Transformer

Transformer

In this session

- We will walkthrough the codes of the vision transformer model introduced in An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale.
- The walkthrough will be done with powerpoint & google colab.
- For those who does not have the link : https://colab.research.google.com/drive/1CS4JAXRY3dl2nFfqlFZTHRn8f0PbK2nQ?usp=sharing

Transformer

About

- Follows the Encoder-Decoder of the seq2seq models with out using any recurrence.
- Currently used as the state-of-the-art model in various NLP and Computer Vision tasks.

Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

Transformer

Overall Architecture

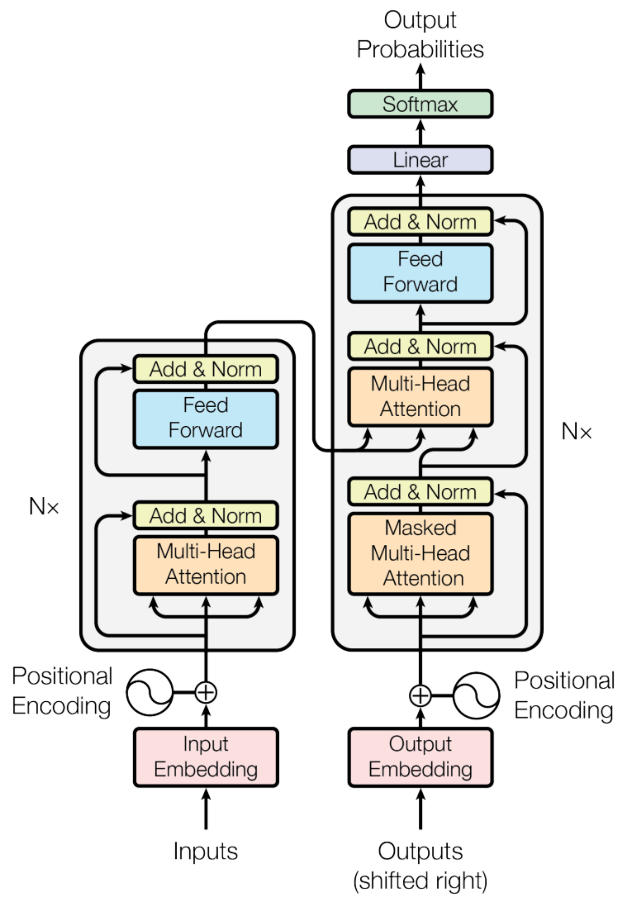


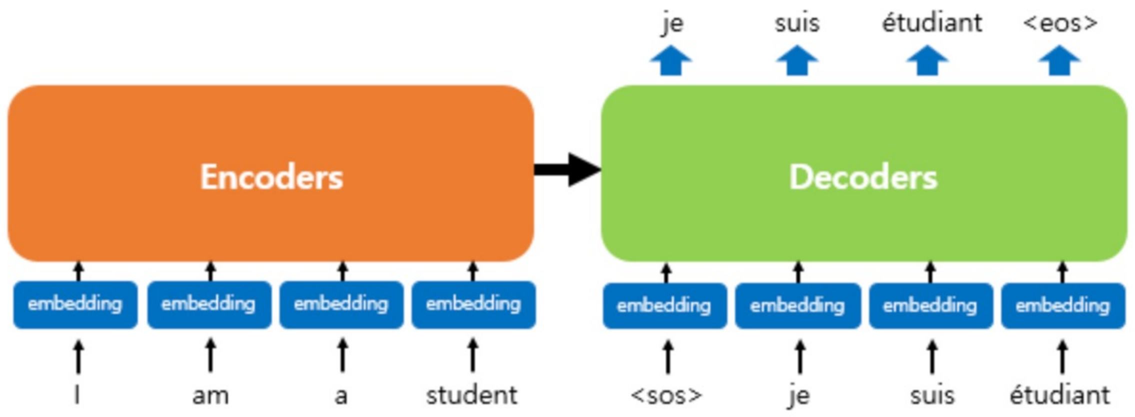
Figure 1: The Transformer - model architecture.

Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

Transformer

Encoder-Decoder

- The encoder takes in the tokenized inputs and encodes the information of the input. This can be thought as the encoder is understanding the context of the input.
- The decoder takes the input and the context from the encoder to decode the input to a certain sequence to solve the task.

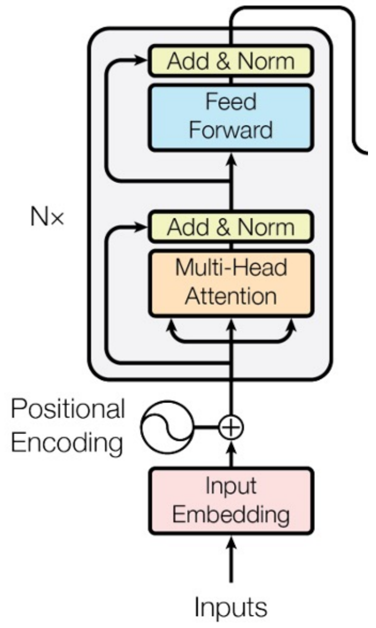


Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).
https://wikidocs.net/31379

Transformer

Encoder

- The encoder takes in the tokenized inputs and encodes the information of the input. This can be thought as the encoder is understanding the context of the input.

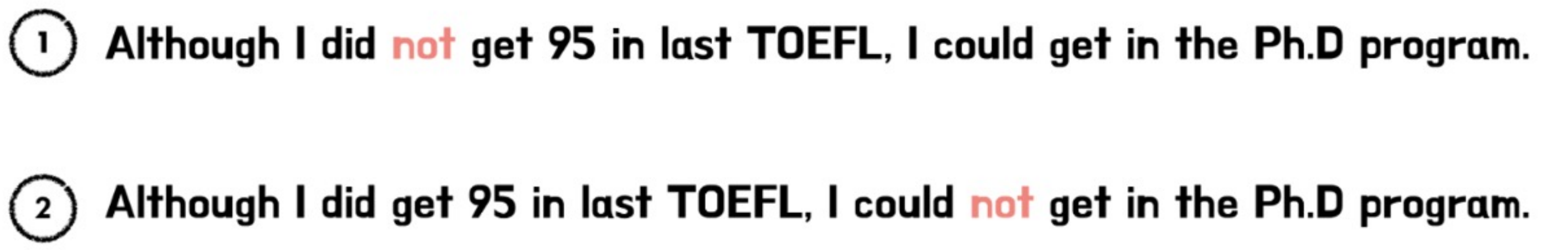


Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

Transformer

Encoder-Positional Encoding

- As the transformer does not have any positional information, we need to add a positional encoding to the input embeddings.
- Without the positional encoding, the transformer should always output the same output even after the input is permuted randomly.



① Although I did not get 95 in last TOEFL, I could get in the Ph.D program.

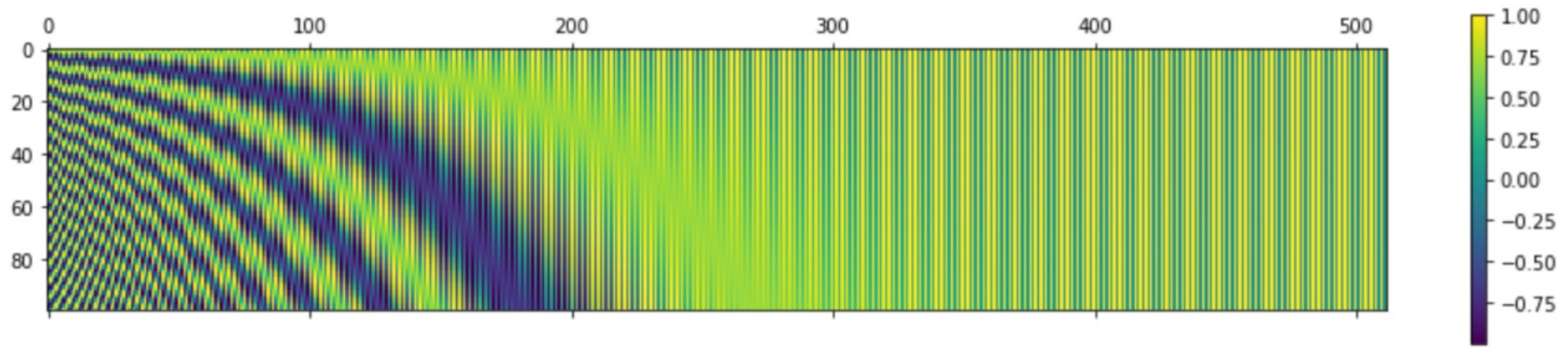
② Although I did get 95 in last TOEFL, I could not get in the Ph.D program.

Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).
https://www.blossominkyung.com/deeplearning/transfomer-positional-encoding

Transformer

Encoder-Positional Encoding

- Learnable Embedding : Let the model learn the positional embedding by setting the embedding as a learnable parameter.
- Sinusoidal Embedding : Use the Sine and Cosine Function and add the values from the sinusoids at the positions of the input embeddings.



Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).
https://gaussian37.github.io/dl-concept-positional_encoding/

Transformer

Encoder-Positional Encoding

- Sinusoidal Embedding : Use the Sine and Cosine Function and add the values from the sinusoids at the positions of the input embeddings.
- pos : position of the data.
- i : position of dimension.

$$\mathrm{PE}_{(\mathrm{pos},2i)} = \sin\left(\frac{\mathrm{pos}}{10000^{2i/d_{\mathrm{model}}}}\right)$$

$$\mathrm{PE}_{(\mathrm{pos},2i+1)} = \cos\left(\frac{\mathrm{pos}}{10000^{2i/d_{\mathrm{model}}}}\right)$$

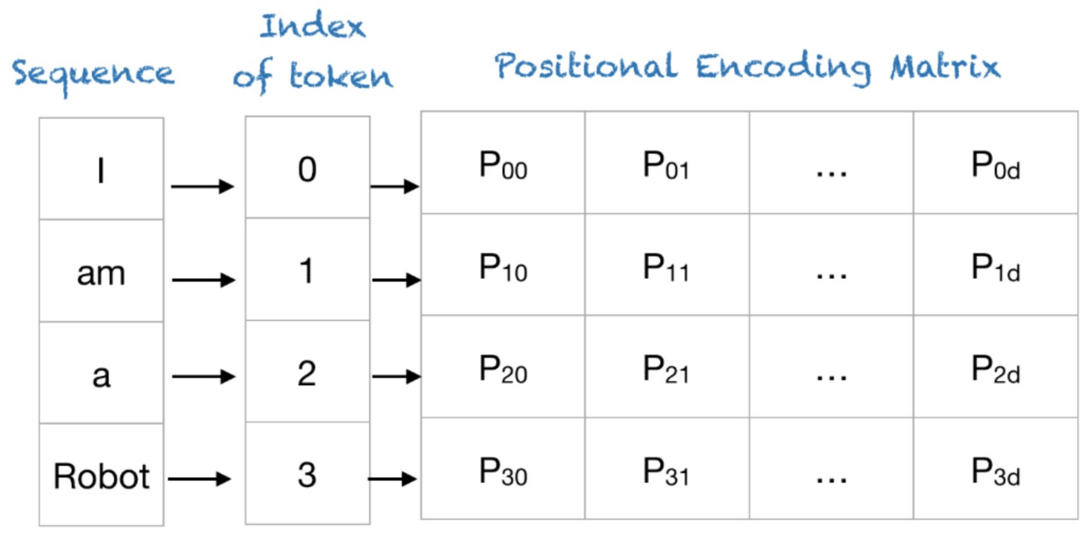
$$0 \leq i \leq d_{\mathrm{model}}/2$$

Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).
https://gaussian37.github.io/dl-concept-positional_encoding/

Transformer

Encoder-Positional Encoding

- Sinusoidal Embedding : Use the Sine and Cosine Function and add the values from the sinusoids at the positions of the input embeddings.
- Example when d_{model} = 4, frequency bound : 100



Sequence	Index of token				
I	0	P_{00}	P_{01}	…	P_{0d}
am	1	P_{10}	P_{11}	…	P_{1d}
a	2	P_{20}	P_{21}	…	P_{2d}
Robot	3	P_{30}	P_{31}	…	P_{3d}

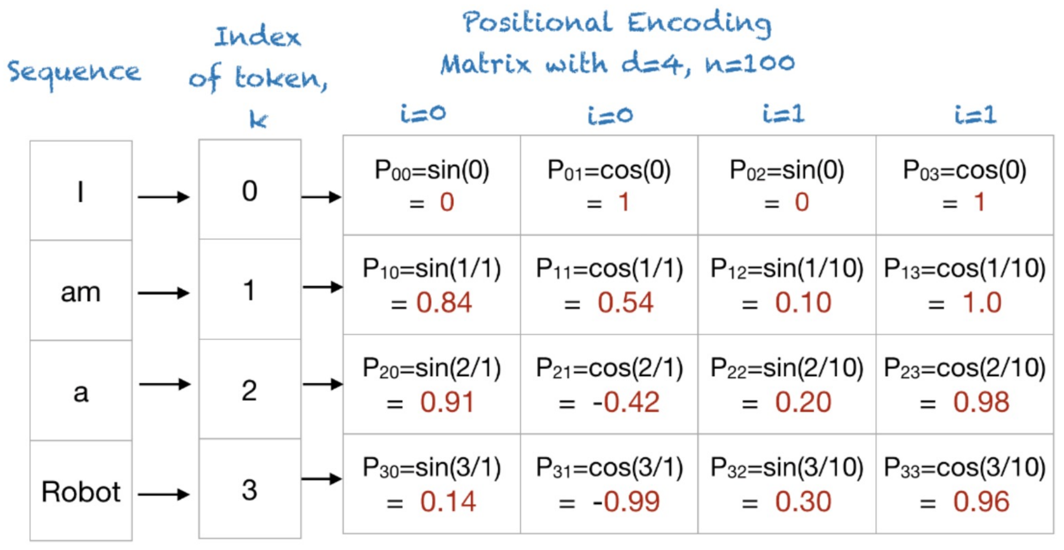
Positional Encoding Matrix for the sequence 'I am a robot'

Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).
https://gaussian37.github.io/dl-concept-positional_encoding/

Transformer

Encoder-Positional Encoding

- Sinusoidal Embedding : Use the Sine and Cosine Function and add the values from the sinusoids at the positions of the input embeddings.
- Example when d_{model} = 4, frequency bound : 100



Sequence	Index of token, k	Positional Encoding Matrix with d=4, n=100			
		i=0	i=0	i=1	i=1
I	0	P_{00}=sin(0) = 0	P_{01}=cos(0) = 1	P_{02}=sin(0) = 0	P_{03}=cos(0) = 1
am	1	P_{10}=sin(1/1) = 0.84	P_{11}=cos(1/1) = 0.54	P_{12}=sin(1/10) = 0.10	P_{13}=cos(1/10) = 1.0
a	2	P_{20}=sin(2/1) = 0.91	P_{21}=cos(2/1) = -0.42	P_{22}=sin(2/10) = 0.20	P_{23}=cos(2/10) = 0.98
Robot	3	P_{30}=sin(3/1) = 0.14	P_{31}=cos(3/1) = -0.99	P_{32}=sin(3/10) = 0.30	P_{33}=cos(3/10) = 0.96

Positional Encoding Matrix for the sequence 'I am a robot'

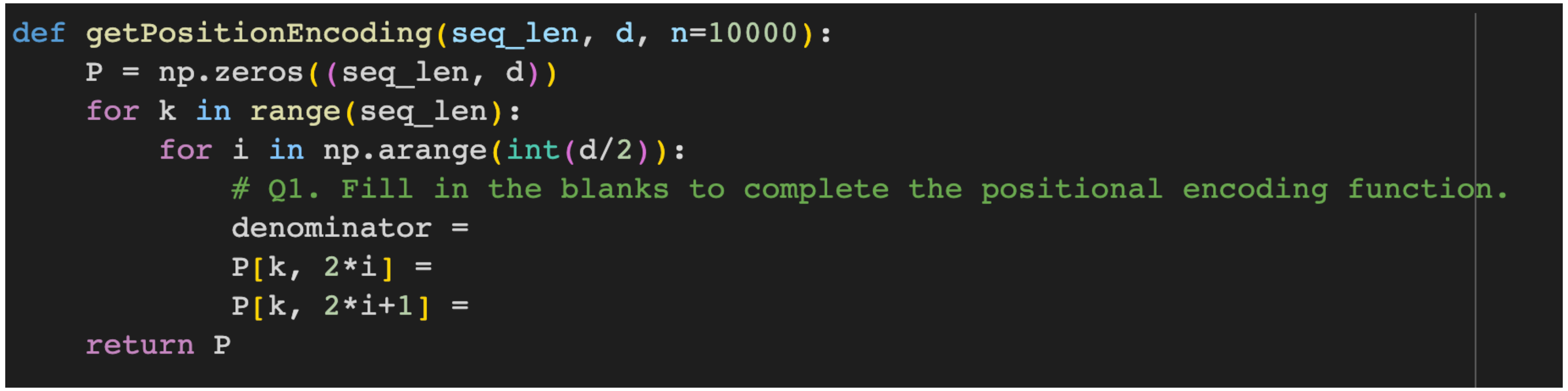
Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).
https://gaussian37.github.io/dl-concept-positional_encoding/

Transformer

Encoder-Positional Encoding

Q1. Positional Encoding

Fill in the blanks of the function getPositionEncoding() that outputs the values from previous slides.



```
def getPositionEncoding(seq_len, d, n=10000):
    P = np.zeros((seq_len, d))
    for k in range(seq_len):
        for i in np.arange(int(d/2)):
            # Q1. Fill in the blanks to complete the positional encoding function.
            denominator =
            P[k, 2*i] =
            P[k, 2*i+1] =
    return P
```

Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

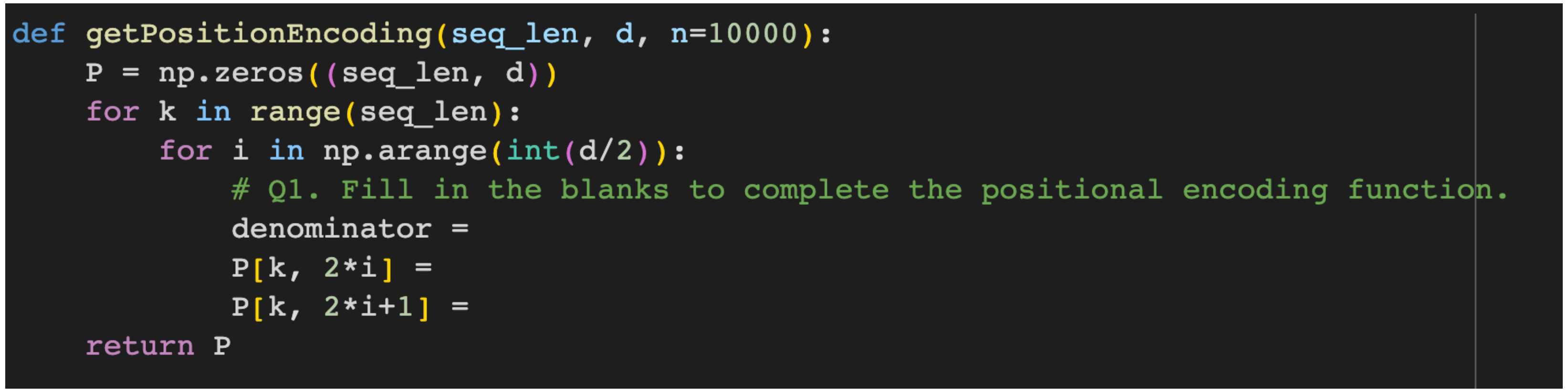
Transformer

Encoder-Positional Encoding

Q1. Positional Encoding

Fill in the blanks of the function getPositionEncoding() that outputs the values from previous slides.

Hint: Try comparing the values from slide 13. (d = 4, n = 100 case)

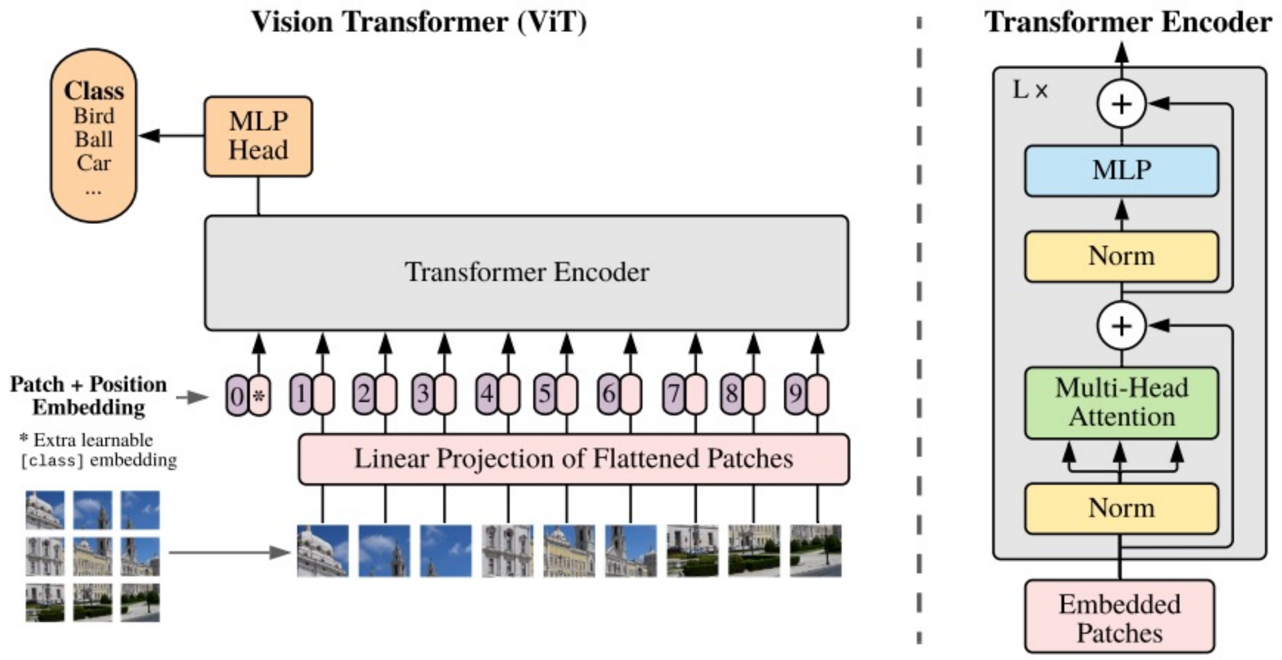


```
def getPositionEncoding(seq_len, d, n=10000):
    P = np.zeros((seq_len, d))
    for k in range(seq_len):
        for i in np.arange(int(d/2)):
            # Q1. Fill in the blanks to complete the positional encoding function.
            denominator =
            P[k, 2*i] =
            P[k, 2*i+1] =
    return P
```

Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

Vision Transformer

Overall Architecture

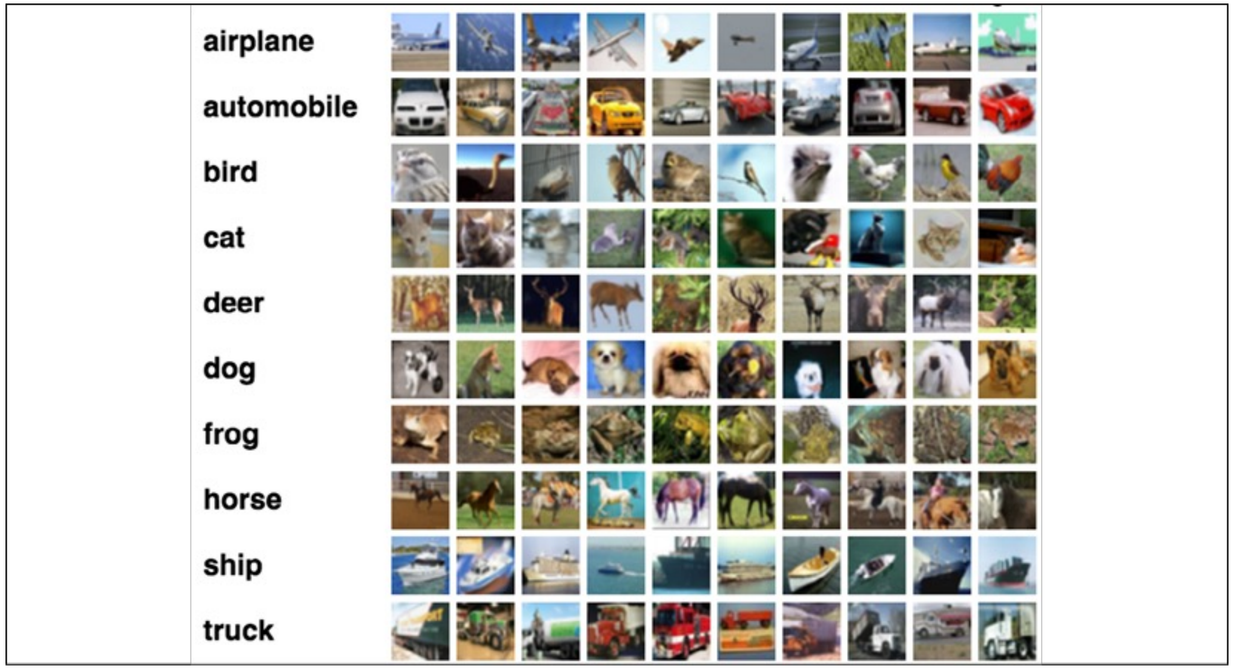


Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929* (2020).

Vision Transformer

Dataset

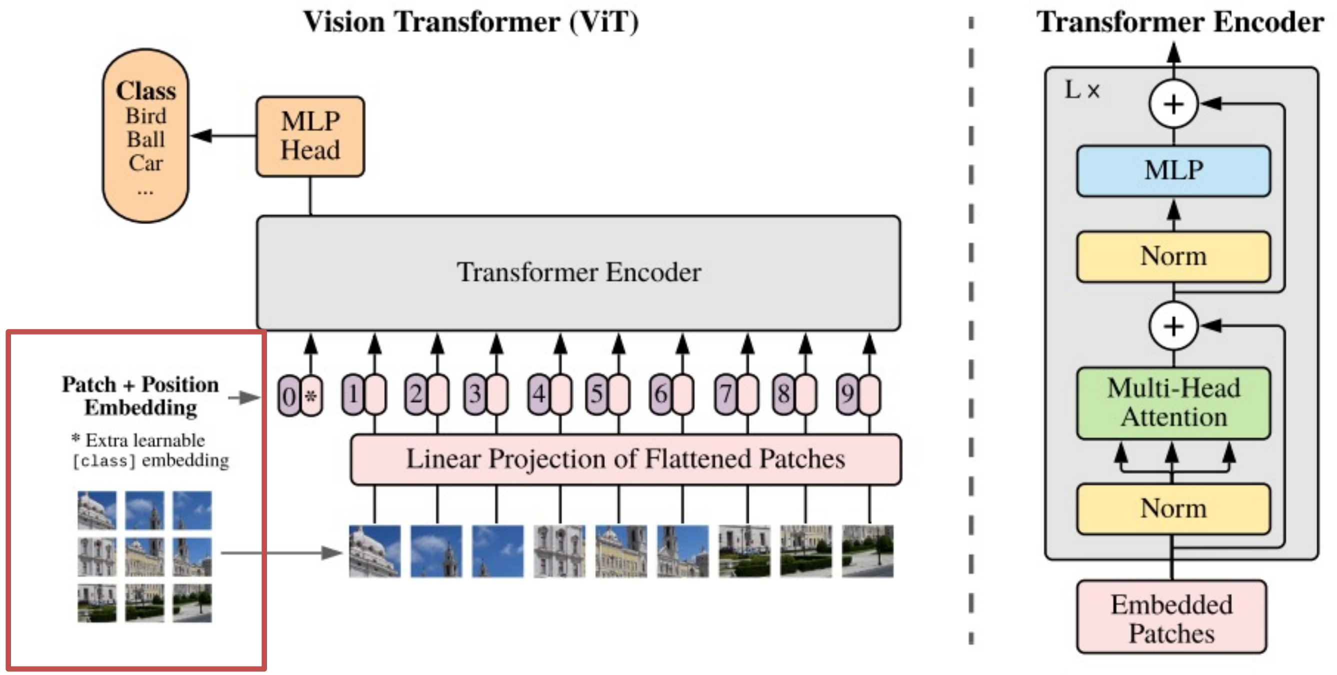
- Cifar10 dataset : A dataset containing tiny images of size 32x32 from 10 classes.
- The dataset was widely used for image classification and is easy to download so we will use this dataset in this session.



Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929* (2020).

Vision Transformer

Patch Embedding

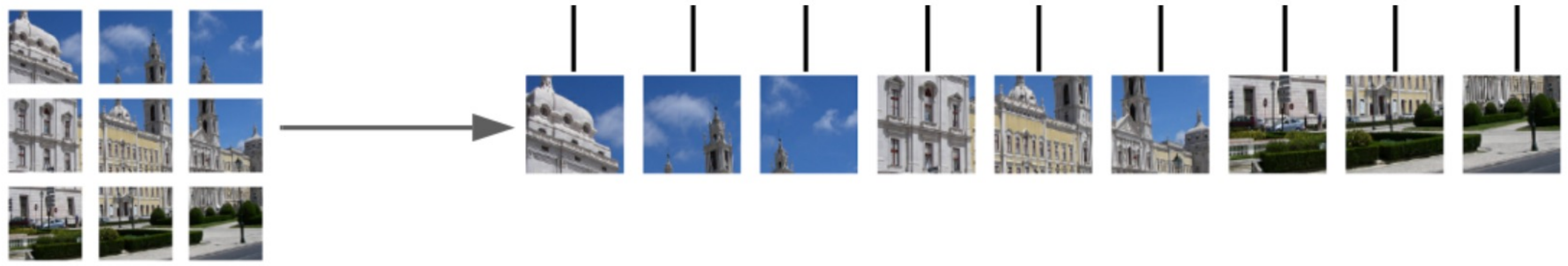


Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929* (2020).

Vision Transformer

Patch Embedding

- To process images with a transformer, we need to embed the image into a sequence of inputs.
- As an image can be understood as a sequence of pixels, we will define non-overlapping patches as a single word.



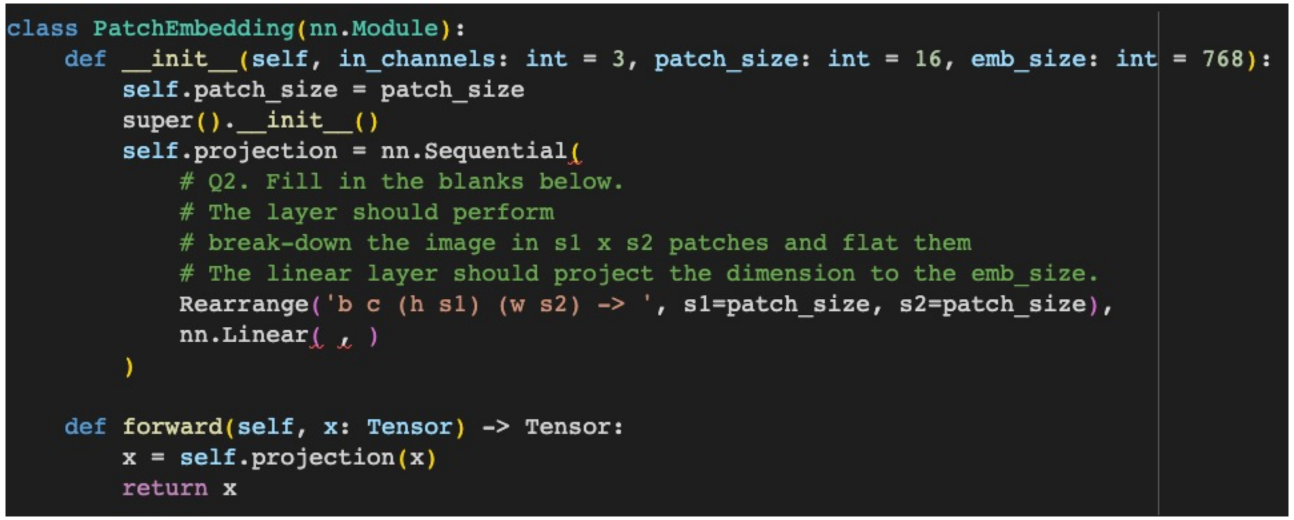
Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929* (2020).

Vision Transformer

Patch Embedding

Q2. Patch Embedding

Fill in the blanks in self.projection in the class PatchEmbedding() which will be used to embed the input images into patches and project them into emb_size dimensions.



```python
class PatchEmbedding(nn.Module):
    def __init__(self, in_channels: int = 3, patch_size: int = 16, emb_size: int = 768):
        self.patch_size = patch_size
        super().__init__()
        self.projection = nn.Sequential(
            # Q2. Fill in the blanks below.
            # The layer should perform
            # break-down the image in s1 x s2 patches and flat them
            # The linear layer should project the dimension to the emb_size.
            Rearrange('b c (h s1) (w s2) -> ', s1=patch_size, s2=patch_size),
            nn.Linear( , )
        )

    def forward(self, x: Tensor) -> Tensor:
        x = self.projection(x)
        return x
```

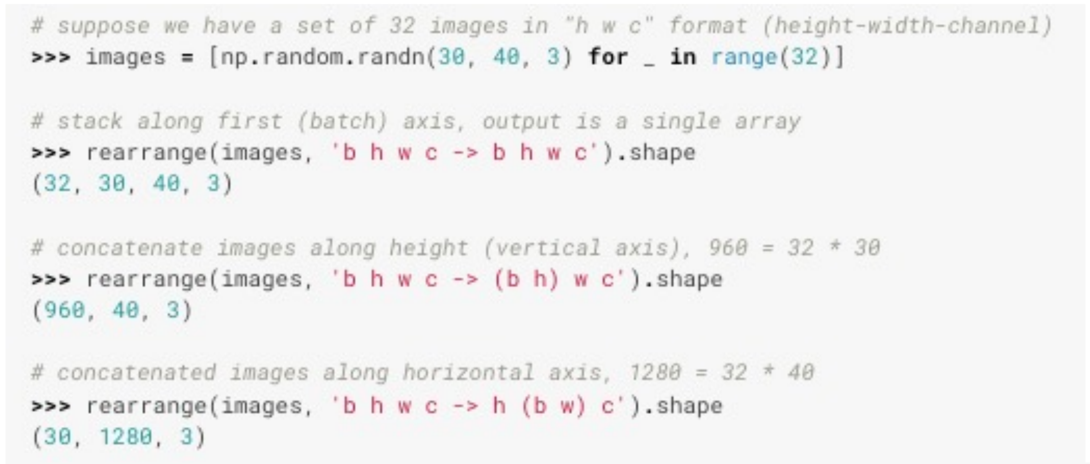
Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929* (2020).

Vision Transformer

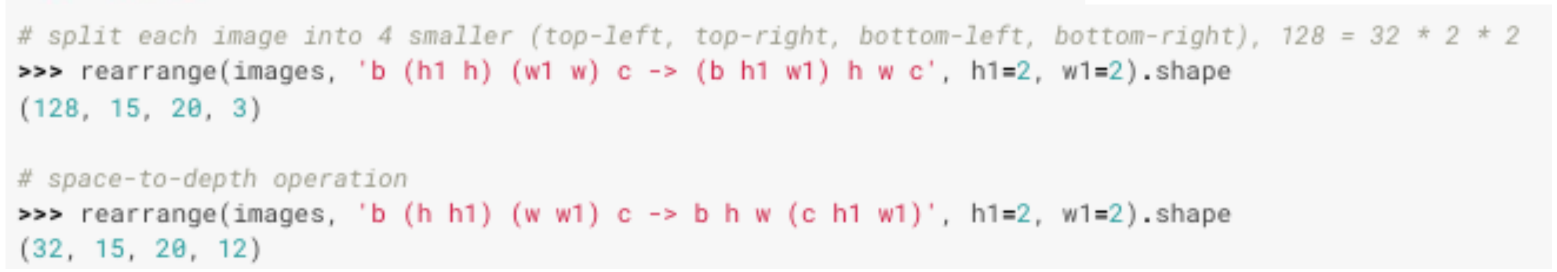
Patch Embedding

Q2. Patch Embedding

To understand the usage of Rearrange from the einops library, these are some examples.



```
# suppose we have a set of 32 images in "h w c" format (height-width-channel)
>>> images = [np.random.randn(30, 40, 3) for _ in range(32)]

# stack along first (batch) axis, output is a single array
>>> rearrange(images, 'b h w c -> b h w c').shape
(32, 30, 40, 3)

# concatenate images along height (vertical axis), 960 = 32 * 30
>>> rearrange(images, 'b h w c -> (b h) w c').shape
(960, 40, 3)

# concatenated images along horizontal axis, 1280 = 32 * 40
>>> rearrange(images, 'b h w c -> h (b w) c').shape
(30, 1280, 3)
```



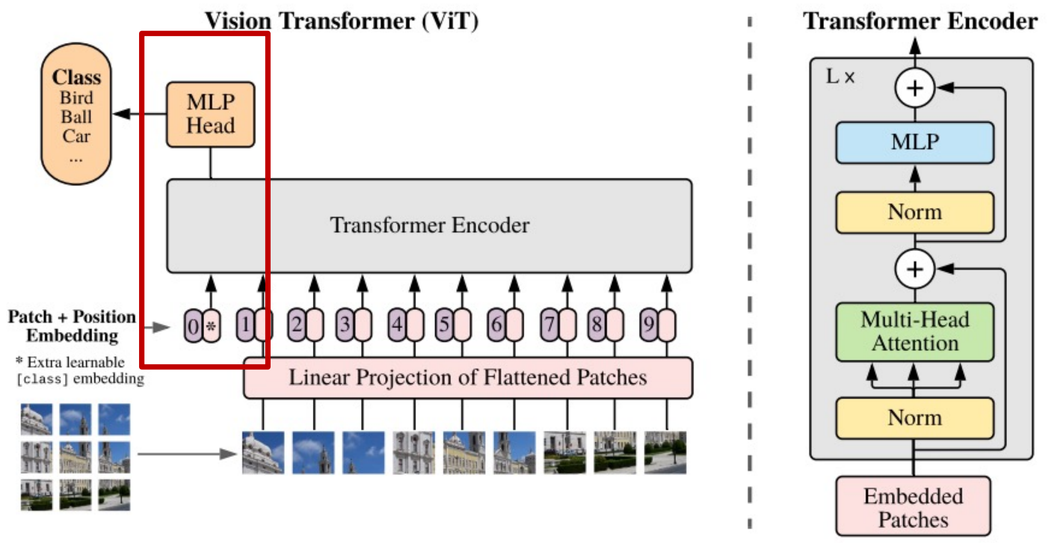
```
# split each image into 4 smaller (top-left, top-right, bottom-left, bottom-right), 128 = 32 * 2 * 2
>>> rearrange(images, 'b (h1 h) (w1 w) c -> (b h1 w1) h w c', h1=2, w1=2).shape
(128, 15, 20, 3)

# space-to-depth operation
>>> rearrange(images, 'b (h h1) (w w1) c -> b h w (c h1 w1)', h1=2, w1=2).shape
(32, 15, 20, 12)
```

Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929* (2020).

Vision Transformer

Class Token

- After the input is processed through the transformer encoder, a MLP layer takes the class token as the input to estimate the class. For this we should concatenate a class token to the embedded features.



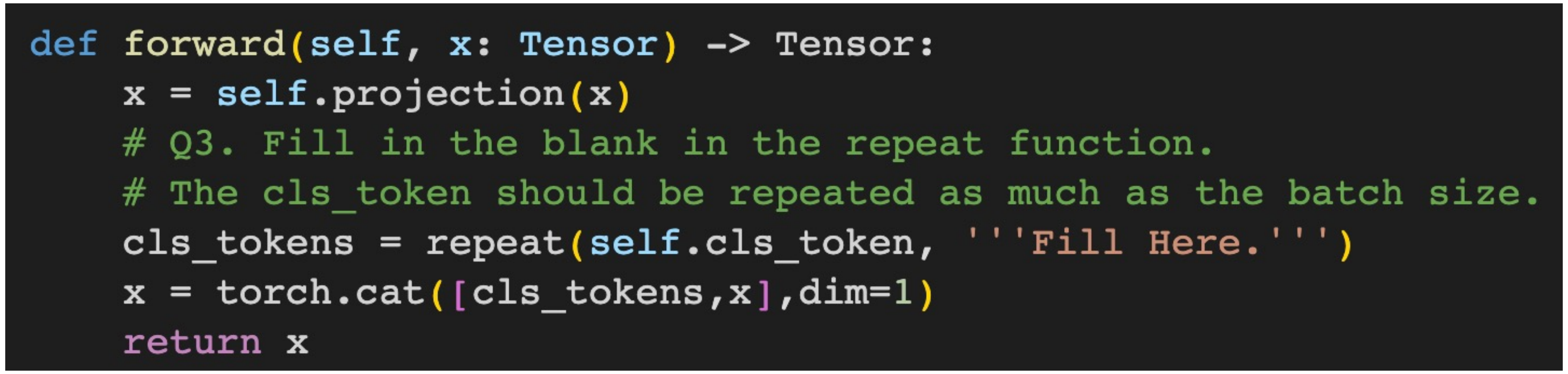
Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929* (2020).

Vision Transformer

Class Token

Q3. Adding the class token

Fill in the blanks in the repeat function. The cls_token should be repeated as much as the batch size.



```python
def forward(self, x: Tensor) -> Tensor:
    x = self.projection(x)
    # Q3. Fill in the blank in the repeat function.
    # The cls_token should be repeated as much as the batch size.
    cls_tokens = repeat(self.cls_token, '''Fill Here.''')
    x = torch.cat([cls_tokens,x],dim=1)
    return x
```

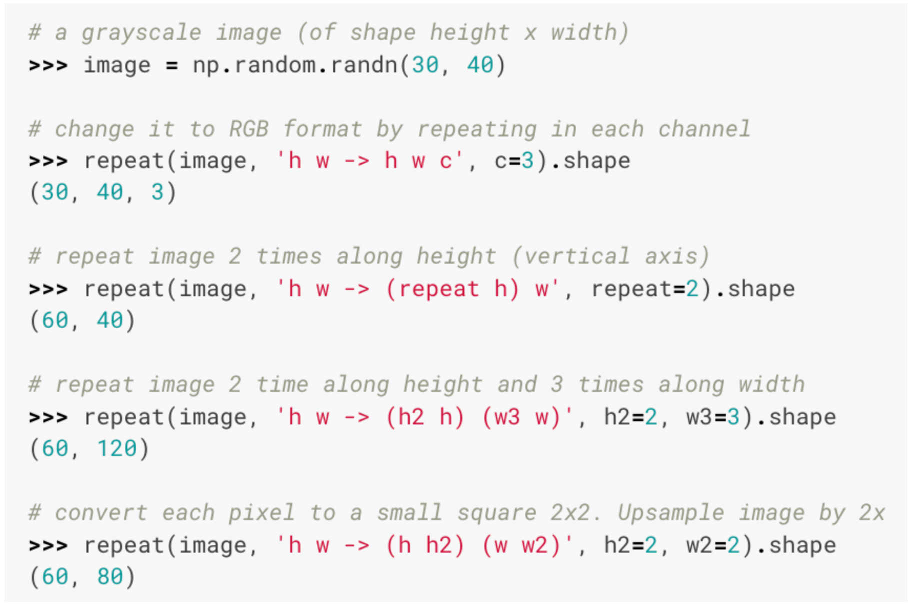
Vision Transformer

Patch Embedding

Q3. Adding the class token

To understand the usage of repeat from the einops library, these are some examples.

Hint : try using the () and b which is defined as the batch size.



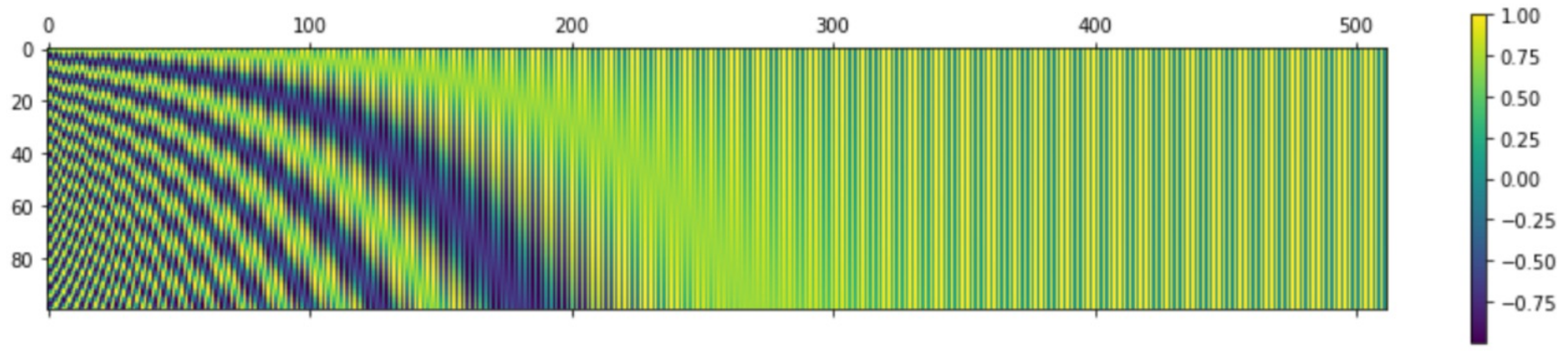
```
# a grayscale image (of shape height x width)
>>> image = np.random.randn(30, 40)

# change it to RGB format by repeating in each channel
>>> repeat(image, 'h w -> h w c', c=3).shape
(30, 40, 3)

# repeat image 2 times along height (vertical axis)
>>> repeat(image, 'h w -> (repeat h) w', repeat=2).shape
(60, 40)

# repeat image 2 time along height and 3 times along width
>>> repeat(image, 'h w -> (h2 h) (w3 w)', h2=2, w3=3).shape
(60, 120)

# convert each pixel to a small square 2x2. Upsample image by 2x
>>> repeat(image, 'h w -> (h h2) (w w2)', h2=2, w2=2).shape
(60, 80)
```

Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929* (2020).

Vision Transformer

Positional Embeddings

- After the image is tokenized, the positional embeddings should be added to each of the tokens.



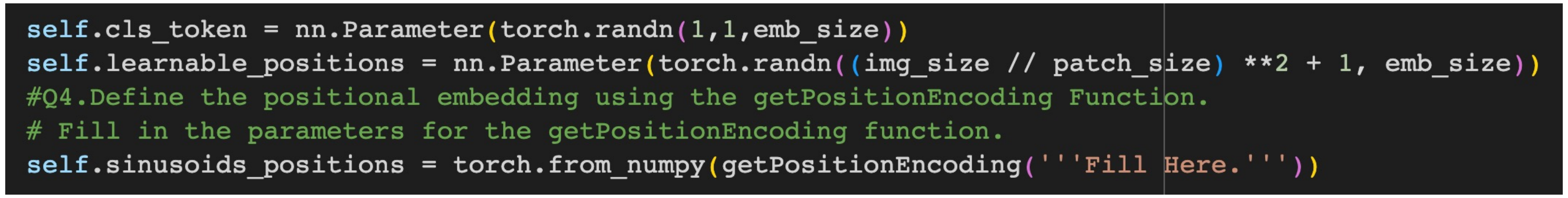
Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929* (2020).

Vision Transformer

Positional Embeddings

Q4. Adding the positional embedding.

Define the positional embedding using the sinusoids by the predefined function getPositionEmbedding(). Fill in the parameters for getPositionEmbedding().



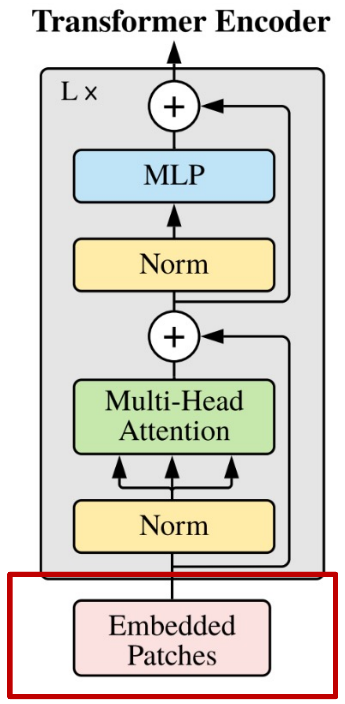
```python
self.cls_token = nn.Parameter(torch.randn(1,1,emb_size))
self.learnable_positions = nn.Parameter(torch.randn((img_size // patch_size) **2 + 1, emb_size))
#Q4.Define the positional embedding using the getPositionEncoding Function.
# Fill in the parameters for the getPositionEncoding function.
self.sinusoids_positions = torch.from_numpy(getPositionEncoding('''Fill Here.'''))
```

Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929* (2020).

Vision Transformer

Transformer Encoder

- The Vision Transformer only utilizes the encoder part of the transformer model.
- We currently implemented the Embedded Patches Part.

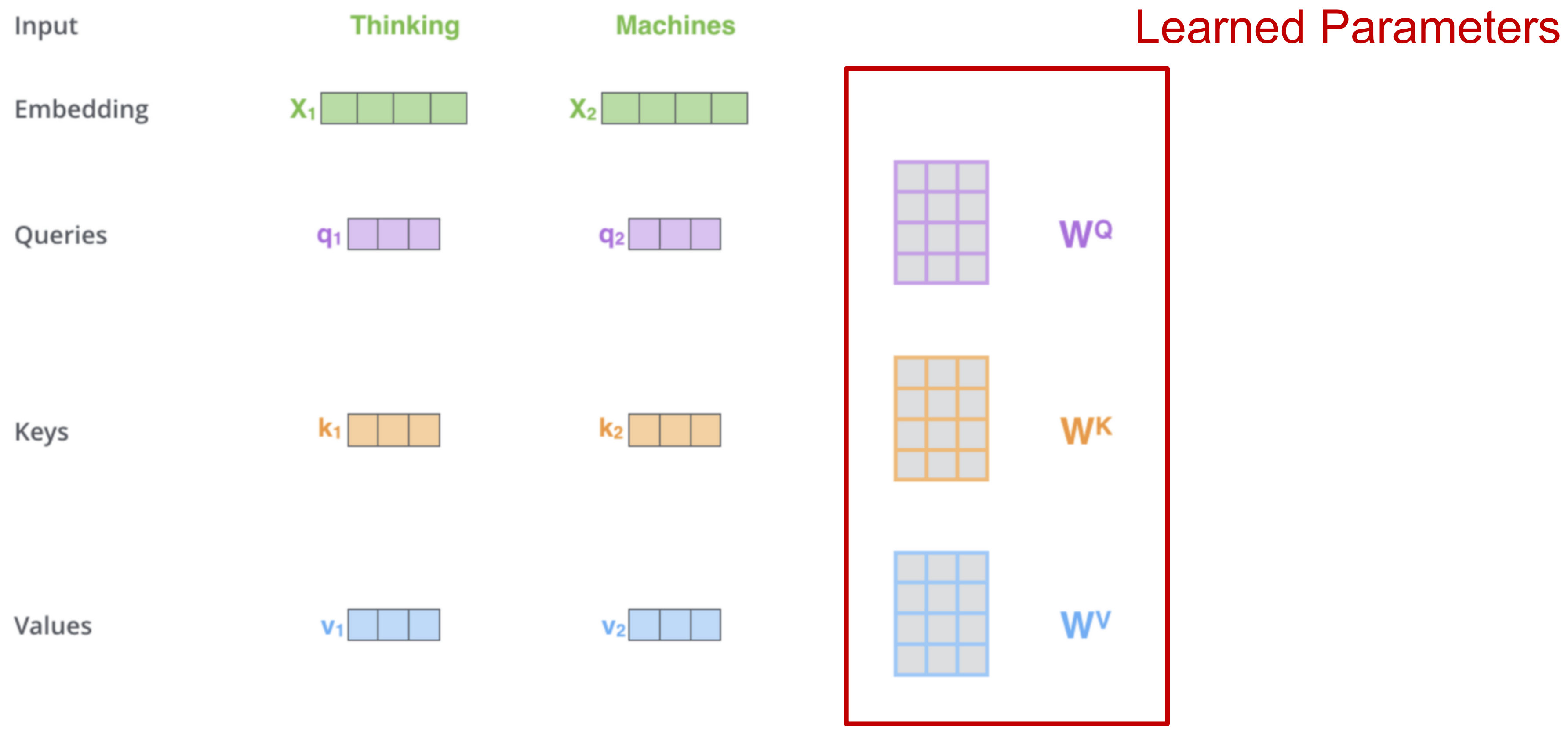


Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929* (2020).

Vision Transformer

Self-Attention

- Query is dot-producted to all key values.

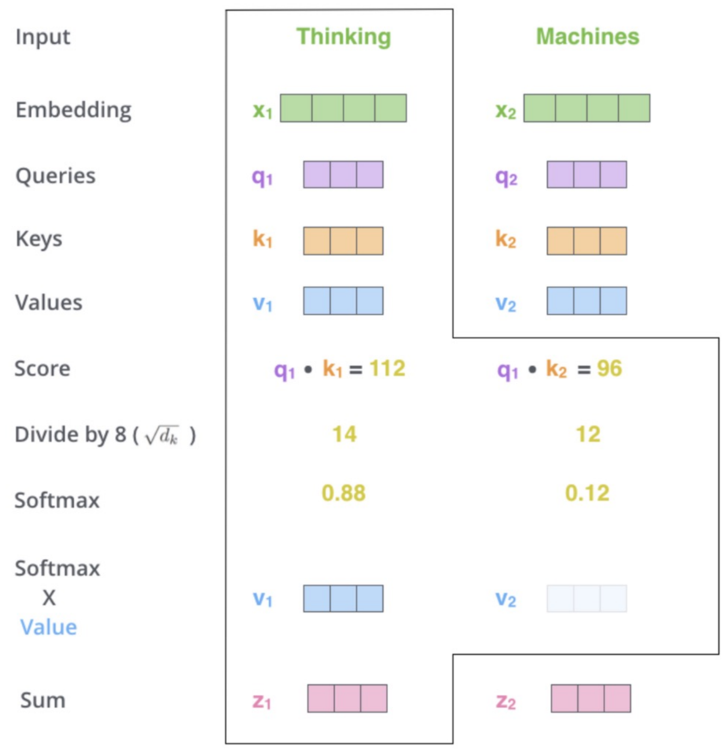


Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929* (2020).
https://ahnjg.tistory.com/57

Vision Transformer

Self-Attention

- The softmaxed scores are multiplied with the values to produce the final vector.

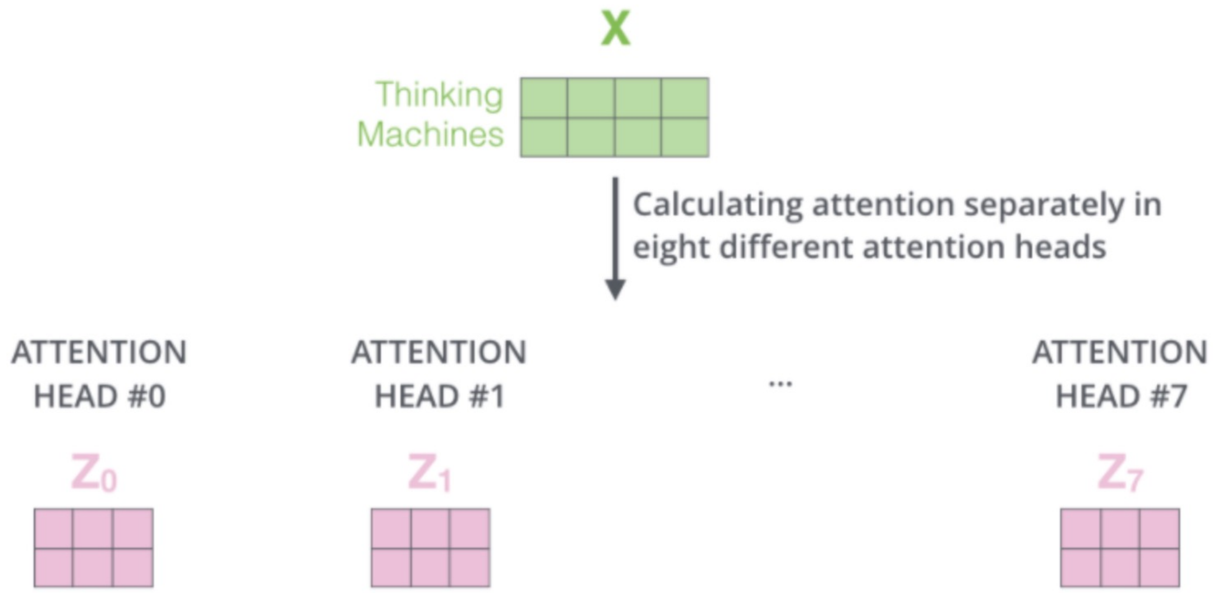


Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929* (2020).
https://ahnjg.tistory.com/57

Vision Transformer

Multi Head Self-Attention

- Use multiple weights to produce query, key and value.
- This can be done by dot-producting the key and values number of multi heads times.

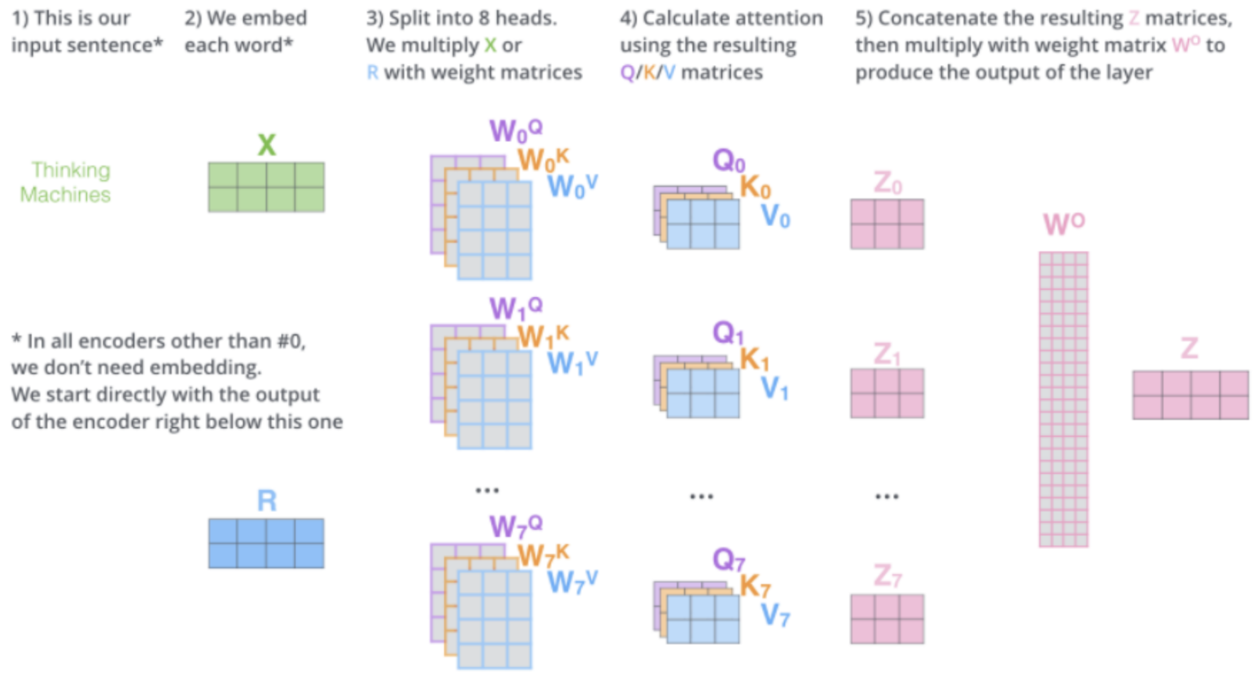


Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929* (2020).
https://ahnjg.tistory.com/57

Vision Transformer

Multi Head Self-Attention

- Use multiple weights to produce query, key and value.
- This can be done by dot-producting the key and values number of multi heads times.


1) This is our input sentence*
2) We embed each word*
3) Split into 8 heads. We multiply X or R with weight matrices
4) Calculate attention using the resulting Q/K/V matrices
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer
Thinking Machines
X
R
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one
W0Q W0K W0V
W1Q W1K W1V
W7Q W7K W7V
Q0 K0 V0
Q1 K1 V1
Q7 K7 V7
Z0 Z1 Z7
WO
Z

Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929* (2020).
https://ahnjg.tistory.com/57

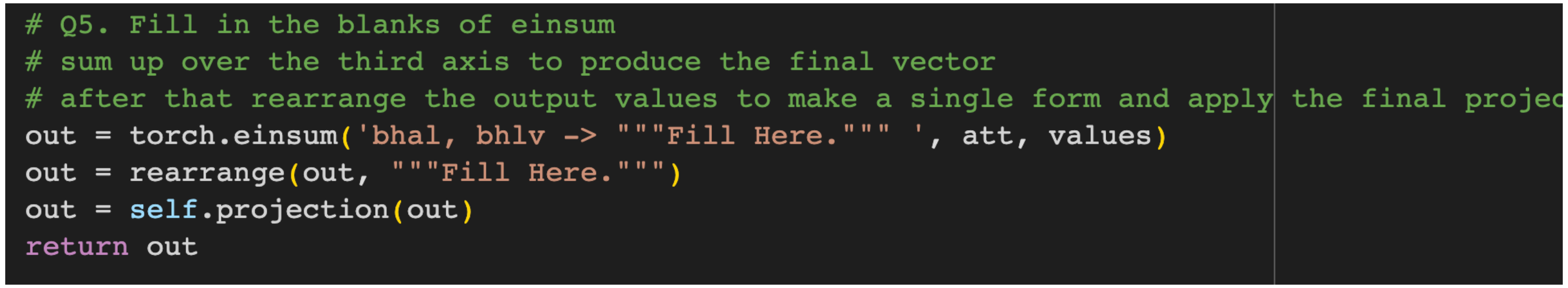
Vision Transformer

Multi Head Self-Attention

Q5. Produce the final vector

Fill in the blanks to produce the final vector which is the result of the softmaxed score multipled to the values. After that fill in the parameters to gather the multi-head values to one.

Hint: Try defining the class and print the dimensions of the intermediate vectors.



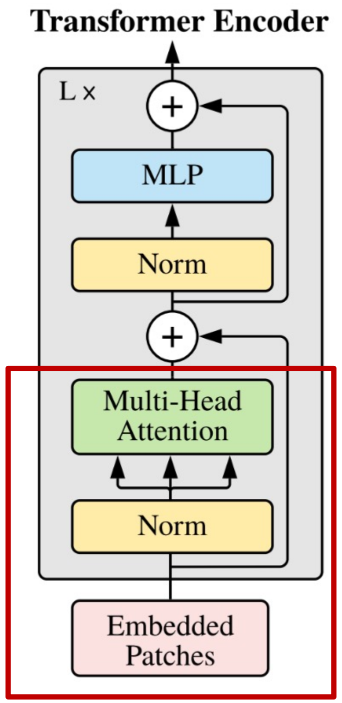
```
# Q5. Fill in the blanks of einsum
# sum up over the third axis to produce the final vector
# after that rearrange the output values to make a single form and apply the final projec
out = torch.einsum('bhal, bhlv -> """Fill Here.""" ', att, values)
out = rearrange(out, """Fill Here.""")
out = self.projection(out)
return out
```

Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929* (2020).
https://ahnjg.tistory.com/57

Vision Transformer

Transformer Encoder

- The Vision Transformer only utilizes the encoder part of the transformer model.
- We currently implemented to the Multi-Head Attention Part.

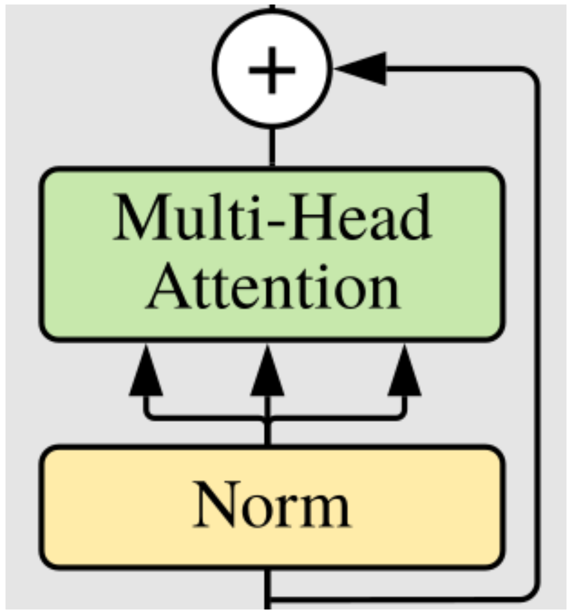


Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929* (2020).

Vision Transformer

Residual Sum

- The Transformer includes a residual sum inside the encoder block.



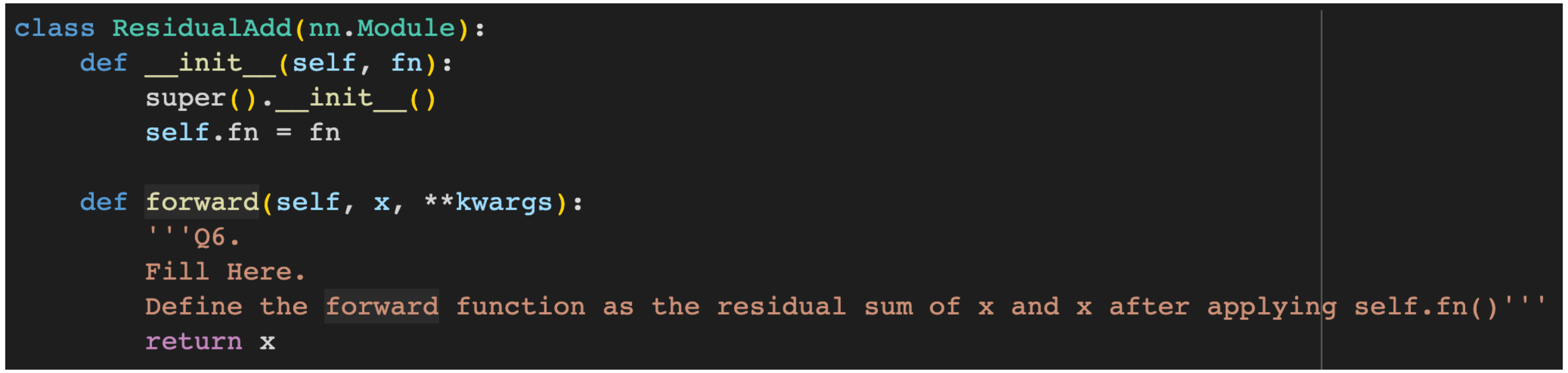
Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929* (2020).

Vision Transformer

Residual Sum

Q6. Define the forward function of the Residual Block

The forward function should return the sum of x and self.fn(x)



```
class ResidualAdd(nn.Module):
    def __init__(self, fn):
        super().__init__()
        self.fn = fn

    def forward(self, x, **kwargs):
        '''Q6.
        Fill Here.
        Define the forward function as the residual sum of x and x after applying self.fn()'''
        return x
```

Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929* (2020).

Vision Transformer

Feed Forward Block

- The transformer encoder block passes the input through a feed forward block with one hidden layer.
- Conventionally, the hidden dimension is 4 times the input dimension.

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

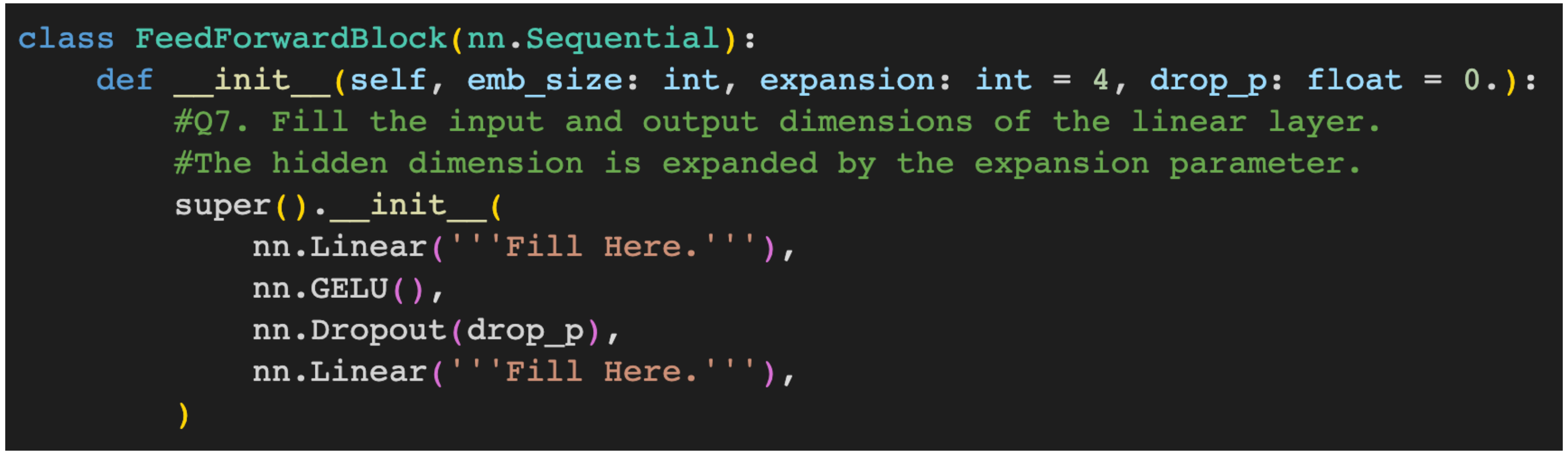
Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929* (2020).

Vision Transformer

Feed Forward Block

Q7. Feed Forward Block

Fill in the input and output dimensions of the linear layer for the feed forward block.



```python
class FeedForwardBlock(nn.Sequential):
    def __init__(self, emb_size: int, expansion: int = 4, drop_p: float = 0.):
        #Q7. Fill the input and output dimensions of the linear layer.
        #The hidden dimension is expanded by the expansion parameter.
        super().__init__(
            nn.Linear('''Fill Here.'''),
            nn.GELU(),
            nn.Dropout(drop_p),
            nn.Linear('''Fill Here.'''),
        )
```

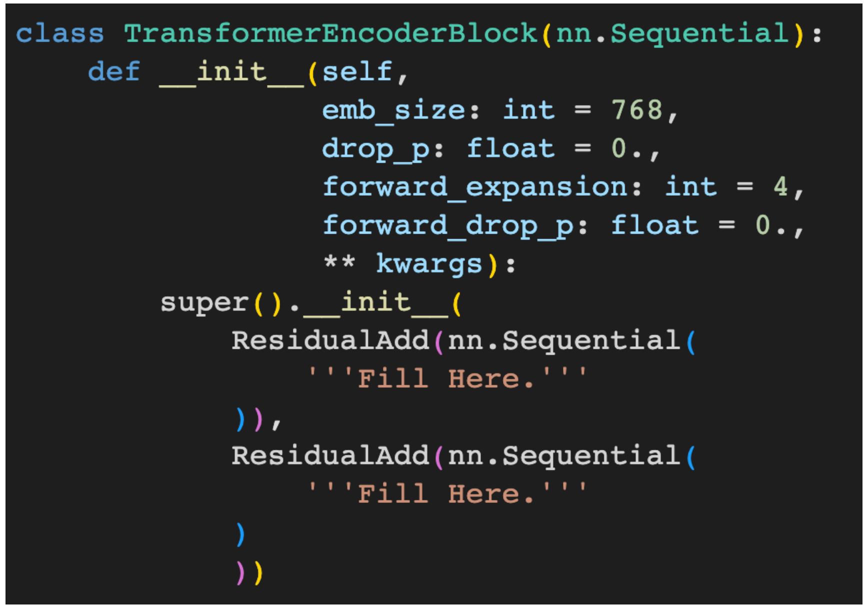
Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929* (2020).

Vision Transformer

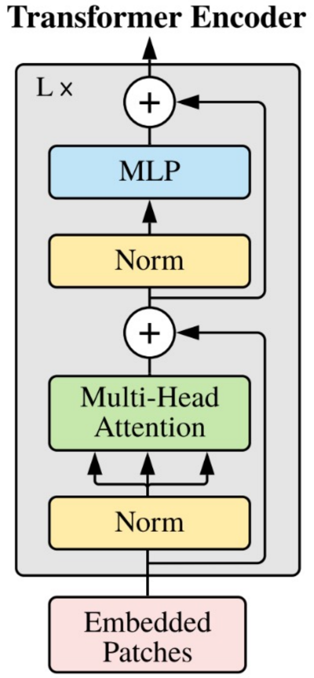
Feed Forward Block

Q7. Transformer Encoder Block

Fill in the blanks in the nn.Sequential() to complete the transformer encoder block with the predefined classes.



```python
class TransformerEncoderBlock(nn.Sequential):
    def __init__(self,
                 emb_size: int = 768,
                 drop_p: float = 0.,
                 forward_expansion: int = 4,
                 forward_drop_p: float = 0.,
                 ** kwargs):
        super().__init__(
            ResidualAdd(nn.Sequential(
                '''Fill Here.'''
            )),
            ResidualAdd(nn.Sequential(
                '''Fill Here.'''
            )
            ))
```



Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929* (2020).

Thank you!

Q & A