

Under review as a conference paper at ICLR 2023

IMAGE AS SET OF POINTS

Anonymous authors

Paper under double-blind review

问题：图像的可解释性以及如何提取特征

主要对比：

1、ConvNets：将图像视为矩形形状的organized像素，并通过**局部区域**中的卷积运算提取特征

2、ViTs：将图像视为一系列patch，并通过全局范围内的**注意力**机制提取特征

缺点：一味的改进卷积和注意力会被引诱到追求增量改进的陷阱中，应从其他角度入手

提出：Context clusters (CoCs)

整体而言，我们将图像视为一组数据点，并将所有点分组为簇。

然后，每个簇中的特征点将被聚合，然后被分派回去。

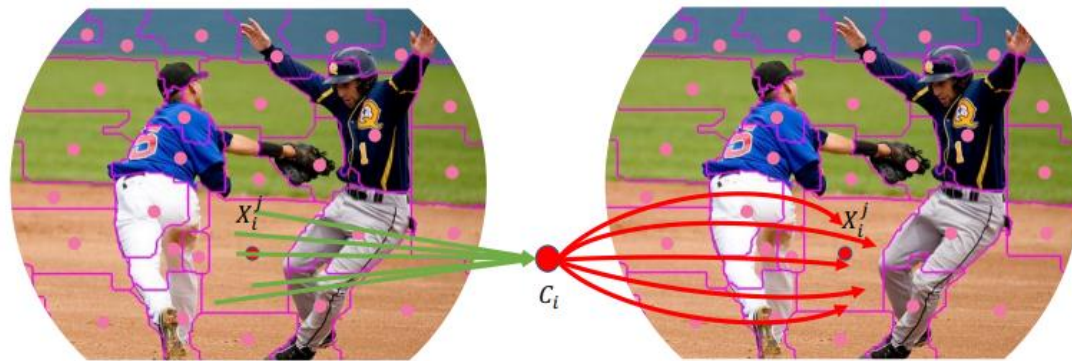


Figure 1: A context cluster in our network trained for image classification. We view *an image as a set of points* and sample c centers for points clustering. Point features are aggregated and then dispatched within a cluster. For cluster center C_i , we first aggregated all points $\{x_i^0, x_i^1, \dots, x_i^n\}$ in i th cluster, then the aggregated result is distributed to all points in the clusters dynamically. See § 3 for details.

From Image to Set of Points->Context Clusters block->Context Cluster architectures

From Image to Set of Points. given an input image $\mathbf{I} \in \mathbb{R}^{3 \times w \times h}$, we begin by enhancing the image with the 2D coordinates of each pixel $\mathbf{I}_{i,j}$, where each pixel's coordinate is presented as $\left[\frac{i}{w} - 0.5, \frac{j}{h} - 0.5\right]$. It is feasible to investigate further positional augmentation techniques to potentially improve performance. This design is taken into consideration for its simplicity and practicality. The augmented image is then converted to a collection of points (*i.e.*, pixels) $\mathbf{P} \in \mathbb{R}^{5 \times n}$, where $n = w \times h$ is the number of points, and each point contains both feature (color) and position (coordinates) information; hence, the points set could be unordered and disorganized.



Given a set of feature points $\mathbf{P} \in \mathbb{R}^{n \times d}$

linearly project \mathbf{P} to \mathbf{P}_s

SuperPixel method

均匀地提出了空间中的 c 个中心,
并通过平均其 k 个最近点来计算中心特征

计算 \mathbf{P}_s 与生成的中心点集之间的余弦相似矩阵
the pair-wise cosine similarity matrix $\mathbf{S} \in \mathbb{R}^{c \times n}$

将每个点分配给最相似的中心, 从而产生 c 个簇

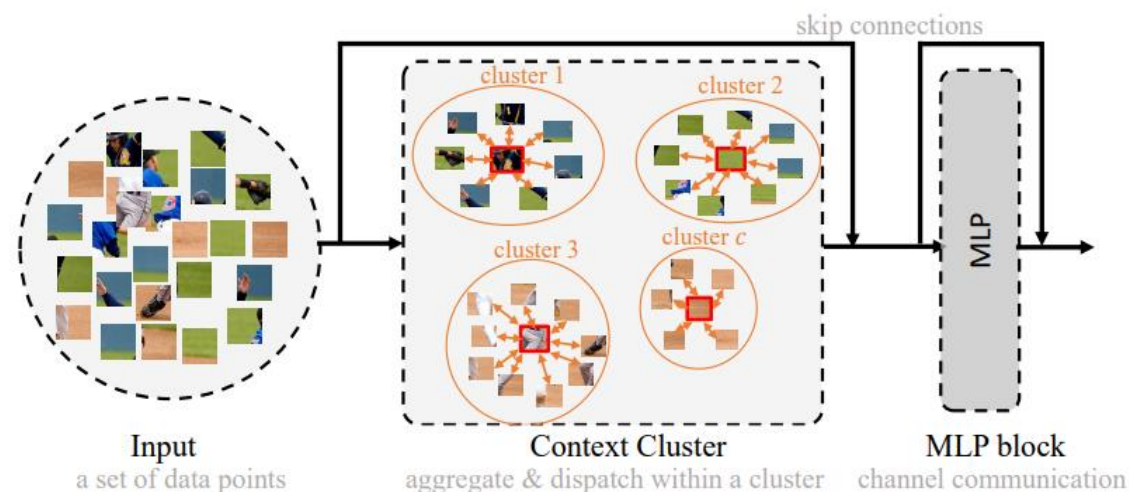


Figure 2: A Context Cluster block. We use a context cluster operation to first group a set of unorganized data points; then, we communicate the points within clusters. After that, an MLP block (2 FC layers with a GELU activation between) is applied. The original input image comes from Fig. 1.

Feature Aggregating. We dynamically aggregate all points in a cluster based on the similarities to the center point. Assuming a cluster contains m points (a subset in \mathbf{P}) and the similarity between the m points and the center is $s \in \mathbb{R}^m$ (a subset in \mathbf{S}), we map the points to a value space to get

$P_v \in \mathbb{R}^{m \times d'}$, where d' is the value dimension. We also propose a center v_c in the value space like the clustering center proposal. The aggregated feature $g \in \mathbb{R}^{d'}$ is given by:

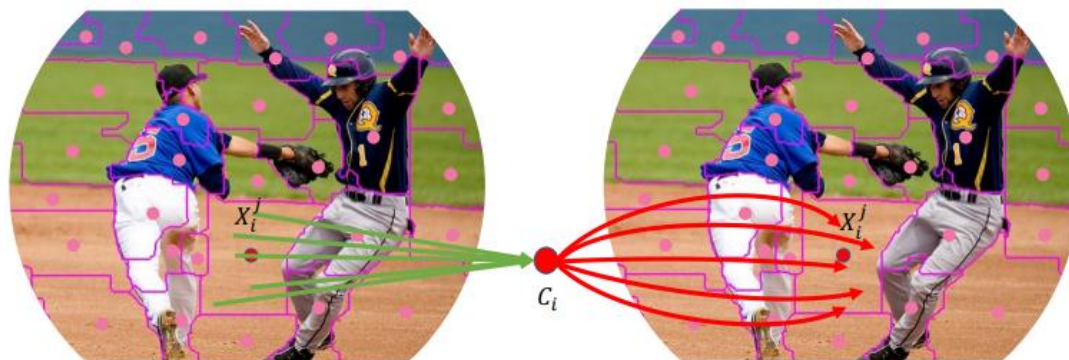
$$g = \frac{1}{C} \left(v_c + \sum_{i=1}^m \text{sig}(\alpha s_i + \beta) * v_i \right), \quad \text{s.t., } C = 1 + \sum_{i=1}^m \text{sig}(\alpha s_i + \beta). \quad (1)$$

Here α and β are learnable scalars to scale and shift the similarity and $\text{sig}(\cdot)$ is a sigmoid function to re-scale the similarity to $(0, 1)$. v_i indicates i -th point in P_v . Empirically, this strategy would achieve much better results than directly applying the original similarity because no negative value is involved. Softmax is not considered since the points do not contradict with one another. We incorporate the value center v_c in Eq. 1 for numerical stability¹ as well as further emphasize the locality. To control the magnitude, the aggregated feature is normalized by a factor of C .

Feature Dispatching. The aggregated feature g is then adaptively dispatched to each point in a cluster based on the similarity. By doing so, the points can communicate with one another and shares features from all points in the cluster, as shown in Fig. 1. For each point p_i , we update it by

$$p'_i = p_i + \text{FC}(\text{sig}(\alpha s_i + \beta) * g). \quad (2)$$

Here we follow the same procedures to handle the similarity and apply a fully-connected (FC) layer to match the feature dimension (from value space dimension d' to original dimension d).



对于分类，我们对最后一个块的输出的所有点进行平均，并使用FC层进行分类。

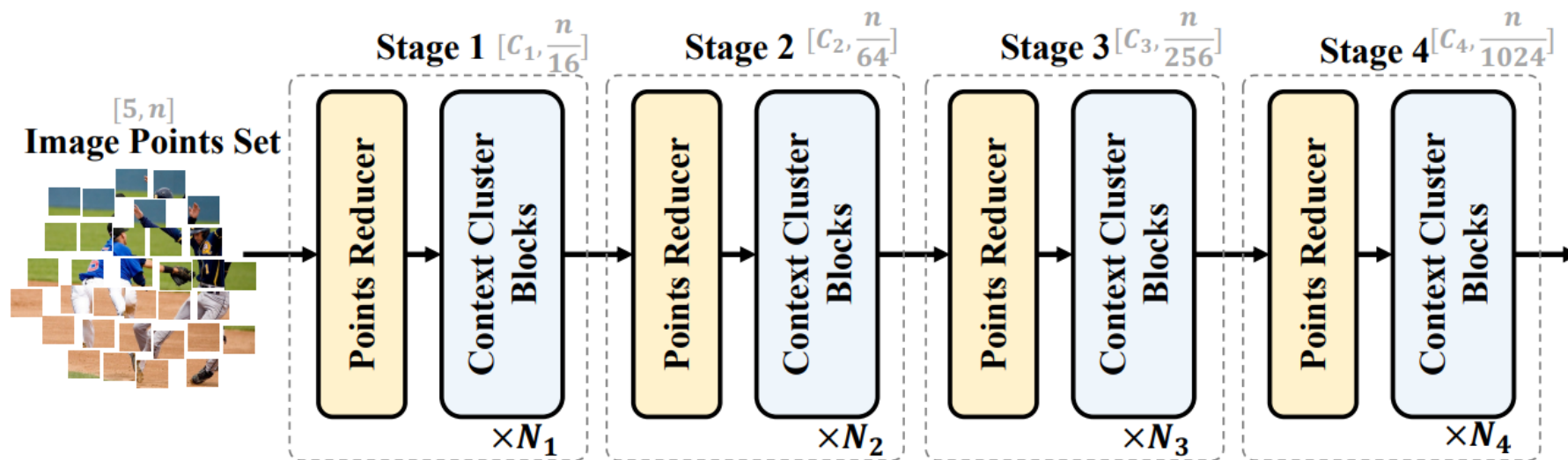


Figure 3: Context Cluster architecture with four stages. Given a set of image points, Context Cluster gradually reduces the point number and extracts deep features. Each stage begins with a points reducer, after which a succession of context cluster blocks is used to extract features.

While Context Cluster is fundamentally distinct from convolution and attention, the design philosophies from ConvNets and ViTs, such as hierarchical representations and meta Transformer architecture (Yu et al., 2022c), are still applicable to Context Cluster. To align with other networks and make our method compatible with most detection and segmentation algorithms, we progressively reduce the number of points by a factor of 16, 4, 4, and 4 in each stage. We consider 16 nearest neighbors for selected anchors in the first stage, and we choose their 9 nearest neighbors in the rest stages.

An underlying issue is computational efficiency. Assume we have n d -dimensional points and c clusters, the time complexity to calculate the feature similarity would be $O(ncd)$, which is unacceptable when the input image resolution is high (*e.g.*, 224×224). To circumvent this problem, we introduce region partition by splitting the points into several local regions like Swin Transformer (Liu et al., 2021b), and compute similarity locally. As a result, when the number of local regions is set to r , we noticeably lower the time complexity by a factor of r , from $O(ncd)$ to $O\left(r^{\frac{n}{r}} \frac{c}{r} d\right)$. See appendix § A

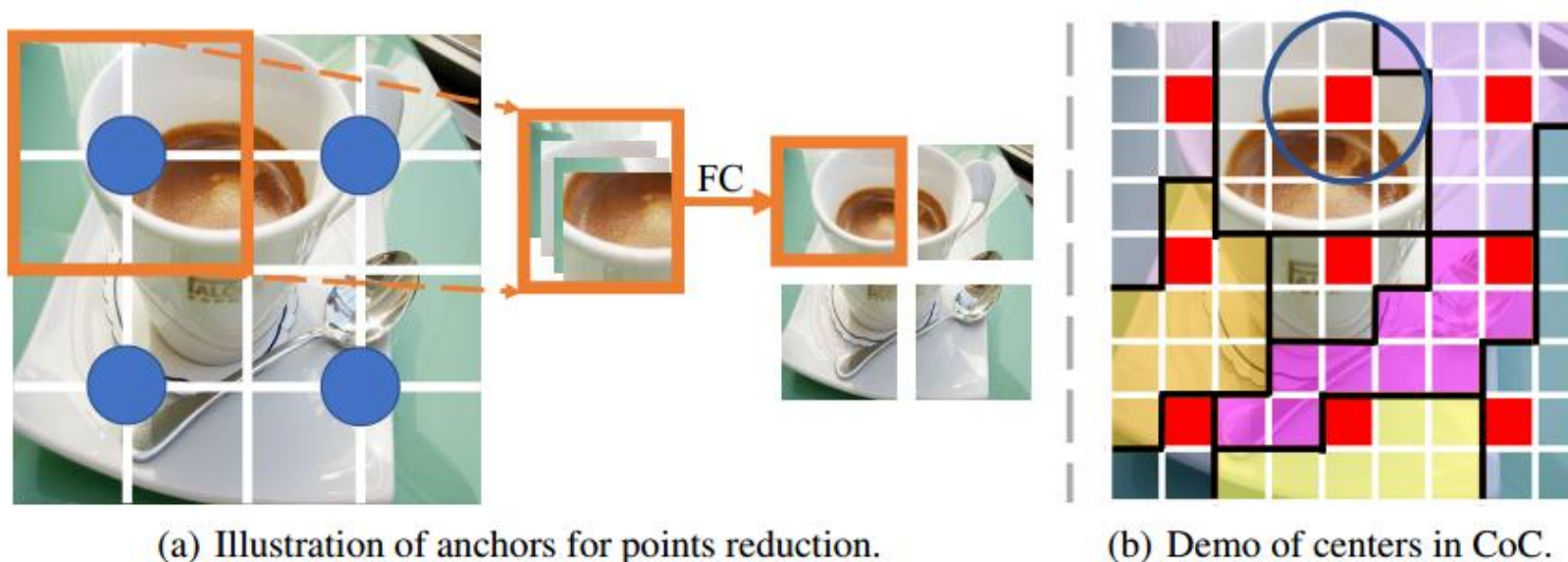


Figure 5: Detail explanations on the anchors in points reducer block and centers for context cluster block. The image points (represented in patch) are organized for easy understanding and illustration. In a sense, the anchors are for reducing point numbers, and centers are used for clustering. Both of them are evenly distributed in design. **On the left**, we evenly propose 4 anchors (marked in blue dot) with 4 neighbors for each anchor. **On the right**, we evenly sample 9 centers (marked in red block), hence leading to 9 irregular clusters. The center feature value is achieved by averaging its k neighbors. In this figure, we show the neighbors in the big blue circle for the second center.

Table 1: Comparison with representative small backbones on ImageNet-1k benchmark. Throughput (images / s) is measured on a single V100 GPU with a batch size of 128, and is averaged by the last 500 iterations. All models are trained and tested at 224×224 resolution, except ViT-B and ViT-L.

	Method	Param.	GFLOPs	Top-1	Throughputs (images/s)
MLP	♣ ResMLP-12 (Touvron et al., 2021a)	15.0	3.0	76.6	511.4
	♣ ResMLP-24 (Touvron et al., 2021a)	30.0	6.0	79.4	509.7
	♣ ResMLP-36 (Touvron et al., 2021a)	45.0	8.9	79.7	452.9
	♣ MLP-Mixer-B/16 (Tolstikhin et al., 2021)	59.0	12.7	76.4	400.8
	♣ MLP-Mixer-L/16 (Tolstikhin et al., 2021)	207.0	44.8	71.8	125.2
	♣ gMLP-Ti (Liu et al., 2021a)	6.0	1.4	72.3	511.6
	♣ gMLP-S (Liu et al., 2021a)	20.0	4.5	79.6	509.4
Attention	♦ ViT-B/16 (Dosovitskiy et al., 2020)	86.0	55.5	77.9	292.0
	♦ ViT-L/16 (Dosovitskiy et al., 2020)	307	190.7	76.5	92.8
	♦ PVT-Tiny (Wang et al., 2021)	13.2	1.9	75.1	-
	♦ PVT-Small (Wang et al., 2021)	24.5	3.8	79.8	-
	♦ T2T-ViT-7 (Yuan et al., 2021a)	4.3	1.1	71.7	-
	♦ DeiT-Tiny/16 (Touvron et al., 2021b)	5.7	1.3	72.2	523.8
	♦ DeiT-Small/16 (Touvron et al., 2021b)	22.1	4.6	79.8	521.3
Convolution	♠ ResNet18 (He et al., 2016)	12	1.8	69.8	584.9
	♠ ResNet50 (He et al., 2016)	26	4.1	79.8	524.8
	♠ ConvMixer-512/16 (Trockman et al., 2022)	5.4	-	73.8	-
	♠ ConvMixer-1024/12 (Trockman et al., 2022)	14.6	-	77.8	-
	♠ ConvMixer-768/32 (Trockman et al., 2022)	21.1	-	80.16	142.9
Cluster	♥ Context-Cluster-Ti _(ours)	5.3	1.0	71.8	518.4
	♥ Context-Cluster-Ti _‡ _(ours)	5.3	1.0	71.7	510.8
	♥ Context-Cluster-Small _(ours)	14.0	2.6	77.5	513.0
	♥ Context-Cluster-Medium _(ours)	27.9	5.5	81.0	325.2

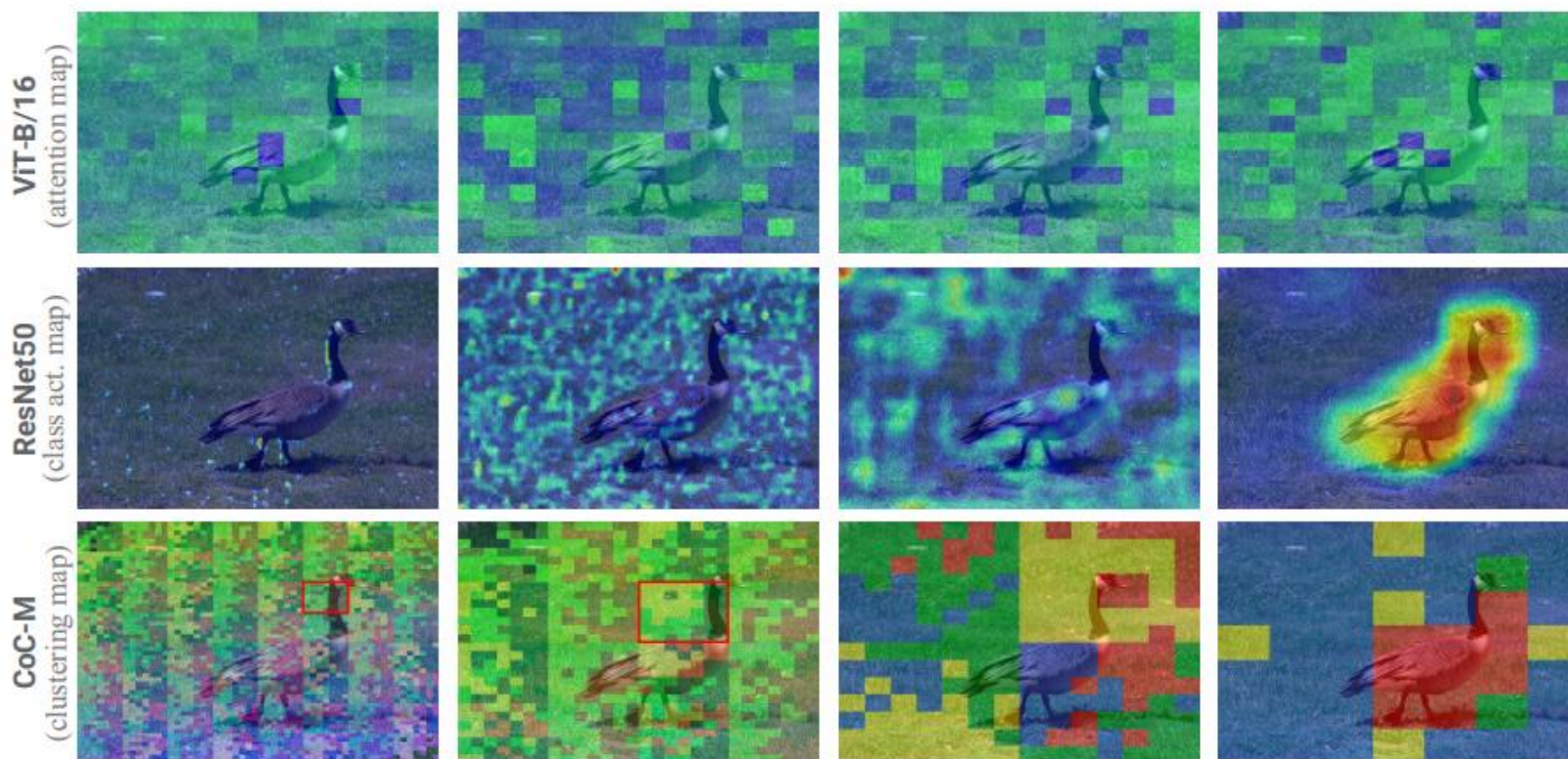


Figure 4: Visualization of activation map, class activation map, and clustering map for ViT-B/16, ResNet50, and our CoC-M, respectively. We plot the results of the last block in the four stages from left to right. For ViT-B/16, we select the [3rd, 6th, 9th, 12th] blocks, and show the cosine (instead of dot-product) attention map for the `cls`-token. We randomly select a head for both ViT-B/16 and our CoC-M. The clustering map shows that our Context Cluster is able to cluster similar contexts together (please zoom in to see details), showing what model learned visually.

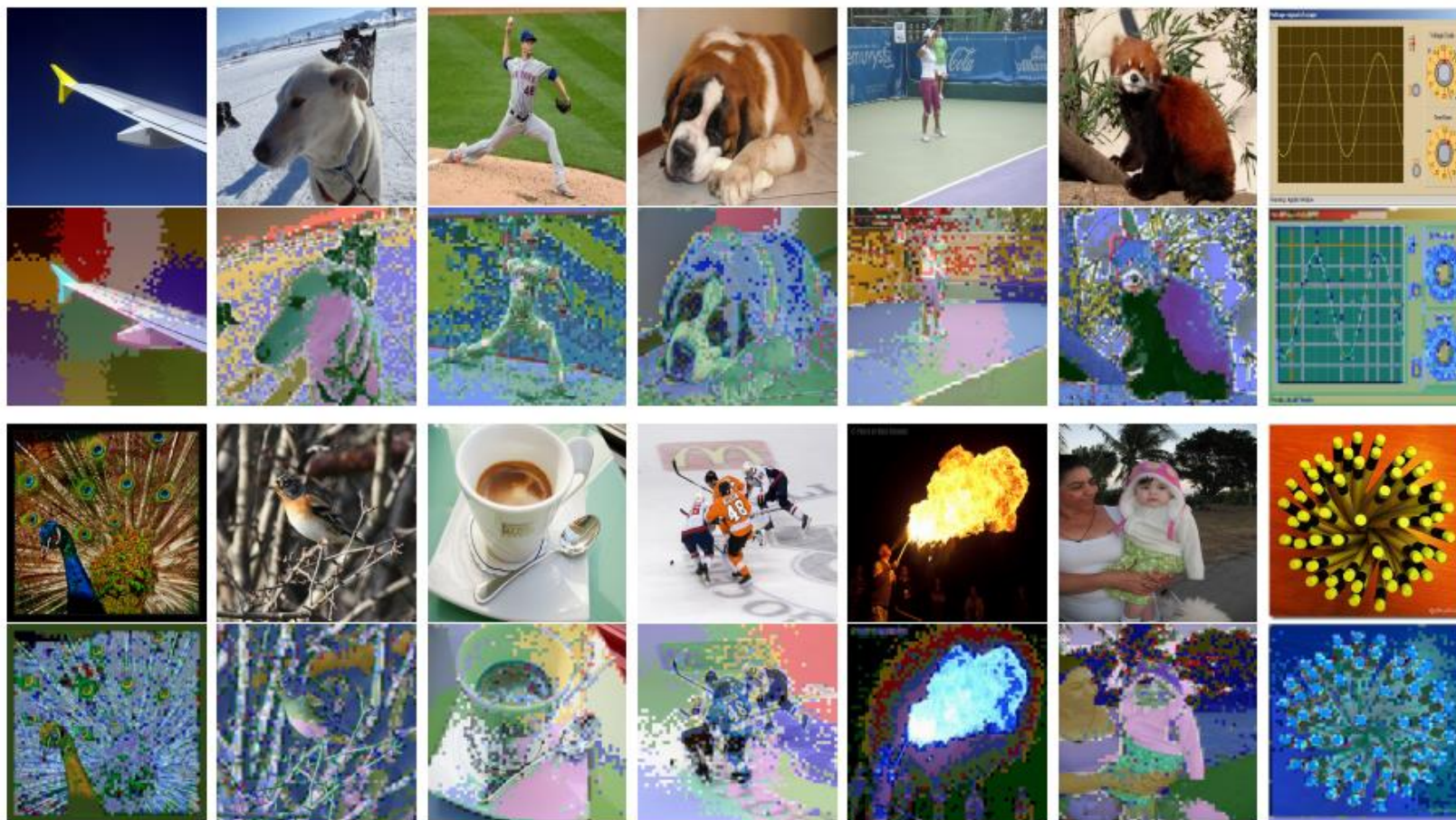






Figure 8: The clustering results of the last context cluster block in the first CoC-Tiny stage (without region partition). Without region partition, Our Context Cluster astonishingly displays "superpixel"-like clustering results, even in the early stage. we pick the most intriguing one out of the four heads.

	ResNet-50	Mixup [48]	Cutout [3]	CutMix
Image				
Label	Dog 1.0	Dog 0.5 Cat 0.5	Dog 1.0	Dog 0.6 Cat 0.4
ImageNet Cls (%)	76.3 (+0.0)	77.4 (+1.1)	77.1 (+0.8)	78.6 (+2.3)
ImageNet Loc (%)	46.3 (+0.0)	45.8 (-0.5)	46.7 (+0.4)	47.3 (+1.0)
Pascal VOC Det (mAP)	75.6 (+0.0)	73.9 (-1.7)	75.1 (-0.5)	76.7 (+1.1)

- Stage I: sample two instances $(x_i, y_i), (x_j, y_j)$ from the training set.
- Stage II: sample the interpolation factor λ from the Beta distribution $\text{Beta}(\alpha, \alpha)$.
- Stage III: mixing the sampled instances with interpolation factor λ according to the following mixing formulation:

$$x_{mix} = \lambda x_i + (1 - \lambda)x_j, y_{mix} = \lambda y_i + (1 - \lambda)y_j, \lambda \sim \text{Beta}(\alpha, \alpha).$$

$$\tilde{x} = \mathbf{M} \odot x_A + (\mathbf{1} - \mathbf{M}) \odot x_B$$

$$\tilde{y} = \lambda y_A + (1 - \lambda)y_B,$$

$$\mathbf{B} = (r_x, r_y, r_w, r_h)$$

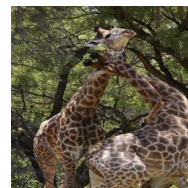
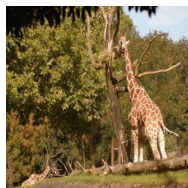
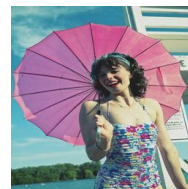
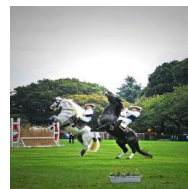
$$r_x \sim \text{Unif}(0, W), \quad r_w = W\sqrt{1 - \lambda},$$

$$r_y \sim \text{Unif}(0, H), \quad r_h = H\sqrt{1 - \lambda}$$

- 1、选择图像
- 2、Mask的设计
- 3、loss
- 4、长尾问题的采样概率问题

1、选择图像

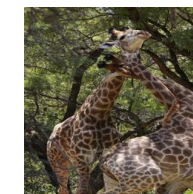
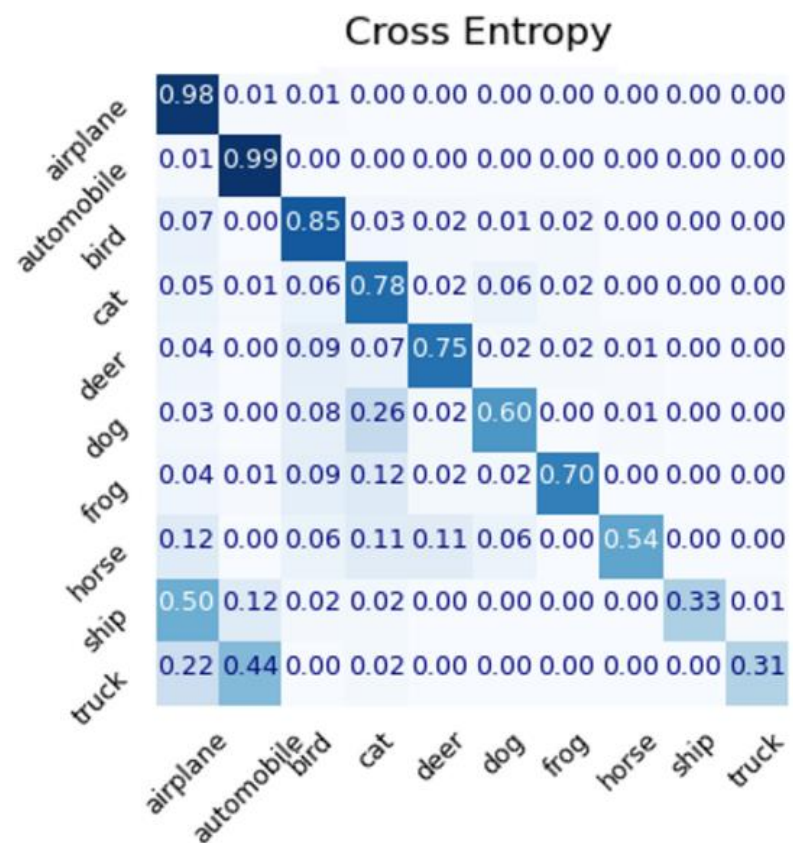
选择图像的目的：突出两张图像之间的关联



1、选择图像

突出两张图像之间的关联的方法：固定的混淆矩阵和动态的OT

混淆矩阵：



1、选择图像

v100(单卡)	cifar-100_0.01				
CIFAR-100-LT-0.01	Top1	Top5	many	med	few
cmo	41.15	69.15	68.91	39.46	10.73
cmo+直接训练得到的混淆矩阵	41.69	70.08	71.17	40.23	9
cmo+cmo训练后得到的混淆矩阵	41.93	70.06	72.3	40.9	8.8

1、选择图像



动态的OT:
根据 T_{ij} 来选图像

最优传输损失。为了从背景分布 P 中选择与前景分布 Q 关联度最匹配的概率值，我们要计算两个分布之间的 OT 距离对分布 P 进行重新加权：↵

$$OT(P, Q) = \min_{T \in \Pi(P, Q)} \langle T, C \rangle \leftarrow$$

其中 C 是成本矩阵，传输概率矩阵 T 满足 $\Pi(P, Q) := \{T | \sum_{i=1}^N T_{ij} = w_j, \sum_{j=1}^N T_{ij} = w_i\}$ 。我们采用带有熵约束的正则化 OT 距离，并将优化问题表示为：↵

$$OT_{\epsilon}(P, Q) \stackrel{\text{def}}{=} \sum_{i,j}^{N,N} C_{ij} T_{ij} - \epsilon \sum_{i,j}^{N,N} -T_{ij} \ln T_{ij} \leftarrow$$

其中 $\epsilon > 0$ ， $C \in R_{\geq 0}^{N \times N}$ 是运输损失矩阵， C_{ij} 表示背景样本 i 和前景样本 i 之间的损失。想要最小化计算两个分布之间的成本损失需先计算成本矩阵。成本矩阵的计算过程如下：↵

$$\begin{aligned} C_{ij} &= \text{Dis} \left(f(x_i^f; \theta), f(x_i^b; \theta) \right) \leftarrow \\ &= 1 - \cos \left(f(x_i^f; \theta), f(x_i^b; \theta) \right) \leftarrow \end{aligned}$$

其中 $\text{Dis}(\cdot)$ 表示余弦相似度的距离度量， $f(x_i^f; \theta)$ 和 $f(x_i^b; \theta)$ 分别是前景样本和背景样本的特征向量， θ 同时受到两个特征空间的影响。↵

1、选择图

Algorithm 1 CMO+OT

Require: Dataset D_{back} , model parameter θ , Q , hyper-parameters $\{\alpha, \beta\}$, loss function $L(\cdot)$.

```
1: Randomly initialize  $\theta$ .  
2: Sample weighted dataset  $D_{fore} \sim Q$ .  
3: for epoch = 1, 2, ...,  $T$  do  
4:   for batch  $i = 1, 2, \dots, B$  do  
5:     Draw a mini-batch  $(x_i^b, y_i^b)$  from  $D_{back}$   
6:     Draw a mini-batch  $(x_i^f, y_i^f)$  from  $D_{fore}$   
7:     Compute  $L_{ot}(f(x_i^f; \theta), f(x_i^b; \theta))$  with (1) and  $T_{ij}$  with (2)  
8:      $\lambda \sim \text{Beta}(\alpha, \alpha)$   
9:     for  $t = 1, 2, \dots, i$  do  
10:       $index = \text{ArgMax}(T_{ij}, t)$   
11:       $\tilde{x}_{i,t} = \mathbf{M} \odot x_{i,index}^b + (\mathbf{1} - \mathbf{M}) \odot x_{i,t}^f$   
12:       $\tilde{y}_{i,t} = \lambda y_{i,index}^b + (1 - \lambda) y_{i,t}^f$   
13:    end for  
14:    Update  $\theta \leftarrow \theta - \beta \nabla_{\theta} (L(f(\tilde{x}_i; \theta), \tilde{y}_i) + L_{ot}(f(x_i^f; \theta), f(x_i^b; \theta)))$   
15:  end for  
16: end for
```


1、选择图像



```
b, c, w, h = input.size()
f = model(input)
f2 = model(input2)
cost, Pi, C = sinkhorn(f, f2)

input3 = torch.zeros_like(input2)
target3 = torch.zeros_like(target)
Pi = Pi.cpu().detach().numpy()
lam = np.random.beta(args.beta, args.beta)
bbx1, bby1, bbx2, bby2 = rand_bbox(input.size(), lam)
for k in range(b):
    index = np.argmax(Pi[:, k], axis=None)
    input3[k, :, :, :] = input[index, :, :, :]
    input3[k, :, bbx1:bbx2, bby1:bby2] = input2[k, :, bbx1:bbx2, bby1:bby2]
    target3[k] = target[index]

lam = 1 - ((bbx2 - bbx1) * (bby2 - bby1) / (input.size()[-1] * input.size()[-2]))
# compute output
output = model(input3)
loss = criterion(output, target3) * lam + criterion(output, target2) * (1. - lam) + cost
```

1、选择图像

	Imbalance ratio	Top1	Top5	many	med	few
CE + CMO	100	42.5	70.25	70.77	40.91	11.37
	50	46.16	75.19			
	10	60.13	86.61			
CE + OT (argmax+Ti[k])	100	42.03	70.32	68.89	40.74	12.2
CE + OT (argmax+Ti[:k])	100	42.8	72.19	70.06	40.4	13.8
	50	46.94	77.57			
	10	59.63	87.06			
CE + OT (argmin+Ti[:k])	100	42.12	70.45	68.26	40.94	13
CE + OT (argmax+Ti[:k]+cost)	100	42.29	72.74	68.74	40.74	13.23
	50	47.59	77.78			
	10	60.35	87.19			
CE + OT (argmax+Ti[:k]+cost+DataParallel8)	100	45.23	74.41	71.57	44.6	15.23
	50	49.82	79.29			
	10	61.18	87.47			

1、选择图像

根据 C_{ij} 来选图像

v100(单卡)	cifar-100_0.01					
	Top1	Top5	many	med	few	
cmo	41.86	70.28	69.83	40.26	11.1	work1
	42.47	70.58	68.94	42.23	11.87	work2
	42.81	70.28	71.09	41.23	11.67	work3
	42.38	70.38	69.95	41.24	11.55	
input+row	41.99	71.24	69.89	40.83	10.8	work4
	41.27	70.61	69.03	40.56	9.37	work5
	42.34	70.38	70.06	41.06	11.5	work6
	41.87	70.74	69.66	40.82	10.56	
input+column	41.02	69.73	68.03	40.29	10.37	work7
	41.79	69.7	70.11	40.11	10.7	work8
	41.92	69.93	68.23	41.57	11.63	work9
	41.58	69.79	68.79	40.66	10.9	
feature+row	42.36	70.59	70.63	41.14	10.8	work10
	41.95	70.67	69.17	41.43	10.8	work11
	41.47	70.52	69.69	40.89	9.23	work12
	41.93	70.59	69.83	41.15	10.28	
feature+column	42.67	72.95	69.94	40.77	13.07	work13
	43.04	72.47	70.29	41.46	13.1	work14
	43.44	73.27	70.06	42.43	13.57	work15
	43.05	72.9	70.1	41.55	13.25	

AlignMixup: Improving Representations By Interpolating Aligned Features

Shashanka Venkataramanan¹ Ewa Kijak¹ Laurent Amsaleg¹ Yannis Avrithis²

¹Inria, Univ Rennes, CNRS, IRISA ²Athena RC

$$\text{Mix}_{\lambda}^{f_1, f_2}(x, x') := f_2(\text{Mix}_{\lambda}(f_1(x), f_1(x')))$$

$$\text{feature}(\mathbf{A}) : f_1 := F, f_2 := \text{id}, \quad \mathbf{A} = F(x)$$

$$P^* = \arg \min_{P \in U_r} \langle P, M \rangle - \epsilon H(P),$$

$$\text{The assignment matrix } R := rP^*$$

$$\tilde{A} := A'R^{\top}$$

$$\tilde{A}' := AR.$$

$$\text{mix}_{\lambda}(\mathbf{A}, \tilde{\mathbf{A}}).$$

$$\text{mix}_{\lambda}(\mathbf{A}', \tilde{\mathbf{A}}').$$

```
def forward(self, x, targets, lam, mode):
```

```
    if (mode == 'train'):
```

```
        layer_mix = random.randint(0,1)
```

```
        if layer_mix == 0:
```

```
            x,t_a,t_b = mixup_process(x, targets, lam)
```

```
        out = self.encoder(x, lin=0, lout=0)
```

```
        out = self.encoder.layer1(out)
```

```
        out = self.encoder.layer2(out)
```

```
        out = self.encoder.layer3(out)
```

```
        out = self.encoder.layer4(out)
```

```
        if layer_mix == 1:
```

```
            out,t_a,t_b = mixup_aligned(out, targets, lam)
```

```
        out = F.avg_pool2d(out, 4)
```

```
        out = out.reshape(out.size(0), -1)
```

```
        cls_output = self.classifier(out)
```

```
    return cls_output, t_a, t_b
```

```
def mixup_aligned(out, y, lam):
```

```
    # out shape = batch_size x 512 x 4 x 4 (cifar10/100)
```

```
    indices = np.random.permutation(out.size(0))
```

```
    feat1 = out.view(out.shape[0], out.shape[1], -1) # batch_size x 512 x 16
```

```
    feat2 = out[indices].view(out.shape[0], out.shape[1], -1) # batch_size x 512 x 16
```

```
    sinkhorn = SinkhornDistance(eps=0.1, max_iter=100, reduction=None)
```

```
    P = sinkhorn(feat1.permute(0,2,1), feat2.permute(0,2,1)).detach() # optimal plan batch x 16 x 16
```

```
    P = P*(out.size(2)*out.size(3)) # assignment matrix
```

```
    align_mix = random.randint(0,1) # uniformly choose at random, which alignmix to perform
```

```
    if (align_mix == 0):
```

```
        #  $\tilde{A} = A'R^T$ 
```

```
        f1 = torch.matmul(feat2, P.permute(0,2,1).cuda()).view(out.shape)
```

```
        final = feat1.view(out.shape)*lam + f1*(1-lam)
```

```
    elif (align_mix == 1):
```

```
        #  $\tilde{A}' = AR$ 
```

```
        f2 = torch.matmul(feat1, P.cuda()).view(out.shape).cuda()
```

```
        final = f2*lam + feat2.view(out.shape)*(1-lam)
```

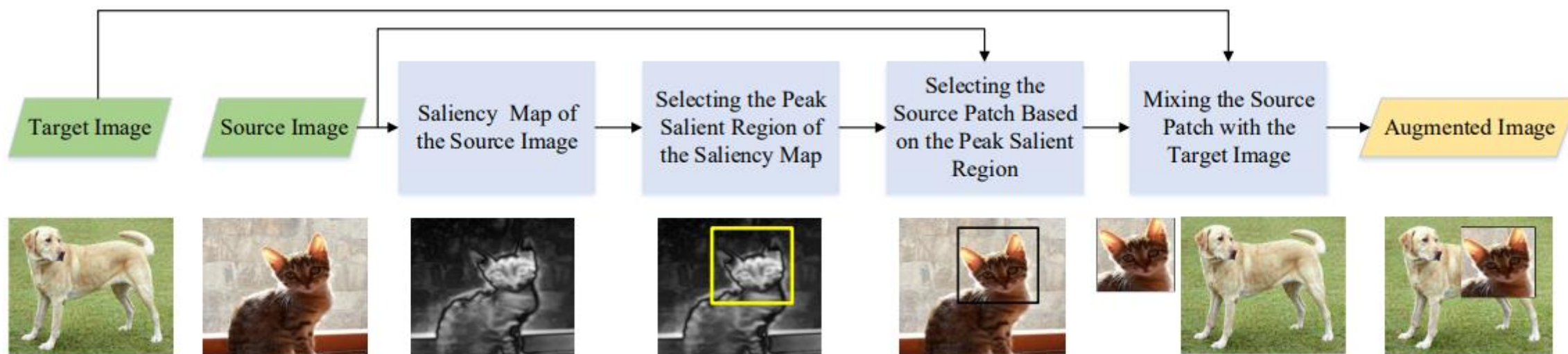
```
    y_a, y_b = y,y[indices]
```

```
    return final, y_a, y_b
```

- 1、 OT计算特征->输入空间
OT计算特征->特征空间
- 2、 OT计算的特征是特征提取器+分类器得到的

2、Mask的设计+3、loss

显著性区域/感兴趣区域 $\rightarrow (cx, cy)$



$$I_{vs} = f(I_s),$$

$$i, j = \operatorname{argmax}(I_{vs}),$$

2、Mask的设计+3、loss



```
def sal_corr(img, lam):  
  
    size = img.size()  
    W = size[1]  
    H = size[2]  
    cut_rat = np.sqrt(1. - lam)  
    cut_w = int(W * cut_rat)  
    cut_h = int(H * cut_rat)  
  
    temp_img = img.cpu().numpy().transpose(1, 2, 0)  
    saliency = cv2.saliency.StaticSaliencyFineGrained_create()  
    (success, saliencyMap) = saliency.computeSaliency(temp_img)  
    saliencyMap = (saliencyMap * 255).astype("uint8")  
  
    maximum_indices = np.unravel_index(np.argmax(saliencyMap, axis=None), saliencyMap.shape)  
    cx = maximum_indices[0]  
    cy = maximum_indices[1]  
  
    bbx1 = np.clip(cx - cut_w // 2, 0, W)  
    bby1 = np.clip(cy - cut_h // 2, 0, H)  
    bbx2 = np.clip(cx + cut_w // 2, 0, W)  
    bby2 = np.clip(cy + cut_h // 2, 0, H)  
  
    return bbx1, bby1, bbx2, bby2
```


2、Mask的设计+3、loss



```
def predict_region_corr_rand_bbox(input2, input2_pre, lam, gpu):

    input2_pre = input2_pre.reshape(1, 1, input2_pre.shape[0], -1)
    output = F.interpolate(input2_pre, (input2.shape[1], input2.shape[2]), mode="bilinear", align_corners=False)
    output = output.reshape(output.shape[2], output.shape[3])

    output = nn.Sigmoid()(output)
    size = output.size()
    W = size[0]
    H = size[1]
    cut_rat = np.sqrt(1. - lam)
    cut_w = int(W * cut_rat)
    cut_h = int(H * cut_rat)

    maximum_indices = np.unravel_index(np.argmax(output.cpu().detach().numpy(), axis=None), output.shape)
    cx = maximum_indices[0]
    cy = maximum_indices[1]

    bbx1 = np.clip(cx - cut_w // 2, 0, W)
    bby1 = np.clip(cy - cut_h // 2, 0, H)
    bbx2 = np.clip(cx + cut_w // 2, 0, W)
    bby2 = np.clip(cy + cut_h // 2, 0, H)

    return bbx1, bby1, bbx2, bby2
```

2、Mask的设计+3、loss



```
def rewrite_CE(output, target3, target2, beta, num_class):  
    output = torch.nn.functional.log_softmax(output, dim=1)  
    y3 = torch.nn.functional.one_hot(target3, num_classes=num_class)  
    y2 = torch.nn.functional.one_hot(target2, num_classes=num_class)  
  
    loss3 = y3*(output + 0.000000000001)  
    loss3 = torch.sum(loss3, dim=1) * beta  
  
    loss2 = y2*(output + 0.000000000001)  
    loss2 = torch.sum(loss2, dim=1) * (1. - beta)  
  
    loss = loss3 + loss2  
    loss = torch.mean(loss, dim=0)  
    hand_cross_entropy = -1*loss  
    return hand_cross_entropy
```

2、Mask的设计+3、loss



```
input3 = torch.zeros_like(input2)
target3 = torch.zeros_like(target)
C = C.cpu().detach().numpy()
#lam = np.random.beta(args.beta, args.beta)
#bbx1, bby1, bbx2, bby2 = rand_bbox(input.size(), lam)
for k in range(b):
    index = np.argmin(C[:, k], axis=None)
    input3[k, :, :, :] = input[index, :, :, :]
    lam = np.random.beta(args.beta, args.beta)
    #bbx1, bby1, bbx2, bby2 = predict_region_corr_rand_bbox(input2[k], input2_pre[k], lam, args.gpu)
    #bbx1, bby1, bbx2, bby2 = sal_corr(input2[k], lam)
    #input3[k, :, bbx1:bbx2, bby1:bby2] = input2[k, :, bbx1:bbx2, bby1:bby2]
    bbx1_1, bby1_1, bbx2_1, bby2_1, bbx1_2, bby1_2, bbx2_2, bby2_2 = sal_non_sal_rand_bbox(input2[k], input[index], lam)
    input3[:, :, bbx1_2:bbx2_2, bby1_2:bby2_2] = input2[:, :, bbx1_1:bbx2_1, bby1_1:bby2_1]
    target3[k] = target[index]

    #proportion = 1 - ((bbx2 - bbx1) * (bby2 - bby1) / (input.size()[-1] * input.size()[-2]))
    proportion = 1 - ((bbx2_1 - bbx1_1) * (bby2_1 - bby1_1) / (input.size()[-1] * input.size()[-2]))
    proportions.append(proportion)

# compute output
output = model(input3)
#loss = criterion(output, target3) * lam + criterion(output, target2) * (1. - lam) + cost
proportions = torch.tensor(proportions).cuda(args.gpu)
loss = rewrite_CE(output, target3, target2, proportions, output.shape[1]) + cost + cost_img1
```

2、Mask的设计+3、loss

v100(单卡)	cifar-100_0.01				
	Top1	Top5	many	med	few
base	41.15	69.15	68.91	39.46	10.73
base+sal_corr	42.23	69.96	69.34	42.34	10.47
base+sal_sal	41.31	70.42	69.54	40.66	9.13
base+sal_non_sal	41.16	69.25	68.43	40.89	9.67
base+non_sal_sal	41.01	69.25	68.97	40.6	8.87
base+non_sal_non_sal	40.27	69.1	67.6	39.46	9.3

predict_corr	41.93	71.9	69.29	40.37	11.83
	42.49	71.83	70.29	40.51	12.37
sal_corr	41.6	71.74	68.69	40.34	11.47
修改了loss之后的					
sal_corr	42.26	71.73	69.17	40.74	12.63
调整lam的位置	42.49	72.81	69.57	40.89	12.77
	42.91	72.36	70.63	41.49	12.23
predict_corr	41.84	71.33	69.34	41.6	10.03
	41.57	70.91	69.34	39.77	11.27
sal_nonsal	38.7	67.52	68.2	37.14	6.1

4、长尾问题的采样概率问题



Let n_k be the number of samples in the k -th class, then for the C classes, the total number of samples is $N = \sum_{k=1}^C n_k$. Then, the generalized sampling probability for the k -th class can be defined by

$$q(r, k) = \frac{1/n_k^r}{\sum_{k'=1}^C 1/n_{k'}^r}, \quad (2)$$

where the k -th class has a sampling weight inversely proportional to n_k^r . As r increases, the weight of the minor class becomes increasingly larger than that of the major class. By adjusting the value of r , we can examine diverse sampling strategies. Setting $r = 1$ uses the inverse class frequency [22, 51] while setting $r = 1/2$ uses the smoothed inverse class frequency, as in [35, 36]. We can also use the effective number [12] instead of n_k^r , which is defined as

Before detailing how to combine oversampling of minority classes with MixUp regularization, we introduce some further notation to describe sampling strategies. Given a training set $\mathcal{D} = \{(x_i, y_i), i = 1, \dots, N\}$ for a multi-class problem with K classes, if each class k contains n_k examples, we have that $\sum_{k=1}^K n_k = N$. We can then describe common data sampling strategies as follows:

$$p_j = \frac{n_j^q}{\sum_{k=1}^K n_k^q}, \quad (2)$$

being p_j the probability of sampling from class j during training. With this, selecting $q = 1$ amounts to picking examples with a probability equal to the frequency of their class in the training set (*instance-based sampling*), whereas choosing $q = 0$ leads to a uniform probability $p_j = 1/K$ of sampling from each class, this is, *class-based sampling* or oversampling of minority classes. Another popular choice is *square-root sampling*, which stems from selecting $q = 1/2$.