In [5]:
```
import os
os.environ["KMP_DUPLICATE_LIB_OK"] = "TRUE"
import torch
import torch.nn as nn
import torch.nn.functional as F
from d2l import torch as d2l
```

模型搭建

In [19]:
```
from torchvision import models

class ResNet18(nn.Module):
    def __init__(self):
        super(ResNet18, self).__init__()
        # 加载预训练的 ResNet18 模型
        self.model = models.resnet18(pretrained=True)
        # 替换最后的全连接层，适配猫狗分类任务（输出2个类别）
        num_features = self.model.fc.in_features
        self.model.fc = nn.Linear(num_features, 2)

    def forward(self, x):
        return self.model(x)
```

In [20]:
```
x=torch.randn(2,3,224,224)
model = ResNet18()
y = model(x)
y.shape
```

Out[20]:
```
torch.Size([2, 2])
```

训练

In [34]:
```
from torch.optim import lr_scheduler
from torchvision import datasets, transforms
from torch.utils.data import Dataset, DataLoader

root_train = "data/train"
root_test = "data/val"

#将图像的像素值归一化到[-1,1]之间
#由于使用的是 torchvision.models 的预训练模型，所以用 ResNet 训练时的标准化参数
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,


train_transform = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(15), # 随机旋转
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1
    transforms.RandomAffine(5, translate=(0.1, 0.1)),  # 随机仿射变换
    transforms.ToTensor(),
    normalize])

val_transform = transforms.Compose([
```

```
    transforms.Resize((224,224)),
    transforms.ToTensor(),
    normalize])


train_dataset = datasets.ImageFolder(root_train, train_transform)
val_dataset = datasets.ImageFolder(root_test, val_transform)

batch_size=64
train_dataloader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True
val_dataloader = DataLoader(val_dataset, batch_size=batch_size, shuffle=True)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = ResNet18().to(device)

#定义损失函数
loss_fn = nn.CrossEntropyLoss()

#定义优化器（Adam）
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)


#调整学习率，每隔10epoch，变为原来的0.5
lr_scheduler = lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.5)

#训练批次
epoch=20
```

定义训练函数

In [35]:
```
def train(dataloader, model, loss_fn, optimizer):
    loss,current,n = 0.0, 0.0, 0
    for batch_idx, (X, targets) in enumerate(dataloader):
        images, targets = X.to(device), targets.to(device)
        outputs = model(images)
        cur_loss = loss_fn(outputs, targets)
        _, pred = outputs.max(dim=1)
        cur_acc = torch.sum(pred == targets)/ outputs.shape[0]

        #反向传播
        optimizer.zero_grad()
        cur_loss.backward()
        optimizer.step()
        loss += cur_loss.item()
        current += cur_acc.item()
        n = n + 1


    train_acc = current / n
    train_loss = loss / n


    print("train loss: %.4f, train acc: %.4f" % (train_loss, train_acc))
    return train_loss, train_acc
```

定义验证函数

In [36]:

```
def val(dataloader, model, loss_fn, optimizer):
    model.eval()
    loss,current,n = 0.0, 0.0, 0
    with torch.no_grad():
        for batch_idx, (X, targets) in enumerate(dataloader):
            images, targets = X.to(device), targets.to(device)
            outputs = model(images)
            cur_loss = loss_fn(outputs, targets)
            _, pred = outputs.max(dim=1)
            cur_acc = torch.sum(pred == targets)/ outputs.shape[0]
            loss += cur_loss.item()
            current += cur_acc.item()

            n = n + 1

    val_acc = current / n
    val_loss = loss / n
    print("val loss: %.4f, val acc: %.4f" % (val_loss, val_acc))
    return val_loss, val_acc
```

查看模型在训练中的梯度和权重分布

开始训练

In [37]:

```
loss_train_list=[]
acc_train_list=[]
loss_val_list=[]
acc_val_list=[]


# 定义模型保存的文件夹
folder = 'save_model'
if not os.path.exists(folder):
    os.makedirs(folder)

max_acc = 0
for t in range(epoch):
    lr_scheduler.step()
    print('Epoch %d/%d------' % (t + 1, epoch))
    train_loss, train_acc = train(train_dataloader, model, loss_fn, optimizer)
    val_loss, val_acc = val(val_dataloader, model, loss_fn, optimizer)


    loss_train_list.append(train_loss)
    acc_train_list.append(train_acc)
    loss_val_list.append(val_loss)
    acc_val_list.append(val_acc)

    if val_acc > max_acc:
        max_acc = val_acc
        torch.save(model, os.path.join(folder, 'resnet18_best.pt'))
        print(f'Saving model, 第{t + 1}轮...')

# 在最后一次训练结束后保存模型
torch.save(model, os.path.join(folder, 'resnet18_last.pt'))

print('最高精确值:',max_acc)
print("Done!")
```

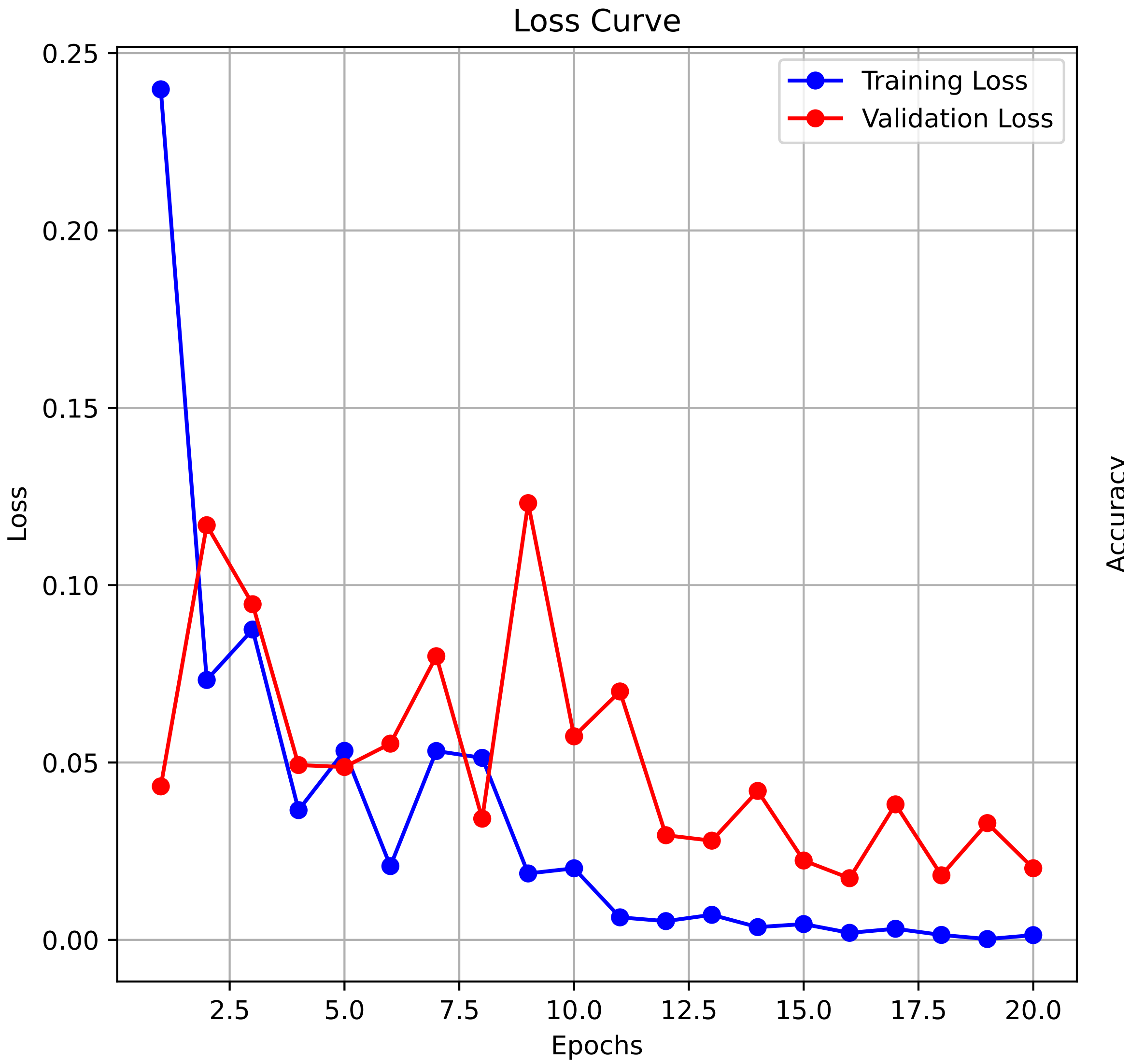
```
Epoch 1/20------
train loss: 0.2398, train acc: 0.9028
val loss: 0.0433, val acc: 0.9812
Saving model, 第1轮...
Epoch 2/20------
train loss: 0.0733, train acc: 0.9714
val loss: 0.1169, val acc: 0.9625
Epoch 3/20------
train loss: 0.0875, train acc: 0.9661
val loss: 0.0946, val acc: 0.9615
Epoch 4/20------
train loss: 0.0366, train acc: 0.9887
val loss: 0.0493, val acc: 0.9792
Epoch 5/20------
train loss: 0.0533, train acc: 0.9748
val loss: 0.0487, val acc: 0.9781
Epoch 6/20------
train loss: 0.0208, train acc: 0.9922
val loss: 0.0553, val acc: 0.9906
Saving model, 第6轮...
Epoch 7/20------
train loss: 0.0532, train acc: 0.9835
val loss: 0.0800, val acc: 0.9625
Epoch 8/20------
train loss: 0.0513, train acc: 0.9783
val loss: 0.0342, val acc: 0.9906
Epoch 9/20------
train loss: 0.0187, train acc: 0.9948
val loss: 0.1231, val acc: 0.9760
Epoch 10/20------
train loss: 0.0202, train acc: 0.9922
val loss: 0.0574, val acc: 0.9917
Saving model, 第10轮...
Epoch 11/20------
train loss: 0.0064, train acc: 0.9991
val loss: 0.0700, val acc: 0.9854
Epoch 12/20------
train loss: 0.0053, train acc: 0.9983
val loss: 0.0295, val acc: 0.9938
Saving model, 第12轮...
Epoch 13/20------
train loss: 0.0071, train acc: 0.9974
val loss: 0.0280, val acc: 0.9938
Epoch 14/20------
train loss: 0.0036, train acc: 0.9991
val loss: 0.0420, val acc: 0.9906
Epoch 15/20------
train loss: 0.0045, train acc: 0.9991
val loss: 0.0224, val acc: 0.9969
Saving model, 第15轮...
Epoch 16/20------
train loss: 0.0020, train acc: 1.0000
val loss: 0.0174, val acc: 0.9906
Epoch 17/20------
train loss: 0.0031, train acc: 0.9991
val loss: 0.0382, val acc: 0.9885
Epoch 18/20------
train loss: 0.0014, train acc: 1.0000
val loss: 0.0182, val acc: 0.9969
Epoch 19/20------
```

```
train loss: 0.0002, train acc: 1.0000
val loss: 0.0329, val acc: 0.9938
Epoch 20/20------
train loss: 0.0013, train acc: 1.0000
val loss: 0.0202, val acc: 0.9938
最高精确值: 0.996875
Done!
```

绘制训练曲线

In [41]:
```python
import matplotlib.pyplot as plt

def plot_training_curves(loss_train_list, acc_train_list, loss_val_list, acc_val
    """
    绘制训练过程中的损失值曲线和精确度曲线
    """
    epochs = range(1, len(loss_train_list) + 1)  # x 轴: 训练轮次

    # 绘制损失值曲线
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)  # 第一张子图
    plt.plot(epochs, loss_train_list, label="Training Loss", marker='o', color='
    plt.plot(epochs, loss_val_list, label="Validation Loss", marker='o', color='
    plt.title("Loss Curve")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.legend()
    plt.grid(True)

    # 绘制精确度曲线
    plt.subplot(1, 2, 2)  # 第二张子图
    plt.plot(epochs, acc_train_list, label="Training Accuracy", marker='o', colo
    plt.plot(epochs, acc_val_list, label="Validation Accuracy", marker='o', colo
    plt.title("Accuracy Curve")
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.legend()
    plt.grid(True)

    # 显示图形
    plt.tight_layout()
    plt.show()
```

In [42]:
```python
plot_training_curves(loss_train_list, acc_train_list, loss_val_list, acc_val_lis
```



测试模型

In [45]:
```python
import random

# 定义一个函数，绘制图像和标题
def show_images_with_predictions(imgs, preds, labels, num_rows, num_cols, classe
    """绘制图像列表，并显示预测结果与真实标签"""
    figsize = (num_cols * scale, num_rows * scale)
    _, axes = plt.subplots(num_rows, num_cols, figsize=figsize)
    axes = axes.flatten()  # 将 axes 从二维数组转换为一维数组
    for i, (ax, img, pred, label) in enumerate(zip(axes, imgs, preds, labels)):
        # 反归一化后转换为 numpy
        img = img.permute(1, 2, 0).numpy()  # (C, H, W) -> (H, W, C)
        ax.imshow(img)
        ax.axes.get_xaxis().set_visible(False)  # 隐藏 X 轴
        ax.axes.get_yaxis().set_visible(False)  # 隐藏 Y 轴
        ax.set_title(f"Truth: {classes[label]} \nPred: {classes[pred]}")
    plt.tight_layout()
    plt.show()

# 加载已训练的模型
model = torch.load('save_model/resnet18_best.pt')
model = model.to(device)
classes = ['cat', 'dog']
```

```python
model.eval()
imgs, preds, labels = [], [], []

# 批量处理数据
with torch.no_grad():
    for _ in range(10):  # 获取前 10 张图片
        i=random.randint(0, len(val_dataset)-1)
        img, label = val_dataset[i][0], val_dataset[i][1]
        img_with_batch = img.unsqueeze(0).to(device)  # 添加 batch 维度
        output = model(img_with_batch)  # 模型推理
        pred = torch.max(output, 1)[1].item()  # 获取预测类别索引

        imgs.append(img)  # 原始图像（张量）
        preds.append(pred)  # 预测结果
        labels.append(label)  # 真实标签

# 显示图片及预测结果
num_rows = 2
num_cols = 5
show_images_with_predictions(imgs, preds, labels, num_rows, num_cols, classes)
```

```
C:\Users\SUN\AppData\Local\Temp\ipykernel_24708\836601485.py:20: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default value for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this experimental feature.
  model = torch.load('save_model/resnet18_best.pt')
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```


Truth: cat
Pred: cat
Truth: cat
Pred: cat
Truth: dog
Pred: dog
Truth: cat
Pred: cat
Truth: dog
Pred: dog
Truth: cat
Pred: cat
Truth: dog
Pred: dog
Truth: dog
Pred: dog