TransPose: Keypoint Localization via Transformer — Supplementary Material

A. 2D Sine Position Embedding

Without the position information embedded in the input sequence, the Transformer Encoder is a permutation-equivariant architecture:

$$\text{Encoder}\left(\rho\left(\mathbf{X}\right)\right) = \rho\left(\text{Encoder}\left(\mathbf{X}\right)\right), \tag{1}$$

where ρ is any permutation for the pixel locations or the order of sequence. To make the order of sequence or the spatial structure of the image pixels matter, we follow the sine positional encodings but further hypothesize that the position information is independent at x (horizontal) and y (vertical) direction of an image, like the ways of [4, 2]. Concretely, we keep the original 2D-structure respectively with $d/2$ channels for x, y-direction:

$$\begin{aligned} PE_{(2i,p_y,:)} &= \sin\left(2\pi * p_y/(H * 10000^{2i/\frac{d}{2}})\right), \\ PE_{(2i+1,p_y,:)} &= \cos\left(2\pi * p_y/(H * 10000^{2i/\frac{d}{2}})\right), \\ PE_{(2i,:,p_x)} &= \sin\left(2\pi * p_x/(W * 10000^{2i/\frac{d}{2}})\right), \\ PE_{(2i+1,:,p_x)} &= \cos\left(2\pi * p_x/(W * 10000^{2i/\frac{d}{2}})\right), \end{aligned} \tag{2}$$

where $i = 0, 1, ..., d/2 - 1$, p_x or p_y is the position index along x or y-direction. Then they are stacked and flattened into a shape $\mathbb{R}^{L\times d}$. The position embedding is injected into the input sequences before self-attention computation. We use 2D sine position embedding by default for all models.

B. What position information has been learned in the TransPose model with learnable position embedding?

We show what position information has been learned in the TransPose (TransPose-R) with learnable position embedding. It has been discussed in the paper. As shown in Fig. 1, we visualize the similarities by calculating the cosine similarity between vectors at any pair of locations of the learnable position embedding and reshaping it into a 2D grid-like map. We find that the embedding in each location of learnable position embedding has a unique vector value in the d-dim vector space, but it has relatively higher cosine similarity values with the neighbour locations in 2D-grid and lower values with those far away from it. The results indicate the coarse 2D position information has been implicitly learned in the learnable position embedding. We suppose that the learning sources of the position information might be the 2D-structure groundtruth heatmaps and the similar features existing in the 1D-structure sequences. The model learns to build associations between position embedding and input sequences, as a result it can predict the target heatmaps with 2D Gaussian peaking at groundtruth keypoints locations.

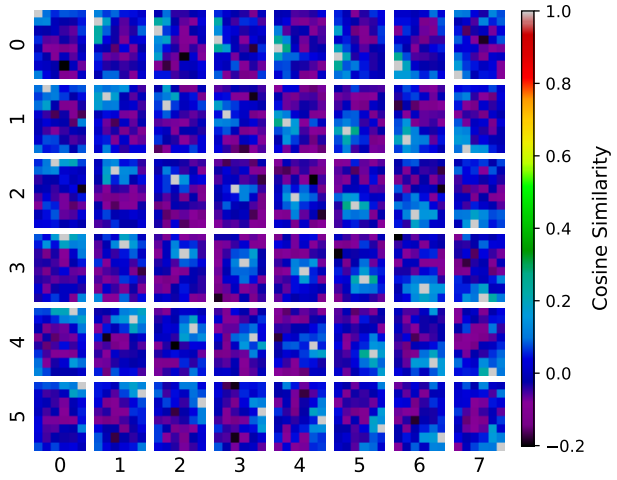


Figure 1. The cosine similarities between the learned position embedding vectors, which have been reshaped into 2D grid and interpolated with 0.25 scale factor for a better illustration (the original shape is $(24, 32)$). Each map in x-row and y-col of the figure represents the cosine similarities between the embedding vector in position (x, y) and the embedding vectors at other locations.

In the paper, we find that position embedding helps to generalize better on unseen input resolutions, particularly 2D sine position embedding. We conjecture that 1) the models with a fixed receptive field may be hard to adapt the changes in scales; 2) building associations with *position information* encoded in Sine position embedding [9] may help model generalize better on different sizes.

C. Transformer Encoder Layer

The Transformer Encoder layer [9] we used can be formulated as:

$$\begin{aligned}\mathbf{Z} &= \mathrm{LayerNorm}\left(\mathrm{MultiheadSelfAttention}\left(\mathbf{X}\right)+\mathbf{X}\right),\\ \mathbf{X}^{*} &= \mathrm{LayerNorm}\left(\mathrm{FFN}\left(\mathbf{Z}\right)+\mathbf{Z}\right),\end{aligned} \tag{3}$$

where $\mathbf{X}$ is the original input sequence that has not yet been added with position embedding. The position embedding will be added to $\mathbf{X}$ for computing querys and keys excluding values. $\mathbf{X}^{*}$ is the output sequence of the current Transformer Encoder layer, as the input sequence of next encoder layer. The formulations of Multihead Self-Attention and FFN are defined in [9].

D. Gradient Analysis

From the view of an activation at some location i of the predicted heatmaps, the network weights associating all input tokens across the whole image/sequence with this activation can be seen as a discriminator that judges the presence or absence of a certain keypoint at this location. As revealed by [5, 7, 1, 6], the gradient information can indicate the importance (sensitivity) of the input features to a specific output of a non-linear model. That assumption is based on that tiny change in the input (pixel/feature/token) with the most important feature value causes a large change in what the output of the model would be.

Suppose we have a trained model and a specific image, $\boldsymbol{h}_i \in \mathbb{R}^K$ is the scores for all K types of keypoints at location i of the predicted heatmaps; $\boldsymbol{z}_i \in \mathbb{R}^d$ is the intermediate feature outputted by the last self-attention layer before being fed into FFN. There is only a ReLU excluding the linear and convolutions[1] (head) layers after the last attention layer. ReLU (rectified linear unit) activation function in FFN can be empirically regarded as a negative contribution filter, which only retains positive contributions and maintains the linearity. Next, we choose numerator layout for computing the derivative of a vector with respect to a vector. We thus assume the mapping from $\boldsymbol{z}_i$ to $\boldsymbol{h}_i$ can be approximated as a linear function f with learned weights $\mathbf{W}_f \in \mathbb{R}^{K\times d}$ and bias $\mathbf{b} \in \mathbb{R}^K$ by computing the first-order Taylor expansion at a given local point $\boldsymbol{z}_i^0$, *i.e.*, $\boldsymbol{h}_i \approx \mathbf{W}_f\boldsymbol{z}_i + \mathbf{b}$, $\mathbf{W}_f = \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{z}_i}\Big|_{\boldsymbol{z}_i^0}$. Then we compute the partial derivative of $\boldsymbol{h}_i$ at location i of the output heatmaps w.r.t the token $\boldsymbol{x}_j$ at location j of the input sequence of the last attention layer:

$$\begin{aligned}\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{x}_j} &= \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{z}_i}\frac{\partial \boldsymbol{z}_i}{\partial \boldsymbol{x}_j}\\ &= \frac{\partial f(\boldsymbol{z}_i)}{\partial \boldsymbol{z}_i}(\mathbf{1}+\frac{\partial \mathbf{w}_i\mathbf{V}}{\partial \boldsymbol{x}_j})\\ &\approx \mathbf{W}_f(\mathbf{1}+\frac{\partial \mathbf{w}_{i,0}\boldsymbol{v}_0+...+\mathbf{w}_{i,j}\boldsymbol{v}_j+...+\mathbf{w}_{i,L-1}\boldsymbol{v}_{L-1}}{\partial \boldsymbol{x}_j})\\ &= \mathbf{W}_f(\mathbf{1}+\frac{\partial \mathbf{w}_{i,j}\boldsymbol{v}_j}{\partial \boldsymbol{x}_j})\\ &= \mathbf{W}_f(\mathbf{1}+\frac{\partial \mathbf{A}_{i,j}\mathbf{W}_v^{\top}\boldsymbol{x}_j}{\partial \boldsymbol{x}_j})\end{aligned} \tag{4}$$

where $\boldsymbol{v}_j \in \mathbb{R}^d$ is the value vector transformed by: $\boldsymbol{v}_j = \mathbf{W}_v^{\top}\boldsymbol{x}_j$. $\mathbf{A}_{i,j}$ is a scalar value that is computed by the dot-product between $\boldsymbol{q}_i$ and $\boldsymbol{k}_j$. We assume $G := \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{x}_j}$ as a function w.r.t. a given attention score $\mathbf{A}_{i,j}$. Under this assumption $\mathbf{A}_{i,j}$ is deemed as an observed variable that has blocked its parent nodes. Then we define:

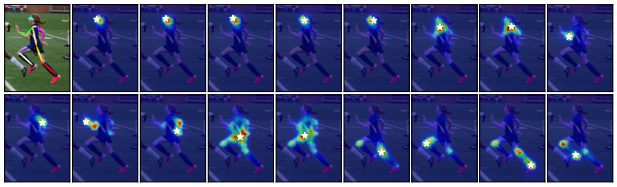
$$\begin{aligned}G\left(\mathbf{A}_{i,j}\right) &= \mathbf{W}_f(\mathbf{1}+\frac{\partial \mathbf{A}_{i,j}\mathbf{W}_v^{\top}\boldsymbol{x}_j}{\partial \boldsymbol{x}_j})\\ &= \mathbf{W}_f\left(\mathbf{1}+\mathbf{A}_{i,j}\mathbf{W}_v^{\top}\right)\\ &= \mathbf{A}_{i,j}\mathbf{W}_f\mathbf{W}_v^{\top}+\mathbf{W}_f\\ &= \underbrace{\mathbf{A}_{i,j}}_{\textbf{Image-Specific: dynamic weights}}\cdot\underbrace{\mathbf{W}_f\cdot\mathbf{W}_v^{\top}+\mathbf{W}_f}_{\textbf{Learned: static weights}}\\ &= \mathbf{A}_{i,j}\cdot\mathbf{K}+\mathbf{B}\end{aligned} \tag{5}$$

where $\mathbf{K},\mathbf{B} \in \mathbb{R}^{K\times d}$ are static weights shared across all positions. We can see that the function G is approximately linear with $\mathbf{A}_{i,j}$, *i.e.*, the degrees of contribution to the prediction $\boldsymbol{h}_i$ directly depend on its attention scores at those locations.

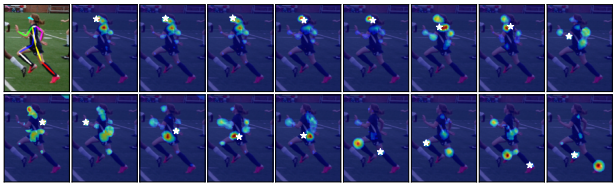
The last attention layer in Transformer Encoder, whose attention scores are seen as the image-specific weights, aggregate contributions from all locations according to attention scores and finally form the maximum activations in the output heatmaps. Though the layers in FFN and head cannot be ignored[2], they are position-wise operators, which almost linearly transform the attention scores from all the positions with the same transformation. In addition, $\mathbf{Q} = (\mathbf{X}+\mathbf{P})\,\mathbf{W}_q, \mathbf{K} = (\mathbf{X}+\mathbf{P})\,\mathbf{W}_k, \mathbf{V} = \mathbf{X}\mathbf{W}_v$ where $\mathbf{P}$ is the position embedding. Because $\mathbf{A}_{i,j} \propto \mathbf{Q}_i\mathbf{K}_j^{\top}$, the position embedding values also affect the attention scores to some extent.

[1] a 1×1 convolution is also a position-wise linear layer; the 4×4 deconvolution used in TP-R acts as the upsampling operation.

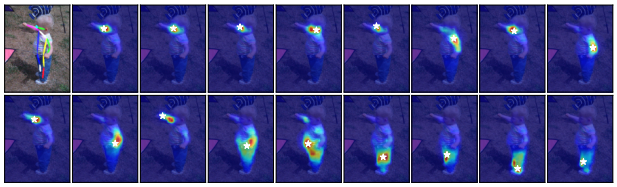
[2] 1. Assuming that the used convolutions extract feature in a limited patch, the global interactions mostly occur at the attention layers. 2. The layer normalization does not affect the interactions between locations.



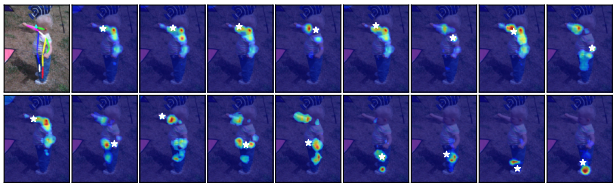
(a) **TP-R-A4:** predictions and dependency areas for **input 1**.



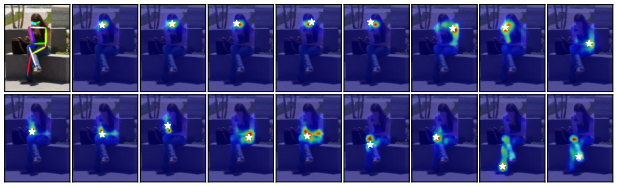
(b) **TP-H-A4:** predictions and dependency areas for **input 1**.



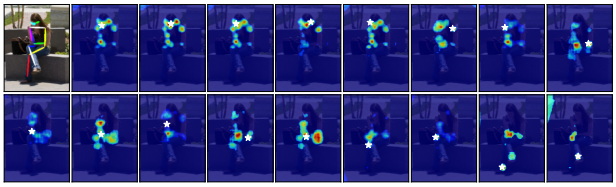
(c) **TP-R-A4:** predictions and dependency areas for **input 2**.



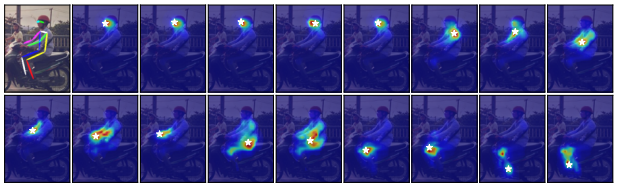
(d) **TP-H-A4:** predictions and dependency areas for **input 2**.



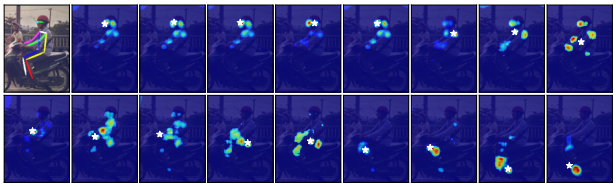
(e) **TP-R-A4:** predictions and dependency areas for **input 3**.



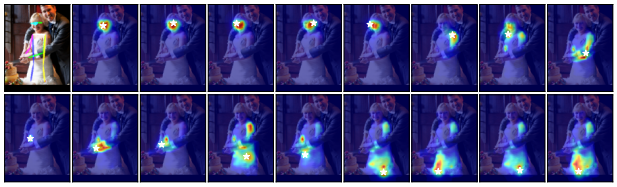
(f) **TP-H-A4:** predictions and dependency areas for **input 3**.



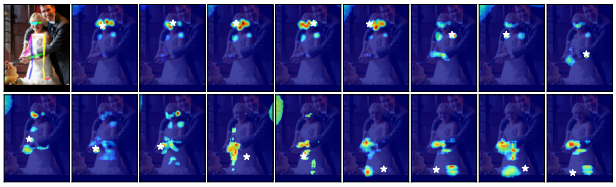
(g) **TP-R-A4:** predictions and dependency areas for **input 4**.



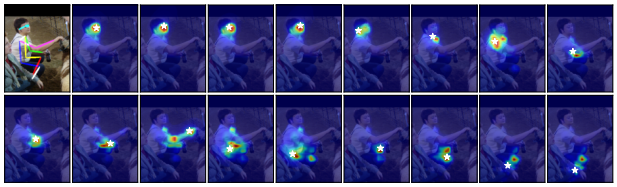
(h) **TP-H-A4:** predictions and dependency areas for **input 4**.



(i) **TP-R-A4:** predictions and dependency areas for **input 5**.



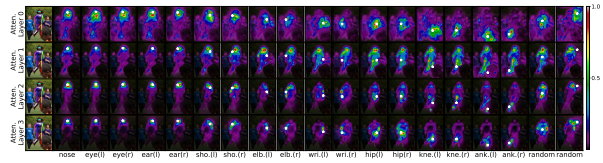
(j) **TP-H-A4:** predictions and dependency areas for **input 5**.



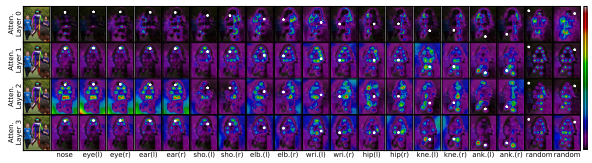
(k) **TP-R-A4:** predictions and dependency areas for **input 5**.

(l) **TP-H-A4:** predictions and dependency areas for **input 5**.

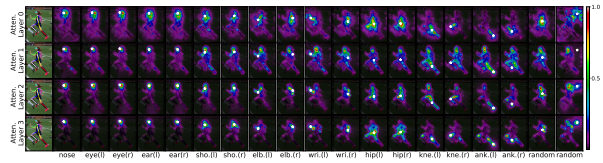
Figure 2. Predicted locations and the dependency areas for different types of keypoints in different models: TP-R-A4 (left column) and TP-H-A4 (right column). In each sub-figure, the first one is the original input image plotted with predicted skeleton. The other maps visualized by the defined dependency area ($A_{i,:}$) of the attention matrix in the last layer with a threshold value (0.00075). The predicted location of a keypoint is annotated by a WHITE color pentagram ($\star$) in each sub-map. Redder area indicates higher attention scores.



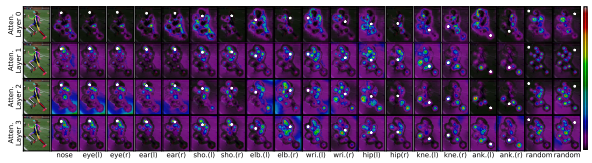
(a) **TP-R-A4:** predictions and dependency areas of each keypoint in different attention layers.



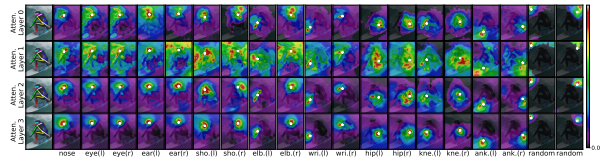
(b) **TP-H-A4:** predictions and dependency areas of each keypoint in different attention layers.



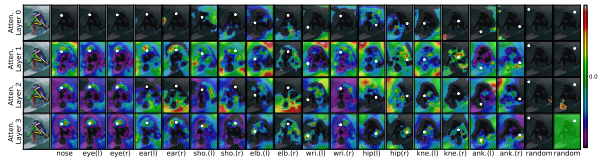
(c) **TP-R-A4:** predictions and dependency areas of each keypoint in different attention layers.



(d) **TP-H-A4:** predictions and dependency areas of each keypoint in different attention layers.



(e) **TP-R-A4:** predictions and affect areas of each keypoint in different attention layers.



(f) **TP-H-A4:** predictions and affect areas of each keypoint in different attention layers.

Figure 3. **Dependency areas** (the first two rows) and **Affected areas** (the last row) in different attention layers for different input images.

Backbone	ResNet-S
Stem	Conv-k7-s2-c64, BN, ReLU Pooling-k3-s2
Blocks	3×Bottleneck-c64 Bottleneck-s2-c128 3×Bottleneck-c128 Conv-k1-s1-c256

Table 1. The detailed configurations for ResNet-S. Conv-k7-s2-c64 means a convolutional layer with 7×7 kernel size, 2 stride, and 64 output channels, followed by a BN and ReLU; the same below. The Bottleneck-c64 includes Conv-k1-s1-c64-BN-ReLU, Conv-k3-s1-c64-BN-ReLU, and Conv-k1-s1-c256-BN. Bottleneck-c128 includes Conv-k1-s1-c128-BN-ReLU, Conv-k3-s1-c128-BN-ReLU, and Conv-k1-s1-c512-BN. See details in [3].

Backbone	HRNet-S-W32(48)
Stem	Conv-k3-s2-c64, BN, ReLU Conv-k3-s2-c64, BN, ReLU 4×Bottleneck-c64
Blocks	transition1∼stage2 transition2∼stage3 Conv-k1-s1-c64(92)

Table 2. The detailed configurations for HRNet-S-W32(48). More detailed information about the transition layer and stage blocks are described in the HRNet paper [8].

E. Architecture Details

We report the architecture details of ResNet-S and HRNet-S-W32(48) in Tab. 1 and Tab. 2. The ResNet-S* only differs from ResNet-S in that ResNet-S* has 10 Bottleneck-c128 blocks. More details about HRNet-W32 and HRNet-W48 are described in [8].

F. More Attention Maps Visualizations

In this section, we show more visualization results of the attention maps from TP-R-A4 (TransPose-R-A4) and TP-H-A4 (TransPose-H-A4) models. The attention maps of the last attention layers of two models are shown in Fig. 2. The attention maps in different attention layers of two models are shown in Fig. 3.

References

[1] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015. 2

[2] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*, pages 213–229, Cham, 2020. 1

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 4

[4] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *ICML*, 2018. 1

[5] Wojciech Samek, Grégoire Montavon, Andrea Vedaldi, Lars Kai Hansen, and Klaus-Robert Müller. *Explainable AI: interpreting, explaining and visualizing deep learning*, volume 11700. Springer Nature, 2019. 2

[6] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *ICCV*, pages 618–626, 2017. 2

[7] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *ICLR (Workshop Poster)*, 2013. 2

[8] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. Deep high-resolution representation learning for human pose estimation. In *CVPR*, June 2019. 4

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, pages 5998–6008, 2017. 1, 2