

# Conditional Positional Encodings for Vision Transformers

Xiangxiang Chu<sup>1</sup>, Zhi Tian<sup>2</sup>, Bo Zhang<sup>1</sup>, Xinlong Wang<sup>2</sup>, Xiaolin Wei<sup>1</sup>, Huaxia Xia<sup>1</sup>, Chunhua Shen<sup>2</sup>

<sup>1</sup>Meituan Inc., <sup>2</sup>The University of Adelaide

{chuxiangxiang,zhangbo97,weixiaolin02,xiahuaxia}@meituan.com, zhi.tian@outlook.com, xinlong.wang96, chhshen@gmail.com

## Abstract

We propose a conditional positional encoding (CPE) scheme for vision Transformers [10, 30]. Unlike previous fixed or learnable positional encodings, which are predefined and independent of input tokens, CPE is dynamically generated and conditioned on the local neighborhood of the input tokens. As a result, CPE can easily generalize to the input sequences that are longer than what the model has ever seen during training. Besides, CPE can keep the desired translation-invariance in the image classification task, resulting in improved classification accuracy. CPE can be effortlessly implemented with a simple Position Encoding Generator (PEG), and it can be seamlessly incorporated into the current Transformer framework. Built on PEG, we present Conditional Position encoding Vision Transformer (CPVT). We demonstrate that CPVT has visually similar attention maps compared to those with learned positional encodings. Benefit from the conditional positional encoding scheme, we obtain state-of-the-art results on the ImageNet classification task compared with vision Transformers to date. Our code will be made available at <https://github.com/Meituan-AutoML/CPVT>.

## 1. Introduction

Recently, Transformers [31] have been viewed as a strong alternative to Convolutional Neural Networks (CNNs) in visual recognition tasks such as classification [10] and detection [4, 36]. Unlike the convolution operation in CNNs, which has a limited receptive field, the self-attention mechanism in the Transformers can capture the long-distance information and dynamically adapt the receptive field according to the image content. As a result, the Transformers are considered more flexible and powerful than CNNs, thus being promising to achieve more progress in visual recognition.

However, the self-attention operation in Transformers is permutation-invariant, which cannot leverage the order of the tokens in an input sequence. To mitigate this issue, previous works [31, 10] add the absolute positional encod-

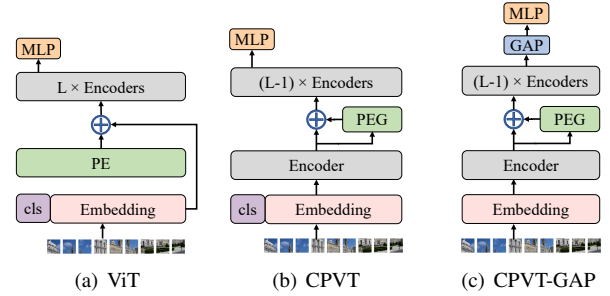


Figure 1. Vision Transformers: (a) ViT [10] with explicit 1D learnable positional encodings (PE) (b) CPVT with conditional positional encoding from the proposed Position Encoding Generator (PEG) plugin, which is the *default* choice. (c) CPVT-GAP without class token (cls), but with global average pooling (GAP) over all items in the sequence. Note that GAP is a bonus version which has boosted performance.

ings to each token in the input sequence (see Figure 1a), which enables order-awareness. The positional encoding can either be learnable or fixed with sinusoidal functions of different frequencies. Despite being effective and non-invasive, these positional encodings seriously harm the flexibility of the Transformers, hampering their broader applications. Taking the learnable version as an example, the encodings are often a vector of equal length to the input sequence, which are jointly updated with the network weights during training. Thus, the length and the value of the positional encodings are fixed once trained. During testing, it causes difficulty handling the sequences longer than the ones in the training data.

The inability to adapt to longer input sequences during testing greatly limits the generalization, because in vision tasks like object detection we expect the model can be applied to the images of any size during inference, which might be much larger than the training images. A possible remedy is to use bicubic interpolation to upsample the positional encodings to the target length, but it may degrade the performance without fine-tuning as later shown in our

experiments. Additionally, in computer vision, we expect that the models have translation-invariance. For example, in the classification task, the location of the target in an image should not alter the network’s activations so as to have the same classification result. However, the absolute positional encoding scheme breaks the translation-invariance because it adds unique positional encodings to each token (or each image patch). One may also use relative positional encodings as in [26]. However, relative positional encodings not only come with extra computational costs, but also require modifying the implementation of the standard Transformers. Last but not least, the relative positional encodings cannot work equally well as the absolute ones because the image recognition task still requires absolute position information (as mentioned in [13]), which the relative positional encodings fail to provide.

In this work, we advocate a novel positional encoding scheme to incorporate the position information into Transformers. Unlike the positional encodings used in previous works [10, 31, 26] which are predefined and input-agnostic, the proposed encodings are dynamically generated and conditioned on the local neighborhood of an input token. Thus, our positional encodings change according to the input size and can keep the desired translation-invariance. We demonstrate that the vision Transformer with our new encoding (*i.e.* CPVT, see Figure 1(c)) can lead to even better performance than previous vision Transformers [10, 30].

We summarize our contributions as follows.

- We propose a novel positional encoding scheme, termed *conditional position encodings* (CPE). CPE is dynamically generated with Positional Encoding Generators (PEG) and can be effortlessly implemented in the modern deep learning frameworks [18, 1, 6], requiring no changes to the current Transformer APIs.
- CPE is conditioned on the local neighborhood of input tokens and is adaptable to arbitrary input sizes, which enables processing images of much larger resolutions.
- As opposed to widely-used absolute positional encodings, CPE can keep the desired *translation-invariance* which helps to improve the image classification performance.
- Built on CPE, we propose Conditional Position encoding Vision Transformer (CPVT) and it achieves new state-of-the-art performance on ImageNet among prior vision Transformers [10, 30].
- Apart from the above, we also provide an orthogonal option to use translation-invariant global average pooling (GAP) for class prediction while removing the default class token [10]. By using GAP here, CPVT can be totally translation-invariant and CPVT’s performance thereby can be *further boosted* by about **1%**. In

contrast, the models based on absolute positional encodings can only obtain minor performance gain from GAP because its positional encodings already break the translation-invariance.

## 2. Related Work

### 2.1. Vision Transformers

Transformer [31] has achieved great success in natural language processing (NLP) and it is used in many state-of-the-art models such as BERT [9], GPT [19, 20], and GPT-3 [3]. Due to the flexibility of the transformer, recent works attempt to apply it to vision tasks such as classification [10, 30]. It has been shown that the self-attention module is able to replace the convolution operation [23], since self-attention layers can attend pixel-grid patterns similarly to the CNN layers [7]. One of the issues is that applying self-attention to vision tasks may lead to huge computational costs because an image may have tens of thousands of pixels. ViT [10] bypasses the issue by splitting an image into patches. Specifically, it decomposes a  $224 \times 224$  image into a series of 196 flattened patches ( $16 \times 16$  pixels each), which are analogous to a sequence of words in language processing. Based on ViT, DeiT [30] further makes the transformers more data-efficient, and it can be directly trained on ImageNet. Besides, Transformer is also used to solve other vision tasks such as object detection [4, 36], image segmentation [33, 35], image generation [17], and low-level vision tasks [5].

### 2.2. Positional Encodings

Since self-attention itself is permutation-equivalent, positional encodings are commonly employed to incorporate the order of sequences [31]. The positional encodings can either be fixed or learnable, while either being absolute or relative. Vision transformers follow the same fashion to imbue the network with positional information. These encoding methods are elaborated below.

**Absolute Positional Encoding.** The absolute positional encoding is the most widely used one. In the original transformer paper [31], the encodings are generated with the sinusoidal functions of different frequencies and then they are added to the inputs. Alternatively, the positional encodings can be learnable, where they are implemented with a fixed-dimension matrix and jointly updated with the model’s parameters with SGD.

**Relative Positional Encoding.** The relative position encoding [26] considers distances between the tokens in the input sequence. Compared to the absolute ones, the relative positional encodings can be translation-invariant and can naturally handle the sequences longer than the longest

sequences during training (*i.e.*, being inductive). A 2-D relative position encoding is proposed for image classification in [2], showing superiority to 2D sinusoidal embeddings. The relative positional encoding is further improved in XLNet [34], T5 [22] and DeBERTa [12], showing better performance.

**Positional Encoding with Dynamical Systems.** A new position encoding named FLOATER [15] was proposed to model the position information with a continuous dynamical model. FLOATER is not limited by the maximum length of the sequences during training, meanwhile it is data-driven and parameter-efficient.

Compared with the above methods, we propose a new positional encoding method, which keeps all the desired properties of these methods while being much simpler.

### 3. Vision Transformer with Conditional Position Encodings

#### 3.1. Motivations

In vision transformers, given an image of input size  $H \times W$ , it is split into patches with size  $S \times S$ , the number of patches is  $N = \frac{HW}{S^2}$ <sup>1</sup>. The patches are added with the same number of learnable absolute positional encoding vectors. In this work, we argue that the positional encodings used here have two issues. First, **it prevents the model from handling the sequences longer than the longest training sequences.** Second, **it makes the model not translation-invariant because a unique positional encoding vector is added to every one patch.** The translation invariance plays an important role in classification tasks because we hope the networks give the same responses wherever the object is in the image.

One may note that the first issue can be remedied by removing the positional encodings since except for the positional encodings, all other components (*e.g.*, MHSA and FFN) of the vision transformer can directly be applied to longer sequences. However, this solution severely deteriorates the performance. This is understandable because the order of the input sequence is an important clue and the model has no way to employ the order without the positional encodings. The experiment results on ImageNet are shown in Table 1. By removing the positional encodings, DeiT-tiny’s performance on ImageNet dramatically degrades from 72.2% to 68.2% (compared with the original learnable positional encodings). This confirms that position information plays a very important role in visual perception. Second, in DeiT [30], they show that we can interpolate the position encodings to make them have the same length as the longer sequences. However, this method requires fine-

Model	Encoding	Top-1@224(%)	Top-1@384(%)
DeiT-tiny [30]	✗	68.2	68.6
DeiT-tiny [30]	learnable	72.2	71.2
DeiT-tiny [30]	sin-cos	72.3	70.8
DeiT-tiny	2D RPE [26]	70.5	69.8

Table 1. Comparison of various positional encoding (PE) strategies tested on ImageNet validation set in terms of the top-1 accuracy. Removing the positional encodings greatly damages the performance. The relative positional encodings have inferior performance to the absolute ones.

tuning the model a few more epochs, otherwise the performance will remarkably drop. This goes contrary to what we would expect. With the higher-resolution inputs, we often expect a remarkable performance improvement without any fine-tuning. The experimental results are shown in Table 1. Finally, the relative position encodings [26, 2] can cope with both the aforementioned issues. However, the relative positional encoding cannot provide any *absolute position information*. As shown in [13], the absolute position information is also important to the classification task. This is also confirmed in our experiments. As shown in Table 1, the model with relative position encodings has inferior performance (70.5% vs. 72.2%).

#### 3.2. Conditional Positional Encodings

In this work, we argue that a successful positional encoding for vision tasks should meet the following requirements,

- (1) Making the input sequence *permutation-variant* but *translation-invariant*.
- (2) Being inductive and able to handle the sequences longer than the ones during training.
- (3) Having the ability to provide the absolute position to a certain degree. This is important to the performance as shown in [13].

In this work, we find that characterizing the local relationship by positional encodings is sufficient to meet all of the above. First, it is *permutation-variant* because the permutation of input sequences also affects the order in some local neighborhoods. In contrast, the translation of an object in an input image might not change the order in its local neighborhood, *i.e.*, translation-invariant. Second, the model can easily generalize to longer sequences since only the local neighborhoods of a token are involved. Besides, if the absolute position of any input token is known, the absolute position of all the other tokens can be inferred by the mutual relation between input tokens. We will show that the tokens on the borders can be aware of their absolute positions if we use zero paddings.

<sup>1</sup>  $H$  and  $W$  shall be divisible by  $S$ , respectively.

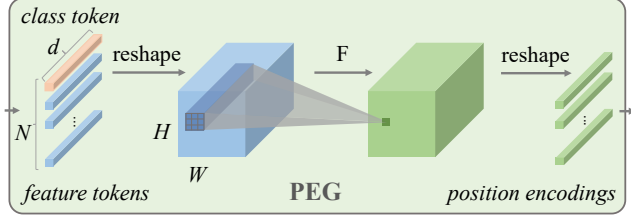


Figure 2. Schematic illustration of Positional Encoding Generator (PEG). Note  $d$  is the embedding size,  $N$  is the number of tokens. The function  $\mathcal{F}$  can be depth-wise, separable convolution or other complicated blocks.

Therefore, we propose *positional encoding generators* (PEG) to dynamically produce the positional encodings conditioned on the local neighborhood of an input token.

**Positional Encoding Generator.** PEG is illustrated in Figure 2. To condition on the local neighbors, we first reshape the flattened input sequence  $X \in \mathbb{R}^{B \times N \times C}$  of DeiT back to  $X' \in \mathbb{R}^{B \times H \times W \times C}$  in the 2-D image space. Then, a function (denoted by  $\mathcal{F}$  in Figure 2) is repeatedly applied to the local patch in  $X'$  to produce the conditional positional encodings  $E^{B \times H \times W \times C}$ . PEG can be efficiently implemented with a 2-D convolution with kernel  $k$  ( $k \geq 3$ ) and  $\frac{k-1}{2}$  zero paddings. Note that the zero paddings here are important to make the model be aware of the absolute positions, and  $\mathcal{F}$  can be of various forms such as separable convolutions and many others.

### 3.3. Conditional Positional Encoding Vision Transformers

Built on the conditional positional encodings, we propose our Conditional Positional Encoding Vision Transformers (CPVT). Except that our positional encodings are conditional, we exactly follow ViT and DeiT to design our vision transformers. We also have three sizes CPVT-Ti, CPVT-S and CPVT-B. Similar to the original positional encodings in DeiT, the conditional positional encodings are also added to the input sequence, as shown in Fig. 1 (b). In CPVT, the position where PEG is applied is also important to the performance, which will be studied in the experiments.

In addition, both DeiT and ViT utilize an extra learnable class token to perform classification (*i.e.*, `cls.token` shown in Fig. 1 (a) and (b)). By design, the class token is not translation-invariant although it can learn to be translation-invariant. A simple alternative is to directly replace it with a global average pooling (GAP), which is inherently translation-invariant. Therefore, we also propose CPVT-GAP, where the class token is replaced with a global average pooling. Together with the translation-invariant positional encodings, CPVT-GAP is uttermost translation-

Model	#channels	#heads	#layers	#params
CPVT-Ti	192	3	12	6M
CPVT-S	384	6	12	22M
CPVT-B	768	12	12	86M

Table 2. CPVT architecture variants. The larger model, CPVT-B, has the same architecture as ViT-B [10] and DeiT-B [30]. CPVT-S and CPVT-Ti have the same architecture as DeiT-small and DeiT-tiny respectively.

invariant and can achieve much better performance.

## 4. Experiments

### 4.1. Setup

**Datasets.** Following DeiT [30], we use ILSVRC-2012 ImageNet dataset [8] with 1K classes and 1.3M images to train all our models. We report the results on the validation set with 50K images. Unlike ViT [10], we do not use the much larger undisclosed JFT-300M dataset [27].

**Model variants.** As mentioned before, we also have three models with various sizes to adapt to various computing scenarios. The detailed settings are shown in Table 2. All experiments in this paper are performed on Tesla V100 machines. Training the tiny model for 300 epochs takes about 1.3 days on a single node with 8 V100 GPU cards. CPVT-S and CPVT-B take about 1.6 and 2.5 days, respectively. Our proposed PEG has little impact on the computational complexity of the models and uses even fewer parameters than the original learnable positional encodings in DeiT, as shown in Sec. 4.4.

**Training details** All the models (except for CPVT-B) are trained for 300 epochs with a global batch size of 2048 on Tesla V100 machines using AdamW optimizer [16]. We don't tune the hyper-parameters and strictly comply with the settings in DeiT [30]. The learning rate is scaled with this formula  $lr_{scale} = \frac{0.0005 * BatchSize_{global}}{512}$ . Although it may be sub-optimal for our method, our approach can obtain competitive results compared with [30]. The detailed hyperparameters are in the supplementary.

### 4.2. Generalization to Higher Resolutions

As mentioned before, our proposed PEG can directly generalize to larger image sizes without any fine-tuning. We confirm this here by evaluating the models trained with  $224 \times 224$  images on the  $384 \times 384$  images. The results are shown in Table 3. With the  $384 \times 384$  input images, The DeiT-tiny with learnable positional encodings degrades from 72.2% to 71.2%. When equipped with sine encoding, the tiny model degrades from 72.2% to 70.8%. In contrast,



CPVT model with the proposed PEG can directly process the larger input images. As a result, CPVT-Ti’s performance is boosted from 73.4% to 74.2% when applied to  $384 \times 384$  images. The gap between DeiT-tiny and CPVT-Ti is further enlarged to 3.0%.

Model	Params	Top-1 @ 224(%)	Top-1 @ 384(%)
DeiT-tiny [30]	6M	72.2	71.2
DeiT-tiny (sine)	6M	72.3	70.8
CPVT-Ti	6M	72.4	73.2
DeiT-small [30]	22M	79.9	78.1
CPVT-S	22M	79.9	80.4
DeiT-base [30]	86M	81.8	79.7
CPVT-B	86M	81.9	82.3

Table 3. Direct evaluation on higher resolutions without fine-tuning. A simple PEG of single layer of  $3 \times 3$  depth-wise convolution is used here.

We further evaluate the performance of fine-tuning the models on larger images here. Specifically, we fine-tune the small model trained on 224 to 384 (fixed) for 20 epochs with a batch size of 512 using a learning rate of  $5e-6$ . As for fine-tuning DeiT, we adopt a bicubic interpolation that approximately preserves the norm of the vectors for position embedding as [30] or we observe a significant drop of more than 6% as [30]. While DeiT-small obtains 81.5% top-1 accuracy, CPVT-B obtains 82.4% without any interpolations. This indicates fine-tuning can decrease the gap. Whereas, the performance gap does exist even using fine-tuning.

### 4.3. CPVT with Global Average Pooling

By design, the proposed PEG is translation-invariant. Thus, if we use the translation-invariant global average pooling (GAP) instead of the `cls_token` before the final classification layer of CPVT. CPVT can be totally translation-invariant, which should be beneficial to the ImageNet classification task. Note the using GAP here results in even less computation complexity because we do not need to compute the attention interaction between the class token and the image patches. As shown in Table 4, using GAP here can boost CPVT as least **1%**. For example, equipping CPVT-Ti with GAP obtains 74.9% top-1 accuracy on ImageNet validation dataset, which outperforms DeiT-tiny by a large margin (+2.7%). Moreover, it even exceeds DeiT-tiny model with distillation (74.5%). In contrast, DeiT with GAP cannot gain so much improvement (only 0.4% as shown in Table 4) because the original absolute positional encodings already break the translation-invariance. Given the superior performance, we hope our find can be a strong alternative for further studies on vision transformers.

Model	Head	Params	Top-1 Acc(%)	Top-5 Acc(%)
DeiT-tiny [30]	CLT	6M	72.2	91.0
DeiT-tiny	GAP	6M	72.6	91.2
CPVT-Ti ‡	CLT	6M	73.4	91.8
CPVT-Ti ‡	GAP	6M	<b>74.9</b>	<b>92.6</b>
DeiT-small [30]	CLT	22M	79.9	95.0
CPVT-S ‡	CLT	23M	80.5	95.2
CPVT-S ‡	GAP	23M	<b>81.5</b>	<b>95.7</b>

‡: Insert one PEG each after the first encoder till the fifth encoder

Table 4. Performance comparison of Class Token (CLT) and global average pooling (GAP) on ImageNet. CPVT’s can be further boosted when equipped with GAP.

### 4.4. Complexity of PEG

**Few Parameters.** Given the model dimension  $d$ , the extra number of parameters introduce by PEG is  $d \times l \times k^2$  if we choose  $l$  depth-wise convolutions with kernel  $k$ . Even if we use  $l$  separable convolutions, this value becomes  $l(d^2 + k^2d)$ . When  $k = 3$  and  $l = 1$ , CPVT-Ti ( $d = 192$ ) brings about 1728 parameters. Note that DeiT-tiny utilizes learnable position encodings with  $192 \times 14 \times 14 = 37632$  parameters. Therefore, CPVT-Ti has 35904 fewer number of parameters than DeiT-tiny. Even using 4 layers of separable convolutions, CPVT-Ti introduces only  $38952 - 37632 = 960$  more parameters, which is negelectable compared to the 5.7M model parameters of DeiT-tiny.

**FLOPs.** As for FLOPs,  $l$  layers of  $k \times k$  depth-wise convolutions possesses  $14 \times 14 \times d \times l \times k^2$  FLOPs. Taking the tiny model for example, it involves  $196 \times 192 \times 9 = 0.34M$  FLOPs for the simple case  $k = 3, l = 1$ , which is neglectable because the model has 2.1G FLOPs in total.

### 4.5. Qualitative Analysis of CPVT

Thus far, we have shown that PEG can have better performance than the original positional encodings. However, because PEG provides the position in an implicit way it is interesting to see if PEG can indeed provide the position information as the original positional encodings. Here we investigate this by visualizing the attention weights of the transformers. Specifically, given a  $224 \times 224$  image (i.e.  $14 \times 14$  patches), the score matrix within a single head is  $196 \times 196$ . We visualize the normalized self-attention score matrix of the second encoder block.

We first visualize the attention weights of DeiT with the original positional encodings. As shown in Figure 3 (middle), the diagonal element interacts strongly with its local neighbors but weakly with those far-away elements, which suggests that DeiT with the original positional encodings learn to attend the local neighbors of each patch. After the positional encodings are removed (denoted by DeiT w/o

PE), all the patches produce similar attention weights and fail to attend to the patches near themselves, see Figure 3 (left).

Finally, we show the attention weights of our CPVT model with PEG. As shown in Fig. 3 (right), like the original positional encodings, the model with PEG can also learn a similar attention pattern, which indicates that the proposed PEG can provide the position information as well.

#### 4.6. Comparison with State-of-the-art Methods

We evaluate the performance of CPVT models on the ImageNet validation dataset and report the results in Table 5. Compared with DeiT, *CPVT models have much better top-1 accuracy with similar throughputs*. Our models enjoy performance improvement on input upscaling for free, while DeiT degrades as discussed in Section 4.2, see also Figure 4 for a clear comparison. Noticeably, Our GAP version as explained in Section 4.3, marks a new state-of-the-art for vision Transformers.

Models	Params(M)	Input	throughput*	Top-1(%)
Convnets				
ResNet-50 [11]	25	224 <sup>2</sup>	1226.1	76.2
ResNet-101 [11]	45	224 <sup>2</sup>	753.6	77.4
ResNet-152 [11]	60	224 <sup>2</sup>	526.4	78.3
RegNetY-4GF [21]	21	224 <sup>2</sup>	1156.7	80.0
EfficientNet-B0 [28]	5	224 <sup>2</sup>	2694.3	77.1
EfficientNet-B1 [28]	8	240 <sup>2</sup>	1662.5	79.1
EfficientNet-B2 [28]	9	260 <sup>2</sup>	1255.7	80.1
EfficientNet-B3 [28]	12	300 <sup>2</sup>	732.1	81.6
EfficientNet-B4 [28]	19	380 <sup>2</sup>	349.4	82.9
Transformers				
ViT-B/16 [10]	86	384 <sup>2</sup>	85.9	77.9
ViT-L/16	307	384 <sup>2</sup>	27.3	76.5
DeiT-tiny w/o PE [30]	6	224 <sup>2</sup>	2536.5	68.2
DeiT-tiny [30]	6	224 <sup>2</sup>	2536.5	72.2
DeiT-tiny (sine)	6	224 <sup>2</sup>	2536.5	72.3
CPVT-Ti <sup>‡</sup>	6	224 <sup>2</sup>	2500.7	73.4
<b>CPVT-Ti-GAP<sup>‡</sup></b>	6	224 <sup>2</sup>	2520.1	<b>74.9</b>
DeiT-small[30]	22	224 <sup>2</sup>	940.4	79.9
CPVT-S <sup>‡</sup>	23	224 <sup>2</sup>	930.5	80.5
<b>CPVT-S-GAP<sup>‡</sup></b>	23	224 <sup>2</sup>	942.3	<b>81.5</b>
DeiT-base[30]	86	224 <sup>2</sup>	292.3	81.8
<b>CPVT-B<sup>‡</sup></b>	88	224 <sup>2</sup>	285.5	<b>82.3</b>
DeiT-tiny [30] <sup>⌘</sup>	6	224 <sup>2</sup>	2536.5	74.5
<b>CPVT-Ti<sup>⌘</sup></b>	6	224 <sup>2</sup>	2500.7	<b>75.9</b>

\*: Measured in img/s on a 16GB V100 GPU as in [30].

‡: Insert one PEG each after the first encoder till the fifth encoder

⌘: trained with hard distillation using RegNetY-160 as the teacher.

Table 5. Comparison with ConvNets and Transformers on ImageNet. CPVT models have much better performance compared with prior Transformers.

We further train CPVT-Ti and DeiT-tiny using the above-mentioned training setting plus hard distillation proposed in [30]. Specifically, we use RegNetY-160 [21] as the teacher.

CPVT obtains 75.9% top-1 accuracy, exceeding DeiT-tiny by 1.4%.

## 5. Ablation Study

### 5.1. Positions of PEG in CPVT

We also experiment by varying the position of the PEG in the model. Table 6 presents the ablations for variable positions based on the tiny model. *We denote the input of the first encoder by index -1*. Therefore, position 0 is the output of the first encoder block. Our method shows strong performance ( $\sim 72.4\%$ ) when PEG is placed at [0, 3].

It is interesting to note that positioning the PEG at 0 can have much better performance than positioning it at -1 (*i.e.*, before the first encoder), as shown in Table 6. We observe that the difference between the two is they have different receptive fields. Specifically, the former has a global field while the latter only attends to a local area. Hence, *it is supposed to work similarly well if we enlarge the convolution's kernel size*. To verify our hypothesis, we use quite a large kernel size 27 with a padding size 13 at position -1, whose result is reported in Table 7. It achieves similar performance to the one positioning the PEG at 0 (72.5% top-1 accuracy), which verifies our assumption.

Position Idx	Top-1 Acc(%)	Top-5 Acc(%)
none	68.2	88.7
-1	70.6	90.2
0	<b>72.4</b>	<b>91.2</b>
3	72.3	91.1
6	71.7	90.8
10	69.0	89.1

Table 6. Performance of different plugin positions using the architecture of DeiT-tiny on ImageNet.

PosIdx	kernel	Params	Top-1 Acc(%)	Top-5 Acc(%)
-1	3×3	5.7M	70.6	90.2
-1	27×27	5.8M	<b>72.5</b>	<b>91.3</b>

Table 7. Performance of different kernels (position -1).

### 5.2. Single PEG vs. Multiple PEGs

We further evaluate whether using *multi-position* encodings benefits the performance in Table 8. Notice we denote by  $i-j$  the inserted positions of PEG which start from  $i$ -th and end at  $j-1$ -th encoder. By inserting PEGs to five positions, the top-1 accuracy of the tiny model is further improved to 73.4%, which surpasses DeiT-tiny by 1.2%. Similarly, CPVT-S achieves a new state of the art (80.5%).

### 5.3. Comparisons with other positional encodings

In this section, we compare PEG with other commonly used encodings: absolute positional encoding (e.g. sinu-

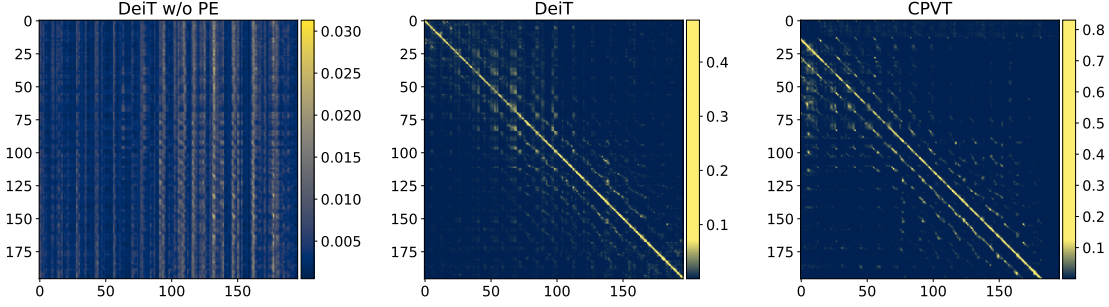


Figure 3. Normalized attention scores from the second encoder block of DeiT, DeiT without position encoding (DeiT w/o PE) [30], and CPVT on the same input sequence. Position encodings are key to developing a schema of locality in lower layers of DeiT. Meantime, CPVT profits from conditional encodings and follows a similar locality pattern.

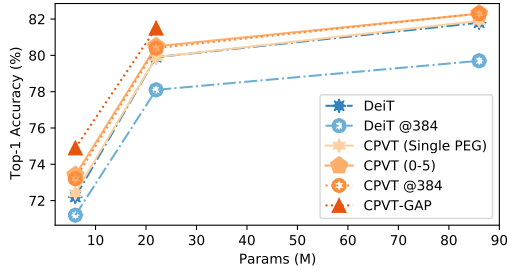


Figure 4. Comparison of CPVT and DeiT models under various configurations.

Positions	Model	Params(M)	Top-1 Acc(%)	Top-5 Acc(%)
0-1	tiny	5.7	72.4	91.2
0-5	tiny	5.9	<b>73.4</b>	<b>91.8</b>
0-11	tiny	6.1	<b>73.4</b>	<b>91.8</b>
0-1	small	22.0	79.9	95.0
0-5	small	22.9	80.5	<b>95.2</b>
0-11	small	23.8	<b>80.6</b>	<b>95.2</b>

Table 8. CPVT’s sensitivity to number of plugin positions.

soidal [31]), *relative positional encoding* (RPE) [26] and *learnable encoding* (LE) [9, 19]. We show the results in Table 9. All models are the same except for the positional encodings.

Model	PEG Pos	Encoding	Top-1(%)	Top-5 (%)
DeiT-tiny [30]	-	LE	72.2	91.0
DeiT-tiny	-	2D sin-cos	72.3	91.0
DeiT-tiny	-	2D RPE	70.5	90.0
CPVT-Ti	0-1	PEG	72.4	91.2
CPVT-Ti	0-1	PEG + LE	72.9	91.4
CPVT-Ti	0-1	4×PEG + LE	72.9	91.4
<b>CPVT-Ti</b>	0-5	PEG	<b>73.4</b>	<b>91.8</b>

Table 9. Comparison of various encoding strategies. LE: learnable encoding. RPE: relative positional encoding.

DeiT-tiny obtains 72.2% with learnable absolute encoding. We also experiment with the 2-D sinusoidal encodings to achieve on-par performance. As for RPE, we follow [26] and set the local range hyper-parameter  $K$  as 8, with which we obtain 70.5% top-1 accuracy.

Moreover, we combine the learnable absolute encoding with a single-layer PEG. This boosts the baseline CPVT-Ti (0-1) by 0.5. If we use 4-layer PEG, it can achieve 72.9%. Moreover, if we add a single layer PEG to the first five blocks, we can obtain 73.4% top-1 accuracy, which indicates that it is better to add the positional encodings to the more levels of the encoders.

#### 5.4. Importance of Zero Paddings

We design an experiment to verify the importance of the absolute position inferred from *zero paddings*. Specifically, we use CPVT-S and simply remove the zero paddings from CPVT while keeping all other parts unchanged. Table 10 shows that this can only obtain 70.5%, which indicates that absolute positional information plays an important role in classifying objects. This is the case because the category for each image is mainly labeled by the object in the center. The models require the absolute position to determine which patch is in the center.

Model	Padding	Top-1 Acc(%)	Top-5 Acc(%)
CPVT-Ti	✓	<b>72.4</b>	<b>91.2</b>
	✗	70.5	89.8

Table 10. ImageNet Performance w/ or w/o zero paddings.

#### 5.5. Where are the improvements from?

One might suspect that the PEG’s improvement comes from the *learnable parameters* introduced by the convolutional layers in PEG, instead of the local relationship retained by PEG. To disprove it, we experiment with a randomly-initialized  $3 \times 3$  PEG and fix its weights during

the training. As shown in Table 11, we still obtain 71.3% accuracy, which is much higher (3.1% $\uparrow$ ) than DeiT without any encodings (68.2%). Because the weights of the PEG are fixed and thus the performance improvement can only be due to the retained local relationship. As a comparison, we also use 12 convolutional layers with kernel sizes being 1 to replace the PEG, these layers has much more learnable parameters than PEG. However, it only boosts the performance to 68.6% (0.4% $\uparrow$ ).

Kernel	Style	Params(M)	Top-1 Acc(%)
none	-	5.68	68.2
3	fixed (random init)	5.68	71.3
3	fixed (learned init)	5.68	72.3
1 (12 $\times$ )	learnable	6.13	68.6
3	learnable	5.68	<b>72.4</b>

Table 11. Ablation to decide the impact factor.

Another interesting finding is that we use a learned PEG instead of the random initialized one and train the tiny version of the model from scratch (keeping the PEG fixed). It can also achieve 72.3% top-1 accuracy on ImageNet. This is very close to the learnable PEG (72.4), which suggests PEGs are important to extract the position information.

## 5.6. PEG on other Transformer Architectures

PVT [32] is a recently-proposed vision transformer with the multi-stage design like ResNet [11]. We also experiment with it to demonstrate the generalization of our method. Specifically, we remove learned position embedding and apply PEG in position 0 of each stage with a GAP head. With our PEG, the number of parameters can be reduced by 0.4M. We use the same training settings to make a fair comparison and show the results in Table 12. Our method can significantly boost PVT-tiny by 2.1% on ImageNet.

Model	Params	Top-1 Acc(%)	Top-5 Acc(%)
PVT-tiny [32]	13.2M	75.0	92.4
PVT-tiny + PEG	12.8M	<b>77.1</b>	93.6

Table 12. Our method boosts the performance of PVT [32].

## 5.7. Object Detection

We further apply our PEG to the transformer-based object detector DETR [4]. We report the performance on the COCO [14] dataset. Compared to the original DETR model, we only change the positional encoding strategy of its encoder part, which originally uses the absolute 2D sine and cosine positional encodings. We make some changes to the training strategy. Specifically, we reduce the training epochs from 500 to 50 since it takes about 2000 GPU hours to train DETR for 500 epochs [4, 36]. We hope this setting

will help the community to form a resource-efficient baseline. Following the DETR paper, we also use AdamW [16] with a total batch size of 32 and 0.0001 weight decay. The initial learning rates of the backbone and transformer are  $2 \times 10^{-5}$  and  $1 \times 10^{-4}$  respectively. The learning rates are decayed by  $0.1 \times$  at epoch 40. We use the same loss functions as DETR (*i.e.*,  $l_1$  loss for bounding box (5.0), classification loss (1.0) and GIoU loss (2.0) [25]).

Table 13 shows the results if the positional encodings are removed, where the mAP of DETR degrades from 33.7% to 32.8%. PEG improves the performance to 33.9%, which is even better than DETR with the original positional encodings. The same results hold for Deformable DETR [36]. Replacing the original 2-D since positional encodings with PEG obtains better performance under various settings. The excellent performance on object detection also suggests that PEG can provide the absolute position information because the object detection task requires the absolute coordinates of the bounding boxes.

Method	Epoch	AP	AP <sub>50</sub>	AP <sub>75</sub>	Params.	FPS
FCOS [29]	36	41.0	59.8	44.1	-	23
Faster R-CNN [24]	109	42.0	62.1	45.5	42M	26
DETR [30]	500	42.0	62.4	44.2	41M	28
DETR-DC5	500	43.3	63.1	45.9	41M	12
DETR w/o PE*	50	32.8	54.0	33.5	41M	28
DETR*	50	33.7	54.5	34.7	41M	28
<b>DETR w/ PEG</b>	50	<b>33.9</b>	<b>54.9</b>	<b>34.7</b>	41M	28
DD [36]	50	39.4	-	-	34M	27
<b>DD w/ PEG</b>	50	<b>39.7</b>	60.1	42.3	34M	27
DETR-DC5	50	35.3	55.7	36.8	41M	12
DD-DC5 [36]	50	41.5	61.5	44.8	34M	22
<b>DD-DC5 w/ PEG</b>	50	<b>41.9</b>	<b>61.8</b>	<b>45.1</b>	34M	22

<sup>1</sup> DD: Deformable DETR [36]. For DD, we always use single scale.

\* reproduced results using the released code.

Table 13. Results on COCO 2017 val set. PEG is a strong competitor to the 2D sinusoidal positional encodings.

## 6. Conclusion

In this paper, we have introduced CPVT, a novel method to provide the position information in vision Transformers, which dynamically generates the position encodings based on the local neighbors of each token. Through extensive experimental studies, we demonstrate that our proposed positional encodings can achieve stronger performance than the previous positional encodings. The transformer models with our positional encodings can naturally process longer input sequences and keep the desired translation-invariance in vision tasks. Moreover, our positional encodings are easy to implement and come with negligible cost. We look forward to a broader application of our proposed method in transformer-driven vision tasks like segmentation and video processing.



## References

- [1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016. 2
- [2] Irwan Bello, Barret Zoph, Ashish Vaswani, Jonathon Shlens, and Quoc V Le. Attention augmented convolutional networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3286–3295, 2019. 3
- [3] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020. 2
- [4] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision*, pages 213–229. Springer, 2020. 1, 2, 8
- [5] Hanting Chen, Yunhe Wang, Tianyu Guo, Chang Xu, Yiping Deng, Zhenhua Liu, Siwei Ma, Chunjing Xu, Chao Xu, and Wen Gao. Pre-trained image processing transformer. *arXiv preprint arXiv:2012.00364*, 2020. 2
- [6] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015. 2
- [7] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. On the relationship between self-attention and convolutional layers. In *International Conference on Learning Representations*, 2020. 2
- [8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 4
- [9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019. 2, 7
- [10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. 1, 2, 4, 6, 11
- [11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 6, 8, 12
- [12] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. DeBERTa: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654*, 2020. 3
- [13] Md Amirul Islam, Sen Jia, and Neil DB Bruce. How much position information do convolutional neural networks encode? In *International Conference on Learning Representations*, 2020. 2, 3
- [14] 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 *European conference on computer vision*, pages 740–755. Springer, 2014. 8
- [15] Xuanqing Liu, Hsiang-Fu Yu, Inderjit Dhillon, and Cho-Jui Hsieh. Learning to encode position for transformer with continuous dynamical model. In *International Conference on Machine Learning*, pages 6327–6335. PMLR, 2020. 3
- [16] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. 4, 8
- [17] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4055–4064, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. 2
- [18] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32:8026–8037, 2019. 2
- [19] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018. 2, 7
- [20] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019. 2
- [21] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10428–10436, 2020. 6
- [22] 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. *arXiv preprint arXiv:1910.10683*, 2019. 3
- [23] Prajit Ramachandran, Niki Parmar, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jon Shlens. Stand-alone self-attention in vision models. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 68–80. Curran Associates, Inc., 2019. 2
- [24] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: towards real-time object detection with region proposal networks. In *Proceedings of the 28th In-*

ternational Conference on Neural Information Processing Systems-Volume 1, pages 91–99, 2015. 8

- [25] Hamid Rezaatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 658–666, 2019. 8
- [26] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2*, pages 464–468, 2018. 2, 3, 7, 11
- [27] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pages 843–852, 2017. 4
- [28] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019. 6
- [29] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: Fully convolutional one-stage object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9627–9636, 2019. 8
- [30] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. *arXiv preprint arXiv:2012.12877*, 2020. 1, 2, 3, 4, 5, 6, 7, 8, 11
- [31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 6000–6010, 2017. 1, 2, 7, 10
- [32] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. *arXiv preprint arXiv:2102.12122*, 2021. 8
- [33] Yuqing Wang, Zhaoliang Xu, Xinlong Wang, Chunhua Shen, Baoshan Cheng, Hao Shen, and Huaxia Xia. End-to-end video instance segmentation with transformers. *arXiv preprint arXiv:2011.14503*, 2020. 2
- [34] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*, 2019. 3
- [35] Sixiao Zheng, Jiachen Lu, Hengshuang Zhao, Xiatian Zhu, Zekun Luo, Yabiao Wang, Yanwei Fu, Jianfeng Feng, Tao Xiang, Philip HS Torr, et al. Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers. *arXiv preprint arXiv:2012.15840*, 2020. 2
- [36] Xizhou Zhu, Weiye Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. In *International Conference on Learning Representations*, 2021. 1, 2, 8

## A. Preliminary Knowledge

Transformer [31] features a series of encoders and decoders of an identical structure. Every encoder and decoder has a *multi-head self-attention layer* (MHSA) and a *feed-forward network layer* (FFN), while each decoder has an extra attention layer to process the output of the encoder.

**Self-attention.** An attention function on input  $x = \{x_1, \dots, x_n\}$  is computed simultaneously on a set of queries  $Q$  with keys  $K$  and values  $V$  by the following,

$$\text{Att}(x) = \text{softmax}\left(\frac{Q \cdot K^\top}{\sqrt{d_k}}\right) \cdot V \quad (1)$$

$$Q = W^Q x, K = W^K x, V = W^V x$$

where  $W^Q$ ,  $W^K$ , and  $W^V$  are weight matrices to generate  $Q$ ,  $K$ , and  $V$  via linear transformations on  $x$ . And  $Q \cdot K^\top$  calculates attention score as the dot product of the query and all the keys, scaled by the dimension  $d_k$  of keys  $K$ .

**Multi-head Self-Attention.** First,  $Q$ ,  $K$ , and  $V$  are linearly projected for  $h$  times with different learned weights. Then the self-attention function is applied in parallel to generate  $h$  outputs, so-called *heads*. All heads are concatenated to give the final output, *i.e.*,

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h) W^O$$

where  $\text{head}_i = \text{Att}(QW_i^Q, KW_i^K, VW_i^V)$  (2)

**Feed-forward network.** The attention output is typically processed by a two-layer linear transformation with an activation in between,

$$\text{FFN}(x) = \max(0, W_1 x + b_1) W_2 + b_2 \quad (3)$$

**Layer Normalization.** A residual function and a layer normalization is added for each of the sub-layers (*e.g.*, attention layer and feed-forward layer) in every encoder and decoder, *i.e.*,  $\text{LayerNorm}(x + \text{Sublayer}(x))$ .

**Positional Encoding.** Transformer [31] adopts a sinusoidal function to encode positions, *e.g.*

$$\begin{aligned} PE(pos, 2i) &= \sin(pos/10000^{2i/d_{\text{model}}}) \\ PE(pos, 2i+1) &= \cos(pos/10000^{2i/d_{\text{model}}}) \end{aligned} \quad (4)$$

where  $pos$  denotes the word position in the sequence,  $d_{\text{model}}$  means the total encoding dimension, and  $i$  is the current dimension. However, such absolute positions can be more naturally encoded as relative positional encodings (RPE)

[26] like following,

$$\begin{aligned} \text{Att}(x_i) &= \sum_{j=1}^n \alpha_{ij} (W^V x_j + a_{ij}^V) \\ \alpha_{ij} &= \frac{\exp e_{ij}}{\sum_{k=1}^n \exp e_{ik}} \\ e_{ij} &= \frac{x_i W^Q (x_j W^K + a_{ij}^K)^T}{\sqrt{d_k}} \end{aligned} \quad (5)$$

where  $a_{ij} \in \mathbb{R}^{d_k}$  denotes edge distances between  $x_i$  and  $x_j$  when seeing input elements as a directed and fully-connected graph. RPE creates practical space complexity for implementation, which require more efficient version as in [?]. Moreover, it also requires substantial intrusion into standard Transformer API.

## B. Experiment Details

### B.1. Hyperparameter settings

Table 14 gives the hyper-parameter details of CPVT.

Methods	ViT-B [10]	DeiT-B [30]	CPVT
Epochs	300	300	300
Batch size	4096	1024	1024
Optimizer	AdamW	AdamW	AdamW
Learning rate decay	cosine	cosine	cosine
Weight decay	0.3	0.05	0.05
Warmup epochs	3.4	5	5
Label smoothing $\epsilon$ [?]	$\times$	0.1	0.1
Dropout [?]	0.1	$\times$	$\times$
Stoch. Depth [?]	$\times$	0.1	0.1
Repeated Aug [?]	$\times$	$\checkmark$	$\checkmark$
Gradient Clip.	$\checkmark$	$\times$	$\times$
Rand Augment [?]	$\times$	9/0.5	9/0.5
Mixup prob. [?]	$\times$	0.8	0.8
Cutmix prob. [?]	$\times$	1.0	1.0
Erasing prob. [?]	$\times$	0.25	0.25

Table 14. Hyper-parameters for ViT-B, DeiT-B and CPVT.

## C. Example Code

### C.1. PEG

In the simplest form, we use a single depth-wise convolution and show its usage in Transformer by the following PyTorch snippet. Through experiments, we find that such a simple design (*i.e.*, depth-wise  $3 \times 3$ ) readily achieves on par or even better performance than the recent SOTAs.

### C.2. PEG for Detection

Note that the masked padding should be carefully dealt with to avoid wrong gradient. The self attention component

---

### Algorithm 1 PyTorch snippet of PEG.

---

```
import torch
import torch.nn as nn
class VisionTransformer:
    def __init__(self, layers=12, dim=192, nhead=3, img_size=224,
        patch_size=16):
        self.pos_block = PEG(dim)
        self.blocks = nn.ModuleList([TransformerEncoderLayer(dim,
            nhead, dim*4) for _ in range(layers)])
        self.patch_embed = PatchEmbed(img_size, patch_size, dim
            *4)
    def forward_features(self, x):
        B, C, H, W = x.shape
        x, patch_size = self.patch_embed(x)
        _H, _W = H // patch_size, W // patch_size
        x = torch.cat((self.cls_tokens, x), dim=1)
        for i, blk in enumerate(self.blocks):
            x = blk(x)
        if i == 0:
            x = self.pos_block(x, _H, _W)
        return x[:, 0]
```

---

```
class PEG(nn.Module):
    def __init__(self, dim=256, k=3):
        self.proj = nn.Conv2d(dim, dim, k, 1, k//2, groups=dim)
        # Only for demo use, more complicated functions are
        # effective too.
    def forward(self, x, H, W):
        B, N, C = x.shape
        cls_token, feat_token = x[:, 0], x[:, 1:]
        cnn_feat = feat_token.transpose(1, 2).view(B, C, H, W)
        x = self.proj(cnn_feat) + cnn_feat
        x = x.flatten(2).transpose(1, 2)
        x = torch.cat((cls_token.unsqueeze(1), x), dim=1)
        return x
```

---



---

### Algorithm 2 PyTorch snippet of PEG for detection.

---

```
from torch import nn
class PEGDetection(nn.Module):
    def __init__(self, in_chans):
        super(PEGDetection, self).__init__()
        self.proj = nn.Sequential(nn.Conv2d(in_chans, in_chans,
            3, 1, 1, bias=False, groups=in_chans), nn.
            BatchNorm2d(in_chans), nn.ReLU())
    def forward(self, x, mask, H, W):
        """
        x N, B, C ; mask B N
        """
        _, B, C = x.shape
        _tmp = x.transpose(0, 1)[mask]
        x = x.permute(1, 2, 0).view(B, C, H, W)
        x = x + self.proj(cnn_feat)
        x = x.flatten(2).transpose(1, 2)
        x[mask] = _tmp
        return x.transpose(0, 1)
```

---

in most standard libraries supports masked tokens. PEG can efficiently attend to the masked padding by using some basic tensor operations as following.

## D. Under the Hood: Why is the Encoding Conditional?

We further study the underlying working mechanism of CPVT. Without loss of generalization, we ignore the batch dimension and use the model dim as 1. We denote the output sequence of the first encoder as  $X = (x_1, x_2, \dots, x_N)$

and  $N = H^2$ . We can write the convolution weight  $W$  as,

$$\begin{bmatrix} w_{-k,-k} & \cdots & w_{-k,0} & \cdots & w_{-k,k} \\ \vdots & & \vdots & & \vdots \\ w_{0,-k} & \cdots & w_{0,0} & \cdots & w_{0,k} \\ \vdots & & \vdots & & \vdots \\ w_{k,-k} & \cdots & w_{k,0} & \cdots & w_{k,k} \end{bmatrix}$$

We define the output of this convolution as  $Y = (y_1, y_2, \dots, y_N) = \mathcal{F}(X)$ . We define the mapping  $y_{m/H, m\%H} = y_m$  and  $x_{m/H, m\%H} = x_m$ . The transform function of  $X$  can be formulated as

$$y_m = x_m + \sum_{i=-k}^k \sum_{j=-k}^k x_{m/H+i, m\%H+j} w_{i,j} \quad (6)$$

For simplicity, we degrade the projection weight matrices to three scalars  $w_q, w_k, w_v$  and we ignore multi-heads. The self-attention function for  $z_m$  can then be written as

$$z_m = \sum_{n=1}^N \frac{e^{w_q w_k y_m y_n}}{\sum_{l=1}^N e^{w_q w_k y_m y_l}} w_v y_n \quad (7)$$

Substituting Eq 6 into Eq 7, we can derive that

$$\begin{aligned} y_m y_n &= (x_m + \sum_{i=-k}^k \sum_{j=-k}^k x_{m/H+i, m\%H+j} w_{i,j}) \\ &\quad \times (x_n + \sum_{p=-k}^k \sum_{q=-k}^k x_{n/H+p, n\%H+q} w_{p,q}) \\ &= x_m x_n + x_m \sum_{p=-k}^k \sum_{q=-k}^k x_{n/H+p, n\%H+q} w_{p,q} \\ &\quad + x_n \sum_{i=-k}^k \sum_{j=-k}^k x_{m/H+i, m\%H+j} w_{i,j} + \\ &\quad \sum_{i=-k}^k \sum_{j=-k}^k \sum_{p=-k}^k \sum_{q=-k}^k x_{m/H+i, m\%H+j} x_{n/H+p, n\%H+q} w_{p,q} w_{i,j} \end{aligned} \quad (8)$$

From the perspective of encoding, CPVT can be regarded as a *conditional encoding approach* if we consider the transformation function  $\mathcal{F}$  as a function to generate encodings.

Note that we add  $k$  zero paddings to make sure  $Y$  has the same length as  $X$ . This is reflected by variables in the boundary position such as  $x_{-1,-1}$  in Eq 8. This difference may bring absolute positional information.

If  $y_m$  and  $y_n$  are near to each other within the kernel range, there is a high probability that the same goes for the elements of  $X$ . The dot production (generalized in high dimension) will contribute to a positive attention score, which is weighted by the learnable  $w$ . This mechanism resembles

relative position encoding in that it processes relative information. Therefore, the conditional encoding can also be regarded as a *mixed encoding mechanism*.

## E. More Analysis

### E.1. Comparison to Lambda Networks

Our work is also related to Lambda Networks [?] which uses 2D relative positional encodings. We evaluate its lambda module with an embedding size of 128, where we denote its encoding scheme as RPE2D-d128. Noticeably, this configuration has about 5.9M parameters (comparable to DeiT-tiny) but only obtains 68.7%. We attribute its failure to the limited ability in capturing the correct positional information. After all, lambda layers are designed with the help of many CNN backbones components such as down-sampling to form various stages, to replace ordinary convolutions in ResNet [11]. In contrast, CPVT is transformer-based.

### E.2. Does each encoder need positional information?

We have shown that positional information is critical to visual transformers. A natural question is to know whether the position information is necessary for all the blocks. To verify this, we retain positional information within the first encoder and stop its forward propagation to the rest of the encoders. Specifically, we *only inject learnable positional encodings into the query and the keys of the first encoder* in the DeiT-tiny model. If blocks after the first one don't require such information, the performance should be on par with 72.2% top-1 accuracy on ImageNet. However, this group only obtains 71.4%.

## F. Figures

### F.1. DeiT

Figure 5 presents normalized attention maps from lower attention layers of DeiT where it learns locality information.

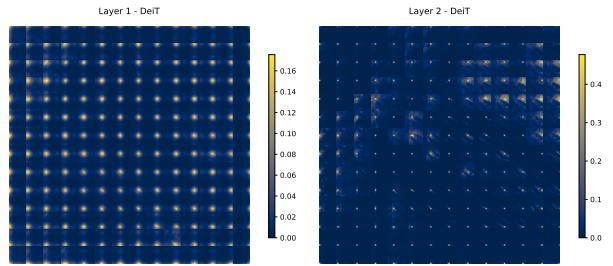


Figure 5. First and second layer attention map of DeiT (same input as PoVT in Figure 6 in main text ).



## F.2. CPVT

Figure 6 gives the normalized attention map of CPVT vs. DeiT w/o PE, reshaped to  $14 \times 14$  grids.

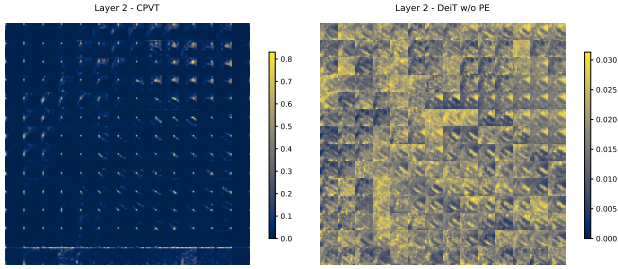


Figure 6. Normalized attention scores from the second encoder of CPVT vs. DeiT without position encodings (DeiT w/o PE), reshaped to  $14 \times 14$  grids. It corresponds to Figure 3 (main text).

Figure 7 shows the learned kernel weights for CPVT-Ti model.

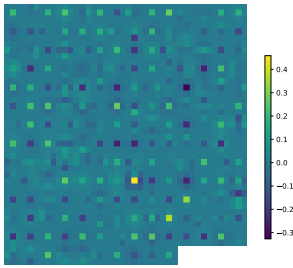


Figure 7.  $192 \times 3 \times 3$  PoVT plugin kernel weights. The center pixels generally have the most weight.

The schematics of plugin position ablation can be shown in the Figure 8.

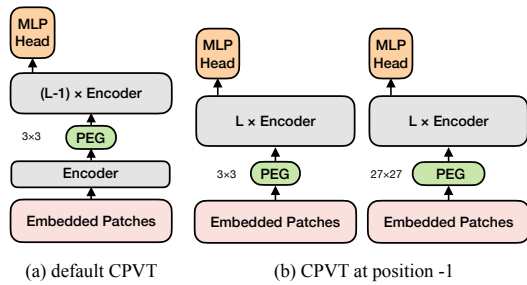


Figure 8. The same plugin at position -1 (b left) underperforms the default choice at position 0 (a). When replacing with a larger kernel, the problem is mitigated (b right) since it captures global information.