代码参考了前人的工作，地址 https://download.csdn.net/download/u012737234/8493615 以及 https://blog.csdn.net/weixin_39566131/article/details/100578284，点击 trainLenet.m 即可运行，在此致谢。train-images.idx3-ubyte 超 25MB，上传不了，请见谅。

实验结论：LeNet-5 是一种用于手写体字符识别的非常高效的卷积神经网络，准确率达 99.1%，越到后期耗时越长；卷积层的参数较少，这是由卷积层的局部连接和共享权重所决定的。LeNe 的训练是一个循环过程：前向传播训练—>反向传播调参—>更新参数—>再次训练，直至训练结束就可拿去测试。其结构为输入的二维图像，先经过两次卷积层到池化层，再经过全连接层，最后使用 softmax 分类作为输出层。

LeNet-5 的基本结构

层次	层名	作用	实现前向训练过程
0	输入层	输入样本，大小为28*28	每次训练都会输入40张28x28的样本图片，构成28x28x40的3维矩阵
1	C1，卷积层	6种卷积核，size5*5，随机生成6x5x5个weight，6个bias	首先，28x28x40的输入样本与6组5x5卷积核卷积，得到结果为6组24x24x40的矩阵。然后，给6组24x24x40矩阵的每一个组种的每个点加上相应的6组bias，送入Sigmoid函数激活，此时得到结果为6组24x24x40矩阵。
2	S2，池化层	6种6x1组weight，6种bias	把上层得到的24x24x40矩阵做平均池化，即卷积上[0.25 0.25; 0.25 0.25]，步长为2，得到6组12x12x40的矩阵。
3	C3，卷积层	卷积核有16种，size为5x5，对上面6种卷积结果分别用上本层16种卷积核卷积，所以初始化随机生成16x6x5x5	S2层输出的6组12x12x40矩阵与C3层的6组5x5卷积核，组对应卷积后，累加6组，得到一组8x8x40的矩阵，给6组8x8x40矩阵的每一个点加上第一组bias，送入Sigmoid函数激活；换第二种6组5x5的卷积核继续和S2层输出的6组12x12x40矩阵组对应卷积后，累加6组，得到第二组8x8x40的矩阵，给6组8x8x40矩阵的每一个点加上第二组bias，送入Sigmoid函数激活；如此循环16次，得到16组8x8x40矩阵。
4	S4，池化层	随机生成16种6x1组weight，16种bias	把上层得到的8x8x40矩阵做平均池化，即卷积上[0.25 0.25; 0.25 0.25]，步长为2，得到16组4x4x40的矩阵。
5	C5，卷积层	随机生成120x256个weight，120个bias	把S4层输出的16组4x4x40的矩阵展开为256x40的矩阵，用C5层120x256的weight卷积256x40的展开矩阵得到120x40的矩阵，给矩阵的每一行加上不同的bias，共120行，之后送入Sigmoid函数激活。
6	F6，全连接层	随机生成84x120个weight，84个bias	F6层84x120的weight卷积C5层输出的120x40矩阵，得到84x40的矩阵，给矩阵的每一行加上不同的bias，共84行，之后送入Sigmoid函数激活。
7	Soft,输出层	全连接层，初始化随机生成10x84个weight	Soft层10x84的weight卷积F6层输出的84x40矩阵，得到10x40的矩阵，给矩阵的每一行加上不同的bias，共10组；找该矩阵的每一列最大值，用原10x40矩阵的每一列，减去该列的最大值；对10x40矩阵的每一个数值求e指数，即送入exp函数，其结果除以10x40矩阵的行求和（1x40），得到10x40的结果。

核心代码阐释 trainLenet.m：

```
clear;clc;

%%
================================================================
====
%load data
imageDim = 28;
numclasses = 10;
images = loadMNISTImages('train-images.idx3-ubyte');
images = reshape(images,imageDim,imageDim,[]);
labels = loadMNISTLabels('train-labels.idx1-ubyte');
labels(labels == 0) = 10;
%%输入层：输入训练样本，为了适应数据集把大小为 28x28

%% initialparameters
```

```
lenet.layers = {
    struct('type','i')
    struct('type','C1','outputmaps',6,'kernelsize',5) %卷积层，卷积核有 6 种，size 为 5x5，初始化随机生成 6x5x5 个 weight，6 个 bias，等待学习
    struct('type','S2','scale',2) %代码里池化层，初始化随机生成 6 种 6x1 组 weight，6 种 bias
    struct('type','C3','outputmaps',16,'kernelsize',5)% ：卷积层，卷积核有 16 种，size 为 5x5，对上面 6 种卷积结果分别用上本层 16 种卷积核卷积，所以初始化随机生成 16x6x5x5 个 weight
    struct('type','S4','scale',2) %池化层，代码里初始化随机生成 16 种 6x1 组 weight，16 种 bias
    struct('type','C5','outputmaps',120) %卷积层，初始化随机生成 120x256 个 weight，120 个 bias，等待学习
    struct('type','F6','outputmaps',84) %全连接层，初始化随机生成 84x120 个 weight，84 个 bias，等待学习
    struct('type','Soft','num_classes',10) %输出层，全连接层，初始化随机生成 10x84 个 weight，10 个 bias，等待学习
};

lenet = initialParameters(lenet,images);

%% train the cnn
opts.alpha = 1;

opts.batchsize = 40; %代码中每次训练都会输入 40 张 28x28 的样本图片，构成 28x28x40 的 3 维矩阵

opts.numepochs = 20;
lenet = cnnTrain(lenet,images,labels,opts);%cnnTrain 源代码涉及到 cnnff 和 cnnbp，cnnff 是前向传播训练，cnnbp 即反向传播调参——delta 反向传播就是从最底层，即 Soft 层，向上层传参。

%% test the cnn
testImages = loadMNISTImages('t10k-images.idx3-ubyte');
testImages = reshape(testImages,imageDim,imageDim,[]);
testLabels = loadMNISTLabels('t10k-labels.idx1-ubyte');
testLabels(testLabels == 0) = 10;

lenet = cnnff(lenet,testImages);

a = lenet.layers{8}.a;
[~,preds] = max(lenet.layers{8}.a,[],1);
preds = preds';

acc = sum(preds == testLabels) / length(preds);
```

```
fprintf('Accuracy is %f\n',acc);
```

程序结果：

```
epoch1/20
历时 48.254550 秒。
epoch2/20
历时 49.346237 秒。
epoch3/20
历时 52.720203 秒。
epoch4/20
历时 55.095996 秒。
epoch5/20
历时 58.191246 秒。
epoch6/20
历时 60.588026 秒。
epoch7/20
历时 65.960499 秒。
epoch8/20
历时 74.379831 秒。
epoch9/20
历时 84.641069 秒。
epoch10/20
历时 99.915916 秒。
epoch11/20
历时 122.869474 秒。
epoch12/20
历时 150.449686 秒。
epoch13/20
历时 181.499596 秒。
epoch14/20
历时 205.628320 秒。
epoch15/20
历时 232.310099 秒。
epoch16/20
历时 260.782082 秒。
epoch17/20
历时 289.569491 秒。
epoch18/20
历时 299.611985 秒。
epoch19/20
历时 320.018952 秒。
epoch20/20
历时 346.295471 秒。
Accuracy is 0.991000
```