

实验理论:

算法原理与伪代码

卷积网络与普通神经网络的区别在，卷积神经网络包含了一个由卷积层(convolution layer)

和子采样层(pooling layer)构成特征抽取器。在卷积网络的卷积层由一个神经元只与部分邻

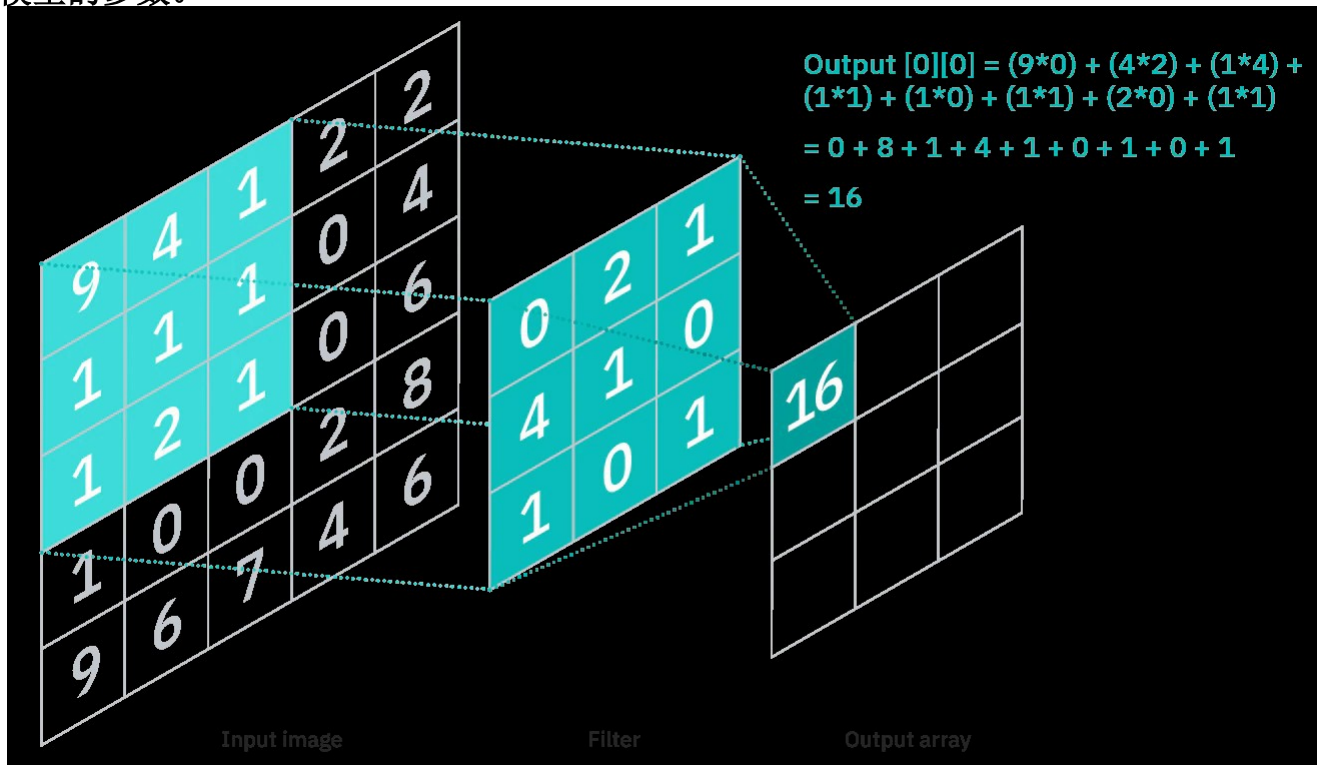
层神经元连接。在 CNN 的卷积里面，通常包含若干个特征图(feature map), 每个特征图有

一些排列的神经元组成，这里共享的权值就是卷积核。卷积核一般以随机小数矩阵的形式

初始化，在网络的训练过程中卷积核将学习得到合理的权值。共享权值(卷积核)带来的

直接好处是减少网络各层之间的连接，同时又降低了过拟合的风险。子采样也叫做池化(pooling)，通常有均值子采样(mean pooling)和最大值子采样(max pooling)两种

形式。子采样可以看作一种特殊的卷积过程。卷积和子采样大大简化了模型复杂度，减少了模型的参数。



卷积层为 CNN 的核心组成部分，也是主要计算发生的地方。它需要一些重要的元素，例如输入数据，过滤器和特征图。假设输入为一个 RGB 图像，则由一个个 pixel 组成的图像，这

意味着图像会是三维，代表图像的 R, G, B 值。我们也有个特征检测器，也叫做内核或者过滤器，他会在图片上的感受野(receptive field)移动，检测是否存在该特征。这一个过程叫做卷积。

特征检测器是一个权重的 2D 数组，代表图像的一部分，通常大小为 3x3。这也就是感受野

的大小。这样，过滤器就应用到该图像的部分，然后计算输入与过滤器之间的点积，然后

该值就成为输出层对应位置的值。之后，过滤器会根据一个 stride 移动，重复过程一直到

整个图像到扫了。最终的输出有一系列的点积就是特征图。

注意到在扫图像的过程中，filter 的值是维持不变的。这个特点叫做 parameter sharing。

有些参数，例如权重值，在训练过程中通过 gradient descent 或者 backpropagation 改变参数

值。但是这些超参量会影响到输出层的参数大小，而这些维度需要在训练神经网络前设定

包括：

过滤器的数量：这影响网络的深度

Stride: 过滤器在图像每次移动的长度

Zero padding: 通常被使用当过滤器不匹配输入图像的大小，这设定所有在图像以外过

滤器的值为 0，产生一个更大或者同大小的输出。有三种 padding:

valid padding: 也叫做 no padding。在这种情况下，如果维度不匹配，最后的卷积就被略到。

Same padding: 这种 padding 保证输出层与输入层的大小相同。

Full padding: 这种 padding 增加输出图像的大小，通过在输入边界增加 0 值。

在每次卷积后，CNN 会应用 ReLU 转变到特征图，从而使得模型有非线性的元素。

Pooling layer 也叫做 Downsampling, 负责降低维度，减小输入图像的参数数量。类似于卷积层，

pooling 层通过在整个图像扫一个过滤器，但不同是过滤器并没有任何的权重。反而，内核

应用一个聚合函数到感应野，最后输出到输出层。有两种主要的 pooling:

max pooling: 随着过滤器在输入图像移动，选择最大值到输出层

average pooling: 随着过滤器在输入图像移动，选择平均值到输出层

Fully connected layer: 输出层的节点直接连接到前一层的节点，这一层通过在前一

层所采样的节点和它的不同过滤器进行分类，通常使用 softmax 函数来分类输入。

实验方法:

我们通过使用 pytorch 来实现我们的基于 cnn 的面型辨别模型。

我们的 cnn 结构大概为三层卷积层，最后一层为全链接层。另外我们在每一个卷积层后面加上批标准化层，ReLU 层，和最后的最大池化层。最后的全链接层为一个线性层，标准化

层, ReLU 层, dropout 层和最后的线性层。当中这个 cnn 结构的批标准化层是一种为神经网络中的任何层提供 0 均值/单位方差输入的技术, 能够改善人工神经网络的稳定性和性能。二 dropout 层以概率 p , 随机将第一个先行曾的神经元的输出置为 0, 作为第二个线性层的输出, 这样可以防止模型的预测过于以来一些特定的神经元。我们使用的 optimizer 是 stochastic gradient descent(SGD), 当中他的算法如下:

输入 X 为样本, n 为样本数量

初始化 $m = c = 0$, max_iterations 为迭代数量

算法:

for $i = 1$ to max_iterations

$$Y^{(i)} = mX + c$$

$$\text{Loss} = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2$$

$$\nabla_m = \frac{-2}{n} \sum_{i=1}^n (X^{(i)} (y^{(i)} - \hat{y}^{(i)}))$$

$$\nabla_c = \frac{-2}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})$$

$$m \leftarrow m - \text{Loss} \nabla_m$$

$$c \leftarrow c - \text{Loss} \nabla_c$$

输出: $y = mX + c$

实验结果:

我们通过 `cnn.py` 迭代 50 次得到最好的训练模型, 并最终在测试集运行结果, 得到 46% 的准确率:

```
(cnnProject) C:\Users\user\OneDrive\Desktop\jin_neural_network>python cnn_inference.py
cuda
[C:\Users\user\anaconda3\envs\cnnProject\lib\site-packages\torch\nn\functional.py:1331: Us
a 2-D input to dropout2d, which is deprecated and will result in an error in a future rel
d silence this warning, please use dropout instead. Note that dropout2d exists to provide
with 2 spatial dimensions, a channel dimension, and an optional batch dimension (i.e. 3D
warnings.warn(warn_msg)
0.46
```