Efficient Two-Stage Detection of Human–Object Interactions with a Novel Unary–Pairwise Transformer Supplementary Materials

Frederic Z. Zhang[1,3] Dylan Campbell[2,3] Stephen Gould[1,3]
[1]The Australian National University [2]University of Oxford
[3]Australian Centre for Robotic Vision
https://fredzzhang.com/unary-pairwise-transformers

A. Pairwise positional encodings

We describe the details of the pairwise positional encodings as introduced in Section 3.1 of the main paper. Formally, denote a bounding box as $\mathbf{b} = [x, y, w, h]^T \in [0, 1]^4$, where x and y represent the centre coordinates of the bounding box while w and h represent the width and height. Note that these values have been normalised by the image dimensions. For a pair of bounding boxes $\mathbf{b}_1$ and $\mathbf{b}_2$, we start by encoding the unary terms besides the box representation itself, including box areas and aspect ratios as below

$$\mathbf{u} = \mathbf{b}_1 \oplus \mathbf{b}_2 \oplus \left[w_1 h_1, w_2 h_2, \frac{w_1}{h_1}, \frac{w_2}{h_2} \right]^T, \tag{1}$$

where $\oplus$ denotes vector concatenation. We then proceed to encode the pairwise terms as follows

$$\mathbf{p} = \left[\frac{w_1 h_1}{w_2 h_2}, \texttt{IoU}(\mathbf{b}_1, \mathbf{b}_2) \right]^T \oplus f(d_x) \oplus f(d_y), \tag{2}$$

$$d_x = \frac{x_1 - x_2}{w_1}, d_y = \frac{y_1 - y_2}{h_1}, \tag{3}$$

$$f(d) = \left[\texttt{ReLU}(d), \texttt{ReLU}(-d) \right]^T. \tag{4}$$

This includes additional features such as the ratio of box areas, intersection over union (IoU) and directional encodings that characterise the distance between box centers. Note that the directional encodings are normalised by the dimension of the first (human) bounding box instead of that of the image. In addition, function $f(\cdot)$ ensures the componentwise positivity of the feature vector. Finally, denote a multi-layer perceptron as MLP, the complete pairwise positional encoding is computed as below

$$\mathbf{y} = \text{MLP}(\mathbf{u} \oplus \mathbf{p} \oplus \log(\mathbf{u} \oplus \mathbf{p} + \epsilon)), \tag{5}$$

where ϵ is a small constant added to the vector to avoid taking the logarithm of zero.

B. Numerical stability in the loss function

For the sake of numerical stability, loss function for logits is often preferred to that for normalised scores. In our case, due to the fact that the final interaction score is the product of multiple factors, we cannot directly use the loss function for logits. Therefore, we first need to recover the scale prior to normalisation. Denote the normalised object confidence score and action logit as $\hat{y}_1 \in [0, 1]$ and $\hat{y}_2 \in \mathbb{R}$ respectively, the final score is computed as $\hat{y} = \hat{y}_1 \cdot \sigma(\hat{y}_2)$, where σ denotes the sigmoid function. We can then retrieve the corresponding logit $\tilde{y}$ as below

$$\tilde{y} = \sigma^{-1}(\hat{y}), \tag{6}$$

$$= \log \left(\frac{\hat{y}_1}{1 + \exp(-\hat{y}_2) - \hat{y}_1} + \epsilon \right), \tag{7}$$

where ϵ is a small constant added to the term to avoid taking the logarithm of zero.

C. Multi-branch fusion

The multi-branch fusion (MBF) module [5] employs multiple homogeneous branches, wherein each branch maps the two input features into a subspace of reduced dimension and performs fusion (elementwise product by default). The resultant feature is then mapped back to the original size. Afterwards, elementwise sum is used to aggregate results across all branches. The reduced representation size in a branch is intentionally configured in a way that makes the total number of parameters independent of the number of branches. For brevity of exposition, let us assume the number of branches is 1, for two input vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, the output vector $\mathbf{z} \in \mathbb{R}^n$ is computed as

$$\mathbf{z} = W_3^T \phi \left((W_1^T x + \mathbf{b}_1) \otimes (W_2^T y + \mathbf{b}_2) \right) + \mathbf{b}_3, \tag{8}$$

where $W_1, W_2, W_3 \in \mathbb{R}^{n \times n}$ and $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3 \in \mathbb{R}^n$ are parameters of linear layers, ϕ refers to the rectified linear unit (ReLU), and $\otimes$ denotes elementwise product. The implementation for this module in PyTorch is shown in Listing 1.

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class MultiBranchFusion(nn.Module):
    """
    Parameters:
    -----------
    appearance_size: int
        Size of the appearance features
    spatial_size: int
        Size of the spatial features
    hidden_state_size: int
        Size of the intermediate representations
    cardinality: int
        The number of homogeneous branches
    """
    def __init__(self,
        appearance_size: int = 256, spatial_size: int = 256,
        hidden_state_size: int = 256, cardinality: int = 8
    ) -> None:
        super().__init__()
        self.cardinality = cardinality
        sub_repr_size = int(hidden_state_size / cardinality)
        assert sub_repr_size * cardinality == hidden_state_size, \
            "The given representation size should be divisible by cardinality"

        self.fc_1 = nn.ModuleList([
            nn.Linear(appearance_size, sub_repr_size) for _ in range(cardinality)])
        self.fc_2 = nn.ModuleList([
            nn.Linear(spatial_size, sub_repr_size) for _ in range(cardinality)])
        self.fc_3 = nn.ModuleList([
            nn.Linear(sub_repr_size, hidden_state_size) for _ in range(cardinality)])

    def forward(self, appearance: torch.Tensor, spatial: torch.Tensor) -> torch.Tensor:
        return torch.stack([
            fc_3(F.relu(fc_1(appearance) * fc_2(spatial)))
            for fc_1, fc_2, fc_3 in zip(self.fc_1, self.fc_2, self.fc_3)
        ]).sum(dim=0)
```

Listing 1. PyTorch implementation of the multi-branch fusion module.

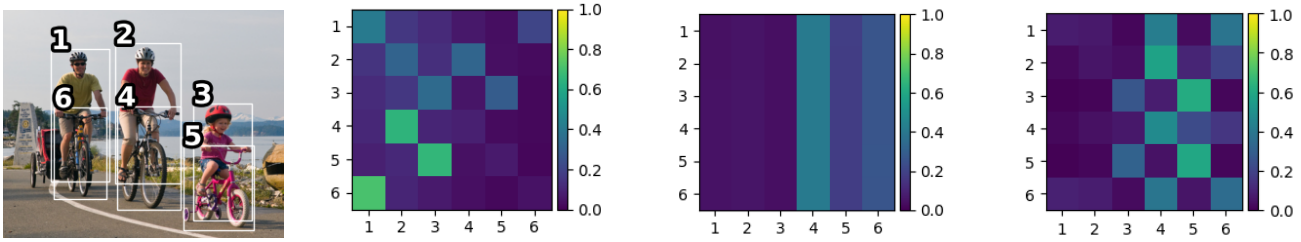
Table 1. Performance comparison amongst different variants of the cooperative layer on HICO-DET [1] test set under default setting. All variants below use ResNet50 [3] as the backbone CNN. The acronym M.E. stands for modified encoder layer.

Variant	Full	Rare	Non-rare
Vanilla	31.15 $\pm$ .03	25.70	32.77
Vanilla w/ add. pos. enc.	31.14	25.59	32.80
M.E. w/o pairwise terms	30.93	24.53	32.84
M.E. w/ FFN	**31.33** $\pm$ .04	**26.02**	**32.91**

D. Modified transformer encoder layer

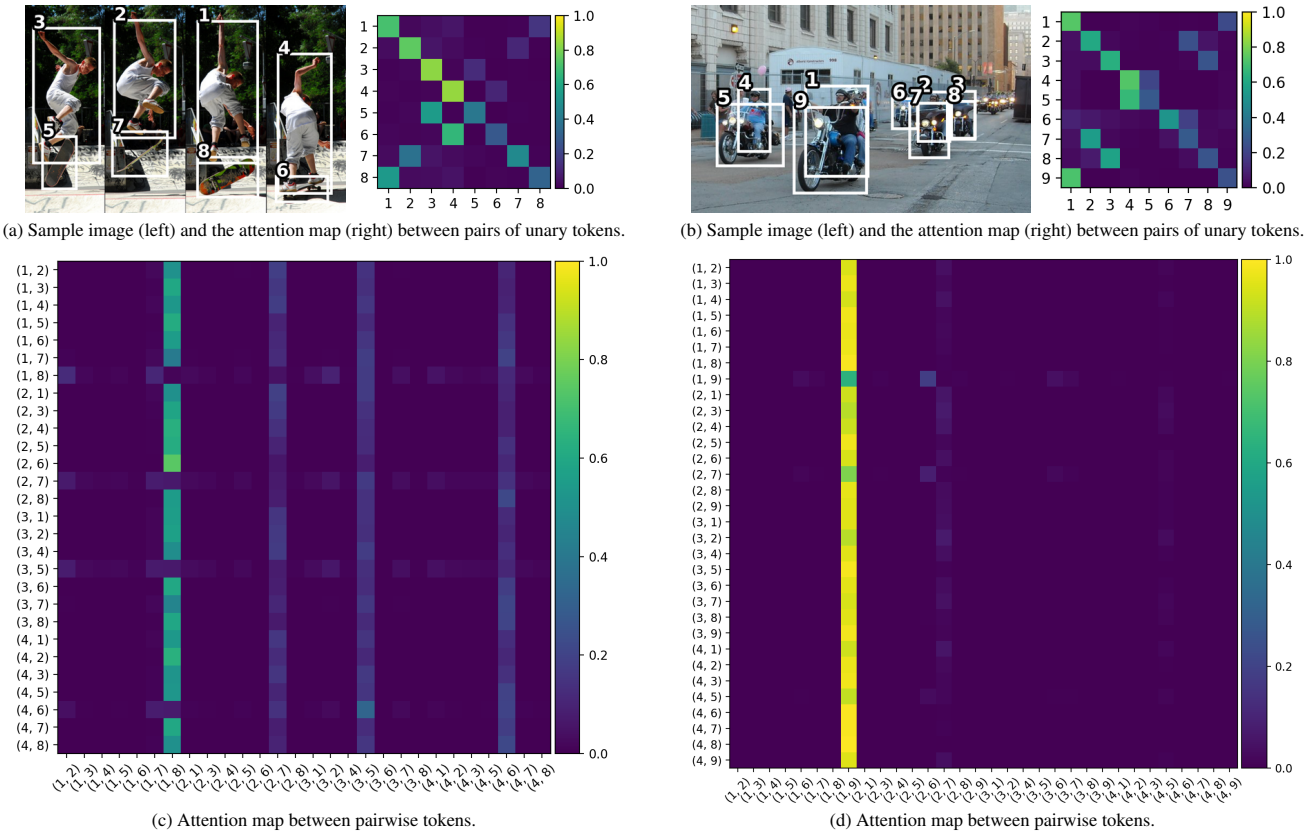
In this section, we compare the performance and interpretability of the modified transformer encoder layer to alternative formulations. To this end, we test multiple variants of the cooperative layer. In particular, we added a feedforward network (FFN) [4] to our modified encoder to better align with the standard transformer architecture. As shown in Tab. 1, using a vanilla transformer encoder results in a small decrease (0.2 mAP) in performance. This gap persists after applying additive positional encodings learned from the unary box terms shown in Eq. (1). We then demonstrate the importance of the pairwise terms in Eq. (2) by removing them from the positional encodings, which resulted in a 0.4 mAP decrease. Together, these results indicate that the pairwise terms provide useful information for the cooperative layer and a consistent mAP performance boost.

In addition, we show the attention maps in different variants of the cooperative layer in Fig. 1. Notably, our modified encoder (Fig. 1b) accurately infers the correspondence between instances, where the interactive humans and objects attend to each other. This suggests that the pairwise positional encoding instills an inductive bias in the modified encoder that allows it to identify interactive and non-interaction pairs, and preferentially share information be-



(a) *sample image* (b) *attn. map from M.E.* (c) *attn. map from M.E. w/o pairwise terms* (d) *attn. map from vanilla encoder*

Figure 1. An image with detected instances (a) and attention maps in the cooperative layer with different implementations, including the vanilla encoder (d), modified encoder w/o pairwise terms (c) and the modified encoder (b).



(a) Sample image (left) and the attention map (right) between pairs of unary tokens.

(b) Sample image (left) and the attention map (right) between pairs of unary tokens.

(c) Attention map between pairwise tokens.

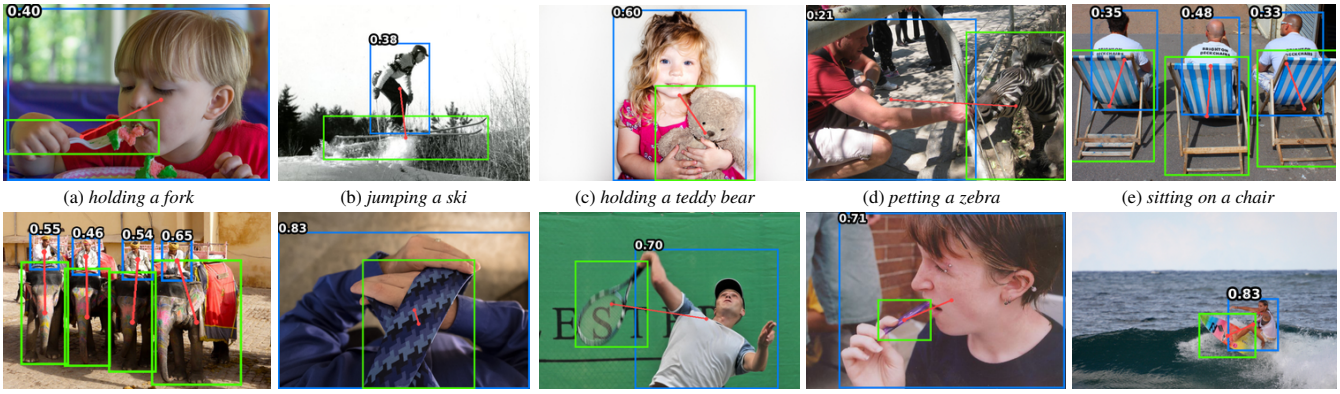
(d) Attention map between pairwise tokens.

Figure 2. Our model exhibits the same behavior on sample images with numerous human and object instances. Specifically, the attention map for unary tokens shows high symmetry, where potentially interactive instances attend to each other. And the pairwise attention map indicates that non-interactive pairs attend to the most dominant pairs to be suppressed.

tween the interactive ones. Furthermore, we show that, without the pairwise terms, as shown in Fig. 1c, the attention map becomes uniform along one dimension. Similarly, the vanilla encoder does not make use of the pairwise spatial information either. This results in the attention maps being much less interpretable as shown in Fig. 1d. In particular, there is no strong mutual attention between interactive instances, but more attention between non-interactive ones. The complete implementation of the modified encoder in PyTorch is shown in Listing 2.

E. Additional qualitative results

We show more visualisations of attention maps in Fig. 2. We intentionally avoided images with very few human and object instances and instead selected those with more complicated scenes for the purpose of demonstration. As shown by the attention maps, our model behaves consistently across different interaction types. More qualitative results for detected HOIs can be found in Fig. 3 and Fig. 4. We also show some false positives of our model in Fig. 5.



(a) *holding a fork* (b) *jumping a ski* (c) *holding a teddy bear* (d) *petting a zebra* (e) *sitting on a chair*

(f) *riding an elephant* (g) *wearing a tie* (h) *swinging a tennis racket* (i) *holding a toothbrush* (j) *standing on a surfboard*

Figure 3. Additional qualitative results for detected human–object pairs on HICO-DET [1] test set.

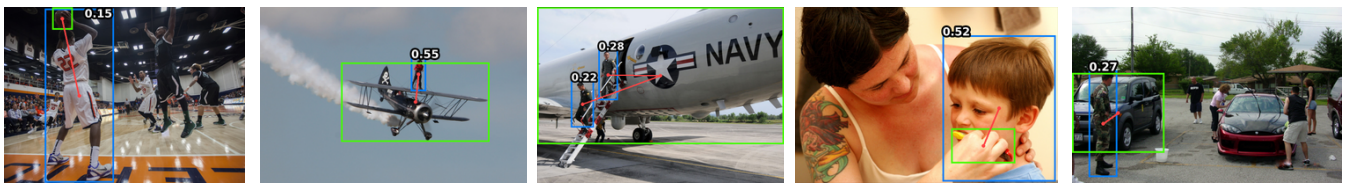
(a) *carrying a backpack* (b) *cutting with a knife* (c) *drinking from a bottle* (d) *eating with a fork* (e) *holding a surfboard*

(f) *eating a hotdog* (g) *holding a knife* (h) *jumping a skateboard* (i) *kicking a sports ball* (j) *eating a pizza*

(k) *holding a tennis racket* (l) *reading a book* (m) *laying on a bed* (n) *sitting on a couch* (o) *riding a horse*

(p) *talking on a phone* (q) *working on a computer* (r) *riding a boat* (s) *throwing a frisbee* (t) *riding a motorcycle*

Figure 4. Additional qualitative results for detected human–object pairs on V-COCO [2] test set.



(a) *blocking a sports ball* (b) *flying an airplane* (c) *boarding an airplane* (d) *brushing with a toothbrush* (e) *washing a car*

Figure 5. False positives on HICO-DET [1] test set.

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from typing import Tuple

class ModifiedEncoderLayer(nn.Module):
    def __init__(self, hidden_size: int = 256, repr_size: int = 256, num_heads: int = 8) -> None:
        super().__init__()
        if repr_size % num_heads != 0:
            raise ValueError(
                f"The given representation size {repr_size} "
                f"should be divisible by the number of attention heads {num_heads}.")
        self.sub_repr_size = int(repr_size / num_heads)
        self.hidden_size = hidden_size
        self.repr_size = repr_size
        self.num_heads = num_heads

        self.unary = nn.Linear(hidden_size, repr_size)
        self.pairwise = nn.Linear(repr_size, repr_size)
        self.attn = nn.ModuleList([nn.Linear(3 * self.sub_repr_size, 1) for _ in range(num_heads)])
        self.message = nn.ModuleList([
            nn.Linear(self.sub_repr_size, self.sub_repr_size) for _ in range(num_heads)])
        self.aggregate = nn.Linear(repr_size, hidden_size)
        self.dropout = nn.Dropout(0.1)
        self.norm = nn.LayerNorm(hidden_size)

    def reshape(self, x: torch.Tensor) -> torch.Tensor:
        new_x_shape = x.size()[:-1] + (self.num_heads, self.sub_repr_size)
        x = x.view(*new_x_shape)

        if len(new_x_shape) == 3: return x.permute(1, 0, 2)
        elif len(new_x_shape) == 4: return x.permute(2, 0, 1, 3)
        else: raise ValueError("Incorrect tensor shape")

    def forward(self, x: torch.Tensor, y: torch.Tensor) -> Tuple[torch.Tensor, torch.Tensor]:
        """
        Parameters:
        -----------
        x: torch.Tensor
            Unary tokens of size (N, K)
        y: torch.Tensor
            Pairwise positional encodings of size (N, N, K)
        """
        device = x.device; n = len(x)
        u = F.relu(self.unary(x))
        p = F.relu(self.pairwise(y))
        # Unary features (H, N, K/H)
        u_r = self.reshape(u)
        # Pairwise features (H, N, N, K/H)
        p_r = self.reshape(p)
        # Generate indices for pairwise concatenation
        i, j = torch.meshgrid(torch.arange(n, device=device), torch.arange(n, device=device))
        # Features used to compute attention (H, N, N, 3K/H)
        attn_features = torch.cat([u_r[:, i], u_r[:, j], p_r], dim=-1)
        # Attention weights (H,) (N, N, 1)
        weights = [F.softmax(l(f), dim=0) for f, l in zip(attn_features, self.attn)]
        # Repeated unary feaures along the third dimension (H, N, N, K/H)
        u_r_repeat = u_r.unsqueeze(dim=2).repeat(1, 1, n, 1)
        messages = [l(f_1 * f_2) for f_1, f_2, l in zip(u_r_repeat, p_r, self.message)]
        aggregated_messages = self.aggregate(F.relu(
            torch.cat([(w * m).sum(dim=0) for w, m in zip(weights, messages)], dim=-1)
        ))
        aggregated_messages = self.dropout(aggregated_messages)
        x = self.norm(x + aggregated_messages)
        return x, weights
```

Listing 2. PyTorch implementation of the modified transformer encoder layer.

References

[1] Yu-Wei Chao, Yunfan Liu, Xieyang Liu, Huayi Zeng, and Jia Deng. Learning to detect human-object interactions. In *Proceedings of the IEEE Winter Conference on Applications of Computer Vision*, 2018. 2, 4

[2] Saurabh Gupta and Jitendra Malik. Visual semantic role labeling. *arXiv preprint arXiv:1505.04474*, 2015. 4

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 770–778, 2016. 2

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Adv. Neural Inform. Process. Syst.*, volume 30, 2017. 2

[5] Frederic Z. Zhang, Dylan Campbell, and Stephen Gould. Spatially conditioned graphs for detecting human-object interactions. In *Int. Conf. Comput. Vis.*, pages 13319–13327, October 2021. 1