TransPose: Towards Explainable Human Pose Estimation by Transformer笔记

- Paper: TransPose: Towards Explainable Human Pose Estimation by Transformer
- Code: yangsenius/TransPose

0. Summary Keywords

- interpretable
- lightweight
- efficient
- SOTA

1. Introduction

1.1 Why

1.1.1 Motivation

- 当前，对于CNN如何定位人体关键点，尚未有显式的理解；而且模型中学习到的结构化变量之间的空间依赖关系也是未知的。(There is **no explicit understanding of how** the locations of body keypoints are predicted by CNN, and it is also unknown what spatial dependency relationships between structural variables are learned in the model.)

1.1.2 Challenges

关键点检测可解释性方面的障碍：

- Deepness：CNN网络往往很深，我们不能很轻松地指出每一层扮演什么样的角色，尤其是在深层。
- Relationships and features are coupled：空间关系和特征图是耦合的。身体各部分之间的整体空间关系不仅隐式地编码在了卷积核的参数中，而且也体现在了神经元的激活值中。如果只看中间特征激活值，是不能揭示空间关系的。而且，从单纯从数值里解耦和理解那些耦合的关系是具有挑战的。
- Limited memory and expressiveness for large numbers of images：我们期望的可解释性应该是可变的、与图像有特定关联性的、细粒度的。CNN在进行推理的时候，只有静态的卷积核参数，由于CNN起作用的记忆力是有限的，所以它在表征变化特性方面的能力是有限的，因此让CNN来显式地展示与图像相关的空间依赖关系的变化特性是存在困难的，因为在很大的关节结构空间中人体的姿态变化是很大的。
- Lack of tools：当前的方法没能解释相关变量和结构化预测变量之间的细粒度依赖关系。

1.2 What

- **TransPose**：本文基于CNN模块(浅层)和Transformer结构，建立了一个可解释的模型。(To explore these questions, we construct an explainable model named TransPose based on Transformer architecture and low-level convolutional blocks.)
    - TransPose-R: ResNet-S(backbone)（这里的“S”表示的是simplicity，作者只采用了刚开始的部分用于提取浅层特征）
    - TransPose-H: HRNet-S(backbone)（这里的“S”表示的是simplicity，作者只采用了刚开始的部分用于提取浅层特征，只用了HRNet的前3个Stage）

1.3 How

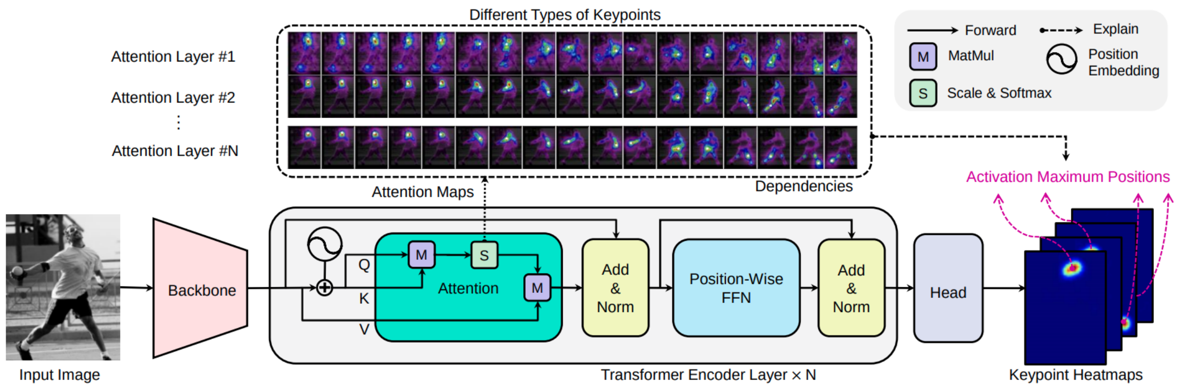
- Transformer中的注意力层能够捕获关键点之间的长距离空间关系，同时解释了所预测的关键点在定位的时候更依靠哪些依赖关系。(Given an image, the attention layers built in Transformer can capture **long-range spatial relationships** between keypoints and explain what **dependencies** the predicted keypoints locations highly rely on.)
- 所揭示的依赖关系是与图像特定关联的，而且是随关键点类型、网络层深度、训练模型的不同而变化的。(The revealed dependencies are image-specific and variable for different keypoint types, layer depths, or trained models.)

1.4 Contributions

- 本文是第一篇应用Transformer结构来捕捉结构化的人体部位之间的空间关系的作品。(To the best of our knowledge, we are the first to use Transformer architecture to capture the spatial relationships between structural human body parts.)
- 本文提出的TransPose能够解释所预测的关键点依靠什么样的空间依赖关系。定性分析揭示了细粒度的空间依赖关系，它是与图像特定关联的，而且是随关键点类型、网络层深度、训练模型的不同而变化的。(TransPose can explain what spatial dependencies the predicted keypoints rely on. Qualitative analysis reveals the fine-grained spatial dependencies, which are image-specific and variable for different keypoint types, layer depths, and trained models.)
- 精度高、参数少、速度快。(TransPose achieves 75.8 AP and 75.0 AP on COCO validation set and testdev set, with 73% fewer parameters and 1.4× faster than HRNet-W48.)
- 本文发现：位置编码对于准确预测关键点的位置而言很重要，而且它在预测未曾见过的分辨率的图像时泛化性能更好。(We find that position embedding is important for accurately predicting the 2D positions of keypoints and helps model generalize better on unseen input resolutions.)

2. Method

2.1 Architecture



IPO结构：

- Input: Image
- Process
    - Backbone
    - Transformer Encoder
    - Head
- Output: Keypoint Heatmaps

注意：本文只采用了Transformer中的编码器，作者认为纯的heatmap预测任务是一个简单的编码任务。（回想起HRNet及在其基础上的改进论文，其中的encoder就是指将关键点坐标编码为heatmap，decoder就是指将神经网络生成的heatmap解码成关键点坐标。再联想起PRTR，它是基于回归的关键点检测算法，用到了Transformer的解码器直接预测出关键点坐标。）

2.2 Resolution Settings

CNN相关的方法采用下采样的方式将图像分辨率降到32倍，以期能获取全局信息。与之不同的是，本文的方法在更高分辨率的特征图上能直接捕获长距离的相互作用，同时能保留细粒度的局部特征信息。(By contrast, our model can directly capture long-range interactions at a higher resolution, while preserving the fine-grained local feature information.)

2.3 Position Embedding

image patches拉平后，如果不加位置信息，Transformer编码就成了变换恒等结构(联想交换律？)了。这显然是不合适的。

疑问：2D Sine Position Embedding?

位置编码使模型在对未曾见过的分辨率的图像进行预测的时候泛化性能更强，因为固定感受野的CNN很难适应特征尺度中的变化，而相对位置信息编码能帮助模型泛化到不同尺度的输入图像或特征图。(We conjecture that: 1) it is hard for the models with a fixed receptive field to adapt the changes in the feature scale; 2) building associations with relative position information encoded in Sine position embedding [44] may help the model generalize on different sizes of inputs or feature maps.)

2.4 Explaining by Attention

Transformer编码器中的最后一个注意力层，它的注意力得分数可以看成动态权重(依赖图像的权重)，这个注意力层对预测有最直接的影响。(Specifically, the last attention layer in Transformer Encoder, whose attention scores are seen as the dynamic (image-dependent) weights, has the most direct effect on the predictions.)

疑问：动态权重?

详细推导可以参考附录G中的梯度分析。

针对某个预测的关键点i，如果那些点集合J中的各个点j对点i具有较高的注意力得分数，那么说明它们与点i有依赖关系，它们对这个预测点i有较大的贡献。(Those locations J, whose element j has higher attention score (≥ δ) with i, are the dependencies that significantly contribute to the prediction.)

对于最后一个attention层的attention map：

- Ai,:可以展示1个预测的关键点i高度依靠哪些依赖关系，作者称之为依赖区域；
- A:,j可以展示1个特定的点j主要影响哪些位置，作者称之为影响区域。

3. Code=f(method)

- 以在COCO数据集Train以HRNet为Backbone的TransPose模型为例：

```
python tools/train.py --cfg
experiments/coco/transpose_h/TP_H_w32_256x192_stage3_1_4_d64_h128_relu_enc4_mh1.
yaml
```

- 以experiments/coco/transpose_h/TP_H_w32_256x192_stage3_1_4_d64_h128_relu_enc4_mh1.yaml为例：

```yaml
AUTO_RESUME: true
CUDNN:
  BENCHMARK: true
  DETERMINISTIC: false
  ENABLED: true
DATA_DIR: ''
GPUS: (0,1)
OUTPUT_DIR: 'output'
LOG_DIR: 'log'
WORKERS: 24
PRINT_FREQ: 100

DATASET:
  COLOR_RGB: true
  DATASET: 'coco'
  DATA_FORMAT: jpg
  FLIP: true
  NUM_JOINTS_HALF_BODY: 8
  PROB_HALF_BODY: 0.3
  ROOT: 'data/coco/'
  ROT_FACTOR: 45
  SCALE_FACTOR: 0.35
  TEST_SET: 'val2017'
  TRAIN_SET: 'train2017'
MODEL:
  # Transformer Encoder
  DIM_MODEL: 64
  DIM_FEEDFORWARD: 128
  N_HEAD: 1
  ENCODER_LAYERS: 4
  ATTENTION_ACTIVATION: relu
  POS_EMBEDDING: sine
  # #
  INIT_WEIGHTS: true
  NAME: transpose_h
  NUM_JOINTS: 17
  PRETRAINED: 'models/pytorch/imagenet/hrnet_w32-36af842e.pth'
  TARGET_TYPE: gaussian
  IMAGE_SIZE:
  - 192
  - 256
  HEATMAP_SIZE:
  - 48
  - 64
  SIGMA: 2
  EXTRA:
    PRETRAINED_LAYERS:   ##使用预训练模型中的某些层的权重来进行初始化。论文以HRNet为Backbone来提取特征时只用了3个Stage，所以模型做到很小。
    - 'conv1'
    - 'bn1'
    - 'conv2'
    - 'bn2'
    - 'layer1'
    - 'transition1'
    - 'stage2'
    - 'transition2'
    - 'stage3'
    FINAL_CONV_KERNEL: 1
    STAGE2:
      NUM_MODULES: 1
      NUM_BRANCHES: 2
      BLOCK: BASIC
      NUM_BLOCKS:
      - 4
      - 4
      NUM_CHANNELS:
      - 32
      - 64
      FUSE_METHOD: SUM
    STAGE3:
      NUM_MODULES: 4
      NUM_BRANCHES: 3
      BLOCK: BASIC
      NUM_BLOCKS:
      - 4
      - 4
      - 4
      NUM_CHANNELS:
      - 32
      - 64
      - 128
      FUSE_METHOD: SUM
LOSS:
  USE_TARGET_WEIGHT: true
TRAIN:
  BATCH_SIZE_PER_GPU: 16
  SHUFFLE: true
  BEGIN_EPOCH: 0
  END_EPOCH: 240
  OPTIMIZER: adam
  LR: 0.0001  # Initial learning rate
  LR_END: 0.00001  # Final learning rate
  LR_FACTOR: 0.25  # for MultiStepLR
  LR_STEP:  # for MultiStepLR
  - 100
  - 150
  - 200
  - 220
  WD: 0.1
  GAMMA1: 0.99
  GAMMA2: 0.0
  MOMENTUM: 0.9
  NESTEROV: false
TEST:
  BLUR_KERNEL: 11
  BATCH_SIZE_PER_GPU: 4
  COCO_BBOX_FILE:
'data/coco/person_detection_results/COCO_val2017_detections_AP_H_56_person.json'
  BBOX_THRE: 1.0
  IMAGE_THRE: 0.0
  IN_VIS_THRE: 0.2
  MODEL_FILE:
models/pytorch/transpose_coco/tp_h_32_256x192_enc4_d64_h128_mh1.pth
  NMS_THRE: 1.0
  OKS_THRE: 0.9
```

```
    USE_GT_BBOX: true
    FLIP_TEST: true
    POST_PROCESS: true
    SHIFT_HEATMAP: true
  DEBUG:
    DEBUG: true
    SAVE_BATCH_IMAGES_GT: true
    SAVE_BATCH_IMAGES_PRED: true
    SAVE_HEATMAPS_GT: true
    SAVE_HEATMAPS_PRED: true
```

3.1 Input

神经网络输入的是单个人体实例的图像，并将关键点的heatmaps作为GT Labels用于在训练过程中计算损失函数，与HRNet/DARK/UDP-Pose等的编码过程一致：

- 提取人体：原图坐标系统(source image coordinate system)->神经网络输入坐标系统(network input coordinate system)
- 由关键点坐标生成heatmaps：神经网络输入坐标系统(network input coordinate system)->神经网络输出坐标系统(network output coordinate system)

3.2 Process

lib/models/transpose_h.py

```
class TransPoseH(nn.Module):

    def __init__(self, cfg, **kwargs):
        self.inplanes = 64
        extra = cfg['MODEL']['EXTRA']
        super(TransPoseH, self).__init__()

        # stem net
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=2, padding=1,
                               bias=False)
        self.bn1 = nn.BatchNorm2d(64, momentum=BN_MOMENTUM)
        self.conv2 = nn.Conv2d(64, 64, kernel_size=3, stride=2, padding=1,
                               bias=False)
        self.bn2 = nn.BatchNorm2d(64, momentum=BN_MOMENTUM)
        self.relu = nn.ReLU(inplace=True)
        self.layer1 = self._make_layer(Bottleneck, 64, 4)

        self.stage2_cfg = extra['STAGE2']
        num_channels = self.stage2_cfg['NUM_CHANNELS']
        block = blocks_dict[self.stage2_cfg['BLOCK']]
        num_channels = [
            num_channels[i] * block.expansion for i in range(len(num_channels))
        ]
        self.transition1 = self._make_transition_layer([256], num_channels)
        self.stage2, pre_stage_channels = self._make_stage(
            self.stage2_cfg, num_channels)

        self.stage3_cfg = extra['STAGE3']
        num_channels = self.stage3_cfg['NUM_CHANNELS']
        block = blocks_dict[self.stage3_cfg['BLOCK']]
        num_channels = [
            num_channels[i] * block.expansion for i in range(len(num_channels))
        ]
        self.transition2 = self._make_transition_layer(
            pre_stage_channels, num_channels)
        self.stage3, pre_stage_channels = self._make_stage(
            self.stage3_cfg, num_channels, multi_scale_output=False)

        d_model = cfg.MODEL.DIM_MODEL
        dim_feedforward = cfg.MODEL.DIM_FEEDFORWARD
        encoder_layers_num = cfg.MODEL.ENCODER_LAYERS
        n_head = cfg.MODEL.N_HEAD
        pos_embedding_type = cfg.MODEL.POS_EMBEDDING
        w, h = cfg.MODEL.IMAGE_SIZE

        self.reduce = nn.Conv2d(pre_stage_channels[0], d_model, 1, bias=False)
        self._make_position_embedding(w, h, d_model, pos_embedding_type)

        encoder_layer = TransformerEncoderLayer(
            d_model=d_model, nhead=n_head, dim_feedforward=dim_feedforward,
            activation='relu')
        self.global_encoder = TransformerEncoder(
            encoder_layer, encoder_layers_num)

        self.final_layer = nn.Conv2d(
            in_channels=d_model,
            out_channels=cfg['MODEL']['NUM_JOINTS'],
            kernel_size=extra['FINAL_CONV_KERNEL'],
            stride=1,
            padding=1 if extra['FINAL_CONV_KERNEL'] == 3 else 0
        )

        self.pretrained_layers = extra['PRETRAINED_LAYERS']

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.conv2(x)
        x = self.bn2(x)
        x = self.relu(x)
        x = self.layer1(x)

        x_list = []
        for i in range(self.stage2_cfg['NUM_BRANCHES']):
            if self.transition1[i] is not None:
                x_list.append(self.transition1[i](x))
            else:
                x_list.append(x)
        y_list = self.stage2(x_list)

        x_list = []
        for i in range(self.stage3_cfg['NUM_BRANCHES']):
            if self.transition2[i] is not None:
                x_list.append(self.transition2[i](y_list[-1]))
            else:
                x_list.append(y_list[i])
        y_list = self.stage3(x_list)

        x = self.reduce(y_list[0])    ##卷积后，输出特征的通道数是64
        bs, c, h, w = x.shape  ##c为64，h为64，w为48
        x = x.flatten(2).permute(2, 0, 1)  ##变换后，x.shape为torch.Size([3072, BS, 64])
        x = self.global_encoder(x, pos=self.pos_embedding)  ##进入Transformer编码器层
        x = x.permute(1, 2, 0).contiguous().view(bs, c, h, w)
        x = self.final_layer(x)  ##输出heatmap

        return x
```

3.3 Output

神经网络输出的是单个人体实例的关键点的heatmaps，后处理与HRNet/DARK/UDP-Pose等的解码过程一致：

- 由heatmaps生成关键点坐标：神经网络输出坐标系统(network output coordinate system)->原图坐标系统(source image coordinate system)

更新于2021-05-27