Lab1 卷积神经网络

和泳毅 PB19010450

一、实验目标

1. 使用Pytorch实现CNN和ResNet，并在Tiny ImageNet数据集上进行图片分类；
2. 研究dropout对卷积神经网络泛化性能的影响；
3. 研究normalization对卷积神经网络的影响；
4. 研究residual connection对深层卷积神经网络性能的影响；
5. 研究学习率、学习率衰减对卷积神经网络性能的影响；
6. 研究网络深度对卷积神经网络性能的影响。

二、数据集介绍

Tiny Imagenet是斯坦福大学提供的图像分类数据集，其中包含200个类别，每个类别包含500张训练图像，50张验证图像及50张测试图像。

三、实验原理

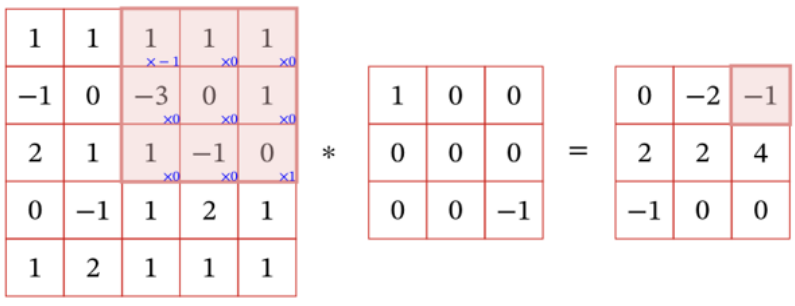
1. 二维卷积

在图像处理中，图像是以二维矩阵的形式输入到神经网络中，因此我们需要二维卷积。

一个输入信息X和滤波器W的二维卷积定义如下:

$$\boldsymbol{Y} = \boldsymbol{W} * \boldsymbol{X},$$

$$y_{ij} = \sum_{u=1}^{U} \sum_{v=1}^{V} w_{uv} x_{i-u+1, j-v+1}.$$

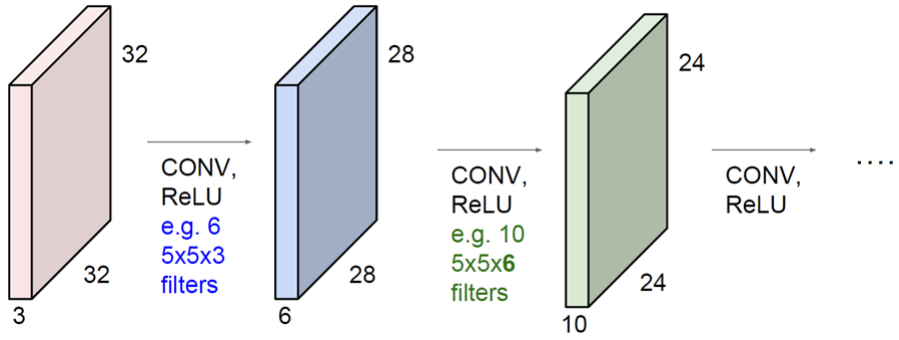


通常记N为原矩阵大小，F为卷积核大小，S为步幅大小，则二维卷积输出矩阵大小为：

$$\mathrm{OUT} = (N - F)/S + 1.$$

2、多层卷积

卷积神经网络是用卷积层代替全连接层。

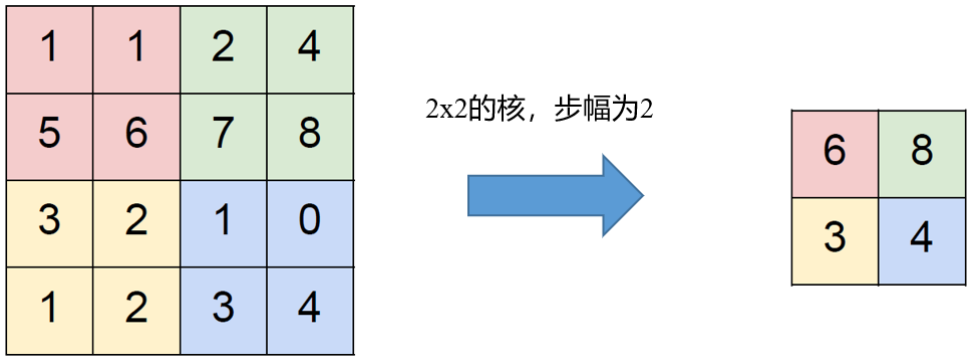


3. 填充（PADDING）

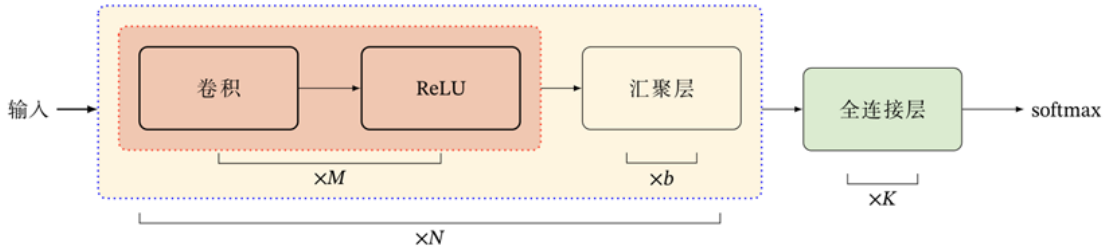
32×32的输入，每次通过5×5的核卷积，大小都会发生改变!(32 -> 28 -> 24 ...)。大小改变太快是不好的，实际中的性能很差。所以在输入周边填0，以解决特征图大小改变太快问题。填0时，经常要求保持输出大小与输入大小一样。

4. 池化

池化函数使用某一位置的相邻输出的总体统计特征来代替网络在该位置的输出，使得特征图尺寸更小且更易于管理。最大池化（max pooling）函数给出相邻矩形区域内的最大值。其他常用的池化函数包括相邻矩形区域内的平均值、$L2$ 范数以及基于据中心像素距离的加权平均函数。



5. CNN典型结构



一个卷积块为连续M个卷积层和b个汇聚层（M通常设置为$2 \sim 5$，b为0或1）。一个卷积网络中可以堆叠N个连续的卷积块，然后在接着K 个全连接层（N的取值区间比较大，比如$1 \sim 100$或者更大；K一般为$0 \sim 2$）。

6. ResNet

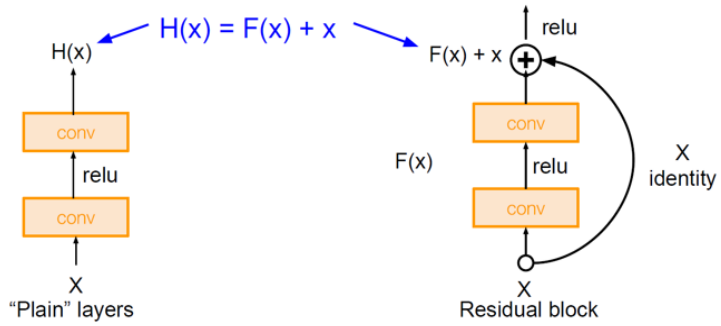
在卷积神经网络上不断加深网络，训练集的错误率出现上升的情况，这种现象背后的问题是优化问题，越深的模型越难训练。

残差网络（Residual Network，ResNet）是通过给非线性的卷积层增加直连边的方式来提高信息的传播效率。假设在一个深度网络中，我们期望一个非线性单元（可以为一层或多层的卷积层）$F(x, \theta)$去逼近一个目标函数为$h(x)$。将目标函数拆分成两部分：**恒等函数**和**残差函数**。

$$h(\boldsymbol{x}) = \boldsymbol{x} + (h(\boldsymbol{x}) - \boldsymbol{x})$$

恒等函数　残差函数

$F(x, \theta)$



四、核心代码

以ResNet为例。

1. ResNet卷积块

模型的主体结构由基础卷积块堆叠而成。基础块由两层3×3的卷积网络构成，可以通过参数设定该基础块的输入通道数，输出通道数，可以选择是否在第一层卷积网络中使用长度为2的步长来缩减特征图的大小。

```
class ResNetLayer(nn.Module):
    def __init__(self, in_feature_maps, out_feature_maps, downsample = True):
        super(ResNetLayer, self).__init__()

        self.stride = 2 if downsample == True else 1
```

```
        self.conv0 = nn.Conv2d(in_feature_maps, out_feature_maps, 3, stride = self.stride, padding = 1)
        self.bn0 = nn.BatchNorm2d(out_feature_maps)
        self.conv1 = nn.Conv2d(out_feature_maps, out_feature_maps, 3, stride = 1, padding = 1)
        self.bn1 = nn.BatchNorm2d(out_feature_maps)

        self.skipconn_cnn = nn.Conv2d(in_feature_maps, out_feature_maps, kernel_size=1, stride=self.stride, padding = 0)
        self.skipconn_bn = nn.BatchNorm2d(out_feature_maps)
        self.dropout = nn.Dropout(0.2)
    def forward(self, input):
        identity = input
        x = F.relu(self.bn0(self.dropout(self.conv0(input))))
        x = self.bn1(self.conv1(x))
        x += self.skipconn_bn(self.skipconn_cnn(identity))
        x = F.relu(self.dropout(x))
        return x
```

2. ResNet模型

固定网络的输入层和输出层，中间层使用若干基础块堆叠，可以自由调整网络的深度。

```
class ResNet(nn.Module):
    def __init__(self):
        super(ResNet, self).__init__()

        self.conv0 = nn.Conv2d(3, 64, 5, stride = 1, padding = 2)
        self.bn0 = nn.BatchNorm2d(64)

        self.maxpool = nn.MaxPool2d(3, 2, 1)
        self.resnetlayer1 = ResNetLayer(64, 64, False)
        self.resnetlayer2 = ResNetLayer(64, 128)
        self.resnetlayer3 = ResNetLayer(128, 128, False)
        self.resnetlayer4 = ResNetLayer(128, 256)
        self.resnetlayer5 = ResNetLayer(256, 256, False)
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.dropout1 = nn.Dropout(0.2)
        self.fc = nn.Linear(256, 200)

        self.dropout = nn.Dropout(0.15)

    def forward(self, input):
        x = F.relu(self.bn0(self.dropout(self.conv0(input))))
```

```
        x = self.maxpool(x)
        x = self.resnetlayer1(x)
        x = self.resnetlayer2(x)
        x = self.resnetlayer3(x)
        x = self.resnetlayer4(x)
        x = self.resnetlayer5(x)
        x = self.avgpool(x)
        x = x.view(x.shape[0], -1)
        x = self.dropout1(x)
        x = self.fc(x)
```

五、实验结果

1. CNN

CNN默认参数如下：

```
batch_size = 128
max_epochs = 100
lr = 0.015 # 初始学习率
optimizer = torch.optim.Adam(model.parameters(), lr=lr) # Adma优化器
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'max', factor=0.1, patience=2,threshold=0.0001, verbose=True, min_lr=1e-8) # 以正确率为更新学习率目标，衰减因子为0.1，patience为2
criterion = nn.CrossEntropyLoss()
dropout1 = 0.2 # 卷积层dropout
dropout2 = 0.2 # 全连接层dropout
```

CNN默认结构如下：

```
Input
5x5 conv,64
maxpool
Layer(64,128)
--3x3 conv,128
--3x3 conv,128
--maxpool
Layer(128,256)
--3x3 conv,256
--3x3 conv,256
--maxpool
Layer(256,512)
--3x3 conv,512
```

```
--3x3 conv,512
--maxpool
avgpool
fc,256
fc,200
```

1.1 Dropout

固定其他参数，调整dropout为以下几组值：

```
'dropout':(dropout1,dropout2)=[(0,0),(0.1,0.3),(0.2,0.2),(0.2,0.5),(0.3,0.7)]
```

在验证集上验证，结果如下

Dropout	Loss	Top 1 acc	Top 5 acc
(0,0)	0.0213	0.4183	0.6772
(0.1,0.1)	0.0199	0.4405	0.6869
(0.1,0.3)	0.0201	0.4435	0.6827
(0.2,0.2)	**0.0189**	**0.4512**	**0.7009**
(0.2,0.5)	0.0239	0.3233	0.5837
(0.3,0.7)	0.0357	0.0675	0.2175

dropout概率太小或者不使用dropout技术，过拟合风险大。dropout概率太大，会导致欠拟合。应该选择合适的dropout概率来降低过拟合风险，提高模型性能。

2.2 Normalization

Normalization	Loss	Top 1 acc	Top 5 acc
True	**0.0189**	**0.4512**	**0.7009**
False	0.0419	0.0050	0.0250

如果不加标准化层，网络学不到东西。

2.3 学习率衰减因子

固定其他参数，调整学习率衰减因子为0.1、0.2、0.5、0.99：

Factor	Loss	Top 1 acc	Top 5 acc
0.1	**0.0189**	**0.4512**	**0.7009**
0.2	0.0213	0.3833	0.6421
0.5	0.0205	0.4208	0.6714
0.99	0.0208	0.3925	0.6487

在训练网络时，应该动态调节学习率。初始学习率可以略大来加快收敛速度，在接近极小值时，如果不减小学习率，会产生震荡而无法收敛。

2.4 网络深度

固定其他参数，调整网络深度为以下层数：

```
8层: [5x5 conv,64],[Layer(64,128)],[Layer(128,256)],[Layer(256,512)],[fc,256]
10层: [5x5 conv,64],[Layer(64,128)],[Layer(128,256)],[Layer(256,512)],
[Layer(512,1024)],[fc,512]
12层: [5x5 conv,64],[Layer(64,128)],[Layer(128,256)],[Layer(256,512)],
[Layer(512,1024)],[Layer(1024,2048)],[fc,1024]
```

深度	Loss	Top 1 acc	Top 5 acc
8层	**0.0189**	**0.4512**	**0.7009**
10层	0.0224	0.4105	0.6701
12层	0.0227	0.4031	0.6458

网络越深，性能不一定越好。更深层的网络更难优化，更容易陷入局部极小值。

2.ResNet

ResNet默认参数如下：

```
batch_size = 128
max_epochs = 100
lr = 0.015 # 初始学习率
optimizer = torch.optim.Adam(model.parameters(), lr=lr) # Adma优化器
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'max', factor=0.1,
patience=2,threshold=0.0001, verbose=True, min_lr=1e-8) # 以正确率为更新学习率目标，
衰减因子为0.1，patience为2
criterion = nn.CrossEntropyLoss()
dropout1 = 0.2 # 卷积层dropout
dropout2 = 0.2 # 全连接层dropout
```

ResNet默认结构如下：(11层)

```
Input
5x5 conv,64
maxpool
Layer(64,64)
Layer(64,128)
Layer(128,128)
Layer(128,256)
Layer(256,256)
avgpool
fc,200
```

1.1 Dropout

固定其他参数，调整dropout为以下几组值：

```
'dropout':(dropout1,dropout2)=[(0,0),(0.1,0.3),(0.2,0.2),(0.2,0.5),(0.3,0.7)]
```

在验证集上验证，结果如下

Dropout	Loss	Top 1 acc	Top 5 acc
(0,0)	0.0224	0.3864	0.6417
(0.1,0.1)	0.0201	0.4321	0.6889
(0.1,0.3)	0.0204	0.4146	0.6724
(0.2,0.2)	**0.0196**	**0.4604**	**0.7210**
(0.2,0.5)	0.0217	0.4031	0.6458
(0.3,0.7)	0.0255	0.2708	0.5492

dropout概率太小或者不使用dropout技术，过拟合风险大。dropout概率太大，会导致欠拟合。应该选择合适的dropout概率来降低过拟合风险，提高模型性能。

2.2 Normalization

Normalization	Loss	Top 1 acc	Top 5 acc
True	**0.0196**	**0.4604**	**0.7210**
False	0.0419	0.0050	0.0250

如果不加标准化层，网络学不到东西。

2.3 学习率衰减因子

固定其他参数，调整学习率衰减因子为0.1、0.2、0.5、0.99：

Factor	Loss	Top 1 acc	Top 5 acc
0.1	**0.0196**	**0.4604**	**0.7210**

Factor	Loss	Top 1 acc	Top 5 acc
0.2	0.0212	0.4329	0.6931
0.5	0.0214	0.4220	0.6794
0.99	0.0220	0.4358	0.6891

在训练网络时，应该动态调节学习率。初始学习率可以略大来加快收敛速度，在接近极小值时，如果不减小学习率，会产生震荡而无法收敛。

2.4 网络深度

固定其他参数，调整网络深度为以下层数：

```
11层: [5x5 conv,64],[Layer(64,64)],[Layer(64,128)],[Layer(128,128)],
[Layer(128,256)],[Layer(256,256)]
15层: [5x5 conv,64],[Layer(64,64)],[Layer(64,128)],[Layer(128,128)],
[Layer(128,256)],[Layer(256,256)],[Layer(256,512)],[Layer(512,512)]
19层: [5x5 conv,64],[Layer(64,64)],[Layer(64,128)],[Layer(128,128)],
[Layer(128,256)],[Layer(256,256)],[Layer(256,512)],[Layer(512,512)],
[Layer(512,1024)],[Layer(1024,1024)]
```

深度	Loss	Top 1 acc	Top 5 acc
11层	**0.0196**	**0.4604**	**0.7210**
15层	0.0207	0.4322	0.6880
19层	0.0205	0.4231	0.6815

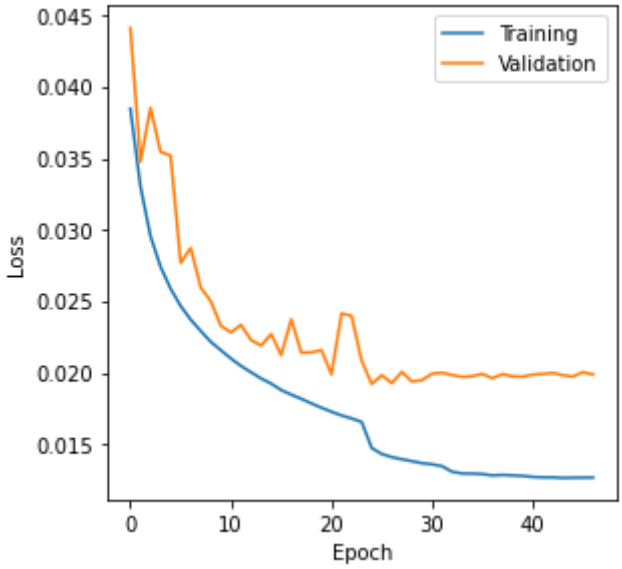
网络越深，性能不一定越好。更深层的网络更难优化，更容易陷入局部极小值。

3. 测试集效果

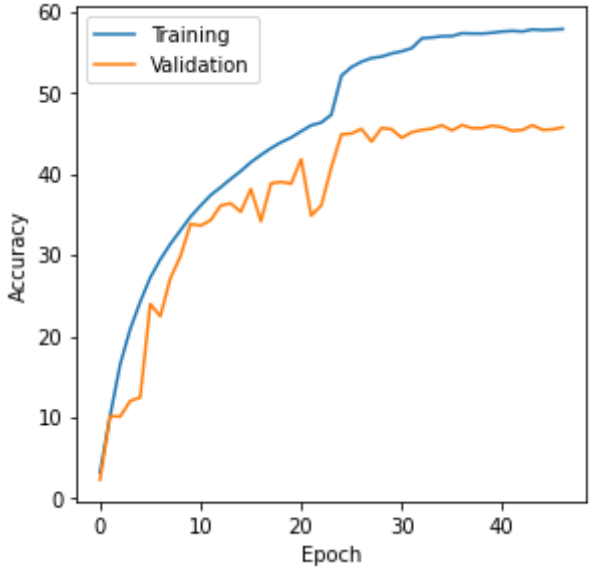
选用在验证集上表现最佳的网络来测试，即：

```
ResNet 11: [5x5 conv,64],[Layer(64,64)],[Layer(64,128)],[Layer(128,128)],
[Layer(128,256)],[Layer(256,256)]
```

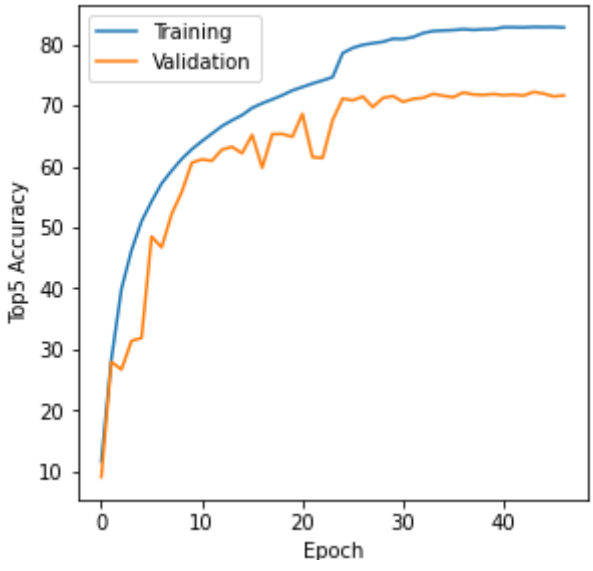
Loss曲线：



正确率曲线：



Top5-正确率曲线：



测试集结果：

```
Test Loss:  0.0198  |  Test Accuracy: 46.0700  |  Test Top5Accuracy: 71.7300
```

六、实验总结

通过本次实验，熟悉了Pytorch的使用与卷积网络的搭建，并通过调参过程理解各个参数的作用。