AIL 862

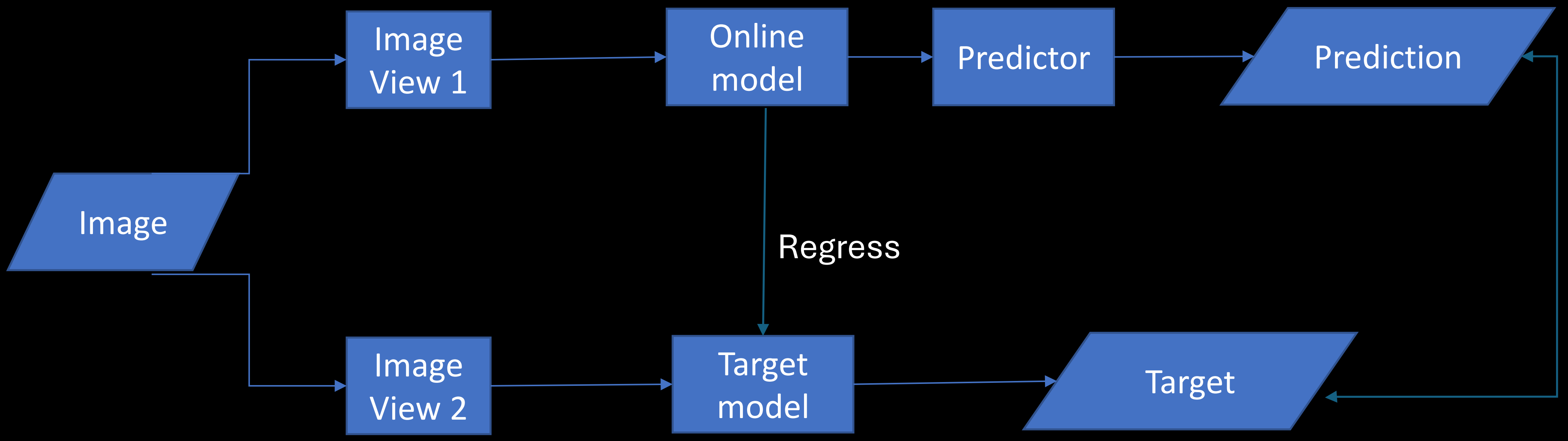
Lecture 16 and 17

BYOL recap

BYOL

- Image: generate two views
- Two different networks: online network and target network
- One input is processed through the online network and the other through the target network

BYOL - mechanism



Algorithm 1: BYOL: **B**ootstrap **Y**our **O**wn **L**atent

Inputs :

$\mathcal{D}$, $\mathcal{T}$, and $\mathcal{T}'$	set of images and distributions of transformations
θ, f_θ, g_θ, and q_θ	initial online parameters, encoder, projector, and predictor
ξ, f_ξ, g_ξ	initial target parameters, target encoder, and target projector
optimizer	optimizer, updates online parameters using the loss gradient
K and N	total number of optimization steps and batch size
$\{\tau_k\}_{k=1}^{K}$ and $\{\eta_k\}_{k=1}^{K}$	target network update schedule and learning rate schedule

for $k = 1$ **to** K **do**

- $\mathcal{B} \leftarrow \{x_i \sim \mathcal{D}\}_{i=1}^{N}$ `// sample a batch of` N `images`
- **for** $x_i \in \mathcal{B}$ **do**
    - $t \sim \mathcal{T}$ and $t' \sim \mathcal{T}'$ `// sample image transformations`
    - $z_1 \leftarrow g_\theta(f_\theta(t(x_i)))$ and $z_2 \leftarrow g_\theta(f_\theta(t'(x_i)))$ `// compute projections`
    - $z_1' \leftarrow g_\xi(f_\xi(t'(x_i)))$ and $z_2' \leftarrow g_\xi(f_\xi(t(x_i)))$ `// compute target projections`
    - $l_i \leftarrow -2 \cdot \left(\frac{\langle q_\theta(z_1), z_1' \rangle}{\|q_\theta(z_1)\|_2 \cdot \|z_1'\|_2} + \frac{\langle q_\theta(z_2), z_2' \rangle}{\|q_\theta(z_2)\|_2 \cdot \|z_2'\|_2} \right)$ `// compute the loss for` x_i
- **end**
- $\delta\theta \leftarrow \frac{1}{N} \sum_{i=1}^{N} \partial_\theta l_i$ `// compute the total loss gradient w.r.t.` θ
- $\theta \leftarrow \mathrm{optimizer}(\theta, \delta\theta, \eta_k)$ `// update online parameters`
- $\xi \leftarrow \tau_k \xi + (1 - \tau_k)\theta$ `// update target parameters`

end

Output : encoder f_θ

Ack:
https://theaisummer.com/byol/

```python
class Augment:
    """
    A stochastic data augmentation module
    Transforms any given data example randomly
    resulting in two correlated views of the same example,
    denoted x ĩ and x j̃, which we consider as a positive pair.
    """

    def __init__(self, img_size, s=1):
        color_jitter = T.ColorJitter(
            0.8 * s, 0.8 * s, 0.8 * s, 0.2 * s
        )
        blur = T.GaussianBlur((3, 3), (0.1, 2.0))

        self.train_transform = T.Compose([
            T.ToTensor(),
            T.RandomResizedCrop(size=img_size),
            T.RandomHorizontalFlip(p=0.5),  # with 0.5 probability
            T.RandomApply([color_jitter], p=0.8),
            T.RandomApply([blur], p=0.5),
            T.RandomGrayscale(p=0.2),
            # imagenet stats
            T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
        ])
    def __call__(self, x):
        return self.train_transform(x), self.train_transform(x),
```

```python
    self.net = net
    self.student_model = AddProjHead(model=net, in_features=in_features,
                                     layer_name=layer_name,
                                     embedding_size=projection_size,
                                     hidden_size=projection_hidden_size,
                                     batch_norm_mlp=batch_norm_mlp)
    self.use_momentum = use_momentum
    self.teacher_model = self._get_teacher()
    self.target_ema_updater = EMA(moving_average_decay)
    self.student_predictor = MLP(projection_size, projection_size, projection_hidden_size)


@torch.no_grad()
def _get_teacher(self):
    return copy.deepcopy(self.student_model)
```

```python
# student projections: backbone + MLP projection
student_proj_one = self.student_model(image_one)
student_proj_two = self.student_model(image_two)

# additional student's MLP head called predictor
student_pred_one = self.student_predictor(student_proj_one)
student_pred_two = self.student_predictor(student_proj_two)

with torch.no_grad():
    # teacher processes the images and makes projections: backbone + MLP
    teacher_proj_one = self.teacher_model(image_one).detach_()
    teacher_proj_two = self.teacher_model(image_two).detach_()

loss_one = loss_fn(student_pred_one, teacher_proj_one)
loss_two = loss_fn(student_pred_two, teacher_proj_two)

return (loss_one + loss_two).mean()
```

```python
def loss_fn(x, y):
    # L2 normalization
    x = F.normalize(x, dim=-1, p=2)
    y = F.normalize(y, dim=-1, p=2)
    return 2 - 2 * (x * y).sum(dim=-1)
```

```python
class EMA():
    def __init__(self, alpha):
        super().__init__()
        self.alpha = alpha

    def update_average(self, old, new):
        if old is None:
            return new
        return old * self.alpha + (1 - self.alpha) * new
```

Fine-tuning with small dataset

1% means fine-tuned with only 1% of ImageNet's training set

Method	Top-1		Top-5	
	1%	10%	1%	10%
Supervised [77]	25.4	56.4	48.4	80.4
InstDisc	-	-	39.2	77.4
PIRL [35]	-	-	57.2	83.8
SimCLR [8]	48.3	65.6	75.5	87.8
BYOL (ours)	**53.2**	**68.8**	**78.4**	**89.0**

(a) ResNet-50 encoder.

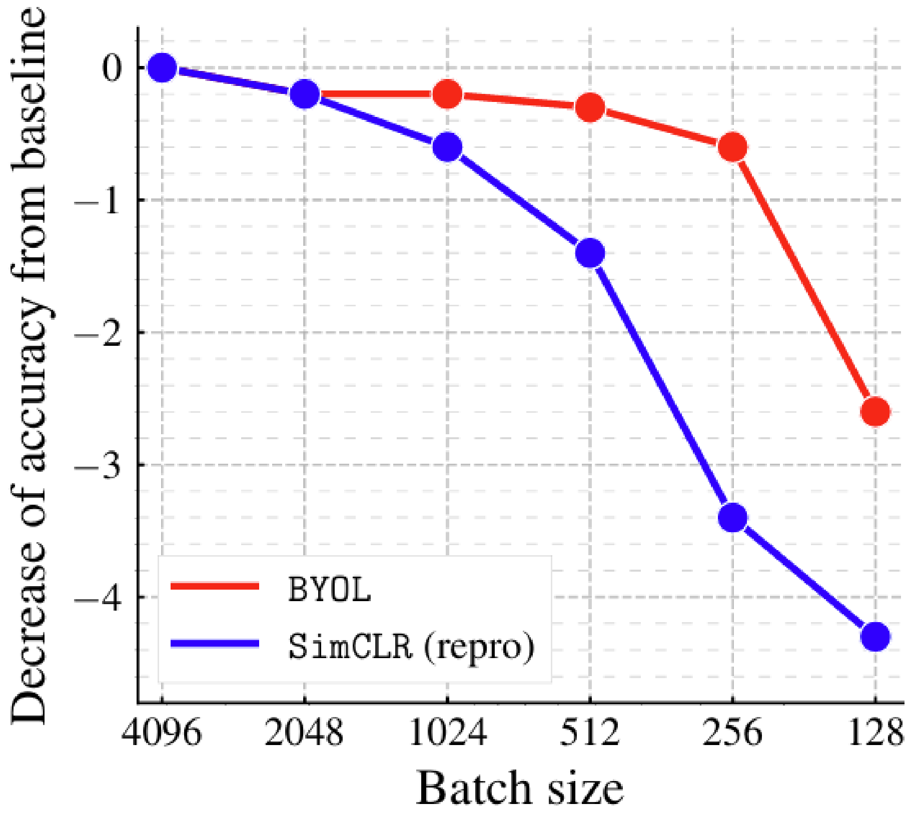
BYOL - sensitive

- To augmentation choice
- Projection dimension
- To batch size

Projector g_θ output dim	Top-1	Top-5
16	$69.9_{\pm 0.3}$	89.9
32	71.3	90.6
64	72.2	90.9
128	72.5	91.0
256	72.5	90.8
512	**72.6**	**91.0**

(b) Projection dimension.

Batch Size



(a) Impact of batch size

Moving ahead to DINO

DINO

- SSL with ViT backbone

DINO

Algorithm 1 DINO PyTorch pseudocode w/o multi-crop.

```
# gs, gt: student and teacher networks
# C: center (K)
# tps, tpt: student and teacher temperatures
# l, m: network and center momentum rates
gt.params = gs.params
for x in loader: # load a minibatch x with n samples
    x1, x2 = augment(x), augment(x) # random views

    s1, s2 = gs(x1), gs(x2) # student output n-by-K
    t1, t2 = gt(x1), gt(x2) # teacher output n-by-K

    loss = H(t1, s2)/2 + H(t2, s1)/2
    loss.backward() # back-propagate

    # student, teacher and center updates
    update(gs) # SGD
    gt.params = l*gt.params + (1-l)*gs.params
    C = m*C + (1-m)*cat([t1, t2]).mean(dim=0)

def H(t, s):
    t = t.detach() # stop gradient
    s = softmax(s / tps, dim=1)
    t = softmax((t - C) / tpt, dim=1) # center + sharpen
    return - (t * log(s)).sum(dim=1).mean()
```

Augmentations

- Like BYOL
- Local to global correspondence

No BN

- DINO with ViT backbone is BN-free

Avoiding collapse

- The center c is updated with an exponential moving average

Performance gap with supervised training

Method	Arch.	Param.	im/s	Linear	k-NN
Supervised	RN50	23	1237	79.3	79.3
SCLR [12]	RN50	23	1237	69.1	60.7
MoCov2 [15]	RN50	23	1237	71.1	61.9
InfoMin [67]	RN50	23	1237	73.0	65.3
BarlowT [81]	RN50	23	1237	73.2	66.0
OBoW [27]	RN50	23	1237	73.8	61.9
BYOL [30]	RN50	23	1237	74.4	64.8
DCv2 [10]	RN50	23	1237	75.2	67.1
SwAV [10]	RN50	23	1237	**75.3**	65.7
DINO	RN50	23	1237	**75.3**	**67.5**
Supervised	ViT-S	21	1007	79.8	79.8
BYOL* [30]	ViT-S	21	1007	71.4	66.6
MoCov2* [15]	ViT-S	21	1007	72.7	64.4
SwAV* [10]	ViT-S	21	1007	73.5	66.3
DINO	ViT-S	21	1007	**77.0**	**74.5**

What we are looking for?

- Good features

How can we use good features?

How can we use good features?

- Nearest neighbor retrieval

Copy detection

Copy detection. We also evaluate the performance of ViTs trained with DINO on a copy detection task. We report the mean average precision on the "strong" subset of the INRIA Copydays dataset [21]. The task is to recognize images that have been distorted by blur, insertions, print and scan, etc. Following prior work [5], we add 10k distractor images randomly sampled from the YFCC100M dataset [66]. We perform copy detection directly with cosine similarity on the features obtained from our pretrained network. The features

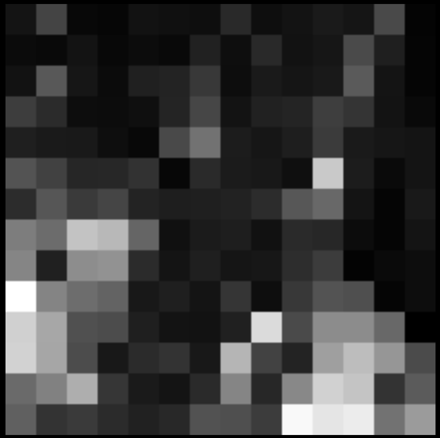
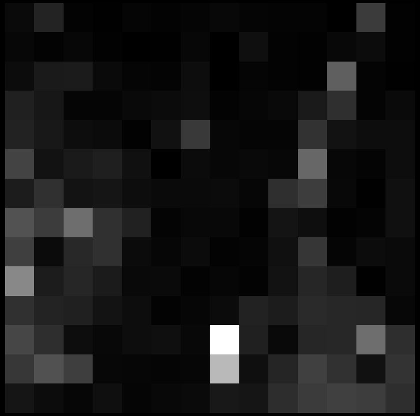
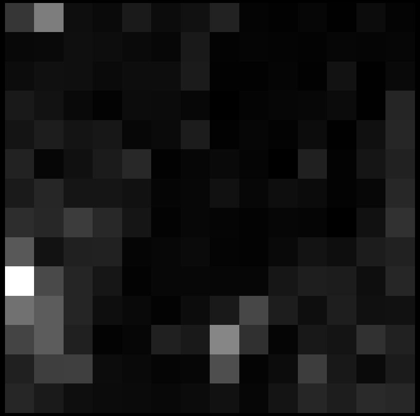
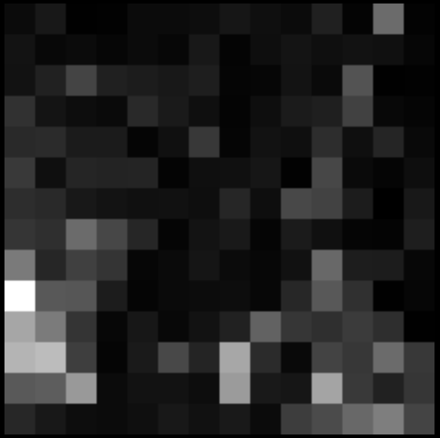
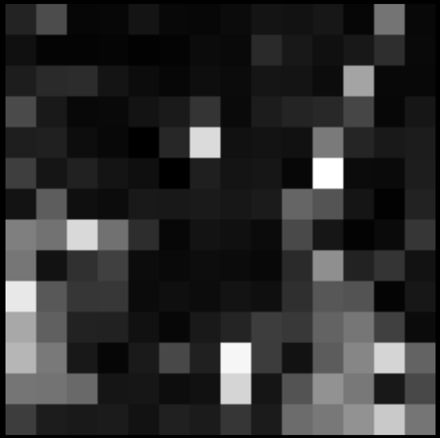
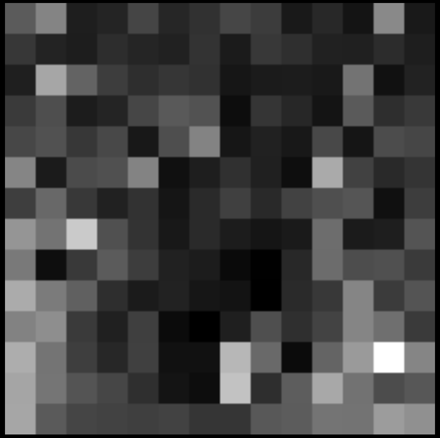
Table 4: **Copy detection.** We report the mAP performance in copy detection on Copydays "strong" subset [21]. For reference, we also report the performance of the multigrain model [5], trained specifically for particular object retrieval.

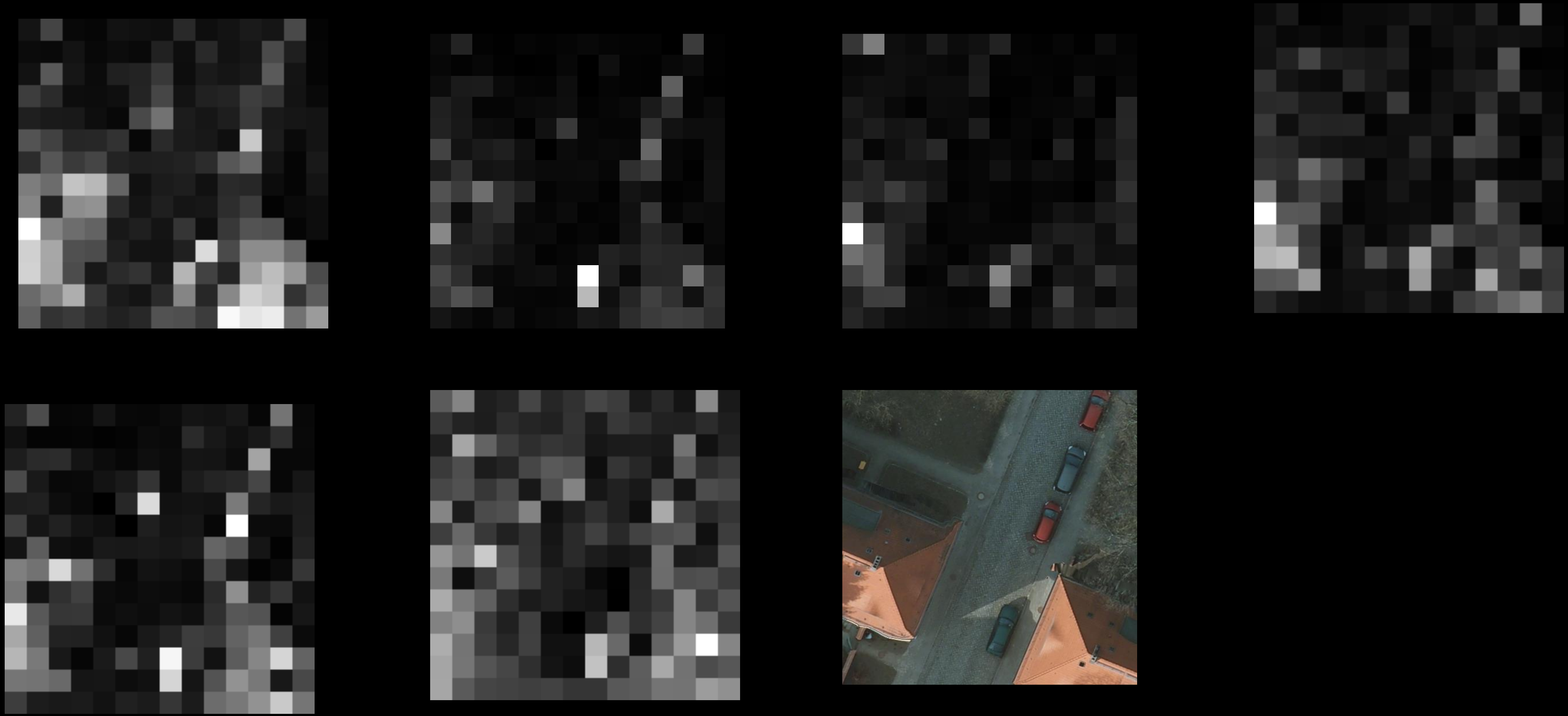
Method	Arch.	Dim.	Resolution	mAP
Multigrain [5]	ResNet-50	2048	224^2	75.1
Multigrain [5]	ResNet-50	2048	largest side 800	82.5
Supervised [69]	ViT-B/16	1536	224^2	76.4
DINO	ViT-B/16	1536	224^2	81.7
DINO	ViT-B/8	1536	320^2	**85.5**

Can we semantic segmentation

- Good features – clustering – semantic segmentation

```python
## Attention
attentions = model.get_last_selfattention(inputImage)
print(attentions.shape)

## getting number of heads and w_featmap and h_featmap
numberOfHead = attentions.shape[1]
w_featmap = inputImage.shape[-2] // vitPatchSize
h_featmap = inputImage.shape[-1] // vitPatchSize


# we keep only the output patch attention
attentions = attentions[0, :, 0, 1:].reshape(numberOfHead, -1)

print(attentions.shape)

attentions = attentions.reshape(numberOfHead, w_featmap, h_featmap)

print(attentions.shape)

attentions = nn.functional.interpolate(attentions.unsqueeze(0), scale_factor=vitPatchSize, mode="nearest")[0].cpu().detach().numpy()

print(attentions.shape)

for attentionIter in range(numberOfHead):
        fname = os.path.join('./result/savedAttentionMaps/', "attn-head" + str(attentionIter) + ".png")

        attentionMapThisHead = attentions[attentionIter]

        print(np.amax(attentionMapThisHead))
        print(np.amin(attentionMapThisHead))

        attentionMapThisHeadNormalized = (attentionMapThisHead - np.amin(attentionMapThisHead)) / (np.amax(attentionMapThisHead) - np.amin(attentionMapThisHead))


        attentionMapThisHeadNormalized = np.expand_dims(attentionMapThisHeadNormalized, axis=2)


        attentionMapThisHeadNormalizedForDisplay = np.repeat(attentionMapThisHeadNormalized,3,axis=2)


        plt.imsave(fname=fname, arr=attentionMapThisHeadNormalizedForDisplay, format='png')
        print(f"{fname} saved.")
```

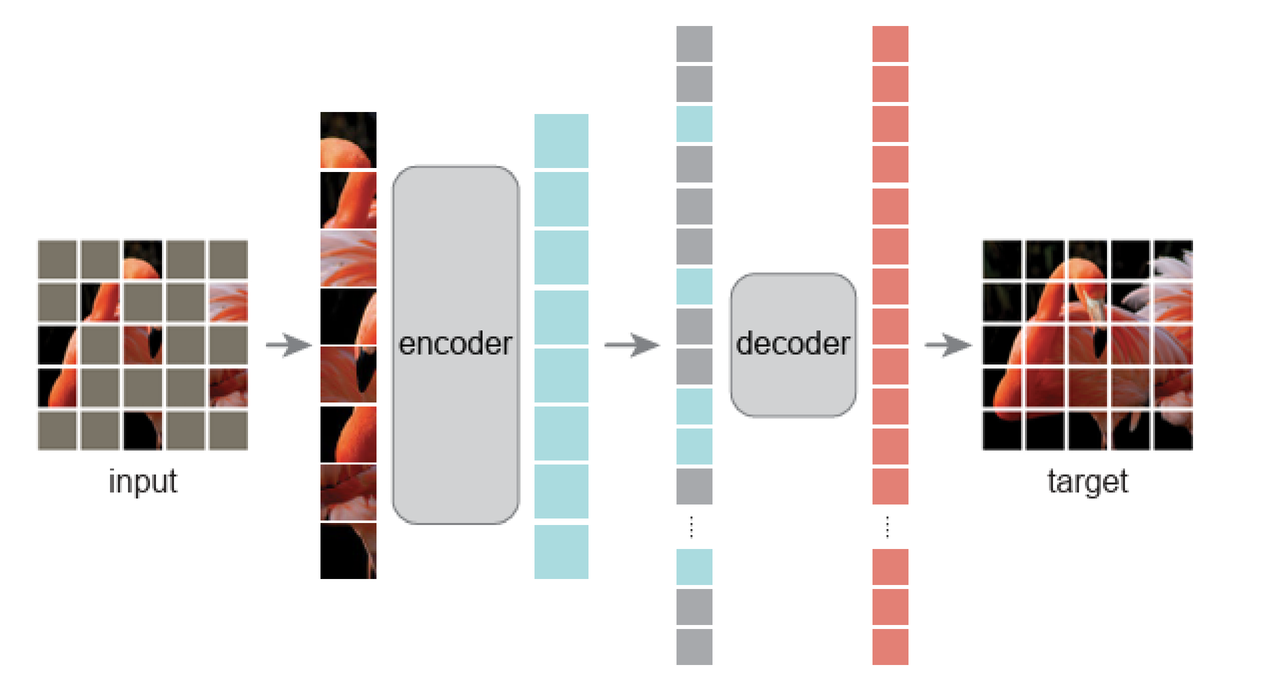


Autoencoder

Denoising autoencoder

Masked image modeling

MAE



MAE encoder

- Just like a standard ViT

- However, operates only on a small subset of the full set of the patches

Non-overlapping patches

- Why?

Non-overlapping patches

- Overlapping patches introduce redundancy.

- Non-overlapping patches enforce a stronger learning signal since the model must infer missing parts without redundant information.

MAE decoder

- The input to the MAE decoder is the full set of tokens consisting of encoded visible patches and mask tokens.
- Positional embeddings are added to all tokens in this full set.
- The decoder has another series of Transformer blocks.

Reconstruction target

Reconstructs the input by predicting the pixel values for each masked patch. Each element in the decoder’s output is a vector of pixel values representing a patch.

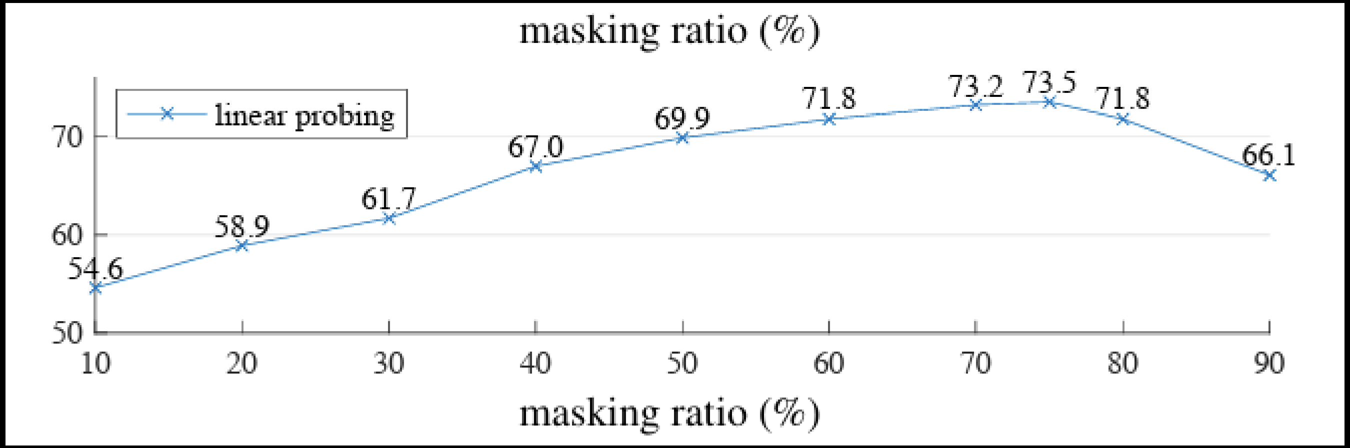
The last layer of the decoder is a linear projection whose number of output channels equals the number of pixel values in a patch.

The decoder’s output is reshaped to form a reconstructed image.

Loss function computes the (MSE between the reconstructed and original images in the pixel space.

Masking ratio

Masking ratio



Comparison to supervision

to overfit. The following is a comparison between ViT-L trained from scratch *vs*. fine-tuned from our baseline MAE:

scratch, original [16]	scratch, our impl.	baseline MAE
76.5	82.5	84.9