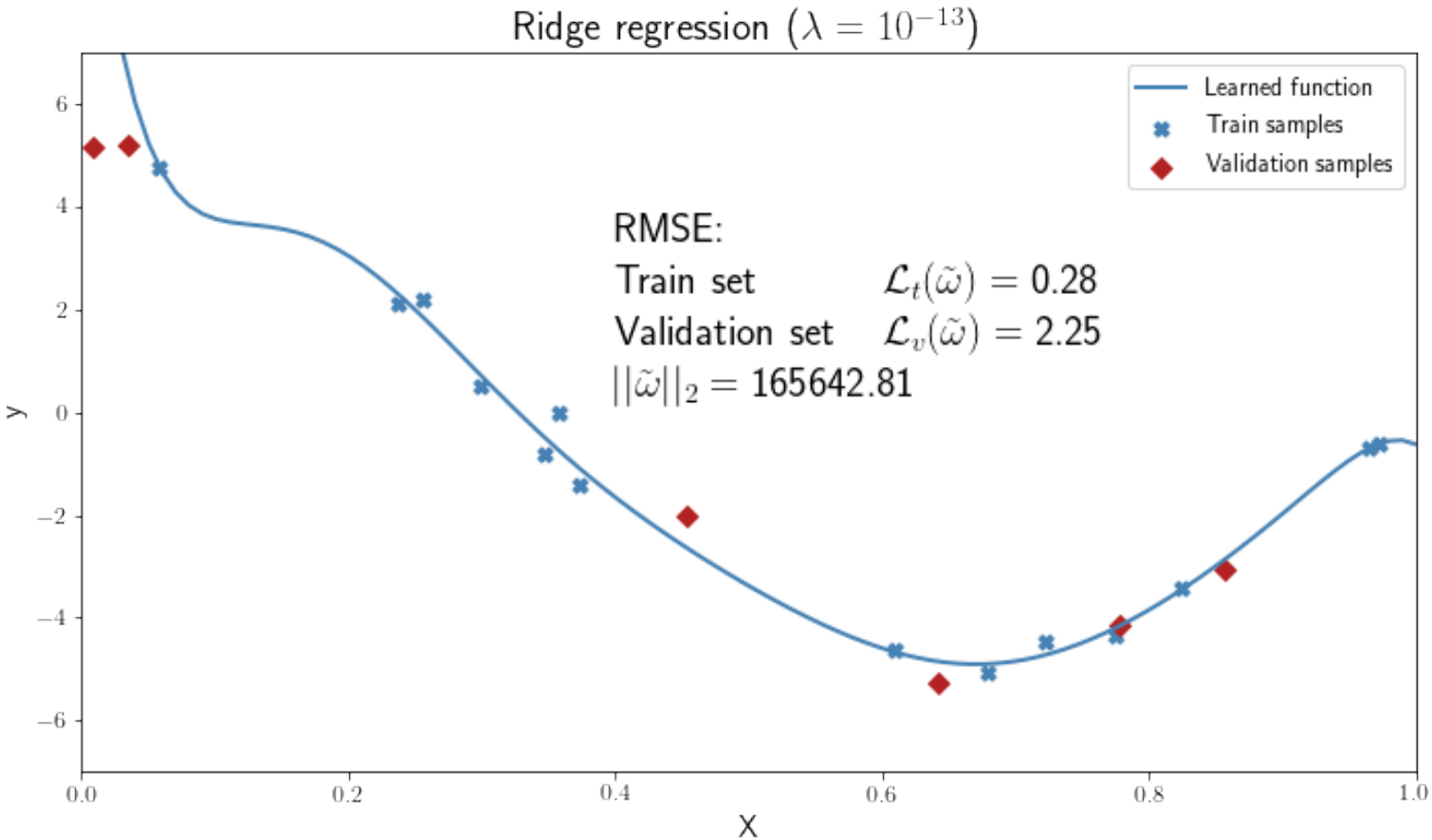
Illustration



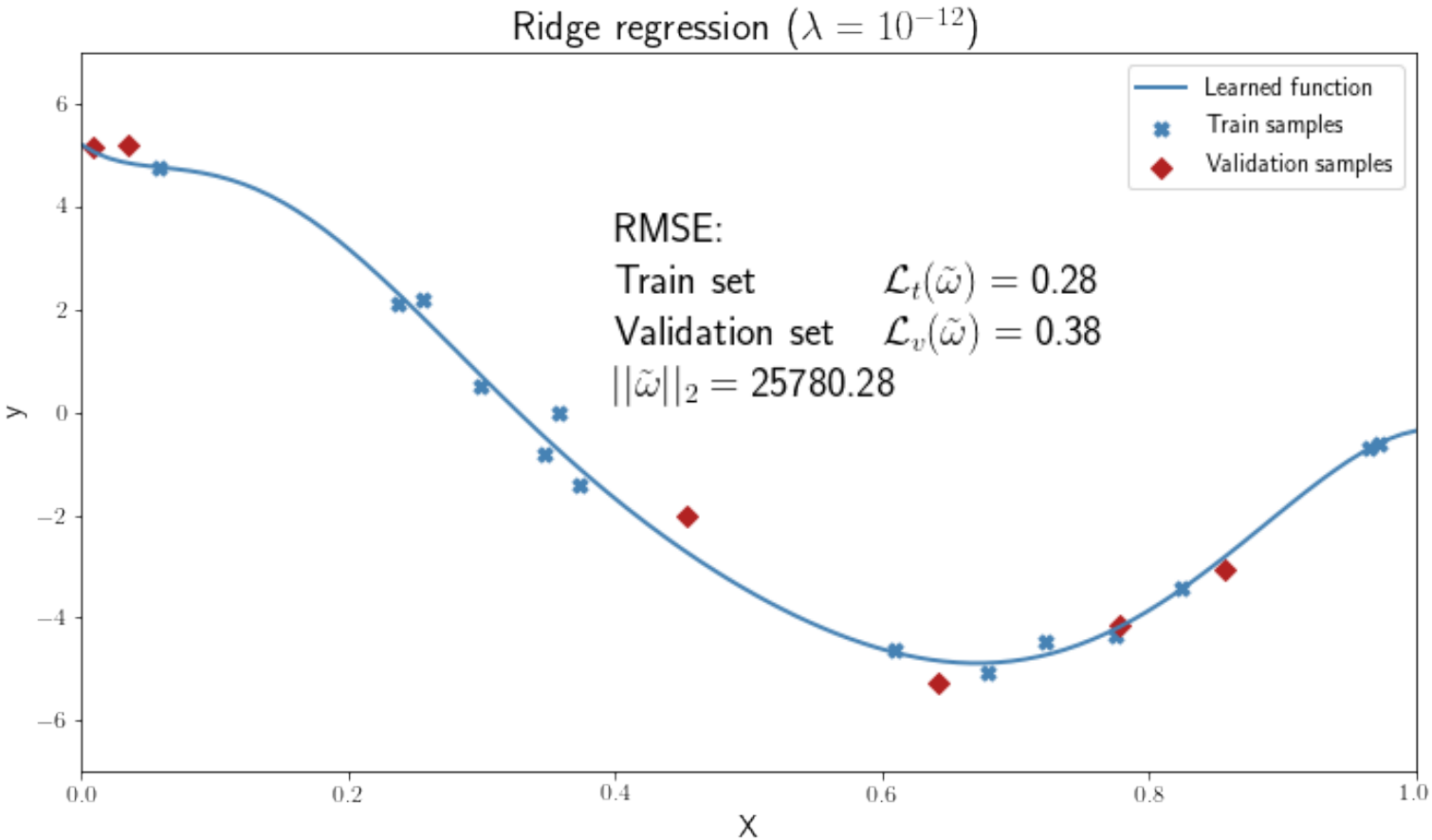
Regression


Polynomial regression (degree 10)
Learned function
Train samples
Validation samples
RMSE:
Train set $\mathcal{L}_t(\tilde{\omega}) = 0.27$
Validation set $\mathcal{L}_v(\tilde{\omega}) = 91.65$
y
X
6
4
2
0
−2
−4
−6
0.0
0.2
0.4
0.6
0.8
1.0

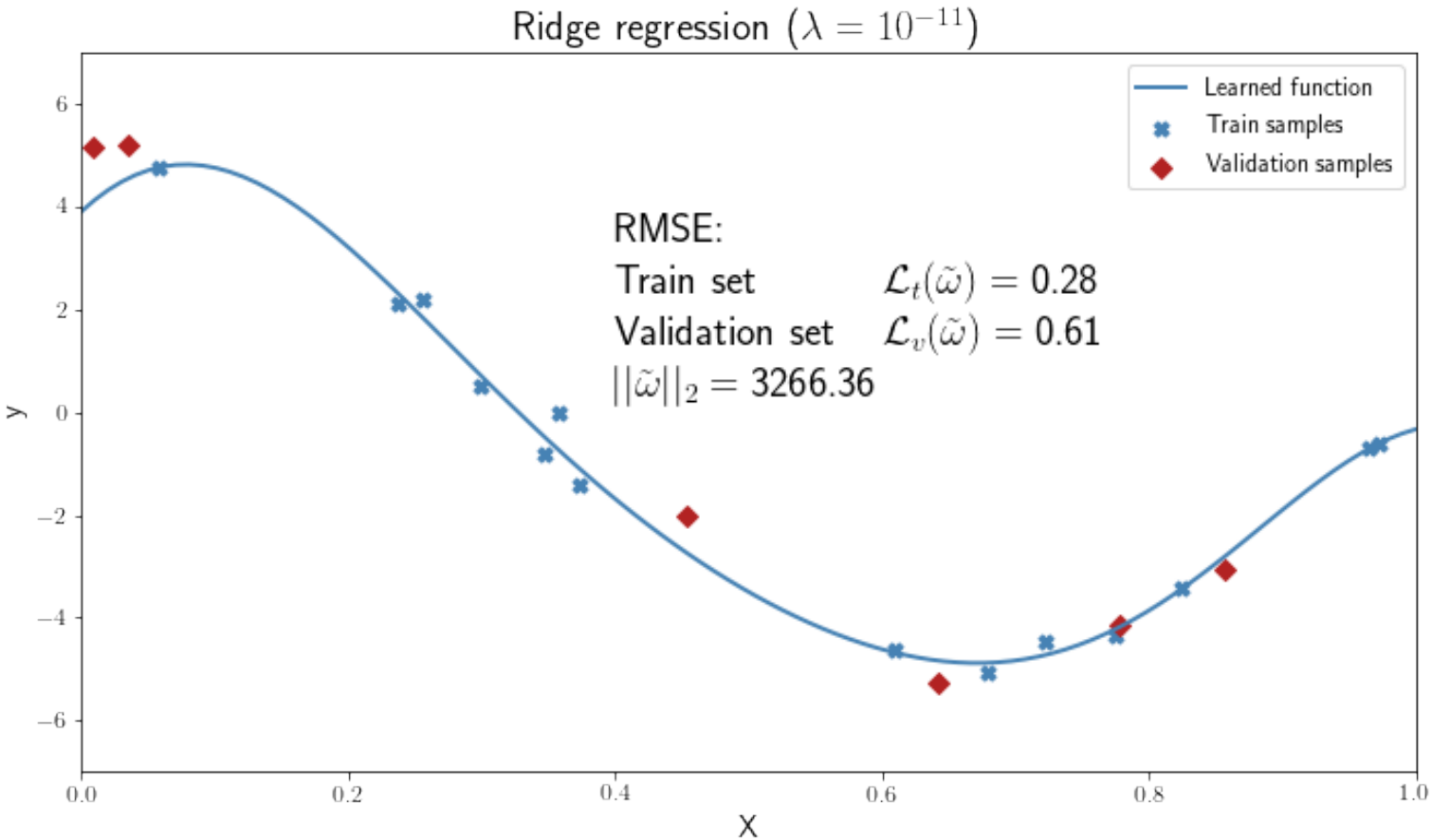
Regression


Ridge regression ($\lambda = 10^{-13}$)
Learned function
Train samples
Validation samples
RMSE:
Train set $\mathcal{L}_t(\tilde{\omega}) = 0.28$
Validation set $\mathcal{L}_v(\tilde{\omega}) = 2.25$
$||\tilde{\omega}||_2 = 165642.81$
y
X
6
4
2
0
−2
−4
−6
0.0
0.2
0.4
0.6
0.8
1.0

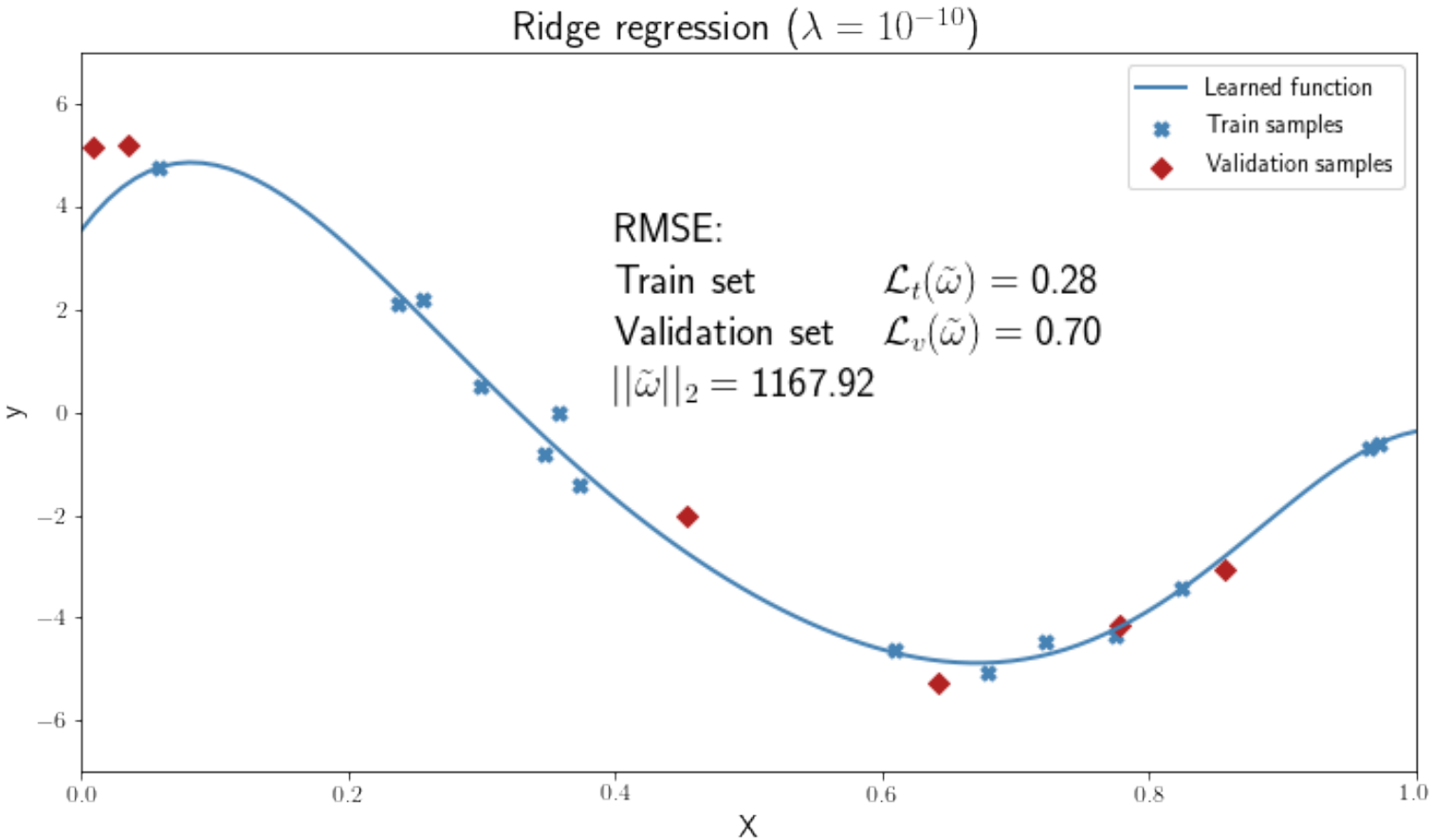
Regression


Ridge regression ($\lambda = 10^{-12}$)
Learned function
Train samples
Validation samples
RMSE:
Train set $\mathcal{L}_t(\tilde{\omega}) = 0.28$
Validation set $\mathcal{L}_v(\tilde{\omega}) = 0.38$
$||\tilde{\omega}||_2 = 25780.28$
y
X
6
4
2
0
−2
−4
−6
0.0
0.2
0.4
0.6
0.8
1.0

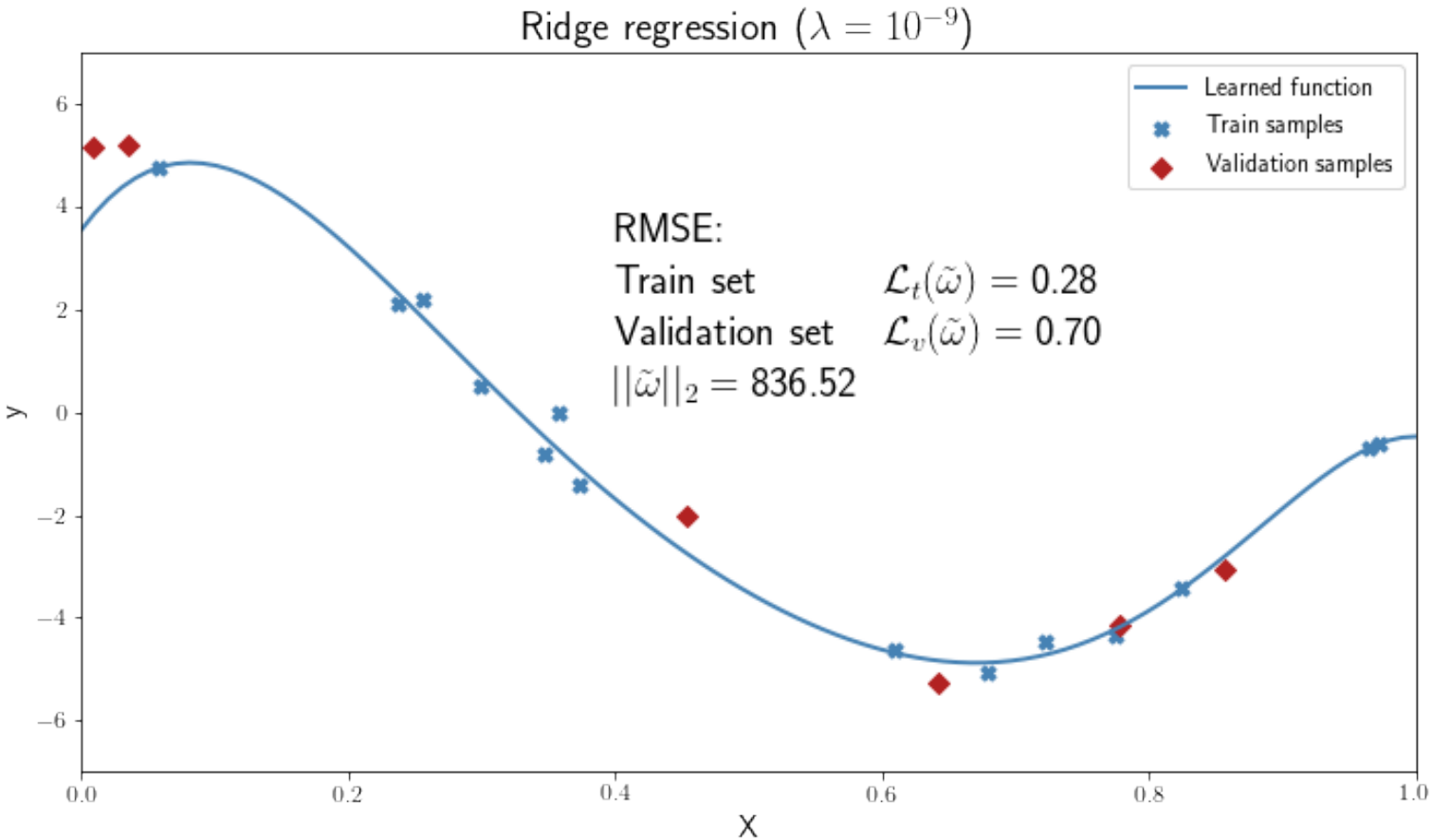
Regression


Ridge regression ($\lambda = 10^{-11}$)
Learned function
Train samples
Validation samples
RMSE:
Train set $\mathcal{L}_t(\tilde{\omega}) = 0.28$
Validation set $\mathcal{L}_v(\tilde{\omega}) = 0.61$
$||\tilde{\omega}||_2 = 3266.36$
y
X
6
4
2
0
−2
−4
−6
0.0
0.2
0.4
0.6
0.8
1.0

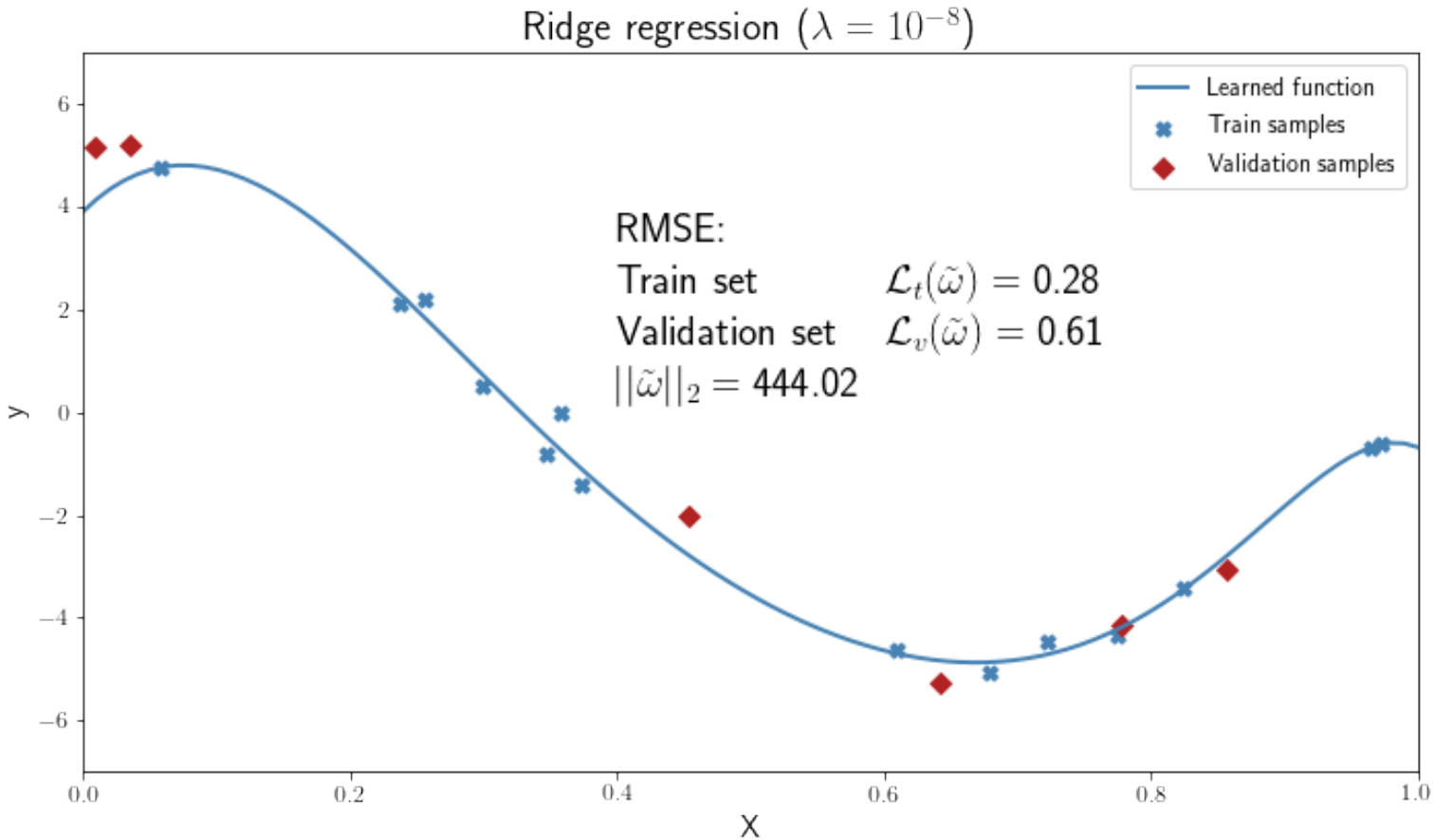
Regression


Ridge regression ($\lambda = 10^{-10}$)
Learned function
Train samples
Validation samples
RMSE:
Train set $\mathcal{L}_t(\tilde{\omega}) = 0.28$
Validation set $\mathcal{L}_v(\tilde{\omega}) = 0.70$
$||\tilde{\omega}||_2 = 1167.92$
y
X
6
4
2
0
−2
−4
−6
0.0
0.2
0.4
0.6
0.8
1.0

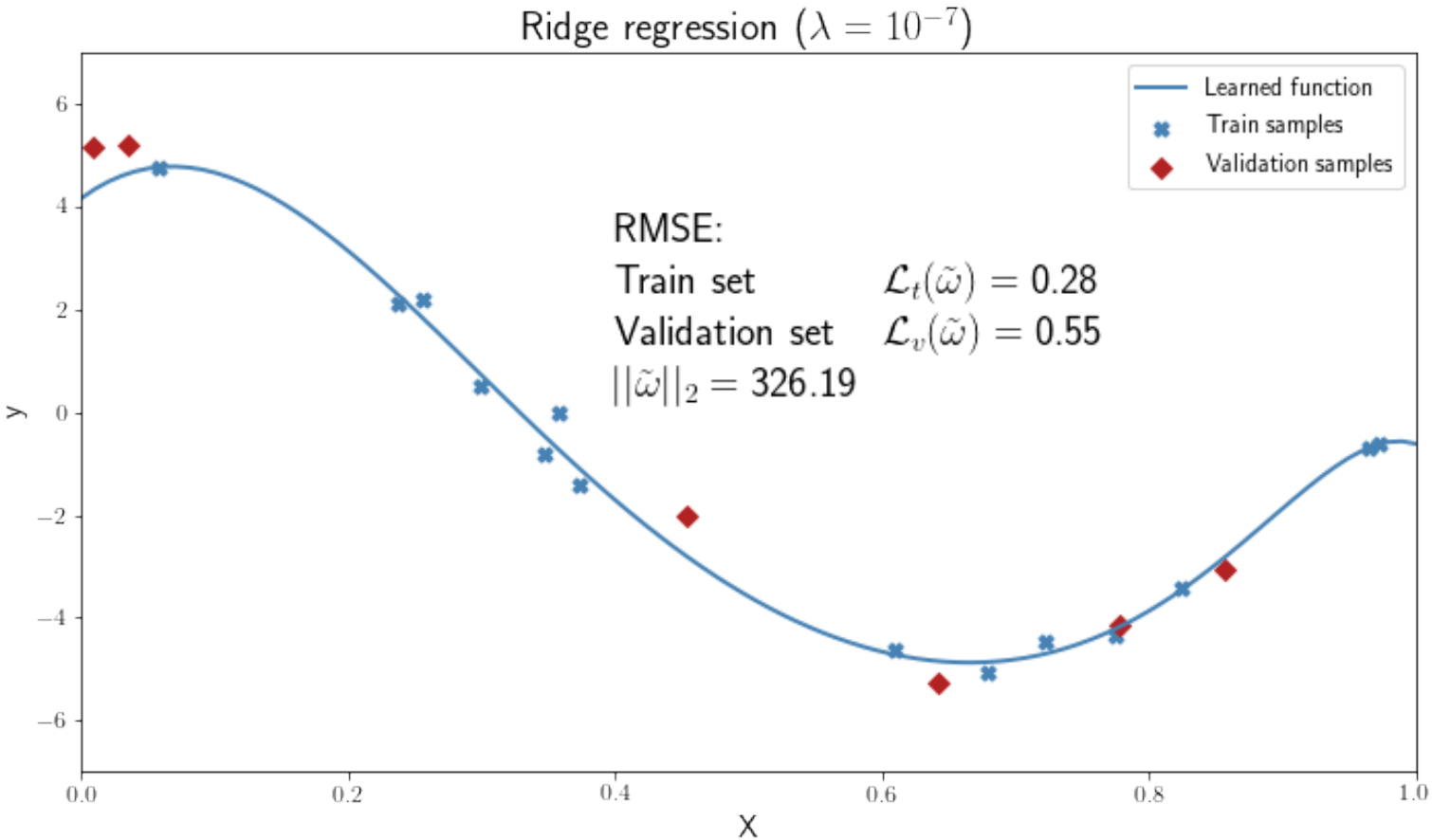
Regression


Ridge regression ($\lambda = 10^{-9}$)
Learned function
Train samples
Validation samples
RMSE:
Train set $\mathcal{L}_t(\tilde{\omega}) = 0.28$
Validation set $\mathcal{L}_v(\tilde{\omega}) = 0.70$
$||\tilde{\omega}||_2 = 836.52$
y
X
6
4
2
0
−2
−4
−6
0.0
0.2
0.4
0.6
0.8
1.0

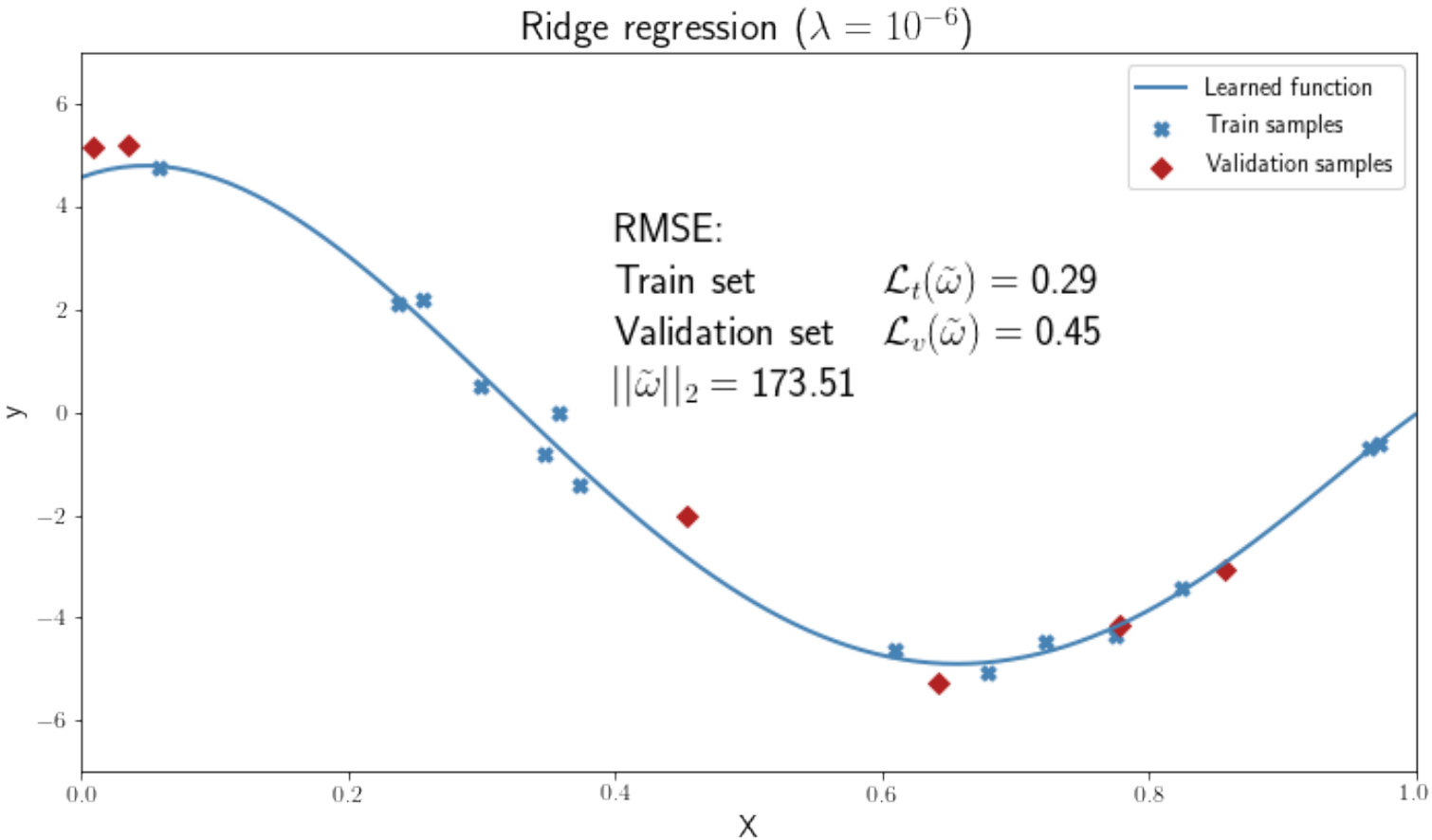
Regression


Ridge regression ($\lambda = 10^{-8}$)
Learned function
Train samples
Validation samples
RMSE:
Train set $\mathcal{L}_t(\tilde{\omega}) = 0.28$
Validation set $\mathcal{L}_v(\tilde{\omega}) = 0.61$
$||\tilde{\omega}||_2 = 444.02$
y
X
6
4
2
0
−2
−4
−6
0.0
0.2
0.4
0.6
0.8
1.0

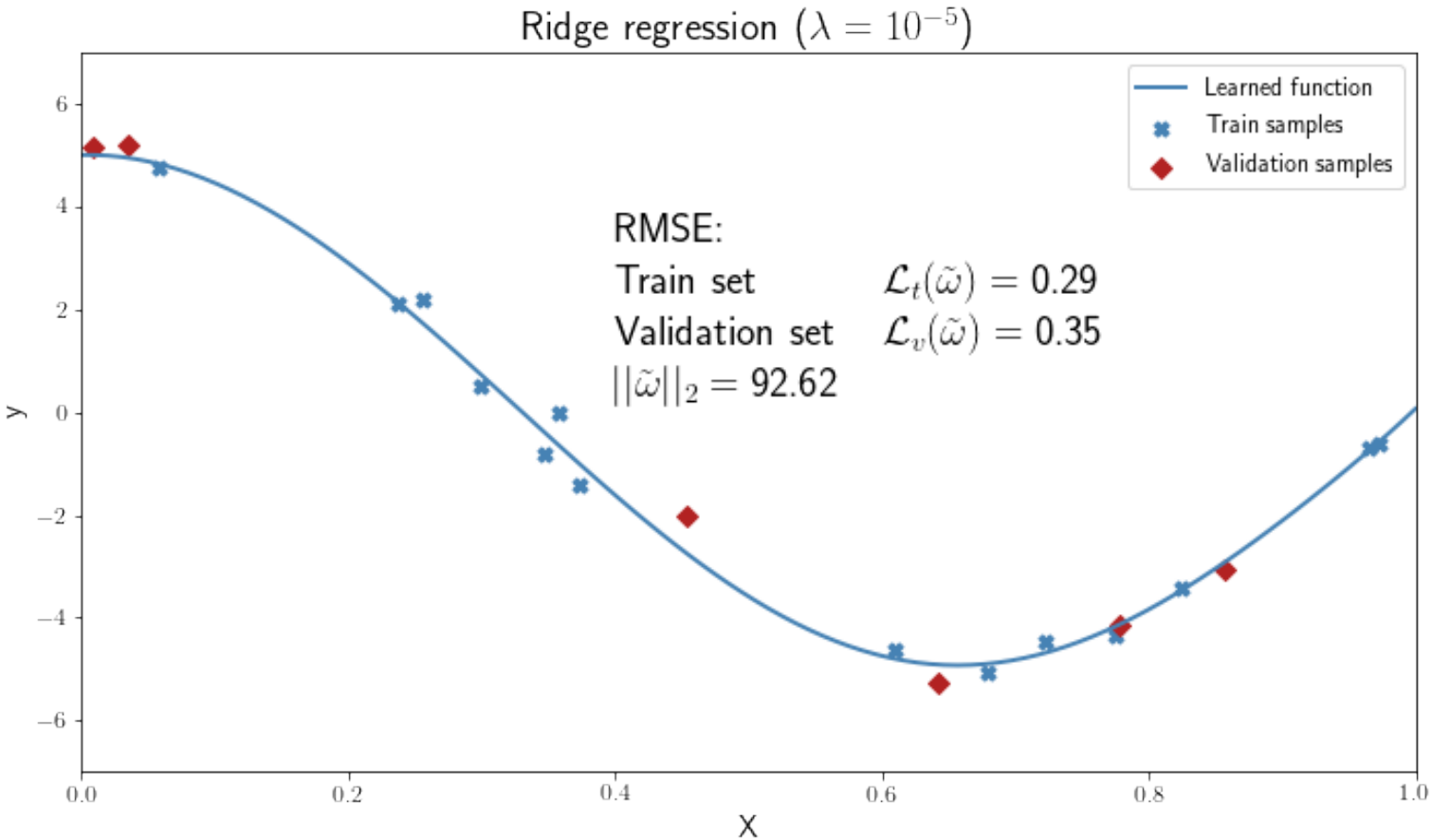
Regression


Ridge regression ($\lambda = 10^{-7}$)
Learned function
Train samples
Validation samples
RMSE:
Train set $\mathcal{L}_t(\tilde{\omega}) = 0.28$
Validation set $\mathcal{L}_v(\tilde{\omega}) = 0.55$
$||\tilde{\omega}||_2 = 326.19$
y
X
6
4
2
0
−2
−4
−6
0.0
0.2
0.4
0.6
0.8
1.0

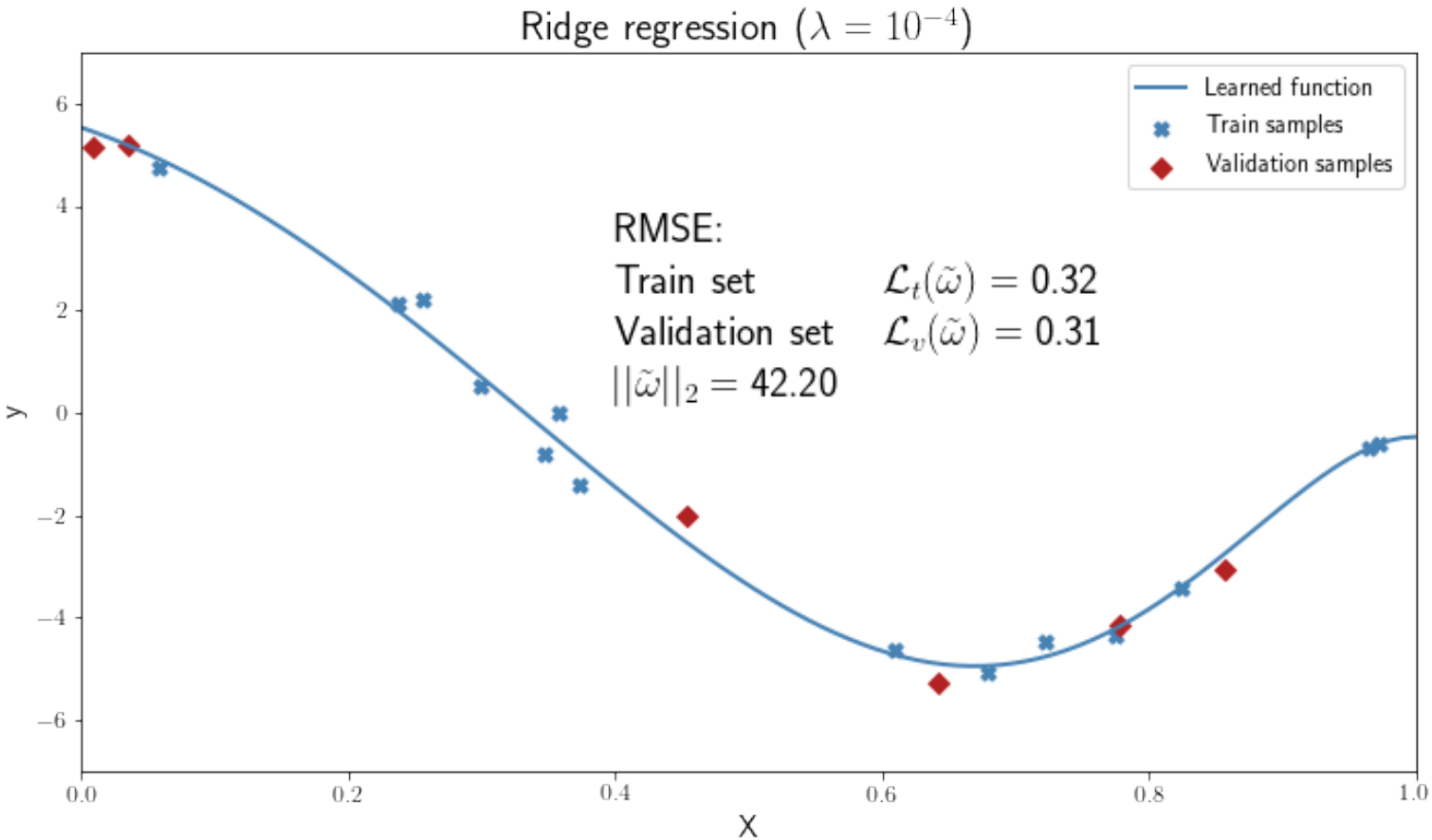
Regression


Ridge regression ($\lambda = 10^{-6}$)
Learned function
Train samples
Validation samples
RMSE:
Train set $\mathcal{L}_t(\tilde{\omega}) = 0.29$
Validation set $\mathcal{L}_v(\tilde{\omega}) = 0.45$
$||\tilde{\omega}||_2 = 173.51$
y
X
6
4
2
0
−2
−4
−6
0.0
0.2
0.4
0.6
0.8
1.0

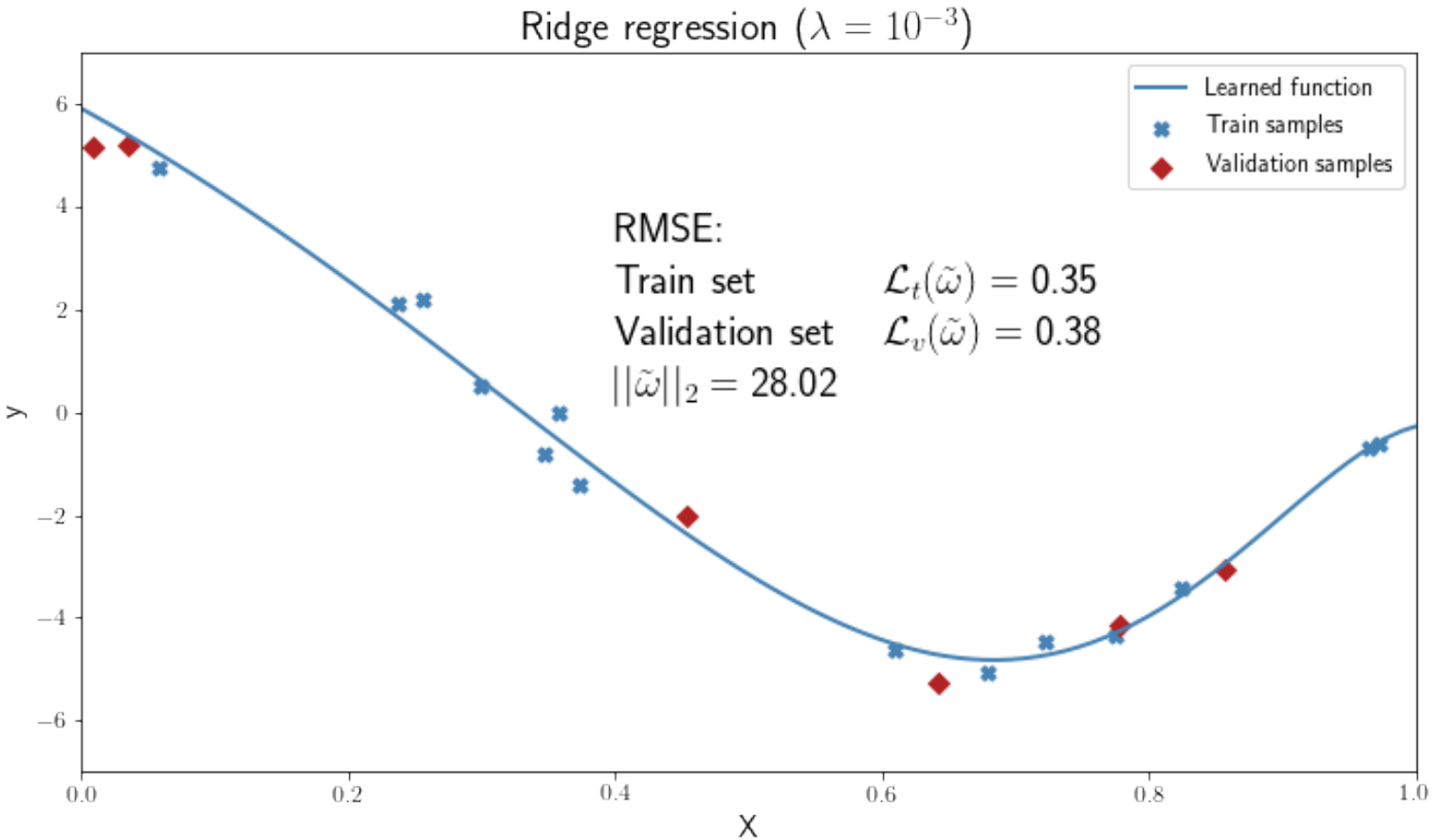
Regression


Ridge regression ($\lambda = 10^{-5}$)
Learned function
Train samples
Validation samples
RMSE:
Train set $\mathcal{L}_t(\tilde{\omega}) = 0.29$
Validation set $\mathcal{L}_v(\tilde{\omega}) = 0.35$
$||\tilde{\omega}||_2 = 92.62$
y
X
6
4
2
0
−2
−4
−6
0.0
0.2
0.4
0.6
0.8
1.0

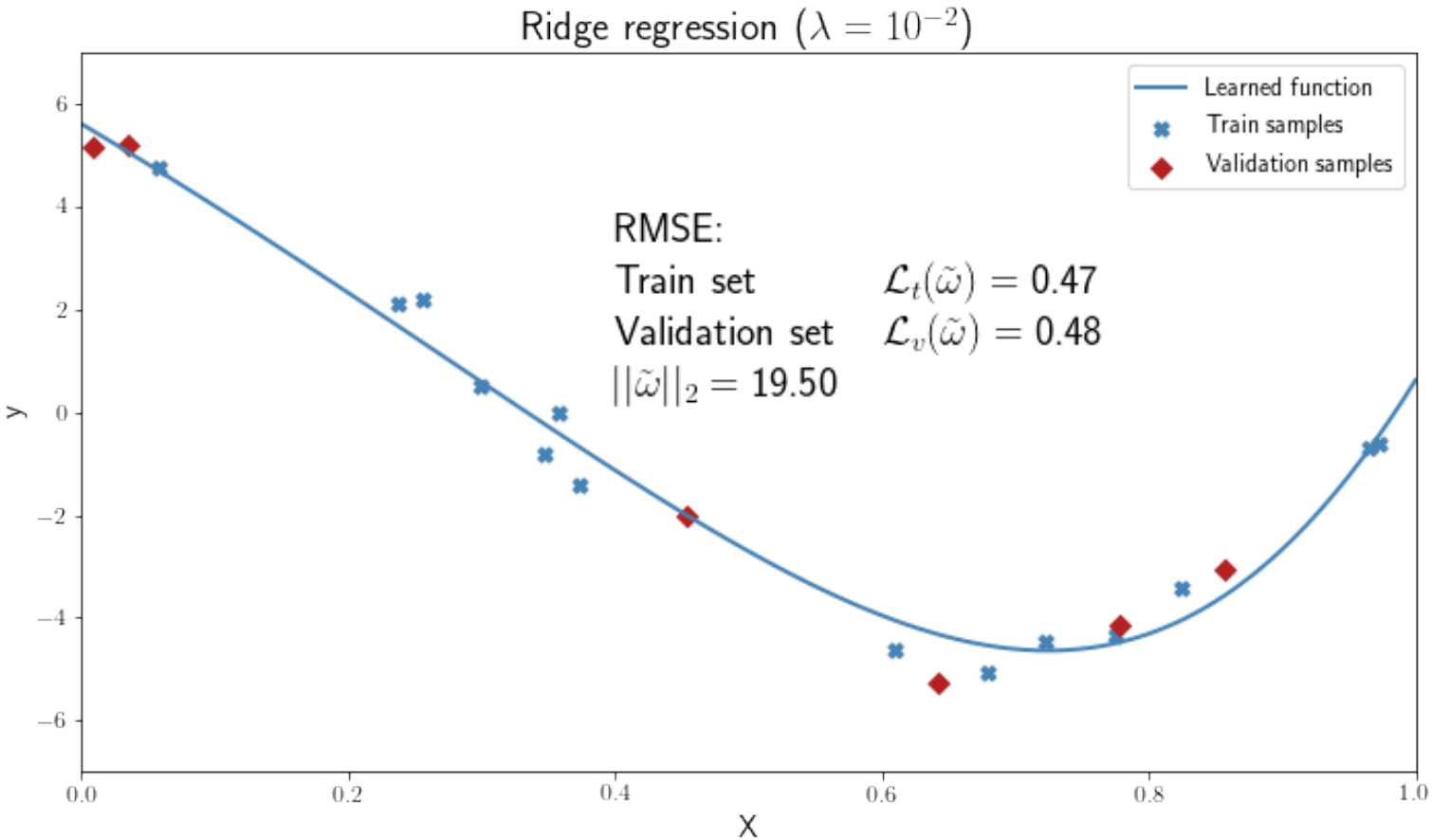
Regression


Ridge regression ($\lambda = 10^{-4}$)
Learned function
Train samples
Validation samples
RMSE:
Train set $\mathcal{L}_t(\tilde{\omega}) = 0.32$
Validation set $\mathcal{L}_v(\tilde{\omega}) = 0.31$
$||\tilde{\omega}||_2 = 42.20$
y
X
6
4
2
0
−2
−4
−6
0.0
0.2
0.4
0.6
0.8
1.0

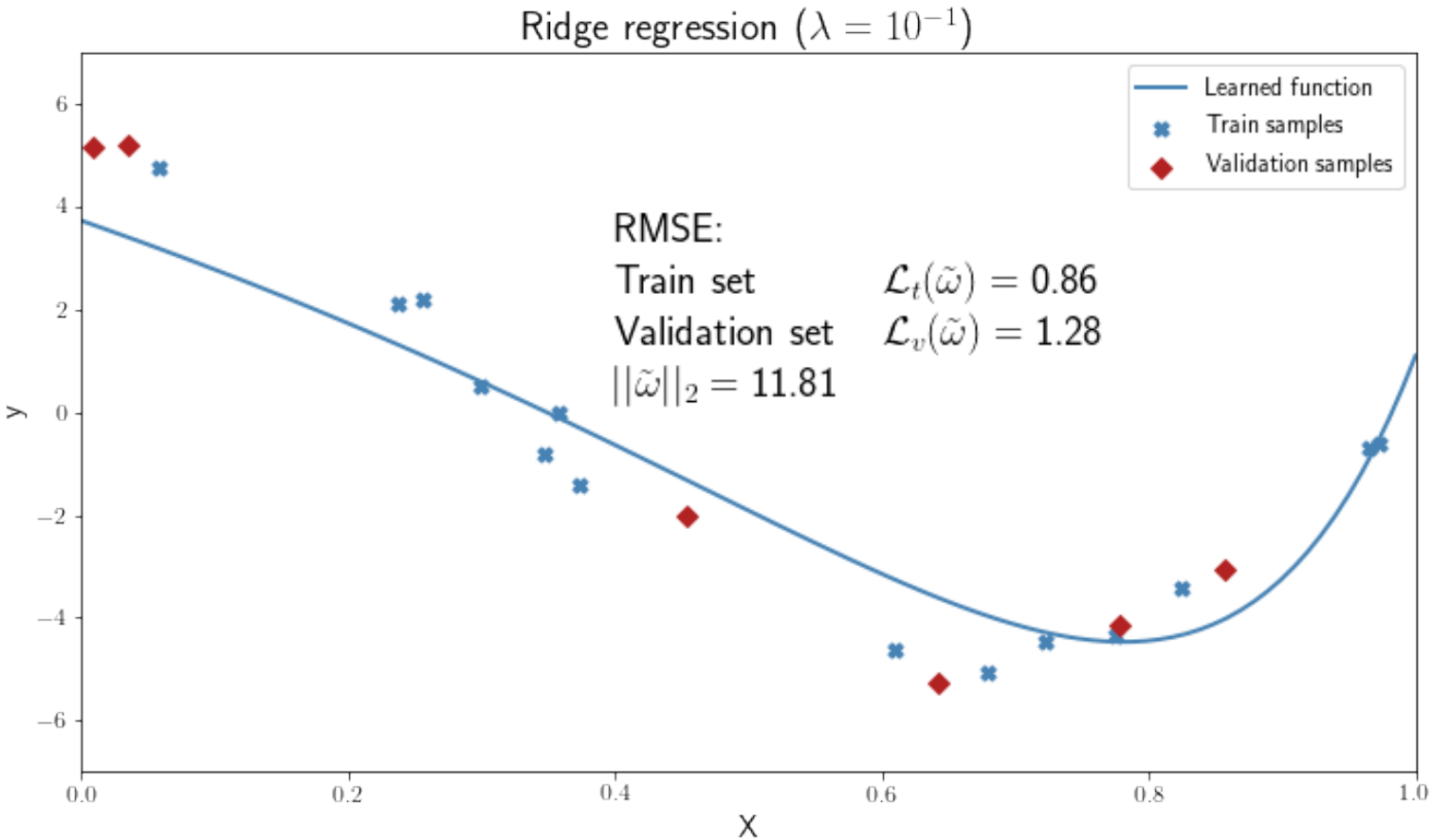
Regression


Ridge regression ($\lambda = 10^{-3}$)
Learned function
Train samples
Validation samples
RMSE:
Train set $\mathcal{L}_t(\tilde{\omega}) = 0.35$
Validation set $\mathcal{L}_v(\tilde{\omega}) = 0.38$
$||\tilde{\omega}||_2 = 28.02$
y
X
6
4
2
0
−2
−4
−6
0.0
0.2
0.4
0.6
0.8
1.0

Regression


Ridge regression ($\lambda = 10^{-2}$)
Learned function
Train samples
Validation samples
RMSE:
Train set $\mathcal{L}_t(\tilde{\omega}) = 0.47$
Validation set $\mathcal{L}_v(\tilde{\omega}) = 0.48$
$||\tilde{\omega}||_2 = 19.50$
y
X
6
4
2
0
−2
−4
−6
0.0
0.2
0.4
0.6
0.8
1.0

Regression


Ridge regression ($\lambda = 10^{-1}$)
Learned function
Train samples
Validation samples
RMSE:
Train set $\mathcal{L}_t(\tilde{\omega}) = 0.86$
Validation set $\mathcal{L}_v(\tilde{\omega}) = 1.28$
$||\tilde{\omega}||_2 = 11.81$
y
X
6
4
2
0
−2
−4
−6
0.0
0.2
0.4
0.6
0.8
1.0

L_1 penalty

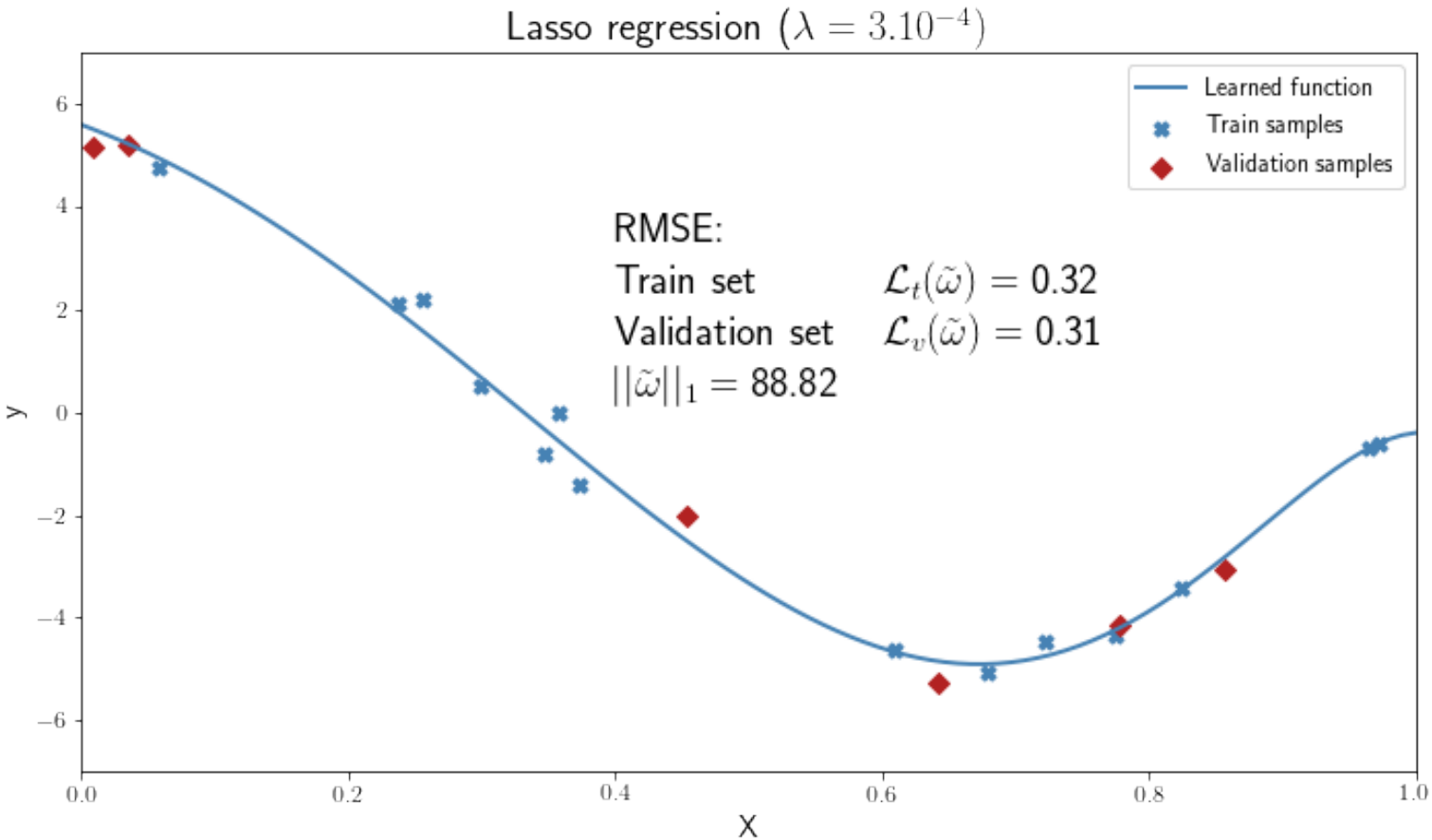
$$\mathcal{R}(\mathbf{W}) = \sum_{l=1}^{L} \sum_{i=1}^{n_l} \sum_{j=0}^{n_{l-1}} |\omega_{ij}^{(l)}|$$

- Example: Lasso regression

$$\tilde{\omega}^\star = \arg\min_{\tilde{\omega}} ||\mathbf{y} - \tilde{\omega}^t \tilde{X}||_2^2 + \lambda ||\tilde{\omega}||_1$$

- No explicit solution (iterative algorithm)
- Bayesian interpretation: $\tilde{\omega}^\star$ is the maximum a posteriori (MAP) when the residual is Gaussian and the weights have a Laplace prior (with variance controlled by λ)

Lasso Regression



	Regr
ω_0	310.4
ω_1	$-1.1e04$
ω_2	$1.4e05$
ω_3	$-9.2e05$
ω_4	$3.7e06$
ω_5	$-9.5e06$
ω_6	$1.6e07$
ω_7	$-1.8e07$
ω_8	$1.2e07$
ω_9	$-4.9e06$
ω_{10}	$8.5e05$
$\|\|\tilde{\omega}\|\|_1$	$6.6e07$
$\|\|\tilde{\omega}\|\|_2$	$2.9e07$
$\mathcal{L}_t(\tilde{\omega})$	0.27
$\mathcal{L}_v(\tilde{\omega})$	91.7

	Regr	$L_2(10^{-4})$
ω_0	310.4	5.5
ω_1	$-1.1e04$	-8.9
ω_2	$1.4e05$	-29.4
ω_3	$-9.2e05$	9.0
ω_4	$3.7e06$	20.5
ω_5	$-9.5e06$	14.3
ω_6	$1.6e07$	4.8
ω_7	$-1.8e07$	-2.1
ω_8	$1.2e07$	-5.2
ω_9	$-4.9e06$	-5.4
ω_{10}	$8.5e05$	-3.7
$\|\|\tilde{\omega}\|\|_1$	$6.6e07$	158.8
$\|\|\tilde{\omega}\|\|_2$	$2.9e07$	42.2
$\mathcal{L}_t(\tilde{\omega})$	0.27	0.32
$\mathcal{L}_v(\tilde{\omega})$	91.7	0.31

	Regr	$L_2(10^{-4})$	$L_1(3.10^{-4})$
ω_0	310.4	5.5	5.6
ω_1	$-1.1e04$	-8.9	-9.7
ω_2	$1.4e05$	-29.4	-25.8
ω_3	$-9.2e05$	9.0	-25.8
ω_4	$3.7e06$	20.5	$\times$
ω_5	$-9.5e06$	14.3	38.6
ω_6	$1.6e07$	4.8	$\times$
ω_7	$-1.8e07$	-2.1	$\times$
ω_8	$1.2e07$	-5.2	$\times$
ω_9	$-4.9e06$	-5.4	$\times$
ω_{10}	$8.5e05$	-3.7	-9.0
$\|\|\tilde{\omega}\|\|_1$	$6.6e07$	158.8	88.82
$\|\|\tilde{\omega}\|\|_2$	$2.9e07$	42.2	51.45
$\mathcal{L}_t(\tilde{\omega})$	0.27	0.32	0.32
$\mathcal{L}_v(\tilde{\omega})$	91.7	0.31	0.31

Why does L_1 regularization lead to sparse models?

$$\mathcal{C}(\mathbf{W}) = \mathcal{L}(\mathbf{W}) + \lambda \mathcal{R}(\mathbf{W})$$

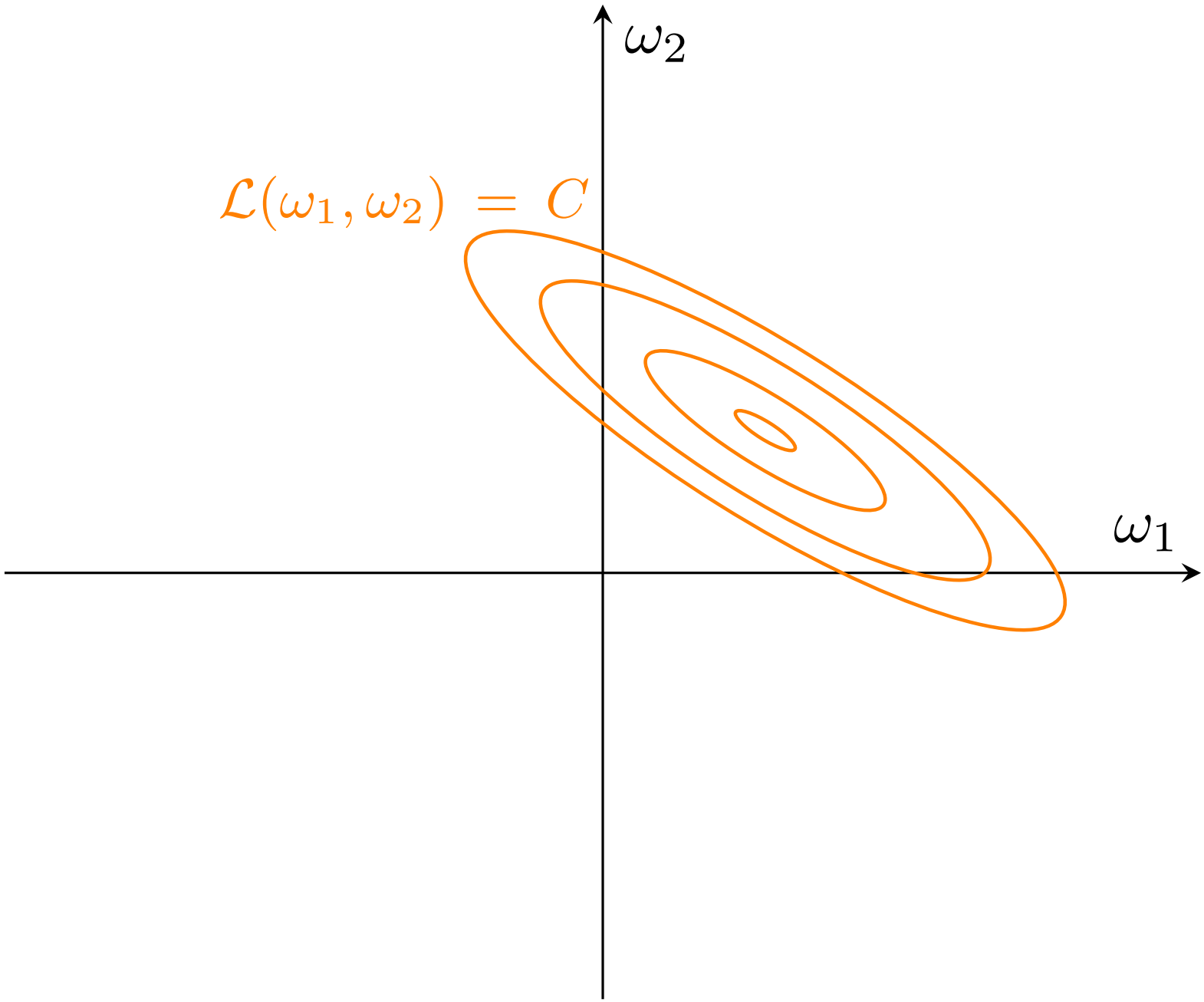
Penalty or constraint? (Bach et al., 2012)

When $\mathcal{L}$ and $\mathcal{R}$ are convex, finding $\mathbf{W}$ which minimizes $\mathcal{C}(\mathbf{W})$ is equivalent to find $\mathbf{W}$ which minimizes $\mathcal{L}(\mathbf{W})$ under the constraint than

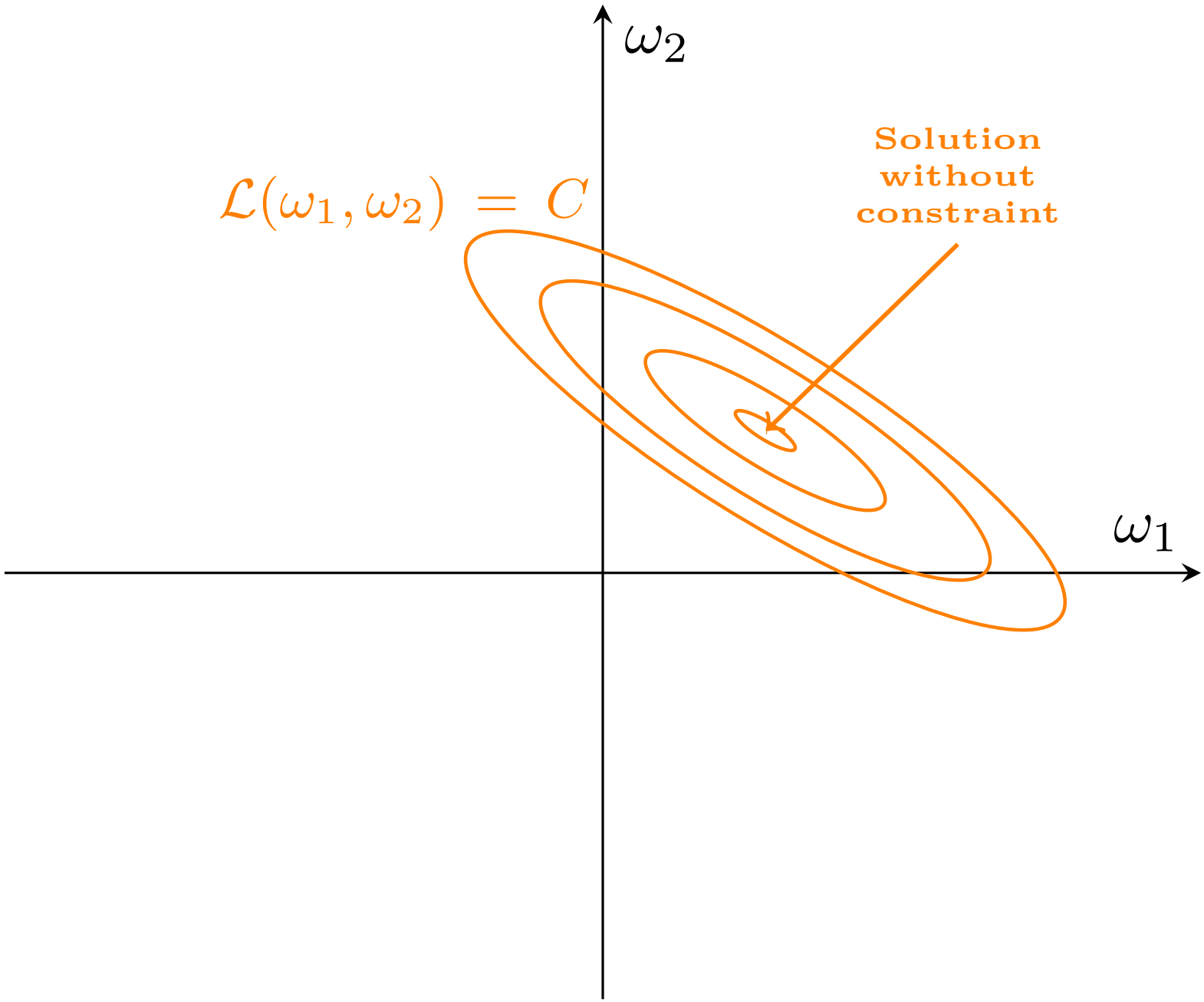
$$\mathcal{R}(\mathbf{W}) \leq t$$

for a given $t > 0$ which depends on λ

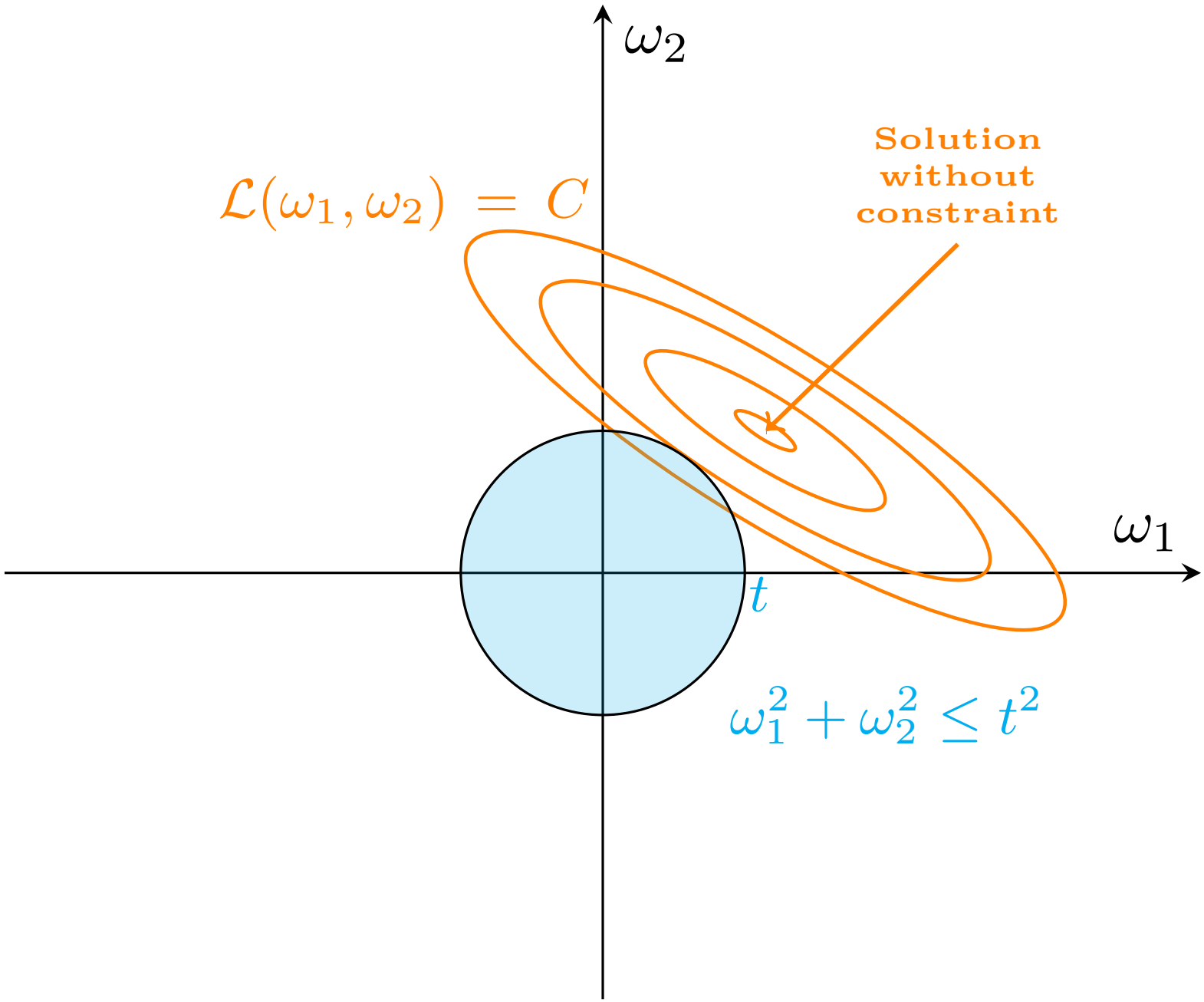
Geometrical interpretation



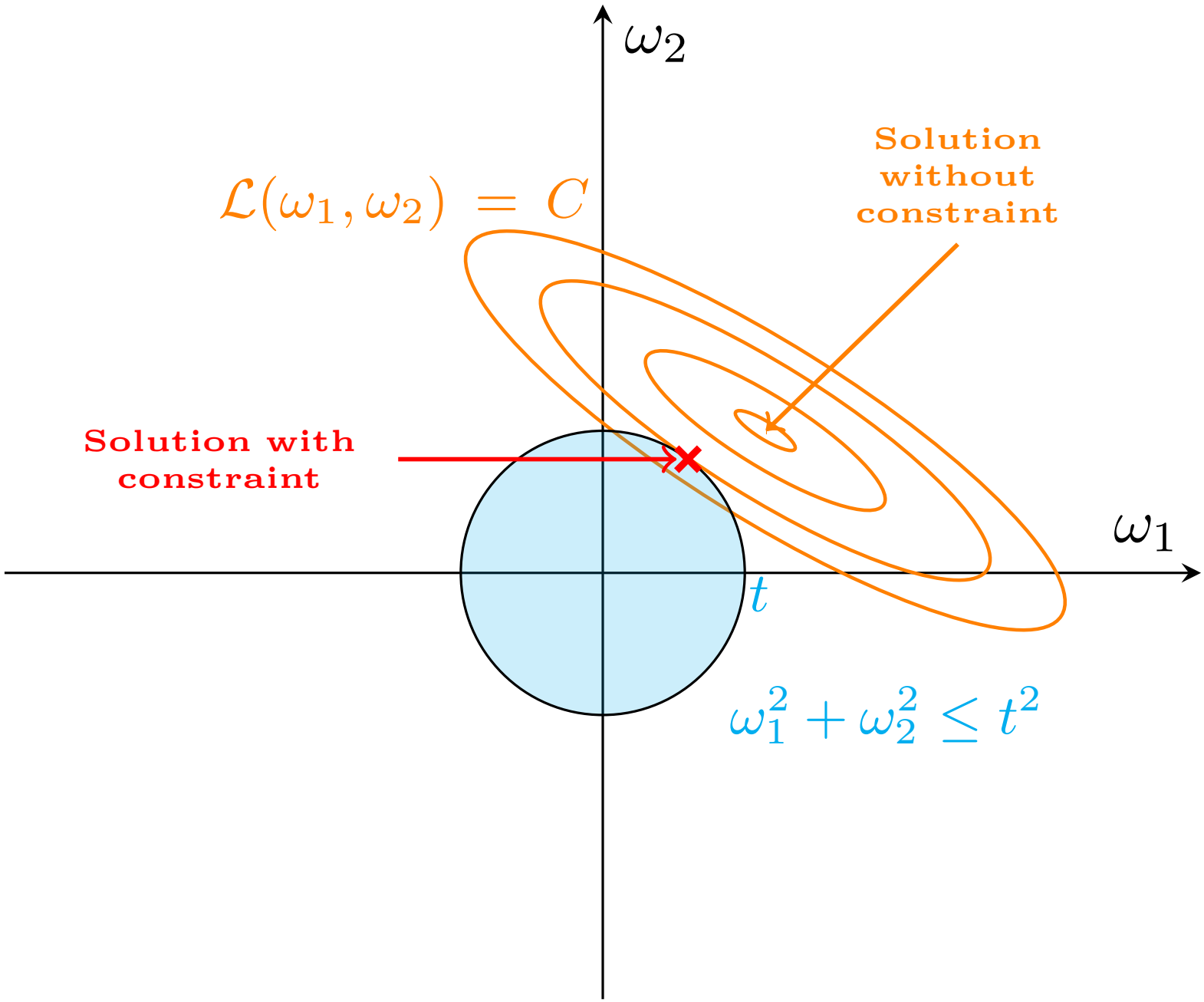
Geometrical interpretation



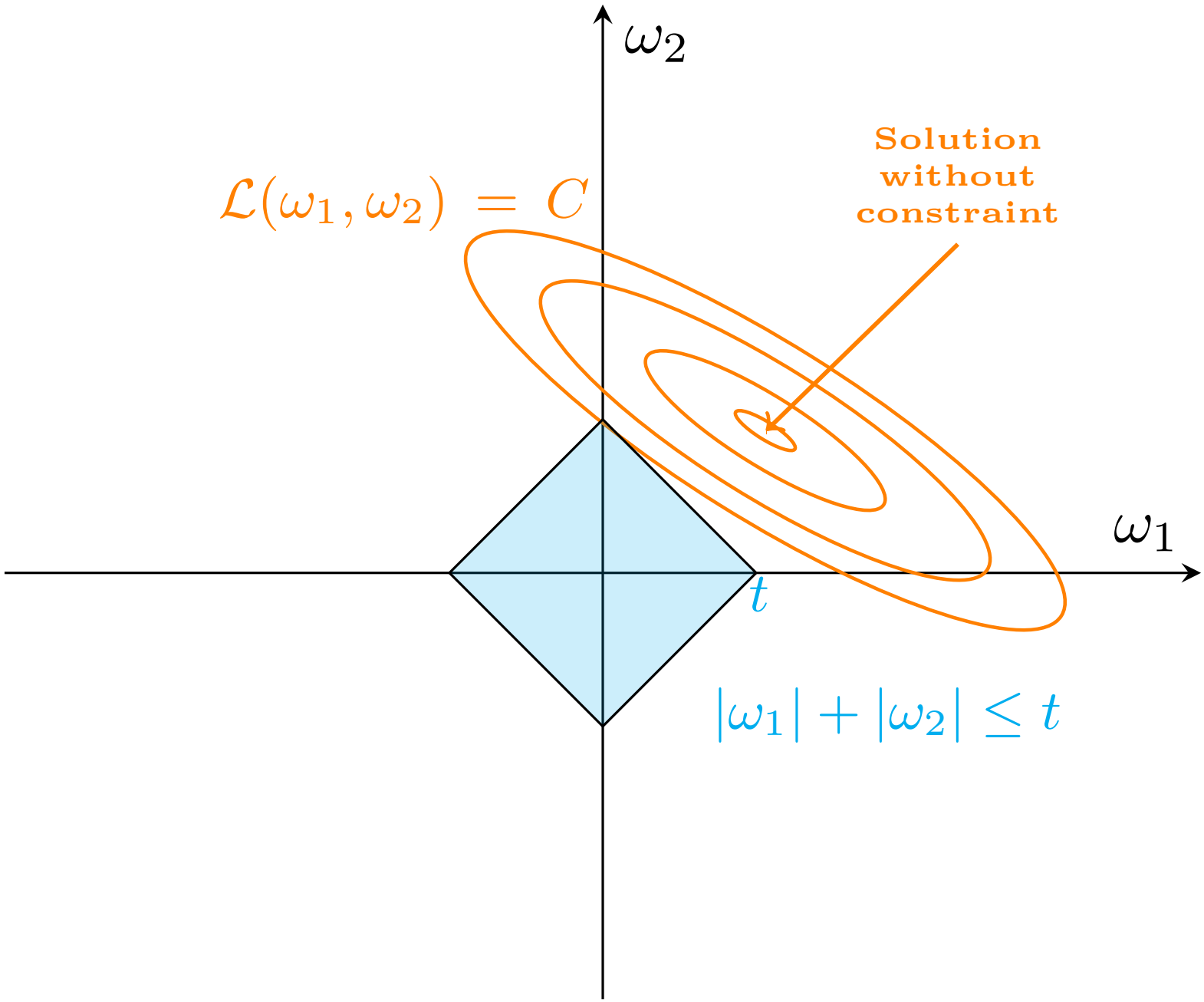
Geometrical interpretation



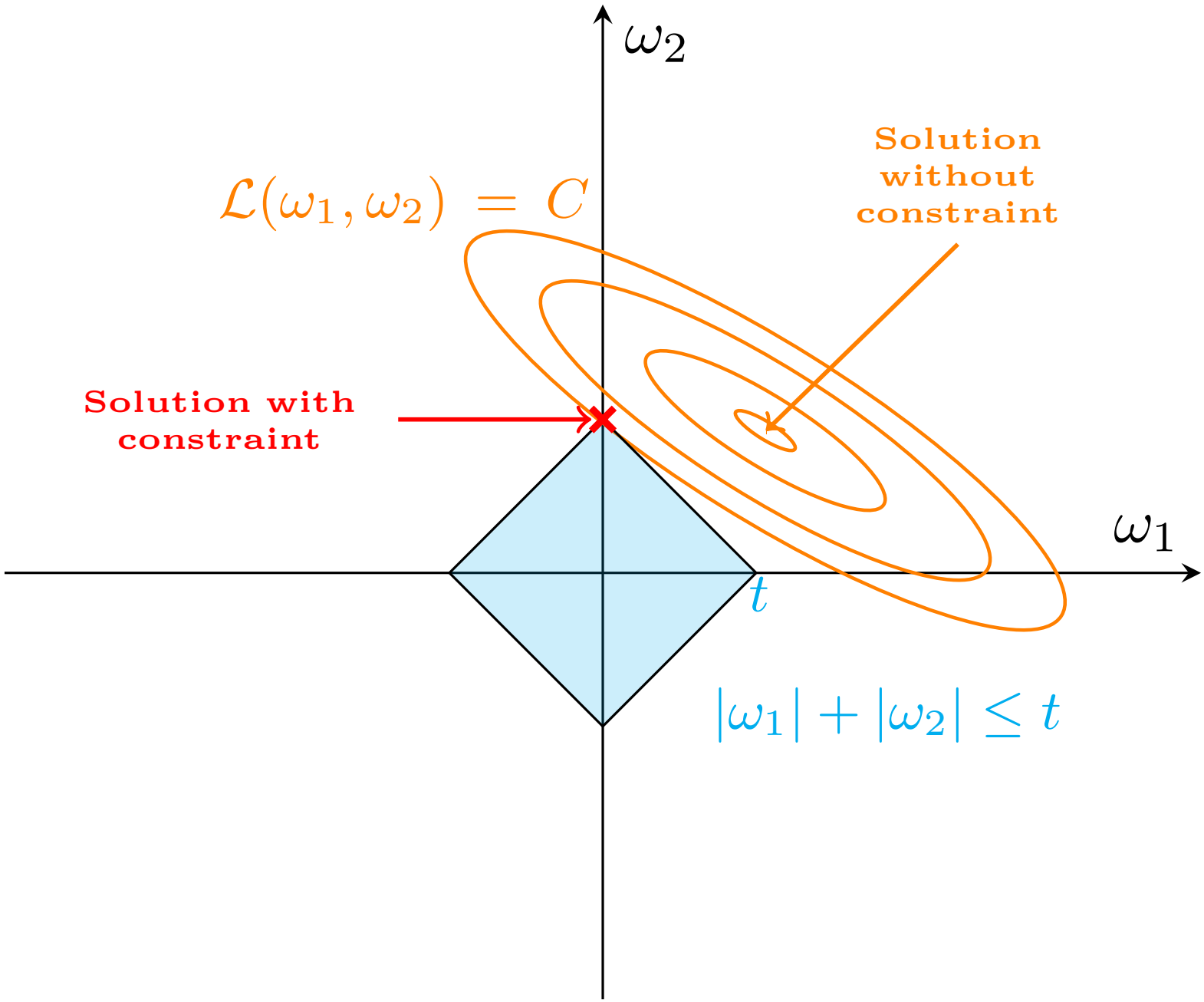
Geometrical interpretation



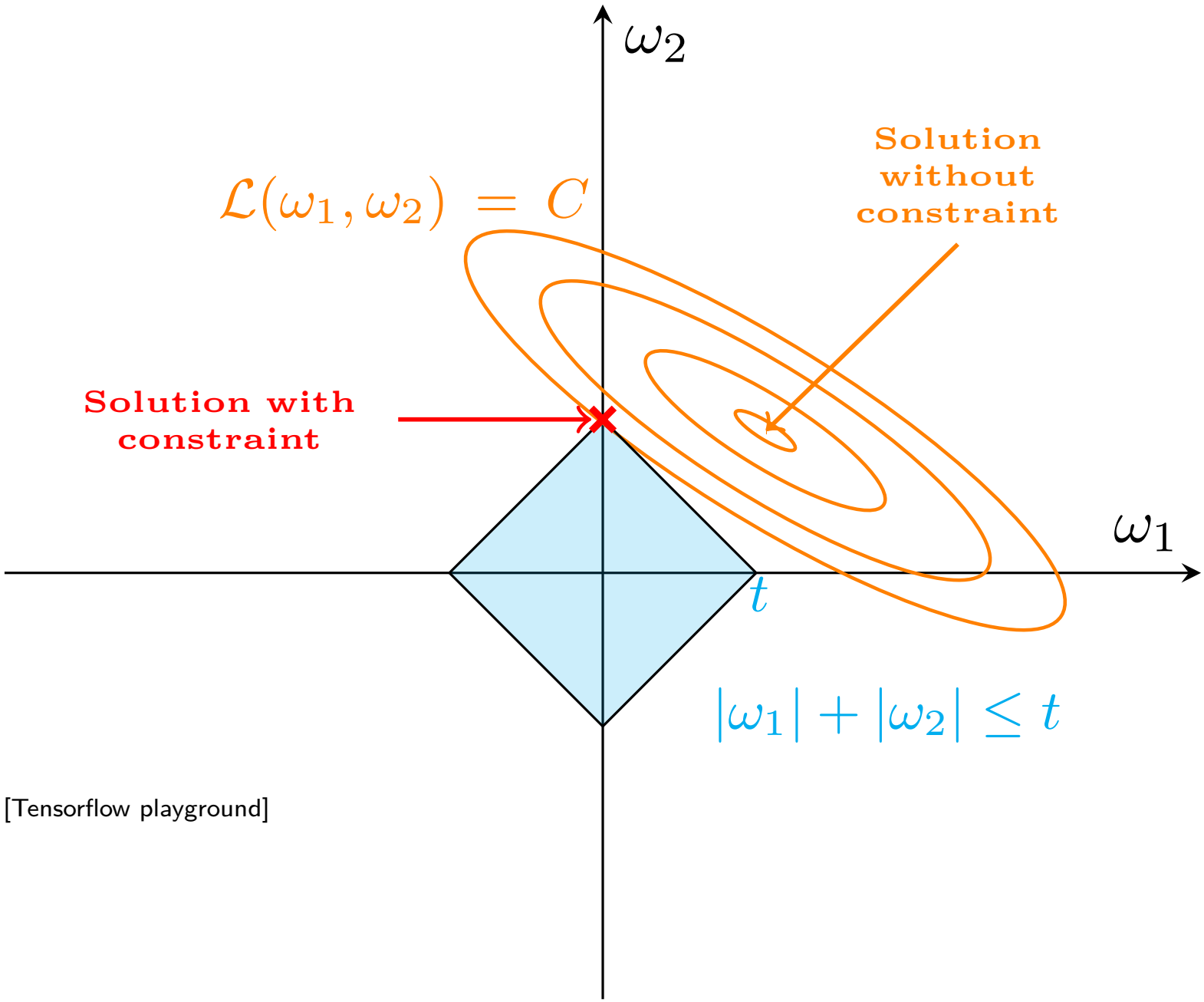
Geometrical interpretation



Geometrical interpretation



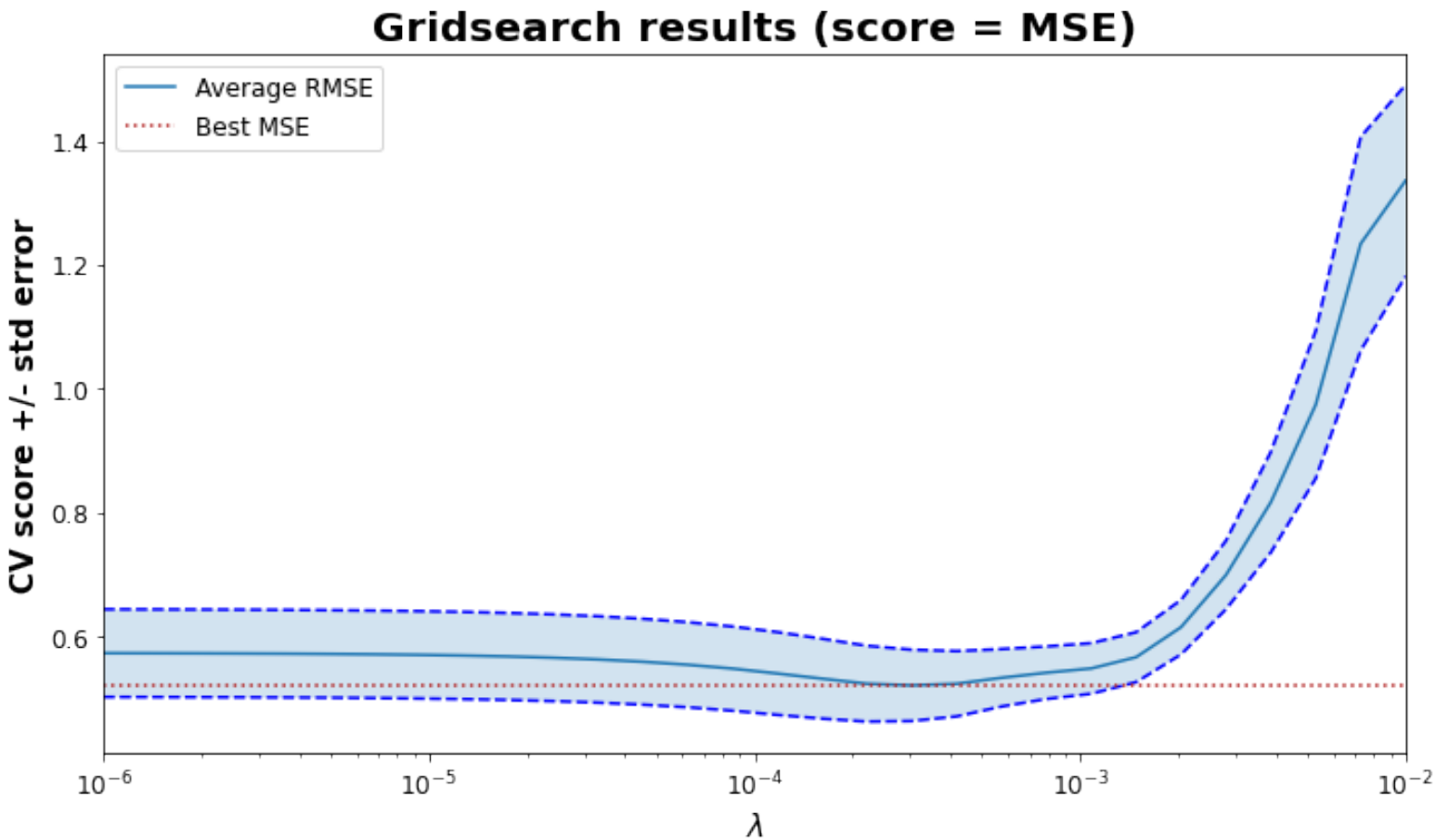
Geometrical interpretation



[Tensorflow playground]

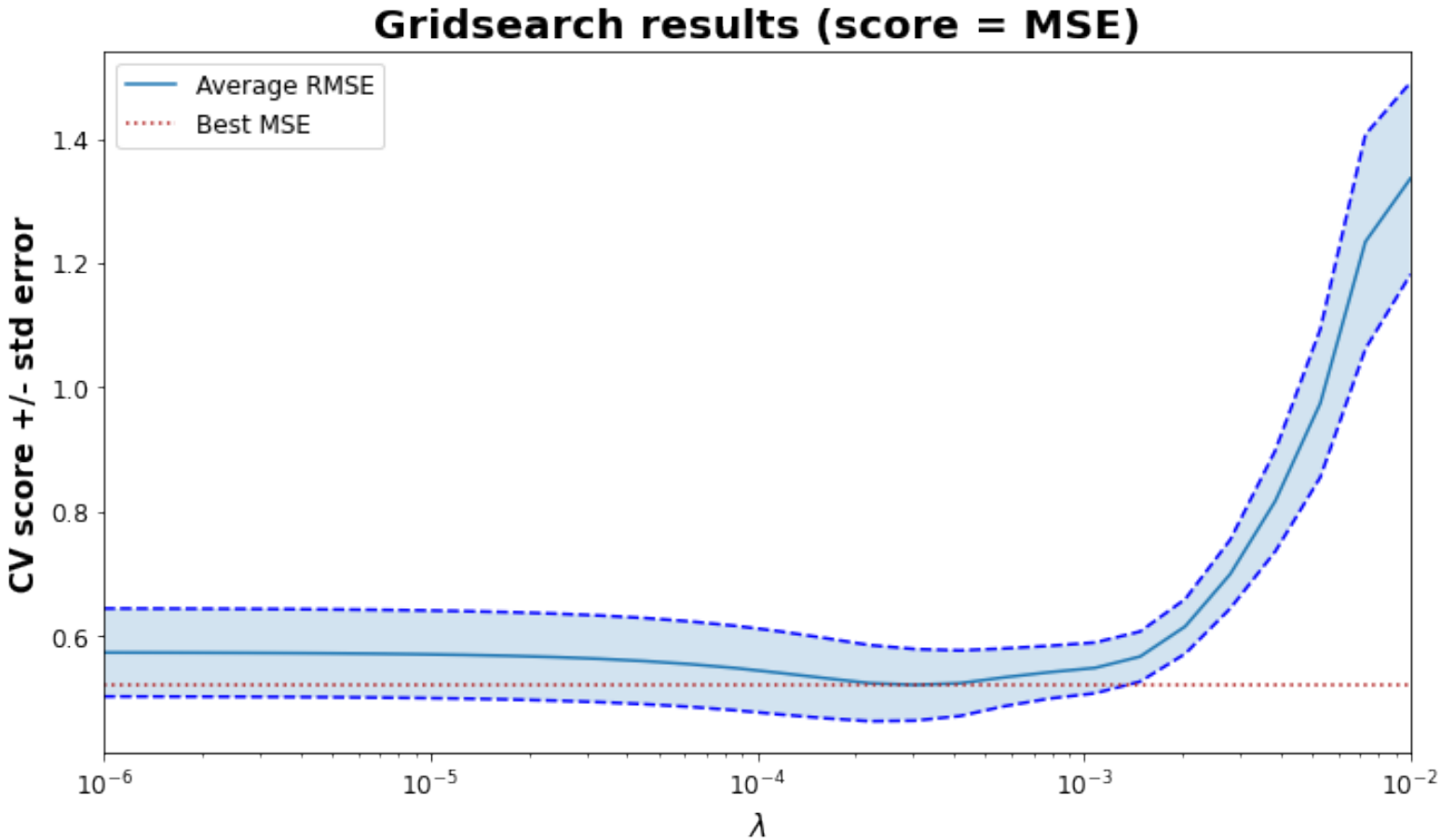
How to choose the regularization parameters?

Cross-validation on a validation set (with grid search)



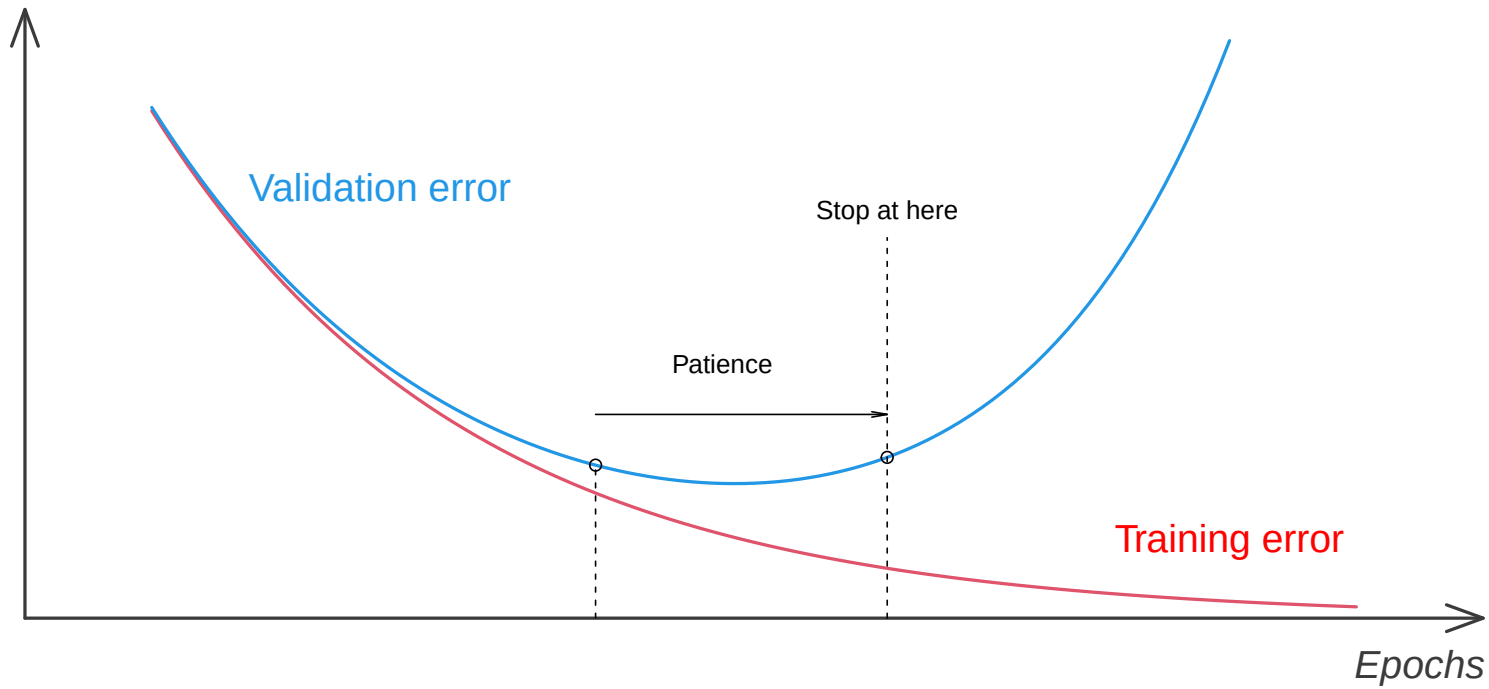
How to choose the regularization parameters?

Cross-validation on a validation set (with grid search)



If the validation set is too much used for tuning hyper-parameters, it is recommended test the final model on a third data set (test set)

Early-stop


Validation error
Stop at here
Patience
Training error
Epochs

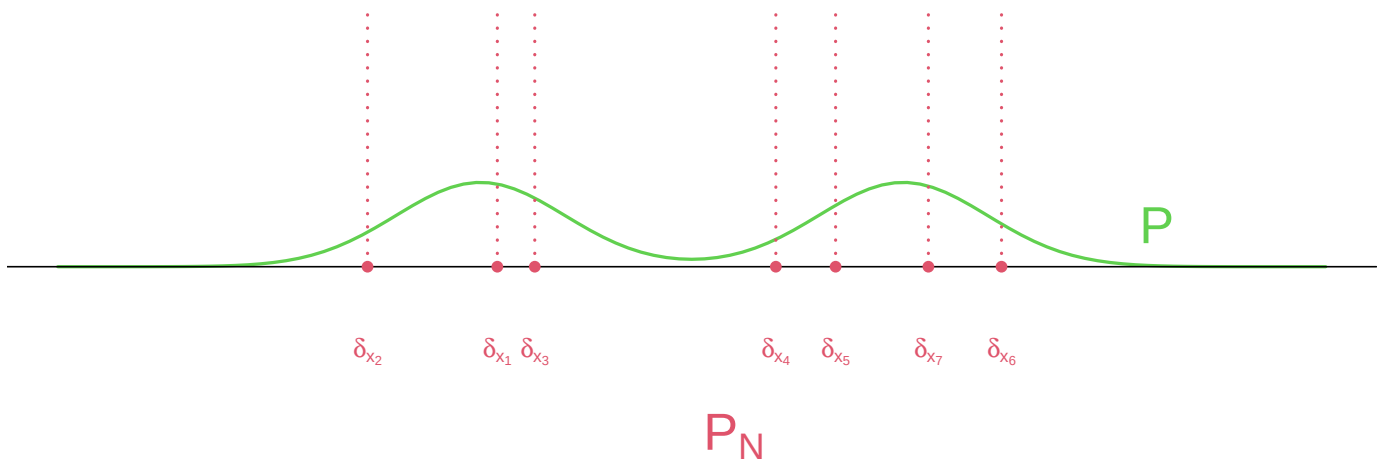
Data augmentation

Vicinal Risk Minimization (VRM) (Chapelle et al., 2001)

- Create some new examples by adding noise (e.g Gaussian) around the features: $(\mathbf{x}_i + \varepsilon_x, \mathbf{y}_i + \varepsilon_y)$
- Idea: P is approximated by the vicinity distribution

$$P_{v_\sigma}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{N} v_\sigma((\mathbf{x}, \mathbf{y})|\mathbf{x}_i, \mathbf{y}_i)$$

where $v_\sigma(.|\mathbf{x}_i, \mathbf{y}_i)$ is the gaussian density with mean $(\mathbf{x}_i, \mathbf{y}_i)$ and variance σ^2 to be chosen



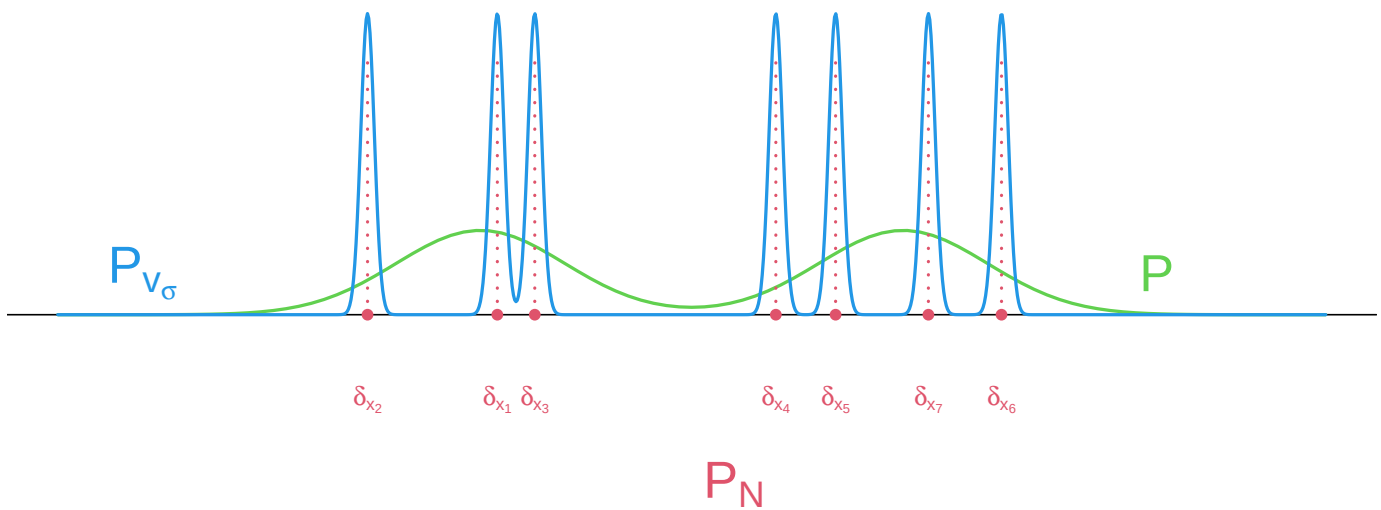
Data augmentation

Vicinal Risk Minimization (VRM) (Chapelle et al., 2001)

- Create some new examples by adding noise (e.g Gaussian) around the features: $(\mathbf{x}_i + \varepsilon_x, \mathbf{y}_i + \varepsilon_y)$
- Idea: P is approximated by the vicinity distribution

$$P_{v_\sigma}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{N} v_\sigma((\mathbf{x}, \mathbf{y})|\mathbf{x}_i, \mathbf{y}_i)$$

where $v_\sigma(.|\mathbf{x}_i, \mathbf{y}_i)$ is the gaussian density with mean $(\mathbf{x}_i, \mathbf{y}_i)$ and variance σ^2 to be chosen



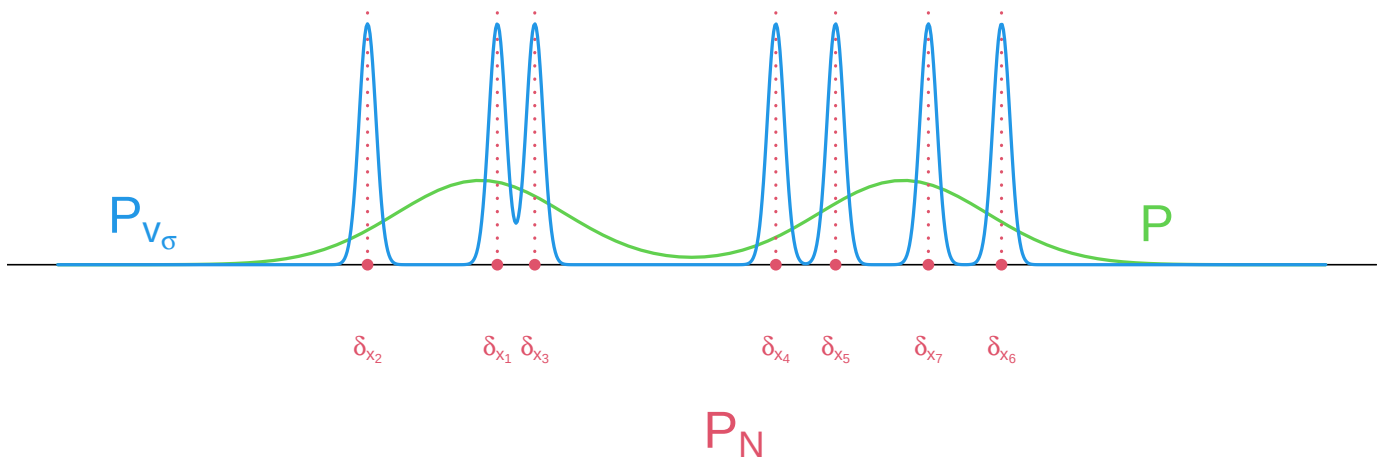
Data augmentation

Vicinal Risk Minimization (VRM) (Chapelle et al., 2001)

- Create some new examples by adding noise (e.g Gaussian) around the features: $(\mathbf{x}_i + \varepsilon_x, \mathbf{y}_i + \varepsilon_y)$
- Idea: P is approximated by the vicinity distribution

$$P_{v_\sigma}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{N} v_\sigma((\mathbf{x}, \mathbf{y})|\mathbf{x}_i, \mathbf{y}_i)$$

where $v_\sigma(.|\mathbf{x}_i, \mathbf{y}_i)$ is the gaussian density with mean $(\mathbf{x}_i, \mathbf{y}_i)$ and variance σ^2 to be chosen



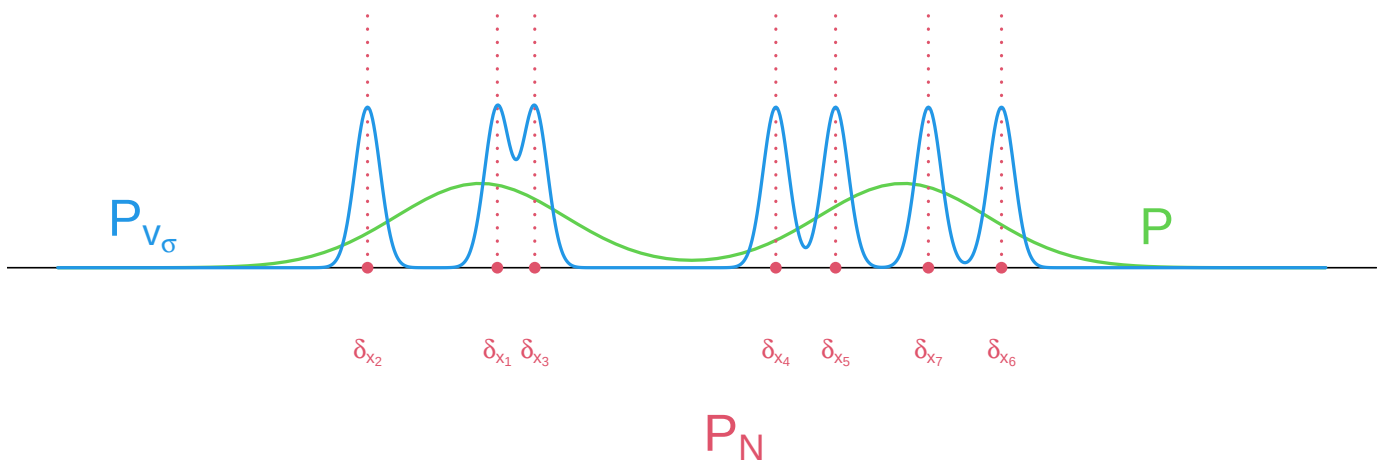
Data augmentation

Vicinal Risk Minimization (VRM) (Chapelle et al., 2001)

- Create some new examples by adding noise (e.g Gaussian) around the features: $(\mathbf{x}_i + \varepsilon_x, \mathbf{y}_i + \varepsilon_y)$
- Idea: P is approximated by the vicinity distribution

$$P_{v_\sigma}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{N} v_\sigma((\mathbf{x}, \mathbf{y})|\mathbf{x}_i, \mathbf{y}_i)$$

where $v_\sigma(.|\mathbf{x}_i, \mathbf{y}_i)$ is the gaussian density with mean $(\mathbf{x}_i, \mathbf{y}_i)$ and variance σ^2 to be chosen



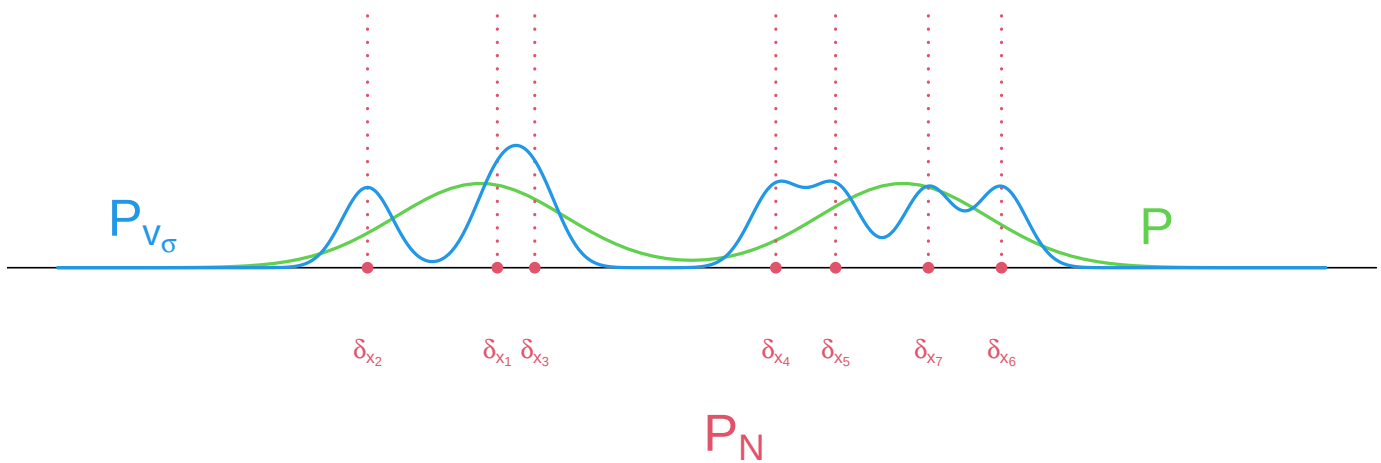
Data augmentation

Vicinal Risk Minimization (VRM) (Chapelle et al., 2001)

- Create some new examples by adding noise (e.g Gaussian) around the features: $(\mathbf{x}_i + \varepsilon_x, \mathbf{y}_i + \varepsilon_y)$
- Idea: P is approximated by the vicinity distribution

$$P_{v_\sigma}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{N} v_\sigma((\mathbf{x}, \mathbf{y})|\mathbf{x}_i, \mathbf{y}_i)$$

where $v_\sigma(.|\mathbf{x}_i, \mathbf{y}_i)$ is the gaussian density with mean $(\mathbf{x}_i, \mathbf{y}_i)$ and variance σ^2 to be chosen



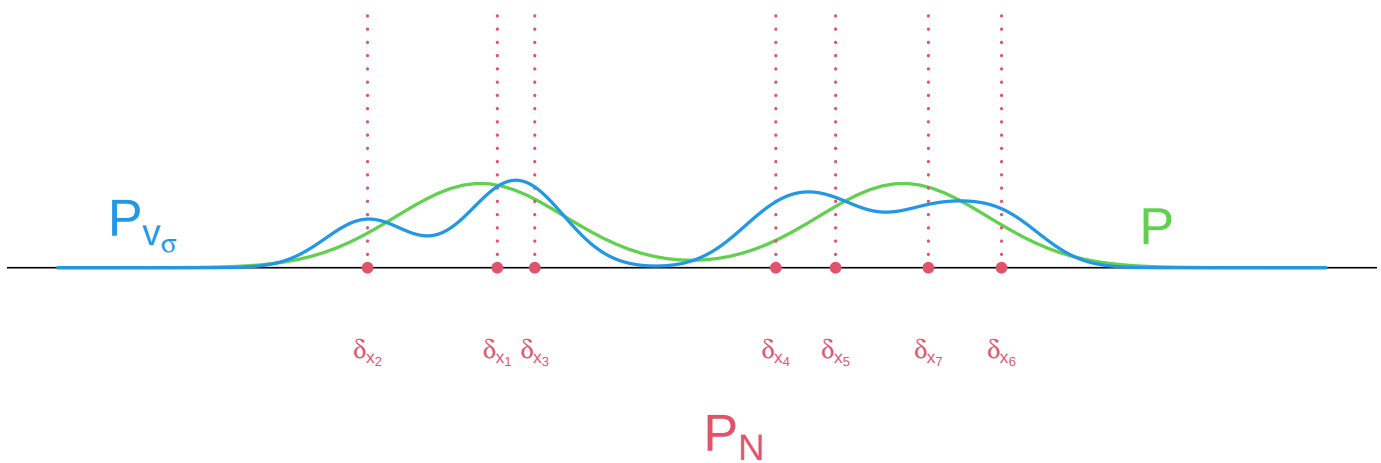
Data augmentation

Vicinal Risk Minimization (VRM) (Chapelle et al., 2001)

- Create some new examples by adding noise (e.g Gaussian) around the features: $(\mathbf{x}_i + \varepsilon_x, \mathbf{y}_i + \varepsilon_y)$
- Idea: P is approximated by the vicinity distribution

$$P_{v_\sigma}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{N} v_\sigma((\mathbf{x}, \mathbf{y})|\mathbf{x}_i, \mathbf{y}_i)$$

where $v_\sigma(.|\mathbf{x}_i, \mathbf{y}_i)$ is the gaussian density with mean $(\mathbf{x}_i, \mathbf{y}_i)$ and variance σ^2 to be chosen



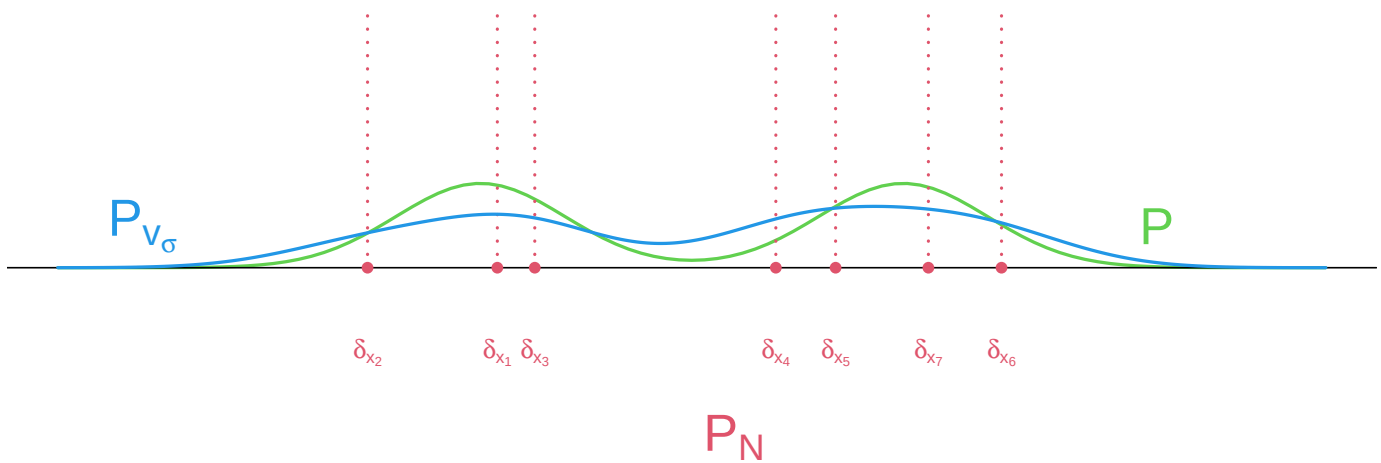
Data augmentation

Vicinal Risk Minimization (VRM) (Chapelle et al., 2001)

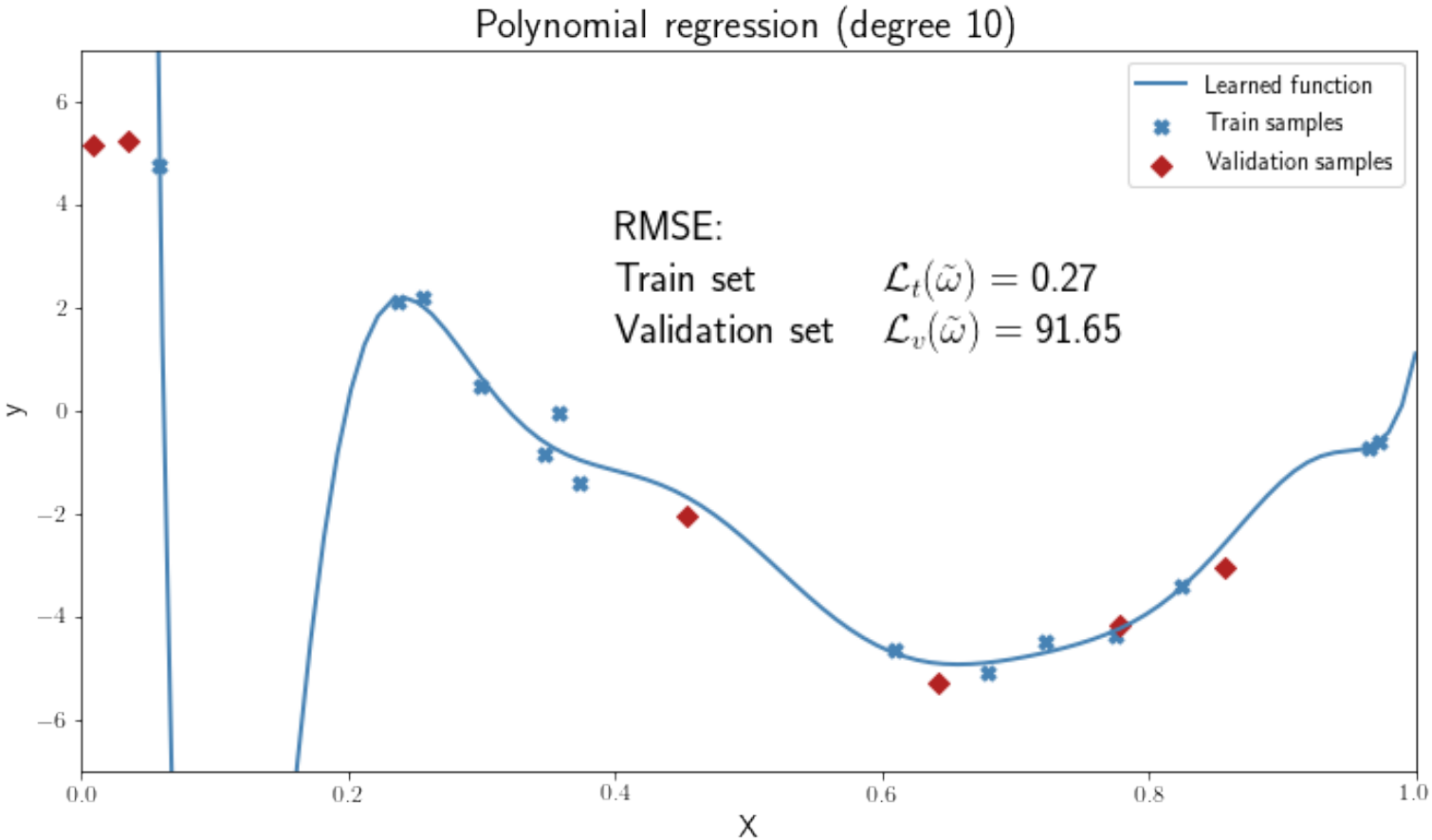
- Create some new examples by adding noise (e.g Gaussian) around the features: $(\mathbf{x}_i + \varepsilon_x, \mathbf{y}_i + \varepsilon_y)$
- Idea: P is approximated by the vicinity distribution

$$P_{v_\sigma}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{N} v_\sigma((\mathbf{x}, \mathbf{y})|\mathbf{x}_i, \mathbf{y}_i)$$

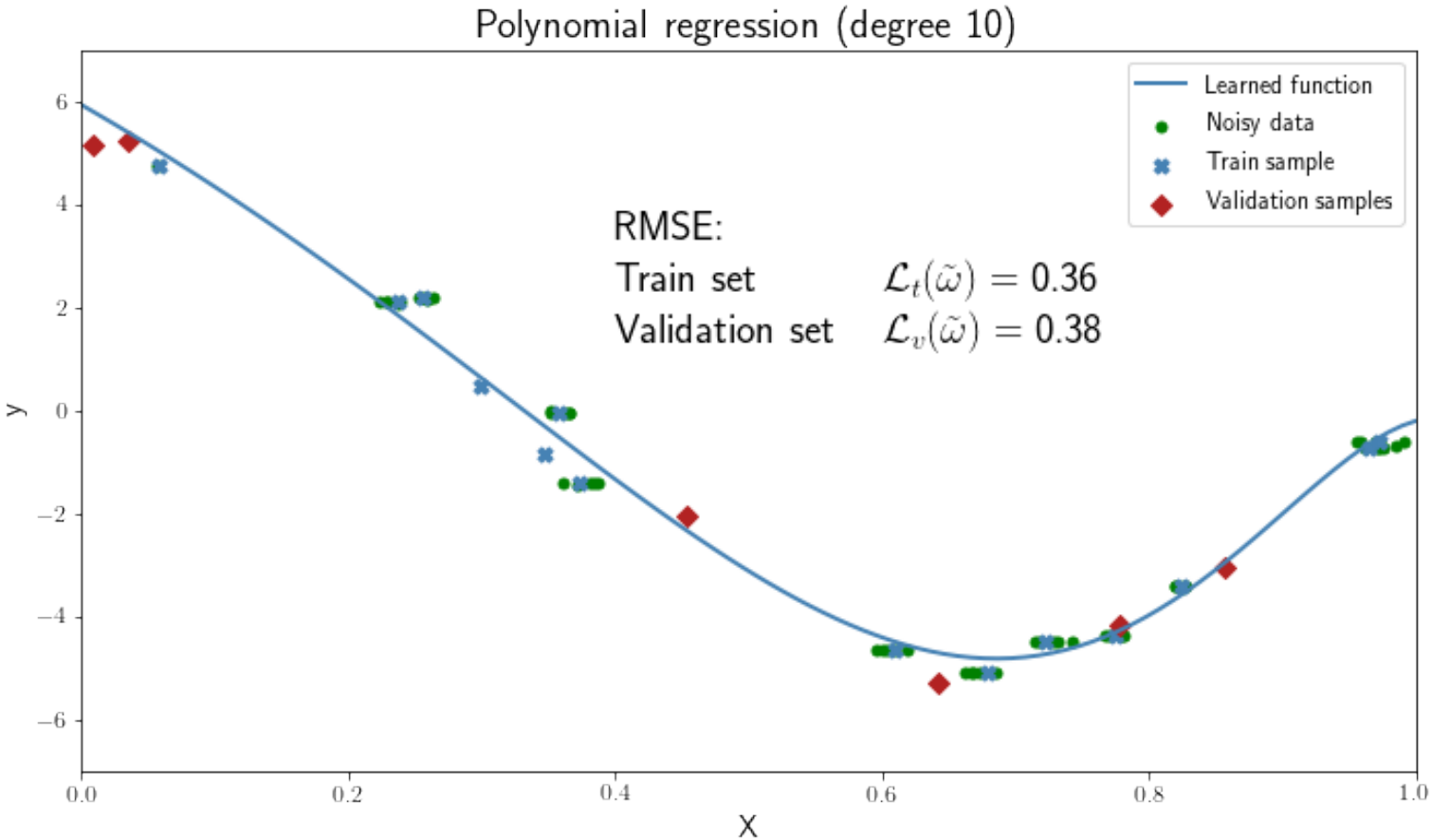
where $v_\sigma(.|\mathbf{x}_i, \mathbf{y}_i)$ is the gaussian density with mean $(\mathbf{x}_i, \mathbf{y}_i)$ and variance σ^2 to be chosen



Data augmentation



Data augmentation



Data augmentation for images

Apply some basic (random) transformations on the images

Original image

Data augmentation for images

Apply some basic (random) transformations on the images

Original image

Examples

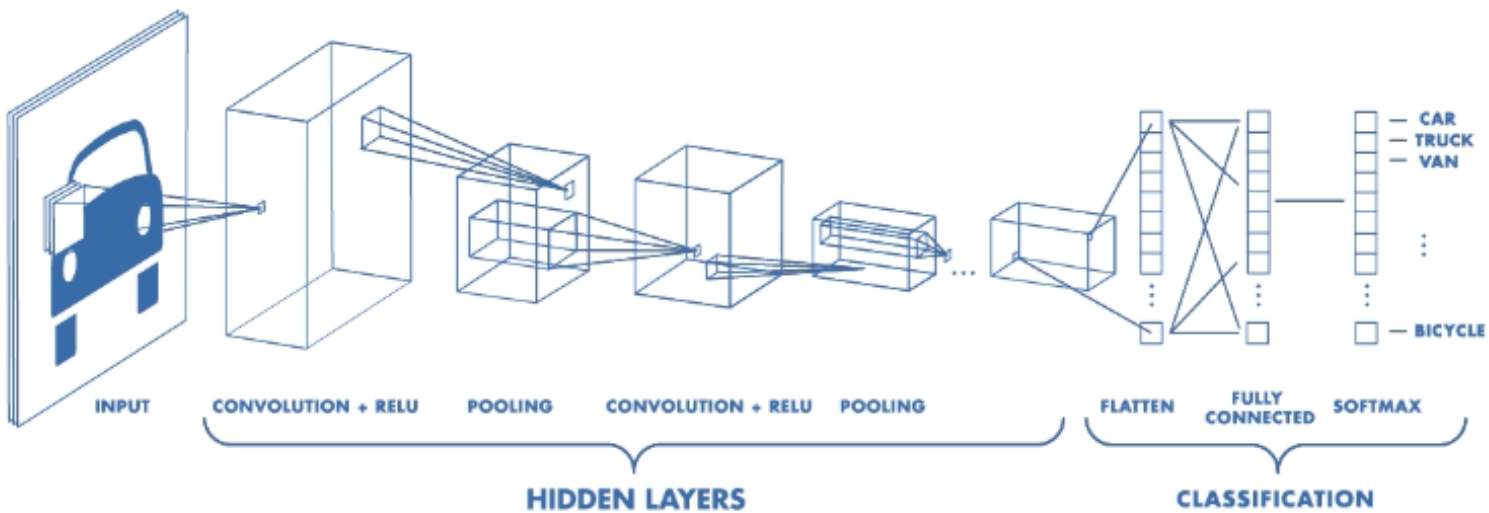
Reflexion

Contrast change

Zoom

Idea

- First, build a hierarchy of 2d representations called feature map (presence or absence of specific features in different part of the image)
- Then classify based on the organisation of the different features
- Everything is learned by back-propagation



Convolutional neural networks (CNN or ConvNet)

- With standard image resolutions, the number of input can be large (for instance $3 \times 100 \times 100 = 30000$) and fully connected layers leads to a huge number of parameters
- However, building representations with all the pixels is a waste of parameters
- Classification is based on the organisation in the image of local features (edges, presence of shapes,...)
- Detection of local features can be made with "stationary" local weights

Edge detection (vertical)

3×3 filter (or kernel)

Input image

20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0

Filter

1	0	-1
1	0	-1
1	0	-1

Output image

Edge detection (vertical)

3×3 filter (or kernel)

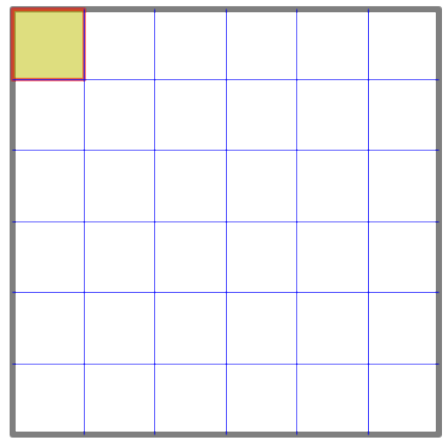
Input image

20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0

Filter

1	0	-1
1	0	-1
1	0	-1

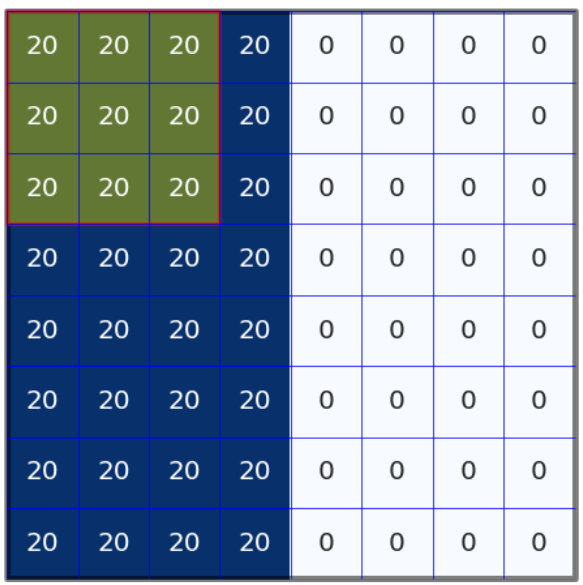
Output image



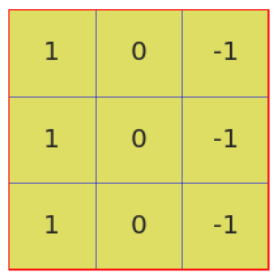
Edge detection (vertical)

3×3 filter (or kernel)

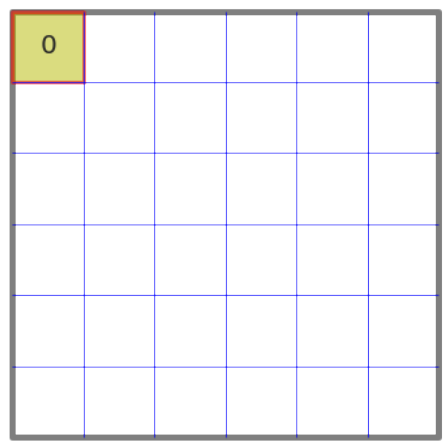
Input image



Filter



Output image



Edge detection (vertical)

3×3 filter (or kernel)

Input image

20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0

Filter

1	0	-1
1	0	-1
1	0	-1

Output image

0	0				

Edge detection (vertical)

3×3 filter (or kernel)

Input image

20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0

Filter

1	0	-1
1	0	-1
1	0	-1

Output image

0	0	60			

Edge detection (vertical)

3×3 filter (or kernel)

Input image

20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0

Filter

1	0	-1
1	0	-1
1	0	-1

Output image

0	0	60	60		

Edge detection (vertical)

3×3 filter (or kernel)

Input image

20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0

Filter

1	0	-1
1	0	-1
1	0	-1

Output image

0	0	60	60	0	

Edge detection (vertical)

3×3 filter (or kernel)

Input image

20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0

Filter

1	0	-1
1	0	-1
1	0	-1

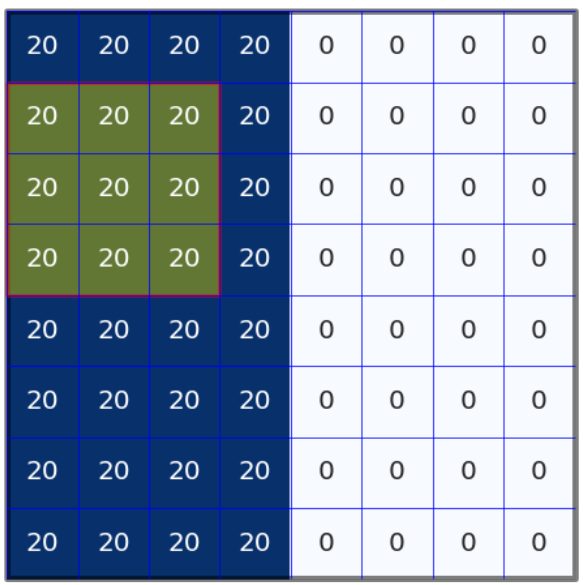
Output image

0	0	60	60	0	0

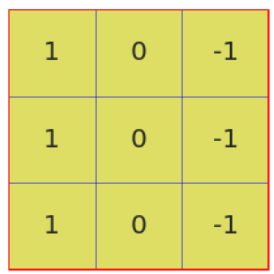
Edge detection (vertical)

3×3 filter (or kernel)

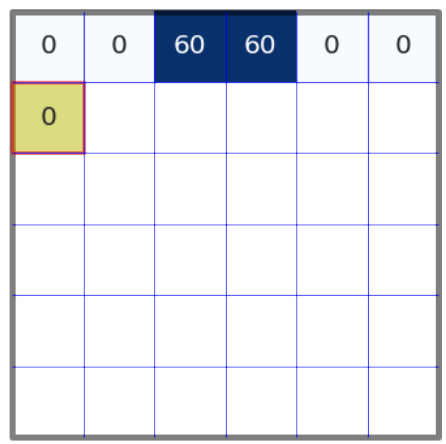
Input image



Filter



Output image



Edge detection (vertical)

3×3 filter (or kernel)

Input image

20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0

Filter

1	0	-1
1	0	-1
1	0	-1

Output image

0	0	60	60	0	0
0	0				

Edge detection (vertical)

3×3 filter (or kernel)

Input image

20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0

Filter

1	0	-1
1	0	-1
1	0	-1

Output image

0	0	60	60	0	0
0	0	60	60	0	0
0	0	60	60	0	0
0	0	60	60	0	0
0	0	60	60	0	0
0	0	60	60	0	0

Edge detection (vertical)

3×3 filter (or kernel)

Input image

20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0

Filter

1	0	-1
1	0	-1
1	0	-1

Output image

0	0	60	60	0	0
0	0	60	60	0	0
0	0	60	60	0	0
0	0	60	60	0	0
0	0	60	60	0	0
0	0	60	60	0	0

Example on a "real" image

Example on a "real" image

1	1	1
0	0	0
-1	-1	-1

Horizontal edges detection

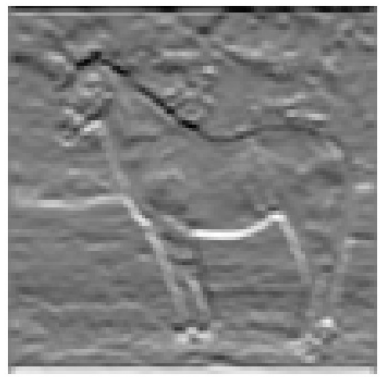
1	0	-1
1	0	-1
1	0	-1

Vertical edges detection

Example on a "real" image

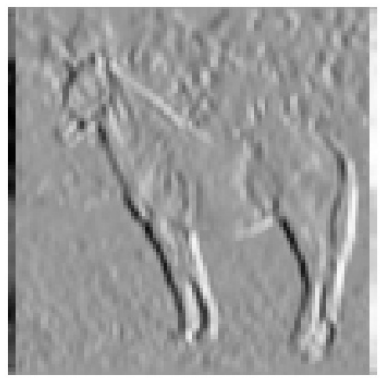
1	1	1
0	0	0
-1	-1	-1

Horizontal edges detection

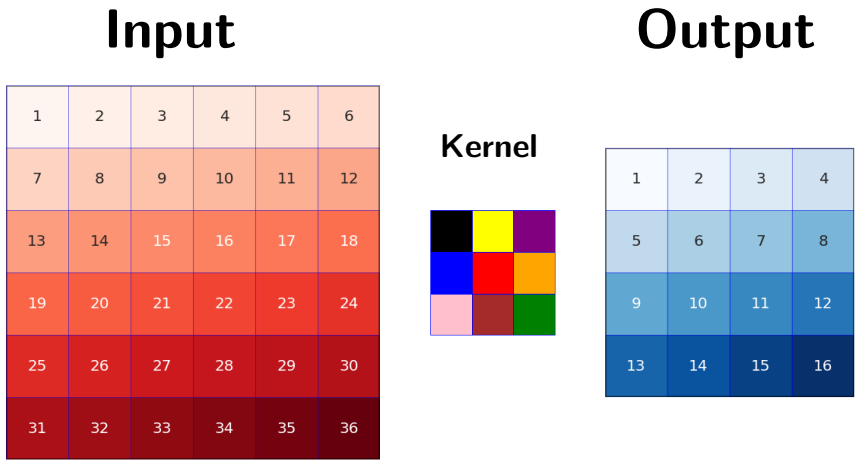


1	0	-1
1	0	-1
1	0	-1

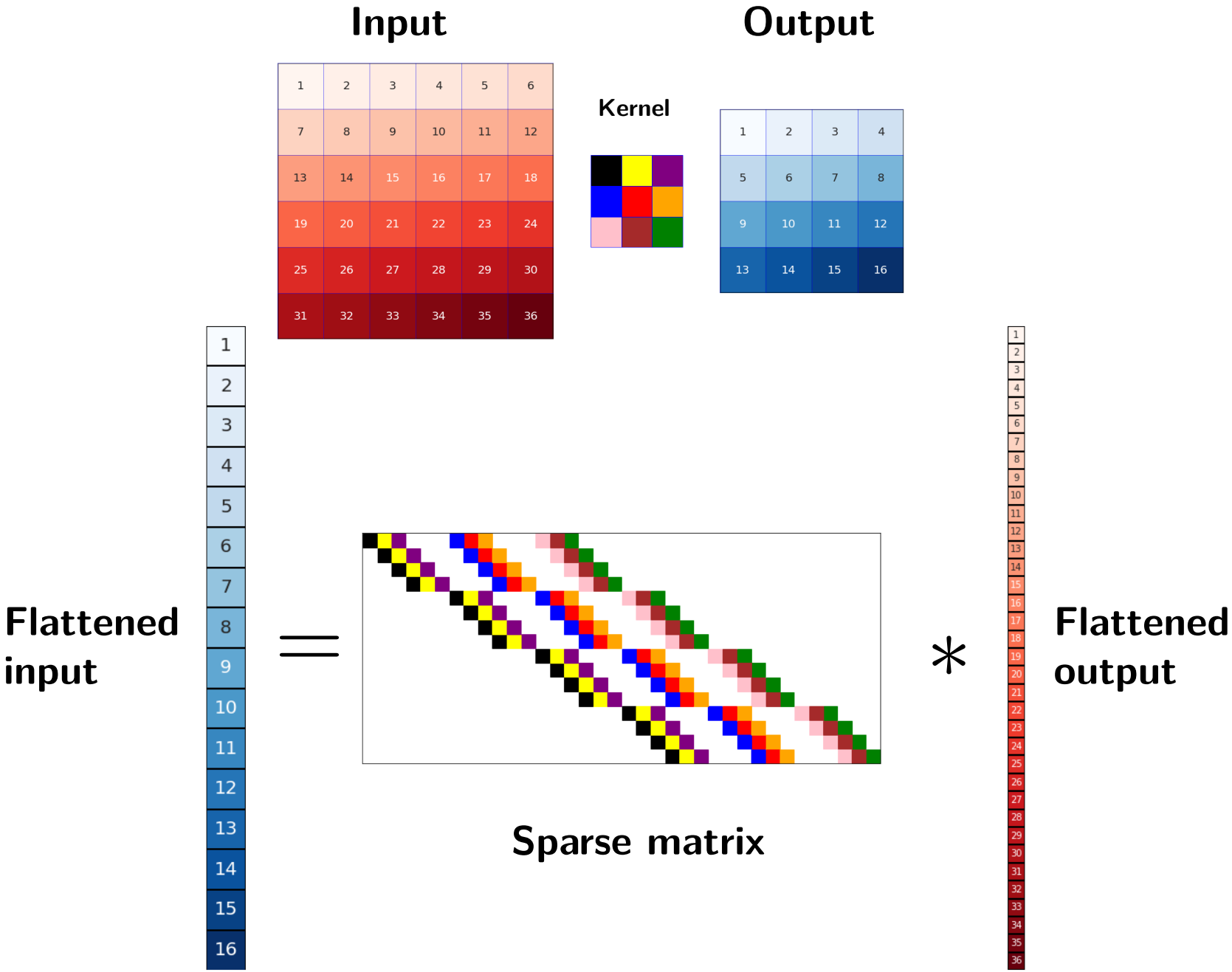
Vertical edges detection



Link with neural networks



Link with neural networks



Padding

3×3 filter - 1×1 padding

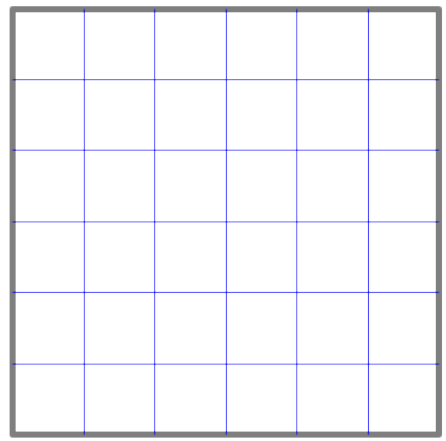
Input image

20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0
20	20	20	20	0	0	0	0

Filter

1	0	-1
1	0	-1
1	0	-1

Output image



Padding

3×3 filter - 1×1 padding

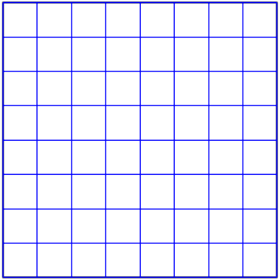
Input image

0	0	0	0	0	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Filter

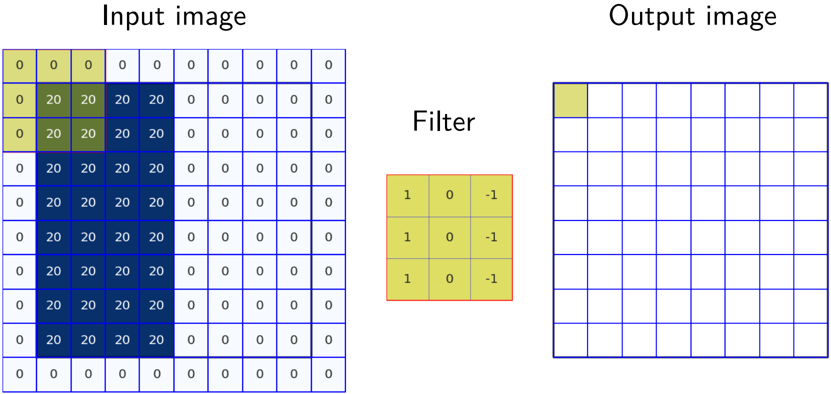
1	0	-1
1	0	-1
1	0	-1

Output image



Padding

3×3 filter - 1×1 padding



Padding

3×3 filter - 1×1 padding

Input image

0	0	0	0	0	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Filter

1	0	-1
1	0	-1
1	0	-1

Output image

-40							

Padding

3×3 filter - 1×1 padding

Input image

0	0	0	0	0	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Filter

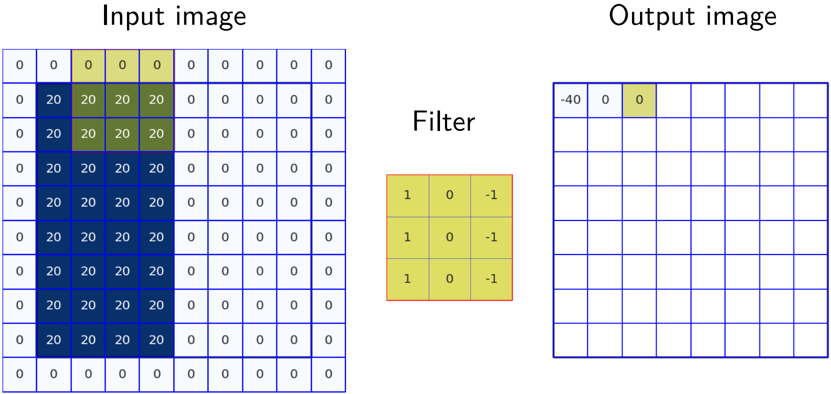
1	0	-1
1	0	-1
1	0	-1

Output image

-40	0						

Padding

3×3 filter - 1×1 padding



Padding

3×3 filter - 1×1 padding

Input image

0	0	0	0	0	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Filter

1	0	-1
1	0	-1
1	0	-1

Output image

-40	0	0	40				

Padding

3×3 filter - 1×1 padding

Input image

0	0	0	0	0	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Filter

1	0	-1
1	0	-1
1	0	-1

Output image

-40	0	0	40	40			

Padding

3×3 filter - 1×1 padding

Input image

0	0	0	0	0	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Filter

1	0	-1
1	0	-1
1	0	-1

Output image

-40	0	0	40	40	0		

Padding

3×3 filter - 1×1 padding

Input image

0	0	0	0	0	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Filter

1	0	-1
1	0	-1
1	0	-1

Output image

-40	0	0	40	40	0	0	

Padding

3×3 filter - 1×1 padding

Input image

0	0	0	0	0	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Filter

1	0	-1
1	0	-1
1	0	-1

Output image

-40	0	0	40	40	0	0	0

Padding

3×3 filter - 1×1 padding

Input image

0	0	0	0	0	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Filter

1	0	-1
1	0	-1
1	0	-1

Output image

-40	0	0	40	40	0	0	0
-60							

Padding

3×3 filter - 1×1 padding

Input image

0	0	0	0	0	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Filter

1	0	-1
1	0	-1
1	0	-1

Output image

-40	0	0	40	40	0	0	0
-60	0						

Padding

3×3 filter - 1×1 padding

Input image

0	0	0	0	0	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Filter

1	0	-1
1	0	-1
1	0	-1

Output image

-40	0	0	40	40	0	0	0
-60	0	0					

Padding

3×3 filter - 1×1 padding

Input image

0	0	0	0	0	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Filter

1	0	-1
1	0	-1
1	0	-1

Output image

-40	0	0	40	40	0	0	0
-60	0	0	60				

Padding

3×3 filter - 1×1 padding

Input image

0	0	0	0	0	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Filter

1	0	-1
1	0	-1
1	0	-1

Output image

-40	0	0	40	40	0	0	0
-60	0	0	60	60			

Padding

3×3 filter - 1×1 padding

Input image

0	0	0	0	0	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Filter

1	0	-1
1	0	-1
1	0	-1

Output image

-40	0	0	40	40	0	0	0
-60	0	0	60	60	0		

Padding

3×3 filter - 1×1 padding

Input image

0	0	0	0	0	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Filter

1	0	-1
1	0	-1
1	0	-1

Output image

-40	0	0	40	40	0	0	0
-60	0	0	60	60	0	0	

Padding

3×3 filter - 1×1 padding

Input image

0	0	0	0	0	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Filter

1	0	-1
1	0	-1
1	0	-1

Output image

-40	0	0	40	40	0	0	0
-60	0	0	60	60	0	0	0

Padding

3×3 filter - 1×1 padding

Input image

0	0	0	0	0	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Filter

1	0	-1
1	0	-1
1	0	-1

Output image

-40	0	0	40	40	0	0	0
-60	0	0	60	60	0	0	0
-60	0	0	60	60	0	0	0
-60	0	0	60	60	0	0	0
-60	0	0	60	60	0	0	0
-60	0	0	60	60	0	0	0
-60	0	0	60	60	0	0	0
-40	0	0	40	40	0	0	0

Padding

3×3 filter - 1×1 padding

Input image

0	0	0	0	0	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	20	20	20	20	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

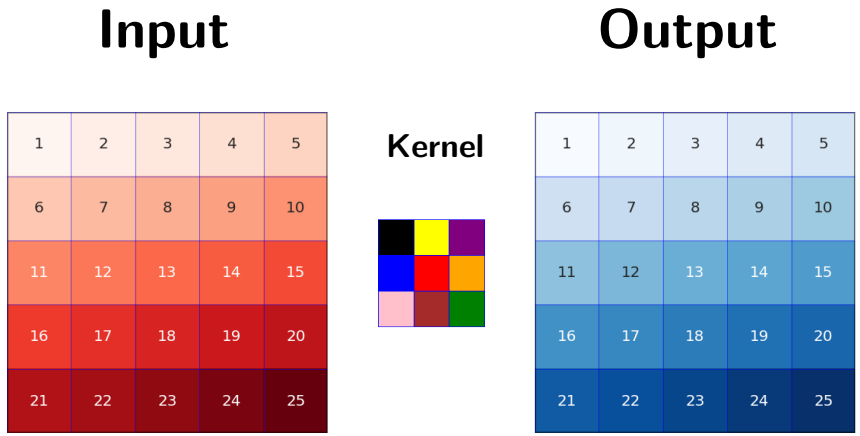
Filter

1	0	-1
1	0	-1
1	0	-1

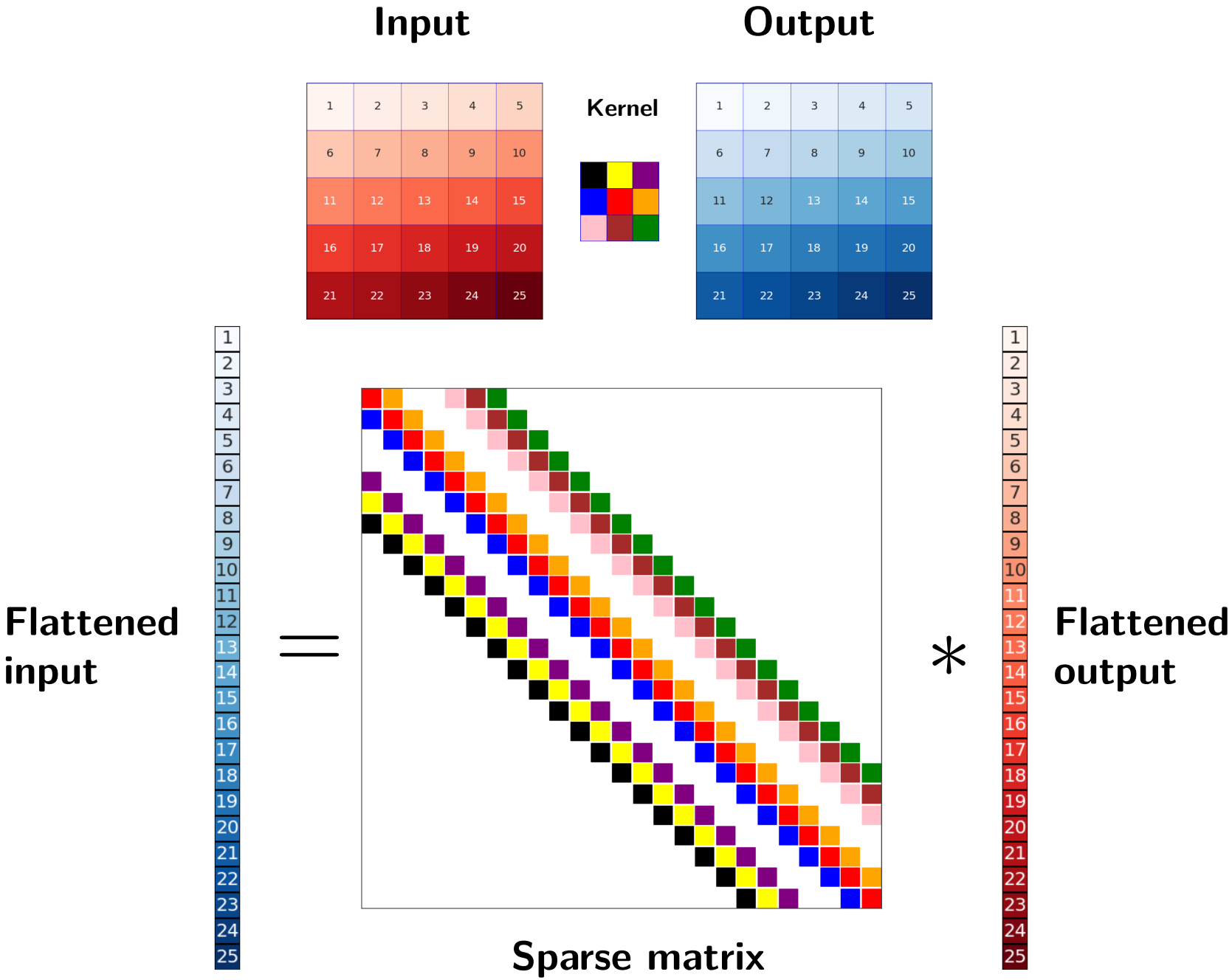
Output image

-40	0	0	40	40	0	0	0
-60	0	0	60	60	0	0	0
-60	0	0	60	60	0	0	0
-60	0	0	60	60	0	0	0
-60	0	0	60	60	0	0	0
-60	0	0	60	60	0	0	0
-60	0	0	60	60	0	0	0
-40	0	0	40	40	0	0	0

With padding

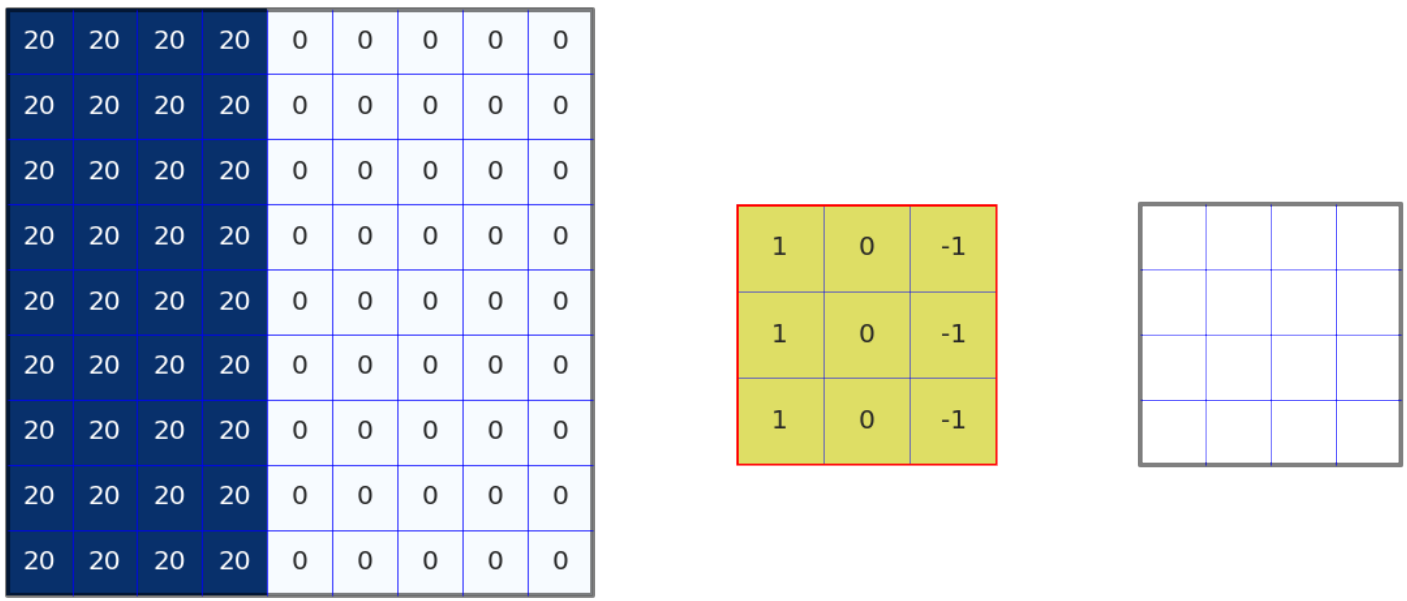


With padding



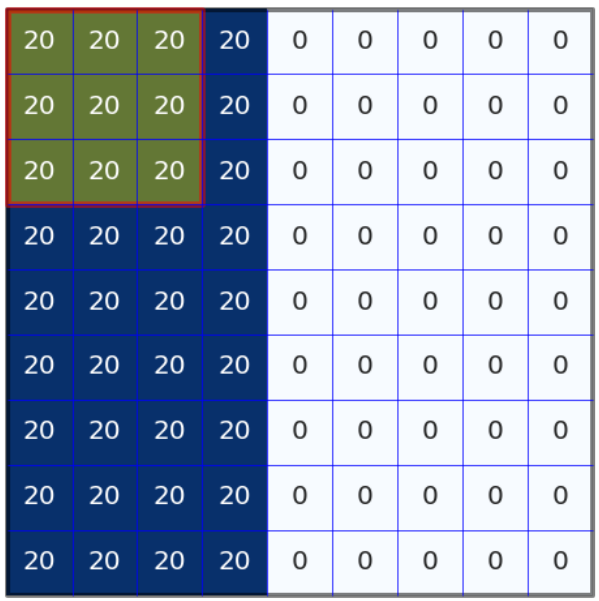
Stride

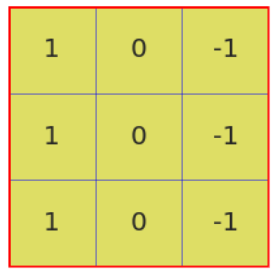
3×3 filter - No padding - 2×2 stride

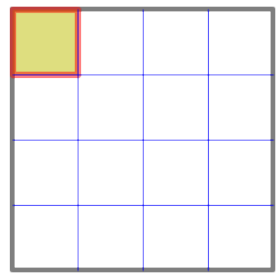


Stride

3×3 filter - No padding - 2×2 stride

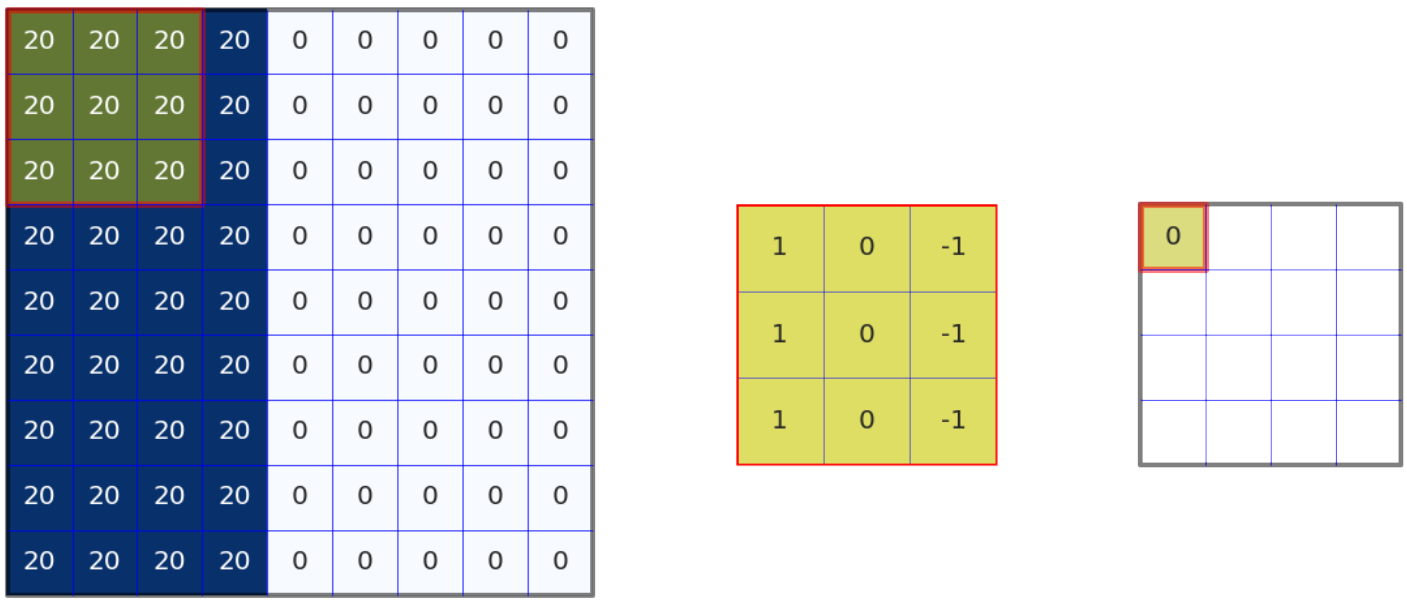






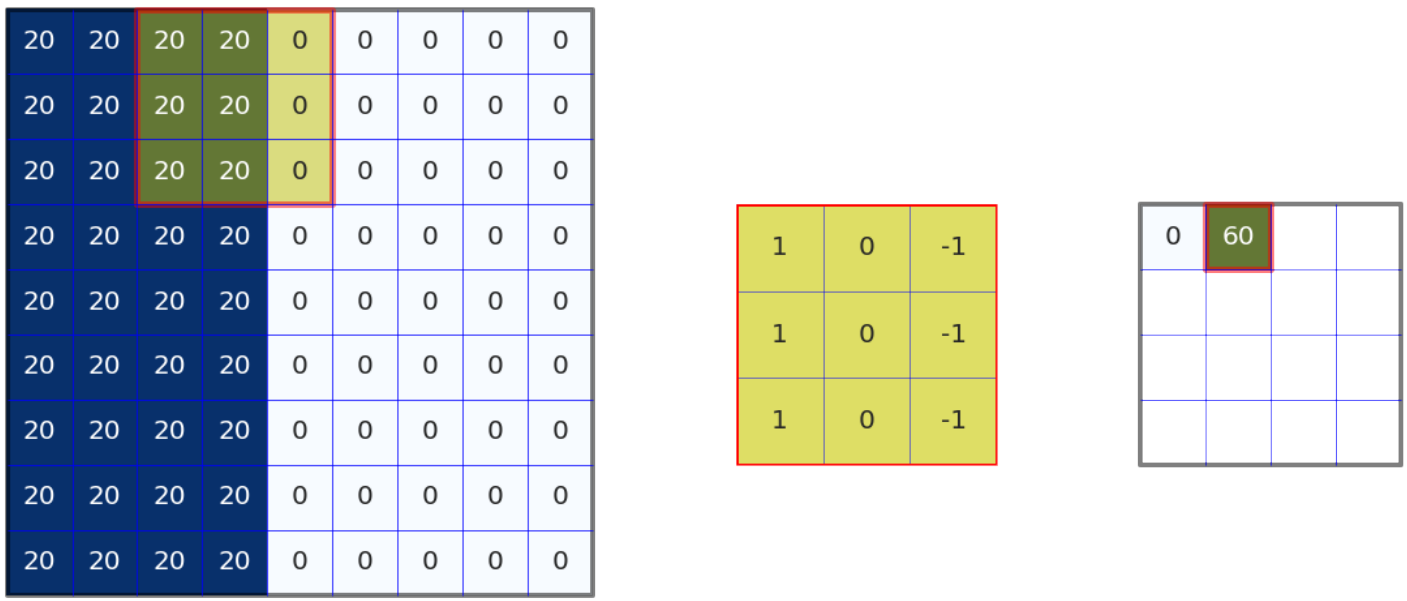
Stride

3×3 filter - No padding - 2×2 stride



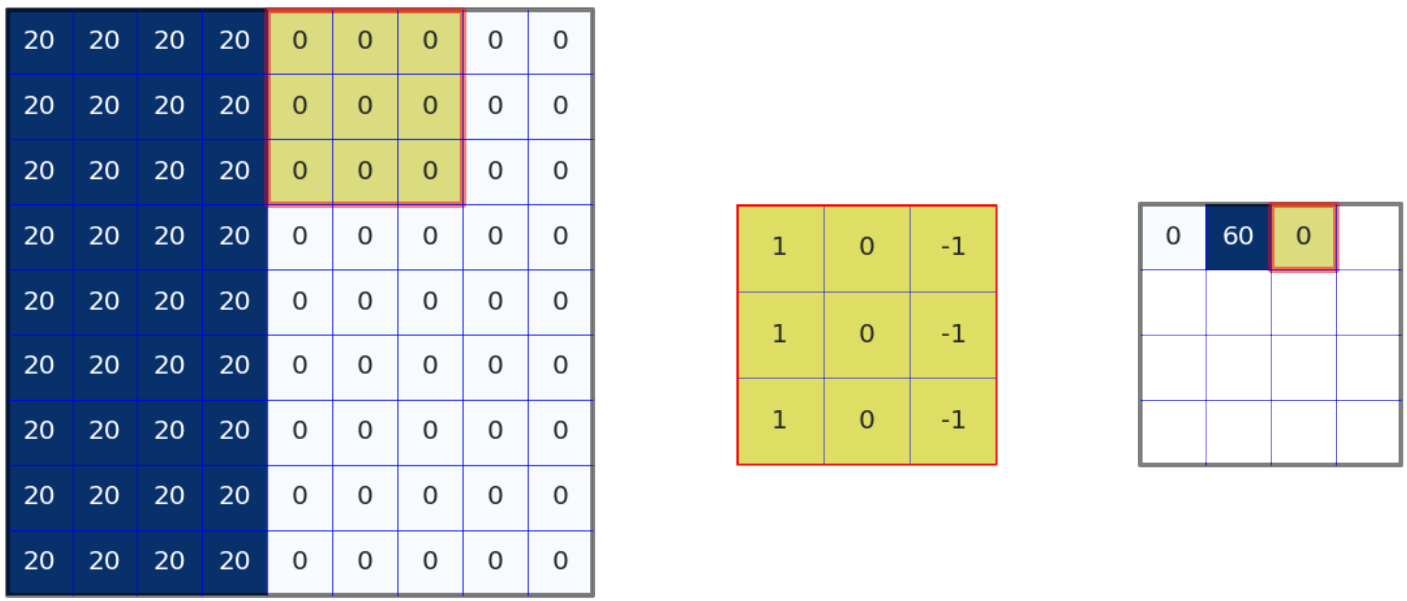
Stride

3×3 filter - No padding - 2×2 stride



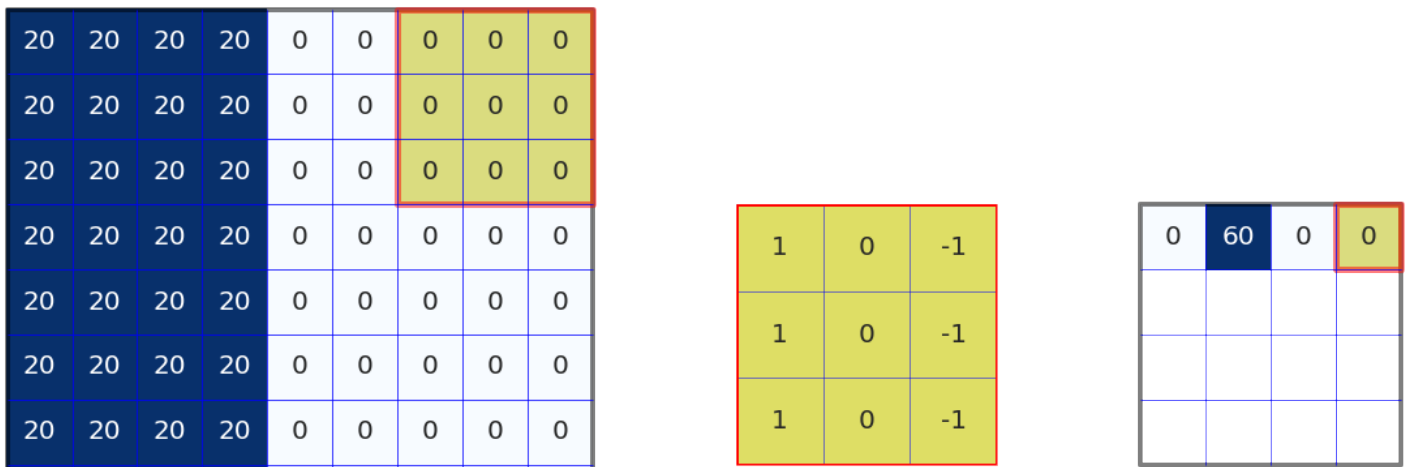
Stride

3×3 filter - No padding - 2×2 stride



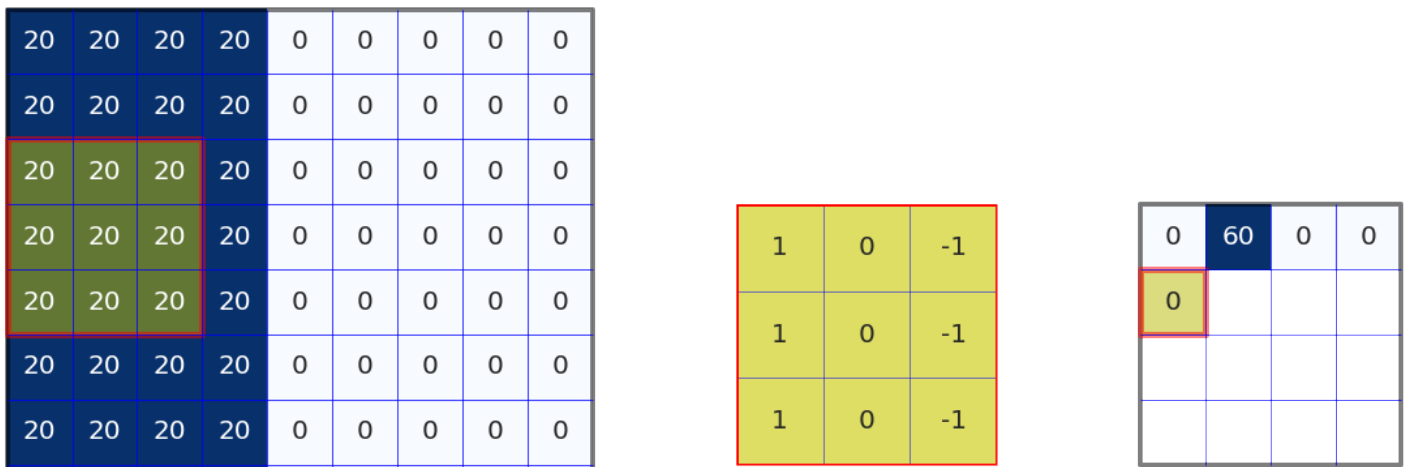
Stride

3×3 filter - No padding - 2×2 stride



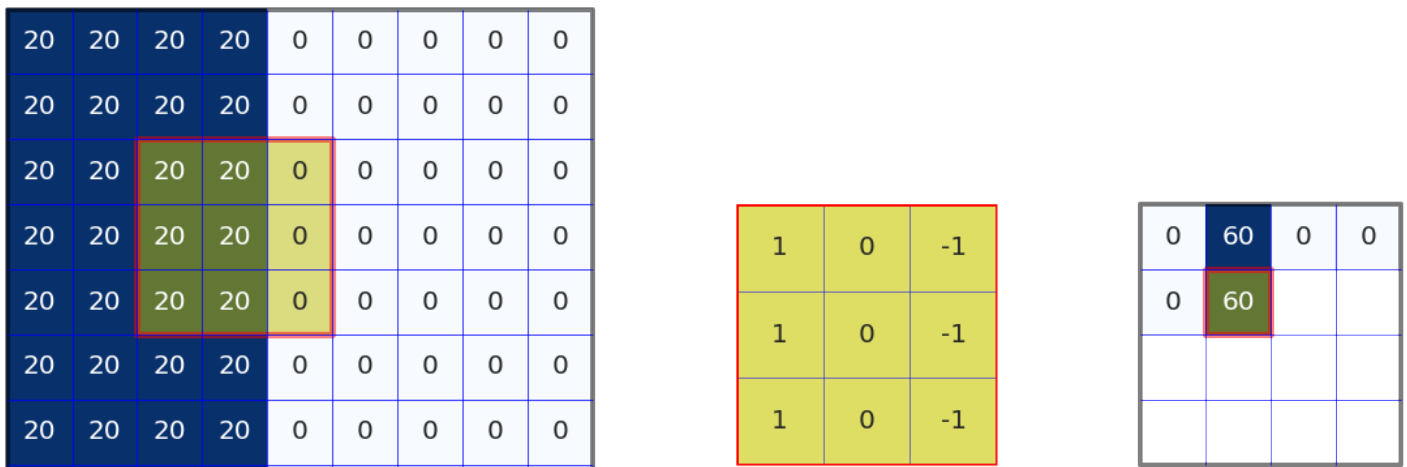
Stride

3×3 filter - No padding - 2×2 stride



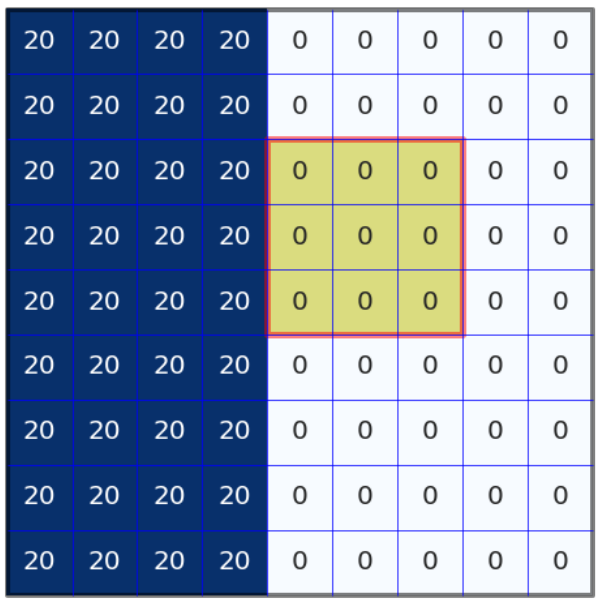
Stride

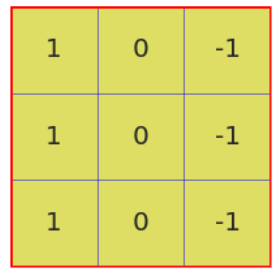
3×3 filter - No padding - 2×2 stride

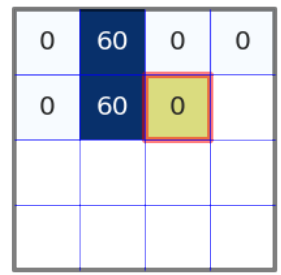


Stride

3×3 filter - No padding - 2×2 stride

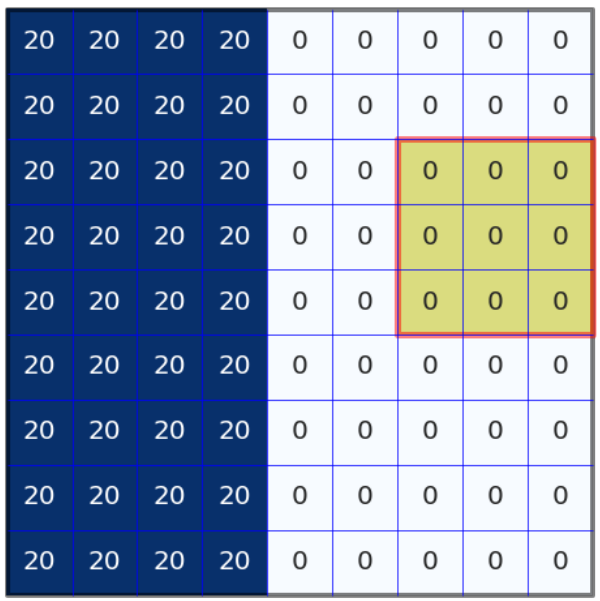


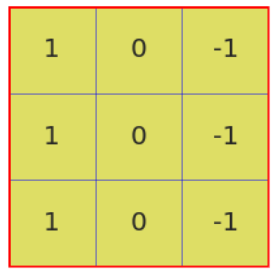


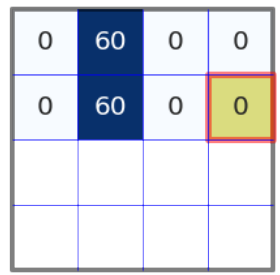


Stride

3×3 filter - No padding - 2×2 stride

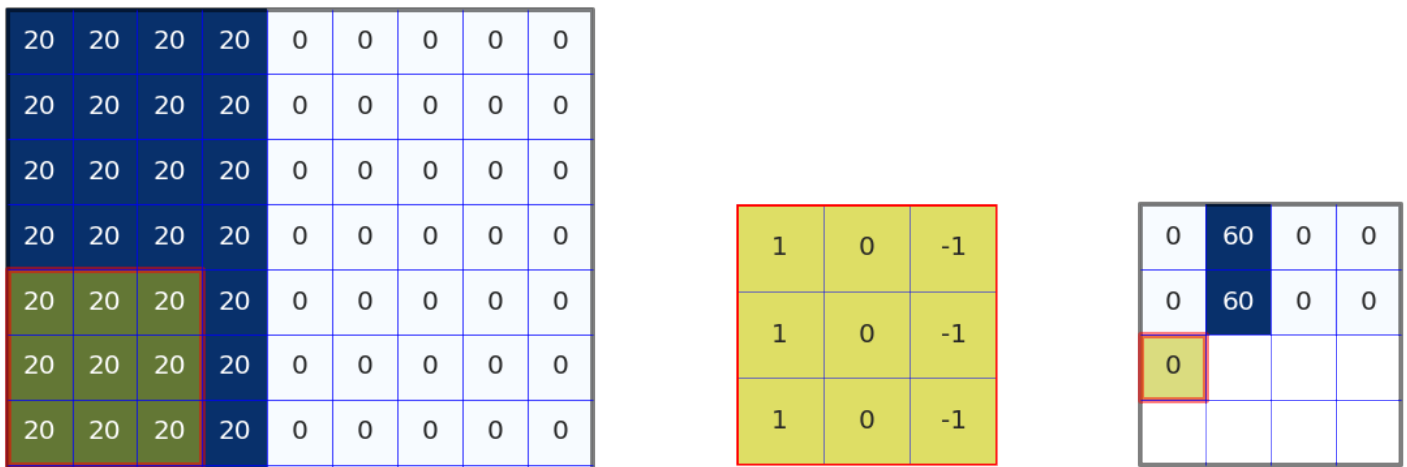






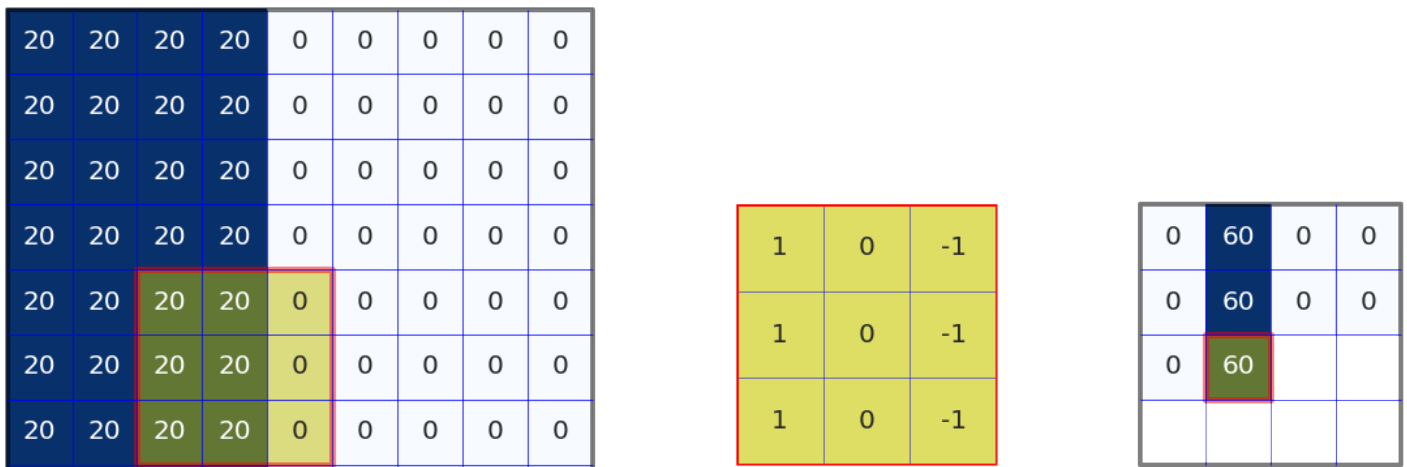
Stride

3×3 filter - No padding - 2×2 stride



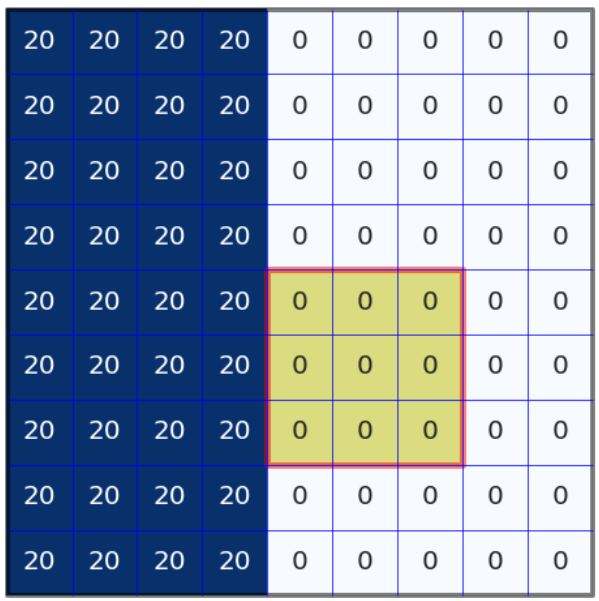
Stride

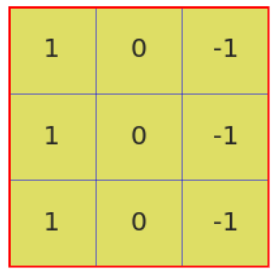
3×3 filter - No padding - 2×2 stride

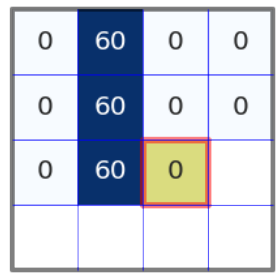


Stride

3×3 filter - No padding - 2×2 stride

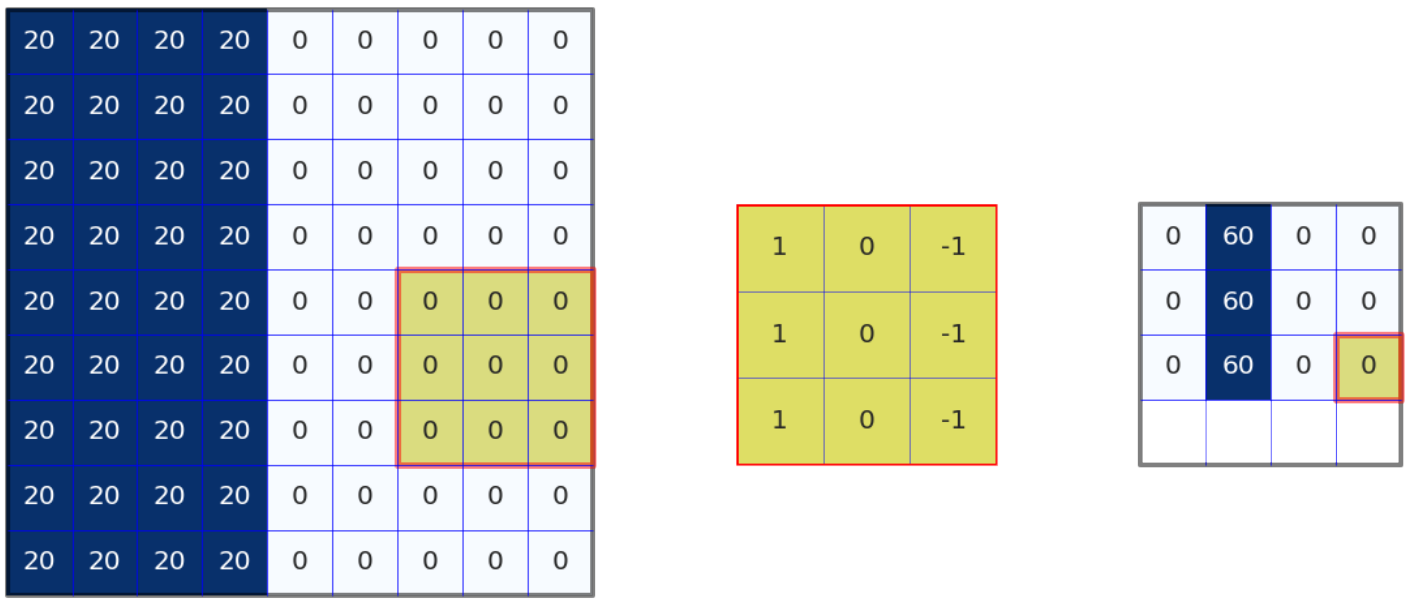






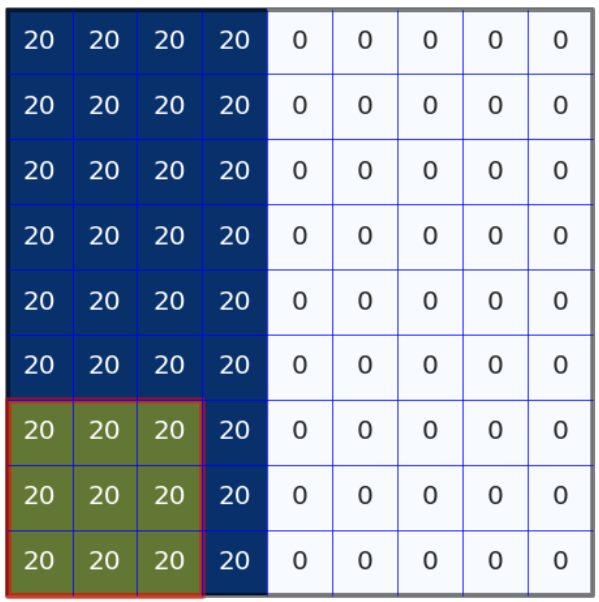
Stride

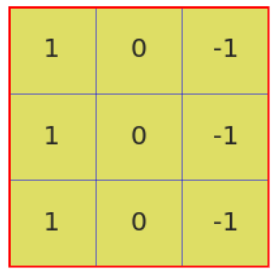
3×3 filter - No padding - 2×2 stride

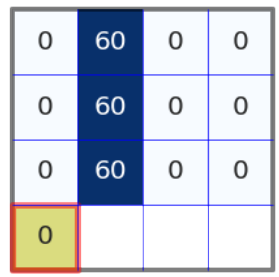


Stride

3×3 filter - No padding - 2×2 stride

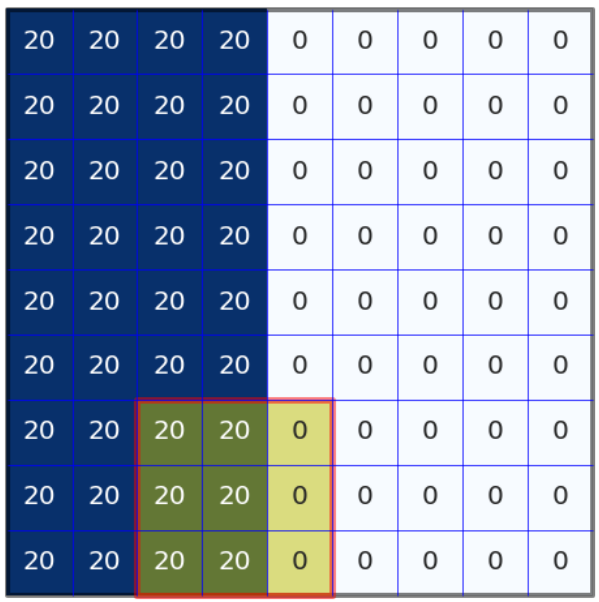


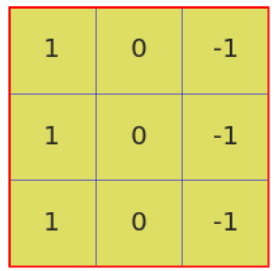


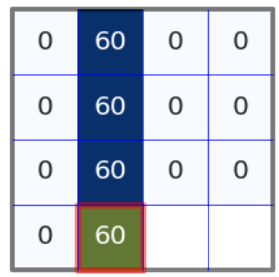


Stride

3×3 filter - No padding - 2×2 stride

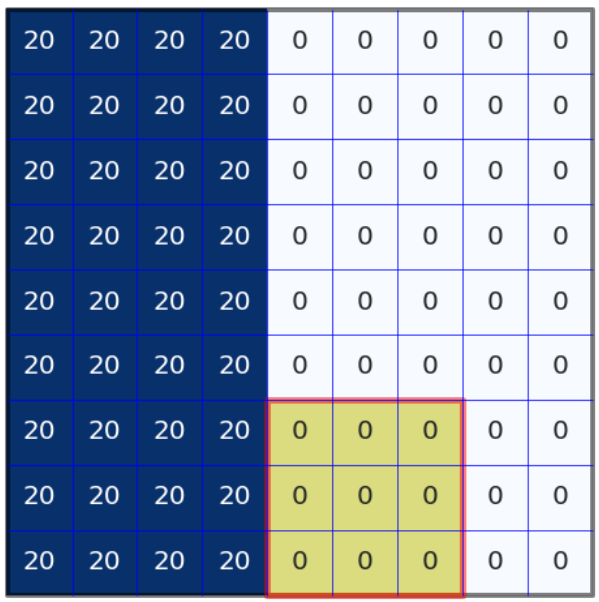


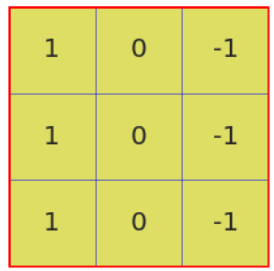


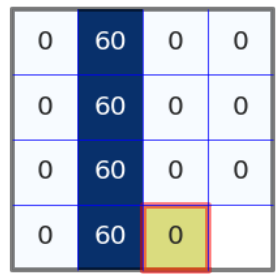


Stride

3×3 filter - No padding - 2×2 stride

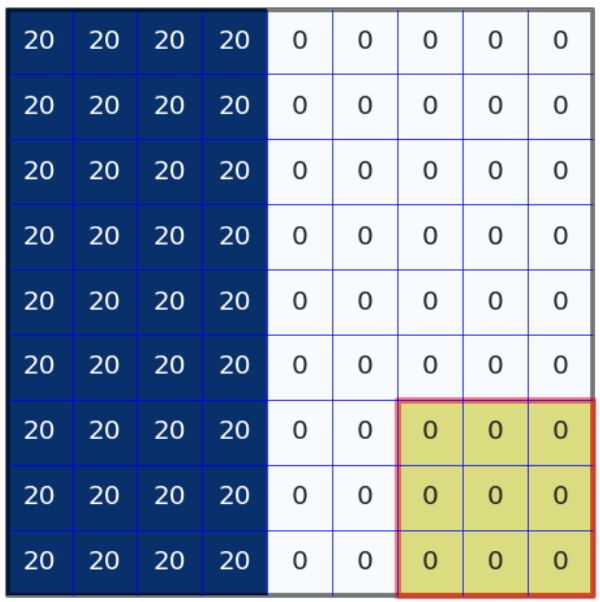


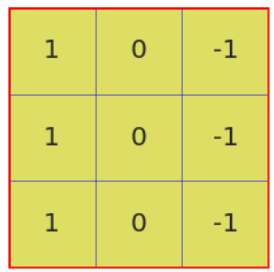


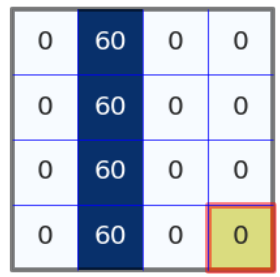


Stride

3×3 filter - No padding - 2×2 stride







Stride

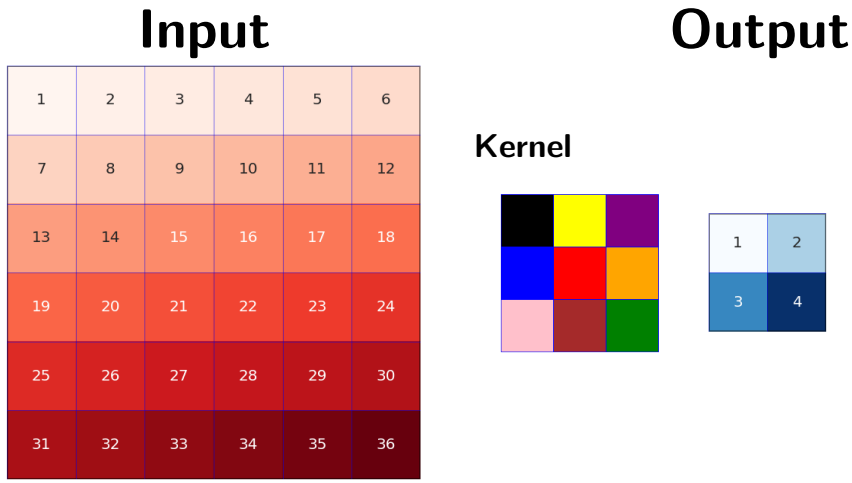
3×3 filter - No padding - 2×2 stride

20	20	20	20	0	0	0	0	0
20	20	20	20	0	0	0	0	0
20	20	20	20	0	0	0	0	0
20	20	20	20	0	0	0	0	0
20	20	20	20	0	0	0	0	0
20	20	20	20	0	0	0	0	0
20	20	20	20	0	0	0	0	0
20	20	20	20	0	0	0	0	0
20	20	20	20	0	0	0	0	0

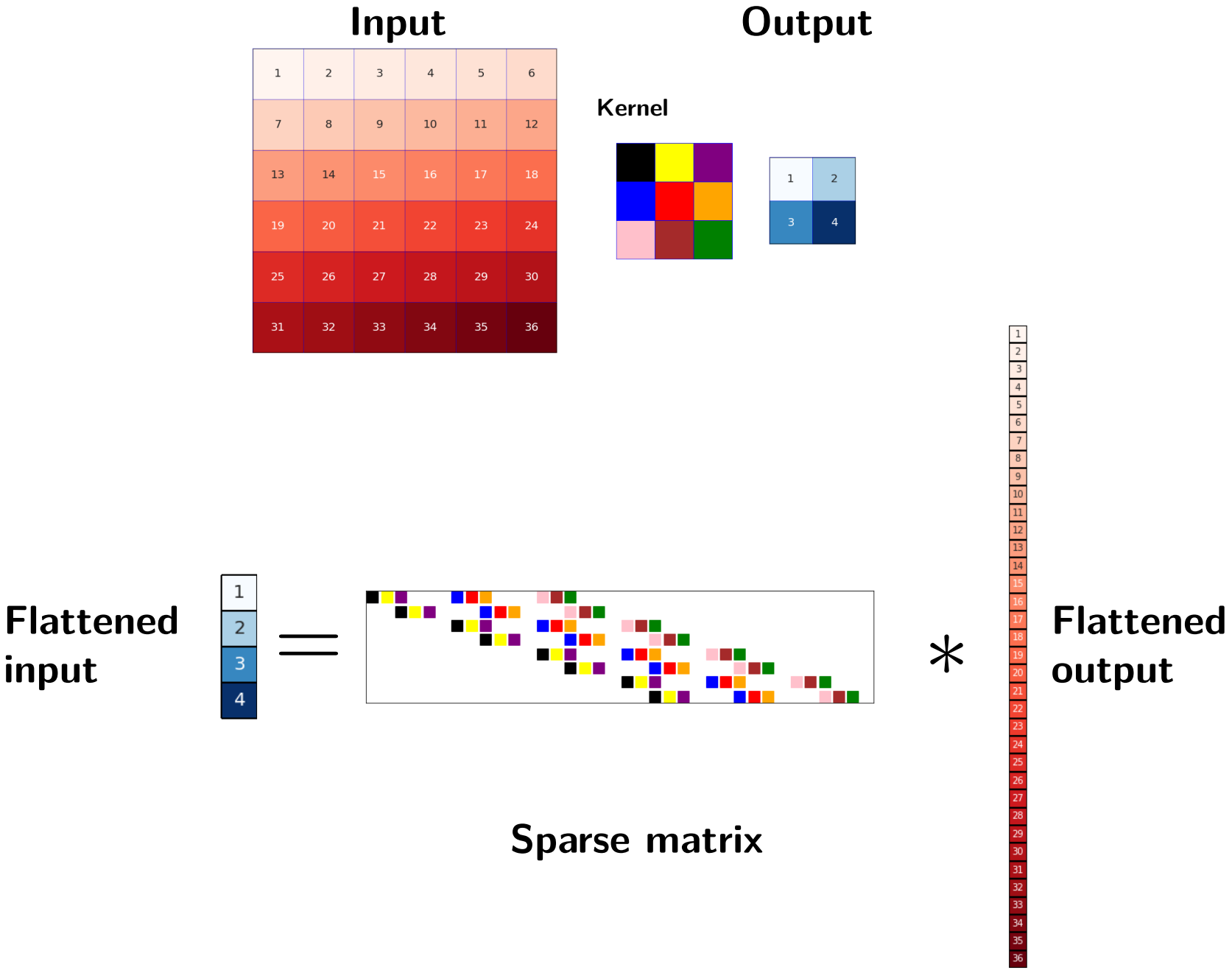
1	0	-1
1	0	-1
1	0	-1

0	60	0	0
0	60	0	0
0	60	0	0
0	60	0	0

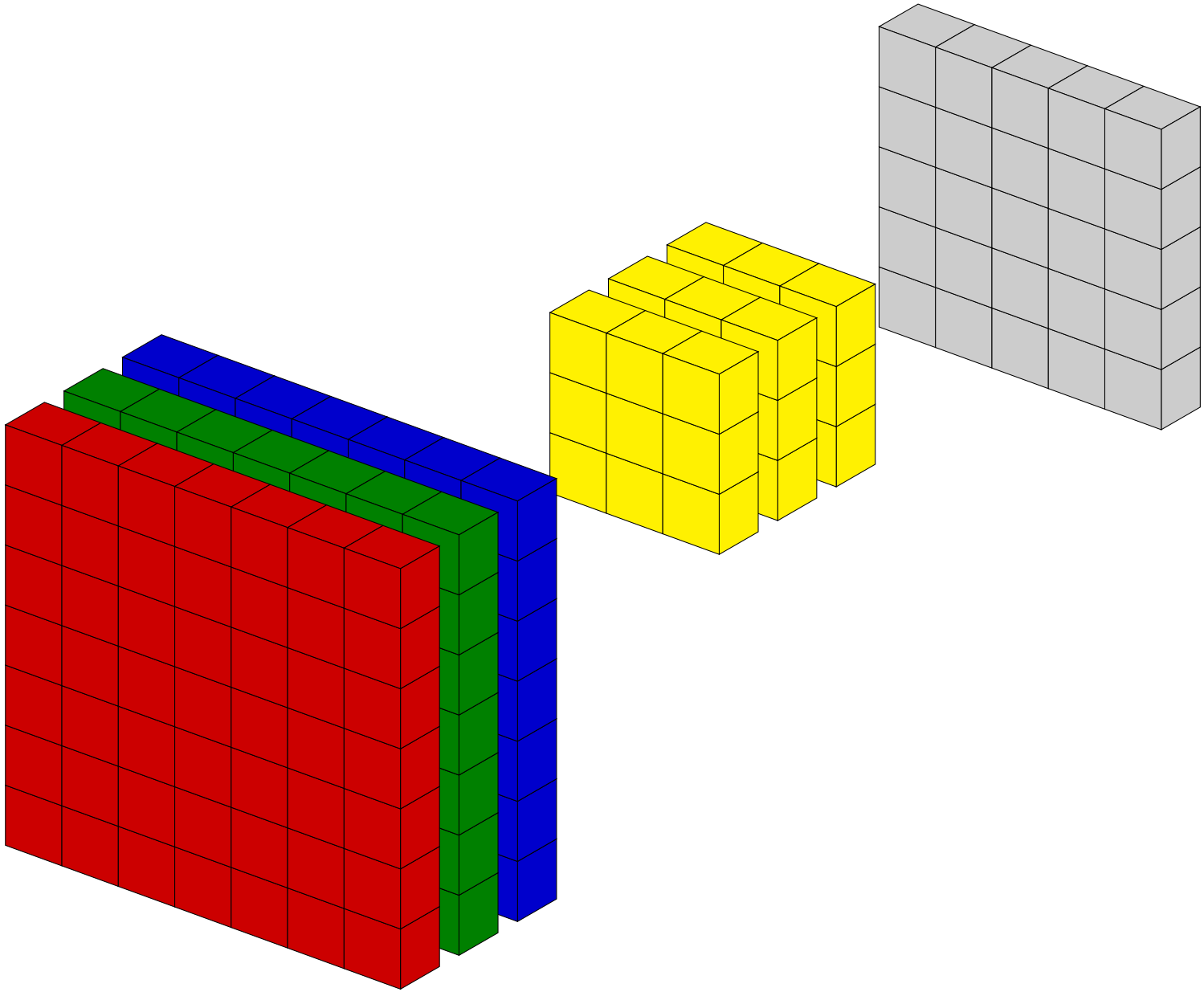
With stride


Input
Output
1 2 3 4 5 6
7 8 9 10 11 12
13 14 15 16 17 18
19 20 21 22 23 24
25 26 27 28 29 30
31 32 33 34 35 36
Kernel
1 2
3 4

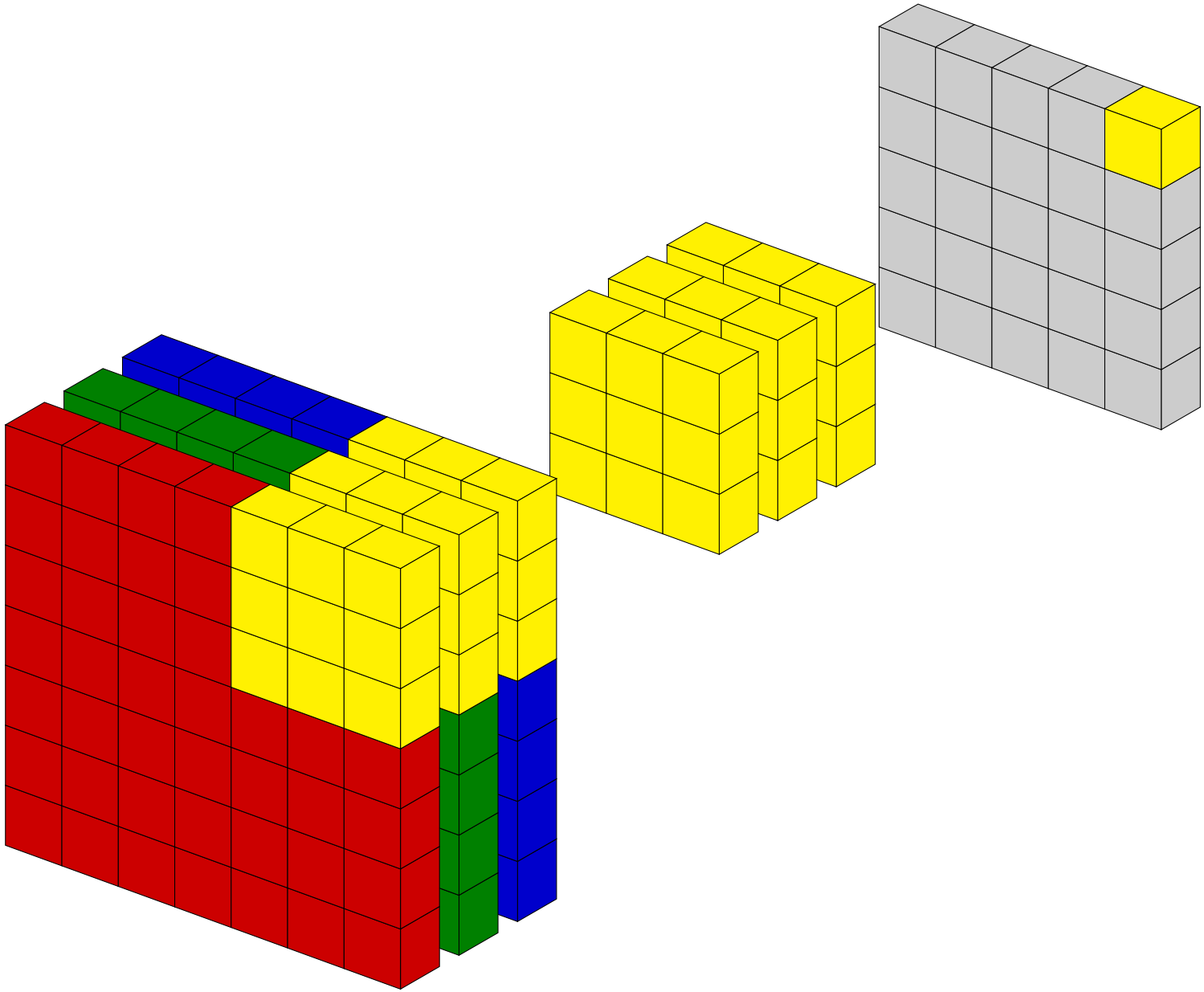
With stride


Input
Output
1 2 3 4 5 6
7 8 9 10 11 12
13 14 15 16 17 18
19 20 21 22 23 24
25 26 27 28 29 30
31 32 33 34 35 36
Kernel
1 2
3 4
Flattened input
1
2
3
4
Sparse matrix
Flattened output
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36

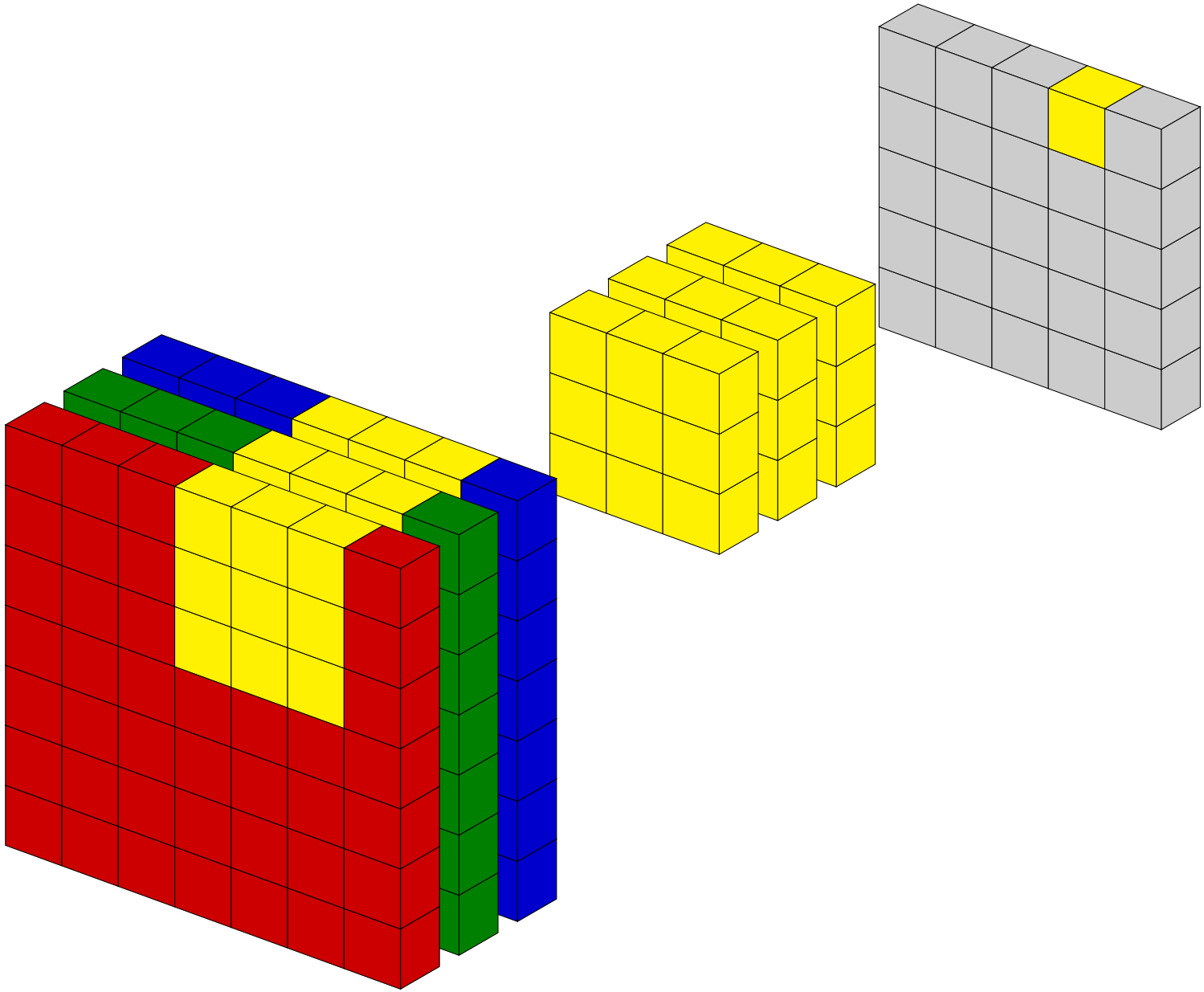
Multi-channel convolution



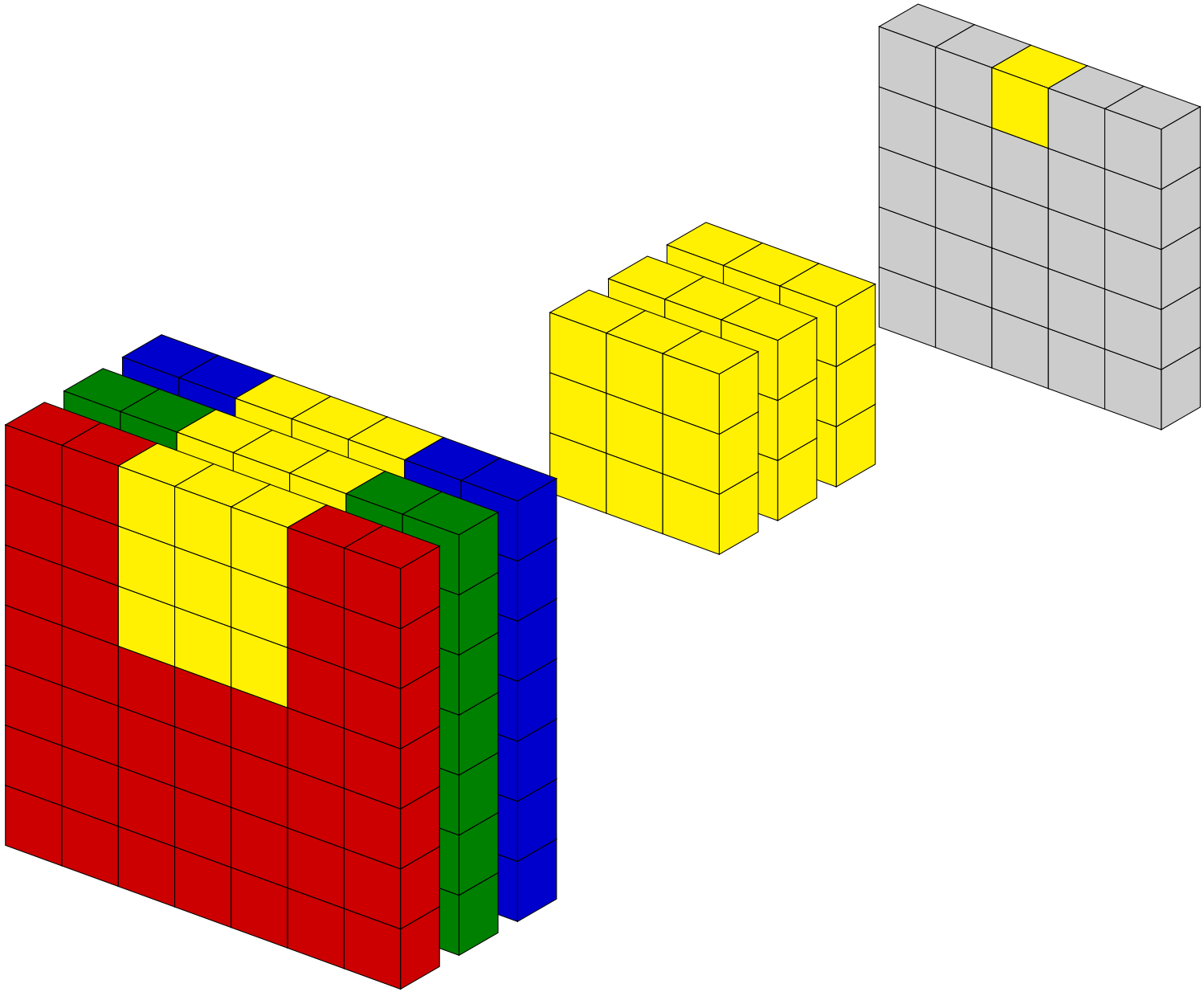
Multi-channel convolution



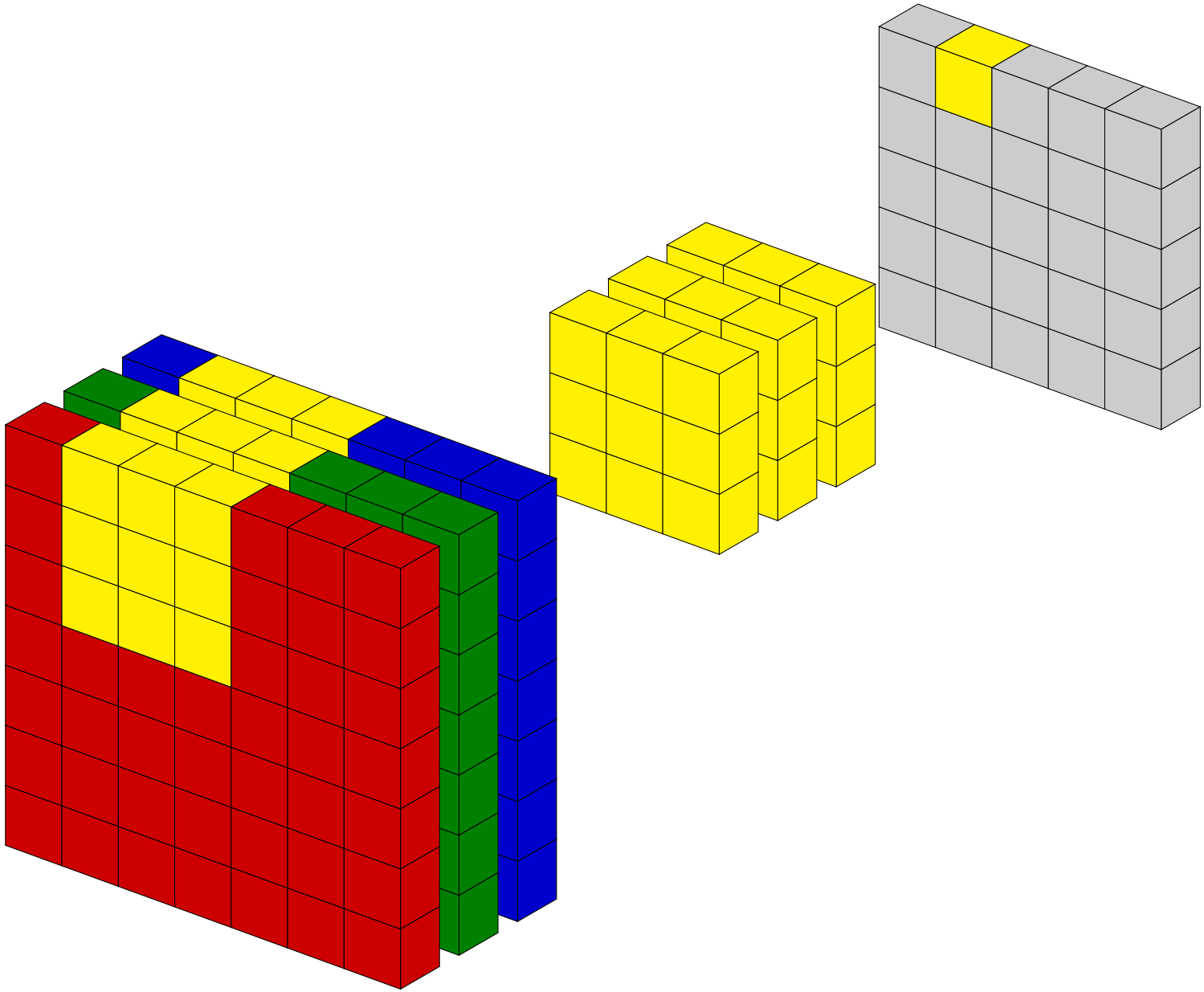
Multi-channel convolution



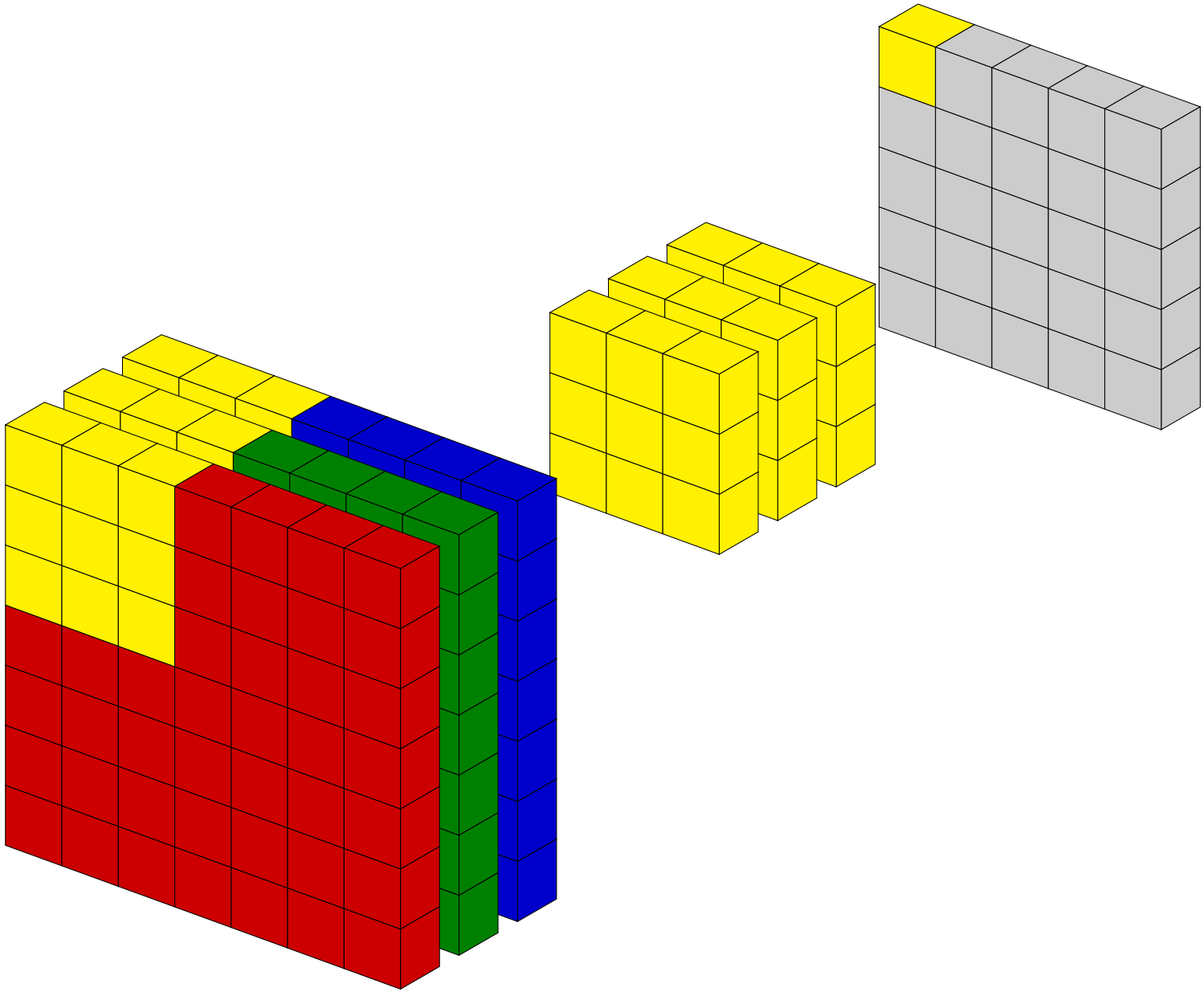
Multi-channel convolution



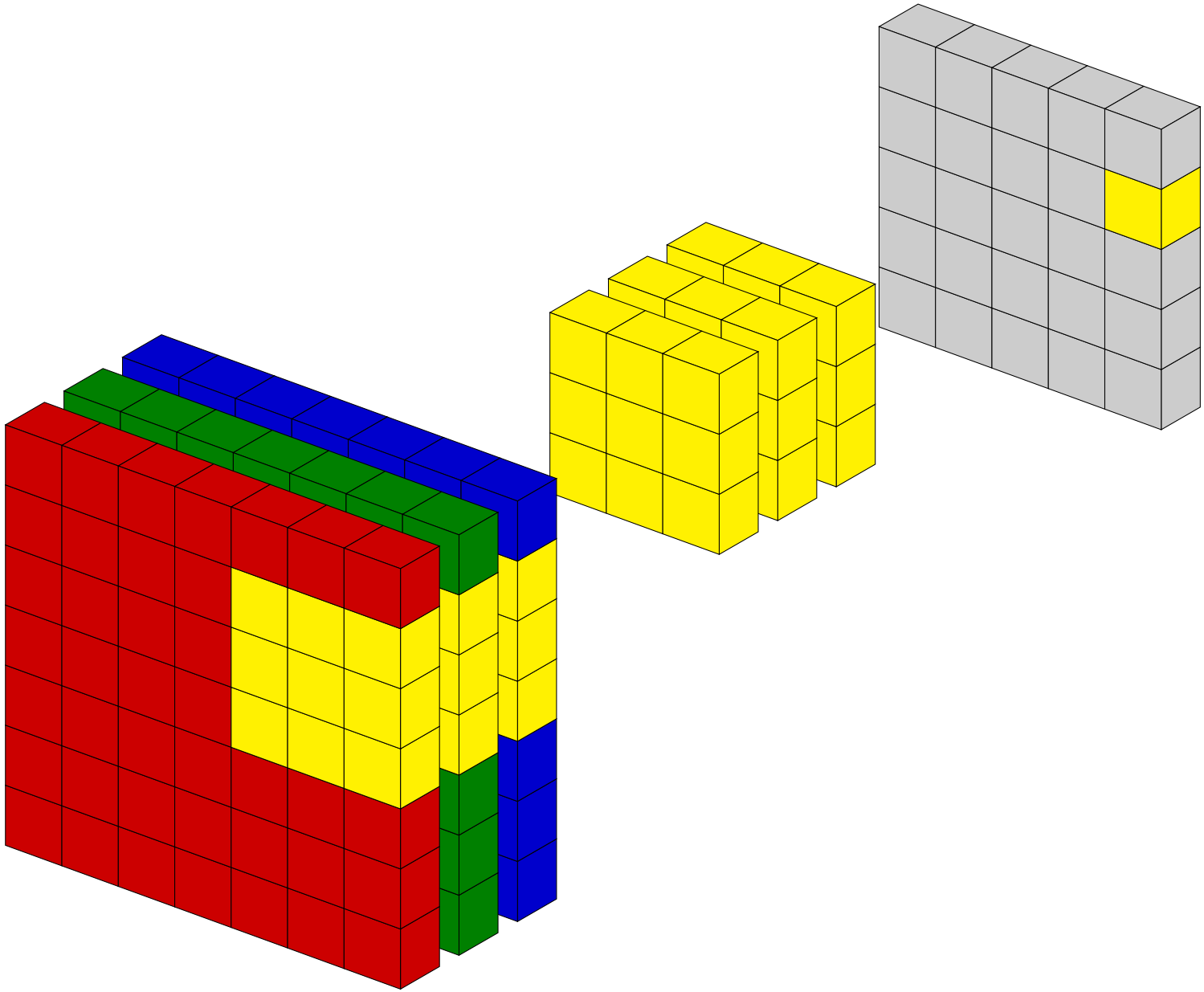
Multi-channel convolution



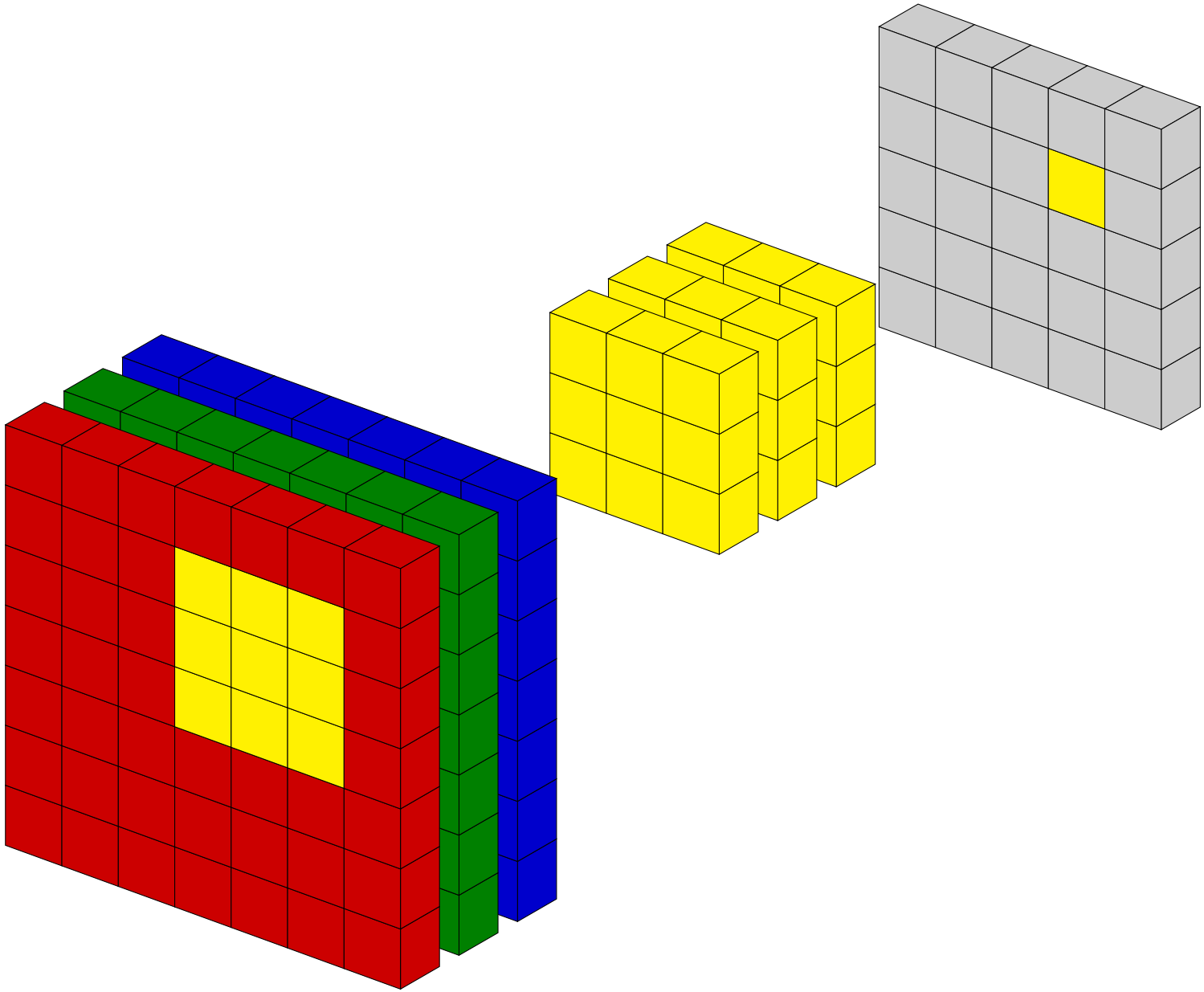
Multi-channel convolution



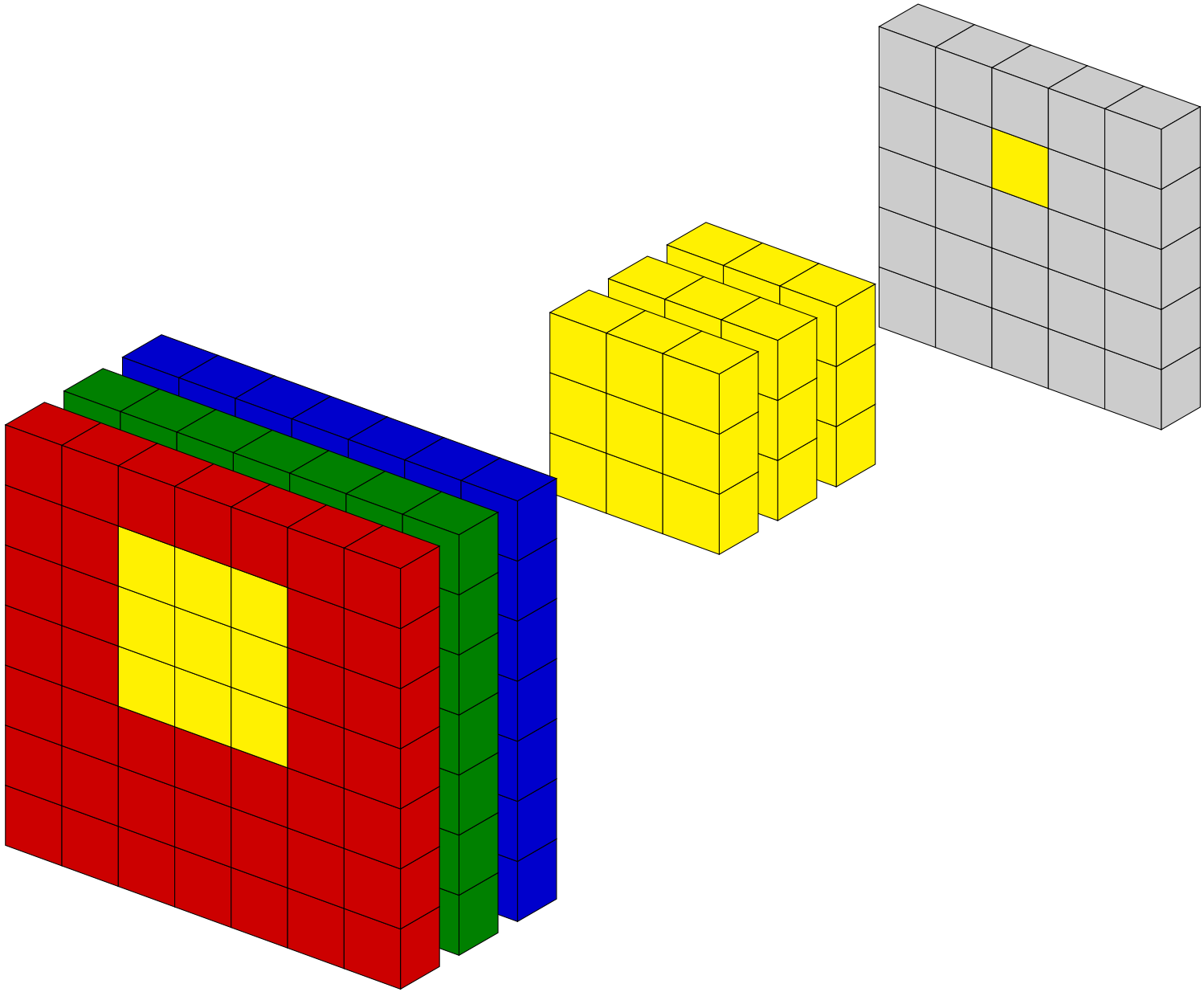
Multi-channel convolution



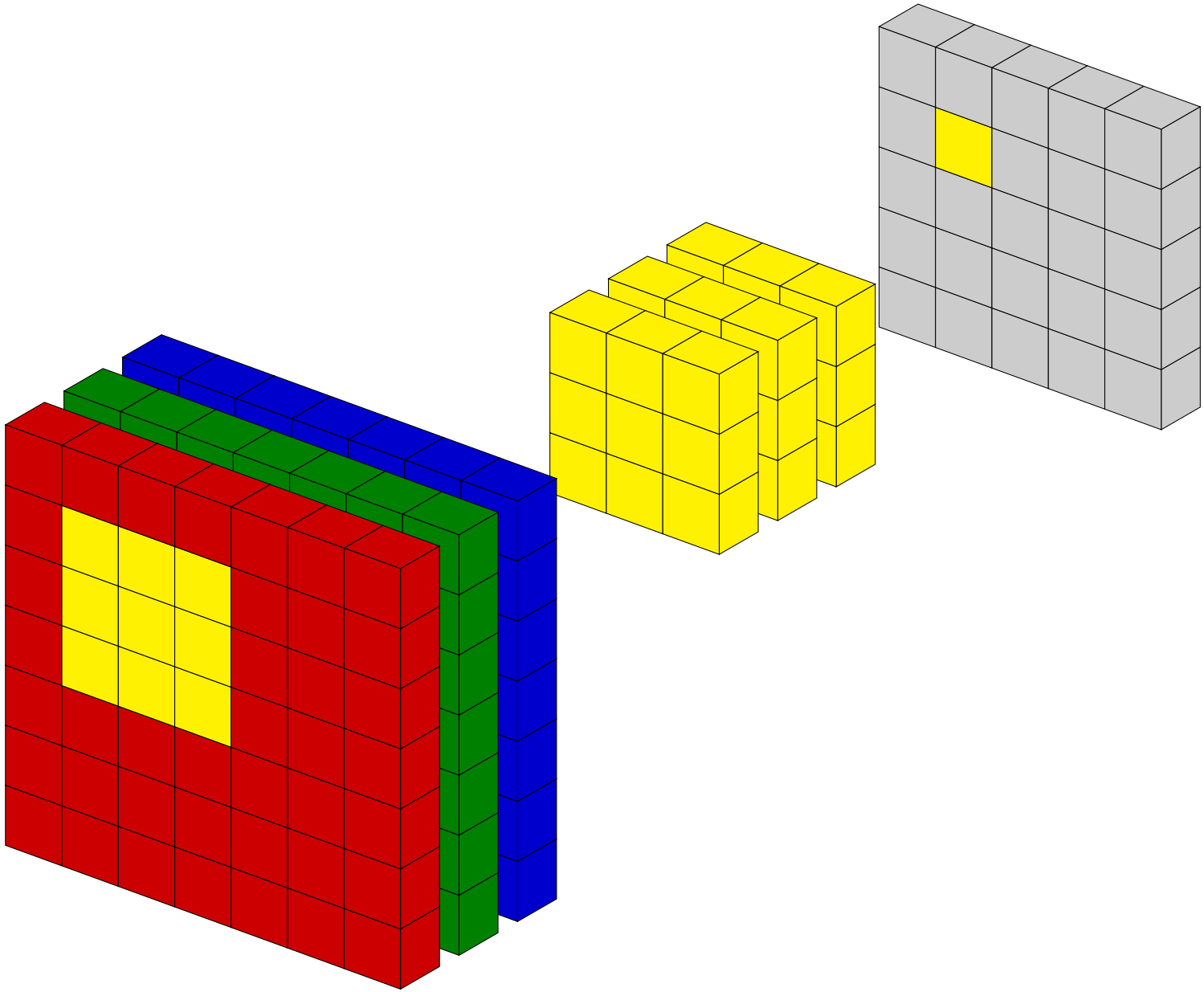
Multi-channel convolution



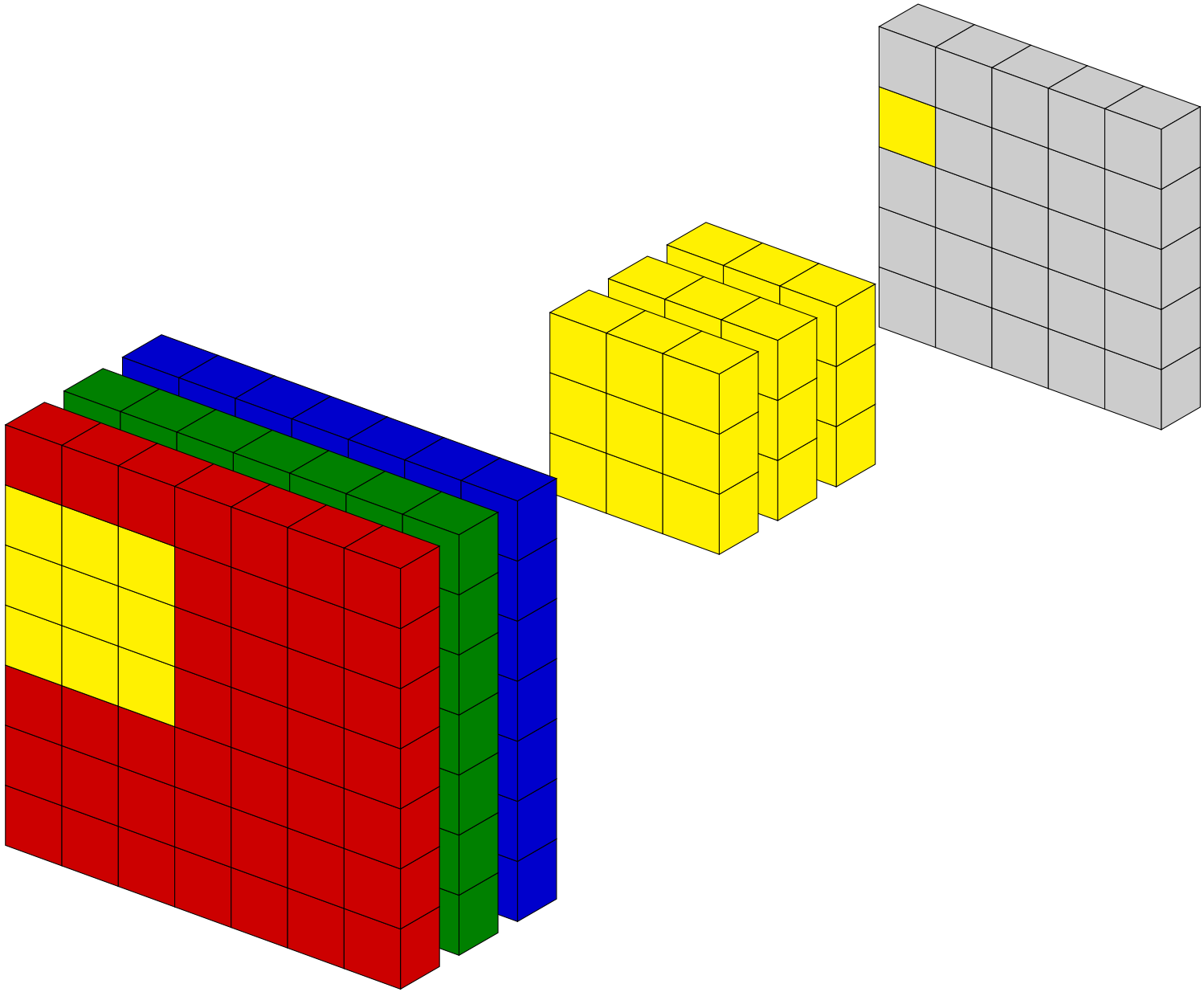
Multi-channel convolution



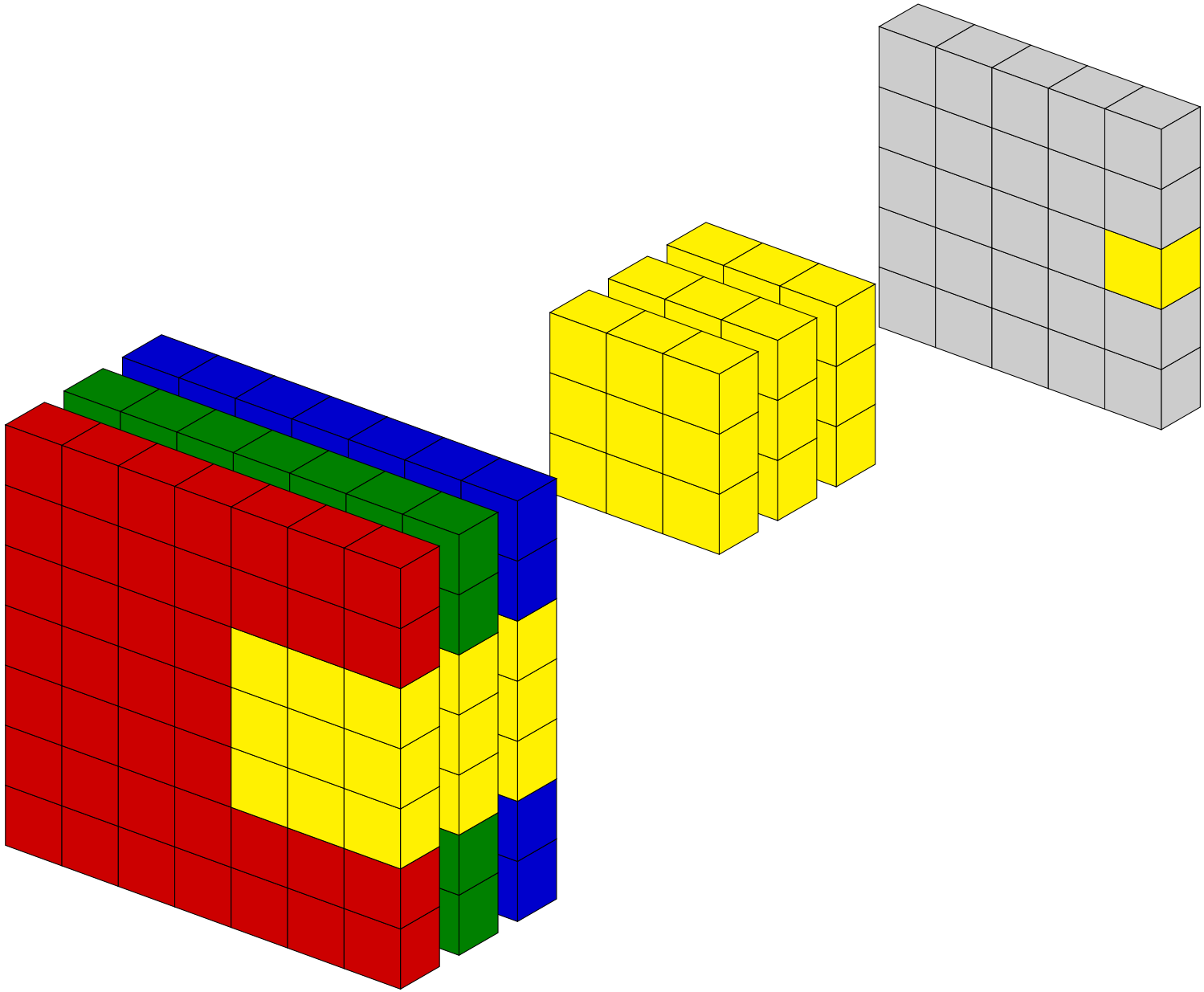
Multi-channel convolution



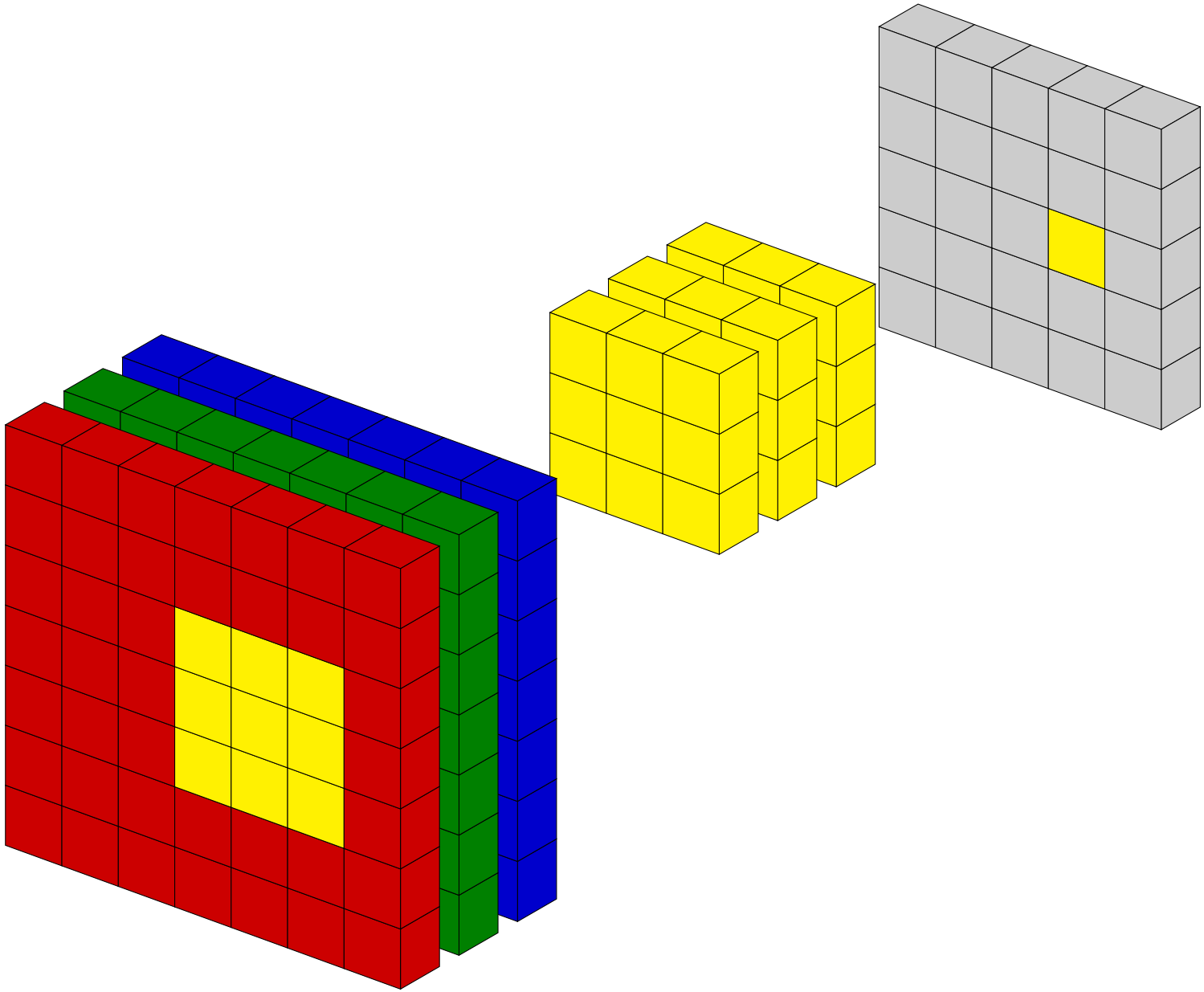
Multi-channel convolution



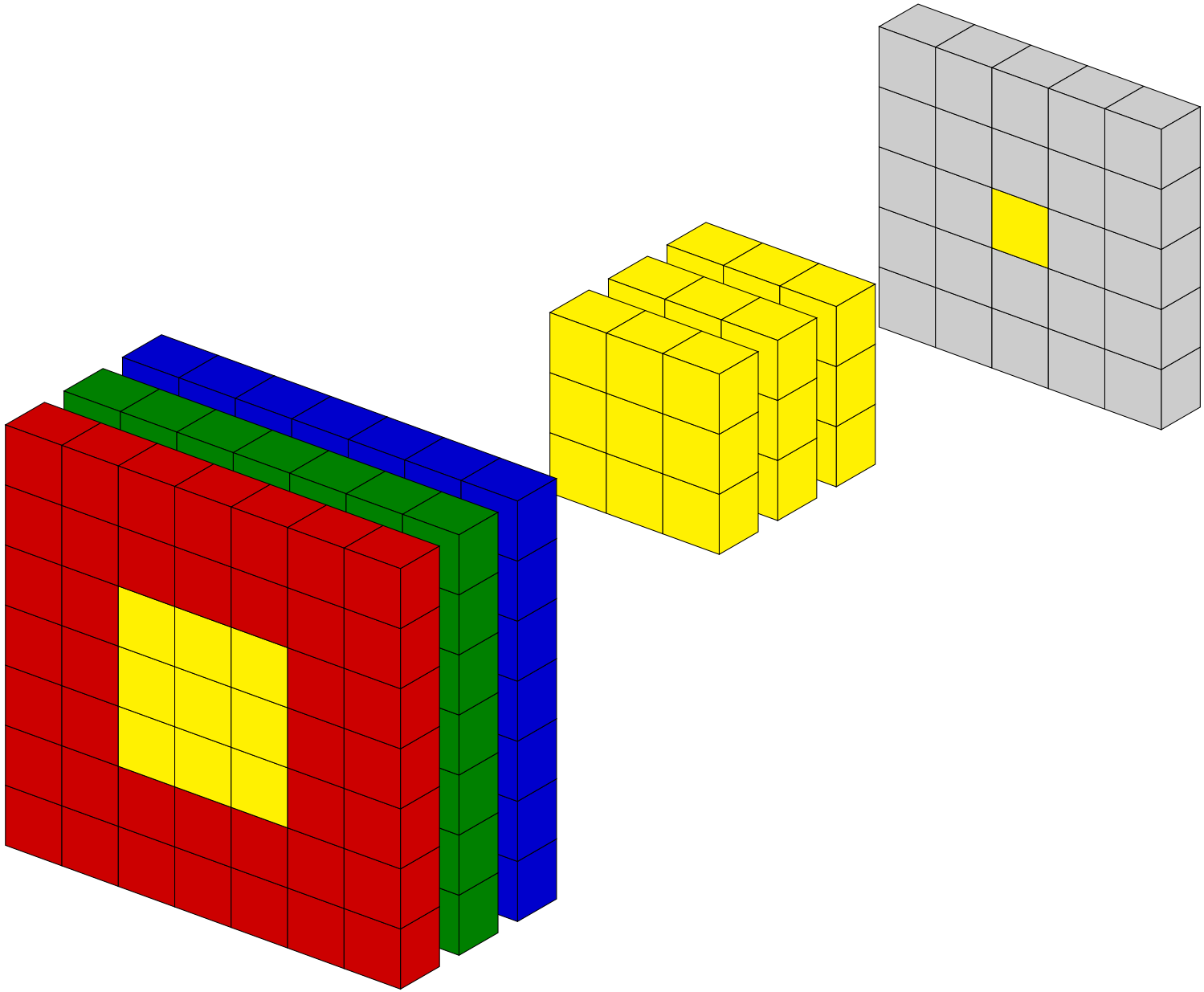
Multi-channel convolution



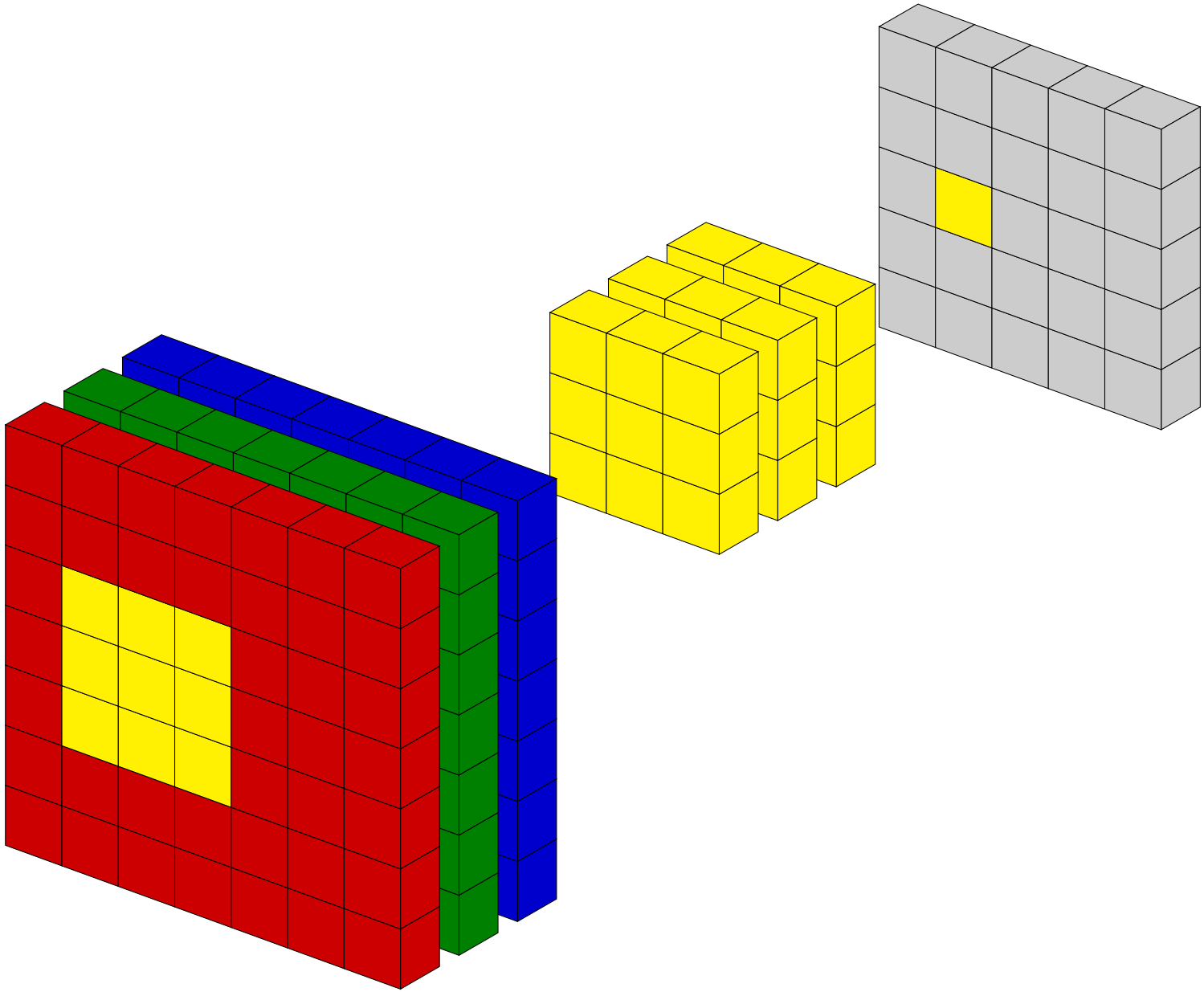
Multi-channel convolution



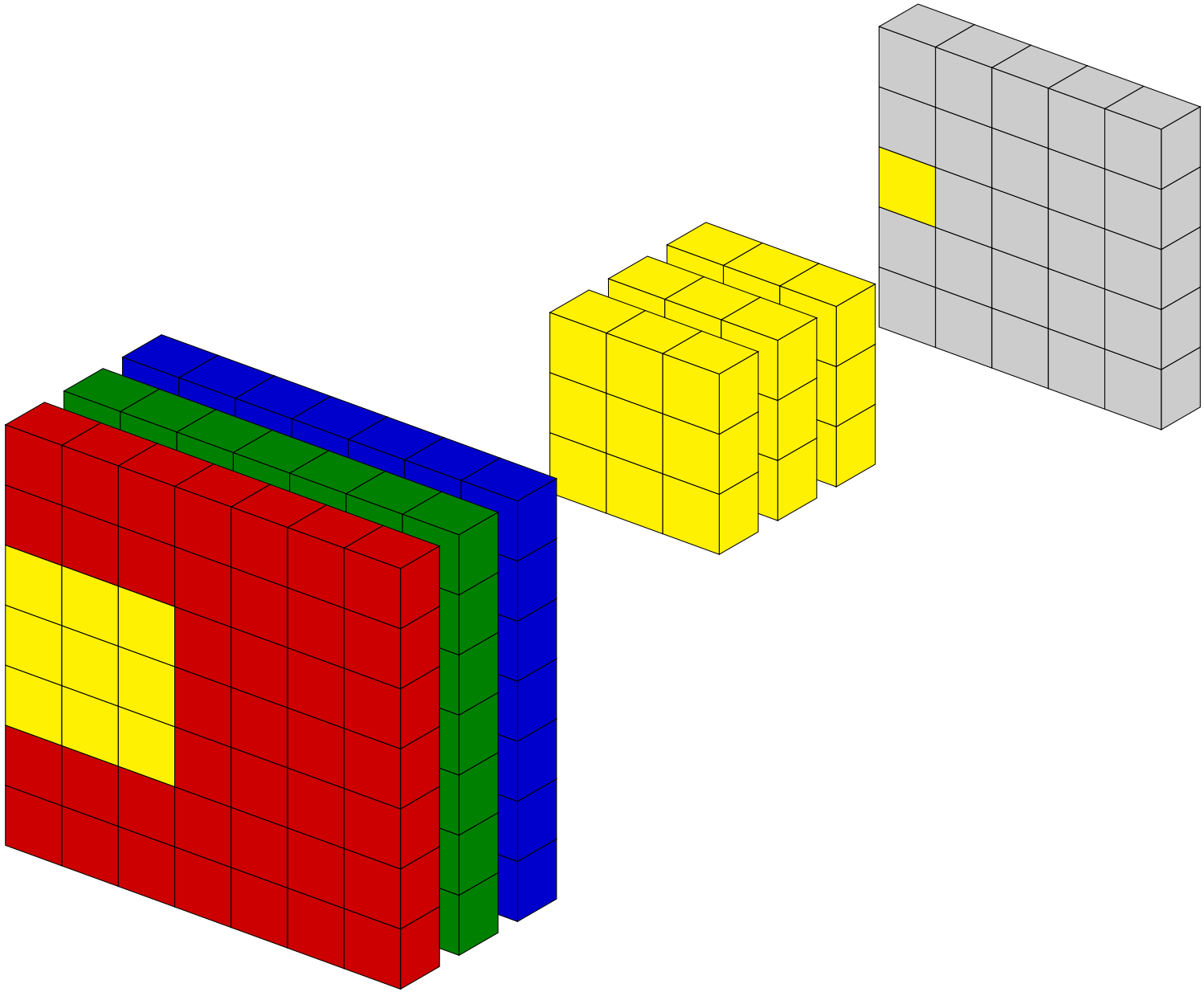
Multi-channel convolution



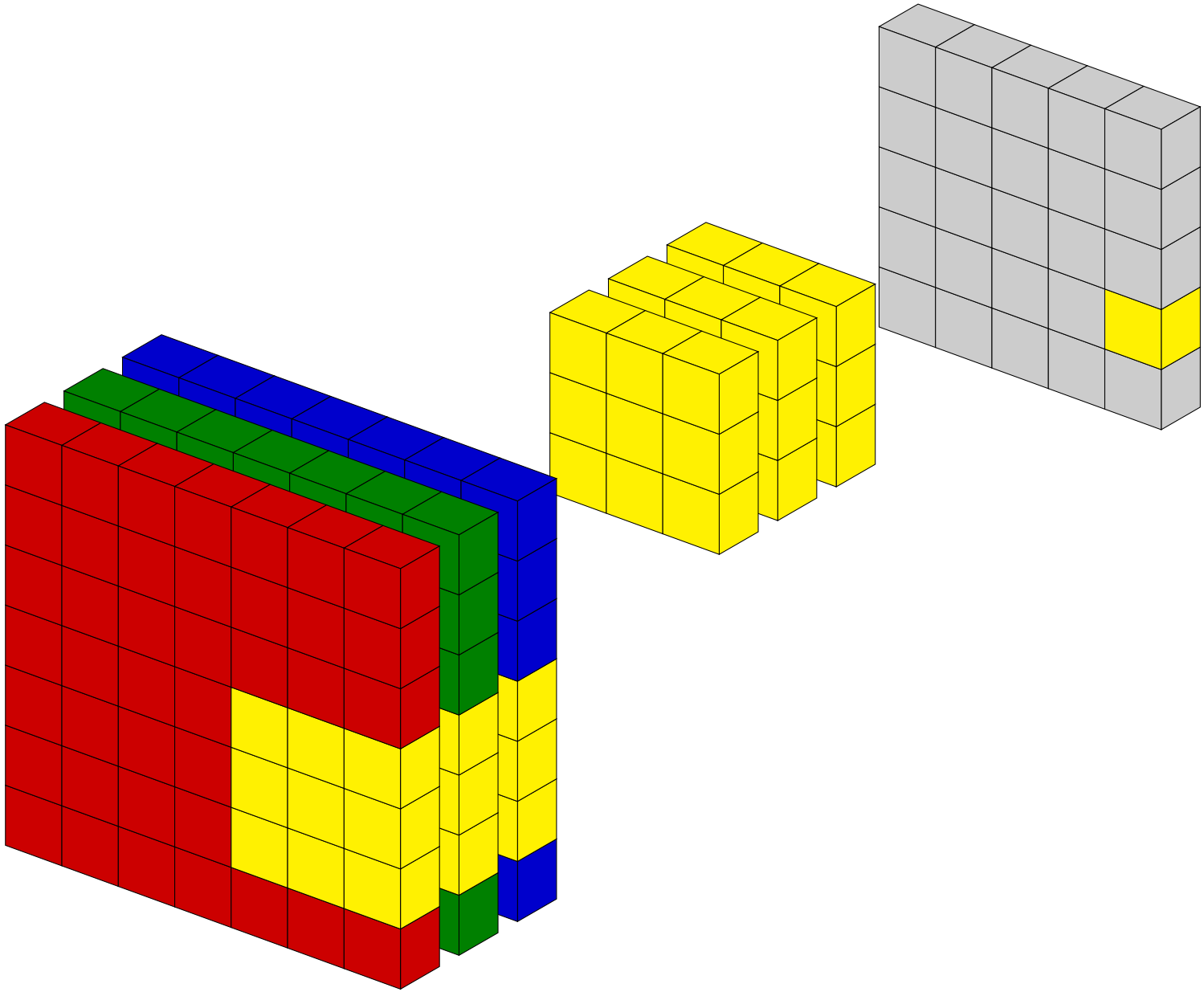
Multi-channel convolution



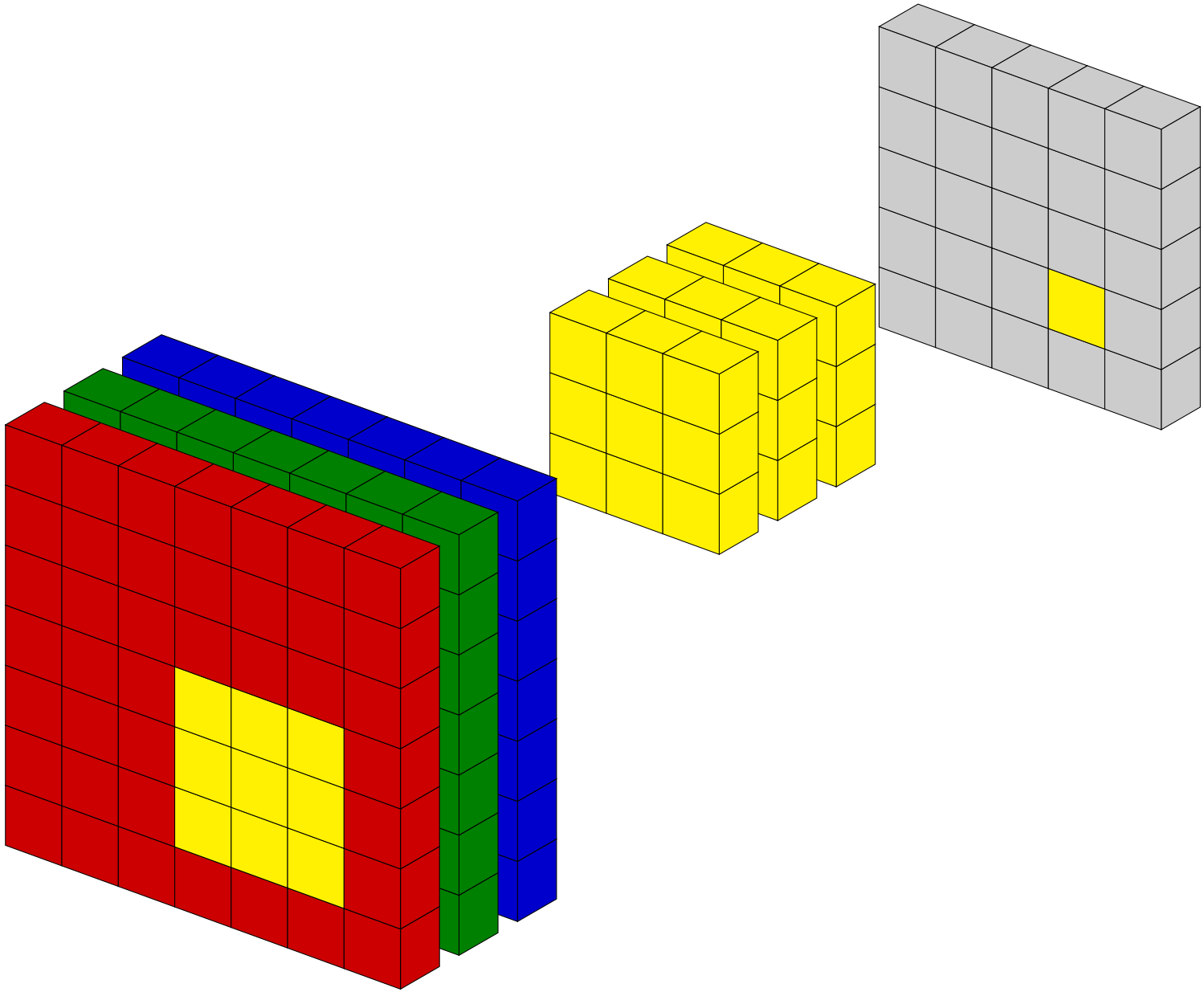
Multi-channel convolution



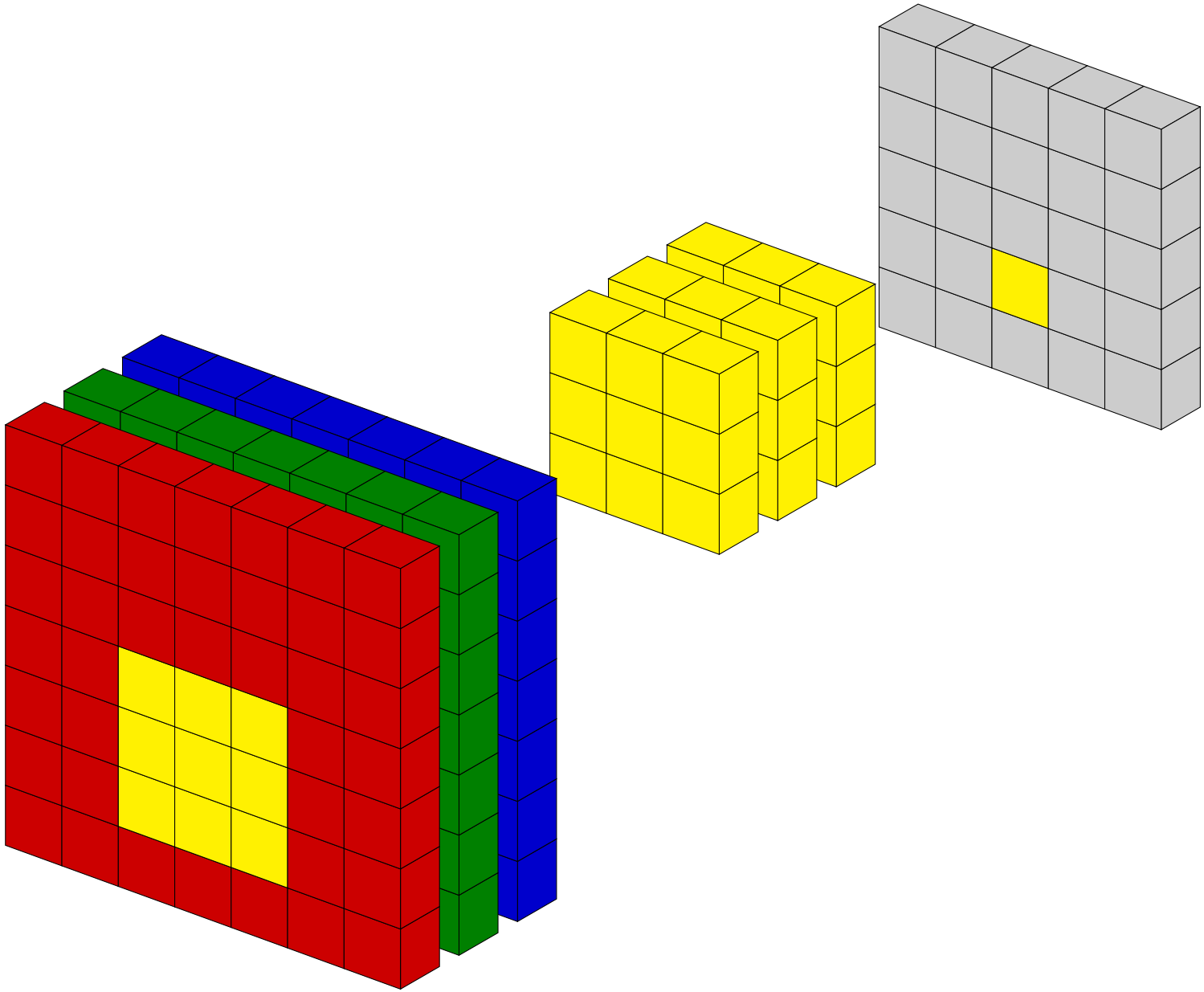
Multi-channel convolution



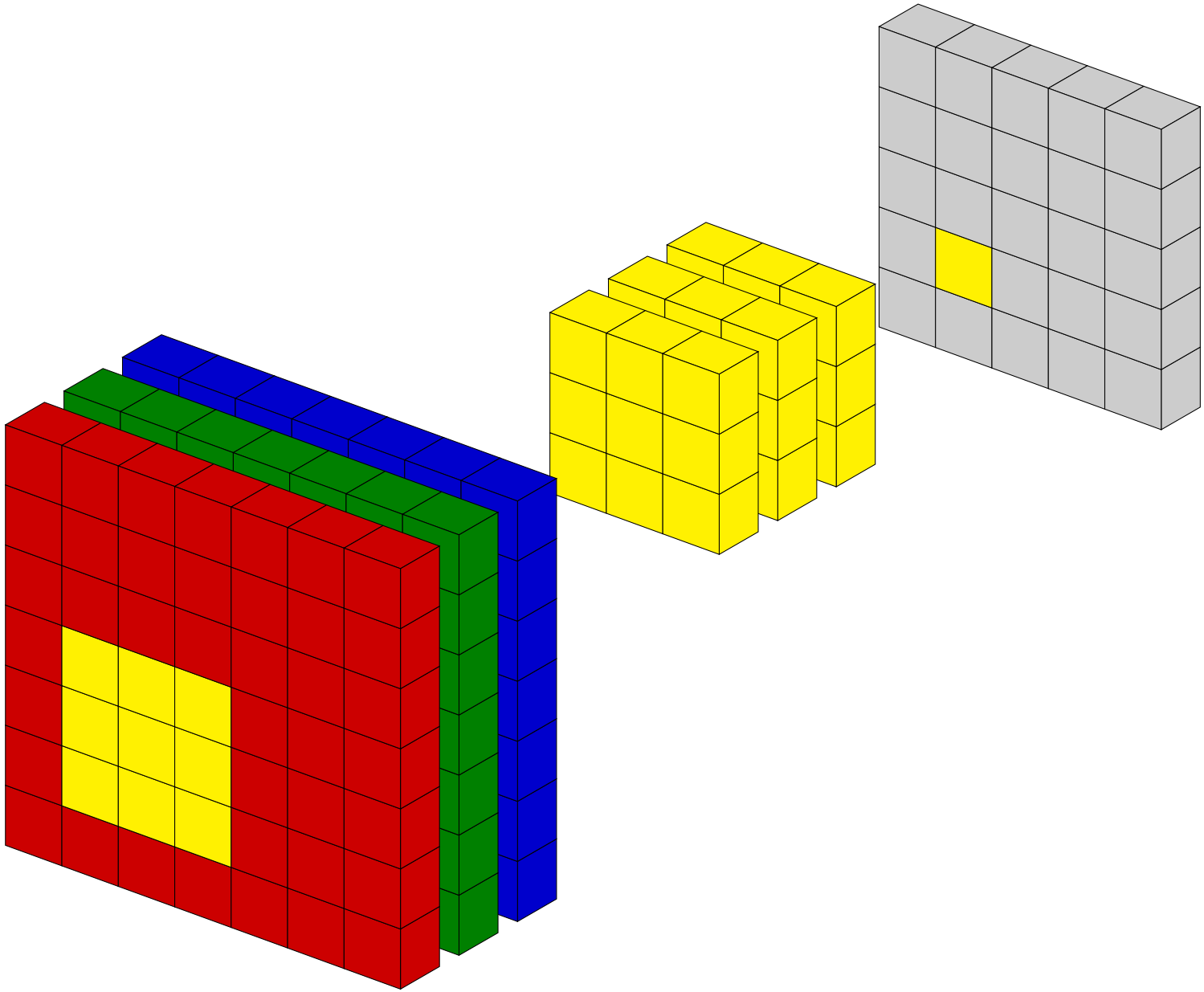
Multi-channel convolution



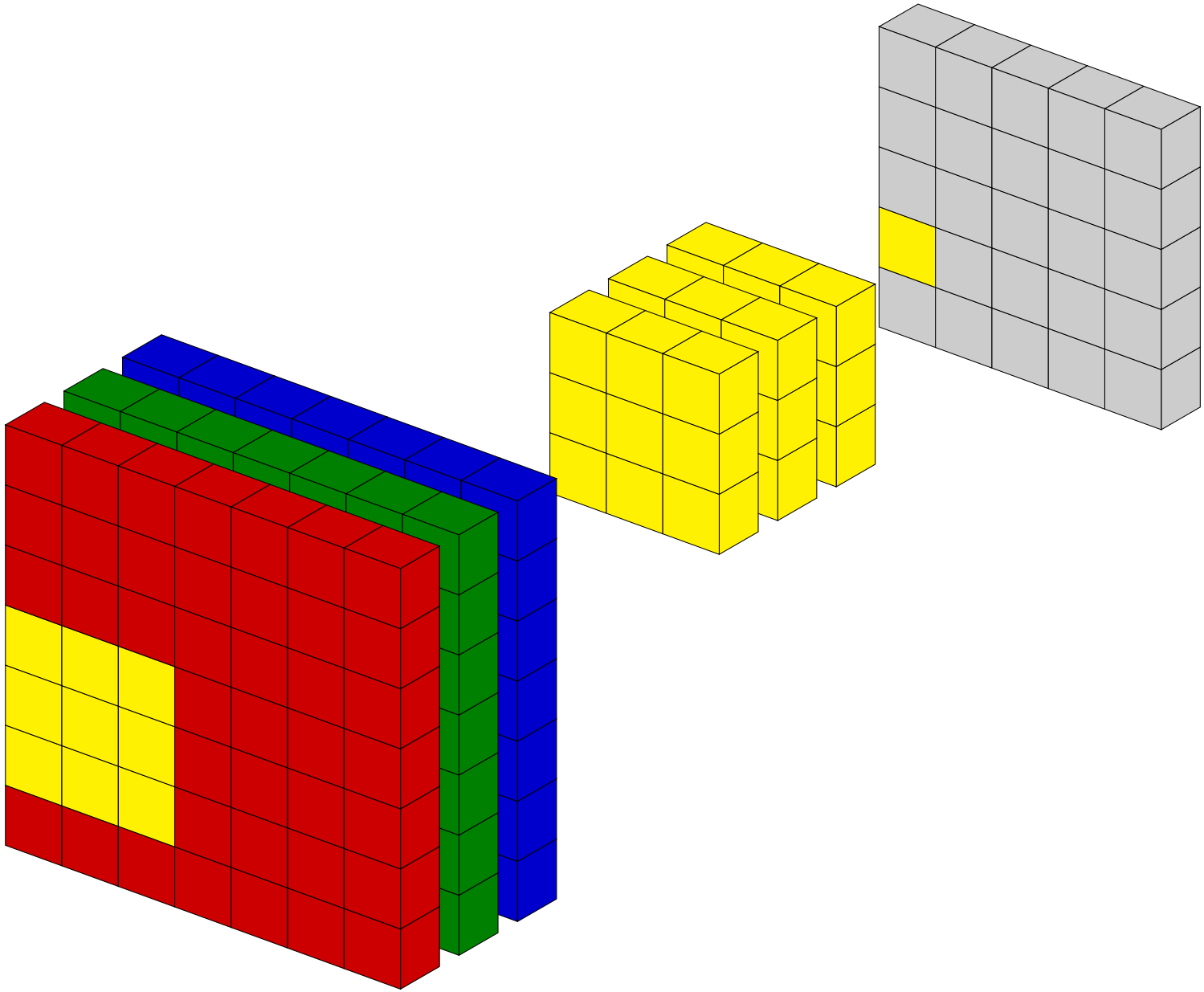
Multi-channel convolution



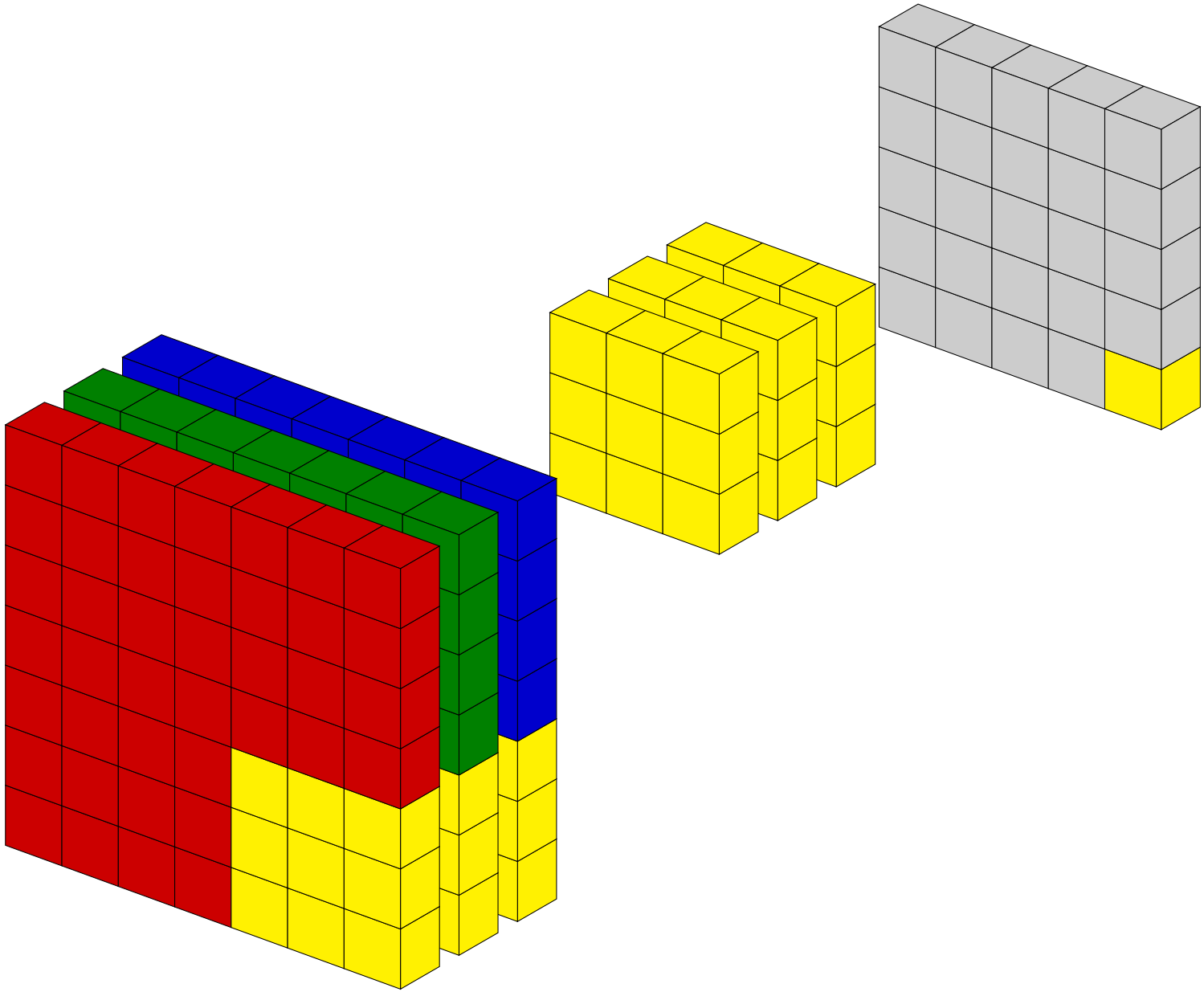
Multi-channel convolution



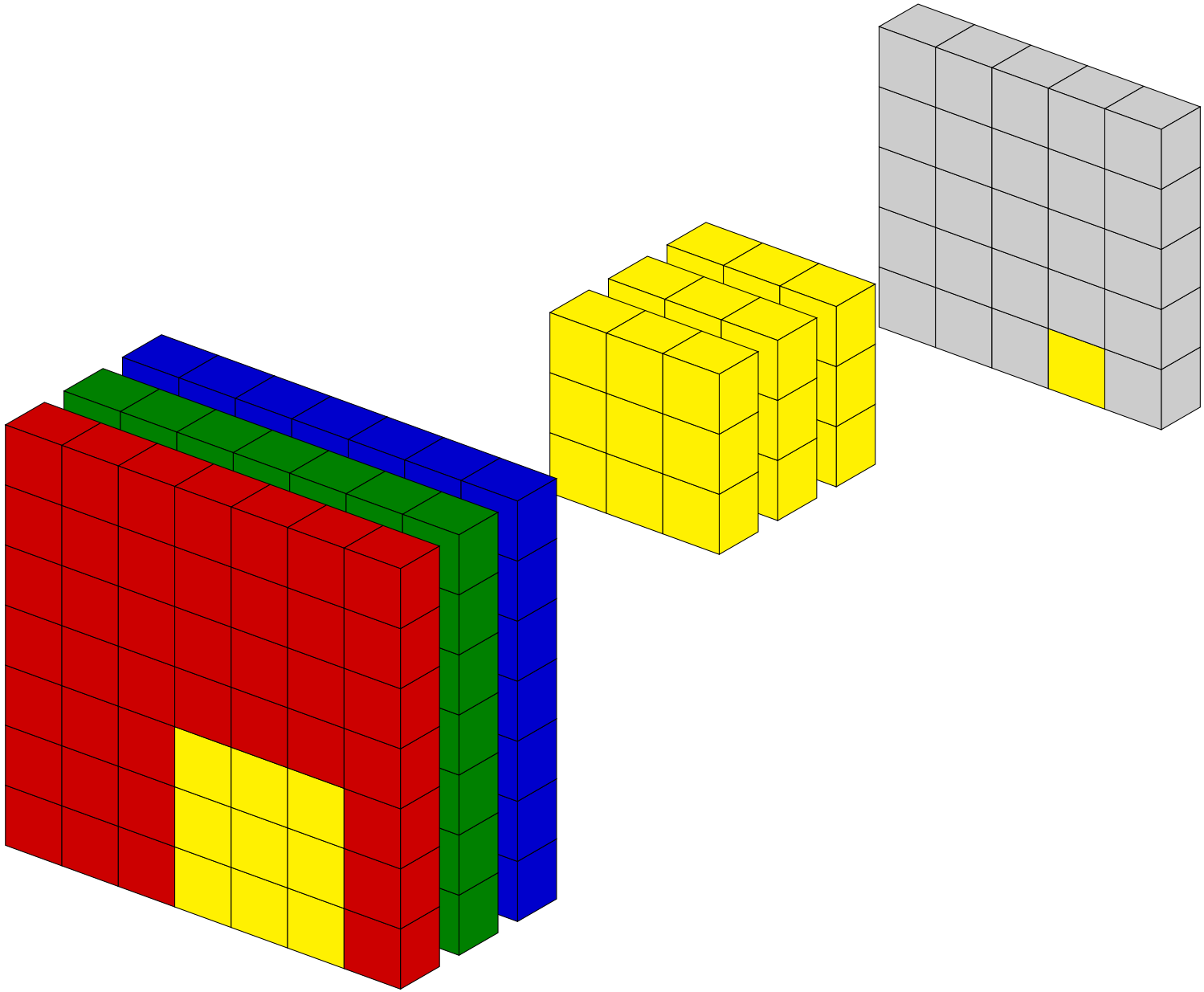
Multi-channel convolution



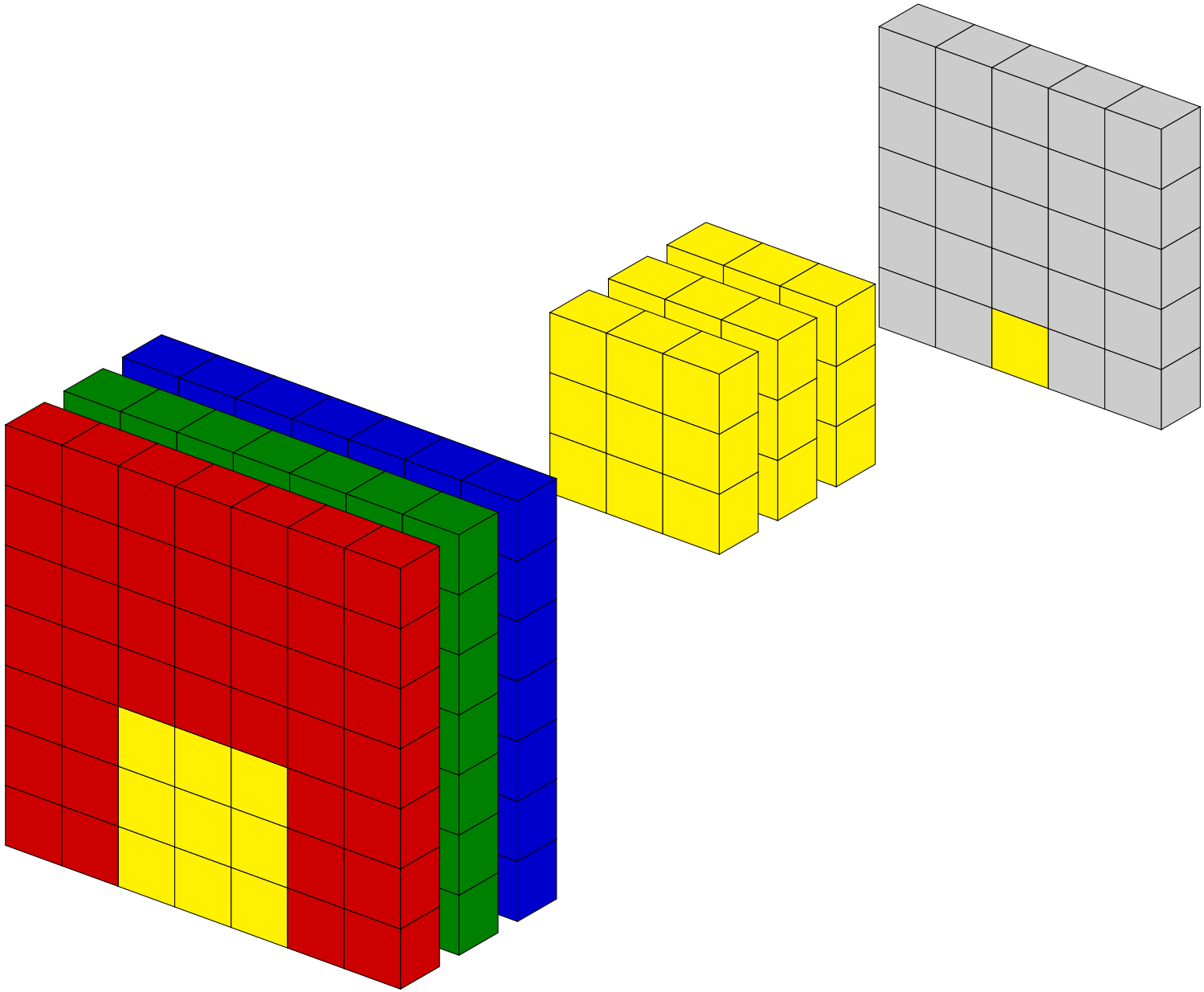
Multi-channel convolution



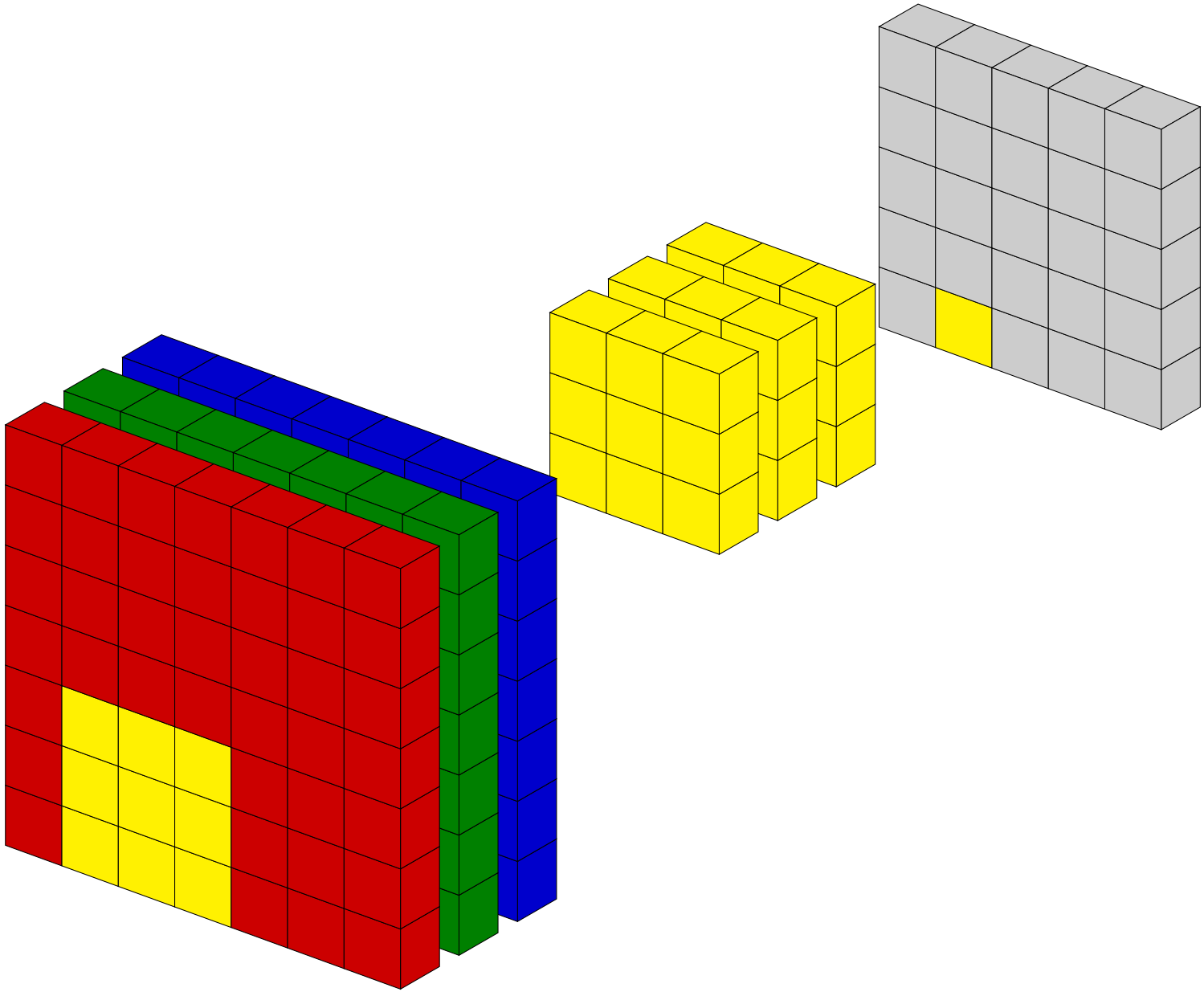
Multi-channel convolution



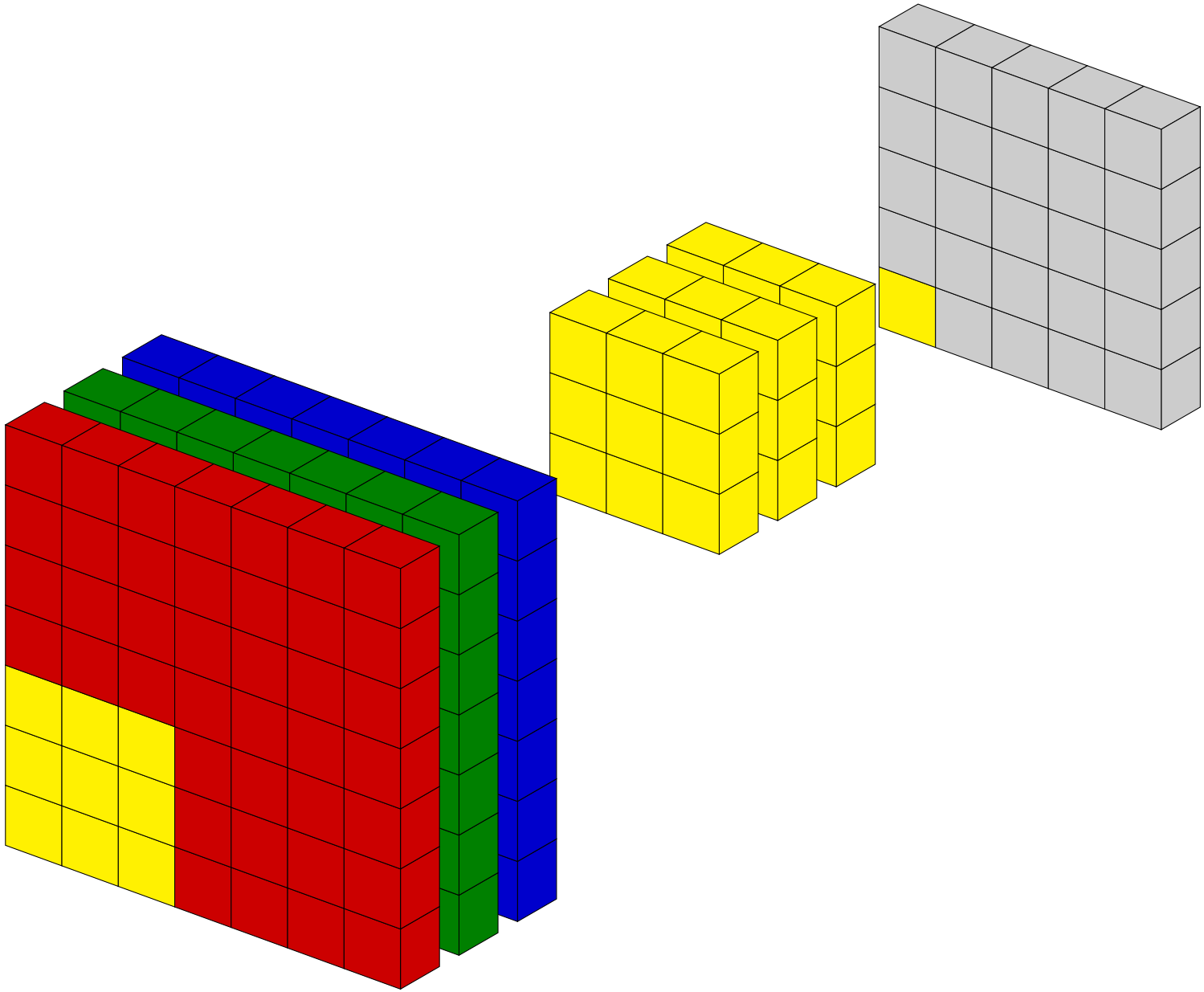
Multi-channel convolution



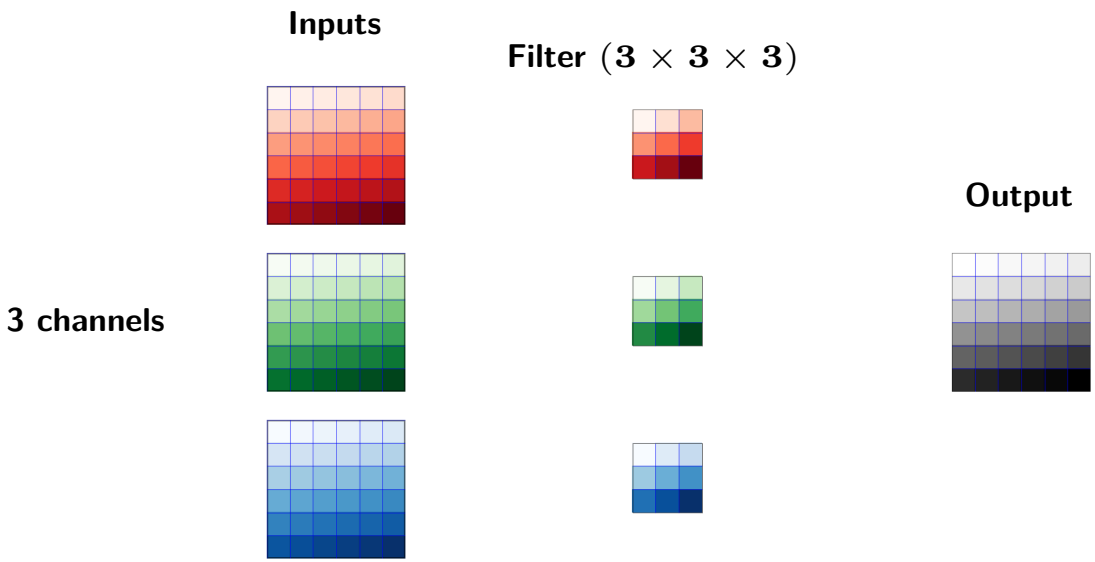
Multi-channel convolution



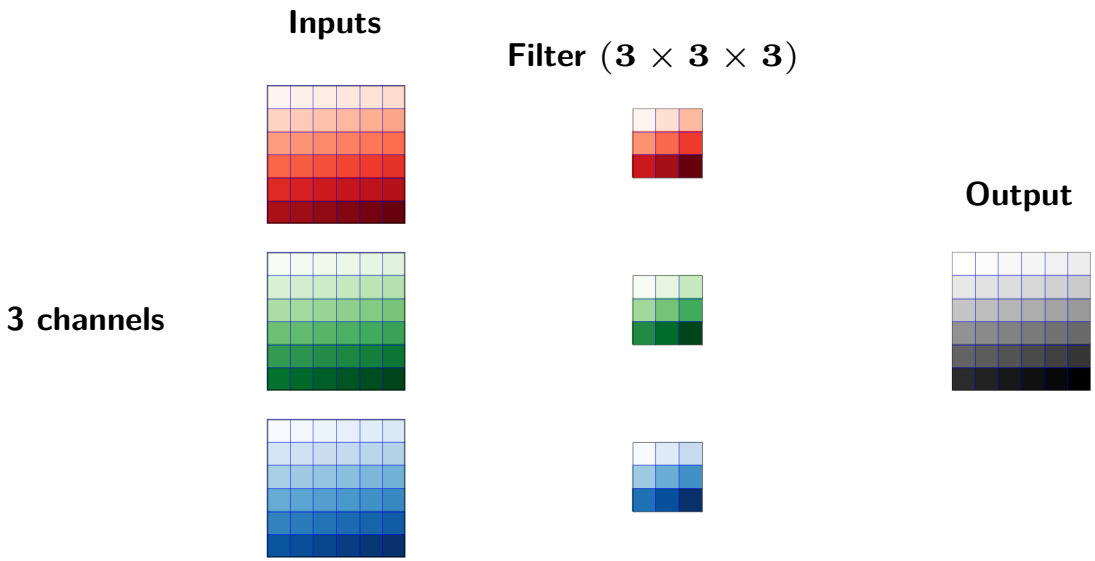
Multi-channel convolution



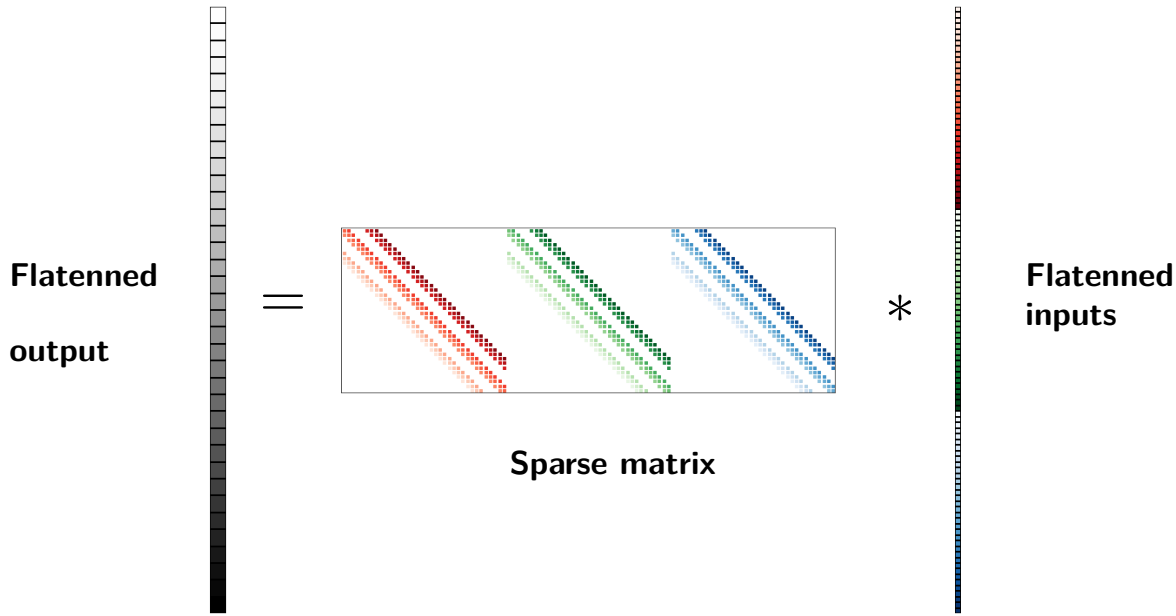
Multi-channel convolution



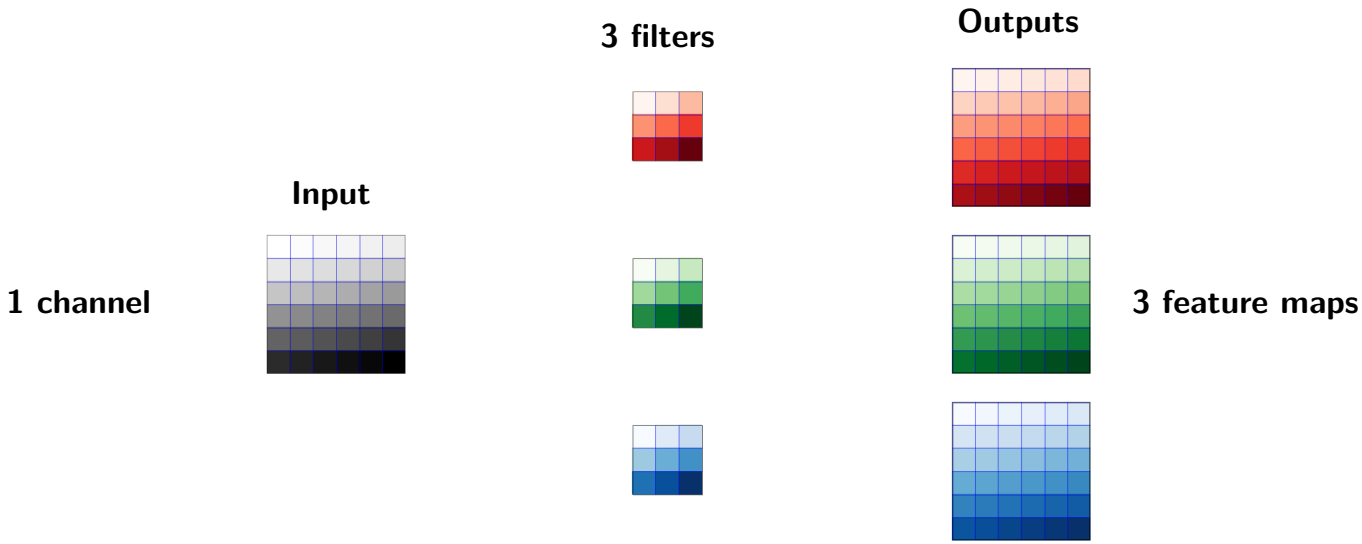
Multi-channel convolution



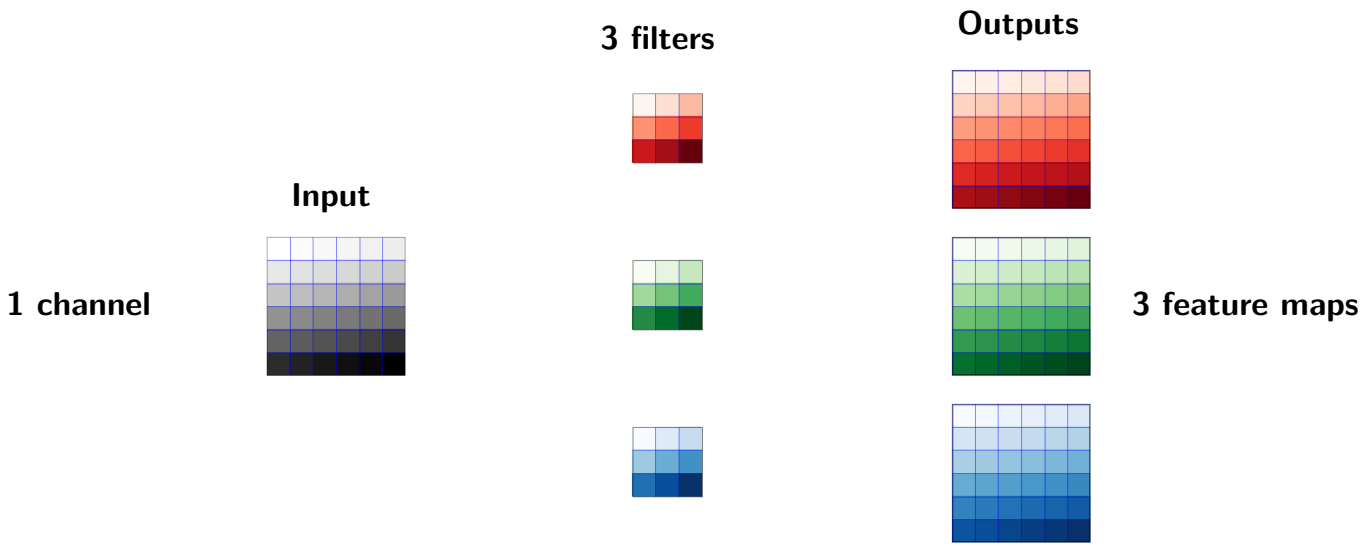
Matricial form:



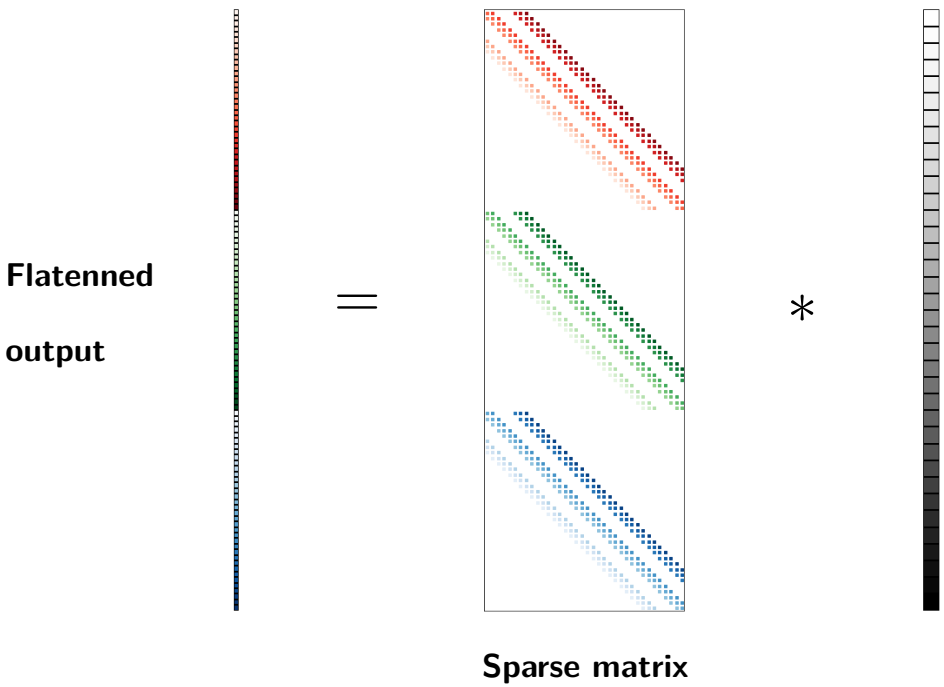
Multi-filter convolution



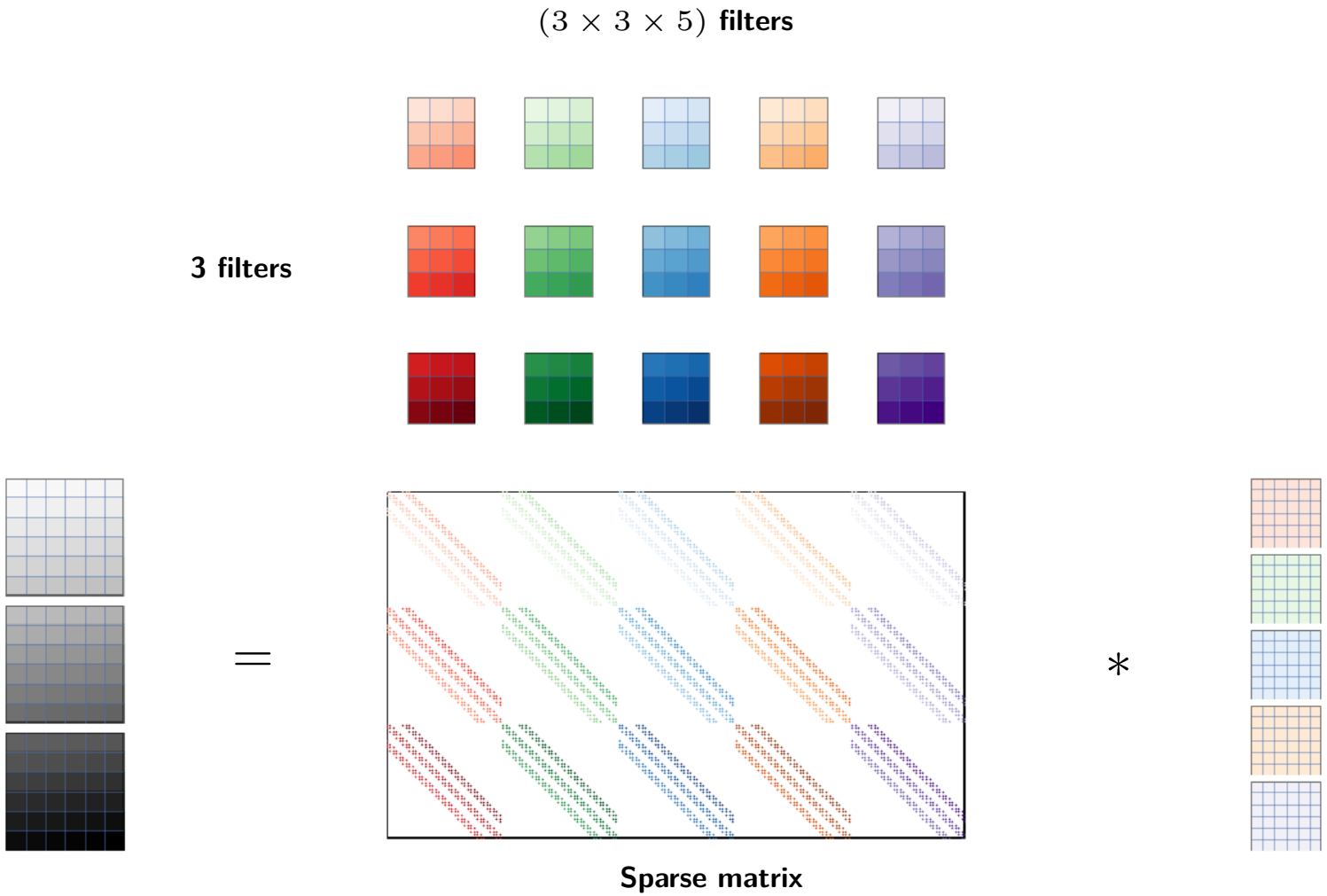
Multi-filter convolution



Matricial form:



Multi-filter for multi-channels



Some numbers

Number of parameters of a convolutional layer

$$N_p = c_i \times c_o \times k_1 \times k_2$$

where

- c_i is the number of input channels
- c_o is the number of output channels
- $k_1 \times k_2$ are the filter dimensions

Size of the output image $n_1^{(o)} \times n_2^{(o)}$

$$n_j^{(o)} = \text{floor}\left[\frac{n_j^{(i)} + 2p_j - (k_j - 1) - 1}{s_j} + 1\right]$$

where

- $n_1^{(i)} \times n_2^{(i)}$ are the input dimensions
- $p_1 \times p_2$ are the padding dimensions
- $s_1 \times s_2$ are the stride dimensions

One line of code

```
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride = 1, padding = 0,...)
```

Non-linearities

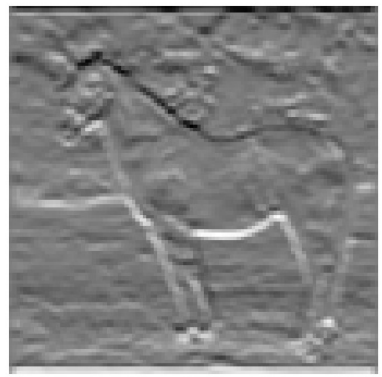
Activation function

The activation function (ReLU or Leaky ReLU) is applied on each pixel after the convolution

Example on a "real" image

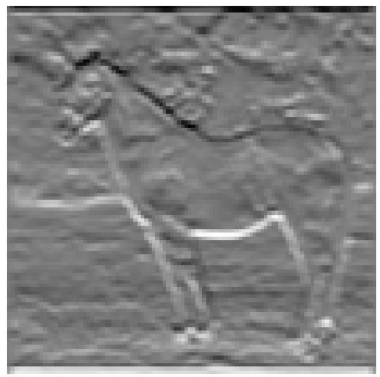
Example on a "real" image

After convolution

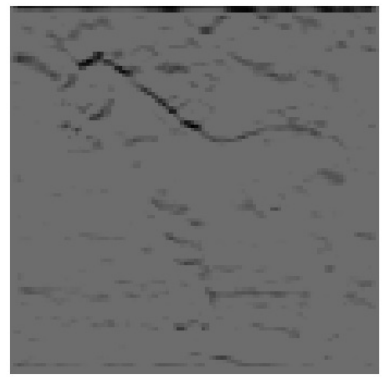


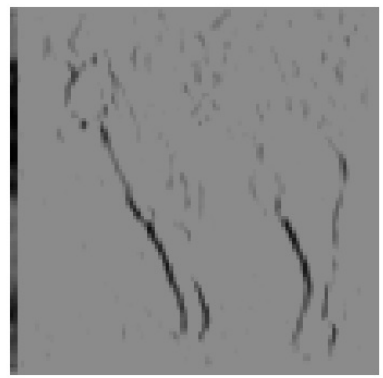
Example on a "real" image

After convolution



After ReLU

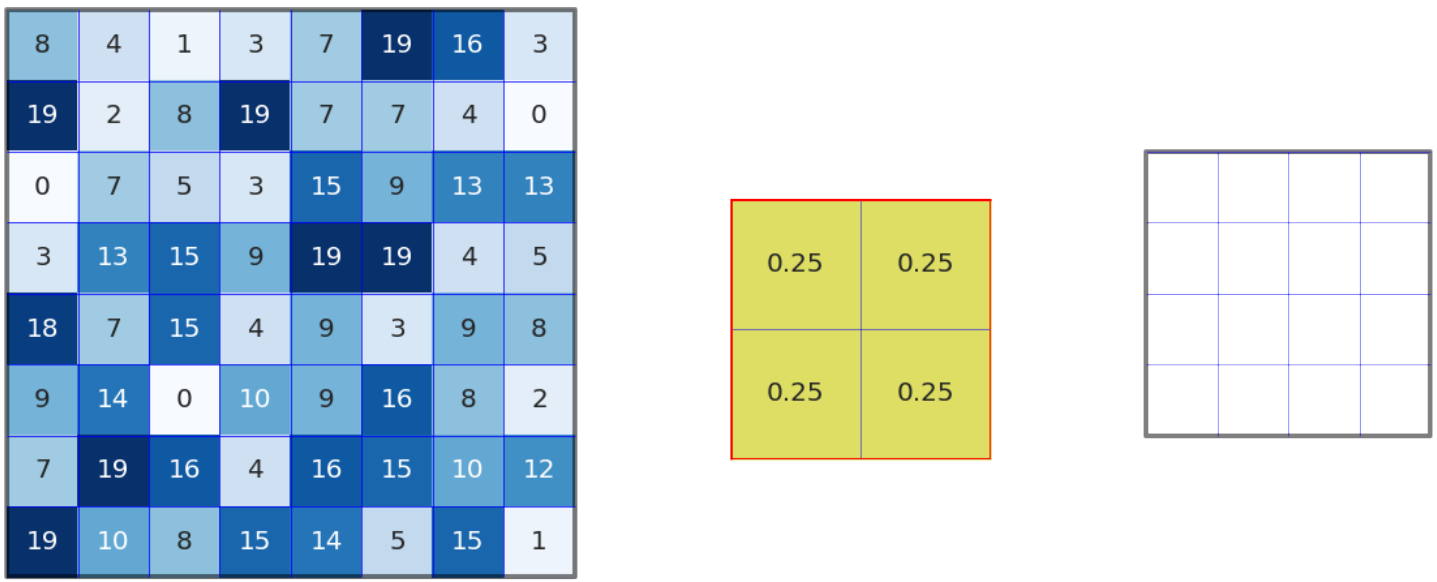




Pooling (or downscaling)

Average pooling

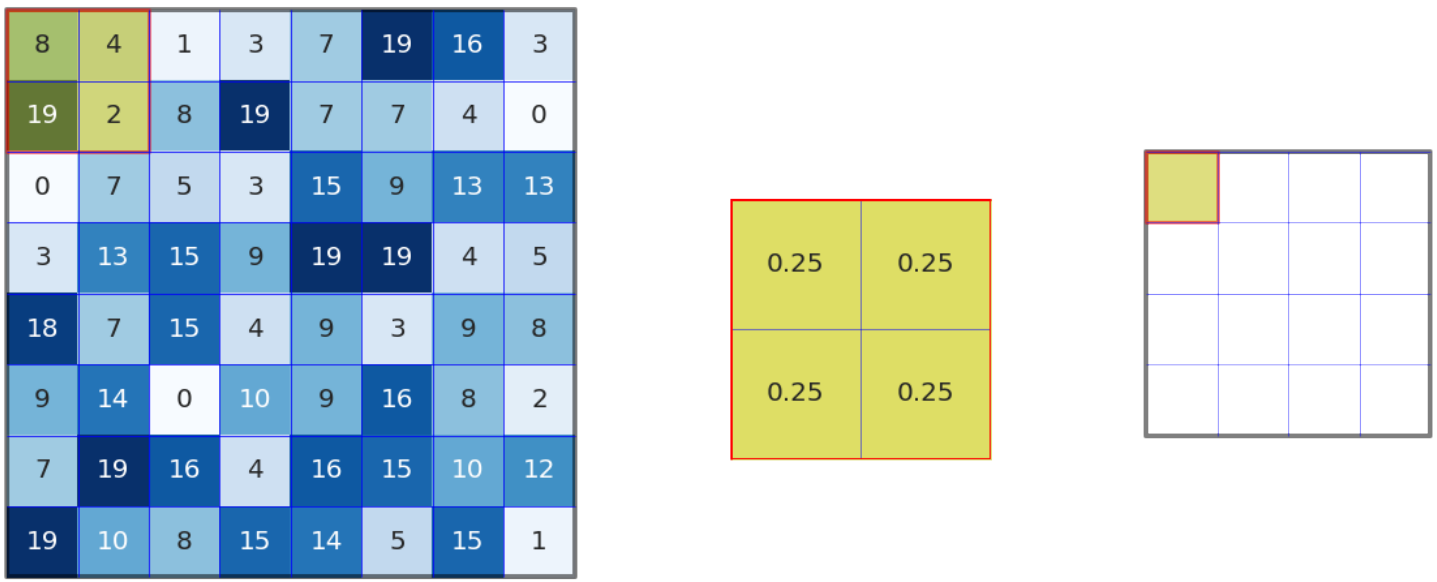
- Stride generally equal to kernel size
- No parameter to learn (no trainable parameters)



Pooling (or downscaling)

Average pooling

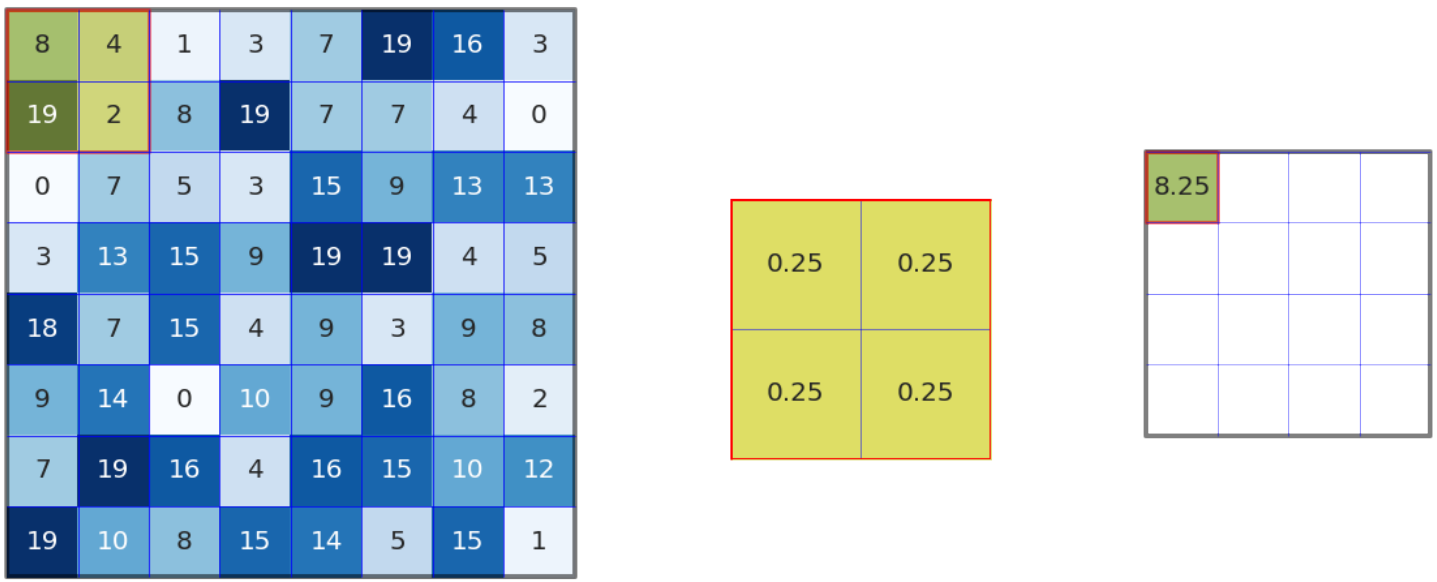
- Stride generally equal to kernel size
- No parameter to learn (no trainable parameters)



Pooling (or downscaling)

Average pooling

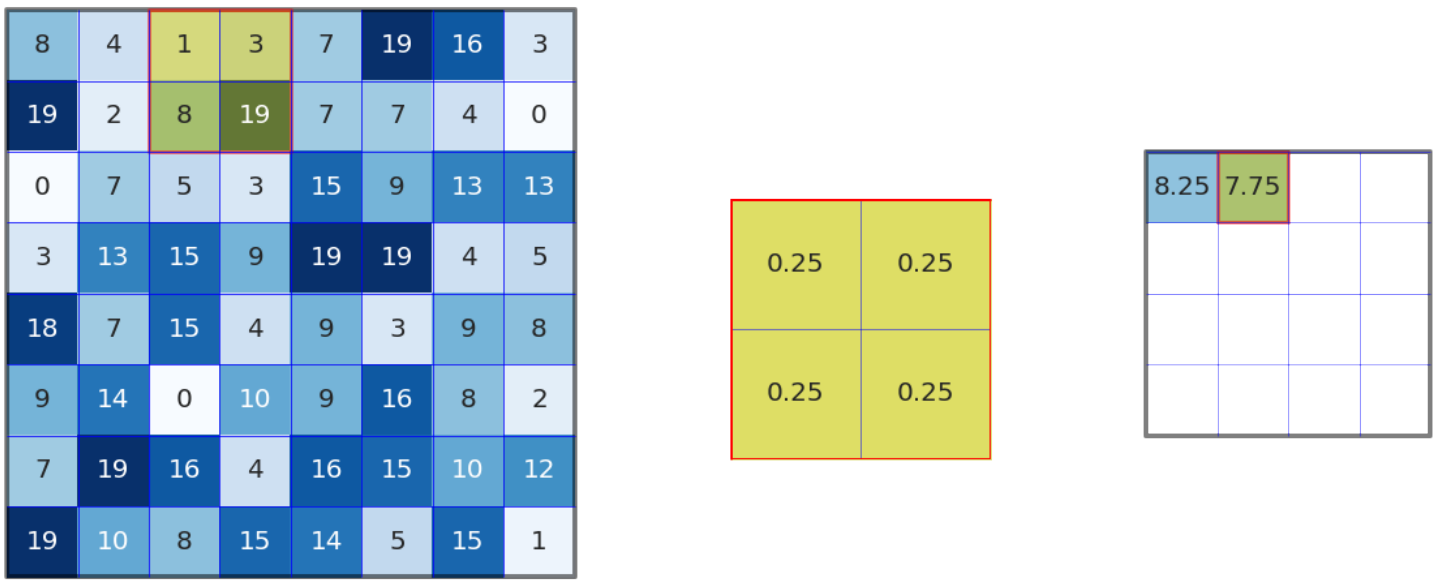
- Stride generally equal to kernel size
- No parameter to learn (no trainable parameters)



Pooling (or downscaling)

Average pooling

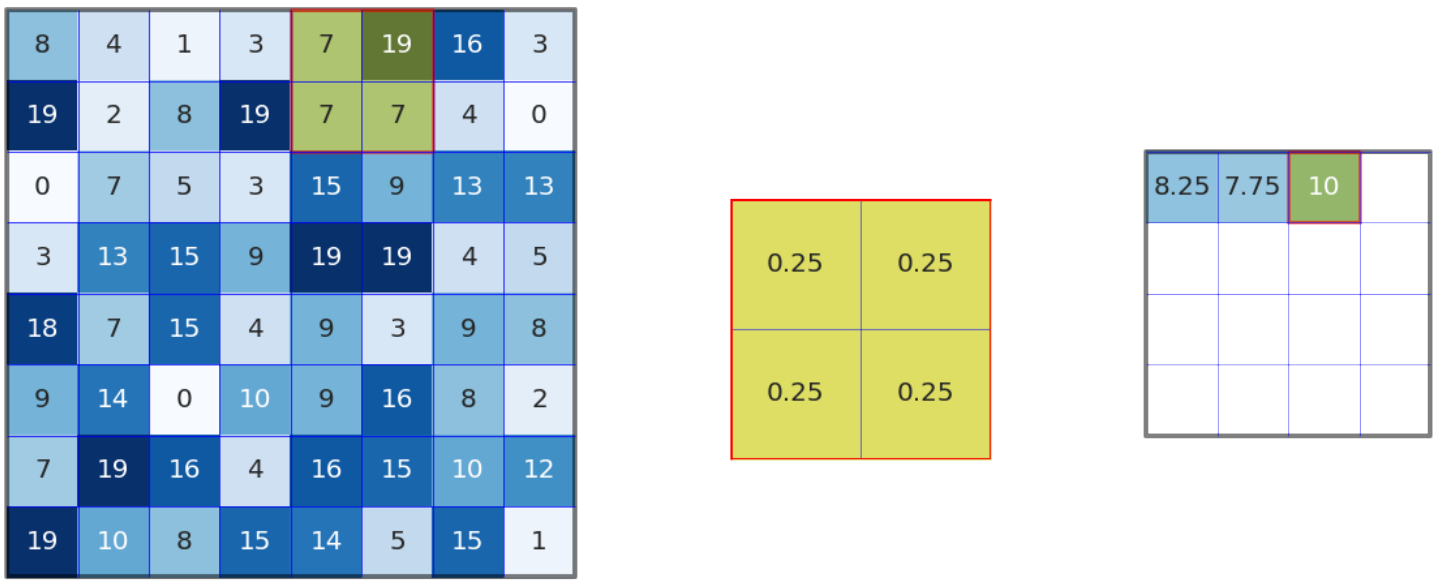
- Stride generally equal to kernel size
- No parameter to learn (no trainable parameters)



Pooling (or downscaling)

Average pooling

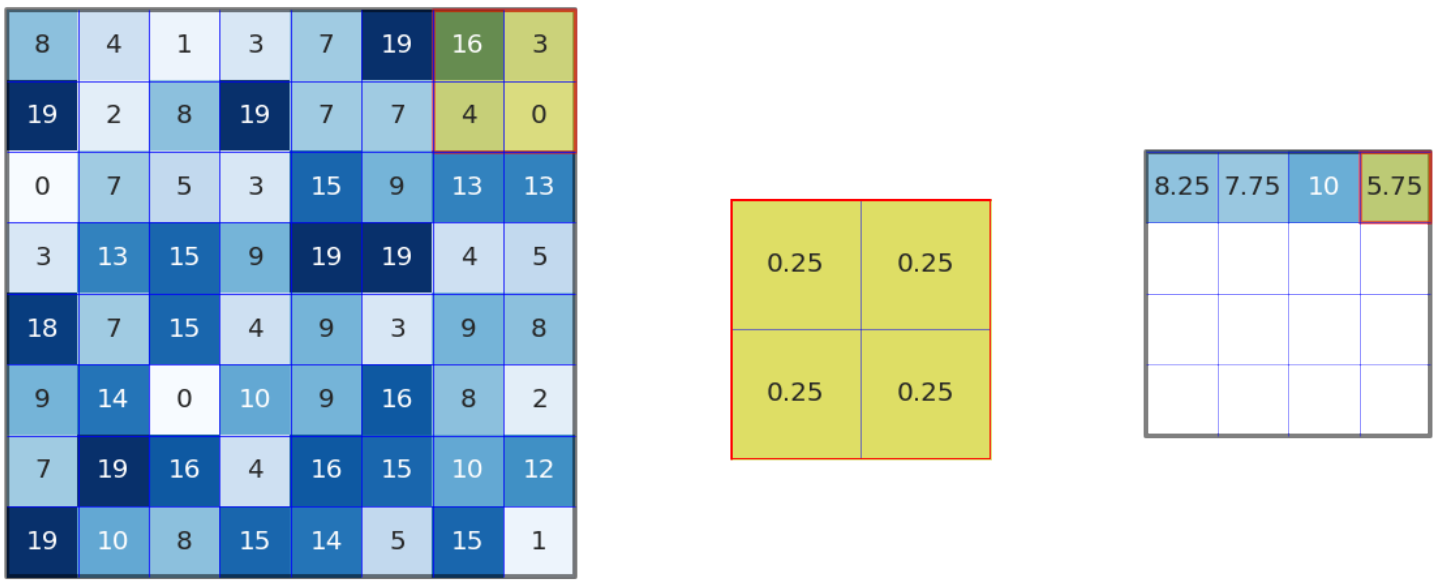
- Stride generally equal to kernel size
- No parameter to learn (no trainable parameters)



Pooling (or downscaling)

Average pooling

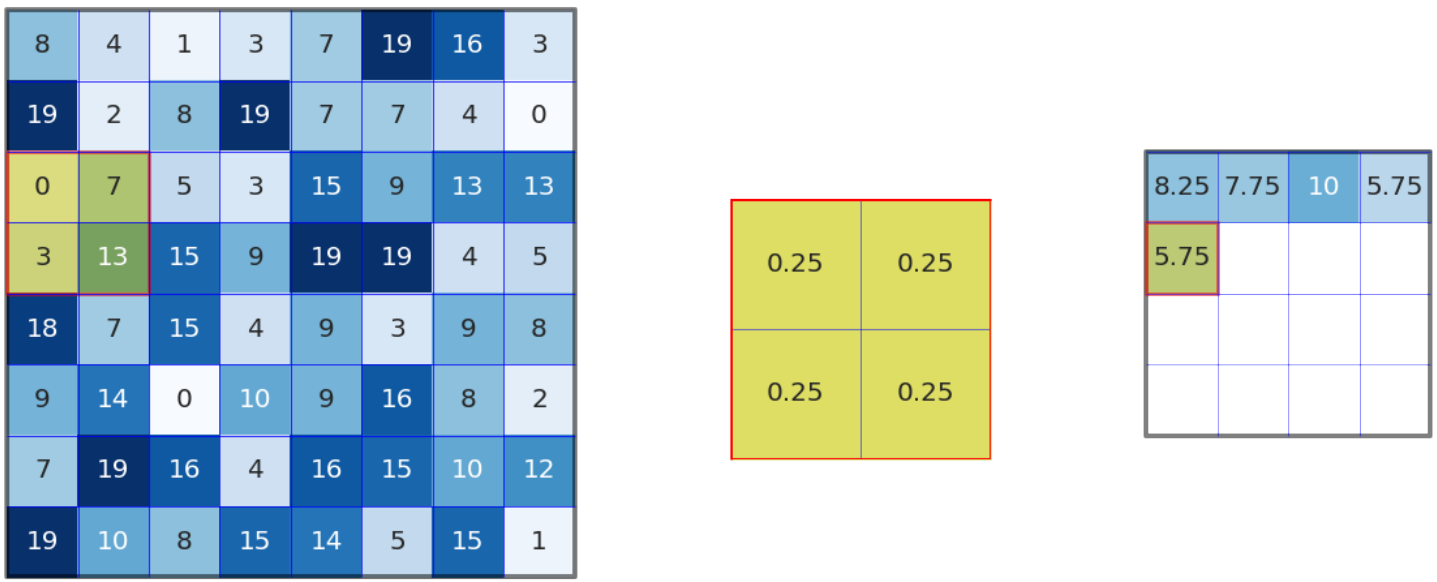
- Stride generally equal to kernel size
- No parameter to learn (no trainable parameters)



Pooling (or downscaling)

Average pooling

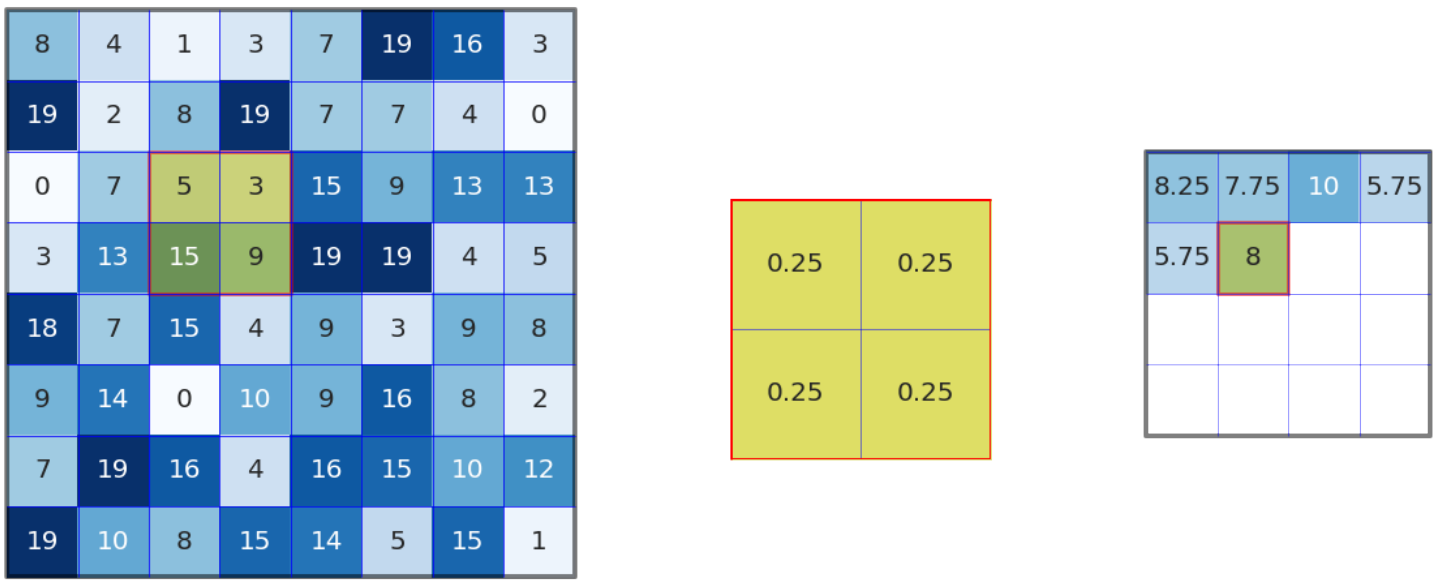
- Stride generally equal to kernel size
- No parameter to learn (no trainable parameters)



Pooling (or downscaling)

Average pooling

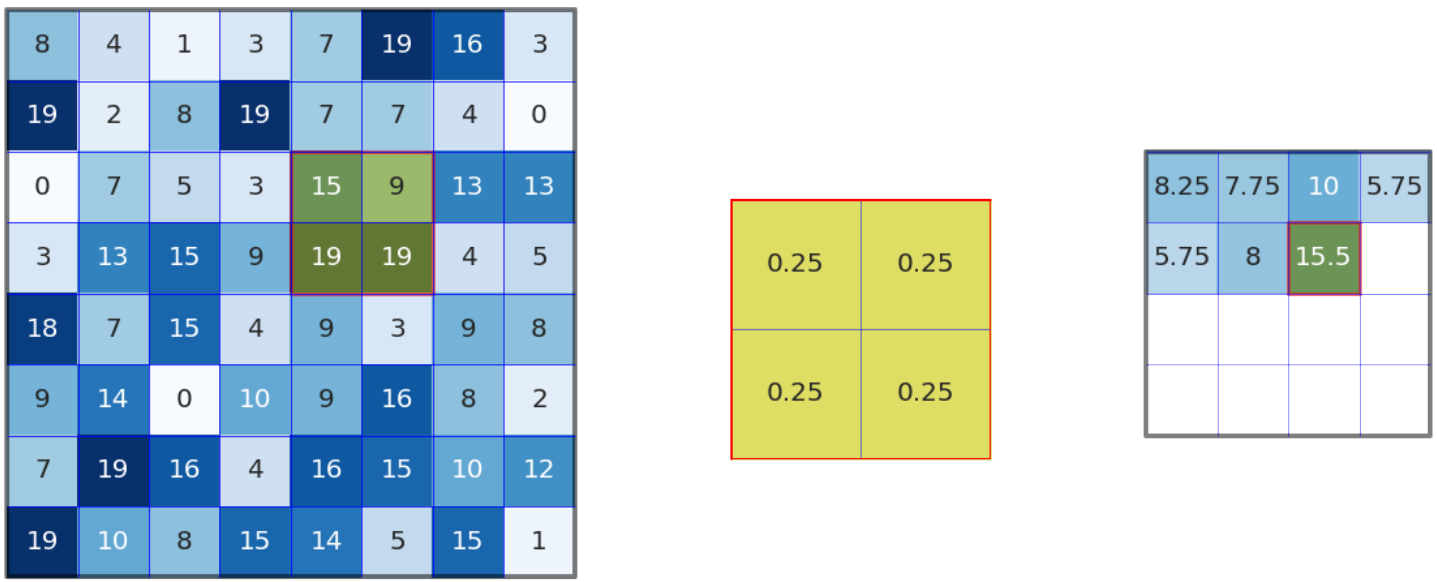
- Stride generally equal to kernel size
- No parameter to learn (no trainable parameters)



Pooling (or downscaling)

Average pooling

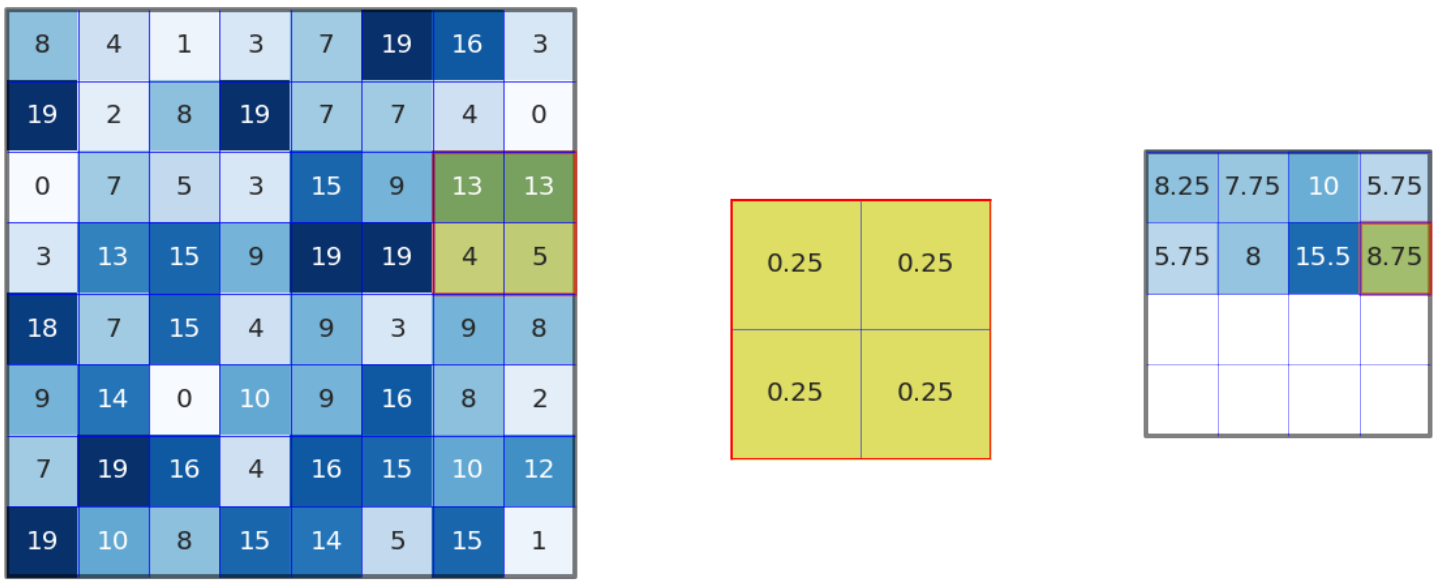
- Stride generally equal to kernel size
- No parameter to learn (no trainable parameters)



Pooling (or downscaling)

Average pooling

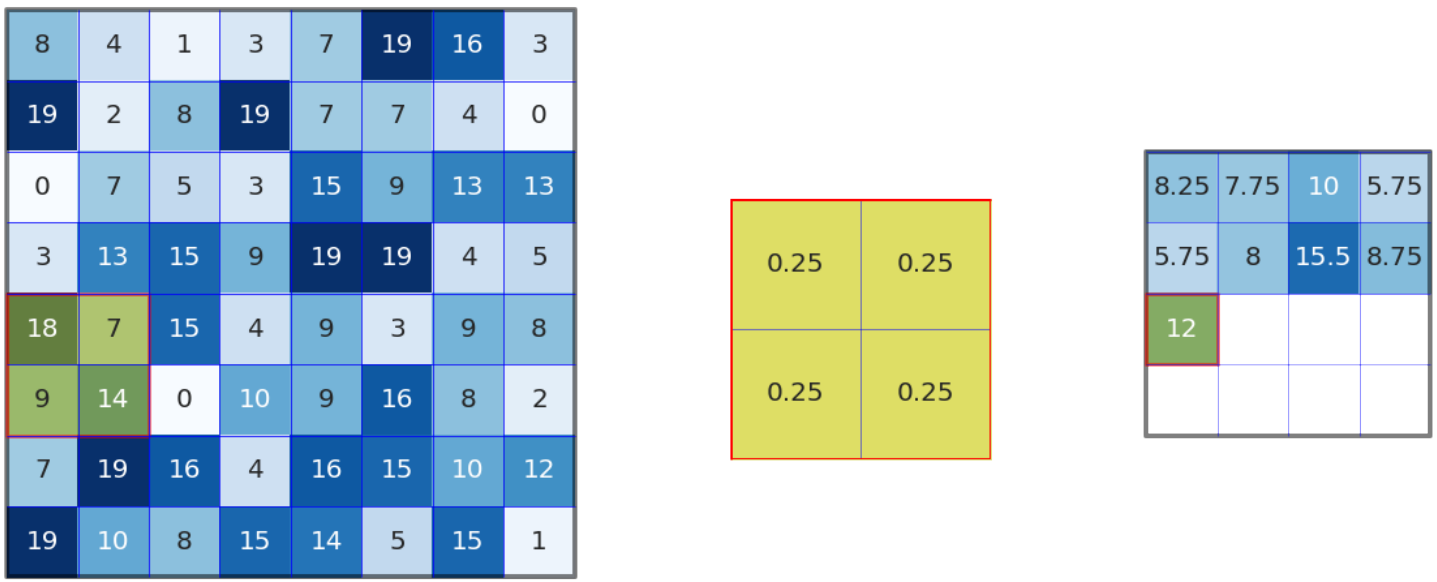
- Stride generally equal to kernel size
- No parameter to learn (no trainable parameters)



Pooling (or downscaling)

Average pooling

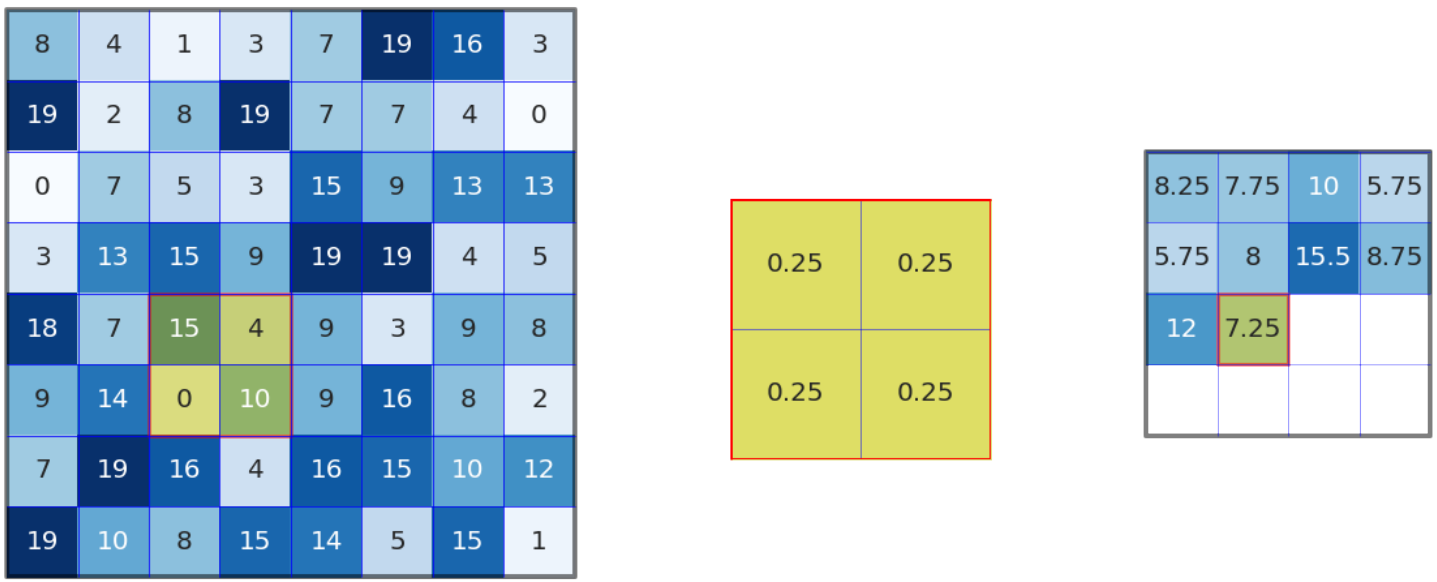
- Stride generally equal to kernel size
- No parameter to learn (no trainable parameters)



Pooling (or downscaling)

Average pooling

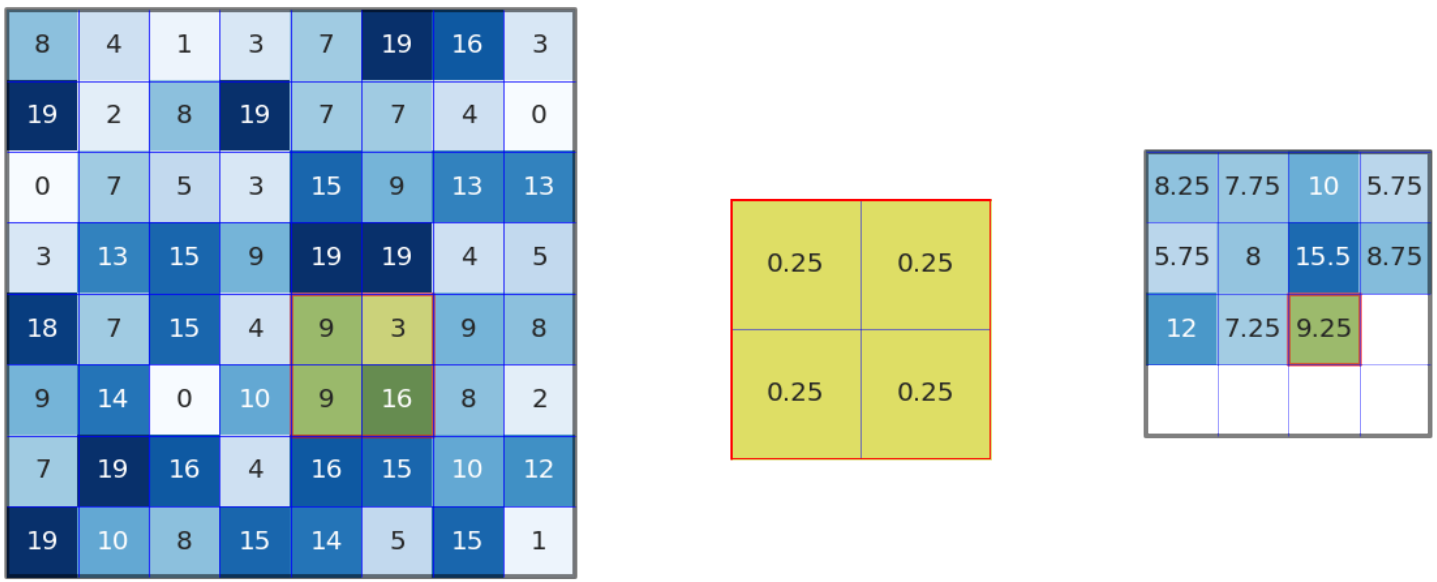
- Stride generally equal to kernel size
- No parameter to learn (no trainable parameters)



Pooling (or downscaling)

Average pooling

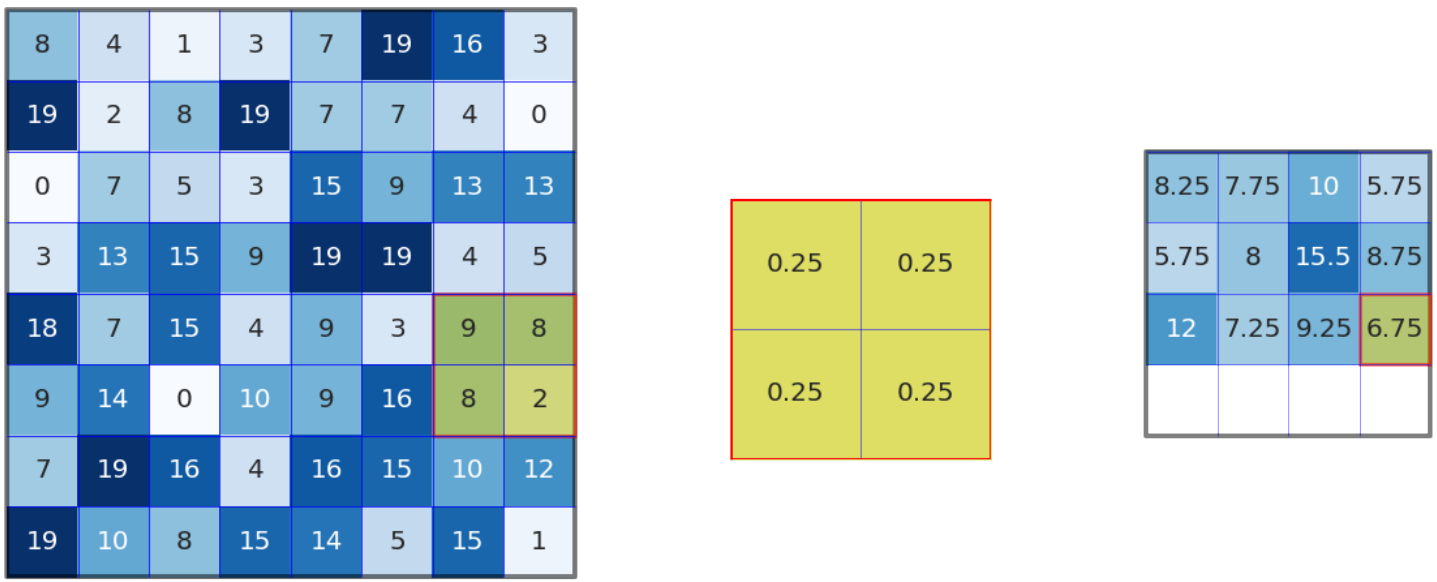
- Stride generally equal to kernel size
- No parameter to learn (no trainable parameters)



Pooling (or downscaling)

Average pooling

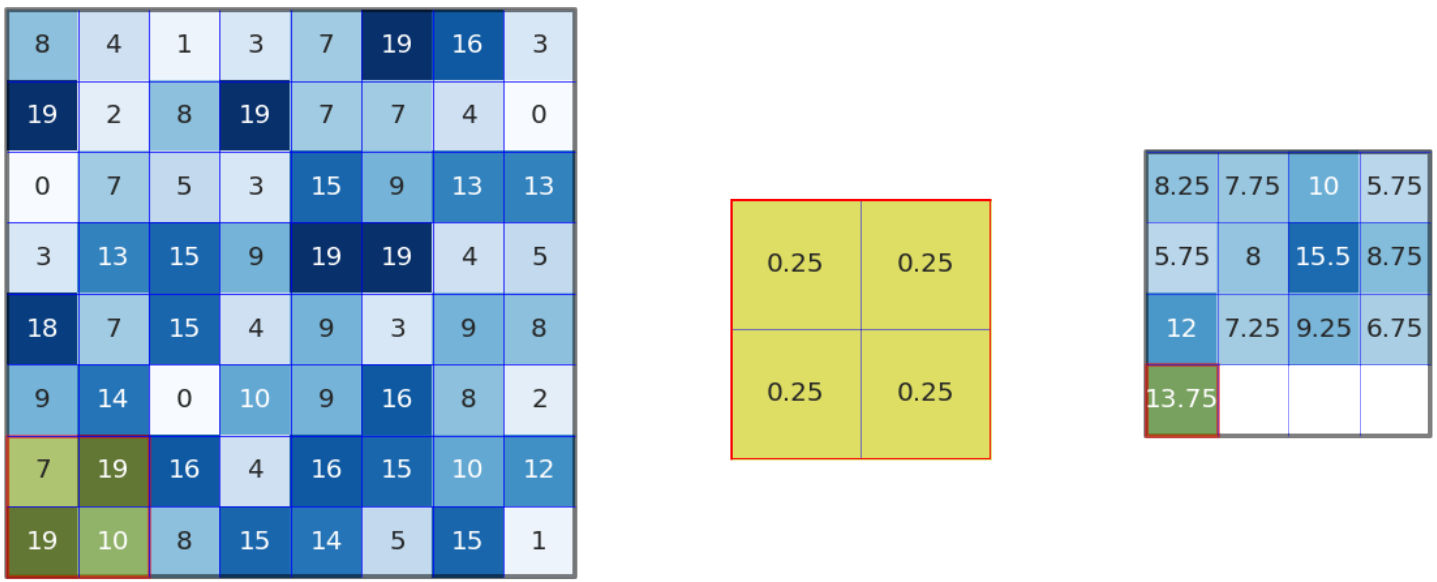
- Stride generally equal to kernel size
- No parameter to learn (no trainable parameters)



Pooling (or downscaling)

Average pooling

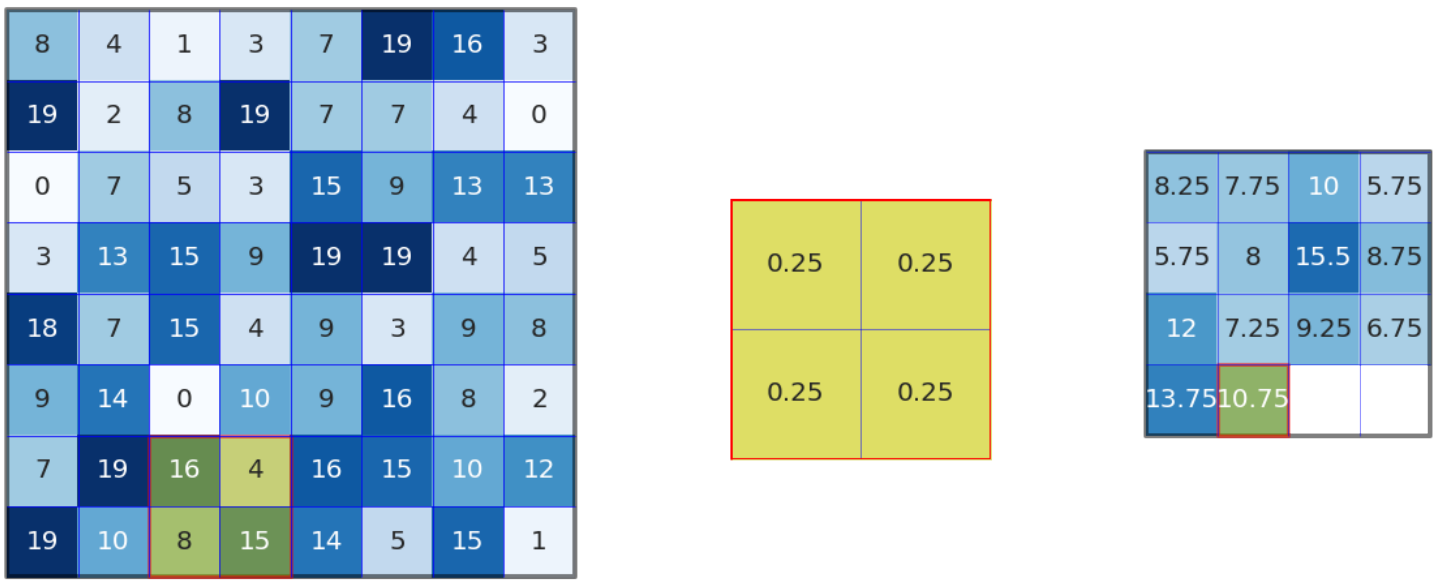
- Stride generally equal to kernel size
- No parameter to learn (no trainable parameters)



Pooling (or downscaling)

Average pooling

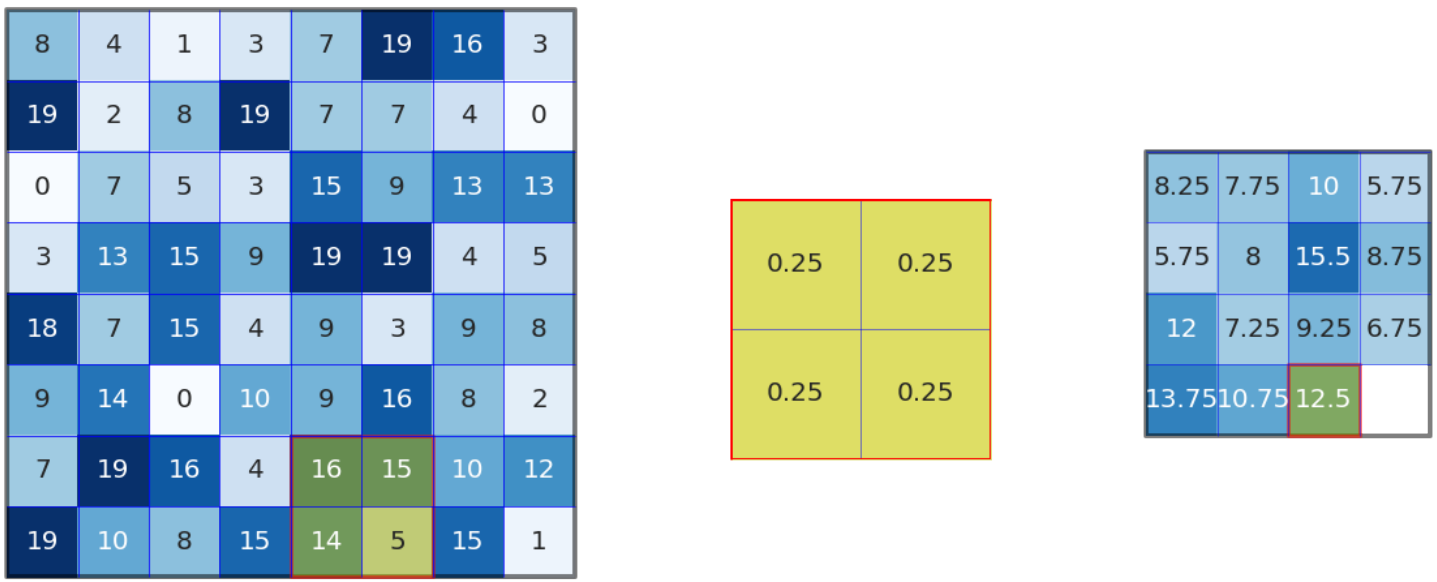
- Stride generally equal to kernel size
- No parameter to learn (no trainable parameters)



Pooling (or downscaling)

Average pooling

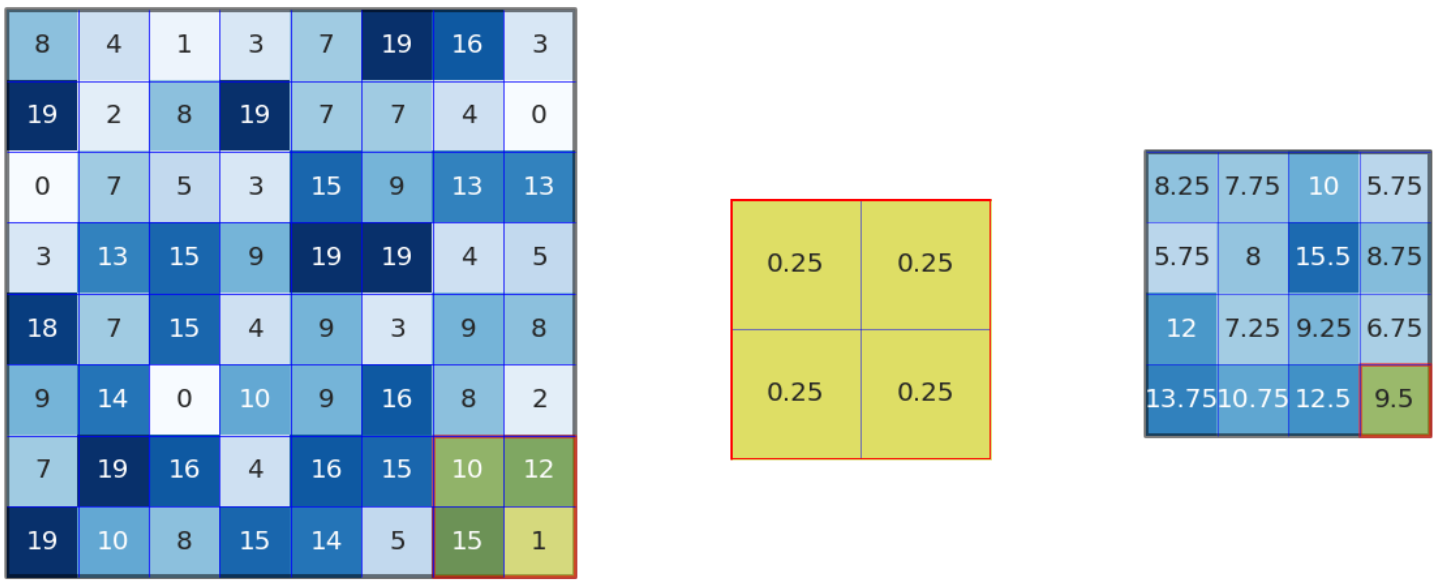
- Stride generally equal to kernel size
- No parameter to learn (no trainable parameters)



Pooling (or downscaling)

Average pooling

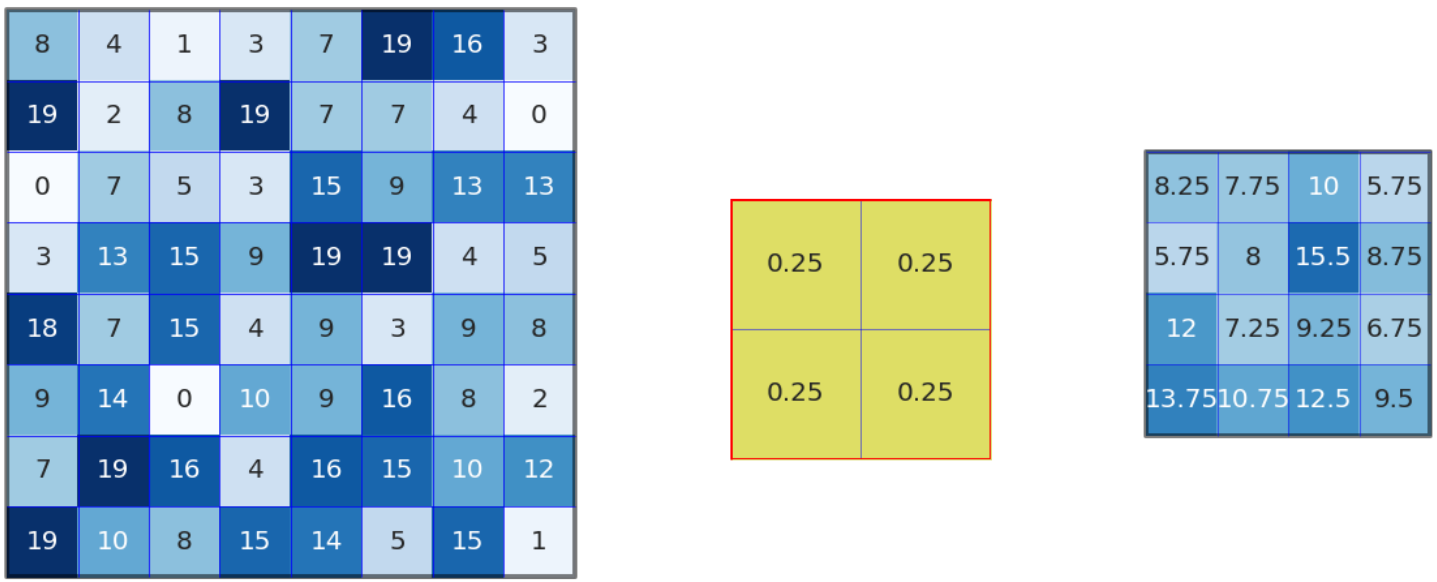
- Stride generally equal to kernel size
- No parameter to learn (no trainable parameters)



Pooling (or downscaling)

Average pooling

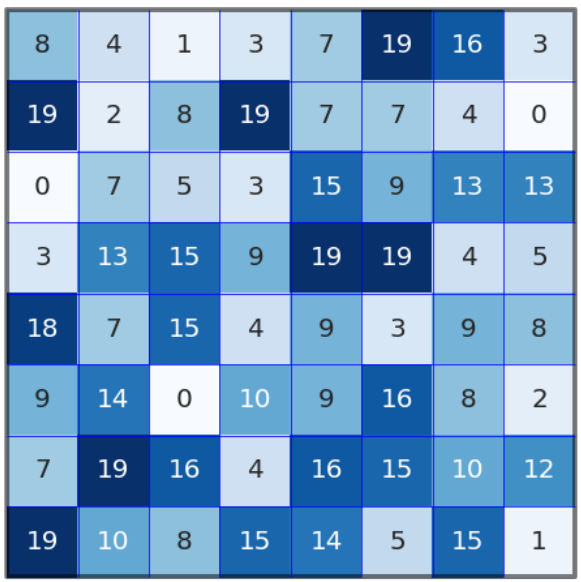
- Stride generally equal to kernel size
- No parameter to learn (no trainable parameters)

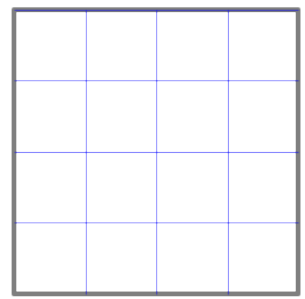


Pooling (or downscaling)

Max pooling

- Stride generally equal to kernel size
- No parameters to learn (no trainable parameters)

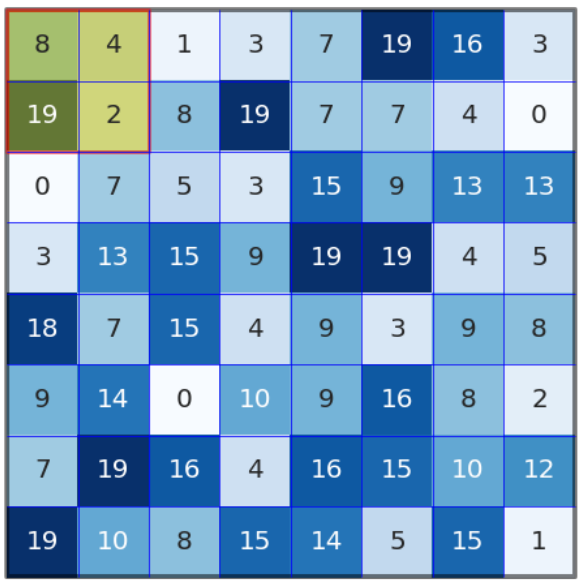


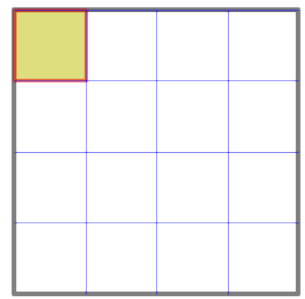


Pooling (or downscaling)

Max pooling

- Stride generally equal to kernel size
- No parameters to learn (no trainable parameters)

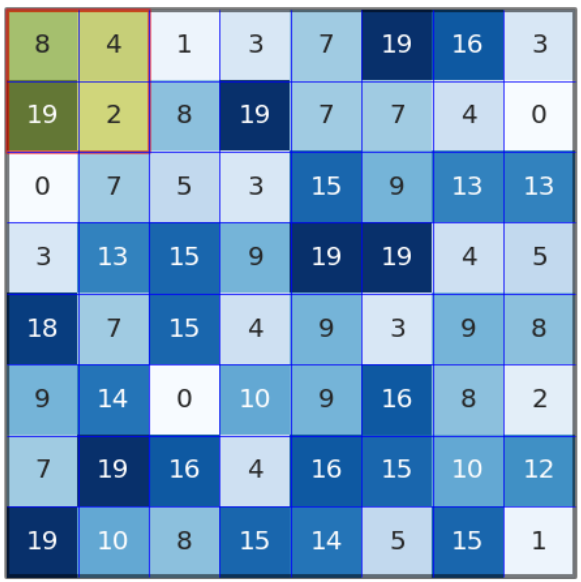


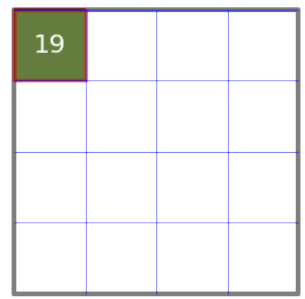


Pooling (or downscaling)

Max pooling

- Stride generally equal to kernel size
- No parameters to learn (no trainable parameters)

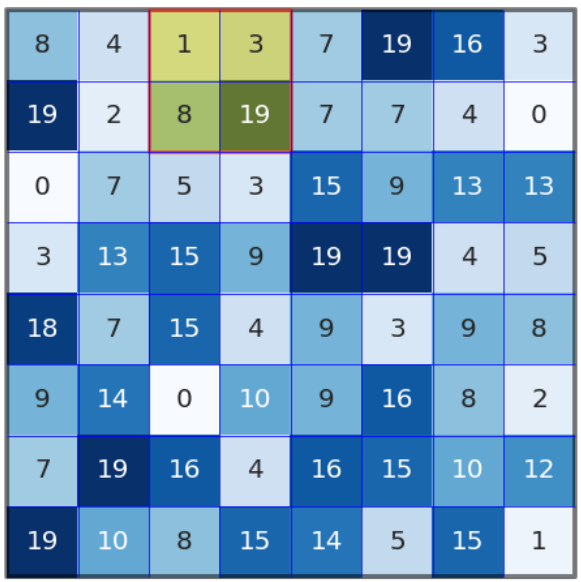




Pooling (or downscaling)

Max pooling

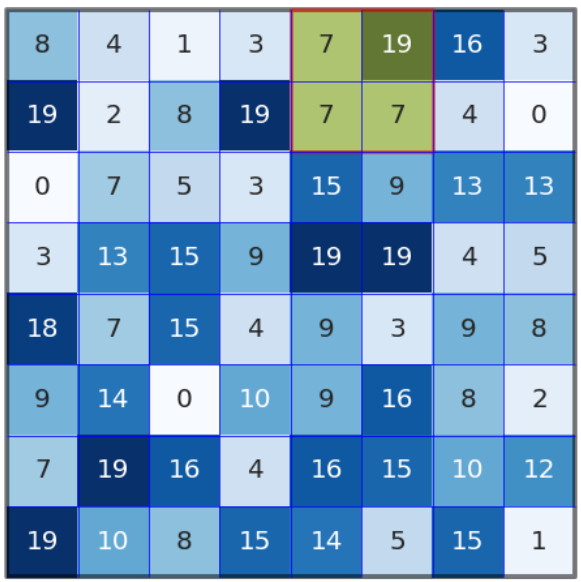
- Stride generally equal to kernel size
- No parameters to learn (no trainable parameters)



Pooling (or downscaling)

Max pooling

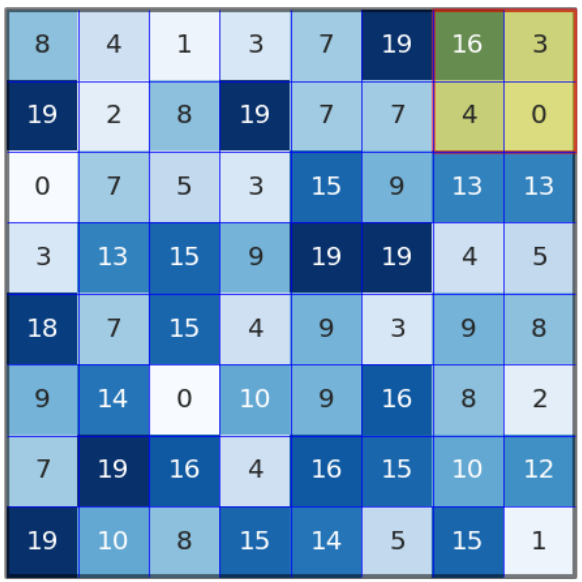
- Stride generally equal to kernel size
- No parameters to learn (no trainable parameters)



Pooling (or downscaling)

Max pooling

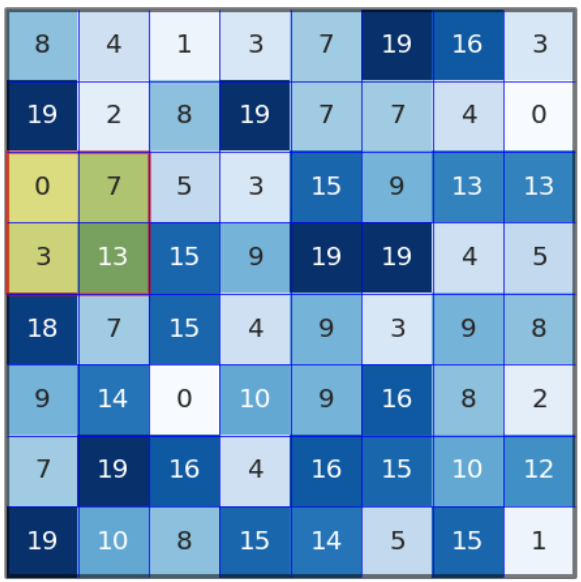
- Stride generally equal to kernel size
- No parameters to learn (no trainable parameters)



Pooling (or downscaling)

Max pooling

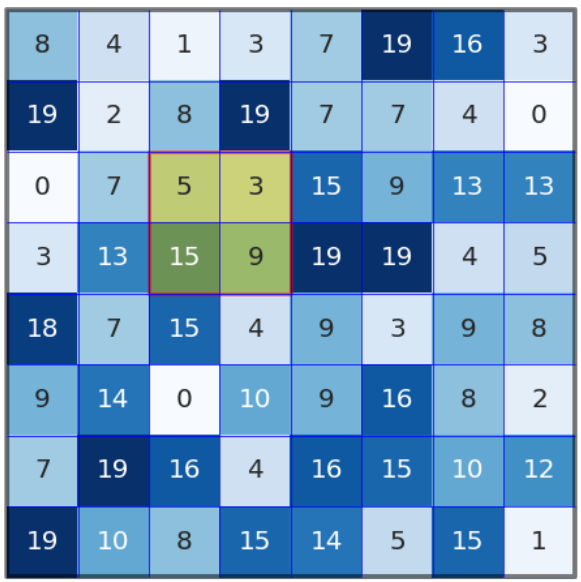
- Stride generally equal to kernel size
- No parameters to learn (no trainable parameters)

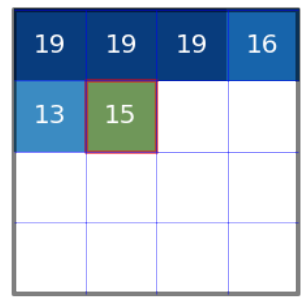


Pooling (or downscaling)

Max pooling

- Stride generally equal to kernel size
- No parameters to learn (no trainable parameters)

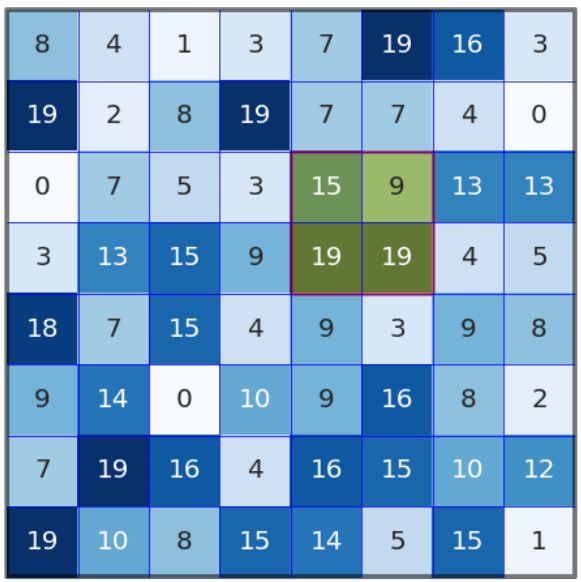


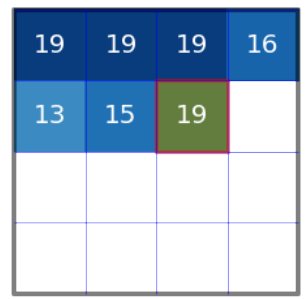


Pooling (or downscaling)

Max pooling

- Stride generally equal to kernel size
- No parameters to learn (no trainable parameters)

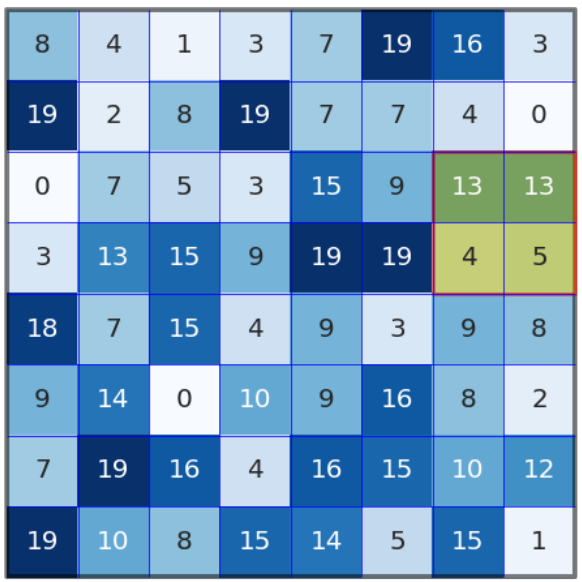




Pooling (or downscaling)

Max pooling

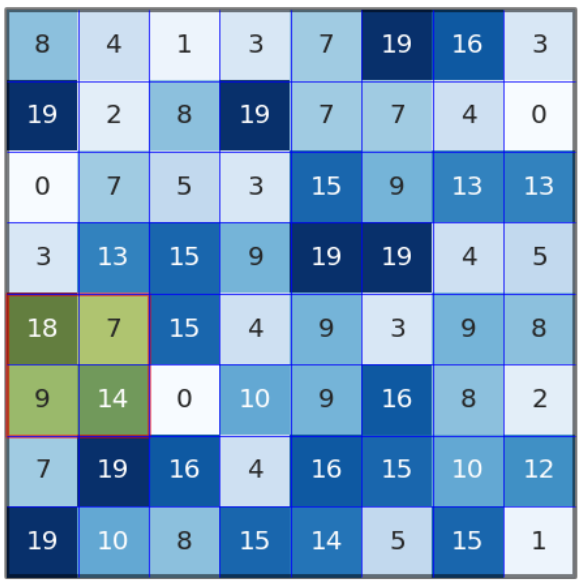
- Stride generally equal to kernel size
- No parameters to learn (no trainable parameters)

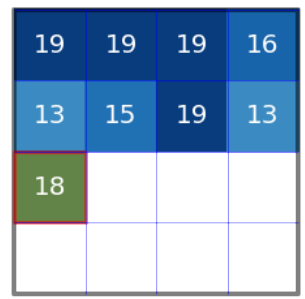


Pooling (or downscaling)

Max pooling

- Stride generally equal to kernel size
- No parameters to learn (no trainable parameters)

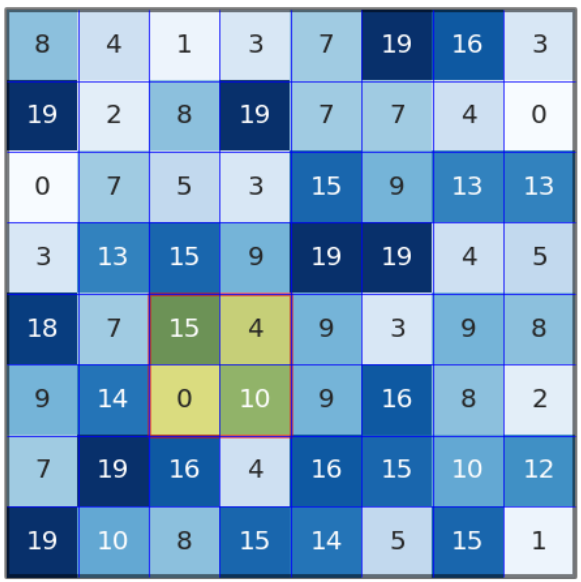


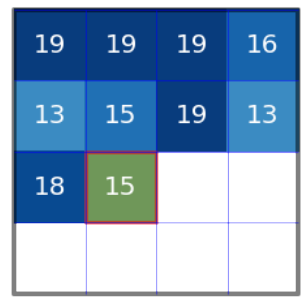


Pooling (or downscaling)

Max pooling

- Stride generally equal to kernel size
- No parameters to learn (no trainable parameters)

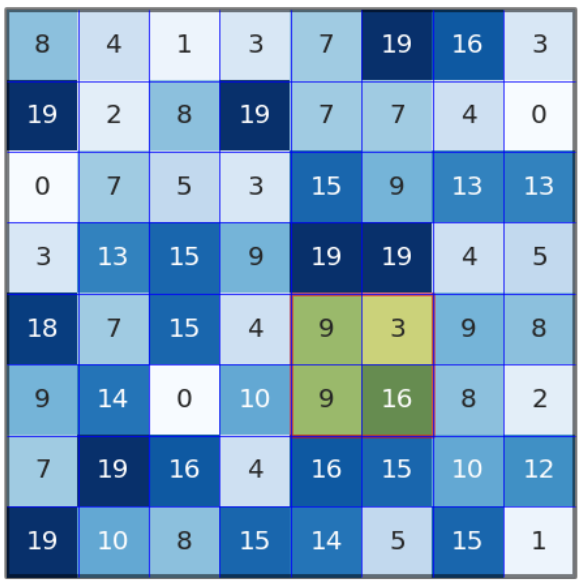


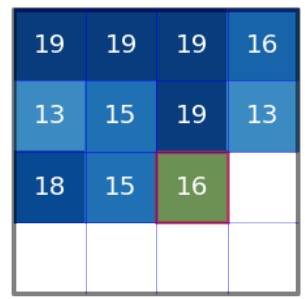


Pooling (or downscaling)

Max pooling

- Stride generally equal to kernel size
- No parameters to learn (no trainable parameters)

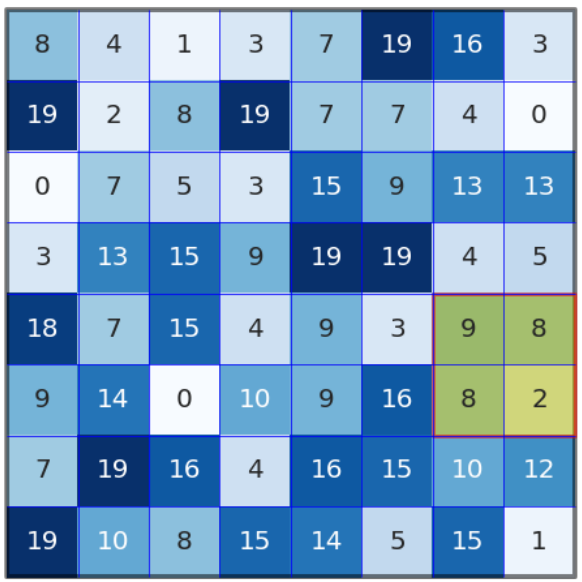


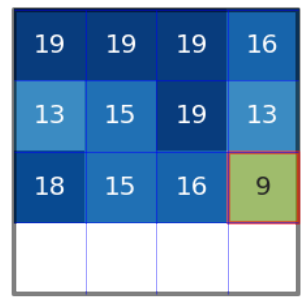


Pooling (or downscaling)

Max pooling

- Stride generally equal to kernel size
- No parameters to learn (no trainable parameters)

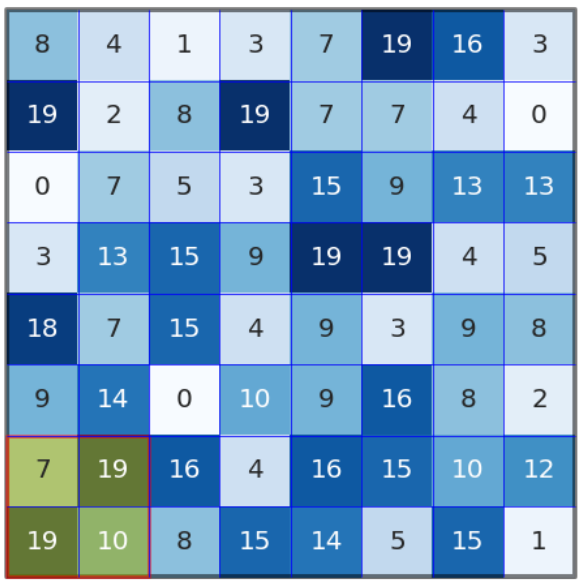


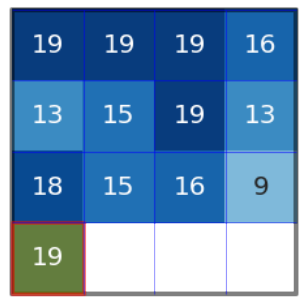


Pooling (or downscaling)

Max pooling

- Stride generally equal to kernel size
- No parameters to learn (no trainable parameters)

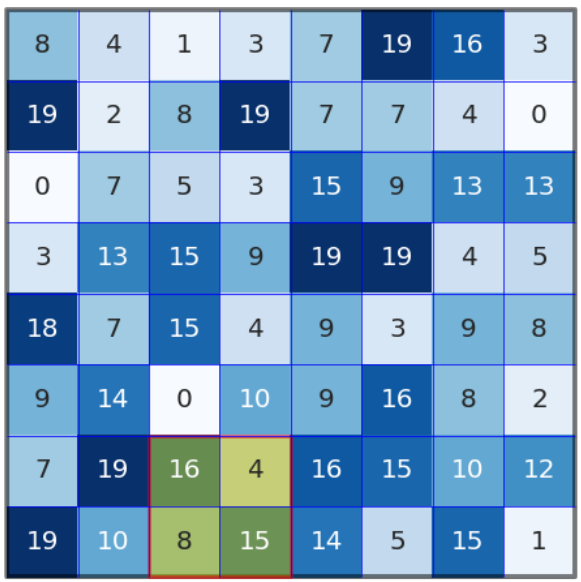


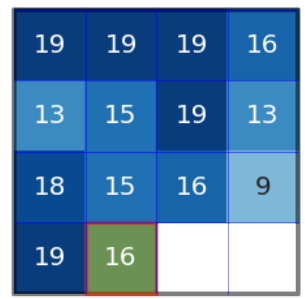


Pooling (or downscaling)

Max pooling

- Stride generally equal to kernel size
- No parameters to learn (no trainable parameters)

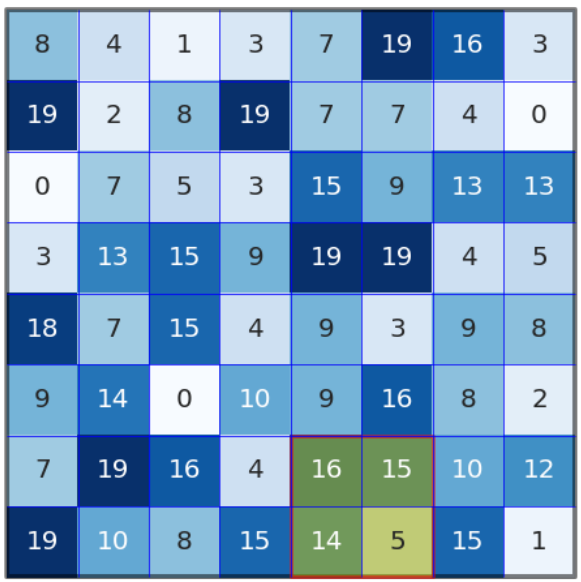


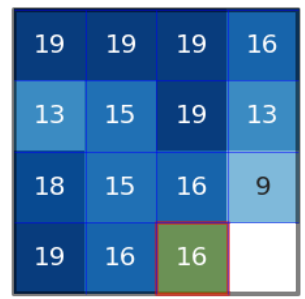


Pooling (or downscaling)

Max pooling

- Stride generally equal to kernel size
- No parameters to learn (no trainable parameters)

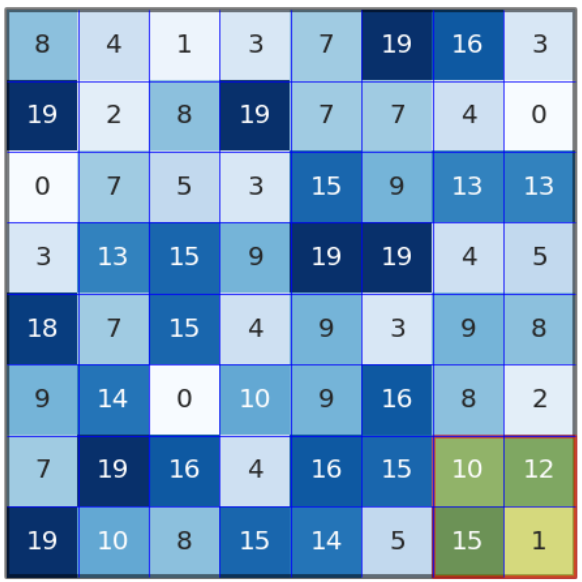


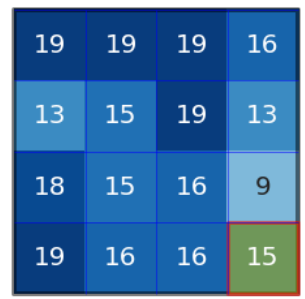


Pooling (or downscaling)

Max pooling

- Stride generally equal to kernel size
- No parameters to learn (no trainable parameters)

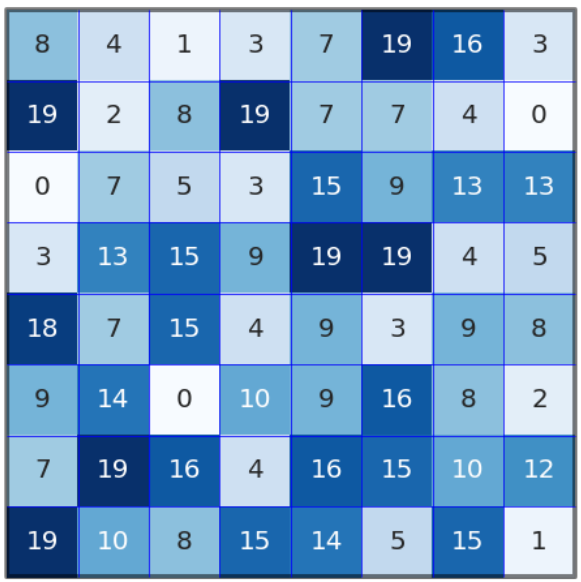




Pooling (or downscaling)

Max pooling

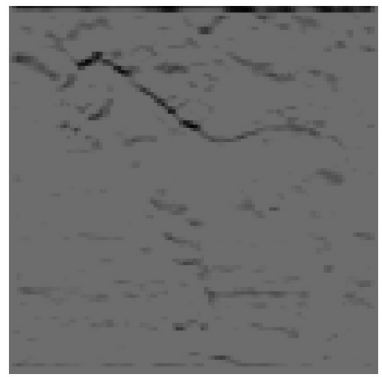
- Stride generally equal to kernel size
- No parameters to learn (no trainable parameters)

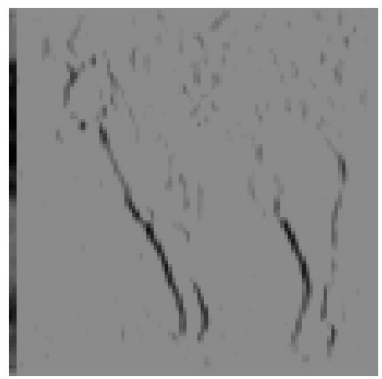


Example on a "real" image

Example on a "real" image

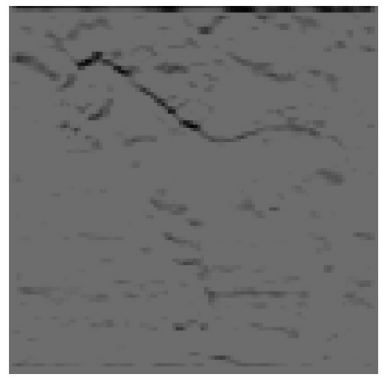
After ReLU



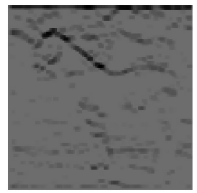


Example on a "real" image

After ReLU

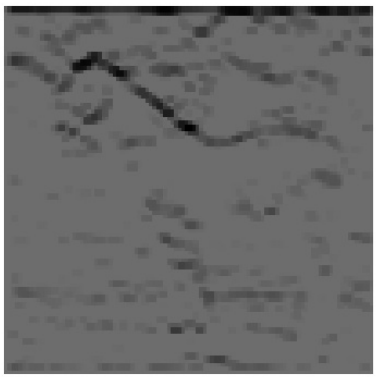
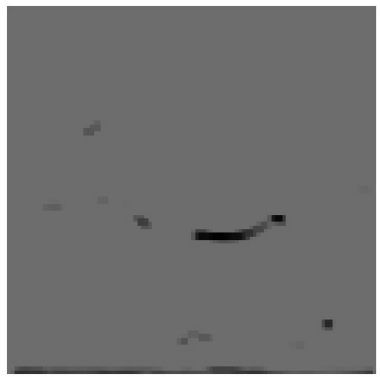
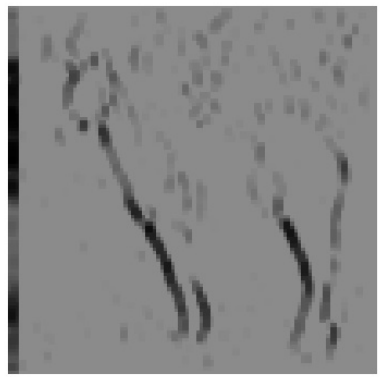
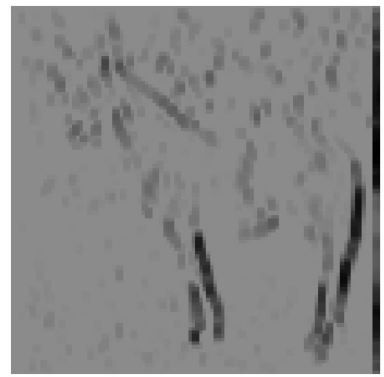


After max-pooling



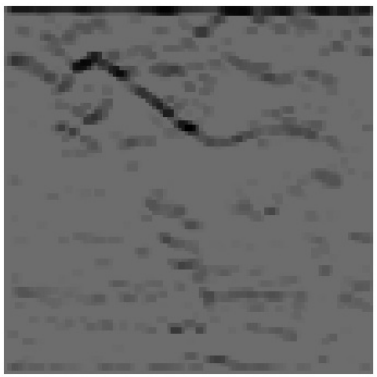
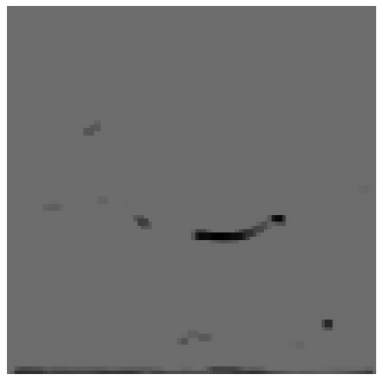
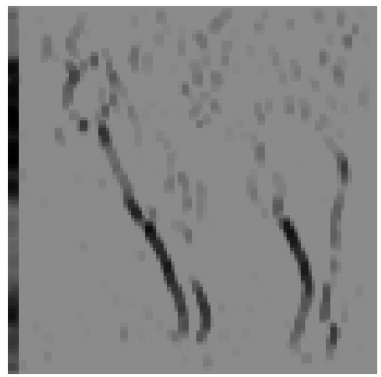
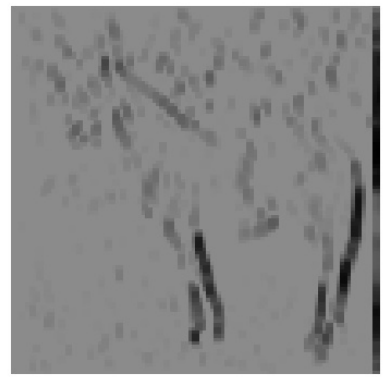
Example on a "real" image

Example on a "real" image

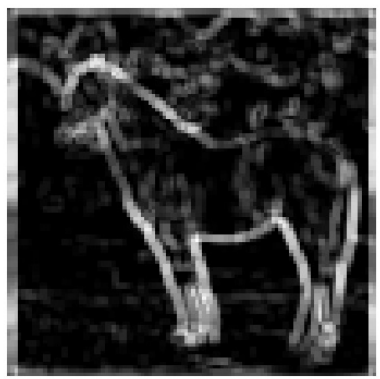





4 filters

Example on a "real" image

4 filters

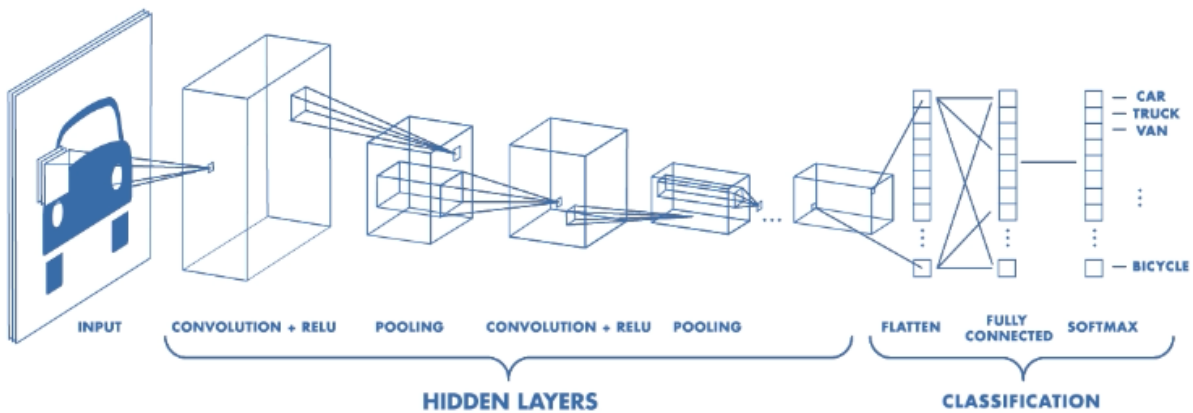


After a (1×1) convolution

With stride

Typical architecture

- Most modern convolutional networks architectures stack several blocks composed of the following operations:
  - A convolutional layer (product by a sparse matrix)
  - An activation function (ReLU)
  - A max-pooling layer
- The sizes of the feature maps progressively reduce whilst the number of channels increases
- Finally a few fully connected layers are applied until the categories.



Source: Photo via FloydHub.com

Batch-normalization (Ioffe and Szegedy, 2015)

- Used to stabilize the training (reduce covariance shift?)
- Ingredient to "plug" after linear layers (in general before activation)
- The values y_i of each output d of a layer are normalized over the batch:

$$\hat{y}_i = \frac{y_i - \mu_d}{\sqrt{\sigma_d^2 + \varepsilon}}$$

  i is the index of the example in the batch, μ_d and σ_d^2 are the empirical mean and variance of the output computed over the batch and ε is a small number which prevent numerical instabilities
- Then the activation function is applied on

$$\beta_d + \gamma_d \hat{y}_i$$

  where β_d and γ_d are new parameters learned by back-propagation.

Remarks on batch-norm

- Bias in a layer before batch-normalization (BN) is useless (since outputs are centered)

Remarks on batch-norm

- Bias in a layer before batch-normalization (BN) is useless (since outputs are centered)
- At test phase, one may use the trained network on only one input (no batch). Therefore μ_d and σ_d have to be fixed during the training phase. It is done by an exponentially moving weighted average:

Remarks on batch-norm

- Bias in a layer before batch-normalization (BN) is useless (since outputs are centered)
- At test phase, one may use the trained network on only one input (no batch). Therefore μ_d and σ_d have to be fixed during the training phase. It is done by an exponentially moving weighted average:

Algorithm for μ_d

Initialization at first epoch:

$$\hat{\mu}_d^{(1)} = \mu_d^{(1)}$$

At epoch $t+1$:

$$\hat{\mu}_d^{(t+1)} = \alpha \mu_d^{(t)} + (1-\alpha)\hat{\mu}_d^{(t)}$$

Remarks on batch-norm

- Bias in a layer before batch-normalization (BN) is useless (since outputs are centered)
- At test phase, one may use the trained network on only one input (no batch). Therefore μ_d and σ_d have to be fixed during the training phase. It is done by an exponentially moving weighted average:

Algorithm for μ_d

Initialization at first epoch:

$$\hat{\mu}_d^{(1)} = \mu_d^{(1)}$$

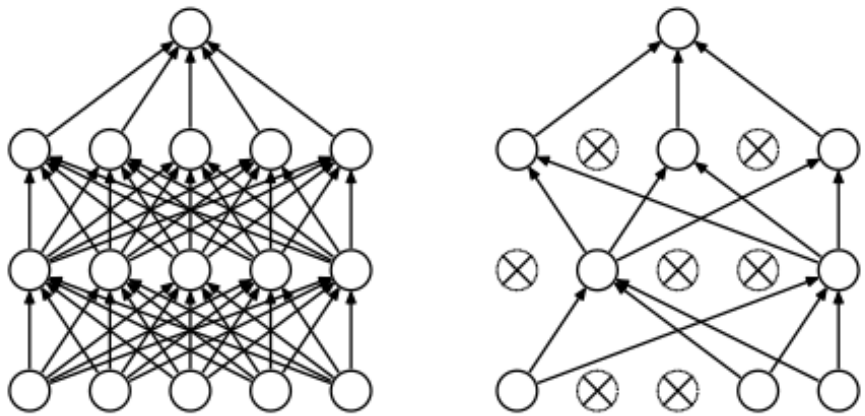
At epoch $t+1$:

$$\hat{\mu}_d^{(t+1)} = \alpha \mu_d^{(t)} + (1-\alpha)\hat{\mu}_d^{(t)}$$

- Other normalization can be used (e.g layer normalization,...)

Dropout (Srivastava et al., 2014)

- During training phase, at each epoch, a proportion p of the units are randomly suppressed (their incoming and outcoming weights are set to zero).
- At test phase, all the units are used and their weights a multiplied by p



(a) Standard Neural Net (b) After applying dropout.

- Avoids specialization of group of neurons (regularization)
- Doubles the number of iterations required to converge
- Reduces the computing time of each epoch
- Has to be used on Fully Connected Layers only
- p can be set between 0.2 and 0.4.

Vanishing/Exploding gradient

$$\frac{d\mathcal{L}}{d\omega_{i,j}^{(L)}} = \frac{d\mathcal{L}}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}} \frac{d\mathbf{s}^{(L)}}{d\omega_{i,j}^{(L)}}$$

$$\frac{d\mathcal{L}}{d\omega_{i,j}^{(L-1)}} = \frac{d\mathcal{L}}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}} \frac{d\mathbf{s}^{(L)}}{d\mathbf{v}^{(L-1)}} \frac{d\mathbf{v}^{(L-1)}}{d\mathbf{s}^{(L-1)}} \frac{d\mathbf{s}^{(L-1)}}{d\omega_{i,j}^{(L-1)}}$$

$$\frac{d\mathcal{L}}{d\omega_{i,j}^{(L-2)}} = \underbrace{\underbrace{\underbrace{\frac{d\mathcal{L}}{d\mathbf{v}^{(L)}} \frac{d\mathbf{v}^{(L)}}{d\mathbf{s}^{(L)}}}_{E^{(L)}} \frac{d\mathbf{s}^{(L)}}{d\mathbf{v}^{(L-1)}} \frac{d\mathbf{v}^{(L-1)}}{d\mathbf{s}^{(L-1)}}}_{E^{(L-1)}} \frac{d\mathbf{s}^{(L-1)}}{d\mathbf{v}^{(L-2)}} \frac{d\mathbf{v}^{(L-2)}}{d\mathbf{s}^{(L-2)}}}_{E^{(L-2)}} \frac{d\mathbf{s}^{(L-2)}}{d\omega_{i,j}^{(L-2)}}$$

- For very deep networks, the components of the gradient of the cost function for the parameters of the early layers (close to the inputs) often explode or vanish.

Some first solutions

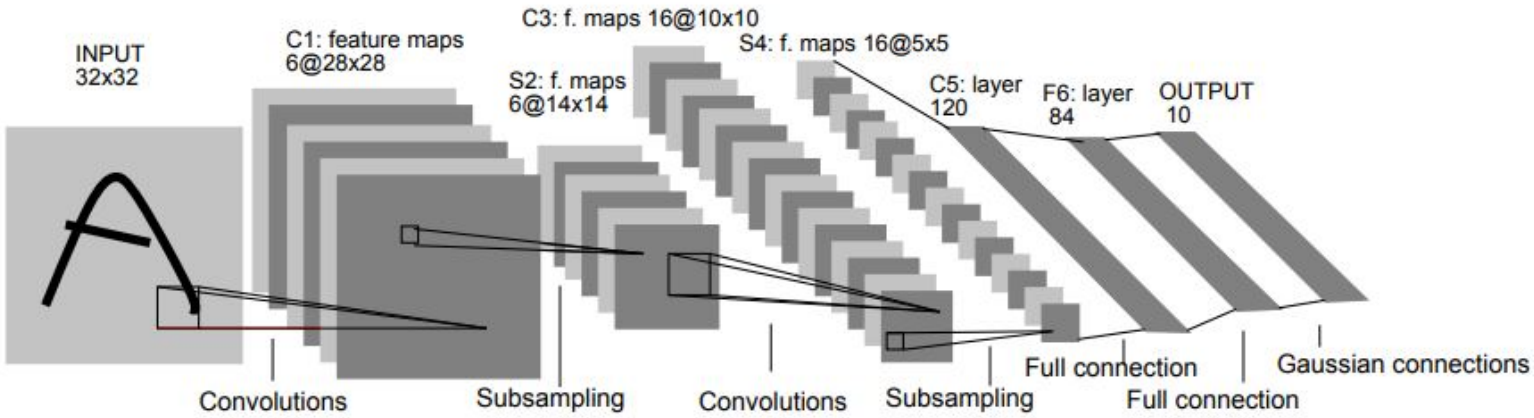
- Use ReLu (or Leaky ReLU) activation functions in the intermediate layers
- Initialize the weights of a layer according to the Xavier's rule (Glorot and Bengio, 2010). For instance with a gaussian with variance

$$\frac{u}{n}$$

where n is the number of inputs of the layer and $u = 1$ or 2 (for ReLU).
- Other specific solutions are proposed (see later)

LeNet

LeCun et al. (1998)



First convolutional neural network (about 60 000 parameters)

AlexNet (Krizhevsky et al., 2012)

- Won ImageNet with a large margin (outperformed in 2015)
- 5 convolutional layers, 3 fully connected layers
- 61 millions of parameters
- Use of GPU

GoogLeNet

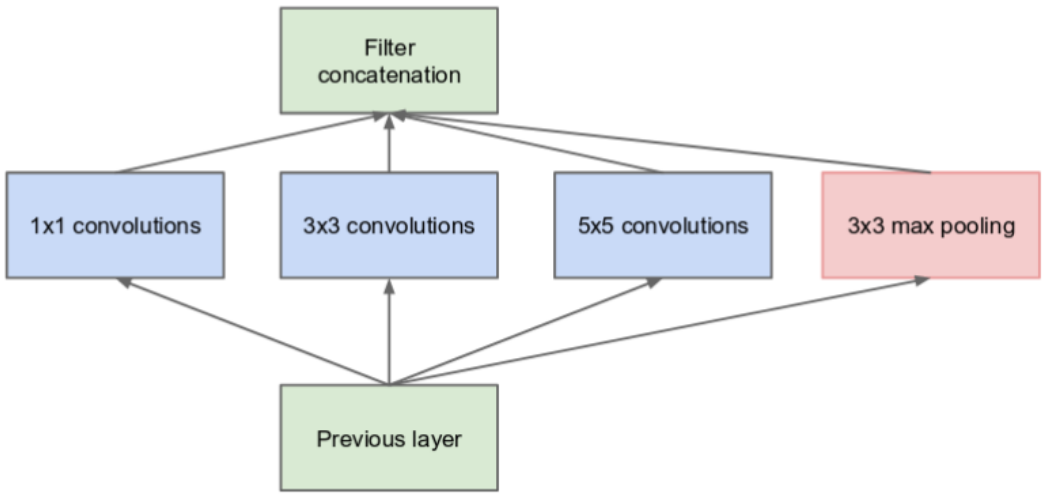
Or Inception networks (Szegedy et al., 2015)

Source: A simple Guide to the versions of the inception network

Interesting parts of the images (with respect to the classification task) can have different scales

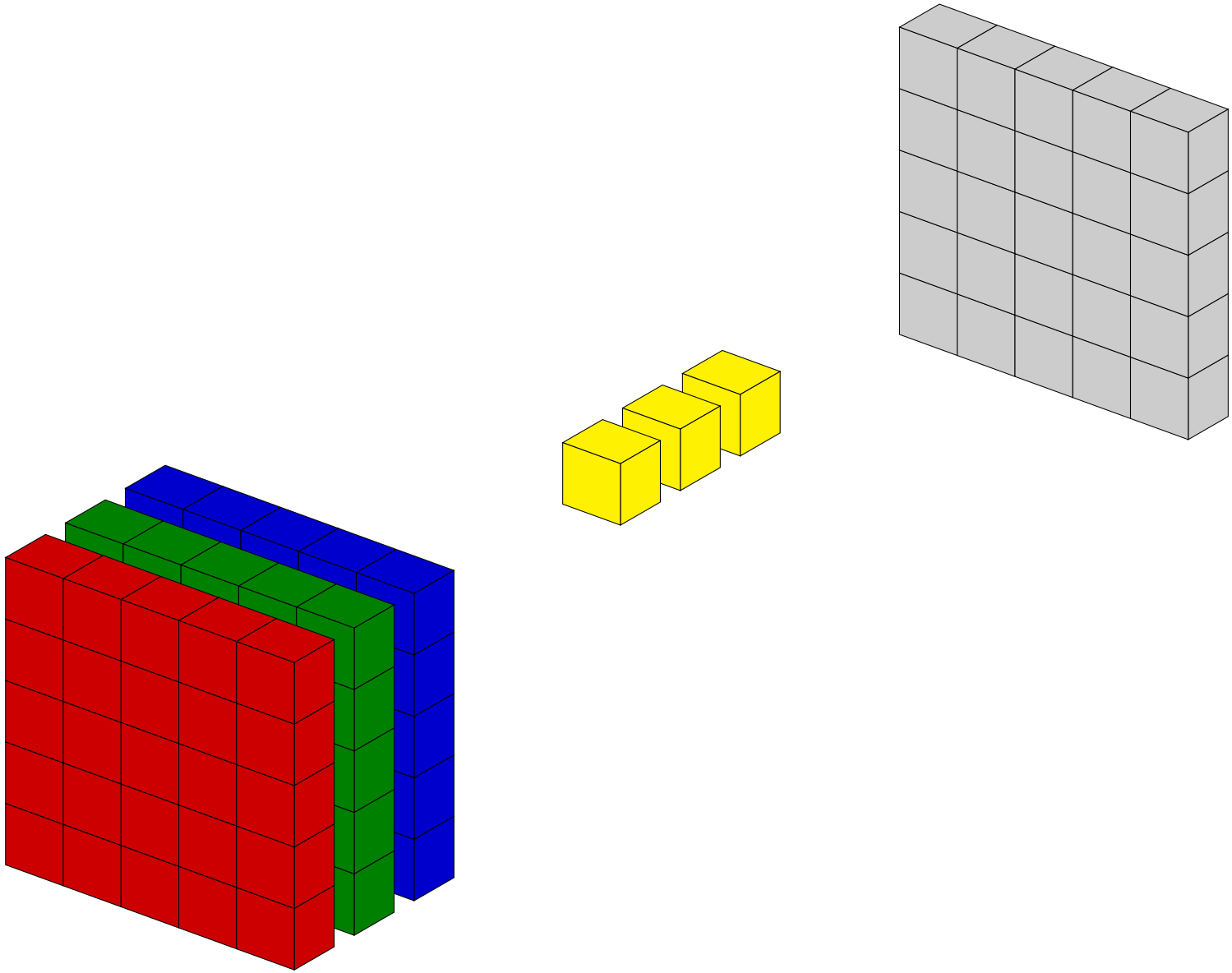
Inception V1

Basic block

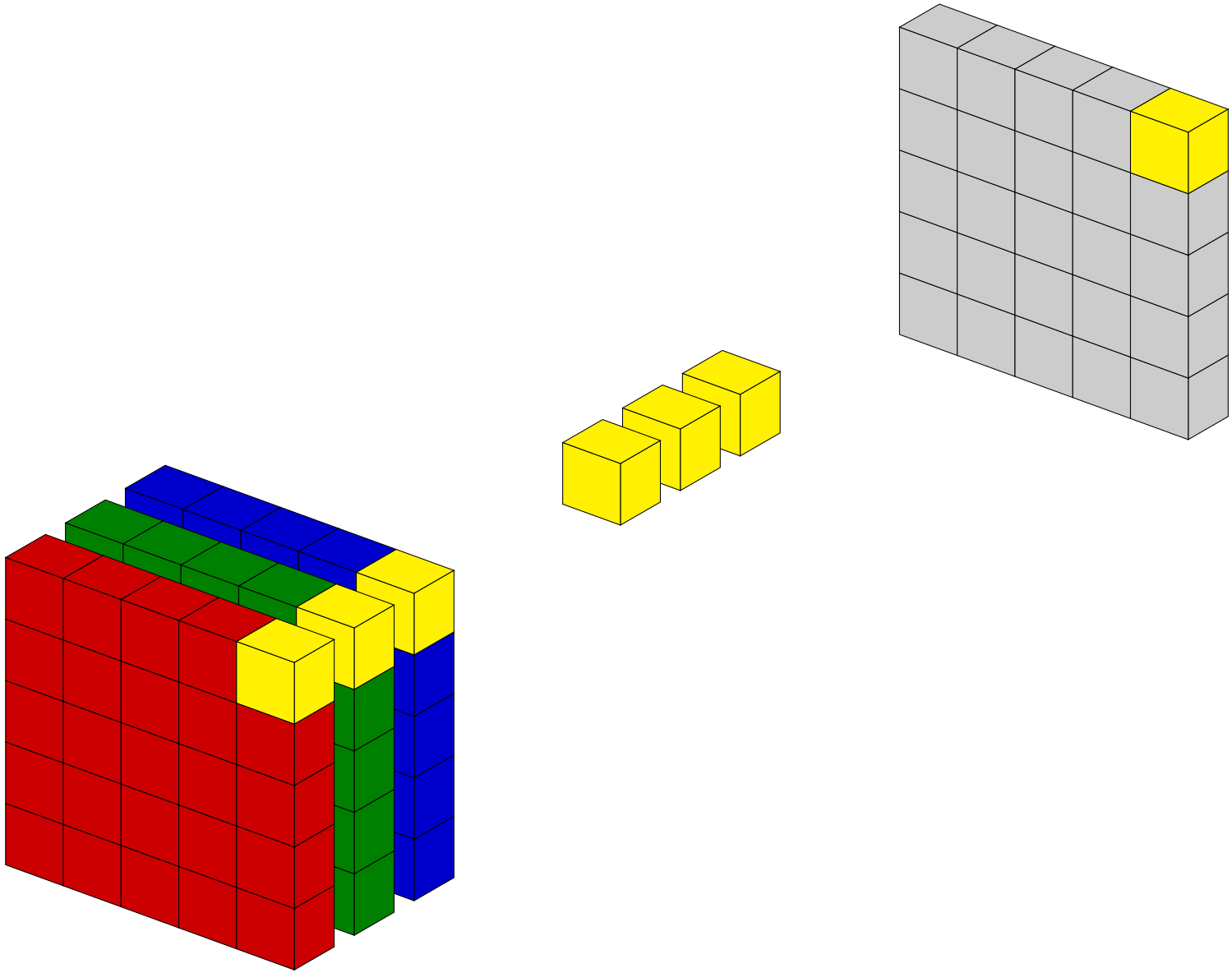


Filter with various sizes in a layer

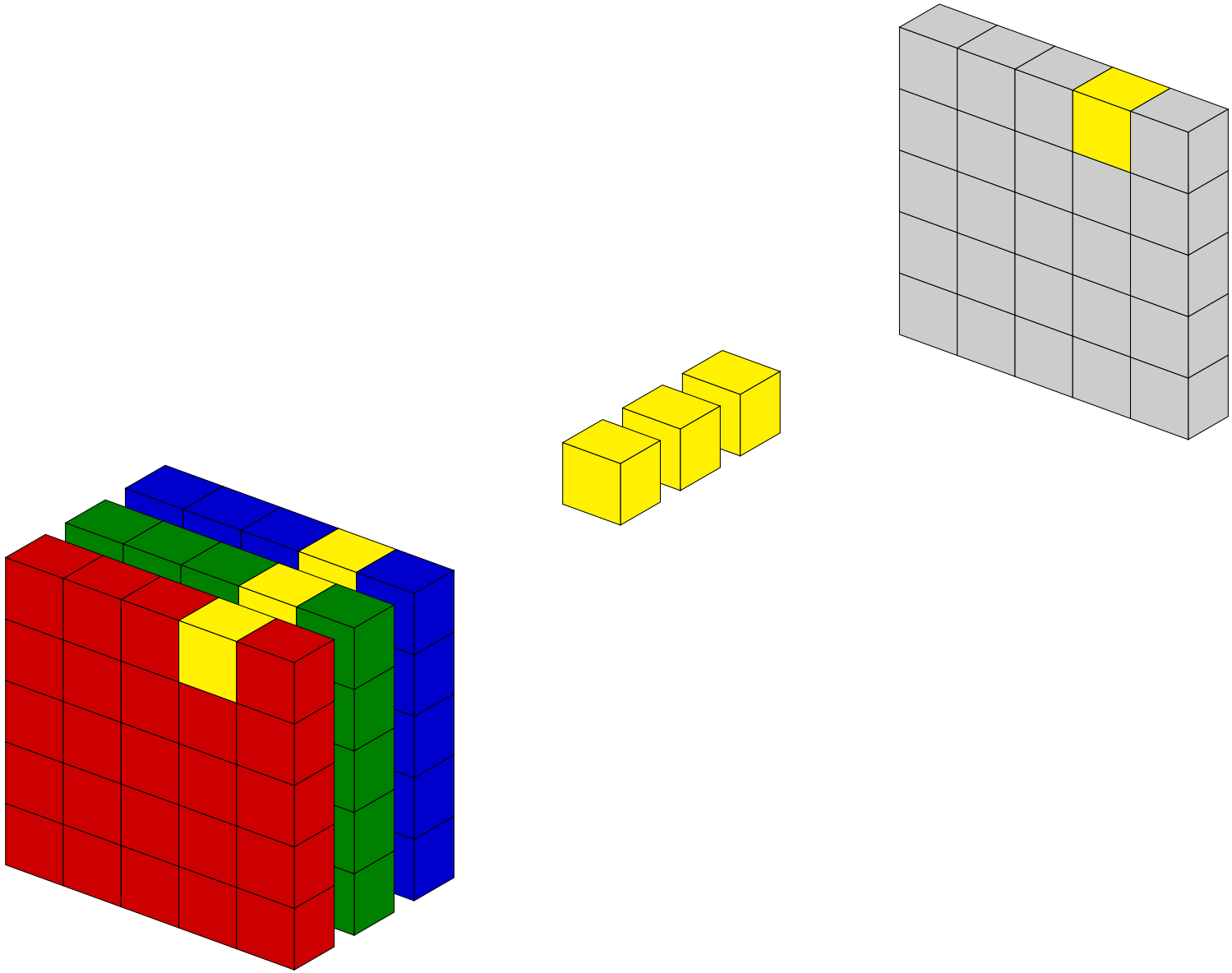
(1×1) convolution



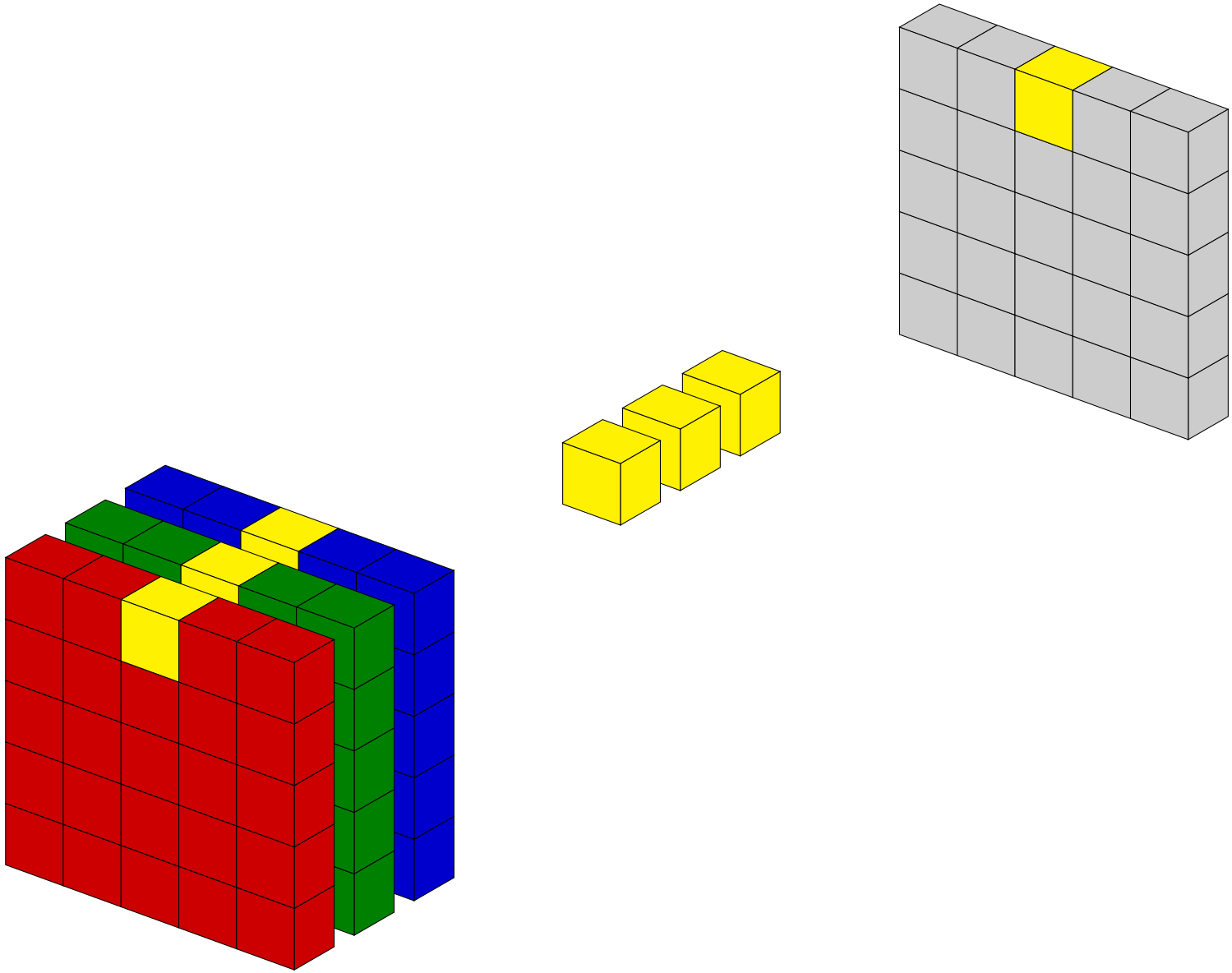
(1×1) convolution



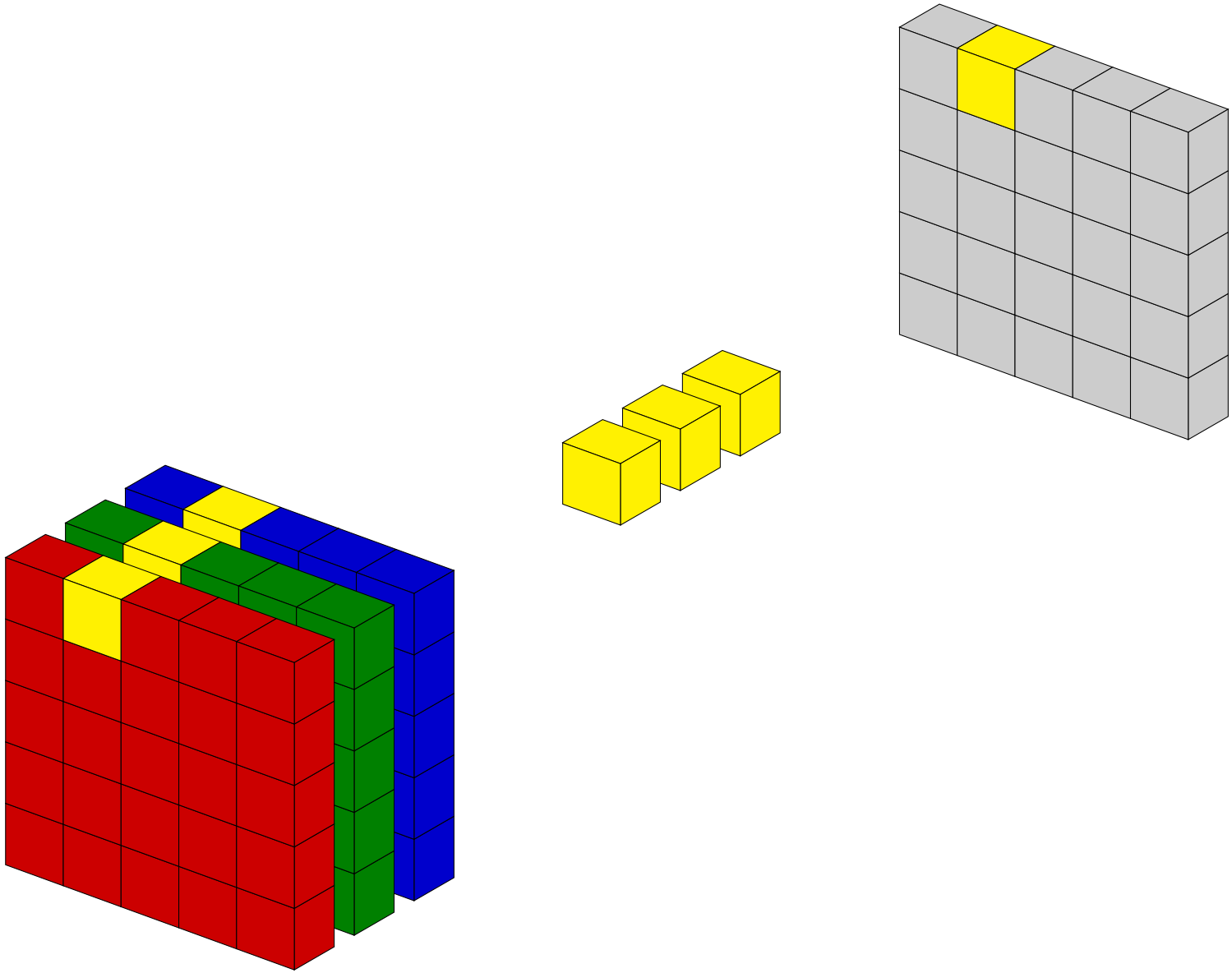
(1×1) convolution



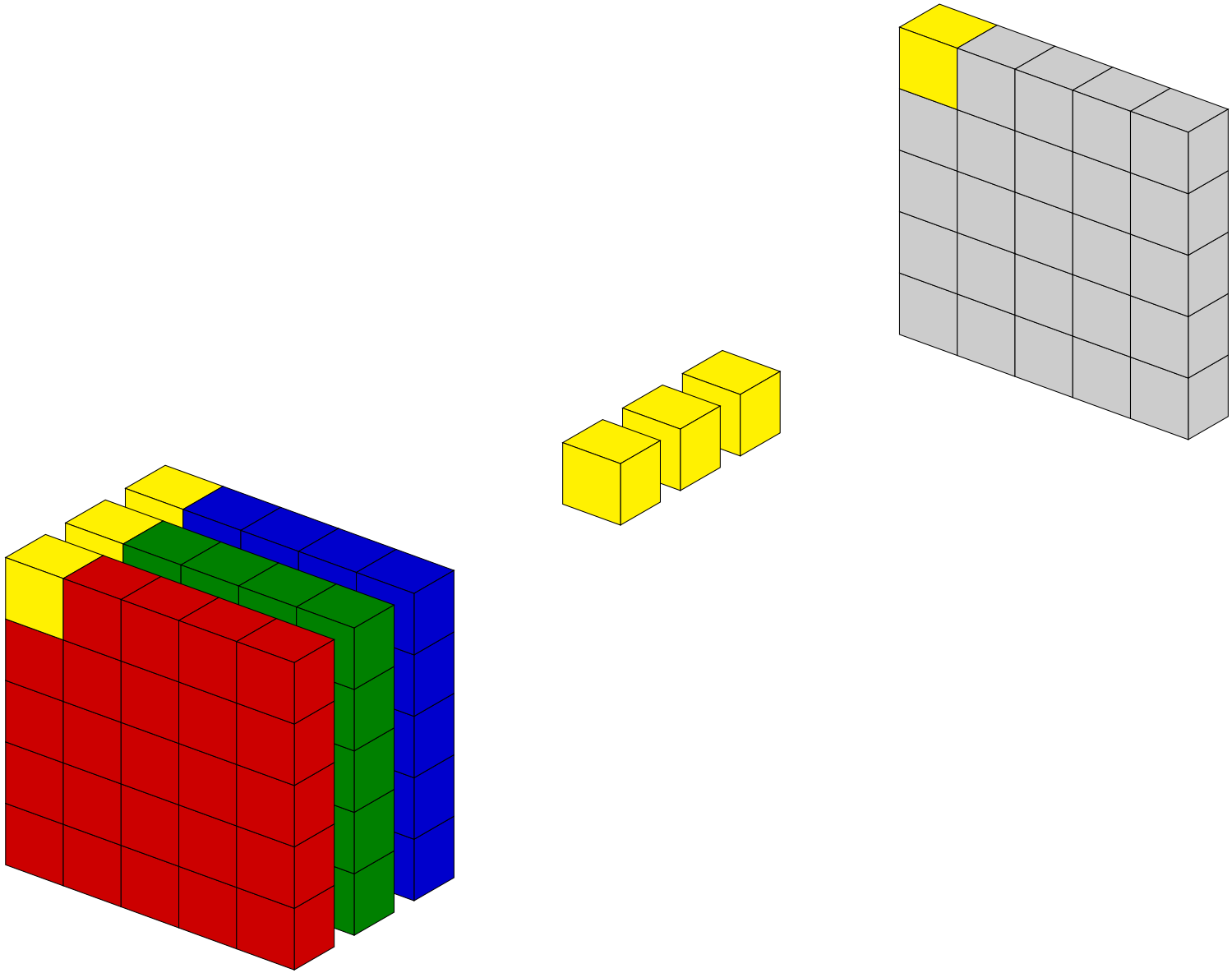
(1×1) convolution



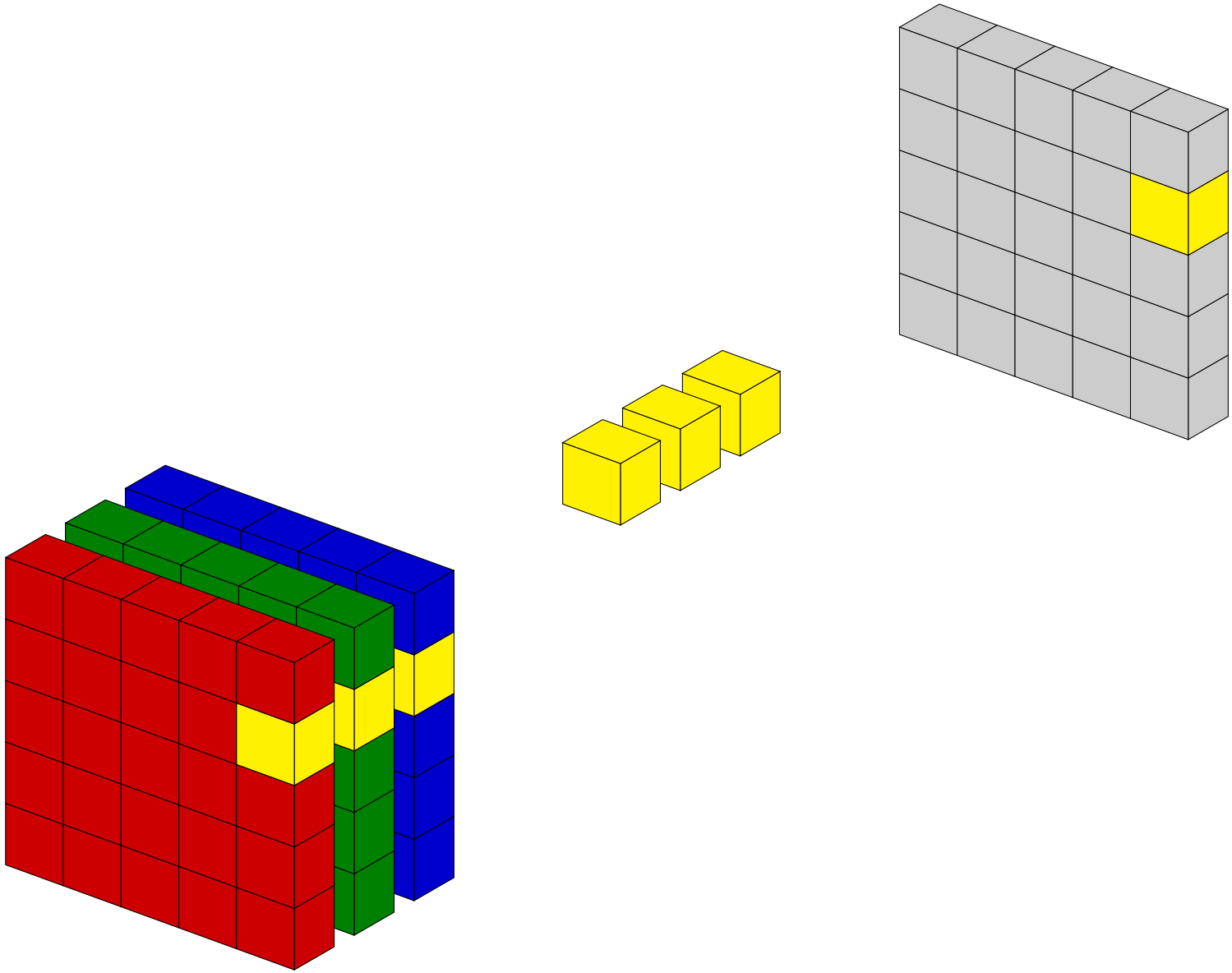
(1×1) convolution



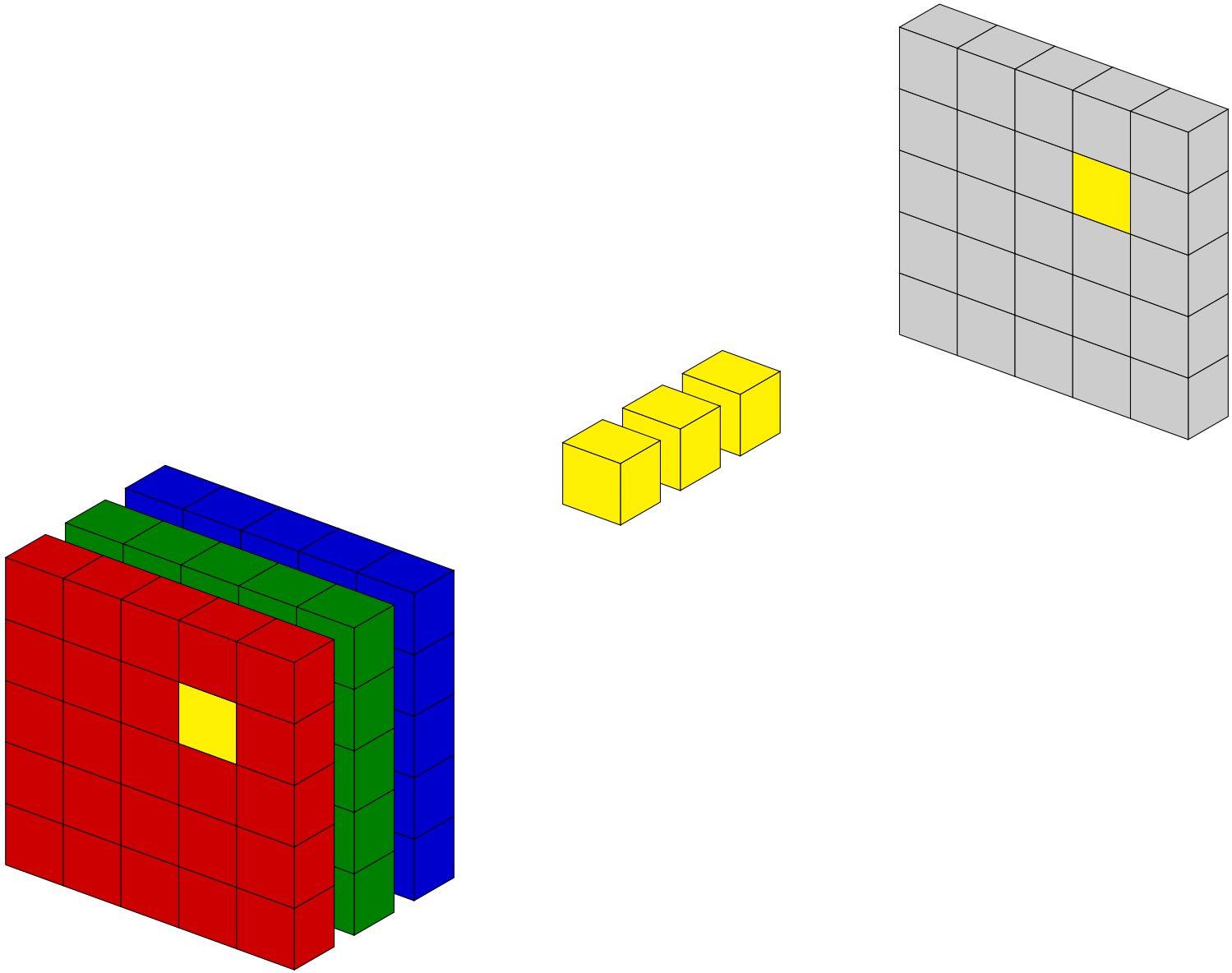
(1×1) convolution



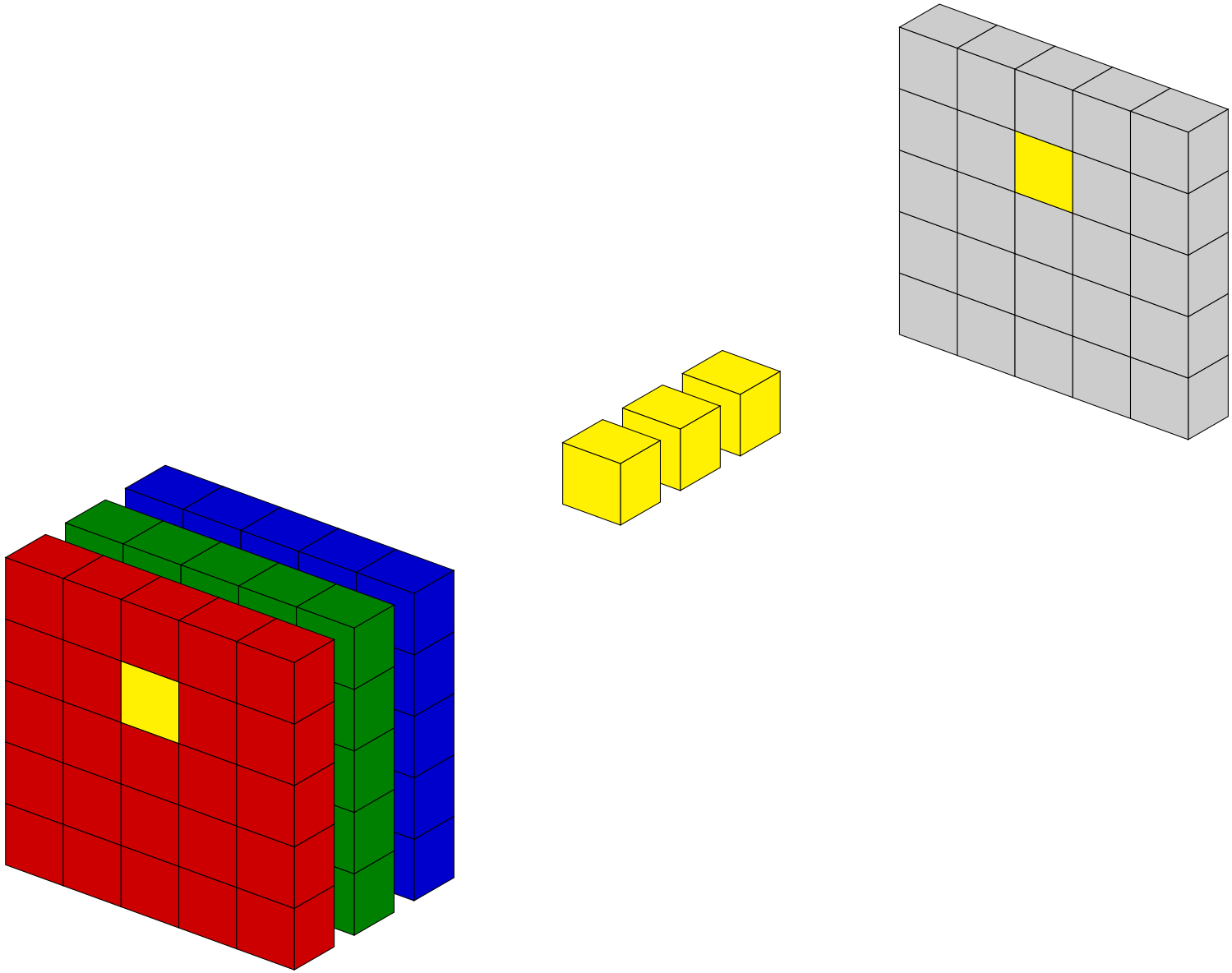
(1×1) convolution



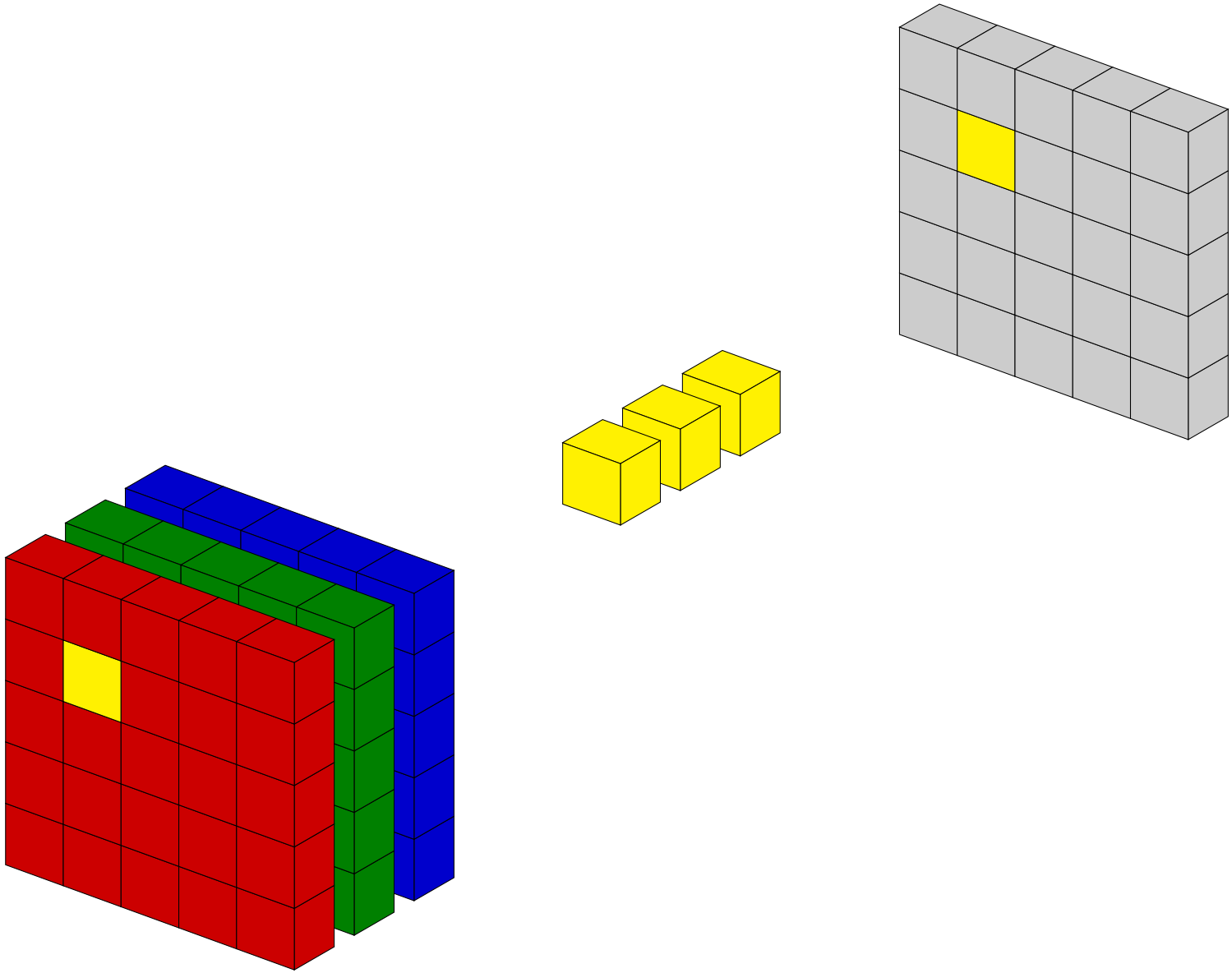
(1×1) convolution



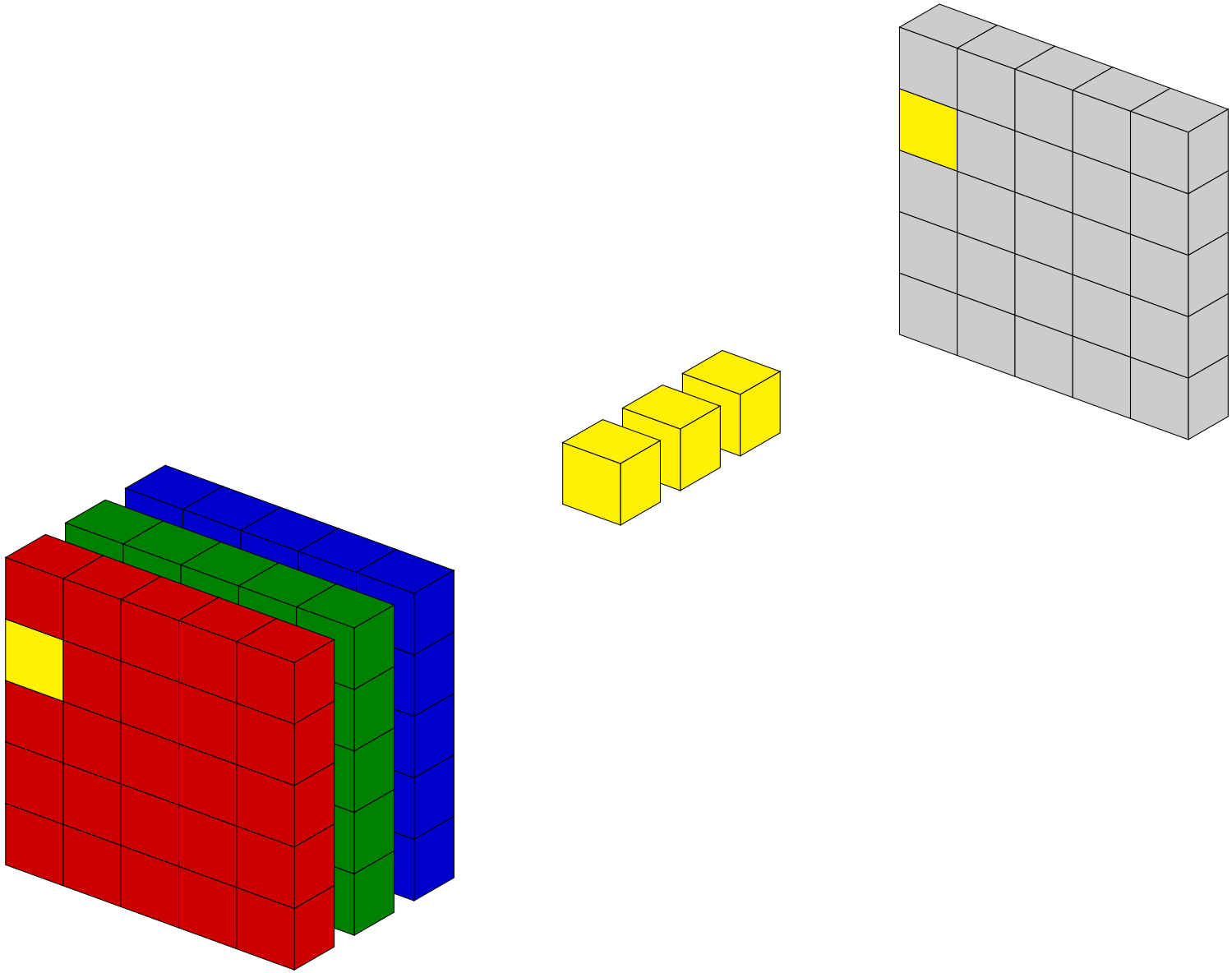
(1×1) convolution



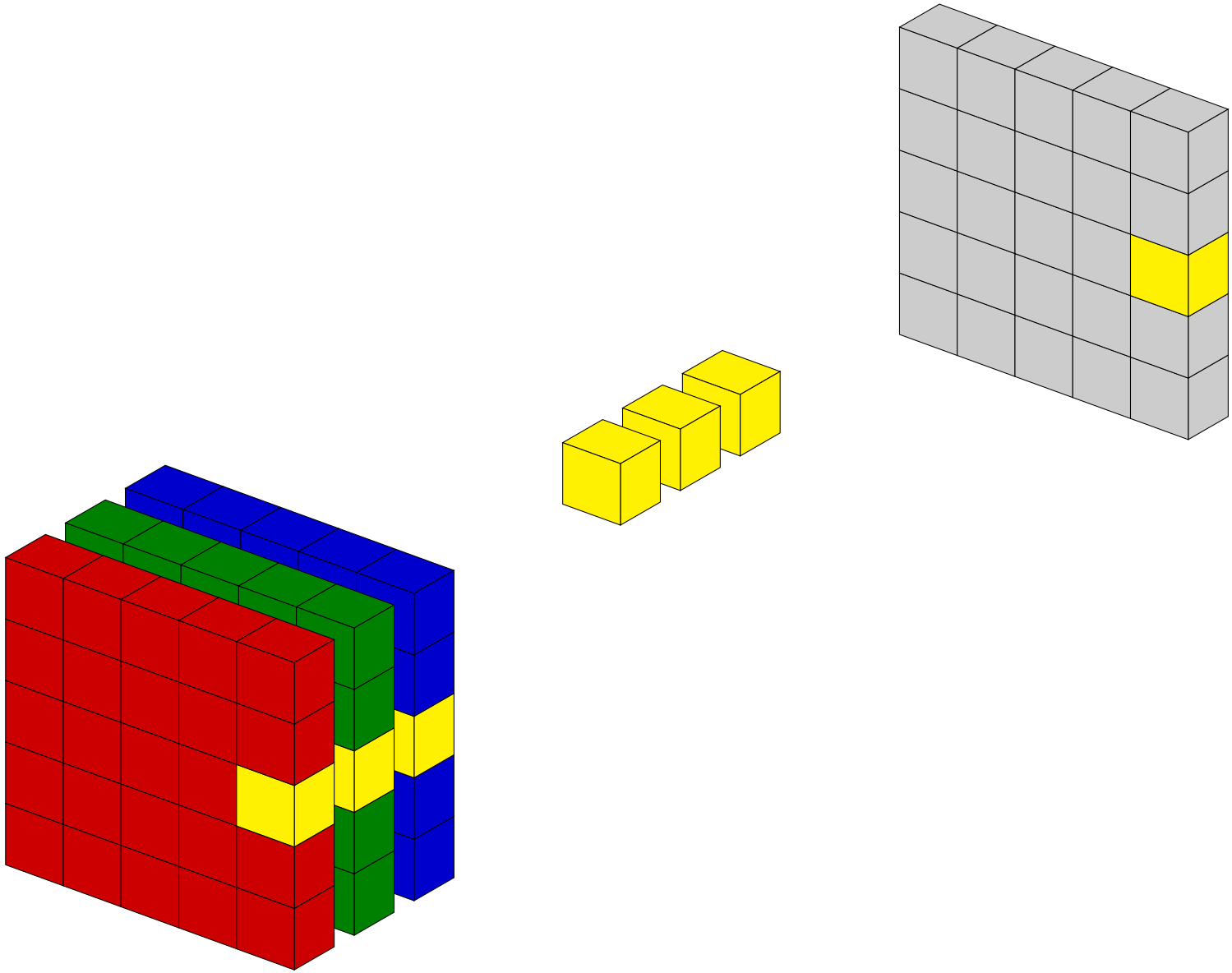
(1×1) convolution



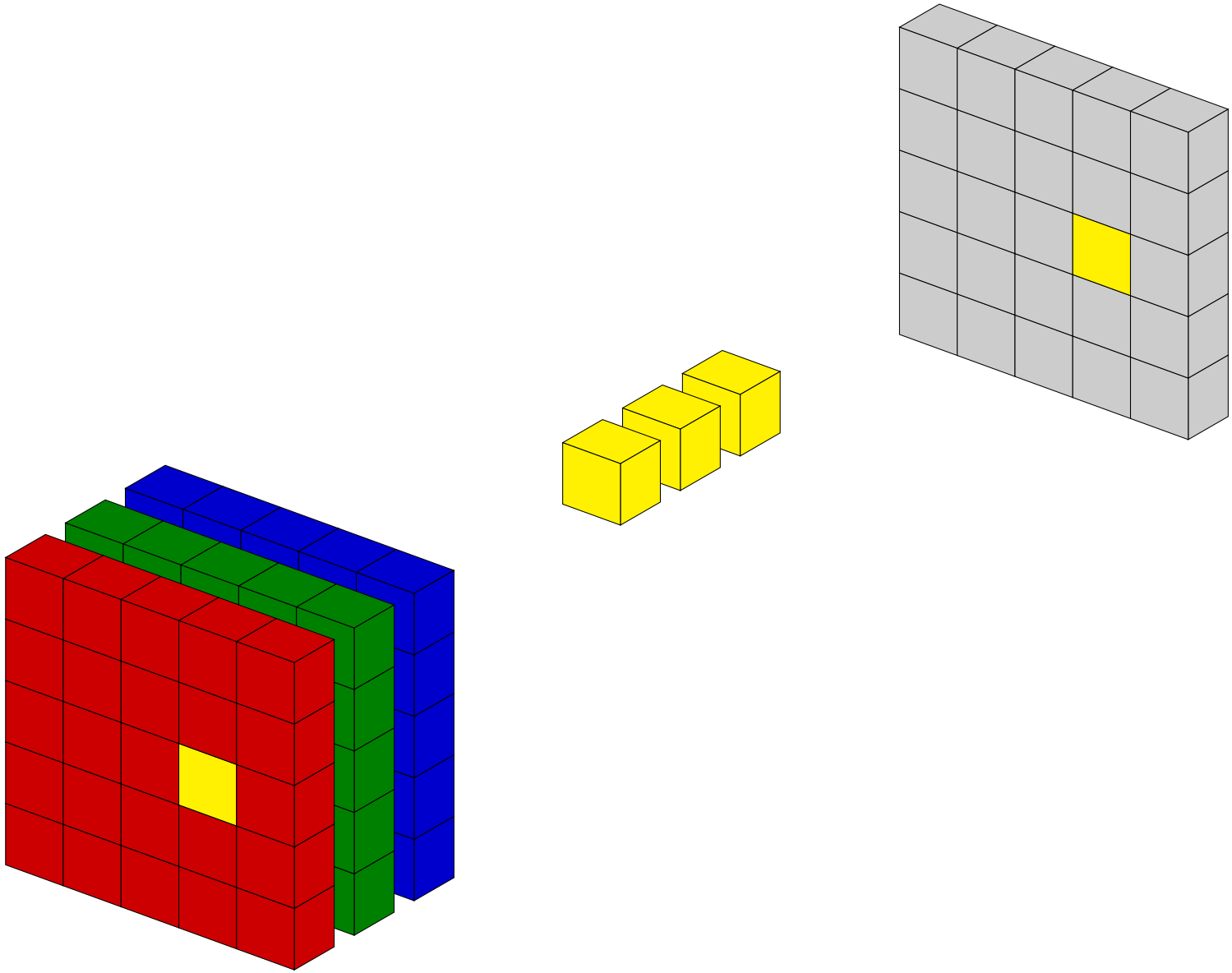
(1×1) convolution



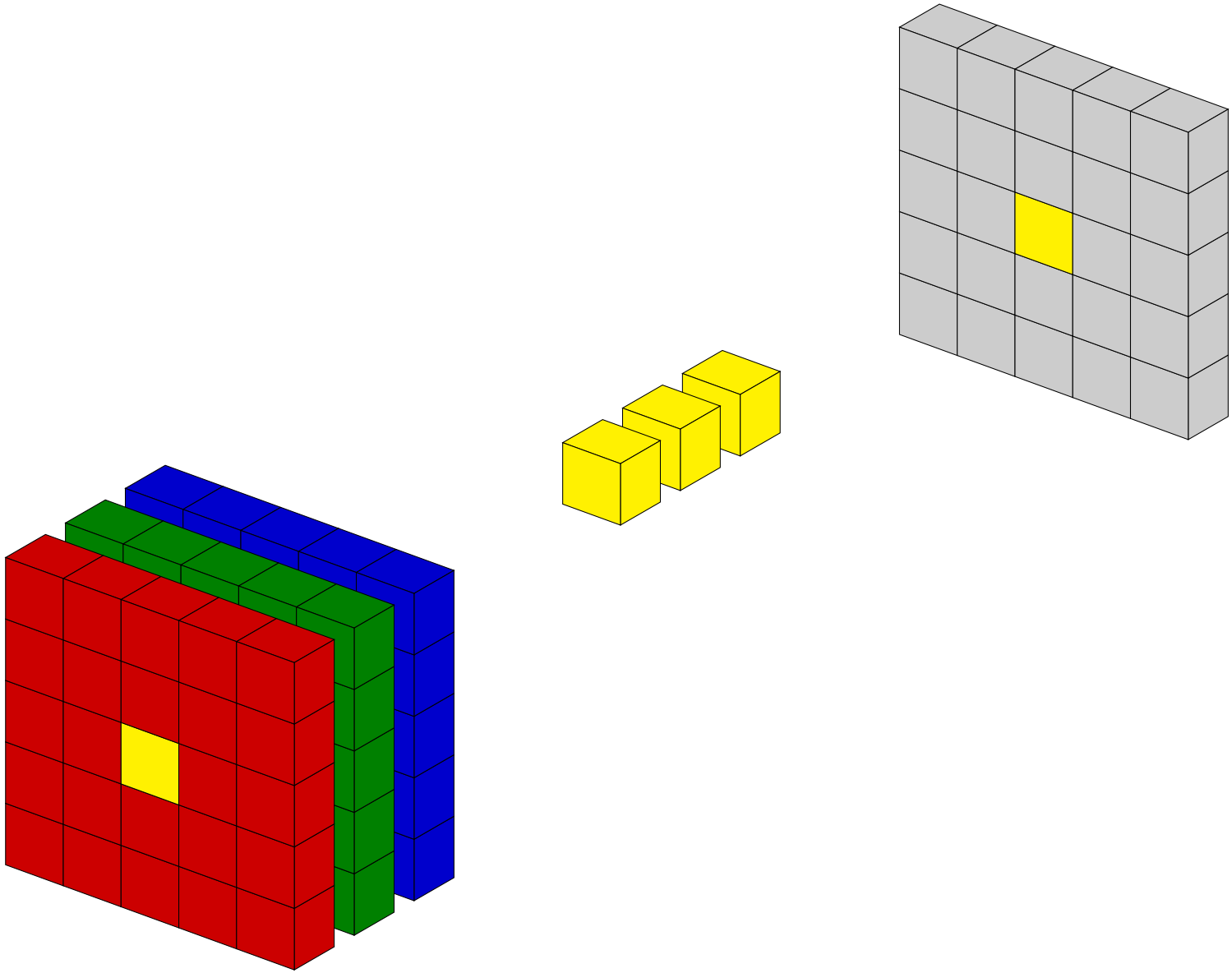
(1×1) convolution



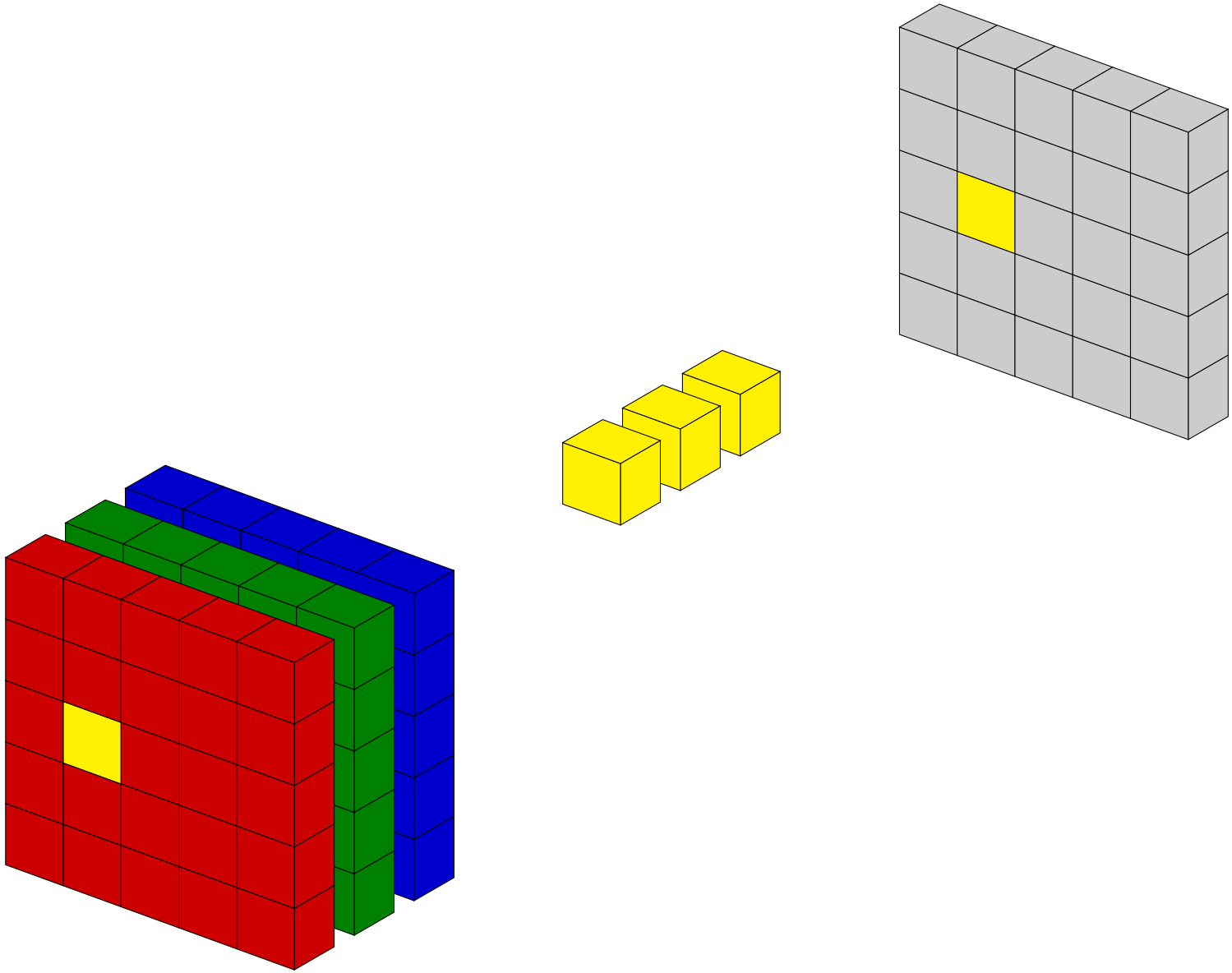
(1×1) convolution



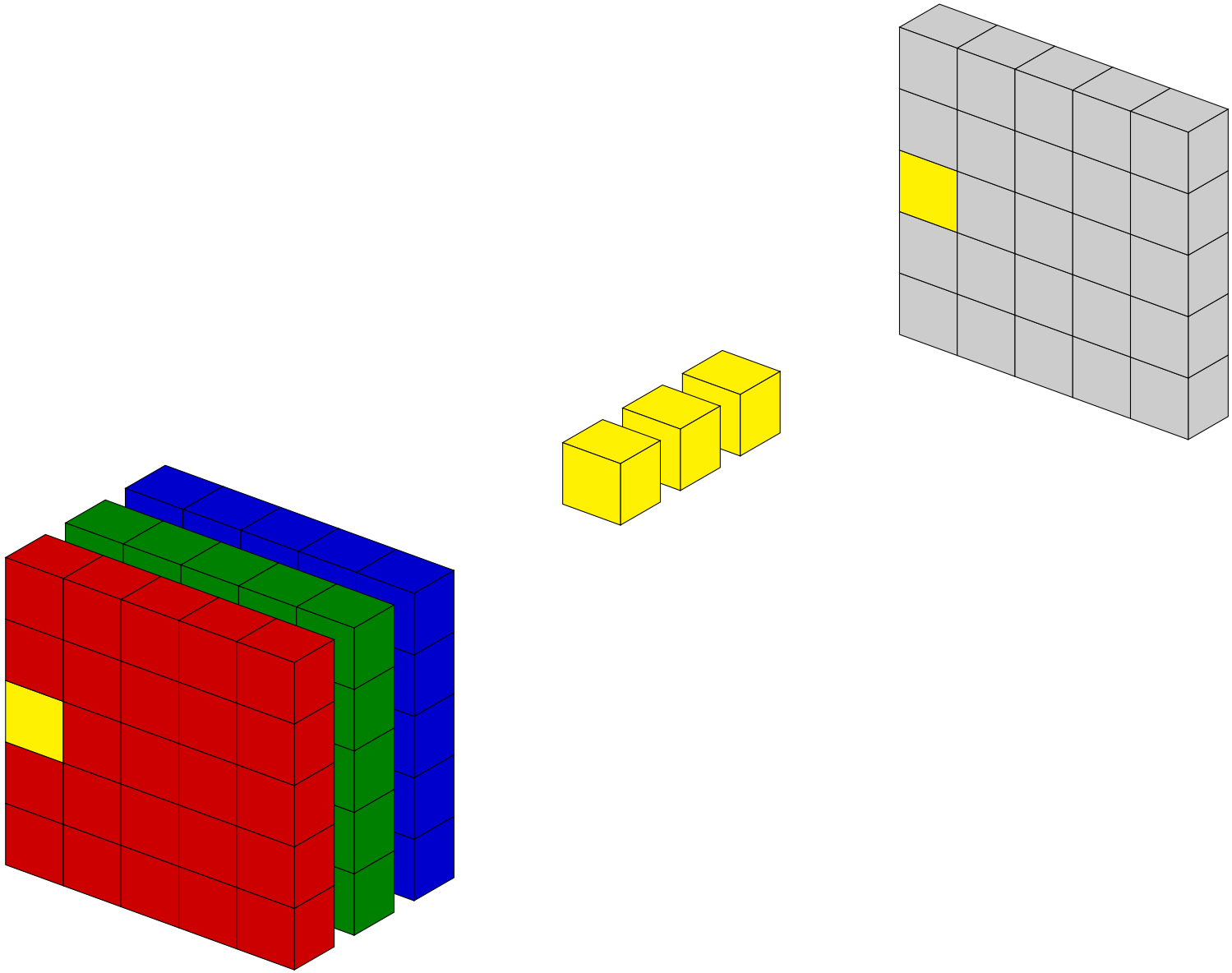
(1×1) convolution



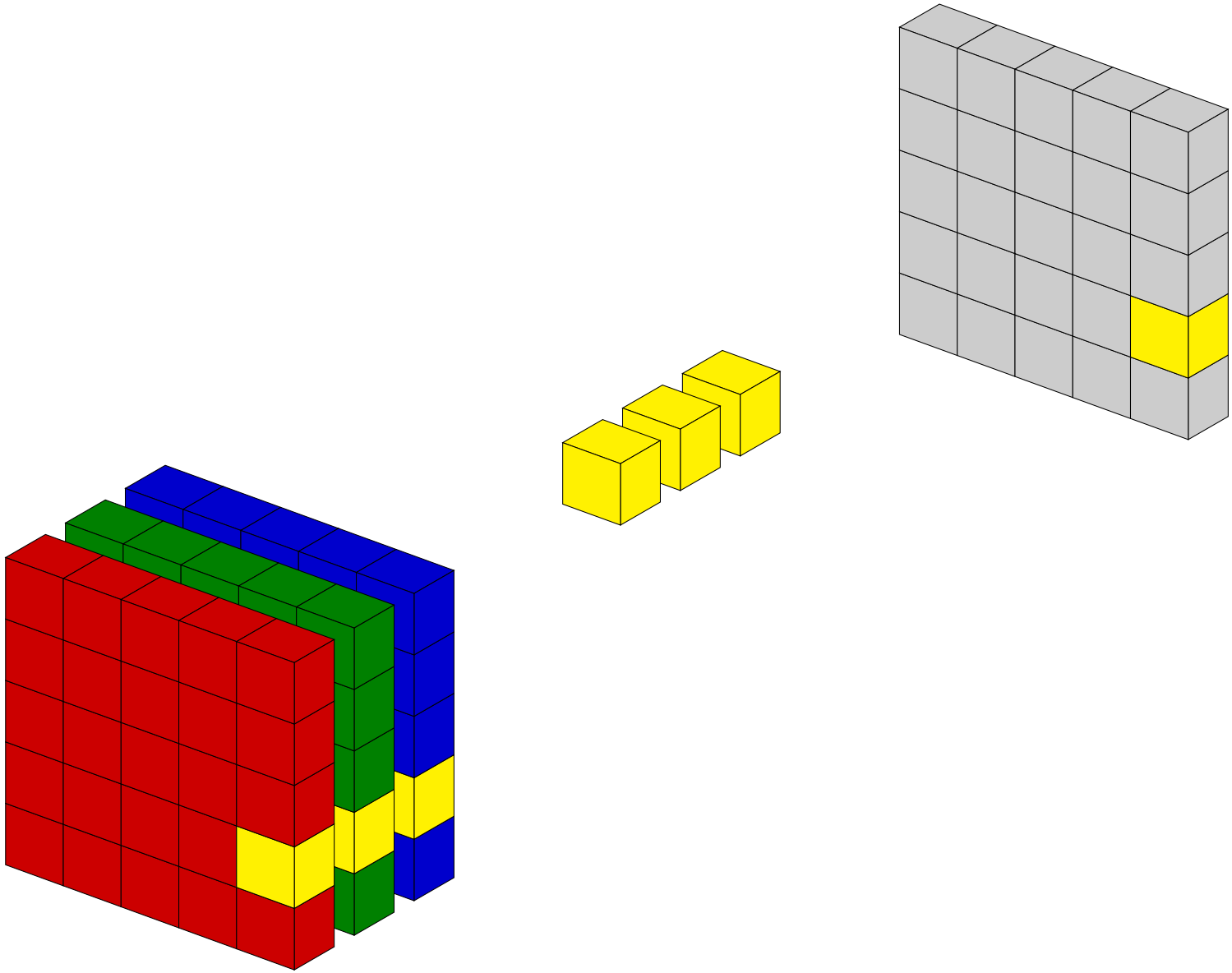
(1×1) convolution



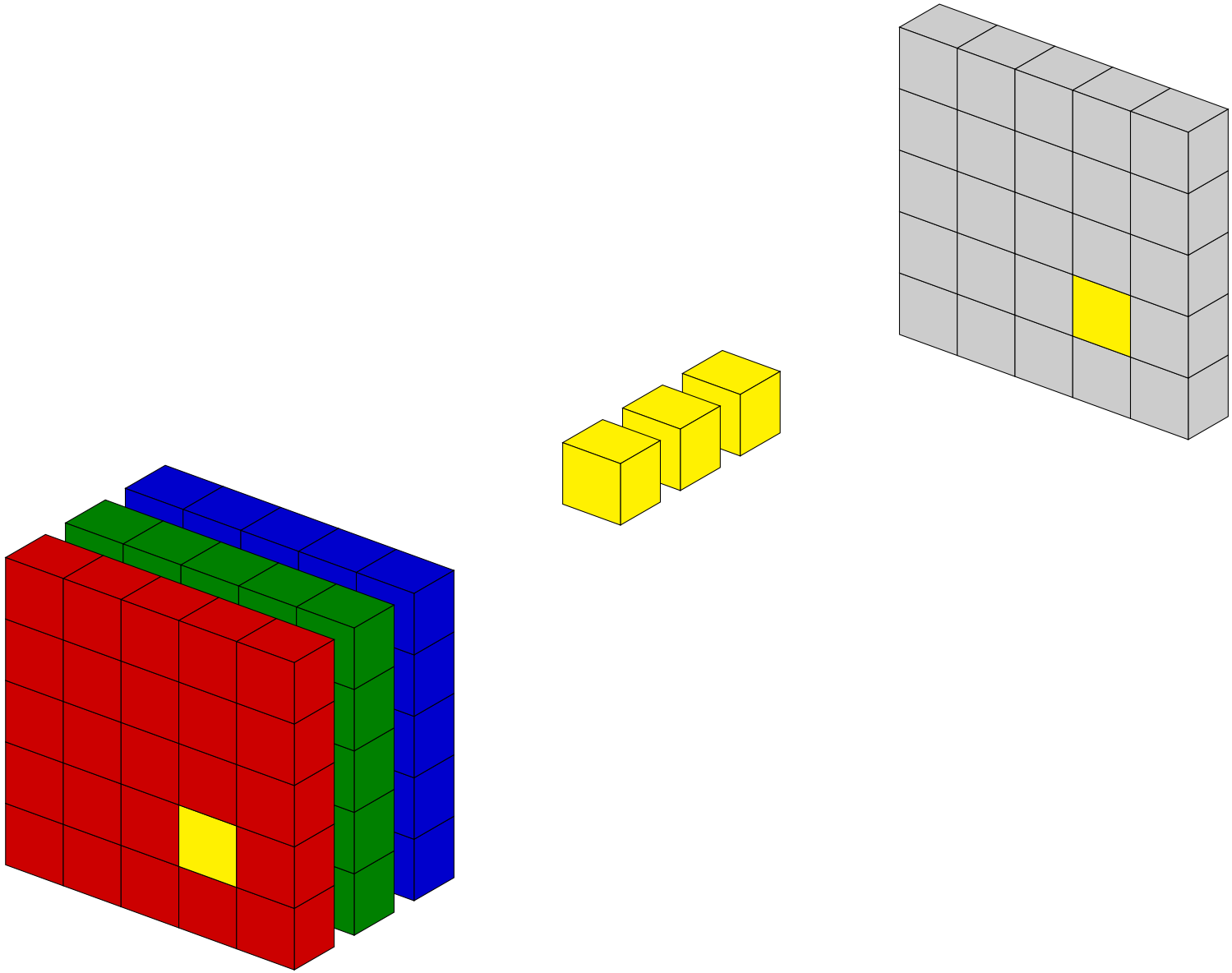
(1×1) convolution



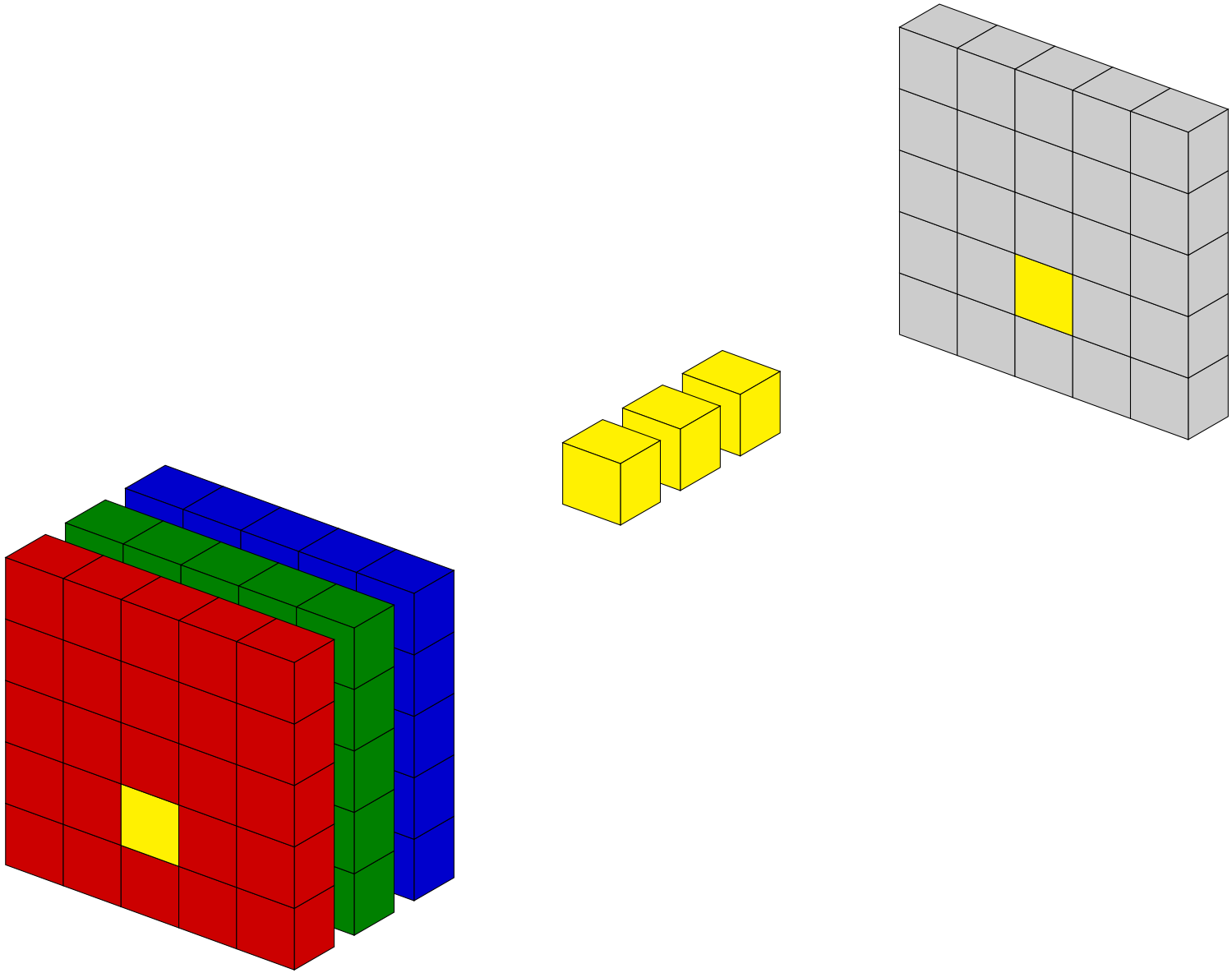
(1×1) convolution



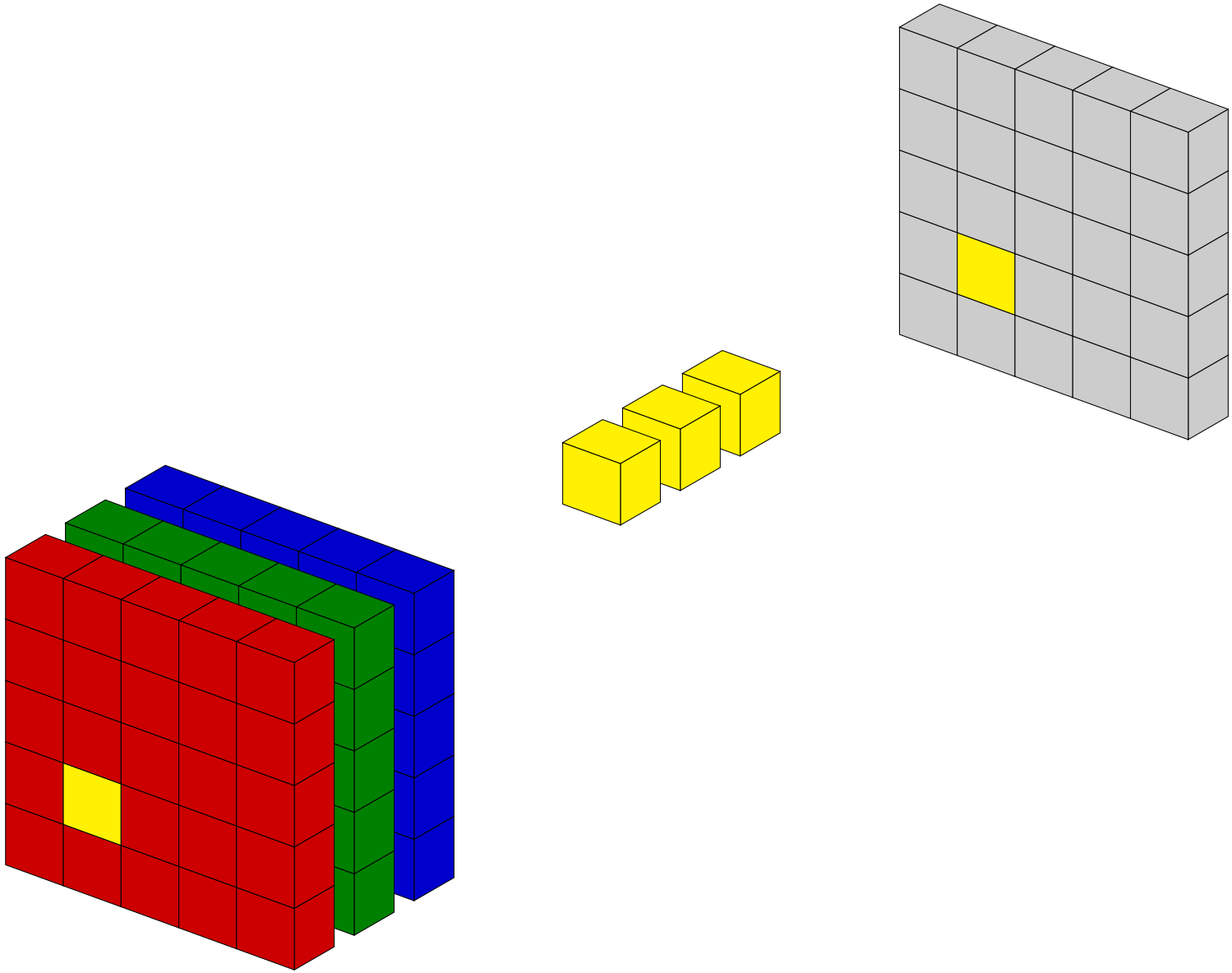
(1×1) convolution



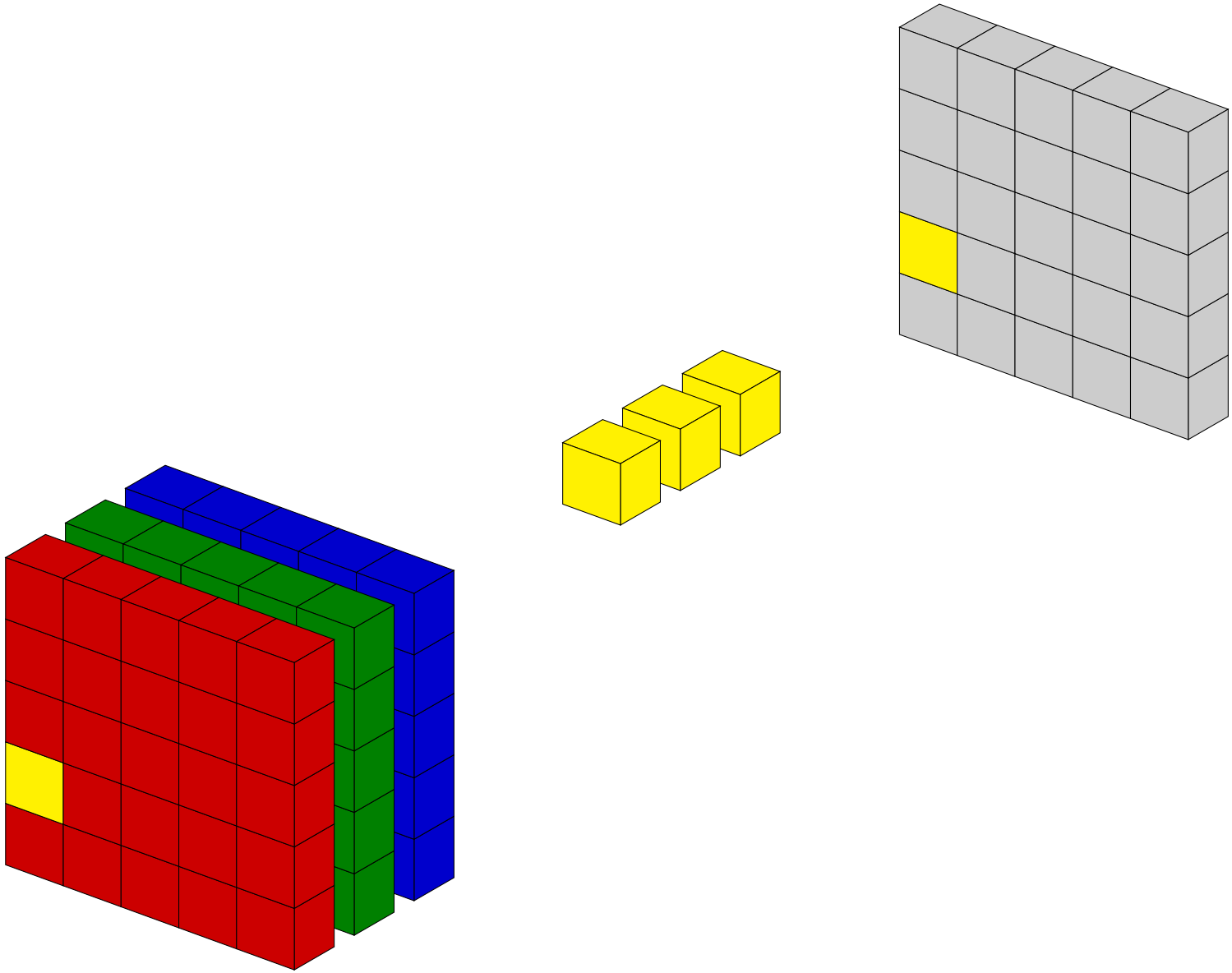
(1×1) convolution



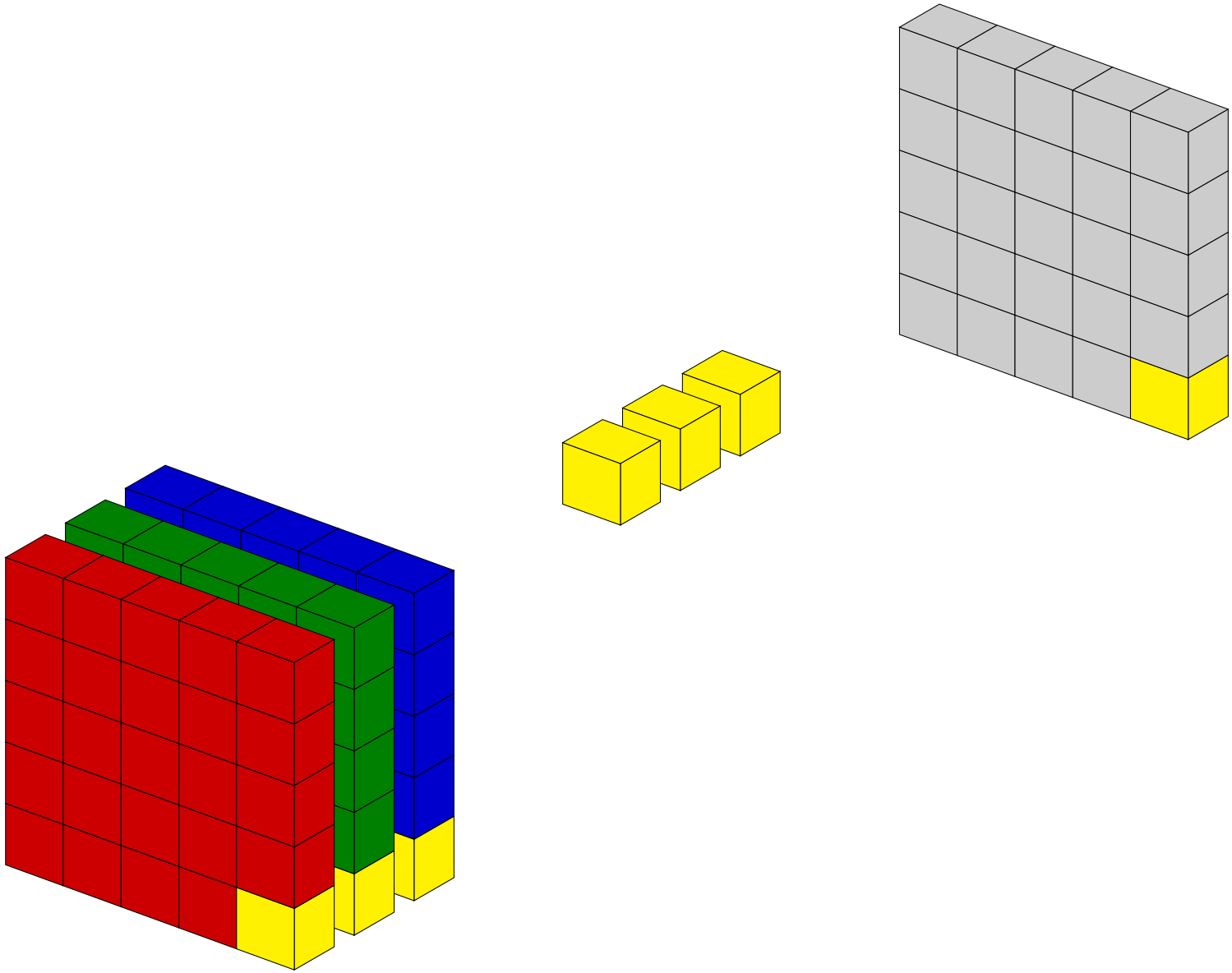
(1×1) convolution



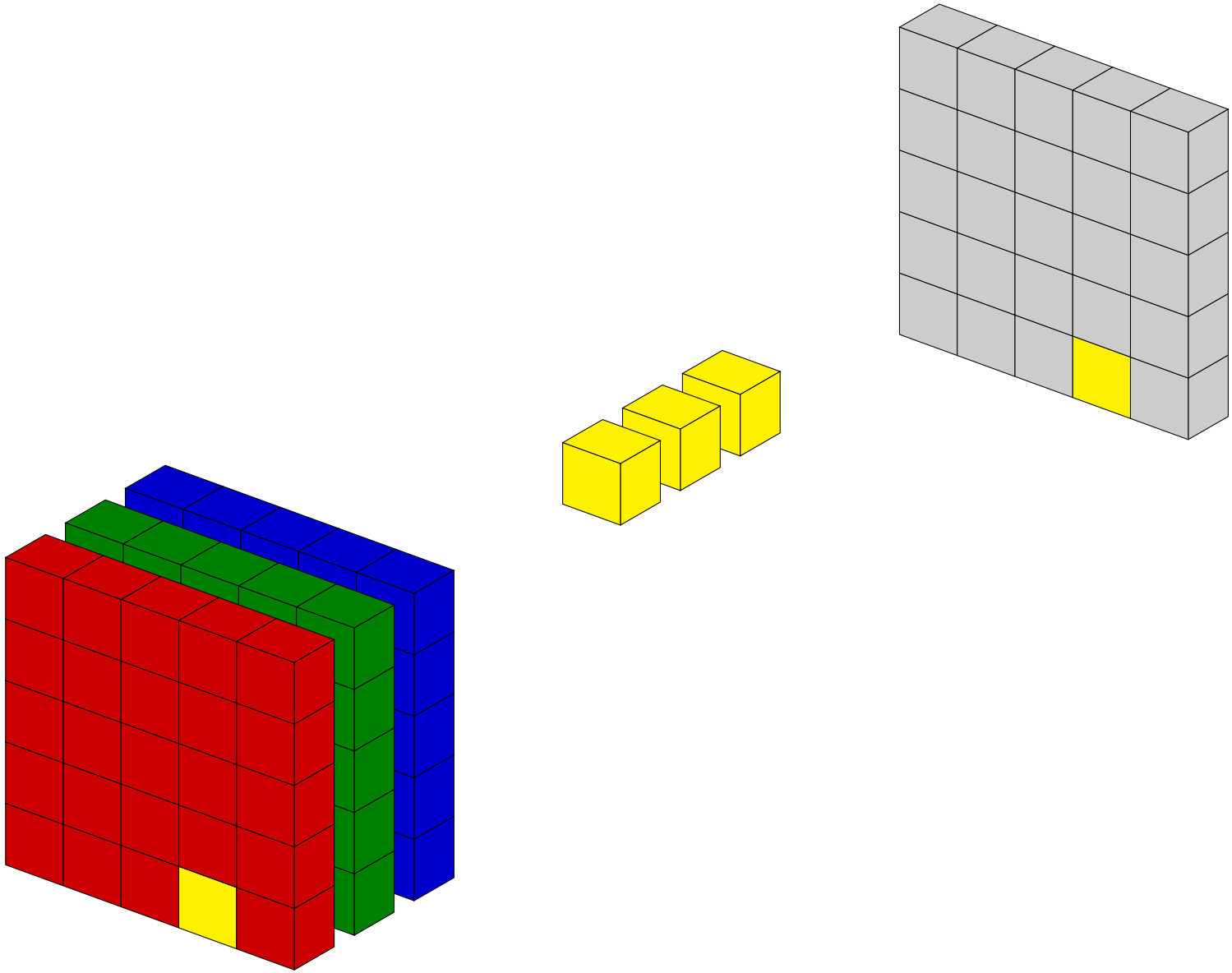
(1×1) convolution



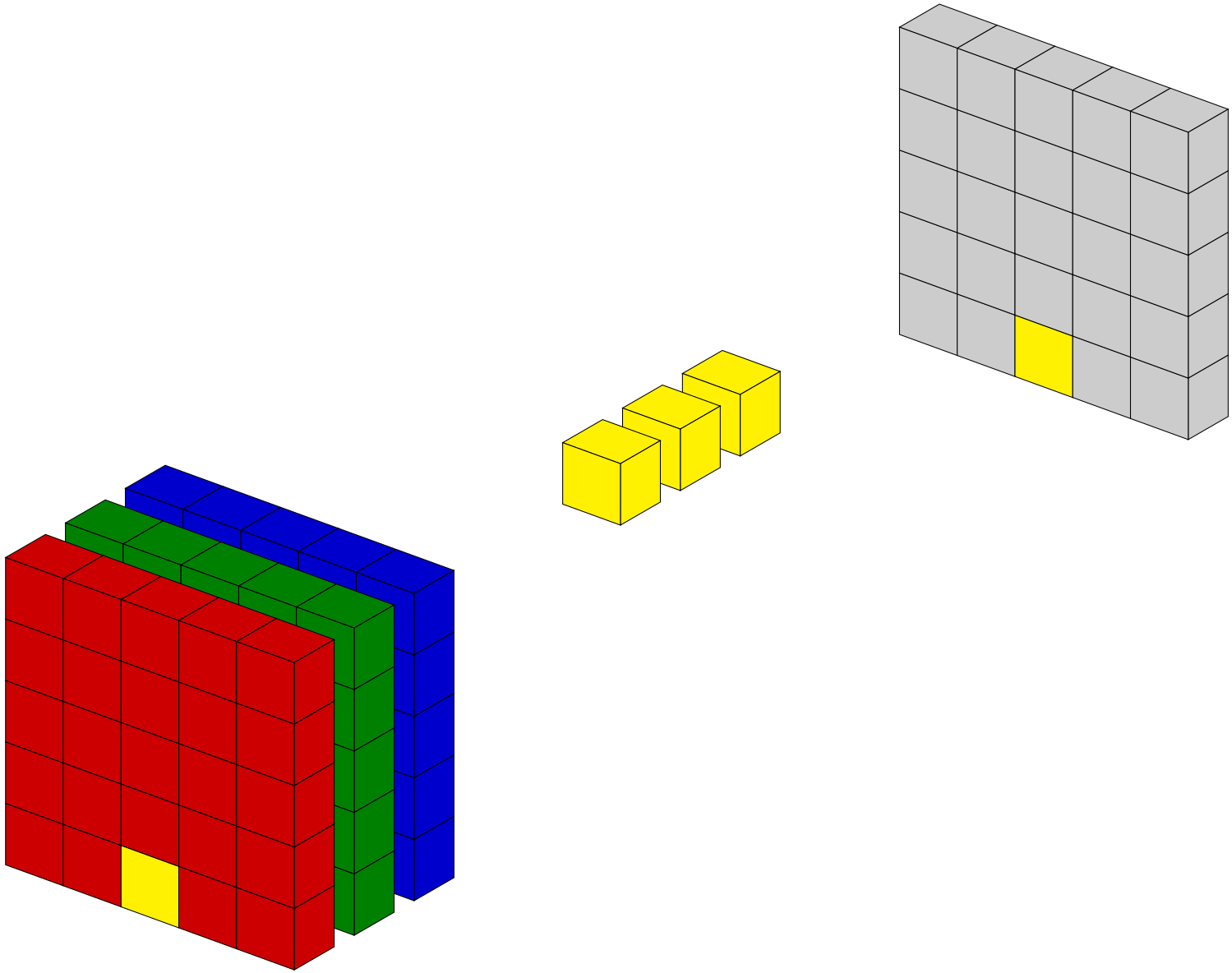
(1×1) convolution



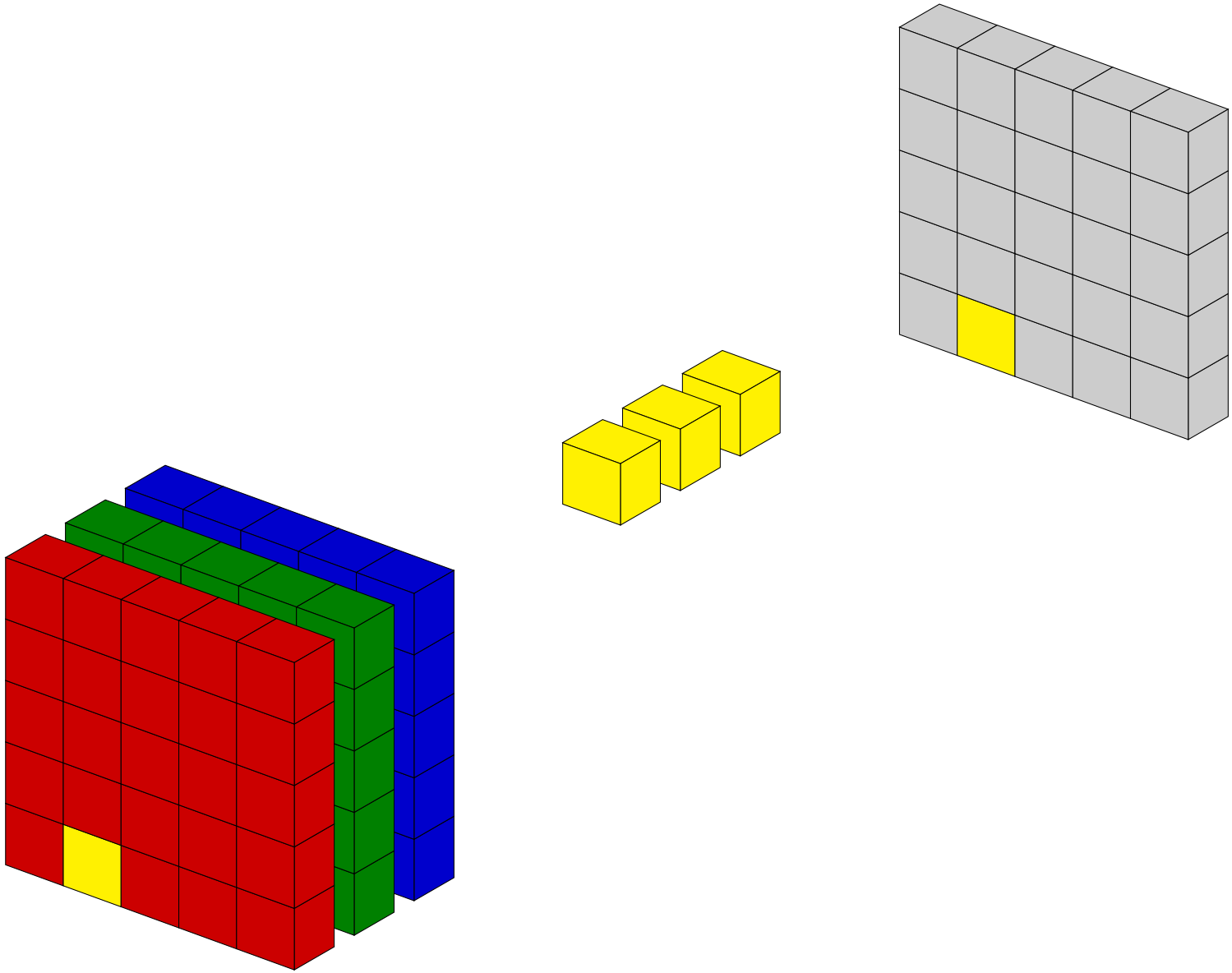
(1×1) convolution



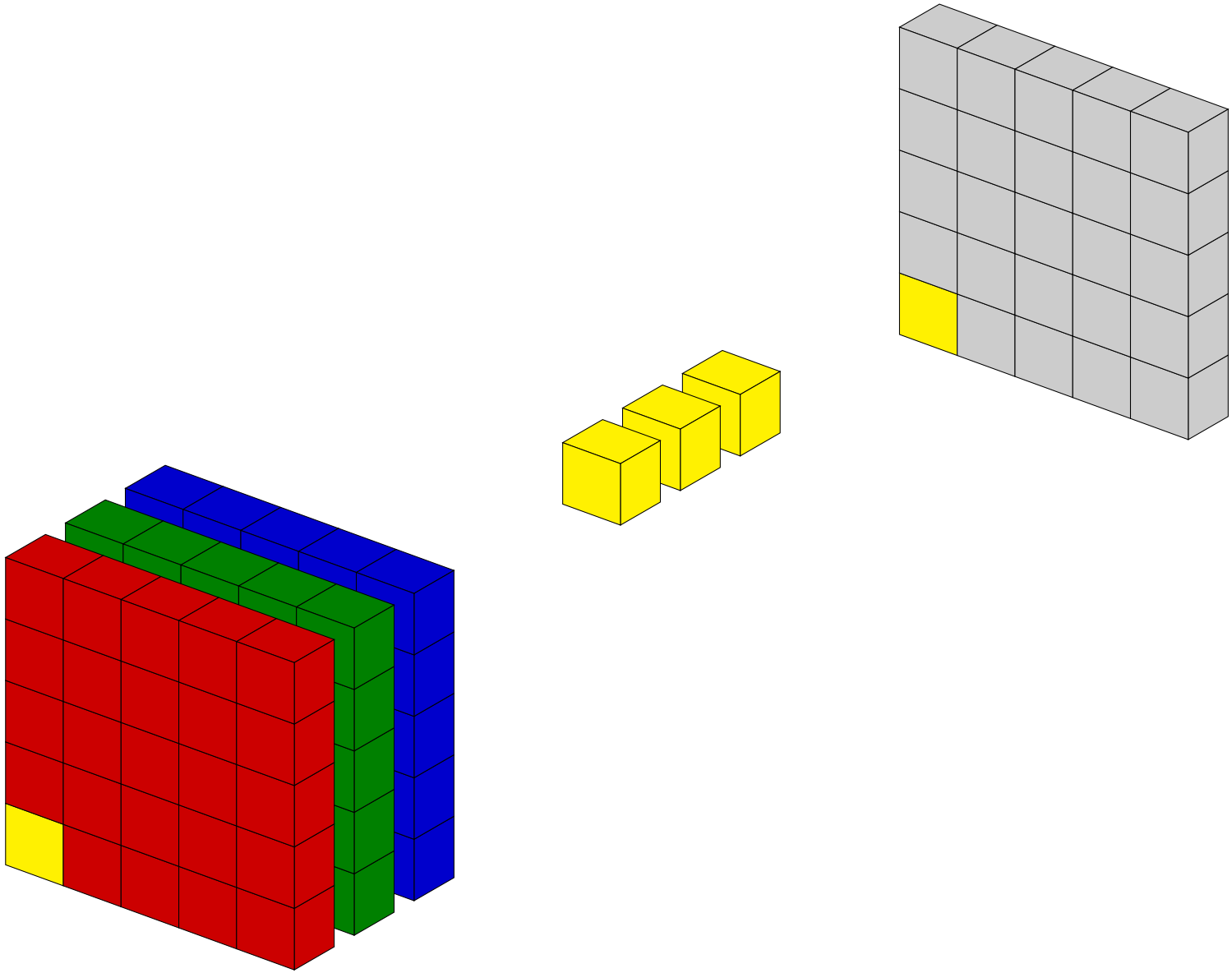
(1×1) convolution



(1×1) convolution

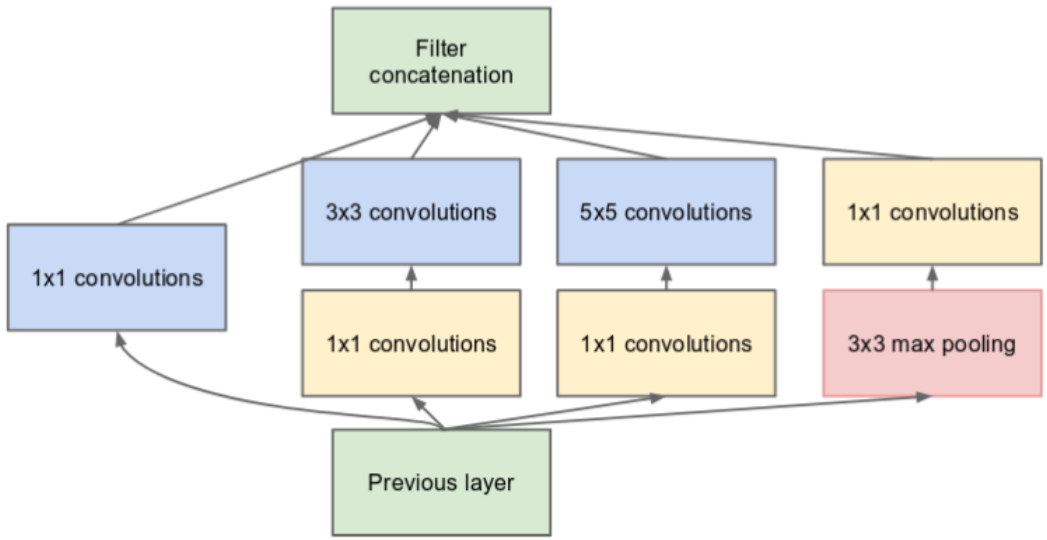


(1×1) convolution



Inception V1

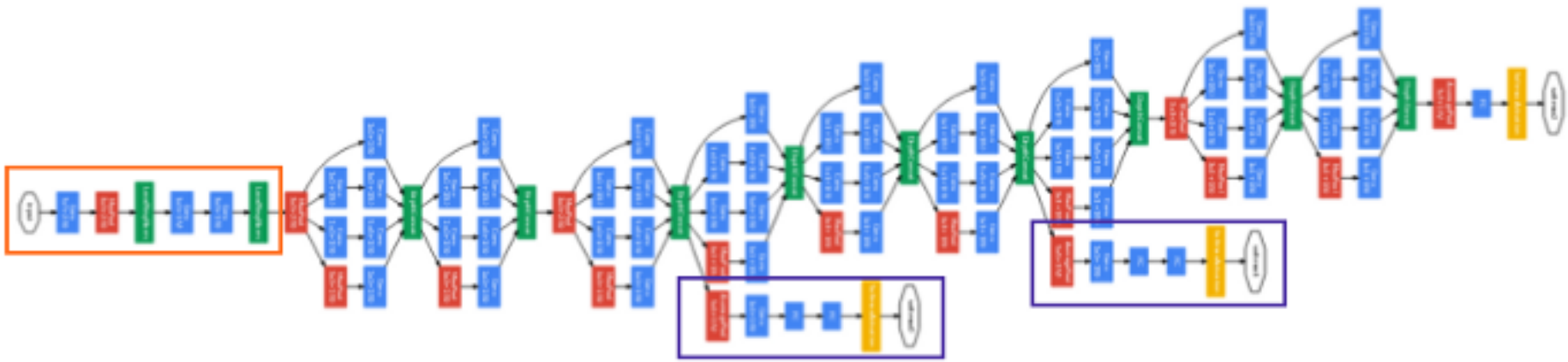
Basic block



Less parameters with intermediate (1×1) convolutions

Inception networks

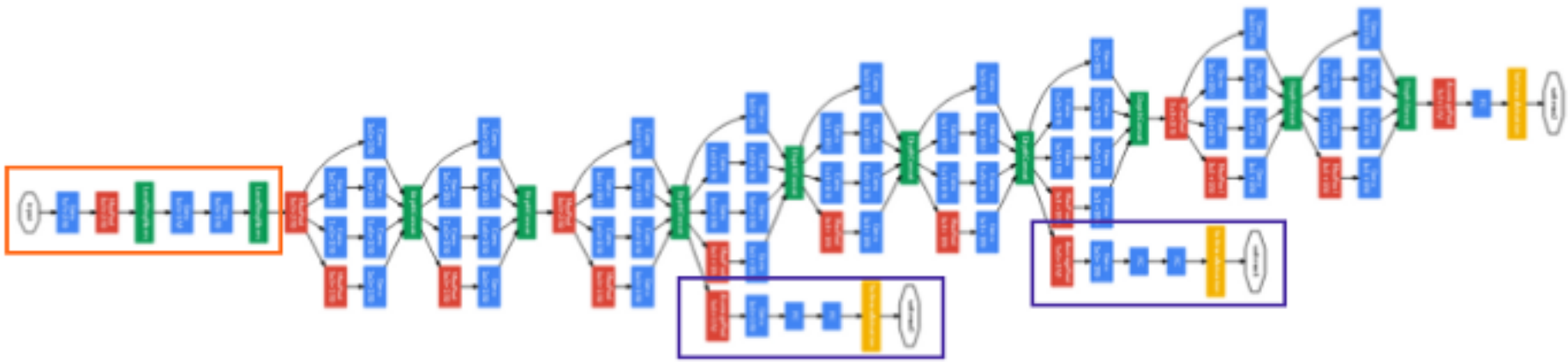
Full network



Several outputs at different depths

Inception networks

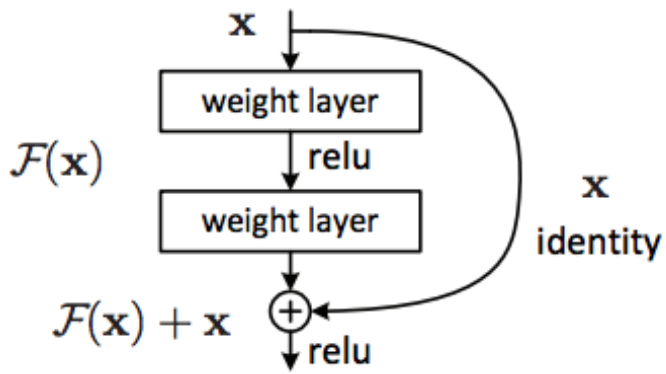
Full network



Several outputs at different depths

ResNet (He et al., 2016)

Residual networks



Basic block

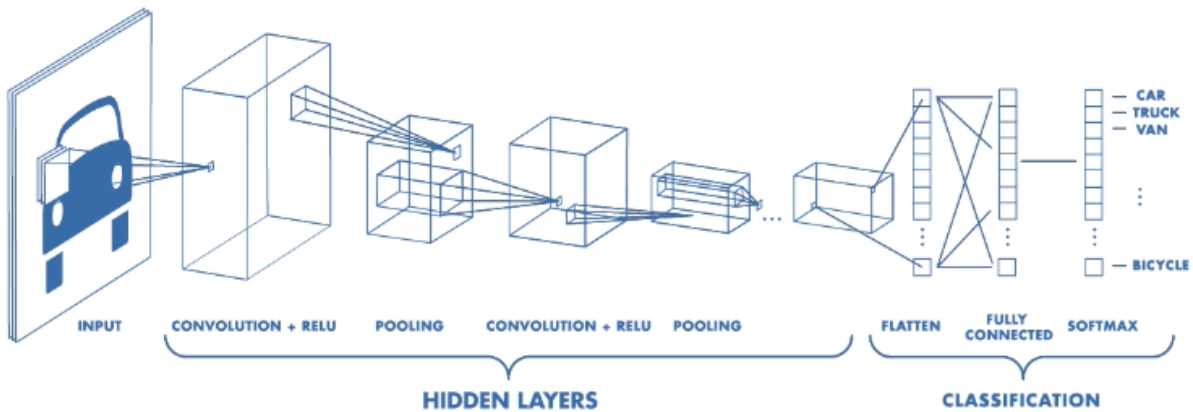
ResNet (He et al., 2016)

Residual networks

Full network

Transfer learning

- Idea: Use very deep networks already trained on huge data sets (e.g ImageNet) to do a new specific task on a new data set
- Use the pre-trained weights as initial weights the specific task
- The weights of the convolutional layers can also be frozen.
- To optimize the training, one can pre-compute the feature maps before FC layers, on the entire data set and only train the weights of the FC layers on this new transformed data set.



Source: Photo via FloydHub.com

References

Bach, F., R. Jenatton, J. Mairal, and G. Obozinski (2012). Optimization with sparsity-inducing penalties. *Foundations and Trends® in Machine Learning 4*(1), 1–106.

Chapelle, O., J. Weston, L. Bottou, and V. Vapnik (2001). Vicinal risk minimization. *Advances in neural information processing systems*, 416–422.

Glorot, X. and Y. Bengio (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings.

He, K., X. Zhang, S. Ren, and J. Sun (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Ioffe, S. and C. Szegedy (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems 25*, 1097–1105.

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*(11), 2278–2324.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research 15*(56), 1929–1958.

Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9.

Vapnik, V. (2013). *The nature of statistical learning theory*. Springer science & business media.