上海大学

SHANGHAIUNIVERSITY

毕业论文（设计）

UNDERGRADUATE THESIS (PROJECT)

题目:基于卷积神经网络的增量学习方法

学院	计算机工程与科学学院
专业	智能科学与技术
学号	18121959
学生姓名	李晓昳
指导教师	马丽艳
起讫日期	2022.02.21－2022.06.03

目录

基于卷积神经网络的增量学习方法

摘要

增量学习目前在深度学习领域逐渐成为热门研究课题，支持模型从持续输入的数据流中进行学习，该训练模式可以在更有效使用资源的同时更接近人类的学习模式。其算法包括基于微调的传统增量学习方式以及基于深度学习的新增量学习方式。因增量学习所处理的数据为持续输入而非固定的，传统微调方法或者基于分类器的改进方法易被数据及特定分类器的局限所影响，难以在多项任务上得到拓展。随着深度学习相关研究已在众多人工智能领域都得到广泛应用，依托卷积神经网络进行增量学习方法研究已日渐成为主流，对于图像分类、目标识别、图像分割等领域均具有重要意义。目前基于卷积神经网络的增量学习方法也逐渐应用至各分支，其可行性及鲁棒性已被证明，最初研究工作集中在任务增量学习，现已向类增量学习以及领域增量学习转变。

本文主要研究基于卷积神经网络的增量学习方法在图像分类任务上的应用，对基于池化知识蒸馏的类增量学习模型（PODNet）上做了四方面改进，一是代替网络特征选择对新旧模型分类结果做知识蒸馏，使得新旧模型分类结果更接近；二是在分类网络中引入注意力机制，使得网络中间特征先后通过通道和空间两个维度的注意力机制，更有效提取图像信息；三是设置记忆率动态对每个任务学习过程中旧样本数量进行选择；四是对于输入数据集做了图像压缩处理，使得同样大小的空间下能存储更多图像进行训练。本文对 PODNet 的改进在 CIFAR100 数据集上平均准确率取得了 1～3%的提升。

关键词：卷积神经网络，增量学习，知识蒸馏，图像分类，注意力机制

Incremental Learning Method Based on Convolutional Neural Network

ABSTRACT

Incremental learning is an increasingly popular research topic in the field of deep learning, supporting models to learn from a continuous stream of input data. The algorithms can be divided into traditional incremental learning methods. As incremental learning deals with data that is continuously input, traditional fine-tuning methods tend to be difficult to apply on multiple tasks. With the widespread use of deep learning methods in a wide range of fields, research on incremental learning based on convolutional neural networks (CNN) has gradually become mainstream in areas such as image classification. The feasibility and robustness of incremental learning methods based on CNN have been demonstrated in various branches, initially focusing on task-based incremental learning, but now shifting to class-based incremental learning and domain-based incremental learning.

This paper focuses on the application of incremental learning methods based on CNN to image classification tasks. Four improvements are made to the pooled knowledge distillation-based class incremental learning model (PODNet). Firstly, doing knowledge distillation on the classification results instead of network feature selection, so that the old and new model classification results are closer to each other; Then, introducing attention mechanism to make the intermediate features be more effective in extracting image information in both channel and space dimensions; Thirdly, setting memory rate to dynamically select the number of old samples in the learning process for each task; Lastly, doing image compression for the input dataset, so that more images can be stored for training in the same size space. The improvements to PODNet in this paper achieved an average accuracy improvement of 1% ~ 3% on the CIFAR100 dataset.

Keywords: Convolutional Neural Network, Incremental Learning, Knowledge Distillation, Image Classification, Attention Mechanisms

第 1 章 绪论

本章主要阐明增量学习的背景和意义，分析相关方向的国内外研究现状、当前存在的问题以及现阶段研究遇到的难点，最后提出本文即将研究的内容及目标。

§1.1 课题背景及其意义

随着数据库以及互联网技术的发展，各行业部门积累了大量数据，且数据量每天都会呈指数级增长从而导致未处理数据的堆积，如何处理这些多变的数据并且有效提取有用信息是当前研究热点。传统机器学习是在批量设置中进行训练的，无法处理不断变化数据且难以将新数据整合到已建立的系统模型，进而导致模型准确率降低的情况。因此，需要通过增量学习方法在计算和存储资源有限的条件下从非平稳数据中获取新知识，同时克服灾难性遗忘问题。

增量学习通过近年来的研究，现已在多个领域获得进展，主流方法也从早期基于微调的方式转变成基于回放、正则化约束以及网络结构的深度学习方式。由于该问题本身可适应场景的复杂性和待处理任务的多样化，增量学习的研究被限定在特定设置下，目前逐渐发展成任务增量学习、领域增量学习以及所涉范围更广的类增量学习三个增量学习范式。

随着深度学习的进一步发展，卷积神经网络在图像分类、图像分割、图像分割等领域均得到较好应用，对于图像特征的提取能力越来越强，为了使得上述任务能够在同等条件下适应不断输入的数据集且仍维持较高的准确率。本课题主要基于卷积神经网络进行图像特征提取和结果分类，利用回放思想存储部分旧任务样本，和新任务样本共同组成新任务训练集，以限制新模型仍能记忆旧任务上学习的特征；利用正则化思想通过对新旧模型最后分类结果以及中间层特征进行池化后引入蒸馏操作以解决灾难性遗忘问题，并基于表征学习思想代替单一特征选择多特征进行余弦分类器损失函数计算，更好适应同一类别内多样性需求。结合多种增量学习方法实现具有学习能力的图像分类。

§1.2 增量学习的研究现状

目前主流的基于深度学习的增量学习策略可以划分为基于正则化的增量学习方法[3][5][6][7][9][10][15]、基于回放的增量学习方法[6][8][9][10][15][18]以及基于网络结构的增量学习方法[11][12][13]。基于正则化的增量学习方法是通过损失函数的设计进行知识保护，主要针对新任务中已更新模型的输出结果增加约束以保护旧知识不被遗忘；基于回放的增量学习思想主要是在训练新任务时，训练数据部分来自于具有代表性的旧数据，采用新旧数据结合的方式使得模型减少对先前知识的遗忘，重点在于如何选择代表性旧样本

以及新旧样本存储比例的问题；基于网络结构的方法是通过修改网络结构，冻结部分神经元或者动态扩展网络等方法减少灾难性遗忘。以下将分别介绍上述三类主流增量学习方法的研究现状。

§1.2.1 基于正则化的增量学习方法的研究现状

基于正则化的增量学习方法是利用在损失函数中加入某个特定的正则化项来减少灾难性遗忘现象，核心思想是限制参数的更新以提高模型稳定性，从而缓解灾难性遗忘。根据不同关注点，正则化策略可以进一步分为权重正则化策略以及蒸馏正则化策略。

权重正则化策略根据对重要性权重的测量，通过限制学习率来保护旧知识。神经网络模型的权重将通过反向转播(BP)[1]和随机梯度下降(SGD)[2]更新，而在增量学习的情况下，由以前数据训练出来的旧模型权重会被更新为更适应新知识的版本，从而导致灾难性遗忘。通过识别对旧任务有较大影响的参数并抑制其更新，可以使得模型在学习新知识的同时保护对旧知识的掌握。其中，2017 年提出的 EWC[3]模型是权重正则化的代表性模型，其关键为通过 Fisher 信息矩阵[4]来评估权重参数的重要性，并借此观察观察随机变量所携带的信息。图 1-1 为 EWC 模型算法示意[3]。

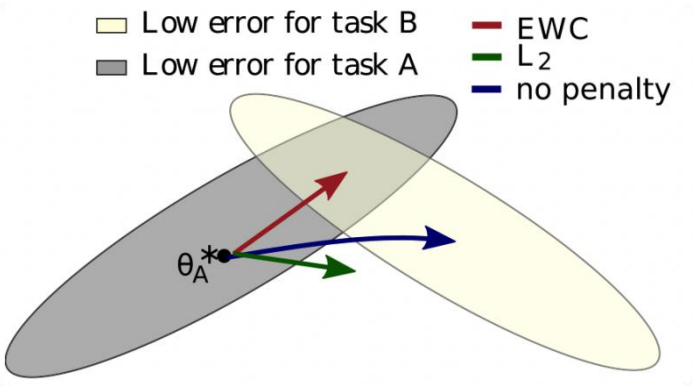

图 1-1 EWC 算法描述

EWC 模型因为是在不扩大网络和保留旧数据的情况下缓解灾难性遗忘的方法，可以有效节省存储空间，然而 EWC 只考虑了最后学习阶段而非所有学习阶段的 Fisher 信息矩阵，因此存在区间遗忘问题。Zenke 等人提出的 SI[5]模型通过计算训练新任务后在欧几里得空间的距离差的累计变化判断权重重要性，数值越大意味该权重对该任务的影响越大，相比 EWC 能更直观地衡量参数重要性。Aljundi 等人提出的 MAS[6]模型是增量学习领域第一个无监督学习方法，根据模型敏感性衡量权重重要性，其中模型敏感性通过经 L2 正则化的输出的梯度计算得到。

蒸馏正则化策略通过对新旧模型的输出进行约束，将旧模型针对同一类任务的输出结果作为新模型对该任务输出结果的约束，通过训练使两者差异最小。2016 年 Li 等人

提出的 LwF[7]模型是知识蒸馏在增量学习方法中的开山之作，该模型对每次传入的任务训练单独分类器，新任务的数据根据就模型获得的输出进行标注，这些标注被用来约束知识蒸馏模型参数的更新，由此知识蒸馏被广泛应用于增量学习方法，配合其他策略达到较好的效果。图 1-2 为 LwF 算法结构图[7]。

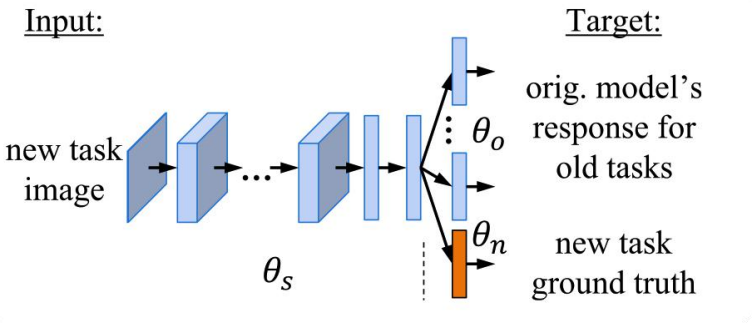

图 1-2 LwF 算法描述

§1.2.2 基于回放的增量学习方法的研究现状

基于回放的增量学习方法的整体思路是，为了避免新模型对于先前知识的遗忘，则每次训练后保留部分旧知识共同参与后续训练，使得新模型在训练时除了学习新知识，对于旧知识一起进行回顾，而非每个任务割裂学习新知识，减轻灾难性遗忘。对于旧知识的存储，除了存储样本的选择上有所不同，对于待存储的知识也有多种方式，其一为存储原始数据样本，其二为存储伪样本[8]，即针对原始数据样本构建包含特征的与原始数据非常接近的伪样本，新训练过程将同时收到伪样本和新样本的监督。

2017 年 Rebuffi 等提出的 iCaRL[9]模型奠定了回放机制在增量学习领域中的地位，在此之后的众多模型都是基于 iCaRL 做的改进。图 1-3 为 iCaRL 算法示意图[9]，iCaRL 是类增量学习方法，同时结合知识蒸馏策略进行学习，对于旧样本的存储选择，首先计算每个类所有样本的平均特征向量，其次选择类内与平均特征最接近 k 个样本进行存储。每当有新数据传入时，模型会动态提取已获得数据类平均特征并同步计算保留的样本，对记忆缓冲区(memory buffer)内存储的样本进行更新。

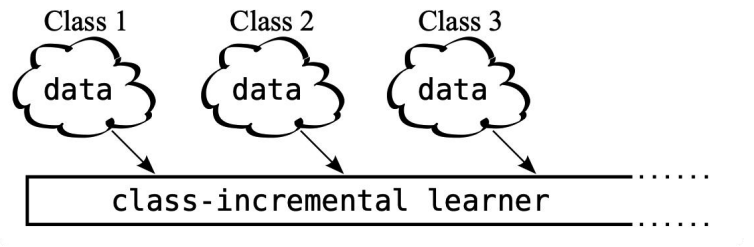

图 1-3 iCaRL 算法描述

鉴于不同旧样本的存储选择会对模型效果有着较大影响，以及有限的存储内存造成存储的旧样本和待训练的新样本之间有着较大的不平衡。为了克服这一困难，Hou

等人设计的 UCIR[10]模型选择余弦相似性对输出层进行修改，消除分类器权重的影响，并结合知识蒸馏策略，直接对特征层而非输出结果层进行约束，更好保留旧类别信息，图 1-4 为 UCIR 算法示意图[10]。

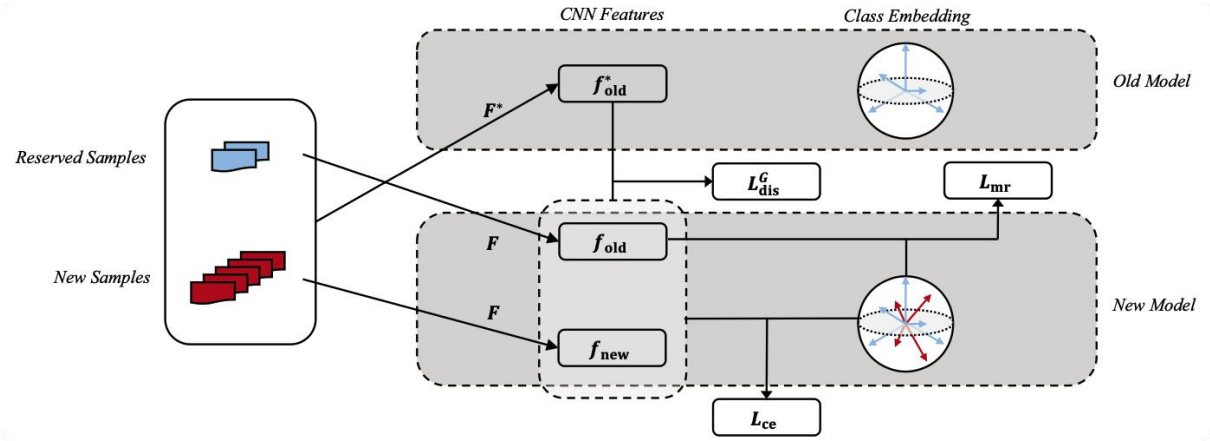

图 1-4 UCIR 算法描述

§1.2.3 基于网络结构的增量学习方法的研究现状

基于网络结构的增量学习策略是为每个连续的增量任务训练单独模型，然后设置选择器决定推理阶段选择哪一个模型。Rusu 等人提出渐进式神经网络 Progressive Net(PNN)[11]模型，相比先前任务所训练的网络参数是固定的以防止灾难性遗忘，PNN 在训练一个新任务时，首先考虑之前网络输出，引入先前学习的知识经验，通过保留已训练好的结构的方式保护新模型在旧任务上的性能，图 1-5 为 PNN 网络结构图[11]。

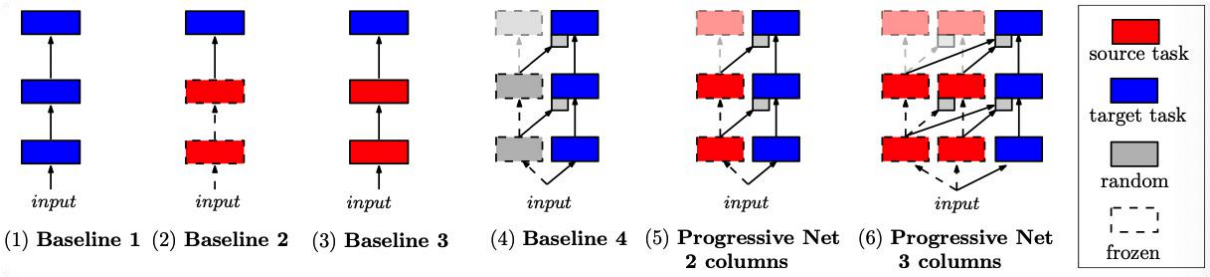

图 1-5 Prograsive Net Architecture 网络结构（右 2）

由于 PNN 在训练过程中需不断扩展网络，造成网络容量的消耗，为了更有效利用网络并且节省空间的消耗，一种动态确定网络结构的模型由此产生。Yoon 等人提出的基于相关性的动态扩展网络 DEN[12]可以在系列任务中动态决定网络容量，在每个任务到来时，只用必要神经单元进行容量扩展，同时通过分割或者复制神经单元的方式有效防止语义漂移，节省存储空间，如图 1-6 所示[12]。

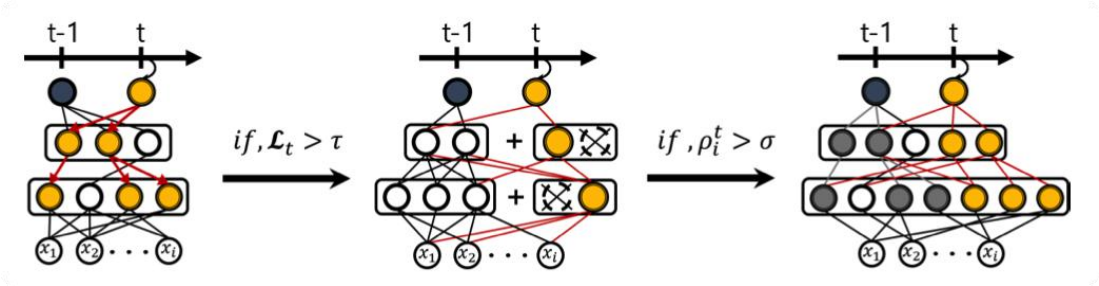

图 1-6 DEN 网络结构

Gepperth 等人提出的 SOM[13]模型使用自组织的图重组二维坐标网络，只在输入与先前任务不同时才更新，以防止模型快速变化，在网络扩展方面与 DEN 有类似的优势。

§1.3 增量学习存在的问题

对于目前增量学习方法在系列问题上的应用，主要存在以下几个问题[14]：

(1) 存储限制，即对于一些受物理存储空间的系统或者场景来说，无法存储不同阶段看到的所有数据，无法采用联合训练策略，因此系统在不同时间或者面临不同任务时仍需能够执行先前学过的任务，难度较大。

(2) 数据隐私安全限制，即对于一些应用场景。受法律保护用户数据不能在系统存储空间上进行留存，因此渐进式学习是该类场景下持续提供服务的关键。

(3) 领域适应，即对于部分在特定领域进行的训练方案通常是有监督的学习方式，对于动态变化程度较高的真实环境难以直接应用。

(4) 类别差异，现阶段对于增量学习方法其研究对象如图像分类任务，通常限制在特定数据集范围内，即隐性要求任务性质差异需维持在较小范围内，当任务本质性质和待处理问题难度差异太大时，会造成部分增量学习方法性能严重下降，相较于普通分类模型的结果甚至都可能无法达到。

(5) 参数敏感，由于增量学习任务大多依赖于模型以及系列优化策略，对于超参数、数据性质、模型结构等设定都很敏感，所以寻找一组能适应于各种任务场景的超参数需要不断实验。

(6) 记忆限制，即增量学习任务存在背景是需在不同任务中获得持续学习知识的能力，在一些增量学习方法探讨过程中利用额外超参数的引用达到模型稳定性和可塑性之间的平衡，但该方法本质上违反了当前任务无法获得未来数据的前提，所以在实际迁移至真实生产环境时会导致效果降低。

§1.4 本课题的研究难点

(1) 本课题使用 Pytorch 的深度学习框架进行增量学习在图像分类任务上的模型构建，对于 Python 以及 Pytorch 各类库的应用以及常用深度学习框架的搭建需要较为熟悉。

(2) 对于增量学习任务，不同任务分配以及设置会得到不同结果，通过阶段任务持续学完所有的类才能得到最终结果，训练时间一般较长。

(3) 损失函数的设计对模型结果有较大影响，基于不同策略的增量学习方法所产生的效果需逐一进行实验对比，同时参数选择对于模型准确率也有影响，需多做实验进行合适参数的选取。

(4) 采用公开的图像数据集进行训练，虽然数据集来源易获得，但是整体训练周期比较长，特别是对于含有较多图像数据的数据集来说，对训练环境的要求较高，需租 GPU 服务器进行训练。

§1.5 本文研究内容及目标

Douillard 等人提出基于池化知识蒸馏策略的用于小任务增量学习的模型(PODNet)[15]结合了正则化策略以及回放策略，在 CIFAR100、ImageNet1000 等数据集上都取得了当时 SOTA 的效果，证明了多种增量学习策略叠加使用的可行性以及池化操作对知识蒸馏策略的效果改进，能更好的在模型稳定性和可塑性之间得到平衡。本文首先基于 Pytorch 框架对 PODNet 进行复现，同时受到传统知识蒸馏思想的启发对 PODNet 知识蒸馏板块进行改进；此外，基于注意力机制在众多领域的出色应用，在原先模型中引入注意力机制模块获取更有代表的图像特征；并采用动态调整方法对每类存储样本数量进行分配；最后考虑到图像本身 RGB 性质，利用图像压缩方式尽可能扩大同一容量内对样本存储个数的提升，最终对上述四类修改作相应实验并与原先实验结果进行对比。

§1.5.1 研究内容

本文主要针对 PODNet 算法中增量学习策略部分进行改进，具体内容概括如下：

(1) 将 PODNet 对新旧模型特征提取器板块输出特征进行知识蒸馏，选择对分类结果代替蒸馏正则化进行约束，以更直观的蒸馏方式限制新模型在旧任务上的表现结果。

(2) 在 PODNet 模型分类器部分先后引进通道注意力模块和空间注意力模块，通过两类注意力机制的同时引入，使得图像分类模型更好的捕捉到图像特征，基于任务提升分类效果。

(3) 在 PODNet 使用 herding 策略选择旧样本进行存储的基础上，代替原先每类旧样本存储限定个数的设置，使用动态调整策略对每类样本设置不同记忆存储率控制每阶段所有待存储样本的类内个数，更好利用固定大小的记忆存储库。

(4) 对 PODNet 采用的数据集本身进行图像压缩处理，在保证图像质量的要求下尽可能压缩图像大小，使得同等大小的记忆存储库能存放下更多的样本，解决旧样本和新样本样本数量不均衡的问题。

§1.5.2 研究目标

针对本文研究内容，制定了如下研究目标：

(1) 学会熟练使用 Pytorch 框架以及基于卷积神经网络的图像分类模型的应用，同时对增量学习方法等前沿知识进行掌握。

(2) 能够基于 PODNet 算法思想对类增量学习模型进行复现，有效验证其性能。

(3) 通过深入阅读类增量学习领域的最新研究文章，对常用增量学习方法策略进行了解，并能根据原始模型融入不同改进方式，并对改善后的实验结果以及准确率指标等进行测试，通过消融实验和进一步对比实验做出结论判断。

§1.6 本文组织结构

整篇论文共分为五个章节。

第一章介绍了增量学习方法相关研究背景和意义，根据国内外现阶段对类增量学习的研究现状、目前存在的灾难性遗忘问题以及本课题研究难点，提出本文研究内容以及研究目标。

第二章介绍了增量学习方法在卷积神经网络上的相关理论研究，首先介绍增量学习方法的基本框架以及卷积神经网络结构，其次针对类增量学习模型的组成进行公式化描述，并着重讲解了增量学习在图像分类上的学习策略和类增量学习领域常用数据集。

第三章详细介绍了 PODNet 模型的基本思想以及四种改进方式，是全文核心章节。首先介绍了 PODNet 的模型结构以及核心损失函数定义方式以及所使用的增量学习策略，其次介绍了针对 PODNet 的四方面改进措施。

第四章介绍了本文为实现 PODNet 模型所依赖的开发环境、训练细节以及学习策略，并对第三章所提及的四类改进措施分别进行对比实验、消融实验以及融合实验。

第五章是针对全文有关增量学习内容的总结和展望，归纳了本课题在类增量领域中的工作内容与创新点，并同时提出目前工作中的不足之处及其未来可参考改进方向。

第 2 章 基于卷积神经网络的增量学习方法理论基础

本章主要讲述卷积神经网络下增量学习方法的理论知识，首先介绍增量学习方法基本框架以及卷积神经网络结构，其次针对类增量学习模型的组成进行公式化描述，最后着重讲解了增量学习在图像分类上的学习策略以及类增量学习领域常用数据集。

§2.1 增量学习的问题构建

在本节，将介绍基于深度学习的增量学习问题基本描述。

通常，由于增量学习任务的数据来源为持续输入的数据集，所以一个增量学习问题$\mathcal{T}$由一连串n个任务组成，公式描述为(2-1)。

$$\mathcal{T} = \left[(C^1, D^1), (C^2, D^2), ..., (C^n, D^n)\right] \tag{2-1}$$

其中每个任务t由一组类组成$C^t = \{c_1^t, c_2^t, ..., c_{n^t}^t\}, (C^i \cap C^j = \emptyset, if\ i \neq j)$，对应类别训练集为$D^t$。对于已学习的所有类别个数$N^t$，公式记作$N^t = \sum_{i=1}^{t} |C^i|$，包含从第 1 个任务到当前$t$任务所学习到的所有类。$D^t$为对应$t$任务的训练集，假定该数据集共含有$m^t$个样本，$x$代表训练样本的输入特征，$y \in \{0,1\}^{N^t}$代表对应$x_i$的 one-hot 编码真实标签，对应公式为(2-2)。

$$D^t = \{(x_1, y_1), (x_2, y_2), ..., (x_{m^t}, y_{m^t})\} \tag{2-2}$$

本文所研究的为深度学习下基于卷积神经网络的增量学习方法，将网络结构整体拟合成一个参数为θ的模型，定义$o(x) = h(x; \theta)$作为模型在输入x条件下的全连接层输出(logits)。在特定领域比如类增量学习领域中，对于图像分类任务的处理一般神经网络会分为特征提取器f部分以及线性分类器g部分，其中特征提取器f受参数φ控制，线性分类器g受参数V控制，则$o(x)$可改写成公式(2-3)。

$$o(x) = g(f(x; \varphi); V) \tag{2-3}$$

对于神经网络输出结果的预测，可表示为$\hat{y} = \sigma(h(x; \theta))$，其中$\sigma$为 softmax 函数，$\hat{y}$即为网络最终对于输入$x$的分类结果输出。

类增量学习任务的效果评估通常是针对所有已学习过的类进行效果评估，即针对t任务上学习到的C^t类数据，需评估其在所有$\bigcup_{i=1}^{t} C^i$已学习类上的分类准确率。

有关损失函数的定义，通常类增量任务采用交叉熵损失函数进行计算，当只对当前任务t的数据进行训练时，其一损失函数是考虑截止当前任务为止的所有类别的交叉熵，具体公式如(2-4)所示。

$$\mathcal{L}_c(x, y; \theta^t) = \sum_{k=1}^{N^t} y_k \, log \frac{exp(o_k)}{\sum_{i=1}^{N^t} exp(o_i)} \tag{2-4}$$

其二损失函数的定义是只考虑当前任务t针对在该任务学习的C^t类上的网络输出，具体公式如(2-5)所示。

$$\mathcal{L}_{c*}(x,y;\theta^t)=\sum_{k=1}^{|C^t|} y_{N^{t-1}+k}\, log\frac{exp(o_{N^{t-1}+k})}{\sum_{i=1}^{|C^t|} exp(o_{N^{t-1}+i})} \tag{2-5}$$

§2.2 卷积神经网络

本节重点讲述卷积神经网络的网络结构，该网络通常在图像分类、图像分割等各领域任务都有较为广泛的应用。

卷积神经网络相较于人工神经网络来说区别不大，由通过学习进行自我更新迭代的神经元组成。可以接受图像作为输入，为图像中各个目标分配重要性参数（可学习权重和偏差），同时具有区分不同目标的能力。

因为卷积神经网络以图像形式作为输入，所以卷积神经网络中的各层神经元由三个维度所构成，依次是宽度(width)、高度(height)以及深度(depth)。一个基本卷积神经网络是由不同层堆叠所构成的，每层利用一个可微分函数完成激活量的变换，而组成卷积神经网络的主要层类型是卷积层(Convolutional Layer)、池化层(Pooling Layer)和全连接层(Fully-Connected Layer)，利用各层的相互堆叠，最终形成完整卷积神经网络结构。

例如针对 CIFAR10 数据集（图像尺寸为 $32\times32\times3$）的分类任务，卷积神经网络结构由输入层、卷积层、激活层、池化层以及全连接层等构成。对于如图 2-1 这样含有五层网络结构的 CNN 来说，输入层将存储 $32\times32\times3$ 原始图像像素值，对应图像宽度为 32、高度为 32、色彩通道为 3 个（R、G、B）；卷积层将计算与部分输入的相连区域神经元的输出，每个神经元输出都代表了连接区域间与其权重间点积操作的值，若使用 12 个过滤器(filter)，则对应图像维度为 $32\times32\times12$；激活层可选择常见激活函数在体积不变的情况下对卷积输出进行映射；池化层沿着宽度和高度两个空间维度进行降采样处理，生成图像维度为 $16\times16\times12$；各类别得分将由全连接层计算得出，生成维度为 $1\times1\times10$，其中 10 对应 CIFAR10 数据集的本身类别数 10。

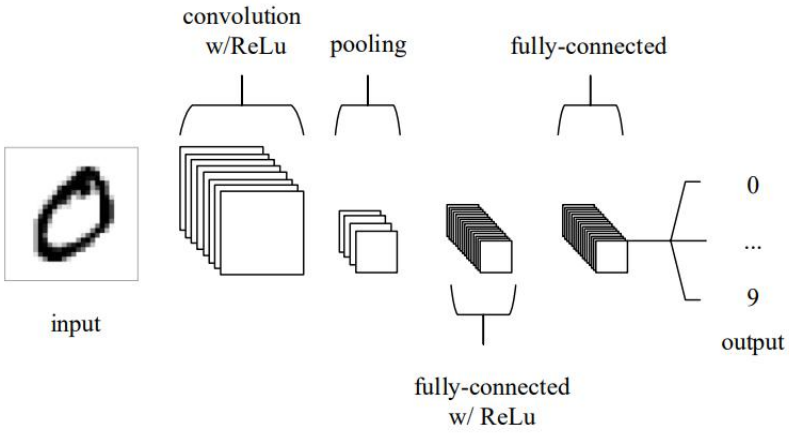

图 2-1 CNN 五层结构示意图

总结而言，卷积层由一组远小于输入图像尺寸的滤波器（卷积核）构成，通过在空间上进行稀疏运算即滑动窗口计算过程与图像进行融合，从而创建激活图；池化层通过对经过卷积层输出的图像特征进行下采样处理以实现特征图降维运算，从而实现减少过拟合的目的；激活层通过选取不同激活函数，并导入非线性因素，以提升神经网络中对模型特征的表达能力；全连接层通过实现将已学习特征到样本标记空间的映射，完成分类及回归任务。

§2.3 类增量模型的组成构件

本节介绍基于卷积神经网络的类增量学习模型的基本框架，主要分为骨干网络、旧数据存储以及损失函数三个板块。

§2.3.1 骨干网络

类增量学习本质是基于图像分类任务，和传统图像分类任务相比，主要应对新类别数据的连续输入，模型需正确将每个任务学习的新数据进行类别分类。目前大多数图像分类任务采用 CNN 模型进行，第一步便是基于 Backbone 核心框架对对图像进行特征编码，此处将介绍本文使用到的骨干网络 ResNet。

深度学习中随着网络层数不断堆叠，主要面临梯度消失或梯度爆炸等问题，对网络性能造成影响。为克服训练深层次网络所带来的上述问题，He-Kaiming 团队于 2015 年提出 ResNet[16]结构模型，引入残差网络结构，通过使用多个有参层进行输入输出间残差表达的学习，导致有参层学习残差收敛速度更快，同时更进一步提升分类准确度，核心思路是引入恒等的捷径连接。如图 2-2 所示[16]，$H(x)$代表理想映射，$F(x)$代表残差映射，$H(x)=F(x)+x$，通过对拟合目标函数$H(x)$进行重新拟合的处理，将输出合并为拟合和输入的残差叠加，使得网络能更灵活地应对输出$H(x)$与输入x之间存在的微小变动。通过层与层之间的捷径连接，使得我们能够训练比先前更深的网络。

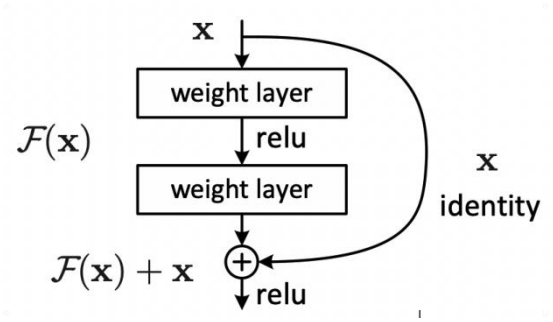

图 2-2 添加捷径连接的残差网络单元

ResNet 基于不同残差模块和网络层数的组合，可以搭配形成具有不同深度且更具有针对性的网络结构，本文中所采用的是 ResNet-32 结构，其网络结构如图 2-3 所示，网络由三个主要部分组成：输入部分、输出部分以及卷积组部分。ResNet 网络输入由

卷积核（$size = 7 \times 7, stride = 2$）以及最大池化层（$size = 3 \times 3, stride = 2$）组成，卷积组部分通过大小为 3×3 的卷积堆叠实现信息提取，每个 block 重复堆叠次数为 [5,4,3,2]，四个卷积组后经过平均池化下采样层，将特征图大小调整为 1×1，输出部分即为全连接层，输出节点个数与预测类别个数一致。

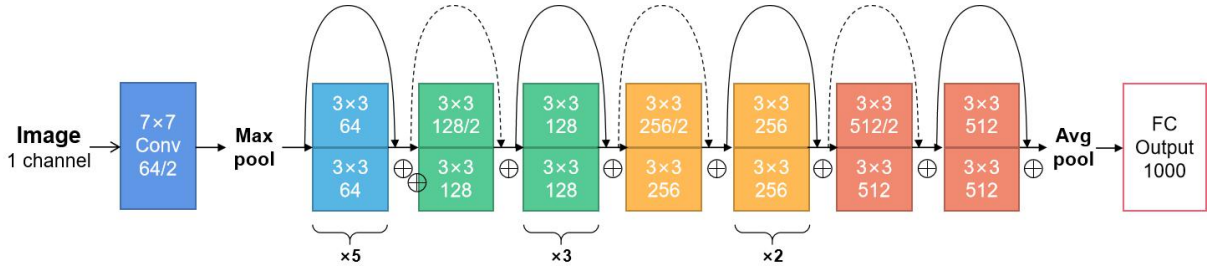

图 2-3 ResNet32 网络结构图

在类增量学习任务中，通常采用 ResNet 模型作为预训练模型，改写网络去掉最后一层全连接层以适应类增量学习过程中总分类输出结果的阶段性增加。

§2.3.2 旧数据存储

增量学习任务中基于回放的思想通常需要在每阶段任务训练结束后存储部分旧样本以供后续阶段和新样本一起进行训练。而存储空间一般情况下是恒定的，这限制了每类样本存储数量以及引起对旧样本的选择性存储。

一种类增量学习方法对于旧数据的存储是假设存储空间大小为M，所有待学习数据的类别总数为N，则对于每类样本设置静态存储数量即M/N，每阶段任务t学习后对该任务中所学习到的C^t类别分别存储M/N个样本，而对于样本选择通常基于与类内平均特征的相似性程度进行排序选择，即为 herding 选择策略，效果如图 2-4 所示[9]。

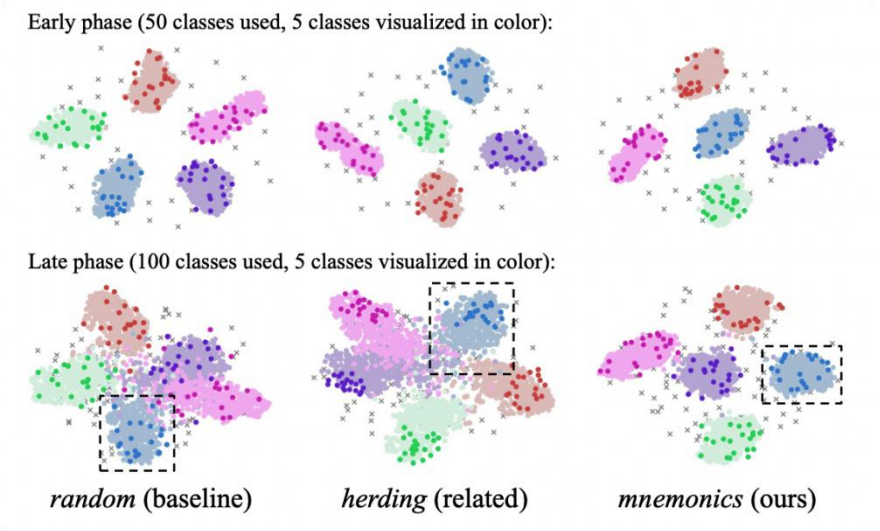

图 2-4 herding 选择策略

另一种类增量学习方式选择动态方式更新存储样本，iCaRL 模型动态在每阶段任务训练后对已存储旧样本和待存储样本都重新进行选择，RMM 模型则是在训练开始前设置两阶段任务，结合强化学习想法先在一阶段动态学习最合适的新旧样本存储比例，确定旧样本存储数量，再利用二阶段强化学习方法训练最适宜的旧样本间各类存储数量，完成各类旧样本的回放，模型如图 2-5 所示[21]。

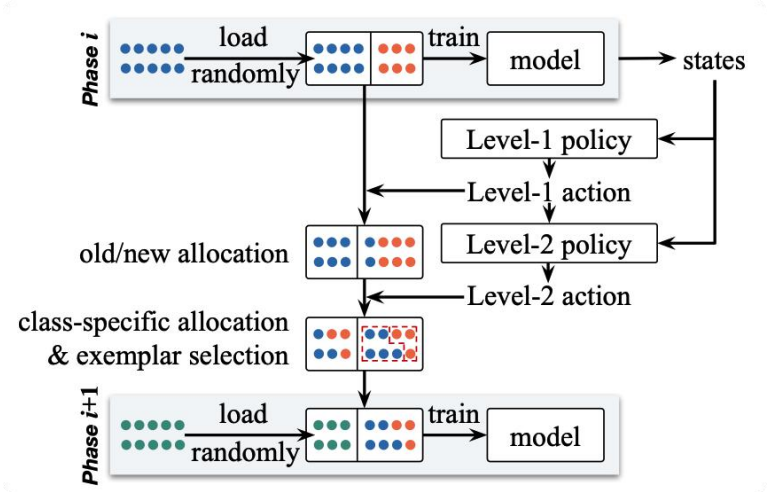

图 2-5 RMM 两阶段算法模型

§2.3.3 损失函数

损失函数的应用在增量学习任务中起到很强大的作用，如对于类增量学习，模型需在持续不断学习新知识的同时维持对先前已学习知识的掌握，最大程度减少灾难性遗忘。而通过不同损失函数定义，比如三大增量学习策略之一的正则化策略，就是引入知识蒸馏作为损失函数进行模型效果限制，如图 2-6 所示。

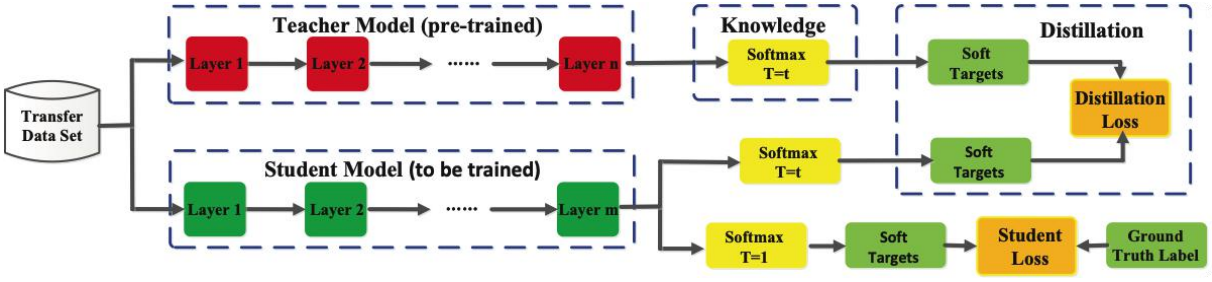

图 2-6 知识蒸馏模型结果

最基本的损失函数定义公式如(2-6)所示，由分类损失函数$\mathcal{L}_c$和知识蒸馏损失函数$\mathcal{L}_d$共同组成，其中$\lambda \in [0,1]$为超参数权重。

$$\mathcal{L} = \lambda \times \mathcal{L}_c + (1-\lambda) \times \mathcal{L}_d \tag{2-6}$$

对当前学习阶段任务t的知识蒸馏损失函数定义$\mathcal{L}_d$如公式(2-7)。

$$\mathcal{L}_d^t = \sum_{(x,y)\in D^t \cup M} \sum_{j=1}^{N^{t-1}} -\phi_j^{t-1}(x) log[\phi_j^t(x)] \tag{2-7}$$

其中ϕ是旧模型作为老师模型提供的软标签(soft label)即旧模型在相同数据集上预测的分类结果，基于第 j 类数据的ϕ分类结果定义可参考公式(2-8)，其中T为知识蒸馏温度标量。

$$\phi_j^t(x) = \frac{e^{o_j^t(x)/T}}{\sum_{l=1}^{N^t} e^{o_l^t(x)/T}} \tag{2-8}$$

§2.4 增量学习在图像分类任务上的学习策略

本节主要介绍类增量学习任务的常用训练和测试策略。

§2.4.1 增量学习在图像分类任务上的训练策略

本节将从数据存储容量、新旧样本选择、阶段性任务设置以及损失函数四个方面介绍目前增量学习在图像分类任务上的训练策略。

(1) 数据存储容量

数据存储容量对应前文提到的 memory buffer，一般受系统或者本身数据库存储容量限定，在训练过程中对于 CIFAR100 的数据集一般设置容量为 2000，随着阶段性任务的进行，memory buffer 的总 size 不改变，但是存储的旧样本数据会随着各种替换算法进行更新。

(2) 新旧样本选择

新旧样本选择对应的问题主要是当 memory buffer 固定 size 的情况下，存储的旧样本数据势必和待训练的新样本数据之间有较大的量差异，比如 CIFAR100 数据集，共 100 累的数据，每类包含共 600 个数据样本，相较于当 memory buffer size 为 2000，采用静态存储方式每类存储 20 个样本时，20 旧样本对比 600 新样本有着较大的差异，会导致模型学习特征更多偏向于新样本。所以如何对新旧样本存储比例进行控制是其中一个问题。其二就是对于旧样本的选择，在上一节中有介绍到目前可以通过而对于样本选择可以基于与类内平均特征的相似性程度进行排序选择。一种经典的样本选择方式就是 iCaRL 模型中提出的基于 herding 优先样本选择策略(herding selection)，后文会详细介绍。

首先，herding selection 策略中对于每类样本存储数目同前文介绍的，假设存储空间大小为M，所有待续学习数据的类别总数为N，则对于每类样本设置静态存储数量即M/N，即规定每类样本存储数目，保证可用存储空间内内存被充分利用切不会超过。因为涉及到动态调整 memory buffer 内存储的数据，herding selection 策略主要涉及到三部分内容，样本建立，样本更新以及样本删减。样本建立指的是根据有限序列填入筛选好的旧样本，样本删减则是根据优先顺序剔除每类排名最后的旧样本，添加和删除的关键都在于如何选择排名优先的样本。具体选择方式是通过计算当前各类别类内平

均特征，并在类内进行各特征与平均特征的差，找到最接近平均特征的样本进行存储，具体公式如(2-9)和(2-10)所示。

$$\mu = \frac{1}{n}\sum_{x\in X}\varphi(x) \tag{2-9}$$

$$p_k = \underset{x\in X}{argmin}\left\|\mu - \frac{1}{k}[\varphi(x) + \sum_{j=1}^{k-1}\varphi(p_j)]\right\| \tag{2-10}$$

其中k代表类别数，φ代表特征提取方程，输入数据集为$X = \{x_1, x_2, ...x_n\}$，μ为类平均特征，p_k为遍历M/N待存储样本数返回的第k排序样本。

因为类增量学习模型还涉及到模型不断随新阶段任务的学习而更新，因此新旧模型对同一图像输出的特征也会随之改变，在每个阶段任务学习过后，之前存储的旧样本即原始图像也会再次经过新模型输出新图像特征并根据选择策略进行新一轮的优先顺序排序，对已存储的各类数据进行更新。

(3) 阶段性任务设置

虽然类增量学习任务大目标是处理持续的输入流数据进行图像分类任务，但是阶段任务数量不同影响到每个任务学习数据的类个数，如对于 CIFAR100 数据集，共 100 类数据，如果初始学习设定为学习 50 个类，后续共 5 个任务则后续的每个任务需学习$(100-50)/5 = 10$ 个类；若后续共 10 个任务则后续的每个任务需学习$(100-50)/10 = 5$ 个类；若后续共 25 个任务则后续的每个任务需学习$(100-50)/25 = 2$ 个类；若后续共 50 个任务则后续的每个任务需学习$(100-50)/50 = 1$ 个类。

在上述几种阶段任务训练设定过程中，通常任务越多即每阶段学习任务数更少，最终准确率会越低，因为随着任务数的增多对模型在稳定性和可塑性之间的平衡能力要求越高，越难进行灾难性遗忘的缓解。

(4) 损失函数

损失函数对应前文介绍的类增量模型组成构件中的损失函数模块，不同损失函数定义会影响到模型约束效果，除上文介绍的分类结果损失函数即交叉上损失函数以及知识蒸馏损失函数即针对新旧模型特征进行蒸馏的函数，还有其他 loss，也是众多模型寻求改进的板块。

§2.4.2 增量学习在图像分类任务上的测试策略

由于类增量学习的基本任务是图像分类，所以主要测试策略即测试模型在类别分类上的准确率。相比于传统分类任务，每批都学习所有数据集，测试范围也是针对所有数据类的，类增量学习有两种测试方法，第一种是测试当前阶段任务模型对于所有类的分类效果，第二种是只对于已学习的类进行分类效果测试。

§2.4.3 增量学习在图像分类任务上的评价指标

增量学习在图像分类任务上常用的评价指标有准确率以及遗忘率。对于输出的准确率，因为增量学习任务每阶段学习数据不同，故对新模型的测试还可细分类别，如在初始学习 50 类，之后共 5 个 tasks 每个 task 学习 10 个类的增量学习任务中，对第三个 task 对应模型的检测可以细分到 00-09，10-19 等部分类别上的准确率，即测试集选择对应上述细分类别的数据进行测试。具体准确率计算和以下四个评价指标有关：

(1) True Positive(TP)：标签和分类同时为正的样本数目

(2) False Negative(FN)：标签为正但是分类为负的样本数目

(3) False Positive(FP)：标签为负但是分类为正的样本数目

(4) True Negative(TN)：标签和分类同时为负的样本数目

准确率计算公式为$(TP+TN)/(TP+FN+FP+TN)$，代表分类结果正确的样本占全部样本的比例。通过对遗忘率进行计算，可以知道每阶段任务学习后对于先前所学知识的遗忘程度，动态评估模型效果的改变，关于任务t，遗忘率$\mathcal{F}$计算公式如(2-11)所示。

$$\mathcal{F}=\sum_{i=1}^{\cup_{j=1}^{t}C^{j}}(max(m,acc_i)-acc_{i-1})\Big/\cup_{j=1}^{t}C^{j} \tag{2-11}$$

其中acc_i为任务i后的准确率，$max(m,acc_i)-acc_{i-1}$对应连续任务之间准确率差值，$\cup_{j=1}^{t}C^{j}$代表截止任务i已学习的数据类总数。

§2.5 类增量学习的数据集

类增量学习所采用的数据集通常和图像分类领域公开数据集相似，主要有 CIFAR10, CIFAR100, ImageNet100, ImageNet1000, Celeb1000 等。考虑到增量学习任务的学习批次都很长，对于较大体量数据集训练时间会很长，对 GPU 要求也很高，因此本章节只对本文选用的 CIFAR 和 ImageNet 两个数据集进行介绍。

(1) CIFAR 数据集

CIFAR 100 数据集是 Tiny Images 数据集的一个子集，由 60000 张 32×32 彩色图像组成，共包括 100 个类别，其中每个类别共 600 张图像，再分为训练图像（每类 500 张）以及测试图像（每类 100 张）。对于 100 类数据，总共被分为 20 个超类，对应图 2-7 结构图，其中每张图像除了有所属类别即细节标签，还有对应超类即粗略标签。学术界和工业界常使用 CIFAR10 和 CIFAR100 两个带分类标签的小型图像数据集处理图像分类任务。Pytorch 或者 Tensorflow 中可以直接在线下载数据集，无需本地导入。

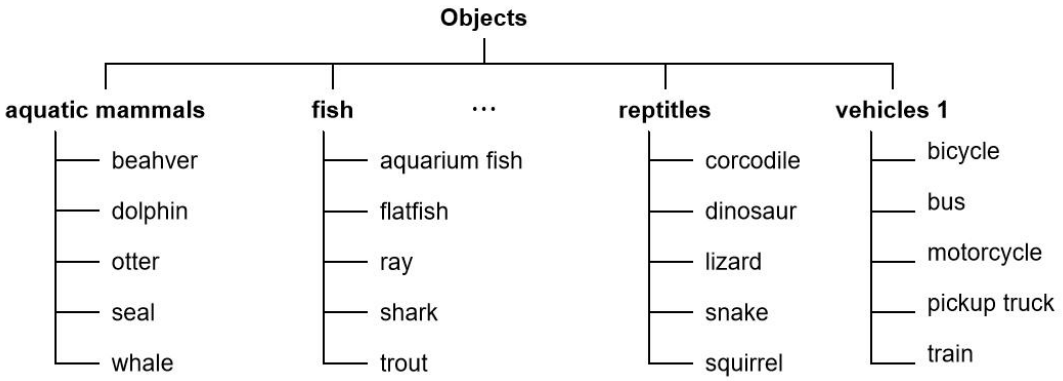

图 2-7 CIFAR100 数据集结构图

(2) ImageNet 数据集

ImageNet 数据集来自 2012 年 ImageNet 大规模视觉识别挑战赛，由三个部分组成：训练集、验证集和图像标签，训练数据包含 1000 个类别和 120 万张图像，部分数据集图像展示如图 2-8。由于 ImageNet1000 图像集太大，训练过程中对硬件要求较高，因此在一般任务中选择 ImageNet100 即 ImageNet1000 数据集的一个子集进行替代。ImageNet100 目前包含从 1000 类数据集中随机选择的 100 类数据。其中训练集每类包含 1300 张图像，验证集每类包含 50 张图像。相比 CIFAR10，ImageNet 数据集图片数量更多且包含更多类别，由于存储的都是高分辨率的 JPG 格式图像，因此随着分辨率提升图像中会夹杂更多无关特征和噪声，因此基于 ImageNet 数据集的图像领域任务难度会更高。由于 ImageNet 数据集普遍内存较大，因此训练过程中只支持手动导入，图像存储形式为 JPG。

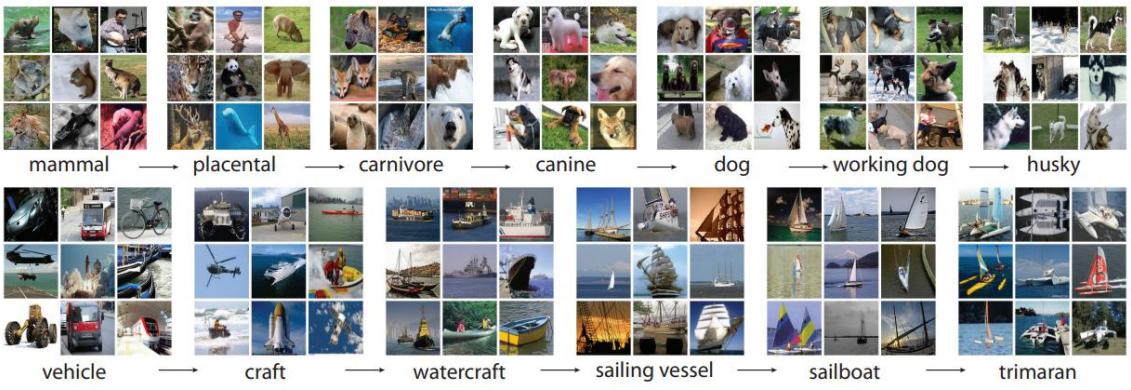

图 2-8 ImageNet 数据集展示

§2.6 本章小结

本章具体介绍了基于卷积神经网络的增量学习方法所含的理论知识，首先介绍了增量学习方法的基本框架以及卷积神经网络结构，其次针对类增量学习模型的组成进

行了公式化描述，并着重讲解了增量学习在图像分类上的学习策略，最后总结了类增量学习领域常用数据集。

第 3 章 基于卷积神经网络的增量学习方法的模型架构

本章详细介绍了融合回放策略和正则化策略的类增量学习方法池化知识蒸馏模型(PODNet)的基本思想以及相关改进方法，是全文重点章节。本章将从 PODNet 模型的理论基础以及模型架构开始，逐步介绍基于知识蒸馏、网络结构、样本存储以及图像压缩四方面的改进思路，并给出选择该改进方式的理由。

§3.1 PODNet 模型

2020 年 Douillard 等人提出基于回放以及知识蒸馏策略的 Pooled Outputs Distillation (PODNet)[15]模型完成图像分类任务，同时引入池化操作改进知识蒸馏形式，并设置了多表征向量改进模型分类器，性能达到了当时的 SOTA，特别是在小样本任务增量学习上取得较好的表现，不仅证实了现有增量学习策略叠加使用的有效性，同时推动了基于知识蒸馏策略改进以及小样本增量学习任务的研究。

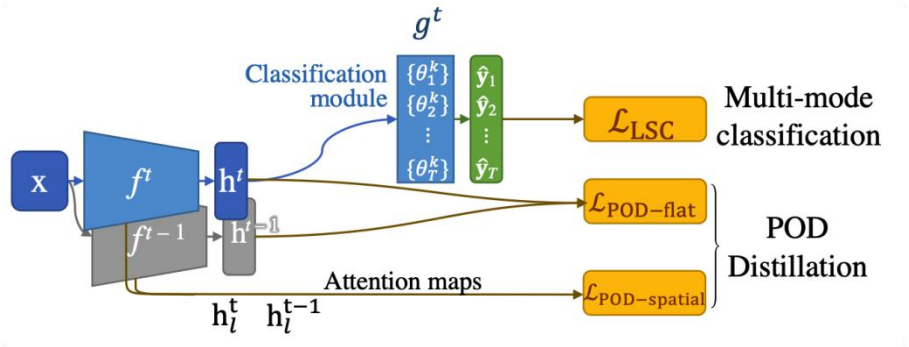

图 3-1 PODNet 模型图

PODNet 模型的整体框架如图 3-1 所示[15]，模型采用深度卷积神经网络作为主要结构，模型由特征提取器、分类器以及损失函数三部分构成。训练过程采用 CIFAR100 作为数据集，使用 ResNet-32 作为骨干网络(Backbone)。

PODNet 模型是针对小样本增量学习任务建立的，小样本是指增量学习任务数多，而每次学习类别数少的情况。一共包含T个任务，模型在第一个任务中学习来自 50 个不同分类的训练样本，在之后的t个任务中，每次学习 $50/t$个任务并不断更新网络参数使得模型在学习新分类图像时仍能维持对旧任务图像的分类能力。对于第t个任务的学习共包含C_N^t个新类的训练样本以及C_O^t个旧类的训练样本数据，其中$C_O^t = C_N^{t-1} \cup C_O^{t-1}$，对于$C_O^t$个旧类的训练样本数据，每个旧类限制存储M个样本，对于旧样本存储的选择以及 memory buffer 更新策略，参考了前文介绍的 iCaRL 模型提出的 herding selection 方法。

PODNet 模型整体基于卷积神经网络$\hat{y} = g\big(f(x)\big)$进行设计，其中$x$为输入图像，$y$代表关于图像所属分类的可能性向量输出，$h = f(x)$代表神经网络特征提取器部分，$\hat{y} = g(h)$代表神经网络最后的分类层，$h$输出分类层前代表图像特征的最后 embedding。

损失函数是每个增量学习方法最重要的板块，PODNet 模型的改进主要集中在知识蒸馏以及表征学习两部分。

在知识蒸馏部分，传统基于正则化的增量学习方法可以利用知识蒸馏的概念计算新旧模型之间的二元交叉熵，通过对神经网络中的重要性参数进行控制使模型能够应对灾难性遗忘问题。PODNet 基于以上思想通过计算蒸馏损失(Distillation Loss)进行模型修改，代替原先直接对新旧模型每层输出特征进行差异计算，但容易导致训练后新模型太像旧模型，网络过于极端，从而难以学习新知识的方法。不直接使用新旧模型差异进行训练，而是对旧模型输出特征进行池化操作，利用池化后特征和新模型进行特征差异计算，可提高新模型的可塑性，使得新模型性能取得折中效果。

相比于一般基于正则化的增量学习方法只针对模型输出图像特征的最后 embedding，即$h^t = f^t(x)$进行计算。PODNet 考虑了中间层的图像特征 embedding 即$h_l^t = f_l^t(x), f_l^t(x) = f_1^t \circ f_2^t \circ \ldots \circ f_l^t$，因此最后分类层前的图像特征可表示为$h^t = f^t(x) = f_1^t(x) \circ f_2^t(x) \circ \ldots \circ f_L^t(x)$，其中$L$为神经网络层数。

对于神经网络卷积层输出h_l^t，除了包含层信息，还包括通道(Channel)以及空间坐标宽度(Width)和高度(Height)信息。池化操作在计算机视觉中是很重要的操作，通过池化操作可以根据图像不同区域进行聚合，使得池化层的图像维度降低，同时由于池化层只涉及平面上的操作，故输入和输出的深度是一样的。利用池化层的思想，将池化后的参数融入蒸馏损失的计算，可以使模型更新具有不变性的同时，损失函数的计算过程更为简洁。通过引用池化操作可以使得模型具有更高灵活性，即对于重要性参数的限制有所减少，更加适应新任务。

在知识蒸馏损失函数的计算中，一般未使用池化操作的$\mathcal{L}_{origin}$计算方式如公式(3-1)，其中$h_{l,c,w,h}^{t-1}$为对应$t-1$任务的第l层神经网络卷积层输出，$h_{l,c,w,h}^{t}$为对应t任务的第l层神经网络卷积层输出，下标l, c, w, h分别对应神经网络层数、通道、宽度以及高度四个参数。

$$\mathcal{L}_{origin}(h_l^{t-1}, h_l^t) = \sum_{c=1}^{C} \sum_{w=1}^{W} \sum_{h=1}^{H} \left\| h_{l,c,w,h}^{t-1} - h_{l,c,w,h}^{t} \right\|^2 \tag{3-1}$$

而 PODNet 中所有基于池化的蒸馏损失计算，首先采用 L2 正则化计算欧氏距离，通过对空间维度即宽度和高度分别进行池化操作并相加，代替未使用池化操作进行知识蒸馏损失函数的计算，使模型具有更多的可塑性。融合池化操作的空间$\mathcal{L}_{POD-spatial}$计算公式如(3-2)所示，以下公式将由宽度损失$\mathcal{L}_{POD-width}$和高度损失$\mathcal{L}_{POD-height}$共同组成，分别对应公式(3-3)和公式(3-4)。

$$\mathcal{L}_{POD-spatial}(h_l^{t-1}, h_l^t) = \mathcal{L}_{POD-width}(h_l^{t-1}, h_l^t) + \mathcal{L}_{POD-height}(h_l^{t-1}, h_l^t) \quad (3\text{-}2)$$

$$\mathcal{L}_{POD-width}(h_l^{t-1}, h_l^t) = \sum_{c=1}^{C}\sum_{h=1}^{H}\left\|\sum_{w=1}^{W} h_{l,c,w,h}^{t-1} - \sum_{w=1}^{W} h_{l,c,w,h}^{t}\right\|^2 \quad (3\text{-}3)$$

$$\mathcal{L}_{POD-height}(h_l^{t-1}, h_l^t) = \sum_{c=1}^{C}\sum_{w=1}^{W}\left\|\sum_{h=1}^{H} h_{l,c,w,h}^{t-1} - \sum_{h=1}^{H} h_{l,c,w,h}^{t}\right\|^2 \quad (3\text{-}4)$$

在上述基于空间维度进行知识蒸馏损失函数计算的基础上，对于最后一层神经网络 embedding 即$h^t = f^t(x)$设置平面约束$\mathcal{L}_{POD-flat}$，计算过程见公式(3-5)。

$$\mathcal{L}_{POD-flat}(h_l^{t-1}, h_l^t) = \left\|h^{t-1} - h^t\right\|^2 \quad (3\text{-}5)$$

最终，PODNet 知识蒸馏部分损失函数$\mathcal{L}_{POD}$的计算公式由空间维度和平面约束两部分组成，公式如(3-6)所示。

$$\mathcal{L}_{POD}(x) = \frac{\lambda_c}{L-1}\sum_{l=1}^{L}\mathcal{L}_{POD-spatial}(f_l^{t-1}(x), f_l^t(x)) + \lambda_f\mathcal{L}_{POD-flat}(f^{t-1}(x), f^t(x)) \quad (3\text{-}6)$$

其中λ_c, λ_f是用于衡量空间维度和平面约束两部分的参数，取值在 0-1 之间。当$\lambda_c = 0$时，相当于只对神经网络的最终 embedding 进行约束。

在表征学习部分，一般基于卷积神经网络的增量学习方法，为了消除新旧任务样本不均衡导致的模型预测结果更偏向于新任务的问题，会选择余弦相似性分类器对神经网络最终 embedding 进行分类预测的计算。而基于余弦相似性进行设计的分类器对于单一输入$h^t = f^t(x)$的变化很敏感，因此 PODNet 采用多个表征向量代替单一表征向量$h^t = f^t(x)$进行分类器置信度的计算，以K个余弦分类器分类结果作为最终分类结果，计算公式为(3-7)。

$$s_{c,k} = \frac{exp\langle\theta_{c,k}, h\rangle}{\sum_i exp\langle\theta_{c,i}, h\rangle} \quad (3\text{-}7)$$

其中，$s_{c,k}$为对于c类的K个表征向量的相似性计算，$\langle\cdot,\cdot\rangle$为余弦相似性计算，$\widehat{y_c}$为平均相似性，计算每个类的最终分类器输出，具体过程如(3-8)所示。引入多表征向量可以使每个样本的预测结果更接近真实类别。

$$\widehat{y_c} = \sum_k s_{c,k}\langle\theta_{c,k}, h\rangle \quad (3\text{-}8)$$

其中$\theta_{c,k}$为类别c的第k个权重，对于c类所有表征向量的权重$\{\theta_{c,k}|\forall c \in C_N^t, \forall k \in 1,\ldots,K\}$初始值定义如下，对于$c$类别的训练样本，首先进行特征提取，其次利用 K-Means 算法对所获取到的各样本特征进行总共K聚类划分，并使用每个聚类的中心簇的值作为$\theta_{c,k}$初始值，其中K为表征向量个数。最终表征学习部分损失函数即改进分类损失函数的公式为(3-9)。其中η是可学习参数，δ是增强类间间隔的小数值。

$$\mathcal{L}_{LSC} = \left[-log\frac{exp(\eta(\widehat{y_y} - \delta))}{\sum_{i\neq y} exp(\eta\widehat{y_i})}\right]_+ \quad (3\text{-}9)$$

PODNet 总体模型是基于经典卷积神经网络的结构$\hat{y} = g\big(f(x)\big)$进行设计的，$f(x)$和$g(x)$分别作为特征提取器以及分类器，在此基础上提出融入空间维度和平面约束的

全新知识蒸馏损失计算方法，同时为每个类保留了多个表征向量，增加模型的表现力，使分类器对于$h^t = f^t(x)$的变化不那么敏感。

最终对于任务t，模型$g^t \circ f^t$损失函数由上述两部分组成，计算公式如(3-10)所示。

$$\mathcal{L}_{\{f^t;g^t\}} = \mathcal{L}_{POD} + \mathcal{L}_{LSC} \tag{3-10}$$

§3.2 改进的 PODNet 模型

本节主要介绍基于 PODNet 的四种改进方式。

§3.2.1 基于知识蒸馏的 PODNet 改进

原先 PODNet 模型基于知识蒸馏进行约束是针对神经网络新旧模型的图像特征即最后分类层前的 embedding 进行差异约束，使新旧模型在图像特征层面更接近。

PODNet 模型即这一类增量学习方法针对的都是图像分类问题，而在图像分类领域模型一般分为特征提取器以及分类器两部分，使用 Softmax 作为神经网络的最后一个激活单元是较为典型的处理方式，通过接受特征提取器生成的 logits 作为输入并输出离散类别上的概率分布，使用 Softmax 信息比起独热编码(one-hot)会更具有代表性，因为能直观表现出目标所属类别。

由于传统知识蒸馏方式都是先在一个数据集上训练出具有较好实验效果的教师模型，再利用教师模型输出作为参考，在同一数据集上指导训练出小体量的学生模型，具体参考方式为利用教师模型的 Softmax 输出预测作为真实标签(Ground-Truth)，即软标签(Soft label)训练学生模型，使得学生模型模仿教师模型的输出。为了使得学生模型捕捉到更多有用信息，设置温度(temperature)将教师模型的原始 logits 进行一定程度缩放，使得可用类标签分布范围更广泛，并依次和学生模型的预测进行误差损失函数的计算，训练方式如图 3-2 所示。

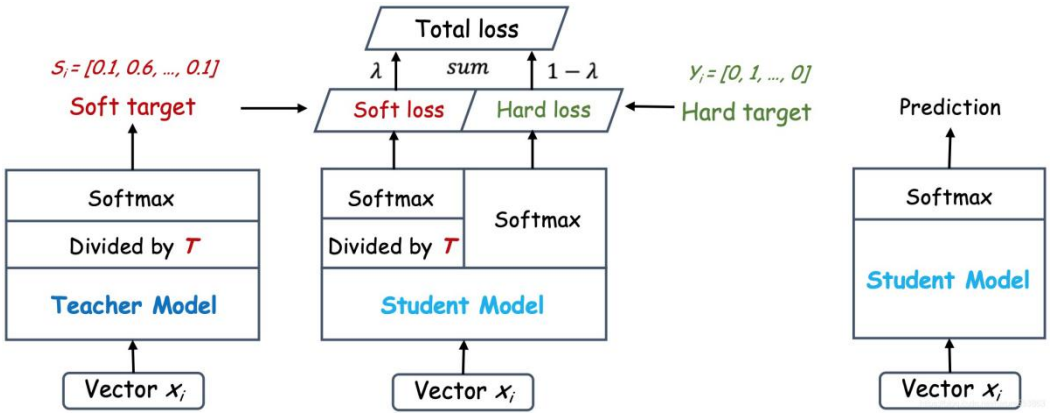

图 3-2 知识蒸馏训练方式

受上述传统知识蒸馏方式的影响，对 PODNet 模型基于图像特征进行知识蒸馏的方法进行改进，选择新旧模型最终分类结果进行约束。

基于知识蒸馏的 PODNet 改进的详细实验数据和分析将在本文§4.3.2 给出。

§3.2.2 基于网络结构的 PODNet 改进

2018 年 Woo 等人提出轻量级卷积块注意力模块(CBAM)[17]，能在固定了神经网络中间层特征图的情况下，先后沿通道维度和空间维度生成注意力映射，从而实现对注意力特征图和输入特征的相乘操作，并借此达到自适应优化的目的，通过注意力机制的使用提高表征能力，更有效处理网络中的信息流。

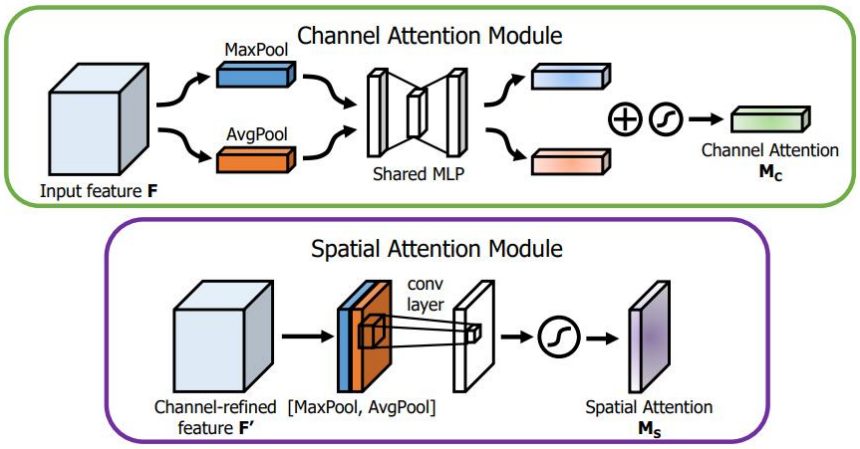

图 3-3 CBAM 模块子模块结构

CBAM 关键在于其包括了两个顺序子模块，如图 3-3 所示[17]。通道注意模块(CAM)和空间注意模块(SAM)，给定维度为$C \times H \times W$的中间层特征图(feature map) F作为输入，依次生成维度为$C \times 1 \times 1$ 的通道注意力映射M_c和维度为 $1 \times H \times W$的空间注意力映射M_s，对应计算公式如(3-11)和(3-12)所示。

$$M_c(F) = \sigma\left(MLP\left(AvgPool(F)\right) + MLP\left(MaxPool(F)\right)\right) \tag{3-11}$$

$$M_c(F) = \sigma(W_1\left(W_0\left(F^c_{avg}\right)\right) + W_1\left(W_0(F^c_{max})\right)) \tag{3-12}$$

其中F^c_{avg}和F^c_{max}分别表示平均集合特征和最大特征集合，σ代表 sigmoid 函数，W_1, W_0为共享权重，计算得到的$M_c(F)$将和原始特征F进行逐元素相乘运算(element-wise)，对得到结果即$F' = M_c(F) \otimes F$进行空间注意力映射计算，计算公式可参考(3-13)和(3-14)。

$$M_s(F) = \sigma\left(f^{7\times7}([AvgPool(F); MaxPool(F)])\right) \tag{3-13}$$

$$M_s(F) = \sigma(f^{7\times7}\left(\left[F^s_{avg}; F^s_{max}\right]\right)) \tag{3-14}$$

CBAM 框架来源于 2016 年图像描述生成(Image Caption)领域的研究，在该领域图像特征提取方面得到较好改进，融入注意力机制的生成特征以便描述生成模块生成更贴近符合原始图像的描述。因此基于注意力机制能够带来的效果改进，本改进方式希望在 ResNet 模型结构中引入 CBAM 模块使得在图像分类任务特征提取器模块本身提取效果得到改进，再结合增量学习方法适应持续输入数据。

由于空间注意力模块和通道注意力模块的不同排列组合方式会导致模型产生效应有差异，因此基于[17]研究结果，本文选择先通道注意力后空间注意力模块的结合方法进行改进研究。相关详细实验数据和分析在本文§4.3.3 给出。

§3.2.3 基于样本存储的 PODNet 改进

PODNet 模型中采用固定了 memory buffer 尺寸以及每类存储样本数量，由于每类存储样本的数量在小样本增量学习中对模型最终准确率影响比较大，因此对存储数量的动态调整可以有助于提升分类准确率。Liu 等人提出的 RMM[21]模型通过增量学习的方式动态学习新旧样本存储比例以及不同旧类样本存储比例，每阶段对存储样本数都进行动态调整。Yan 等人提出的 DER[22]模型利用基于微分信道级掩码的方法对特征提取器滤波器(filter)进行剪枝操作，对通道增加掩码，减少 feature 尺寸进而降低存储和运算消耗。

上述基于回放的策略基本围绕缓存容量、存储数据的选择、数据的存储形式、新旧数据混合等方面进行改进，考虑到模型改进难度以及效果，本节采用的方法为扩展存储容量以及动态调节每阶段任务各类存储样本数量，以提升分类准确率。训练过程中，设计 memory list 根据不同增量学习任务(5 tasks, 10 tasks, 25 tasks, 50 tasks)进行记忆存储率的分配，具体 list 设置详见§3.2.3 实验数据板块。

相比原先 PODNet 采用固定 2000 的 size 进行存储，每类存储 20 类样本，该改进方法动态扩展了样本总量，若对于任务t，此时记忆存储率为α，则对存储容量M进行如下修正$M = M + \alpha * increment * img_pre_cls$，其中$increment$为每阶段任务所需学习数据类别总量，$img_pre_cls$对应数据集内每类样本数。由于$M$受$increment$限制，因此当处理小任务增量学习任务(50 tasks)时，对于存储总量的需求并不会扩展太多。

同时，由于从第一阶段任务存储容量就已改为上述扩展后的M，因此相比 PODNet 任务在第 0 阶段学习 50 类任务时只使用了一半(1000/2000)存储容量相比，可以从训练初始阶段就充分利用存储空间，尽可能存储更多旧数据样本。相关实验数据和分析在本文§4.3.4 给出。

§3.2.4 基于图像压缩的 PODNet 改进

上一节中基于样本存储的改进方式是利用回放策略的一些重要影响因素进行的调整，每类数据的旧样本存储数量受总存储容量以及数据类总数限制。为了最大程度在有限空间存储更多样本，因此本节考虑从图像本身入手进行适当处理，降低旧样本的存储成本，从而增加其在 memory buffer 中的存储量。

Zhao 等人提出的高效率内存类增量学习 MECIL[18]模型使用数据压缩对原始样本进行处理，构建原始样本-伪样本-真实标签的三元组进行指导训练，对于伪样本即压缩

样本的生成采用 PCA 等降维方式，相当于在整个 CNN 模型（特征提取器+分类器）前对图像做了 encoder 和 decoder 处理，同时每阶段训练过程中不断更新 decoder 解码过程，尽可能还原具有较高保真度的伪样本，具体模型训练过程如图 3-4 所示[18]。

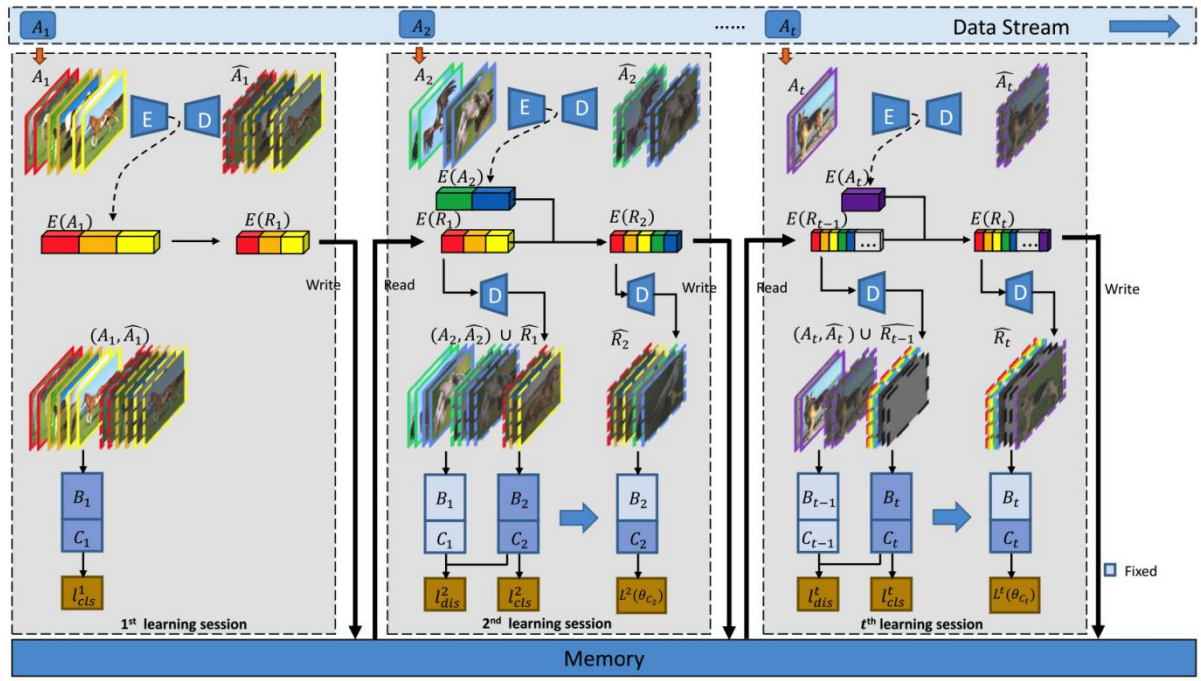

图 3-4 MECIL 模型训练过程

MECIL 采用 encoder-decoder 模型对数据进行压缩的方式可以在同样 memory buffer 下存储更多样本，但是同时也提升了训练成本，包括构造伪数据集三元组以及原始样本-伪样本相似性等多方面训练，因此本节借鉴其对原始样本做处理的思想，直接对数据集进行压缩处理，使用压缩后数据进行后续类增量学习。

有关图像压缩过程可以定义为$F_q^c(\cdot)$函数，对于原始图像$x_{t,i}$得到经过压缩处理后的压缩图像$x_{q,t,i} = F_q^c(x_{t,i})$，其中$q$为压缩率。因为压缩后图像的尺寸小于原始图像尺寸，因此同样 memory buffer 下存储压缩图像的数量$N_{q,t}^{mb}$大于存储原始图像的数量N_t^{mb}，任务t对应数据集定义为$D_{q,t}^{mb}$，具体公式见(3-15)。

$$D_{q,t}^{mb} = \{(x_{q,t,i}, y_{t,i})\}_{i=1}^{N_{q,t}^{mb}} \tag{3-15}$$

图像压缩率的选择影响着压缩后图像对原始图像特征的保留程度，因此合适的压缩率q尤为重要，本文参考 Wang 等人提出的 MRDC 模型[19]对图像压缩率和质量的对比实验结果，选择 75%作为压缩率，在保证图像特征的前提下尽可能压缩图像尺寸，对于图像压缩率和质量的相互影响的可视化过程可参考图 3-5。相关详细实验数据和分析在本文§4.3.5 给出。

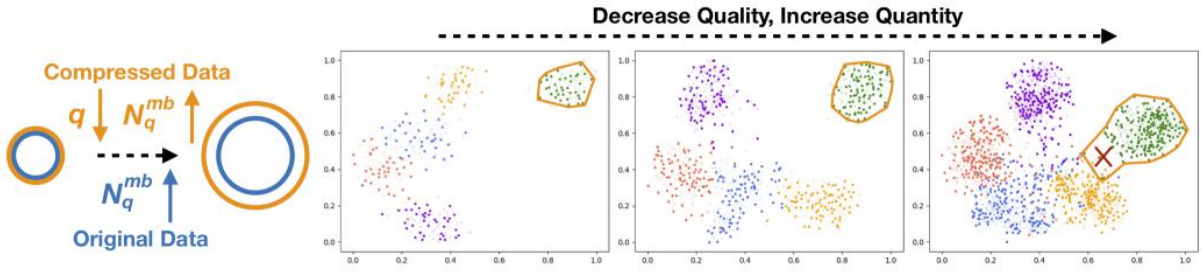

图 3-5 图像压缩 quality 和 quantity 的关系

§3.3 本章小结

本章详细介绍了本文使用的 PODNet 模型结构、核心损失函数以及所使用的增量学习策略，然后介绍了对 PODNet 四个方面的改进思路。一是针对知识蒸馏对象进行调整，二是融入注意力机制模块增强分类模型对图像特征的掌握，三是动态扩充样本存储空间以解决新旧样本不平衡问题，四是利用图像压缩策略对原始图像进行压缩以扩充每类存储样本数量。

第 4 章 实验结果与分析

本章主要阐述复现 PODNet 模型所依赖的开发环境、训练细节以及学习策略，并依次对应第三章所提到的各改进方法进行对比实验、消融实验以及融合实验。

§4.1 开发环境

本文模型使用的语言为 Python，采用的深度学习框架为 Pytorch，使用数据集为 CIFAR100 以及 ImageNet100。训练过程在矩池云、AutoDL 两个 GPU 租用平台上完成。

§4.1.1 开发语言

本文基于 Python 和深度学习框架 Pytorch 进行开发，使用 Python 3.8、Pytorch 1.5.1 的配置进行 PODNet 实验复现以及系列改进任务，通过 Visual Code 远程连接服务器进行训练。

§4.1.2 开发平台

本文使用的开发平台主要为如下两个：

(1) 矩池云

国内专注于人工智能领域的云服务商，是一个深度学习云平台，免费提供 5GB 的网盘存储空间以及 2 小时 GPU 服务。租赁服务器是按小时计算的，可在主机市场根据不同需求以及环境配置自行选择，矩池云运行程序以及输出日志都是存储在矩池云网盘上的，每次通过 Visual Code 的远程连接，进入网盘存储路径调用程序。对于租用机器可以选择保存环境再释放，在下次再次进行租用时可以直接拉取已存储环境直接进行训练。通常我租用的是 NVIDIA Tesla K80、NVIDIA GeForce RTX 2080 两种显卡，环境选择 Python3.8、CUDA 10.2、Pytorch 1.5.0，内存约 12GB。虽然矩池云使用界面比较清晰明了、远程连接以及数据存储下载等较为方便，但是由于高性能 GPU 单价较贵，网盘免费空间比较小需额外扩容，因此只在前期使用了矩池云进行训练，后续转至另一开发平台进行后续实验。

(2) AutoDL

为用户提供稳定可靠的 GPU 算力平台，学生认证后可以享受会员价进行 GPU 租用，租赁服务器同样按小时计费，提供多种型号 GPU 进行租用，提供免费 20GB 网盘容量存储程序以及运行结果，连接同样通过 Visual Code 进行远程连接。对比矩池云，可对已关机实例重新进行开机，继续训练，无需再配环境。训练过程中主要使用 NVIDIA RTX 2080 Ti 显卡进行模型训练，环境选择 Python 3.8、Cuda 10.1、PyTorch 1.5.1，内

存约 16GB。后续消融实验、对比实验等需较长训练时间的程序多挂载在该平台上进行训练。

§4.2 模型的实现细节

本文实验的模型代码支持 Cifar 以及 ImageNet 数据集，本文多在 CIFAR100 数据集上进行实验，数据集共包含 100 类图像，选用 train 作为训练集，每类包含训练样本数为 500，选用 val 作为测试集，每类包含测试样本数为 100。骨干网络选择基于 ResNet 网络结构的 ResNet-32，不冻结模型参数。

§4.2.1 训练细节

训练过程主要分为预先数据处理操作以及模型超参数设置两部分。

根据不同数据集采用不同预处理方式，针对 CIFAR100 数据集，将图像转为 Tensor 并进行归一化处理；对于 ImageNet100 数据集，随机裁剪图像并最终缩放到同一大小 224×224，以 0.5 概率对给定图像叠加随机水平旋转变换，并对图像亮度进行调整。

有关模型超参数的设置，根据不同增量学习任务，训练步长进行相应调整，5 steps 对应每阶段增量学习 10 类数据，10 steps 对应每阶段增量学习 5 类数据，25 steps 对应每阶段增量学习 2 类数据，50 steps 对应每阶段增量学习 1 类数据。设置固定随机种子[3,3,3]进行训练，训练周期(epochs)默认为 160 个，每周期内批次(batch size)为 128，使用随机梯度下降算法(SGD)作为优化器，相关超参数设置如下：初始学习率(learning rate)设为 0.1，动量(momentum)设为 0.9，权重衰减(weight decay)设为 0.0005。在每次训练过程中，除了 160 个 epochs 的训练，在增量学习数据的阶段中设置 20 个 epochs 的微调。训练分类器分别选择 Softmax 分类算法和最近均值分类算法(NME)输出图像预测分类，并进行对比实验。

§4.2.2 测试细节

对于 CIFAR100 数据集，测试集为转化为 Tensor 并进行归一化处理的 val；对于 ImageNet100 数据集，测试集等比例缩放到最小边长为 256 的大小，经中心裁剪，最终图像大小为 224×224。

§4.3 实验结果分析与对比

§4.3.1 复现 PODNet 模型

本文对于 PODNet 的复现所使用的是基于 ResNet 结构的 ResNet-32 预训练模型进行实验，为了符合原文的训练细节，有关 SGD 优化器以及分类器等模型超参数的选择同原文一样，为了更准确的复现模型结果，对于每个实验设计进行了三次实验并最终取平均准确率值以及正负偏差作为最终输出结果。

由于 ImageNet100 数据集较大，复现起来 5 steps 的实验都要跑上 25 小时以上，因此在复现阶段都采用 CIFAR100 数据集进行实验，实验共根据每类增量学习数据类别数不同分为四种训练阶段(5, 10, 25, 50 steps)，每个 steps 训练分别基于 CNN 分类器和 NME 分类器进行实验。具体复现结果以及和原实验结果对比见表 4.1，其中带*号的为原文实验结果。

表 4.1 PODNet 复现实验结果对比表

	CIFAR100			
	50steps	25steps	10steps	5steps
New classes per step	1	2	5	10
UCIR(NME)*	-	-	60.83±0.70	63.63±0.87
UCIR(NME)	**44.02±7.63**	**50.07±7.82**	**55.77±5.34**	**59.74±4.73**
PODNet(CNN)*	57.98±0.46	60.72±1.36	63.19±1.16	64.83±0.98
PODNet(CNN)	**58.01±0.56**	**60.65±1.12**	**62.46±0.01**	**64.56±1.03**
PODNet(NME)*	61.40±0.68	62.71±1.26	64.03±1.30	64.48±1.32
PODNet(NME)	**61.43±0.95**	**63.50±1.00**	**64.72±1.04**	**65.69±1.10**

上表前两行数据为基于 UCIR 模型进行的结果复现，具体会在§4.3.5 再有所涉及；对于 PODNet 的复现，加粗数据为本文复现结果，基本符合原文实验结果，准确率差异在 1%之内。对于 PODNet 的 8 个复现实验在内存为 12GB 的 NVDIA Tesla K80 显卡上进行，训练 batch size 为 128，随着 steps 的增加训练时间会随之增长，对于 5 steps 的实验基本在 6-7 小时左右完成，对于 50 steps 的实验基本耗时 30 小时左右。

通过上述表格的实验结果对比，以复现后的两组数据 PODNet(CNN)以及 PODNet(NME)为例，具体增量学习每阶段准确率如表 4.2 所示。

表 4.2 PODNet 各阶段准确率表

CIFAR100	Number of classes (10 steps)											Number of classes (5 steps)					
Method	50	55	60	65	70	75	80	85	90	95	100	50	60	70	80	90	100
PODNet (CNN)	0.78	0.75	0.74	0.71	0.69	0.68	0.67	0.65	0.64	0.63	**0.62**	0.78	0.75	0.72	0.69	0.67	**0.65**
PODNet(NME)	0.77	0.75	0.74	0.72	0.71	0.69	0.68	0.67	0.66	0.65	**0.64**	0.81	0.77	0.74	0.71	0.69	**0.67**

上表各数值代表的是当学习到第 50/60/70/80/90/100 类数据时，训练模型对于当前阶段所学习的所有类别数据进行测试时输出的准确率值。可以发现模型分类准确率随着学习类别数的增多而降低；对于 CNN 和 NME 两个分类器，通过上述结果也可以证实最近均值分类算法计算得出的新旧数据预测值更接近 ground truth，在类增量学习任务中表现较好。

表 4.3 各阶段学习类别准确率表

CIFAR100	Accuracy on each classes (5 steps)										
	00-09	10-19	20-29	30-39	40-49	50-59	60-69	70-79	80-89	90-99	**Avg**
PODNet(CNN)	0.61	0.54	0.52	0.56	0.52	0.44	0.45	0.57	0.56	0.71	**0.55**
PODNet(NME)	0.77	0.67	0.70	0.71	0.68	0.43	0.45	0.43	0.39	0.35	**0.56**

CIFAR100	Accuracy on each classes (10 steps)										
	00-09	10-19	20-29	30-39	40-49	50-59	60-69	70-79	80-89	90-99	**Avg**
PODNet(CNN)	0.58	0.51	0.53	0.57	0.50	0.42	0.41	0.50	0.58	0.66	**0.53**
PODNet(NME)	0.75	0.66	0.67	0.69	0.67	0.42	0.44	0.42	0.44	0.35	**0.55**

表 4.3 为训练结束时，在每 10 类为一组的数据中的准确率，可以看到由于本文实验设置的 initial classes 都是 50，即在第 0 阶段先学习数据集的一半类别数据，因此前 50 类的准确率之间都比较接近；最新学习的类别准确率会比较高（对应 5 steps 和 10 steps 在 90-99 类别上的准确率）；处于第 0 阶段任务到最新任务之间的数据类别准确率相较首尾两端学习任务的准确率相对较低，对比 5 steps 和 10 steps 的数据，可以发现对于数据集分割越细，即小任务增量学习的分类准确率相比其他增量学习分类任务会有较明显的下降，随着增量学习任务总数的增加，虽然每次只学习很少的类，但是模型遗忘能力也更加严重，属于类增量学习领域更难处理的问题。

§4.3.2 基于知识蒸馏的改进 PODNet 实验结果

本章节将 PODNet 池化蒸馏板块对新旧网络 feature 进行正则化操作替换成对新旧网络 softmax 结果进行正则化操作后的实验结果。由于知识蒸馏方式进行约束受温度和系数两个参数的影响，因此首先选择 NME 作为分类器进行 5 steps 实验确定不同参数的情况下哪一组参数能够达到最高预测准确率（选择 5 steps 做实验是因为实验耗时相较于其他 steps 会少很多，可以加快确定合适参数的时间）。

由于原文 PODNet 在知识蒸馏相关计算部分包含了两部分，层间池化蒸馏以及最后一层 embedding 蒸馏，此处替换了最后一层 embedding 蒸馏，保留层间池化蒸馏部分。同时，为了更好的了解改进后的效果，因此在本模块除了做替换后的层间+softmax 知识蒸馏，还做了层间+embedding+softmax 蒸馏实验，来判断是单一 softmax 蒸馏效果好还是两者结合效果好，表 4.4 为基于 NME 分类器的 5 steps 任务下消融实验数据，根据实验结果得到表现效果最好的一组参数。

表 4.4 基于知识蒸馏的 PODNet 消融试验数据表

PODNet(NME)	T=2				T=10	
Number	1	2	3	4	1	2
KD-embedding	0.5	-	0.5*factor	-	0.5*factor	-
KD-softmax	0.5	factor	0.5*factor	0.9	0.5*factor	0.9
Accuracy	65.57±1.2	65.71±1.13	66.71±1.21	66.92±1.13	66.83±1.05	**67.03±1.1**

上表中 factor 为适应性权重系数$\lambda = \lambda_{base}\sqrt{|C_n|/|C_0|}$，$C_n, C_o$分别代表阶段任务学习的新类总数以及已学习的旧类总数，λ_{base}为基于不同数据集的固定参数。通过消融实验，根据分类准确率，确定选择只有层间+softmax 的约束方式，其中 softmax 部分知识蒸馏的参数选择为 temperature 为 10，参数为 0.9。因此基于以上选择在 CNN 和 NME 分类器上进行 5/10/25/50 steps 的实验，与原 PODNet 实验结果进行对比，实验结果如表 4.5 所示。

表 4.5 基于知识蒸馏的改进 PODNet 实验数据表

	CIFAR100			
	50steps	25steps	10steps	5steps
New classes per step	1	2	5	10
PODNet(CNN)	58.01±0.56	60.65±1.12	62.46±0.01	64.56±1.03
PODNet(CNN)-KD	**58.34±0.8**	**61.56±0.79**	**63.54±0.93**	**65.26±0.44**
PODNet(NME)	61.43±0.95	63.50±1.00	64.72±1.04	65.69±1.10
PODNet(NME)-KD	**62.69±1.28**	**64.68±1.02**	**66.3±1.29**	**67.03±1.1**

第二行和第四行数据为基于知识蒸馏的改进结果，与原始结果相比取得了 0.3%-1.5%左右的提升，基于 NME 分类器的效果提升较基于 CNN 分类器的提升更明显。上述结果说明选择对 softmax 结果进行知识蒸馏在类增量学习任务中会比直接基于网络特征进行约束取得更好的效果，但是总体提升幅度不是特别大，这可能是因为本身损失函数已经包括了层间池化蒸馏部分以及其他 loss，所以部分改动不会引起全局较大变化；其次可能是随着阶段任务模型不断更新，但是 embedding 包含信息和 softmax 包含信息只隔着最后一个分类层，所以这两部分包含的信息不会有特别明显的差异。

该阶段实验是在 AutoDL 平台上租用 NVIDIA RTX 2080 Ti 显卡进行训练的，选用数据集为 CIFAR100，四组 steps 的实验耗时分别为 6 小时、11 小时、24 小时、30 小时左右。

总的来说，本章节通过对知识蒸馏正则化策略部分板块的改动，可以证明新旧模型直接分类预测结果之间约束比特征约束有更好的效果，可考虑结合其他改进策略共同提升。

§4.3.3 基于网络结构的改进 PODNet 实验结果

本章节在 PODNet 模型特征提取以及图像分类板块融入注意力机制模块进行改进，主要是在 ResNet-32 网络中引入通道注意力模块和空间注意力模块，并修改块内组合，从网络结构入手增强对图像特征的获取。改进后实验结果如表 4.6 所示，同样在 5/10/25/50 steps 以及 CNN 和 NME 分类器上分别进行了实验。

表 4.6 基于网络结构的改进 PODNet 实验数据表

	CIFAR100			
	50steps	25steps	10steps	5steps
New classes per step	1	2	5	10
PODNet(CNN)	58.01±0.56	60.65±1.12	62.46±0.01	64.56±1.03
PODNet(CNN)-CBAM	**60.18±2.56**	**61.78±0.82**	**64.47±1.05**	**66.17±0.97**
PODNet(NME)	61.43±0.95	63.50±1.00	64.72±1.04	65.69±1.10
PODNet(NME)-CBAM	**59.83±0.74**	**63.89±1.12**	**65.62±1.42**	**66.20±1.2**

通过上述实验数据可以看到，引入注意力机制模块后整体准确率除了 50 steps 且使用 NME 分类器时没有得到提升外，都取得了 0.3%-2%之间的提升，且上升幅度随着 steps 减少而提升，对于两种分类器上各自的结果，可以看到引入 CBAM 后采用两种分类器进行预测的差距也逐渐缩小。

由于引入的 CBAM 模块选择的是先通道注意力后空间注意力机制，且是添加到每个 block 中的，因此对于准确率的提升可以总结为注意力机制能够有效帮助捕捉到更多图像特征信息，证实了注意力机制在图像相关任务的普及性，除了能应用在图像描述、图像分割等领域，适合添加进各类 Backbone 中进行效果提升，对于类增量学习来说可以更好维持 plasticity 和 stability 的平衡。相比于先前改进措施是基于策略的，本章改进措施更多是从图像分类任务本身入手，所以使得原特征提取器和分类器就能根据更加具体特征进行分类。

CBAM 模块的引入由于是嵌入在网络结构中的，因此训练时间相较于原始 PODNet 模型会相应延长，整体运行时间如果是 50steps 的实验会多 1/3 左右的运行时间。该阶段实验也是在 AutoDL 平台上租用 NVIDIA RTX 2080 Ti 显卡进行模型训练的，选用数据集为 CIFAR100。

总的来说，引入注意力机制的模型能够在任务本身维度上对模型分类准确率进行提升，可配合其他改进策略共同提升。

§4.3.4 基于样本存储的改进 PODNet 实验结果

本章节采用的方法为扩展存储容量以及动态调节每阶段任务各类存储样本数量针对不同 steps 设置相应 memory rate 进行存储容量分配，表 4.7 为对应四种 steps 的记忆率列表。

表 4.7 记忆存储率对应表

	Memory rate list
5 steps	[0.8,0.8,0.6,0.6,0.6,0.6]
10 steps	[0.7,0.7,0.7,0.7,0.7,0.6,0.6,0.6,0.7,0.7]
25 steps	[0.7,0.7,0.7,0.7,0.7,0.7,0.7,0.7,0.7,0.7,0.8,0.8,0.8,0.8,0.8,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9,0.9]
50 steps	[0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9, 0.9]

上表中对应各 steps 的 memory rate list 的每一个参数对应阶段学习任务对存储容量的动态调整，继而影响到每类旧样本存储数量。基于上述记忆率进行改进的 PODNet 实验结果如下，实验同在 AutoDL 平台上租用 NVIDIA RTX 2080 Ti 显卡进行训练，选用数据集为 CIFAR100。

表 4.8 基于样本存储的改进 PODNet 实验数据表

	CIFAR100			
	50steps	25steps	10steps	5steps
New classes per step	1	2	5	10
PODNet(CNN)	58.01±0.56	60.65±1.12	62.46±0.01	64.56±1.03
PODNet(CNN)-MEM	**63.26±0.34**	**64.55±0.66**	**66.23±0.7**	**67.55±0.84**
PODNet(NME)	61.43±0.95	63.50±1.00	64.72±1.04	65.69±1.10
PODNet(NME)-MEM	**62.59±1.04**	**63.67±1.11**	**64.59±1.06**	**64.91±1.13**

综合表 4.8 数据，可以发现通过动态控制存储样本数量可以较大幅度提升准确率，其中对于使用 CNN 分类器的准确率平均上升 5%左右，而使用 NME 分类器准确率没有什么提升，和原先准确率差不多。通过总体准确率的提升，可以说明对基于回放策略的改进可以使得模型在学习不同 task 时更加灵活对新旧样本进行处理；相比先前改进方法，本方法对准确率提升更明显，也可以证明合理分配新旧样本存储比例以及旧

样本存储个数对于类增量学习任务是至关重要的，也是目前类增量学习受限制的原因所在，因为需要在有限存储空间内进行旧样本存储，若存储空间可以提升，即使是小幅度提升都能对准确率带来较大改善。

对于 CNN 和 NME 两种分类器策略对上述改进方式的较大差异，本文认为与最近均值分类算法是基于类间图像平均特征进行判断的有关，可能是引入的同类别图像特征较为相似，导致没有为最终分类带来很多帮助。而对于 CNN 的提升较为客观，可能是随着训练样本的增加，模型最后分类层同样得到训练，使得分类准确率提升。

§4.3.5 基于图像压缩的改进 PODNet 实验结果

本章节采用的方法为对输入图像集进行压缩处理，在仍能保留图像特征的前提下压缩图像大小，使得同样存储空间下能存放更多样本。由于 CIFAR100 数据集格式不是 RGB 图像，对于 CIFAR100 数据集进行处理的话需涉及到先对数据集进行下载，然后原始数据转换成 RGB 图像再进行压缩处理，过程中会影响对图像特征的保存。因此本章方法是基于 ImageNet100 数据集以及 ResNet-18 网络结构进行的实验，对比实验是在 UCIR 模型上进行的，在本方法实验中只在 5 steps 上进行了实验，因为 ImageNet 图像集本身包含较多数据且数据量较大，跑 10/25/50 steps 实验基本耗时几天，所以只在 5 steps 的情况下进行了实验，initial classes 为 500，后续每个 step 学习 100 个类，具体结果如表 4.9 所示。

表 4.9 基于图像压缩的改进 PODNet 实验数据表

ImaageNet100			
quality	100	75	50
UCIR(CNN)	46.17	**46.2**	44.01
UCIR(NME)	49.02	**49.32**	46.13

上表中 quality 是指对原始图像的压缩比例，100 即为原始图像，75 即为对原始图像采用 75%的 RGB 压缩率，通过上述数据可以看到当在一定范围内进行图像压缩时，可以对准确率进行提升，但是当压缩率过大，压缩过程中会损失部分图像特征，导致准确率反而下降。对于 CNN 和 NME 两种分类器的实验，准确率提升都不是很明显也没有较大差异，本文认为可能是单纯对图像特征进行压缩处理而没有动态压缩导致，之后可以尝试其他压缩方法或者降维方法对图像特征进行转变，同时应该配合对回放机制的处理，即旧样本存储个数受样本本身大小控制而非数据集类别。本章实验证明了对原始图像本身做处理以更好适应增量学习的可行性，后续改进需进一步探讨。

§4.3.6 多种方法结合的改进 PODNet 实验结果

先前章节从四个维度对 PODNet 进行了改进，部分改进方式也从实验结果本身证明了改进的可行性以及有效性，因此本章节对前文介绍的基于知识蒸馏的改进方式、基于网络结构的赶紧方式以及基于样本存储的改进方式进行融合，并做了相应实验。最后根据所做实验可视化展示对比原 PODNet 的效果，评估对准确率提升最高的改进措施。

首先对知识蒸馏和网络结构两类方法进行融合，采用 temperature 为 10，参数为 0.9 的 Softmax 知识蒸馏以及先后加入通道注意力模块和空间注意力模块的 CBAM 机制进行实验。实验同样基于 AutoDL 平台上 NVIDIA RTX 2080 Ti 显卡进行，采用 CIFAR100 数据集，各超参数设置同 PODNet 模型训练过程，表 4.10 为实验结果展示。

表 4.10 基于知识蒸馏和注意力机制的改进 PODNet 实验结果

	CIFAR100			
	50steps	25steps	10steps	5steps
New classes per step	1	2	5	10
PODNet(CNN)	58.01±0.56	60.65±1.12	62.46±0.01	64.56±1.03
PODNet(CNN)-KD	58.34±0.8	61.56±0.79	63.54±0.93	65.2±0.44
PODNet(CNN)-CBAM	**60.18±2.56**	**61.78±0.82**	64.47±1.05	66.17±0.97
PODNet(CNN)-KD+CBAM	59.65±1.2	61.62±0.86	**64.72±1.07**	**66.33±0.86**
PODNet(NME)	61.43±0.95	63.50±1.00	64.72±1.04	65.69±1.10
PODNet(NME)-KD	**62.69±1.28**	**64.68±1.02**	**66.3±1.29**	**67.03±1.1**
PODNet(NME)-CBAM	59.83±0.74	63.89±1.12	65.62±1.42	66.20±1.2
PODNet(NME)-KD+CBAM	59.62±1.03	63.78±1.76	65.84±1.4	66.8±1.23

两种改进策略结合的实验结果如上，可以看出对采用 CNN 和 NME 两种不同分类器时使用单一或者结合的方式作用会不一样，对于 CNN 分类器来说，在小任务增量学习问题上，使用单独结合 CBAM 的方式会得到较高提升，对于每阶段学习较多类任务的情况，采用两种结合的方式能达到较好的效果；对于 NME 分类器来说，CBAM 的改进效果不是很明显，采取单一新旧 Softmax 结果差作为约束方式更有效。

其次对网络结构和样本存储两类方法进行融合，实验同样基于 AutoDL 平台上 NVIDIA RTX 2080 Ti 显卡进行，采用 CIFAR100 数据集，各超参数设置同 PODNet 模型训练过程，表 4.11 为实验结果展示。

表 4.11 基于注意力机制和样本存储的改进 PODNet 实验结果

	CIFAR100			
	50steps	25steps	10steps	5steps
New classes per step	1	2	5	10
PODNet(CNN)	58.01±0.56	60.65±1.12	62.46±0.01	64.56±1.03
PODNet(CNN)-CBAM	60.18±2.56	61.78±0.82	64.47±1.05	66.17±0.97
PODNet(CNN)-MEM	63.26±0.34	64.55±0.66	66.23±0.7	67.55±0.84
PODNet(CNN)-CBAM+MEM	**64.12±0.48**	**66.00±0.83**	**67.98±1.02**	**69.45±1.1**
PODNet(NME)	61.43±0.95	63.50±1.00	64.72±1.04	65.69±1.10
PODNet(NME)-CBAM	59.83±0.74	63.89±1.12	65.62±1.42	66.20±1.2
PODNet(NME)-MEM	**62.59±1.04**	63.67±1.11	64.59±1.06	**66.82±1.13**
PODNet(NME)-CBAM+MEM	62.39±1.28	**63.94±1.06**	**65.7±1.34**	66.72±1.48

两种改进策略结合的实验结果如上，可以看出在动态调整样本存储空间的基础上叠加 CBAM 注意力模块可以在大多数场景上得到进一步准确率提升。由此，在能够调整存储数据总量的前提下，使用网络结构与样本存储的结合改进策略可以达到最优效果，而在限定存储输入总量的前提下，在改进网络结构的模型上融入知识蒸馏方法改进可以得到效果提升。

图 4-1 和图 4-2 为本文所做复现实验以及各类改进实验的综合结果展示，体现出经过不同类别学习后模型实时准确率展示，考虑到小样本总量学习任务总数过多不便展示，因此此处展示的是增量学习任务总数为 5 steps（初始学习 50 个类，之后每次增量任务学习 10 个类的数据）和 10 steps（初始学习 50 个类，之后每次增量任务学习 5 个类的数据）的实验结果。

图中横坐标即为当前已学习的类别总数，初始值为 50，对应初始学习后模型对已学习的 50 个类数据进行分类预测的准确率值；随着不断学习新任务，横坐标的数值代表已学习类别总数的递增，对应的纵坐标表示截止到目前阶段的预测结果，将得到同步更新。

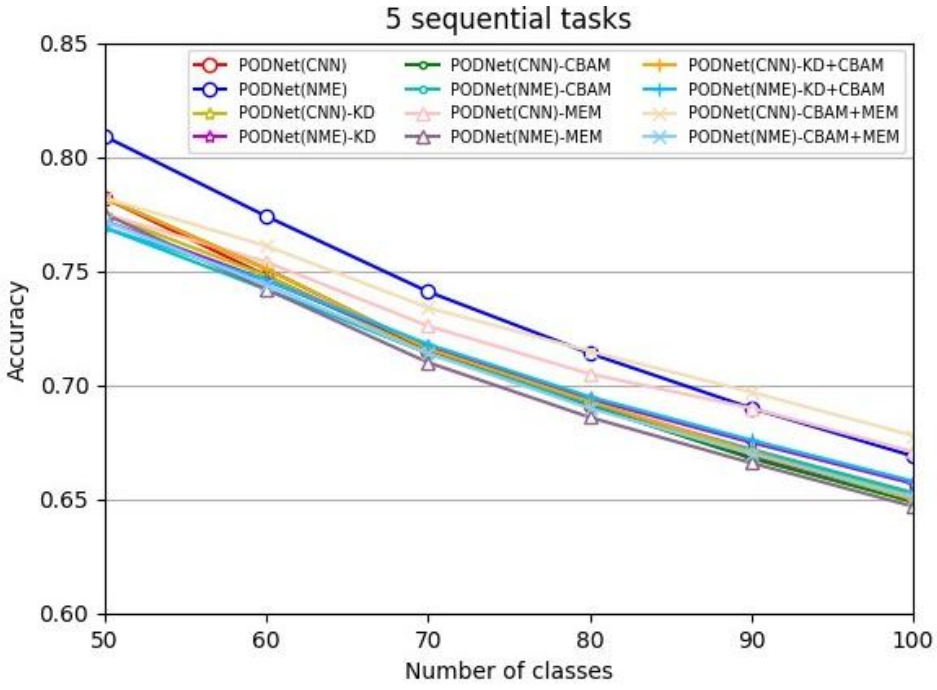

图 4-1 本文实验结果汇总展示（5 steps）

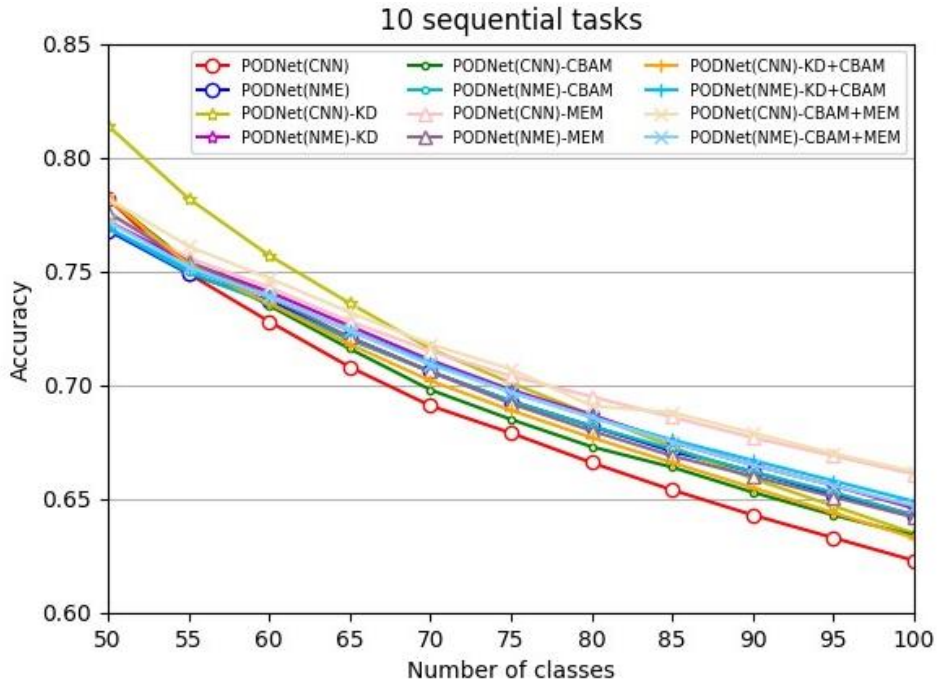

图 4-2 本文实验结果汇总展示（10 steps）

§4.4 本章小结

本章首先介绍了复现 PODNet 模型过程中所依赖的开发环境以及实验部署细节，并与原文实验结果进行对比，其次展示了基于四种改进策略的实验结果以及多种改进方式结合的效果并分析模型准确率上升或下降的原因，最终对所有所做实验进行可视化展示。

第 5 章 总结与展望

本章对全文针对类增量学习领域的工作内容和创新点进行概括与总结，并对当前研究现存不足之处及其未来可参考改进方向进行展望。

§5.1 本文总结

§5.1.1 本文的主要工作

本文主要研究的是卷积神经网络中增量学习方法，研究场景为类增量学习即增量学习方法在图像分类任务中的作用，采用的主体算法框架为 Douillard 等人提出的 PODNet 模型，本文基于 PODNet 相关思想进行了四方面的改进尝试。

(1) 基于知识蒸馏的改进 PODNet 模型，使用新旧模型 softmax 分类预测结果进行知识蒸馏操作，探索传统知识蒸馏策略对模型准确率的影响。

(2) 基于网络结构的改进 PODNet 模型，在 ResNet-32 网络结构中引入通道注意力模块和空间注意力模块，通过获取更具有代表性的图像特征提升分类准确率，从图像分类任务本质进行改进。

(3) 基于样本存储的改进 PODNet 模型，代替固定样本存储空间，选择动态调整各类别旧样本存储数，验证回放机制的改进对准确率的影响程度，并进行相应融合实验。

(4) 基于图像压缩的改进 PODNet 模型，对原始图像数据集进行了不同程度压缩处理，使得同样存储空间限制下存放更多样本，探索其有效性以及可行性。

§5.1.2 本文的主要创新点

本文基于 PODNet 模型实现的类增量学习算法主要有以下几个创新点：

针对类增量学习任务，在 Backbone 中引入注意力机制模块，以拓展注意力机制在不同领域上的运用，并借此增强模型分类效果。

对于类增量学习知识蒸馏正则化策略的扩展进行探索，寻求不同策略叠加以达到模型效果提升，了解类增量学习中对特征提取器和分类器的不同约束带来的效果影响。

对样本存储空间以及旧样本存储数进行动态调整，最大限度利用有限空间解决新旧样本差异较大的问题，发现对回放机制的改进对于类增量学习其他策略来说有更大的上升空间。

§5.2 展望

虽然本文基于 PODNet 做了几个改进，从回放策略/正则化策略/网络结构入手进行相应研究，但是实验过程以及阅读文献的过程中仍有一些疑虑或者难点待解决：

(1) 如何控制在恒定的样本存储条件下进一步动态调整样本数量，包括新旧样本数量对比以及旧样本类间数据量。

(2) 对于原始图像进行压缩的方式，有没有能更适应网络参数更新的压缩处理方式。

(3) 对于分类器方面，多采用单一分类器，可否尝试使用多分类器等方式进行进一步训练。

对于第一个问题，2021 年 Liu[21]等人提出了结合强化学习思想的类增量学习模型，设置两阶段强化学习目标，第一阶段确定新旧样本存储比例，第二阶段确定所有旧样本存储空间内不同类别的动态划分。

对于第二个问题，2021 年 Zhao[18]等人提出一多重学习策略的类增量学习模型，利用降维操作生成基于原始图像的低维样本，并将原始样本和低维样本共同送入网络学习，基于正则化策略使得原始样本和低维样本最终分类结果接近同时动态调整图像压缩模块的参数。曾经尝试过借鉴该方法思想进行复现，但是没有源码复现难度太大，中间需要复杂训练过程，需要再花时间仔细阅读原文进行理解。

针对第三个问题，2020 年 Liu[20]等人提出了多分类器范式解决类增量学习问题，相比单一分类器，设置 k 个副分类器利用 k 个副分类器结果和主分类器结果结合，解决正则化策略容易被单一分类器结果影响的可能，同时引入分布外数据进行副分类器训练，利用分类器结果差异进行约束。之后可以考虑结合拓展分类器的方法，进行更多实验。

致谢

终于到了提笔写下致谢的时刻，心中也有无限感慨，伴随着 2018 年盛夏蝉鸣声我开启了这四年在上海大学的学习旅程，也即将在 2022 的夏天和她说再见，开启下一个人生阶段。目光所至，皆是回忆，纵有万般不舍，也仍心存感激。

想要感谢的有很多，首先要感我的指导老师——马丽艳老师。很荣幸能够在本科期间遇见这样一位优秀且可爱的老师，从大二专业课开始直到大四毕设，三年的师生缘分让我非常珍惜。在专业课的学习中，老师会用鼓励的方式让我们不对陌生的知识畏惧，也让我有了更多自信；也很感谢马老师能帮我写推荐信，让我有机会我开启另一段求学路；有关毕设，对于我来说很幸运能够继续跟着马老师继续做，这是我第一次接触增量学习领域，难免会有点胆怯或者无从下手，马老师在开题前就给我提供了很多学习资源供参考，在选题等思路上也非常尊重每个人的意见，在我有不理解的地方会及时解答或者指出另外可行的方法，以一种启发的方式让我们主动学习。我其实是一个有拖延症的人，所以每周组会我都很开心的接受这种“push”，努力向优秀同学学习交流，也逐渐成为一个更有计划的人。也很感谢辛明军老师一直以来对我的认可，让我更有勇气去追逐自己的目标。

感谢我的父母，世界上最爱我的两个人，谢谢他们一直以来对我的无条件支持以及帮助，在我需要作出重大决定时会提供建议但从不干涉我的选择；也会在我难过不自信的时候永远相信我，是我永远最且坚强的后盾，我也要努力成长为一个可以让他们依靠的存在。

感谢我的朋友们，因为有了你们，我的生活有了不一样的色彩。谢谢我的好朋友茅至文和杨家逸，在我们认识的七年间始终包容我、支持我，参与我所有的难过以及喜悦；谢谢我的室友杨诣程、辛童、付昊洋、董韵涵在本科期间的陪伴，让我拥有了美好的大学生活，我们来自五湖四海，奔向天南海北；谢谢虞季南学长在毕设期间的答疑解惑，为我的改进方向提供意见；以及对谢谢我的学妹苏千一和徐煜，虽然不在同一年级但是我们之间有着很多默契；还有很多新结识的好朋友，在每次的沟通中都让我重新找到继续努力以及不断前行的动力。

最后感谢一直以来不断前进的自己，拥有不畏挫折的勇气，我相信我终能看到想要的风景，找到属于自己的光。

感恩所有的相遇相识相知，来日方长，后会有期。愿我们都能拥有光明的未来！

参考文献

[1] LeCun Y, Touresky D, Hinton G, et al. A theoretical framework for back-propagation[C]//Proceedings of the 1988 connectionist models summer school. 1988, 1: 21-28.

[2] Bottou L. Stochastic gradient learning in neural networks[J]. Proceedings of Neuro-Nımes, 1991, 91(8): 12.

[3] Kirkpatrick J, Pascanu R, Rabinowitz N, et al. Overcoming catastrophic forgetting in neural networks[J]. Proceedings of the national academy of sciences, 2017, 114(13): 3521-3526.

[4] Martens J. New insights and perspectives on the natural gradient method[J]. arXiv preprint arXiv:1412.1193, 2014.

[5] Zenke F, Poole B, Ganguli S. Continual learning through synaptic intelligence[C]//International Conference on Machine Learning. PMLR, 2017: 3987-3995.

[6] Aljundi R, Babiloni F, Elhoseiny M, et al. Memory aware synapses: Learning what (not) to forget[C]//Proceedings of the European Conference on Computer Vision (ECCV). 2018: 139-154.

[7] Li Z, Hoiem D. Learning without forgetting[J]. IEEE transactions on pattern analysis and machine intelligence, 2017, 40(12): 2935-2947.

[8] Robins A. Catastrophic forgetting, rehearsal and pseudorehearsal[J]. Connection Science, 1995, 7(2): 123-146.

[9] Rebuffi S A, Kolesnikov A, Sperl G, et al. icarl: Incremental classifier and representation learning[C]//Proceedings of the IEEE conference on Computer Vision and Pattern Recognition. 2017: 2001-2010.

[10] Hou S, Pan X, Loy C C, et al. Learning a unified classifier incrementally via rebalancing[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019: 831-839.

[11] Rusu A A, Rabinowitz N C, Desjardins G, et al. Progressive neural networks[J]. arXiv preprint arXiv:1606.04671, 2016.

[12] Yoon J, Yang E, Lee J, et al. Lifelong learning with dynamically expandable networks[J]. arXiv preprint arXiv:1708.01547, 2017.

[13] Gepperth A, Karaoguz C. A bio-inspired incremental learning architecture for applied perceptual problems[J]. Cognitive Computation, 2016, 8(5): 924-934.

[14] Belouadah E, Popescu A, Kanellos I. A comprehensive study of class incremental learning algorithms for visual tasks[J]. Neural Networks, 2021, 135: 38-54.

[15] Douillard A, Cord M, Ollion C, et al. Podnet: Pooled outputs distillation for small-tasks incremental learning[C]//European Conference on Computer Vision. Springer, Cham, 2020: 86-102.

[16] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

[17] Woo S, Park J, Lee J Y, et al. Cbam: Convolutional block attention module[C]//Proceedings of the European conference on computer vision (ECCV). 2018: 3-19.

[18] Zhao H, Wang H, Fu Y, et al. Memory efficient class-incremental learning for image classification[J]. IEEE Transactions on Neural Networks and Learning Systems, 2021.

[19] Wang L, Zhang X, Yang K, et al. Memory Replay with Data Compression for Continual Learning[J]. arXiv preprint arXiv:2202.06592, 2022.

[20] Liu Y, Parisot S, Slabaugh G, et al. More classifiers, less forgetting: A generic multi-classifier paradigm for incremental learning[C]//European Conference on Computer Vision. Springer, Cham, 2020: 699-716.

[21] Liu Y, Schiele B, Sun Q. RMM: Reinforced memory management for class-incremental learning[J]. Advances in Neural Information Processing Systems, 2021, 34.

[22] Yan S, Xie J, He X. Der: Dynamically expandable representation for class incremental learning[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2021: 3014-3023.

附录：部分源程序清单

采取 herding 选择策略进行旧样本选择：

```
    # 损失函数定义
    def _compute_loss(self, inputs, outputs, targets, onehot_targets,
memory_flags):
        features, logits, atts = outputs["raw_features"], outputs["logits"],
outputs["attention"]

        if self._post_processing_type is None:
            scaled_logits = self._network.post_process(logits)
            outputs_now_results = self._network.post_process(logits/2)
        else:
            scaled_logits = logits * self._post_processing_type

        if self._old_model is not None:
            with torch.no_grad():
                old_outputs = self._old_model(inputs)
                old_features = old_outputs["raw_features"]
                old_results = old_outputs["logits"]
                old_atts = old_outputs["attention"]
                #                     scaled_old_logits                      =
self._old_model.post_process(old_results)
                #                   outputs_old_results                      =
self._old_model.post_process(scaled_old_logits/2)

        # 分类 loss
        self._metrics["nca"]和 self.metrics["cce"]
        if self._nca_config:
            nca_config = copy.deepcopy(self._nca_config)
            if self._network.post_processor:
                nca_config["scale"] = self._network.post_processor.factor

            loss = losses.nca(
                logits,
                targets,
                memory_flags=memory_flags,
                class_weights=self._class_weights,
                **nca_config
            )
            self._metrics["nca"] += loss.item()
        elif self._softmax_ce:
            loss = F.cross_entropy(scaled_logits, targets)
            self._metrics["cce"] += loss.item()
            logger.info('scaled_logits:{}'.format(scaled_logits))
logger.info('scaled_logits_size:{}'.format(scaled_logits.size()))

        # 知识蒸馏 loss 部分
        if self._old_model is not None:
```

```
            # if self._use_mimic_score:
            #         old_class_logits  =  logits[...,  :self._n_classes  -
self._task_size]
            #     old_class_old_logits = old_logits[..., :self._n_classes -
self._task_size]

            #                 mimic_loss    =    F.mse_loss(old_class_logits,
old_class_old_logits)
            #     mimic_loss *= (self._n_classes - self._task_size)
            #     loss += mimic_loss
            #     self._metrics["mimic"] += mimic_loss.item()

            if self._ranking_loss:
                ranking_loss      =      self._ranking_loss["factor"]      *
losses.ucir_ranking(
                    logits,
                    targets,
                    self._n_classes,
                    self._task_size,
                    nb_negatives=min(self._ranking_loss["nb_negatives"],
self._task_size),
                    margin=self._ranking_loss["margin"]
                )
                loss += ranking_loss
                self._metrics["rank"] += ranking_loss.item()

             # flat
            if self._pod_flat_config:
                if self._pod_flat_config["scheduled_factor"]:
                    factor   =   self._pod_flat_config["scheduled_factor"]   *
math.sqrt(
                        self._n_classes / self._task_size
                    )
                else:
                    factor = self._pod_flat_config.get("factor", 1.)
                #flat-loss 最后一层 embedding 的新旧 feature loss
                # alpha = 0.5
                pod_flat_loss              =              factor              *
losses.embeddings_similarity(old_features, features)
                loss += pod_flat_loss
                self._metrics["flat"] += pod_flat_loss.item()

                # 把 flat-loss 知识蒸馏改成对 softmax 结果进行知识蒸馏,logits 为新
模型的 logits，old_results 为老模型的 logits
                # T = 10
                # # new_results = logits[0:len(old_results)]
                # criterion = torch.nn.KLDivLoss()
                #  outputs_old_results  =  F.softmax(old_results/T,dim=-1)  #
teacher
                # # logger.info('')
                #             #             logger.info('size             of
```

```
outputs-old-features:{}'.format(outputs_old_results))
                #  outputs_now_results  =  F.log_softmax(logits/T,dim=-1)   #
student
                #             #             logger.info('size             of
outputs-now-features:{}'.format(outputs_now_results.size()))
                #                     outputs_new_results                    =
outputs_now_results[:,-outputs_old_results.shape[1]:]
                #             #             logger.info('size             of
outputs-new-features:{}'.format(outputs_new_results))
                #                                                             #
logger.info('new-equal-old?:{}'.format(outputs_old_results.equal(outputs_n
ew_results)))
                #       pod_flat_softmax_loss       =             0.9       *
criterion(outputs_new_results,outputs_old_results)*T*T
                #                                                             #
logger.info('criteration:{}'.format(criterion(outputs_new_results,outputs_
old_results)))
                # loss += pod_flat_softmax_loss
                # self._metrics["softmax"] += pod_flat_softmax_loss.item()

            # spatial loss
            if self._pod_spatial_config:
                if self._pod_spatial_config.get("scheduled_factor", False):
                    factor  =  self._pod_spatial_config["scheduled_factor"]  *
math.sqrt(
                        self._n_classes / self._task_size
                    )
                else:
                    factor = self._pod_spatial_config.get("factor", 1.)

                pod_spatial_loss = factor * losses.pod(
                    old_atts,
                    atts,
                    memory_flags=memory_flags.bool(),
                    task_percent=(self._task + 1) / self._n_tasks,
                    **self._pod_spatial_config
                )
                loss += pod_spatial_loss
                self._metrics["pod"] += pod_spatial_loss.item()
                self._old_model.zero_grad()
                self._network.zero_grad()

        return loss

def pod(
    list_attentions_a,
    list_attentions_b,
    collapse_channels="spatial",
    normalize=True,
    memory_flags=None,
    only_old=False,
    **kwargs
```

```
):
    assert len(list_attentions_a) == len(list_attentions_b)

    loss = torch.tensor(0.).to(list_attentions_a[0].device)
    for i, (a, b) in enumerate(zip(list_attentions_a, list_attentions_b)):
        # shape of (b, n, w, h)
        assert a.shape == b.shape, (a.shape, b.shape)

        if only_old:
            a = a[memory_flags]
            b = b[memory_flags]
            if len(a) == 0:
                continue

        a = torch.pow(a, 2)
        b = torch.pow(b, 2)

        if collapse_channels == "channels":
            a = a.sum(dim=1).view(a.shape[0], -1)  # shape of (b, w * h)
            b = b.sum(dim=1).view(b.shape[0], -1)
        elif collapse_channels == "width":
            a = a.sum(dim=2).view(a.shape[0], -1)  # shape of (b, c * h)
            b = b.sum(dim=2).view(b.shape[0], -1)
        elif collapse_channels == "height":
            a = a.sum(dim=3).view(a.shape[0], -1)  # shape of (b, c * w)
            b = b.sum(dim=3).view(b.shape[0], -1)
        elif collapse_channels == "gap":
            a = F.adaptive_avg_pool2d(a, (1, 1))[..., 0, 0]
            b = F.adaptive_avg_pool2d(b, (1, 1))[..., 0, 0]
        elif collapse_channels == "spatial":
            a_h = a.sum(dim=3).view(a.shape[0], -1)
            b_h = b.sum(dim=3).view(b.shape[0], -1)
            a_w = a.sum(dim=2).view(a.shape[0], -1)
            b_w = b.sum(dim=2).view(b.shape[0], -1)
            a = torch.cat([a_h, a_w], dim=-1)
            b = torch.cat([b_h, b_w], dim=-1)
        else:
            raise      ValueError("Unknown      method      to      collapse:
{}".format(collapse_channels))

        if normalize:
            a = F.normalize(a, dim=1, p=2)
            b = F.normalize(b, dim=1, p=2)

        layer_loss = torch.mean(torch.frobenius_norm(a - b, dim=-1))
        loss += layer_loss

    return loss / len(list_attentions_a)
```

ResNet-32+CBAM：

```
from ast import Import
from re import M
from grpc import Channel
import torch.nn as nn
import torch.utils.model_zoo as model_zoo
from torch.nn import functional as F

import math
try:
    from torch.hub import load_state_dict_from_url
except ImportError:
    from torch.utils.model_zoo import load_url as load_state_dict_from_url
import torch

def conv3x3(in_planes, out_planes, stride=1):
    return nn.Conv2d(in_planes, out_planes, kernel_size=3, stride=stride,
padding=1, bias=False)

def conv1x1(in_planes, out_planes, stride=1):
    return nn.Conv2d(in_planes, out_planes, kernel_size=1, stride=stride,
bias=False)

class BasicBlock(nn.Module):
    expansion = 1
    # inplanes 输入维度 planes 输出维度 stride=1 不会改变高和宽
    def __init__(self, inplanes, planes, stride=1, downsample=None,
last_relu=True):
        super(BasicBlock, self).__init__()
        self.conv1 = conv3x3(inplanes, planes, stride)
        self.bn1 = nn.BatchNorm2d(planes)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = conv3x3(planes, planes)
        self.bn2 = nn.BatchNorm2d(planes)
        self.ca = ChannelAttention(planes)
        self.sa = SpatialAttention()
        self.downsample = downsample
        self.stride = stride
        self.last_relu = last_relu

    def forward(self, x):
        identity = x
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)
        out = self.conv2(out)
        out = self.bn2(out)
        out = self.ca(out) * out
        out = self.sa(out) * out
        if self.downsample is not None:
            identity = self.downsample(x)
```

```
        out += identity
        if self.last_relu:
            out = self.relu(out)
        return out

class Bottleneck(nn.Module):
    expansion = 4
    def  __init__(self,  inplanes,  planes,  stride=1,  downsample=None,
last_relu=True):
        super(Bottleneck, self).__init__()
        self.conv1 = conv1x1(inplanes, planes)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = conv3x3(planes, planes, stride)
        self.bn2 = nn.BatchNorm2d(planes)
        self.conv3 = conv1x1(planes, planes * self.expansion)
        self.bn3 = nn.BatchNorm2d(planes * self.expansion)
        self.relu = nn.ReLU(inplace=True)\
        self.ca = ChannelAttention(planes * self.expansion)
        self.sa = SpatialAttention()
        self.downsample = downsample
        self.stride = stride
        self.last_relu = last_relu

    def forward(self, x):
        identity = x
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)
        out = self.conv2(out)
        out = self.bn2(out)
        out = self.relu(out)
        out = self.conv3(out)
        out = self.bn3(out)
        out = self.ca(out) * out
        out = self.sa(out) * out
        if self.downsample is not None:
            identity = self.downsample(x)
        out += identity
        if self.last_relu:
            out = self.relu(out)
        return out

class ResNet(nn.Module):
    def __init__(
        self,
        block,
        layers,
        zero_init_residual=True,
        nf=16,
        last_relu=False,
        initial_kernel=3,
        **kwargs
```

```
    ):
        super(ResNet, self).__init__()
        self.last_relu = last_relu
        self.inplanes = nf
        self.conv1 = nn.Conv2d(3, nf, kernel_size=initial_kernel, stride=1,
padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(nf)
        self.relu = nn.ReLU(inplace=True)

        #  在网络的第一层加入注意力机制
        # self.ca = ChannelAttention(nf)
        # self.sa = SpatialAttention()

        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        self.layer1 = self._make_layer(block, 1 * nf, layers[0])
        self.layer2 = self._make_layer(block, 2 * nf, layers[1], stride=2)
        self.layer3 = self._make_layer(block, 4 * nf, layers[2], stride=2)
        self.layer4 = self._make_layer(block, 8 * nf, layers[3], stride=2,
last=True)

        # 在网络的卷积层的最后一层加入注意力机制
        # self.ca1 = ChannelAttention(nf)
        # self.sa1 = SpatialAttention()
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.out_dim = 8 * nf * block.expansion
        print("Features dimension is {}.".format(self.out_dim))
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(m.weight,          mode='fan_out',
nonlinearity='relu')
            elif isinstance(m, nn.BatchNorm2d):
                nn.init.constant_(m.weight, 1)
                nn.init.constant_(m.bias, 0)
        if zero_init_residual:
            for m in self.modules():
                if isinstance(m, Bottleneck):
                    nn.init.constant_(m.bn3.weight, 0)
                elif isinstance(m, BasicBlock):
                    nn.init.constant_(m.bn2.weight, 0)
    def _make_layer(self, block, planes, blocks, stride=1, last=False):
        downsample = None
        if stride != 1 or nf != planes * block.expansion:
            downsample = nn.Sequential(
                conv1x1(nf, planes * block.expansion, stride), #plane*block
= channel*4
                nn.BatchNorm2d(planes * block.expansion),
            )
        layers = []
        layers.append(block(nf, planes, stride, downsample))
        nf = planes * block.expansion
        for i in range(1, blocks):
```

```
            if i == blocks - 1 or last:
                layers.append(block(nf, planes, last_relu=False))
            else:
                layers.append(block(nf, planes, last_relu=self.last_relu))

        return nn.Sequential(*layers)
    @property
    def last_block(self):
        return self.layer4
    @property
    def last_conv(self):
        return self.layer4[-1].conv2
    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        # 加入注意力机制
        x = self.ca(x) * x
        x = self.sa(x) * x
        x = self.maxpool(x)
        x_1 = self.layer1(x)
        x_1 = self.ca(x_1) * x_1
        x_1 = self.sa(x_1) * x_1
        x_2 = self.layer2(self.end_relu(x_1))
        x_2 = self.ca(x_2) * x_2
        x_2 = self.sa(x_2) * x_2
        x_3 = self.layer3(self.end_relu(x_2))
        x_3 = self.ca(x_3) * x_3
        x_3 = self.sa(x_3) * x_3
        x_4 = self.layer4(self.end_relu(x_3))
        # 加入注意力机制
        # x_4 = self.ca1(x_4) * x_4
        # x_4 = self.sa1(x_4) * x_4
        raw_features = self.end_features(x_4)
        features = self.end_features(F.relu(x_4, inplace=False))
        return {
            "raw_features": raw_features,
            "features": features,
            "attention": [x_1, x_2, x_3, x_4]}
    def end_features(self, x):
        x = self.avgpool(x)
        x = x.view(x.size(0), -1)
        return x

    def end_relu(self, x):
        if hasattr(self, "last_relu") and self.last_relu:
            return F.relu(x)
        return x

class ChannelAttention(nn.Module):
    def __init__(self, in_planes, ratio = 16):
```

```
        super(ChannelAttention, self).__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.max_pool = nn.AdaptiveAvgPool2d(1)

        self.fc1 = nn.Conv2d(in_planes, in_planes // 16, 1, bias = False)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Conv2d(in_planes // 16, in_planes, 1, bias = False)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        avg_out = self.fc2(self.relu1(self.fc1(self.avg_pool(x))))
        max_out = self.fc2(self.relu1(self.fc1(self.max_pool(x))))
        out = avg_out + max_out
        return self.sigmoid(out)

class SpatialAttention(nn.Module):
    def __init__(self, kernel_size = 7):
        super(SpatialAttention, self).__init__()

        assert kernel_size in (3,7), ' kernel size must be 3 or 7'
        padding = 3 if kernel_size == 7 else 1

        self.conv1 = nn.Conv2d(2, 1, kernel_size, padding = padding, bias
= False)
        self.sigmoid = nn.Sigmoid()

    def forward(self,x):
        avg_out = torch.mean(x, dim = 1, keepdim = True)
        max_out, _ = torch.max(x, dim = 1, keepdim = True)
        x = torch.cat([avg_out, max_out], dim = 1)
        x = self.conv1(x)
        return self.sigmoid(x)

def resnet32(**kwargs):
    model = ResNet(BasicBlock, [5, 4, 3, 2], **kwargs)
    if pretrained:
        pretrained_state_dict = model_zoo.load_url(model_urls['resnet32'])
        now_state_dict = model.state_dict()
        now_state_dict.update(pretrained_state_dict)
        model.load_state_dict(now_state_dict)
    return model

model.load_state_dict(model_zoo.load_url(model_urls['resnet152']))
    return model
```

采取 herding 选择策略进行旧样本选择：

```
def build_examplars(
self,      inc_dataset,      herding_indexes,      memory_per_class=None,
data_source="train"
):
logger.info("Building & updating memory.")
memory_per_class = memory_per_class or self._memory_per_class
herding_indexes = copy.deepcopy(herding_indexes)

data_memory, targets_memory = [], []
data_memory_feature = []
class_means = np.zeros((self._n_classes, self._network.features_dim))

for class_idx in range(3):
inputs, loader = inc_dataset.get_custom_loader(
class_idx, mode="test", data_source=data_source
)
features, targets = utils.extract_features(self._network, loader)
print('icarl-class_idx: {}','features:{}').format(class_idx,features)
features_flipped, _ = utils.extract_features(
self._network,
inc_dataset.get_custom_loader(class_idx,                    mode="flip",
data_source=data_source)[1]
)
if class_idx >= self._n_classes - self._task_size:

if self._herding_selection["type"] == "icarl":
selected_indexes = herding.icarl_selection(features, memory_per_class)
else:
raise ValueError(
"Unknown herding selection {}.".format(self._herding_selection)
)

herding_indexes.append(selected_indexes)

# Reducing examplars:
try:
selected_indexes = herding_indexes[class_idx][:memory_per_class]
herding_indexes[class_idx] = selected_indexes
except:
import pdb
pdb.set_trace()

examplar_mean = self.compute_examplar_mean(
features, features_flipped, selected_indexes, memory_per_class
)

data_memory.append(inputs[selected_indexes])
targets_memory.append(targets[selected_indexes])
data_memory_feature.append()
```

```
class_means[class_idx, :] = examplar_mean

data_memory = np.concatenate(data_memory)
targets_memory = np.concatenate(targets_memory)

return data_memory, targets_memory, herding_indexes, class_means
```

按不同 quality 压缩数据处理：

```
import glob
import os
import argparse
from tqdm import tqdm
from skimage import io

def get_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('--input_dir',
default='datapath/JPEG-quality-imagenet256/n01440764/')
    parser.add_argument('--output_dir',
default='datapath/JPEG-quality-imagenet256/n01440764_Q=5/')
    parser.add_argument('--quality', default=5)
    args = parser.parse_args()
    print(args)
    input_dir = args.input_dir if args.input_dir.endswith('/') else
args.input_dir + '/'
    output_dir = args.output_dir if args.output_dir.endswith('/') else
args.output_dir + '/'
    return input_dir, output_dir, args.quality

def get_JPEG_img(img_dir):
    path_list = glob.glob(img_dir+'*.JPEG')
    print('Image directory: ', img_dir)
    print('Number of JPEG files: ', len(path_list))
    return path_list

img_dir, img_dir_new, quality = get_args()
img_path_list = get_JPEG_img(img_dir)

if not os.path.isdir(img_dir_new):
    os.makedirs(img_dir_new)
print('New image directory: ', img_dir_new)

for img_path in tqdm(img_path_list):
    image = io.imread(fname=img_path)
    img_path_new = img_path.replace(img_dir, img_dir_new).replace('.JPEG',
f"_Q={quality}.JPEG")
    # print(img_path_new)
    # exit()
    io.imsave(fname=img_path_new, arr=image, quality=quality)
```