

ViT (Vision Transformer)

Published as a conference paper at ICLR 2021

AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE

**Alexey Dosovitskiy^{*,†}, Lucas Beyer^{*}, Alexander Kolesnikov^{*}, Dirk Weissenborn^{*},
Xiaohua Zhai^{*}, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer,
Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby^{*,†}**

^{*}equal technical contribution, [†]equal advising
Google Research, Brain Team
{adosovitskiy, neilhoulbsby}@google.com

ABSTRACT

While the Transformer architecture has become the de-facto standard for natural language processing tasks, its applications to computer vision remain limited. In vision, attention is either applied in conjunction with convolutional networks, or used to replace certain components of convolutional networks while keeping their overall structure in place. We show that this reliance on CNNs is not necessary and a pure transformer applied directly to sequences of image patches can perform very well on image classification tasks. When pre-trained on large amounts of data and transferred to multiple mid-sized or small image recognition benchmarks (ImageNet, CIFAR-100, VTAB, etc.), Vision Transformer (ViT) attains excellent results compared to state-of-the-art convolutional networks while requiring substantially fewer computational resources to train¹

<https://arxiv.org/abs/2010.11929>

ViT (Vision Transformer)

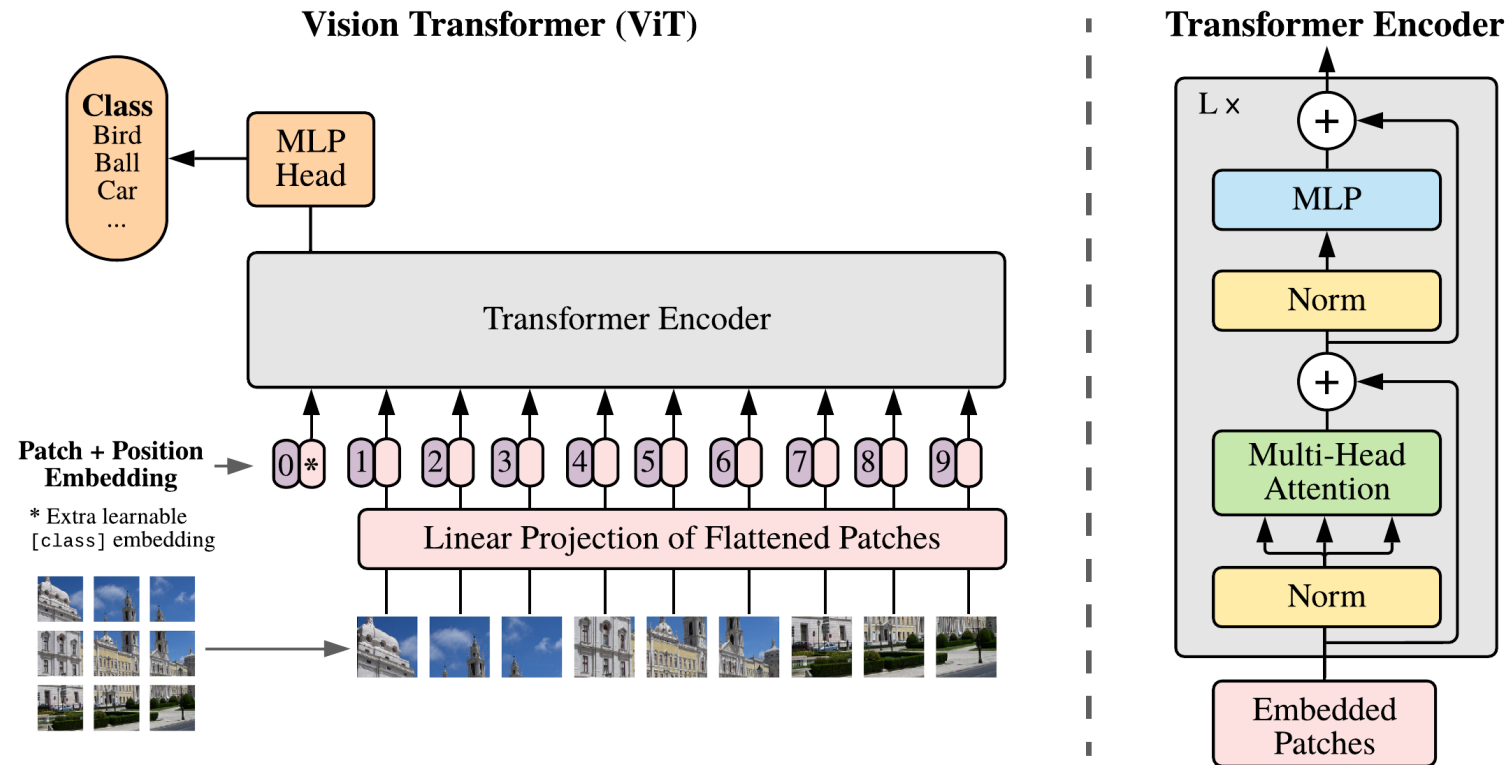
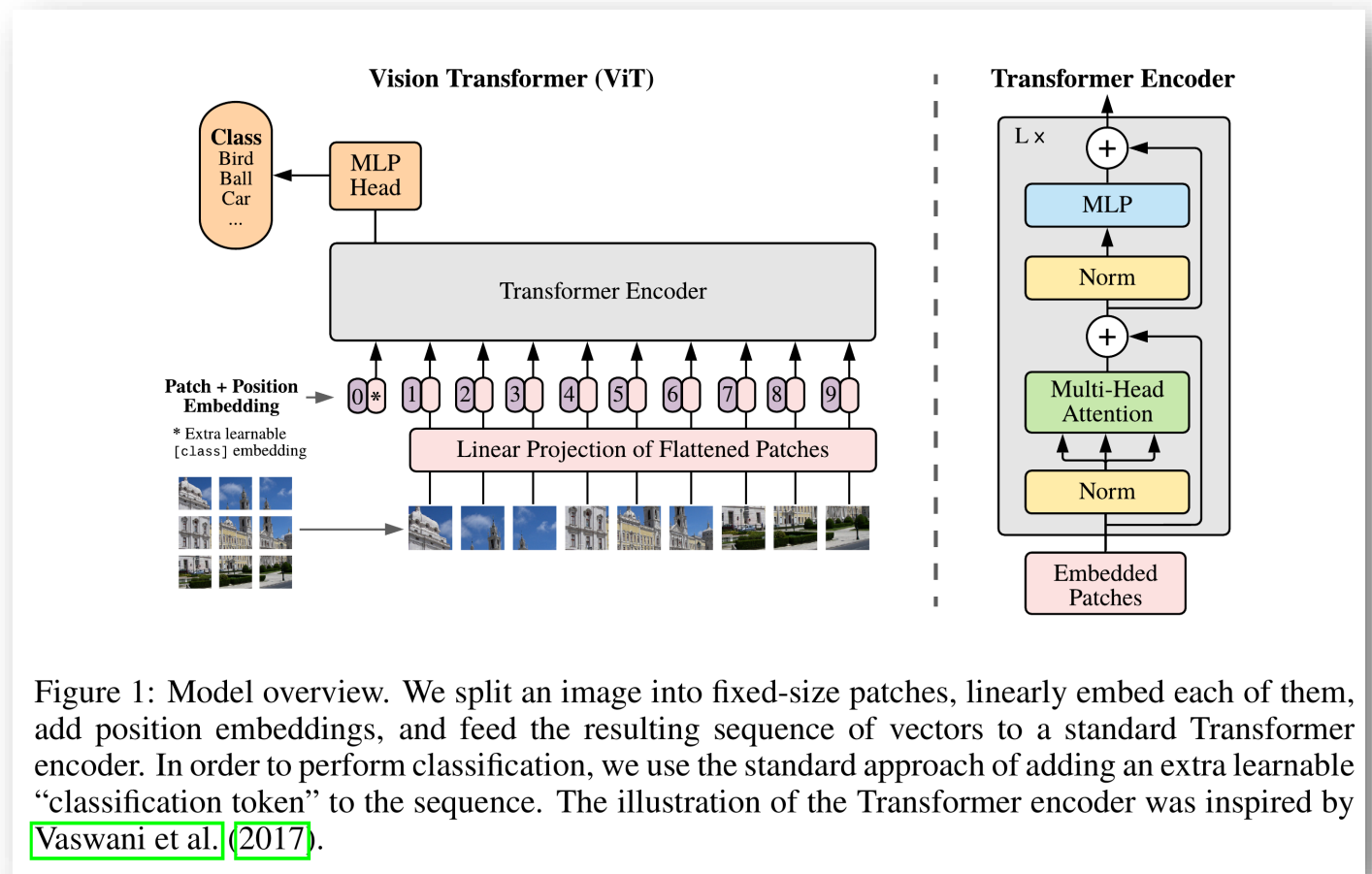


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable “classification token” to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

ViT (Vision Transformer)

High Level Overview

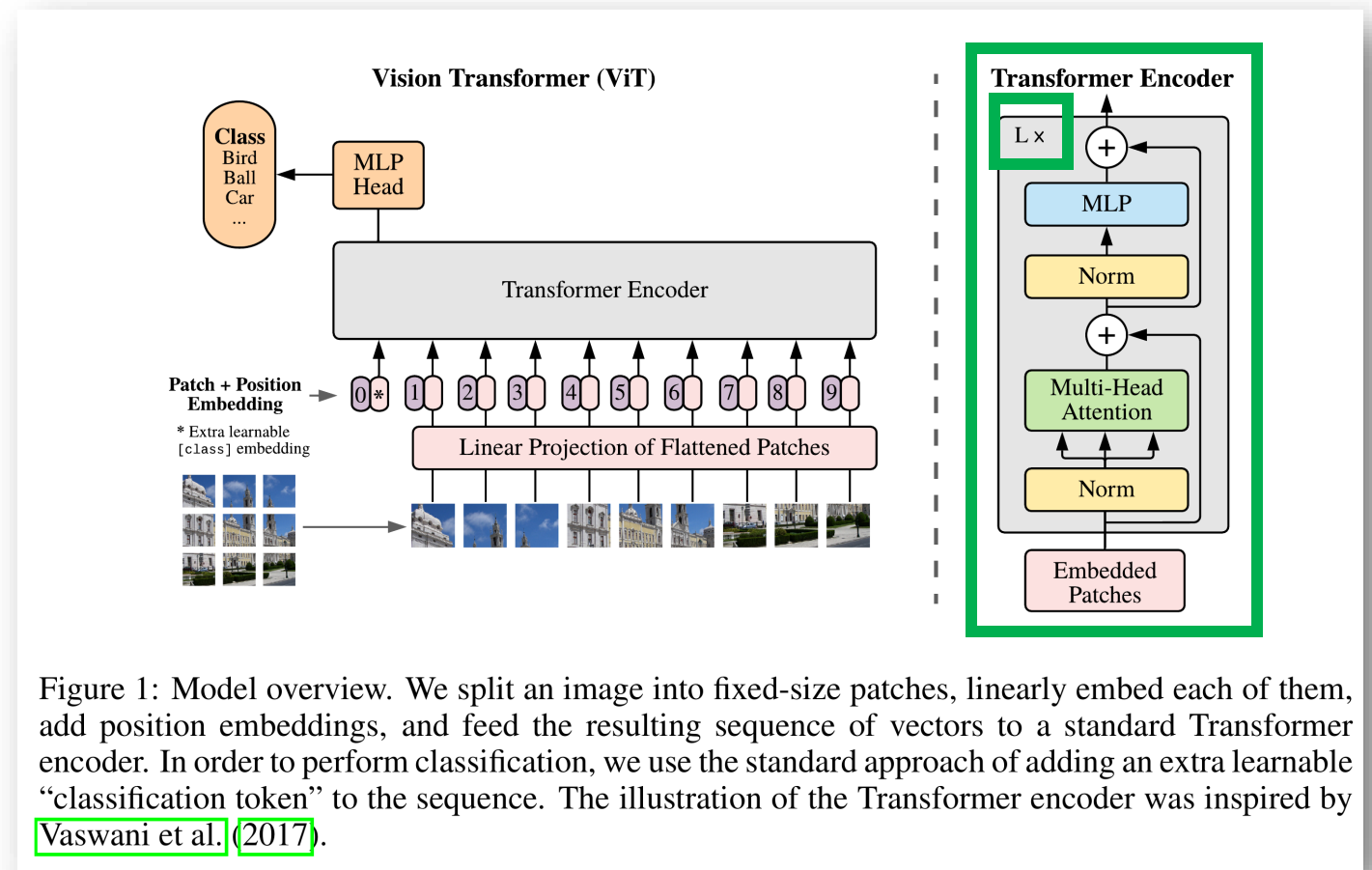
- in general (not only for ViT) we create a network from blocks that consists of several layers



ViT (Vision Transformer)

High Level Overview

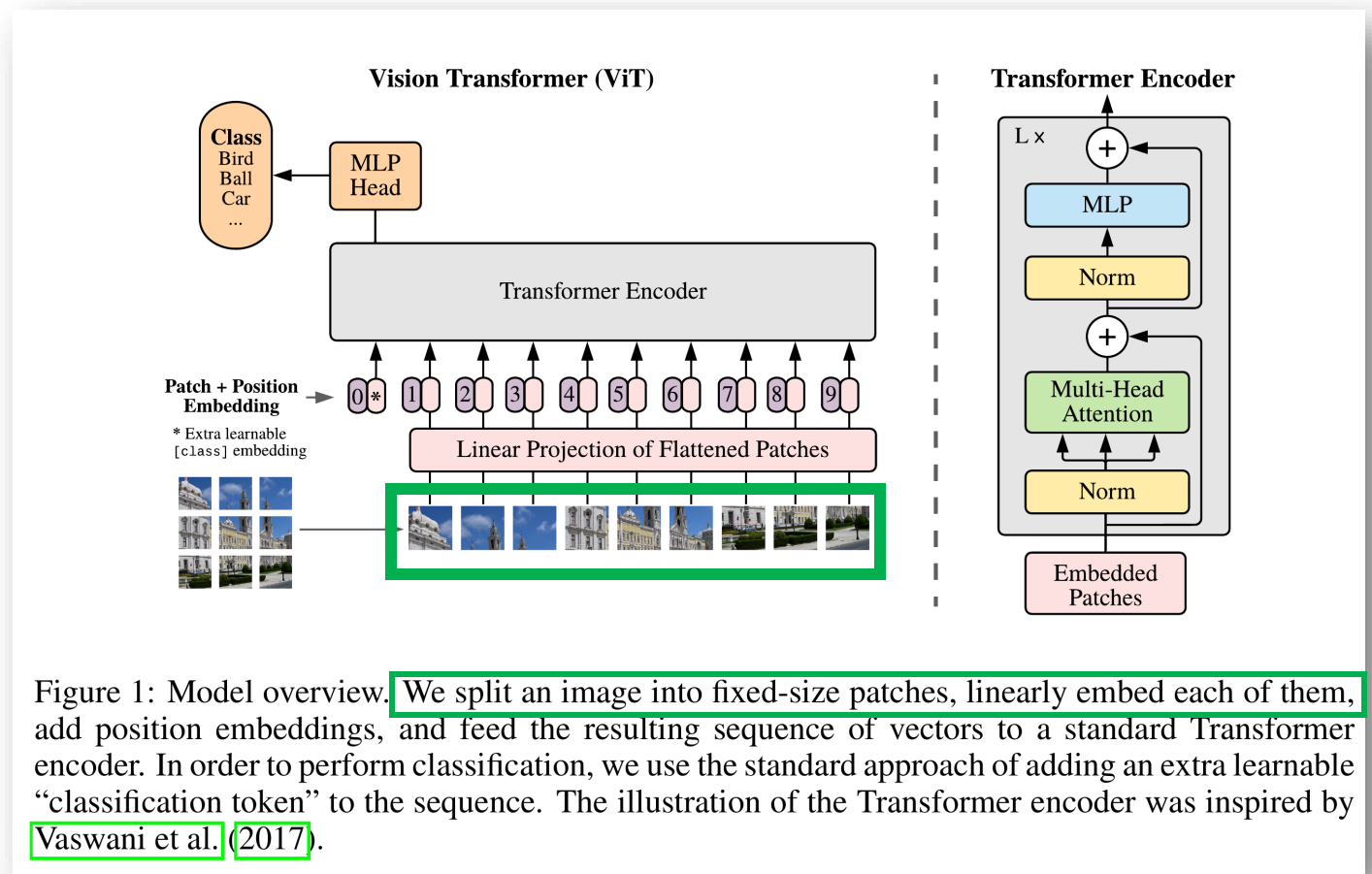
- in general (not only for ViT) we create a network from blocks that consists of several layers
- e.g. inception blocks, residual blocks, **transformer encoder blocks**
- what is input for these models (blocks)?



ViT (Vision Transformer)

High Level Overview

- in general (not only for ViT) we create a network from blocks that consists of several layers
- e.g. inception blocks, residual blocks, **transformer encoder blocks**
- what is input for these models (blocks)?
- In the case of CNN, we use raw images
- In the case of **ViT**, we use **fixed size patches**



ViT (Vision Transformer)

Step 1 - Input

- transform image into 16 x 16 size (patches)
- embed each patch into 768 dimension
- i.e. one patch can be described with 1 x 768 values
- In the case that we have 196 patches with size of 16 x 16, we obtain [14, 14, 768] tensor
- with the use of flatten, we obtain [196, 768] matrix

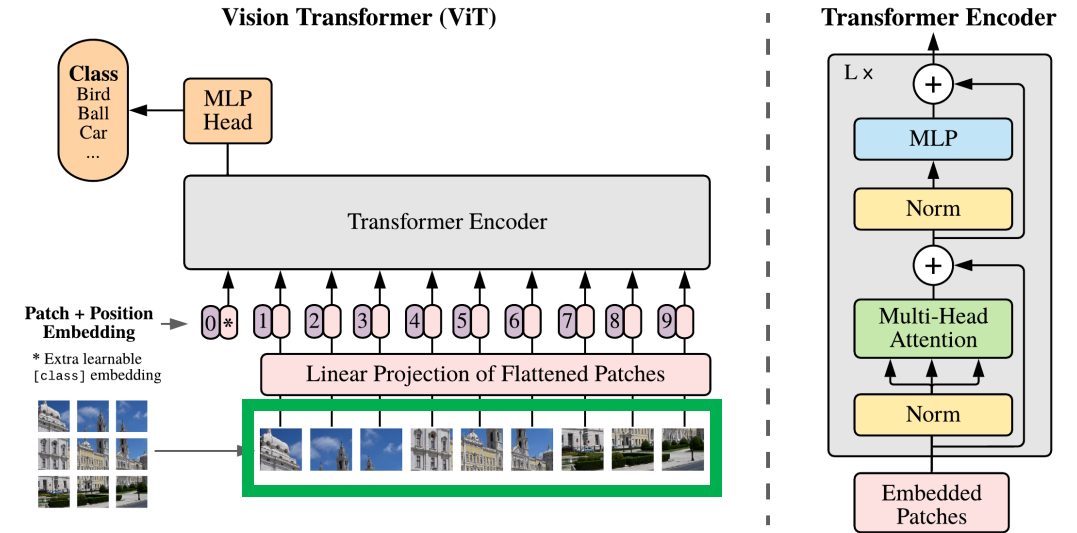


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable “classification token” to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

An overview of the model is depicted in Figure 1. The standard Transformer receives as input a 1D sequence of token embeddings. To handle 2D images, we reshape the image $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ into a sequence of flattened 2D patches $\mathbf{x}_p \in \mathbb{R}^{N \times (P^2 \cdot C)}$, where (H, W) is the resolution of the original image, C is the number of channels, (P, P) is the resolution of each image patch, and $N = HW/P^2$ is the resulting number of patches, which also serves as the effective input sequence length for the Transformer. The Transformer uses constant latent vector size D through all of its layers, so we flatten the patches and map to D dimensions with a trainable linear projection (Eq. 1). We refer to the output of this projection as the patch embeddings.

ViT (Vision Transformer)

Step 1 - Input

- transform image into 16 x 16 size (patches)
- embed each patch into 768 dimensions
- i.e. one patch can be described by 1 x 768 values
- In the case that we have 196 patches with size of 16 x 16, we obtain [14, 14, 768] tensor
- with the use of flatten, we obtain [196, 768] matrix

Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Table 1: Details of Vision Transformer model variants.

An overview of the model is depicted in Figure 1. The standard Transformer receives as input a 1D sequence of token embeddings. To handle 2D images, we reshape the image $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ into a sequence of flattened 2D patches $\mathbf{x}_p \in \mathbb{R}^{N \times (P^2 \cdot C)}$, where (H, W) is the resolution of the original image, C is the number of channels, (P, P) is the resolution of each image patch, and $N = HW/P^2$ is the resulting number of patches, which also serves as the effective input sequence length for the Transformer. The Transformer uses constant latent vector size D through all of its layers, so we flatten the patches and map to D dimensions with a trainable linear projection (Eq. 1). We refer to the output of this projection as the patch embeddings.

ViT (Vision Transformer)

Step 1 - Input

- transform image into 16 x 16 size (patches)
- embed each patch into 768 dimensions
- i.e. one patch can be described by 1 x 768 values
- In the case that we have 196 patches with size of 16 x 16, we obtain [14, 14, 768] tensor
- with the use of flatten, we obtain [196, 768] matrix

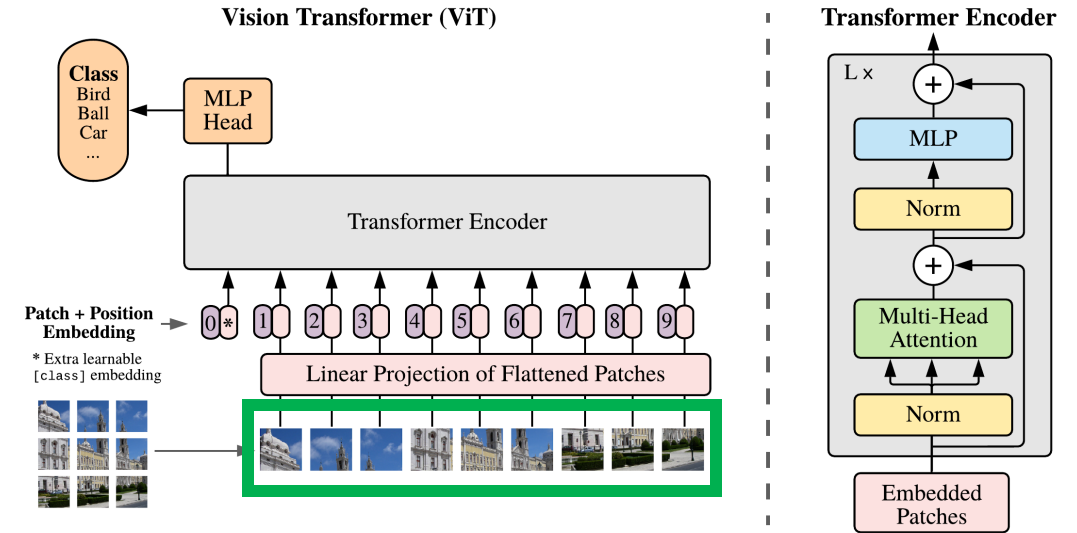


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable “classification token” to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

An overview of the model is depicted in Figure 1. The standard Transformer receives as input a 1D sequence of token embeddings. To handle 2D images, we reshape the image $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ into a sequence of flattened 2D patches $\mathbf{x}_p \in \mathbb{R}^{N \times (P^2 \cdot C)}$, where (H, W) is the resolution of the original image, C is the number of channels, (P, P) is the resolution of each image patch, and $N = HW/P^2$ is the resulting number of patches, which also serves as the effective input sequence length for the Transformer. The Transformer uses constant latent vector size D through all of its layers, so we flatten the patches and map to D dimensions with a trainable linear projection (Eq. 1). We refer to the output of this projection as the patch embeddings.

ViT (Vision Transformer)

Step 1 - Input

- transform image into 16 x 16 size (patches)
- embed each patch into 768 dimensions
- i.e. one patch can be described by 1 x 768 values
- In the case that we have 196 patches with size of 16 x 16, we obtain [14, 14, 768] tensor
- with the use of flatten, we obtain [196, 768] matrix

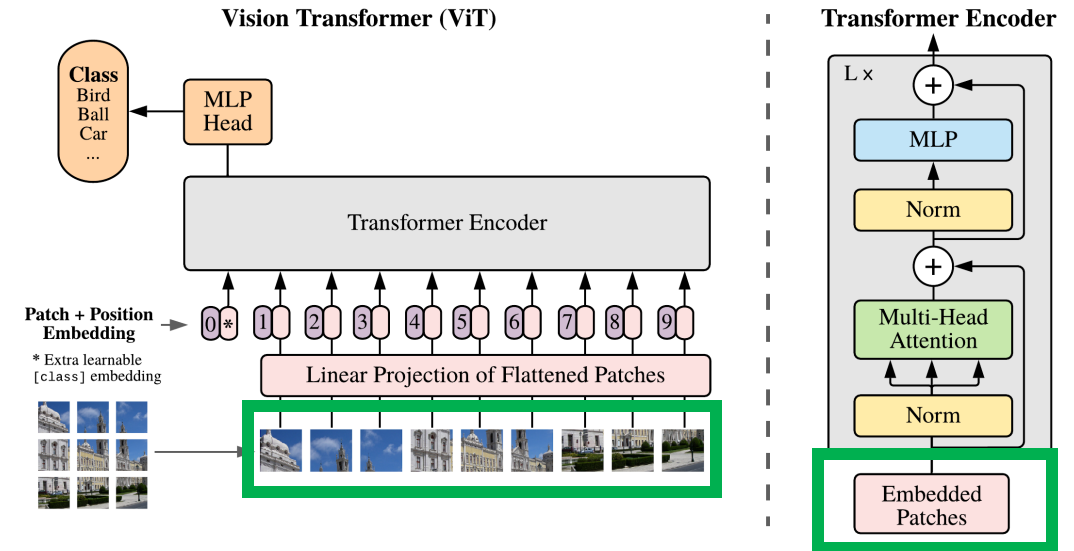


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable “classification token” to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

An overview of the model is depicted in Figure 1. The standard Transformer receives as input a 1D sequence of token embeddings. To handle 2D images, we reshape the image $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ into a sequence of flattened 2D patches $\mathbf{x}_p \in \mathbb{R}^{N \times (P^2 \cdot C)}$, where (H, W) is the resolution of the original image, C is the number of channels, (P, P) is the resolution of each image patch, and $N = HW/P^2$ is the resulting number of patches, which also serves as the effective input sequence length for the Transformer. The Transformer uses constant latent vector size D through all of its layers, so we flatten the patches and map to D dimensions with a trainable linear projection (Eq. 1). We refer to the output of this projection as the patch embeddings.

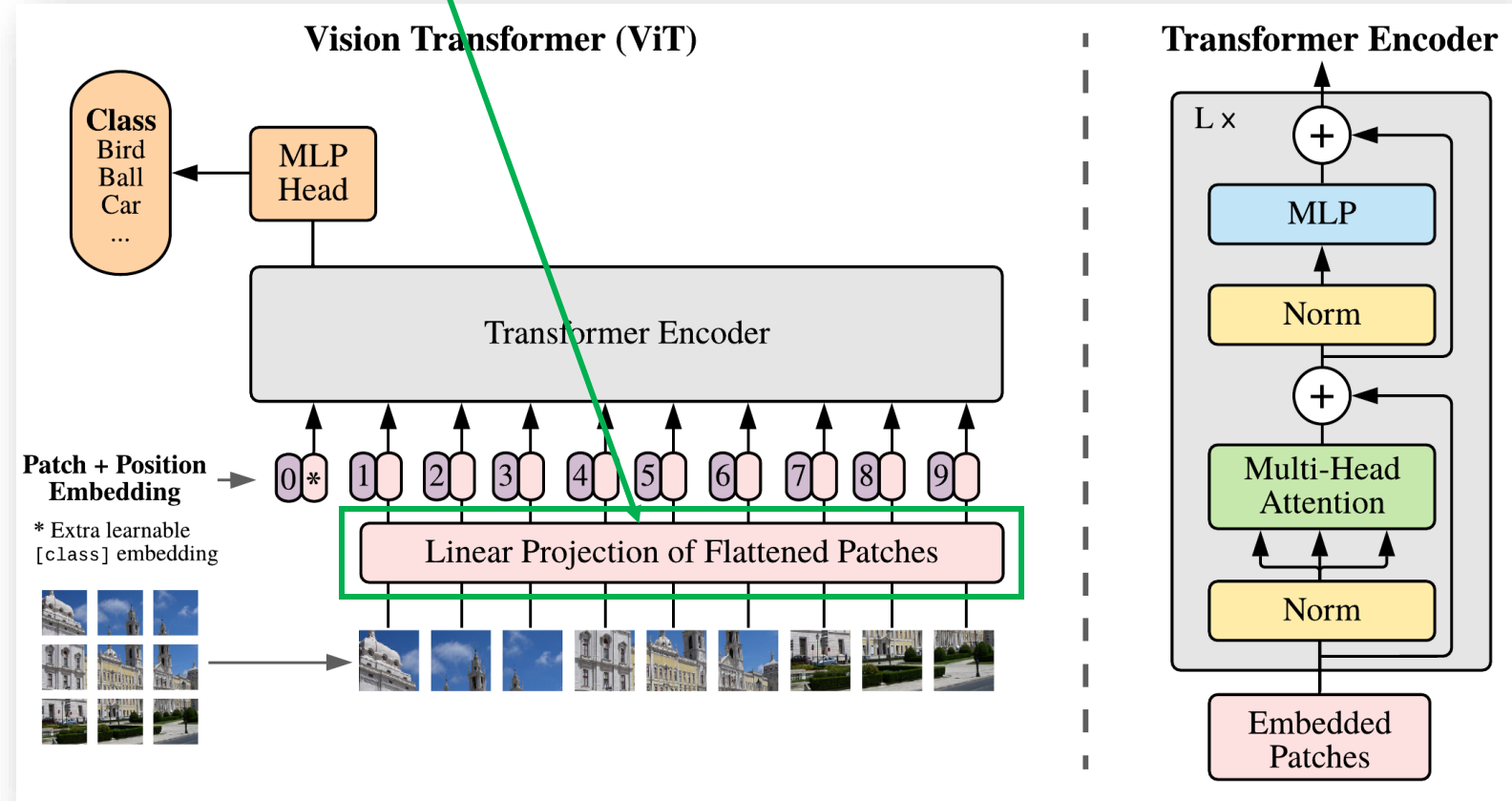
ViT (Vision Transformer)

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1 \mathbf{E}; \mathbf{x}_p^2 \mathbf{E}; \dots; \mathbf{x}_p^N \mathbf{E}] + \mathbf{E}_{\text{pos}}, \quad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}, \mathbf{E}_{\text{pos}} \in \mathbb{R}^{(N+1) \times D} \quad (1)$$

$$\mathbf{z}'_\ell = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \quad \ell = 1 \dots L \quad (2)$$

$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, \quad \ell = 1 \dots L \quad (3)$$

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0) \quad (4)$$



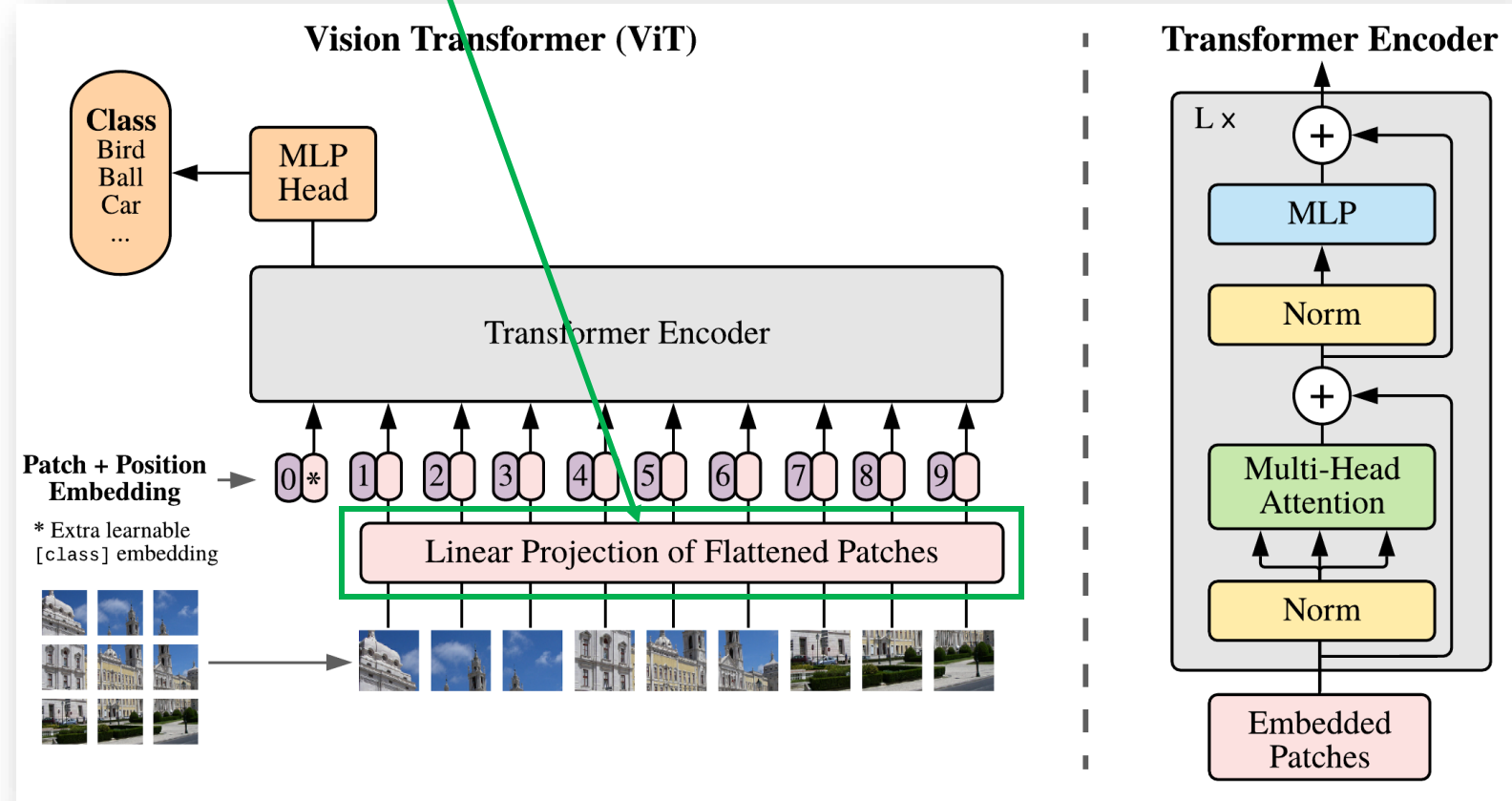
ViT (Vision Transformer)

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1 \mathbf{E}; \mathbf{x}_p^2 \mathbf{E}; \dots; \mathbf{x}_p^N \mathbf{E}] + \mathbf{E}_{\text{pos}}, \quad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}, \mathbf{E}_{\text{pos}} \in \mathbb{R}^{(N+1) \times D} \quad (1)$$

$$\mathbf{z}'_\ell = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \quad \ell = 1 \dots L \quad (2)$$

$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, \quad \ell = 1 \dots L \quad (3)$$

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0) \quad (4)$$



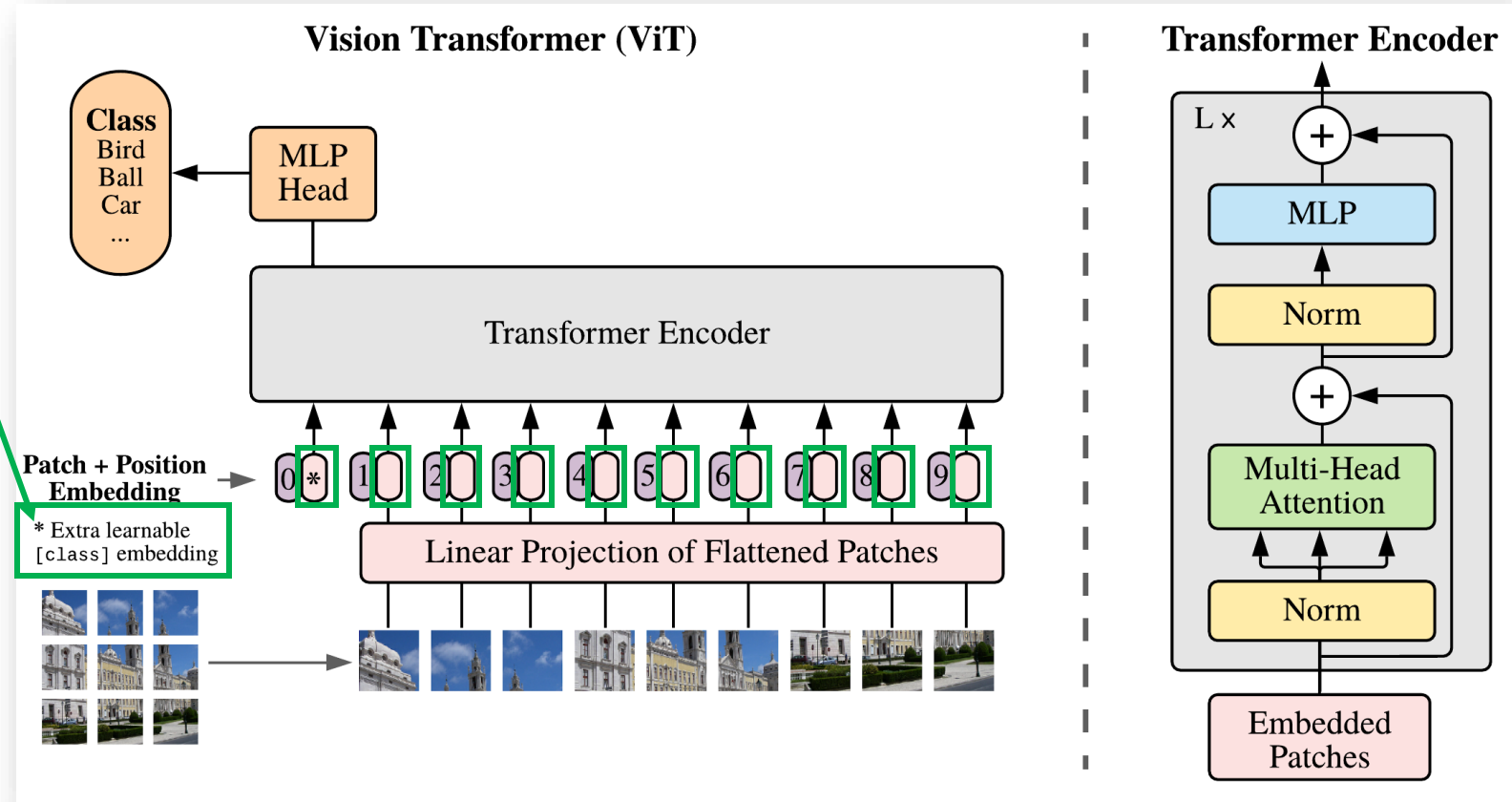
ViT (Vision Transformer)

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1 \mathbf{E}; \mathbf{x}_p^2 \mathbf{E}; \dots; \mathbf{x}_p^N \mathbf{E}] + \mathbf{E}_{\text{pos}}, \quad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}, \mathbf{E}_{\text{pos}} \in \mathbb{R}^{(N+1) \times D} \quad (1)$$

$$\mathbf{z}'_\ell = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \quad \ell = 1 \dots L \quad (2)$$

$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, \quad \ell = 1 \dots L \quad (3)$$

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0) \quad (4)$$



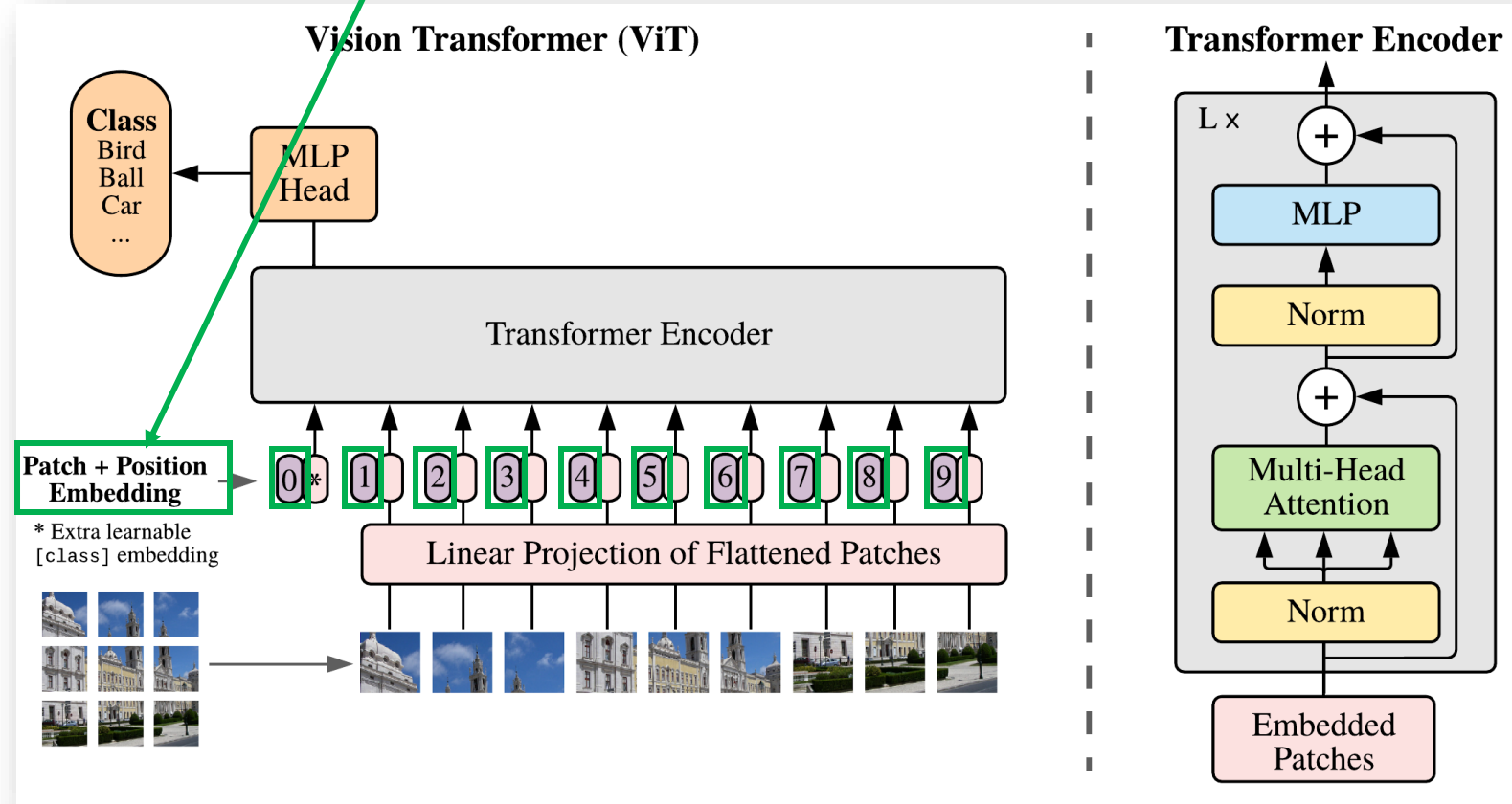
ViT (Vision Transformer)

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1 \mathbf{E}; \mathbf{x}_p^2 \mathbf{E}; \dots; \mathbf{x}_p^N \mathbf{E}] + \mathbf{E}_{pos}, \quad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}, \mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D} \quad (1)$$

$$\mathbf{z}'_\ell = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \quad \ell = 1 \dots L \quad (2)$$

$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, \quad \ell = 1 \dots L \quad (3)$$

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0) \quad (4)$$



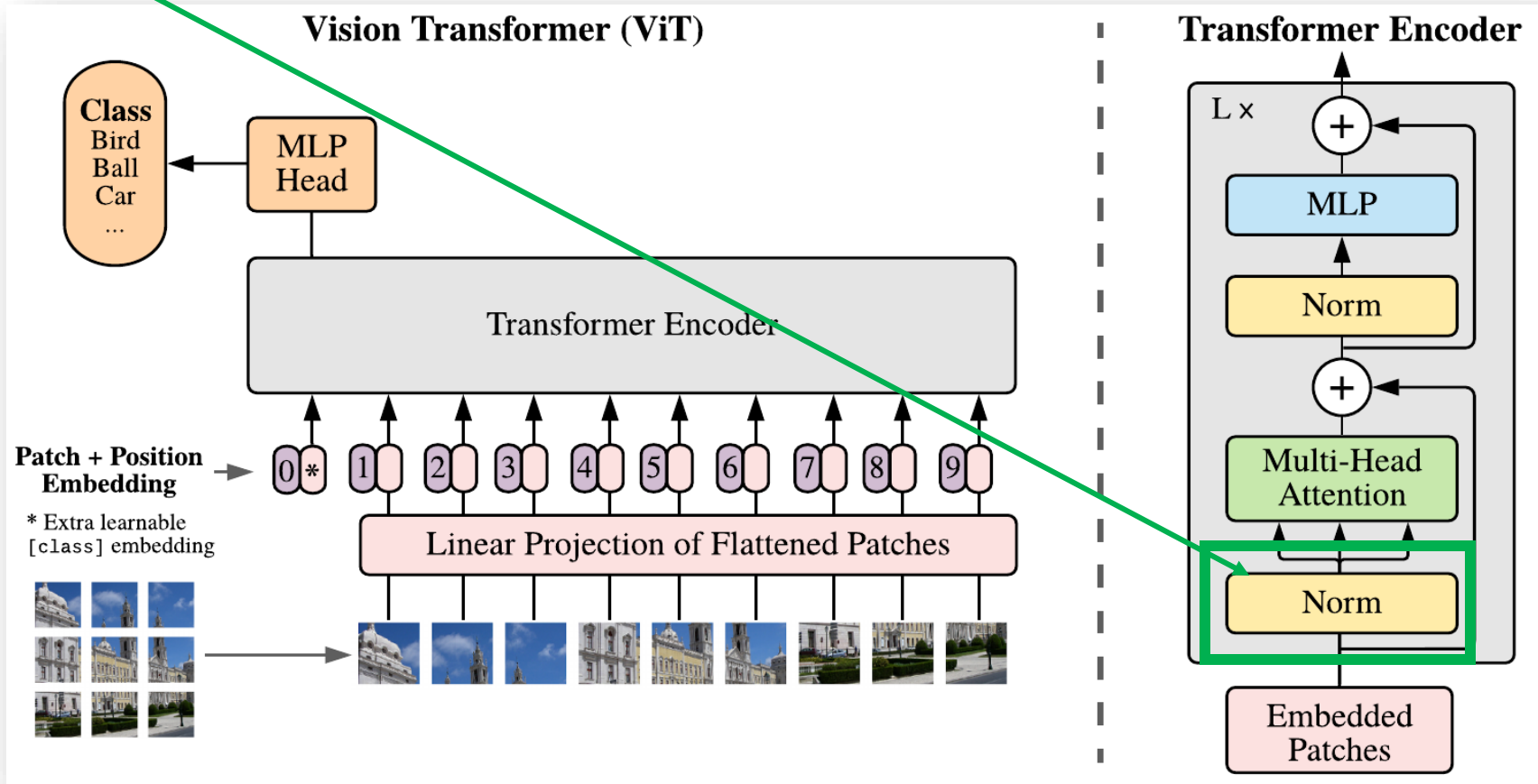
ViT (Vision Transformer)

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1 \mathbf{E}; \mathbf{x}_p^2 \mathbf{E}; \dots; \mathbf{x}_p^N \mathbf{E}] + \mathbf{E}_{\text{pos}}, \quad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}, \mathbf{E}_{\text{pos}} \in \mathbb{R}^{(N+1) \times D} \quad (1)$$

$$\mathbf{z}'_{\ell} = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \quad \ell = 1 \dots L \quad (2)$$

$$\mathbf{z}_{\ell} = \text{MLP}(\text{LN}(\mathbf{z}'_{\ell})) + \mathbf{z}'_{\ell}, \quad \ell = 1 \dots L \quad (3)$$

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0) \quad (4)$$



LayerNorm

```
CLASS torch.nn.LayerNorm(normalized_shape, eps=1e-05, elementwise_affine=True, bias=True, device=None, dtype=None) [SOURCE]
```

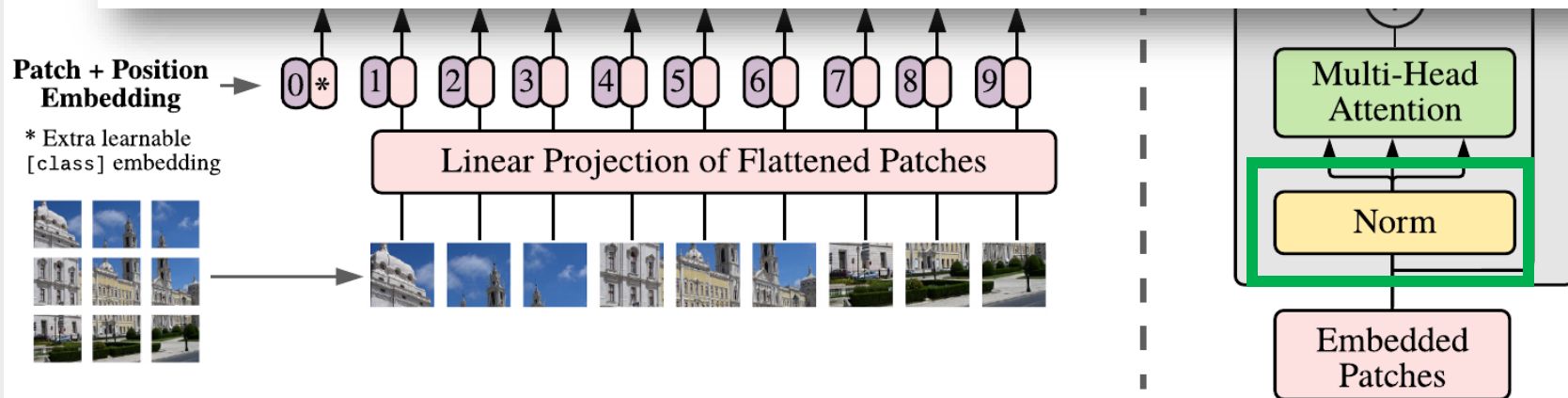
Applies Layer Normalization over a mini-batch of inputs.

This layer implements the operation as described in the paper [Layer Normalization](#)

$$y = \frac{x - E[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

The mean and standard-deviation are calculated over the last D dimensions, where D is the dimension of `normalized_shape`. For example, if `normalized_shape` is `(3, 5)` (a 2-dimensional shape), the mean and standard-deviation are computed over the last 2 dimensions of the input (i.e. `input.mean((-2, -1))`). γ and β are learnable affine transform parameters of `normalized_shape` if `elementwise_affine` is `True`. The standard-deviation is calculated via the biased estimator, equivalent to `torch.var(input, unbiased=False)`.

$$\begin{aligned} \mathbf{z}_0 &= [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1 \mathbf{E}; \\ \mathbf{z}'_\ell &= \text{MSA}(\text{LN}(\mathbf{z}_\ell)) \\ \mathbf{z}_\ell &= \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) \\ \mathbf{y} &= \text{LN}(\mathbf{z}_L^0) \end{aligned}$$



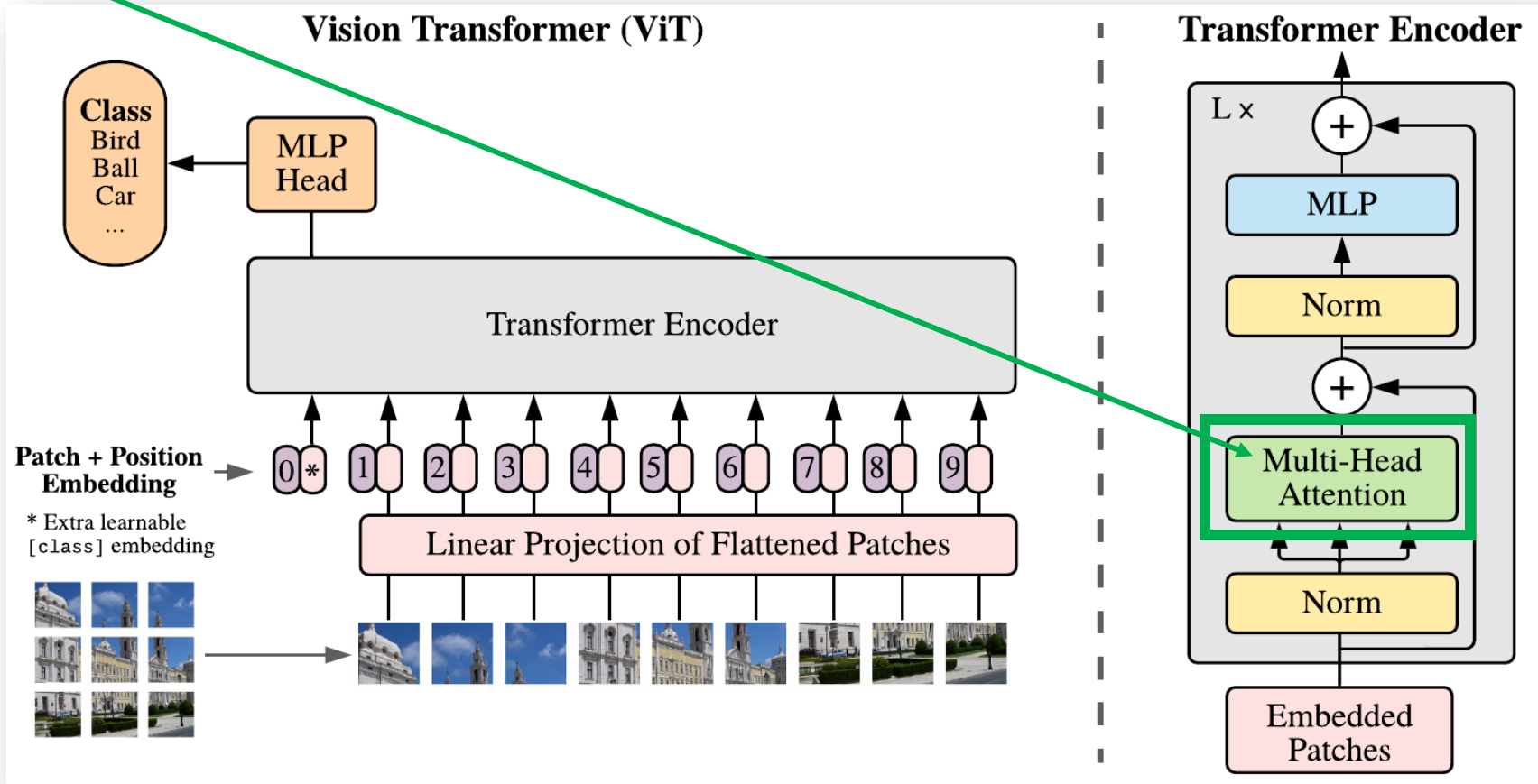
ViT (Vision Transformer)

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1 \mathbf{E}; \mathbf{x}_p^2 \mathbf{E}; \dots; \mathbf{x}_p^N \mathbf{E}] + \mathbf{E}_{\text{pos}}, \quad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}, \mathbf{E}_{\text{pos}} \in \mathbb{R}^{(N+1) \times D} \quad (1)$$

$$\mathbf{z}'_\ell = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \quad \ell = 1 \dots L \quad (2)$$

$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, \quad \ell = 1 \dots L \quad (3)$$

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0) \quad (4)$$



MultiheadAttention

```
CLASS torch.nn.MultiheadAttention(embed_dim, num_heads, dropout=0.0, bias=True,
    add_bias_kv=False, add_zero_attn=False, kdim=None, vdim=None, batch_first=False,
    device=None, dtype=None) [SOURCE]
```

Allows the model to jointly attend to information from different representation subspaces.

Method described in the paper: [Attention Is All You Need](#).

Multi-Head Attention is defined as:

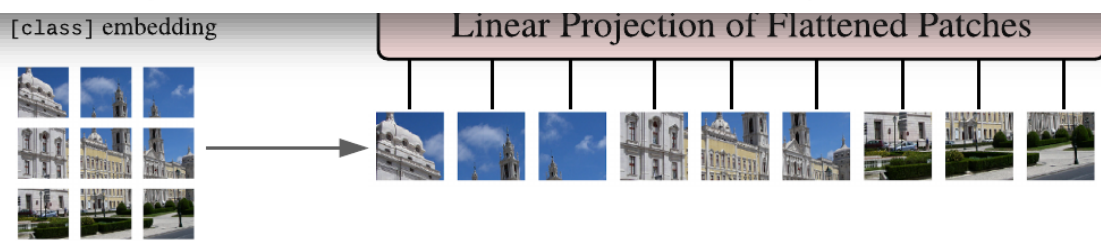
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

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

`nn.MultiheadAttention` will use the optimized implementations of `scaled_dot_product_attention()` when possible.

In addition to support for the new `scaled_dot_product_attention()` function, for speeding up Inference, MHA will use fastpath inference with support for Nested Tensors, iff:

- self attention is being computed (i.e., `query`, `key`, and `value` are the same tensor).



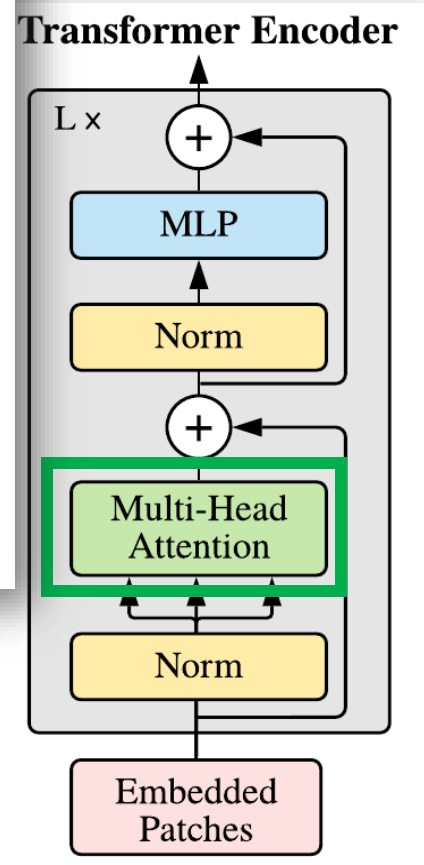
Transformer

$$pos \in \mathbb{R}^{(N+1) \times D} \quad (1)$$

$$(2)$$

$$(3)$$

$$(4)$$



A MULTIHEAD SELF-ATTENTION

Standard **qkv** self-attention (SA, Vaswani et al. (2017)) is a popular building block for neural architectures. For each element in an input sequence $\mathbf{z} \in \mathbb{R}^{N \times D}$, we compute a weighted sum over all values $\mathbf{v}$ in the sequence. The attention weights A_{ij} are based on the pairwise similarity between two elements of the sequence and their respective query $\mathbf{q}^i$ and key $\mathbf{k}^j$ representations.

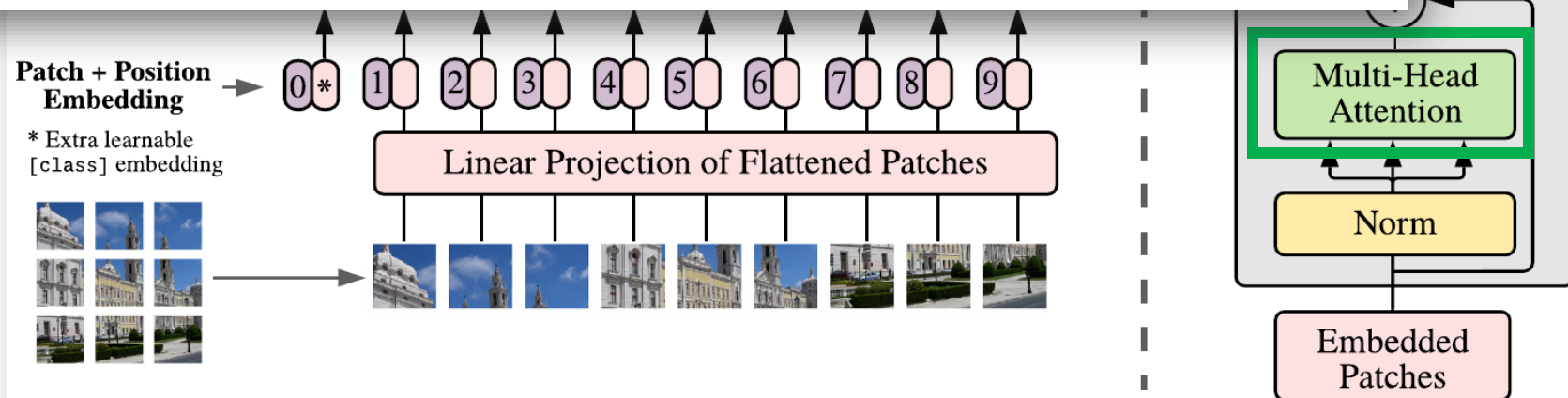
$$[\mathbf{q}, \mathbf{k}, \mathbf{v}] = \mathbf{z} \mathbf{U}_{qkv} \quad \mathbf{U}_{qkv} \in \mathbb{R}^{D \times 3D_h}, \quad (5)$$

$$\mathbf{A} = \text{softmax} \left(\mathbf{q} \mathbf{k}^\top / \sqrt{D_h} \right) \quad \mathbf{A} \in \mathbb{R}^{N \times N}, \quad (6)$$

$$\text{SA}(\mathbf{z}) = \mathbf{A} \mathbf{v}. \quad (7)$$

Multihead self-attention (MSA) is an extension of SA in which we run k self-attention operations, called “heads”, in parallel, and project their concatenated outputs. To keep compute and number of parameters constant when changing k , D_h (Eq. 5) is typically set to D/k .

$$\text{MSA}(\mathbf{z}) = [\text{SA}_1(\mathbf{z}); \text{SA}_2(\mathbf{z}); \dots; \text{SA}_k(\mathbf{z})] \mathbf{U}_{msa} \quad \mathbf{U}_{msa} \in \mathbb{R}^{k \cdot D_h \times D} \quad (8)$$



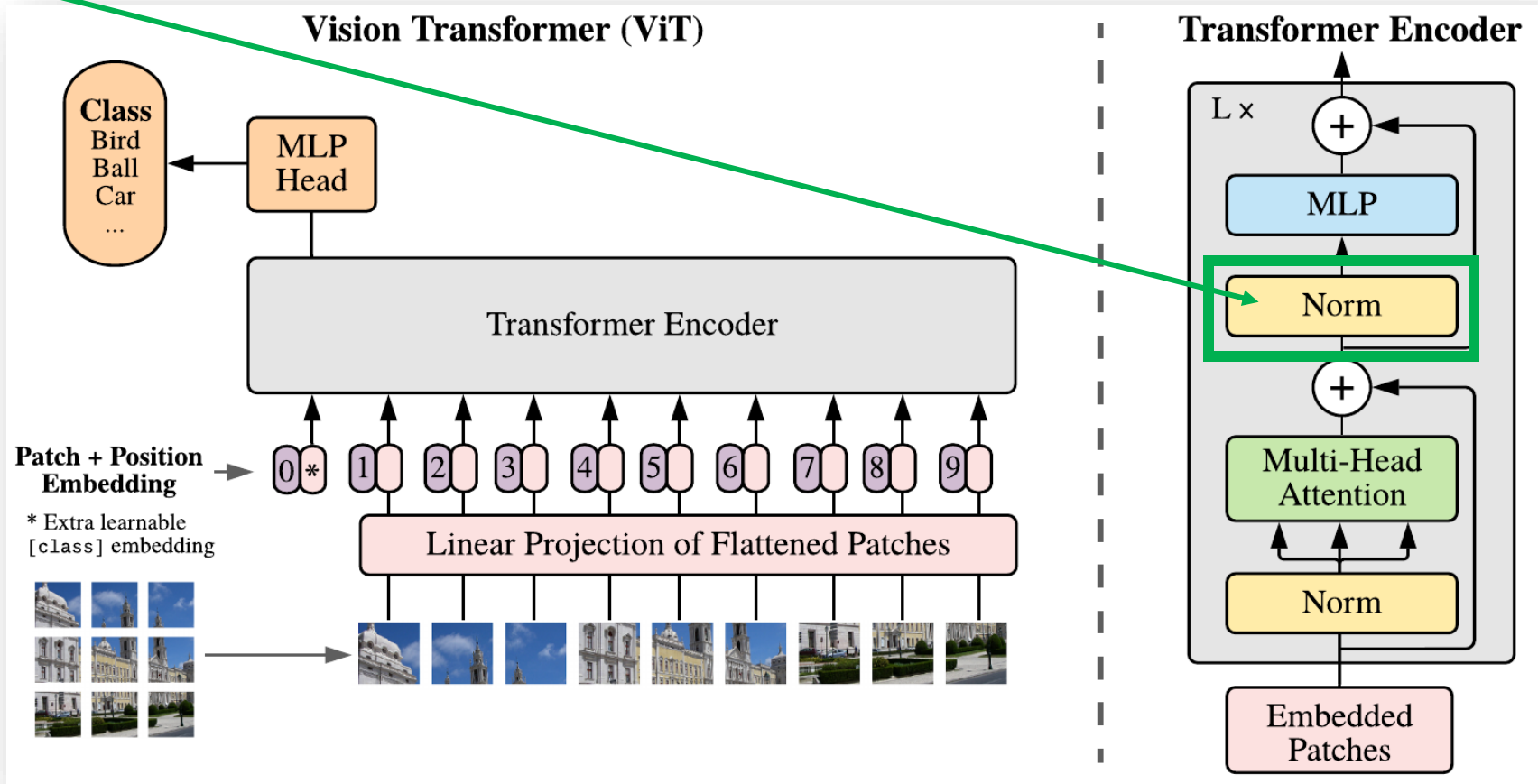
ViT (Vision Transformer)

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1 \mathbf{E}; \mathbf{x}_p^2 \mathbf{E}; \dots; \mathbf{x}_p^N \mathbf{E}] + \mathbf{E}_{\text{pos}}, \quad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}, \mathbf{E}_{\text{pos}} \in \mathbb{R}^{(N+1) \times D} \quad (1)$$

$$\mathbf{z}'_\ell = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \quad \ell = 1 \dots L \quad (2)$$

$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, \quad \ell = 1 \dots L \quad (3)$$

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0) \quad (4)$$



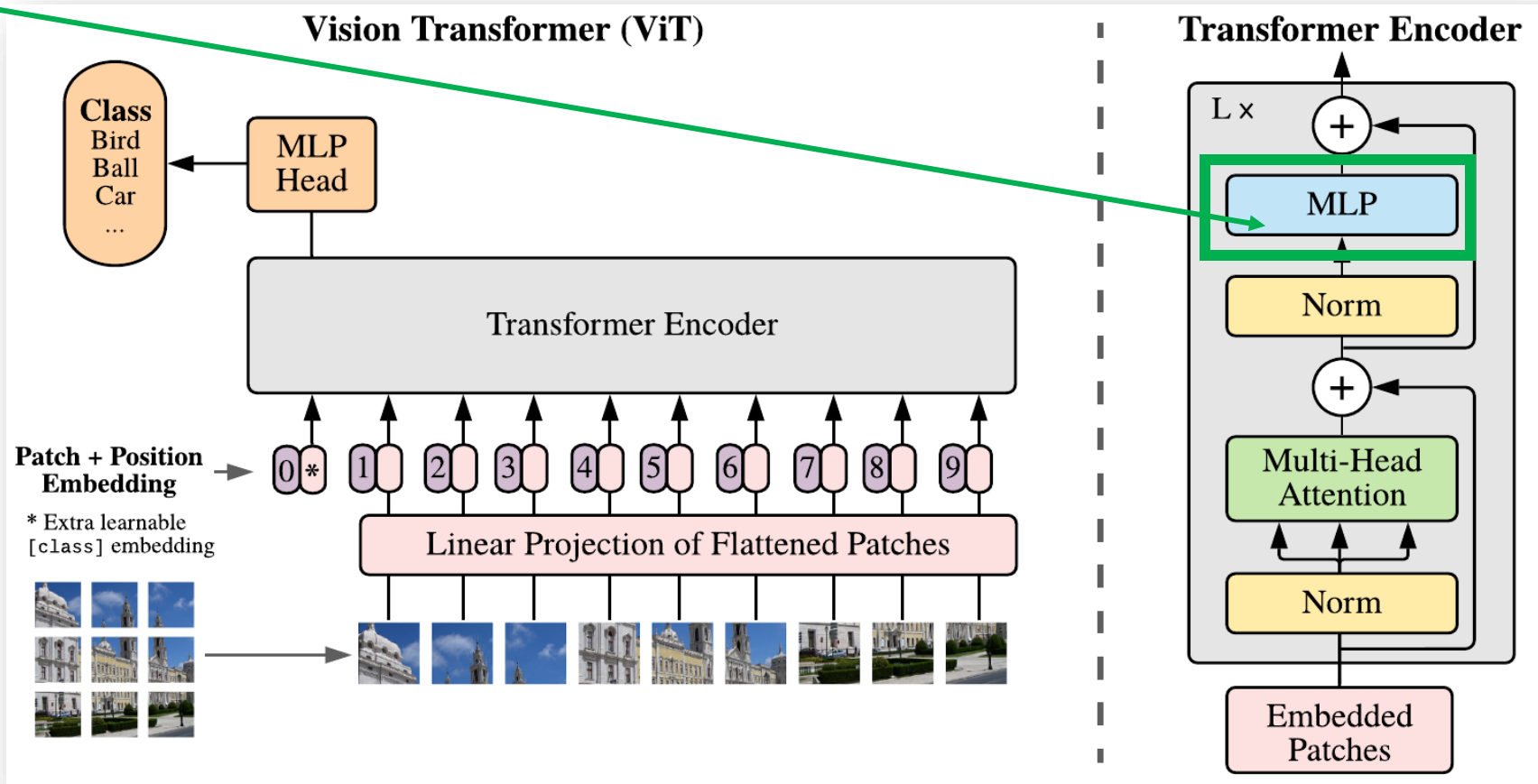
ViT (Vision Transformer)

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1 \mathbf{E}; \mathbf{x}_p^2 \mathbf{E}; \dots; \mathbf{x}_p^N \mathbf{E}] + \mathbf{E}_{\text{pos}}, \quad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}, \mathbf{E}_{\text{pos}} \in \mathbb{R}^{(N+1) \times D} \quad (1)$$

$$\mathbf{z}'_\ell = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \quad \ell = 1 \dots L \quad (2)$$

$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, \quad \ell = 1 \dots L \quad (3)$$

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0) \quad (4)$$



MLP

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1 \mathbf{E};$$

$$\mathbf{z}'_\ell = \text{MSA}(\text{LN}(\mathbf{z}_\ell$$

$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z}'_\ell$$

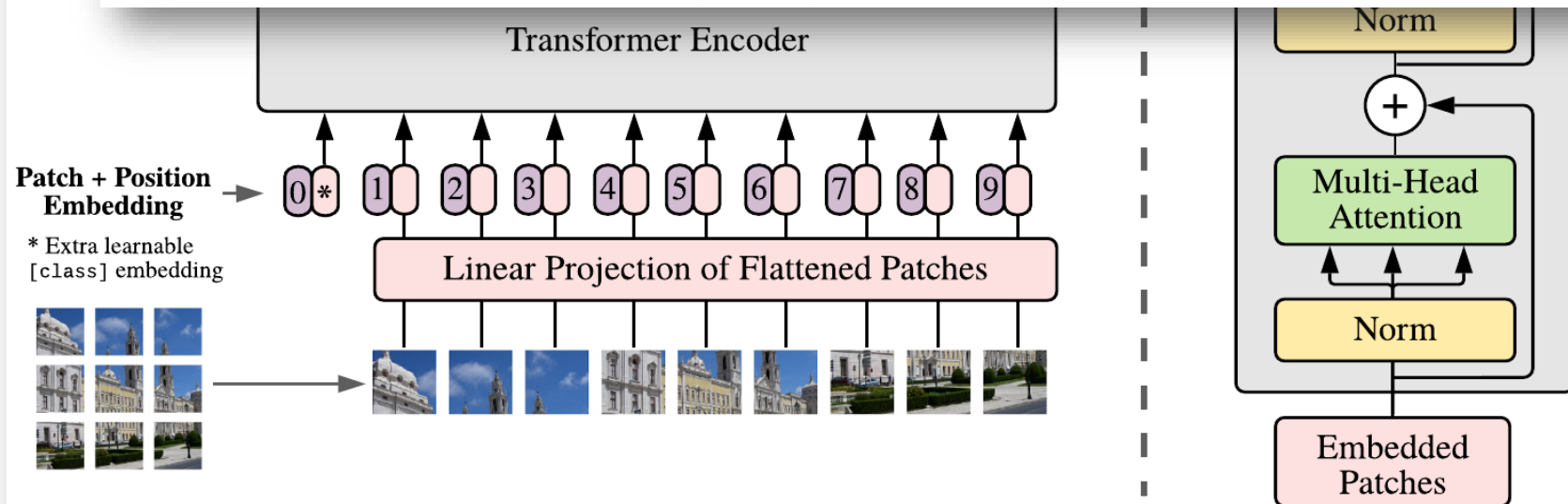
$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0)$$

```
CLASS torchvision.ops.MLP(in_channels: int, hidden_channels: ~typing.List[int],
    norm_layer: ~typing.Optional[~typing.Callable[[...],
    ~torch.nn.modules.module.Module]] = None, activation_layer:
    ~typing.Optional[~typing.Callable[[...], ~torch.nn.modules.module.Module]] =
    <class 'torch.nn.modules.activation.ReLU'>, inplace: ~typing.Optional[bool] =
    None, bias: bool = True, dropout: float = 0.0) [SOURCE]
```

This block implements the multi-layer perceptron (MLP) module.

Parameters:

- **in_channels** (*int*) – Number of channels of the input
- **hidden_channels** (*List[int]*) – List of the hidden channel dimensions



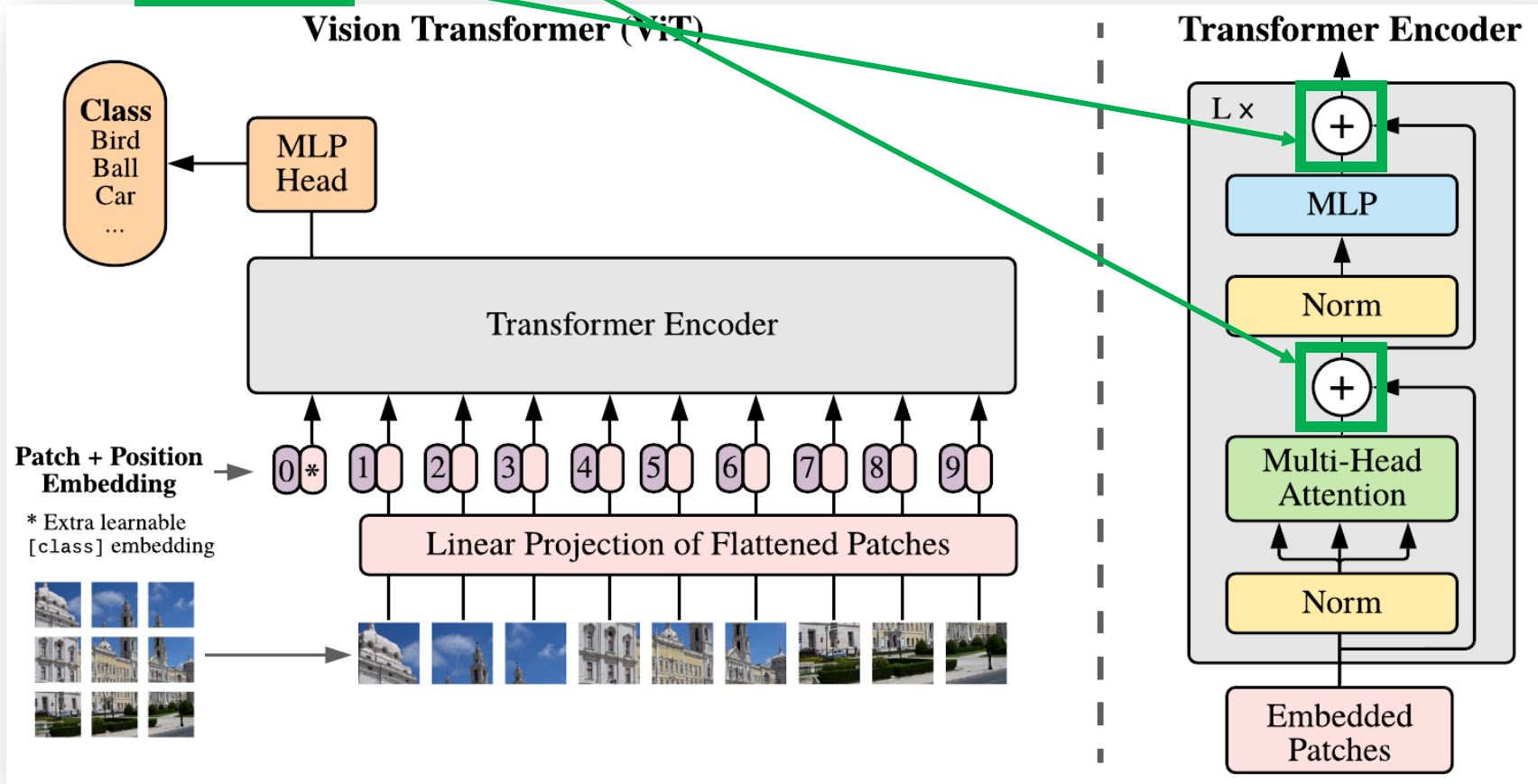
The Transformer encoder (Vaswani et al. 2017) consists of alternating layers of multiheaded self-attention (MSA, see Appendix A) and MLP blocks (Eq. 2, 3). Layernorm (LN) is applied before every block, and residual connections after every block (Wang et al. 2019; Baevski & Auli 2019).

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1 \mathbf{E}; \mathbf{x}_p^2 \mathbf{E}; \dots; \mathbf{x}_p^N \mathbf{E}] + \mathbf{E}_{\text{pos}}, \quad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}, \mathbf{E}_{\text{pos}} \in \mathbb{R}^{(N+1) \times D} \quad (1)$$

$$\mathbf{z}'_\ell = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \quad \ell = 1 \dots L \quad (2)$$

$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, \quad \ell = 1 \dots L \quad (3)$$

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0) \quad (4)$$



The Transformer encoder (Vaswani et al. 2017) consists of alternating layers of multiheaded self-attention (MSA, see Appendix A) and MLP blocks (Eq. 2, 3). Layernorm (LN) is applied before every block, and residual connections after every block (Wang et al. 2019; Baevski & Auli 2019).

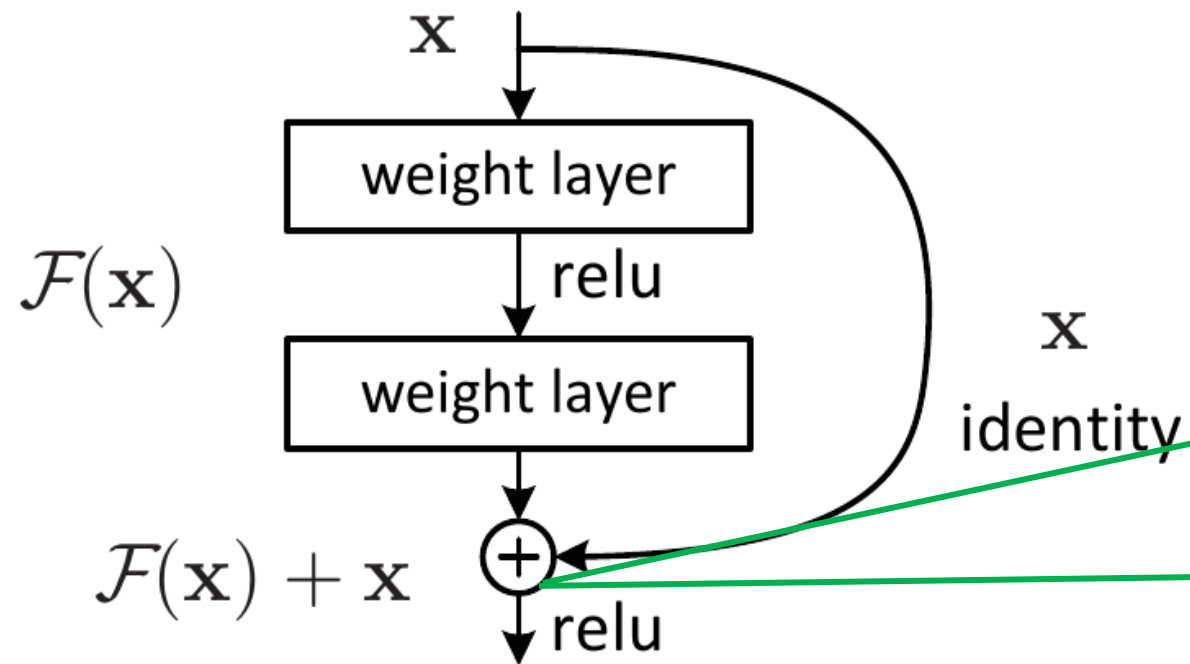


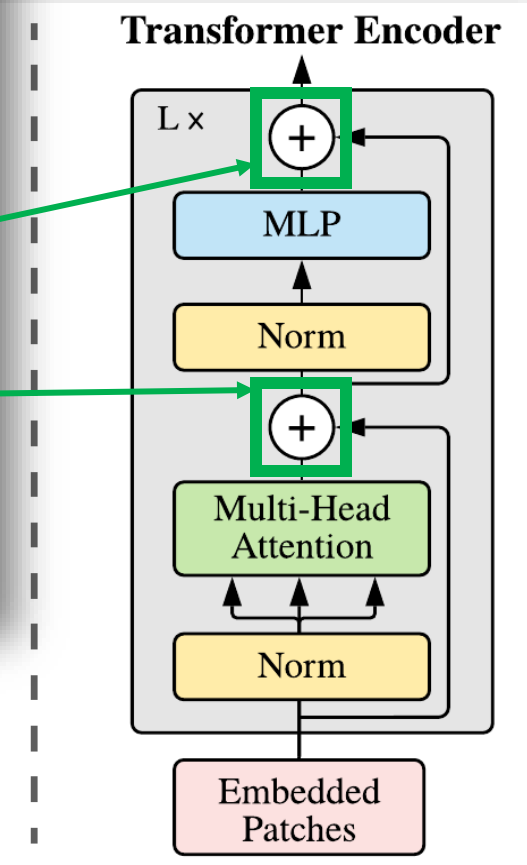
Figure 2. Residual learning: a building block.

$(\mathbf{P}^2, \mathbf{C}) \in \mathbb{R}^{(N+1) \times D}, \mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D}$ (1)

(2)

(3)

(4)



Deep Residual Learning for Image Recognition

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun

Microsoft Research

{kahe, v-xiangz, v-shren, jiansun}@microsoft.com



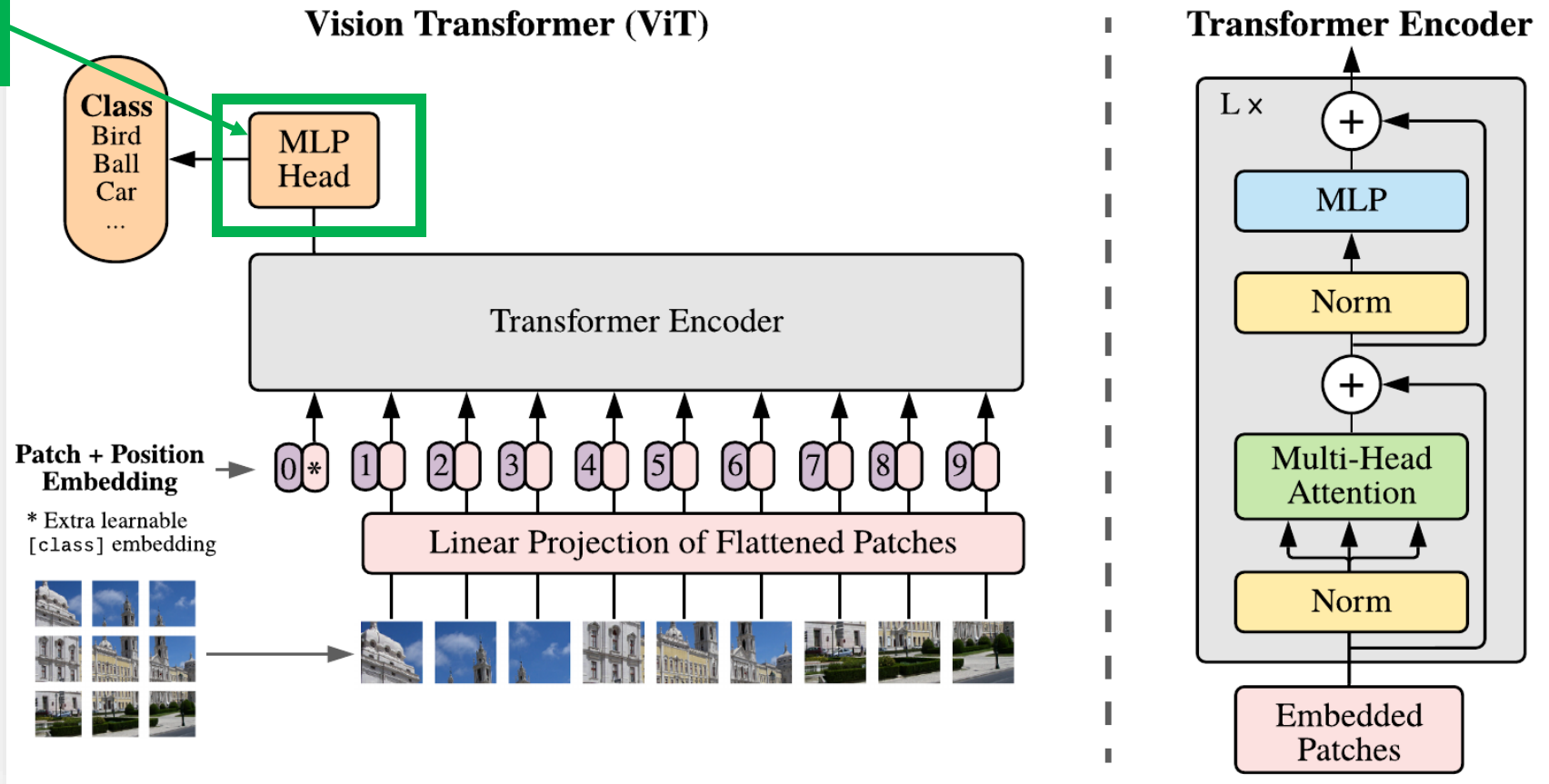
The Transformer encoder (Vaswani et al. 2017) consists of alternating layers of multiheaded self-attention (MSA, see Appendix A) and MLP blocks (Eq. 2, 3). Layernorm (LN) is applied before every block, and residual connections after every block (Wang et al. 2019; Baevski & Auli 2019).

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1 \mathbf{E}; \mathbf{x}_p^2 \mathbf{E}; \dots; \mathbf{x}_p^N \mathbf{E}] + \mathbf{E}_{\text{pos}}, \quad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}, \mathbf{E}_{\text{pos}} \in \mathbb{R}^{(N+1) \times D} \quad (1)$$

$$\mathbf{z}'_\ell = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \quad \ell = 1 \dots L \quad (2)$$

$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, \quad \ell = 1 \dots L \quad (3)$$

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0) \quad (4)$$



ViT (Vision Transformer)

Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Table 1: Details of Vision Transformer model variants.

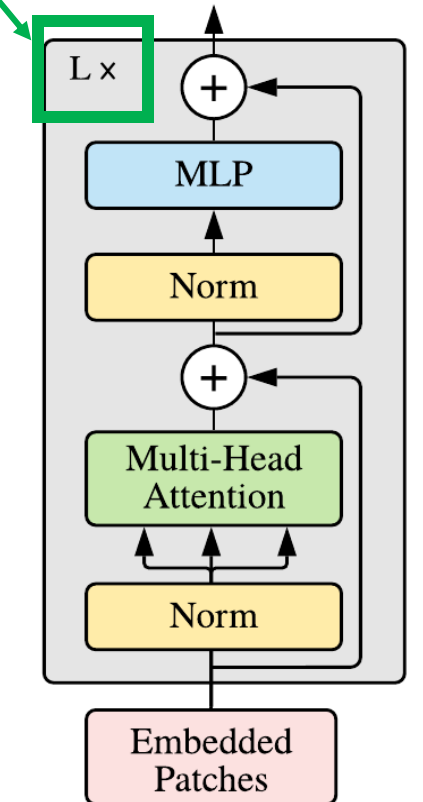
$$\mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}, \mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D} \quad (1)$$

$$\ell = 1 \dots L \quad (2)$$

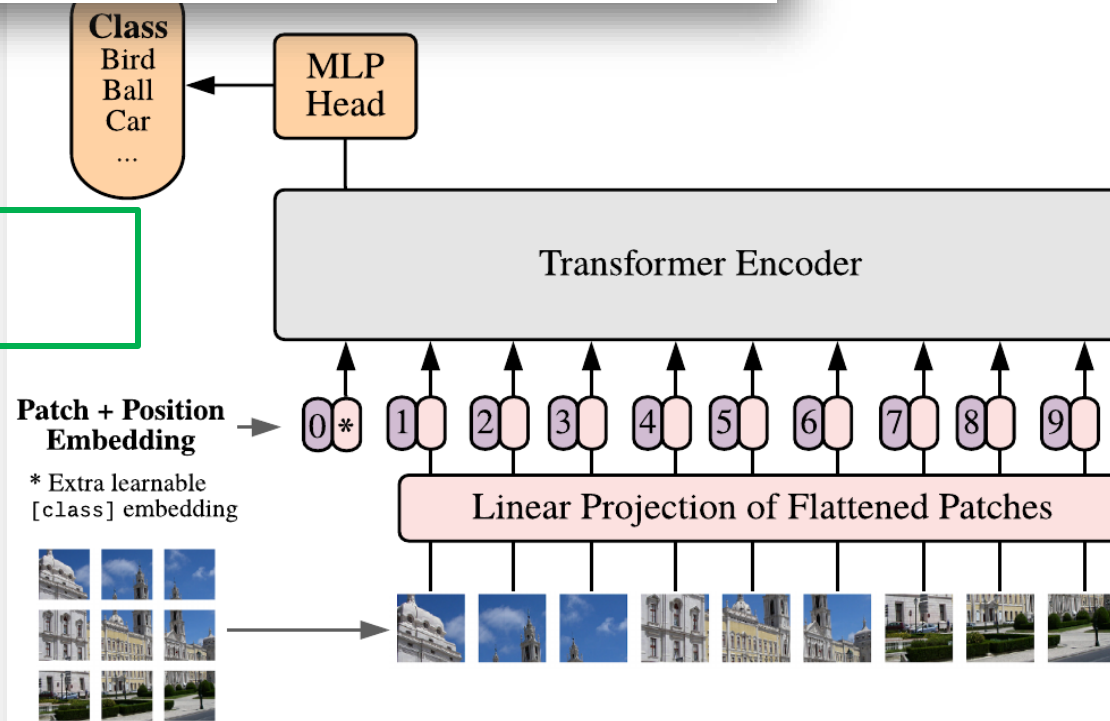
$$\ell = 1 \dots L \quad (3)$$

$$(4)$$

Transformer Encoder



number of Transformer Encoder blocks



ViT (Vision Transformer)

Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Table 1: Details of Vision Transformer model variants.

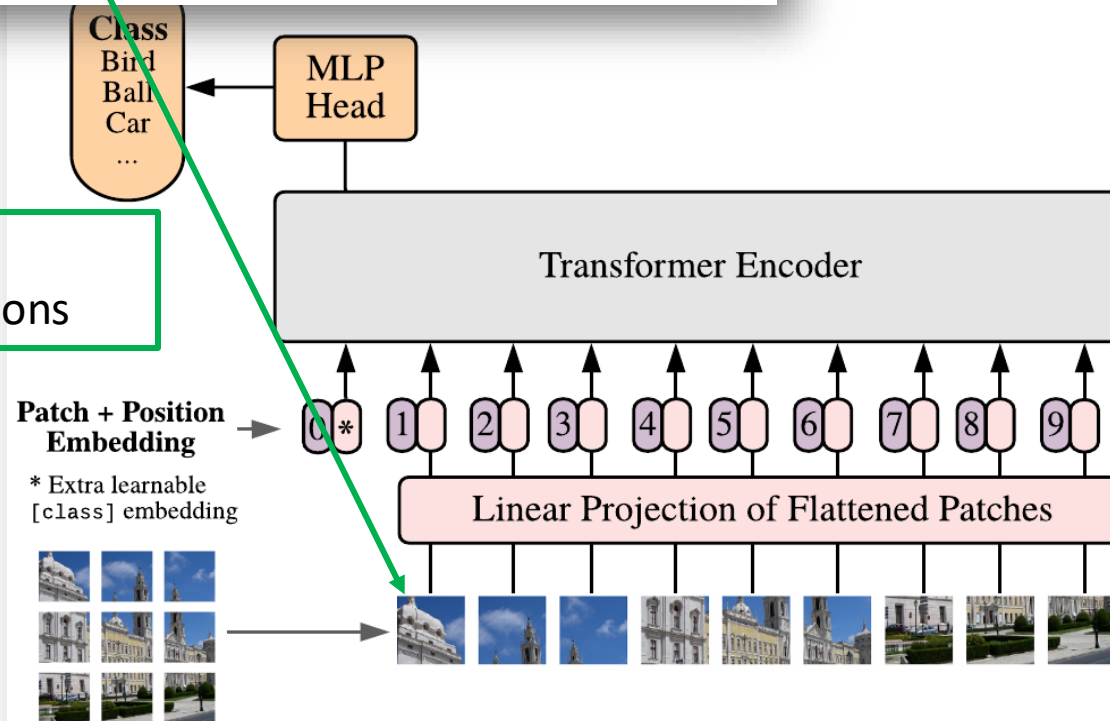
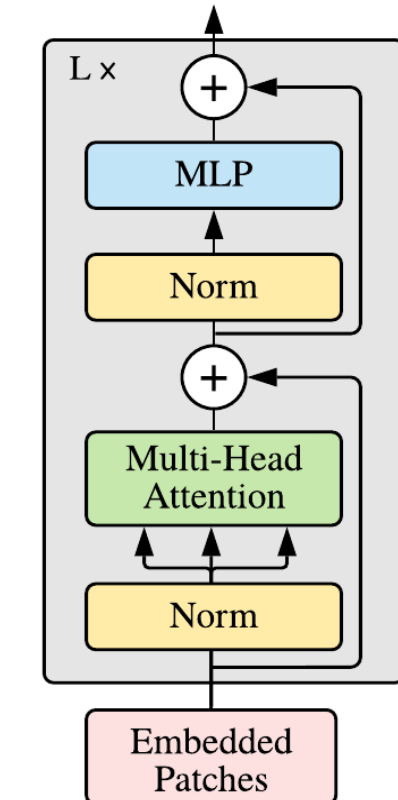
$$\mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}, \mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D} \quad (1)$$

$$\ell = 1 \dots L \quad (2)$$

$$\ell = 1 \dots L \quad (3)$$

$$(4)$$

Transformer Encoder



embed each patch into
768/1024/1280 dimensions

ViT (Vision Transformer)

Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Table 1: Details of Vision Transformer model variants.

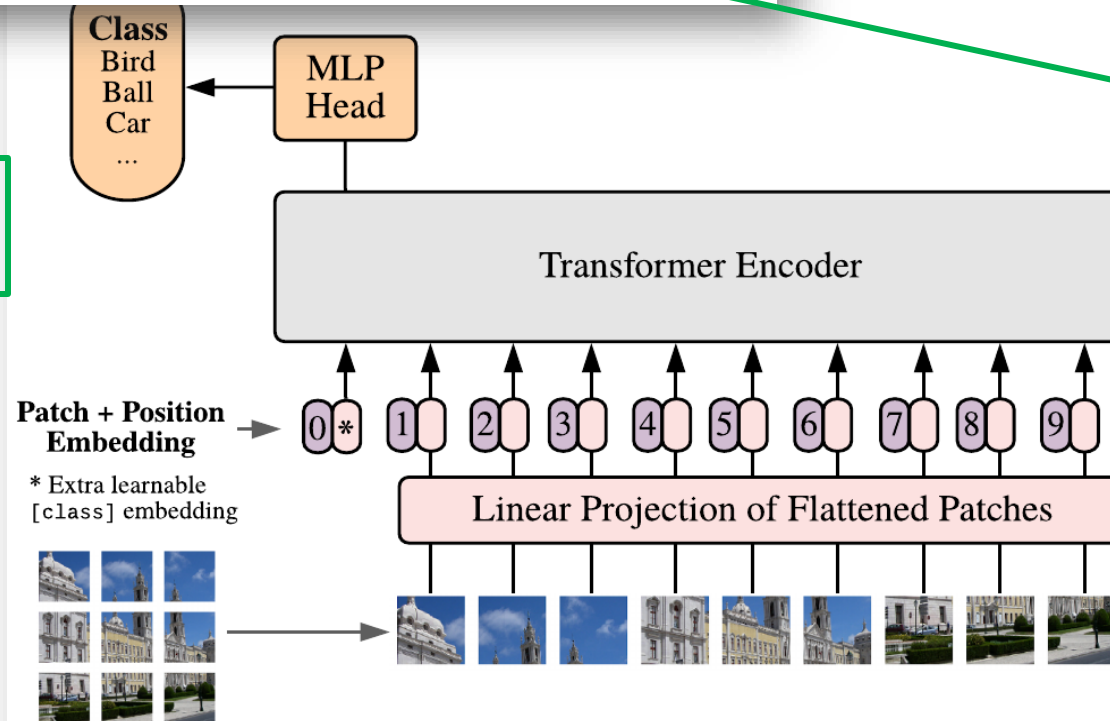
$$\mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}, \mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D} \quad (1)$$

$$\ell = 1 \dots L \quad (2)$$

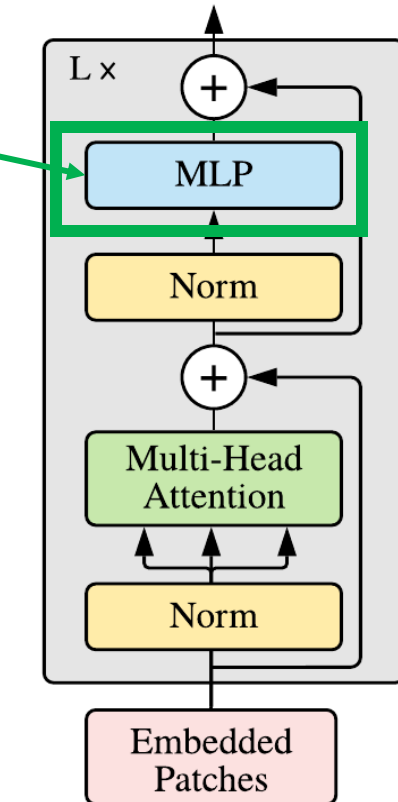
$$\ell = 1 \dots L \quad (3)$$

$$(4)$$

Number of hidden units in MLP block



Transformer Encoder



ViT (Vision Transformer)

Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Table 1: Details of Vision Transformer model variants.

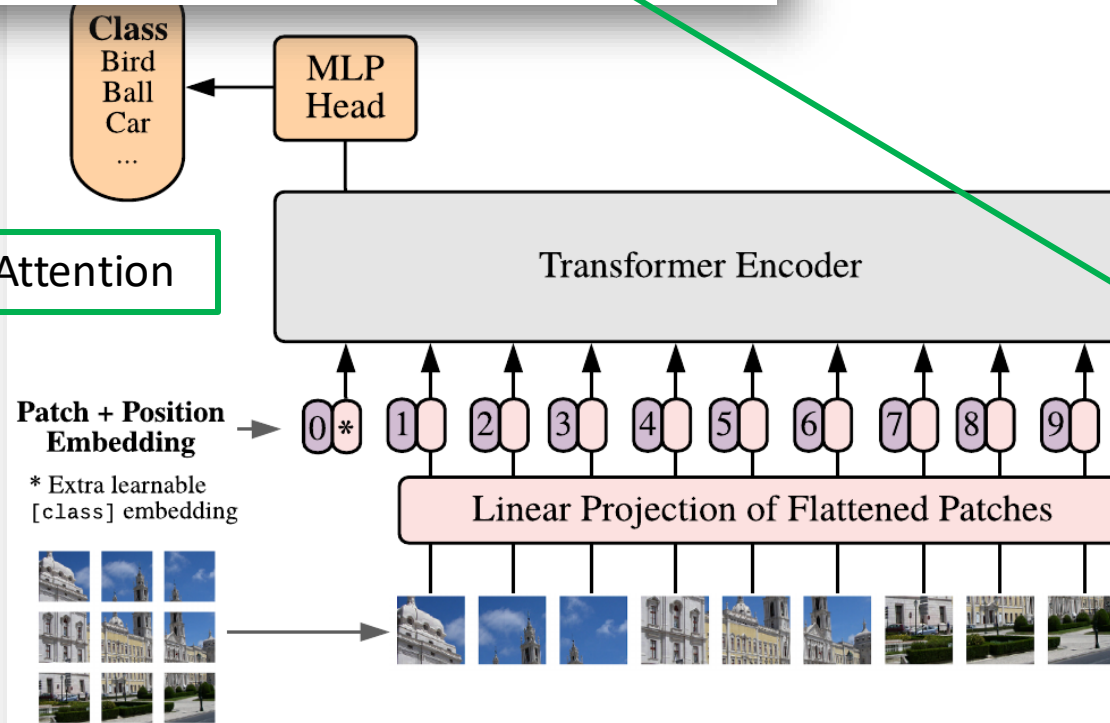
$$\mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}, \mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D} \quad (1)$$

$$\ell = 1 \dots L \quad (2)$$

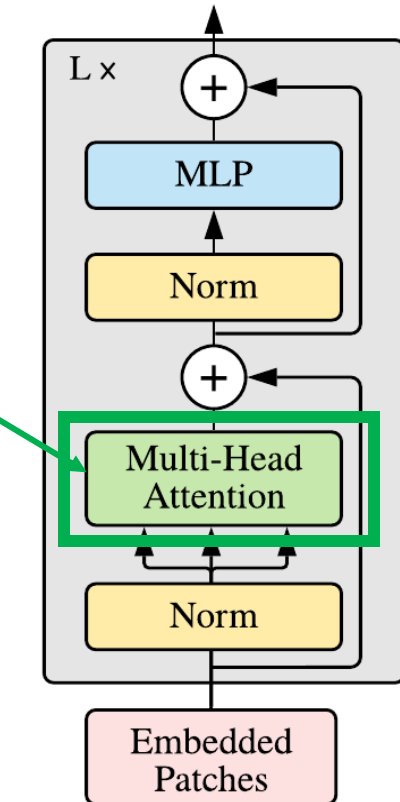
$$\ell = 1 \dots L \quad (3)$$

$$(4)$$

Number of Multi-Head Attention



Transformer Encoder



ViT (Vision Transformer)

Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Table 1: Details of Vision Transformer model variants.

$\mathbf{E} \in \mathbb{R}^{n \times d}$

$\ell = 1$

$\ell = 1$

Number of Multi-Head Attention

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

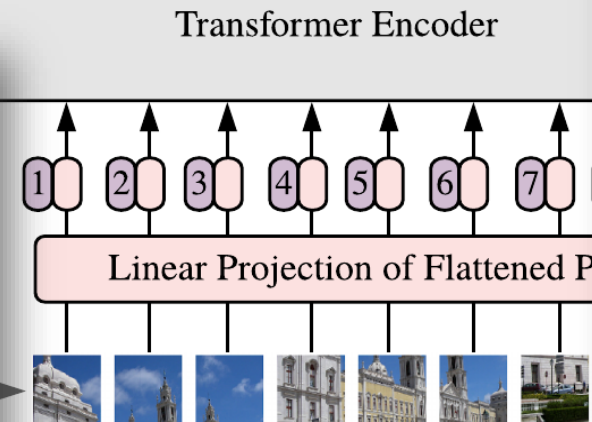
Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

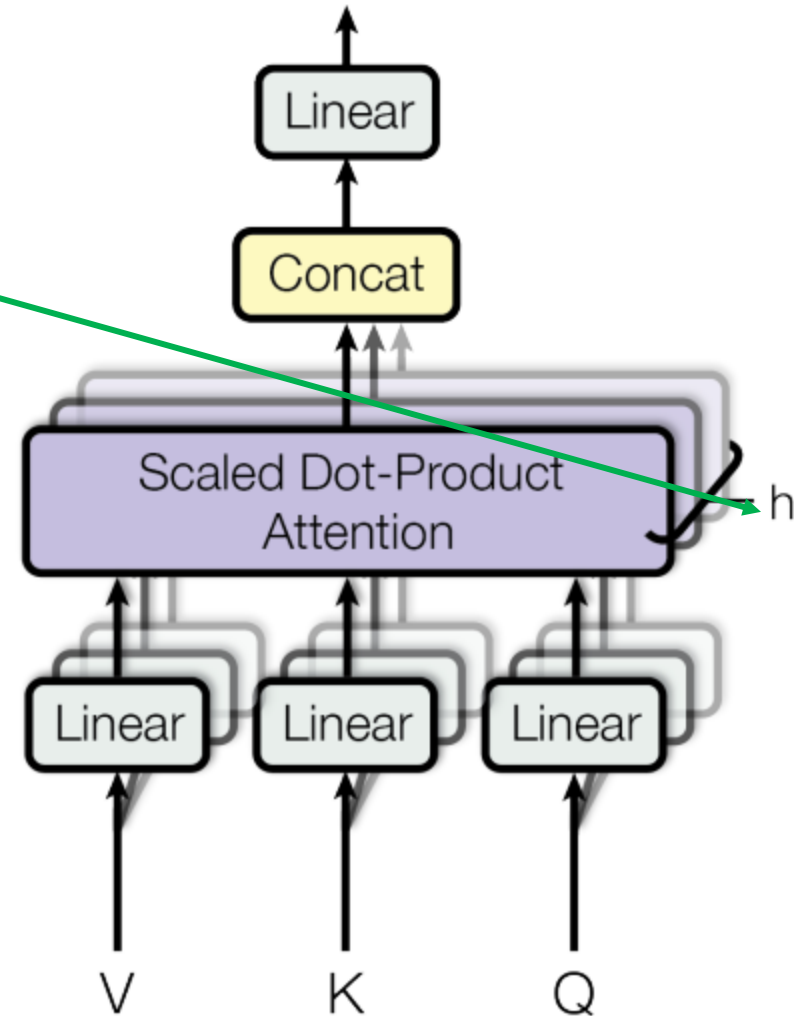
Aidan N. Gomez*[†]
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser*
Google Brain
lukaszkaier@google.com

Illia Polosukhin*[‡]
illia.polosukhin@gmail.com



Multi-Head Attention



<https://arxiv.org/pdf/1706.03762>

<https://arxiv.org/abs/2010.11929>

ViT (Vision Transformer)

Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Table 1: Details of Vision Transformer models

$$\mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}, \mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D} \quad (1)$$

$$\ell = 1 \dots L \quad (2)$$

$$\ell = 1 \dots L \quad (3)$$

(4)

MultiheadAttention

```
CLASS torch.nn.MultiheadAttention(embed_dim, num_heads, dropout=0.0, bias=True,
add_bias_kv=False, add_zero_attn=False, kdim=None, vdim=None, batch_first=False,
device=None, dtype=None) [SOURCE]
```

Allows the model to jointly attend to information from different representation subspaces.

Method described in the paper: [Attention Is All You Need](#).

Multi-Head Attention is defined as:

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

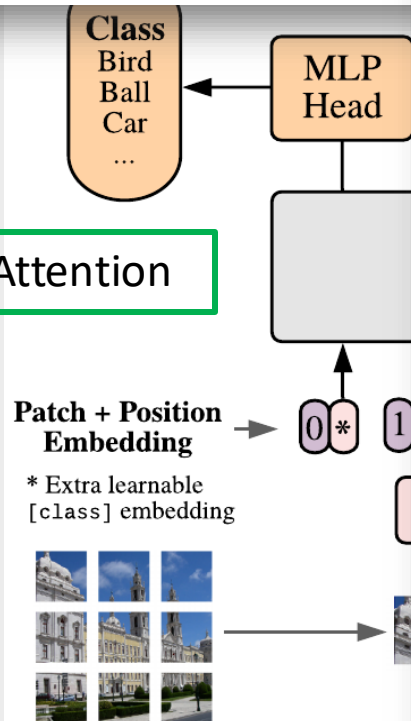
where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$.

`nn.MultiHeadAttention` will use the optimized implementations of `scaled_dot_product_attention()` when possible.

In addition to support for the new `scaled_dot_product_attention()` function, for speeding up Inference, MHA will use fastpath inference with support for Nested Tensors, iff:

- self attention is being computed (i.e., `query`, `key`, and `value` are the same tensor).

Number of Multi-Head Attention



ViT (Vision Transformer)

VisionTransformer

The VisionTransformer model is based on the [An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale](#) paper.

Model builders

The following model builders can be used to instantiate a VisionTransformer model, with or without pre-trained weights. All the model builders internally rely on the `torchvision.models.vision_transformer.VisionTransformer` base class. Please refer to the [source code](#) for more details about this class.

vit_b_16(*[, weights, progress])

Constructs a vit_b_16 architecture from [An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale](#).

vit_b_32(*[, weights, progress])

Constructs a vit_b_32 architecture from [An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale](#).

vit_l_16(*[, weights, progress])

Constructs a vit_l_16 architecture from [An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale](#).

vit_l_32(*[, weights, progress])

Constructs a vit_l_32 architecture from [An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale](#).

vit_h_14(*[, weights, progress])

Constructs a vit_h_14 architecture from [An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale](#).

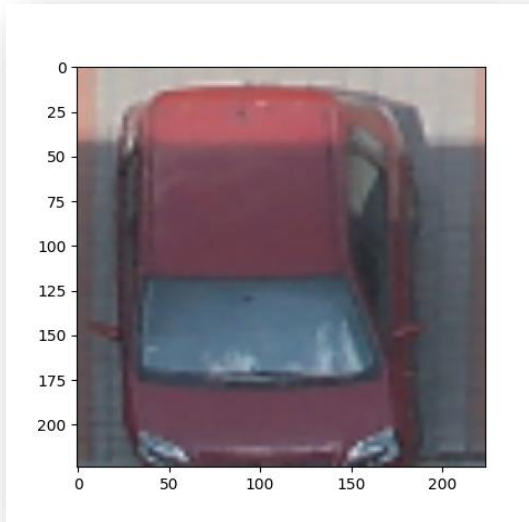
Published as a conference paper at ICLR 2021

		ViT-B/16	ViT-B/32	ViT-L/16	ViT-L/32	ViT-H/14
ImageNet	CIFAR-10	98.13	97.77	97.86	97.94	-
	CIFAR-100	87.13	86.31	86.35	87.07	-
	ImageNet	77.91	73.38	76.53	71.16	-
	ImageNet ReaL	83.57	79.56	82.19	77.83	-
	Oxford Flowers-102	89.49	85.43	89.66	86.36	-
	Oxford-IIIT-Pets	93.81	92.04	93.64	91.35	-
ImageNet-21k	CIFAR-10	98.95	98.79	99.16	99.13	99.27
	CIFAR-100	91.67	91.97	93.44	93.04	93.82
	ImageNet	83.97	81.28	85.15	80.99	85.13
	ImageNet ReaL	88.35	86.63	88.40	85.65	88.70
	Oxford Flowers-102	99.38	99.11	99.61	99.19	99.51
	Oxford-IIIT-Pets	94.43	93.02	94.73	93.09	94.82
JFT-300M	CIFAR-10	99.00	98.61	99.38	99.19	99.50
	CIFAR-100	91.87	90.49	94.04	92.52	94.55
	ImageNet	84.15	80.73	87.12	84.37	88.04
	ImageNet ReaL	88.85	86.27	89.99	88.28	90.33
	Oxford Flowers-102	99.56	99.27	99.56	99.45	99.68
	Oxford-IIIT-Pets	95.80	93.40	97.11	95.83	97.56

Table 5: Top1 accuracy (in %) of Vision Transformer on various datasets when pre-trained on ImageNet, ImageNet-21k or JFT300M. These values correspond to Figure 3 in the main text. Models are fine-tuned at 384 resolution. Note that the ImageNet results are computed without additional techniques (Polyak averaging and 512 resolution images) used to achieve results in Table 2.

ViT – Selected Implementation Details – Equation 1.

We can use the **nn.Conv2d** layer to split the input image into patches (blocks) and create the embedding vector (with size = 768, in this example) for each patch.



```

h = 224
w = 224
c = 3
ps = 16

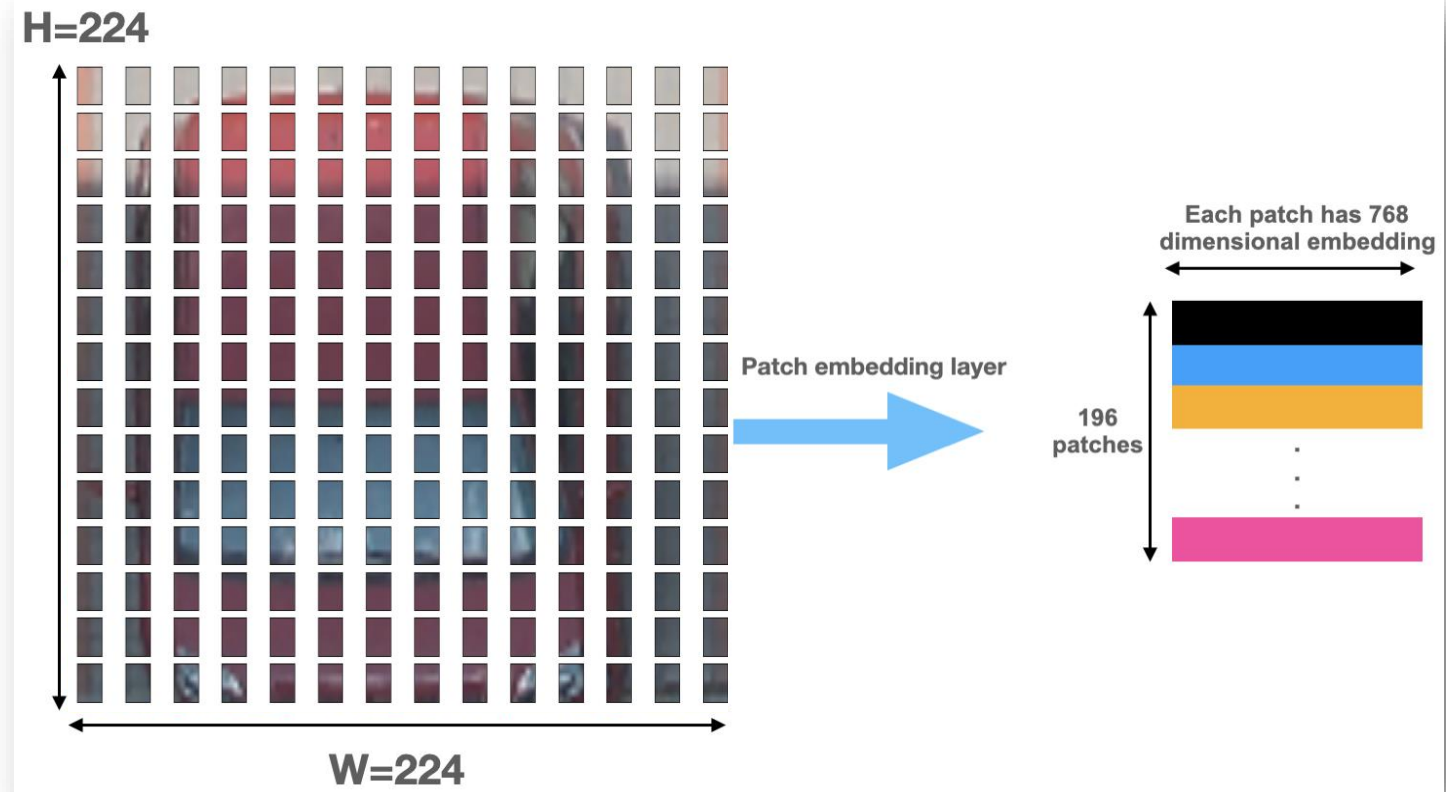
conv = nn.Conv2d(
    in_channels = 3,
    out_channels = 768,
    kernel_size = (ps,ps),
    stride       = ps,
    padding      = 0,
    bias         = False)
    
```

shape of input 224 x 244 x 3 image after nn.Conv2d:
([768, 14, 14])

ViT – Selected Implementation Details – Equation 1.

- 768 feature maps of size 14 x 14 are generated
- we can use flatten on these 14 x 14 feature maps to obtain: ([batch_size, 768, 196])
- it is necessary to switch dimensions so that the number of patches is in the second place: ([batch_size, 196, 768])

```
block_permute = nn.Sequential(
    nn.Conv2d(
        in_channels=3,
        out_channels=768,
        kernel_size=(ps,ps),
        stride=ps,
        padding = 0,
        bias=False),
    nn.Flatten(start_dim=2, end_dim=-1),
    MyPermute((0, 2, 1))
)
```



ViT – Selected Implementation Details – Equation 1.

- `class_token = nn.Parameter(torch.ones(batch_size, 1, 768))`
- `z_0_with_class = torch.cat((class_token, block_out_permute), dim=1)`
- `position_embeddings = nn.Parameter(torch.ones(1, 197, 768))`
- `z_0_with_class_with_pos = z_0_with_class + position_embeddings`

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1 \mathbf{E}; \mathbf{x}_p^2 \mathbf{E}; \dots; \mathbf{x}_p^N \mathbf{E}] + \mathbf{E}_{\text{pos}}, \quad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}, \mathbf{E}_{\text{pos}} \in \mathbb{R}^{(N+1) \times D} \quad (1)$$

$$\mathbf{z}'_{\ell} = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \quad \ell = 1 \dots L \quad (2)$$

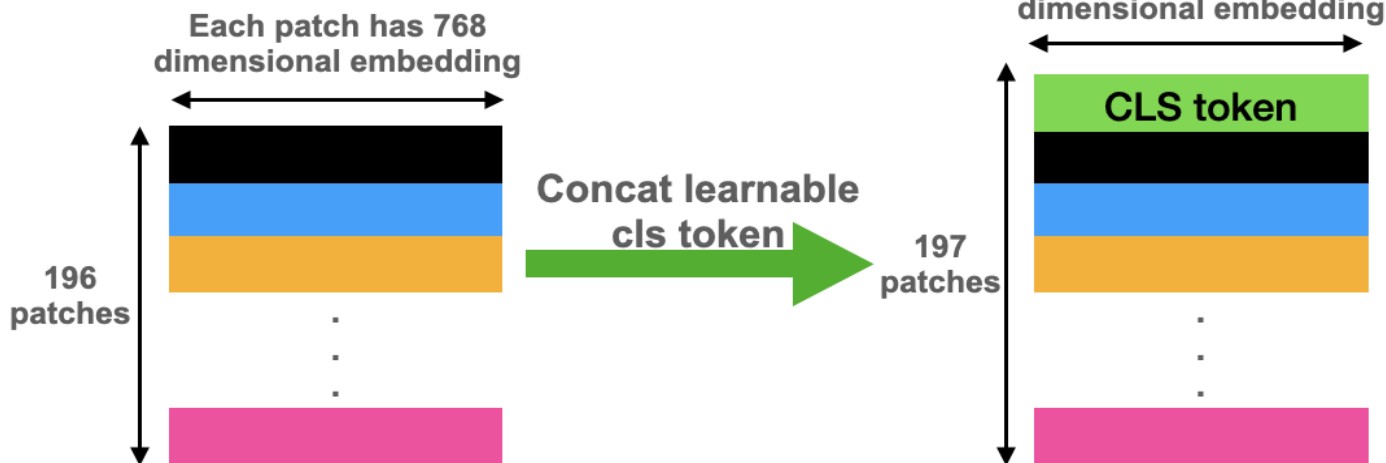
$$\mathbf{z}_{\ell} = \text{MLP}(\text{LN}(\mathbf{z}'_{\ell})) + \mathbf{z}'_{\ell}, \quad \ell = 1 \dots L \quad (3)$$

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0)$$

`block_permute = nn.Sequential(`

```
nn.Conv2d(
    in_channels=3,
    out_channels=768,
    kernel_size=(ps,ps),
    stride=ps,
    padding = 0,
    bias=False),
```

```
nn.Flatten(start_dim=2, end_dim=-1),
MyPermute((0, 2, 1))
)
```



ViT – Selected Implementation Details – Equation 1.

- `class_token = nn.Parameter(torch.ones(batch_size, 1, 768))`
- `z_0_with_class = torch.cat((class_token, block_out_permute), dim=1)`
- `position_embeddings = nn.Parameter(torch.ones(1, 197, 768))`
- `z_0_with_class_with_pos = z_0_with_class + position_embeddings`

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1 \mathbf{E}; \mathbf{x}_p^2 \mathbf{E}; \dots; \mathbf{x}_p^N \mathbf{E}] + \mathbf{E}_{pos}, \quad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}, \mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D} \quad (1)$$

$$\mathbf{z}'_\ell = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \quad \ell = 1 \dots L \quad (2)$$

$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, \quad \ell = 1 \dots L \quad (3)$$

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0) \quad (4)$$

A learnable parameter tensor of size 197x768



⋮



Position embedding



Element wise addition

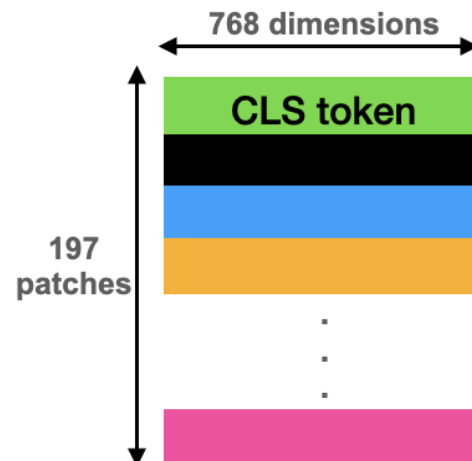


Image patches with classification token

Output of size 197x768



⋮



This is input into the vision transformer

Input to Vision Transformer



ViT – Selected Implementation Details – Equation 1.

- `class_token = nn.Parameter(torch.ones(batch_size, 1, 768))`
- `z_0_with_class = torch.cat((class_token, block_out_permute), dim=1)`
- `position_embeddings = nn.Parameter(torch.ones(1, 197, 768))`
- `z_0_with_class_with_pos = z_0_with_class + position_embeddings`

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1 \mathbf{E}; \mathbf{x}_p^2 \mathbf{E}; \dots; \mathbf{x}_p^N \mathbf{E}] + \mathbf{E}_{pos},$$

$$\mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}, \mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D} \quad (1)$$

$$\mathbf{z}'_{\ell} = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1},$$

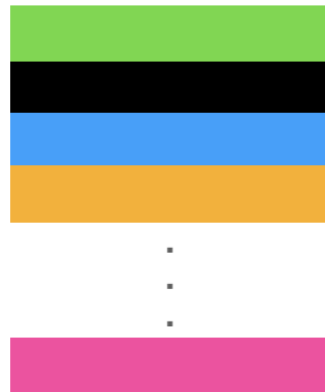
$$\ell = 1 \dots L \quad (2)$$

$$\mathbf{z}_{\ell} = \text{MLP}(\text{LN}(\mathbf{z}'_{\ell})) + \mathbf{z}'_{\ell},$$

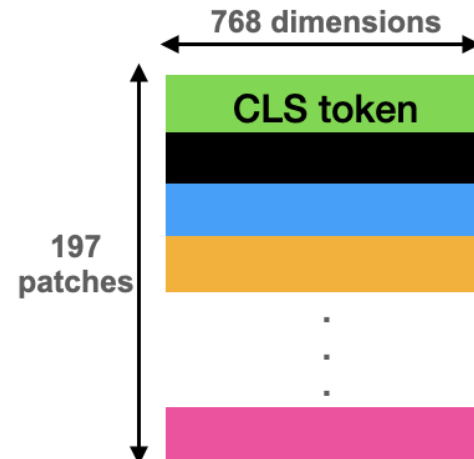
$$\ell = 1 \dots L \quad (3)$$

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0)$$

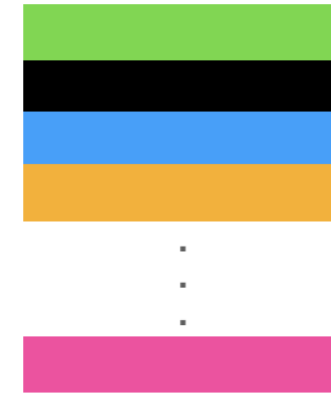
A learnable parameter tensor of size 197x768



Position embedding



Output of size 197x768



Input to Vision Transformer



ViT – Selected Implementation Details – Equation 2.

```
layer_norm = nn.LayerNorm(768)
layer_norm_out = layer_norm(z_0_with_class_with_pos)
multihead_attn = nn.MultiheadAttention(embed_dim = 768, num_heads = 12, batch_first = True)
attn_output, attn_output_weights = multihead_attn(query = layer_norm_out,
                                                    key = layer_norm_out,
                                                    value = layer_norm_out)
```

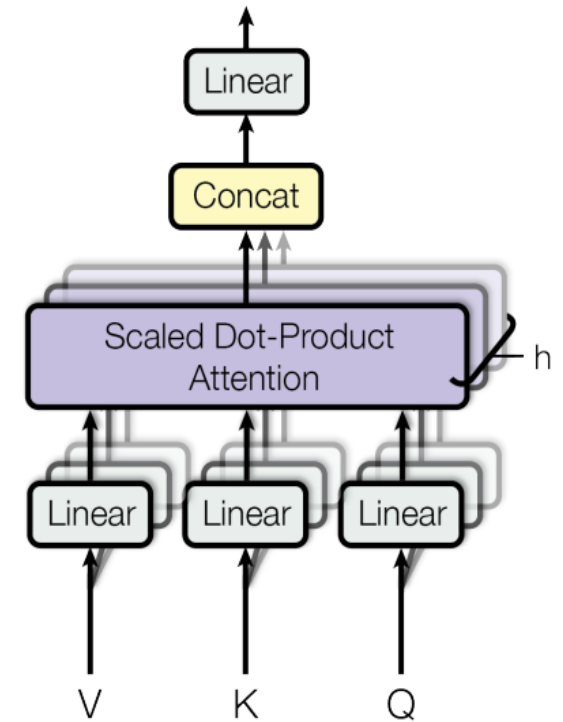
$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1 \mathbf{E}; \mathbf{x}_p^2 \mathbf{E}; \dots; \mathbf{x}_p^N \mathbf{E}] + \mathbf{E}_{\text{pos}}, \quad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}, \mathbf{E}_{\text{pos}} \in \mathbb{R}^{P^2 \times D}$$

$$\mathbf{z}'_\ell = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \quad \ell = 1 \dots L$$

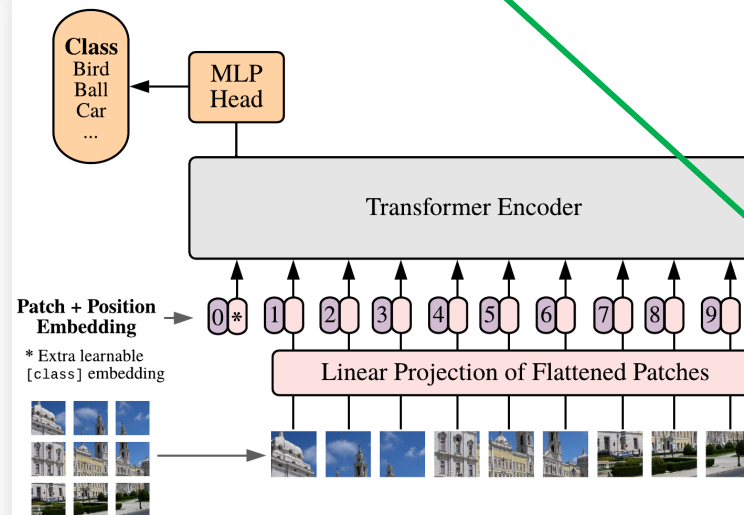
$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, \quad \ell = 1 \dots L$$

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0)$$

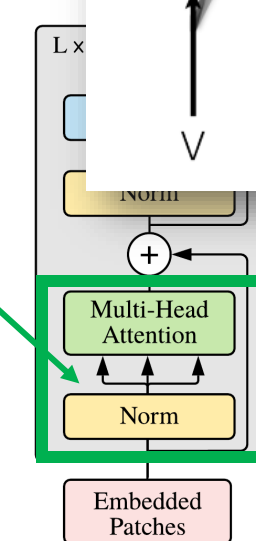
Multi-Head Attention



Vision Transformer (ViT)



Transformer Block



You can continue in a similar way with equations 3 and 4 during the exercise (it is appropriate to organise the ViT blocks into individual classes)

Where to go from here?

<https://github.com/lucidrains/vit-pytorch>

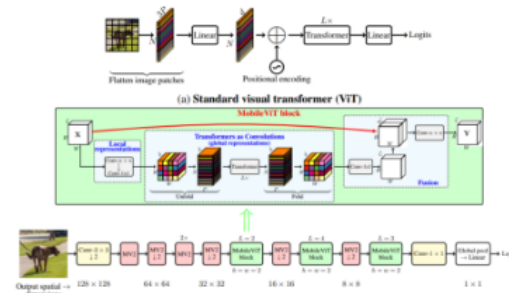
README MIT license

Table of Contents

- [Vision Transformer - Pytorch](#)
- [Install](#)
- [Usage](#)
- [Parameters](#)
- [Simple ViT](#)
- [NaViT](#)
- [Distillation](#)
- [Deep ViT](#)
- [CaiT](#)
- [Token-to-Token ViT](#)
- [CCT](#)
- [Cross ViT](#)
- [PiT](#)
- [LeViT](#)
- [CvT](#)
- [Twins SVT](#)
- [CrossFormer](#)
- [RegionViT](#)
- [ScalableViT](#)
- [SepViT](#)
- [MaxViT](#)
- [NesT](#)
- [MobileViT](#)
- [XCiT](#)
- [Masked Autoencoder](#)
- [Simple Masked Image Modeling](#)
- [Masked Patch Prediction](#)

README MIT license

MobileViT



(b) MobileViT. Here, $\text{Conv-}n \times n$ in the MobileViT block represents a standard $n \times n$ convolution and MV2 refers to MobileNetV2 block. Blocks that perform down-sampling are marked with $\downarrow 2$.

This [paper](#) introduce MobileViT, a light-weight and general purpose vision transformer for mobile devices. MobileViT presents a different perspective for the global processing of information with transformers.

You can use it with the following code (ex. mobilevit_xs)

```
import torch
from vit_pytorch.mobile_vit import MobileViT

mbvit_xs = MobileViT(
    image_size = (256, 256),
    dims = [96, 120, 144],
    channels = [16, 32, 48, 48, 64, 64, 80, 80, 96, 96, 384],
    num_classes = 1000
)

img = torch.randn(1, 3, 256, 256)

pred = mbvit_xs(img) # (1, 1000)
```