HW3 Instructions

Deep Learning

Tasks

- Learning to design an Transformer architecture.
- Learning to implement single (and multi-head) attention mechanism.
- Learning to implement the position-wise feedforward (FFN) Layer.

Grading

- Part 5: Position-wise feedforward (FFN) Layer 20%
- Part 6: Single-head attention 20%
- Part 6: Multi-head attention 20% (Bonus)
- Part 7: Transformer architecture 20%
- Model size 15%:
  - 10%: If your model size is smaller than **1MB**, you will get 10%.
  - 5%: The remaining 5% will depend on your ranking within the class.

Grading

- Model accuracy 15%:
  - 10%: If your accuracy is higher than **57%**, you will get 10%.
  - 5%: The remaining 5% will depend on your ranking within the class.
- Model accuracy on another dataset 10%: it will depend on your ranking within the class.

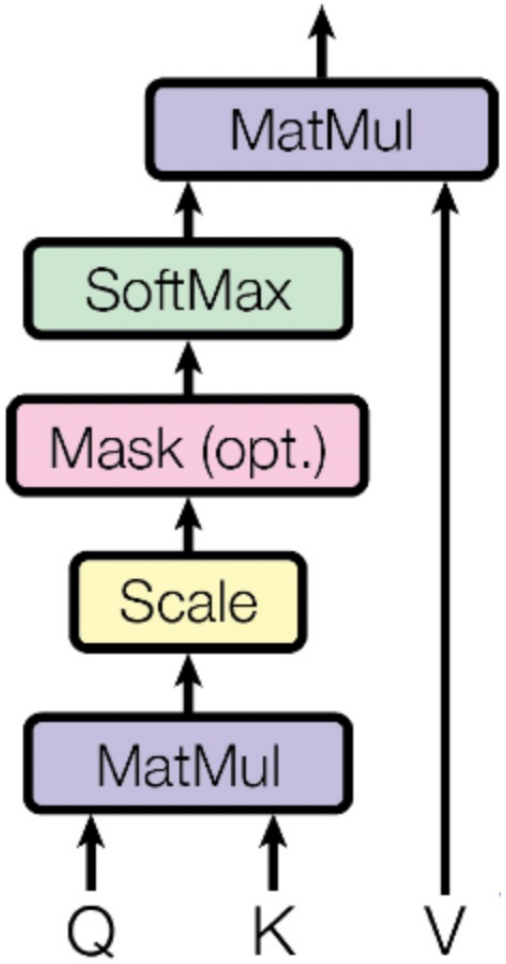
Rules

- You can use all **torch** and **einops** functions for your implementations
- **Except torch.nn.MultiheadAttention()**
- Please still avoid using other modules not mentioned above
- Please note that you can modify hyperparameters in part 9

Position-wise feedforward (FFN) Layer

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2$$

Scaled dot-product attention



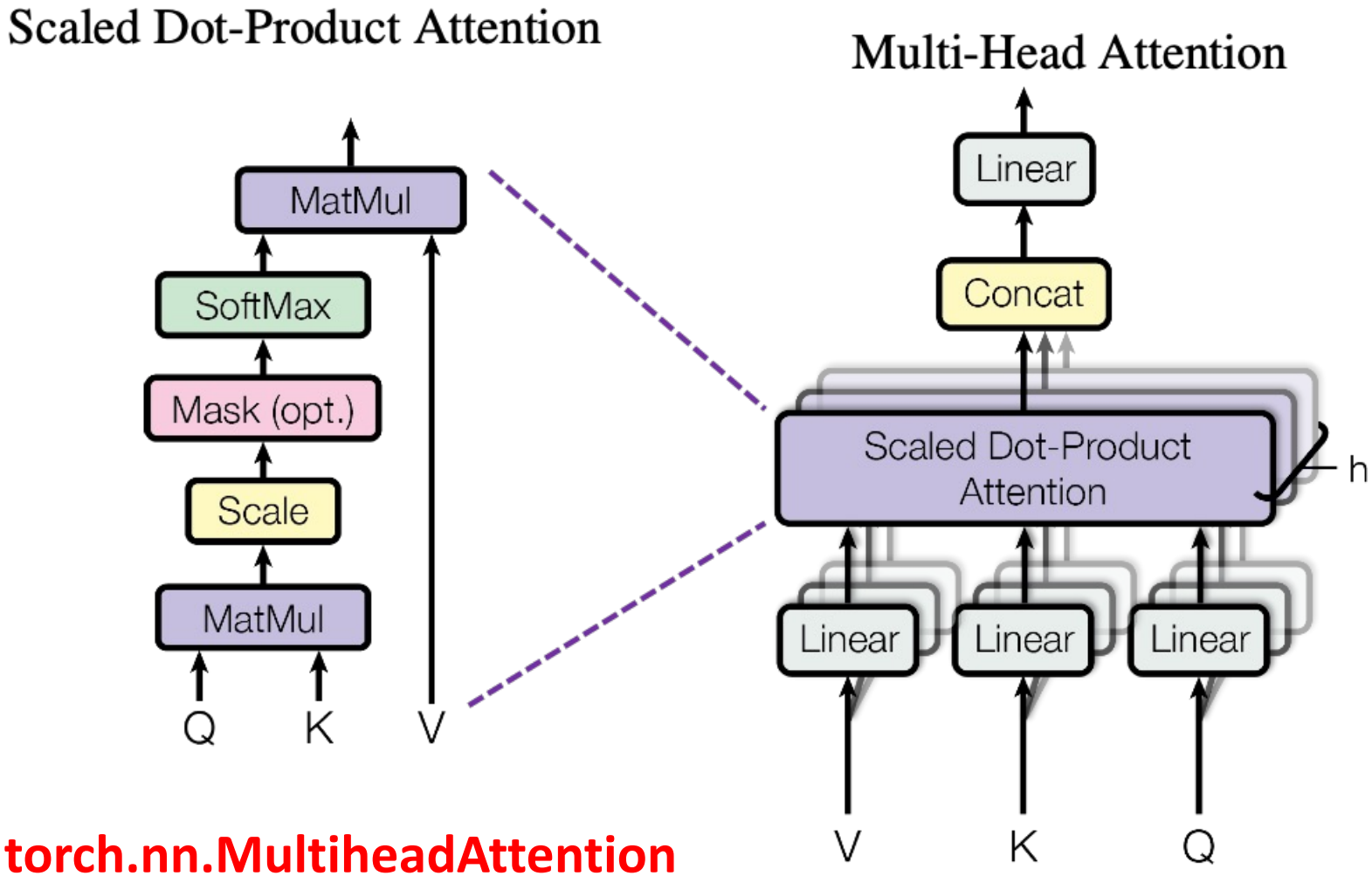
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

$$\begin{aligned} \alpha_{ij} &= \text{softmax}(score(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j) \;\; \forall j \in e) \\ &= \frac{\exp(score(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j)}{\sum_k \exp(score(\mathbf{h}^d_{i-1}, \mathbf{h}^e_k))} \end{aligned}$$

$$\mathbf{c}_i = \sum_j \alpha_{ij}\, \mathbf{h}^e_j$$

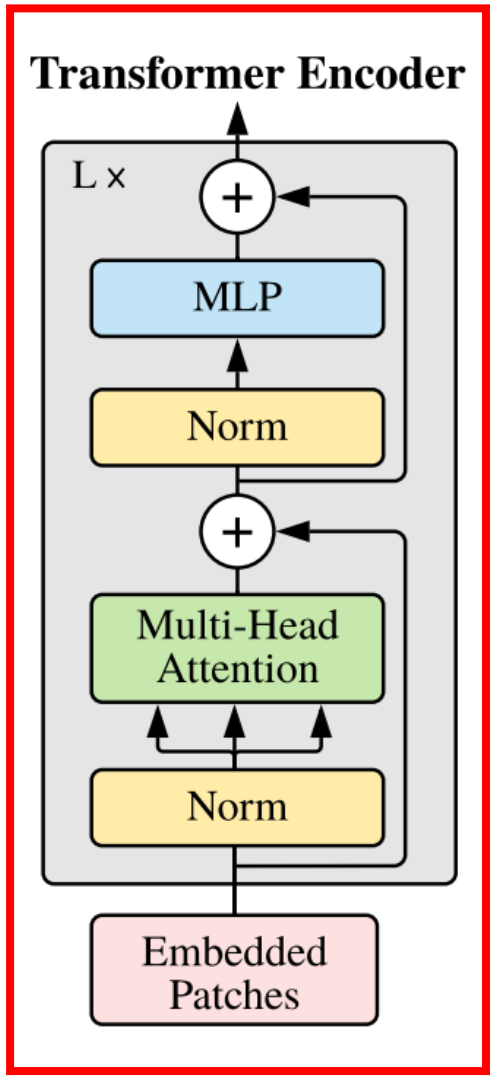
Don’t directly use **torch.nn.MultiheadAttention**

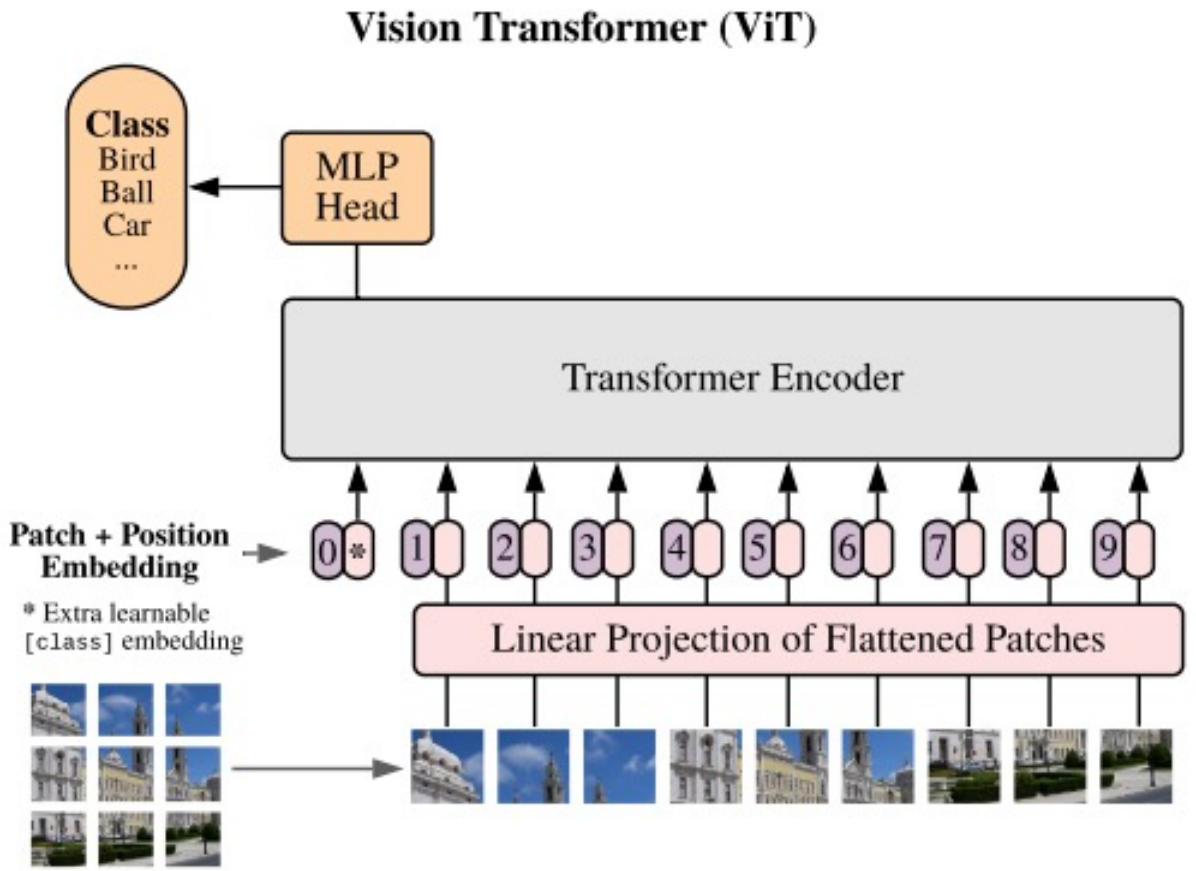
Multi-head attention



Don’t directly use **torch.nn.MultiheadAttention**

Transformer architecture





You can modify hyperparameters in part 9

Einops

- Module for flexible and powerful tensor operations
- Make code more readable and reliable
- Support numpy, pytorch, tensorflow and other modules
- Installation: pip install einops
- Documentation: https://einops.rocks

Einops

- Example:
- Shape: (6, 3, 3)

0	1	2
3	4	5
6	7	8

9	10	11
12	13	14
15	16	17

18	19	20
21	22	23
24	25	26

27	28	29
30	31	32
33	34	35

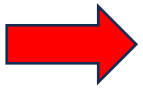
36	37	38
39	40	41
42	43	44

45	46	47
48	49	50
51	52	53

Rearrange

- Transposition, stack
- from einops import rearrange
- rearrange(example[0], '**h w -> w h**')

0	1	2
3	4	5
6	7	8



0	3	6
1	4	7
2	5	8

Rearrange

- Compose batch and height to a new dimension
- Concatenate
- rearrange(example, '**b h w -> (b h) w**')
- b = 6, h = 3, w = 3
- New shape: (18, 3)

0	1	2
3	4	5
6	7	8
9	10	11
12	13	14
15	16	17
18	19	20
21	22	23
24	25	26
27	28	29
30	31	32
33	34	35
36	37	38
39	40	41
42	43	44
45	46	47
48	49	50
51	52	53

Rearrange

- compose batch and width to a new dimension
- rearrange(example, '**b h w -> h (b w)**')
- New shape: (3, 18)

0	1	2	9	10	11	18	19	20	27	28	29	36	37	38	45	46	47
3	4	5	12	13	14	21	22	23	30	31	32	39	40	41	48	49	50
6	7	8	15	16	17	24	25	26	33	34	35	42	43	44	51	52	53

Rearrange

- Different sort
- rearrange(example, ‘b h w -> h **(w b)**’)
- New shape: (3, 18)

0	9	18	27	36	45	1	10	19	28	37	46	2	11	20	29	38	47
3	12	21	30	39	48	4	13	22	31	40	49	5	14	23	32	41	50
6	15	24	33	42	51	7	16	25	34	43	52	8	17	26	35	44	53

Rearrange

- Flatten
- rearrange(example, ‘**b h w -> (b h w)**’)
- New shape: (54,)

Rearrange

- Decomposition of axis
- rearrange(example, ‘(b1 b2) h w, **(b1 h) (b2 w)**’, b1 = 2)
- New Shape: (6, 9)

0	1	2	9	10	11	18	19	20
3	4	5	12	13	14	21	22	23
6	7	8	15	16	17	24	25	26
27	28	29	36	37	38	45	46	47
30	31	32	39	40	41	48	49	50
33	34	35	42	43	44	51	52	53

Rearrange

- Decomposition of axis
- rearrange(example, **‘(b1 b2) h w, (b2 h) (b1 w)’**, **b1 = 2**)
- New Shape: (9, 6)

0	1	2	27	28	29
3	4	5	30	31	32
6	7	8	33	34	35
9	10	11	36	37	38
12	13	14	39	40	41
15	16	17	42	43	44
18	19	20	45	46	47
21	22	23	48	49	50
24	25	26	51	52	53

Rearrange

- Move part of width dimension to height
- rearrange(example, **‘b h (w w2) -> (h w2) (b w)’, w2 = 3**)
- New shape: (9, 6)

0	9	18	27	36	45
1	10	19	28	37	46
2	11	20	29	38	47
3	12	21	30	39	48
4	13	22	31	40	49
5	14	23	32	41	50
6	15	24	33	42	51
7	16	25	34	43	52
8	17	26	35	44	53

Rearrange

- Move part of height dimension to width
- rearrange(example, **‘b (h h2) w -> (b h) (h2 w)’, h2 = 3**)
- New shape: (6, 9)

0	1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16	17
18	19	20	21	22	23	24	25	26
27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44
45	46	47	48	49	50	51	52	53

Rearrange

- Reorder
- rearrange(example, **‘(b1 b2) h w -> h (b2 b1 w)’, b1 = 2**)
- New Shape: (3, 18)

0	1	2	27	28	29	9	10	11	36	37	38	18	19	20	45	46	47
3	4	5	30	31	32	12	13	14	39	40	41	21	22	23	48	49	50
6	7	8	33	34	35	15	16	17	42	43	44	24	25	26	51	52	53

Rearrange

- Expand dimensions
- rearrange(torch.arrange(3), 'd -> d **1**’)
- tensor([0, 1, 2]) -> tensor([[0], [1], [2]])

Rearrange

- Squeeze
- rearrange(last_tensor, ‘d **1** -> d’)
- tensor([[0], [1], [2]]) -> tensor([0, 1, 2])

Rearrange

from einops.layers.torch import Rearrange

```
class myTokenization(nn.Module):

    def __init__(self, output_dim, patch_size, channels):

        super().__init__()

        patch_dim = patch_size * patch_size * channels

        self.to_patch_tokens = nn.Sequential(
            Rearrange('b c (h p1) (w p2) -> b (h w) (p1 p2 c)', p1 = patch_size, p2 = patch_size),
            nn.LayerNorm(patch_dim),
            nn.Linear(patch_dim, output_dim)
        )

    def forward(self, x):

        return self.to_patch_tokens(x)
```

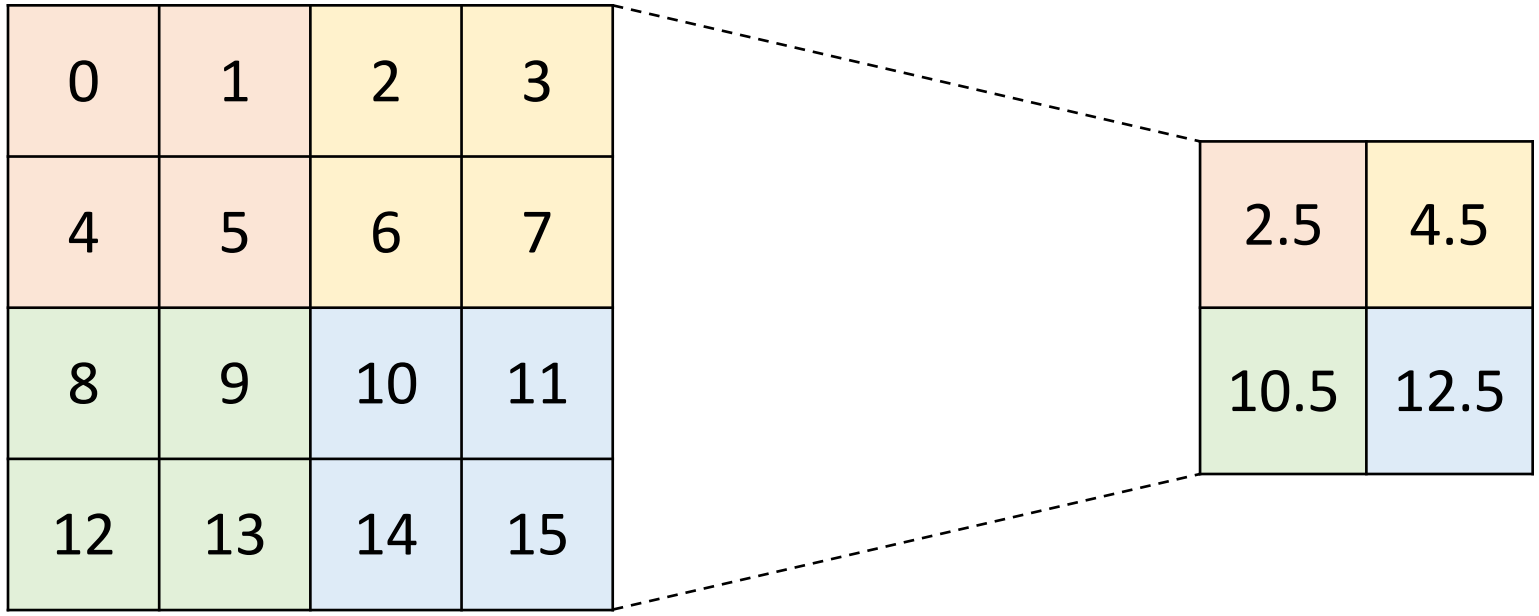
Reduce

- from einops import reduce
- mean, min, max , sum, prod
- Average over batch
- reduce(example.**float**(), **'b h w -> h w', 'mean'**)
- New shape: (3, 3)

22.5	23.5	24.5
25.5	26.5	27.5
28.5	29.5	30.5

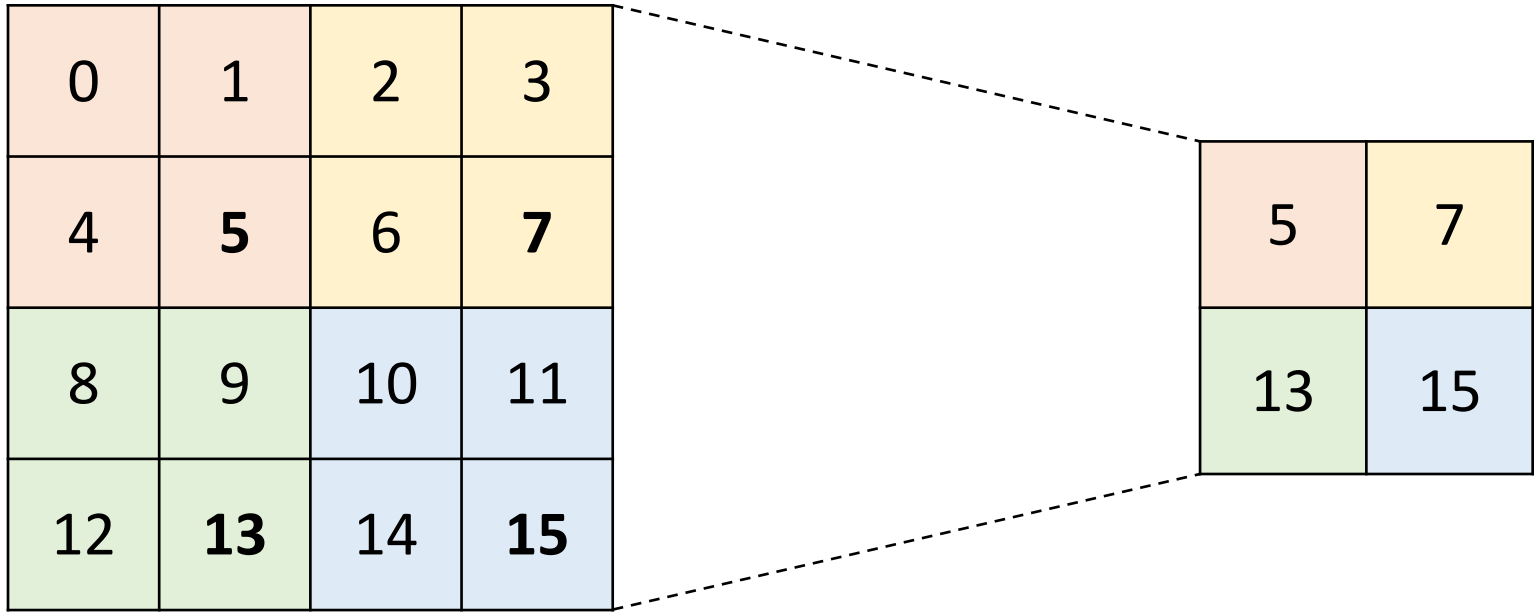
Reduce

- Average pooling
- reduce(example2.float(), **'(h h2) (w w2) -> h w', 'mean', h2 = 2, w2 = 2**)
- kernel size = 2 * 2, stride = 2



Reduce

- Max pooling
- reduce(example2, **'(h h2) (w w2) -> h w', ‘max', h2 = 2, w2 = 2**)
- kernel size = 2 * 2, stride = 2



Reduce

- Compute max in each image individually
- reduce(example, 'b h w -> b () ()', 'max’)
- New shape: (6, 1, 1)

Reduce

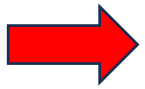
- Sum over batch
- reduce(example, **'b h w -> h w', ‘sum’**)
- New shape: (3, 3)

135	141	147
153	159	165
171	177	183

Repeat

- from einops import repeat
- repeat along existing axis
- repeat(example[0], 'h w -> h (**repeat** w)', repeat = 3)

0	1	2
3	4	5
6	7	8

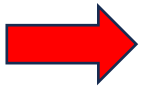


0	1	2	0	1	2	0	1	2
3	4	5	3	4	5	3	4	5
6	7	8	6	7	8	6	7	8

Repeat

- Repeat along more existing axes
- repeat(example[0], 'h w -> (2 h) (2 w)')

0	1	2
3	4	5
6	7	8

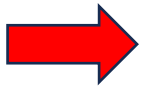


0	1	2	0	1	2
3	4	5	3	4	5
6	7	8	6	7	8
0	1	2	0	1	2
3	4	5	3	4	5
6	7	8	6	7	8

Repeat

- Repeat each element
- repeat(example[0], 'h w -> h (**w 2**)')

0	1	2
3	4	5
6	7	8



0	0	1	1	2	2
3	3	4	4	5	5
6	6	7	7	8	8

Submission

- Upload your zip file to NTU Cool
- **StudentID_HW3.zip**
  - DL_HW3_StudentID.ipynb
  - StudentID_submission.pt
  - StudentID_submission.csv
- **Deadline: 5/20 23:59**