实验报告：猫狗分类

韦俊林 (201928016029023)

2020 年 5 月 15 日

1 摘要

使用 MxNet 框架调用 Gluon 构建好的一个残差网络，运用迁移学习 (transfer learning) 技术微调网络，实现只需对数据集进行简单的训练便能得到非常好的效果。

本次实验使用的数据集是 kaggle 的猫狗分类，相比于第一次实验对数据集进行简单的预处理外，本次实验还对数据集进行一系列图像增广操作的预处理。

2 问题描述

无论是医疗诊断、无人车、摄像监控，还是智能滤镜，计算机视觉领域的诸多应用都与我们当下和未来的生活息息相关。而深度学习技术极大的推动了计算机视觉系统性能的提升。

鉴于此，本次实验将基于 kaggle 提供的猫狗数据集利用深度学习技术，实现图像的分类，从而进一步理解和掌握卷积神经网络中卷积层、卷积步长、卷积核、池化层、池化核等概念以及深度学习处理图像分类问题的流程与技巧。

3 解决方案

3.1 损失函数

同实验一，使用 Softmax 交叉熵损失函数，细节见实验一手写数字识别实验报告。

3.2 网络结构

本次实验使用的网络是 ResNet-34 网络后边接上两层全连接层作为输出层构成的网络，其中前边的 ResNet 模型是预训练模型，即 ResNet 网络中的参数经过 ImageNet 数据集训练，也就是说本次实验中真正要训练的参数是该网络最后两层全连接层，运用的是迁移学习的技术，降低训练开销。

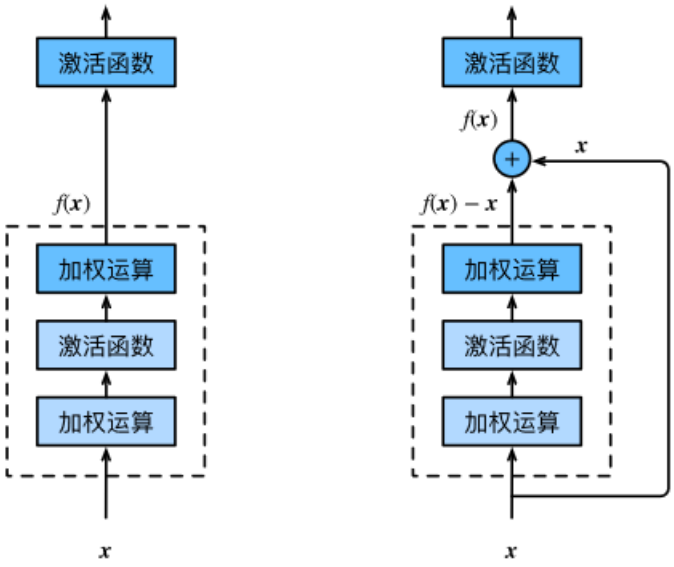
```python
def get_net(labels_num,ctx):
    finetune_net=model_zoo.vision.resnet34_v2(pretrained=True)

    #定义新的输出层
    finetune_net.output_new=nn.HybridSequential(prefix='')
    finetune_net.output_new.add(\
        nn.Dense(256,activation='relu'),\
        nn.Dense(labels_num))
    #初始化输出层参数
    finetune_net.output_new.initialize(init.Xavier(),ctx)
    #将网络参数分配到相应设备上
    finetune_net.collect_params().reset_ctx(ctx)
    return finetune_net
```

```
)
HybridSequential(
  (0): Dense(None -> 256, Activation(relu))
  (1): Dense(None -> 2, linear)
)
```

后接的两层全连接层

残差网络的核心在于残差块，



ResNet-34 网络具体结构如下图：

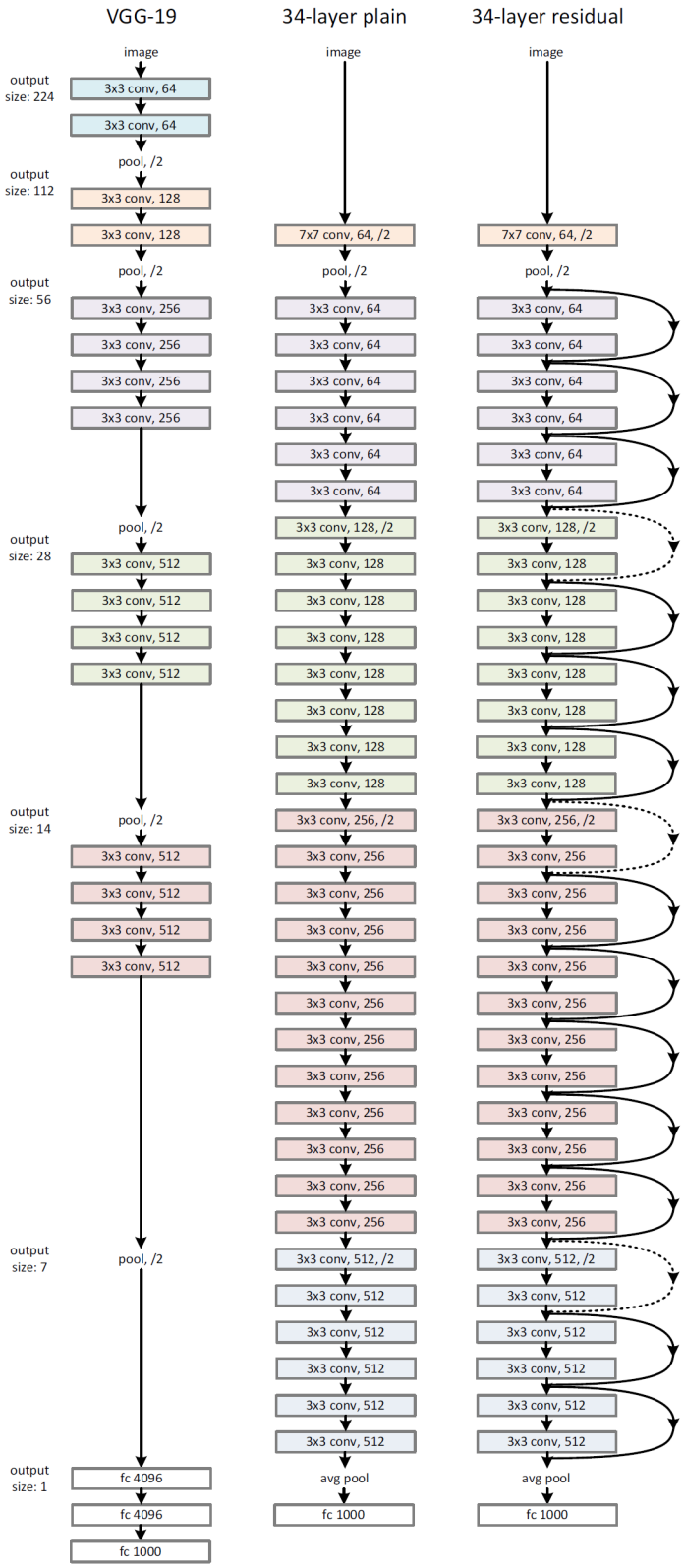


Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Table 1. Architectures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of blocks stacked. Down-sampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.

3.3 相关技巧

图像增广 (image augmentation) 技术在数据集不充足时，通过对训练图像做一系列随机改变，来产生相似又不相同的训练样本，从而扩大训练集的规模。图像增广随机改变训练样本同时可以降低模型对某些属性的依赖，从而提高模型的泛化能力。例如，对图像进行不同方式的裁剪，使目标物体出现在不同位置，从而减轻模型对物体位置的依赖性，同样的也可以调整亮度、色彩等因素来降低模型对色彩的敏感度。

本次实验对训练集进行的图像增广操作如图所示，

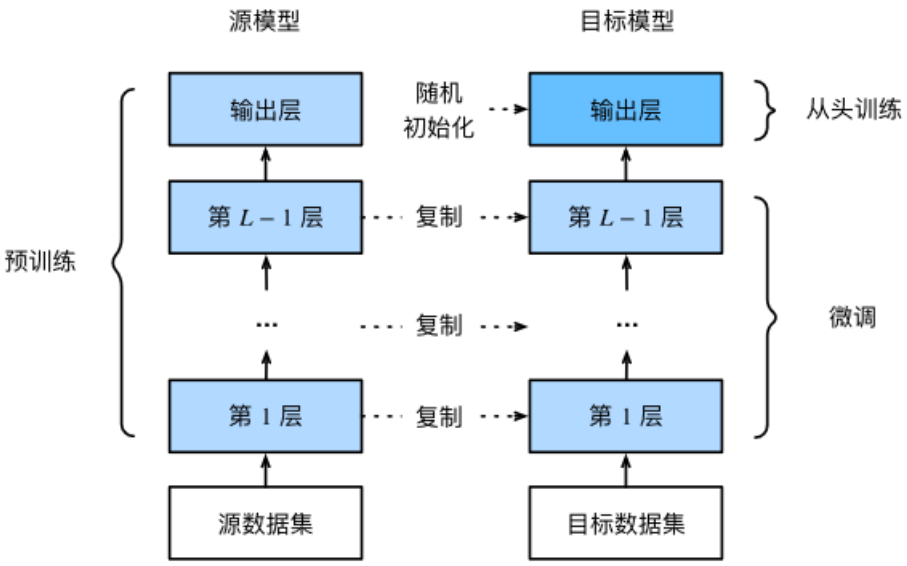
```
if type == 'train':
    transform=img_transforms.Compose([\
        #随机对图像裁剪出面积为原图像面积的0.08~1倍
        #高/宽：3/4 ~ 4/3，最后高度与宽度都缩放到224像素
        img_transforms.RandomResizedCrop(224,scale=(0.08,1.0),ratio=(3.0/4.0,4.0/3.0)),\
        #随机左右翻转
        img_transforms.RandomFlipLeftRight(),\
        #随机变化亮度、对比度、饱和度
        img_transforms.RandomColorJitter(brightness=0.4,contrast=0.4,saturation=0.4),\
        #随机噪声
        img_transforms.RandomLighting(0.1),\
        img_transforms.ToTensor(),\
        # 对图像的每个通道做标准化
        img_transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])])
    return gdata.DataLoader(dataset.transform_first(transform),batch_size=batch_size,shuff
elif type == 'test':
```

在测试时，使用确定的图像，对图像只做标准化预处理。

```
elif type == 'test':
    transform=img_transforms.Compose([\
        img_transforms.Resize(256),\
        img_transforms.CenterCrop(224),\
        img_transforms.ToTensor(),\
        img_transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])])
    return gdata.DataLoader(dataset.transform_first(transform),batch_size=ba
```

迁移学习 (transfer learning) 是将从源数据集学到的知识迁移到目标数据集上。虽然源数据集的图像大多与目标数据集无关，但源数据集上模型隐层训练出的参数可以抽取较通用的图像特征，从而可以帮助快速的识别边缘、纹理、形状等图像特征，这样只需微调隐层中的参数，最后简单的搭建输出层，并重新重点训练输出层即可。

而由于本次实验数据集非常复杂，需要复杂的网络结构以及进行大量的训练（数据集、迭代轮次等方面），个人笔记本性能有限，为了达到较好的性能，本次实验使用 Gluon 的 model_zoo 包提供的 ResNet-34 预训练模型。该预训练模型基于完整的 ImageNet 数据集上进行训练抽取图像特征。



4 实验分析

4.1 数据集介绍

下载 kaggle 猫狗数据集解压后分为 3 个文件 train.zip、test.zip 和 sample_submission.csv。

train 训练集包含了 25000 张猫狗的图片，猫狗各一半，每张图片包含图片本身和图片名。命名规则根据“type.num.jpg”方式命名。

test 测试集包含了 12500 张猫狗的图片，没有标定是猫还是狗，每张图片命名规则根据“num.jpg”，需要注意的是测试集编号从 1 开始，而训练集的编号从 0 开始。

sample_submission.csv 需要将最终测试集的测试结果写入.csv 文件中，上传至 kaggle 进行打分。

由于笔记本性能有限，本次实验只使用了 train.zip 文件并且只选取部分样本进行实验，从中将编号 1 ~ 1500 的猫和狗作为训练集，共 3000 个样本；编号 1501 ~ 1750 的猫和狗作为测试集，共 500 个样本；编号 1751 ~ 2000 的猫和狗作为验证集，共 500 个样本。

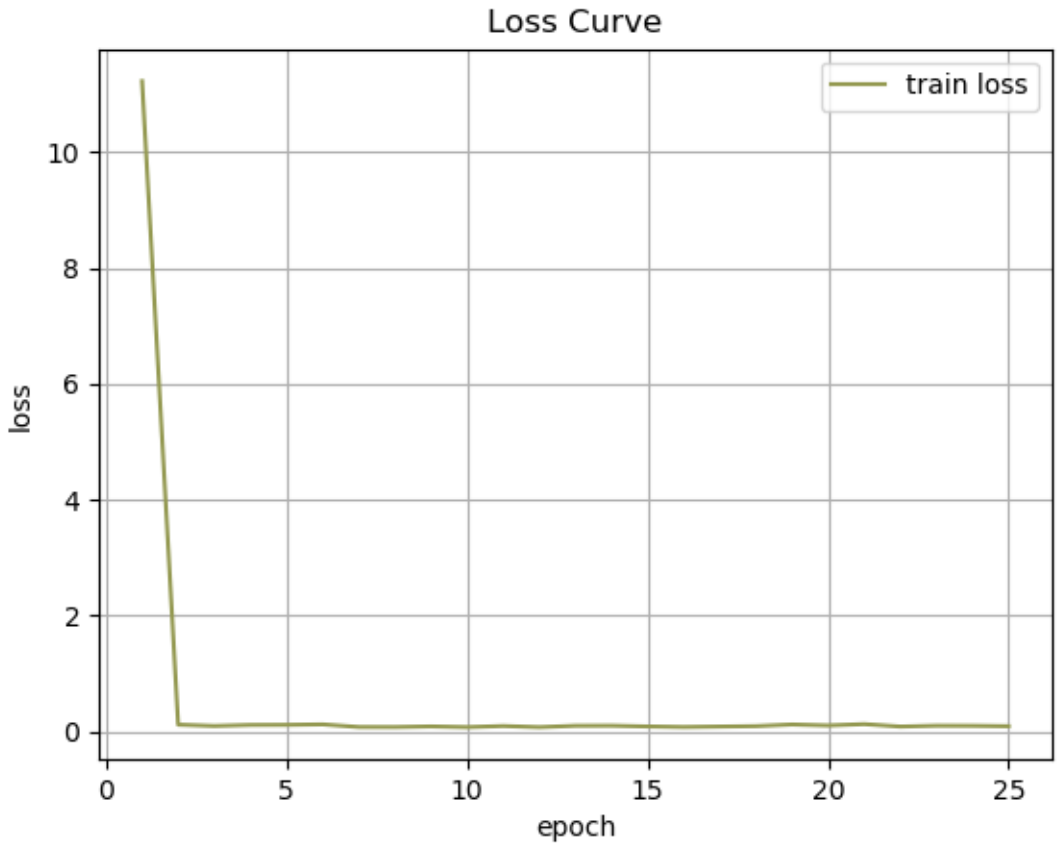
三个数据集相互之间没有交集，训练集用于训练参数，验证集用于在训练过程中测试模型性能，测试集用于最终模型准确率的测试。

加载本地数据集代码如下，

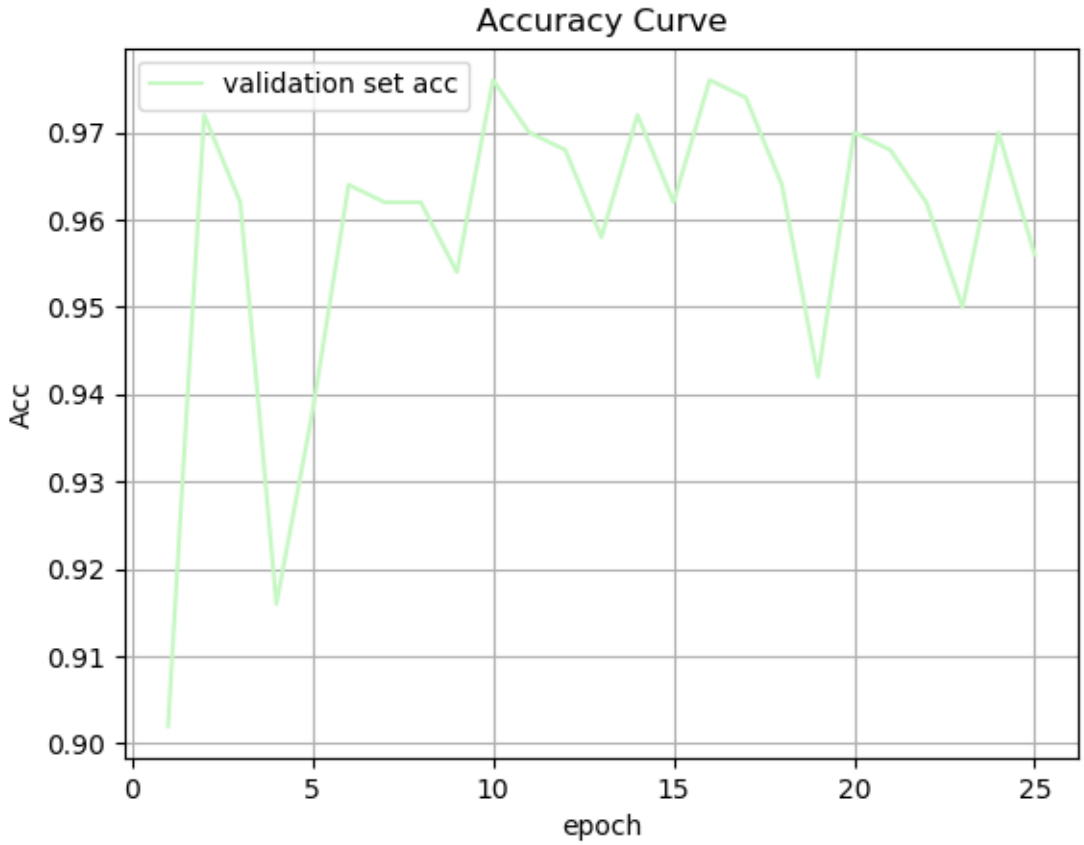
```python
def load_dataset(dataset_name,data_type):
    '''
    加载本地数据集
    number: cat vs dog: 2000:2000
    train set: cat vs dog: 1500:1500
    validation set: cat vs dog: 250:250
    test set: cat vs dog: 250:250
    '''
    PATH=path.abspath(path.join(path.dirname("__file__"),dataset_name))
    file_name=data_type + ' set'
    imgs=gdata.vision.ImageFolderDataset(path.join(PATH,file_name),flag=1)
    return imgs
```

4.2 实验结果

训练过程中，损失函数曲线：



验证集，准确率曲线：



测试集，模型最终正确率 0.992：

```
done.
epoch 1, train loss: 11.230, test acc: 0.902, time: 44.75 sec
epoch 2, train loss: 0.118, test acc: 0.972, time: 86.39 sec
epoch 3, train loss: 0.096, test acc: 0.962, time: 128.35 sec
epoch 4, train loss: 0.113, test acc: 0.916, time: 169.95 sec
epoch 5, train loss: 0.114, test acc: 0.938, time: 211.08 sec
epoch 6, train loss: 0.123, test acc: 0.964, time: 252.02 sec
epoch 7, train loss: 0.078, test acc: 0.962, time: 292.63 sec
epoch 8, train loss: 0.076, test acc: 0.962, time: 333.19 sec
epoch 9, train loss: 0.087, test acc: 0.954, time: 373.66 sec
epoch 10, train loss: 0.075, test acc: 0.976, time: 414.11 sec
epoch 11, train loss: 0.095, test acc: 0.970, time: 454.54 sec
epoch 12, train loss: 0.075, test acc: 0.968, time: 494.94 sec
epoch 13, train loss: 0.099, test acc: 0.958, time: 535.59 sec
epoch 14, train loss: 0.101, test acc: 0.972, time: 575.91 sec
epoch 15, train loss: 0.087, test acc: 0.962, time: 616.53 sec
epoch 16, train loss: 0.077, test acc: 0.976, time: 656.73 sec
epoch 17, train loss: 0.085, test acc: 0.974, time: 697.27 sec
epoch 18, train loss: 0.093, test acc: 0.964, time: 737.50 sec
epoch 19, train loss: 0.120, test acc: 0.942, time: 777.90 sec
epoch 20, train loss: 0.104, test acc: 0.970, time: 818.66 sec
epoch 21, train loss: 0.127, test acc: 0.968, time: 860.32 sec
epoch 22, train loss: 0.085, test acc: 0.962, time: 903.04 sec
epoch 23, train loss: 0.100, test acc: 0.950, time: 945.08 sec
epoch 24, train loss: 0.099, test acc: 0.970, time: 987.73 sec
epoch 25, train loss: 0.092, test acc: 0.956, time: 1030.88 sec
test set acc: 0.992
cat: 0, dog: 1
```

从测试集中，猫与狗各随机抽取三个样本，进行实例检验，实验结果如下：






```
cat: 0, dog: 1
true label:
[[1 0 0]
 [1 0 1]]
predict label:
[[1 0 0]
 [1 0 1]]
```

5 总结与分析

本次实验数据集相对于上次 MNIST 的手写数字数据集，提供的图像不仅结构不同，而且在尺寸、色彩等各方面都更加复杂，因此在数据预处理阶段需要学习很多技术技巧。尤其是本次数据集，目标无关的背景、目标的形态等各方面差异巨大，使用了图像增广技术。为了节省训练开销，使用了迁移学习技术。最终测试集上达到了 99.2% 的准确率。

经过本次实验，熟悉了在较为复杂的数据集上进行深度学习从数据导入到最终预测的整个流程。由于本次实验结果较为理想，笔记本性能有限，所以本次实验未对超参数进行训练。