

浙江大学

ZHEJIANG UNIVERSITY



机器学习 实验报告

实验名称 Classify Leaves

实验地点 地点

姓 名 樊铨纬

学 号 3220104373

实验日期 December 12, 2024

指导老师 赵 洲、唐栎

Contents

1 实验目的和要求	1
2 数据集	1
3 训练过程与结果	2
4 准备阶段	5
5 第一阶段——自己搭建网络	8
5.1 增加 label smoothing	13
5.2 不同层使用不同的学习率	15
5.3 k-fold	16
6 第二阶段——使用预训练模型进行 fine tuning	17
6.1 使用 vit 和 v2e 模型	17
6.2 使用 timm 库进行训练	17
7 第三阶段——集成学习	21
8 感受	21
A 附录 1: 文档结构	22

1 实验目的和要求

Question.1. The task is predicting categories of leaf images. This dataset contains 176 categories, 18353 training images, 8800 test images. Each category has at least 50 images for training. The test set is split evenly into the public and private leaderboard.

The evaluation metric for this competition is Classification Accuracy.

2 数据集

`train.csv`: the training set `test.csv`: the test set `sample_submission.csv`: a sample submission file in the correct format (the labels are randomly generated so the initial accuracy is roughly 1/176) `images/`: the folder contains all images

Data fields: `image`: the image path, such as `images/0.jpg` `label`: the category name

使用 pandas 库的常见功能，我们可以查看数据集的基本信息。

数据集基本信息：

- 总样本数: 18353
- 总类别数: 176

样本数最多的前 10 个类别:

- `maclura_pomifera`: 353 个样本
- `ulmus_rubra`: 235 个样本
- `prunus_virginiana`: 223 个样本
- `acer_rubrum`: 217 个样本
- `broussonettia_papyrifera`: 214 个样本
- `prunus_sargentii`: 209 个样本
- `ptelea_trifoliata`: 193 个样本
- `ulmus_pumila`: 189 个样本
- `abies_concolor`: 176 个样本
- `asimina_triloba`: 174 个样本

样本数最少的前 10 个类别:

- aesculus_flava: 68 个样本
- betula_lenta: 68 个样本
- pinus_thunbergii: 67 个样本
- acer_griseum: 64 个样本
- ailanthus_altissima: 58 个样本
- cedrus_deodara: 58 个样本
- ulmus_procera: 58 个样本
- crataegus_crus-galli: 54 个样本
- evodia_daniellii: 53 个样本
- juniperus_virginiana: 51 个样本

类别分布统计:

- 平均每类样本数: 104.28
- 最大样本数: 353
- 最小样本数: 51
- 样本数标准差: 38.67

	image	label
count	18353	18353
unique	18353	176
top	images/18352.jpg	maclura_pomifera
freq	1	353

Table 1: 数据集基本统计信息

3 训练过程与结果

这次训练我租用了 Autodl 平台的两张 3090 显卡进行训练, 使用 pytorch 框架, 使用 AdamW 优化器, 使用交叉熵损失函数, 使用余弦退火学习率调度器, 使用 early stopping 防止过拟合。

Method	Epoch	Private Score	Public Score
resnet18 Naive	50	0.79522	0.79886
resnet18 with trick	50	0.95181	0.94977
resnet18.a1_in1k	50	0.94909	0.9484
resnet50d	90	0.88568	0.88363
resnet50d	30	0.92204	0.92113
resnet50d	48	0.96181	0.96022
resnet50d	50	0.9659	0.9625
mixnet_xl	50	0.96977	0.9634
inception_resnet_v2 label smoothing	50	0.96909	0.96204
inception_resnet_v2.tf_ens_adv_in1k	50	0.97181	0.96727
tf_mobilenetv3_small_minimal_100.in1k	50	0.93022	0.92727
mobilenetv2_100.ra_in1k	50	0.96113	0.95659
resnet101.a1h_in1k	33	0.96113	0.95886
resnet101.a1h_in1k	44	0.9709	0.96863
resnext	50	0.93318	0.93
seresnext50_32x4d.racm_in1k	20	0.95636	0.95409
seresnext50_32x4d.racm_in1k	33	0.9659	0.96113
seresnext50_32x4d.racm_in1k	42	0.96863	0.96636
seresnext50_32x4d.racm_in1k	50	0.96954	0.9675
seresnext50_32x4d.racm_in1k with finetuning	45	0.98204	0.97886
seresnext50_32x4d.racm_in1k with finetuning	50	0.9834	0.97795
选择使用正确率低的分类重新训练小模型	/	0.8834	0.8859
5 个 96 分学习器进行 bagging 集成, 投票法	/	0.9659	0.9625
4 个 97 分学习器进行 bagging 集成, 投票法	/	0.98704	0.98545

Table 2: 训练过程与结果










 submission.csv Complete (after deadline) · 1h ago · 使用4个97分学习器进行集成学习	0.98704	0.98545	<input type="checkbox"/>
 output.csv Complete (after deadline) · 3h ago · resnext50_32x4d.fb_sws_ligfb_in1k	0.98340	0.97795	<input type="checkbox"/>
 submission.csv Complete (after deadline) · 3h ago	0.98204	0.97886	<input type="checkbox"/>
 output.csv Complete (after deadline) · 3h ago · seresnext50 50epoch	0.96954	0.96750	<input type="checkbox"/>
 output.csv Complete (after deadline) · 3h ago	0.96863	0.96636	<input type="checkbox"/>
 output.csv Complete (after deadline) · 3h ago · seresnext50_32x4d.racm_in1k	0.96863	0.96636	<input type="checkbox"/>
 output_inception_resnet_v2.csv Complete (after deadline) · 4h ago · seresnext50 33epoch	0.96590	0.96113	<input type="checkbox"/>
 ensemble_predictions.csv Complete (after deadline) · 4h ago · 17个分类重新训练小模型	0.88340	0.88590	<input type="checkbox"/>
 output_re.csv Complete (after deadline) · 4h ago · seresnext50_32x4d.racm_in1k 20 epoch	0.95636	0.95409	<input type="checkbox"/>

Figure 1: Kaggle 平台截图










	ensemble_predictions.csv Complete (after deadline) · 6h ago · 5个96分学习器进行bagging集成	0.96590	0.96250	<input type="checkbox"/>
	output_inception_resnet_v2.csv Complete (after deadline) · 6h ago · inception_resnet_v2 50epoch	0.96886	0.96204	<input type="checkbox"/>
	output_inception_resnet_v2.csv Complete (after deadline) · 6h ago · inception_resnet_v2 label smoothing	0.96909	0.96204	<input type="checkbox"/>
	output_resnext.csv Complete (after deadline) · 7h ago · resnext	0.93318	0.93000	<input type="checkbox"/>
	output_resnet101.csv Complete (after deadline) · 8h ago · resnet101 epoch 44	0.97090	0.96863	<input type="checkbox"/>
	output_resnet101.csv Complete (after deadline) · 8h ago · resnet101_a1h_in1k epoch33	0.96113	0.95886	<input type="checkbox"/>
	output.csv Complete (after deadline) · 8h ago · resnet18_a1_in1k	0.94909	0.94840	<input type="checkbox"/>
	output2.csv Complete (after deadline) · 16h ago · mobilenetv2_300.ra_in1k	0.96113	0.95659	<input type="checkbox"/>
	output.csv Complete (after deadline) · 16h ago · tf_mobilenetv3_small_minimal_300_in1k	0.93022	0.92727	<input type="checkbox"/>

Figure 2: Kaggle 平台截图










	output.csv Complete (after deadline) · 17h ago · inception_resnet_v2_tf_ens_adv_in1k 50 epoch	0.97181	0.96727	<input type="checkbox"/>
	output.csv Complete (after deadline) · 18h ago · mixnet_xl pretrained model:50 epoch	0.96977	0.96340	<input type="checkbox"/>
	output.csv Complete (after deadline) · 20h ago · resnet50d	0.96590	0.96250	<input type="checkbox"/>
	output.csv Complete (after deadline) · 1d ago · resnet50d 49/50 epoch	0.96181	0.96022	<input type="checkbox"/>
	output.csv Complete (after deadline) · 1d ago · resnet50d 30/50 epoch	0.92204	0.92113	<input type="checkbox"/>
	output.csv Complete (after deadline) · 1d ago · resnet50d pretrained from timm [epoch] 90 lr begin with 0.001	0.88568	0.88363	<input type="checkbox"/>
	output.csv Complete (after deadline) · 1d ago · ERROR: resnet50 ,bugs in prediction code	0.00204	0.00181	<input type="checkbox"/>
	output.csv Complete (after deadline) · 3d ago	0.79522	0.79886	<input type="checkbox"/>
	output.csv Complete (after deadline) · 3d ago · resnet18 Naive n_split = 10	0.95181	0.94977	<input type="checkbox"/>

Figure 3: Kaggle 平台截图

4 准备阶段

Listing 1: 查看显卡信息

```
1 gpustat
```

Listing 2: 把图片分文件夹放好

```
1 import os
2 import shutil
3 import pandas as pd
4 import argparse
5
6 # 添加命令行参数解析
7 def parse_args():
8     parser = argparse.ArgumentParser(description='整理图片到训练和测试目录')
9     parser.add_argument('root_dir', type=str, help='项目根目录的路径')
10    return parser.parse_args()
11
12 # 全局变量现在需要通过函数来设置
13 def get_output_dirs(root_dir):
14     return {
15         'train': os.path.join(root_dir, 'img_train'),
16         'test': os.path.join(root_dir, 'img_test')
17     }
18
19 def setup_directories(dirs):
20     """创建必要的输出目录"""
21     for directory in dirs.values():
22         if not os.path.exists(directory):
23             os.makedirs(directory)
24
25 def process_train_images(dirs, train_csv_path='train.csv'):
26     """处理训练集图片"""
27     print("开始处理训练集图片...")
28
29     df_train = pd.read_csv(train_csv_path)
30
31     labels = df_train['label'].unique()
```

```
32     for label in labels:
33         label_dir = os.path.join(dirs['train'], label)
34         if not os.path.exists(label_dir):
35             os.makedirs(label_dir)
36
37     for index, row in df_train.iterrows():
38         src = row['image']
39         dst = os.path.join(dirs['train'], row['label'], os.path.basename(
40             src))
41
42         try:
43             shutil.copy2(src, dst)
44             print(f"复制 {src} 到 {dst}")
45         except FileNotFoundError:
46             print(f"警告: 找不到文件 {src}")
47         except Exception as e:
48             print(f"复制 {src} 时发生错误: {str(e)}")
49
50     print("训练集图片整理完成!")
51
52 def process_test_images(dirs, test_csv_path='test.csv'):
53     """处理测试集图片"""
54     print("\n开始处理测试集图片...")
55
56     df_test = pd.read_csv(test_csv_path)
57     test_images = df_test['image'].tolist()
58
59     for img_path in test_images:
60         try:
61             src = img_path
62             dst = os.path.join(dirs['test'], os.path.basename(img_path))
63
64             shutil.copy2(src, dst)
65             print(f"复制 {src} 到 {dst}")
66         except FileNotFoundError:
67             print(f"警告: 找不到文件 {src}")
68         except Exception as e:
69             print(f"复制 {src} 时发生错误: {str(e)}")
```



```
69
70     print("测试集图片整理完成!")
71
72 def main():
73     """主函数"""
74     args = parse_args()
75     output_dirs = get_output_dirs(args.root_dir)
76
77     setup_directories(output_dirs)
78     process_train_images(output_dirs)
79     process_test_images(output_dirs)
80
81 if __name__ == "__main__":
82     main()
```

```
1 train_df.describe()
```

5 第一阶段——自己搭建网络

这个数据集是李沐老师《动手学深度学习》课程中的一个数据集，正好之前也看过这门课程前面的一些内容，所以最开始的时候，我打算自己搭建一个网络，然后使用这个数据集进行训练。

Listing 3: 残差神经网络的 Baseline

```
1 class Residual(nn.Module):
2     def __init__(self, input_channels, num_channels, use_1x1conv=False,
3         strides=1):
4         super().__init__()
5         self.conv1 = nn.Conv2d(input_channels, num_channels,
6             kernel_size=3, padding=1, stride=strides)
7         self.conv2 = nn.Conv2d(num_channels, num_channels,
8             kernel_size=3, padding=1)
9         if use_1x1conv:
10             self.conv3 = nn.Conv2d(input_channels, num_channels,
11                 kernel_size=1, stride=strides)
12         else:
13             self.conv3 = None
14         self.bn1 = nn.BatchNorm2d(num_channels)
15         self.bn2 = nn.BatchNorm2d(num_channels)
16
17     def forward(self, X):
18         Y = F.relu(self.bn1(self.conv1(X)))
19         Y = self.bn2(self.conv2(Y))
20         if self.conv3:
21             X = self.conv3(X)
22             Y += X
23         return F.relu(Y)
24
25 class CustomResNet(nn.Module):
26     def __init__(self, num_classes=10):
27         super().__init__()
28         # 修改第一层以接受3通道RGB图像
29         self.b1 = nn.Sequential(
30             nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3),
31             nn.BatchNorm2d(64),
32             nn.ReLU(),
```

```
32         nn.Dropout(0.1),
33         nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
34     )
35
36     # 构建残差块
37     self.b2 = self._make_layer(64, 64, 2, first_block=True)
38     self.b3 = self._make_layer(64, 128, 2)
39     self.b4 = self._make_layer(128, 256, 2)
40     self.b5 = self._make_layer(256, 512, 2)
41
42     self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
43     self.flatten = nn.Flatten()
44     self.dropout = nn.Dropout(0.5)
45     self.fc = nn.Linear(512, num_classes)
46
47     def _make_layer(self, input_channels, num_channels, num_residuals,
48                     first_block=False):
49         layers = []
50         for i in range(num_residuals):
51             if i == 0 and not first_block:
52                 layers.append(Residual(input_channels, num_channels,
53                                         use_1x1conv=True, strides=2))
54             else:
55                 layers.append(Residual(num_channels, num_channels))
56         return nn.Sequential(*layers)
57
58     def forward(self, x):
59         x = self.b1(x)
60         x = self.b2(x)
61         x = self.b3(x)
62         x = self.b4(x)
63         x = self.b5(x)
64         x = self.avgpool(x)
65         x = self.flatten(x)
66         x = self.dropout(x)
67         x = self.fc(x)
68         return x
```

Listing 4: 训练代码

```
1  # 创建模型
2  model = CustomResNet(num_classes=num_classes)
3  model = model.to(device)
4
5  # 定义损失函数和优化器
6  criterion = nn.CrossEntropyLoss()
7  # 使用SGD优化器替代Adam
8  # optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate,
9  #                               momentum=0.9, weight_decay=5e-4)
10 optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
11 # 添加学习率调度器
12 scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=
13   num_epochs)
14
15 # 记录最佳验证准确率和对应的模型路径
16 best_val_acc = 0.0
17 best_model_path = None
18
19 # 初始化记录列表
20 train_losses = []
21 train_accs = []
22 val_losses = []
23 val_accs = []
24
25 # 训练循环
26 for epoch in range(num_epochs):
27     model.train()
28     epoch_train_loss = 0
29     train_correct = 0
30     train_total = 0
31
32     # 训练阶段
33     train_pbar = tqdm(train_loader, desc=f'Epoch [{epoch+1}/{num_epochs}]
34         训练')
35     for images, labels in train_pbar:
36         images, labels = images.to(device), labels.to(device)
```

```
35     optimizer.zero_grad()
36     outputs = model(images)
37     loss = criterion(outputs, labels)
38     loss.backward()
39     optimizer.step()
40
41     epoch_train_loss += loss.item()
42     _, predicted = outputs.max(1)
43     train_total += labels.size(0)
44     train_correct += predicted.eq(labels).sum().item()
45
46     train_pbar.set_postfix({
47         'loss': f'{epoch_train_loss/train_total:.4f}',
48         'acc': f'{100.*train_correct/train_total:.2f}%'
49     })
50
51     # 计算训练集平均损失和准确率
52     epoch_train_loss = epoch_train_loss / len(train_loader)
53     epoch_train_acc = 100. * train_correct / train_total
54
55     # 验证阶段
56     model.eval()
57     epoch_val_loss = 0
58     val_correct = 0
59     val_total = 0
60
61     val_pbar = tqdm(val_loader, desc='验证中')
62     with torch.no_grad():
63         for images, labels in val_pbar:
64             images, labels = images.to(device), labels.to(device)
65             outputs = model(images)
66             loss = criterion(outputs, labels)
67
68             epoch_val_loss += loss.item()
69             _, predicted = outputs.max(1)
70             val_total += labels.size(0)
71             val_correct += predicted.eq(labels).sum().item()
72
```

```
73         val_pbar.set_postfix({
74             'loss': f'{epoch_val_loss/val_total:.4f}',
75             'acc': f'{100.*val_correct/val_total:.2f}%'
76         })
77
78     # 计算验证集平均损失和准确率
79     epoch_val_loss = epoch_val_loss / len(val_loader)
80     epoch_val_acc = 100. * val_correct / val_total
81
82     # 记录损失和准确率
83     train_losses.append(epoch_train_loss)
84     train_accs.append(epoch_train_acc)
85     val_losses.append(epoch_val_loss)
86     val_accs.append(epoch_val_acc)
87
88     # 只在验证准确率更高时保存最佳模型
89     if epoch_val_acc > best_val_acc:
90         # 如果已存在之前的最佳模型文件，则删除它
91         if best_model_path and os.path.exists(best_model_path):
92             os.remove(best_model_path)
93
94         best_val_acc = epoch_val_acc
95         best_model_path = os.path.join(save_dir, f'best_model_acc_{
96             epoch_val_acc:.2f}.pth')
97         torch.save({
98             'epoch': epoch + 1,
99             'model_state_dict': model.state_dict(),
100             'optimizer_state_dict': optimizer.state_dict(),
101             'val_accuracy': epoch_val_acc,
102             'train_loss': epoch_train_loss,
103             'val_loss': epoch_val_loss
104         }, best_model_path)
105         print(f'保存最佳模型到: {best_model_path}')
106
107     # 绘制并保存训练曲线
108     plot_training_curves(
109         train_losses, train_accs,
110         val_losses, val_accs,
```

```

110     save_dir, 'training_curves'
111 )
112
113 print(f'Epoch [{epoch+1}/{num_epochs}]')
114 print(f'训练损失: {epoch_train_loss:.4f}, 训练准确率: {
    epoch_train_acc:.2f}%')
115 print(f'验证损失: {epoch_val_loss:.4f}, 验证准确率: {epoch_val_acc:.2
    f}%')
116 print('-----')
117
118 # 更新学习率
119 scheduler.step()

```

这份代码里面是有一个 bug

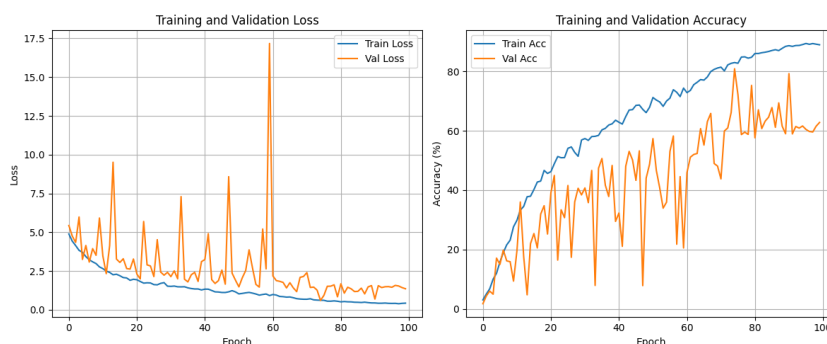


Figure 4: 第一次训练曲线

这份代码在 $lr=0.001$, $batch_size=64$, $num_epochs=100$, $save_dir='./checkpoints'$ 的配置下, 训练了 50 个 epoch, 验证集的准确率达到 85.5% 左右。

为了后续的实验, 我需要将这份代码封装成一个函数, 方便后续的实验。

所以我将这份代码的不同部分拆成了不同的 py 文件。

详见我的代码。

5.1 增加 label smoothing

标签平滑 (Label Smoothing) 是一种正则化技术, 用于防止模型过度自信。它通过将原始的 one-hot 标签转换为软标签来实现。

具体来说, 对于一个 K 类分类问题: - 传统的 one-hot 标签: 正确类别为 1, 其他类别为 0 - 标签平滑后: 正确类别为 $1 - \epsilon$, 其他类别为 $\epsilon / (K - 1)$, 其中 ϵ 是平滑系数

标签平滑的优点:

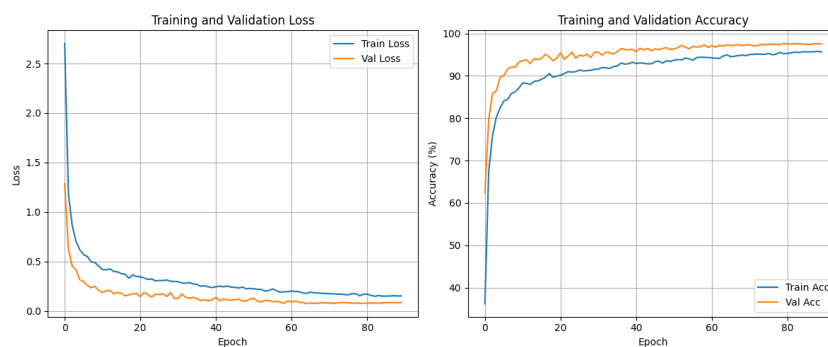


Figure 5: resnet50d 训练

- 防止模型过度自信, 提高泛化能力
- 增加模型的鲁棒性
- 缓解过拟合问题

在这个实验中, 使用了标签平滑技术, 平滑系数设为 0.1。这意味着:

- 正确类别的标签值从 1 变为 0.9
- 其他类别的标签值从 0 变为 0.1/175 (因为总共有 176 个类别)

Listing 5: 增加 label smoothing

```

1  class LabelSmoothingLoss(nn.Module):
2      """
3      标签平滑损失函数
4
5      参数:
6          smoothing (float): 平滑因子, 默认0.1表示使用0.1的平滑系数
7      """
8      def __init__(self, smoothing=0.1):
9          super().__init__()
10         self.confidence = 1.0 - smoothing
11         self.smoothing = smoothing
12
13     def forward(self, x: torch.Tensor, target: torch.Tensor) -> torch.
14         Tensor:
15         """
16         计算平滑后的损失值

```



```
17     参数:
18         x (torch.Tensor): 模型输出的logits
19         target (torch.Tensor): 目标标签
20
21     返回:
22         torch.Tensor: 计算得到的损失值
23     """
24     logprobs = F.log_softmax(x, dim=-1)
25     nll_loss = -logprobs.gather(dim=-1, index=target.unsqueeze(1))
26     nll_loss = nll_loss.squeeze(1)
27     smooth_loss = -logprobs.mean(dim=-1)
28     loss = self.confidence * nll_loss + self.smoothing * smooth_loss
29     return loss.mean()
30
31 def create_criterion(name="crossentropy", smoothing=0.1):
32     """
33     创建损失函数
34
35     参数:
36         name: 损失函数名称,支持 'crossentropy' 和 'labelsmoothing'
37         smoothing: 标签平滑系数,仅在使用labelsmoothing时有效
38
39     返回:
40         nn.Module: 损失函数
41     """
42     if name.lower() == "crossentropy":
43         return nn.CrossEntropyLoss()
44     elif name.lower() == "labelsmoothing":
45         return LabelSmoothingLoss(smoothing=smoothing)
46     else:
47         raise NotImplementedError(f"Loss function {name} not implemented")
```

5.2 不同层使用不同的学习率

- **针对性训练:** 对于预训练模型的微调, 通常最后几层需要更大的学习率以适应新任务, 而前面的层保持较小的学习率以保留预训练的特征
- **提高训练效率:** 通过为不同层设置合适的学习率, 可以加快模型收敛速度

- **防止过拟合**: 较小的学习率可以防止底层特征提取器过度适应新数据集
- **平衡迁移学习**: 在迁移学习中, 这种策略可以很好地平衡预训练知识的保留和新任务的适应

在本实验中, 我们对全连接层使用了 10 倍于其他层的学习率, 这是因为全连接层需要从头开始学习新的分类任务, 而其他层主要是微调预训练的特征。

Listing 6: 不同层使用不同的学习率

```
1 optimizer = torch.optim.AdamW([{'params': params},
2     {'params': model.fc.parameters(),
3         'lr': args.lr * 10}],
4     lr=args.lr, weight_decay=2e-4)
5
6 scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(
7     optimizer, T_max=args.epochs, eta_min=args.lr/20
8 )
```

5.3 k-fold

使用 StratifiedKFold 对数据集进行分层, 然后进行 k-fold 交叉验证, 详见训练记录

k-fold 交叉验证是一种评估机器学习模型性能的方法。它将数据集分成 k 个相等大小的子集, 每次使用 k-1 个子集作为训练集, 剩下的 1 个子集作为验证集。这个过程重复 k 次, 每个子集都会作为一次验证集。

在本实验中, 我使用了 StratifiedKFold 进行 5 折交叉验证。StratifiedKFold 是 k-fold 的一个变体, 它在划分数据时会**保持每个类别的样本比例**, 这对于不平衡的数据集特别有用。

k-fold 交叉验证的主要优点包括:

- 充分利用有限的的数据, 每个样本都会被用作训练和验证
- 得到更可靠的模型性能评估
- 减少过拟合的风险
- 评估模型的稳定性

具体实现时, 我使用了 sklearn 库中的 StratifiedKFold 类:

```
1 from sklearn.model_selection import StratifiedKFold
2
3 skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
4 for fold, (train_idx, val_idx) in enumerate(skf.split(X, y)):
5     train_data = dataset[train_idx]
6     val_data = dataset[val_idx]
7     # 训练模型...
```

6 第二阶段——使用预训练模型进行 fine tuning

由于第一阶段训练的模型效果并不理想，所以第二阶段我打算使用预训练模型进行 fine tuning。

首先我尝试了 pytorch 的 nn 库自带的预训练模型进行训练，详见训练记录



Figure 6: vit 训练失败，v2e 训练显存不够

6.1 使用 vit 和 v2e 模型

在实验中，我也尝试使用了 vit 和 v2e 模型，但是由于显卡的显存不够，训练时长已经超过了我的预算范围，所以只能放弃，希望之后可以有机会拿到更好的卡，进行实验。

vit 由于代码有一些 bug，训练了 20 个 epoch 左右，系统正确率一直在 20% 左右徘徊，中途我结束了训练

6.2 使用 timm 库进行训练

这里需要注意的是，因为我的显卡没有办法连接外网，所以 pretrained 的模型需要我手动下载，放到本地

timm 库是一个开源的深度学习库，全称为 PyTorch Image Models，它提供了大量的预训练模型和训练工具。主要特点包括：

- 提供了超过 500 个预训练模型，包括 ResNet、VGG、Inception 等经典模型
- 支持多种数据增强方法和训练技巧
- 提供了模型训练和评估的工具函数
- 支持模型导出和部署
- 与 PyTorch 生态系统完全兼容

在本次实验中, 我主要使用了 `timm` 库提供的预训练模型进行迁移学习。由于显卡无法连接外网, 我需要手动下载预训练模型权重文件并放到本地。

使用 `timm` 库的主要步骤如下:

- (1). 使用 `timm.create_model()` 创建模型
- (2). 加载预训练权重
- (3). 修改最后的全连接层以适应本任务的类别数
- (4). 进行模型训练

Listing 7: 使用 `timm` 库进行训练, 核心代码

```
1 model = timm.create_model('inception_resnet_v2.tf_ens_adv_in1k',  
    pretrained=False)  
2 model.load_state_dict(torch.load('result/mixnet_xl/pytorch_model.bin',  
    map_location=device))  
3 model.fc = nn.Linear(model.num_features, num_classes)  
4 print(model.fc.in_features)  
5 nn.init.xavier_uniform_(model.fc.weight)  
6 model.cuda()
```

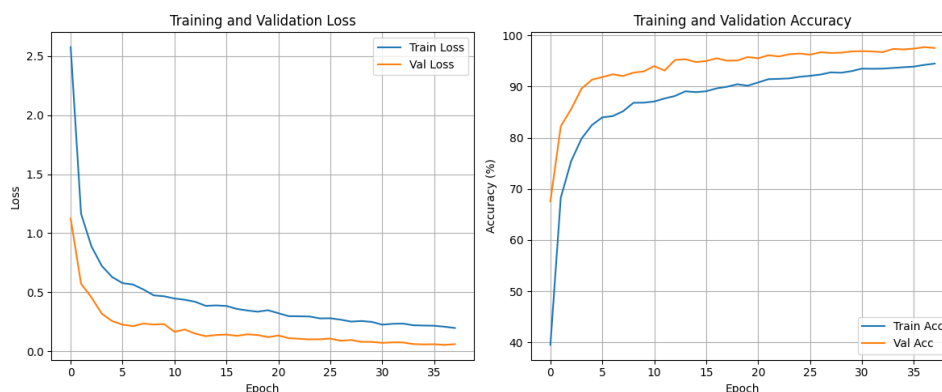


Figure 7: mixnet_xl score = 0.969

后续还有一些图片, 由于比较类似, 所以就不放了

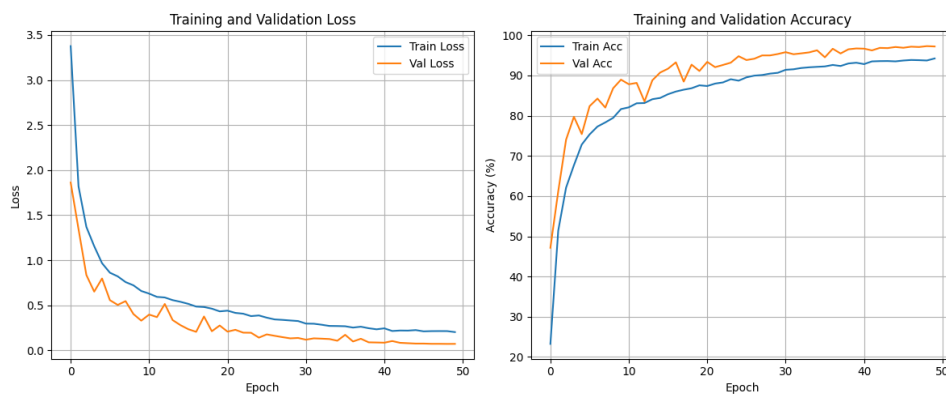


Figure 8: inception_resnet_v2.tf_ens_adv_in1k score 97.181

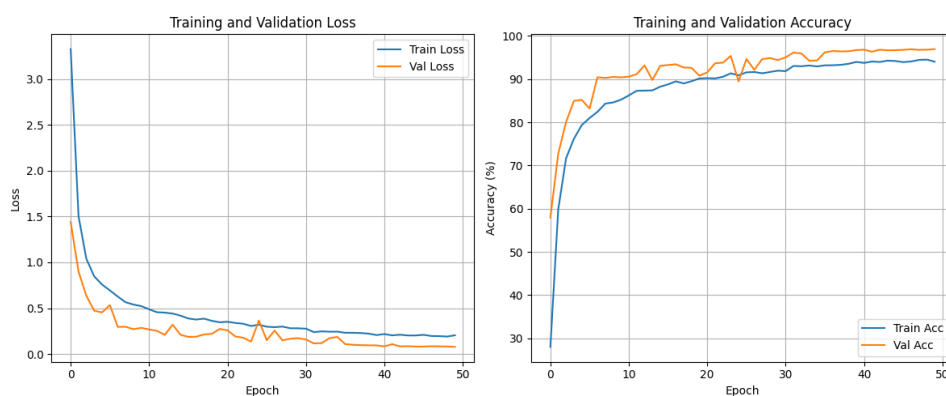


Figure 9: mobilenetv2_100.ra_in1k.bin score=96.133

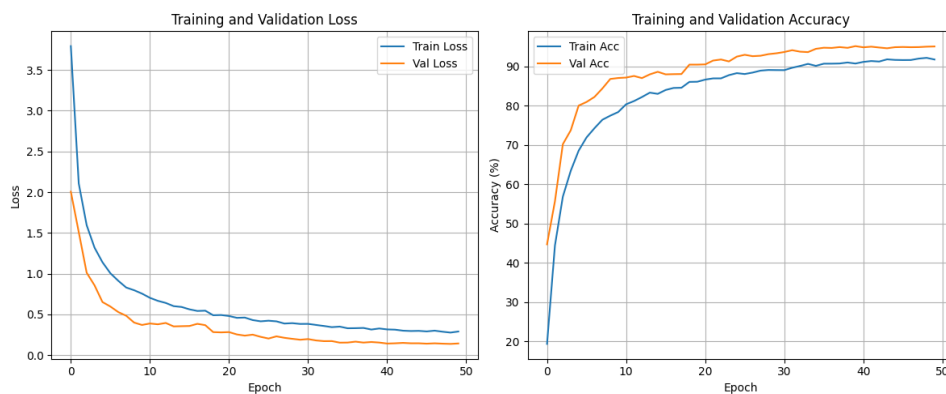


Figure 10: tf_mobilenetv3_small_minimal_100.in1k score=93.022

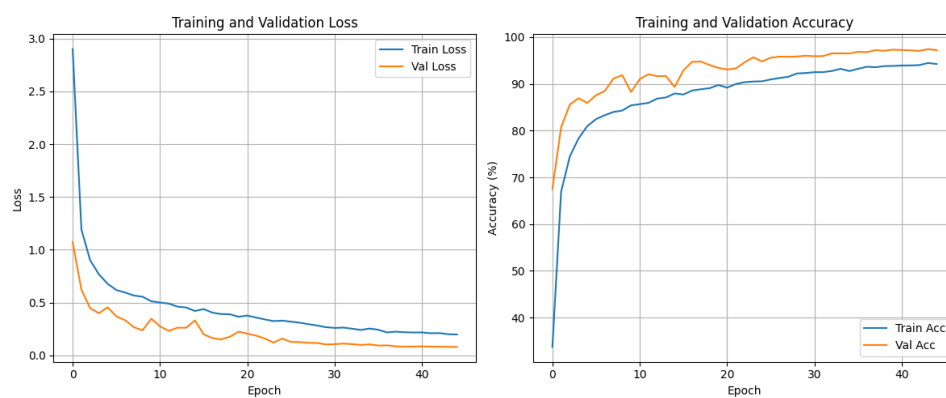


Figure 11: resnet101.a1_in1k epoch 33: score = 96.133 epoch 50: score = 97.090

7 第三阶段——集成学习

课上我们已经学过集成学习，这次实验我也尝试使用了集成学习进行训练。

首先我使用了当时使用 `inception_resnet_v2.tf_ens_adv_in1k` 模型训练出的 97 的模型进行统计，发现有一些分类正确率比较低，所以把这些小分类抽出来，单独训练了一个数据集。

但是训练的结果并不好。反而比单独的模型分数低了。

Listing 8: 使用前后两个模型进行训练

```
1 model2.fc = nn.Linear(model2.num_features, 17)
2 model2.load_state_dict( torch.load('result/20241212_143856/0_best_model.
  pth', map_location=device)['state_dict'])
3 model2 = model2.to(device)
4 model2.eval()
```

所以使用了 bagging 的方法，采用简单投票法进行实验。

最开始我使用 5 个 96 分左右的学习器进行预测，最后的分数并不高，没有超过 97

后来我调整了我的图像预处理，使用数据增强，使用 dropout，使用 label smoothing 后，训练出了 4 个 97 分左右的学习器，使用这四个 97 分的学习器进行预测，最后的分数超过了 98。

后续调整了 batchsize 继续训练，给 4 个模型按照原始分数给予不同的权重，最后分数达到了 98。

8 感受

经过连续 3 天的实验，我深刻感受到了调参的不易，以及数据集的重要性。

有些时候，花了很多时间加了很多 trick，分数不增反降，我也意识到，对于深度学习模型，我需要学习的内容还有很多。

通过这次实践的训练，我更加了解了 pytorch 进行训练的基础步骤，以及 timm 和 huggingface 的使用，更重要的是我积累了一些使用服务器和显卡进行训练的经验，这对后续的研究很有帮助。

有关使用服务器和 pytorch 的笔记，我记录在了[个人网站](#)上面

A 附录 1: 文档结构

```
├── code
│   ├── 00-copy_hard_samples.py
│   ├── 00-count_images.py
│   ├── 00-info.py
│   ├── 00-organize_images.py
│   ├── 00-test_gpu.py
│   ├── 01-dataset.py
│   ├── 01-models.py
│   ├── 01-Resnet50.py
│   ├── 02-train2.py
│   ├── 02-train.py
│   ├── 02-train_utils.py
│   ├── 03-ensemble_predict.py
│   └── 03-predict.py
├── final_model
│   ├── inception_resnet_v2.tf_ens_adv_in1k.pth
│   ├── mixnet_xl.pth
│   ├── mobilenetv2_100.ra_in1k.pth
│   ├── resnet101.a1h_in1k.pth
│   ├── resnet50d.pth
│   ├── resnext50_32x4d.fb_sws1_ig1b_ft_in1k_2.pth
│   ├── resnext50_32x4d.fb_sws1_ig1b_ft_in1k.pth
│   └── seresnext50_32x4d.racm_in1k.pth
├── model
│   ├── convnext_tiny.in12k_ft_in1k.bin
│   ├── convnextv2_large.fcmae_ft_in22k_in1k.bin
│   ├── eva02_large_patch14_448.mim_m38m_ft_in22k_in1k.bin
│   ├── inception_resnet_v2.tf_ens_adv_in1k.bin
│   ├── mixnet_xl.bin
│   ├── mobilenetv2_100.ra_in1k.bin
│   ├── mobilenetv3_small_minimal_100.in1k.bin
│   ├── resnet101.a1h_in1k.bin
│   ├── resnet18.a1_in1k.bin
│   ├── resnet50d_ra2-464e36ba.pth
│   ├── resnext50_32x4d.fb_sws1_ig1b_ft_in1k.bin
│   ├── seresnext50_32x4d.racm_in1k.bin
│   └── vit_base_patch16_224.augreg2_in21k_ft_in1k.bin
├── README.md
├── requirements.txt
├── Makefile
├── result
│   ├── 20241209_152338
│   │   ├── best_model_acc_94.93.pth
│   │   └── exp.log
```

- └─ final_model.pth
- └─ training_curves.png
- 20241209_163042
 - └─ best_model.pth
 - └─ checkpoint.pth
 - └─ exp.log
 - └─ training_curves.png
- 20241209_172359
 - └─ exp.log
 - └─ training_curves.png
- 20241211_153734
 - └─ best_model.pth
 - └─ checkpoint.pth
 - └─ exp.log
 - └─ training_curves.png
- 20241211_164351
 - └─ best_model.pth
 - └─ checkpoint.pth
 - └─ exp.log
 - └─ training_curves.png
- 20241211_235611
 - └─ best_model.pth
 - └─ checkpoint.pth
 - └─ exp.log
 - └─ training_curves.png
- 20241212_001640
 - └─ best_model.pth
 - └─ checkpoint.pth
 - └─ exp.log
 - └─ training_curves.png
- 20241212_013554
 - └─ exp.log
 - └─ training_curves.png
- 20241212_020242
 - └─ 0_best_model.pth
 - └─ exp.log
 - └─ training_curves.png
- 20241212_021324
 - └─ exp.log
 - └─ training_curves.png
- 20241212_110329
 - └─ 0_best_model.pth
 - └─ exp.log
 - └─ training_curves.png
- 20241212_111033
 - └─ exp.log
 - └─ training_curves.png

```
├── 20241212_142840
│   ├── exp.log
│   └── training_curves.png
├── 20241212_143214
│   ├── 0_best_model.pth
│   ├── 0_checkpoint.pth
│   ├── exp.log
│   └── training_curves.png
├── 20241212_143856
│   ├── exp.log
│   └── training_curves.png
├── model.csv
├── resnet101.a1h_in1k
│   ├── exp.log
│   └── training_curves.png
├── resnet18.a1_in1k
│   ├── exp.log
│   └── training_curves.png
├── test.csv
└── train.csv
```