

基于ResNet的船舶图像分类

- 实验步骤、性能评价与分析

- 2023年5月26日 -

一、分类问题介绍（二分类）

提取图像种物体的多
维
度特征，然后根据这些
特征信息来判断图像中
的物体是否属于某一类
别/图像中是否存在某
一类别的物体

分类网络

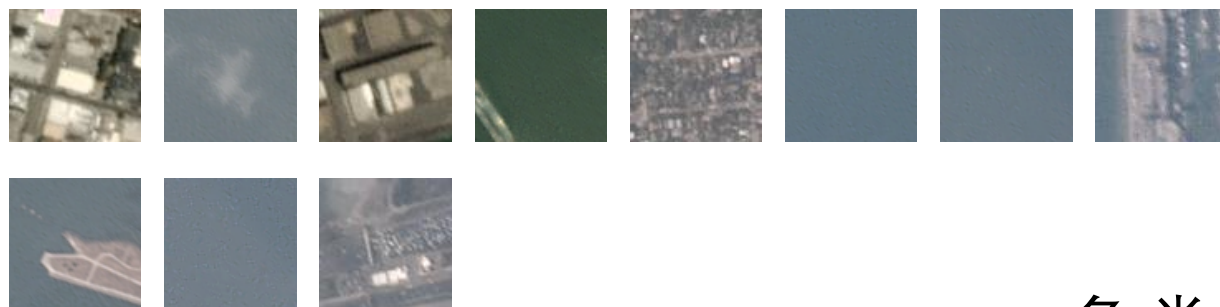


待分类的图像数据



船舶

正类

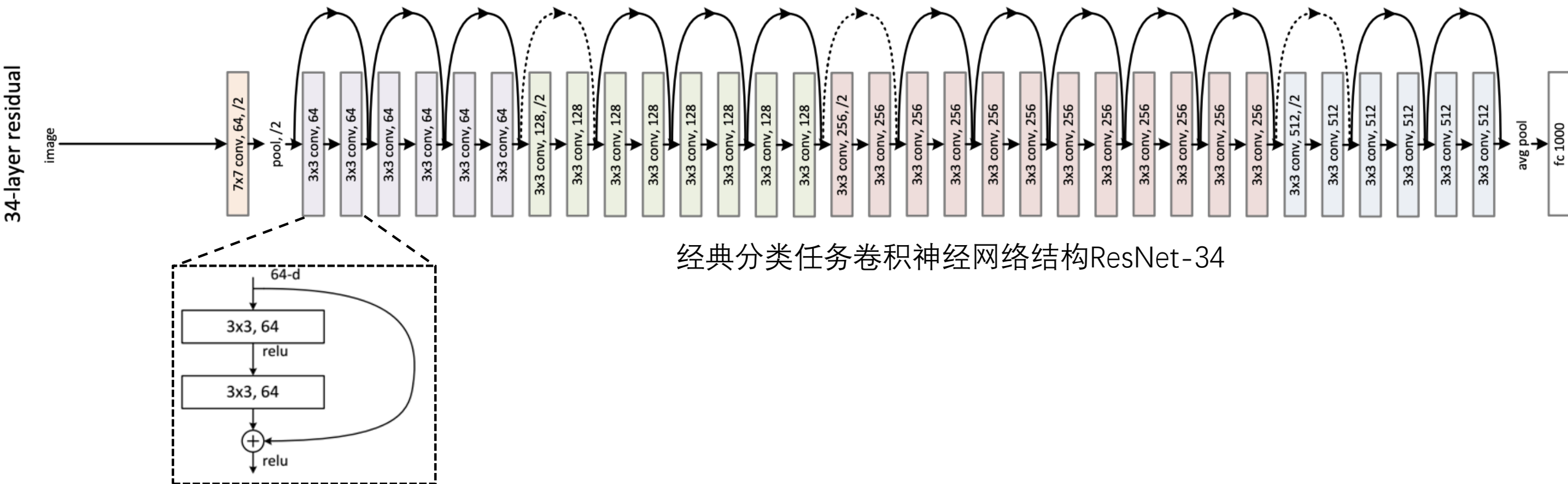


非船舶

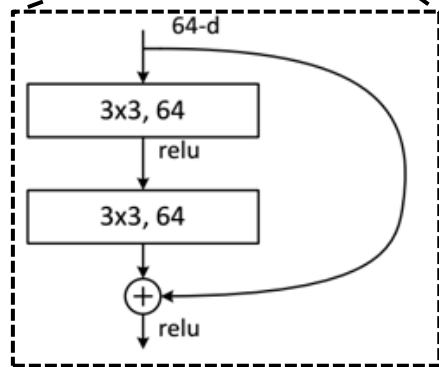
负类

分类结果

二、用于分类的卷积神经网络：ResNet



具有残差连接结构的残差块



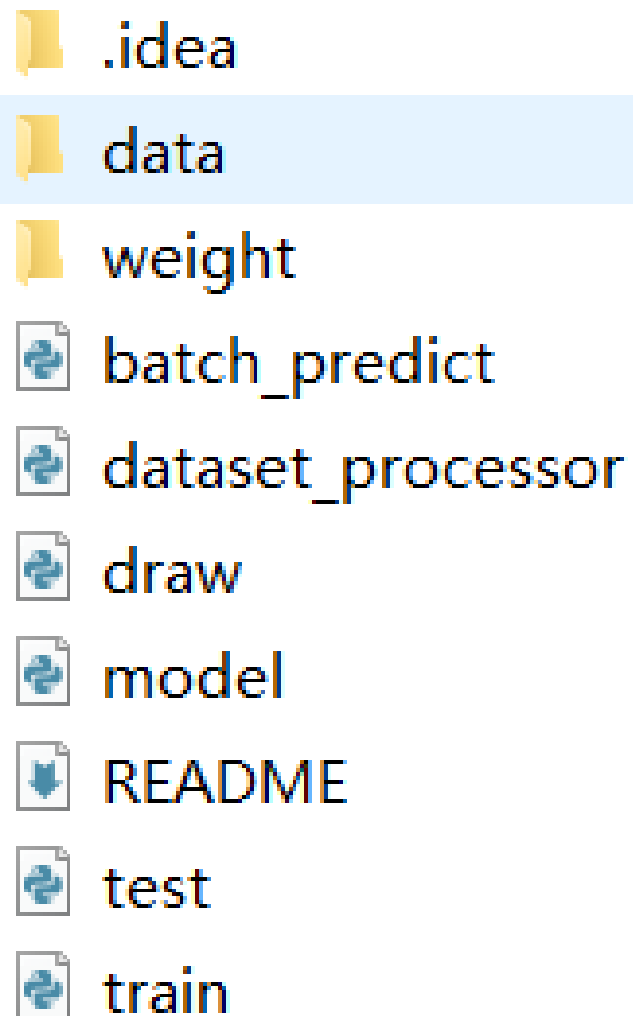
三、训练一个能够分类船舶数据的ResNet分类网络

- 为了训练得到一个能够对我们的船舶数据集进行分类的网络模型（类别为：船舶/非船舶），我们有以下步骤：
 1. 下载并部署好分类网络模型代码；
 2. 对数据集进行合理的训练集/测试集划分，存入对应目录；
 3. 使用训练代码在训练集上训练模型，观察训练损失直到训练收敛；
 4. 训练完毕后，使用训练好的模型在测试集上测试网络分类性能；
 5. 分析测试结果。

※ 代码所需Python版本最低为3.7，其他环境包配置放在项目文件夹下的requirements.txt中，可以使用pip进行环境安装。

三、训练ResNet分类网络：代码结构（PyTorch框架下）

以下是一个十分简单的ResNet分类网络的代码结构，我们的网络训练将用到这些代码。



- data文件夹：用于存储原始数据，以及存储划分好的训练集/测试集数据；
- weight文件夹：用于存储权重文件（如预训练权重）；
- batch_predict.py：在大量图片上测试网络分类预测性能的代码。在整个代码中主要起验证性作用（第六部分会讲）；
- data_processor.py：用于对原始数据进行训练集/测试集划分；
- draw.py：用于绘制损失函数曲线和准确度曲线；
- model.py：定义了ResNet网络具体结构的代码；
- test.py：用于测试网络性能、计算模型分类指标；
- train.py：用于训练网络的代码。

三、训练ResNet分类网络：数据准备



划分数据集必须注意以下几点:

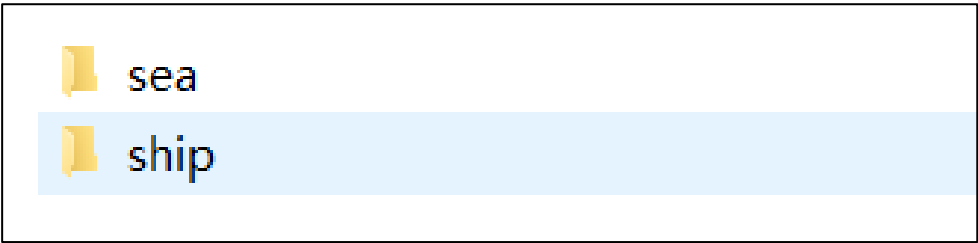
1. 不同类别数据**不允许有交叉混合**, 确保每个类别的图像数据确属于本类别;
2. 划分训练集/测试集从原始数据集中采样时, **必须保证随机从各个类别的数据中采样**;
3. 训练集和测试集的比例有多种比例组合, 如5:5、6:4、7:3、8:2、9:1等。对于数据量不大的情况, 一般会选择7:3以及其他训练集占比更大的比例组合 (如本次实验中选择的8:2) 。

三、训练ResNet分类网络：数据准备（PyTorch代码框架下）

• 下载好原始数据集后，数据集中的内容一般放置在代码文件夹下的 data 路径中，如下图所示：

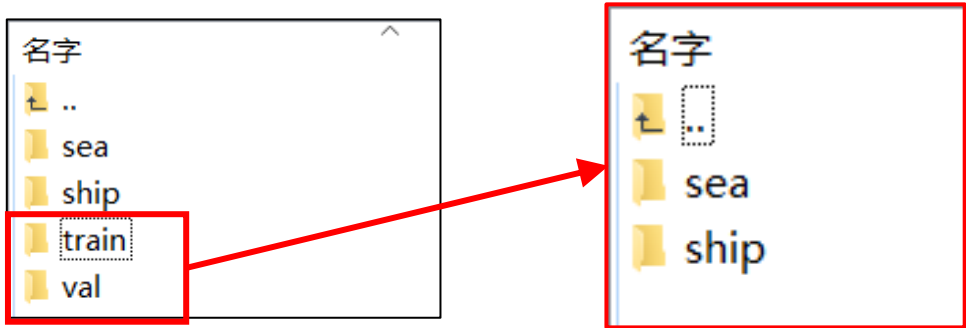
名称	修改日期	类型	大小
.idea	2022/4/28 12:24	文件夹	
data	2022/4/28 10:31	文件夹	
batch_predict	2022/4/28 11:26	Python 源文件	4 KB
dataset_processor	2022/4/28 11:05	Python 源文件	3 KB

1. 未划分的数据集的组织形式如下图所示：



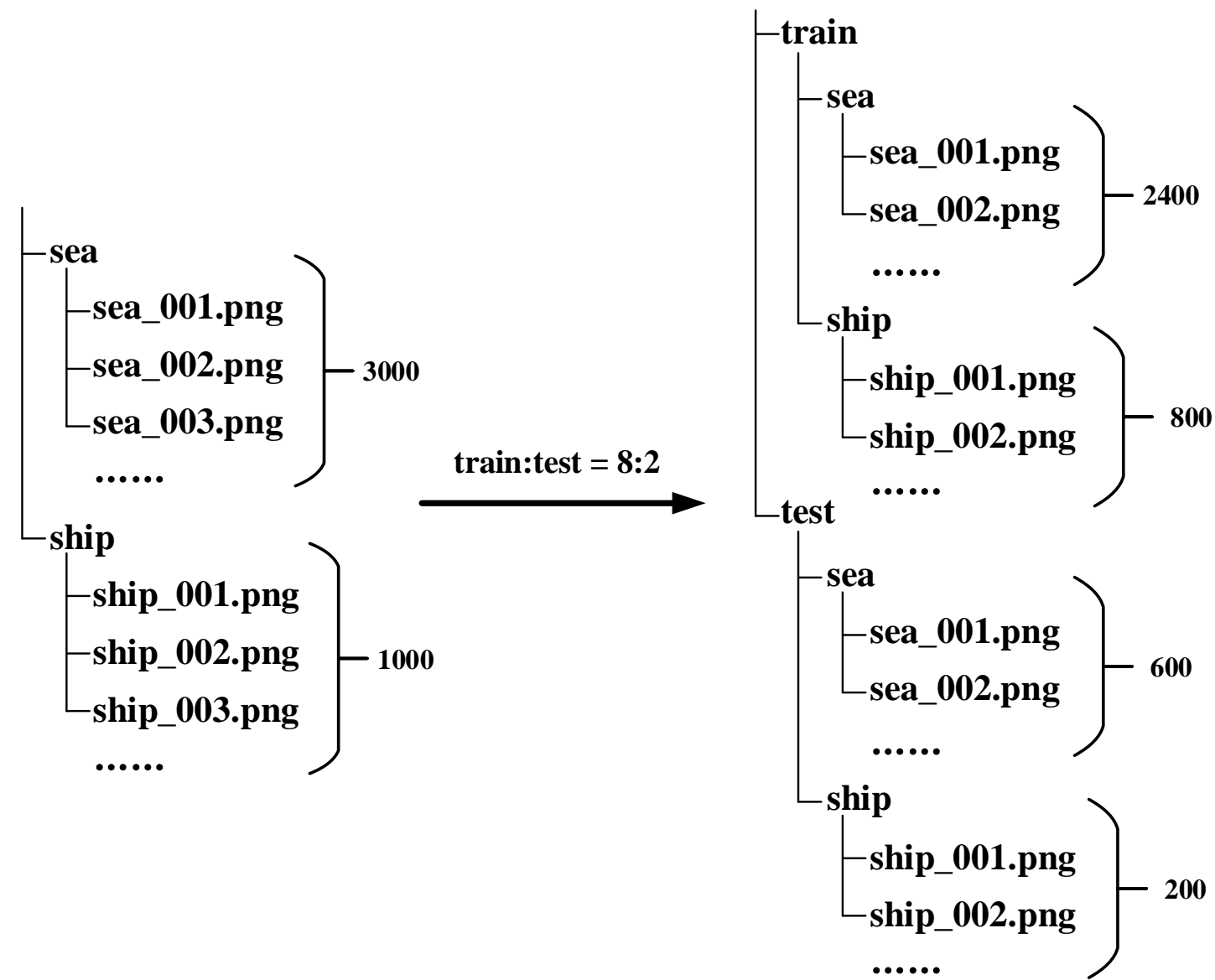
不同类别的图像数据分别存储在不同的文件夹中
(ship代表船舶数据， sea代表非船舶数据)。

2. 经过划分的数据组织形式如下图所示：



经过划分的数据集被分为训练集（train文件夹）以及测试集（val文件夹）。这两个文件夹内部的图像依旧根据分类的不同，分为sea和ship两个文件夹，用于区别不同类型的数据。

三、训练ResNet分类网络：数据准备



代码设计思路：

1. 获取船舶/非船舶类别中图像数据的总量；
2. 每个类别的数据总量乘以0.8即为该类别数据的训练数据量，剩下的数据均作为测试集；
3. 根据以上获得的每个类别中训练集数量，随机从各个类别的数据中采样训练数据；
4. 训练数据采样后，每个类中的剩下的数据即为测试集。将训练集和测试集单独存储起来；
5. 新建存放训练集和测试集的文件夹，分别在这两个文件夹中再新建对应的类别文件夹；
6. 将不同类别划分出来的训练集/测试集图像数据分别复制到对应的文件夹中。

三、训练ResNet分类网络：训练代码（PyTorch代码框架下）

训练代码主要由以下几部分组成：

1. 加载组织好的训练数据
2. 建立网络模型
3. 初始化训练必要的优化器、损失函数和迭代器
4. 根据迭代设计训练网络，保存网络权重文件

三、训练ResNet分类网络：训练代码（PyTorch代码框架下）

数据加载

使用PyTorch中提供的接口，只需要提供train文件夹的路径，即可自动加载划分好类别的训练数据。

```
data_root = args.dataset_root 即train和val文件夹所在的 data 文件夹
assert os.path.exists(data_root), "{} path does not exist.".format(data_root)
# 训练集
train_data = datasets.ImageFolder(root=os.path.join(data_root, "train"),
                                  transform=data_transform["train"])
```

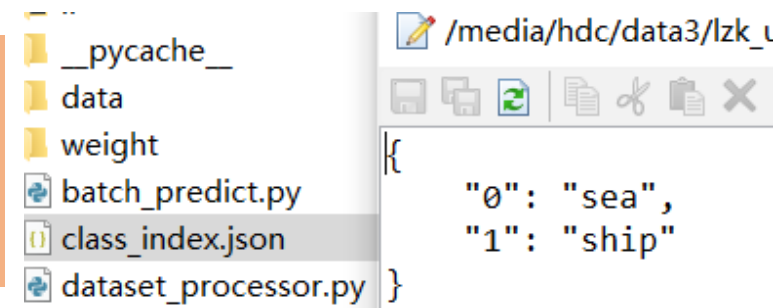
直接通过train文件夹的路径来加载训练数据

设置训练集dataloader

```
train_dataloader = DataLoader(
    train_data,
    batch_size=batch_size,
    shuffle=True,
    num_workers=nw
)
train_num = len(train_data)
```

加载好的训练数据还会进一步根据批大小等参数做进一步划分，以便于后续迭代训练的时候能够直接调用数据块

训练代码会自动根据文件夹的分类，在代码的根目录保存一个名字为class_index.json的文件，用于存储类别名字和类别序号的对应信息。



三、训练ResNet分类网络：训练代码（PyTorch代码框架下）

数据加载

测试数据的加载方式与训练数据的加载方式相同，只需要将测试数据的文件夹路径名规定好即可。

```
val_data = datasets.ImageFolder(  
    root=os.path.join(data_root, "val"),  
    transform=data_transform["val"]  
)  
val_num = len(val_data)  # 验证集长度  
# 设置验证集dataloader  
val_dataloader = DataLoader(  
    val_data,  
    batch_size=batch_size,  
    shuffle=False,  
    num_workers=nw  
)
```

加载测试集的方式和训练数据类似，
只需要把测试集的路径定义好

数据集（Dataset）和数据集加载器（DataLoader）的区别：

前者用于保存数据集中所有图像的相关信息（例如路径、文件名、分类标签等）；后者主要用于网络训练或测试，用于根据一些超参数（如批量大小）划定每一批量中的图像数量用于训练和测试等。

三、训练ResNet分类网络：训练代码（PyTorch代码框架下）

建立网络模型

通过已定义好的网络模型类resnet34建立网络对象net，然后加载网络的预训练权重，再修改分类任务中类别的数量。

```
net = resnet34()
```

 建立ResNet-34网络模型

```
weight_path = args.pretrained_model
```

 定义网络需要加载的预训练权重的路径

```
assert os.path.exists(weight_path), "weight file {} is not exists".format(weight_path)
```

```
net.load_state_dict(torch.load(weight_path, map_location=device))
```

 将权重加载到网络模型中

```
# change fc layer structure, 改变全连接层结构
```

```
# 因为在resnet网络，默认进行1000数的分类，我们花数据集只需要分5类
```

```
inchannels = net.fc.in_features
```

```
net.fc = nn.Linear(inchannels, args.num_classes)
```

```
net.to(device) # 将模型放入设备（cpu或者GPU）中
```

```
# print(net.fc)
```

原始ResNet是针对一个1000类分类任务设计的，网络最后一层的输出维度是1000。我们的分类任务只有2类，所以需要更改最后一层的输出维度为2。
args.num_classes是一个定义分类数量的变量。

三、训练ResNet分类网络：训练代码（PyTorch代码框架下）

初始化训练网络模型的超参数

训练网络的重要超参数包括epoch数量、初始学习率、批量大小等。除此之外我们还需要定义训练网络的优化器类型，以及损失函数的类型。

```
parser.add_argument(
    '--epoch',
    type=int,
    default=200
)
parser.add_argument(
    '--lr',
    type=float,
    default=0.001
)
parser.add_argument(
    '--batch_size',
    type=int,
    default=32
)
```

Epoch数量
(相当于不断迭代训练的回合数)

学习率（梯度学习中的一个重要优化参数）

批大小（网络一次学习并计算损失的样本数量）

```
params = [p for p in net.parameters() if p.requires_grad]
learning_rate = args.lr
optimizer = torch.optim.Adam(params, lr=learning_rate)
# 损失函数，使用交叉熵损失
loss_function = nn.CrossEntropyLoss()
```

交叉熵损失
(通过损失函数来不断优化网络模型以达到收敛)

Adam优化器
(一种基于梯度优化方法的网络学习策略)

参数调整策略：

1. Epoch：可以先从50开始尝试。如果50个epoch过后方法依然没有收敛，则说明50 epoch不够，需要更多的epoch来迭代训练网络；
2. 学习率：如果损失收敛太慢，可以尝试稍稍增大学习率（如从0.0001增大为0.001）；如果训练过程中损失出现NaN值，则需要减小学习率（如0.01到0.001）；
3. 批大小：一般设置值在4~32之间，取决于硬件条件。如果硬件条件不足/训练太慢，可以考虑适当降低批大小（以1/2的比例降低）

三、训练ResNet分类网络：训练代码（PyTorch代码框架下）

迭代训练网络

训练网络的重要超参数包括epoch数量、初始学习率、批量大小等。除此之外我们还需要定义训练网络的优化器类型，以及损失函数的类型。

```
for epoch in range(epochs):
    # 训练
    net.train()
    running_loss = 0.0
    # file = sys.stdout的意思是, print函数会将内容打印输出到标准输出流(即 sys.stdout)
    # train_bar是tqdm用于显示进度
    train_bar = tqdm(train_data_loader, file=sys.stdout)
    # data = train_data[0]
    for step, data in enumerate(train_bar):
        images, labels = data
        # images是一个batch的图片, [batch_size, 224, 224]
        # labels是每个图片的标签, [batch_size, ], 如[1, 0, 4], 数字代表类别
        optimizer.zero_grad()
        pre = net(images.to(device))
        # print(pre)
        loss = loss_function(pre, labels.to(device))
        loss.backward()
        optimizer.step()
    # loss 统计
    running_loss += loss.item()
```

将网络设定为训练模式

train_data_loader中存放了由批大小分组的训练数据

循环每次都拿出一组数据用于训练，数据包括图像数据及其分类标签

将图像数据输入网络，获得网络的分类预测结果

计算损失

反向传播过程

左侧图片代表了训练网络时的循环代码结构。

- ① 每个epoch中，所有训练数据会按照批大小划分不同的数据组，然后每个数据组会被一同输入到网络中共同训练；
- ② 在每组数据送入网络之前，优化器会重置其所记录的梯度信息，用于重新记录某组数据在网络上梯度结果以针对地优化网络；
- ③ 网络输出结果后，该结果会被送到损失函数中计算对应的损失，然后网络会进行损失的梯度反向传播，紧接着使用优化器根据这些反向传播结果优化网络权值。

训练过程中损失函数的值理论上应当快速下降，然后逐渐收敛。当损失函数收敛时，模型基本训练完毕。

三、训练ResNet分类网络：训练代码（PyTorch代码框架下）

训练时对网络性能的验证（在训练的循环里）

网络在训练的过程中，还可以在每个epoch训练完毕后，使用刚刚训练的模型在测试集上直接测试效果。这种做法可以直观地看到每个epoch训练结束后的网络在测试集上的性能效果，反映训练情况。

验证模式

net.eval() ← 将网络调整为验证模式

acc = 0.0 # 预测正确个数

with torch.no_grad():

val_bar = tqdm(val_dataloader, file=sys.stdout)

for val_d in val_bar:

val_image, val_label = val_d

output = net(val_image.to(device))

torch.max比较后，第0个是每个最大值，第1个是最大值的下标，所以取第1个

predict_y = torch.max(output, dim=1)[1]

acc += torch.eq(predict_y, val_label.to(device)).sum().item()

val_bar.desc = "valid epoch[{} / {}]".format(epoch + 1, epochs)

val_accurate = acc / val_num

acc_r.append(val_accurate)

print('[epoch %d] train_loss: %.3f val_accuracy: %.3f' %

(epoch + 1, running_loss / train_step, val_accurate))

以类似的方式，
分组将测试集数
据送入网络，然
后获得网络的分
类预测结果

非船舶

0.56	0.10	0.99	0.74	0.95
0.98	1.00	0.71	0.97	0.33

船舶

网络的输出是一组分类的权值。在二分类任务中，我们可以默认输出权值较大的那一类为预测的类别结果，然后输出该类别的下标作为分类结果。

计算分类正确的样本数量。如果预测结果和真实结果相同，那么返回True，否则返回False。最后将所有的True值相加即得网络在该组数据上的准确预测样本数量。

计算完所有组的数据后，所有的正确分类样本除以总样本数量即可得到分类的准确度（Accuracy）

三、训练ResNet分类网络：训练代码（PyTorch代码框架下）

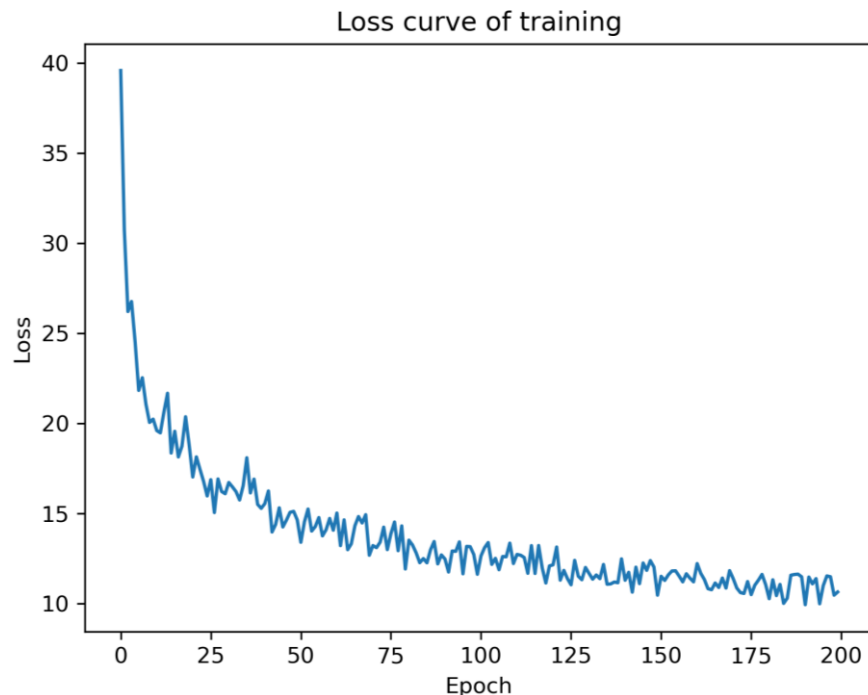
保存网络最好性能时的权重文件

在每个epoch里网络都会计算分类的精确度，如果某一个epoch中的精确度高于之前的最高精确度，那么网络就将这一次epoch训练的网络模型权重保存下来。这个最优权重可以用于后续的网络测试。

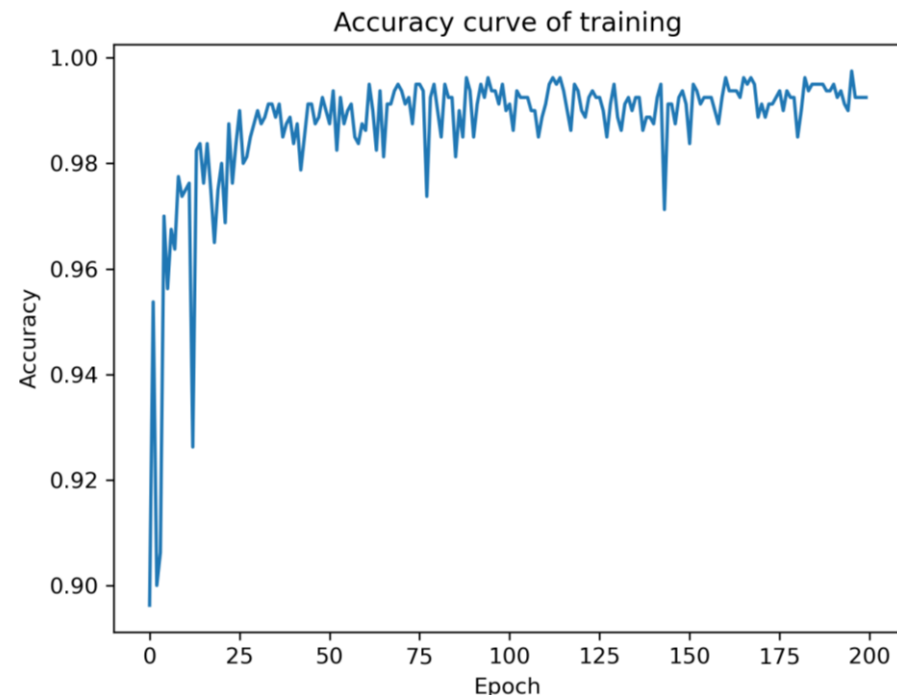
```
if val_accurate > best_acc:
    best_acc = val_accurate
    torch.save(net.state_dict(), save_path)
```

保存网络训练时的损失和精确度数据

后期可以根据这些数据来绘制损失曲线以及精确度曲线，以可视化的角度来观察网络训练的过程。正常收敛曲线是一个逐渐平滑的过程，如果发现可以继续收敛，那就说明epoch可以设置的更大。



损失曲线



准确度曲线

四、分类任务的性能评价标准

一、混淆矩阵 Confusion Matrix

- TP: True Positive 真阳性: 正样本, 预测为正
- FP: False Positive 假阳性: 正样本, 预测为负
- TN: True Negative 真阴性: 负样本, 预测为负
- FN: False Negative 假阴性: 负样本, 预测为正

	预测为正样本	预测为负样本
标签为正样本	TP	FN
标签为负样本	FP	TN

4. F1-Score (F1分数) : 综合精确度和召回率的指标

$$F1 = \frac{2 * precision * recall}{precision + recall}$$

二、常见二分类评价指标

1. Accuracy (准确度) : 所有正确分类的样本数量占所有样本数量的比例

$$Accuracy = \frac{TN + TP}{FP + TN + TP + FN}$$


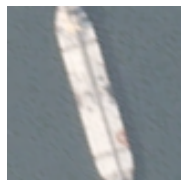

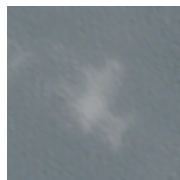
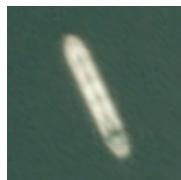
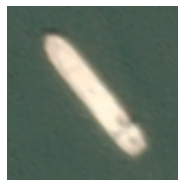
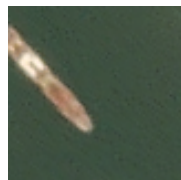


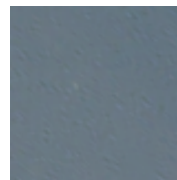
2. Precision (精确度) : 正确分类的正样本个数占所有预测为正样本个数的比例

$$Precision = \frac{TP}{TP + FP}$$

3. Recall (召回率) : 正确分类的正样本个数占所有实际正样本个数的比例

$$Recall = \frac{TP}{TP + FN}$$

四、分类任务的性能评价标准

10个样本										
真实分类	0	1	1	0	1	1	1	0	1	0
										
预测分类	1	1	1	0	1	1	0	1	1	0
	FP	TP	TP	TN	TP	TP	FN	FP	TP	TN

$$\text{Accuracy} = 7 / 10 = 0.70$$

$$\text{Precision} = 5 / 7 = 0.71$$

$$\text{Recall} = 5 / 6 = 0.83$$

$$\text{F1} = 2 * 0.71 * 0.83 / (0.71 + 0.83) = 0.77$$

五、测试代码

1. 测试代码中比较重要的参数主要是以下几个：

① 测试集的图片路径；

```
parser.add_argument(  
    '--img_paths',  
    type=str,  
    default='data/val/*/*.png'  
)
```

② 之前训练好的权重文件的路径；

```
parser.add_argument(  
    '--weight_path',  
    type=str,  
    default='weight/resnet34_best.pth'  
)
```

③ 类别数量和批大小；

```
parser.add_argument(  
    '--num_classes',  
    type=int,  
    default=2  
)
```

```
parser.add_argument(  
    '--batch_size',  
    type=int,  
    default=8  
)
```

2. 测试代码中关键的代码步骤有如下几个：

① 加载所有的测试及图像的路径，后需要根据这些路径读取图片；

```
test_paths = args.img_paths  
img_path_list = glob(test_paths, recursive=True)
```

② 根据测试集图片的类别属性，把每一个测试样本的类别序号存储在一个列表中，方便后续计算指标；

```
# read class_indict  
json_path = args.cls_index  
assert os.path.exists(json_path), f"file: '{json_path}' dose not exist."  
json_file = open(json_path, "r")  
class_indict = json.load(json_file)  
class_indict_reverse = {v: k for k, v in class_indict.items()}  
  
ground_truths = [int(class_indict_reverse[x.split('/')[-2]])  
                  for x in img_path_list]
```

五、测试代码

2. 测试代码中关键的代码步骤有如下几个：

③ 定义所有样本的TP/TN/FP/FN，方便后续计算；

```
TPs, TNs, FPs, FNs = 0, 0, 0, 0
```

④ 比对一批(batch)图片的真实类别标签和预测的类别标签，然后把一批的比对结果加到总的数量中；

```
TP = sum([1 for g, p in zip(batch_ground_truths, batch_predicted_cls) if g == p == 1])
TN = sum([1 for g, p in zip(batch_ground_truths, batch_predicted_cls) if g == p == 0])
FP = sum([1 for g, p in zip(batch_ground_truths, batch_predicted_cls) if g == 0 and p == 1])
FN = sum([1 for g, p in zip(batch_ground_truths, batch_predicted_cls) if g == 1 and p == 0])
```

```
TPs += TP
TNs += TN
FPs += FP
FNs += FN
```

⑤ 在所有batch都测试完毕后，使用总的TN/TP/FP/FN来计算各个指标；

```
accuracy = (TNs + TPs) / len(img_path_list)
precision = TPs / (TPs + FPs)
recall = TPs / (TPs + FNs)
f1 = 2 * precision * recall / (precision + recall)
```

可供参考的船舶分类模型的分类指标

Accuracy	Precision	Recall	F1-Score
0.9975	0.9950	1.0000	0.9975

六、验证性代码：找出低置信度分类图像

验证性代码的作用主要是找出分类置信度低于某个阈值的图像。

即使我们的模型的测试指标很高，但是我们的目标永远都是如何做到所有指标都为1（即模型的泛化性能达到完美）。每一张图片的分类结果都是依据其在某一类上的分类置信度最高而得到的，这个验证性代码想要做的就是将所有分类置信度不是1的图像抓出来（包括船舶类和非船舶类）。这些图像找出来之后，我们可以分析为什么这些图像的分类置信度不高，然后根据这样的结果作进一步分析，该如何改进网络结构以进一步提高分类指标。





























在batch_predict.py代码中有一个新添加的变量，这个变量定义的就是分类置信度的阈值。如果一张图片的分类置信度低于这个阈值，那么代码就认为这张图像的分类结果不够可信。

随后这张图片会被复制到一个在根目录的新文件夹 low_probs 里。这里面的文件都是分类置信度低于阈值的图像，并且文件名的结尾处还添加了该图像的类别预测名和置信度。

```
parser.add_argument(  
    '--conf_thr',  
    type=float,  
    default=1.  
)
```

```
if os.path.exists('low_probs/'):   
    shutil.rmtree('low_probs/')  
os.makedirs('low_probs/')  
for low_conf_pred in lower_conf_preds:  
    shutil.copyfile(  
        low_conf_pred[0],  
        os.path.join('low_probs',  
                      f'{Path(low_conf_pred[0]).stem}_{low_conf_pred[1]}_{low_conf_pred[2]:.4f}.png')  
    )  
print(f"Probability lower than {args.conf_thr} are copied to 'low_probs/' directory.")
```


六、验证性代码：找出低置信度分类图像

名字	大小	已改变	权限	拥有者
..		2022/4/28 17:44:06	rw-rwxr-x	hdc
 Sea_20160622_170157_0c64_-122.34994908648167_37.78274932761714_sea_1.0000.png	9 KB	2022/4/28 17:44:06	rw-rw-r--	hdc
 sea_20160710_182140_0c78_-122.36167510647154_37.72651739424844_sea_0.9748.png	12 KB	2022/4/28 17:44:06	rw-rw-r--	hdc
 Sea_20160820_233143_0c53_-122.36042556233372_37.79259487116029_sea_1.0000.png	9 KB	2022/4/28 17:44:06	rw-rw-r--	hdc
 sea_20160905_193457_1_0c37_-122.46023886886336_37.77216594085852_sea_1.0000.png	12 KB	2022/4/28 17:44:06	rw-rw-r--	hdc
 Sea_20160905_193457_1_0c37_-122.48541134123563_37.71327812922112_sea_0.9976.png	11 KB	2022/4/28 17:44:06	rw-rw-r--	hdc
 sea_20160905_193458_0c37_-122.340008531414_37.74740774072642_sea_1.0000.png	9 KB	2022/4/28 17:44:06	rw-rw-r--	hdc
 sea_20161011_185736_0c72_-122.33929242287445_37.77049533872591_sea_0.9998.png	9 KB	2022/4/28 17:44:06	rw-rw-r--	hdc
 sea_20161018_192144_0c24_-122.4790600312469_37.85209765752398_sea_0.9999.png	13 KB	2022/4/28 17:44:06	rw-rw-r--	hdc
 sea_20161022_180611_0e19_-122.1130711132218_37.73704965610468_sea_1.0000.png	12 KB	2022/4/28 17:44:06	rw-rw-r--	hdc
 sea_20161103_195402_1_0c24_-122.16304622327651_37.8674721477702_sea_1.0000.png	12 KB	2022/4/28 17:44:06	rw-rw-r--	hdc
 sea_20161116_180802_0e14_-122.2722366298265_37.80376400338651_sea_1.0000.png	14 KB	2022/4/28 17:44:06	rw-rw-r--	hdc
 sea_20161116_180802_0e14_-122.3750085568293_37.80073082520651_sea_0.9999.png	12 KB	2022/4/28 17:44:06	rw-rw-r--	hdc
 sea_20161116_180802_0e14_-122.49999218351591_37.88828552016498_sea_1.0000.png	12 KB	2022/4/28 17:44:06	rw-rw-r--	hdc
 sea_20161116_180803_0e14_-122.49593166657236_37.748015479953914_sea_1.0000.png	13 KB	2022/4/28 17:44:06	rw-rw-r--	hdc
 sea_20161116_180804_0e14_-122.48298739296371_37.684929808845375_sea_1.0000.png	12 KB	2022/4/28 17:44:06	rw-rw-r--	hdc
 sea_20161202_180734_0e30_-122.13331872158157_37.73795736932523_sea_1.0000.png	13 KB	2022/4/28 17:44:06	rw-rw-r--	hdc
 sea_20161202_180734_0e30_-122.14901617140781_37.761622800589684_sea_1.0000.png	13 KB	2022/4/28 17:44:06	rw-rw-r--	hdc
 sea_20161202_180734_0e30_-122.16246148728032_37.78345849707674_sea_1.0000.png	13 KB	2022/4/28 17:44:06	rw-rw-r--	hdc
 sea_20161202_180734_0e30_-122.34752499991899_37.767837278289846_sea_0.9999.png	11 KB	2022/4/28 17:44:06	rw-rw-r--	hdc
 sea_20161212_180855_0e30_-122.51663917013924_37.883433955397216_sea_0.9996.png	11 KB	2022/4/28 17:44:06	rw-rw-r--	hdc
 sea_20161212_180856_0e30_-122.43019018359024_37.808203095072_sea_1.0000.png	11 KB	2022/4/28 17:44:06	rw-rw-r--	hdc
 sea_20161218_180845_0e26_-122.47539012073287_37.802112899211274_sea_1.0000.png	13 KB	2022/4/28 17:44:06	rw-rw-r--	hdc
 sea_20161218_180845_0e26_-122.47995641469784_37.802133144979855_sea_1.0000.png	12 KB	2022/4/28 17:44:06	rw-rw-r--	hdc
 sea_20161218_180845_0e26_-122.53009410357646_37.81683689258303_sea_1.0000.png	13 KB	2022/4/28 17:44:06	rw-rw-r--	hdc
 sea_20161218_180846_0e26_-122.41511264989016_37.69315727796003_sea_1.0000.png	13 KB	2022/4/28 17:44:06	rw-rw-r--	hdc
 sea_20161218_180846_0e26_-122.49705079949882_37.71346355501664_sea_1.0000.png	12 KB	2022/4/28 17:44:06	rw-rw-r--	hdc
 sea_20161227_180931_0e30_-122.37023118966783_37.805518360485074_sea_0.9999.png	12 KB	2022/4/28 17:44:06	rw-rw-r--	hdc
 sea_20161227_180931_0e30_-122.47546630554815_37.80097758659708_sea_0.9998.png	13 KB	2022/4/28 17:44:06	rw-rw-r--	hdc
sea_20161227_180931_0e30_-122.47767286303282_37.81185719305492_sea_1.0000.png	11 KB	2022/4/28 17:44:06	rw-rw-r--	hdc
sea_20161227_180931_0e30_-122.50069033170207_37.78478243871842_sea_1.0000.png	13 KB	2022/4/28 17:44:06	rw-rw-r--	hdc