Do We Really Need Explicit Position Encodings for Vision Transformers?

Xiangxiang Chu[1], Bo Zhang[1], Zhi Tian[2], Xiaolin Wei[1], Huaxia Xia[1]
[1]Meituan Inc., [2]The University of Adelaide
{chuxiangxiang,zhangbo97,weixiaolin02,xiahuaxia}@meituan.com, zhi.tian@outlook.com

Abstract

Almost all visual transformers such as ViT [14] or DeiT [41] rely on predefined positional encodings to incorporate the order of each input token. These encodings are often implemented as learnable fixed-dimension vectors or sinusoidal functions of different frequencies, which are not possible to accommodate variable-length input sequences. This inevitably limits a wider application of transformers in vision, where many tasks require changing the input size on-the-fly.

In this paper, we propose to employ an implicit ***conditional position encodings*** *scheme, which is conditioned on the local neighborhood of the input token. It is effortlessly implemented as what we call Position Encoding Generator (PEG), which can be seamlessly incorporated into the current transformer framework. Our new model with PEG is named Conditional Position encodings Visual Transformer (CPVT) and can naturally process the input sequences of arbitrary length. We demonstrate that CPVT can result in visually similar attention maps and even better performance than those with predefined positional encodings. We obtain state-of-the-art results on the ImageNet classification task compared with visual Transformers to date. Our code will be made available at* https://github.com/Meituan-AutoML/CPVT .

1. Introduction

Recently, Transformer [42] has been viewed as a strong alternative to the convolutional neural networks (CNNs) in visual recognition tasks such as classification [14] and detection [5, 51]. Unlike the convolution operation in CNNs, which has a limited and fixed receptive field, the self-attention mechanism in the transformers can capture the long-distance information and dynamically adapts the receptive field according to the image content. As a result, the transformers are considered more flexible and powerful than CNNs, thus being promising.

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 gap, previous works [42, 14] add an absolute positional encoding 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 explicit 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 model parameters during training. Thus, the length of the positional encodings is fixed once trained. This makes the model difficult to handle the sequences with different lengths, particularly longer ones.

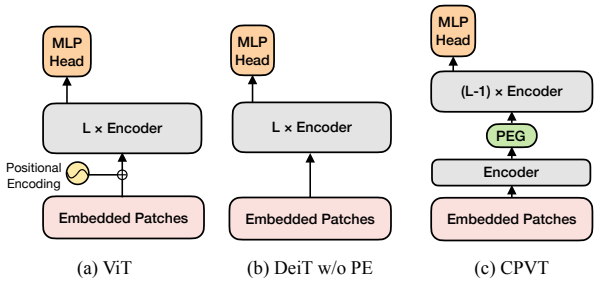


Figure 1: Visual Transformers: (a) with hardcoded positional encoding [14] (b) without positional encoding (c) with the proposed Position Encoding Generator (PEG) plugin. Attention in (c) automatically perceives position through learning.

Failing to adapt to a variable input length during testing greatly limits its application because many vision tasks (*e.g.*, object detection) require changing the image size (*i.e.*, input length in the transformer) on-the-fly. 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. One may also use relative positional encodings as in [35]. However, relative positional encodings not only

slow down the training and testing, but also requires modifying the implementation of the standard transformers. Last but not least, the relative positional encodings cannot work equally well as the absolute ones, as also later shown in our experiments. We conjecture that it is because the image recognition task still requires absolute position information [22].

In this work, we advocate a novel position encoding scheme to implicitly incorporate the position information into Transformer. Unlike the positional encodings used in previous works [14, 42, 35] which are predefined and input-agnostic, ours are generated on-the-fly and conditioned on the local neighborhood of an input token. We demonstrate that the Visual Transformer with this new encoding (*i.e.* CPVT, see Figure 1c) can produce visually similar attention maps and result in even better performance than the previous vision transformers [14, 41].

We summarize our contributions as follows.

- We propose a novel position encoding scheme, termed *conditional position encodings* (CPE), which conditionally imbues Transformer with implicit positional information on-the-fly. By doing so, Transformers are unlocked to process input images of arbitrary size without bicubic interpolation or fine-tuning.
- We demonstrate that positional encoding is crucial to vision transformers and it helps learn *a schema of locality*. We empirically prove CPE helps learn locality information as well.
- CPE is generated by so-called Positional Encoding Generators (PEG), whose implementation is effortless and doesn't require tampering much with the current Transformer API compared with [35]. It is also widely supported in mainstream deep learning frameworks [29, 1, 8].
- Our refurbished vision transformer with CPE is called Conditional Position encodings Visual Transformer (CPVT) and it achieves new state-of-the-art performance on ImageNet compared with prior arts [14, 41].

2. Related Work

2.1. Transformers in Language

Transformer [42] renovates language models after RNNs (typically in the form of GRU [9] and LSTM [16]), via an encoder-decoder architecture based on self-attention mechanism and feed-forward networks. Due to its parallel-friendly computation, Transformer and its variants like BERT [13] and GPT models [30, 31] can be pre-trained on very large datasets which allow outstanding knowledge transfer on many real-world language tasks. Remarkably, when pre-trained with an excessively large plaintext dataset, GPT-3 [4] even works on downstream tasks out-of-the-box without any fine-tuning.

2.2. Visual Transformers

Attention is shown to be able to replace convolution in vision tasks [32]. It turns out that self-attention layers attend pixel-grid patterns similarly to CNN layers [10]. Most recently, attention-based transformers have gone viral in vision.

Detection. DETR [5] applies Transformer for object detection, effectively removing the need of non-max suppression and anchor generation. Deformable DETR [51] involves sampling-based attention to speed up DETR. These Transformer detectors [5, 51] use CNNs for feature extraction, so the fixed-length position encodings are applied.

Classification. After reducing image resolution and color space, iGPT [7] trains Transformer on pixels so that it can later be finetuned for classification tasks. Closely related to our work, ViT [14] makes Transformers scalable for classification tasks. Specifically, it decomposes a 224×224 image into a series of 196 flattened patches (16×16 each), which are analogous to a sequence of words in language processing. DeiT [41] is noted for its data efficiency and can be directly trained on ImageNet, it largely eases the need for pre-training models on the excessively large dataset as in [14].

Segmentation. Transformer reassures its advantage in instance segmentation [45], panoptic segmentation [43] and semantic segmentation [49]. SETR [49] uses ViT encoders for feature extraction and appends a multi-level feature aggregation module for segmentation.

Low-level vision. Transformers are also proved powerful in many low-level vision tasks such as image generation [28]. IPT [6] profits from the multi-head feature to unify super-resolution, image denoising, and deraining within a single framework to develop an all-in-one model.

2.3. Local vs. Global Attention

Attention is a mechanism to give importance to the most relevant part of an input signal. Convolution operates both on spatial dimension and channel dimension within local receptive fields. To capture information with global receptive fields, convolutions are typically stacked with downsampling and non-linear modules. SENet [18] explicitly models channel-wise interdependencies with a *squeeze-and-excitation* module. CBAM [46] sequentially processes in channel and spatial dimension to generate attention map. Self-attention is used to draw global dependencies between input and output [42]. Similarly in non-local networks [44], a generalized form of self-attention called non-local operation is developed to capture long-range dependencies in time.

2.4. Positional Encodings

Positional encodings are crucial to exploiting the order of sequences. Convolution is found to implicitly encode absolute positions [23], especially zero padding and borders act as anchors to derive spatial information. However, the self-attention mechanism in Transformer does not explicitly model relative or absolute position information. To this end, Transformer [42] adopts explicit absolute sinusoidal positional encodings added into input embeddings. CoordConv [26] uses concatenation instead of addition. Relative position encoding [35] considers distances between sequence elements and proves to be beneficial. A 2D relative position encoding is proposed for image classification in [3], showing superiority to 2D sinusoidal embeddings. LambdaNetworks [2] proposes a lambda layer to model long-range content and position-based interactions, which bypasses the need for expensive quadratic attention maps. They find that positional interactions are necessary for performance while content-based interactions only give marginal improvement. These current positional encodings are either inflexible to implement or unadaptable for variable input lengths, which motivates us for a more thorough review.

3. Vision Transformer with Conditional Position Encodings

3.1. Preliminary Knowledge

Transformer [42] 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 $x = \{x_1, \ldots, x_n\}$ is computed simultaneously on a set of queries Q with keys K and values V by the following,

$$\begin{aligned} \mathrm{Att}(x) &= \mathrm{softmax}(\frac{Q \cdot K^\top}{\sqrt{d_k}}) \cdot V \\ Q = W^Q x, K &= W^K x, V = W^V x \end{aligned} \tag{1}$$

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.*,

$$\begin{aligned} \mathrm{MultiHead}(Q, K, V) &= \mathrm{Concat}(\mathrm{head}_1, ..., \mathrm{head}_\mathrm{h}) W^O \\ \text{where} \quad \mathrm{head}_\mathrm{i} &= \mathrm{Att}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{2}$$

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

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

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.*, $\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$.

Positional Encoding. Transformer [42] adopts a sinusoidal function to encode positions, e.g.

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

where pos denotes the word position in the sequence, d_model means the total encoding dimension, and i is the current dimension. However, such absolution positions can be more naturally encoded as relative positional encodings (RPE) [35] like following,

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

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 [19]. Moreover, it also requires substantial intrusion into standard Transformer API.

3.2. Motivation

Vision Transformers are constrained by fixed image sizes. Given an image of input size $H \times W$, it is flattened into patches with word size $S \times S$, the number of patches is determined as $N = \frac{HW}{S^2}$[1]. A pre-trained transformer is presumed to process images with the same size. Noticeably, all other components (*e.g.* MHSA and FFN) of a vision transformer can *scale well* with the spatial dimension *except for the position encoding* as the position is directly related to input size.

[1] S shall be divided by H and W.

There are three superficial approaches to break this constraint: removing positional encodings, using relative position encodings [35, 3] or interpolating absolute position encodings [41].

The first approach is simple but severely deteriorates the classification performance. This is as expected since the explicit positional encodings are meant to give the order of the input sequence [41, 14]. We measure the effect of disabling the position encodings and show the result in Table 1. Quite noticeably, DeiT-tiny's performance on ImageNet degrades from 72.2% to 68.2% when the learnable encoding is removed. This indicates that the position of small patches plays a very important role in visual perception. It also suggests that the learnable encoding is on par with the popular sinusoidal function encoding [42], consistent with [14, 41]. In a word, **position encodings are crucial to Vision Transformers.**

Model	Encoding	Top-1 Acc(%)	Top-5 Acc(%)
DeiT-tiny [41]	✗	68.2	88.7
DeiT-tiny [41]	learnable	72.2	91.0
DeiT-tiny [41]	sin-cos	72.3	91.0
DeiT-tiny	2D RPE [35]	70.5	90.0

Table 1: Comparison of various positional encoding (PE) strategies tested on ImageNet validation set. Removing PE greatly damages the performance.

The second one, relative position encodings [35], complicates the implementation and is less efficient, which requires substantial structural intrusions into standard Transformer functions (shown in Eq 5). Moreover, our controlled experiment shows that it behaves worse with 70.5% top-1 accuracy (Table 1), which indicates absolute position information is required for better performance.

The last one, interpolating the position encodings, is *expedient* for testing pre-trained vision Transformers for classification on higher-resolution images. However, under training scenarios in object detection and segmentation where input spatial dimensions change frequently, interpolation becomes infeasible.

Taking the above discussions into account, we are driven to think of a new strategy that **relaxes the limit on fixed input sizes** and meanwhile **imposes position encodings**.

3.3. Conditional Position Encodings

To handle different input sizes, the new position encodings need to have variable lengths according to input tokens. This leads to implicit encodings, *conditionally* generated to match input sizes on-the-fly.

Additionally, a successful design should meet the following **requirements**,

(1) retaining strong performance,

(2) avoiding *permutation equivariance* so that permuting input orders have different responses,

(3) efficient and easy to implement under modern deep learning frameworks, *non-invasive* to standard transformer API.

Before we proceed, we roughly review how to define absolute positions for a sequence of length N. One straightforward approach is to assign a position for each element directly. Another one is to define a reference point and to describe the relationship within a local neighborhood. When requested for the absolute position for a given element, we can reconstruct the relative relations from its neighbors. Combining the relative information with the reference point we can derive the actual position. As per visual transformers, we can use a similar mechanism as the latter approach to implicitly define the positions for input patches.

To build the relationship of local neighbors, we utilize the 2D characteristics of images and reshape the flattened sequence $X \in \mathbb{R}^{B\times N\times C}$ back to 2D space $X' \in \mathbb{R}^{B\times C\times H\times W}$. We then apply a 2D transformation $\mathcal{F}$ to impose local regularization on X' and reshape the output back to the sequence space, which we designate as $X'' \in \mathbb{R}^{B\times N\times C}$. Since the class token $Y \in \mathbb{R}^{B\times C}$ doesn't involve position information, we keep it unchanged. The output is formed by concatenating the unchanged class token Y and regularized X'' along the last dimension.

To construct the reference point, we utilize the boundary padding of 2D features. A toy example is that when we perform convolutions on an image or its feature maps, zero padding would indicate the position of the boundary point.

Positional Encoding Generator. There is a handy instantiation of $\mathcal{F}$ that meets requirements (2) and (3): a learnable 2D convolution with kernel k ($k \geq 3$) and $\frac{k-1}{2}$ zero paddings. It only adds a minimum touch of Transformer implementation. We call the overall module as a *Positional Encoding Generator* (PEG), visualized in Figure 2. It is supposed to capture 2D position information that feeds into the attention pipeline. By design, it supports flexible scaling for various input spatial dimensions. We also give an exemplary implementation in Section B.1 (supplementary). In practice, however, $\mathcal{F}$ can be more versatile like separable convolutions and many others.

We then propose Conditional Position encodings Visual Transformer (CPVT) that inserts PEGs between encoders. Next, we empirically study its potential functionality. We leave the requirement (1) to be verified in Section 4.2.

3.4. Position Encodings Help Learn A Schema of Locality

Knowing that position encodings are crucial to Transformers, we are curious to find out what it potentially does to contribute better performance and whether CPVT does

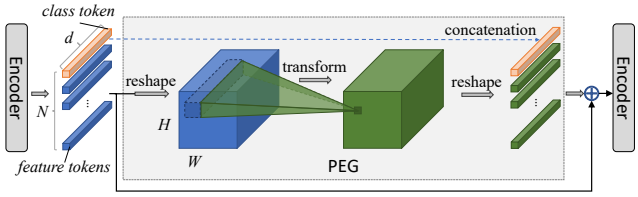


Figure 2: Schematic illustration of PEG. Note d is the embedding size, N is the number of tokens. The transformation unit can be depth-wise, separable convolution or other complicated blocks.

the same. We begin at comparing the failure case (w/o position encoding) with DeiT [41].

A good point to watch is the outcome. It has been shown [14] that lower layers in Transformer have shorter attention distances on average. As the network depth increases, the attention distances are also enlarged. We investigate whether the failure case without position encoding follows this observation.

Specifically, given a 224×224 image (i.e. 14×14 patches), the score matrix within a single head is 196×196. We calculate the normalized self-attention score matrix of the second encoder block and have it visualized in Figure 3. To the left it shows that DeiT learns to form **a schema of locality**, where *the diagonal element interacts strongly with its local neighbors while weakly with those far-away elements*. When position encoding is removed (DeiT w/o PE), it turns out that the patch develops much weaker interaction with its neighbors. This leads us to hypothesize that the failure might be caused by the weak attention to neighboring elements.

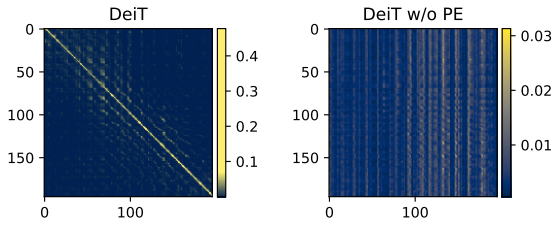


Figure 3: Normalized scores from the second encoder block of DeiT vs. without position encoding (DeiT w/o PE) [41] on the same input. With position encoding, DeiT develops a schema a locality in lower layers.

CPVT also learns locality information. It turns out that CPVT, just like DeiT [41], learns locality information in the lower layers too, as shown by the normalized attention map in Figure 4. It suggests that CPVT learns a similar pattern that DeiT managed to do under absolute encodings. Next, we evaluate whether this behavior helps for better performance through experiments.

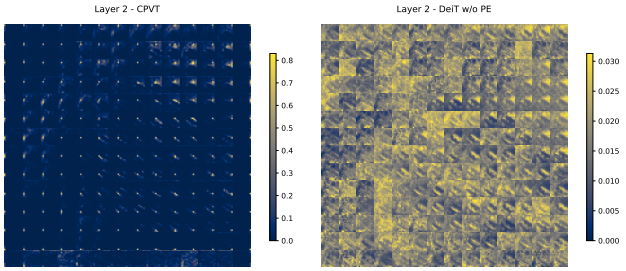


Figure 4: Comparison of the second layer attention maps reshaped to 14×14 grids) of CPVT vs. DeiT w/o PE.

4. Experiments

4.1. Setup

Dataset We use ILSVRC-2012 ImageNet dataset [12] with 1k classes and 1.3M images to train all our models following DeiT [41]. We report the results on the validation set throughout the paper. We don't use the much larger JFT-300M dataset [37], which is used in ViT [14] but undisclosed.

Model variants We directly comply with the model setting variants as [41] and use three models with different throughputs to adapt for various computing scenarios. The detailed setting is shown in Table 2. This strategy leads to controlled experiments to those recent SOTAs [14, 41]. 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 V100 with 8 GPU cards. Besides, CPVT-S and CPVT-B take about 1.6 and 2.5 days respectively. The added PEG plugin in CPVT models comes with neglectable costs, see Section 4.4.

Model	embedding dimension	#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 [14] and DeiT-B [41]. CPVT-S and CPVT-Ti have the same architecture as DeiT-small and DeiT-tiny respectively.

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 [27]. We don't tune the hyper-parameters and strictly comply with the settings in DeiT [41]. 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 [41]. The detailed hyperparameters are shown in Table 13 (supplementary).

4.2. 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 3. Compared with DeiT, *CPVT modes have much better top-1 accuracy with similar throughputs*. Since convolution is already efficiently implemented in popular deep learning frameworks, CPVT is quite efficient and has good throughput performance.

Models	Params.	Input	throughput* (img/s)	Top-1 Acc (%) @input	Top-1 Acc (%) @384*
Convnets					
ResNet-50 [15]	25M	224^2	1226.1	79.0	78.6
EfficientNet-B0 [39]	5M	224^2	2694.3	77.1	77.3
EfficientNet-B1 [39]	8M	240^2	1662.5	79.1	80.0
EfficientNet-B2 [39]	9M	260^2	1255.7	80.1	81.0
EfficientNet-B3 [39]	12M	300^2	732.1	81.6	82.1
EfficientNet-B4 [39]	19M	380^2	349.4	82.9	82.9
EfficientNet-B5 [39]	30M	456^2	169.1	83.6	82.7
EfficientNet-B6 [39]	43M	528^2	96.9	84.0	83.0
EfficientNet-B7 [39]	66M	600^2	55.1	84.3	83.1
Transformers					
ViT-B/16 [14]	86M	384^2	85.9	77.9	77.9
ViT-L/16 [14]	307M	384^2	27.3	76.5	76.5
DeiT-tiny w/o PE [41]	6M	224^2	2536.5	68.2	68.6
DeiT-tiny [41]	6M	224^2	2536.5	72.2	71.2
CPVT-Ti (0-0) 4 PEGs †	6M	224^2	2512.5	72.9	73.4
CPVT-Ti (0-5)‡	6M	224^2	2500.7	**73.4**	**74.2**
DeiT-small [41]	22M	224^2	940.4	79.9	78.1
CPVT-S (0-5)‡	23M	224^2	930.5	**80.5**	**80.8**
DeiT-base [41]	86M	224^2	292.3	81.8	79.7
CPVT-B	86M	224^2	290.1	**81.9**	**82.3**

⋆: The throughput is measured on one 16GB V100 GPU as in [41].
* : Directly tested on 384×384 without fine-tuning.
†: Insert 4 PEGs after the first encoder
‡: Insert one PEG each after the first encoder till the fifth encoder

Table 3: Comparison with ConvNets and Transformers on ImageNet. CPVT models have much better performance compared with prior Transformers, and also benefit from direct scaling without fine-tuning while DeiT degrades.

Direct scaling to a higher resolution. The most popular method for scaling pre-trained visual Transformer models to process higher resolution images is through simple interpolation of position encodings [14, 41]. However, this interpolation potentially damages the performance. To verify this, we took all the models firstly trained using the input images of 224×224 and tested based on a higher resolution of 384×384 without fine-tuning. The result is shown in the right-most column in Table 3. DeiT-tiny degrades from 72.2% to 71.2%. In contrast, as we removed explicit position encodings, CPVT model can directly process arbitrary size of images. As a result, CPVT-Ti's performance is boosted from 73.4% to 74.2%. Therefore, the gap between DeiT-tiny and CPVT-Ti enlarged to 3.0%.

The scaling result is overall promising since CPVT doesn't require any extra endeavors to achieve good scalability. In other words, *it solely relies on the neural network itself to generalize to different spatial inputs*. This result resonates with the motivation of the conditional position encodings scheme.

4.3. Object Detection

We further evaluate the detection performance of our method on the COCO [25] dataset. Specifically, we only change the positional encoding strategy of the encoder part. We keep almost the same setting as [5] except that we shorten the training epochs from 500 to 50 since it takes about 2000 GPU hours to train DETR for 500 epochs [5, 51]. We hope this setting will help the community to form a resource-efficient baseline. Specifically, we train DETR using AdamW [27] with a total batch size of 32 and 0.0001 weight decay. The initial learning rate of the backbone and transformer is 2×10^{-5} and 1×10^{-4} respectively. The learning rate is scheduled following the stepLR strategy and decay by 0.1 × at epoch 40. As for the loss function, there are three components: l_1 loss for bounding box (5.0), classification loss (1.0) and GIoU loss (2.0) [34].

For DETR models, we use 100 object queries and don't utilize Focal loss [24] to make fair comparison. Note that both DETR and Deformable-DETR make use of 2D sine positional encoding. Table 4 shows that if this encoding is removed, the mAP of DETR degrades to 32.8% from 33.7%, which is consistent with the ablation study of DETR. However, its performance can be improved to 33.9% if PEG is plugged in. As for Deformable-DETR, replacing 2D since positional encoding with PEG obtains better performance under various settings. In summary, PEG helps learn-able positional embedding outperform its human designed counterpart in the task of object detection.

Method	Epochs	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	params	FPS
FCOS [40]	36	41.0	59.8	44.1	26.2	44.6	52.2	-	23
Faster R-CNN [33]	109	42.0	62.1	45.5	26.6	45.4	53.4	42M	26
DETR [41]	500	42.0	62.4	44.2	20.5	45.8	61.1	41M	28
DETR-DC5	500	43.3	63.1	45.9	22.5	47.3	61.1	41M	12
DETR w/o PE⋆	50	32.8	54.0	33.5	13.7	35.5	50.5	41M	28
DETR⋆	50	33.7	54.5	34.7	13.2	35.8	51.5	41M	28
DETR w/o PE + PEG	50	**33.9**	**54.9**	**34.7**	**13.5**	**36.7**	**52.2**	41M	28
DD [51]	50	39.4	-	-	20.6	43.0	55.0	34M	27
DD w/o PE + PEG	50	**39.7**	60.1	42.3	20.3	**43.8**	**56.0**	34M	27
DETR-DC5	50	35.3	55.7	36.8	15.2	37.5	53.6	41M	12
DD-DC5 [51]	50	41.5	61.5	44.8	24.1	45.3	56.0	34M	22
DD-DC5 w/o PE + PEG	50	**41.9**	**61.8**	**45.1**	24.1	**45.9**	**57.0**	34M	22

[1] DD: Deformable DETR [51]. For DD, we always use single scale.
⋆ reproduced results using the released code.

Table 4: Comparison on COCO 2017 val set. PEG is a strong alternative to 2D sine positional encoding.

We give an example snippet of our implementation in Section B.2 (supplementary).

4.4. Complexity Analysis of PEG in CPVT Models

Neglectable Parameters. Given the model dimension d, the extra number of parameters introduce by PEG is

dlk^2 if we choose l depth-wise convolutions with kernel k. Even if we use l separable convolutions, this value becomes $l(d^2 + k^2 d)$. When $k = 3$ and $l = 1$, CPVT-Ti ($d = 192$) brings about 1728 parameters. Note that DeiT-tiny utilizes learnable positional embedding 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 $38952 - 37632 = 960$ more parameters. Note that DeiT-tiny has 5.7M parameters, therefore the added cost can be neglected.

Neglectable FLOPs. As per FLOPs, l layers of $k \times k$ depth-wise convolutions possesses the cost of $14 \times 14 dlk^2$. Taking the tiny model for example, it involves $196 \times 192 \times 9 = 0.34M$ for the simple case $k = 3, l = 1$, which can be neglected considering the tiny model has 2.1G FLOPs.

5. Ablation Study

Regarding highly expensive computing cost (see Section 4.1), we use the tiny model (by default, only one PEG of depth-wise 3×3 is used) for most of the ablations. If otherwise specified, all experiments are done on ImageNet and use exactly the same hyper-parameter settings as Section 4.1.

5.1. Positional Encoding Types

Previously, there were several types of commonly used encodings: absolute positional encoding (e.g. sinusoidal [42]), *relative positional encoding* (RPE) [35] and *learnable encoding* (LE) [13, 30]. These encodings are added to the input patch tokens before the first encoder block. We first compare the proposed PEG with these strategies in Table 5. Notice we denote i-j as the inserted positions of PEG which start from i and end at j-1.

Model	PEG Pos	Encoding	Top-1(%)	Top-5 (%)
DeiT-tiny [41]	-	LE	72.2	91.0
DeiT-tiny [41]	-	2D sin-cos	72.3	91.0
DeiT-tiny	-	2D RPE	70.5	90.0
CPVT-Ti	0-0	PEG	72.4	91.2
CPVT-Ti	0-0	PEG + LE	72.9	91.4
CPVT-Ti	0-0	4×PEG + LE	72.9	91.4
CPVT-Ti	0-5	PEG	**73.4**	**91.8**

Table 5: Comparison of various encoding strategies. LE: learnable encoding. RPE: relative positional encoding

DeiT-tiny obtains 72.2% with learnable absolute encoding. We also extend the sinusoidal encoding in Equation 4 into 2D space to achieve on par performance. As for RPE, we follow [35] and set the local range hyper-parameter K as 8. This requires changing the self-attention formulation in Equation 5 and we obtain 70.5% top-1 accuracy.

Moreover, we combine the learnable absolute encoding with a single PEG. This boosts the baseline CPVT-Ti (0-0) by 0.5%[2]. We attribute it to the limited representative capacity of a single PEG, because if we use stack 4 PEG layers it can achieve 72.9% and match the performance. Moreover, if we add a single layer of PEG to the first five blocks, we can obtain 73.4% top-1 accuracy, which indicates that features at different transformer blocks may have different optimal positions.

5.2. Plugin Positions

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 first encoder by index -1.* Therefore, position 0 is the output of the first encoder block. Our method shows strong performance (∼72.4%) when PEG placed at [0, 3].

Position Idx	Top-1 Acc(%)	Top-5 Acc(%)
none	68.2	88.7
-1	70.6	90.2
0	**72.4**	**91.2**
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.

5.3. Large Kernel at Position -1

Why is placing PEG at position -1 much worse than that at 0? (Table 6). We observe that the largest 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 range*. To test 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 72.5% top-1 accuracy, which verifies our assumption and thus can be regarded as another variant of CPVT. We don't use it by default because it is a little slower and worse than placing PEG at 0.

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

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

5.4. Single Position vs. Multiple Positions

We further evaluate whether using *multi-position* encodings benefits the performance in Table 8. We follow the notations in Section 5.1. By inserting five positions,

[2]This setting cannot scale with spatial dimensions of input

the top-1 accuracy of the tiny model is further improved to 73.4%, which surpasses the recent SOTA DeiT-tiny by 1.2%. Similarly, CPVT-S achieves a new state-of-the-art accuracy (80.5%).

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

Table 8: CPVT's performance sensitivity to number of plugin positions on ImageNet validation dataset.

5.5. Kernel Sizes

We further evaluate the performance of various *regularization ranges* determined by PEG kernel sizes. Specifically, we utilize a single depth-wise layer without batch normalization [21] or activation. We fix the plugin position at 0. Table 9 shows kernel 1×1 performs badly (68.9% top-1 accuracy). We attribute its failure to *the loss of positional information*. Moreover, there are no extra gains with larger kernel sizes, e.g. 5 and 7.

Kernel	Params(M)	Top-1 Acc(%)	Top-5 Acc(%)
1	5.68	68.9	89.3
3	5.68	**72.4**	**91.2**
5	5.68	72.0	90.9
7	5.69	72.2	91.1

Table 9: Performance of different regularization ranges.

5.6. Padding Strategies

We design an experiment to quantify the importance of the absolute positional encoding from *zero paddings*. Specifically, we use CPVT-S and simply remove such paddings from CPVT while keeping all other parts unchanged. This shortens the input sequence length from 196 to 144. Table 10 shows that the tiny model only obtains 70.5% if the padding is removed, 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. Transformer has to know which patch is in the center.

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

Table 10: ImageNet Performance w.r.t padding strategies.

5.7. What Matters the most for Performance?

Representational power? One might suspect that the performance improvement mainly comes from the *representational power* introduced by PEG. To disprove it, we simply use a weak PEG: a 3×3 depth-wise convolution with random initialization. We also fix its weight during the training, in which case we obtain 71.3% top-1 accuracy on ImageNet (see Table 11), way higher (3.1%↑) than one without any encoding (68.2%). Representational power alone would not make such a big difference. As a comparison, when we add 12 more fully connected layers (*i.e.*, kernel size 1) with skip connections after the first encoder, it brings about 0.5M more parameters. However, this only boosts the performance to 68.6% (0.4%↑).

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	**72.4**

Table 11: Ablation to decide the impact factor.

Positional Encoding? Then, why can a single non-learnable depth-wise convolution make such a big difference? We have seen that fixed weight has slightly weaker performance (71.3%) compared with the learnable baseline (72.4%). This suggests that the positional constraints imposed by the fixed-weight PEG is attenuated. The network may have more trouble in adapting its weights since the encoding is not adaptive, i.e. less accurate.

To prove the above hypothesis, we fix a learned PEG instead of one with random initialization and train the tiny version of the model from scratch. It achieves 72.3% top-1 accuracy on ImageNet. Compared with the learnable baseline, the learned PEG already conveys most of the positional information.

Based on these aspects, *we can conclude that it is the position encoding that matters the most*. In the next section, we design various encoding functions for a better analysis.

5.8. Efficiency vs. Complexity

Table 12 shows the performance of PEG choices. We insert various types of PEGs at position 0. Depth-wise convolution acts as a baseline with 72.4% accuracy. Direct replacement for depth-wise convolution using a separable one can marginally boost the performance with 0.1%. Stacking three more layers of PEG (adding 0.1M more parameters) achieves 72.9%. We attribute its success to *better encoding from more powerful representational capacity*.

Stack Number	Type	Params(M)	Top-1 Acc(%)
1×	Depth-wise	5.7	72.4
1×	Separable	5.7	72.5
4×	Separable	5.8	**72.9**

Table 12: Ablation on the performance sensitivity to representation capacity. PEGs are at position 0.

6. Conclusion

In this paper, we've introduced CPVT, a novel method to coerce position perception capability in vision Transformers. Through both theoretical and extensive experimental studies, we systematically analyze how important positional encoding is to vision Transformers. We discover that position information is crucial but does not necessarily have to be explicitly specified. A new out-of-box plugin is proposed to replace explicit hardcoded position encodings, which leads to strong performance with negligible extra cost. With the gifted freedom of changing the input size on-the-fly and plug-and-play nature, we look forward to a broader application of our proposed method in transformer-driven vision tasks like segmentation and video processing, or even in natural language 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. Lambdanetworks: Modeling long-range interactions without attention. In *International Conference on Learning Representations*, 2021. 3, 13

[3] 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, 4

[4] 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

[5] 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, 6

[6] 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

[7] Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. Generative pre-training from pixels. In *International Conference on Machine Learning*, pages 1691–1703. PMLR, 2020. 2

[8] 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

[9] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014. 2

[10] 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

[11] Ekin Dogus Cubuk, Barret Zoph, Jon Shlens, and Quoc Le. Randaugment: Practical automated data augmentation with a reduced search space. *Advances in Neural Information Processing Systems*, 33, 2020. 12

[12] 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. 5

[13] 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

[14] 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, 5, 6, 12

[15] 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, 13

[16] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 2

[17] Elad Hoffer, Tal Ben-Nun, Itay Hubara, Niv Giladi, Torsten Hoefler, and Daniel Soudry. Augment your batch: Improving generalization through instance repetition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8129–8138, 2020. 12

[18] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018. 2

[19] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M Dai, Matthew D Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer. In *Advances in Neural Processing Systems*, 2018. 3

[20] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In

European conference on computer vision, pages 646–661. Springer, 2016. 12

[21] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015. 8

[22] 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

[23] Md Amirul Islam, Matthew Kowal, Sen Jia, Konstantinos G Derpanis, and Neil DB Bruce. Position, padding and predictions: A deeper look at position information in cnns. *arXiv preprint arXiv:2101.12322*, 2021. 3

[24] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017. 6

[25] 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. 6

[26] Rosanne Liu, Joel Lehman, Piero Molino, Felipe Petroski Such, Eric Frank, Alex Sergeev, and Jason Yosinski. An intriguing failing of convolutional neural networks and the coordconv solution. In *Advances in Neural Information Processing Systems*, page 9628–9639, 2018. 3

[27] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. 5, 6

[28] 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

[29] 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

[30] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018. 2, 7

[31] 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

[32] 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

[33] 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 International Conference on Neural Information Processing Systems-Volume 1*, pages 91–99, 2015. 6

[34] Hamid Rezatofighi, 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. 6

[35] 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. 1, 2, 3, 4, 7

[36] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014. 12

[37] 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. 5

[38] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016. 12

[39] 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

[40] 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. 6

[41] 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, 4, 5, 6, 7, 12

[42] 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, 3, 4, 7

[43] Huiyu Wang, Yukun Zhu, Hartwig Adam, Alan Yuille, and Liang-Chieh Chen. Max-deeplab: End-to-end panoptic segmentation with mask transformers. *arXiv preprint arXiv:2012.00759*, 2020. 2

[44] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7794–7803, 2018. 2

[45] 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

[46] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018. 2

[47] 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 *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6023–6032, 2019. 12

[48] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018. 12

[49] 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

[50] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 13001–13008, 2020. 12

[51] Xizhou Zhu, Weijie 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, 6

A. Experiment Details

A.1. Hyperparameter settings

Table 13 gives the hyper-parameter details of CPVT.

Methods	ViT-B [14]	DeiT-B [41]	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 ε [38]	✗	0.1	0.1
Dropout [36]	0.1	✗	✗
Stoch. Depth [20]	✗	0.1	0.1
Repeated Aug [17]	✗	✓	✓
Gradient Clip.	✓	✗	✗
Rand Augment [11]	✗	9/0.5	9/0.5
Mixup prob. [48]	✗	0.8	0.8
Cutmix prob. [47]	✗	1.0	1.0
Erasing prob. [50]	✗	0.25	0.25

Table 13: Hyper-parameters for ViT-B, DeiT-B and CPVT.

B. Example Code

B.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 3×3) readily achieves on par or even better performance than the recent SOTAs.

B.2. PEG for Detection

Note that the masked padding should be carefully dealt with to avoid wrong gradient. The self attention component in most standard libraries supports masked tokens. PEG can efficiently attend to the masked padding by using some basic tensor operations as following.

C. 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 $X = (x_1, x_2, ..., x_N)$ and $N = H^2$. We can write the convolution weight W as,

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

Algorithm 1 PyTorch snippet of PEG.

```
import torch
import torch.nn as nn
class VisionTransformer:
  def __init__(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)
```

We define the output of this convolution as $Y = (y_1, y_2, ..., y_N) = \mathcal{F}(X)$. 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

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

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

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

Substituting Eq 6 into Eq 7, we can derive that

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

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*.

D. More Analysis

D.1. Comparison to Lambda Networks

Our work is also related to Lambda Networks [2] 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 downsampling to form various stages, to replace ordinary convolutions in ResNet [15]. In contrast, CPVT is transformer-based.

D.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%.

E. Figures

E.1. DeIT

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

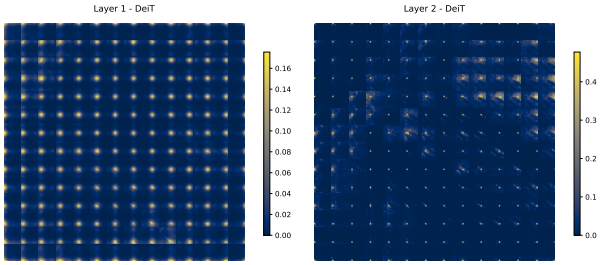


Figure 5: First and second layer attention map of DeIT (same input as PoVT in Figure 4 in main text).

E.2. CPVT

Figure 6 gives the normalized attention map of CPVT vs. DeiT w/o PE.

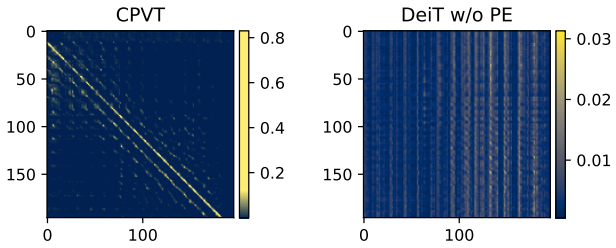


Figure 6: Normalized scores from the second encoder block (196×196) of CPVT vs. DeiT w/o PE.

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

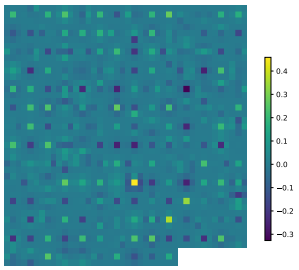
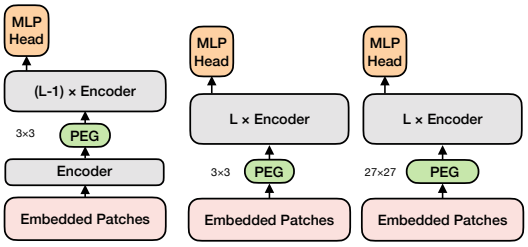


Figure 7: 192 3×3 PoVT plugin kernel weights. The center pixels generally have the most weight.

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



(a) default CPVT (b) CPVT at position -1

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.