Machine Learning and Transportation - Convolutional Neural Networks

Peter Chen

Shanghai University of Engineering Science

December 7-15, 2019

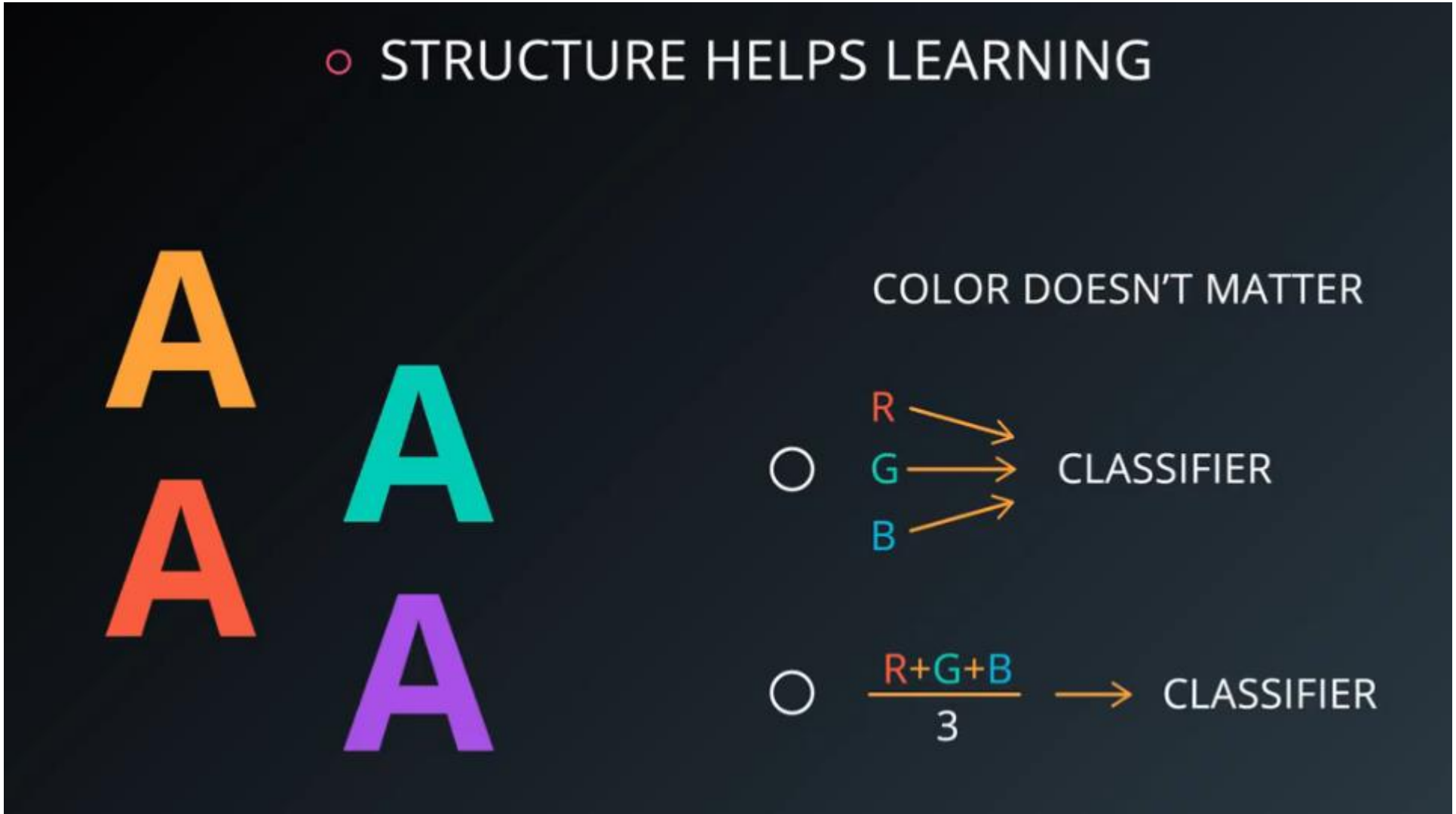
Neural Networks

The following lessons contain introductory and intermediate material on neural networks, building a neural network from scratch, using TensorFlow, and Convolutional Neural Networks:

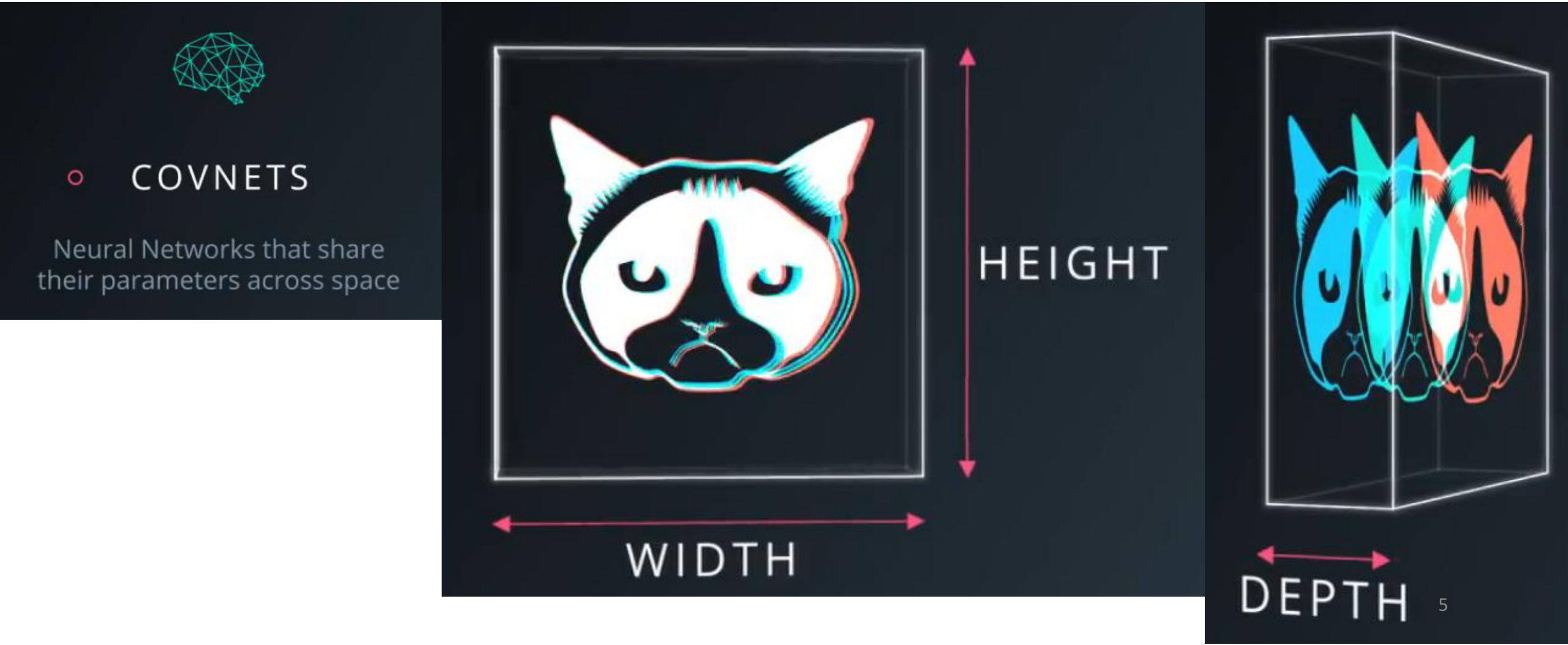
- **Neural Networks**
- **TensorFlow**
- **Deep Neural Networks**
- **Convolutional Neural Networks**

Convolutional Neural Networks

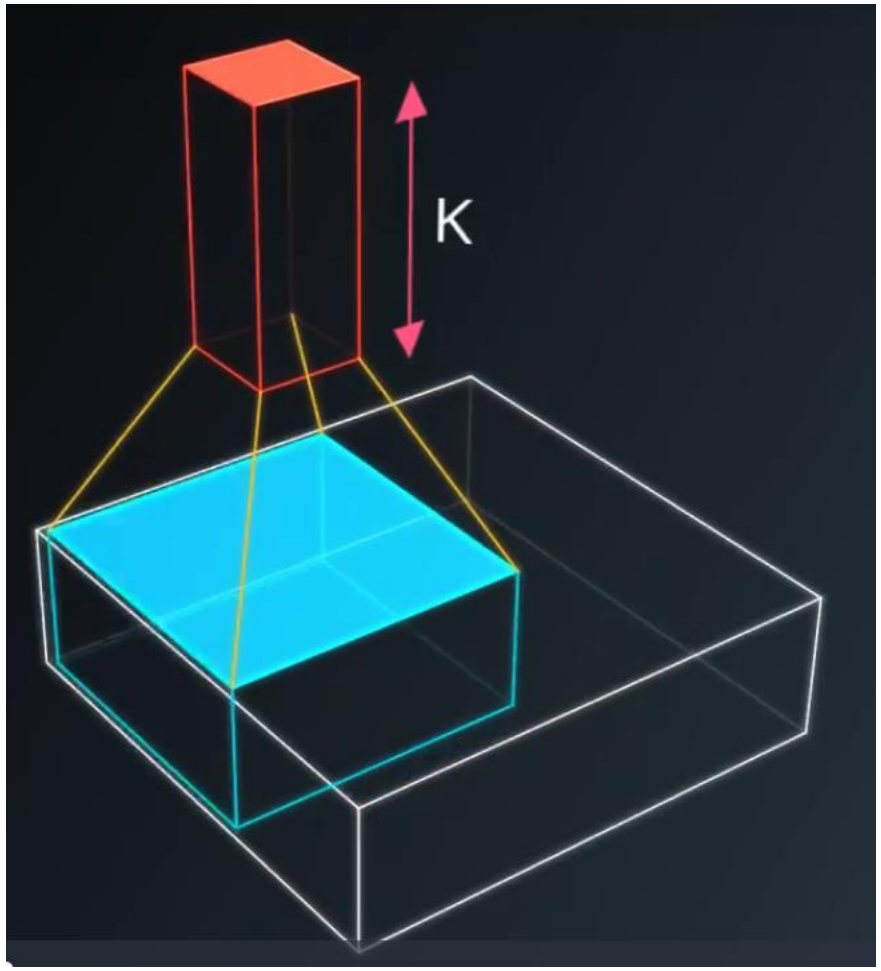
Color

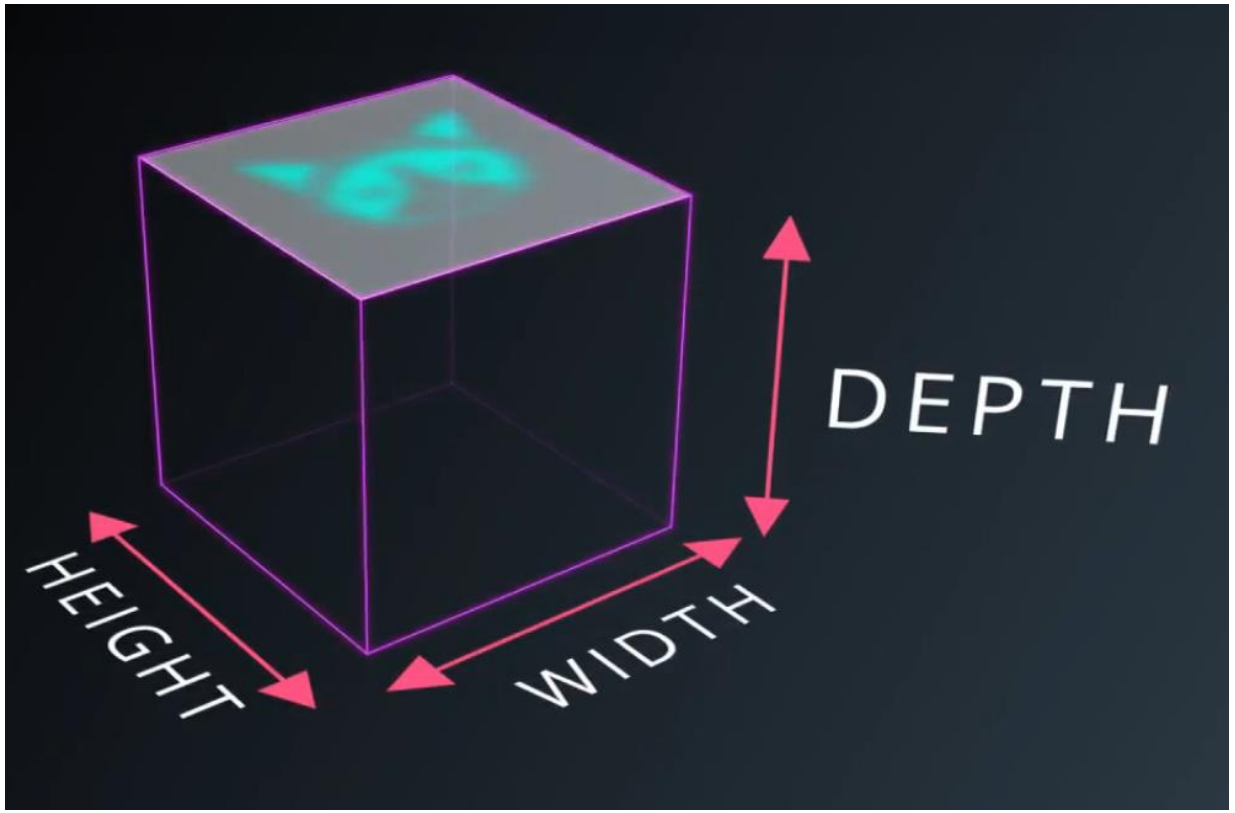

STRUCTURE HELPS LEARNING
A
A
A
A
COLOR DOESN'T MATTER
R
G
B
CLASSIFIER
R+G+B
3
CLASSIFIER

Convolutional Networks

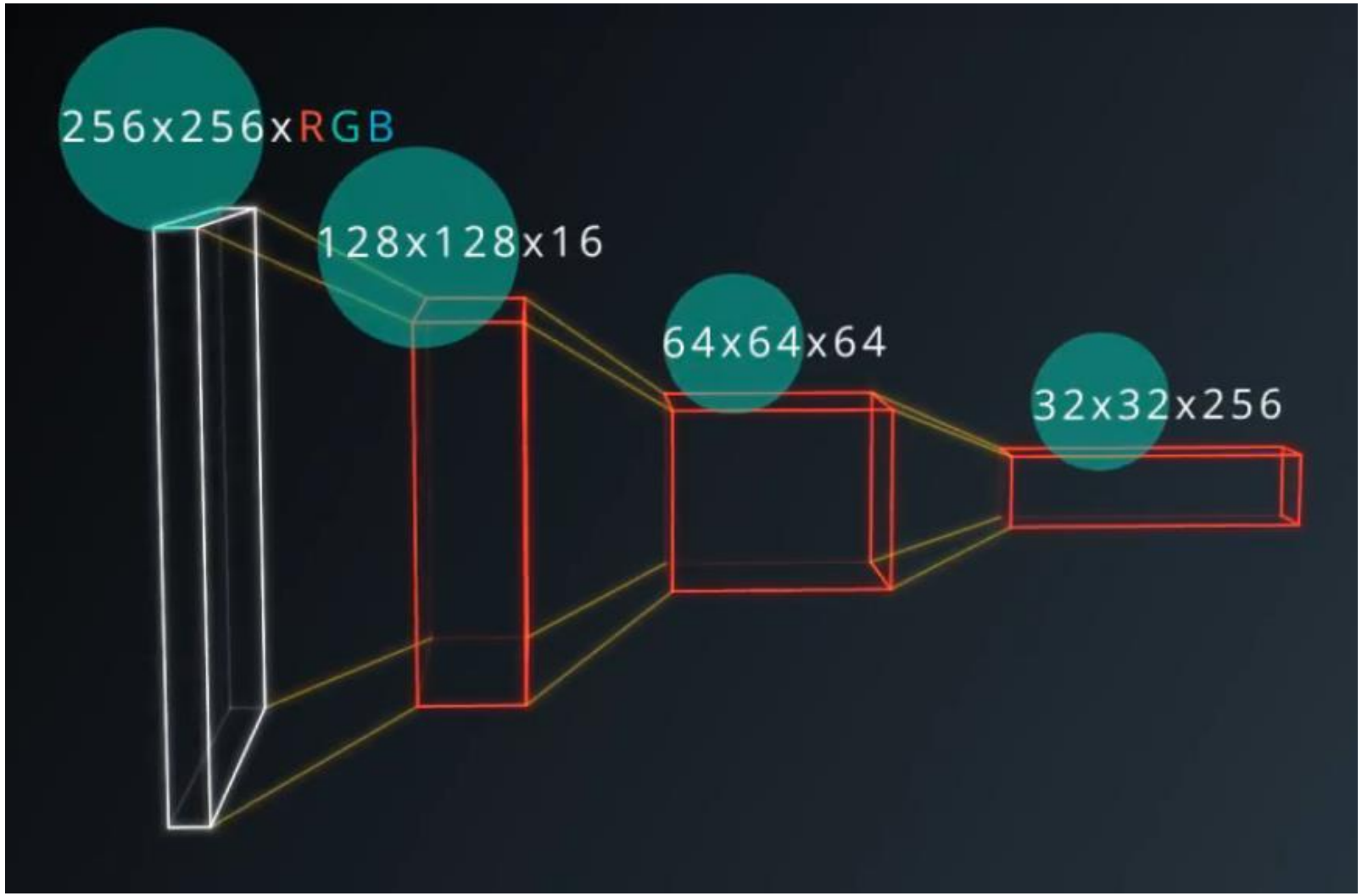


Convolutional Networks

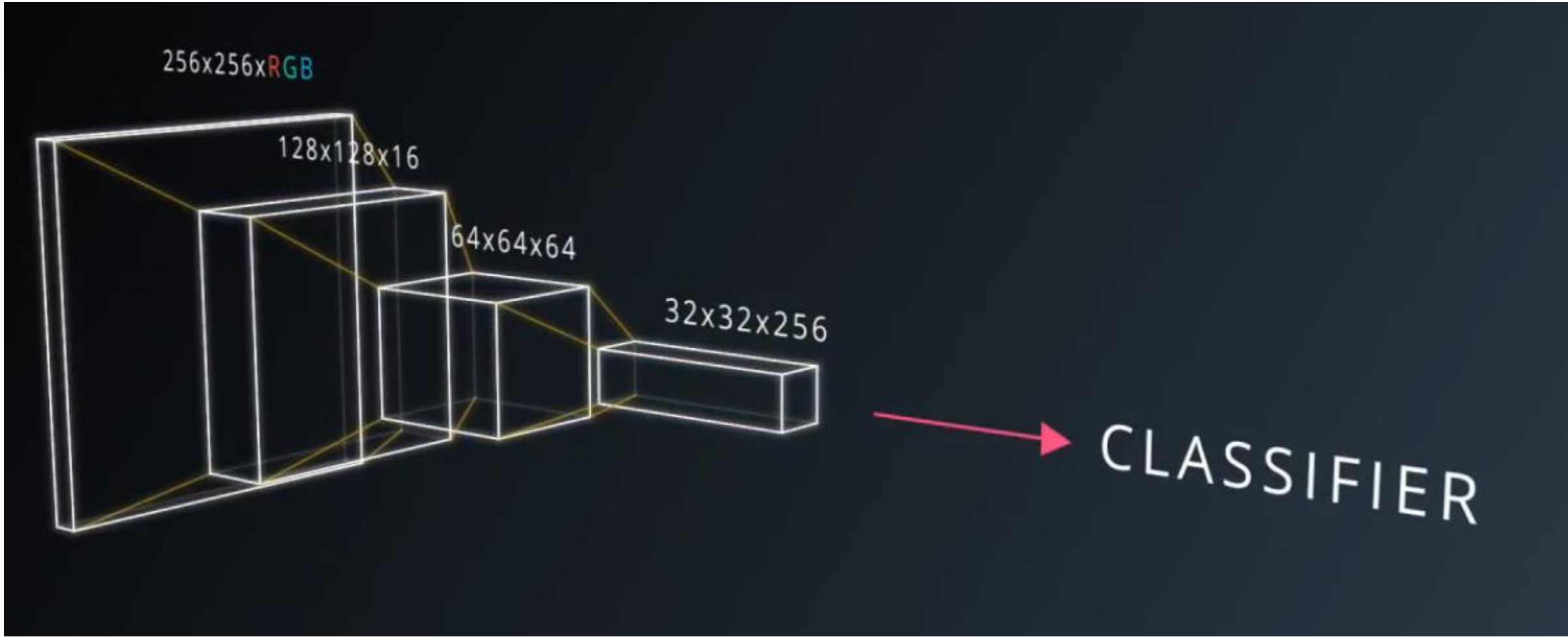




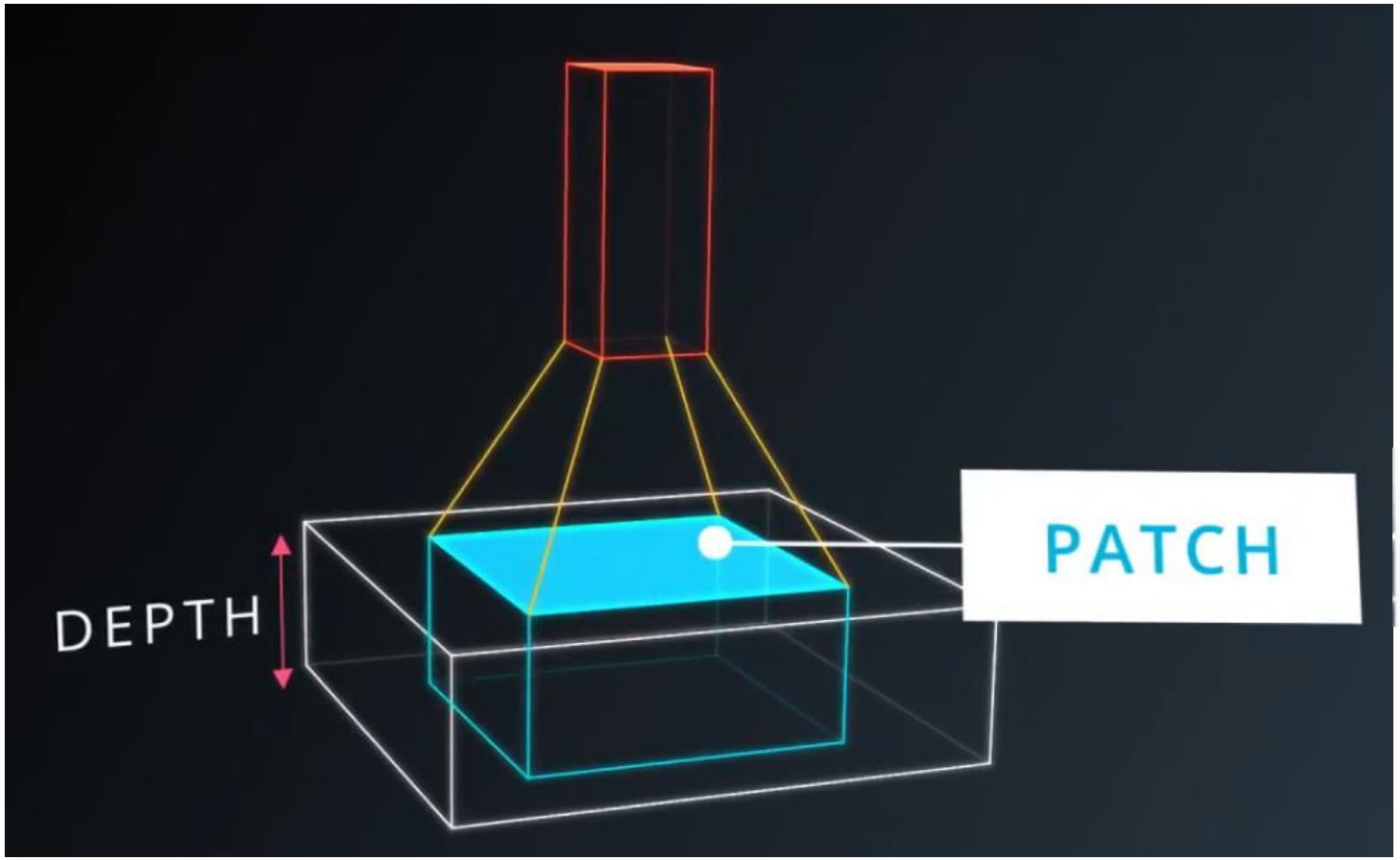
Convolutional Networks



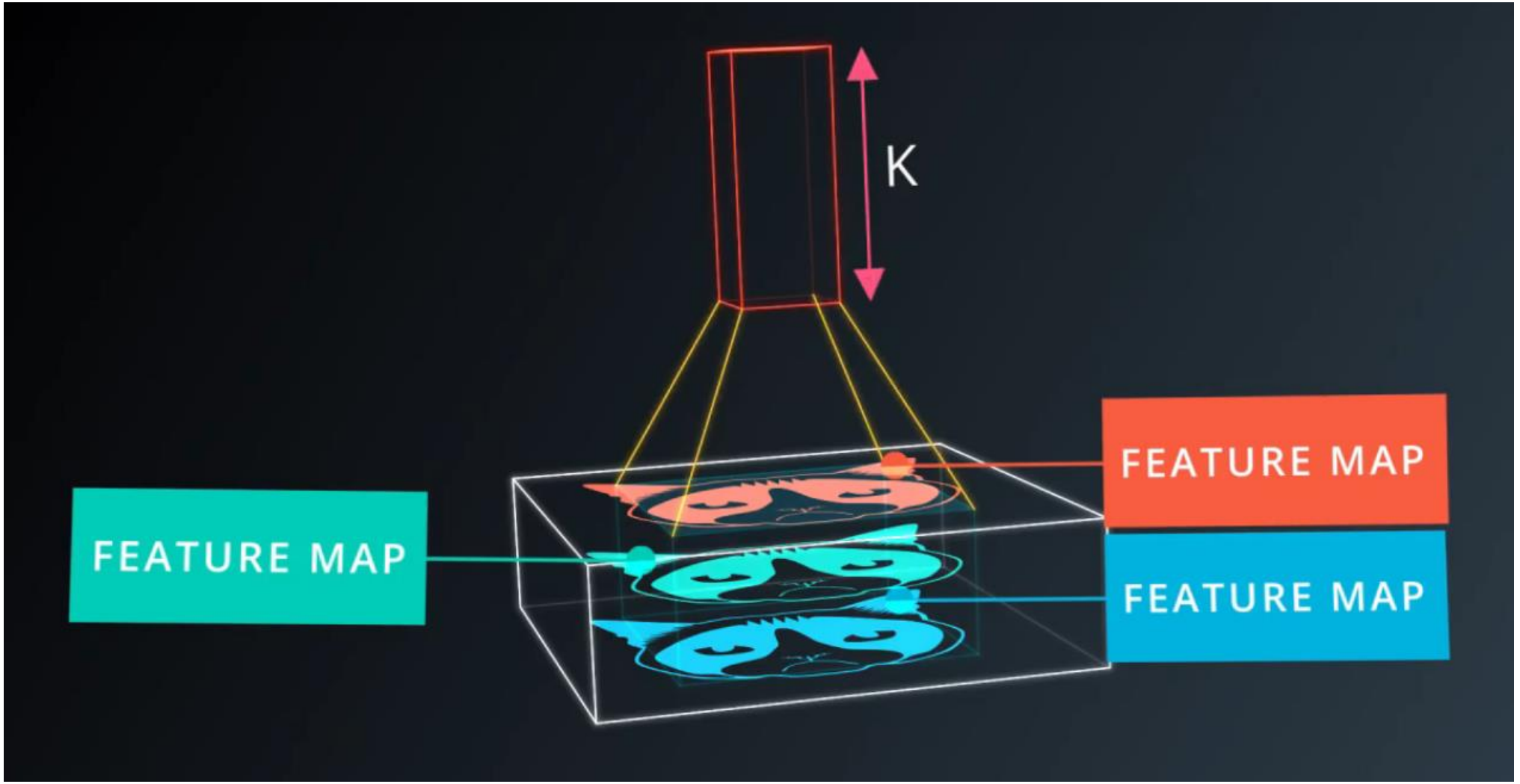
Convolutional Networks



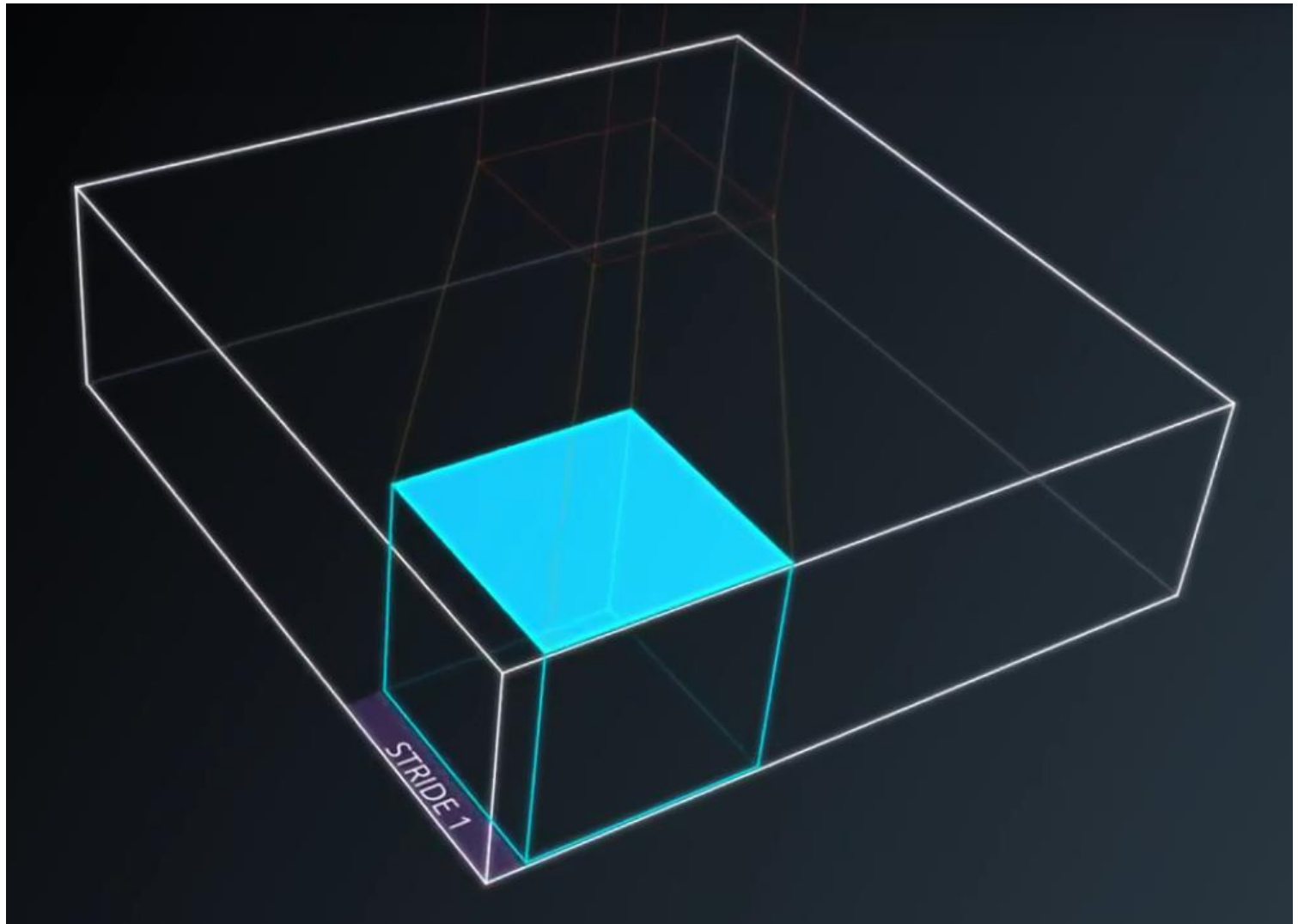
Convolutional Networks



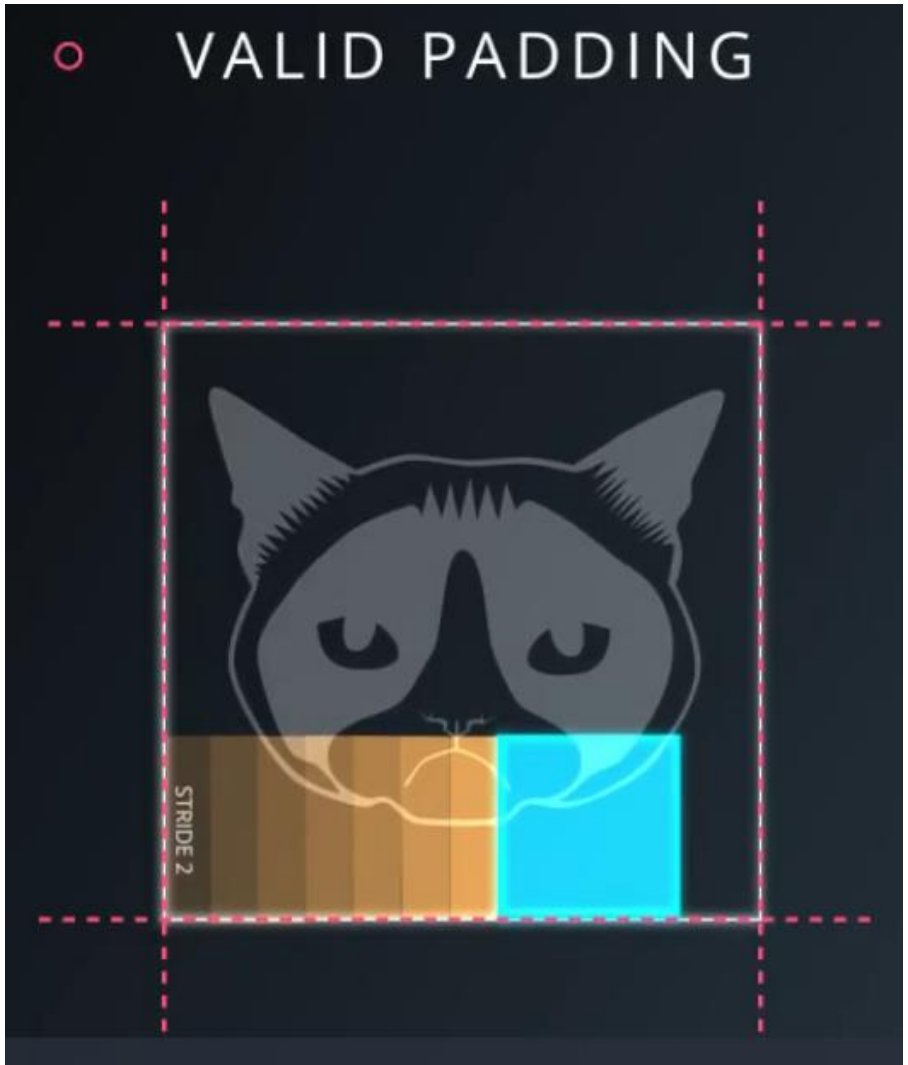
Convolutional Networks



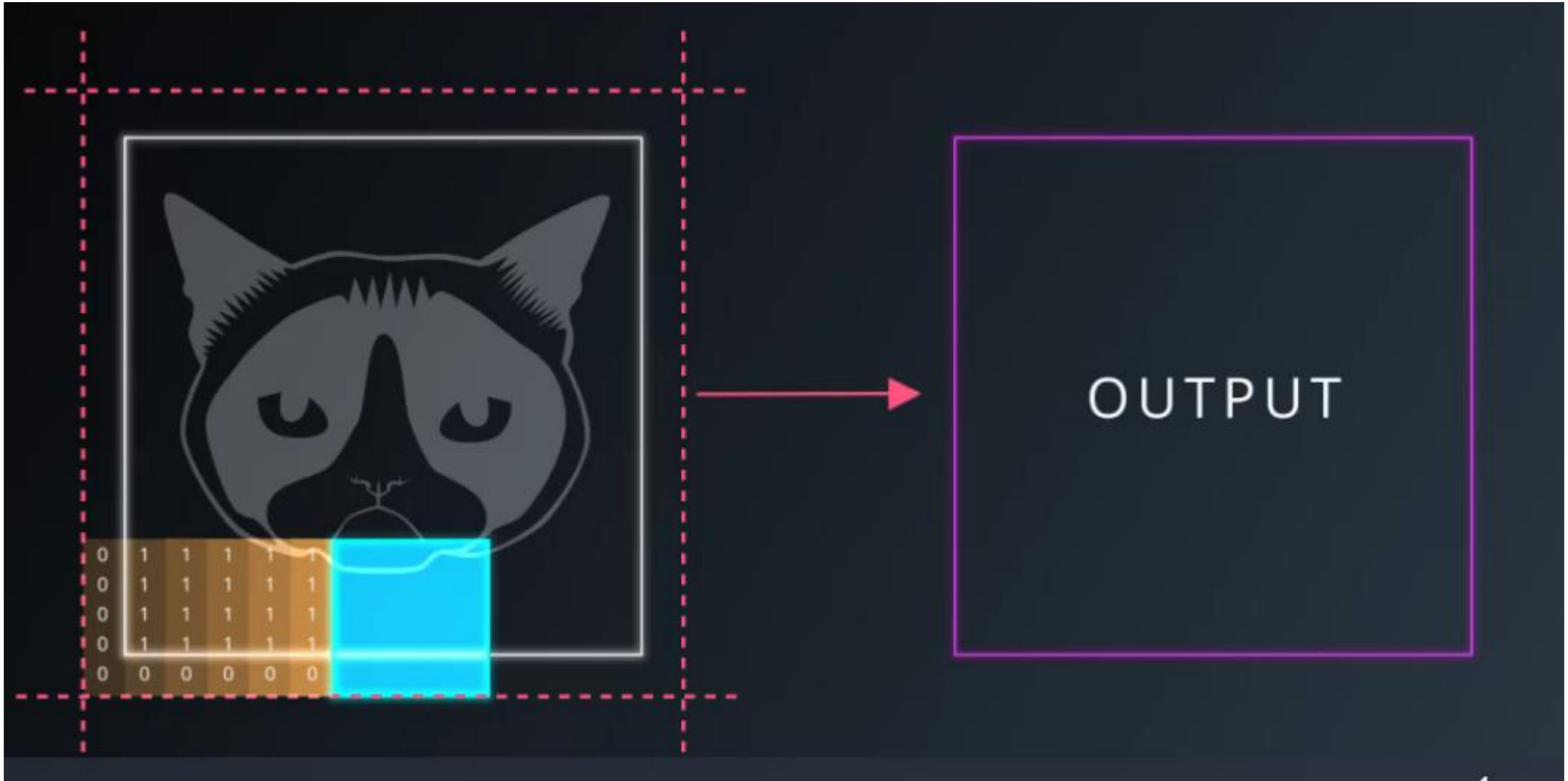
Convolutional Networks



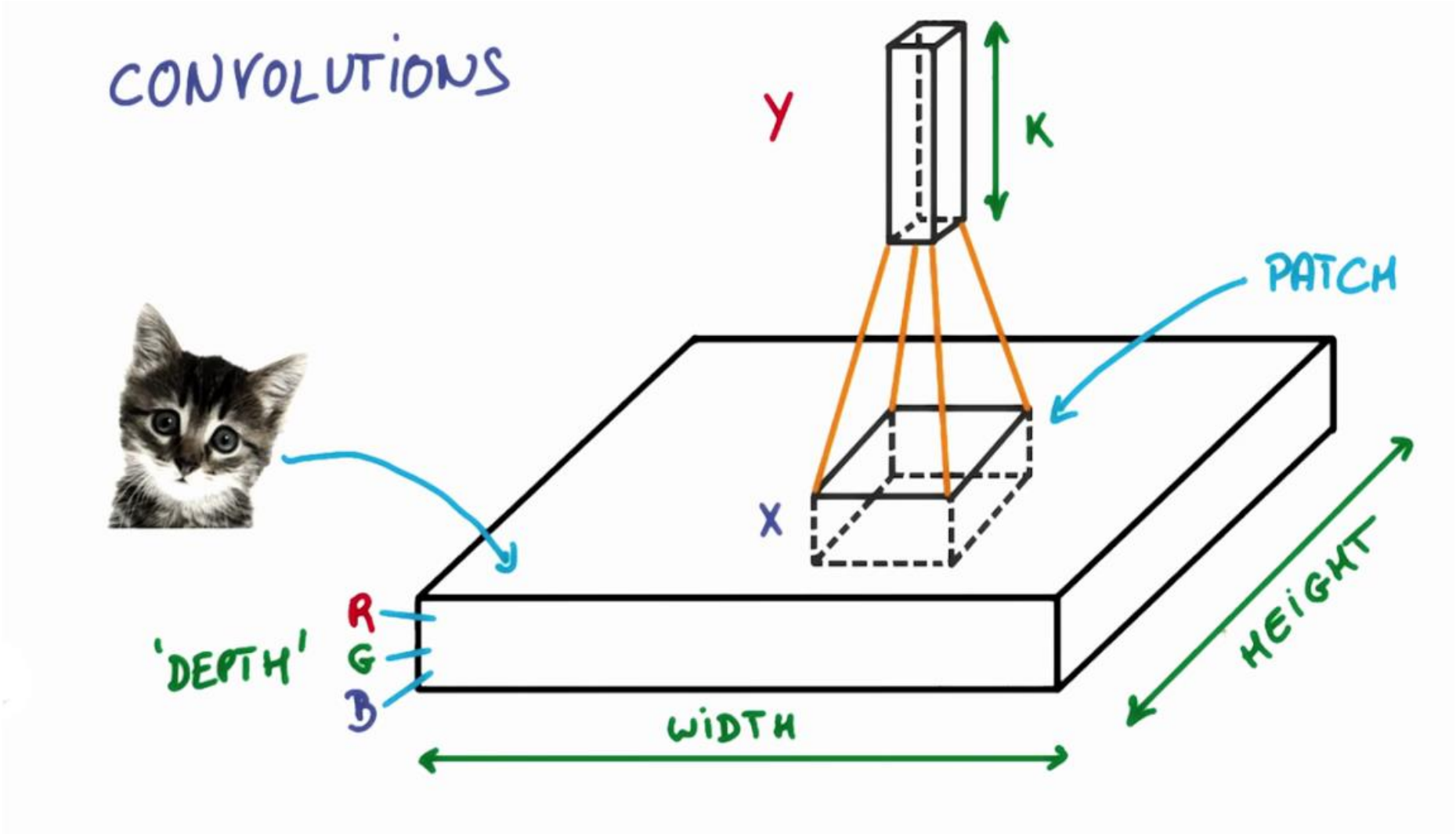
Convolutional Networks



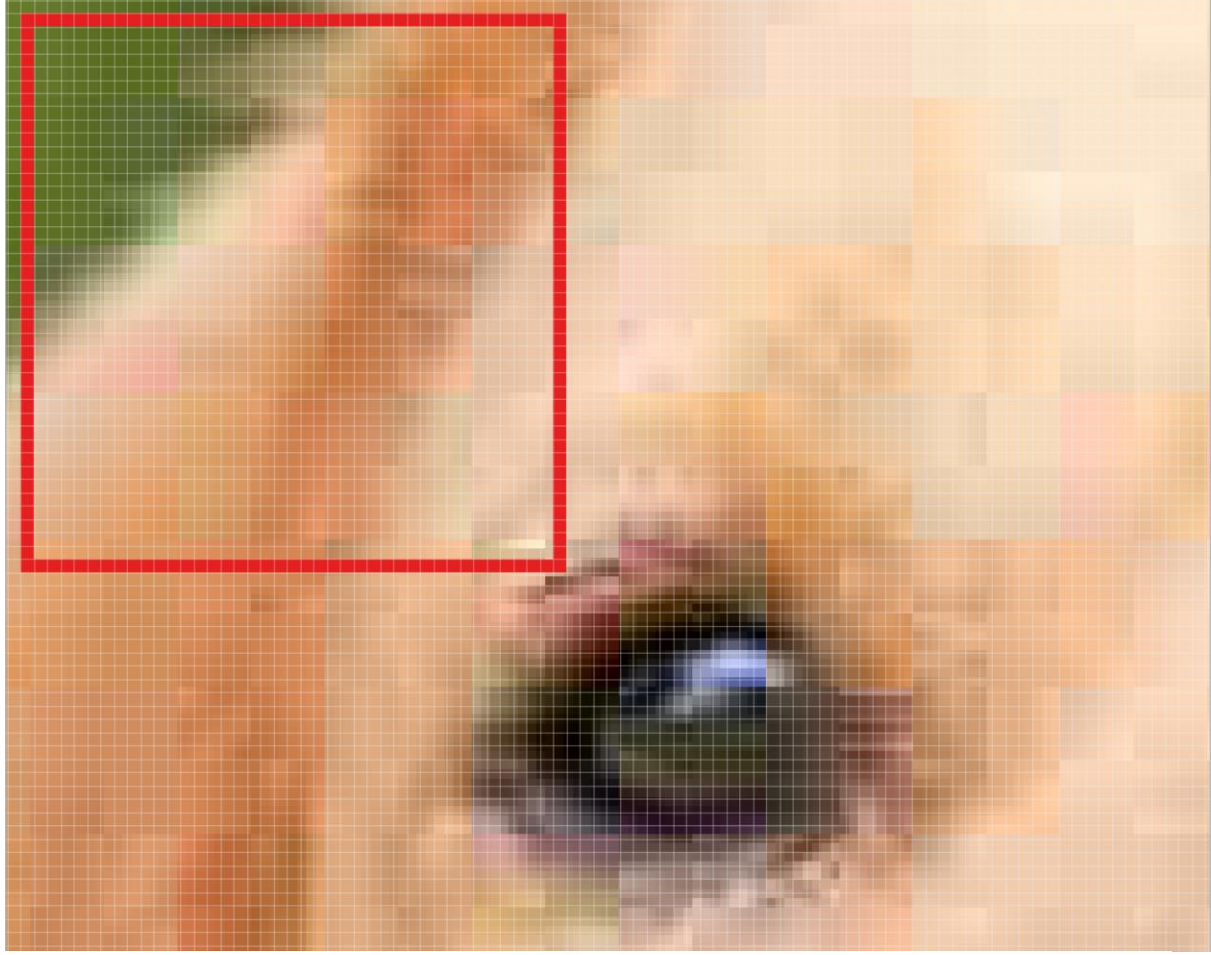
Convolutional Networks



Filters



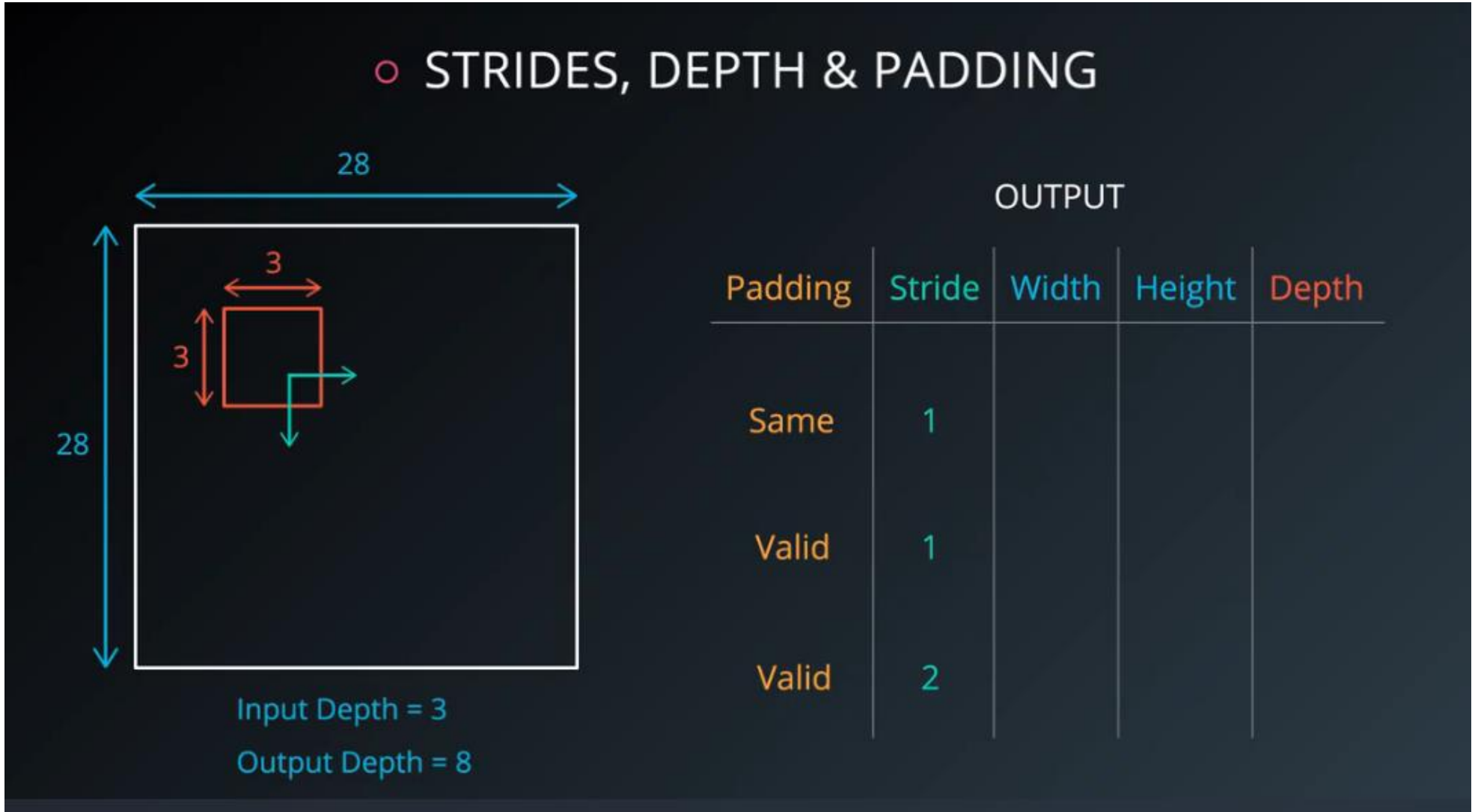
Filters



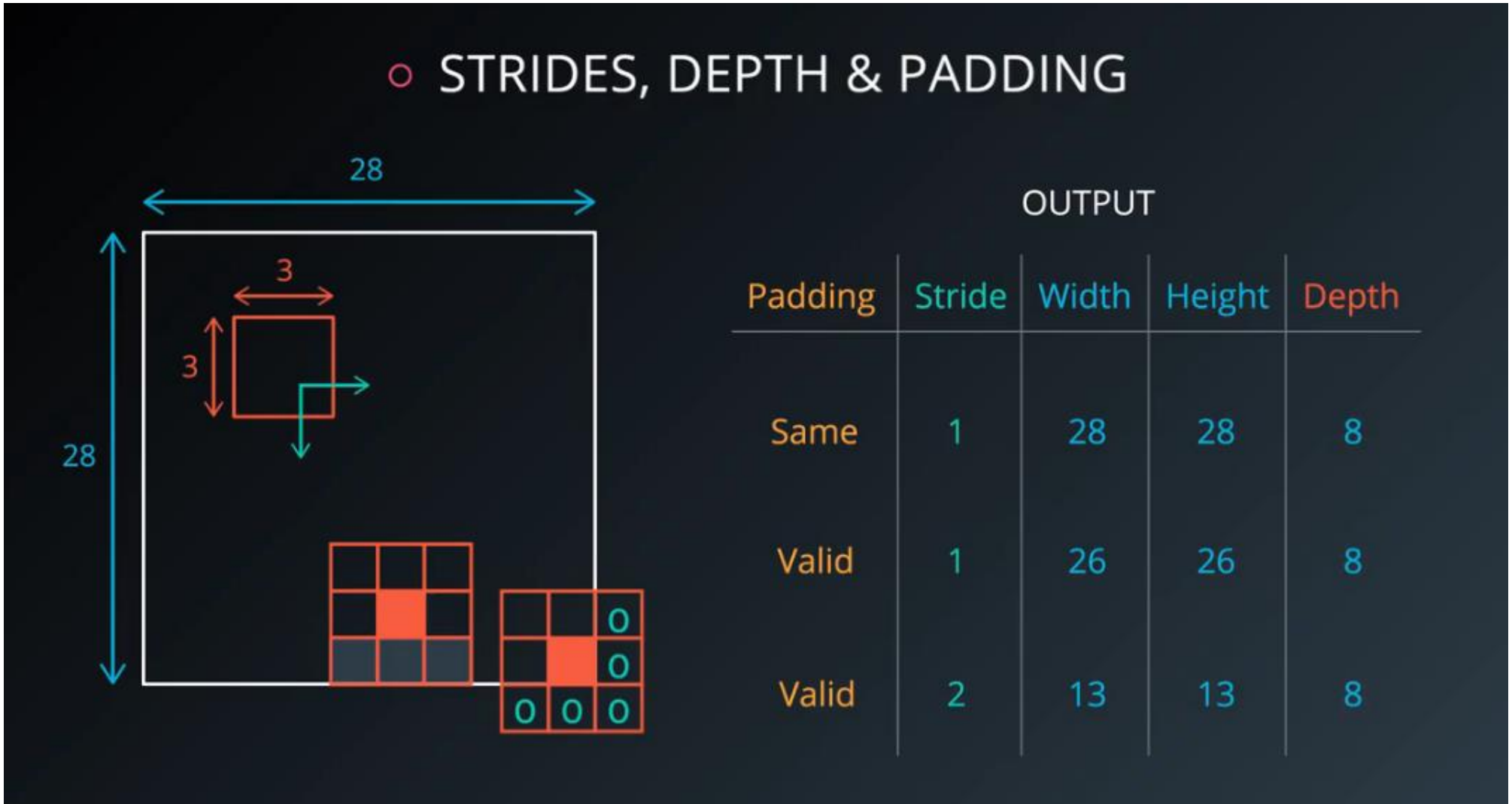
Filter Depth

It's common to have more than one filter. Different filters pick up different qualities of a patch. For example, one filter might look for a particular color, while another might look for a kind of object of a specific shape. The amount of filters in a convolutional layer is called the *filter depth*.

Feature Map Sizes



Feature Map Sizes



OUTPUT

Padding	Stride	Width	Height	Depth
Same	1	28	28	8
Valid	1	26	26	8
Valid	2	13	13	8

Dimensionality

From what we've learned so far, how can we calculate the number of neurons of each layer in our CNN?

Given:

- our input layer has a width of `W` and a height of `H`
- our convolutional layer has a filter size `F`
- we have a stride of `S`
- a padding of `P`
- and the number of filters `K`,

the following formula gives us the width of the next layer: `W_out =[ (W-F+2P)/S] + 1`.

The output height would be `H_out = [(H-F+2P)/S] + 1`.

And the output depth would be equal to the number of filters `D_out = K`.

The output volume would be `W_out * H_out * D_out`.

Knowing the dimensionality of each additional layer helps us understand how large our model is and how our decisions around filter size and stride affect the size of our network.

Convolution Output Shape

Introduction

For the next few quizzes we'll test your understanding of the dimensions in CNNs. Understanding dimensions will help you make accurate tradeoffs between model size and performance. As you'll see, some parameters have a much bigger impact on model size than others.

Setup

H = height, W = width, D = depth

- We have an input of shape 32x32x3 (HxWxD)
- 20 filters of shape 8x8x3 (HxWxD)
- A stride of 2 for both the height and width (S)
- With padding of size 1 (P)

Recall the formula for calculating the new height or width:

```
new_height = (input_height - filter_height + 2 * P)/S + 1
new_width = (input_width - filter_width + 2 * P)/S + 1
```

Convolutional Layer Output Shape

What's the shape of the output?

The answer format is **HxWxD**, so if you think the new height is 9, new width is 9, and new depth is 5, then type 9x9x5.

Solution

The answer is **14x14x20**.

We can get the new height and width with the formula resulting in:

```
(32 - 8 + 2 * 1)/2 + 1 = 14
(32 - 8 + 2 * 1)/2 + 1 = 14
```

The new depth is equal to the number of filters, which is 20.

This would correspond to the following code:

```
input = tf.placeholder(tf.float32, (None, 32, 32, 3))
filter_weights = tf.Variable(tf.truncated_normal((8, 8, 3, 20))) # (height, width, input_depth, output_dept
filter_bias = tf.Variable(tf.zeros(20))
strides = [1, 2, 2, 1] # (batch, height, width, depth)
padding = 'SAME'
conv = tf.nn.conv2d(input, filter_weights, strides, padding) + filter_bias
```

Note the output shape of `conv` will be [1, 16, 16, 20]. It's 4D to account for batch size, but more importantly, it's not [1, 14, 14, 20]. This is because the padding algorithm TensorFlow uses is not exactly the same as the one above. An alternative algorithm is to switch `padding` from `'SAME'` to `'VALID'` which would result in an output shape of [1, 13, 13, 20]. If you're curious how padding works in TensorFlow, read **this document**.

In summary TensorFlow uses the following equation for 'SAME' vs 'VALID'

SAME Padding, the output height and width are computed as:

`out_height` = ceil(float(in_height) / float(strides[1]))

`out_width` = ceil(float(in_width) / float(strides[2]))

VALID Padding, the output height and width are computed as:

`out_height` = ceil(float(in_height - filter_height + 1) / float(strides[1]))

`out_width` = ceil(float(in_width - filter_width + 1) / float(strides[2]))

Number of Parameters

We're now going to calculate the number of parameters of the convolutional layer. The answer from the last quiz will come into play here!

Being able to calculate the number of parameters in a neural network is useful since we want to have control over how much memory a neural network uses.

Setup

H = height, W = width, D = depth

- We have an input of shape 32x32x3 (HxWxD)
- 20 filters of shape 8x8x3 (HxWxD)
- A stride of 2 for both the height and width (S)
- Zero padding of size 1 (P)

Output Layer

- 14x14x20 (HxWxD)

Solution

There are `756560` total parameters. That's a HUGE amount! Here's how we calculate it:

```
(8 * 8 * 3 + 1) * (14 * 14 * 20) = 756560
```

`8 * 8 * 3` is the number of weights, we add `1` for the bias. Remember, each weight is assigned to every single part of the output (`14 * 14 * 20`). So we multiply these two numbers together and we get the final answer.

Parameter Sharing

Now we'd like you to calculate the number of parameters in the convolutional layer, if every neuron in the output layer shares its parameters with every other neuron in its same channel.

This is the number of parameters actually used in a convolution layer (`tf.nn.conv2d()`).

Setup

H = height, W = width, D = depth

- We have an input of shape 32x32x3 (HxWxD)
- 20 filters of shape 8x8x3 (HxWxD)
- A stride of 2 for both the height and width (S)
- Zero padding of size 1 (P)

Output Layer

- 14x14x20 (HxWxD)

Hint

With parameter sharing, each neuron in an output channel shares its weights with every other neuron in that channel. So the number of parameters is equal to the number of neurons in the filter, plus a bias neuron, all multiplied by the number of channels in the output layer.

Convolution Layer Parameters 2

How many parameters does the convolution layer have (with parameter sharing)?

Solution

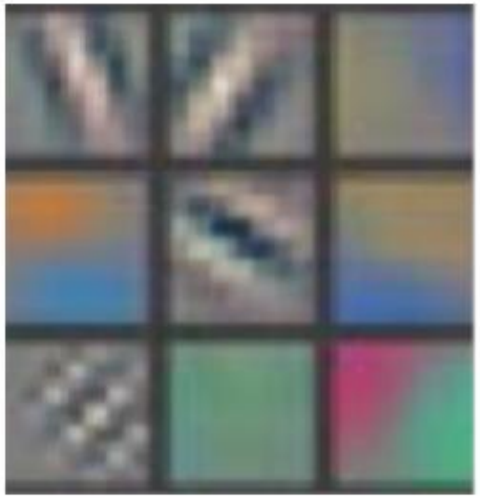
There are `3860` total parameters. That's 196 times fewer parameters! Here's how the answer is calculated:

```
(8 * 8 * 3 + 1) * 20 = 3840 + 20 = 3860
```

That's `3840` weights and `20` biases. This should look similar to the answer from the previous quiz. The difference being it's just `20` instead of (`14 * 14 * 20`). Remember, with weight sharing we use the same filter for an entire depth slice. Because of this we can get rid of `14 * 14` and be left with only `20`.

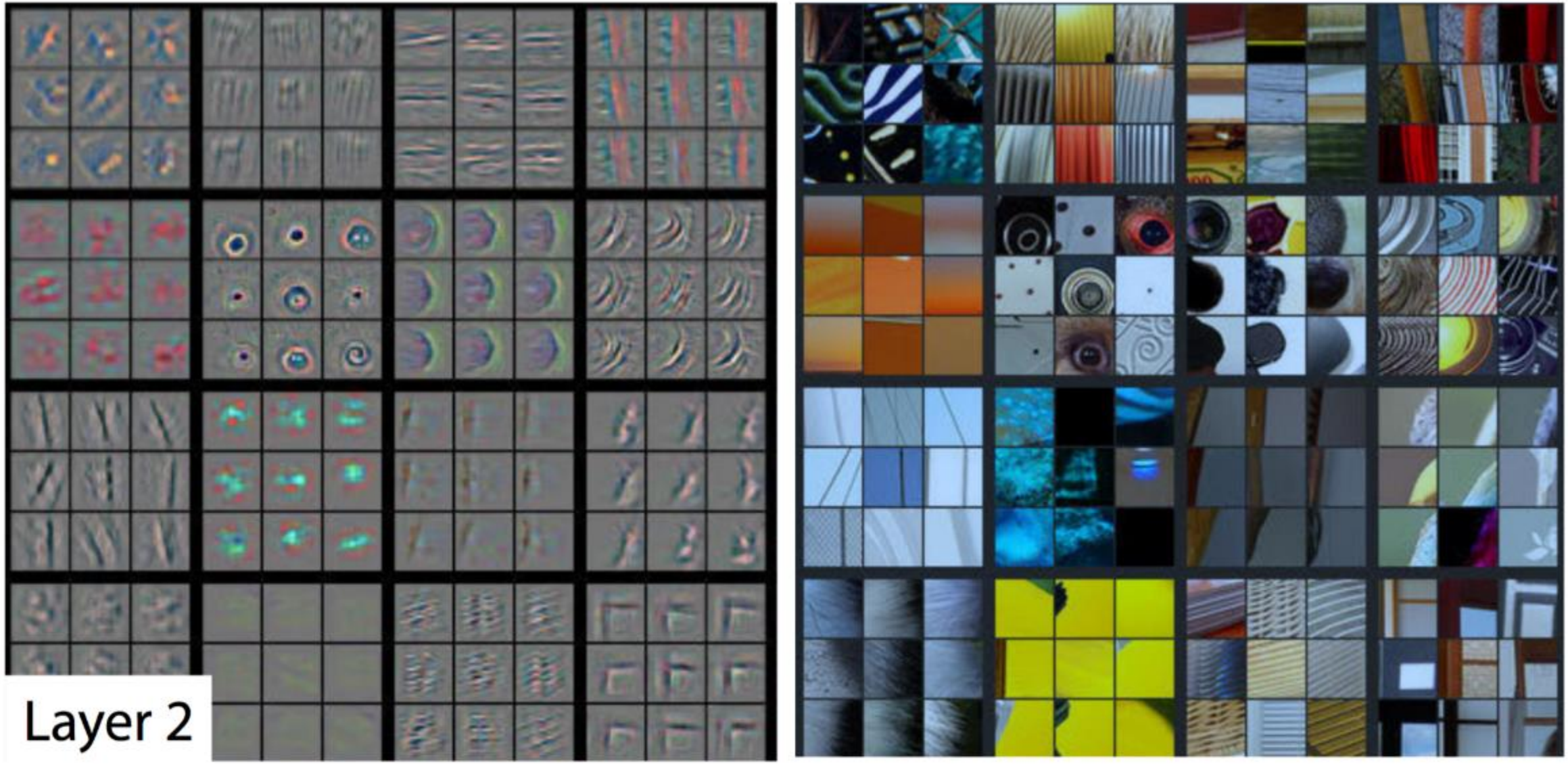
Visualizing CNNs

Layer 1

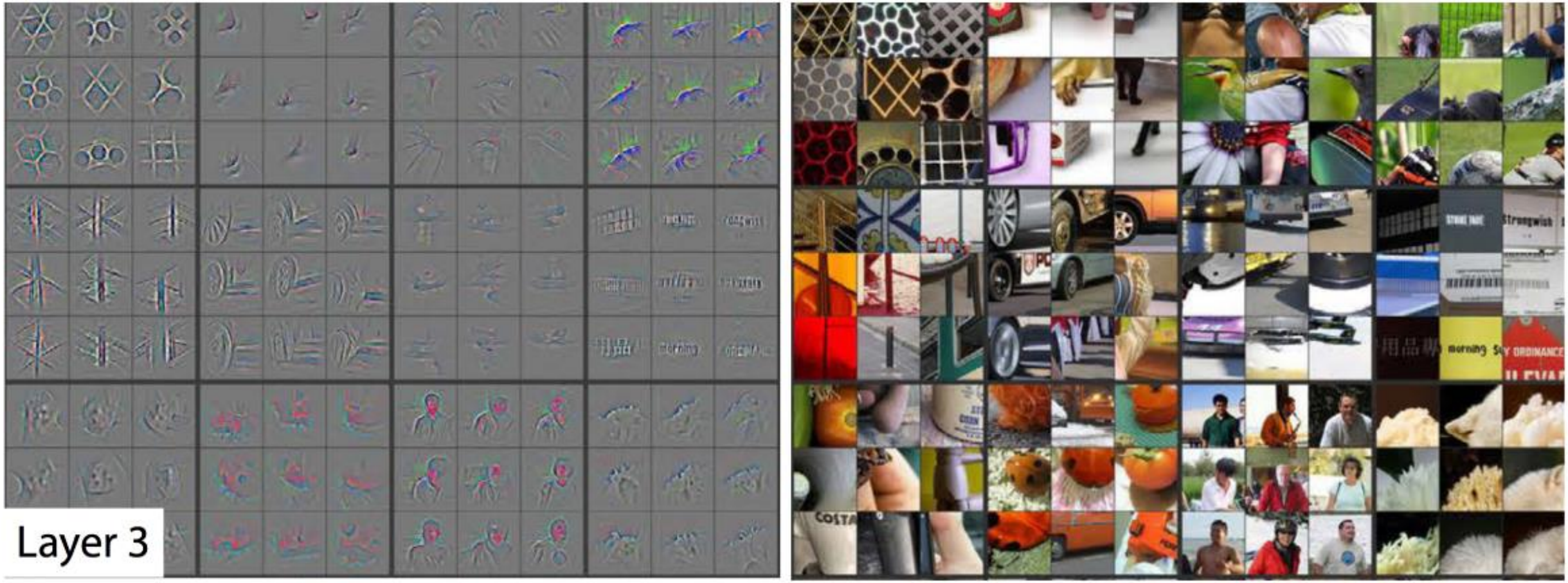


Example patterns that cause activations in the first layer of the network. These range from simple diagonal lines (top left) to green blobs (bottom middle).

The images above are from Matthew Zeiler and Rob Fergus' **deep visualization toolbox**, which lets us visualize what each layer in a CNN focuses on.

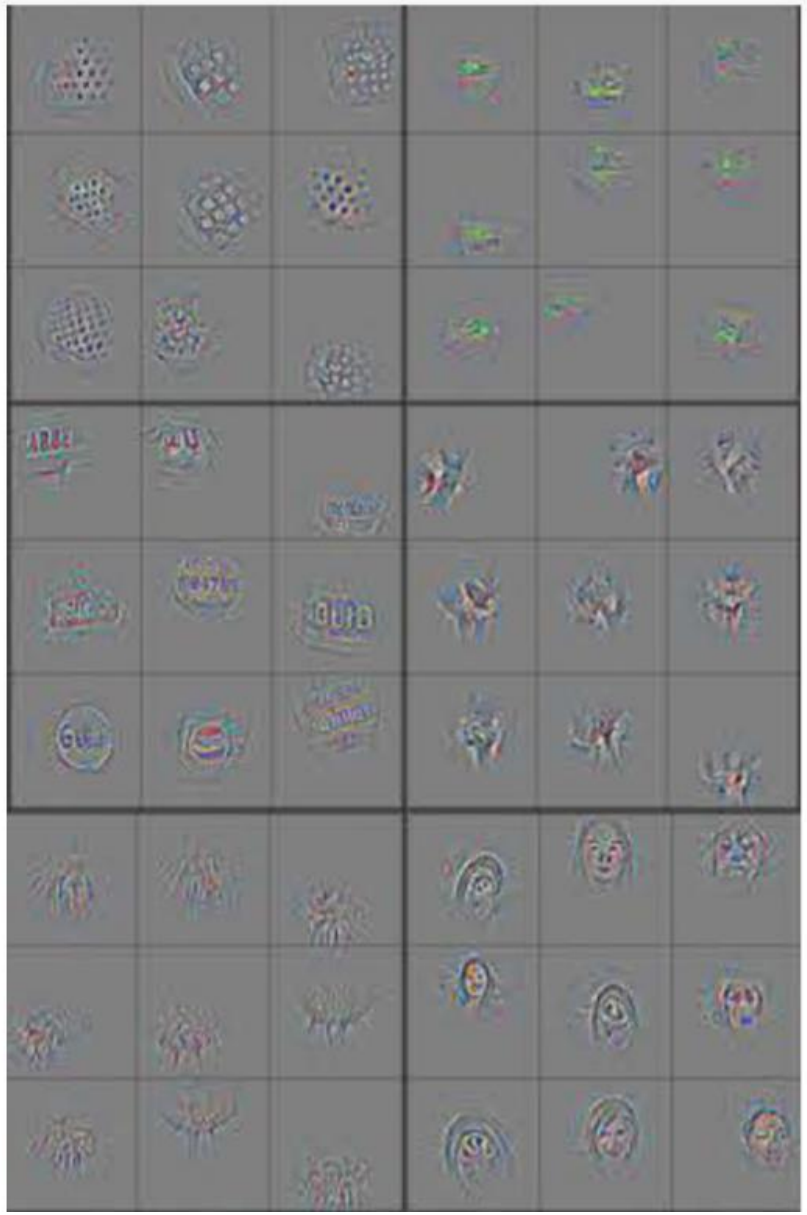


A visualization of the second layer in the CNN. Notice how we are picking up more complex ideas like circles and stripes. The gray grid on the left represents how this layer of the CNN activates (or "what it sees") based on the corresponding images from the grid on the right.



A visualization of the third layer in the CNN. The gray grid on the left represents how this layer of the CNN activates (or "what it sees") based on the corresponding images from the grid on the right.

We'll skip layer 4, which continues this progression, and jump right to the fifth and final layer of this CNN.




BARBER SHOP
GULF

A visualization of the fifth and final layer of the CNN. The gray grid on the left represents how this layer of the CNN activates (or "what it sees") based on the corresponding images from the grid on the right.

TensorFlow Convolution Layer

TensorFlow Convolution Layer

Let's examine how to implement a CNN in TensorFlow.

TensorFlow provides the `tf.nn.conv2d()` and `tf.nn.bias_add()` functions to create your own convolutional layers.

```
# Output depth
k_output = 64

# Image Properties
image_width = 10
image_height = 10
color_channels = 3

# Convolution filter
filter_size_width = 5
filter_size_height = 5

# Input/Image
input = tf.placeholder(
    tf.float32,
    shape=[None, image_height, image_width, color_channels])

# Weight and bias
weight = tf.Variable(tf.truncated_normal(
    [filter_size_height, filter_size_width, color_channels, k_output]))
bias = tf.Variable(tf.zeros(k_output))

# Apply Convolution
conv_layer = tf.nn.conv2d(input, weight, strides=[1, 2, 2, 1], padding='SAME')
# Add bias
conv_layer = tf.nn.bias_add(conv_layer, bias)
# Apply activation function
conv_layer = tf.nn.relu(conv_layer)
```

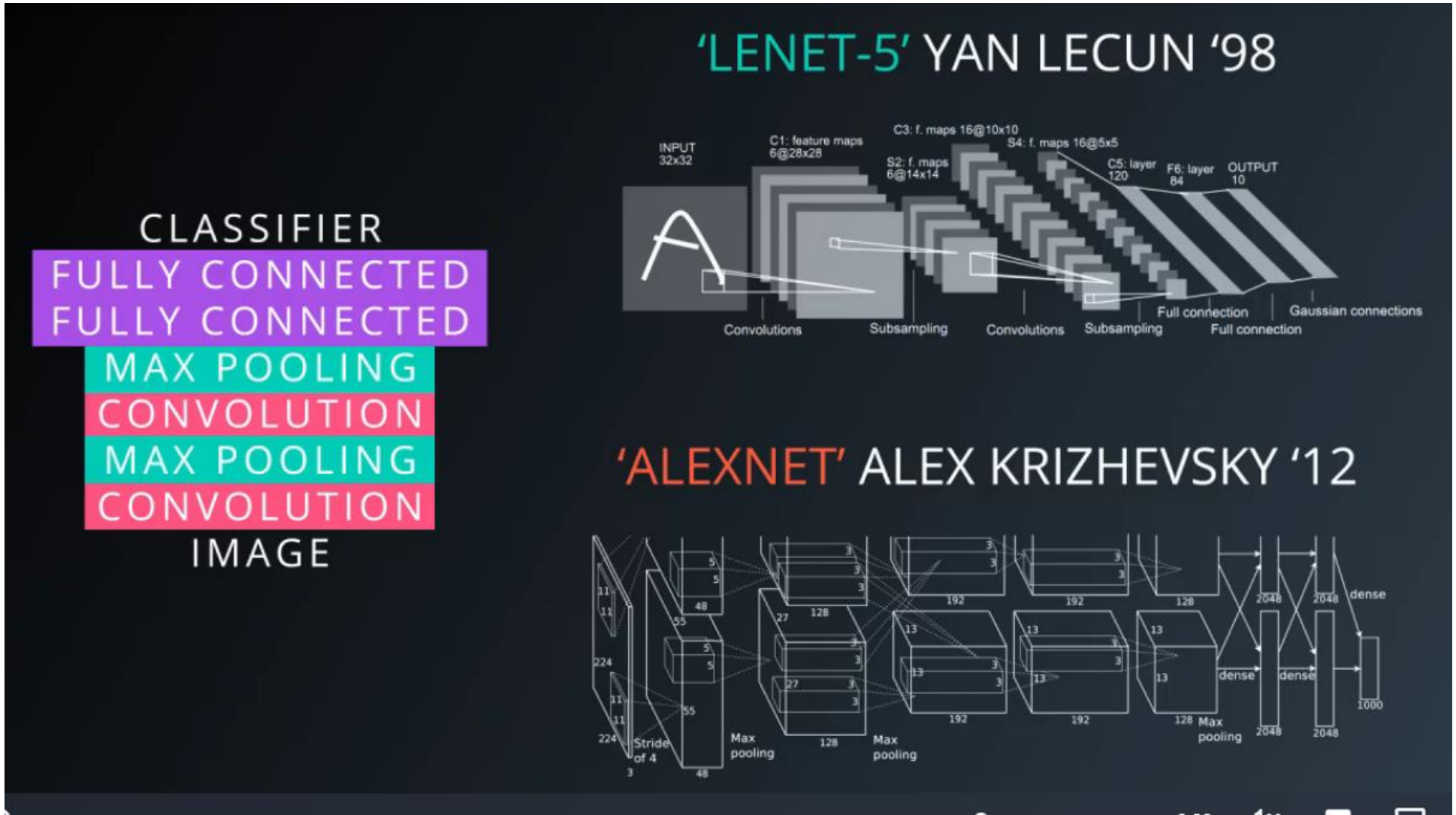
The code above uses the `tf.nn.conv2d()` function to compute the convolution with `weight` as the filter and `[1, 2, 2, 1]` for the strides. TensorFlow uses a stride for each `input` dimension, `[batch, input_height, input_width, input_channels]`. We are generally always going to set the stride for `batch` and `input_channels` (i.e. the first and fourth element in the `strides` array) to be `1`.

You'll focus on changing `input_height` and `input_width` while setting `batch` and `input_channels` to 1. The `input_height` and `input_width` strides are for striding the filter over `input`. This example code uses a stride of 2 with 5x5 filter over `input`.

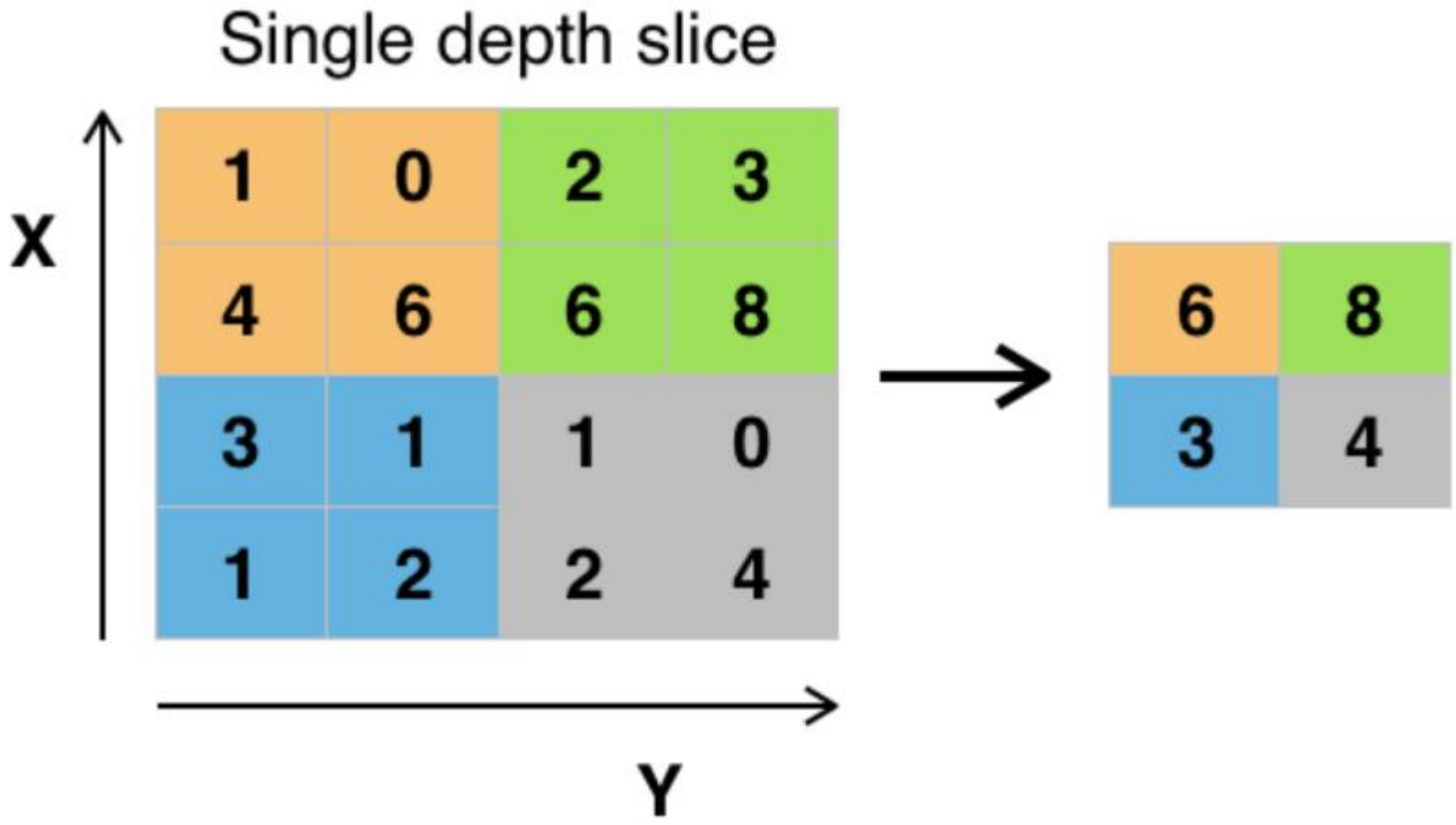
The `tf.nn.bias_add()` function adds a 1-d bias to the last dimension in a matrix.

Explore The Design Space

Explore The Design Space



TensorFlow Max Pooling



TensorFlow provides the `tf.nn.max_pool()` function to apply **max pooling** to your convolutional layers.

```
...
conv_layer = tf.nn.conv2d(input, weight, strides=[1, 2, 2, 1], padding='SAME')
conv_layer = tf.nn.bias_add(conv_layer, bias)
conv_layer = tf.nn.relu(conv_layer)
# Apply Max Pooling
conv_layer = tf.nn.max_pool(
    conv_layer,
    ksize=[1, 2, 2, 1],
    strides=[1, 2, 2, 1],
    padding='SAME')
```

The `tf.nn.max_pool()` function performs max pooling with the `ksize` parameter as the size of the filter and the `strides` parameter as the length of the stride. 2x2 filters with a stride of 2x2 are common in practice.

The `ksize` and `strides` parameters are structured as 4-element lists, with each element corresponding to a dimension of the input tensor (`[batch, height, width, channels]`). For both `ksize` and `strides`, the batch and channel dimensions are typically set to `1`.

Pooling

QUIZ QUESTION

A pooling layer is generally used to ...

- [ ] Increase the size of the output
- [x] Decrease the size of the output
- [x] Prevent overfitting
- [ ] Gain information

Pooling

The correct answer is **decrease the size of the output** and **prevent overfitting**. Preventing overfitting is a consequence of reducing the output size, which in turn, reduces the number of parameters in future layers.

Recently, pooling layers have fallen out of favor. Some reasons are:

- Recent datasets are so big and complex we're more concerned about underfitting.
- Dropout is a much better regularizer.
- Pooling results in a loss of information. Think about the max pooling operation as an example. We only keep the largest of *n* numbers, thereby disregarding *n-1* numbers completely.

Pooling Mechanics

Setup

H = height, W = width, D = depth

- We have an input of shape 4x4x5 (HxWxD)
- Filter of shape 2x2 (HxW)
- A stride of 2 for both the height and width (S)

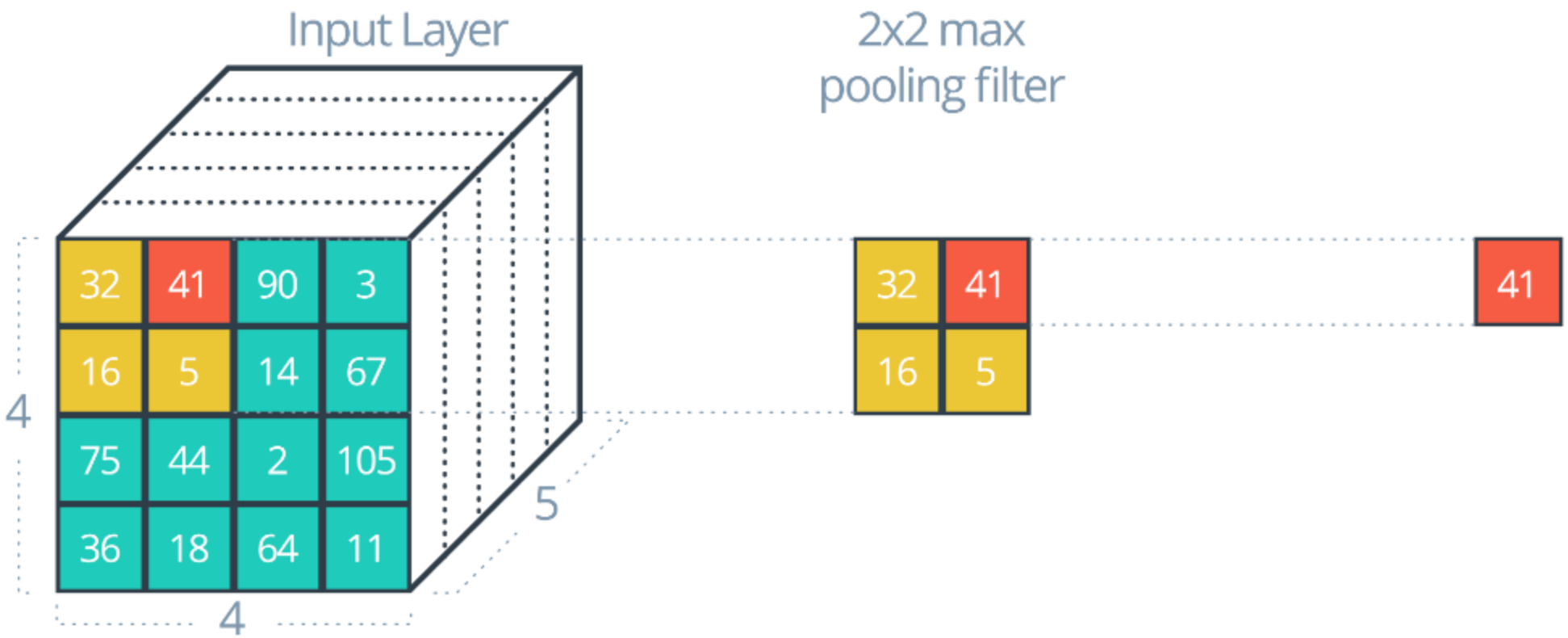
Recall the formula for calculating the new height or width:

```
new_height = (input_height - filter_height)/S + 1
new_width = (input_width - filter_width)/S + 1
```

NOTE: For a pooling layer the output depth is the same as the input depth. Additionally, the pooling operation is applied individually for each depth slice.

The image below gives an example of how a max pooling layer works. In this case, the max pooling filter has a shape of 2x2. As the max pooling filter slides across the input layer, the filter will output the maximum value of the 2x2 square.

Pooling Mechanics Quiz



Solution

The answer is **2x2x5**. Here's how it's calculated using the formula:

```
(4 - 2)/2 + 1 = 2
(4 - 2)/2 + 1 = 2
```

The depth stays the same.

Here's the corresponding code:

```
input = tf.placeholder(tf.float32, (None, 4, 4, 5))
filter_shape = [1, 2, 2, 1]
strides = [1, 2, 2, 1]
padding = 'VALID'
pool = tf.nn.max_pool(input, filter_shape, strides, padding)
```

The output shape of `pool` will be [1, 2, 2, 5], even if `padding` is changed to `'SAME'`.

Convolutional Network in TensorFlow

- See pdf and code

LeNet in TensorFlow

- See pdf and code

Source: 1. https://github.com/udacity 2. udacity.com

Enjoy Python and Machine Learning!

Peter Chen

chpchen@gmail.com

github:@peter-c12