PJ2 Report

Chen Hao 22307110062

2025 年 5 月 26 日

摘要

Implementation can be found in the codes at `https://github.com/Laplx/DLPJ` and will also be gradually explained in the following answers. The *models* folder contains several model configurations, each corresponding to a specific question or modification and *best.pth* is the final selected optimal version with an accuracy of **93.58%** on the test set.

1 Task1: Train a Network on CIFAR-10

1.1 Basic Implementation

我们使用了 Bottleneck 和 Resnet 结构，通过降低计算复杂度更高效的表达特征，同时残差连接缓解了梯度消失的问题。

```
class Bottleneck(nn.Module):
  expansion = 4
  def __init__(self, in_planes, planes, stride=1):
    super(Bottleneck, self).__init__()
    self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=1,
        bias=False)
    self.bn1 = nn.BatchNorm2d(planes)
    self.conv2 = nn.Conv2d(planes, planes, kernel_size=3,
        stride=stride, padding=1, bias=False)
    self.bn2 = nn.BatchNorm2d(planes)
    self.conv3 = nn.Conv2d(planes, planes * self.expansion,
        kernel_size=1, bias=False)
    self.bn3 = nn.BatchNorm2d(planes * self.expansion)
```

```
class ResNet(nn.Module):
  def __init__(self, block, num_blocks, num_classes=10):
    super(ResNet, self).__init__()
    self.in_planes = 64
    self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1,
        bias=False)
    self.bn1 = nn.BatchNorm2d(64)
    self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1)
    self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
    self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
    self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)
    self.linear = nn.Linear(512 * block.expansion, num_classes)

def Net():
  return ResNet(Bottleneck, [3, 4, 6, 3])

model = Net().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=1e-4)
scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=100)
```

测试集准确率为 93.43%。

1.2 Different Configurations

- Different number of filters. 我们将全部通道数折半，同时训练 epoch 数由 100 缩减为 75，最终仍然几乎保持了测试集准确率（93.16%）。
- Different loss functions. （保持了通道数折半，）将损失函数换为对标签经过光滑处理的交叉熵，它对除本标签外的其他标签均匀分配以 $\frac{\epsilon}{K-1}$ 的概率，其中 K 为总标签数。或写为（其中 CE 为标准交叉熵函数）

$$\mathcal{L} = (1-\epsilon)\cdot \mathrm{CE}(y,p) + \epsilon \cdot \mathrm{CE}(u,p)$$

```
class LabelSmoothingCrossEntropy(nn.Module):
  def __init__(self, smoothing=0.1):
    super(LabelSmoothingCrossEntropy, self).__init__()
    self.smoothing = smoothing
```

```
        self.confidence = 1.0 - smoothing

    def forward(self, output, target):
        log_probs = F.log_softmax(output, dim=-1)
        n_classes = output.size(-1)
        true_dist = torch.zeros_like(log_probs)
        true_dist.fill_(self.smoothing / (n_classes - 1))
        true_dist.scatter_(1, target.data.unsqueeze(1),
            self.confidence)
        return torch.mean(torch.sum(-true_dist * log_probs, dim=-1))
```

测试集准确率达到了更优的水平，为 93.52%；猜测软化真实标签可能缓解了模型对硬标签的过拟合，提升泛化能力。

- Different activations. 我们将各层的激活函数由 RELU 换为了更为光滑的 GELU。（且此时可以发现 epoch 基本只需 20 - 30 个即已基本收敛到最优。）测试集准确率为 93.58%。激活函数本身形式没有造成特别大的改进或退化的影响。

- Different optimizers.

```
optimizer = optim.AdamW(model.parameters(), lr=0.001,
    weight_decay=0.01)
scheduler = optim.lr_scheduler.OneCycleLR(optimizer, max_lr=0.003,
    total_steps=100 * len(trainloader), pct_start=0.3)
```

测试集准确率反而下降到了 92.14%；可能是自适应学习率在数据噪声较大时梯度估计不稳定，或较高的初始学习率超出模型容忍范围。

1.3 Network Visualization

- Filters. 我们绘制了第一层各卷积核的权重图（按最大最小归一，可视化具体实现见代码）。

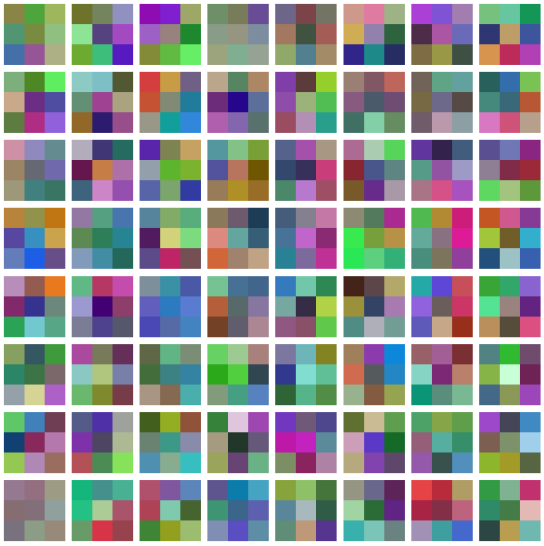

图 1: Filters of conv1

- Loss landscape. 我们通过在模型参数空间的两个随机方向上扰动参数并计算损失值，生成 3D 图形展示损失函数的变化趋势；可以看到虽然表面凹凸非常不平，但是整体又呈现显著的一个下降方向，这显示了神经网络优化问题的共性。

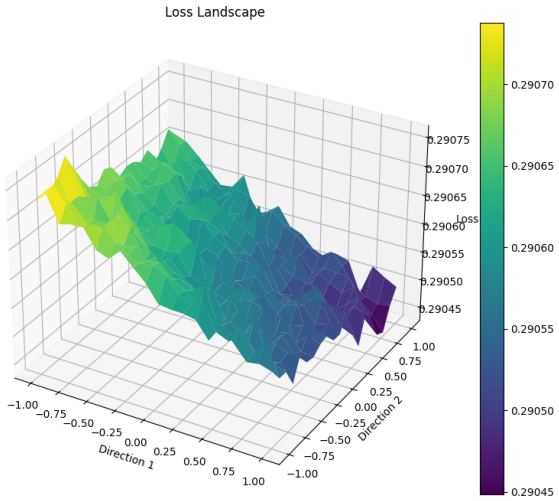

图 2

- Network interpretation. 我们通过 Grad-CAM（梯度加权类激活映射）的可视化，提取模型最后一个卷积层的特征和梯度，生成热力图来展示输入图像中对分类决策最重要的区域。可以看到网络学习明显到了各类别物体的所在和判断，并且深色处较为可信的聚焦在特征部位。

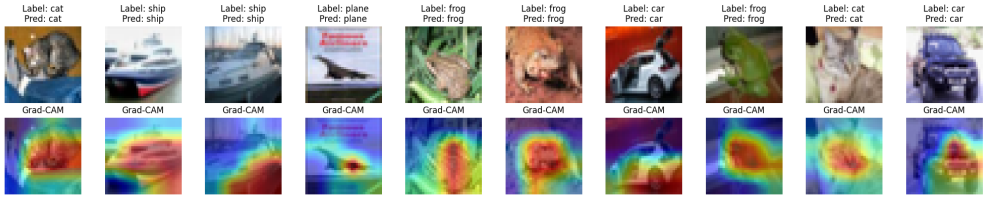

图 3: Grad-CAM visualization

2 Task2: Batch Normalization

2.1 VGG-A with and without BN

实现了 VGG-A 的归一化层之后我们可以画出两边训练的损失图景作比较。

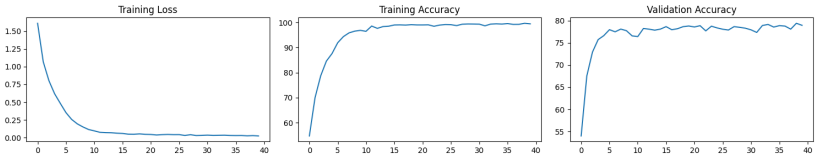

图 4: Training curve of VGG-A without BN at epoch 40

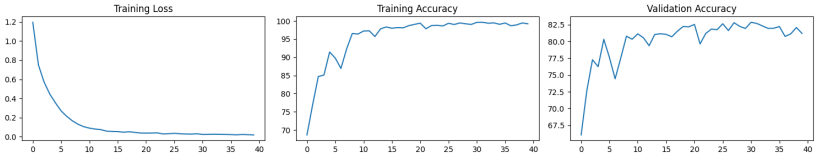

图 5: Training curve of VGG-A with BN at epoch 40

值得注意的是，尽管在后期包括最终的准确率上带归一层的网络胜出；但同样在第 40 个 epoch，有归一化层的网络反而情况不如没有的理想，一个猜测可能是归一层导致网络携带了更多的参数量，同时本身网络参数量可能已经略微过度，所以在最初相同批次下收敛速度不快。

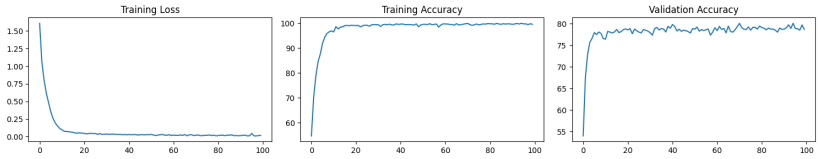

图 6: Training curve of VGG-A with BN at epoch 100

2.2 Loss Landscape

遵照文件里的指示，我们选取了 $[1e-3, 2e-3, 1e-4, 5e-4]$ 几个学习率分别对两种网络进行训练，以画出最终的损失景观的对比图。

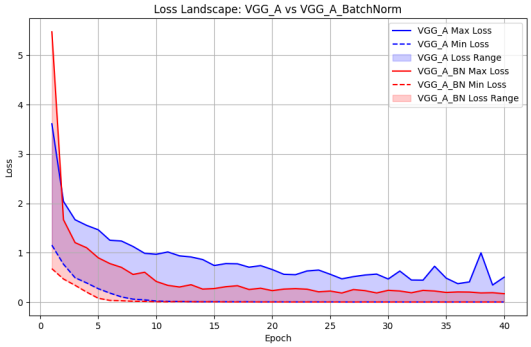

图 7

可以看出归一层的一个显著作用是控制了损失值的波动；但这里后段下界基本接近于 0 可能是由于网络拟合能力已经基本超出了 32 * 32 所需要的处理能力。