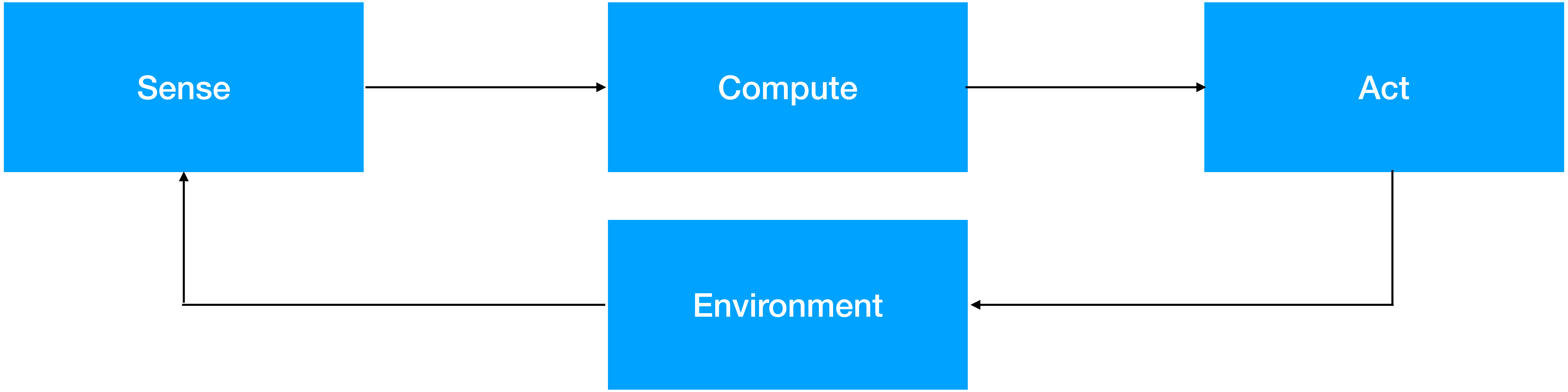
Perception

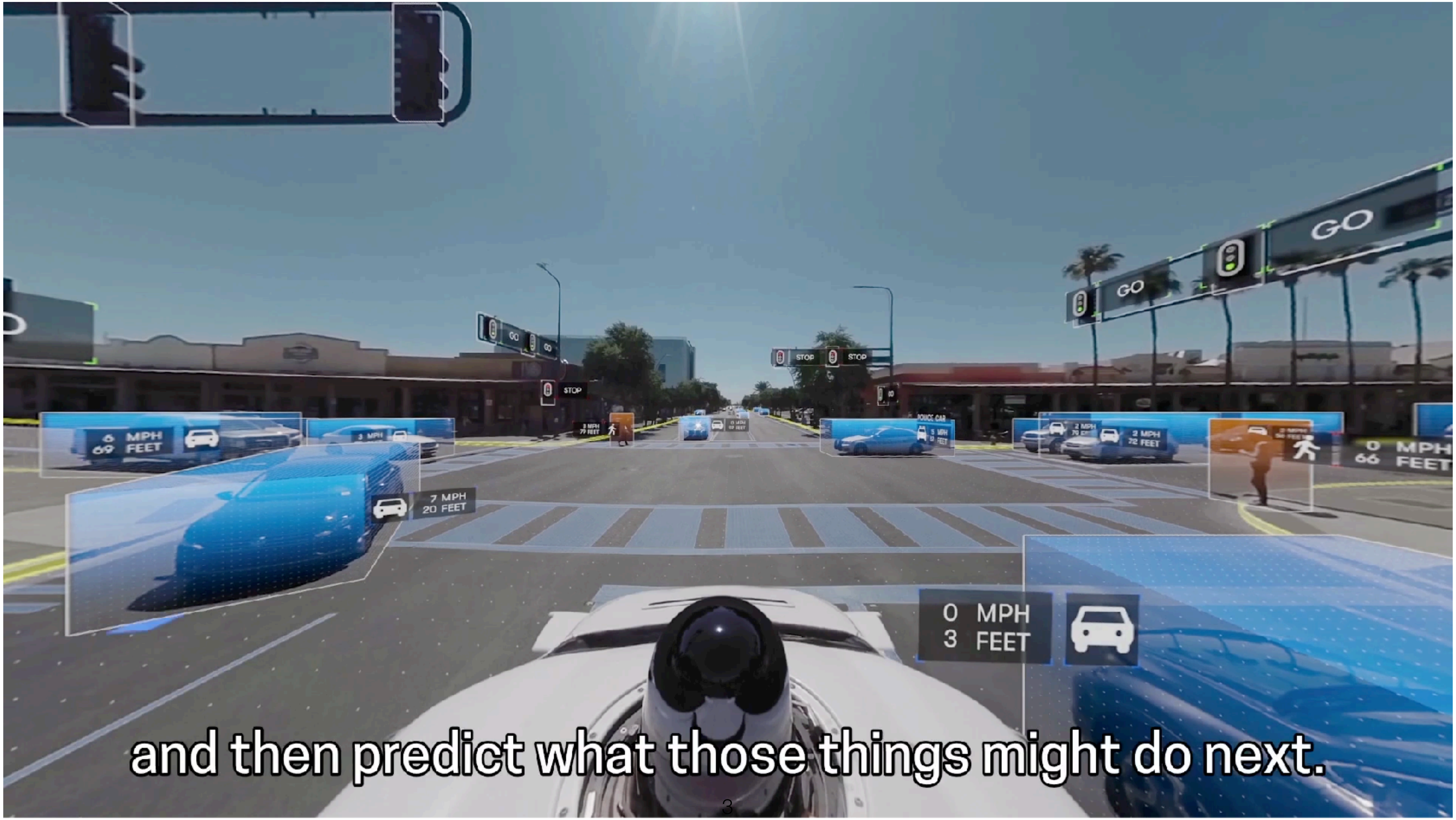
CS4501 - Robotics for Software Engineers

By Carl Hildebrandt

Robot Conceptual Architecture

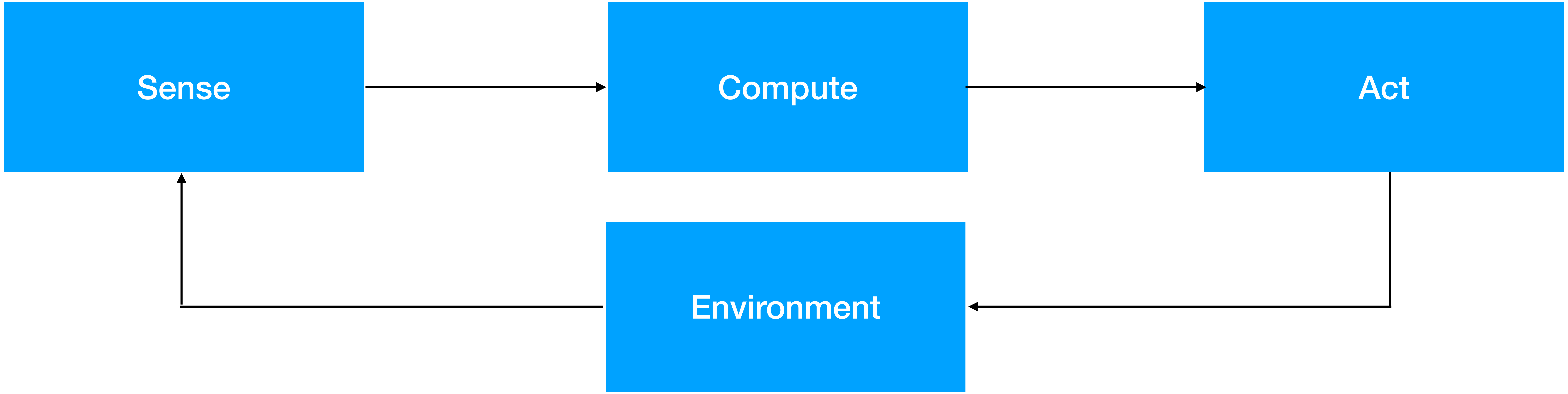


Self-driving Case Study



and then predict what those things might do next.

Robot Conceptual Architecture



Sense

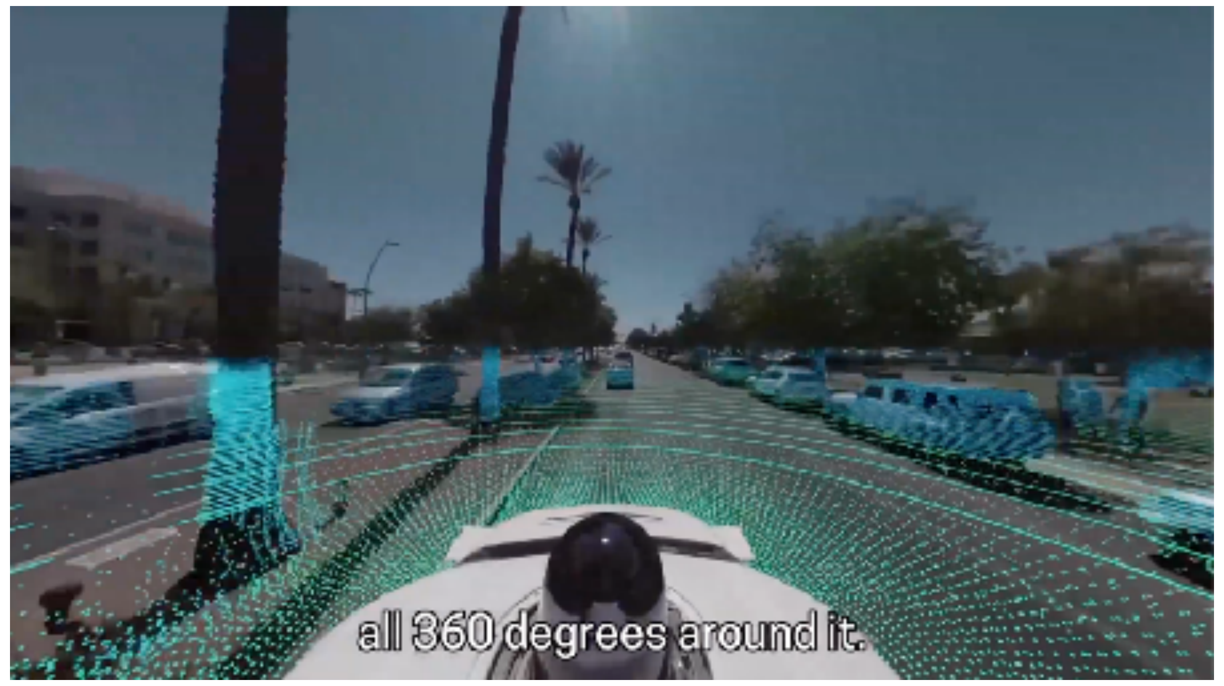
Compute

Act

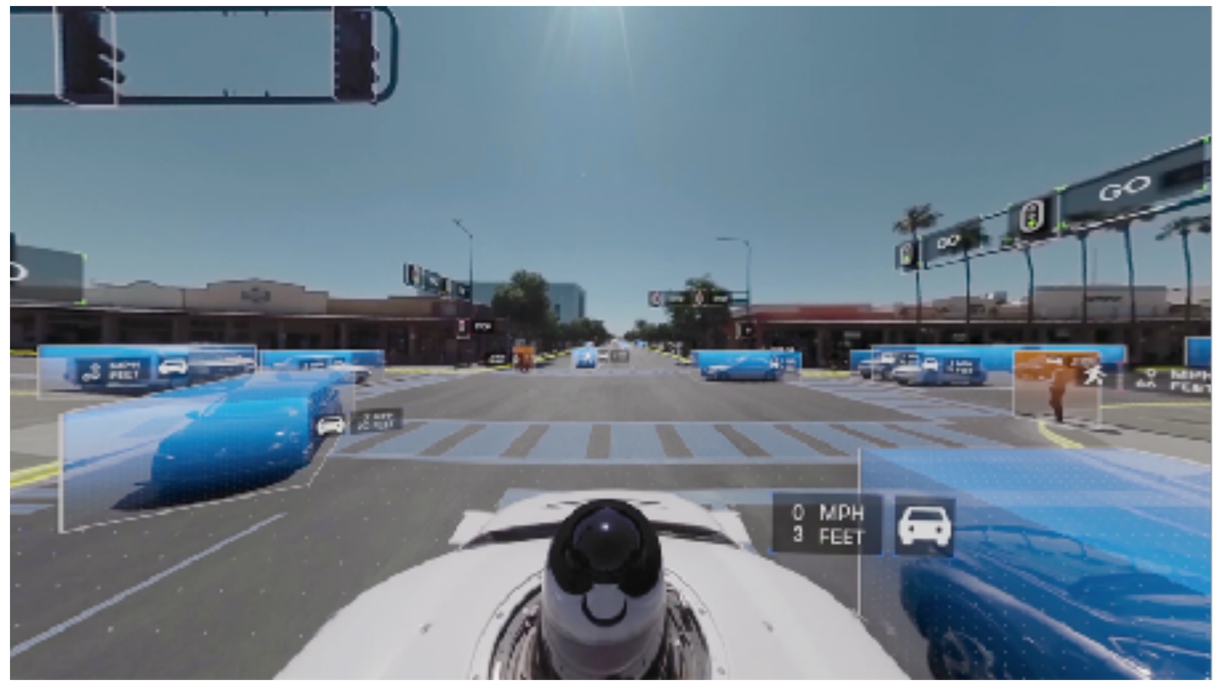
Environment

Robot Conceptual Architecture

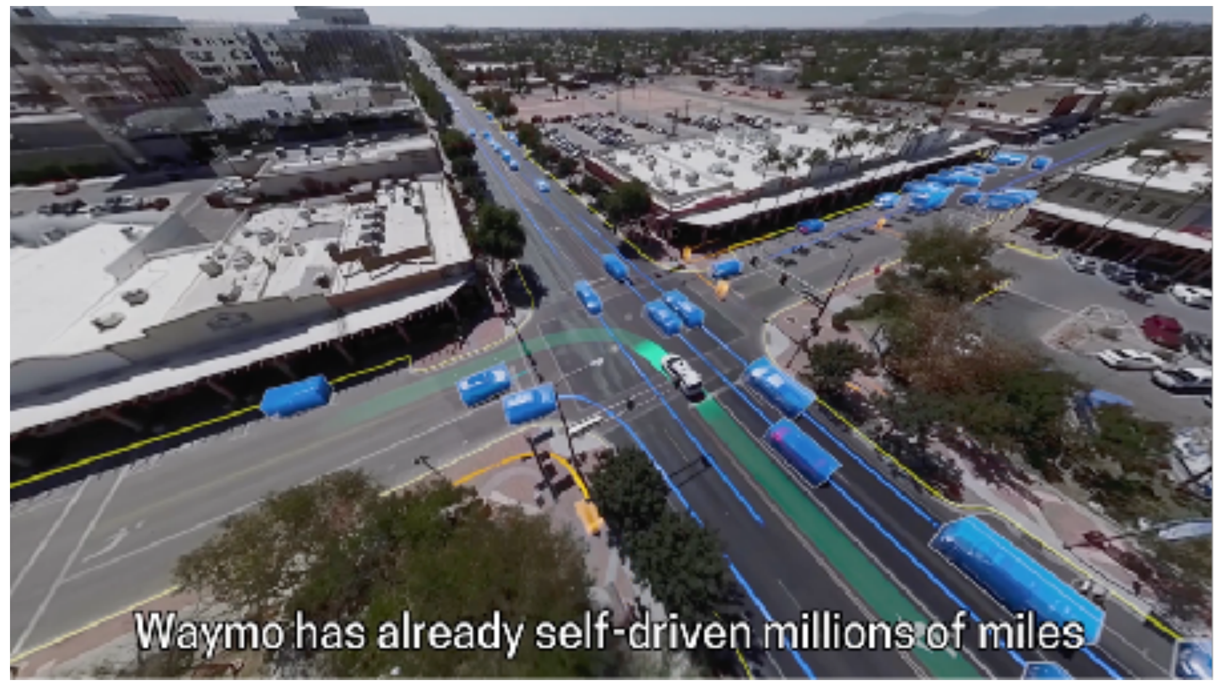
Sense



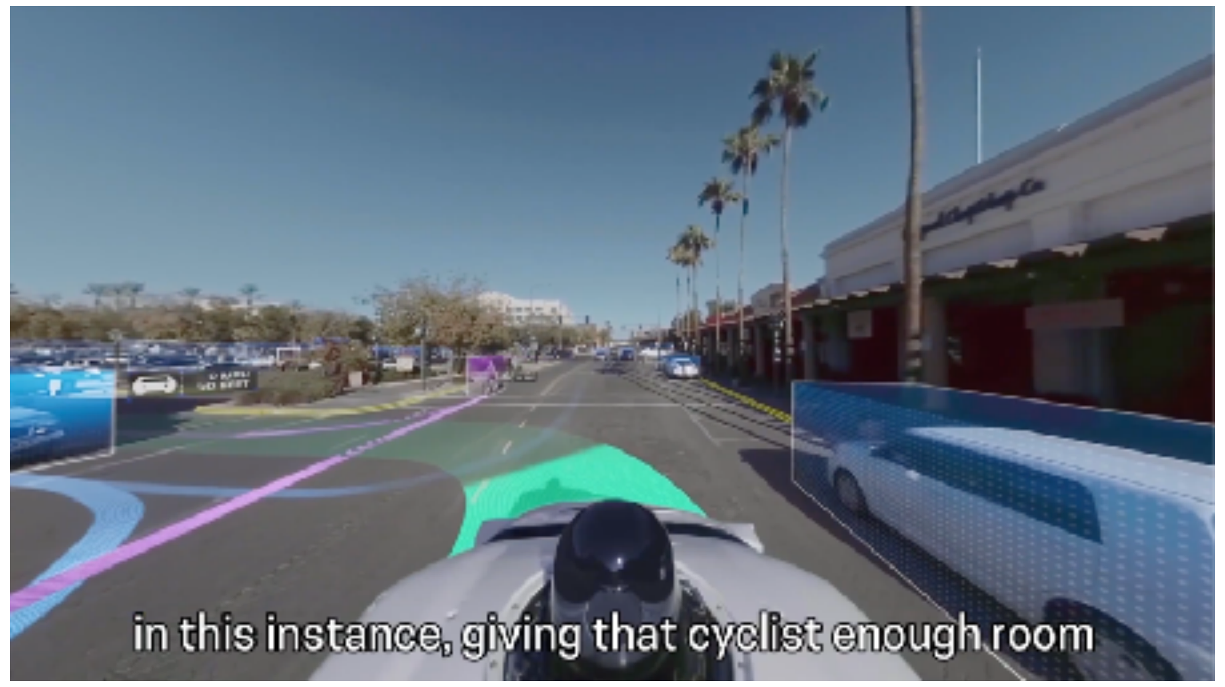
Perceive

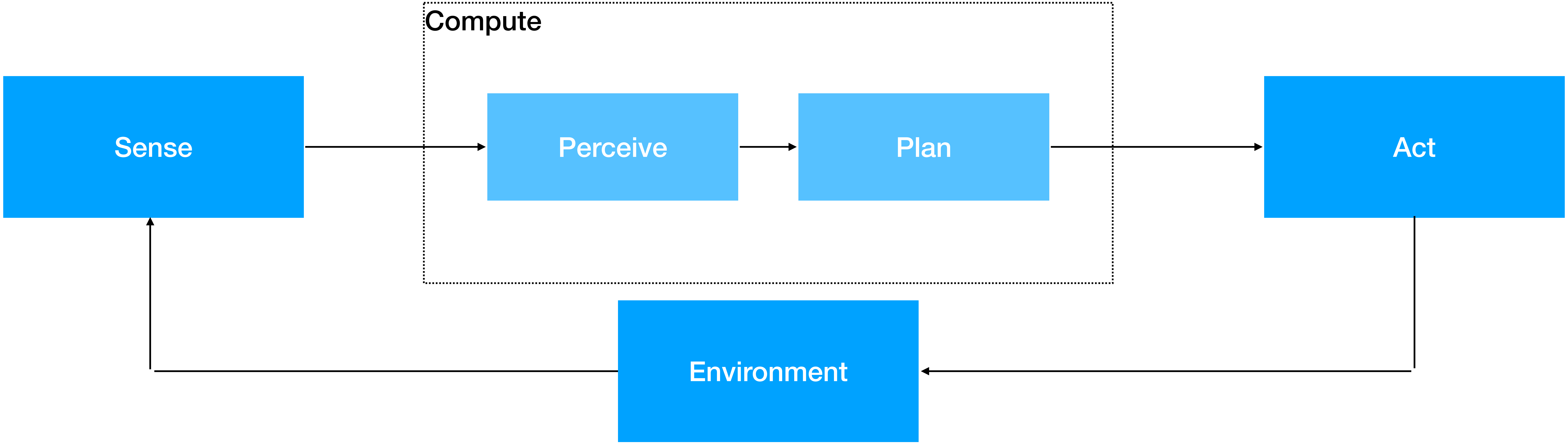


Plan



Act





Compute

Sense

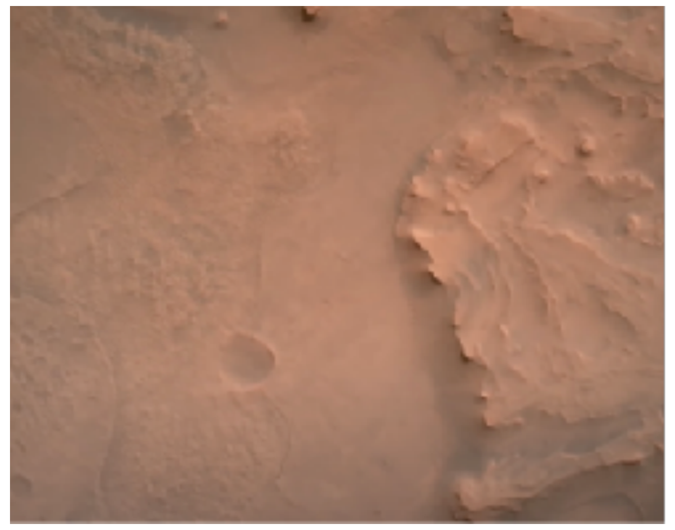
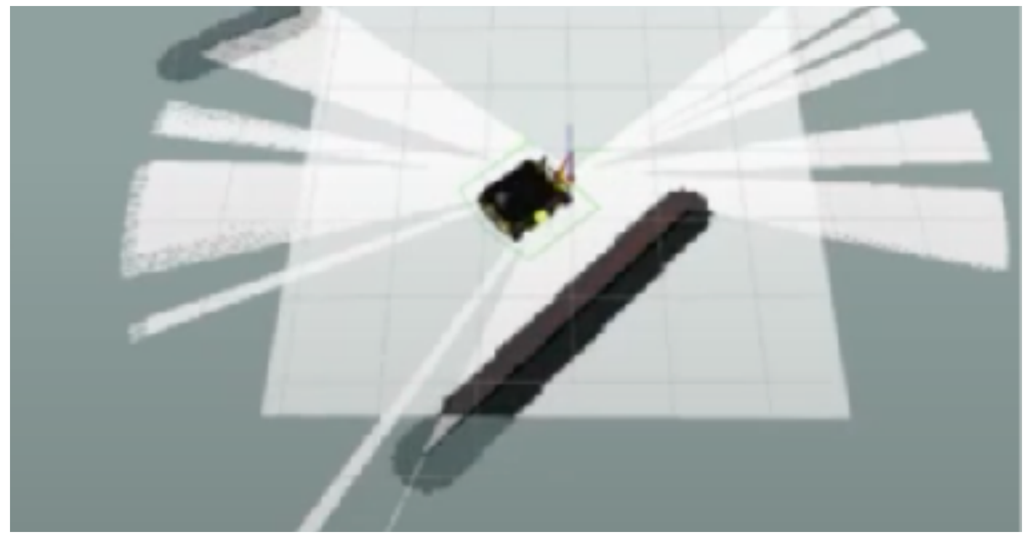
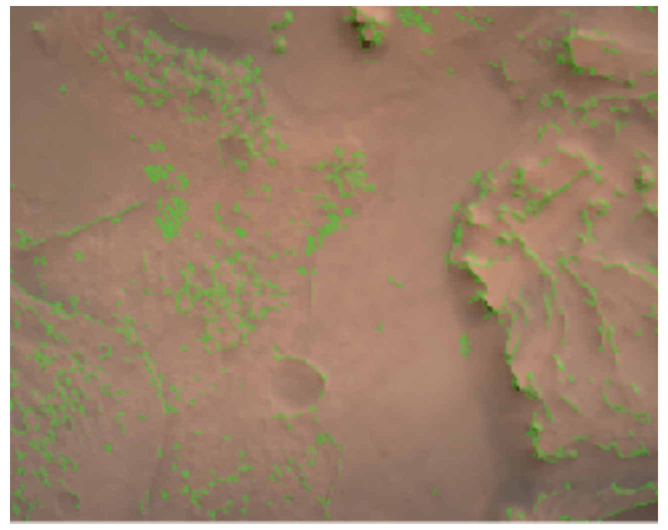
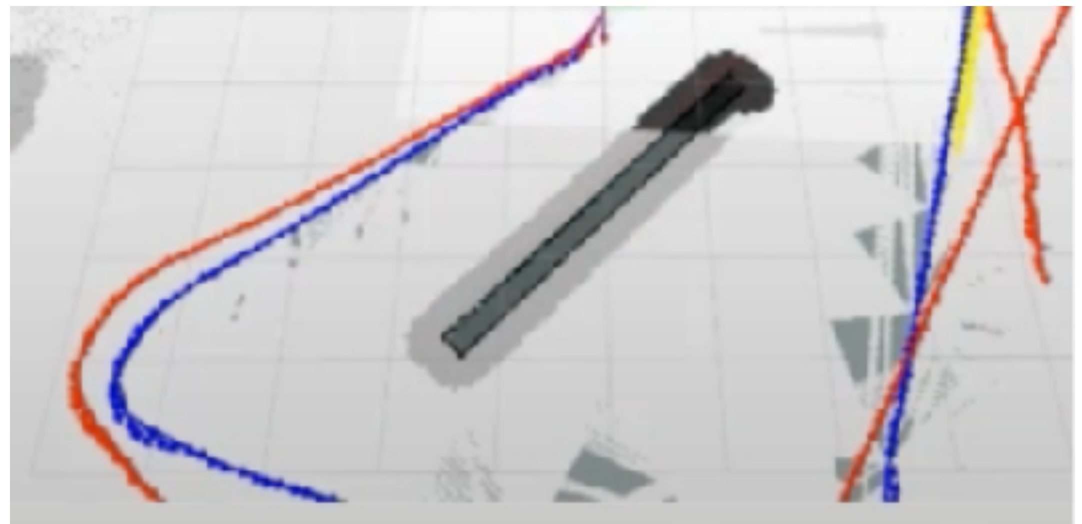
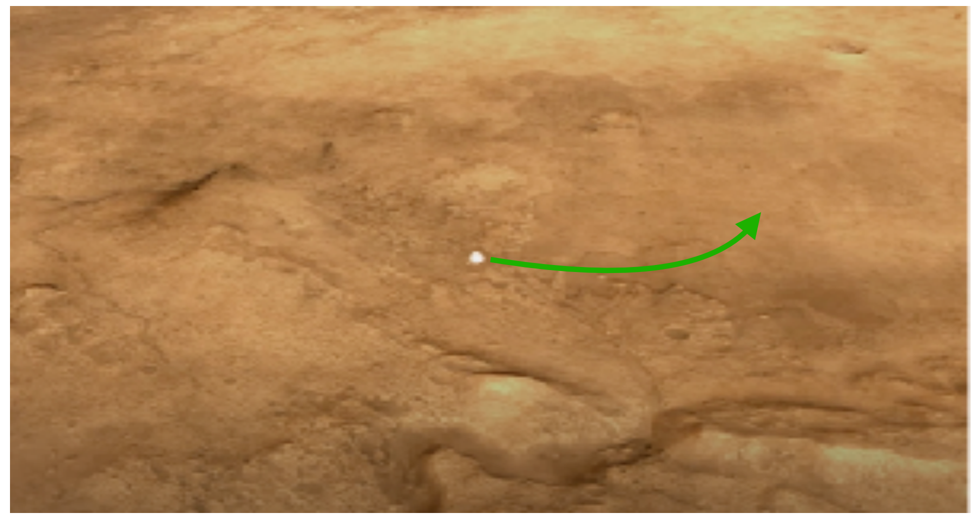
Perceive

Plan

Act

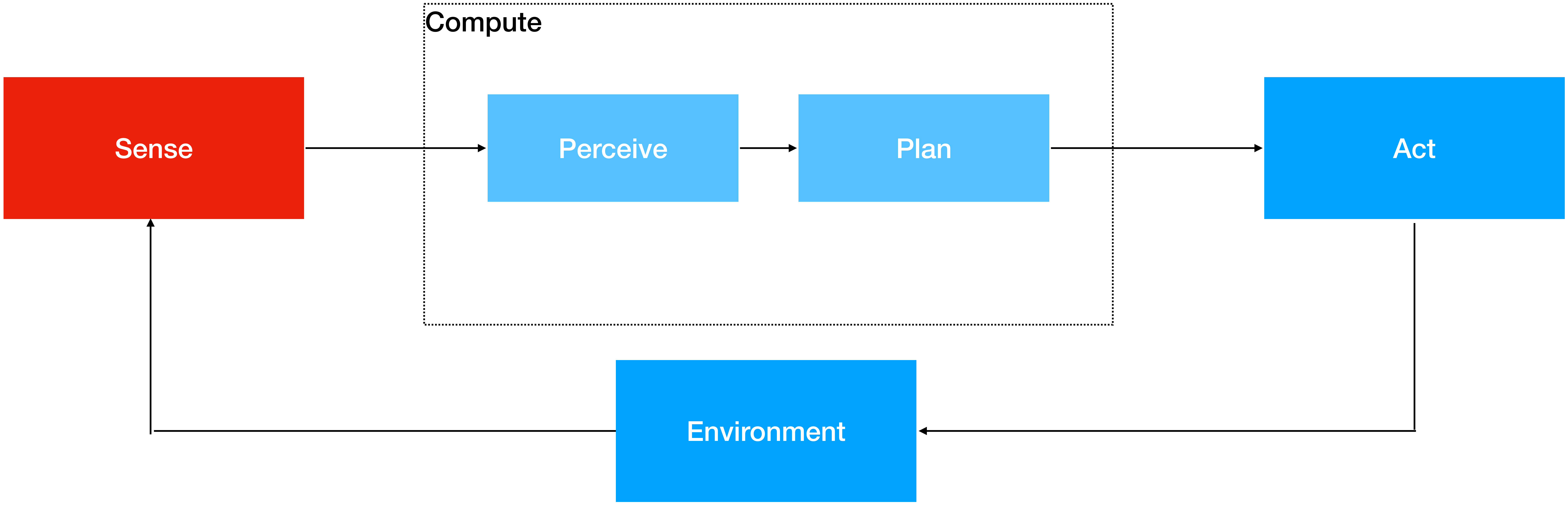
Environment

Robot Conceptual Architecture

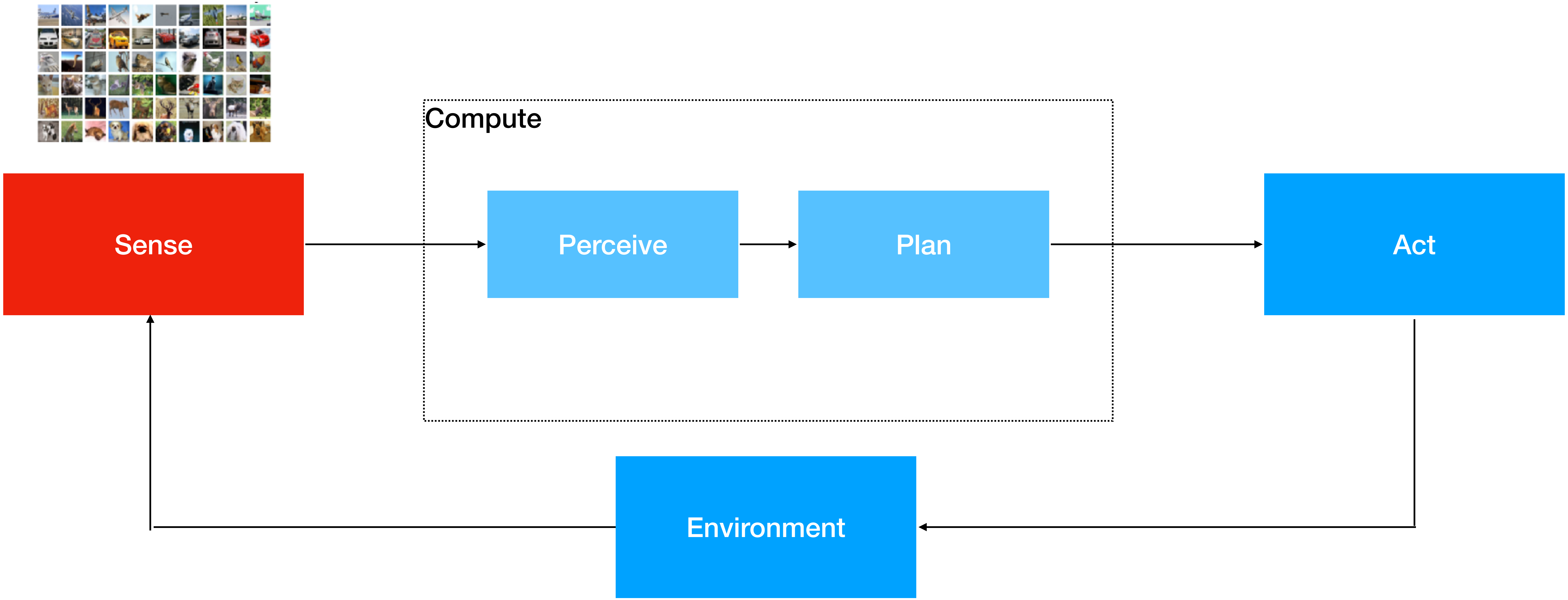
	Drone	Husky	Self Driving Car	Perseverance
Sense				
Perceive				
Plan				
Act				

all 360 degrees around it.
LEFT RIGHT
RGB Stereo Cameras
STEREO RECONSTRUCTION
0 MPH
3 FEET
The highest recorded speed was 18 m/s (40 mph).
Waymo has already self-driven millions of miles
in this instance, giving that cyclist enough room

Robot Conceptual Architecture



Robot Conceptual Architecture

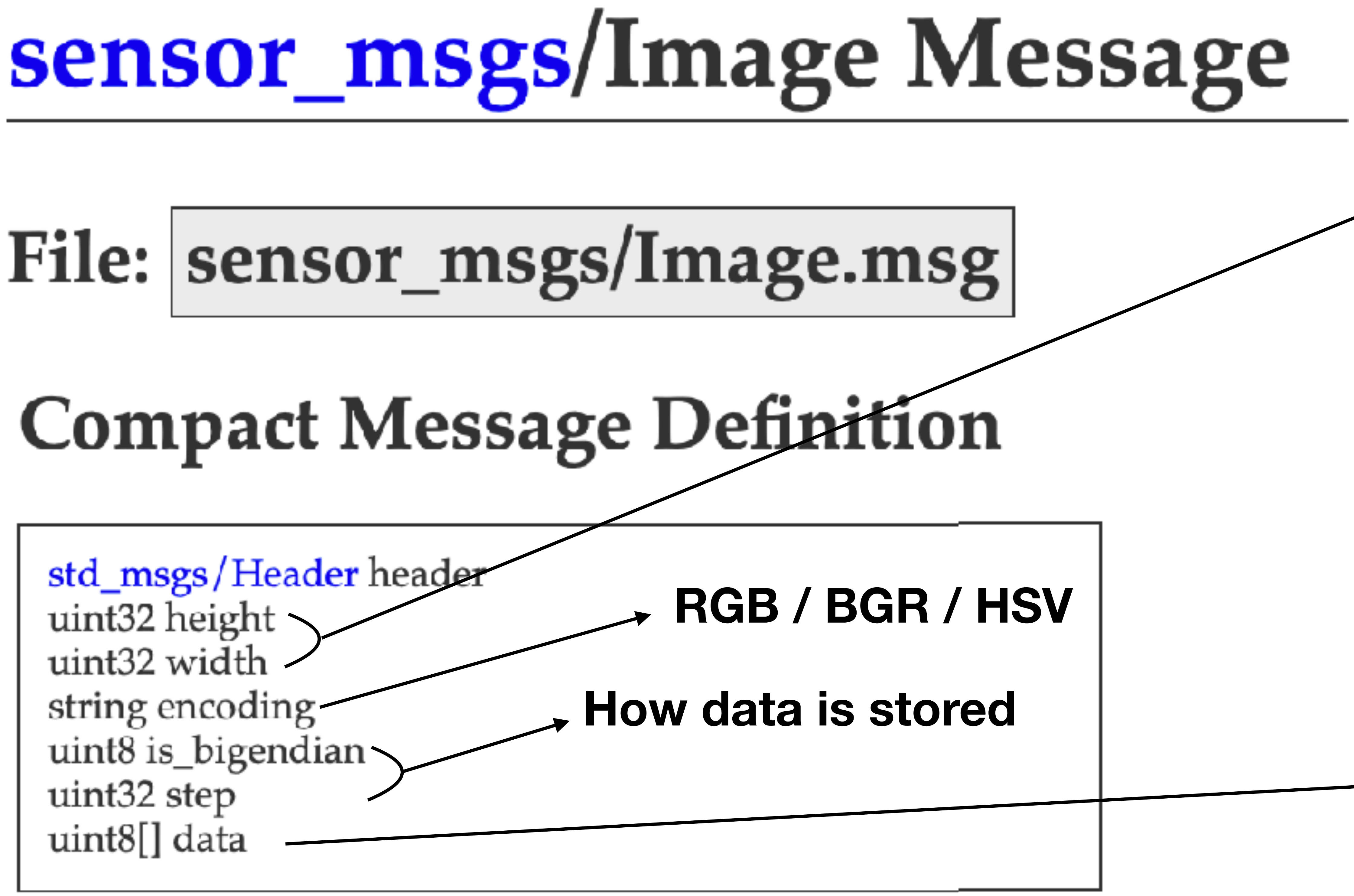


Question

What is an image to a robot?

Image Data

ROS: sensor_msgs/Image



sensor_msgs/Image Message

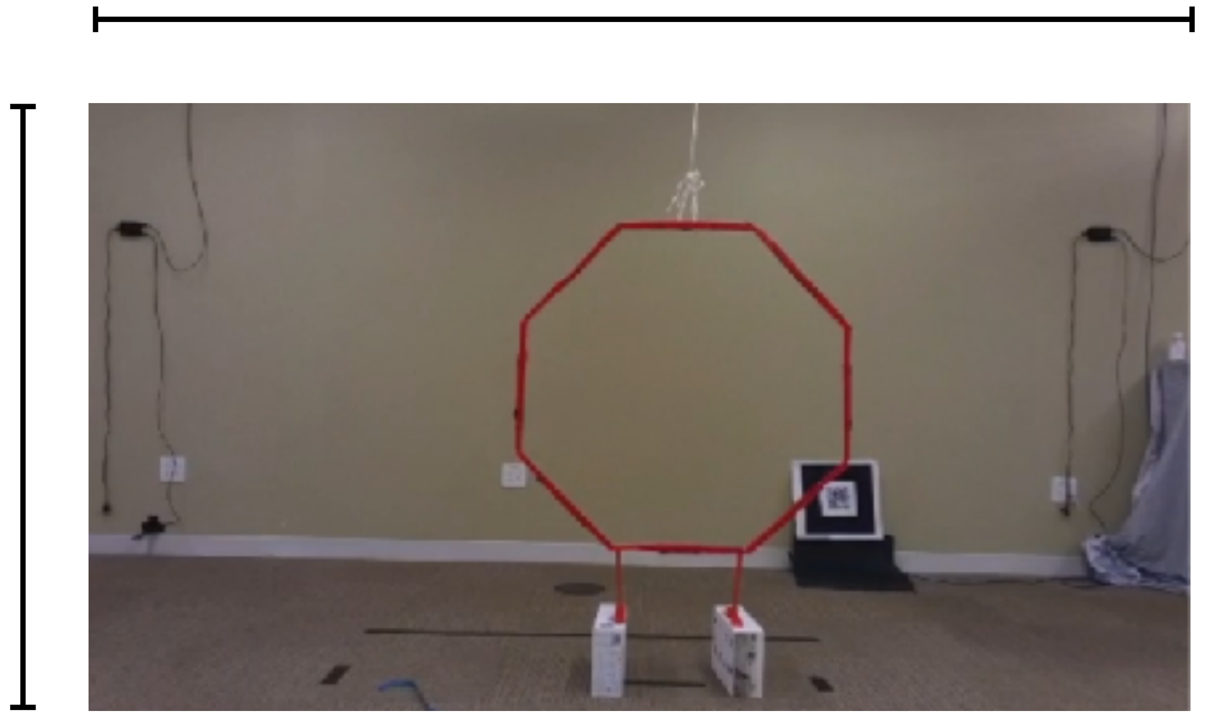
File: sensor_msgs/Image.msg

Compact Message Definition

```
std_msgs/Header header
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step
uint8[] data
```

RGB / BGR / HSV

How data is stored



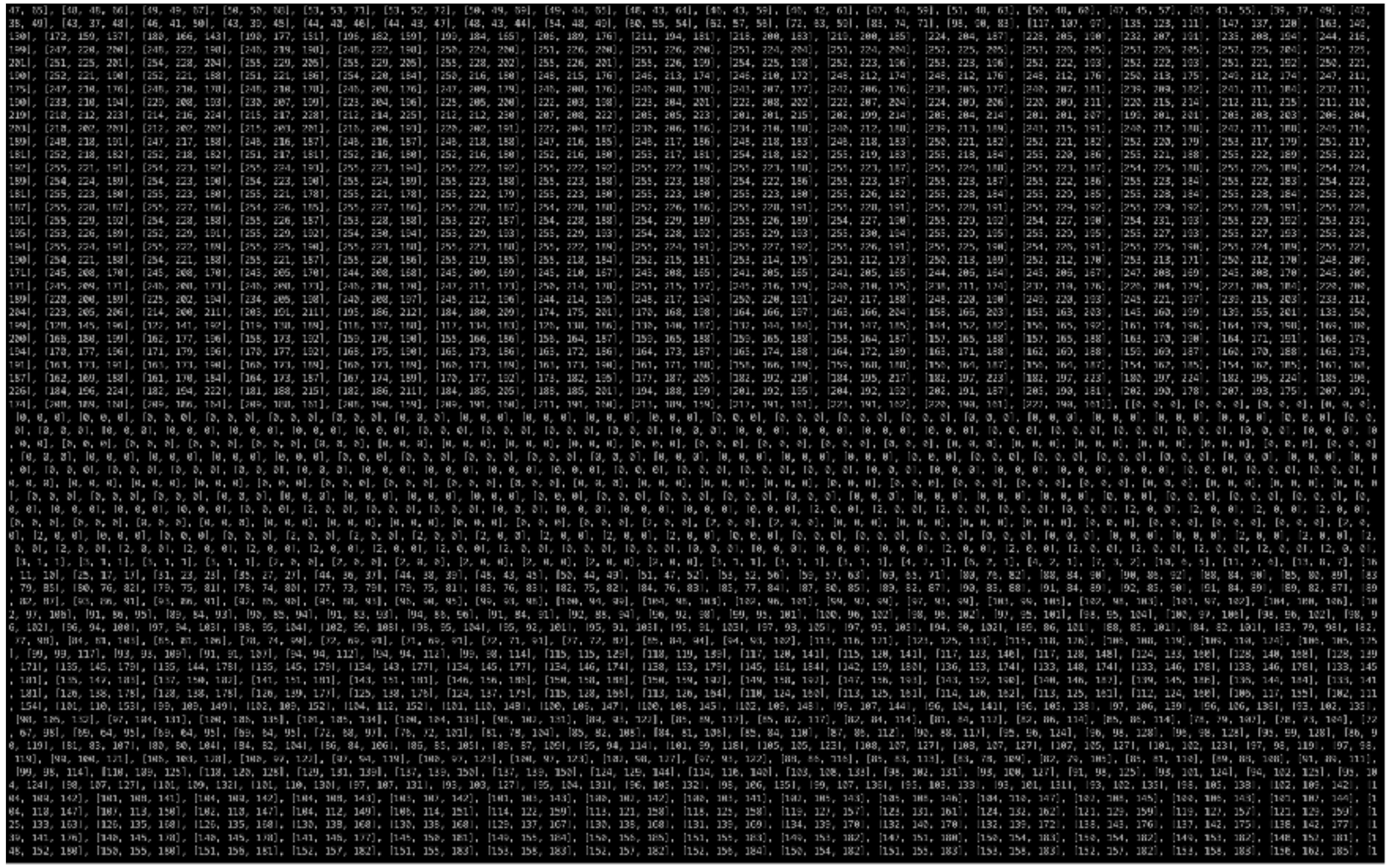
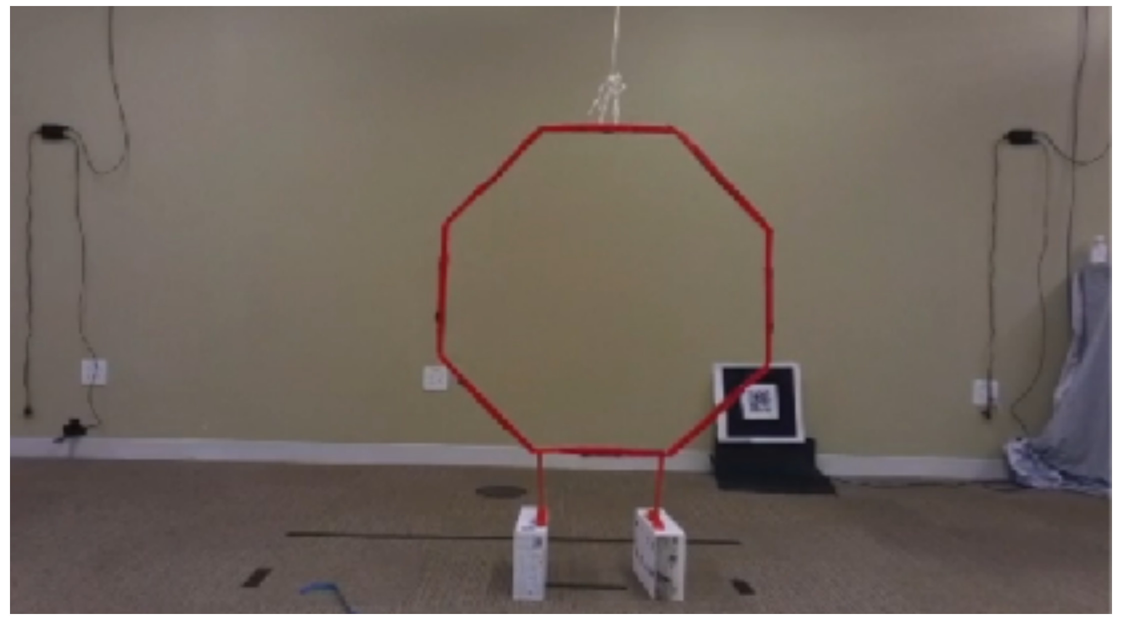
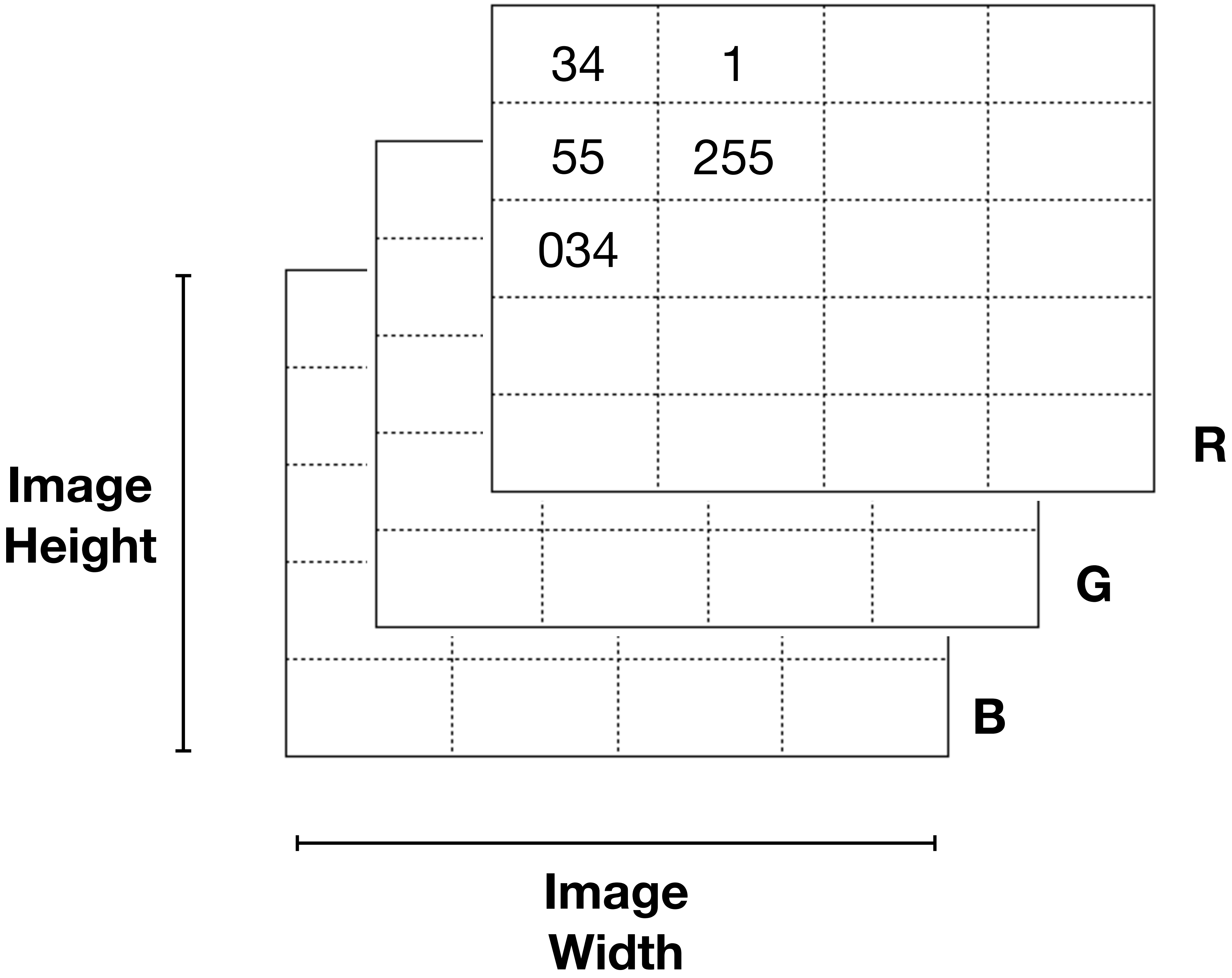
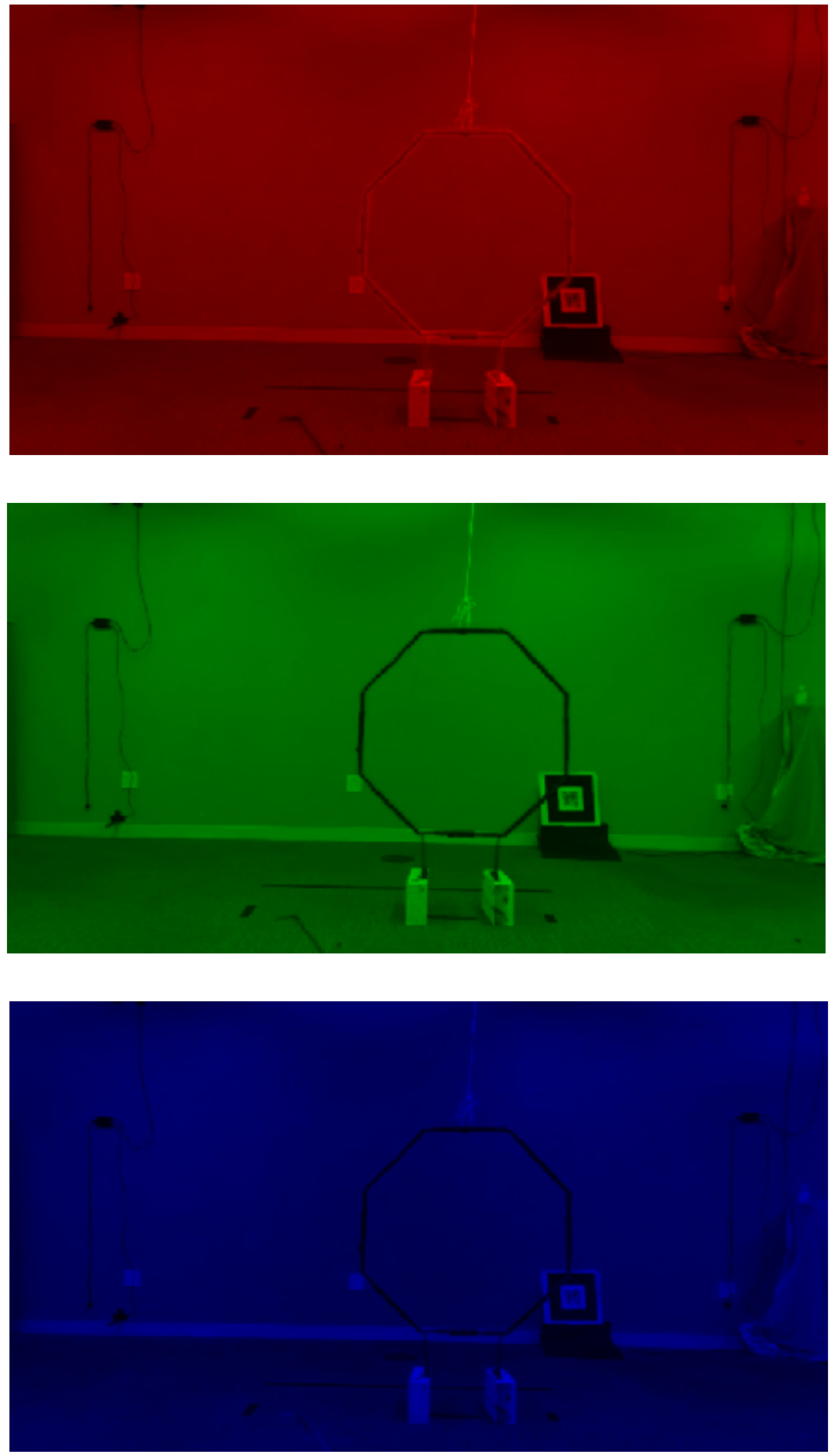


Image Data



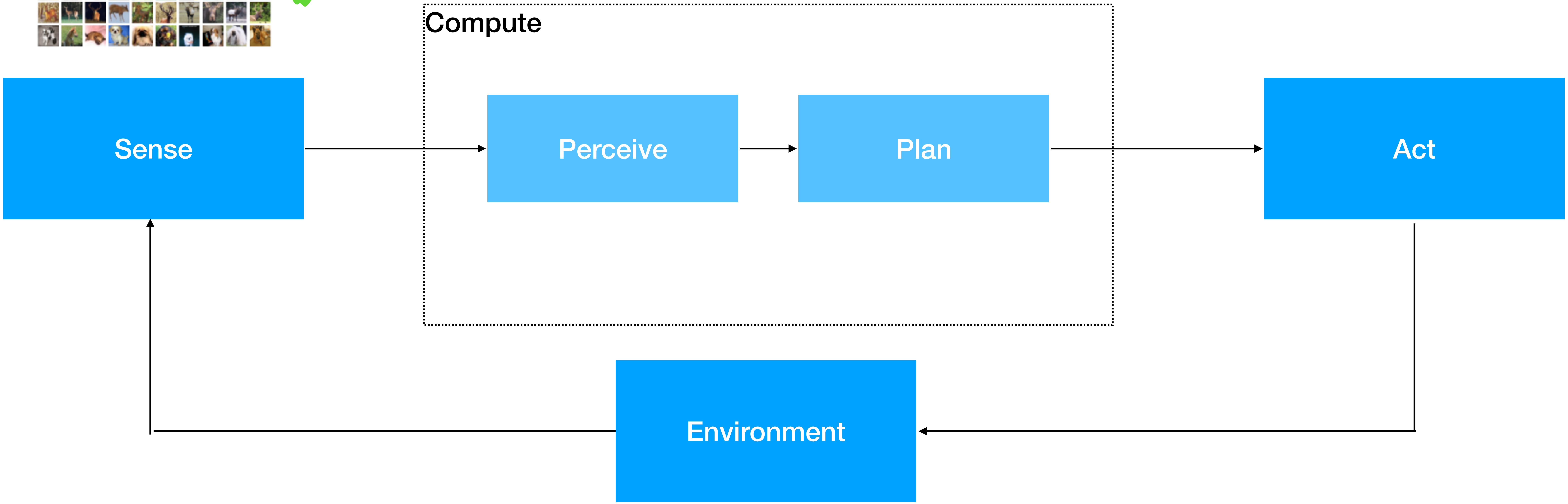


34	1		
55	255		
034			

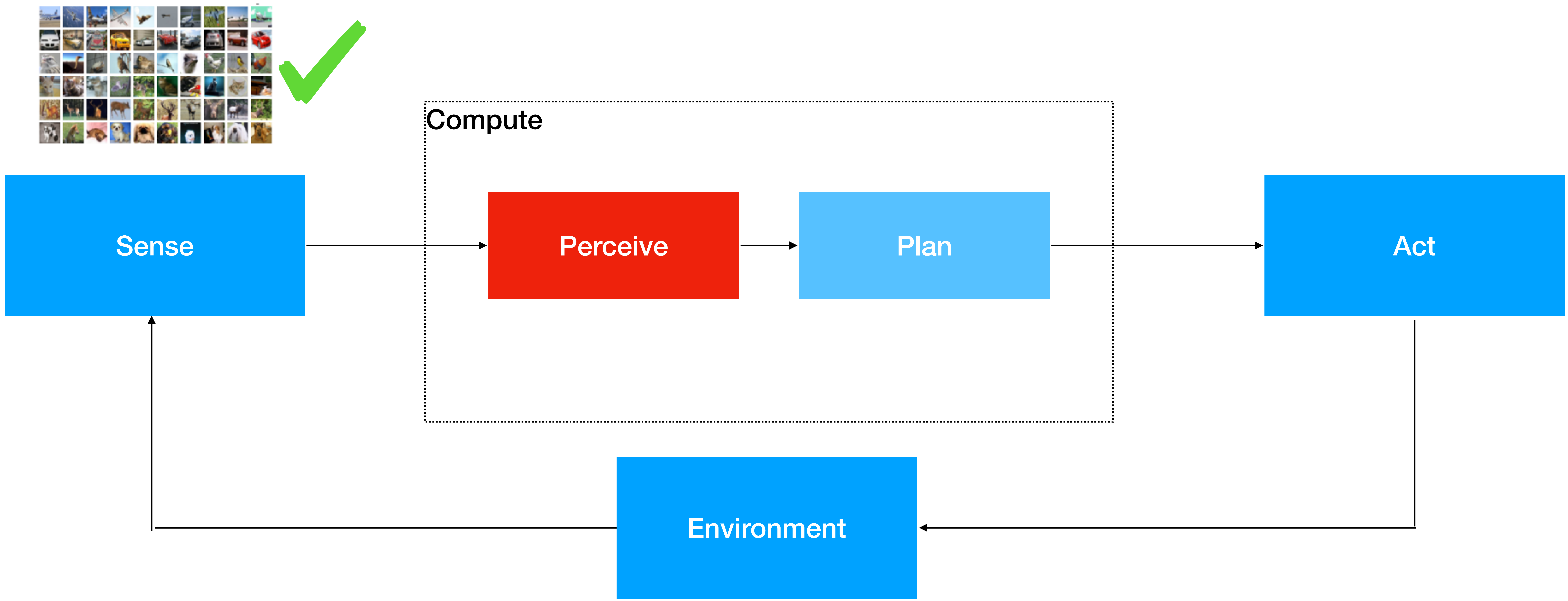


Robot Conceptual Architecture

✓



Robot Conceptual Architecture



Perception

“Perception refers to the ability of an autonomous system to collect information and extract relevant knowledge from the environment.”

–Pendleton, Scott Drew, et al. "Perception, planning, control, and coordination for autonomous vehicles." Machines 5.1 (2017)

Perception

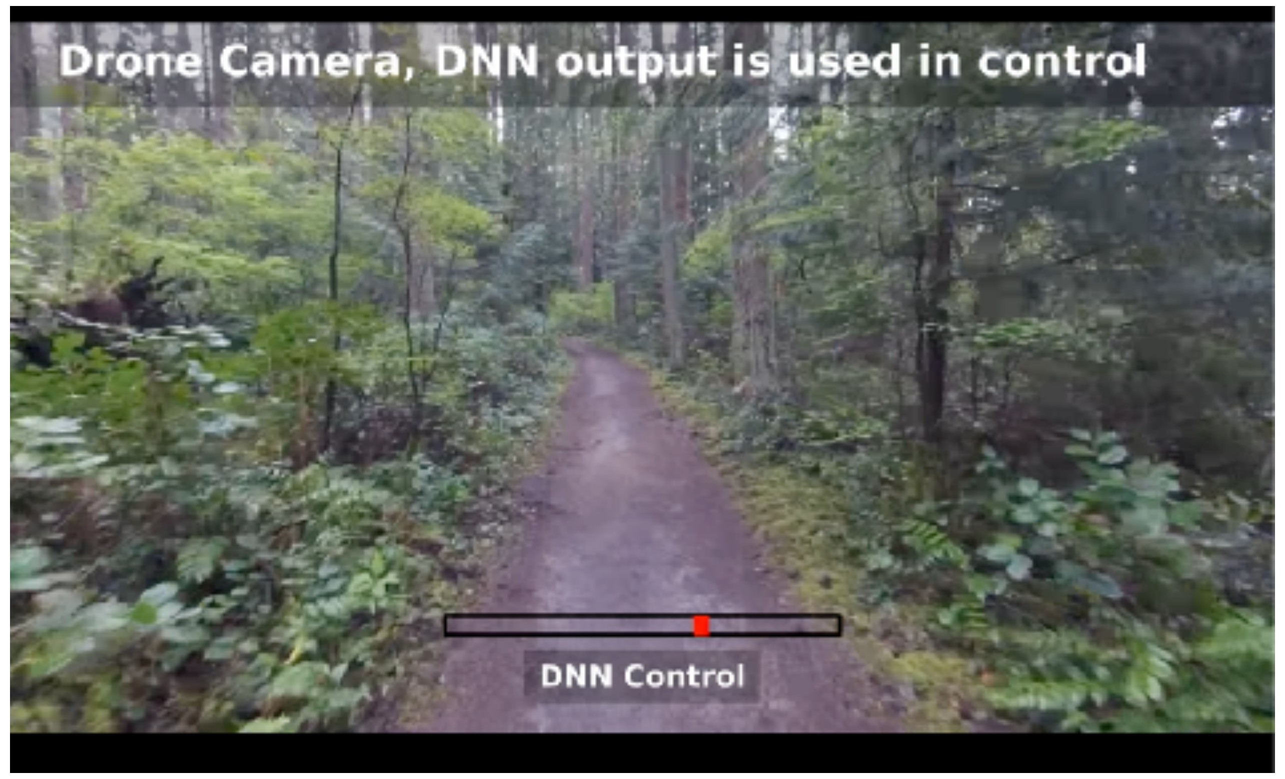
“Perception refers to the ability of an autonomous system to collect information and extract relevant knowledge from the environment.”

–Pendleton, Scott Drew, et al. "Perception, planning, control, and coordination for autonomous vehicles." Machines 5.1 (2017)

Perception Examples

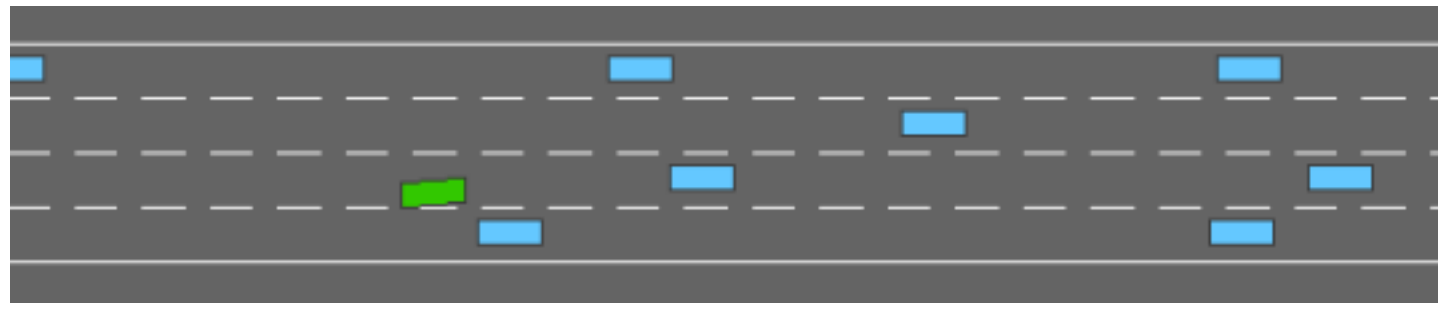
How: Processing sensor data to create a higher-level abstraction of the data

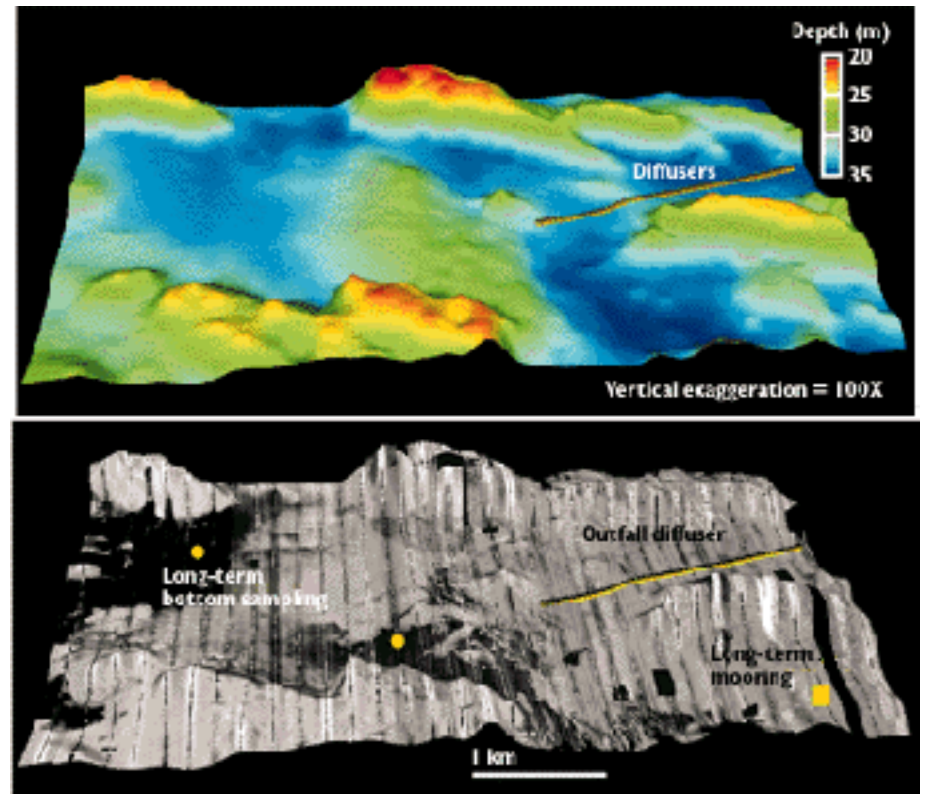
Camera Sensor

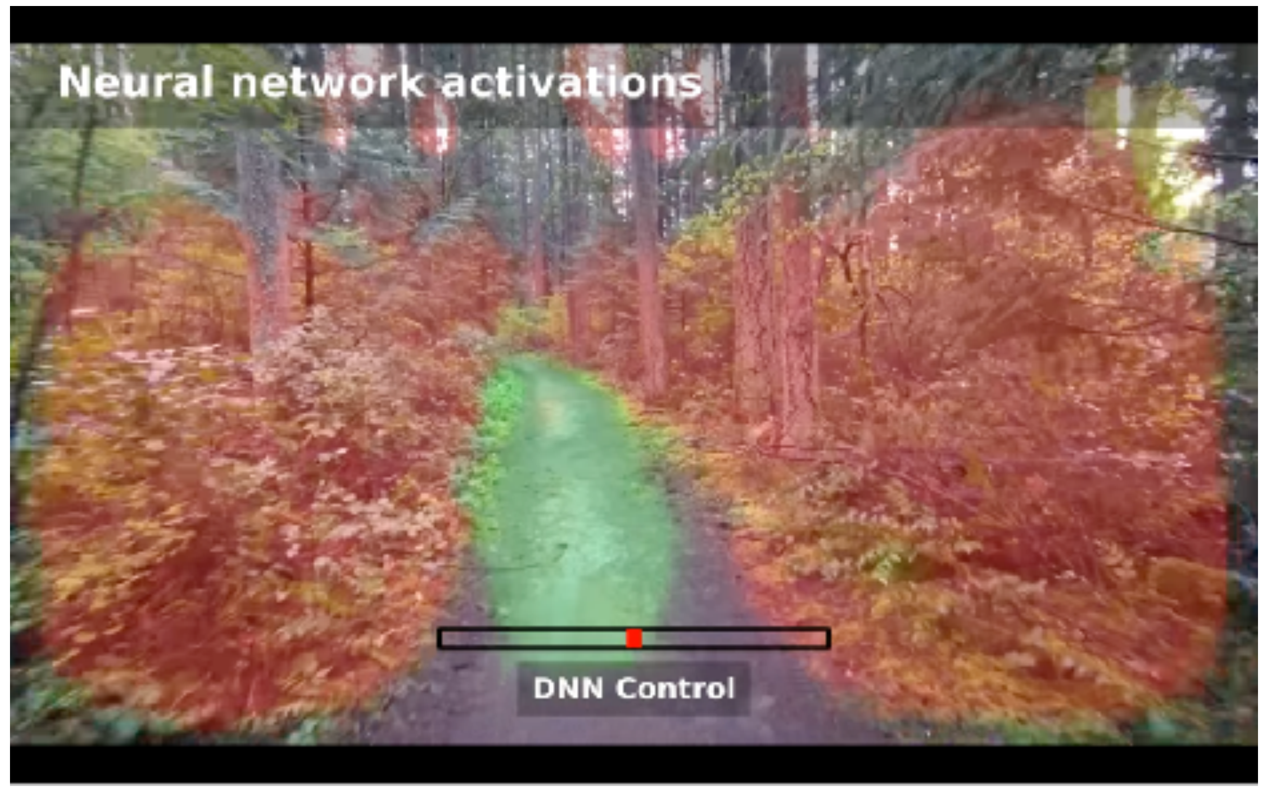


Perception

Traffic Light: Stop



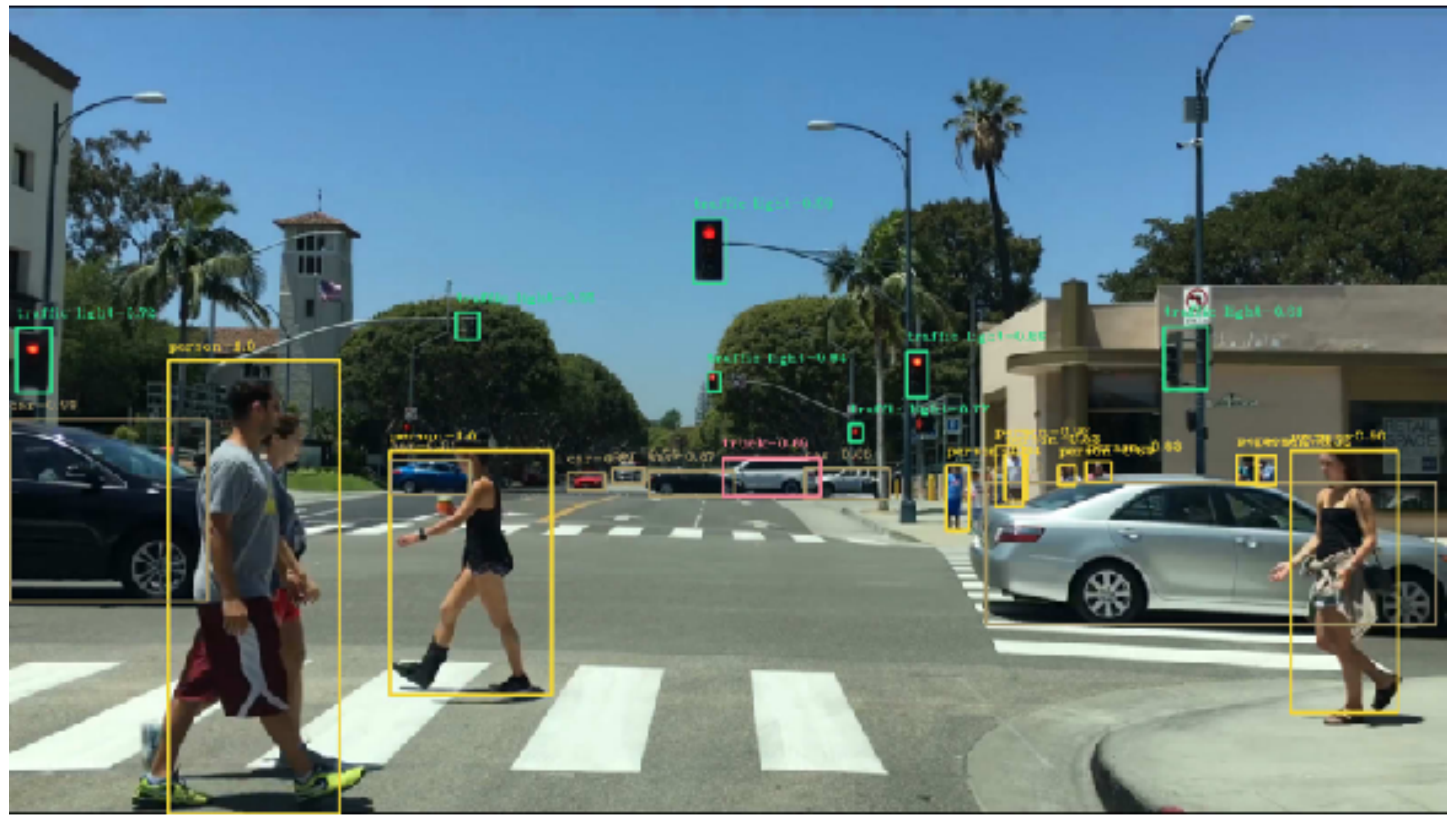




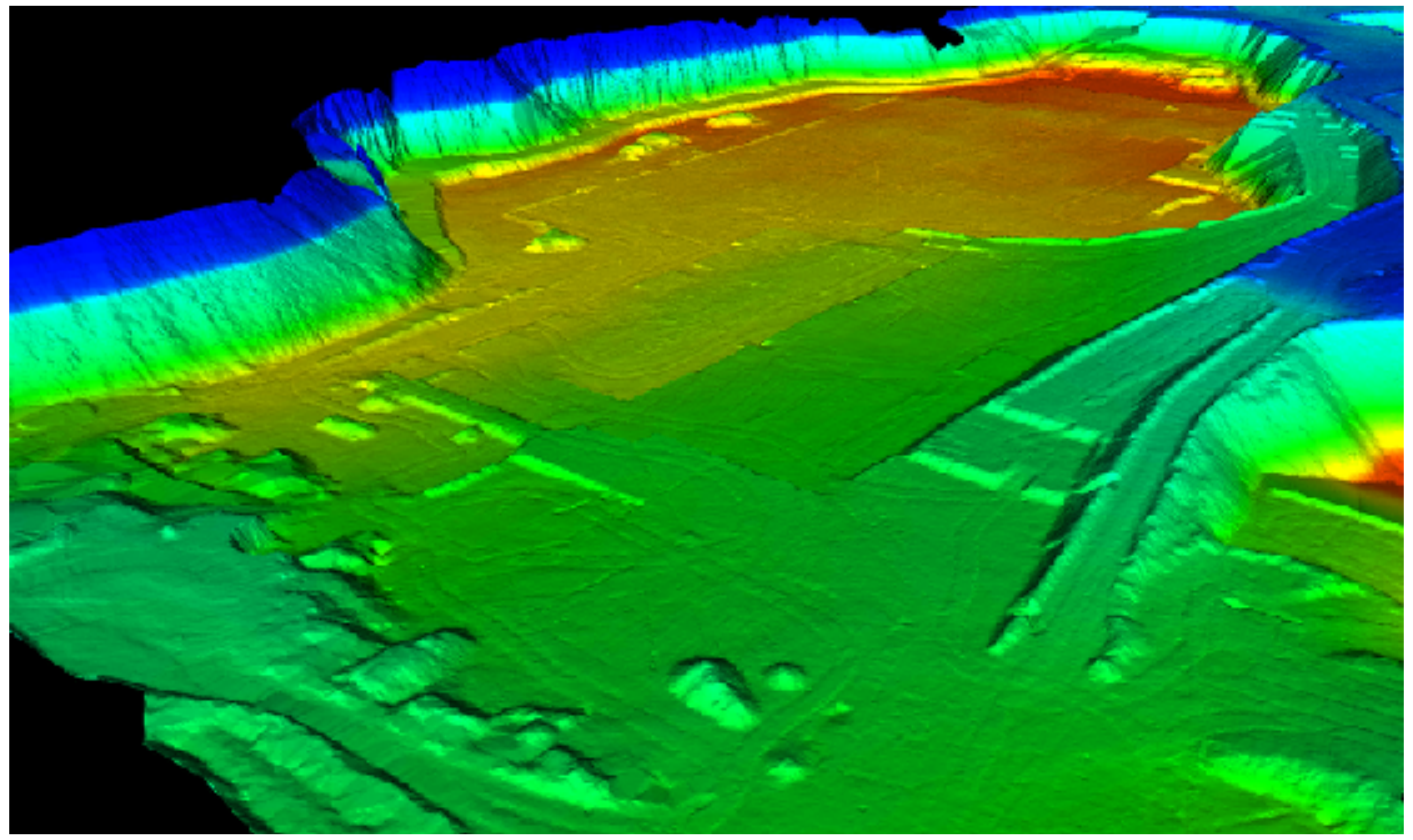
Main Types of Perception

Classification

Object Detection



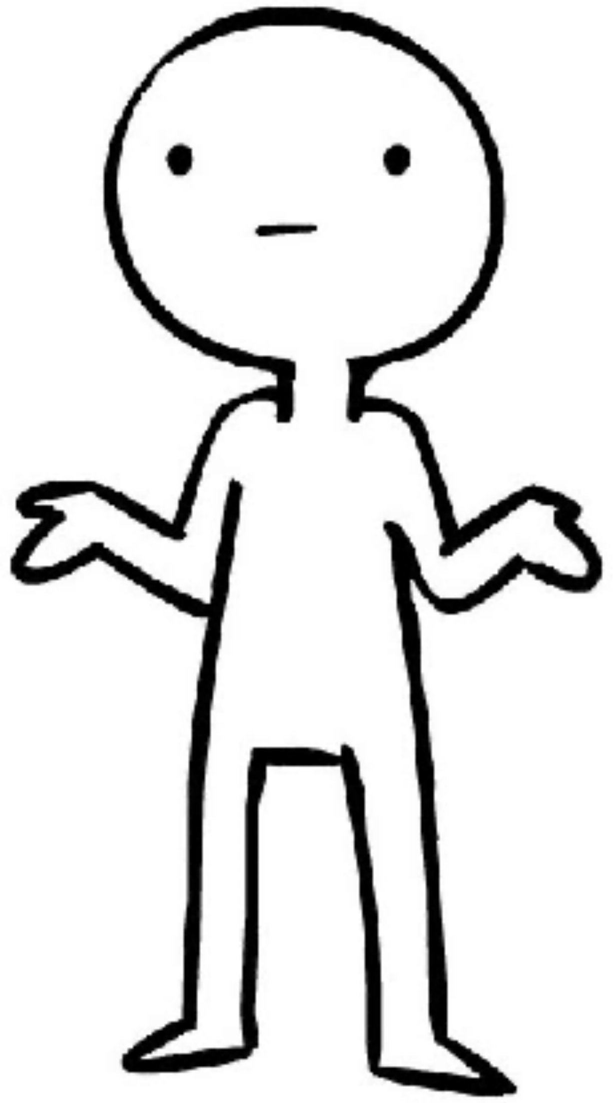
Interpretation



Perception

What: extract relevant knowledge from the environment
Input: Raw data
Output: Classication / Object Detection / Interpretation ✓

How: ?



Perception Algorithms

Perception estimates the state of the environment

Image Processing Algorithms
An image is processed through parameterized transformations.

Key: We define this function

Machine Learning
Gather large amounts of data to learn or approximate the desired function.

Key: The computer learns this function

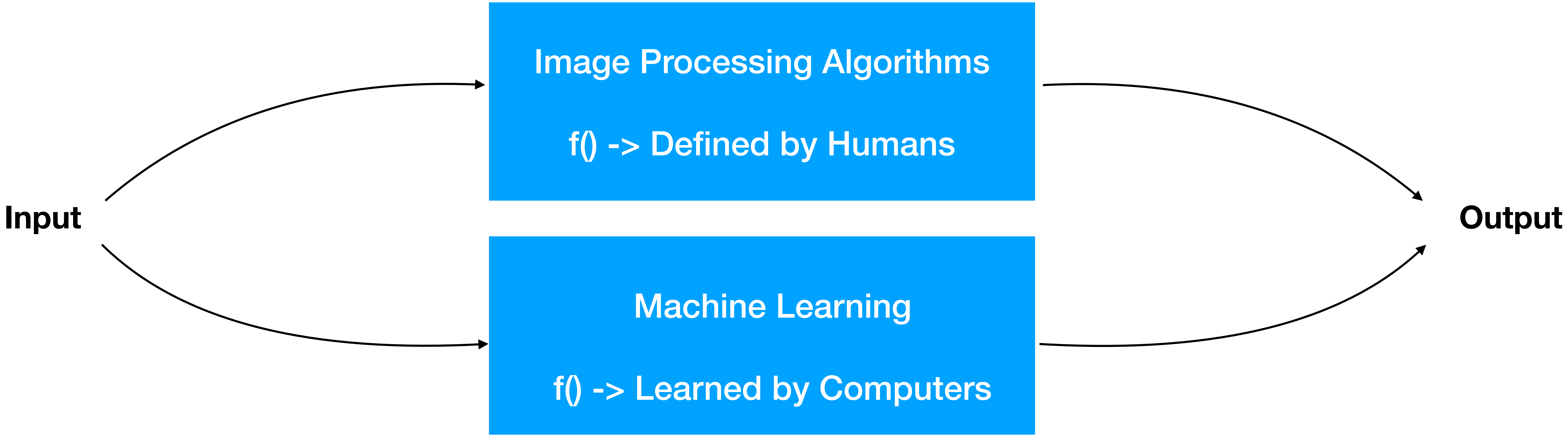

Input

Image Processing Algorithms

f() -> Defined by Humans

Machine Learning

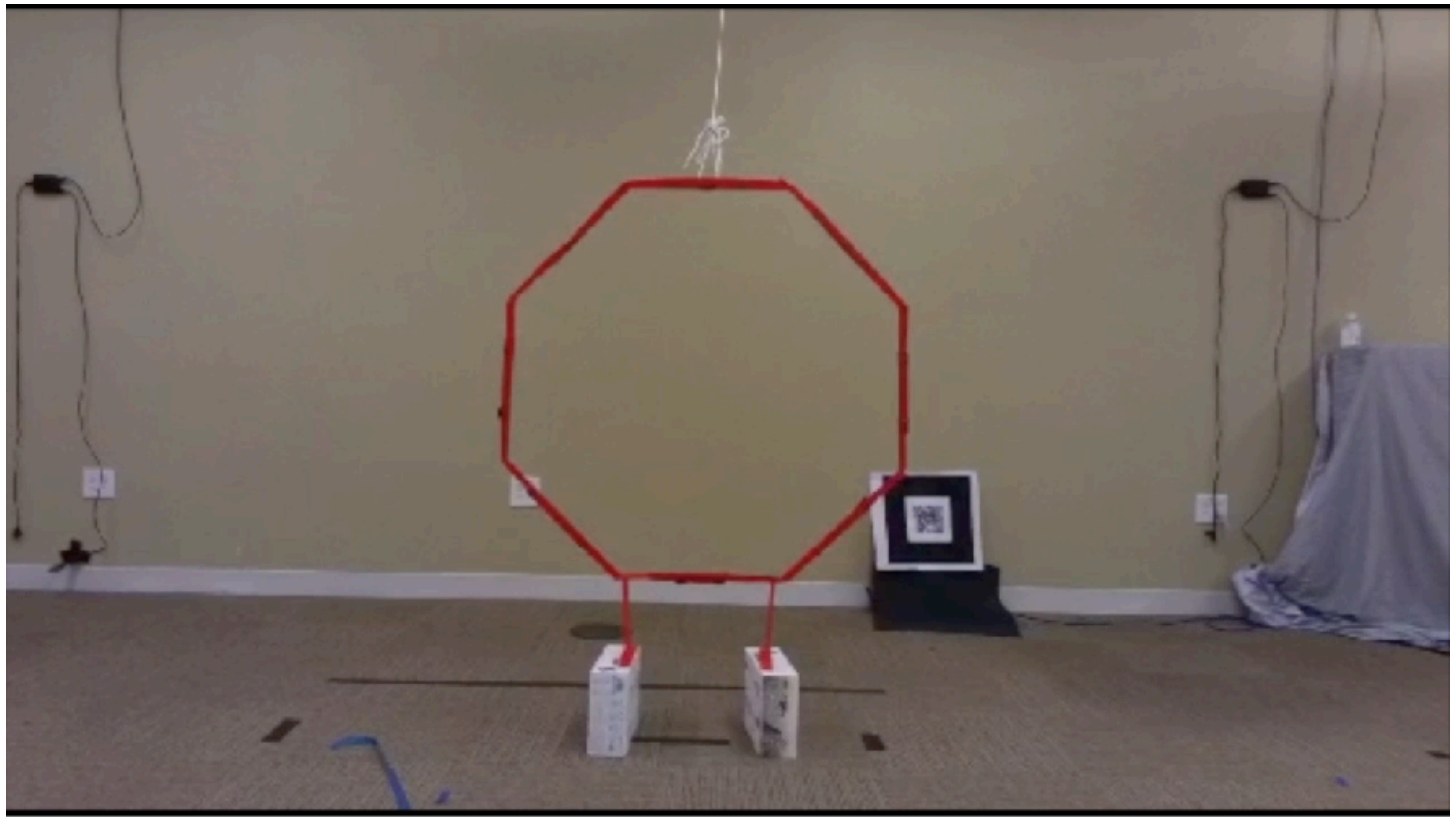
f() -> Learned by Computers

Output

Example

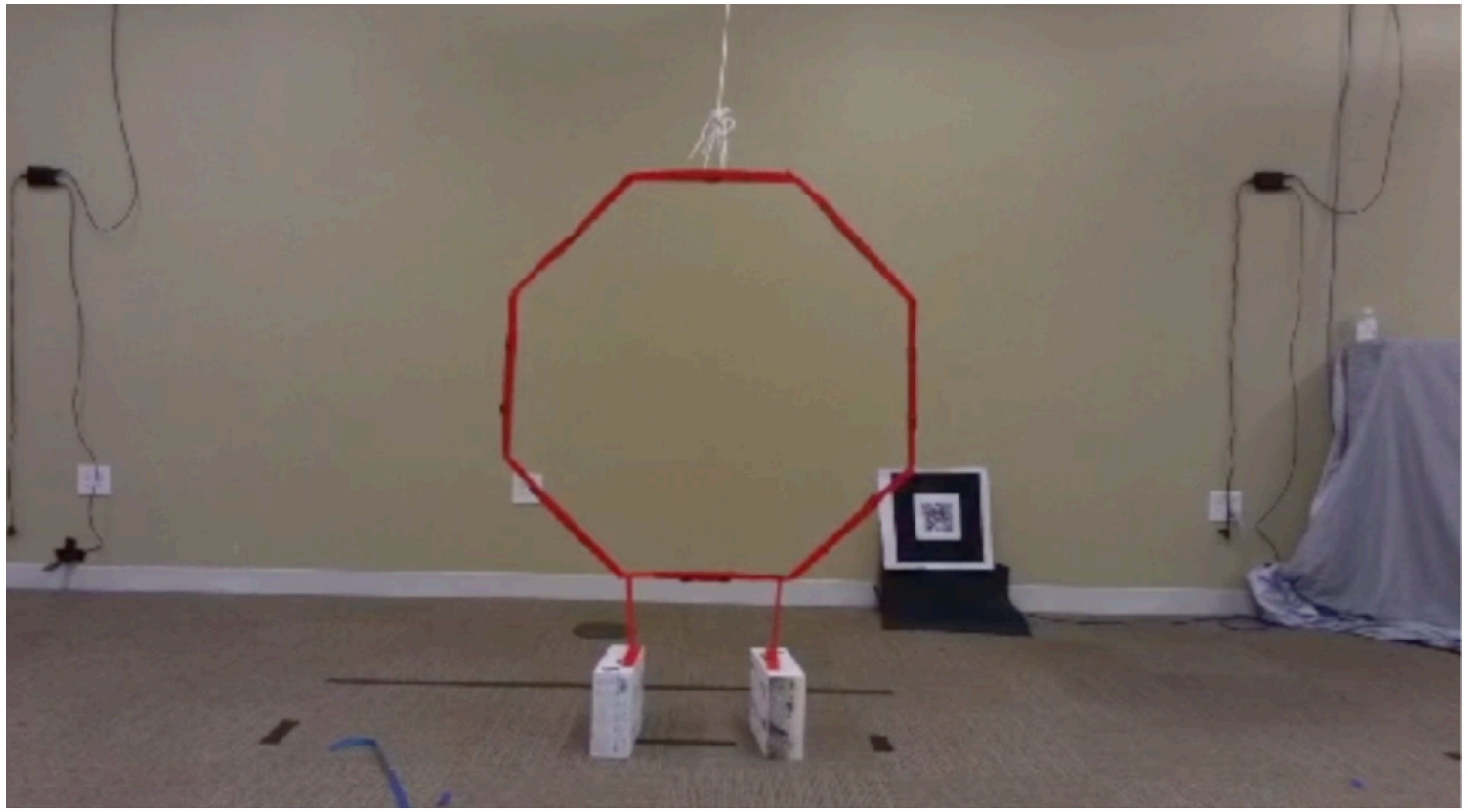
Example

Input



Example

Input



Output

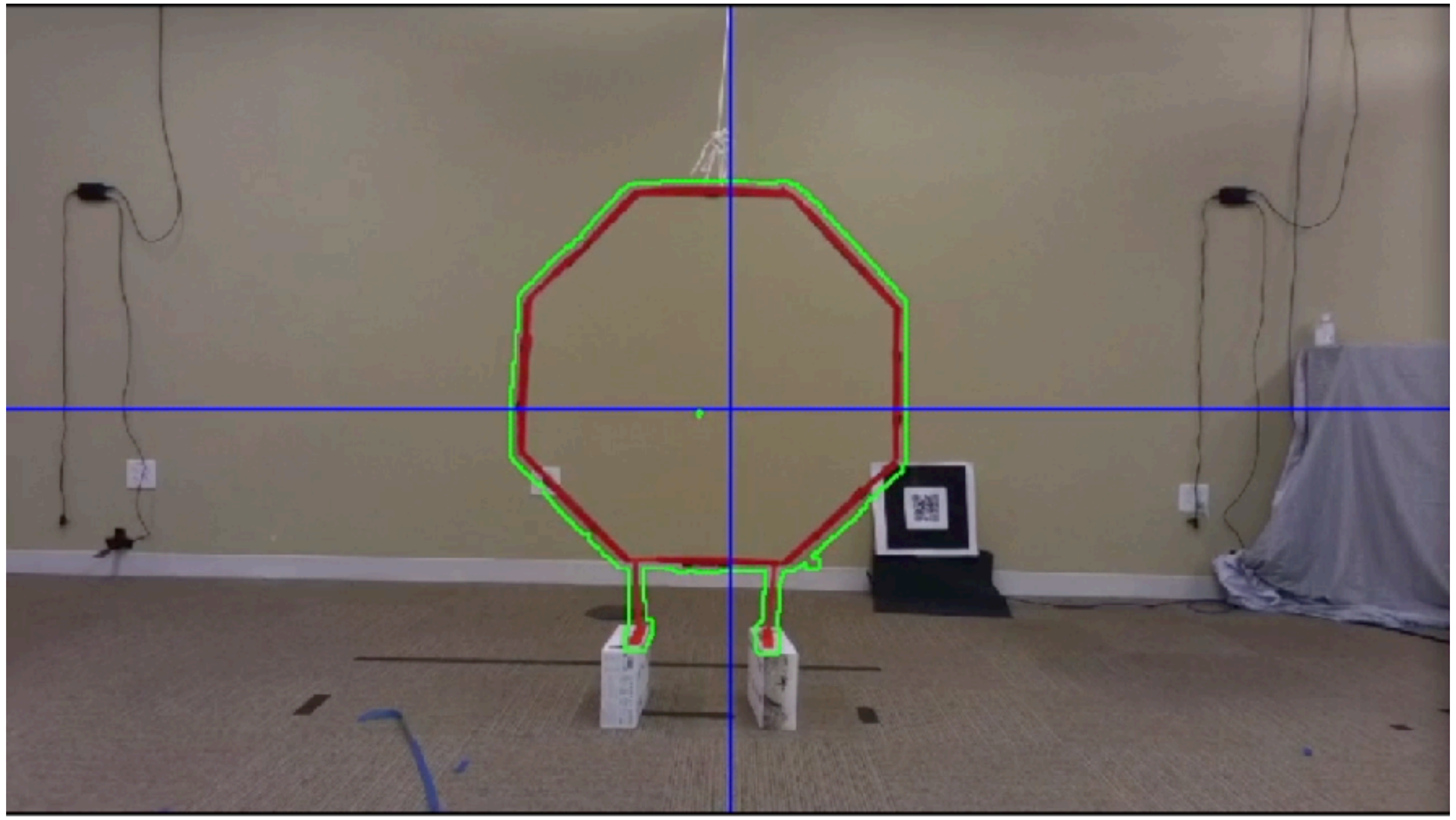


Image Processing Techniques

- Thresholding
- Color Filtering
- Blurring
- Smoothing
- Background subtraction
- Edge Detection
- Corner Detection
- Feature Matching
- Haar Cascade Object Detection
- ...

OpenCV

NumPy

scikit-image
image processing in python

SciPy

Image Processing Techniques

- **Thresholding**
- **Color Filtering**
- **Blurring**
- **Smoothing**
- **Background subtraction**
- **Edge Detection**
- **Corner Detection**
- **Feature Matching**
- **Haar Cascade Object Detection**
- **…**

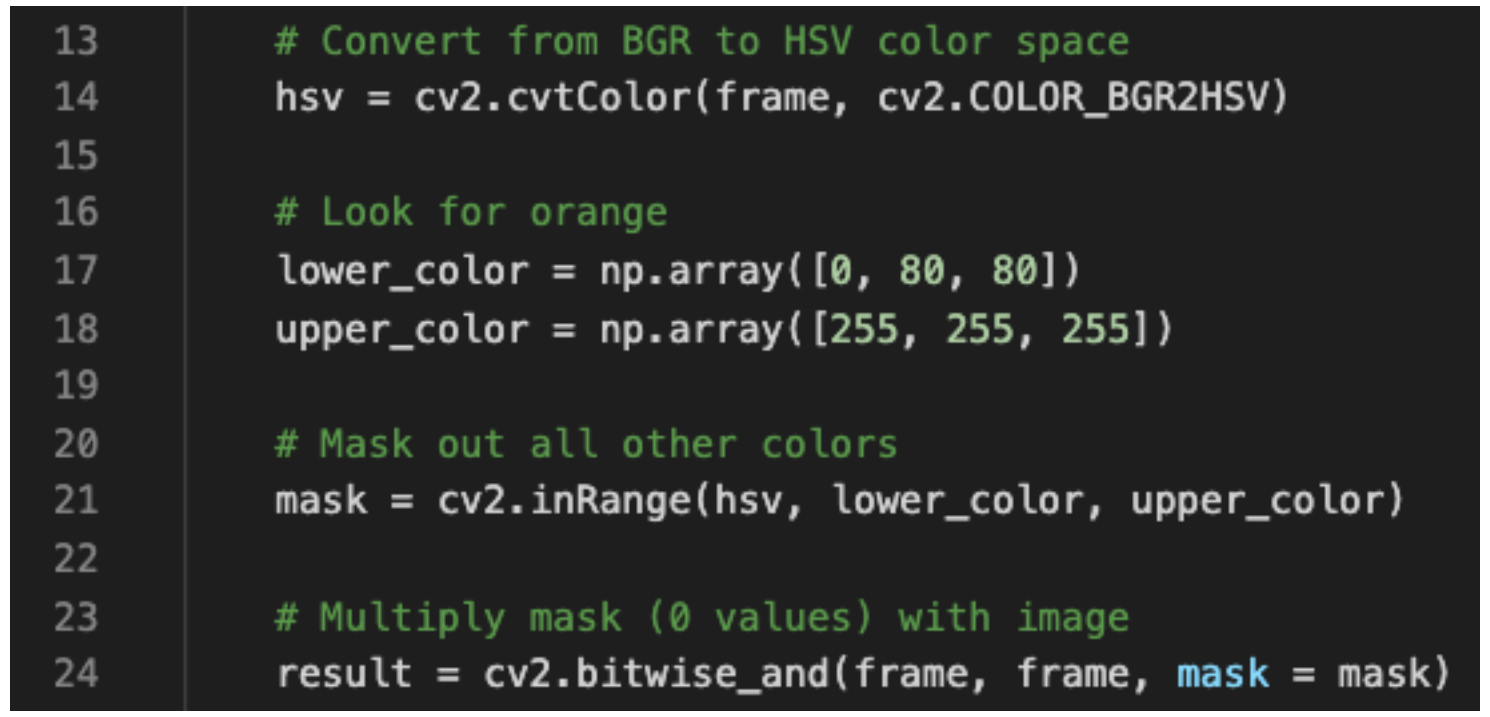
Color Filtering

Idea
Remove a range of colors from an image

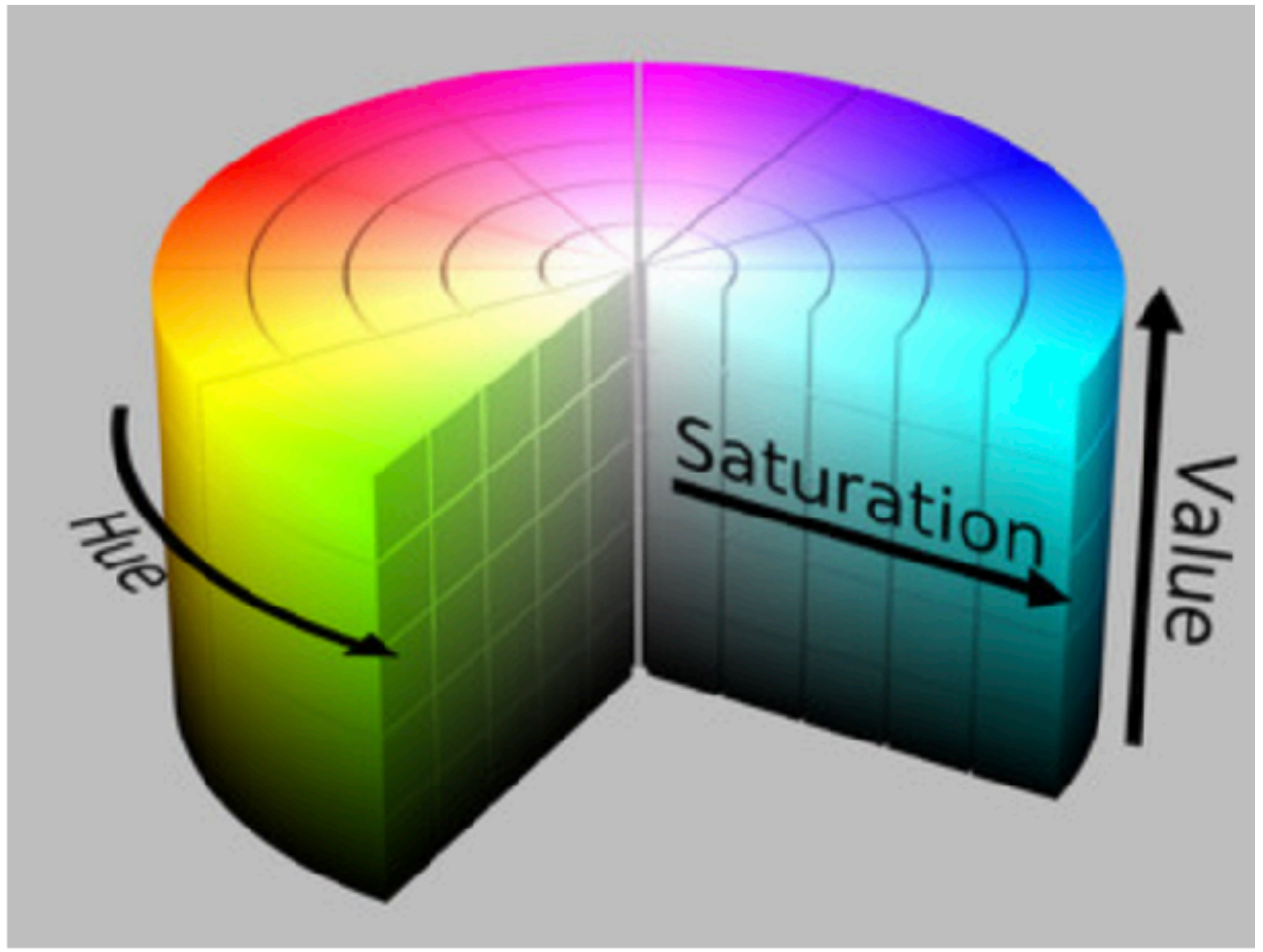
Technical Implementation
Convert image into a format that makes selecting colors easy
Look at each pixel, if it is not in your selected range remove it

HSV Image Format
HSV stands for **H**ue, **S**aturation, **V**alue, and is a cylindrical color space.
Hue: Are colors rotating around a central vertical axis
Saturation: Defines the shade of the color from least saturated to most
Value: Defines brightness from darkest to brightest

Code

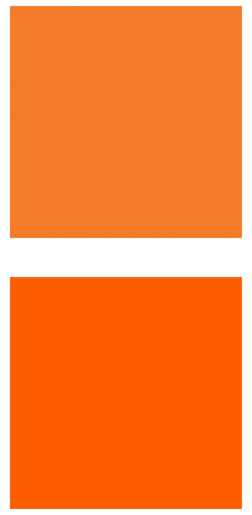


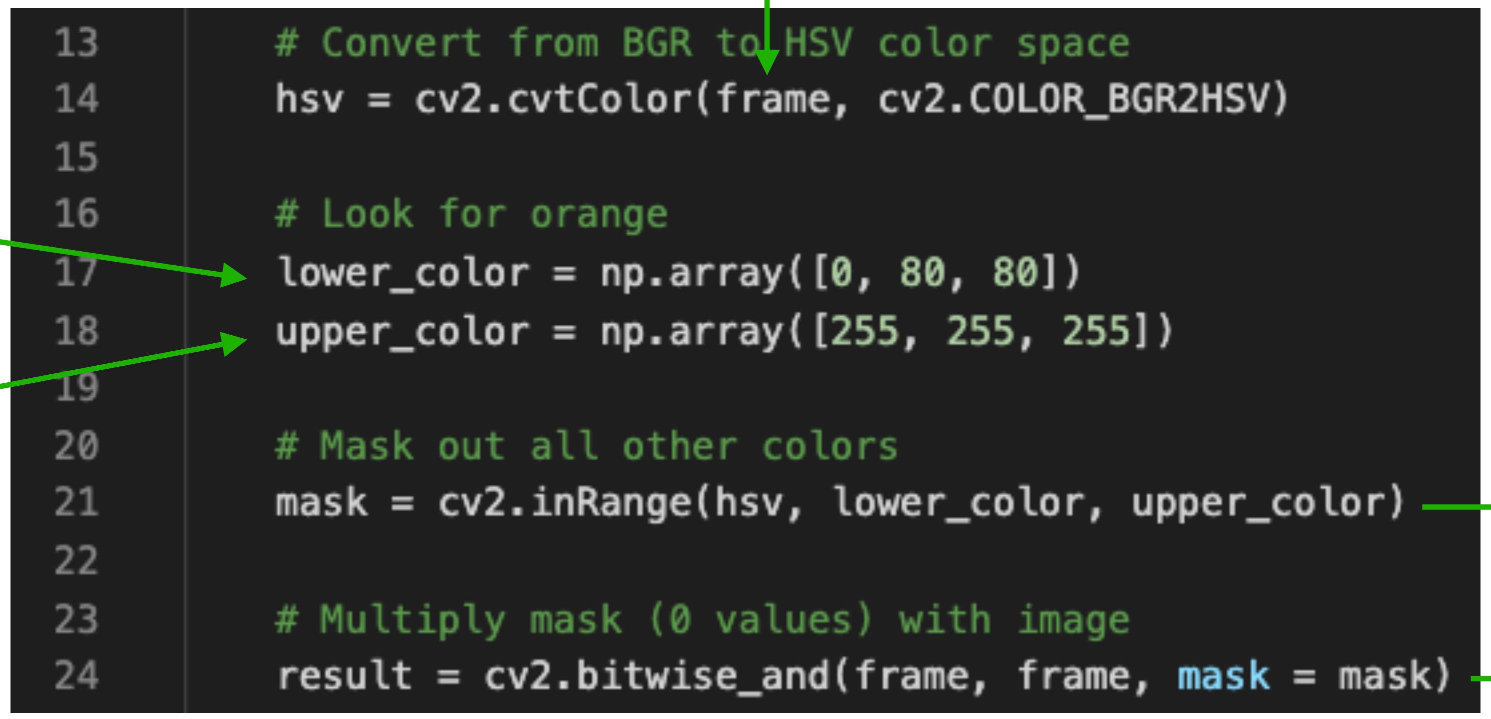
```
# Convert from BGR to HSV color space
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

# Look for orange
lower_color = np.array([0, 80, 80])
upper_color = np.array([255, 255, 255])

# Mask out all other colors
mask = cv2.inRange(hsv, lower_color, upper_color)

# Multiply mask (0 values) with image
result = cv2.bitwise_and(frame, frame, mask = mask)
```



By SharkDderivative work: SharkD [CC BY-SA 3.0 or GFDL], via Wikimedia Commons

Example: Color Filtering





```
    # Convert from BGR to HSV color space
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # Look for orange
    lower_color = np.array([0, 80, 80])
    upper_color = np.array([255, 255, 255])

    # Mask out all other colors
    mask = cv2.inRange(hsv, lower_color, upper_color)

    # Multiply mask (0 values) with image
    result = cv2.bitwise_and(frame, frame, mask = mask)
```

Question

Raw Data

Mask

Output

What are some limitations of this approach?

Image Processing Techniques

Basic Image Operations

- **Thresholding**
- **Color Filtering**
- **Blurring**
- **Smoothing**
- **Background subtraction**
- **Edge Detection**
- **Corner Detection**
- **Feature Matching**
- **Haar Cascade Object Detection**
- **…**

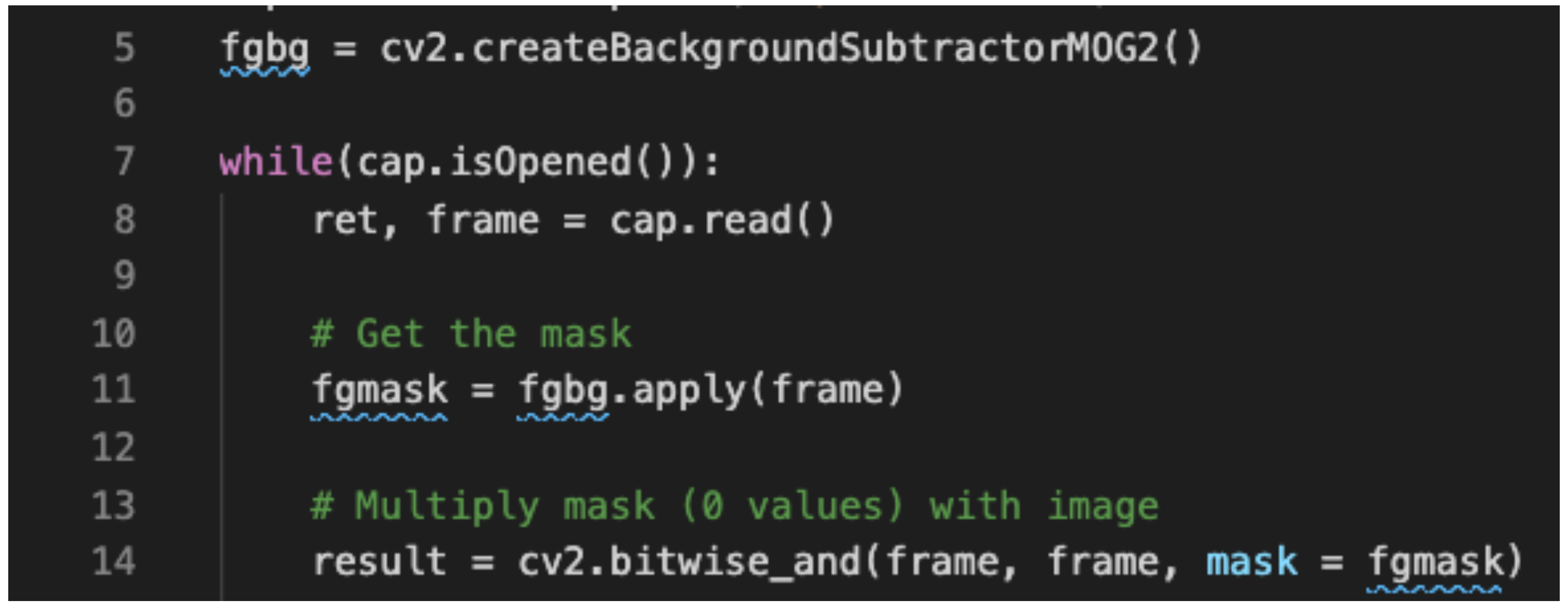
Background Subtraction

Idea
Remove background from current image

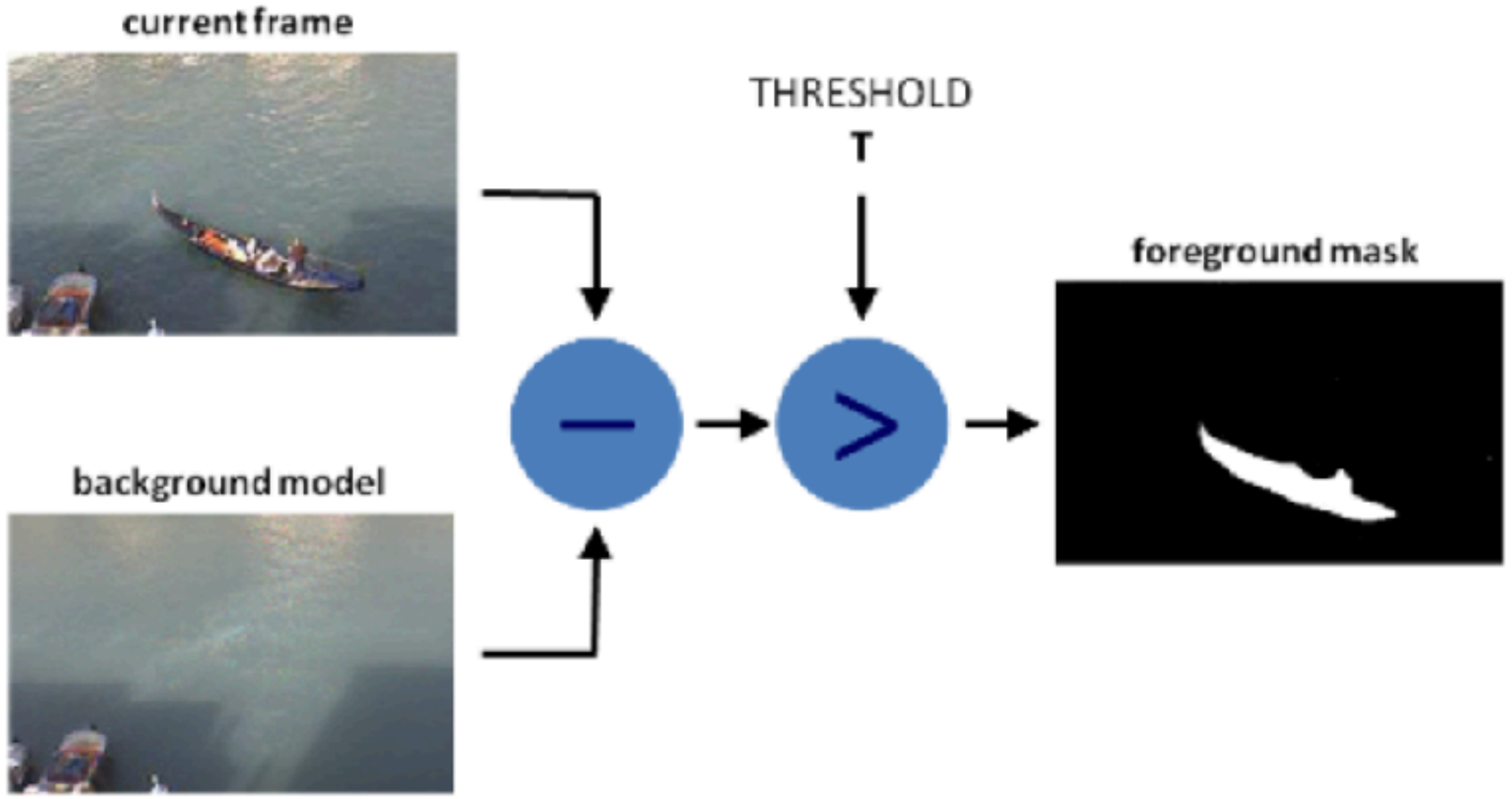
Technical Implementation
1) Estimate background for time t
2) Subtract estimated background from current frame
3) Apply threshold to absolute difference

Background Model
This technique requires a background model that contains the static part of the scene. Best suited for a static camera.

Code



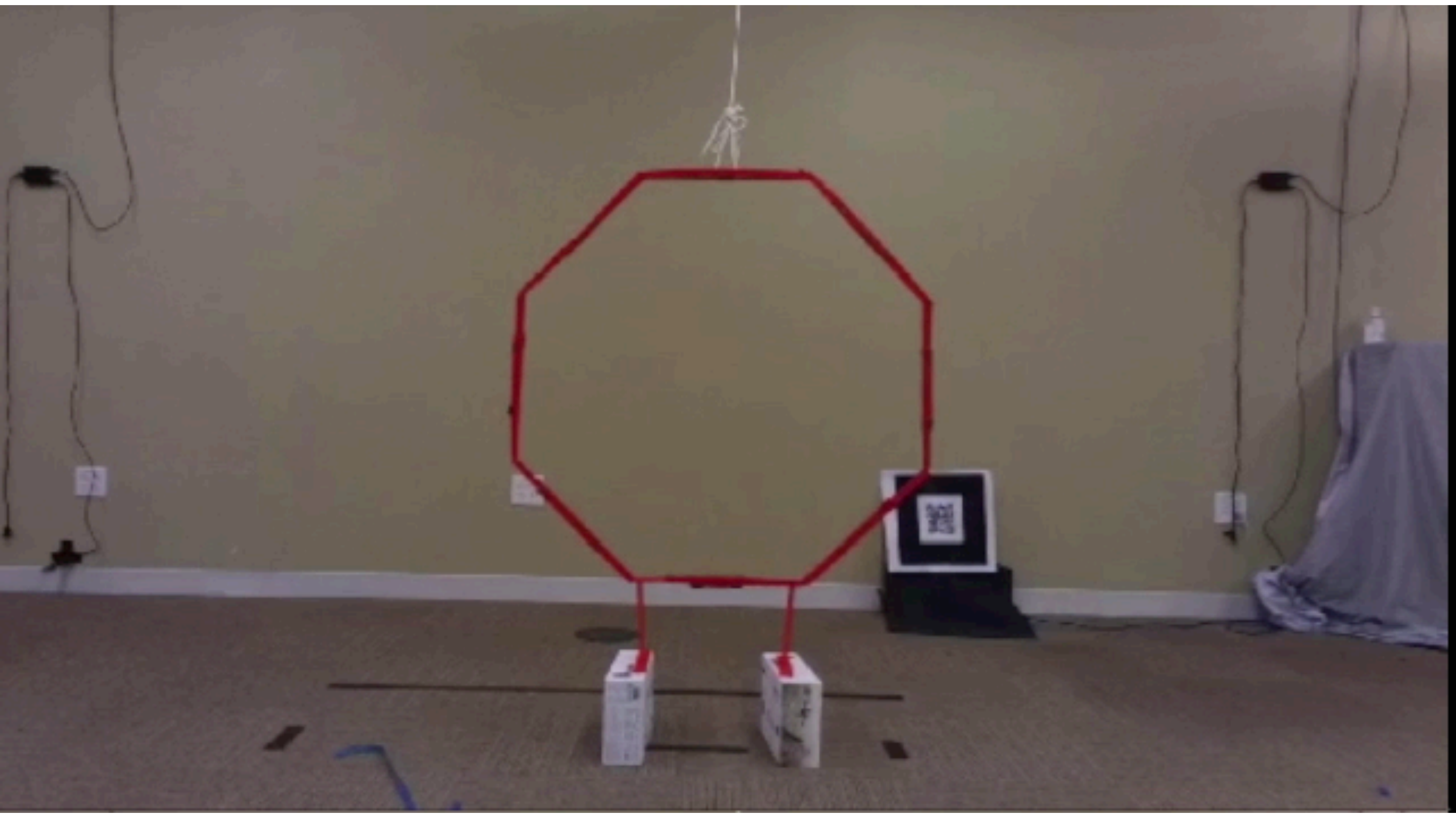
```
fgbg = cv2.createBackgroundSubtractorMOG2()

while(cap.isOpened()):
    ret, frame = cap.read()

    # Get the mask
    fgmask = fgbg.apply(frame)

    # Multiply mask (0 values) with image
    result = cv2.bitwise_and(frame, frame, mask = fgmask)
```



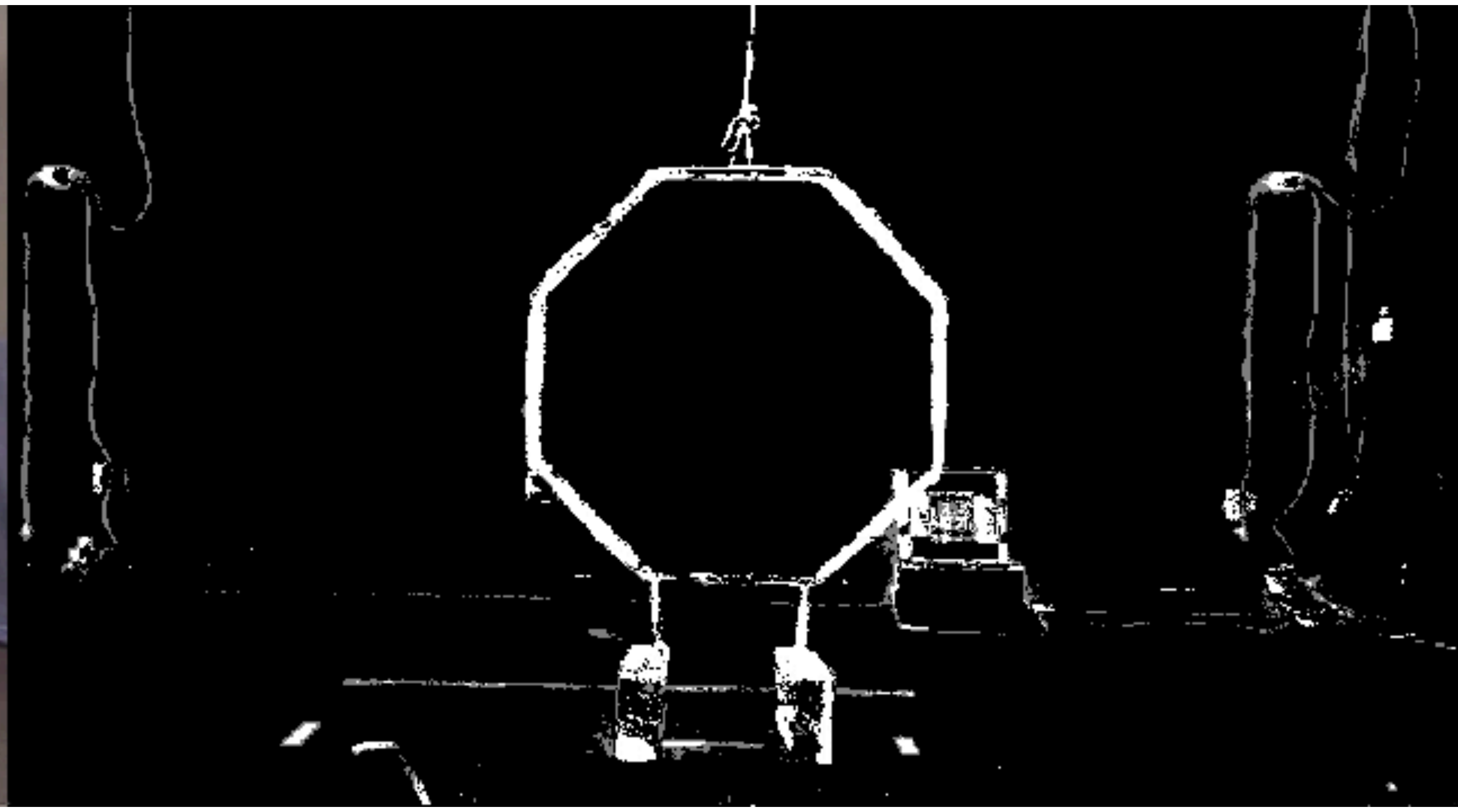
OpenCV Docs: https://docs.opencv.org/3.4/d1/dc5/tutorial_background_subtraction.html

Example: Background Subtraction

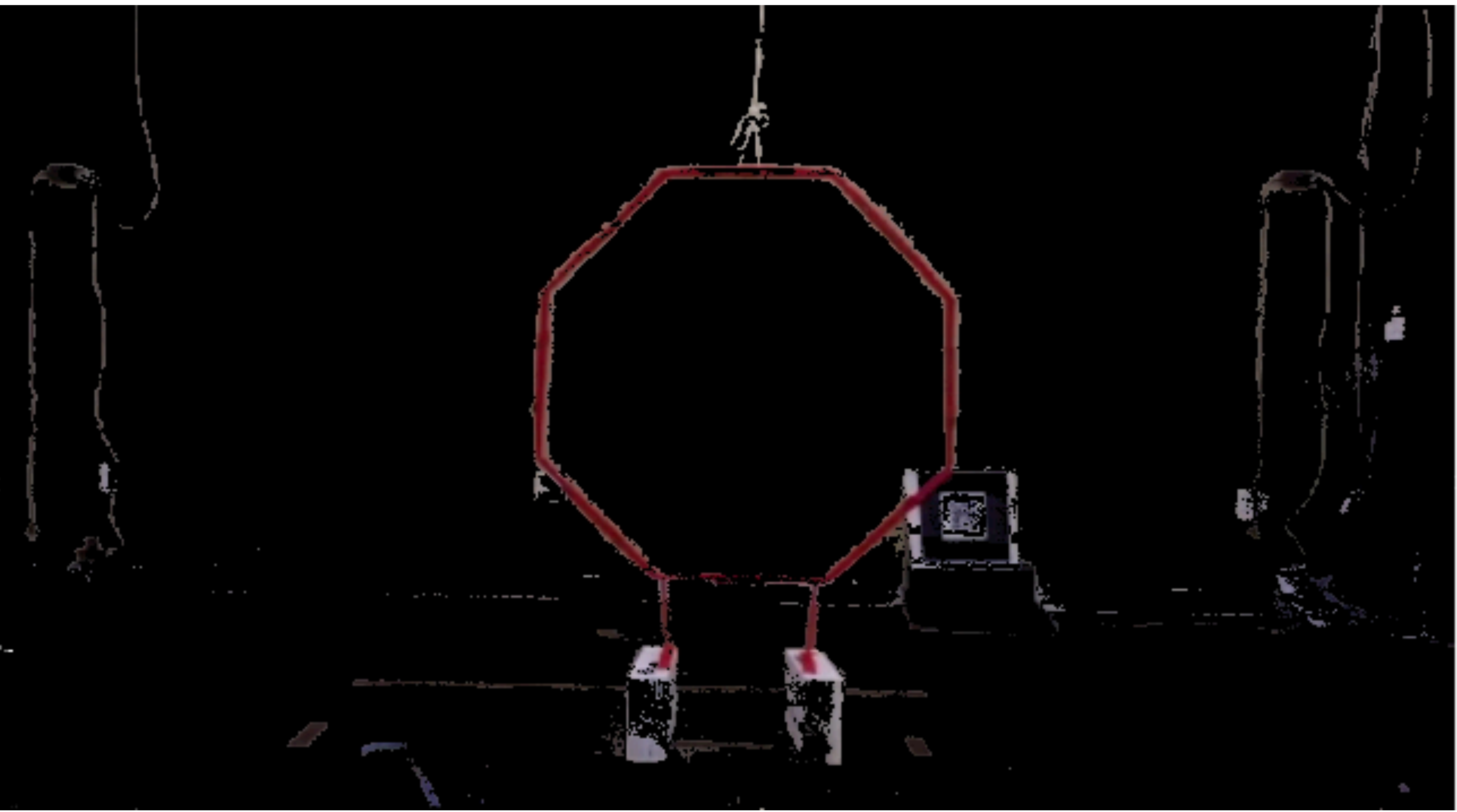
Raw Data

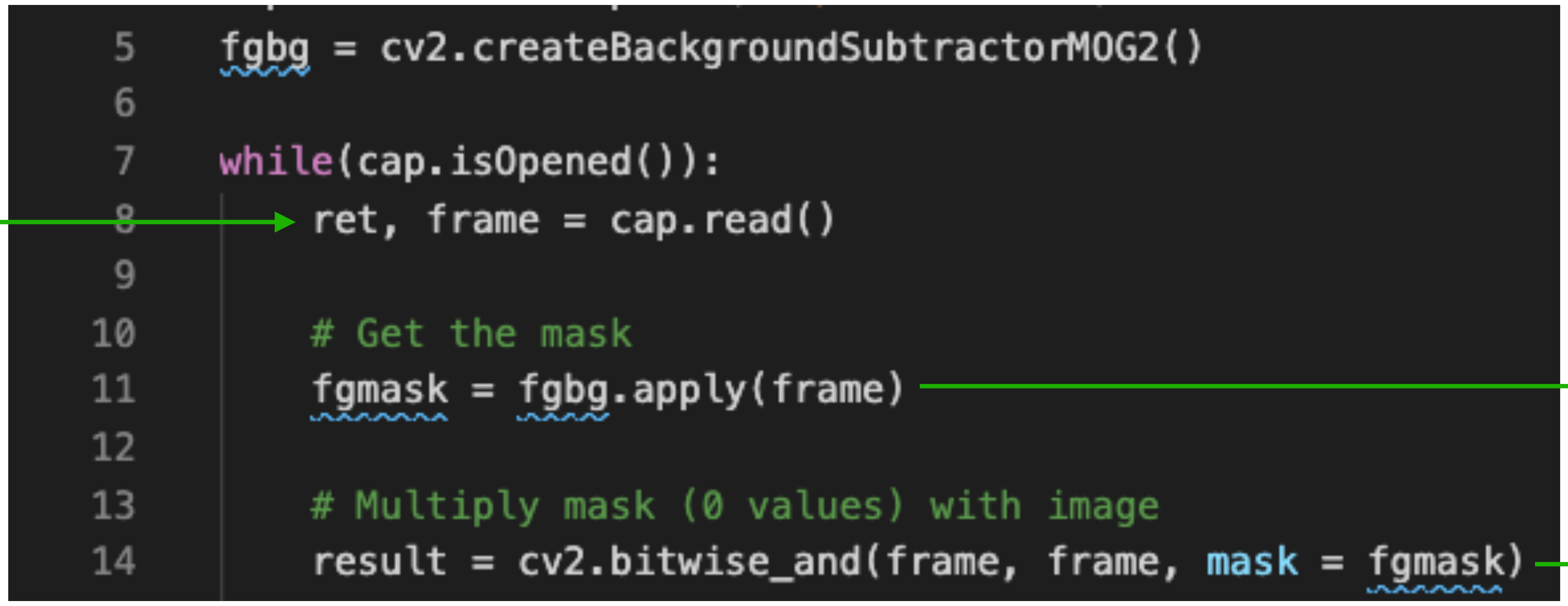


Mask



Output

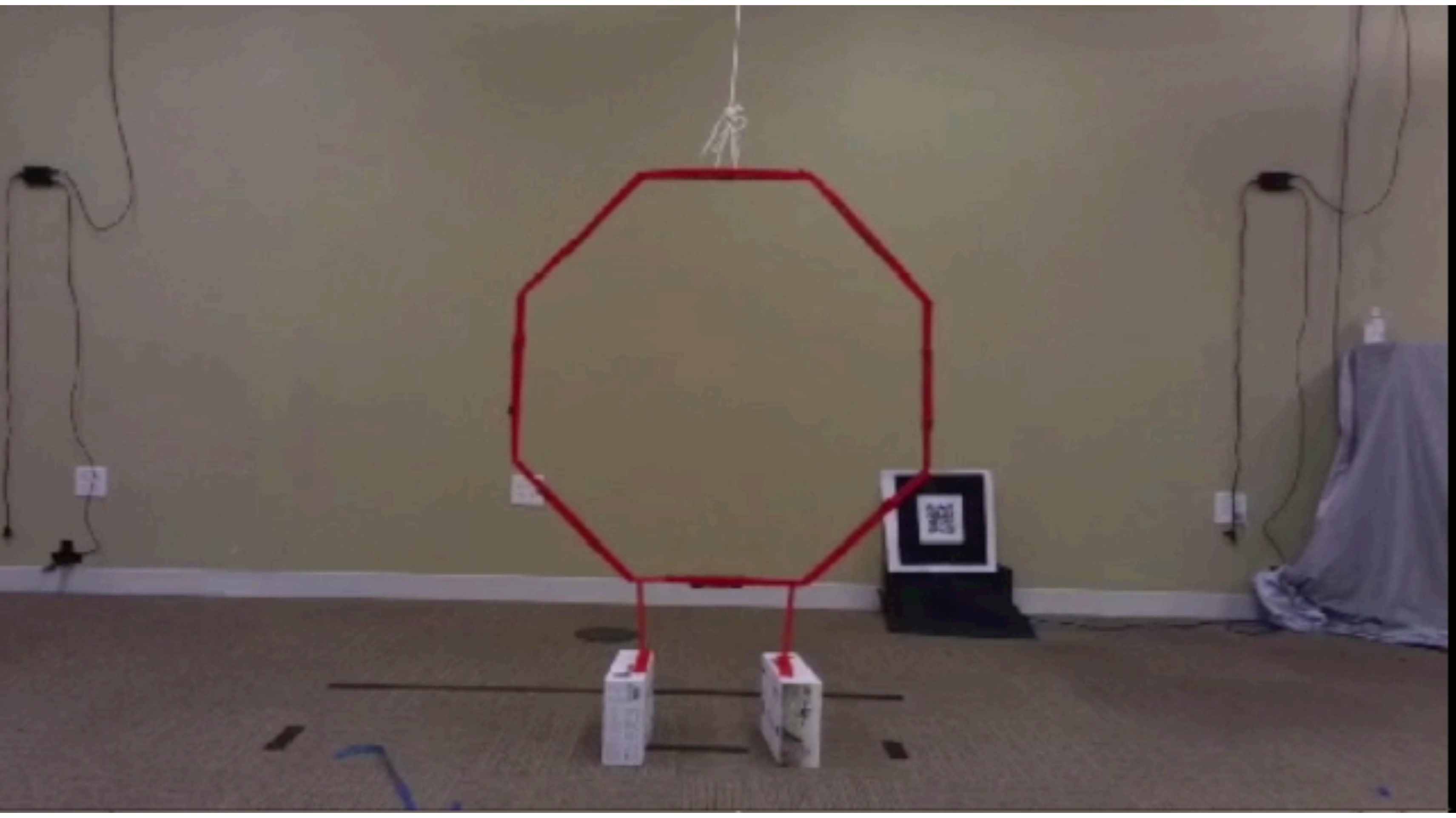




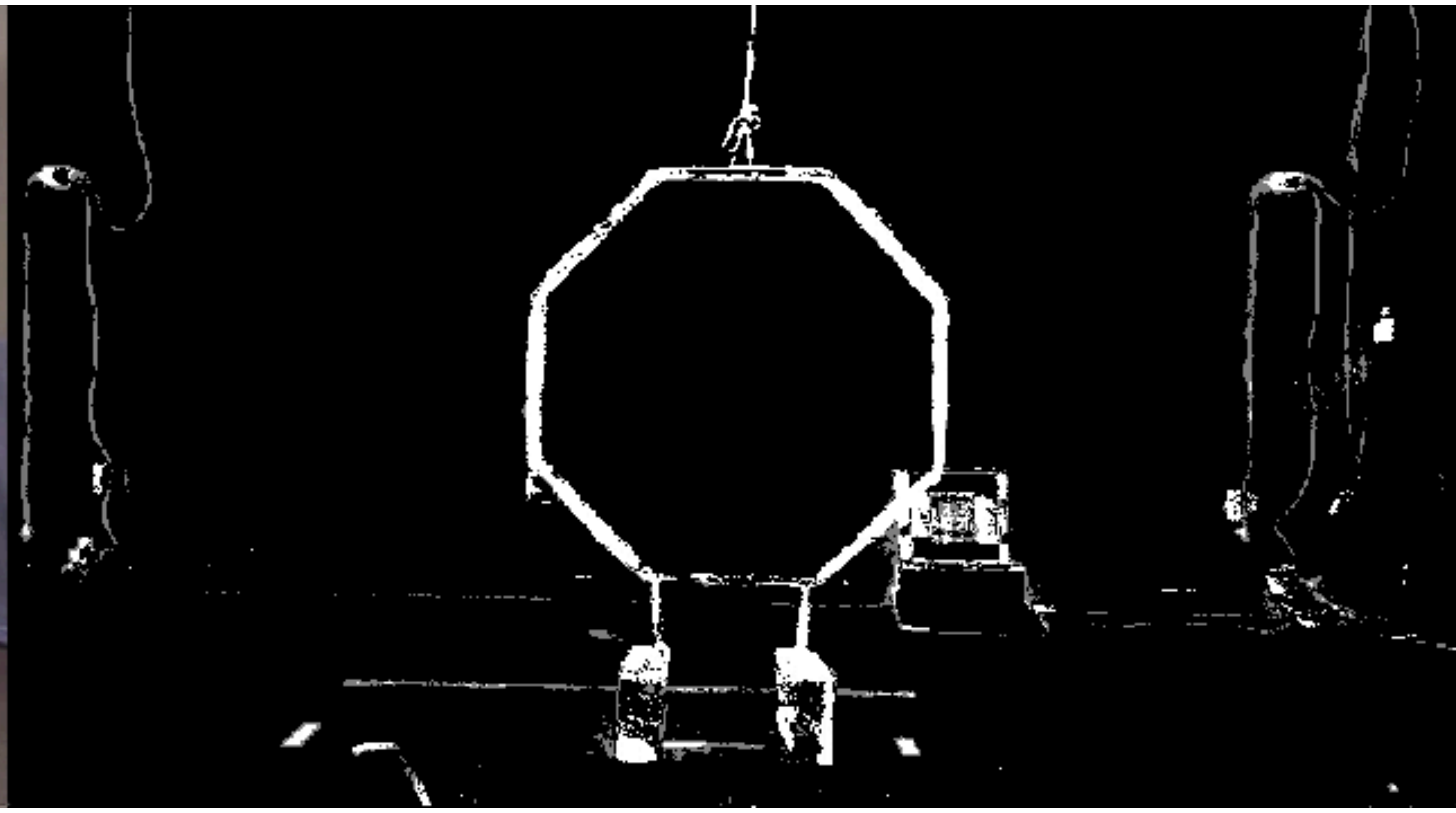
```
fgbg = cv2.createBackgroundSubtractorMOG2()

while(cap.isOpened()):
    ret, frame = cap.read()

    # Get the mask
    fgmask = fgbg.apply(frame)

    # Multiply mask (0 values) with image
    result = cv2.bitwise_and(frame, frame, mask = fgmask)
```

Question

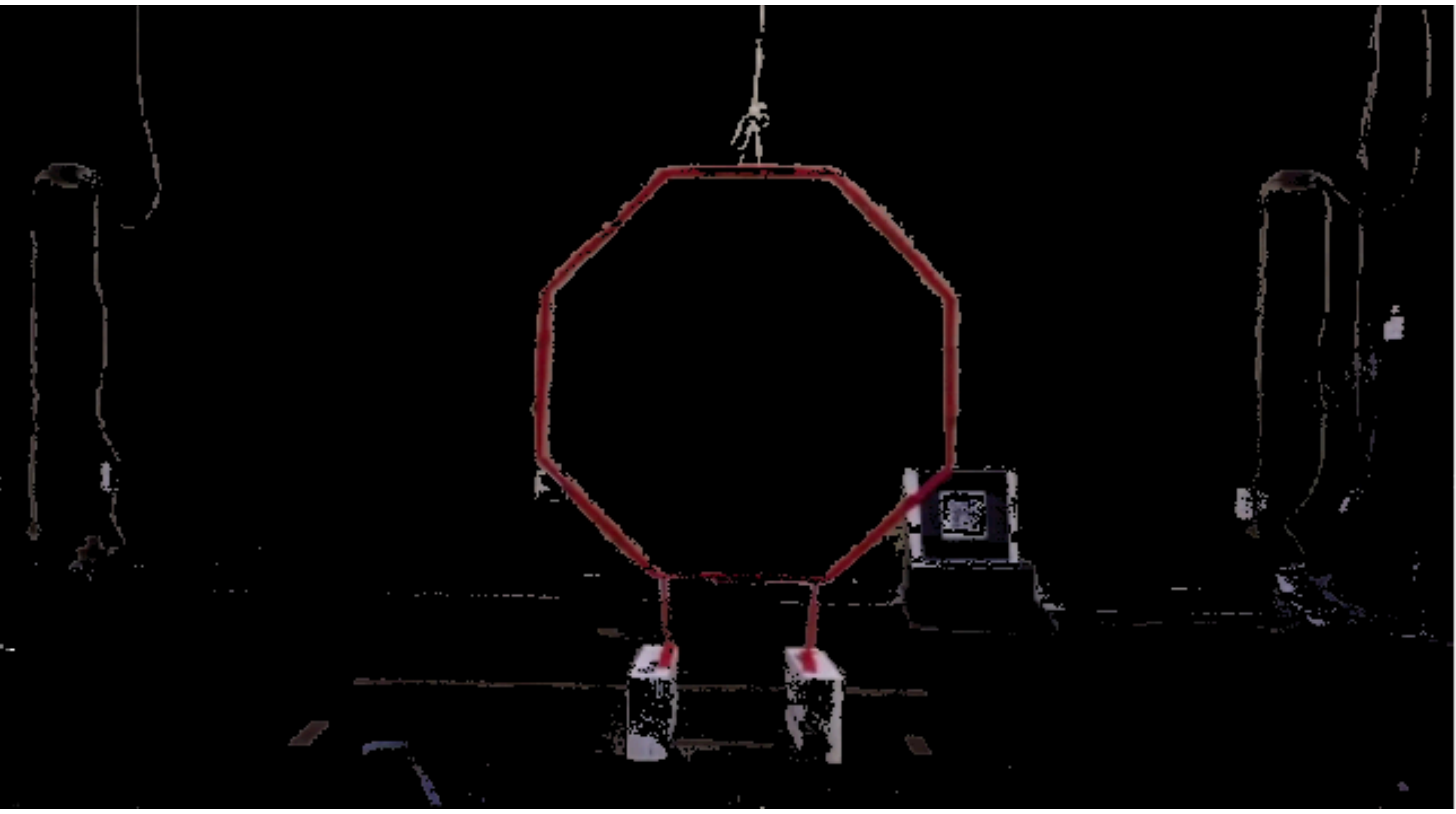
Raw Data



Mask



Output



What are some limitations of this approach?
(Other than requiring a more or less static camera)

Image Processing Techniques

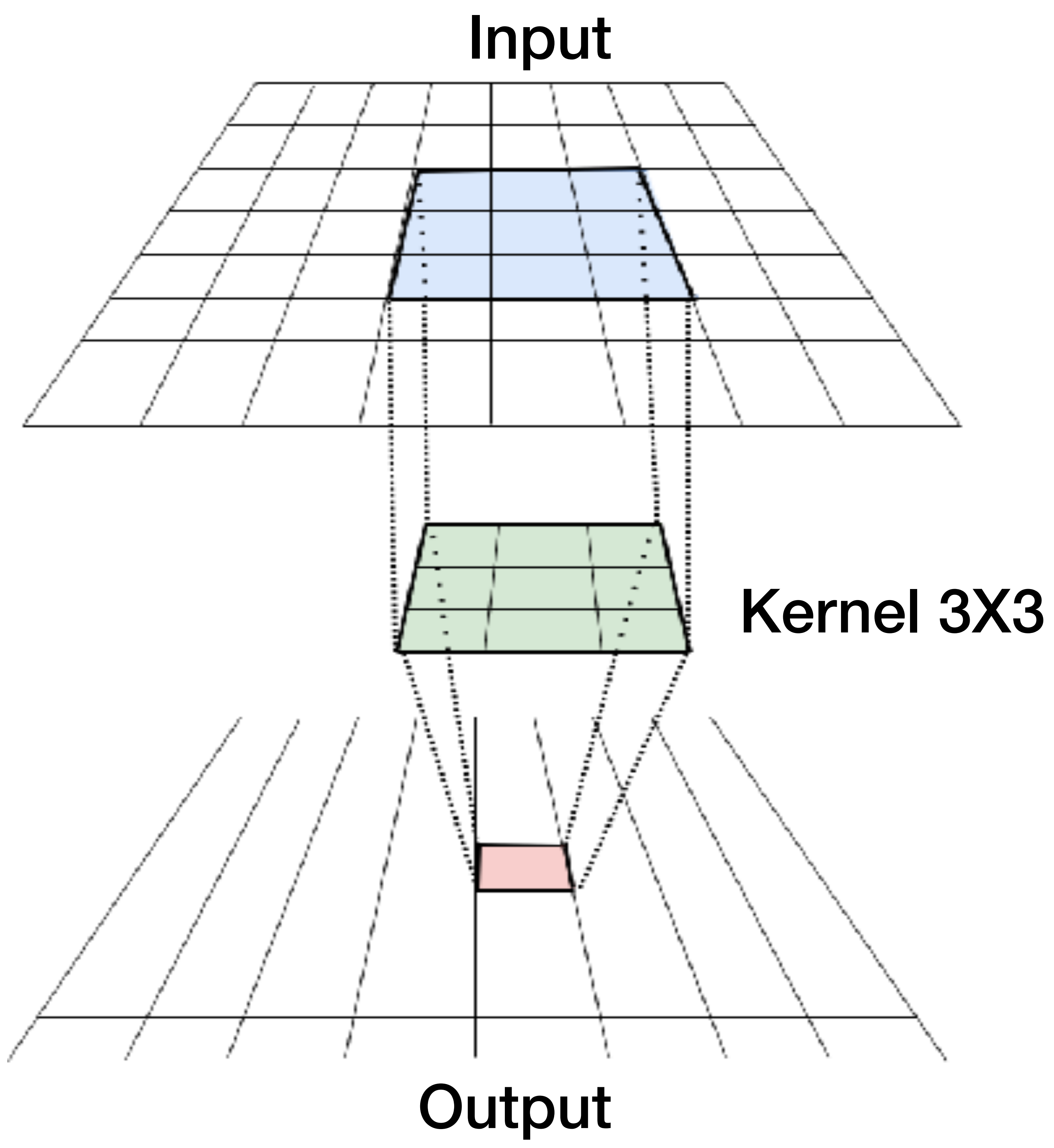
- **Thresholding**
- **Color Filtering**
- **Blurring**
- **Smoothing**
- **Background subtraction**
- **Edge Detection**
- **Corner Detection**
- **Feature Matching**
- **Haar Cascade Object Detection**
- **…**

Convolution

Definition: Convolution is the process of adding each element of the image to its local neighbors, weighted by the kernel

$$g(x,y) = \omega * f(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega(dx,dy) f(x+dx, y+dy),$$

Filtered Image **Filter Kernel** **Original Image**

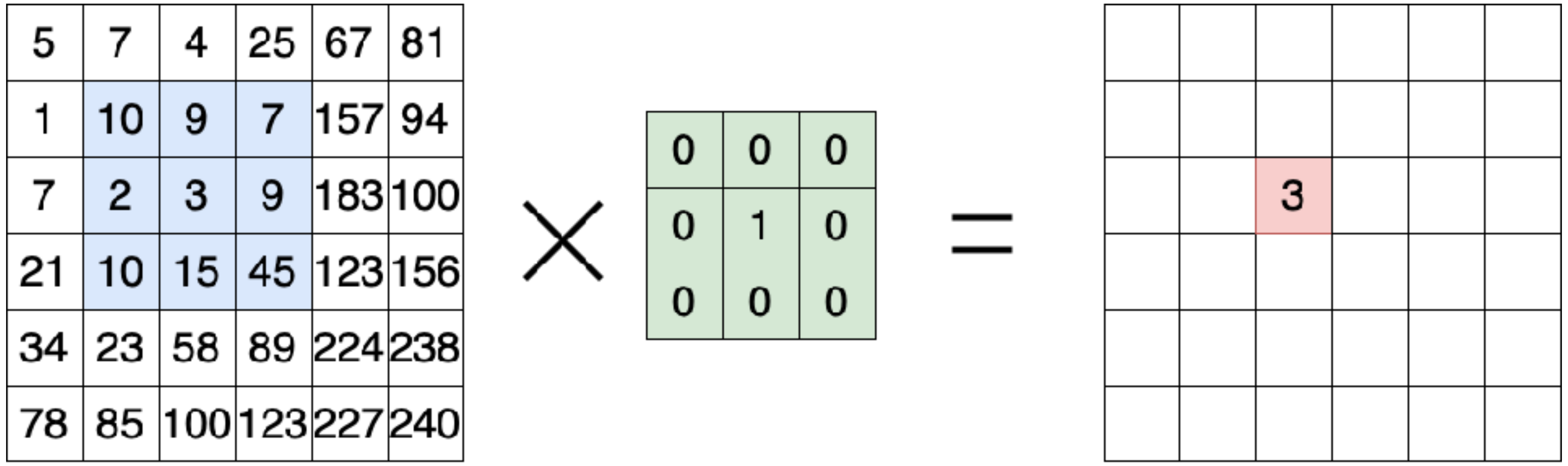


Convolution

Definition: Convolution is the process of adding each element of the image to its local neighbors, weighted by the kernel

$$g(x,y) = \omega * f(x,y) = \sum_{dx=-a}^{a}\sum_{dy=-b}^{b} \omega(dx,dy)\, f(x+dx, y+dy),$$

Filtered Image **Filter Kernel** **Original Image**



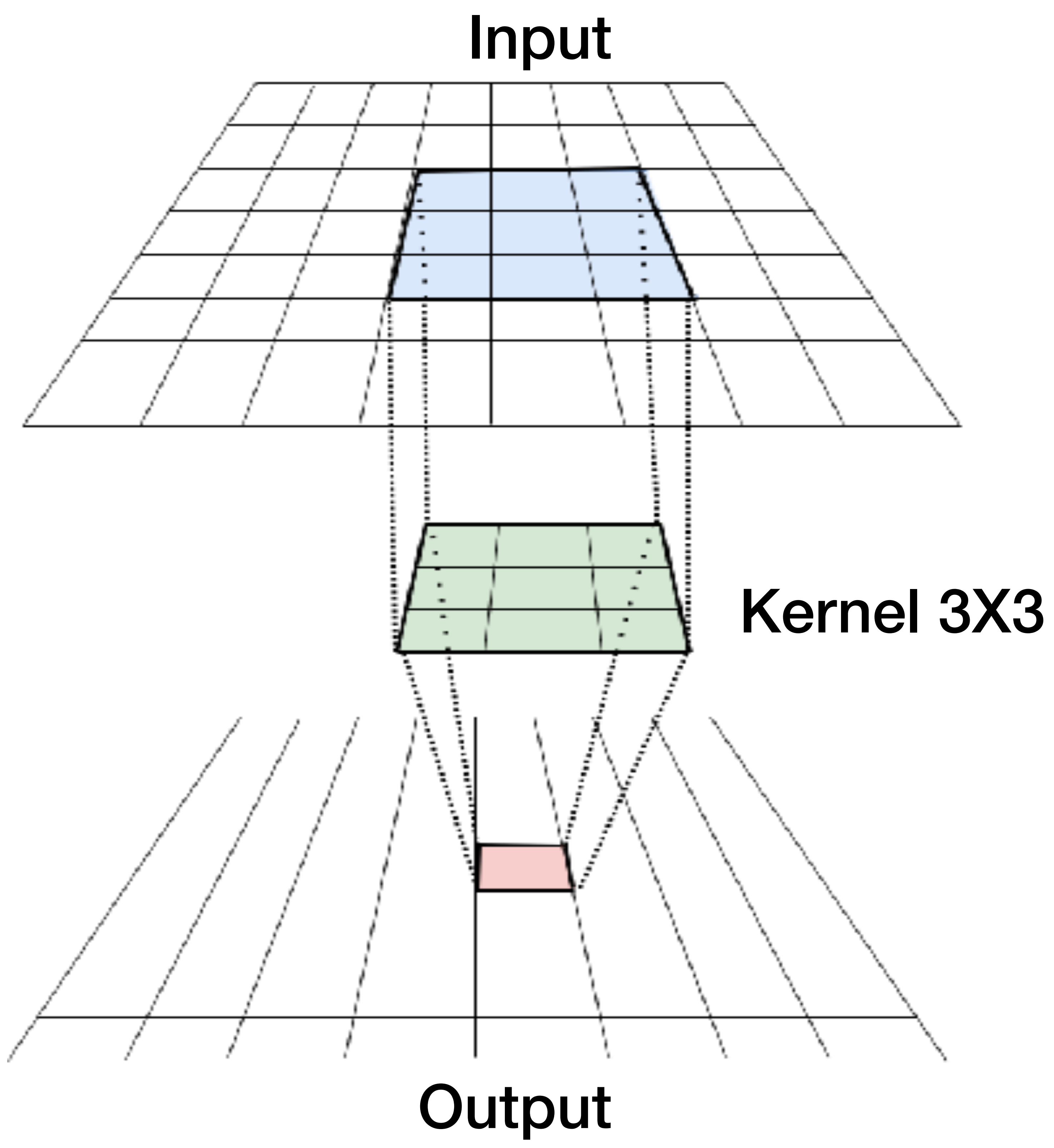
5	7	4	25	67	81
1	10	9	7	157	94
7	2	3	9	183	100
21	10	15	45	123	156
34	23	58	89	224	238
78	85	100	123	227	240

×

0	0	0
0	1	0
0	0	0

=

		3			

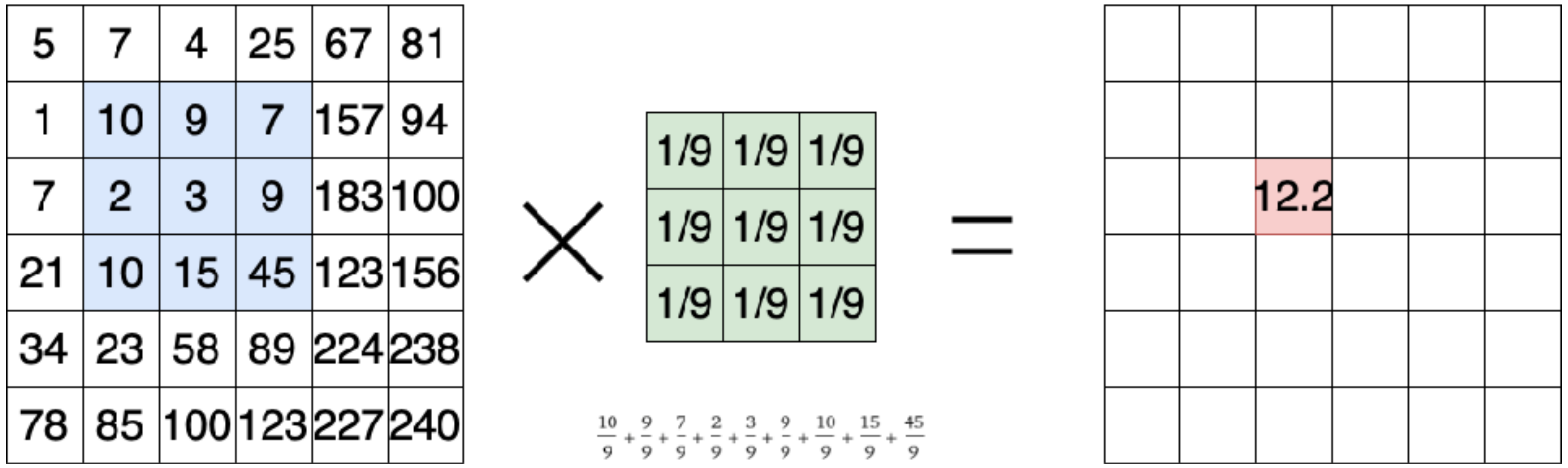


Convolution

Definition: Convolution is the process of adding each element of the image to its local neighbors, weighted by the kernel

$$g(x,y) = \omega * f(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega(dx,dy) f(x+dx, y+dy),$$

Filtered Image **Filter Kernel** **Original Image**



5	7	4	25	67	81
1	10	9	7	157	94
7	2	3	9	183	100
21	10	15	45	123	156
34	23	58	89	224	238
78	85	100	123	227	240

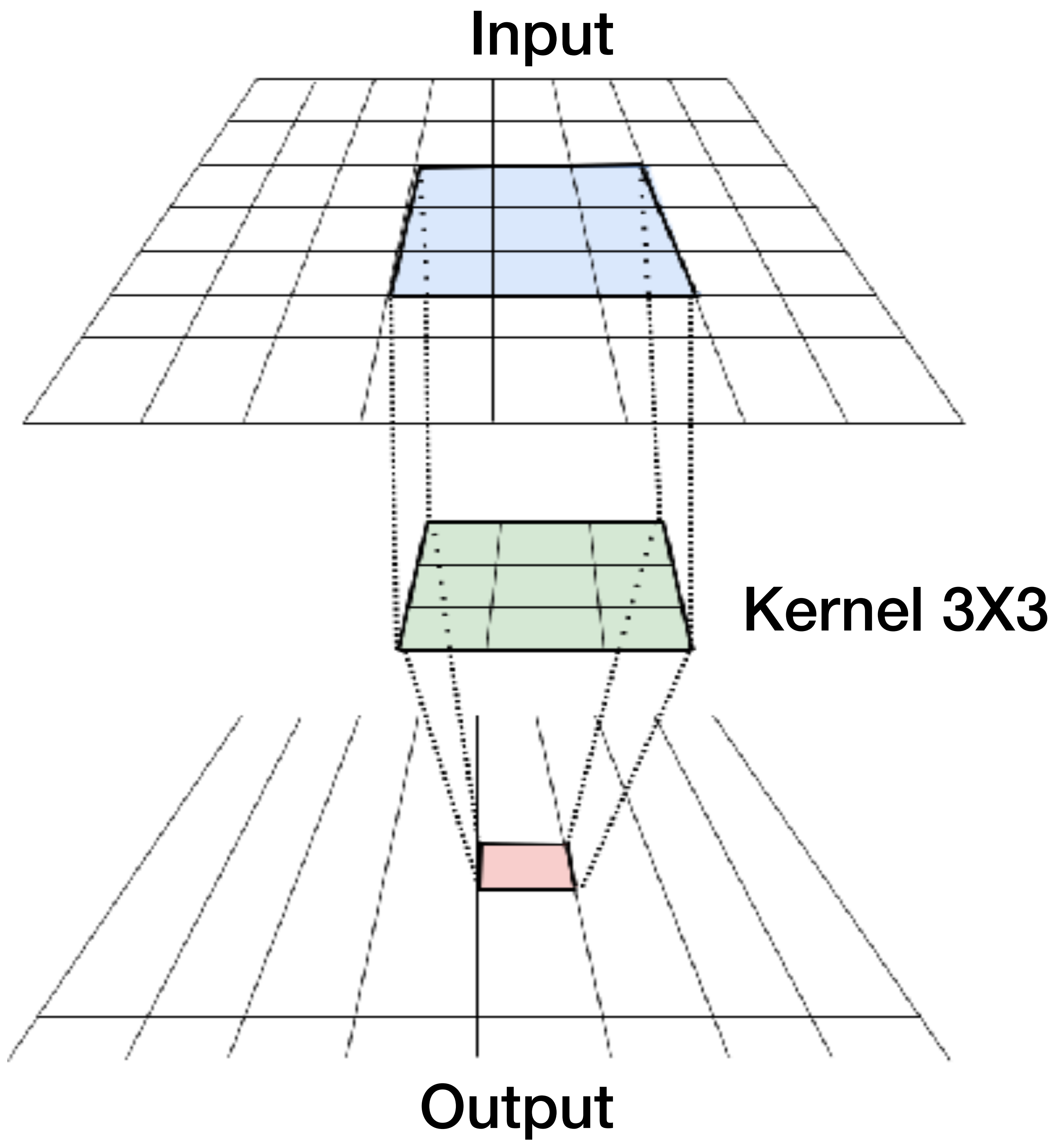
×

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

$$\frac{10}{9}+\frac{9}{9}+\frac{7}{9}+\frac{2}{9}+\frac{3}{9}+\frac{9}{9}+\frac{10}{9}+\frac{15}{9}+\frac{45}{9}$$

=

		12.2			



Blurring

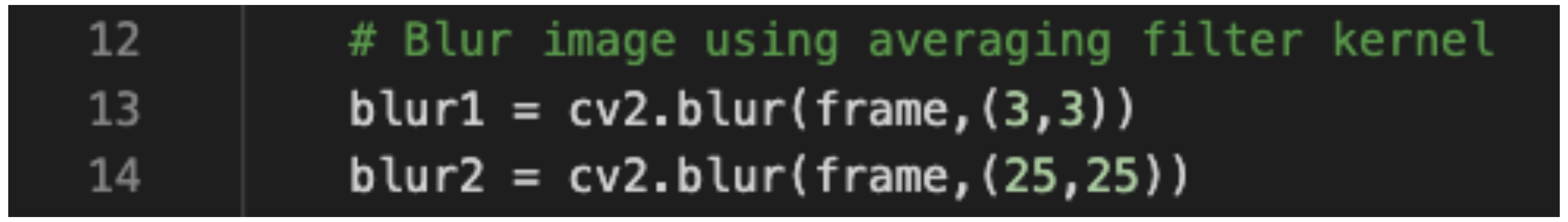
Idea
Remove high frequency content (e.g. noise, edges, etc)

Technical Implementation
Convolve image with a normalized box filter
i.e. take an average of all pixel under the kernel area
and replace the central element with this average.

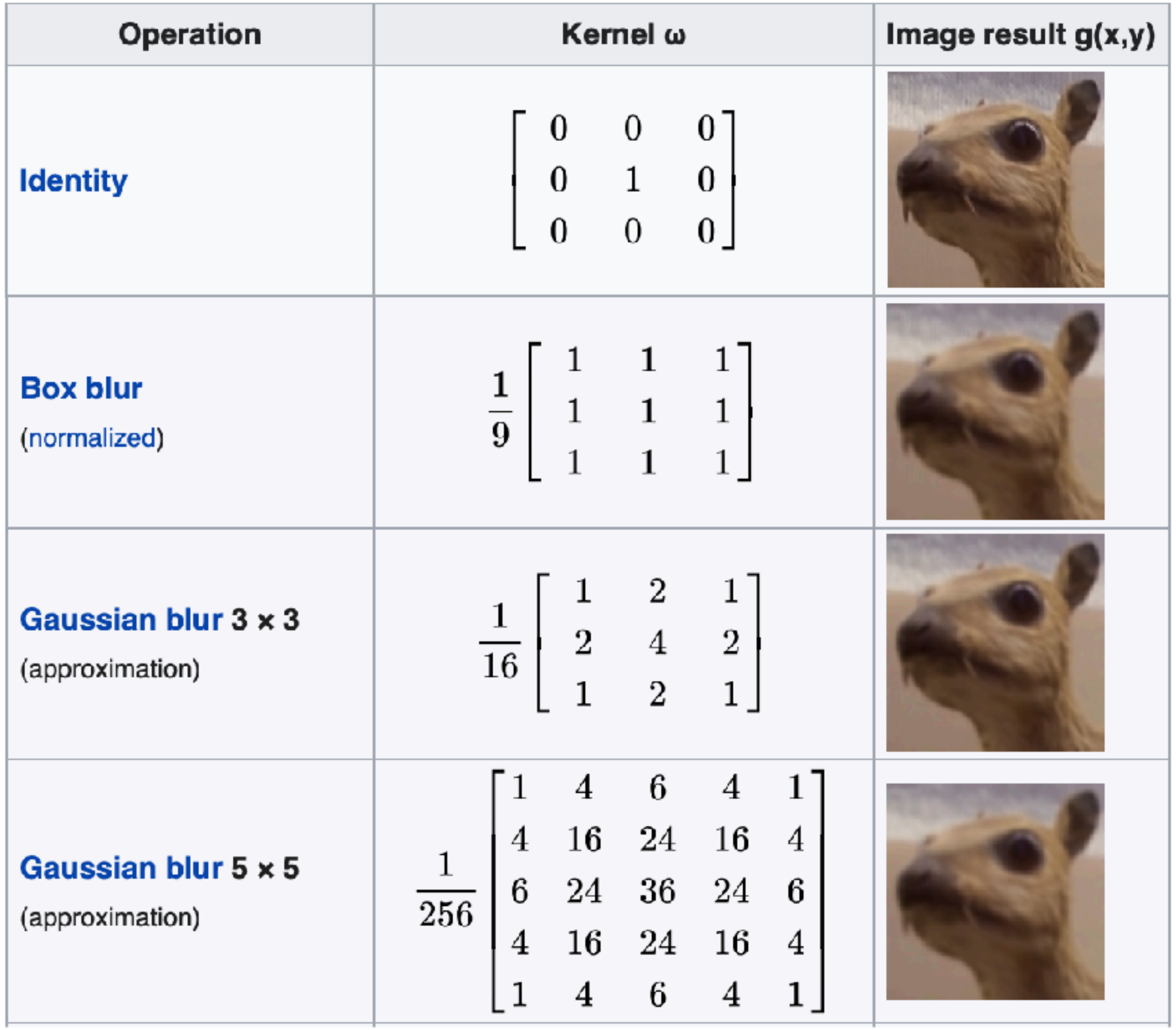
Kernel

$$k = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Code



```
# Blur image using averaging filter kernel
blur1 = cv2.blur(frame,(3,3))
blur2 = cv2.blur(frame,(25,25))
```

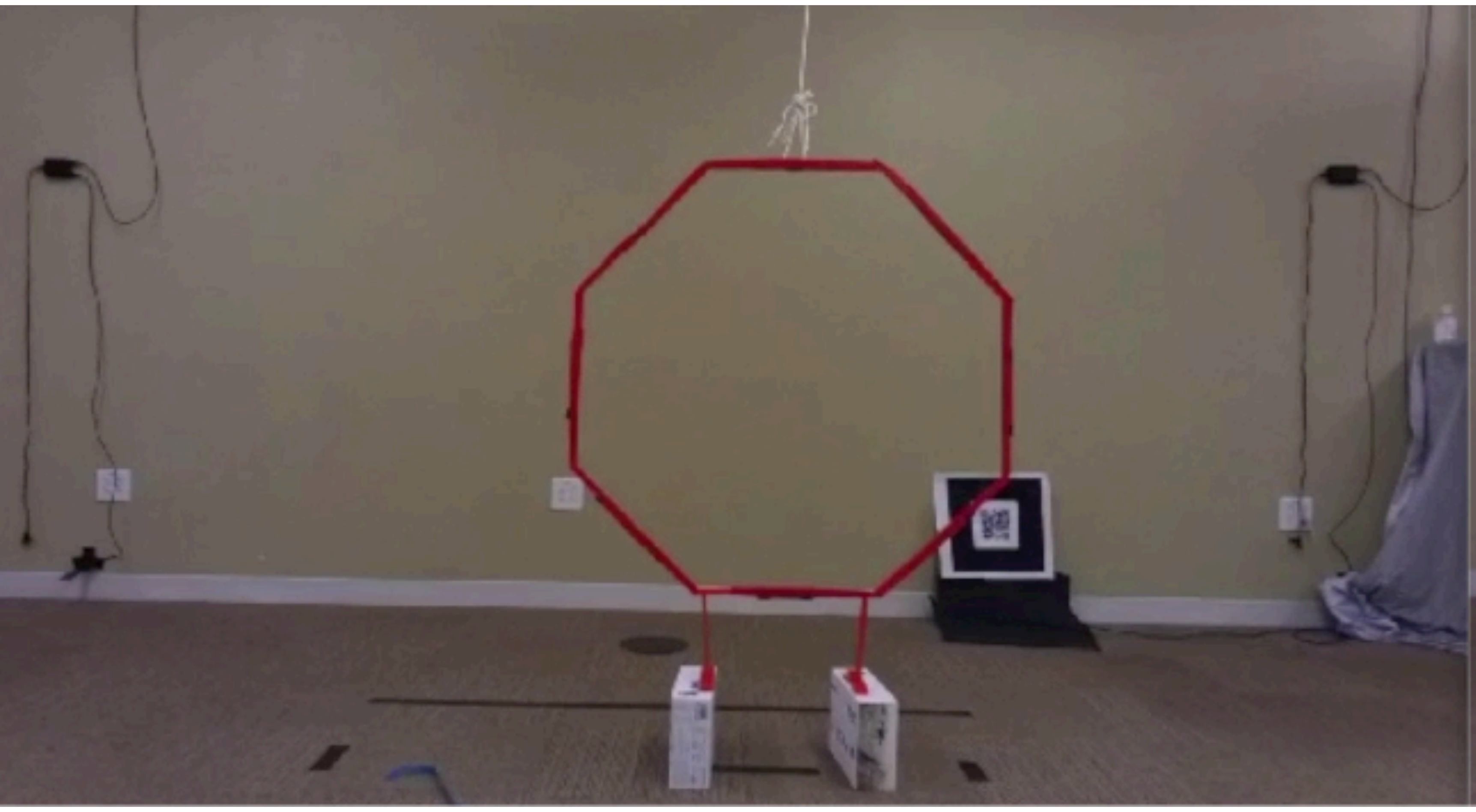


Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5 × 5 (approximation)	$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

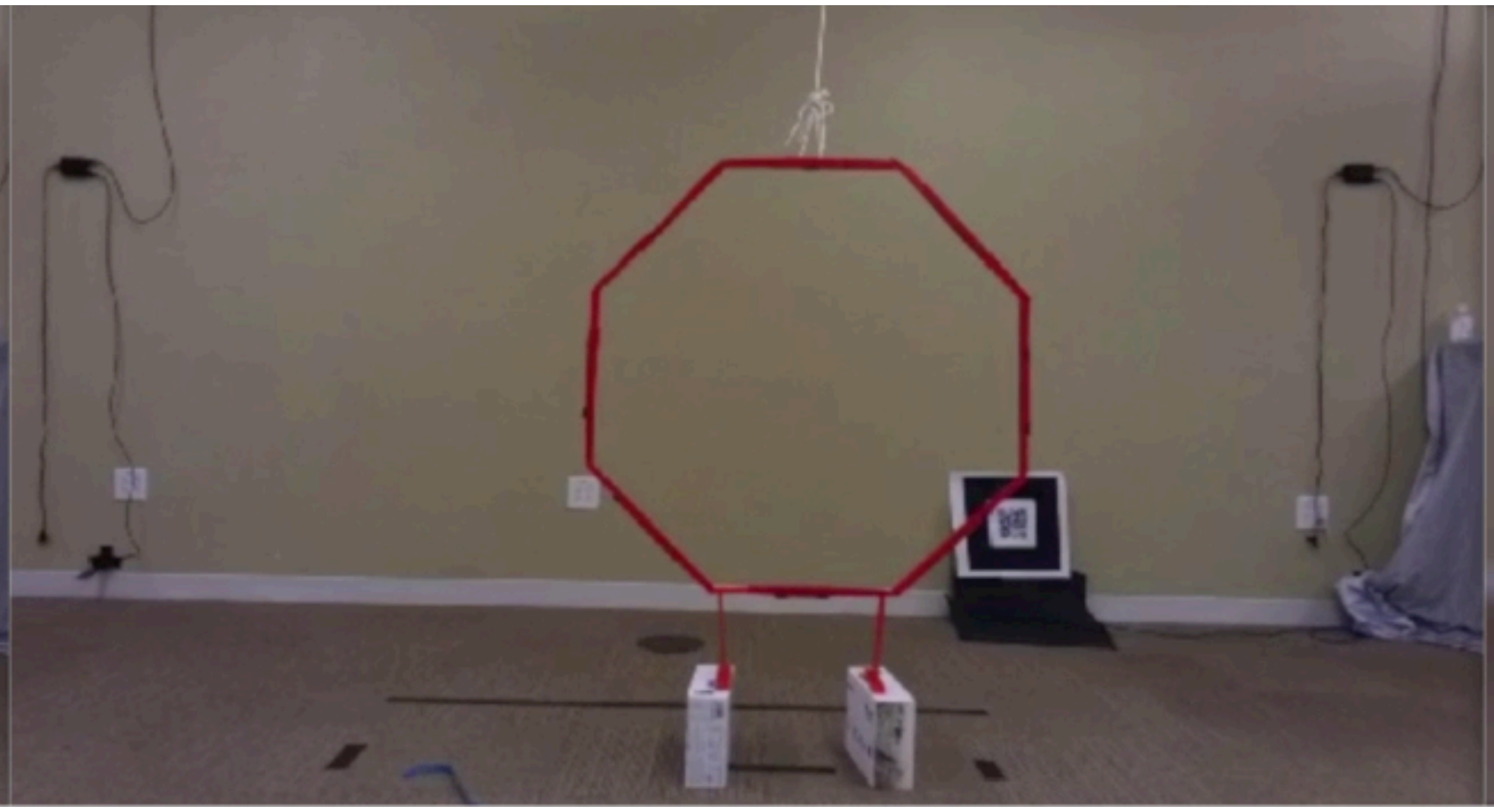
Kernel: https://en.wikipedia.org/wiki/Kernel_(image_processing)

Example: Blurring

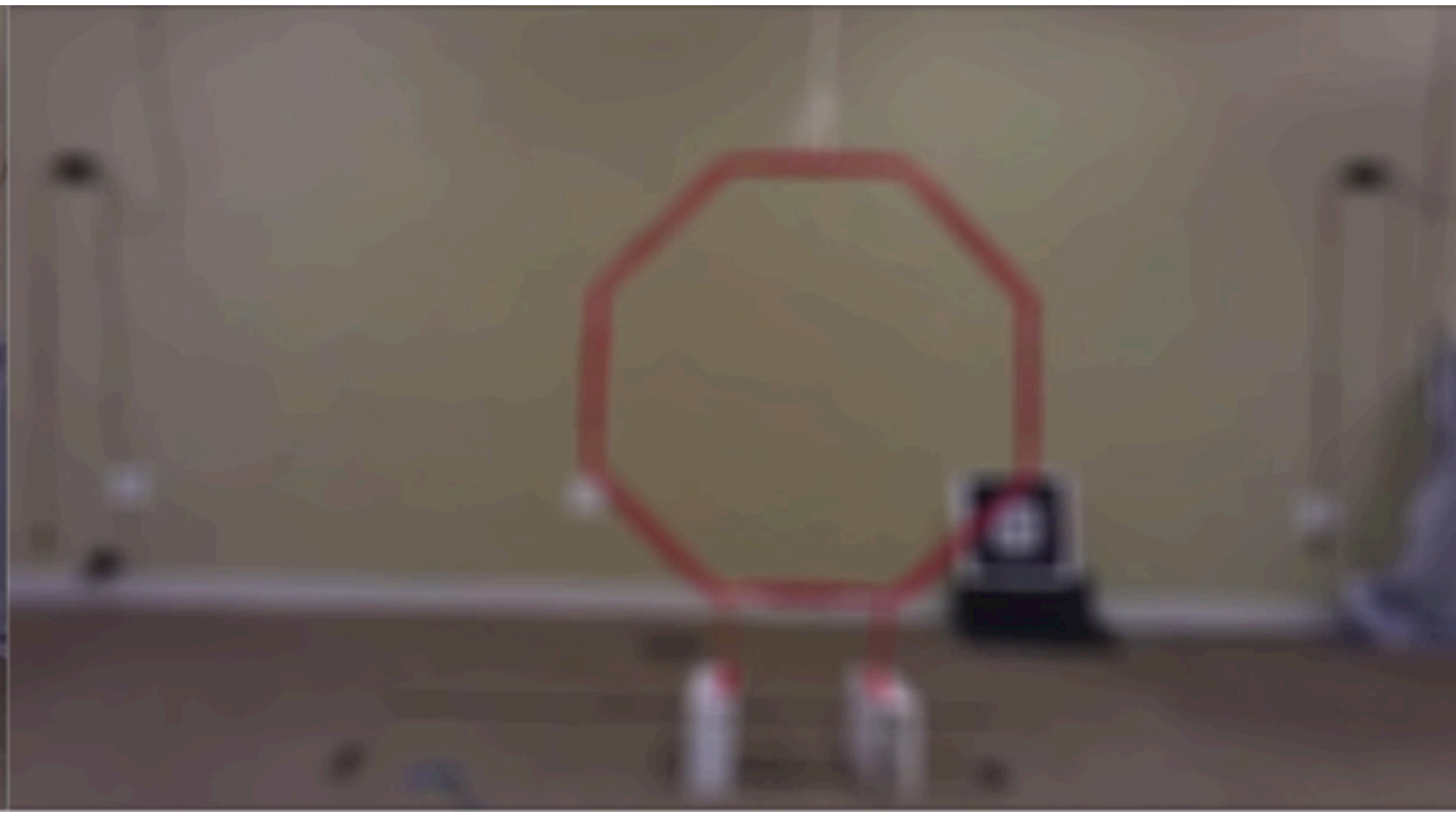
Raw Data

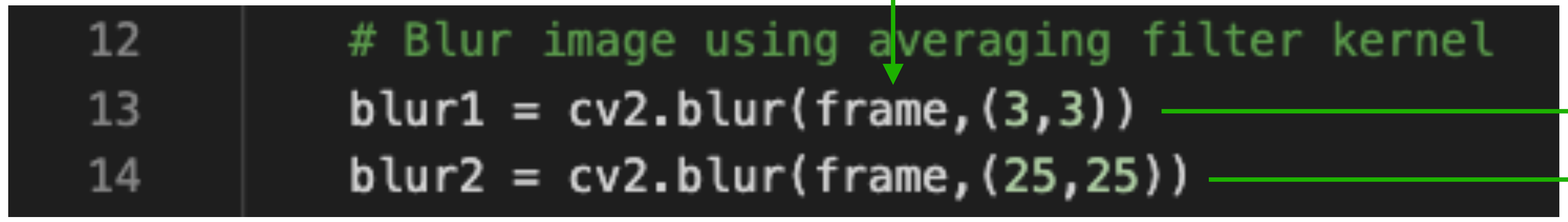


3x3 Kernel



25x25 Kernel

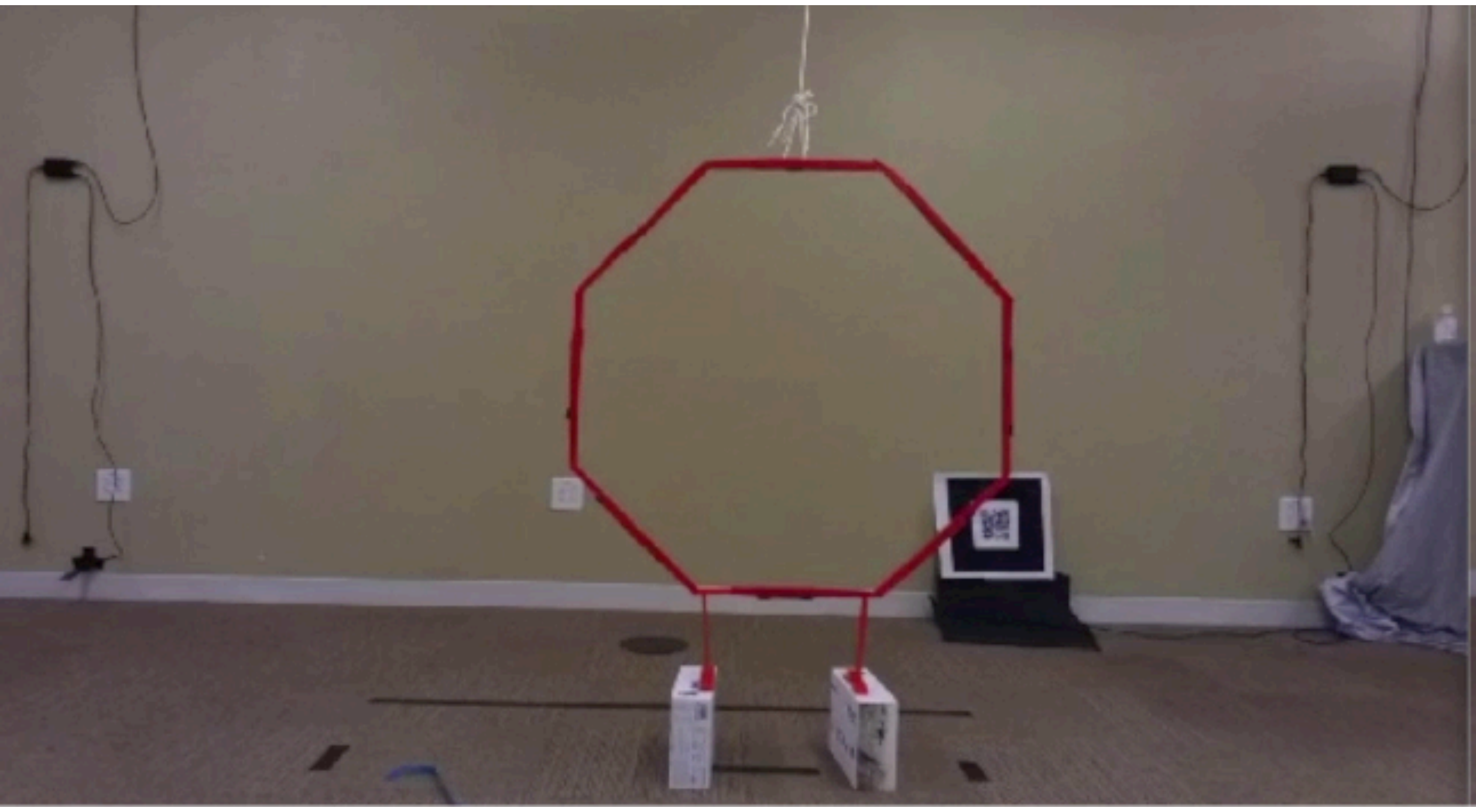




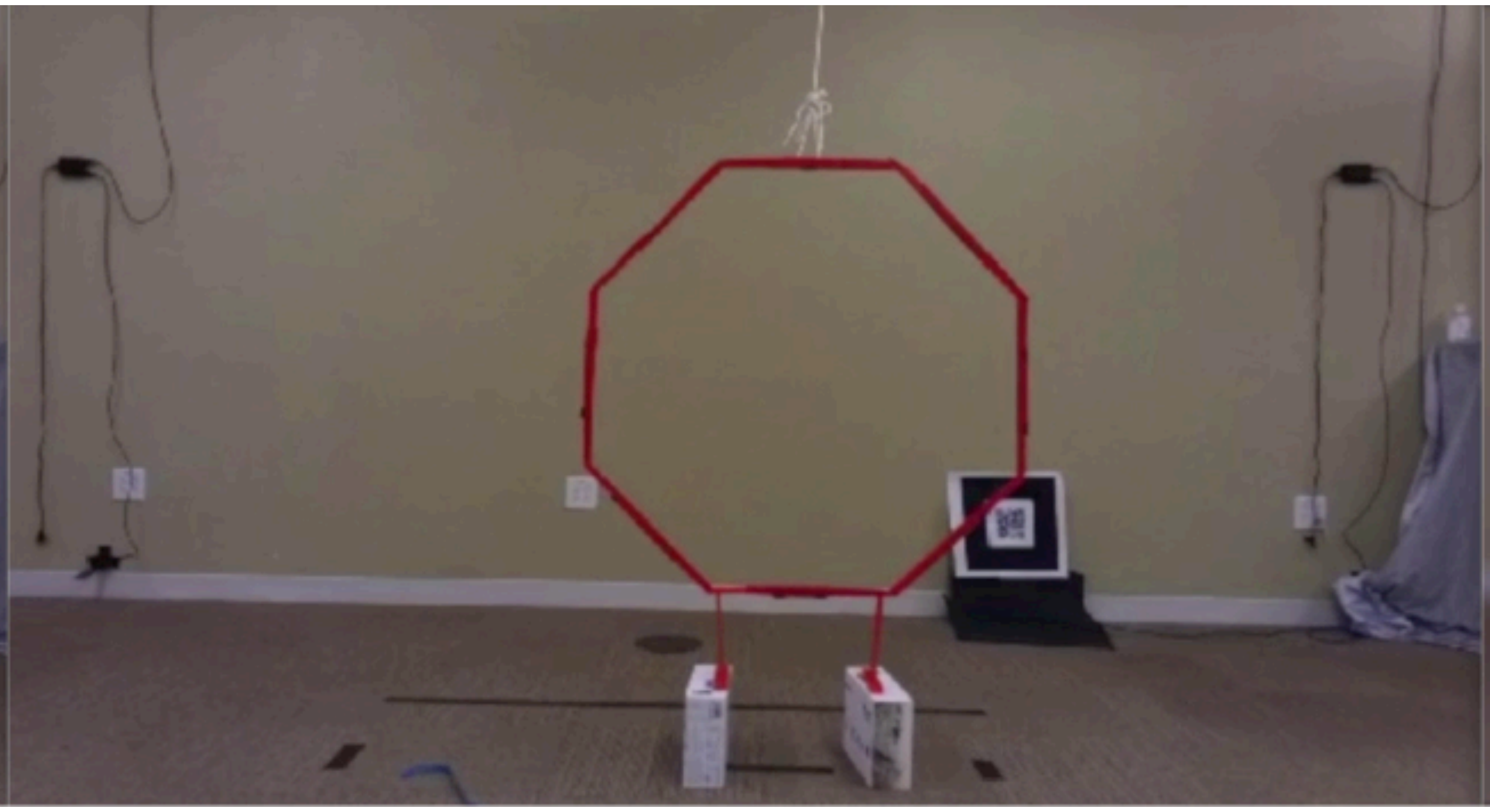
```
# Blur image using averaging filter kernel
blur1 = cv2.blur(frame,(3,3))
blur2 = cv2.blur(frame,(25,25))
```

Question

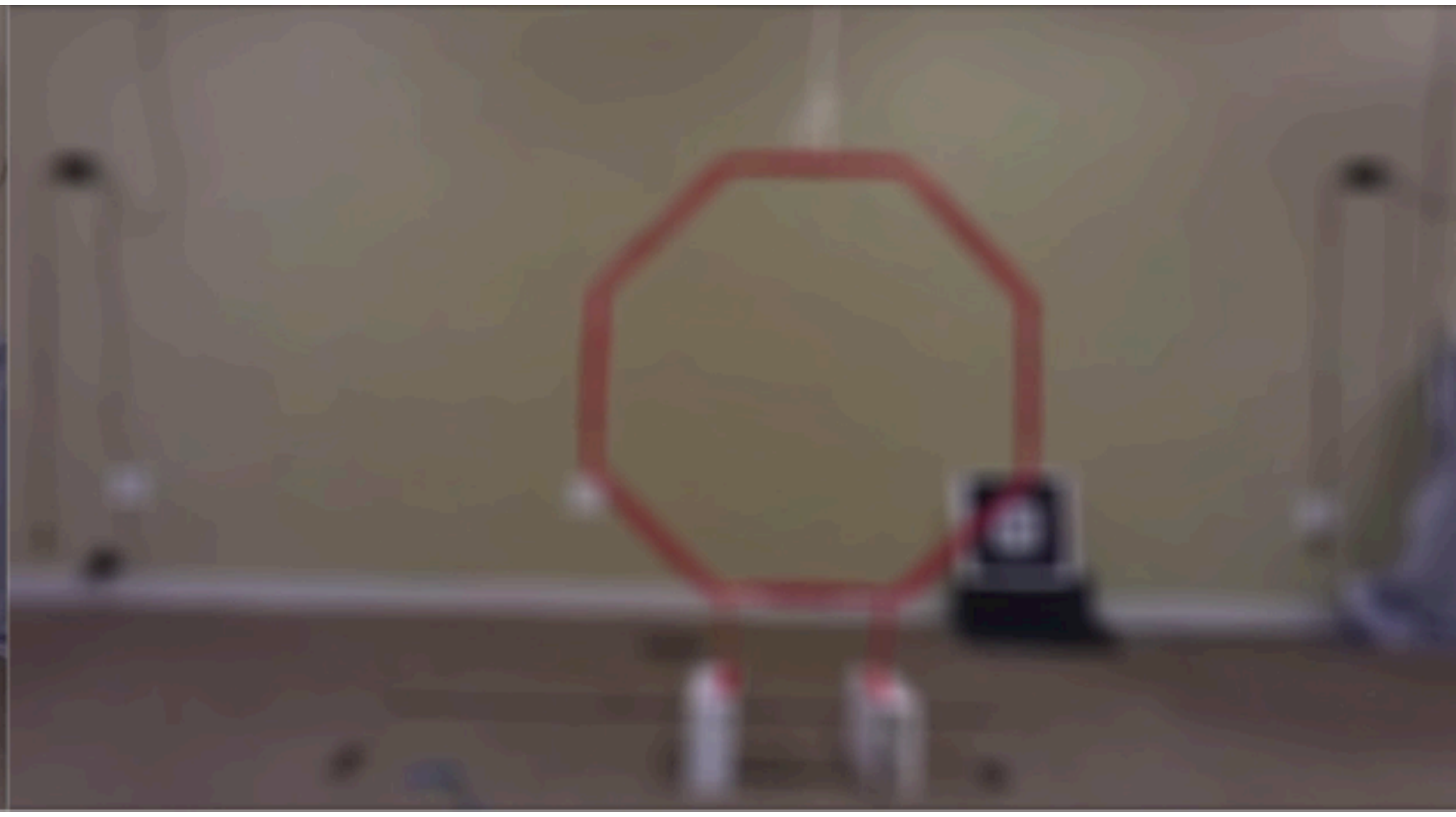
Raw Data



3x3 Kernel



25x25 Kernel



Why would we want to do this?

Image Processing Techniques

- **Thresholding**
- **Color Filtering**
- **Blurring**
- **Smoothing**
- **Background subtraction**
- **Edge Detection**
- **Corner Detection**
- **Feature Matching**
- **Haar Cascade Object Detection**
- **…**

(Canny) Edge Detection

Idea
Determine the horizontal and vertical gradient, large gradient == edge

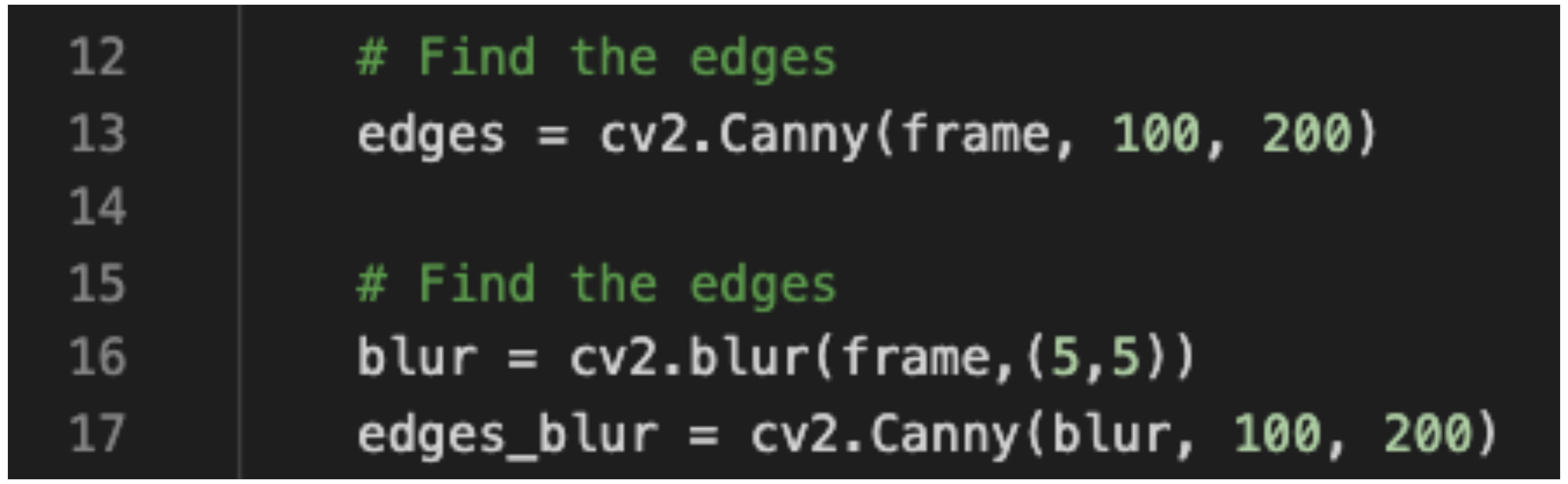
Technical Key

1) Apply gaussian filter to smooth the image and remove noise
2) Find the gradients of the image using Sobel operator
3) Apply non max suppression to thin edges
4) Apply double threshold to determine strong and weak edges
5) Track edges to remove edges that are not connected to a strong edge

Finding Gradients (Sobel Operator)

$$L_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} L \quad \text{and} \quad L_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} L.$$

Code



```
    # Find the edges
    edges = cv2.Canny(frame, 100, 200)

    # Find the edges
    blur = cv2.blur(frame,(5,5))
    edges_blur = cv2.Canny(blur, 100, 200)
```

Original Image

Canny Edge Detector: https://sbme-tutorials.github.io/2018/cv/notes/4_week4.html

(Canny) Edge Detection

Idea

Determine the horizontal and vertical gradient, large gradient == edge

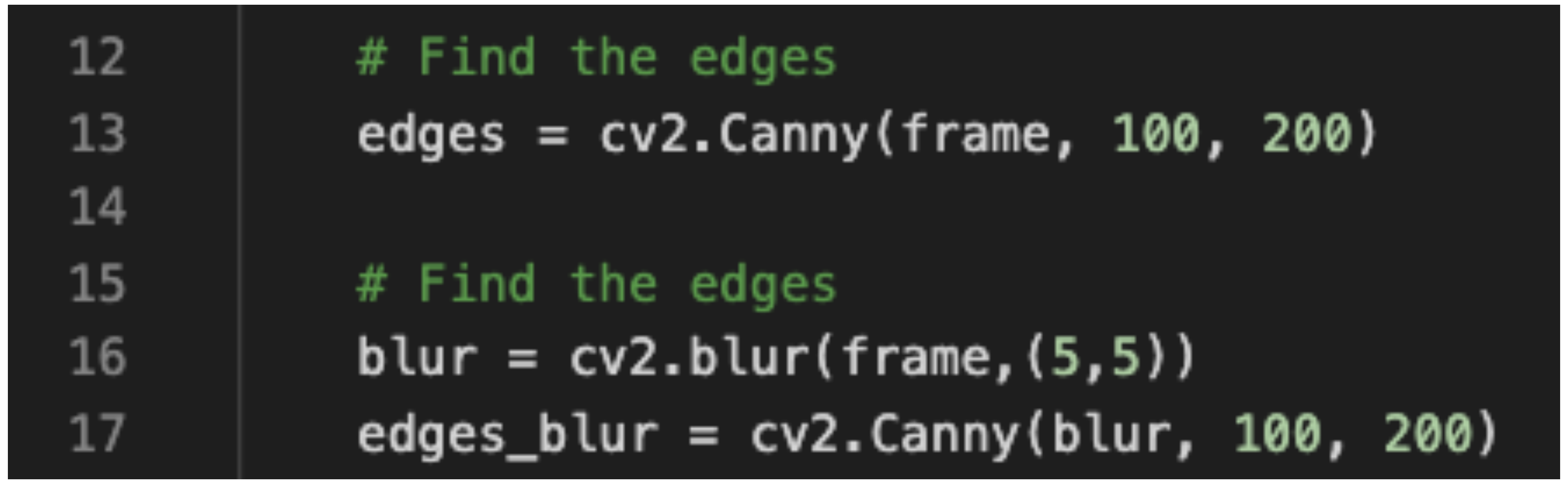
Technical Key

1) Apply gaussian filter to smooth the image and remove noise
2) Find the gradients of the image using Sobel operator
3) Apply non max suppression to thin edges
4) Apply double threshold to determine strong and weak edges
5) Track edges to remove edges that are not connected to a strong edge

Finding Gradients (Sobel Operator)

$$L_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} L \quad \text{and} \quad L_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} L.$$

Code



```
    # Find the edges
    edges = cv2.Canny(frame, 100, 200)

    # Find the edges
    blur = cv2.blur(frame,(5,5))
    edges_blur = cv2.Canny(blur, 100, 200)
```

Gaussian Filter

Canny Edge Detector: https://sbme-tutorials.github.io/2018/cv/notes/4_week4.html

(Canny) Edge Detection

Idea
Determine the horizontal and vertical gradient, large gradient == edge

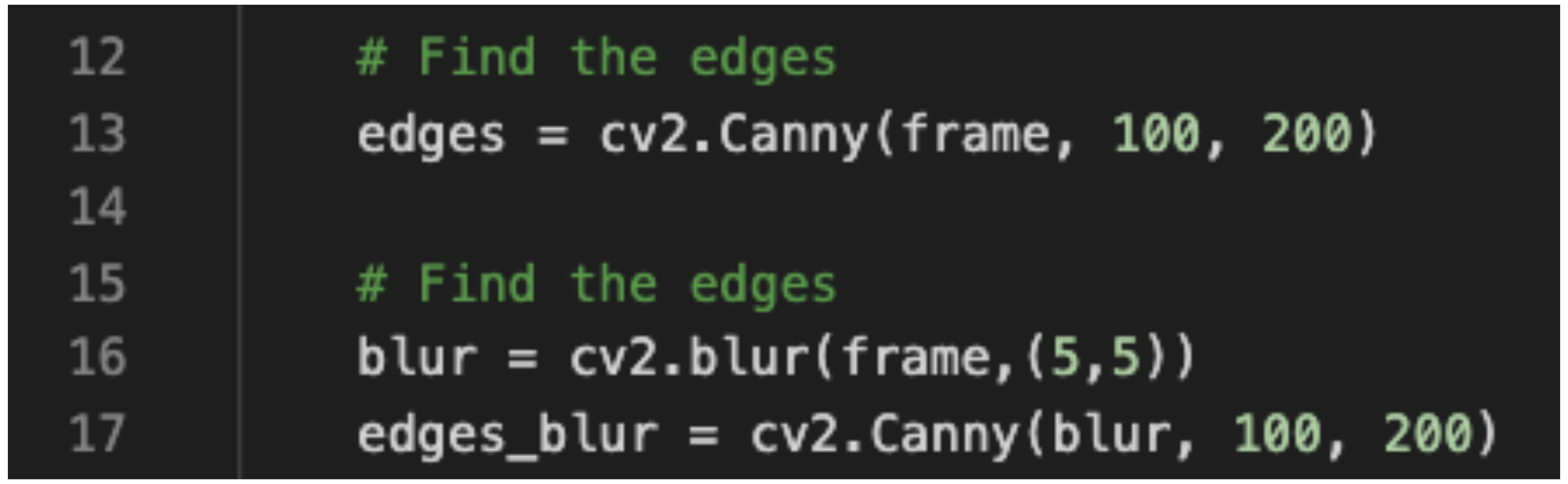
Technical Key

1) Apply gaussian filter to smooth the image and remove noise
2) Find the gradients of the image using Sobel operator ←
3) Apply non max suppression to thin edges
4) Apply double threshold to determine strong and weak edges
5) Track edges to remove edges that are not connected to a strong edge

Finding Gradients (Sobel Operator)

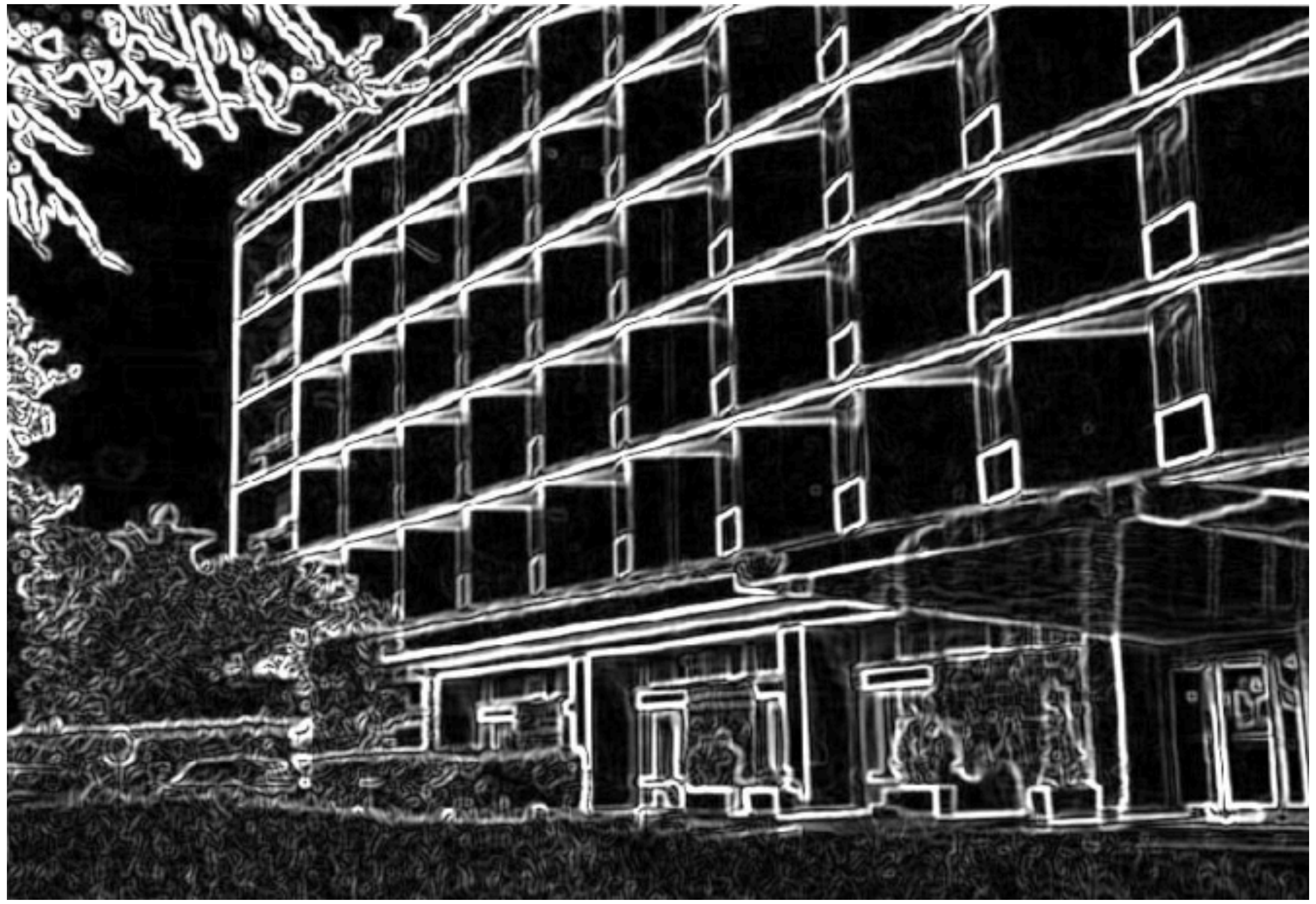
$$L_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} L \quad \text{and} \quad L_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} L.$$

Code



```python
    # Find the edges
    edges = cv2.Canny(frame, 100, 200)

    # Find the edges
    blur = cv2.blur(frame,(5,5))
    edges_blur = cv2.Canny(blur, 100, 200)
```

Gradient Magnitude



Canny Edge Detector: https://sbme-tutorials.github.io/2018/cv/notes/4_week4.html

(Canny) Edge Detection

Idea

Determine the horizontal and vertical gradient, large gradient == edge

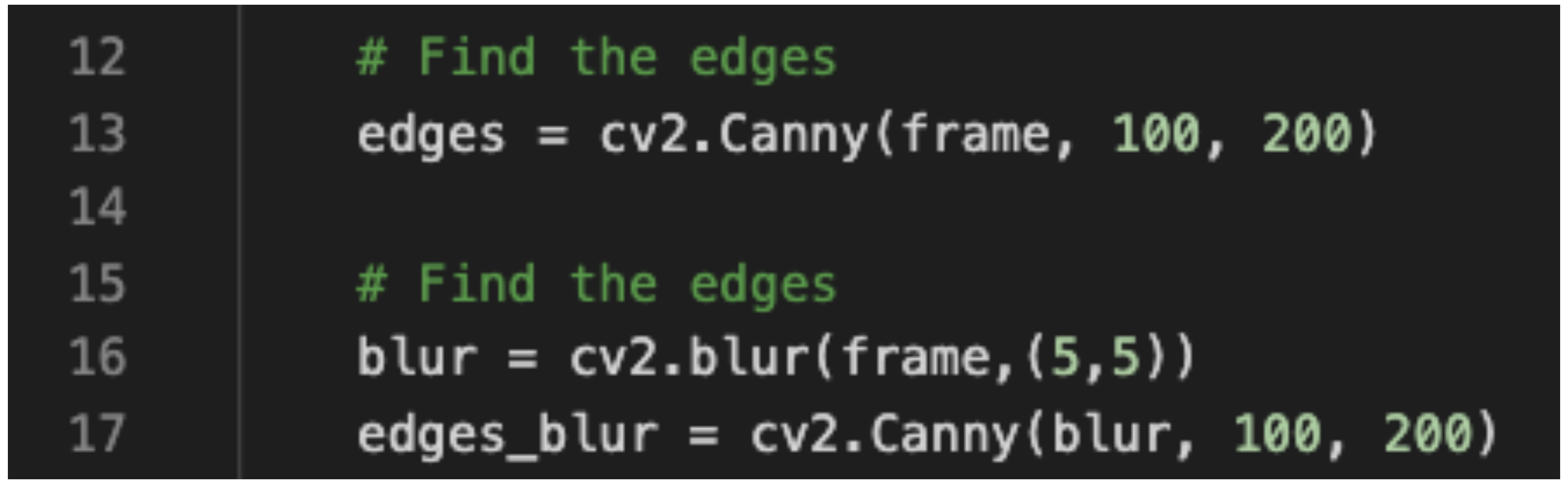
Technical Key

1) Apply gaussian filter to smooth the image and remove noise
2) Find the gradients of the image using Sobel operator
3) Apply non max suppression to thin edges ←
4) Apply double threshold to determine strong and weak edges
5) Track edges to remove edges that are not connected to a strong edge

Finding Gradients (Sobel Operator)

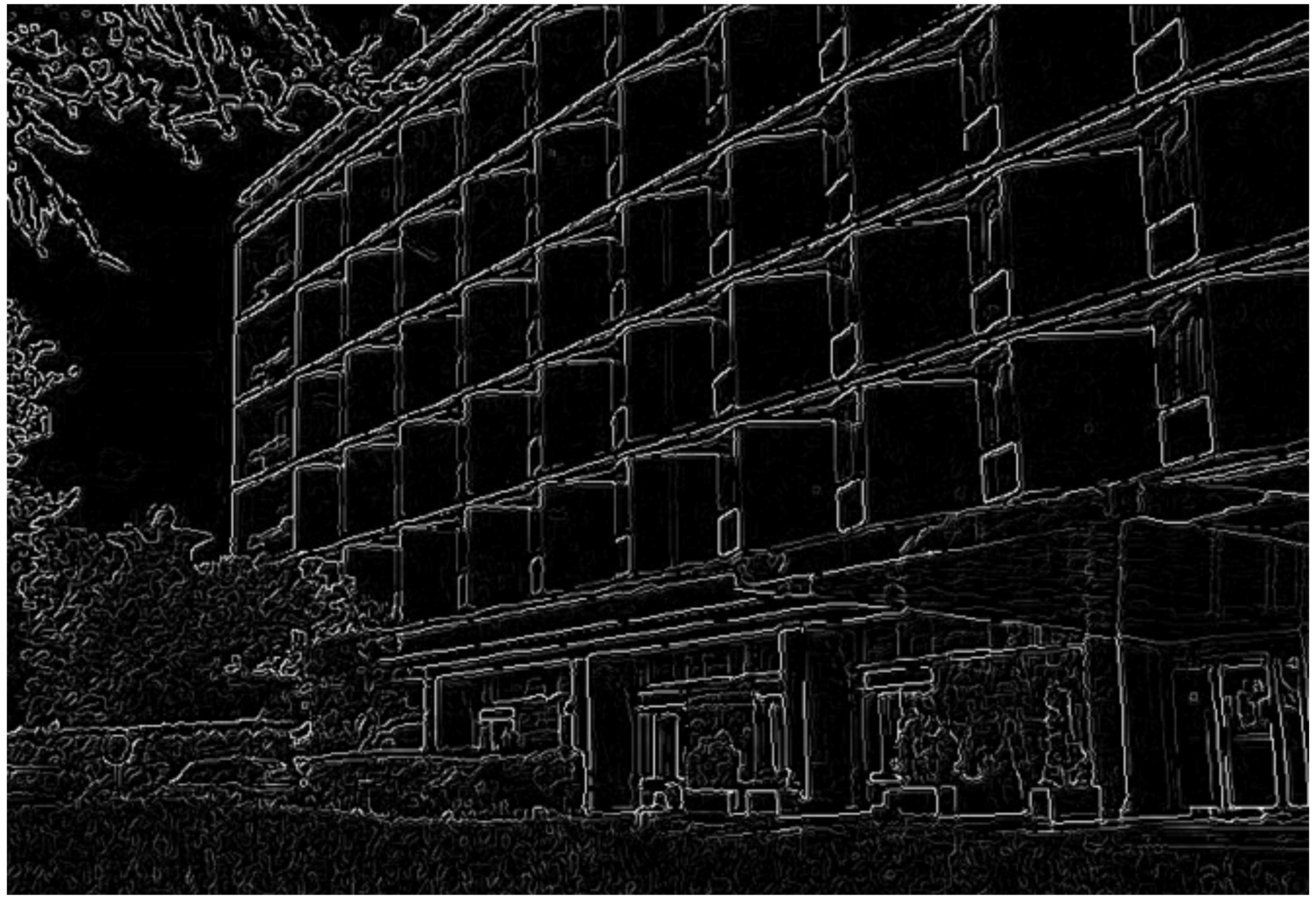
$$L_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} L \quad \text{and} \quad L_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} L.$$

Code



```
    # Find the edges
    edges = cv2.Canny(frame, 100, 200)

    # Find the edges
    blur = cv2.blur(frame,(5,5))
    edges_blur = cv2.Canny(blur, 100, 200)
```

Non Max Suppression



Canny Edge Detector: https://sbme-tutorials.github.io/2018/cv/notes/4_week4.html

(Canny) Edge Detection

Idea

Determine the horizontal and vertical gradient, large gradient == edge

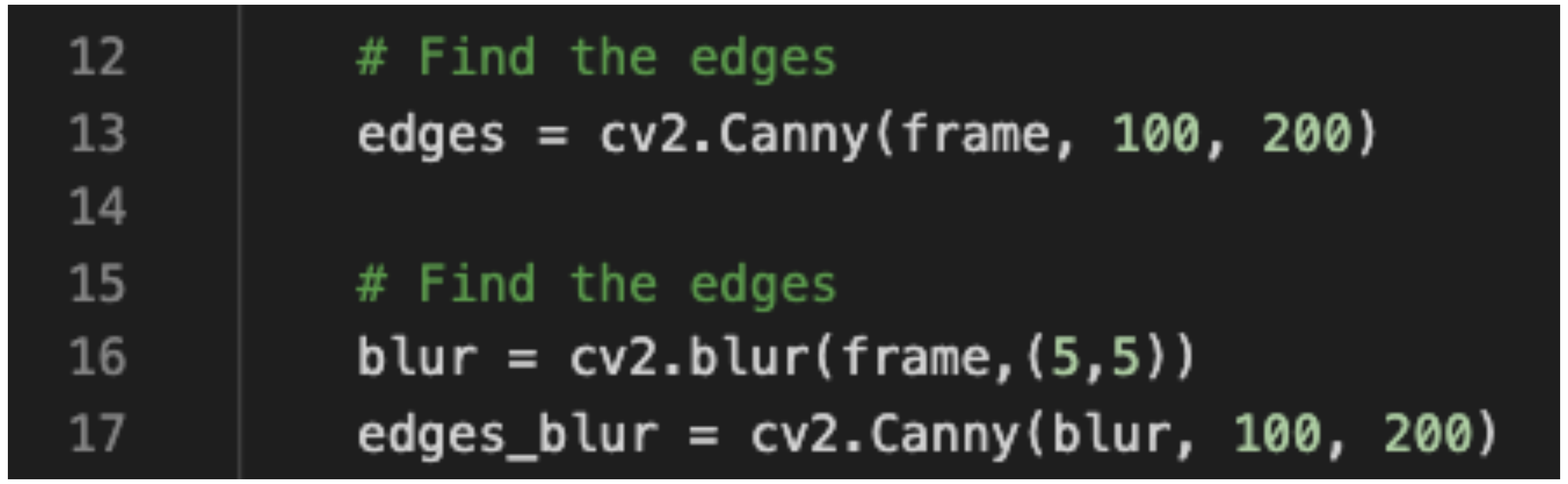
Technical Key

1) Apply gaussian filter to smooth the image and remove noise
2) Find the gradients of the image using Sobel operator
3) Apply non max suppression to thin edges
4) Apply double threshold to determine strong and weak edges ←
5) Track edges to remove edges that are not connected to a strong edge

Finding Gradients (Sobel Operator)

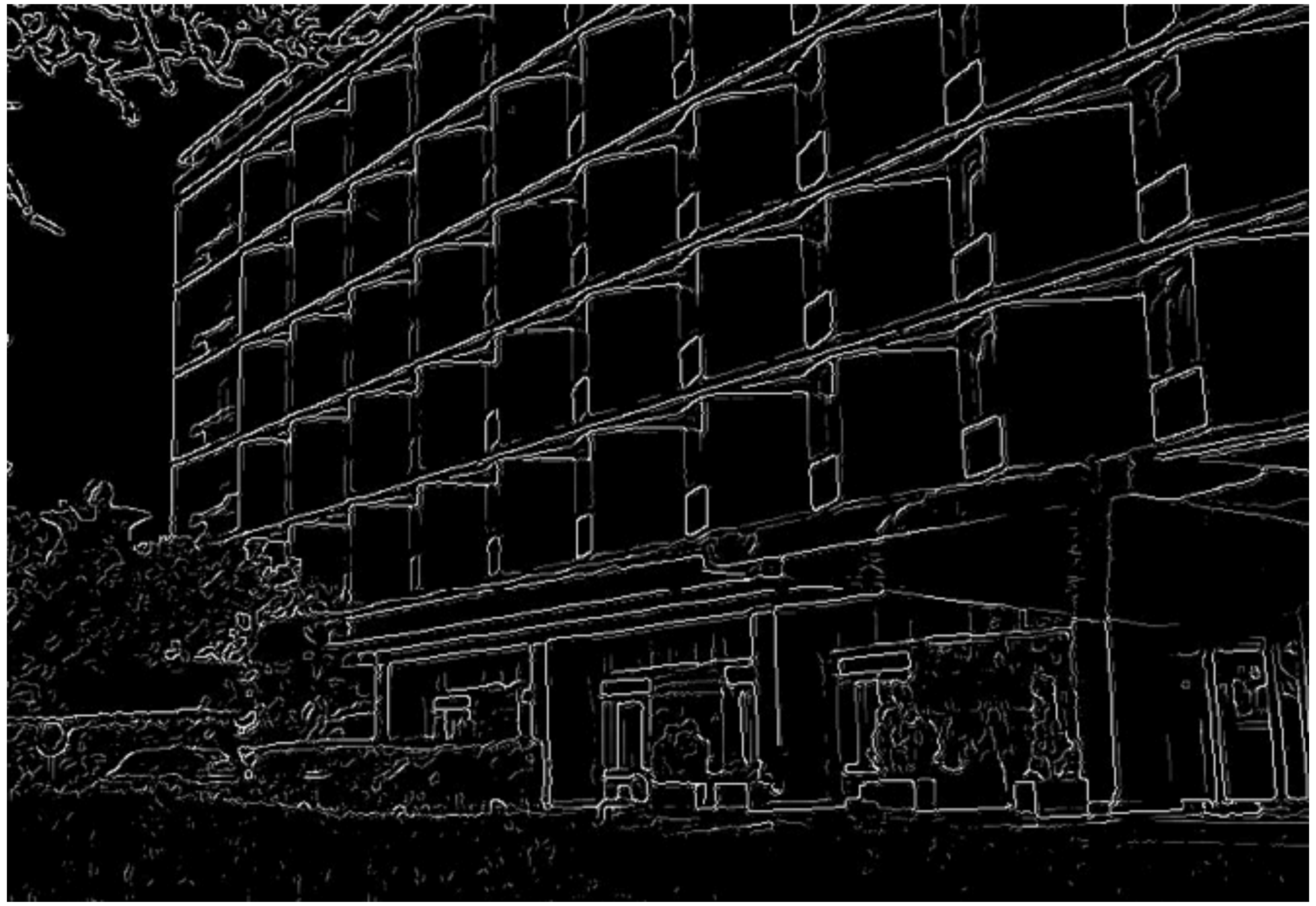
$$L_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} L \quad \text{and} \quad L_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} L.$$

Code



```
    # Find the edges
    edges = cv2.Canny(frame, 100, 200)

    # Find the edges
    blur = cv2.blur(frame,(5,5))
    edges_blur = cv2.Canny(blur, 100, 200)
```

Double Thresholding



Canny Edge Detector: https://sbme-tutorials.github.io/2018/cv/notes/4_week4.html

(Canny) Edge Detection

Idea

Determine the horizontal and vertical gradient, large gradient == edge

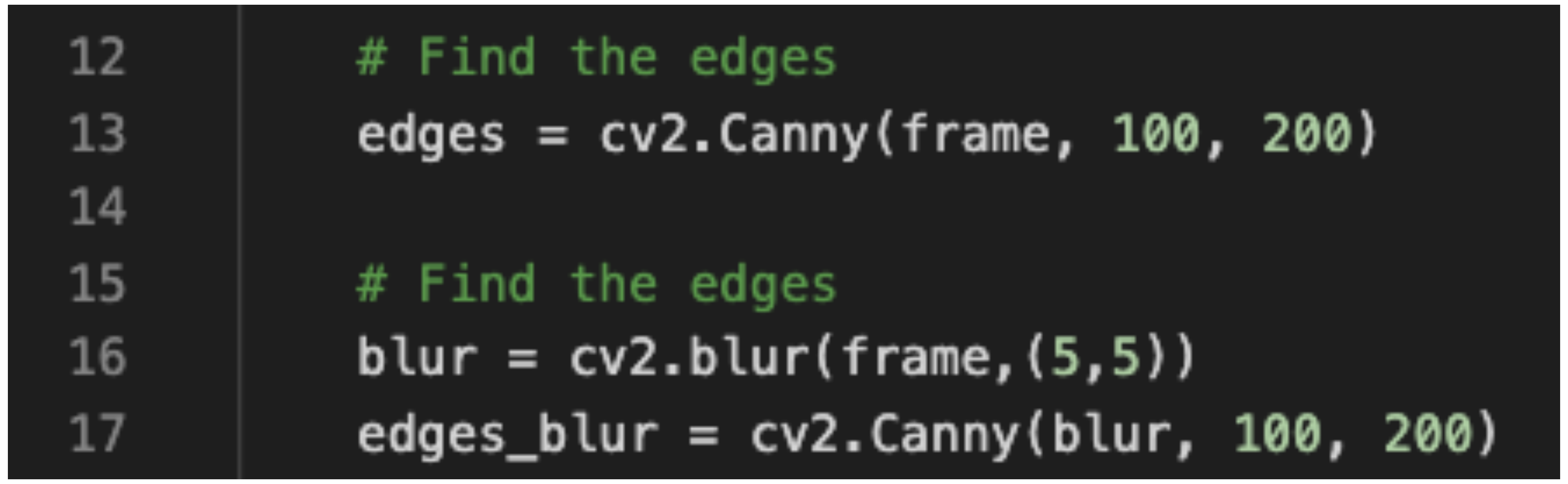
Technical Key

1) Apply gaussian filter to smooth the image and remove noise
2) Find the gradients of the image using Sobel operator
3) Apply non max suppression to thin edges
4) Apply double threshold to determine strong and weak edges
5) Track edges to remove edges that are not connected to a strong edge ←

Finding Gradients (Sobel Operator)

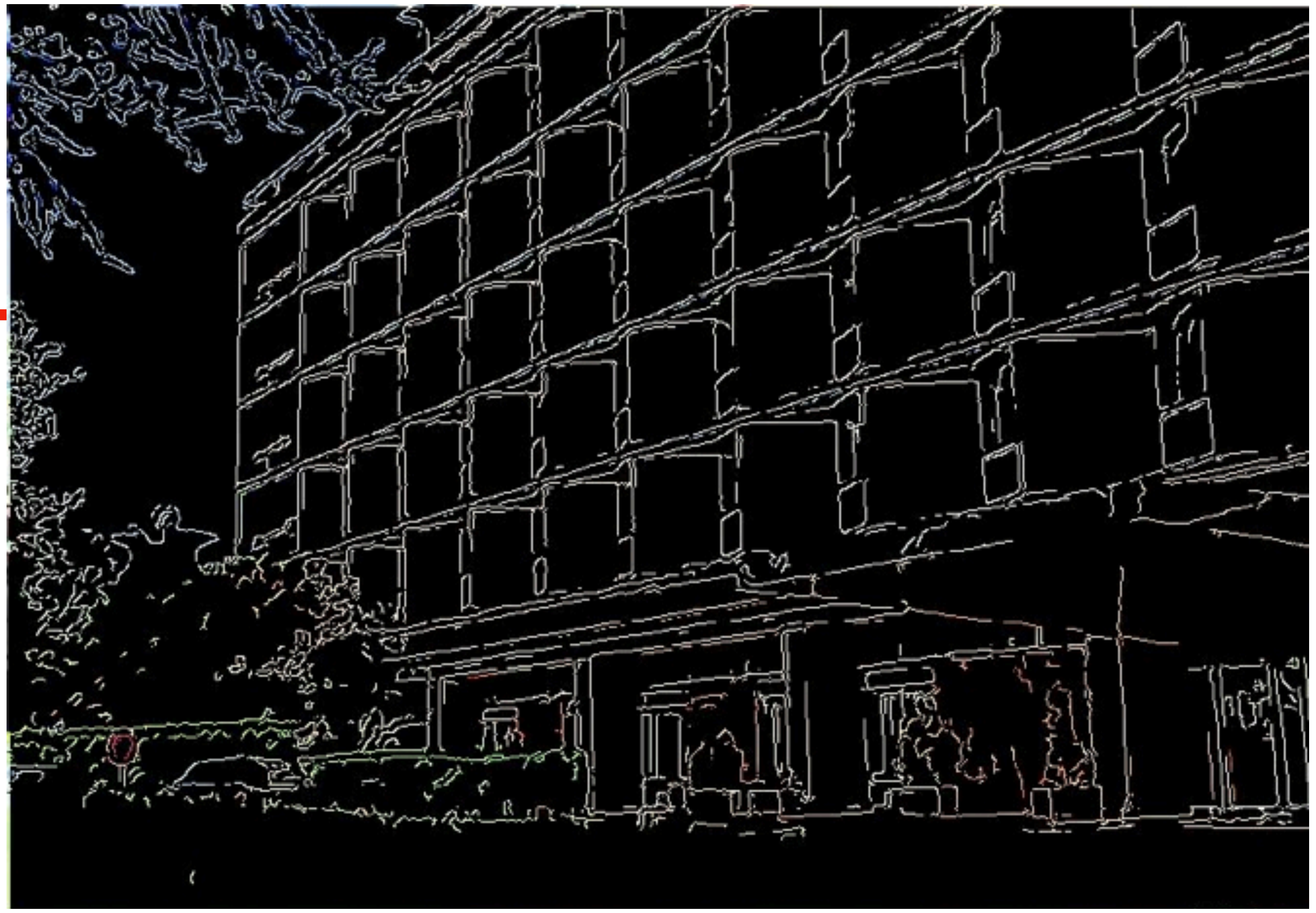
$$L_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} L \quad \text{and} \quad L_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} L.$$

Code



```
    # Find the edges
    edges = cv2.Canny(frame, 100, 200)

    # Find the edges
    blur = cv2.blur(frame,(5,5))
    edges_blur = cv2.Canny(blur, 100, 200)
```

Edge Tracking



Canny Edge Detector: https://sbme-tutorials.github.io/2018/cv/notes/4_week4.html

(Canny) Edge Detection

Idea

Determine the horizontal and vertical gradient, large gradient == edge

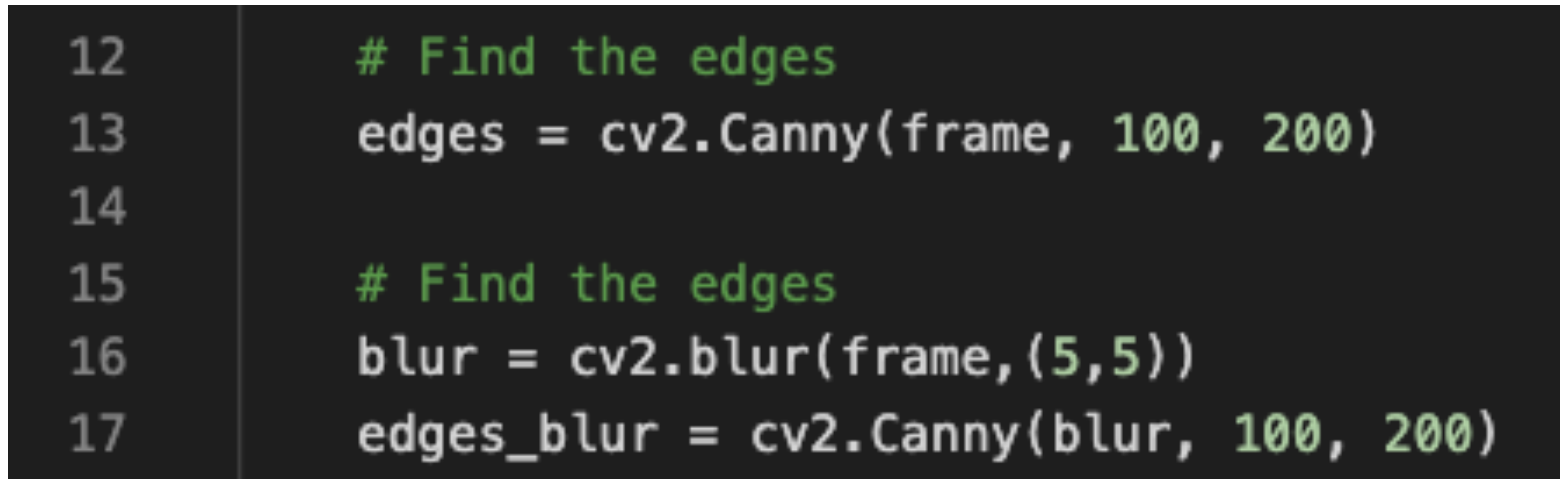
Technical Key

1) Apply gaussian filter to smooth the image and remove noise
2) Find the gradients of the image using Sobel operator
3) Apply non max suppression to thin edges
4) Apply double threshold to determine strong and weak edges
5) Track edges to remove edges that are not connected to a strong edge

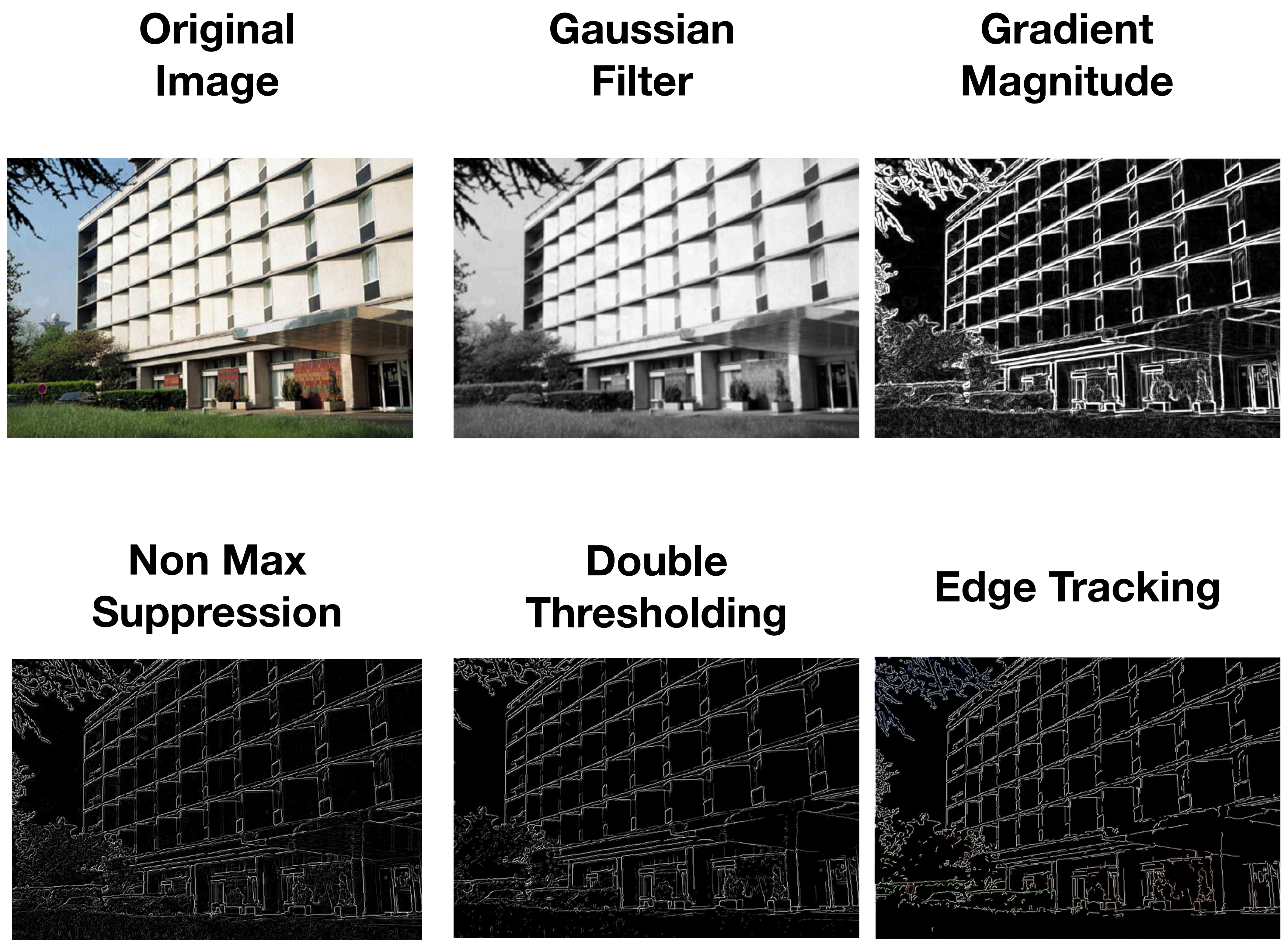
Finding Gradients (Sobel Operator)

$$L_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} L \quad \text{and} \quad L_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} L.$$

Code



```
    # Find the edges
    edges = cv2.Canny(frame, 100, 200)

    # Find the edges
    blur = cv2.blur(frame,(5,5))
    edges_blur = cv2.Canny(blur, 100, 200)
```



Original Image

Gaussian Filter

Gradient Magnitude

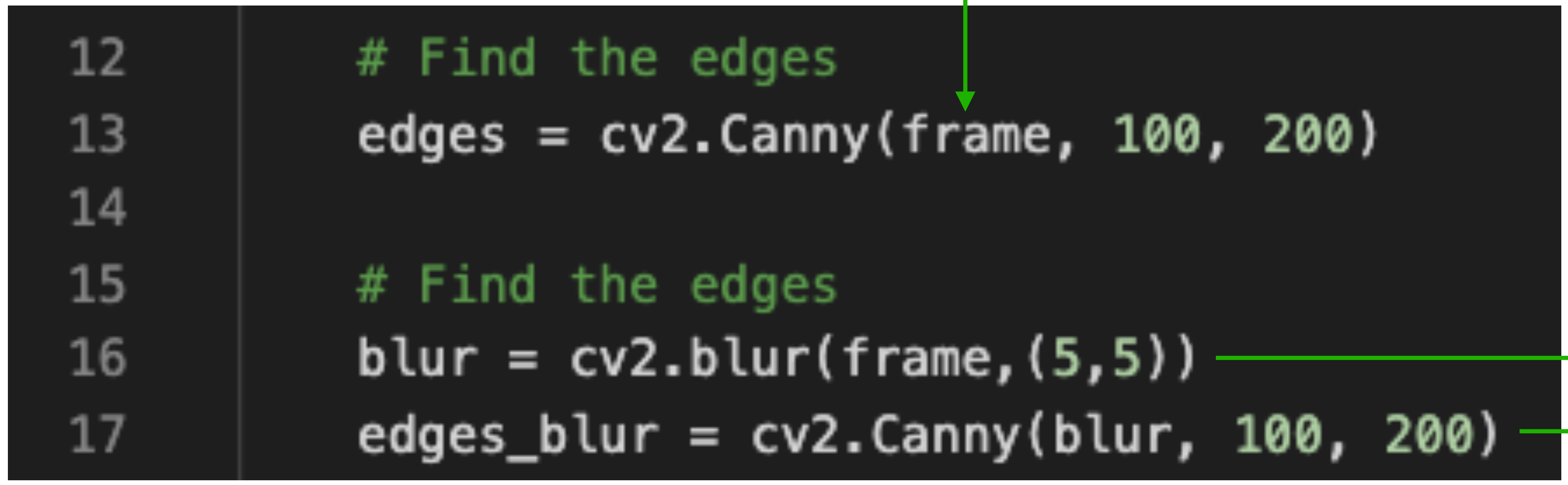
Non Max Suppression

Double Thresholding

Edge Tracking

Canny Edge Detector: https://sbme-tutorials.github.io/2018/cv/notes/4_week4.html

Example: Edge Detection



```
    # Find the edges
    edges = cv2.Canny(frame, 100, 200)

    # Find the edges
    blur = cv2.blur(frame,(5,5))
    edges_blur = cv2.Canny(blur, 100, 200)
```

Example: Edge Detection

Raw Data **Edge Detection** **5x5 Blur -> Edge Detection**

What are some of the limitations of this?

Perception Algorithms

Perception estimates the state of the environment

Image Processing
An image is processed through parameterized transformations.

Key: We define this function

Machine Learning
Gather large amounts of data a to learn or approximate the desired function.

Key: We learn this function

Perception Algorithms

Perception estimates the state of the environment

Image Processing
An image is processed through parameterized transformations.

Key: We define this function

Machine Learning
Gather large amounts of data a to learn or approximate the desired function.

Key: We learn this function

What are the pros and cons of image processing?

Perception Algorithms

Perception estimates the state of the environment

Image Processing Algorithms
An image is processed through parameterized transformations.

Key: We define this function

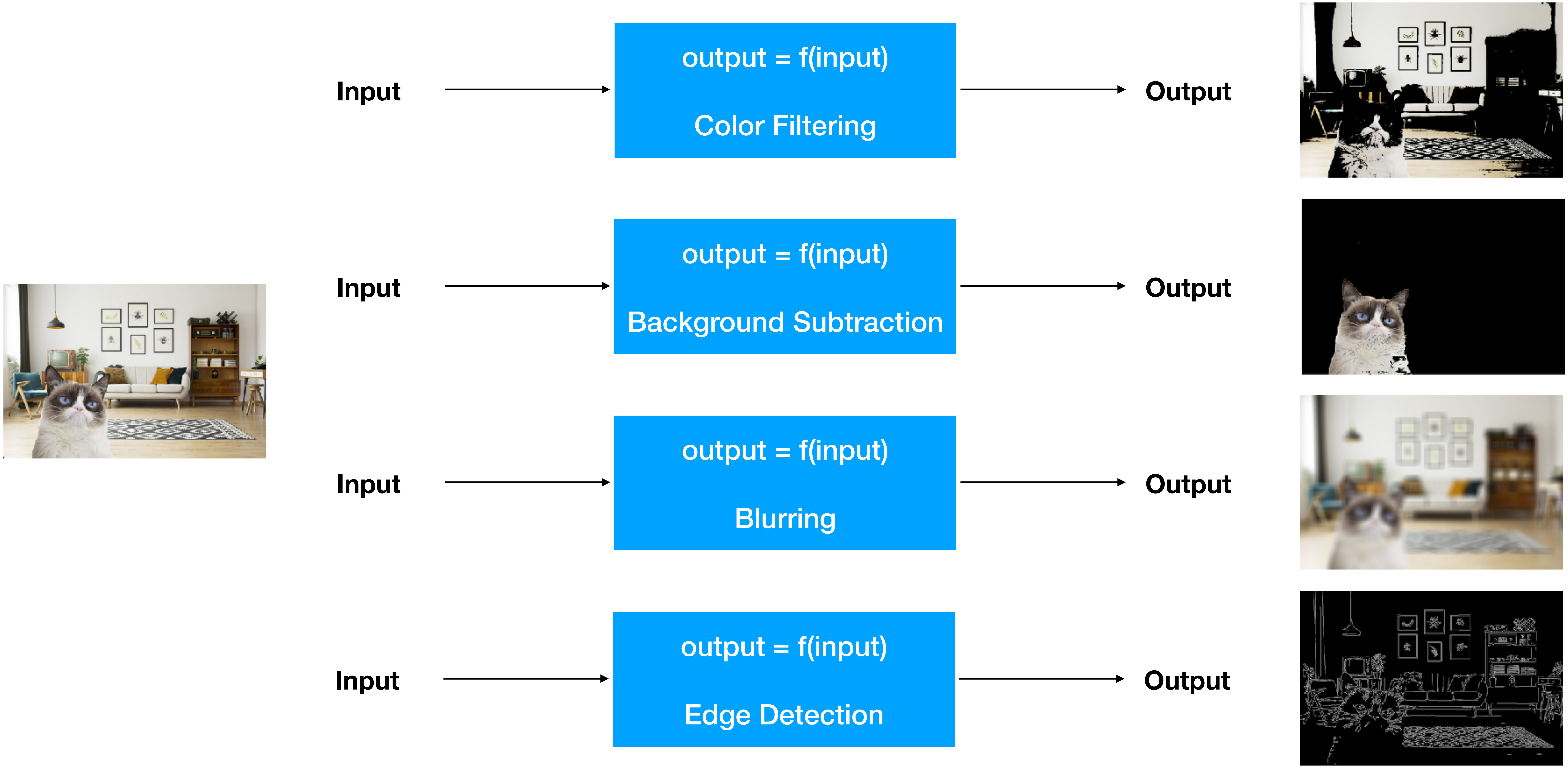
Pros:
Does not require datasets at all
Are easier to interpret by humans
Most do not require heavy computation resources
Libraries available to perform most standard functions

Cons:
Encode relatively simple functions

Machine Learning
Gather large amounts of data a to learn or approximate the desired function.

Key: We learn this function

Perception Algorithms



Input → output = f(input)

Color Filtering → Output

Input → output = f(input)

Background Subtraction → Output

Input → output = f(input)

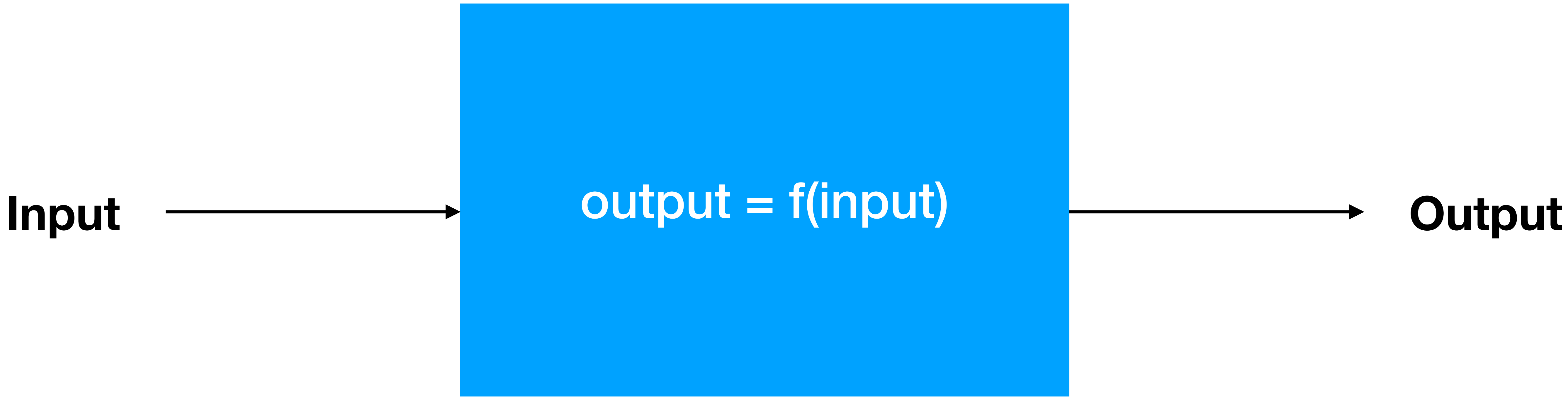
Blurring → Output

Input → output = f(input)

Edge Detection → Output

Machine Learning

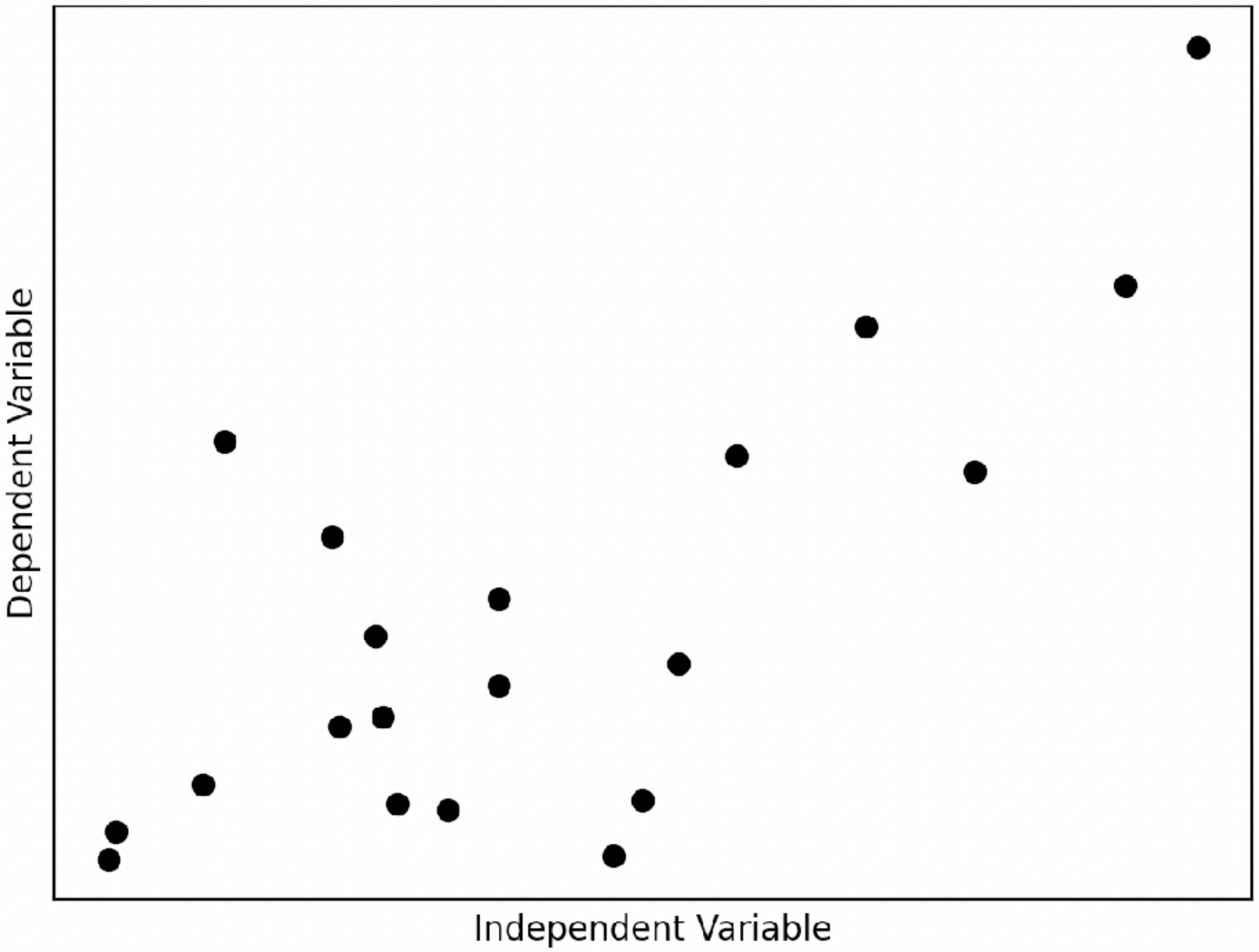
What happens if we don’t know exactly how to define the function?



Regression

What is the relationship between a dependent and one or more independent variables?

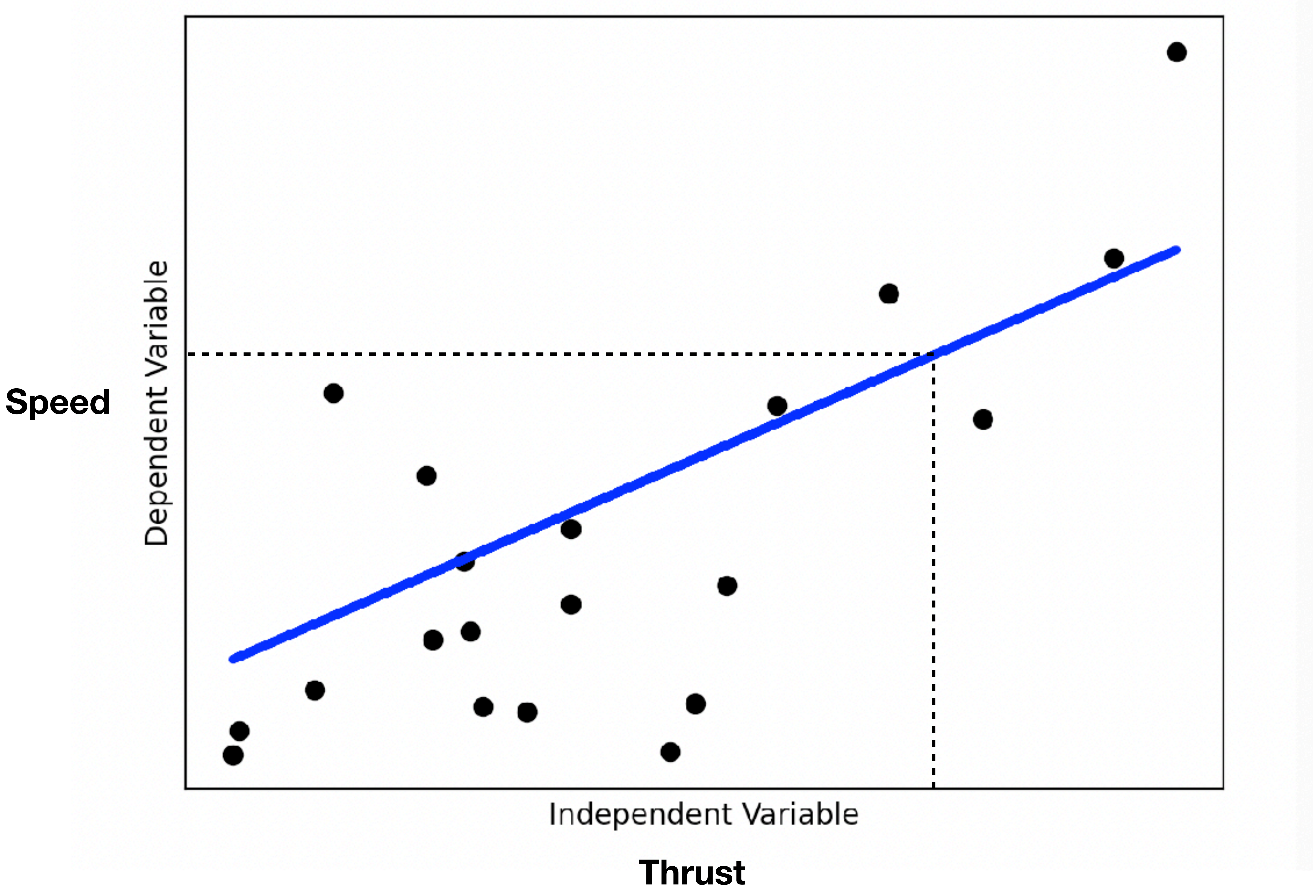
Speed

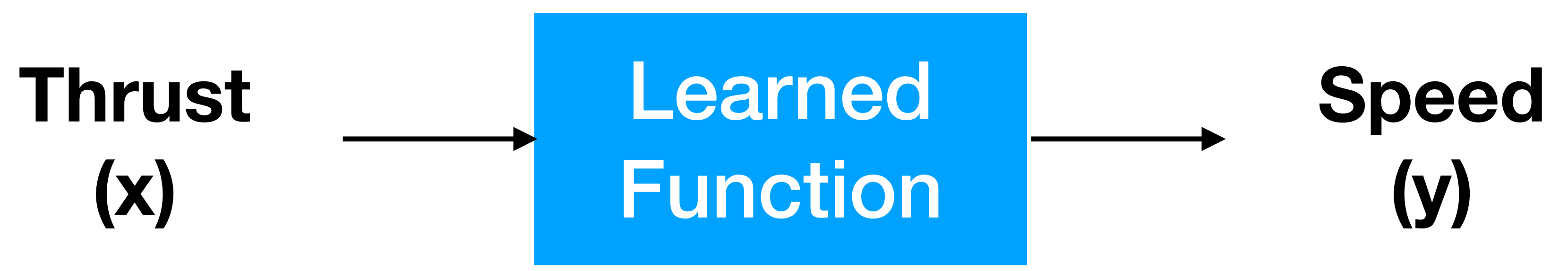


Thrust

Regression

What is the relationship between a dependent and one or more independent variables?



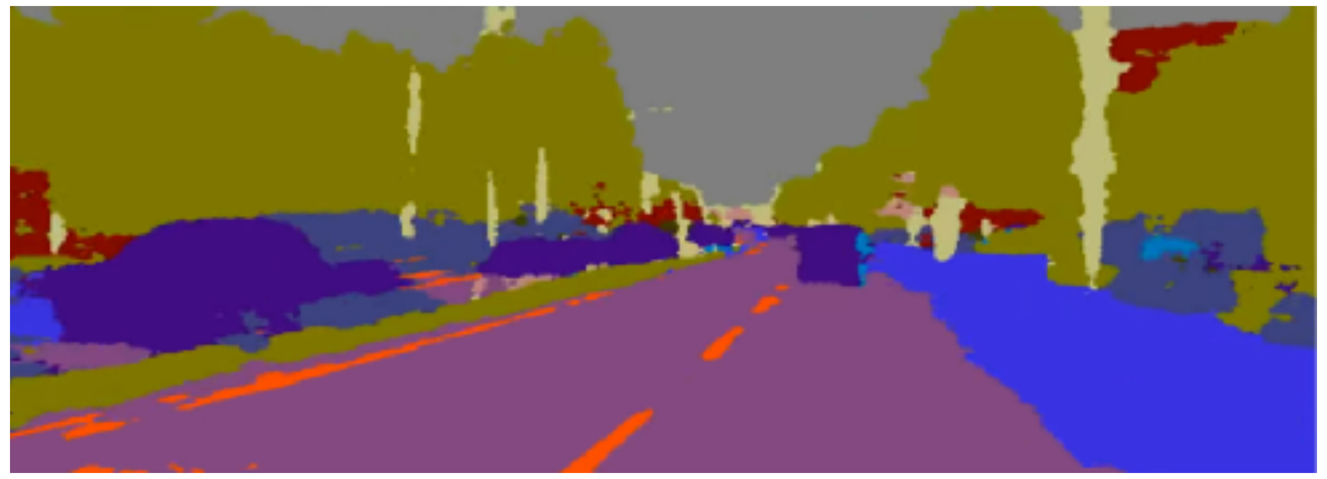

Thrust (x)

Learned Function

Speed (y)

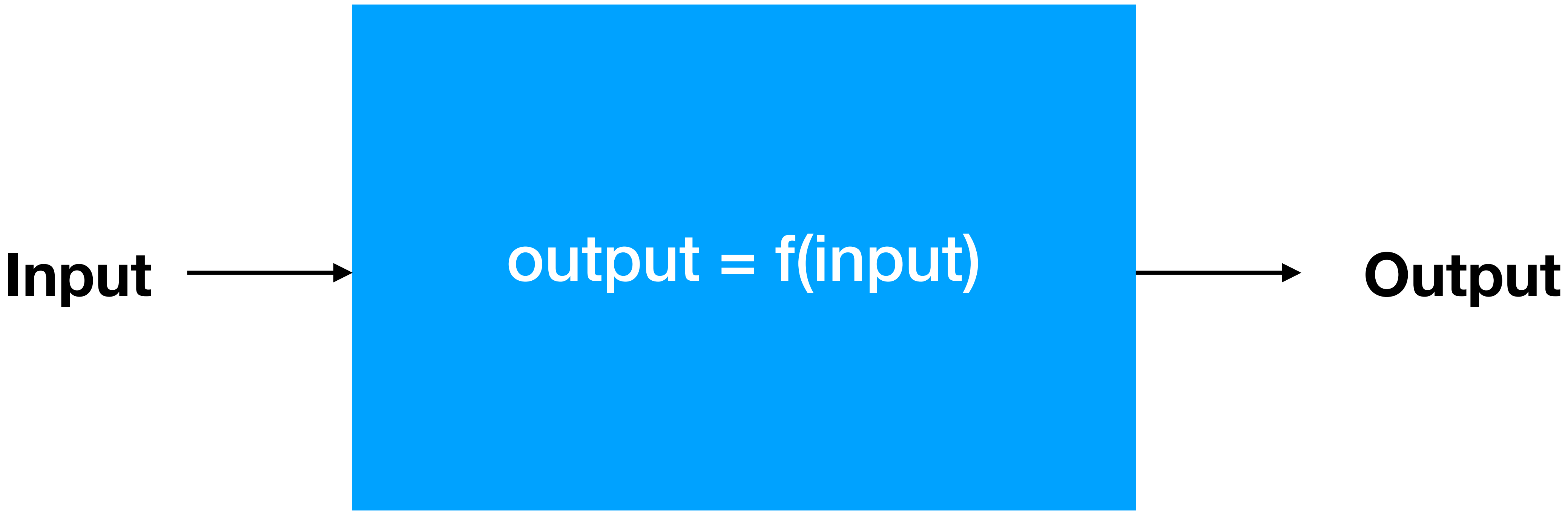
Machine Learning

What happens if we have a much more complicated task?

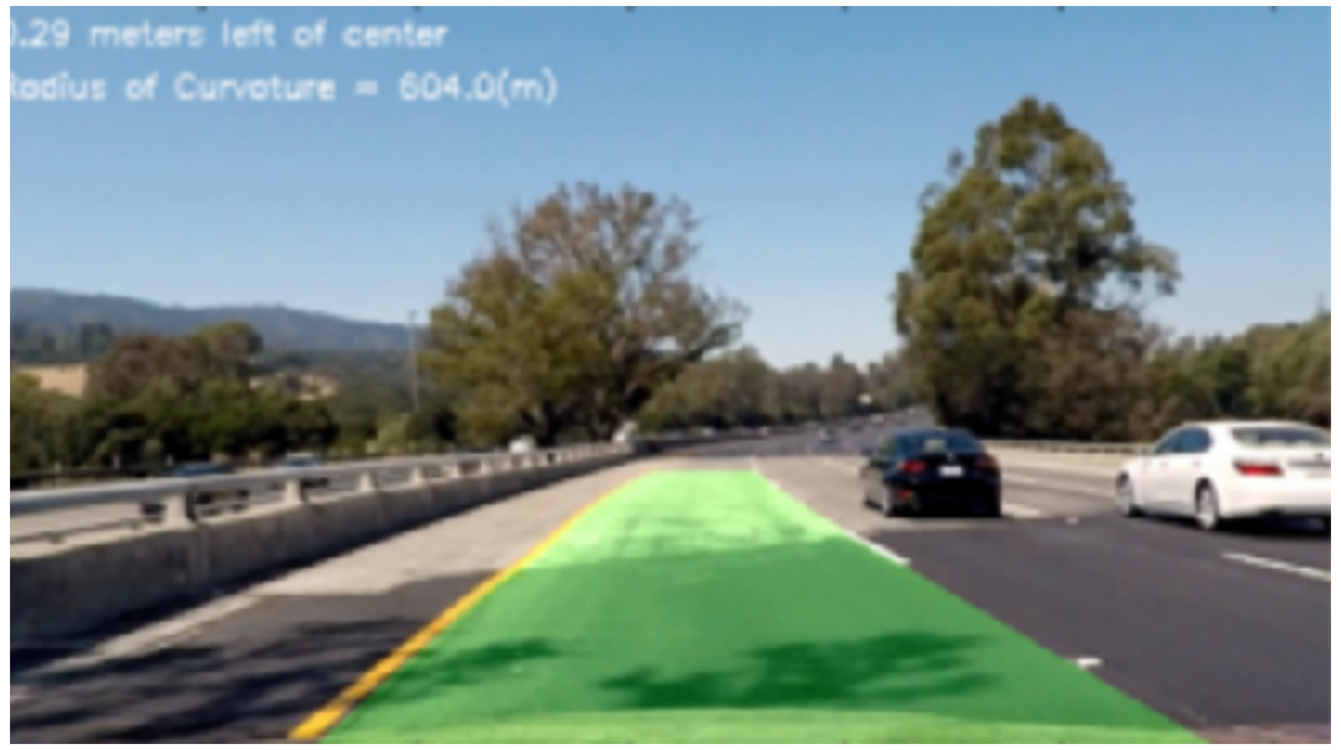


Machine learning learns this function given enough data

3779

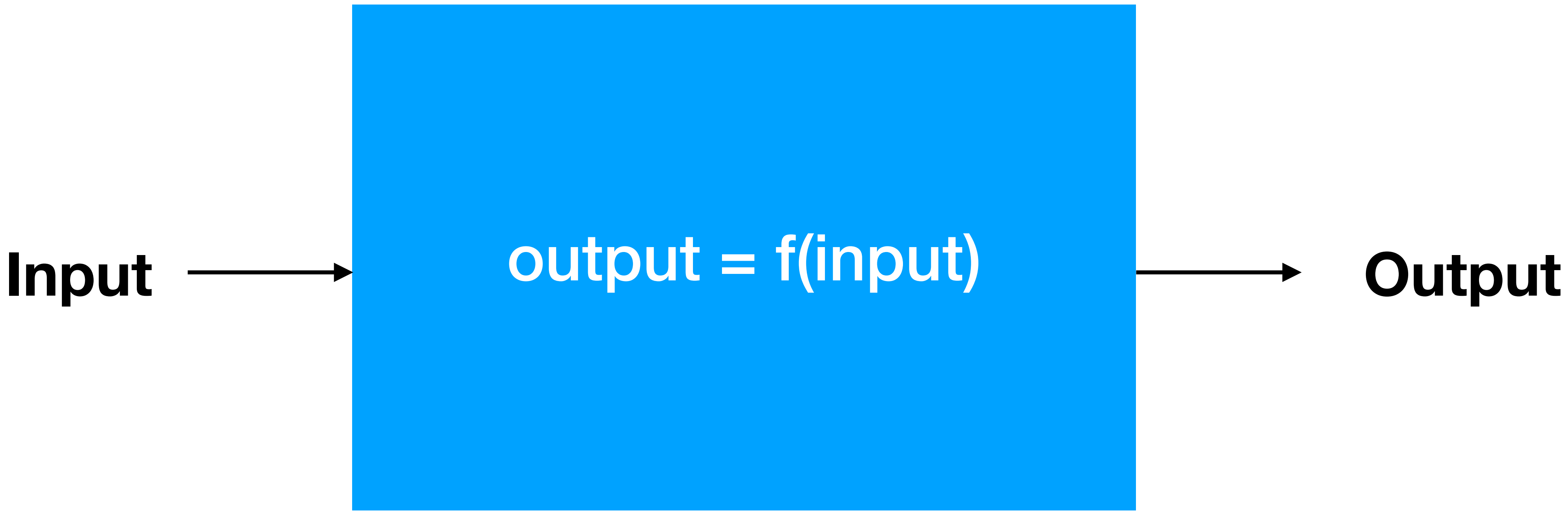


Plane, Car, Bird, Cat



Neural Networks

How do we learn a function?



Neural Networks - Training

Label: Cat

Label: Dog

Label: Cat

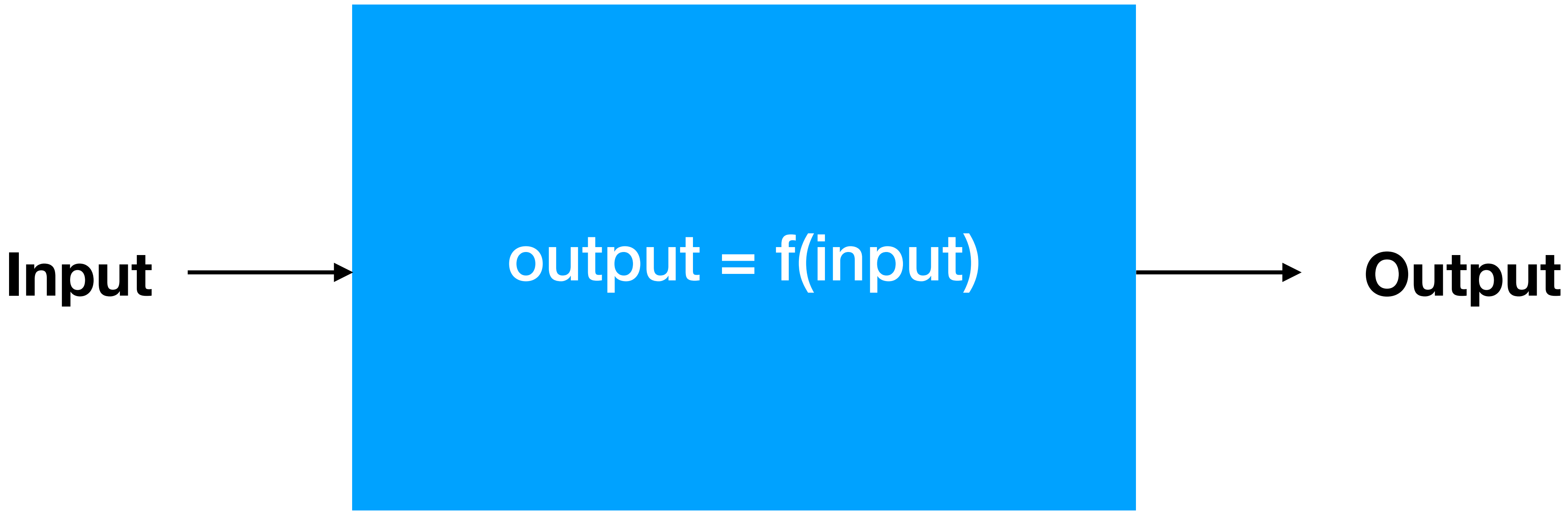
Label: Dog

Label: Cat

...

...

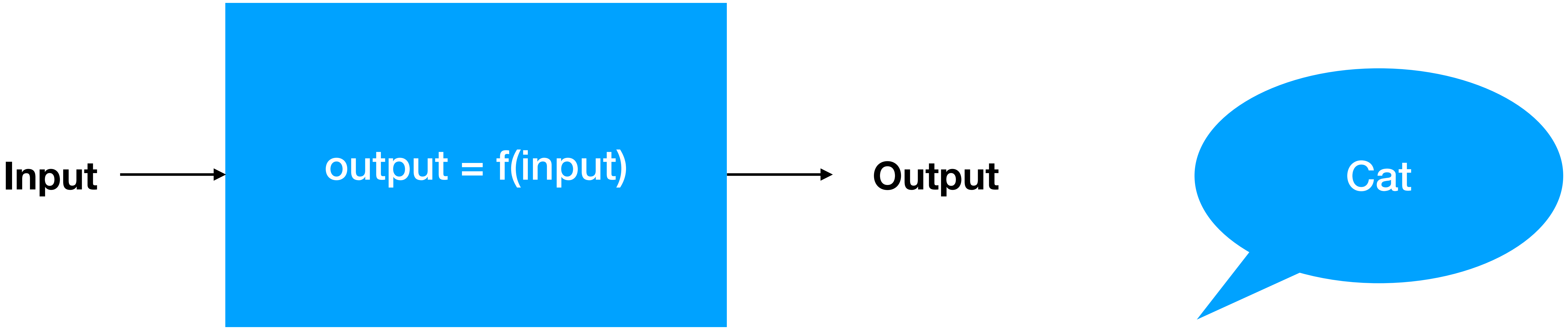
Label: Cat



Input → output = f(input) → **Output**

Neural Networks

Input



Input

output = f(input)

Output

Cat

Output

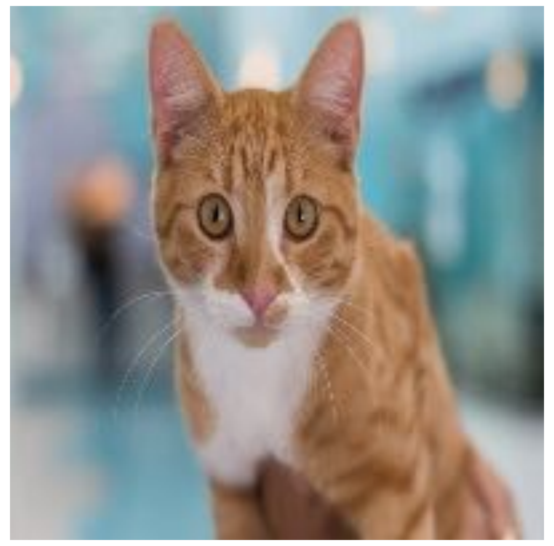
Neural Networks - Data Augmentation

Neural networks needs LOTS of data

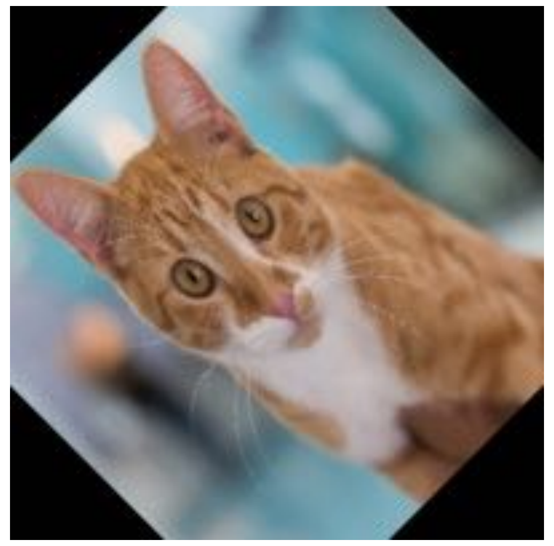
14 million labeled images

Data augmentation can increase the amount of data by adding slightly modified copies of already existing data.

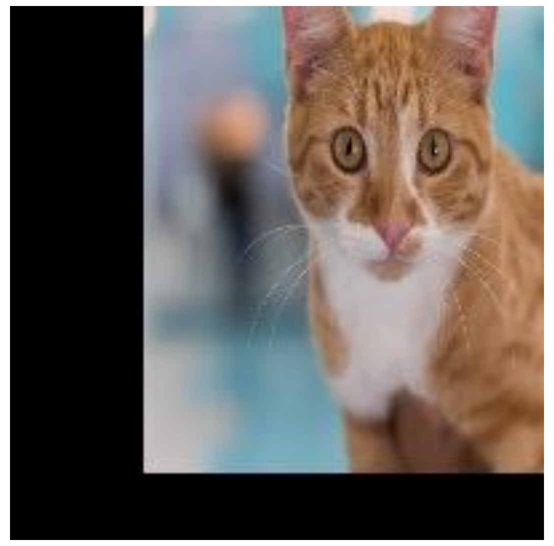
Original



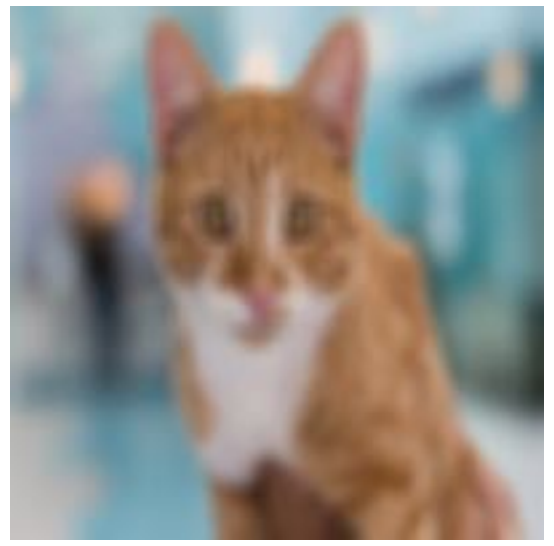
Rotate



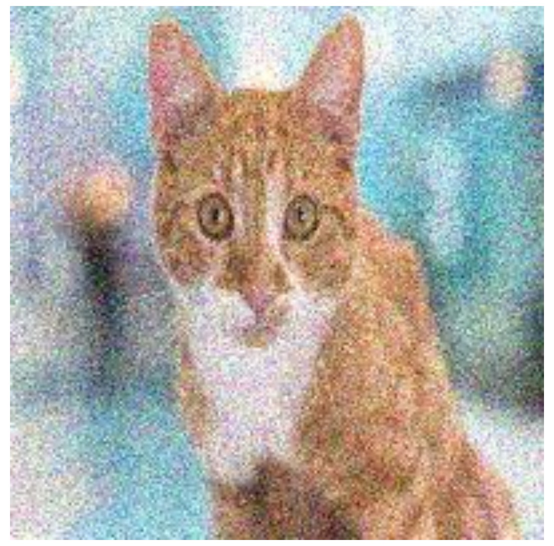
Translate



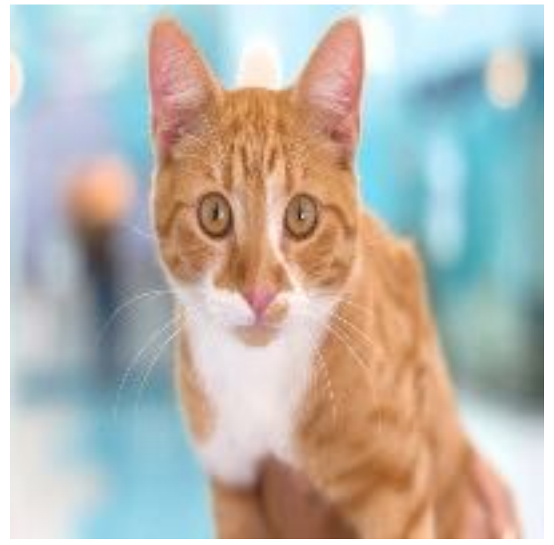
Blur



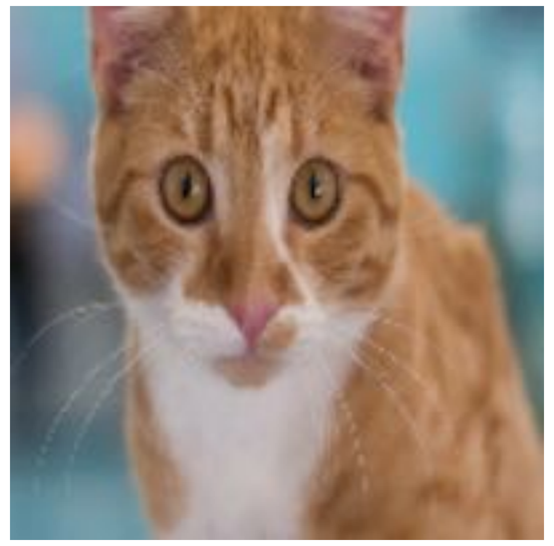
Add Noise



Change Brightness



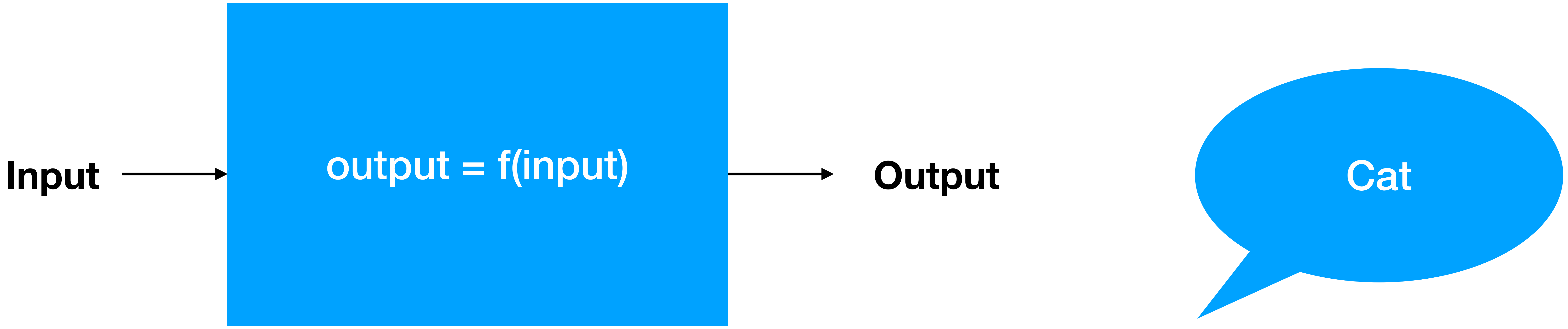
Zoom



Neural Networks

So how does this work?

Input



Input → output = f(input) → **Output**

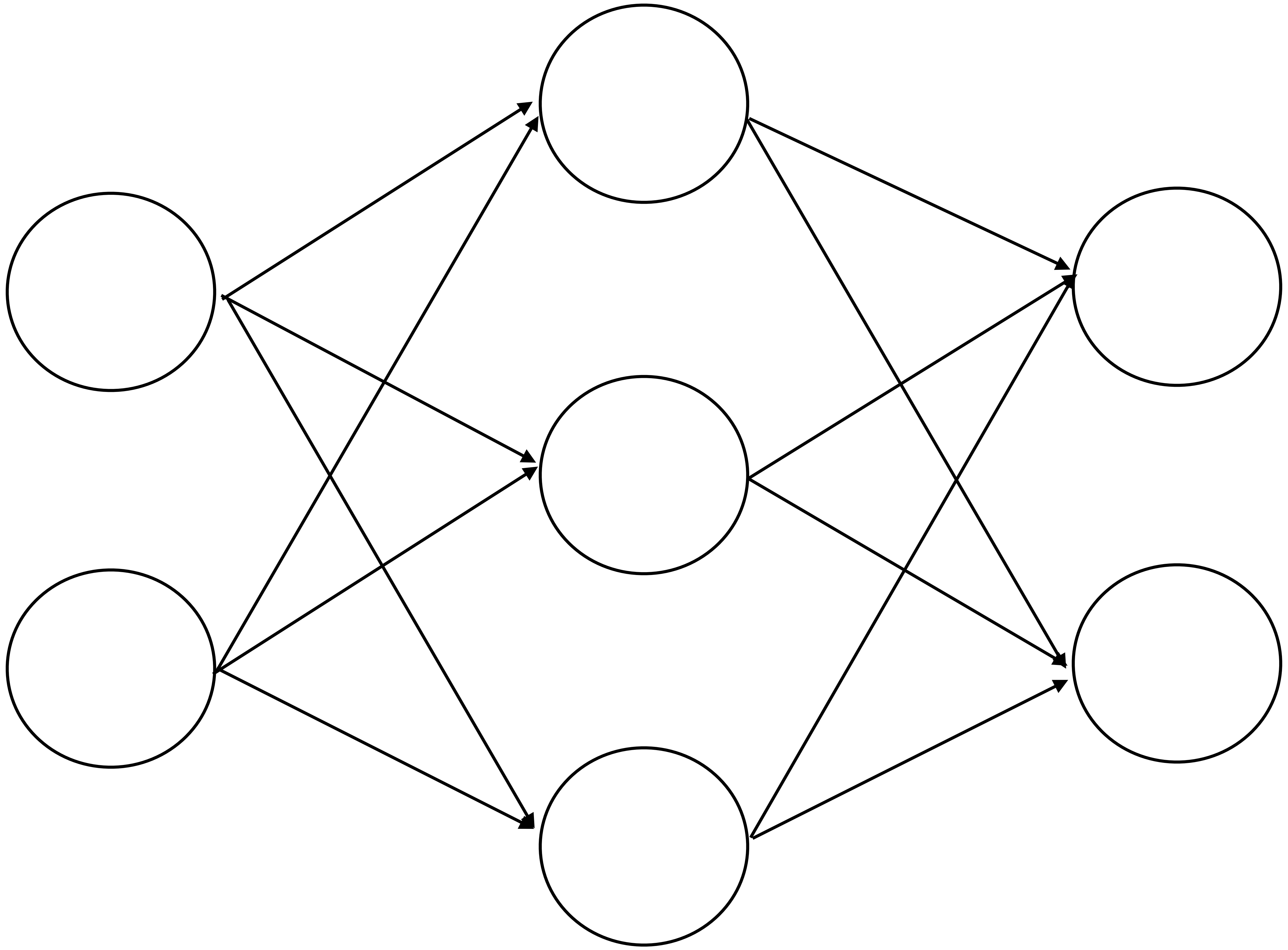
Cat

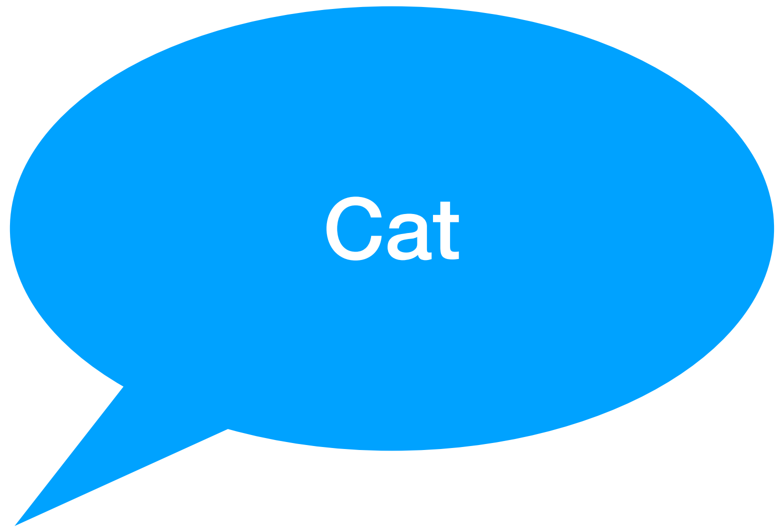
Output

Neural Networks

Input

Input Layer **Hidden Layer** **Output Layer**





Output

Neuron

Output of a Neuron:

$$y = \sigma(w^T x + b)$$

$y = \text{output}$

$\sigma = \text{activation function}$

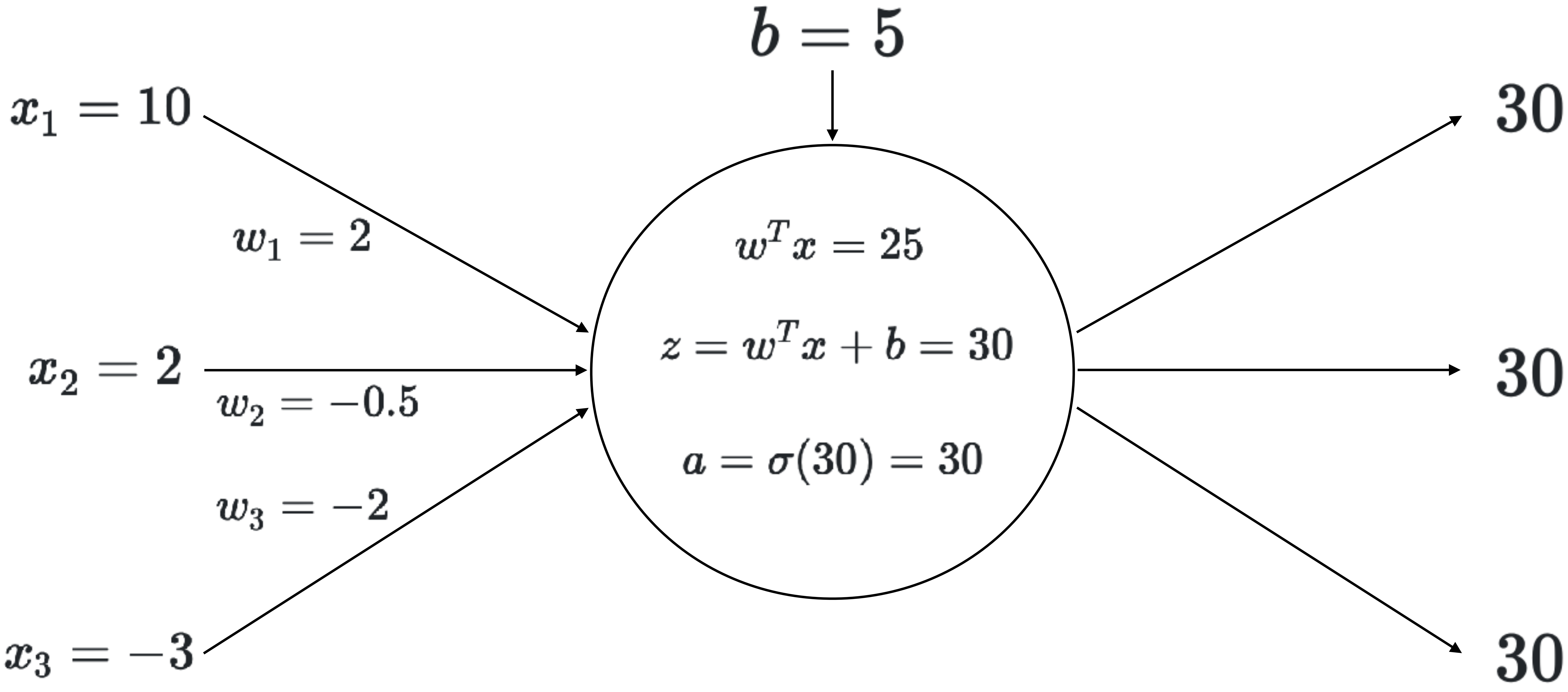
$w = \text{weights}$

$x = \text{input}$

$b = \text{bias}$

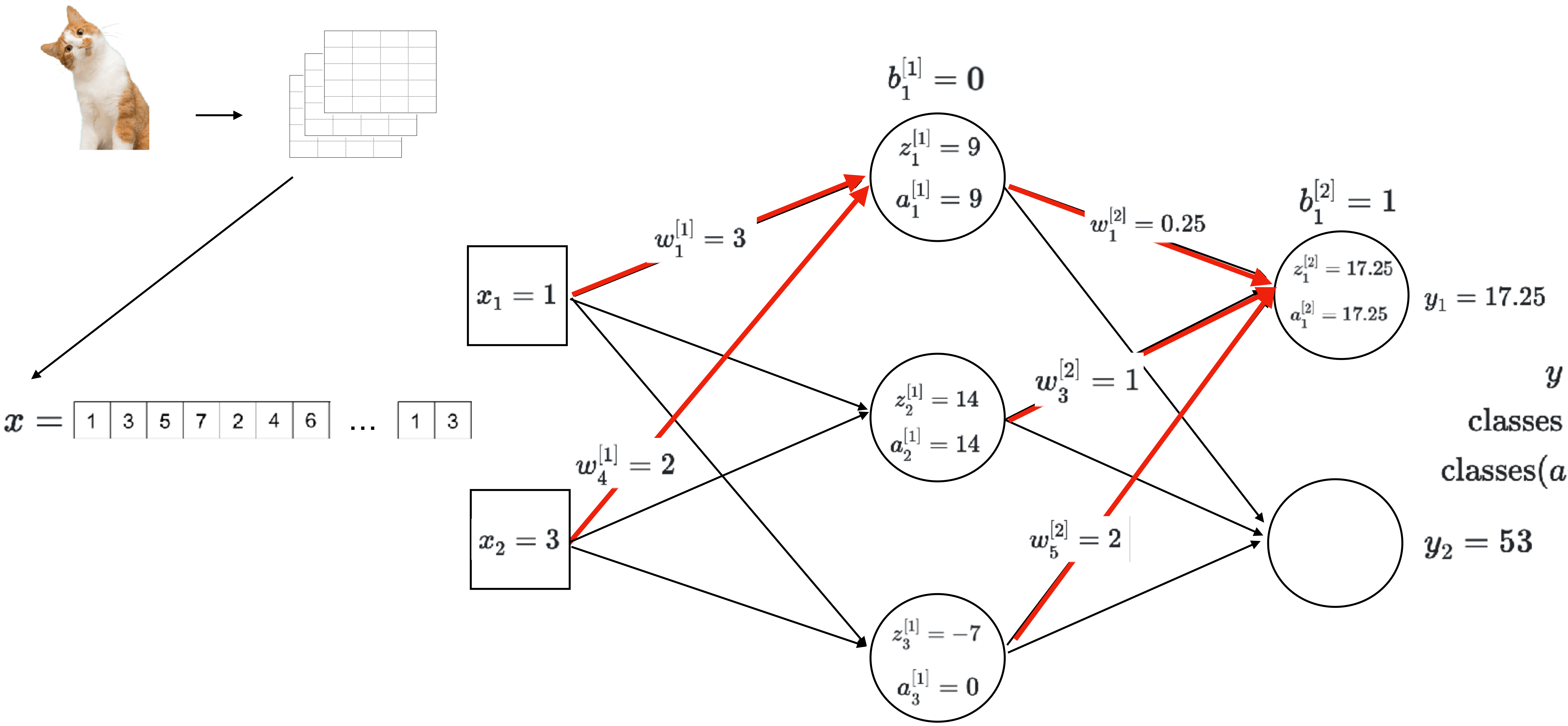
Activation Function (ReLu)

$$\sigma(z) = \begin{cases} 0 & z < 0 \\ z & z \geq 0 \end{cases}$$



$$w^T x = 10 \times 2 + 2 \times -0.5 + -3 \times -2 = 25$$

Neural Networks - Feedforward



$x =$ 1 3 5 7 2 4 6 … 1 3

$x_1 = 1$

$x_2 = 3$

$w_1^{[1]} = 3$

$w_4^{[1]} = 2$

$b_1^{[1]} = 0$

$z_1^{[1]} = 9$

$a_1^{[1]} = 9$

$z_2^{[1]} = 14$

$a_2^{[1]} = 14$

$z_3^{[1]} = -7$

$a_3^{[1]} = 0$

$w_1^{[2]} = 0.25$

$w_3^{[2]} = 1$

$w_5^{[2]} = 2$

$b_1^{[2]} = 1$

$z_1^{[2]} = 17.25$

$a_1^{[2]} = 17.25$

$y_1 = 17.25$

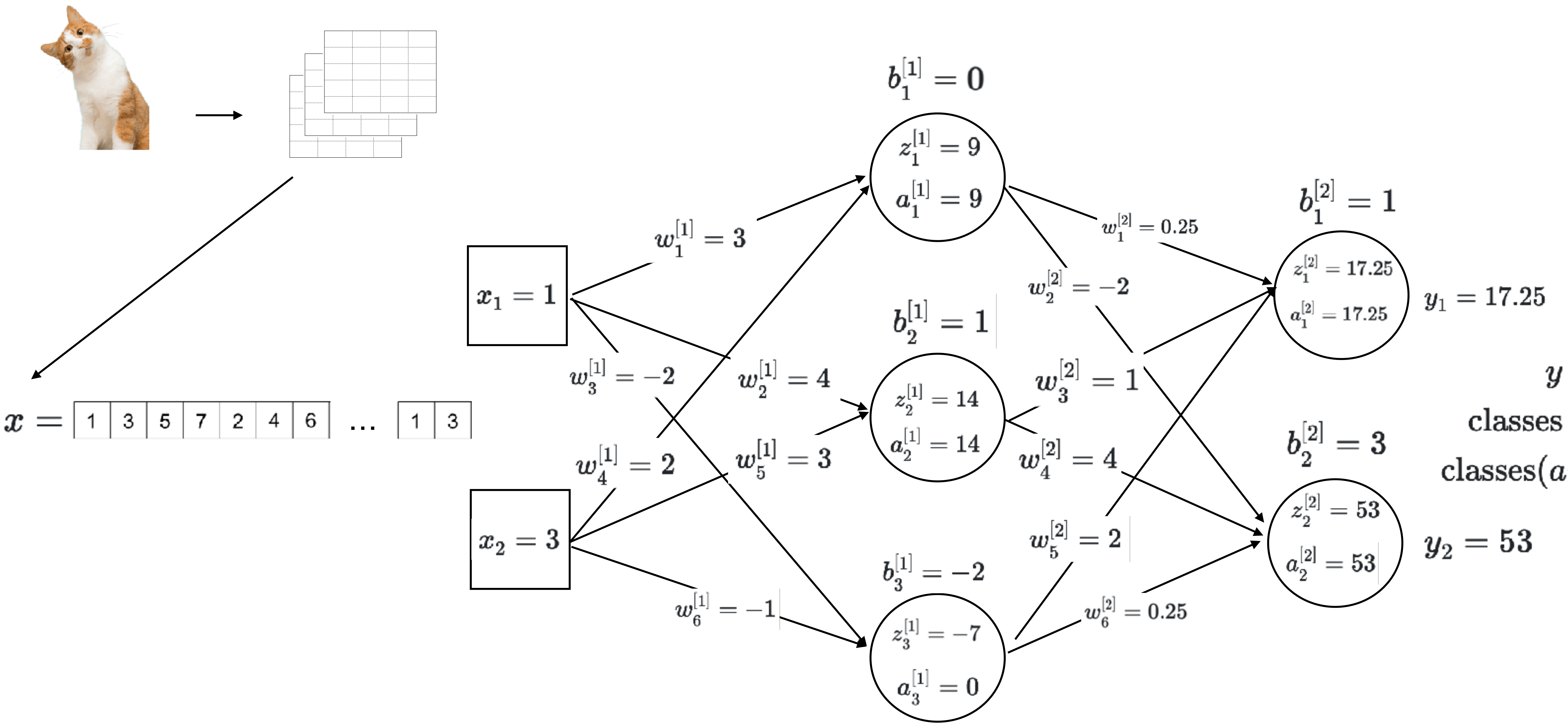
$y_2 = 53$

$$y = [17.25, 53]$$

$$\text{classes} = [\text{“dog ”}, \text{“cat ”}]$$

$$\text{classes}(argmax(y)) = \text{“cat ”}$$

Neural Networks - Feedforward



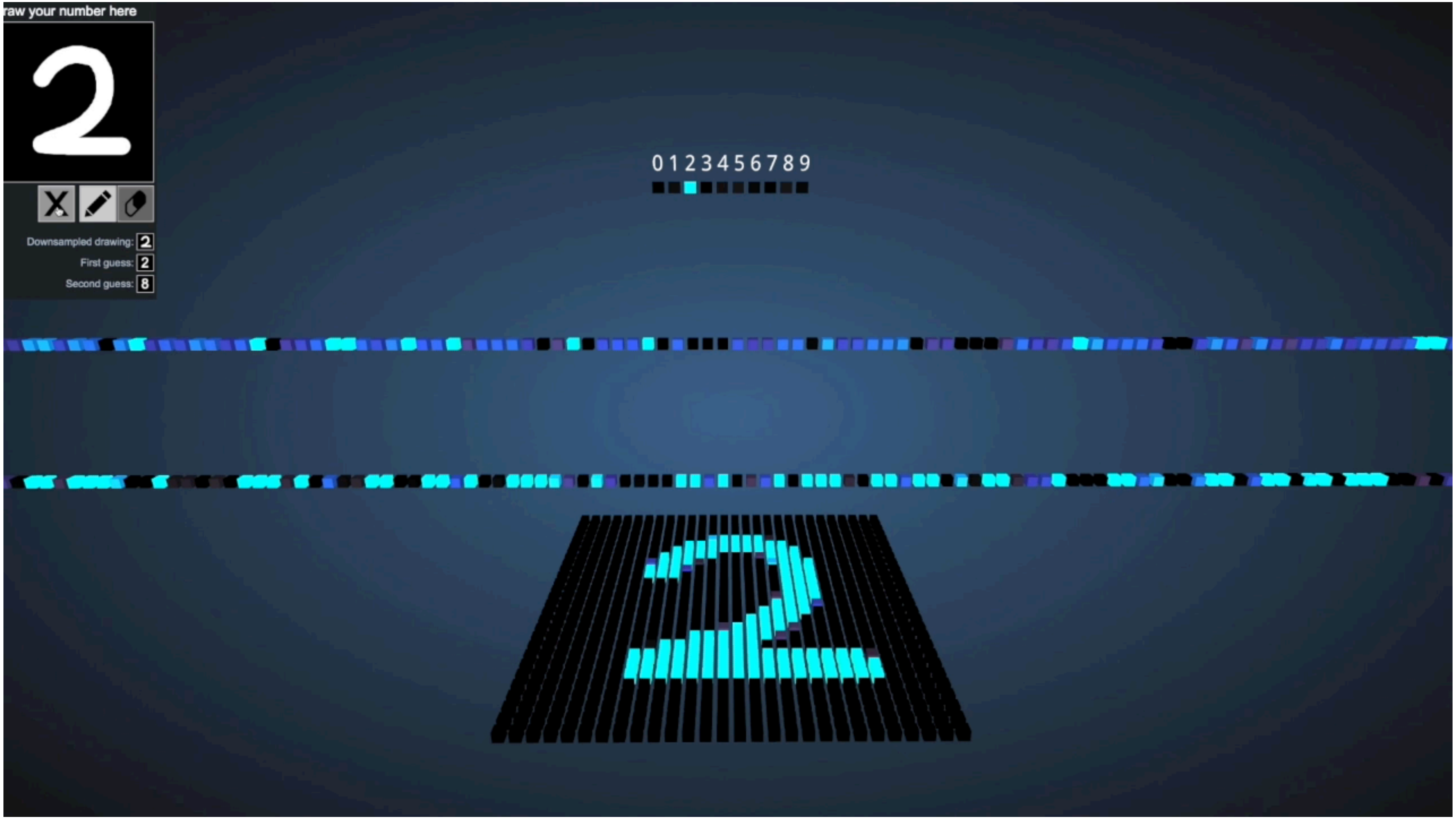
$x = $ | 1 | 3 | 5 | 7 | 2 | 4 | 6 | … | 1 | 3 |

$$y = [17.25, 53]$$

$$\text{classes} = [\text{``dog ''}, \text{``cat ''}]$$

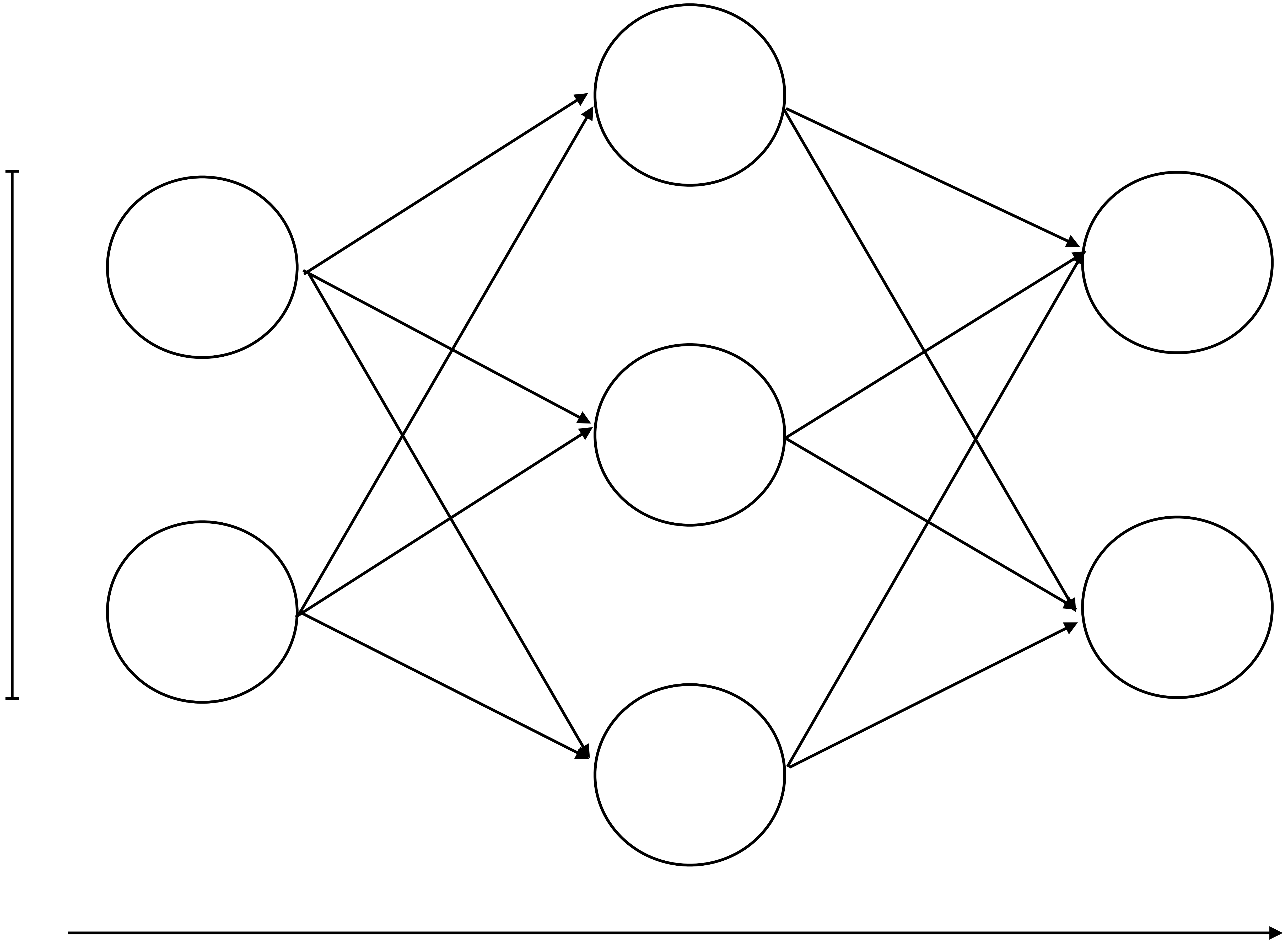
$$\text{classes}(argmax(y)) = \text{``cat ''}$$

Neural Networks - Visualization



Source: https://www.cs.ryerson.ca/~aharley/vis/fc/

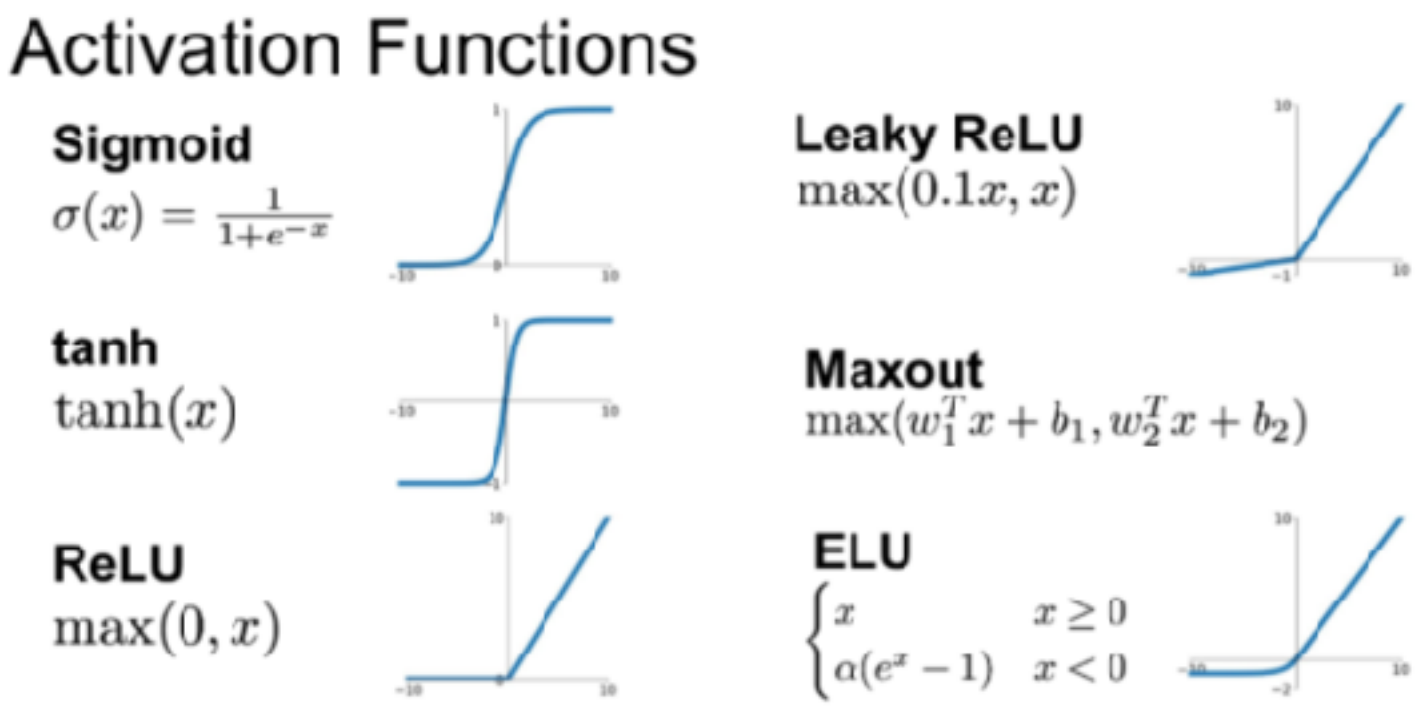
Neural Networks - Structure



Input Size

Number of Layers

Activation functions

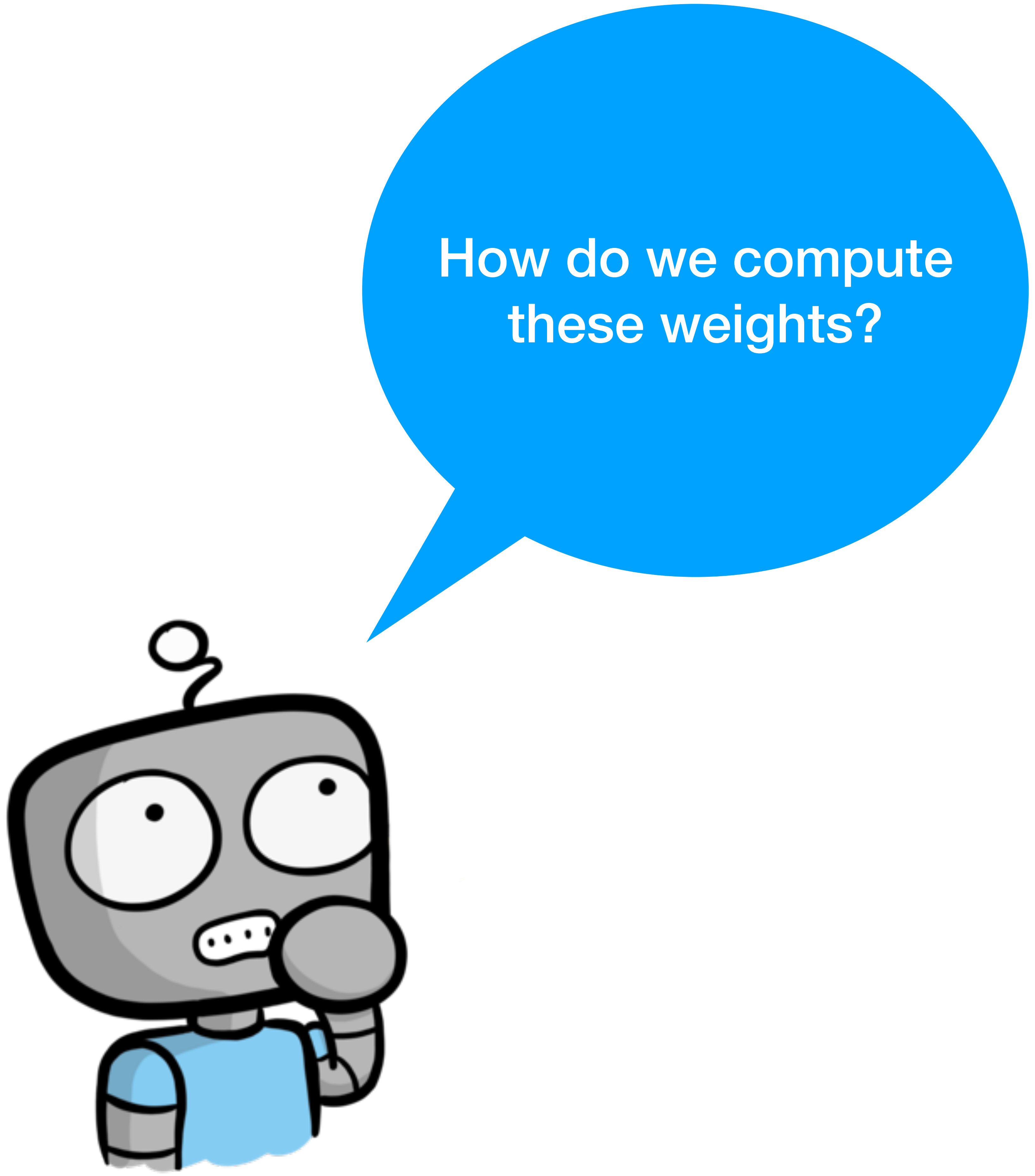


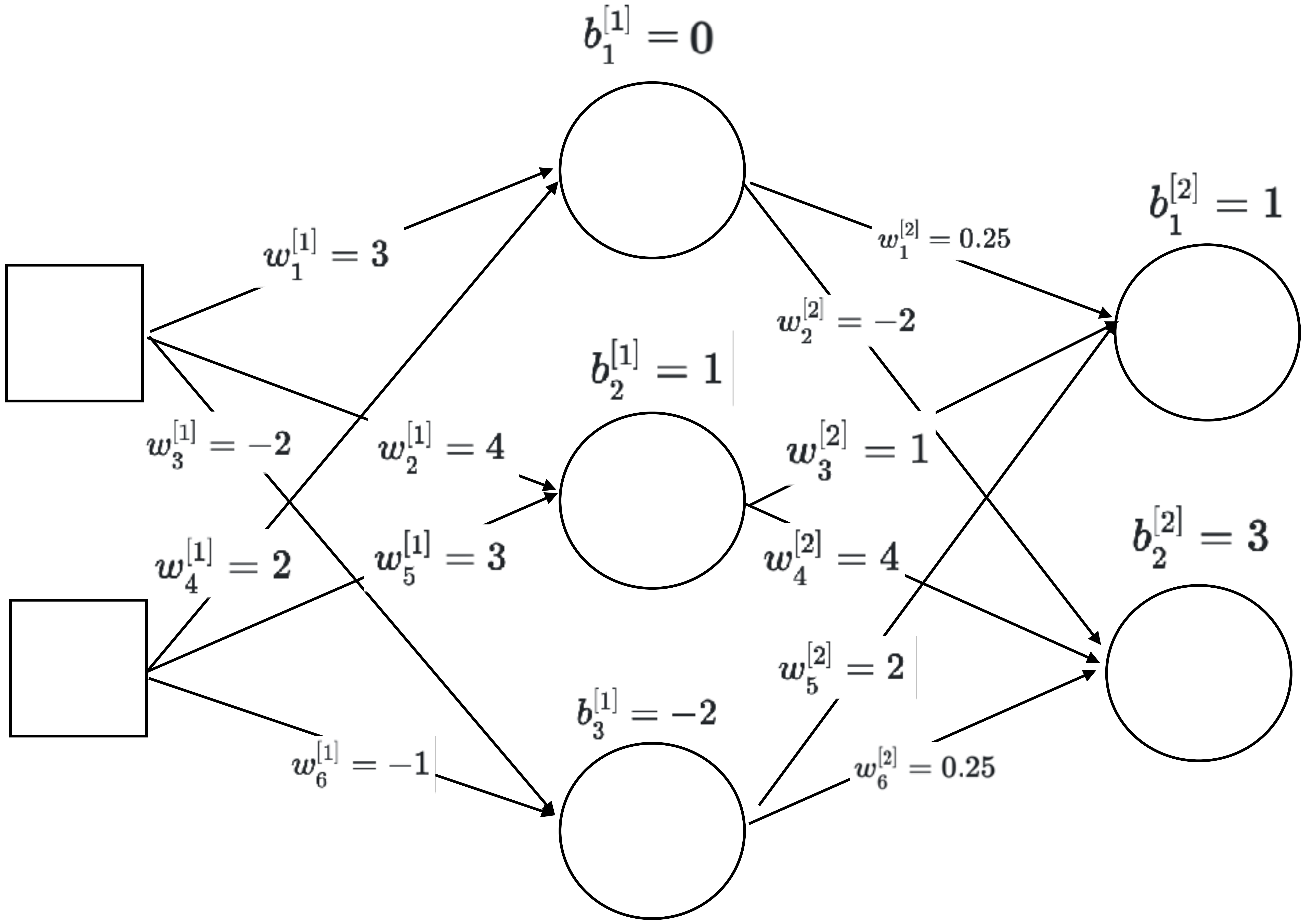
Source: Shruti Jadon

Learning Parameters:

- Learning rate
- Optimizer
- Batch Size
- Early stopping
- Number training epochs

Neural Networks - Weights





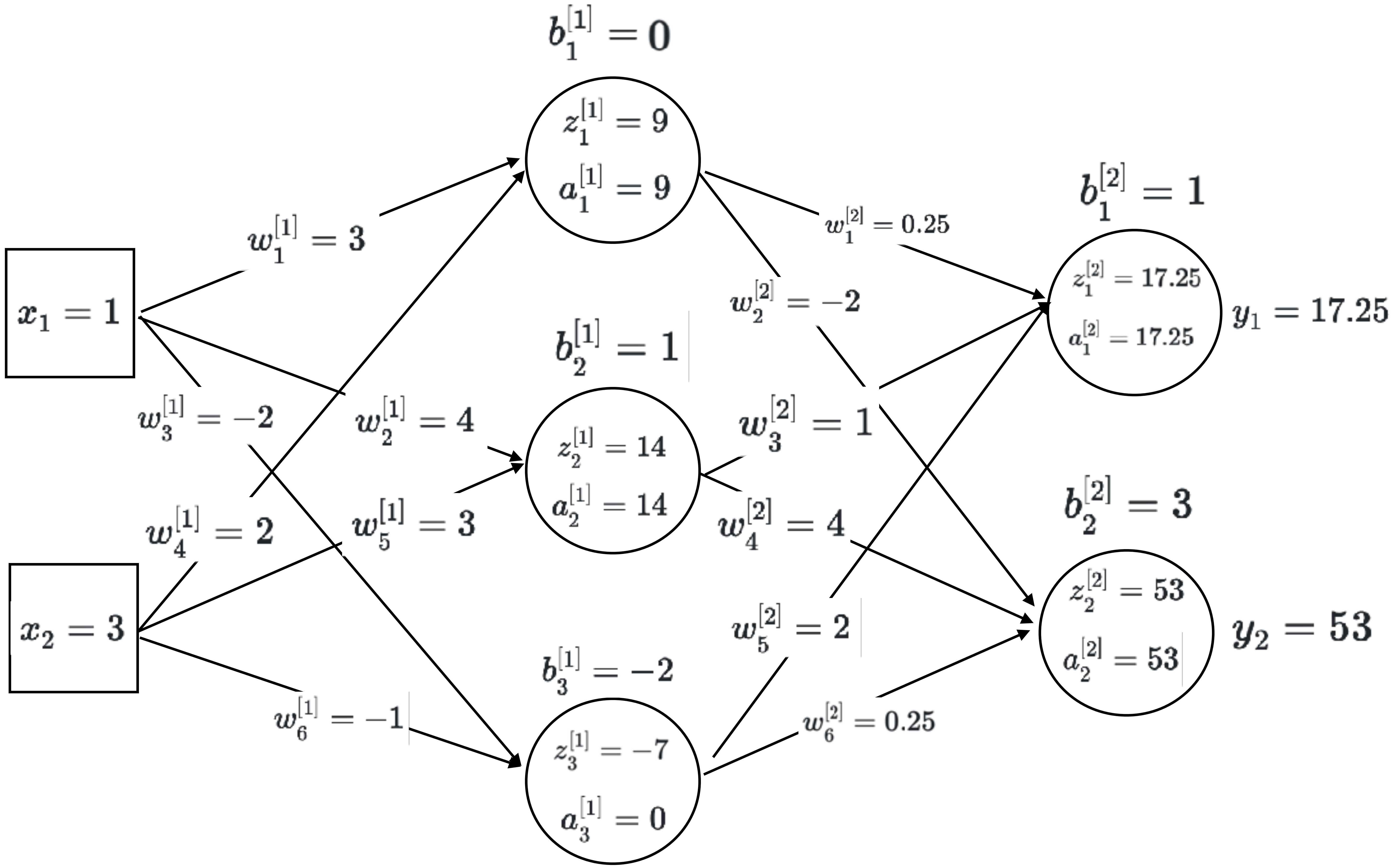
Neural Networks - Updating Weights

Computing networks weights:
1) For each observation in training set:
2) Feedforward the observation ✓
3) Compute error
4) Run gradient descent to update weights

Input Data

$$\mathbf{y}' = [0, 1]$$

Output Label



Neural Networks - Prediction Error

Computing networks weights:

1) For each observation in training set:

2) Feedforward the observation ✓

3) Compute error?

4) Run gradient descent to update weights

Prediction Error:

Input Data

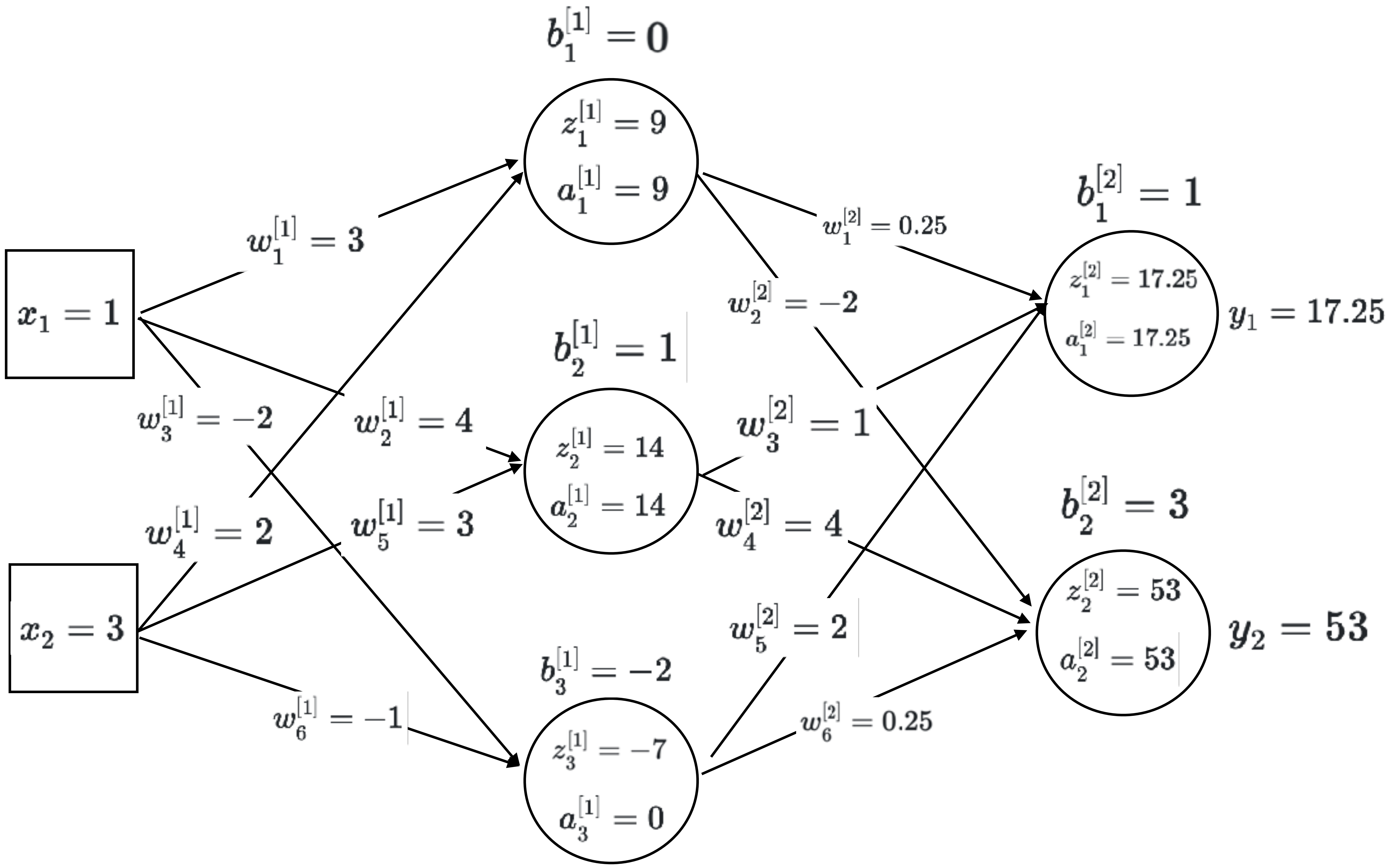
$$\mathbf{y}' = [0, 1]$$

Output Label

Mean squared error:

$$error = \sum \frac{1}{2}(\mathbf{y}' - \mathbf{y})^2$$

Also known as the cost function



Neural Networks - Prediction Error

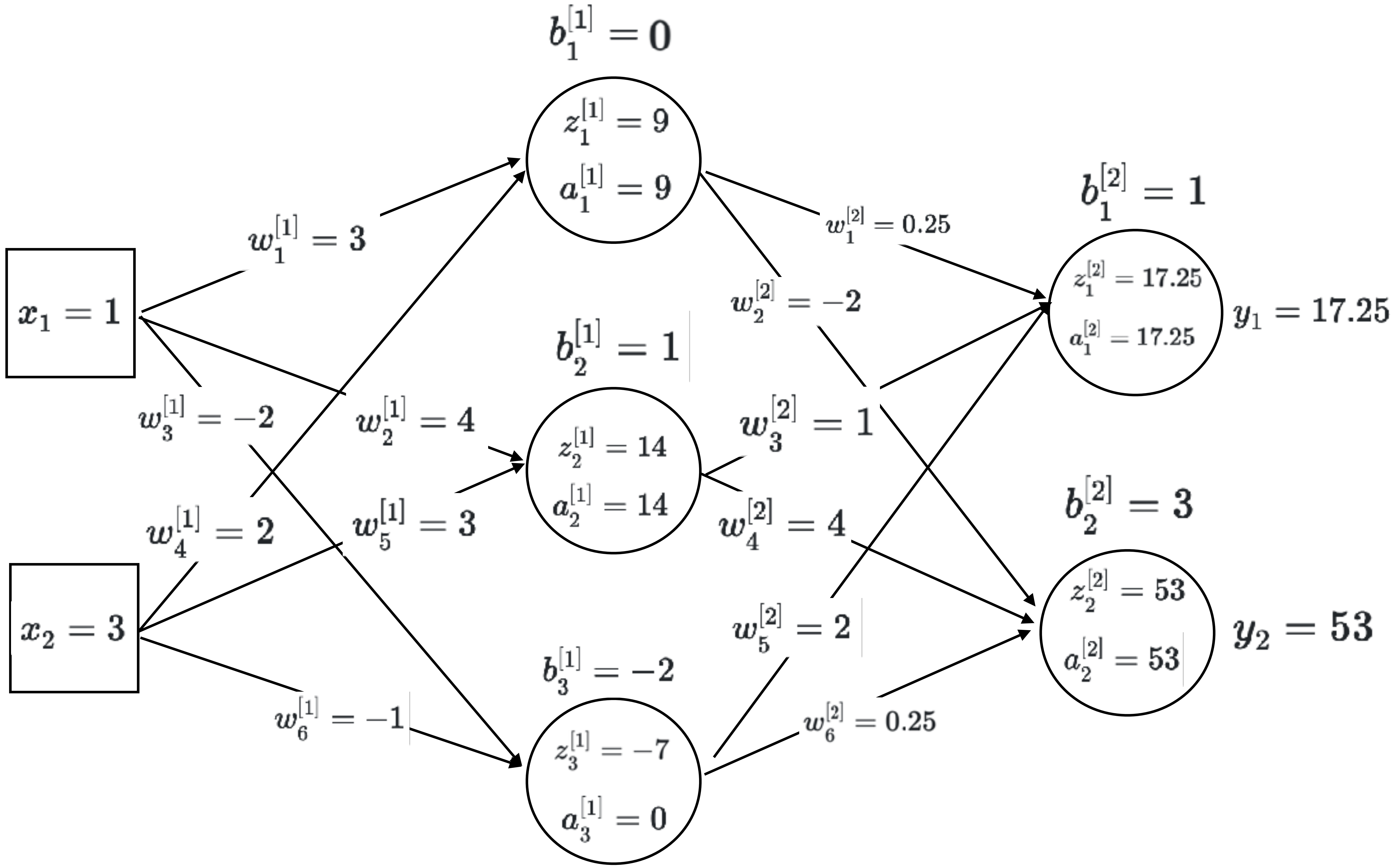
Computing networks weights:

1) For each observation in training set:

2) Feedforward the observation ✓

3) Compute error?

4) Run gradient descent to update weights



Prediction Error:

Input Data

$$\mathbf{y}' = [0, 1]$$

Output Label

$$error = \sum \frac{1}{2}(\mathbf{y}' - \mathbf{y})^2$$

$$error = \frac{1}{2}(y_1' - y_1)^2 + \frac{1}{2}(y_2' - y_2)^2$$

$$error = \frac{1}{2}(0 - 17.25)^2 + \frac{1}{2}(1 - 53)^2$$

$$error = 1500.7813$$

Neural Networks - Gradient Descent

Computing networks weights:
1) For each observation in training set:
2) Feedforward the observation ✓
3) Compute error ✓
4) Run gradient descent to update weights ?

Gradient descent:

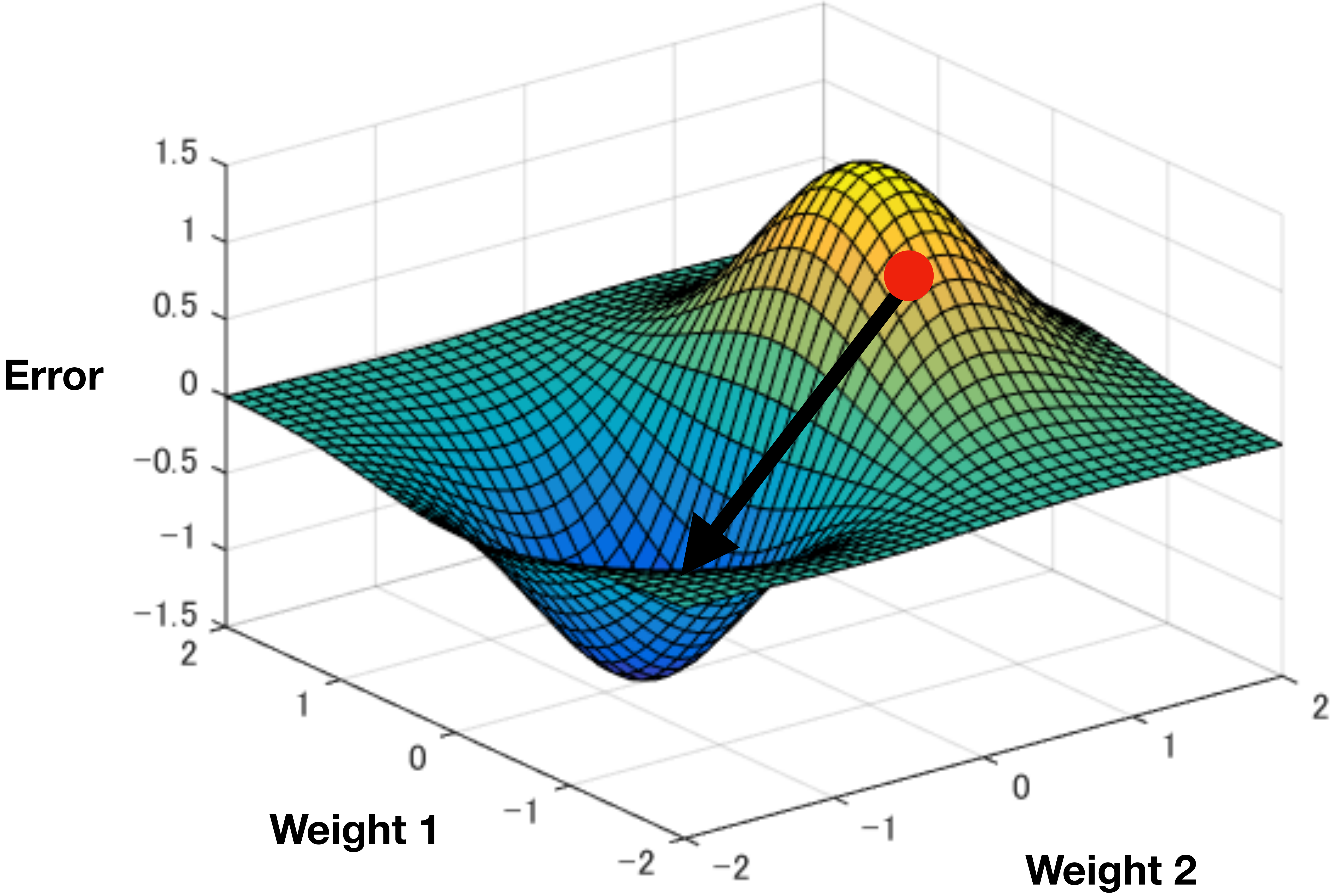
$$error = \sum \frac{1}{2}(\mathbf{y}' - \mathbf{y})^2$$

↑

Function of the weights

To minimize the error, we can change the weights

$$w_k = w_k - \eta \left(\frac{\partial error}{\partial w_k} \right)$$



Neural Networks - Gradient Descent

Computing networks weights:

1) For each observation in training set:

2) Feedforward the observation ✓

3) Compute error ✓

4) Run gradient descent to update weights ?

$$error = 1500.7813$$

Gradient descent:

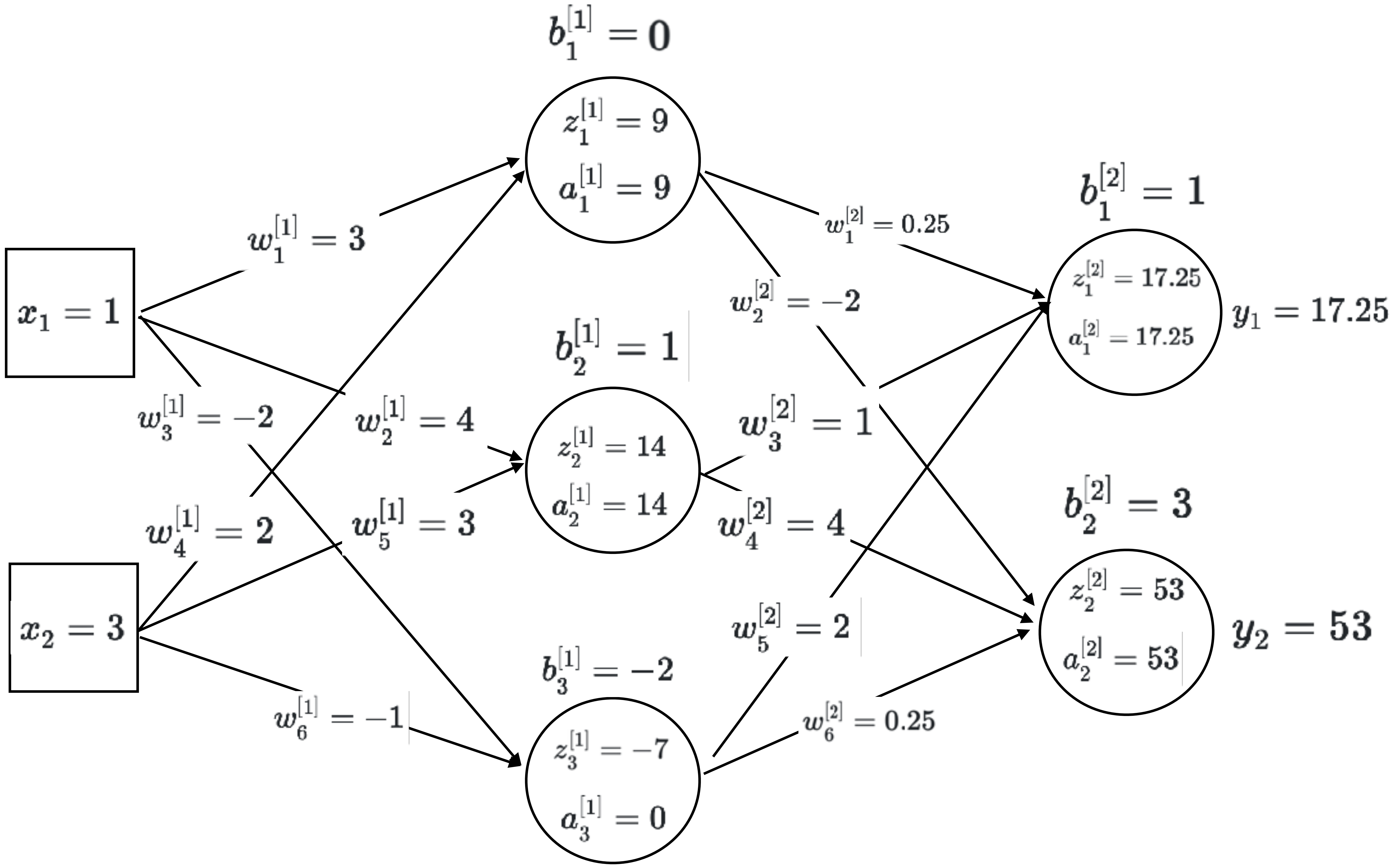
Goal: Update the weights

$$error = \sum \frac{1}{2}(\mathbf{y}' - \mathbf{y})^2$$

$$w_k = w_k - \eta\left(\frac{\partial error}{\partial w_k}\right)$$

Chain Rule

$$\frac{\partial error}{\partial w_k} = \frac{\partial error}{\partial y_1} \times \frac{\partial y1}{\partial z_1^{[2]}} \times \frac{\partial z_1^{[2]}}{\partial w_1^{[2]}}$$



Neural Networks - Gradient Descent

$$\frac{\partial error}{\partial w_k} = \frac{\partial error}{\partial y_1} \times \frac{\partial y1}{\partial z_1^{[2]}} \times \frac{\partial z_1^{[2]}}{\partial w_1^{[2]}}$$

$$error = \frac{1}{2}(y_1' - y_1)^2 + \frac{1}{2}(y_2' - y_2)^2$$

$$\frac{\partial error}{\partial y_1} = 2 \times \frac{1}{2}(y_1' - y_1) \times -1 + 0$$

$$\frac{\partial error}{\partial y_1} = -1 \times (0 - 17.25)$$

$$\frac{\partial error}{\partial y_1} = 17.25$$

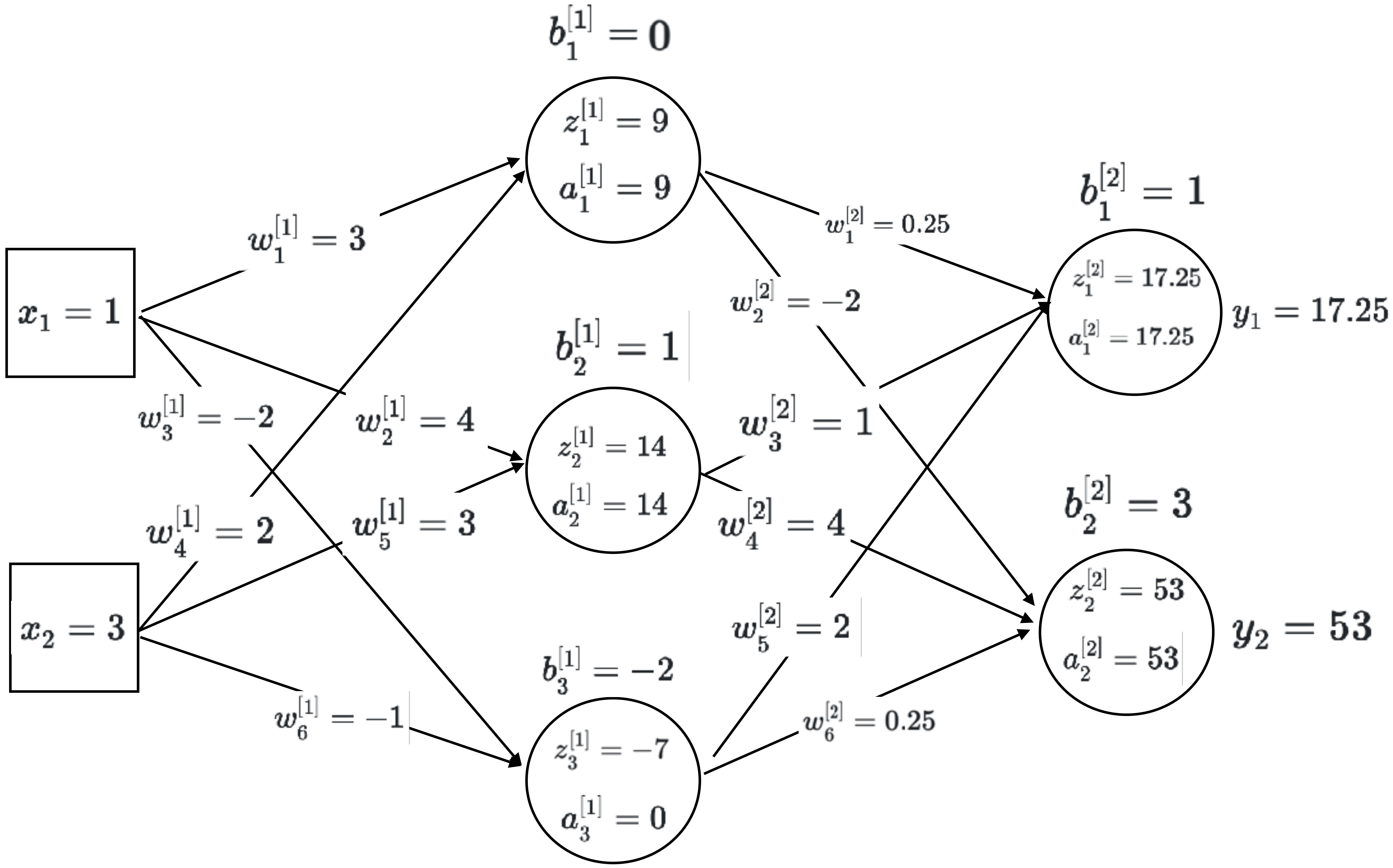
$$y_1 = a_1 = \sigma(z_1^{[2]}) = \begin{cases} 0 & z < 0 \\ z & z \geq 0 \end{cases}$$

$$\frac{\partial y1}{\partial z_1^{[2]}} = |x| = \begin{cases} 0 & z < 0 \\ 1 & z \geq 0 \end{cases}$$

$$\frac{\partial y1}{\partial z_1^{[2]}} = 1$$

$$z_1^{[2]} = a_1^{[1]} \times w_1^{[2]} + a_2^{[1]} \times w_3^{[2]} + a_3^{[1]} \times w_5^{[2]}$$

$$\frac{\partial z_1^{[2]}}{\partial w_1^{[2]}} = a_1^{[1]} = 9$$



Neural Networks - Gradient Descent

$\mathbf{y'} = [0, 1]$

Computing networks weights:
1) For each observation in training set:
2) Feedforward the observation ✓
3) Compute error ✓
4) Run gradient descent to update weights ✓

$$error = 1500.7813$$

Gradient descent:

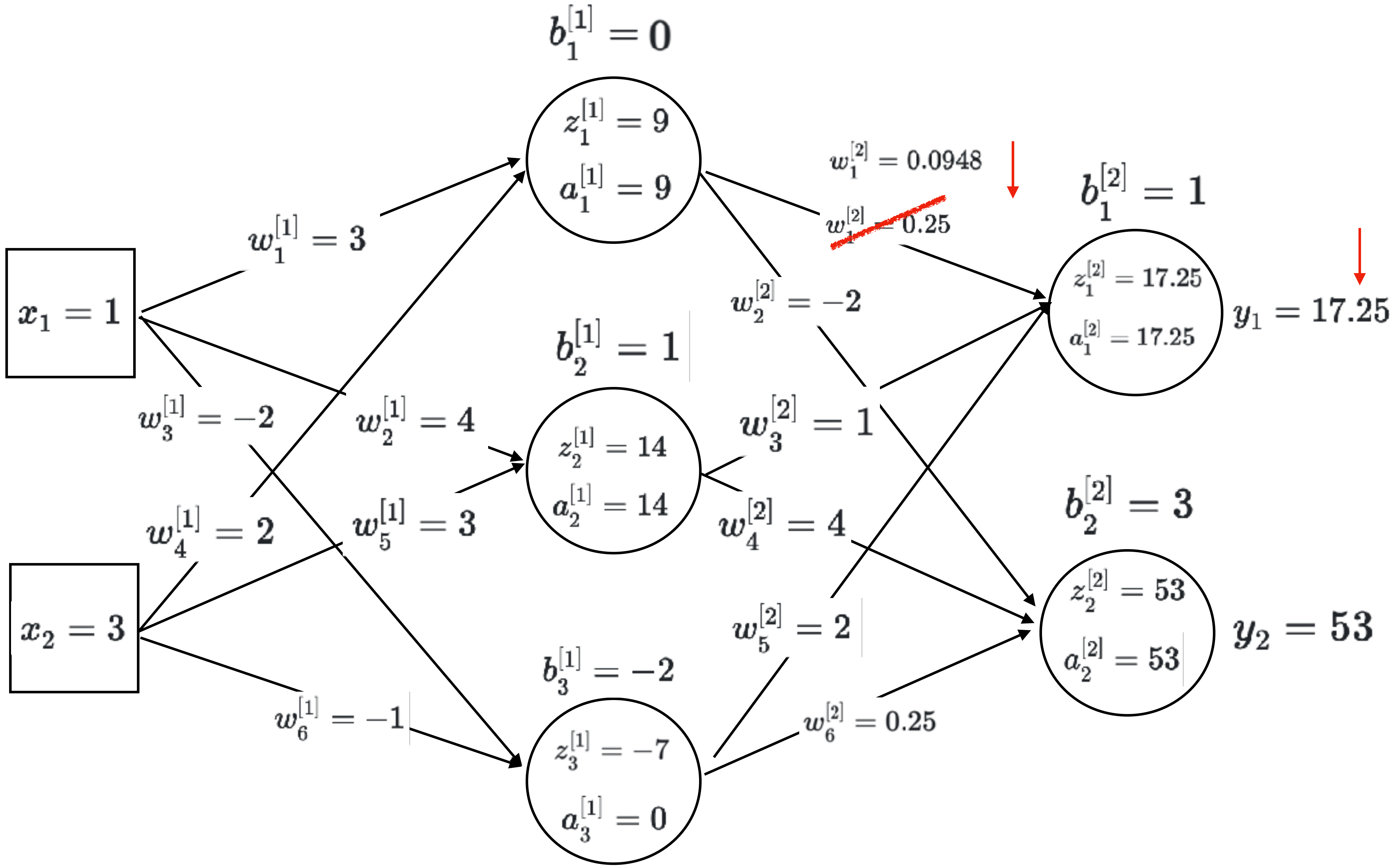
Goal: Update the weights

$$w_k = w_k - \eta\left(\frac{\partial error}{\partial w_k}\right)$$

$$\frac{\partial error}{\partial w_k} = 17.25 \times 1 \times 9 = 155.25$$

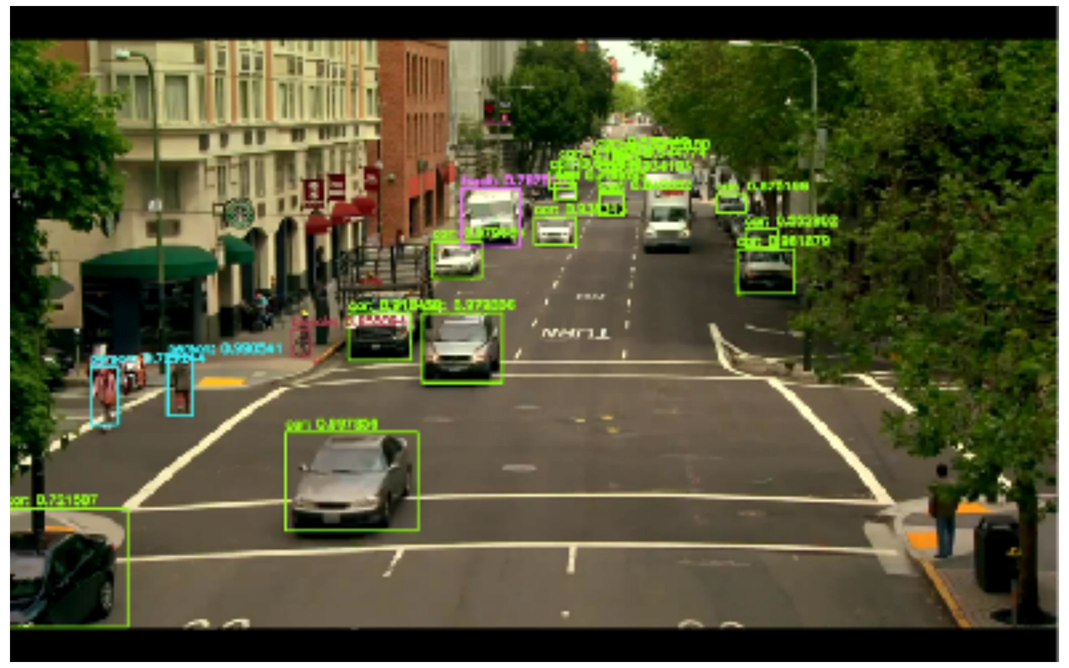
$$\eta = \text{learning rate} = 0.001$$

$$w_k = 0.25 - 0.001\,(155.25) = 0.0948$$



Neural Networks

Classification

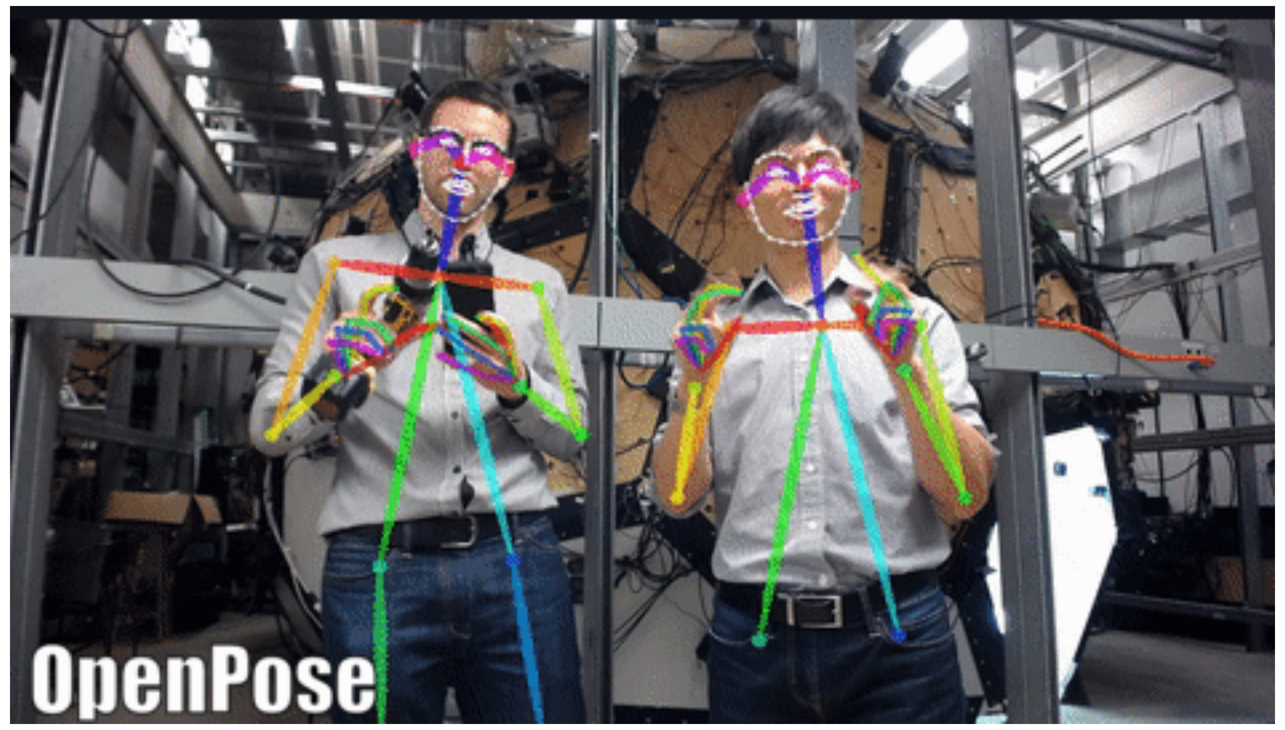


Yolov3: https://pjreddie.com/darknet/yolo/

MiconNet: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8481688

Object Detection

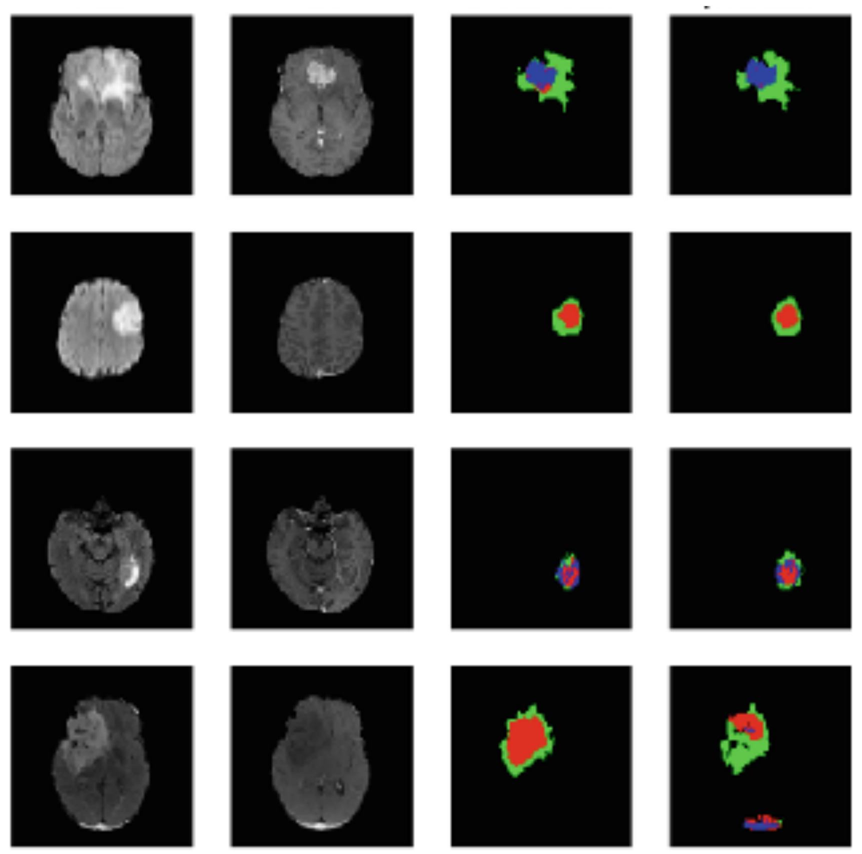
Source: https://www.nvidia.com/en-us/self-driving-cars/drive-videos



Openpose: https://github.com/CMU-Perceptual-Computing-Lab/openpose

Image Segmentation

NVIDIA Redtail: https://github.com/NVIDIA-AI-IOT/redtail



S3D-UNet: https://link.springer.com/chapter/10.1007/978-3-030-11726-9_32

Perception Algorithms

Perception estimates the state of the environment

Image Processing Algorithms
An image is processed through transformations, filters, or algorithms. We can then use this information to infer something about that image.
Key Difference: We define this function

Pros:
Does not require huge labeled datasets
Are easier to interpret by humans
Does not require heavy computation resources

Cons:
Encode relatively simple functions

Machine Learning
Gather large amounts of data and use this data to learn or approximate the desired function. We can then use this information to infer something about that image.
Key Difference: We learn this function

Pros:
Improves with more data
Can learn complicated functions
Can be used as an end-to-end an solution

Cons:
Requires huge labeled datasets
Requires heavy computation resources to train
Difficult to interpret what they have learned
Not robust to scenarios outside its training data

Research