

# 技术报告

## 使用方法介绍

使用卷积神经网络(CNN)提取图像特征进行识别。

## 方法实现细节

1. 使用pytorch架构。运用nn工具包中的卷积层、批规范化层、激活函数、最大池化、全连接层等搭建网络。
2. 先用Dataloader导入数据集，将60000个数据以5: 1的比例分为训练集和验证集

```
valid_data = train_data[50000:60000]
train_data = train_data[0:50000]
train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True)
valid_loader = DataLoader(valid_data, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(x_test_data, batch_size=batch_size, shuffle=False)
```

3. 用神经网络训练数据集，调整batch size, learning rate等参数，寻找到最优的网络。

将网络输出的结果与label输入损失函数计算损失值loss

在实现梯度反向传递时主要需要三步：

- 初始化梯度值： `optimizer.zero_grad()`
- 反向求解梯度： `loss.backward()`
- 更新参数： `optimizer.step()`

在指定的epoch次数内进行训练

4. 用验证集对训练集训练出的网络进行验证，计算输出的正确率

```
_, pred = torch.max(out, 1)
num_correct = (pred == label).sum()
```

不断调整参数选出最优的网络

5. 最后将测试数据输入训练好的网络，生成结果。

## 模型结构

layer1: 卷积层，批规范化，激活函数relu

layer2: 最大池化层

layer3: 卷积层，批规范化，激活函数relu

layer4: 最大池化层

layer5: 卷积层

fc: 三个全连接层 + 激活函数relu

```
self.layer1 = nn.Sequential(
    nn.Conv2d(1, 32, kernel_size=3),
```

```

        nn.BatchNorm2d(32),
        nn.ReLU(inplace=True)
    )

    self.layer2 = nn.Sequential(
        nn.MaxPool2d(kernel_size=2, stride=2)
    )

    self.layer3 = nn.Sequential(
        nn.Conv2d(32, 64, kernel_size=3),
        nn.BatchNorm2d(64),
        nn.ReLU(inplace=True)
    )

    self.layer4 = nn.Sequential(
        nn.MaxPool2d(kernel_size=2, stride=2)
    )

    self.layer5 = nn.Conv2d(64, 120, kernel_size=3, padding=1)

    self.fc = nn.Sequential(
        # 28x28 -> 26x26 -> 13x13 -> 11x11 -> 5x5
        nn.Linear(120 * 5 * 5, 1024),
        nn.ReLU(inplace=True),
        nn.Linear(1024, 128),
        nn.ReLU(inplace=True),
        nn.Linear(128, 10)
    )

```

## 调参过程

### 1、学习率

| 学习率 大 | 学习率 小           |                |
|-------|-----------------|----------------|
| 学习速度  | 快               | 慢              |
| 使用时间点 | 刚开始训练时          | 一定轮数过后         |
| 副作用   | 1.易损失值爆炸；2.易振荡。 | 1.易过拟合；2.收敛速度慢 |

先设置学习率为1，发现loss为nan，然后以0.1的倍数减少学习率，验证模型准确率，发现学习率为0.1时学习效果最好。

后来发现增大训练次数epoch时偶尔loss会变大，说明发生震荡。因此learning rate应随epoch增大变小，我设置每训练10000次  $\text{learning rate} = \text{learning rate} \times \gamma (\gamma = 0.5)$  以减小震荡。

### 2、batch size

先设置batch size为128，分别增大两倍减小两倍比较学习效果，得出batch size为32时学习效果最好。

| batch size小     | batch size 大           |
|-----------------|------------------------|
| 1.易损失值爆炸；2.易振荡。 | 1.易陷入局部最小值，过拟合；2.收敛速度慢 |
| 限于时间            | 限于空间                   |

### 3、网络层数，神经元个数，和通道数

因为数据集较为简单，不需要很深的网络和很多神经元。对网络层数作了一些增删改动之后，发现差别不大。

### 4、其他参数

其它参数直接使用了已知效果较好的方法，如激活函数使用relu函数，损失函数用了交叉熵，优化方法使用随机梯度下降(SGD)

## 遇到的问题解决方法

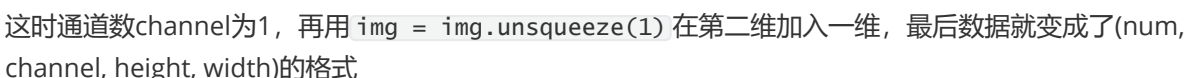
1. 用cuda进行计算加速。
2. 输入con2d的数据维数不对，应为4维(num, channel, height, width)，而经处理后的数据维数为两维(num, content)。

数据集中的train\_data和train\_label是分别给出的，不能直接shuffle。所以先将数据内容(height × width)变成一维(content)，再用np.hstack将两个数据集连接。

```
x_train_data = np.array(x_train_data).reshape(x_train_data.shape[0], -1)
y_train_data = np.array([y_train_data]).T
train_data = np.hstack((x_train_data, y_train_data))
```

训练时取最后一列为label，前n-1列为data。但此时data为两维(num, content)，再将一维的数据内容(content)变成两维(height × width)

```
img = img.reshape(img.shape[0], 28, 28)。
```

这时通道数channel为1，再用img = img.unsqueeze(1)在第二维加入一维，最后数据就变成了(num, channel, height, width)的格式

### 3. .npz数据集的加载

- o npz是numpy提供的数组存储方式，利用np.load函数读取后得到一个类似字典的对象，可以通过关键字进行值查询，关键字对应的值其实就是一个numpy数据。例如：

```
x_train_data = np.load('X_kannada_MNIST_train.npz')['arr_0']
```

## “北航数据工作站”评测分数和排名（附截图）

评测分数：0.9768

排名：33

|    |              |    |          |             |
|----|--------------|----|----------|-------------|
| 33 | 17377118_栗亚舟 | 25 | 12/01/19 | 0.9768 (32) |
|----|--------------|----|----------|-------------|