电 子 科 技 大 学

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

实验报告

Experimental Report

报告题目 基于迁移学习的图像二分类

学　　院 机械与电气工程学院

专　　业 机器人工程

学　　号 2021040902007

作者姓名 经彭宇

指导教师 张鑫

目 录

第一章 迁移学习

1.1 总体思路

使用深度学习模型 ResNet 对蜜蜂和蚂蚁图像进行分类，使用了数据扩充技术和迁移学习，利用预训练的 ResNet 模型作为基础网络，在数据集上进行微调以提高分类精度。

首先数据预处理使用 PyTorch 的`transforms`模块进行数据增强和标准化，包括随机裁剪、水平翻转和归一化等操作。其次模型搭建，使用 ResNet-18 作为基础模型，通过修改最后一层全连接层实现二分类任务。接着使用 SGD 优化器和交叉熵损失函数，采用学习率衰减策略，分阶段进行训练和验证。最后利用训练好的模型对单张图像进行分类预测。

1.2 实现方法

（1）数据预处理：

使用 torchvision 和 torch.utils.data 包来加载数据。但由于原始数据集不包含训练集和测试集，需要创建。

```
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}

data_dir = 'F:\Pycharm\Computer Vision\Final\hymenoptera_data\hymenoptera_data'
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
                                          data_transforms[x])
                  for x in ['train', 'val']}
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=4,
                                              shuffle=True, num_workers=4)
               for x in ['train', 'val']}
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}
class_names = image_datasets['train'].classes
```

图 1-1 创建训练集和测试集

因为没有较大的图像数据集，随意将随机但现实的转换应用于训练图像（例如旋转或水平翻转）来人为引入样本多样性。这有助于使模型暴露于训练数据的不同方面并减少过拟合，如图 1-2 所示。

图 1-2 数据增强

（2）创建模型

根据 Resnet18 模型来创建基础模型，包含全局平均池化层和密集预测层。此模型已基于 ImageNet 数据集进行预训练。接下来需要定义可视化函数，用于模型的评估。

```
def visualize_model(model, num_images=6):
    was_training = model.training
    model.eval()
    images_so_far = 0
    fig = plt.figure()

    with torch.no_grad():
        for i, (inputs, labels) in enumerate(dataloaders['val']):
            inputs = inputs.to(device)
            labels = labels.to(device)

            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)

            for j in range(inputs.size()[0]):
                images_so_far += 1
                ax = plt.subplot(num_images//2, 2, images_so_far)
                ax.axis('off')
                ax.set_title(f'predicted: {class_names[preds[j]]}')
                imshow(inputs.cpu().data[j])

                if images_so_far == num_images:
                    model.train(mode=was_training)
                    return
        model.train(mode=was_training)
```

图 1-3 可视化函数

（3）训练模型

加载预训练模型并重置最终的全连接层，设置周期为 25，如图 1-4 所示。

```
model_ft = models.resnet18(weights='IMAGENET1K_V1')
num_ftrs = model_ft.fc.in_features
# Here the size of each output sample is set to 2.
# Alternatively, it can be generalized to ``nn.Linear(num_ftrs, len(class_names))``.
model_ft.fc = nn.Linear(num_ftrs, 2)

model_ft = model_ft.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)

model_ft = train_model(model_ft, criterion, optimizer_ft, exp_lr_scheduler,
                       num_epochs=25)
```

图 1-4 训练模型

（4）微调

冻结除最后一层之外的所有网络，需要设置冻结参数，以便梯度不计算在中。在特征提取实验中，仅在基础模型的顶部训练了一些层，因此微调少量顶层进一步提高模型的性能。

```
model_conv = torchvision.models.resnet18(weights='IMAGENET1K_V1')
for param in model_conv.parameters():
    param.requires_grad = False

# Parameters of newly constructed modules have requires_grad=True by default
num_ftrs = model_conv.fc.in_features
model_conv.fc = nn.Linear(num_ftrs, 2)

model_conv = model_conv.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that only parameters of final layer are being optimized as
# opposed to before.
optimizer_conv = optim.SGD(model_conv.fc.parameters(), lr=0.001, momentum=0.9)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_conv, step_size=7, gamma=0.1)

model_conv = train_model(model_conv, criterion, optimizer_conv,
                         exp_lr_scheduler, num_epochs=25)
```

图 1-5 微调

（5）模型评估

使用测试集在新数据上验证模型的性能，需要定义 visualize_model_predictions 函数对自定义图像进行预测并可视化预测的类标签以及图像。

```python
def visualize_model_predictions(model,img_path):
    was_training = model.training
    model.eval()

    img = Image.open(img_path)
    img = data_transforms['val'](img)
    img = img.unsqueeze(0)
    img = img.to(device)

    with torch.no_grad():
        outputs = model(img)
        _, preds = torch.max(outputs, 1)

        ax = plt.subplot(2,2,1)
        ax.axis('off')
        ax.set_title(f'Predicted: {class_names[preds[0]]}')
        imshow(img.cpu().data[0])

        model.train(mode=was_training)

visualize_model_predictions(
    model_conv,
    img_path='F:\\Pycharm\\Computer Vision\\Final\\hymenoptera_data\\hymenoptera_data\\val\\bees\\72100438_73de9f17af.jpg'
)
```

图 1-6 定义评估函数并评估模型

1.3 结果展示

首先使用的是预训练 Resnet18 模型，通过更改模型的学习率 lr 与数据样本数量 batch_size 来比较不同的学习率和数据样本数量对训练的模型的影响，这里进行对照实验，保证 lr=0.001，取 batch_size 分别为 8 和 4，保证 batch_size 为 4，取 lr 分别为 0.001 和 0.1。

（1）A 组：lr=0.001，batch_size=8

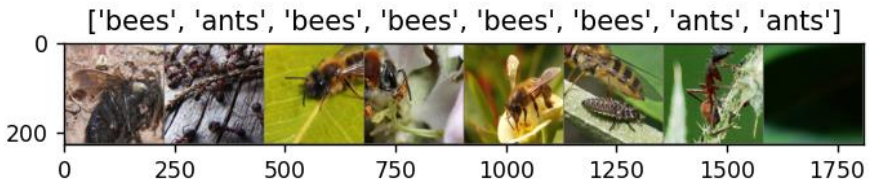

图 1-7 A 组数据集样本显示

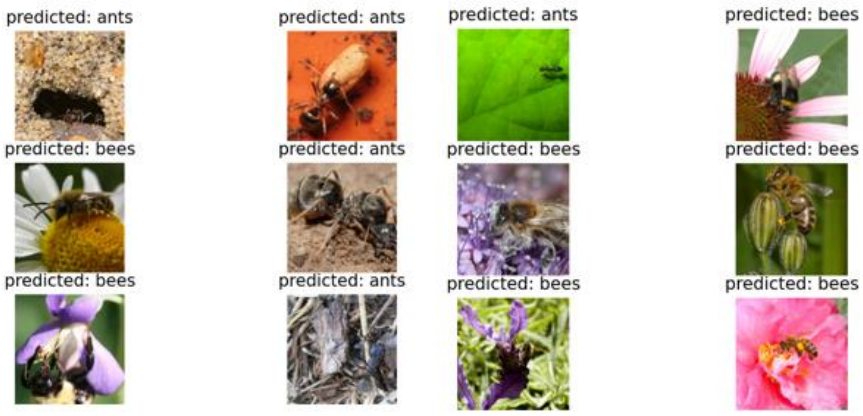

图 1-8 A 组微调前图像预测（左）和微调后图像预测（右）

```
Epoch 23/24
----------
train Loss: 0.0910 Acc: 0.9590
val Loss: 0.1829 Acc: 0.9477

Epoch 24/24
----------
train Loss: 0.1448 Acc: 0.9385
val Loss: 0.1808 Acc: 0.9477

Training complete in 11m 22s
Best val Acc: 0.954248
```

```
Epoch 23/24
----------
train Loss: 0.2291 Acc: 0.9057
val Loss: 0.1937 Acc: 0.9281

Epoch 24/24
----------
train Loss: 0.2153 Acc: 0.8975
val Loss: 0.1898 Acc: 0.9412

Training complete in 8m 58s
Best val Acc: 0.941176
```

图 1-9 A 组微调前训练结果（左）和微调后训练结果（右）

可以看到当 lr=0.001，batch_size=8 时，微调后的模型精确度比微调前的精确度低了 1%以上，将微调后的模型用于预测图像，结果如图 1-10 所示。

图 1-10 A 组图像预测

（2）B 组：lr=0.001，batch_size=4

相较于 A 组，B 组将样本数据集调至 4，以比较 batch_size 对模型的影响。

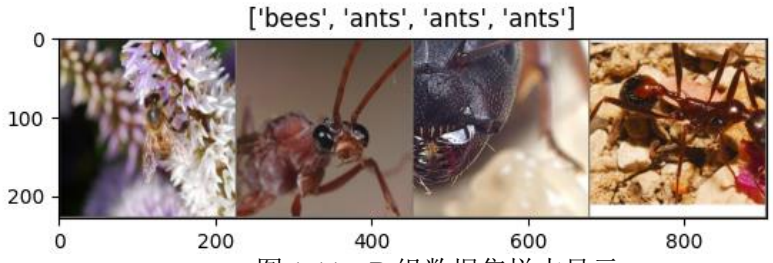

图 1-11 B 组数据集样本显示

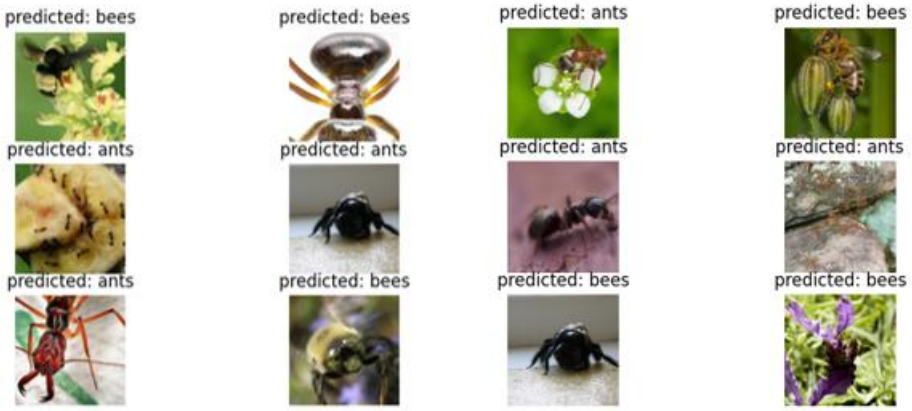

图 1-12 B 组微调前图像预测（左）和微调后图像预测（右）

```
Epoch 23/24
----------
train Loss: 0.2542 Acc: 0.8893
val Loss: 0.2278 Acc: 0.9020

Epoch 24/24
----------
train Loss: 0.2419 Acc: 0.9057
val Loss: 0.2397 Acc: 0.9085

Training complete in 14m 53s
Best val Acc: 0.934641
```

```
Epoch 23/24
----------
train Loss: 0.2520 Acc: 0.9016
val Loss: 0.1594 Acc: 0.9542

Epoch 24/24
----------
train Loss: 0.3543 Acc: 0.8443
val Loss: 0.1693 Acc: 0.9477

Training complete in 11m 26s
Best val Acc: 0.954248
```

图 1-13 B 组微调前训练结果（左）和微调后训练结果（右）

可以看到当 lr=0.001，batch_size=4 时，微调后的模型精确度比微调前的精确度高了 2%以上，可以推测，batch_size 越小，对微调后的模型的精确度提高有作用，将微调后的模型用于预测图像，结果如图 1-14 所示。

图 1-14 B 组图像预测

（3）C 组：Lr=0.1,batch_size=4

相较于 B 组，C 组将学习率调至 0.1，以比较学习率对模型的影响。

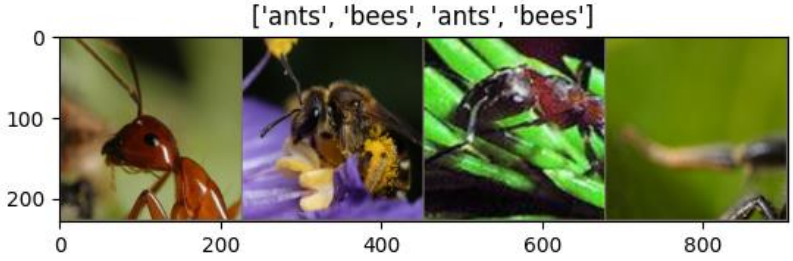

图 1-15 C 组数据集样本显示

图 1-16 C 组微调前图像预测（左）和微调后图像预测（右）

```
Epoch 23/24
----------
train Loss: 0.3081 Acc: 0.8566
val Loss: 0.1583 Acc: 0.9477

Epoch 24/24
----------
train Loss: 0.3122 Acc: 0.8361
val Loss: 0.1724 Acc: 0.9477

Training complete in 9m 24s
Best val Acc: 0.954248
```

```
Epoch 23/24
----------
train Loss: 0.6804 Acc: 0.5984
val Loss: 0.6259 Acc: 0.6536

Epoch 24/24
----------
train Loss: 0.6634 Acc: 0.5820
val Loss: 0.6295 Acc: 0.6536

Training complete in 11m 56s
Best val Acc: 0.679739
```

图 1-17 C 组微调前训练结果（左）和微调后训练结果（右）

可以看到当 lr=0. 1，batch_size=4 时，微调后的模型精确度比微调前的精确度低了将近 30%，比 B 组微调后的精确度低了将近 30%，可以推测，lr 越小，对微调后的模型的精确度提高有作用，将微调后的模型用于预测图像。

图 1-18 C 组图像预测

此外采用预训练的 VGG16 模型，取 lr=0.001，batch_size=8，同 Resnet18 模型的 A 组进行比较，对比不同的神经网络的性能

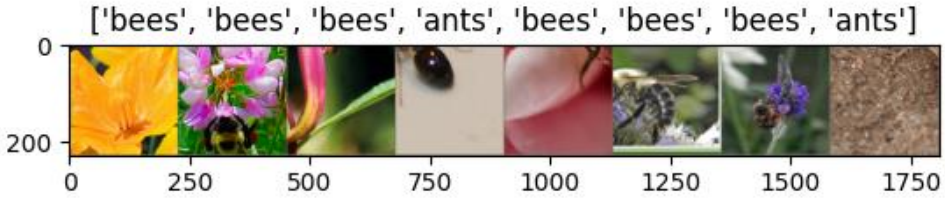

图 1-19 VGG16 模型下数据集样本显示

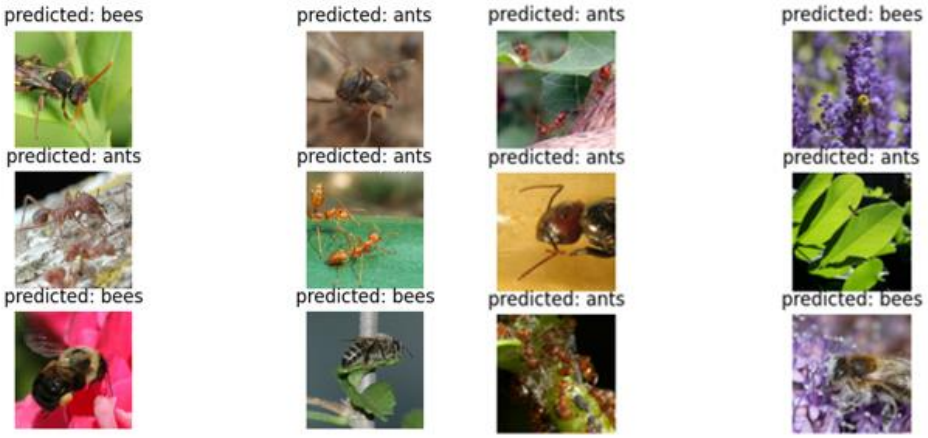

图 1-20 VGG16 模型下微调前图像预测（左）和微调后图像预测（右）

```
Epoch 18/19
----------
train Loss: 0.1231 Acc: 0.9590
val Loss: 0.2364 Acc: 0.9346

Epoch 19/19
----------
train Loss: 0.0981 Acc: 0.9672
val Loss: 0.2368 Acc: 0.9281

Training complete in 37m 36s
Best val Acc: 0.934641
```

```
Epoch 18/19
----------
train Loss: 0.0912 Acc: 0.9754
val Loss: 0.0721 Acc: 0.9739

Epoch 19/19
----------
train Loss: 0.0987 Acc: 0.9713
val Loss: 0.0720 Acc: 0.9739

Training complete in 18m 34s
Best val Acc: 0.973856
```

图 1-21 VGG16 模型微调前训练结果（左）和微调后训练结果（右）

可以看到当学习率和样本数据相同时，VGG16 微调后的模型精确度比 Resnet 微调后的精确度高了 2%以上，二者微调前的精确度差别不大，但无论是微调前还是微调后，VGG16 训练的时间都比 Resnet 多且 VGG16 微调前训练的时间是微调后的两倍。

图 1-22 VGG16 模型图像预测

1.4 遇到的问题及解决方式

（1）子进程启动异常

```
RuntimeError:
        An attempt has been made to start a new process before the
        current process has finished its bootstrapping phase.

        This probably means that you are not using fork to start your
        child processes and you have forgotten to use the proper idiom
        in the main module:

            if __name__ == '__main__':
                freeze_support()
                ...

        The "freeze_support()" line can be omitted if the program
        is not going to be frozen to produce an executable.
```

图 1-23 子进程启动异常

这个错误通常出现在使用 Python 的 multiprocessing 模块创建子进程时，尤其

是在 Windows 系统上。这个错误的原因是因为 Python 在启动子进程时需要进行一些初始化工作，而代码中可能在初始化完成之前就尝试启动了新的子进程。具体来说，Python 在启动子进程时需要确保子进程独立于父进程，这通常通过操作系统提供的 fork 机制实现。但是在 Windows 上，由于没有 fork 机制，Python 使用了一种不同的方法来启动子进程，这就需要在主模块中使用`if __name__ =='__main__':`语句来保证子进程的正确启动。

```
15 ▶ if __name__=='__main__':
16       cudnn.benchmark = True
17       plt.ion()   # interactive mode
18
19       # Data augmentation and normalization for training
20       # Just normalization for validation
21       data_transforms = {
22           'train': transforms.Compose([
23               transforms.RandomResizedCrop(224),
24               transforms.RandomHorizontalFlip(),
```

图 1-24 使用`if __name__ =='__main__':

（2）VGG 模型中没有全连接层

```
AttributeError: 'VGG' object has no attribute 'fc'
```

图 1-25 VGG 模型无全连接层

因为 VGG 网络没有全连接层(fc)。相反，VGG 网络使用了全局平均池化层，将卷积层的输出转换为向量，然后将该向量输入到分类器中进行分类，因此，如果使用 VGG 网络进行迁移学习，需要替换最后一个线性层。

```
169     model_ft = torchvision.models.vgg16(pretrained=True)
170
171     model_ft.classifier._modules['6'] = nn.Linear(4096, 8)
```

图 1-26 解决方法

1.5 思考和总结

总结起来，通过使用预训练模型，可以加快模型收敛速度并提高分类精度。此外随机裁剪和翻转等数据增强操作可以增加模型的泛化能力，根据任务需求选择合适的模型和调整超参数，对模型性能有重要影响。

在结果展示中，进行了对比实验，比较了不同学习率和数据样本数量对模型训练和微调后性能的影响。通过实验结果可以看出，较小的数据样本数量和适当的学习率可以提高微调后模型的精确度，但过大或过小的学习率可能导致性能下降。

实验中可以看出学习率不宜过大，过大的学习率可能导致模型参数在训练过

程中快速调整，快速收敛可能导致跳过最优解，会导致模型无法收敛到全局最优解或局部最优解，影响模型的性能和泛化能力。除此之外过大的学习率可能导致训练过程中出现震荡和不稳定的现象，使得模型在训练过程中难以收敛。这样会增加训练时间和难度，并且可能使模型的性能无法达到预期。还有就是过大的学习率可能导致梯度爆炸的问题，即梯度值变得非常大，使得模型参数更新过大，进而影响模型的稳定性和收敛性。最后一点是过大的学习率使得参数更新幅度过大，导致无法细致调整模型参数，从而影响模型对数据的拟合能力。

同时样本数据集不宜过多，主要有以下几个方面的考虑：如果样本数据集过大，模型可能会过度拟合训练数据，导致在训练集上表现良好但在测试集或实际应用中表现较差，此外大规模数据集需要更长的训练时间，因为模型需要对更多的样本进行学习和参数调整。这会增加模型开发和调试的时间成本，特别是在调参过程中可能需要多次训练和评估模型。

另外，本实验对比了使用预训练的 VGG16 模型和 ResNet-18 模型的性能，发现在相同的学习率和数据样本数量下，VGG16 微调后的模型精确度略高于 ResNet-18，但 VGG16 的训练时间较长。

综合来看，通过合适的数据处理、模型选择和调参，可以提高深度学习模型在图像分类任务中的性能和泛化能力，但需要根据具体场景进行进一步优化和调整，避免过拟合和欠拟合等问题。

第二章 其他迁移学习

2.1 总体思路

使用 TensorFlow 进行图像分类的迁移学习是一种有效的方法，能够构建高性能的模型。首先，需要准备数据集。使用了一个包含猫和狗图像的数据集。这个数据集经过预处理，包括将图像调整为相同的尺寸（160x160 像素），并分为训练集和验证集。

接下来，使用 TensorFlow 中的 ImageDataGenerator 来进行数据增强，通过对训练图像进行随机变换来生成更多的训练样本。这有助于模型学习到更多的特征，并提高泛化能力。

接着，构建迁移学习模型。在这里使用了 MobileNet V2 作为基础模型，并在其顶部添加了全局平均池化层和密集层。通过迁移学习，可以利用 MobileNet V2 在 ImageNet 数据集上学到的特征来帮助模型更好地理解图像。

然后，编译模型并进行训练。使用了 Adam 优化器和二元交叉熵损失函数，并监控模型的准确率。在初始训练之后，对模型进行了微调，通过解冻部分层并使用更小的学习率来进一步优化模型。

最后，评估模型的性能并进行预测。通过绘制训练和验证的准确率和损失曲线来分析模型的学习情况，并在测试集上进行评估来检验模型的泛化能力。同时，需要展示模型在测试集上的预测结果，以便更直观地了解模型的表现。

2.2 实现方法

（1）数据预处理：

下载数据集并解压，然后使用 tf.keras.utils.image_dataset_from_directory 效用函数创建一个 tf.data.Dataset 进行训练和验证。

```
16 train_dataset = tf.keras.utils.image_dataset_from_directory(train_dir,
17                                                             shuffle=True,
18                                                             batch_size=BATCH_SIZE,
19                                                             image_size=IMG_SIZE)
20
```

图 2-1 创建 Dataset

为确保数据的准确性，显示训练集中的前九个图像和标签。如图 1-2 所示。

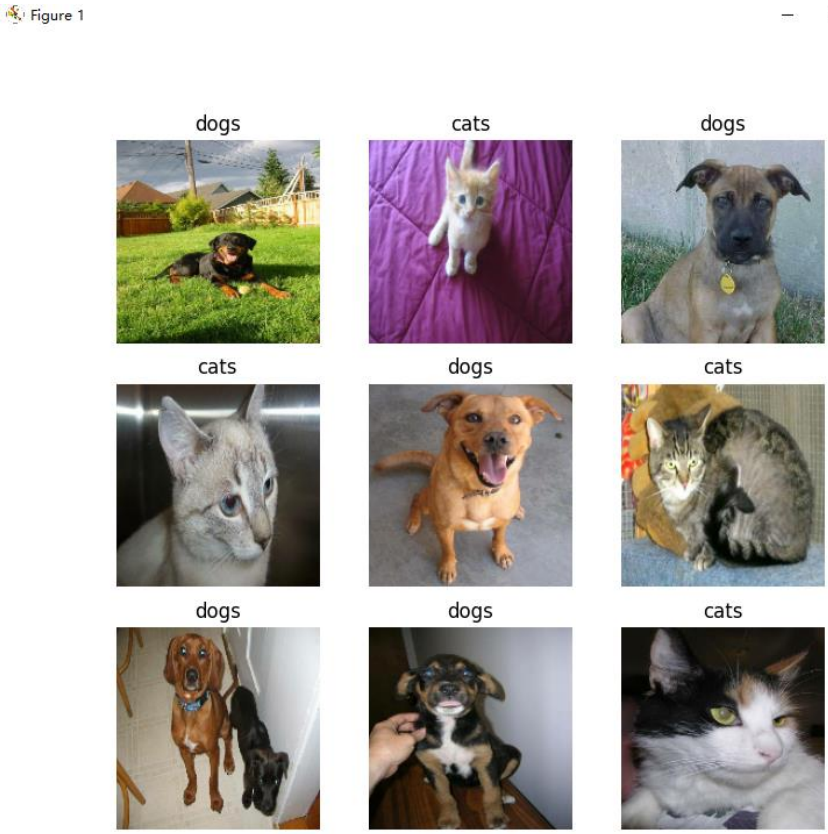

图 2-2 前九个图像和标签

但由于原始数据集不包含测试集，需要创建一个，使用 tf.data.experimental.cardinality 确定验证集中有多少批次的数据，然后将其中的 20% 移至测试集。

```
35 val_batches = tf.data.experimental.cardinality(validation_dataset)
36 test_dataset = validation_dataset.take(val_batches // 5)
37 validation_dataset = validation_dataset.skip(val_batches // 5)
38
39 print('Number of validation batches: %d' % tf.data.experimental.cardinality(validation_dataset))
40 print('Number of test batches: %d' % tf.data.experimental.cardinality(test_dataset))
```

图 2-3 创建测试集

```
Number of validation batches: 26
Number of test batches: 6
```

图 2-4 训练集和测试集的数量

因为没有较大的图像数据集，随意将随机但现实的转换应用于训练图像（例如旋转或水平翻转）来人为引入样本多样性。这有助于使模型暴露于训练数据的不同方面并减少过拟合，如图 1-5 所示。

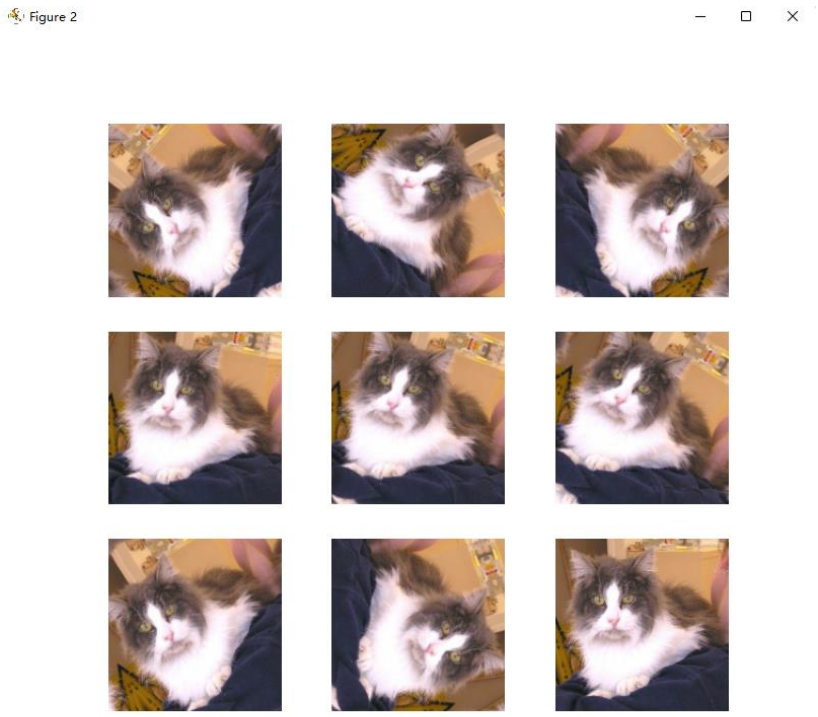

图 2-5 数据扩充

最后需要重新缩放像素值，因为下载的模型期望像素值处于 [-1, 1] 范围内，但此时，图像中的像素值处于 [0, 255] 范围内，需要重新缩放这些像素值。

```
62  preprocess_input = tf.keras.applications.mobilenet_v2.preprocess_input
63
64  rescale = tf.keras.layers.Rescaling(1./127.5, offset=-1)
```

图 2-6 缩放像素值

（2）创建模型

根据 Google 开发的 MobileNet V2 模型来创建基础模型，包含全局平均池化层和密集预测层。此模型已基于 ImageNet 数据集进行预训练，ImageNet 数据集是一个包含 140 万个图像和 1000 个类的大型数据集。ImageNet 是一个研究训练数据集，具有各种各样的类别，例如 jackfruit 和 syringe。

冻结在上一步中创建的卷积基，并用作特征提取程序，冻结可避免在训练期间更新给定层中的权重。MobileNet V2 具有许多层，因此将整个模型的 trainable 标记设置为 False 会冻结所有这些层。

```
76  base_model.trainable = False
```

图 2-7 冻结卷积层

接下来需要添加分类头，使用 tf.keras.layers.GlobalAveragePooling2D 层在 5x5 空间位置内取平均值，从特征块生成预测，以将特征转换成每个图像一个向量（包含 1280 个元素）。应用 tf.keras.layers.Dense 层将这些特征转换成每个图像一个预测。因为此预测将被视为 logit 或原始预测值，所以在此处不需要激活函数。正数预测 1 类，负数预测 0 类。通过使用 Keras 函数式 API 将数据扩充、重新缩放、base_model 和特征提取程序层链接在一起来构建模型。

```
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)

prediction_layer = tf.keras.layers.Dense(1)
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)

inputs = tf.keras.Input(shape=(160, 160, 3))
x = data_augmentation(inputs)
x = preprocess_input(x)
x = base_model(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)
```

图 2-8 添加分类头

（3）训练模型

在训练模型前，需要先编译模型。由于存在两个类，并且模型提供线性输出，所以将 tf.keras.losses.BinaryCrossentropy 损失与 from_logits=True 结合使用。

Model: "functional_2"

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 160, 160, 3)	0
sequential (Sequential)	(None, 160, 160, 3)	0
true_divide (TrueDivide)	(None, 160, 160, 3)	0
subtract (Subtract)	(None, 160, 160, 3)	0
mobilenetv2_1.00_160 (Functional)	(None, 5, 5, 1280)	2,257,984
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0
dropout (Dropout)	(None, 1280)	0
dense (Dense)	(None, 1)	1,281

Total params: 2,259,265 (8.62 MB)
Trainable params: 1,281 (5.00 KB)
Non-trainable params: 2,257,984 (8.61 MB)

图 2-9 编译完的模型

在训练数据集上进行初始 epoch 训练，并在验证数据集上进行验证。经过 10 个周期的训练后，在验证集上看到约 92% 的准确率。

```
initial loss: 0.90
initial accuracy: 0.41
Epoch 1/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 156ms/step - accuracy: 0.8978 - loss: 0.2379 - val_accuracy: 0.9443 - val_loss: 0.1336
Epoch 2/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 153ms/step - accuracy: 0.9000 - loss: 0.2296 - val_accuracy: 0.9455 - val_loss: 0.1273
Epoch 3/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 156ms/step - accuracy: 0.9092 - loss: 0.2062 - val_accuracy: 0.9505 - val_loss: 0.1286
Epoch 4/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 157ms/step - accuracy: 0.9104 - loss: 0.2081 - val_accuracy: 0.9530 - val_loss: 0.1159
Epoch 5/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 155ms/step - accuracy: 0.9116 - loss: 0.2058 - val_accuracy: 0.9554 - val_loss: 0.1119
Epoch 6/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 156ms/step - accuracy: 0.9129 - loss: 0.2041 - val_accuracy: 0.9592 - val_loss: 0.1005
Epoch 7/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 155ms/step - accuracy: 0.9136 - loss: 0.1824 - val_accuracy: 0.9592 - val_loss: 0.1002
Epoch 8/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 161ms/step - accuracy: 0.9207 - loss: 0.1708 - val_accuracy: 0.9653 - val_loss: 0.1008
Epoch 9/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 154ms/step - accuracy: 0.9317 - loss: 0.1722 - val_accuracy: 0.9666 - val_loss: 0.0922
Epoch 10/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 159ms/step - accuracy: 0.9228 - loss: 0.1746 - val_accuracy: 0.9703 - val_loss: 0.0892
```

图 2-10 模型训练

查看使用 MobileNet V2 基础模型作为固定特征提取程序时训练和验证准确率/损失的学习曲线。

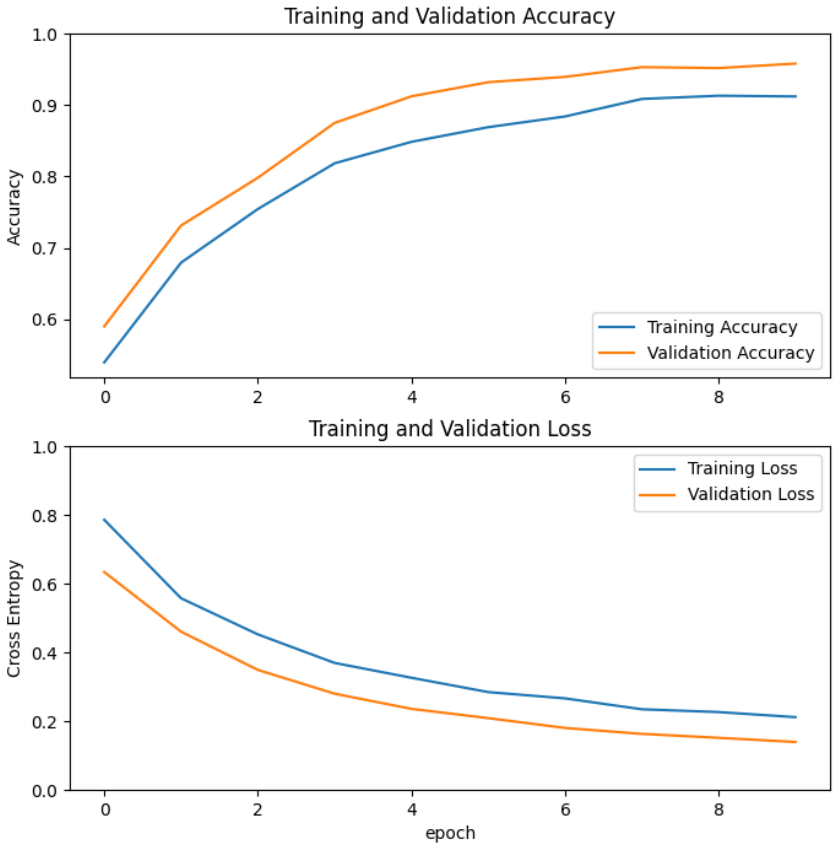

图 2-11 学习曲线

tf.keras.layers.BatchNormalization 和 tf.keras.layers.Dropout 等层会影响训练期间的准确率。在计算验证损失时，它们处于关闭状态，所以验证指标明显优于训练指标。训练指标报告的是某个周期的平均值，而验证指标则在经过该周期后才进行评估，因此验证指标会看到训练时间略长一些的模型。

（4）微调

在特征提取实验中，仅在 MobileNet V2 基础模型的顶部训练了一些层。预训练网络的权重在训练过程中未更新。因此微调少量顶层进一步提高模型的性能。

解冻 base_model 并将底层设置为不可训练。重新编译模型然后恢复训练。

```
143 base_model.trainable = True
144
145 print("Number of layers in the base model: ", len(base_model.layers))
```

图 2-12 解冻顶层

```
Number of layers in the base model:  154
```

图 2-13 base_model 中的层数

随后编译模型，继续训练模型，并绘制学习曲线

```
Model: "functional_2"
```

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 160, 160, 3)	0
sequential (Sequential)	(None, 160, 160, 3)	0
true_divide (TrueDivide)	(None, 160, 160, 3)	0
subtract (Subtract)	(None, 160, 160, 3)	0
mobilenetv2_1.00_160 (Functional)	(None, 5, 5, 1280)	2,257,984
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0
dropout (Dropout)	(None, 1280)	0
dense (Dense)	(None, 1)	1,281

```
Total params: 2,259,265 (8.62 MB)
Trainable params: 1,281 (5.00 KB)
Non-trainable params: 2,257,984 (8.61 MB)
```

图 2-14 解冻后编译完的模型

```
Epoch 10/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 18s 208ms/step - accuracy: 0.8339 - loss: 0.4328 - val_accuracy: 0.9715 - val_loss: 0.0937
Epoch 11/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 209ms/step - accuracy: 0.9034 - loss: 0.2684 - val_accuracy: 0.9802 - val_loss: 0.0827
Epoch 12/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 14s 220ms/step - accuracy: 0.8953 - loss: 0.2472 - val_accuracy: 0.9814 - val_loss: 0.0605
Epoch 13/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 209ms/step - accuracy: 0.9240 - loss: 0.1867 - val_accuracy: 0.9827 - val_loss: 0.0564
Epoch 14/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 203ms/step - accuracy: 0.9245 - loss: 0.1754 - val_accuracy: 0.9839 - val_loss: 0.0554
Epoch 15/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 14s 217ms/step - accuracy: 0.9417 - loss: 0.1536 - val_accuracy: 0.9827 - val_loss: 0.0501
Epoch 16/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 209ms/step - accuracy: 0.9395 - loss: 0.1284 - val_accuracy: 0.9814 - val_loss: 0.0477
Epoch 17/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 14s 217ms/step - accuracy: 0.9510 - loss: 0.1359 - val_accuracy: 0.9839 - val_loss: 0.0470
Epoch 18/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 14s 221ms/step - accuracy: 0.9479 - loss: 0.1450 - val_accuracy: 0.9839 - val_loss: 0.0441
Epoch 19/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 14s 224ms/step - accuracy: 0.9527 - loss: 0.1150 - val_accuracy: 0.9814 - val_loss: 0.0423
Epoch 20/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 213ms/step - accuracy: 0.9517 - loss: 0.1146 - val_accuracy: 0.9827 - val_loss: 0.0438
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 105ms/step - accuracy: 0.9658 - loss: 0.0387
```

图 2-15 训练解冻后的模型

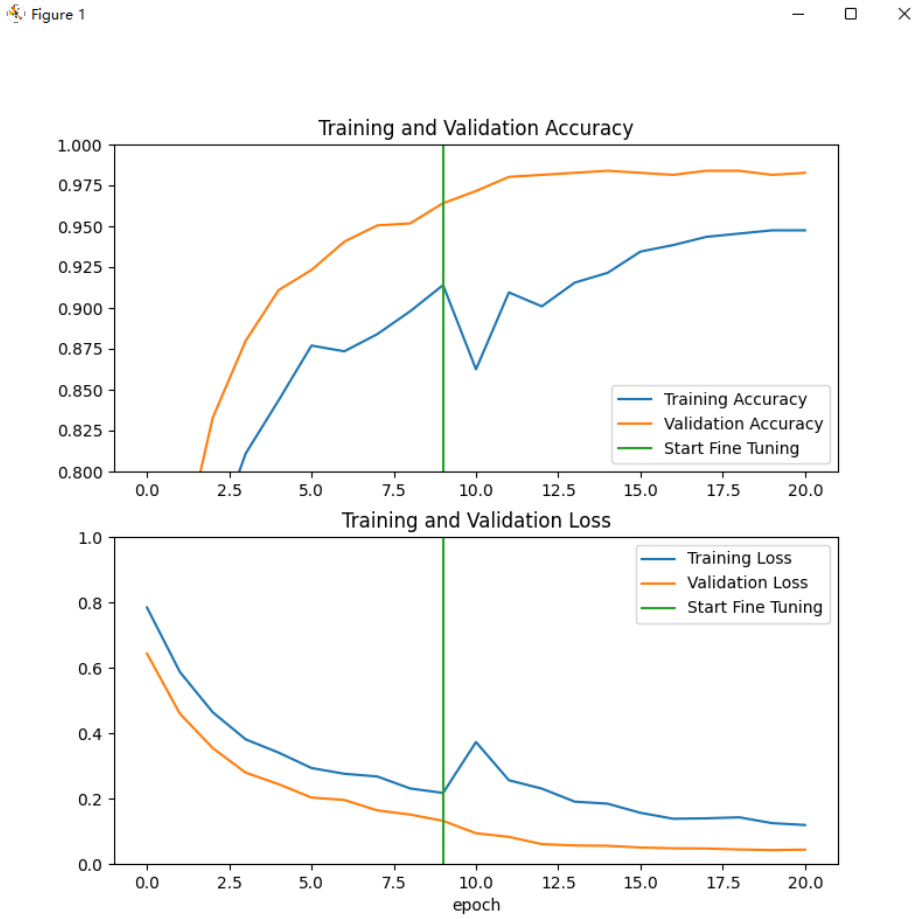

图 2-16 微调后的学习曲线

在微调 MobileNet V2 基础模型的最后几层并在这些层上训练分类器时，验证损失比训练损失高得多，因此可能存在一些过拟合。当新的训练集相对较小且与原始 MobileNet V2 数据集相似时，也可能存在一些过拟合。经过微调后，模型在验证集上的准确率几乎达到 95%。

（5）模型评估

使用测试集在新数据上验证模型的性能。

```
195  loss, accuracy = model.evaluate(test_dataset)
196  print('Test accuracy :', accuracy)
```

图 2-17 测试模型的性能

```
Test accuracy : 0.9791666865348816
```

图 2-18 模型的性能

2.3 效果展示

展示测试集中的图像网格，并显示模型预测的类别标签，以展示模型的预测结果。

```
Predictions:
 [1 0 0 1 0 1 0 1 0 0 1 0 0 0 0 1 1 1 1 1 1 0 1 0 1 0 0 0 1 0 0 0]
Labels:
 [1 0 0 1 0 1 0 1 0 0 1 0 0 0 0 1 1 1 1 1 1 0 1 0 1 0 0 0 1 0 0 0]
```

图 2-19 测试结果

2.4 思考和总结

使用预训练模型进行特征提取时，对于小型数据，常见做法是利用基于相同域中的较大数据集训练的模型所学习的特征，需要实例化预训练模型并在顶部添加一个全连接分类器。预训练模型处于“冻结状态”，训练过程中仅更新分类器的权重。在这种情况下，卷积基提取了与每个图像关联的所有特征。

为了进一步提高性能，需要通过微调将预训练模型的顶层重新用于新的数据集。在本例中，通过调整权重以使模型学习特定于数据集的高级特征。当训练数据集较大且与训练预训练模型所使用的原始数据集非常相似时，准确性达到 95%。

第三章 总结

文本中涵盖了使用深度学习模型（ResNet、MobileNet V2、VGG16）进行图像分类的方法和实验结果，并对比了不同参数设置对模型性能的影响。

使用 ResNet 进行图像分类时，对数据预处理包括数据增强和标准化操作，如随机裁剪、水平翻转和归一化等，通过微调最后一层全连接层来进行二分类任务。实验结果对比了不同学习率和数据样本数量对模型训练和微调后性能的影响，发现较小的数据样本数量和适当的学习率可以提高微调后模型的精确度。此外使用了 VGG16 进行图像分类任务，并对比了它们在相同条件下的性能，在相同的学习率和数据样本数量下，VGG16 微调后的模型精确度略高于 ResNet-18，但 VGG16 的训练时间较长。

在使用 MobileNet V2 时，构建了图像分类模型，并通过编译和训练得到了模型。在训练集和验证集上达到了一定的准确率和较低的损失，但初始训练可能存在一定程度的过拟合。通过微调顶层模型，提高了模型的泛化能力和性能。在测试集上进行了评估和预测，得到了模型在未见数据上的表现。实验结果显示微调后的模型在验证集上的准确率较高，达到了 95%左右。

迁移学习模型利用了预训练模型的特征提取能力，适用于数据量较小的情况。模型结构简单清晰，易于理解和调整。但对于特定任务的表现可能不如从零开始训练的模型，因为预训练模型的特征可能不完全适用于目标任务。此外可能存在过拟合问题，特别是当训练数据集较小或者预训练模型特征不够泛化时。

综合来看，合适的数据处理、模型选择和调参对于深度学习模型在图像分类任务中的性能和泛化能力至关重要。通过对比实验和分析，可以更好地理解模型训练过程中的影响因素，进而优化模型设计和训练策略。