WSDAN论文+代码复现

论文部分

摘要

data augmentation用于增加数据量，防止过拟合，但是实际上随机对图像裁剪等方法效率低，引入噪声，而WSDAN通过首先弱监督学习生成attention map表示辨别的部分，再在其基础上进行增强，包括attention裁剪和attention丢弃

两个部分

1.提取到了更多能够帮助分辨的部分，可以更好观察图像

2.attention region提供了物体准确位置，能更加细化查看物体产生更好的效果

attention map对辨别部分表示？为什么提取到？如何进行后续的增强？

导言

data augmentation的重要性----在计算机视觉中有效

深度模型有多种，如裁剪旋转转换灰度图等

但是由于随机抽样，很大一部分都包含了噪声，导致训练效率降低

因此模型需要了解对象的空间信息

细粒度视觉分类（FGVC）对于更细粒度进行分类如猫头鹰和喜鹊等等

三个挑战在于

1.类内部差异大，同一类别的物体在姿态和视角有差异

2.类之间差异小，例如鸟类之间可能要通过头部颜色来确定类别

3.训练数据有限。进行标注需要专业知识和大量标注时间，并且很难找出有区分度的特点

因此我们使用弱监督，只给图像级别的标注不给出边界框，**通过卷积生成attention map**来表示出物体的各个部分或视觉模式，而且还提出了**双线性注意力池化层**和**注意力正则化损失**来进行弱监督，这个模型更加容易定位大量的物体特征（10个以上）来达到更好的效果，获取到位置以后提出了**注意力引导的增强方法**，解决类内差异大，类间差异小的问题。

注意力引导的两个优点

1.对于不同的细粒度类别，物体通常非常相似，只有极少数的差异，注意力裁剪可以通过裁剪和调整部分区域大小来进行区分，从而提取出更鲜明的区分点

2.精确定位对象使得更加近距离地观察对象并且完善预测，首先是从粗粒度预测再放大到细粒度

主要贡献

1.weakly supervised attention learning产生attention map来辨别各部分的分布，提取连续的局部特征解决细粒度分类

2.在attention map基础上提出注意力引导的数据增强，以提高数据增强的效率，包括裁剪和丢弃，裁剪会随机裁剪和调整注意力部分的大小，丢弃随机抹去图像一个注意力区域，鼓励模型从多个辨别部分提取特征

3.用attention map准确定位整个对象，并且进行放大，进一步提高分类准确性

相关部分

细粒度视觉分类

CNN无法关注到细粒度水平

产生了part RCNN和extended RCNN

在geometric prior下定位部分，预测细粒度类别通过pose-normalized representation（姿态归一化）预测细粒度的类别，以及DEEP LAC 反馈控制框架将对齐和分类错误反向传播到定位中，并且提出了VLF连接定位和分类模块

数据增强

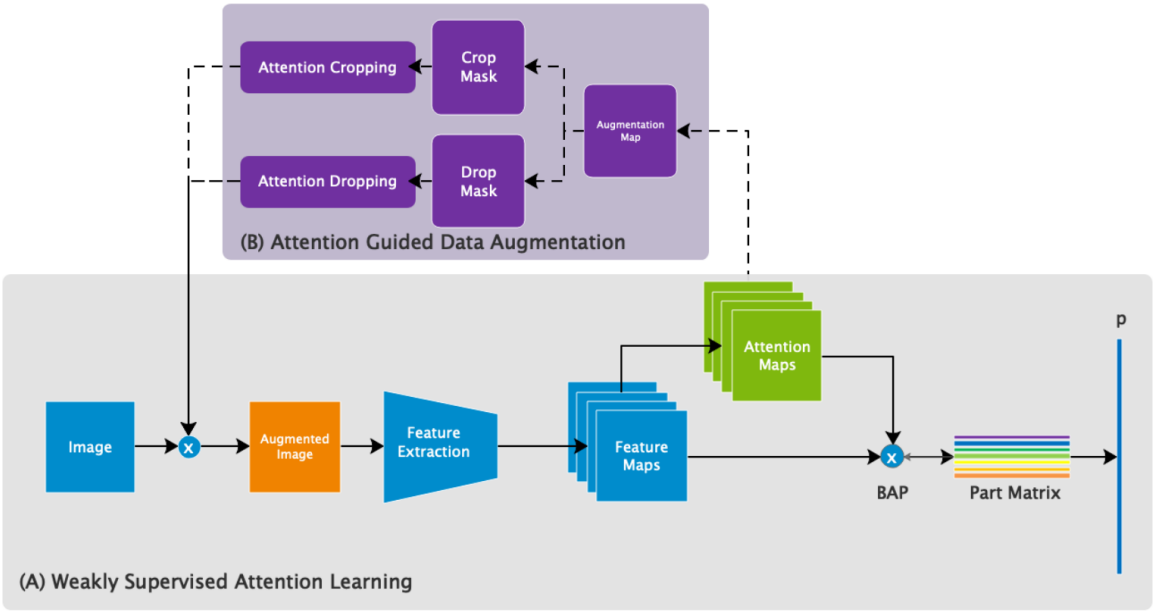
随机空间图像增强方法，如图像裁剪和丢弃

其中一种的缺点在于只能移除每幅图像中的一个区分区域限制了性能

另一种通过随机屏蔽训练图像中的许多方形区域，但是许多被抹去的是无关的背景或者整个对象都被抹去了，尤其对于小的物体

又产生了其他一些方法

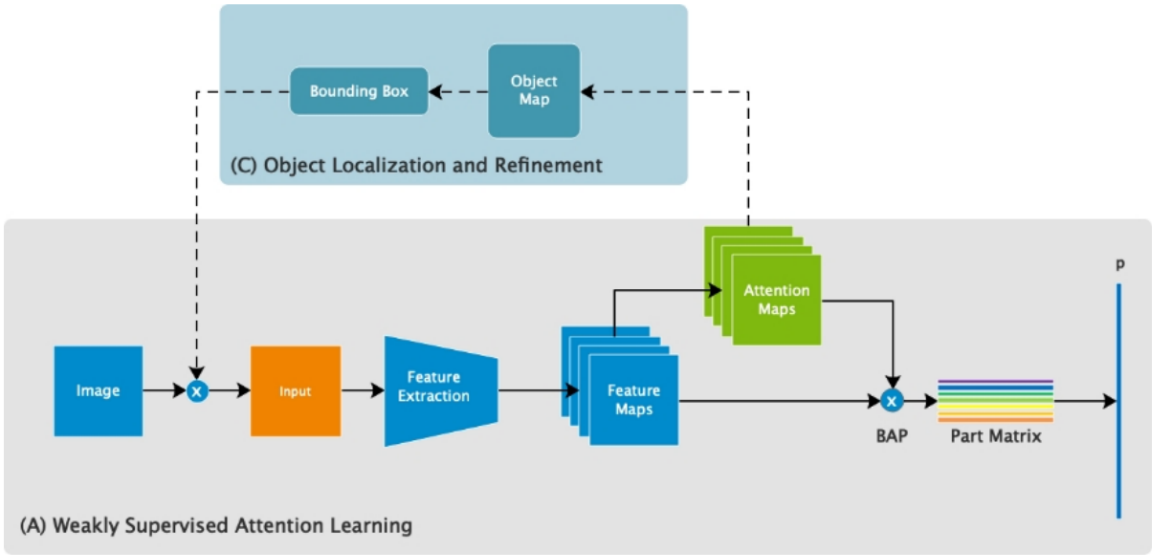
MSDAN方法



首先训练部分

1.弱监督attention学习获得每个图片的attention maps来表示特征部分

2.然后注意力引导随机选择一张图进行增强，包括裁剪和丢弃，将原始数据和增强数据一起输入进行训练



测试部分

首先从原始图像输出对象的类别概率和attention map

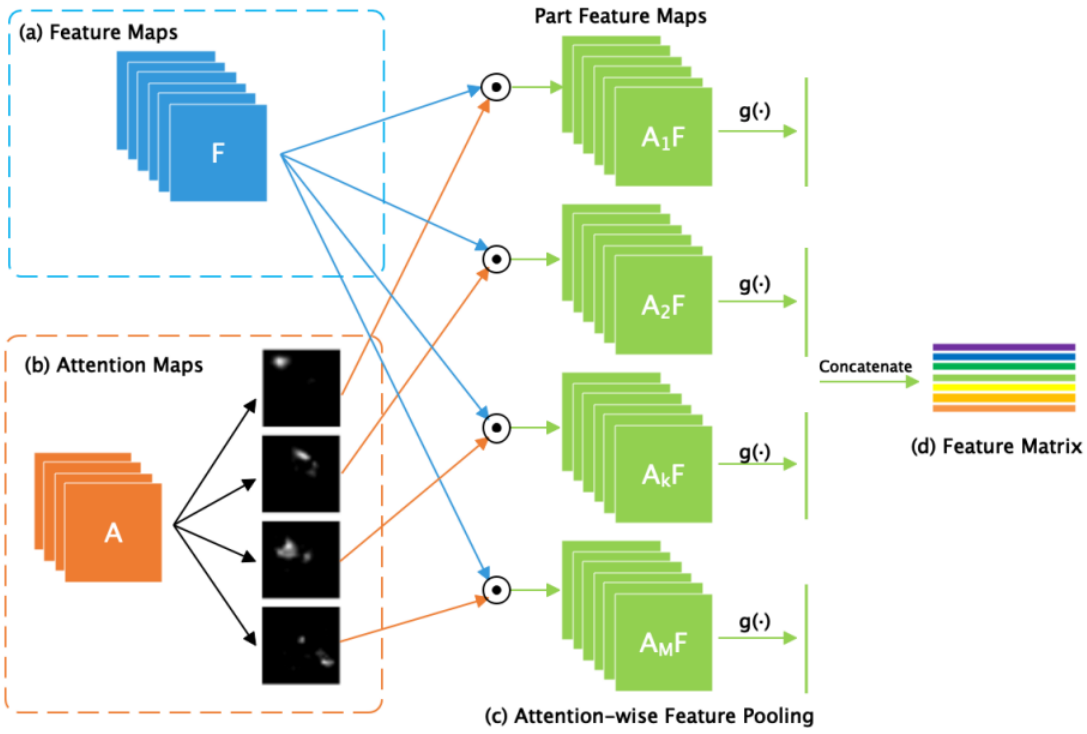
根据这再进行放大细化分类的概率，最终将两种概率合并为最终预测结果

spatial representation

$$A = f(F) = \bigcup_{k=1}^{M} A_k$$

Ak代表一个特征，如鸟的头，一共M个attention map

We element-wise multiply feature maps F by each attention map Ak in order to generate M part feature maps Fk



Bilinear Attention Pooling

提取完到attention map之后再用feature map和attention map进行相乘，获得M组Fk图像

$$F_k = A_k \odot F \quad (k = 1, 2, ..., M) \tag{2}$$

where $\odot$ denotes element-wise multiplication for two tensors.

然后通过

Global Average Pooling (GAP), Global Maximum Pooling (GMP) or convolutions

再提取更加细分的特征

$$f_k = g(F_k)$$

Attention Regularization

将得到的特征fk与特征中心ck进行正则化

$$L_A = \sum_{k=1}^{M} \| f_k - c_k \|_2^2$$

$$c_k \leftarrow c_k + \beta(f_k - c_k)$$

β只在原始图像应用，控制ck更新速率以及attention正则化

Attention-guided data augmentation

attention map

对于每一张训练图像，随机选择他的其中一张attention map Ak来进行数据增强，并且进行了标准化

guide the data augmentation process, and normalize it as k_{th} Augmentation Map $A_k^* \in R^{H \times W}$.

$$A_k^* = \frac{A_k - \min(A_k)}{\max(A_k) - \min(A_k)} \tag{7}$$

attention cropping

获得处理后的attention map

对其进行掩码处理，大于阈值的设置为1，，否则设置为0

然后找一个bounding box圈住ck的正区域进行放大，作为增强输入数据

attention dropping

保留了一个Dk

和Ck正好相反

$$D_k(i,j) = \begin{cases} 0, & \text{if } A_k^*(i,j) > \theta_d \\ 1, & \text{otherwise.} \end{cases}$$

因为将这个部分舍弃或覆盖掉后网络就寻找剩余的特征，分类的鲁棒性和位置的准确性将会提升

Object Localization and Refinement

获取到细化特征再结合原本网络能够产生的粗粒度对象进行一同处理，通过细粒度特征求和平均获得整体

$$A_m = \frac{1}{M}\sum_{k=1}^{M} A_k$$

整体的算法

Algorithm 1 Object Localization and Refinement

Require: Trained WS-DAN model W

Require: Raw image I_r

1: Predict coarse-grained probability p_1: $p_1 = W(I_r)$ and output attention maps A;
2: Calculate object map A_m of A by Equ. 10;
3: Obtain the bounding box B of object from A_m;
4: Enlarge the region in B as I_o;
5: Predict fine-grained probability p_2: $p_2 = W(I_o)$;
6: Calculate the final prediction: $p = (p_1 + p_2)/2$;
7: **return** p

首先预测出概率，输出attention map

计算Am

保留Am的细粒度的bounding box

放大bounding box作为Io

通过bounding box预测细粒度的概率

计算最终的预测（取平均值）

代码部分

双线性化池化

```
class BAP(nn.Module):
    def __init__(self,  **kwargs):
        super(BAP, self).__init__()
    def forward(self,feature_maps,attention_maps):
        feature_shape = feature_maps.size() ## 12*768*26*26*
        attention_shape = attention_maps.size() ## 12*num_parts*26*26
```

```
        # print(feature_shape,attention_shape)
        phi_I = torch.einsum('imjk,injk->imn', (attention_maps, feature_maps)) ##
12*32*768
        phi_I = torch.div(phi_I, float(attention_shape[2] *
attention_shape[3]))## 进行标准化除以了height*width
        phi_I = torch.mul(torch.sign(phi_I), torch.sqrt(torch.abs(phi_I) + 1e-
12))
        phi_I = phi_I.view(feature_shape[0],-1)
        raw_features = torch.nn.functional.normalize(phi_I, dim=-1) ##12*(32*768)
        pooling_features = raw_features*100
        # print(pooling_features.shape)
        return raw_features,pooling_features
```

首先得到了feature map和attention map

通过einsum得到了对特征图和注意力图在空间维度上进行逐元素乘法并求和

也就是part feature map

将 `phi_I` 除以注意力图的空间维度大小 `(height * width)` 进行标准化。

使用 `torch.sign` 和 `torch.sqrt` 对 `phi_I` 进行元素级的符号和平方根操作，这是为了实现双线性池化，避免负值带来的问题。

加上一个很小的值 `1e-12` 来防止平方根计算时出现数值不稳定。

使用 `view` 将 `phi_I` 展平为 `(batch_size, num_parts * channels)`。

使用 `torch.nn.functional.normalize` 对特征进行 L2 归一化。

将归一化后的特征乘以 100 放大。

返回 `raw_features` 和 `pooling_features`。

attention正则化

```
def calculate_pooling_center_loss(features, centers, label, alfa=0.95):

    features = features.reshape(features.shape[0], -1)

    centers_batch = centers[label]

    centers_batch = torch.nn.functional.normalize(centers_batch, dim=-1)
    diff =  (1-alfa)*(features.detach() - centers_batch)
    distance = torch.pow(features - centers_batch,2)
    distance = torch.sum(distance, dim=-1)
    center_loss = torch.mean(distance)

    return center_loss, diff
```

center进行正则化

feature和center相减的平方求和得到La

最终返回Ck La

attention数据增强

找出Cmask和Dmask

```
def attention_crop_drop(attention_maps,input_image):
    # start = time.time()
    B,N,W,H = input_image.shape
    input_tensor = input_image
    batch_size, num_parts, height, width = attention_maps.shape
    attention_maps =
torch.nn.functional.interpolate(attention_maps.detach(),size=
(W,H),mode='bilinear')
    part_weights = F.avg_pool2d(attention_maps.detach(),
(W,H)).reshape(batch_size,-1)
    part_weights = torch.add(torch.sqrt(part_weights),1e-12)
    part_weights =
torch.div(part_weights,torch.sum(part_weights,dim=1).unsqueeze(1)).cpu()
    part_weights = part_weights.numpy()
    # print(part_weights.shape)
    ret_imgs = []
    masks = []
    # print(part_weights[3])
    for i in range(batch_size):
        attention_map = attention_maps[i]
        part_weight = part_weights[i]
        selected_index = np.random.choice(np.arange(0, num_parts), 1,
p=part_weight)[0]
        selected_index2 = np.random.choice(np.arange(0, num_parts), 1,
p=part_weight)[0]
        ## create crop imgs
        mask = attention_map[selected_index, :, :]
        # mask = (mask-mask.min())/(mask.max()-mask.min())
        threshold = random.uniform(0.4, 0.6)
        # threshold = 0.5
        itemindex = np.where(mask >= mask.max()*threshold)
        # print(itemindex.shape)
        # itemindex = torch.nonzero(mask >= threshold*mask.max())
        padding_h = int(0.1*H)
        padding_w = int(0.1*W)
        height_min = itemindex[0].min()
        height_min = max(0,height_min-padding_h)
        height_max = itemindex[0].max() + padding_h
        width_min = itemindex[1].min()
        width_min = max(0,width_min-padding_w)
        width_max = itemindex[1].max() + padding_w
        # print('numpy',height_min,height_max,width_min,width_max)
        out_img = input_tensor[i]
[:,height_min:height_max,width_min:width_max].unsqueeze(0)
        out_img = torch.nn.functional.interpolate(out_img,size=
(W,H),mode='bilinear',align_corners=True)
        out_img = out_img.squeeze(0)
        ret_imgs.append(out_img)

        ## create drop imgs
        mask2 = attention_map[selected_index2:selected_index2 + 1, :, :]
```

```
            threshold = random.uniform(0.2, 0.5)
            mask2 = (mask2 < threshold * mask2.max()).float()
            masks.append(mask2)
        # bboxes = np.asarray(bboxes, np.float32)
        crop_imgs = torch.stack(ret_imgs)
        masks = torch.stack(masks)
        drop_imgs = input_tensor*masks
        return (crop_imgs,drop_imgs)
```

通过一个函数进行了attention map的处理

```
# mask = (mask-mask.min())/(mask.max()-#mask.min())
```

返回crop和drop image

测试阶段测试集的目标定位与图像精修

```
def mask2bbox(attention_maps,input_image):
    input_tensor = input_image
    B,C,H,W = input_tensor.shape
    batch_size, num_parts, Hh, Ww = attention_maps.shape
    attention_maps = torch.nn.functional.interpolate(attention_maps,size=
(W,H),mode='bilinear',align_corners=True)
    ret_imgs = []
    for i in range(batch_size):
        attention_map = attention_maps[i]
        mask = attention_map.mean(dim=0)
        threshold = 0.1
        max_activate = mask.max()
        min_activate = threshold * max_activate
        itemindex = torch.nonzero(mask >= min_activate)
        print(itemindex.shape)
        padding_h = int(0.05*H)
        padding_w = int(0.05*W)
        # 找到非零元素出现时最小的行数,并且将抠出的图放大一定范围
        height_min = itemindex[:, 0].min()
        height_min = max(0,height_min-padding_h)
        height_max = itemindex[:, 0].max() + padding_h
        width_min = itemindex[:, 1].min()
        width_min = max(0,width_min-padding_w)
        width_max = itemindex[:, 1].max() + padding_w
        out_img = input_tensor[i]
[:,height_min:height_max,width_min:width_max].unsqueeze(0)
        out_img = torch.nn.functional.interpolate(out_img,size=
(W,H),mode='bilinear',align_corners=True)
        out_img = out_img.squeeze(0)
        ret_imgs.append(out_img)
    ret_imgs = torch.stack(ret_imgs)
    return ret_imgs
```

复现

```
root@I1b816133b2004010c5:/hy-tmp/wsdan/WS-DAN.PyTorch# nohup: ignoring input and redirecting stderr to stdout
tail -f progress.bar
Epoch 2/30: 100%|██| 556/556 [04:00<00:00,  2.31 batches/s, Loss 2.3050, Raw Acc (68.23, 91.71), Crop Acc (47.26, 75.97), Drop Acc (58.53
Epoch 3/30:  65%|▋ | 363/556 [02:57<01:39,  1.95 batches/s, Loss 1.5425, Raw Acc (80.56, 96.99), Crop Acc (61.39, 86.91), Drop Acc (73.32
```

可以看出

val acc即测试集准确率

```
2024-08-07 15:41:35,038: INFO: [eval.py:76]: Network loaded from ./FGVC/CUB-200-2011/ckpt/model.ckpt
Validation: 100%|[redacted]| 278/278 [01:18<00:00,  3.56 batches/s, Val Acc: Raw (92.77, 97.78), Refine (92.80, 97.87)]
```

```
batch_info = 'Val Acc: Raw ({:.2f}, {:.2f}), Refine ({:.2f}, {:.2f})'.format(
                epoch_raw_acc[0], epoch_raw_acc[1], epoch_ref_acc[0],
epoch_ref_acc[1])
```

在数据增强前后有比较明显的提升77->80 78->87

并且原本准确率也比较高了92%（按照官方说法训练160epoch会94.5%）

通过可视化图像也可以看出

这是初始照片

这是attention处理以后

这是热力图

通过attention map机制定位到机翼、机尾、机身涂鸦能够更加帮助识别飞机品牌与型号，屏蔽掉无关信息

PS：模型真的应该往轻量化发展，4090拿来跑aircraft数据集只跑30个epoch都跑了1个小时+，还好最后跑出来结果了

附录

部分函数用法

```
torch.einsum
#爱因斯坦求和约定计算
As = torch.randn(3, 2, 5)
>>> Bs = torch.randn(3, 5, 4)
>>> torch.einsum('bij,bjk->bik', As, Bs)
tensor([[[-1.0564, -1.5904,  3.2023,  3.1271],
         [-1.6706, -0.8097, -0.8025, -2.1183]],

         [[ 4.2239,  0.3107, -0.5756, -0.2354],
         [-1.4558, -0.3460,  1.5087, -0.8530]],

         [[ 2.8153,  1.8787, -4.3839, -1.2112],
         [ 0.3728, -2.1131,  0.0921,  0.8305]]])
```

```
phi_I.view
#展开铺平
```

```
features.reshape
```

```
torch.stack
#官方解释：沿着一个新维度对输入张量序列进行连接。 序列中所##有的张量都应该为相同形状。

#浅显说法：把多个2维的张量凑成一个3维的张量；多个3维的凑成##一个4维的张量…以此类推
```

```
nn.functional.interpolate
#该函数实现插值和上采样
#指定部分进行处理
#例如
    a1 = torch.Tensor(4,3,14,14)
    a3 = torch.Tensor(4,5,12,12)
    resized_at3 = nn.functional.interpolate(at3,(H,W))
    #则at3变成（4，5，14，14）
    cat的时候成为
    （4，3+5，14，14）
```

```
torch.mean
#mean()函数的参数：dim=0,按列求平均值，返回的形状是（1，列数）；dim=1,按行求平均值，返回的形状是（行数，1）,默认不设置dim的时#候，返回的是所有元素的平均值。
```