1 切分方式与数据集

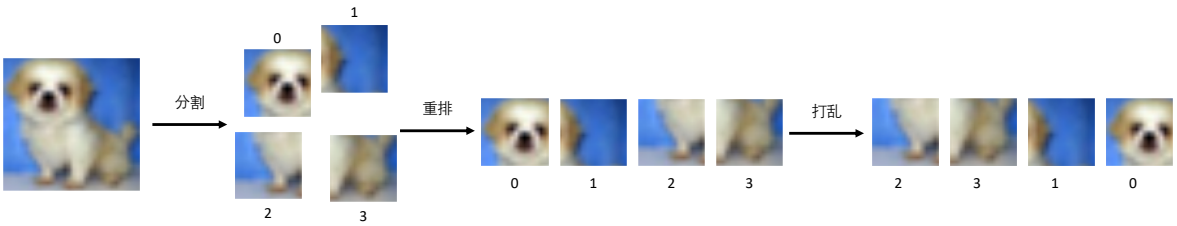

图 1: 数据集切分方式示例

如图所示，对于每张 $(3, 32, 32)$ 的图片，将其划分为左上、右上、左下、右下四个部分，每个部分的大小为 $(3, 16, 16)$。新划分的数据集需要新的标签，为了方便模型计算 loss 和 accuracy，我在模型的标签实现上既使用了 DeepPermNet 中采用的双随机矩阵的方法，又保留了模型本身位置的排列数。

对于一张图片，其切分后重排默认的顺序即 $(0, 1, 2, 3)$，在本例中，随机重排的结果是 $(2, 3, 1, 0)$。`getitem` 函数会返回一个三元组，包括四个一组的切分图片，排列数对应的标签，还有 4×4 的位置矩阵。

2 网络结构

```
PermNet(
    extractor: Extractor(
        conv1: Conv(3, 64, (3, 3), (1, 1), (1, 1), (1, 1), 1, float32[64,], None, Kw=None, fan=None, i=
            None, bound=None)
        bn1: BatchNorm(64, 1e-05, momentum=0.1, affine=True, is_train=True, sync=True)
        conv2: Conv(64, 256, (3, 3), (1, 1), (1, 1), (1, 1), 1, float32[256,], None, Kw=None, fan=None, i=
            None, bound=None)
        bn2: BatchNorm(256, 1e-05, momentum=0.1, affine=True, is_train=True, sync=True)
        pool: AvgPool2d(
            layer: Pool((2, 2), (2, 2), padding=(0, 0), dilation=None, return_indices=None, ceil_mode=
                False, count_include_pad=False, op=mean)
        )
        fc: Linear(16384, 512, float32[512,], None)
    )
    mlp: MLP(
        fc1: Linear(2048, 4096, float32[4096,], None)
        fc2: Linear(4096, 16, float32[16,], None)
    )
)
```

整个网络架构与 DeepPermNet 是类似的，整个网络结构分为 extractor 和 mlp 两个部分，extractor 使用卷积层提取图片特征，mlp 使用全连接层计算排列数的概率分布，并将其映射为 16 维的向量。模型直接将未归

一化的向量作为输出，而对于 `sinkhorn` 的使用，会放在 loss 函数中统一处理。

3 loss 和 accuracy 定义

实际上，即使不使用 `sinkhorn` 函数，也可以直接用模型线性层的输出结果，与真实的标签矩阵做 loss。这样的输出结果虽然没有概率上的可解释性，但是仍然可以指导模型正确地训练。不过实际上，我还是使用了 `sinkhorn` 函数，将模型的输出结果归一化为概率分布，再与真实的标签矩阵做 loss。具体来说，对于神经网络全连接层的输出结果，将其 reshape 为 4×4 的矩阵，作 `sinkhorn` 变换之后，直接与真实的 label 取均方误差。模型并没有选取交叉熵损失函数，因为本任务的标签是一个 4×4 的矩阵，而不是一个 one-hot 向量。

在计算 accuracy 时，使用 `numpy.argmax` 提取每行中最大值的位置，得到一组排列数，再与真实的标签作比较，来计算单个输入中的正确率均值。注意这里正确率的定义方式并不要求模型完全正确地预测出四个子图片的全部位置，当模型部分地预测正确时，也会有一定的正确率。最终整个模型的准确率是所有输入的正确率均值。

4 实现预训练

4.1 一种可能的方法

我首先想到的方法是，直接沿用 task3 所训练好的网络，但是在进行 task2 的分类任务时，把最后的全连接层的输出通道数修改，然后将每张图片裁剪后再输入神经网络中。这种方法不需要对现有的网络进行较大的修改，而对数据集的处理方式也已经比较完善。

具体来说，首先使用经过裁剪和乱序排列的数据集训练本报告第二节中的网络，引导卷积层学习图片的局部特征，但是这里得到的信息是有关于整个数据集的全体特征，所以对后面的缺失数据集也有一定的引导作用。之后在后续的分类任务中，直接将全连接层最后的通道数从 16 改为 10，然后对于每个输入图片，还是将其裁剪为 4 张子图片，然后再输入网络进行若干轮的训练，之后得到分类结果。

不过，经过与同学的研讨，我最终没有采用这种方法。一方面是因为分类和切分排序的任务目标还是有着一定的差别，每次都刻意地将一张完整的图片进行切分再输入网络，可能会导致网络对于图片的全局特征学习的不充分，进而影响网络对图片整体语义信息的识别；另一方面是因为直接将全连接层的最后一层的输出分类直接粗暴的改为 10，而不是逐层进行调整，并不能让模型很好地整合两个任务。

4.2 实际采用的方法

最后我的实现的网络中整合了一个卷积模块和两个全连接模块。在实际训练的过程中，先用 task3 的数据集对卷积模块和全连接 Permuter 进行训练，全连接 Permuter 在后续的分类任务中不起作用，在这里训练的意义仅仅是通过其输出更好地引导卷积层参数的更新；在训练一定的 epoch 过后，冻结卷积层的参数，并换用全连接 Classifier，并继续训练一定的 epoch。整个网络架构如下。

```
1  PermuteClassifier(
2  extractor: ConvEncoder(
3      conv1: Conv(3, 64, (3, 3), (1, 1), (1, 1), (1, 1), 1, float32[64,], None, Kw=None, fan=None, i=None,
           bound=None)
4      bn1: BatchNorm(64, 1e-05, momentum=0.1, affine=True, is_train=True, sync=True)
```

```
        conv2: Conv(64, 256, (3, 3), (1, 1), (1, 1), (1, 1), 1, float32[256,], None, Kw=None, fan=None, i=None
            , bound=None)
        bn2: BatchNorm(256, 1e-05, momentum=0.1, affine=True, is_train=True, sync=True)
        pool: AvgPool2d(
            layer: Pool((2, 2), (2, 2), padding=(0, 0), dilation=None, return_indices=None, ceil_mode=False,
                count_include_pad=False, op=mean)
        )
    )
    mlp2order: MLPPermuter(
        fc1: Linear(65536, 2048, float32[2048,], None)
        fc2: Linear(2048, 4096, float32[4096,], None)
        fc3: Linear(4096, 16, float32[16,], None)
    )
    mlp2class: MLPClassifier(
        fc1: Linear(65536, 4096, float32[4096,], None)
        fc2: Linear(4096, 512, float32[512,], None)
        fc3: Linear(512, 96, float32[96,], None)
        fc4: Linear(96, 10, float32[10,], None)
    )
)
```

5 实验结果

数据指标	默认参数	LR.3	LR.5	LR.7	WD.3	WD.5	WD.7
Train Loss	.0681	.0921	.0849	.1094	.0928	.1182	.1247
Test Loss	.1152	.1347	.1283	.1673	.1328	.1523	.1582
Accuracy	91.24%	90.37%	90.43%	90.01%	90.89%	91.44%	91.32%

表 1: 不同参数设置下的图片拼接实验结果

数据指标	原始数据集	非平衡	过采样	数据增强
Loss	0.8482	1.9138	1.4627	1.3592
Acc.	75.43%	59.52%	62.24%	63.72%

表 2: 对 task2 的预训练实验结果

可以看到，图片拼接的准确率还是比较高的，不同参数的设置对于拼接结果没有本质性的影响。部分原因是我们采取了比较宽松的准确率定义，并没有要求对整个拼接全部预测正确，即使是部分正确也能统计到准确率中。另外，用 task3 模型对 task2 预训练的效果并不明显。

6 其他说明

测试代码完整性可以运行 `python task3.py --debug`，会执行 task3 的主任务训练，并用 task3 的模型进行预训练之后重新执行 task2 的任务。本默认的训练参数已在 `parser.py` 文件中配置好。