**卷积神经网络实现手写数字识**

吴凯

人工智能2019-01班 2019114458

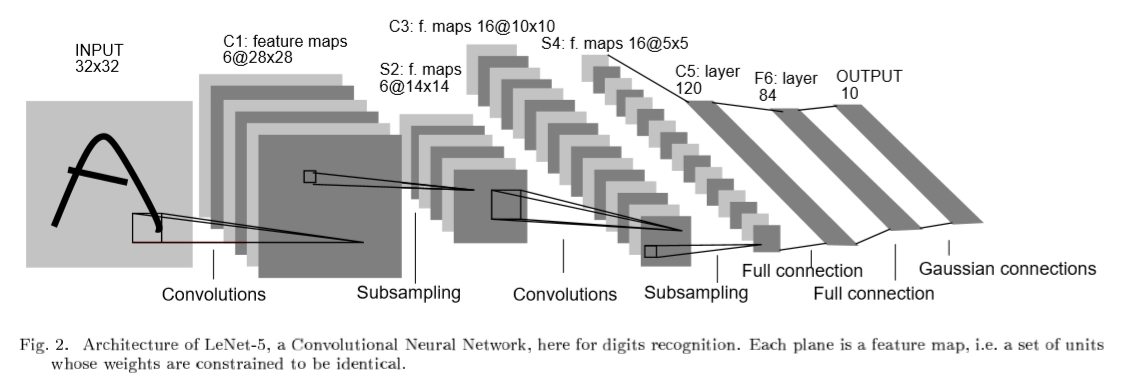
## 报告摘要

要求：

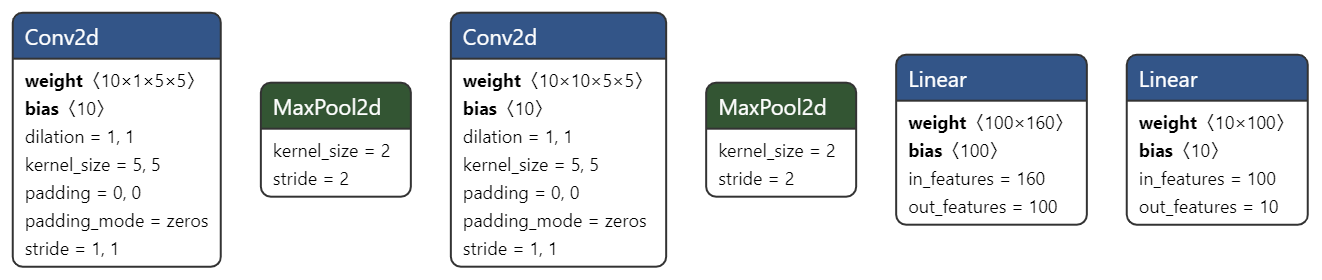
1. 基于 Pytorch 深度学习框架，搭建多层卷积神经网络模型，实现手写数字识 别，采用精确率、召回率、F1 值评测模型表现。
2. 调整学习率，报告不同值对应的模型表现，并加以分析。
3. 调整优化器，报告不同对应的模型表现，并加以分析。
4. 对比方法包括但不限于 SVM、决策树，报告其模型表现。
5. 课程报告内容包括但不限于神经网络模型结构介绍，损失函数介绍等。

思路：

本次采用CNN神经网络，此模型的结构是卷积层--池化层--卷积层--池化层--全连接层。具体LeNet-5结构如下图



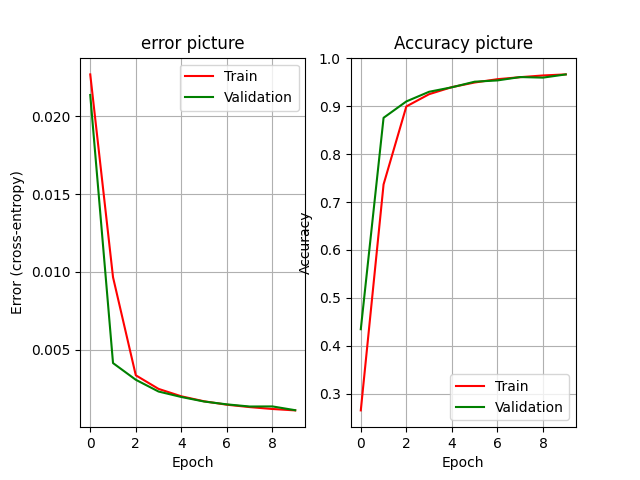
采用tensoboard可以记录训练过程的数字、图像等内容进行可视化以及采用Netron对模型结构可视化。（**本文模型结构**如下）



结论：

模型评测

|  |  |  |
| --- | --- | --- |
| 评价指标 划分算法 | 验证集 | 测试集 |
| 精确率 | 0.980 | 0.981 |
| 召回率micro | 0.980 | 0.981 |
| F1值 micro | 0.980 | 0.981 |



交叉熵损失图 (SGD) 与 精准率图

**交叉熵loss值 曲线 类似于滑梯式下降说明收敛效果良好，开始采用Adam自适应学习优化器观察优化过程，然后精致调SGD取得不错得成效。**

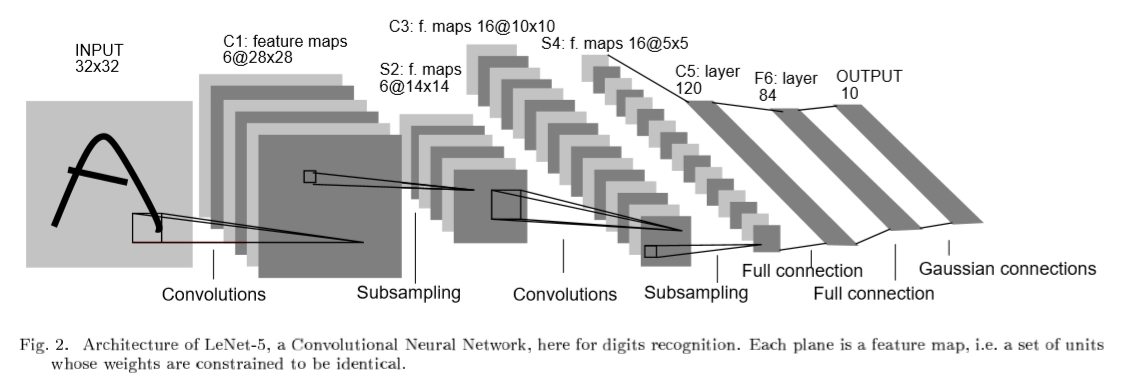
**精准度在Train中随着epoch升高，逐渐收敛，精准度也挺高，效果不错**

**但是在Validation中出现震荡起伏，可能出现过拟合现象或者是Adam自适应优化器的问题。接着精调SGD++Momentum的确解决掉了验证集的精准度出现震荡的情况，解决掉了过拟合。**

## 2 实验内容

### 2.1 CNN结构

本次采用CNN神经网络，此模型的结构是卷积层--池化层--卷积层--池化层--全连接层。具体LeNet-5结构如下图



#### 2.1.1卷积层

卷积层（Convolutional layer）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最佳化得到的。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，更多层的网路能从低级特征中迭代提取更复杂的特征。

主要的工作原理就是通过n\*n大小的kernel对每张图片的每各通道进行扫描，每次都是选取n\*n大小的数据跟kernel上的数据进行运算，得出一个值，而后继续一定，直到扫描完所有数据。（下图为本此实验的卷积可视化）

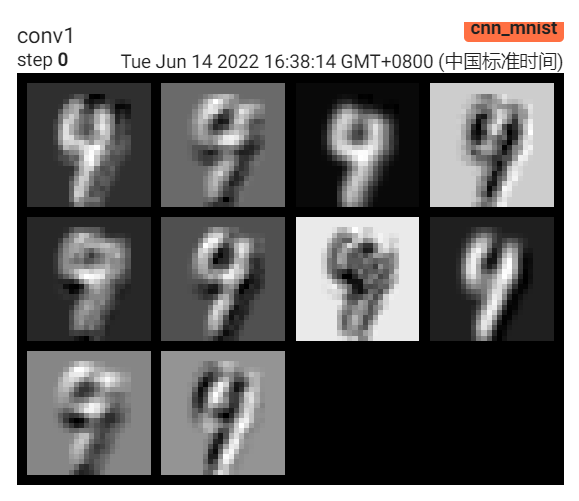
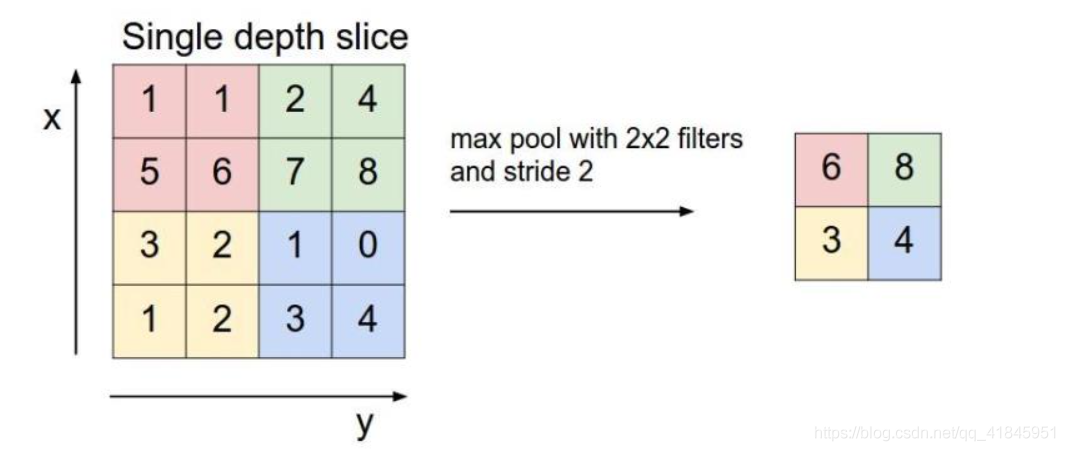


图 数字4第一次卷积的结果

#### 3.1.2池化层

池化层是夹在连续卷积层中间的，用于压缩数据和参数的量，减小过拟合。简而言之，如果输入是图像的话，那么池化层的最主要作用就是压缩图像。

下采样层也叫池化层，其具体操作与卷积层的操作基本相同，只不过下采样的卷积核为只取对应位置的最大值、平均值等（最大池化、平均池化），即矩阵之间的运算规律不一样，并且不经过反向传播的修改。如图

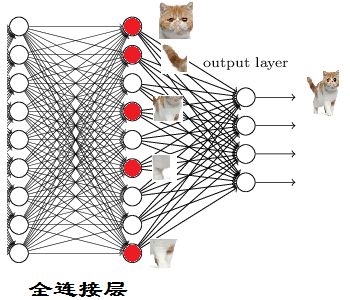


例如2\*2的kernel对4\*4的数据进行扫描，每次移动两步，那么算出来的值也就是一个2\*2大小的数据。

 因此池化层的作用就是保证**特征不变**，因为每次都还是会采样，保留重要的特征，其次就是**特征降维，**当特征量太多，我们会选择将一些没有用的特征丢掉，只保留有用的特征。（通常选择maxpool）

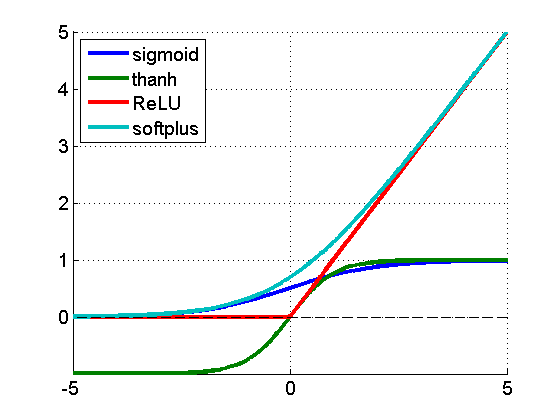
#### 3.1.3全连接层

全连接层的作用主要是进行**分类**。前面通过卷积和池化层得出的特征，在全连接层对这些总结好的特征做分类。全连接层就是一个完全连接的神经网络，根据权重每个神经元反馈的比重不一样，最后通过调整权重和网络得到分类的结果。



#### 3.1.4 激活函数

神经网络中的每个神经元节点接受上一层神经元的输出值作为本神经元的输入值，并将输入值传递给下一层，输入层神经元节点会将输入属性值直接传递给下一层（隐层或输出层）。在多层神经网络中，上层节点的输出和下层节点的输入之间具有一个函数关系，这个函数称为激活函数。

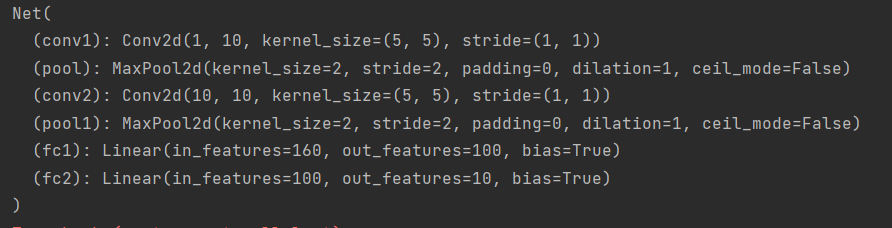


本次实验采用ReLU 激活函数，ReLU有以下优势：**对于线性函数而言，ReLU的表达能力更强**，尤其体现在深度网络中；而对于非线性函数而言，ReLU由于非负区间的梯度为常数，因此**不存在梯度消失问题(Vanishing Gradient Problem)**，使得**模型的收敛速度维持在一个稳定状态**。

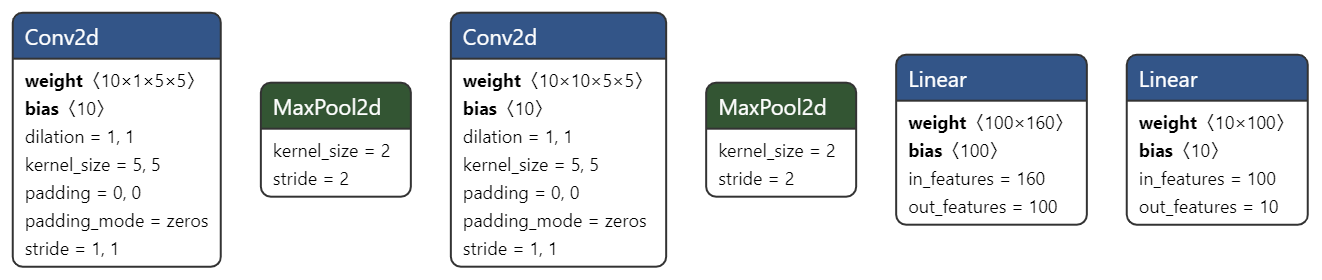
### 2.2 实现细节

#### 2.2.1 网络结构代码

class Net(nn.Module):  
 def \_\_init\_\_(self):  
 super(Net, self).\_\_init\_\_()  
 self.conv1 = nn.Conv2d(1, 10, kernel\_size=5, stride=1) # 卷积核个数10 卷积核尺寸 5 步长1 无边缘填充  
 self.pool = nn.MaxPool2d(kernel\_size=2, stride=2) # 2x2 maxpool  
 self.conv2 = nn.Conv2d(10, 10, kernel\_size=5, stride=1)  
 self.pool1 = nn.MaxPool2d(kernel\_size=2, stride=2) # 2x2 maxpool  
 self.fc1 = nn.Linear(4 \* 4 \* 10, 100) # 全连接层  
 self.fc2 = nn.Linear(100, 10)  
  
 def forward(self, x): # 28\*28\*1  
 x = F.relu(self.conv1(x)) # 24x24x10  
 x = self.pool(x) # 12x12x10  
 x = F.relu(self.conv2(x)) # 8x8x10  
 x = self.pool(x) # 4x4x10  
 x = x.view(-1, 4 \* 4 \* 10) # flattening  
 x = F.relu(self.fc1(x))  
 x = self.fc2(x)  
 return x



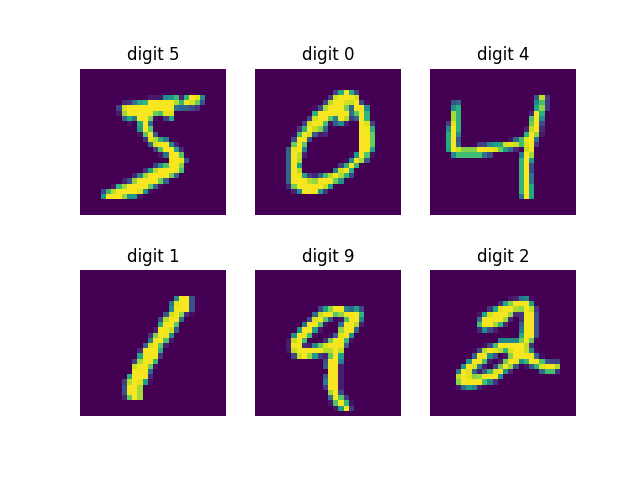
网络结构可视化如下



#### 2.2.2 查看mnist数据

def im\_convert(tensor):  
 image = tensor.to("cpu").clone().detach()  
 image = image.numpy().squeeze()  
 return image

for row in ax:  
 for col in row:  
 col.imshow(im\_convert(train\_loader.dataset[i][0]))  
 col.set\_title("digit "+str(train\_loader.dataset[i][1]))  
 col.axis("off")  
 i+=1



#### 2.2.3 参数设置

model = Net().cuda() # 切换到GPU运算  
print(model)  
optimizer = optim.Adam(model.parameters(),lr=0.01,betas=(0.9, 0.999),eps=1e-08,weight\_decay=0,amsgrad=False)  
# optimizer = optim.SGD(model.parameters(), lr=0.01) # 定义优化器 lr = 0.01  
# optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)  
# optimizer = optim.RMSprop(model.parameters(), lr=0.01, alpha=0.5)  
  
criterion = nn.CrossEntropyLoss() # 交叉熵损失函数

#### 2.2.4 CNN训练过程

for i in range(10):  
 total\_loss = 0  
 total\_acc = 0  
 c = 0  
 for images, labels in train\_loader:  
 images = images.cuda() # 切换为GPU数据  
 labels = labels.cuda()  
  
 optimizer.zero\_grad() # 梯度设置为0  
 output = model(images)  
 loss = criterion(output, labels) # 计算交叉熵loss  
 loss.backward() # 反向传播  
 optimizer.step() # 更新参数  
  
 total\_loss += loss.item()  
 total\_acc += torch.sum(torch.max(output, dim=1)[1] == labels).item() \* 1.0  
 c += 1  
  
 # validation  
  
 total\_loss\_val = 0  
 total\_acc\_val = 0  
 c = 0  
 for images, labels in validation\_loader:  
 images = images.cuda()  
 labels = labels.cuda()  
 output = model(images)  
 loss = criterion(output, labels)  
  
 total\_loss\_val += loss.item()  
 total\_acc\_val += torch.sum(torch.max(output, dim=1)[1] == labels).item() \* 1.0  
 c += 1  
  
 train\_errors.append(total\_loss / n\_train)  
 train\_acc.append(total\_acc / n\_train)  
 val\_errors.append(total\_loss\_val / n\_val)  
 val\_acc.append(total\_acc\_val / n\_val)  
  
print("Trainig complete")

#### 2.2.5 卷积过程可视化

请在pycharm Terminal中输入：点击http://localhost:6006/

tensorboard --logdir runs

writer = SummaryWriter('runs/cnn\_mnist', comment='feature map1')  
for i, data in enumerate(train\_loader, 0):  
 # 获取训练数据  
 inputs, labels = data  
 x = inputs[1].unsqueeze(0) # x 在这里呀  
 break  
  
img\_grid = vutils.make\_grid(x, normalize=True, scale\_each=True, nrow=2)  
model.eval()  
for name, layer in model.\_modules.items():  
  
 # 为fc层预处理x  
 x = x.view(x.size(0), -1) if "fc" in name else x  
 print(x.size())  
  
 x = layer(x)  
 print(format(name))  
  
 # 查看卷积层的特征图  
 if 'layer' in name or 'conv' in name:  
 x1 = x.transpose(0, 1) # C，B, H, W ---> B，C, H, W  
 img\_grid = vutils.make\_grid(x1, normalize=True, scale\_each=True, nrow=4) # normalize进行归一化处理  
 writer.add\_image(format(name), img\_grid, global\_step=0)

### 2.3 实验数据介绍 以及 实验配置

MNIST是一个手写体数字的图片数据集，该数据集来由美国国家标准与技术研究所（National Institute of Standards and Technology (NIST)）发起整理，一共统计了来自250个不同的人手写数字图片，其中50%是高中生，50%来自人口普查局的工作人员。该数据集的收集目的是希望通过算法，实现对手写数字的识别。1998年，Yan LeCun 等人发表了论文《Gradient-Based Learning Applied to Document Recognition》，首次提出了LeNet-5 网络，利用上述数据集实现了手写字体的识别。

****

**实验配置; PyCharm community 2021 显卡 GTX 1650**

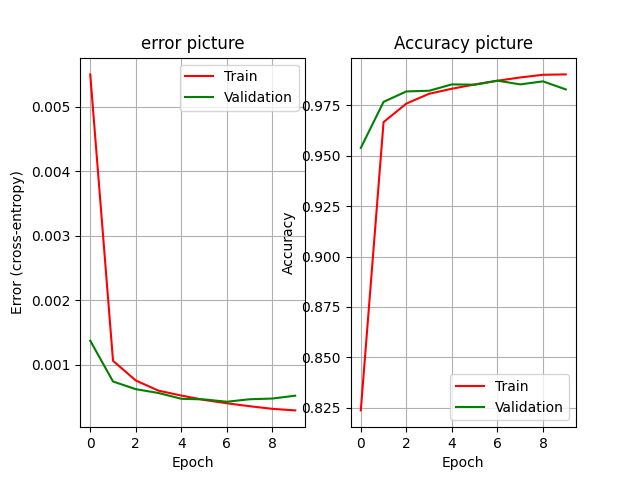
### 2.4 评价指标介绍

**召回率**：以实际样本为判断依据，实际为正例的样本中，被预测正确的正例占总实际正例样本的比例。召回率的另一个名字，叫做“查全率”

**精确度**：以预测结果为判断依据，预测为正例的样本中预测正确的比例。

**F1 ：** F1分数可以看作是模型精确率和 召回率 的一种调和平均

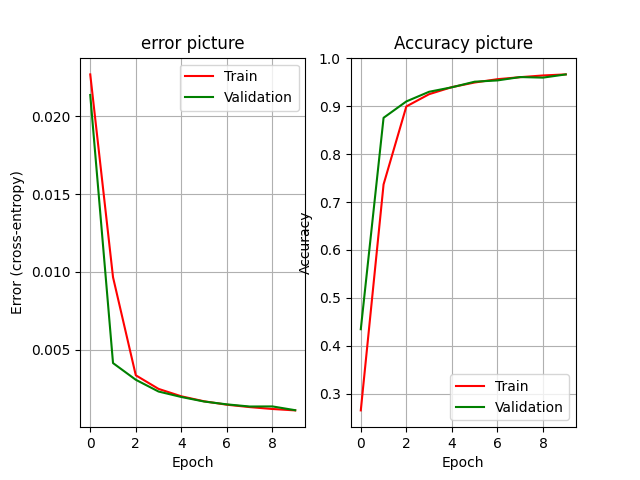
### 2.5 实验结果分析



**交叉熵loss值 曲线 类似于滑梯式下降说明收敛效果良好，开始采用Adam自适应学习优化器观察优化过程，然后精致调SGD取得不错得成效。**

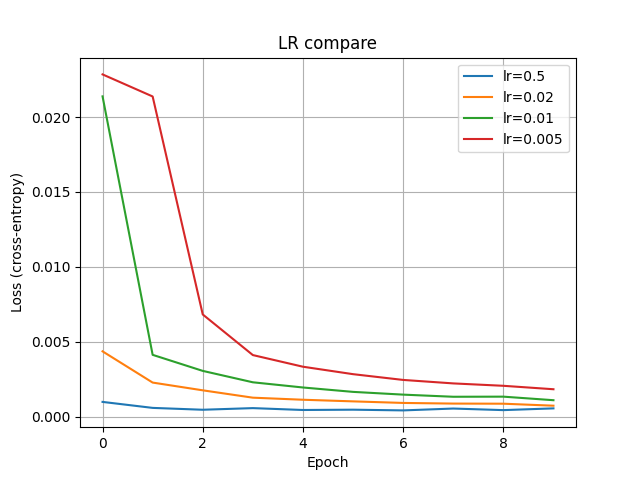
**精准度在Train中随着epoch升高，逐渐收敛，精准度也挺高，效果不错**

**但是在Validation中出现震荡起伏，可能出现过拟合现象或者是Adam自适应优化器的问题。**



接着精调SGD++Momentum的确解决掉了验证集的精准度出现震荡的情况，解决掉了过拟合。

#### 2.5.1 不同学习率的比较

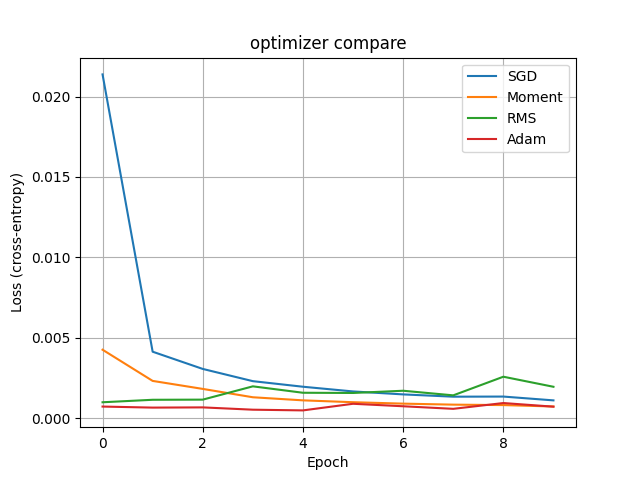


优化器为SGD

当学习率设置的**过小**时，**收敛过程将变得十分缓慢**。而当学习率设置的**过大**时，**梯度可能会在最小值附近来回震荡，甚至可能无法收敛**。

有图可知学习率0.02或者0.01在**SGD**随机梯度下降较为优的学习率。本次实验采用的Adam**逐参数适应学习率方法**

#### 2.5.2 不同优化器的比较



* 随机梯度下降法(SGD)：虽然SGD需要走很多步的样子，但是对梯度的要求很低（计算梯度快）。而对于引入噪声，大量的理论和实践工作证明，只要噪声不是特别大，SGD都能很好地收敛。但是SGD在随机选择梯度的同时会引入噪声，使得权值更新的方向不一定正确。

此外，SGD也没能单独克服局部最优解的问题。

* 使用动量(Momentum)的随机梯度下降法(SGD) 动量主要解决SGD的两个问题：一是随机梯度的方法（引入的噪声）；二是Hessian矩阵病态问题（可以理解为SGD在收敛过程中和正确梯度相比来回摆动比较大的问题）。
* RMSProp算法修改了AdaGrad的梯度积累为指数加权的移动平均，使得其在非凸设定下效果更好。本次实验震荡幅度大，不收敛
* Adam算法

Adam等自适应学习率算法对于稀疏数据具有优势，且收敛速度很快；但精调参数的SGD（+Momentum）往往能够取得更好的最终结果。**由图可知本次实验由 SGD（+Momentum）也能取得不错的效果**

**调试过程中tricks：在模型设计实验过程中，要快速验证新模型的效果，可以先用Adam进行快速实验优化；在结果出来前，可以用精调的SGD进行模型的极致优化。（数据集一定要充分的打散（shuffle）。这样在使用自适应学习率算法的时候，可以避免某些特征集中出现，而导致的有时学习过度、有时学习不足，使得下降方向出现偏差的问题。）**

**考虑不同算法的组合也可以。先用Adam进行快速下降，而后再换到SGD进行充分的调优。**

#### 2.5.3 不同模型的比较

采用了SVM、决策树算法、朴素贝叶斯算法、KNN四中算法对MINIST数据进行分类。从实验结果来看，就准确率而言：朴素贝叶斯算法 > SVM> KNN>决策树算法

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 算法种类 | SVM | 朴素贝叶斯 | 决策树 | KNN | **CNN** |
| 准确率(validation) | **0.976** | 0.825 | 0.862 | 0.969 | **0.980** |
| 准确率(test) | **0.978** | 0.837 | 0.872 | 0.968 | **0.981** |

CNN > SVM > KNN> 决策树 > 朴素贝叶斯

1.SVM

优点：适合小样本、非线性、高维模式识别。

缺点：对于大规模数据开销大，不合适多分类；对缺失数据敏感；需要选择适当的核函数。

2.决策树

优点：简单易于理解，能够处理多路输出问题。

缺点：容易过拟合；决策树的生成不稳定，微小的数据变化可能导致生成的决策树不同。

3.KNN

优点：简单易于理解，无需训练，无需估计参数准确性高；适合多标签问题。

缺点：懒惰算法，预测慢，开销大类的样本数不平衡时准确率受影响；可解释性差。

4.朴素贝叶斯

优点：分类稳定，适合小规模数据和增量式训练，对缺失数据不敏感。

缺点：属性相关性大时效果不好，需要知道先验概率，对数据的表达形式敏感

5.CNN

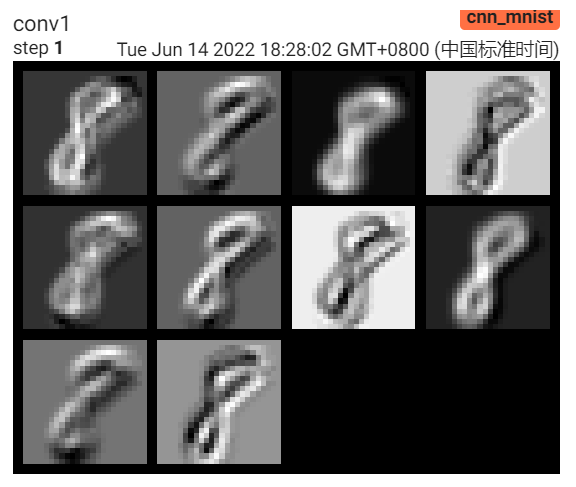
优点：优点共享卷积核，处理高维数据无压力；可以自动进行特征提取

缺点：采用梯度下降算法很容易使训练结果收敛于局部最小值而非全局最小值；

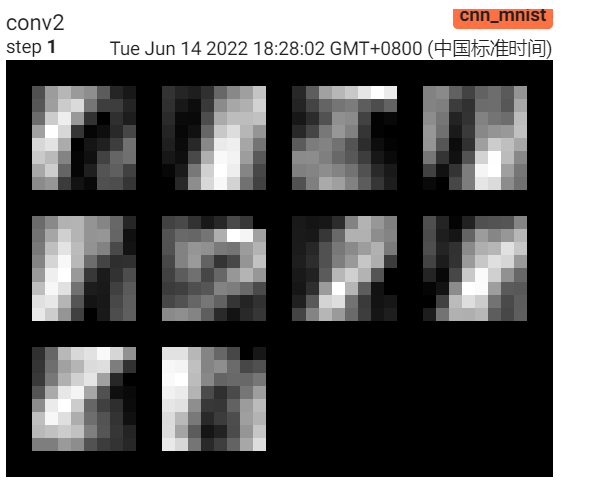
#### 2.5.4 卷积过程可视化



原始数字8

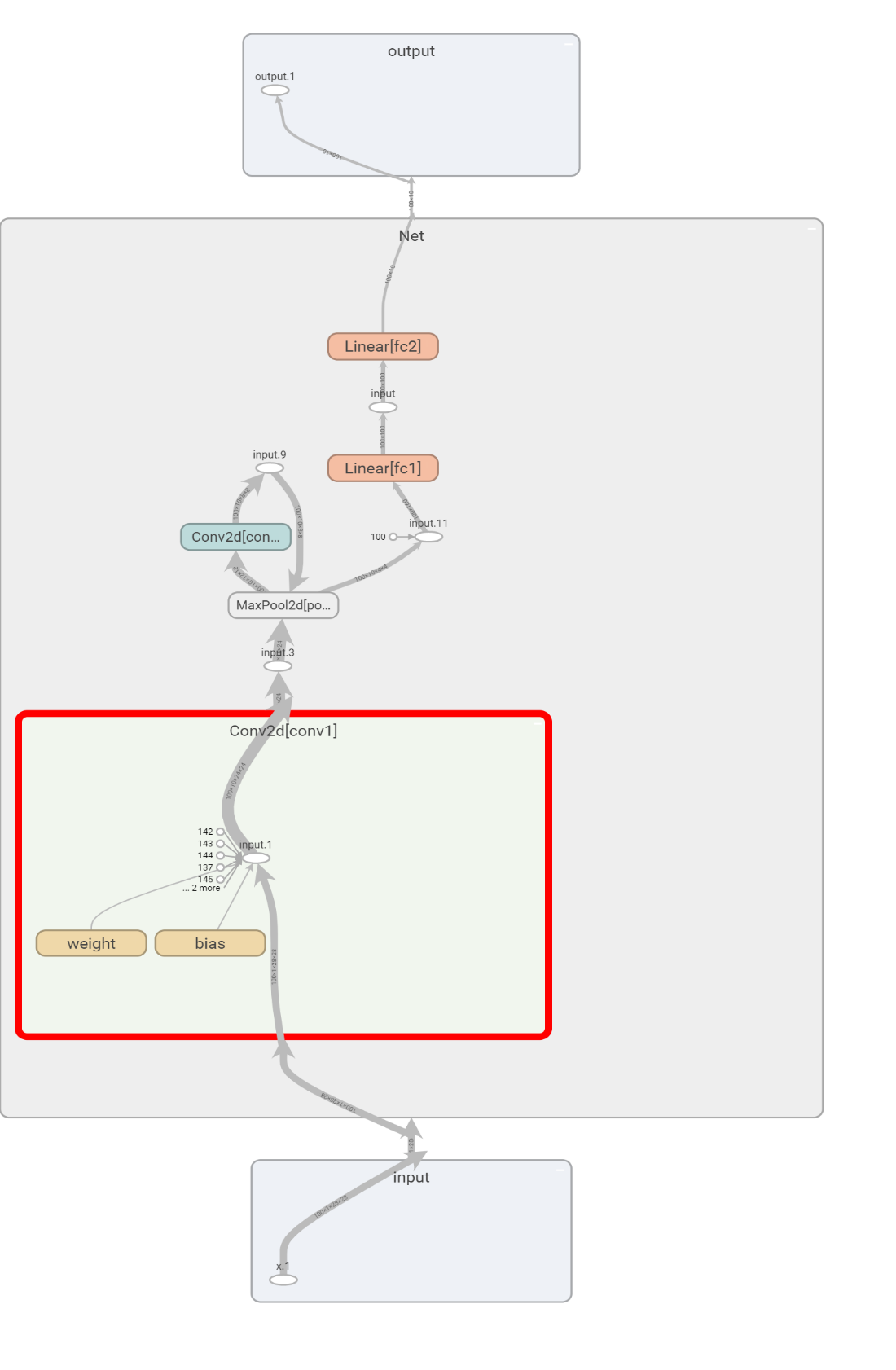


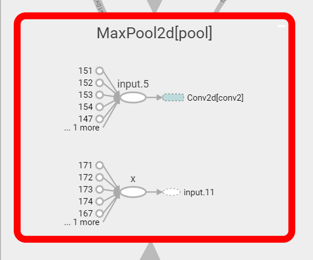
第一次卷积可视化



第二次卷积可视化

#### 2.5.5 模型过程可视化





## 3 总结及问题说明

本次课设为建模，调参，可视化，优化，

将已经分类好的数据通过像素的方式输入到有预设参数的模型中，模型根据输入的数据，不停的优化自身的参数，类似线性函数不断拟合w和b一样，最终建立具有最优参数的模型。神经网络以及这些数据都是很抽象的，在学习深度学习中可视化跟建模一样重要。

模型也就是神经网络，里面的卷积层细化特征，而池化层是取样，取出有用的特征值，进而降维，再是用全连接层对特征值进行分类，其间包括归一化，激活函数等优化操作，之所以正确率会提高，主要是因为前向传播以及反向传播，不断的更新参数，而梯度的更新，归根揭底就是梯度下降算法。

在解决实际问题时，我们需要做的就是选取一个合理的模型，预设合理的参数，采用各种优化处理，最终达到想要的效果。