KeypointDetection笔记——UDP

- Paper: The Devil is in the Details: Delving into Unbiased Data Processing for Human Pose Estimation
- Code: HuangJunJie2017/UDP-Pose
- Blog: 魔鬼在细节中：人体姿态估计中无偏的数据处理方法(黄骏杰 知乎)
- Blog: 2020 COCO Keypoint Challenge 冠军之路(黄骏杰 知乎)

0. Idea

- 在hrnet源码中，把由水平反转的图得到的热图shift了一个像素，这个操作没有任何文档记载描述，issue里也有相同的疑问，作者并没有回答，关键是这个操作对指标影响非常大。在打19年coco时察觉到数据流有偏的问题，经过深入研究、推理还有对其他repo的分析，才发现这个shift一个像素的操作就是为了弥补数据流有偏所以起效的。

```
# lib/dataset/JointDataset.py中的def __getitem__(self, idx):函数中
            # 进行数据水平翻转
            if self.flip and random.random() <= 0.5:
                data_numpy = data_numpy[:, ::-1, :]
                joints, joints_vis = fliplr_joints(
                    joints, joints_vis, data_numpy.shape[1], self.flip_pairs)
```

```
# 在lib/utils/transforms.py中
def fliplr_joints(joints, joints_vis, width, matched_parts):
    """
    flip coords
    """
    # Flip horizontal
    joints[:, 0] = width - joints[:, 0] - 1

    # Change left-right parts
    for pair in matched_parts:
        joints[pair[0], :], joints[pair[1], :] = \
            joints[pair[1], :], joints[pair[0], :].copy()
        joints_vis[pair[0], :], joints_vis[pair[1], :] = \
            joints_vis[pair[1], :], joints_vis[pair[0], :].copy()

    return joints*joints_vis, joints_vis
```

Standard approaches [32, 26] explicitly shift the flipped result by 1 pixel before averaging operation to narrow this gap:

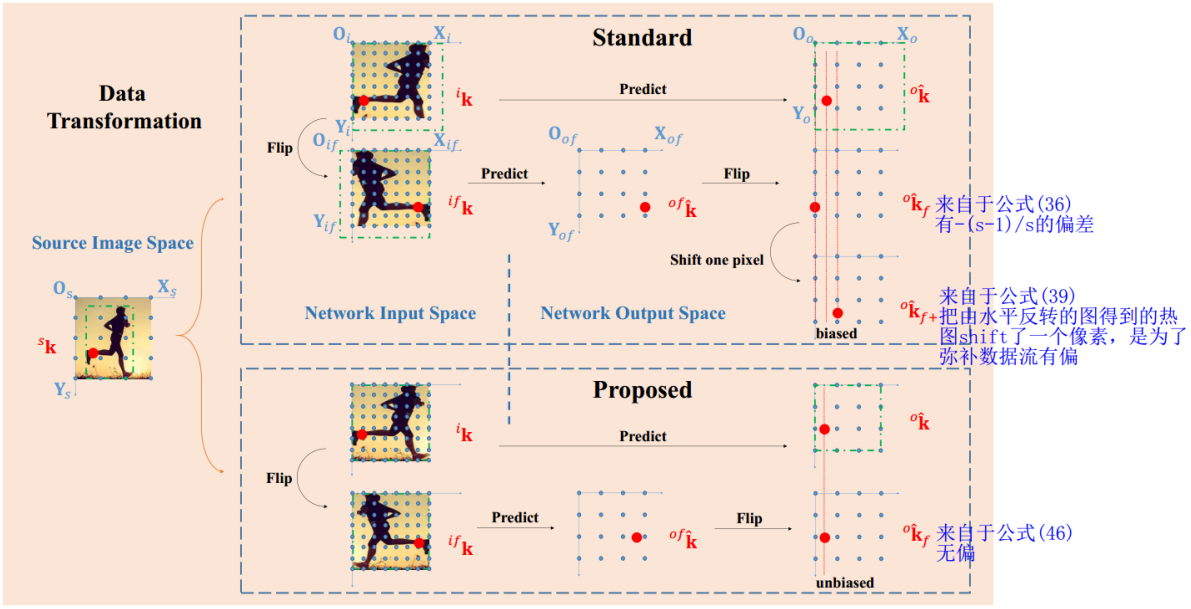
$$
\begin{aligned}
{}^{o}\hat{\mathbf{k}}_{f+} &= \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} {}^{o}\hat{\mathbf{k}}_{f} \\
&= \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -\frac{s-1}{s} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} {}^{o}\hat{\mathbf{k}} \\
&= \begin{bmatrix} 1 & 0 & \frac{1}{s} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} {}^{o}\hat{\mathbf{k}}
\end{aligned}
\tag{39}
$$

In this way, the final error can be reduced to ${}^{o}e(x)' = |\frac{1}{2s}|$. ${}^{o}e(x)' < {}^{o}e(x)$ when $s > 2$, which makes sense in most

- 解决有偏的思路关键是作者觉得离散空间分析问题时有量化误差，所以想在连续空间中进行分析避免有偏。因此使用单位长度去度量图像大小，而非像素多少

1. Introduction

- this is the first work to systematically address the data processing in pose community;
- this paper formulates a principled Unbiased Data Processing (UDP) strategy, which equips with unit length-based measurement and offset-based encoding-decoding;
- UDP promotes state-of-the-arts by large margin among variable backbones and input sizes, It is worth noting that our approach only increases negligible calculation burden during training and inference.



Then, the final result of flipped image ${}^{o}\hat{\mathbf{k}}_f$ can be obtained by flipping back:

$$
\begin{aligned}
{}^{o}\hat{\mathbf{k}}_f &= \begin{bmatrix} -1 & 0 & {}^{p}w_o - 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} {}^{of}\hat{\mathbf{k}} \\
&= \begin{bmatrix} -1 & 0 & {}^{p}w_o - 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \frac{1}{s} {}^{if}\mathbf{k} \\
&= \begin{bmatrix} -1 & 0 & {}^{p}w_o - 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{{}^{p}w_o}{{}^{p}w_i} & 0 & 0 \\ 0 & \frac{{}^{p}w_o}{{}^{p}w_i} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & {}^{p}w_i - 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} {}^{i}\mathbf{k} \\
&= \begin{bmatrix} 1 & 0 & -\frac{{}^{p}w_i - {}^{p}w_o}{{}^{p}w_i} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{{}^{p}w_o}{{}^{p}w_i} & 0 & 0 \\ 0 & \frac{{}^{p}w_o}{{}^{p}w_i} & 0 \\ 0 & 0 & 1 \end{bmatrix} {}^{i}\mathbf{k} \\
&= \begin{bmatrix} 1 & 0 & -\frac{s-1}{s} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} {}^{o}\hat{\mathbf{k}}
\end{aligned}
\tag{36}
$$

In this way, the result of flipped image ${}^{o}\hat{\mathbf{k}}_f$ is exactly aligned with the original result ${}^{o}\hat{\mathbf{k}}$:

$$
\begin{aligned}
{}^{o}\hat{\mathbf{k}}_f &= \begin{bmatrix} -1 & 0 & {}^{p}w_o - 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} {}^{of}\hat{\mathbf{k}} \\
&= \begin{bmatrix} -1 & 0 & {}^{p}w_o - 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \frac{1}{t} {}^{if}\mathbf{k} \\
&= \begin{bmatrix} -1 & 0 & {}^{p}w_o - 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{{}^{p}w_o - 1}{{}^{p}w_i - 1} & 0 & 0 \\ 0 & \frac{{}^{p}w_o - 1}{{}^{p}w_i - 1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & {}^{p}w_i - 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} {}^{i}\mathbf{k} \\
&= \begin{bmatrix} \frac{{}^{p}w_o - 1}{{}^{p}w_i - 1} & 0 & 0 \\ 0 & \frac{{}^{p}w_o - 1}{{}^{p}w_i - 1} & 0 \\ 0 & 0 & 1 \end{bmatrix} {}^{i}\mathbf{k} \\
&= {}^{o}\hat{\mathbf{k}}
\end{aligned}
\tag{46}
$$

1.1 What

- unit length-based measurement
- offset-based encoding-decoding

1.2 Why

- 在测试过程中，如果使用flip ensemble时，由翻转图像得到的结果和原图得到的结果并不对齐
- 使用的编码解码（encoding-decoding）方法存在较大的统计误差

1.3 How

- 在数据处理的时候，使用单位长度去度量图像的大小，而非像素的多少，以解决第一个问题
- 引入一种在理想情况下无统计误差的编码解码方法，以解决第二个问题

2. 公式&程序

2.1 数据转换

2.1.1 原图坐标系统->神经网络输入坐标系统

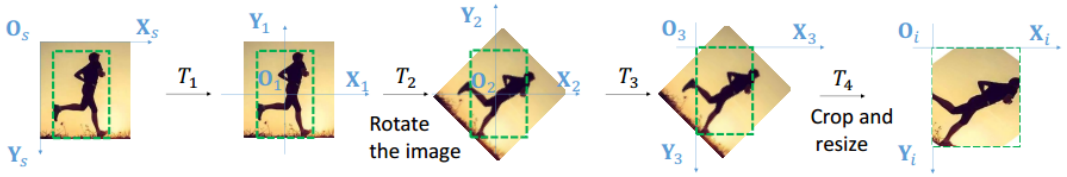


Figure 4. Illustration for the processes of standard biased data transformation and proposed unbiased data transformation. The network input size is assumed to be $({}^{p}w_i, {}^{p}h_i) = (8, 8)$ and the stride factor s is assumed to be 2.

- simple baselines/hrnet使用的有偏数据转换方法

$$
\begin{aligned}
{}^{i}\mathbf{k} &= T_4 T_3 T_2 T_1 {}^{s}\mathbf{k} \\
&= \begin{bmatrix} \frac{{}^{p}w_i}{{}^{s}w_b} & 0 & 0 \\ 0 & \frac{{}^{p}h_i}{{}^{s}h_b} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0.5{}^{s}w_b \\ 0 & -1 & 0.5{}^{s}h_b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -{}^{s}x_b \\ 0 & -1 & {}^{s}y_b \\ 0 & 0 & 1 \end{bmatrix} {}^{s}\mathbf{k} \\
&= \begin{bmatrix} \frac{{}^{p}w_i}{{}^{s}w_b}\cos\theta & -\frac{{}^{p}w_i}{{}^{s}w_b}\sin\theta & \frac{{}^{p}w_i}{{}^{s}w_b}(-{}^{s}x_b\cos\theta + {}^{s}y_b\sin\theta + 0.5{}^{s}w_b) \\ \frac{{}^{p}h_i}{{}^{s}h_b}\sin\theta & \frac{{}^{p}h_i}{{}^{s}h_b}\cos\theta & \frac{{}^{p}h_i}{{}^{s}h_b}(-{}^{s}x_b\sin\theta - {}^{s}y_b\cos\theta + 0.5{}^{s}h_b) \\ 0 & 0 & 1 \end{bmatrix} {}^{s}\mathbf{k}
\end{aligned}
\tag{30}
$$

```
# lib/dataset/JointDataset.py中的def __getitem__(self, idx):函数中
        # 进行仿射变换，样本数据关键点发生角度旋转之后，每个像素也旋转到对应位置
        # 获得旋转矩阵
        trans = get_affine_transform(c, s, r, self.image_size)
        # 根据旋转矩阵进行仿射变换
```

```
        input = cv2.warpAffine(
            data_numpy,
            trans,
            (int(self.image_size[0]), int(self.image_size[1])),
            flags=cv2.INTER_LINEAR)
        # 对人体关键点也进行仿射变换
        for i in range(self.num_joints):
            if joints_vis[i, 0] > 0.0:
                joints[i, 0:2] = affine_transform(joints[i, 0:2], trans)
```

- UDP使用的无偏数据转换方法

$$
\begin{aligned}
{}^{i}\mathbf{k} &= T_4' T_3 T_2 T_1 {}^{s}\mathbf{k} \\
&= \begin{bmatrix} \frac{{}^{p}w_i - 1}{{}^{s}w_b} & 0 & 0 \\ 0 & \frac{{}^{p}h_i - 1}{{}^{s}h_b} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0.5{}^{s}w_b \\ 0 & -1 & 0.5{}^{s}h_b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -{}^{s}x_b \\ 0 & -1 & {}^{s}y_b \\ 0 & 0 & 1 \end{bmatrix} {}^{s}\mathbf{k} \\
&= \begin{bmatrix} \frac{{}^{p}w_i - 1}{{}^{s}w_b}\cos\theta & -\frac{{}^{p}w_i - 1}{{}^{s}w_b}\sin\theta & \frac{{}^{p}w_i - 1}{{}^{s}w_b}(-{}^{s}x_b\cos\theta + {}^{s}y_b\sin\theta + 0.5{}^{s}w_b) \\ \frac{{}^{p}h_i - 1}{{}^{s}h_b}\sin\theta & \frac{{}^{p}h_i - 1}{{}^{s}h_b}\cos\theta & \frac{{}^{p}h_i - 1}{{}^{s}h_b}(-{}^{s}x_b\sin\theta - {}^{s}y_b\cos\theta + 0.5{}^{s}h_b) \\ 0 & 0 & 1 \end{bmatrix} {}^{s}\mathbf{k}
\end{aligned}
\tag{41}
$$

```
# lib/dataset/JointDataset.py中的def __getitem__(self, idx):函数中
        # 进行仿射变换，样本数据关键点发生角度旋转之后，每个像素也旋转到对应位置
        # 获得旋转矩阵，使用单位长度去度量图像的大小，而非像素的多少，因此是self.image_size-
1.0，见公式(41)
        trans = get_warpmatrix(r,c*2.0,self.image_size-1.0,s)
        # 根据旋转矩阵进行仿射变换
        input = cv2.warpAffine(
            data_numpy,
            trans,
            (int(self.image_size[0]), int(self.image_size[1])),
            flags=cv2.WARP_INVERSE_MAP|cv2.INTER_LINEAR)
        # 对人体关键点也进行仿射变换
        joints[:, 0:2] = rotate_points(joints[:, 0:2], r, c, self.image_size, s,
False)
```

2.1.2 神经网络输入坐标系统->神经网络输出坐标系统

- simple baselines/hrnet使用的有偏数据转换方法

The training sample generated by this method is semantically aligned with the original sample (*i.e.* the pose annotations are still in the proper positions). When producing the ground truth in heatmap whose size is $({}^{p}w_o, {}^{p}h_o)$, the standard methods transform the input keypoint positions by the stride factor $s = {}^{p}w_i/{}^{p}w_o = {}^{p}h_i/{}^{p}h_o$:

$$
{}^{o}\mathbf{k} = \frac{1}{s}{}^{i}\mathbf{k} \tag{32}
$$

```
# lib/dataset/JointDataset.py中的def generate_target(self, joints, joints_vis):函
数中
                # 先计算出原图到输出热图的缩小倍数
                feat_stride = self.image_size / self.heatmap_size
                # 计算输入原图的关键点，转换到热图的位置
                mu_x = int(joints[joint_id][0] / feat_stride[0] + 0.5)
                mu_y = int(joints[joint_id][1] / feat_stride[1] + 0.5)
```

- UDP使用的无偏数据转换方法

While producing the ground truth label on heatmap, we should utilize factor $t = ({}^{p}w_i - 1)/({}^{p}w_o - 1) = ({}^{p}h_i - 1)/({}^{p}h_o - 1)$:

$$
{}^{o}\mathbf{k} = \frac{1}{t}{}^{i}\mathbf{k} \tag{43}
$$

```
# lib/dataset/JointDataset.py中的def generate_target(self, joints, joints_vis):函数中
                # 先计算出原图到输出热图的缩小倍数，注意这里使用单位长度去度量图像的大小，而非像素的多少
                feat_stride = (self.image_size-1.0) / (self.heatmap_size-1.0)
                # 计算输入原图的关键点，转换到热图的位置
                mu_x = int(joints[joint_id][0] / feat_stride[0] + 0.5)
                mu_y = int(joints[joint_id][1] / feat_stride[1] + 0.5)
```

2.1.3 神经网络输出坐标系统->原图坐标系统

- simple baselines/hrnet使用的有偏数据转换方法

In inference stage, the standard method maps the predicted result ${}^{o}\mathbf{k}$ to the source image space by:

$$
{}^{s}\hat{\mathbf{k}} = \begin{bmatrix} \frac{{}^{s}w_b}{{}^{p}w_o}\cos\theta & \frac{{}^{s}h_b}{{}^{p}h_o}\sin\theta & -0.5{}^{s}w_b\cos\theta - 0.5{}^{s}h_b\sin\theta + {}^{s}x_b \\ -\frac{{}^{s}w_b}{{}^{p}w_o}\sin\theta & \frac{{}^{s}h_b}{{}^{p}h_o}\cos\theta & 0.5{}^{s}w_b\sin\theta - 0.5{}^{s}h_b\cos\theta + {}^{s}y_b \\ 0 & 0 & 1 \end{bmatrix} {}^{o}\hat{\mathbf{k}} \tag{33}
$$

```
# 在lib/core/inference.py中
def get_final_preds(config, batch_heatmaps, center, scale):
    coords, maxvals = get_max_preds(batch_heatmaps)

    heatmap_height = batch_heatmaps.shape[2]
    heatmap_width = batch_heatmaps.shape[3]

    # post-processing
    if config.TEST.POST_PROCESS:
        for n in range(coords.shape[0]):
            for p in range(coords.shape[1]):
                hm = batch_heatmaps[n][p]
                px = int(math.floor(coords[n][p][0] + 0.5))
                py = int(math.floor(coords[n][p][1] + 0.5))
                if 1 < px < heatmap_width-1 and 1 < py < heatmap_height-1:
                    diff = np.array(
                        [
                            hm[py][px+1] - hm[py][px-1],
                            hm[py+1][px]-hm[py-1][px]
                        ]
                    )
                    coords[n][p] += np.sign(diff) * .25

    preds = coords.copy()

    # Transform back
    for i in range(coords.shape[0]):
        preds[i] = transform_preds(
            coords[i], center[i], scale[i], [heatmap_width, heatmap_height]
        )

    return preds, maxvals
```

```
# 在lib/utils/transforms.py中
def transform_preds(coords, center, scale, output_size):
    target_coords = np.zeros(coords.shape)
    trans = get_affine_transform(center, scale, 0, output_size, inv=1)
    for p in range(coords.shape[0]):
        target_coords[p, 0:2] = affine_transform(coords[p, 0:2], trans)
    return target_coords
```

- UDP使用的无偏数据转换方法

Finally the prediction ${}^{s}\mathbf{k}$ in source image space should be obtained by the following inverse transformation:

$$
{}^{s}\hat{\mathbf{k}} = \begin{bmatrix} \frac{{}^{s}w_b}{{}^{p}w_g - 1}\cos\theta & \frac{{}^{s}h_b}{{}^{p}h_g - 1}\sin\theta & -0.5{}^{s}w_b\cos\theta - 0.5{}^{s}h_b\sin\theta + {}^{s}x_b \\ -\frac{{}^{s}w_b}{{}^{p}w_o - 1}\sin\theta & \frac{{}^{s}h_b}{{}^{p}h_o - 1}\cos\theta & 0.5{}^{s}w_b\sin\theta - 0.5{}^{s}h_b\cos\theta + {}^{s}y_b \\ 0 & 0 & 1 \end{bmatrix} {}^{o}\hat{\mathbf{k}} \tag{47}
$$

```
# 在lib/core/inference.py中
def get_final_preds(config, batch_heatmaps, center, scale):
    heatmap_height = batch_heatmaps.shape[2]
    heatmap_width = batch_heatmaps.shape[3]
    if config.MODEL.TARGET_TYPE == 'gaussian':
        coords, maxvals = get_max_preds(batch_heatmaps)
        if config.TEST.POST_PROCESS:
            coords = post(coords,batch_heatmaps)
    elif config.MODEL.TARGET_TYPE == 'offset':
        net_output = batch_heatmaps.copy()
        kps_pos_distance_x = config.LOSS.KPD
        kps_pos_distance_y = config.LOSS.KPD
        batch_heatmaps = net_output[:,::3,:]
        offset_x = net_output[:,1::3,:] * kps_pos_distance_x
        offset_y = net_output[:,2::3,:] * kps_pos_distance_y
        for i in range(batch_heatmaps.shape[0]):
            for j in range(batch_heatmaps.shape[1]):
                batch_heatmaps[i,j,:,:] =
cv2.GaussianBlur(batch_heatmaps[i,j,:,:],(15, 15), 0)
                offset_x[i,j,:,:] = cv2.GaussianBlur(offset_x[i,j,:,:],(7, 7),
0)
                offset_y[i,j,:,:] = cv2.GaussianBlur(offset_y[i,j,:,:],(7, 7),
0)
        coords, maxvals = get_max_preds(batch_heatmaps)
        for n in range(coords.shape[0]):
            for p in range(coords.shape[1]):
                px = int(coords[n][p][0])
                py = int(coords[n][p][1])
                coords[n][p][0] += offset_x[n,p,py,px]
                coords[n][p][1] += offset_y[n,p,py,px]

    preds = coords.copy()

    # Transform back
    for i in range(coords.shape[0]):
        preds[i] = transform_preds(
            coords[i], center[i], scale[i], [heatmap_width, heatmap_height]
        )

    return preds, maxvals
```

```
# 在lib/utils/transforms.py中
def transform_preds(coords, center, scale, output_size):
    scale = scale * 200.0
    scale_x = scale[0]/(output_size[0]-1.0)
    scale_y = scale[1]/(output_size[1]-1.0)
    target_coords = np.zeros(coords.shape)
    target_coords[:,0] = coords[:,0]*scale_x + center[0]-scale[0]*0.5
    target_coords[:,1] = coords[:,1]*scale_y + center[1]-scale[1]*0.5
    return target_coords
```

2.2 编解码

2.2.1 编码

- simple baselines/hrnet使用的有统计误差的编码方法

$$\mathcal{H}(x, y, \mathbf{k}_q) = exp(-\frac{(x-m_q)^2+(y-n_q)^2}{2\delta^2}) \quad (17)$$

```
# lib/dataset/JointDataset.py中的def generate_target(self, joints, joints_vis):函数中
    def generate_target(self, joints, joints_vis):
        '''
        :param joints:  [num_joints, 3]
        :param joints_vis: [num_joints, 3]
        :return: target, target_weight(1: visible, 0: invisible)
        '''
        # target_weight形状为[17, 1]
        target_weight = np.ones((self.num_joints, 1), dtype=np.float32)
        target_weight[:, 0] = joints_vis[:, 0]

        # 检测制作热图的方式是否为gaussian，如果不是则报错
        assert self.target_type == 'gaussian', \
            'Only support gaussian map now!'

        # 如果使用高斯模糊的方法制作热图
        if self.target_type == 'gaussian':
            # 形状为[17, 64, 48]
            target = np.zeros((self.num_joints,
                               self.heatmap_size[1],
                               self.heatmap_size[0]),
                              dtype=np.float32)

            # self.sigma 默认为2，tmp_size=6
            tmp_size = self.sigma * 3

            # 为每个关键点生成热图target以及对应的热图权重target_weight
            for joint_id in range(self.num_joints):
                # 先计算出原图到输出热图的缩小倍数
                feat_stride = self.image_size / self.heatmap_size

                # 计算输入原图的关键点，转换到热图的位置
                mu_x = int(joints[joint_id][0] / feat_stride[0] + 0.5)
                mu_y = int(joints[joint_id][1] / feat_stride[1] + 0.5)

                # Check that any part of the gaussian is in-bounds
                # 根据tmp_size参数，计算出关键点范围左上角和右下角坐标
```

```
                ul = [int(mu_x - tmp_size), int(mu_y - tmp_size)]
                br = [int(mu_x + tmp_size + 1), int(mu_y + tmp_size + 1)]

                # 判断该关键点是否处于热图之外；如果处于热图之外，则把该热图对应的
target_weight设置为0，然后continue
                if ul[0] >= self.heatmap_size[0] or ul[1] >= self.heatmap_size[1]
\
                        or br[0] < 0 or br[1] < 0:
                    # If not, just return the image as is
                    target_weight[joint_id] = 0
                    continue

                # # Generate gaussian
                # 产生高斯分布的大小
                size = 2 * tmp_size + 1
                # x[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12.]
                x = np.arange(0, size, 1, np.float32)
                # y[[ 0.][ 1.][ 2.][ 3.][ 4.][ 5.][ 6.][ 7.][ 8.][ 9.][10.][11.]
[12.]]
                y = x[:, np.newaxis]
                # x0 = y0 = 6
                x0 = y0 = size // 2

                # The gaussian is not normalized, we want the center value to
equal 1
                # g形状[13, 13]，该数组中间的[7, 7]=1，离开该中心点越远数值越小
                g = np.exp(- ((x - x0) ** 2 + (y - y0) ** 2) / (2 * self.sigma **
2))

                # Usable gaussian range
                # 判断边界，获得有效高斯分布的范围
                g_x = max(0, -ul[0]), min(br[0], self.heatmap_size[0]) - ul[0]
                g_y = max(0, -ul[1]), min(br[1], self.heatmap_size[1]) - ul[1]

                # Image range
                # 判断边界，获得有有效的图片像素边界
                img_x = max(0, ul[0]), min(br[0], self.heatmap_size[0])
                img_y = max(0, ul[1]), min(br[1], self.heatmap_size[1])

                # 如果该关键点对应的target_weight>0.5(即表示该关键点可见)，则把关键点附近
的特征点赋值成gaussian
                v = target_weight[joint_id]
                if v > 0.5:
                    target[joint_id][img_y[0]:img_y[1], img_x[0]:img_x[1]] = \
                        g[g_y[0]:g_y[1], g_x[0]:g_x[1]]

        # 如果各个关键点训练权重不一样
        if self.use_different_joints_weight:
            target_weight = np.multiply(target_weight, self.joints_weight)

        # img = np.transpose(target.copy(),[1,2,0])*255
        # img = img[:,:,0].astype(np.uint8)
        # img = np.expand_dims(img,axis=-1)
        # cv2.imwrite('./test.jpg', img) # 关键点的热图

        return target, target_weight
```

- 本文使用的offset-based编码方法

Inspired by [23], this paper adopts the offset-based encoding-decoding method, whose expected value of error is zero. Each ground truth label point $\mathbf{k} = (m, n)$ is encoded into one heatmap:

$$\mathcal{H}(x, y, \mathbf{k}) = \begin{cases} 1 & \textbf{if } (x-m)^2 + (y-n)^2 < R \\ 0 & \textbf{otherwise} \end{cases} \tag{24}$$

and two offset maps:

$$\mathcal{X}(x, y, \mathbf{k}) = m - x \tag{25}$$

$$\mathcal{Y}(x, y, \mathbf{k}) = n - y \tag{26}$$

```
# lib/dataset/JointDataset.py中的def generate_target(self, joints, joints_vis):函数中
        elif self.target_type == 'offset':
            # self.heatmap_size: [48,64] [w,h]
            target = np.zeros((self.num_joints,
                               3,
                               self.heatmap_size[1]*
                               self.heatmap_size[0]),
                              dtype=np.float32)
            feat_width = self.heatmap_size[0]
            feat_height = self.heatmap_size[1]
            feat_x_int = np.arange(0, feat_width)
            feat_y_int = np.arange(0, feat_height)
            feat_x_int, feat_y_int = np.meshgrid(feat_x_int, feat_y_int)
            feat_x_int = feat_x_int.reshape((-1,))
            feat_y_int = feat_y_int.reshape((-1,))
            kps_pos_distance_x = self.kpd
            kps_pos_distance_y = self.kpd
            feat_stride = (self.image_size - 1.0) / (self.heatmap_size - 1.0)
            for joint_id in range(self.num_joints):
                # 计算输入原图的关键点，转换到热图的位置
                mu_x = joints[joint_id][0] / feat_stride[0]
                mu_y = joints[joint_id][1] / feat_stride[1]
                # Check that any part of the gaussian is in-bounds

                x_offset = (mu_x - feat_x_int) / kps_pos_distance_x
                y_offset = (mu_y - feat_y_int) / kps_pos_distance_y

                dis = x_offset ** 2 + y_offset ** 2
                keep_pos = np.where((dis <= 1) & (dis >= 0))[0]
                v = target_weight[joint_id]
                if v > 0.5:
                    target[joint_id, 0, keep_pos] = 1
                    target[joint_id, 1, keep_pos] = x_offset[keep_pos]
                    target[joint_id, 2, keep_pos] = y_offset[keep_pos]

target=target.reshape((self.num_joints*3,self.heatmap_size[1],self.heatmap_size[0]))
```

2.2.2 解码

- simple baselines/hrnet使用的有统计误差的解码方法

$$\hat{\mathbf{k}}_q = (\hat{m}_q, \hat{n}_q) = argmax(\hat{\mathcal{H}}) \qquad (18)$$

```
# 在lib/core/inference.py中
def get_max_preds(batch_heatmaps):
    '''
    get predictions from score maps
    heatmaps: numpy.ndarray([batch_size, num_joints, height, width])
    '''
    assert isinstance(batch_heatmaps, np.ndarray), \
        'batch_heatmaps should be numpy.ndarray'
    assert batch_heatmaps.ndim == 4, 'batch_images should be 4-ndim'

    batch_size = batch_heatmaps.shape[0]
    num_joints = batch_heatmaps.shape[1]
    width = batch_heatmaps.shape[3]
    heatmaps_reshaped = batch_heatmaps.reshape((batch_size, num_joints, -1))
    idx = np.argmax(heatmaps_reshaped, 2)
    maxvals = np.amax(heatmaps_reshaped, 2)

    maxvals = maxvals.reshape((batch_size, num_joints, 1))
    idx = idx.reshape((batch_size, num_joints, 1))

    preds = np.tile(idx, (1, 1, 2)).astype(np.float32)

    preds[:, :, 0] = (preds[:, :, 0]) % width
    preds[:, :, 1] = np.floor((preds[:, :, 1]) / width)

    pred_mask = np.tile(np.greater(maxvals, 0.0), (1, 1, 2))
    pred_mask = pred_mask.astype(np.float32)

    preds *= pred_mask
    return preds, maxvals
```

- 本文使用的offset-based编码方法

$$\hat{\mathbf{k}} = \hat{\mathbf{k}}_h + [\hat{\mathcal{X}}(\hat{\mathbf{k}}_h, \mathbf{k}) \otimes K, \hat{\mathcal{Y}}(\hat{\mathbf{k}}_h, \mathbf{k}) \otimes K]^T \qquad (29)$$

```
# lib/core/inference.py中的def get_final_preds(config, batch_heatmaps, center,
scale):函数中
    elif config.MODEL.TARGET_TYPE == 'offset':
        net_output = batch_heatmaps.copy()
        kps_pos_distance_x = config.LOSS.KPD
        kps_pos_distance_y = config.LOSS.KPD
        batch_heatmaps = net_output[:,::3,:]
        offset_x = net_output[:,1::3,:] * kps_pos_distance_x
        offset_y = net_output[:,2::3,:] * kps_pos_distance_y
        for i in range(batch_heatmaps.shape[0]):
            for j in range(batch_heatmaps.shape[1]):
                batch_heatmaps[i,j,:,:] =
cv2.GaussianBlur(batch_heatmaps[i,j,:,:],(15, 15), 0)
                offset_x[i,j,:,:] = cv2.GaussianBlur(offset_x[i,j,:,:],(7, 7),
0)
```

```
                offset_y[i,j,:,:] = cv2.GaussianBlur(offset_y[i,j,:,:],(7, 7),
0)
        coords, maxvals = get_max_preds(batch_heatmaps)
        for n in range(coords.shape[0]):
            for p in range(coords.shape[1]):
                px = int(coords[n][p][0])
                py = int(coords[n][p][1])
                coords[n][p][0] += offset_x[n,p,py,px]
                coords[n][p][1] += offset_y[n,p,py,px]
```

3. 参考资料

问雪更新于2021-01-23