手写数字识别

1.4 LeNet-5

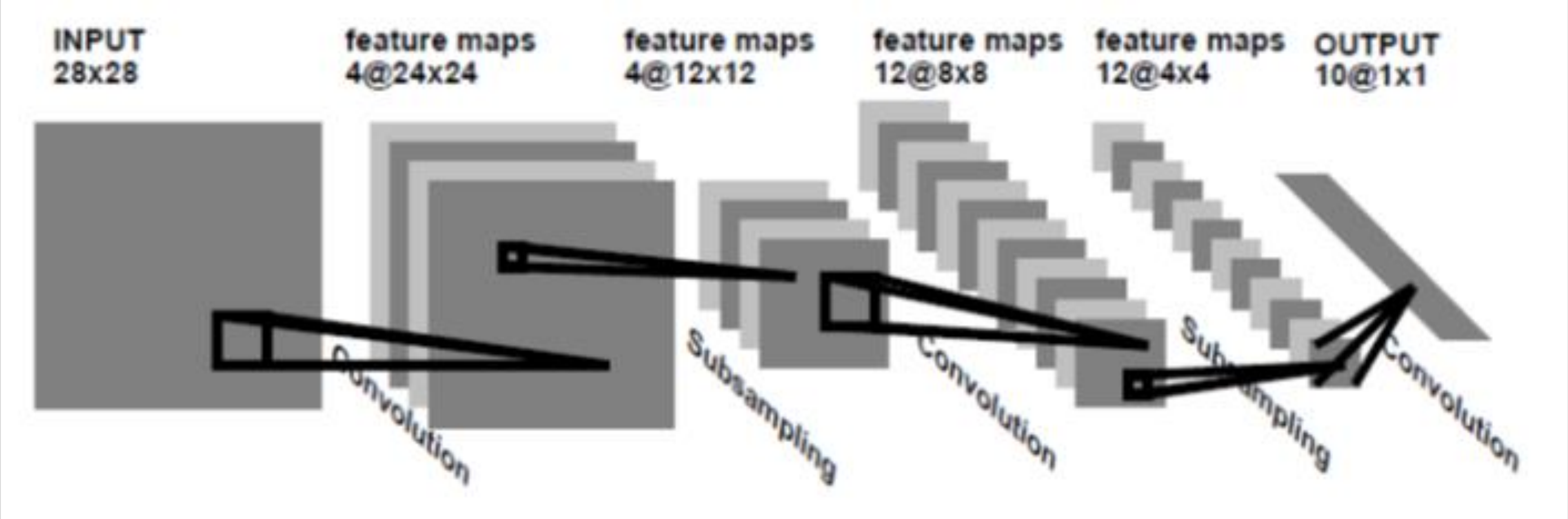
@tm9161

L5

LeNet-5

1. LeNet-5网络结构
2. LeNet-5搭建与训练
3. LeNet-5保存、调用和预测

LeNet-5 发展历史



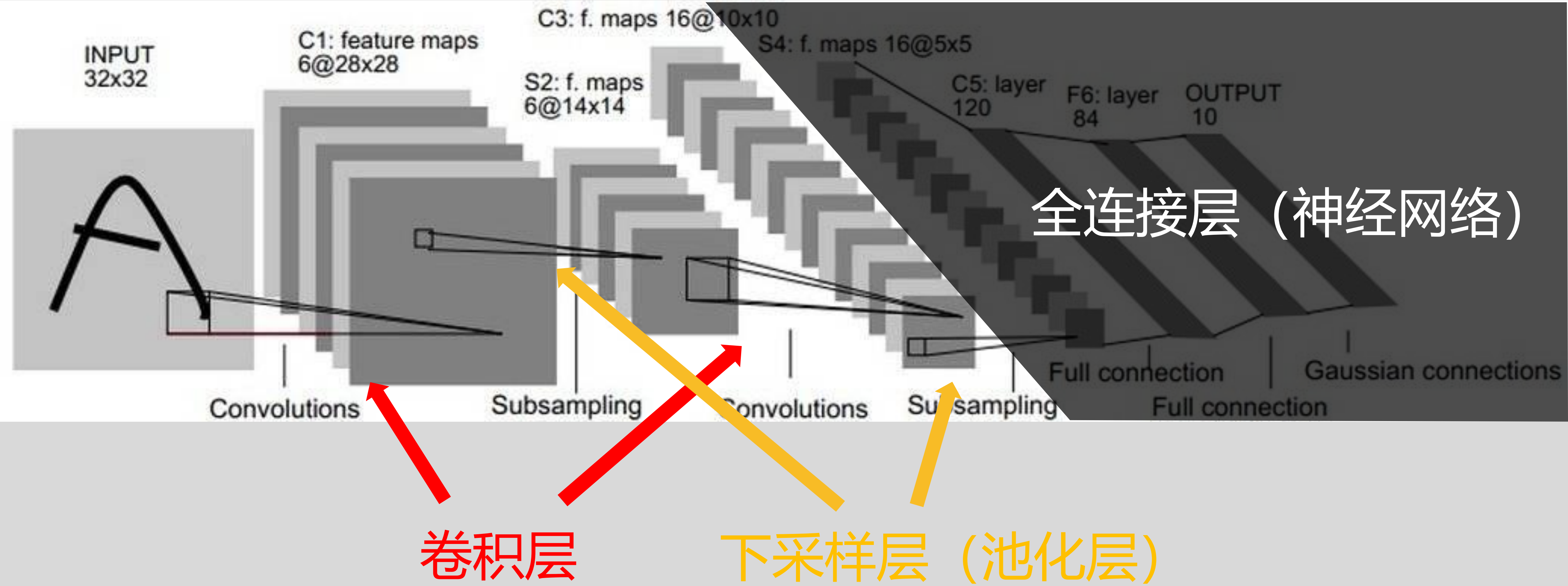
LeNet-1

1989年：Yann LeCun等人，结合反向传播算法的卷积神经网络来识别手写数字，并成功地用于识别手写邮政编码。

1990年：他们的模型在美国邮政总局提供的邮政编码数字数据的测试结果表明，错误率仅为1%，拒绝率约为9%。

1998年：他们将手写数字识别的各种方法在标准的手写数字识别基准上进行比较，结果表明他们的网络优于所有其他模型，经过多年的研究和迭代，最终发展成为LeNet-5。

网络结构



INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections
全连接层（神经网络）
卷积层
下采样层（池化层）

Convolution: Trying every possible match

$$
\begin{bmatrix}
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\
-1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\
-1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\
-1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\
-1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1
\end{bmatrix}
\otimes
\begin{bmatrix}
1 & -1 & -1 \\
-1 & 1 & -1 \\
-1 & -1 & 1
\end{bmatrix}
=
\begin{bmatrix}
0.77 & -0.11 & 0.11 & 0.33 & 0.55 & -0.11 & 0.33 \\
-0.11 & 1.00 & -0.11 & 0.33 & -0.11 & 0.11 & -0.11 \\
0.11 & -0.11 & 1.00 & -0.33 & 0.11 & -0.11 & 0.55 \\
0.33 & 0.33 & -0.33 & 0.55 & -0.33 & 0.33 & 0.33 \\
0.55 & -0.11 & 0.11 & -0.33 & 1.00 & -0.11 & 0.11 \\
-0.11 & 0.11 & -0.11 & 0.33 & -0.11 & 1.00 & -0.11 \\
0.33 & -0.11 & 0.55 & 0.33 & 0.11 & -0.11 & 0.77
\end{bmatrix}
$$

Pooling

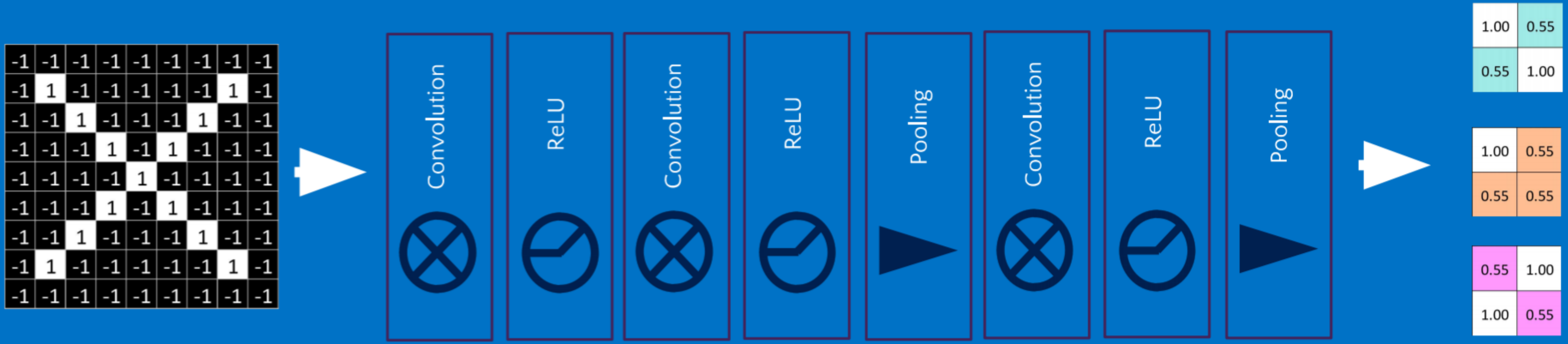
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

max pooling

1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

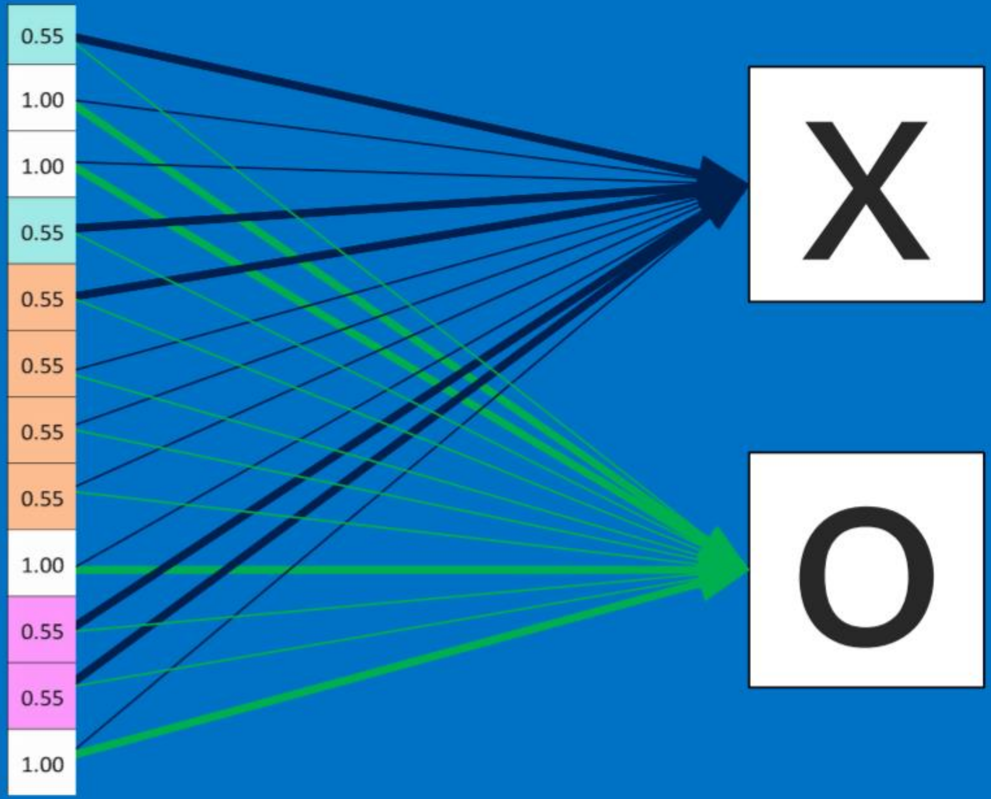
Deep stacking

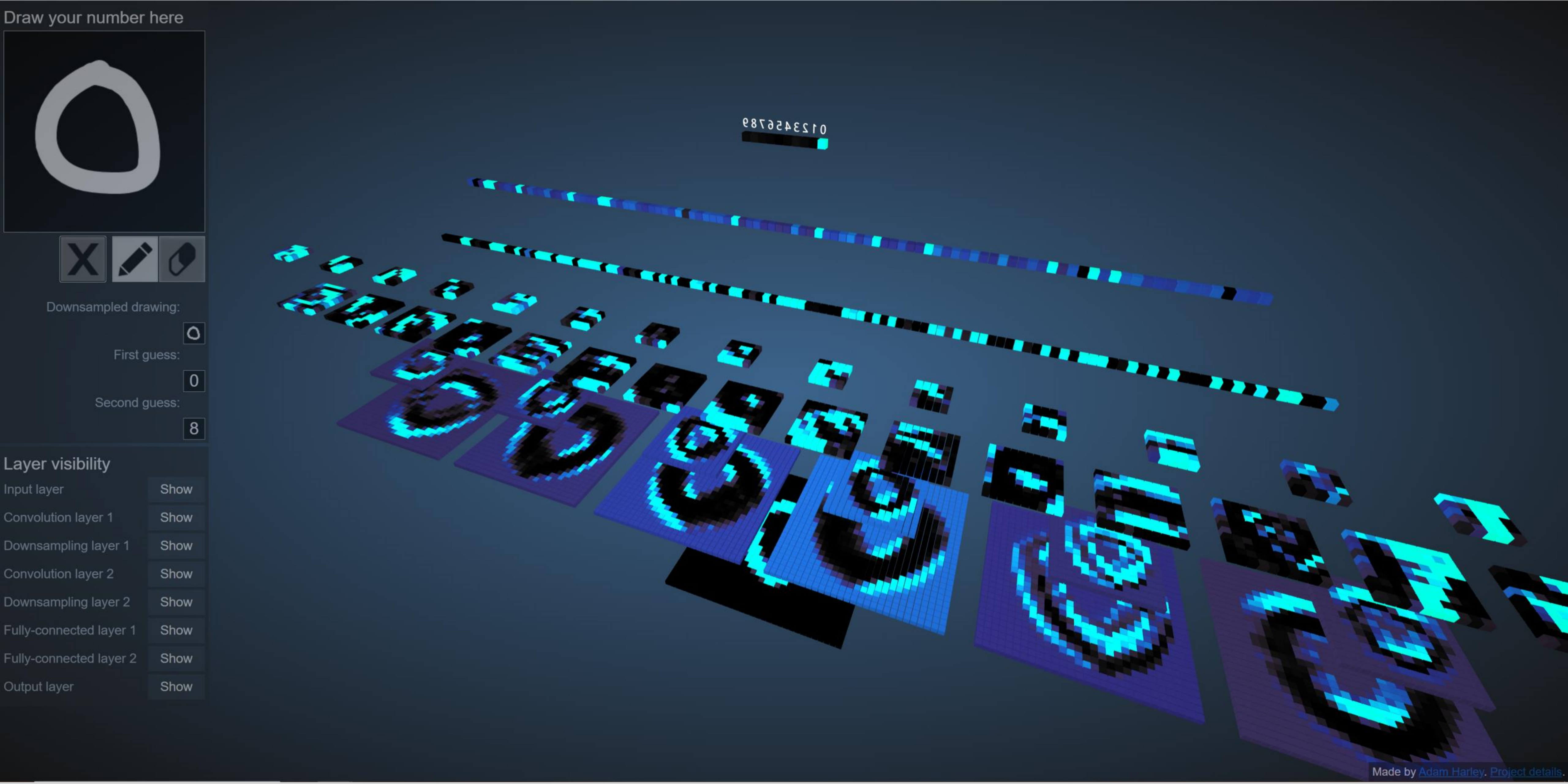
Layers can be repeated several (or many) times.



Fully connected layer

Vote depends on how strongly a value predicts X or O





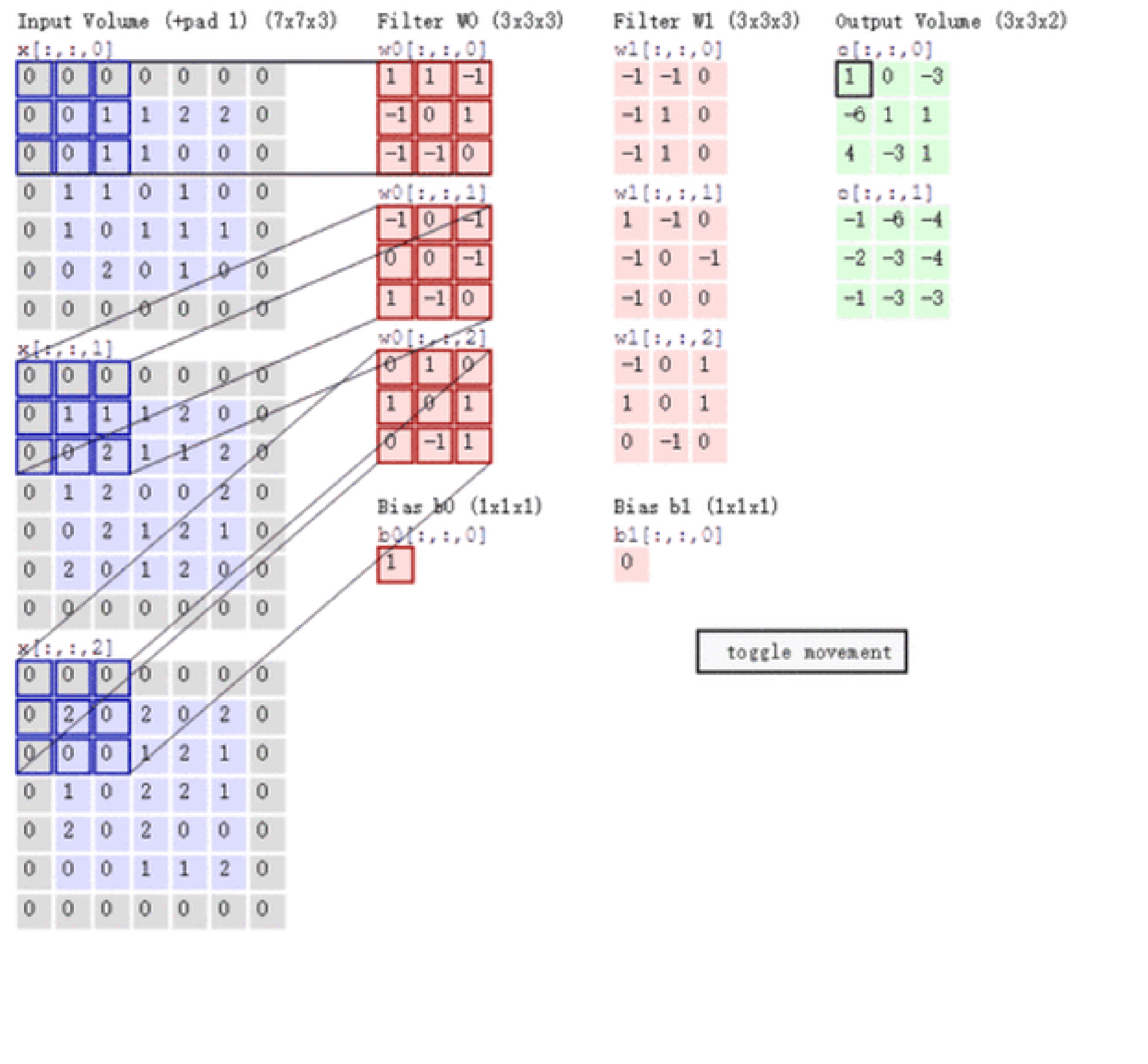
https://www.cs.ryerson.ca/~aharley/vis/conv/

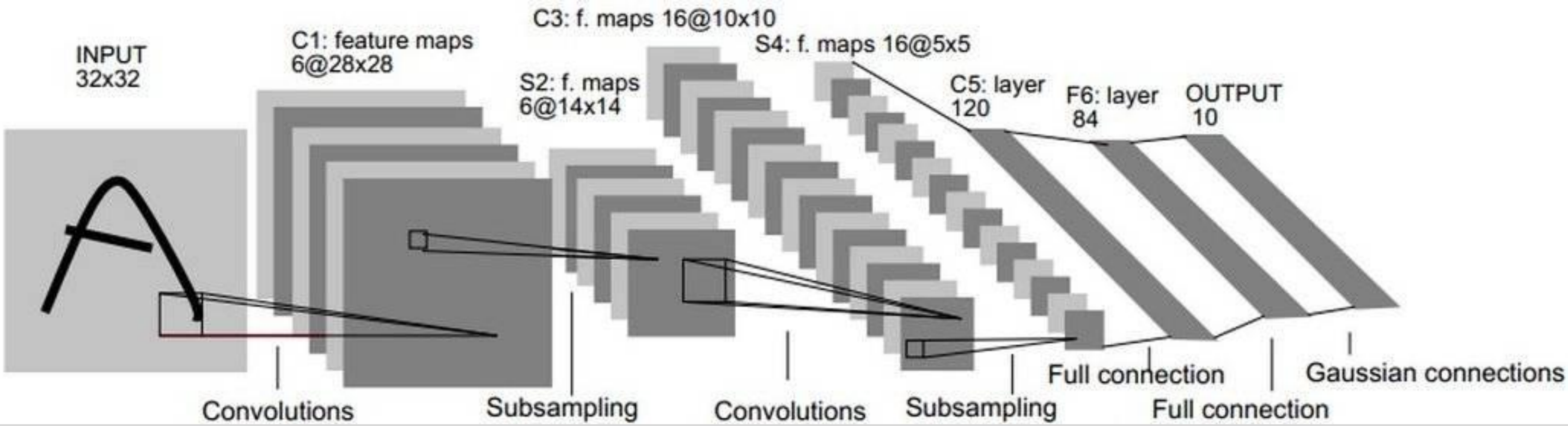
训练参数

卷积层：卷积核中的参数+偏置项；（3层*3x3大小 +1*偏置）*2个

池化层：无参数需要训练。

*卷积核维度&卷积核个数区分





	输入	卷积、池化、神经元	输出	训练参数
输入层*			32*32	0
卷积层1	32*32	6个 5*5卷积核 步长为1	6*28*28（32-5+0）/1+1	1*（5*5）*6 +6=156
池化层1	6*28*28	2*2 步长为2	6*14*14	0
卷积层2	6*14*14	16个 5*5卷积核 步长为1	16*10*10 （14-5+0）/1+1	6*（5*5）*16 +16=2416
池化层2	16*10*10	2*2 步长为2	16*5*5	0
全连接层1	16*5*5	120个 5*5卷积核 步长为1	120*1*1（5-5+0）/1+1	16*（5*5）*120+120=48120
全连接层2	120		84	120*84+84=10164
输出层	84		10	84*10+10=850

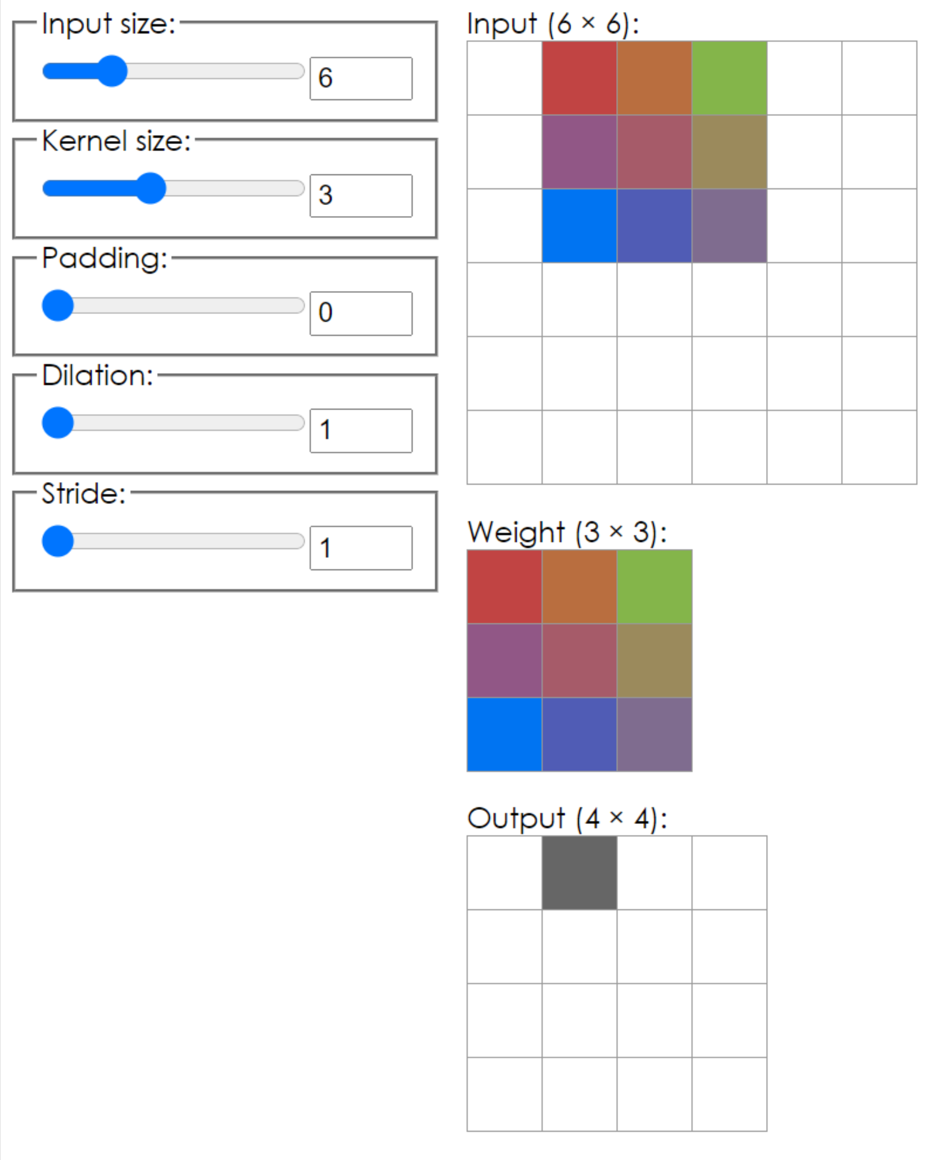
步长 Stride & 加边 Padding

卷积后尺寸=
（输入-卷积核+加边像素数）/步长 +1
(6-3+0)/1 +1 =4

Tensorflow 默认：
Padding= ‘valid ’（丢弃），strides=1

*长宽的改变

https://ezyang.github.io/convolution-visualizer/index.html

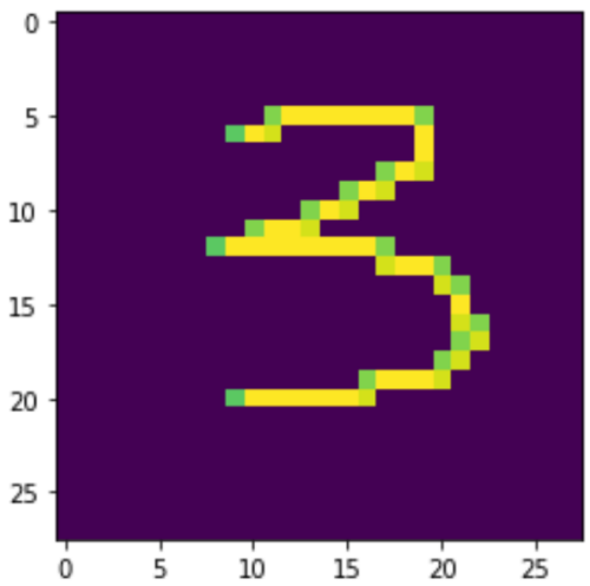


图片读取&预处理

```
In [3]: img = cv2.imread('3.png',0)
        #读取图片
```

```
In [4]: plt.imshow(img)
```

```
Out[4]: <matplotlib.image.AxesImage at 0x1ca56b9ae80>
```




```
In [5]: img = cv2.resize(img,(28,28))
        img = img.reshape(1,28,28,1)
        img = img/255
```

1.图片读取：cv2.imread

2.图片大小调整：cv2.resize

3.图片维度调整：reshape

```
train_images.shape
```

```
(60000, 28, 28)
```

```
train_images = train_images.reshape(60000, 28, 28, 1)
test_images = test_images.reshape(10000, 28, 28, 1)
```

```
train_images.shape
```

```
(60000, 28, 28, 1)
```

4.归一化：/255

维度改变 reshape

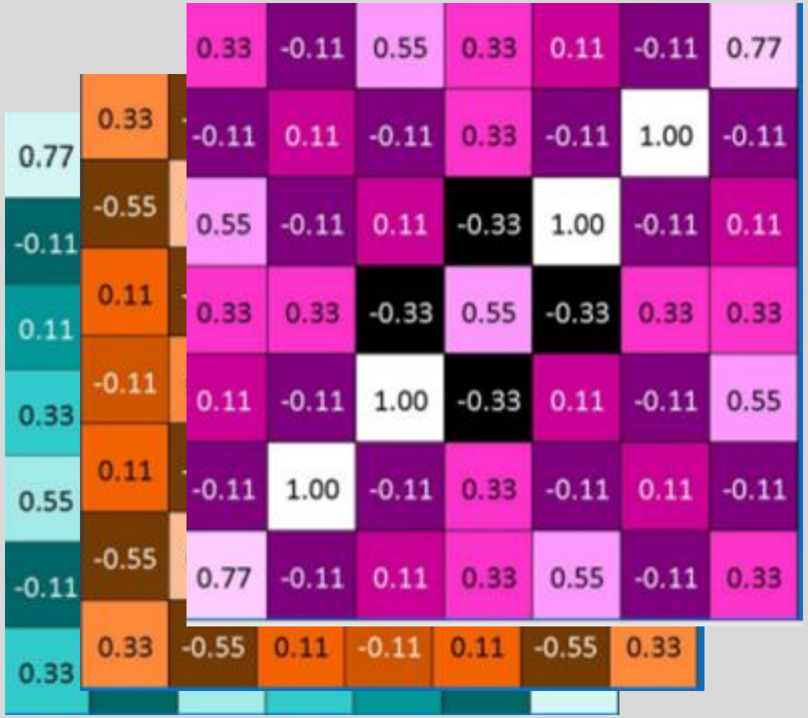
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

9*9*1

1	-1	-1
-1	1	-1
-1	-1	1

1	-1	1
-1	1	-1
1	-1	1

-1	-1	1
-1	1	-1
1	-1	-1



7*7*3

模型载入&预测

```
In [ ]: # 保存模型
        model.save('mnist.h5')
```

```
In [2]: new_model = tf.keras.models.load_model('mnist.h5')
        #调用模型
```

```
In [6]: predict = new_model.predict(img)
```

1

数据采集

In [1]:
```python
import tensorflow as tf

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
```

In [12]:
```python
(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.mnist.load_data()
```

In [13]:
```python
train_images = train_images.reshape(60000, 28, 28, 1)
test_images = test_images.reshape(10000, 28, 28, 1)
#增加维度，用于卷积操作
```

In [14]:
```python
train_images = train_images / 255
test_images = test_images/ 255
```

In [15]:
```python
train_labels = np.array(pd.get_dummies(train_labels))
test_labels = np.array(pd.get_dummies(test_labels))
```

2

建立模型

```
In [16]: model = tf.keras.Sequential()
```

```
In [17]: model.add(tf.keras.layers.Conv2D(filters = 6,kernel_size = (5,5),input_shape=(28,28,1),padding = 'same' activation = "sigmoid"))
model.add(tf.keras.layers.AveragePooling2D(pool_size = (2, 2)))
model.add(tf.keras.layers.Conv2D(filters = 16,kernel_size = (5,5),activation = "sigmoid"))
model.add(tf.keras.layers.AveragePooling2D(pool_size = (2, 2)))
model.add(tf.keras.layers.Conv2D(filters = 120,kernel_size = (5,5),activation = "sigmoid"))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(84, activation='sigmoid'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))
```

```
In [18]: model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 28, 28, 6)         156
_________________________________________________________________
average_pooling2d_6 (Average (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 10, 10, 16)        2416
_________________________________________________________________
average_pooling2d_7 (Average (None, 5, 5, 16)          0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 1, 1, 120)         48120
_________________________________________________________________
flatten_2 (Flatten)          (None, 120)               0
_________________________________________________________________
dense_9 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_10 (Dense)             (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
```

3

模型训练

```
In  [19]: model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['acc'])

In  [20]: history = model.fit(train_images,train_labels,epochs = 10,validation_data=(test_images,test_labels))

Epoch 1/10
1875/1875 [==============================] - 37s 19ms/step - loss: 1.4908 - acc: 0.4763 - val_loss: 0.2464 - val_acc: 0.9286
Epoch 2/10
1875/1875 [==============================] - 31s 17ms/step - loss: 0.2135 - acc: 0.9357 - val_loss: 0.1389 - val_acc: 0.9558
Epoch 3/10
1875/1875 [==============================] - 31s 17ms/step - loss: 0.1353 - acc: 0.9581 - val_loss: 0.0920 - val_acc: 0.9710
Epoch 4/10
1875/1875 [==============================] - 30s 16ms/step - loss: 0.0919 - acc: 0.9715 - val_loss: 0.0752 - val_acc: 0.9771
Epoch 5/10
1875/1875 [==============================] - 28s 15ms/step - loss: 0.0774 - acc: 0.9759 - val_loss: 0.0612 - val_acc: 0.9813
Epoch 6/10
1875/1875 [==============================] - 28s 15ms/step - loss: 0.0611 - acc: 0.9820 - val_loss: 0.0628 - val_acc: 0.9799
Epoch 7/10
1875/1875 [==============================] - 28s 15ms/step - loss: 0.0533 - acc: 0.9840 - val_loss: 0.0476 - val_acc: 0.9865
Epoch 8/10
1875/1875 [==============================] - 29s 15ms/step - loss: 0.0469 - acc: 0.9856 - val_loss: 0.0488 - val_acc: 0.9840
Epoch 9/10
1875/1875 [==============================] - 31s 16ms/step - loss: 0.0461 - acc: 0.9847 - val_loss: 0.0459 - val_acc: 0.9842
Epoch 10/10
1875/1875 [==============================] - 28s 15ms/step - loss: 0.0366 - acc: 0.9879 - val_loss: 0.0457 - val_acc: 0.9840
```

4

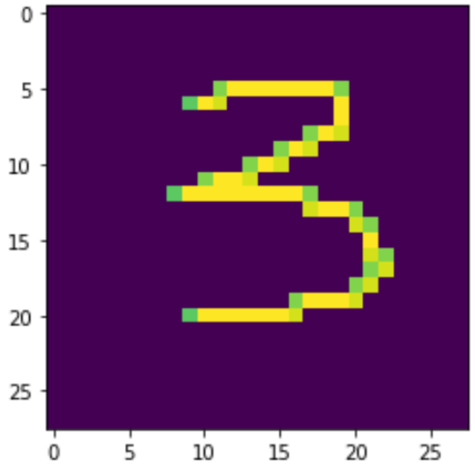
模型测试

```
In [2]: new_model = tf.keras.models.load_model('mnist.h5')
        #调用模型

In [3]: img = cv2.imread('3.png',0)
        #读取图片

In [4]: plt.imshow(img)

Out[4]: <matplotlib.image.AxesImage at 0x1ca56b9ae80>
```




```
In [5]: img = cv2.resize(img,(28,28))
        img = img.reshape(1,28,28,1)
        img = img/255

In [6]: predict = new_model.predict(img)

In [7]: predict

Out[7]: array([[2.9374178e-07, 4.0664068e-05, 4.3683460e-05, 9.9903995e-01,
                5.0486175e-08, 1.3586319e-04, 1.6269293e-09, 6.7247183e-04,
                3.3387743e-05, 3.3583085e-05]], dtype=float32)

In [8]: np.argmax(predict)

Out[8]: 3
```

参考资料：

1. Gradient-Based Learning Applied to Document Recognition
http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

2. How Deep Neural Networks Work
https://end-to-end-machine-learning.teachable.com/p/how-deep-neural-networks-work

3. 卷积神经网络3D可视化
https://www.cs.ryerson.ca/~aharley/vis/conv/

4.保存和恢复模型
https://tensorflow.google.cn/tutorials/keras/save_and_load?hl=zh-cn