GENERAL ASSEMBLY

CONVOLUTIONAL NEURAL NETWORKS I

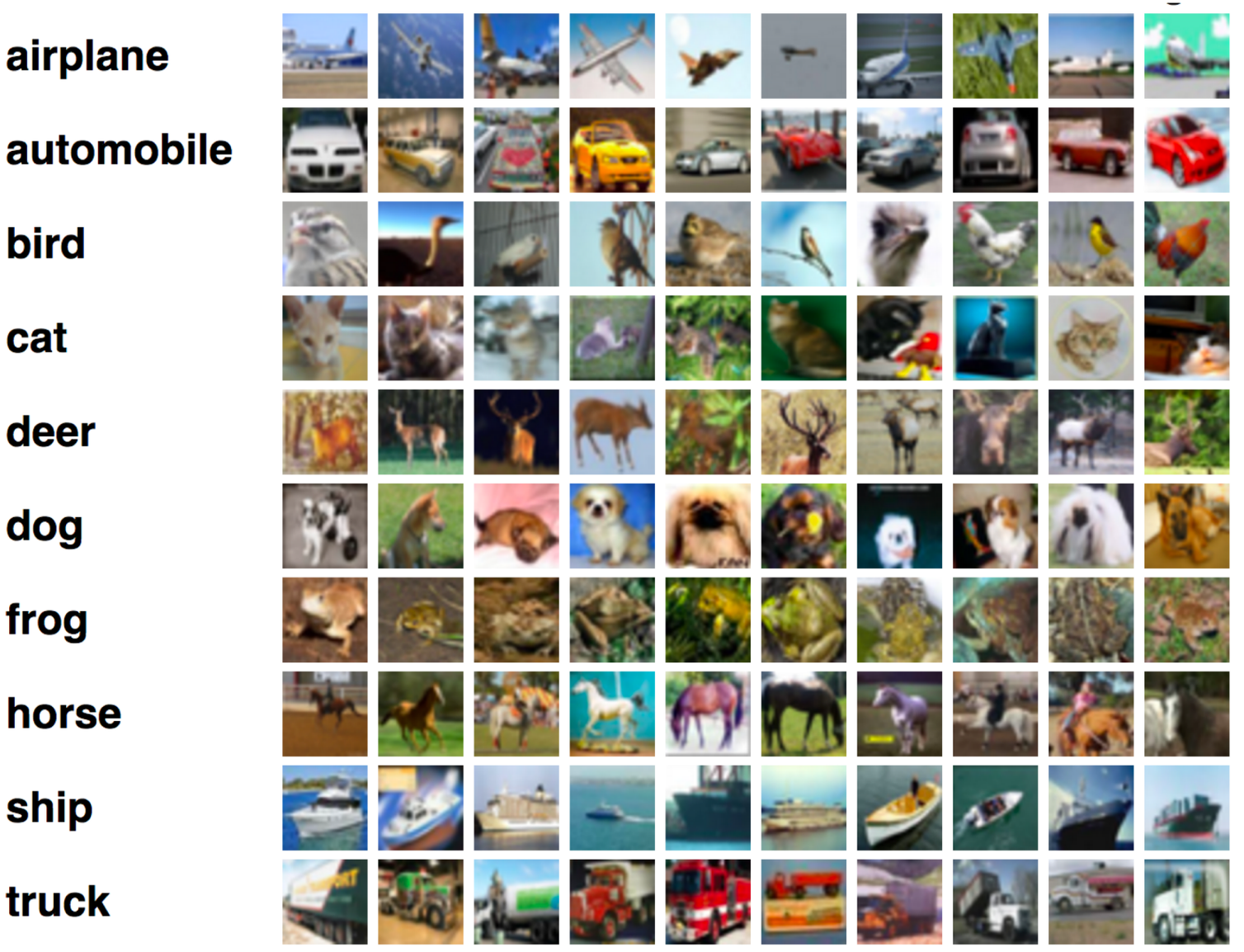
Matt Brems

Data Science Immersive, GA DC

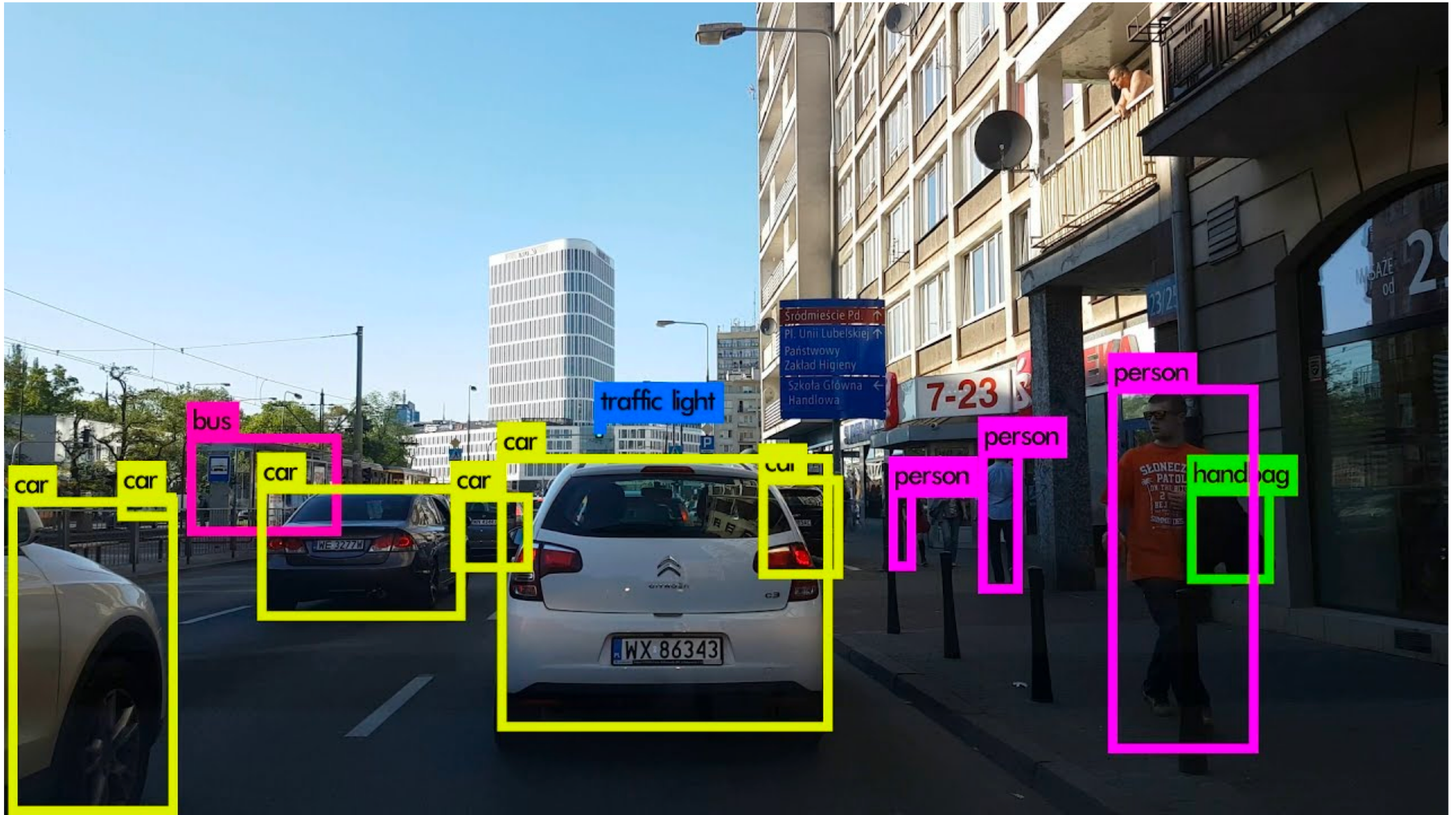
LEARNING OBJECTIVES

- By the end of this lesson, students should be able to:
  - Identify use cases for convolutional neural networks.
  - Understand how edge detection works in CNNs.
  - Define and describe padding and strided convolutions.
  - Understand how convolutions operate on volumes.
  - Define pooling and implement max pooling.

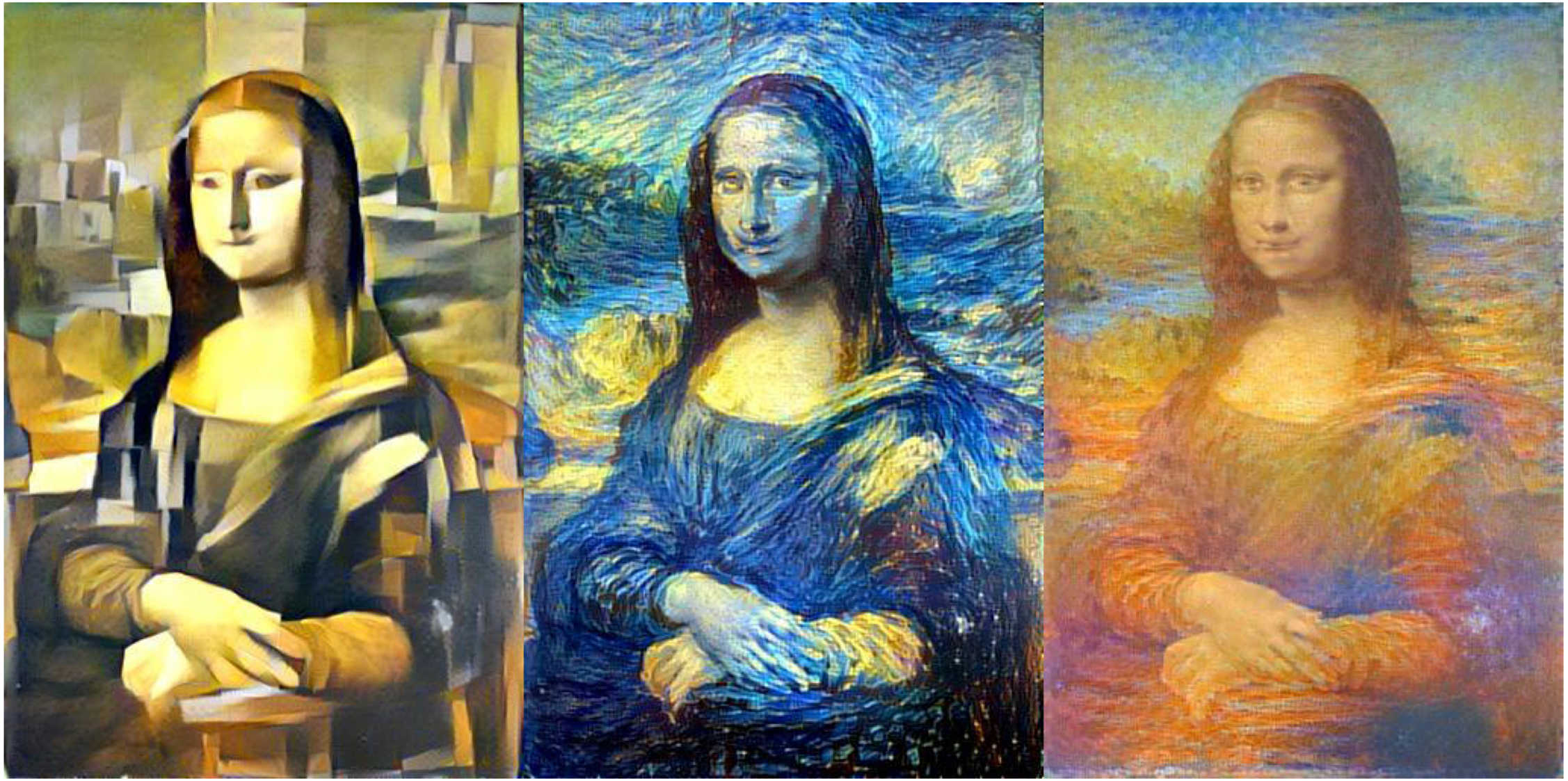
IMAGE CLASSIFICATION



OBJECT DETECTION



NEURAL STYLE TRANSFER



LET'S TRY USING DENSELY CONNECTED LAYERS

- Suppose I have a grayscale image that's 64 pixels x 64 pixels.
- My first hidden layer contains 100 nodes.
- How many parameters do we need to learn here?

LET'S TRY USING DENSELY CONNECTED LAYERS

- Instead of a grayscale image, we have an RGB color image.

- Instead of 64 x 64, we have 1,000 x 1,000.

PROBLEMS ASSOCIATED WITH THIS MANY PARAMETERS

- If we have tens of thousands or hundreds of thousands of parameters to learn:
  - We better have a very large sample size.
  - Learning will be slow.
  - We're almost surely going to overfit our model to the data.

- We need alternatives.

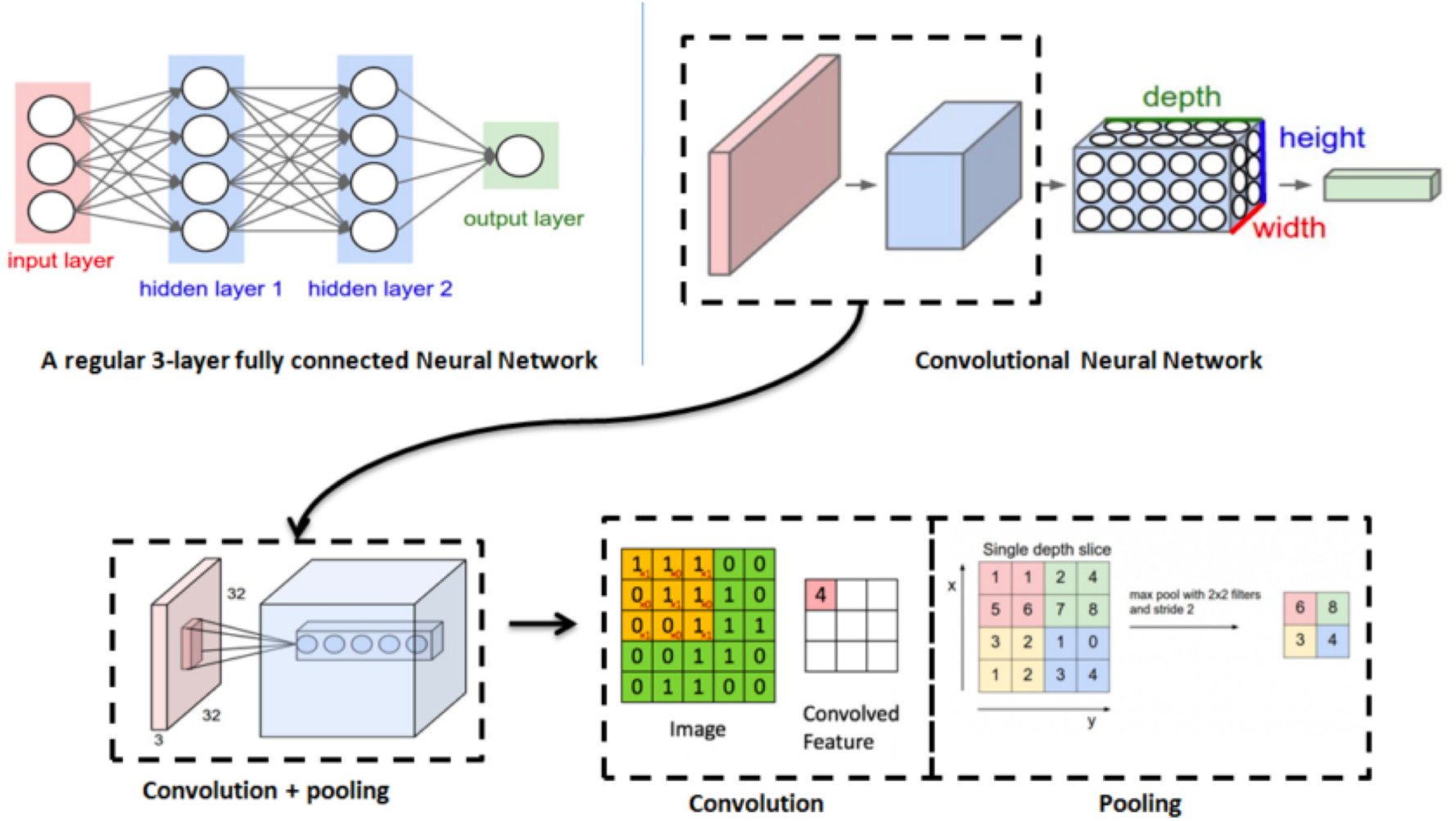
CONVOLUTIONAL NEURAL NETWORKS

- Convolutional neural networks (CNNs) will allow us to learn far, far fewer parameters.

- It takes advantage of the structure in images by making calculations on pixels that are close to one another.

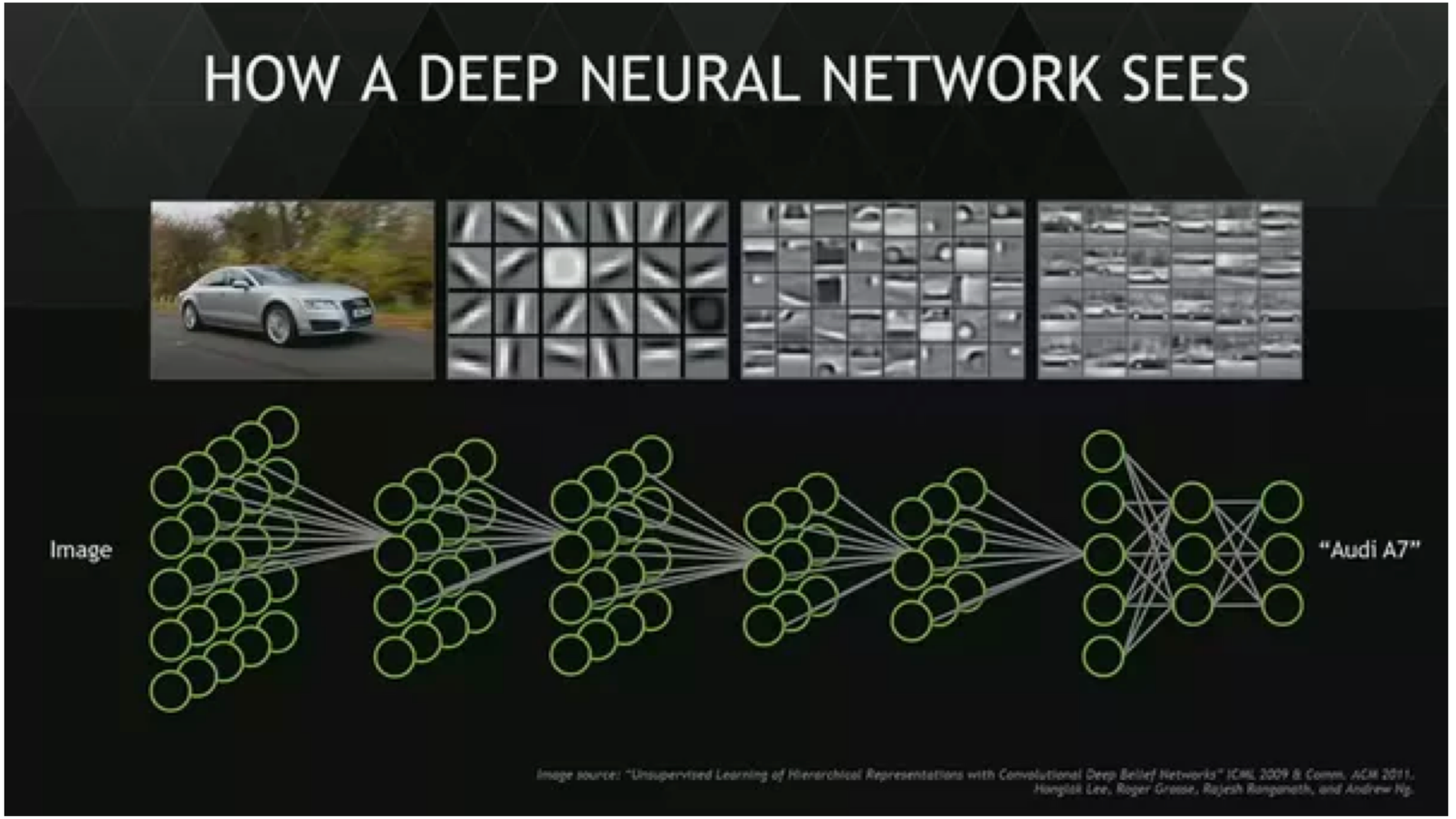
- CNNs will generally consist of:
  - one or more convolutional layers,
  - one or more pooling layers, and
  - one or more fully connected layers.

CNN VISUAL



Source: http://ufldl.stanford.edu/tutorial/images/Convolution_schematic.gif, http://cs231n.github.io/assets/nn1/neural_net2.jpeg, http://cs231n.github.io/assets/cnn/depthcol.jpeg, http://cs231n.github.io/assets/cnn/maxpool.jpeg

DETECT EDGES, THEN STRUCTURES, THEN OBJECTS



CONVOLUTIONAL LAYER

- Convolutional layers are used to detect “things” in images.
  - Early convolutional layers detect edges (i.e. edge of nose).
  - Later layers detect structures (i.e. nose).
  - Final layers detect entire objects (i.e. faces).

EDGE DETECTION – INTRODUCTION

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

*

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

=

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

EDGE DETECTION – SQUARE 1

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

*

1	**0**	**-1**
1	**0**	**-1**
1	**0**	**-1**

=

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

EDGE DETECTION – SQUARE 2

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

*

1	**0**	**-1**
1	**0**	**-1**
1	**0**	**-1**

=

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

EDGE DETECTION – YOU TRY!

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

*

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

=

EDGE DETECTION – CHANGE THE FILTER

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

*

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

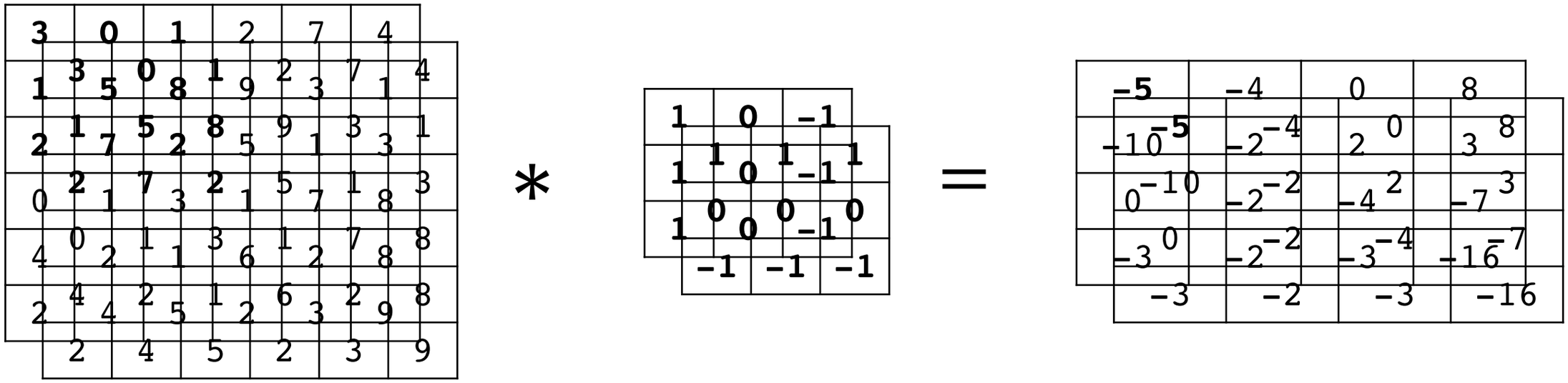
=

CONVOLUTIONAL FILTERS

- The two filters we used were geared toward detecting horizontal and vertical lines.

- Source: http://aishack.in/tutorials/image-convolution-examples/

EDGE DETECTION – MULTIPLE FILTERS



CONVOLUTIONAL FILTERS

- Other convolutional filters exist for different types of transformations.

- We can also create a filter like this

w1	w2	w3
w4	w5	w6
w7	w8	w9

  that we can update through backpropagation.
  - While we'll never do this by hand, it's useful to have a convolutional filter that can do more than to blur images or to learn straight lines.

CORNERS AND EDGES

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

*

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

=

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

- How many times does each corner get included in the final result?

- What about edges?

PADDING

- As we take an image and apply a filter to it, we notice that our dimensions get smaller.
  - A 6x6 image and 3x3 filter yields a 4x4 result.
  - An $n \times n$ image and $f \times f$ filter yields an $(n - f + 1) \times (n - f + 1)$ result.
    - As the size of our filter increases, the size of our result decreases.

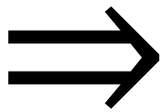
- "Our dimensions get smaller" = we discard more information!

- **Padding** addresses this issue.

PADDING

- To pad an image, add a frame of 0s around the image.

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



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

CORNERS AND EDGES

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

*

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

=

- Note that the “old corners” get included more frequently.

PADDING

- The “valid” convolution is where we have zero padding.

- The “same” convolution is where we set $p = \frac{f-1}{2}$.
  - This ensures that our result will be of the same dimensions as our original image.
  - f is usually odd, which allows for us to have a “central” position in the image.
    - Convenient for distance, landmarks, etc.

STRIDE = 1

- The last component of a convolutional layer is the **stride**.
- This determines how large our step (stride) is when moving the filter.

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

$*$

1	0	−1
1	0	−1
1	0	−1

$=$

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

STRIDE = 2

- The last component of a convolutional layer is the **stride**.
- This determines how large our step (stride) is when moving the filter.

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

$*$

1	0	−1
1	0	−1
1	0	−1

$=$

−5	0
0	−4

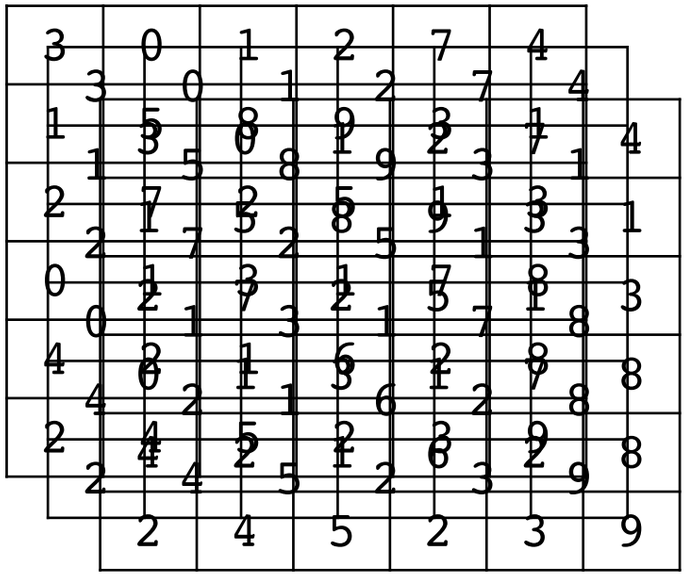
WHAT ABOUT COLOR?

- Thus far, we've assumed a grayscale image, where each feature is a number representing how dark/light a given pixel is.

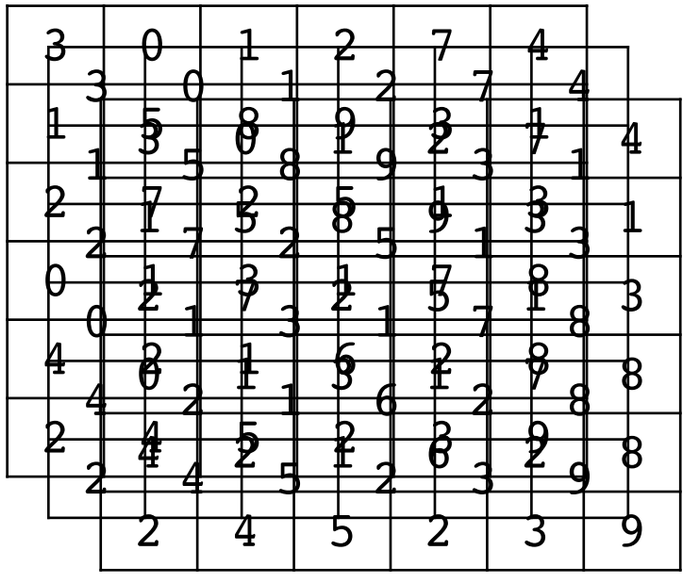
- If we want to process a color image (which, let's be honest, is what we'll want to do), then we need to represent our image differently.
  - We represent it with **volume**.

FROM GRAYSCALE TO COLOR (RGB)

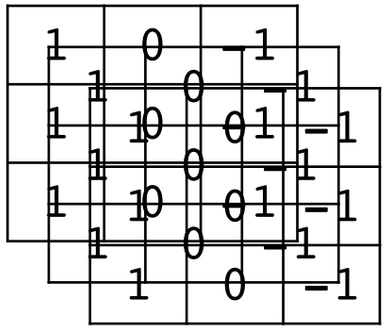
3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9



FROM GRAYSCALE TO COLOR (RGB)



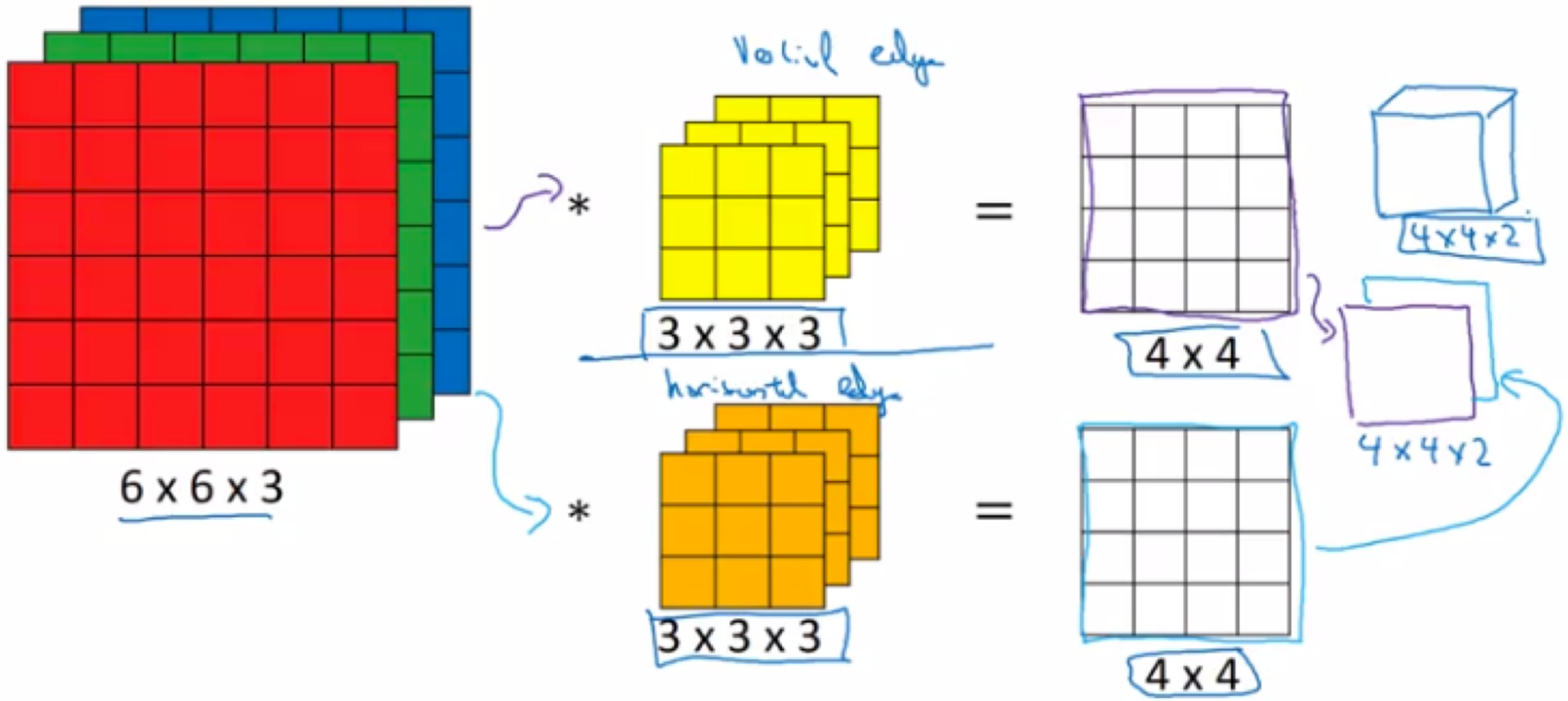
*



=

−15	−12	0	24
−30	−6	6	9
0	−6	−12	−21
−9	−6	−9	−48

FROM GRAYSCALE TO COLOR (RGB) – MULTIPLE FILTERS



ONE CONVOLUTIONAL LAYER

CONVOLUTIONAL NEURAL NETWORKS

- Convolutional neural networks (CNNs) will allow us to learn far, far fewer parameters.

- It takes advantage of the structure in images by making calculations on pixels that are close to one another.

- CNNs will generally consist of:
  - one or more convolutional layers,
  - one or more pooling layers, and
  - one or more fully connected layers.

POOLING LAYER

- The **pooling layer** will allow us to substantially compress the outputs from the convolutional layer.

1	3	2	1
2	9	1	1
1	3	2	3
5	6	1	2

- We generally do **max pooling**.

POOLING LAYER

- We generally do **max pooling**.
  - It empirically works well.
  - It identifies strong features.
  - We don’t have any parameters to learn; no need for gradient descent!

- Average pooling exists but is infrequently used.

- We pool independently across channels. (If we applied three filters and had three separate channels, pooling doesn’t change the number of channels.)

PUTTING ALL OF THE LAYERS TOGETHER: EXAMPLE CNN

- Goal: handwritten digit recognition in RGB; 32 x 32 image.

WRAP-UP

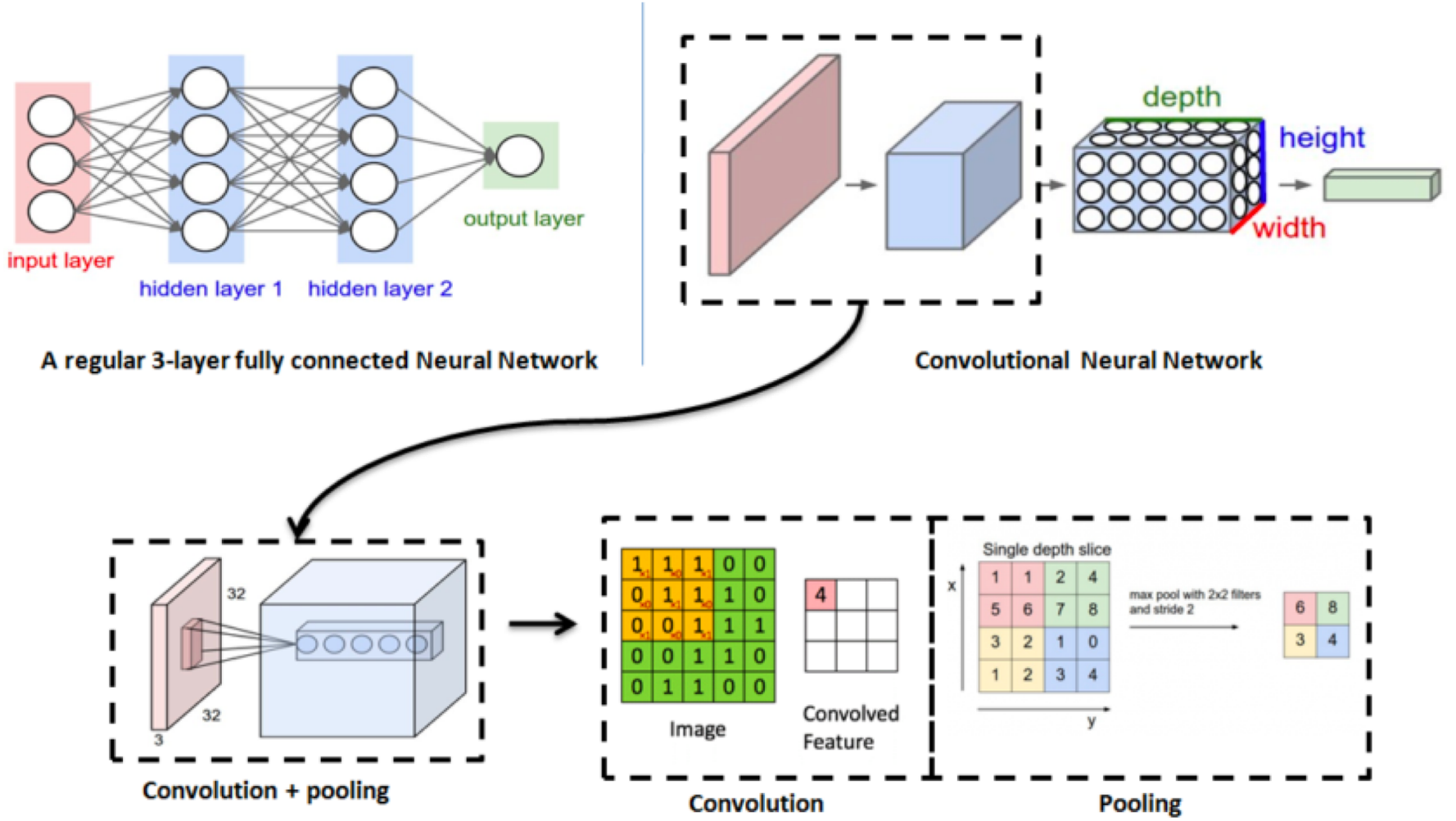
	Activation Shape	Activation Size	# Parameters
Input Layer:	(32, 32, 3)	3072	0
CONV1 (f=5, s=1)	(28, 28, 8)	6272	608
POOL1	(14, 14, 8)	1568	0
CONV2 (f=5, s=1)	(10, 10, 16)	1600	3216
POOL2	(5, 5, 16)	400	0
FC3	(120, 1)	120	48120
FC4	(84, 1)	84	10164
Softmax	(10, 1)	10	850

WRAP-UP

- In CNNs, as we move from layer to layer, we usually see:
  - The height **decrease**.
  - The width **decrease**.
  - The number of channels **increase**.

- The structure of CNNs is usually:
  - CONV – POOL – CONV – POOL – FC – FC – FC – …

CNN VISUAL



Source: http://ufldl.stanford.edu/tutorial/images/Convolution_schematic.gif, http://cs231n.github.io/assets/nn1/neural_net2.jpeg, http://cs231n.github.io/assets/cnn/depthcol.jpeg, http://cs231n.github.io/assets/cnn/maxpool.jpeg