道路工程深度学习技术

第八周 Transformer 实践 2—图像数据处理

童峥

东南大学 交通学院

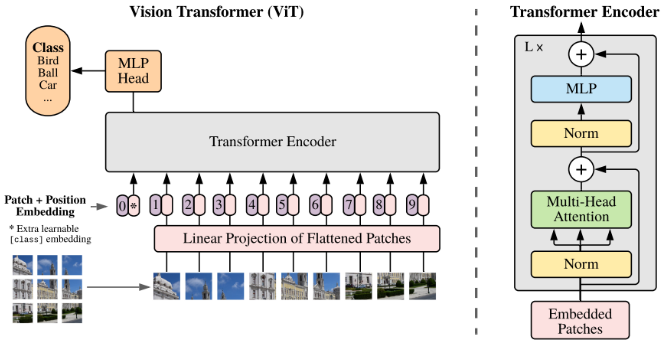
2023 年 4 月 25 日

1. ViT 图像分类模型
   - 模型原理
   - 代码讲解
2. Segmentation Transformer 模型

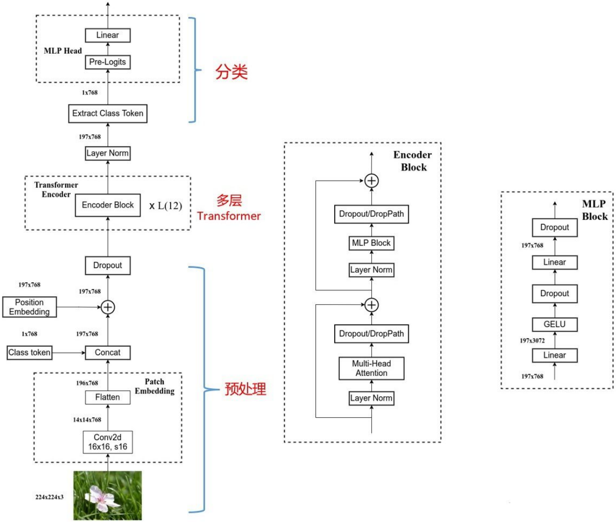
1. ViT 图像分类模型
   - 模型原理
   - 代码讲解
2. Segmentation Transformer 模型

Vision Transformer (ViT) 模型

- *An Image Is Worth 16*16 Words: Transformers For Image Recognition at scale*
- `https://arxiv.org/abs/2010.11929`



模型详解


Linear
MLP Head
Pre-Logits
分类
1x768
Extract Class Token
197x768
Layer Norm
Transformer Encoder
Encoder Block
x L(12)
多层 Transformer
Dropout
197x768
197x768
Position Embedding
1x768
197x768
Class token
Concat
196x768
Patch Embedding
Flatten
14x14x768
Conv2d 16x16, s16
224x224x3
预处理
Encoder Block
Dropout/DropPath
MLP Block
Layer Norm
Dropout/DropPath
Multi-Head Attention
Layer Norm
MLP Block
Dropout
197x768
Linear
Dropout
GELU
197x3072
Linear
197x768

模型超参数

- Layers 是 Transformer 编码层数量
- Hidden size 是分块时卷积升维后的通道数量
- MLP size 是多层感知机输出尺寸
- Heads 是多头注意力数量

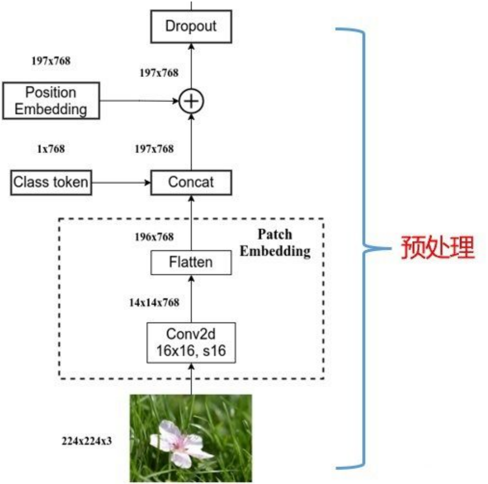
Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

1. ViT 图像分类模型
   - 模型原理
   - 代码讲解
2. Segmentation Transformer 模型

预处理模块

- 一张 $224 \times 224 \times 3$ 的图片，通过一个卷积核大小为 16×16、步长为 16、输出通道为 768 的卷积，得到 $14 \times 14 \times 768$ 的输出。
- $14 \times 14 \times 768$ 的输出，将其按照宽高进行 Flatten 操作，其尺寸变成 $196 * 768$，表示为 196 个序列，每个序列长度为 768。
- 在 196×768 的数据上，聚合一个 1×768 的分类 token 在最前面。则尺寸变成 $197 * 768$。我们设这个 197×768 的矩阵为 $\boldsymbol{A}$。
- 设置一个 $1 * 197 * 768$ 的 Position Embedding，对应值相加至 $\boldsymbol{A}$ 。

预处理模块


Dropout
197x768
197x768
Position Embedding
1x768
197x768
Class token
Concat
196x768
Patch Embedding
Flatten
14x14x768
Conv2d 16x16, s16
224x224x3
预处理

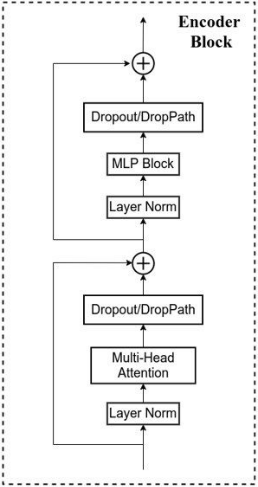
预处理模块代码

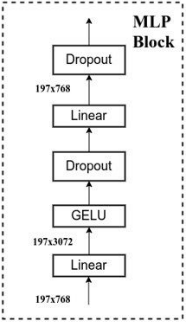
```
class PatchEmbed(nn.Module):
    """
    2D Image to Patch Embedding，二维图像patch Embedding
    """
    def __init__(self, img_size=224, patch_size=16, in_c=3, embed_dim=768, norm_layer=
        super().__init__()
        img_size = (img_size, img_size)  # 图片尺寸224*224
        patch_size = (patch_size, patch_size)  #下采样倍数，一个grid cell包含了16*16的图
        self.img_size = img_size
        self.patch_size = patch_size
        # grid_size是经过patchembed后的特征层的尺寸
        self.grid_size = (img_size[0] // patch_size[0], img_size[1] // patch_size[1])
        self.num_patches = self.grid_size[0] * self.grid_size[1] #path个数 14*14=196

        # 通过一个卷积，完成patchEmbed
        self.proj = nn.Conv2d(in_c, embed_dim, kernel_size=patch_size,stride=patch_siz
        # 如果使用了norm层，如BatchNorm2d，将通道数传入，以进行归一化，否则进行恒等映射
        self.norm = norm_layer(embed_dim) if norm_layer else nn.Identity()
```

预处理模块代码

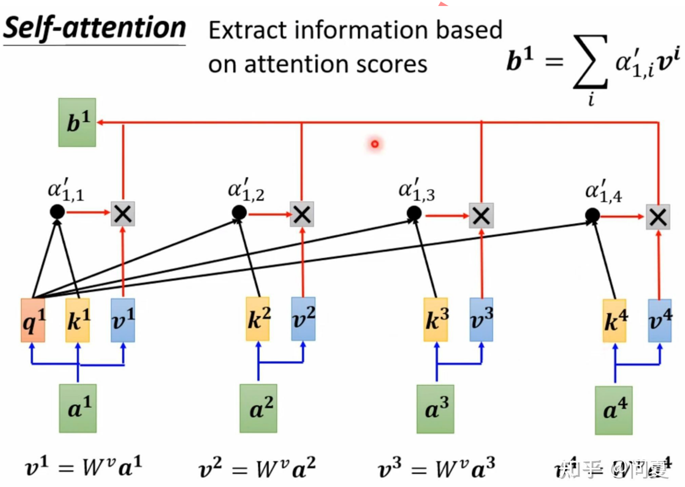
```
def forward(self, x):
    B, C, H, W = x.shape  #batch,channels,heigth,weigth
    # 输入图片的尺寸要满足既定的尺寸
    assert H == self.img_size[0] and W == self.img_size[1], \
        f"Input image size ({H}*{W}) doesn't match model ({self.img_size[0]}*{self

    # proj: [B, C, H, W] -> [B, C, H, W] , [B,3,224,224]-> [B,768,14,14]
    # flatten: [B, C, H, W] -> [B, C, HW] , [B,768,14,14]-> [B,768,196]
    # transpose: [B, C, HW] -> [B, HW, C] , [B,768,196]-> [B,196,768]
    x = self.proj(x).flatten(2).transpose(1, 2)
    x = self.norm(x)
    return x
```

Transformer 编码层





Muti-head Attention 层



Muti-head Attention 层代码

```
class Attention(nn.Module):
    """
    muti-head attention模块，也是transformer最主要的操作
    """
    def __init__(self,
                 dim,   # 输入token的dim,768
                 num_heads=8, #muti-head的head个数，实例化时base尺寸的vit默认为12
                 qkv_bias=False,
                 qk_scale=None,
                 attn_drop_ratio=0.,
                 proj_drop_ratio=0.):
        super(Attention, self).__init__()
        self.num_heads = num_heads
        head_dim = dim // num_heads  #平均每个head的维度
        self.scale = qk_scale or head_dim ** -0.5  #进行query操作时，缩放因子
        # qkv矩阵相乘操作，dim * 3使得一次性进行qkv操作
        self.qkv = nn.Linear(dim, dim * 3, bias=qkv_bias)
        self.attn_drop = nn.Dropout(attn_drop_ratio)
        self.proj = nn.Linear(dim, dim)  #一个卷积层
        self.proj_drop = nn.Dropout(proj_drop_ratio)
```

Muti-head Attention 层代码

```python
def forward(self, x):
    # [batch_size, num_patches + 1, total_embed_dim] 如 [bactn,197,768]
    B, N, C = x.shape  # N:197 , C:768

    # qkv进行注意力操作，reshape进行muti-head的维度分配，permute维度调换以便后续操作
    # qkv(): -> [batch_size, num_patches + 1, 3 * total_embed_dim] 如 [b,197,2304]
    # reshape: -> [batch_size, num_patches + 1, 3, num_heads, embed_dim_per_head] 如 [t
    # permute: -> [3, batch_size, num_heads, num_patches + 1, embed_dim_per_head]
    qkv = self.qkv(x).reshape(B, N, 3, self.num_heads, C // self.num_heads).permute(2,
    # qkv的维度相同，[batch_size, num_heads, num_patches + 1, embed_dim_per_head]
    q, k, v = qkv[0], qkv[1], qkv[2]  # make torchscript happy (cannot use tensor as tı
```

Muti-head Attention 层代码

```
# transpose: -> [batch_size, num_heads, embed_dim_per_head, num_patches + 1]
# @: multiply -> [batch_size, num_heads, num_patches + 1, num_patches + 1]
attn = (q @ k.transpose(-2, -1)) * self.scale  #矩阵相乘操作
attn = attn.softmax(dim=-1) #每一path进行softmax操作
attn = self.attn_drop(attn)

# [b,12,197,197]@[b,12,197,64] -> [b,12,197,64]
# @: multiply -> [batch_size, num_heads, num_patches + 1, embed_dim_per_head]
# 维度交换 transpose: -> [batch_size, num_patches + 1, num_heads, embed_dim_per_hea
# reshape: -> [batch_size, num_patches + 1, total_embed_dim]
x = (attn @ v).transpose(1, 2).reshape(B, N, C)
x = self.proj(x)  #经过一层卷积
x = self.proj_drop(x)  #Dropout
return x
```

MLP 层代码

MLP 是一个两层感知机，隐藏层通道数升维为原来 4 倍。

```
class Mlp(nn.Module):
    """
    MLP as used in Vision Transformer, MLP-Mixer and related networks
    """
    def __init__(self, in_features, hidden_features=None, out_features=None,
                 act_layer=nn.GELU,  # GELU是更加平滑的relu
                 drop=0.):
        super().__init__()
        out_features = out_features or in_features  #如果out_features不存在，则为in_f
        hidden_features = hidden_features or in_features #如果hidden_features不存在，
        self.fc1 = nn.Linear(in_features, hidden_features) #fc层1
        self.act = act_layer() #激活
        self.fc2 = nn.Linear(hidden_features, out_features)  #fc层2
        self.drop = nn.Dropout(drop)

    def forward(self, x):
        x = self.fc1(x)
        x = self.act(x)
        x = self.drop(x)
        x = self.fc2(x)
        x = self.drop(x)
        return x
```

Transformer 编码层代码

```
class Block(nn.Module):
    """
    基本的Transformer模块
    """
    def __init__(self,
                 dim,num_heads, mlp_ratio=4.,
                 qkv_bias=False, qk_scale=None, drop_ratio=0.,
                 attn_drop_ratio=0., drop_path_ratio=0.,
                 act_layer=nn.GELU, norm_layer=nn.LayerNorm):
        super(Block, self).__init__()
        self.norm1 = norm_layer(dim)  #norm层
        self.attn = Attention(dim, num_heads=num_heads, qkv_bias=qkv_bias, qk_scale=qk
                              attn_drop_ratio=attn_drop_ratio, proj_drop_ratio=drop_ra
        # NOTE: drop path for stochastic depth, we shall see if this is better than dr
        # 代码使用了DropPath, 而不是原版的dropout
        self.drop_path = DropPath(drop_path_ratio) if drop_path_ratio > 0. else nn.Ide
        self.norm2 = norm_layer(dim) #norm层
        mlp_hidden_dim = int(dim * mlp_ratio)  #隐藏层维度扩张后的通道数
        # 多层感知机
        self.mlp = Mlp(in_features=dim, hidden_features=mlp_hidden_dim, act_layer=act_
```

Transformer 编码层代码

```
def forward(self, x):
    x = x + self.drop_path(self.attn(self.norm1(x)))  # attention后残差连接
    x = x + self.drop_path(selfpython'.mlp(self.norm2(x)))   # mlp后残差连接
    return x
```

分类层代码

分类头很简单，就是取特征层如 197×768 的第一个向量，即 1×768，再对此进行线性全连接层进行多分类即可。

```
# self.num_features=768
# num_classes为分类任务的类别数量
self.head = nn.Linear(self.num_features, num_classes) if num_classes > 0 else nn.Ident
```

分类准确性

name	Epochs	ImageNet	ImageNet ReaL	CIFAR-10	CIFAR-100	Pets	Flowers	exaFLOPs
ViT-B/32	7	80.73	86.27	98.61	90.49	93.40	99.27	55
ViT-B/16	7	84.15	88.85	99.00	91.87	95.80	99.56	224
ViT-L/32	7	84.37	88.28	99.19	92.52	95.83	99.45	196
ViT-L/16	7	86.30	89.43	99.38	93.46	96.81	99.66	783
ViT-L/16	14	87.12	89.99	99.38	94.04	97.11	99.56	1567
ViT-H/14	14	88.08	90.36	99.50	94.71	97.11	99.71	4262
ResNet50x1	7	77.54	84.56	97.67	86.07	91.11	94.26	50
ResNet50x2	7	82.12	87.94	98.29	89.20	93.43	97.02	199
ResNet101x1	7	80.67	87.07	98.48	89.17	94.08	95.95	96
ResNet152x1	7	81.88	87.96	98.82	90.22	94.17	96.94	141
ResNet152x2	7	84.97	89.69	99.06	92.05	95.37	98.62	563
ResNet152x2	14	85.56	89.89	99.24	91.92	95.75	98.75	1126
ResNet200x3	14	87.22	90.15	99.34	93.53	96.32	99.04	3306
R50x1+ViT-B/32	7	84.90	89.15	99.01	92.24	95.75	99.46	106
R50x1+ViT-B/16	7	85.58	89.65	99.14	92.63	96.65	99.40	274
R50x1+ViT-L/32	7	85.68	89.04	99.24	92.93	96.97	99.43	246
R50x1+ViT-L/16	7	86.60	89.72	99.18	93.64	97.03	99.40	859
R50x1+ViT-L/16	14	87.12	89.76	99.31	93.89	97.36	99.11	1668

分类优越性

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	**88.55** ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02	88.4/88.5*
ImageNet ReaL	**90.72** ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54	90.55
CIFAR-10	**99.50** ± 0.06	99.42 ± 0.03	99.15 ± 0.03	99.37 ± 0.06	–
CIFAR-100	**94.55** ± 0.04	93.90 ± 0.05	93.25 ± 0.05	93.51 ± 0.08	–
Oxford-IIIT Pets	**97.56** ± 0.03	97.32 ± 0.11	94.67 ± 0.15	96.62 ± 0.23	–
Oxford Flowers-102	99.68 ± 0.02	**99.74** ± 0.00	99.61 ± 0.02	99.63 ± 0.03	–
VTAB (19 tasks)	**77.63** ± 0.23	76.28 ± 0.46	72.72 ± 0.21	76.29 ± 1.70	–
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k

1. ViT 图像分类模型
2. Segmentation Transformer 模型
   - 模型原理
   - 代码讲解

1 ViT 图像分类模型

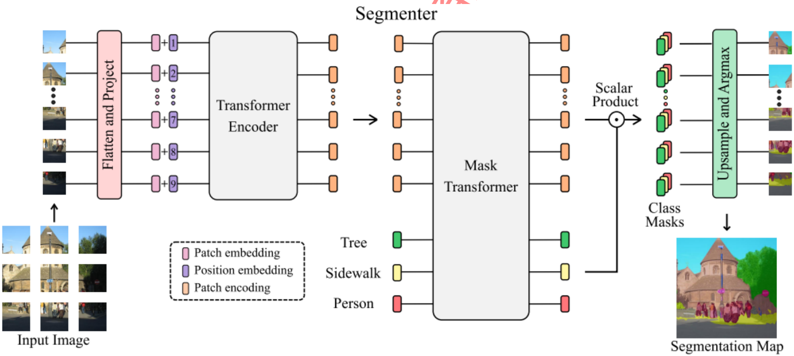
2 Segmentation Transformer 模型

模型原理

代码讲解

Segmentation Transformer 模型

- *Segmenter: Transformer for Semantic Segmentation*
- `https://arxiv.org/abs/2105.05633`



Decoder

最后一个 Transformer 编码层的输出上采样至原尺寸

输入的 $\mathbf{z_L} \in \mathbb{R}^{N\times D}$ 首先经过 point-wise linear layer 变换到 $\mathbf{z_{lin}} \in \mathbb{R}^{N\times K}$；

$\mathbf{z_{lin}} \in \mathbb{R}^{N\times K}$ reshape 到 $\mathbf{s_{lin}} \in \mathbb{R}^{H/P\times W/P\times K}$；

$\mathbf{s_{lin}} \in \mathbb{R}^{H/P\times W/P\times K}$ 再经过双线性上采样到原图像尺寸，得到最后的分割图像 $\mathbf{s} \in \mathbb{R}^{H\times W\times K}$。

1 ViT 图像分类模型

2 Segmentation Transformer 模型

- 模型原理
- 代码讲解

Decoder 代码

```
class DecoderLinear(nn.Module):
    def __init__(self, n_cls, patch_size, d_encoder):
        super().__init__()

        self.d_encoder = d_encoder
        self.patch_size = patch_size
        self.n_cls = n_cls

        self.head = nn.Linear(self.d_encoder, n_cls)
        self.apply(init_weights)

    @torch.jit.ignore
    def no_weight_decay(self):
        return set()

    def forward(self, x, im_size):
        H, W = im_size
        GS = H // self.patch_size
        x = self.head(x)
        x = rearrange(x, "b (h w) c -> b c h w", h=GS)

        return x
```

Decoder 代码


```python
class MaskTransformer(nn.Module):
    def __init__(
        self,
        n_cls,
        patch_size,
        d_encoder,
        n_layers,
        n_heads,
        d_model,
        d_ff,
        drop_path_rate,
        dropout,
    ):
        super().__init__()
        self.d_encoder = d_encoder
        self.patch_size = patch_size
        self.n_layers = n_layers
        self.n_cls = n_cls
        self.d_model = d_model
        self.d_ff = d_ff
        self.scale = d_model ** -0.5
```

```python
dpr = [x.item() for x in torch.linspace(0, drop_path_rate, n_layers)]
self.blocks = nn.ModuleList(
    [Block(d_model, n_heads, d_ff, dropout, dpr[i]) for i in range(n_layers)]
)

self.cls_emb = nn.Parameter(torch.randn(1, n_cls, d_model))
self.proj_dec = nn.Linear(d_encoder, d_model)

self.proj_patch = nn.Parameter(self.scale * torch.randn(d_model, d_model))
self.proj_classes = nn.Parameter(self.scale * torch.randn(d_model, d_model))

self.decoder_norm = nn.LayerNorm(d_model)
self.mask_norm = nn.LayerNorm(n_cls)

self.apply(init_weights)
trunc_normal_(self.cls_emb, std=0.02)
```

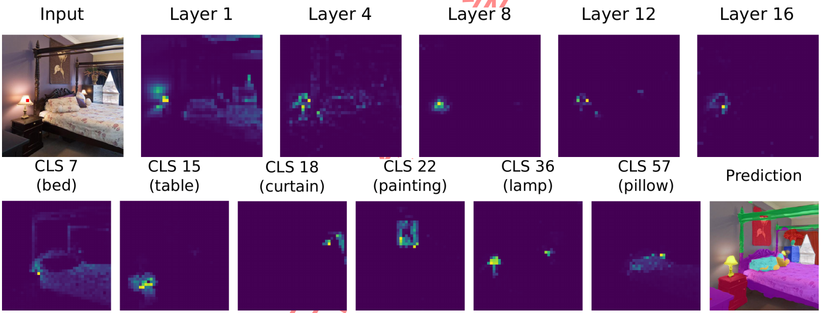
Decoder 代码

```python
def no_weight_decay(self):
    return {"cls_emb"}

def forward(self, x, im_size):
    H, W = im_size
    GS = H // self.patch_size

    x = self.proj_dec(x)
    cls_emb = self.cls_emb.expand(x.size(0), -1, -1)
    x = torch.cat((x, cls_emb), 1)
    for blk in self.blocks:
        x = blk(x)
    x = self.decoder_norm(x)

    patches, cls_seg_feat = x[:, : -self.n_cls], x[:, -self.n_cls :]
    patches = patches @ self.proj_patch
    cls_seg_feat = cls_seg_feat @ self.proj_classes

    patches = patches / patches.norm(dim=-1, keepdim=True)
    cls_seg_feat = cls_seg_feat / cls_seg_feat.norm(dim=-1, keepdim=True)
```

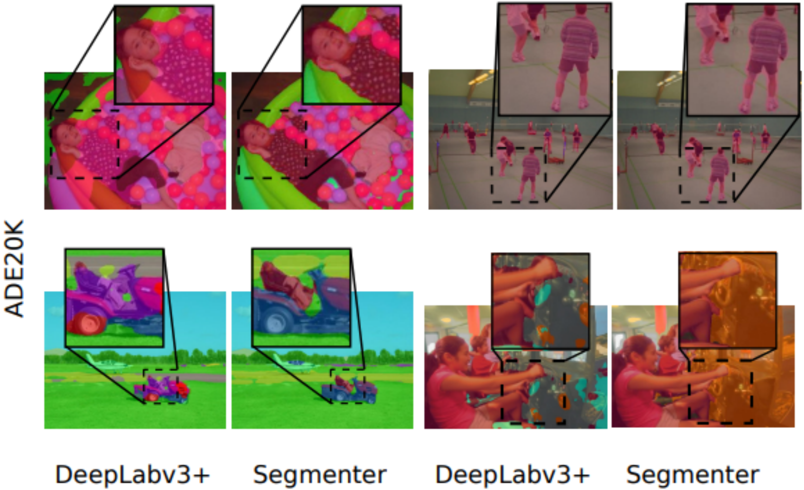
attention map 代码

```python
def get_attention_map(self, x, layer_id):
    if layer_id >= self.n_layers or layer_id < 0:
        raise ValueError(
            f"Provided layer_id: {layer_id} is not valid. 0 <= {layer_id} < {self.n_layers}."
        )
    x = self.proj_dec(x)
    cls_emb = self.cls_emb.expand(x.size(0), -1, -1)
    x = torch.cat((x, cls_emb), 1)
    for i, blk in enumerate(self.blocks):
        if i < layer_id:
            x = blk(x)
        else:
            return blk(x, return_attention=True)
```

attention map 结果

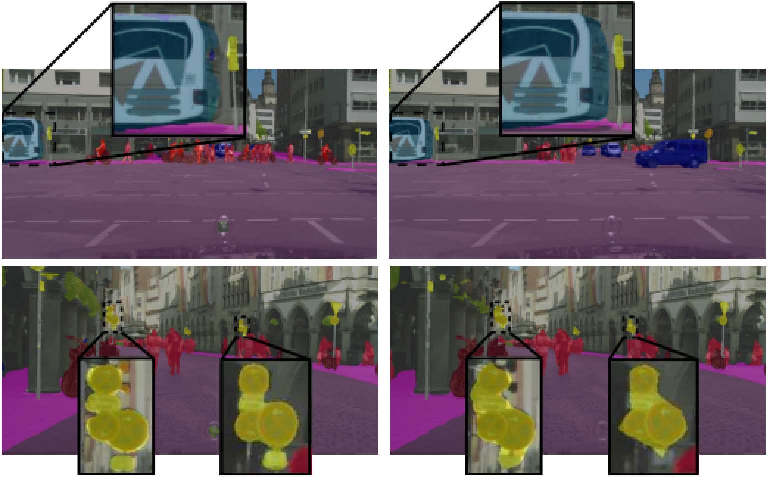


分割结果



分割结果

CITYSCAPES



结果对比：ADE20K

The ADE20K semantic segmentation dataset contains more than 20K scene-centric images exhaustively annotated with pixel-level objects and object parts labels. There are totally 150 semantic categories, which include stuffs like sky, road, grass, and discrete objects like person, car, bed.

Training set

25.574 images

All images are fully annotated with objects and, many of the images have parts too.

Validation set

2.000 images

Fully annotated with objects and parts

Test set

Images to be released later.

Consistency set

64 images and annotations used for checking the annotation consistency (download)

Method	Backbone	Im/sec	mIoU	+MS
OCR [60]	HRNetV2-W48	83	-	45.66
ACNet [24]	ResNet-101	-	-	45.90
DNL [57]	ResNet-101	-	-	45.97
DRANet [22]	ResNet-101	-	-	46.18
CPNet [58]	ResNet-101	-	-	46.27
DeepLabv3+ [10]	ResNet-101	76	45.47	46.35
DeepLabv3+ [10]	ResNeSt-101	15	46.47	47.27
DeepLabv3+ [10]	ResNeSt-200	-	-	48.36
SETR-L MLA [67]	ViT-L/16	34	48.64	50.28
Swin-L UperNet [35]	Swin-L/16	34	52.10	**53.50**
Seg-B†/16	DeiT-B/16	77	47.08	48.05
Seg-B†-Mask/16	DeiT-B/16	76	48.70	50.08
Seg-L/16	ViT-L/16	33	50.71	52.25
Seg-L-Mask/16	ViT-L/16	31	51.82	**53.63**

结果对比：Pascal 和 Cityscapes 数据集

Method	Backbone	mIoU (MS)
DeepLabv3+ [10]	ResNet-101	48.5
DANet [23]	ResNet-101	52.6
ANN [69]	ResNet101	52.8
CPNet [58]	ResNet-101	53.9
CFNet [64]	ResNet-101	54.0
ACNet [24]	ResNet-101	54.1
APCNet [26]	ResNet101	54.7
DNL [57]	HRNetV2-W48	55.3
DRANet [22]	ResNet-101	55.4
OCR [60]	HRNetV2-W48	56.2
SETR-L MLA [67]	ViT-L/16	55.8
Seg-B†/16	DeiT-B/16	53.9
Seg-B†-Mask/16	DeiT-B/16	55.0
Seg-L/16	ViT-L/16	56.5
Seg-L-Mask/16	ViT-L/16	**59.0**

Method	Backbone	mIoU (MS)
PSANet [66]	ResNet-101	79.1
DeepLabv3+ [10]	Xception-71	79.6
ANN [69]	ResNet-101	79.9
MDEQ [5]	MDEQ	80.3
DeepLabv3+ [10]	ResNeSt-101	80.4
DNL [57]	ResNet-101	80.5
CCNet [31]	ResNet-101	81.3
Panoptic-Deeplab [12]	Xception-71	81.5
DeepLabv3+ [10]	ResNeSt-200	**82.7**
SETR-L PUP [67]	ViT-L/16	**82.2**
Seg-B†/16	DeiT-B/16	80.5
Seg-B†-Mask/16	DeiT-B/16	80.6
Seg-L/16	ViT-L/16	80.7
Seg-L-Mask/16	ViT-L/16	81.3