January 8, 2024

Module

Self-supervised and Semi-supervised Learning

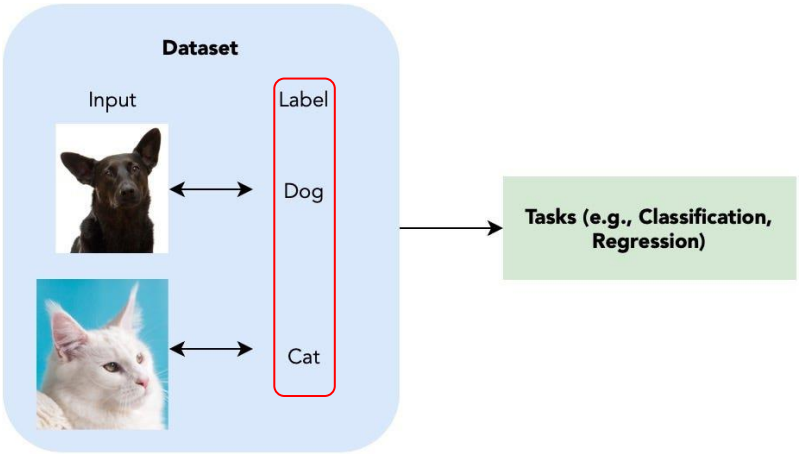
Ph.D. Ngo Ba Hung

Email: ngohung@jnu.ac.kr
Facebook: https://www.facebook.com/hung.ngo.7121

Graduate School of Data Science
Chonnam National University

Overview

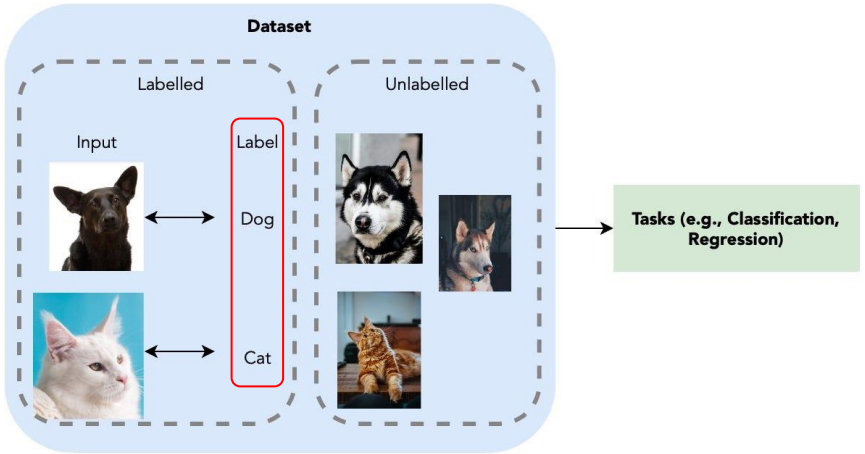
Supervised Learning



supervised learning provides a set of input-output pairs.

The entire dataset to be trained on samples has the corresponding labels.

Semi-supervised Learning

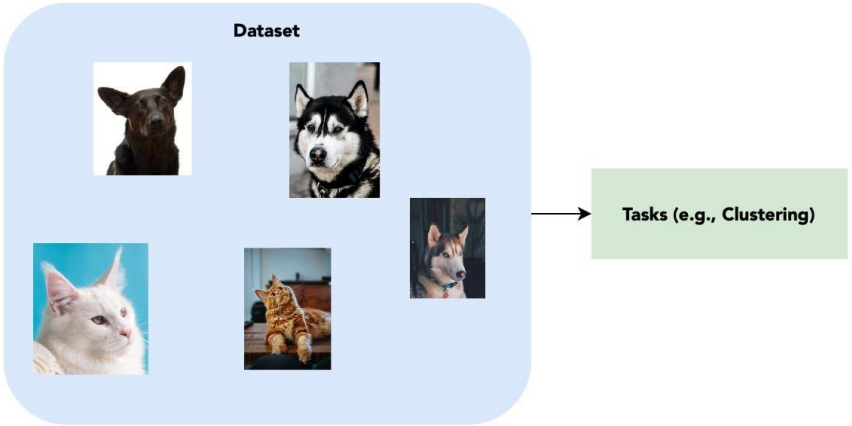


Semi-supervised learning provides a small set of input-output pairs.

The model trained on small labeled data is tested on the large unlabeled data (the unlabeled data is also used during training).

Overview

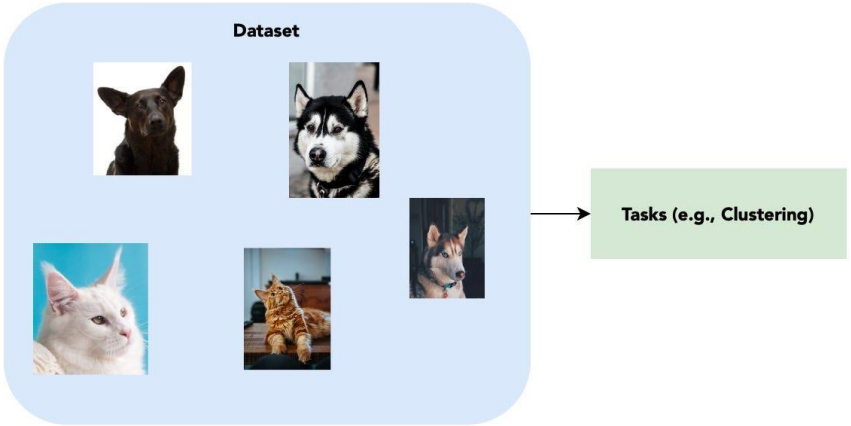
Unsupervised Learning



Unsupervised learning only input data has no corresponding labels.

Using method for exploiting extracted representations (samples having the similar features are grouped (clustered) together, but samples having the different features keep far from each other.
(K-NN, K-Mean, K-Medoids methods)

Self-supervised Learning



Self-supervised learning is categorized into to **unsupervised learning** because no labels were given.

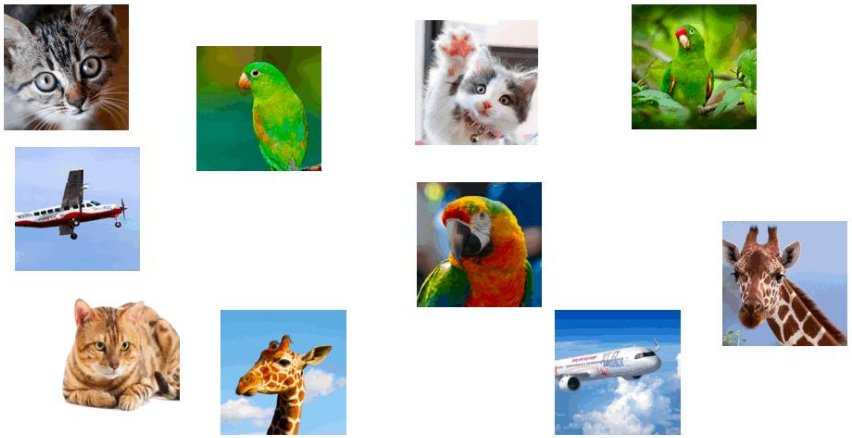
Overview

- Families of SSL
  - Deep Metric Learning
    - SimCLR
    - NNCLR
    - SCL
  - Self-distillation
    - SimSIAM
    - BYOL
    - DINO, DINO v2
  - Canonical Correlation Analysis
    - VICReg
    - BarlowTwins
    - SWAV
    - EMP-SSL

Topic 1: Self-supervised Learning

Deep Metric Learning

- SimCLR*
  - Concept of contrastive learning



Mathematic explanation and its formulation:

$$\mathcal{L}_{\text{cont}} = -\log \frac{\exp(\text{sim}(\mathbf{z_i}, \mathbf{z_j})/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\text{sim}(\mathbf{z_i}, \mathbf{z_k})/\tau)}$$

$$= \log \frac{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\text{sim}(\mathbf{z_i}, \mathbf{z_k})/\tau)}{\exp(\text{sim}(\mathbf{z_i}, \mathbf{z_j})/\tau)}$$

$$= \log \left(\frac{\exp(\text{sim}(\mathbf{z_i}, \mathbf{z_j})/\tau) + \sum_{k=1}^{2N} \mathbb{1}_{[k \neq i,j]} \exp(\text{sim}(\mathbf{z_i}, \mathbf{z_k})/\tau)}{\exp(\text{sim}(\mathbf{z_i}, \mathbf{z_j})/\tau)} \right)$$

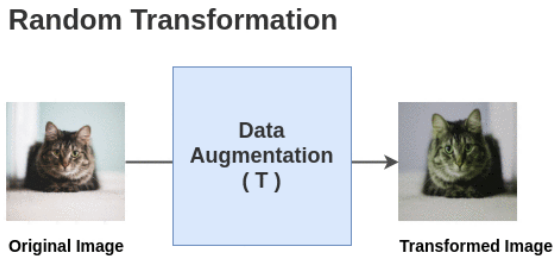
$$= \log \left(1 + \frac{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i,j]} \exp(\text{sim}(\mathbf{z_i}, \mathbf{z_k})/\tau)}{\exp(\text{sim}(\mathbf{z_i}, \mathbf{z_j})/\tau)} \right)$$

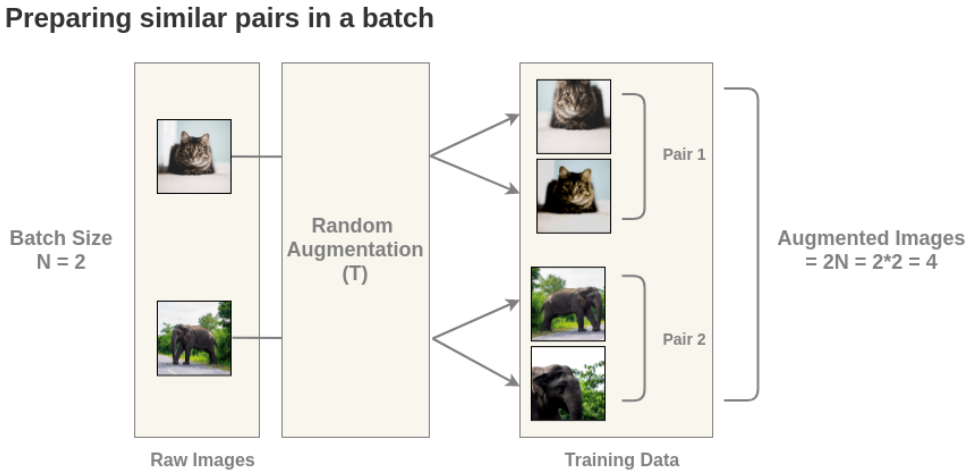
$$\downarrow \mathcal{L}_{\text{cont}} = \log \left(1 + \frac{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i,j]} \exp(\downarrow \text{sim}(\mathbf{z_i}, \mathbf{z_k})/\tau)}{\exp(\uparrow \text{sim}(\mathbf{z_i}, \mathbf{z_j})/\tau)} \right)$$

Topic 1: Self-supervised Learning

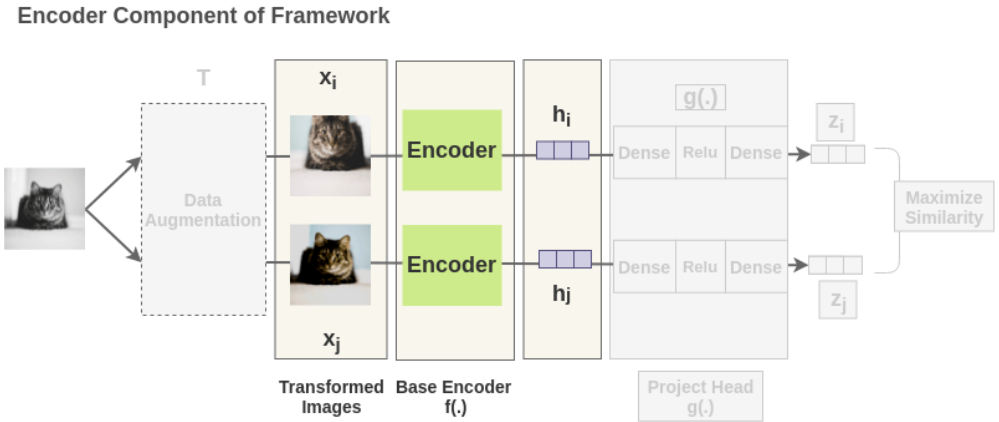
Deep Metric Learning

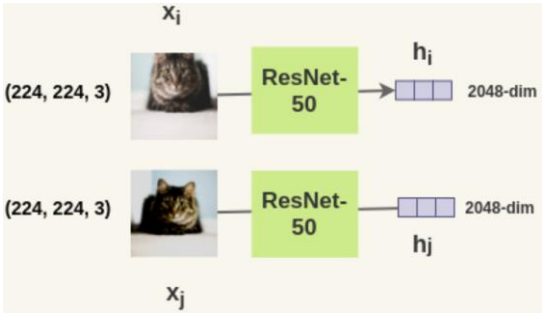
- SimCLR*
  - Image transformation





  - Getting representation



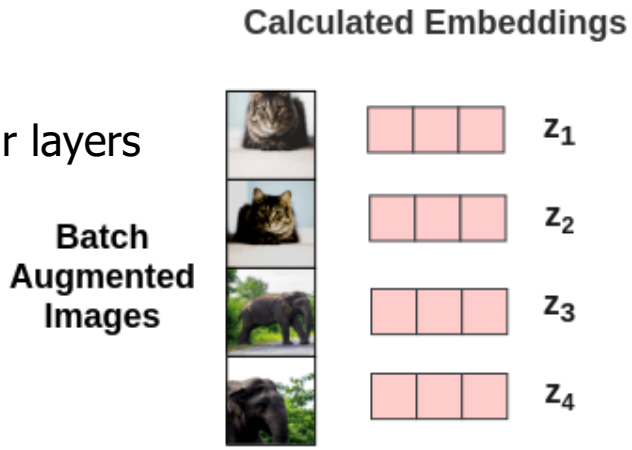


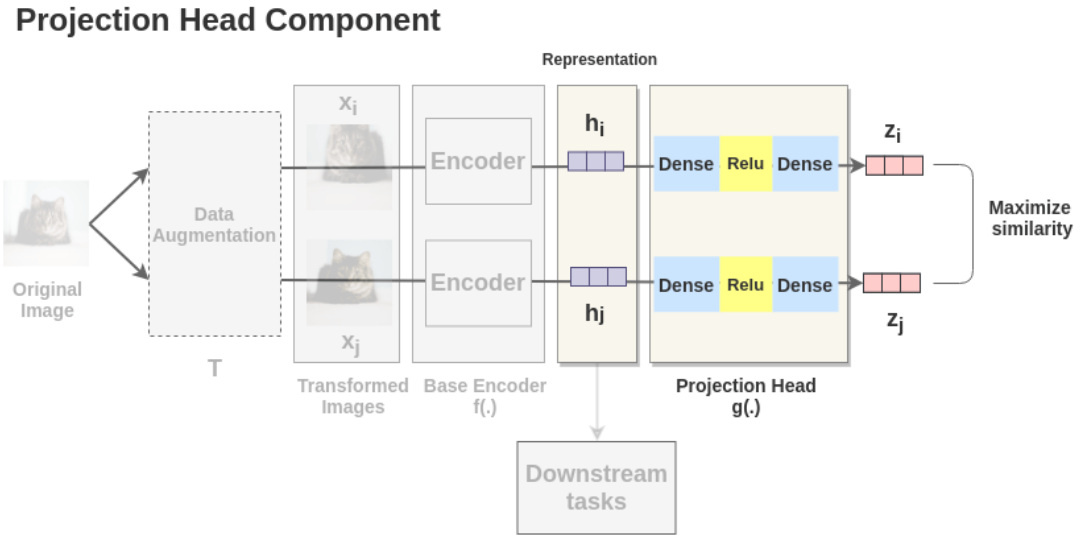
Topic 1: Self-supervised Learning

Deep Metric Learning

- SimCLR*
  - Projection head
    - h_i and h_j are passed through a series of non-linear layers

Dense → Relu → Dense

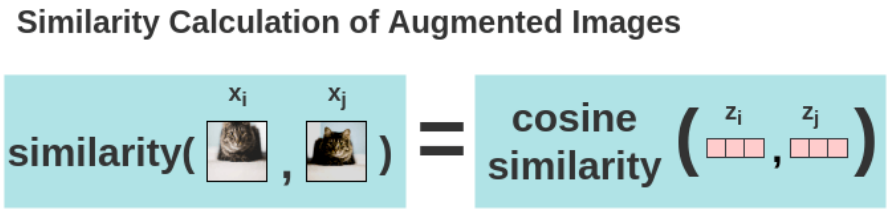


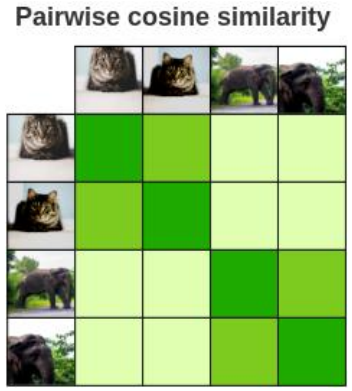


Topic 1: Self-supervised Learning

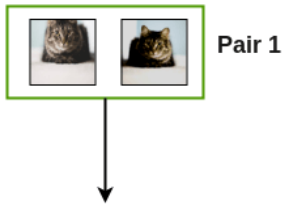
Deep Metric Learning

- SimCLR*
  - Calculation of cosine similarity

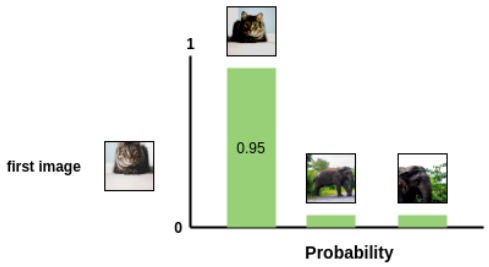




  - Calculate the probability of two similarity images



$$\text{Softmax} = \frac{e^{\text{similarity}(\cdot,\cdot)}}{e^{\text{similarity}(\cdot,\cdot)} + e^{\text{similarity}(\cdot,\cdot)} + e^{\text{similarity}(\cdot,\cdot)}}$$



*Chen, Xinlei, and Kaiming He. "Exploring simple siamese representation learning." *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)*. **2021**.

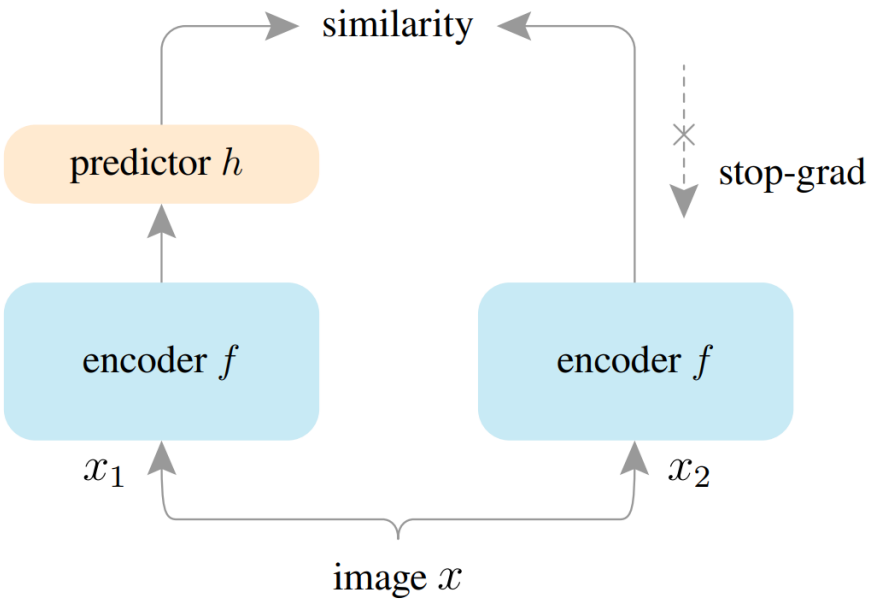
Self-distillation

- **Sim**ple **Siam**ese network (SimSIAM)*
  - Architecture
  - Pseudo code of SimSIAM

Algorithm 1 SimSiam Pseudocode, PyTorch-like

```
# f: backbone + projection mlp
# h: prediction mlp

for x in loader: # load a minibatch x with n samples
   x1, x2 = aug(x), aug(x) # random augmentation
   z1, z2 = f(x1), f(x2) # projections, n-by-d
   p1, p2 = h(z1), h(z2) # predictions, n-by-d

   L = D(p1, z2)/2 + D(p2, z1)/2 # loss

   L.backward() # back-propagate
   update(f, h) # SGD update

def D(p, z): # negative cosine similarity
   z = z.detach() # stop gradient

   p = normalize(p, dim=1) # l2-normalize
   z = normalize(z, dim=1) # l2-normalize
   return -(p*z).sum(dim=1).mean()
```

Encoder uses ResNet backbone network
Projection uses multi-layer perceptron (MLP)



x_1 and x_2 augmented images of an input image x

Topic 1: Self-supervised Learning

Self-distillation

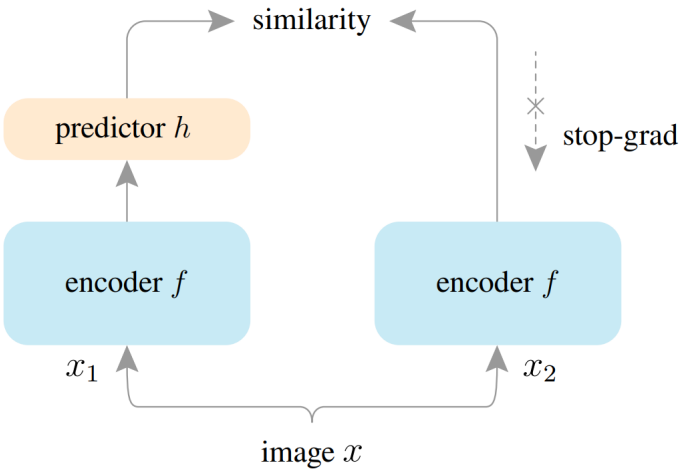
- **Sim**ple **Siam**ese network (SimSIAM)*
  - Output of the encoder

$$z_1 \triangleq f(x_1) \qquad z_2 \triangleq f(x_2)$$

  - Output of the predictor

$$p_1 \triangleq h(f(x_1)) \qquad p_2 \triangleq h(f(x_2))$$

  - Minimize the negative cosine similarity



$$\mathcal{D}(p_1, z_2) = -\frac{p_1}{\|p_1\|_2} \cdot \frac{z_2}{\|z_2\|_2} \qquad \mathcal{L} = \frac{1}{2}\mathcal{D}(p_1, z_2) + \frac{1}{2}\mathcal{D}(p_2, z_1).$$

Self-distillation

- **Sim**ple **Siam**ese network (SimSIAM)*
  - Output of the encoder

$$z_1 \triangleq f(x_1) \qquad z_2 \triangleq f(x_2)$$

  - Output of the predictor

$$p_1 \triangleq h(f(x_1)) \qquad p_2 \triangleq h(f(x_2))$$

  - Minimize the negative cosine similarity

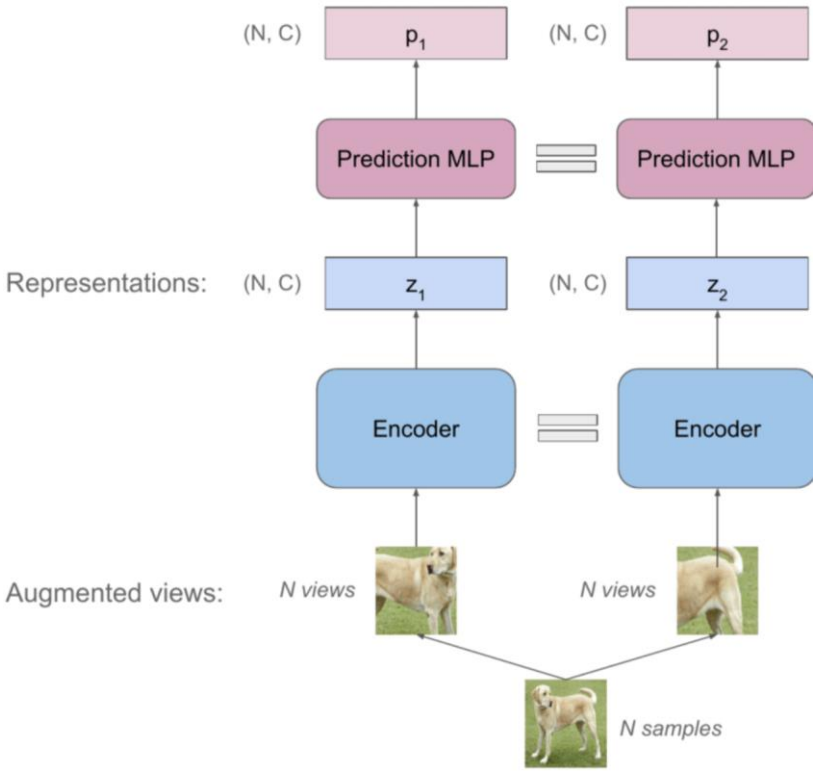
$$\mathcal{D}(p_1, z_2) \rightarrow \mathcal{D}(p_1, \texttt{stopgrad}(z_2)). \rightarrow \mathcal{L} = \frac{1}{2}\mathcal{D}(p_1, \texttt{stopgrad}(z_2)) + \frac{1}{2}\mathcal{D}(p_2, \texttt{stopgrad}(z_1)).$$

*Chen, Xinlei, and Kaiming He. "Exploring simple siamese representation learning." *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)*. **2021**.

Topic 1: Self-supervised Learning

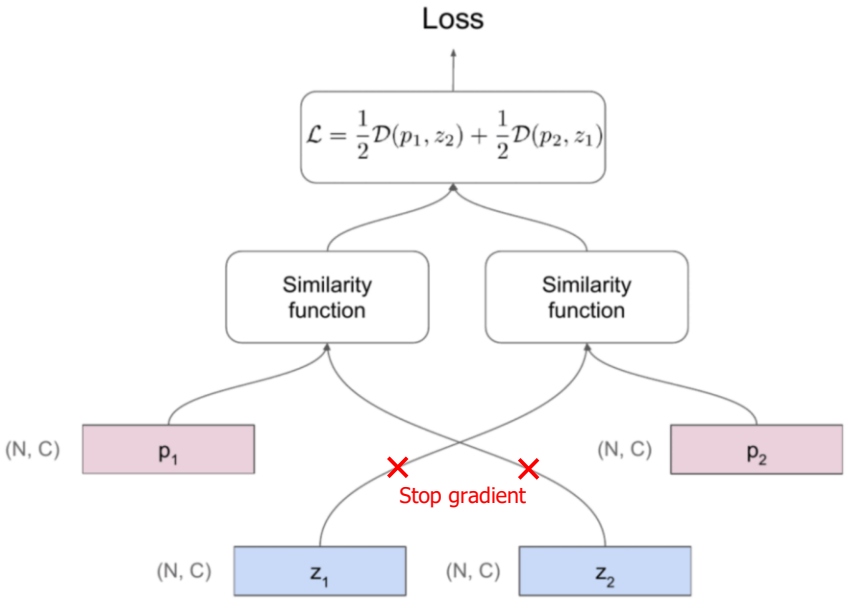
Self-distillation

- SimSIAM*
  - Summary



Algorithm 1 SimSiam Pseudocode, PyTorch-like

```
# f: backbone + projection mlp
# h: prediction mlp

for x in loader: # load a minibatch x with n samples
   x1, x2 = aug(x), aug(x) # random augmentation
   z1, z2 = f(x1), f(x2) # projections, n-by-d
   p1, p2 = h(z1), h(z2) # predictions, n-by-d

   L = D(p1, z2)/2 + D(p2, z1)/2 # loss

   L.backward() # back-propagate
   update(f, h) # SGD update

def D(p, z): # negative cosine similarity
   z = z.detach() # stop gradient

   p = normalize(p, dim=1) # l2-normalize
   z = normalize(z, dim=1) # l2-normalize
   return -(p*z).sum(dim=1).mean()
```



Implementation

- Implementation
  - Dataset
    - CIFAR-10
      - 4,000 images (labeled)
      - 56,000 image (unlabeled)
  - Code

```
parser = argparse.ArgumentParser(description='Train SimSiam')
parser.add_argument('--feature_dim', default=2048, type=int, help='Feature dim for out vector')
parser.add_argument('--k', default=200, type=int, help='Top k most similar images used to predict the label')
parser.add_argument('--batch_size', default=512, type=int, help='Number of images in each mini-batch')
parser.add_argument('--epochs', default=800, type=int, help='Number of sweeps over the dataset to train')
```

  - Data prepare

```
# data prepare
train_data = CIFAR10Pair(root='data', train=True, transform=train_transform, download=True)
train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True, num_workers=16, pin_memory=True)
memory_data = CIFAR10Pair(root='data', train=True, transform=test_transform, download=True)
memory_loader = DataLoader(memory_data, batch_size=batch_size, shuffle=False, num_workers=16, pin_memory=True)
test_data = CIFAR10Pair(root='data', train=False, transform=test_transform, download=True)
test_loader = DataLoader(test_data, batch_size=batch_size, shuffle=False, num_workers=16, pin_memory=True)
```

*Chen, Xinlei, and Kaiming He. "Exploring simple siamese representation learning." *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)*. **2021**.

Implementation

- Implementation
  - Optimizer

```
# model setup and optimizer config
model = Model(feature_dim).cuda()

optimizer = SGD(model.parameters(), lr=0.03, momentum=0.9, weight_decay=5e-4)
lr_scheduler = LambdaLR(optimizer, lr_lambda=lambda i: 0.5 * (math.cos(i * math.pi / epochs) + 1))
c = len(memory_data.classes)

results = {'train_loss': [], 'test_acc@1': [], 'test_acc@5': []}
```

  - Training

```
# training loop
for epoch in range(1, epochs + 1):
    train_loss = train(model, train_loader, optimizer)

    results['train_loss'].append(train_loss)
    lr_scheduler.step()
    test_acc_1, test_acc_5 = test(model, memory_loader, test_loader)

    results['test_acc@1'].append(test_acc_1)
    results['test_acc@5'].append(test_acc_5)
    # save statistics
    data_frame = pd.DataFrame(data=results, index=range(1, epoch + 1))
    data_frame.to_csv('results/{}_statistics.csv'.format(save_name_pre), index_label='epoch')
    if test_acc_1 > best_acc:
        best_acc = test_acc_1
        torch.save(model.state_dict(), 'results/{}_model.pth'.format(save_name_pre))
```

*Chen, Xinlei, and Kaiming He. "Exploring simple siamese representation learning." *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)*. **2021**.

Algorithm 1 SimSiam Pseudocode, PyTorch-like

```
# f: backbone + projection mlp
# h: prediction mlp

for x in loader: # load a minibatch x with n samples
   x1, x2 = aug(x), aug(x) # random augmentation
   z1, z2 = f(x1), f(x2) # projections, n-by-d
   p1, p2 = h(z1), h(z2) # predictions, n-by-d

   L = D(p1, z2)/2 + D(p2, z1)/2 # loss

   L.backward() # back-propagate
   update(f, h) # SGD update

def D(p, z): # negative cosine similarity
   z = z.detach() # stop gradient

   p = normalize(p, dim=1) # l2-normalize
   z = normalize(z, dim=1) # l2-normalize
   return -(p*z).sum(dim=1).mean()
```


Implementation

- Implementation
  - Training

```
def train(net, data_loader, train_optimizer):
    net.train()
    total_loss, total_num, train_bar = 0.0, 0, tqdm(data_loader, dynamic_ncols=True)
    for pos_1, pos_2, _ in train_bar:

        pos_1, pos_2 = pos_1.cuda(non_blocking=True), pos_2.cuda(non_blocking=True)

        feature_1, proj_1 = net(pos_1)
        feature_2, proj_2 = net(pos_2)

        # compute loss
        sim_1 = -(F.normalize(proj_1, dim=-1) * F.normalize(feature_2.detach(), dim=-1)).sum(dim=-1).mean()
        sim_2 = -(F.normalize(proj_2, dim=-1) * F.normalize(feature_1.detach(), dim=-1)).sum(dim=-1).mean()

        loss = 0.5 * sim_1 + 0.5 * sim_2

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

*Chen, Xinlei, and Kaiming He. "Exploring simple siamese representation learning." *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)*. **2021**.

Implementation

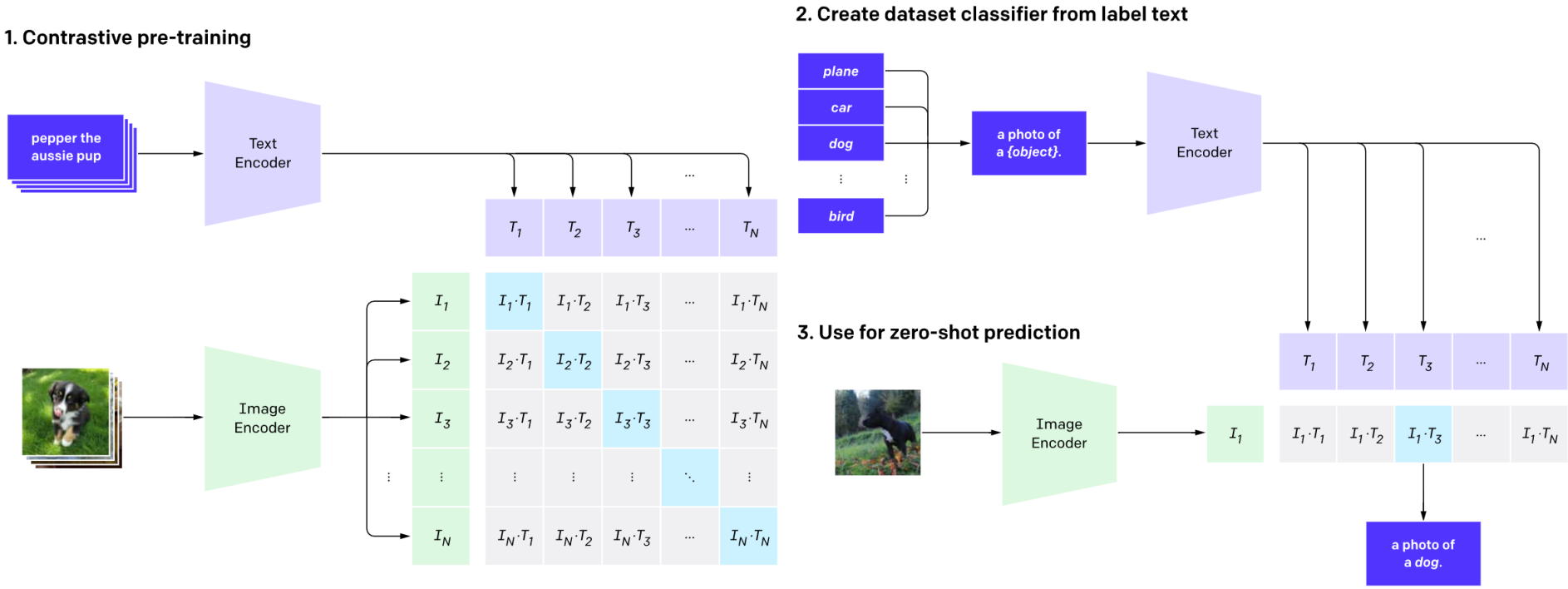
- Implementation
  - Results

```
Feature extracting: 100%|██████████| 98/98 [00:46<00:00,  2.09it/s]
  5%|▌         | 1/20 [00:35<11:16, 35.63s/it]Test Epoch: [470/800] Acc@1:89.06% Acc@5:99.02%
Test Epoch: [470/800] Acc@1:88.28% Acc@5:99.22%
 15%|█▌        | 3/20 [00:35<02:37,  9.27s/it]Test Epoch: [470/800] Acc@1:87.63% Acc@5:99.41%
Test Epoch: [470/800] Acc@1:87.06% Acc@5:99.32%
Test Epoch: [470/800] Acc@1:86.88% Acc@5:99.30%
Test Epoch: [470/800] Acc@1:86.98% Acc@5:99.22%
 30%|███       | 6/20 [00:35<00:50,  3.61s/it]Test Epoch: [470/800] Acc@1:86.75% Acc@5:99.16%
Test Epoch: [470/800] Acc@1:86.72% Acc@5:99.15%
 45%|████▌     | 9/20 [00:35<00:21,  1.94s/it]Test Epoch: [470/800] Acc@1:87.00% Acc@5:99.13%
Test Epoch: [470/800] Acc@1:86.64% Acc@5:99.16%
Test Epoch: [470/800] Acc@1:86.70% Acc@5:99.20%
 60%|██████    | 12/20 [00:36<00:09,  1.18s/it]Test Epoch: [470/800] Acc@1:86.70% Acc@5:99.17%
Test Epoch: [470/800] Acc@1:86.79% Acc@5:99.19%
Test Epoch: [470/800] Acc@1:86.73% Acc@5:99.15%
 75%|███████▌  | 15/20 [00:36<00:03,  1.31it/s]Test Epoch: [470/800] Acc@1:86.73% Acc@5:99.17%
Test Epoch: [470/800] Acc@1:86.74% Acc@5:99.18%
Test Epoch: [470/800] Acc@1:86.70% Acc@5:99.18%
 90%|█████████ | 18/20 [00:36<00:01,  1.95it/s]Test Epoch: [470/800] Acc@1:86.87% Acc@5:99.16%
Test Epoch: [470/800] Acc@1:86.82% Acc@5:99.18%
Test Epoch: [470/800] Acc@1:86.78% Acc@5:99.20%
100%|██████████| 20/20 [00:36<00:00,  1.85s/it]
```

Topic 1: Self-supervised Learning

Extension

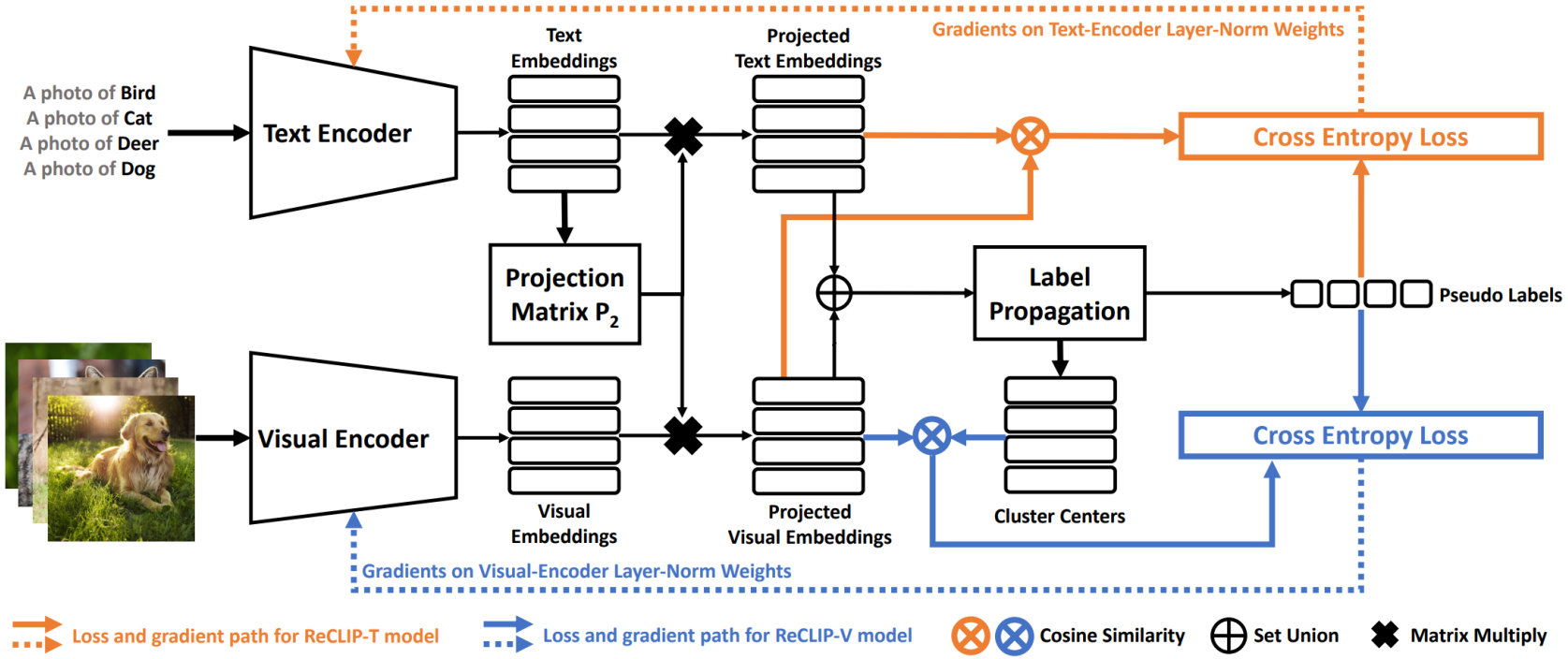
❑ CLIP - Contrastive Language-Image Pre-training



Topic 1: Self-supervised Learning

Extension

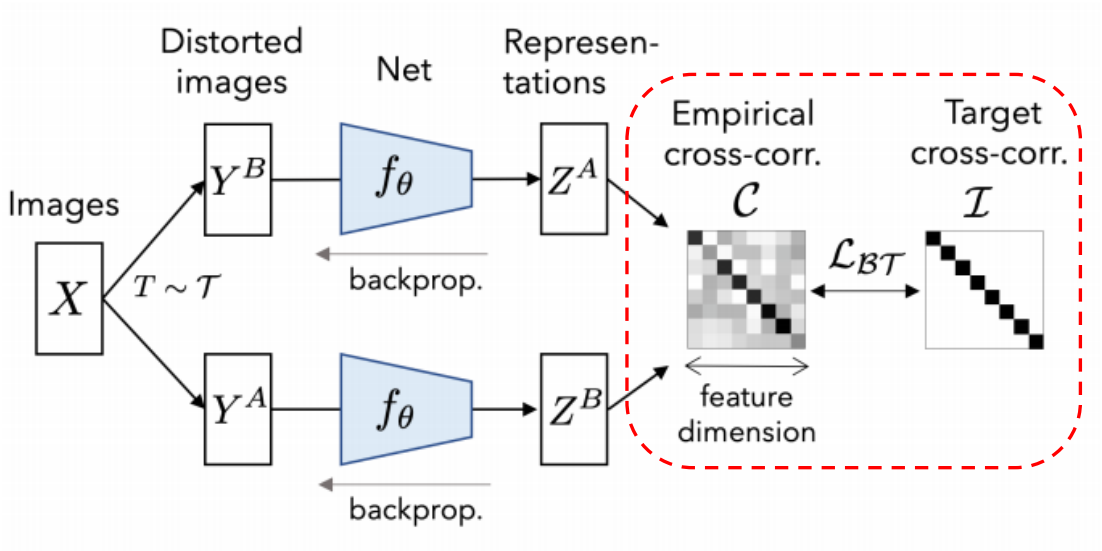
❑ ReCLIP



Method

Canonical Correlation Analysis

- Barlow Twins*
  - Self-Supervised Learning via Redundancy Reduction
    - Architecture



Algorithm 1 PyTorch-style pseudocode for Barlow Twins.

```
# f: encoder network
# lambda: weight on the off-diagonal terms
# N: batch size
# D: dimensionality of the embeddings
#
# mm: matrix-matrix multiplication
# off_diagonal: off-diagonal elements of a matrix
# eye: identity matrix

for x in loader: # load a batch with N samples
    # two randomly augmented versions of x
    y_a, y_b = augment(x)

    # compute embeddings
    z_a = f(y_a) # NxD
    z_b = f(y_b) # NxD

    # normalize repr. along the batch dimension
    z_a_norm = (z_a - z_a.mean(0)) / z_a.std(0) # NxD
    z_b_norm = (z_b - z_b.mean(0)) / z_b.std(0) # NxD

    # cross-correlation matrix
    c = mm(z_a_norm.T, z_b_norm) / N # DxD

    # loss
    c_diff = (c - eye(D)).pow(2) # DxD
    # multiply off-diagonal elems of c_diff by lambda
    off_diagonal(c_diff).mul_(lambda)
    loss = c_diff.sum()

    # optimization step
    loss.backward()
    optimizer.step()
```

Canonical Correlation Analysis

- Barlow Twins*
  - Self-Supervised Learning via Redundancy Reduction
    - Operation
      - Invariance term
        - » Make the embedding from different image views (augmentation versions) of the same image close each other
      - Redundancy reduction term

$$\mathcal{C}_{ij} \triangleq \frac{\sum_b z^A_{b,i} z^B_{b,j}}{\sqrt{\sum_b \left(z^A_{b,i}\right)^2} \sqrt{\sum_b \left(z^B_{b,j}\right)^2}}$$

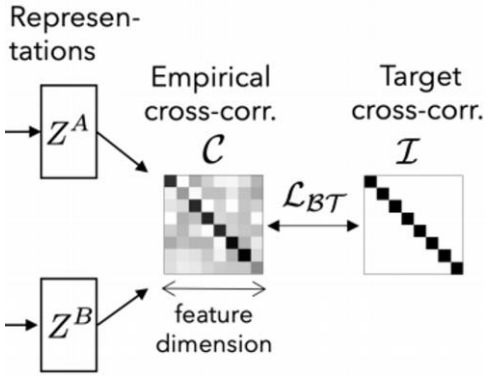
$$\mathcal{L}_{\mathcal{BT}} \triangleq \underbrace{\sum_i (1 - \mathcal{C}_{ii})^2}_{\text{invariance term}} + \lambda \underbrace{\sum_i \sum_{j \neq i} \mathcal{C}_{ij}{}^2}_{\text{redundancy reduction term}}$$

Deal with the on-diagonal terms

As close as possible to one

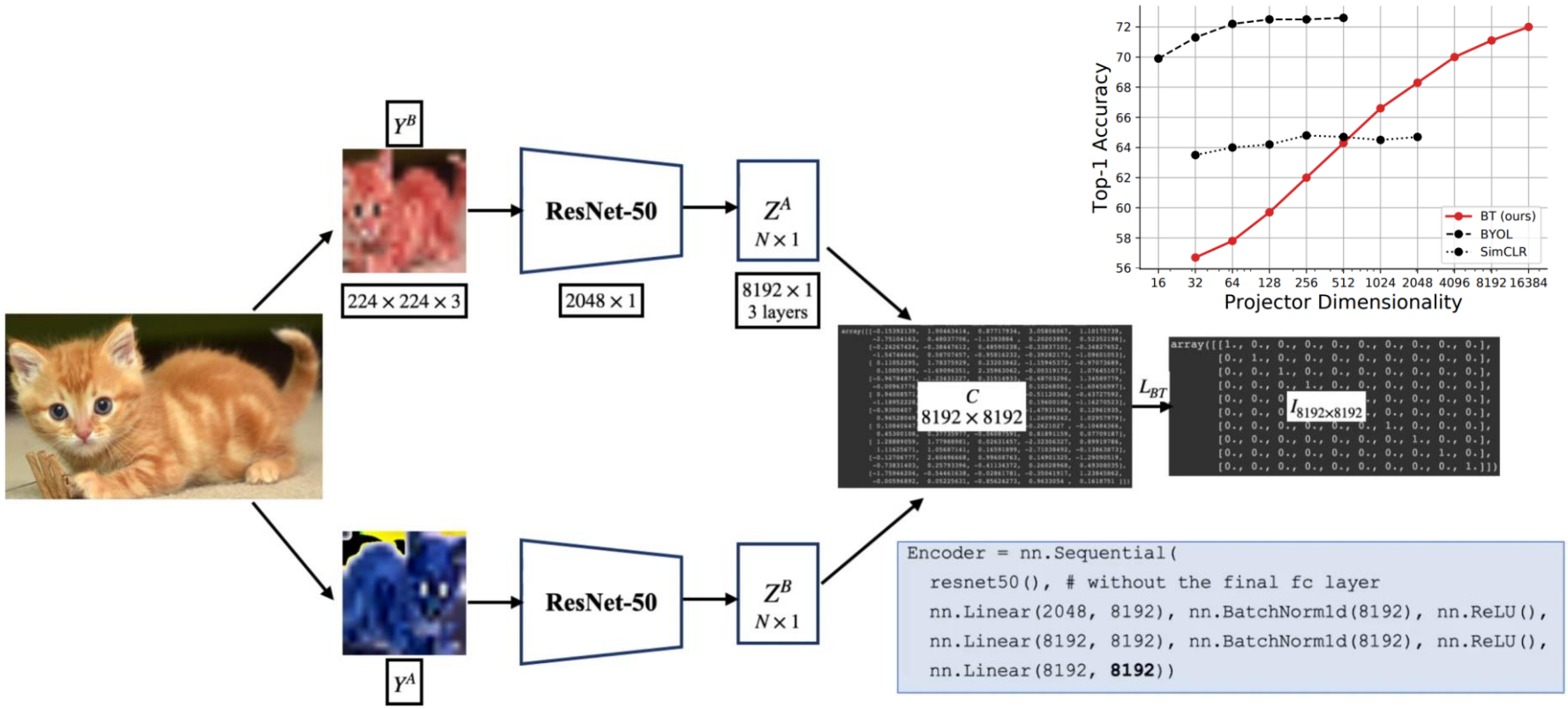
Deal with the off-diagonal terms

As close as possible to zero



Canonical Correlation Analysis

- Barlow Twins*
  - Self-Supervised Learning via Redundancy Reduction




```
Encoder = nn.Sequential(
  resnet50(), # without the final fc layer
  nn.Linear(2048, 8192), nn.BatchNorm1d(8192), nn.ReLU(),
  nn.Linear(8192, 8192), nn.BatchNorm1d(8192), nn.ReLU(),
  nn.Linear(8192, 8192))
```

January 8, 2024

Module

Semi-supervise Learning (SSL)

Ph.D. Ba Hung Ngo

Department of Multimedia Engineering

Dongguk University

*Sohn, Kihyuk, et al. "Fixmatch: Simplifying semi-supervised learning with consistency and confidence." *Advances in neural information processing systems* 33 (NeurIPS), **2020**.

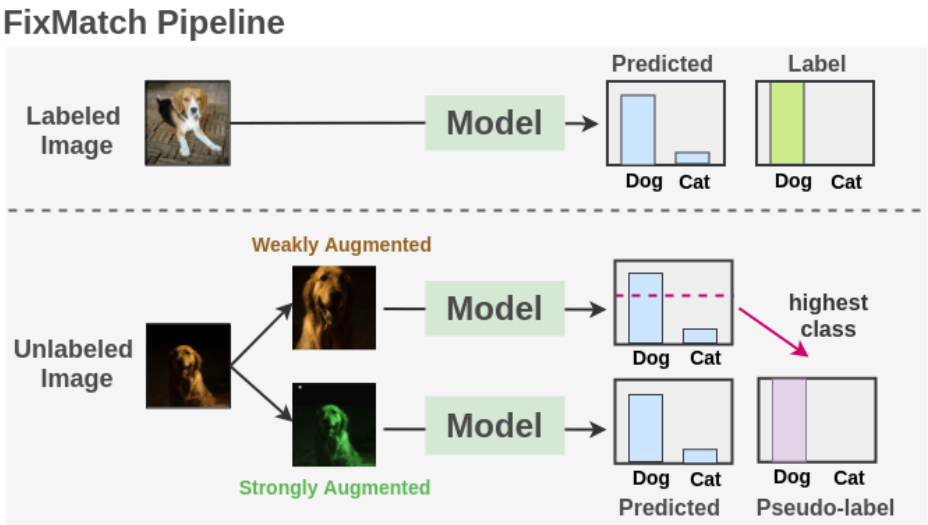
Topic 2: Semi-supervised Learning

Method

FixMatch*

Data Setting:

Training Procedure:

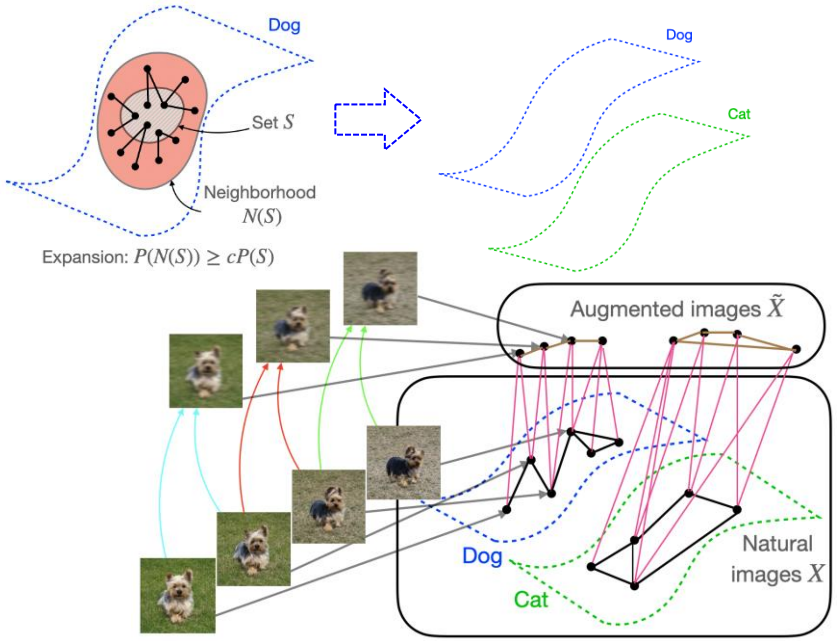


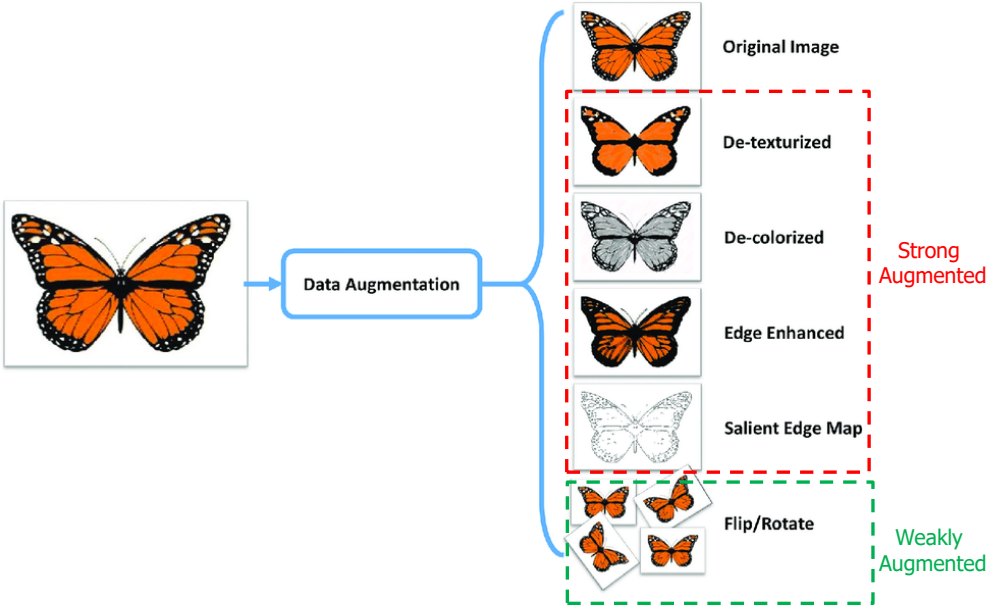
*Sohn, Kihyuk, et al. "Fixmatch: Simplifying semi-supervised learning with consistency and confidence." *Advances in neural information processing systems* 33 (NeurIPS), **2020**.

Method

FixMatch*

❑ Data Augmentation



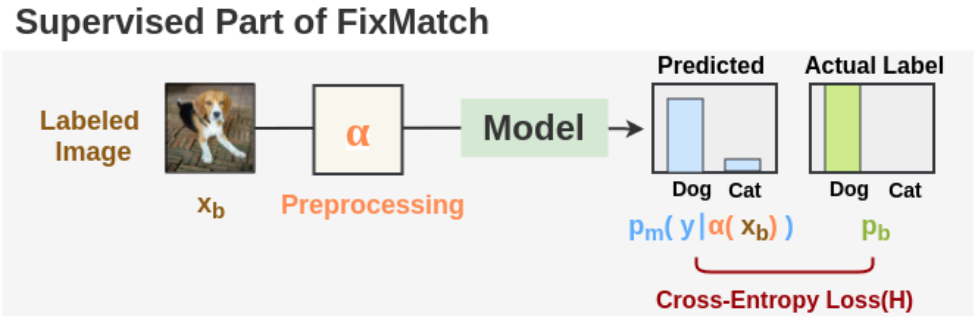


*Sohn, Kihyuk, et al. "Fixmatch: Simplifying semi-supervised learning with consistency and confidence." *Advances in neural information processing systems* 33 (NeurIPS), **2020**.

Method

FixMatch*

- Model Training and Loss Functions
  - Supervised Learning



$$l_s = \frac{1}{B}\sum_{b=1}^{B} H(p_b, p_m(y|\alpha(x_b)))$$

*Sohn, Kihyuk, et al. "Fixmatch: Simplifying semi-supervised learning with consistency and confidence." *Advances in neural information processing systems* 33 (NeurIPS), **2020**.

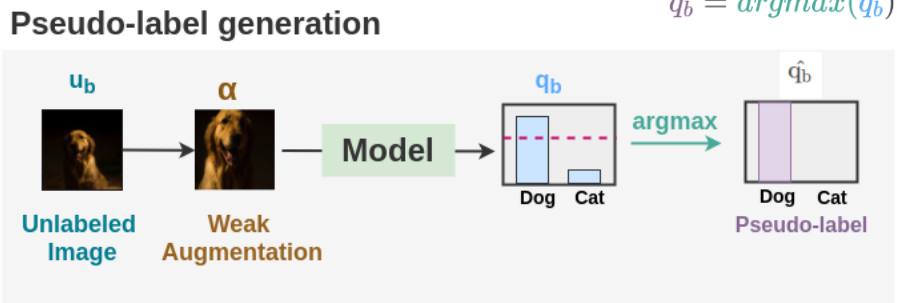
Method

FixMatch*

- Model Training and Loss Functions
  - Unsupervised Learning
    - Pseudo labeling

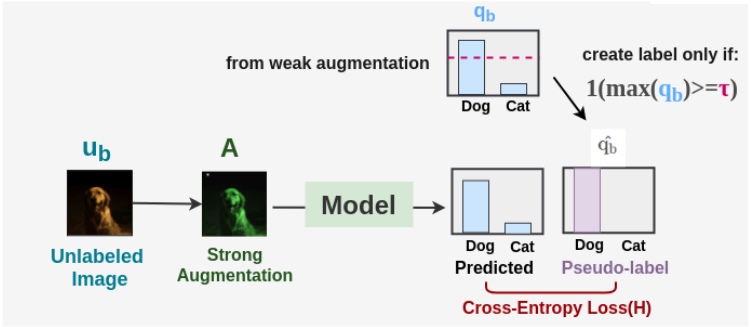
$$q_b = p_m(y|\alpha(u_b))$$

$$\hat{q}_b = argmax(q_b)$$



    - Consistency regularization

$$l_u = \frac{1}{\mu B}\sum_{b=1}^{\mu B} 1(max(q_b) >= \tau)\ H(\hat{q}_b, p_m(y|A(u_b)))$$

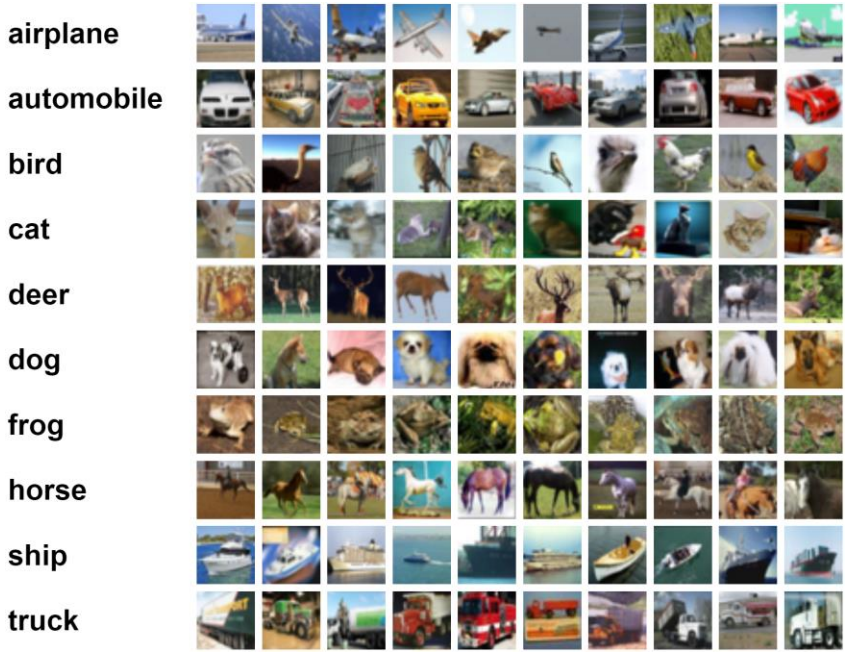


*Sohn, Kihyuk, et al. "Fixmatch: Simplifying semi-supervised learning with consistency and confidence." *Advances in neural information processing systems* 33 (NeurIPS), **2020**.

Method

FixMatch*

- Implementation
  - Dataset
    - CIFAR-10
      - 60,000 color images
      - 10 classes
      - 50,000 training images (labeled)
      - 10,000 training images (unlabeled)



*Sohn, Kihyuk, et al. "Fixmatch: Simplifying semi-supervised learning with consistency and confidence." *Advances in neural information processing systems* 33 (NeurIPS), **2020**.

Method

FixMatch*

- ❑ Implementation
  - Dataset
    - CIFAR-10
      - 4,000 images (labeled)
      - 56,000 image (unlabeled)
  - Code

```
if args.dataset == 'cifar10':
    args.num_classes = 10
    if args.arch == 'wideresnet':
        args.model_depth = 28
        args.model_width = 2
    elif args.arch == 'resnext':
        args.model_cardinality = 4
        args.model_depth = 28
        args.model_width = 4
```

```
elif args.dataset == 'cifar100':
    args.num_classes = 100
    if args.arch == 'wideresnet':
        args.model_depth = 28
        args.model_width = 8
    elif args.arch == 'resnext':
        args.model_cardinality = 8
        args.model_depth = 29
        args.model_width = 64
```

*Sohn, Kihyuk, et al. "Fixmatch: Simplifying semi-supervised learning with consistency and confidence." *Advances in neural information processing systems* 33 (NeurIPS), **2020**.

Method

FixMatch*

❑ Implementation

- Data preparation

```
DATASET_GETTERS = {'cifar10': get_cifar10, 'cifar100': get_cifar100}
```

- Data Loader

```
def get_cifar10(args, root):

    transform_labeled = transforms.Compose([
        transforms.RandomHorizontalFlip(),
        transforms.RandomCrop(size=32, padding=int(32 * 0.125), padding_mode='reflect'),
        transforms.ToTensor(),
        transforms.Normalize(mean=cifar10_mean, std=cifar10_std)
    ])

    transform_val = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(mean=cifar10_mean, std=cifar10_std)
    ])
    base_dataset = datasets.CIFAR10(root, train=True, download=True)

    train_labeled_idxs, train_unlabeled_idxs = x_u_split(args, base_dataset.targets)

    train_labeled_dataset = CIFAR10SSL(root, train_labeled_idxs, train=True, transform=transform_labeled)

    train_unlabeled_dataset = CIFAR10SSL(root, train_unlabeled_idxs, train=True,
                                         transform=TransformFixMatch(mean=cifar10_mean, std=cifar10_std))

    test_dataset = datasets.CIFAR10(root, train=False, transform=transform_val, download=False)

    return train_labeled_dataset, train_unlabeled_dataset, test_dataset
```

*Sohn, Kihyuk, et al. "Fixmatch: Simplifying semi-supervised learning with consistency and confidence." *Advances in neural information processing systems* 33 (NeurIPS), **2020**.

Method

FixMatch*

- Implementation
  - Data transformation

```
class TransformFixMatch(object):
    def __init__(self, mean, std):
        self.weak = transforms.Compose([
            transforms.RandomHorizontalFlip(),
            transforms.RandomCrop(size=32, padding=int(32 * 0.125), padding_mode='reflect')])

        self.strong = transforms.Compose([
            transforms.RandomHorizontalFlip(),
            transforms.RandomCrop(size=32, padding=int(32 * 0.125), padding_mode='reflect'),
            RandAugmentMC(n=2, m=10)])

        self.normalize = transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize(mean=mean, std=std)])

    def __call__(self, x):
        weak = self.weak(x)
        strong = self.strong(x)
        return self.normalize(weak), self.normalize(strong)
```

*Sohn, Kihyuk, et al. "Fixmatch: Simplifying semi-supervised learning with consistency and confidence." *Advances in neural information processing systems* 33 (NeurIPS), **2020**.

Method

FixMatch*

❑ Implementation

- Model selection

```
def main():
    parser = argparse.ArgumentParser(description='PyTorch FixMatch Training')
    parser.add_argument('--gpu-id', default='0', type=int, help='id(s) for CUDA_VISIBLE_DEVICES')
    parser.add_argument('--num-workers', type=int, default=4, help='number of workers')
    parser.add_argument('--dataset', default='cifar10', type=str, choices=['cifar10', 'cifar100'], help='dataset name')
    parser.add_argument('--num-labeled', type=int, default=4000, help='number of labeled data')
    parser.add_argument("--expand-labels", action="store_true", help="expand labels to fit eval steps")
    parser.add_argument('--arch', default='wideresnet', type=str, choices=['wideresnet', 'resnext'], help='dataset name')
```

```
def create_model(args):
    if args.arch == 'wideresnet':
        import models.wideresnet as models
        model = models.build_wideresnet(depth=args.model_depth, widen_factor=args.model_width, dropout=0, num_classes=args.num_classes)
    elif args.arch == 'resnext':
        import models.resnext as models
        model = models.build_resnext(cardinality=args.model_cardinality, depth=args.model_depth, width=args.model_width, num_classes=args.num_classes)
```

*Sohn, Kihyuk, et al. "Fixmatch: Simplifying semi-supervised learning with consistency and confidence." *Advances in neural information processing systems* 33 (NeurIPS), **2020**.

Topic 2: Semi-supervised Learning

FixMatch*

- ❑ Implementation
  - Model

```
def forward(self, x):
    x = self.conv_1_3x3.forward(x)
    x = self.act(self.bn_1.forward(x))
    x = self.stage_1.forward(x)
    x = self.stage_2.forward(x)
    x = self.stage_3.forward(x)
    x = F.adaptive_avg_pool2d(x, 1)
    x = x.view(-1, self.stages[3])
    return self.classifier(x)
```

```
class CifarResNeXt(nn.Module):
    """
    ResNext optimized for the Cifar dataset, as specified in
    https://arxiv.org/pdf/1611.05431.pdf
    """

    def __init__(self, cardinality, depth, num_classes,
                 base_width, widen_factor=4):
        """ Constructor
        Args:
            cardinality: number of convolution groups.
            depth: number of layers.
            nlabels: number of classes
            base_width: base number of channels in each group.
            widen_factor: factor to adjust the channel dimensionality
        """
super().__init__()
self.cardinality = cardinality
self.depth = depth
self.block_depth = (self.depth - 2) // 9
self.base_width = base_width
self.widen_factor = widen_factor
self.nlabels = num_classes
self.output_size = 64
self.stages = [64, 64 * self.widen_factor, 128 *
               self.widen_factor, 256 * self.widen_factor]

self.conv_1_3x3 = nn.Conv2d(3, 64, 3, 1, 1, bias=False)
self.bn_1 = nn.BatchNorm2d(64, momentum=0.001)
self.act = mish
self.stage_1 = self.block('stage_1', self.stages[0], self.stages[1], 1)
self.stage_2 = self.block('stage_2', self.stages[1], self.stages[2], 2)
self.stage_3 = self.block('stage_3', self.stages[2], self.stages[3], 2)
self.classifier = nn.Linear(self.stages[3], num_classes)
```

Method

*Sohn, Kihyuk, et al. "Fixmatch: Simplifying semi-supervised learning with consistency and confidence." *Advances in neural information processing systems* 33 (NeurIPS), **2020**.

Method

FixMatch*

❑ Implementation

- Load data

```
labeled_dataset, unlabeled_dataset, test_dataset = DATASET_GETTERS[args.dataset](args, './data')
```

```
labeled_trainloader = DataLoader(
    labeled_dataset,
    sampler=train_sampler(labeled_dataset),
    batch_size=args.batch_size,
    num_workers=args.num_workers,
    drop_last=True)

unlabeled_trainloader = DataLoader(
    unlabeled_dataset,
    sampler=train_sampler(unlabeled_dataset),
    batch_size=args.batch_size * args.mu,
    num_workers=args.num_workers,
    drop_last=True)

test_loader = DataLoader(
    test_dataset,
    sampler=SequentialSampler(test_dataset),
    batch_size=args.batch_size,
    num_workers=args.num_workers)
```

*Sohn, Kihyuk, et al. "Fixmatch: Simplifying semi-supervised learning with consistency and confidence." *Advances in neural information processing systems* 33 (NeurIPS), **2020**.

Method

FixMatch*

- Implementation
  - Training

```
train(args, labeled_trainloader, unlabeled_trainloader, test_loader, model, optimizer, ema_model, scheduler)
```

```
2 usages (1 dynamic)
def train(args, labeled_trainloader, unlabeled_trainloader, test_loader, model, optimizer, ema_model, scheduler):
    if args.amp:
        from apex import amp
    global best_acc
    test_accs = []
    end = time.time()

    if args.world_size > 1:
        labeled_epoch = 0
        unlabeled_epoch = 0
        labeled_trainloader.sampler.set_epoch(labeled_epoch)
        unlabeled_trainloader.sampler.set_epoch(unlabeled_epoch)

    labeled_iter = iter(labeled_trainloader)
    unlabeled_iter = iter(unlabeled_trainloader)
```

```
for batch_idx in range(args.eval_step):
    # read labeled data
    try:
        inputs_x, targets_x = labeled_iter.next()
    except:
        if args.world_size > 1:
            labeled_epoch += 1
            labeled_trainloader.sampler.set_epoch(labeled_epoch)
        labeled_iter = iter(labeled_trainloader)
        inputs_x, targets_x = labeled_iter.next()

    # read unlabeled data
    try:
        (inputs_u_w, inputs_u_s), _ = unlabeled_iter.next()

    except:
        if args.world_size > 1:
            unlabeled_epoch += 1
            unlabeled_trainloader.sampler.set_epoch(unlabeled_epoch)
        unlabeled_iter = iter(unlabeled_trainloader)
        (inputs_u_w, inputs_u_s), _ = unlabeled_iter.next()
```

*Sohn, Kihyuk, et al. "Fixmatch: Simplifying semi-supervised learning with consistency and confidence." *Advances in neural information processing systems* 33 (NeurIPS), **2020**.

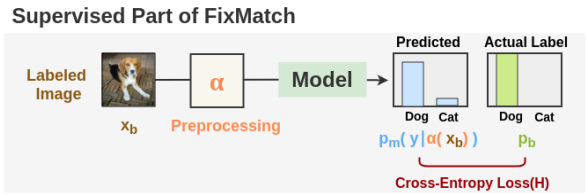
Topic 2: Semi-supervised Learning

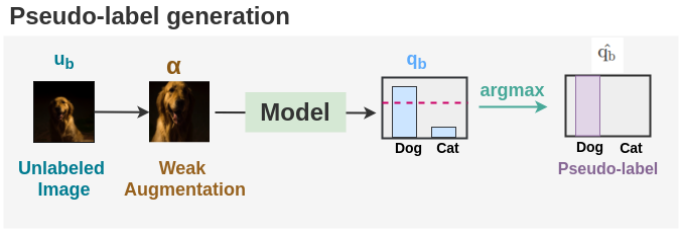
FixMatch*

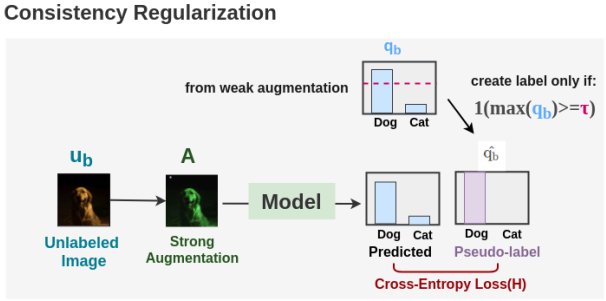
- ❑ Implementation
  - Training
    - On labeled data

Method

```
representation_labeled_data, _ = model(inputs_x)
representation_unlabeled_data_weak, _ = model(inputs_u_w)
representation_unlabeled_data_strong, _ = model(inputs_u_s)

# extract representation of unlabeled data
logits_labeled_data = model.classifier(representation_labeled_data)
logits_unlabeled_data_weak = model.classifier(representation_unlabeled_data_weak)
logits_unlabeled_data_strong = model.classifier(representation_unlabeled_data_strong)
```



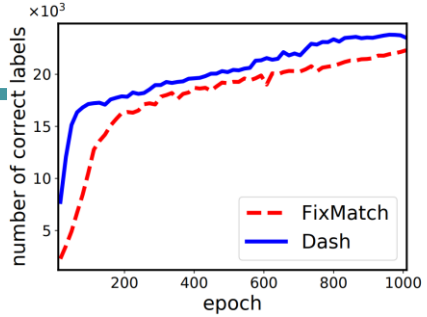


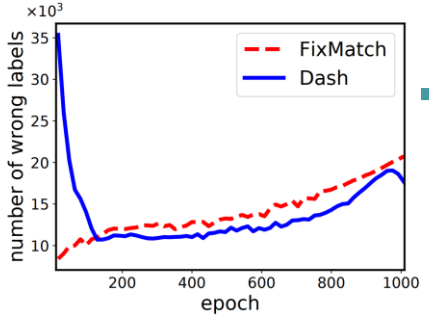



```
# Cross-entropy loss on labeled data
Lx = F.cross_entropy(logits_labeled_data, targets_x, reduction='mean')

# Semi-supervised loss on unlabeled data
pseudo_label = torch.softmax(logits_unlabeled_data_weak.detach() / args.T, dim=-1)
max_probs, targets_u = torch.max(pseudo_label, dim=-1)
mask = max_probs.ge(args.threshold).float()

Lu = (F.cross_entropy(logits_unlabeled_data_strong, targets_u, reduction='none') * mask).mean()

loss = Lx + args.lambda_u * Lu
```

*Xu, Yi, et al. "Dash: Semi-supervised learning with dynamic thresholding." *International Conference on Machine Learning*. PMLR, 2021.

DASH*





Method

- Semi-Supervised Learning with Dynamic Thresholding

Algorithm 1 Dash: Semi-Supervised Learning with **Dyn**amic Thre**sh**olding

Input: learning rate η_0 and mini-batch size m_0 for stage one, learning rate η and parameter m of mini-batch size for stage two, two parameters $C > 1$ and $\gamma > 1$ for computing threshold, and violation probability δ.

// Warm-up Stage: run SGD in T_0 iterations.

Initialization: $\mathbf{u}_0 = \mathbf{w}_0$

for $t = 0, 1, \ldots, T_0 - 1$ **do**

Sample m_0 examples $\xi_{t,i}$ $(i = 1, \ldots, m_0)$ from $\mathbf{D}_l$,

$\mathbf{u}_{t+1} = \mathbf{u}_t - \eta_0 \tilde{\mathbf{g}}_t$ where $\tilde{\mathbf{g}}_t = \frac{1}{m_0} \sum_{i=1}^{m_0} \nabla f_s(\mathbf{u}_t; \xi_{t,i})$

end for

// Selection Stage: run SGD in T iterations.

Initialization: $\mathbf{w}_1 = \mathbf{u}_{T_0}$.

Compute the value of $\widehat{\rho}$ as in (16). // In practice, $\widehat{\rho}$ can be obtained as in (17).

for $t = 1, \ldots, T$ **do**

1) Sample $n_t = m\gamma^{t-1}$ examples from $\mathbf{D}_u$, where the pseudo labels in $\mathbf{D}_u$ are generated by FixMatch

2) Set the threshold $\rho_t = C\gamma^{-(t-1)}\widehat{\rho}$.

3) Compute truncated stochastic gradient $\mathbf{g}_t$ as (18).

4) Update solution by SGD using stochastic gradient $\mathbf{g}_t$ and learning rate η: $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \mathbf{g}_t$.

end for

Output: $\mathbf{w}_{T+1}$

$$\widehat{\rho} \approx \frac{1}{|\mathbf{D}_l|} \sum_{\xi_i \in \mathbf{D}_l} f(\mathbf{w}_1; \xi_i), \tag{17}$$

THANK YOU