机器学习大作业：Mnist实验

胡若凡 3200102312

1. 实验简介

1.1 数据集简介

MNIST 数据集来自美国国家标准与技术研究所, National Institute of Standards and Technology (NIST). 训练集 (training set) 由来自 250 个不同人手写的数字构成, 其中 50% 是高中学生, 50% 来自人口普查局 (the Census Bureau) 的工作人员. 测试集(test set) 也是同样比例的手写数字数据。通过此数据集训练，可以达到识别手写数据识别的效果。

1.2 实验任务

本次实验中，主要任务是使用任意一种的方法，对Mnist数据进行机器学习实验。特别的，本次实验的重点在于对实验结果进行数据可视化，通过更改不同的参数等，进行消融实验，并通过课堂学习的知识原理对所出现的结果进行合理解释。

2. 实验方法及环境

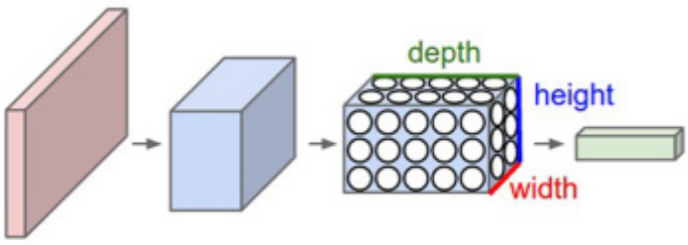
2.1 实验方法

本次实验使用的是CNN方法。下面对此方法进行概括简介。

卷积神经网络（ Convolutional Neural Network, CNN ）是一类包含卷积计算且具有深度结构的前馈神经网络，是深度学习的代表算法之一。它与普通神经网络非常相似，他们都由具有可学习的权重和偏置常量的神经元组成。每个神经元都接受一些输入，并做一些点积运算，输出的是每个分类的分数。整个网络依旧是一个可导的评分函数：该函数的输入是原始的图像像素，输出是不同类别的评分。

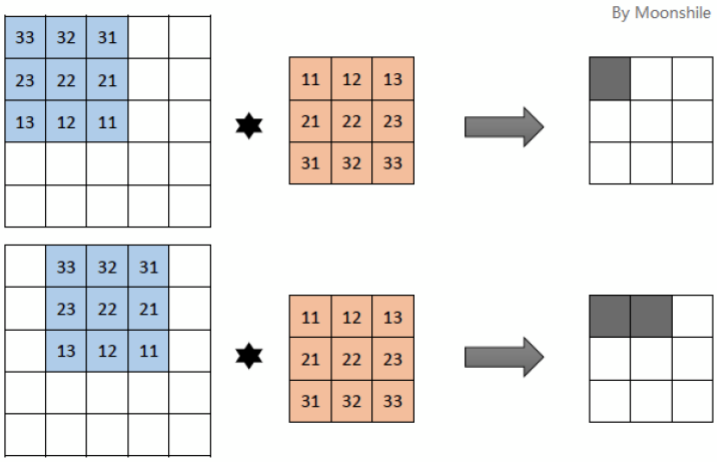
然而，由于卷积神经网络默认输入为图像，可以将特定性质的编码输入网络结构，能够让前馈函数更有效率，并大量减少参数。

CNN 具有三维体积的神经元，利用了输入图片的特点，将神经元设计成了 width，height，depth 三个维度。



卷积层（Convolutional layer）

卷积神经网路中每层卷积层由若干卷积单元组成，每个卷积单元的参数都是通过反向传播算法优化得到的。卷积层的参数是有一些可学习的滤波器集合构成的。每个滤波器在空间上（宽度和高度）都比较小，但是深度和输入数据一致.卷积运算的目的是提取输入的不同特征，第一层卷积层可能只能提取一些低级的特征如边缘、线条和角等层级，更多层的网络能从低级特征中迭代提取更复杂的特征。



- 输入数据体的尺寸为

$$W1 * M1 * D1$$

- 4个超参数：
  - 滤波器的数量K
  - 滤波器的空间尺寸F
  - 步长S
  - 零填充数量P
- 输出数据体的尺寸为

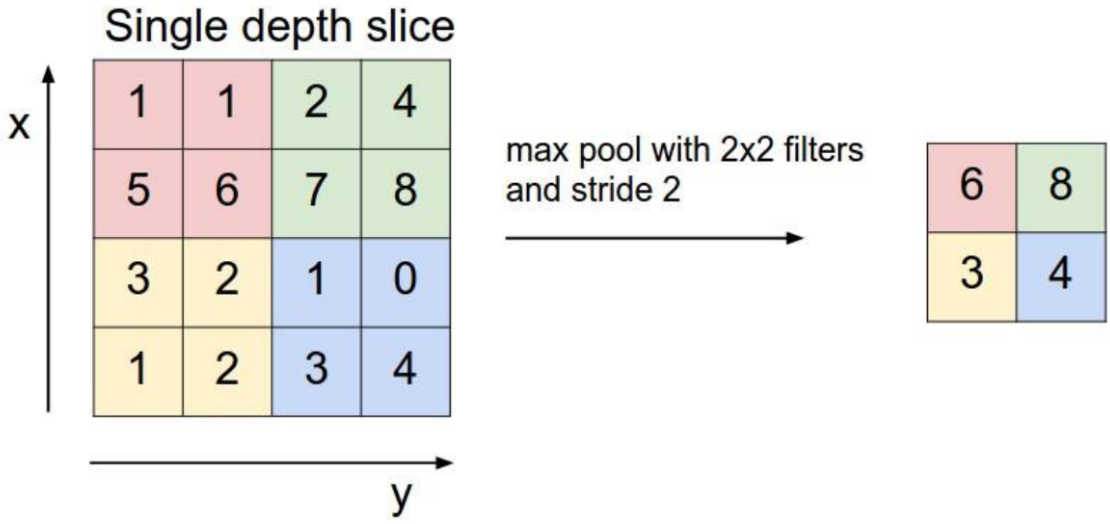
$$W_2 * H_2 * D_2$$
$$W_2 = (W_1 - F + 2P)/S + 1$$
$$H_2 = (H_1 - F + 2P)/S + 1$$
$$D_2 = K$$

线性整流层（Rectified Linear Units layer, ReLU layer）

这一层神经的活性化函数（Activation function）使用线性整流 Rectified Linear Units,ReLU）。

池化层（Pooling layer）

池化层的引入是仿照人的视觉系统对视觉输入对象进行降维和抽象。通常在卷积层之后会得到维度很大的特征，将特征切成几个区域，取其最大值或平均值，得到新的、维度较小的特征。这一层的特点是特征不变性、特征降维、一定程度上防止过拟合。



- 输入数据体尺寸

$$W1.H1.D1$$

- 有两个超参数：

    - 空间大小F
    - 步长S
- 输出数据体尺寸

$$W_2 * H_2 * D_2$$
$$W_2 = (W_1 - F)/S + 1$$
$$H_2 = (H_1 - F)/S + 1$$
$$D2 = D1$$

全连接层（Fully Connected layer）
把所有局部特征结合变成全局特征，用来计算最后每一类的得分。通过这些层一定顺序的组合，能够构建具有强图形处理能力的卷积神经网络。

2.2 实验环境

平台：windows2010

软件：Anaconda下的Tensorflow的Spyder

使用语言：Python

3. 实验代码

3.1 消融实验

Robert Long对消融研究（或消融实验）定义：通常用于神经网络，尤其是相对复杂的神经网络，如R-CNN。我们的想法是通过删除部分网络并研究网络的性能来了解网络，看哪个模块可以提升其性能。

这里举一个网上的例子而言：

Girshick及其同事描述了一个由三个“模块”组成的物体检测系统：第一个使用选择性搜索算法提出图像区域，在该区域内搜索物体。进入一个大的卷积神经网络（有5个卷积层和2个完全连接的层），进行特征提取，然后进入一组支持向量机进行分类。为了更好地理解该系统，作者进行了一项消融研究，其中系统的不同部分被移除 - 例如，移除CNN的一个或两个完全连接的层导致性能损失惊人地少。

这可以得出，CNN的大部分代表性力量来自其卷积层，而不是来自更大的密集连接层。

因此，进行消融实验时，例如同时加入ABC模块发现有较好效果，可以保留AB、AC、BC模块，观察性能是否下降或保持不变，以此来对比是否此模块改变对性能有较大的影响。

3.2 实验源代码

以下代码主要实现识别手写数字集的最基础的卷积神经网络（输入后经过 1 个卷积层、1 个池化层，扁平化后经过 2 个全连接层输出，全程使用 ReLU 作为激活函数），经过10次迭代后，使用 K Fold 的 5 倍交叉验证，并对 10 次迭代的交叉信息熵损失（Cross Entropy Loss）与分类准确率 Cla ssification A ccuracy）进行可视化，返回 5 次交叉测试准确率信息并形成箱线图。

```
from numpy import mean
from numpy import std
from matplotlib import pyplot as plt
from sklearn.model_selection import KFold
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Dense
```

```
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.optimizers import SGD

# 创建加载数据并划分训练集、测试集的函数 load_dataset()
def load_dataset():
    # 加载数据
    (trainX, trainY), (testX, testY) = mnist.load_data() # 重塑数据集以实现单一通道
    trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
    testX = testX.reshape((testX.shape[0], 28, 28, 1)) # 对目标值进行 one_hot 编码
    trainY = to_categorical(trainY)
    testY = to_categorical(testY)
    return trainX, trainY, testX, testY

# 归一化图像
def prep_pixels(train, test):
    # 将训练集灰度值从 int 调整为 float train_norm = train.astype('float32')
    train_norm = train.astype('float32')
    test_norm = test.astype('float32')
    # 将灰度值归一化到 0-1 之间train_norm = train_norm / 255.0 test_norm =
test_norm / 255.0
    train_norm = train_norm / 255.0
    test_norm = test_norm / 255.0
    # 返回归一化后的图像集
    return train_norm, test_norm


# 定义基础 CNN 模型
def define_model(): #全程使用 ReLU 激活函数
    model = Sequential()
    model.add(Conv2D(32, (5, 5), activation='relu',
    kernel_initializer='he_uniform', input_shape=(28, 28, 1))) #使用 32 个过滤器、
3*3 卷积核
    model.add(MaxPooling2D((2, 2))) #2*2Pooling
    model.add(Flatten()) #扁平化
    model.add(Dense(1024, activation='relu',
    kernel_initializer='he_uniform')) #第一个全连接层
    model.add(Dense(10, activation='softmax')) #softmax 归一# 编写模型
    opt = SGD(lr=0.01, momentum=0.9)
    #保守的随机梯度下降器配置，学习速度 0.01，动量 0.9
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=
['accuracy'])
    return model

# 使用 K-Fold 交叉验证评估模型
def evaluate_model(dataX, dataY, n_folds=5):
    scores, histories = list(), list()
    kfold = KFold(n_folds, shuffle=True, random_state=1) # 列出划分
    for train_ix, test_ix in kfold.split(dataX): # 定义模型
        model = define_model()
        # 从训练集与测试集中选取记录
        trainX, trainY, testX, testY = dataX[train_ix], dataY[train_ix],
dataX[test_ix], dataY[test_ix]
        # fit
        history = model.fit(trainX, trainY, epochs=10, batch_size=32,
validation_data=(testX, testY), verbose=0)
        # 评估模型
        _, acc = model.evaluate(testX, testY, verbose=0)
```

```
        print('> %.3f' % (acc * 100.0))
        # 将结果记录
        scores.append(acc)
        histories.append(history)
    return scores, histories

# 对 5 次交叉验证的 Cross Entropy Loss 与 Classification Accuracy 进行可视化
def summarize_diagnostics(histories):
    for i in range(len(histories)):
        # Loss
        plt.subplot(2, 1, 1)
        plt.title('Cross Entropy Loss')
        plt.plot(histories[i].history['loss'], color='blue',label='train')
        plt.plot(histories[i].history['val_loss'], color='orange', label='test')
        # Accuracy
        plt.subplot(2, 1, 2)
        plt.title('Classification Accuracy')
        plt.plot(histories[i].history['acc'], color='blue',label='train')
        plt.plot(histories[i].history['val_acc'], color='orange', label='test')
    plt.show()

# 结果统计
def summarize_performance(scores): # 评估结果
    print('Accuracy: mean=%.3f std=%.3f, n=%d' % (mean(scores)*100,
std(scores)*100, len(scores)))
    # 箱线图
    plt.boxplot(scores)
    plt.show()

# 运行程序
def run_test_harness(): # 加载数据集
    trainX, trainY, testX, testY = load_dataset() # 预处理图像数据
    trainX, testX = prep_pixels(trainX, testX) # 评估模型
    scores, histories = evaluate_model(trainX, trainY)
    # 学习曲线
    summarize_diagnostics(histories)
    # 估计结果总结
    summarize_performance(scores)

# 运行
run_test_harness()
```

4. 消融实验及结果解释

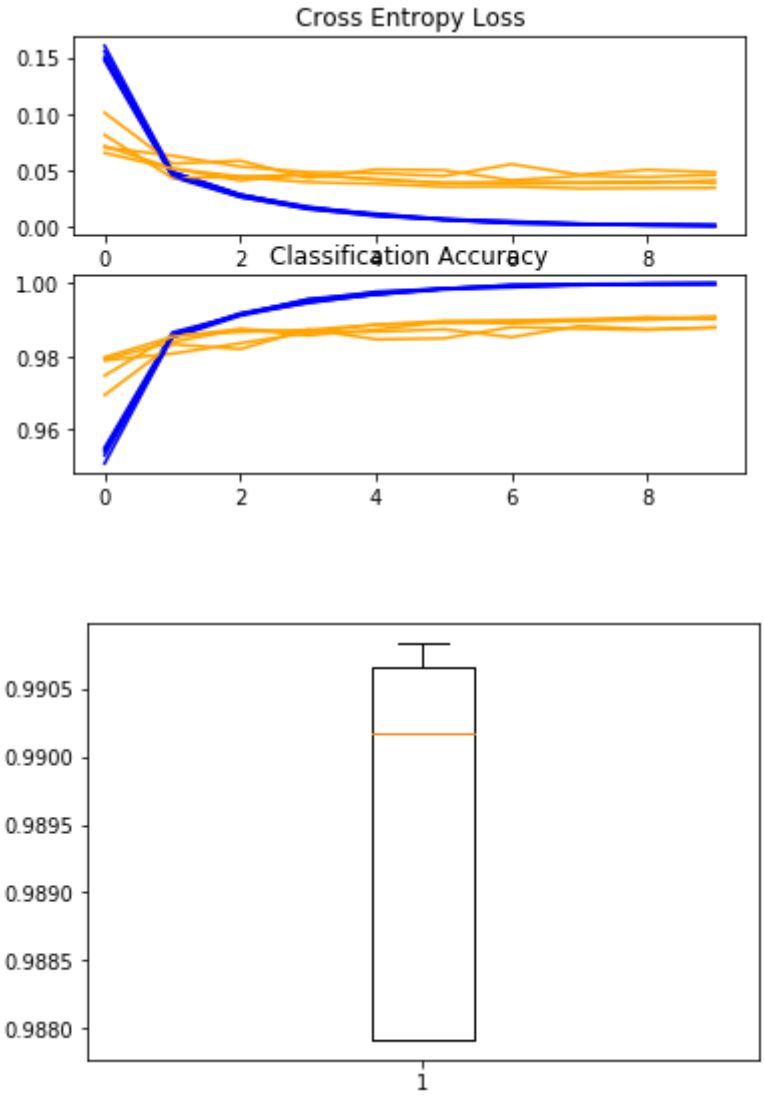
4.1 基础代码

由于上述代码的训练集与测试集的并集构成了整个数据集，且迭代次数为 10 次，出于时间效率考虑，可将单次运行的结果作为较为普遍的结果。

运行上述代码，编译器输出如下

Accuracy: mean=98.940 std=0.131, n=5

```
98.792
99.067
98.792
99.083
99.017
```



根据返回的图像可以看到，随着训练次数从 0 增加到 10 ，预测的信息熵损失逐接近 0 ，预测的准确率逐渐向 1 靠拢。由于学习过程面向整个 MINST 集，模型在五次交叉测试中的结果均较好，准确率高达 0 .9894 ，标准差仅 0.131 ，非常稳定。

4.2 模型深度消融实验

4.2.1 卷积层与池化层

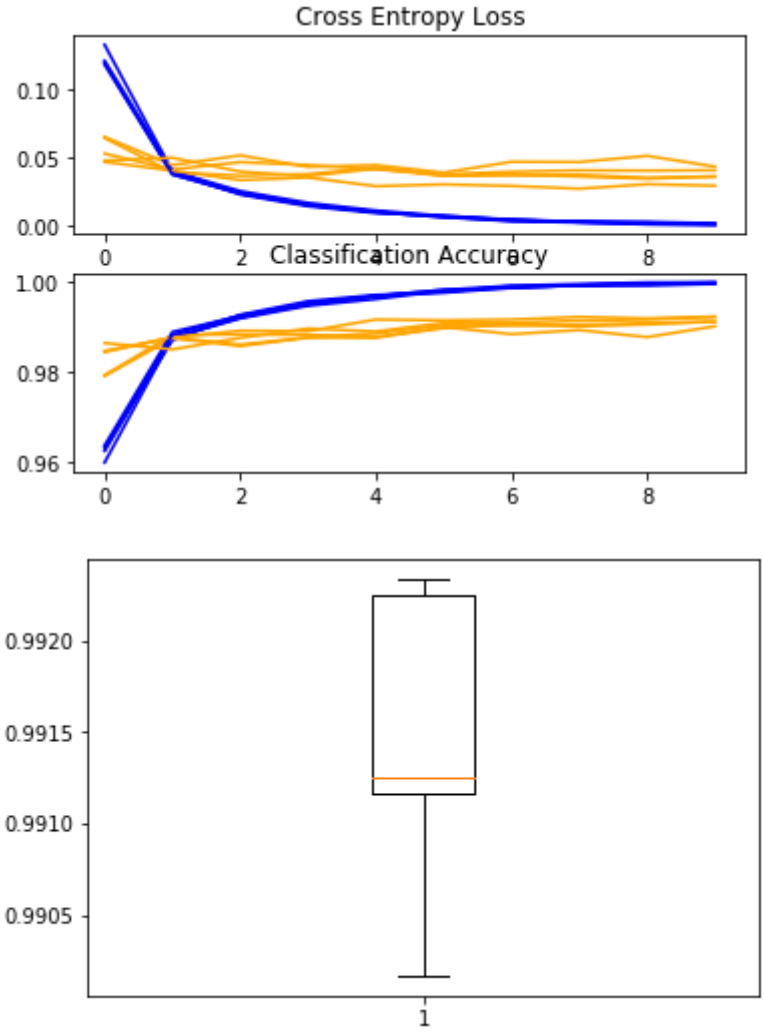
上述仅为实现识别 MINST 的基本 CNN 网络。除了使用的激励函数为 ReLU 而非较早的 sigmoid 之外，该神经网络的深度甚至小于了LeNet 。为了探究 CNN 模型深度是否会对预测准确率的影响，在原有的 MaxPooling 层后在添加一层卷积层和一层池化层，新的模型代码如下：

```
def define_model(): #全程使用 ReLU 激活函数
    model = Sequential()
    model.add(Conv2D(32, (5, 5), activation='relu',
    kernel_initializer='he_uniform', input_shape=(28, 28, 1))) #使用 32 个过滤器、
3*3 卷积核
    model.add(MaxPooling2D((2, 2))) #2*2Pooling
```

```
    model.add(Conv2D(64, (5, 5), activation='relu',
    kernel_initializer='he_uniform'))
    model.add(MaxPooling2D((2, 2))) #2*2Pooling
    model.add(Flatten()) #扁平化
    model.add(Dense(1024, activation='relu',
    kernel_initializer='he_uniform')) #第一个全连接层
    model.add(Dense(10, activation='softmax')) #softmax 归一# 编写模型
    opt = SGD(lr=0.01, momentum=0.9)
    #保守的随机梯度下降器配置，学习速度 0.01，动量 0.9
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=
['accuracy'])
    return model
```

```
> 99.117
> 99.125
> 99.017
> 99.233
> 99.225
```

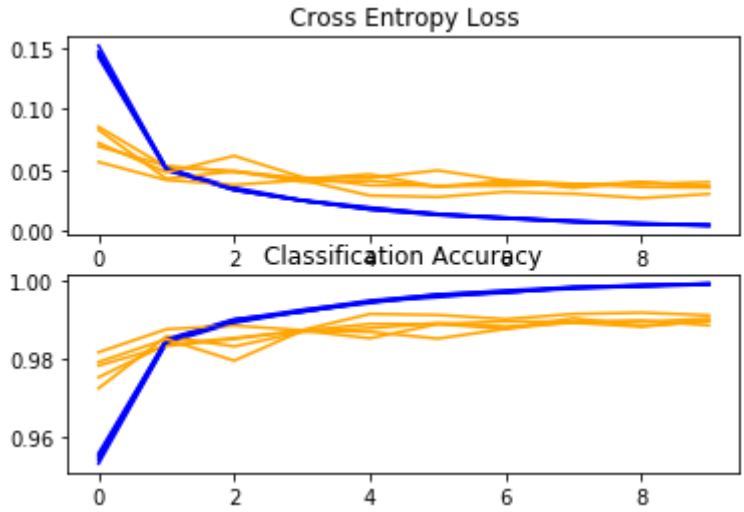
Accuracy: mean=99.143 std=0.080, n=5



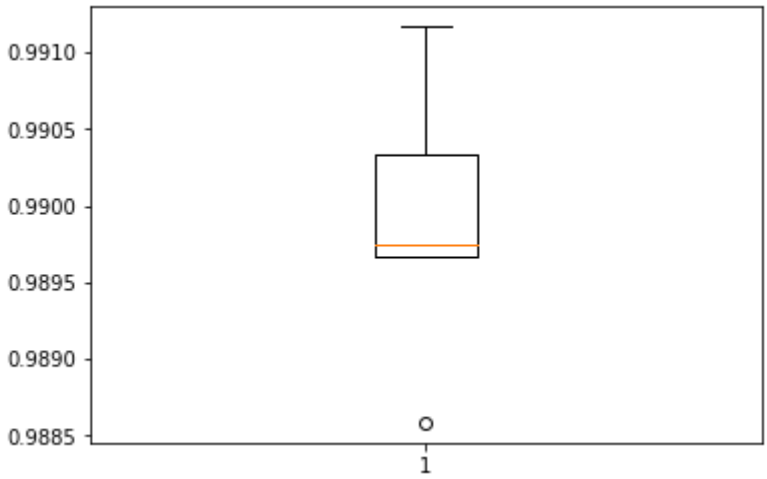
可以看到，对MNIST而言，在增加了层数之后，准确率依旧有将近0.2%的提高。这里可以认为是原有效果已达较高标准，因此准确率的提升并不是非常大，但其积极作用依旧是明显的。

如果只加入一个池化层，即消除对卷积层的加入，效果如下：

```
> 98.858
> 98.967
> 98.975
> 99.117
> 99.033
```



Accuracy: mean=98.990 std=0.085, n=5

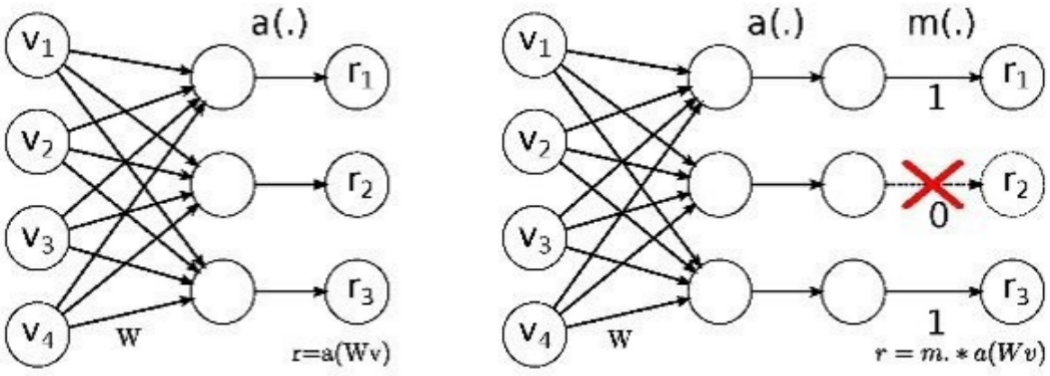


可以发现，单纯加入一个池化层，也有0.4%左右的准确率增加，也是有效的，说明池化层的增加对准确率产生了积极的影响。

4.2.2 Dropout层

如图左，为没有Dropout的普通2层全连接结构，记为 r=a(Wv)，其中a为激活函数。

如图右，为在第2层全连接后添加Dropout层的示意图。即在模型训练时随机让网络的某些节点不工作输出置0，其它过程不变。



对于Dropout这样的操作，强迫一个神经单元，和随机挑选出来的其他神经单元共同工作，达到好的效果。消除减弱了神经元节点间的联合适应性，增强了泛化能力，一般加在全连接层之后，可以尝试从0.2到0.5调整丢弃率。可以防止训练过拟合，主要原因如下：

学习普遍共性

由于随机的让一些节点不工作了，因此可以避免某些特征只在固定组合下才生效，有意识地让网络去学习一些普遍的共性，而不是某些训练样本的一些特性。

缩短时间

由于大大让网络变瘦了，运行时间也会大大提高。

下面测试，在上述代码的基础上，对于dropout进行调整，是否能产生影响。

0.2

Accuracy: mean=98.952 std=0.094, n=5

```
98.867
98.908
98.867
99.108
99.008
```

0.3

Accuracy: mean=98.973 std=0.105, n=5

```
98.875
98.883
98.950
99.167
98.975
```

0.4

```
98.817
98.942
99.008
99.108
98.900
```

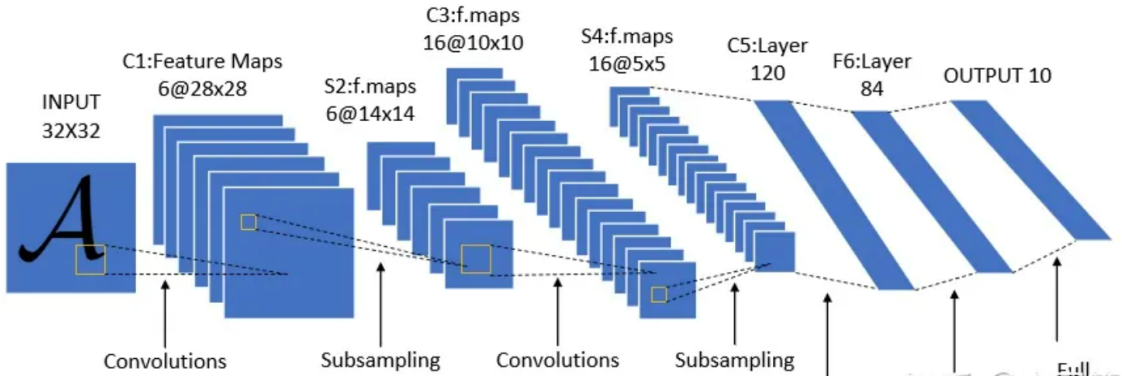
Accuracy: mean=98.955 std=0.099, n=5

	0.2	0.3	0.4	null
1	98.867	98.875	98.817	98.792
2	98.908	98.883	98.942	99.067
3	98.867	98.950	99.008	98.792
4	99.108	99.167	99.108	99.083
5	99.008	98.975	98.900	99.011
平均/方差	98.952/0.094	98.973/0.105	98.955/0.099	98.940/0.131

可以看到，在对dropout的不断调整之中，准确率在逐步提高，但是对于这个值的选取需要进行设置和调整，在选取了0.3的dropout率时，发现Accuracy一共提高了0.3%左右。属于是有效的正增长，并且考虑到较高的正确率，这个调整是非常有效的。

4.2.3 全连接层

通常情况下都会接2～3层。原因在于通常CNN在提取要特征后，flatten一个向量后维度会比较高（常见可能是1024，或者是2048），而对于最后一步的分类任务来说通常维度（也就是分类类别数）会很低。为了消除这种巨变，通常会由2到3个全连接层来完成分类任务，且全连接层的维度是逐步减少的。如卷积的输出特征为1024维度，分类任务数为10，那么最后可以加两个全连接层（1024，512，10），也可以三个（1024，512，256，10）。



在基础代码上，增加一层256的全连接层查看效果：

```
98.792
99.017
98.967
99.042
99.025
```

Accuracy: mean=98.968 std=0.092, n=5

在基础代码上，增加一层512、256的全连接层查看效果：

```
98.808
99.008
98.925
99.100
99.008
```

Accuracy: mean=98.980 std=0.098, n=5

可以看到，在增加了层数后，增加一层全连接层提高了0.2%，再增加了一层精确率增加了0.4%左右，并且std稳定在0.098，因此对全连接层的增加可以有效的进行提高。

	1024	1024、512	1024、512、256
1	98.792	98.892	98.808
2	99.067	99.017	99.008
3	98.792	98.967	98.980
4	99.083	99.042	99.100
5	99.011	99.025	99.008
平均/方差	98.940/0.131	98.971/0.092	98.980/0.098

4.3 层间参数验证

在此模块中，对每一层验证添加时想达到最大效果需要进行怎样的添加，否则将会失去效果。

4.3.1 卷积核大小

仍使用原有的最基础CNN模型，为了研究卷积核大小的变化对准确率的影响，分别调整卷积核为（2，2），（3，3），（4，4），（6，6），（7，7），（8，8），观察准确率的变化：

对于卷积核的大小，在各类实验中，一般都会设定为奇数*奇数的形式，其原因主要有三个：

（1）奇数更容易padding

在卷积时，我们有时候需要卷积前后的尺寸不变。这时候我们就需要用到padding。假设图像的大小，也就是被卷积对象的大小为**n*n**，卷积核大小为**k*k**，padding的幅度设为**(k-1)/2**时，卷积后的输出就为**(n-k+2*((k-1)/2))/1+1=n**，即卷积输出为**n*n**，保证了卷积前后尺寸不变。但是如果k是偶数的话，**(k-1)/2**就不是整数了。

（2）更容易找到卷积锚点

在CNN中，进行卷积操作时一般会以卷积核模块的一个位置为基准进行滑动，这个基准通常就是卷积核模块的中心。若卷积核为奇数，卷积锚点很好找，自然就是卷积模块中心，但如果卷积核是偶数，这时候就没有办法确定了，让谁是锚点似乎都不怎么好。

（3）获取中心信息。

在CNN中，由于奇数核拥有天然的绝对中心点，因此在做卷积的时候能更好地获取到中心这样的概念信息。

因此，我们希望在此处观测到：当换了卷积核尺寸为偶数时，精确率会有所下降。

（2，2）

```
98.525
98.642
98.508
98.808
98.742
```

Accuracy: mean=98.645 std=0.118, n=5

(3，3)

```
98.717
98.900
98.733
98.967
98.883
```

Accuracy: mean=98.840 std=0.098, n=5

(4，4)

```
98.775
98.825
98.833
98.925
98.892
```

Accuracy: mean=98.850 std=0.053, n=5

(6，6)

```
98.800
98.992
98.892
98.992
98.975
```

Accuracy: mean=98.930 std=0.075, n=5

(7，7)

```
98.908
99.058
98.892
99.050
99.000
```

Accuracy: mean=98.982 std=0.070, n=5

我们可以得出的结论为：随着核大小的增大，测试的准确率程上升趋势；且对比 3与4，我们发现精确率的变化仅仅是百分位中1的增加，但是在5与6，7与8中，甚至出现了后者更大的核反而精确率不如前者，因此也印证了要选取奇数而非偶数的结论。

	2 * 2	3 * 3	4 * 4	5 *5	6 * 6	7 * 7
1	98.525	98.717	98.775	98.792	98.800	98.908
2	98.642	98.900	98.825	99.067	98.992	99.058
3	98.508	98.733	98.833	98.792	98.892	98.892
4	99.808	98.967	98.925	99.083	98.992	99.050
5	98.742	98.883	98.892	99.011	98.975	99.000
平均/方差	98.645/0.118	98.840/0.098	98.850/0.053	98.940/0.131	98.930/0.075	98.982/0.070

4.3.2 卷积层过滤器

此处验证卷积层过滤器的数量对实验准确性的影响。

```
98.808
98.992
98.892
99.008
98.942
```

深度调整为16

Accuracy: mean=98.928 std=0.073, n=5

深度调整为64

```
98.800
98.950
98.808
99.000
98.967
```

Accuracy: mean=98.905 std=0.084, n=5

因此，可以看到，过滤器的数量变化后，观察到，过滤器数量在 32的基础上无论是折半还是翻倍，对准确率的影响都非常小，变化维持在0.01%左右，但显示出的变化率处于下降。考虑到实验存在偶然性，该效果几乎可以忽略不计，需要选择正确的过滤器数量。

	16	**32**	**64**
1	98.808	98.792	98.800
2	98.992	99.067	98.950
3	98.892	98.792	98.808
4	99.008	99.083	99.000
5	98.942	99.011	98.967
平均/方差	98.928/0.073	98.940/0.131	98.905/0.084

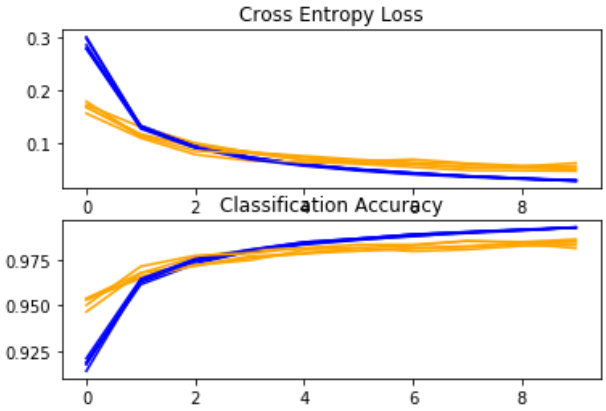
4.3.3 学习速度

对于梯度下降，一般选取的参数为0.01，这里尝试修改为别的参数

学习速度为0.001

Accuracy: mean=98.392 std=0.171, n=5

```
> 98.292
> 98.117
> 98.450
> 98.492
> 98.608
```



可以看到，在选择了其余的学习速度参数后，将会有0.5%的精确率下降，变化的幅度较大，所以选择的参数最好选择试验出的0.01。

4.4 图像加噪验证

4.4.1 高斯噪声

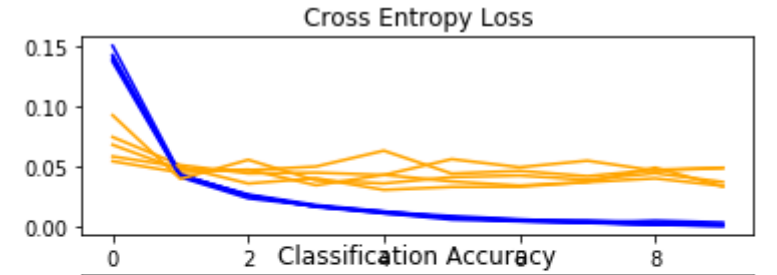
修改数据预处理部分，对于训练集，

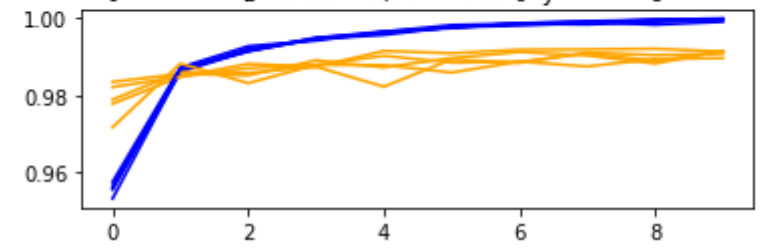
```
def prep_pixels(train, test):
    # 将训练集灰度值从 int 调整为 float train_norm = train.astype('float32')
    train_norm = train.astype('float32')
    test_norm = test.astype('float32')
    #train_norm = train_norm + np.random.randn(28, 28) * sigma / 255
    # 将灰度值归一化到 0-1 之间train_norm = train_norm / 255.0 test_norm =
test_norm / 255.0
    train_norm = train_norm / 255.0
    test_norm = test_norm / 255.0
    var = random.uniform(0.0001, 0.04)
    train_norm = skimage.util.random_noise(train_norm, mode='gaussian', var=var)
    # 返回归一化后的图像集
    return train_norm, test_norm
```

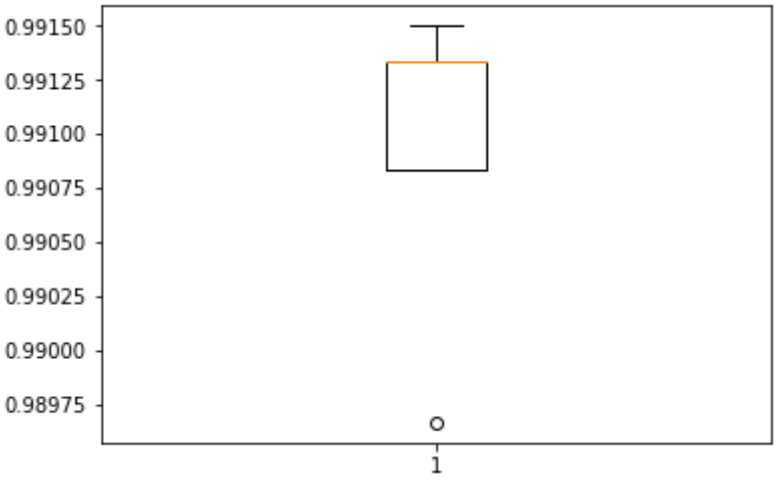
得到的结果如下：

```
98.967
99.150
99.083
99.133
99.133
```

Accuracy: mean=99.093 std=0.067, n=5





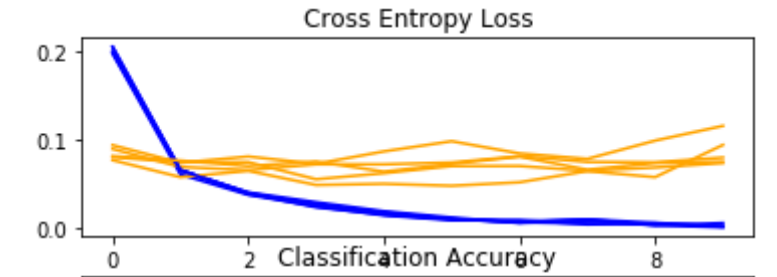


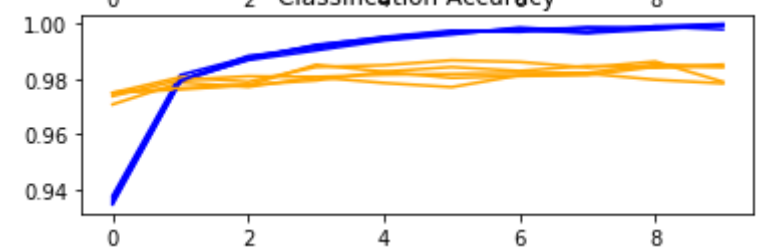
4.4.2 椒盐噪声

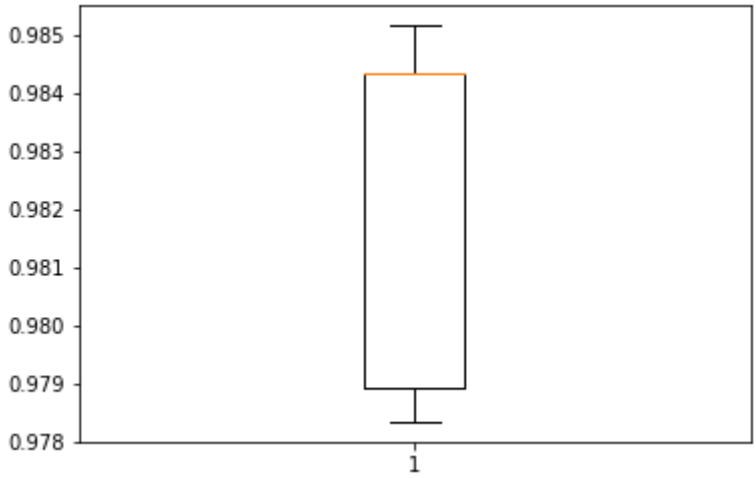
若对训练集加上椒盐噪声，得到的结果如下：

```
97.833
98.517
98.433
97.892
98.433
```

Accuracy: mean=98.222 std=0.295, n=5





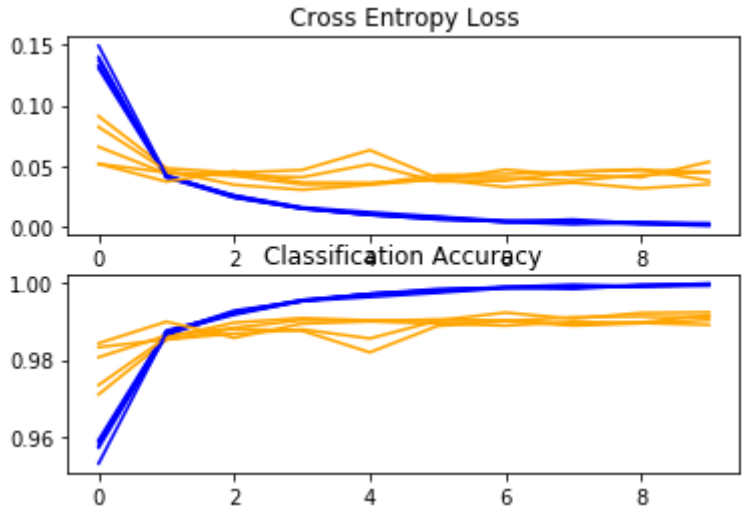


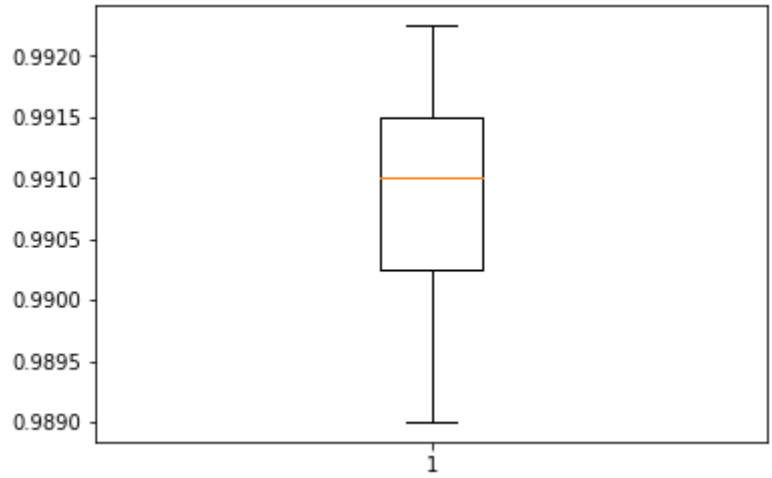
4.4.3 乘法噪声

只对训练集加上乘法噪声，得到的结果如下：

```
99.100
99.150
99.025
99.225
98.900
```

Accuracy: mean=99.080 std=0.111, n=5





4.5 实验结果

在通过了本次消融实验后，概括了几个有效的结论：

- 根据实验，变化最大的添加了一组相应的池化层和卷积层
- 增加全连接层的效果是本实验中第二好的
- 对于dropout层，其效果在本实验中并不十分显著，通过查阅资料，对于dropout层的使用，也有研究者提出其作用并不每次都很优越，如果需要使用，可以选择调整方位和选择较好参数不断实验。
- 对于几个层的参数选择，本次消融实验也验证了一些性质，例如过滤器数量没有很显著的影响，添加时选择32即可，例如卷积核的大小要选择奇数。
- 最后，综合此实验，最终选择了以下参数，进行了最终实验结果的输出，Accuracy: mean=99.187 std=0.055, n=5，得到的结果体现了较高的准确率。

```
def define_model(): #全程使用 ReLU 激活函数
    model = Sequential()
    model.add(Conv2D(32, (5, 5), activation='relu',
    kernel_initializer='he_uniform', input_shape=(28, 28, 1))) #使用 32 个过滤器、
3*3 卷积核
    model.add(MaxPooling2D((2, 2))) #2*2Pooling
    model.add(Conv2D(32, (5, 5), activation='relu',
    kernel_initializer='he_uniform'))
    model.add(MaxPooling2D((2, 2))) #2*2Pooling
    model.add(Flatten()) #扁平化
    model.add(Dense(1024, activation='relu',
    kernel_initializer='he_uniform')) #第一个全连接层
    model.add(Dense(256, activation='relu',
    kernel_initializer='he_uniform')) #第二个全连接层
    model.add(Dense(10, activation='softmax')) #softmax 归一# 编写模型
    opt = SGD(lr=0.01, momentum=0.9)
    #保守的随机梯度下降器配置，学习速度 0.01，动量 0.9
    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=
['accuracy'])
    return model
```

```
99.125
99.258
99.125
99.233
99.192
```

5. 实验心得

在本学期的教学任务中，在老师的带领下，我们学习了SVM，Transformer、CNN、VAE等许多的机器学习种的概念与工具，老师讲述了基础概念，数学推导和实现形式，并通过大作业的配合实践使我们从理论和实践两方面对于机器学习有了更加深刻的认识。

关于个人项目训练，在拿到题目之后的晚上我先是使用了第一次作业中的LibSVM跑了MNIST数据集，并且仅在把图像灰度数据scale到[0,1]，选择c=2，其它使用基础参数的时候，达到了95%左右的准确率。由于libsvm的使用较为简单，因此我又使用了较为典型的CNN对MNIST进行课程需要的消融实验。

在实验代码的设计中，我首先是对数据集调整到28*28，然后调整灰度值到了0-1之间，然后开始定义CNN模型，一开始我只保留了最为基础，并且较为常见的一些参数，并且使用了K-Fold交叉验证评估模型，并对验证结果中的Cross Entropy Loss与Classification Accuracy进行了可视化，可以更直观的显示代码的结果。

在测试之中，我进行的消融实验主要是对一些可以调整和选择的参数进行的，比如模型的深度，池化层、卷积层，Dropout层等影响，并且比较了这些层想要达到较好效果时应该选取怎样的参数，才能体现出作用。最终对于如何进行消融实验以及参数的调整都有了理解。

感谢赵老师一学期的教导，通过这节课，我对机器学习有了更好的理解。