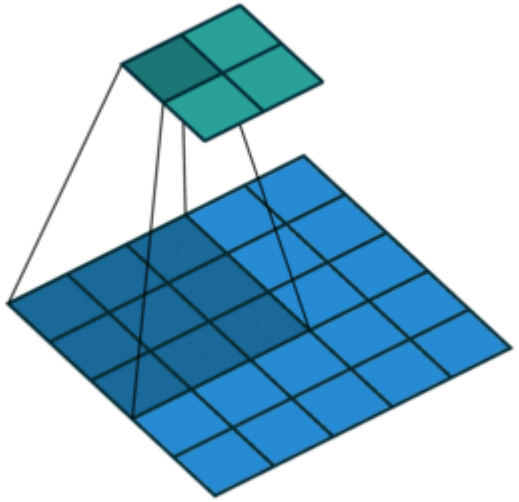
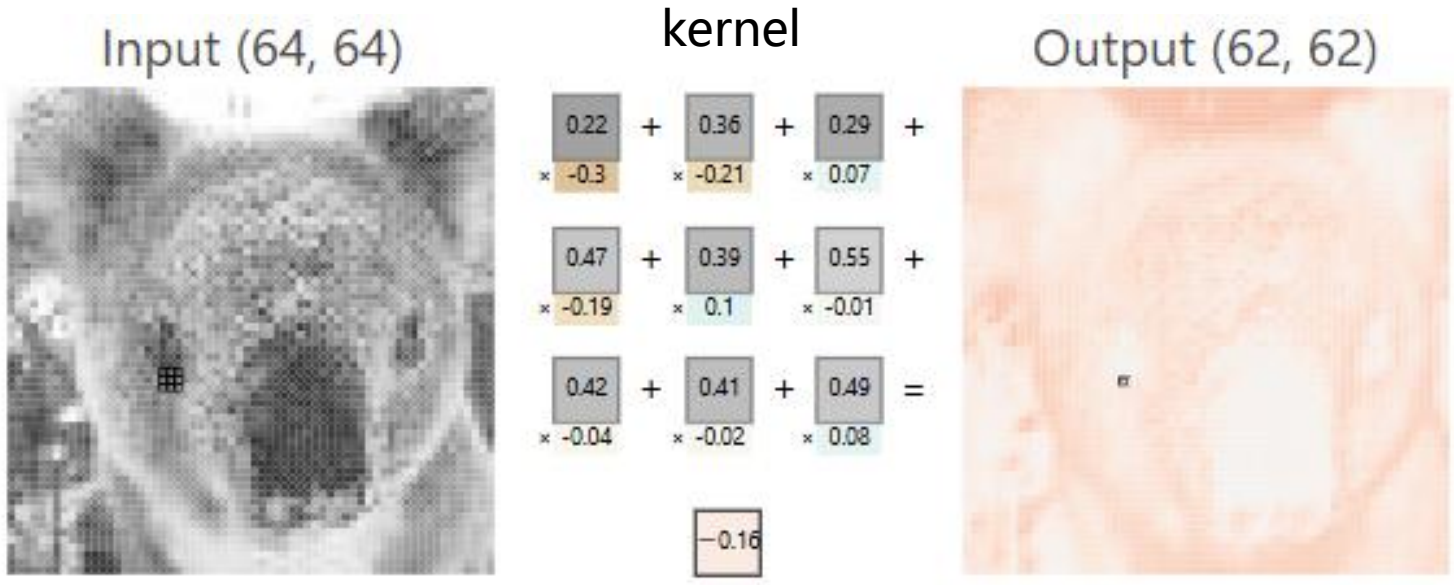
第三章 卷积神经网络

什么是卷积

- 在卷积神经网络（CNN）中，卷积（Convolution）是一个非常核心的操作，它用来从输入数据（如图像）中提取特征。
- 简单来说，卷积操作就是通过滑动一个小的“卷积核”或“滤波器”在输入数据上进行计算，生成一个新的特征图

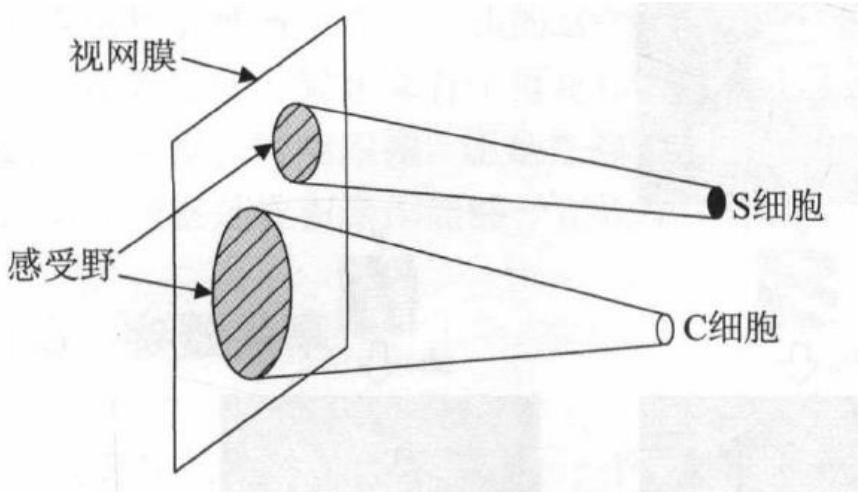


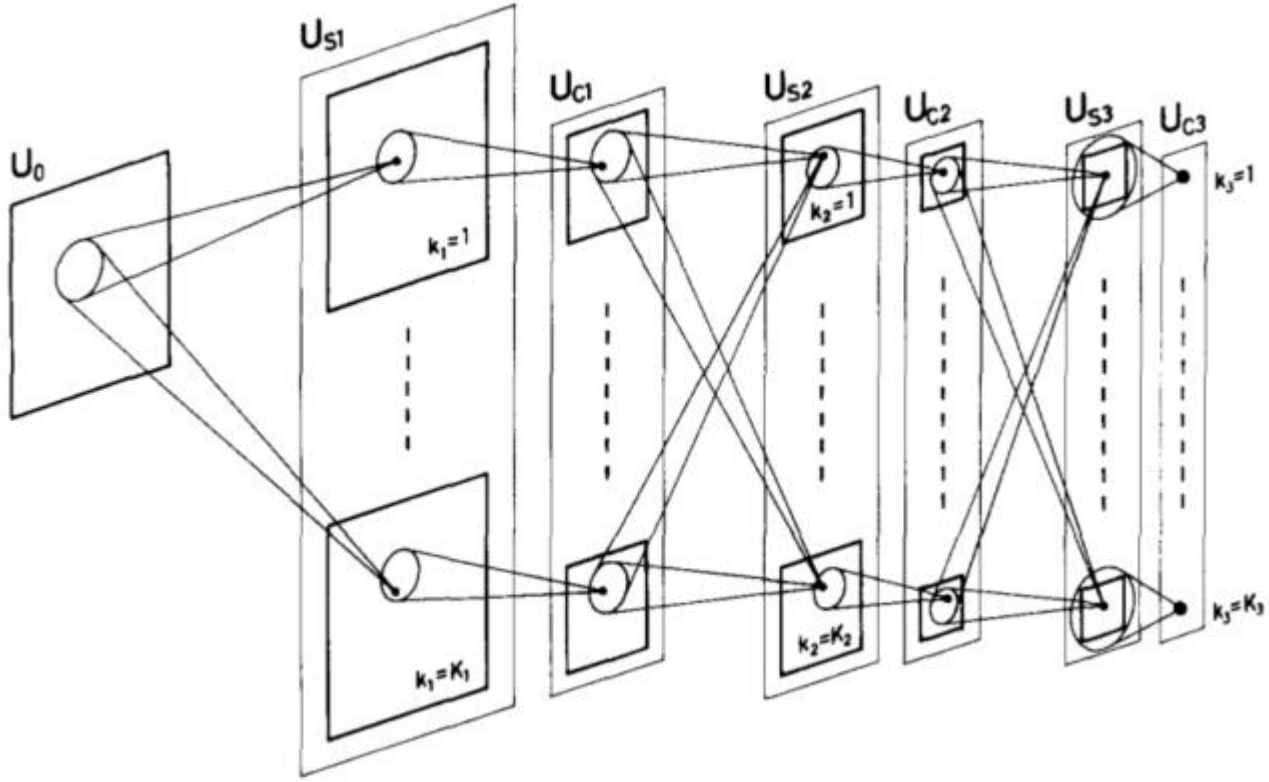


https://github.com/vdumoulin/conv_arithmetic/
https://poloclub.github.io/

CNN的小历史

- 1980年，Kunihiko Fukushima 提出 Neocognitron，包含了“局部感受野”和“层级+下采样”的核心思想（后来对应卷积/池化的雏形）





Fukushima, *Biological Cybernetics*, 1980

CNN的小历史

- 1989 年：Yann LeCun 等人让“卷积”真正落地，奠定了现代 CNN 的工程基石
- 把“局部连接 + 权重共享 + 下采样”的卷积式结构，与反向传播系统化结合，用于手写数字/文档识别（LeNet 系列）

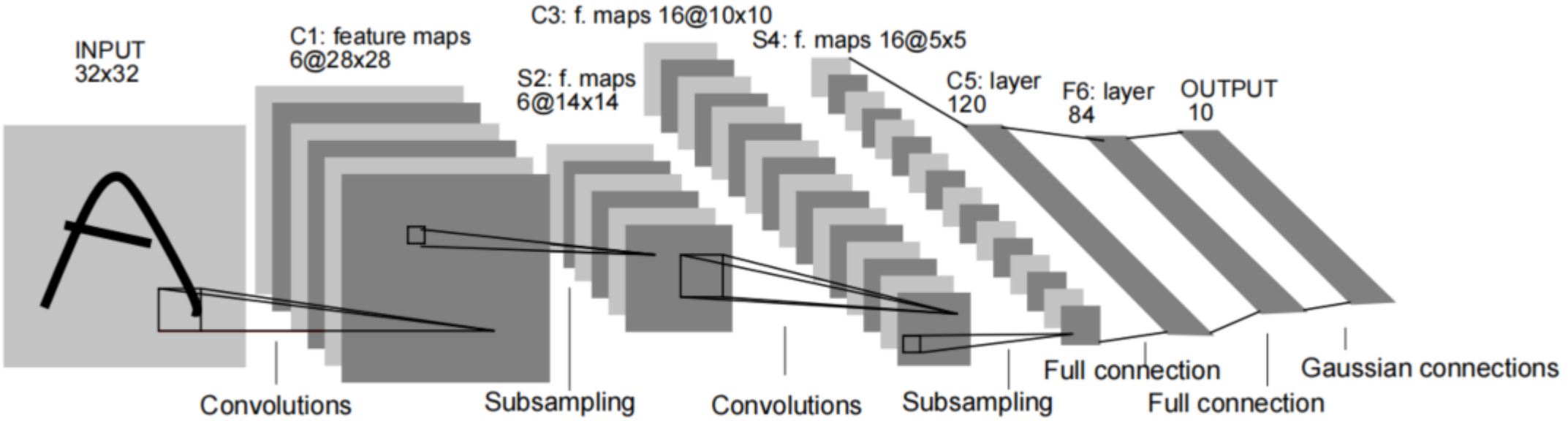
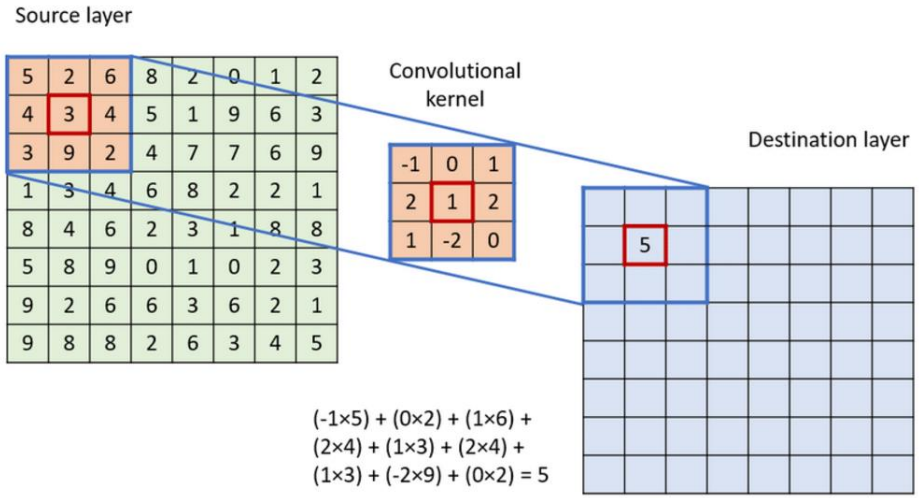


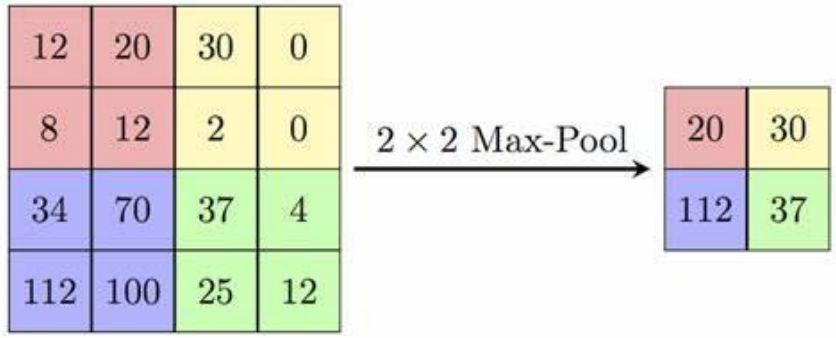
Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

CNN的成分

- 卷积层

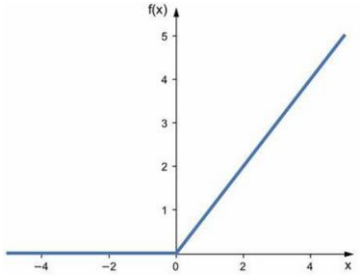


- 最大池化（Max pooling）

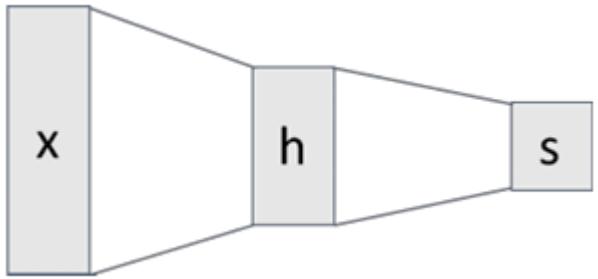


- 神经元激活函数

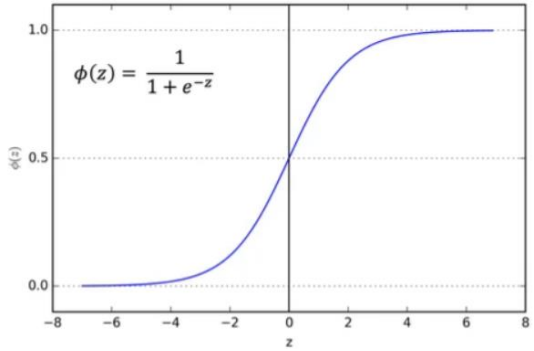
$$\mathrm{ReLU}(x) = \begin{cases} x, x \geq 0 \\ 0, x < 0 \end{cases}$$



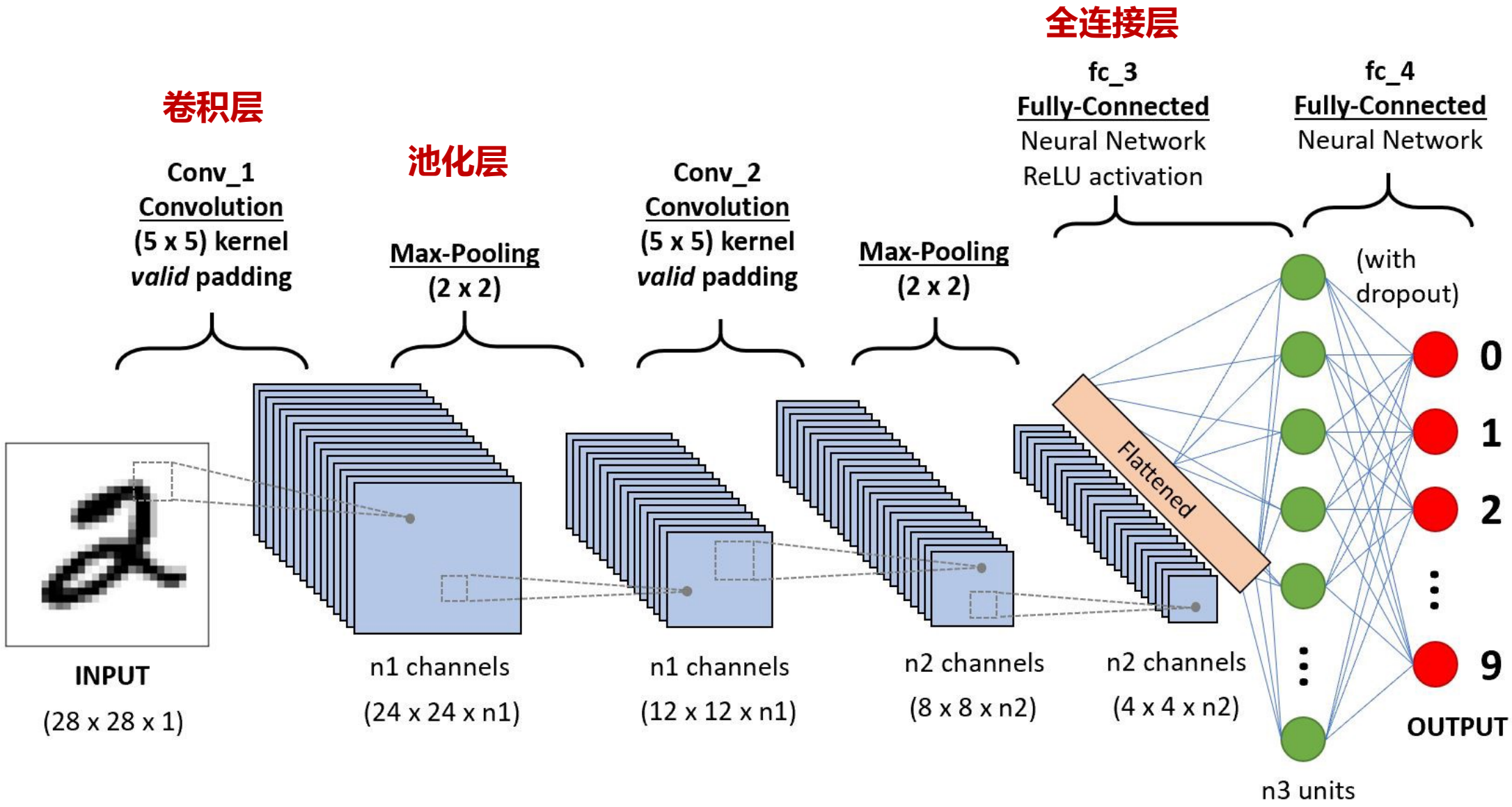
- 全连接层



- 输出：sigmoid函数



CNN的架构


全连接层
卷积层
池化层
Conv_1
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
Conv_2
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
fc_3
Fully-Connected
Neural Network
ReLU activation
fc_4
Fully-Connected
Neural Network
(with
dropout)
Flattened
0
1
2
9
INPUT
(28 x 28 x 1)
n1 channels
(24 x 24 x n1)
n1 channels
(12 x 12 x n1)
n2 channels
(8 x 8 x n2)
n2 channels
(4 x 4 x n2)
n3 units
OUTPUT

为什么要用CNN

- MLP（全连接网络）的不足

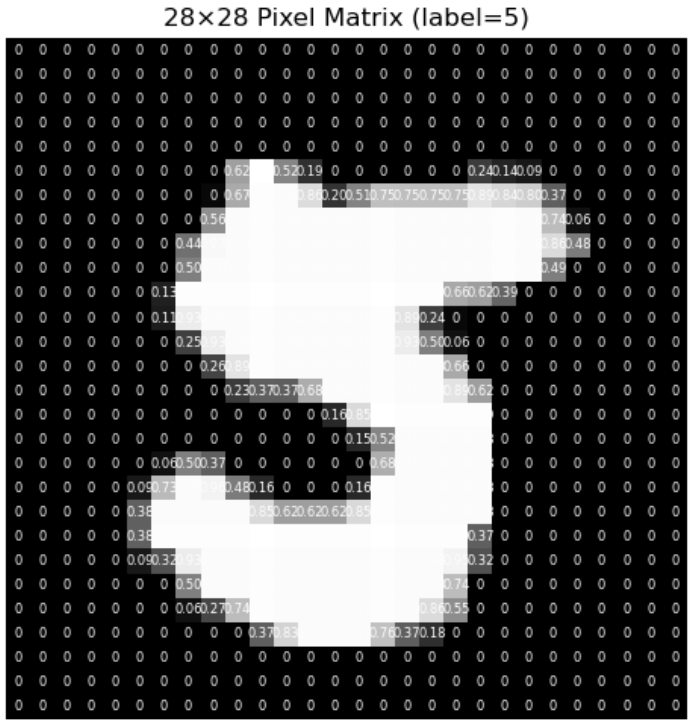
每个像素位置都有独立权重，易过拟合

参数过多

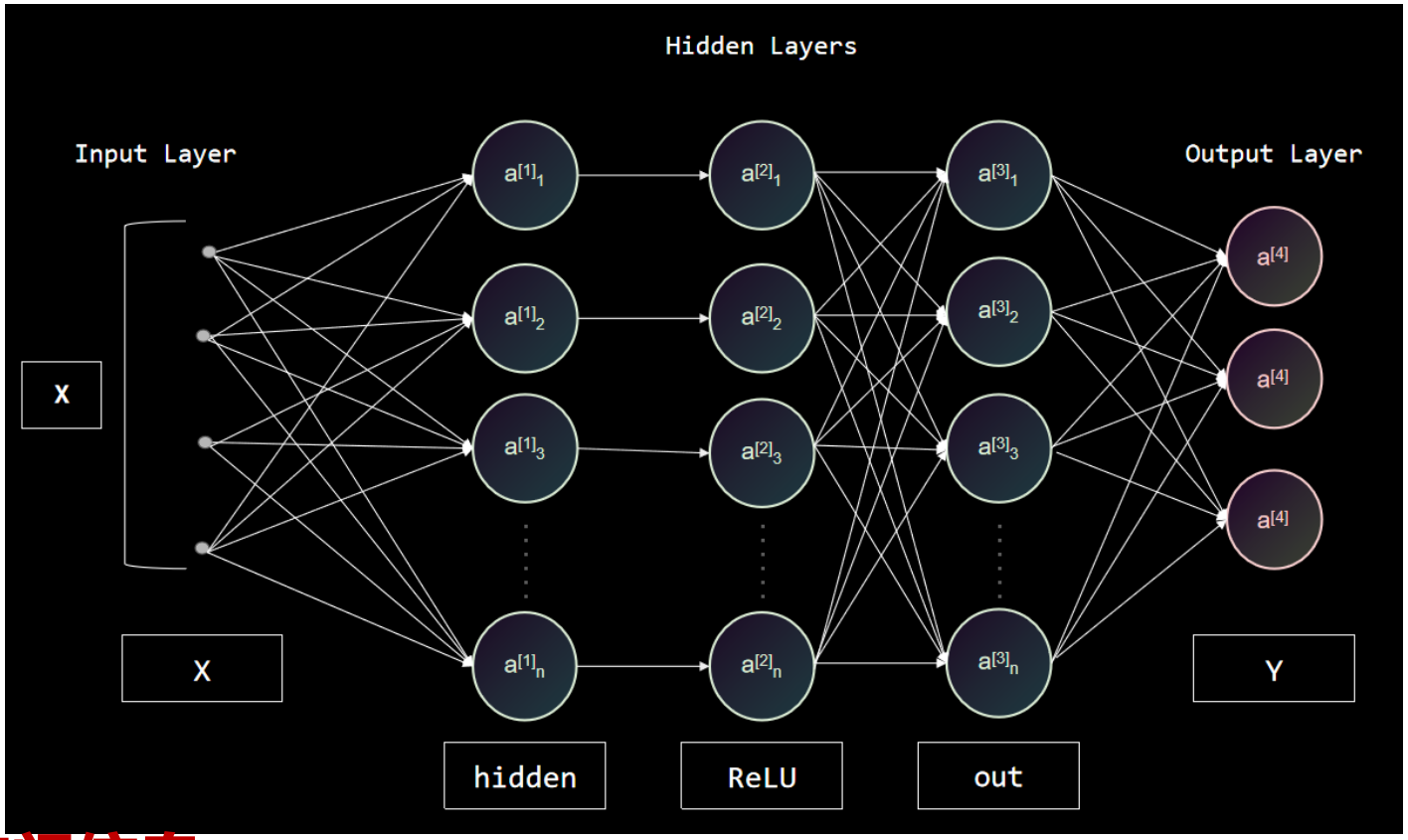
n=128

784

$(728 \times 128) + 128 = 100480$



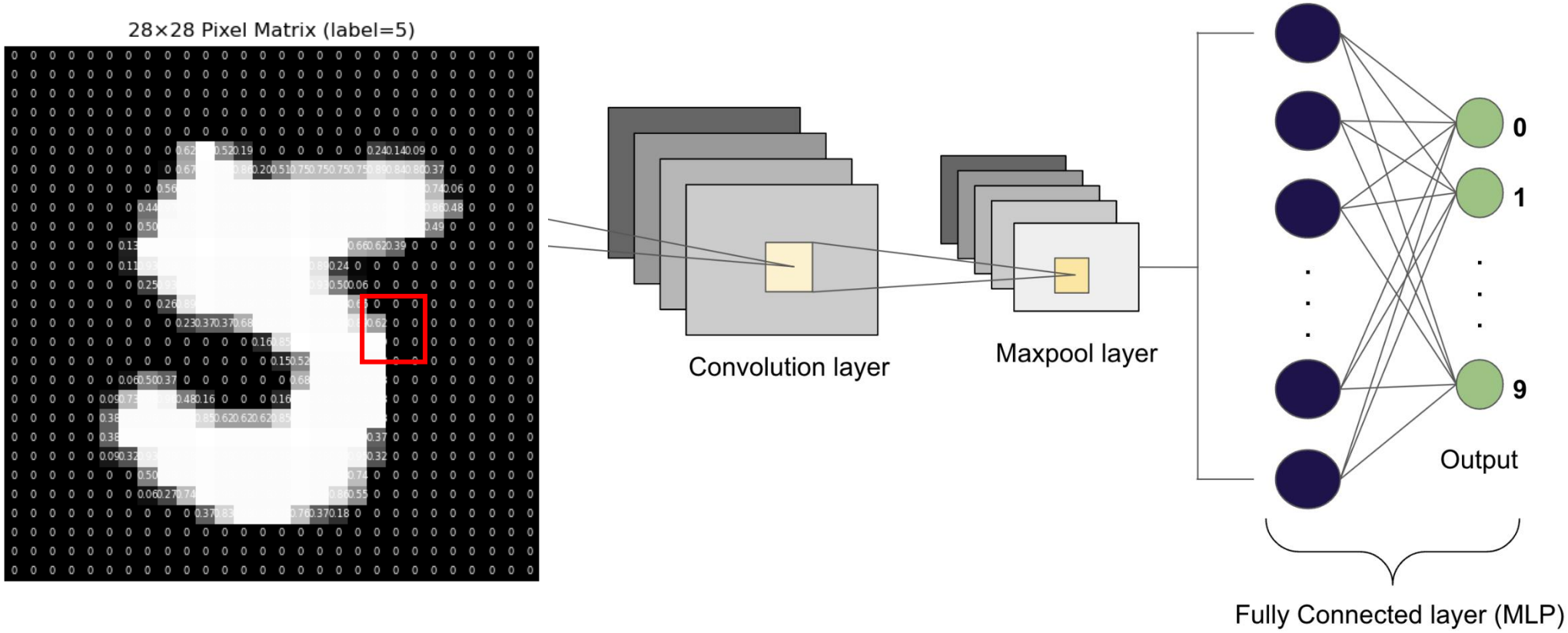
展开



展开后丢失空间信息

为什么要用CNN

- 使用卷积在输入数据上进行滑动，从而学习局部的**空间特征**
- 共享的卷积核，**大大减少了参数数量**，提高了网络的训练效率
- 卷积网络能够捕捉到**平移不变性**特征



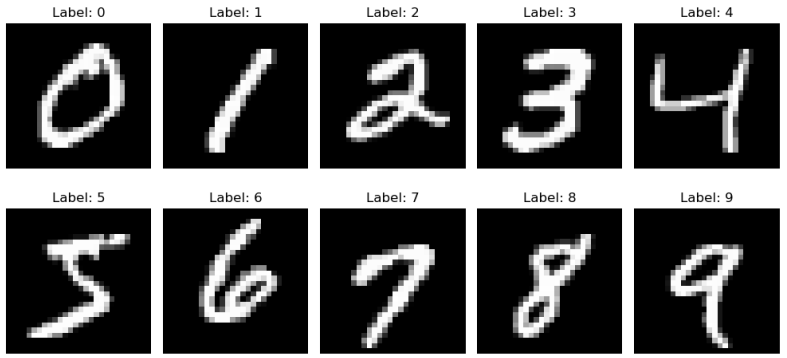
使用CNN进行图像识别

- **必要的库**

```
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
```

- **加载MNIST数据集**

```
# 加载 MNIST 数据集
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```



MNIST 是 28×28 的灰度图片，标签为 0–9

数据预处理

- 将像素归一化到 [0,1]，并在最后增加一个通道维度

```
x_train = (x_train.astype("float32") / 255.0)[..., None]
x_test = (x_test.astype("float32") / 255.0)[..., None]
print(f"x_train shape: {x_train.shape}")
print(f"x_test shape: {x_test.shape}")
```

- 因为在 Keras / TensorFlow 里，卷积层 Conv2D 要求输入张量的形状是 (batch_size, height, width, channels)

RGB 彩色图 = 3
灰度图（MNIST） = 1

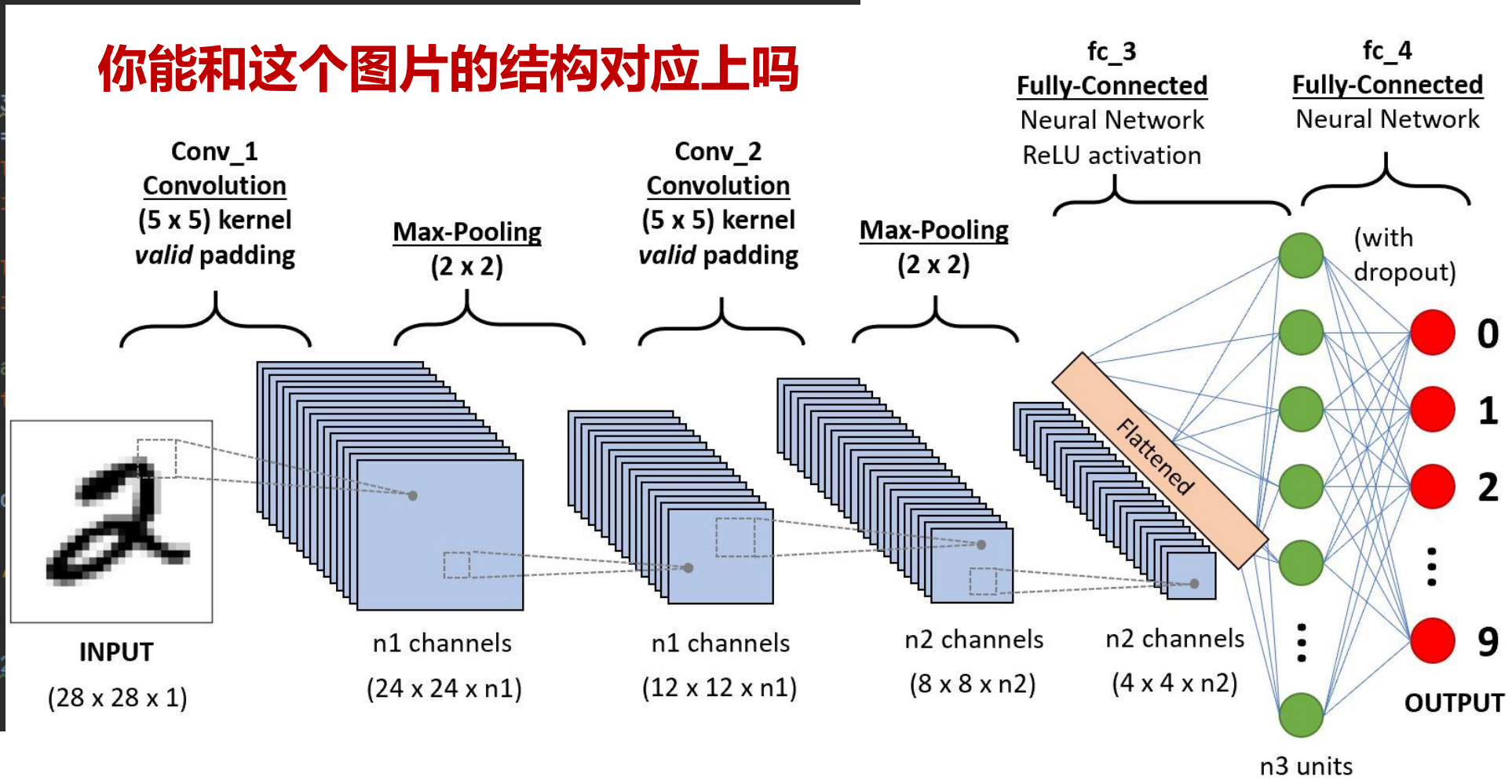
定义CNN模型

```python
num_classes = 10
input_shape = (28, 28, 1)

def build_model(n1=16, n2=32, n3=128, dropout=0.5):
    inputs = layers.Input(shape=input_shape, name="input")
    x = layers.Conv2D(n1, kernel_size=5, padding="valid", activation="relu", name="conv1")(inputs)
    x = layers.MaxPool2D(pool_size=2, name="pool1")(x)

    x = layers.Conv2D(n2, kernel_size=5, padding="valid", activation="relu", name="conv2")(x)
    x = layers.MaxPool2D(pool_size=2, name="pool2")(x)

    x = layers.Flatten(name="flatten")(x)
    x = layers.Dense(n3, activation="relu", name="fc3")(x)
    x = layers.Dropout(dropout, name="dropout")(x)

    outputs = layers.Dense(num_classes, activation="softmax", name="fc4")(x)

    model = models.Model(inputs, outputs, name="CNN")
    return model

model = build_model(n1=16, n2=32, n3=128, dropout=0.5)
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input (InputLayer)           [(None, 28, 28, 1)]       0

conv1 (Conv2D)               (None, 24, 24, 16)        416

pool1 (MaxPooling2D)         (None, 12, 12, 16)        0

conv2 (Conv2D)               (None, 8, 8, 32)          12832

pool2 (MaxPooling2D)         (None, 4, 4, 32)          0

flatten (Flatten)            (None, 512)               0

fc3 (Dense)                  (None, 128)               65664

dropout (Dropout)            (None, 128)               0

fc4 (Dense)                  (None, 10)                1290

=================================================================
```

定义CNN模型

```
num_classes = 10
input_shape = (28, 28, 1)

def build_model(n1=16, n2=32, n3
    inputs = layers.Input(shape=
    x = layers.Conv2D(n1, kernel
    x = layers.MaxPool2D(pool_si

    x = layers.Conv2D(n2, kernel
    x = layers.MaxPool2D(pool_si

    x = layers.Flatten(name="fla
    x = layers.Dense(n3, activat
    x = layers.Dropout(dropout,

    outputs = layers.Dense(num_c

    model = models.Model(inputs,
    return model

model = build_model(n1=16, n2=32
model.summary()
```



模型训练

```python
model.compile(
    optimizer=tf.keras.optimizers.Adam(1e-3),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=["accuracy"]
)

history = model.fit(
    x_train, y_train,
    batch_size=128,
    epochs=12,
    validation_split=0.1,
    verbose=1
)
```

```
Epoch 1/12
422/422 [==============================] - 11s 21ms/step - loss: 0.3827 - accuracy: 0.8813 - val_loss: 0.0823 - val_accuracy: 0.9742
Epoch 2/12
422/422 [==============================] - 9s 21ms/step - loss: 0.1195 - accuracy: 0.9641 - val_loss: 0.0529 - val_accuracy: 0.9848
Epoch 3/12
422/422 [==============================] - 8s 20ms/step - loss: 0.0877 - accuracy: 0.9741 - val_loss: 0.0449 - val_accuracy: 0.9862
Epoch 4/12
422/422 [==============================] - 8s 20ms/step - loss: 0.0674 - accuracy: 0.9797 - val_loss: 0.0384 - val_accuracy: 0.9887
Epoch 5/12
422/422 [==============================] - 9s 22ms/step - loss: 0.0575 - accuracy: 0.9823 - val_loss: 0.0361 - val_accuracy: 0.9895
Epoch 6/12
422/422 [==============================] - 9s 22ms/step - loss: 0.0496 - accuracy: 0.9851 - val_loss: 0.0383 - val_accuracy: 0.9888
Epoch 7/12
422/422 [==============================] - 9s 20ms/step - loss: 0.0449 - accuracy: 0.9868 - val_loss: 0.0369 - val_accuracy: 0.9887
Epoch 8/12
422/422 [==============================] - 9s 20ms/step - loss: 0.0401 - accuracy: 0.9876 - val_loss: 0.0329 - val_accuracy: 0.9900
Epoch 9/12
422/422 [==============================] - 10s 24ms/step - loss: 0.0357 - accuracy: 0.9893 - val_loss: 0.0286 - val_accuracy: 0.9923
Epoch 10/12
422/422 [==============================] - 12s 29ms/step - loss: 0.0327 - accuracy: 0.9903 - val_loss: 0.0293 - val_accuracy: 0.9915
Epoch 11/12
422/422 [==============================] - 9s 21ms/step - loss: 0.0287 - accuracy: 0.9909 - val_loss: 0.0304 - val_accuracy: 0.9917
Epoch 12/12
422/422 [==============================] - 11s 25ms/step - loss: 0.0275 - accuracy: 0.9917 - val_loss: 0.0292 - val_accuracy: 0.9920
```

每个参数是什么意思？ 能否更改得到更合适的参数？

模型参数

```
model.compile(
    optimizer=tf.keras.optimizers.Adam(1e-3),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=["accuracy"]
)

history = model.fit(
    x_train, y_train,
    batch_size=128,
    epochs=12,
    validation_split=0.1,
    verbose=1
)
```

优化器：决定参数更新的规则

Adam 是一种常用优化算法，结合了动量和自适应学习

1e-3 ，是学习率（ learning rate ），决定每次更新的步长大小。值太大训练会不稳定，太小收敛会很慢

损失函数：衡量预测和真实标签的差距。

评估指标：常用的还有 precision、 recall 等

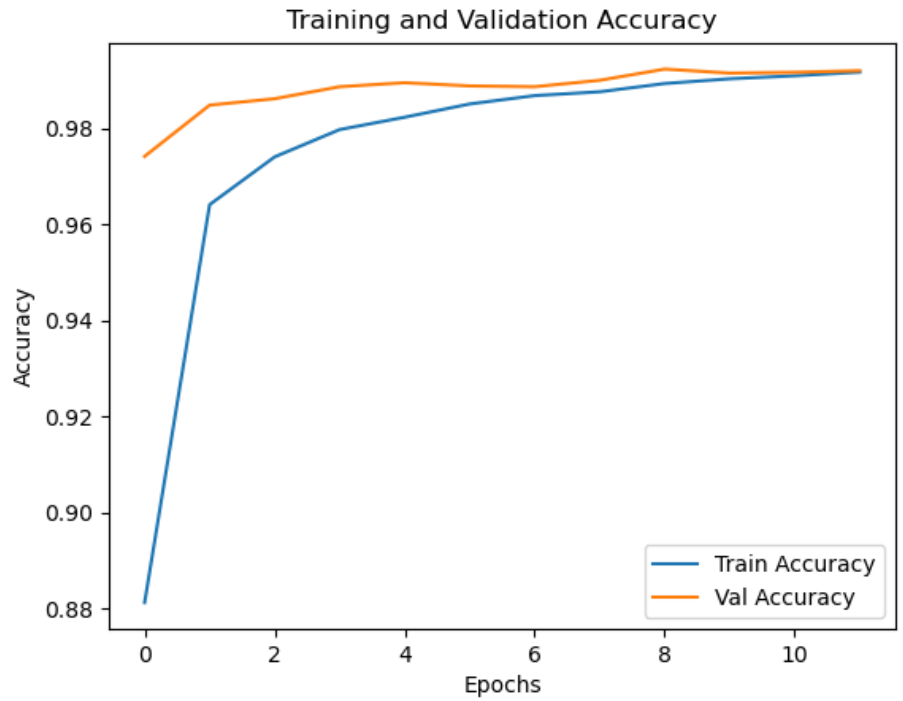
每次参数更新时，用 128 张图片一起计算梯度

一个 epoch = 把整个训练集都过一遍

启动训练

训练过程

```
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



测试集评估

```
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
print(f"Test Acc = {test_acc:.4f} | Test Loss = {test_loss:.4f}")
```

```
Test Acc = 0.9923 | Test Loss = 0.0265
```

- **模型保存**

```
model.save("mnist_cnn_model.h5")
print("Model saved to 'mnist_cnn_model.h5'")
```

加载模型并使用

```python
loaded_model = tf.keras.models.load_model("mnist_cnn_model.h5")
print("Model loaded successfully")


def predict_new_data(model, new_data):
    new_data = (new_data.astype("float32") / 255.0)[..., None]
    new_data = np.expand_dims(new_data, axis=0)

    predictions = model.predict(new_data)
    predicted_class = np.argmax(predictions, axis=-1)
    return predicted_class[0]

new_image = x_test[0]
predicted_label = predict_new_data(loaded_model, new_image)
print(f"Predicted label: {predicted_label}")
```

使用CNN进行蛋白质序列预测

- **AKT激酶的蛋白质磷酸化位点预测**


```
df = pd.read_csv("akt_data.tsv", sep="\t")
seqs = df["sequence"].astype(str).tolist()
labels = df["label"].astype(int).to_numpy()
```

One-hot 编码序列

```python
AA = "ACDEFGHIKLMNPQRSTVWY"
PAD = "X"
AA_LIST = list(AA + PAD)
AA2IDX = {a:i for i,a in enumerate(AA_LIST)}

def one_hot_encode(seqs):
    n = len(seqs); L = len(seqs[0]); V = len(AA_LIST)
    arr = np.zeros((n, L, V), dtype=np.float32)
    for i, seq in enumerate(seqs):
        for j, ch in enumerate(seq):
            arr[i, j, AA2IDX.get(ch, AA2IDX[PAD])] = 1.0
    return arr

X = one_hot_encode(seqs)
y = labels
print("X shape:", X.shape)
```

划分数据集并构建模型

```
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.30, stratify=y, random_state=2025)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.50, stratify=y_temp, random_state=2025)

X_train.shape, X_val.shape, X_test.shape
```

```
def build_cnn(input_len=31, vocab_size=21, filters=64, dense_units=64, dropout=0.5):
    inp = layers.Input(shape=(input_len, vocab_size))

    x = layers.Conv1D(filters=filters, kernel_size=3, padding="same", activation="relu")(inp)
    x = layers.BatchNormalization()(x)
    x = layers.MaxPooling1D(pool_size=2)(x)

    x = layers.Conv1D(filters=filters*2, kernel_size=5, padding="same", activation="relu")(x)
    x = layers.BatchNormalization()(x)
    x = layers.GlobalMaxPooling1D()(x)

    x = layers.Dropout(dropout)(x)
    x = layers.Dense(dense_units, activation="relu")(x)
    x = layers.Dropout(dropout)(x)
    out = layers.Dense(1, activation="sigmoid")(x)

    model = models.Model(inp, out, name="AKT_CNN")
    return model

model = build_cnn(input_len=31, vocab_size=21)
model.compile(optimizer=tf.keras.optimizers.Adam(1e-3),
              loss="binary_crossentropy",
              metrics=["accuracy", tf.keras.metrics.AUC(name="AUC")])
model.summary()
```

模型训练

```python
os.makedirs("checkpoints", exist_ok=True)
ckpt_path = "checkpoints/akt_cnn_best.keras"

callbacks = [
    ReduceLROnPlateau(monitor="val_auc", factor=0.5, patience=2, mode="max", verbose=1),
    EarlyStopping(monitor="val_auc", patience=5, mode="max", restore_best_weights=True, verbose=1),
    ModelCheckpoint(
        filepath=ckpt_path,
        monitor="val_auc",
        mode="max",
        save_best_only=True,
        save_weights_only=False,
        verbose=1
    )
]

history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=20,
    batch_size=64,
    callbacks=callbacks,
    verbose=1
)
print("Best model saved to:", ckpt_path)
```

训练中自动保存“验证集 AUC 最高”的模型

```python
test_loss, test_acc, test_auc = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Accuracy = {test_acc:.4f}, AUC = {test_auc:.4f}")
```

模型保存

- 保存模型

```
model.save(final_sm)
print("Saved SavedModel dir:", final_sm)
```

- 仅保存权重

```
model.save_weights(weights_path)
print("Saved weights:", weights_path)
```

课堂总结 & 课外学习

- 1. 了解什么是卷积，以及卷积神经网络的优势
- 2. 知道了如何定义、训练、保存卷积神经网络模型
- 3. 使用用卷积神经网络做图像处理和序列处理

- 参考资料：
  - https://www.youtube.com/watch?v=FrKWiRv254g