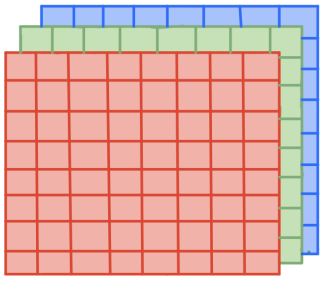
How are the images represented?

Previously, we loaded the images into an array of shape (# of pictures x 256 x 256 x 3). Each image is represented by a 256x256x3 matrix. What does this mean?

- 256x256: Each image is stored as a square image with a width of 256 pixels and a height of 256 pixels.
- 3: The three color channels (red, green, and blue) represent the image.
- We can think of these representations as a stack of three 256x256 matrices. In each color matrix, the values at a pixel location range from 0 to 255. Higher values at a pixel location indicate higher intensities of a certain color (a value of 0 at a particular pixel in the red matrix indicates no red in that location while a value of 255 represents full red intensity).

For the sake of illustration, let us suppose that our image is instead represented by a 8x8x3 matrix.



How does the conv 2D layer work?

- The conv2D layer works by sliding a filter over the input data (image representation) and multiplying the filter with the input data at the current window.
- Each filter (or kernel) is responsible for capturing a specific pattern in the data.
- If we have 32 filters, then each of the 32 filters will slide across all the data and locate unique patterns in that data.

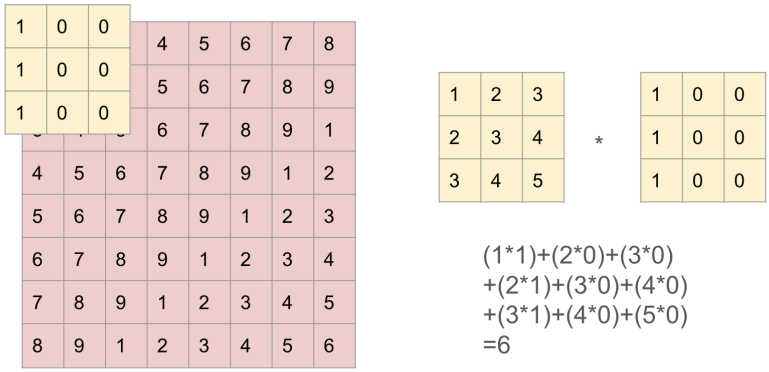
To better understand the convolutional layer, let us only look at the red matrix and a single filter.

- Here, the yellow block represents a filter. The filter is responsible for detecting certain features/ attributes of the image. Filters can detect edges, textures, corners, shapes, patterns and more.
- Again, the values in the red matrix represent the intensity of red in the particular pixel.

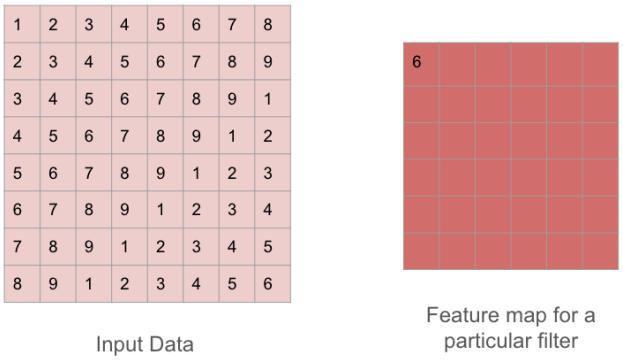
1	2	3	4	5	6	7	8
2	3	4	5	6	7	8	9
3	4	5	6	7	8	9	1
4	5	6	7	8	9	1	2
5	6	7	8	9	1	2	3
6	7	8	9	1	2	3	4
7	8	9	1	2	3	4	5
8	9	1	2	3	4	5	6

1	0	0
1	0	0
1	0	0

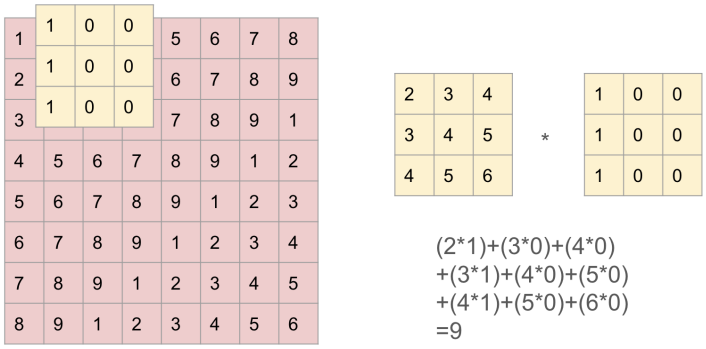
We slide the 3x3 kernel over the first 3x3 window in the data.



We take the dot product of the filter and the values in the current window. This value gets stored in the feature map for the particular filter.



Slide the kernel to the next window and compute the dot product.



Update feature map

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

6	9				

Complete the feature map by sliding the filter across all the input data

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

6	9	12	15	18	21
9	12	15	18	21	24
12	15	18	21	24	18
15	18	21	24	18	12
18	21	24	18	12	6
21	24	18	12	6	9

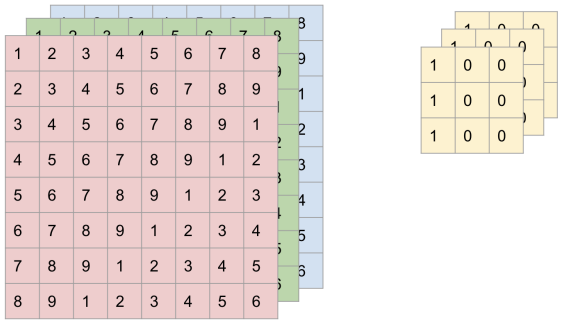
We have completed a feature map for 1 filter. We now do this for all 32 filters. After this, there will be 32 filter maps. The output of this layer is a 6x6x32 matrix.

The final step before passing the darker red block into the next layer is RELU activation. RELU introduces nonlinearity into the data by replacing all negative values with 0s and leaving the positive values unchanged. Since our output feature map has no negative values, we do not make any changes.

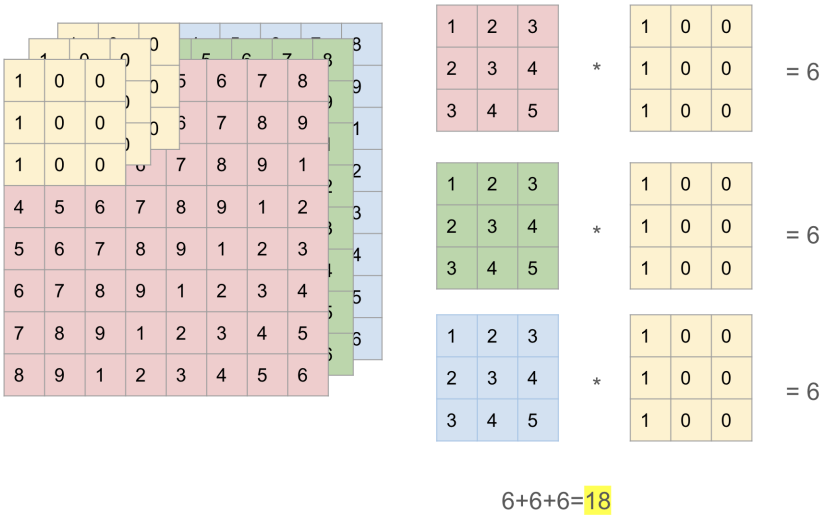
How does the conv2d layer work with RGB images?

Recall that we are working with three color channels instead of one. To adjust for this, we slide a 3D filter ***block*** along the data instead of a 2D filter matrix. Notice that the depth of the filter (3) matches the number of channels (3) in the original matrix. If the matrix had another channel, say an orange channel, then the filter block would be 3x3x4.

Note that the yellow filter cube is simply three of the original filter matrices stacked on top of each other.

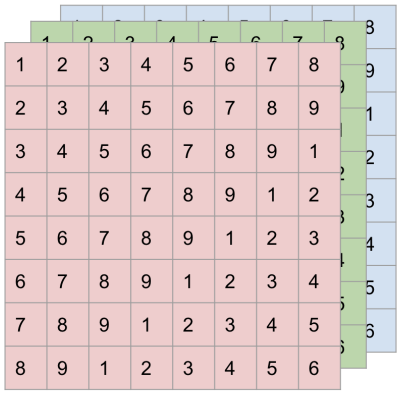


We slide the filter cube onto the first window and take the dot product of the values at the current window with the values in the filter cube. The value we get out of the initial filter pass is the sum of 27 multiplications instead of the nine we had in the 2D case.

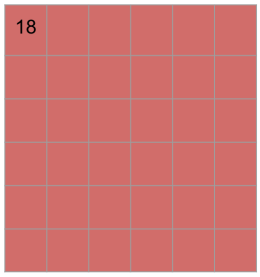


6+6+6=18

Updating our feature map for the particular filter

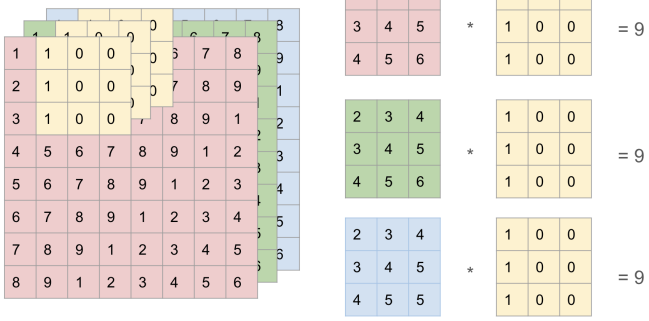


Input Data



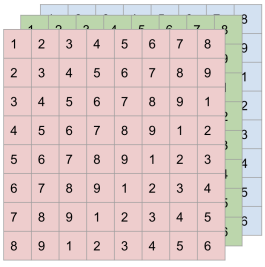
Feature Map for
a particular filter

Sliding the filter along the input sequence



9+9+9=27

Updating the feature map

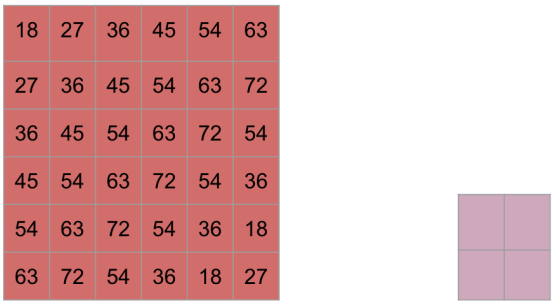


Input Data

Feature Map for
a particular filter

This is the complete feature map of a single filter over all the data. Remember that if there are 32 filters, there will be 32 feature maps. The output of this layer is a 6x6x32 matrix.

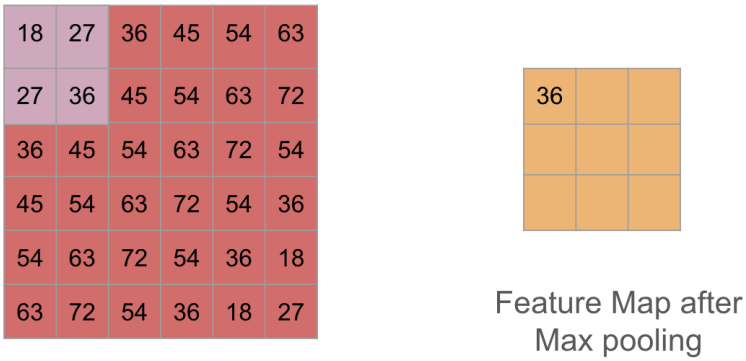
Global max pooling
We next perform global max pooling as a form of dimensionality reduction.



Feature map of a particular feature

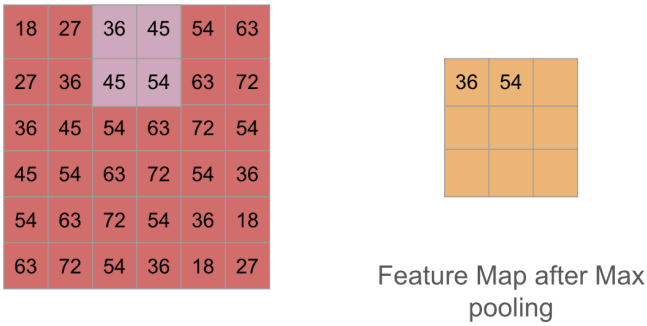
Max pooling block

We apply the max pooling block on the first 2x2 window and select the largest value to put into the reduced matrix.



Feature Map after Max pooling

Slide onto the next 2x2 ***non overlapping*** region.



Feature Map after Max pooling

Do this for all the non overlapping regions.

18	27	36	45	54	63
27	36	45	54	63	72
36	45	54	63	72	54
45	54	63	72	54	36
54	63	72	54	36	18
63	72	54	36	18	27

36	54	72
54	72	72
72	72	72

Feature Map after Max pooling

Do this for all 32 feature maps. The final output shape after max pooling will be a 3x3x32 matrix. This is the result after one step of the Conv2D and max pooling layer.

Adding Additional Conv2D layers

Most neural networks use several Conv2D and max pooling layers. If we were to apply another Conv2D layer, we would simply use our resulting 3x3x32 matrix as an *input* for the layer. If we were to keep our window size as 3x3 and the number of filters as 32, our filter block would then be 3x3x32. Notice that the last dimension (32–the depth) should always match the depth of the matrix being fed into the data.

Flattening, Dense Layers, and Softmax Activation

Once we have added all the Conv2D layers we want, we generally flatten the 3D output from the Conv2D layer into a 1D output that will be fed into the fully connected dense layers. The dense layers consist of nodes that analyze the flattened features. Each node in the layer is associated with a weight that is learned during training. The number of nodes in the dense layers and the number of layers themselves are design choices.

The final step for a multiclass classification problem is softmax activation. Softmax converts the raw output of the previous dense layer into probabilities for each class. The number of units in the softmax layers represents the number of classes in the classification problem (in our case 4: glioma, meningioma, and pituitary, and no tumor). The network predicts the class of the observation by assigning it to the class with the highest probability.