Convolutional Neural Networks 1

Xiaolong Wang

Last Class

- Multi-layer Neural Networks

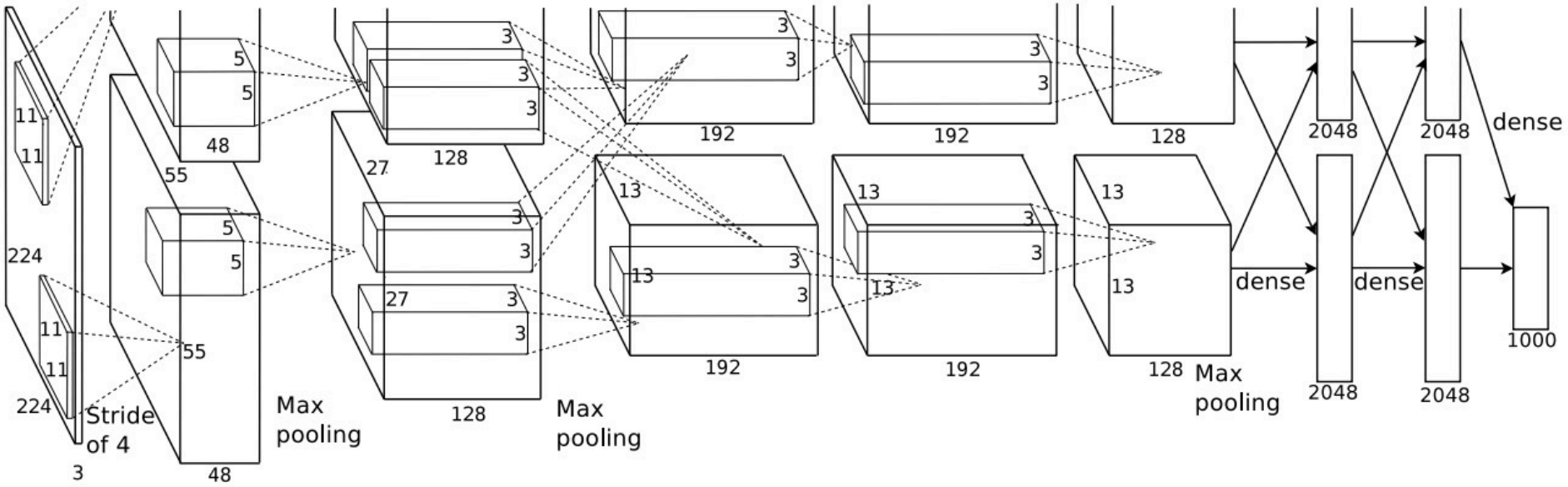
- Training Neural Networks with back-propagation

This Class

- Convolutional Operation
- Convolutional Neural Networks
- Different Elements in Training ConvNets

Slides partially from: http://cs231n.stanford.edu/, https://slazebni.cs.illinois.edu/fall20/

Convolutional Neural Networks

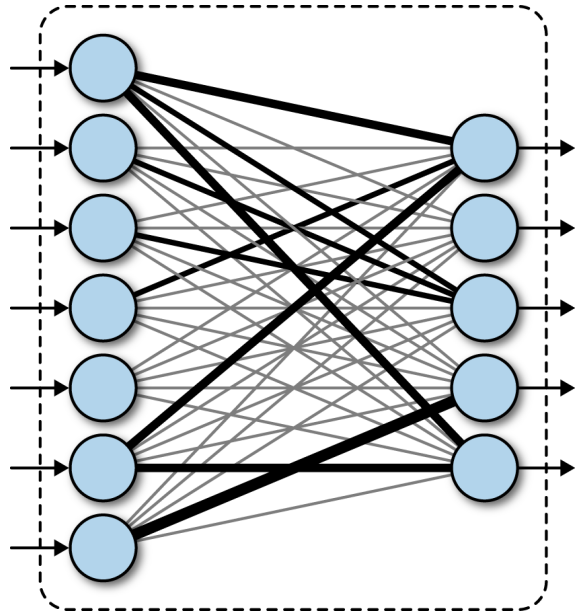


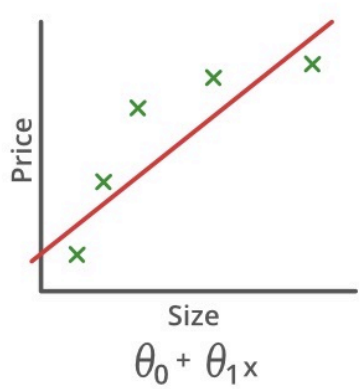
AlexNet (Krizhevsky et al. 2012)

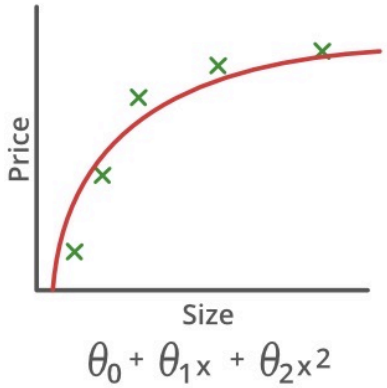
Structural Prior in Images

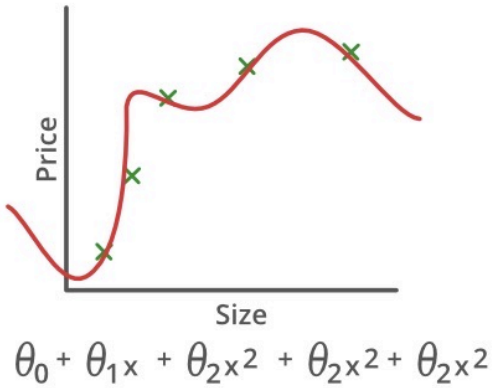
- Disadvantage of MLP:
  - Large number of parameters
  - First layer: (32 x 32 x 3) x 128 = 393,216

- Why is large number of parameters bad?
  - Overfitting



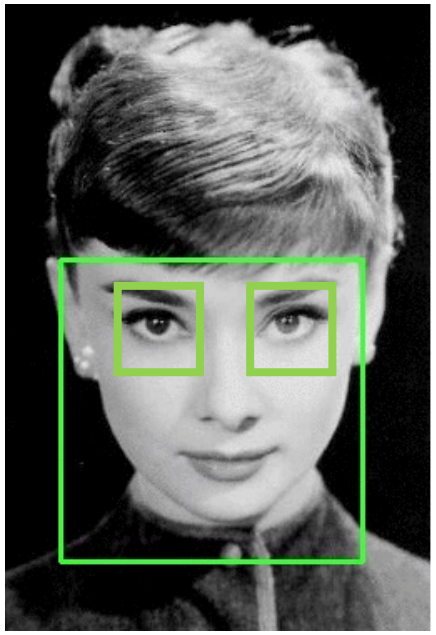




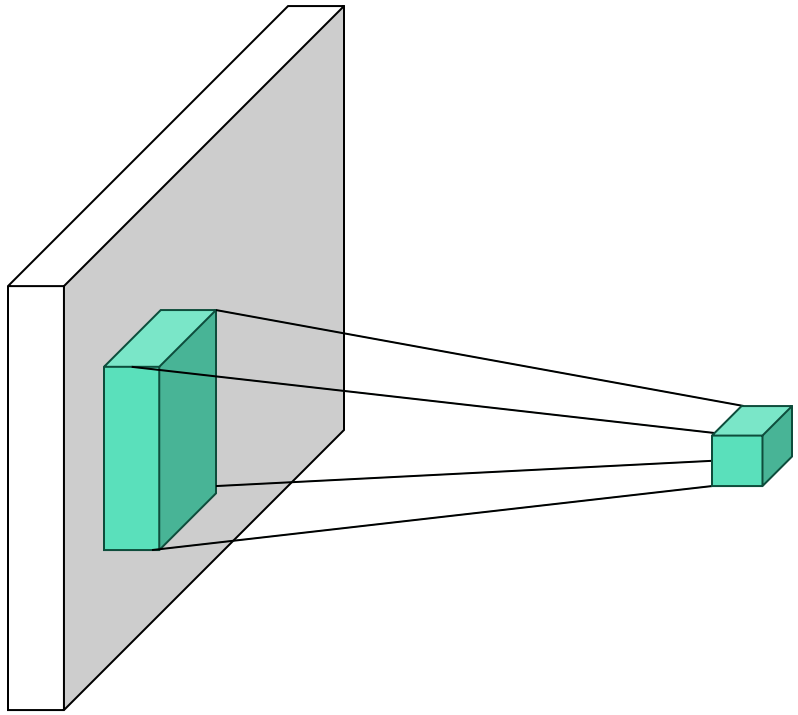


Structural Prior in Images

- ConvNets use the structural prior in images:
  - There are repetitive patterns in images
  - We should re-use and share the filter across the whole image
  - Reduce parameters, avoid overfitting



Convolution



image

Given a 3×3 filter

Compute the response at location (k, l)

$$z(k,l) = \sum_{i,j=-1}^{1} W(i,j)\, x(k+i, l+j)$$

Convolution

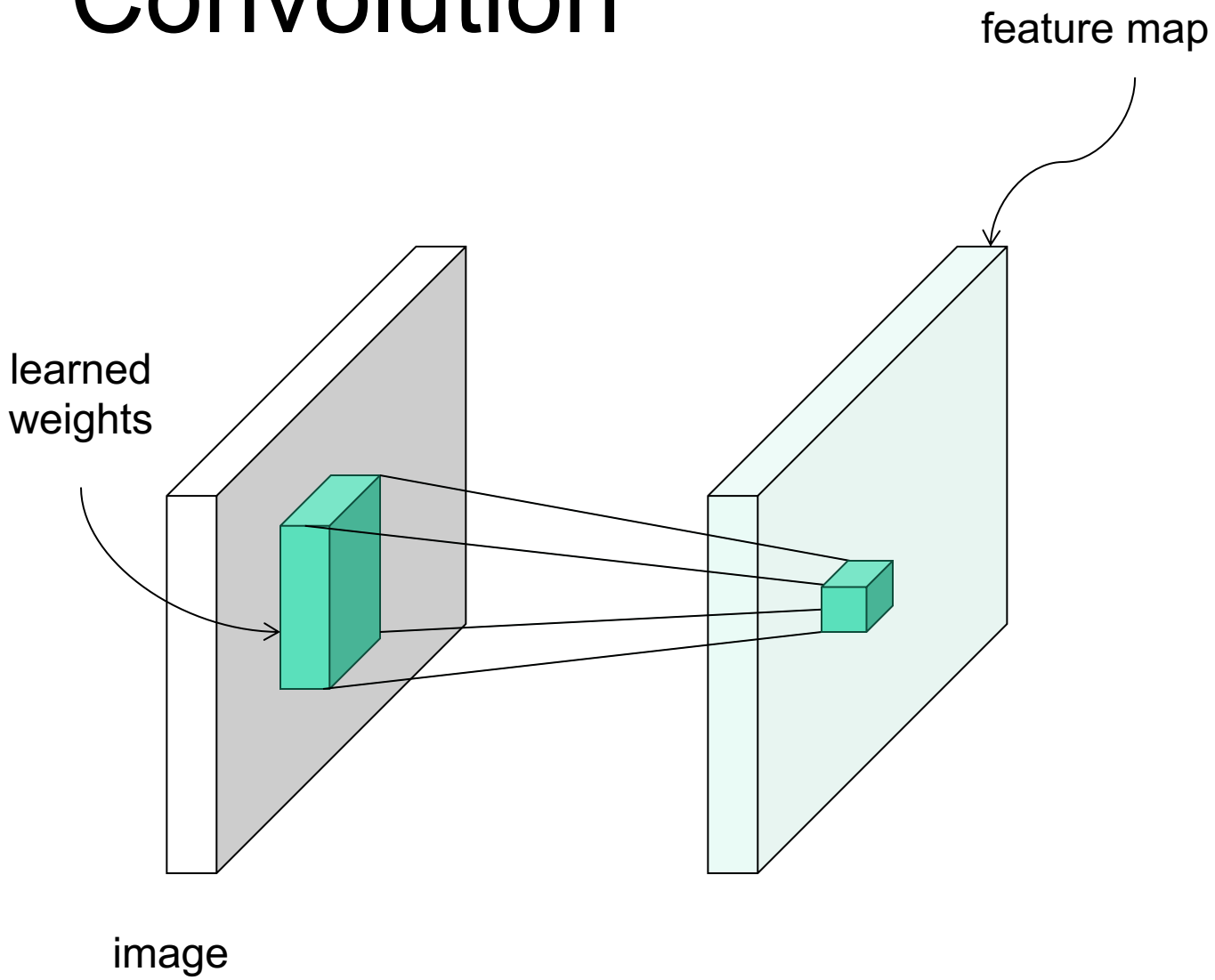
0	1	2	2	2	0	1
1	0	2	1	2	0	2
2	1	0	2	0	0	1
1	0	2	1	2	0	2
0	1	2	2	2	0	1
1	0	2	1	2	0	2
2	1	0	2	0	0	1

$*$

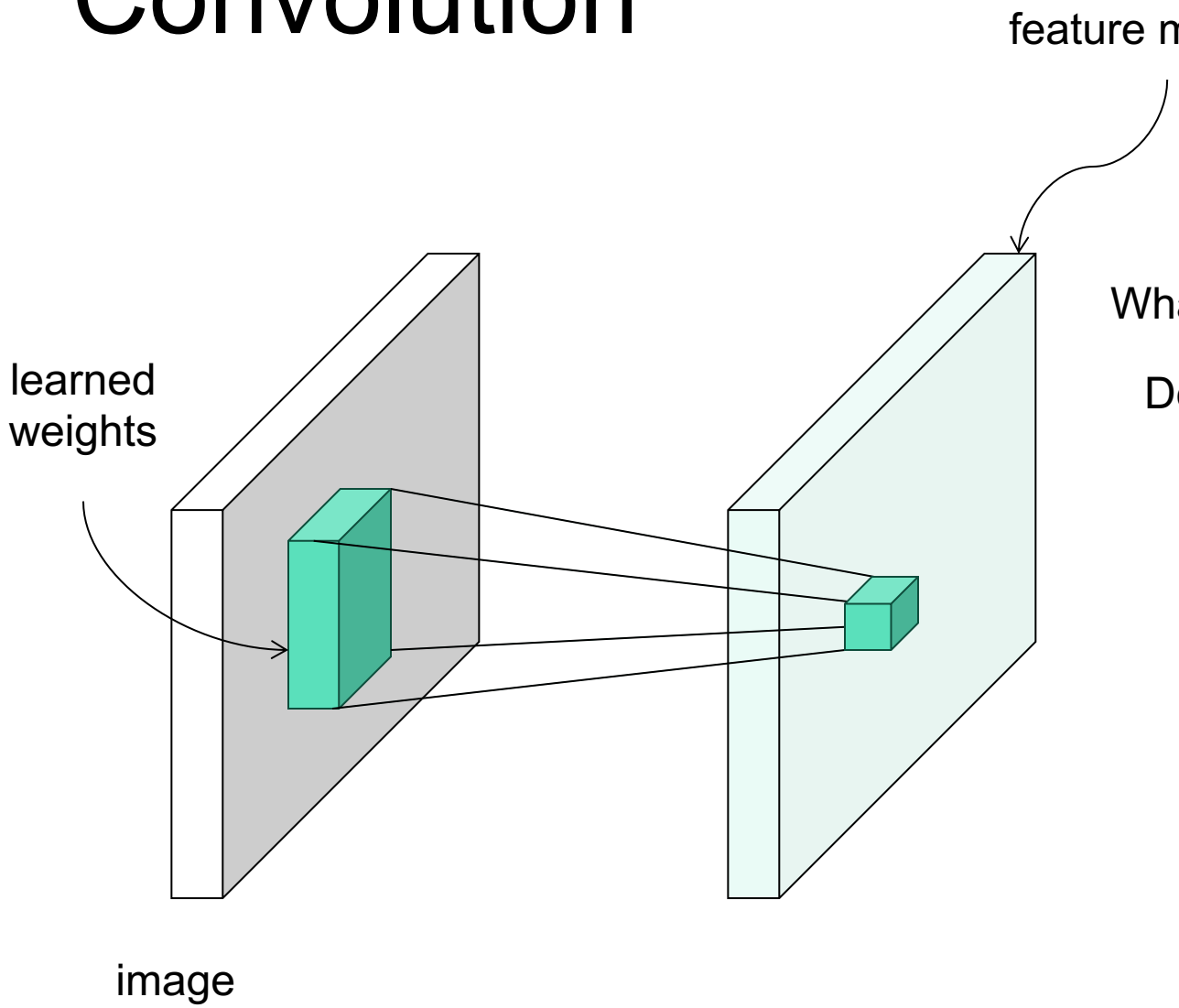
1	0	-1
0	-1	1
-1	1	0

$$0 \times 1 + 1 \times 0 + 2 \times (-1) + 1 \times 0 + 0 \times (-1) + 2 \times 1 + 2 \times (-1) + 1 \times 1 + 0 \times 0 = -1$$

Convolution

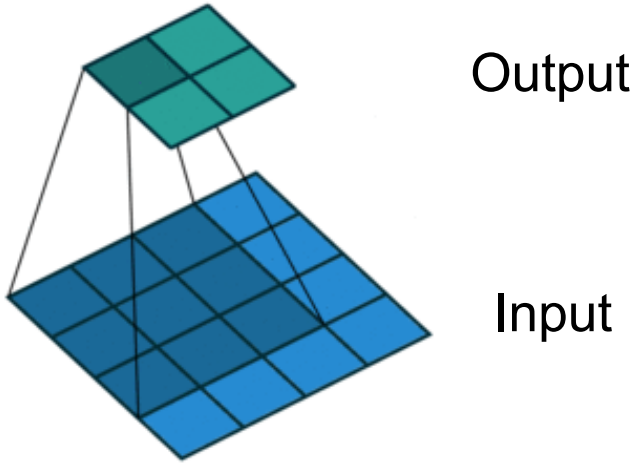


Convolution



What is the feature map resolution?

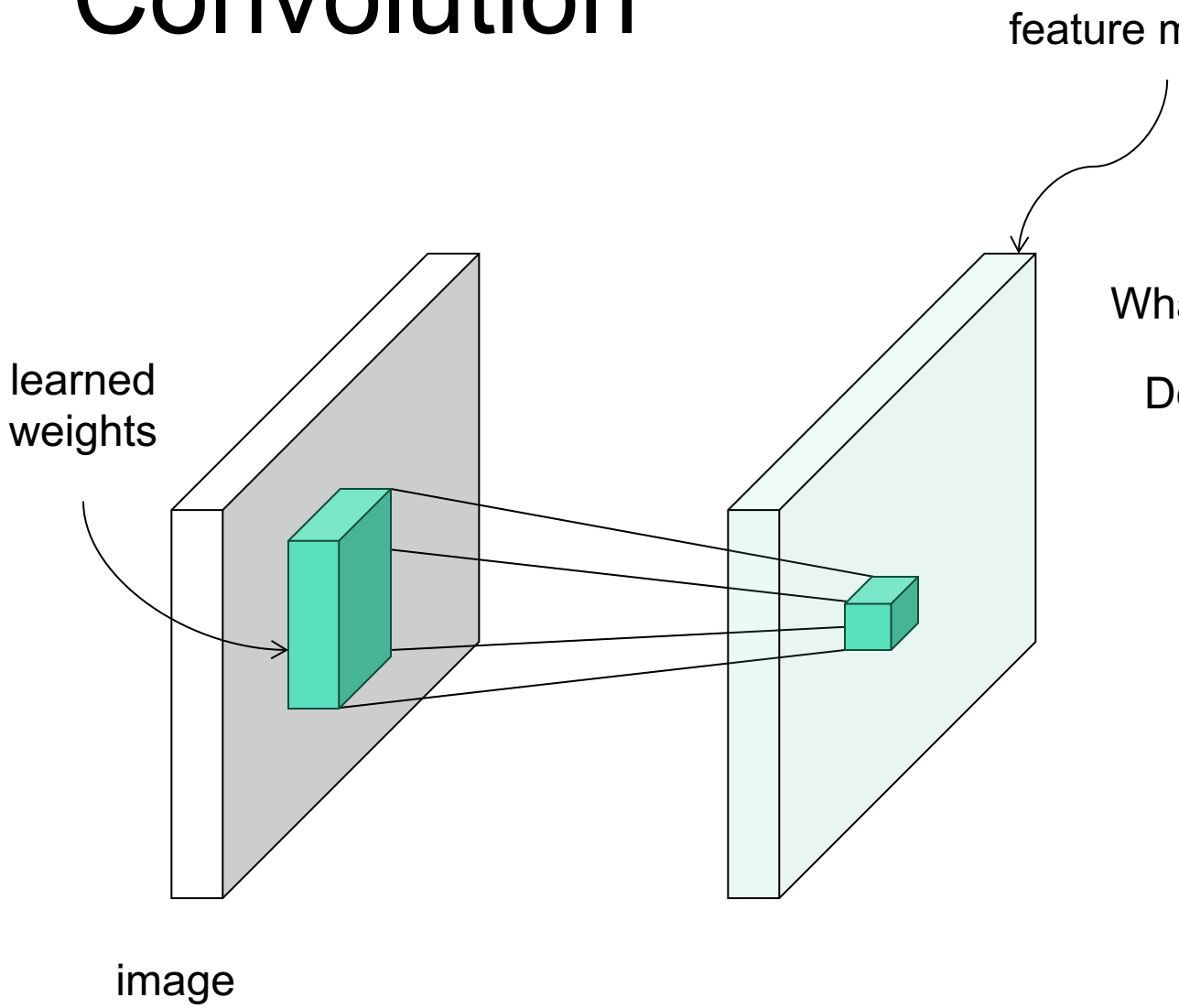
Depends on *padding* and *stride*:



No padding, stride 1

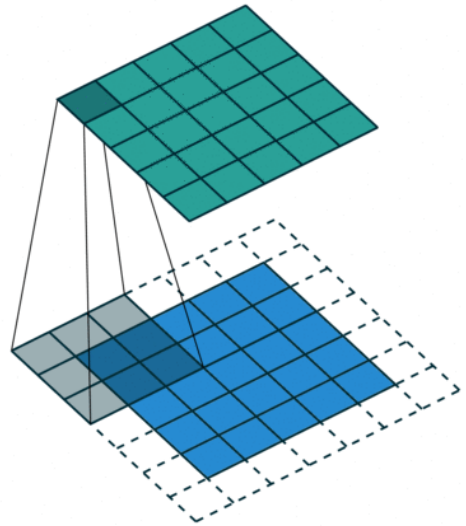
https://github.com/vdumoulin/conv_arithmetic

Convolution



What is the feature map resolution?

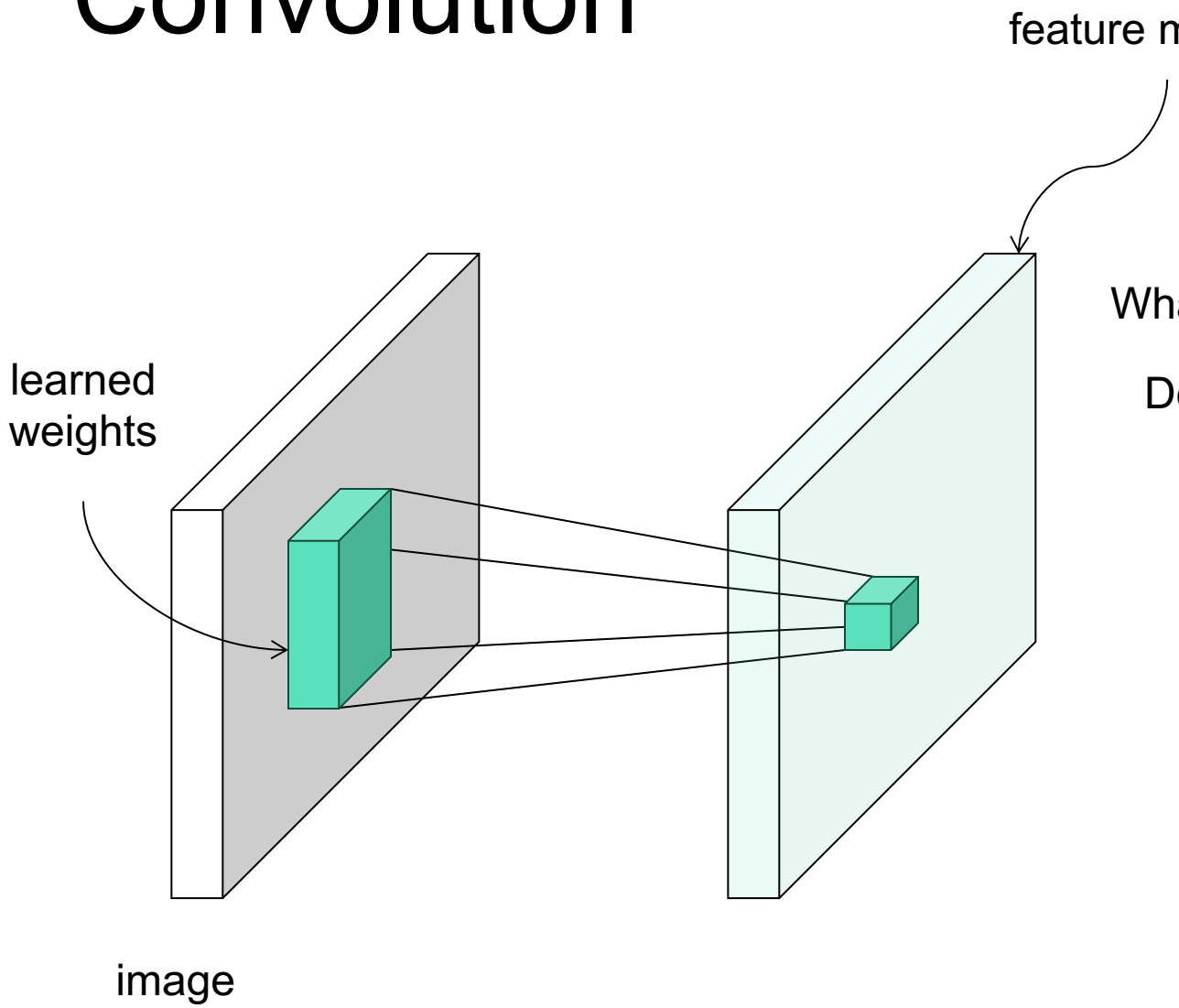
Depends on *padding* and *stride*:



With padding, stride 1

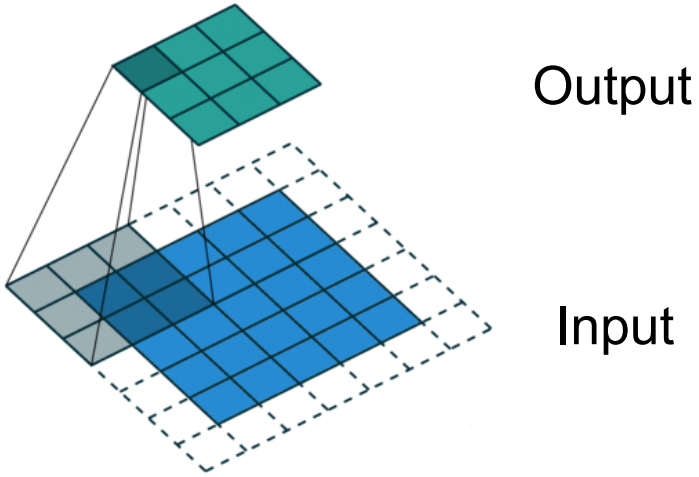
https://github.com/vdumoulin/conv_arithmetic

Convolution



What is the feature map resolution?

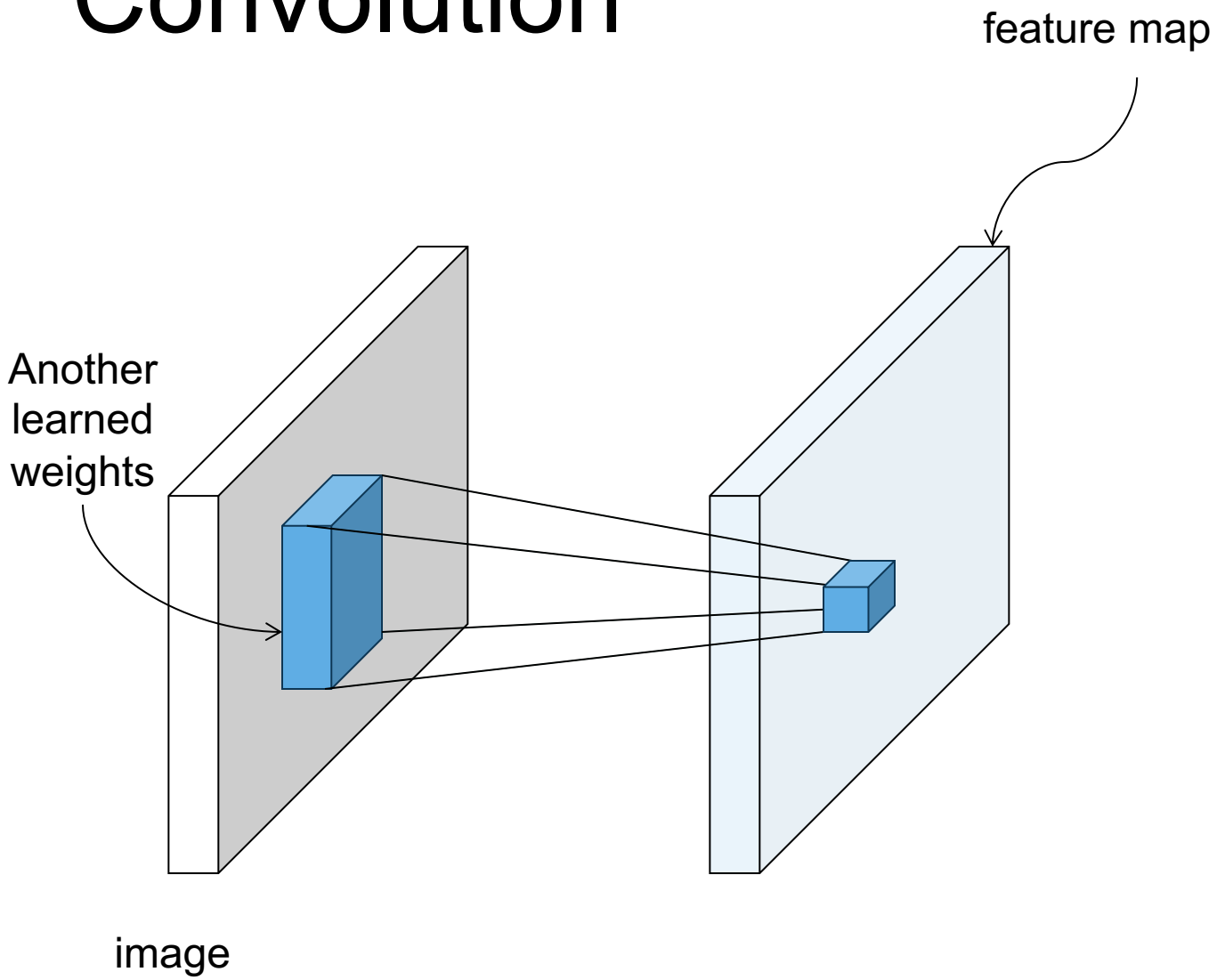
Depends on *padding* and *stride*:



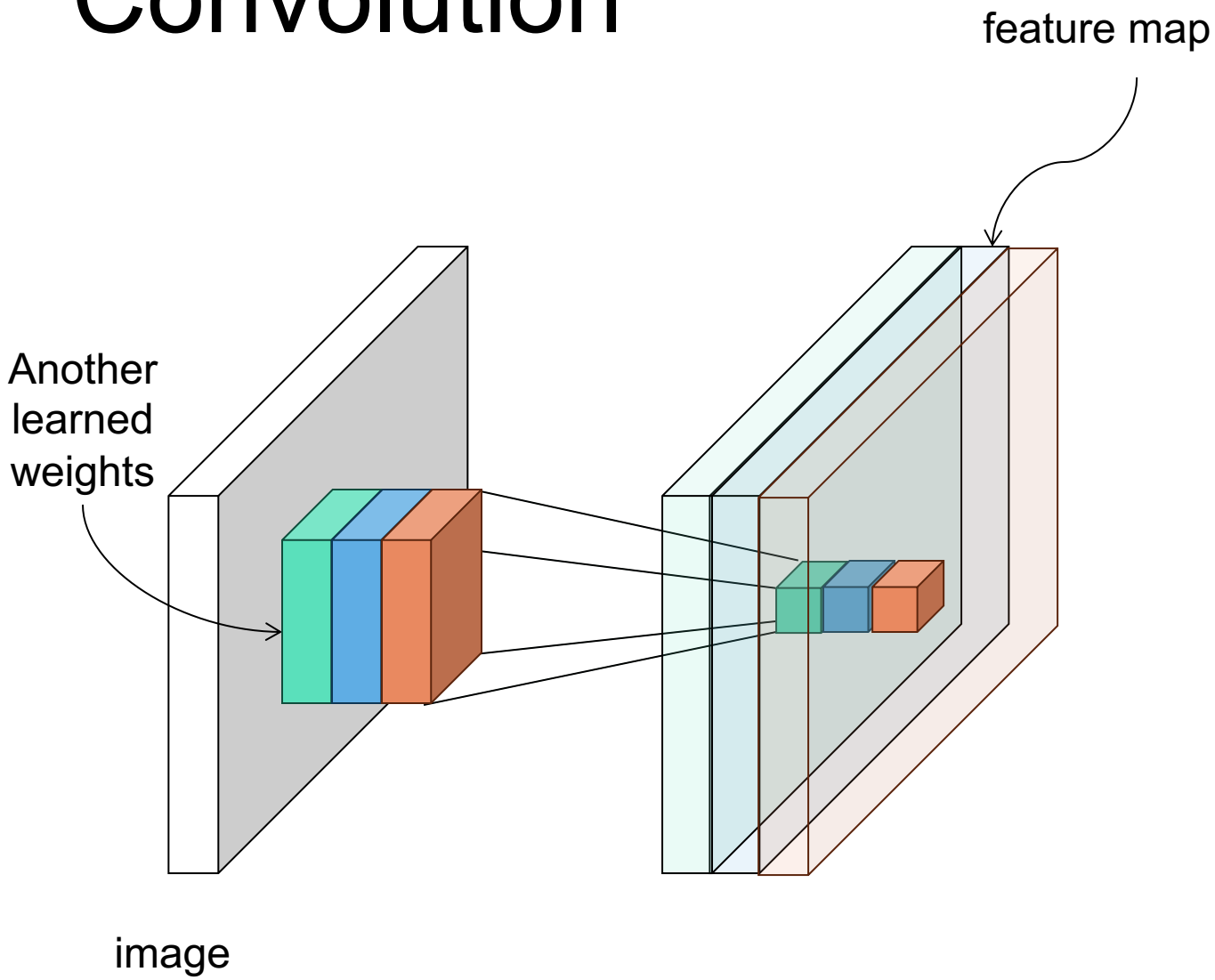
With padding, stride 2

https://github.com/vdumoulin/conv_arithmetic

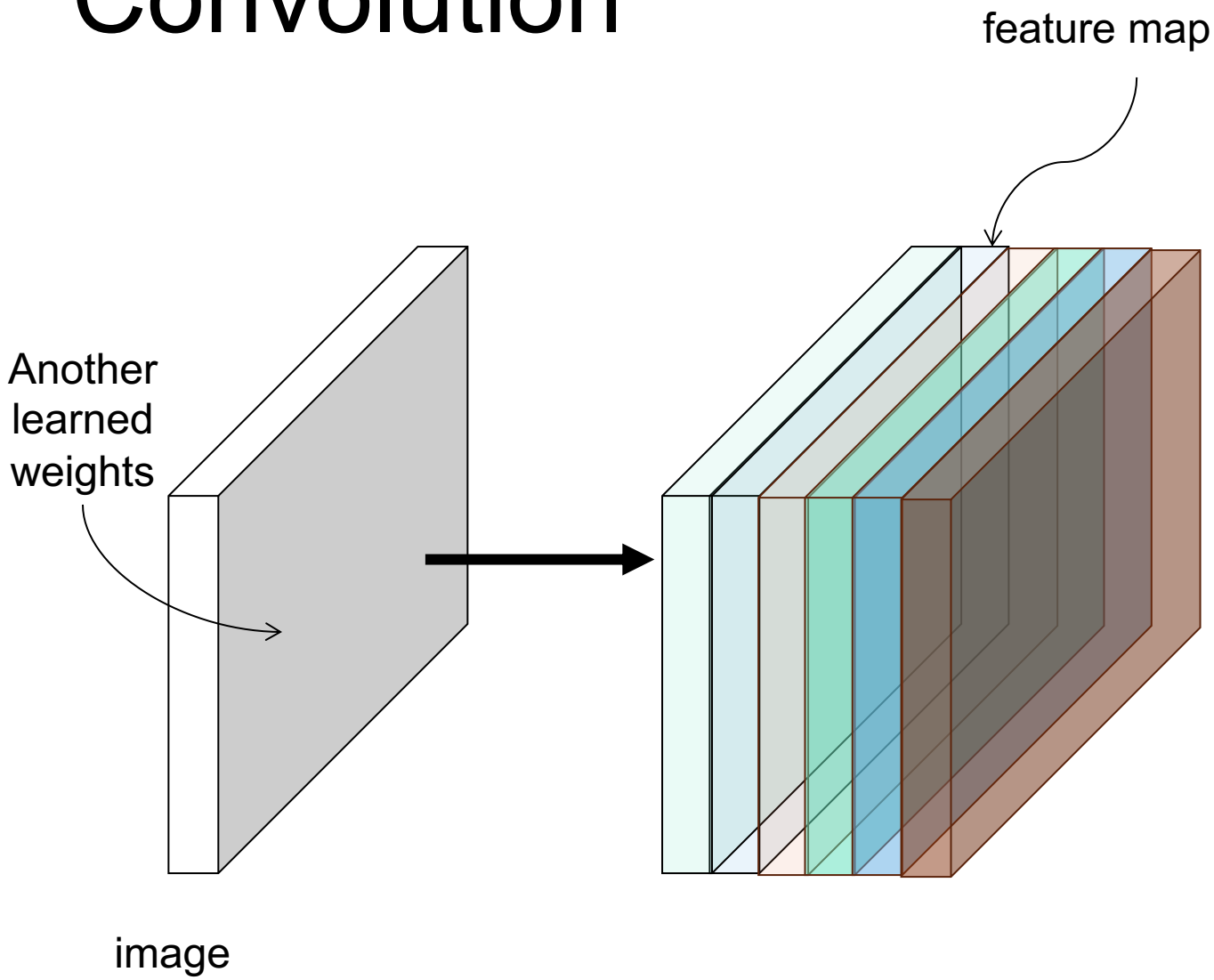
Convolution



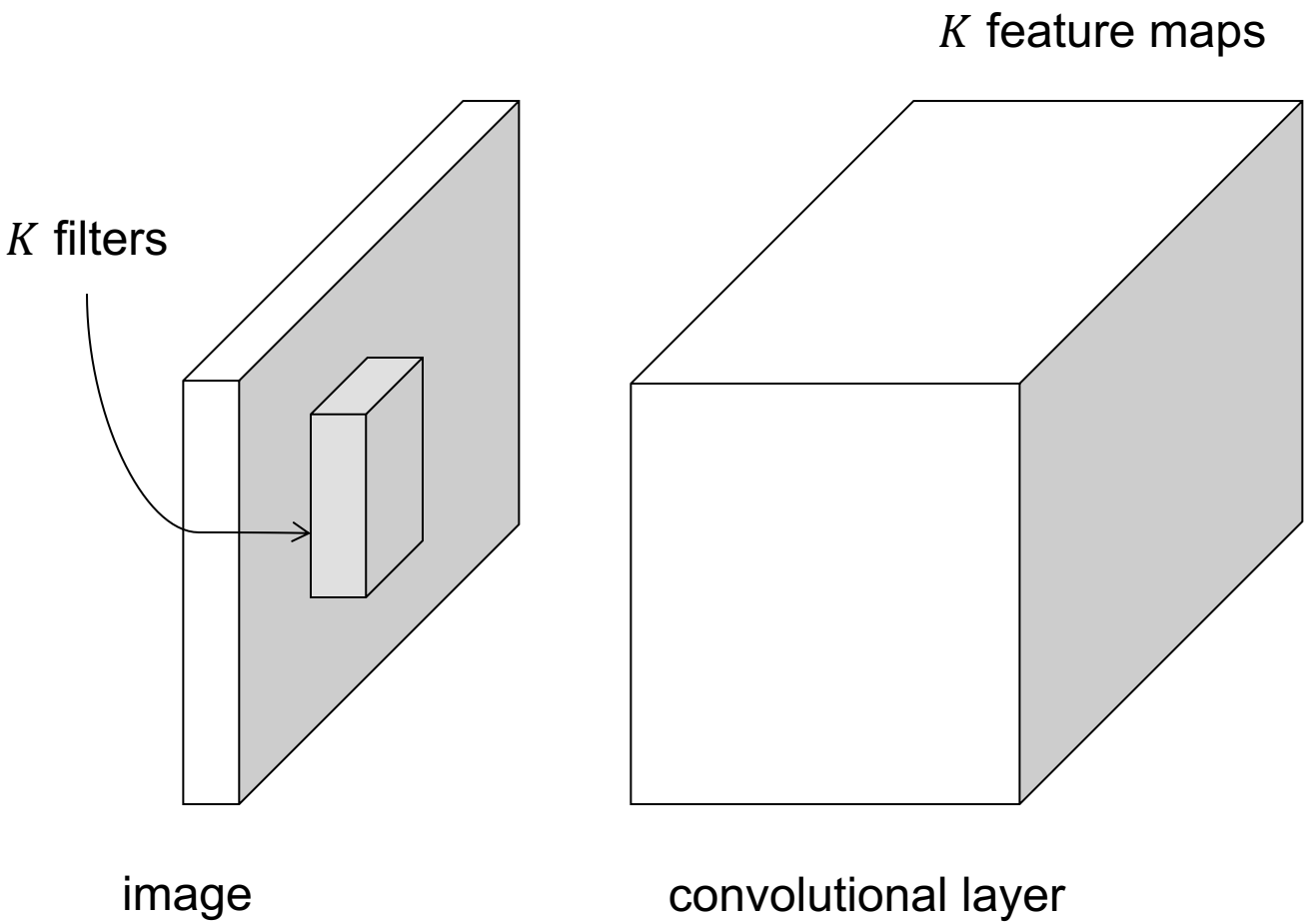
Convolution



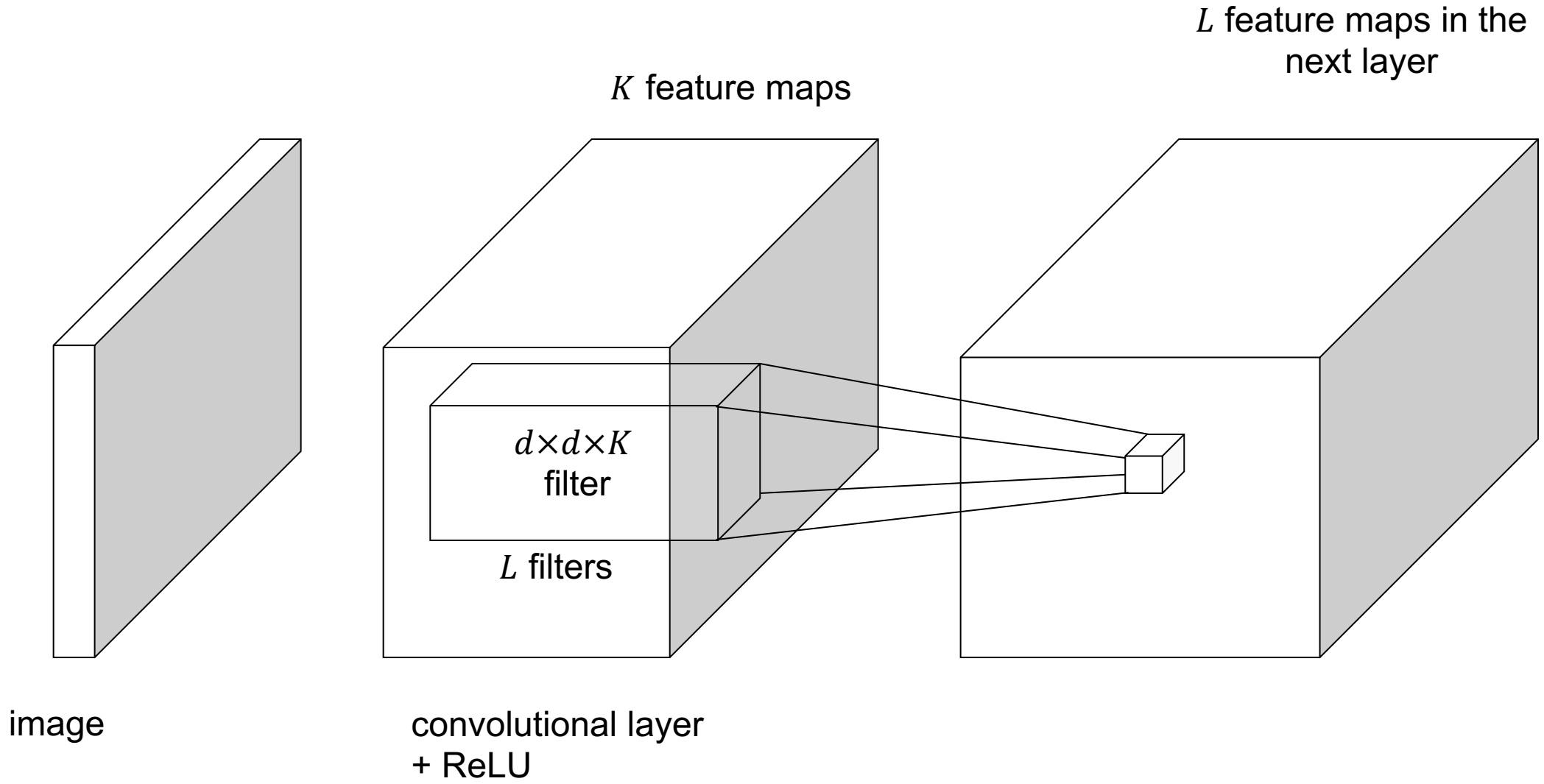
Convolution



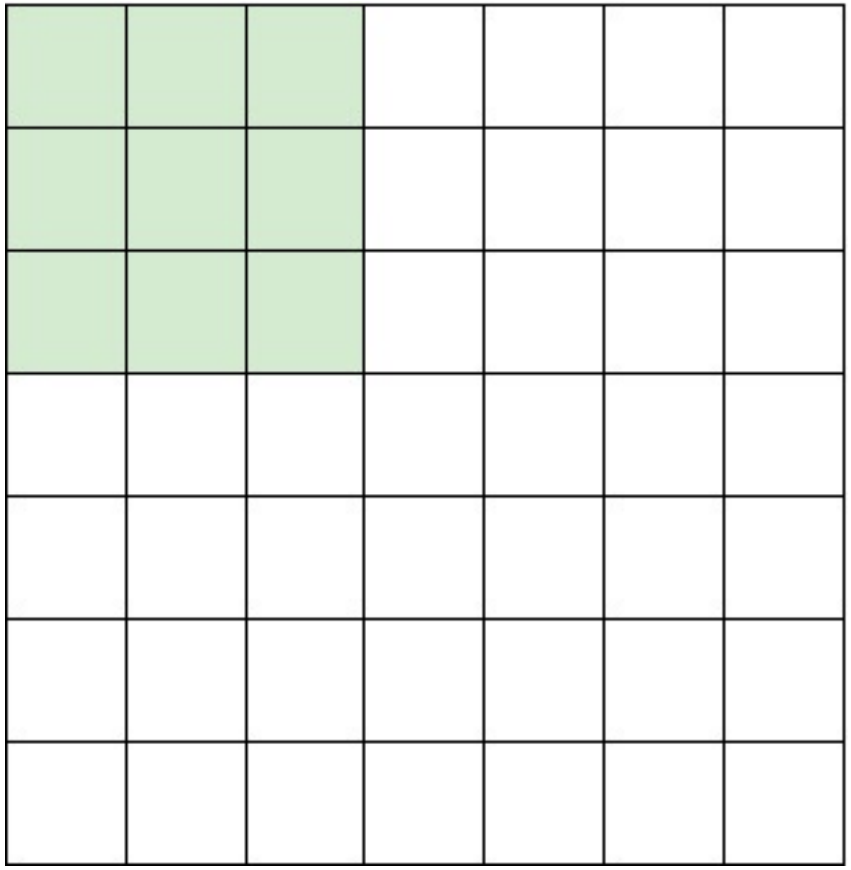
Convolution



Convolution



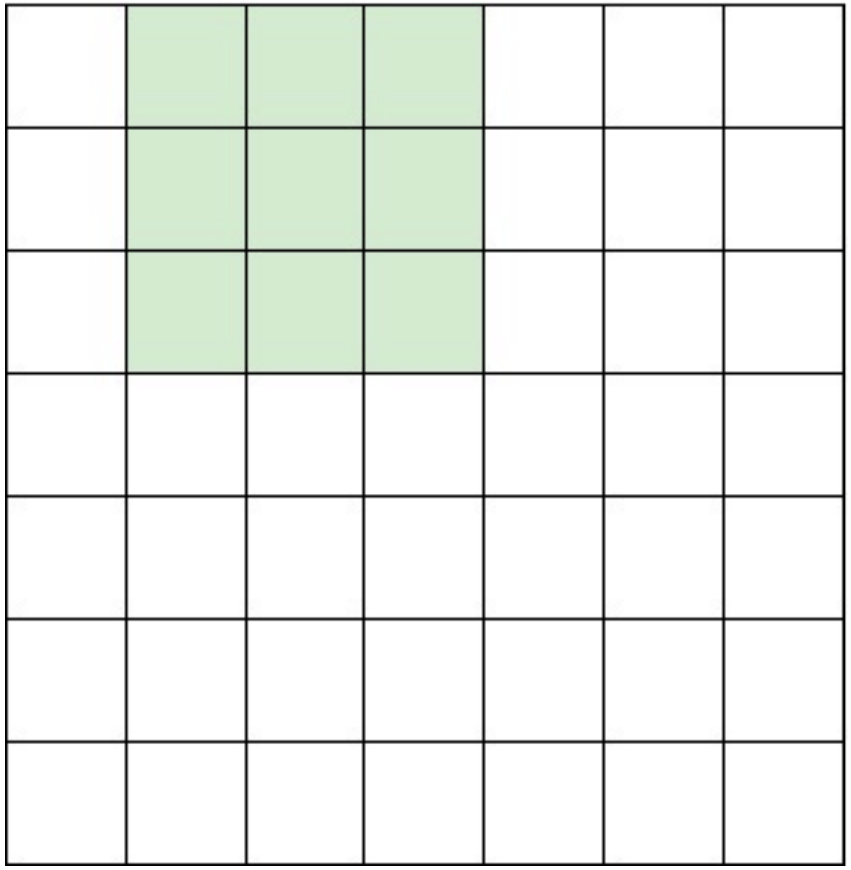
More examples on stride, padding, filter size



Input: 7×7 feature map

Filter: 3×3, stride=1

More examples on stride, padding, filter size



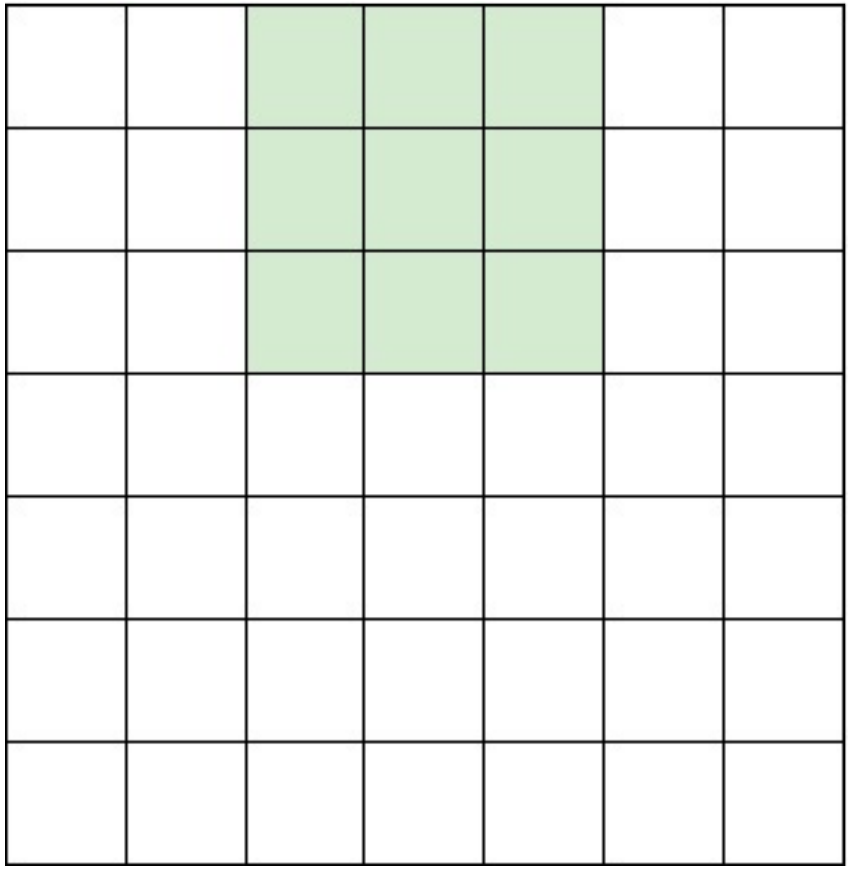
7

7

Input: 7×7 feature map

Filter: 3×3, stride=1

More examples on stride, padding, filter size



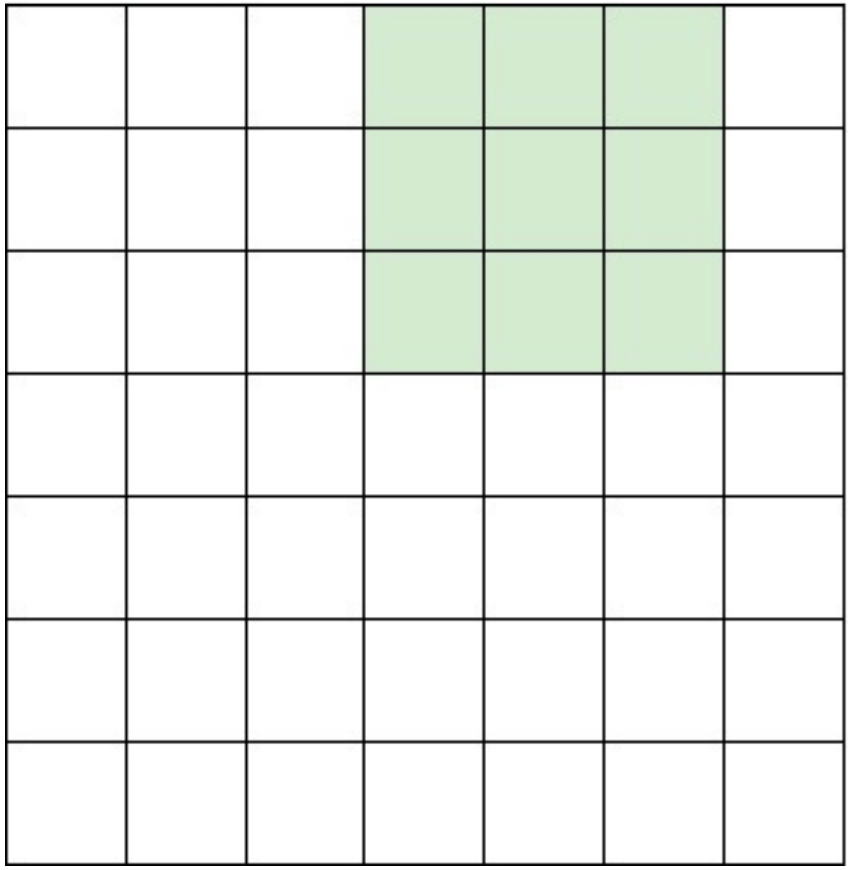
7

7

Input: 7×7 feature map

Filter: 3×3, stride=1

More examples on stride, padding, filter size



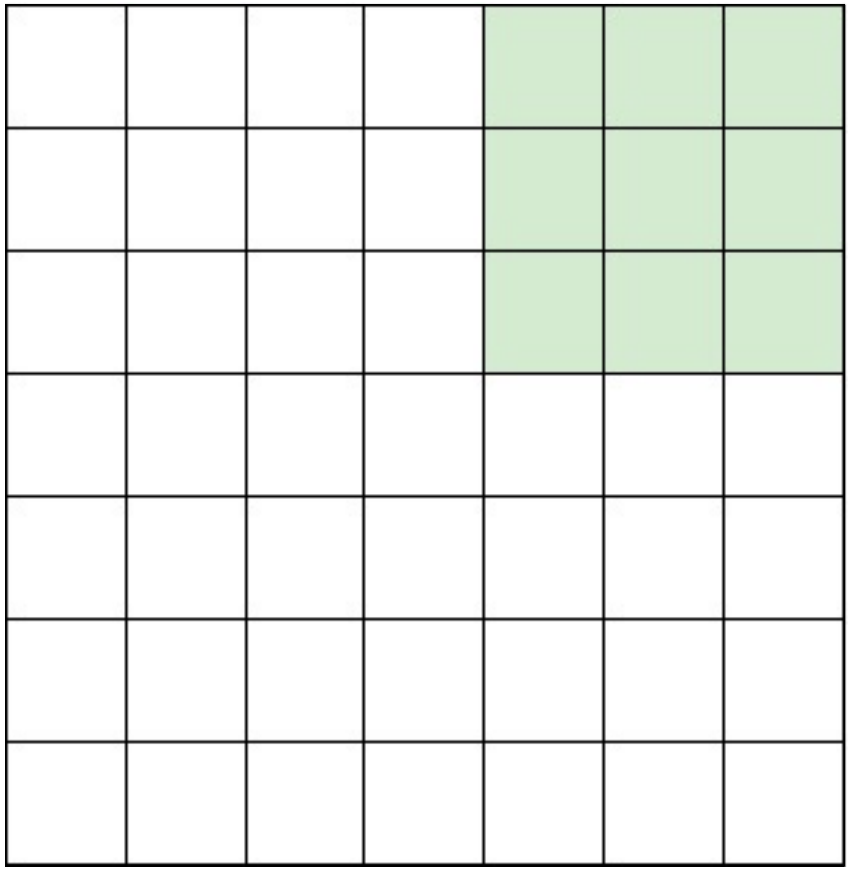
7

7

Input: 7×7 feature map

Filter: 3×3, stride=1

More examples on stride, padding, filter size



7

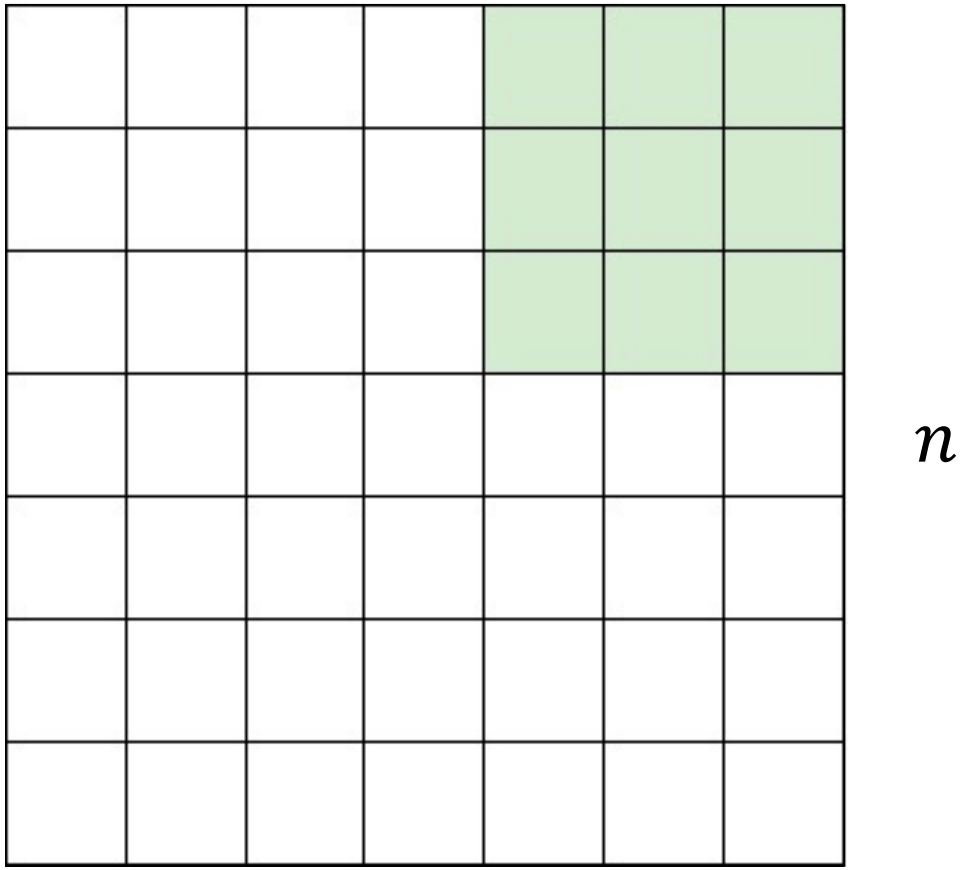
7

Input: 7×7 feature map

Filter: 3×3, stride=1

Output: 5×5 feature map

More examples on stride, padding, filter size



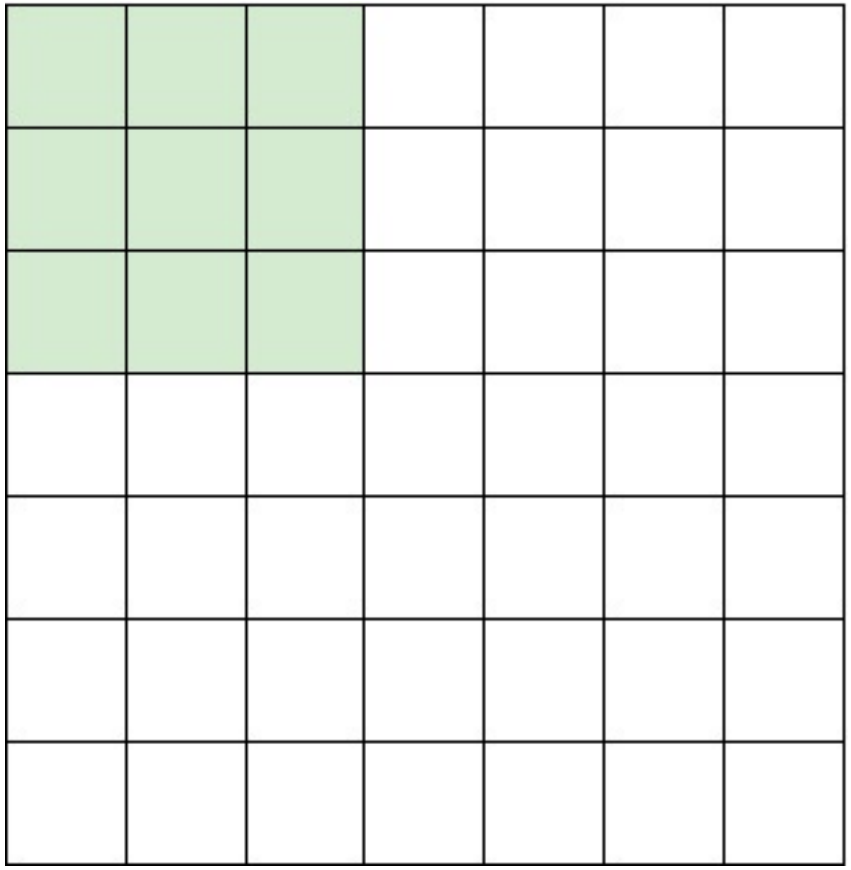
Input: $n \times n$ feature map

Filter: $d \times d$, stride=1

Output size:

$$\frac{n-d}{stride}+1$$

More examples on stride, padding, filter size



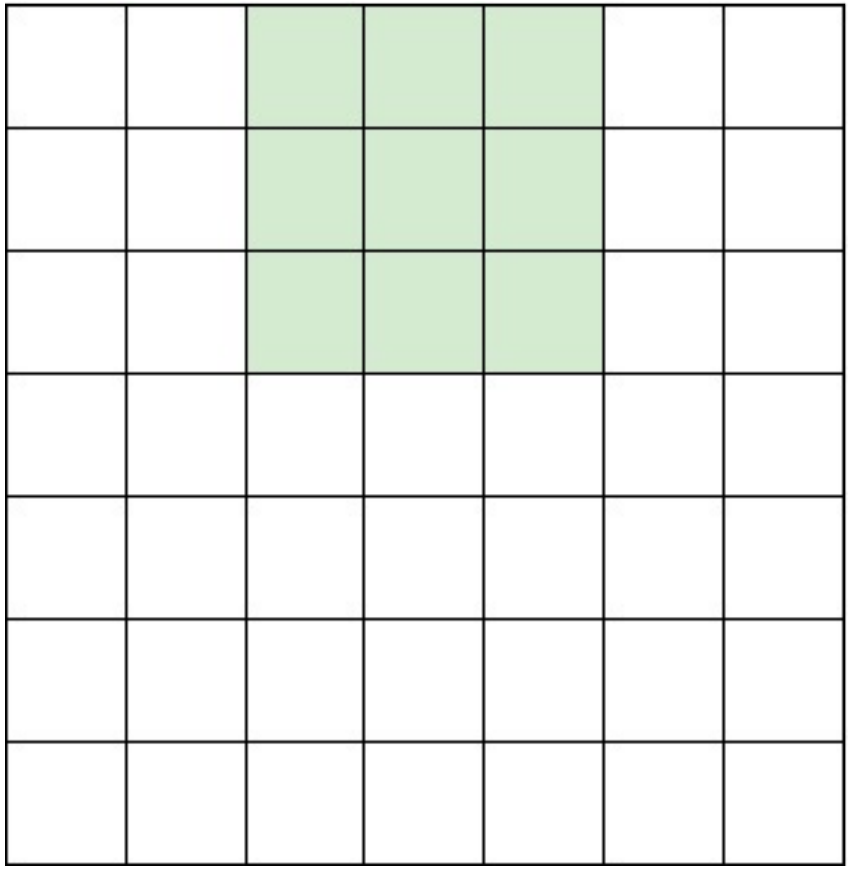
7

7

Input: 7×7 feature map

Filter: 3×3, stride=2

More examples on stride, padding, filter size



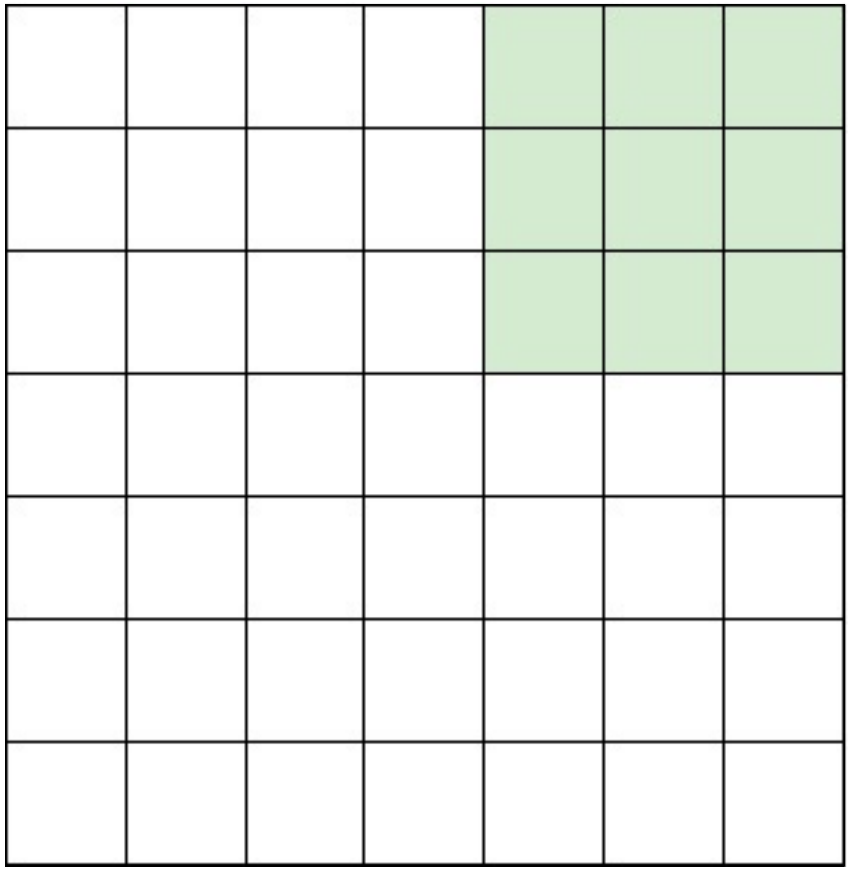
7

7

Input: 7×7 feature map

Filter: 3×3, stride=2

More examples on stride, padding, filter size



7

7

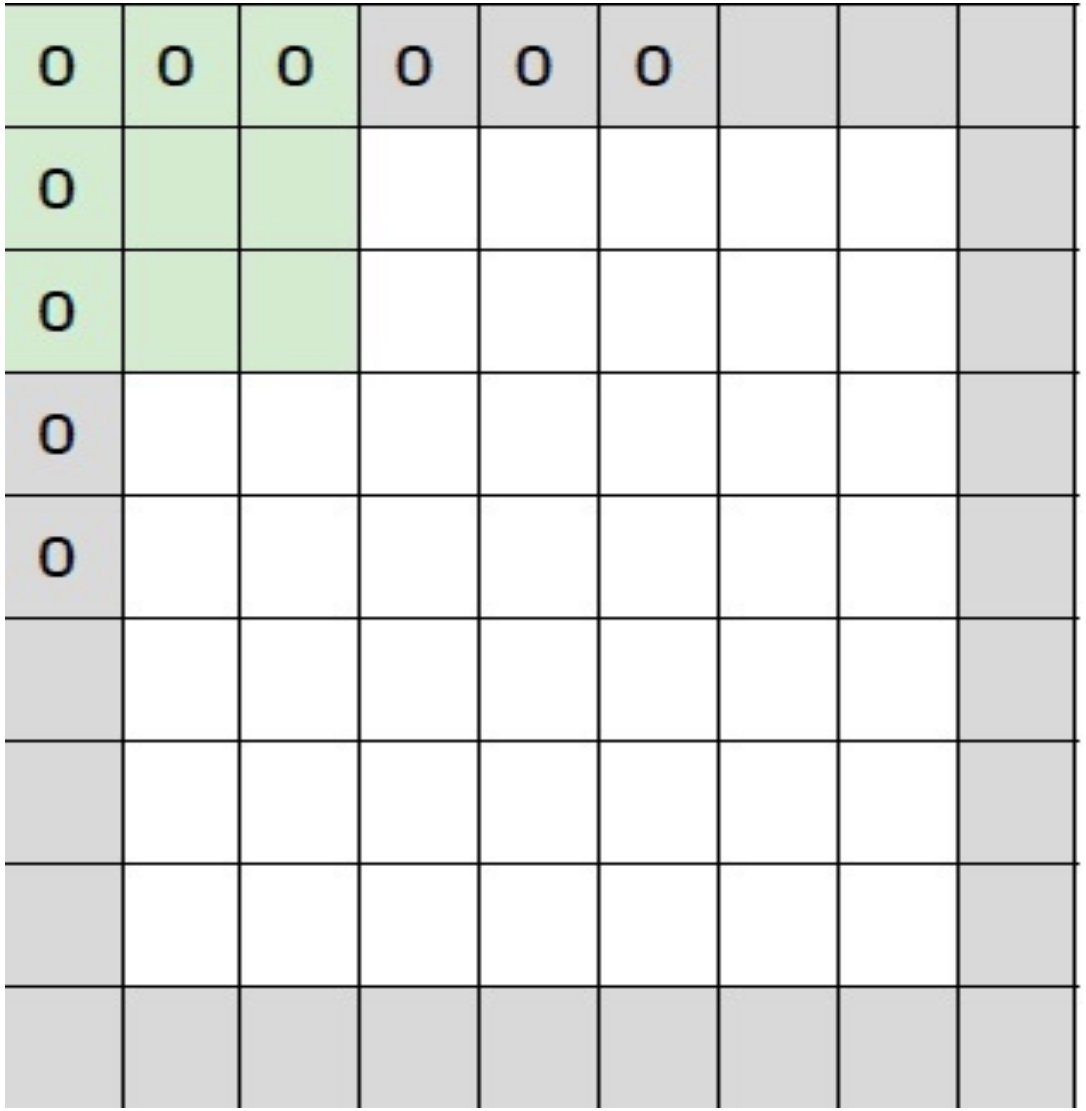
Input: 7×7 feature map

Filter: 3×3, stride=2

Output: 3×3 feature map

$$\frac{n-d}{stride}+1=\frac{7-3}{2}+1=3$$

More examples on stride, padding, filter size



Input: 7×7 feature map

Filter: 3×3, stride=2

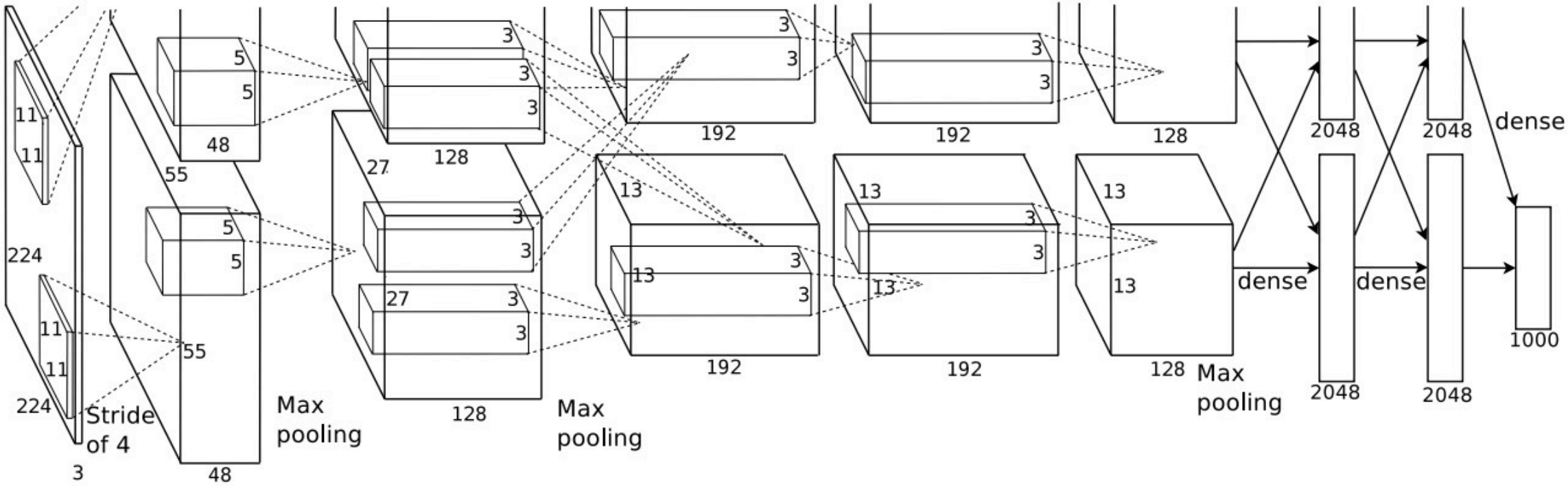
Padding: 1

Output: 4×4 feature map

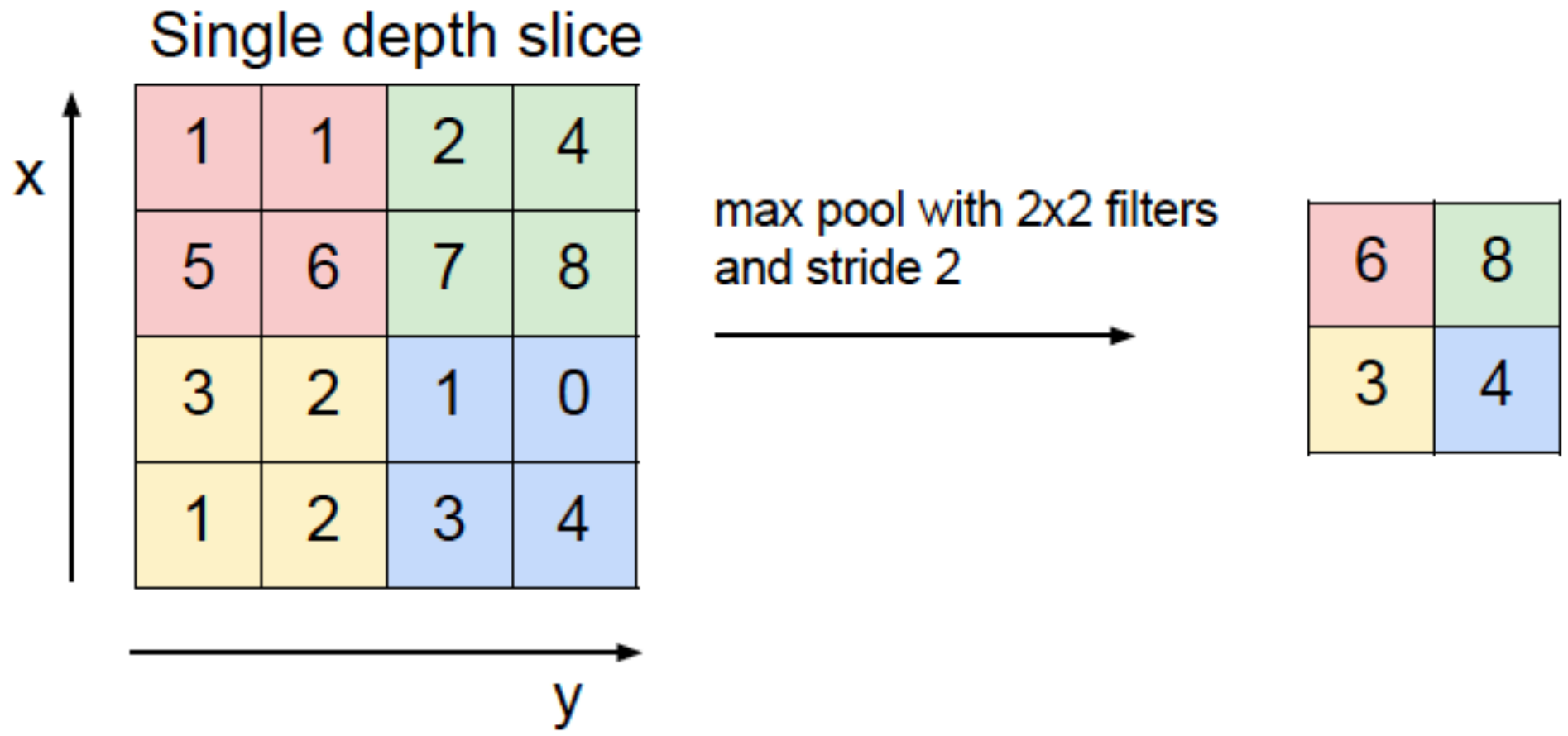
$$\frac{n + 2 \times pad - d}{stride} + 1 = \frac{7 + 2 - 3}{2} + 1 = 4$$

Max Pooling

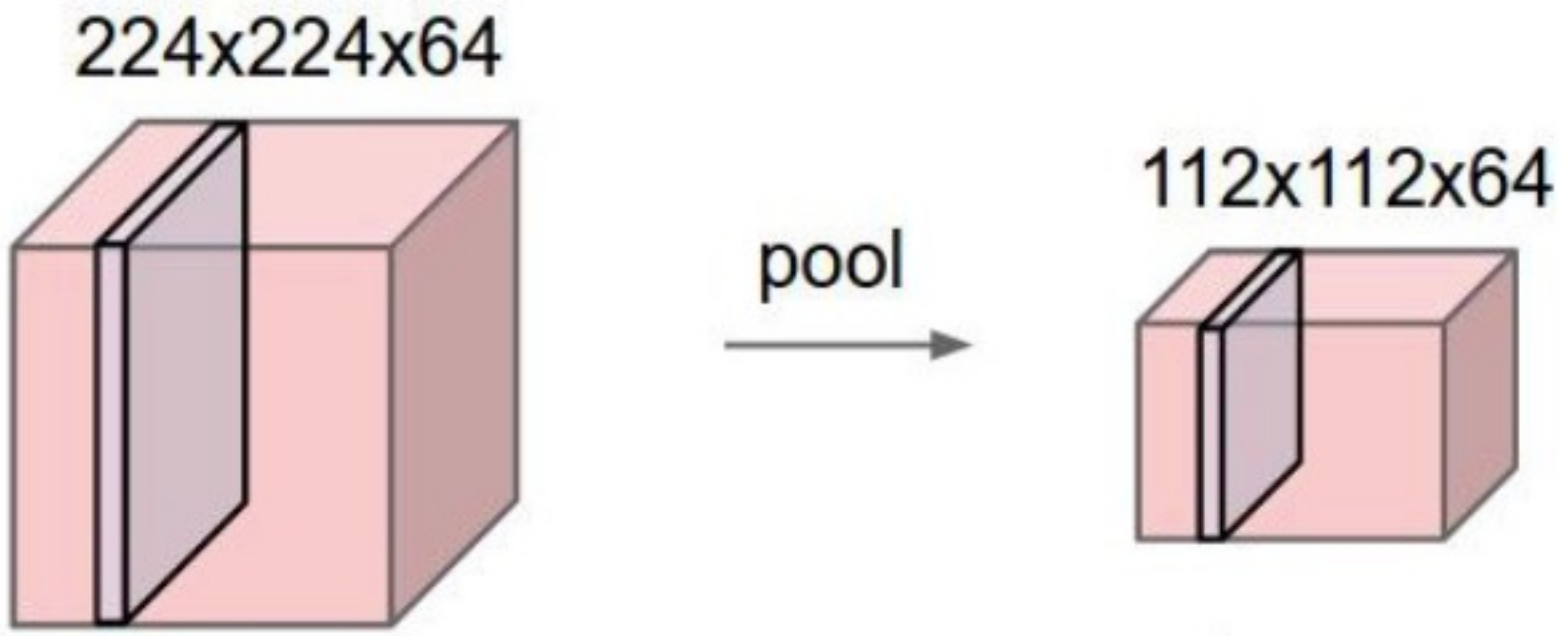
Max Pooling


224
224
3
11
11
11
11
Stride
of 4
55
55
48
48
5
5
5
5
Max
pooling
27
27
128
128
3
3
3
3
3
3
Max
pooling
13
13
13
13
192
192
192
192
128
128
3
3
3
3
3
3
3
3
3
3
Max
pooling
dense
dense
2048
2048
2048
2048
dense
1000

Max Pooling

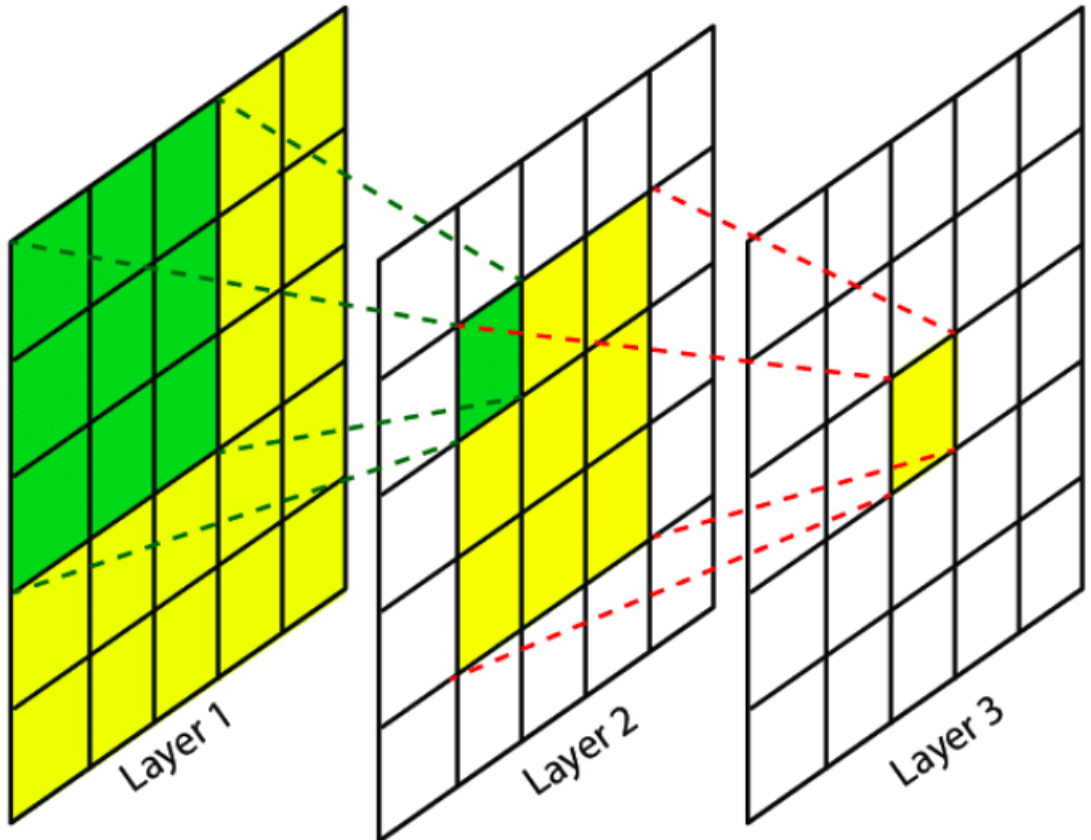

Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

Max Pooling



Why Max Pooling

- Save computation and memory
- Increase receptive field



Optimization methods

Mini-batch SGD

Loop:

- Sample a batch of data with size m: $(x_1, y_1), \ldots, (x_m, y_m)$, and forward the data to compute the loss $l(W, x_i, y_i)$.
- Use back-prop to compute the gradients for each layer:

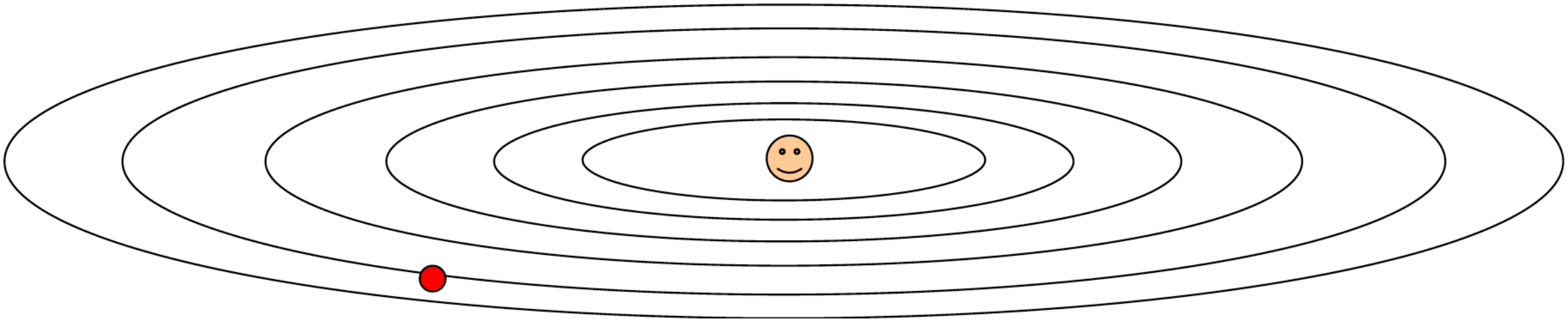
$$\nabla \hat{L} = \frac{1}{m} \sum_{i=1}^{m} \nabla l(W, x_i, y_i)$$

- Update the model parameters with learning rate α:

$$W \leftarrow W - \alpha \nabla \hat{L}$$

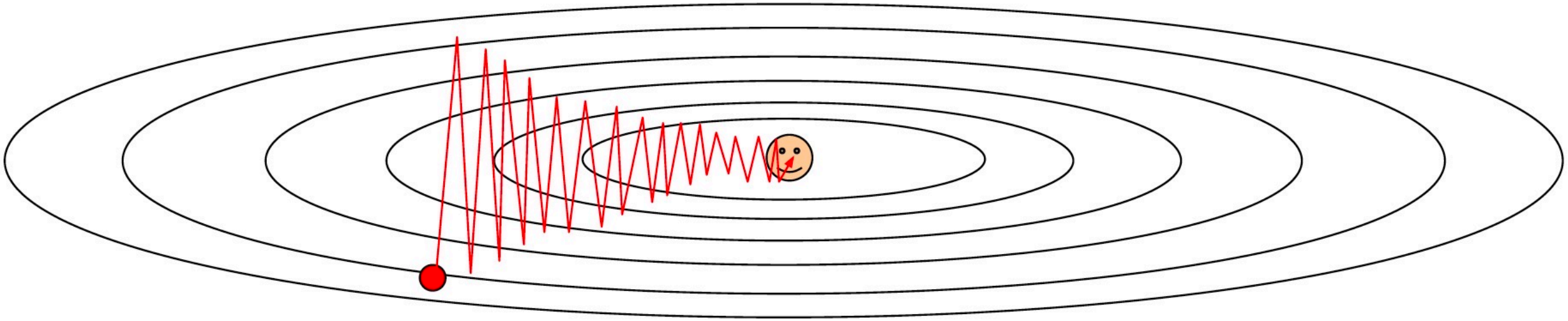
Problems with SGD

- Gradients are unstable



Problems with SGD

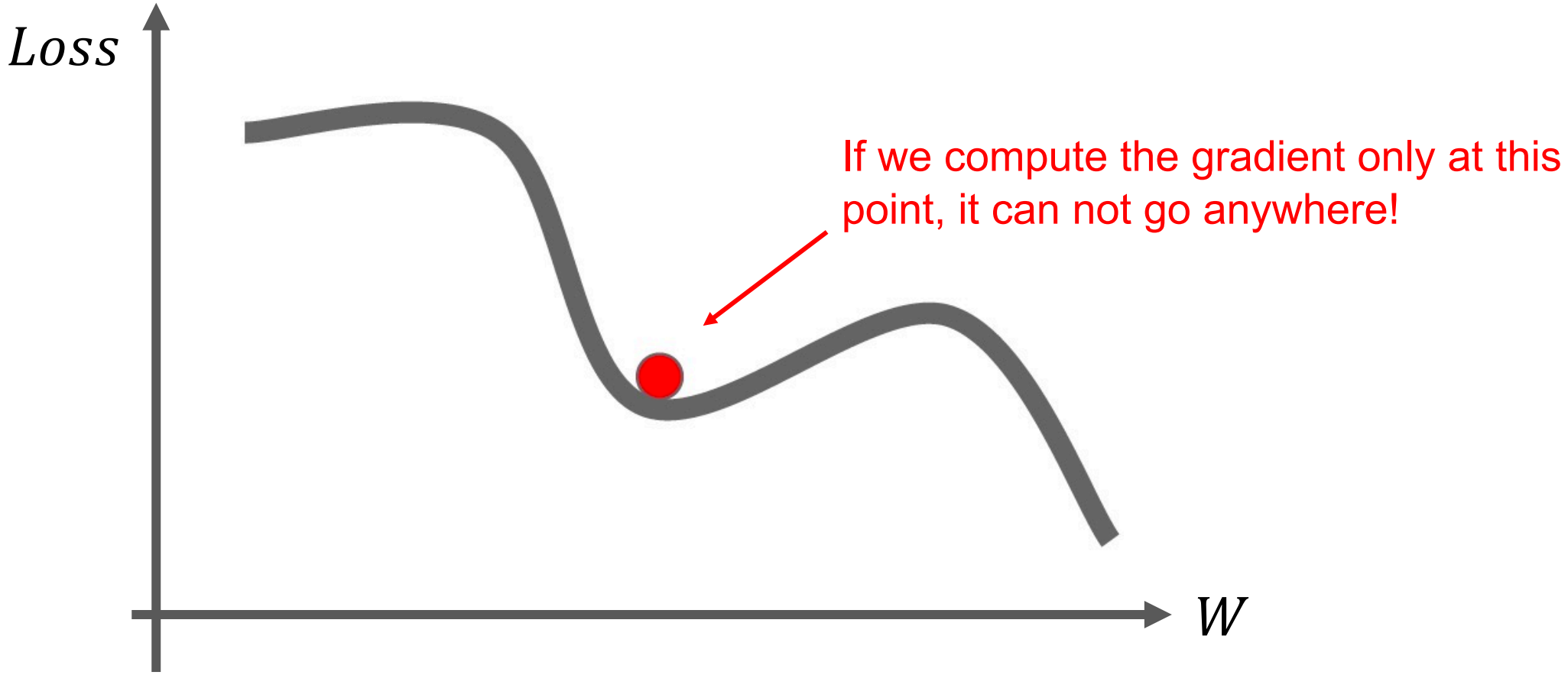
- Gradients are unstable



- Each time the gradient is estimated on a small batch, it will jitter a lot.

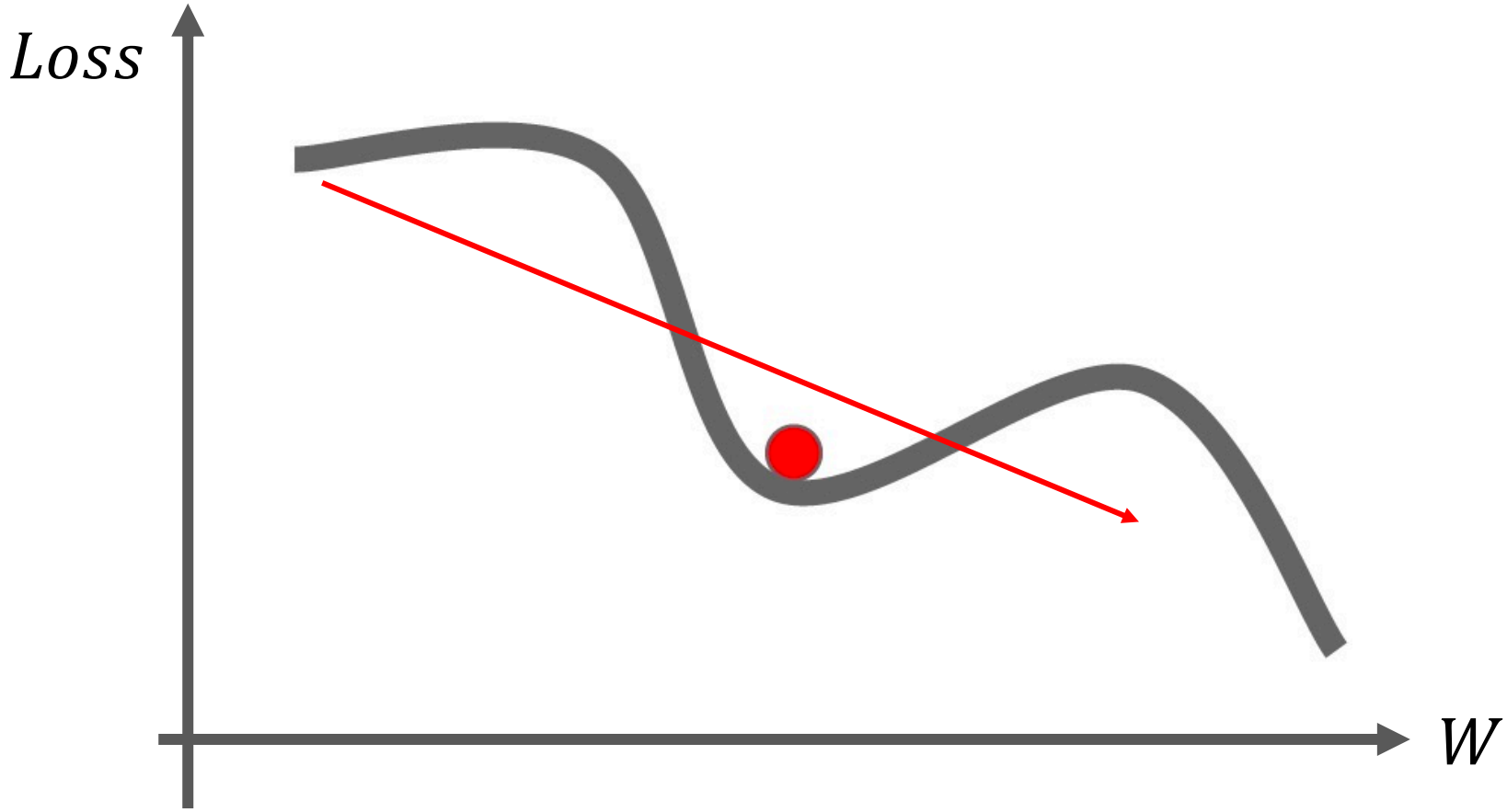
Problems with SGD

- It is easy to get stuck in a local minima:



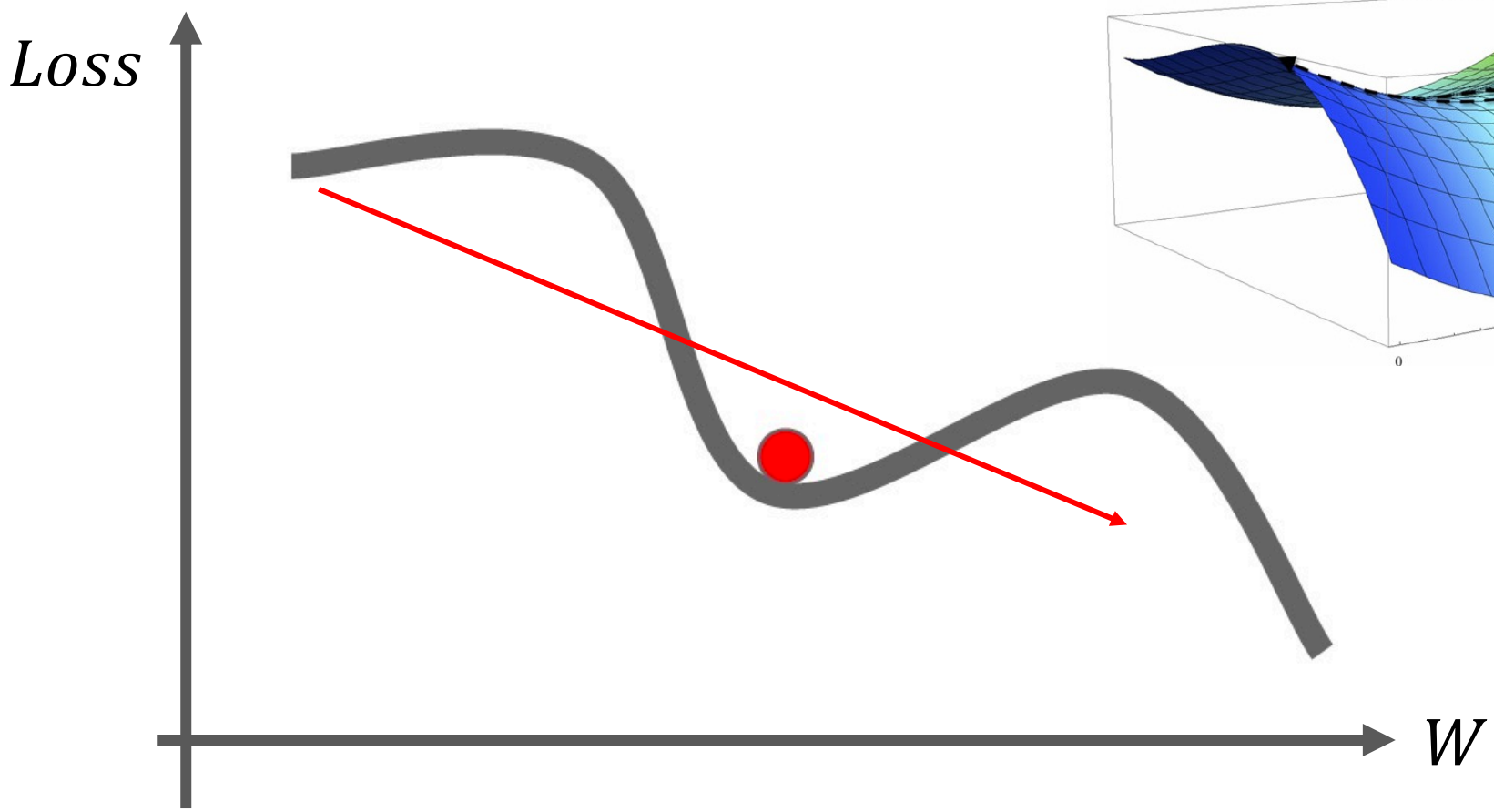
Problems with SGD

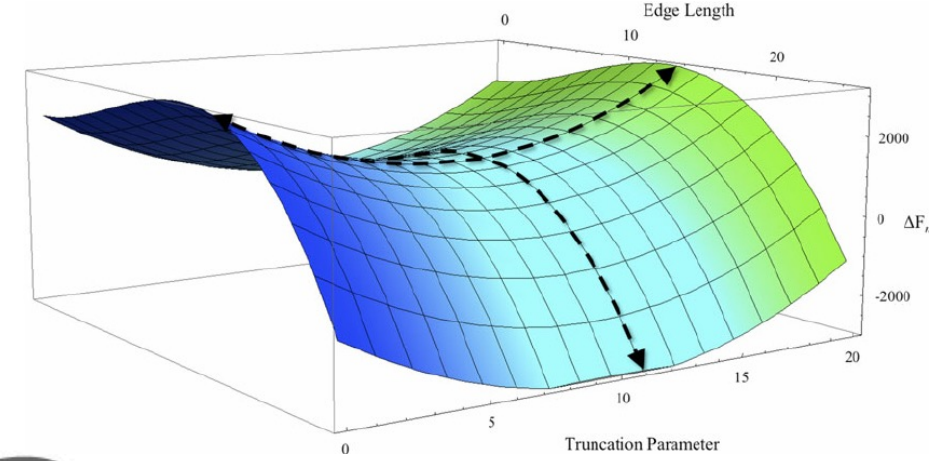
- Can we follow the trend instead just a local gradient?



Problems with SGD

- Can we follow the trend instead just a local gradient?





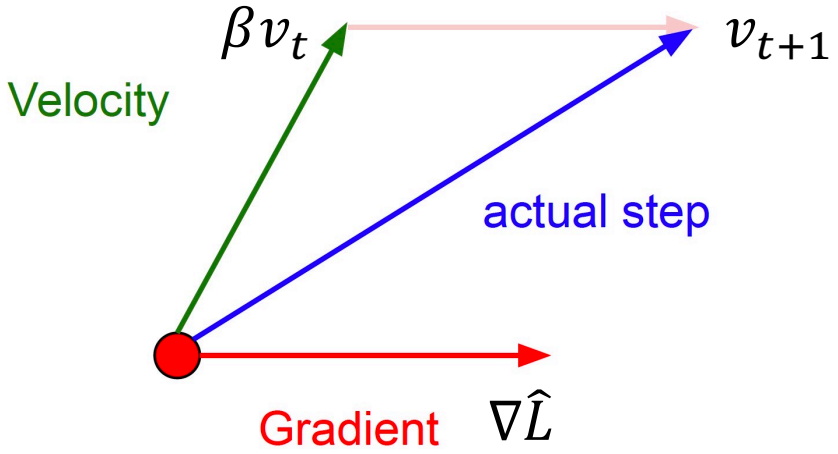
Saddle Points

SGD with momentum

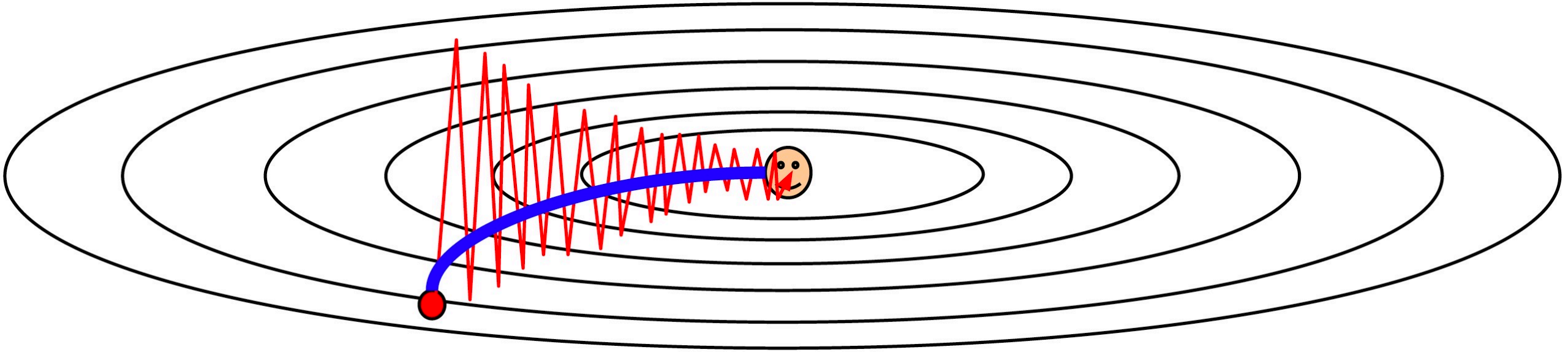
- Use a momentum variable as a weighted average of previous gradients:

$$v_{t+1} = \beta v_t + \nabla \hat{L}$$
$$W_{t+1} = W_t - \alpha v_{t+1}$$

We use t to index different training iterations, $\beta = 0.9$ for training image classifiers.



SGD with momentum



Adaptive Gradient Descent

- Different parameters / weights can converge in different speed
- Especially different weights in different layers have different scales of gradients
- Can we adjust the learning rate automatically?

AdaGrad

- Track the previous gradients, accumulate the magnitudes, and adjust the learning rate.

- We want to give large learning rate for gradients with low magnitudes and give small learning rate for gradients with high magnitudes:

$$m_{t+1} \leftarrow m_t + \left\| \frac{\partial L}{\partial W_t} \right\|^2$$

$$W_{t+1} \leftarrow W_t - \frac{\alpha}{\sqrt{m_{t+1}} + \epsilon} \frac{\partial L}{\partial W_t}$$

RMSProp

- The history of gradients are accumulated without forgetting in AdaGrad

- Introducing a decay factor β (≥ 0.9) to reduce the effect of the previous gradients

$$m_{t+1} \leftarrow \beta m_t + (1-\beta) \left\| \frac{\partial L}{\partial W_t} \right\|^2$$

$$W_{t+1} \leftarrow W_t - \frac{\alpha}{\sqrt{m_{t+1}} + \epsilon} \frac{\partial L}{\partial W_t}$$

J. Duchi, Adaptive subgradient methods for online learning and stochastic optimization, JMLR 2011

Adam

- Combining RMSProp and momentum:

$$v_{t+1} = \beta_1 v_t + \frac{\partial L}{\partial W_t}$$

$$m_{t+1} \leftarrow \beta_2 m_t + (1 - \beta_2) \left\| \frac{\partial L}{\partial W_t} \right\|^2$$

$$W_{t+1} \leftarrow W_t - \frac{\alpha}{\sqrt{m_{t+1}} + \epsilon} v_{t+1}$$

Default parameters from paper: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\eta = 1e-3$, $\epsilon = 1e-8$

D. Kingma and J. Ba, Adam: A method for stochastic optimization, ICLR 2015

Which Optimizer to use? (Common Practice)

- SGD is widely used in (with larger lr, e.g., lr = 1e-2):
  - Image classification
  - Object detection
  - Other recognition tasks

- Adam is widely used in (with lower lr, e.g., lr = 2e-4):
  - Image synthesis
  - Image reconstruction
  - Other reconstruction tasks

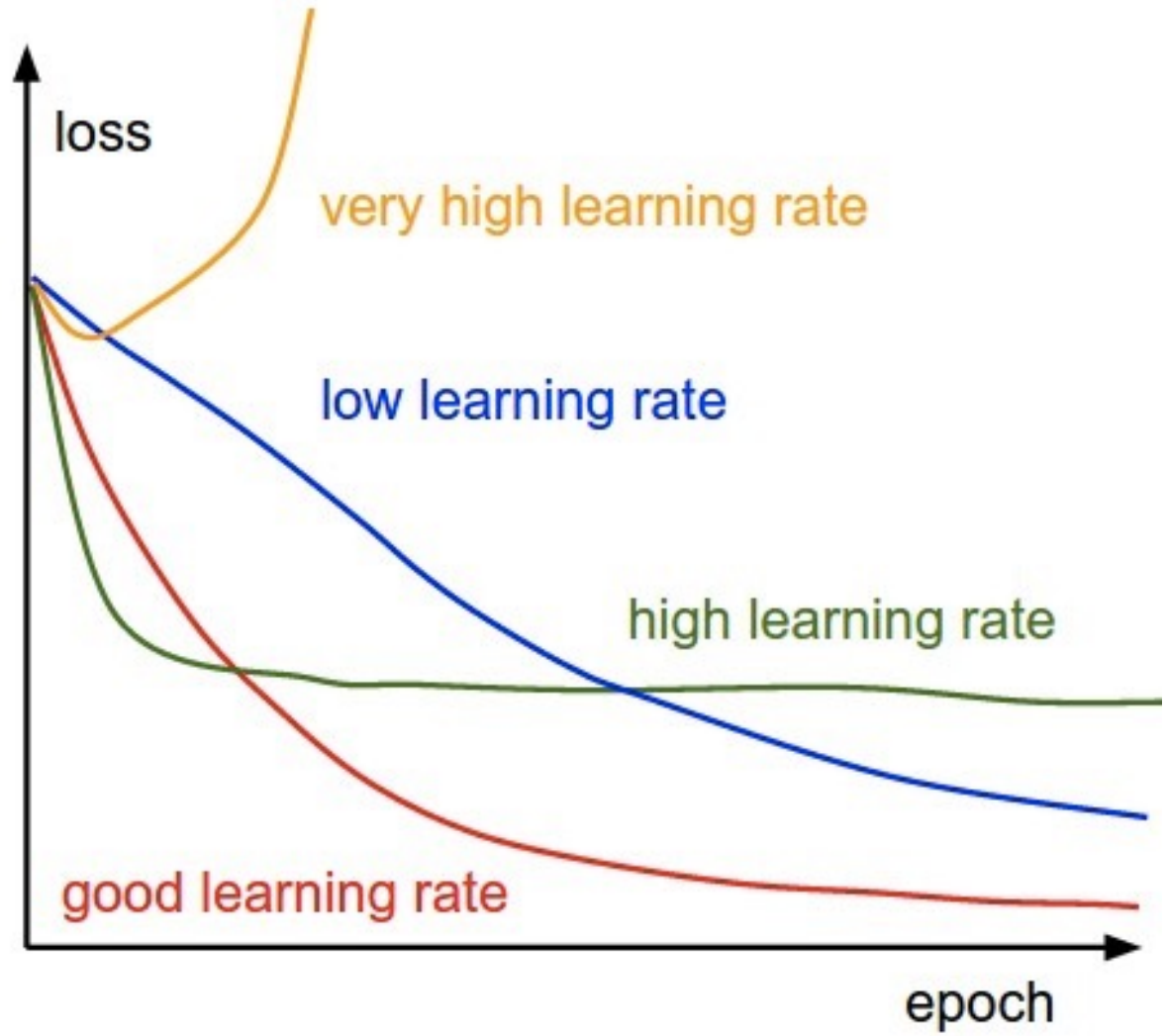
Which Optimizer to use? (Common Practice)

- SGD gives better performance than Adam in recognition tasks, but Adam is usually more stable and converge faster

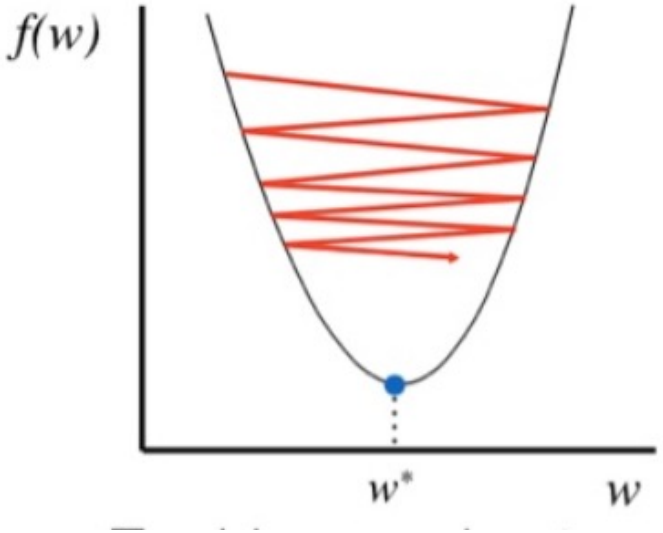
- For challenging tasks, if SGD does not work, we can try to use Adam

Learning Rate α

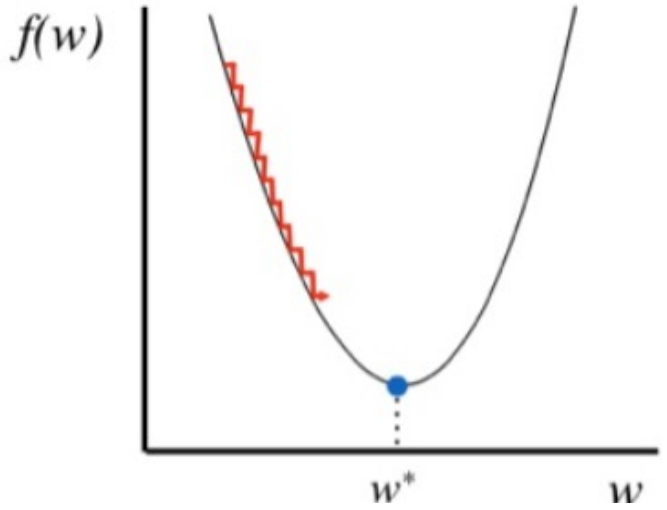
Learning rate



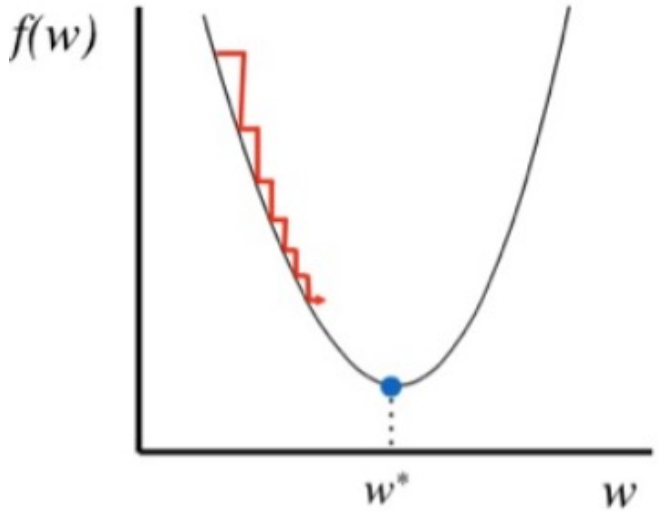
Learning rate



Learning rate too large:
Hard to converge
and will even diverge



Learning rate too small:
Converge very slowly
and can easily fall in local minima

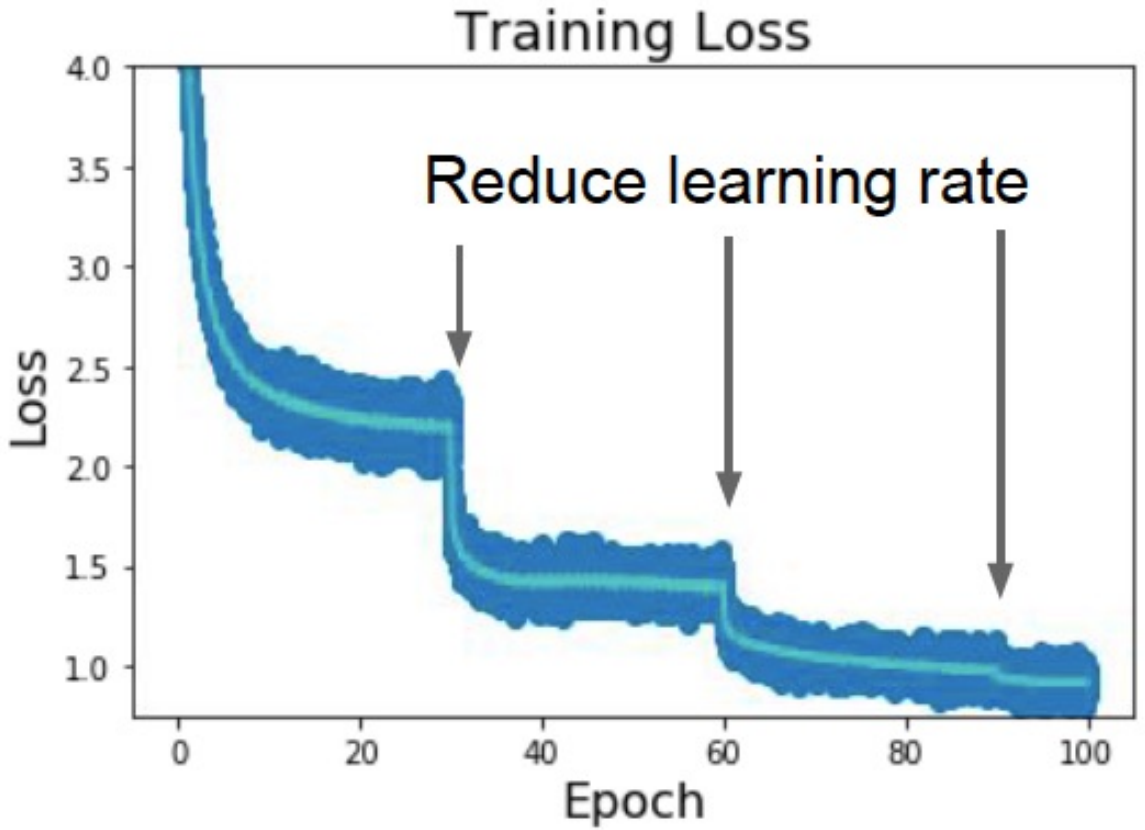


Start with large learning rate
and reduce over time

https://www.slideshare.net/ssuserf88631/scalable-machine-learning-73621818

Learning Rate Decay

- The most common practice: multiply the LR with a constant every K epochs



Training image classifier with ImageNet:

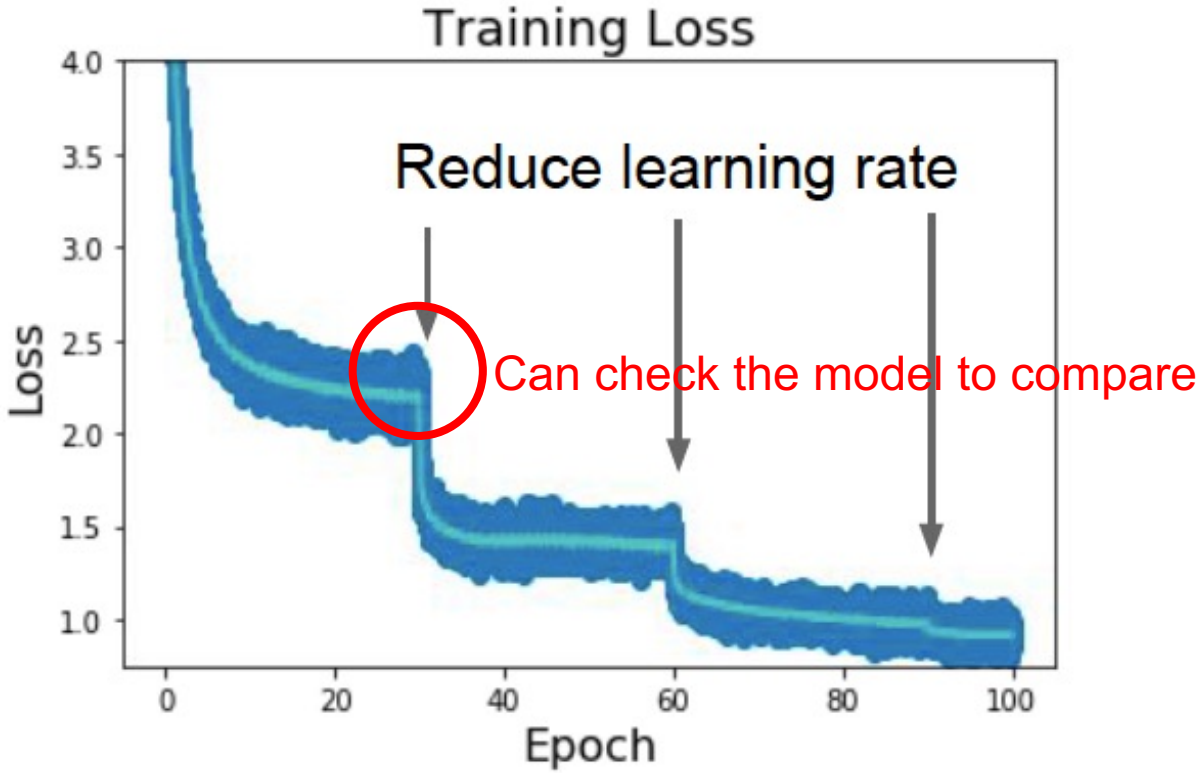
Start LR=0.1

Decay LR by multiplying 0.1 every 30 epochs:
0~30 epochs: 0.1
30~60 epochs: 0.01
60~90 epochs: 0.001

Some experience on setting learning rate

- Try to make the initial LR as large as possible
  - Good for exploration
- The initial LR is the most important one
  - Most related to the performance
  - Less easy to overfit
- Check the model performance after first LR for debug

Some experience on setting learning rate



- Instead of decaying LR at 30,60,90 epochs, if you want to save time, you can train with the first LR longer (50 epochs) and train with the following LR with shorter time (10 epochs each)

Empirical rule

- If you times the batch size by k, you should increase the learning rate by k.

- For example, if you train a classification network using batch size 128, and learning rate 0.1

-> batch size 256, learning rate 0.2

-> batch size 512, learning rate 0.4

Next Class

More elements in training Convolutional Neural Networks