浙江大学

本科实验报告

Learning CNN++

课程名称：计算机视觉

姓　　名：秦雨扬

学　　院：竺可桢学院

专　　业：自动化（控制）

学　　号：3210104591

电　　话：13588789194

邮　　件：qinyuyang2003@zju.edu.cn

指导老师：潘纲

2024 年 1 月 23 日

Contents

List of Figures

浙江大学实验报告

专业：自动化（控制）
姓名：秦雨扬
学号：3210104591
日期：2024 年 1 月 23 日
地点：玉泉曹光彪西-103

课程名称：计算机视觉　指导老师：潘纲　助教：周健均
实验名称：Learning CNN++　实验类型：综合　成绩：

I 功能简述及运行说明

1. 功能简述

在本实验主要为探索性实验，针对 Kaggle 公开数据集 Pokemon Image Dataset 上的精灵宝可梦类型预测任务展开。

- 为 Pokemon Image Dataset 预处理并自定义了数据集，同时实现了数据增广
- 设计了三个 model, 并对 MobileNet 进行了微调
- 对比了一些超参数

2. 运行说明

- 总程序为 *prepare.py*(本来只想写个数据预处理一不小心写完了整个 project)
- 自行设计的三个模型在 *prepare.py* 而 MobileNet 在 *MobileNet.py*，注意使用 MobileNet 对应的主程序为 *prepare_for_mobilenet.py* ，
- 运行后会依次输出 crossentropy 中的 weight,type name 与其对应的数字编码，每个 Pokemon 对应的两个 type 的数字编码, 缺失值为 nan
- 之后会输出训练集与测试集的划分，以及每个训练的 loss 和 Acc
- 训练时每个 epoch 会分别记录 train 和 validation 的 accuarcy 和 loss, 将会打印并存在 runs 文件夹下，可使用 tensorboard logdir=runs 可视化查看。

所有本文涉及到的训练数据，均在 run_results 文件夹下，可以使用 tensorboard 查看，因此未注明精度细节。

II 开发与运行环境

实验使用 python 语言, 测试系统为 *Ubuntu 20.04.6 LTS*, 测试环境为 *Anaconda 23.7.3* 下的 *python 3.8.18* ，其它使用的包分别为

1) tensorboard 2.14.0
2) torch 2.0.1+cu118
3) torchvision 0.15.2+cu118
4) matplotlib 3.7.4

III 算法原理与具体实现

1. 数据预处理

1.1 Label

首先将所有 type1 与 type2 都放入在同一个 list 中，从而可以得到每个 type name 的数量。使用先 set 在 list 进行 unique, 从而得到 type 到编码数字的映射关系, 其中 nan 也作为一类。

基于此映射，我们对每个 Pokemon 得到了一个二元 list, 即为其 label。

而在 cross entropy 中有一个 weight 选项，本实验中我们尝试了使用

$$w_i = \frac{1}{numbers\ of\ this\ type}$$

并通过线性映射实现 w_i 的均值为 1，使用这个作为 binary cross entropy 的 weight，效果见实验结果与分析。

```
## prepare data
df = pd.read_csv('pokemon.csv')

labels0 = df['Type1'].tolist()
labels0.extend(df['Type2'].tolist())
labels=list(set(labels0))
labels_num=len(labels)
dict2str={}
dict2int={}
en_weight=torch.zeros(labels_num,device=device)
for i,x in enumerate(labels):
    dict2str[i]=x
    dict2int[x]=i
    # en_weight[i]=1.0/labels0.count(x)
    en_weight[i]=1.0
en_weight=en_weight/en_weight.sum()*labels_num
print(en_weight)
print(dict2int)
NONE=0
while(not pd.isna(dict2str[NONE])):
    NONE+=1
print(NONE)
# en_weight[NONE]=0
dict_all={}
# read df line by line
for index, row in df.iterrows():
    # print(row['Type1'],row['Type2'])
    # print(dict2int[row['Type1']],dict2int[row['Type2']])
    dict_all[row['Name']]=[dict2int[row['Type1']],dict2int[row['Type2']]]
print(dict_all)
```

1.2 数据增强

在读取图像时，发现竟然有些图像时 3 维的有的图像是 4 维的，最离谱的是有一维的彩色图像，后来查阅资料 [1][2] 终于搞明白 1，3，4 维分别是 LodePNG，RGB,RGB-A 三种图像格式，解决方案是.convert('RGB')

这里主要运用了三个函数 RandomRotation，RandomResizedCrop，ColorJitter, 分别对旋转，大小，色彩等方面做了调整，当然最后也要也 toTensor。最终的大小保持 120 不变。

在 MobileNet 中，我们需要 resize 到 224，并且归一化到 [0.485, 0.456, 0.406], [0.229, 0.224, 0.225]，这是 mobilenet_v3_large 的 model card 里面写明的。

1.3 自定义数据集

我们现在虽然解决了 Label, 可以通过 pandas 读取到了 train 和 test dataset，但是直接使用 DataLoader 自然是不行。后来通过官网文档 [3] 发现可以自定义 dataset。

于是我自定义了 PokemonImageDataset，其中会使用 PIL 读取对应的图片，做前述的数据增强, 最终返回 tensor 作为 input

而在 label 方面，我定义了一个 type num 长度的全零 tensor, 对二元 list 中的每一个的对应位置 1。从而实现 binary cross entropy。

此外由于使用了数据增强，我们这里设置的 repeat 的参数，使得数据量得到扩充。

```
69  class PokemonImageDataset(Dataset):
70      def __init__(self, data,img_dir,label_dict,is_transform=None,repeat=1):
71          self.repeat=repeat
72          self.data=data
73          self.img_dir = img_dir
74          self.label_dict=label_dict
75          height=width=120
76          self.transform =T.Compose([
77              # T.Resize([232]),
78              T.RandomRotation(90,center=(height/2,width/2)),
79              T.RandomResizedCrop((height, width),scale = (0.7,1.0)),
80              T.ColorJitter(brightness=0.5, contrast=0.5, saturation=0.5, hue=0.5),
81              # T.CenterCrop([224]),
82              # T.Random
83              # T.RandomHorizontalFlip(0.1),  # 进行随机水平翻转
84              # T.RandomVerticalFlip(0.1),  # 进行随机竖直翻转
85              T.ToTensor(),  # 转化为张量
86              # T.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]),  # 归一化
87          ]) if is_transform else T.Compose([
88              T.Resize([120]),
89              # T.RandomRotation(15,center=(height/2,width/2)),
90              # T.RandomResizedCrop((height, width),scale = (0.7,1.0)),
91              # T.ColorJitter(brightness=0.5, contrast=0.5, saturation=0.5, hue=0.5),
92              # T.CenterCrop([224]),
93              # T.Random
94              # T.RandomHorizontalFlip(0.1),  # 进行随机水平翻转
95              # T.RandomVerticalFlip(0.1),  # 进行随机竖直翻转
96              T.ToTensor(),  # 转化为张量
97              # T.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]),  # 归一化
98          ])
```

```
99
100     def __len__(self):
101         return len(self.data)*self.repeat
102 
103     def __getitem__(self, idx):
104         idx=idx%len(self.data)
105         img_path = os.path.join(self.img_dir, self.data[idx]+'.png')
106         if(not os.path.isfile(img_path)):
107             img_path = os.path.join(self.img_dir, self.data[idx]+'.jpg')
108             # print("!")
109 
110         # image=read_image(img_path)
111         image = Image.open(img_path).convert('RGB')
112         if self.transform:
113             image = self.transform(image)
114         # print(image.size(0))
115         # if(image.size(0)==4):
116         #     print(self.data[idx])
117         #     image=image[:3,:,:]
118         label0 = self.label_dict[self.data[idx]]
119         # print(label0)
120         label=torch.zeros(labels_num)
121         label[label0[0]]=1
122         label[label0[1]]=1
123         # label[NONE]=0
124         # if self.transform:
125         #     image = self.transform(image)
126         return image, label
```

2. 模型

这里首先按照 [4] 中相同的结构，完成了 model1。然后在训练过程中发现出现了过拟合，于是就将 max-pooling 变大，并稍微去掉了几层，得到了 model2，但是发现仍然具有过拟合，边又改小了 feature number 得到了 model3, 但是模型的精度始终不太理想。

对于 MobileNet, 这里采用了 mobilenet_v3_large, 由于采用 finetune, 这里冻结了 backbone 的参数。

```
129 class myNet(nn.Module):
130     def __init__(self, classes=2):
131         super(myNet, self).__init__()
132 
133         # self.model1= nn.Sequential(
134         #     nn.Conv2d(3, 16, (3, 3)),
135         #     nn.BatchNorm2d(16),
136         #     nn.ReLU(),
137         #     nn.MaxPool2d((2, 2)),
138         #
139         #     nn.Conv2d(16, 32, (3, 3)),
140         #     nn.BatchNorm2d(32),
141         #     nn.ReLU(),
```

```
        #     nn.MaxPool2d((2, 2)),
        #
        #     nn.Conv2d(32, 64, (3, 3)),
        #     nn.BatchNorm2d(64),
        #     nn.ReLU(),
        #     nn.MaxPool2d((2, 2)),
        #
        #     nn.Conv2d(64, 128, (3, 3)),
        #     nn.BatchNorm2d(128),
        #     nn.ReLU(),
        #     nn.MaxPool2d((2, 2)),
        #
        #     nn.Conv2d(128, 150, (3, 3)),
        #     nn.BatchNorm2d(150),
        #     nn.ReLU(),
        #     nn.MaxPool2d((2, 2)),
        #
        #     nn.Flatten(),
        #     nn.Linear(150,64),
        #     nn.ReLU(),
        #     nn.Linear(64,classes),
        #     nn.Softmax()
        # )
        # self.model2= nn.Sequential(
        #     nn.Conv2d(3, 16, (3, 3)),
        #     nn.BatchNorm2d(16),
        #     nn.ReLU(),
        #     nn.MaxPool2d((4, 4)),
        #
        #     nn.Conv2d(16, 64, (3, 3)),
        #     nn.BatchNorm2d(64),
        #     nn.ReLU(),
        #     nn.MaxPool2d((4, 4)),
        #
        #     nn.Conv2d(64, 64, (3, 3)),
        #     nn.BatchNorm2d(64),
        #     nn.ReLU(),
        #     # nn.MaxPool2d((4, 4)),
        #
        #     nn.Flatten(),
        #     nn.Linear(1024,classes),
        #     nn.Softmax()
        # )
        self.model3= nn.Sequential(
            nn.Conv2d(3, 8, (3, 3)),
            nn.BatchNorm2d(8),
            nn.ReLU(),
            nn.MaxPool2d((4, 4)),

```

```
            nn.Conv2d(8, 16, (3, 3)),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d((4, 4)),

            nn.Conv2d(16, 32, (3, 3)),
            nn.BatchNorm2d(32),
            nn.ReLU(),

            nn.Flatten(),
            nn.Linear(512,classes),
            nn.Softmax()
        )

        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(m.weight, mode="fan_out")
                if m.bias is not None:
                    nn.init.zeros_(m.bias)
            elif isinstance(m, (nn.BatchNorm2d, nn.GroupNorm)):
                nn.init.ones_(m.weight)
                nn.init.zeros_(m.bias)
            elif isinstance(m, nn.Linear):
                nn.init.normal_(m.weight, 0, 0.1)
                nn.init.zeros_(m.bias)

    def forward(self, x):
        x = self.model3(x)
        return x
```

3. 训练与评估

这一块就是普通的训练，除了前述的 binary cross entropy 也没有特殊之处，但是这里值得一提的是**精度**的计算

若 type2 为空, 则 type1 是否正确占比 100%, 否则 type2 与 type1 是否正确各占 50%。这个自定义的精度应当是比较准确的，但是也导致精度较为难看。

此外，binary cross entropy 之前 output 需要经过 softmax

```
loss = criterion(pred_y, y,weight=en_weight)
pred_y = torch.sort(pred_y, dim=1,descending=True)[1]
for y1,y2 in zip(pred_y,y):
    if y1[1]==NONE:
        total_acc1+=y2[y1[0]]
    elif y1[0]==NONE:
        total_acc1+=y2[y1[1]]
    else:
        total_acc1+=y2[y1[0]]*0.5+y2[y1[1]]*0.5
```

IV 实验结果与分析

1. 综述

对于本实验，由于数据集太小，导致模型泛化效果太差。而另一方面，对于这样小的数据集，预测一个 class 就比较困难，两个 class 更是雪上加霜。

在给的参考 [4] 中，他精度相对高的原因在于采用了 top k accuracy[5]。这里 k 取 2。但是仔细想想它的机制，就会发现其实是与题目要求南辕北辙的。top k accuracy 是指 output 最大的 k 个 class 中是否有一个值为 1，而并不是最大的两个都预测对了，实际上这中间还是有很大差别的。

所以这里认为本实验精度比较低是合理的，特别是不考虑正确预测 None 的情况下。

而且由于梯度较小，所以训练也精度不太高，这里我们应该放下精度，更加关心做了什么工作，学到了什么东西。

然后感觉这里主要还是聚焦了一些实践中遇到的和上课的知识点联系起来的部分展开，虽然训练的次数不少，但是专门做对比实验训练时间还是太长了一点。

2. 优化器比较

在之前的实验中，通常 Adam 会是相对好用的一个，但是在本实验中，发现 loss 下降的非常缓慢，lr 从 0.005,0.01,0.5 之间变化的时候, 发现 loss 的下降似乎没什么改变。后来忽然想起来上课讲的 momentum 的原理，想起来是不是可能因为梯度太小 Adam 以为到最低点了开始减小 momentum, 果不其然，换用了 SDG 后 loss 下降明显变快。这算是很好地应用了上课学到的知识。

结果如图1所示,SDG 对应的 loss 不断减小，训练精度不断提升，但是会发现这对于验证集的精度并没有什么帮助，个人感觉还是训练集太小，无法拟合如此大量的参数。

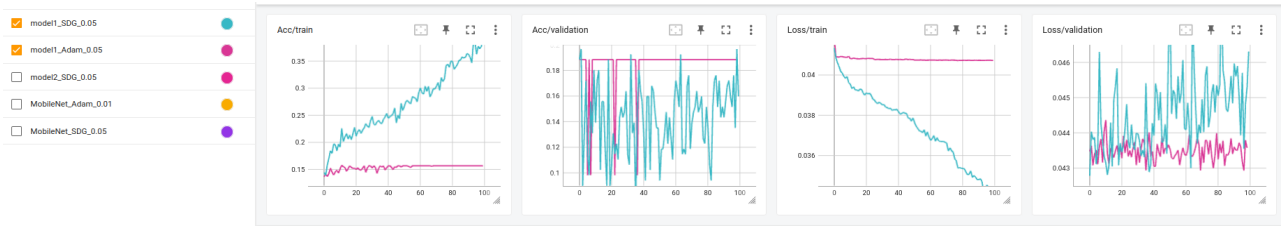

Figure 1: 优化器比较

3. 模型比较

实验结果如图2所示。我们先讨论自定义的三个模型，从复杂到简单，所以看训练精度的话可以看出其上限也不尽相同，越复杂的模型可以拟合的精度上限越大，也就是上课所讲的 bias 的问题。

从验证集精度上来看 variance 倒是还都挺大的，这个我个人觉得还是数据集太小的问题。不过不难发现 model2 其实是效果最好的，它在验证集上的精度最高的时候能够达到 23.36%，而且还反复出现多次 20% 以上，这里姑且认为它在 bias 与 variance 之间找到了一个比较好的平衡吧。

而对于 MobileNet，其 acc 与 loss 变化不明显，很可能是由于 finetune, 它最后两层 linear 很快就训练到饱和了，由于 backbone 不能修改，所以精度无法进一步提升。所以可以看到它在验证集上的精度就表现得非常稳定，且相对不错。

4. Batch Size

在课堂上我们学到小 batch size 虽然 noisy 但是具有更好的 Optimization 和 Generalization。而较大的 batchsize 会比较 stable。实验结果如图3所示。

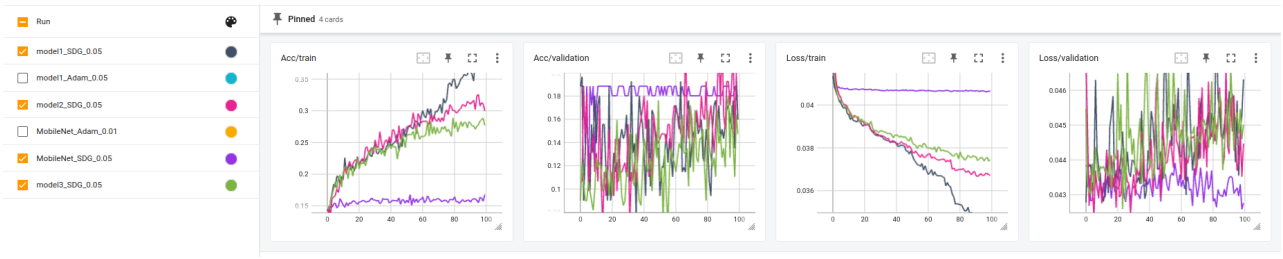

Figure 2: 模型比较

然而实验结果表明，batch size 太小由于波动较大，其精度提升速度不佳，而 batch size 太大其 Optimization 相对较弱，精度提升速度也不佳，还是当 batch size 不大也不小的时候可以得到较好的效果。

图中的三个 batch size 分别为 2,8,32。

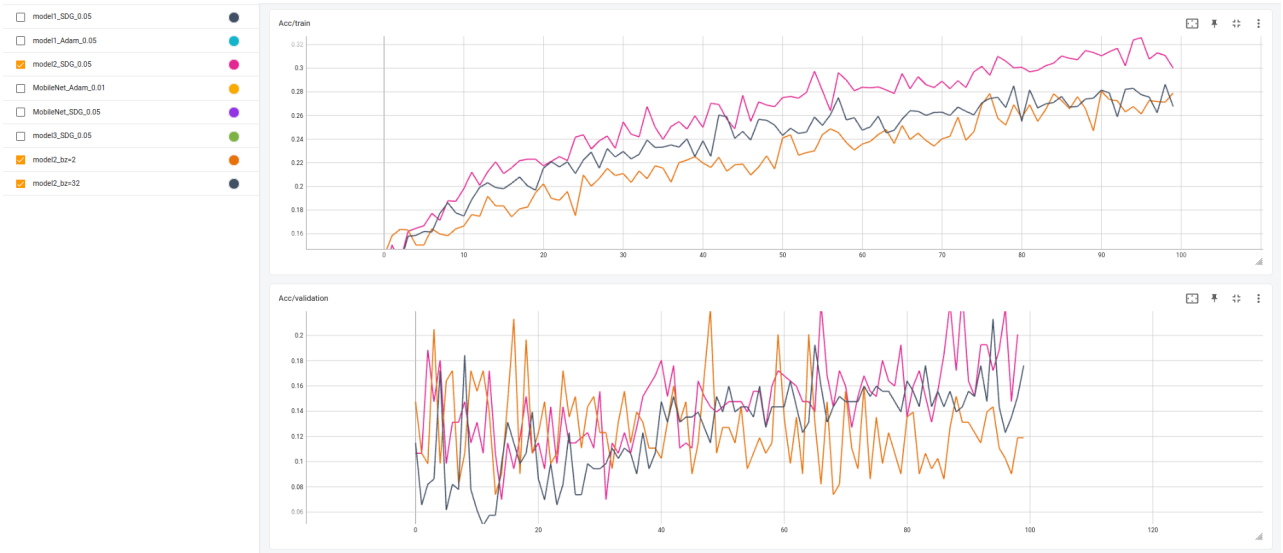

Figure 3: Batch Size

5. binary cross entropy weight

本实验中对 binary cross entropy 尝试加权以弥补数据集不均衡的影响，但最后看下来主要也就是降低了 nan 的权重。

weight 如下，其中第一个是 nan。tensor([0.1538, 0.5702, 0.9417, 1.8279, 1.0358, 1.1509, 0.8071, 1.4453, 0.7579, 0.6342, 1.3811, 1.2684, 0.9711, 0.4744, 1.2948, 0.6407, 1.3511, 0.9711, 1.3223], device='cuda:0')

实验结果如图4所示，感觉差别不大，所以在其他实验中 weight 默认全一。

Figure 4: binary cross entropy weight

V 心得体会

- 在本次实验中，对 Pokemon 数据集进行了自定义、数据增强，这是我第一次自定义数据集，感觉让我的视野突然开阔，因为之前只能用别人已经封装好的数据集，现在我可以自己任意定义了。
- 此外，虽然模型不尽如人意，但是我也尝试了好几天不断地去修改我们模型，也尝试了很多新的 trick，虽然不一定管用，但是还是收获了很多。
- 在实验中，和课上学的很多知识串联起来，可能正是因为期末考完，所以对整个课程的知识有了一个更深入的理解。
- 对于这个数据集，其实我还是挺好奇它到底应该怎么样去构建模型和用各种各样的 trick 去提高精度。我也花了不少时间，但是由于我 27 号早上还有托福考试，不能浪费这 2100 块钱，只能姑且放下。不然按照我这个学期期末做大作业的作风，肯定会很多天一直在钻研，直到有一个满意的答案。其实如果算上 nan 的话之前验证集精度能在 40% 以上，现在这个最高 23% 还挺让我难受的，呜呜呜。

注：**所有本文涉及到的训练数据，均在 run_results 文件夹下**，可以使用 tensorboard 查看，因此未注明精度细节，验证集精度最高为 23.36%。

VI Appendix

参考文献

[1] stackoverflow1, "Lodepng," 2023, https://stackoverflow.com/questions/57565234/pil-not-always-using-3-channels-for-png.

[2] stackoverflow2, "Rgba," 2023, https://stackoverflow.com/questions/51923503/why-do-some-images-have-third-dimension-3-while-others-have-4.

[3] P. tutorials, "Datasets & dataloaders," 2023, https://pytorch.org/tutorials/beginner/basics/data_tutorial.html.

[4] rshnn, "pokemon-types," 2023, https://github.com/rshnn/pokemon-types.

[5] P. ignite, "Topkcategoricalaccuracy," 2023, https://pytorch.org/ignite/generated/ignite.metrics.TopKCategoricalAccuracy.html.