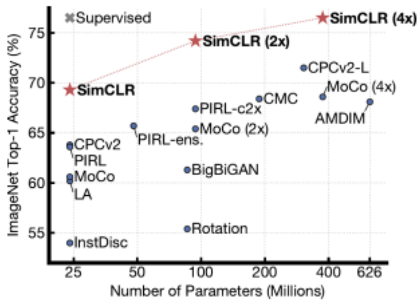
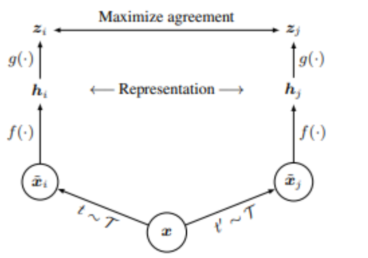
SimCLR

논문명 : A Simple Framework for Contrastive Learning of Visual Representations learn

- No decoder
- No proxy task
- With contrastive loss을 통해 학습
- Premise: Augmentation은 semantic한 정보를 바꾸지 않는다.
- Positive sample과 Negative sample을 비교하며 학습
    - Positive sample : 같은 이미지를 Augmentation 한 데이터



성능 표



데이터를 1개를 Random Augmentation을 통해 2개의 transform 된 데이터 생성 → f함수 (resnet50과 같은 모델)을 통해 Representation 추출 → g (projection head)를 통해 enbeding된 데이터 추출 → 비교

1. Data Augmentation의 조합에 따른 성능을 실험
2. Projection Head의 개념을 사용.
    a. layer를 통화한 테이터를 통해 enbeding을 얻는 layer
3. InfoNCE loss을 이용한 Contrastive learning을 SSL에 적용함.
    a. infoNCE loss

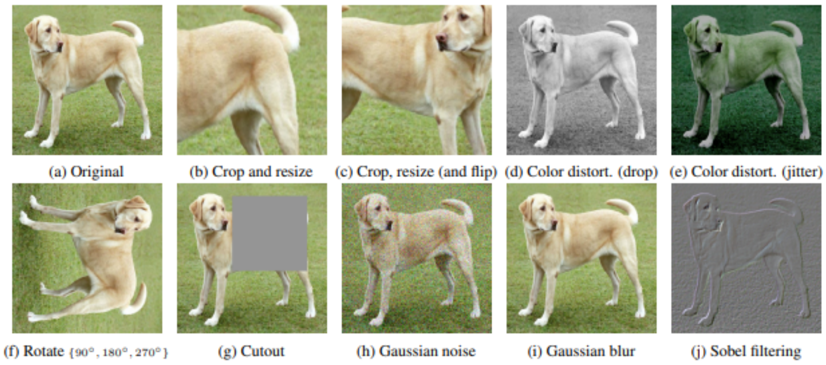
    전체 비교군(positive + negative)에서 positive sample을 positive라고 할 확률

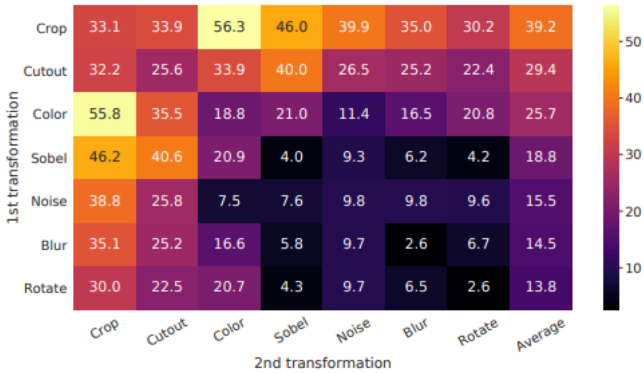
$$\ell_{i,j} = -\log \frac{\exp(\mathrm{sim}(\boldsymbol{z}_i, \boldsymbol{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\mathrm{sim}(\boldsymbol{z}_i, \boldsymbol{z}_k)/\tau)},$$

같은 데이터를 통해 나온 embeding데이터가 아니면 다 Negative sample로 판단

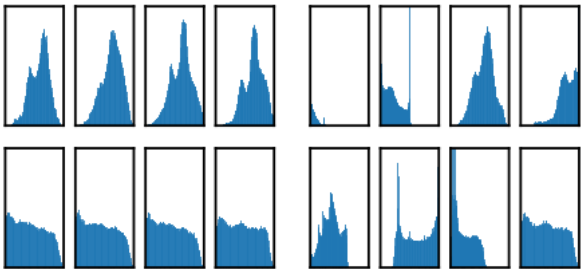
각각의 백터의 Simularity 추출 후 비교(loss)

4. 큰 배치 사이즈와 긴 학습 에폭으로 성능을 향상시킬 수 있음.



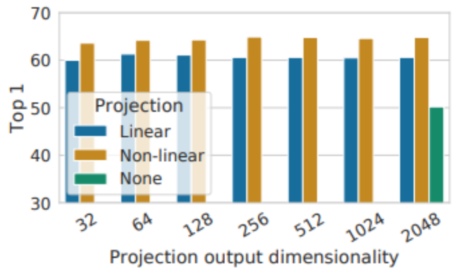


💡 단일 transformation을 할 경우 결과가 좋지 않음



(a) Without color distortion. (b) With color distortion.

💡 color distortion을 하지 않으면 이미지의 색상만으로 short cut 발생
color distortion을 통해 같은 이미지여도 차이를 주어 학습



liner : FC layer 1개

Non-liner : FC layer n개

none : resnet50 output 그대로 사용 → 2048개

💡 Non-liner가 그나마 성능이 좋고 dimension과 상관 없다

What to predict?	Random guess	Representation $\boldsymbol{h}$	Representation $g(\boldsymbol{h})$
Color vs grayscale	80	99.3	97.4
Rotation	25	67.6	25.6
Orig. vs corrupted	50	99.5	59.6
Orig. vs Sobel filtered	50	96.6	56.3

💡 Projection Head 전 후의 값을 통한 비교 분석

1. color와 grayscale은 둘 다 높은 성능

→ 둘 다 컬러 정보 포함되어 있음

2. rotation, corrupted, sobel filter시 random guess 와 비슷

→Projection Head 후 local 정보 소실

Down stream task에 적용할 때는 h를 사용

Name	Negative loss function	Gradient w.r.t. $\boldsymbol{u}$
NT-Xent	$\boldsymbol{u}^T\boldsymbol{v}^+/\tau - \log\sum_{\boldsymbol{v}\in\{\boldsymbol{v}^+,\boldsymbol{v}^-\}}\exp(\boldsymbol{u}^T\boldsymbol{v}/\tau)$	$(1-\frac{\exp(\boldsymbol{u}^T\boldsymbol{v}^+/\tau)}{Z(\boldsymbol{u})})/\tau\boldsymbol{v}^+ - \sum_{\boldsymbol{v}^-}\frac{\exp(\boldsymbol{u}^T\boldsymbol{v}^-/\tau)}{Z(\boldsymbol{u})}/\tau\boldsymbol{v}^-$
NT-Logistic	$\log\sigma(\boldsymbol{u}^T\boldsymbol{v}^+/\tau) + \log\sigma(-\boldsymbol{u}^T\boldsymbol{v}^-/\tau)$	$(\sigma(-\boldsymbol{u}^T\boldsymbol{v}^+/\tau))/\tau\boldsymbol{v}^+ - \sigma(\boldsymbol{u}^T\boldsymbol{v}^-/\tau)/\tau\boldsymbol{v}^-$
Margin Triplet	$-\max(\boldsymbol{u}^T\boldsymbol{v}^- - \boldsymbol{u}^T\boldsymbol{v}^+ + m, 0)$	$\boldsymbol{v}^+ - \boldsymbol{v}^-$ if $\boldsymbol{u}^T\boldsymbol{v}^+ - \boldsymbol{u}^T\boldsymbol{v}^- < m$ else $\boldsymbol{0}$

💡 NT-Xent의 gradient는, 분류가 어려운 negative sample에 더 큰 가중치를 적용.

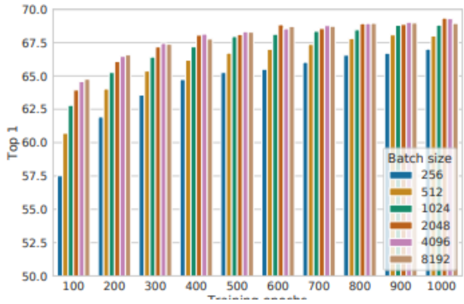
Relative hardness를 반영함.

Margin	NT-Logi.	Margin (sh)	NT-Logi.(sh)	NT-Xent
50.9	51.6	57.5	57.9	63.9

Table 4. Linear evaluation (top-1) for models trained with different loss functions. "sh" means using semi-hard negative mining.

ℓ_2 norm?	τ	Entropy	Contrastive acc.	Top 1
Yes	0.05	1.0	90.5	59.7
	0.1	4.5	87.8	64.4
	0.5	8.2	68.2	60.7
	1	8.3	59.1	58.0
No	10	0.5	91.7	57.2
	100	0.5	92.1	57.0

💡 L2 norm 이 성능이 더 좋음



비슷한 데이터(노란 고양이, 회색 고양이)의 비교 시 batch size가 높으면 negative sample이 많아져 불필요한 데이터의 비율이 적어지고 학습 개선됨

💡 batch가 크고 epoch가 클수록 좋아진다

평가

A Simple Framework for Contrastive Learning of Visual Representations

	Food	CIFAR10	CIFAR100	Birdsnap	SUN397	Cars	Aircraft	VOC2007	DTD	Pets	Caltech-101	Flowers
Linear evaluation:												
SimCLR (ours)	**76.9**	**95.3**	80.2	48.4	**65.9**	60.0	61.2	**84.2**	**78.9**	89.2	**93.9**	**95.0**
Supervised	75.2	**95.7**	**81.2**	**56.4**	64.9	**68.8**	**63.8**	83.8	**78.7**	**92.3**	**94.1**	94.2
Fine-tuned:												
SimCLR (ours)	**89.4**	**98.6**	**89.0**	**78.2**	**68.1**	**92.1**	**87.0**	**86.6**	**77.8**	92.1	**94.1**	97.6
Supervised	88.7	98.3	**88.7**	**77.8**	67.0	91.4	**88.0**	86.5	**78.8**	**93.2**	**94.2**	**98.0**
Random init	88.3	96.0	81.9	**77.0**	53.7	91.3	84.8	69.4	64.1	82.7	72.5	92.5

💡 Contrastive learning의 첫 논문으로 기존의 SSL 방법론보다 성능이 앞섬

하지만 Negative sampling에 대한 의존성이 커, 큰 batch size를 필요로 하기 때문에 한계가 존재.

▼ 코드

```
import argparse
import builtins
import math
import os
import random
import shutil
import time
import warnings
import torch
```

```
import torch.nn as nn
import torch.nn.parallel
import torch.backends.cudnn as cudnn
import torch.distributed as dist
import torch.optim
import torch.multiprocessing as mp
import torch.utils.data
import torch.utils.data.distributed
import torchvision.transforms as transforms
import torchvision.datasets as datasets
import torchvision.models as models
import torchvision
import torch.nn.functional as F
from PIL import ImageFilter
from pathlib import Path
from tqdm import tqdm
from PIL import Image, ImageOps, ImageFilter

model_names = sorted(name for name in models.__dict__
                     if name.islower() and not name.startswith("__")
                     and callable(models.__dict__[name]))

parser = argparse.ArgumentParser(description='PyTorch ImageNet Training')
parser.add_argument('--data', metavar='DIR', default='/mnt/MONG/benchmark/ImageNet/ILSVRC/Data/ImageNet',
                    help='path to dataset')
parser.add_argument('-a', '--arch', metavar='ARCH', default='resnet50', choices=model_names,
                    help='model architecture: ' + ' | '.join(model_names) +
                         ' (default: resnet50)')
parser.add_argument('-j', '--workers', default=64, type=int, metavar='N',
                    help='number of data loading workers (default: 32)')
parser.add_argument('--epochs', default=100, type=int, metavar='N', help='number of total epochs to run')
parser.add_argument('--start-epoch', default=0, type=int, metavar='N', help='manual epoch number (useful on restarts)')
parser.add_argument('-b', '--batch-size', default=256, type=int, metavar='N',
                    help='mini-batch size (default: 512), this is the total '
                         'batch size of all GPUs on the current node when '
                         'using Data Parallel or Distributed Data Parallel')
parser.add_argument('--lr', '--learning-rate', default=0.2, type=float, metavar='LR',
                    help='initial (base) learning rate', dest='lr')
parser.add_argument('--momentum', default=0.9, type=float, metavar='M', help='momentum of SGD solver')
parser.add_argument('--wd', '--weight-decay', default=1e-4, type=float, metavar='W',
                    help='weight decay (default: 1e-4)', dest='weight_decay')
parser.add_argument('-p', '--print-freq', default=500, type=int, metavar='N', help='print frequency (default: 10)')
parser.add_argument('--resume', default=True, type=bool)
parser.add_argument('--world-size', default=-1, type=int, help='number of nodes for distributed training')
parser.add_argument('--rank', default=-1, type=int, help='node rank for distributed training')
parser.add_argument('--dist-url', default='tcp://224.66.41.62:23456', type=str,
                    help='url used to set up distributed training')
parser.add_argument('--gpu', default='0,1,2,3', type=str, help='GPU id to use.')

parser.add_argument('--projector', default='8192-8192-8192', type=str, metavar='MLP', help='projector MLP')
parser.add_argument('--tau', default=0.1, type=float, help='temperature (default: 0.1)')
parser.add_argument('--dim', default=2048, type=int, help='feature dimension (default: 2048)')
parser.add_argument('--pred-dim', default=512, type=int, help='hidden dimension of the predictor (default: 512)')
parser.add_argument('--fix-pred-lr', action='store_true', help='Fix learning rate for the predictor')
parser.add_argument('--checkpoint', default='./checkpoint', type=Path, metavar='DIR',
                    help='path to checkpoint directory')


def main():
    args = parser.parse_args()
    os.environ["CUDA_VISIBLE_DEVICES"] = args.gpu
    args.ngpus_per_node = len(args.gpu.split(','))
    args.rank = 0
    args.dist_url = f'tcp://localhost:{random.randrange(49152, 65535)}'
    args.world_size = args.ngpus_per_node
    args.distributed = True

    if args.rank == 0:
        args.checkpoint.mkdir(parents=True, exist_ok=True)
    print('start')
    torch.multiprocessing.spawn(main_worker, (args,), args.ngpus_per_node)


def main_worker(gpu, args):
    args.gpu = gpu
    args.rank = gpu

    torch.distributed.init_process_group(
        backend='nccl', init_method=args.dist_url,
        world_size=args.world_size, rank=args.rank)

    # create model
    print("=> creating model '{}'".format(args.arch))
    model = simclr(args)

    # infer learning rate before changing batch size
```

```
    init_lr = args.lr * args.batch_size / 256

    torch.cuda.set_device(args.gpu)
    torch.backends.cudnn.benchmark = True
    model.cuda(args.gpu)

    model = torch.nn.SyncBatchNorm.convert_sync_batchnorm(model)
    # When using a single GPU per process and per
    # DistributedDataParallel, we need to divide the batch size
    # ourselves based on the total number of GPUs we have
    per_batch_size = int(args.batch_size / args.ngpus_per_node)
    args.workers = args.ngpus_per_node
    model = torch.nn.parallel.DistributedDataParallel(model, device_ids=[args.gpu])
    optim_params = model.parameters()
    optimizer = torch.optim.SGD(optim_params, init_lr, momentum=args.momentum, weight_decay=args.weight_decay)

    # optionally resume from a checkpoint
    if args.resume:
        if (args.checkpoint / 'checkpoint.pth').is_file():
            print("=> loading checkpoint '{}'".format(args.checkpoint))
            checkpoint = torch.load(args.checkpoint / 'checkpoint.pth', map_location='cpu')
            args.start_epoch = checkpoint['epoch']
            model.load_state_dict(checkpoint['model'])
            print("=> loaded checkpoint '{}' (epoch {})"
                  .format(args.checkpoint, '{}/checkpoint.pth'.format(checkpoint['epoch'])))
        else:
            print("=> no checkpoint found at '{}'".format(args.checkpoint))

    cudnn.benchmark = True

    # Data loading code
    traindir = args.data

    augmentation = [
        transforms.RandomResizedCrop(224, scale=(0.2, 1.)),
        transforms.RandomApply([
            transforms.ColorJitter(0.4, 0.4, 0.4, 0.1)  # not strengthened
        ], p=0.8),
        transforms.RandomGrayscale(p=0.2),
        transforms.RandomApply([GaussianBlur([.1, 2.])], p=0.5),
        transforms.RandomHorizontalFlip(),
        Solarization(p=0.2),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
    ]

    train_dataset = datasets.ImageFolder(traindir, TwoCropsTransform(transforms.Compose(augmentation)))
    train_sampler = torch.utils.data.distributed.DistributedSampler(train_dataset)
    train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=per_batch_size, num_workers=args.workers, drop_last=True,
                                               pin_memory=True, sampler=train_sampler)


    for epoch in range(args.start_epoch, args.epochs):
        train_sampler.set_epoch(epoch)
        adjust_learning_rate(optimizer, init_lr, epoch, args)

        # train for one epoch
        train(train_loader, model, optimizer, epoch, args)
        if args.rank == 0:
            save_model(args, args.checkpoint, epoch, model.module)

    if args.rank == 0:
        # save final model
        torch.save(model.module.backbone.state_dict(), args.checkpoint / 'resnet50.pth')

def train(train_loader, model, optimizer, epoch, args):
    batch_time = AverageMeter('Time', ':6.3f')
    data_time = AverageMeter('Data', ':6.3f')
    losses = AverageMeter('Loss', ':.4f')
    progress = ProgressMeter(
        len(train_loader),
        [batch_time, data_time, losses],
        prefix="Epoch: [{}]".format(epoch))

    print('=>  train start')
    # switch to train mode
    model.train()

    end = time.time()
    for i, (images, _) in enumerate(tqdm(train_loader)):
        # measure data loading time
        data_time.update(time.time() - end)

        images[0] = images[0].cuda(args.gpu, non_blocking=True)
        images[1] = images[1].cuda(args.gpu, non_blocking=True)
```

```
        # compute output and loss
        loss = model(images[0], images[1])

        losses.update(loss.item(), images[0].size(0))

        # compute gradient and do SGD step
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if i % args.print_freq == 0 and args.gpu == 0:
            progress.display(i)



def adjust_learning_rate(optimizer, init_lr, epoch, args):
    """Decay the learning rate based on schedule"""
    cur_lr = init_lr * 0.5 * (1. + math.cos(math.pi * epoch / args.epochs))
    for param_group in optimizer.param_groups:
        param_group['lr'] = cur_lr



class simclr(nn.Module):
    def __init__(self, args):
        super().__init__()
        self.args = args
        self.tau = 0.1
        self.backbone = torchvision.models.resnet50(zero_init_residual=True)
        self.backbone.fc = nn.Identity()

        # projector
        sizes = [2048] + list(map(int, '128-128-128'.split('-')))
        layers = []
        for i in range(len(sizes) - 2):
            layers.append(nn.Linear(sizes[i], sizes[i + 1], bias=False))
            layers.append(nn.BatchNorm1d(sizes[i + 1]))
            layers.append(nn.ReLU(inplace=True))
        layers.append(nn.Linear(sizes[-2], sizes[-1], bias=False))
        self.projector = nn.Sequential(*layers)

    def forward(self, y1, y2):
        '''Get embeddings'''
        z1 = self.projector(self.backbone(y1))
        z2 = self.projector(self.backbone(y2))

        '''All gather'''
        z1_list = [torch.zeros_like(z1, dtype=torch.float, device=z1.device) for _ in range(self.args.ngpus_per_node)]
        z2_list = [torch.zeros_like(z2, dtype=torch.float, device=z2.device) for _ in range(self.args.ngpus_per_node)]
        z1_list = AllGather.apply(z1_list, z1)
        z2_list = AllGather.apply(z2_list, z2)
        z1 = torch.cat(z1_list, 0)
        z2 = torch.cat(z2_list, 0)

        '''Normalize & concat'''
        z1 = F.normalize(z1, dim=1)
        z2 = F.normalize(z2, dim=1)
        y = torch.cat([z1, z2], dim=0)

        '''calculate logit'''
        logits = y @ y.T
        logits.fill_diagonal_(0)

        '''make label'''
        label1 = torch.arange(self.args.batch_size, self.args.batch_size * 2, dtype=torch.long, device=y1.device)
        label2 = torch.arange(0, self.args.batch_size, dtype=torch.long, device=y1.device)
        labels = torch.cat([label1, label2], dim=0)

        '''calculate loss'''
        loss = F.cross_entropy(logits / self.tau, labels, reduction='mean')
        return loss




class GaussianBlur(object):
    def __init__(self, p):
        self.p = p

    def __call__(self, img):
        if random.random() < self.p:
            sigma = random.random() * 1.9 + 0.1
            return img.filter(ImageFilter.GaussianBlur(sigma))
        else:
            return img
```

```
class Solarization(object):
    def __init__(self, p):
        self.p = p

    def __call__(self, img):
        if random.random() < self.p:
            return ImageOps.solarize(img)
        else:
            return img


class TwoCropsTransform:
    """Take two random crops of one image as the query and key."""

    def __init__(self, base_transform):
        self.base_transform = base_transform

    def __call__(self, x):
        q = self.base_transform(x)
        k = self.base_transform(x)
        return q, k

def save_model(args, checkpoint, epoch, model):
    torch.save(
        {
            'epoch': epoch,
            'args': args,
            'model': model.state_dict()
        },
        f=checkpoint+'/checkpoint.pth'
    )

class AverageMeter(object):
    """Computes and stores the average and current value"""
    def __init__(self, name, fmt=':f'):
        self.name = name
        self.fmt = fmt
        self.reset()

    def reset(self):
        self.val = 0
        self.avg = 0
        self.sum = 0
        self.count = 0

    def update(self, val, n=1):
        self.val = val
        self.sum += val * n
        self.count += n
        self.avg = self.sum / self.count

    def __str__(self):
        fmtstr = '{name} {val' + self.fmt + '} ({avg' + self.fmt + '})'
        return fmtstr.format(**self.__dict__)


class ProgressMeter(object):
    def __init__(self, num_batches, meters, prefix=""):
        self.batch_fmtstr = self._get_batch_fmtstr(num_batches)
        self.meters = meters
        self.prefix = prefix

    def display(self, batch):
        entries = [self.prefix + self.batch_fmtstr.format(batch)]
        entries += [str(meter) for meter in self.meters]
        print('\t'.join(entries))

    def _get_batch_fmtstr(self, num_batches):
        num_digits = len(str(num_batches // 1))
        fmt = '{:' + str(num_digits) + 'd}'
        return '[' + fmt + '/' + fmt.format(num_batches) + ']'


def accuracy(output, target, topk=(1,)):
    """Computes the accuracy over the k top predictions for the specified values of k"""
    with torch.no_grad():
        maxk = max(topk)
        batch_size = target.size(0)

        _, pred = output.topk(maxk, 1, True, True)
        pred = pred.t()
        correct = pred.eq(target.view(1, -1).expand_as(pred))

        res = []
        for k in topk:
```

```python
            correct_k = correct[:k].reshape(-1).float().sum(0, keepdim=True)
            res.append(correct_k.mul_(100.0 / batch_size))
        return res

class AllGather(torch.autograd.Function):
    """
    all_gather with gradient back-propagation
    """

    @staticmethod
    def forward(ctx, tensor_list, tensor):
        dist.all_gather(tensor_list, tensor)
        return tuple(tensor_list)

    @staticmethod
    def backward(ctx, *grad_list):
        grad_list = list(grad_list)
        rank = dist.get_rank()

        dist_ops = [
            dist.reduce(grad_list[i], i, async_op=True) for i in range(dist.get_world_size())
        ]

        for op in dist_ops:
            op.wait()

        return None, grad_list[rank]

if __name__ == '__main__':
    main()
```