Rethinking and Improving Relative Position Encoding for Vision Transformer

Kan Wu[1,2,*], Houwen Peng[2,*,†], Minghao Chen[2], Jianlong Fu[2], Hongyang Chao[1]

[1] Sun Yat-sen University   [2] Microsoft Research Asia

Abstract

Relative position encoding (RPE) is important for transformer to capture sequence ordering of input tokens. General efficacy has been proven in natural language processing. However, in computer vision, its efficacy is not well studied and even remains controversial, e.g., whether relative position encoding can work equally well as absolute position? In order to clarify this, we first review existing relative position encoding methods and analyze their pros and cons when applied in vision transformers. We then propose new relative position encoding methods dedicated to 2D images, called image RPE (iRPE). Our methods consider directional relative distance modeling as well as the interactions between queries and relative position embeddings in self-attention mechanism. The proposed iRPE methods are simple and lightweight. They can be easily plugged into transformer blocks. Experiments demonstrate that solely due to the proposed encoding methods, DeiT [22] and DETR [1] obtain up to 1.5% (top-1 Acc) and 1.3% (mAP) stable improvements over their original versions on ImageNet and COCO respectively, without tuning any extra hyperparameters such as learning rate and weight decay. Our ablation and analysis also yield interesting findings, some of which run counter to previous understanding. Code and models are open-sourced at here.

1. Introduction

Transformer recently has drawn great attention in computer vision because of its competitive performance and superior capability in capturing long-range dependencies [1, 6, 22, 25]. The core of transformer is self-attention [23], which is capable of modeling the relationship of tokens in a sequence. Self-attention, however, has an inherent deficiency — it cannot capture the ordering of input tokens. Therefore, incorporating explicit representations of position information is especially important for transformer, since the model is otherwise entirely invariant to sequence ordering, which is undesirable for modeling structured data.

There are mainly two classes of methods to encode positional representations for transformer. One is absolute, while the other is relative. Absolute methods [7, 23] encode the absolute positions of input tokens from 1 to maximum sequence length. That is, each position has an individual encoding vector. The encoding vector is then combined with the input token to expose positional information to the model. On the other hand, relative position methods [18, 3] encode the relative distance between input elements and learn the pairwise relations of tokens. Relative position encoding (RPE) is commonly calculated via a look-up table with learnable parameters interacting with queries and keys in self-attention modules [18]. Such scheme allows the modules to capture very long dependencies between tokens. Relative position encoding has been verified to be effective in natural language processing [5, 3, 26, 16]. However, in computer vision, the efficacy is still unclear. There are few recent works [6, 2, 19] shedding light on it, but obtaining controversial conclusions in vision transformers. For example, Dosovitskiy *et al.* [6] observed that the relative position encoding does not bring any gain comparing to the absolute one (please refer to Tab. 8 in [6]). On the contrary, Srinivas *et al.* [19] found that relative position encoding can induce an apparent gain, being superior to the absolute one (please refer to Tab. 4 in [19]). Moreover, the mostly recent work [2] claims that the relative positional encoding cannot work equally well as the absolute ones (please refer to Tab. 5 in [2]). These works draw different conclusions on the effectiveness of relative position encoding in models, that motivates us to review and rethink the usage of relative positional encoding in vision transformer.

On the other hand, the original relative position encoding is proposed for language modeling, where the input data is 1D word sequences [23, 3, 18]. But for vision tasks, the inputs are usually 2D images or video sequences, where the pixels are highly spatially structured. It is unclear that: whether the naive extension from 1D to 2D is suitable for vision models; whether the directional information is important in vision tasks?

In this paper, we first review existing relative position encoding methods, and then propose new methods dedicated to 2D images. We make the following contributions.

---

*Equal contributions.

- We analyze several key factors in relative position encoding, including the relative direction, the importance of context, the interactions between queries, keys, values and relative position embeddings, and computational cost. The analysis presents a comprehensive understanding of relative position encoding, and provides empirical guidelines for new method design.

- We introduce an efficient implementation of relative encoding, which reduces the computational cost from the original $\mathcal{O}(n^2 d)$ to $\mathcal{O}(nkd)$, where $k \ll n$. Such implementation is suitable for high-resolution input images, such as object detection and semantic segmentation, where the token number might be very large.

- We propose four new relative position encoding methods, called image RPE (iRPE), dedicated to vision transformers, considering both efficiency and generalizability. The methods are simple and can be easily plugged into self-attention layers. Experiments show that, without adjusting any hyperparameters and settings, the proposed methods can improve DeiT-S [22] and DETR-ResNet50 [1] by 1.5% (top-1 Acc) and 1.3% (mAP) over their original models on ImageNet [4] and COCO [12], respectively.

- We answer previous controversial questions. We empirically demonstrate that relative position encoding can replace the absolute encoding for image classification task. Meanwhile, the absolute encoding is necessary for object detection, where the pixel position is important for object localization.

2. Background

2.1. Self-Attention

Self-attention plays a fundamental role in transformer. It maps a query and a set of key-value pairs to an output. More specifically, for an input sequence, *e.g.*, the embeddings of words or image patches, $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ of n elements where $\mathbf{x}_i \in R^{d_x}$, self-attention module computes an output sequence $\mathbf{z} = (\mathbf{z}_1, \ldots, \mathbf{z}_n)$ of the same length where $\mathbf{z}_i \in \mathbb{R}^{d_z}$. Each output element, $\mathbf{z}_i$, is computed as weighted sum of a linearly transformed input elements:

$$\mathbf{z}_i = \sum_{j=1}^{n} \alpha_{ij} (\mathbf{x}_j \mathbf{W}^V). \tag{1}$$

Each weight coefficient, α_{ij}, is computed using a softmax:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{n} \exp(e_{ik})}, \tag{2}$$

where e_{ij} is calculated using a scaled dot-product attention:

$$e_{ij} = \frac{(\mathbf{x}_i \mathbf{W}^Q)(\mathbf{x}_j \mathbf{W}^K)^T}{\sqrt{d_z}}. \tag{3}$$

Here, the projections $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V \in \mathbb{R}^{d_x \times d_z}$ are parameter matrices, which are unique per layer.

Rather than computing the self-attention once, Multi-head self-attention (MHSA) [23] runs the self-attention multiple times in parallel, *i.e.*, employing h attention heads. The attention head outputs are simply concatenated and linearly transformed into the expected dimensions.

2.2. Position Encoding

Absolute Position Encoding. Since transformer contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, we need to inject some information about the position of the tokens. The original self-attention considers the *absolute position* [23], and add the absolute positional encodings $\mathbf{p} = (\mathbf{p}_1, \ldots, \mathbf{p}_n)$ to the input token embedding $\mathbf{x}$ as

$$\mathbf{x}_i = \mathbf{x}_i + \mathbf{p}_i, \tag{4}$$

where the positional encoding $\mathbf{p}_i, \mathbf{x}_i \in \mathcal{R}_x^d$. There are several choices of absolute positional encodings, such as the fixed encodings by sine and cosine functions with different frequencies and the learnable encodings through training parameters [7, 23].

Relative Position Encoding. Besides the absolute position of each input element, recent works also consider the pairwise relationships between elements, *i.e.*, *relative position* [18]. Relative relation is presumably important for tasks where the relative ordering or distance of the elements matters. This type of methods encode the relative position between the input elements $\mathbf{x}_i$ and $\mathbf{x}_j$ into vectors $\mathbf{p}_{ij}^V, \mathbf{p}_{ij}^Q, \mathbf{p}_{ij}^K \in \mathbb{R}^{d_z}$, where $d_z = d_x$. The encoding vectors are embedded into the self-attention module, which re-formulates Eq. (1) and Eq. (3) as

$$\mathbf{z}_i = \sum_{j=1}^{n} \alpha_{ij} (\mathbf{x}_j \mathbf{W}^V + \mathbf{p}_{ij}^V), \tag{5}$$

$$e_{ij} = \frac{(\mathbf{x}_i \mathbf{W}^Q + \mathbf{p}_{ij}^Q)(\mathbf{x}_j \mathbf{W}^K + \mathbf{p}_{ij}^K)^T}{\sqrt{d_z}}. \tag{6}$$

In this fashion, the pairwise positional relation is learned during transformer training. Such relative position encoding can be either shared across attention heads or not.

3. Method

In this section, we first review previous relative position encoding methods and analyze their differences. Then, we propose four new methods dedicated to vision transformer, and an efficient implementation.
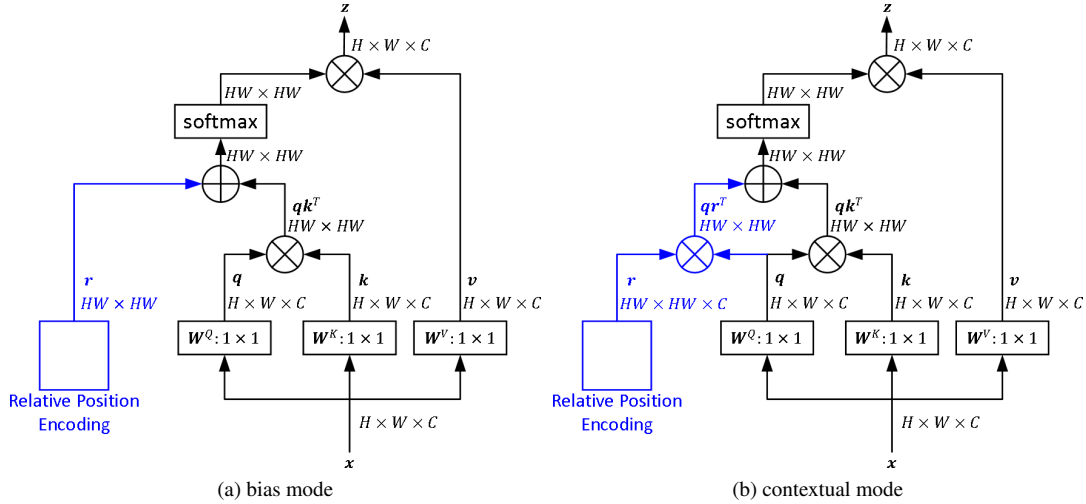
Figure 1: Illustration of self-attention modules with 2D relative position encoding on keys. The blue parts are newly added.

3.1. Previous Relative Position Encoding Methods

Shaw's RPE. Shaw *et al.* [18] propose a relative position encoding for self-attention. The input tokens are modeled as a directed and fully-connected graph. Each edge between two arbitrary positions i and j is presented by a learnable vector $\mathbf{p}_{ij} \in \mathbb{R}^{d_z}$, namely relative position encoding. Besides, the authors deemed that precise relative position information is not useful beyond a certain distance, so introduced a clip function to reduce the number of parameters. The encoding is formulated as

$$\mathbf{z}_i = \sum_{j=1}^{n} \alpha_{ij}(\mathbf{x}_j \mathbf{W}^V + \mathbf{p}_{clip(i-j,k)}^V), \quad (7)$$

$$e_{ij} = \frac{(\mathbf{x}_i \mathbf{W}^Q)(\mathbf{x}_j \mathbf{W}^K + \mathbf{p}_{clip(i-j,k)}^K)^T}{\sqrt{d_z}}, \quad (8)$$

$$clip(x,k) = max(-k, min(k,x)), \quad (9)$$

where $\mathbf{p}^V$ and $\mathbf{p}^K$ are the trainable weights of relative position encoding on values and keys, respectively. $\mathbf{p}^V = (\mathbf{p}_{-k}^V, ..., \mathbf{p}_k^V)$ and $\mathbf{p}^K = (\mathbf{p}_{-k}^K, ..., \mathbf{p}_k^K)$ where $\mathbf{p}_i^V, \mathbf{p}_i^K \in \mathbb{R}^{d_z}$. The scalar k is the maximum relative distance.

RPE in Transformer-XL. Dai *et al.* [3] introduce additional bias terms for queries, and uses the sinusoid formulation for relative position encoding, which is formulated as

$$e_{ij} = \frac{(\mathbf{x}_i \mathbf{W}^Q + \mathbf{u})(\mathbf{x}_j \mathbf{W}^K)^T + (\mathbf{x}_i \mathbf{W}^Q + \mathbf{v})(\mathbf{s}_{i-j}\mathbf{W}^R)^T}{\sqrt{d_z}}, \quad (10)$$

where $\mathbf{u}, \mathbf{v} \in \mathbb{R}^{d_z}$ are two learnable vectors.

The sinusoid encoding vector $\mathbf{s}$ provides the prior of relative position [23]. $\mathbf{W}^R \in \mathbb{R}^{d_z \times d_z}$ is a trainable matrix, projecting $\mathbf{s}_{i-j}$ into a location-based key vector.

Huang's RPE. Huang *et al.* [11] propose a new method considering the interactions of queries, keys and relative positions simultaneously. The equation is given as follows

$$e_{ij} = \frac{(\mathbf{x}_i \mathbf{W}^Q + \mathbf{p}_{ij})(\mathbf{x}_j \mathbf{W}^K + \mathbf{p}_{ij})^T - \mathbf{p}_{ij}\mathbf{p}_{ij}^T}{\sqrt{d_z}}, \quad (11)$$

where $\mathbf{p}_{ij} \in \mathbb{R}^{d_z}$ is the relative position encoding shared by queries and keys.

RPE in SASA. The above three methods are all designed for 1D word sequence in language modeling. Ramachandran *et al.* [17] propose an encoding method for 2D images. The idea is simple. It divides the 2D relative encoding into horizontal and vertical directions, such that each direction can by modeled by a 1D encoding. The method formulation is given as follows

$$e_{ij} = \frac{(\mathbf{x}_i \mathbf{W}^Q)(\mathbf{x}_j \mathbf{W}^K + concat(\mathbf{p}_{\delta\tilde{x}}^K, \mathbf{p}_{\delta\tilde{y}}^K))^T}{\sqrt{d_z}}, \quad (12)$$

where $\delta\tilde{x} = \tilde{x}_i - \tilde{x}_j$ and $\delta\tilde{y} = \tilde{y}_i - \tilde{y}_j$ denote the relative position offsets on x-axis and y-axis of the image coordinate respectively, $\mathbf{p}_{\delta\tilde{x}}^K$ and $\mathbf{p}_{\delta\tilde{y}}^K$ are learnable vectors with length $\frac{1}{2}d_z$, the *concat* operation concatenates the two encodings to form a final relative encoding with length of d_z. In other words, the same offsets on x-axis or y-axis share the same relative position encoding, so this method is able to reduce the number of learnable parameters and computational cost. However, the encoding is only applied on keys. In our experiments, we observe that the RPE imposed on keys, queries and values simultaneously is the most effective one, as presented in Tab. 4 and Tab. 5.

RPE in Axial-Deeplab. Wang *et al.* [24] introduce a position-sensitive method that adds qkv-dependent positional bias into self-attention. The position sensitivity is applied on axial attention that propagates information along

height-axis and width-axis sequentially. However, when the relative distance is larger than a threshold, the encoding is set to zero. We observe that long-range relative position information is useful, as analysed in Tab. 6. The position-sensitivity might be competitive when imposed on the standard self-attention. If equipped with the proposed piecewise function, it can be further improved and become more efficient for modeling long-range dependencies.

3.2. Proposed Relative Position Encoding Methods

We design our image RPE (iRPE) methods to analyze several factors which are not well studied in prior works (see the analysis in Sec. 4.2). First, to study whether the encoding can be independent of the input embeddings, we introduce two relative position modes: bias and contextual. We present a piecewise function to map relative positions to encodings, being different from the conventional clip function. After that, to study the importance of directivity, we design two undirected and two directed methods. Finally we provide an efficient implementation for our methods.

Bias Mode and Contextual Mode. Previous relative position encoding methods all depend on input embeddings. It brings a question, *i.e.*, whether the encoding can be independent of the input? We introduce bias mode and contextual mode of relative position encoding to study the question. The former one is independent of input embeddings, while the latter one considers the interaction with queries, keys or values. More specifically, we introduce a unified formulation as

$$e_{ij} = \frac{(\mathbf{x}_i \mathbf{W}^Q)(\mathbf{x}_j \mathbf{W}^K)^T + b_{ij}}{\sqrt{d_z}}, \tag{13}$$

where $b_{ij} \in \mathbb{R}$ is the 2D relative position encoding, defining the bias or contextual mode. For bias mode,

$$b_{ij} = r_{ij}, \tag{14}$$

where $r_{ij} \in \mathbb{R}$ is a learnable scalar and represents the relative position weight between the position i and j. For contextual mode,

$$b_{ij} = (\mathbf{x}_i \mathbf{W}^Q)\mathbf{r}_{ij}{}^T, \tag{15}$$

where $\mathbf{r}_{ij} \in \mathbb{R}^{d_z}$ is a trainable vector, interacted with the query embedding. There are multiple variants for b_{ij} in contextual mode. For example, the relative position encoding operated on both queries and keys can be presented as

$$b_{ij} = (\mathbf{x}_i \mathbf{W}^Q)(\mathbf{r}_{ij}^K)^T + (\mathbf{x}_j \mathbf{W}^K)(\mathbf{r}_{ij}^Q)^T, \tag{16}$$

where $\mathbf{r}_{ij}^K, \mathbf{r}_{ij}^Q \in \mathbb{R}^{d_z}$ are both learnable vectors. Besides, contextual mode can also be applied on value embeddings,

$$\mathbf{z}_i = \sum_{j=1}^{n} \alpha_{ij}(\mathbf{x}_j \mathbf{W}^V + \mathbf{r}_{ij}^V), \tag{17}$$

where $\mathbf{r}_{ij}^V \in \mathbb{R}^{d_z}$. The relative position weights $\mathbf{r}_{ij}^Q$, $\mathbf{r}_{ij}^K$ and $\mathbf{r}_{ij}^V$ can be constructed in the same way. For a unified representation, we use $\mathbf{r}_{ij}$ to denote them in bias mode and contextual mode in the following discussion. Fig. 1 shows the illustration of self-attention modules with 2D relative position encoding on keys in the propsoed two modes.

A Piecewise Index Function. Before describing the 2D relative position weight $\mathbf{r}_{ij}$, we first introduce a many-to-one function, mapping a relative distance into an integer in finite set, then $\mathbf{r}_{ij}$ can be indexed by the integer and share encodings among different relation positions. Such index function can largely reduce computation costs and the number of parameters for long sequence (*e.g.*, high resolution images). Although the clip function $h(x) = max(-\beta, min(\beta, x))$ used in [18] also reduces the cost, the positions whose relative distance is larger than β are assigned to the same encoding. This method inevitably drops out the contextual information of long-range relative positions. Inspired by [16], we introduce a piecewise function $g(x) : \mathbb{R} \to \{y \in \mathbb{Z} | -\beta \le y \le \beta\}$ for indexing relative distances to corresponding encodings. The function is based on a hypothesis that the closer neighbors are more important than the further ones, and distributes the attention by the relative distance. It is presented as

$$g(x) = \begin{cases} [x], & |x| \le \alpha \\ sign(x) \times min(\beta, [\alpha + \frac{\ln(|x|/\alpha)}{\ln(\gamma/\alpha)}(\beta - \alpha)]), & |x| > \alpha \end{cases} \tag{18}$$

where $[\cdot]$ is a round operation, $sign()$ determines the sign of a number, *i.e.*, returning 1 for positive input, -1 for negative, and 0 for otherwise. α determines the piecewise point, β controls the output in the range of $[-\beta, \beta]$, and γ adjusts the curvature of the logarithmic part.

We compare the piecewise function $g(x)$ with the clip function $h(x) = min(-\beta, max(\beta, x))$, *i.e.* Eq. (9). In Fig. 2, the clip function $h(x)$ distributes uniform attention and leaves out long distance positions, but the piecewise function $g(x)$ distributes different levels of attention by relative distance. We suppose that the potential information in long-range position should be preserved, especially for high resolution images or the tasks requiring long-range feature dependencies, so $g(x)$ is selected to construct our mapping method for $\mathbf{r}_{ij}$.

2D Relative Position Calculation. In order to calculate relative position on 2D image plane and define the relative weight $\mathbf{r}_{ij}$, we propose two undirected mapping methods, namely Euclidean and Quantization, as well as two directed mapping methods, namely Cross and Product.

Euclidean method. On image plane, the relative position $(\tilde{x}_i - \tilde{x}_j, \tilde{y}_i - \tilde{y}_j)$ is a 2D coordinate. We compute Euclidean distance between two positions, and maps the distance into the corresponding encoding. The method is undirected and formulated as
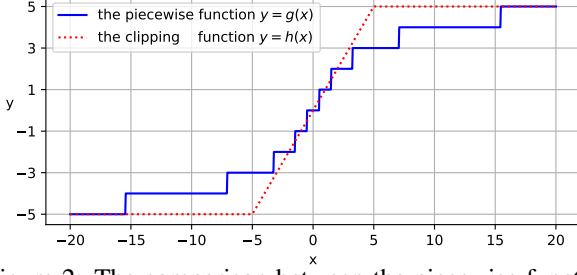
$$\mathbf{r}_{ij} = \mathbf{p}_{I(i,j)}, \tag{19}$$

4

Figure 2: The comparison between the piecewise function $g(x)$ and the clip function $h(x)$.

$$I(i,j) = g(\sqrt{(\tilde{x}_i - \tilde{x}_j)^2 + (\tilde{y}_i - \tilde{y}_j)^2}), \qquad (20)$$

where $\mathbf{p}_{I(i,j)}$ is either a learnable scalar in bias mode or a vector in contextual mode. We regard $\mathbf{p}_{I(i,j)}$ as a bucket, which stores the relative position weight. The number of buckets is $2\beta + 1$, as defined in Eq. (18).

Quantization method. In the above Euclidean method, the closer two neighbors with different relative distances may be mapped into the same index, e.g. the 2D relative positions $(1, 0)$ and $(1, 1)$ are both mapped into the index 1. We suppose that the close neighbors should be separated. Therefore, we quantize Euclidean distance, i.e., different real number is mapped into different integer. We revise $I(i,j)$ in Eq. (19) as

$$I(i,j) = g(quant(\sqrt{(\tilde{x}_i - \tilde{x}_j)^2 + (\tilde{y}_i - \tilde{y}_j)^2})). \quad (21)$$

The operation $quant(\cdot)$ maps a set of real numbers $\{0, 1, 1.41, 2, 2.24, ...\}$ into a set of integers $\{0, 1, 2, 3, 4, ...\}$. This method is also undirected.

Cross method. Positional direction of pixels is also important for images, we thereby propose directed mapping methods. This method is called Cross method, which computes encoding on horizontal and vertical directions separately, then summarizes them. The method is given as follows,

$$\mathbf{r}_{ij} = \mathbf{p}^{\tilde{x}}_{I^{\tilde{x}}(i,j)} + \mathbf{p}^{\tilde{y}}_{I^{\tilde{y}}(i,j)}, \qquad (22)$$

$$I^{\tilde{x}}(i,j) = g(\tilde{x}_i - \tilde{x}_j), \qquad (23)$$

$$I^{\tilde{y}}(i,j) = g(\tilde{y}_i - \tilde{y}_j), \qquad (24)$$

where $\mathbf{p}^{\tilde{x}}_{I(i,j)}$ and $\mathbf{p}^{\tilde{y}}_{I(i,j)}$ are both learnable scalars in bias mode, or a learnable vectors in contextual mode. Similar to the encoding in SASA [17], the same offsets on x-axis or y-axis share the same encoding, but the main difference is that we use a piecewise function to distribute attention by relative distance. The number of buckets is $2 \times (2\beta + 1)$.

Product method. The Cross method encodes different relative positions into the same embedding if the distance on one direction is identical, either horizontal or vertical.

Besides, the addition operation in Eq. (22) brings extra computational cost. To improve efficiency and involve more directional information, we design Product method which is formulated as below

$$\mathbf{r}_{ij} = \mathbf{p}_{I^{\tilde{x}}(i,j), I^{\tilde{y}}(i,j)}. \qquad (25)$$

The right side of the equation is a trainable scalar in bias mode, or a trainable vector in contextual mode. $I^{\tilde{x}}(i,j)$ and $I^{\tilde{y}}(i,j)$ are defined in Eq. (23) and Eq. (24), and the combination of them is a 2D index for $\mathbf{p}$. The number of buckets is $(2\beta + 1)^2$.

An Efficient Implementation. For the above proposed methods in contextual mode, there is a common term $(\mathbf{x}_i \mathbf{W})\mathbf{p}_{I(i,j)}^T$ when putting Eq. (19), Eq. (22) or Eq. (25) into Eq. (15). Let y_{ij} denote the common term as follows,

$$y_{ij} = (\mathbf{x}_i \mathbf{W})\mathbf{p}_{I(i,j)}^T. \qquad (26)$$

It takes time complexity $\mathcal{O}(n^2 d)$ to compute all y_{ij}, where n and d are the length of the input sequence and the number of feature channels, respectively. Due to the many-to-one property of $I(i,j)$, the set size k of $I(i,j)$ is usually less than n in vision transformer. Therefore, we provide an efficient implementation as follows,

$$z_{i,t} = (\mathbf{x}_i \mathbf{W})\mathbf{p}_t^T, t \in \{I(i,j)|i,j \in [0,n)\}, \qquad (27)$$

$$y_{ij} = z_{i,I(i,j)}. \qquad (28)$$

It first takes time complexity $\mathcal{O}(nkd)$ to pre-compute all $z_{i,t}$ by Eq. (27), then assigns $z_{i,t}$ to all y_{ij} by the mapping $t = I(i,j)$ by Eq. (28). The assignment operation takes time complexity $\mathcal{O}(n^2)$, whose cost is much smaller than that of the pre-computation procedure. Thus, the computational cost of relative position encoding reduces from the original $\mathcal{O}(n^2 d)$ to $\mathcal{O}(nkd)$.

4. Experiments

In this section, we first provide some analysis by comparing different position embeddings, followed by experiments on the effects of key factors in relative position encoding. Then, we compare the proposed methods with the state-of-the-art methods on image classification and object detection tasks. Finally, we visualize the relative position encoding and explain why it works.

4.1. Implementation Details

We choose the recent vision transformer model DeiT [22] as the baseline for most experiments. The relative position encoding is added into all self-attention layers. If not specified, the relative position encoding is only added on keys. We set $\alpha{:}\beta{:}\gamma$ = 1:2:8 for the piecewise function $g(x)$, and adjust the number of buckets by changing β. An

5

Method based on DeiT-S [22]	Is Directed	Mode	Top-1 Acc(%)	Δ Acc(%)
Original [22]	-	-	79.9	–
Euclidean	×	bias	80.1	+0.2
		contextual	80.4	+0.5
Quantization	×	bias	80.3	+0.4
		contextual	80.5	+0.6
Cross	✓	bias	80.5	+0.6
		contextual	80.8	+0.9
Product	✓	bias	80.5	+0.6
		contextual	80.9	+1.0

Table 1: Ablation of our relative position encoding methods on ImageNet [4]. The original model is DeiT-S [22], which only uses absolute position encoding. We equip the model with the proposed four relative encoding methods, *i.e.*, Eq. (19), Eq. (21), Eq. (22) and Eq. (25). We select the best numbers of buckets for each method, which are 20, 51, 56 and 50 respectively.

Mode	Shared	#Param. (M)	MACs (M)	Top-1 Acc(%)
Bias	×	22.05	4613	80.54 ± 0.06
	✓	22.05	4613	80.05 ± 0.04
Contextual	×	22.28	4659	80.99 ± 0.16
	✓	22.09	4659	80.89 ± 0.04

Table 2: Ablation of shared and unshared relative position encoding across attention heads. The experiments are conducted over DeiT-S [22] on ImageNet [4] with 50 buckets. The models are trained and evaluated by three times.

Function	Mode	Top-1 Acc(%)	Top-5 Acc(%)
clip	bias	80.1	94.9
	contextual	80.9	95.5
piecewise	bias	80.0	95.0
	contextual	80.9	95.5

Table 3: Ablation for clip function and piecewise function. The experiments are conducted over DeiT-S [22] model with product shared-head relative position encoding on ImageNet [4]. The number of buckets is 50.

extra bucket is used to store the relative position encodings of the classification token.

For fair comparison, we adopt the same training settings as DeiT [22]: AdamW [14] optimizer with weight decay 0.05, initial learning rate $1x10^{-3}$ and minimal learning $1x10^{-5}$ with cosine scheduler, 5 epochs warmup, batch size of 1024, 0.1 label smoothing [20], and stochastic depth with survival rate probability of 0.9. For training, the images are split into 14x14 non-overlapping patches. Data augmentation methods [28, 27] are also consistent with DeiT [22]. All models are trained from scratch for 300 epochs with 8 NVIDIA Tesla V100 GPUs.

4.2. Analysis on Relative Position Encoding

Directed v.s. *Undirected.* As shown in Tab. 1, directed methods (Cross and Product), in general, perform better than undirected ones (Euclidean and Quantization) in vision transformer. This phenomenon illustrates that the directivity is important for vision transformers, because image pixels are highly structured and semantically correlative.

Bias v.s. *Contextual.* Tab. 1 shows that the contextual mode achieves superior performance to that of bias mode, regardless of which method uses. The underlying reason might be that contextual mode changes the encoding with the input feature while bias mode keeps static.

Shared v.s. *Unshared.* Self-attention contains multiple heads. The relative position encoding can be either shared or unshared across different heads. We show the effects of these two schemes in bias and contextual modes in Tab. 2, respectively. For bias mode, the accuracy drops significantly when sharing encoding across the heads. By contrast, in contextual mode, the performance gap between two schemes is negligible. Both of them achieve an average top-1 accuracy of 80.9%. We conjecture that different head

needs different relative position encoding (RPE for short) to capture different information. In contextual mode, each head can compute its own RPE by the Eq. (15) while in bias mode the shared RPE forces all heads to pay the same attention on patches. For parameter-saving, we adopt the share scheme in our final methods.

Piecewise v.s. *Clip.* We compare the efficacy of the piecewise function $g(x)$ defined in Eq. (18) and the clip function $h(x)$ defined in Eq. (9) in Tab. 3. There is a very small, even negligible, performance gap between these two functions in image classification task. However, in object detection task, we found that clip function is worse than the piecewise one as illustrated in Tab. 6 (#5 *v.s.* #6). The underlying reason is that the two functions are very similar when the sequence length is short. The piecewise function is effective especially when the sequence size is much larger than the number of buckets. Object detection uses a much higher resolution input compared to classification, leading to a much longer input sequence. We therefore conjecture that when the input sequence is long, the piecewise function should be used since it is able to distribute different attentions to the positions with relative large distance, while the clip function assigns the same encoding when the relative distance is larger than β.

Number of buckets. The number of buckets largely affects model parameters, computational complexities and performance. In order to find a balance, we explore the influence of varying the number of buckets for the contextual Product method. Fig. 3 shows the change of top-1 accuracy along with the number of buckets. The accuracy increase
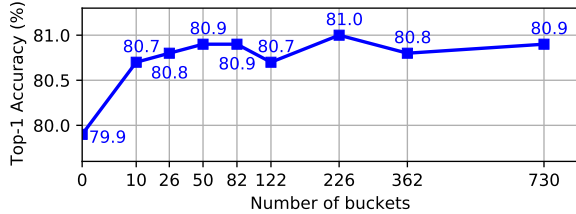
Figure 3: Ablation for the number of buckets in contextual product model with shared relative position encodings on ImageNet [4].
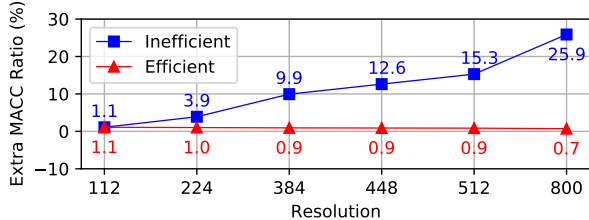

Figure 4: The extra computational cost of relative position encoding with different implementation in different resolutions. The baseline model is DeiT-S [22]. The number of buckets is 50. MACs means multiply-accumulate operations.

#	Abs Pos.	p_{ij}^Q	p_{ij}^K	p_{ij}^V	Top-1	Top-5
1 [22]	learnable	×	×	×	79.9	95.0
2	×	×	×	×	77.6(-2.3)	93.8
3	×	✓	×	×	80.9(+1.0)	95.4
4	×	×	✓	×	80.9(+1.0)	95.3
5	×	×	×	✓	80.2(+0.3)	95.0
6	×	✓	✓	×	81.0(+1.1)	95.5
7	×	✓	✓	✓	81.3(+1.4)	**95.7**
8	learnable	✓	×	×	80.9(+1.0)	95.5
9	learnable	×	✓	×	80.9(+1.0)	95.5
10	learnable	×	×	✓	80.2(+0.3)	95.1
11	learnable	✓	✓	×	81.1(+1.2)	95.4
12	learnable	✓	✓	✓	**81.4**(+1.5)	95.6

Table 4: Component-wise analysis on ImageNet [4]. We add contextual product shared-head relative position encodings into DeiT-S [22]. The number of buckets is 50. Abs Pos. represents the absolute position encoding. p_{ij}^Q, p_{ij}^K and p_{ij}^V present relative position encodings on queries, keys and values.

from 79.9 to 80.9 before 50 buckets. After that, there is no significant improvement. It shows that the number of buckets 50 is a good balance between the computational cost and the accuracy for 14×14 feature map in DeiT-S [22].

Component-wise analysis. We perform a component-wise analysis to study the effects of different position en-

Model	#Param.	Input	MACs (M)	Top-1 Acc (%)
Convnets				
ResNet-50 [10]	25M	224^2	4121	79.0
RegNetY-4.0GF [15]	21M	224^2	4012	79.4
EfficientNet-B1 [21]	8M	240^2	712	79.1
EfficientNet-B5 [21]	30M	456^2	10392	83.6
Transformers				
ViT-B/16 [6]	86M	384^2	55630	77.9
ViT-L/16 [6]	307M	384^2	191452	76.5
DeiT-Ti [22]	5M	224^2	1261	72.2
CPVT-Ti(0-5) [2]	6M	224^2	1262	73.4
DeiT-Ti with iRPE-K(**Ours**)	6M	224^2	1284	73.7
DeiT-S [22]	22M	224^2	4613	79.9
CPVT-S(0-5) [2]	23M	224^2	4616	80.5
DeiT-S(Shaw's) [22, 18]+	22M	224^2	4659	80.9
DeiT-S(Trans.-XL's) [22, 3]+	23M	224^2	4828	80.8
DeiT-S(Huang's) [22, 11]+	22M	224^2	4706	81.0
DeiT-S(SASA's) [22, 17]*	22M	224^2	4639	80.8
DeiT-S with iRPE-K(**Ours**)	22M	224^2	4659	80.9
DeiT-S with iRPE-QK(**Ours**)	22M	224^2	4706	81.1
DeiT-S with iRPE-QKV(**Ours**)	22M	224^2	4885	81.4
DeiT-B [22]	86M	224^2	17592	81.8
CPVT-B(0-5) [2]	86M	224^2	17598	81.9
DeiT-B with iRPE-K(**Ours**)	87M	224^2	17684	82.4

+ We utilize our product method to adapt 1D encoding for 2D images with the clip function. The encoding weight is shared across heads.
* DeiT-S [22] with SASA [17]'s relative position encoding.

Table 5: Comparison on ImageNet [4].

codings for vision transformer models. We select DeiT-S model [22] as the baseline, and only change the position encoding methods. The learnable absolute position encoding is used in the original model. The relative position encodings are computed by contextual Product method with 50 buckets. The conclusions we got from Tab. 4 are as follows: 1) Removing absolute position encoding from original DeiT-S will cause that the Top-1 accuracy drops from 79.9 to 77.6 (#1 *v.s.* #2). 2) The models with only relative position encoding surpass the one with only absolute position encoding (#3-5 *v.s.* #1). It shows that relative position encoding works well as the absolute one. 3) When equipped with relative position encoding, the absolute one does not bring any gains (#3-5 *v.s.* #8-10). We suppose that the local information is more important than the global one in classification task. 4) The relative position encoding on queries or keys brings more gain than that on values (#3,4 *v.s.* #5). 5) The combination of the encodings on queries, keys and values brings further improvements (#6,7,11,12 *v.s.* others).

Complexity Analysis. We evaluate the computational cost of our proposed methods with respect to different input resolutions. The baseline model is DeiT-S [22] with only absolute position encoding. We adopt contextual product shared-head relative position encoding to the baseline with 50 buckets. Fig. 4 shows our method takes at most 1% extra computational cost with efficient implementation.

#	Abs Pos.	Rel Pos.	#buckets	epoch	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
1 [1]	sinusoid	none	-	150	39.5	60.3	41.4	17.5	43.0	59.1
2	none	none	-	150	30.4 (-9.1)	52.5	30.2	9.4	31.2	50.5
3	sinusoid	bias	9×9	150	40.6 (+1.1)	61.2	42.8	19.0	43.9	60.2
4	none	contextual	9×9	150	38.7 (-0.8)	60.1	40.4	18.2	41.8	56.7
5	sinusoid	ctx clip	9×9	150	40.4 (+0.9)	60.9	42.4	19.1	43.7	59.8
6	sinusoid	contextual	9×9	150	40.8 (+1.3)	61.5	42.5	18.5	44.4	60.5
7	sinusoid	contextual	15×15	150	40.8 (+1.3)	61.7	42.6	18.5	44.2	61.2
8 [1]	sinusoid	none	-	300	40.6	61.6	-	19.9	44.3	60.2
9	sinusoid	contextual	9×9	300	42.3 (+1.7)	62.8	44.3	20.7	46.2	61.1

Table 6: Component-wise analysis on DETR [1].

4.3. Comparison on Image Classification

We compare our proposed methods with the state-of-the-art methods on image classification tasks. We select DeiT [22] as the baseline. We adopt contextual Product shared-head method with a buckets number of 50. As shown in Tab. 5, our method brings improvement on all three DeiT models. In particular, we improve the DeiT-Ti/DeiT-S/DeiT-B models by 1.5%/1.0%/0.6% respectively, through adding relative position encoding only on keys. We show that the models could be further improved by adding the proposed relative position on both queries and values. When compared with other methods, ours achieve superior performance with less parameters and MACs.

4.4. Comparison on Object Detection

To verify the generality of our method, we further evaluate it on COCO 2017 detection dataset [12]. We use the transformer-based detection model DETR [1] as our baseline. We follow the same training and testing settings (including hyperparameters) as DETR [1], except injecting relative position encoding into all self-attention modules in the encoder. As shown in Tab. 6 (#1,6 and #8,9), our method consistently improve the performance of DETR by 1.3mAP and 1.7mAP under 150 and 300 training epochs.

In addition, we conduct ablation studies analyzing that the effects of position encoding on object detection task. Comparing #1, #2 and #4 in Tab. 6, we give the conclusion that position encoding is crucial for DETR. We also show that absolute position embedding is better than relative position embedding in DETR, which is contrast to the observation in classification. We conjecture that DETR needs the prior of absolute position encoding to locate objects.

4.5. Visualization

We explore the underlying reason of relative position encoding in this subsection. We visualize the extra weights b_{ij} (defined in Eq. (13)) added into the attention by relative position encoding for different positions in each block. From Fig. 5, we could see that relative position encoding makes

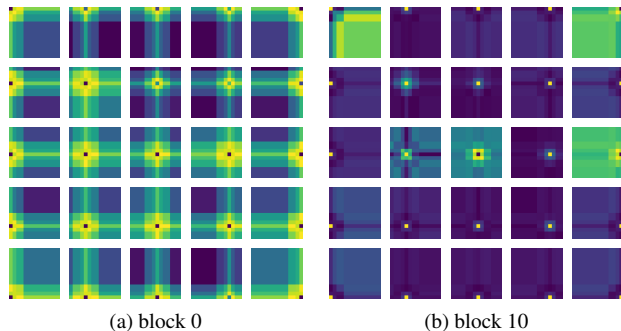|	
(a) block 0	(b) block 10

Figure 5: Visualization of relative position encoding (RPE) in contextual product method. We show the extra weights added to the attention by relative position encoding for different position. (a), (b) display the extra weights on attention for 5×5 reference patches uniformly sampled from 14×14 patches in block 0 and 10.

the current patch focus more on its neighboring patches in block 0. However, when it turns to higher block, this phenomenon disappears. We conjecture this is because after passing through multiple layers, the model has already captured enough local information. The shallow layers in transformer are also global attentions, which pay attention to the whole image (consisting of small patches). It is different from CNN models in which shallow layers only capture local information. In theory, without RPEs (or other additional operations such as local windows). transformer does not explicitly capture locality. RPEs inject Conv-like inductive bias (including locality) into transformer, improving the model capability of capturing local patterns.

5. Related Work

Transformer. Transformer was originally introduced by Vaswani *et al.* [23] for natural language processing, and recently extended to computer vision [6, 22, 1]. In this work we study vision transformers in image classification and object detection tasks, and select DeiT [22] and DETR [1] as

our baseline models. In ViT [6] and DeiT [22] models, an image is split into multiple fixed-size patches. The embedded features of patches are added with absolute position encoding to fed in a standard transformer encoder. An extra trainable classification token is added into the sequence for classification. In DETR [1], a CNN backbone is used for feature extraction first. It outputs a feature map downsampled $32\times$. Then it is flatten and fed to a transformer. The transformer outputs a certain number of bounding boxes. A learnable or sinusoid absolute position encoding is added in both transformer encoder and decoder.

Relative Position Encoding. Relative position encoding is proposed firstly by Shaw *et al.* [18], where relative position encodings are added into keys and values. Dai *et al.* [3] proposed relative position encoding with the prior of the sinusoid matrix and more learnable parameters. Huang *et al.* [11] proposed several 1D encoding variants. The effectiveness of relative position encoding has been verified in natural language processing. There are also some works utilizing relative position encoding on 2D visual tasks. Ramachandran *et al.* [17, 19] proposed 2D relative position encoding that computes and concatenates separate encodings of each dimension. Chu *et al.* [2] proposed position encoding generator, inserted between encoders. However, the efficacy of relative position encoding in visual transformer is still unclear, which is discussed and addressed in this work.

6. Conclusions and Remarks

In this paper, we review existing relative position encoding methods, and propose four methods dedicated to visual transformers. The abundant experiments show that our methods bring a clear improvement on both classification and detection tasks with negligible extra complexity. Our method could be easily plugged into the self-attention modules in vision models. In addition, we give comparison of different methods and analysis on relative position encoding with following conclusions. 1) Relative position encoding can be shared among different heads for parameter-saving. It is able to achieve comparable performance with the non-shared one in contextual mode. 2) Relative position encoding can replace absolute one in image classification task. However, absolute position encoding is necessary for object detection task, which needs to predict locations of objects. 3) Relative position encoding should consider the positional directivity, which is important to structured 2D images. 4) Relative position encoding forces the shallow layers in transformers to pay more attention to local patches.

In future work, we plan to extend our method to other attention-based models and scenarios, such as high-resolution input tasks like semantic segmentation [30], and non-pixel input tasks like point cloud classification [29, 9].

References

[1] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*, 2020. 1, 2, 8, 9, 11, 13

[2] Xiangxiang Chu, Bo Zhang, Zhi Tian, Xiaolin Wei, and Huaxia Xia. Do we really need explicit position encodings for vision transformers? *arXiv preprint arXiv:2102.10882*, 2021. 1, 7, 9

[3] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In *ACL*, 2019. 1, 3, 7, 9, 13

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 2, 6, 7, 13

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 1

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*, 2021. 1, 7, 8, 9

[7] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *ICML*, 2017. 1, 2

[8] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, pages 249–256, 2010. 13

[9] Meng-Hao Guo, Jun-Xiong Cai, Zheng-Ning Liu, Tai-Jiang Mu, Ralph R Martin, and Shi-Min Hu. Pct: Point cloud transformer. *arXiv preprint arXiv:2012.09688*, 2020. 9

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 7, 13

[11] Zhiheng Huang, Davis Liang, Peng Xu, and Bing Xiang. Improve transformer models with better relative position embeddings. In *EMNLP*, 2020. 3, 7, 9, 13

[12] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014. 2, 8

[13] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *arXiv preprint arXiv:2103.14030*, 2021. 11, 13

[14] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 6, 13

[15] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *CVPR*, 2020. 7

[16] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and

Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 21(140), 2020. 1, 4

[17] Prajit Ramachandran, Niki Parmar, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jonathon Shlens. Stand-alone self-attention in vision models. *arXiv preprint arXiv:1906.05909*, 2019. 3, 5, 7, 9, 13

[18] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *ACL*, 2018. 1, 2, 3, 4, 7, 9, 13

[19] Aravind Srinivas, Tsung-Yi Lin, Niki Parmar, Jonathon Shlens, Pieter Abbeel, and Ashish Vaswani. Bottle-neck transformers for visual recognition. *arXiv preprint arXiv:2101.11605*, 2021. 1, 9

[20] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016. 6

[21] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019. 7

[22] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herv'e J'egou. Training data-efficient image transformers & distillation through attention. *arXiv preprint arXiv:2012.12877*, 2020. 1, 2, 5, 6, 7, 8, 9, 11, 13

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. 1, 2, 3, 8

[24] Huiyu Wang, Yukun Zhu, Bradley Green, Hartwig Adam, Alan Yuille, and Liang-Chieh Chen. Axial-deeplab: Stand-alone axial-attention for panoptic segmentation. In *ECCV*, 2020. 3

[25] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *CVPR*, 2018. 1

[26] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *NeurIPS*, 32, 2019. 1

[27] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *ICCV*, 2019. 6

[28] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017. 6

[29] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip Torr, and Vladlen Koltun. Point transformer. *arXiv preprint arXiv:2012.09164*, 2020. 9

[30] Sixiao Zheng, Jiachen Lu, Hengshuang Zhao, Xiatian Zhu, Zekun Luo, Yabiao Wang, Yanwei Fu, Jianfeng Feng, Tao Xiang, Philip H.S. Torr, and Li Zhang. Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers, 2020. 9

Rethinking and Improving Relative Position Encoding for Vision Transformer
—— Supplementary Material ——

This supplementary material presents additional details of Section 3.2, 4.2, 4.3 and 4.4. Besides, two extra experiments are added to demonstrate the effectiveness and generality of the proposed iRPE. We also provide comparisons on the inference time.

- **Visualization of 2D relative position.** To provide an intuitive understanding, we visualize the proposed 2D relative position in Section 3.2, including Euclidean, Quantization, Cross and Product methods.
- **Weight initialization.** We elaborate the weight setting of the proposed relative position encoding methods, including the weight initialization and whether to equip with weight decay.
- **Computation complexity.** We provide a detailed explanation of why the computational costs are the same for shared and unshared relative position encodings across attention heads in Tab. 2 of Section 4.2.
- **Injecting previous RPE methods into DeiT.** We elaborate how to inject previous relative position encoding methods into DeiT [22] in Tab. 5 of Section 4.3.
- **Training and test settings of DETR.** We provide the details of training and test settings of DETR [1] in Section 4.4.
- **The effectiveness on other vision transformers.** We show the effectiveness of the proposed iRPE on the recent Swin transformer [13].
- **Transfer learning on fine-grained datasets.** To verify the generalizability, we evaluate our models on fine-grained datasets, including Stanford Cars and CUB200_2011 datasets.
- **Inference performance.** We compare the proposed iRPE with previous methods in terms of inference time and memory cost.

1. Visualization of 2D Relative Position

We visualize the proposed four relative position methods, *i.e.*, Euclidean, Quantization, Cross and Product, and present their difference. In DeiT [22], an image is split into 14×14 non-overlapping patches, so the number of tokens is 14×14 (except for the classification token). Therefore, in theory, each token has 14×14 relative positions. For visualization, we select the top-left position $(0, 0)$ and the center position $(7, 7)$ as the reference positions (presented by a red star ★ in the following figures), and then compute the relative offsets $\delta x = x_i - x_j$ and $\delta y = y_i - y_j$ between the reference position and the remaining $14 \times 14 - 1$ posi-
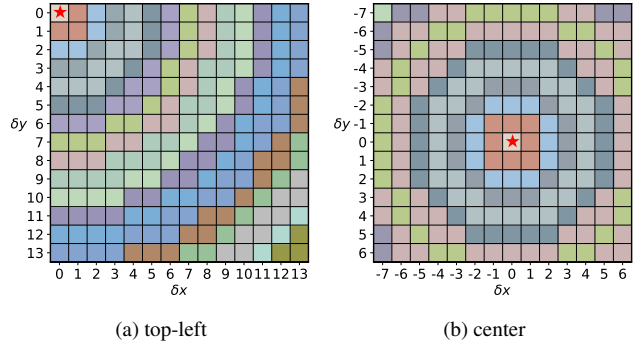

(a) top-left   (b) center

Figure 6: Visualization of Euclidean method. The red star ★ presents the reference position. Different color means different bucket. The relative positions with the same color share the same encoding.

tions. Let $(\delta x, \delta y)$ denote a 2D relative position. We plot the map of the relative encoding $\mathbf{r}_{ij}$, defined in Eq. (19), Eq. (21), Eq. (22) and Eq. (25), where i is the reference position and j is one of the 14×14 positions. Notice that $\mathbf{r}_{ij}$ is either a learnable scalar in bias mode or a vector in contextual mode. Multiple $\mathbf{r}_{ij}$ may share an identical bucket, which is presented by the same color in Fig. 6 - 9. Different bucket is presented by different color.

Euclidean method. Fig. 6 shows Euclidean method. It is an undirected method, since the relative position encodings only depend on relative Euclidean distance. For example, in Fig. 6b since the relative positions $(-1, 0)$ and $(1, 0)$ have the same relative Euclidean distance of 1, they are mapped into the same bucket (the grids with orange color).

Quantization method. Fig. 7 presents Quantization method, another undirected method. It is an improved version of Euclidean method, and addresses the problem that the close two neighbors with different relative distances might be mapped into the same bucket (*e.g.*, he relative position $(1, 0)$ and $(1, 1)$ are both mapped into the same bucket in Euclidean method). Besides, the number of buckets in Quantization method is larger than that in Euclidean method. The reason is that Quantization method quantize Euclidean distance from a set of real numbers $\{0, 1, 1.41, 2, 2.24, ...\}$ to a set of integers $\{0, 1, 2, 3, 4, ...\}$, increasing the number of buckets for adjacent positions.

Cross method. Fig. 8 shows Cross method. It is a directed method, in which the relative position encoding depends on relative distances and relative directions simultaneously. It computes the encodings on horizontal and verti-
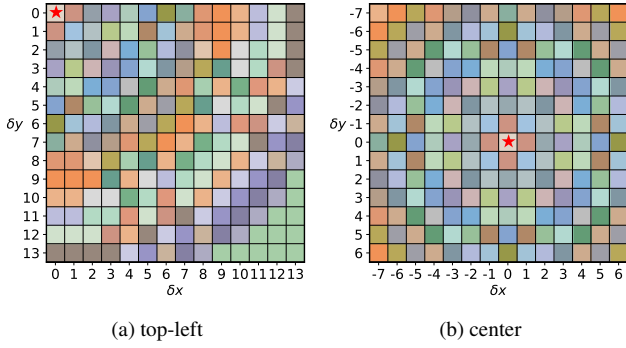
11

(a) top-left   (b) center

Figure 7: Visualization of Quantization method. The red star ★ presents the reference position. Different color means different bucket. The relative positions with the same color share the same encoding.
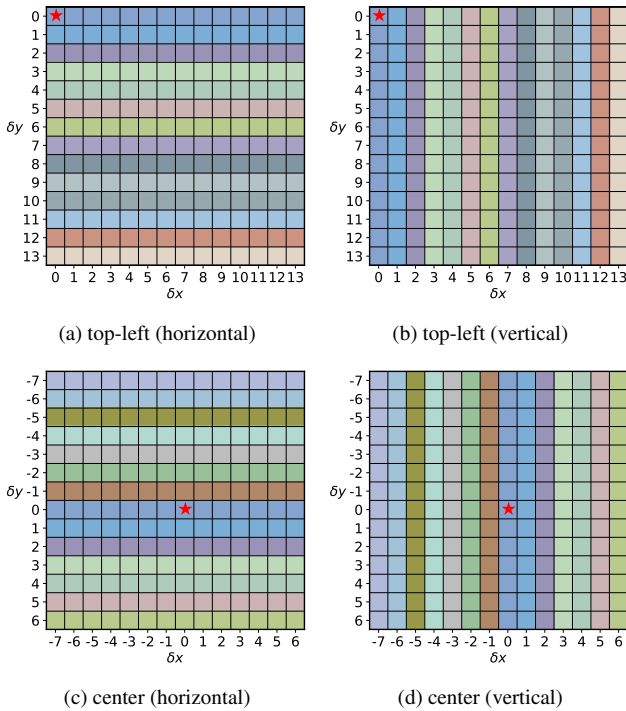

(a) top-left (horizontal)   (b) top-left (vertical)

(c) center (horizontal)   (d) center (vertical)

Figure 8: Visualization of Cross method. The red star ★ presents the reference position. Different color means different bucket. The relative positions with the same color share the same encoding.

cal directions separately, then summarizes them. The same offsets along x-axis (or y-axis) direction share the same horizontal (or vertical) encoding. For example, the two relative positions $(-1, 0)$ and $(1, 0)$ share the same encoding on horizontal in Fig. 8c, but not on vertical in Fig. 8d.

Product method. Fig. 9 shows Product method, which is also a directed method. Unlike Cross method, Product method does not share the same encoding even if the offsets are the same along x-axis or y-axis direction. For example,
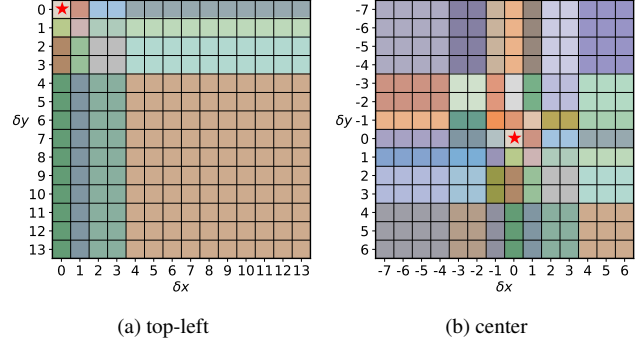

(a) top-left   (b) center

Figure 9: Visualization of Product method. The red star ★ presents the reference position. Different color means different bucket. The relative positions with the same color share the same encoding.

in Fig. 9b, the two relative positions $(-1, 0)$ and $(1, 0)$ have independent encodings. Moreover, it is more efficient than Cross method, since there is no extra addition operation in Eq. (22).

2. Weight Initialization

The relative position weight $\mathbf{r}_{ij}$ in Eq. (14) and Eq. (15) is initialized with zero. We found that there is no difference between zero and normal-distribution initialization. Besides, we do not impose weight decay on the weight of relative position encodings, because its effects on the final performance is negligible.

3. Computation Complexity

As shown in Tab. 2 (in the main manuscript), the computational costs MACs of shared and unshared relative position encodings across attention heads are the same. Here, we provide the detailed explanation. Let h, n, d, k denote the number of heads, the length of a sequence, the number of channels and the number of buckets, respectively. For bias mode, in Eq. (13), the broadcast addition on the dot-product attention $(\mathbf{x}_i \mathbf{W}^Q)(\mathbf{x}_j \mathbf{W}^K)^T$ with the shape of $h \times n \times n$ and the encoding b_{ij} with the shape of $n \times n$ in shared scheme or $h \times n \times n$ in unshared scheme takes the computational cost of $\mathcal{O}(hn^2)$. For contextual mode, in Eq. (27), the broadcast multiplication on the input embedding $\mathbf{x}_i \mathbf{W}$ with the shape $h \times n \times d$ and the relative position weight $\mathbf{p}$ with the shape of $d \times k$ in shared scheme or $h \times d \times k$ in unshared scheme takes the computational cost of $\mathcal{O}(hndk)$. Due to the broadcast operations, the computational cost of shared and unshared schemes is the same.

4. Injecting Previous RPE Methods into DeiT

In the Tab. 5, in order to compare with previous 1D relative position encoding methods, we utilize our Prod-
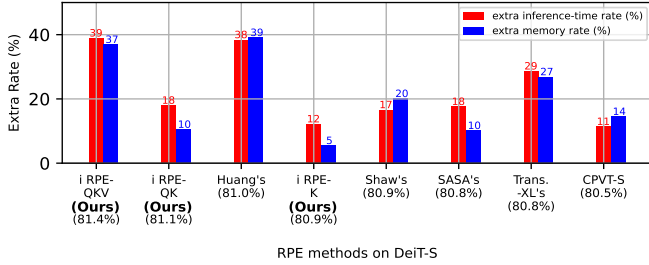
Figure 10: The extra cost brought by RPEs. The reference model is DeiT-S [22] without RPE, taking 1,096 images/s and 8,930 Mb memory.

uct method (defined in Sec. 3.2 in the main manuscript) to adapt 1D encoding methods for 2D images. We replace the piecewise function $g(x)$ with the clip function $h(x)$, which is matched with previous methods. The encoding weight is shared across attention heads. DeiT-S(Shaw's), DeiT-S(Trans.-XL's), DeiT-S(Huang's) are DeiT-S [22] models with Shaw's relative position encoding [18], relative position encoding in Transformer-XL [3] and Huang's relative position encoding [11], respectively. Besides, the 2D relative position encoding in SASA [17] is equipped on DeiT-S [22] directly.

5. Training and Test Settings of DETR

We follow the same training protocol and hyperparameter configurations as the original DETR [1]. The backbone model of DETR [1] is ResNet-50 [10], pretrained on ImageNet [4], and the BatchNorm layers are frozen during training. All transformer blocks are initialized with Xavier initialization [8]. The image is cropped such that the shortest side is at least 480 and at most 800 pixels while the longest at most 1333. When training, random horizontal flipping and random cropping are utilized. The initial learning rates of transformer and backbone are 10^{-4} and 10^{-5}, respectively. Learning rates are divided by 10 in the last 50 epochs in 150 epochs schedule, and the last 100 epochs in 300 epochs schedule. The optimizer is AdamW [14] with weight decay of 10^{-4} and a mini-batch size of 16. The number of queries is 100. We train the models for 150 epochs and 300 epochs.

6. The Effectiveness on Other Vision Transformers

We further verify the effectiveness of the proposed iRPE on the recent Swin transformer [13]. Specifically, the original Swin-T model without RPE obtains a top-1 accuracy of 80.5% (Tab. 4 in Swin transformer [13]), while using RPE bias mode gets +0.8% improvements. Our contextual RPE on QKV can further improve Swin-T to 81.9% on ImageNet.

7. Transfer Learning on Fine-grained Datasets

We finetune the pretrained models on Stanford Cars and CUB200_2011 datasets using the resolution 224x224 and 300 epochs. DeiT-B [22] with iRPE on keys obtains a top-1 accuracy of 93.4% and 84.9% on the two datasets respectively, outperforming the original DeiT-B (92.1% and 83.4%) by 1.3% and 1.5% points.

8. Inference Performance

The inference runtime and memory cost are reported in Fig. 10, tested on Nvidia V100 GPU with a batch size of 128. We can see that our iRPE on keys is more effective.