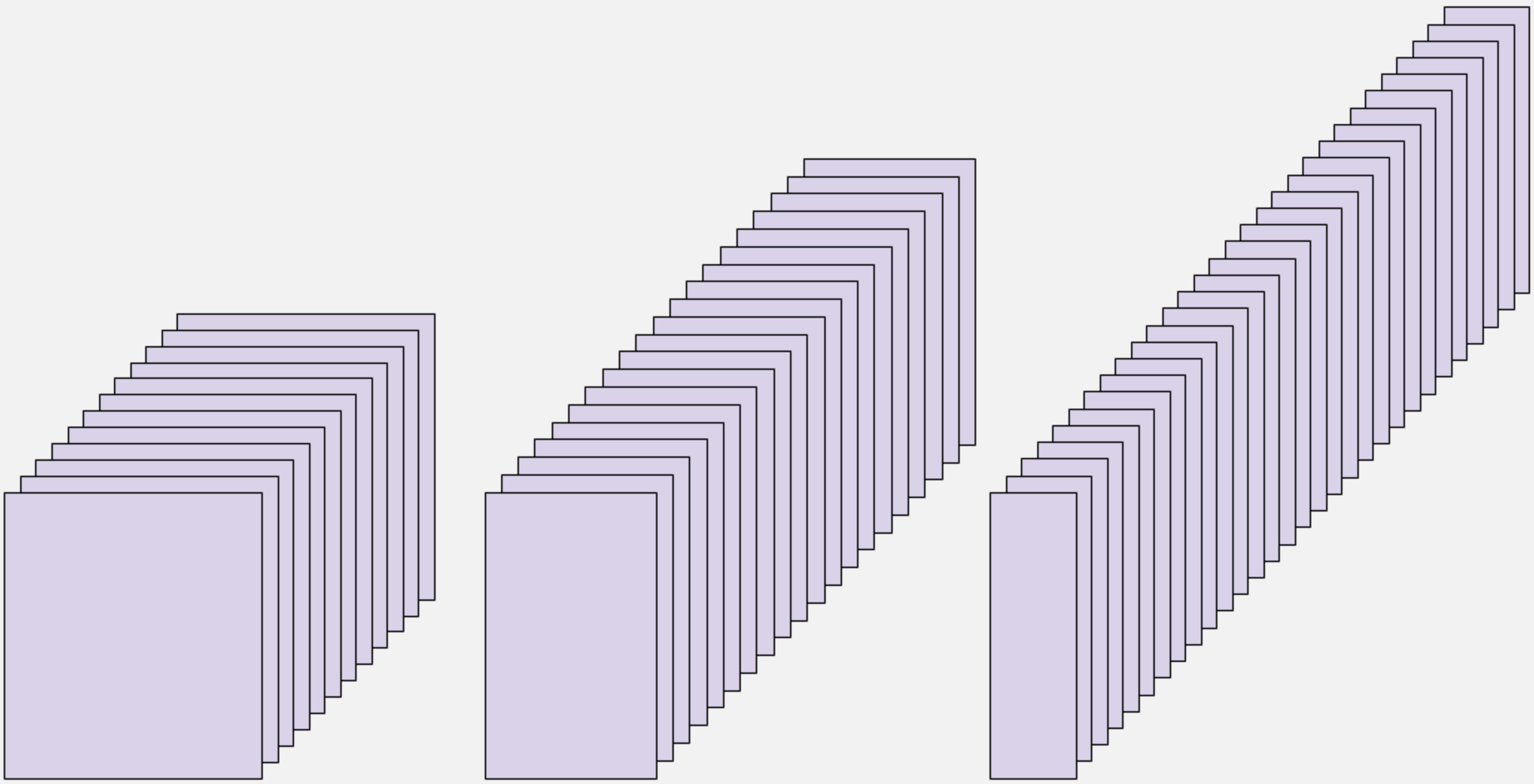
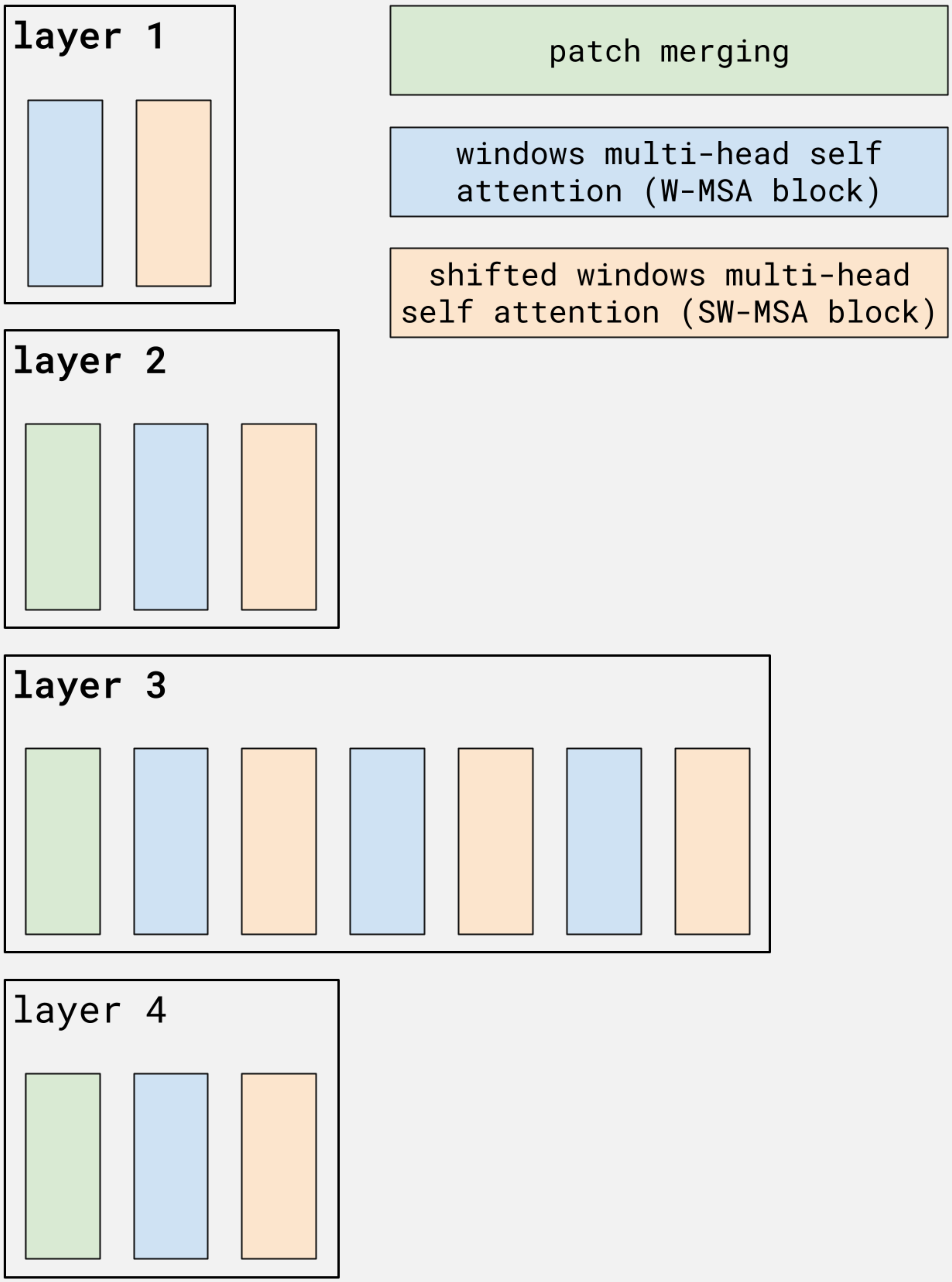
SWIN Transformer

A Walkthrough



youtube.com/@mashaan14

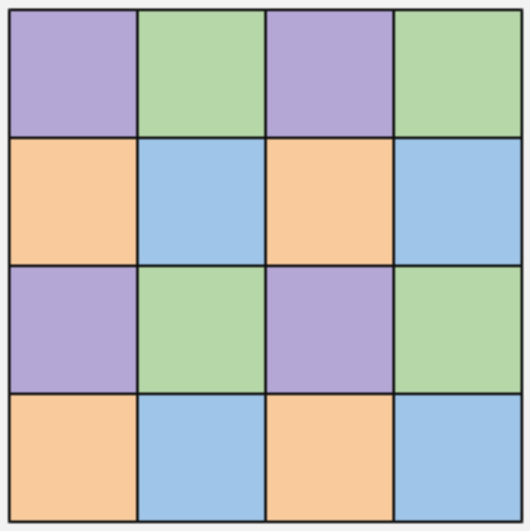
The Architecture

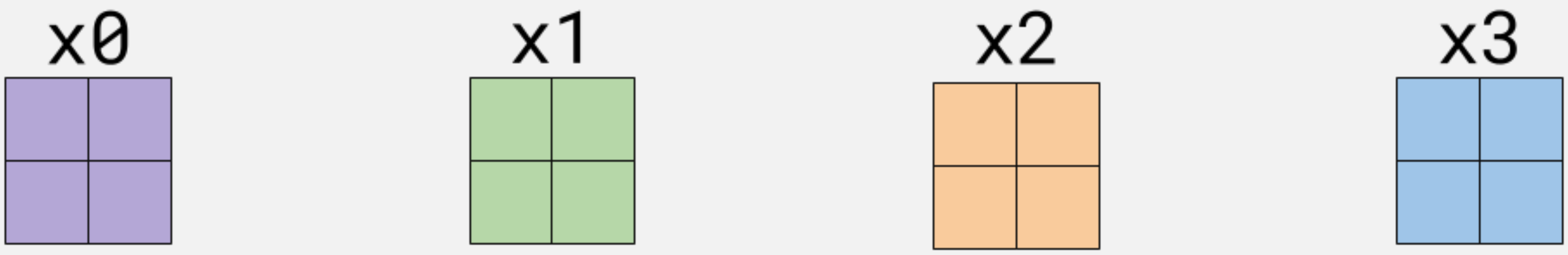


Patch Merging

```
x0 = x[:, 0::2, 0::2, :]   # H: even,  W: even
x1 = x[:, 1::2, 0::2, :]   # H: odd,   W: even
x2 = x[:, 0::2, 1::2, :]   # H: even,  W: odd
x3 = x[:, 1::2, 1::2, :]   # H: odd,   W: odd
```

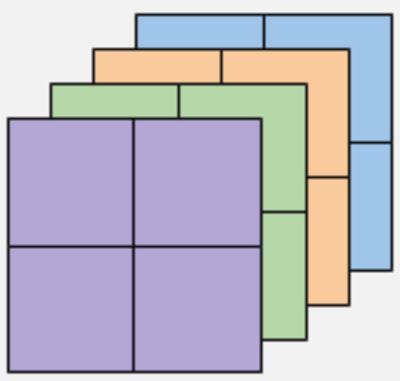
x





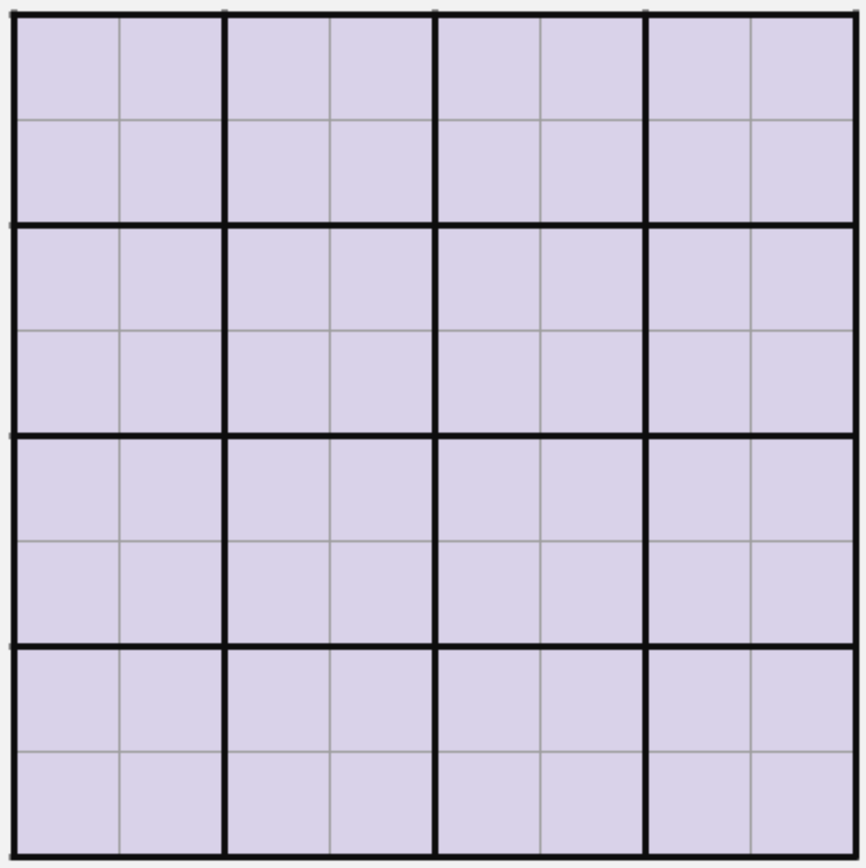

```
x = torch.cat([x0, x1, x2, x3], -1)
```

x

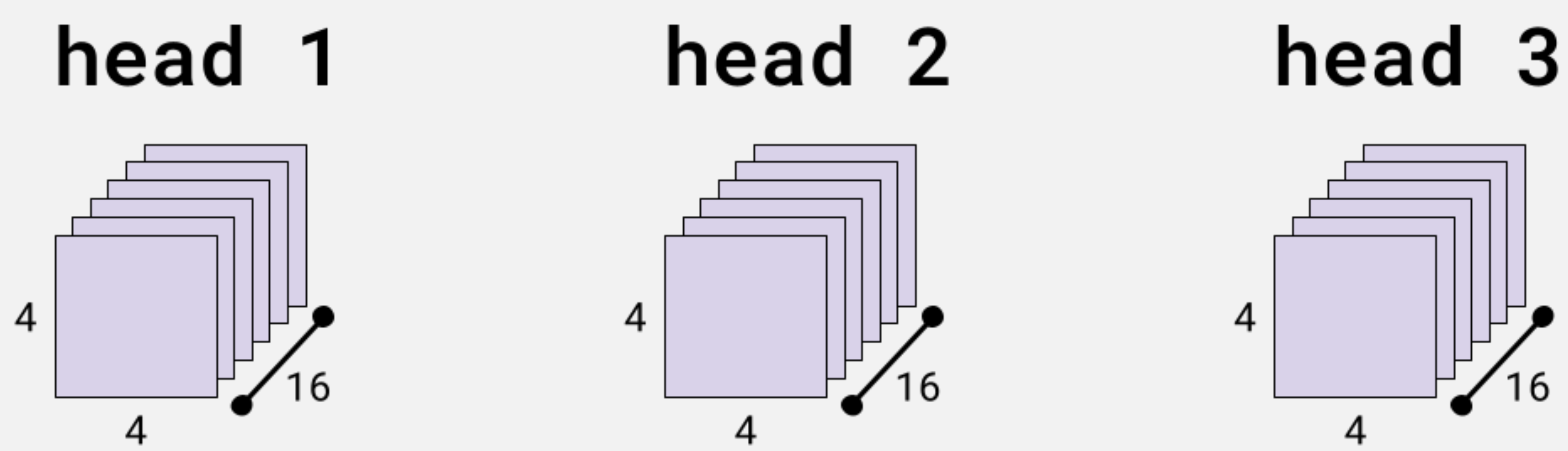


W-MSA Block

8 patches / 2 window size = 4 windows



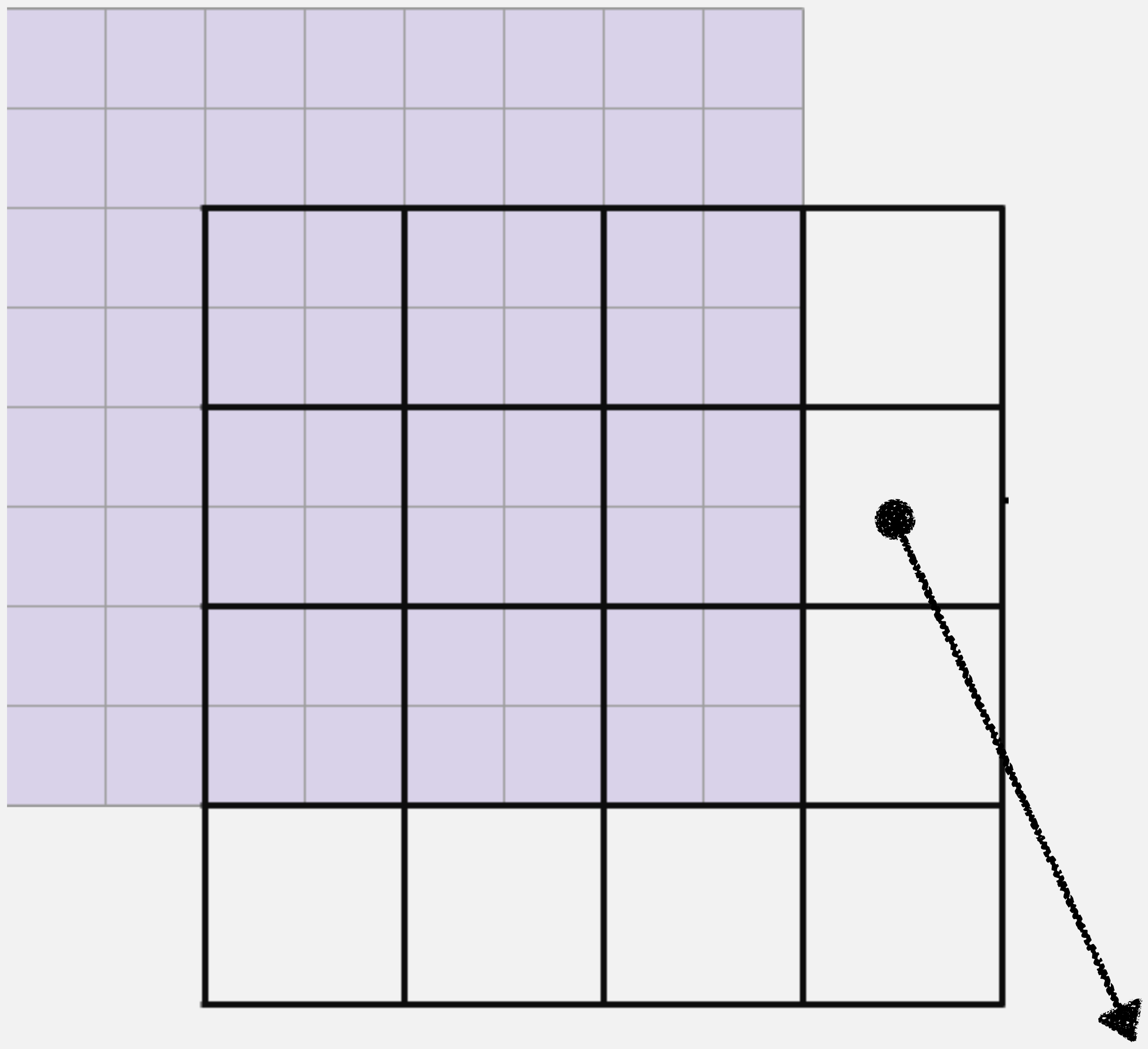
An attention head outputs a tensor of shape:
(number of patches per window) x
(number of patches per window) x
(number of windows)



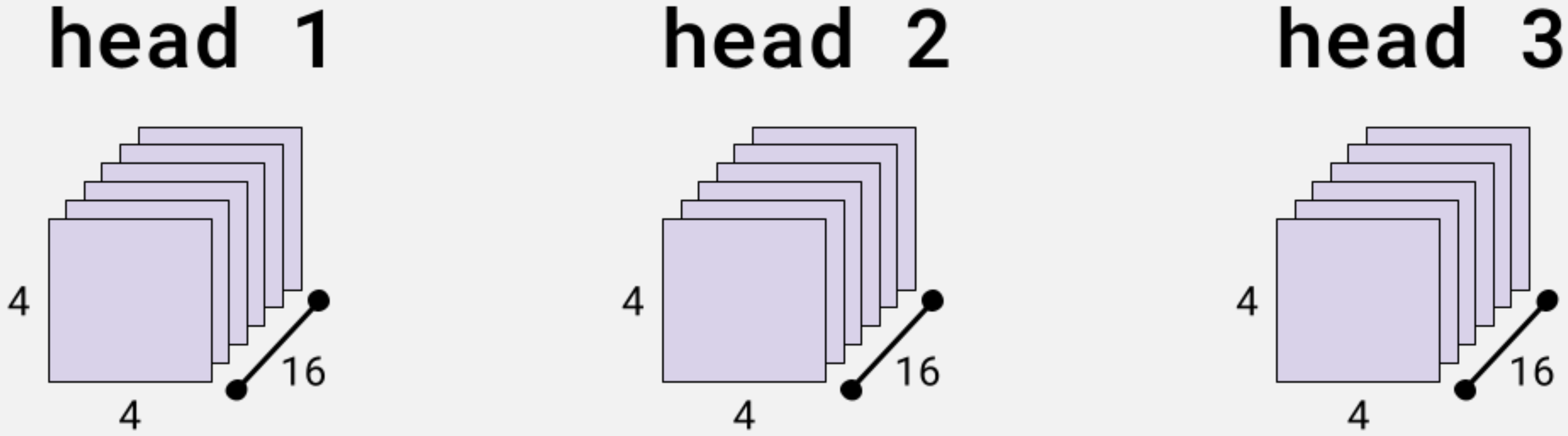
SW-MSA Block

8 patches / 2 window size = 4 windows

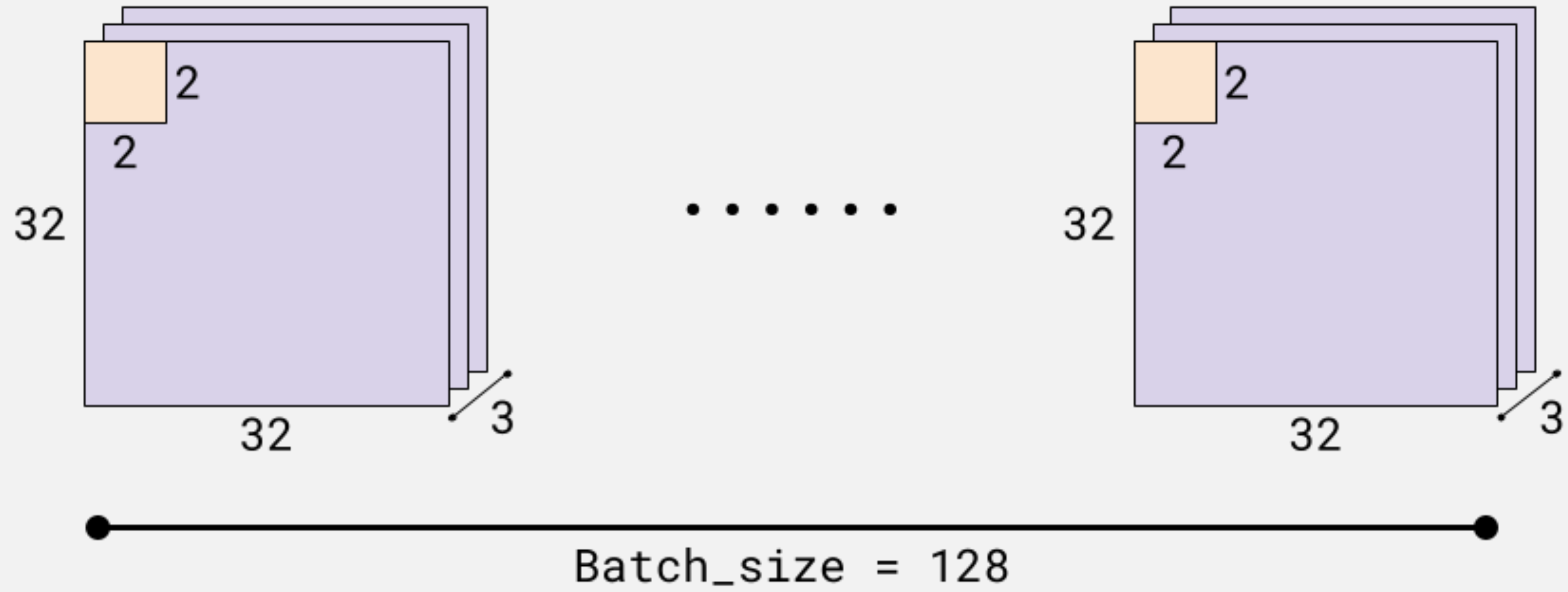
We shift the original patch values, then split the patches into windows at their new positions to perform attention.



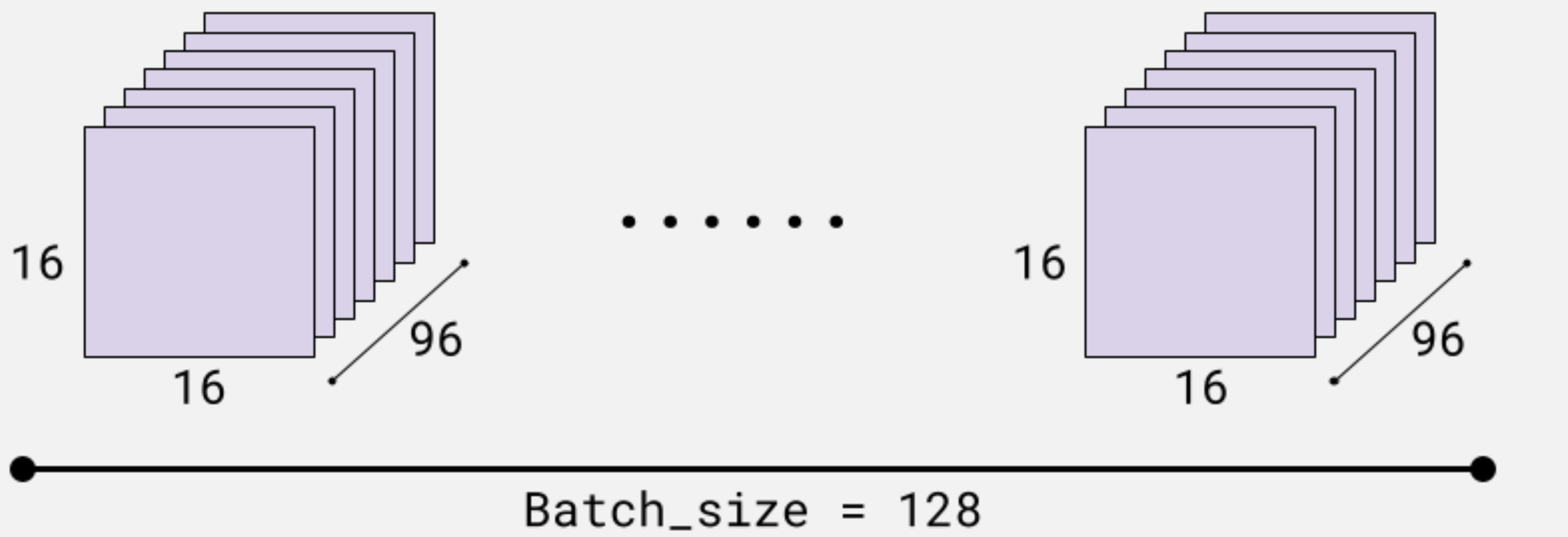
An attention mask removes the attention scores of these patch values, as they are not part of the original image.



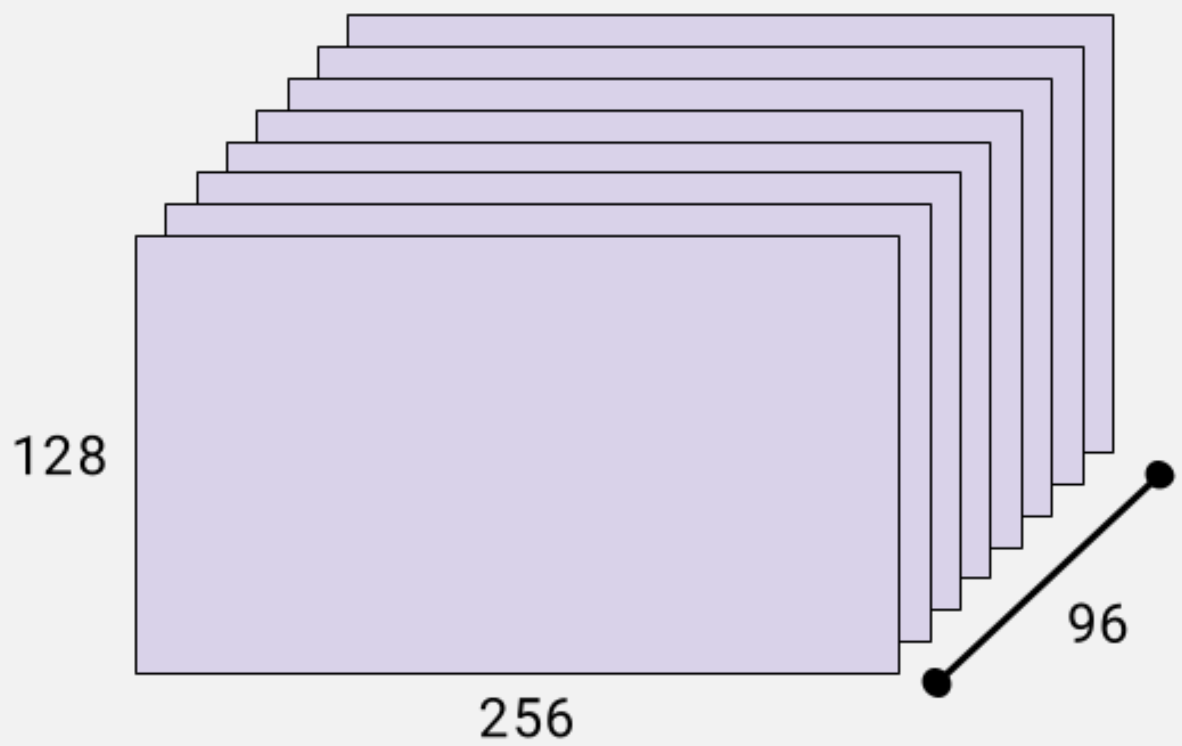
Patch Embedding




```
nn.Conv2d(in_chans, embed_dim,kernel_size=patch_size,
          stride=patch_size)
```



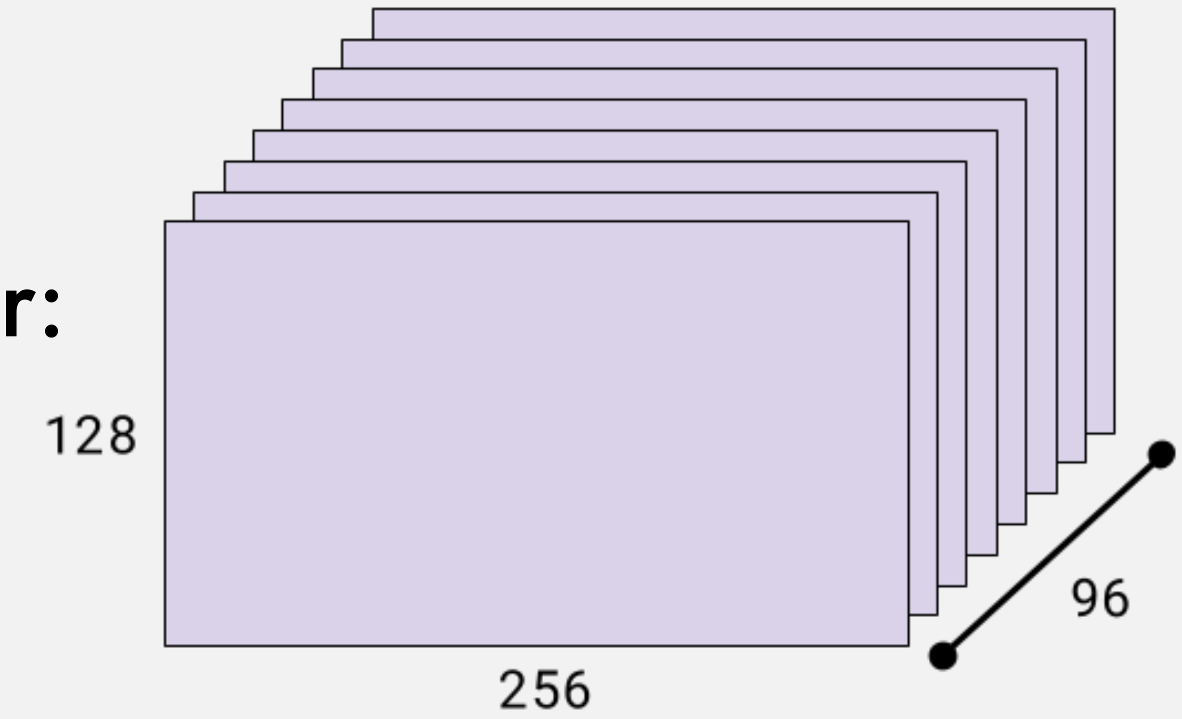

```
x.flatten(1,2)
```



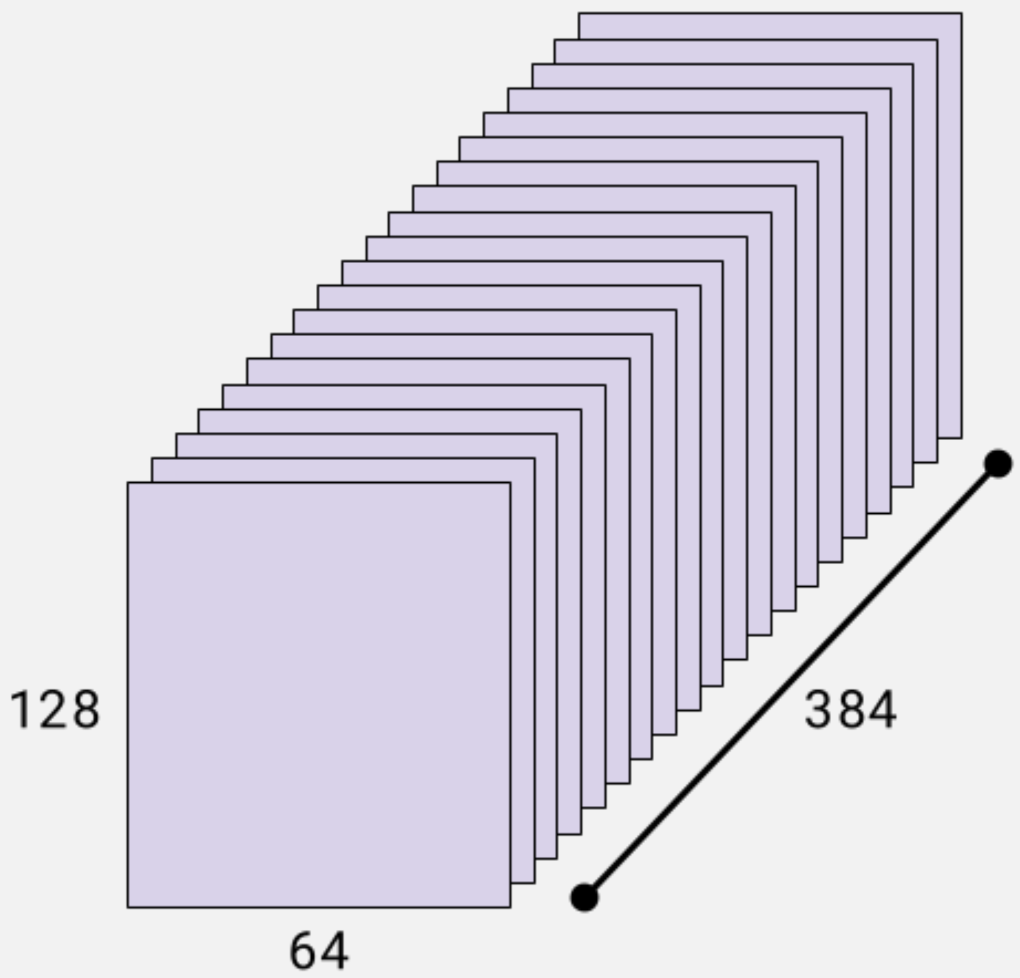
youtube.com/@mashaan14

1st layer

This is the input to the 1st layer:



Displayed below is the output from the 1st layer:

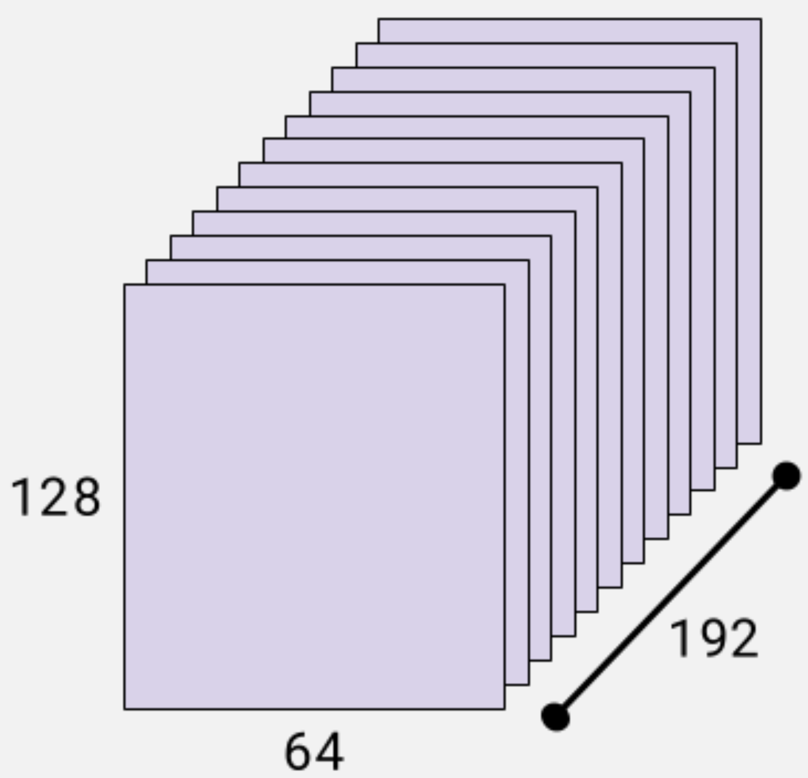


Why is the HxW dimension 64?
By dividing the 32x32 image into 2x2 patches, we get 16x16 patches. Downsampling by a factor of two further reduces this to 8x8 patches, resulting in 64 patches.

Why is the C dimension 384?
Patch merging stacks four patches along the channel (C) dimension, increasing the channel count to 4C, 4x96=384.

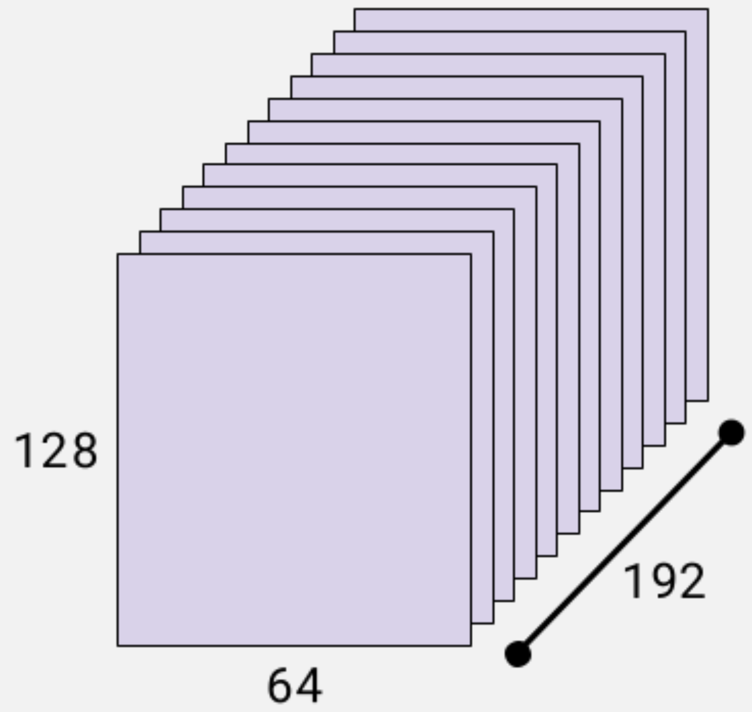
```
nn.Linear(384,192)
```

What are the inputs to the nn.Linear layer?
At the end of each layer, nn.Linear reduces the channel dimension from 4C to 2C.

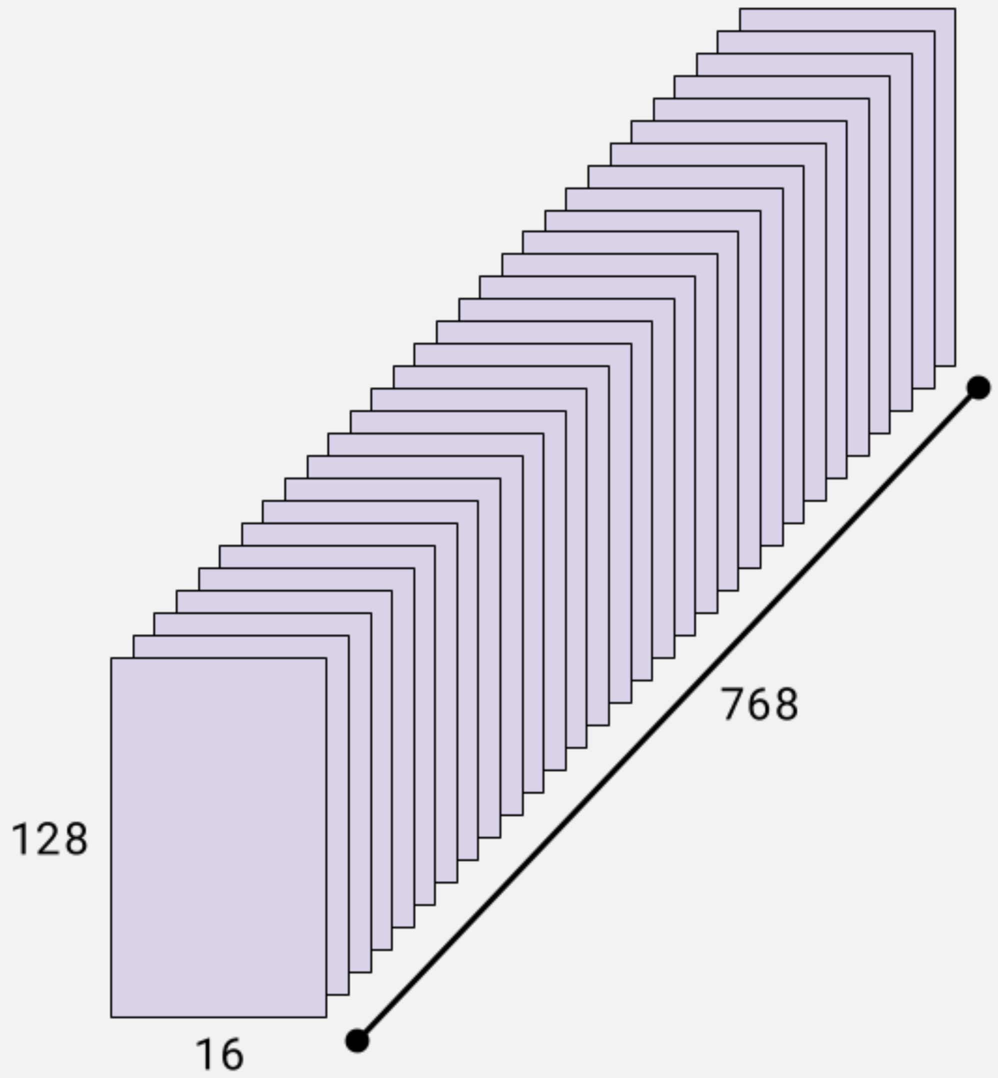


2nd layer

This is the input to the 2nd layer:



Displayed below is the output from the 2nd layer:

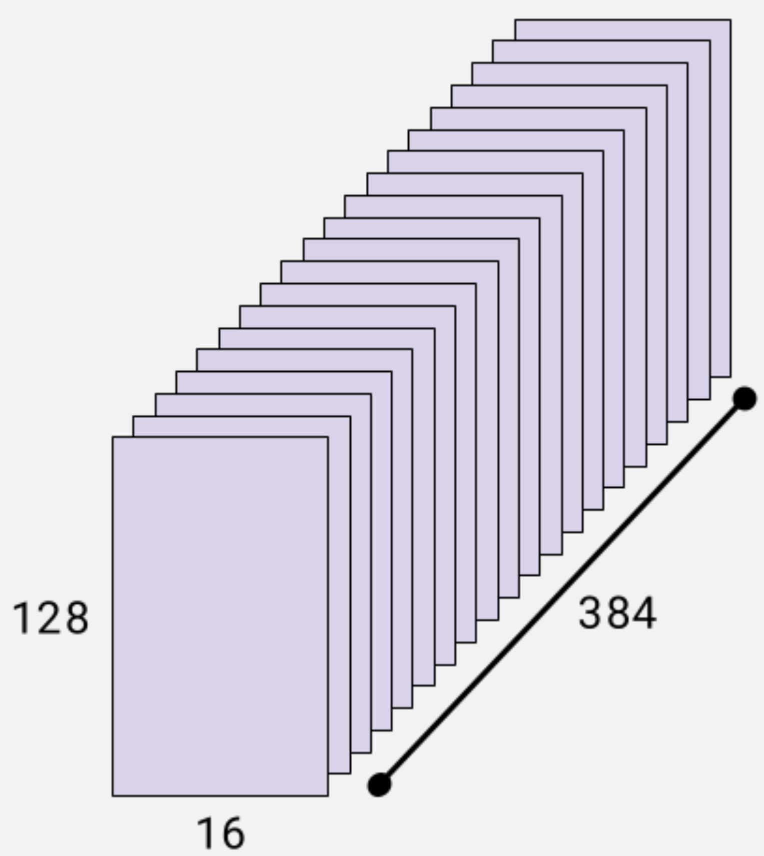


Why is the HxW dimension 16?
We started the 2nd layer with 8x8 patches. Downsampling by a factor of two further reduces this to 4x4 patches, resulting in 16 patches.

Why is the C dimension 768?
Patch merging stacks four patches along the channel (C) dimension, increasing the channel count to 4C, 4x192=768.

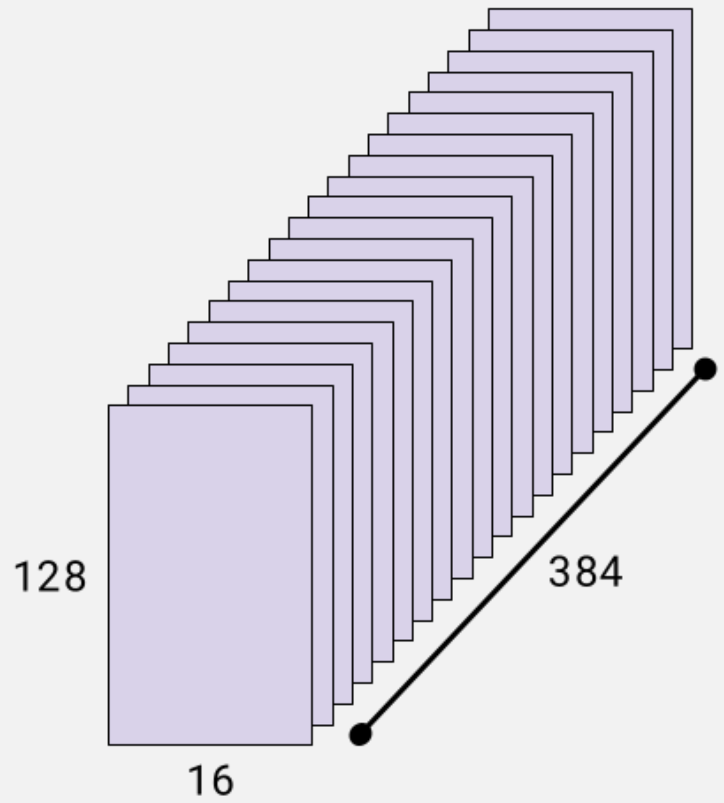
```
nn.Linear(768,384)
```

What are the inputs to the nn.Linear layer?
At the end of each layer, nn.Linear reduces the channel dimension from 4C to 2C.

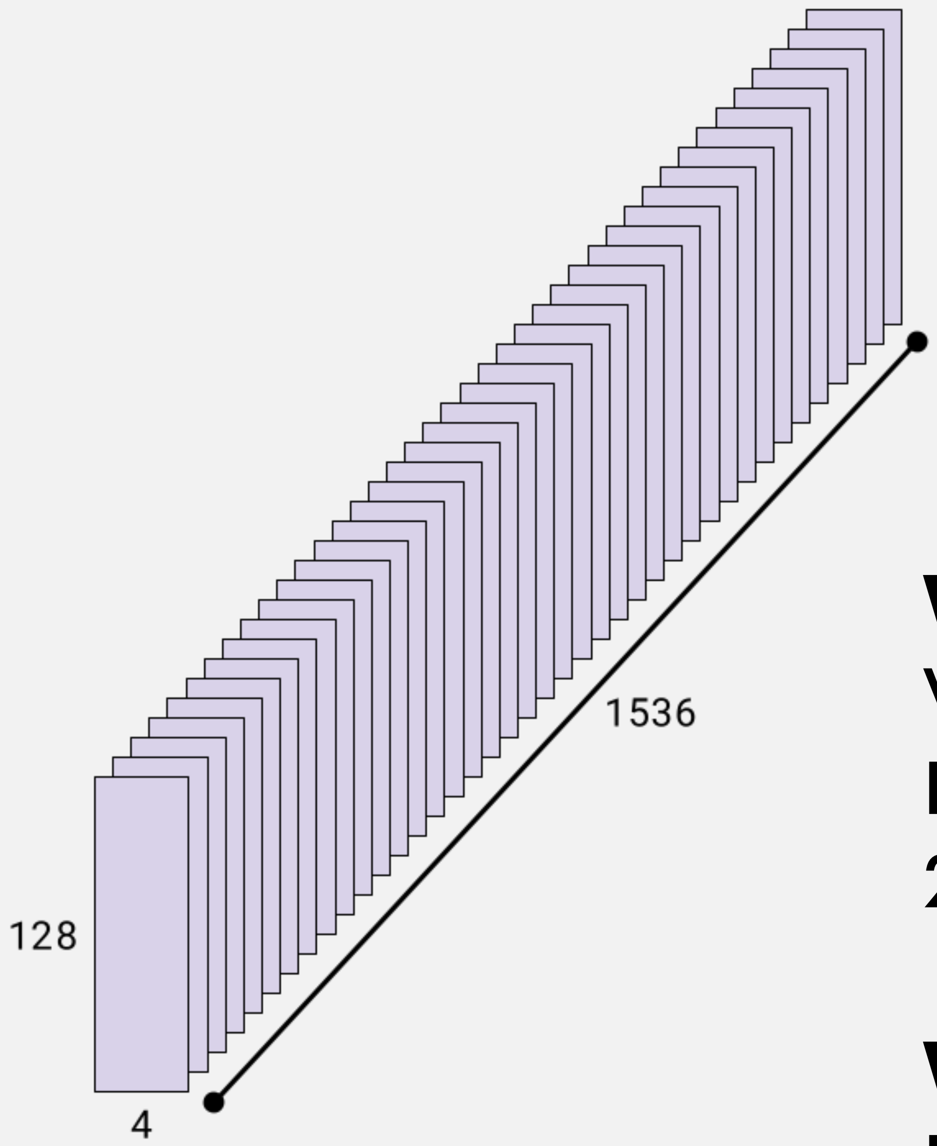


3rd layer

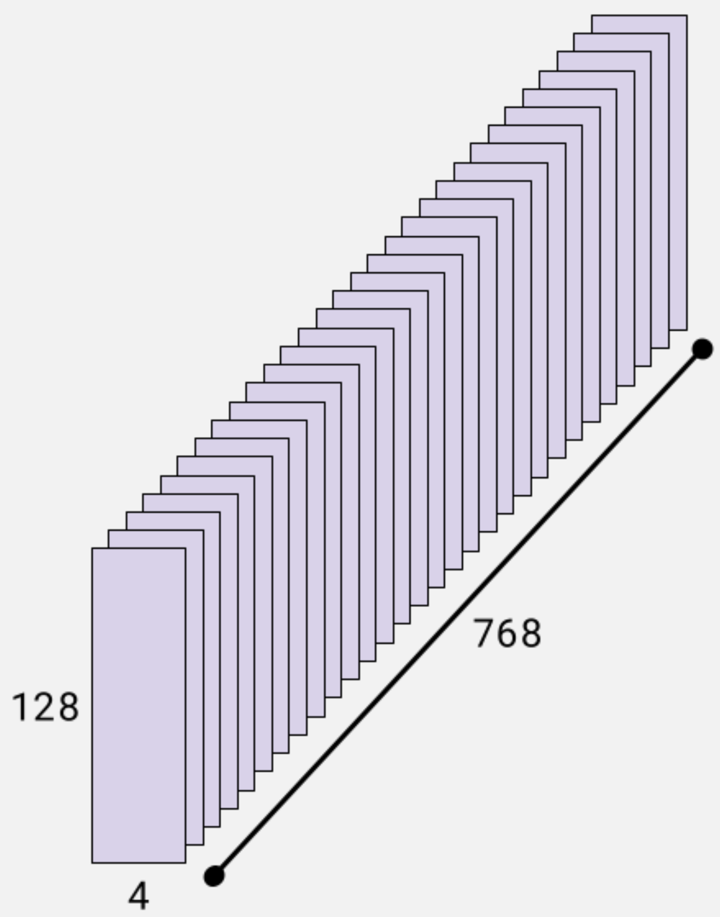
This is the input to the 3rd layer:



Displayed below is the output from the 3rd layer:




```
nn.Linear(1536,768)
```



Why is the HxW dimension 4?
We started the 3rd layer with 4x4 patches. Downsampling by a factor of two further reduces this to 2x2 patches, resulting in 4 patches.

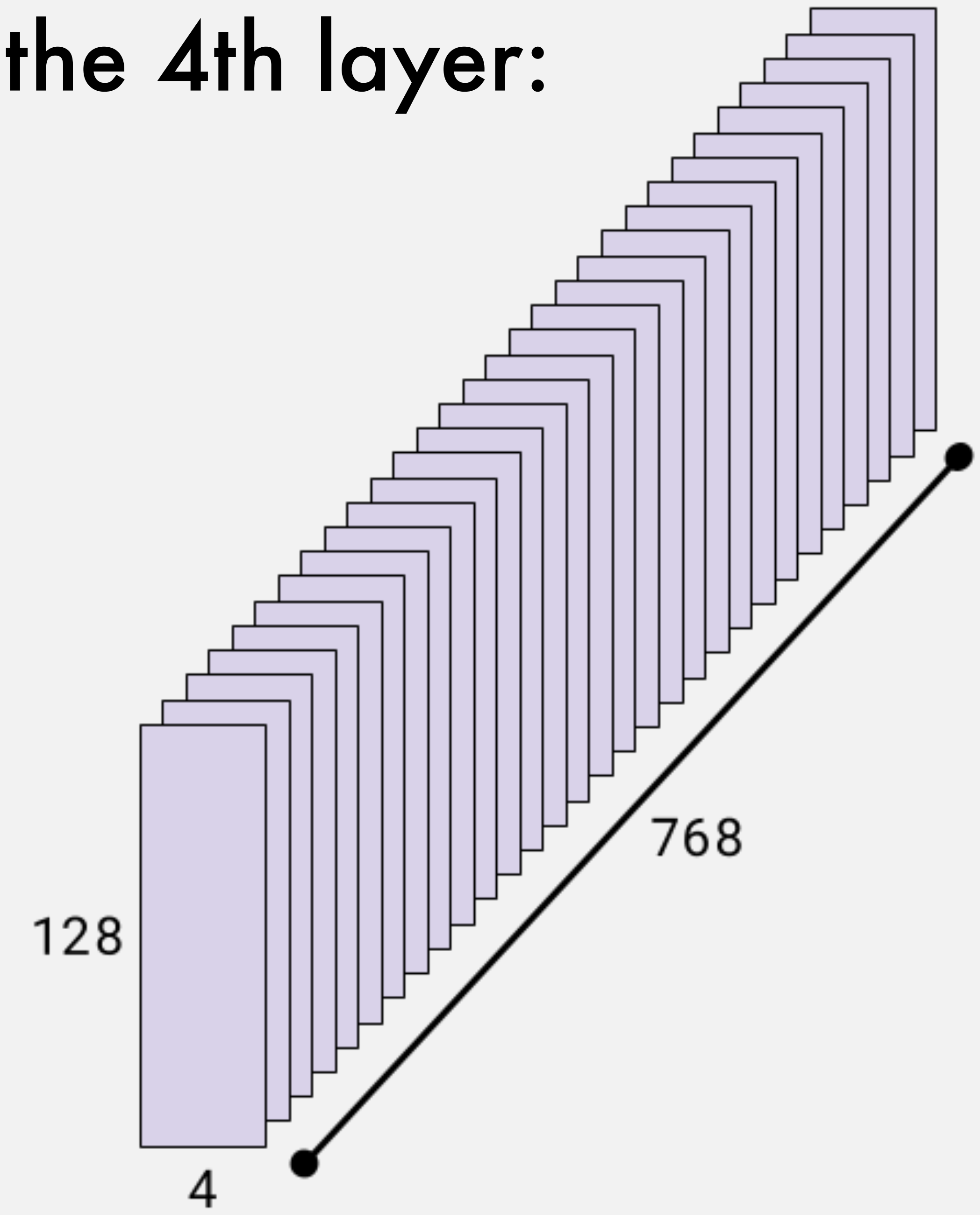
Why is the C dimension 1536?
Patch merging stacks four patches along the channel (C) dimension, increasing the channel count to 4C, 4x384=1536.

What are the inputs to the nn.Linear layer?
At the end of each layer, nn.Linear reduces the channel dimension from 4C to 2C.

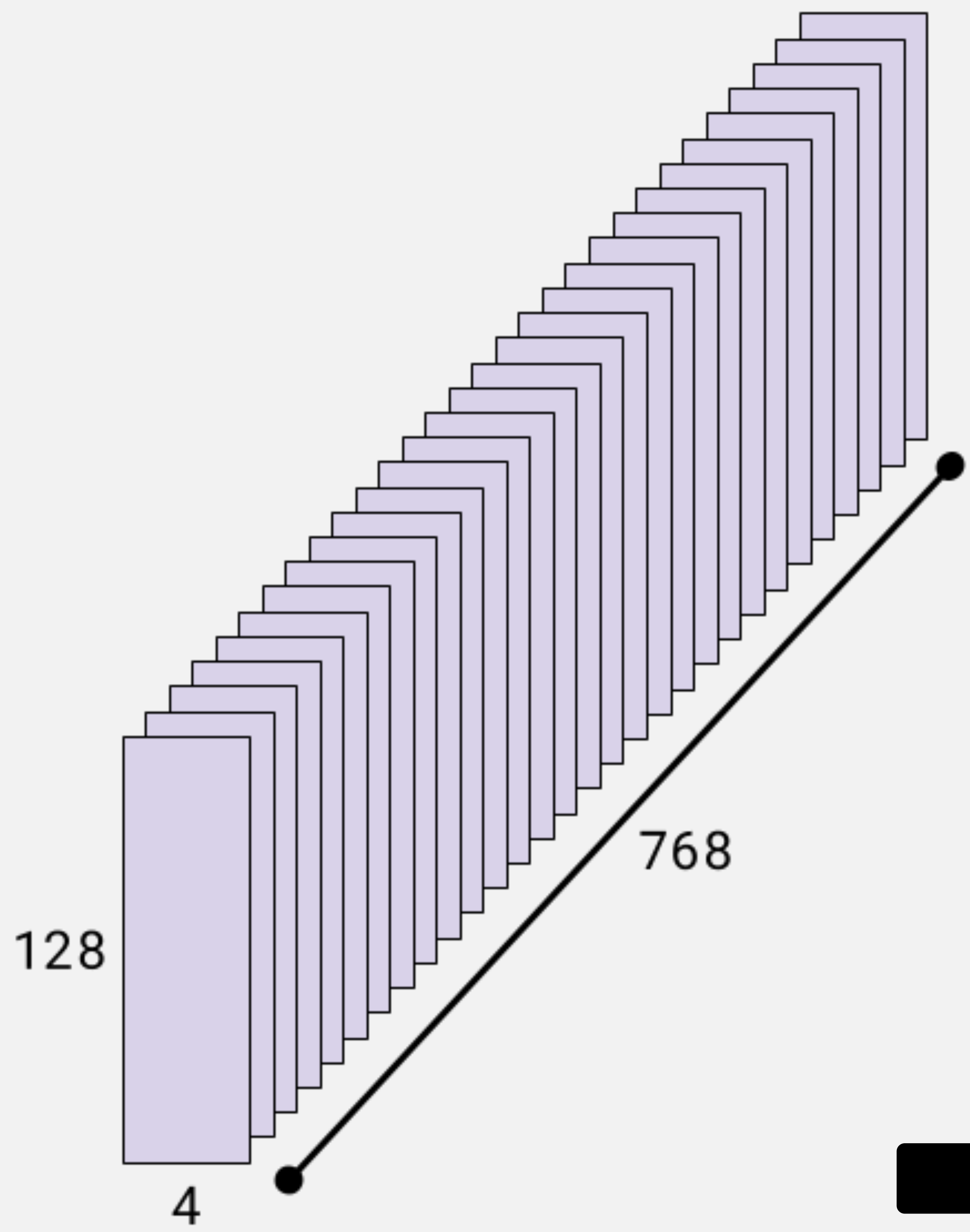
youtube.com/@mashaan14

4th layer

This is the input to the 4th layer:

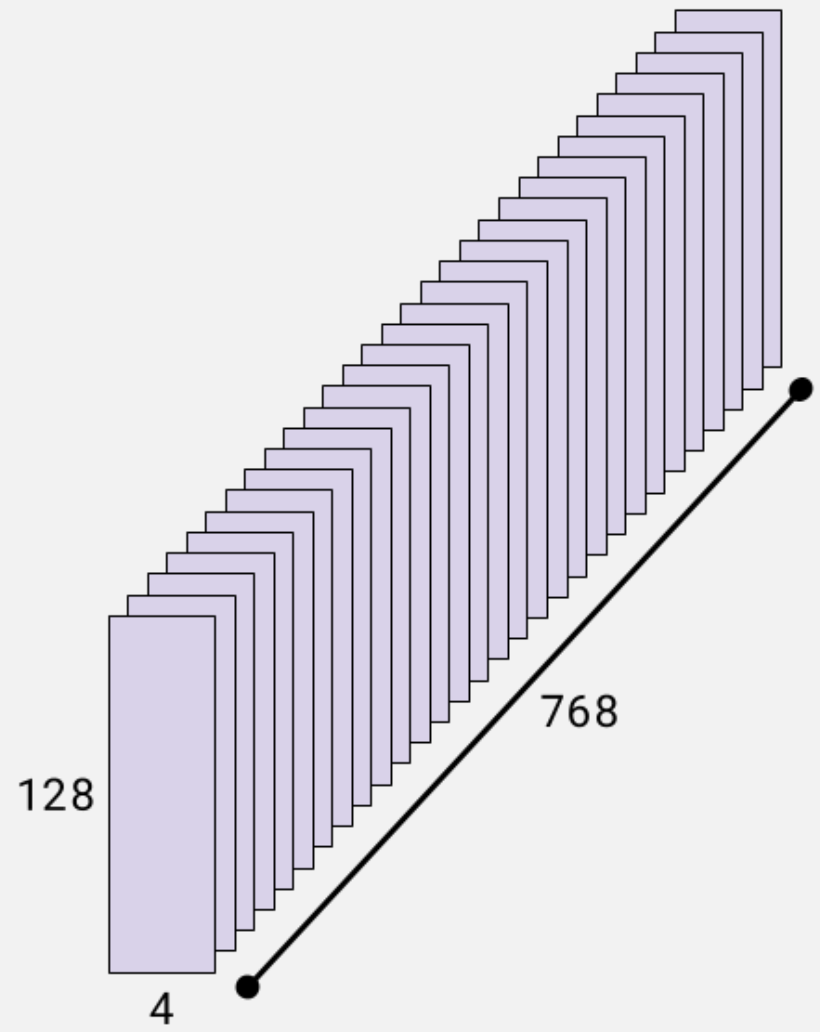


The PatchMerging operation is skipped at the 4th layer. Its output matches the 3rd layer's output.

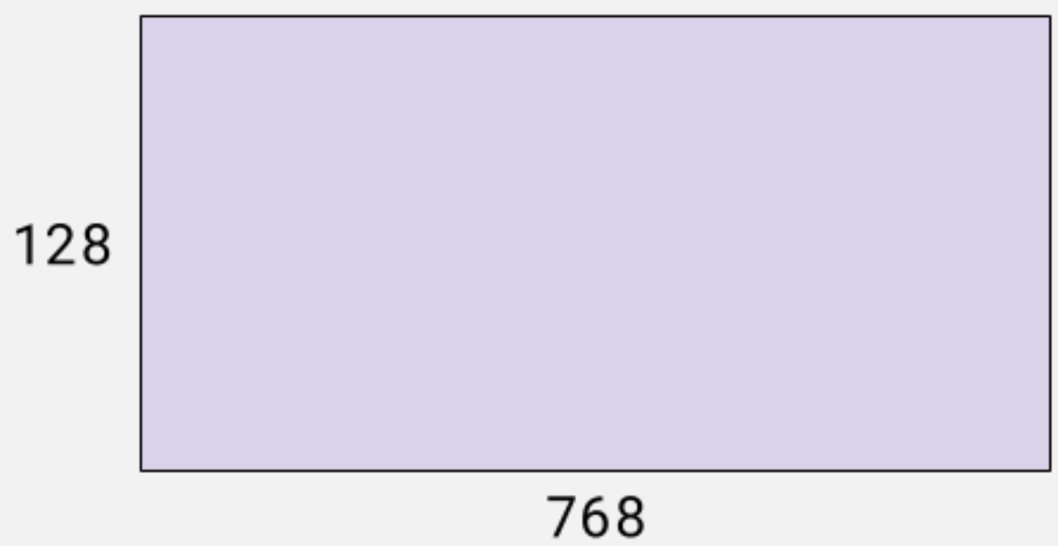


youtube.com/@mashaan14

Class Prediction




```
# average pooling along HxW dimension
x = self.avgpool(x.transpose(1, 2))
```




```
# num_features=768, num_classes=10 for cifar-10
nn.Linear(self.num_features, num_classes)
```