CNN

Step 1: Assuming we are having a 6x6 data:

1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	1	0	0
0	0	0	1	1	0
0	0	0	0	1	0
0	0	0	0	1	0

1: wet
2: dry

Step 1: Assuming we are having a 6x6 data:

1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	1	0	0
0	0	0	1	1	0
0	0	0	0	1	0
0	0	0	0	1	0

1: wet
2: dry

Step 2: the neuron (or filter) in CNN includes some basic information of the data ~ a filter is a matrix, e.g., the following example gives two 3x3 filters

Filter 1

1	-1	-1
-1	1	-1
-1	-1	1

Filter 2

-1	1	-1
-1	1	-1
-1	1	-1

- The elements in each filter is the parameters to be determined through the training process (like the weights in ANN).
- in this example, if we have a 3x3 filter, then CNN only look at the area of 3x3 in the image at once instead of every grid point.

Step 1: Assuming we are having a 6x6 data:

1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	1	0	0
0	0	0	1	1	0
0	0	0	0	1	0
0	0	0	0	1	0

1: wet
2: dry

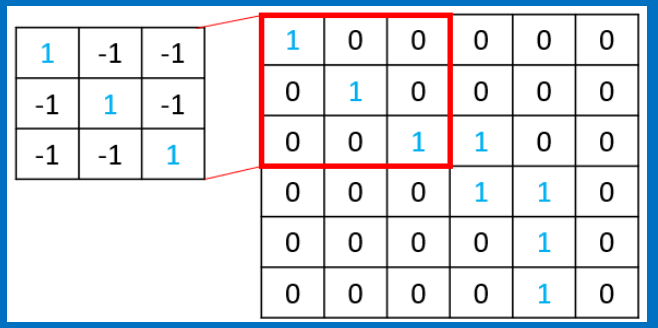
Step 2: the neuron (or filter) in CNN includes some basic information of the data ~ a filter is a matrix, e.g., the following example gives two 3x3 filters

Filter 1		
1	-1	-1
-1	1	-1
-1	-1	1

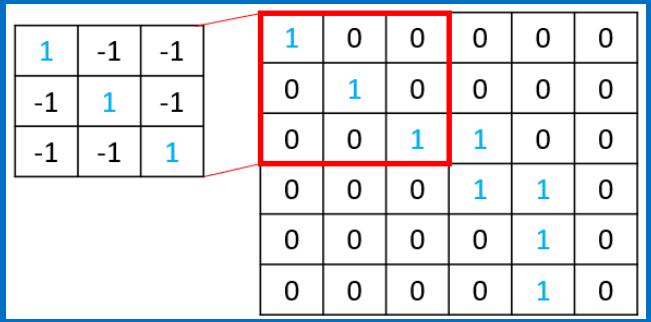
Filter 2		
-1	1	-1
-1	1	-1
-1	1	-1

- The elements in each filter is the parameters to be determined through the training process (like the weights in ANN).
- in this example, if we have a 3x3 filter, then CNN only look at the area of 3x3 in the image at once instead of every grid point.

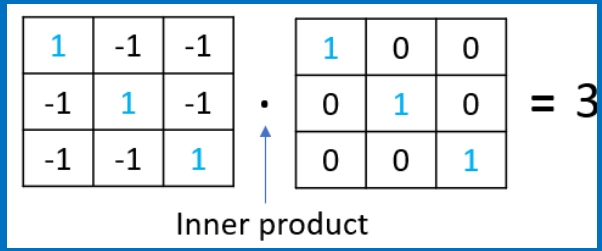
Step 3: Applying the Filter onto the image from the top-left corner:



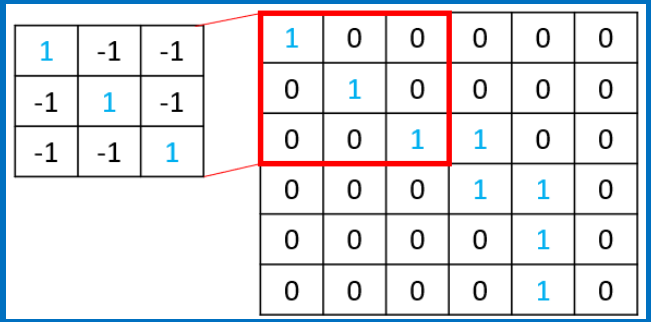
Step 3: Applying the Filter onto the image from the top-left corner:



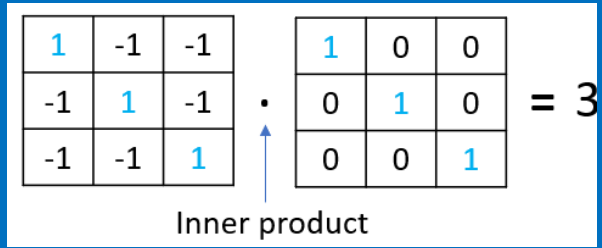
Step 4: Generating the inner product between the filter the corresponding area of the image



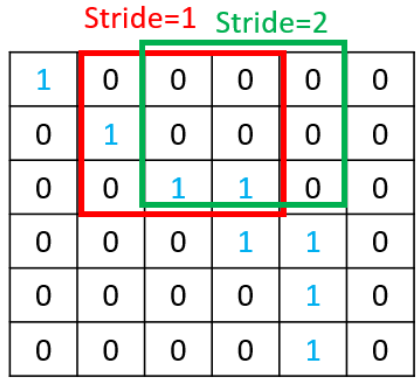
Step 3: Applying the Filter onto the image from the top-left corner:



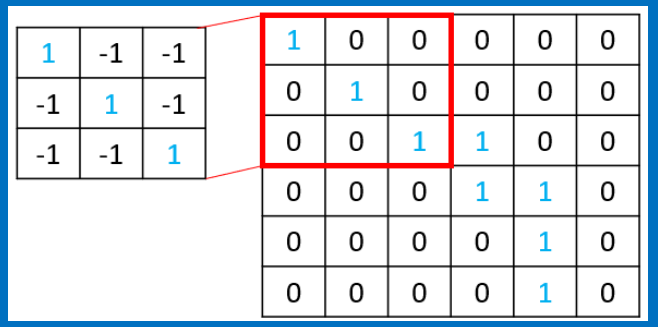
Step 4: Generating the inner product between the filter the corresponding area of the image



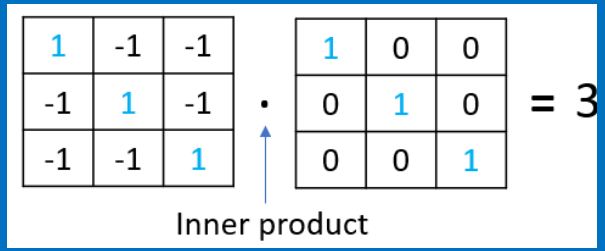
Step 4: moving the filter the entire dataset based on a predefined “stride”, and get the inner product for each movement



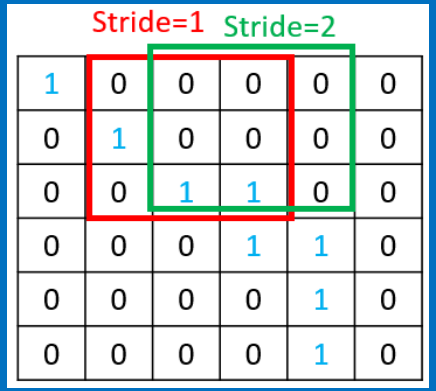
Step 3: Applying the Filter onto the image from the top-left corner:



Step 3.1: Generating the inner product between the filter the corresponding area of the image

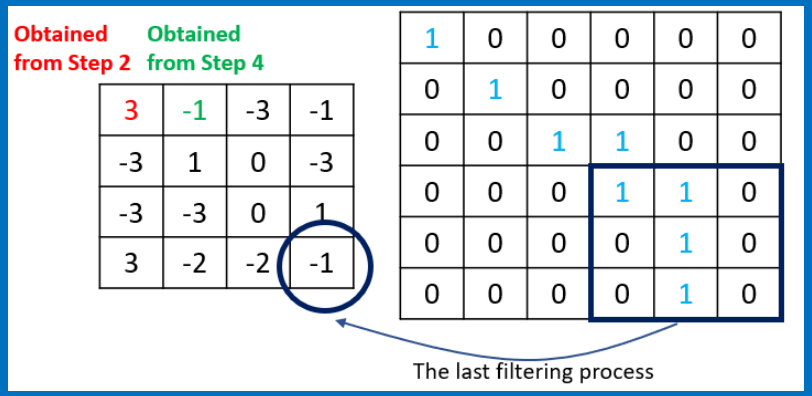


Step 4: moving the filter the entire dataset based on a predefined "stride", and get the inner product for each movement



the purpose of this filter is to detect if there is similar pattern of the filter in the image. The higher number in the inner product matrix usually means the higher similarity between the filter and the corresponding area of image

Note that in a CNN there are usually a dozen of filters.



1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	1	0	0
0	0	0	1	1	0
0	0	0	0	1	0
0	0	0	0	1	0

6x6 data

Input data matrix

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

4x4 data

The inner product matrix from a filter

1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	1	0	0
0	0	0	1	1	0
0	0	0	0	1	0
0	0	0	0	1	0

6x6 data

↑

Input data matrix

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

4x4 data

↑

The inner product matrix from a filter

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

pooling

↑

Using ”pooling” to reduce the size of the inner product

1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	1	0	0
0	0	0	1	1	0
0	0	0	0	1	0
0	0	0	0	1	0

6x6 data

↑

Input data matrix

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

4x4 data

↑

The inner product matrix from a filter

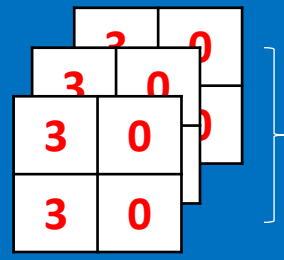
Taking the maximum value from this subdomain

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

pooling

↑

Using ”pooling” to reduce the size of the inner product



We can have many of these small matrices (depth = the number of filters)

2x2 data

1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	1	0	0
0	0	0	1	1	0
0	0	0	0	1	0
0	0	0	0	1	0

6x6 data

1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	1	0	0
0	0	0	1	1	0
0	0	0	0	1	0
0	0	0	0	1	0

6x6 data

1	**1**
2	0
3	0
4	0
5	0

7	**0**
8	1
9	0
	0
	0

13	**0**
14	0
15	1
	0

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

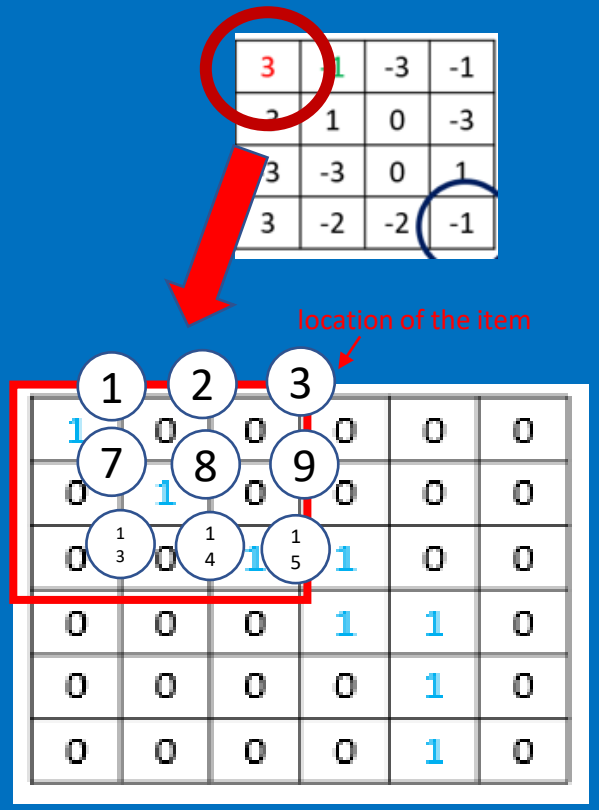
1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	1	0	0
0	0	0	1	1	0
0	0	0	0	1	0
0	0	0	0	1	0

6x6 data

1	**1**
2	0
3	0
4	0
5	0

7	**0**
8	1
9	0
	0
	0

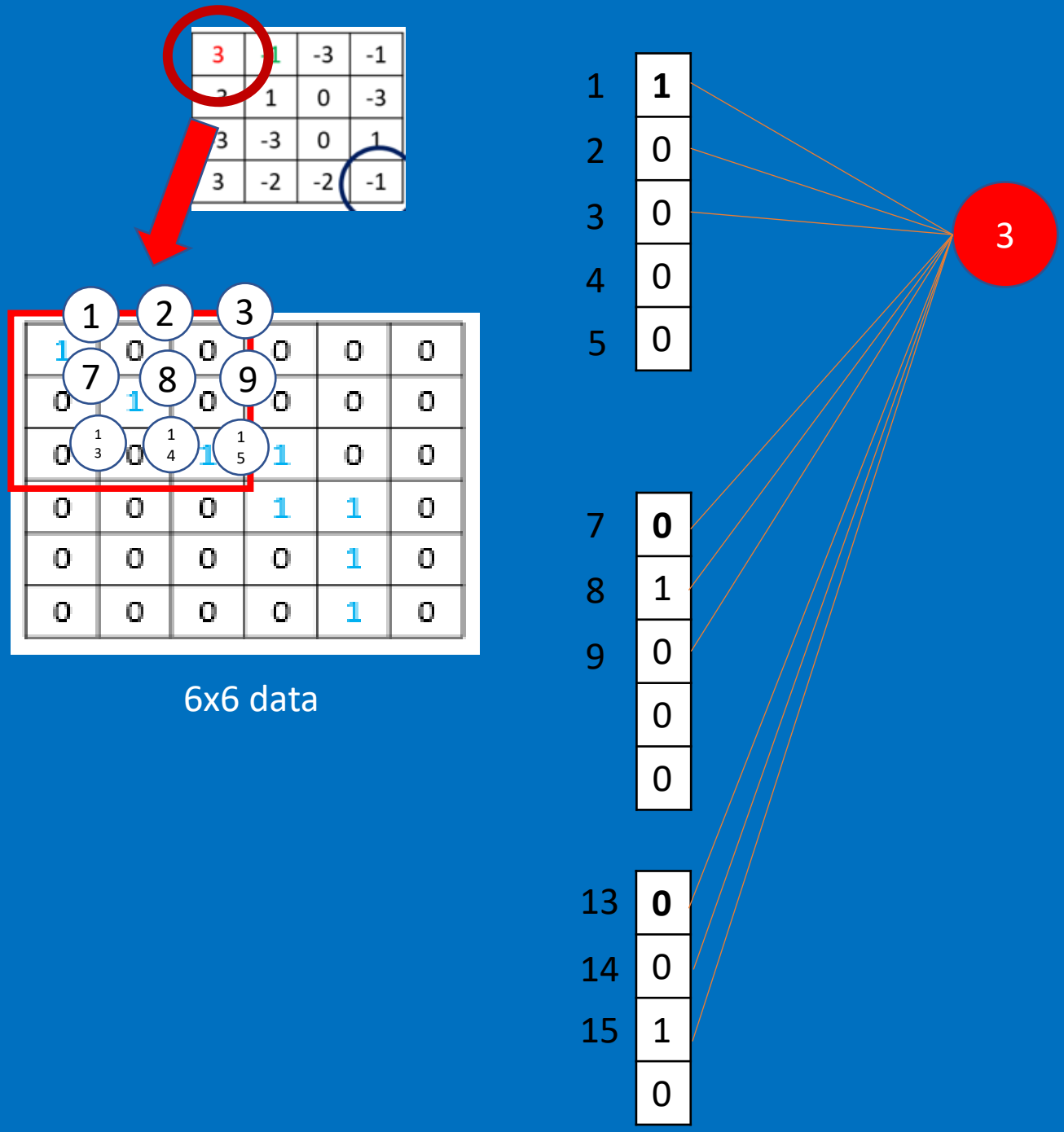
13	**0**
14	0
15	1
	0

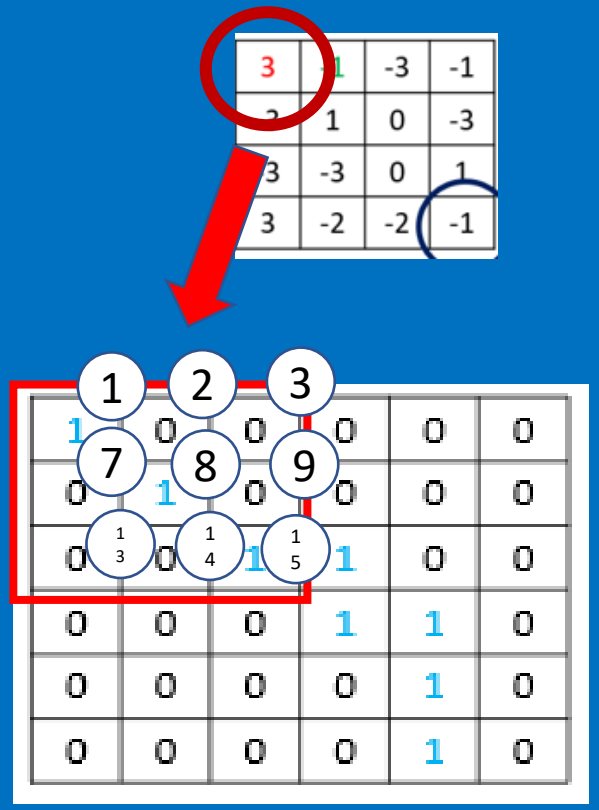


1	**1**
2	0
3	0
4	0
5	0

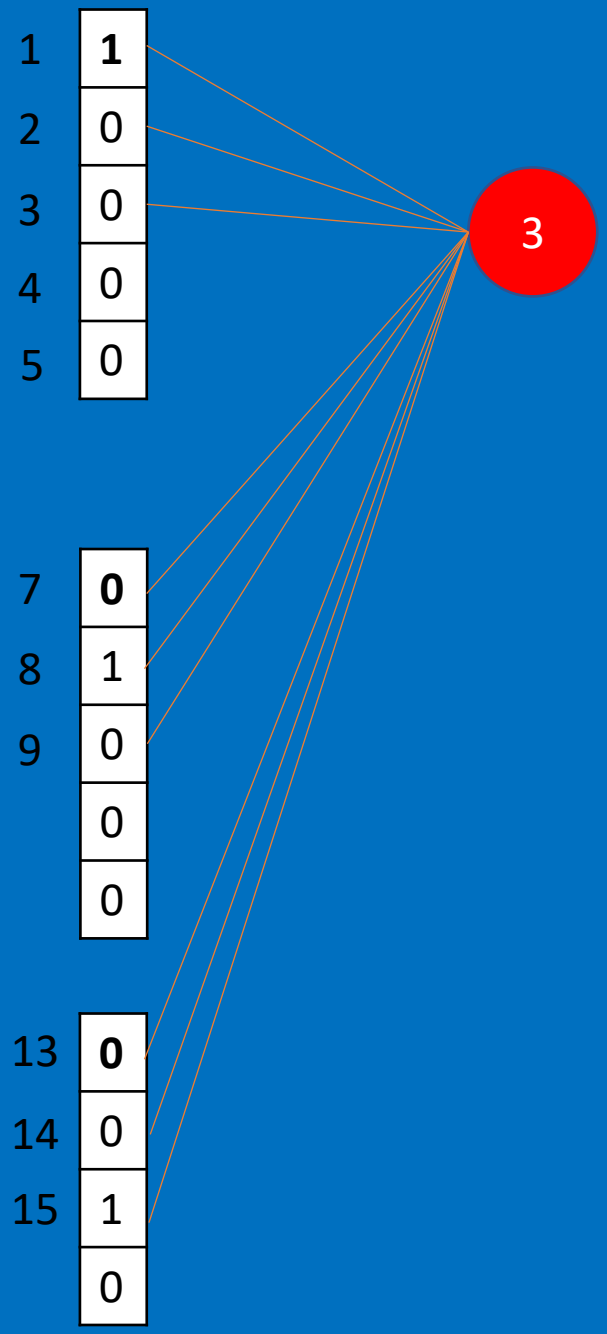
7	**0**
8	1
9	0
	0
	0

13	**0**
14	0
15	1
	0

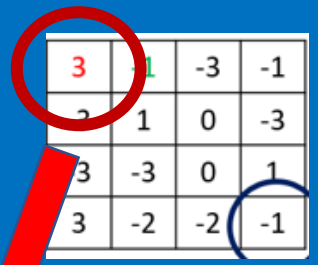

3 1 -3 -1
1 0 -3
3 -3 0 1
3 -2 -2 -1
1 2 3
7 8 9
13 14 15
1 0 0 0 0 0
0 1 0 0 0 0
0 0 1 1 0 0
0 0 0 1 1 0
0 0 0 0 1 0
0 0 0 0 1 0
6x6 data
1 1
2 0
3 0
4 0
5 0
7 0
8 1
9 0
0
0
13 0
14 0
15 1
0
3

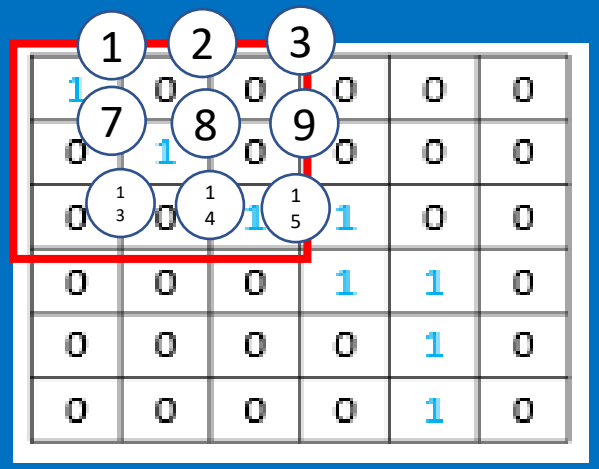


6x6 data

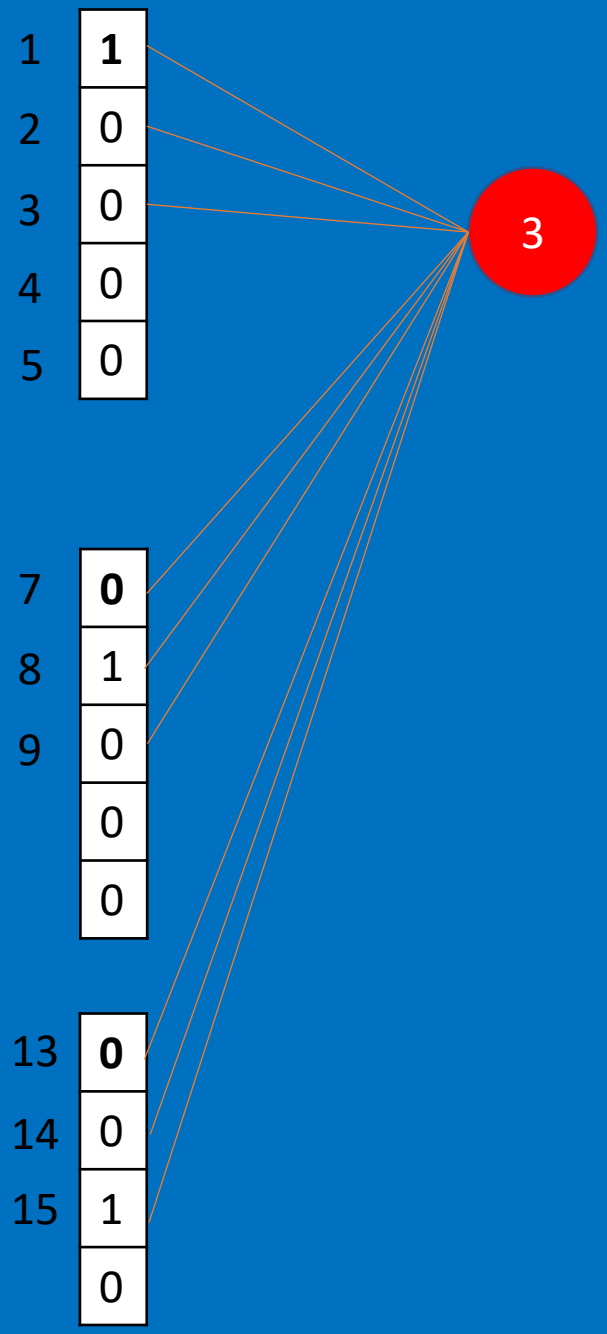


So for this neuron (“3”) it only needs to have the weights for some of the elements





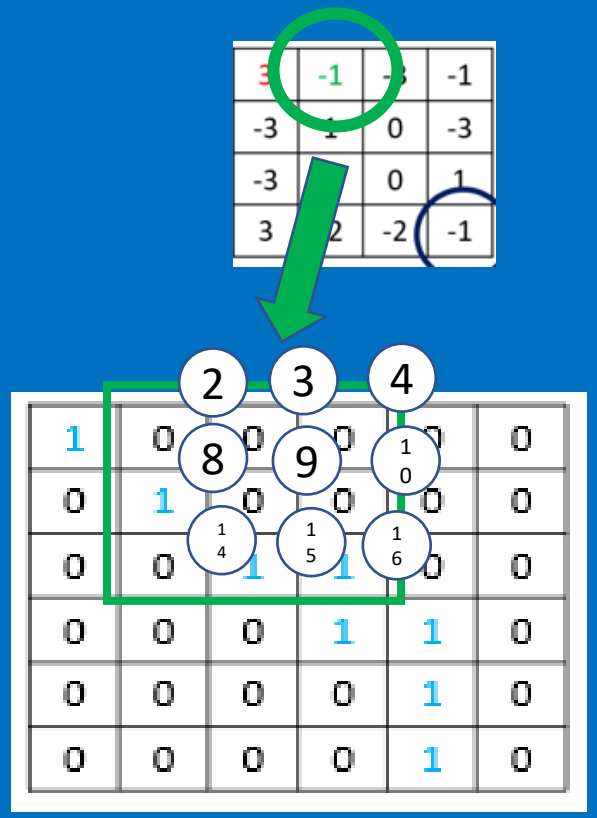
6x6 data



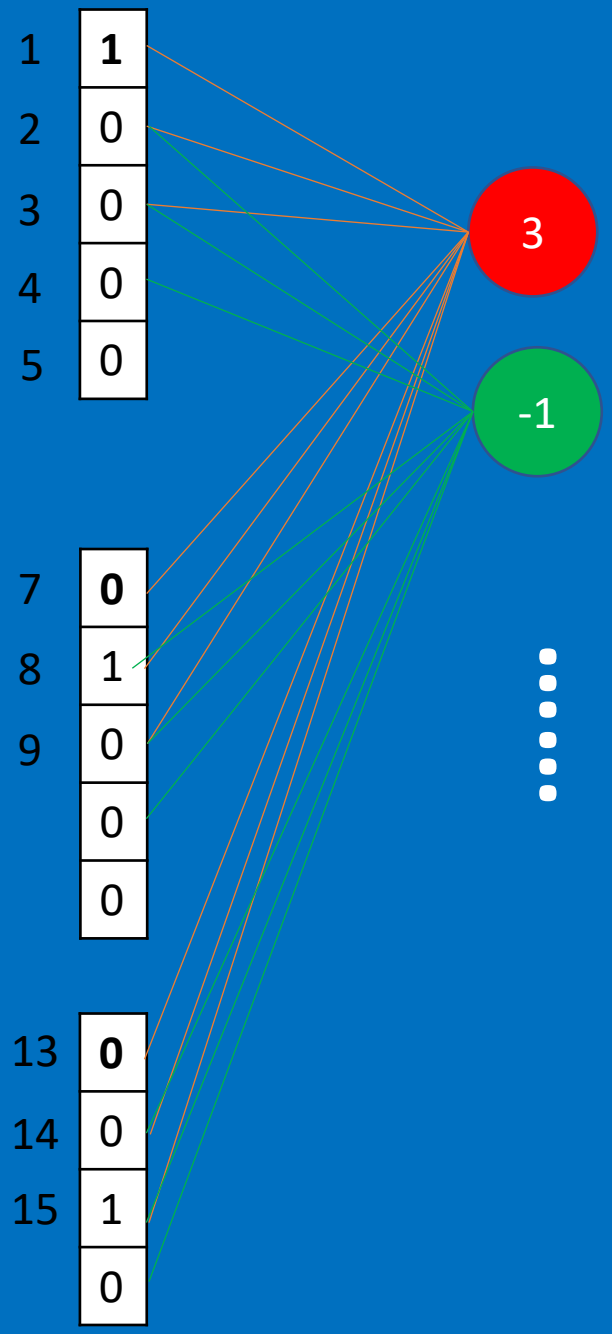
So for this neuron (“3”) it only needs to have the weights for some of the elements

1	-1	-1
-1	1	-1
-1	-1	1

And the weight for these 9 synapses is “defined” by the filter itself



6x6 data

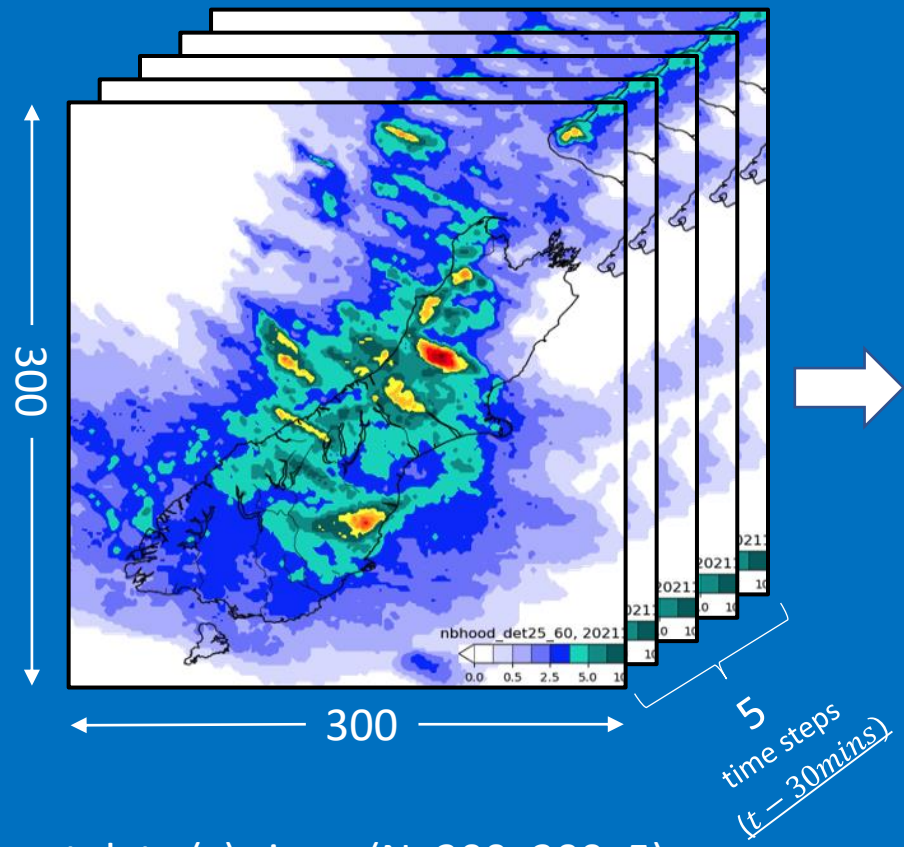


Therefore, the solution of CNN turns to something like ANN, all we need to do is just to solve the weights (which included in the filter) through the training (e.g., the cost function in ANN).

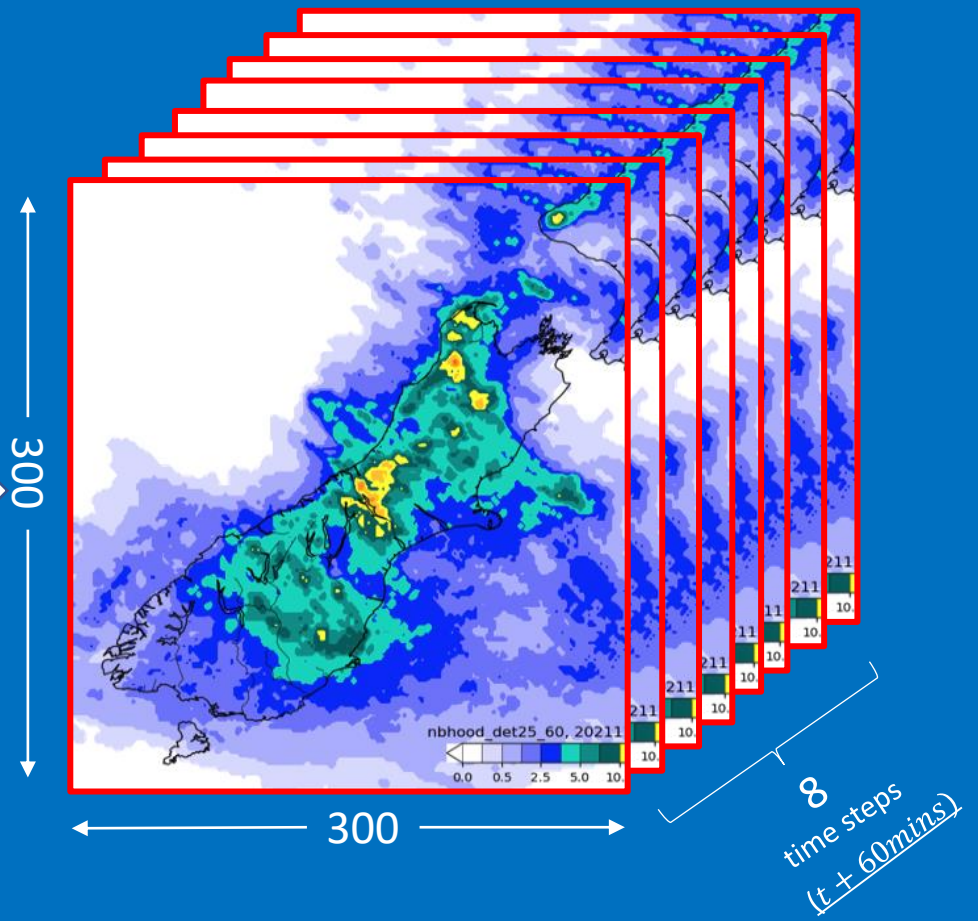
Best practice for CNN

https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d7

https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/



A Deep Learning model



Input data (x) size: (N, 300, 300, 5)

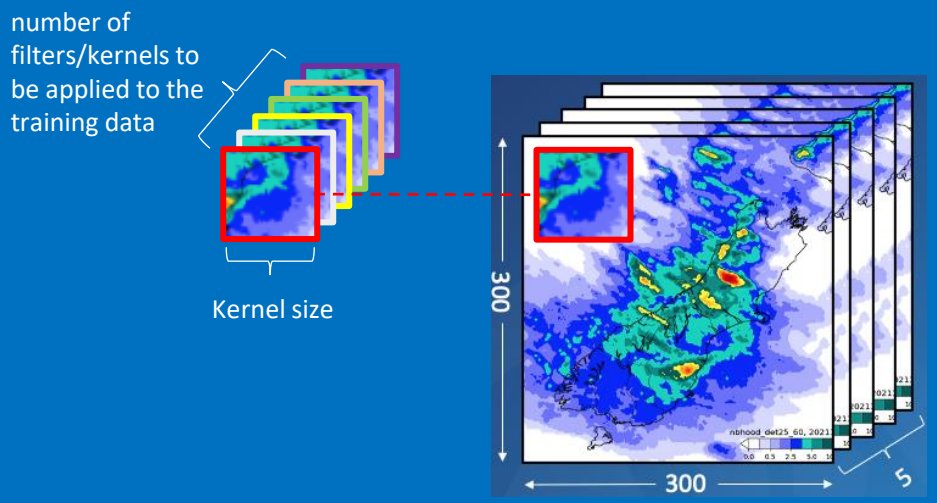
number of cases

Input data (y) size: (N, 300, 300, 8)

number of cases

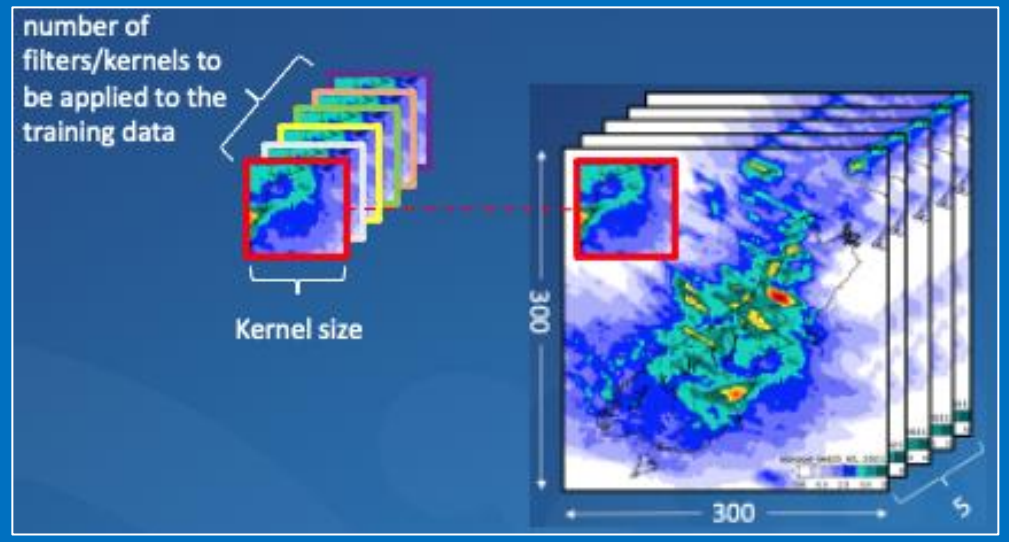
Best practice for CNN: How filter works?

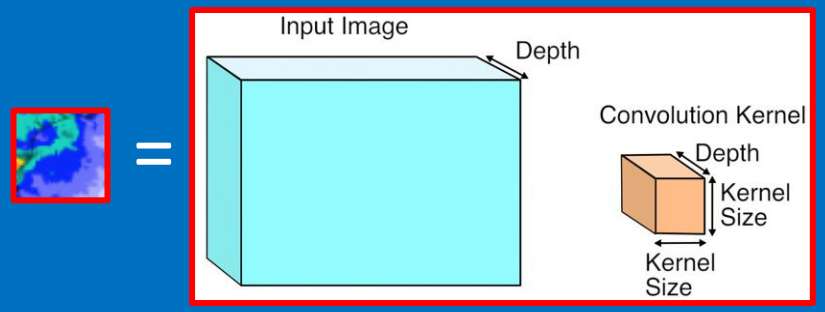
https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/
https://stackoverflow.com/questions/43306323/keras-conv2d-and-input-channels



Best practice for CNN: How filter works?

https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/
https://stackoverflow.com/questions/43306323/keras-conv2d-and-input-channels



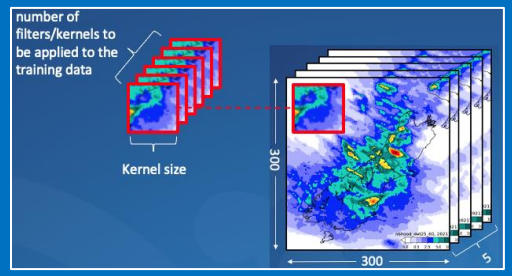


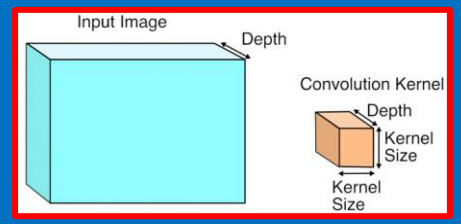
in Conv2D, a filter actually is 3D which contains the depth of the image (in this case the depth is 5)

Best practice for CNN: How filter works?

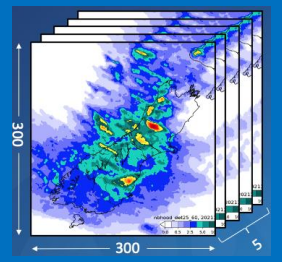
https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/
https://stackoverflow.com/questions/43306323/keras-conv2d-and-input-channels

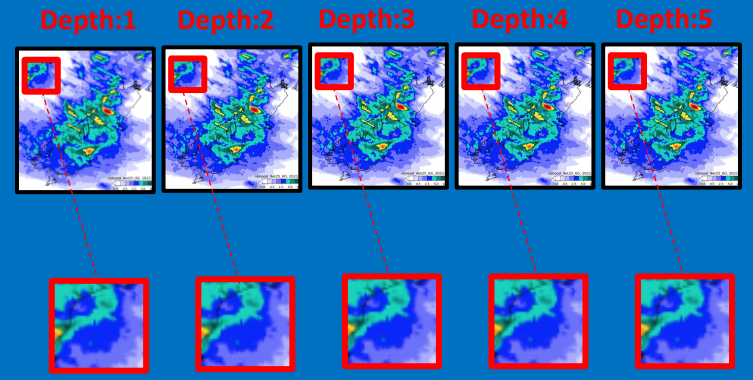
How the filter works ?





in Conv2D, a filter actually is 3D which contains the depth of the image (in this case the depth is 5)



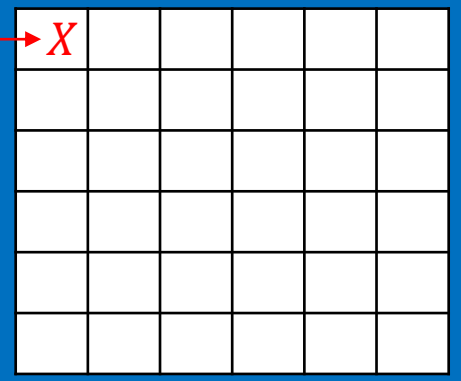


Inner product

Inner product combination

$X = X_1 + X_2 + X_3 + X_4 + X_5$

Put the combined inner product into the "inner product matrix"



A good example from
https://stackoverflow.com/questions/43306323/keras-conv2d-and-input-channels

Best practice for CNN: How filter works?

https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/
https://stackoverflow.com/questions/43306323/keras-conv2d-and-input-channels

Thumb of rules: How to set the number of filters

In most cases, over layers the number of filters is generally ascending. The reason is that:

The higher the number of filters, the higher the number of *abstractions* that your Network can extract from image data.

The reason why the number of filters is generally ascending is that at the input layer the Network receives raw pixel data. Raw data are always noisy, and this is especially true for image data.
Because of this, we let CNNs extract first some relevant information from noisy, "dirty" raw pixel data. Once the useful features have been extracted, then we make the CNN elaborate more complex abstractions on it.

That is why the number of filters usually increases as the Network gets deeper, even though it doesn't necessarily have to be like that.

Best practice for CNN: Why max pooling

Reducing Computational Load

Since max pooling is reducing the resolution of the given output of a convolutional layer, the network will be looking at larger areas of the image at a time going forward, which reduces the amount of parameters in the network and consequently reduces computational load.

Reducing Overfitting

Additionally, max pooling may also help to reduce overfitting.

For example, from a radar image, a filter looks for the pattern which is closest to itself. From the output of the convolutional layer, we can think of the higher valued pixels as being the ones that are the most similar to the filter.

With max pooling, as we're going over each region from the convolutional output, we're able to pick out the most activated pixels and therefore “only” preserve the features match the filter the most while discarding those features which are not very close to the filter

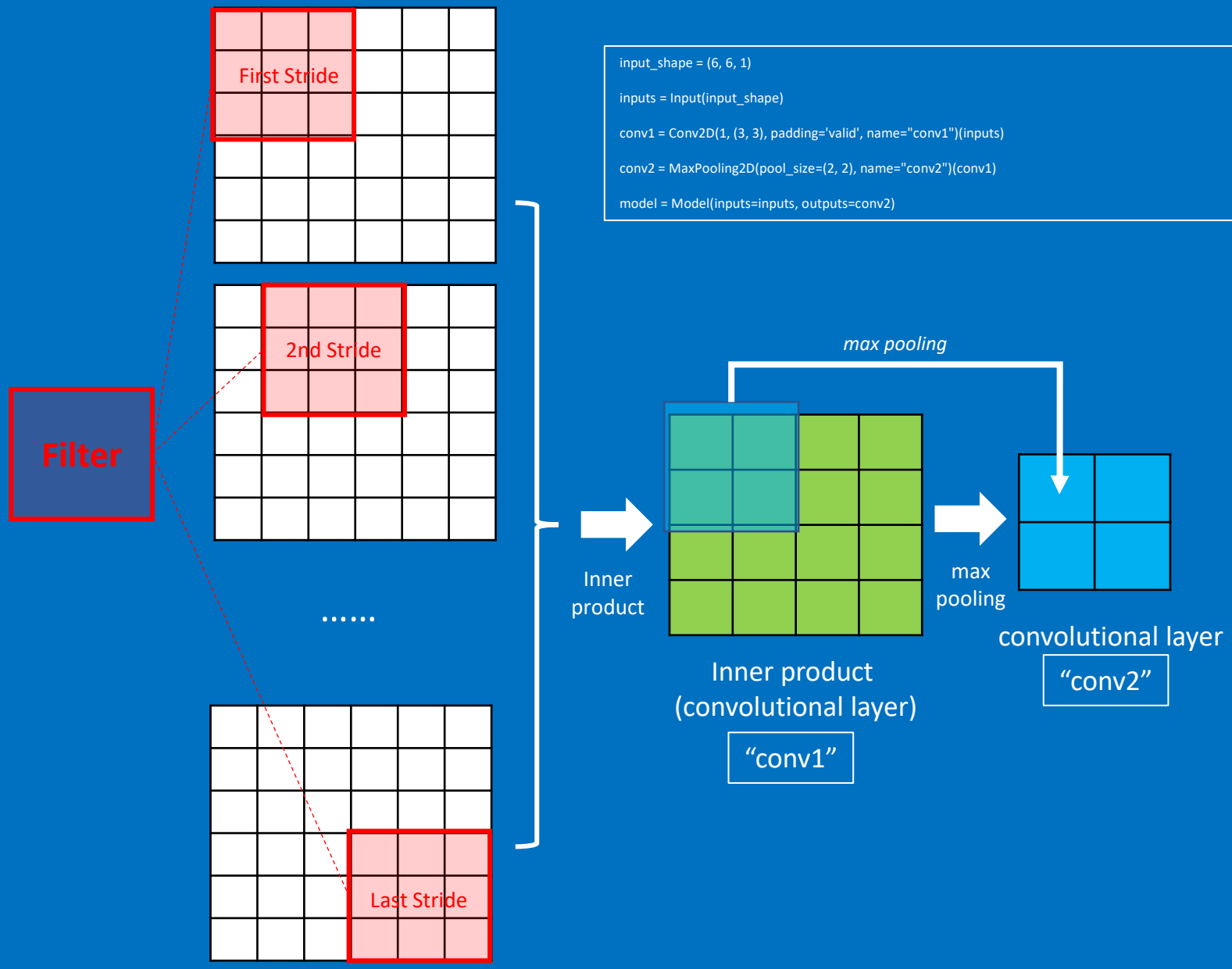
Drop out

We randomly remove neurons in the neural network during the training to reduce the chances of overfitting

https://www.quora.com/How-does-pooling-control-overfitting-in-CNN

Best practice for CNN: up-sampling & conv transpose

https://towardsdatascience.com/understand-transposed-convolutions-and-build-your-own-transposed-convolution-layer-from-scratch-4f5d97b2967




```
input_shape = (6, 6, 1)

inputs = Input(input_shape)

conv1 = Conv2D(1, (3, 3), padding='valid', name="conv1")(inputs)

conv2 = MaxPooling2D(pool_size=(2, 2), name="conv2")(conv1)

model = Model(inputs=inputs, outputs=conv2)
```

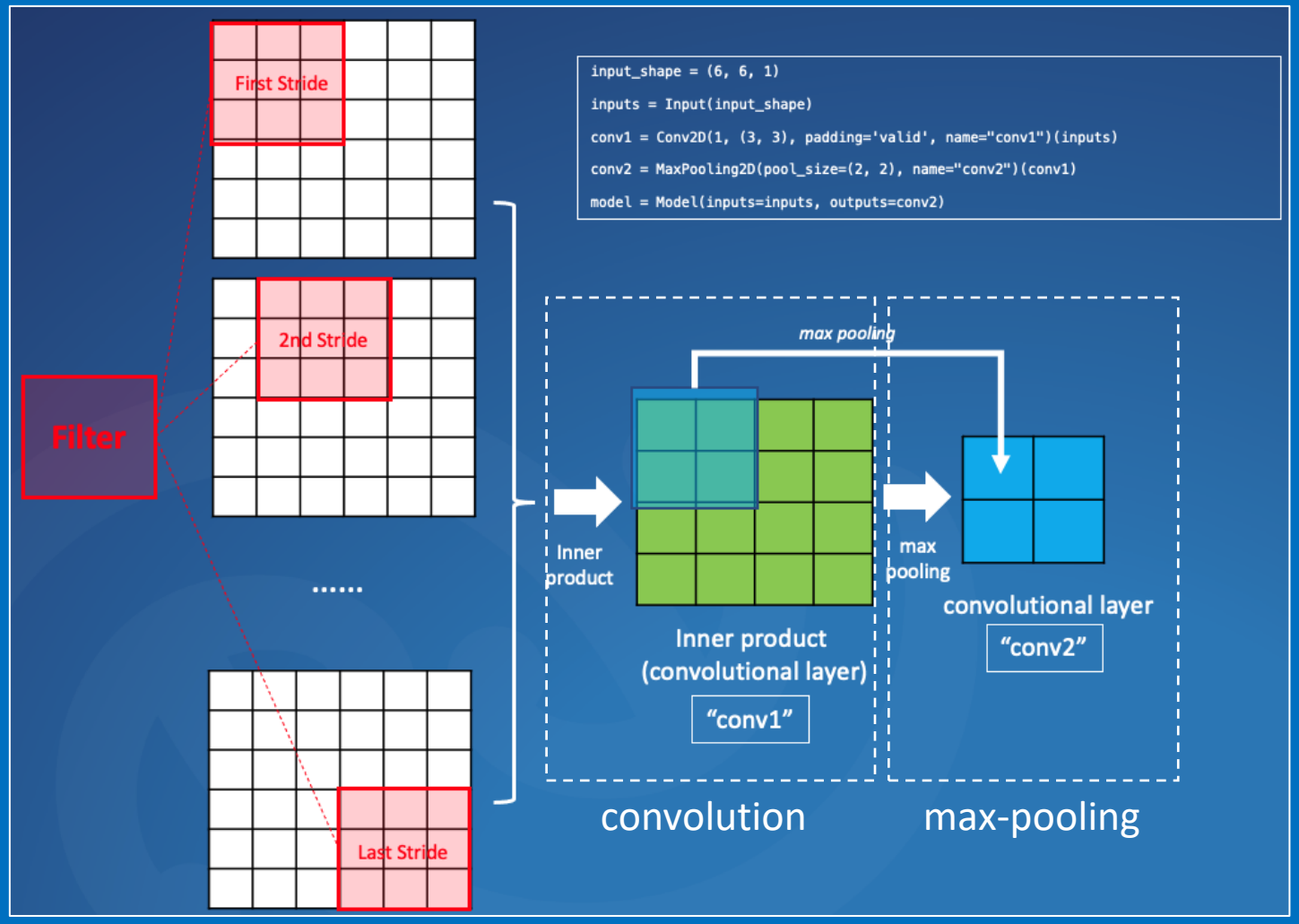
Without any padding, this operation transforms a "6x6" matrix into a "4x4" matrix, after max-pooling the matrix becomes "2x2"

Now, our question is: what if we want to go backward from a 2x2 matrix to a 6x6 matrix?

(Note that an additional "padding" can be used to address the shape issue for "conv1", but not for "conv2")

Best practice for CNN: up-sampling & conv transpose

https://towardsdatascience.com/understand-transposed-convolutions-and-build-your-own-transposed-convolution-layer-from-scratch-4f5d97b2967



In order to restore the shape from "2x2" to "6x6", first we need to understand that two activities made the shape changes:

- convolution
- max-pooling

In order to resume the shape from

- convolution: we can use padding or conv-tranposed
- max-pooling: we can use up-sampling

Best practice for CNN: up-sampling & conv transpose

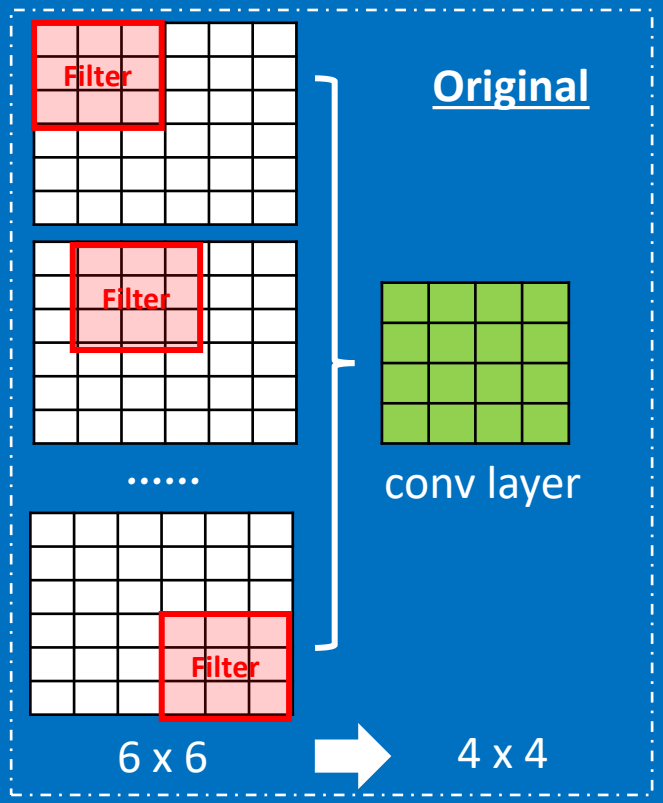
https://towardsdatascience.com/understand-transposed-convolutions-and-build-your-own-transposed-convolution-layer-from-scratch-4f5d97b2967

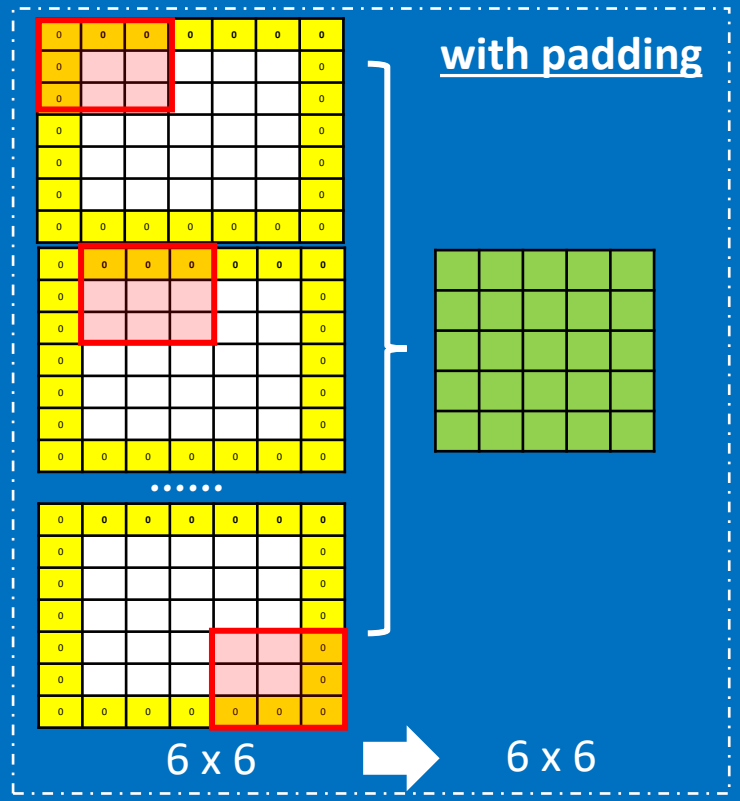
What is padding

In order to resume the shape from
- convolution: we can use padding, or conv-tranposed
- max-pooling: we can use up-sampling or conv-transpose

As we just discussed, the convolutional layers reduce the size of the output.

Padding basically extends the area of an image. The kernel/filter which moves across the image scans each pixel and converts the image into a smaller image. In order to work the kernel with processing in the image, padding is added to the outer frame of the image to allow for more space for the filter to cover in the image. Adding padding to an image processed by a CNN allows for a more accurate analysis of images.



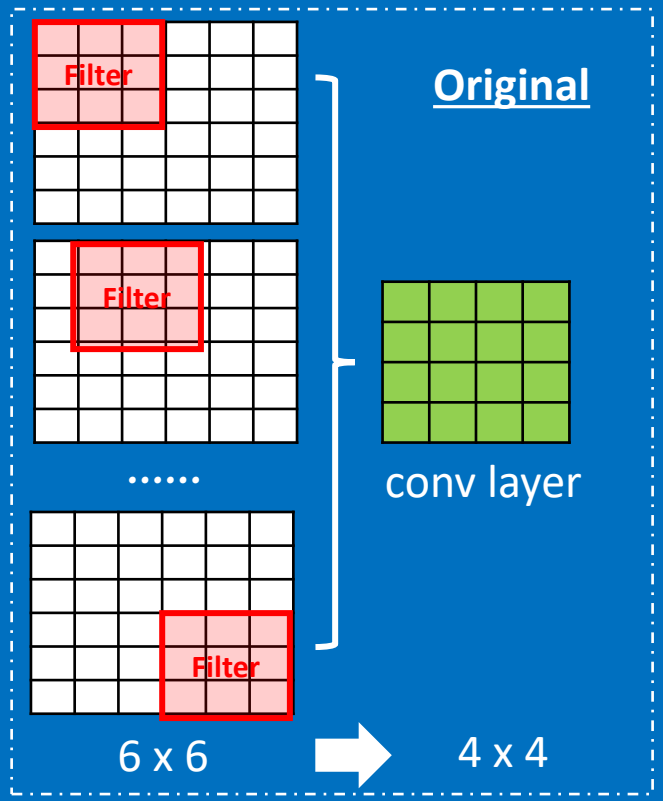


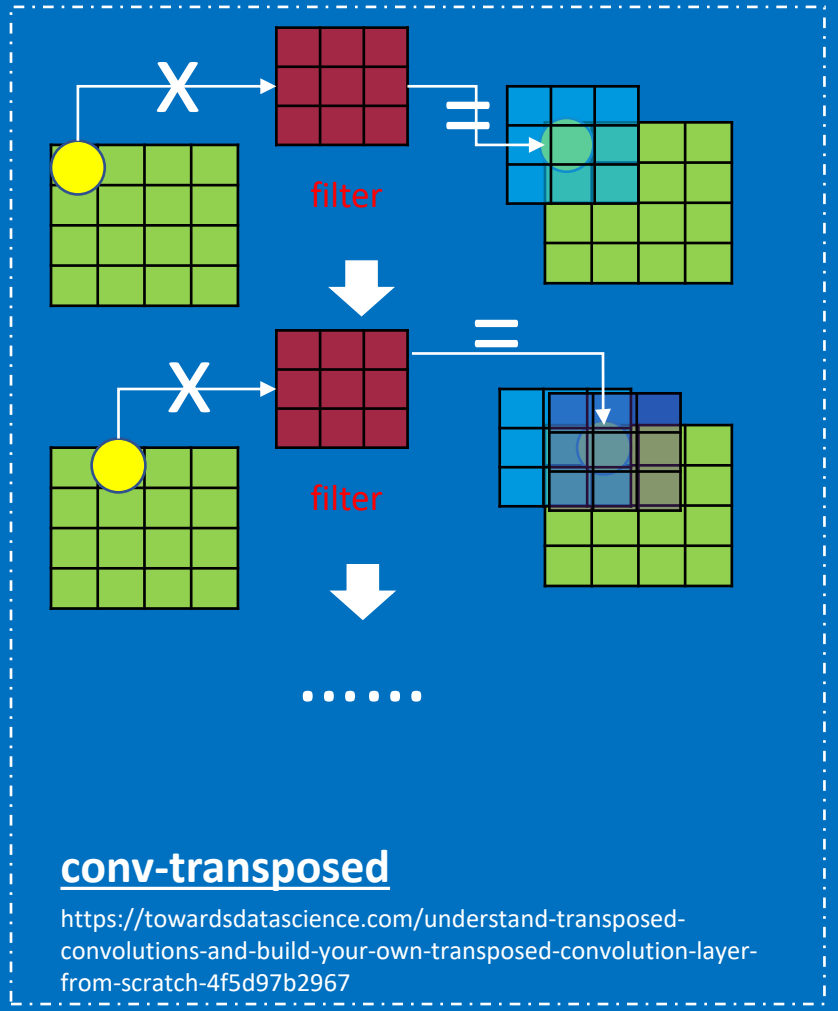
Best practice for CNN: up-sampling & conv transpose

https://towardsdatascience.com/understand-transposed-convolutions-and-build-your-own-transposed-convolution-layer-from-scratch-4f5d97b2967

What is conv-transposed

In order to resume the shape from
- convolution: we can use padding, or conv-tranposed
- max-pooling: we can use up-sampling or conv-transpose





Best practice for CNN: up-sampling & conv transpose

https://towardsdatascience.com/understand-transposed-convolutions-and-build-your-own-transposed-convolution-layer-from-scratch-4f5d97b2967

What is up-sampling

In order to resume the shape from
- convolution: we can use padding, or conv-tranposed
- max-pooling: we can use up-sampling

https://www.machinecurve.com/index.php/2019/12/11/upsampling2d-how-to-use-upsampling-with-keras/#what-is-upsampling

23	17
24	29

output after max-pooling

23	**21.5**	**18.5**	**17**
23.25	**22.44**	**20.81**	**20**
23.75	**24.31**	**25.43**	**26**
24	**25.25**	**27.75**	**29**

up-sampling

interpolation='bilinear'

23	23	17	17
23	23	17	17
24	24	29	29
24	24	29	29

up-sampling

interpolation=nearest'

Best practice for CNN: Why Dropout

https://stackoverflow.com/questions/59717290/does-maxpooling-reduce-overfitting