2022春-机器学习大作业

小组成员：

- 王子悦 191830154
- 周辰熙 191250210

复现论文 **iCaRL: Incremental Classifier and Representation Learning**

1 论文概述

我们选择iCaRL: Incremental Classifier and Representation Learning 这篇论文进行复现。该论文提出了一种新的training strategy，使得模型可以动态地增加分类情况，进行持续学习。该论文使用**CIFAR-100**数据集。

1.1 分类

iCaRL依赖一组**从数据流中动态筛选的样本图片集**进行分类，每个已经被观测到的类别都存在一个样本集。iCaRL保证样本图像的总大小不会超过一个给定上限（**同样意味着当类别增加时，需要动态调整每个分类样本集的大小**）。

Algorithm 1 iCaRL CLASSIFY

input x // image to be classified
require $\mathcal{P} = (P_1, \dots, P_t)$ // class exemplar sets
require $\varphi : \mathcal{X} \to \mathbb{R}^d$ // feature map
  for $y = 1, \dots, t$ **do**
    $\mu_y \leftarrow \frac{1}{|P_y|} \sum_{p \in P_y} \varphi(p)$ // mean-of-exemplars
  end for
  $y^* \leftarrow \underset{y=1,\dots,t}{\operatorname{argmin}} \|\varphi(x) - \mu_y\|$ // nearest prototype
output class label y^*

iCaRL计算**待分类图像的特征**与现有**每个类特征**的距离，将输入图像分类至与其**距离最近**的一个类中，该分类方法被称作**Nearest-Mean-of-Exemplars Classification**。

1.2 训练

iCaRL将所有类别分批次地进行训练，每当新类（之前未出现过）的数据加入时，iCaRL调用更新函数对模型进行更新，调整模型内部的参数。同时由于类型的增加，需要将现有所有样本类数据集减小，以防止超过内存占用上限。

Algorithm 2 iCaRL INCREMENTALTRAIN

input $X^s, \ldots, X^t$ // training examples in per-class sets
input K // memory size
require Θ // current model parameters
require $\mathcal{P} = (P_1, \ldots, P_{s-1})$ // current exemplar sets
  $\Theta \leftarrow \text{UPDATEREPRESENTATION}(X^s, \ldots, X^t; \mathcal{P}, \Theta)$
  $m \leftarrow K/t$ // number of exemplars per class
  for $y = 1, \ldots, s-1$ **do**
    $P_y \leftarrow \text{REDUCEEXEMPLARSET}(P_y, m)$
  end for
  for $y = s, \ldots, t$ **do**
    $P_y \leftarrow \text{CONSTRUCTEXEMPLARSET}(X_y, m, \Theta)$
  end for
  $\mathcal{P} \leftarrow (P_1, \ldots, P_t)$ // new exemplar sets

1.3 架构

iCaRL使用卷积神经网络CNN作为模型进行训练，iCaRL并不直接使用CNN进行分类（即将输出层中具有最大输出的维度作为预测结果），而是将网络作为 *trainable feature extractor* （可训练的特征提取器）$\varphi : x \rightarrow R^d$，所有的输出特征都是$L^2 - normalized$的（$\vec{x} \cdot \vec{x} = 1$），iCaRL的输出为

$$g(x) = \frac{1}{1 + exp(-a_y(x))} \quad with \;\; a_y(x) = \omega_y^T \varphi(x)$$

网络的参数记为Θ，分为两个部分，**特征提取部分的参数**(φ中的参数)以及**每个类的权重向量**(ω_y^T：每个类y对应的权重向量)

iCaRL中网络的主要作用为representation learning

2 框架代码分析

框架代码负责了数据读取、预处理（图像裁剪、归一化）以及持续学习中的各种基础功能

```
# 对原图像84x84进行随机裁剪、随机翻转，转换至32x32的tensor格式数据并归一化
train_transform = transforms.Compose(
        [
            transforms.Resize((inp_size, inp_size)),
            transforms.RandomCrop(inp_size, padding=4),
            transforms.RandomHorizontalFlip(),
            transforms.ToTensor(),
            transforms.Normalize(mean, std),
```

```
        ]
    )

# 将模型的训练任务分成五个步骤、每次新增20个新类，共输入100种类别的数据
class_of_task = [20, 20, 20, 20, 20]

# 持续学习训练算法，需要实现的iCaRL算法正是替换这个实例
# method = Finetune(criterion, device, train_transform, test_transform,
class_of_task[0],        # n_classes, **kwargs)
method = iCaRL(criterion, device, train_transform, test_transform, class_of_task[0],
n_classes, **kwargs)


# 设置当前训练任务的数据集，训练前处理、进行训练以及训练后处理
method.set_current_dataset(cur_train_datalist, cur_test_datalist)
method.before_task(cur_train_datalist)
task_acc, eval_dict = method.train(cur_iter)
method.after_task(cur_iter)
```

3 参数设置

我们将100个类分为5个任务进行持续学习，即每个任务会新增20个之前没有的类，每个任务的epoch设为**30**，**memorysize**设置为2000（Exemplar_set中样本总数为2000）

4 算法复现

复现iCaRL算法主要需要实现论文中的 Algorithm 1（分类）、Algorithm 3 （更新网络）、Algorithm 4（构建Exemplar set）以及Algorithm 5（调整Exemplar set大小），

```
def after_task(self, cur_iter):
        # update the num_learned_class
        self.num_learned_class = self.num_learning_class
        # the size which each exemplar_set should be
        k = self.memory_size // self.num_learned_class

        ## first
        ## reduce the size of current exemplar_set
        logger.info("#"*15 + " reduce exemplar set " + "#"*15)
        self.reduce_exemplar_set(k)

        ## second
        ## construct the new exemplar_set
        logger.info("#"*15 + " construct exemplar set " + "#"*15)
        self.construct_exemplar_set(cur_iter, k)

        ## third
        ## caculate the mean feature representation for every class
        logger.info("#"*15 + " caculate exemplar mean " + "#"*15)
        self.caculate_exemplar_mean()

        ## finally
        ## use the exemplar_mean to classify the test data, and compute the accuracy
        logger.info("#"*10 + " evaluate using nearest mean exempalrs classification " +
"#"*10)
```

```
        self.nearest_mean_exemplars_classify()
```

这些算法主要在网络参数调整之后使用，因此在after_task中调用

4.1 Algorithm 1 分类

Algorithm 1 iCaRL CLASSIFY

input x // image to be classified
require $\mathcal{P} = (P_1, \ldots, P_t)$ // class exemplar sets
require $\varphi : \mathcal{X} \to \mathbb{R}^d$ // feature map
  for $y = 1, \ldots, t$ **do**
    $\mu_y \leftarrow \frac{1}{|P_y|} \sum_{p \in P_y} \varphi(p)$ // mean-of-exemplars
  end for
  $y^* \leftarrow \operatorname*{argmin}_{y=1,\ldots,t} \|\varphi(x) - \mu_y\|$ // nearest prototype
output class label y^*

```
def classify(self, x):
    res = []
    _x = F.normalize(self.model.featrue_extractor(x).detach()).cpu().numpy()
    exemplar_means = [value for _, value in self.exemplar_mean.items()]
    # 获取每个exemplar_set的平均特征
    exemplar_means = numpy.array(exemplar_means)
    for input in _x:
        # 计算输入样本和每个exemplar_set平均特征的差向量
        dif = input - exemplar_means
        # 计算每个差向量的范数 （长度平方）
        dist = numpy.linalg.norm(dif, ord=2, axis=1)
        # 将该样本分类至具有最短差向量的类中
        label = numpy.argmin(dist)
        res.append(label)
    return torch.tensor(res)
```

4.2 Algorithm 3 更新网络

Algorithm 3 iCaRL UpdateRepresentation

input $X^s, \dots, X^t$ // training images of classes $s, \dots, t$
require $\mathcal{P} = (P_1, \dots, P_{s-1})$ // exemplar sets
require Θ // current model parameters
// form combined training set:

$$\mathcal{D} \leftarrow \bigcup_{y=s,\dots,t} \{(x,y) : x \in X^y\} \cup \bigcup_{y=1,\dots,s-1} \{(x,y) : x \in P^y\}$$

// store network outputs with pre-update parameters:
for $y = 1, \dots, s-1$ **do**
$q_i^y \leftarrow g_y(x_i)$ for all $(x_i, \cdot) \in \mathcal{D}$
end for
run network training (*e.g.* BackProp) with loss function

$$\ell(\Theta) = -\sum_{(x_i,y_i)\in\mathcal{D}} \Big[\sum_{y=s}^{t} \delta_{y=y_i} \log g_y(x_i) + \delta_{y\neq y_i} \log(1-g_y(x_i)) + \sum_{y=1}^{s-1} q_i^y \log g_y(x_i) + (1-q_i^y) \log(1-g_y(x_i)) \Big]$$

that consists of *classification* and *distillation* terms.

```
# return the datalist struct which combined the exemplar and new datalist
    def combine_exemplar_and_current(self):
        datalist = []
        # datalist.append({'file_name': img_path, 'label': start_label})
        ## first, add all exemplar
        ## 首先，将所有exemplar_set中的数据加入datalist
        for key, value in self.exemplar_set.items():
            for img_path in value:
                datalist.append({'file_name':img_path, 'label':key})
        ## second, add all new data
        ## 再将所有新数据加入datalist
        ## 之后网络训练时使用的数据就是exempalr_set + newdata
        datalist = datalist + self.train_list
        return datalist
```

4.2 Algorithm 4 构建Exemplar set

Algorithm 4 iCaRL CONSTRUCTEXEMPLARSET

input image set $X = \{x_1, \dots, x_n\}$ of class y
input m target number of exemplars
require current feature function $\varphi : \mathcal{X} \to \mathbb{R}^d$
  $\mu \leftarrow \frac{1}{n}\sum_{x\in X}\varphi(x)$ // current class mean
  for $k = 1, \dots, m$ **do**
    $p_k \leftarrow \underset{x\in X}{\operatorname{argmin}} \left\| \mu - \frac{1}{k}[\varphi(x) + \sum_{j=1}^{k-1}\varphi(p_j)] \right\|$
  end for
  $P \leftarrow (p_1, \dots, p_m)$
output exemplar set P

```
## 传入当前的任务号（从0到4）以及每个Exemplar set的大小k
def construct_exemplar_set(self, cur_iter, k):
        ## every iteration, we train 20 new classes
        new_img_dataset = ImageDataset (
            pd.DataFrame(self.train_list),
            self.dataset,
            self.train_transform
        )
        for label in range(cur_iter * 20, cur_iter * 20 + 20):
            ## 获取某个label下所有图像构成的ImageDataSet
            img_set = ImageDataset(
                new_img_dataset.get_image_class(label),
                self.dataset,
                self.train_transform
            )
            ## select k imgs from the data set
            ## 挑选k个图像作为Exemplar Set的成员（是按顺序加入的，因此缩小数据集时只需截断即可）
            res = self.compute_k_nearest_img(img_set, k)
            ## add to the exemplar set
            self.exemplar_set[label] = res
## 挑选图像
def compute_k_nearest_img(self, dataset : ImageDataset, k):
        ## compute the mean of the current class
        ## and extract the feature for every imgs
        ## 先转为tensor格式
        data = dataset.to_tensor()
        class_mean, feature_output = self.compute_the_class(data)

        exemplar_list = []
        exemplar_sum = numpy.zeros((1, 512))

        ## select k imgs from dataset
        for i in range(k):
            ## compute the distance from each imgs to the class mean
            x = class_mean - (exemplar_sum + feature_output)/(i + 1)
            x = numpy.linalg.norm(x, axis=1)
            ## add the img into the exemplar
            idx = numpy.argmin(x)
            exemplar_sum += feature_output[idx]
            exemplar_list.append(dataset[idx]['image_name'])

        return exemplar_list
```

4.3 Algorithm 5 调整Exemplar set的大小

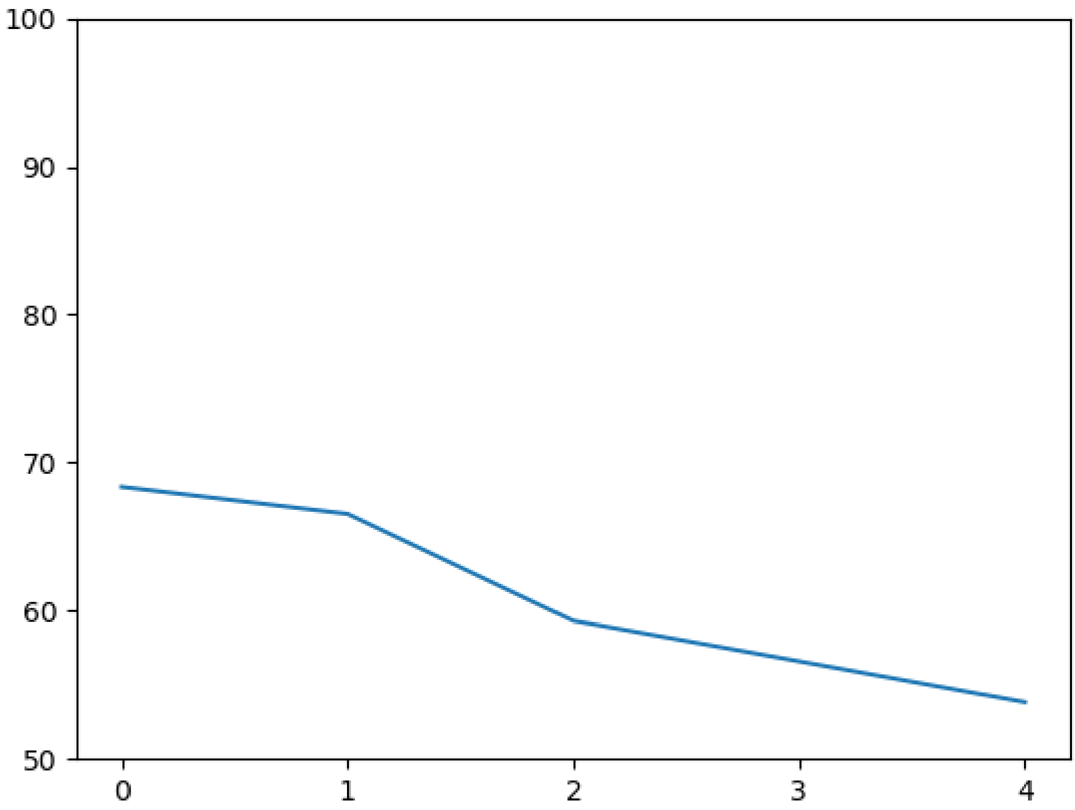
Algorithm 5 iCaRL REDUCEEXEMPLARSET

input m // target number of exemplars
input $P = (p_1, \ldots, p_{|P|})$ // current exemplar set
$P \leftarrow (p_1, \ldots, p_m)$ // *i.e.* keep only first m
output exemplar set P

```
def reduce_exemplar_set(self, k):
    for key, value in self.exemplar_set.items():
        ## 由于加入时就是按顺序加入的，因此直接取前k个即可
        self.exemplar_set[key] = value[:k]
```

5 实验结果

训练过程的日志记录在logs/tmp.log中，训练时间约为1小时18分钟



每次任务过后的分类准确率为 68.35% , 66.525%, 59.317%, 56.588%, 53.84%