

CS181: Computer Vision

Vision Transformer (ViT)

Tuesday, September 29th, 2026



Md Alimoor Reza

Associate Professor of Computer Science

Announcements

- Notebook #3: Underwater Animal Classification
 - It's a 10-class image recognition problem.
 - to submit, download the .ipynb file from Colab
 - directly upload to Blackboard
 - due in two days (by October 2nd)
- Quiz#1 will be released today
 - you will be able to take it offline on Blackboard
 - should be taken within a fixed-hour window (10 hours) during which you are expected to complete the quiz individually. You are free to start anytime within the seven-day period starting today (released on September 29th and due by October 7th)

Review: Transformers



- In 2017, a new mechanism is introduced for context learning called **attention mechanism**
 - more precisely, **self-attention**
- It takes less time to train **advantage**
- Transfer learning on a new task is possible **advantage**
- In subsequent years, it revolutionized the field of AI

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

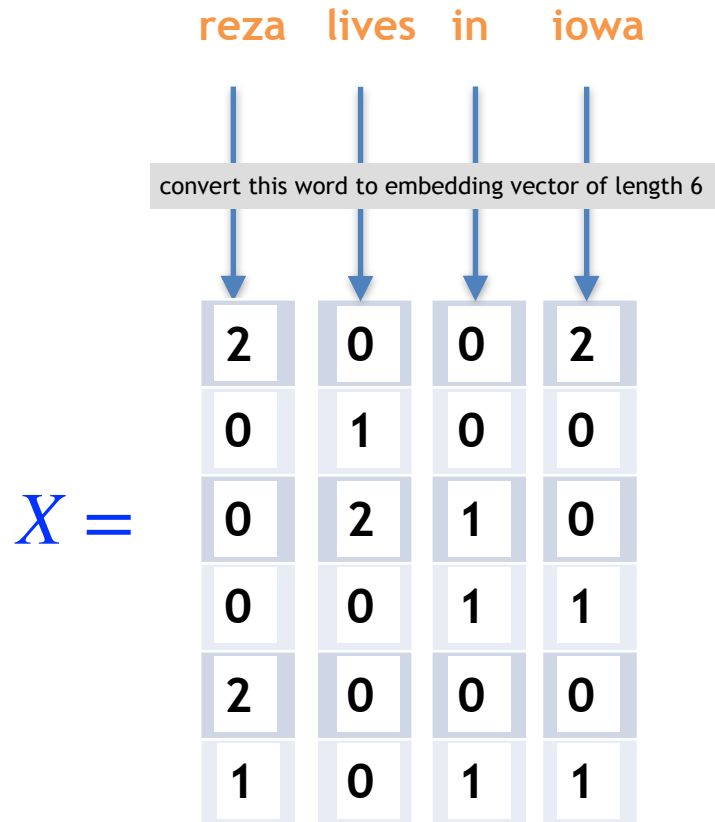
Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

[Attention is all you need - NeurIPS'2017](#)

Review: Self-Attention

- Let's concatenate the vectors x_1 , x_2 , x_3 , and x_4 and build a matrix X from these.



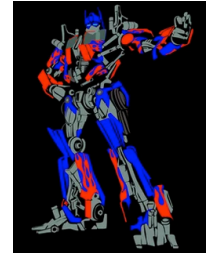
Review: Self-Attention

- These three weight matrices (W_Q , W_K , and W_V) are **learned** during training

W_Q

1	1	0	0	0	0
0	1	0	1	0	0
0	0	1	0	1	1

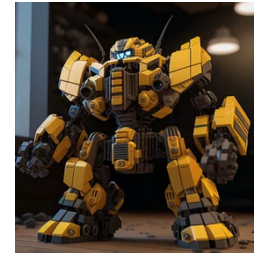
Optimus prime



W_K

0	0	1	0	0	0
0	1	0	0	0	0
1	0	0	0	0	-1

Bumble Bee



W_V

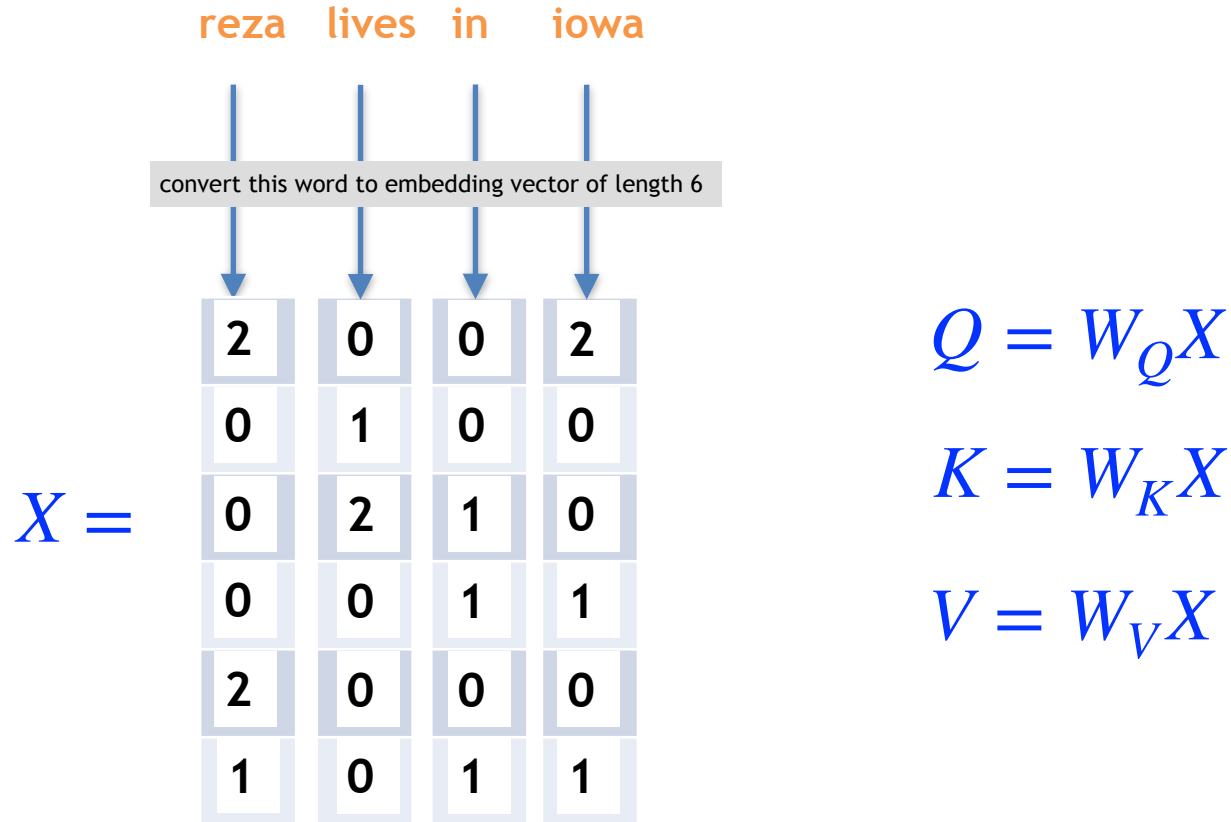
10	0	0	0	0	0
0	0	0	10	0	0
0	10	0	0	0	0

Ironhide



Review: Self-Attention

- Now, let's compute three new matrices Q , K , and V with the help of three weight matrices W_Q , W_K , and W_V . We will multiply input matrix X with each of them in turn as follows:



Review (In-class activity): Self-Attention

- Let's multiply matrices W_Q and X to derive (by matrix-matrix multiplication) Q as follows:

$$Q = W_Q X$$

$=$

1	1	0	0	0	0
0	1	0	1	0	0
0	0	1	0	1	1

W_Q

reza lives in iowa

2	0	0	2
0	1	0	0
0	2	1	0
0	0	1	1
2	0	0	0
1	0	1	1

X

$=$

2	1	0	2
0	1	1	1
3	2	2	1

Review (In-class activity): Self-Attention

- Let's multiply matrices W_K and X to derive (by matrix-matrix multiplication) K as follows:

$$K = W_K X$$

$=$

0	0	1	0	0	0
0	1	0	0	0	0
1	0	0	0	0	-1

W_K

reza lives in iowa

2	0	0	2
0	1	0	0
0	2	1	0
0	0	1	1
2	0	0	0
1	0	1	1

X

$=$

0	2	1	0
0	1	0	0
1	0	-1	1

Review (In-class activity): Self-Attention

- Let's multiply matrices W_V and X to derive (by matrix-matrix multiplication) V as follows:

$$V = W_V X$$

$=$

10	0	0	0	0	0
0	0	0	10	0	0
0	10	0	0	0	0

W_V

reza lives in iowa

2	0	0	2
0	1	0	0
0	2	1	0
0	0	1	1
2	0	0	0
1	0	1	1

X

$=$

20	0	0	20
0	0	10	10
0	10	0	0

Review: Self-Attention

- Finally, attention is calculated using Q, K, and V matrices using the following equation:

$$\text{softmax} \left(\frac{Q \times K^T}{\sqrt{d_k}} \right) V$$

Attention weighted features

=

The diagram illustrates the Self-Attention calculation. It shows the Q, K, and V matrices, the softmax operation, and the resulting attention-weighted features matrix.

Q matrix (Green):

2	1	0	2
0	1	1	1
3	2	2	1

K^T matrix (Purple):

0	0	1
2	1	0
1	0	-1
0	0	1

V matrix (Blue):

20	0	0	20
0	0	10	10
0	10	0	0

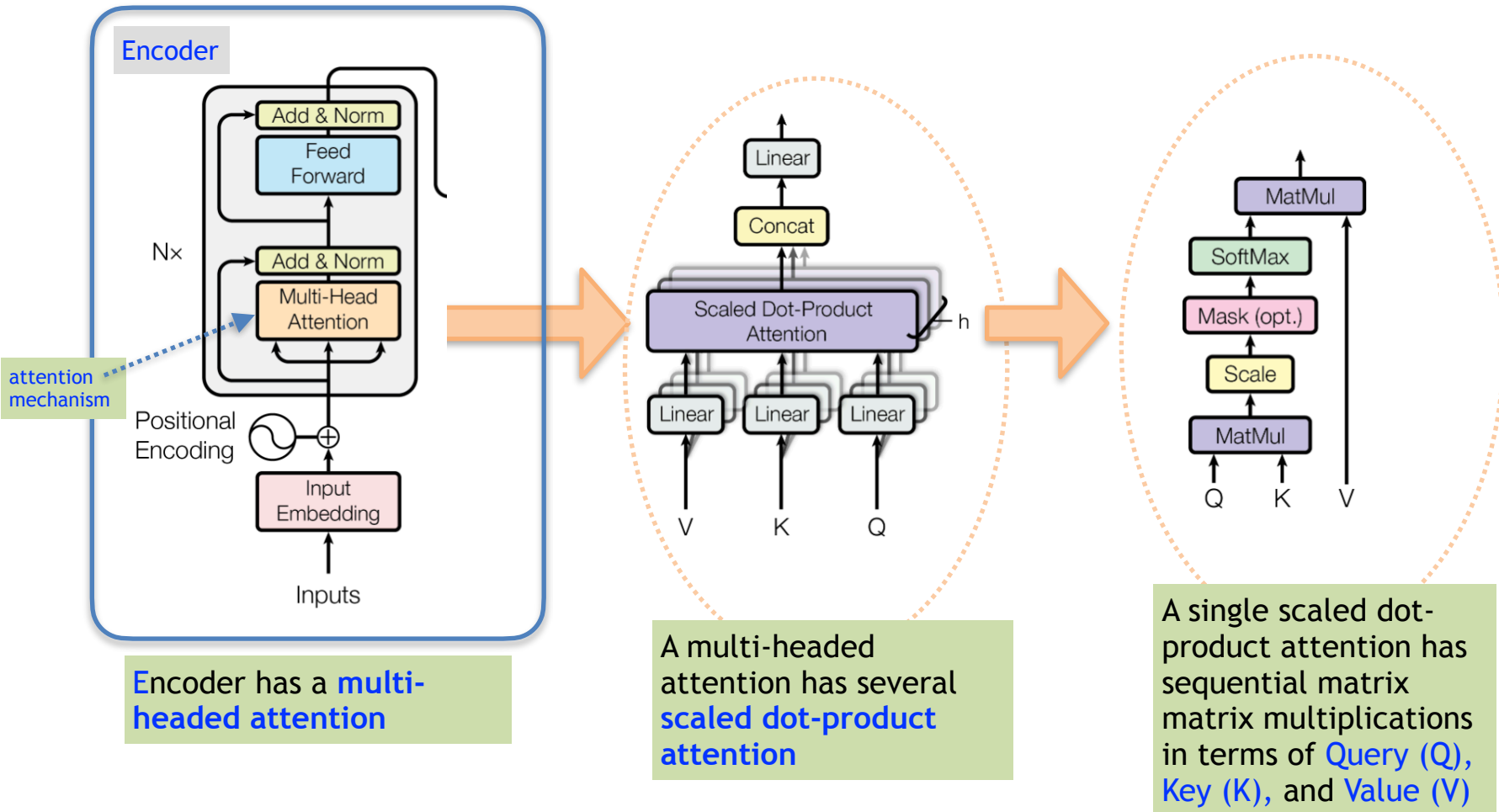
Attention weighted features matrix (Orange):

Review: Figures used in the original paper

Transformers: Encoder



- Lets focus on the encoder to understand what is this **attention mechanism**



Today's agenda

- Vision Transformer (ViT)

Vision Transformer (ViT)

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, neilhoulby}@google.com

- In 2021, the **attention mechanism** is used for computer vision problems
- Excellent results compared to CNN while requiring substantially fewer parameters^{advantage}
- For classification the encoder part of transformer is sufficient^{advantage}

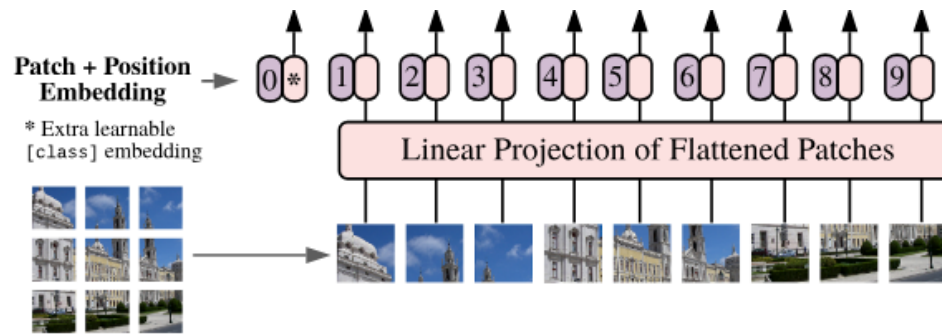
ABSTRACT

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

[An image is worth 16x16 words: transformers for image recognition at scale - ICLR'21](#)

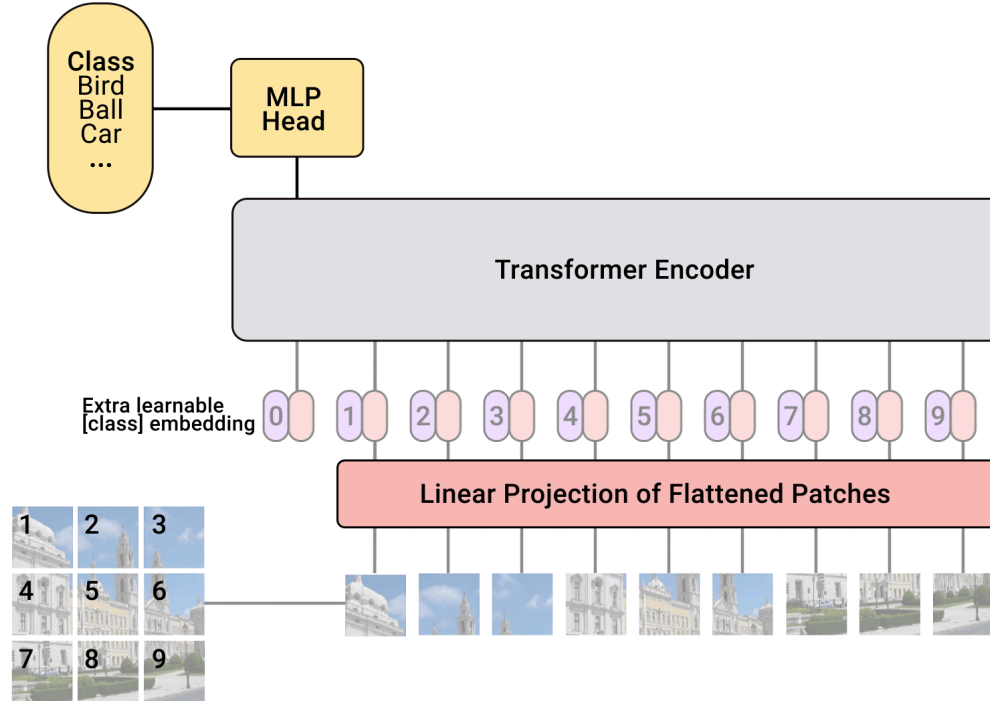
Embedding and Positional Encoding

- Divide an image into 16x16 (hyper-parameter which you can pick and chose). Each patch will contain 256 pixels. There will be several such patches (eg, 9 patches in the image below) for that particular image.



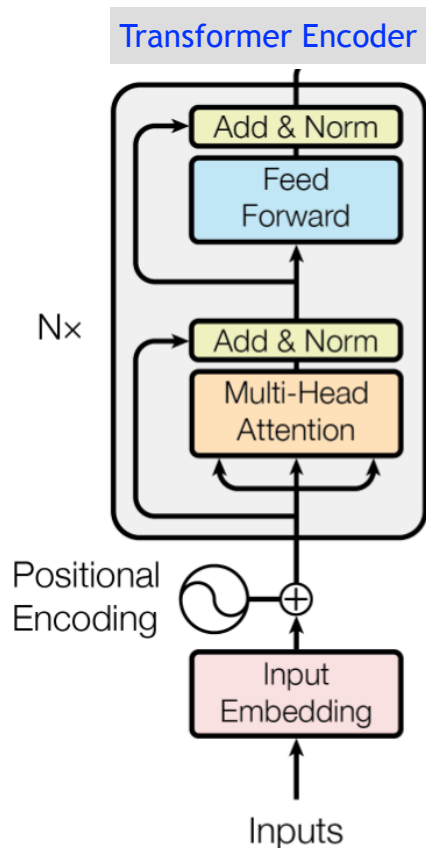
- Linearly embed it to some lower dimension eg, 64-dimensional embedding
- Positional embedding is added to retain the positional information, eg, 1D learnable position
- Classification token is added at zeroth position

Vision Transformer (ViT) for Image Classification

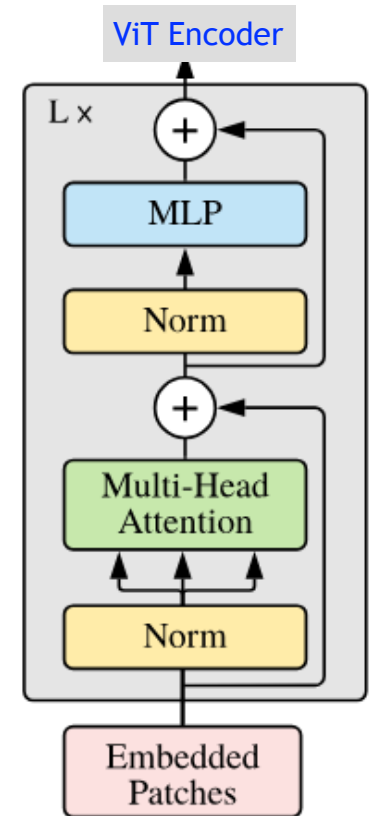


- It's an application of transformer for computer vision problem with fewest possible modifications.
- For image classification task, only need the encoder part of Transformer
- To mimic the setup of a language model, an image is divided into patches (consider each patch as a word token).
- Each patch is converted to fix-sized embedding, and each patch encoding is attached with a positional encoding.

Transformer Encoder vs. ViT Encoder



- Components of encoder are same
- Layers are rearranged
 - Location of norm is important
- Both generate fix-sized embedding
- Both adds positional encoding after embedding generation
- ViT adds an extra **classification token** to the sequence of patches
 - It is learnable parameters (output)



Vision Transformer Comparisons

(PyTorch official implementation)

Vision Transformer (ViT)

Weight	Acc@1	Acc@5	Params	GFLOPS
ViT_B_16_Weights.IMAGENET1K_V1	81.072	95.318	86.6M	17.56
ViT_B_16_Weights.IMAGENET1K_SWAG_E2E_V1	85.304	97.65	86.9M	55.48
ViT_B_16_Weights.IMAGENET1K_SWAG_LINEAR_V1	81.886	96.18	86.6M	17.56
ViT_B_32_Weights.IMAGENET1K_V1	75.912	92.466	88.2M	4.41
ViT_H_14_Weights.IMAGENET1K_SWAG_E2E_V1	88.552	98.694	633.5M	1016.72
ViT_H_14_Weights.IMAGENET1K_SWAG_LINEAR_V1	85.708	97.73	632.0M	167.29
ViT_L_16_Weights.IMAGENET1K_V1	79.662	94.638	304.3M	61.55
ViT_L_16_Weights.IMAGENET1K_SWAG_E2E_V1	88.064	98.512	305.2M	361.99
ViT_L_16_Weights.IMAGENET1K_SWAG_LINEAR_V1	85.146	97.422	304.3M	61.55
ViT_L_32_Weights.IMAGENET1K_V1	76.972	93.07	306.5M	15.38

Swin Transformer (another variant of vision transformer)

Weight	Acc@1	Acc@5	Params	GFLOPS
Swin_B_Weights.IMAGENET1K_V1	83.582	96.64	87.8M	15.43
Swin_S_Weights.IMAGENET1K_V1	83.196	96.36	49.6M	8.74
Swin_T_Weights.IMAGENET1K_V1	81.474	95.776	28.3M	4.49
Swin_V2_B_Weights.IMAGENET1K_V1	84.112	96.864	87.9M	20.32
Swin_V2_S_Weights.IMAGENET1K_V1	83.712	96.816	49.7M	11.55
Swin_V2_T_Weights.IMAGENET1K_V1	82.072	96.132	28.4M	5.94

Vision Transformer Comparisons

(PyTorch official implementation)

Vision Transformer (ViT)

Weight	Acc@1	Acc@5	Params	GFLOPS
ViT_B_16_Weights.IMAGENET1K_V1	81.072	95.318	86.6M	17.56
ViT_B_16_Weights.IMAGENET1K_SWAG_E2E_V1	85.304	97.65	86.9M	55.48
ViT_B_16_Weights.IMAGENET1K_SWAG_LINEAR_V1	81.886	96.18	86.6M	17.56
ViT_B_32_Weights.IMAGENET1K_V1	75.912	92.466	88.2M	4.41
ViT_H_14_Weights.IMAGENET1K_SWAG_E2E_V1	88.552	98.694	633.5M	1016.72
ViT_H_14_Weights.IMAGENET1K_SWAG_LINEAR_V1	85.708	97.73	632.0M	167.29
ViT_L_16_Weights.IMAGENET1K_V1	79.662	94.638	304.3M	61.55
ViT_L_16_Weights.IMAGENET1K_SWAG_E2E_V1	88.064	98.512	305.2M	361.99
ViT_L_16_Weights.IMAGENET1K_SWAG_LINEAR_V1	85.146	97.422	304.3M	61.55
ViT_L_32_Weights.IMAGENET1K_V1	76.972	93.07	306.5M	15.38

ANTI COUNTERFEIT RECOGNITION
 - ViT TRAINING

	PARAM	TRAINING MEMORY	INPUT IMAGE
BASE ViT-B/16 (PATCH)	86.5	6.83 GB	384^2
LARGE ViT-L/16 (PATCH)	304.3M	19.13 GB	384^2
ViT-H/14 (PATCH)	632.0M	18.86 GB	224^2

QUADRO RTX 6000 WITH 24 GB MEMORY
 JOLTEON

Vision Transformer (ViT) for Classification

- Torch vision has pretrained weights for various vision transformers

[Day11 Notes: image classification with vision transformer \(ViT\)](#)

- Let me highlight some important parts of the official source code

Vision Transformer (ViT) PyTorch source code

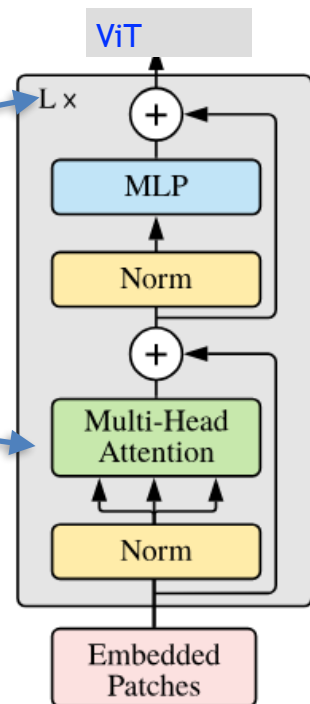
- When we instantiate an object of the ViT transformer network:

```
# download PyTorch's own implementation of Vision Transformer model trained on ImageNet dataset
net = models.vit_b_16(pretrained=True)
self.net = net
```

- It creates an instance of `vision_transformer` with the following parameters:

```
weights = ViT_B_16_Weights.verify(weights)

return _vision_transformer(
    patch_size=16,
    num_layers=12,
    num_heads=12,
    hidden_dim=768,
    mlp_dim=3072,
    weights=weights,
    progress=progress,
    **kwargs,
)
```



Vision Transformer (ViT) PyTorch source code

- Here is the structure of the **vision_transformer** class with **12 encoder-blocks**. Each encoder block has 12 **MultiheadedAttention** + 1 **MLPBlock**

```
ViT(  
  (net): VisionTransformer(  
    (conv_proj): Conv2d(3, 768, kernel_size=(16, 16), stride=(16, 16))  
    (encoder): Encoder(  
      (dropout): Dropout(p=0.0, inplace=False)  
      (layers): Sequential(  
        (encoder_layer_0): EncoderBlock(  
          (ln_1): LayerNorm((768,)), eps=1e-06, elementwise_affine=True)          12 MultiheadedAttention  
          (self_attention): MultiheadAttention(  
            (out_proj): NonDynamicallyQuantizableLinear(in_features=768, out_features=768, bias=True)  
          )  
          (dropout): Dropout(p=0.0, inplace=False)  
          (ln_2): LayerNorm((768,)), eps=1e-06, elementwise_affine=True)  
          (mlp): MLPBlock(  
            (0): Linear(in_features=768, out_features=3072, bias=True)          1 MLPBlock  
            (1): GELU(approximate='none')  
            (2): Dropout(p=0.0, inplace=False)  
            (3): Linear(in_features=3072, out_features=768, bias=True)  
            (4): Dropout(p=0.0, inplace=False)  
          )  
        )  
        (encoder_layer_1): EncoderBlock(  
          (ln_1): LayerNorm((768,)), eps=1e-06, elementwise_affine=True)  
          (self_attention): MultiheadAttention(  
            (out_proj): NonDynamicallyQuantizableLinear(in_features=768, out_features=768, bias=True)  
          )  
        )  
        (encoder_layer_2): ...  
        ...  
      )  
    )  
  )  
)
```

Vision Transformer (ViT) PyTorch source code

- Here is the structure of the `vision_transformer` class:

(encoder_layer_09): ...

...

(encoder_layer_10): ...

```
(encoder_layer_11): EncoderBlock(  
  (ln_1): LayerNorm((768,), eps=1e-06, elementwise_affine=True) 12 MultiheadedAttention  
  (self_attention): MultiheadAttention(  
    (out_proj): NonDynamicallyQuantizableLinear(in_features=768, out_features=768, bias=True)  
  )  
  (dropout): Dropout(p=0.0, inplace=False)  
  (ln_2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)  
  (mlp): MLPBlock(  
    (0): Linear(in_features=768, out_features=3072, bias=True) 1 MLPBlock  
    (1): GELU(approximate='none')  
    (2): Dropout(p=0.0, inplace=False)  
    (3): Linear(in_features=3072, out_features=768, bias=True)  
    (4): Dropout(p=0.0, inplace=False)  
  )  
)  
)  
(ln): LayerNorm((768,), eps=1e-06, elementwise_affine=True)  
)  
(heads): Sequential(  
  (head): Linear(in_features=768, out_features=1000, bias=True)  
)  
)
```

Vision Transformer (ViT) PyTorch source code

```
main ▾ vision / torchvision / models / vision_transformer.py   
Blame 864 lines (760 loc) · 31.4 KB     ▾  
  
✓ class EncoderBlock(nn.Module):  
    """Transformer encoder block."""  
  
    def __init__(  
        self,  
        num_heads: int,  
        hidden_dim: int,  
        mlp_dim: int,  
        dropout: float,  
        attention_dropout: float,  
        norm_layer: Callable[..., torch.nn.Module] = partial(nn.LayerNorm, eps=1e-6),  
    ):  
        super().__init__()  
        self.num_heads = num_heads  
  
        # Attention block  
        self.ln_1 = norm_layer(hidden_dim)  
        self.self_attention = nn.MultiheadAttention(hidden_dim, num_heads, dropout=attention_dropout, batch_first=True)  
        self.dropout = nn.Dropout(dropout)  
  
        # MLP block  
        self.ln_2 = norm_layer(hidden_dim)  
        self.mlp = MLPBlock(hidden_dim, mlp_dim, dropout)  
  
    def forward(self, input: torch.Tensor):  
        torch._assert(input.dim() == 3, f"Expected (batch_size, seq_length, hidden_dim) got {input.shape}")  
        x = self.ln_1(input)  
        x, _ = self.self_attention(x, x, x, need_weights=False)  
        x = self.dropout(x)  
        x = x + input
```

Vision Transformer (ViT) PyTorch source code

This MultiheadAttention layer implements the original architecture described in the [Attention Is All You Need](#) paper. The intent of this layer is as a reference implementation for foundational understanding and thus it contains only limited features relative to newer architectures. Given the fast pace of innovation in transformer-like architectures, we recommend exploring this [tutorial](#) to build efficient layers from building blocks in core or using higher level libraries from the [PyTorch Ecosystem](#).

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.

<https://docs.pytorch.org/docs/2.14/generated/torch.nn.MultiheadAttention.html>

Vision Transformer (ViT) PyTorch source code

- Here is an example of self-attention computation from random query, key, and value matrices using `scaled_dot_product_attention` function

```
▶ torch.manual_seed(2)
import torch
import torch.nn as nn
import torch.nn.functional as F
device = "cuda" if torch.cuda.is_available() else "cpu"

# Multiheaded attention uses scaled_dot_product_attention() function from PyTorch Functional module
query_matrix = torch.randn(1, 3, 4, device=device)
key_matrix    = torch.randn(1, 3, 4, device=device)
value_matrix  = torch.randn(1, 3, 4, device=device)

F.scaled_dot_product_attention(query_matrix, key_matrix, value_matrix)

... tensor([[[[-0.4066,  0.2002,  0.1344, -0.4414],
               [-0.2376,  0.3583,  0.2269, -0.3452],
               [-0.4472,  0.1769,  0.1199, -0.4228]]]])
```

<https://docs.pytorch.org/docs/2.14/generated/torch.nn.MultiheadAttention.html>

Vision Transformer (ViT) for Classification

- Please explore the ViT model and run some test images through it

[Day11 Notes: image classification with vision transformer \(ViT\)](#)