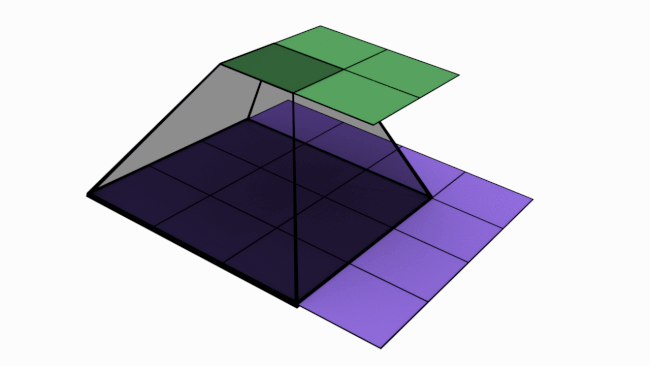
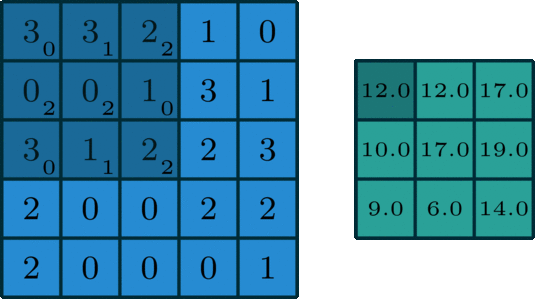
**Session 2 (25 Jul19):**

Every image genrate his own values, we don't have any control over them. For example here 3\*3 kernel is convolving 4\*4 size channel and every square is (represent the value, so here we are looking at the 16 values. Likewise 3\*3 kernel is reading each value and values in kernel is unknown. So our end task to change the values by kernel so we can get the feature of the channel (vertical and horizontal edges).

Green channel is our output channel and 3\*3 kernel is returning sum of 9 values to the green channel.Strip is how many pixels kernel passes to move to the next row/column.\



We can see the other example 5\*5 image and its convolving on 3\*3 kernel . 3x3 kernel has value of

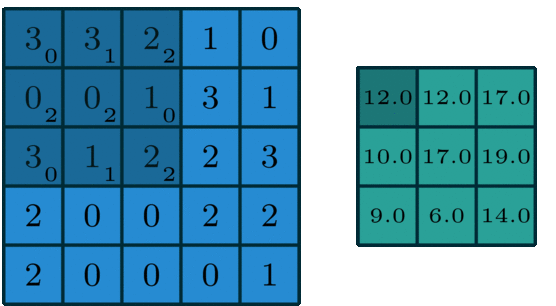
0 1 2

2 2 0

0 1 2

So in the next green layer 3\*3 kernel passing the value which will be the sum of 9 blue box.

For example



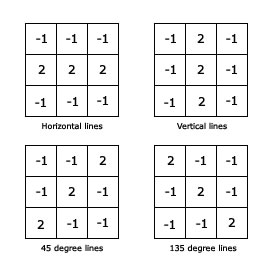
**3\*0+3\*1+2\*2+0\*2+0\*2+1\*0+3\*0+1\*1+2\*2=12**

**3\*0+2\*1+1\*2+0\*2+1\*2+3\*0+1\*0+2\*1+2\*2=12**

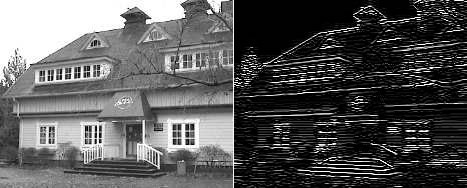
**2\*0+1\*1+0\*2+1\*2+3\*2+1\*0+2\*0+2\*1+3\*2=17**

Etc…

Some examples of edge detectors would be:



When we use the *horizontal edge detector kernel* with the values, as shown above, we get the following result:



Let's look at this through some numbers. Let us look at how a vertical edge would look like in an image:

0.2 0.2 **0.9** 0.2 0.5

0.1 0.1 **0.9** 0.3 0.2

0.0 0.2 **0.8** 0.1 0.1

0.2 0.3 **0.9** 0.1 0.2

0.1 0.1 **0.9** 0.3 0.2

The values shown in **bold** represents a vertical line in this image

Let us define our vertical kernel as:

-1 2 -1

-1 2 -1

-1 2 -1

After convolving the values we get are:

-2.0 4.3 -2.3

-1.7 4.1 -2.1

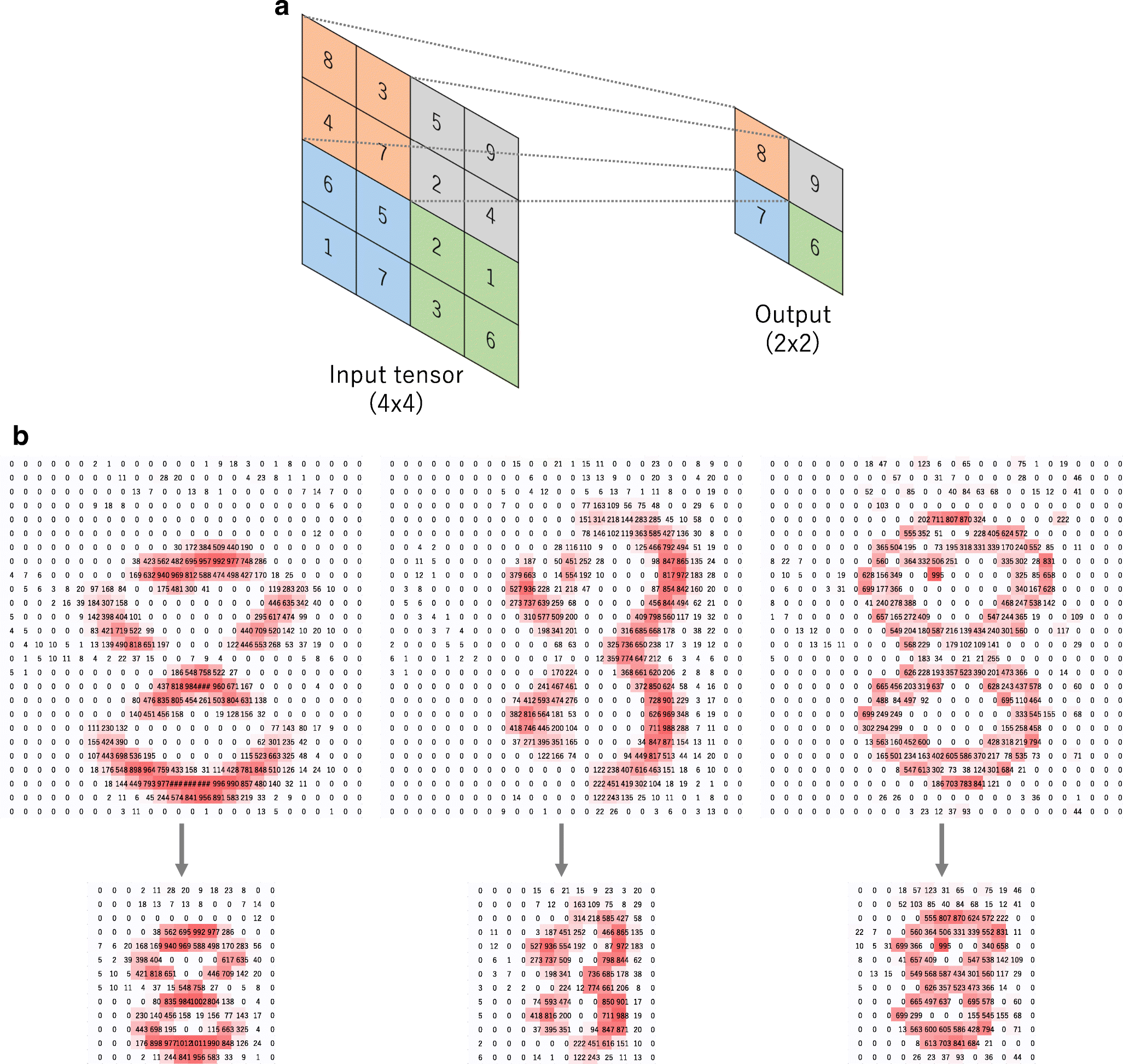
-1.7 4.1 -2.1

central vertical values in the 3x3 output layer above, the detection of the vertical line. **Not only we have detected the vertical line, we are also passing on an image/channel which shows a vertical line.**

For MaxPoo we have to use 2\*2 kernel rather then the 3\*3 kernel.

Example for 400\*400 we need total 400/2 =200 layers but applying the max pooling its reduce to the 27 layers so It will impact the performance and reduce the load on GPU.

### 



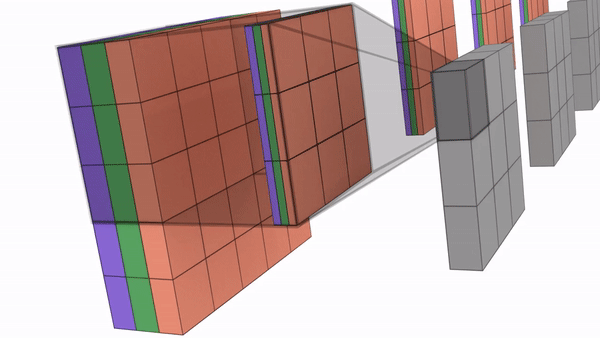
**400 | 398 | 396 | 394 | 392 | 390 | MP (2x2) 390/2=195**

### **195 | 193 | 191 | 189 | 187 | 185 | MP (2x2) 185/2=92.5(92)**

### **92 | 90 | 88 | 86 | 84 | 82 | MP (2x2) 82/2=41**

### **41 | 39 | 37 | 35 | 33 | 31 | MP (2x2) 31/2=15.5(15)**

### **15 | 13 | 11| 9 | 7 | 5 | 3 | 1**



To get the set of edges and gradients to represent the whole image we have to use write number of kernel. So on first layer we have to use 32 or 64 kernel and then we have to increase the size. So if we add 32 kernels in the first layer, 64 in second, 128 in third and so on... then our Network is

400x400x1 | (3x3)x32 | 398x398x32

398x398x32 | (3x3)x64 | 396x396x64

396x396x64 | (3x3)x128 | 394x394x128

394x394x128 | (3x3)x256 | 392x392x256

392x392x256 | (3x3)x512 | 390x390x512

MaxPooling

195x195x512...

We have a problem here. Even though till now we have used 32+64+128+256+512 kernels (which is a small number), we right now have 992 images in our Memory. We solved the issue of large channel size by using MaxPooling,(The use of maxpool will be as per the purpose or requirement of use the image).