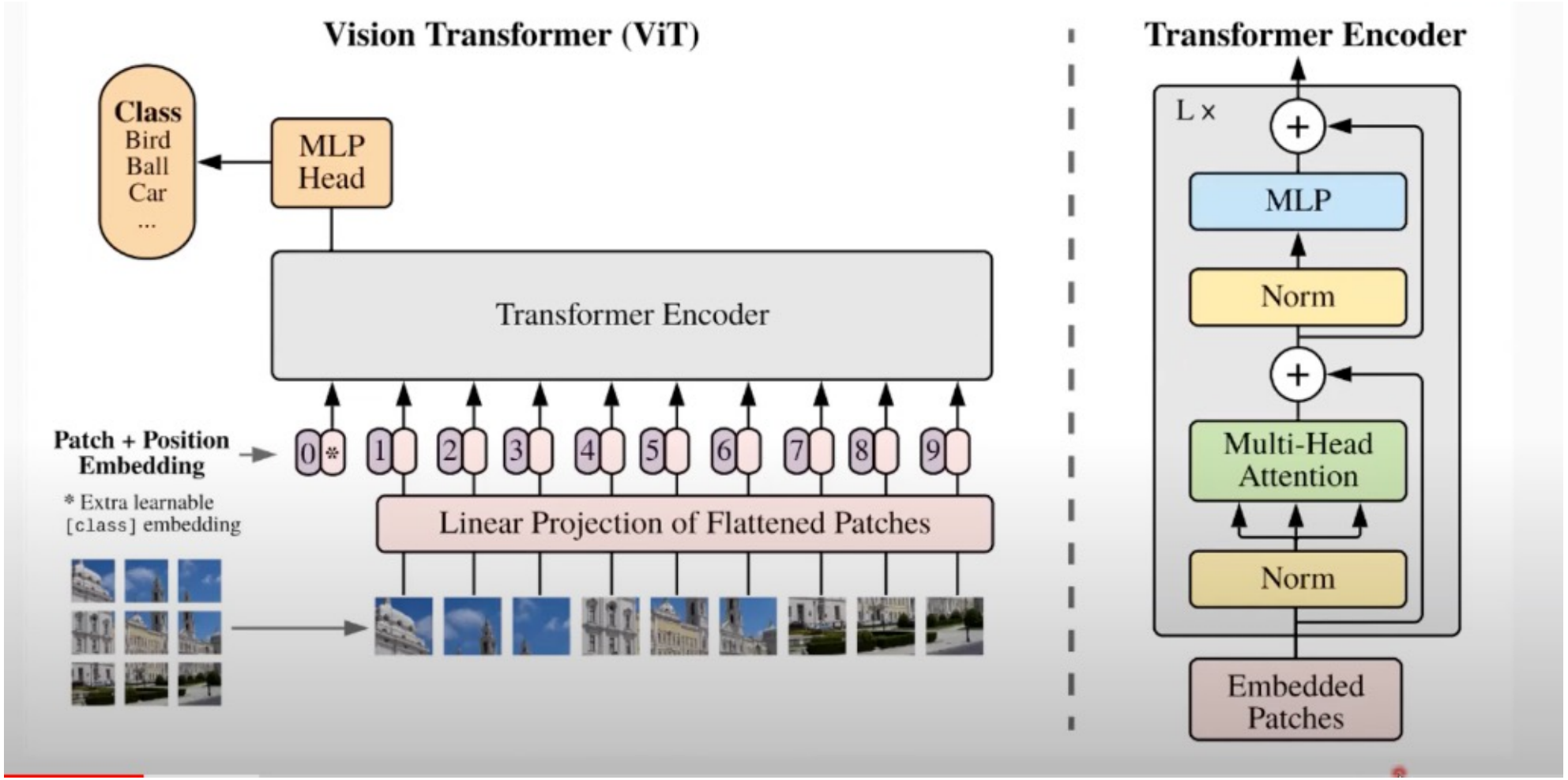
DEEP LEARNING FOR COMPUTER VISION

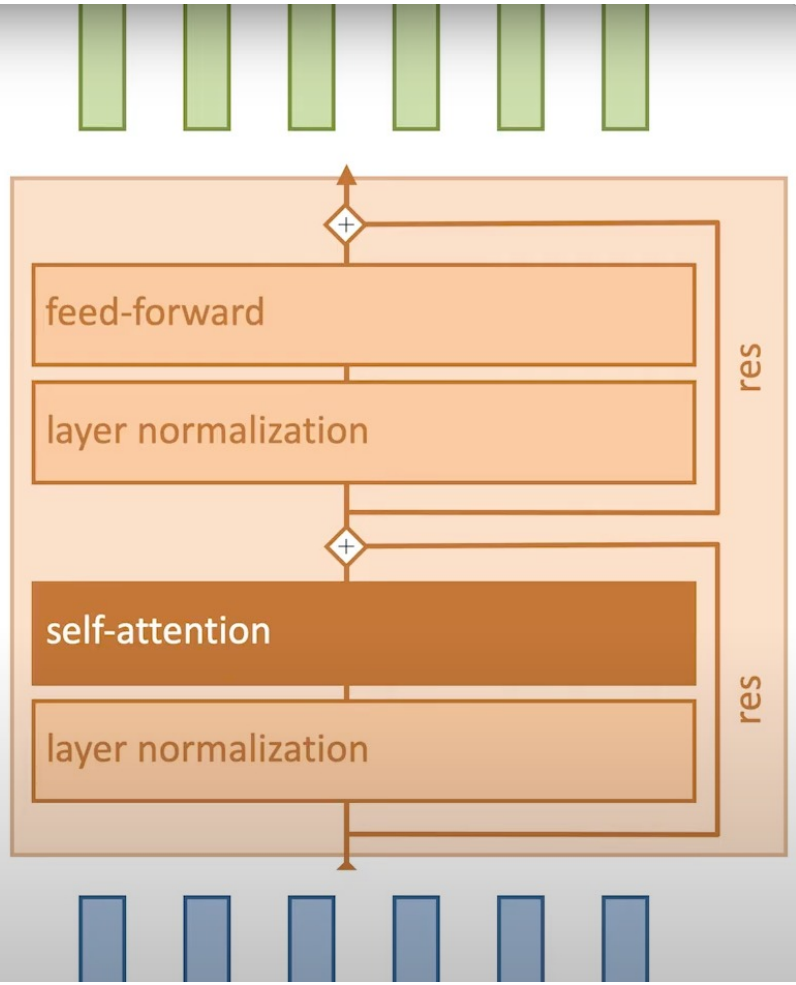
Week11

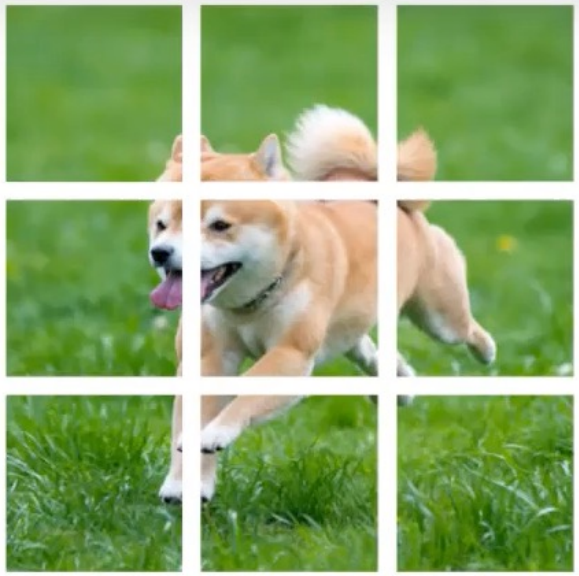
Dr. Tuchsanai. PloySuwan


Vision Transformer (ViT)
Class
Bird
Ball
Car
...
MLP Head
Transformer Encoder
Patch + Position Embedding
* Extra learnable [class] embedding
0 * 1 2 3 4 5 6 7 8 9
Linear Projection of Flattened Patches
Transformer Encoder
L x
MLP
Norm
Multi-Head Attention
Norm
Embedded Patches

TRANSFORMER BLOCK

```
class Block(nn.Module):

    def forward(self, x):

        y = self.layernorm(y)

        y = self.attention(x)

        x = x + y


        y = self.layernorm(x)

        y = self.linear(x)

        return x + y
```

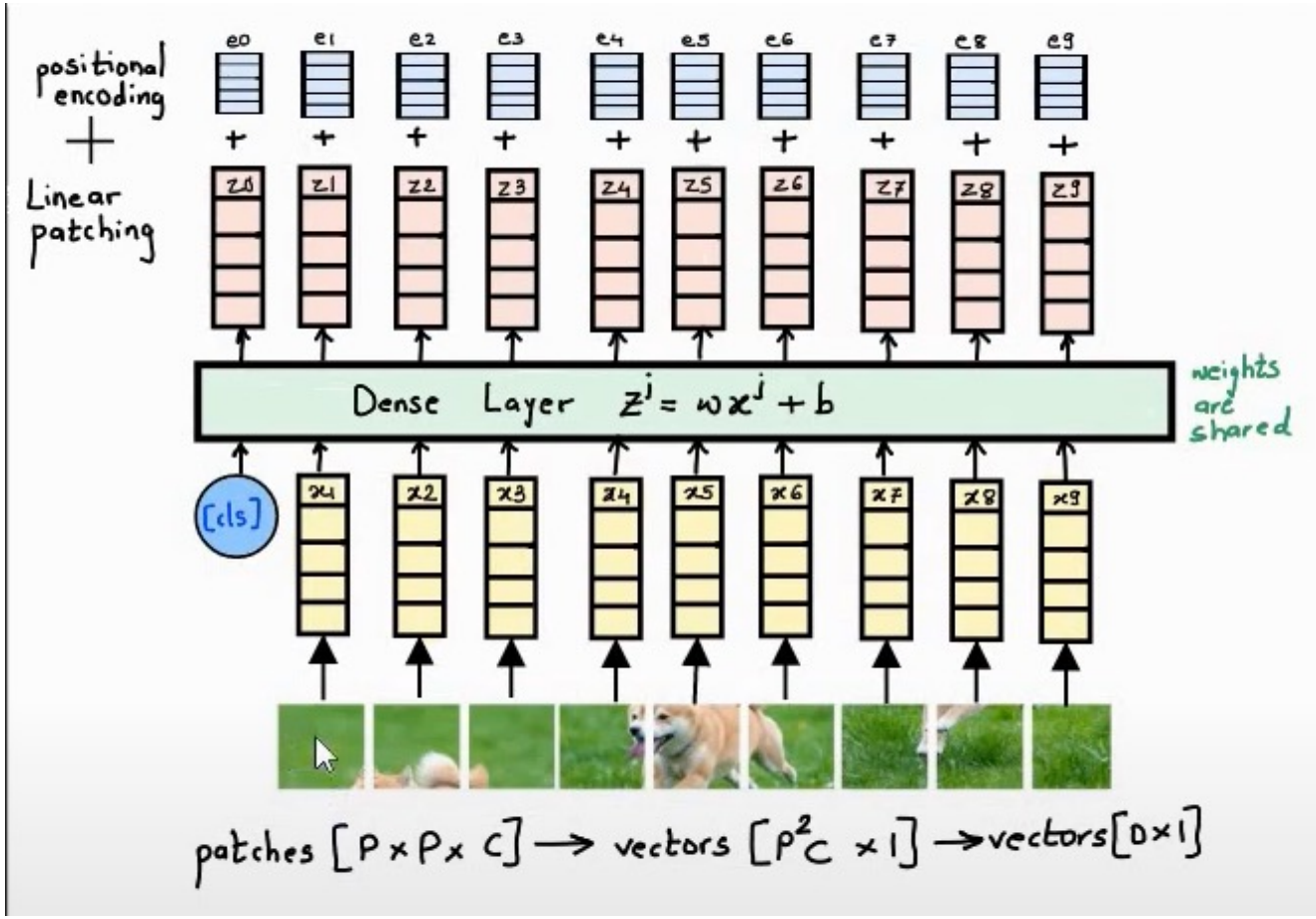




Positional encodings

Linear patching

Share weights

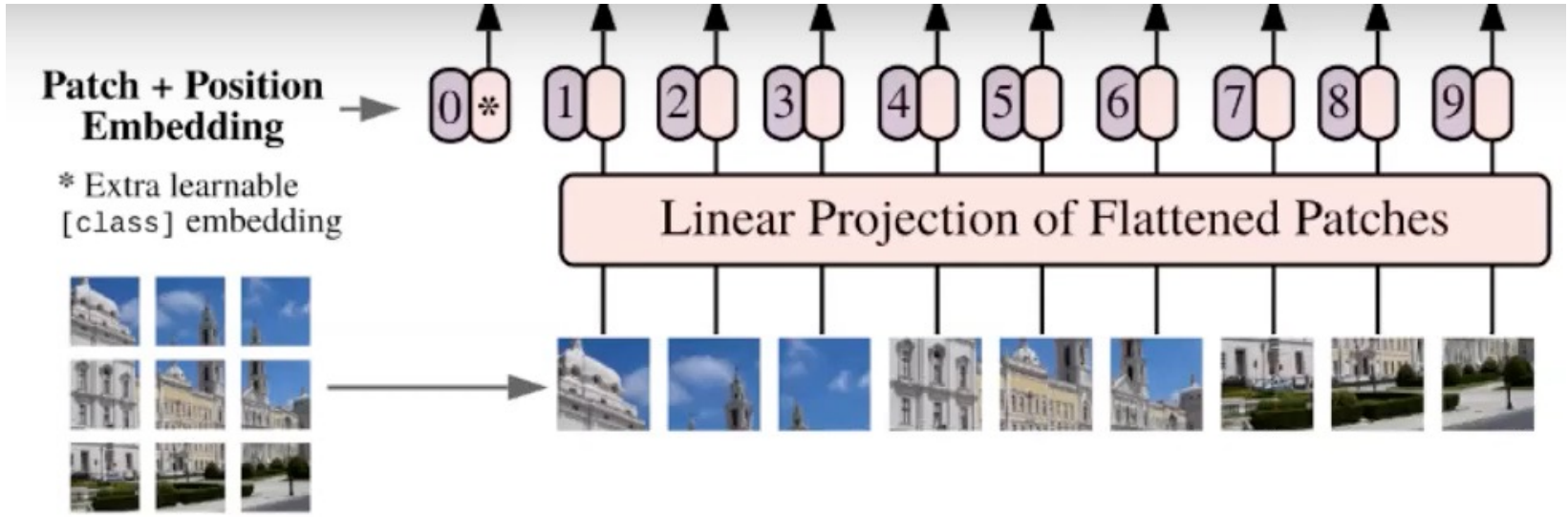


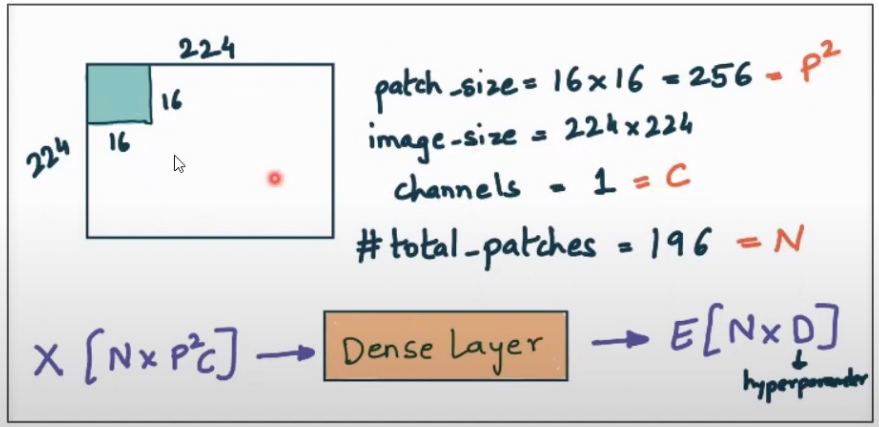
- The linear representation of the patches is learned by the network. The patches do not overlap in the paper but you can overlap them if you want to.
- There is no activation function applied to the Dense Layer that is the activation function is linear.

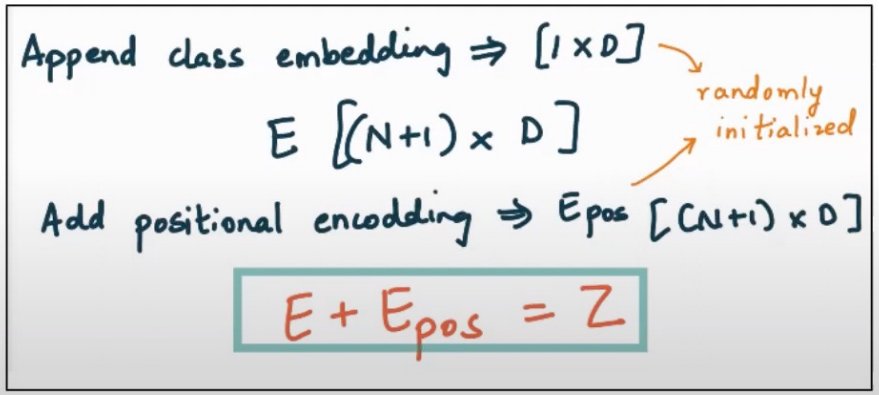
atch_size: 16x16
mbed_dim : D

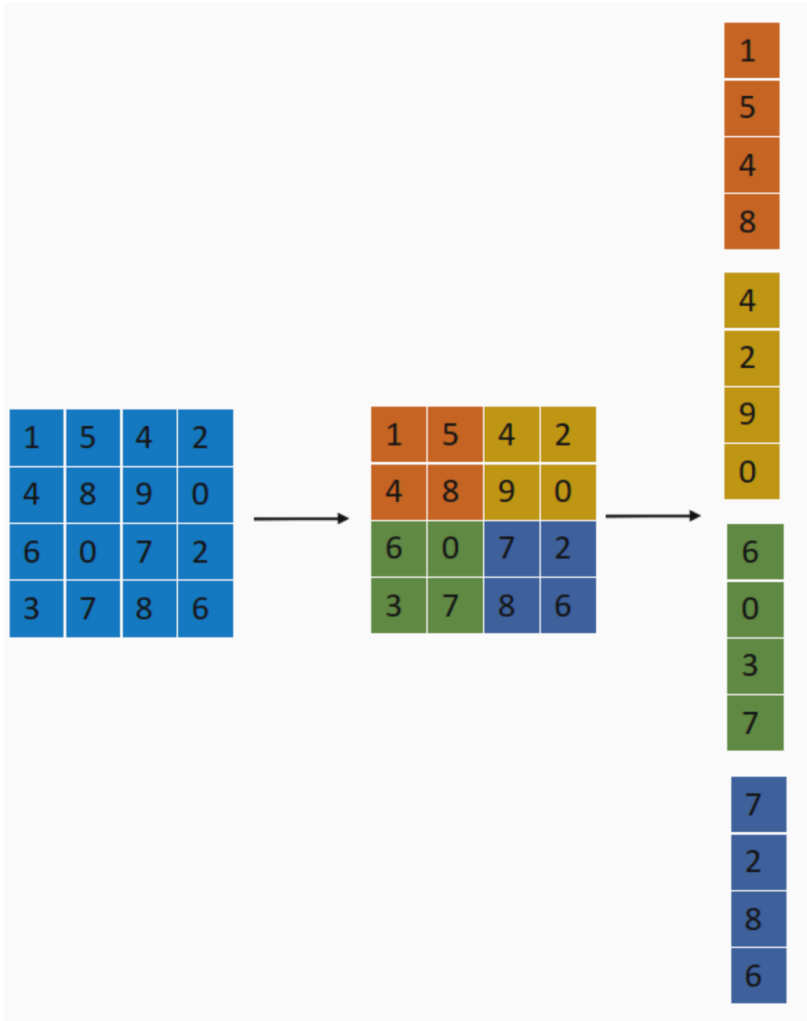
patches [P * P * C] ⇒ vectors [P^2C * 1] ⇒ vectors[D * 1]

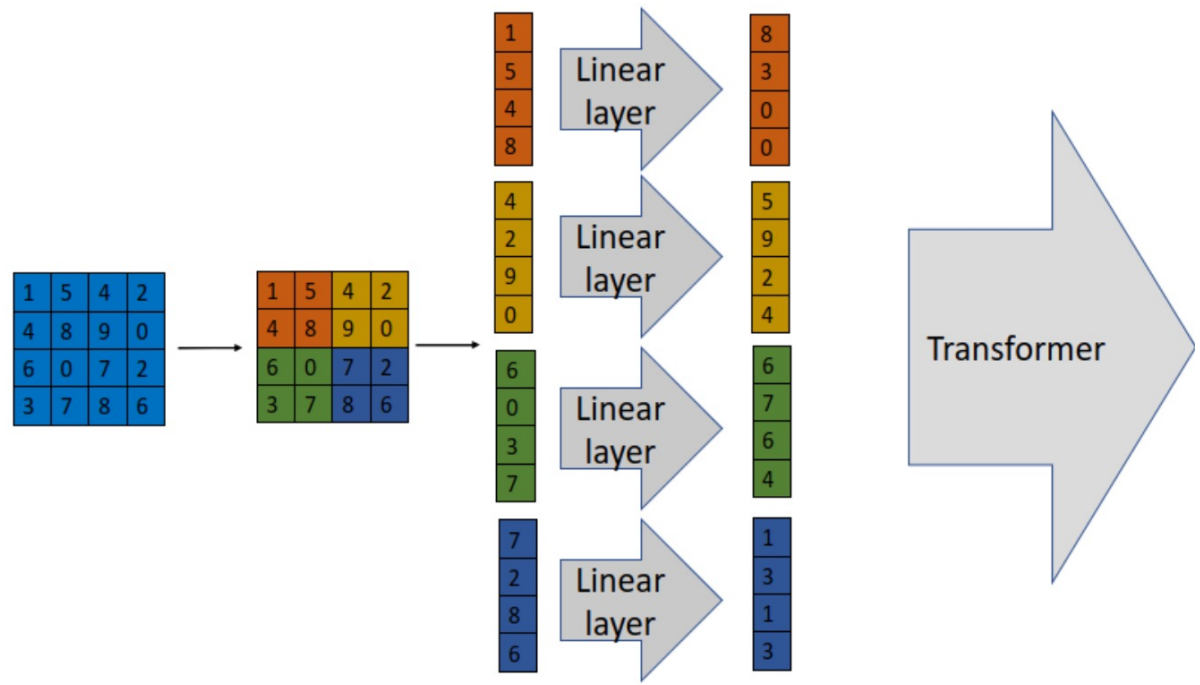

1
2
3
4
5
6
7
8
9

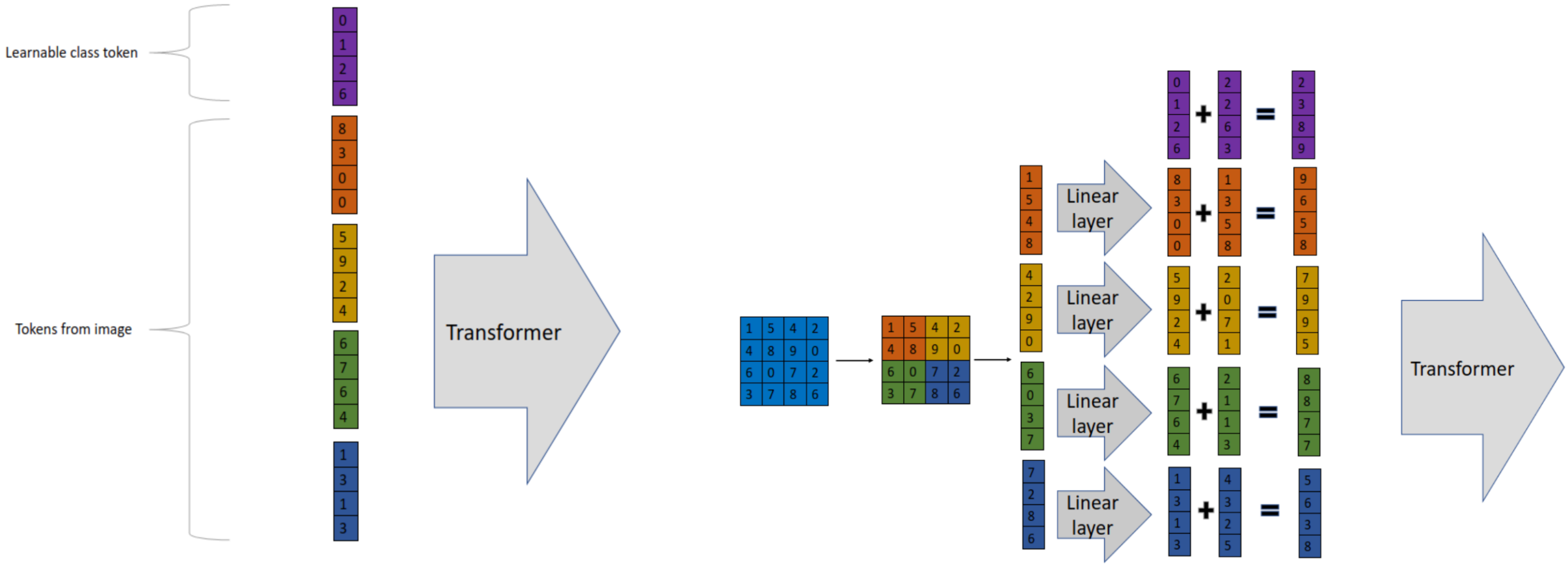

Patch + Position Embedding
* Extra learnable [class] embedding
0 * 1 2 3 4 5 6 7 8 9
Linear Projection of Flattened Patches

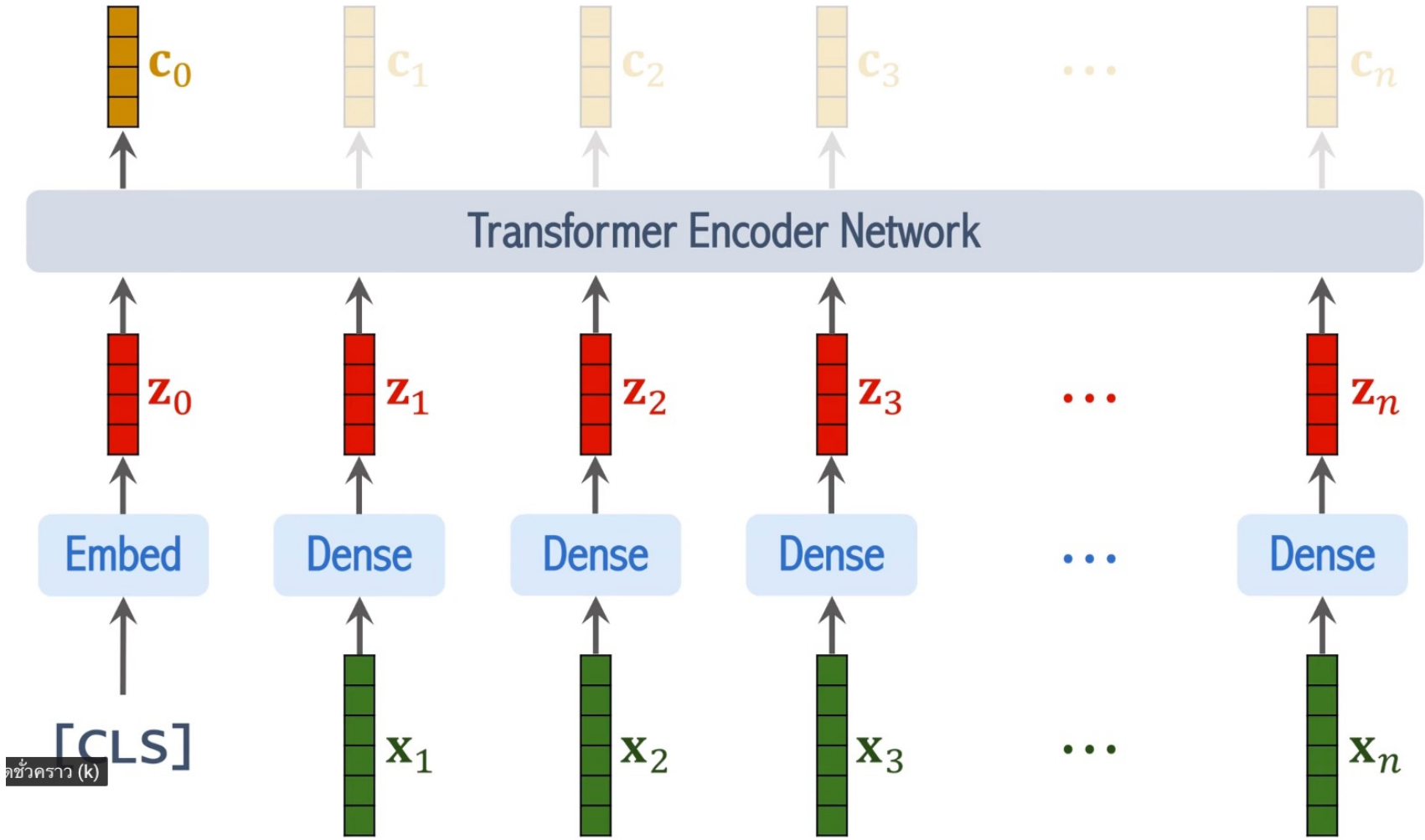

224
224
16
16
patch_size = 16 x 16 = 256 = p^2
image_size = 224 x 224
channels = 1 = C
# total_patches = 196 = N
X [N x p^2C]
Dense Layer
E[N x D]
hyperparameter


Append class embedding ⇒ [1 x D]
randomly initialized
E [(N+1) x D]
Add positional encodding ⇒ Epos [(N+1) x D]
E + Epos = Z

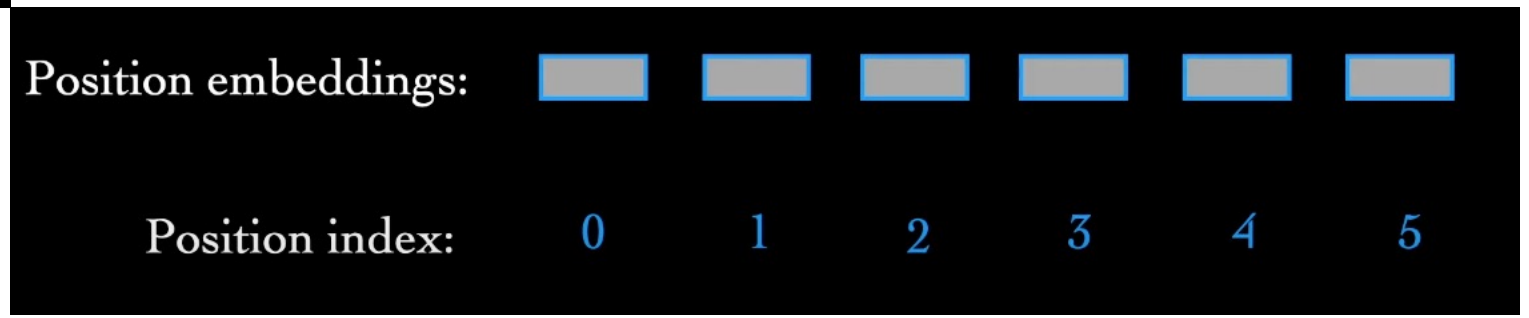

1 5 4 2
4 8 9 0
6 0 7 2
3 7 8 6
1 5 4 2
4 8 9 0
6 0 7 2
3 7 8 6
1 5 4 8
4 2 9 0
6 0 3 7
7 2 8 6


1 5 4 2
4 8 9 0
6 0 7 2
3 7 8 6
1 5 4 2
4 8 9 0
6 0 7 2
3 7 8 6
1 5 4 8
4 2 9 0
6 0 3 7
7 2 8 6
Linear layer
Linear layer
Linear layer
Linear layer
8 3 0 0
5 9 2 4
6 7 6 4
1 3 1 3
Transformer


Learnable class token
Tokens from image
Transformer
Linear layer
Linear layer
Linear layer
Linear layer
Transformer

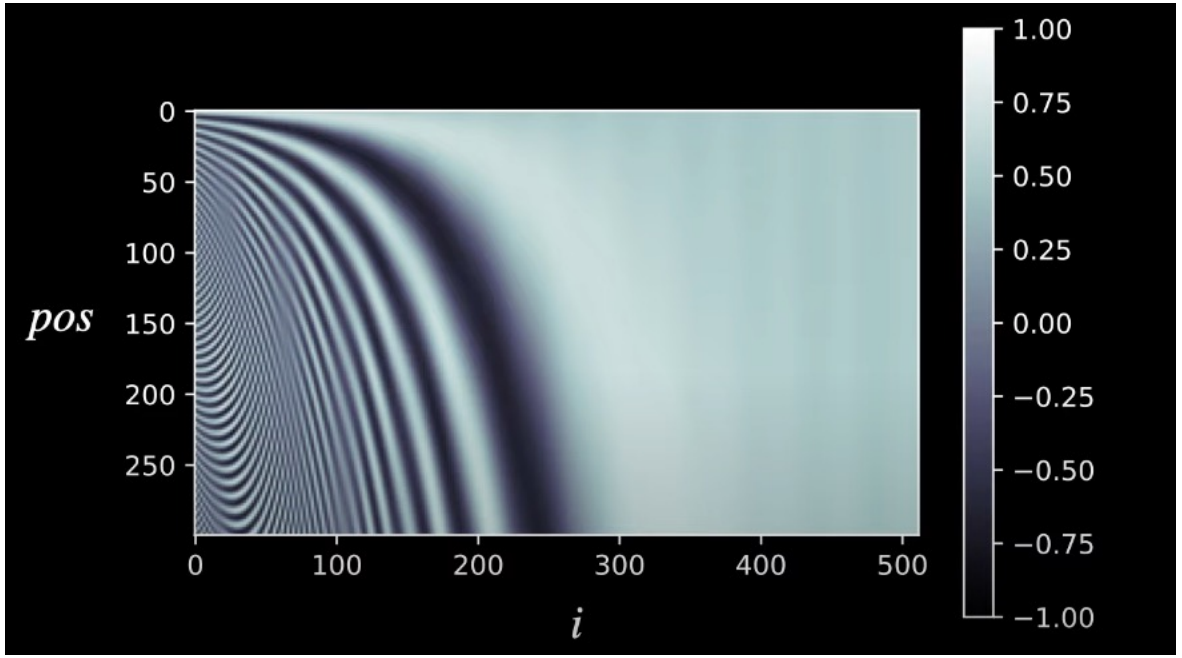

c_0
c_1
c_2
c_3
c_n
Transformer Encoder Network
z_0
z_1
z_2
z_3
z_n
Embed
Dense
Dense
Dense
Dense
[CLS]
x_1
x_2
x_3
x_n

Positional encoding



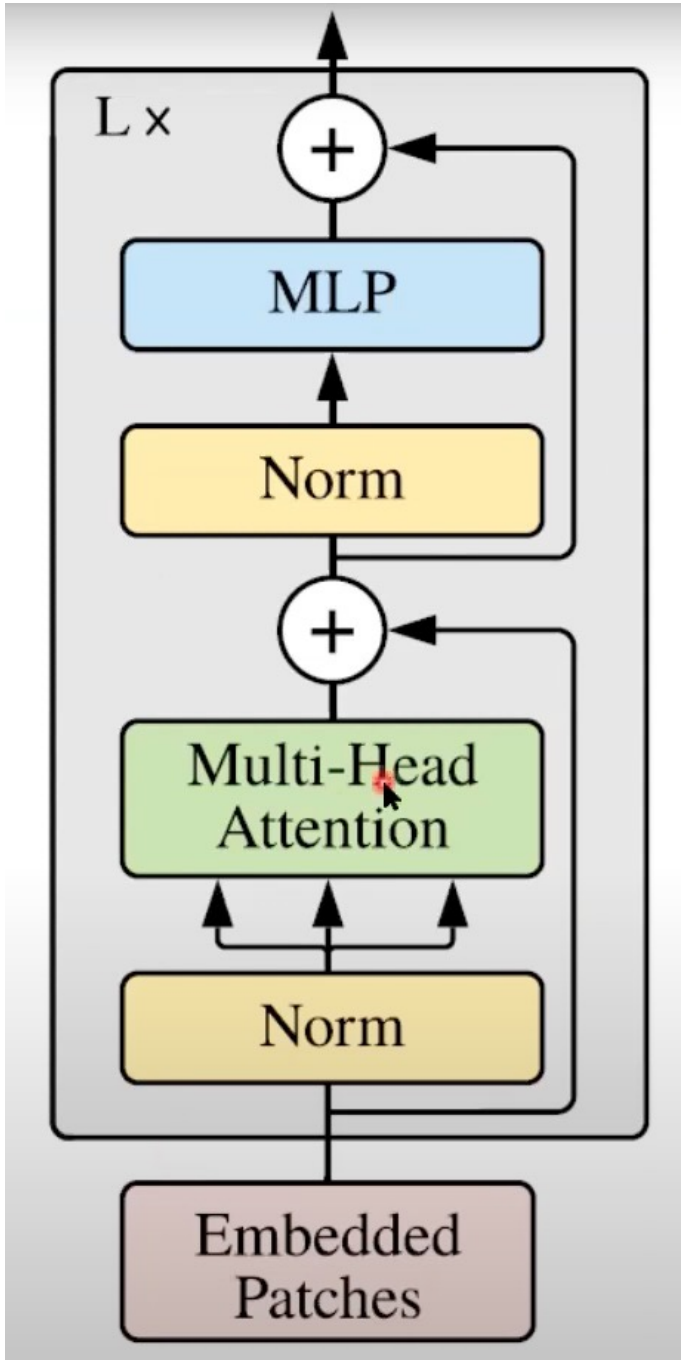
$$PE_{(pos,2i)} = sin(pos/10000^{2i/D})$$

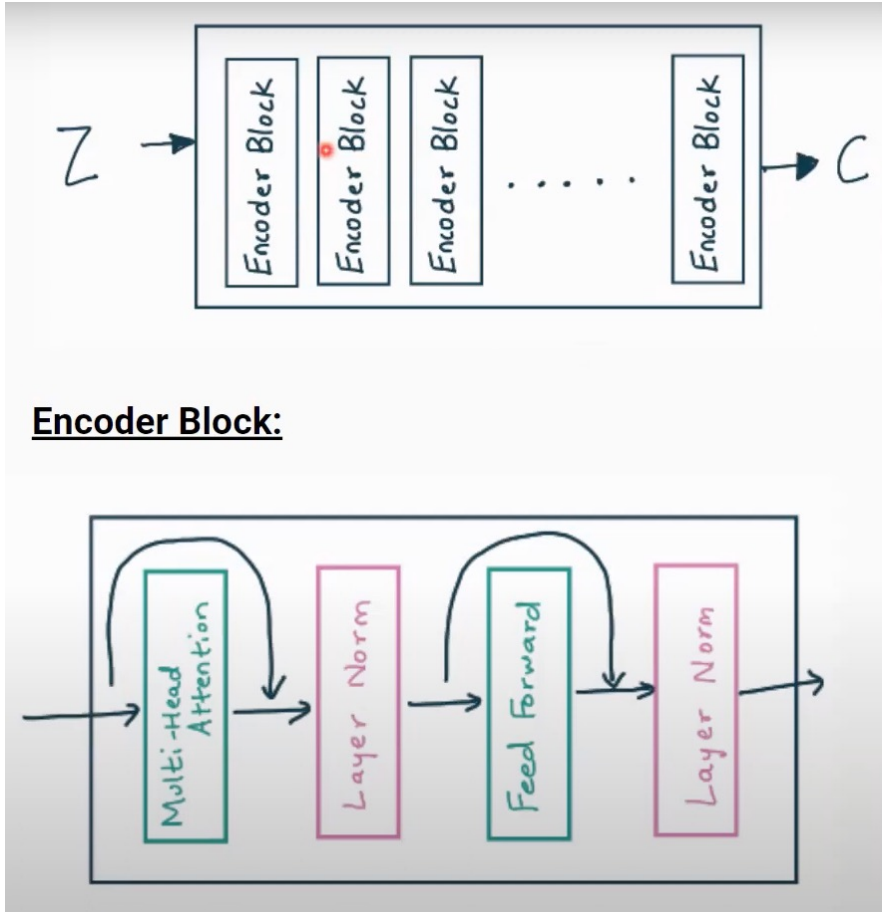
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/D})$$



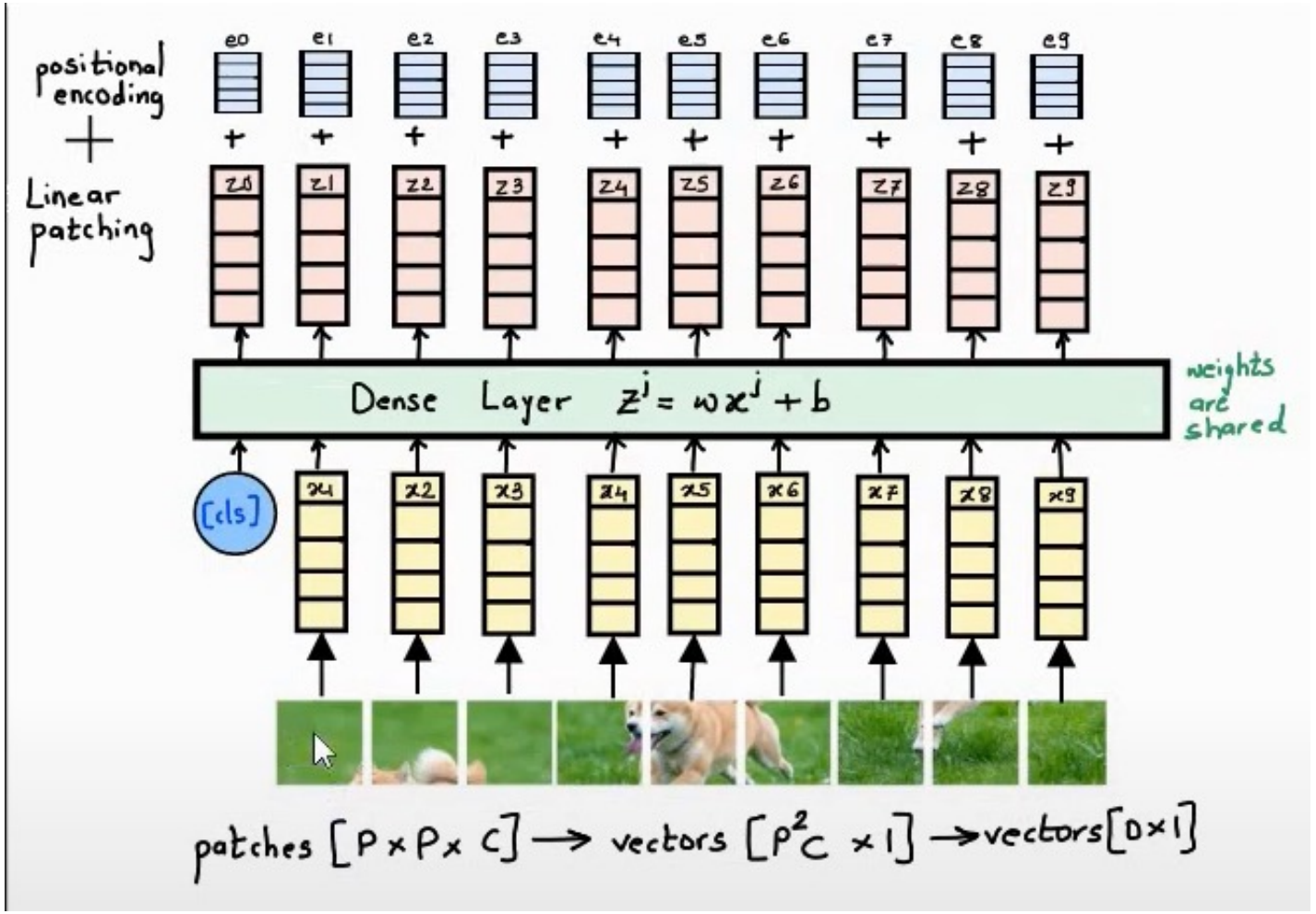
Step 2: Transformer Encoder

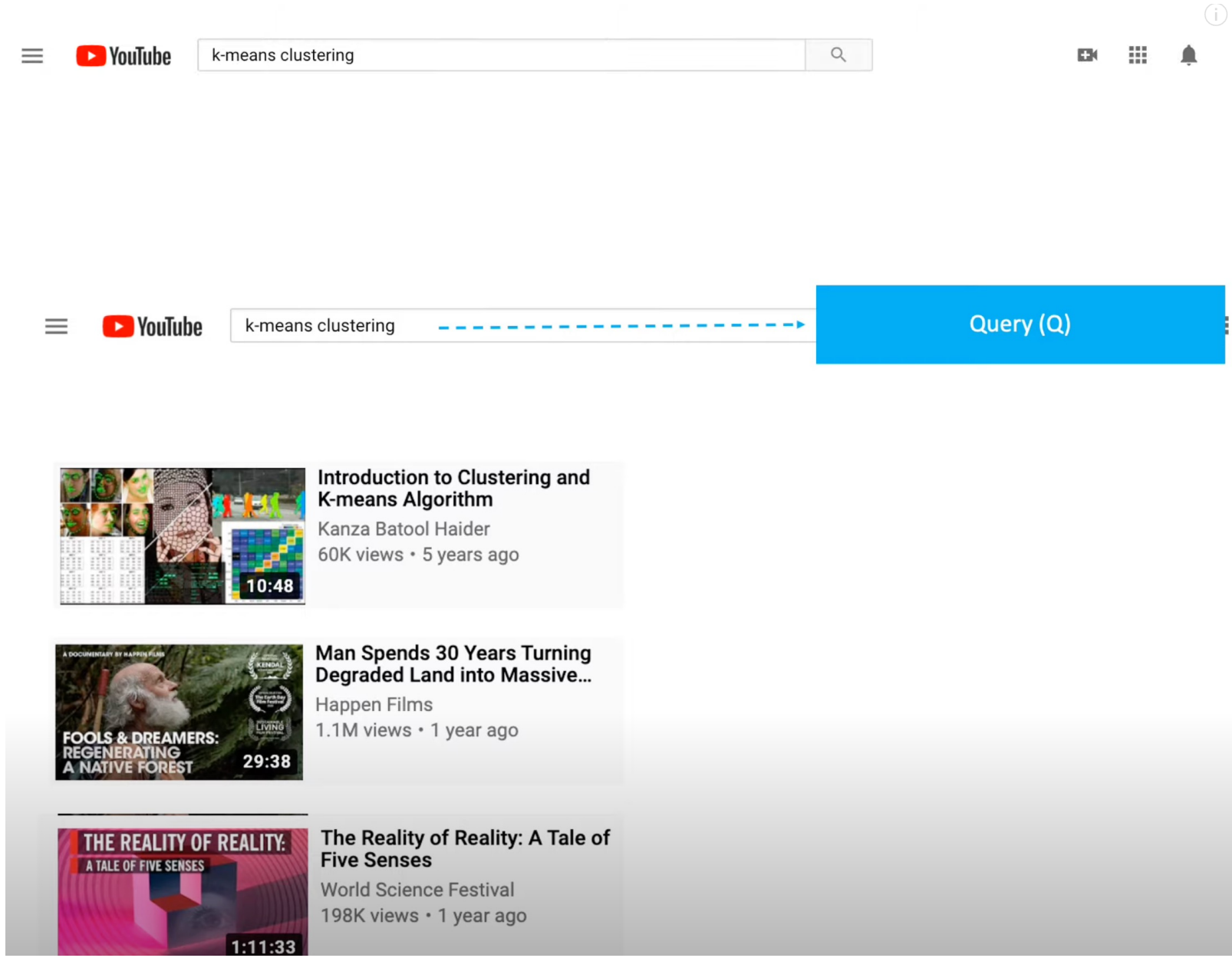
- We then pass the embedded patches and their positional encodings through the transformer encoder architecture.

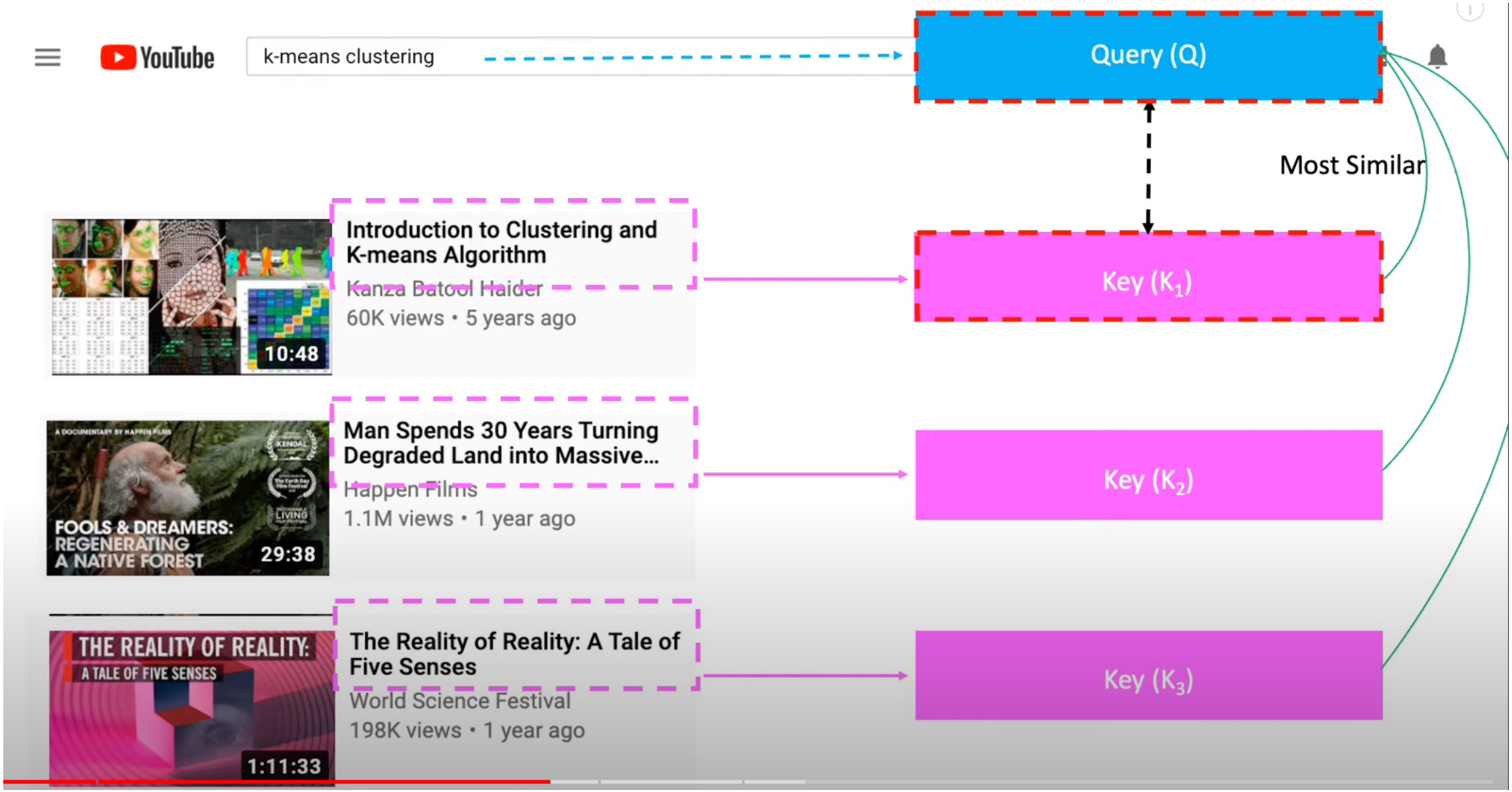




Encoder Block:




YouTube
k-means clustering
YouTube
k-means clustering
Query (Q)
10:48
Introduction to Clustering and K-means Algorithm
Kanza Batool Haider
60K views • 5 years ago
A DOCUMENTARY BY HAPPEN FILMS
FOOLS & DREAMERS: REGENERATING A NATIVE FOREST
29:38
Man Spends 30 Years Turning Degraded Land into Massive...
Happen Films
1.1M views • 1 year ago
THE REALITY OF REALITY:
A TALE OF FIVE SENSES
1:11:33
The Reality of Reality: A Tale of Five Senses
World Science Festival
198K views • 1 year ago


YouTube
k-means clustering
Query (Q)
Most Similar
Introduction to Clustering and K-means Algorithm
Kanza Batool Haider
60K views • 5 years ago
10:48
Key (K_1)
Man Spends 30 Years Turning Degraded Land into Massive...
Happen Films
1.1M views • 1 year ago
FOOLS & DREAMERS: REGENERATING A NATIVE FOREST
29:38
Key (K_2)
The Reality of Reality: A Tale of Five Senses
World Science Festival
198K views • 1 year ago
THE REALITY OF REALITY:
A TALE OF FIVE SENSES
1:11:33
Key (K_3)


YouTube
k-means clustering
Query (Q)
Introduction to Clustering and K-means Algorithm
Kanza Batool Haider
60K views • 5 years ago
10:48
Key (K_1)
Video Contents
Value (V_1)

Similarity?

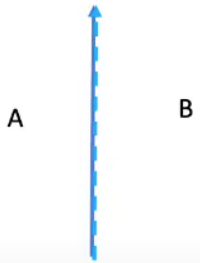
Cosine Similarity

+1 to -1

Most similar

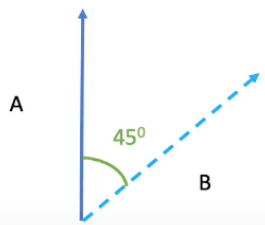
Most dissimilar

Cosine Similarity



Cos (0) = +1

Cosine Similarity



Cos (45) = +0.7

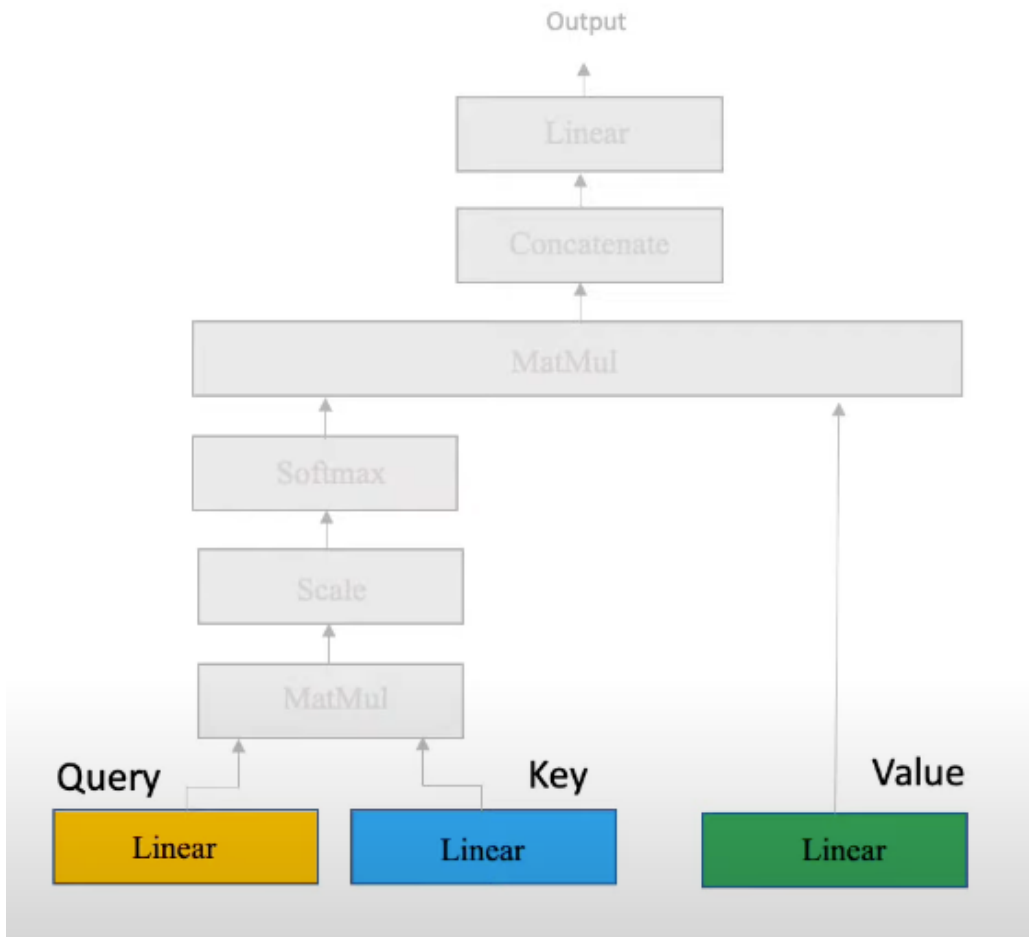
$$\text{Cos}\,(A, B) = \frac{A \,.\, B}{|A|\,|B|}$$

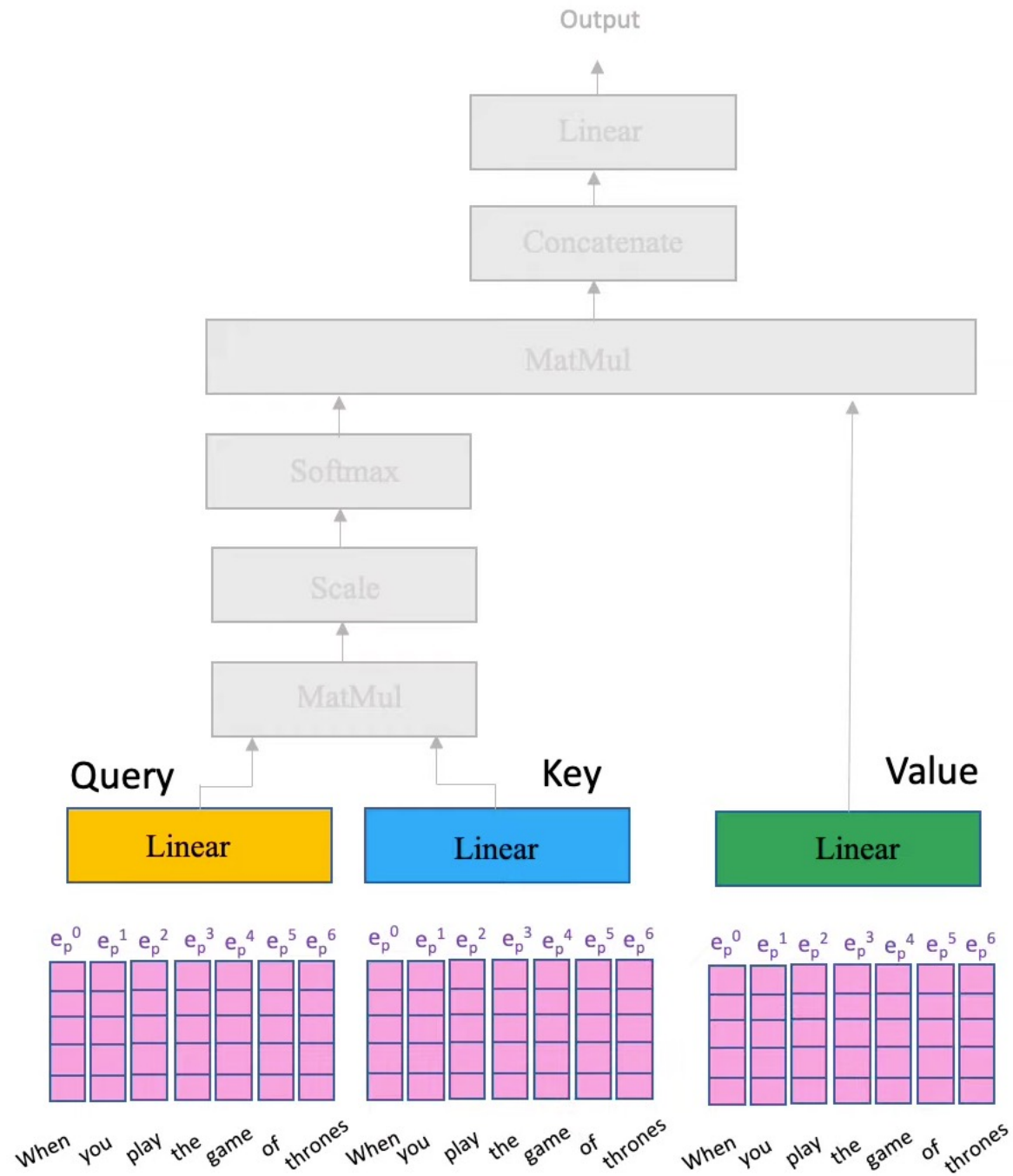
Similarity b/w Matrices

$$\text{similarity}\,(A, B) = \frac{A \,.\, B^{T}}{scaling}$$

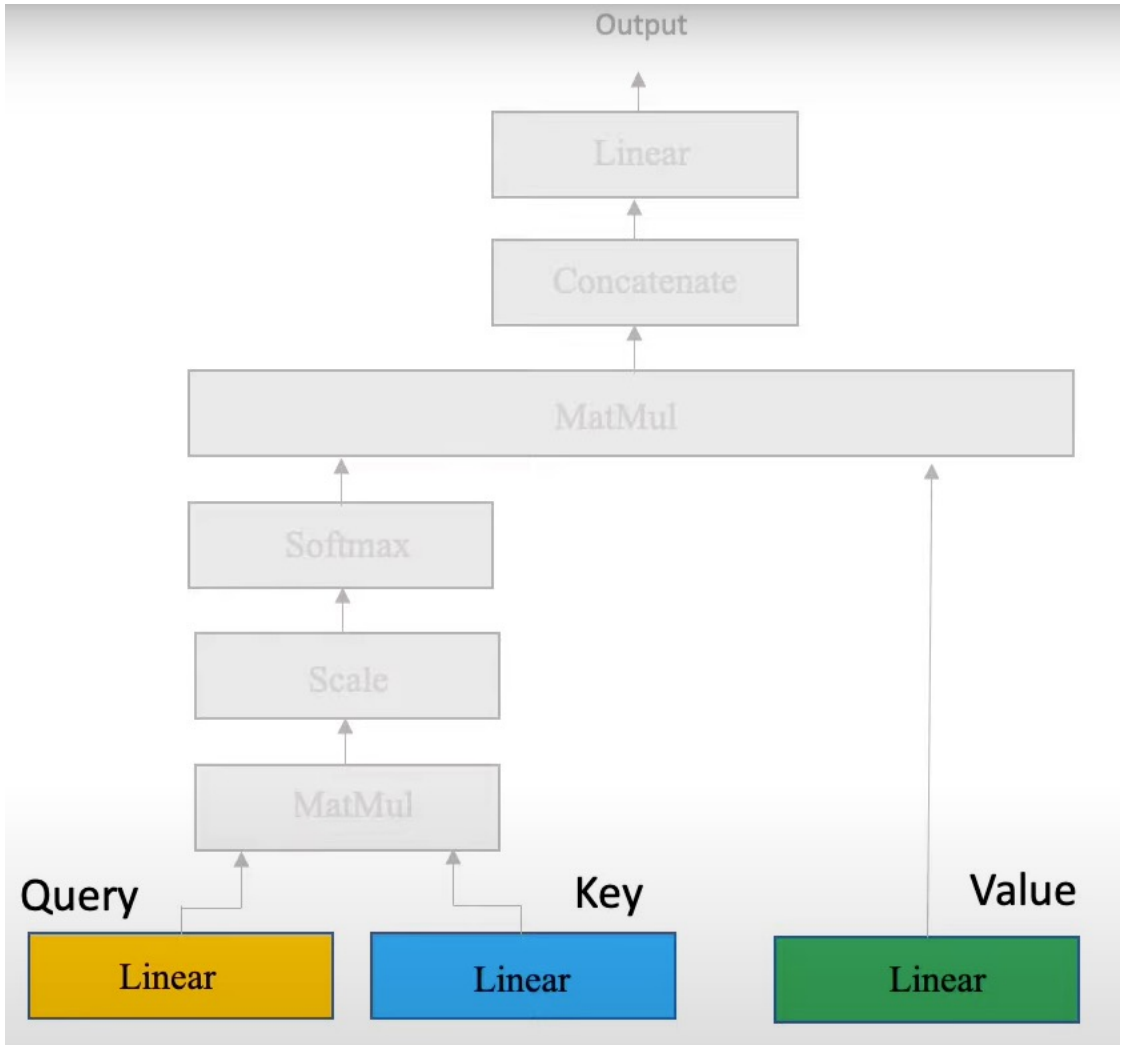
Similarity b/w Query and Key

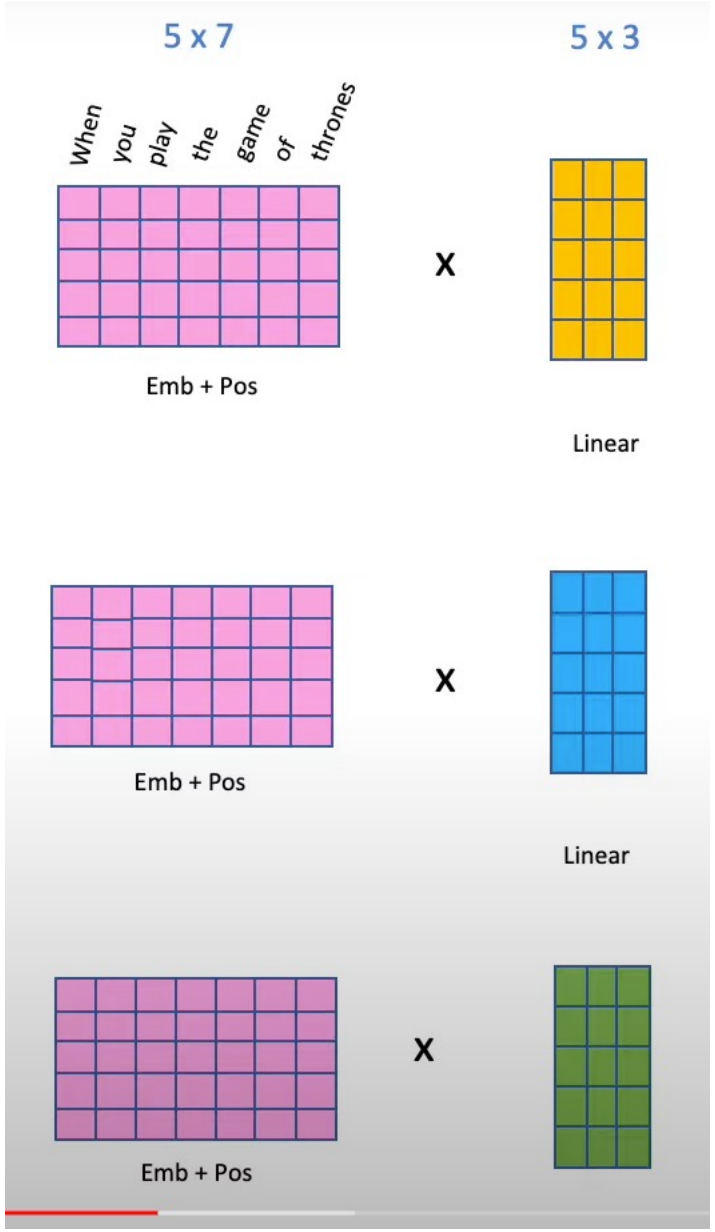
$$\text{similarity}\,(Q, K) = \frac{Q \,.\, K^{T}}{scaling}$$

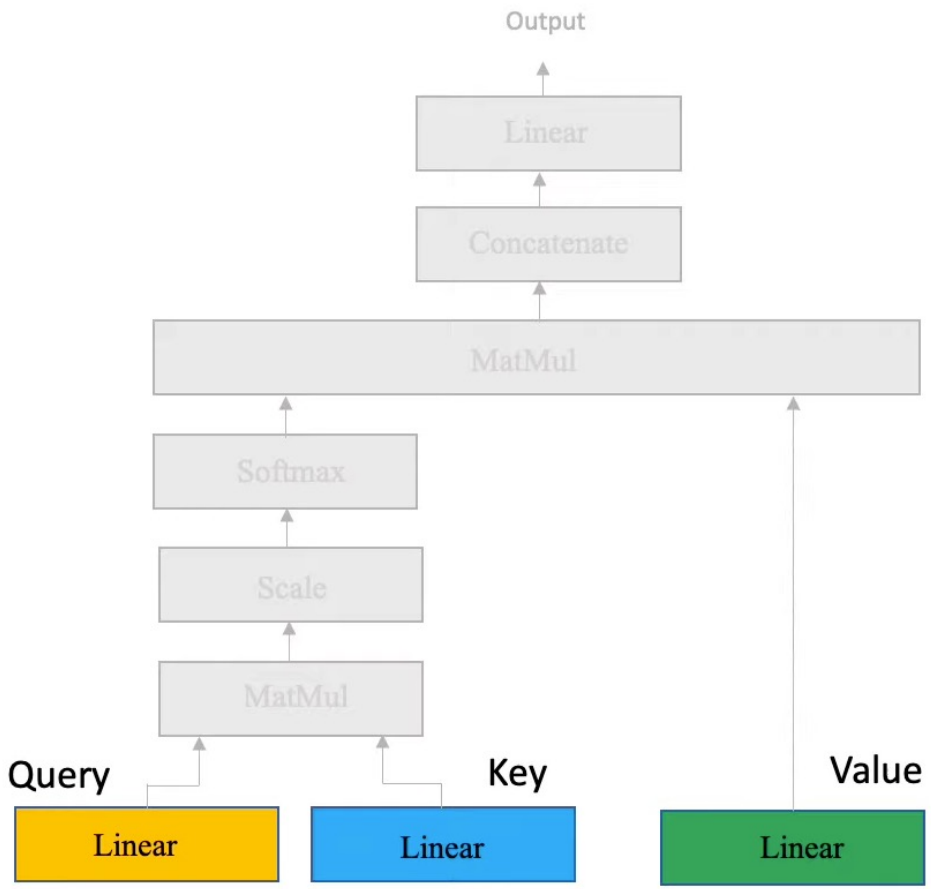



Output
Linear
Concatenate
MatMul
Softmax
Scale
MatMul
Query
Key
Value
Linear
Linear
Linear
When you play the game of thrones
When you play the game of thrones
When you play the game of thrones

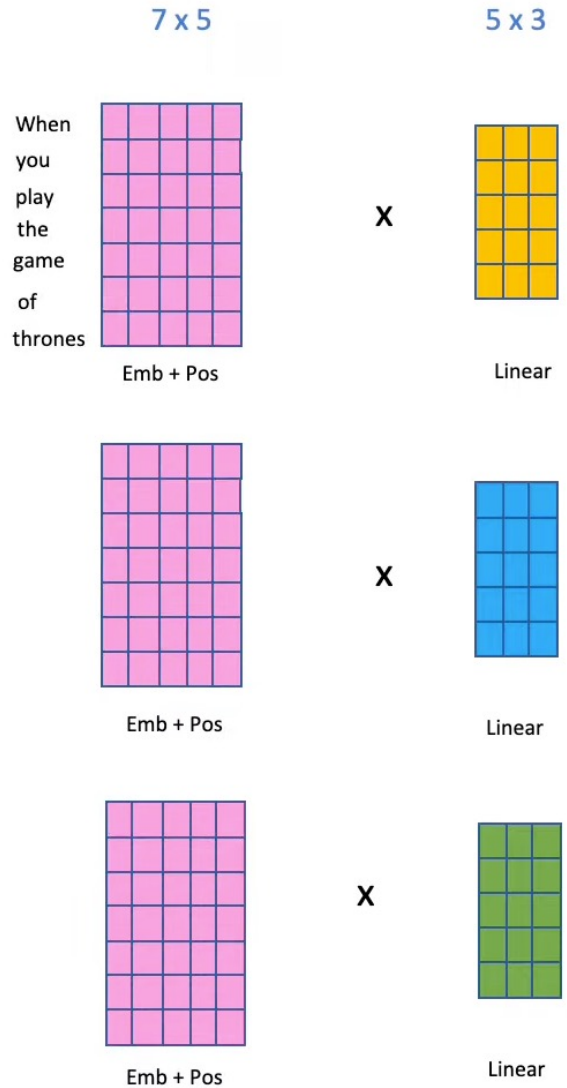
Multi-Head Attention

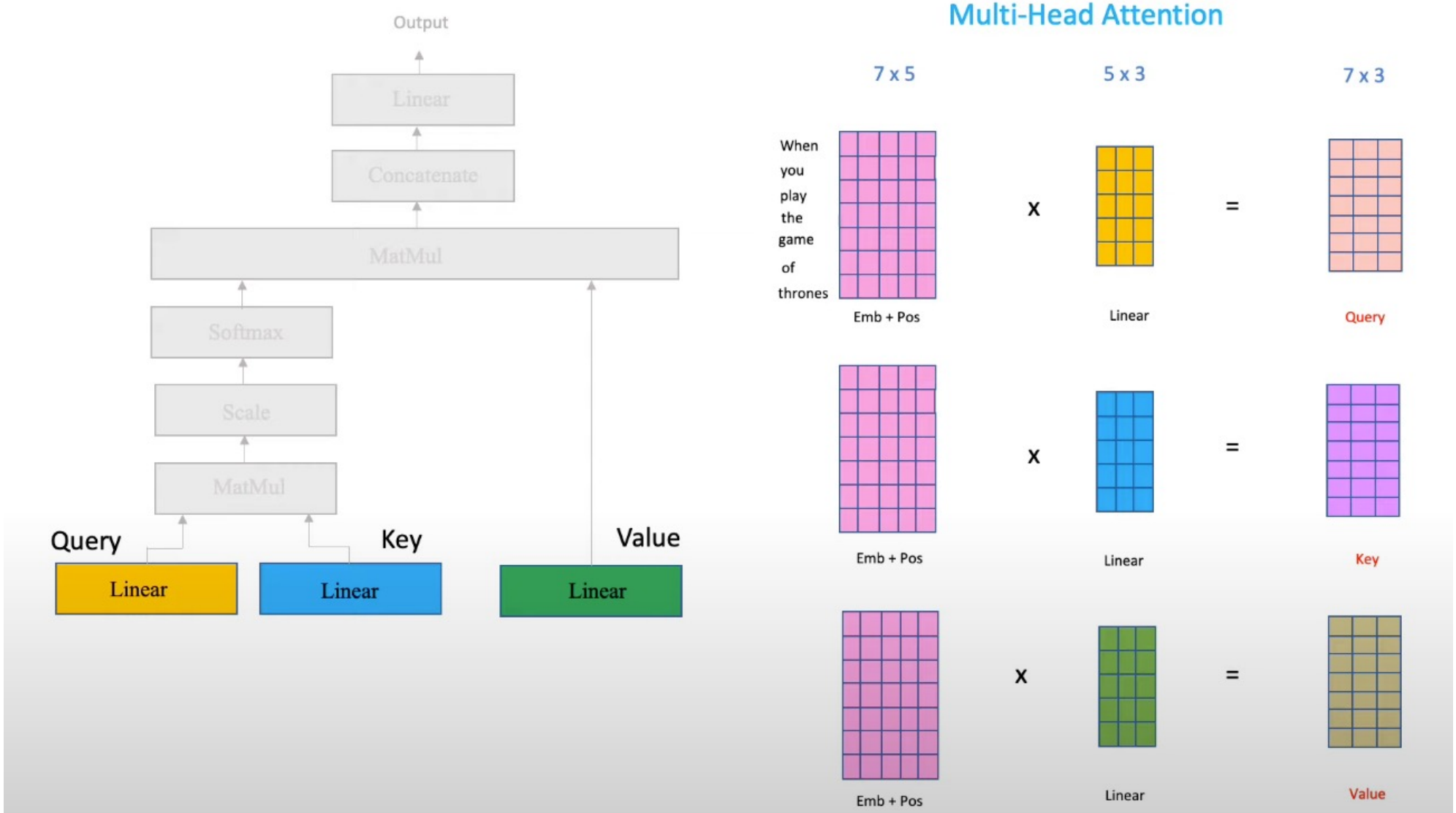


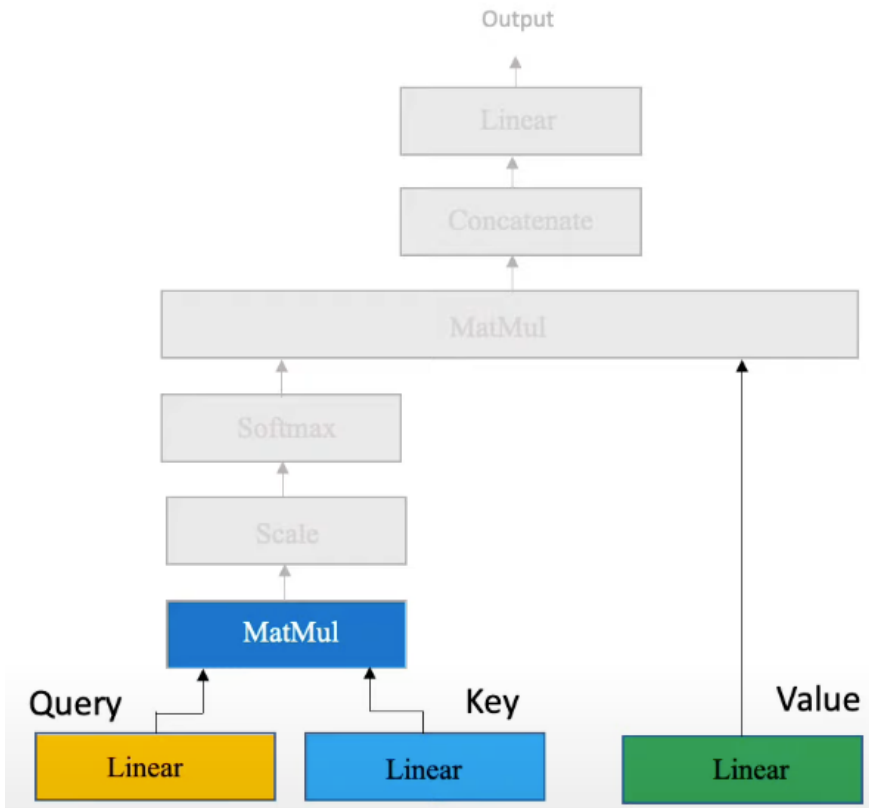




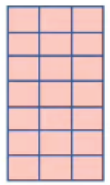
Multi-Head Attention



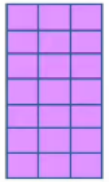

Output
Linear
Concatenate
MatMul
Softmax
Scale
MatMul
Query
Key
Value
Linear
Linear
Linear
Multi-Head Attention
7 x 5
5 x 3
7 x 3
When
you
play
the
game
of
thrones
Emb + Pos
Linear
Query
Emb + Pos
Linear
Key
Emb + Pos
Linear
Value



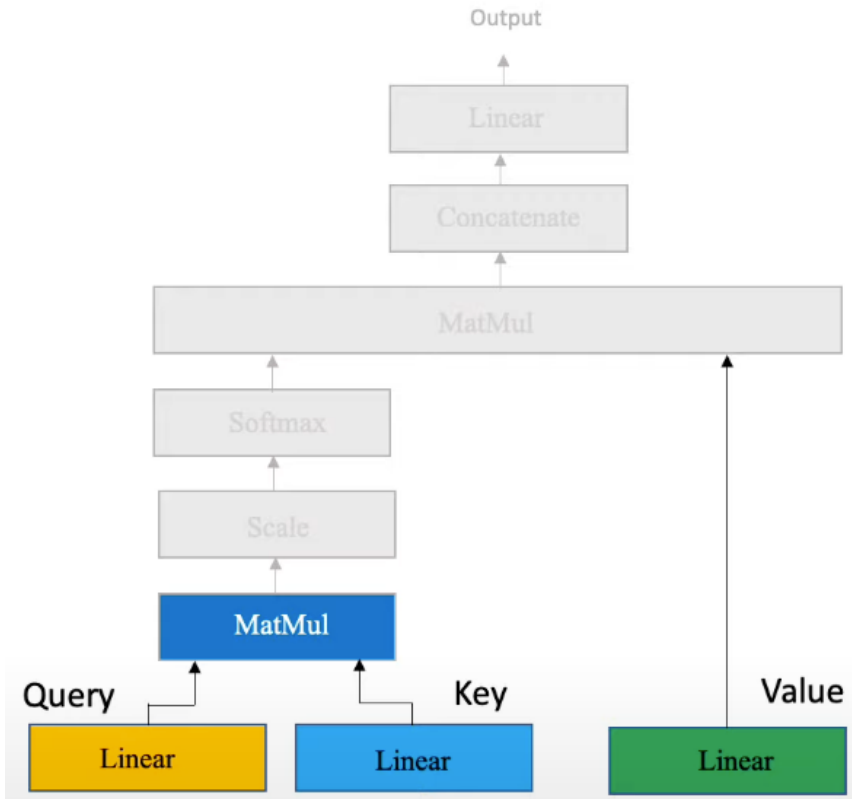
Multi-Head Attention



Query

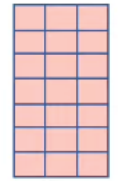


Key

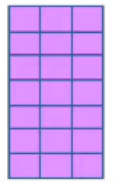


Multi-Head Attention

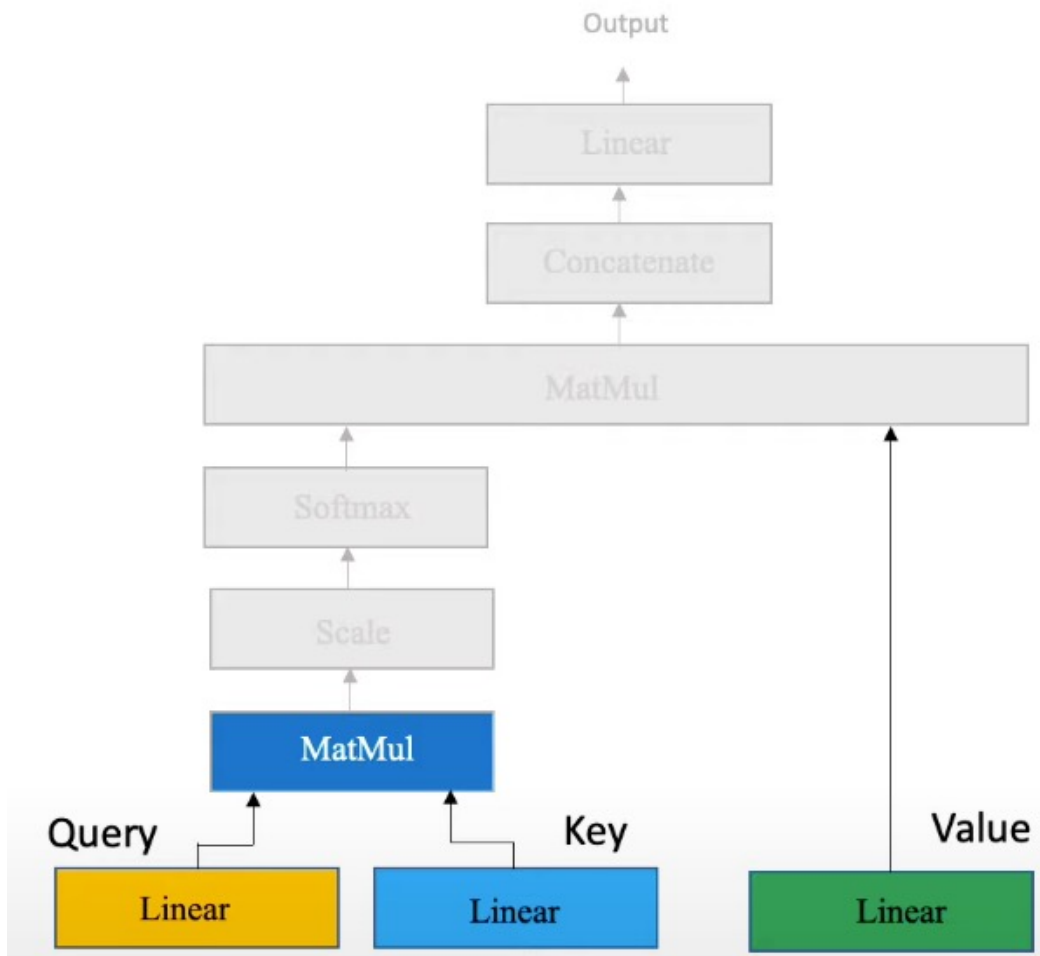
$Q \cdot K^T$



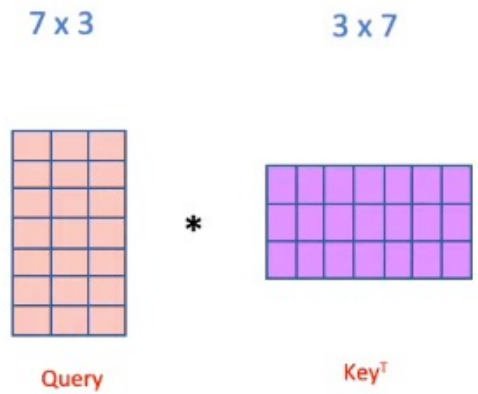
Query

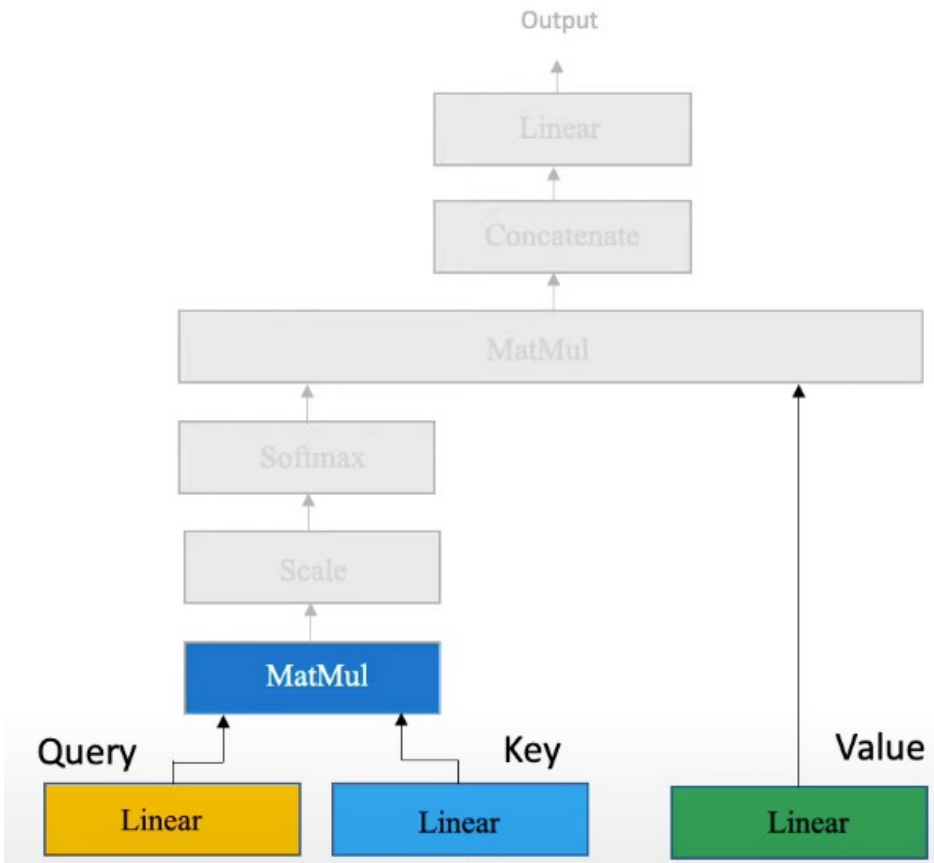


Key

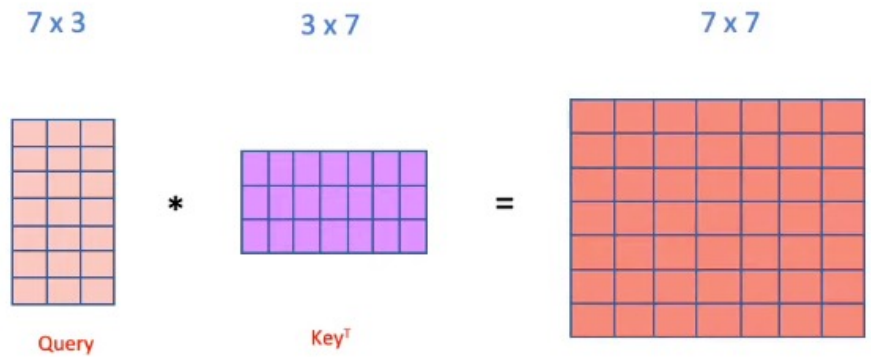


Multi-Head Attention

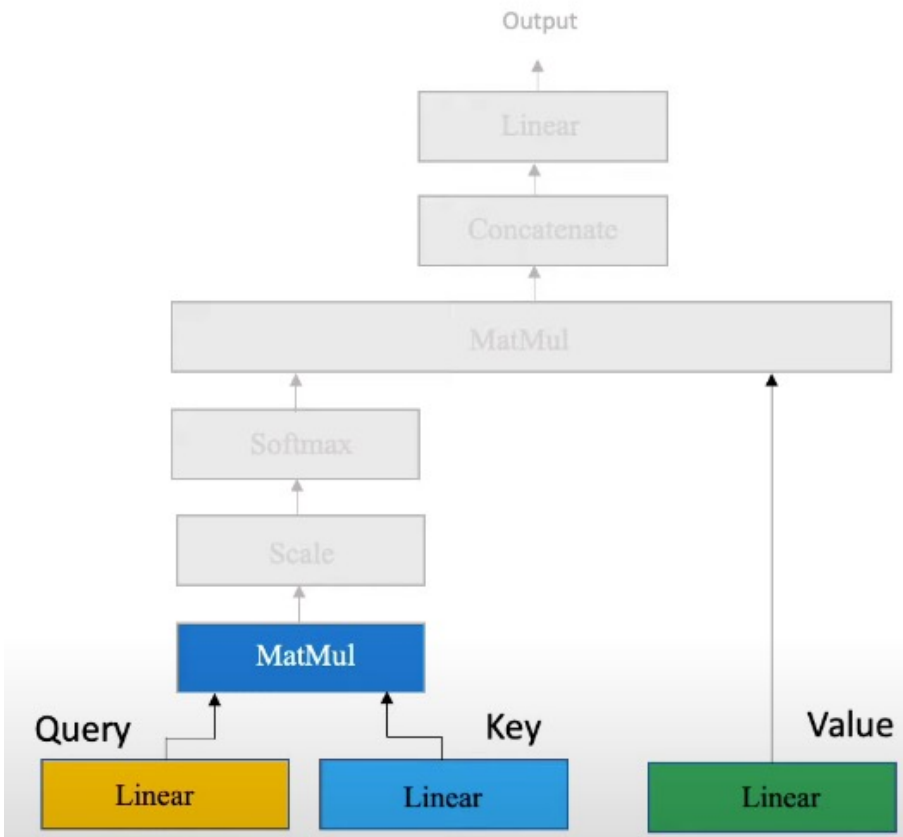




Multi-Head Attention



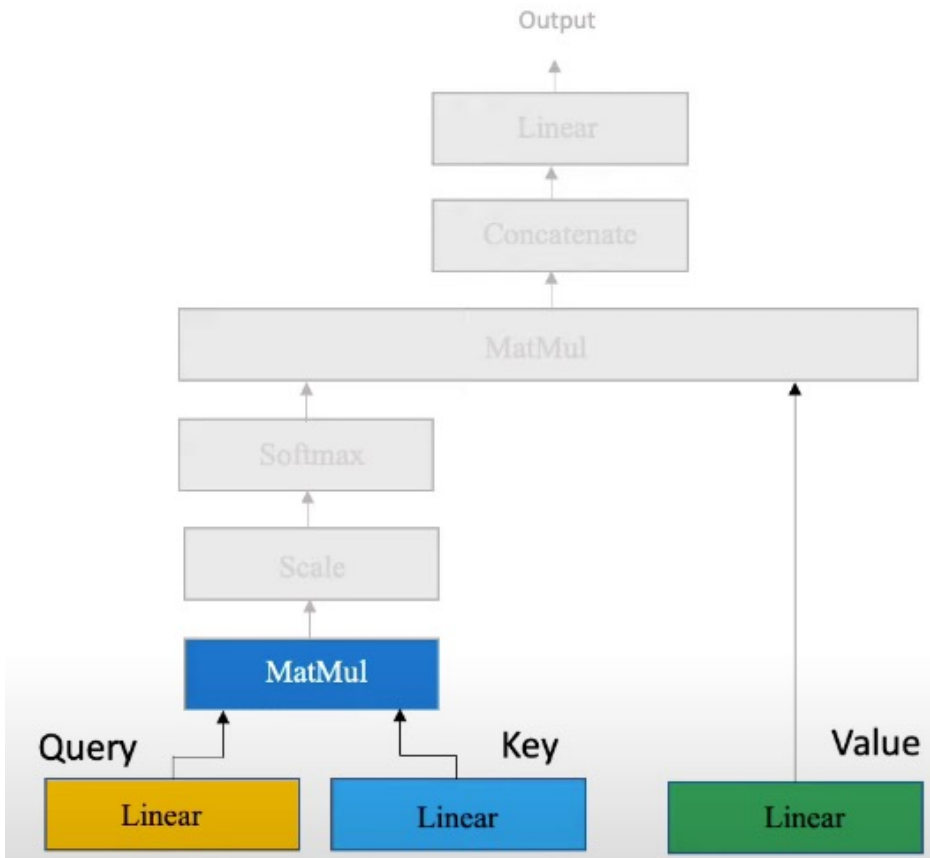
Attention Filter



Multi-Head Attention

7 x 7

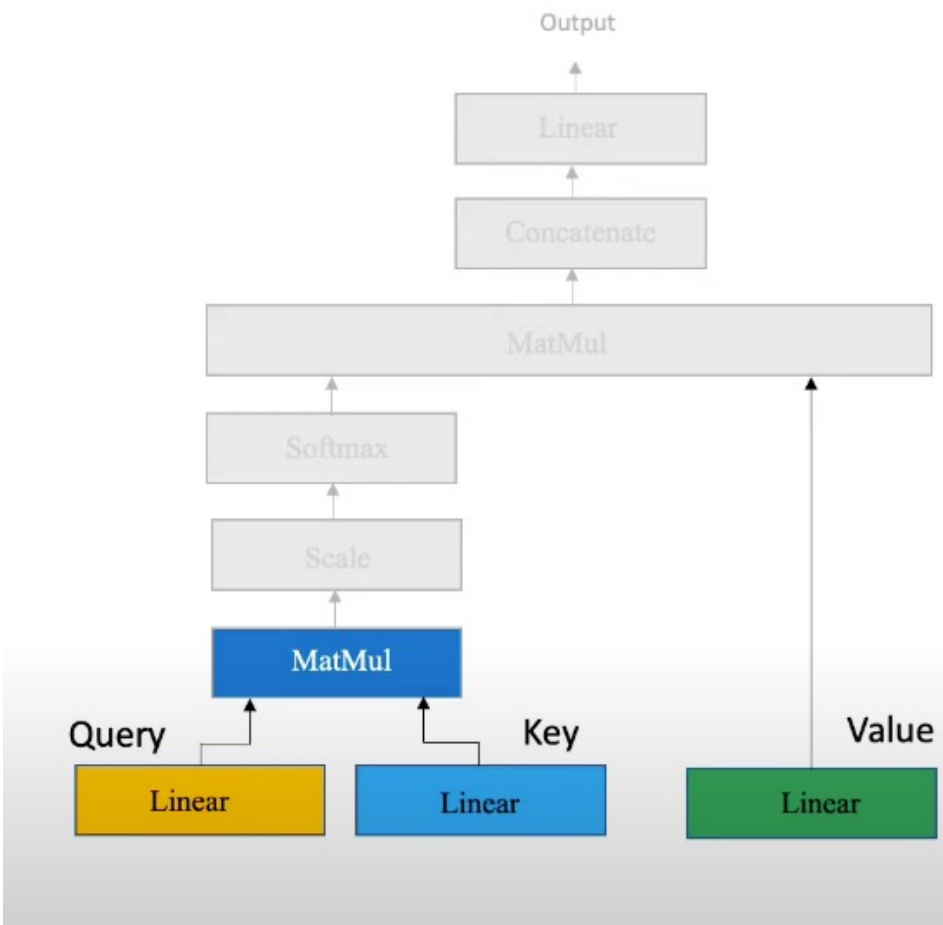
	When	you	play	the	game	of	thrones
When	89	20	41	10	55	10	59
you	20	90	81	22	70	15	72
play	41	81	95	10	90	30	92
the	10	22	10	92	88	40	89
game	55	70	90	88	98	44	87
of	10	15	30	40	44	85	59
thrones	59	72	92	90	95	59	99



Multi-Head Attention

7 x 7

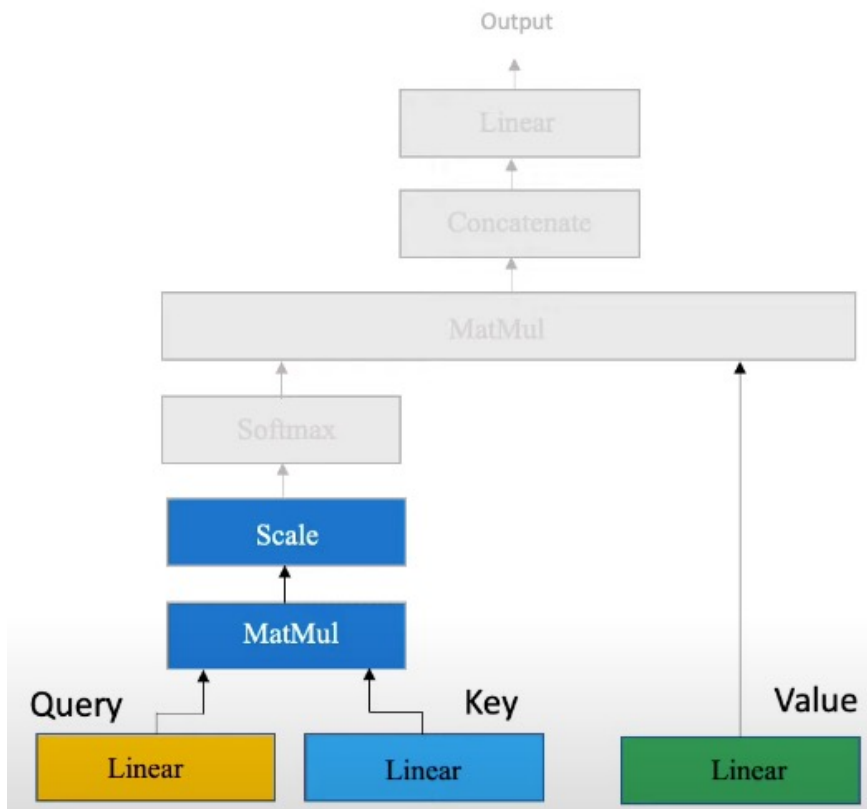
	When	you	play	the	game	of	thrones
When	89	20	41	10	55	10	59
you	20	90	81	22	70	15	72
play	41	81	95	10	90	30	92
the	10	22	10	92	88	40	89
game	55	70	90	88	98	44	87
of	10	15	30	40	44	85	59
thrones	59	72	92	90	95	59	99



Multi-Head Attention

7 x 7

	When	you	play	the	game	of	thrones
When	89	20	41	10	55	10	59
you	20	90	81	22	70	15	72
play	41	81	95	10	90	30	92
the	10	22	10	92	88	40	89
game	55	70	90	88	98	44	87
of	10	15	30	40	44	85	59
thrones	59	72	92	90	95	59	99

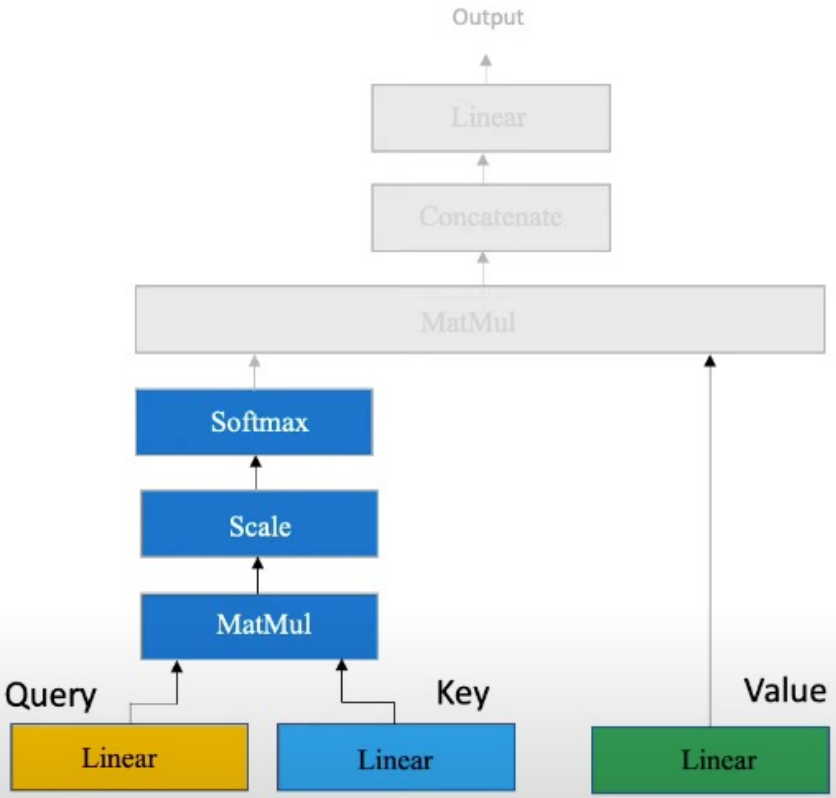


Multi-Head Attention

7 x 7

	When	you	play	the	game	of	thrones
When	89	20	41	10	55	10	59
you	20	90	81	22	70	15	72
play	41	81	95	10	90	30	92
the	10	22	10	92	88	40	89
game	55	70	90	88	98	44	87
of	10	15	30	40	44	85	59
thrones	59	72	92	90	95	59	99

$$\frac{}{\sqrt{d_k}}$$



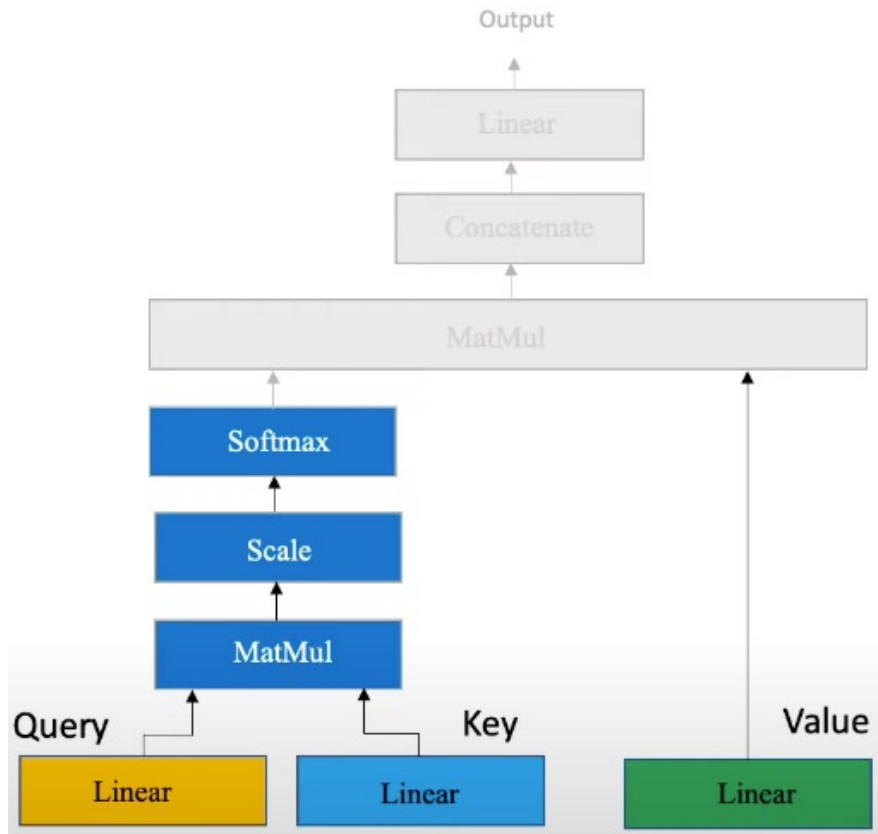
Softmax

Multi-Head Attention

7 x 7

	When	you	play	the	game	of	thrones
When	33.6	7.6	15.5	3.8	20.8	3.8	22.3
you	7.6	34.0	30.6	8.3	26.5	5.7	27.2
play	15.5	30.6	35.9	3.8	34.0	11.3	34.8
the	3.8	8.3	3.8	34.8	33.3	15.1	33.6
game	20.8	26.5	34.0	33.3	37.0	16.6	35.9
of	3.8	5.7	11.3	15.1	16.6	32.1	22.3
thrones	22.3	27.2	34.8	34.0	35.9	22.3	37.4

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_i)}$$



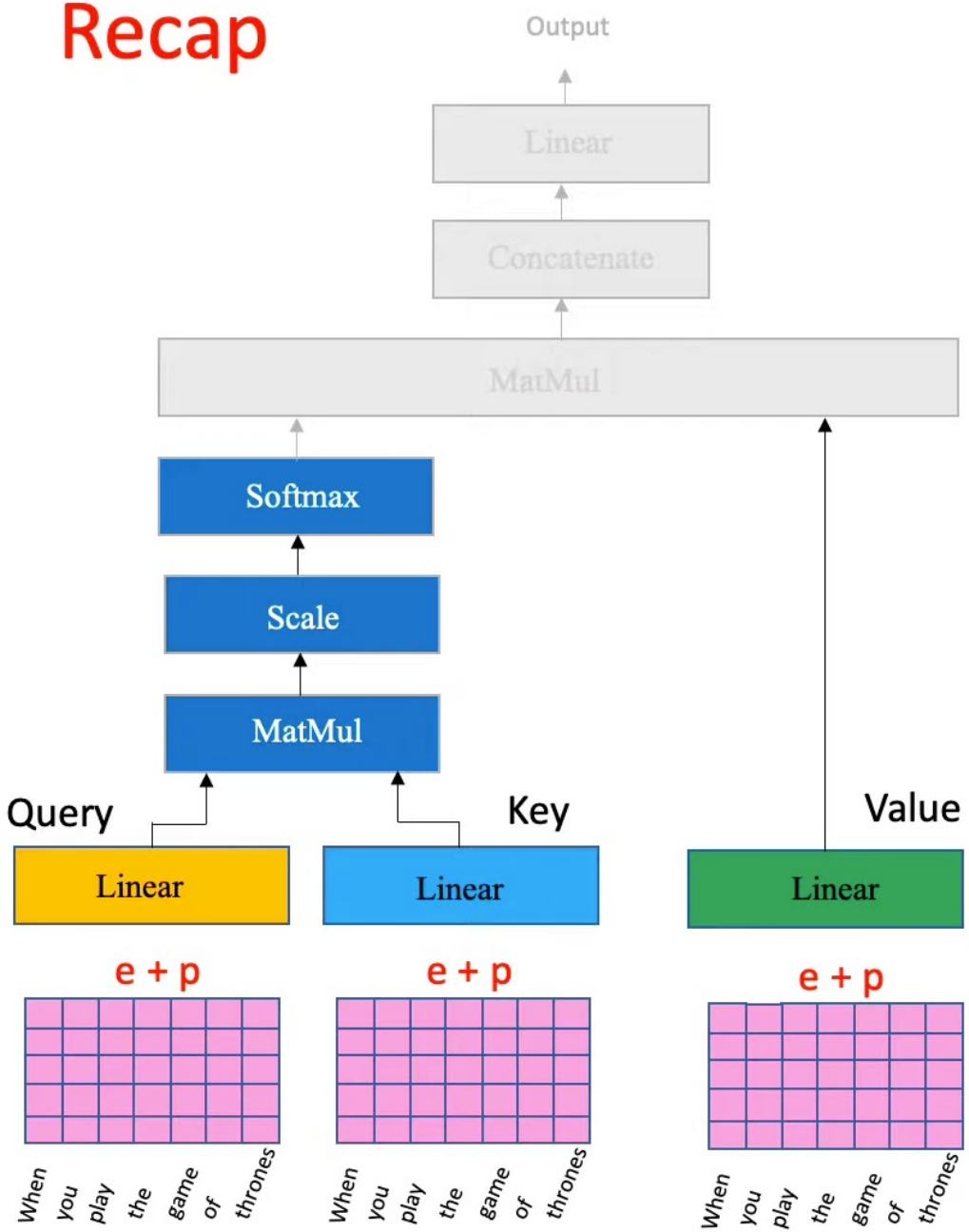
Multi-Head Attention

7 x 7

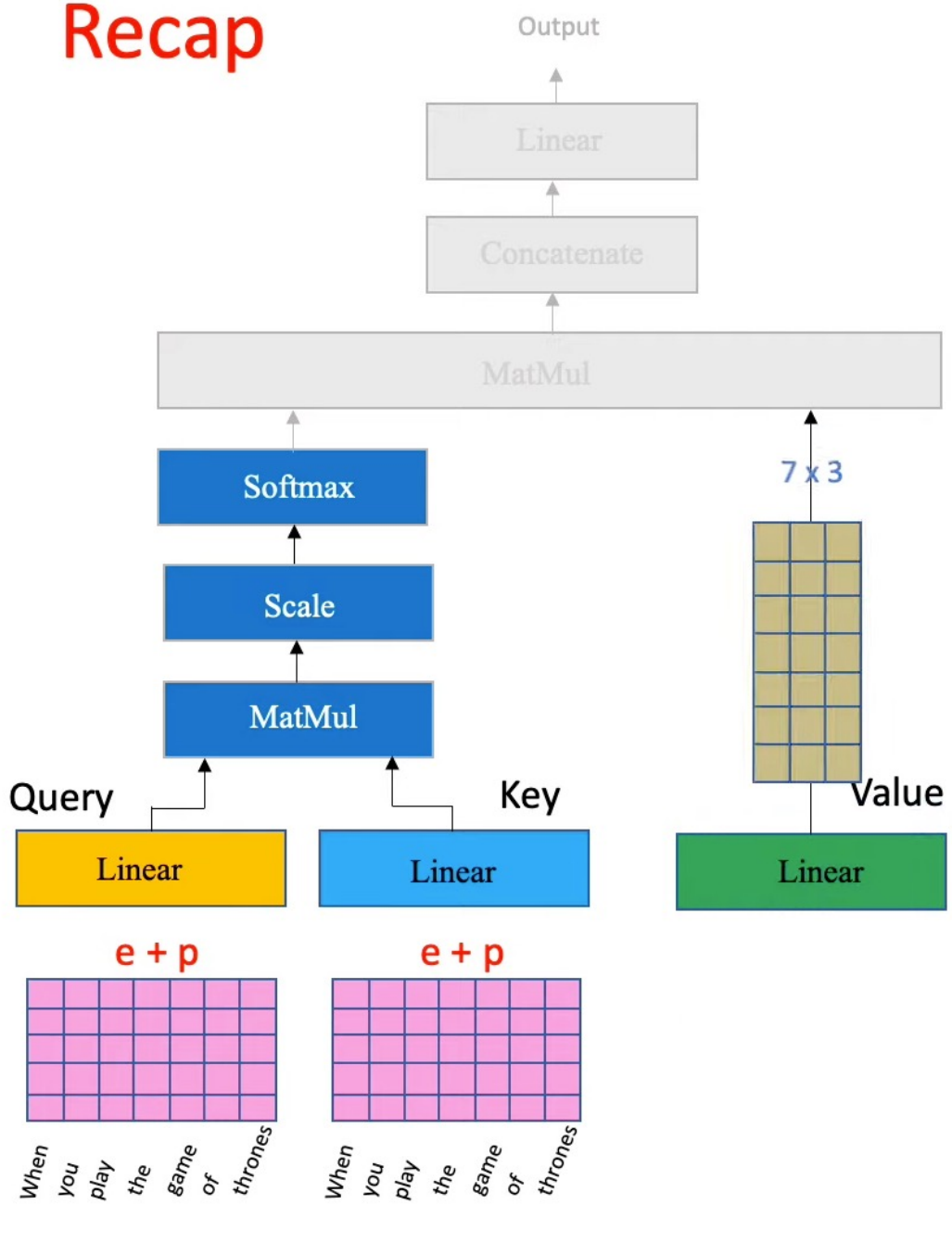
	When	you	play	the	game	of	thrones
When	1.00	0.00	0.00	0.00	0.00	0.00	0.00
you	0.00	0.97	0.03	0.00	0.00	0.00	0.00
play	0.00	0.00	0.68	0.00	0.10	0.00	0.22
the	0.00	0.00	0.00	0.65	0.14	0.00	0.21
game	0.00	0.00	0.03	0.02	0.72	0.00	0.23
of	0.00	0.00	0.00	0.00	0.00	1.00	0.00
thrones	0.00	0.00	0.05	0.03	0.17	0.00	0.75

Attention Filter

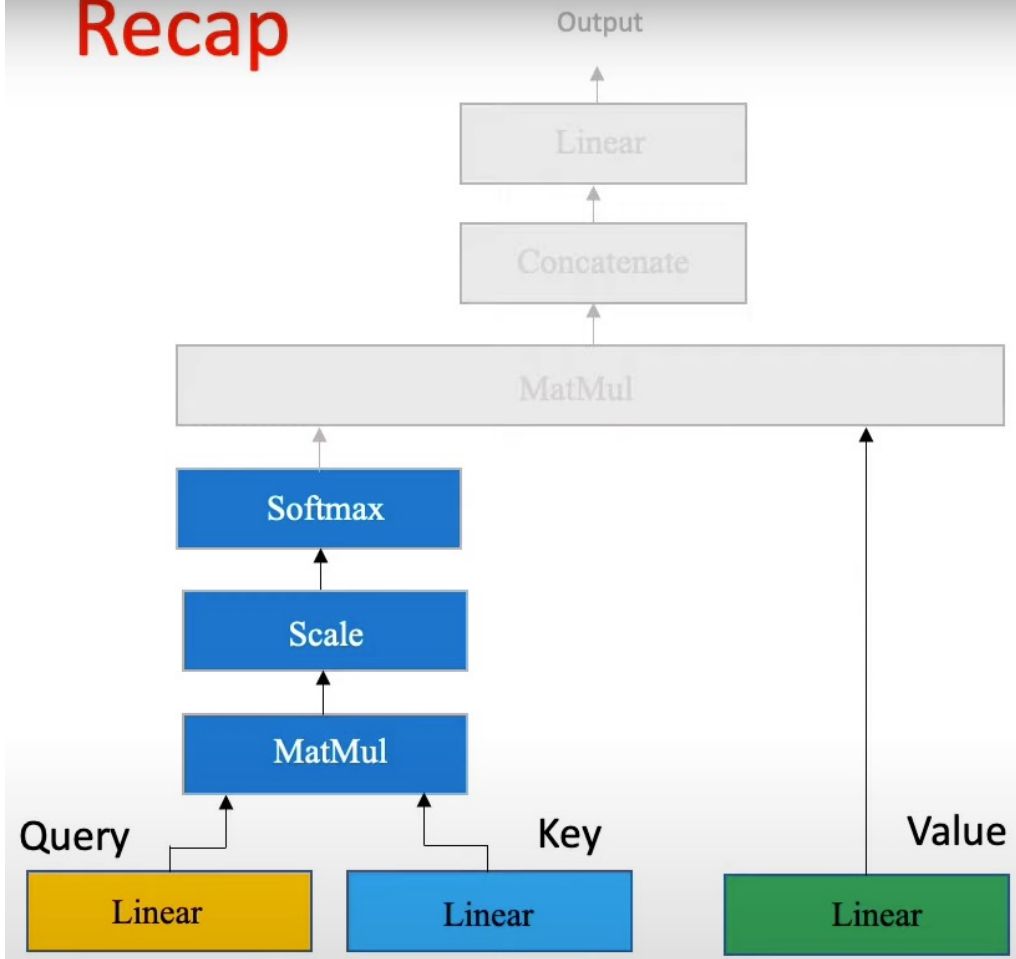
Recap


Output
Linear
Concatenate
MatMul
Softmax
Scale
MatMul
Query
Key
Value
Linear
Linear
Linear
e + p
e + p
e + p
When you play the game of thrones
When you play the game of thrones
When you play the game of thrones

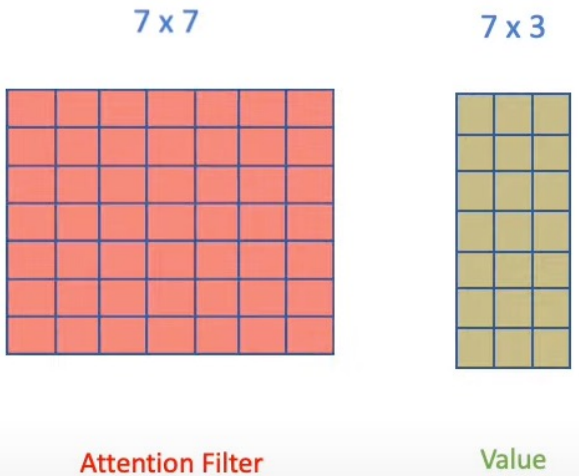
Recap


Output
Linear
Concatenate
MatMul
Softmax
Scale
MatMul
7 x 3
Query
Key
Value
Linear
Linear
Linear
e + p
e + p
When you play the game of thrones
When you play the game of thrones

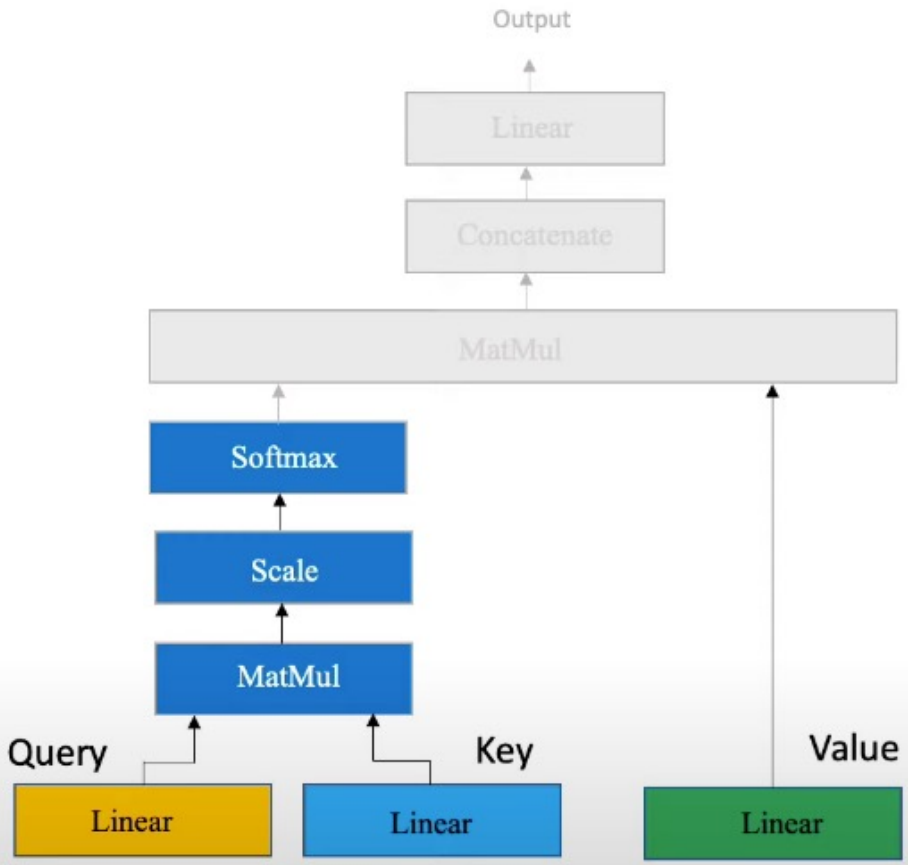
Recap

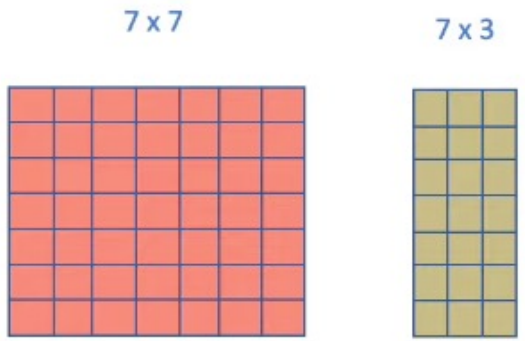


Multi-Head Attentio



Multi-Head Attention

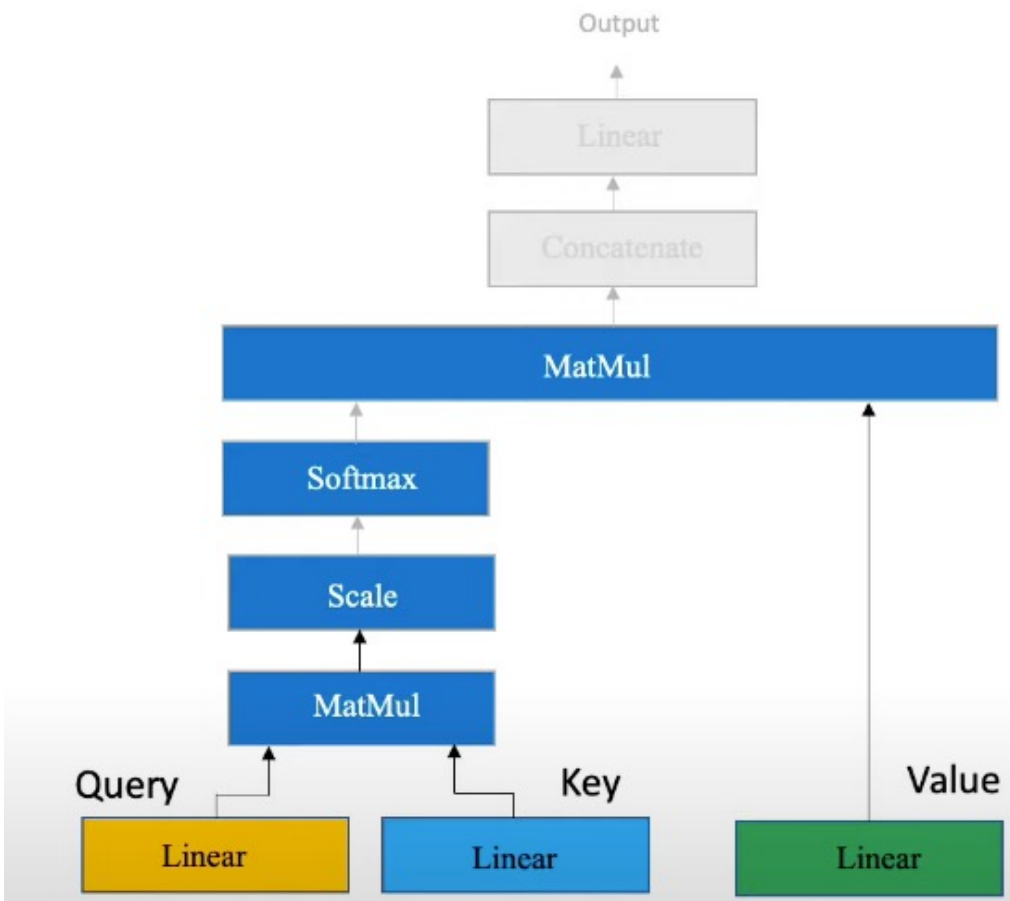




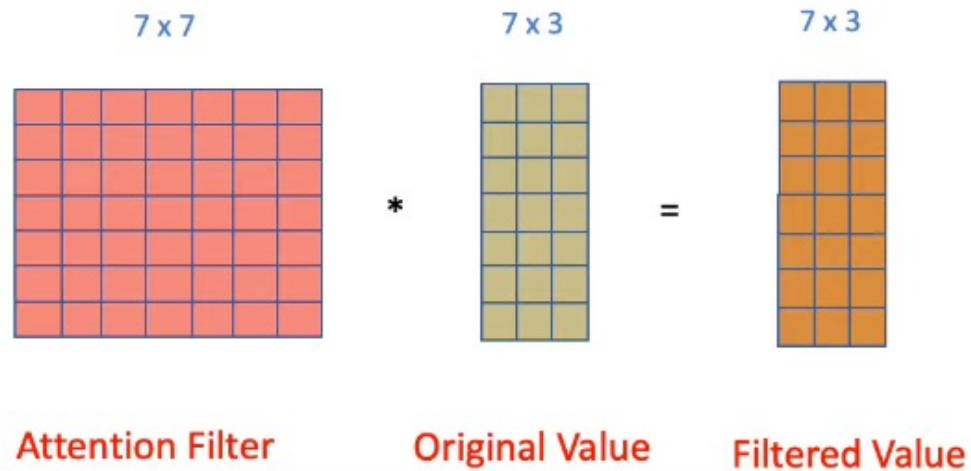
Attention Filter

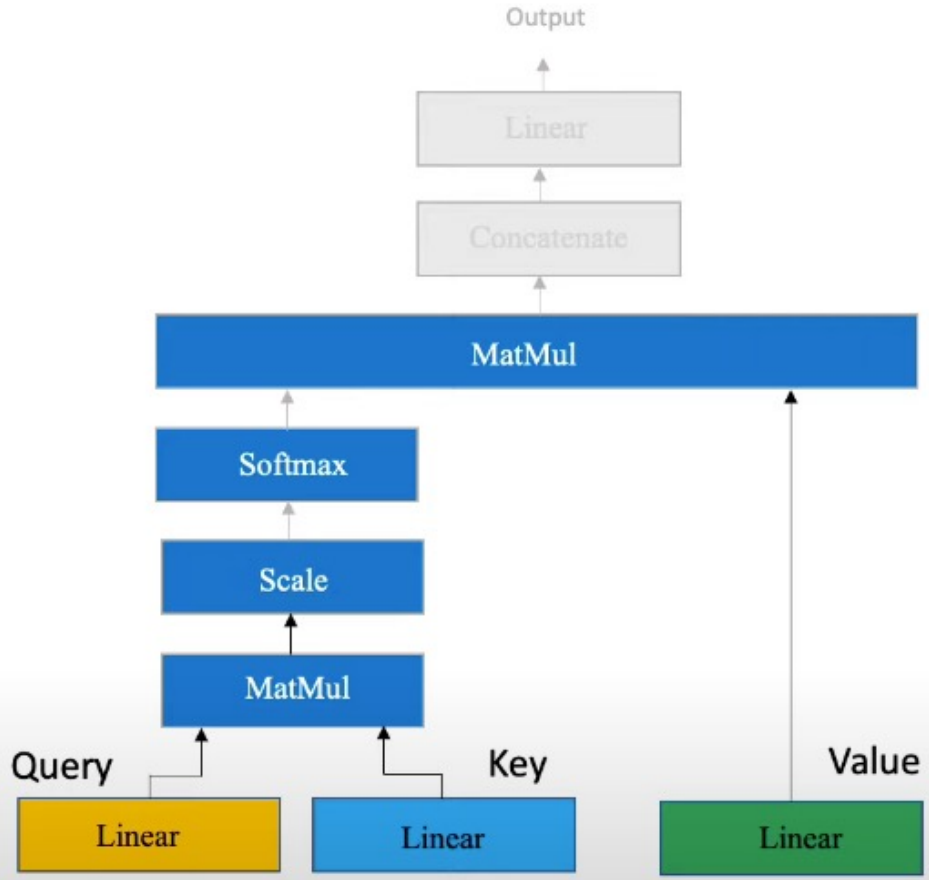
Value

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



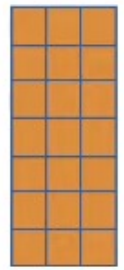
Multi-Head Attention





Multi-Head Attention

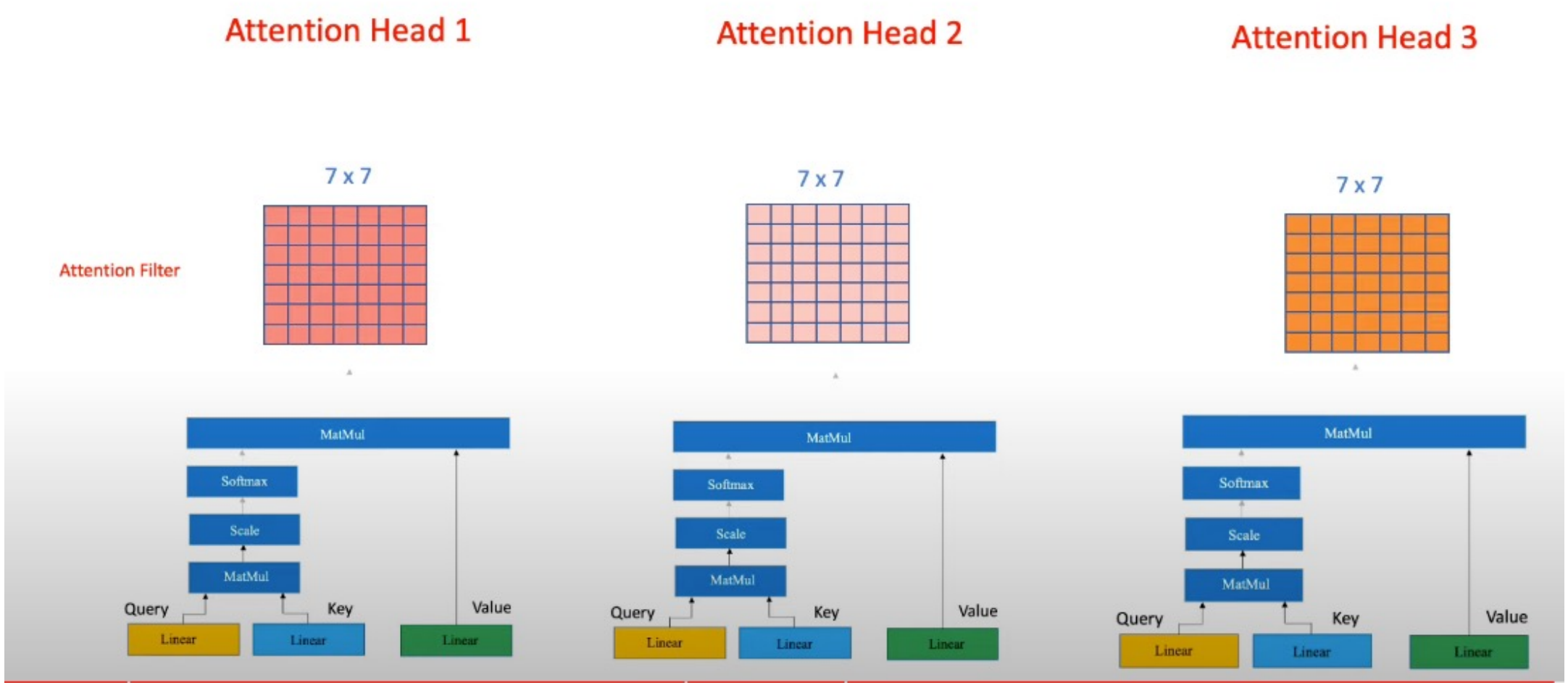
7 x 3

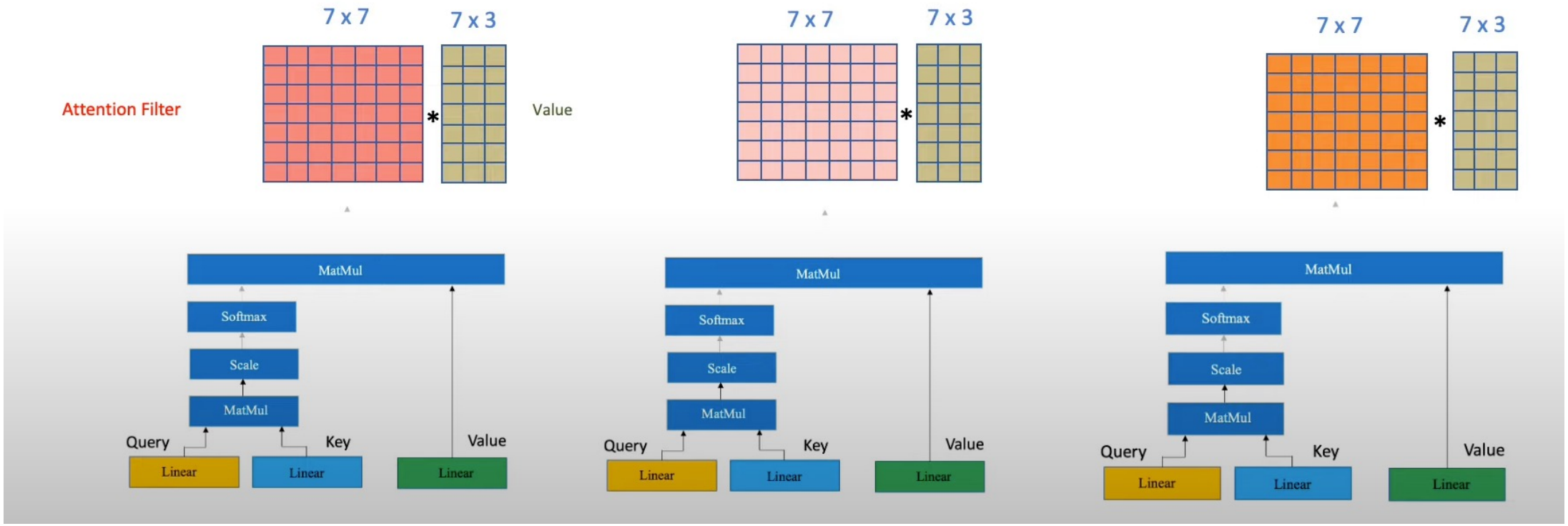


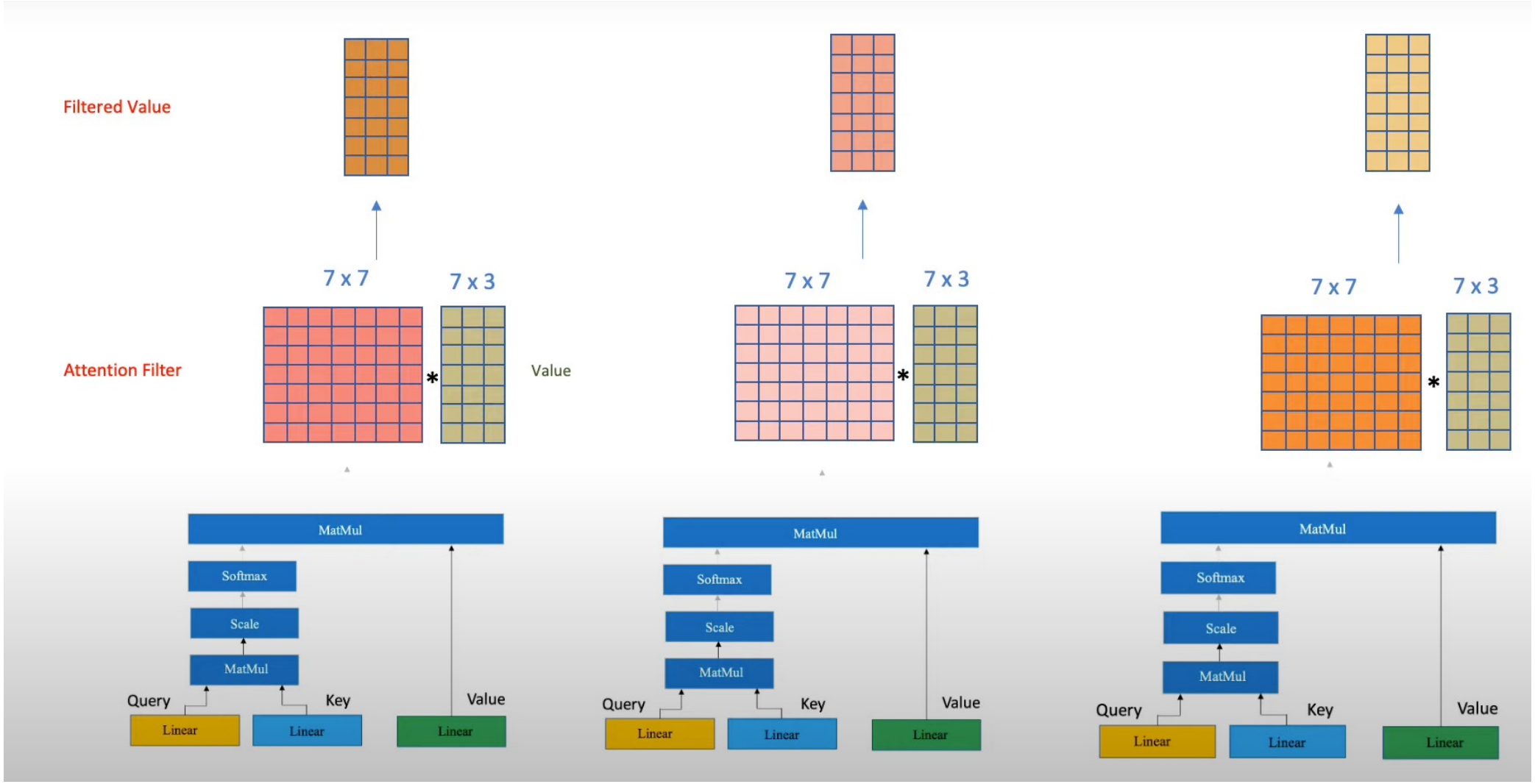
Filtered Value

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

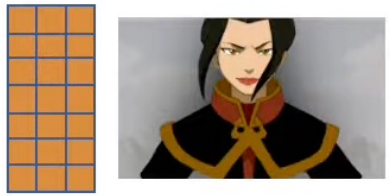
Multi-Head Attention

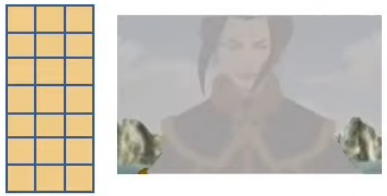



Attention Filter
7 x 7
7 x 3
Value
MatMul
Softmax
Scale
MatMul
Query
Key
Value
Linear
Linear
Linear
7 x 7
7 x 3
MatMul
Softmax
Scale
MatMul
Query
Key
Value
Linear
Linear
Linear
7 x 7
7 x 3
MatMul
Softmax
Scale
MatMul
Query
Key
Value
Linear
Linear
Linear


Filtered Value
7 x 7
7 x 3
Attention Filter
Value
MatMul
Softmax
Scale
MatMul
Query
Key
Value
Linear
Linear
Linear
MatMul
Softmax
Scale
MatMul
Query
Key
Value
Linear
Linear
Linear
7 x 7
7 x 3
MatMul
Softmax
Scale
MatMul
Query
Key
Value
Linear
Linear
Linear

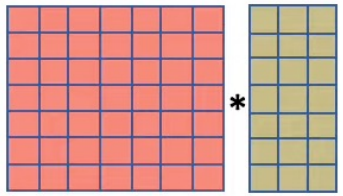
Filtered Value





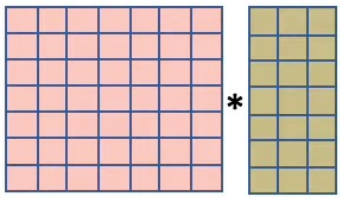
7 x 7 7 x 3

Attention Filter

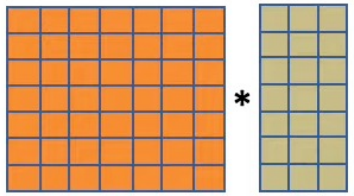


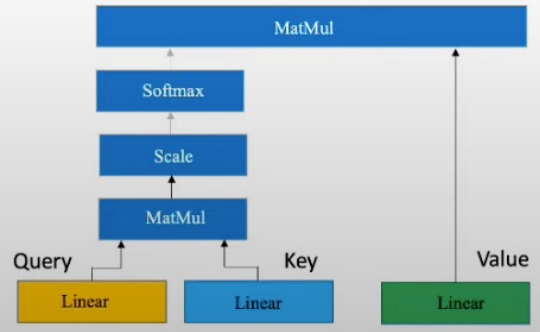
Value

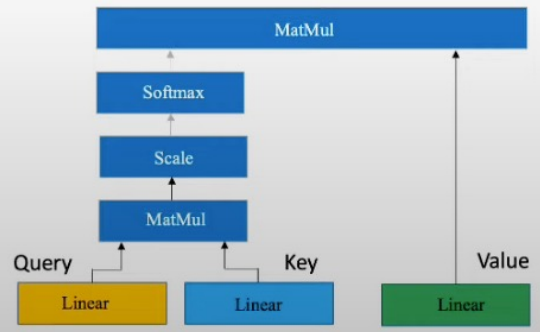
7 x 7 7 x 3

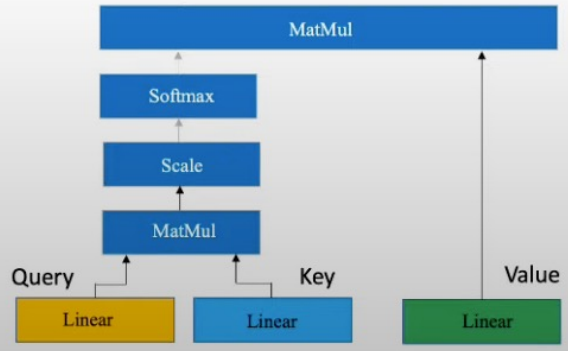


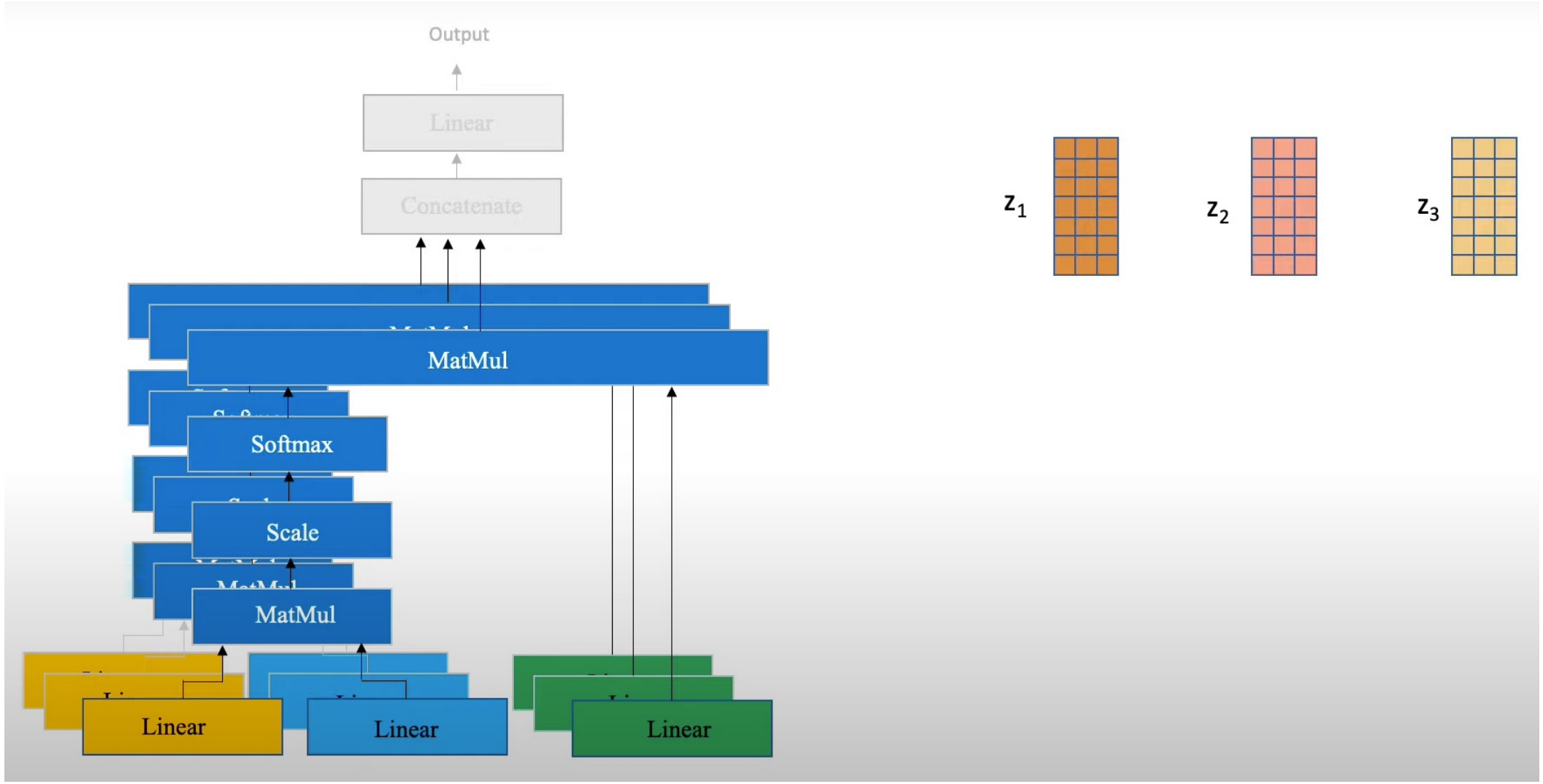
7 x 7 7 x 3

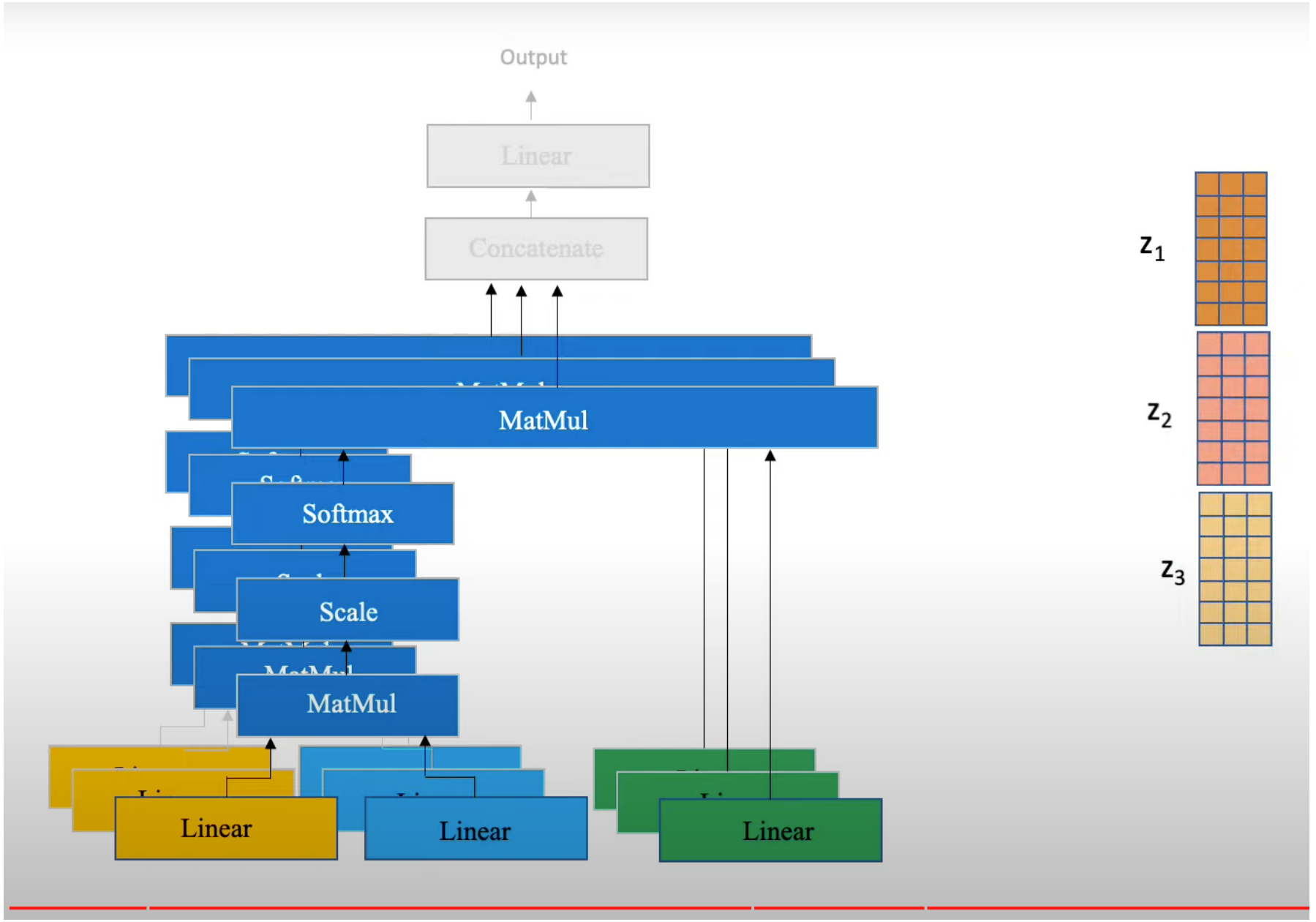


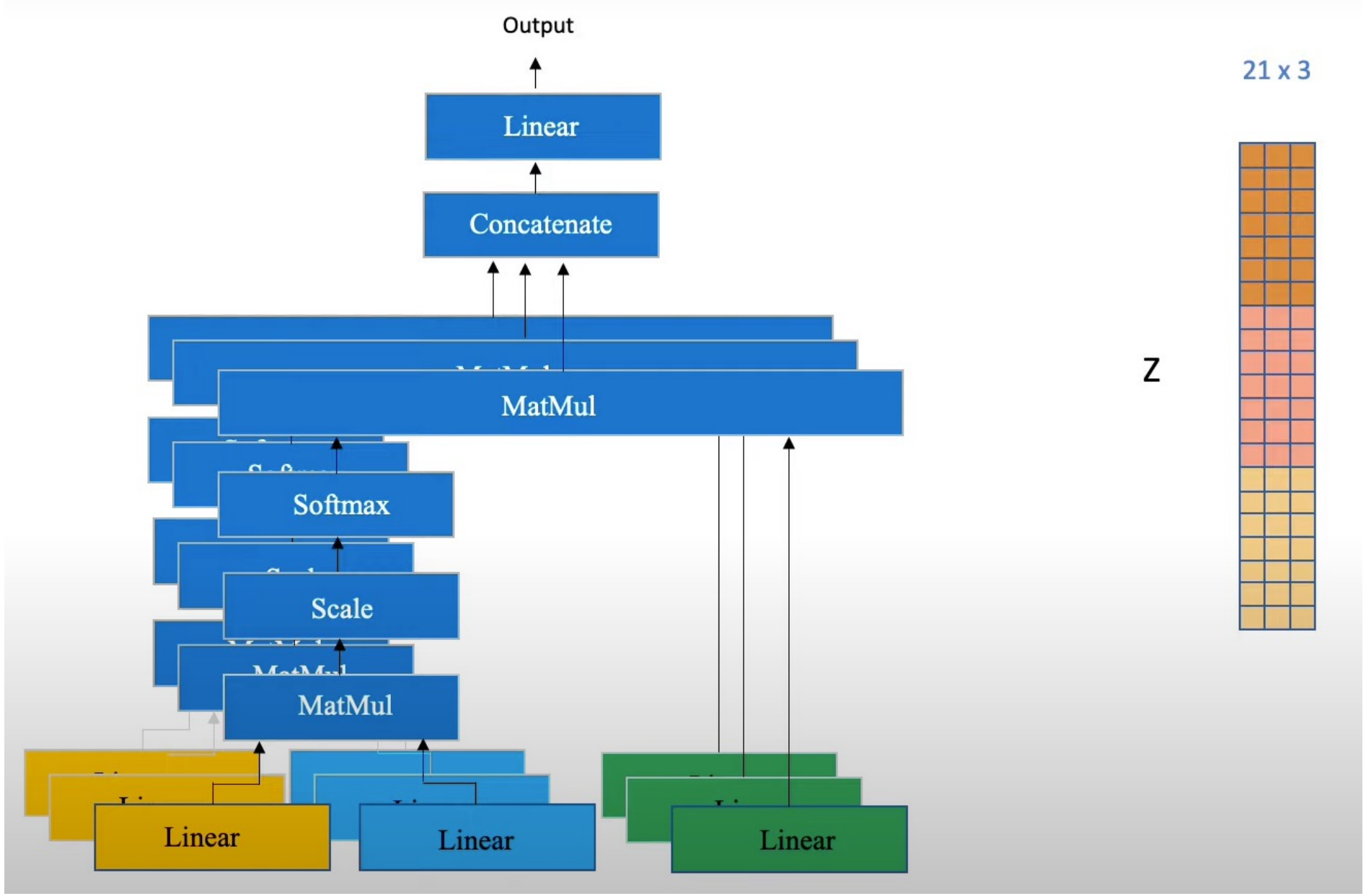


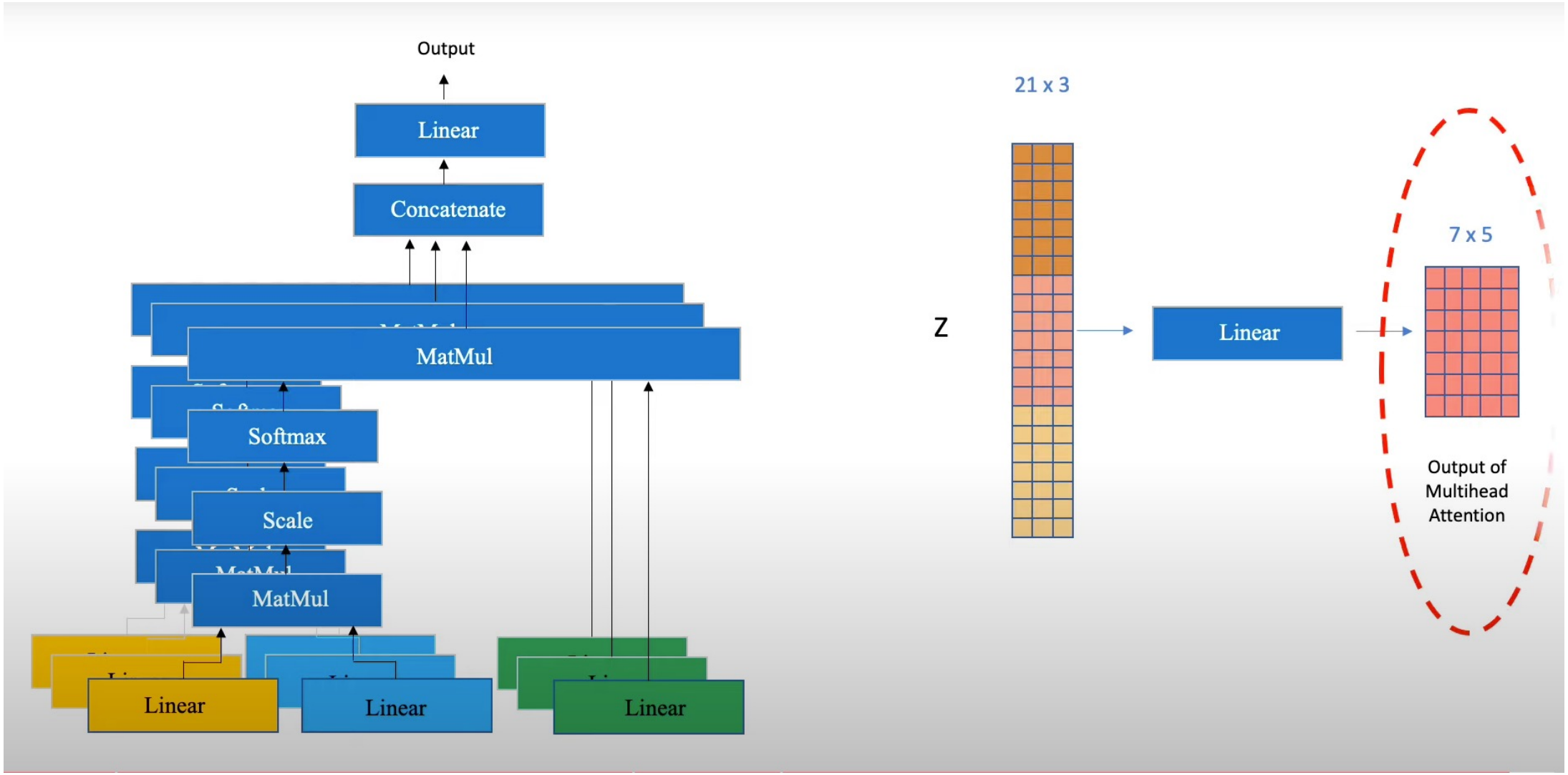





Output
Linear
Concatenate
MatMul
Softmax
Scale
MatMul
Linear
Linear
Linear
z_1
z_2
z_3


Output
Linear
Concatenate
MatMul
Softmax
Scale
MatMul
Linear
Linear
Linear
z1
z2
z3


Output
Linear
Concatenate
MatMul
Softmax
Scale
MatMul
Linear
Linear
Linear
21 x 3
Z

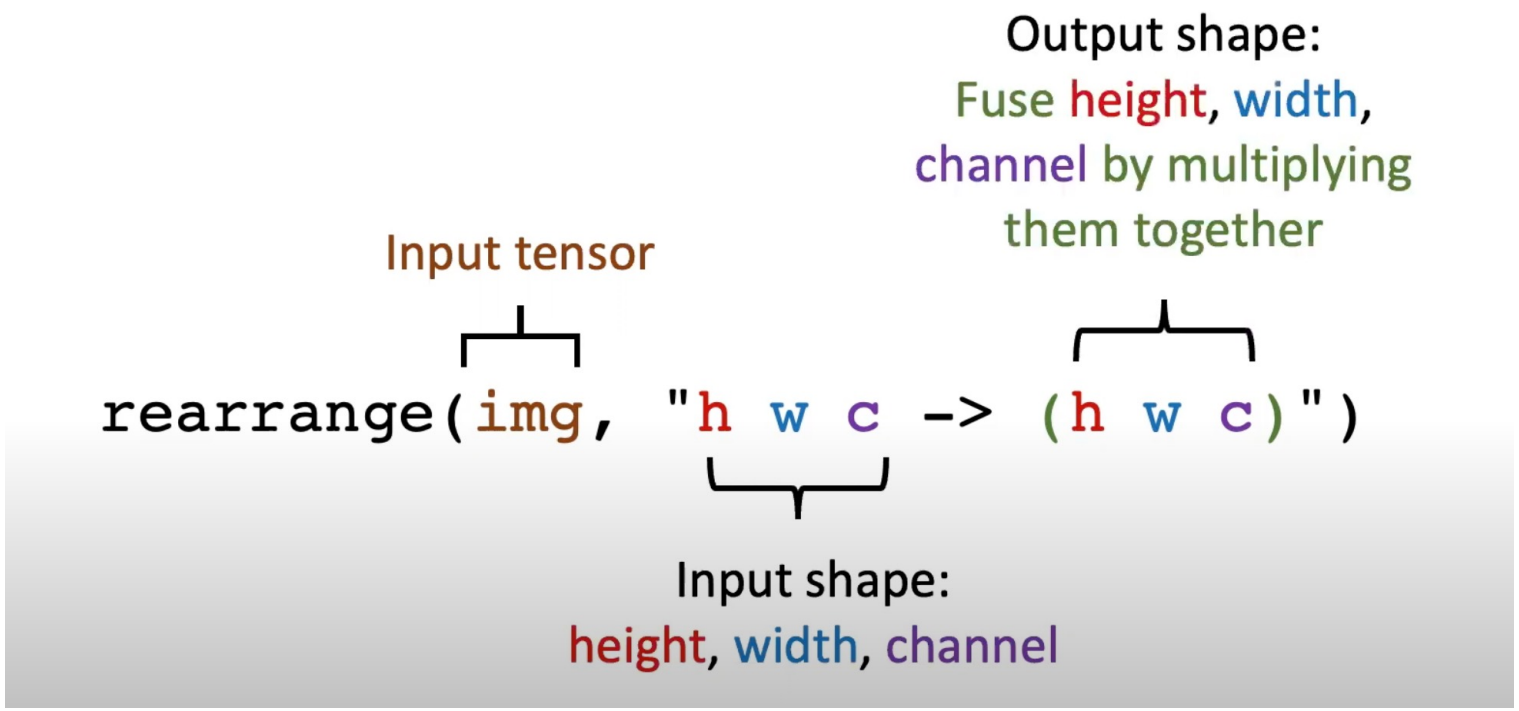

Output
Linear
Concatenate
MatMul
Softmax
Scale
MatMul
Linear
Linear
Linear
21 x 3
Z
Linear
7 x 5
Output of
Multihead
Attention

Reinventing Deep Learning Operation Via Einops

https://einops.rocks/1-einops-basics/

Flatten

```
img = image.imread("aki_dog.jpg")
```


Output shape:
Fuse height, width,
channel by multiplying
them together
Input tensor
rearrange(img, "h w c -> (h w c)")
Input shape:
height, width, channel

Load a batch of images to play with

In [2]:
```
ims = numpy.load('./resources/test_images.npy', allow_pickle=False)
# There are 6 images of shape 96x96 with 3 color channels packed into tensor
print(ims.shape, ims.dtype)
```

```
(6, 96, 96, 3) float64
```

In [3]:
```
# display the first image (whole 4d tensor can't be rendered)
ims[0]
```

Out[3]:

In [4]:
```
# second image in a batch
ims[1]
```

Out[4]:

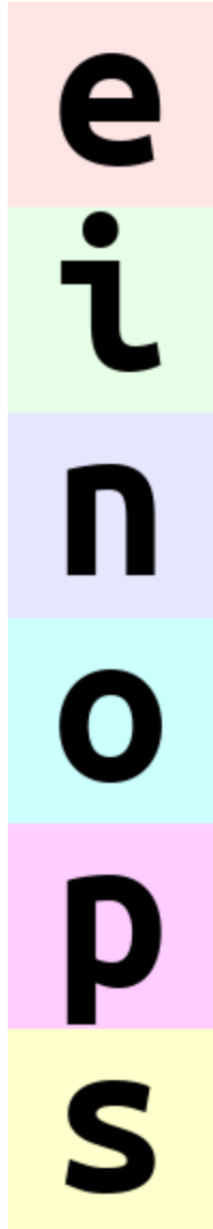
In [5]:
```
# we'll use three operations
from einops import rearrange, reduce, repeat
```

In [6]:
```
# rearrange, as its name suggests, rearranges elements
# below we swapped height and width.
# In other words, transposed first two axes (dimensions)
rearrange(ims[0], 'h w c -> w h c')
```

Out[6]:

In [7]:
```
# einops allows seamlessly composing batch and height to a new height dimension
# We just rendered all images by collapsing to 3d tensor!
rearrange(ims, 'b h w c -> (b h) w c')
```

Out[7]:



Decomposition of axis

In [11]:
```
# decomposition is the inverse process - represent an axis as a combination of new ax
# several decompositions possible, so b1=2 is to decompose 6 to b1=2 and b2=3
rearrange(ims, '(b1 b2) h w c -> b1 b2 h w c ', b1=2).shape
```

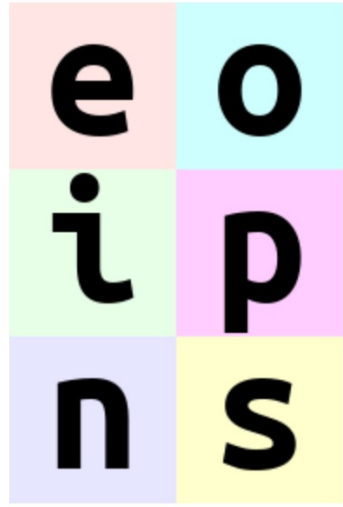
Out[11]: (2, 3, 96, 96, 3)

In [12]:
```
# finally, combine composition and decomposition:
rearrange(ims, '(b1 b2) h w c -> (b1 h) (b2 w) c ', b1=2)
```

Out[12]:

In [13]:
```
# slightly different composition: b1 is merged with width, b2 with height
# ... so letters are ordered by w then by h
rearrange(ims, '(b1 b2) h w c -> (b2 h) (b1 w) c ', b1=2)
```

Out[13]:



Meet einops.reduce

In einops-land you don't need to guess what happened

```
x.mean(-1)
```

Because you write what the operation does

```
reduce(x, 'b h w c -> b h w', 'mean')
```

if axis is not present in the output — you guessed it — axis was reduced.

In [18]:
```
# average over batch
reduce(ims, 'b h w c -> h w c', 'mean')
```

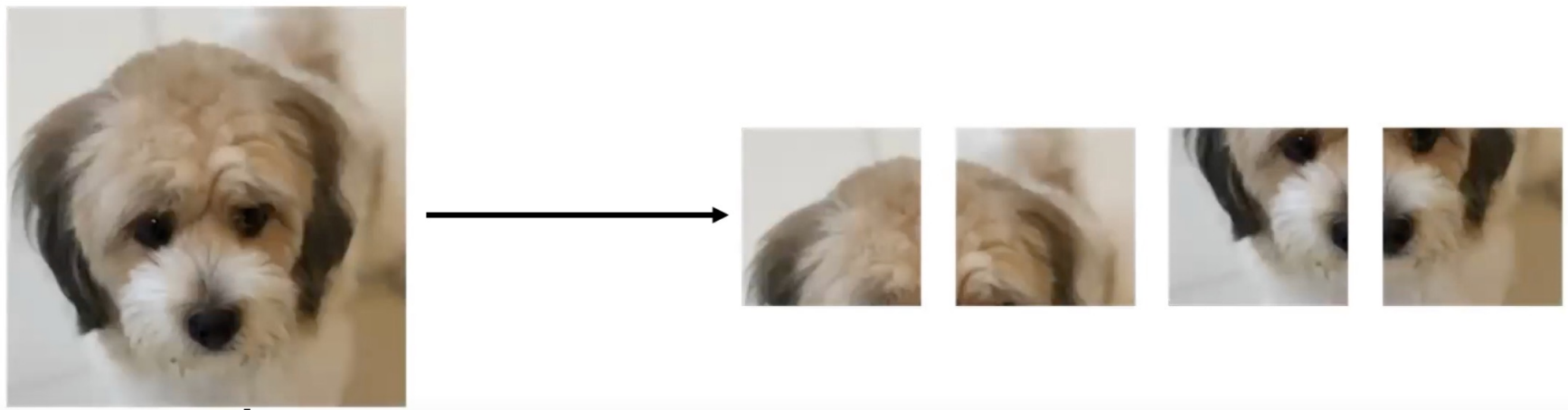
Out[18]:

In [19]:
```
# the previous is identical to familiar:
ims.mean(axis=0)
# but is so much more readable
```

Out[19]:

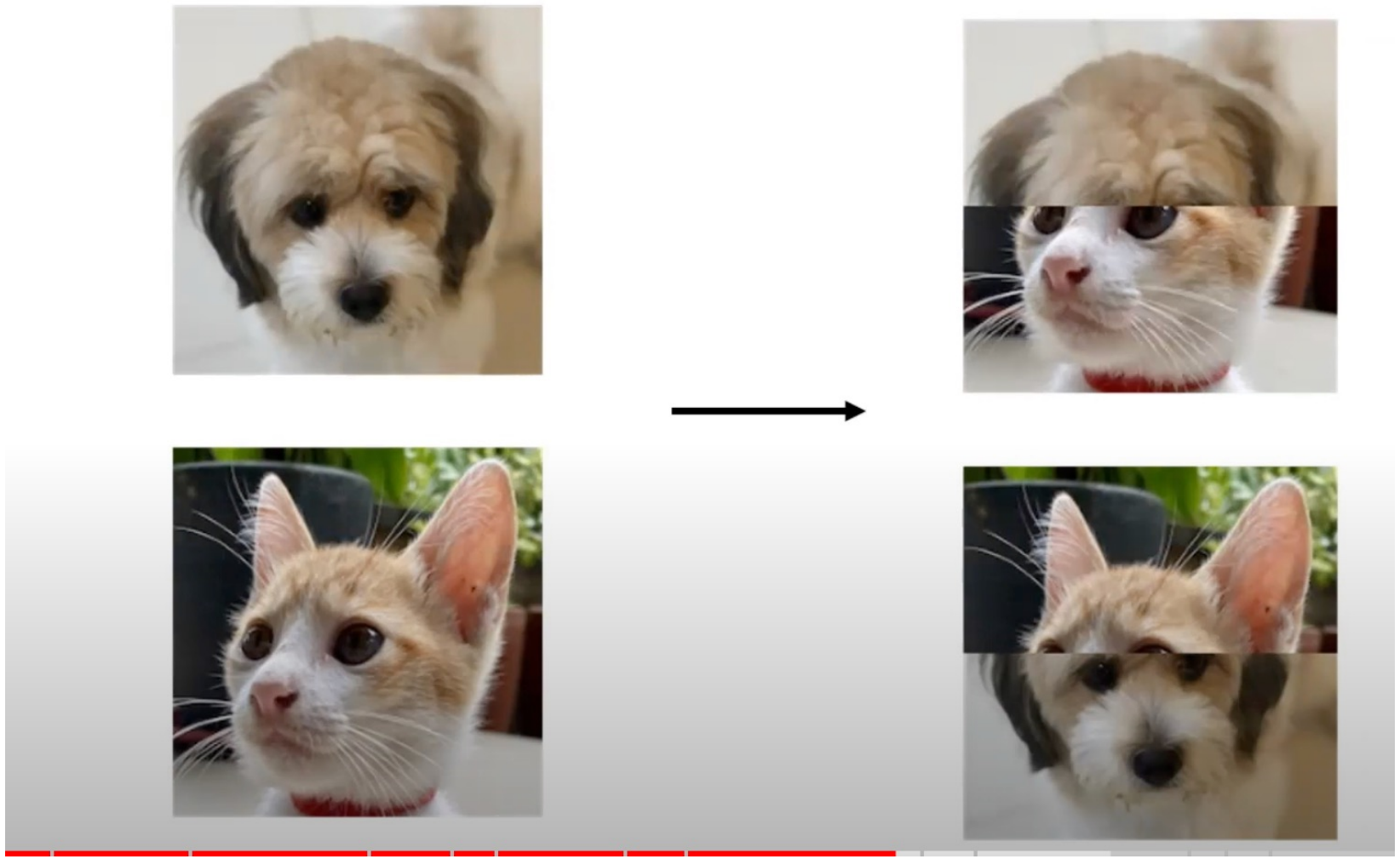
In [20]:
```
# Example of reducing of several axes
# besides mean, there are also min, max, sum, prod
reduce(ims, 'b h w c -> h w', 'min')
```

Out[20]:

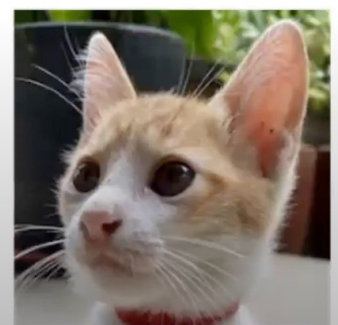


```
rearrange(img, "(p1 h) (p2 w) c -> (p1 p2) h w c", p1=2, p2=2)
```

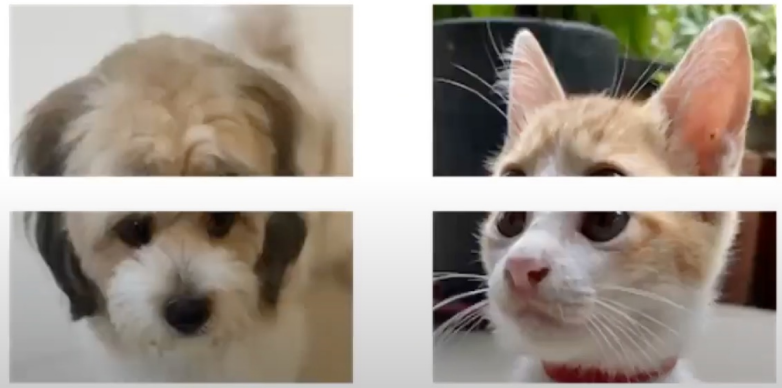
Mixing 2 images



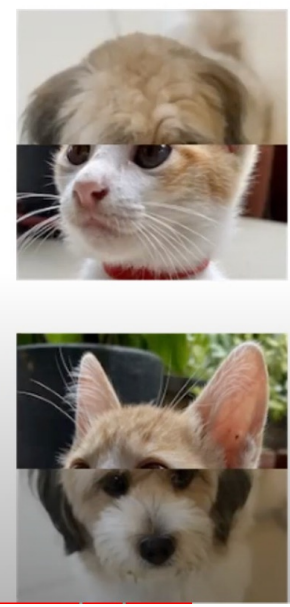
```
img1 = image.imread("aki_dog.jpg")
img2 = image.imread("wonder_cat.jpg")
imgs = np.array([img1, img2])
```



```
imgs = rearrange(imgs, "b (k h) w c -> k b h w c", k =2)
```

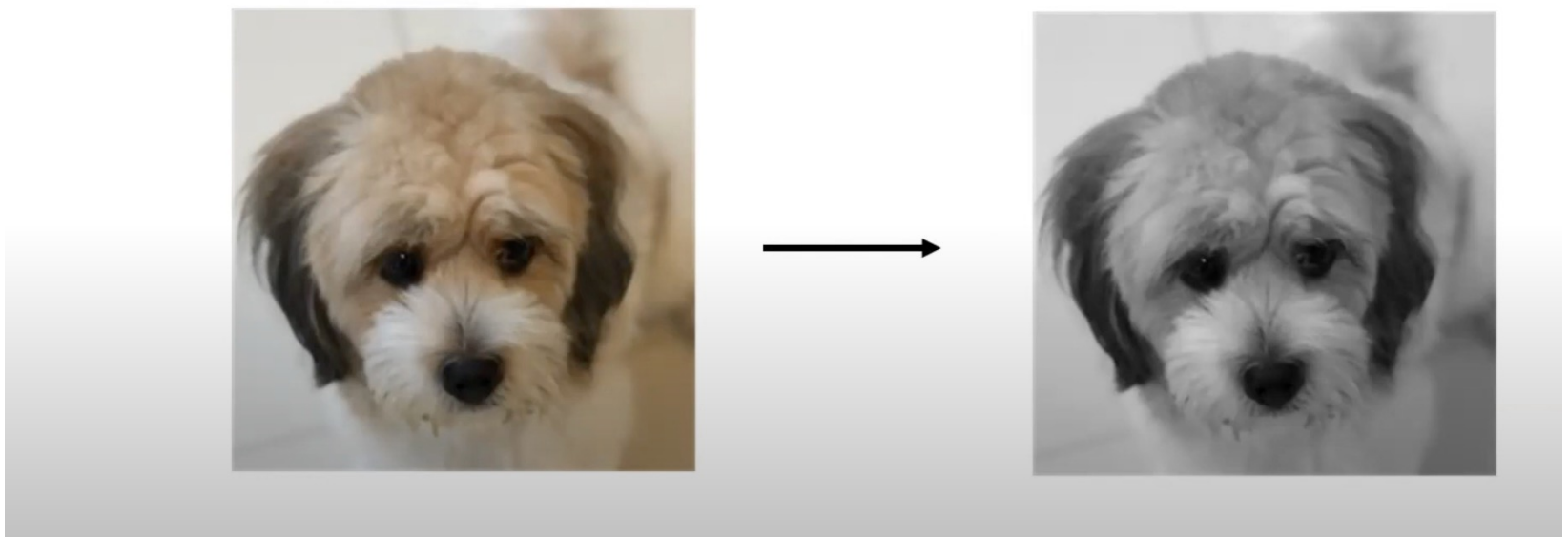


```
imgs = rearrange(imgs, "i j h w c -> j (i h) w c")
```



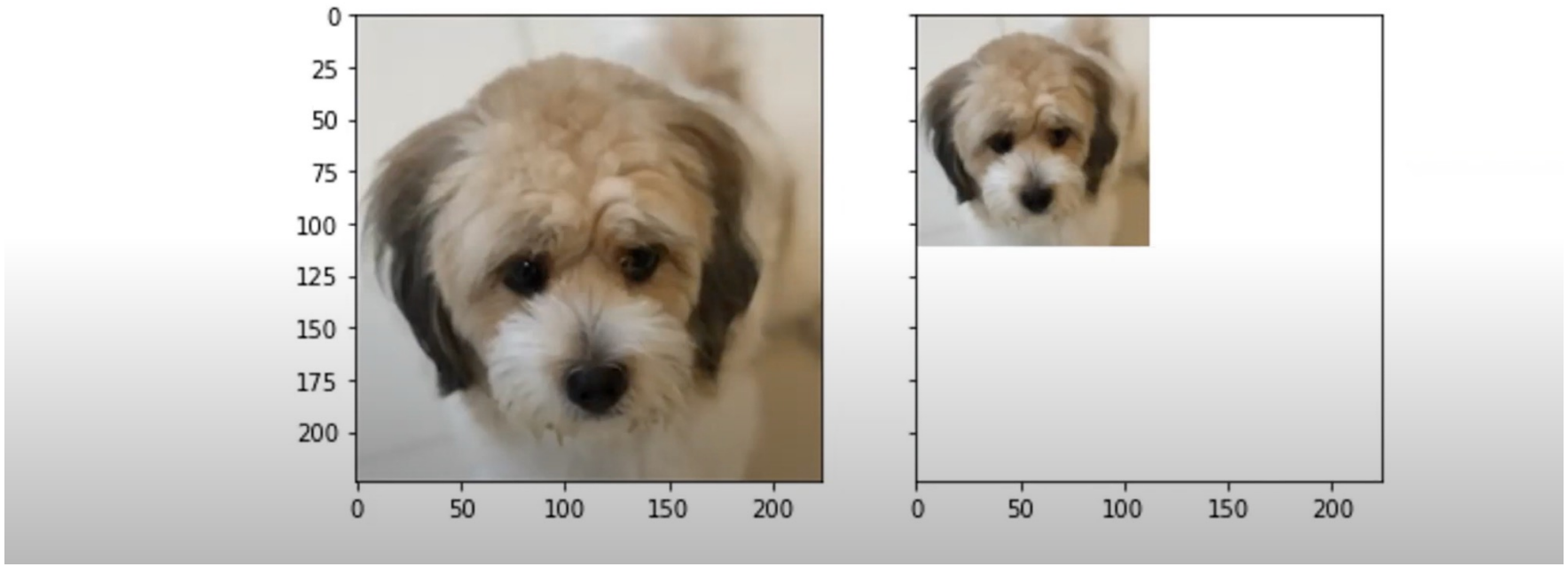
RGB to Grayscale

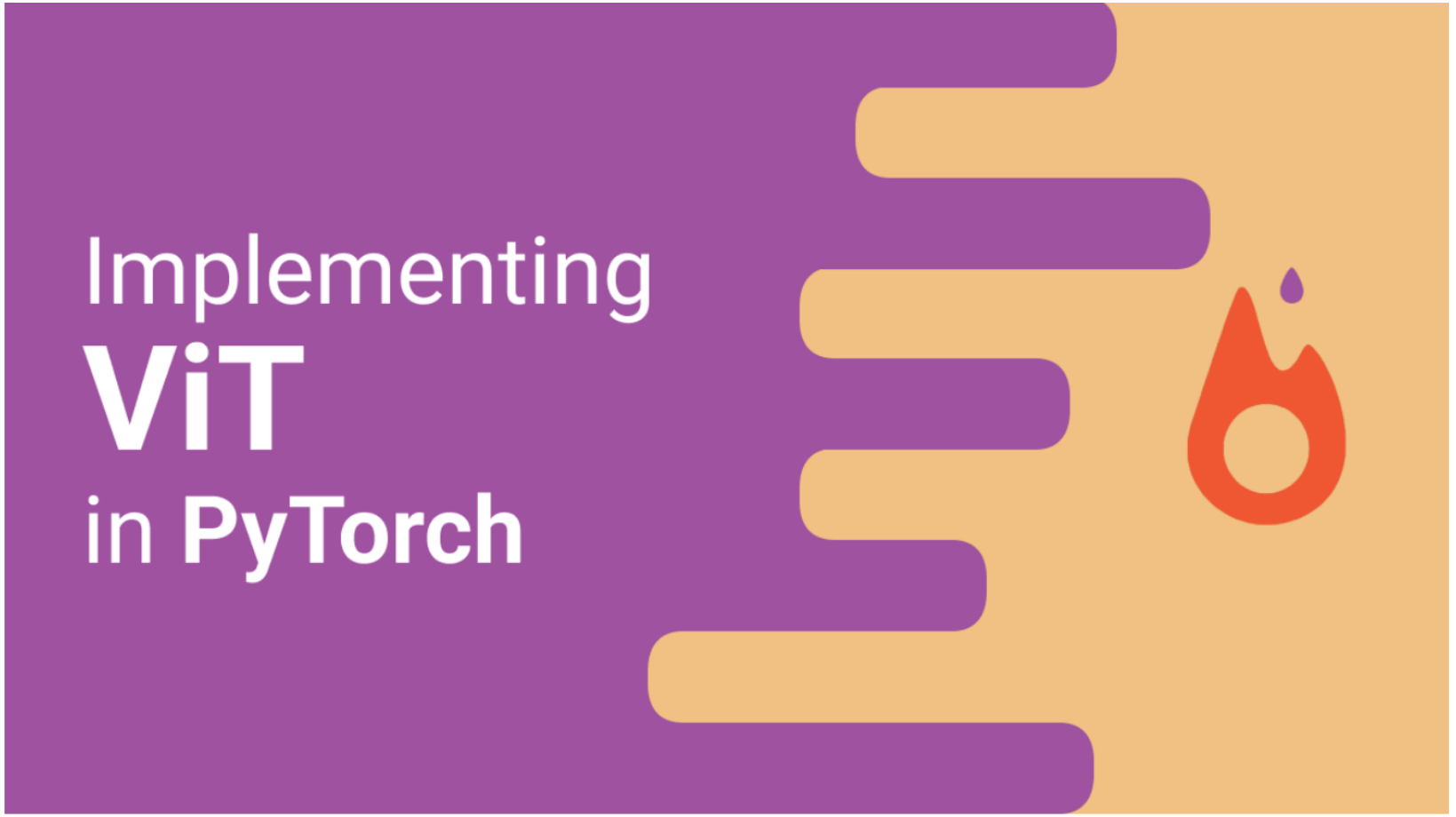
```
img = reduce(img, "h w c -> h w", 'mean')
```



Downsize

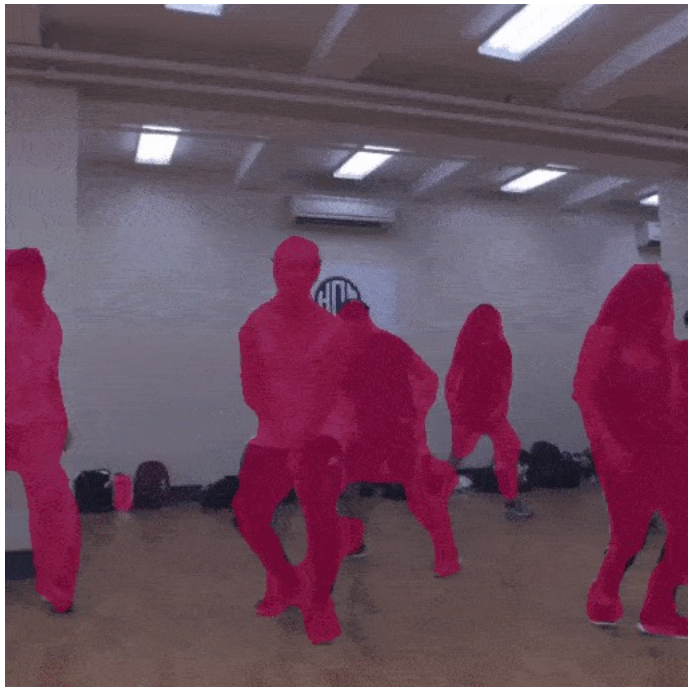
```
reduce(img, "(h 2) (w 2) c -> h w c", 'mean')
```

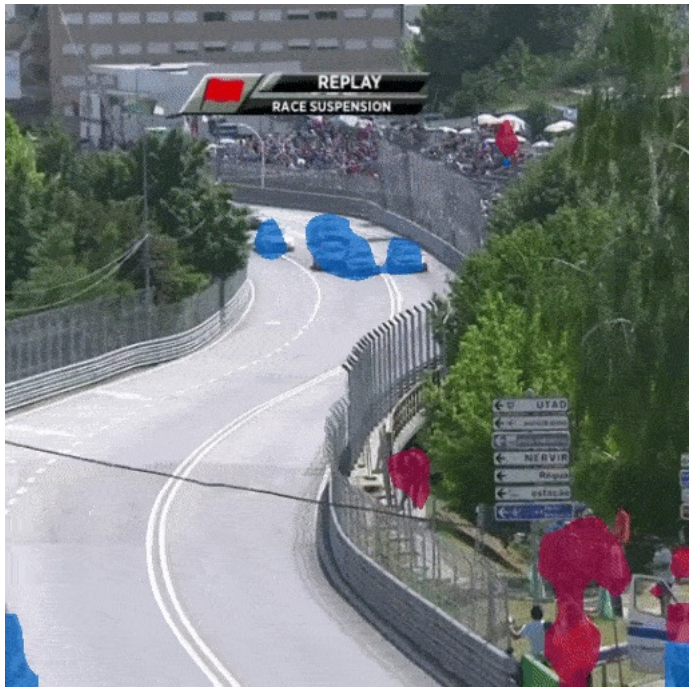


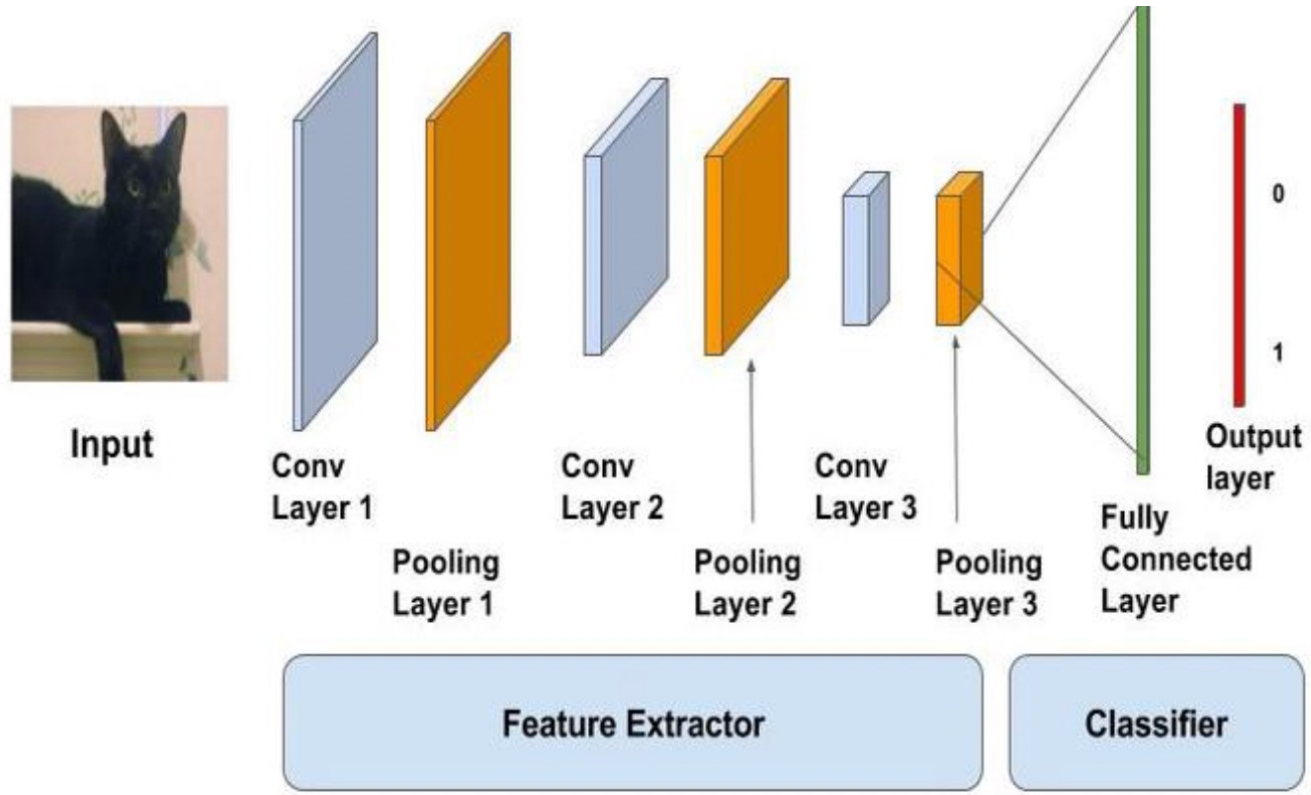


https://medium.com/towards-data-science/implementing-visualttransformer-in-pytorch-184f9f16f632

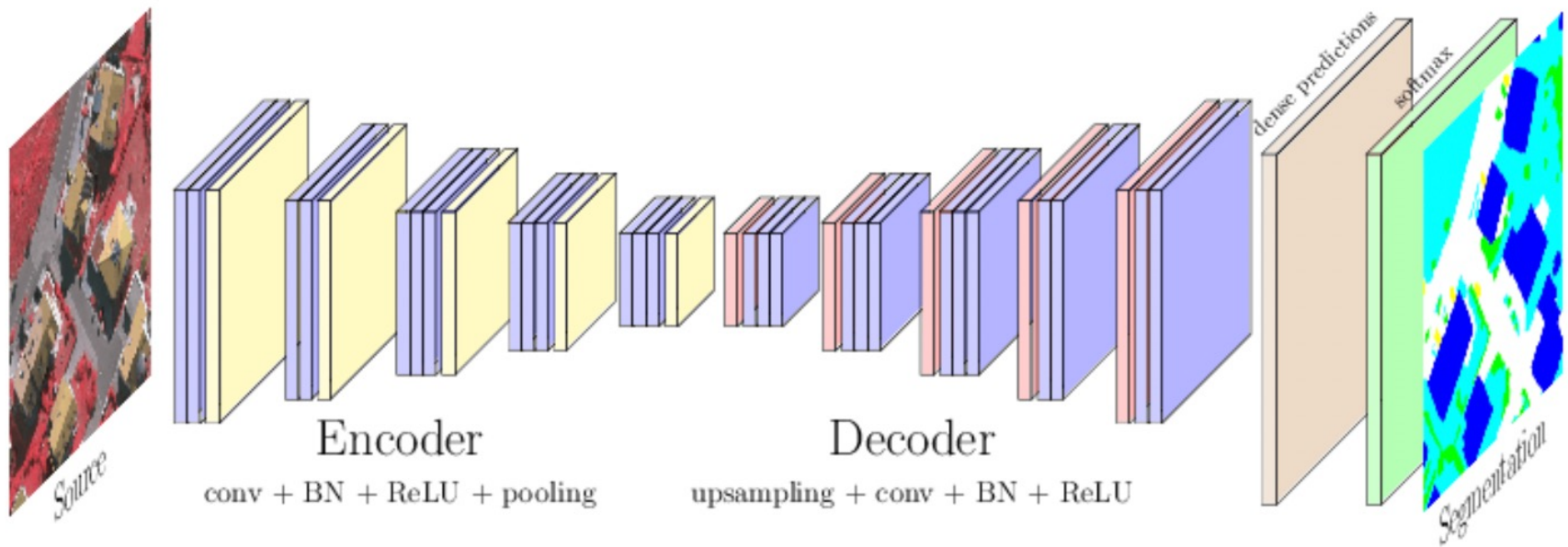
image segmentation

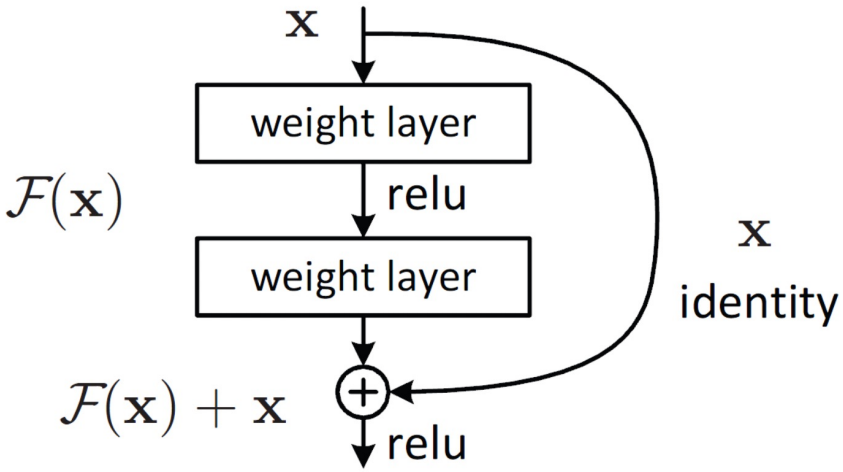


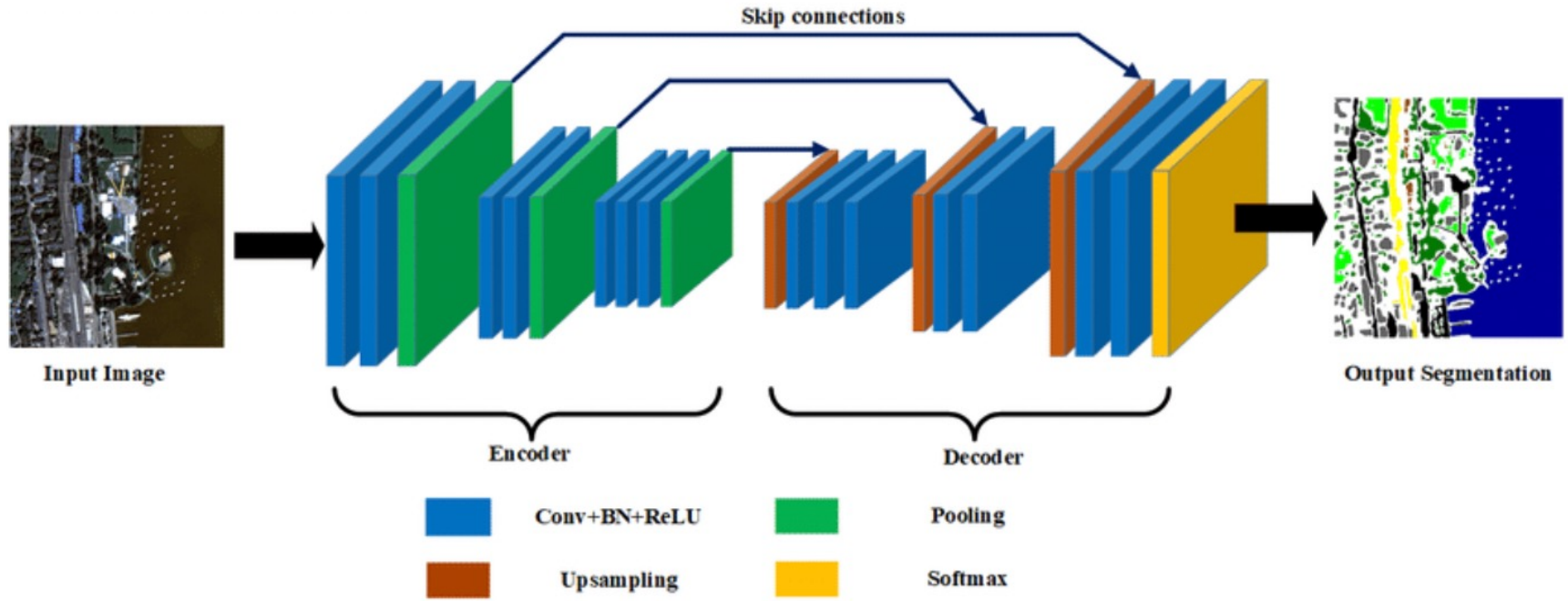

REPLAY
RACE SUSPENSION
UTAD
NERVIR



https://medium.com/analytics-vidhya/introduction-to-semantic-image-segmentation-856cda5e5de8

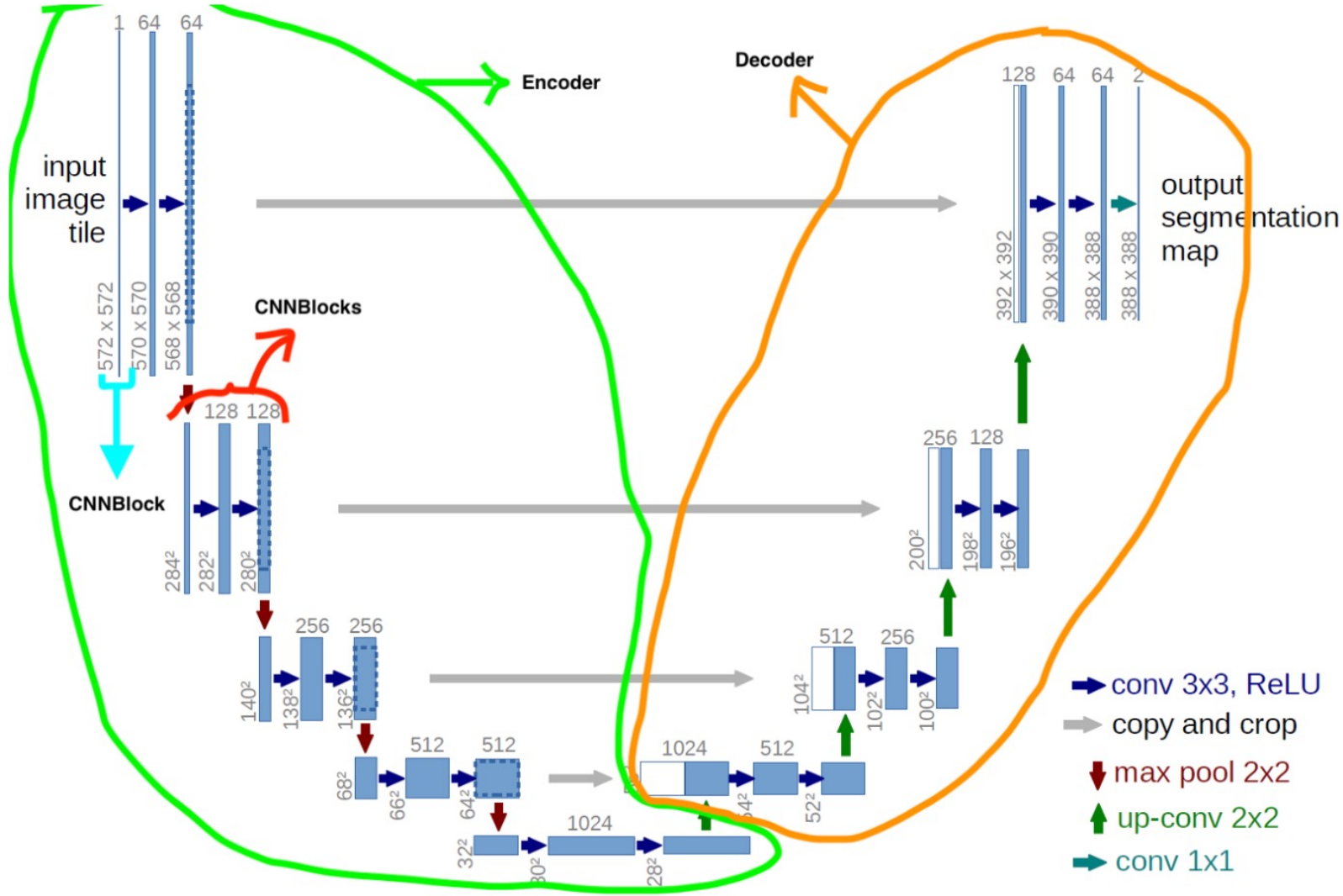

Source
Encoder
conv + BN + ReLU + pooling
Decoder
upsampling + conv + BN + ReLU
dense predictions
softmax
Segmentation


x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

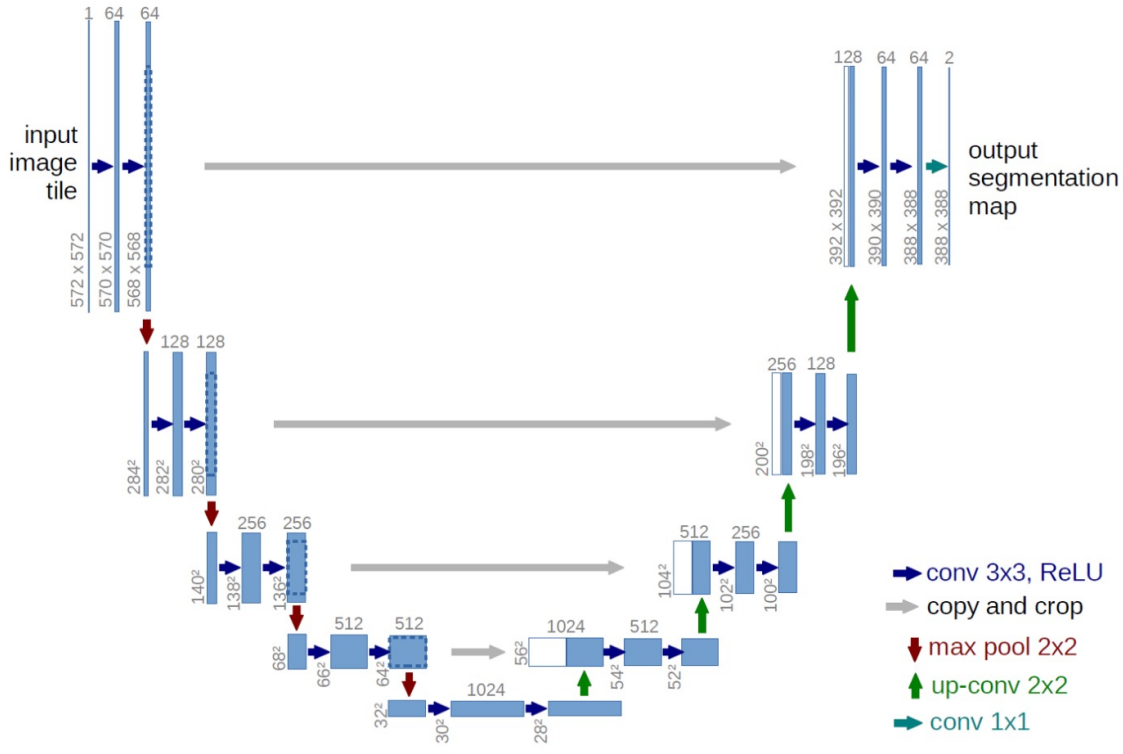

Skip connections
Input Image
Output Segmentation
Encoder
Decoder
Conv+BN+ReLU
Pooling
Upsampling
Softmax

https://medium.com/mlearning-ai/semantic-segmentation-with-pytorch-u-net-from-scratch-502d6565910a

https://medium.com/@Chinmay_Paranjape/satellite-imagery-segmentation-using-u-net-4ec7f265ddbe


Encoder
Decoder
input image tile
CNNBlocks
CNNBlock
output segmentation map
1 64 64
572 x 572
570 x 570
568 x 568
128 128
284²
282²
280²
256 256
140²
138²
136²
512 512
68²
66²
64²
1024
32²
30²
28²
512
104²
102²
100²
256
200²
198²
196²
256 128
128 64 64 2
392 x 392
390 x 390
388 x 388
388 x 388
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

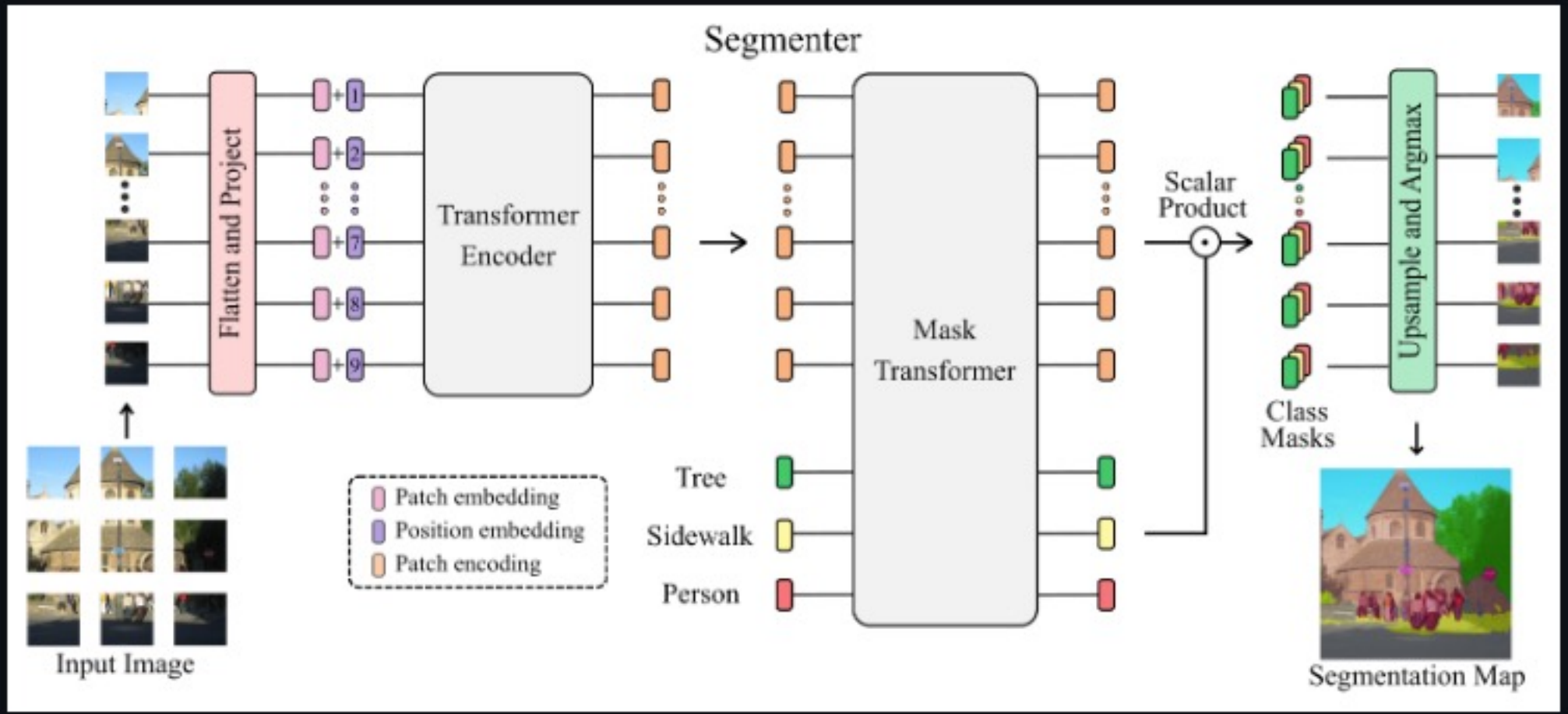
U-net Architecture

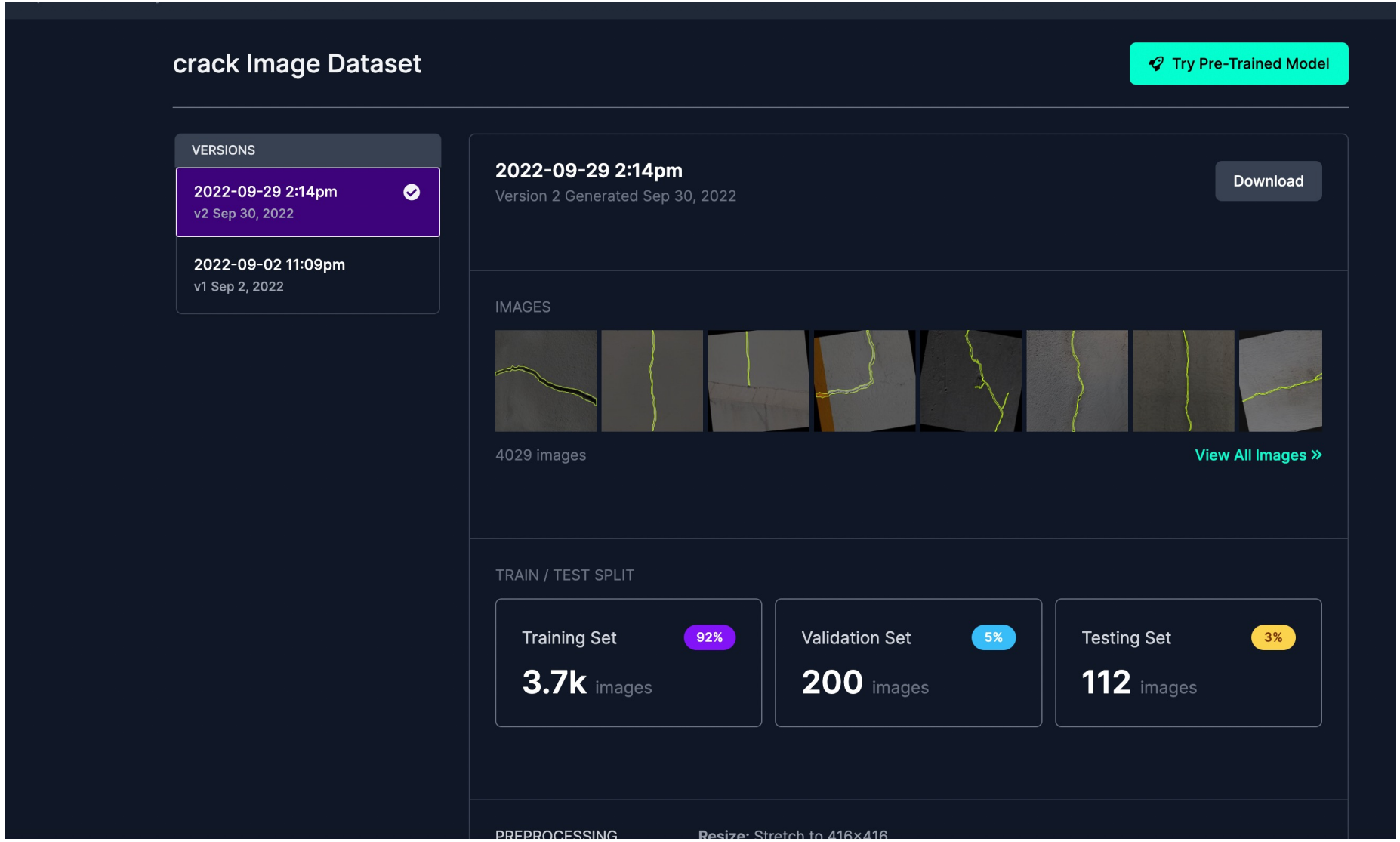



```python
class CNNBlock(nn.Module):
    def __init__(self,
                 in_channels,
                 out_channels,
                 kernel_size=3,
                 stride=1,
                 padding=0):
        super(CNNBlock, self).__init__()

        self.seq_block = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size, stride, padding, bias=False
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        )

    def forward(self, x):
        x = self.seq_block(x)
        return x
```

```python
class CNNBlocks(nn.Module):
    """
    Parameters:
    n_conv (int): creates a block of n_conv convolutions
    in_channels (int): number of in_channels of the first block's convolution
    out_channels (int): number of out_channels of the first block's convolution
    expand (bool) : if True after the first convolution of a blocl the number of channels
    """
    def __init__(self,
                 n_conv,
                 in_channels,
                 out_channels,
                 padding):
        super(CNNBlocks, self).__init__()
        self.layers = nn.ModuleList()
        for i in range(n_conv):

            self.layers.append(CNNBlock(in_channels, out_channels, padding=padding))
            # after each convolution we set (next) in_channel to (previous) out_channels
            in_channels = out_channels

    def forward(self, x):
        for layer in self.layers:
            x = layer(x)
        return x
```

Segmenter: Transformer for Semantic Segmentation





https://universe.roboflow.com/chonnam-national-university/crack-bphdr/dataset/2