卷积神经网络

王树森

内积

向量内积

- $\mathbf{a} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$ and $\mathbf{b} = \begin{bmatrix} 5 \\ 6 \\ 7 \\ 8 \end{bmatrix}$.
- 内积:

$$\langle \mathbf{a}, \mathbf{b} \rangle = \mathbf{a}^T \mathbf{b} = \sum_i a_i b_i = 70.$$

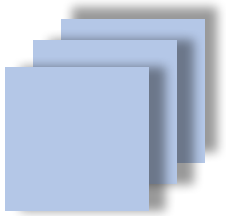
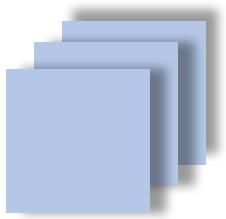
矩阵内积

- $\mathbf{A} = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$ and $\mathbf{B} = \begin{bmatrix} 5 & 7 \\ 6 & 8 \end{bmatrix}$.
- 内积:

$$\langle \mathbf{A}, \mathbf{B} \rangle = \sum_i \sum_j a_{ij} b_{ij} = 70.$$

- 定理 : $\langle \mathbf{A}, \mathbf{B} \rangle = \langle \mathrm{vec}(\mathbf{A}), \mathrm{vec}(\mathbf{B}) \rangle$.

张量内积

- $\mathbf{A} =$  and $\mathbf{B} =$  .

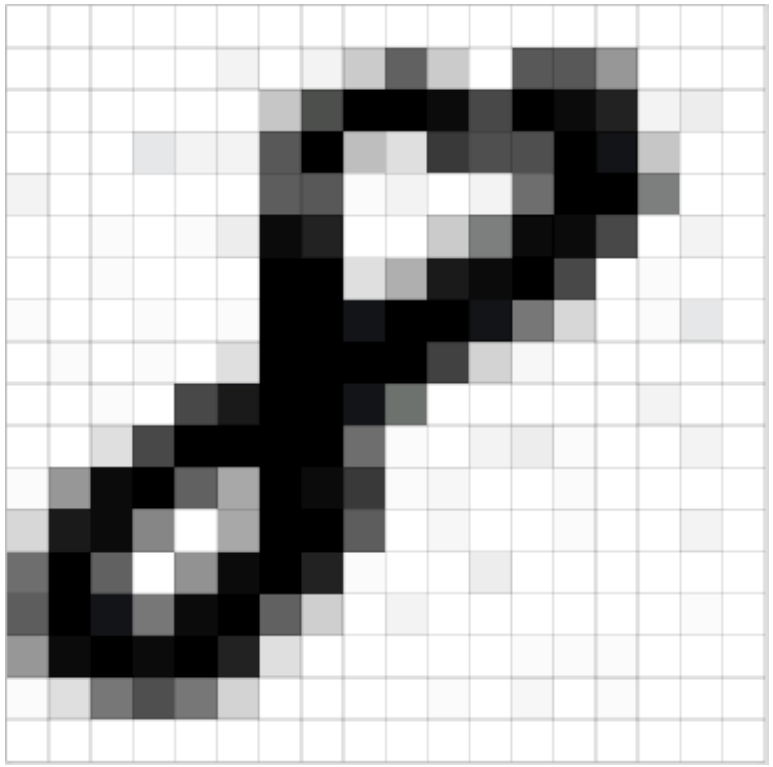
- 内积:

$$\langle \mathbf{A}, \mathbf{B} \rangle = \sum_i \sum_j \sum_k a_{ijk} b_{ijk} .$$

  - 定理: $\langle \mathbf{A}, \mathbf{B} \rangle = \langle \mathrm{vec}(\mathbf{A}), \mathrm{vec}(\mathbf{B}) \rangle.$

图片卷积

图像是像素值的矩阵/张量



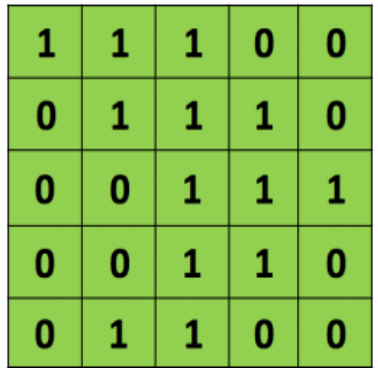
MNIST 手写数字识别数据集

- 每张灰度图像$\mathbf{x}_j$，是一个 28×28 的矩阵.
- (卷积之前, 我们将其重塑为一个 784 维向量)

卷积

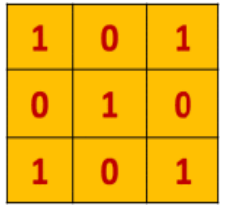
输入图像

5×5



卷积核

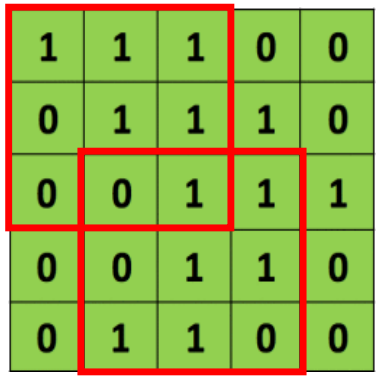
3×3



卷积

输入图像

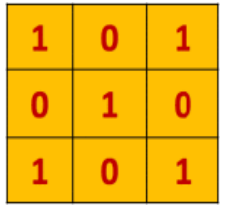
5×5



3×3 patch

卷积核

3×3



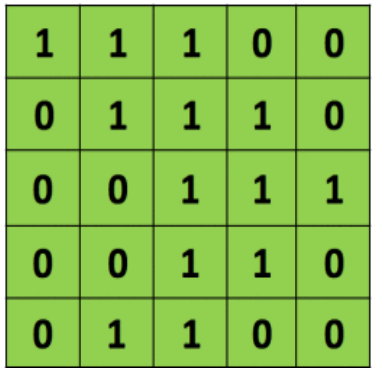
问题: 一幅 5×5 的图像有多少 3×3 的块?

答案: $5-(3+1\times 5)-(3+1=9.)$

卷积

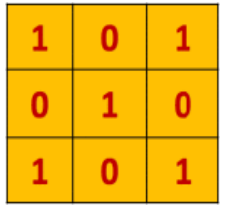
输入图像

5×5

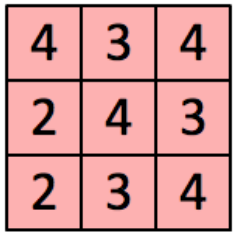


卷积核

3×3



卷积:



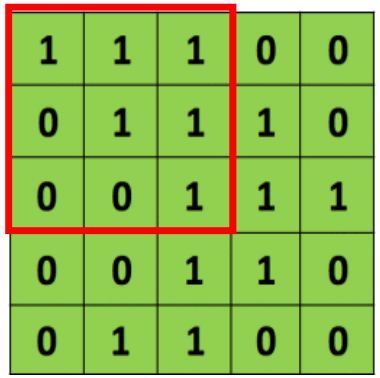
结果

3×3

卷积

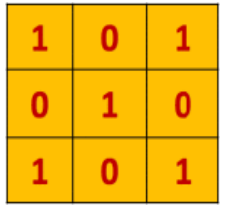
输入图像

5×5

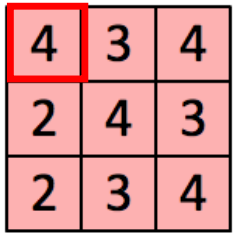


卷积核

3×3



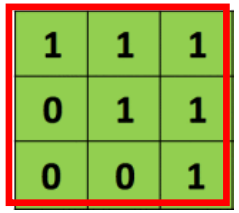
卷积:



结果

3×3

4 是



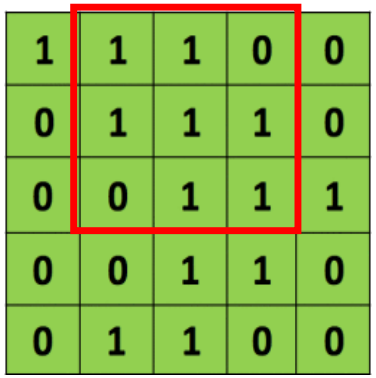
在卷积核

卷积下的值

卷积

输入图像

5×5

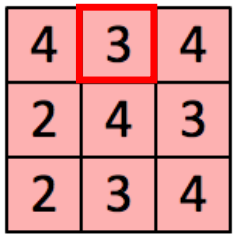


卷积核

3×3

1	0	1
0	1	0
1	0	1

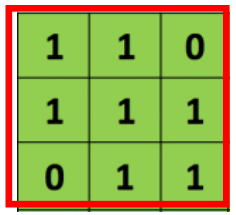
卷积:



结果

3×3

3 是



在卷积核

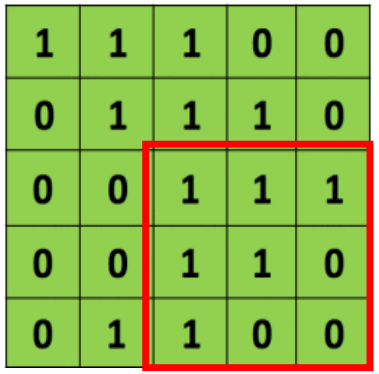
1	0	1
0	1	0
1	0	1

卷积下的值

卷积

输入图像

5×5



卷积核

3×3

1	0	1
0	1	0
1	0	1

卷积:

4	3	4
2	4	3
2	3	4

结果

3×3

4 是

1	1	1
1	1	0
1	0	0

在卷积核

1	0	1
0	1	0
1	0	1

卷积下的值

.

卷积

输入图像

5×5

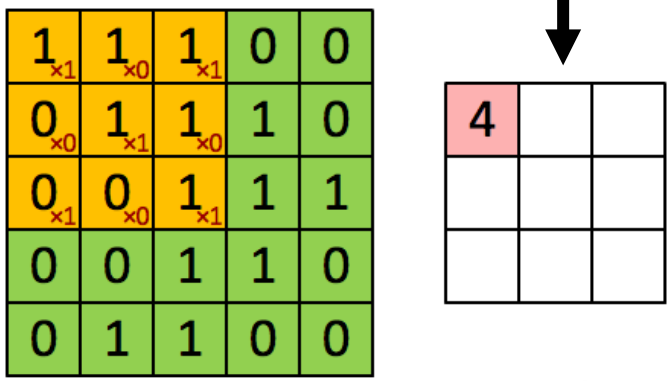
1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

卷积核

3×3

1	0	1
0	1	0
1	0	1

卷积:



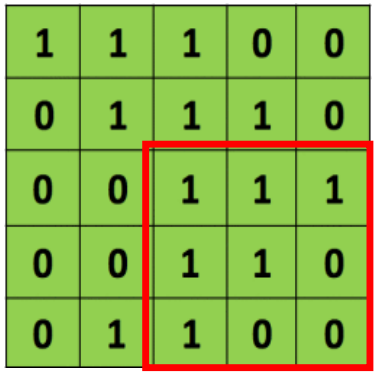
结果

3×3

卷积

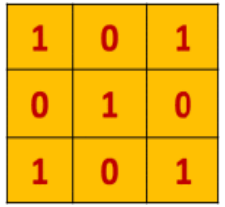
输入图像

5×5

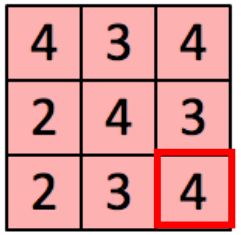


卷积核

3×3



卷积:



结果

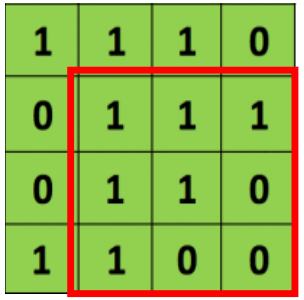
3×3

问题: 为什么结果是一个 3×3 的矩阵?

卷积

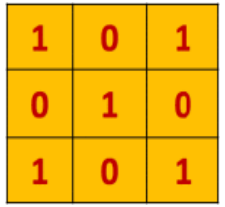
输入图像

4×4

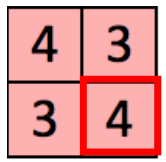


卷积核

3×3



卷积:



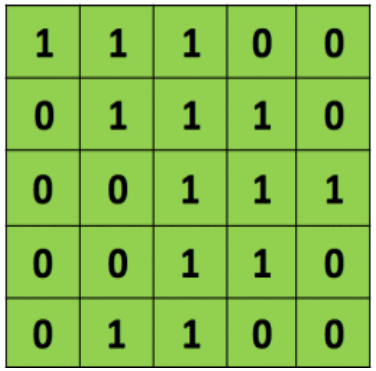
结果

2×2

卷积

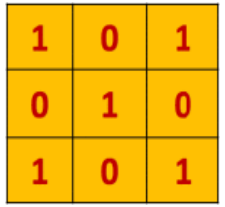
输入图像

5×5

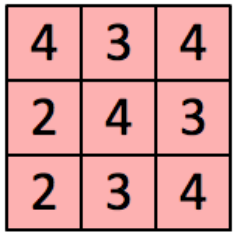


卷积核

3×3



卷积:



尺寸:

- 输入: $d_1 \times d_2$
- 卷积核: $k_1 \times k_2$
- 输出 : $(d_1 - k_1 + 1) \times (d_2 - k_2 + 1)$

卷积

输入图像	恒等卷积核	卷积核	输出
	Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	

卷积

输入图像		卷积核	输出
	边缘检测	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$ 和 $\begin{bmatrix} 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 \end{bmatrix}$ 的内积是0

$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$ 和 $\begin{bmatrix} 0.1 & 0.1 & 0.1 \\ 0.1 & 0.2 & 0.1 \\ 0.1 & 0.1 & 0.1 \end{bmatrix}$ 的内积是0.8

卷积

输入图像	锐化	卷积核	输出
	Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ 和 $\begin{bmatrix} 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 \end{bmatrix}$ 的内积是0.1

$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ 和 $\begin{bmatrix} 0.1 & 0.1 & 0.1 \\ 0.1 & 0.2 & 0.1 \\ 0.1 & 0.1 & 0.1 \end{bmatrix}$ 的内积是0.6

卷积

输入图像

	卷积核	输出 (或特征图)
盒状模糊 (归一化) **Box blur** (normalized)	$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\1 & 1 & 1\\1 & 1 & 1\end{bmatrix}$	
高斯模糊 (近似) **Gaussian blur** (approximation)	$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\2 & 4 & 2\\1 & 2 & 1\end{bmatrix}$	

卷积

卷积：零填充

输入图像

5×5

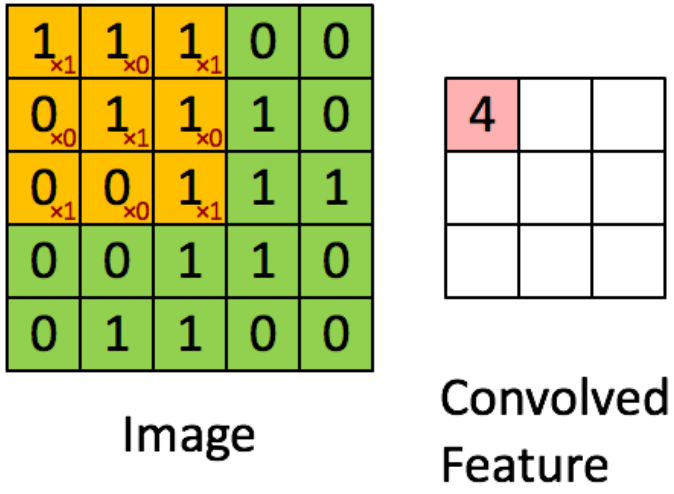
1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

卷积核

3×3

1	0	1
0	1	0
1	0	1

卷积:



尺寸:

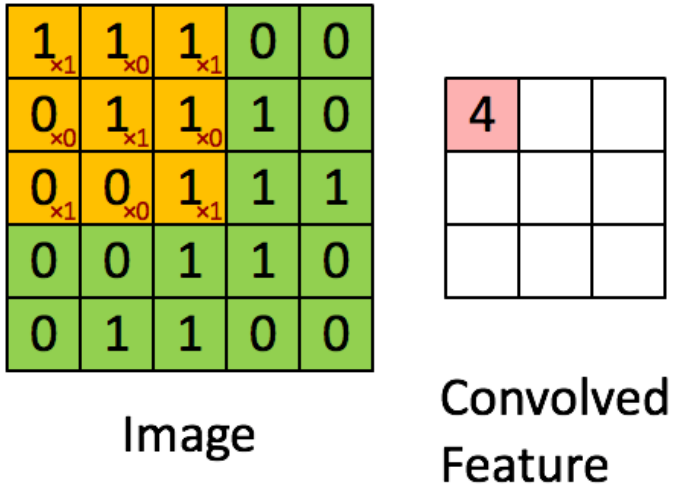
- 输入: $d_1 \times d_2$
- 卷积核: 3×3
- 输出 : $(d_1 - 2) \times (d_2 - 2)$

卷积：零填充

问题：输出比输入小！

如果输入是 28×28，滤波器是 3×3，你最多可以有 13 层卷积层。（第 13 层的输出是 2×2。）

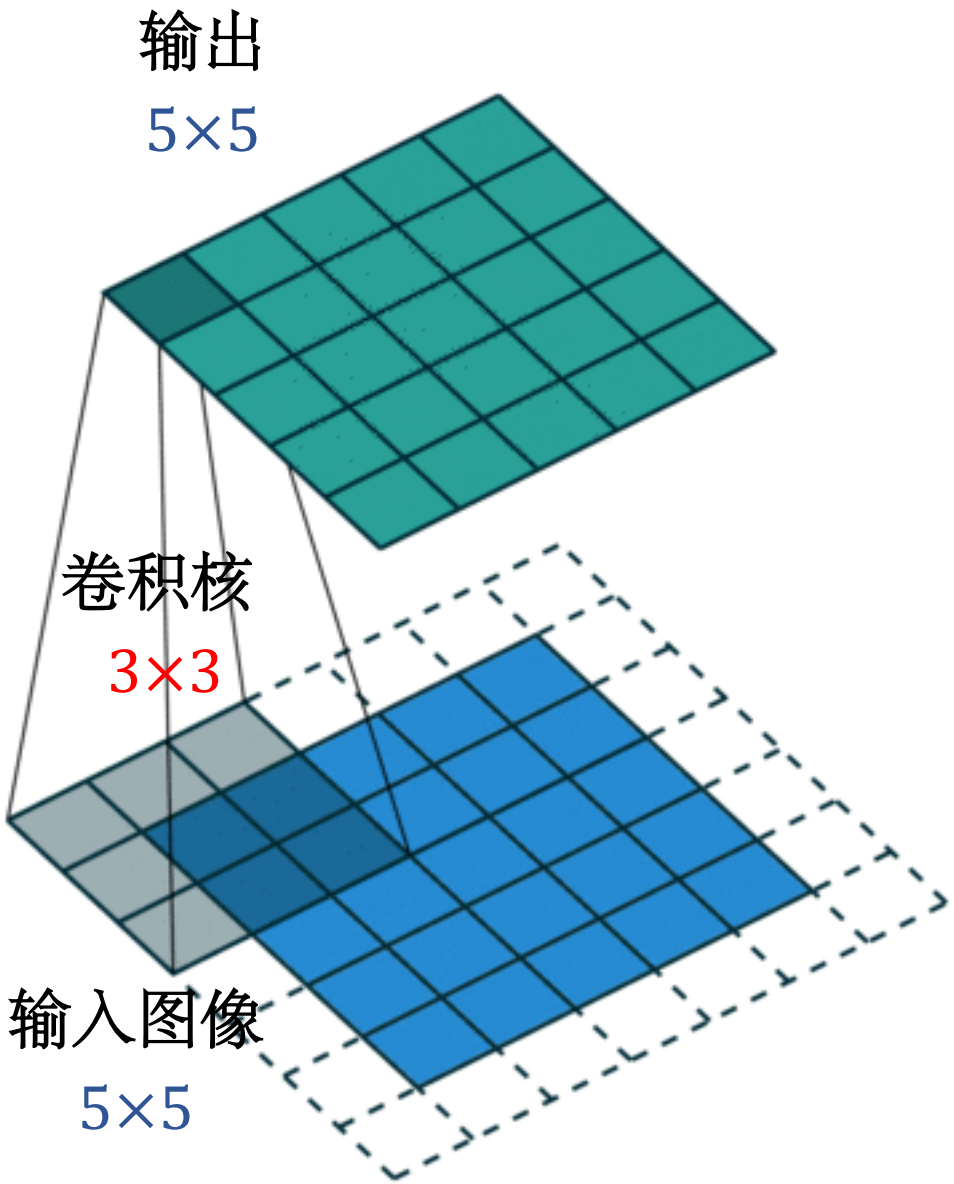
卷积：



尺寸：

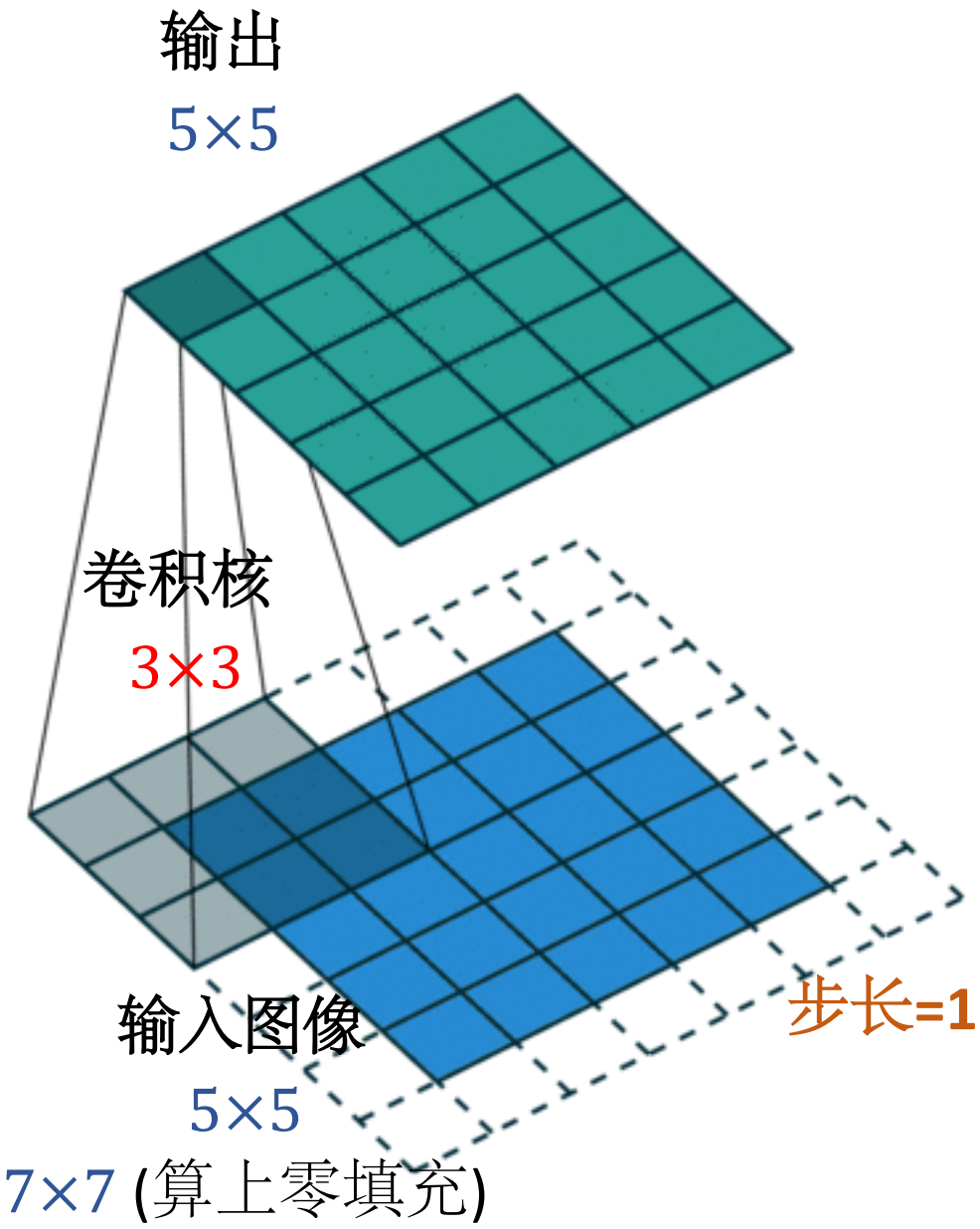
- 输入：$d_1 \times d_2$
- 卷积核：3×3
- 输出：$(d_1 - 2) \times (d_2 - 2)$

卷积：零填充



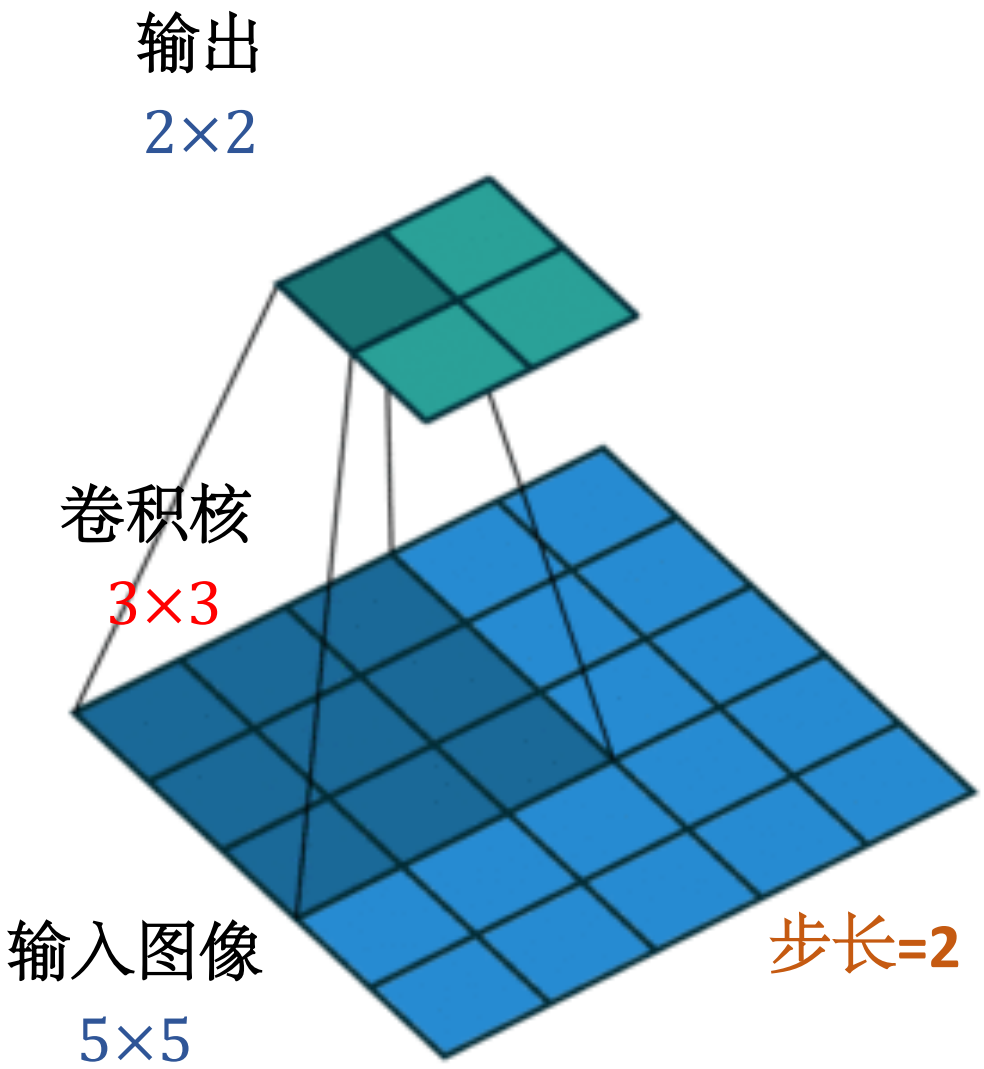
- 增加一个全0的边界
- 增加输入的尺寸：
  从 d1 × d2 到 (d1 + 2) × (d2 + 2)。
- 如果卷积核是 3×3, 那么输出是 $d_1 \times d_2$.

卷积：步长



- 在之前的案例中， 步长 是 1。
  - 卷积核每次移动 1 个步长。

卷积：步长

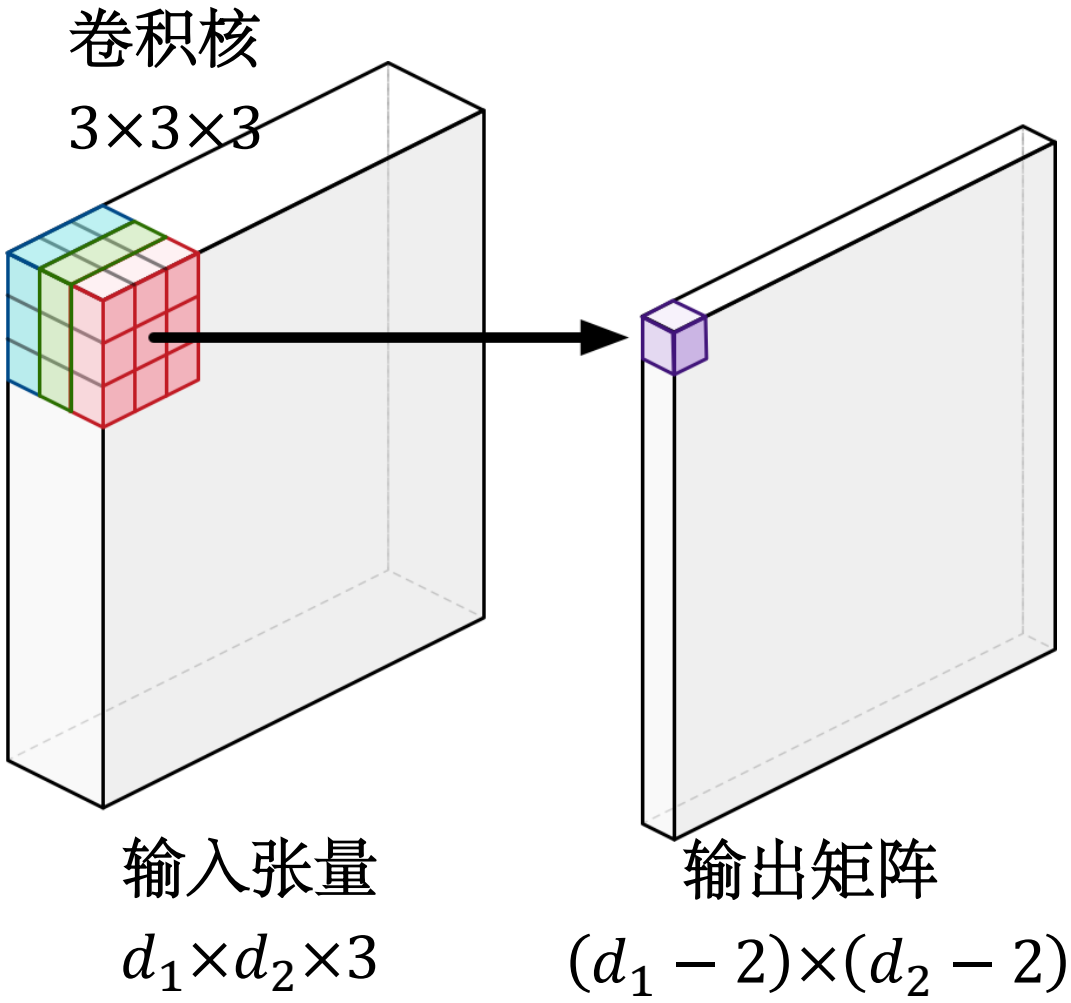


- 在之前的案例中， 步长 是 1。
  - 卷积核每次移动 1 个步长。
- 步长 也可以是 2 或者更长。

尺寸:

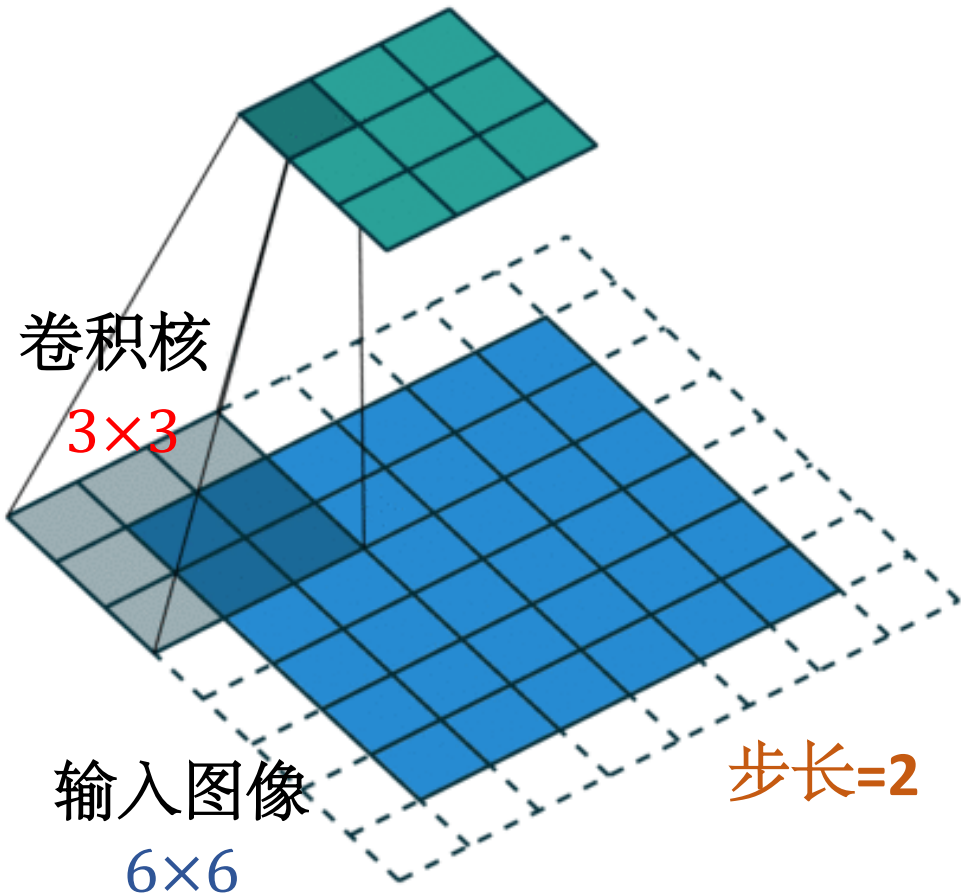
- 输入: $d_1 \times d_2$
- 卷积核: $k_1 \times k_2$
- 步长: s
- 输出: $\left(\left\lfloor \frac{d_1 - k_1}{s} \right\rfloor + 1\right) \times \left(\left\lfloor \frac{d_2 - k_2}{s} \right\rfloor + 1\right)$

张量的卷积



- 三阶张量：指三维数组，例如彩色图像（高度×宽度× 通道数，RGB 图像有 3 个通道）。
- 滤波器：3×3×3 表示滤波器匹配输入的 3 个通道，宽高为 3×3。
- 输出尺寸：由于 3×3 卷积核（步幅为 1，无填充），输出尺寸在宽高上各缩小 2。

卷积：关键概念

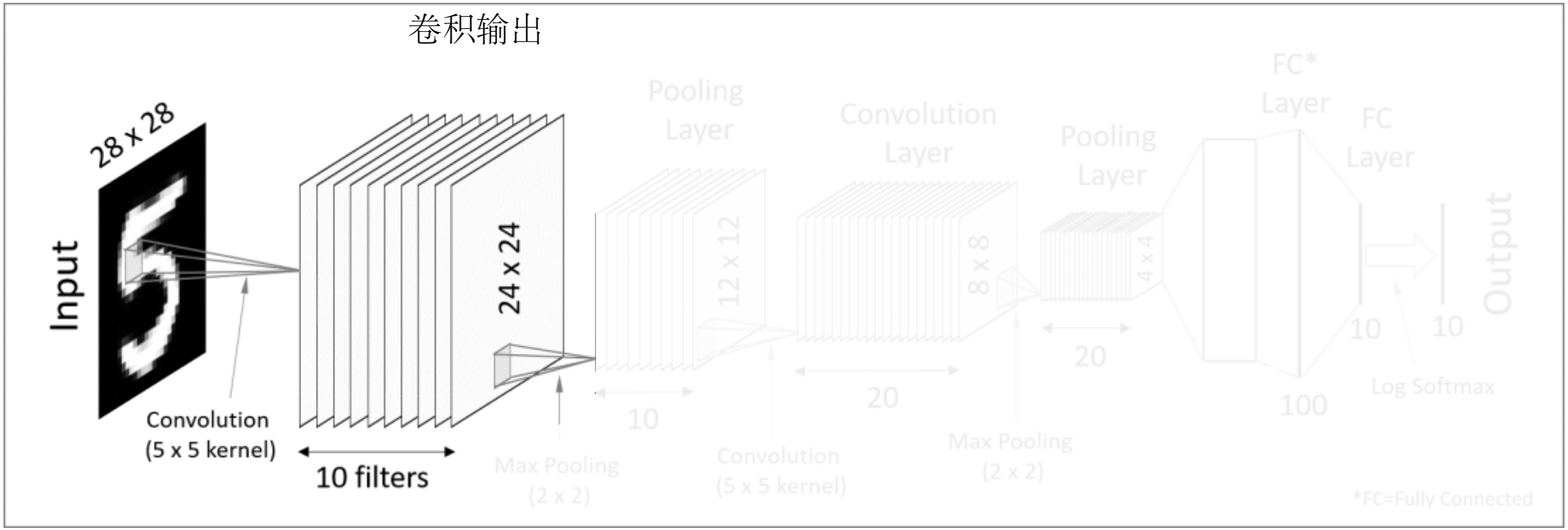


- 输入，卷积核，和输出
- 矩阵（即二阶张量）的卷积
- 三阶张量的卷积
- 零填充
- 步长

卷积神经网络 (CNNs)

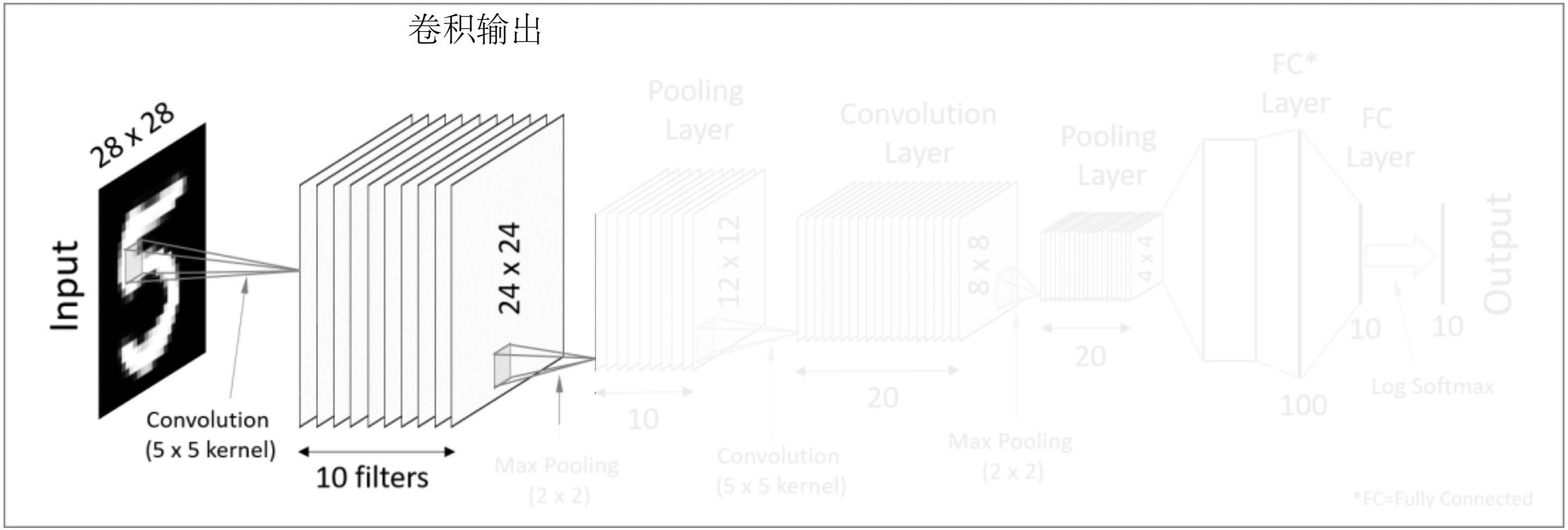
CNN: 卷积层 1

- 使用多个卷积核进行特征映射，例如
  - 使用十个 5×5 卷积核, (大概 $10\times5\times5 = 250$ 个参数!)
  - 结果是十个 24×24 矩阵.



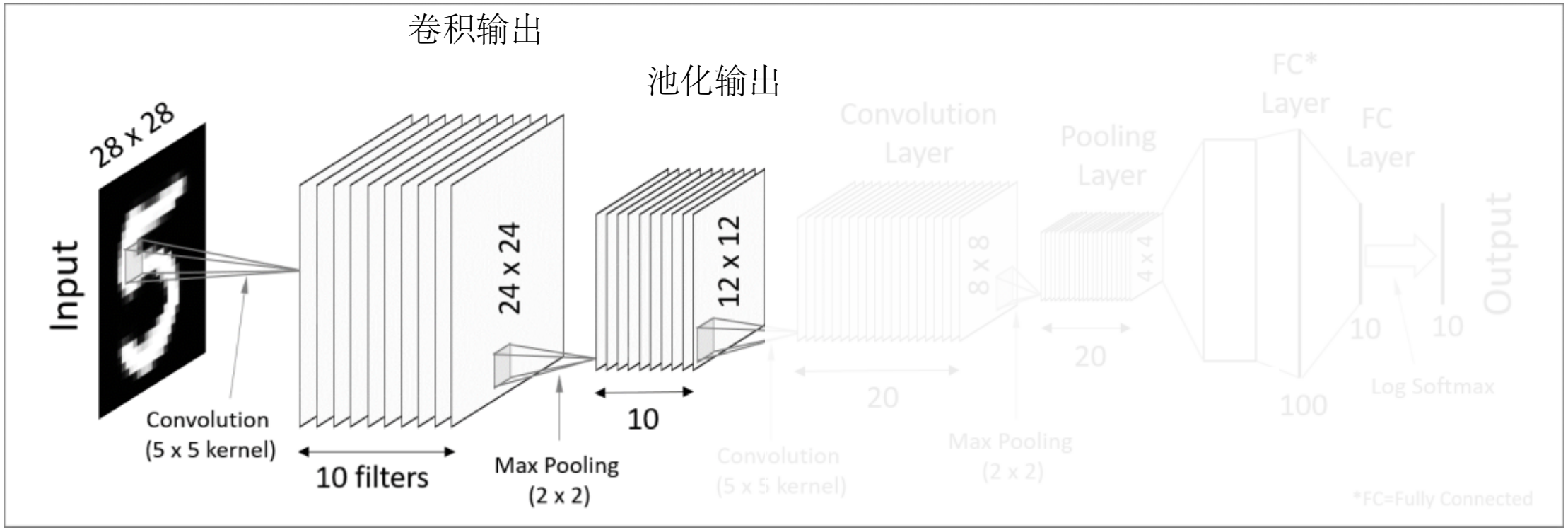
CNN: 卷积层 1

- 使用多个卷积核进行特征映射。
- 使用一个激活函数 (例如 ReLU).
- 一个卷积层 = 卷积 + 激活函数.



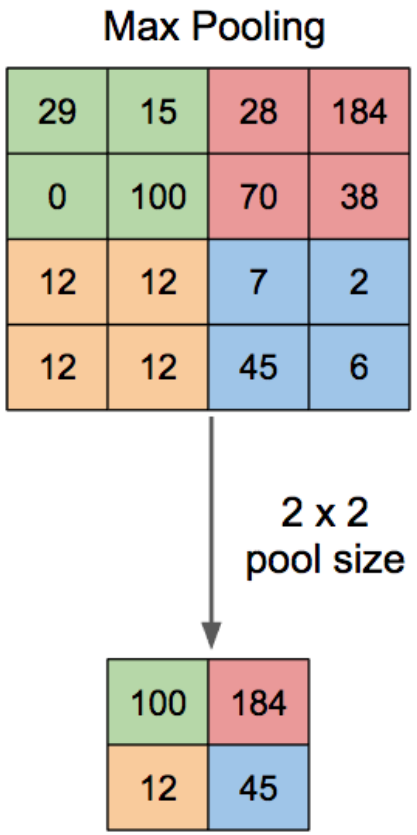
CNN: 卷积层 1

- 池化减少了每个特征图的纬度
- 例如最大池化，平均池化等



池化

- 池化减少了每个特征图的纬度
- 例如最大池化，平均池化等

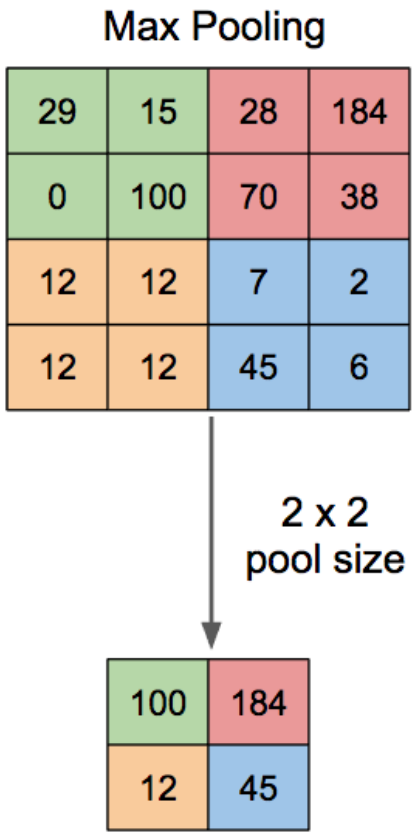


设置:

- 最大池化.
- 池尺寸 = 2×2.
- 无重叠 (步长 = 2×2).

池化

- 池化减少了每个特征图的纬度
- 例如最大池化，平均池化等



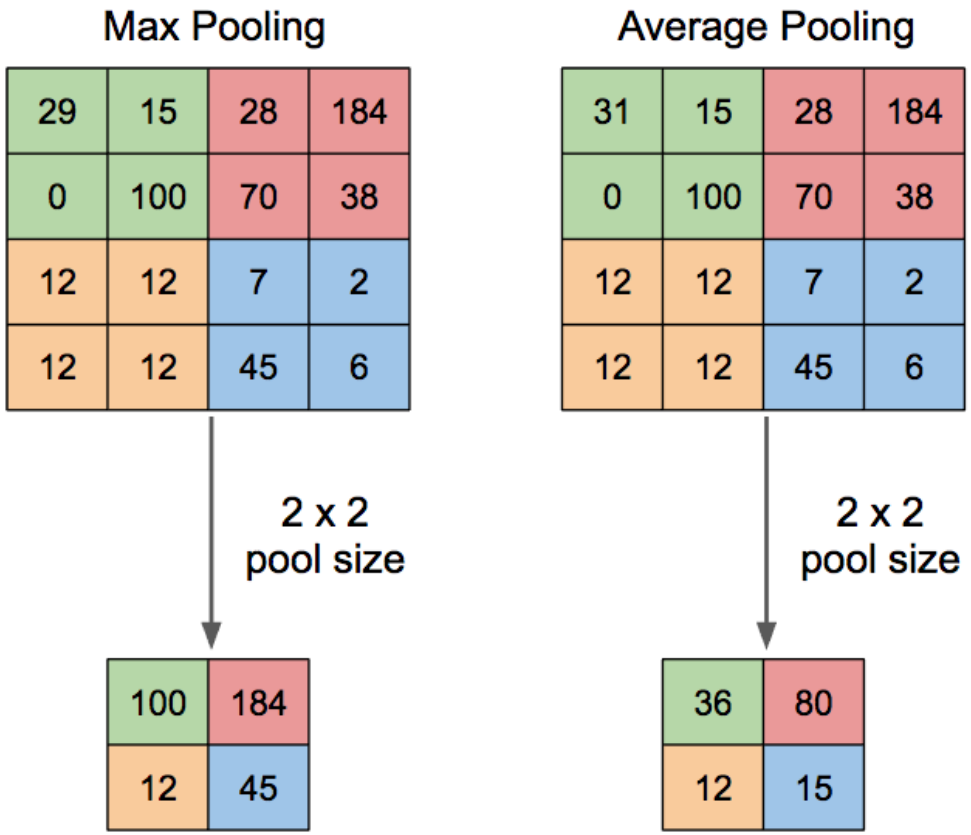
设置:

- 最大池化.
- 池尺寸 = 2×2.
- 无重叠 (步长 = 2×2).

问题: 如果步长是 1×1，结果如何?

池化

- 池化减少了每个特征图的纬度
- 例如最大池化，平均池化等



设置:

- 最大池化.
- 池尺寸 = 2×2.
- 无重叠 (步长 = 2×2).

池化

- 池化减少了每个特征图的纬度
- 例如最大池化，平均池化等

- **问题:** 输出形状如何?
  - 输入形状: 12×12.
  - 池尺寸: 3×3.
  - 无重叠 (步长: 3×3).

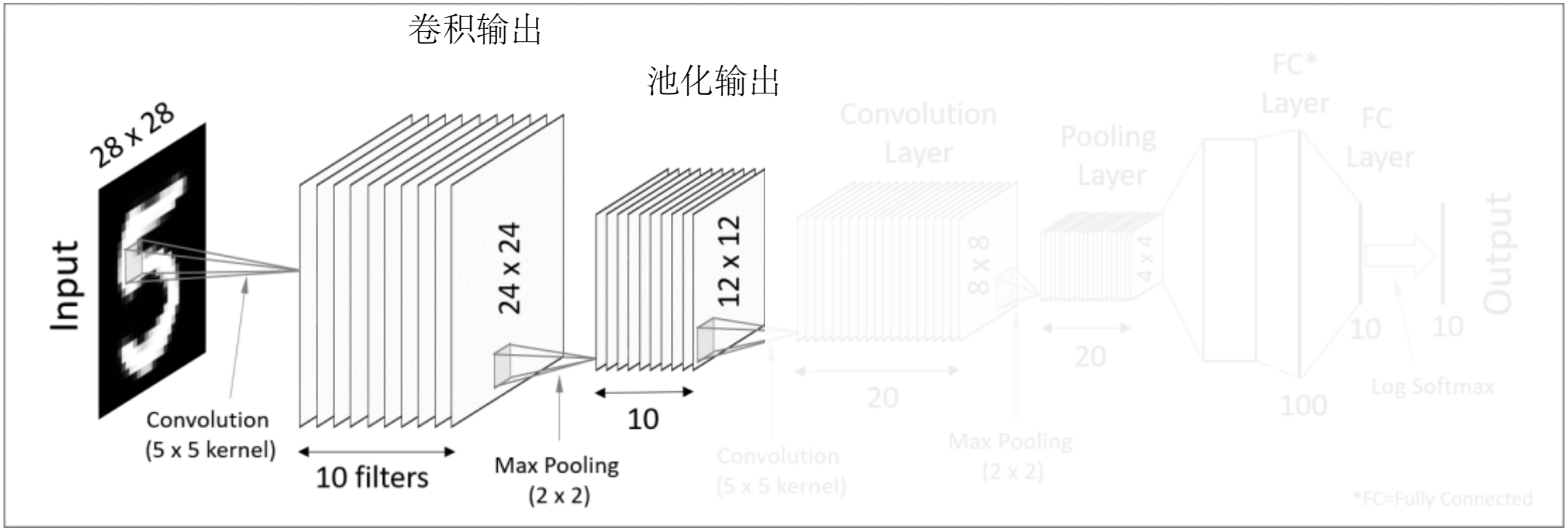
池化

- 池化减少了每个特征图的纬度
- 例如最大池化，平均池化等

- 问题: 输出形状如何?
  - 输入形状: 12×12.
  - 池尺寸: 3×3.
  - 无重叠 (步长: 1×1).

CNN: 池化层 1

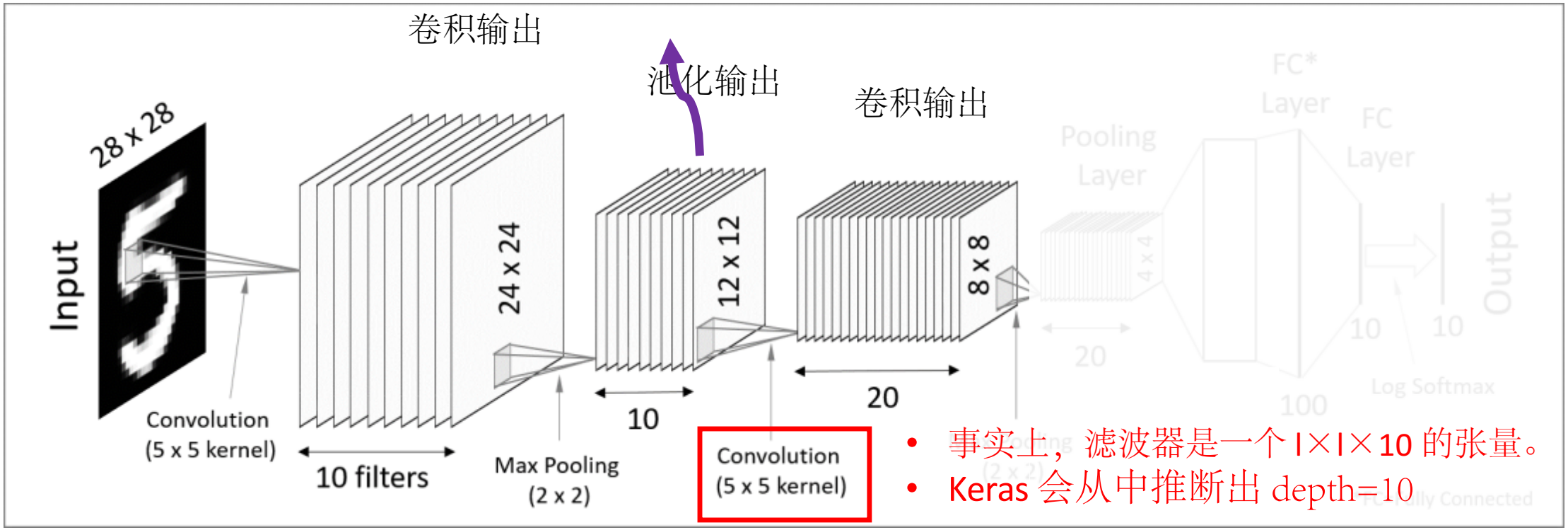
- 池化减少了每个特征图的纬度
- 例如最大池化，平均池化等



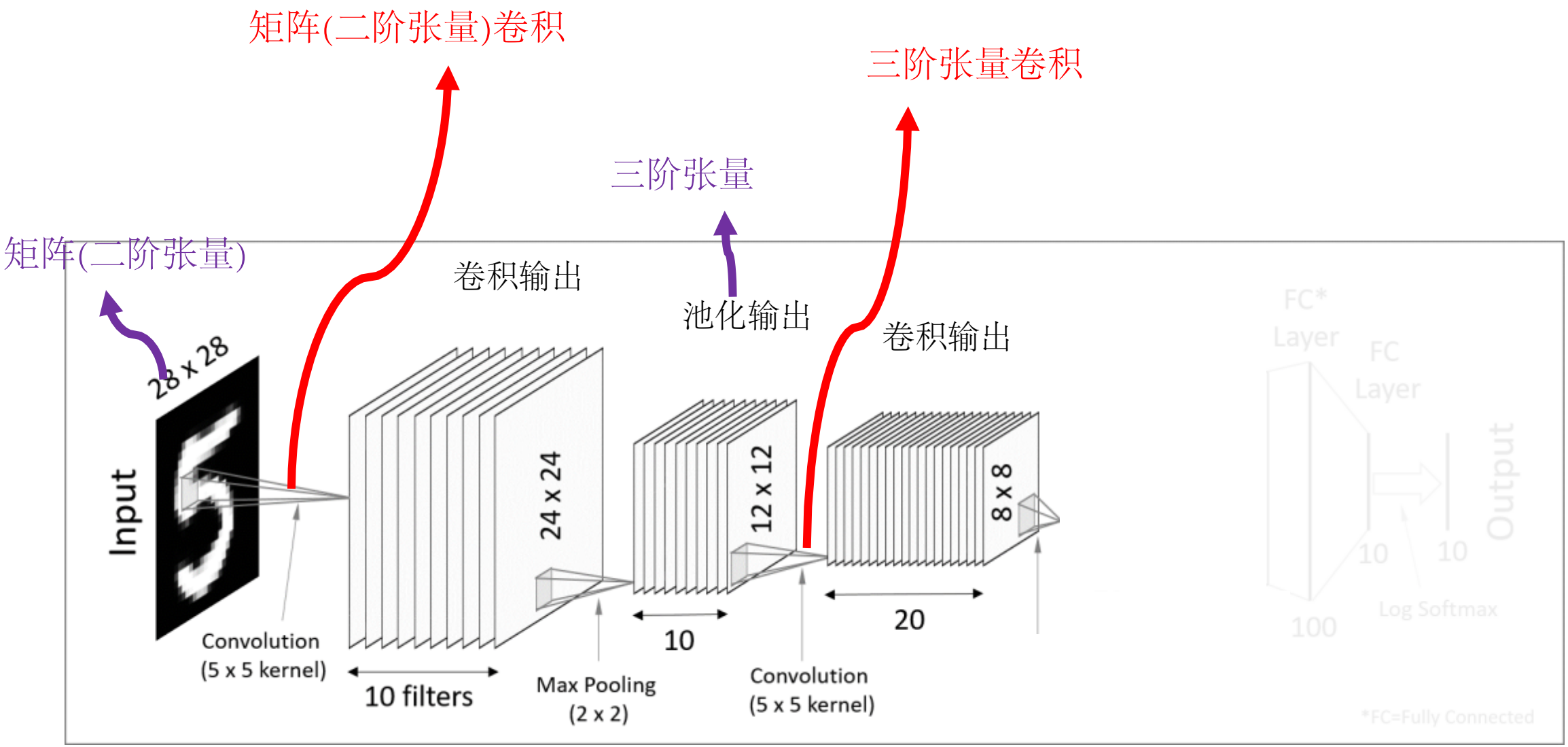
CNN: 卷积层 2

- 执行张量卷积
- 将 5×5×10 的卷积核用于 12×12×10 的张量输入
- 20 个这样的卷积核 ➔ 20 个 8×8 特征图

12×12×10 三阶张量

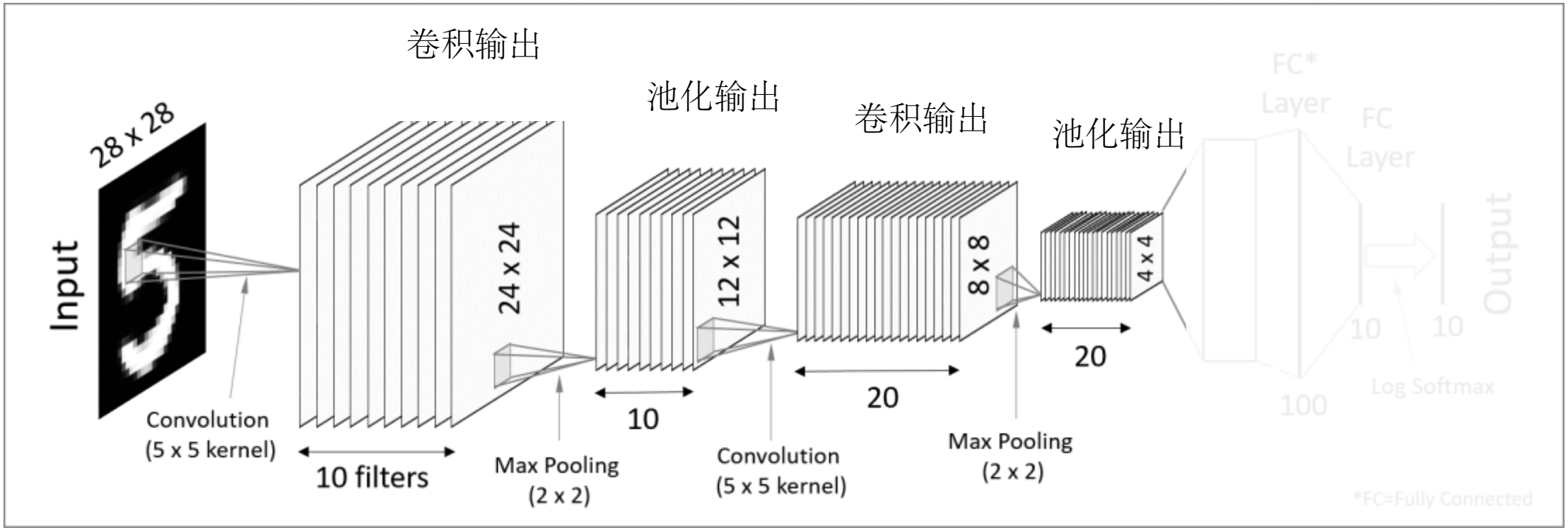


CNN: 卷积层


矩阵(二阶张量)卷积
三阶张量卷积
三阶张量
矩阵(二阶张量)
卷积输出
池化输出
卷积输出
28 x 28
Input
Convolution
(5 x 5 kernel)
10 filters
24 x 24
Max Pooling
(2 x 2)
10
12 x 12
Convolution
(5 x 5 kernel)
20
8 x 8
FC*
Layer
FC
Layer
Output
10
10
100
Log Softmax
*FC=Fully Connected

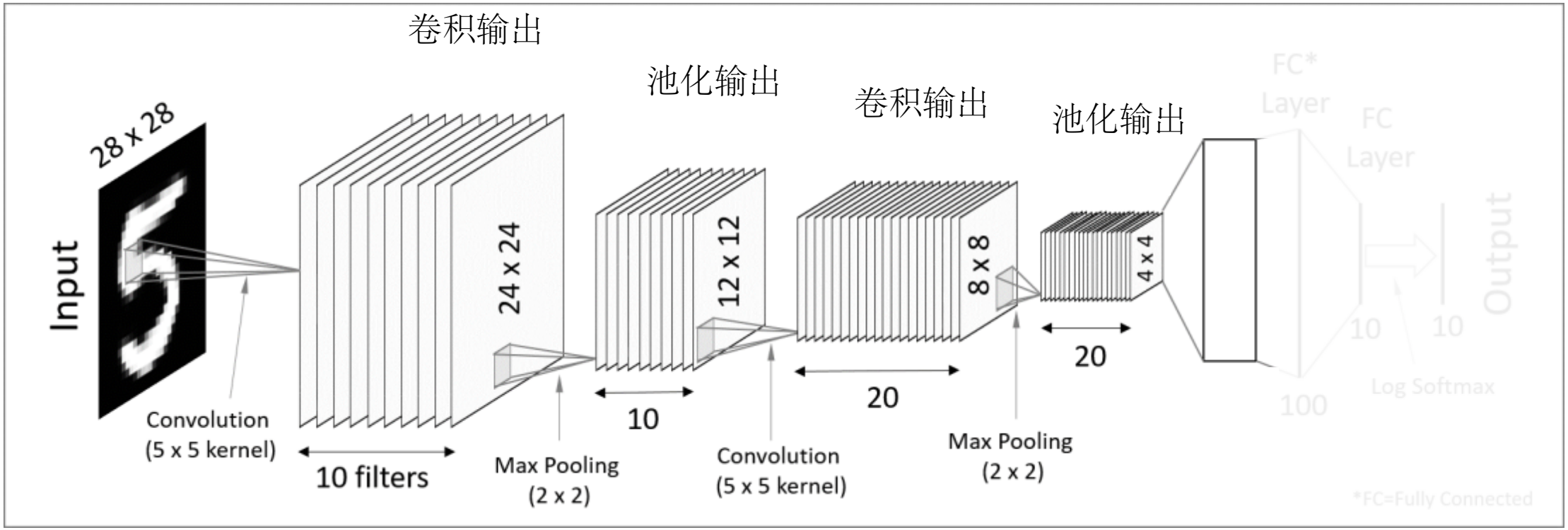
CNN: 池化层 2

- 将 2×2 的最大池化层应用在一个 8×8 的矩阵 ➔ 一个 4×4 的矩阵



CNN: 展平 (向量化)

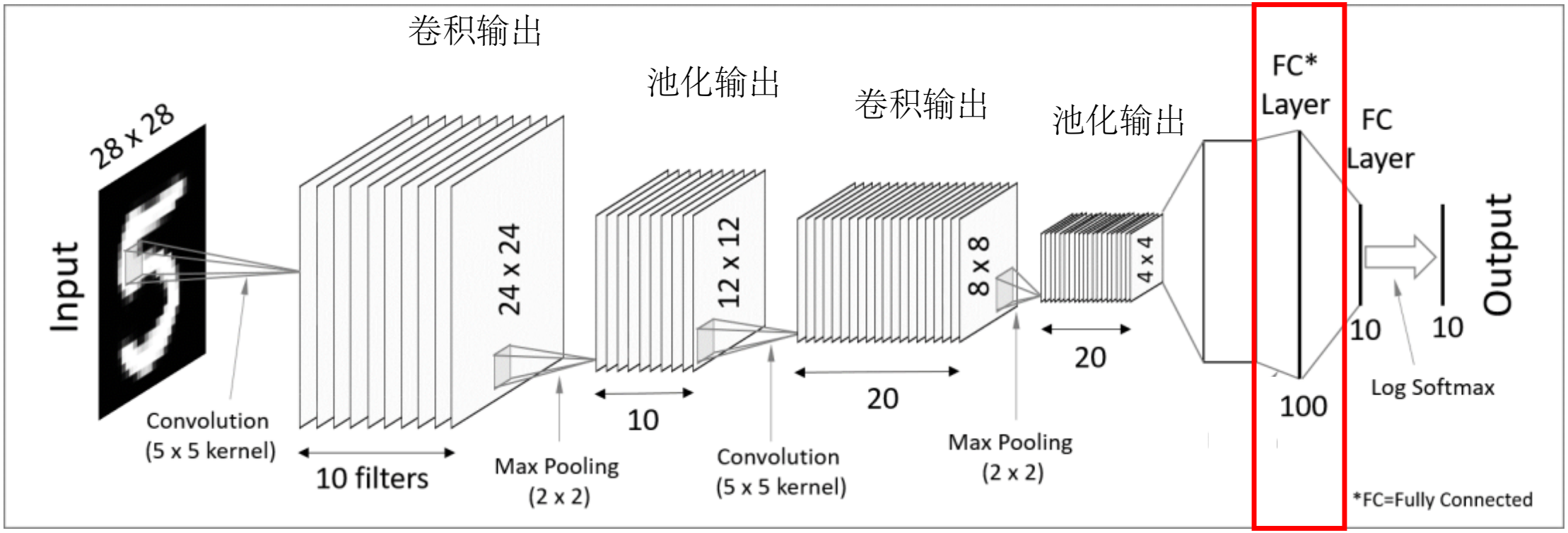
- 将 4×4×20 的张量展平
- 得到一个 320 纬的 向量



CNN: 全连接层

- 增加一层或者多层全连接层

这一层是可选的



CNN: 有多少个可训练的参数?

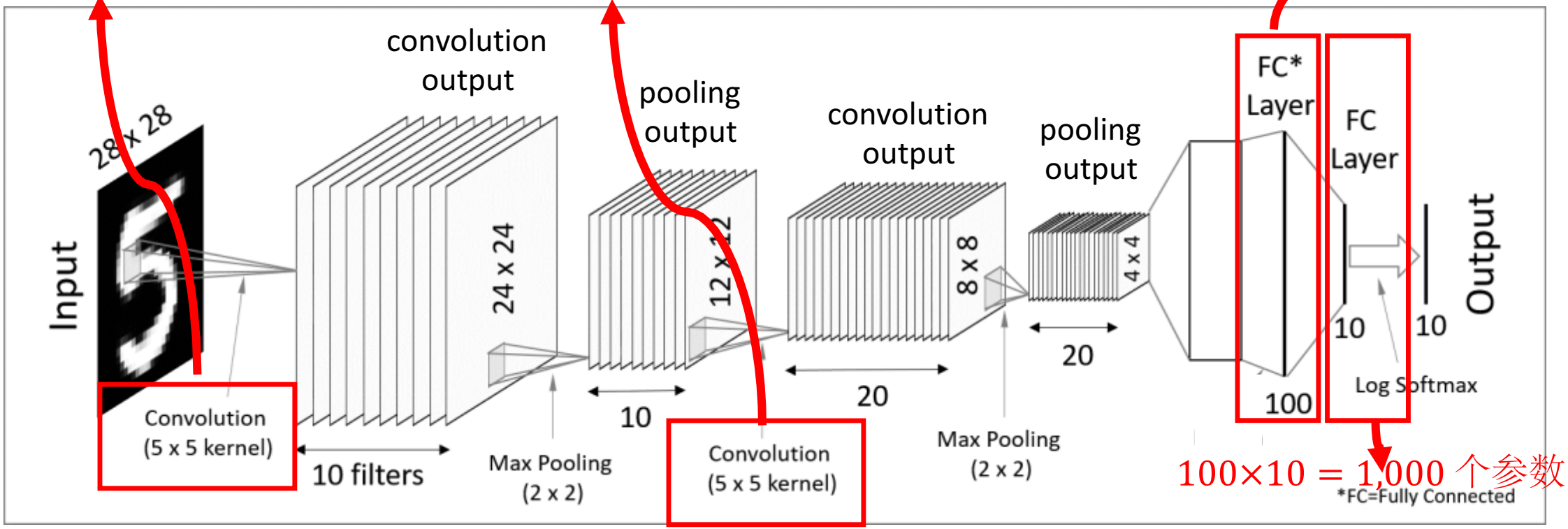
$10\times5\times5 = 250$ 个参数

- 10: 卷积核的数量
- 5×5: 卷积核尺寸

$20\times5\times5\times10 = 5000$ 个参数

- 20: 卷积核的数量
- 5×5×10: 卷积核尺寸

$320\times100 = 32{,}000$ 个参数



$100\times10 = 1{,}000$ 个参数

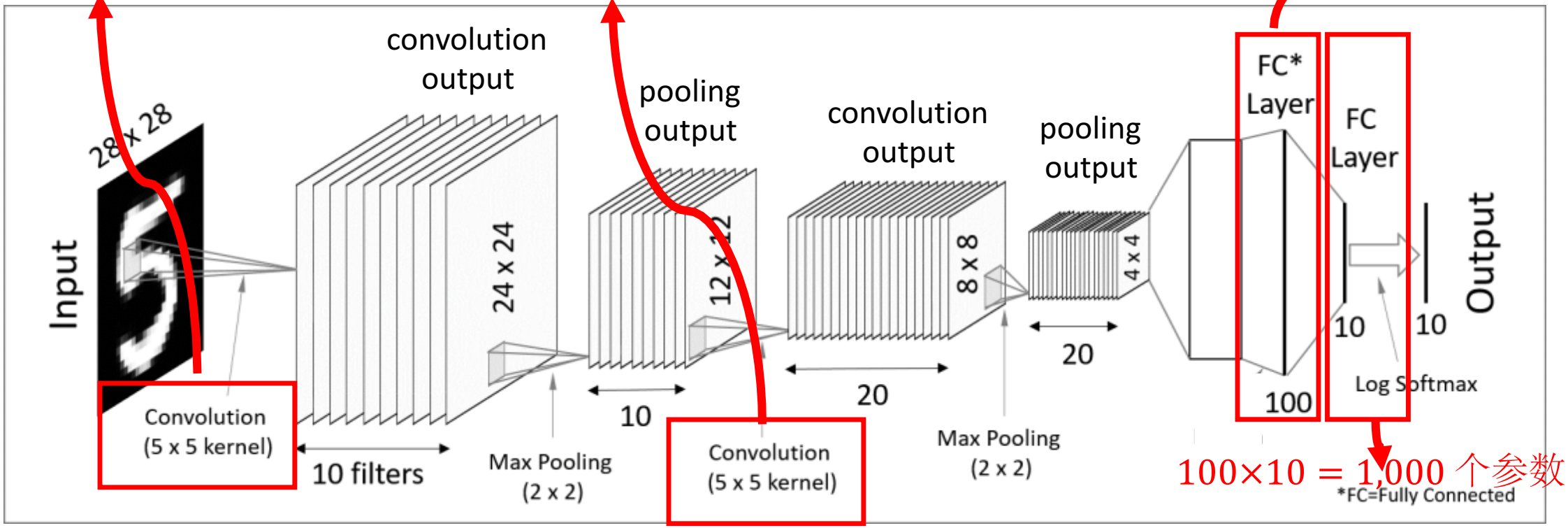
CNN: 有多少个可训练的参数?

答案: > 38,250 个参数 （不包括偏置项 bias ）

$10\times5\times5 = 250$ 个参数
- 10: 卷积核的数量
- 5×5: 卷积核尺寸

$20\times5\times5\times10 = 5000$ 个参数
- 20: 卷积核数量
- 5×5: 卷积核尺寸
- 10: 卷积核深度

$320\times100 = 32{,}000$ 个参数

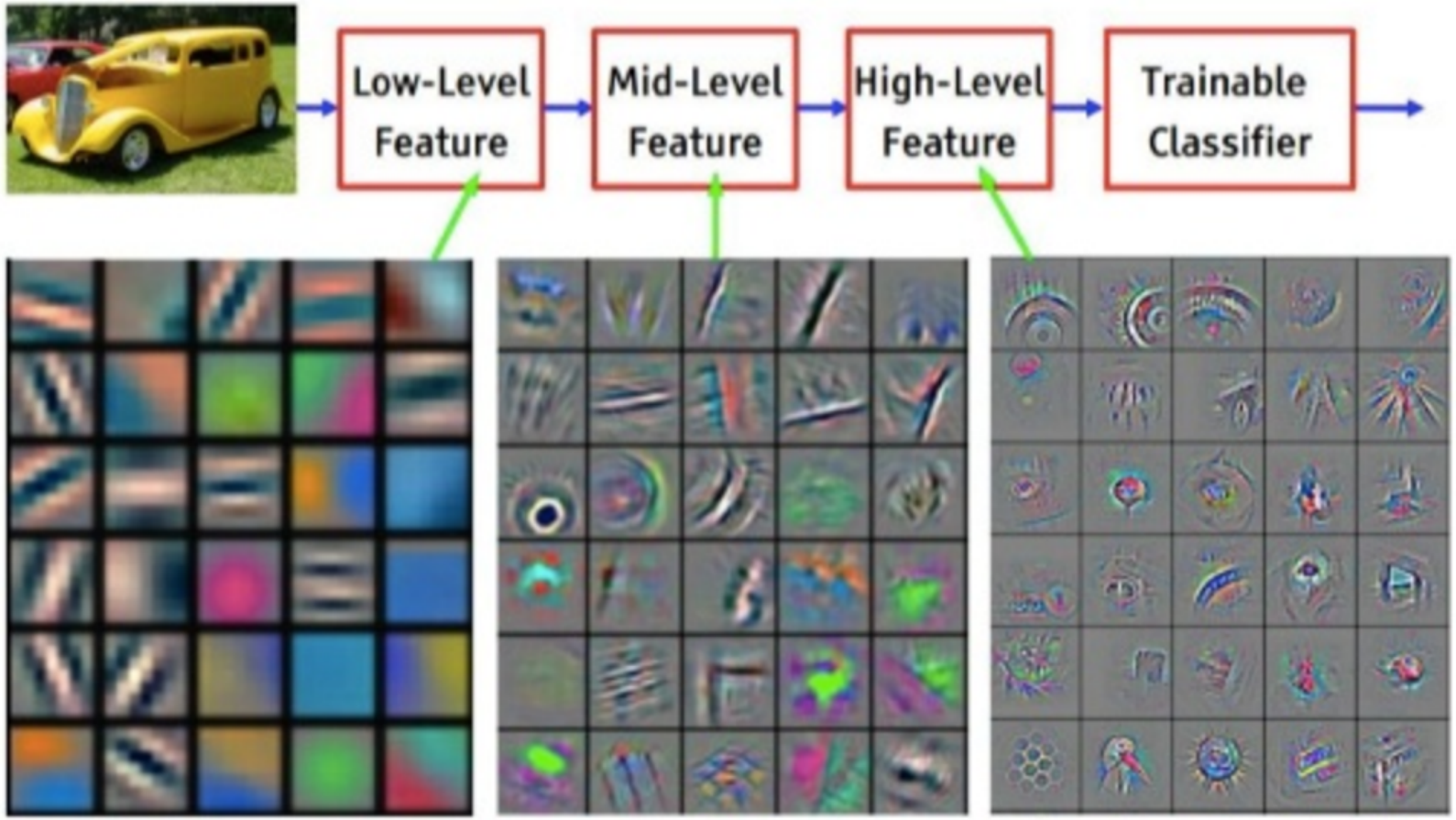


$100\times10 = 1{,}000$ 个参数

CNN: 回顾

问：为什么要有卷积层?

答：检测边缘、形状、模式等特征。



CNN: 回顾

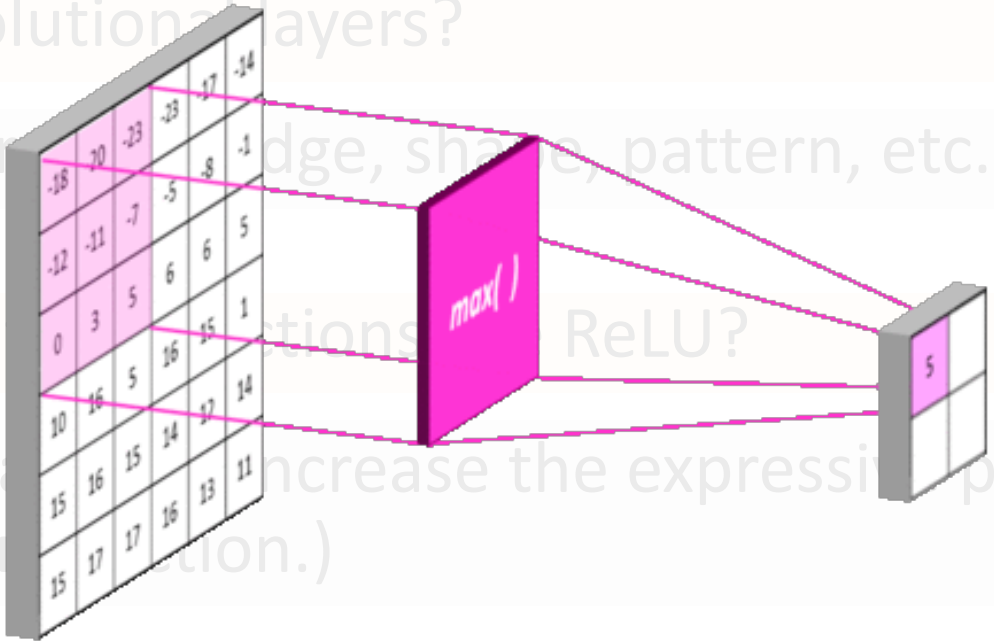
问：为什么要有卷积层?

答：检测边缘、形状、模式等特征。

问题: 为什么要有激活函数，如ReLU?

Answer: 增加非线性以提升表达能力。
(注意，卷积是一个线性函数。)

CNN: 回顾



问 : 为什么要有池化层?

答: 显著减少特征大小

CNN的超参数

网络结构的超参数

- 卷积层
  - # 卷积核
  - 卷积核大小
  - 步长
  - 是否零填充
- 池化层
  - 最大池化法 or 平均池化法
  - 池尺寸
  - 池化步长
- 全连接层
  - 宽度
- 激活函数
  - ReLU
  - SoftMax
  - Sigmoid

训练的超参数

- 损失函数
  - 用于分类的交叉赛
  - 用于回归的L1 或 L2（标签是连续的）
- 优化算法（及其超参数，例如学习率）
  - SGD
  - 带动量的 SGD
  - AdaGrad
  - RMSprop
- 随机初始化
  - 高斯分布/均匀分布
  - Scaling

用 keras 实现一个 CNN

1. 加载并处理 MNIST 数据集

加载数据

```
from keras.datasets import mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()

print('Shape of x_train: ' + str(x_train.shape))
print('Shape of x_test: ' + str(x_test.shape))
print('Shape of y_train: ' + str(y_train.shape))
print('Shape of y_test: ' + str(y_test.shape))
```

```
Shape of x_train: (60000, 28, 28)
Shape of x_test: (10000, 28, 28)
Shape of y_train: (60000,)
Shape of y_test: (10000,)
```

1. Load and Process the MNIST Dataset

Reshape: convert the $60000 \times 28 \times 28$ dataset to a $60000 \times 28 \times 28 \times 1$ tensor.

```
x_train_vec = x_train.reshape((60000, 28, 28, 1)) / 255.0
x_test_vec = x_test.reshape((10000, 28, 28, 1)) / 255.0

print('Shape of x_train_vec is ' + str(x_train_vec.shape))
```

```
Shape of x_train_vec is (60000, 28, 28, 1)
```

1. Load and Process the MNIST Dataset

One-hot encode: convert the labels (scalars in $\{0, 1, \cdots, 9\}$) to 10-dim vectors.

```
def to_one_hot(labels, dimension=10):
    results = np.zeros((len(labels), dimension))
    for i, label in enumerate(labels):
        results[i, label] = 1.
    return results

y_train_vec = to_one_hot(y_train)
y_test_vec = to_one_hot(y_test)
print('Shape of y_train_vec is ' + str(y_train_vec.shape))
```

```
Shape of y_train_vec is (60000, 10)
```

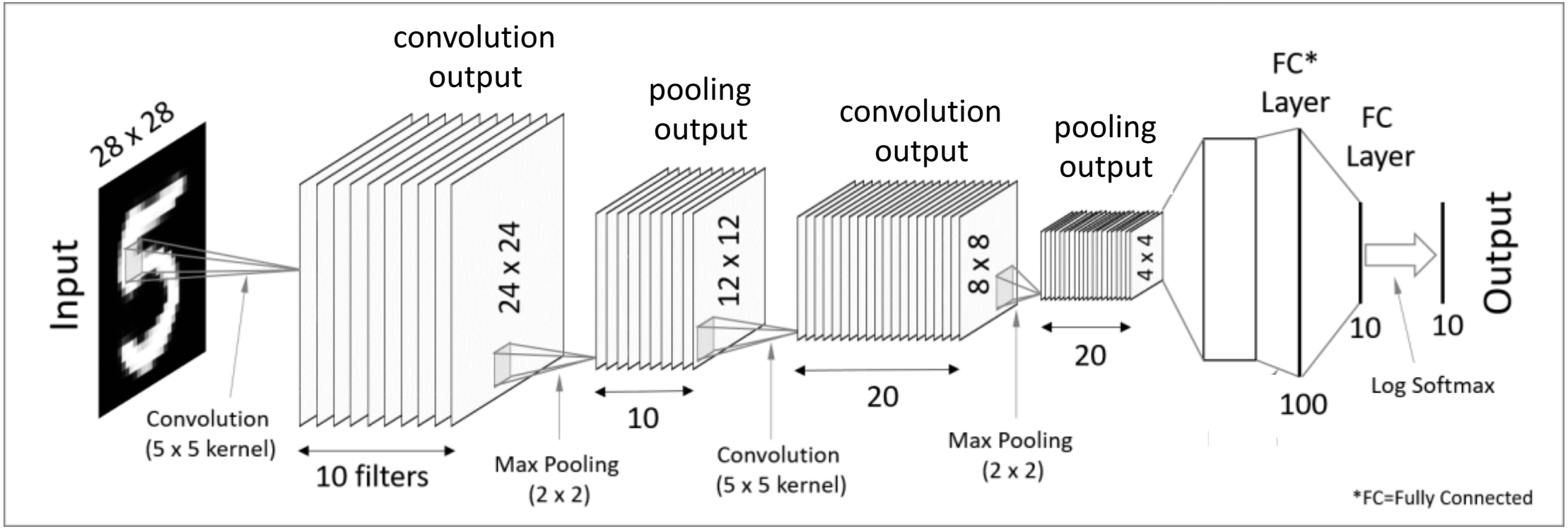
1. Load and Process the MNIST Dataset

Partition the **training** set to **training** and **validation** sets.

```
# Partition to training and validation sets

rand_indices = np.random.permutation(60000)
train_indices = rand_indices[0:50000]
valid_indices = rand_indices[50000:60000]

x_valid_vec = x_train_vec[valid_indices, :, :, :]
y_valid_vec = y_train_vec[valid_indices, :]

x_train_vec = x_train_vec[train_indices, :, :, :]
y_train_vec = y_train_vec[train_indices, :]

print('Shape of x_valid_vec: ' + str(x_valid_vec.shape))
print('Shape of y_valid_vec: ' + str(y_valid_vec.shape))
print('Shape of x_train_vec: ' + str(x_train_vec.shape))
print('Shape of y_train_vec: ' + str(y_train_vec.shape))
```

```
Shape of x_valid_vec: (10000, 28, 28, 1)
Shape of y_valid_vec: (10000, 10)
Shape of x_train_vec: (50000, 28, 28, 1)
Shape of y_train_vec: (50000, 10)
```

2. Build the CNN



2. Build the CNN

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Conv2D(10, (5, 5), activation='relu', input_shape=(28, 28, 1)))
```

Filter number

Filter shape

Default

- Stride: 1
- Zero-padding: no padding

2. Build the CNN

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Conv2D(10, (5, 5), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
```

pool_size

Default

- Stride: pool_size

2. Build the CNN

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Conv2D(10, (5, 5), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))  Output: 12×12×10
model.add(layers.Conv2D(20, (5, 5), activation='relu'))  Output: 8×8×20
model.add(layers.MaxPooling2D((2, 2)))  Output: 4×4×20
```

2. Build the CNN

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Conv2D(10, (5, 5), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))  Output: 12×12×10
model.add(layers.Conv2D(20, (5, 5), activation='relu'))  Output: 8×8×20
model.add(layers.MaxPooling2D((2, 2)))  Output: 4×4×20
model.add(layers.Flatten())  Output: 320
```

2. Build the CNN

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Conv2D(10, (5, 5), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(20, (5, 5), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

Output: 12×12×10

Output: 8×8×20

Output: 4×4×20

Output: 320

Output: 100

Output: 10

2. Build the CNN

```
# print the summary of the model.
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 24, 24, 10)        260
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 10)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 20)          5020
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 20)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 320)               0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               32100
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1010
=================================================================
Total params: 38,390
Trainable params: 38,390
Non-trainable params: 0
_________________________________________________________________
```

3. Train the CNN

Specify: optimization algorithm, learning rate (LR), loss function, and metric.

```
from keras import optimizers
model.compile(optimizers.RMSprop(lr=0.0001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

3. Train the CNN

Specify: batch size and number of epochs.

```
history = model.fit(x_train_vec, y_train_vec,
                    batch_size=128, epochs=50,
                    validation_data=(x_valid_vec, y_valid_vec))
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/50
50000/50000 [==============================] - 12s 239us/step - loss: 1.2455 - acc: 0.7201 - val_loss: 0.4805 - val_acc: 0.8707
Epoch 2/50
50000/50000 [==============================] - 12s 235us/step - loss: 0.3563 - acc: 0.8995 - val_loss: 0.2851 - val_acc: 0.9172
Epoch 3/50
50000/50000 [==============================] - 12s 244us/step - loss: 0.2474 - acc: 0.9271 - val_loss: 0.2230 - val_acc: 0.9365
```

-
-
-

```
Epoch 49/50
50000/50000 [==============================] - 12s 240us/step - loss: 0.0244 - acc: 0.9924 - val_loss: 0.0439 - val_acc: 0.9875
Epoch 50/50
50000/50000 [==============================] - 12s 239us/step - loss: 0.0238 - acc: 0.9927 - val_loss: 0.0446 - val_acc: 0.9881
```

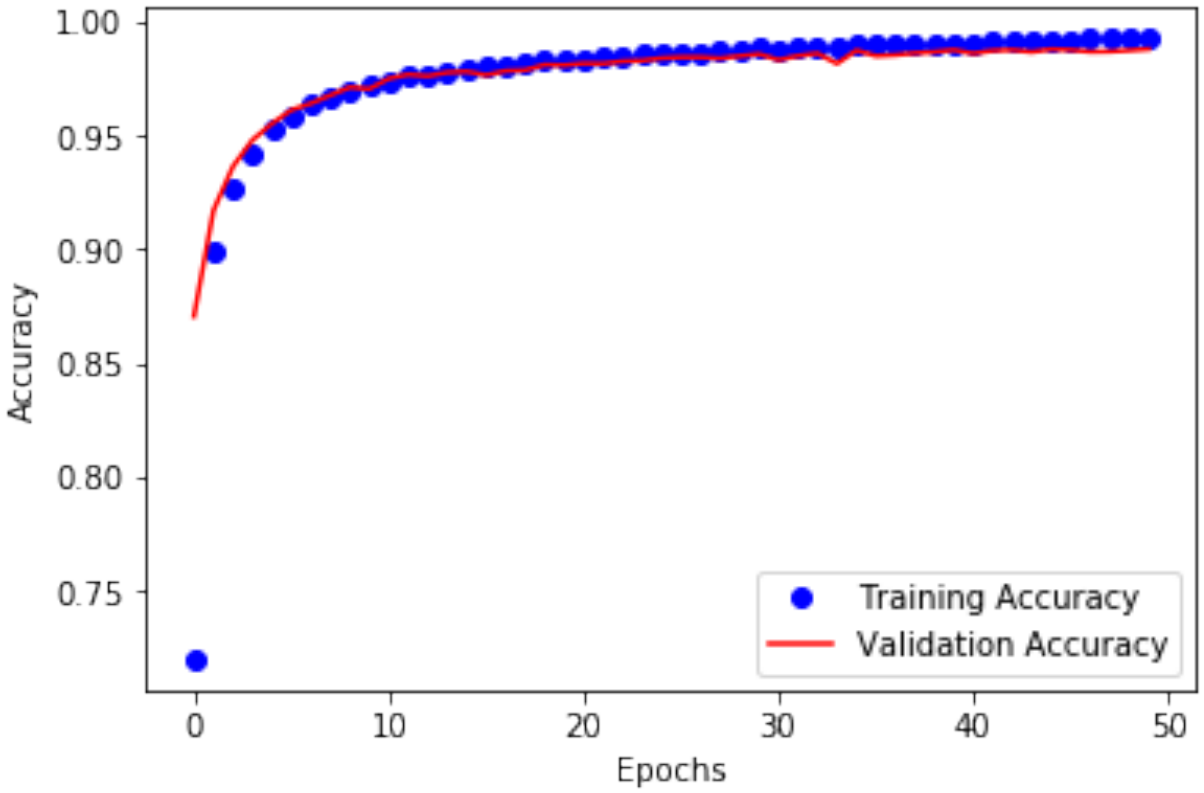
4. Examine the Results

Plot the ***accuracy*** against ***epochs*** (1 epoch = 1 pass over the data).

```
import matplotlib.pyplot as plt
%matplotlib inline

epochs = range(50) # 50 is the number of epochs
train_acc = history.history['acc']
valid_acc = history.history['val_acc']
plt.plot(epochs, train_acc, 'bo', label='Training Accuracy')
plt.plot(epochs, valid_acc, 'r', label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

4. Examine the Results

Plot the ***accuracy*** against ***epochs*** (1 epoch = 1 pass over the data).



Why using the validation set?

- Tune the hyper-parameters, e.g., learning rate, filter shape, filter number, etc.
- Check overfitting or underfitting.

4. Examine the Results

Evaluate the model on the ***test*** set. (So far, the model has not seen the ***test*** set.)

```
loss_and_acc = model.evaluate(x_test_vec, y_test_vec)
print('loss = ' + str(loss_and_acc[0]))
print('accuracy = ' + str(loss_and_acc[1]))
```

```
10000/10000 [==============================] - 1s 127us/step
loss = 0.035708001047297 15
accuracy = 0.9883
```

4. Examine the Results

- Training accuracy: 99.3%
- Validation accuracy: 98.8%
- Test accuracy: 98.8%

Summary

1. 卷积

- 矩阵卷积和张量卷积
- 概念:
  - 卷积核
  - 块
  - 输出（特征图）
  - 零填充
  - 步长

2. 卷积神经网络

- 卷积层
  - 卷积核数量
  - 卷积核尺寸
  - 步长（默认 1）
  - 零填充（默认无）

2. 卷积神经网络

- 卷积层
  - 卷积核数量
  - 卷积尺寸
  - 步长（默认 1）
  - 零填充（默认无）
- 激活函数
- 池化层
  - 最大池化、平均池化等。
  - 池尺寸
  - 池步长

3. 使用 Keras 构建和训练（CNN）

- 搭建神经网络
  - 添加卷积层、池化层和全连接层。
- 编译模型
  - 指定优化算法、损失函数和评估指标。
- 在训练数据上拟合模型。