Deep Learning for everyone

Convolutional neural networks

https://sites.google.com/view/AIforEveryone

Learn once, apply anywhere

- We will learning Deep Learning with Computer Vision , but the same techniques can be applied for speech , text too

- Once you learn how to build neural networks for images, it doesn’t take much time for you to start working on other type such as audio etc

Credits

Friendly introduction to deep learning

How Convolutional Neural Networks work

- Brandon Rohrer

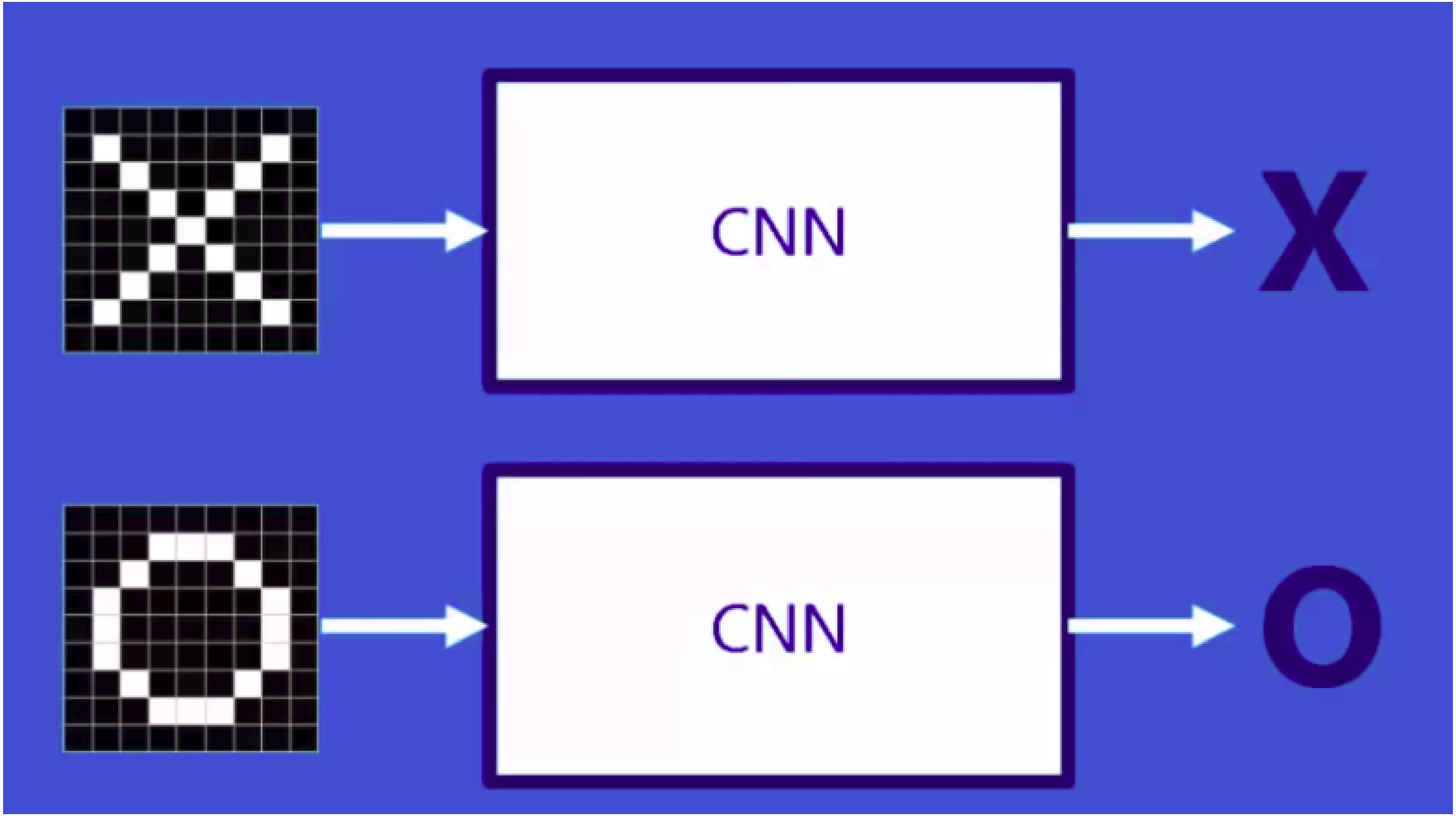
Credits

deeplearning.ai

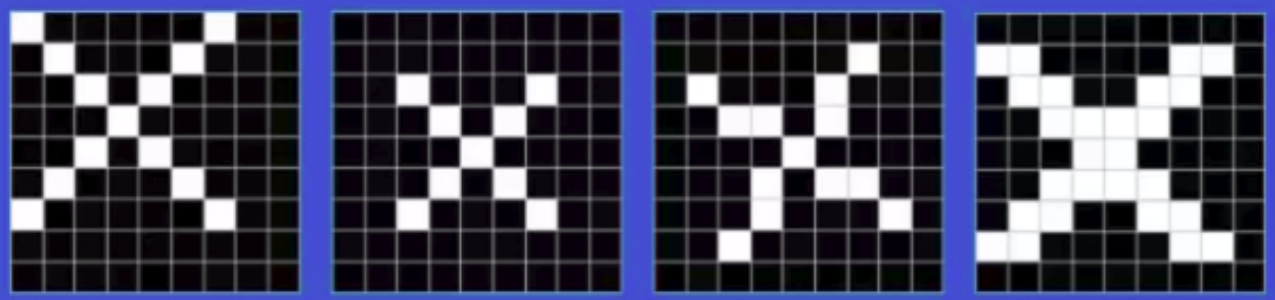
Acknowledgments & Credits are mentioned to inspirational resources presented in the end of presentation. For more like this, https://sites.google.com/view/AIforEveryone

Key takeaways

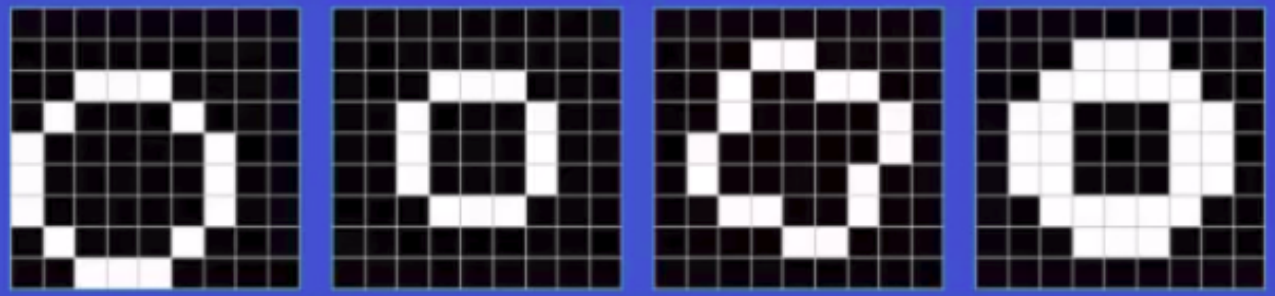
- Convnets are the best type of machine-learning models for computer-vision tasks. It's possible to train one from scratch even on a very small dataset, with decent results.
- On a small dataset, overfitting will be the main issue. Data augmentation is a powerful way to fight overfitting when you're working with image data.
- It's easy to reuse an existing convnet on a new dataset via feature extraction. This is a valuable technique for working with small image datasets.
- As a complement to feature extraction, you can use fine-tuning, which adapts to a new problem some of the representations previously learned by an existing model. This pushes performance a bit further.


CNN
X
CNN
O

Trickier cases



translation scaling rotation weight



CNN

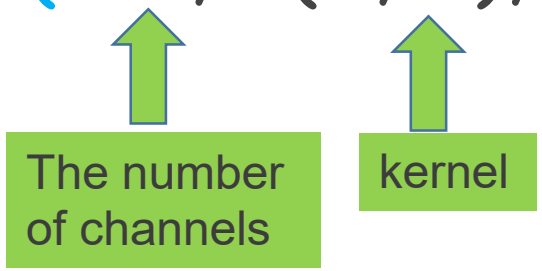
X

CNN

O

Keras code

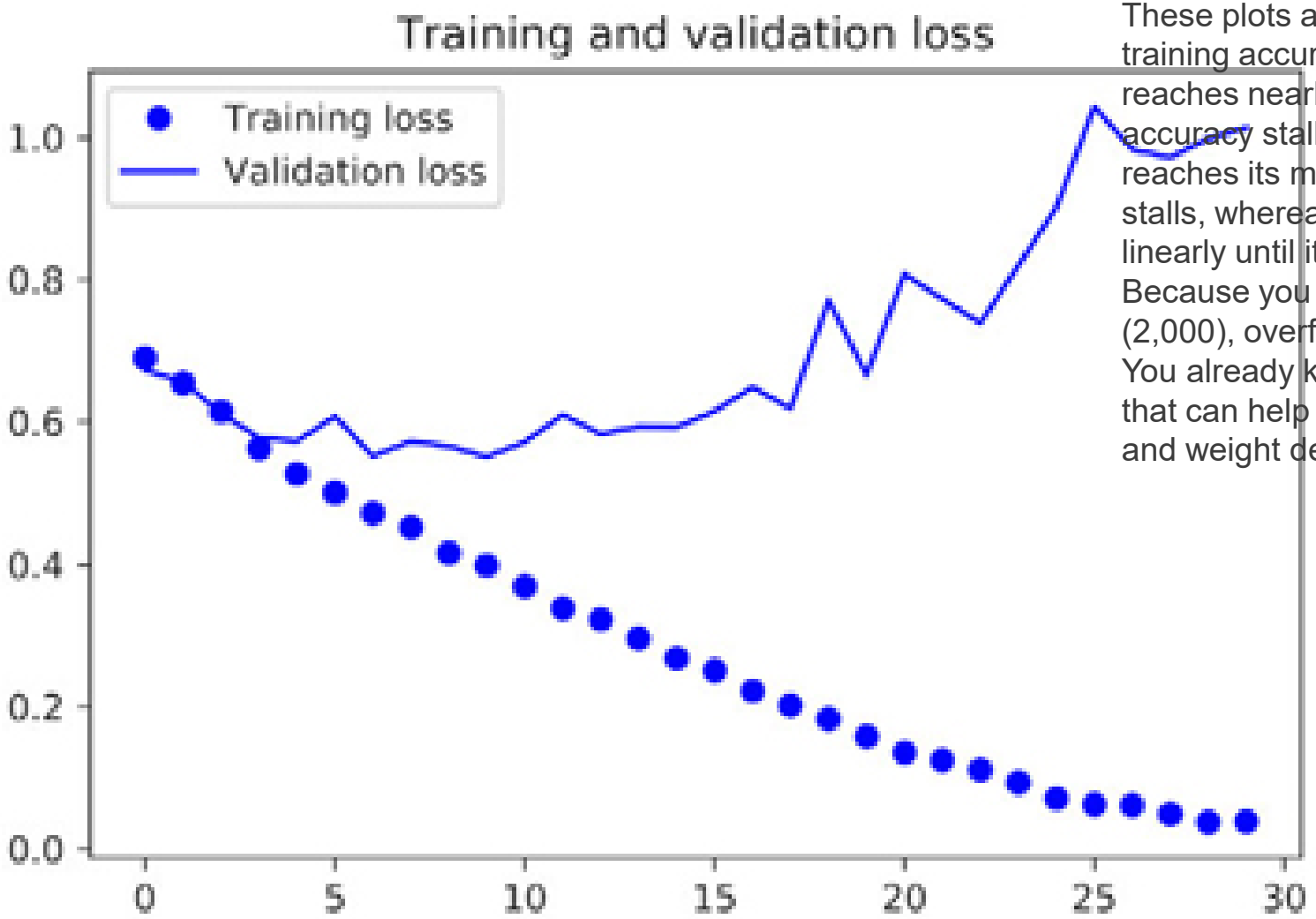
```
layers.Conv2D( 64,  (3, 3),  activation='relu')
```



A small covnet

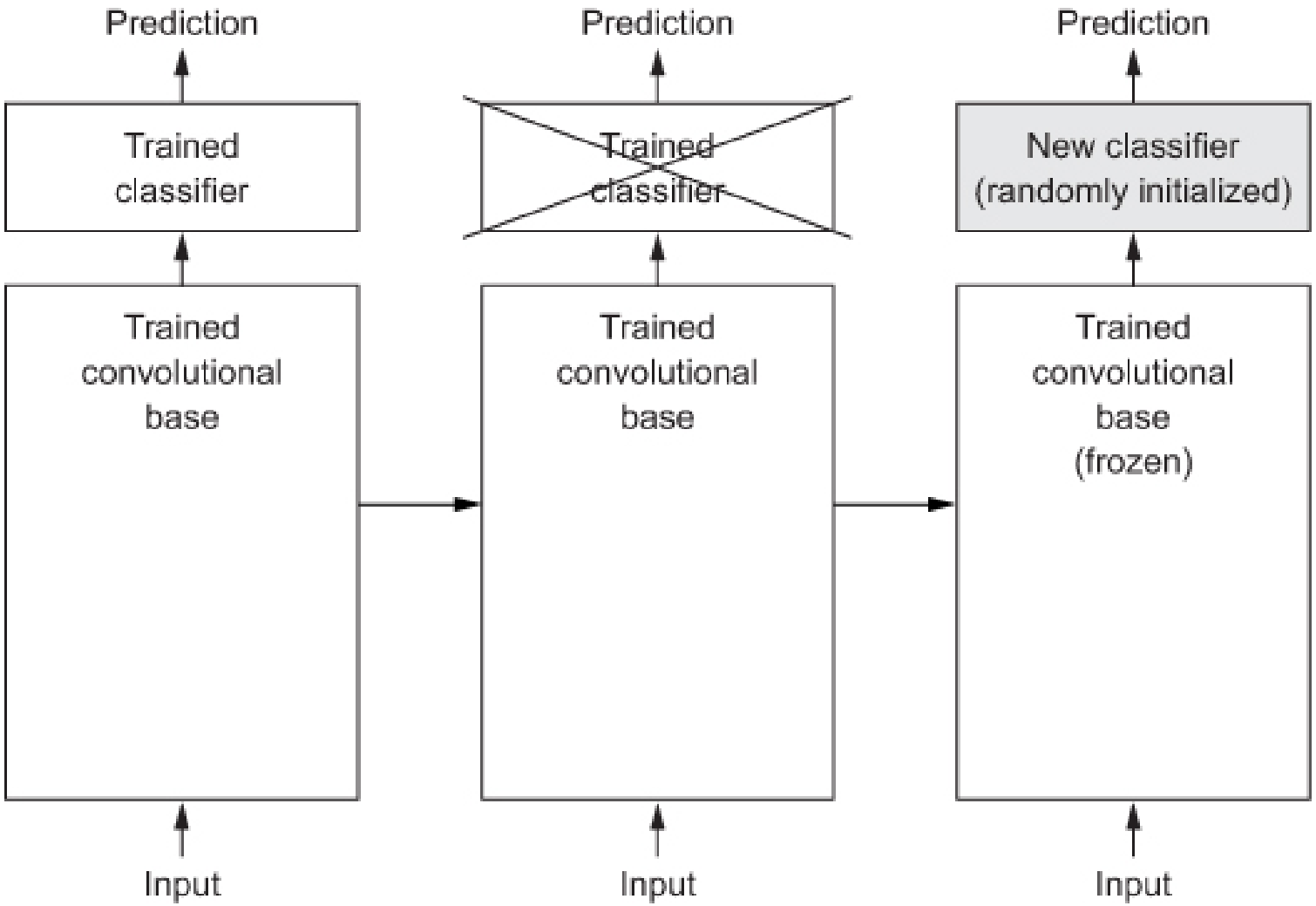
```
from keras import layers
from keras import models
myModel = models.Sequential()

myModel. add (layers.Conv2D(32, (3, 3))
myModel. add (layers .MaxPooling2D() )
myModel. add (layers .Conv2D(64, (3, 3)) )
myModel. add (layers .MaxPooling2D())
myModel. add (layers .Conv2D(64, (3, 3)) )
```



These plots are characteristic of overfitting. The training accuracy increases linearly over time, until it reaches nearly 100%, whereas the validation accuracy stalls at 70–72%. The validation loss reaches its minimum after only five epochs and then stalls, whereas the training loss keeps decreasing linearly until it reaches nearly 0.
Because you have relatively few training samples (2,000), overfitting will be your number-one concern. You already know about a number of techniques that can help mitigate overfitting, such as dropout and weight decay (L2 regularization)

Swapping classifiers while keeping the same convolutional base



- The reason is that the representations learned by the convolutional base are likely to be more generic and therefore more reusable: the feature maps of a convnet are presence maps of generic concepts over a picture, which is likely to be useful regardless of the computer-vision problem at hand

- Layers that come earlier in the model extract local, highly generic feature maps (such as visual edges, colors, and textures), whereas layers that are higher up extract more-abstract concepts (such as “cat ear” or “dog eye”). So if your new dataset differs a lot from the dataset on which the original model was trained, you may be better off using only the first few layers of the model to do feature extraction, rather than using the entire convolutional base

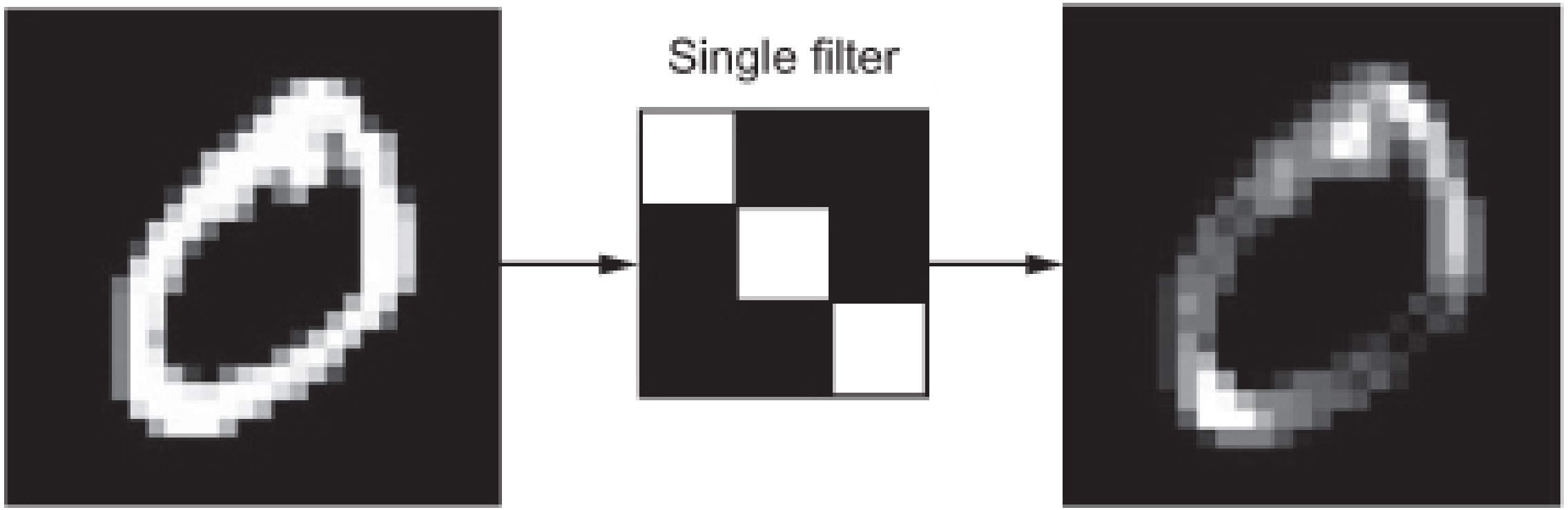
Here’s the list of image-classification mode (all pretrained on the ImageNet dataset) that are available as part of `keras.applicatio`

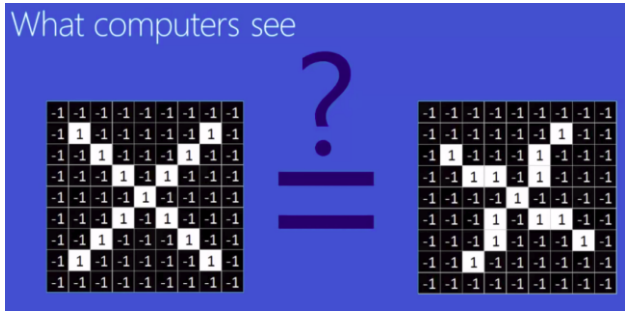
- Xception
- Inception V3
- ResNet50
- VGG16
- VGG19
- MobileNet

Original input

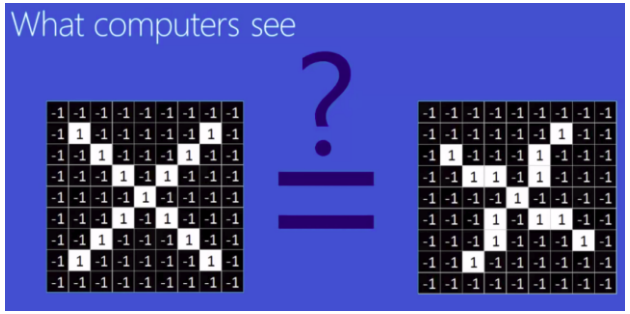
Single filter

Response map, quantifying the presence of the filter's pattern at different locations



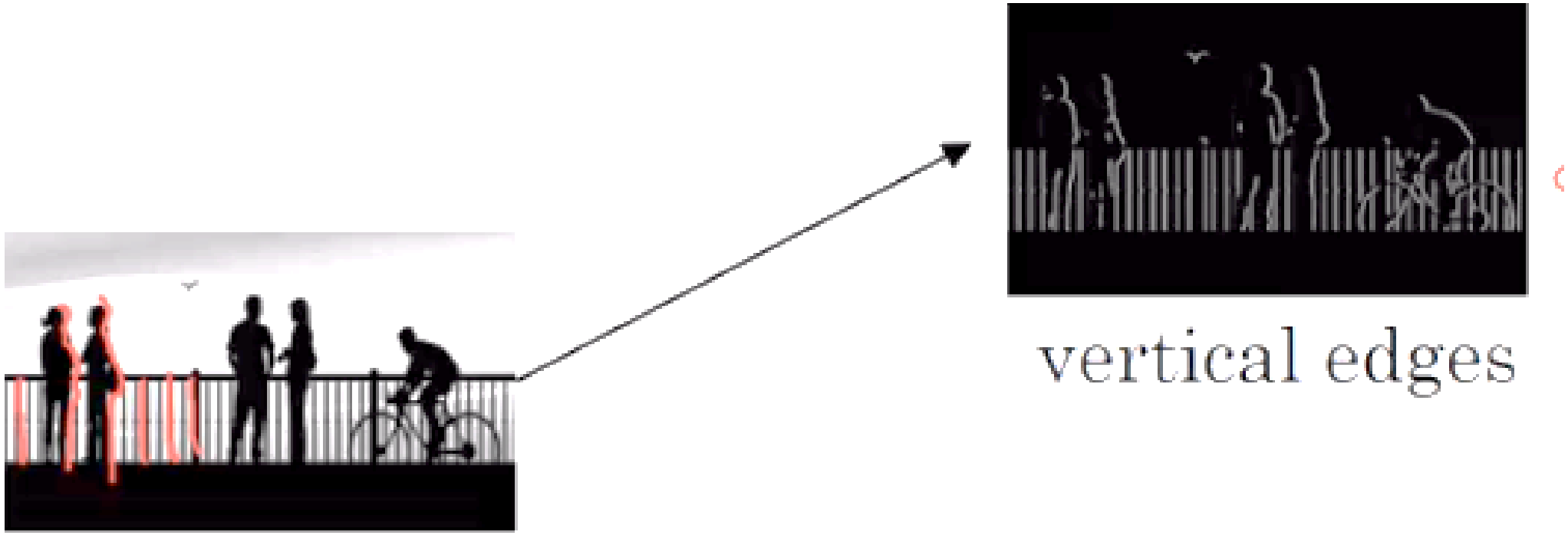


Acknowledgments & Credits are mentioned to inspirational resources presented in the end of presentation. For more like this, https://sites.google.com/view/AIforEveryone



Acknowledgments & Credits are mentioned to inspirational resources presented in the end of presentation. For more like this, https://sites.google.com/view/AIforEveryone

Vertical edge detection



vertical edges

Vertical edge detection

$3\times1 + 1\times1 + 2\times1 + 0\times0 + 5\times0 + 7\times0 + 1\times-1 + 8\times-1 + 2\times-1 = -5$

3 (1)	0 (0)	1 (-1)	2	7	4
1 (1)	5 (0)	8 (-1)	9	3	1
2 (1)	7 (0)	2 (-1)	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

6×6

"convolution"

*

1	0	-1
1	0	-1
1	0	-1

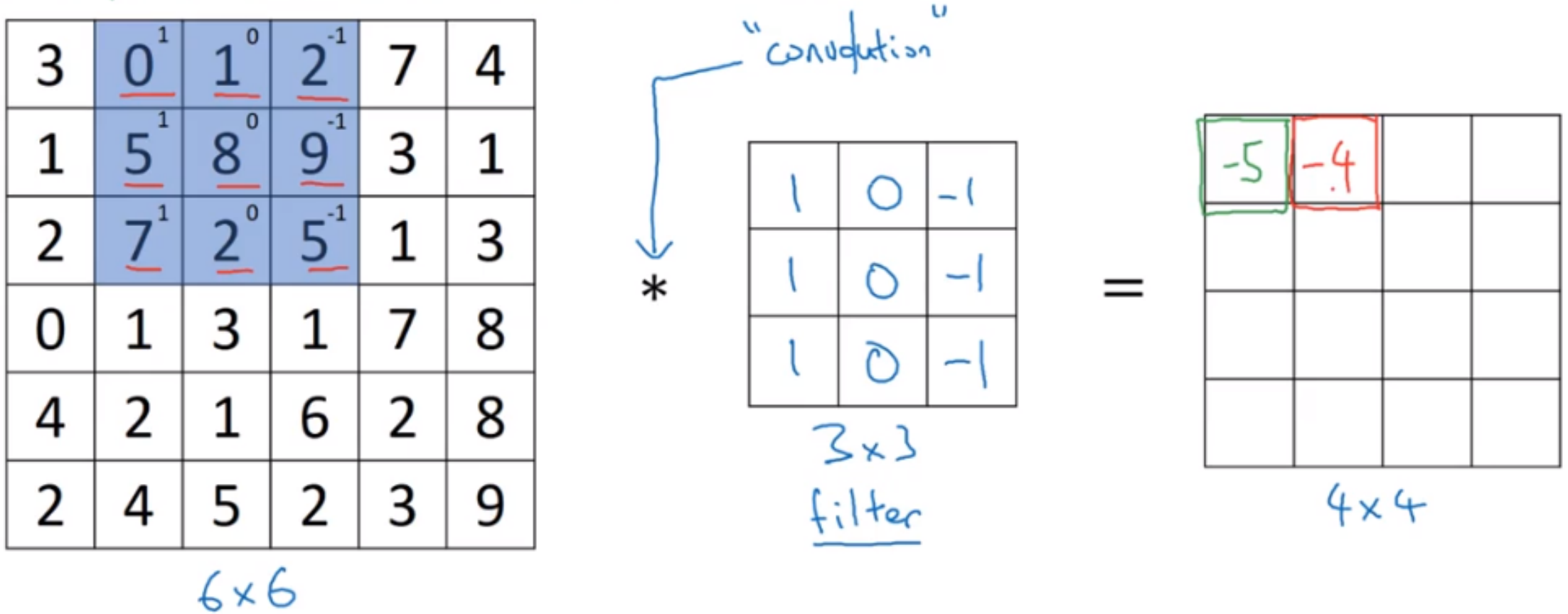
3×3

filter

=

-5			

4×4



Vertical edge detection

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

6x6

*

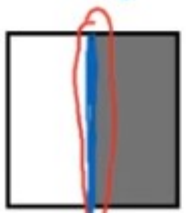
1	0	-1
1	0	-1
1	0	-1

3x3

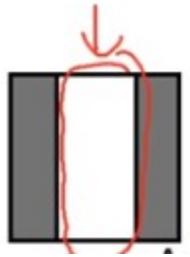
=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

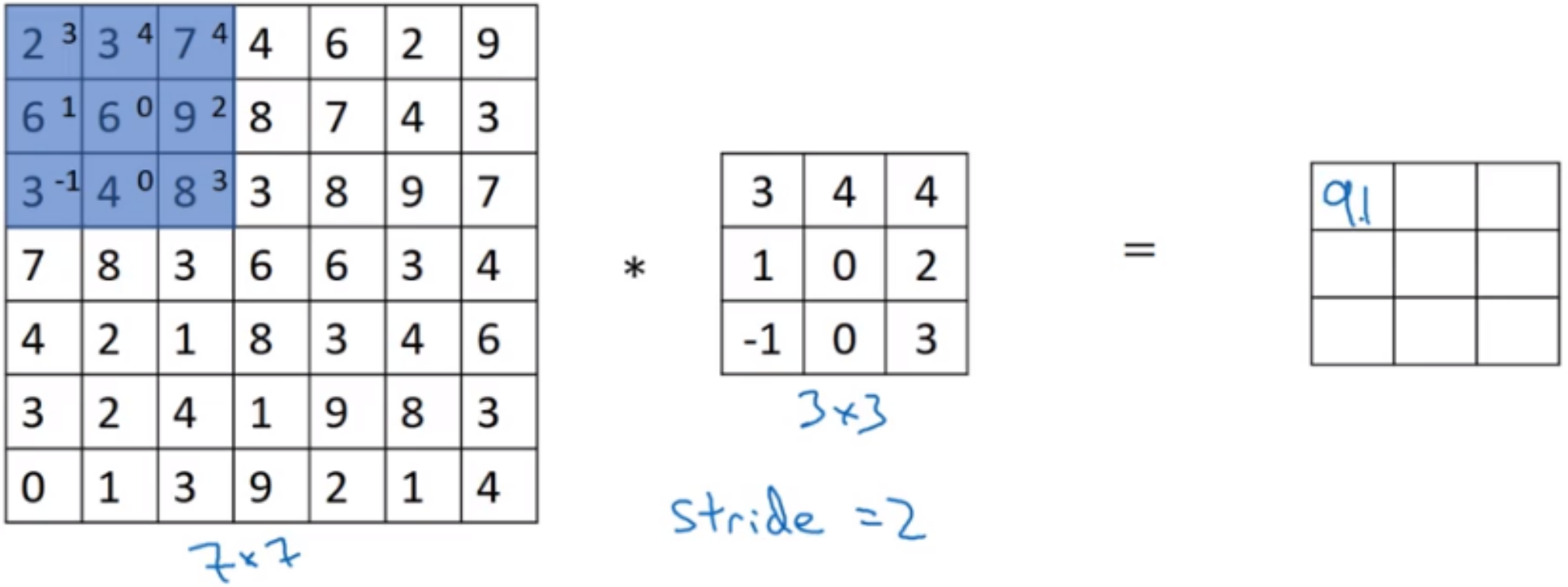
4x4



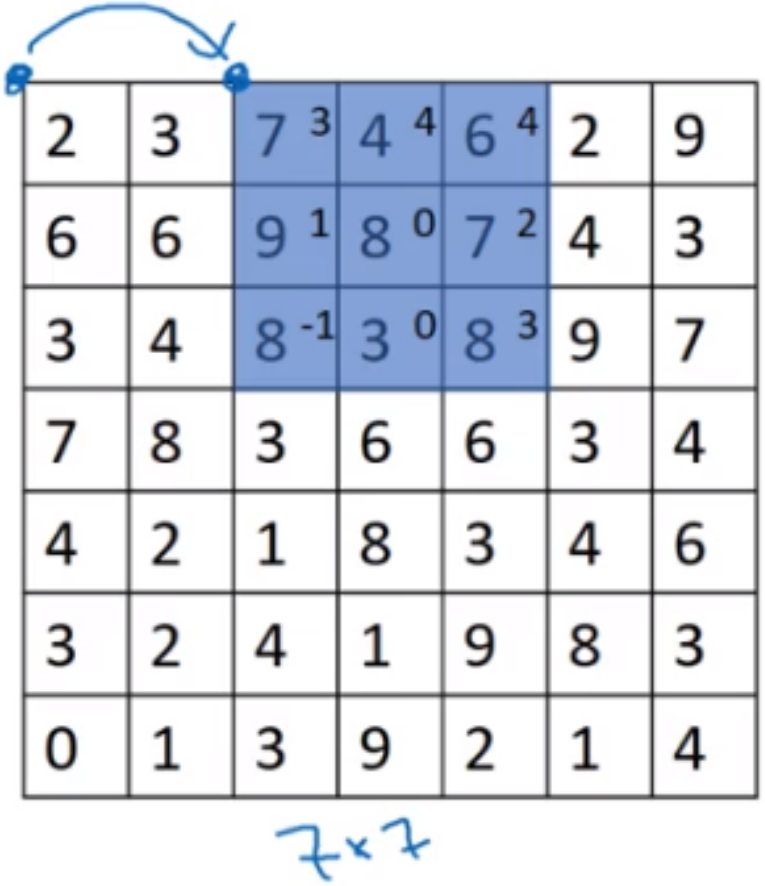
*



Strided convolution



Strided convolution



*

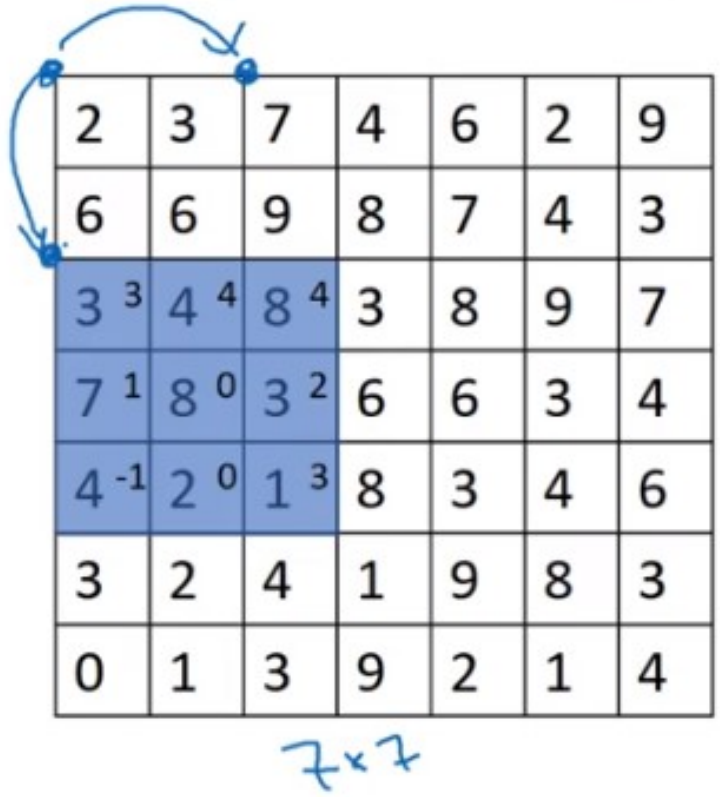
3	4	4
1	0	2
-1	0	3

3×3

stride = 2

=

91	100	



2	3	7	4	6	2	9
6	6	9	8	7	4	3
3^{3}	4^{4}	8^{4}	3	8	9	7
7^{1}	8^{0}	3^{2}	6	6	3	4
4^{-1}	2^{0}	1^{3}	8	3	4	6
3	2	4	1	9	8	3
0	1	3	9	2	1	4

7×7

*

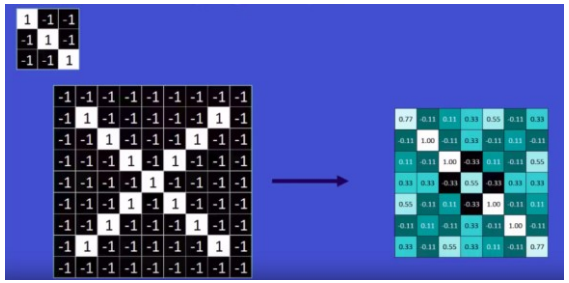
3	4	4
1	0	2
-1	0	3

3×3

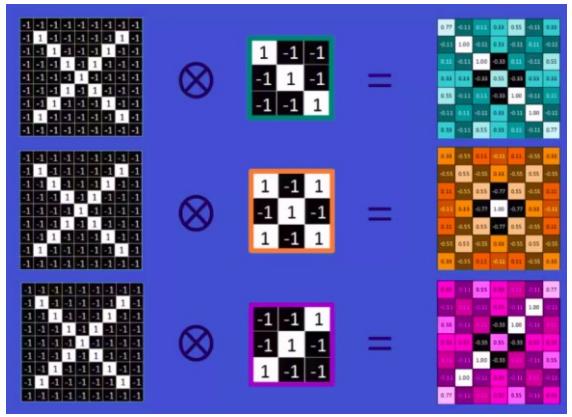
stride = 2

=

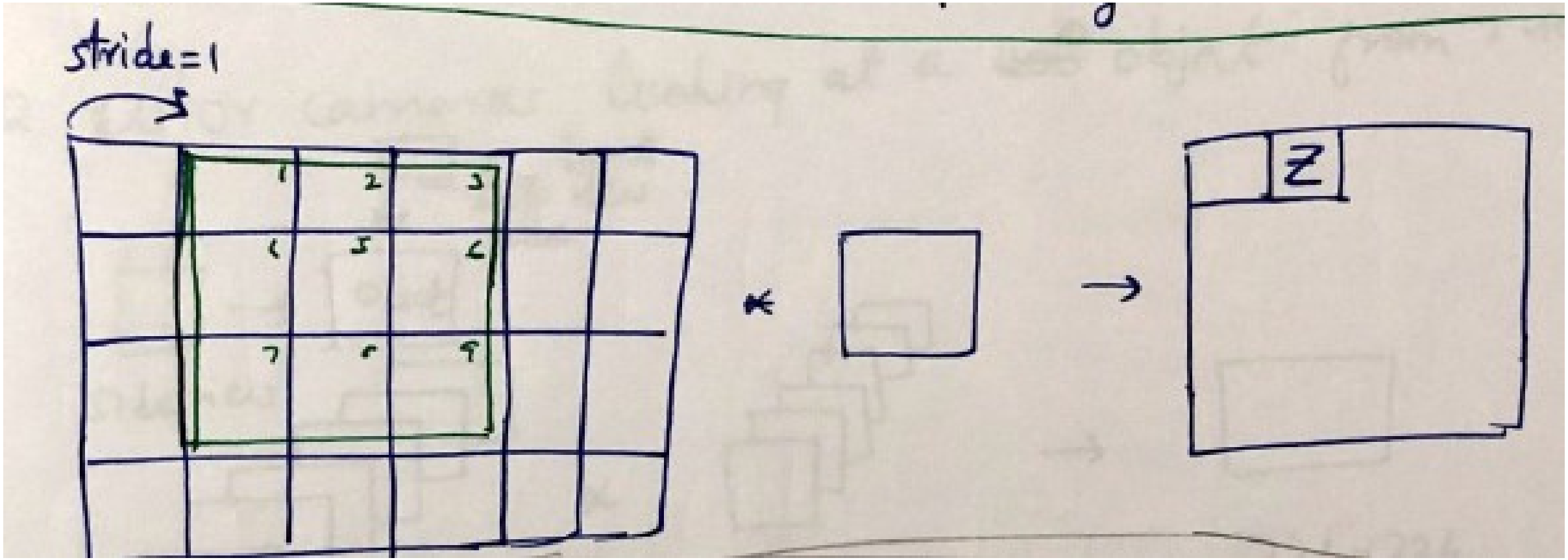
91	100	83

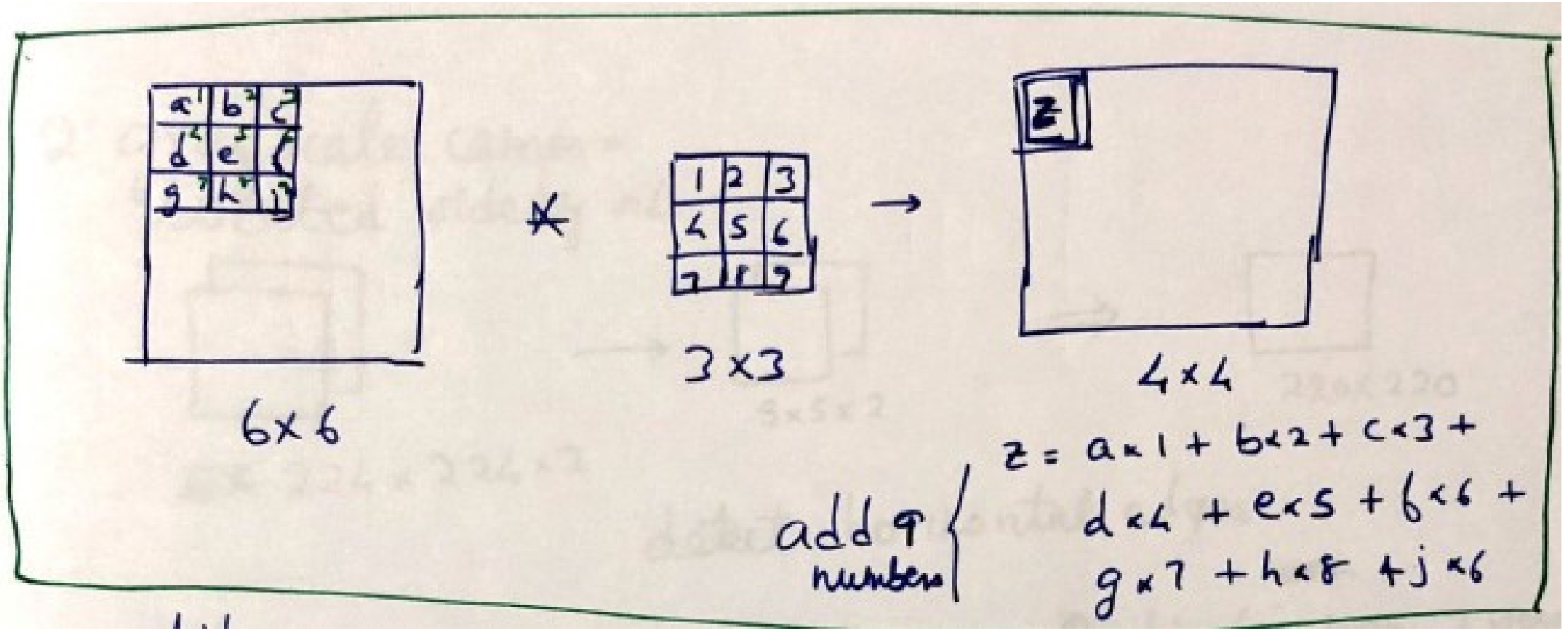


Acknowledgments & Credits are mentioned to inspirational resources presented in the end of presentation. For more like this, https://sites.google.com/view/AIforEveryone

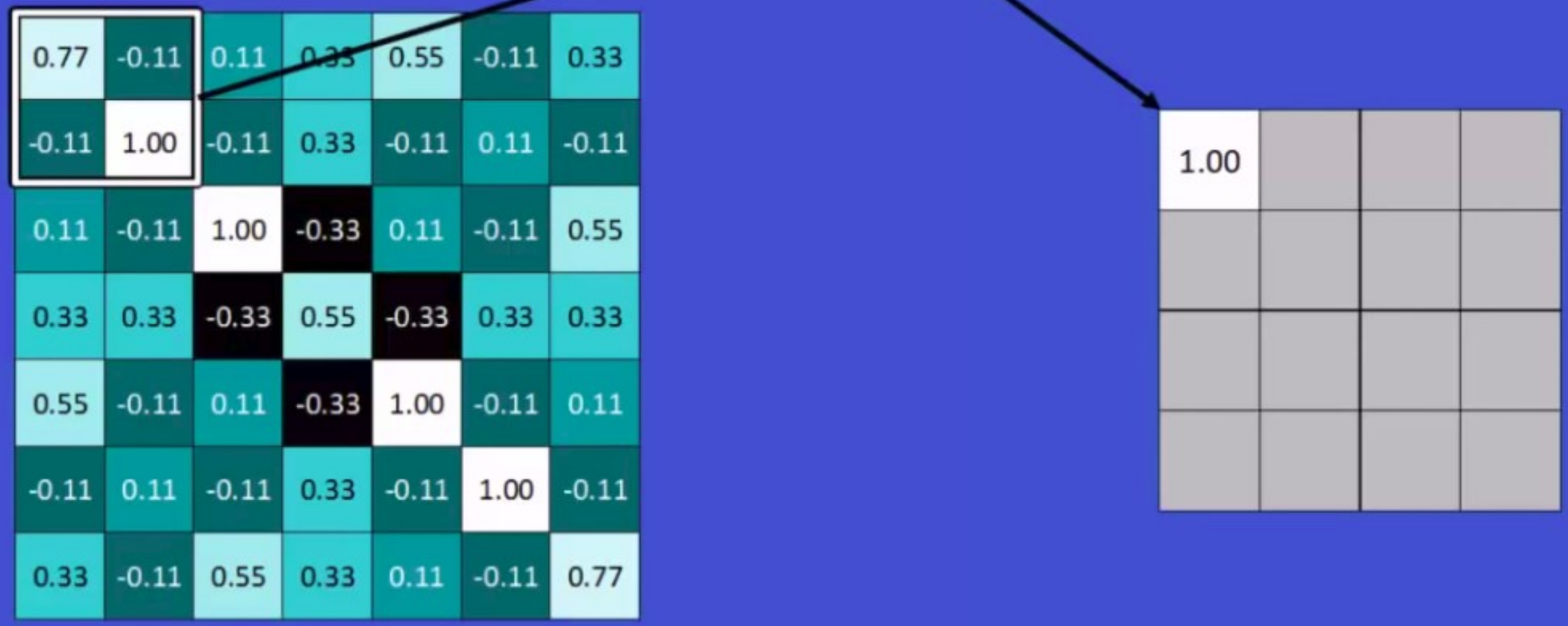


Acknowledgments & Credits are mentioned to inspirational resources presented in the end of presentation. For more like this, https://sites.google.com/view/AIforEveryone

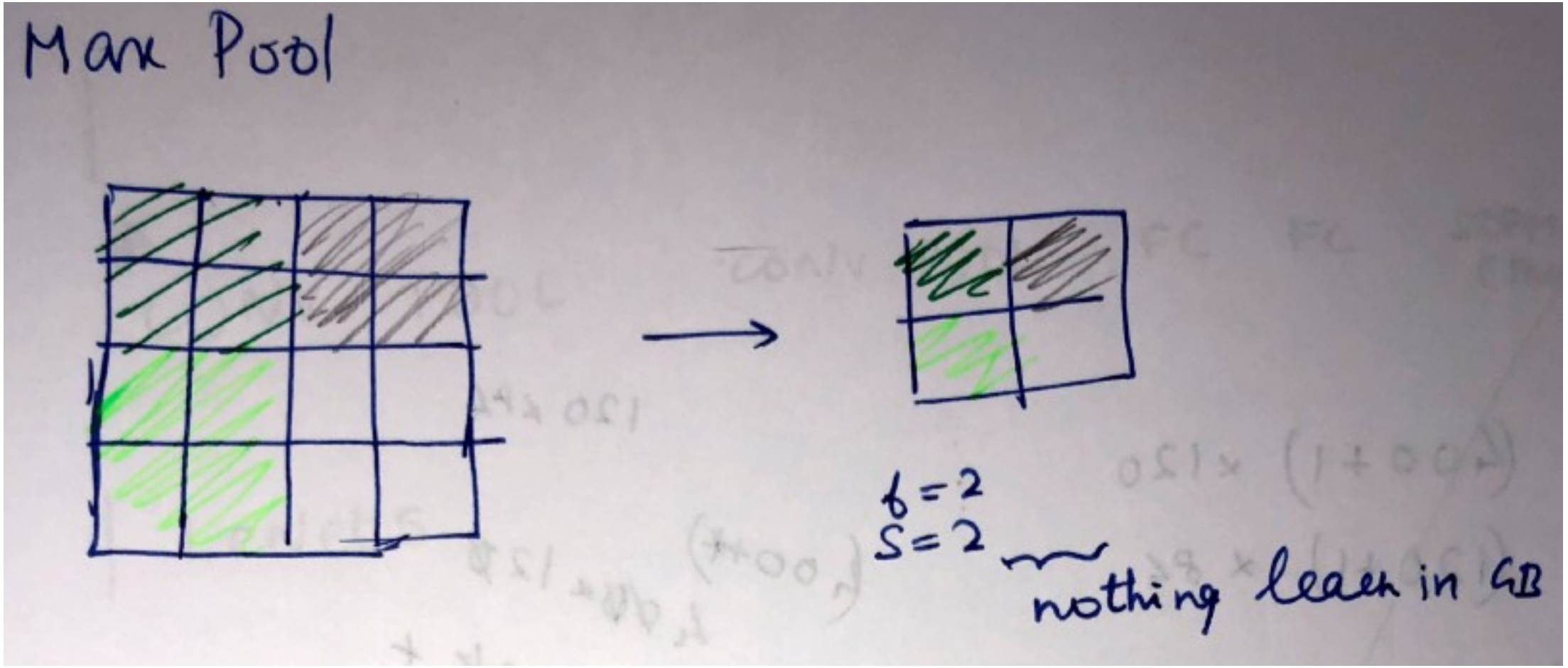




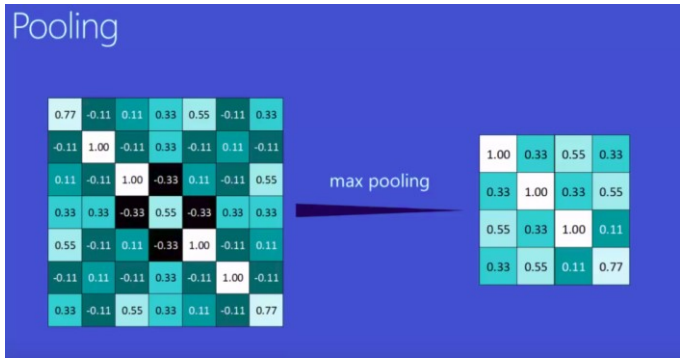
Pooling



Max pooling



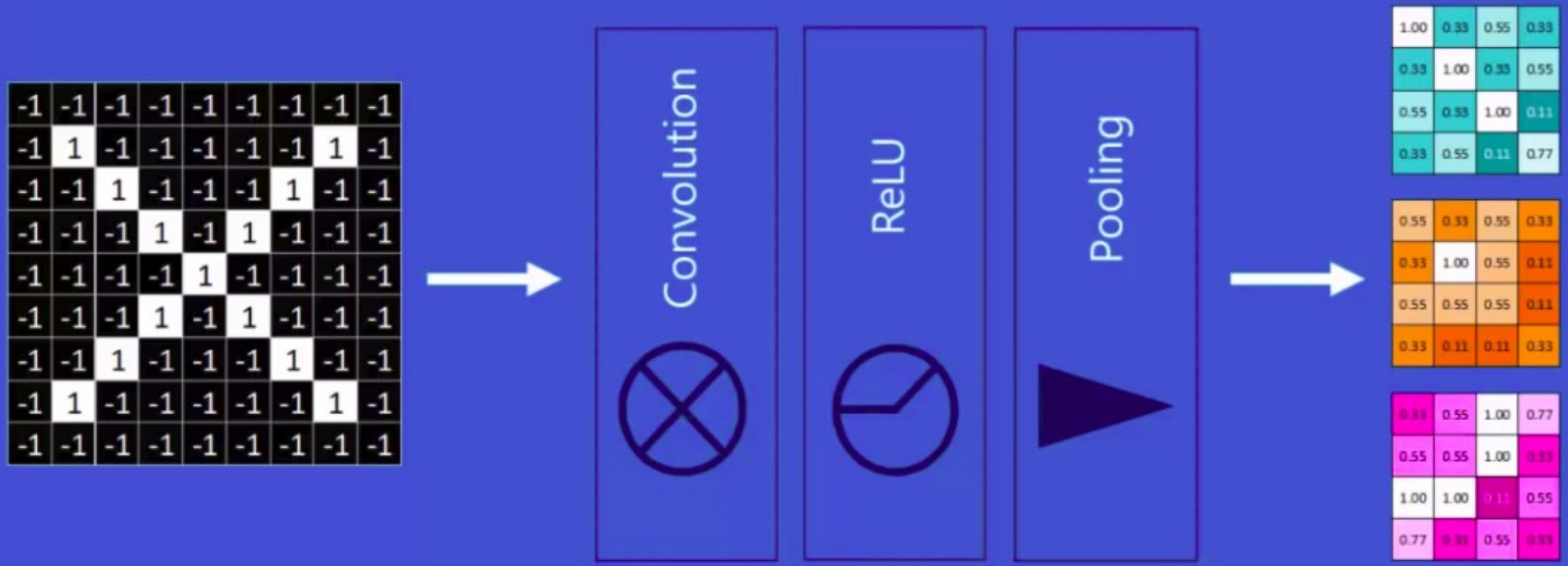
Same pattern is maintained, but smaller representation



Rectified Linear Units (ReLUs)

Layers get stacked

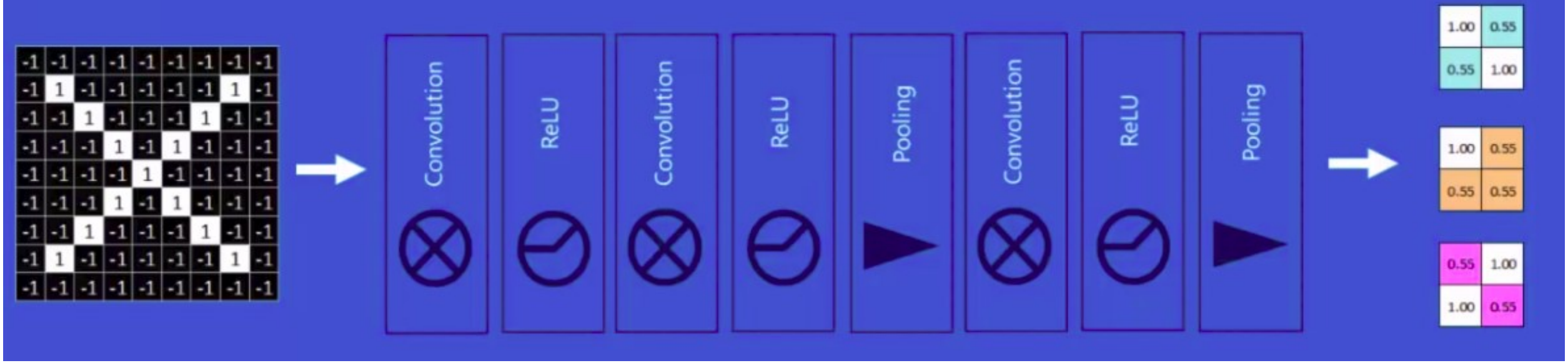
The output of one becomes the input of the next.



Acknowledgments & Credits are mentioned to inspirational resources presented in the end of presentation. For more like this, https://sites.google.com/view/AIforEveryone

Deep stacking

Layers can be repeated several (or many) times.

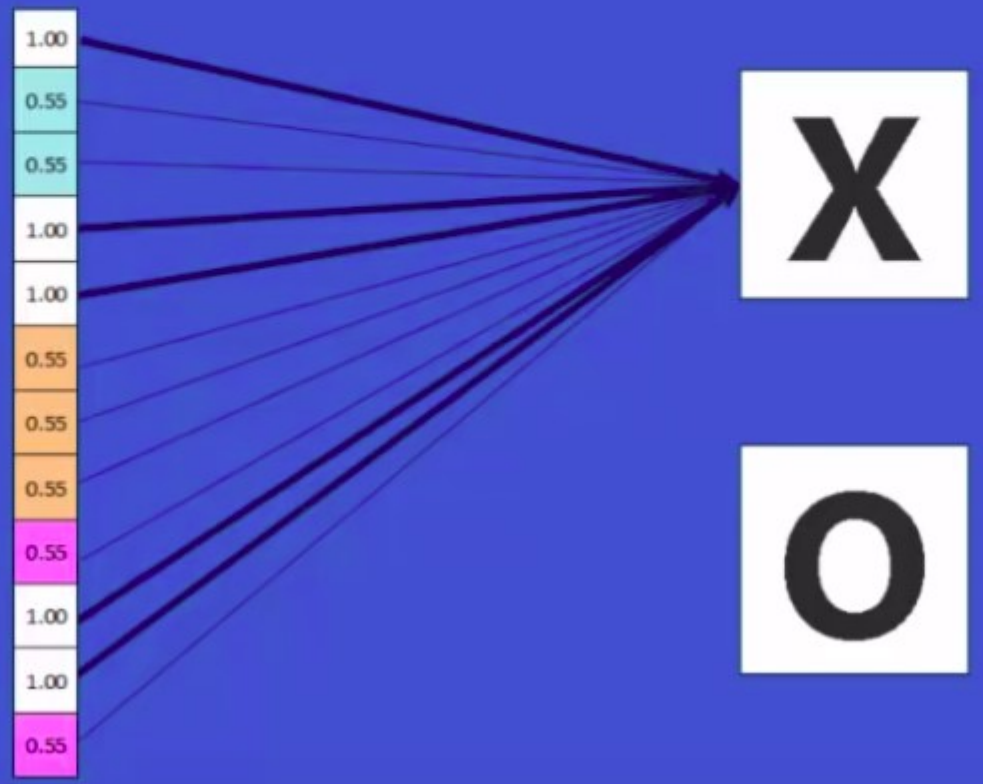


Fully connected layer

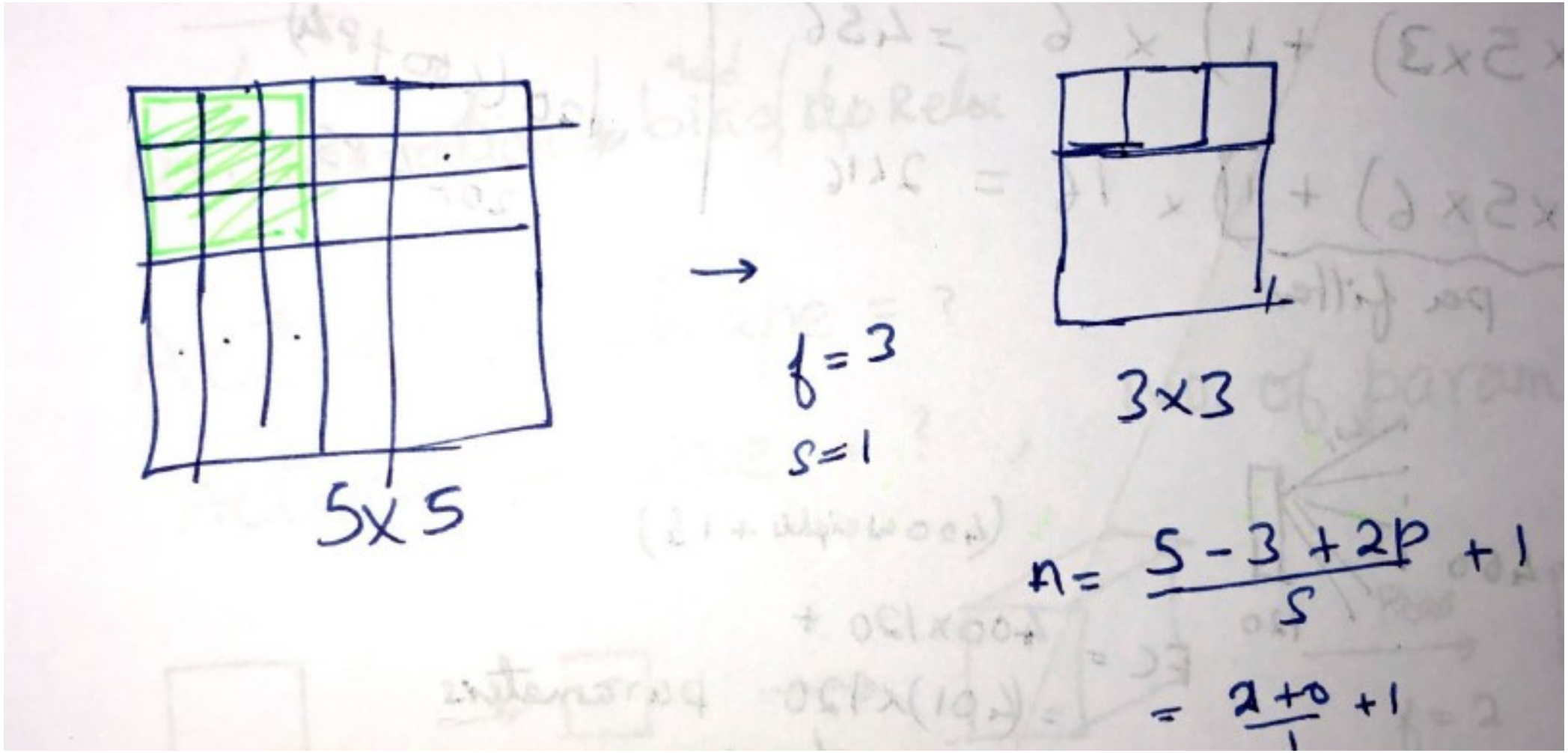
Every value gets a vote

Fully connected layer

Vote depends on how strongly a value predicts X or O



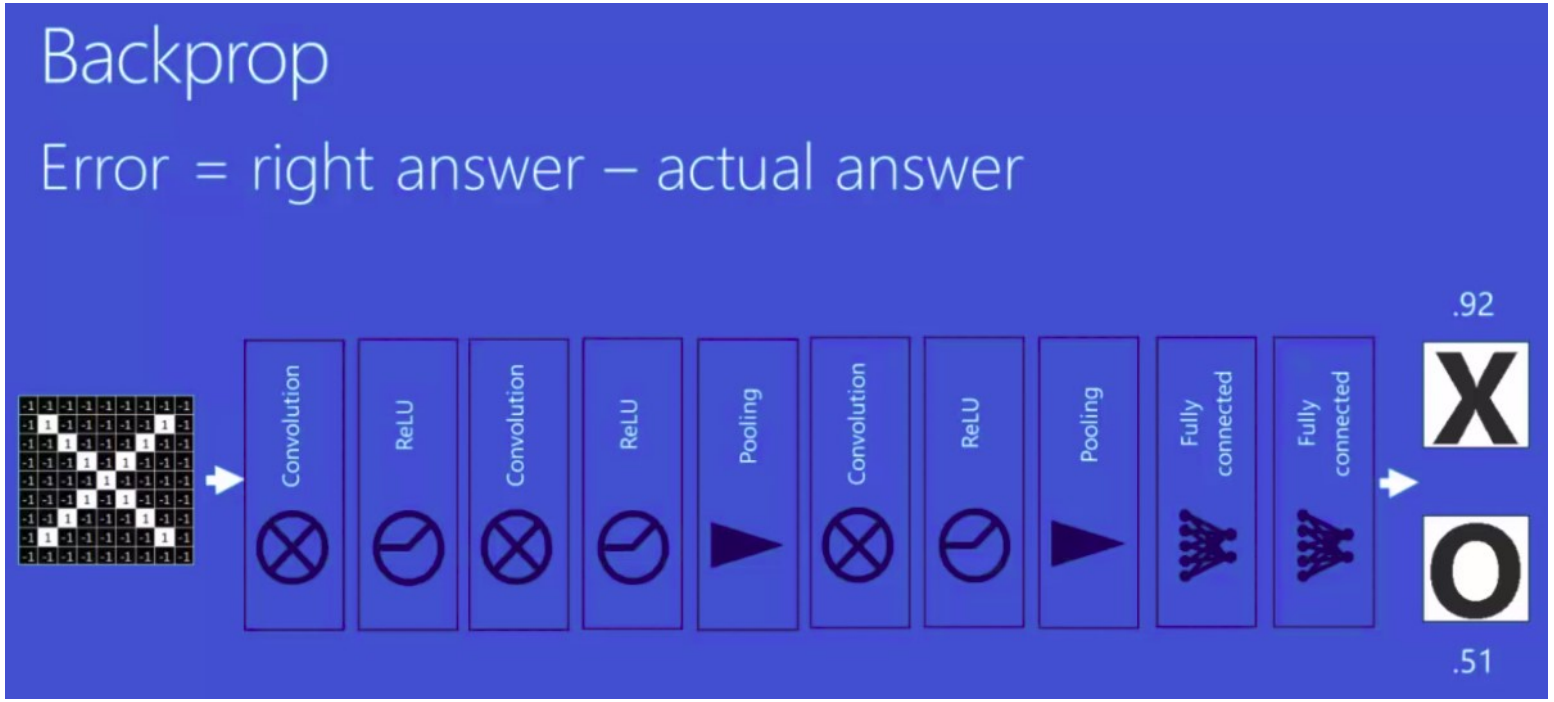
Max pooling



Backprop

Error = right answer – actual answer

Error



	Right answer	Actual answer	Error
X	1	0.92	0.08
O	0	0.51	0.49
		Total	0.57

5. Train the network

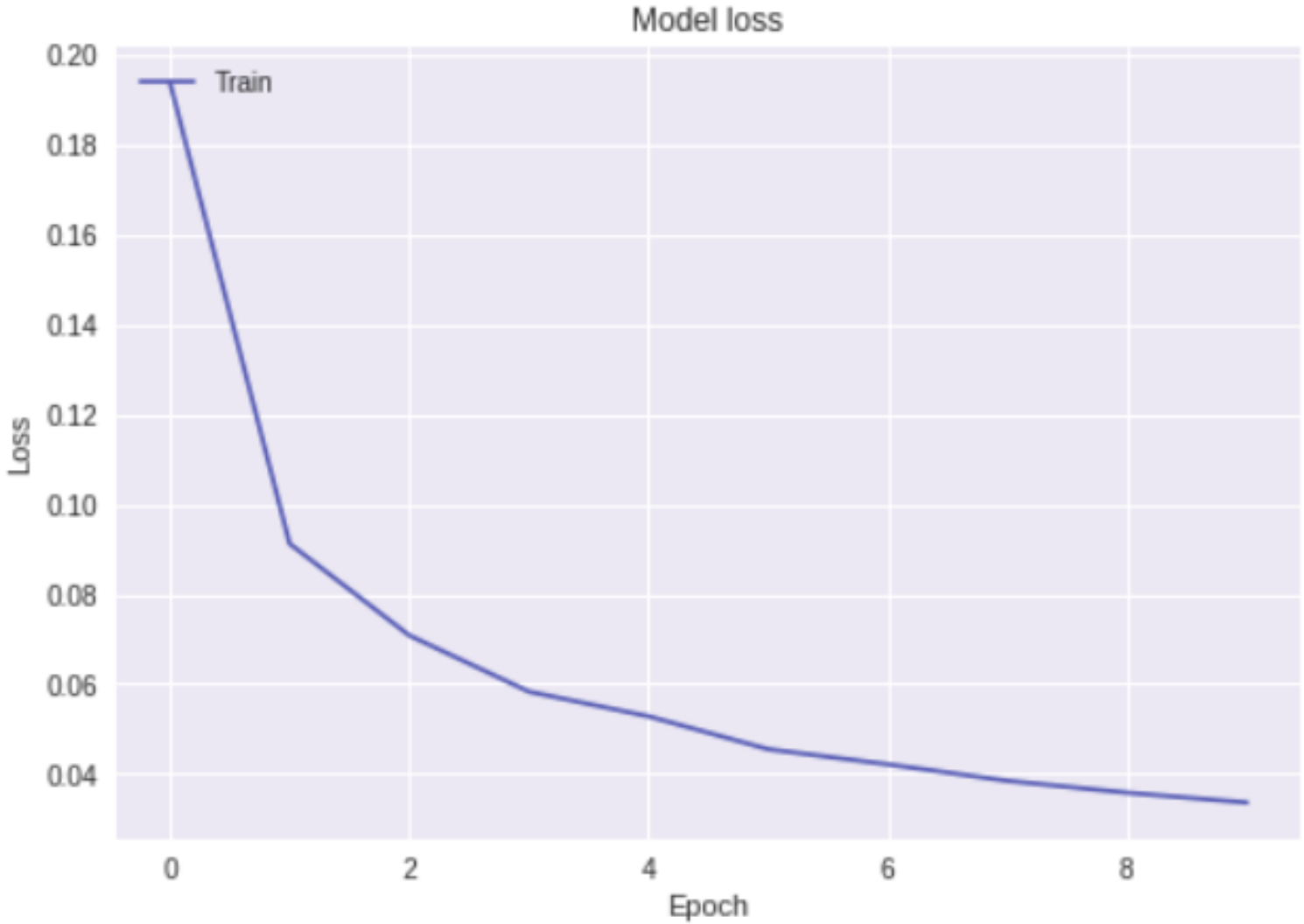
Tip 1 : myModel.fit(X, Y) trains the neural network with dataset (X, Y).

It returns a log history, which is a log of values in each iteration

]:
```
#Step 5: Train the network
history = network.fit( trainXready, trainYready, batch_size=32, epochs=10)
```

```
Epoch 1/10
60000/60000 [==============================] - 14s 228us/step - loss: 0.1939 - acc: 0.9418
Epoch 2/10
60000/60000 [==============================] - 13s 223us/step - loss: 0.0912 - acc: 0.9745
Epoch 3/10
60000/60000 [==============================] - 13s 215us/step - loss: 0.0708 - acc: 0.9814
Epoch 4/10
60000/60000 [==============================] - 13s 216us/step - loss: 0.0583 - acc: 0.9849
Epoch 5/10
60000/60000 [==============================] - 13s 218us/step - loss: 0.0528 - acc: 0.9872
Epoch 6/10
60000/60000 [==============================] - 13s 219us/step - loss: 0.0455 - acc: 0.9891
Epoch 7/10
60000/60000 [==============================] - 13s 216us/step - loss: 0.0421 - acc: 0.9910
Epoch 8/10
60000/60000 [==============================] - 13s 212us/step - loss: 0.0384 - acc: 0.9917
Epoch 9/10
60000/60000 [==============================] - 13s 217us/step - loss: 0.0358 - acc: 0.9925
Epoch 10/10
60000/60000 [==============================] - 13s 218us/step - loss: 0.0336 - acc: 0.9934
```

Error over iterations



Hyperparameters (knobs)

- Convolution
  - Number of features
  - Size of features
- Pooling
  - Window size
  - Window stride
- Fully Connected
  - Number of neurons

Architecture

How many of each type of layer?

In what order?

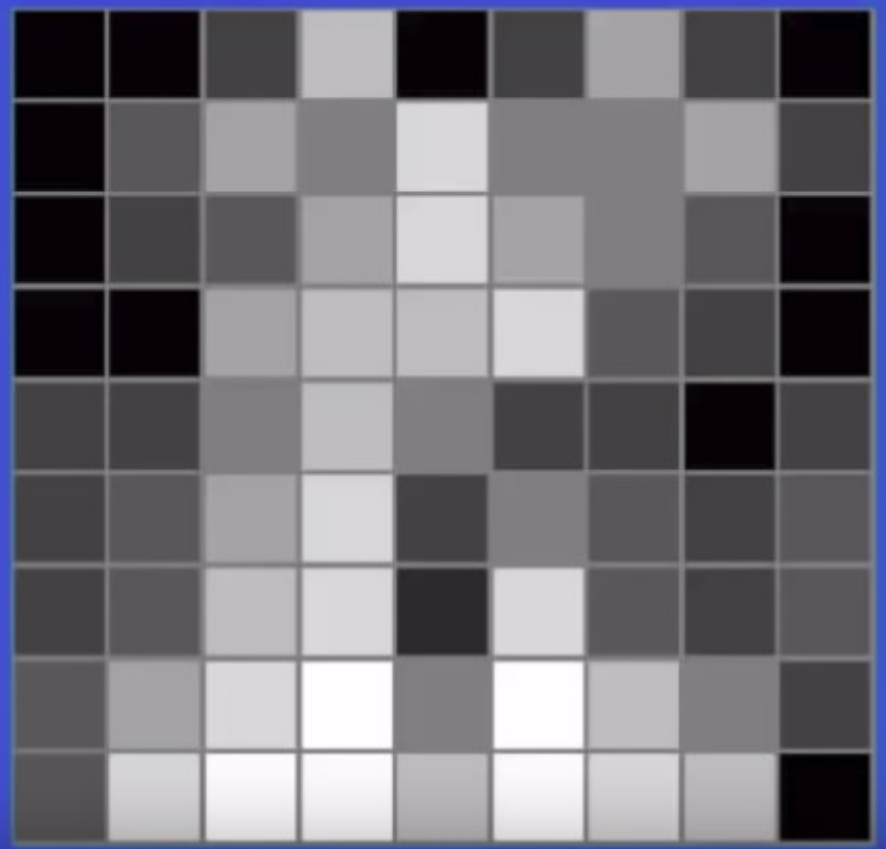
Not just images

Things closer together are more closely related than things far away.

Sound

Time steps →

Intensity in each frequency band ↓



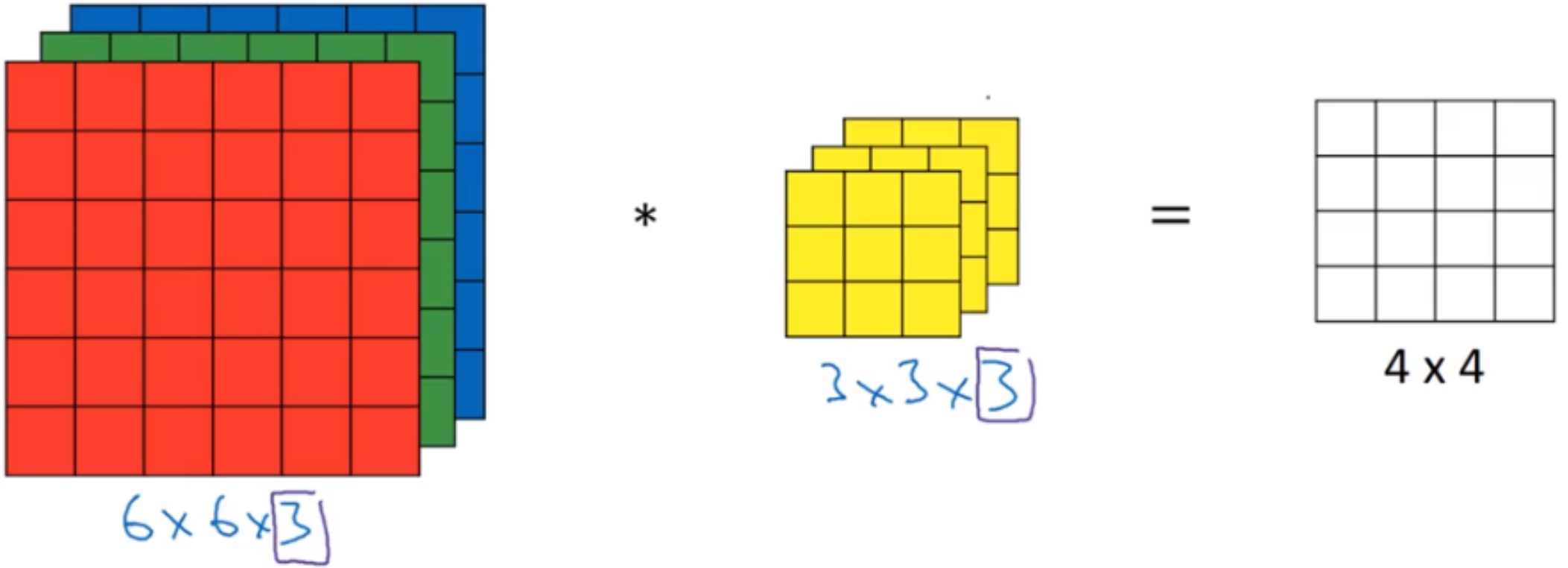
Limitations

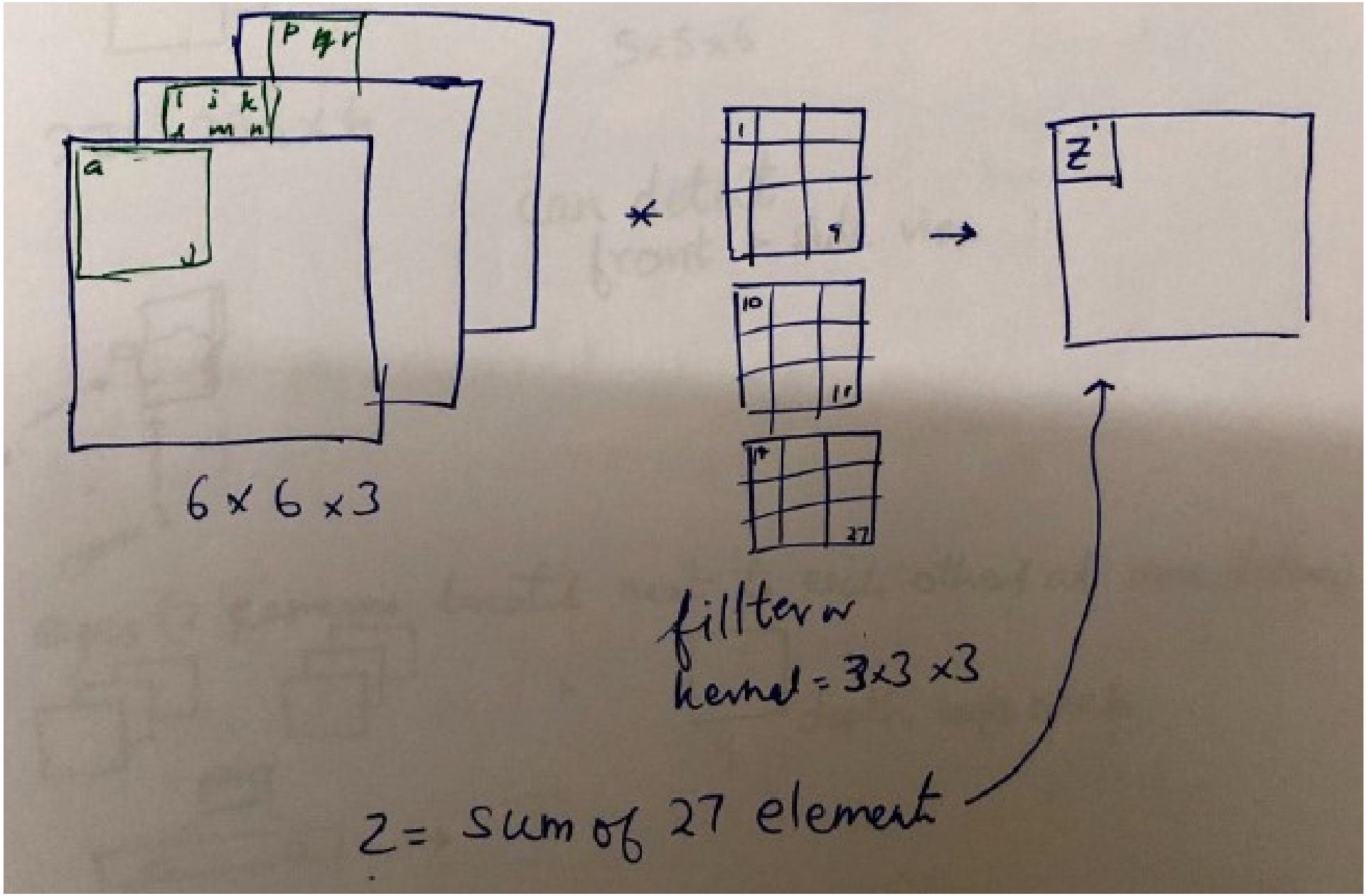
ConvNets only capture local "spatial" patterns in data.

If the data can't be made to look like an image, ConvNets are less useful.

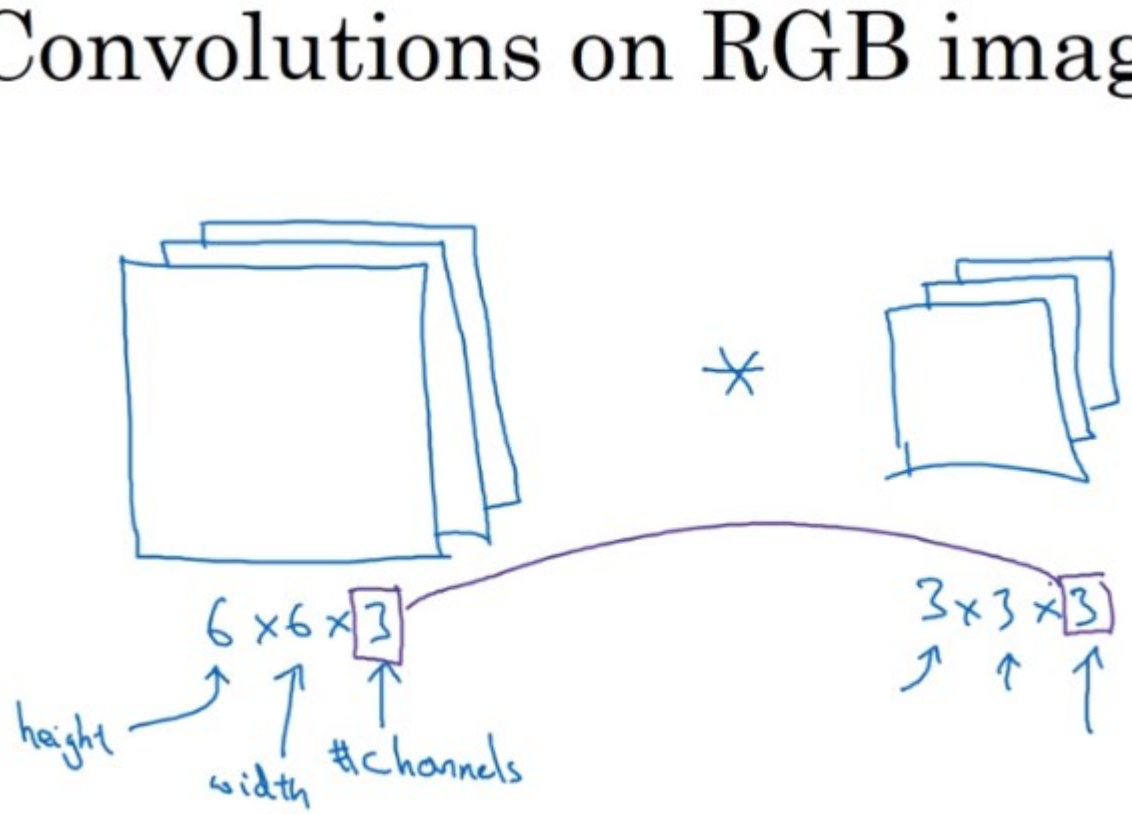
Convolutions over volume

Convolutions on RGB image

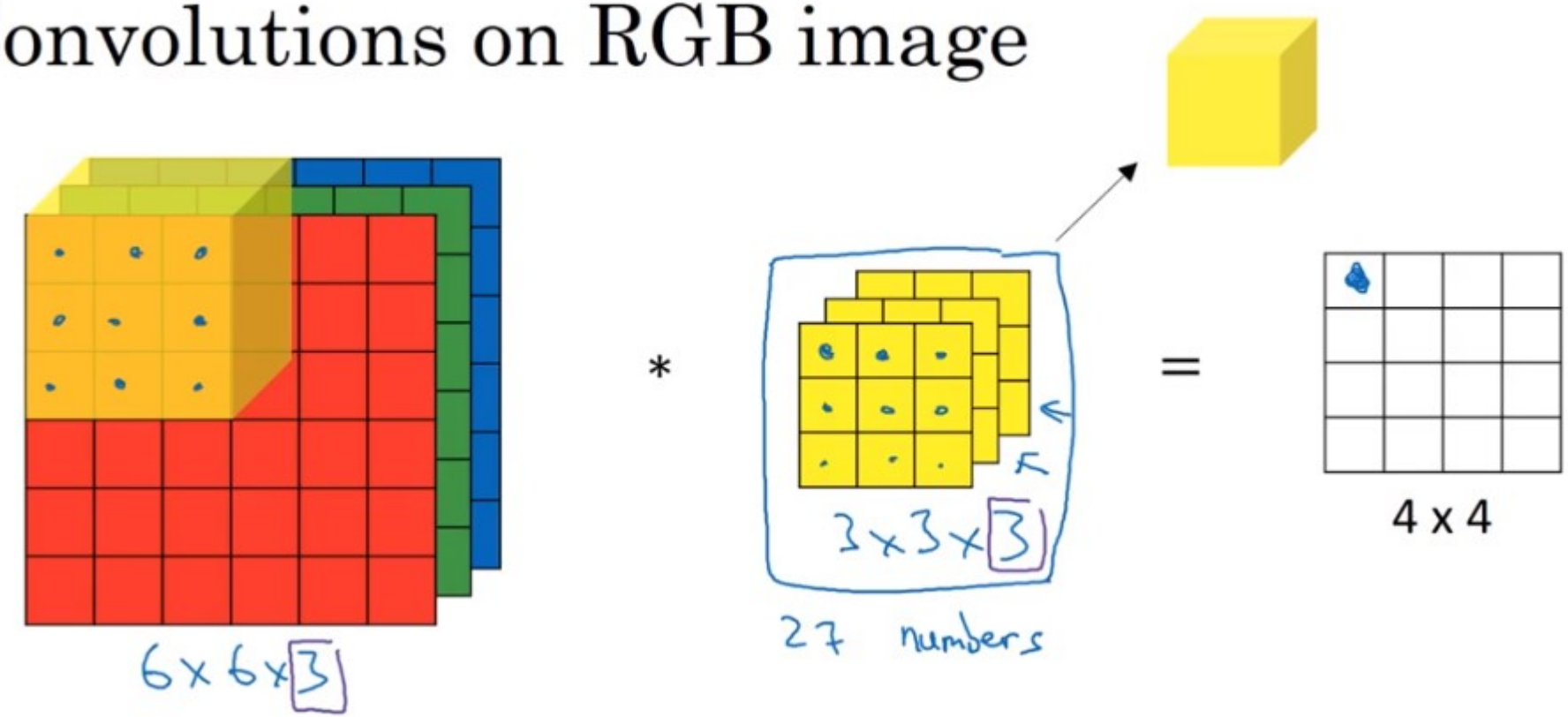




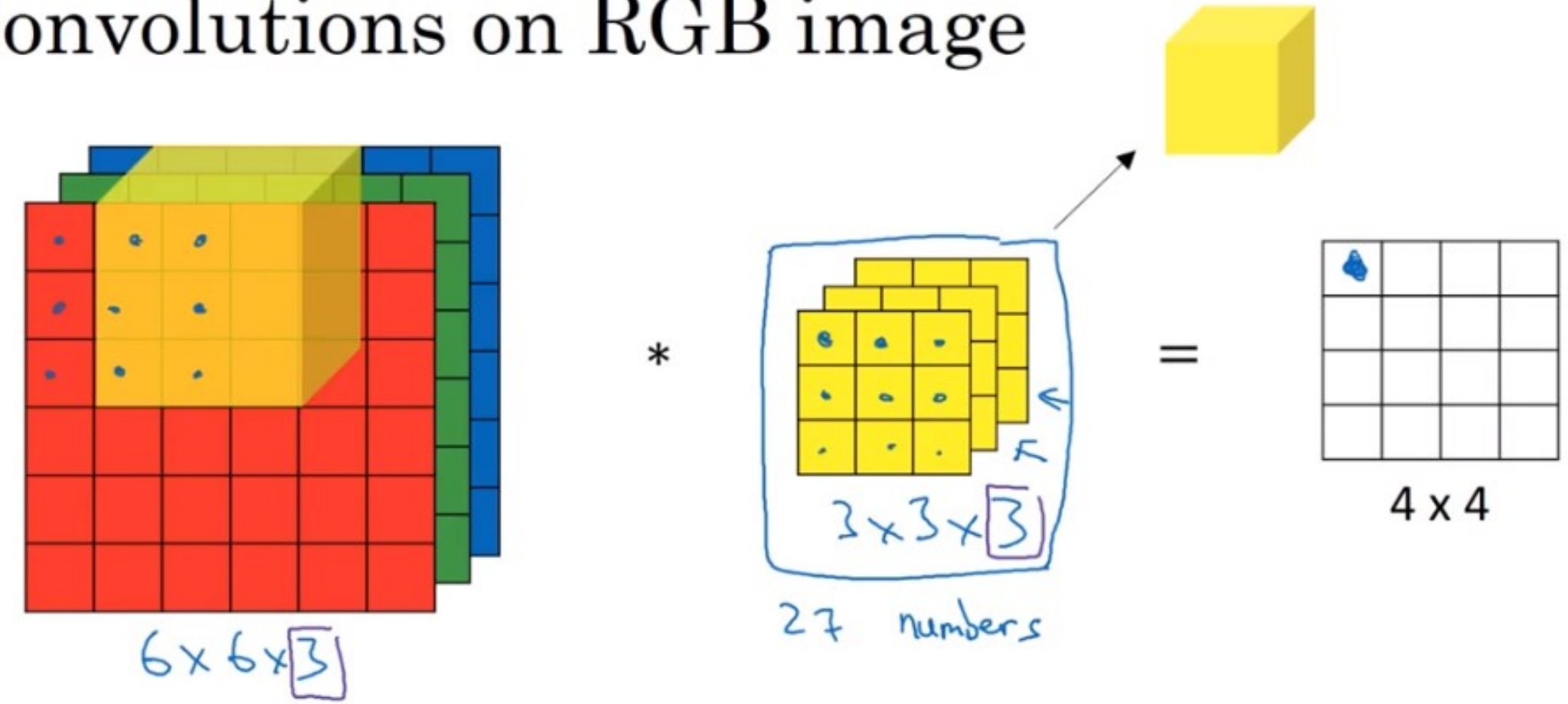
Convolutions over volume



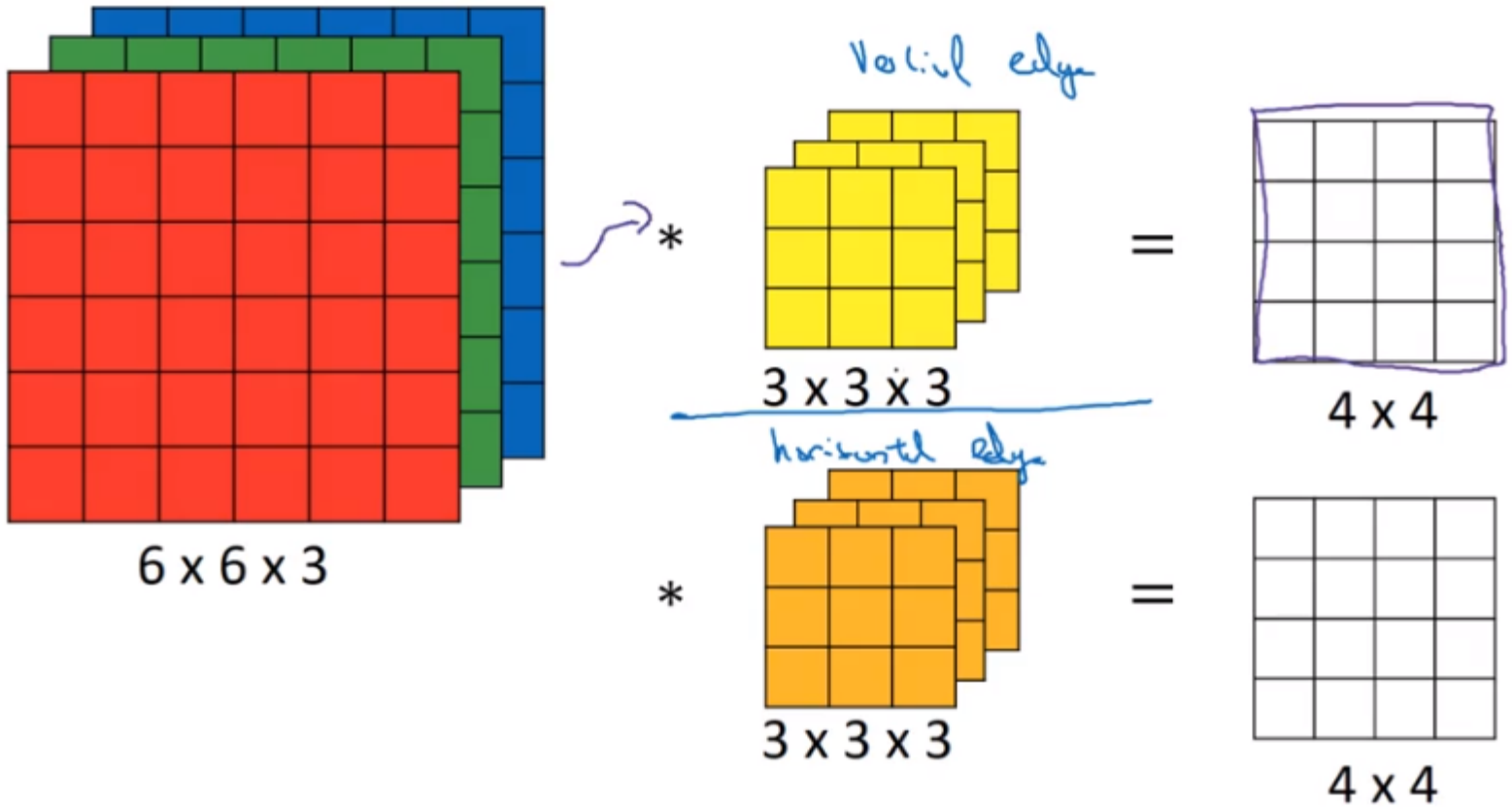
Convolutions on RGB image



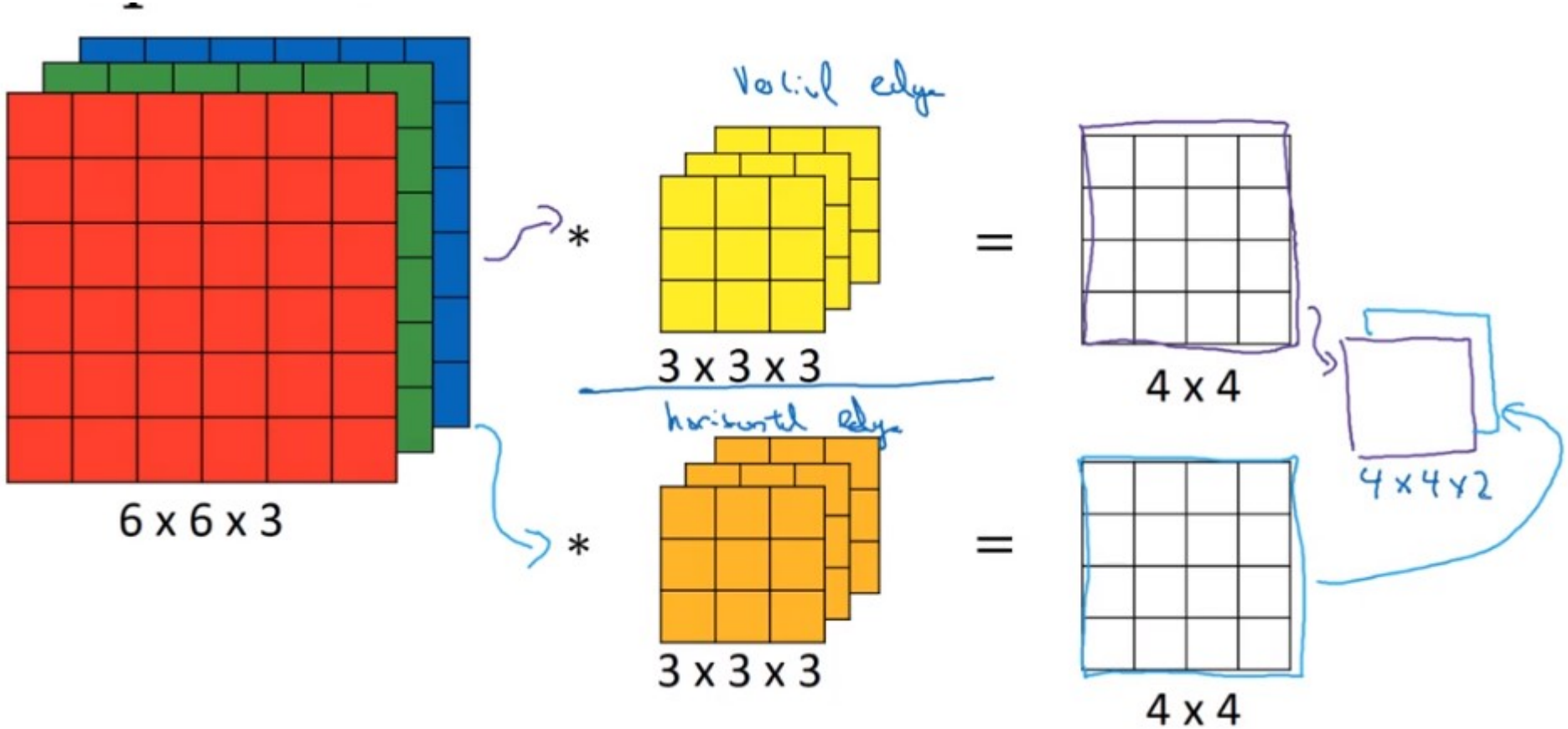
Convolutions on RGB image



2 filters



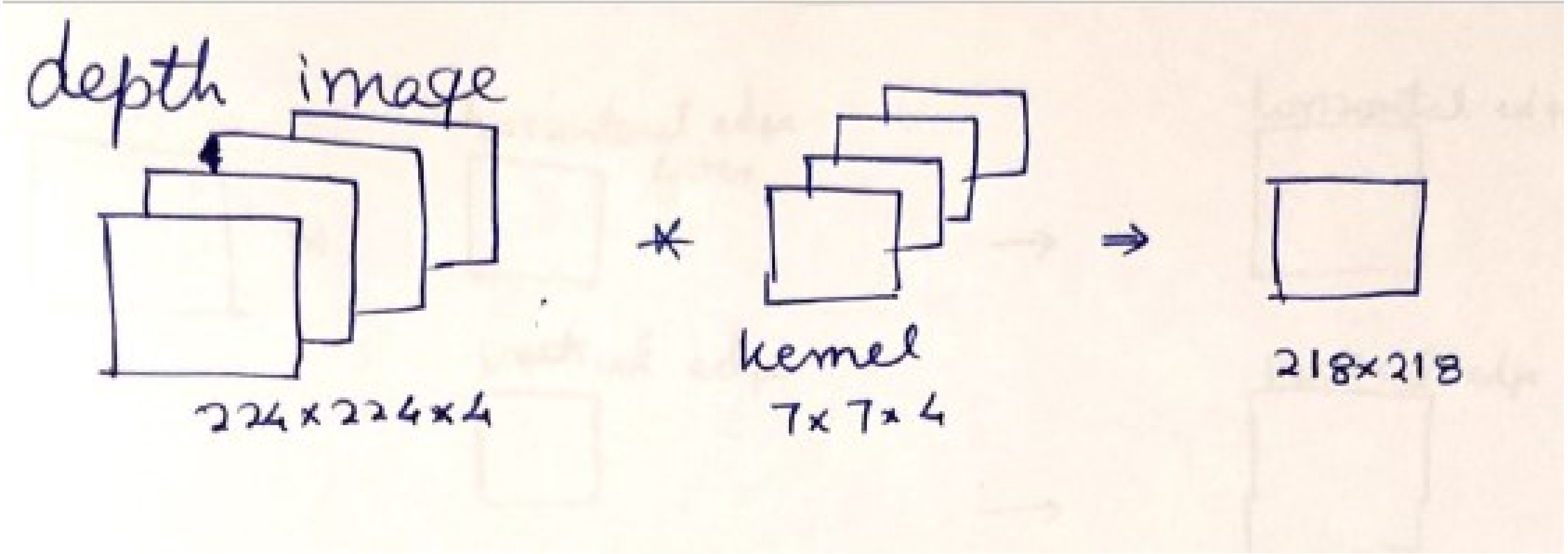
2 filters



$n \times n$ image $\quad$ $f \times f$ filter

padding p $\quad$ stride s

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

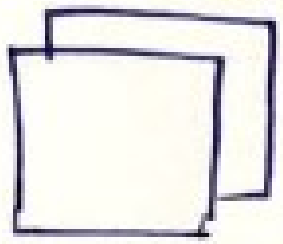


output size = $\left[\frac{n+2p-f}{s}+1\right]$

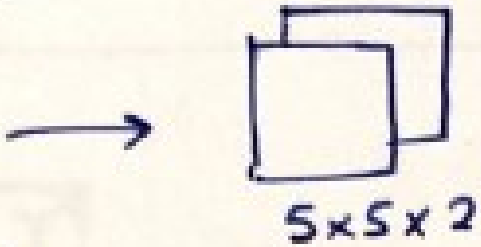
stride

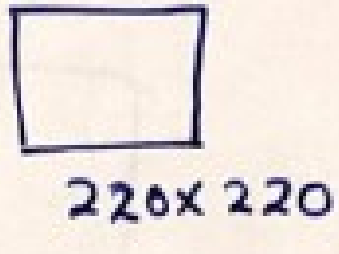
$\lfloor z \rfloor = \text{floor}(z)$

2 grayscale camera located side by side

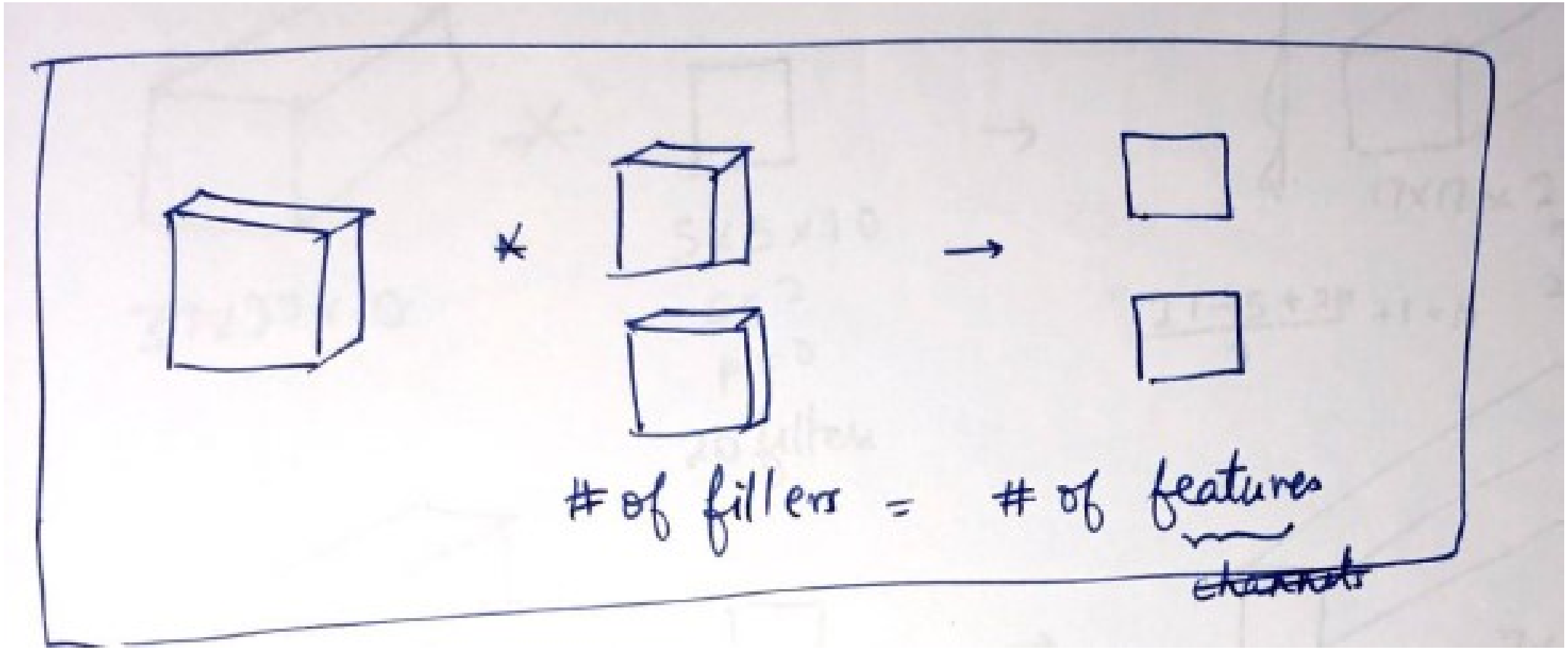


224 x 224 x 2



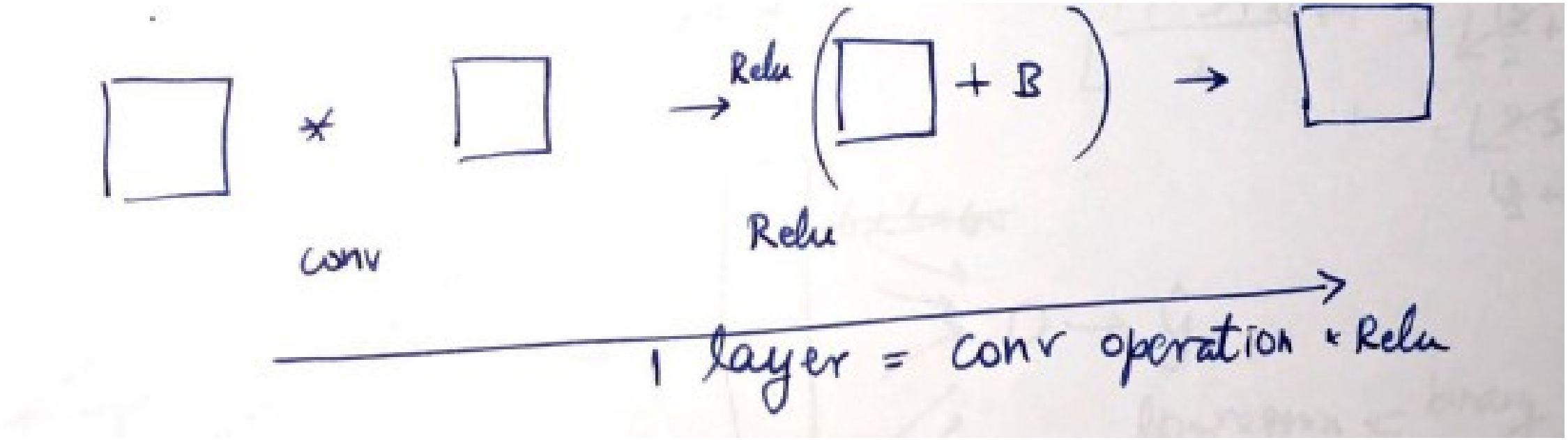


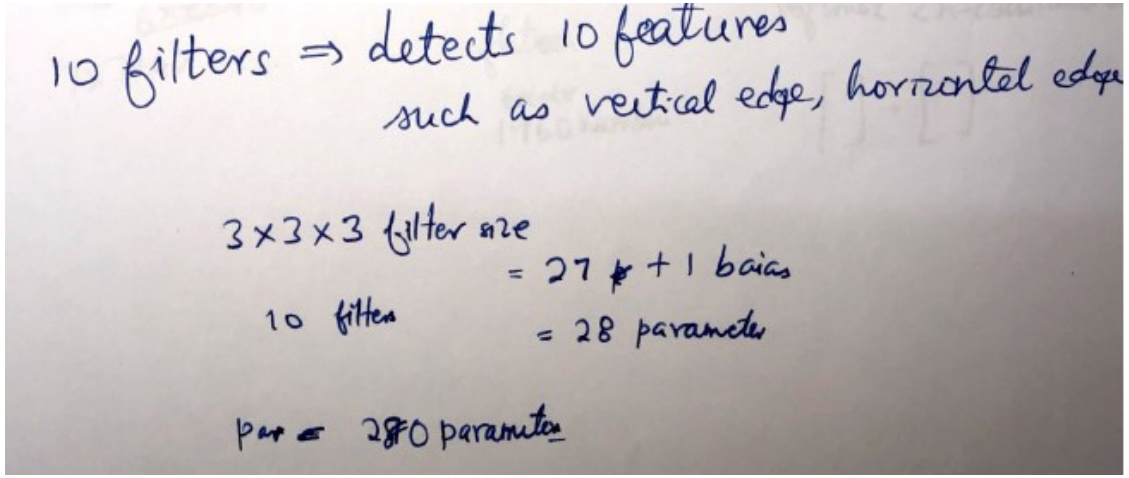
No of features



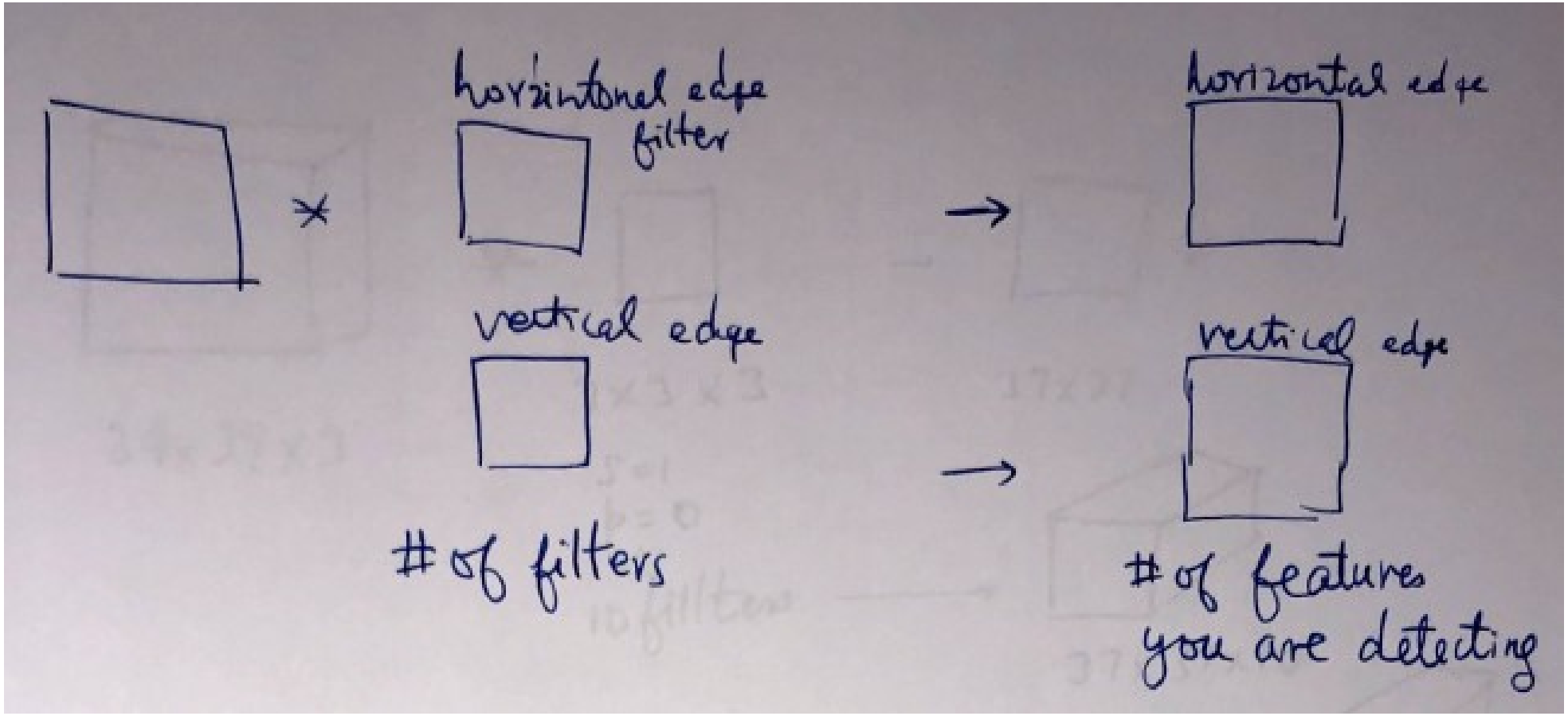
Acknowledgments & Credits are mentioned to inspirational resources presented in the end of presentation. For more like this, https://sites.google.com/view/AIforEveryone

No of parameters





No of features



Total no of parameter

		# param
input	(32,32,3)	0
conv 1 (f=5, s=1)	(28,28, 6)	$(5\times5\times3+1)\times6$
maxpool 1	(14,14, 6)	
conv 2	(5,5,16) = 400	
pool 2		
FC3	(120,1)	
FC4		
softmax		

Key takeaways

- Convnets are the best type of machine-learning models for computer-vision tasks. It's possible to train one from scratch even on a very small dataset, with decent results.
- On a small dataset, overfitting will be the main issue. Data augmentation is a powerful way to fight overfitting when you're working with image data.
- It's easy to reuse an existing convnet on a new dataset via feature extraction. This is a valuable technique for working with small image datasets.
- As a complement to feature extraction, you can use fine-tuning, which adapts to a new problem some of the representations previously learned by an existing model. This pushes performance a bit further.