人工智能大作业 2——海洋动物分类

彭程　2020011075
清华大学　自动化系　自 02 班

日期：2022 年 12 月 14 日

摘　　要

本文为 2022 秋《人工智能基础》大作业二的实验报告。本次作业使用深度学习框架 pytorch 编程实现海洋数据分类任务。完成了全部必做和绝大多数选做任务，以及一些感兴趣的附加拓展。在自己设计的模型上通过参数调整可以达到 60% 左右的正确率，使用预训练模型可达到 80% 左右的正确率。

关键词： Image Classification， Pytorch， Deep Learning

1　前言

1.1　问题描述

本次大作业是基于海洋生物图像数据集（包括 19 类海洋生物，共计 11742 张图像），完成海洋动物图片分类任务。

在本次任务中，我使用 pytorch 框架设计训练流程，使用 wandb 平台记录实验数据，尝试了：自己搭建简单的卷积神经网络，复现经典卷积神经网络，改进经典卷积神经网络，使用预训练的卷积神经网络模型进行 fine-tune 等多种方式完成本次分类任务。同时还尝试了不同的学习率策略、超参数选择、优化器选择等训练技巧的实际作用。

1.2　实验任务

对于实验要求，做如下说明：

- 任务 1
  - 设计一个卷积神经网络模型来解决该分类问题；见 Sec 3.2，Sec 3.3
  - 采用一个深度学习框架来实现你设计的模型；Pytorch
  - 提出方法并测试模型；见 Sec 2.2，Sec 3
  - 提出指标并评价模型。见 Sec 3
- 任务 2（选）
  - 输入数据的处理；见 Sec 2.3
  - 训练中各种超参数的设置；见 Sec 4.1
  - 优化器的选择；见 Sec 4.1
  - 模型结构的改进；见 Sec 3.3
  - 设计方法对模型参数进行压缩；未选做该任务
  - 设计可视化方法，验证模型的可靠性，并尝试对模型决策过程进行解释见 Sec 5
- 其他
  - 对比经典神经网络性能 见 Sec 3
  - 预训练网络上进行 Fine-Tune 见 Sec 4.2

2 数据处理

2.1 数据不均衡

对于实验数据，首先经过观察，具有类别不均衡的问题，海龟类有近 2k 张图片，其余类大约每类 500 张。为解决此问题，通常可以采用：减少数量多的样本（“欠采样”），或者增加数量少的样本（“过采样”），或者调整每一类别学习率比重。本次实验中，我采用了欠采样来解决该问题，将海龟图像减少到 500 张。

2.2 数据集划分

本次实验，由于样本量比较小，我选择使用独立测试集的测试方法进行测试，即在数据划分时，将训练集、验证集、测试集按照 8:1:1 进行划分，划分时保证各类别的比例均衡。最终分别有 8258、1040、1041 张图片。实验时在训练集上进行训练，每个 epoch 后在验证集上验证模型性能决定是否替换或保留，实验结束后，在测试集上进行最终的评价和横向比较。

2.3 数据增强

在数据增强方面，经过查询和尝试，最终我采用了“静态数据增强 + 动态数据增强”的方式。静态数据增强是指，将原始训练集经过旋转、平移、缩放等操作后保存下来，作为训练集的一部分直接增加了训练图片数量。动态数据增强是指，在训练的每个 epoch 加载图像时，利用 transform 工具对图像进行裁剪、旋转、归一化等一系列操作，这样增加了图片的多样性却不改变每个 epoch 的图片数量。具体如下代码所示：（以下讨论的数据增强均只在训练集上使用，验证集和测试集只进行图片尺寸的改变和归一化）

静态数据增强

```
def image_pro(path):
  image0 = Image.open(path)
  # 旋转
  image1 = Image.open(path)
  image1 = image1.rotate(45)
  image1 = image1.resize((256, 256))
  # 镜像
  image2 = Image.open(path)
  image2 = image2.transpose(Image.FLIP_LEFT_RIGHT)
  image2 = image2.resize((256, 256))
  # 随机裁剪
  image3 = Image.open(path)
  image3 = image3.resize((384, 384))
  x_min = random.randint(0, 128)
  y_min = random.randint(0, 128)
  box = (x_min, y_min, x_min + 256, y_min + 256)
  image3 = image3.crop(box)
  return image0, image1, image2, image3
```

非动态数据增强

```
transform = transforms.Compose([
        transforms.Resize((224, 224)),
```

```
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.431, 0.402, 0.293], std=[0.181, 0.184, 0.179]),
    ])
```

动态数据增强

```
transform_pro = transforms.Compose([
        transforms.Resize((300, 300)), # 缩放
        transforms.RandomResizedCrop(250, scale=(0.5, 1.0), ratio=(0.75,
            1.33),interpolation=2),
        transforms.RandomHorizontalFlip(p=0.5),
        transforms.RandomRotation(45),
        transforms.RandomGrayscale(p=0.5), #随机灰度化
        transforms.Resize((224, 224)),
        transforms.ToTensor(), # 转为张量，同时归一化
        transforms.Normalize(mean=[0.431, 0.402, 0.293], std=[0.181, 0.184, 0.179]), # 标准化
    ])
```

我用模型 MyNetPro(参见模型说明的部分) 进行了对比实验。分别使用经过数据增强和未经过数据增强的训练集进行训练。经过试验，未经过数据增强的网络表现明显差于经过数据增强的网络（见 Tab 1，Fig 1）。经过数据增强的训练模型在正确率等评价指标上均有更好的表现。从曲线也可以看出，尽管最终都发生过拟合，但数据增强后的模型明显过拟合程度更低。证实了数据增强的有效性。

model	acc	precision	recall	f1
MyNetPro	0.54	0.586	0.757	0.658
MyNetPro-LessData	0.46	0.542	0.561	0.537

表 1: 数据增强前后对比

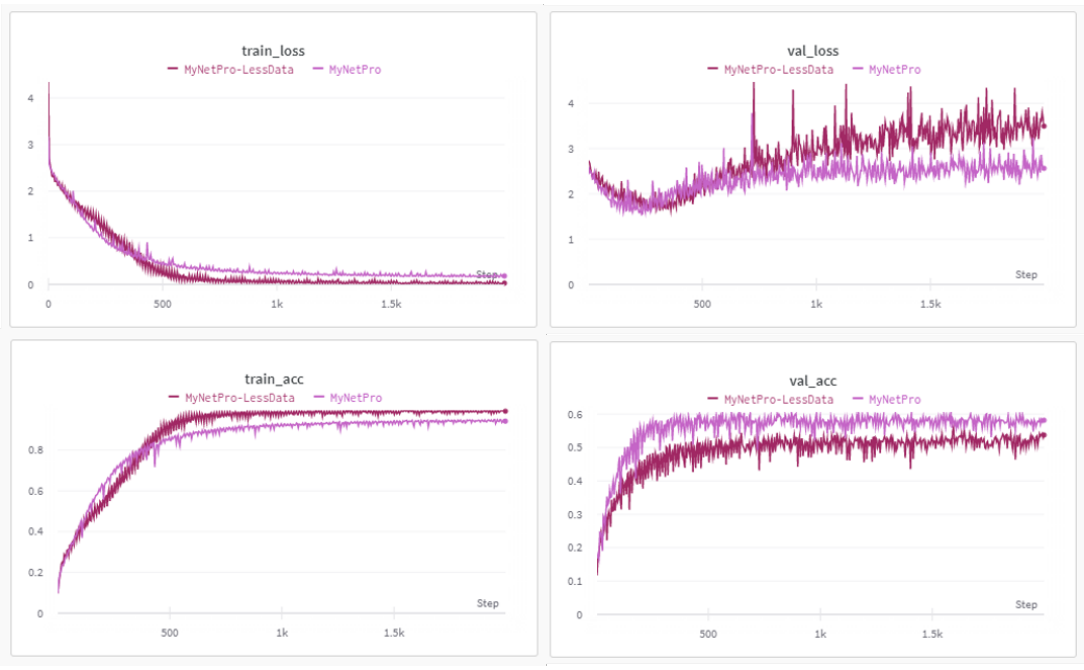

图 1: 有无数据增强对比

3 模型设计与测试

3.1 评价指标说明

在模型评价时，我采用了 acc,precision,recall,f1，混淆矩阵，各类别分类情况来综合评价模型的表现，由于均为常见评价指标，具体含义此处不再过多阐述。

评测指标

```
Precision = precision_score(test_label, test_pred, labels=[0, 1, 2], average='macro')
Recall = recall_score(test_label, test_pred, labels=[0, 1, 2], average='macro')
F1 = f1_score(test_label, test_pred, labels=[0, 1, 2], average='macro')
confusion = confusion_matrix(test_label, test_pred) # 混淆矩阵
report = classification_report(test_label, test_pred) # 每一类明细
```

对于各类别，对应关系为：

```
{'Corals':0, 'Crabs':1, 'Dolphin':2, 'Eel':3, 'Jelly Fish':4, 'Lobster':5, 'Nudibranchs':6,
    'Octopus':7,'Penguin':8, 'Puffers':9, 'Sea Rays':10, 'Sea Urchins':11, 'Seahorse':12,
    'Seal':13, 'Sharks':14,'Squid':15, 'Starfish':16, 'Turtle_Tortoise':17, 'Whale':18}
```

3.2 MyNet

3.2.1 结构

MyNet 是一个简单的两层神经网络，之前在训练 MNIST 时使用过该网络，可以达到 90% 的正确率。所以我本次又先用此简单的网络来进行训练。

MyNet 结构

```
class MyNet(nn.Module):
def __init__(self):
    super(MyNet, self).__init__()
    self.layers = nn.Sequential(
        nn.Conv2d(3,10, kernel_size=(5, 5)),
        nn.MaxPool2d(2),
        nn.Conv2d(10, 20, kernel_size=(5, 5)),
        nn.MaxPool2d(2),
        nn.ReLU(),
        nn.Flatten(),
        nn.Linear(56180, 500),
        nn.ReLU(),
        nn.Linear(500, 19),
    )
def forward(self, x):
    x = self.layers(x)
    return x
```

3.2.2 测试

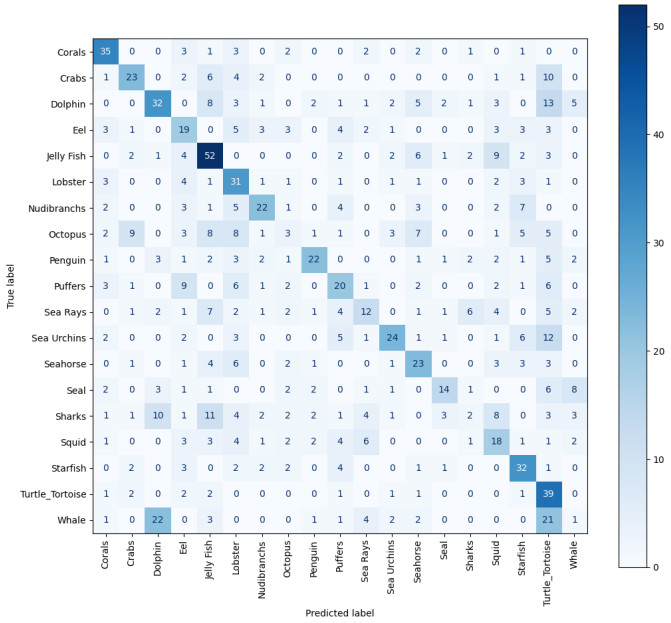

图 2: 混淆矩阵

	precision	recall	f1-score	support
0	0.60	0.70	0.65	50
1	0.53	0.46	0.49	50
2	0.44	0.41	0.42	79
3	0.31	0.38	0.34	50
4	0.47	0.60	0.53	86
5	0.35	0.62	0.45	50
6	0.56	0.44	0.49	50
7	0.12	0.05	0.07	57
8	0.65	0.45	0.53	49
9	0.38	0.37	0.37	54
10	0.35	0.23	0.28	52
11	0.62	0.41	0.49	58
12	0.41	0.48	0.44	48
13	0.58	0.33	0.42	42
14	0.12	0.03	0.05	59
15	0.31	0.37	0.33	49
16	0.48	0.64	0.55	50
17	0.28	0.78	0.42	50
18	0.04	0.02	0.02	58
accuracy			0.41	1041
macro avg	0.40	0.41	0.39	1041
weighted avg	0.40	0.41	0.39	1041

表 2: 各类分类明细

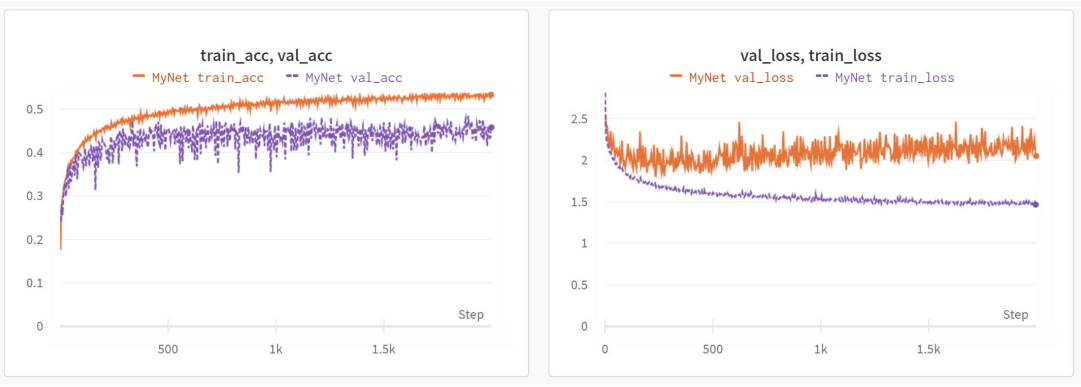

图 3: 训练曲线

model	acc	precision	recall	f1
MyNet	0.41	0.526	0.522	0.521

表 3: 测试集结果

从上述图表中可以看出，MyNet 由于结构过于简单，收敛效果并不好，即使在训练集上也没有很好地收敛，在验证集上更是早早出现了过拟合的情况，从混淆矩阵和分类明细上可以看出，MyNet 在相似动物（例如 Dolphin，Whale，Sharks）上表现并不好，导致对于 Whale 和 Sharks 的分类正确率很低。Octopus 由于数据集图像特征不鲜明所以不好学。

3.3 MyNetPro

3.3.1 结构

MyNetPro 是基于 AlexNet 改进卷积大小和增加 BN 层得到的，和 MyNet 相比，MyNetPro 具有更多的卷积层，增加了更多的 BN 层和 Dropout。在性能表现上 MyNetPro 表现出了更好的性能，在超参数测试和数据增强测试中我均采用 MyNetPro 进行试验。

MyNetPro 结构

```
class MyNetPro(nn.Module):
def __init__(self, num_classes=19):
    super(MyNetPro, self).__init__()
    self.features = nn.Sequential(
        nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
        nn.BatchNorm2d(64),
        nn.ReLU(inplace=True),
        nn.MaxPool2d(kernel_size=3, stride=2),
        nn.Conv2d(64, 192, kernel_size=5, padding=2),
        nn.BatchNorm2d(192),
        nn.ReLU(inplace=True),
        nn.MaxPool2d(kernel_size=3, stride=2),
        nn.Conv2d(192, 384, kernel_size=3, padding=1),
        nn.BatchNorm2d(384),
        nn.ReLU(inplace=True),
        nn.Conv2d(384, 256, kernel_size=3, padding=1),
```

```
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.classifier = nn.Sequential(
            nn.Dropout(0.3),
            nn.Linear(256 * 6 * 6, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(0.3),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes),
        )

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), 256 * 6 * 6)
        x = self.classifier(x)
        return x
```

3.3.2 测试

model	acc	precision	recall	f1
MyNetPro	0.54	0.586	0.757	0.658

表 4: 测试集结果

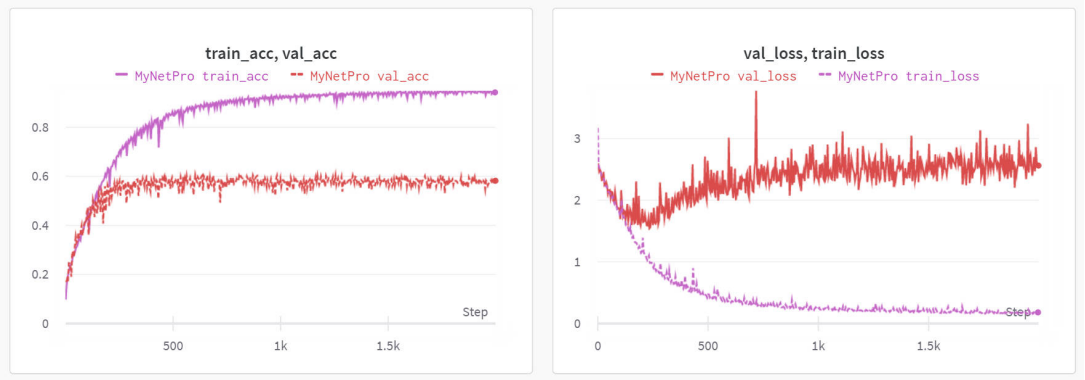

图 4: 训练曲线

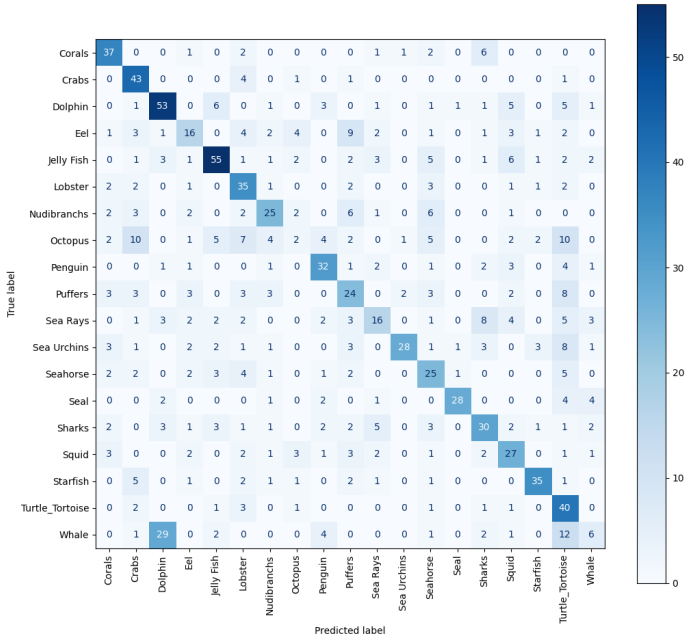

图 5: 混淆矩阵

	precision	recall	f1-score	support
0	0.65	0.74	0.69	50
1	0.55	0.86	0.67	50
2	0.56	0.67	0.61	79
3	0.44	0.32	0.37	50
4	0.70	0.64	0.67	86
5	0.48	0.70	0.57	50
6	0.57	0.50	0.53	50
7	0.12	0.04	0.05	57
8	0.63	0.65	0.64	49
9	0.39	0.44	0.41	54
10	0.46	0.31	0.37	52
11	0.88	0.48	0.62	58
12	0.41	0.52	0.46	48
13	0.90	0.67	0.77	42
14	0.53	0.51	0.52	59
15	0.47	0.55	0.50	49
16	0.80	0.70	0.74	50
17	0.36	0.80	0.50	50
18	0.29	0.10	0.15	58
accuracy			0.54	1041
macro avg	0.53	0.54	0.52	1041
weighted avg	0.54	0.54	0.52	1041

表 5: 各类分类明细

从上述图表中可以看出，相比于 MyNet，MyNetPro 在网络结构上复杂度提升明显，所以对于近似类

别的图片有了更强的分辨能力，收敛效果并不好，在训练集上收敛效果明显更好，但是仍然出现了过拟合问题，初步分析是由于图片数量过少导致的，即使已经增强了数据，但是 1w 张图片对于学习出一个高正确率的图像来说还是过于困难。Octopus 类仍然不好学，进一步证实了由于数据集图像特征不鲜明，有错误图像这一问题。

相比于 MyNet，MyNetPro 凭借更多的卷积层，BN 层，Dropout 获得了更好的表现。实现了任务要求的模型改进与提升。当然这个模型表现得仍然不够理想，因此我接下来又尝试了两种经典模型。

3.4 VGG

3.4.1 结构

VGG [1] 是 ILSVRC 2014 的相关工作，证明了增加网络的深度能够在一定程度上影响网络最终的性能。同时和之前的 SOTA 模型 AlexNet 相比，VGG 采用连续的几个 3x3 的卷积核代替 AlexNet 中的较大卷积核（11x11，7x7，5x5）。对于给定的感受野（与输出有关的输入图片的局部大小），采用堆积的小卷积核是优于采用大的卷积核，因为多层非线性层可以增加网络深度来保证学习更复杂的模式，而且代价还比较小（参数更少）。

本次实验使用了 VGG16，即图中 D。

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

图 6: VGG 模型

3.4.2 测试

model	**acc**	**precision**	**recall**	**f1**
VGG16	0.59	0.641	0.676	0.644

表 6: 测试集结果

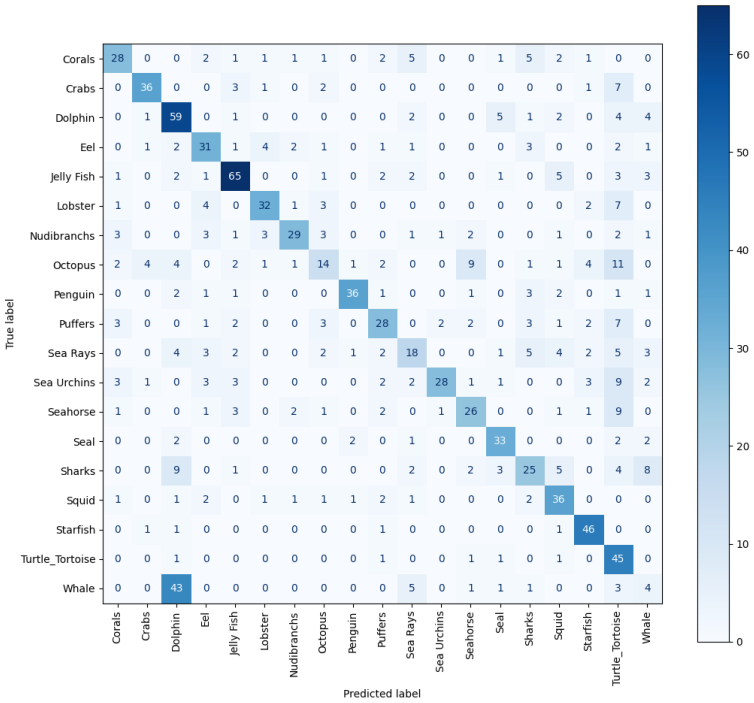

图 7: 混淆矩阵

	precision	recall	f1-score	support
0	0.65	0.56	0.60	50
1	0.82	0.72	0.77	50
2	0.45	0.75	0.56	79
3	0.60	0.62	0.61	50
4	0.76	0.76	0.76	86
5	0.74	0.64	0.69	50
6	0.78	0.58	0.67	50
7	0.44	0.25	0.31	57
8	0.88	0.73	0.80	49
9	0.61	0.52	0.56	54
10	0.45	0.35	0.39	52
11	0.88	0.48	0.62	58
12	0.58	0.54	0.56	48
13	0.70	0.79	0.74	42
14	0.51	0.42	0.46	59
15	0.58	0.73	0.65	49
16	0.74	0.92	0.82	50
17	0.37	0.90	0.53	50
18	0.14	0.07	0.09	58
accuracy			0.59	1041
macro avg	0.61	0.60	0.59	1041
weighted avg	0.61	0.59	0.59	1041

表 7: 各类分类明细

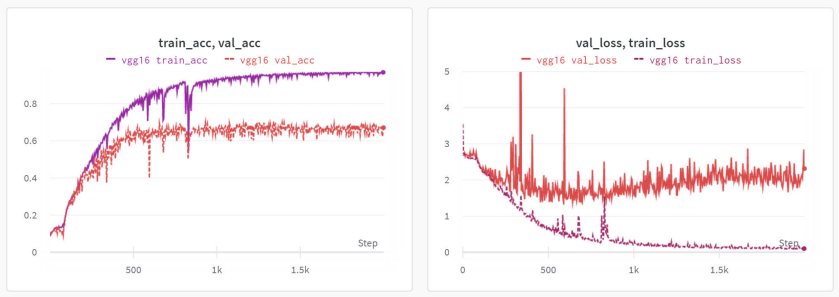

图 8: VGG16 曲线

相比于之前的网络，VGG 在各类上的表现更加的均匀，模型的泛化性能更好，唯一分类表现不好的 Whale 类则是又一次和 Dolphin 类混淆了。在 500 个 epoch 的训练后最终达到了 59% 的正确率。

3.5 ResNet

3.5.1 结构

ResNet [2] 是由 Kaiming He 于 2016 年提出的残差网络结构，其重要思想 “残差”（如图 Fig 9）缓解了深层网络的梯度消失和爆炸问题，同时由于学习的映射关系近似于恒等映射，从而使学习过程变得更加简单。这次实验中我也尝试使用了论文中提出的 ResNet34 网络结构来进行训练，网络结构如 Fig 11所示

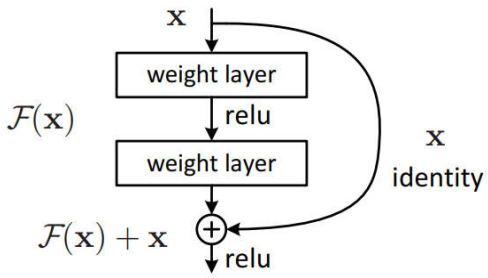

图 9: 残差块结构

model	acc	precision	recall	f1
ResNet34	0.58	0.587	0.798	0.671

表 8: 测试集结果

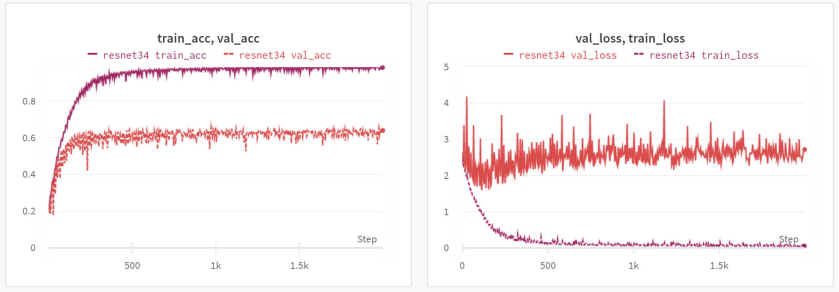

图 10: ResNet34 曲线

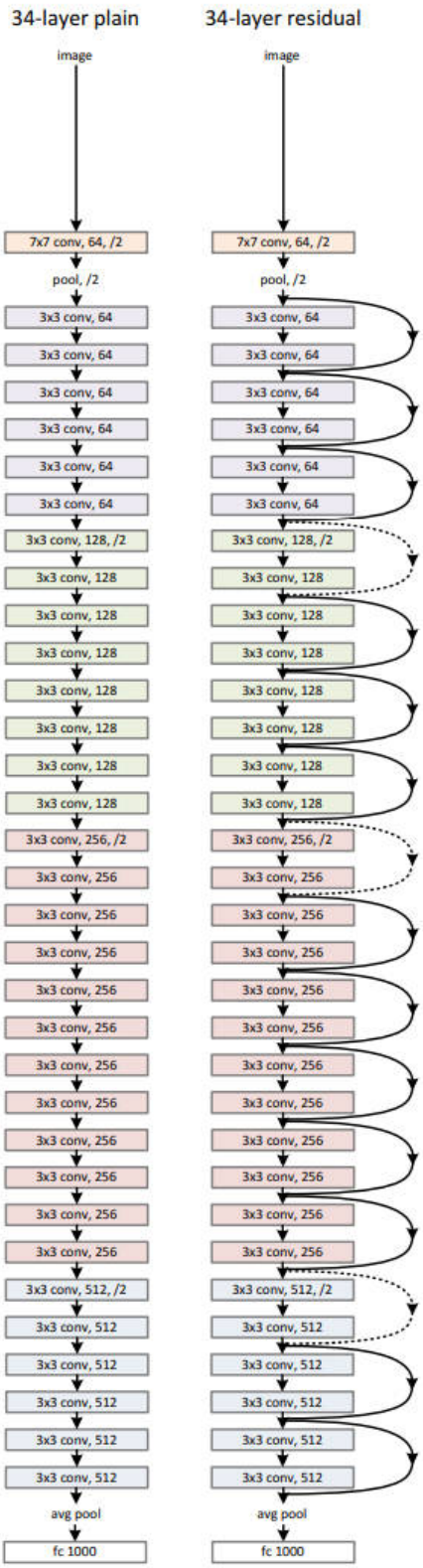

图 11: Res34 网络结构

3.5.2 测试

总体结果来看，resnet 的表现和 vgg 的表现大致相仿，正确率可以达到 58%，在 recall 和 f1 上 resnet 表现更好，在 precision 上则是 vgg 表现更好。

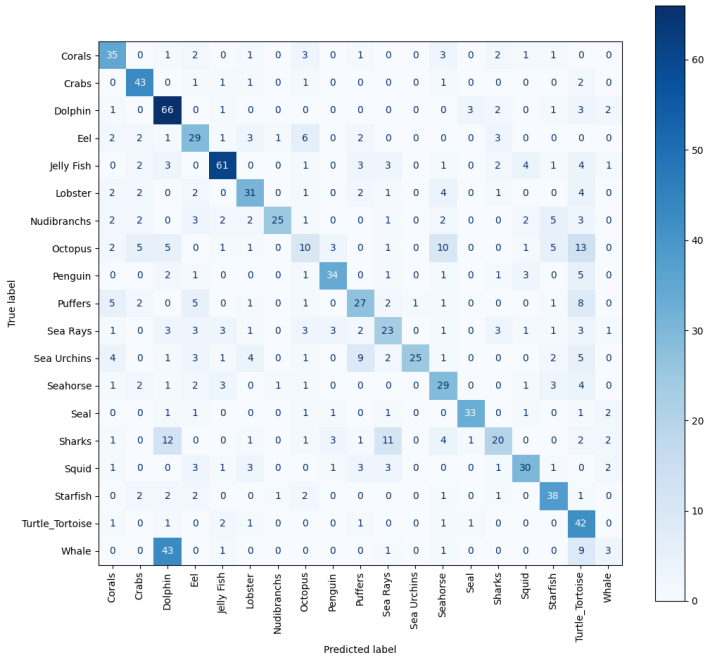

图 12: 混淆矩阵

	precision	recall	f1-score	support
0	0.60	0.70	0.65	50
1	0.69	0.86	0.77	50
2	0.46	0.84	0.60	79
3	0.51	0.58	0.54	50
4	0.78	0.71	0.74	86
5	0.62	0.62	0.62	50
6	0.89	0.50	0.64	50
7	0.29	0.18	0.22	57
8	0.76	0.69	0.72	49
9	0.53	0.50	0.51	54
10	0.46	0.44	0.45	52
11	0.96	0.43	0.60	58
12	0.48	0.60	0.53	48
13	0.87	0.79	0.82	42
14	0.56	0.34	0.42	59
15	0.68	0.61	0.65	49
16	0.64	0.76	0.70	50
17	0.39	0.84	0.53	50
18	0.23	0.05	0.08	58
accuracy			0.58	1041
macro avg	0.60	0.58	0.57	1041
weighted avg	0.60	0.58	0.57	1041

表 9: 各类分类明细

4 模型调试与改进

4.1 优化器和超参数选择

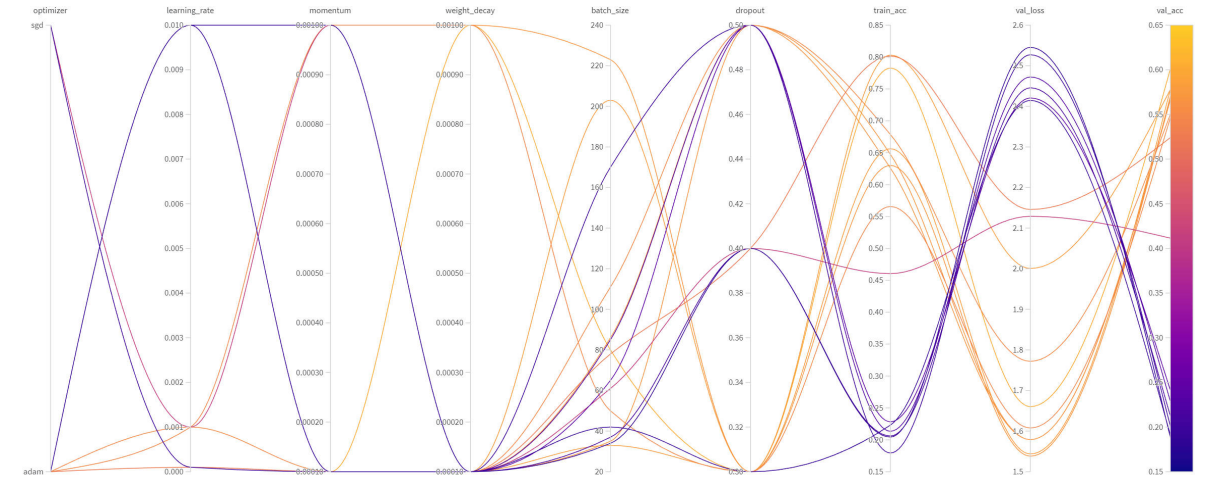

图 13: 超参数扫描曲线

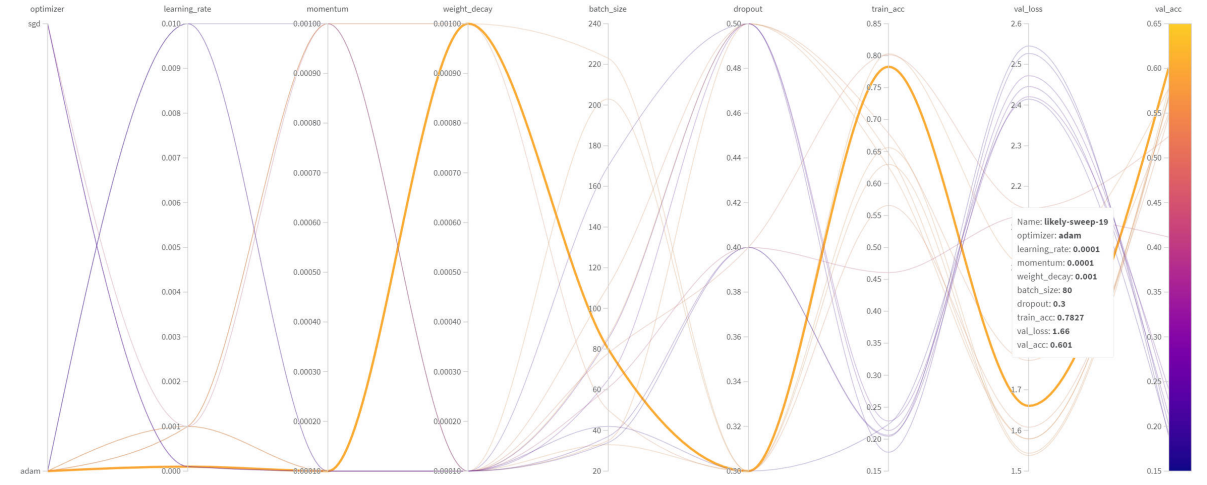

图 14: 扫描最优参数

这里我利用了wandb 的 sweep功能来扫描超参数。最终最优的超参数组合为：Adam, lr:0.0001,weight decay:0.001, batch size:80, dropout:0.3。由于计算时间限制，每组超参数我只设置了 50 个 epoch，扫描得到的最优超参数在验证集上的表现已经超过了我们前述的模型，达到了 60%，可以预见，合理选择超参数可以使模型水平大幅提升。

在优化器方面，由上图可以看出，Adam 优化器 [3] 普遍上表现的比 sgd 更好，这是因为 Adam 实际上就是将 Momentum 和 RMSprop 集合在一起，同时使用一阶动量和二阶动量，实现了：自动调整参数的学习率、大幅提升了训练速度、提高了稳定性。

在其他超参数方面，学习率上没有表现出出太强的规律，在 batch size 上普遍表现为较大的 batch size 表现比较好。其余参数则因条件不同而各有优劣。目前，超参数选择仍是深度学习领域正在研究的问题，本次作业为我提供了一个调试参数的机会，增加了很多调整参数的经验。

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1^{st} moment vector)
  $v_0 \leftarrow 0$ (Initialize 2^{nd} moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

图 15: Adam 算法伪代码

4.2 Pre-Train & Fine-Tune

近年来在深度学习领域，预训练模型一直是一个非常有效的工具，考虑到本次数据集规模较小，故尝试使用前人训练好的模型在此数据集上进行微调。最终取得了非常好的效果。

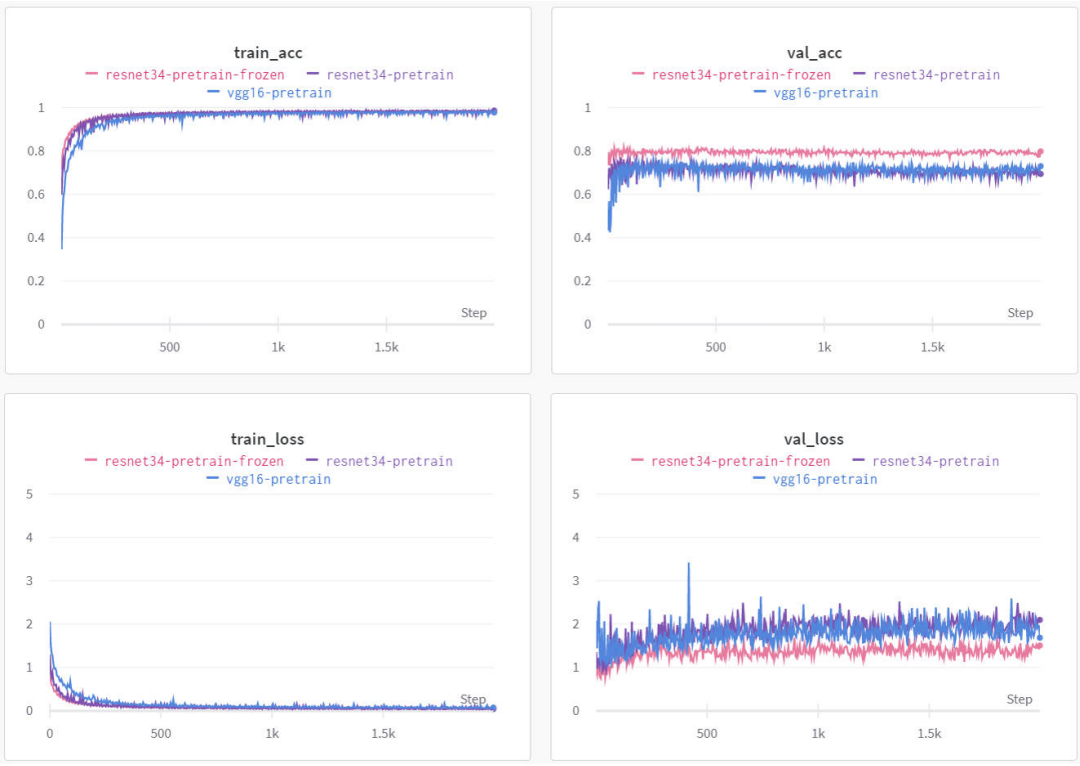

图 16: Pre-Train & Fine-Tune 曲线

4.2.1 Fine Tune VGG16

如下代码，我在原有 VGG16 模型最后添加了一层分类头（1000，19）。然后采用一个比较小的学习率在本次的海洋动物数据集上重新训练。取得的效果如下表

```
def vgg16_bn_pre():
  vgg16 = torchvision.models.vgg16_bn(pretrained=True)
  vgg16.classifier.add_module("head", nn.Linear(1000, 19))
  return vgg16
```

model	acc	precision	recall	f1
VGG16-PreTrain	0.68	0.663	0.868	0.748

表 10: VGG16-PreTrain 测试集结果

4.2.2 Fine Tune ResNet34

如下代码，我在原有 ResNet34 模型最后添加了一层分类头（1000，19）。然后采用一个比较小的学习率在本次的海洋动物数据集上重新训练。与之前不同的是，这次我还尝试了冻住一部分网络，只训练线性头和部分卷积层。

```
def resnet34pre():
  res34 = torchvision.models.resnet34(pretrained=True)
  numFit = res34.fc.in_features
  res34.fc = nn.Linear(numFit, 19)
  return res34
```

model	acc	precision	recall	f1
ResNet34-PreTrain	0.69	0.680	0.903	0.774

表 11: ResNet34-PreTrain 测试集结果

```
def resnet34pre_frozen():
  res34 = torchvision.models.resnet34(pretrained=True)
  numFit = res34.fc.in_features
  res34.fc = nn.Linear(numFit, 19)
  for param in res34.parameters():
      param.requires_grad = False
  for param in res34.fc.parameters():
      param.requires_grad = True
  for param in res34.layer4.parameters():
      param.requires_grad = True
  return res34
```

model	acc	precision	recall	f1
ResNet34-PreTrain-Frozen	0.76	0.760	0.941	0.831

表 12: ResNet34-PreTrain-Frozen 测试集结果

5 可视化

CAM [4]（类激活图）是一种可视化卷积网络关注点的方法。特征图的权重可以认为是被层层卷积核过滤后而保留的有效信息，其值越大，表明特征越有效，对网络预测结果越重要。一个深层的卷积神经网络，通过层层卷积操作，提取空间和语义信息。一般存在其他更难理解的层，例如分类的全连接层、softmax 层等，很难以利用可视化的方式展示出来。所以，CAM 的提取一般发生在卷积层，尤其是最后一层卷积。本质上是利用特征图权重叠加的原理来获得热力图。

Grad-CAM [5] 是在 CAM 基础上的梯度引入改进，相比 CAM，grad-CAM 可适用非 GAP 连接的网络结构且可以提取任意一层的热力图。

Grad-CAM++ [6] 认为梯度 map 上的每一个元素的贡献不同，因此增加了一个额外的权重对梯度 map 上的元素进行加权，总而优化了 Grad-CAM 的结果，使定位更精准。

下面我们实现了 Grad-CAM、Grad-CAM++（参考 这里），使用我们表现最好的 ResNet34-PreTrain-Frozen 模型为部分图片绘制热力图：

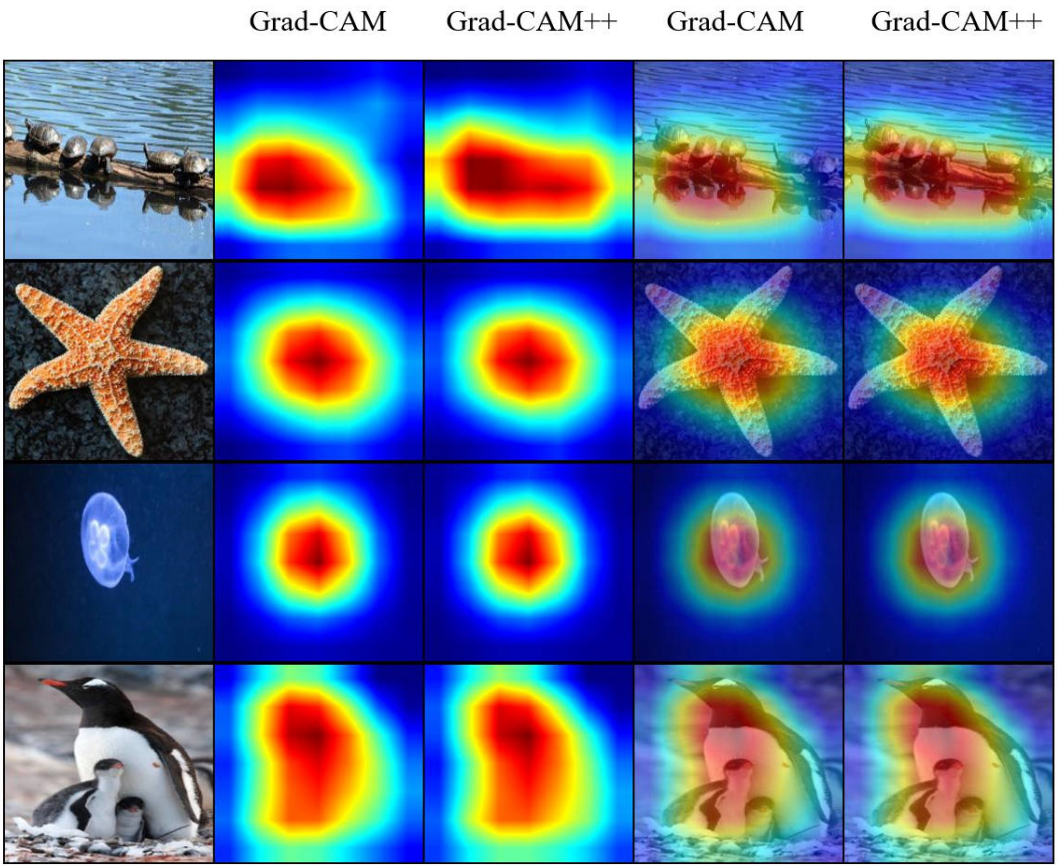

图 17: 模型决策过程可视化

如上图，我们可以观测到该模型的可解释性相当强，乌龟、企鹅、水母的决策过程更接近于通过外形、边缘来决策，而海星的决策和我想的不同，更多是依赖中心部位的分叉而不是整体造型。总体来说，该模型可靠性强、决策过程清晰。

6 总结

model	acc	precision	recall	f1
MyNet	0.41	0.526	0.522	0.521
MyNetPro	0.54	0.586	0.757	0.658
VGG16	0.59	0.641	0.676	0.644
VGG16-pretrain	0.68	0.663	0.868	0.748
ResNet34	0.58	0.587	0.798	0.671
ResNet34-pretrain	0.69	0.68	0.903	0.774
ResNet34-pretrain-frozen	0.76	0.76	0.941	0.831

表 13: 整体实验数据对照

本次大作业完成过程中遇到了很多阻力：例如开始并未发现海龟类的过采样，由于数据增强不足和超参数选择不当导致正确率迟迟无法提升。后来经过仔细的调研和反复尝试，通过 RandomCrop 数据增强，修改归一化中遇到的错误，对海龟进行欠采样等方式逐个解决问题，看起来简单的图片预处理其实到处暗藏杀机直接影响着实验成败。

同时也有很多新的收获和体会：尝试了微调预训练模型，实验了可视化卷积网络决策过程，使用 wandb 的 sweep 功能扫描超参数进行比较，学习并实验了各种各样的经典网络结构，收获十分丰硕。

最后，感谢老师和助教的悉心指导！PS：关于代码运行的方法见 README 文件。

参考文献

[1] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[2] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

[3] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[4] Bolei Zhou et al. “Learning deep features for discriminative localization”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2921–2929.

[5] Ramprasaath R. Selvaraju et al. “Grad-CAM: Why did you say that? Visual Explanations from Deep Networks via Gradient-based Localization”. In: *CoRR* abs/1610.02391 (2016). arXiv: 1610.02391. URL: http://arxiv.org/abs/1610.02391.

[6] Aditya Chattopadhyay et al. “Grad-CAM++: Generalized Gradient-based Visual Explanations for Deep Convolutional Networks”. In: *CoRR* abs/1710.11063 (2017). arXiv: 1710.11063. URL: http://arxiv.org/abs/1710.11063.