High Impact Skills Development Program

in Artificial Intelligence, Data Science, and Blockchain

Module: [Computer Vision]

Lecture 3: [Introduction to Convolutional Neural Networks]

Instructor: [Dr. Rabia Irfan]

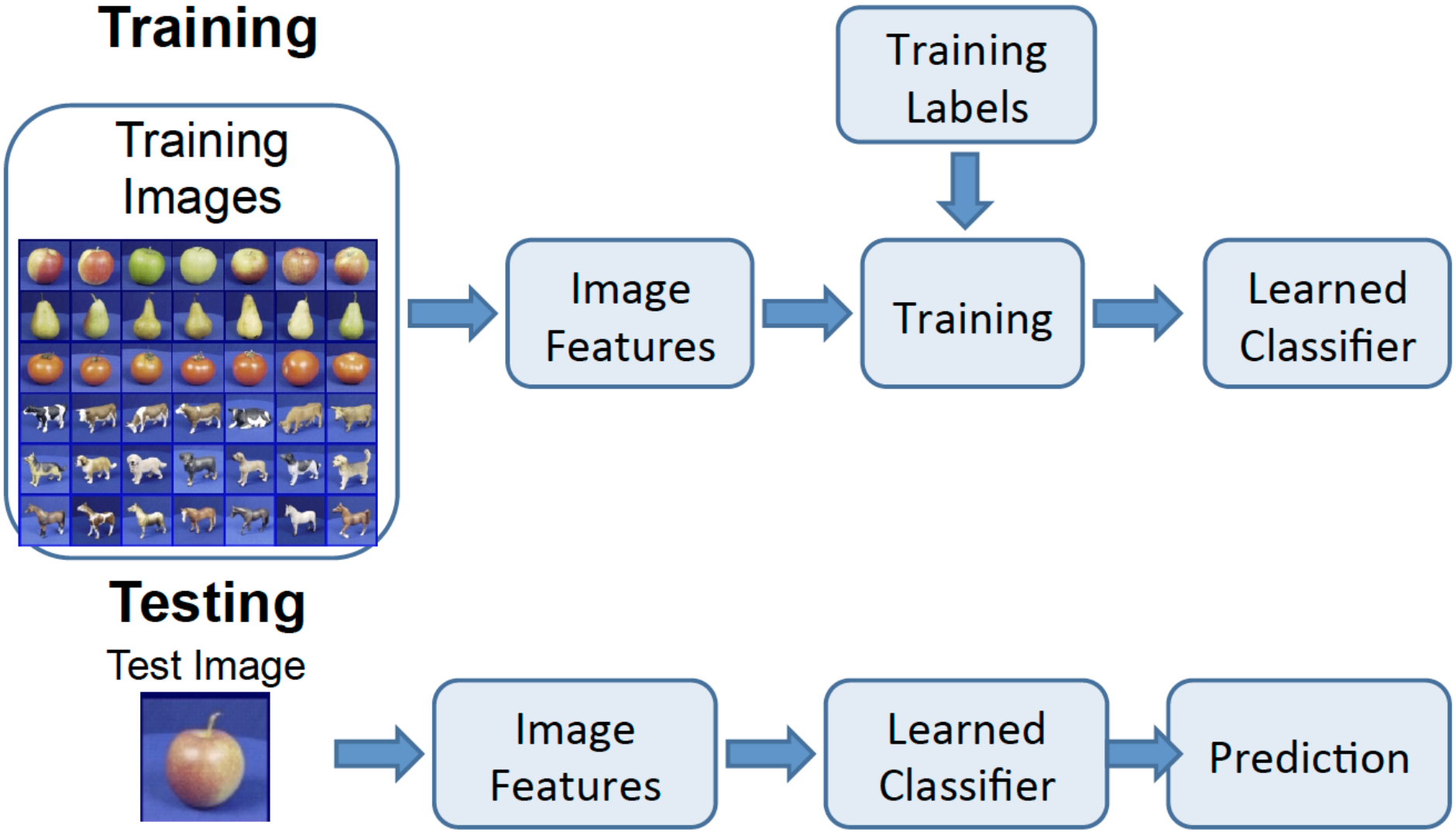
[Assistant Professor], SEECS, NUST

Agenda

- Overview of Image Classification Process
- What are Convolutional Neural Networks?
- Why Convolutional Neural Networks?
- Creating a CNN model

Image Classification

Image Classification Pipeline



Targets/Labels

Supervised Learning

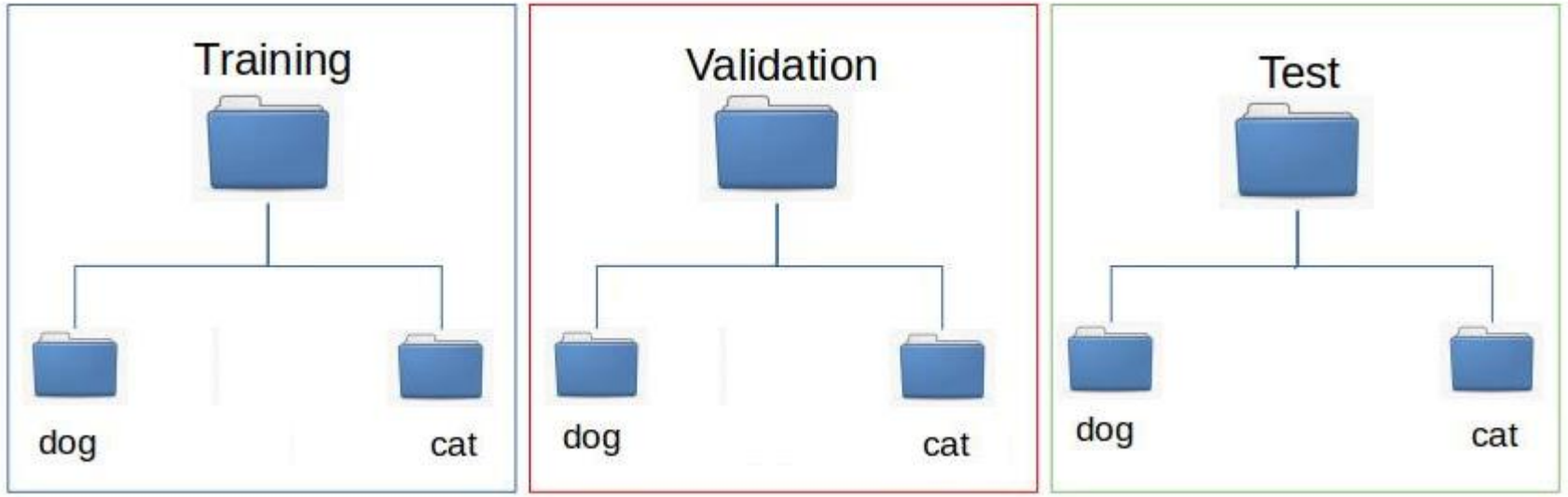
$y = 0$

$y = \text{“cat”}$

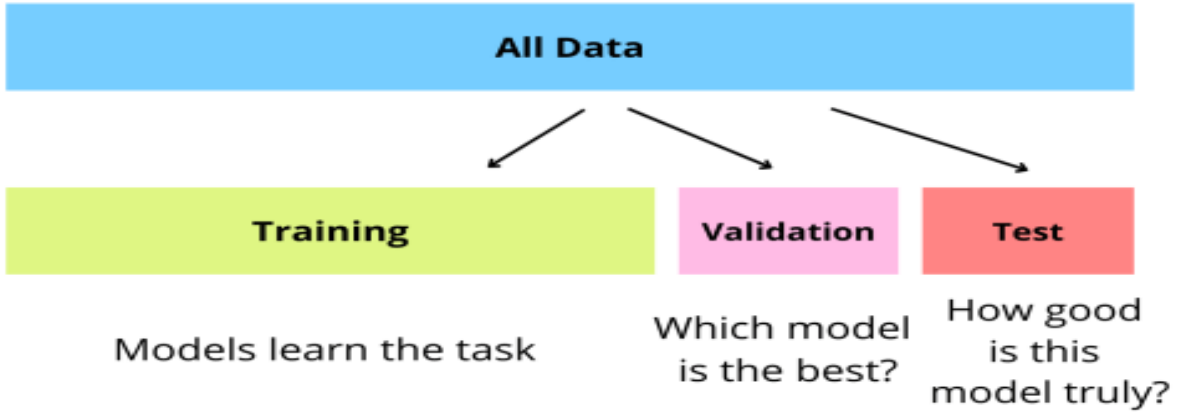
$y = 1$

$y = \text{“dog”}$

Dataset



70:20:10
80:10:10
or
K-Fold



Accuracy

- Number of samples that have been predicted correctly divided by the total number of samples

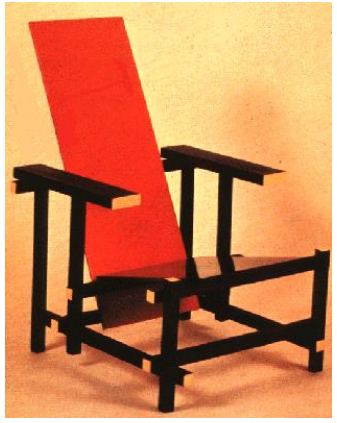
Samples: n	1	2	3	4
y	1	0	1	0
$\hat{y}$	1	0	0	1
$Correct$	1	1	0	0

$$= \frac{1}{4}(1+1+0+0) = 0.5$$

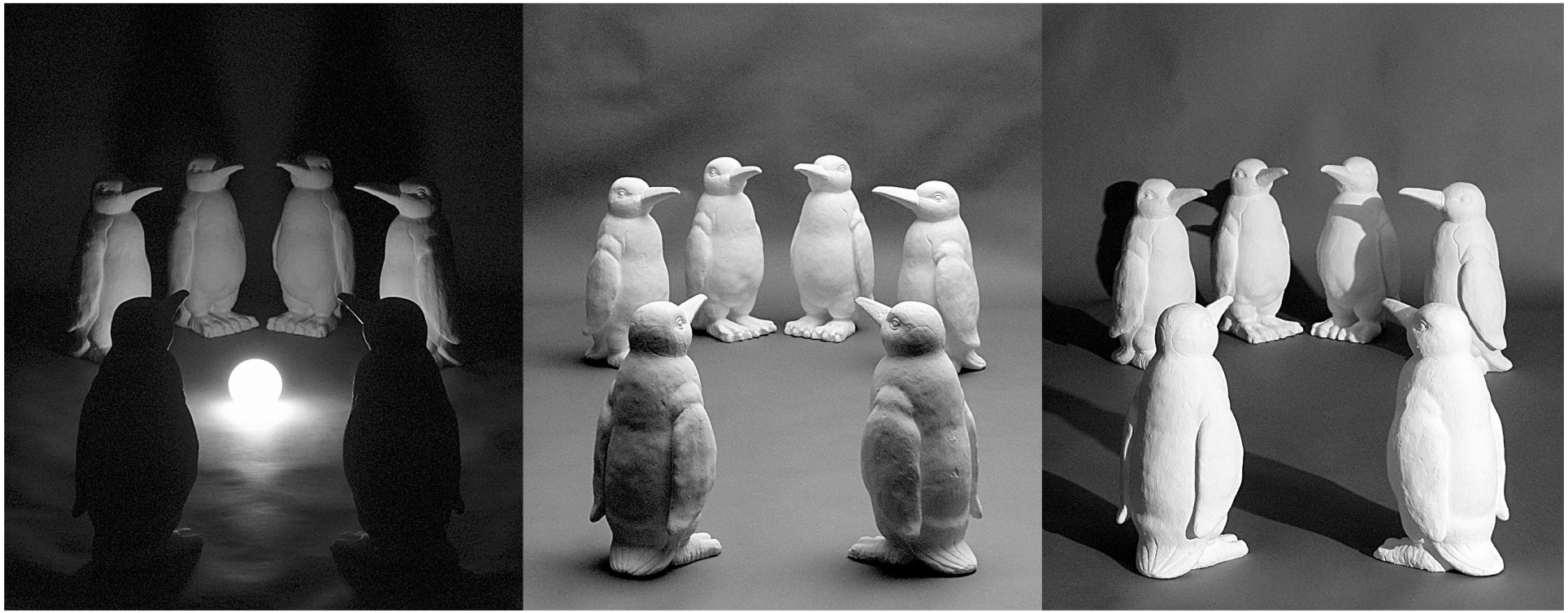
MNIST Dataset

Challenges-IntraClass Variations



Challenges-Illumination Variations



Challenges-Deformation

Challenges-Viewpoint Variations

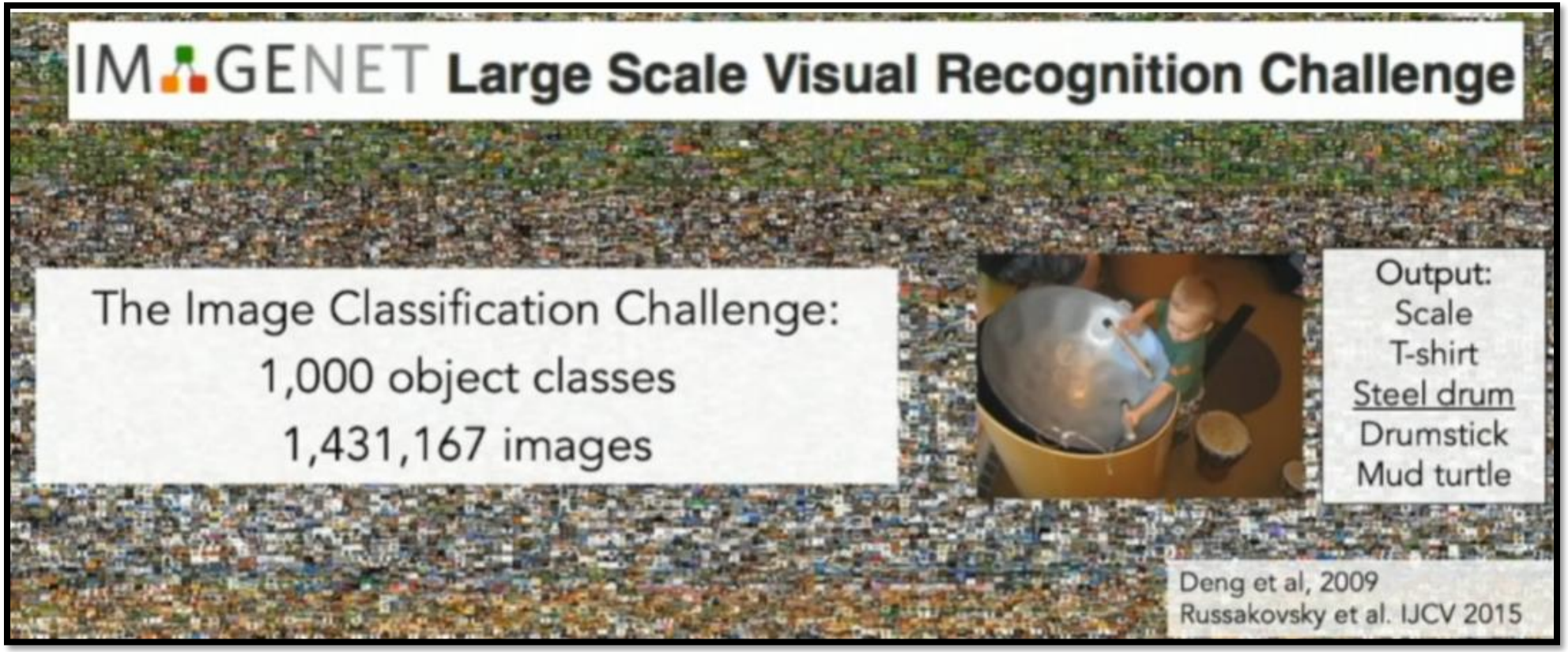
Michelangelo 1475-1564

slide credit: Fei-Fei, Fergus & Torralba

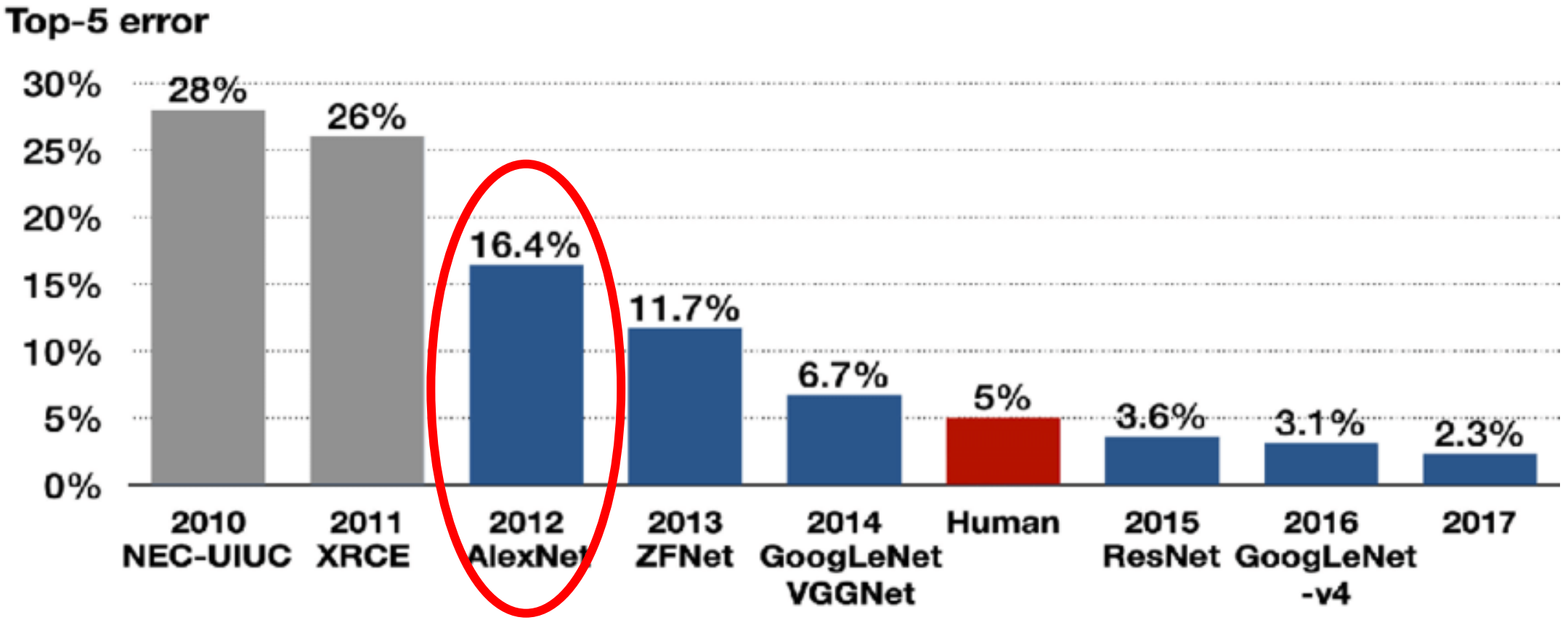
Challenges-Background Clutter

Challenges-Occlusion

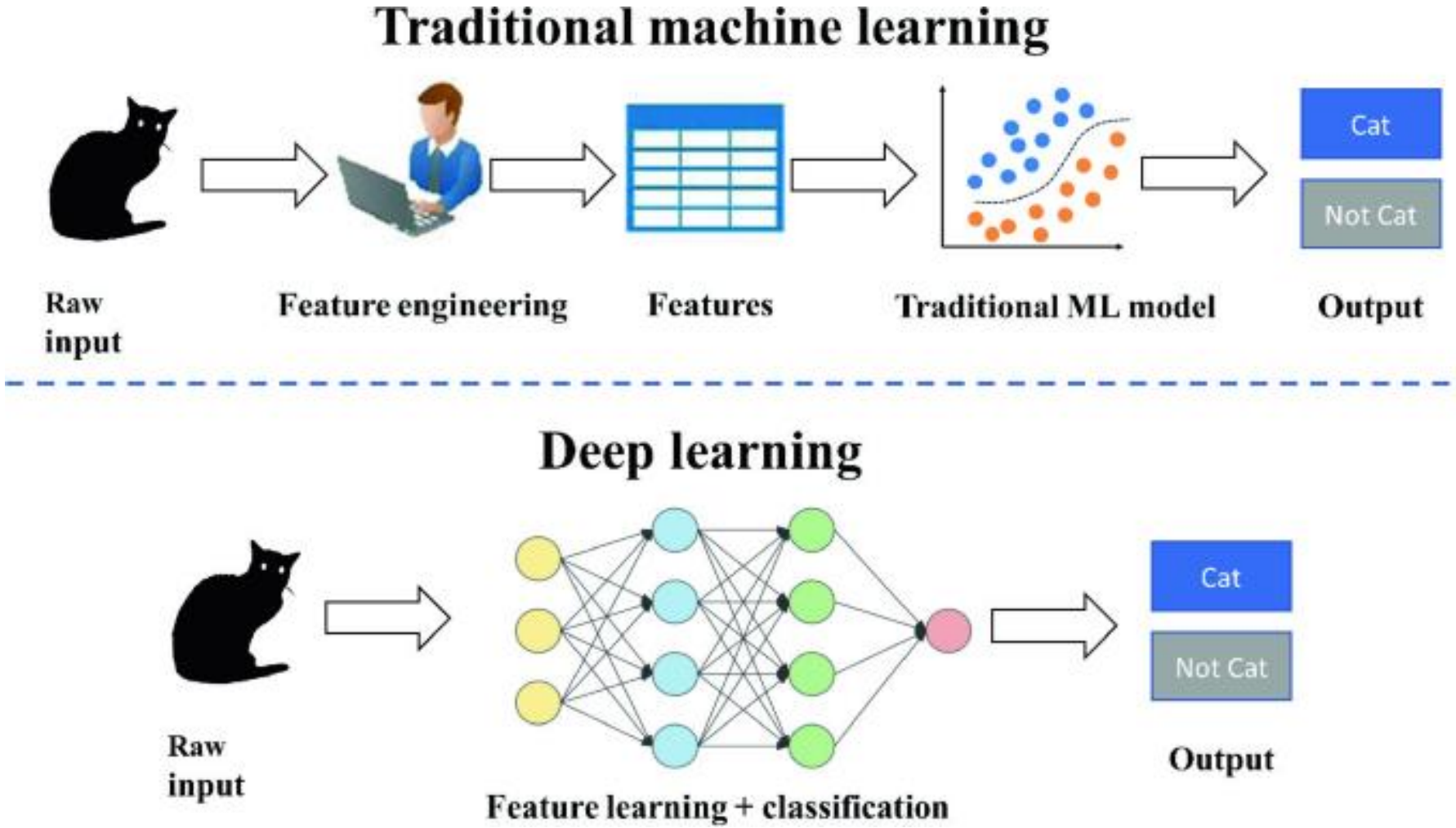
ImageNet Dataset



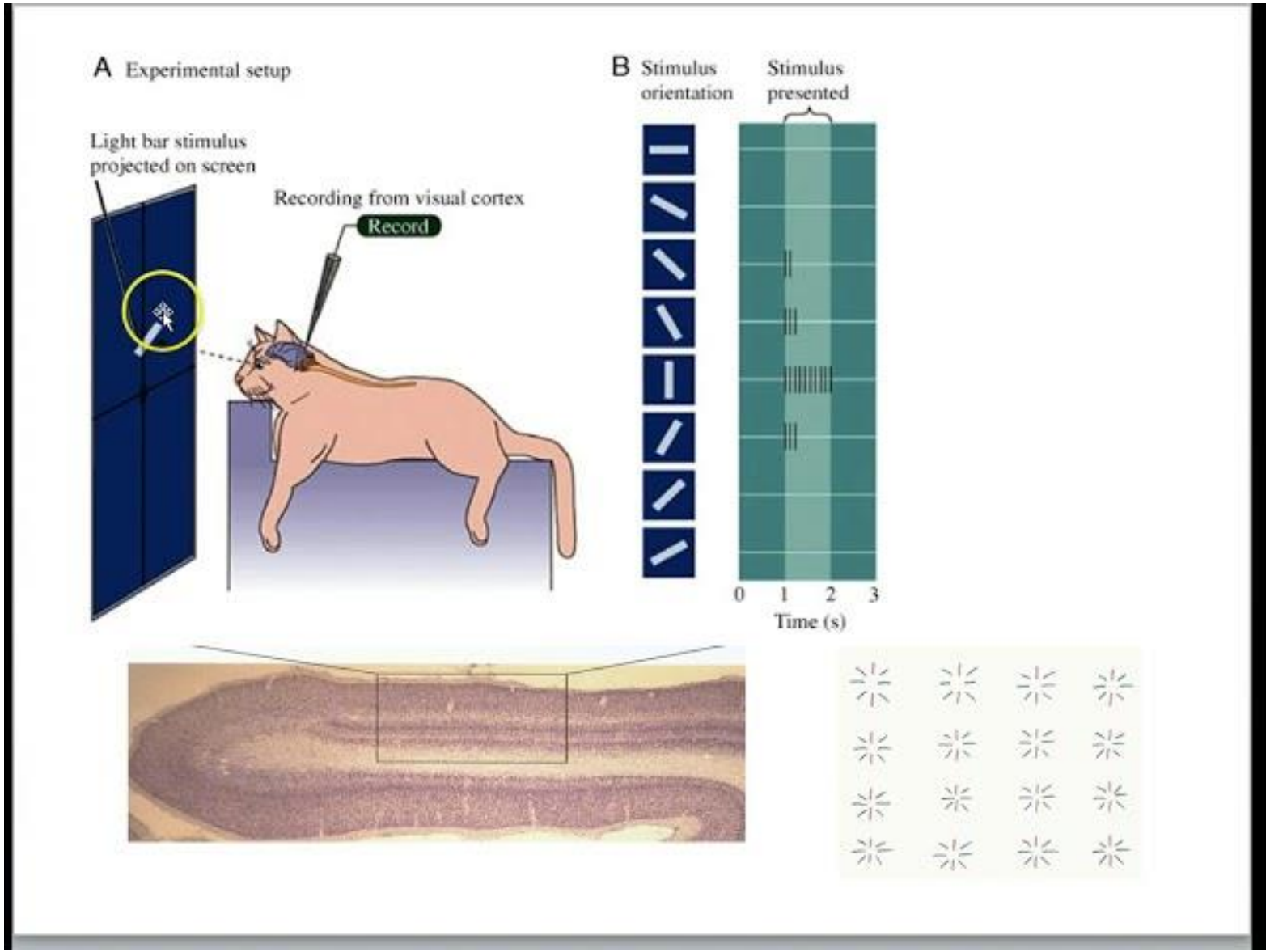
ImageNet Challenge



Deep Learning vs Machine Learning



Convolutional Neural Networks



Convolutional Neural Networks

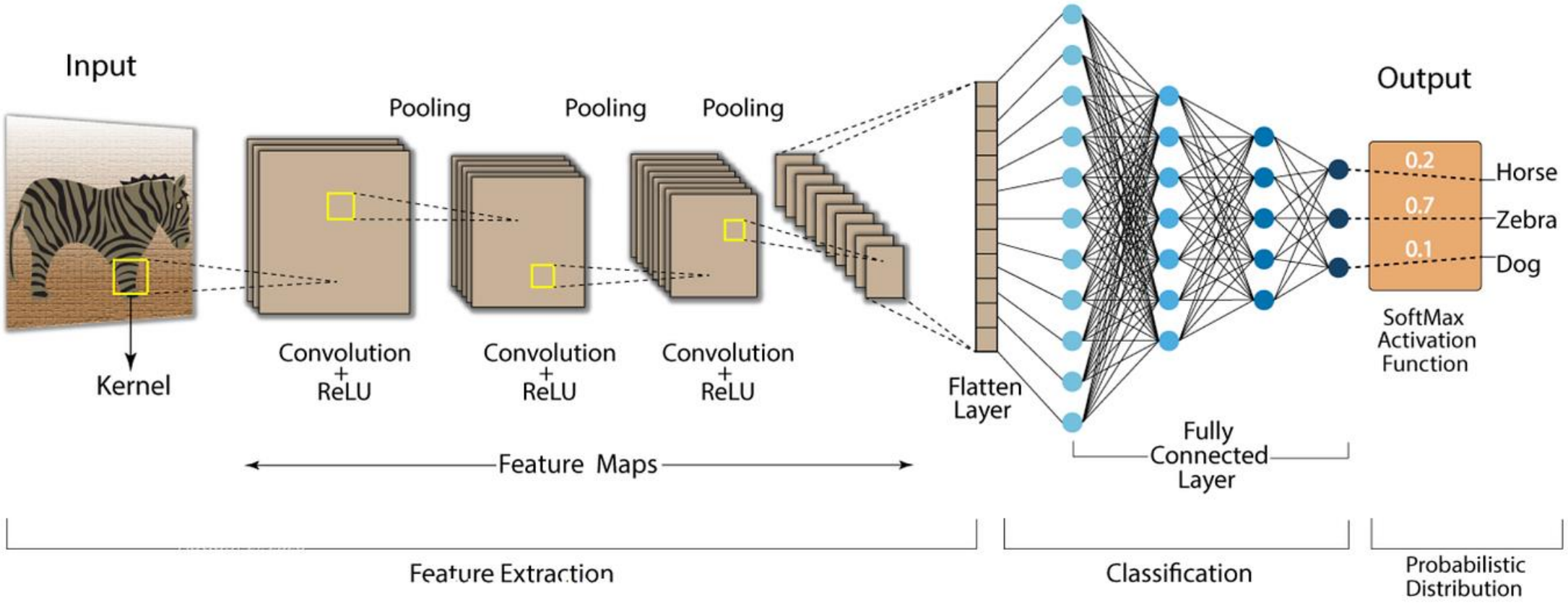


Image credit: https://nafizshahriar.medium.com/what-is-convolutional-neural-network-cnn-deep-learning

Convolutional Neural Networks

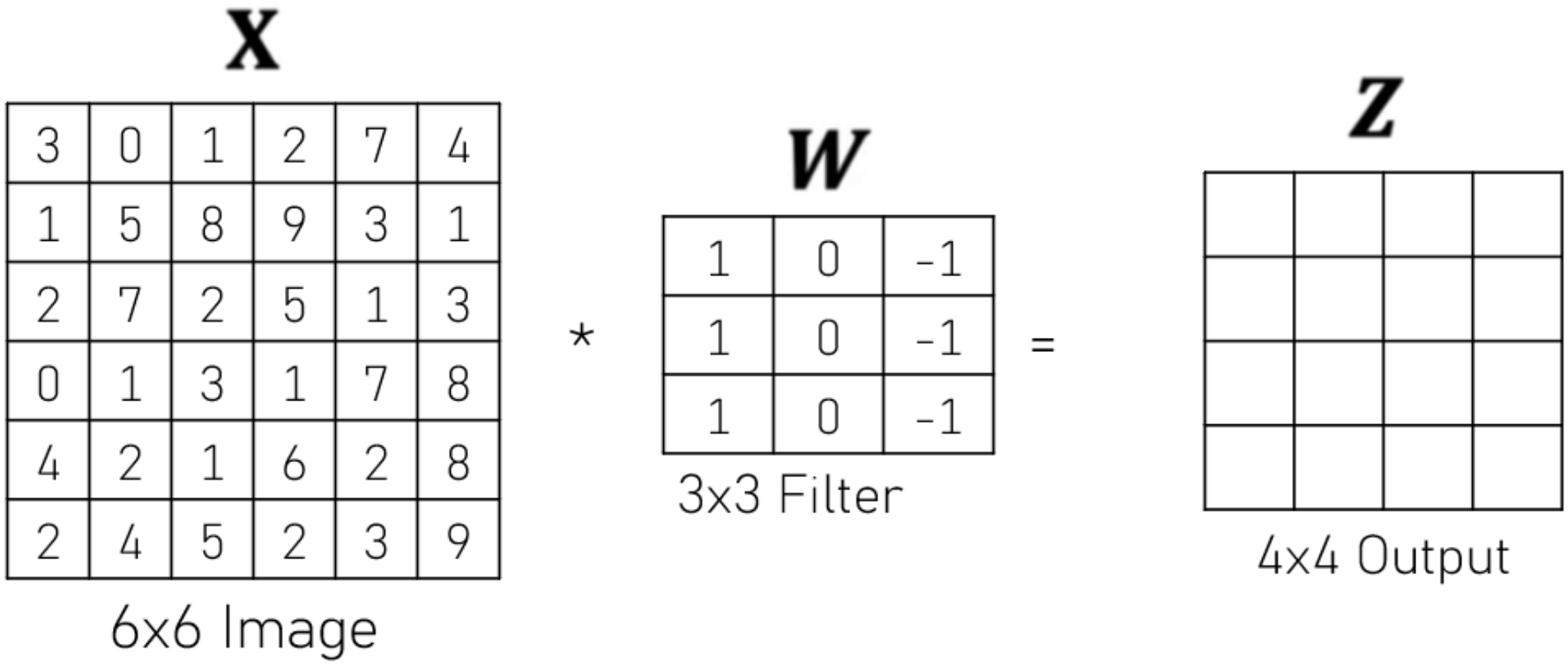
- To Understand Convolutional Neural Networks, we need to know following:

  - Convolution
  - Padding
  - Stride
  - Activation (ReLU)
  - Pooling
  - Sigmoid/Softmax

Convolution – Linear Filtering

Convolution is a linear operation that multiplies the corresponding values of a filter with the pixels in the image and sums the products.

$$Z = W * X$$



Recall - Edge Detection

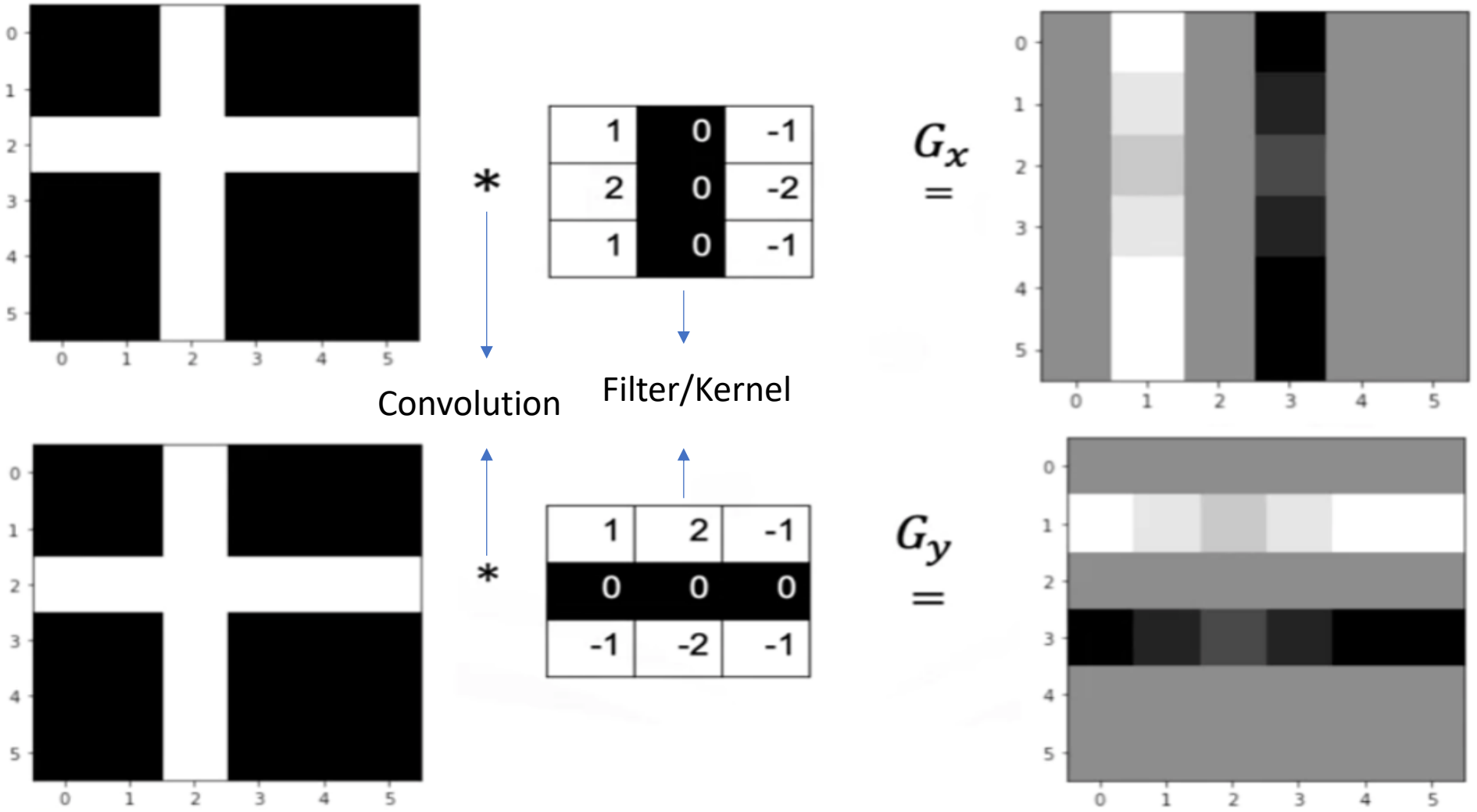
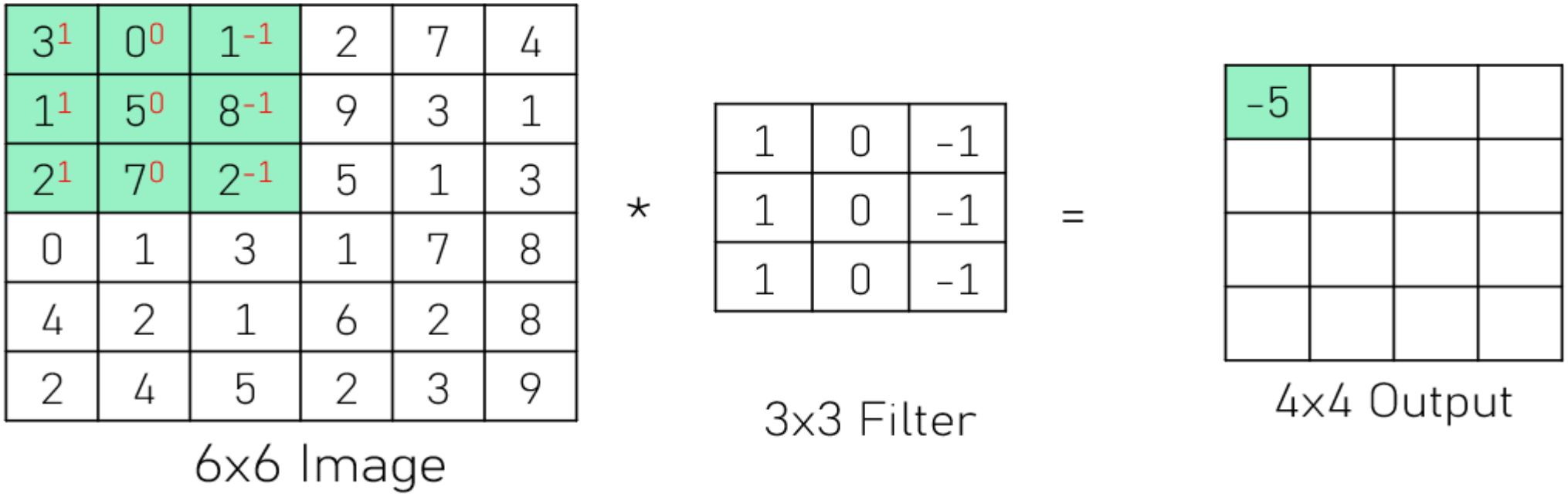


Image credit: coursera

Convolution – Linear Filtering

- To compute the first value of output
  - Take the filter and superimpose it on the (top left of) image
  - Take the element-wise product and add the (9) values together
  - 3x1 + 1x1 + 2x1 + 0x0 + 5x0 + 7x0 + 1x-1 + 8x-1 + 2x-1 = -5



Convolution – Linear Filtering

- To compute the next value of output
  - Shift the filter one position to the right
  - 0x1 + 5x1 + 7x1 + 1x0 +8x0 + 2x0 + 2x-1 + 9x-1 + 5x-1 = -4

3	0^{1}	1^{0}	2^{-1}	7	4
1	5^{1}	8^{0}	9^{-1}	3	1
2	7^{1}	2^{0}	5^{-1}	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

6x6 Image

*

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

3x3 Filter

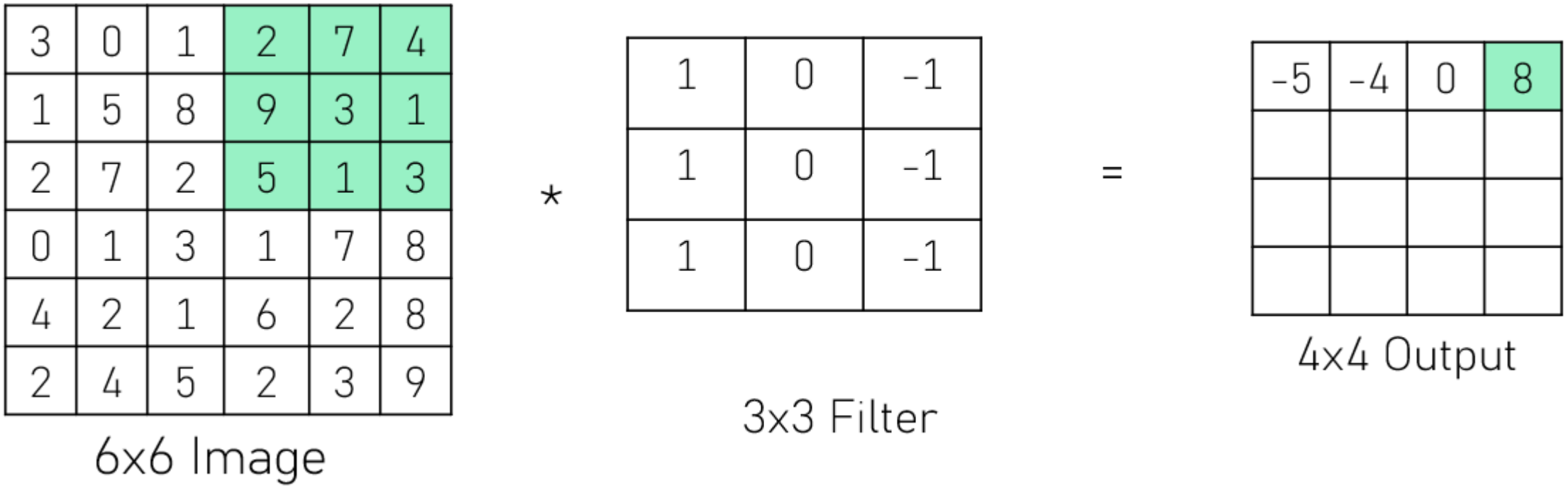
=

-5	-4		

4x4 Output

Convolution – Linear Filtering

- Repeat the process by sliding the filter to the right



Convolution – Linear Filtering

- Arriving at the end of the image – Shift one position down

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

6x6 Image

*

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

3x3 Filter

=

-5	-4	0	8
-10			

4x4 Output

Convolution – Linear Filtering

- Repeat the process for complete image
- For implementation – Keras: *Conv2D*

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

6x6 Image

*

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

3x3 Filter

=

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

4x4 Output

Learning Parameters

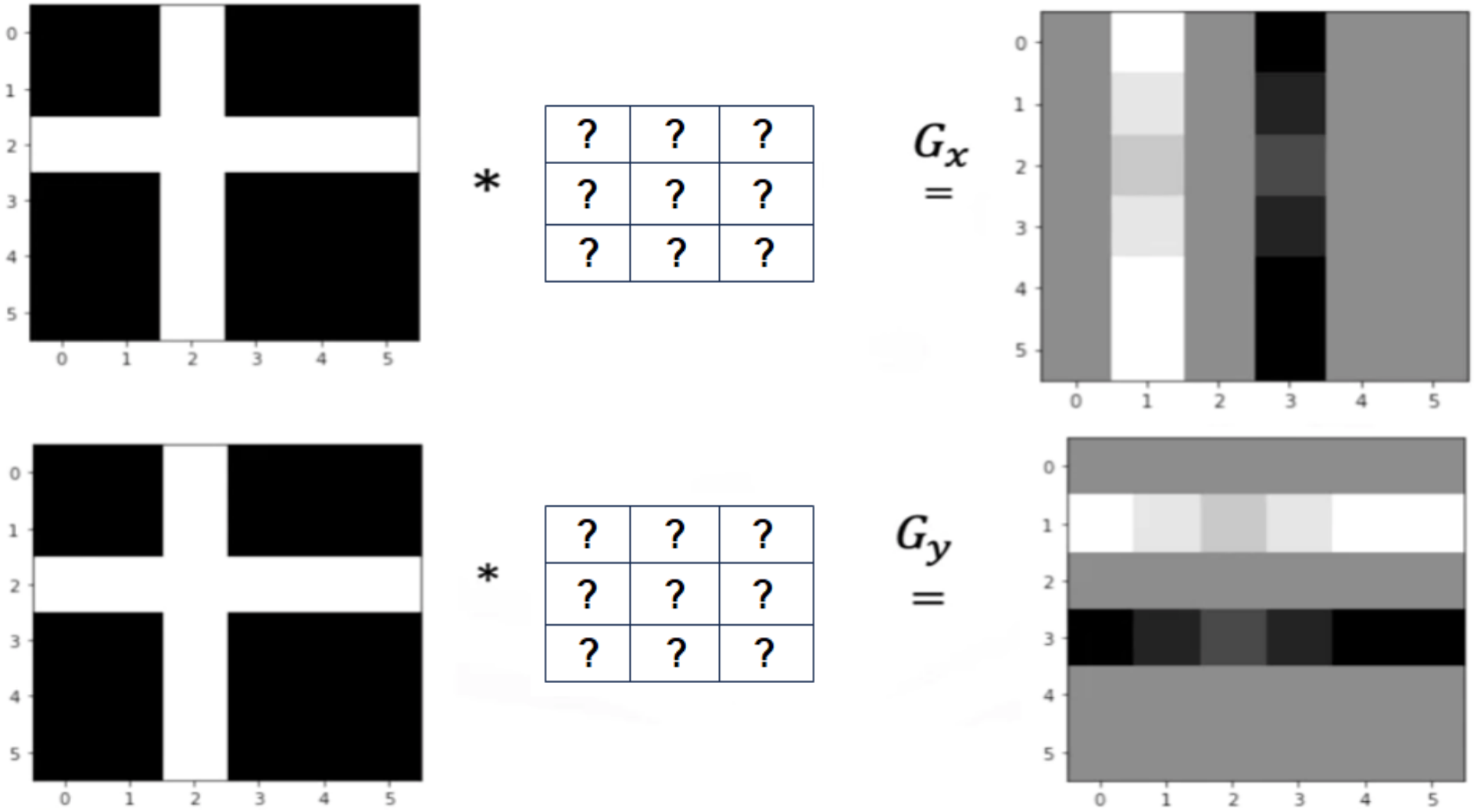


Image credit: coursera

Convolution with Padding

- In general, when $n \times n$ image is convolved with $f \times f$ filter, output size will be:

  $(n - f + 1) \times (n - f + 1)$

- For instance, Convolution of a 6x6 Image with 3x3 filter produced a 4x4 output matrix

  $=(6 - 3 + 1) \times (6 - 3 + 1)$
  $= 4 \times 4$

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

6x6 Image

*

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

3x3 Filter

=

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

4x4 Output

- Problems:
  - Every time convolution is applied – Image Shrinks
- Solution:
  - Pad the image prior to applying convolution

Convolution with Padding

- Pad the 6x6 Image with a border of 1 pixel -> 8 x 8 image

0	0	0	0	0	0	0	0
0	10	10	10	2	2	2	0
0	10	10	10	2	2	2	0
0	10	10	10	2	2	2	0
0	5	5	5	10	10	10	0
0	5	5	5	10	10	10	0
0	5	5	5	10	10	10	0
0	0	0	0	0	0	0	0

$$(n - f + 1) \times (n - f + 1)$$

$$= (8 - 3 + 1) \times (8 - 3 + 1)$$

= 6 x 6

- Convolution of padded image with 3x3 filter gives a 6x6 output same as input image size – Preserves the original spatial dimensions
- For padding amount ***p=1***, output size is:

$$(n + 2p - f + 1) \times (n + 2p - f + 1)$$

Convolution with Padding

- How much to pad?
  - To determine the value of p which ensures that the output size is same as input size:

$$p = \frac{(f - 1)}{2}$$

  - Example, for filter of size 3x3, p = (3-1)/2 = 1
  - For filter of size 5x5, p = 2

  - '*valid*' Convolutions – No Padding
  - '*same*' Convolution – Pad as much as needed to preserve the spatial dimensions i.e. output size is same as input size

Recall - Convolution

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

6x6 Image

*

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

3x3 Filter

=

-5			

4x4 Output

Stride = 1

3	0^{1}	1^{0}	2^{-1}	7	4
1	5^{1}	8^{0}	9^{-1}	3	1
2	7^{1}	2^{0}	5^{-1}	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

6x6 Image

*

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

3x3 Filter

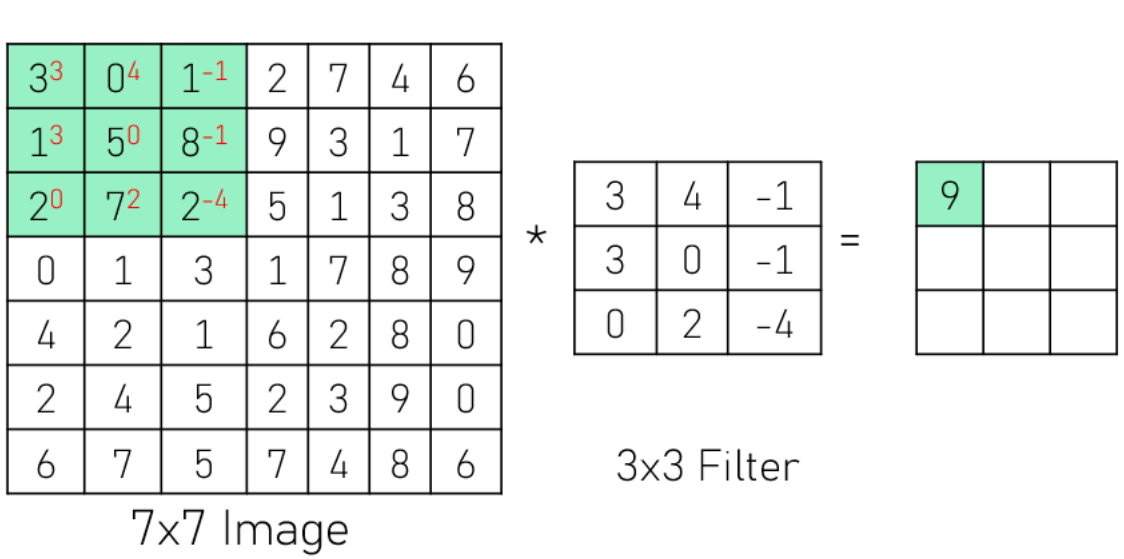
=

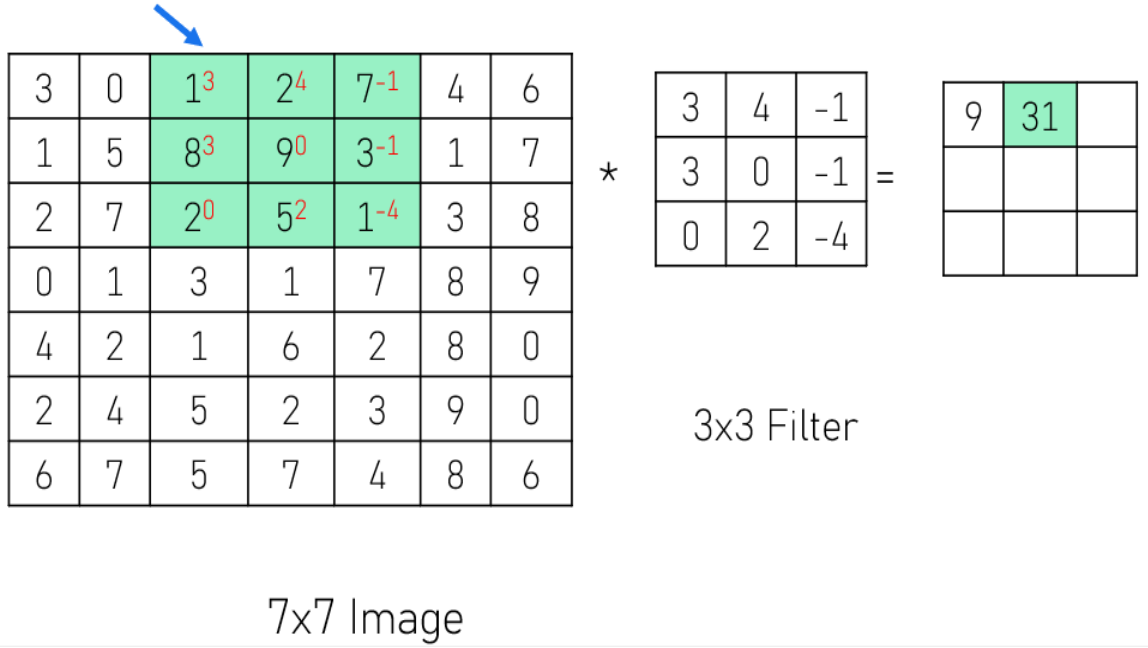
-5	-4		

4x4 Output

Strided Convolution

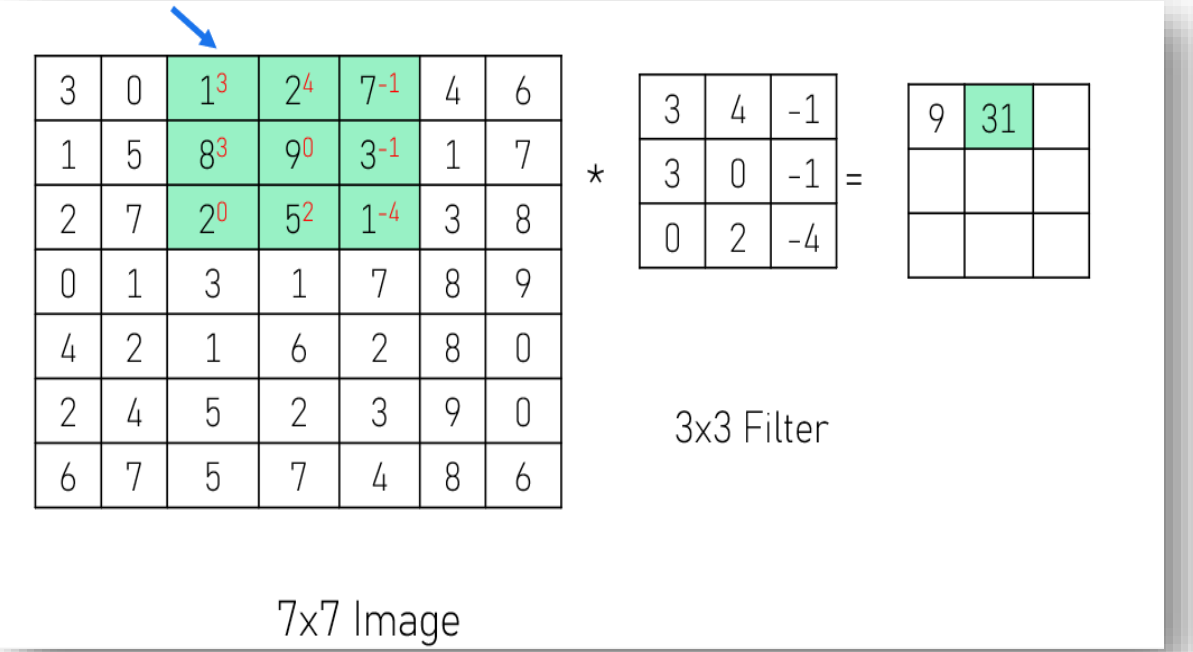
- Apply 3x3 filter on 7x7 image but with Stride = 2
- Instead of stepping the filter by 1 step - Step it over by 2 Steps

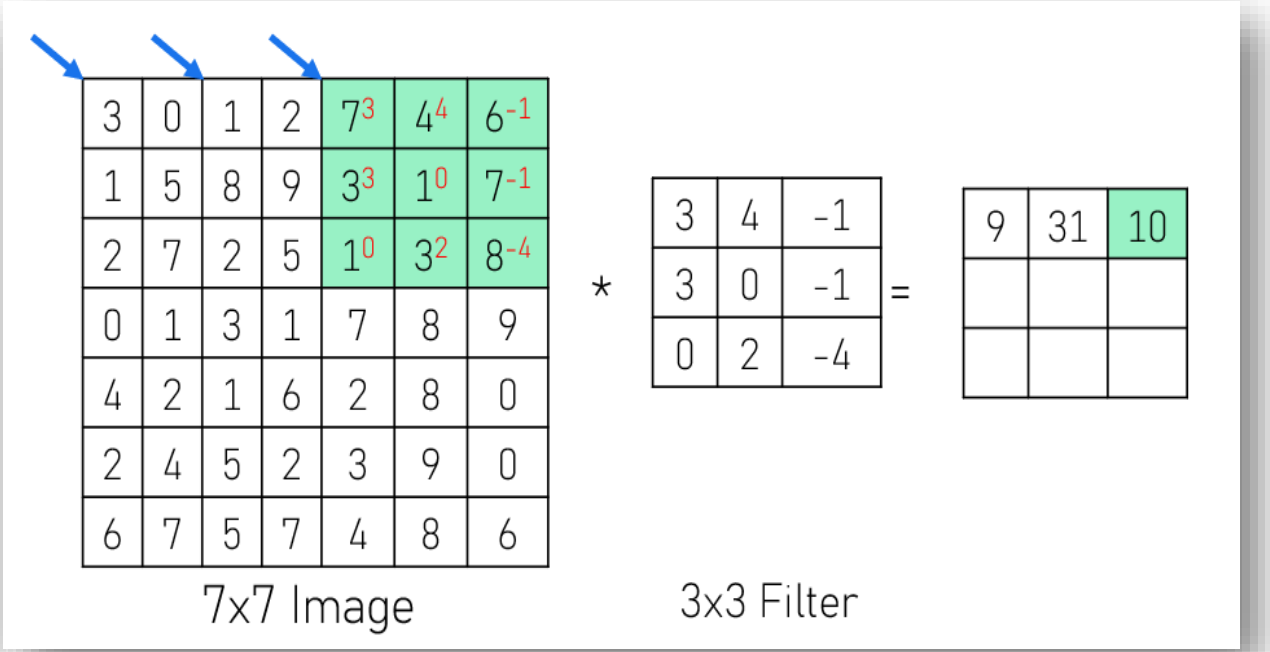




Strided Convolution

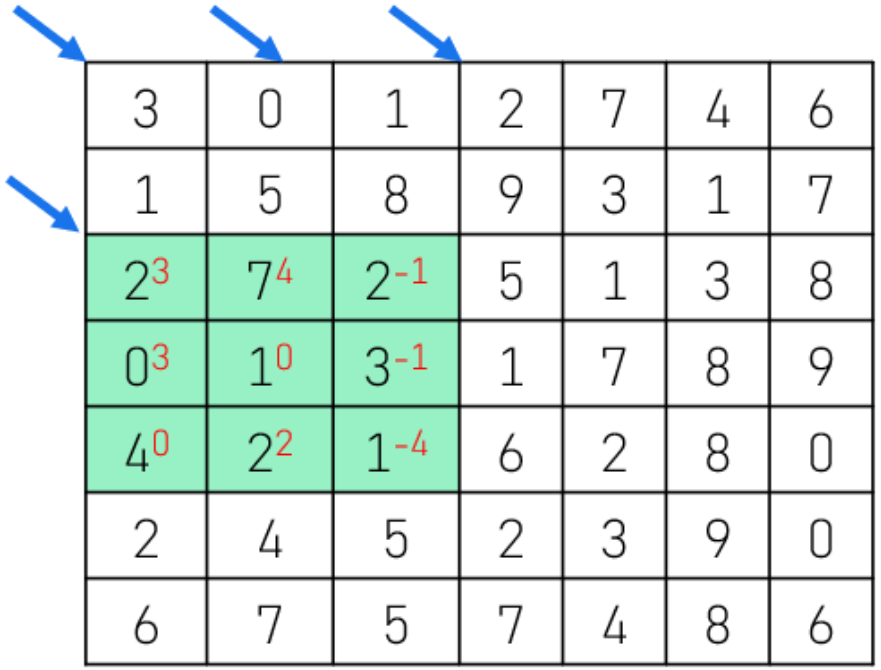
- Apply 3x3 filter on 7x7 image but with Stride = 2
- Instead of stepping the filter by 1 step - Step it over by 2 Steps





Strided Convolution

- Vertical Strides



3	0	1	2	7	4	6
1	5	8	9	3	1	7
2^{3}	7^{4}	2^{-1}	5	1	3	8
0^{3}	1^{0}	3^{-1}	1	7	8	9
4^{0}	2^{2}	1^{-4}	6	2	8	0
2	4	5	2	3	9	0
6	7	5	7	4	8	6

7x7 Image

*

3	4	-1
3	0	-1
0	2	-4

3x3 Filter

=

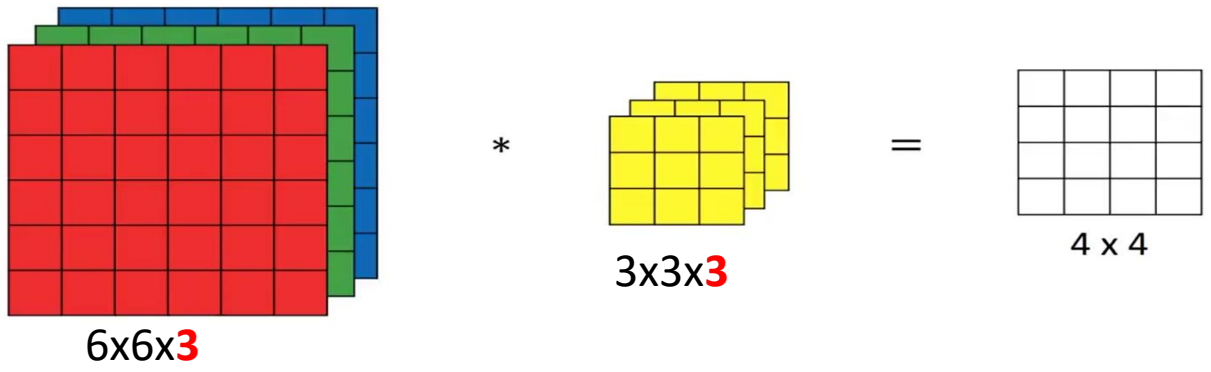
9	31	10
29		

Strided Convolution

- Dimensions
  - Input Image: $n \times n$
  - Filter size: $f \times f$
  - Padding: p
  - Stride: s
  - Output: $\text{floor}(\frac{n+2p-f}{s}) + 1 \times \text{floor}(\frac{n+2p-f}{s}) + 1$
  - Example: For 7x7 image with filter size 3x3 and No padding output = 3x3
  - Note: If the number is not an integer – Round it down i.e. Ignore the part of image (or image+padding) which does not fit with filter

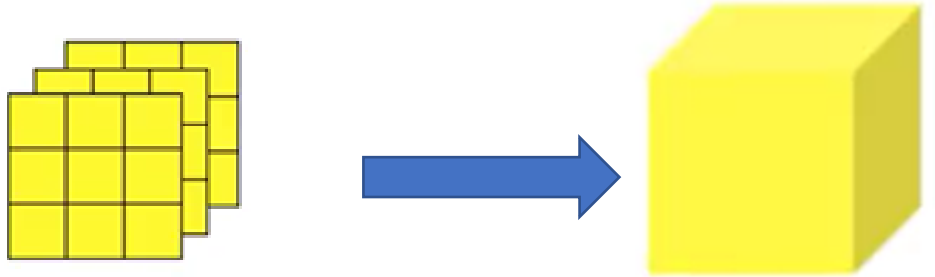
Convolution over Volumes

- Convolutions not on just 2D images but 3D volumes (For example an RGB image)
- Image: 6x6x**3** – Stack of three matrices one each for R, G and B
- To detect features – Convolve the image with 3x3x**3** filter
- The 3rd dimension (Depth) of the filter must match the depth of input volume (number of channels)

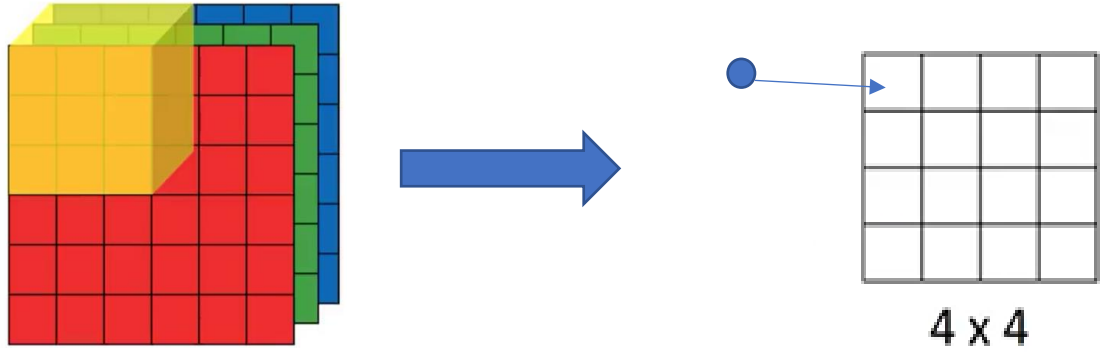


Convolution over Volumes

- Rather than stack of three matrices – Draw the volume as a cube



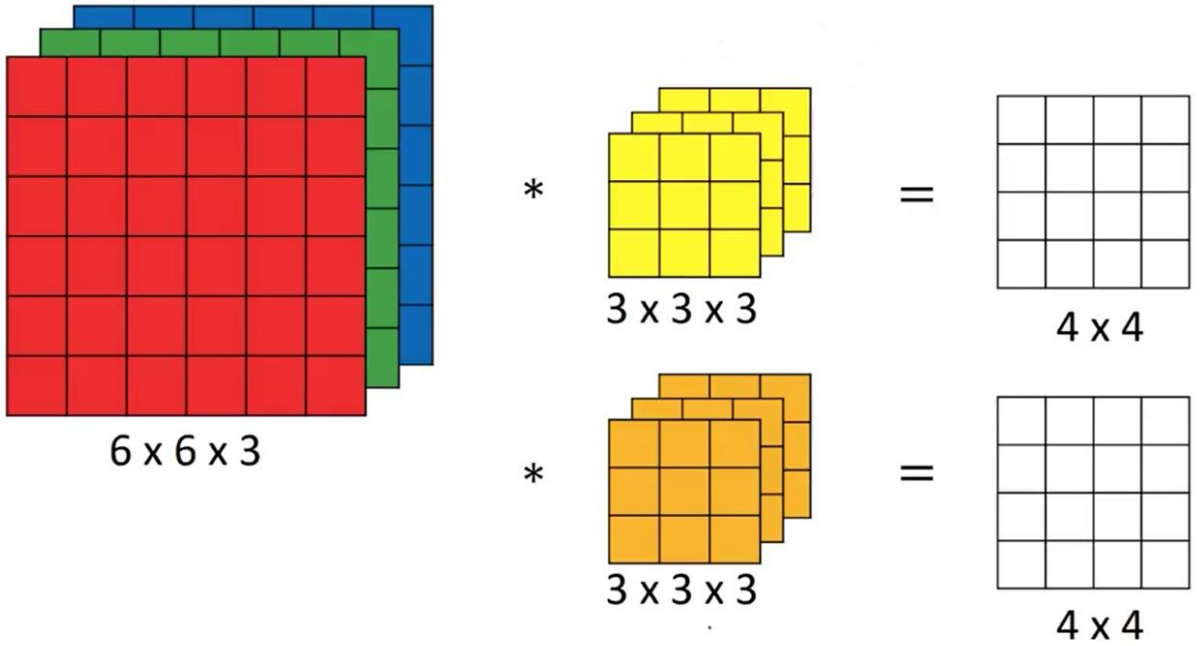
- Conv Operation: Multiply the 27 numbers in the filter with the corresponding 27 numbers in the image



Convolution over Volumes

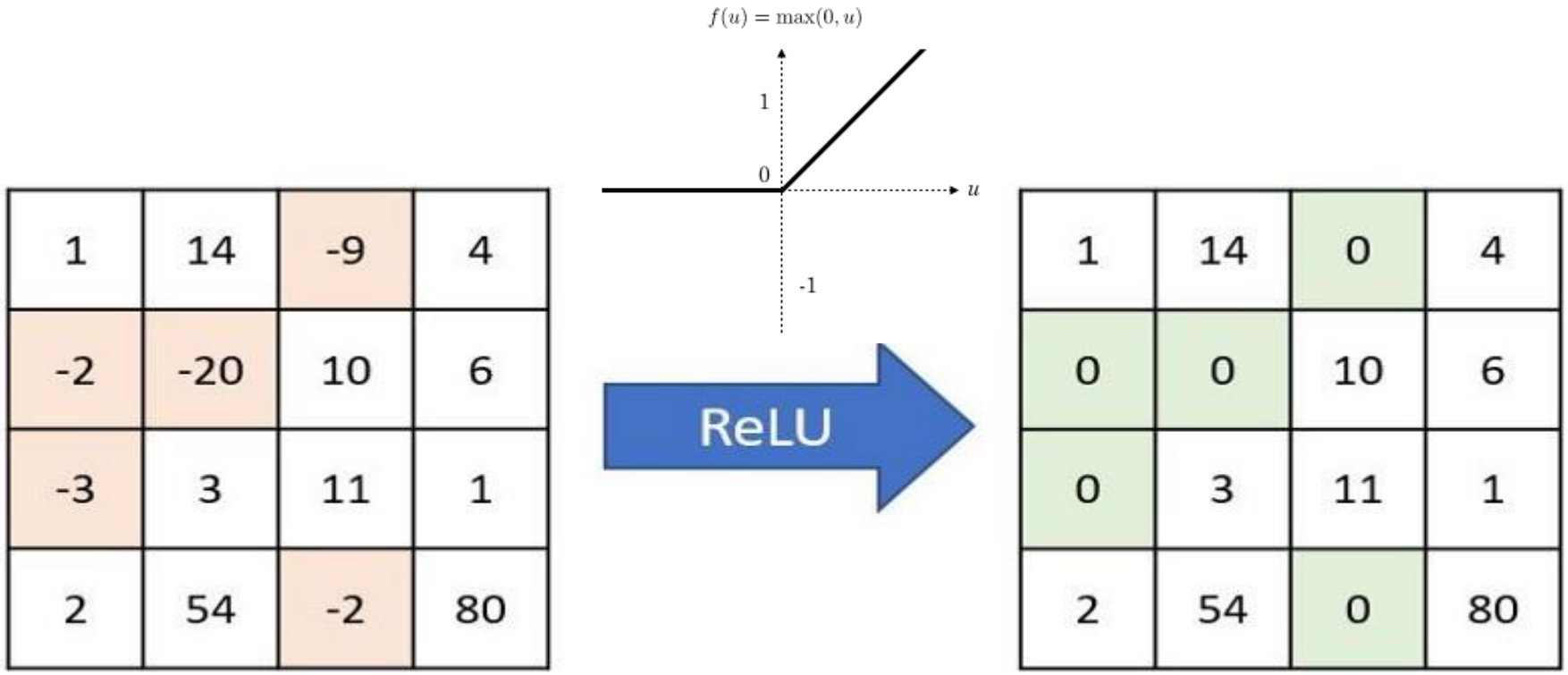
- Input volume: $n \times n \times d$
- Filter size: $f \times f \times d$
- Output: $\frac{n+2p-f}{s} + 1 \times \frac{n+2p-f}{s} + 1$

- k is the number of filters
- For two filters – two 4x4 outputs
- Stack the outputs together – 4x4x2
- If we are using M filters:
  – Output volume 4x4xM



ReLU Activation

- Activation functions add non-linearity to convolution

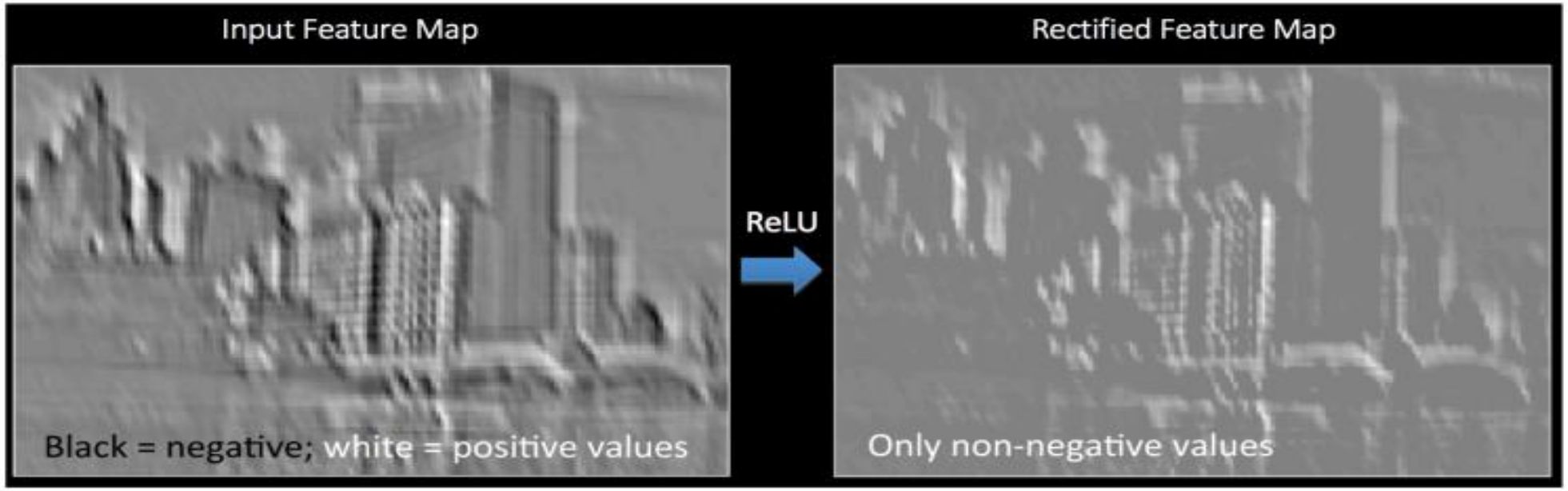


1	14	-9	4
-2	-20	10	6
-3	3	11	1
2	54	-2	80

1	14	0	4
0	0	10	6
0	3	11	1
2	54	0	80

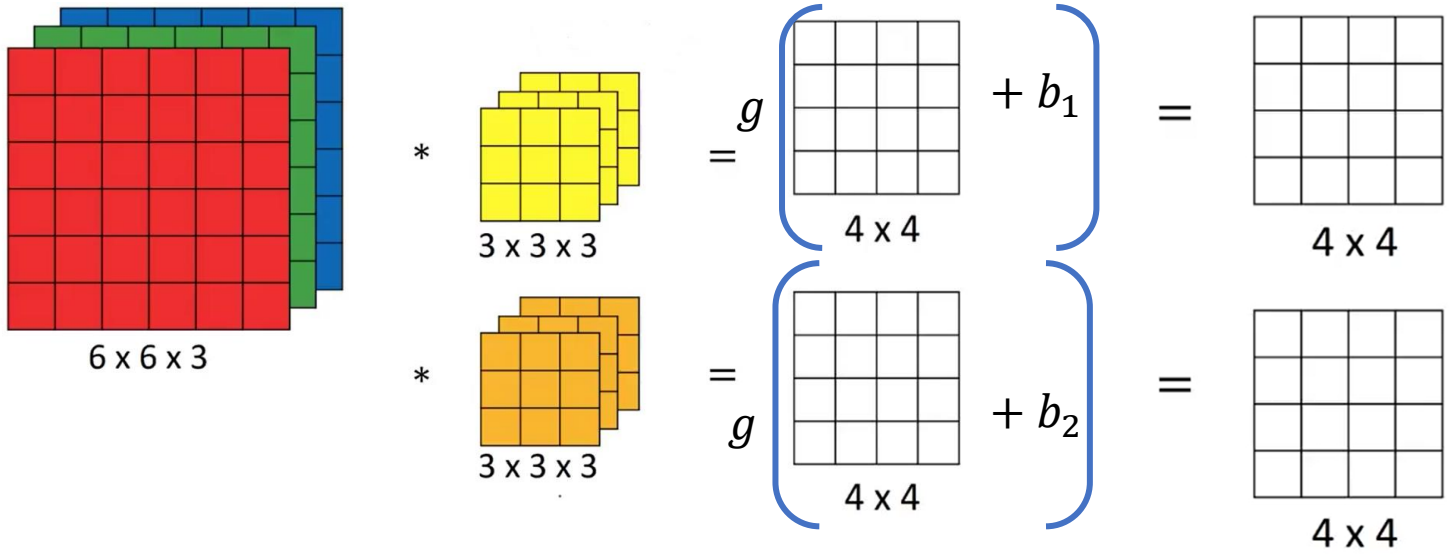
ReLU Activation

- Researchers found out that ***ReLU activation*** works better because the network is able to train a lot faster (because of the computational efficiency) without making a significant difference to the accuracy.
- The ReLU layer applies the function $f(x) = max(0, x)$ to all of the values in the input volume. In basic terms, it changes all the negative activations to 0.



One Convolution Layer

- Take the output of each filter
- Add a bias to the output values
- Apply activation function
- Stack the outputs together
- The process of going from 6x6x3 to 4x4x2 is equivalent to one layer of a ConvNet



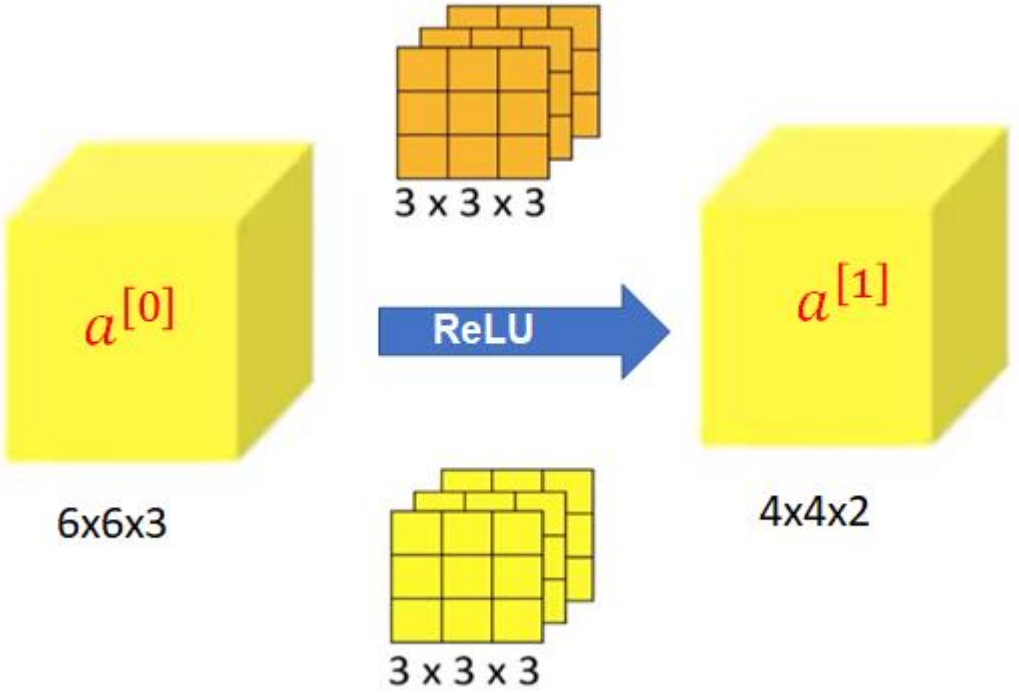
Learnable Parameters

What is the number of parameters to learn in this layer?

Each filter has 3x3x3 = 27 values + 1 bias
Total filters: 2
Total Params: 2 x (27+1) = 56

What is the number of parameters if we have 10 filters of 3x3x3 in the layer?

Each filter has 3x3x3 = 27 values + 1 bias
Total filters: 10
Total Params: 10 x (27+1) = 280

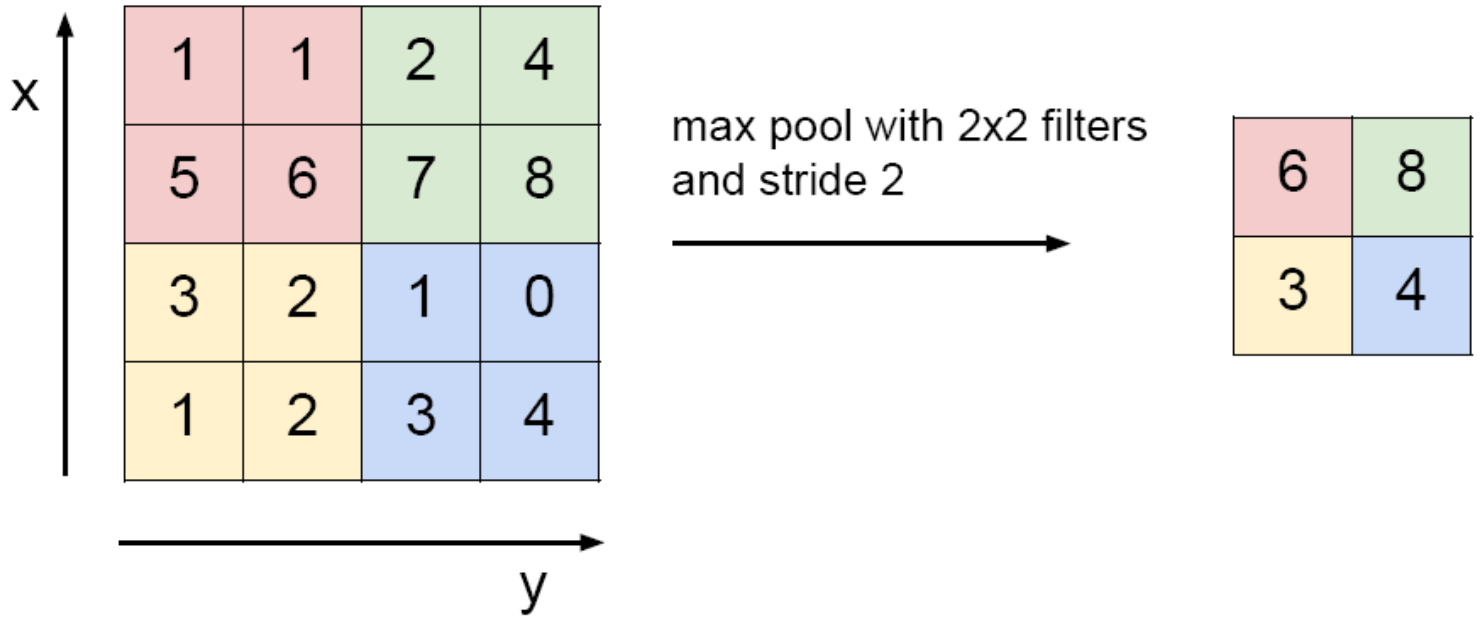


Learnable Parameters

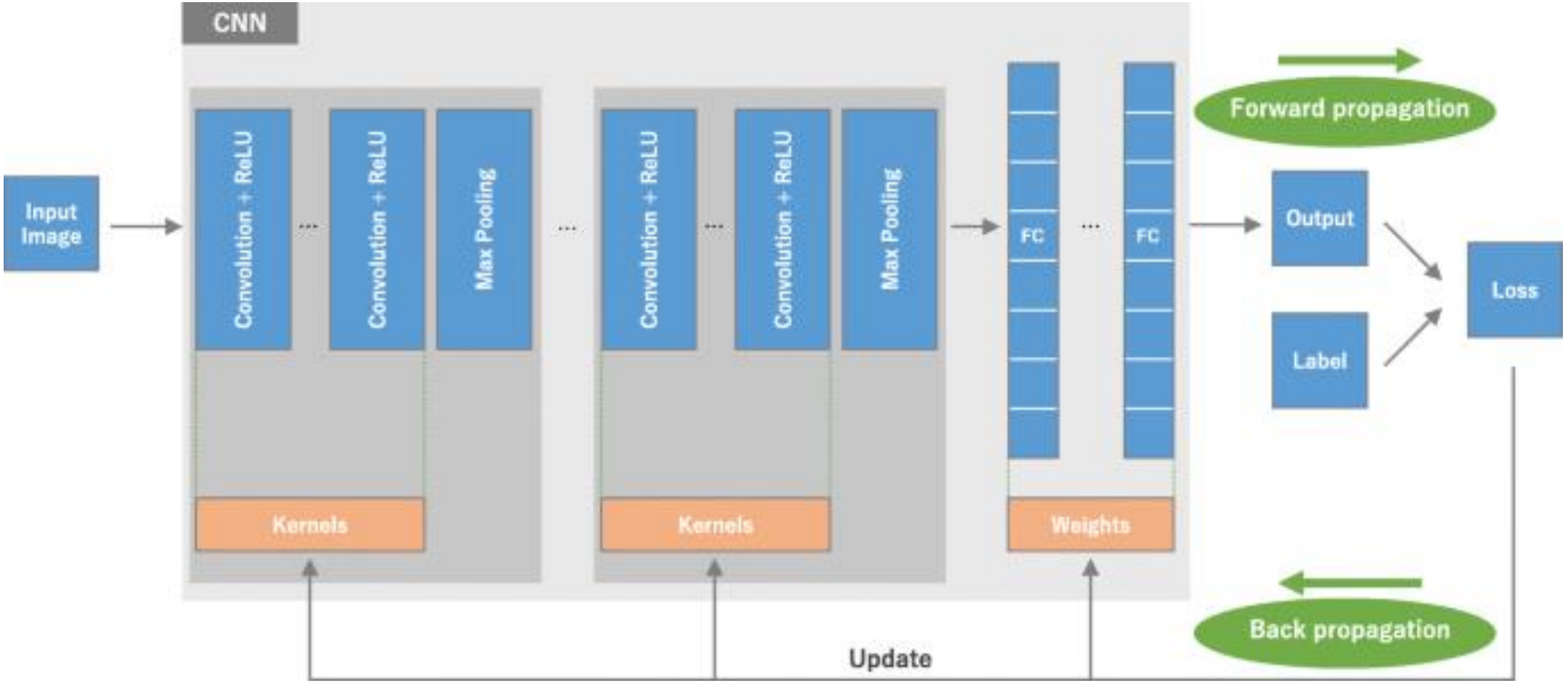
- No matter how big the image (input) is – The number of parameters remain the same
- Input of size 100 x 100 or 1000 x 1000 – Same number of parameters

Pooling – Max Pooling

- Pooling: Down sampling
- Given input of 4x4 – Reduce it to 2x2
- Idea: Take the input, break it into regions and keep the max value from each region

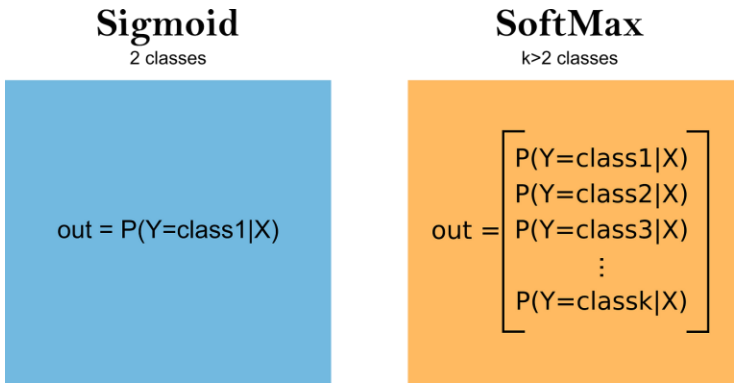


A Typical ConvNet Architecture



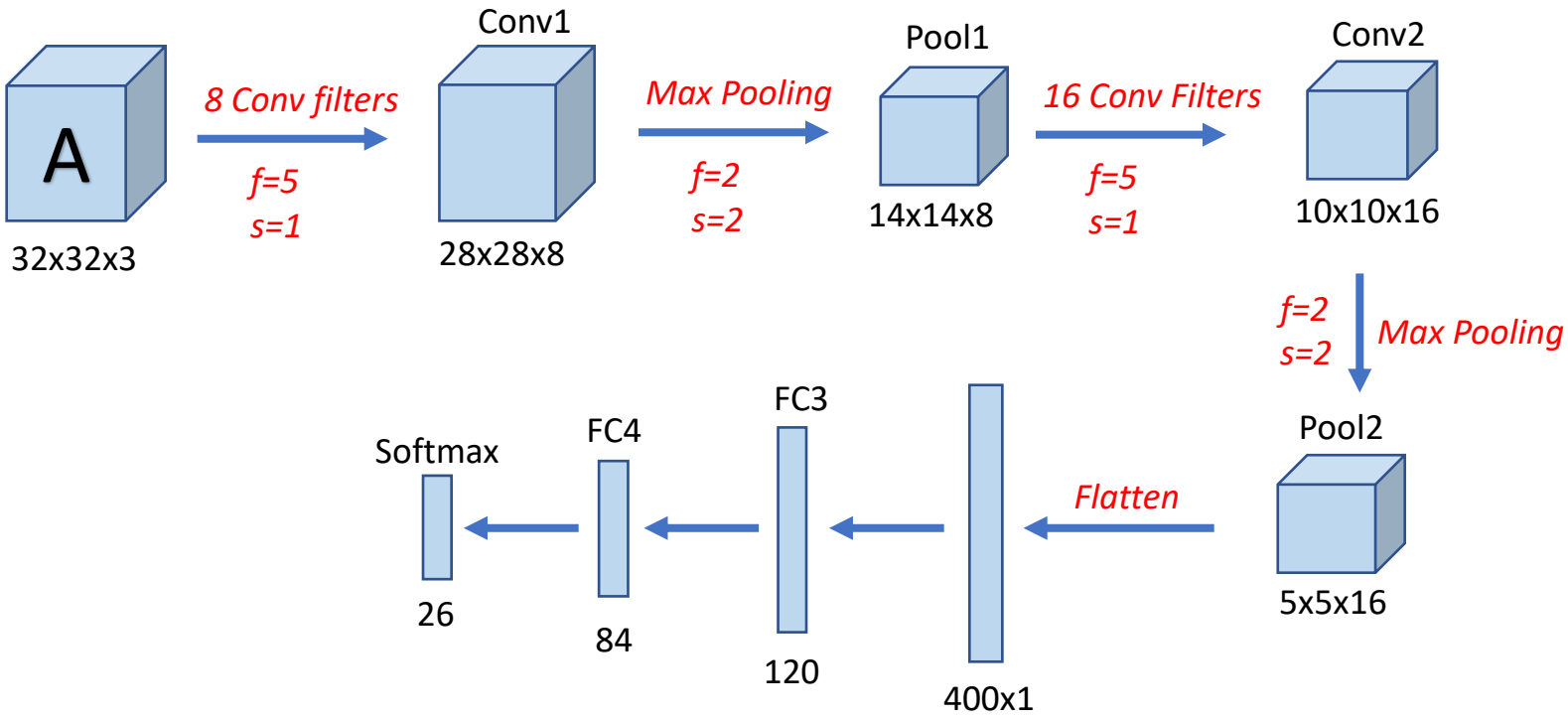
Fully Connected Layers

- Similar to a feed forward ANN
- The last FC layer outputs an N dimensional vector where N is the number of classes that the program has to choose from.
- For example, if we want a digit classification program, N would be 10 since there are 10 digits.
- Each number in this N dimensional vector represents the probability of a certain class.
- Output layer of FC can be a:
  - Sigmoid for Binary Classification
  - Softmax for Multiclassification



A Toy ConvNet

- Character Recognition – 32x32x3 Images



Conv – Pool – Conv – Pool – FC – FC – Softmax

A Toy ConvNet

	Activation Shape	Activation Size	Number of Parameters
Input:	(32,32,3)	3,072	0
CONV1 (f=5,s=1)	(28,28,8)	6,272	75x8+8 = 608
POOL1	(14,14,8)	1,568	0
CONV2(f=5,s=1)	(10,10,16)	1,600	200x16+16 = 3,216
POOL2	(5,5,16)	400	0
FC3	(120,1)	120	400x120+120 = 48,120
FC4	(84,1)	84	120x84+84 = 10,164
Softmax	(26,1)	26	84x26+26 = 2,210

Observations:

- Activation size goes down gradually
- Pooling layers have no parameters
- Conv layers have relatively fewer parameters
- Most of the parameters are in the fully connected layers

Implementation Hints

```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', name='conv_1',
                 input_shape=(150, 150, 3)))
model.add(MaxPooling2D((2, 2), name='maxpool_1'))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', name='conv_2'))
model.add(MaxPooling2D((2, 2), name='maxpool_2'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', name='conv_3'))
model.add(MaxPooling2D((2, 2), name='maxpool_3'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', name='conv_4'))
model.add(MaxPooling2D((2, 2), name='maxpool_4'))
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(512, activation='relu', name='dense_1'))
model.add(Dense(128, activation='relu', name='dense_2'))
model.add(Dense(1, activation='sigmoid', name='output'))
```

Data Augmentation

- Can help generate more training data by applying transformations (flip, rotate, crop etc.) to the existing data.
- Data augmentation can help create a more diverse and robust training set, which can reduce the risk of overfitting.

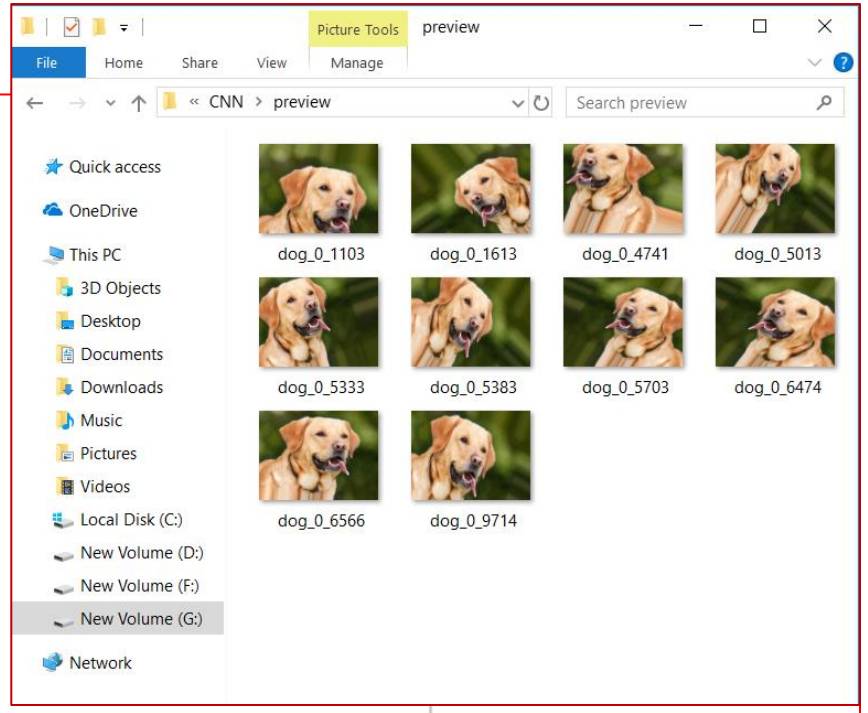
Data Augmentation in Keras

- In *keras*, we can perform all of these transformations using ImageDataGenerator.
- It has a big list of arguments which can be used to pre-process training data.

```
import cv2
import matplotlib.pyplot as plt
from keras.preprocessing.image import ImageDataGenerator

image_path="dog.jpg"
img = cv2.imread(image_path,cv2.IMREAD_COLOR)
img_rgb = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
print(img.shape)

plt.imshow(img_rgb)
plt.axis("off")
plt.show()
```

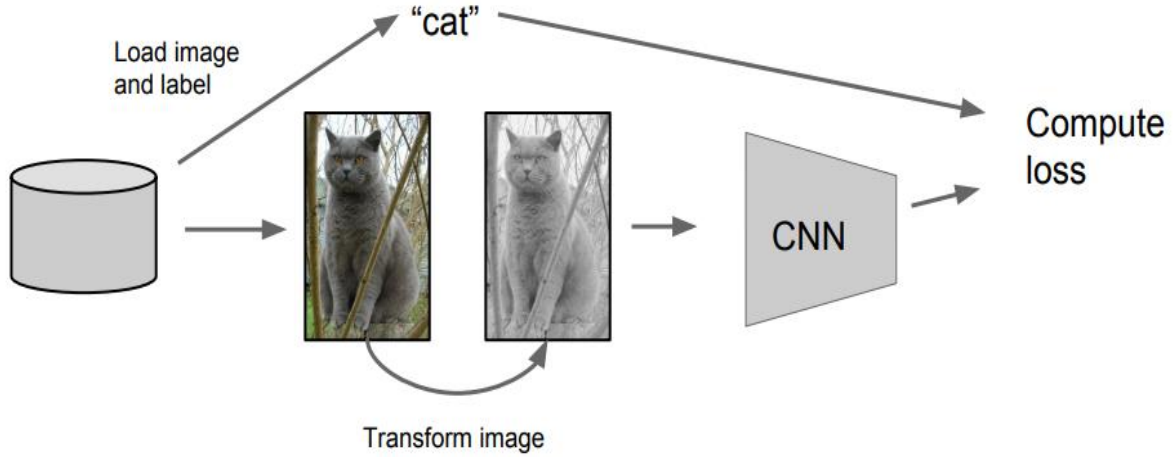
Data Augmentation in Keras

```
#Convert 3D Data to 4D: Add one dimensions
img_rgb = img_rgb.reshape((1,)+img_rgb.shape)
print(img_rgb.shape)

datagen = ImageDataGenerator(rotation_range=40,
                             width_shift_range=0.2,
                             height_shift_range=0.2,
                             zoom_range=0.2,
                             horizontal_flip=True,
                             fill_mode='nearest')

i = 0
for batch in datagen.flow(img_rgb,save_to_dir='preview', save_prefix='dog', save_format='jpeg'):
    i += 1
    if i > 9:
        break
```



Data Augmentation in Keras

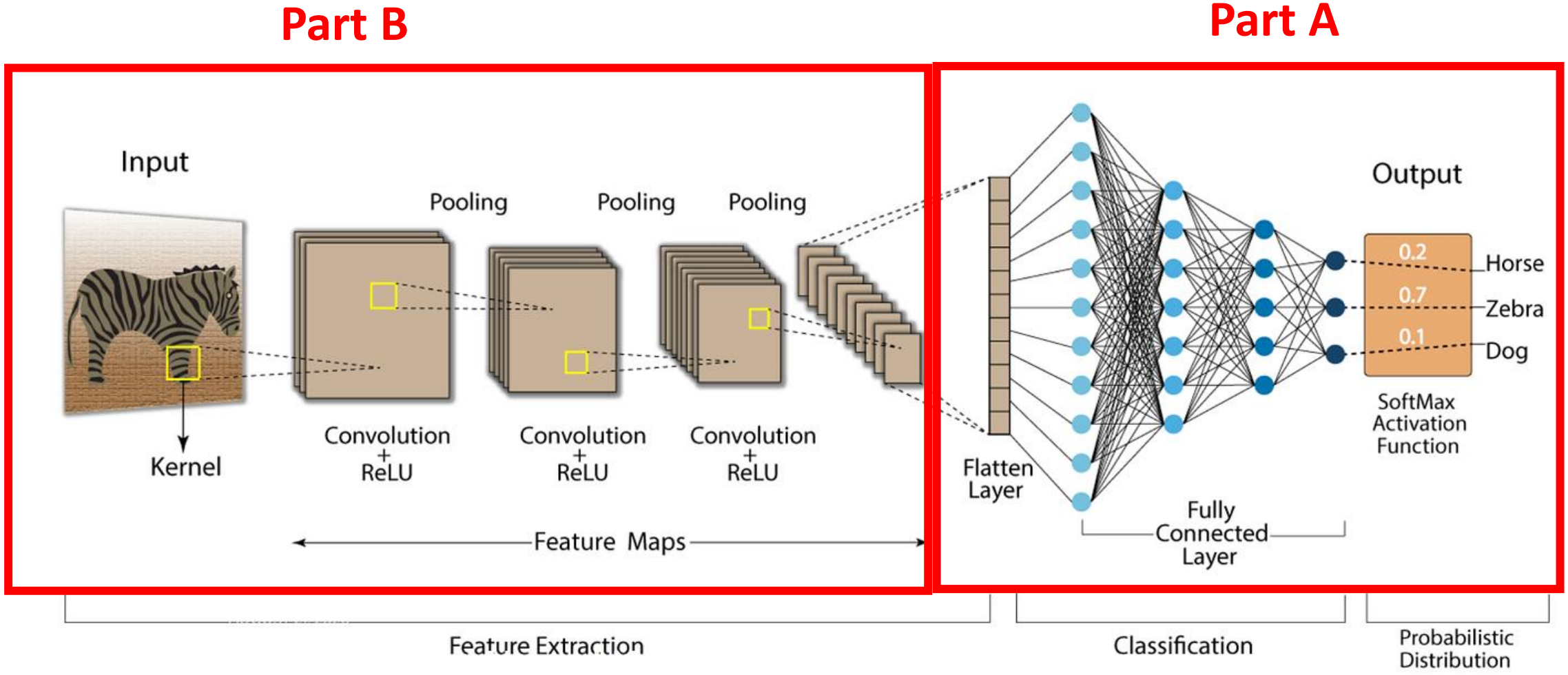
- Saving copies of images to disk is not desirable
- Make copies and pass them directly to the learning algorithm



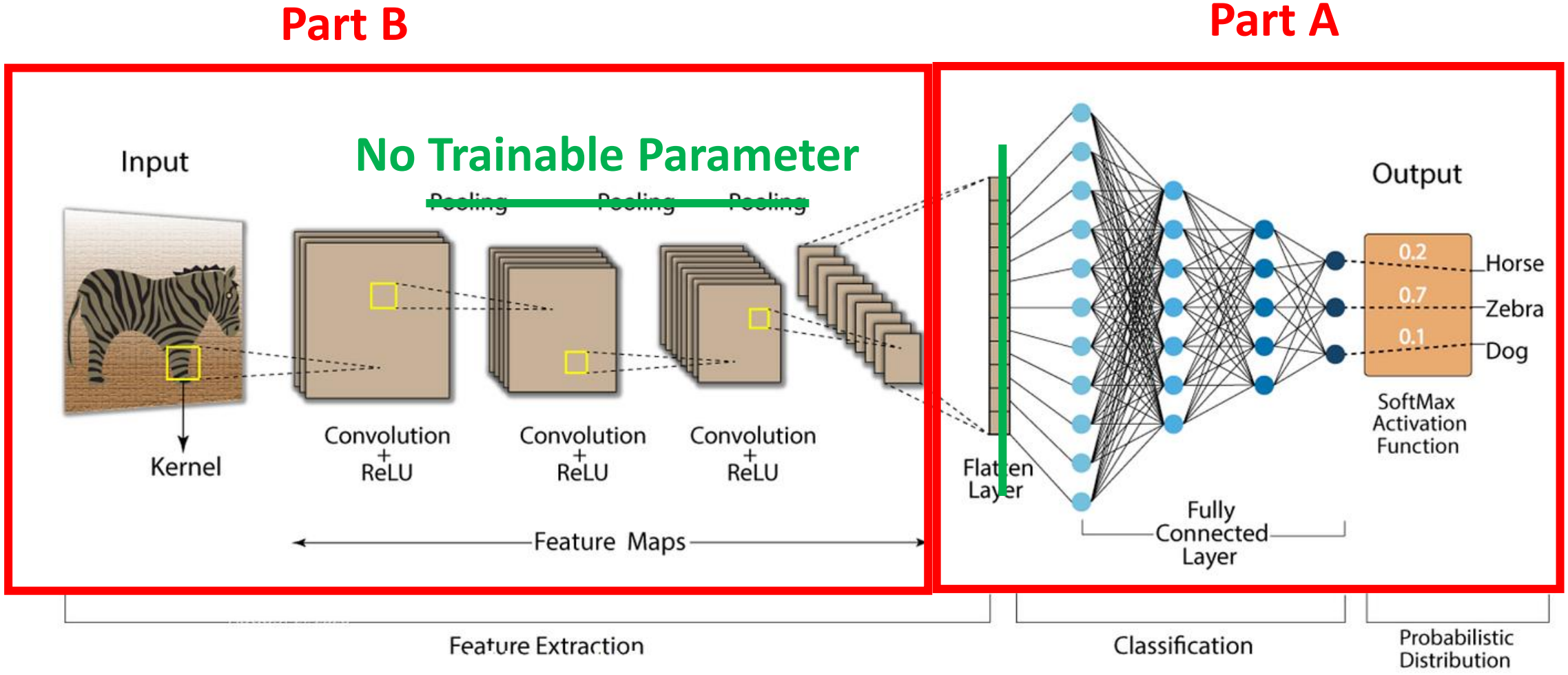

```
datagen = ImageDataGenerator(rotation_range=40,
                             width_shift_range=0.2,
                             height_shift_range=0.2,
                             zoom_range=0.2,
                             horizontal_flip=True,
                             fill_mode='nearest')

datagen.fit(X_train)
```

```
history = model.fit_generator(datagen.flow(X_train,Y_train,batch_size=128),
                              steps_per_epoch=X_train.shape[0]/128,
                              epochs = 2,
                              verbose=1)
```

Back Propagation in CNN



Back Propagation in CNN



Back Propagation in CNN

- In the backward pass, we get the loss gradient with respect to the next layer.
- In CNNs the loss gradient is computed w.r.t the input and also w.r.t the filter.
- Non-trainable parameters like in pooling and flattening only reshaping is required.

Back Propagation in CNN

Convolution Backprop with single Stride

- To understand the computation of loss gradient w.r.t input, let us use the following example:

- Horizontal and vertical stride = 1

X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}

Input **X**

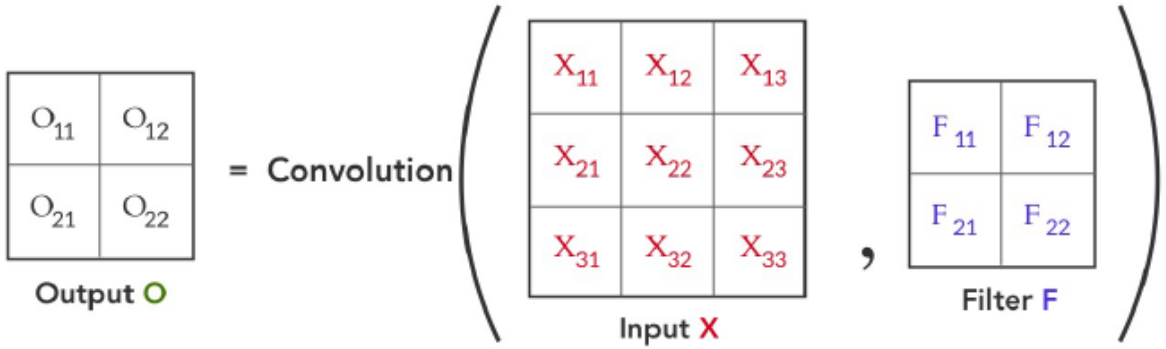
F_{11}	F_{12}
F_{21}	F_{22}

Filter **F**

Back Propagation in CNN

Convolution Forward Pass

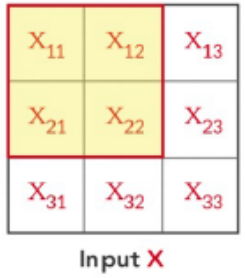
- Convolution between Input X and Filter F, gives us an output O. This can be represented as:



Back Propagation in CNN

Convolution Forward Pass

- Convolution between Input X and Filter F, gives us an output O. This can be represented as:



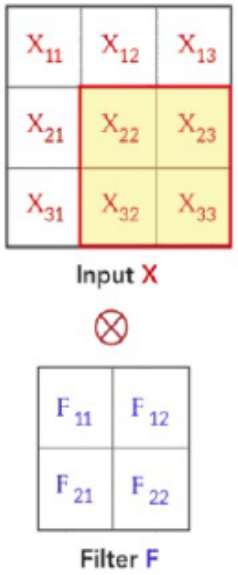
$X_{11}F_{11}$	$X_{12}F_{12}$	X_{13}
$X_{21}F_{21}$	$X_{22}F_{22}$	X_{23}
X_{31}	X_{32}	X_{33}

$$O_{11} = X_{11}F_{11} + X_{12}F_{12} + X_{21}F_{21} + X_{22}F_{22}$$

Back Propagation in CNN

Convolution Forward Pass

- Convolution between Input X and Filter F, gives us an output O. This can be represented as:



X_{11}	X_{12}	X_{13}
X_{21}	$X_{22}F_{11}$	$X_{23}F_{12}$
X_{31}	$X_{32}F_{21}$	$X_{33}F_{22}$

$$O_{11} = X_{11}F_{11} + X_{12}F_{12} + X_{21}F_{21} + X_{22}F_{22}$$

$$O_{12} = X_{12}F_{11} + X_{13}F_{12} + X_{22}F_{21} + X_{23}F_{22}$$

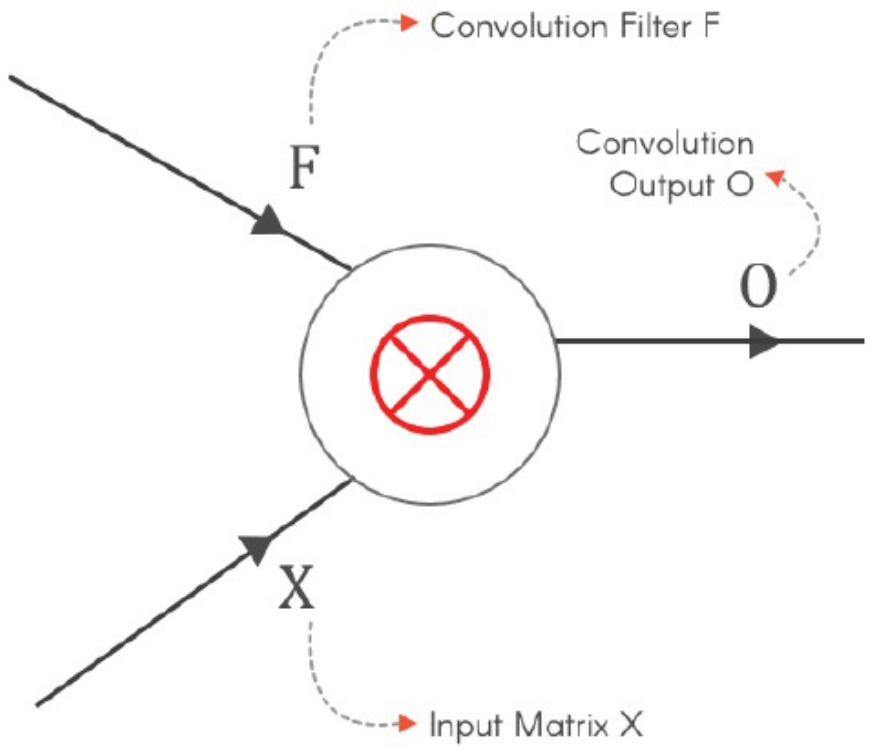
$$O_{21} = X_{21}F_{11} + X_{22}F_{12} + X_{31}F_{21} + X_{32}F_{22}$$

$$O_{22} = X_{22}F_{11} + X_{23}F_{12} + X_{32}F_{21} + X_{33}F_{22}$$

Back Propagation in CNN

Loss gradient

- We want to calculate the gradients wrt to input ‘X’ and filter ‘F’



Back Propagation in CNN

Loss gradient w.r.t the filter

We can use the chain rule to obtain the gradient wrt the filter as shown in the equation.

$$\frac{\partial L}{\partial F} = \frac{\partial L}{\partial O} * \frac{\partial O}{\partial F}$$

Gradient to update Filter F

Loss Gradient from previous layer

Local Gradients

For every element of F

$$\frac{\partial L}{\partial F_i} = \sum_{k=1}^{M} \frac{\partial L}{\partial O_k} * \frac{\partial O_k}{\partial F_i}$$

Back Propagation in CNN

Loss gradient w.r.t the filter

We can expand the chain rule summation as:

For every element of F

$$\frac{\partial L}{\partial F_i} = \sum_{k=1}^{M} \frac{\partial L}{\partial O_k} * \frac{\partial O_k}{\partial F_i}$$

$$\frac{\partial L}{\partial F_{11}} = \frac{\partial L}{\partial O_{11}} * \frac{\partial O_{11}}{\partial F_{11}} + \frac{\partial L}{\partial O_{12}} * \frac{\partial O_{12}}{\partial F_{11}} + \frac{\partial L}{\partial O_{21}} * \frac{\partial O_{21}}{\partial F_{11}} + \frac{\partial L}{\partial O_{22}} * \frac{\partial O_{22}}{\partial F_{11}}$$

$$\frac{\partial L}{\partial F_{12}} = \frac{\partial L}{\partial O_{11}} * \frac{\partial O_{11}}{\partial F_{12}} + \frac{\partial L}{\partial O_{12}} * \frac{\partial O_{12}}{\partial F_{12}} + \frac{\partial L}{\partial O_{21}} * \frac{\partial O_{21}}{\partial F_{12}} + \frac{\partial L}{\partial O_{22}} * \frac{\partial O_{22}}{\partial F_{12}}$$

$$\frac{\partial L}{\partial F_{21}} = \frac{\partial L}{\partial O_{11}} * \frac{\partial O_{11}}{\partial F_{21}} + \frac{\partial L}{\partial O_{12}} * \frac{\partial O_{12}}{\partial F_{21}} + \frac{\partial L}{\partial O_{21}} * \frac{\partial O_{21}}{\partial F_{21}} + \frac{\partial L}{\partial O_{22}} * \frac{\partial O_{22}}{\partial F_{21}}$$

$$\frac{\partial L}{\partial F_{22}} = \frac{\partial L}{\partial O_{11}} * \frac{\partial O_{11}}{\partial F_{22}} + \frac{\partial L}{\partial O_{12}} * \frac{\partial O_{12}}{\partial F_{22}} + \frac{\partial L}{\partial O_{21}} * \frac{\partial O_{21}}{\partial F_{22}} + \frac{\partial L}{\partial O_{22}} * \frac{\partial O_{22}}{\partial F_{22}}$$

Back Propagation in CNN

Loss gradient w.r.t the filter

- Replacing the local gradients of the filter i.e, $\frac{\partial O_i}{\partial F_i}$, we get this:

$$\begin{bmatrix} \frac{\partial L}{\partial F_{11}} & \frac{\partial L}{\partial F_{12}} \\ \frac{\partial L}{\partial F_{21}} & \frac{\partial L}{\partial F_{22}} \end{bmatrix} = \text{Convolution} \left(\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix}, \begin{bmatrix} \frac{\partial L}{\partial O_{11}} & \frac{\partial L}{\partial O_{12}} \\ \frac{\partial L}{\partial O_{21}} & \frac{\partial L}{\partial O_{22}} \end{bmatrix} \right)$$

where

$$\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix} = \text{Input X} \qquad \begin{bmatrix} \frac{\partial L}{\partial O_{11}} & \frac{\partial L}{\partial O_{12}} \\ \frac{\partial L}{\partial O_{21}} & \frac{\partial L}{\partial O_{22}} \end{bmatrix} = \frac{\partial L}{\partial O} \text{ Loss gradient from previous layer}$$

$$\frac{\partial L}{\partial F_{11}} = \frac{\partial L}{\partial O_{11}} * X_{11} + \frac{\partial L}{\partial O_{12}} * X_{12} + \frac{\partial L}{\partial O_{21}} * X_{21} + \frac{\partial L}{\partial O_{22}} * X_{22}$$

$$\frac{\partial L}{\partial F_{12}} = \frac{\partial L}{\partial O_{11}} * X_{12} + \frac{\partial L}{\partial O_{12}} * X_{13} + \frac{\partial L}{\partial O_{21}} * X_{22} + \frac{\partial L}{\partial O_{22}} * X_{23}$$

$$\frac{\partial L}{\partial F_{21}} = \frac{\partial L}{\partial O_{11}} * X_{21} + \frac{\partial L}{\partial O_{12}} * X_{22} + \frac{\partial L}{\partial O_{21}} * X_{31} + \frac{\partial L}{\partial O_{22}} * X_{32}$$

$$\frac{\partial L}{\partial F_{22}} = \frac{\partial L}{\partial O_{11}} * X_{22} + \frac{\partial L}{\partial O_{12}} * X_{23} + \frac{\partial L}{\partial O_{21}} * X_{32} + \frac{\partial L}{\partial O_{22}} * X_{33}$$

Back Propagation in CNN

Loss gradient w.r.t the filter

- If you closely look at it, this represents an operation we are quite familiar with. We can represent it as a **convolution operation between input X** and loss gradient **∂L/∂O as shown below:**

$$\begin{bmatrix} \frac{\partial L}{\partial F_{11}} & \frac{\partial L}{\partial F_{12}} \\ \frac{\partial L}{\partial F_{21}} & \frac{\partial L}{\partial F_{22}} \end{bmatrix} = \text{Convolution} \left(\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix}, \begin{bmatrix} \frac{\partial L}{\partial O_{11}} & \frac{\partial L}{\partial O_{12}} \\ \frac{\partial L}{\partial O_{21}} & \frac{\partial L}{\partial O_{22}} \end{bmatrix} \right)$$

where

$$\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix} = \text{Input X} \qquad \begin{bmatrix} \frac{\partial L}{\partial O_{11}} & \frac{\partial L}{\partial O_{12}} \\ \frac{\partial L}{\partial O_{21}} & \frac{\partial L}{\partial O_{22}} \end{bmatrix} = \frac{\partial L}{\partial O}$$ Loss gradient from previous layer

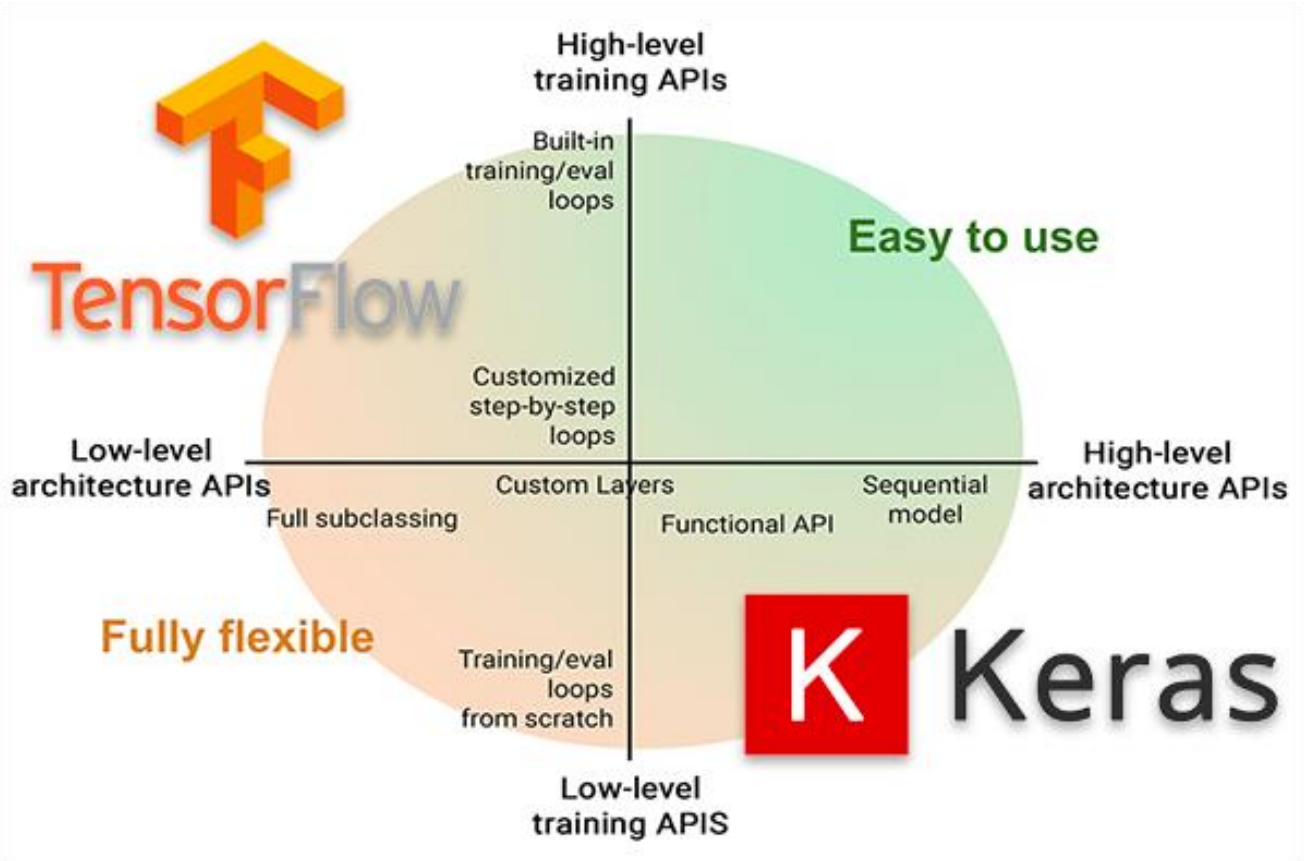
$$\frac{\partial L}{\partial F_{11}} = \frac{\partial L}{\partial O_{11}} * X_{11} + \frac{\partial L}{\partial O_{12}} * X_{12} + \frac{\partial L}{\partial O_{21}} * X_{21} + \frac{\partial L}{\partial O_{22}} * X_{22}$$

$$\frac{\partial L}{\partial F_{12}} = \frac{\partial L}{\partial O_{11}} * X_{12} + \frac{\partial L}{\partial O_{12}} * X_{13} + \frac{\partial L}{\partial O_{21}} * X_{22} + \frac{\partial L}{\partial O_{22}} * X_{23}$$

$$\frac{\partial L}{\partial F_{21}} = \frac{\partial L}{\partial O_{11}} * X_{21} + \frac{\partial L}{\partial O_{12}} * X_{22} + \frac{\partial L}{\partial O_{21}} * X_{31} + \frac{\partial L}{\partial O_{22}} * X_{32}$$

$$\frac{\partial L}{\partial F_{22}} = \frac{\partial L}{\partial O_{11}} * X_{22} + \frac{\partial L}{\partial O_{12}} * X_{23} + \frac{\partial L}{\partial O_{21}} * X_{32} + \frac{\partial L}{\partial O_{22}} * X_{33}$$

Tools Required



Acknowledgements

Some of the content of these slides is taken from:

- www.coursera.com
- https://ai.stanford.edu/~syyeung/cvweb/tutorial4.html
- Slides by CS231n Winter 2016 – Andrej Karpathy
- Convolutional Neural Networks (CNNs): An Illustrated Explanation, Abhineet Saxena
- An Intuitive Explanation of Convolutional Neural Networks
- A Beginner's Guide To Understanding Convolutional Neural Networks, Adit Deshpande
- Deep Learning Andrew Ng