

# 基于 ResNet 无人机图像分类大作业

队长：田梓诺，2024E8014682037，人工智能 1606 班

成员 1：唐湘泉，2024E8014682036，人工智能 1606 班

成员 2：刘谨瑞，2024E8014682022，人工智能 1606 班

成员 3：周一浩，2024E8014682025，人工智能 1606 班

## 1. 队内分工情况

队长田梓诺负责代码主体架构搭建与报告大纲、整体逻辑框架的撰写；

成员 1 唐湘泉负责代码功能模块的具体实现与完善，确保代码准确性与高效性，并为代码添加详细注释，方便团队成员理解与后续维护；

成员 2 刘谨瑞负责代码运行测试、测试结果记录、问题发现与反馈和超参数调节，优化模型性能工作；

成员 3 周一浩负责收集实践过程中的素材资料，基于逻辑框架完成报告具体内容；同时对报告进行润色，优化语言表达，更具可读性；并基于群文件模板完成报告最终排版工作，确保格式规范统一。

## 2. 任务描述

### 2.1 任务背景

随着科技的不断发展，无人机技术已经在多个行业中找到了广泛的应用，尤其是在安防、物流和农业等领域。其优势在于能够以低成

本、高效率的方式完成许多传统方法无法轻松实现的任务，例如实时监控、运输物品、农业喷洒等。随着无人机数量的急剧增加，如何有效监控和识别这些无人机成为了一个亟待解决的问题。无人机图像监测不仅能够提高作业自动化程度，还能为无人机的安全管理和运行效率优化提供重要的数据支持。

另一方面，图像分类作为计算机视觉领域重要研究方向之一，其目的是通过图像或视频输入实现对目标物体的自动识别与分类。在无人机监测中，计算机视觉技术能够帮助识别图像中的无人机以及背景元素（如树木、天空、建筑物等），从而自动化地进行无人机的检测、追踪和识别。结合深度学习算法，尤其是卷积神经网络（CNN），可以大幅提高图像识别的精度和效率。随着深度学习在计算机视觉中的广泛应用，图像分类和目标检测等任务已经取得了显著的进展。

## 2.2 任务目的

本项目旨在模拟实际场景中无人机监测需求，通过构建一个基于计算机视觉的自动化无人机图像识别系统，深入了解计算机视觉和深度学习的技术实现过程及其在实际应用中的重要性，同时掌握在实际应用场景中如何有效解决无人机监测等实际问题，任务具体目标包括：

1) **无人机与背景图像分类：**通过处理无人机图像以及背景图像（如树木、天空、建筑物等），利用计算机视觉技术对图像进行分类和识别，准确区分无人机和背景信息。

2) **计算机视觉与深度学习算法应用实践：**通过项目实践掌握卷积神经网络（CNN）等深度学习算法的基本原理及其在图像分类中的

应用方法，理解其在物体检测和图像识别中的重要作用。

3) **提高图像识别精度与效率**: 通过对多种深度学习算法的实践，探索如何优化算法，提升图像识别的准确性，并实现高效的图像分类与检测。

### 2.3 任务要求

1. 正确整理给定数据集，合理进行图像预处理，为后续模型训练和测试提供高效的、充分的样本；

2. 选择合适的深度学习模型（如卷积神经网络）进行图像分类任务，利用第 1 步中的数据集进行模型训练和验证；

3. 在初步完成图像分类实现的基础上，通过优化模型结构或调整超参数来提升分类精度和运行效率；

4. 通过对模型的测试集进行评估，考察模型的分类准确率、召回率、F1 分数等指标，确保模型具有较好的实际应用效果；

5. 最后完成项目报告，详细描述项目的背景、需求、方法与实施过程，并对实验结果进行分析与总结。

## 3. ResNet 残差网络

本项目基于 ResNet 残差网络实现，下面对残差网络模型进行简要介绍。时间回到 2012 年，当时卷积神经网络（CNN）的研究主要分为两个大方向，一类是以 VGG[1]等为代表的增加网络深度和宽度的研究思路；而另一类则是以 GoogLeNet[2]等为代表的增加卷积核拟合能力的研究思路。VGG 和 GoogLeNet 网络分别达到了史无前例的 19

层和 22 层。然而，随着网络层数的不断增加，计算开销急剧增大、过拟合等问题也日益凸显。虽然过拟合问题可以通过数据增强、正则化等技术手段进行缓解，但是单纯增加算力并不能直接解决此问题，反而可能因为模型复杂度过高而加剧过拟合的风险。此外，研究者们还发现了一个“网络退化”现象，即随着网络深度进一步加深，训练集的损失函数（loss）逐步下降并趋于饱和，此时若继续增加网络深度，反而会导致 loss 不降反升，这一现象无法通过常规的优化手段解决。

面对网络退化问题，一个直观的解决思路是：当网络退化现象出现时，较浅层网络的性能至少不应该差于更深层的网络。于是我们或许可以尝试将浅层网络的特征直接传递到深层网络，这样即便深层网络不能学习到更好的特征表示，至少也能保证其性能不差于浅层网络。基于这种思想，ResNet 残差网络应运而生[3]，它通过引入“直接映射”（Identity Mapping）机制，将不同网络层之间的特征直接相连，从而确保了深层网络至少能够包含浅层网络所学习到的所有特征信息。下面对 ResNet 残差网络做进一步详细介绍。

### 3.1 残差块

众多残差块构成了 ResNet 残差网络，其中一个残差块可以用公式表示为：

$$x_{l+1} = x_l + F(x_l, W_l) \quad 3.1$$

其核心如图 3.1 所示：

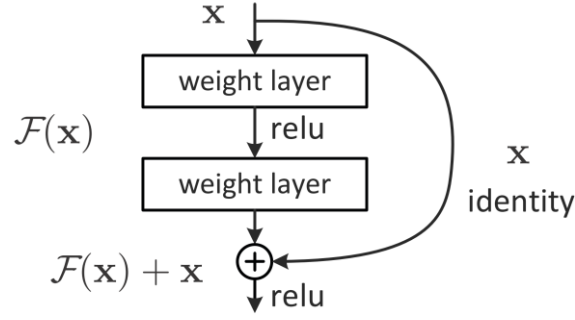


图 3.1 残差块[3]

例如，在该结构中输入  $x$  为为  $3 \times 3$  且填充值均为 1 的特征图，假设经过两层卷积后获得的特征图为  $3 \times 3$  且填充值均为 2，此时经过 identity 求和操作后即获得了一个大小为  $3 \times 3$  且特征值均为 3 的特征图。若卷积后特征图大小不一致，则可以通过用 0 填充方式使大小一致，若通道数不同则可以通过  $1 \times 1$  卷积核进行升维或降维使其一致后再进行求和运算。根据以上基本的残差块进行堆叠即可构成完整的残差网络。

### 3.2 基本原理

下面对残差网络解决深层网络中梯度消失问题的原理做简要分析，为便于表示，这里以将  $F(x) + x$  记作  $y$ ，所以求  $y$  对  $w$  的偏导可得：

$$\frac{\partial y}{\partial w} = \frac{\partial y}{\partial F(x)} \frac{\partial F(x)}{\partial x} \frac{\partial x}{\partial w} + \frac{\partial x}{\partial x} \frac{\partial x}{\partial w} \quad 3.2$$

相比较传统网络，其偏导数为：

$$\frac{\partial F(x)}{\partial w} = \frac{\partial F(x)}{\partial x} \frac{\partial x}{\partial w} \quad 3.3$$

比较式 3.2 和 3.3 可知，残差网络在反向传播过程中多了相加项  $\frac{\partial x}{\partial x} \frac{\partial x}{\partial w}$ ，相比于传统网络只有连乘项，大幅度降低了梯度消失问题出现

的可能性。

再根据图 3.1 分析可知， $F(x)$ 其实即可看作 $x$ 和真实值之间的残差，其存在意义便是在已有 $x$ 的基础上向靠近真实值的方向进行微调，使求得结果更加接近真实值。设想在训练过程中将 $F(x)$ 置零，即不对 $x$ 进行任何操作，此时加入 $F(x)$ 不论其做出贡献大小，至少不会比没有它表现出更差的结果，即对应了随着网络层次的增加，不应当让网络性能变得更差[3]这个结论。

从以上分析可以看出，残差网络并没有增加需要学习或训练的额外参数，且在整个过程中只是添加了一些便于计算机计算的加法运算，具备很好的可行性。

从模型集成角度来看，Veit 等人[4]的论文指出，将一个 3 层的残差网络展开可得如图 3.2 所示的结构：

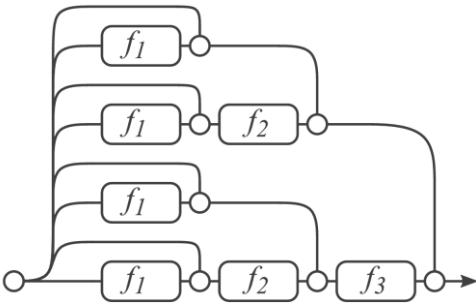


图 3.2 3 层残差网络的展开结构

由图 3.2 可知不同于传统模型其路径长度各不相同，而实际训练中大部分梯度是来自 10-34 层的路径，即通过引入短路径来避免了梯度消失问题，同时这些浅层的梯度可以被携带到非常深的网络范围内。

自残差网络模型出现后，深度学习的网络深度出现了前所未有的突破，百层的网络层出不穷，大大促进了深度学习技术的发展。在如

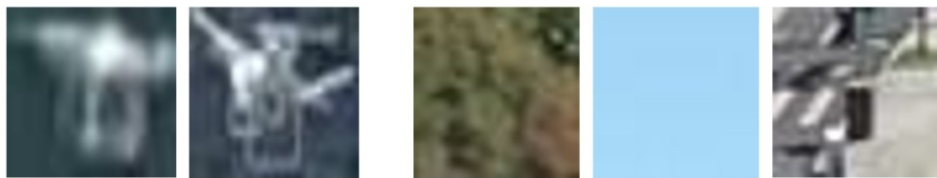
今的模型架构中，随处可见残差连接的身影，可以说是构建深度神经网络的必要手段之一。

## 4. 核心代码与实验设置

本次实践基于 Python 3.9 版本编写开发，所需的第三方库包括 csv, PyTorch, timm 等，并对 mae 项目中的 util 进行了复用，提高了开发效率。

### 4.1 数据读取与预处理

本次实验基于作业所给无人机与背景图像数据集开展，其中无人机图片和背景图片的尺寸均为 32\*32，且训练集按照无人机图片、背景图片 1: 10 的比例构成；验证集、测试集这一比例均为 1: 1。



(a) 无人机图片示例      (b) 背景图片示例（树木、天空、建筑物等）

图 4.1 数据集部分展示

作为一个典型的监督学习问题，我们可以首先明确实验目的：输入图像，经过分析后输出模型预测的生物种类标签，build\_transform 函数如图 4.2 所示：

```
def build_transform(is_train, args): #构建数据预处理的 transform 流程，支持训练和验证阶段的不同数据增强策略 1个用法
    # 训练
    if is_train: #检查 is_train 是否为 True
        print("train transform")
        return torchvision.transforms.Compose([ #顺序执行以下变换
            torchvision.transforms.Resize((args.input_size, args.input_size)), #调整输入图像大小到网络模型所接受的输入
            torchvision.transforms.RandomPerspective(distortion_scale=0.1, p=0.5), #随机透视变换
            torchvision.transforms.ToTensor() #转换为 PyTorch 张量格式并归一化
        ])
    # 验证
    print("eval transform")
    return torchvision.transforms.Compose([ #验证阶段的数据预处理
        torchvision.transforms.Resize((args.input_size, args.input_size)),
        torchvision.transforms.ToTensor()
    ])

```

图 4.2 build\_transform 函数

在该部分中，针对训练和验证阶段设计了不同的数据增强策略，在训练阶段首先调整输入图像大小到网络模型所接受的输入，随后加入了随机透视变换，而在验证阶段则没有实行变换，随后采用如图 4.3 所示代码加载数据集，采用该指令的好处在于可以自动识别类别名并相应分配索引。

```
dataset = torchvision.datasets.ImageFolder(path, transform=transform) #可以自动识别类别名并相应分配索引
info = dataset.find_classes(path) #返回一个包含info0 info1两个元素的元组

```

图 4.3 加载数据集

## 4.2 模型训练与评价

在训练模式下，首先调用 4.1 小节中的函数加载数据，其中训练集数据做打散处理，提高训练效果，该部分代码如图 4.4 所示。



```

#构建数据集
dataset_train = build_dataset(is_train=True, args=args, mode='train')
dataset_val = build_dataset(is_train=False, args=args, mode='val')
# 构建数据加载器，训练集打散
sampler_train = torch.utils.data.RandomSampler(dataset_train)
sampler_val = torch.utils.data.SequentialSampler(dataset_val)
data_loader_train = torch.utils.data.DataLoader(
    dataset_train, sampler=sampler_train,
    batch_size=args.batch_size,
    num_workers=args.num_workers,
    pin_memory=args.pin_mem,
    drop_last=False
)
data_loader_val = torch.utils.data.DataLoader(
    dataset_val, sampler=sampler_val,
    batch_size=args.batch_size,
    num_workers=args.num_workers,
    pin_memory=args.pin_mem,
    drop_last=False
)

```

图 4.4 构建数据集与加载器

数据读入完成后，进行 epoch 循环训练，单个 epoch 流程代码如图 4.5 所示，由循环遍历数据加载器中的数据，同时在每个 epoch 训练完成后（epoch0 除外），都会基于 val 验证集验证当前训练效果并打印准确率等信息。

```

for data_iter_step, (samples, targets) in enumerate(data_loader):
    # 将数据移到设备上
    samples = samples.to(device, non_blocking=True)
    targets = targets.to(device, non_blocking=True)

    outputs = model(samples) #计算前向传播，得到输出

    warmup_lr = args.lr #预热学习率，在训练的初始阶段逐步提高学习率，减小模型开始训练时的震荡
    optimizer.param_groups[0]["lr"] = warmup_lr #更新优化器中第一个参数组的学习率为预热学习率
    #计算损失并进行梯度累积
    loss = criterion(outputs, targets) #计算模型输出与目标标签之间的损失
    loss /= accum_iter
    # 混合精度训练与梯度更新
    loss_scaler(loss, optimizer, clip_grad=max_norm, #防止梯度过大
                parameters=model.parameters(), create_graph=False,
                update_grad=(data_iter_step + 1) % accum_iter == 0) #达到梯度累积的次数时才更新梯度
    loss_value = loss.item() #获取当前损失值
    # 梯度清零，以进行下一次梯度计算
    if (data_iter_step + 1) % accum_iter == 0:
        optimizer.zero_grad()

```

图 4.5 训练单个 epoch

在所有 epoch 训练完成后，程序最终会打印在所有 epoch 中最高

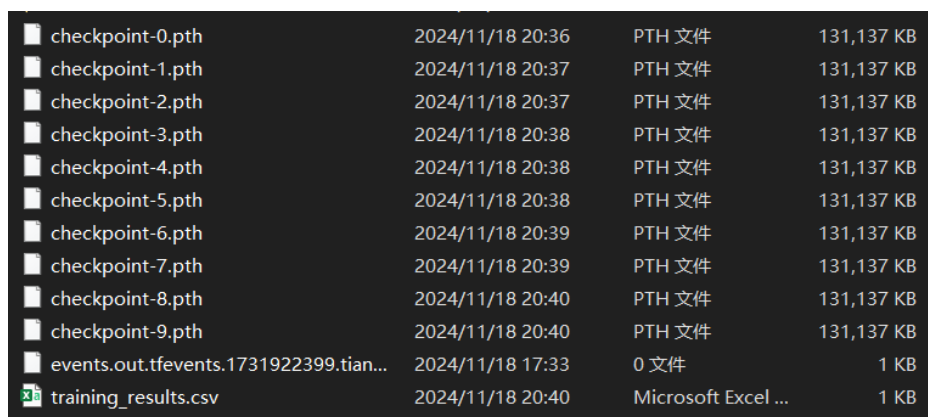
的准确率及其所对应的 epoch 数，方便后续使用该模型对测试集进行预测。

### 4.3 模型预测

参考 README.md 文件中使用方式对代码修改，使其运行在 infer 推理模式，程序会自动打印输出当前模型对 test 测试集内图片的预测准确率。若需获取每张图片的预测结果，可取消 main 函数中相关代码段的注释，程序将逐一打印每张图片的预测标签，便于针对性地分析模型预测错误的样本，进而优化模型性能。

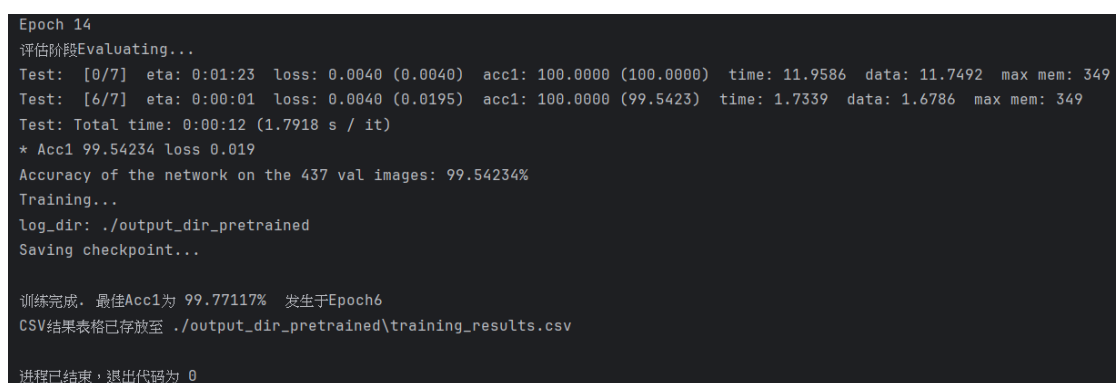
## 5. 结果分析与总结

首先，按照 README.md 文件中的说明，运行 statistic\_mean\_std.py 脚本计算训练数据集的均值和标准差，用于后续数据预处理中的归一化步骤。然后，运行 train.py 脚本进行模型训练，等待设定 epoch 完成后可在 output\_dir\_pretrained 文件夹内得到如图 5.1 所示结果，模型训练过程的日志输出如图 5.2 所示。其中 checkpoint-x.pth 为各 epoch 对应的模型权重文件，用于后续 infer 模型下加载模型并进行测试集预测结果的输出，training\_results.csv 则记录了模型在训练过程中每个 epoch 的验证集准确率。



checkpoint-0.pth	2024/11/18 20:36	PTH 文件	131,137 KB
checkpoint-1.pth	2024/11/18 20:37	PTH 文件	131,137 KB
checkpoint-2.pth	2024/11/18 20:37	PTH 文件	131,137 KB
checkpoint-3.pth	2024/11/18 20:38	PTH 文件	131,137 KB
checkpoint-4.pth	2024/11/18 20:38	PTH 文件	131,137 KB
checkpoint-5.pth	2024/11/18 20:38	PTH 文件	131,137 KB
checkpoint-6.pth	2024/11/18 20:39	PTH 文件	131,137 KB
checkpoint-7.pth	2024/11/18 20:39	PTH 文件	131,137 KB
checkpoint-8.pth	2024/11/18 20:40	PTH 文件	131,137 KB
checkpoint-9.pth	2024/11/18 20:40	PTH 文件	131,137 KB
events.out.tfevents.1731922399.tian...	2024/11/18 17:33	0 文件	1 KB
training_results.csv	2024/11/18 20:40	Microsoft Excel ...	1 KB

图 5.1 运行结果文件夹



```
Epoch 14
评估阶段Evaluating...
Test: [0/7] eta: 0:01:23 loss: 0.0040 (0.0040) acc1: 100.0000 (100.0000) time: 11.9586 data: 11.7492 max mem: 349
Test: [6/7] eta: 0:00:01 loss: 0.0040 (0.0195) acc1: 100.0000 (99.5423) time: 1.7339 data: 1.6786 max mem: 349
Test: Total time: 0:00:12 (1.7918 s / it)
* Acc1 99.54234 loss 0.019
Accuracy of the network on the 437 val images: 99.54234%
Training...
log_dir: ./output_dir_pretrained
Saving checkpoint...

训练完成，最佳Acc1为 99.77117% 发生于Epoch6
CSV结果表格已存放至 ./output_dir_pretrained\training_results.csv

进程已结束，退出代码为 0
```

图 5.2 模型训练过程日志输出

本实践对比了有无预训练的准确率，根据生成的 csv 文件可得到如图 5.3 所示结果，其中红色曲线为使用预训练前提下得到的准确率随 epoch 变化情况，而蓝色线为不使用预训练前提下得到的准确率随 epoch 变化情况，实线和虚线分别代表 acc@1 和 loss 值变化。

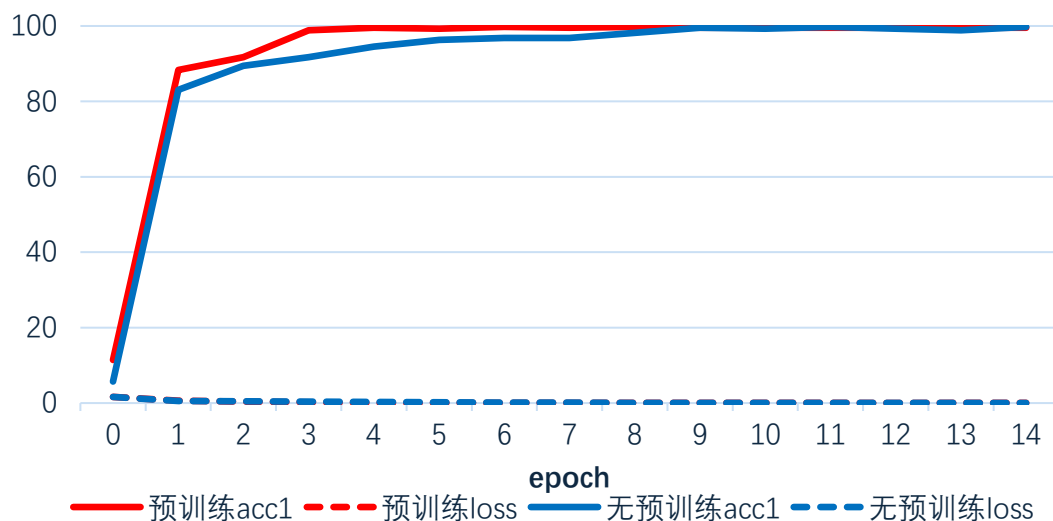


图 5.3 准确率曲线图

由图 5.3 可知在本实验条件的前提下，使用预训练的训练过程起始准确率更高，且在第 3 个 epoch 时便接近 100% 准确率，而不使用预训练的训练过程起始准确率只有 5.7%，低于有预训练的 11.4% 起始准确率，同时花费了 9 个 epoch 才达到了有预训练时 3 个 epoch 的验证集预测准确率。考虑到本实验所采用数据集较为简单，若是对于更加大型、更为复杂的任务，有无预训练在训练时间成本上的差异将是巨大的。

分别用有无预训练在 15 个 epoch 中所得到的最佳验证集准确率模型对 test 测试集图片进行测试，可得到如图 5.4 所示结果。

```
Evaluating on the test set...
Test: [0/1] eta: 0:00:03 loss: 0.0737 (0.0737) acc1: 94.8718 (94.8718) time: 3.9497 data: 2.6965 max mem: 69
Test: Total time: 0:00:04 (4.3417 s / it)
* Acc1 94.87180 loss 0.074
The .pth's acc@1 on the test image is 94.87180%
进程已结束，退出代码为 0
```

预训练

```
Evaluating on the test set...
Test: [0/1] eta: 0:00:03 loss: 0.1795 (0.1795) acc1: 92.3077 (92.3077) time: 3.9562 data: 2.6981 max mem: 69
Test: Total time: 0:00:04 (4.3488 s / it)
* Acc1 92.30769 loss 0.179
The .pth's acc@1 on the test image is 92.30769%

进程已结束，退出代码为 0
```

无预训练

图 5.4 test 测试集预测准确率

由图 5.4 可知，使用预训练的最佳验证集准确率模型在测试集上的预测准确率达到 94.87%，而不使用预训练的模型准确率为 92.31%。尽管从数值上看两者差距不大，但考虑到本实验所用测试集数据量较小，且验证集上的最高准确率相近，尚不能断言二者的预测能力在更大数据集或实际应用场景下也无显著差异。特别是，在数据量较小的情况下，有预训练模型的准确率仍然略高一筹，这表明预训练对模型性能的提升仍然具有积极意义。对于更大型、更复杂的任务，有无预训练的模型在性能、训练成本和泛化能力上的差距可能会更加显著。

## 参考文献

- [1] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv:1409.1556, 2014.
- [2] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In CVPR, 2015.
- [3] He K, Zhang X, Ren S, et al. Deep Residual Learning for Image Recognition[J]. IEEE, 2016.
- [4] Veit A, Wilber M, Belongie S. Residual Networks Behave Like Ensembles of Relatively Shallow Networks[J]. Advances in Neural Information Processing Systems, 2016.