《人工智能与科学计算》

实验报告

题目：	CNN 手写数字识别
姓名：	胡森康
学院：	信息与电子学院
邮箱：	huhbu@outlook.com
日期：	2021 年 6 月 15 日

Content

Figures

Tables

1 实验目的

1. 基于提供的卷积神经网络示例代码，进行代码的优化，例如对 GPU 并行中的一些配置参数的优化。
2. 或基于示例代码，开发 U-Net 等其他当前常用网络。

2 CNN 结构介绍

在利用全连接神经网络处理大尺寸图像具有三个明显的缺点：

1. 将图像展开为向量回丢失空间信息
2. 参数过多效率低下，训练困难
3. 大量参数会导致网络过拟合

而使用卷积神经网络则不同，卷积神经网络中各层中的神经元是三位排列的：宽度、高度和深度。在卷积神经网络中国深度指的是激活数据体的第三个维度，而不是整个网络的深度。

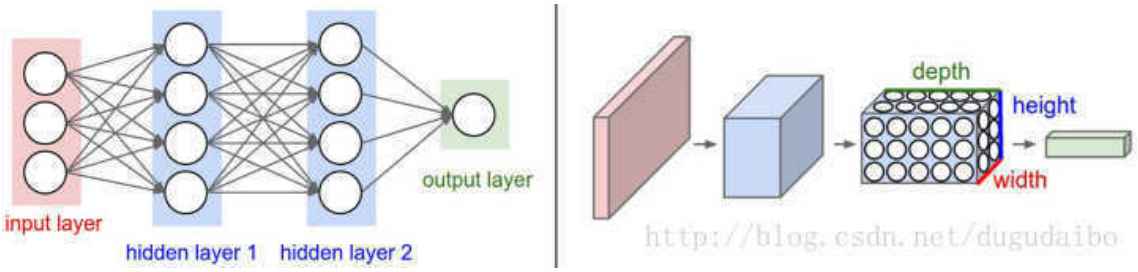

图 2-1: 卷积神经网络和全连接神经网络的对比

图 (2-1) 中左侧是一个三层的神经网络，右侧是一个卷积神经网络，将其神经元在三个维度 (宽度，高度和深度) 进行排列。卷积神经网络的每一层都将 3D 的输入数据变化为神经元 3D 的激活数据并输出。在图 (2-1) 的右侧，红色的输入层代表输入图像，所以它的宽度和高度就是图像的宽度和高度，它的深度是 3(代表了红、绿、蓝 3 种颜色通道)，与红色相邻的蓝色部分是经过卷积和池化之后的激活值 (也可以看做是神经元)，后面是卷积池化层。

3 构建卷积神经网络的层

卷积神经网络主要由这几种层构成：输入层，卷积层，ReLU 层，池化层和全连接层。通过将这些曾叠加起来，就可以构建一个完整的卷积神经网络。在实际应用中往往将卷积层与 ReLU 层共同称之为卷积层，因此卷积层经过卷积操作也是要经过激活函数的。具体来说，卷积层和全连接层对输入执行变换操作的时候，不仅会用到激活函数，还会用到很多参数，即神经元的权值 w 和偏差 b；而 ReLU 层和池化层则是进行一个固定不变的函数操作。卷积层和全连接层中的参数会随着梯度下降被训练，这样卷积神经网络计算出的费雷评分就能和训练基中的每个图像的标签相吻合了。

3.1 卷积层

卷积层是构建卷积神经网络的核心层，他产生网络中的大部分计算量。

3.1.1 卷积层的作用

1. 作为滤波器。卷积层的参数是由一些科学系的滤波器集合构成的，每个滤波器在高度和宽度上都比较小，但是深度和输入数据一致。
2. 可以看作是一个神经元的输出。神经元只观察输入数据的一小部分，并且和空间上左右两边的所有神经元共享参数。
3. 降低参数的数量。由于卷积有“权值共享”这样的特征，可以降低参数数量，达到降低计算开销，放置由于参数过多而造成过拟合。

3.1.2 感受野

在处理图像这样的高维度输入时，让每一个神经元都与前一层中的所有神经元进行全连接是不现实的。相反，让每一个神经元只与输入数据的一个局部区域链接，该链接的空间大小叫做神经元的感受野 (Receptive Field)，其尺寸是一个超参数。在深度方向上，此连接的大小总是和输入量的深度相等。

在图 (3-2) 中展现的卷积神经网络的一部分，其中的红色为输入数据，假设输入数据体尺寸为 $[32\times32\times3]$，如果感受野 (或滤波器尺寸) 是 5×5，那么卷积层中的每个神经元会有输入数据体中 $[5\times5\times3]$ 区域的权重，$5\times5\times3=75$ 个权重。注意这个连接在深度维度上的大小必须为 3，和输入数据体的深度一致。其中还有一点需要注意，对应一个感受野有 75 个权重，这 75 个权重是通过学习进行更新的，所以很大程度上这

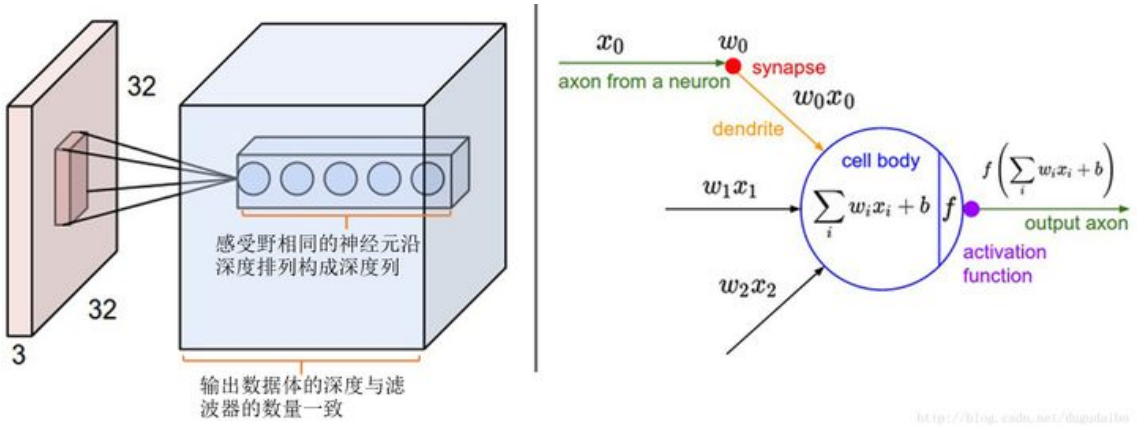

图 3-2: 感受野的连接和尺寸说明

些权值之间是不相等。在这里相当于前面的每一个层对应一个传统意义上的卷积模板，每一层与自己卷积模板做完卷积之后，再将各个层的结果加起来，再加上一个偏置。

3.1.3 神经元的空间排列

感受野讲解了卷积层中每个神经元与输入数据体之间的连接方式，但是尚未讨论输出数据体中神经元的数量，以及它们的排列方式。3 个超参数控制着输出数据体的尺寸：深度 (depth)，步长 (stride) 和零填充 (zero-padding)。

1. 输出数据体的深度。它是一个超参数，和使用的滤波器的数量一致，而每个滤波器在输入数据中寻找一些不同的东西，即图像的某些特征。如图 (3-2) 所示，将沿着深度方向排列、感受野相同的神经元集合称为深度列 (depth column)
2. 在滑动滤波器的时候，必须指定步长。当步长为 1，滤波器每次移动 1 个像素；当步长为 2，滤波器滑动时每次移动 2 个像素，当然步长也可以是不常用的 3，或者更大的数字，但这些在实际中很少使用)。这个操作会让输出数据体在空间上变小。
3. 有时候将输入数据体用 0 在边缘处进行填充是很方便的。这个零填充 (zero-padding) 的尺寸是一个超参数。零填充有一个良好性质，即可以控制输出数据体的空间尺寸 (最常用的是用来保持输入数据体在空间上的尺寸，使得输入和输出的宽高都相等)。

输出数据体在空间上的尺寸 $W_2 \times H_2 \times D_2$ 可以通过输入数据尺寸 $W_1 \times H_1 \times D_1$，卷

积层中神经元的感受野尺寸 F，步长 S，滤波器数量 K 和零填充数量 P 计算得到。

$$W_2 = \frac{W_1 - F + 2P}{S} + 1 \tag{3-1}$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1 \tag{3-2}$$

$$D_2 = K \tag{3-3}$$

一般来说，当步长 $S = 1$ 时，零填充的值为 $P = (F - 1)/2$，这样就能保证输入和输出数据体有相同的空间尺寸。

3.1.4 权值共享

在卷积层中权值共享是用来控制参数的数量。假如在一个卷积核中，每一个感受野采用的都是不同的权重值 (卷积核的值不同)，那么这样的网络中参数数量将是十分巨大的。

权值共享是基于这样的一个合理的假设：如果一个特征在计算某个空间位置 (x_1, y_1) 的时候有用，那么它在计算另一个不同位置 (x_2, y_2) 的时候也有用。基于这个假设，可以显著地减少参数数量。换言之，就是将深度维度上一个单独的 2 维切片看做深度切片 (depth slice)，比如一个数据体尺寸为 $[55 \times 55 \times 96]$ 的就有 96 个深度切片，每个尺寸为 $[55 \times 55]$，其中在每个深度切片上的结果都使用同样的权重和偏差获得的。在这样的参数共享下，假如一个例子中的第一个卷积层有 96 个卷积核，那么就有 96 个不同的权重集了，一个权重集对应一个深度切片，如果卷积核的大小是 11×11 的，图像是 RGB 为 3 通道，那么就共有 $96 \times 11 \times 11 \times 3 = 34848$ 个不同的权重，总共有 34944 个参数 (因为要加上 96 个偏差)，并且在每个深度切片中的 55×55 的结果使用的都是同样的参数。

在反向传播的时候，都要计算每个神经元对它的权重的梯度，但是需要把同一个深度切片上的所有神经元对权重的梯度累加，这样就得到了对共享权重的梯度。这样，每个切片只更新一个权重集。这样做的原因可以通过图 (3-3) 进行解释。

如图 (3-3) 所示，左侧的神经元是将每一个感受野展开为一列之后串联起来 (就是展开排成一列，同一层神经元之间不连接)。右侧的 Deep1i 是深度为 1 的神经元的第 i 个，Deep2i 是深度为 2 的神经元的第 i 个，同一个深度的神经元的权值都是相同的，黄色的都是相同的 (上面 4 个与下面 4 个的参数相同)，蓝色都是相同的。所以现在回过头来看上面说的卷积神经网络的反向传播公式对梯度进行累加求和也是基于这点考虑 (同一深度的不同神经元共用一组参数，所以累加)；而每个切片只更新一个权重集的原

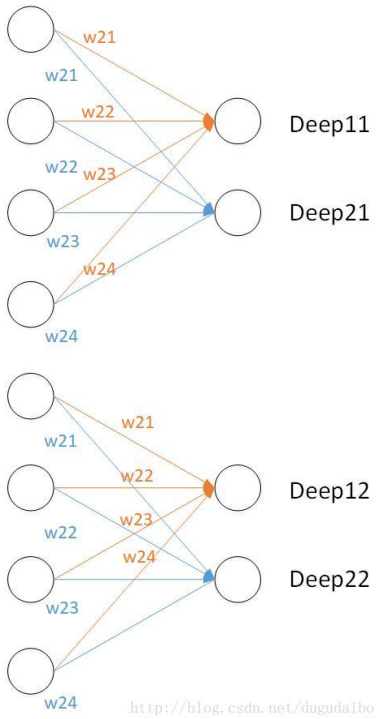

图 3-3: 将卷积层用全连接层的形式表示

因也是这样的，因为从图 3 中可以看到，不同深度的神经元不会公用相同的权重，所以只能更新一个权重集。

3.1.5 卷积层的超参数及选择

由于参数共享，每个滤波器包含 $F \times F \times D_1$ 个权重，卷积层一共有 $F \times F \times D_1 \times K$ 个权重和 K 个偏置。在输出数据体中，第 d 个深度切牌 (空间尺寸为 $W_2 \times H_2$)，用第 d 个滤波器和输入数据进行有效卷积运算的结果 (使用步长 S)，最后在加上第 d 个偏差。

3.1.6 实现卷积的操作

1. 矩阵乘法实现卷积：

   卷积运算本质上就是在滤波器和输入数据的局部区域间做点积。卷积层的常用实现方式就是利用这一点，将卷积层的前向传播变成一个巨大的矩阵乘法。

2. 1×1 卷积：

   一些论文中使用了 1×1 的卷积，这个方法最早是在论文 Network in Network 中出现。人们刚开始看见这个 1×1 卷积的时候比较困惑，尤其是那些具有信号处理专业背景的人。因为信号是 2 维的，所以 1×1 卷积就没有意义。但是，在卷积神经网络中不是这样，因为这里是对 3 个维度进行操作，滤波器和输入数据体的深

度是一样的。比如，如果输入是 $[32 \times 32 \times 3]$，那么 1×1 卷积就是在高效地进行 3 维点积；另外的一种想法是将这种卷积的结果看作是全连接层的一种实现方式。

3. 扩张卷积：

最近一个研究给卷积层引入了一个新的叫扩张 (dilation) 的超参数。到目前为止，我们只讨论了卷积层滤波器是连续的情况。但是，让滤波器中元素之间有间隙也是可以的，这就叫做扩张，如图 (3-4) 所示。

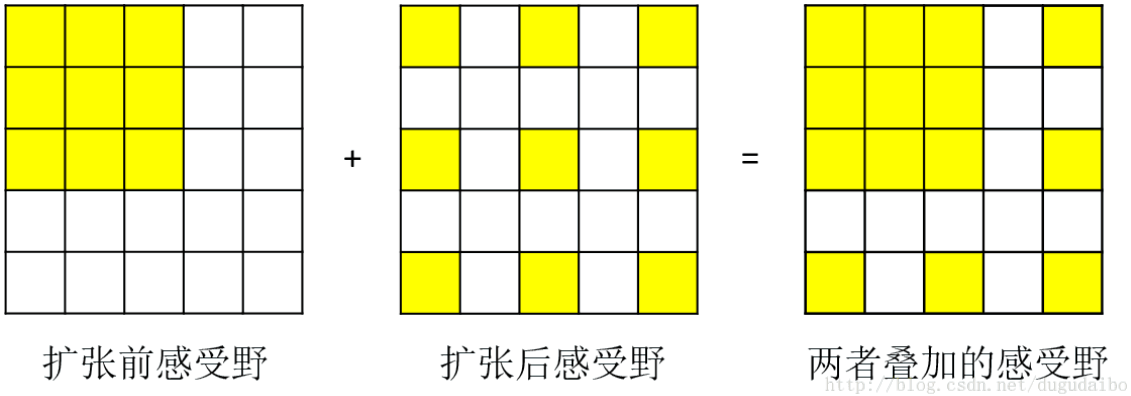

图 3-4: 扩张卷积示意图及扩张前后的叠加效果

3.2 池化层

通常在连续的卷积层之间会周期性地插入一个池化层。它的作用是逐渐降低数据体的空间尺寸，这样的话就能减少网络中参数的数量，使得计算资源耗费变少，也能有效控制过拟合。汇聚层使用 MAX 操作，对输入数据体的每一个深度切片独立进行操作，改变它的空间尺寸。最常见的形式是汇聚层使用尺寸 2×2 的滤波器，以步长为 2 来对每个深度切片进行降采样，将其中 75% 的激活信息都丢掉。每个 MAX 操作是从 4 个数字中取最大值 (也就是在深度切片中某个 2×2 的区域)，深度保持不变。

3.3 归一化层

在卷积神经网络的结构中，提出了很多不同类型的归一化层，有时候是为了实现在生物大脑中观测到的抑制机制。但是这些层渐渐都不再流行，因为实践证明它们的效果即使存在，也是极其有限的。

3.4 全连接层

全连接层和常规神经网络中一样，它们的激活可以先用矩阵乘法，再加上偏差。

3.4.1 将卷积层转化为全连接层

对于任一个卷积层，都存在一个能实现和它一样的前向传播函数的全连接层。该全连接层的权重是一个巨大的矩阵，除了某些特定块 (感受野)，其余部分都是零；而在非 0 部分中，大部分元素都是相等的 (权值共享)。如果把全连接层转化成卷积层，以输出层的 Deep11 为例，与它有关的输入神经元只有上面四个，所以在权重矩阵中与它相乘的元素，除了它所对应的 4 个，剩下的均为 0，这也就解释了为什么权重矩阵中有为零的部分；另外要把“将全连接层转化成卷积层”和“用矩阵乘法实现卷积”区别开，这两者是不同的，后者本身还是在计算卷积，只不过将其展开为矩阵相乘的形式，并不是”将全连接层转化成卷积层”，所以除非权重中本身有零，否则用矩阵乘法实现卷积的过程中不会出现值为 0 的权重。

3.4.2 将全连接层转化为卷积层

任何全连接层都可以被转化为卷积层。比如，一个 $K = 4096$ 的全连接层，输入数据体的尺寸是 $7 \times 7 \times 5127 \times 7 \times 512$，这个全连接层可以被等效地看做一个 $F = 7, P = 0, S = 1, K = 4096, F = 7, P = 0, S = 1, K = 4096$ 的卷积层。换句话说，就是将滤波器的尺寸设置为和输入数据体的尺寸设为一致的。因为只有一个单独的深度列覆盖并滑过输入数据体，所以输出将变成 $1 \times 1 \times 40961 \times 1 \times 4096$，这个结果就和使用初始的那个全连接层相同。对于其中的一个卷积滤波器，这个滤波器的的深度为 512，因此虽然这个卷积滤波器的输出只有 1 个，但是它的权重有 $7 \times 7 \times 5127 \times 7 \times 512$，相当于卷积滤波器的输出为一个神经元，这个神经元与上一层的所有神经元相连接，而这样与前一层所有神经元相连接的神经元一共有 4096 个，即为一个全连接网络。

在上述的两种变换中，将全连接层转化为卷积层在实际运用中更加有用。假设一个卷积神经网络的输入是 $224 \times 224 \times 3$ 的图像，一系列的卷积层和汇聚层将图像数据变为尺寸为 $7 \times 7 \times 512$ 的激活数据体 (在 AlexNet 中就是这样，通过使用 5 个汇聚层来对输入数据进行空间上的降采样，每次尺寸下降一半，所以最终空间尺寸为 $224/2/2/2/2/2 = 7$)。从这里可以看到，AlexNet 使用了两个尺寸为 4096 的全连接层，最后一个有 1000 个神经元的全连接层用于计算分类评分。我们可以将这 3 个全连接转化为 3 个卷积层：

1. 针对第一个连接区域是 $[7 \times 7 \times 512]$ 的全连接层，令其滤波器尺寸为 $F = 7$，这样输出数据体就为 $[1 \times 1 \times 4096]$ 了。
2. 针对第二个全连接层，令其滤波器尺寸为 $F = 1$，这样输出数据体为 $[1 \times 1 \times 4096]$。

3. 对最后一个全连接层也做类似的，令其 $F = 1$，最终输出为 $[1 \times 1 \times 1000]$。

这样做的目的是让卷积网络在一张更大的输入图片上滑动，得到多个输出，这样的转化可以让我们在单个向前传播的过程中完成上述的操作。

4 卷积神经网络结构

卷积神经网络通常是由三种层构成：卷积层，汇聚层 (除非特别说明，一般就是最大值汇聚) 和全连接层 (简称 FC)。ReLU 激活函数也应该算是是一层，它逐元素地进行激活函数操作，常常将它与卷积层看作是同一层。

4.1 层的排列规律

卷积神经网络最常见的形式就是将一些卷积层和 ReLU 层放在一起，其后紧跟汇聚层，然后重复如此直到图像在空间上被缩小到一个足够小的尺寸，在某个地方过渡成成全连接层也较为常见。最后的全连接层得到输出，比如分类评分等。换句话说，最常见的卷积神经网络结构如下：

$$INPUT \rightarrow [CONV \rightarrow RELU] * N \rightarrow [POOL?] * M \rightarrow [FC \rightarrow RELU] * K \rightarrow FC$$

其中 $*$ 指的是重复次数，$POOL?$ 指的是一个可选的汇聚层。其中 $N >= 0$, 通常 $N <= 3, M >= 0, K >= 0$，通常 $K < 3$。

4.2 卷积层大小的选择

几个小滤波器卷积层的组合比一个大滤波器卷积层好。假设一层一层地重叠了 3 个 3×3 的卷积层 (层与层之间有非线性激活函数)。在这个排列下，第一个卷积层中的每个神经元都对输入数据体有一个 3×3 的视野。第二个卷积层上的神经元对第一个卷积层有一个 3×3 的视野，也就是对输入数据体有 5×5 的视野。同样，在第三个卷积层上的神经元对第二个卷积层有 3×3 的视野，也就是对输入数据体有 7×7 的视野。假设不采用这 3 个 3×3 的卷积层，二是使用一个单独的有 7×7 的感受野的卷积层，那么所有神经元的感受野也是 7×7，但是就有一些缺点。首先，多个卷积层与非线性的激活层交替的结构，比单一卷积层的结构更能提取出深层的更好的特征。其次，假设所有的数据有 C 个通道，那么单独的 7x7 卷积层将会包含 $C \times (7 \times 7 \times C) = 49C^2$ 个参数，而 3 个 7×7 的卷积层的组合仅有 $3 \times C \times (3 \times 3 \times C) = 27C^2$ 个参数。直观说来，最好选择带有小滤波器的卷积层组合，而不是用一个带有大的滤波器的卷积层。前者可以表

达出输入数据中更多个强力特征，使用的参数也更少。唯一的不足是，在进行反向传播时，中间的卷积层可能会导致占用更多的内存。

4.3 层的尺寸设置规律

1. 输入层

   大小应为 2 的幂次或可多次被 2 整除。常用数字包括 32(比如 CIFAR-10)，64，96(比如 STL-10) 或 224(比如 ImageNet 卷积神经网络)，384 和 512。

2. 卷积层

   应该使用小尺寸滤波器，使用步长 $S = 1$。还有一点非常重要，就是对输入数据进行零填充，这样卷积层就不会改变输入数据在空间维度上的尺寸。比如，当 $F = 3$，那就使用 $P = 1$ 来保持输入尺寸。当 $F = 5, P = 2$，一般对于任意 F，当 $P = (F - 1)/2$ 的时候能保持输入尺寸。如果必须使用更大的滤波器尺寸，通常只用在第一个面对原始图像的卷积层上。

3. 汇聚层

   负责对输入数据的空间维度进行降采样。最常用的设置是用用 2×2 感受野 (即 $F = 2$) 的最大值汇聚，步长为 2($S = 2$)。注意这一操作将会把输入数据中 75% 的激活数据丢弃 (因为对宽度和高度都进行了 2 的降采样)。另一个不那么常用的设置是使用 3×3 的感受野，步长为 2。最大值汇聚的感受野尺寸很少有超过 3 的，因为汇聚操作过于激烈，易造成数据信息丢失，这通常会导致算法性能变差。

5 CNN 计算上的考量

在构建卷积神经网络结构时，最大的瓶颈是内存瓶颈，所以如何降低内存消耗量是一个值得思考的问题。三种内存占用来源：

1. 来自中间数据体尺寸：

   卷积神经网络中的每一层中都有激活数据体的原始数值，以及损失函数对它们的梯度 (和激活数据体尺寸一致)。通常，大部分激活数据都是在网络中靠前的层中 (比如第一个卷积层)。在训练时，这些数据需要放在内存中，因为反向传播的时候还会用到。但是在测试时可以聪明点：让网络在测试运行时候每层都只存储当前的激活数据，然后丢弃前面层的激活数据，这样就能减少巨大的激活数据量。这实际上是底层问题，在编写框架的过程中，设计者会进行这方面的考虑。

2. 来自参数尺寸：

即整个网络的参数的数量，在反向传播时它们的梯度值，以及使用 momentum、Adagrad 或 RMSProp 等方法进行最优化时的每一步计算缓存。因此，存储参数向量的内存通常需要在参数向量的容量基础上乘以 3 或者更多。

3. 卷积神经网络实现还有各种零散的内存占用，比如成批的训练数据，扩充的数据等等。

一旦对于所有这些数值的数量有了一个大略估计 (包含激活数据，梯度和各种杂项)，数量应该转化为以 GB 为计量单位。把这个值乘以 4，得到原始的字节数 (因为每个浮点数占用 4 个字节，如果是双精度浮点数那就是占用 8 个字节)，然后多次除以 1024 分别得到占用内存的 KB，MB，最后是 GB 计量。如果你的网络工作得不好，一个常用的方法是降低批尺寸 (batch size)，因为绝大多数的内存都是被激活数据消耗掉了。

6 实验过程

6.1 `Seq_CNN_course` 代码分析

对 `Seq_CNN_course` 中的代码进行分析。在 `Seq_CNN_course` 中，主要由以下几个部分组成：

1. `drv_cnn`：

此部分是卷积神经网络的驱动程序，是顶层程序。

2. `Activation_layer`：

此函数是激活层，含有激活函数。所谓激活函数，就是在人工神经网络的神经元上运行的函数，负责将神经元的输入映射到输出端。

激活函数对于人工神经网络模型去学习、理解非常复杂和非线性的函数来说具有十分重要的作用。它们将非线性特性引入到我们的网络中。且在本程序中，采用的激活函数

在本程序中，在类中定义了三种激活函数，分别为 tanh 函数，ReLU 函数，Sigmoid 函数，并分别定义了三种激活函数的梯度函数。下面对这三种激活函数作一简单介绍。

- Sigmoid 函数：Sigmoid 函数是一个在生物学中常见的 S 型函数，也称为 S 型生长曲线。在信息科学中，由于其单增以及反函数单增等性质，Sigmoid 函数常被用作神经网络的阈值函数，将变量映射到 0 至 1 之间。公式如下，其函数图像如

图 (6-5) 所示。

$$f(x) = \frac{1}{1 + e^{-x}} \tag{6-4}$$

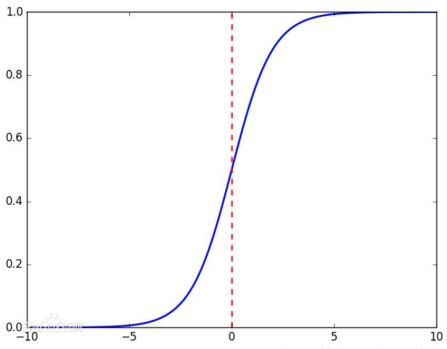

图 6-5: Sigmoid 函数图像

- tanh 函数：tanh 是双曲函数中的一个，tanh() 为双曲正切。在数学中，双曲正切 tanh 是由基本双曲函数双曲正弦和双曲余弦推导而来。公式如下，其函数图像如图 (6-6) 所示。

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{6-5}$$

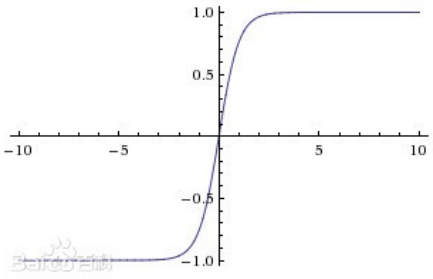

图 6-6: tanh 函数图像

- ReLU 函数：Relu 激活函数 (The Rectified Linear Unit)，用于隐层神经元输出。公式如下, 其函数图像如图 (6-7) 所示。

$$f(x) = \max(0, x) \tag{6-6}$$

在程序 `activation_layer.cc` 中，利用的激活函数为 Tanh，以此来实现非线性效应。

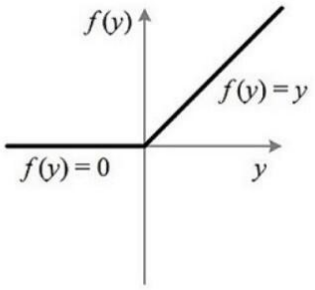

图 6-7: ReLU 函数图像

3. `convolutional_layer`

此部分为卷积层，在头文件中定义了卷积核大小，步长，零填充尺寸等超参数。在.cc 文件中，主要有三个函数：初始化权重，前向卷积和后向卷积。

4. `pooling_layer`

此部分为池化层，同样在头文件中定义了一些超参数，与卷积层中的头文件几乎一样，只是仅仅少了几个超参数。在本池化层，采用的是最常见的最大池化层，池化层的目的主要是模仿人的视觉系统对数据进行降维，用更高层次的特征表示图像。

5. `fully_connected_layer`

此部分为全连接层。同样的，在头文件中定义了一些超参数，在.cc 文件中，主要有三个函数：初始化权重，前向和后向传播函数。

当来到了全连接层之后，可以理解为一个简单的多分类神经网络 (如：BP 神经网络)，通过 softmax 函数得到最终的输出。整个模型训练完毕。两层之间所有神经元都有权重连接，通常全连接层在卷积神经网络尾部。也就是跟传统的神经网络神经元的连接方式是一样的。

6. `softmax_layer`

本层为 softmax 层，softmax 层只是对神经网络的输出结果进行了一次换算，将输出结果用概率的形式表现出来。

6.2 `Cuda_CNN_ptr` 代码分析比较

6.2.1 CUDA 简介

CUDA 是 NVIDIA 公司所开发的 GPU 编程模型，它提供了 GPU 编程的简易接口，基于 CUDA 编程可以构建基于 GPU 计算的应用程序。CUDA 提供了对其它编程

语言的支持，如 C/C++，Python，Fortran 等语言。基于 CUDA 编程可以利用 GPUs 的并行计算引擎来更加高效地解决比较复杂的计算难题。近年来，GPU 最成功的一个应用就是深度学习领域，基于 GPU 的并行计算已经成为训练深度学习模型的标配。目前，最新的 CUDA 版本为 CUDA 9。

GPU 并不是一个独立运行的计算平台，而需要与 CPU 协同工作，可以看成是 CPU 的协处理器，因此当我们在说 GPU 并行计算时，其实是指的基于 CPU+GPU 的异构计算架构。在异构计算架构中，GPU 与 CPU 通过 PCIe 总线连接在一起来协同工作，CPU 所在位置称为为主机端 (host)，而 GPU 所在位置称为设备端 (device)，如下图 (6-8) 所示。

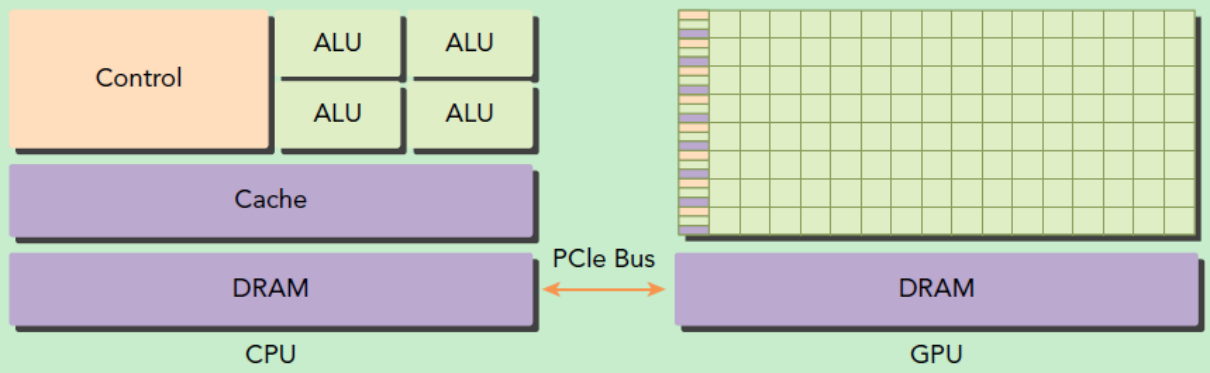

图 6-8: 基于 CPU+GPU 的异构计算

可以看到 GPU 包括更多的运算核心，其特别适合数据并行的计算密集型任务，如大型矩阵运算，而 CPU 的运算核心较少，但是其可以实现复杂的逻辑运算，因此其适合控制密集型任务。另外，CPU 上的线程是重量级的，上下文切换开销大，但是 GPU 由于存在很多核心，其线程是轻量级的。因此，基于 CPU+GPU 的异构计算平台可以优势互补，CPU 负责处理逻辑复杂的串行程序，而 GPU 重点处理数据密集型的并行计算程序，从而发挥最大功效。

6.2.2 CUDA 编程模型基础

在给出 CUDA 的编程实例之前，这里先对 CUDA 编程模型中的一些概念及基础知识做个简单介绍。CUDA 编程模型是一个异构模型，需要 CPU 和 GPU 协同工作。在 CUDA 中，host 和 device 是两个重要的概念，我们用 host 指代 CPU 及其内存，而用 device 指代 GPU 及其内存。CUDA 程序中既包含 host 程序，又包含 device 程序，它们分别在 CPU 和 GPU 上运行。同时，host 与 device 之间可以进行通信，这样它们之间可以进行数据拷贝。典型的 CUDA 程序的执行流程如下：

1. 分配 host 内存，并进行数据初始化；
2. 分配 device 内存，并从 host 将数据拷贝到 device 上；
3. 调用 CUDA 的核函数在 device 上完成指定的运算；
4. 将 device 上的运算结果拷贝到 host 上；
5. 释放 device 和 host 上分配的内存。

上面流程中最重要的一个过程是调用 CUDA 的核函数来执行并行计算，kernel 是 CUDA 中一个重要的概念，kernel 是在 device 上线程中并行执行的函数，核函数用 `__global__` 符号声明，在调用时需要用 `<<<grid, block>>>` 来指定 kernel 要执行的线程数量，在 CUDA 中，每一个线程都要执行核函数，并且每个线程会分配一个唯一的线程号 thread ID，这个 ID 值可以通过核函数的内置变量 `threadIdx` 来获得。

由于 GPU 实际上是异构模型，所以需要区分 host 和 device 上的代码，在 CUDA 中是通过函数类型限定词开区别 host 和 device 上的函数，主要的三个函数类型限定词如下：

- `__global__`：在 device 上执行，从 host 中调用 (一些特定的 GPU 也可以从 device 上调用)，返回类型必须是 `void`，不支持可变参数参数，不能成为类成员函数。注意用 `__global__` 定义的 kernel 是异步的，这意味着 host 不会等待 kernel 执行完就执行下一步。
- `__device__`:在 device 上执行,单仅可以从 device 中调用,不可以和 `__global__` 同时用。
- `__host__`：在 host 上执行，仅可以从 host 上调用，一般省略不写，不可以和 `__global__` 同时用，但可和 `__device__`，此时函数会在 device 和 host 都编译。

要深刻理解 kernel，必须要对 kernel 的线程层次结构有一个清晰的认识。首先 GPU 上很多并行化的轻量级线程。kernel 在 device 上执行时实际上是启动很多线程，一个 kernel 所启动的所有线程称为一个网格 (grid)，同一个网格上的线程共享相同的全局内存空间，grid 是线程结构的第一层次，而网格又可以分为很多线程块 (block)，一个线程块里面包含很多线程，这是第二个层次。线程两层组织结构如下图所示，这是一个 gird 和 block 均为 2-dim 的线程组织。grid 和 block 都是定义为 `dim3` 类型的变量，`dim3` 可以看成是包含三个无符号整数 (x, y, z) 成员的结构体变量，在定义时，缺省值初始化为 1。因此 grid 和 block 可以灵活地定义为 1-dim，2-dim 以及 3-dim 结构，对于图 (6-9) 中结构 (主要水平方向为 x 轴)，定义的 `grid` 和 `block` 如下所示，kernel 在调用

时也必须通过执行配置 `<<<grid, block>>>` 来指定 kernel 所使用的线程数及结构。

```
dim3 grid(3, 2);
dim3 block(5, 3);
kernel_fun<<< grid, block >>>(prams...);
```

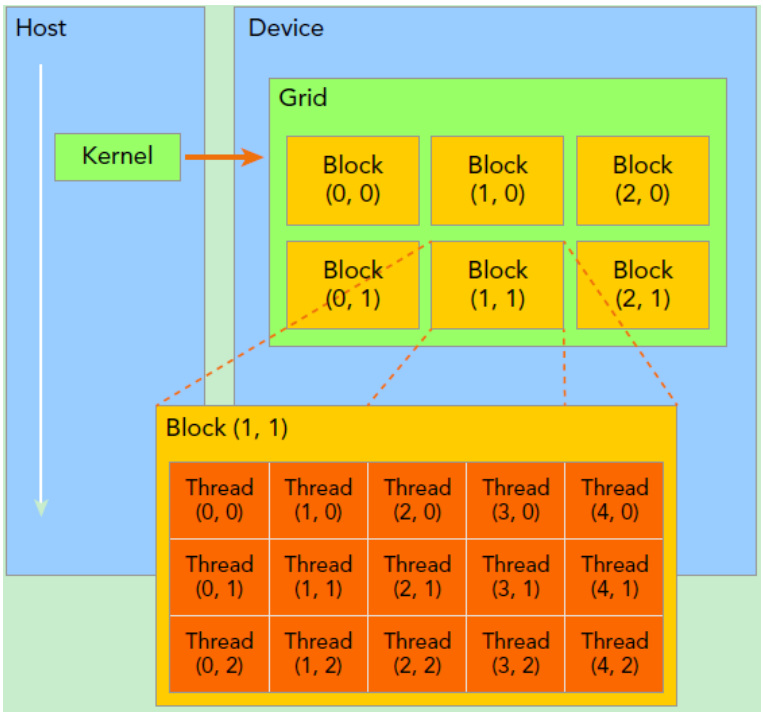

图 6-9: Kernel 上的两层线程组织结构 (2-dim)

所以，一个线程需要两个内置的坐标变量 (blockIdx，threadIdx) 来唯一标识，它们都是 `dim3` 类型变量，其中 `blockIdx` 指明线程所在 grid 中的位置，而 `threaIdx` 指明线程所在 block 中的位置，如图 (6-9) 中的 Thread(1,1) 满足：

```
threadIdx.x = 1
threadIdx.y = 1
blockIdx.x = 1
blockIdx.y = 1
```

一个线程块上的线程是放在同一个流式多处理器 (SM) 上的，但是单个 SM 的资源有限，这导致线程块中的线程数是有限制的，现代 GPUs 的线程块可支持的线程数可达 1024 个。有时候，我们要知道一个线程在 blcok 中的全局 ID，此时就必须还要知道 block 的组织结构，这是通过线程的内置变量 blockDim 来获得。它获取线程块各个维度的大小。对于一个 2-dim 的 $block(D_x, D_y)$ ，线程 (x, y) 的 ID 值为 $(x + y \times D_x)$，如果

是 3-dim 的 $block(D_x, D_y, D_z)$，线程 (x, y, z) 的 ID 值为 $(x + y \times D_x + z \times D_z \times D_y)$。另外线程还有内置变量 `gridDim`，用于获得网格块各个维度的大小。

6.2.3 代码分析比较

对 `Cuda_CNN_ptr` 中的代码进行分析。并与 `Seq_CNN_course` 中的代码进行比较，`Cuda_CNN_ptr` 程序主要由以下几个部分组成：

1. `drv_cnn`
2. `Activation_layer`
3. `convolutional_layer`
4. `pooling_layer`
5. `fully_connected_layer`
6. `softmax_layer`

此程序主页也是由此六大程序组成，分别为顶层程序、激活层、卷积层、池化层、全连接层和 softmax 层组成，其结构与 `Seq_CNN_course` 基本相同，可以认为结构基本一样，只是在每一层的具体实现上，利用了 CUDA 编程，利用 GPU 并行加速程序运行时间。

6.3 更改网络超参数 (炼丹)

6.3.1 学习率调参

更改 `Cuda_CNN_ptr` 程序中的网络学习率。

首先令学习率为 $learning\_rate = 0.01 \times \sqrt{batch\_size\_train} = 0.01\sqrt{240}$，对程序进行测试。然后再令 $learning\_rate = 0.01 \times k \times \sqrt{240},\ k = 2, 3, \cdots$，分别进行测试，并分别统计训练时间、准确率、测试时间，得到结果如下表 (6-1) 所示：

表 6-1: 学习率测试结果

k	1	2	3	4	5	10	20	30
训练时间 (ms)	266.73	265.86	268.17	265.77	268.22	265.63	267.40	267.28
测试时间 (ms)	41.76	41.93	41.90	42.04	41.98	42.25	42.16	41.73
准确率 (%)	94.58	95.98	96.73	96.71	96.77	96.94	95.68	20.32

通过上表 (6-1) 可以看到，学习率的改变对训练时间和测试时间基本没有影响，但是对准确率有很大影响，当 $k = 10$ 学习率为 $0.1\sqrt{240}$ 的时候，可以看到准确率是最高

的，相比学习率为 0.01 $\sqrt{240}$ 时，提高了 2.5 个百分点，可见提高效果显著。

但是当学习率持续增加时，可以看到识别准确率开始下跌，最终开始断崖式下跌，导致网络基本丧失识别功能。

好的学习率应有利于网络学习到参数，即损失函数能够有效的降低。如图 (6-10) 所示，需要函数能够快速的到达谷底。如果学习率太大，则步长过大，会导致损失函数不能收敛。如果学习率太小，就可能学得太慢，同样也不可取。

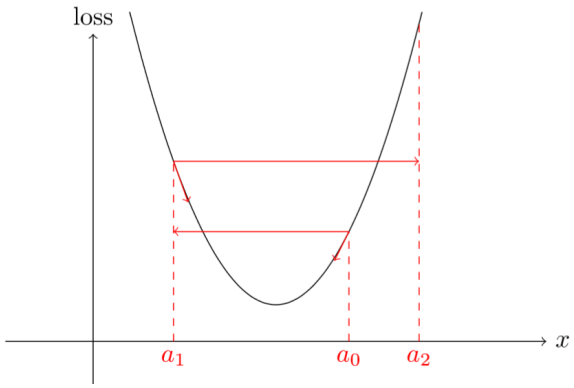

图 6-10: 损失函数的梯度下降示意图

因此在表 (6-1) 中，当学习率过大时，会使损失函数不收敛，导致准确率断崖式下跌，使网络丧失识别功能。

学习率的选取在网络学习中是很重要的，直接关系到网络的性能，在此简单介绍一下学习率的的选取方法。如图 (6-11) 所示，可以尝试不同的初始学习率，然后可以以 5

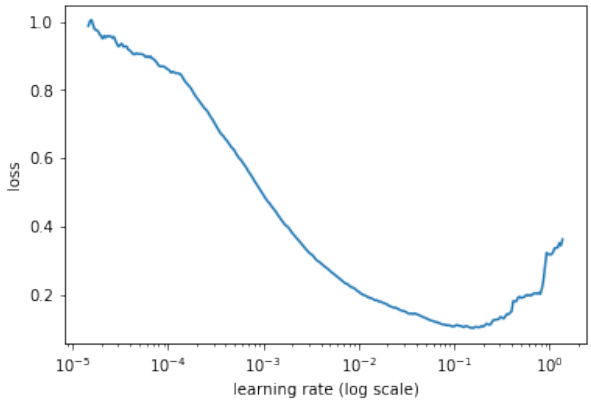

图 6-11: 学习率和损失函数的函数关系

至 10 轮的训练为标准，比较最终学习效果。一般学习率可以从 0.1 ~ 10^{-8} 或者更大的范围去搜索，可以每次缩小 10 倍，这样做的好处是，可以快速的找到一个能够快速收

敛的学习率。

一些流行的学习率调整方法有：

- 循环学习率：
  如图 (6-12) 所示的三角学习率方法，蓝色线条代表学习率在上下限之间进行变化，输入参数 *stepsize* (步长) 是半个周期内的迭代次数。

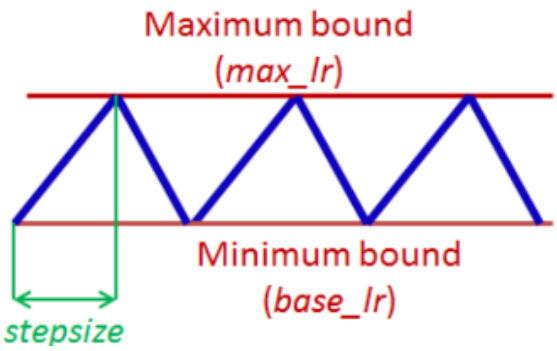

图 6-12: 循环学习率示意图

- 热启动的随机梯度下降 (SGD)：
  循环学习率是讲述的总体学习率调节技巧，然而如果我们不使用预训练的网络，我们应该如何开始我们的网络学习过程呢，就有学者提出热启动的概念。即在刚开始网络经过初始化之后，需要经过一定的调整之后才能达到一个快速学习的状态，需要经过一些柔性的学习率调节之后就能达到一个更加好的初始化状态，这可看作是对网络初始化的一个补充。

6.3.2 改变卷积核步长 (Stride)

初始化程序`Seq_CNN_course`中卷积层的步长 (Stride) 为 1，运行程序得到训练时间为 27571.64ms，预测时间为 1502.56ms，正确率为 95.78%。

改变程序`Seq_CNN_course`中卷积层的步长 (Stride)，将步长改为 2，运行程序，得到训练时间为 6657.11ms，预测时间为 412.14ms，正确率为 10.28%。

可见,更改步长后训练时间从原来的 27571.64ms 变为 6657.11ms,易知 27571/6657 ≈ 4，这和步长变为原来的两倍是一致的。且更改步长后，网络丧失数字识别能力，可见选取合适的步长是非常重要的，否则会导致网络遗漏掉图片中的信息，不能学习到参数。

将步长改为 3，则出现了错误：Segmentation fault (core dumped)，将步长改为 4，也出现了同样的错误。

6.3.3 池化层调参

改变池化层窗口大小，统计训练时间及准确率，见下表 (6-2) 所示。

表 6-2: 改变池化层窗口大小，训练时间及准确率的统计

窗口大小	1	2	3	4	5	10
训练时间 (ms)	26366.30	27739.95	28699.36	28904.23	30576.52	31098.66
准确率 (%)	92.44	94.58	92.03	88.19	86.37	84.59

通过表 (6-2) 可以看到，随着池化层窗口的增大导致训练时间边长，同时准确率也下降，从 90% 以上下降至 85% 以下，这是符合预期的。

首先解释时间增大的原因，此池化层是最大池化层，且在池化窗口中寻找最大值时采用双重 for 循环，因此随着池化窗口的增大，增大的最大值寻找的时间，导致最终训练时间增大；再解释准确率下降的原因，由于采用最大池化层，因此在增大池化层窗口的过程中，会丢失更多的信息，导致准确率下降，一般池化窗口采用 2×2 大小。

池化层的介绍见前文。

6.4 运行速度的比较

在本节，主要对训练的速度进行一个比较，比较 GPU 并行和一般程序的运行速度，并对 CUDA 编程进行介绍。

在程序 `Seq_CNN_course` 中，设置学习率为 $0.01\sqrt{240}$，设置训练集共 250 个批次，每个批次的大小为 240 张图片，因此共 $240 \times 250 = 60000$ 个图片作为训练集，同时将 10000 张图片作为测试集，运行 5 次程序，记录每次的训练时间和测试时间。同时设置 `Cuda_CNN_ptr` 中的参数同上，并运行 5 次，记录每次的训练时间和测试时间，得到结果如下表 (6-3) 所示。

表 6-3: 训练时间和测试时间比较

次数	1	2	3	4	5
无 GPU 并行时训练时间 (ms)	27571.64	27292.08	27287.97	27314.55	27315.76
无 GPU 并行时测试时间 (ms)	1502.59	1504.26	1506.38	1505.95	1505.32
有 GPU 并行时训练时间 (ms)	267.58	267.69	267.08	267.79	267.28
有 GPU 并行时测试时间 (ms)	41.83	41.80	48.27	49.28	48.59

从表 (6-3) 可以看到，有 GPU 并行时的程序的运行效率大大提高，训练时间是无

GPU 并行的 1/100，测试时间是无 GPU 并行的 1/33。效率大大提高。

7 实验感想

在本次课程设计中，我学到了非常多的东西，主要有以下几点：

1. 详细了解学习了卷积神经网络的相关知识。
2. 学习到了卷积神经网络的组成，主要由卷积层、池化层，归一化层，全连接层组成。虽然在之前对此稍有了解，但本次的研究加深了对此的了解，为以后的工作打下了基础。
3. 对网络的参数进行了调整并观察比较结果，学习到了网络超参数对网络的性能有至关重要的影响。
4. 同时，学习到了一些调参的基本技巧。
5. 了解并学习了 CUDA 编程。
6. 体会到的 CUDA 编程在深度学习中的重要地位。
7. 除了关于深度学习的知识外，本次课程设计，加深了我对 C++ 语言的理解，并增强了我阅读代码和写代码的能力。

本次课程设计中，我收获颇丰，在未来的学习工作生活中，我会加倍努力，增强自身素养，包括但不限于程序编写能力，论文撰写能力，发现问题的能力，提出问题的能力，解决问题的能力，英语能力等各方面能力。希望在未来的学习科研工作生活中，平衡好各方面关系，达到自己的目标。