배치 샘플링을 이용한 데이터셋 응축

요 약

데이터셋 응축은 거대한 학습 데이터셋을 작은 데이터셋으로 응축해 모델 학습의 부담을 줄이고자 제안되었고, 분포를 학습하는 데이터셋 응축 연구는 가중치를 이용한 데이터셋 응축보다 높은 성능을 보여준다. 본 연구에서는 샘플의 특징을 기반으로 학습에 사용되는 배치를 구성해 가중치를 이용한 데이터셋 응축 성능을 개선하고자 하였다. 본 연구에서는 CIFAR10 데이터셋과 Convolutional Neural Network(CNN)을 이용하여 데이터셋 응축을 수행한다. Batch Sampling 을 이용한 데이터셋 응축은 성능 향상을 보이지 못하며 Matching Loss 가 증가하는 현상을 보였다. 가중치를 이용한 데이터셋 응축의 성능은 가중치의 L2 Norm 이 클수록 성능 하락을 보인다. 다만 Matching Loss 가 상대적으로 큼에도 불구하고 유사한 테스트 정확도를 보이기 때문에 Matching Loss 의 역할에 대해 추가적인 연구가 필요하다.

1. 서론

1.1. 연구배경

Deep Neural Network 를 시작으로 여러 기계학습 모델은 이미지 처리, 음성 인식 등 여러 분야에서 최고의 성능을 보여주며 이들이 사용되는 분야 또한 확대되고 있다. 데이터셋은 기계학습을 다양한 분야에 적용할 수 있도록 뒷받침하며 그 질과 양이 기계학습 기술과 함께 발전하고 있다. 특히, 데이터의 질과 양이 모델 성능에 큰 영향을 미치기 때문에 높은 성능을 내기 위해 방대한 데이터셋을 여러 분야에서 활용하고 있다[1]. 용량이 큰 데이터셋은 저장하는 것도 어려울 뿐만 아니라 이를 활용하기 위해 높은 컴퓨팅 성능을 요구한다. 결국 제한된 컴퓨팅 자원으로 사용할 수 있는 데이터의 종류와 양은 한계가 존재하고 이는 평생학습(Continual Learning)[2], Neural Architecture Search(NAS)[3]와 같이 다양한 데이터가 요구되는 분야에서 어려움으로 나타난다.

데이터셋 응축(Dataset Condensation)은 제한된 저장 용량과 컴퓨팅 자원으로 학습 데이터의 효율을 극대화하기 위해 원본 데이터셋의 학습 정확도와 유사하도록 아주 작은 데이터셋(e.g. 10 images/class)을 생성하는 분야이다. Zhao et al[4]에서는 응축한 데이터가 실제 데이터와 유사한 가중치를 발생시키도록 합성하는 방법이 처음 제안되었고 이를 토대로 다양한 데이터셋 응축 연구가 활발히 진행되고 있다. CAFÉ[5]는 가중치에서 나아가 원본 데이터셋으로부터 학습된

특징을 응축하도록 모델을 설계했고 가장 높은 정확도를 보여준다. 이 방법으로 응축된 샘플들은 기존 방법보다 넓은 데이터 공간에 분포되어 있기 때문에 원본 데이터셋의 핵심 특징을 충분히

그림 1. 응축 샘플과 샘플의 데이터 공간 분포를 시각화했다. (좌) DC[-]이고 (우) CAFE이다. CAFE는 DC보다 데이터 공간에서 더 넓게 분포되어 있고 다양한 응축 샘플을 생성해낸다.

포착할 수 있다는 장점이 있다. 본 연구는 원본 데이터셋의 학습 특징을 충분히 학습할 수 있도록 유사한 특징들로 학습 배치를 구성해 데이터셋 응축을 수행하는 방법을 제안한다.

1.2. 연구목표

본 연구는 가중치를 학습하는 기존 데이터셋 응축 방법에 간단한 배치 샘플링을 더해 원본 데이터셋의 분포를 충분히 포착할 수 있도록 학습 방법을 개선한다. 나아가 가중치를 이용한 데이터셋 응축 방법에 대해 가중치가 응축 데이터셋의 성능에 미치는 영향을 분석하고, 임의의 배치와 배치 샘플링을 통해 발생되는 가중치를 분석한다. 이를 통해 응축 데이터셋의 성능에 결정적인 영향을 주는 요인을 분석하고 향후 데이터셋 응축 연구에 활용한다.

데이터셋 응축은 그 가치가 데이터 효율성 증가에도 있지만 원본 데이터 보호, 학습 시 사용되는 에너지 절약 등 다양한 사회적 가치를 갖는다. 본 연구를 통해 응축한 데이터셋이 갖는 사회적 가치를 분석하여 응축 데이터셋의 효용을 살펴본다.

2. 관련연구

Wang et al. (2018)[6]은 처음으로 학습 가능한 데이터셋 응축 제안하며 이을 이중 수준 최적화(Bi-Level Optimization) 문제로 설계하여 유의미한 결과를 얻었다. Zhao et al. (2021)[4]는 이전 연구를 확장해 응축된 이미지 샘플들이 원본 데이터와 유사한 학습 과정을 만들어내도록 합성하는 Dataset Condensation(DC)를 제안했다. DC 는 Gradient Matching Loss 를 사용해 원본 데이터셋으로 학습할 때 발생되는 가중치와 응축 데이터셋으로 학습할 때 발생되는 가중치가 같도록 응축 데이터셋을 학습한다. Zhao & Bilen(2021)[7]은 같은 학습 방법에 미분 가능한 데이터 증강 기법을 적용하여 데이터 효율성을 극대화하는 방법을 제안했다.

한편 응축 데이터셋이 원본 데이터셋과 유사한 데이터 공간에 위치하도록 학습하는 Distribution Matching(DM)[8]은 기존 DC 보다 성능이 낮은 경우(i.e. MNIST)가 있지만 연산이 줄어

ImageNet 까지 연구를 확장할 수 있었다. DM 은 응축 데이터셋만 학습하면 되는 반면 DC 는 응축 데이터셋과 이미지 특징을 포착하는 모델을 동시에 학습해야 하기 때문이다. DM 과 유사하게 CAFÉ[5]는 원본 데이터 분포 내에서 최대한 다양한 샘플을 생성하도록 discriminative loss 를 사용한 데이터셋 응축을 제안했다. Kim et al. (2022)[9]는 하나의 응축 샘플에 데이터 증강을 적용해 여러개의 응축 샘플을 생성하고 이를 사용하여 데이터셋 응축을 수행한다. 데이터 증강을 통해 여러개의 응축 샘플로 데이터셋 응축을 수행하면 Gradient Matching Loss 가 크게 줄어든다는 관찰에 기반하였다. Lee et al (2022)[10]은 이와 유사하게 학습 가능한 은닉 공간에서의 *code* 와 간단한 *decoder* 를 사용해 더 다양한 응축 이미지를 생성해낸다. 이외에도 Infinite Wide Neural Network 와 Kernel Ridge Regression 을 이용해 데이터셋을 응축하지만 수백대의 GPU 를 요구한다는 한계점이 있다[11]. 본 연구는 데이터셋 응축에 사용하는 배치를 유사한 특징을 갖는 원본 샘플들로 구성하여 모델이 이미지 특징을 보다 수월하게 포착할 수 있을 것으로 기대한다.

3. 프로젝트 내용

3.1. Batch Sampling

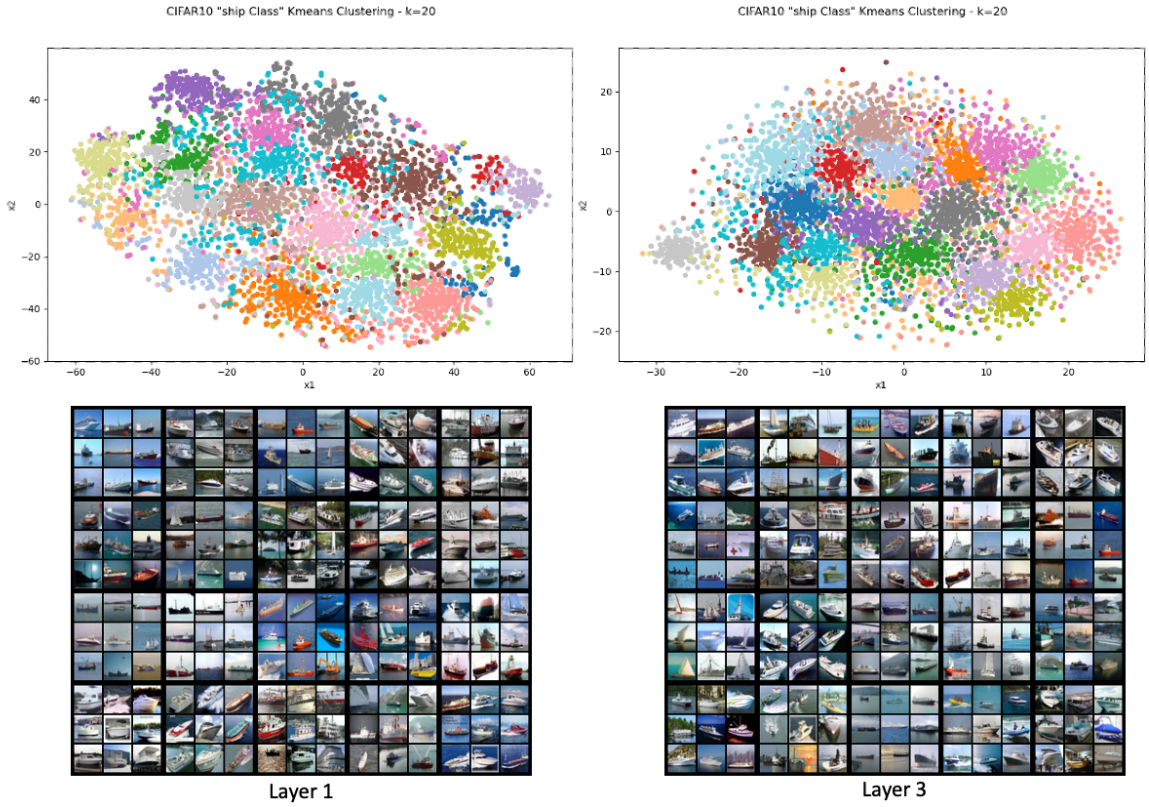

그림 2. CIFAR10 데이터셋의 "Ship" 클래스 이미지 샘플을 K-Means(k=20)으로 클러스터링 한 결과. (좌) Layer 1 특징맵을 이용 (우) Layer 3 특징맵을 이용. Layer3의 클러스터링 결과는 Layer1보다 가까운 데이터 공간에 밀집되어 있는 것을 확인할 수 있다. Layer1은 이미지의 배경 정보가 클러스터링에 많은 영향을 끼치는 것에 비해 Layer3는 배경 정보는 서로 다르지만 배의 형태, 위치, 형상이 유사한 것을 확인할 수 있다

배치 샘플링(Batch Sampling)은 사전에 유사한 특징을 갖는 샘플들을 선별해서 데이터셋 응축 학습에 사용될 학습 배치를 구성하는 방법이다. 본 연구에서는 ImageNet 으로 사전학습된 ResNet18 모델을 CIFAR10 데이터셋에 fine-tuning 을 수행한 뒤 특징 추출기로 사용한다. 또한 CIFAR10 샘플들은 어떤 특징을 포함하고 있는지에 대한 정보가 없기 때문에 특징을 기반으로 클러스터링하기 위해서는 unsupervised 방법을 사용한다.

ResNet18 은 4 개의 Convolutional Block 과 각 클래로 분류하는 Classifier 로 구성되어 있다. 이미지를 모델에 통과시켜 4 개의 특징맵(Feature Map)을 획득하고, 이를 기반으로 K-Means 클러스터링을 수행한다. 모델의 각 레이어들은 유사한 입력 신호를 가까운 특징 공간으로 매핑(mapping)하는 역할을 수행하기 때문에 가까운 데이터 공간에 위치한 입력 샘플들은 유사한 특징을 갖고 있다. 그러나 [그림 2]에서 확인할 수 있듯이 각 레이어들은 포착하는 특징이 다르다. 본 연구에서는 각 레이어로부터 클러스터링한 샘플을 통해 데이터셋 응축을 수행했을 때의 결과를 자세하게 분석한다.

결국 배치 샘플링을 사용하여 각 클러스터로부터 하나의 학습 배치를 구성하면 데이터셋 응축 시 유사한 이미지 특징을 포착할 수 있고, 기존 Gradient Matching 만 사용하여 데이터셋 응축을 수행하는 모델이 원본 데이터셋의 넓은 데이터 공간을 탐색할 수 있도록 도울 수 있다.

3.2. 모델 구조와 Loss 함수

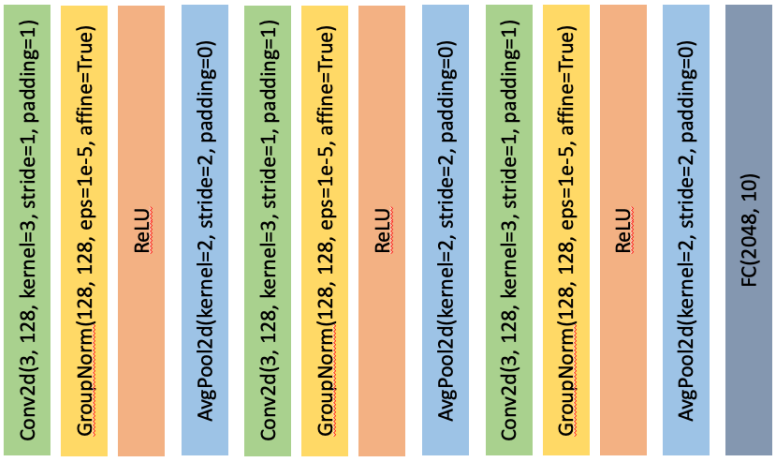

그림 3. 본 연구에서 활용하는 ConvNet. 3개의 Convolutional Block과 1개의 Fully Connected Layer(FC)로 연결되어 있다. ConvNet은 DC[-], DSA[-] 등에서 사용되는 모델이기에 성능 비교를 위해 해당 모델을 선택했다.

본 연구에서 사용하는 모델은 [그림 3]의 ConvNet 이다. ConvNet 은 기존 DC[4], DSA[7]와 같은 연구 방법에서 사용되었고 좋은 성능을 보여주었다. 본 연구에서는 DC 와 DSA 를 baseline 으로 정하여 성능을 비교하기에 해당 모델을 사용한다.

$$d(A,B) = \sum_{i=1}^{out}\left(1 - \frac{A_i \cdot B_i}{\|A_i\|\|B_i\|}\right) \tag{1}$$

본 연구에서는 DC[4], DSA[7]와 같은 Gradient Matching Loss 를 사용한다. 원본 데이터셋 T를 입력으로 사용했을 때 얻을 수 있는 레이어 별 가중치를 L^T 라고 하고 응축 데이터셋 S 를 입력으로 사용했을 때 얻을 수 있는 레이어 별 가중치를 L^S라고 하자. Gradient Matching Loss $D(\cdot,\cdot)$은 L^T와 L^S의 거리를 나타낸다. Gradient Matching Loss 는 다음과 같이 레이어 별 loss 로 분해할 수 있다; $D(L^T, L^S) = \sum_{l=1}^{L} d(\nabla_{\theta^{(l)}} L^T, \nabla_{\theta^{(l)}} L^S)$. 이 때, l은 레이어의 인덱스이고 L은 가중치를 갖는 레이어의 수이고 $d(\cdot,\cdot)$ 은 식 (1)과 같다. 식(1)에서 A_i 와 B_i 는 i 번째 출력 노드에 대한 가중치이다.

3.3. 응축 데이터셋 학습

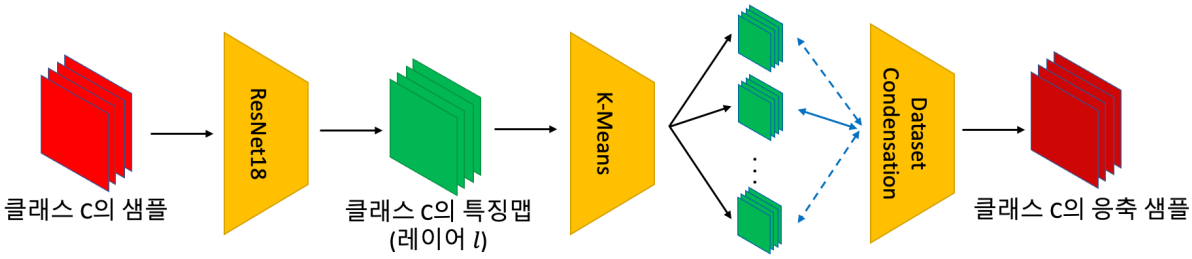

그림 4. 본 연구에서 제안하는 데이터셋 응축 수행 방법. 각 클래스로부터 특징을 추출한 후 K-Means를 통해 k개의 클러스터로 묶는다. 이후 각 클러스터로부터 n개의 샘플을 추출해 배치를 구성하고 해당 배치로 데이터셋 응축을 수행한다.

본 연구에서 제안하는 데이터셋 응축은 [그림 4]와 같이 수행한다. 먼저 3.1 에서 수행한 배치 샘플링을 바탕으로 같은 이미지 특징을 기반으로 구성된 서로 다른 k 개의 클러스터를 구한다. 이후 각 클래스의 임의의 클러스터로부터 n 개의 샘플들을 랜덤으로 선택해 배치를 구성한다. 기존 방법은 각 클래스의 샘플들 전체로부터 n 개의 샘플들을 랜덤으로 선택한다. 해당 방법을 통해 데이터셋 응축을 수행하면 원본 데이터셋 데이터 분포에서 기존 방법보다 넓은 영역의 샘플들을 합성하도록 학습하는 효과를 기대한다.

3.4. 코드 구조

- main.py : Baseline(DC, DSA)를 수행하기 위한 코드
- main_BS.py : 제안 방법을 데이터셋 응축에 결합하여 수행하기 위한 코드
- main_cluster.py : Batch Clustering 을 수행하는 코드
- networks.py : 데이터셋 학습에 사용되는 모델 구현 코드
- utils.py : 데이터셋 응축에 사용되는 여러 함수들(환경설정, 학습 등)

3.4.1. 클러스터링 수행

1. CIFAR10 데이터셋을 불러온 후, 각 클래스 별로 정리한다. 이 때, 데이터 증강은 적용하지 않는다.

```
''' organize the real dataset '''
mean = [0.4914, 0.4822, 0.4465]
std = [0.2023, 0.1994, 0.2010]
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize(mean=mean, std=std)])
dst_train = datasets.CIFAR10(args.data_path, train=True, download=True, transform=transform) # no augmentation
class_names = dst_train.classes

images_all = []
labels_all = []
indices_class = [[] for c in range(10)]

images_all = [torch.unsqueeze(dst_train[i][0], dim=0) for i in range(len(dst_train))]
labels_all = [dst_train[i][1] for i in range(len(dst_train))]
for i, lab in enumerate(labels_all):
    indices_class[lab].append(i)
images_all = torch.cat(images_all, dim=0).to(device)
labels_all = torch.tensor(labels_all, dtype=torch.long, device=device)

def get_images(c, n): # get random n images from class c
    idx_shuffle = np.random.permutation(indices_class[c])[:n]
    return images_all[idx_shuffle]
```

2. ResNet18 모델을 불러와 pretrained weight 를 적용한다.

```
# Model
net = ResNet18(3, 10)
state_dict = torch.load(os.path.join(args.cluster_path, "resnet18_cifar.pt"), map_location='cpu')
net.load_state_dict(state_dict)
net = net.to(device)
net.eval()
```

3. 클래스 별로 클러스터링을 수행한다. 원하는 레이어에 대한 특징맵을 선택하여 K-Means 를 수행한다. 본 연구에서는 KmeansConstrained 를 사용했는데, 이는 각 클러스터의 샘플 수를 일정 범위 내로 유지한다.

```
for class_idx in range(10):
    class_name = class_names[class_idx]
    batch_size = int(len(indices_class[class_idx]) / args.iteration)
    cluster_size = int(len(indices_class[class_idx]) / args.num_cluster)

    features = []
    print(f"Gathering features for class {class_name}")
    for it in range(args.iteration):
        imgs = images_all[indices_class[class_idx][it*batch_size:(it+1)*batch_size]]
        _, _feature = net(imgs)
        features.append(F.avg_pool2d(_feature[args.layer_idx], (_feature[args.layer_idx].shape[2],_feature[args.layer_idx].shape[3])).squeeze())
    features = torch.cat(features, dim=0).detach().cpu().numpy()

    print(f"Calculating cluster center features for class {class_name}")
    if args.norm:

        clf = KMeansConstrained(n_clusters=args.num_cluster, size_min=cluster_size-50, size_max=cluster_size+50, random_state=1)
        clf.fit(features)
        labels = clf.labels_
        centers = clf.cluster_centers_
    else:
        kmeans = KMeans(n_clusters=args.num_cluster, n_init=5, max_iter=300, random_state=1, verbose=0).fit(features)
        labels = kmeans.labels_
        centers = kmeans.cluster_centers_
```

3.4.2. 응축 데이터셋 학습

1. CIFAR10 데이터셋을 불러와 클래스 별로 정리한다. 또한 3.4.1 에서 수행한 클러스터링 결과를 불러와 sub_labels_all 이라는 python dictionary 에 저장한다. 이는 *"get_batches(c, n)"* 함수에서 각 클래스 내 랜덤한 클러스터에서 n 개의 샘플을 선택할 때 사용된다.

```
''' organize the real dataset '''
images_all = []
labels_all = []
indices_class = [[] for c in range(num_classes)]

images_all = [torch.unsqueeze(dst_train[i][0], dim=0) for i in range(len(dst_train))]
labels_all = [dst_train[i][1] for i in range(len(dst_train))]
sub_labels_all = torch.load(os.path.join(args.cluster_path, f'class_idx_cifar10_k{args.num_cluster}_{args.layer_idx}_{str(args.norm)}.pt'))
for i, lab in enumerate(labels_all):
    indices_class[lab].append(i)
images_all = torch.cat(images_all, dim=0).to(args.device)
labels_all = torch.tensor(labels_all, dtype=torch.long, device=args.device)

for c in range(num_classes):
    print('class c = %d: %d real images'%(c, len(indices_class[c])))

def get_images(c, n): # get random n images from class c
    idx_shuffle = np.random.permutation(indices_class[c])[:n]
    return images_all[idx_shuffle]

def get_batches(c, n):
    sub_class = np.random.randint(args.num_cluster)
    sub_class_indices = sub_labels_all[c][sub_class]
    idx_shuffle = np.random.permutation(sub_class_indices)[:n]
    return images_all[idx_shuffle]
```

2. 합성 데이터셋을 초기화한다. 본 연구는 Uniform Distribution 으로 초기화를 수행했다. 설정값을 통해 원본 데이터셋으로 초기화를 할 수 있다.

```
''' initialize the synthetic data '''
image_syn = torch.randn(size=(num_classes*args.ipc, channel, im_size[0], im_size[1]), dtype=torch.float, requires_grad=True, device=args.device)
label_syn = torch.tensor([np.ones(args.ipc)*i for i in range(num_classes)], dtype=torch.long, requires_grad=False, device=args.device).view(-1)

if args.init == 'real':
    print('initialize synthetic data from random real images')
    for c in range(num_classes):
        image_syn.data[c*args.ipc:(c+1)*args.ipc] = get_images(c, args.ipc).detach().data
else:
    print('initialize synthetic data from random noise')
```

3. 합성 이미지와 학습 모델에 대한 optimizer 를 설정한다. 본 연구에서 사용하는 데이터셋 응축은 Bi-Optimization Problem 으로서 합성 데이터셋을 학습하면서 jointly 하게 모델을 응축 데이터셋을 사용해 학습한다.

```
optimizer_img = torch.optim.SGD([image_syn, ], lr=args.lr_img, momentum=0.5)
optimizer_img.zero_grad()
criterion = nn.CrossEntropyLoss().to(args.device)

net = get_network(args.model, channel, num_classes, im_size).to(args.device)
net.train()
net_parameters = list(net.parameters())
optimizer_net = torch.optim.SGD(net.parameters(), lr=args.lr_net)
optimizer_net.zero_grad()
```

4. 응축 데이터셋과 모델을 학습한다. 각 클래스에 대한 원본 이미지와 합성 이미지를 모델에 입력하고, 각 샘플들로부터 발생하는 gradient 가 같도록 모델을 학습한다.

```
for c in range(num_classes):
    img_real = get_images(c, args.batch_real) if args.mix and it > 300 else get_batches(c, args.batch_real)
    lab_real = torch.ones((img_real.shape[0],), device=args.device, dtype=torch.long) * c
    img_syn = image_syn[c*args.ipc:(c+1)*args.ipc].reshape((args.ipc, channel, im_size[0], im_size[1]))
    lab_syn = torch.ones((args.ipc,), device=args.device, dtype=torch.long) * c

    if args.dsa:
        seed = int(time.time() * 1000) % 100000
        img_real = DiffAugment(img_real, args.dsa_strategy, seed=seed, param=args.dsa_param)
        img_syn = DiffAugment(img_syn, args.dsa_strategy, seed=seed, param=args.dsa_param)

    output_real, _ = net(img_real)
    loss_real = criterion(output_real, lab_real)
    gw_real = torch.autograd.grad(loss_real, net_parameters)
    gw_real = list((_.detach().clone() for _ in gw_real))
    gw_real_sums.append(np.array(list(map(norm, gw_real))))

    output_syn, _ = net(img_syn)
    loss_syn = criterion(output_syn, lab_syn)
    gw_syn = torch.autograd.grad(loss_syn, net_parameters, create_graph=True)
    gw_syn_sums.append(np.array(list(map(norm, gw_syn))))

    loss += match_loss(gw_syn, gw_real, args)
gw_real_log.append(np.mean(np.vstack(gw_real_sums), axis=0))
gw_syn_log.append(np.mean(np.vstack(gw_syn_sums), axis=0))

optimizer_img.zero_grad()
loss.backward()
optimizer_img.step()
```

5. 응축 데이터셋의 성능 평가를 수행한다. 응축 데이터셋으로 300 epoch 을 학습시키고 정확도를 측정한다. 이 과정을 20 번 반복한다.

```
print('-------------------------\nEvaluation\nmodel_train = %s, model_eval = %s, iteration = %d'%(args.model, model_eval, it))
if args.dsa:
    args.epoch_eval_train = 1000
    args.dc_aug_param = None
    print('DSA augmentation strategy: \n', args.dsa_strategy)
    print('DSA augmentation parameters: \n', args.dsa_param.__dict__)
else:
    args.dc_aug_param = get_daparam(args.dataset, args.model, model_eval, args.ipc)
    print('DC augmentation parameters: \n', args.dc_aug_param)

if args.dsa or args.dc_aug_param['strategy'] != 'none':
    args.epoch_eval_train = 1000
else:
    args.epoch_eval_train = 300

accs = []
for it_eval in range(args.num_eval):
    net_eval = get_network(model_eval, channel, num_classes, im_size).to(args.device)
    image_syn_eval, label_syn_eval = copy.deepcopy(image_syn.detach()), copy.deepcopy(label_syn.detach())
    _, acc_train, acc_test = evaluate_synset(it_eval, net_eval, image_syn_eval, label_syn_eval, testloader, args)
    accs.append(acc_test)
print('Evaluate %d random %s, mean = %.4f std = %.4f\n-------------------------'%(len(accs), model_eval, np.mean(accs), np.std(accs)))
```

4. 프로젝트 결과

실험은 CIFAR10 데이터셋과 교내 Moana 서버를 이용하여 수행했다. 모든 실험의 결과값은 100 번의 테스트 정확도의 평균과 표준편차를 측정한 값이다. 본 실험에서는 클러스터링을 위해 ImageNet 으로 사전학습된 ResNet18 모델과 ConvNet 모델을 사용했다. 실험에 사용된 baseline 은 DC[4]와 DSA[7]이고, 발표된 정확도를 이용한다.

4.1. 데이터셋 응축 성능 비교

IPC	DC[4]	DSA[7]	Ours
1	28.3 ± 0.5	28.8 ± 0.7	28.1 ± 0.6
10	44.9 ± 0.5	52.1 ± 0.5	42.8 ± 0.6
50	53.9 ± 0.5	60.6 ± 0.5	52.9 ± 0.7

표 1. CIFAR10 데이터셋 응축 성능. ConvNet을 이용하여 응축하여 ConvNet을 300 epoch 동안 학습하여 얻은 테스트 정확도. 위 실험에서 사용된 Batch Sampling은 클러스터 개수 20개, 레이어 3의 특징을 사용했다. IPC는 Images/Class이다.

그림 5. CIFAR10 데이터셋을 10IPC로 응축한 결과. (좌) DC의 응축 결과 (우) Batch Sampling(k=20, Layer=3)의 응축 결과. 두 결과 모두 육안으로 어떤 클래스인지 구분이 가능하다. Batch Sampling으로 응축된 데이터셋은 DC에 비해 중복되는 특징이 덜 나타난다.

위 실험은 본 연구에서 제안하는 Batch Sampling 의 효과를 살펴보기 위해 기존 방법들과 응축 성능을 비교한다. 3 번째 레이어에서 추출한 특징맵을 이용해 20 개의 클러스터를 구성하여 Batch

Sampling 을 수행했다. DC 의 정확도와 아주 근사한 결과를 보여주지만 정확도 향상은 일어나지 않는다. 응축 데이터셋을 육안으로 비교해봤을 때 Batch Sampling 에서 합성한 샘플은 중복된 이미지 특징이 DC 보다 적게 나타나는 것을 확인할 수 있다.

4.2. 클러스터 개수에 따른 성능 변화

IPC	5	10	15	20	25	DC
1	28.0 ± 0.6	27.6 ± 0.5	28.1 ± 0.6	28.1 ± 0.6	$\mathbf{28.6 \pm 0.7}$	$\mathbf{28.3 \pm 0.5}$
10	42.6 ± 0.7	43.1 ± 0.4	$\mathbf{43.1 \pm 0.6}$	42.8 ± 0.6	42.8 ± 0.4	$\mathbf{44.9 \pm 0.5}$
50	$\mathbf{53.5 \pm 0.5}$	53.3 ± 0.6	52.7 ± 0.5	52.9 ± 0.7	53.3 ± 0.6	$\mathbf{53.9 \pm 0.5}$

표 2. Batch Sampling에서 클러스터 개수에 따른 정확도 비교 결과. 위 실험은 CIFAR10 데이터셋, ConvNet, Layer 2에서 추출한 특징맵을 기반으로 클러스터링을 수행 후 데이터셋 응축을 수행했다.

본 실험에서는 클러스터 개수에 따른 정확도 변화를 살펴본다. Baseline 은 DC 의 실험 결과이다. 실험에는 Layer 2 에서 추출한 특징을 기반으로 클러스터링을 수행하고 데이터셋 응축을 수행했다. 각 IPC 마다 정확도가 가장 높은 클러스터 개수가 다르고 그 차이가 1pp 까지 벌어진다. 한편 IPC 가 10 일 경우에는 그 편차가 0.5pp 로 매우 작아 클러스터의 개수는 데이터셋 응축에 미치는 영향이 크지 않다.

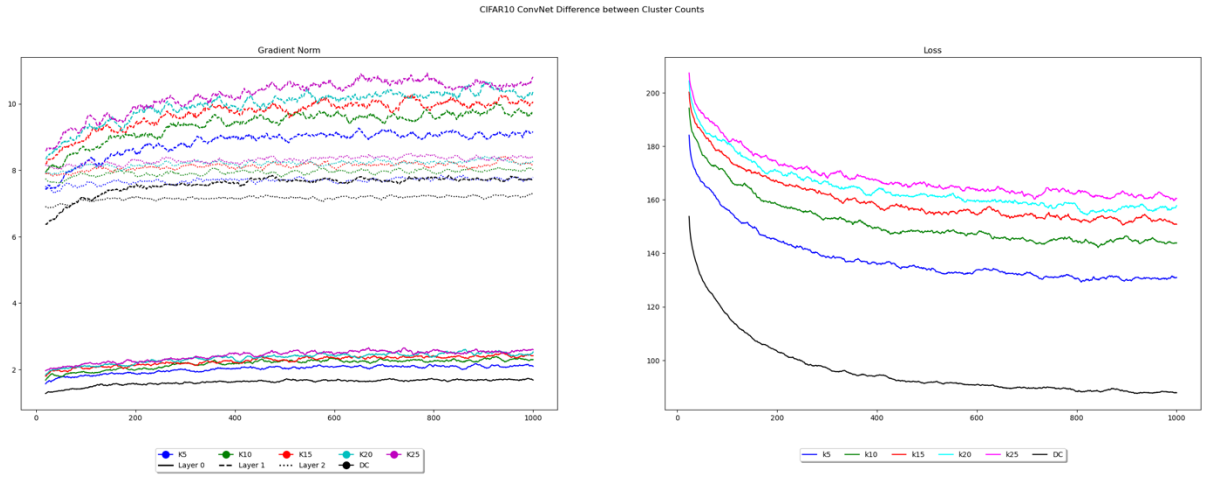

그림 6. CIFAR10 데이터셋, ConvNet, 10IPC, 그리고 Layer 2에서 추출한 특징을 바탕으로 클러스터 개수에 따른 가중치 L2 Norm과 Loss를 비교했다.

클러스터 크기에 대해 보다 자세히 살펴보기 위해 학습 중 발생하는 가중치의 L2 Norm 과 Loss 를 비교한다. 가중치는 DC 다 크게 모든 레이어에서 크게 발생하는 것을 관찰할 수 있고, Loss 또한 모든 클러스터 개수에서 크게 발생한다. 이를 통해 클러스터 개수가 적을수록 가중치가 가장 작게 발생하고 Loss 또한 가장 작다.

정확도와 가중치, Loss 를 살펴본 결과 DC 에서 변화가 적을수록 좋은 수치를 보여주는 경향이 있다. IPC 가 늘어남에 따라 정확도는 클러스터 개수가 적은 것이 높은 정확도를 보이고, Gradient 와 Loss 또한 줄어들고 있다.

4.3. 특징 추출 레이어에 따른 성능 변화

IPC	Layer 0	Layer 1	Layer 2	Layer 3	DC
1	27.8 ± 0.6	$\mathbf{28.3 \pm 0.5}$	28.0 ± 0.5	28.1 ± 0.6	$\mathbf{28.3 \pm 0.5}$
10	$\mathbf{43.6 \pm 0.6}$	42.7 ± 0.6	43.0 ± 0.5	42.8 ± 0.6	$\mathbf{44.9 \pm 0.5}$
50	52.0 ± 0.7	52.8 ± 0.4	52.6 ± 0.5	$\mathbf{52.9 \pm 0.7}$	$\mathbf{53.9 \pm 0.5}$

표 3. Batch Sampling에서 특징맵을 추출하는 Layer에 따른 정확도 비교. CIFAR10 데이터셋, ConvNet, 그리고 클러스터 개수는 20개이다.

본 실험에서는 특징 추출 레이어에 따른 정확도 변화를 살펴본다. Baseline 은 DC 의 실험 결과이다. 실험에는 각 레이어에서 추출한 특징을 기반으로 클러스터링($k = 20$) 을 수행하고 데이터셋 응축을 수행했다. 정확도는 Baseline 과 같거나 1pp 낮은 수치를 보이고 각 IPC 에 해당하는 정확도 차이도 1pp 로 나타난다. 앞선 클러스터 개수에 따른 정확도 변화보다 레이어에 따른 정확도 변화가 큰 것으로 나타나지만 레이어의 위치가 큰 영향을 미치는 것으로 보기는 어렵다.

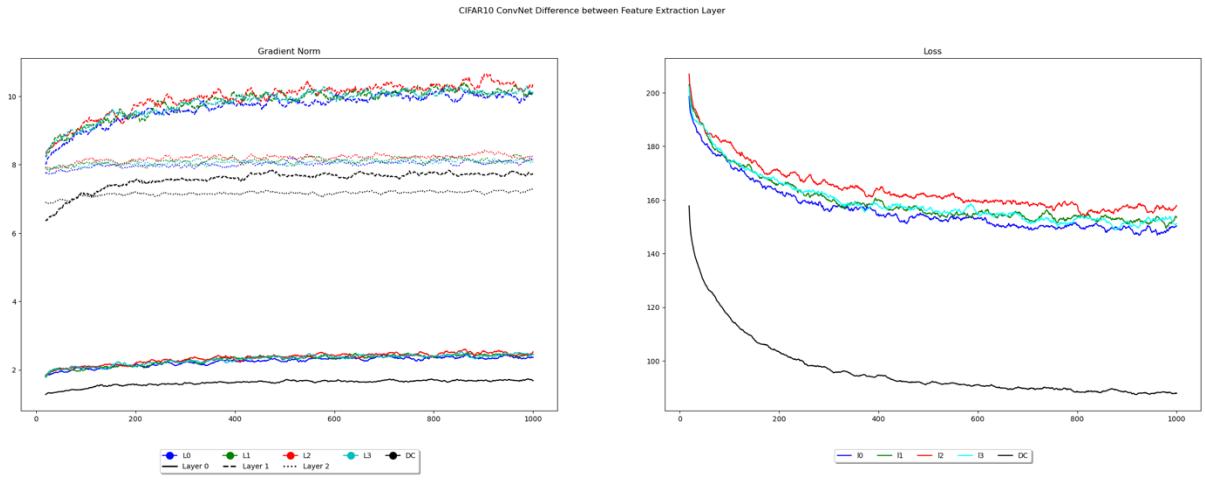

그림 7. CIFAR10 데이터셋, ConvNet, 클러스터 개수 20개로 데이터셋 응축을 수행한 결과. 특징 추출 레이어에 따른 가중치 변화와 Loss 변화이다.

특징 추출 레이어에 따른 가중치 L2 Norm 변화와 Loss 변화를 살펴본다. 가중치의 L2 Norm 과 Loss 역시 DC 보다 높게 나타난다. 한편 특징 추출 레이어 간 가중치 L2 Norm 은 비슷한 수준으로 발생하고 Loss 또한 큰 차이 없이 진행된다. 따라서 어떤 레이어에서 특징을 추출했는지는 Batch Sampling 을 이용한 데이터셋 응축에 큰 영향을 미치지 않고 4.3 에서 보인

클러스터 개수에 따른 차이가 보다 큰 영향을 미치는 것을 알 수 있다. 즉, ResNet18 모델이 CIFAR10 데이터셋에 대해 특징 추출을 모든 레이어에서 유사한 수준으로 수행한 것을 알 수 있다. 다만 0 번째 레이어에서 추출한 특징이 학습 측면에서 가장 낮은 Loss 를 보여주기 때문에 데이터셋 응축에는 큰 공간적 특징(후반 레이어에서 포착하는 특징)보단 작은 공간적 특징(예. 엣지 등)이 더 큰 영향을 주는 것을 알 수 있다.

4.4. 배치 샘플링이 학습에 미치는 효과 분석

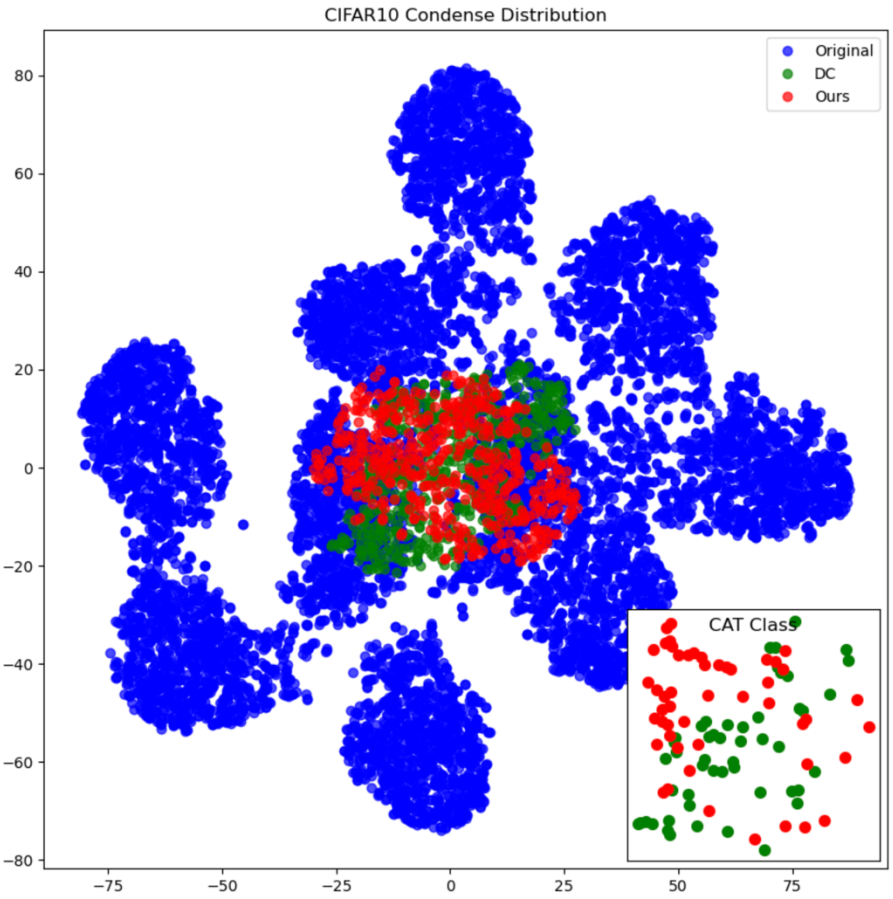

그림 8. CIFAR10 데이터셋을 DC와 Batch Sampling(k=20, Layer=3)를 사용하여 응축한 데이터셋의 분포. DC와 제안한 방법이 유사한 분포를 갖는다.

본 연구는 Batch Sampling 을 통해 원본 데이터셋을 학습할 때 특정 이미지 특징에 대해 큰 가중치를 발생시켜 응축 데이터셋 합성 시 이미지 특징을 보다 빠르게 학습하고 원본 데이터셋 데이터 공간에 대해 넓은 탐색을 할 수 있도록 설계했다. 그러나 [그림 8]을 볼 때 Batch Sampling 의 효과는 크지 않은 것 같다. CIFAR10 데이터셋의 CAT 클래스의 경우 Baseline 인 DC 와 유사한 데이터 분포를 보이고 있다. 또한 가중치를 이용한 데이터셋 응축에서 가중치의 L2 Norm 이 크다고 좋은 학습 결과로 이어지지 않는 것을 알 수 있다. 강한 가중치는 overfitting 을 유발할 수 있고, Loss 를 볼 때 학습에 도움을 주지 못하는 것을 알 수 있다.

5. 결론 및 기대효과

본 연구를 통해 유사한 이미지 특징을 가진 샘플들을 모아 배치를 구성하여 데이터셋 응축에 사용하는 Batch Sampling 을 이용한 데이터셋 응축은 기존 실험에 대해 성능 개선을 이루지 못했다. 먼저, 가중치를 이용한 데이터셋 응축에서 강한 가중치는 학습에 좋은 영향을 미치지 못한다. 가중치의 L2 Norm 이 Baseline 과 가까워질수록(작아질수록) 성능이 개선되고 Loss 또한 줄어드는 것을 보였다. 다만 Batch Sampling 을 이용한 경우 Baseline 보다 Loss 가 상대적으로 컸음에도 유사한 정확도를 보이는 것을 바탕으로 Matching Loss 에 대해 다시 한번 고민해볼 필요가 있다.

Batch Sampling 을 이용하여 원본 데이터셋의 더 넓은 데이터 공간을 활용하는 것은 어려운 것으로 나타났다. 응축 데이터셋의 데이터 분포가 Baseline 과 매우 유사한 것을 실험을 통해 보였다. 따라서 많은 데이터셋 응축 연구에서 보였던 원본 데이터셋의 분포를 학습하도록 하는 학습 방법론은 이미지 단계가 아닌 특징 단계에서 고민되어야 하는 것을 알 수 있다.

향후 연구로는 앞선 실험에서 확인할 수 있었던 Matching Loss 의 효용에 대해 다시 한번 고민이 필요하다. 나아가 실험 4.3 에서 확인할 수 있었듯이 이미지 데이터의 low-level 특징을 포착하고 학습하는 것이 Matching Loss 측면에서 도움을 준다. 따라서 이미지 특징 단계에 따른 데이터셋 응축 효과를 면밀하게 살피고 이를 극대화하는 방향으로 연구가 필요하다.

6. 참고문헌

[1] Krizhevsky, Alex, Ilya Sutskever, and Geffrey E Hinton. "Imagenet classification with deep neural convolutional neural networks." Advances in neural information processing systems 25 (2012): 1097-1105.

[2] Liu, Hanxiao, Karen Simonyan, and Yiming Yang. "Darts: Differentiable architecture search." arXiv preprint arXiv:1806.09055 (2018).

[3] Lopez-Paz, David, and Marc'Aurelio Ranzato. "Gradient episodic memory for continual learning." Advances in neural information processing systems 30 (2017).

[4] Zhao, Bo, Konda Reddy Mopuri, and Hakan Bilen. "Dataset condensation with gradient matching." arXiv preprint arXiv:2006.05929 (2020).

[5] Wang, Kai, Bo Zhao, Xiangyu Peng, Zheng Zhu, Shuo Yang, Shuo Wang, Guan Huang, Hakan Bilen, Xinchao Wang, and Yang You. "Cafe: Learning to condense dataset by aligning features." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 12196-12205. 2022.

[6] Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A Efros. "Dataset distillation." arXiv preprint arXiv:1811.10959 (2018).

[7] Zhao, Bo, and Hakan Bilen. "Dataset Condensation with Differentiable Siamese Augmentation." arxiv preprint arXiv:2102.08259 (2021).

[8] Zhao, Bo, Konda Reddy Mopuri, and Hakan Bilen. "Dataset condensation with gradient matching." arXiv preprint arXiv:2006.05929 (2020).

[9] Kim, Jang-Hyun, Jinuk Kim, Seong Joon Oh, Sangdoo Yun, Hwanjun Song, Joonhyun Jeong, Jung-Woo Ha, and Hyun Oh Song. "Dataset Condensation via Efficient Synthetic-Data Parameterization." arXiv preprint arXiv:2205.14959 (2022).

[10] Lee, H. B., Lee, D. B., & Hwang, S. J. (2022). Dataset Condensation with Latent Space Knowledge Factorization and Sharing. arXiv preprint arXiv:2208.10494.

[11] Nguyen, Timothy, Roman Novak, Lechao Xiao, and Jaehoon Lee. "Dataset distillation with infinitely wide convolutional networks." Advances in Neural Information Processing Systems 34 (2021): 5186-5198.