固定类中心基于距离的开集识别

刘紫琪

摘要

近几年深度神经网络的应用，使得计算机视觉达到了前所未有的高度，在图像分类等任务的性能上有了很大的提升。这些分类任务都是基于封闭世界假设，也就是以训练集包含所有已知的类别为前提进行训练。但是在许多实际应用中，比如自动驾驶系统，真实环境会出现许多训练时从未见过的类别，由于封闭世界假设，训练好的分类器总是会对某一个已知类别产生较高的置信度，这一结果给开集环境下的任务带来非常严重的安全隐患，所以提出开集识别任务。本文所复现的论文针对开集识别任务通过固定类中心，提出一个新的基于距离的损失函数，并且在开集识别任务中达到不错的效果。在本次复现中，通过替换原骨干网络为 ImageNet 预训练模型 EfficientNetb5，在数据集 tinyImageNet 和 CIFAR10 上大大缩短训练时间，减少计算量的情况下，达到与论文一样的精度。使用温度系数缩放并且在输入时添加小扰动后，在数据集 CIFAR10 上高于原论文 6.1%，tinyImageNet 上高于原论文 1%。

关键词：开集识别；图片分类；深度学习；聚类

1 引言

当前许多识别系统都是基于封闭世界假设，也就是假设训练集已经涵盖了所有已知的类别。然而，像自动驾驶系统等许多实际应用中，应用环境将出现许许多多训练时从未出现过的类别，基于封闭世界所训练的模型往往错误地将位置类别识别为某一已知类别，这将造成难以想象的事故，所以开集识别任务的提出至关重要。

开集识别任务需要将已知类正确分类，并且拒绝出现的位置类别。本文目标复现的论文[1]针对开集识别任务，对之前像 OpenMax[2]、CROSR[3]等基于距离的开集识别任务进行分析，发现之前基于距离的模型在 logit 空间上会出现已知类别类间不紧凑和类边界不清晰的问题，基于此提出了新的开集识别算法。

该论文提出一个新的损失函数，通过固定类中心和聚类的方式，在开集识别任务中达到不错的性能效果。该论文对于开集识别提出了新的损失函数，使得已知类的训练数据能够在类中心上聚集的非常紧密，并且在保证闭集分类精度的情况下达到了不错的开集识别效果。该论文介绍了固定类中心这一策略，并说明这一策略在基于距离的学习上是非常有效且灵活的，与在训练中学习到类中心的方法相比，该论文使用固定的类中心，在类内多样性较大且数量较多的情况下的数据集表现效果较好。

在本次复现中，替换原论文的深度神经网络为 ImageNet 预训练模型 EfficientNetb5，使用 CAC 进行 fine-tune，可以缩短训练时间，并且在计算量减少的情况下达到同样的性能表现。并且使用温度系数缩放在输入时添加小扰动，实验结果发现 CAC 和 ODIN 方法的结合可以有效提升开集识别的性能。在数据集 tineImageNet 上达到和论文中一样的精度，结合 ODIN 后超过原论文 1.7%，在数据集 CIFAR10 上比原论文高 3%，结合 ODIN 后超过原论文 6.1%，平均增加了 4.11%。

2 相关工作

此章节对基于距离的开集识别任务的定义以及前人工作做进一步的介绍。

2.1 基于距离的开集识别

开集识别任务是针对测试环境中出现训练环境没有出现过的未知类的任务，需要在保持闭集分类精度的同时，拒绝未知类。基于距离的开集识别是通过使用度量学习领域的距离损失函数学习特征并应用于开集识别领域中。

Bendale 等[2]在 2016 年提出 OpenMax，这是第一个将卷积神经网络应用在开集分类器上的算法，该算法使用神经网络倒数第二层的 logit 空间作为激活向量，对每一个类通过正确分类的训练样本计算类中心，根据类中心和到类中心的距离拟合 Weibull 分布，校准 softmax 值后扩展 K 类分类器为 $K+1$ 类分类器，进而达到开集识别的任务。OpenMax 也是第一个基于距离的开集识别方法，使用正确分类的训练数据的距离最小化开集风险。

Youshihashi 等[3]在 2018 年提出 CROSR，该论文通过分析 OpenMax，发现尽管激活向量在监督网络中可以用来优化对输入图片的分类概率预测，但是激活向量并不支持对输入图片编码，也不能充分测试图片是否有可能属于某个类别。所以该论文提出 DHRNet，用重构损失学习图片的潜在特征，之后再沿用 OpenMax 的思路，对每个类别计算类中心和到类中心的距离拟合 Weibull 分布，校准 softmax 值后输出 $K+1$ 维概率。

Miller 等[1]在 2021 年提出 CAC，也就是本文想要复现的算法，该论文对于之前的 OpenMax、CROSR 等基于距离的开集分类任务，在 logit 空间上有已知类边界不清晰的问题，基于此提出了新的损失函数，利用聚类的思路，使得已知类的数据在 logit 空间上更加紧密地围绕固定类中心聚集，并且最大程度拉开到其他类中心之间的距离，使得该算法在保证闭集分类精度的同时，有很好的未知类检测能力。

3 本文方法

3.1 本文方法概述

CAC 模型希望通过 $f(x)$ 使得输入样本在 logit 空间各类别之间聚集地非常紧凑，这样可以使用距离类中心的度量指标来正确分类已知类的同时，拒绝未知类的输入，其所训练的开集分类器框架主要包括以下三个部分：

1. 深度神经网络 f，将输入图像 x 映射到 logit 空间 $z=f(x)$。这个深度神经网络 f 可以是任何一个 N 维分类器，N 为已知类别的数量。

2. 一个不需要训练的参数 C，是类中心的集合，展开形式为 $(c_1,\ldots,c_N)$

3. 替换 SoftMax 层为 $e(z,C)$，是计算 logit 向量 z 到各个类中心的欧式距离向量。

总结来说，CAC 基于距离的开集分类器的输出如下式所示：

$$d=e(z,C)=(||z-c_1||_2,\ldots,||z-c_N||_2)^T \tag{1}$$

其中，$||\cdot||_2$ 为欧几里得范数。

3.2 固定类中心

固定类中心，就是在训练前固定好各个类别在 logit 空间的聚类类中心。对于每一个已知类别 i，都将有一个在 logit 空间固定的类中心 c_i。对于 N 个已知类的 N 维 logit 空间，将每个已知类的固定类中心放置在各个坐标轴上的某一点。因此，固定类中心的点就等价于一个缩放的基准向量 e_i，或者是一个缩放的独热编码向量。引入一个超参数 α，固定类中心的公式如下：

$$C = (c_1, \ldots, c_N) = (\alpha \cdot e_1, \ldots, \alpha \cdot e_N) \tag{2}$$

$$e_1 = (1, 0, \ldots, 0)^T, e_N = (0, \ldots, 0, 1)^T \tag{3}$$

在训练结束后，固定类中心 C 将是训练数据正确分类的中心点，这使得模型的类中心可以更精确地适应更加复杂的数据集。

3.3 固定类中心的聚类损失函数定义

CAC 模型的目标是设计一个损失函数使得训练数据到正确类别类中心的距离尽可能的近，到其他类别类中心的距离尽可能的远。设计的损失函数如下式所示：

$$L_{CAC}(x, y) = L_T(x, y) + \lambda L_A(x, y) \tag{4}$$

其中，λ 作为平衡两个单独损失项 L_T 和 L_A 引入的超参。

L_{CAC} 的 L_T 项实际上是修正的 Tuplet 损失，可以强制最大化输入样本 x 到正确类别类中心和所有其他类别类中心的距离。L_T 如下式所示：

$$L_T(x, y) = \log(1 + \sum_{j \neq y}^{N} e^{d_y - d_j}) \tag{5}$$

L_{CAC} 的 L_A 项实际是为了确保输入样本到正确类别类中心的距离最小。L_A 如下式所示：

$$L_A(x, y) = d_y = ||f(x) - c_y||_2 \tag{6}$$

通过结合固定类中心和聚类的损失函数，该损失函数最大限度地减少了训练样本到其正确类别类中心地距离，同时最大限度地增加了到其他类别类中心的距离。进而使得各个类别聚集的紧凑有比较明显的边界，同时为未知类提供更多的空间。

3.4 开集识别任务测试阶段

在测试阶段的时候，模型需要正确分类已知类别的输入，同时拒绝未知类别的输入。我们通过计算拒绝值 $\gamma = (\gamma_1, \ldots, \gamma_N)^T$，其含义为分类器对输入样本 x 属于各个已知类的不确定度，γ 计算公式如下：

$$\gamma = \mathbf{d} \otimes (1 - softmin(\mathbf{d})) \tag{7}$$

$$softmin(\mathbf{d})_i = \frac{e^{-\mathbf{d}_i}}{\sum_{k=1}^{N} e^{-\mathbf{b}_k}} \tag{8}$$

引入阈值 θ，如果 γ 中所有的值都高于 θ，则该输入样本不属于任一类别，即作为未知类拒绝。否则，将该输入样本分类为 γ 中最小值所对应的类别标签，具体公式如下式所示：

$$decision = \begin{cases} \text{rejected as unknown} & \text{if } min(\gamma) > \theta \\ \text{class } i = argmin\gamma & \text{if } min(\gamma) \leqslant \theta \end{cases} \tag{9}$$

使用这种基于距离的决策过程可以将开放空间风险最小化：输入样本 x 离固定的类中心越远，就越有可能作为未知类拒绝。

4 复现细节

4.1 与已有开源代码对比

此章节分别对加载数据集部分、加载模型部分、训练部分和验证部分四个部分，将本次复现代码与已有开源代码进行对比。

4.1.1 与加载数据集部分

加载数据集部分的部分代码截图如下图 1,2所示，已有开源代码将不同数据集的加载进行了封装，本复现代码未进行封装，这也是复现代码需要改进的地方。

```
#Create dataloaders for training
print('==> Preparing data..')
with open('datasets/config.json') as config_file:
    cfg = json.load(config_file)[args.dataset]

trainloader, valloader, _, mapping = dataHelper.get_train_loaders(args.dataset, args.trial, cfg)

print('==> Building network..')
net = openSetClassifier.openSetClassifier(cfg['num_known_classes'], cfg['im_channels'], cfg['im_size'],
                                          init_weights = not args.resume, dropout = cfg['dropout'])
```

图 1: 源代码加载数据集部分

```python
# load dataset
if opt.dataset == 'CIFAR10':
    trainval_idxs = "dataload/CIFAR10/trainval_idxs.json"
    class_splits = "dataload/CIFAR10/class_splits/"+str(opt.split)+".json"
    with open(trainval_idxs) as f:
        trainvalIdxs = json.load(f)
        train_idx = trainvalIdxs['Train']
        val_idx = trainvalIdxs['Val']

    with open(class_splits) as f:
        classSplits = json.load(f)
        known_classes = classSplits['Known']

    train_dataset = datasets.CIFAR10(root='data/cifar10', train=True, download=False,
                                     transform=transforms.Compose([
                                         transforms.Resize(32, PIL.Image.BICUBIC),
                                         transforms.CenterCrop(32),
                                         transforms.RandomHorizontalFlip(0.5),
                                         transforms.RandomRotation(10),
                                         transforms.ToTensor(),
                                         transforms.Normalize((0.4858, 0.4771, 0.4326), (0.2422, 0.2374, 0.2547)),
                                     ]))
    val_dataset = datasets.CIFAR10(root='data/cifar10', train=True, download=False,
                                     transform=transforms.Compose([
                                         transforms.Resize(32, PIL.Image.BICUBIC),
                                         transforms.ToTensor(),
                                         transforms.Normalize((0.4858, 0.4771, 0.4326), (0.2422, 0.2374, 0.2547)),
                                     ]))

    train_known_dataset = dataset_split(train_dataset, known_classes, train_idx)
    train_val_dataset = dataset_split(val_dataset, known_classes, val_idx)

    print("size of training known dataset cifar10", len(train_known_dataset))
    print("size of training val dataset cifar10", len(train_val_dataset))

    train_loader = torch.utils.data.DataLoader(train_known_dataset, batch_size=opt.batch_size, shuffle=True)
    val_loader = torch.utils.data.DataLoader(train_val_dataset, batch_size=opt.batch_size, shuffle=True)
```

图 2: 复现代码加载数据集部分

4.1.2 加载模型部分

加载数据集部分的部分代码截图如下图 3,4所示，已有开源代码模型框架与[4]一致，本复现代码使用 ImageNet 预训练网络 EfficientNetb5[5]，使训练时网络收敛速度更快，网络参数量减少，达到与原论文一致的开集识别性能。

```python
# initialising with anchors
anchors = torch.diag(torch.Tensor([args.alpha for i in range(cfg['num_known_classes'])]))
net.set_anchors(anchors)

net = net.to(device)
training_iter = int(args.resume)
```

图 3: 源代码加载模型部分

```
if opt.dataset == 'MNIST':
    # EfficientNet
    model_name = 'efficientnet-b2'
    embedding = EfficientNet.from_pretrained(model_name.lower(), in_channels=1)
    embedding._fc = nn.Sequential()
    classifier = nn.Linear(1408, opt.num_classes)
    model = CACModel(embedding, classifier, opt.num_classes)
else:
    # EfficientNet
    model_name = 'efficientnet-b5'
    embedding = EfficientNet.from_pretrained(model_name.lower())
    embedding._fc = nn.Sequential()
    classifier = nn.Linear(2048, opt.num_classes)
    model = CACModel(embedding, classifier, opt.num_classes)

anchors = torch.diag(torch.Tensor([opt.alpha for i in range(opt.num_classes)]))
model.set_anchors(anchors)

model = model.to(device)
```

图 4: 复现代码加载模型部分

4.1.3 训练部分

训练部分的部分代码截图如下图 5,6所示。

```
max_epoch = cfg['openset_training']['max_epoch'][training_iter]+start_epoch
for epoch in range(start_epoch, max_epoch):
    train(epoch)
    val(epoch)
```

图 5: 源代码训练部分

```
train_cac(model, train_loader, val_loader, opt)
```

图 6: 复现代码训练部分

4.1.4 测试部分

测试部分的部分代码截图如下图 7,8,9所示。本次复现代码测试时，结合 odin[6]的方法，使用温度系数缩放并且在输入时添加小扰动，发现结合 odin 可以提升 CAC 的开集识别性能。

```
print('==> Evaluating open set network accuracy for trial {}..'.format(trial_num))
x, y = gather_outputs(net, mapping, knownloader, data_idx = 1, calculate_scores = True)
accuracy = metrics.accuracy(x, y)
all_accuracy += [accuracy]

print('==> Evaluating open set network AUROC for trial {}..'.format(trial_num))
xK, yK = gather_outputs(net, mapping, knownloader, data_idx = 1, calculate_scores = True)
xU, yU = gather_outputs(net, mapping, unknownloader, data_idx = 1, calculate_scores = True, unknown = True)

auroc = metrics.auroc(xK, xU)
all_auroc += [auroc]
```

图 7: 源代码测试部分

```
to_np = lambda x: x.data.cpu().numpy()
ND_labels = np.hstack(
    [np.zeros(len(test_known_dataset)), np.ones(len(test_unknown_dataset)), np.ones(len(train_unknown_dataset))])
scores_base = np.hstack(
    [np.zeros(len(test_known_dataset)), np.zeros(len(test_unknown_dataset)), np.zeros(len(train_unknown_dataset))])
scores = np.hstack(
    [np.zeros(len(test_known_dataset)), np.zeros(len(test_unknown_dataset)), np.zeros(len(train_unknown_dataset))])

idx = 0
for loader in [test_known_loader, test_unknown_loader, train_unknown_loader]:
    for step, (inputs, labels) in enumerate(loader):
        inputs = Variable(inputs.to(device), requires_grad=True)
        labels = torch.Tensor([opt.mapping[x] for x in labels]).long().to(device)

        # Calculating the confidence of the output, no perturbation added here, no temperature scaling used
        outputs = model.forward1(inputs)
        # Calculating the confidence of the output, no perturbation added here, no temperature scaling used
        nnOutputs = to_np(outputs)
        scores_base[idx:idx + len(nnOutputs)] = np.min(nnOutputs, axis=1)

        # Using temperature scaling
        outputs = outputs / opt.temperature

        # Calculating the perturbation we need to add, that is,
        # the sign of gradient of cross entropy loss w.r.t. input
        cacLoss, anchorLoss, tupletLoss = CACLoss(outputs, labels, opt, device)
        cacLoss.backward()

        # Normalizing the gradient to binary in {0, 1}
        gradient = torch.ge(inputs.grad.data, 0)
        gradient = (gradient.float() - 0.5) * 2
```

图 8: 复现代码测试部分 (1)

```
        gradient = (gradient.float() - 0.5) * 2
        # Normalizing the gradient to the same space of image
        if opt.dataset == 'MNIST':
            gradient = (gradient) / (66.7 / 255.0)
        else:
            gradient[:][0] = (gradient[:][0]) / (63.0 / 255.0)
            gradient[:][1] = (gradient[:][1]) / (62.1 / 255.0)
            gradient[:][2] = (gradient[:][2]) / (66.7 / 255.0)
        # Adding small perturbations to images
        tempInputs = torch.add(inputs.data, -opt.magnitude, gradient)

        outputs = model.forward1(Variable(tempInputs))
        outputs = outputs / opt.temperature
        # Calculating the confidence after adding perturbations
        nnOutputs = outputs.data.cpu()
        nnOutputs = nnOutputs.numpy()
        # nnOutputs = nnOutputs - np.max(nnOutputs, axis=1, keepdims=True)
        # nnOutputs = np.exp(nnOutputs) / np.sum(np.exp(nnOutputs), axis=1, keepdims=True)

        scores[idx:idx + len(nnOutputs)] = np.min(nnOutputs, axis=1)
        idx += len(nnOutputs)

print('BASELINE: AUC ROC: %f' % roc_auc_score(ND_labels, scores_base))
print('ODIN: AUC ROC: %f' % roc_auc_score(ND_labels, scores))
```

图 9: 复现代码测试部分 (2)

4.2 实验环境搭建

本次复现环境使用 NVIDIA A100 单张 GPU，python3.7，numpy1.18.2，torch1.4.0 进行训练以及测试。

4.3 创新点

4.3.1 更改原网络模型

原论文中使用的网络是与[4]一致，本次复现更改网络为 ImageNet 预训练网络 Efficientb5[5]，对比原论文所使用的网络，使用更少的参数量以及更少的计算量，并且使用预训练网络能够加快网络训练收敛速度，带来很好的性能。

4.3.2 使用温度系数并在输入时添加小扰动

该方法方法是 Liang 等[6]提出，使用温度系数，并且在输入时添加小扰动可以拉大已知类与未知类在 logit 空间中的距离。并且，该方法可以与不同的网络架构以及数据集相容，不需要额外的训练。通过实验发现，ODIN 方法与 CAC 方法的结合能够提升开集识别性能。

5 实验结果分析

本部分对实验所得结果进行分析，详细对实验内容进行说明，实验结果进行描述并分析。

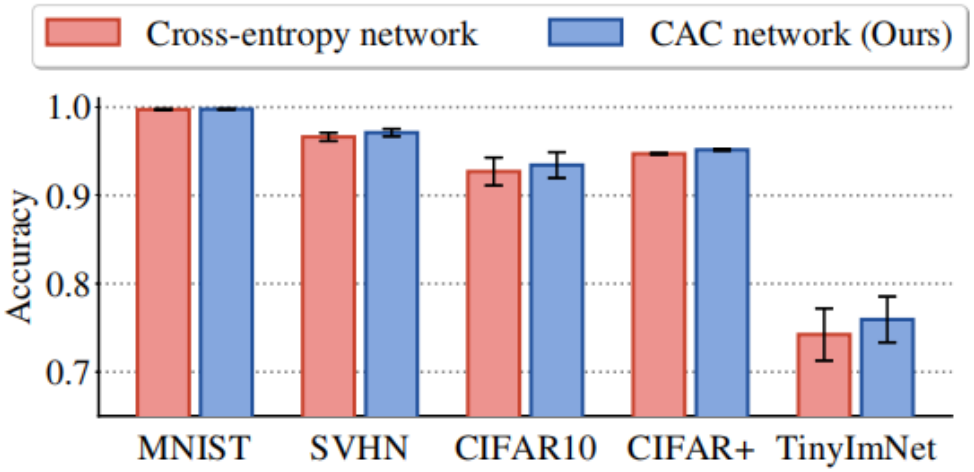

Figure 3: Our CAC classifier maintains the classification accuracy of a standard classifier trained with cross-entropy loss.

图 10: CAC 原论文闭集分类准确度

	CIFAR10	TinyImageNet
Ours(EfficientNet)	93.10	76.10

	CIFAR10	TinyImageNet
CAC	80.1	76.0
Ours	83.47	76.14
Ours+odin	86.51	77.81

6 总结与展望

CAC 基于距离，提出了新的损失函数，通过固定类中心和聚类的方式，解决已知类别类间不紧凑以及类边界不清晰的问题，在开集识别任务中达到不错的效果。

在本次复现中，替换原论文的深度神经网络为 ImageNet 预训练模型 EfficientNetb5，使用 CAC 进行 fine-tune，可以缩短训练时间，并且在网络参数量减少的情况下，达到同样的性能表现。并且使用 odin 论文中的方法，使用温度系数缩放并且在输入时添加小扰动，实验结果发现 CAC 和 odin 方法的结合可以有效提升开集识别的性能。

在数据集 tinyImageNet 上达到和论文中一样的精度，结合 odin 后超过原论文 1.7%；在数据集 cifar10 上比论文高 3%，结合 odin 后超过原论文 6.1%。

综上，本次复现任务受益颇多，不仅提升了代码能力，以动手实践的方式更加理解论文，更深的认识如何训练一个模型。

参考文献

[1] MILLER D, SÜNDERHAUF N, MILFORD M, et al. Class Anchor Clustering: a Loss for Distance-based Open Set Recognition[J]. workshop on applications of computer vision, 2020.

[2] BENDALE A, BOULT T E. Towards Open Set Deep Networks[J]. computer vision and pattern recognition, 2016.

[3] YOSHIHASHI R, SHAO W, KAWAKAMI R, et al. Classification-Reconstruction Learning for Open-Set Recognition[J]. arXiv: Computer Vision and Pattern Recognition, 2018.

[4] NEAL L, OLSON M L, FERN X Z, et al. Open Set Learning with Counterfactual Images.[J]. european conference on computer vision, 2018.

[5] TAN M, LE Q V. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks[J]. international conference on machine learning, 2019.

[6] LIANG S, LI Y, SRIKANT R. Enhancing The Reliability of Out-of-distribution Image Detection in Neural Networks[J]. Learning, 2017.