LeNet-5

LeNet-5是YANN LECUN等人于1998年提出的一种网络结构，是卷积神经网络识别的开山之作，用来处理数字识别

论文地址:<https://ieeexplore.ieee.org/document/726791

整体架构

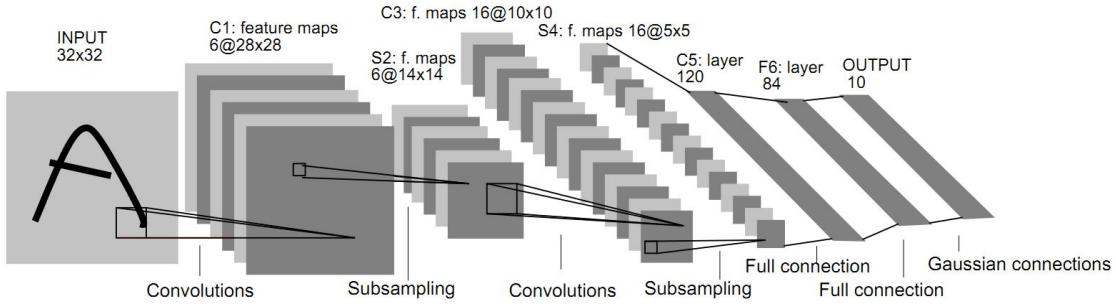

Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

从整体上来看，整个网络经过两次卷积与池化层后再经过卷积层，展开成全连接层，最后输出大小为10的向量，一共**七层**。

输入层

LeNet-5的输入图片是**32*32**的，然而，训练以及测试用的数据集都是**28*28**的，原因是因为图片潜在的**显著特征**比如**笔划的端点**一般都在图片的**边缘部分**，如果不经过填充直接进入卷积层，那么边缘的像素点**被卷积的次数将会很少**，那么提取到的特征也不会很显著，从而影响到整体的识别性能，所以给原图周围填充0的像素点，保证边缘特征能够充分提取。

C1层

C1层是一个卷积层，包含6个特征图(feature map)。卷积核的大小是5*5，步长为1，无填充。

可训练参数：(5*5*1+1)*6=156个。

卷积后的大小：32-5+1=28即28*28。

S2层

S2层是一个下采样层，即池化层，包含6个特征图，池化大小为2*2，即S2层的每个特征图中的每个单元都与C1层相对应的特征图的2*2邻域相连接，即每个单元都有4个输入，将4个输入相加后乘上一个可训练系数，再加上一个可训练偏差，然后通过**sigmoid函数**。2*2邻域是互不重叠的，即步长为2。

可训练参数：2*6=12个

池化后的大小：（28-2）/2+1=14即14*14。

C3层

C3层是一个卷积层，包含16个特征图。卷积核大小为5*5，步长为1，无填充。

值得注意的是，这里的卷积方式和现在普遍的卷积方式不同，一般的卷积方式是16个卷积核对S2层的输出的每个特征图进行卷积，但论文里的方式**没有把S2层的每个特征图连接到C3的每个特征图**，，这里说的比较抽象，还是放图吧。

Table 1 Each Column Indicates Which Feature Map in S2 Are Combined by the Units in a Particular Feature Map of C3

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

论文里对此做法给出的原因有两点，分别是：

- 不完全连接方案使连接数量保持在合理的范围内
- **打破了网络的对称性**，不同的特征映射被迫**提取不同（希望是互补的）特征。**

  可训练参数：6*(5*5*3+1)+9*(5*5*4+1)+1*(5*5*6+1)=1516个

  卷积后的大小：14-5+1=10即10*10。

S4层

S4层是一个池化层，包含有16个特征图。池化大小为2*2，池化方式与S2相似。

可训练参数：2*16=32个

池化后的大小：(10-2)/2+1=5即5*5。

C5层

C5层是一个卷积层，包含160个特征图。卷积核大小为5*5,步长为1，无填充。

注意，这一层的卷积方式和C3层不同，这里又是**普遍的卷积方式**了。

可训练参数：120*(5*5*16+1)=48120个

卷积后的大小：(5-5+1)=1即1*1。

F6层

F6层是一个全连接层，**包含84个神经单元数**，之所以是84，是由于输出层的设计，之后再说。其中每个单元的值x_i是权重矩阵与输入向量做点积，然后加上一个可训练偏差，值为a_i，然后通过一个激活函数f求得。

$$x_i = f(a_i)$$

$$f(a)=Atanh(Sa), A=1.7159, S=\frac{2}{3}$$

可训练参数：84*120+84=10164个

输出层

输出层由RBF单元组成，每个RBF单元y_i计算如下：

$$y_i=\sum_j(x_j-\omega_{ij})^2$$

其中ω代表的是标准的数字图像的编码矩阵，因为论文中给出的标准图像大小为12*7，所以F6层有84个神经单元。y_i代表的是预测数字i与标准数字i的编码矩阵之间的**欧氏距离**，越小代表预测是数字i的可能性更高。

标准图像如下：

损失函数

原论文中使用的是均方差损失**MSE**，不过还额外增加了惩罚项，现在很少这么设计，因为效果貌似不是很明显，损失函数如下：

$$E(W)=\frac{1}{P}\sum_{p=1}^{P}(y_{Dp}(Z^p,W)+\log(e^{-j}+\sum_i e^{-y_i(Z^P,W)}))$$

代码实现

我这里使用`tensorflow2.4.0`以及`numpy1.20.3`实现**整体结构相似**的LeNet-5

```
#导包
import tensorflow as tf
import numpy as np
import cv2

#下载数据集
(x_train_org,y_train_org),(x_test_org,y_test_org) = tf.keras.datasets.mnist.load_data()

#观察一下数据的维度
print(x_train_org.shape)
```

```
print(y_train_org.shape)
print(x_test_org.shape)
print(y_test_org.shape)

#对数据先改变一下维度方便后面的输入
x_train = x_train_org.reshape(60000,28,28,1)
x_test = x_test_org.reshape(10000,28,28,1)

#归一化数据
x_train = x_train/255.
x_test = x_test/255.

#对标签进行独热编码
y_train = tf.keras.utils.to_categorical(y_train_org,num_classes=10)
y_test = tf.keras.utils.to_categorical(y_test_org,num_classes=10)

#构建网络
model = tf.keras.Sequential([tf.keras.layers.Conv2D(6,
(5,5),padding="same",activation="relu",input_shape=(28,28,1)),
                             tf.keras.layers.AveragePooling2D((2,2)),
                             tf.keras.layers.Conv2D(16,(5,5),activation="relu"),
                             tf.keras.layers.AveragePooling2D((2,2)),
                             tf.keras.layers.Conv2D(120,(5,5),activation="relu"),
                             tf.keras.layers.Flatten(),
                             tf.keras.layers.Dense(84,activation="relu"),
                             tf.keras.layers.Dropout(0.8),
                             tf.keras.layers.Dense(10,activation="softmax")])

model.compile(optimizer="adam",loss="categorical_crossentropy",metrics=["accuracy"])

model.fit(x_train,y_train,epochs=10)

test_loss = model.evaluate(x_test,y_test)

#用自己的图片进行预测
def predict(path):
    img = cv2.imread(path,cv2.IMREAD_UNCHANGED)
    img_grey = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
    new_img = cv2.resize(img_grey,(28,28))
    cv2.imshow('digit',new_img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    prediction = model.predict(new_img.reshape(1,28,28,1))
    classes = tf.argmax(prediction[0]).numpy()
    print(classes)
```

结果

epochs=10

	accuracy	loss
training set	0.9715	0.0816
test set	0.9883	0.0525

模型参数概览

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 6)         156
_________________________________________________________________
average_pooling2d (AveragePo (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
average_pooling2d_1 (Average (None, 5, 5, 16)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 1, 1, 120)         48120
_________________________________________________________________
flatten (Flatten)            (None, 120)               0
_________________________________________________________________
dense (Dense)                (None, 84)                10164
_________________________________________________________________
dropout (Dropout)            (None, 84)                0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
```

备注

我实现的这个网络与原论文还是有**很大区别**的：

- 我实现的输入为28*28，原文中为32*32，但是第一层的卷积层有填充，效果是一样的。
- 原文中是在池化层中使用激活函数 `sigmoid`，而我是在卷积时使用激活函数 `relu`。
- 我的池化层是 `平均池化`。
- 我的F6层用的是 `relu`，并且加上了 `Dropout` 正则化。
- 我的输出层是用了 `softmax` 而非论文中的输出方式。
- 我的损失函数是 `categorical_crossentropy（交叉熵损失函数）` 而非 `MSE`。

但是**整体的架构还是一样**的，读者有兴趣可以自行阅读论文来实现。