Project 1
Digit recognition with convolutional neural networks

项目 1
卷积神经网络数字识别

Due date: 23:59 1/17 (2023)

截止日期: 23:591/17(2023)

1. Instructions

1. 说明书

These instructions are also true to all the remaining projects unless indicated. Please read them carefully.

这些说明同样适用于所有剩余的项目，除非另有说明。请仔细阅读。

1. Students are encouraged to discuss projects. However, each student needs to write code and a report all by oneself. Code should NOT be shared or copied. Do NOT use external code unless permitted. Obviously, it is not OK to look at the code and just reimplement with very similar coding structure.

鼓励学生讨论项目。然而，每个学生都需要自己写代码和报告。代码不应该被共享或复制。除非允许，不要使用外部代码。显然，看看代码，然后用非常相似的代码结构重新实现是不合适的。

2. Post questions to Canvas so that everybody can share, unless the questions are private. Please look at Canvas first if similar questions have been posted. For private questions, please use Canvas:Inbox.

把问题发到 Canvas 上，这样每个人都可以分享，除非这些问题是私人的。如果类似的问题已经发布，请先看看 Canvas。对于私人问题，请使用 Canvas: Inbox。

3. When you intend to use free-late days, please specify/write at the top of the write-up (or right after the title) in the first page.

当您打算使用免费延迟日时，请在第一页的注释(或标题之后)的顶部注明/写入。

4. Two files need to be uploaded. First, your write-up (the main document) must be named as {Your-SFUID}.pdf and uploaded to Canvas. Second, a zip package must be uploaded to also Canvas with the following directory structure (they will be different for the other projects but will be similar):
需要上传两个文件。首先，你的报告(主文档)必须命名为{ Your-SFUID }。Pdf 并上传到 Canvas。第二，一个 zip 包必须上传到以下目录结构的 Canvas (它们对于其他项目是不同的，但是是相似的) :
    - {SFUID}/
    { SFUID }/
        - ec
        欧共体
            - ec.m
            Ecm
        - matlab/
        Matlab/
            - col2im_conv.m
            Col2im _ conv. m
            - col2im_conv_matlab.m
            Col2im _ conv _ matlab. m
            - conv_layer_backward.m
            层向后
            - conv_layer_forward.m
            Conv _ layer _ forward
            - conv_net.m
            (英译汉)
            - convnet_forward.m
            前进
            - get_lenet.m
            得到 _lenet. m
            - get_lr.m
            得到
            - im2col_conv.m
            第二季，第 10 集
            - im2col_conv_batch.m
            Im2col _ conv _ batch
            - init_convnet.m
            Init _ convnet. m
            - inner_product_backward.m
            内部产品 _ 反向
            - inner_product_forward.m

内部产品转发

- load_mnist.m

加载 mist

- mlrloss.m

失去

- pooling_layer_backward.m

池 _ 层 _ 向后

- pooling_layer_forward.m

池 _ 层 _ 前进

- relu_forward.m

Relu _ forward. m

- relu_backward.m

后退

- sgd_momentum.m

动量

- test_components.m

测试组件

- test_network.m

测试 _ 网络

- train_lenet.m

列车，列车

- vis_data.m

相对于数据

- lenet_pretrained.mat

预先训练好的垫子

- mnist all.mat

Mnist all.mat

- Project 1 has 13 pts.

Project 1 得了 13 分。

5. File paths: Make sure that any file paths that you use are relative and not absolute so that we can easily run code on our end. For instance, you cannot write “imread(‘/some/absolute/path/data/abc.jpg’)”. Write “imread(‘../data/abc.jpg’)” instead.

文件路径: 确保你使用的任何文件路径都是相对的，而不是绝对的，这样我们就可以很容易地在我们这边运行代码。例如，你不能写“ imread (‘/some/absolute/path/data/abc.jpg’)”。写入“ imread (‘。./data/abc. jpg’)”。

2. Overview

2. 概述

In this assignment you will implement a convolutional neural network (CNN). You will be building a numeric character recognition system trained on the MNIST dataset.
在这个作业中，你将实现一个卷积神经网络(CNN)。你们将建立一个数字字符识别系统，这个系统将在 MNIST 数据集上进行训练。

We begin with a brief description of the architecture and the functions. For more details, you can refer to online resources such as http://cs231n.stanford.edu. Note that the amount of coding in this assignment is a lot less than the other assignments. We will not provide detailed instructions, and one is expected to search online and/or reverse-engineer template code.
我们首先对体系结构和功能进行简要描述。更多细节，你可以参考在线资源，如 http://cs231n.stanford.edu。请注意，这个作业的编码量比其他作业要少得多。我们不会提供详细的说明，其中一个需要在线搜索和/或逆向工程模板代码。

A typical convolutional neural network has four different types of layers.
一个典型的卷积神经网络有四种不同类型的层。

Fully Connected Layer / Inner Product Layer (IP)

全连接层/内部产品层(IP)

The fully connected or the inner product layer is the simplest layer which makes up neural networks. Each neuron of the layer is connected to all the neurons of the previous layer (See Fig 1). Mathematically it is modeled by a matrix multiplication and the addition of a bias term. For a given input x the output of the fully connected layer is given by the following equation,
全连接或内部产品层是构成神经网络的最简单的层。该层的每个神经元都连接到前一层的所有神经元(见图 1)。在数学上，它通过矩阵乘法和偏差项的添加来建模。对于给定的输入 x，完全连通层的输出由以下方程给出,

$$f(x) = Wx + b$$
$$F(x) = wx + b$$

W, b are the weights and biases of the layer. W is a two dimensional matrix of m × n size where n is the dimensionality of the previous layer and m is the number of neurons in this layer. b is a vector with size m × 1.
W，b 是层的权重和偏差。W 是 m × n 大小的二维矩阵，其中 n 是前一层的维数，m 是该层的神经元个数。B 是大小为 m × 1 的向量。

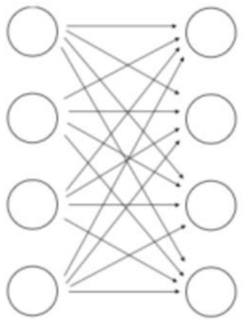


Figure 1: Fully connected layer

Convolutional Layer

卷积层

This is the fundamental building block of CNNs. Before we delve into what a convolution layer is, let's do a quick recap of convolution.
这是 cnn 的基本组成部分。在我们深入研究卷积层是什么之前，让我们快速回顾一下卷积。

As we saw in our lectures, convolution is performed using a k × k filter/kernel and a W × H image. The output of the convolution operation is a feature map. This feature map can bear different meanings according to the filters being used - for example, using a Gaussian filter will lead to a blurred version of the image. Using the Sobel filters in the x and y direction give us the corresponding edge maps as outputs.
正如我们在讲座中看到的，卷积是使用 k × k 滤波器/核和 w × h 图像进行的。卷积运算的输出是一个特征映射。这种特征映射可以根据所使用的滤波器具有不同的含义-例如，使用高斯滤波器将导致图像的模糊版本。在 x 和 y 方向使用 Sobel 滤波器可以得到相应的边缘映射作为输出。

Terminology : Each number in a filter will be referred to as a filter weight. For example, the 3x3 gaussian filter has the following 9 filter weights.
术语: 过滤器中的每个数字都被称为过滤器权重。例如，3x3 高斯滤波器有以下 9 个滤波器权重。

$$W = \begin{pmatrix} 0.0113 & 0.0838 & 0.0113 \\ 0.0838 & 0.6193 & 0.0838 \\ 0.0113 & 0.0838 & 0.0113 \end{pmatrix}$$

When we perform convolution, we decide the exact type of filter we want to use and accordingly decide the filter weights. CNNs try to learn these filter weights and biases from the data. We attempt to learn a set of filters for each convolutional layer.
当我们执行卷积时，我们决定我们要使用的滤波器的确切类型，并相应地决定滤波器的权重。Cnn 试图从数据中学习这些过滤器的权重和偏差。我们试图了解每个卷积层的一组过滤器。

In general there are two main motivations for using convolution layers instead of fully-connected (FC) layers (as used in neural networks).
一般来说，使用卷积层而不是完全连接层(在神经网络中使用)有两个主要的动机。

1. **A reduction in parameters**
   In FC layers, every neuron in a layer is connected to every neuron in the previous layer. This leads to a large number of parameters to be estimated - which leads to over-fitting. CNNs change that by sharing weights (the same filter is translated over the entire image).

参数的减少

   在 FC 层中，一层中的每个神经元都与前一层中的每个神经元相连。这导致了需要估计的大量参数——这导致了过度拟合。Cnn 通过共享权重改变了这种状况(整个图像都采用相同的过滤器)。

2. **It exploits spatial structure**
   Images have an inherent 2D spatial structure, which is lost when we unroll the image into a vector and feed it to a plain neural network. Convolution by its very nature is a 2D operation which operates on pixels which are spatially close.

它利用空间结构

   图像具有固有的二维空间结构，当我们将图像展开成一个向量并将其输入一个普通的神经网络时，这种结构就会丢失。卷积本质上是一种二维运算，它对空间上相近的像素进行操作。

Implementation details: The general convolution operation can be represented by the following equation:

实现细节: 一般卷积运算可以用以下公式表示:

$$f(X, W, b) = X * W + b$$

$$F(x，w，b) = x * w + b$$

where W is a filter of size $k \times k \times C_i$, X is an input volume of size $N_i \times N_i \times C_i$ and b is 1×1 element. The meanings of the individual terms are shown below.
其中 w 为 k × k × Ci 的滤波器，x 为 Ni × Ni × Ci 的输入体积，b 为 1 × 1 元素。个别术语的含义如下所示。

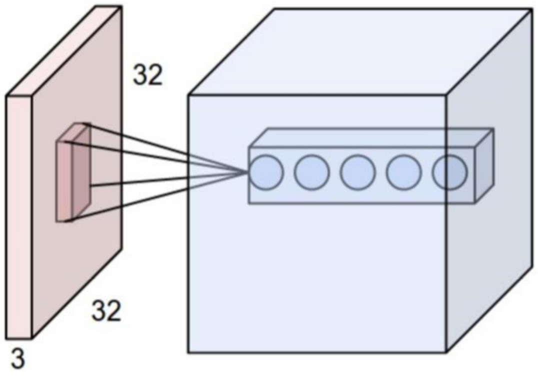


Figure 2: Input and output of a convolutional layer (Image source: Stanford CS231n

In the following example the subscript i refers to the input to the layer and the subscript o refers to the output of the layer.
在下面的例子中，下标 i 表示层的输入，下标 o 表示层的输出。

- N_i - width of the input image
  输入图像的镍宽
- N_i - height of the input image (image has a square shape)
  输入图像的高度(图像有一个正方形)
- C_i - number of channels in the input image
  输入图像中的通道数
- k_i - width of the filter
  滤波器的 ki-width
- s_i - stride of the convolution
  卷积的步伐
- p_i - number of padding pixels for the input image
  输入图像的填充像素数
- num - number of convolution filters to be learnt
  要学习的卷积滤波器的数目

A grayscale image has 1 channel, which is the depth of the image volume. For an image with C_i channels - we will learn num filters of size $k_i \times k_i \times C_i$. The output of convolving with each filter is a feature map with height and width N_o, where

一个灰度图像有一个通道，这是图像体积的深度。对于具有 Ci 通道的图像，我们将学习大小为 ki × ki × Ci 的 num 滤波器。每个滤波器卷积的输出是一个高度和宽度为 No 的特征映射，其中

$$N_o = \frac{N_i - k_i + 2p_i}{s_i} + 1$$

If we stack the num feature maps, we can treat the output of the convolution as another 3D volume/ image with C_o = num channels.
如果我们堆叠的数字特征映射，我们可以把卷积的输出作为另一个 3d 体积/图像与 Co = num 通道。

In summary, the input to the convolutional layer is a volume with dimensions $N_i \times N_i \times C_i$ and the output is a volume of size $N_o \times N_o \times$ num. Figure 2 shows a graphical picture.
综上所述，对卷积层的输入是一个尺寸为 Ni × Ni × Ci 的体积，输出是一个尺寸为 No × No × num 的体积。图 2 显示了一个图形图片。

Pooling layer

池层

A pooling layer is generally used after a convolutional layer to reduce the size of the feature maps. The pooling layer operates on each feature map separately and replaces a local region of the feature map with some aggregating statistics like max or average. In addition to reducing the size of the feature maps, it also makes the network invariant to small translations. This means that the output of the layer doesn't change when the object moves a little.
池层通常在卷积层之后使用，以减少特征映射的大小。池层分别对每个特征映射进行操作，并用一些聚合统计数据(如最大值或平均值)取代特征映射的局部区域。除了减少特征映射的大小，它也使网络不变的小翻译。这意味着当对象移动一点点时，图层的输出不会改变。

In this assignment we will use only a MAX pooling layer shown in figure 3. This operation is performed in the same fashion as a convolution, but instead of applying a filter, we find the max value in each kernel. Let k represent the kernel size, s represent the stride and p represent the padding. Then the output of a pooling function f applied to a padded feature map X is given by:
在这个作业中，我们将只使用如图 3 所示的 MAX 池层。这个操作以卷积的方式执行，但是我们没有应用过滤器，而是在每个内核中找到最大值。让 k 代表内核大小，s 代表步长，p 代表填充。然后，应用于填充特征映射 x 的池函数 f 的输出如下所示:

$$f(X,i,j) = \max_{x\in[i-k/2,i+k/2],y\in[j-k/2,j+k/2]}(X[x,y])$$

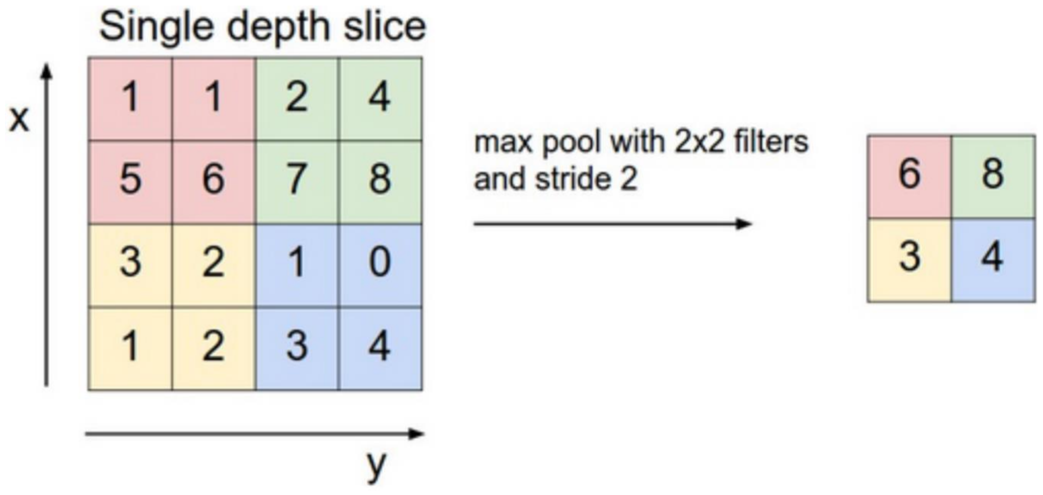


Figure 3: Example MAX pooling layer

Activation layer - ReLU - Rectified Linear Unit

激活层-ReLU 整流线性单位

Activation layers introduce the non-linearity in the network and give the power to learn complex functions. The most commonly used non-linear function is the ReLU function defined as follows,
激活层在网络中引入了非线性，赋予了学习复杂函数的能力。最常用的非线性函数是定义如下的 ReLU 函数,

f (x) = max(x, 0)
f (x) = max (x，0)

The ReLU function operates on each output of the previous layer.
ReLU 函数对前一层的每个输出进行操作。

Loss layer

损失层

The loss layer has a fully connected layer with the same number of neurons as the number of classes. And then to convert the output to a probability score, a softmax function is used. This operation is given by,
损失层有一个完全连接的层，其神经元的数量与类的数量相同。然后使用柔性最大激活函数将输出转换为概率分数。这个操作由,

$$p = \text{softmax}(W x + b)$$
$$p = \text{softmax} (w x + b)$$

where, W is of size C × n where n is the dimensionality of the previous layer and C is the number of classes in the problem.
其中 w 大小为 c × n 其中 n 是前一层的维数 c 是问题中的类数。

This layer also computes a loss function which is to be minimized in the training process. The most common loss functions used in practice are cross entropy and negative log-likelihood. In this assignment, we will just minimize the negative log probability of the given label.
这一层还计算了一个损失函数，这个损失函数将在训练过程中最小化。在实践中最常用的损失函数是交叉熵和负对数似然。在这个任务中，我们只是将给定标签的负对数概率最小化。

Architecture

建筑

In this assignment we will use a simple architecture based on a very popular network called the LeNet (http://ieeexplore.ieee.org/abstract/document/726791/)
在这个作业中，我们将使用一个基于一个非常流行的网络 LeNet (http://ieeexplore.ieee.org/abstract/document/726791/)的简单架构

- Input - 1×28×28
  输入-1 × 28 × 28
- Convolution - k = 5, s = 1, p = 0, 20 filters
  卷积 -k = 5，s = 1，p = 0,20 个滤波器
- ReLU

瑞鲁

- MAXPooling - k=2, s=2, p=0

MAXPooling-k = 2，s = 2，p = 0

- Convolution - k = 5, s = 1, p = 0, 50 filters

卷积 -k = 5，s = 1，p = 0,50 个滤波器

- ReLU

瑞鲁

- MAXPooling - k=2, s=2, p=0

MAXPooling-k = 2，s = 2，p = 0

- Fully Connected layer - 500 neurons

完全连接层-500 个神经元

- ReLU

瑞鲁

- Loss layer

损失层

Note that all types of deep networks use non-linear activation functions for their hidden layers. If we use a linear activation function, then the hidden layers has no effect on the final results, which would become the linear (affine) functions of the input values, which can be represented by a simple 2 layer neural network without hidden layers.
请注意，所有类型的深层网络都对其隐藏层使用非线性激活函数。如果我们使用线性激活函数，那么隐层对最终结果没有影响，这将成为输入值的线性(仿射)函数，它可以用一个简单的 2 层神经网络来表示，没有隐层。

There are a lot of standard Convolutional Neural Network architectures used in the literature, for instance, AlexNet, VGG-16, or GoogLeNet. They are different in the number of parameters and their configurations.
文献中使用了许多标准的卷积神经网络体系结构，例如 AlexNet、 vgg-16 或 GoogLeNet。它们在参数的数量和配置上是不同的。

3. Programming

3. 编程

Most of the basic framework to implement a CNN has been provided. You will need to fill in a few functions. Before going ahead into the implementations, you will need to understand the data structures used in the code.

大部分实现 CNN 的基本框架已经提供。你需要填写一些功能。在开始实现之前，你需要了解代码中使用的数据结构。

Data structures

数据结构

We define four main data structures to help us implement the Convolutional Neural Network which are explained in the following section. Each layer is defined by a data structure, where the field type determines the type of the layer. This field can take the values of DATA, CONV, POOLING, IP, RELU, LOSS which correspond to data, convolution, max-pooling layers, inner-product/ fully connected, ReLU and Loss layers respectively. The fields in each of the layer will depend on the type of layer.
我们定义了四种主要的数据结构来帮助我们实现卷积神经网络，这将在下面的部分进行解释。每一层都由一个数据结构定义，其中字段类型决定了层的类型。该域可以取 DATA、CONV、 POOLING、 IP、 RELU、 LOSS 值，它们分别对应于数据层、卷积层、最大池层、内积/完全连通层、 RELU 和 LOSS 层。每个层中的字段取决于层的类型。

The input is passed to each layer in a structure with the following fields.
将输入传递给具有以下字段的结构中的每一层。

- height - height of the feature maps
  高度-特征映射的高度
- width - width of the feature maps
  特征映射的宽度
- channel - number of channels / feature maps
  频道-频道数/特征映射
- batch size - batch size of the network. In this implementation, you will implement the mini-batch stochastic gradient descent to train the network. The idea behind this is very simple, instead of computing gradients and updating the parameters after each image, we do it after looking at a batch of images. This parameter batch size determines how many images it looks at once before updating the parameters.
  批量大小-网络批量大小。在这个实现中，你将实现小批量随机梯度下降来训练网络。这背后的想法非常简单，我们不需要计算梯度和更新每张图片后的参数，而是在查看一批图片后进行更新。这个参数批量大小决定了在更新参数之前一次看多少张图片。
- data - stores the actual data being passed between the layers. This is always supposed to be of the size [height × width × channel, batch size]. You can

resize this structure during computations, but make sure to revert it to a two-dimensional matrix. The data is stored in a column major order. The row comes next, and the channel comes the last.

Data-存储在图层之间传递的实际数据。这总是被认为是大小[高度 × 宽度 × 通道，批量大小]。你可以在计算过程中调整这个结构的大小，但是一定要把它还原成一个二维矩阵。数据存储在一个列的主要顺序。接下来是行，最后是通道。

- diff - Stores the gradients with respect to the data, it has the same size as data. Each layer's parameters are stored in a structure param. You do not touch this in the forward pass.

存储相对于数据的渐变，它具有与数据相同的大小。每一层的参数都存储在一个结构参数中。在前进过程中你不能碰这个。

- w - weight matrix of the layer

图层的 w 重量矩阵

- b - bias

B 偏见

param_grad is used to store the gradients coupled at each layer with the following properties: w - stores the gradient of the loss with respect to w.

参数梯度用于存储各层耦合的梯度，具有以下特性: w ——存储损失相对于 w 的梯度。

- b - stores the gradient of the loss with respect to the bias term.

存储损失相对于偏置项的梯度。

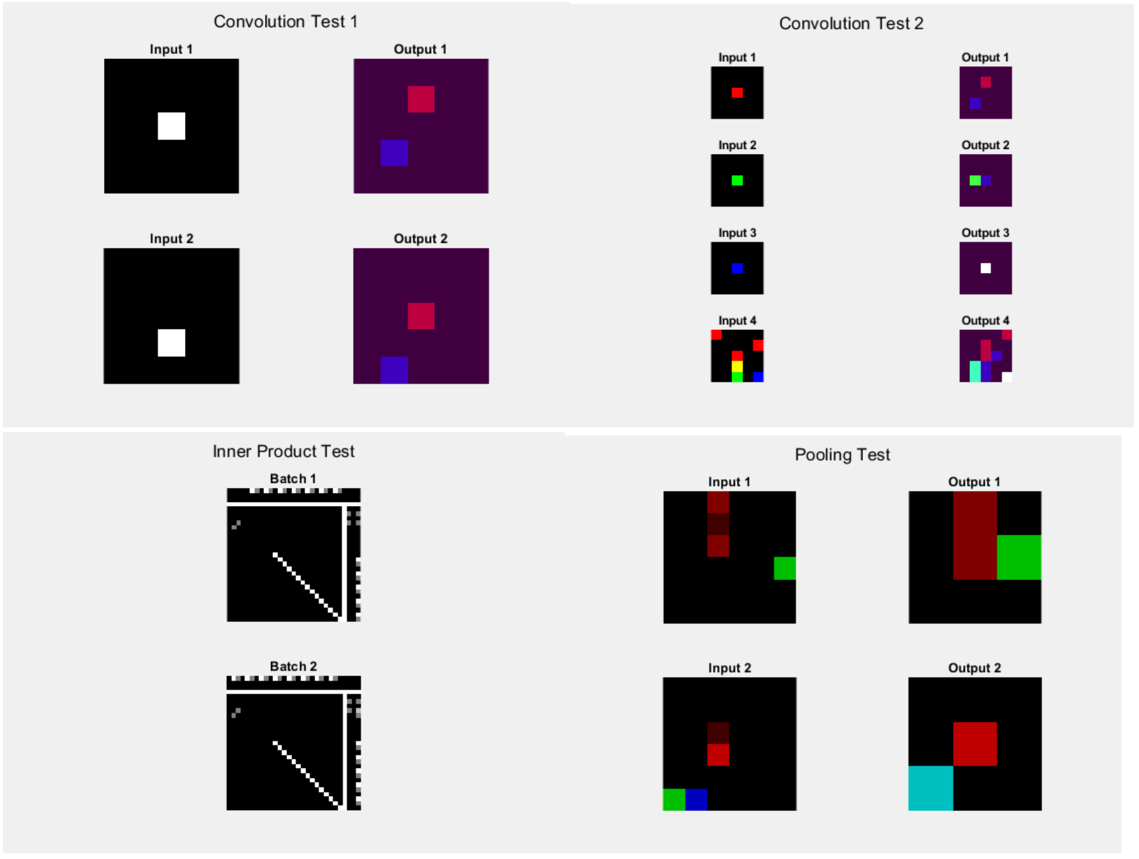
Part 1: Forward Pass

第 1 部分: 向前传球

Now we will start implementing the forward pass of the network. Each layer has a very similar prototype. Each layer's forward function takes input, layer, param as argument. The input stores the input data and information about its shape and size. The layer stores the specifications of the layer (e.g., for a conv layer, it will have k, s, p). The params is an optional argument passed to layers which have weights. This contains the weights and biases to be used to compute the output. In every forward pass function, you are expected to use the arguments and compute the output. You are supposed to fill in the height, width, channel, batch size, data fields of the output before returning from the function. Also make sure that the data field has been reshaped to a 2D matrix.
现在我们将开始实现网络的前进通道。每一层都有一个非常相似的原型。每一层的前向函数都以输入、层、参数为参数。输入存储输入数据和关于其形状和大小的信息。层存储层的规格(例如，对于一个 conv 层，它将有 k，s，p)。参数是一个可选的参数，传递给有

权重的层。它包含用于计算输出的权重和偏差。在每个前向传递函数中，你需要使用参数并计算输出。在从函数返回之前，你需要填写输出的高度，宽度，通道，批量大小，数据字段。还要确保数据字段已经被重塑为 2d 矩阵。

In the past, we asked to provide some visualization of results for every single step. However, there is no meaningful visualization until we implement the forward functions of all the layers. Once you implement all the layers, run test_components.m, then copy/paste the visualization results into the report. Those images should look like the following. (test_components.m was provided by courtesy of **Matthew Marinets** from the class of 2019 Fall at SFU). It is OK that the visualization is not exactly the same.
在过去，我们要求为每一个步骤提供一些可视化的结果。然而，直到我们实现了所有层的前向功能，才有了有意义的可视化。一旦你实现了所有的层，运行 test _ components。然后将可视化结果复制/粘贴到报告中。这些图像应该如下所示。(测试组件。M 由 SFU 2019 年秋季课程的 Matthew Marinets 提供)。可视化不完全相同是可以接受的。



Q 1.1 Inner Product Layer - 1 Pts

Q1.1 内部产品层 -1 pt

The inner product layer of the fully connected layer should be implemented with the following definition

完全连接层的内部产品层应该按照以下定义来实现

[output] = inner_product_forward(input, layer, param)
[output] = inner _ product _ forward (输入、层、参数)

Q 1.2 Pooling Layer - 1 Pts

Q 1.2 池层 -1 pt

Write a function which implements the pooling layer with the following definition.
使用以下定义编写一个实现池层的函数。

[output] = pooling_layer_forward(input, layer)
[output] = pool _ layer _ forward (input，layer)

input and output are the structures which have data and the layer structure has the parameters specific to the layer. This layer has the following fields,
输入和输出是具有数据的结构，层结构具有特定于层的参数。这一层有以下字段,

- pad - padding to be done to the input layer
  填充要做的输入层
- stride - stride of the layer
  图层的跨步
- k - size of the kernel (Assume square kernel)
  内核的 k 大小(假设是方形内核)

Q 1.3 Convolution Layer - 1 Pts

Q 1.3 卷积层 -1 pt

Implement a convolution layer with the following definition.
使用以下定义实现卷积层。

[output] = conv_layer_forward(input, layer, param)
[输出] = conv _ layer _ forward (input，layer，param)

The layer for a convolutional layer has the same fields as that of a pooling layer and param has the weights corresponding to the layer. Do not worry about a field “group”,

which is set to 1 in this assignment. A convolution layer has a field “num”, which is the number of kernels, which is equal to the number of output channels.
卷积层的层具有与池层相同的域，参数具有与该层对应的权重。不要担心字段“组”，在这个赋值中它被设置为 1。一个卷积层有一个字段“ num”，它是内核的数量，等于输出通道的数量。

Q 1.4 ReLU - 1 Pts

Q1.4 ReLU-1 pt

Implement the ReLU function with the following definition.
使用以下定义实现 ReLU 函数。

[output] = relu_forward(input)
[输出] = relu _ forward (input)

Part 2 Back propagation

第二部分反向传播

After implementing the forward propagation, we will implement the back propagation using the chain rule. Let us assume layer i computes a function f_i with parameters of w_i then final loss can be written as the following.
在实现向前传播之后，我们将使用链规则实现向后传播。让我们假设图层 i 计算一个参数为 wi 的函数 fi，那么最终损失可以写成如下。

$$l = f_i(w_i, f_{i-1}(w_{i-1}, \dots))$$

To update the parameters we need to compute the gradient of the loss w.r.t. to each of the parameters.
为了更新参数，我们需要计算每个参数的损失梯度。

$$\frac{\partial l}{\partial w_i} = \frac{\partial l}{\partial h_i}\frac{\partial h_i}{\partial w_i}$$

$$\frac{\partial l}{\partial h_{i-1}} = \frac{\partial l}{\partial h_i}\frac{\partial h_i}{\partial h_{i-1}}$$

where, $h_i = f_i(w_i, h_{i-1})$.
其中，hi = fi (wi，hi-1)。

Each layer's back propagation function takes input, output, layer, param as input and return param_grad and input_od. output.diff stores the $\frac{\partial l}{\partial h_i}$. You are to use this to compute $\frac{\partial l}{\partial w}$ and store it in param_grad.w and $\frac{\partial l}{\partial b}$ to be stored in param_grad.b. You are also expected to return $\frac{\partial l}{\partial h_{i-1}}$ in input_od, which is the gradient of the loss w.r.t the input layer.
各层的反向传播函数以输入、输出、层、参数为输入，返回参数梯度和输入 od。Diff 存储。你要用它来计算并存储在 param _ grad 中。并存储在 param _ grad。B.你也应该在输入 _od 中返回，它是输入层损失 w.r.t 的梯度。

Q 2.1 ReLU - 1 Pts

Q2.1 ReLU-1 pt

Implement the backward pass for the Relu layer in relu_backward.m file. This layer doesn't have any parameters, so you don't have to return the param_grad structure.
在 Relu _ backward 中为 Relu 层实现向后传递。M 文件。这一层没有任何参数，因此您不必返回 param _ grad 结构。

Q 2.2 Inner Product layer - 1 Pts
Q2.2 内部产品层 -1 pt
Implement the backward pass for the Inner product layer in inner_product_backward.m
在 Inner _ product _ backward. m 中实现内部产品层的向后通道

Putting the network together

整合网络

This part has been done for you and is available in the function convnet forward. This function takes the parameters, layers and input data and generates the outputs at each layer of the network. It also returns the probabilities of the image belonging to each class. You are encouraged to look into the code of this function to understand how the data is being passed to perform the forward pass.
这一部分已经为你完成，可以在函数 convnet forward 中使用。这个函数接受参数、层次和输入数据，并在网络的每一层生成输出。它还返回属于每个类的图像的概率。我们鼓励你查看这个函数的代码，以了解数据是如何被传递来执行前进传递的。

Part 3 Training

第三部分训练

The function conv_net puts both the forward and backward passes together and trains the network. This function has also been implemented.
函数 conv _ net 把向前和向后的传递放在一起，训练网络。这个函数也已经被实现了。

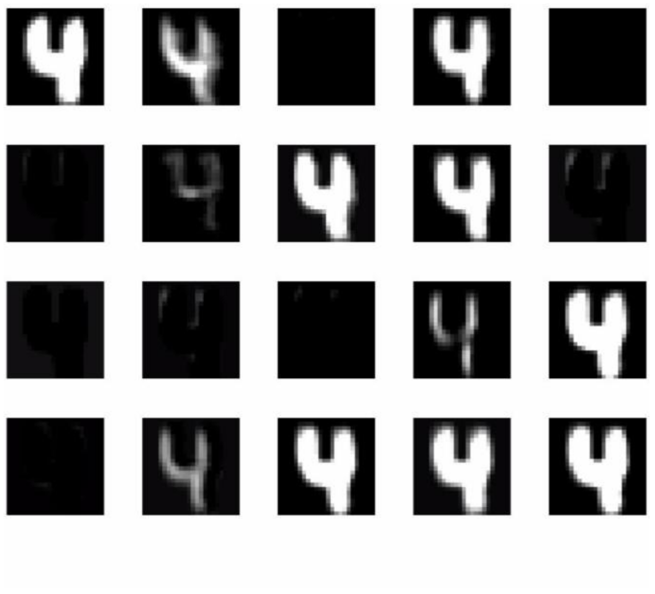


Figure 4: Feature maps of the second layer

Q 3.1 Training - 1 pts

问 3.1 训练 -1 分

The script train_lenet.m defines the optimization parameters and performs the actual updates on the network. This script loads a pretrained network and trains the network for 3000 iterations. Report the test accuracy obtained in your write-up after training for 3000 more iterations. Save the refined network weights as lenet.mat in the same format as lenet_pretrained.mat. The accuracy should be above 95%.
剧本火车 _ lenet。M 定义了优化参数，并在网络上执行实际的更新。这个脚本加载一个预先训练好的网络，并训练网络进行 3000 次迭代。在训练 3000 多次迭代之后，报告你在写作中获得的测试准确性。将精炼后的网络权重保存为 lenet.mat，格式与 lenet _ pretrained 相同。垫子。准确率应该在 95% 以上。

Q 3.2 Test the network - 1 Pts

Q3.2 测试网络 -1 pt

The script test_network.m has been provided which runs the test data through the network and obtains the prediction probabilities. Modify this script to generate the confusion matrix and comment on the top two confused pairs of classes (why they are confusing and etc.)
脚本测试 _ 网络。M 已经被提供，它通过网络运行测试数据并获得预测概率。修改此脚本以生成混淆矩阵，并注释前两对混淆的类(为什么它们是混淆的等等)

Q 3.3 Real-world testing - 1 Pts

Q3.3 真实世界测试 -1 个 pt

Obtain real-world digit examples. Show the results of your system on at least 5 examples, which you obtained yourself (e.g., downloading from Internet, scribble yourself, taking an image yourself. Do not use samples from Part 5 here.). For this step, please manually crop a bounding box containing each digit as opposed to Part 5, which requires you to find a digit automatically.
获取真实世界的数字例子。在至少 5 个你自己得到的例子上显示你的系统的结果(例如，从互联网上下载，自己涂鸦，自己拍照)。不要使用第 5 部分的例子).在这个步骤中，请手动裁剪一个包含每个数字的边框，而不是第 5 部分，第 5 部分要求你自动找到一个数字。

Part 4 Visualization

第四部分可视化

Q 4.1 - 1 Pts

问题 4.1-1 pt

Write a script vis_data.m which can load a sample image from the data, visualize the output of the second and third layers (i.e., CONV layer and ReLU layer). Show 20 images from each layer on a single figure file (use subplot and organize them in 4 × 5 format - like in Fig 4). To clarify, you take one image, run through your network, and visualize 20 features of that image at CONV layer and ReLU layer.
针对 _ data 编写脚本。M 可以从数据中加载一个样本图像，可视化第二层和第三层(即 CONV 层和 ReLU 层)的输出。在单个图形文件上显示来自每层的 20 个图像(使用子图并以 4 × 5 格式组织它们，如图 4 所示)。为了澄清，你取一张图片，在你的网络中运行，在 CONV 层和 ReLU 层可视化该图片的 20 个特征。

Q 4.2 - 1 Pts

Q 4.2-1 pt

Compare the feature maps to the original image and explain the differences.
将特征映射与原始图像进行比较，并解释差异。

Part 5 Image Classification - 2 Pts

第五部分图像分类 -2 个 pt

We will now try to use the fully trained network to perform the task of Optical Character Recognition. You are provided a set of real world images in the images folder. Write a script ec.m which will read these images and recognize the handwritten numbers.
我们现在将尝试使用训练有素的网络来完成光学字符识别的任务。图像文件夹中提供了一组真实世界的图像。写一个脚本 ec.m 来读取这些图片并识别手写的数字。

The network you trained requires a grey-scale image with a single digit in each image. There are many ways to obtain this given a real image. Here is an outline of a possible approach:
你训练的网络需要一个灰度图像，每个图像中只有一个数字。给定一个真实的图像，有很多方法可以得到这个结果。下面是一个可能方法的大纲:

1. Classify each pixel as foreground or background pixel by performing simple operations like thresholding.
   通过执行像阈值这样的简单操作，将每个像素分类为前景像素或背景像素。
2. Find connected components and place a bounding box around each character. You can use a matlab built-in function to do this.
   找到连接的组件，在每个字符周围放置一个边界框。你可以使用 matlab 的内置函数来做到这一点。
3. Take each bounding box, pad it if necessary and resize it to 28×28 and pass it through the network.
   取出每个包围盒，必要时将其填充并调整为 28 × 28 的大小，然后通过网络传递。

There might be errors in the recognition, report the output of your network in the report. For this part, you are allowed to use *graythresh, adaptthresh, bwconncomp, bwlabel, and regionprops* built-in functions.
识别中可能有错误，在报告中报告网络的输出。对于这一部分，你可以使用 graythresh，adaptthresh，bwconncomp，bwlabel 和 regionprops 内置函数。

Appendix: List of all files in the project

附录: 项目中所有文件的列表

- col2im_conv.m Helper function, you can use this if needed col2im_conv_matlab.m Helper function, you can use this if needed

如果需要，可以使用 col2im _ conv _ matlab. m Helper 函数，如果需要，可以使用它

- conv_layer_backward.m - Do not modify

不要修改

- conv_layer forward.m - To implement

Conv _ layer forward.m-实现

- conv_net.m - Do not modify

不要修改

- convnet_forward.m - Do not modify

Convnet _ forward. m-不要修改

- get_lenet.m - Do not modify. Has the architecture.

不要修改。有架构。

- get_lr.m - Gets the learning rate at each iterations im2col_conv.m Helper function, you can use this if needed im2col_conv_batch.m Helper function, you can use this if needed

Get _ lr.获得每次迭代的学习速率。M Helper 函数，如果需要，你可以使用它 im2col _ conv _ batch。M Helper 函数，你可以在需要的时候使用它

- init_convnet.m Initialise the network weights

初始化网络权重

- inner_product_backward.m - To implement

内部产品 _ 反向

- inner_product_forward.m - To implement

内部产品的实施

- load_mnist.m - Loads the training data.

加载培训数据。

- mlrloss.m - Implements the loss layer

M 实现损失层

- pooling_layer_backward.m Implemented, do not modify

池 _ 层 _ 向后。 m 执行，不修改

- pooling_layer_forward.m - To implement

池 _ 层 _ 前进。 m-实现

- relu_backward.m - To implement

  Relu _ backward. m-实现
- relu_forward.m - To implement

  执行
- sgd_momentum.m - Do not modify. Has the update equations

  不要修改。有更新的方程式
- test_network.m - Test script

  网络测试脚本
- train_lenet.m - Train script

  Train _ lenet. m-Train 脚本
- vis_data.m - Add code to visualise the filters

  M-添加可视化过滤器的代码
- lenet_pretrained.mat - Trained weights

  Lenet _ pretrained。 mat-Trained 砝码
- mnist_all.mat - Dataset

  Mist _ all. mat-Dataset

Notes

注释

Here are some points which you should keep in mind while implementing:
以下是一些你在执行时应该记住的要点:

- All the equations above describe the functioning of the layers on a single data point. Your implementation would have to work on a small set of inputs called a ”batch” at once.

  上面的所有方程都描述了单个数据点上层的功能。你的实现必须同时处理一组叫做“批处理”的输入。
- Always ensure that the output.data of each layer has been reshaped to a 2-D matrix.

  始终确保每一层的输出.data 都被重塑为一个二维矩阵。