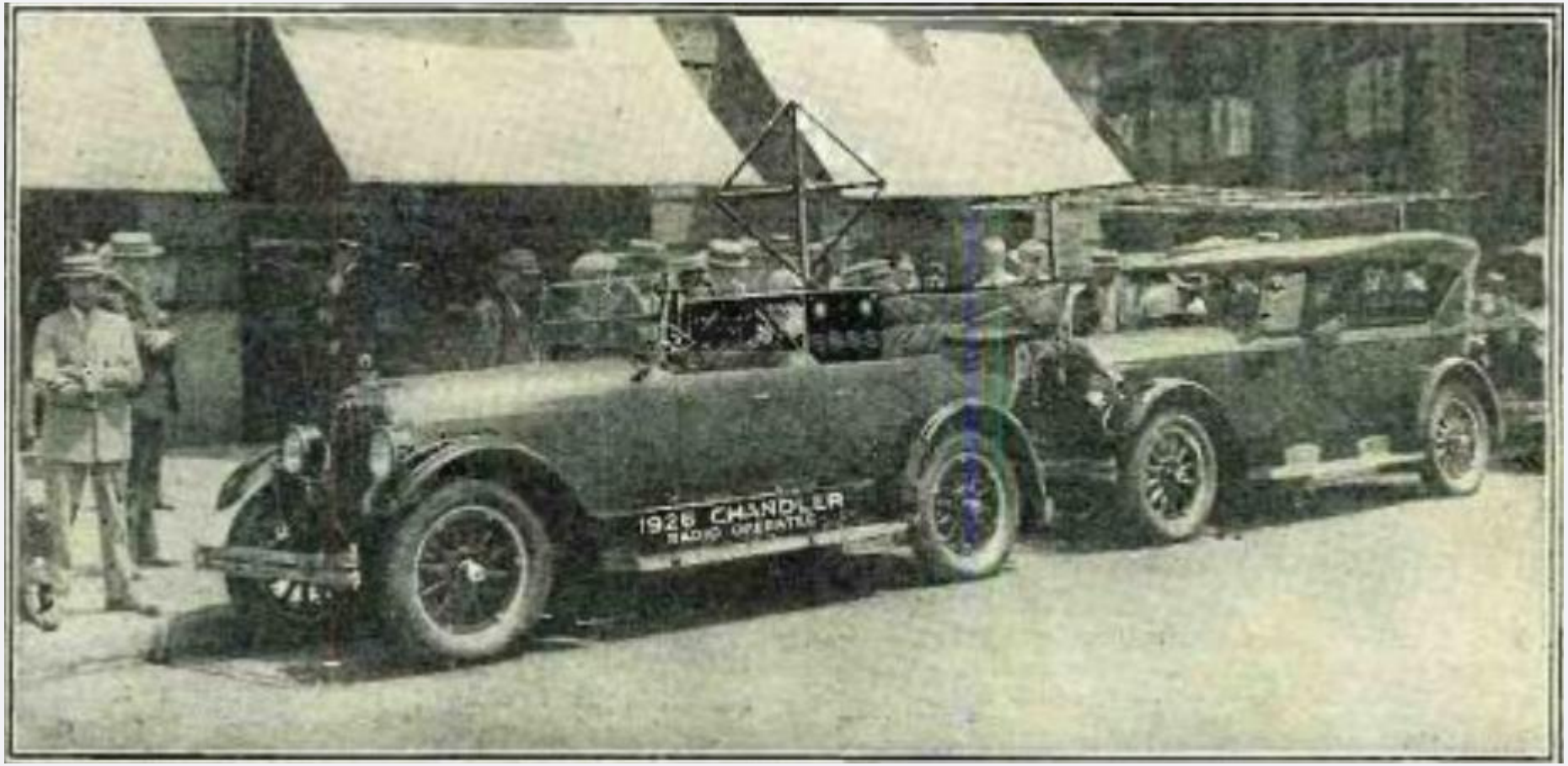
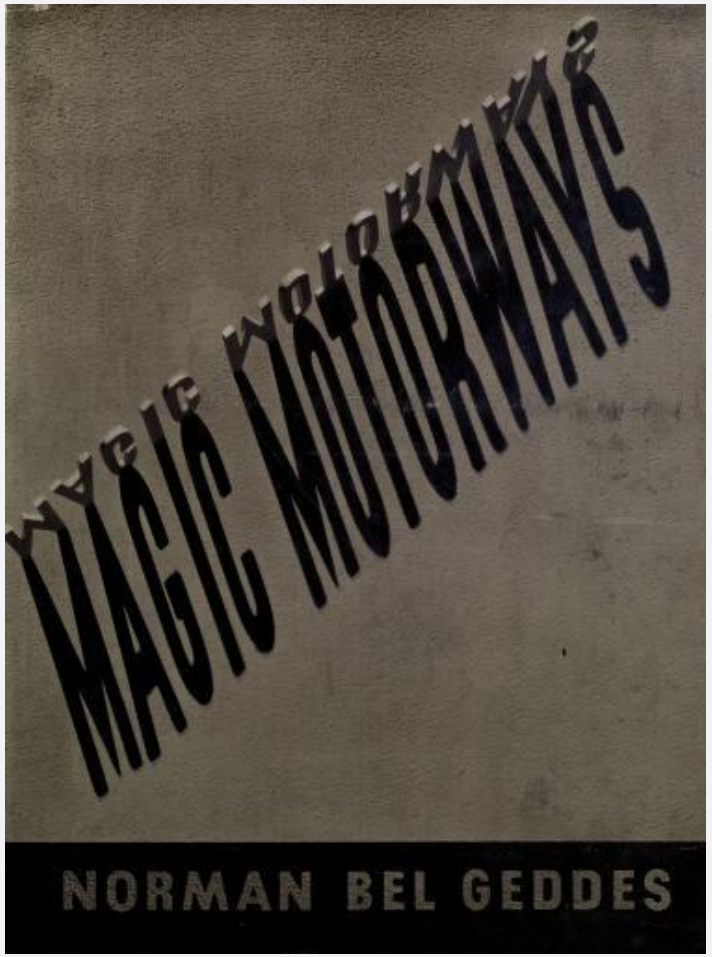
autonomous cars: deep learning and computer vision in python

history of self-driving cars

1925!

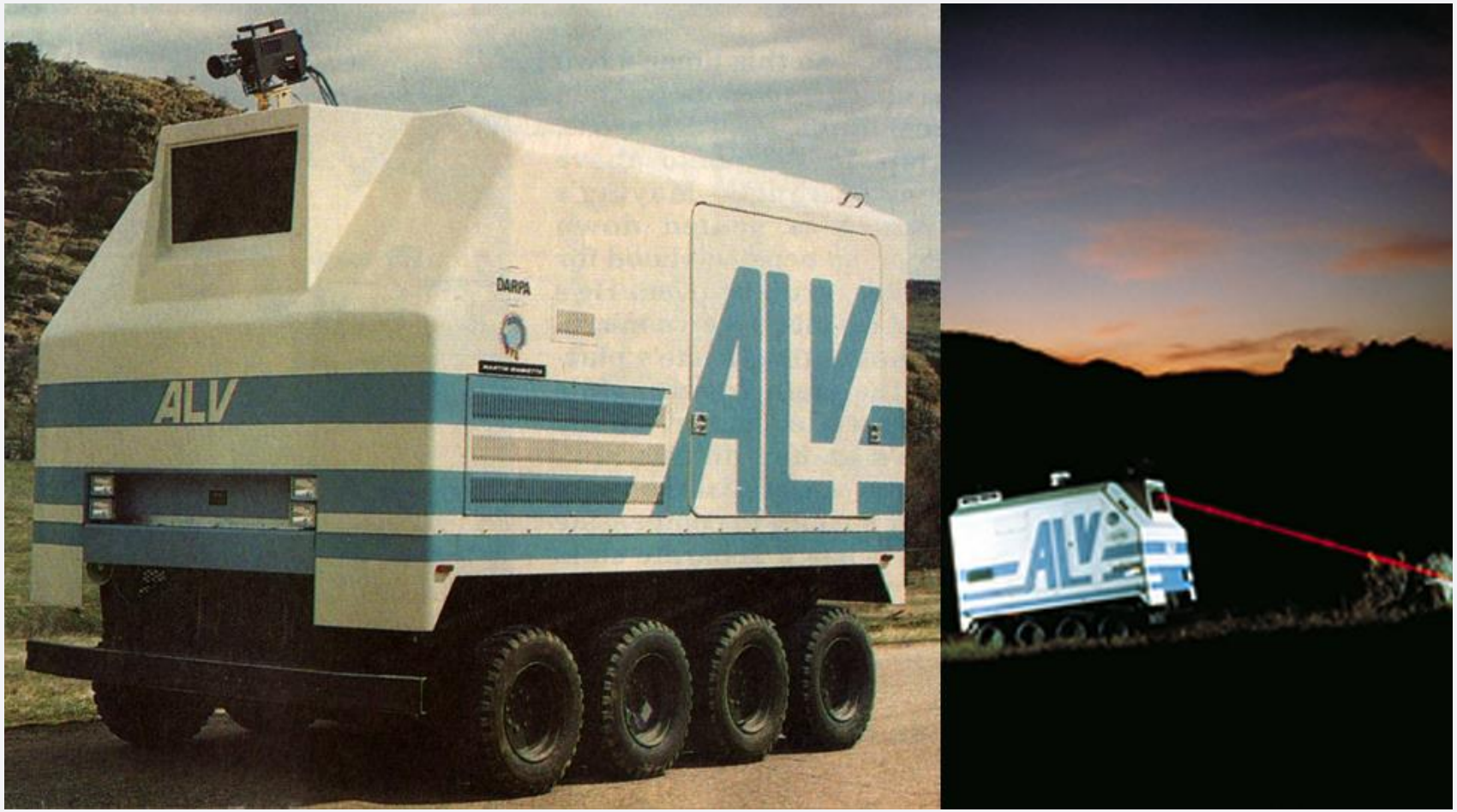


1939



late 50’s / early 60’s

late 80’s : alv project

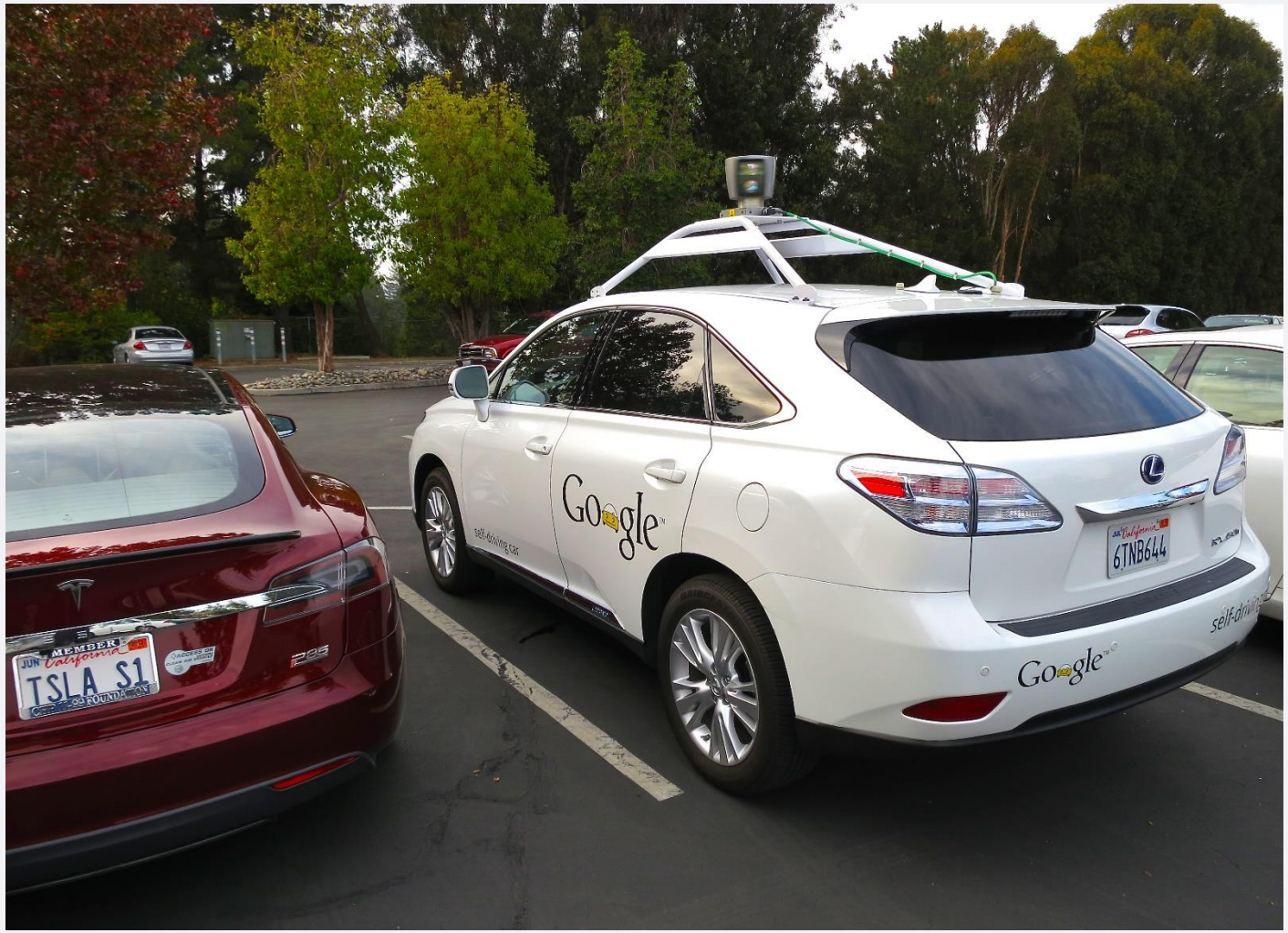


90’s: Demo ‘97


MODE
FULL AUTOMATION
INFO
DEST
GAP
ACTUAL
12 ft
TARGET
12 ft
DEST
AHS DEMO ENDPOINT
DISTANCE
6.9 miles
TIME TO DEST
00:08:00
EXIT

00’s: DARPA grand challenges

today



course overview

course outline

- python crash course
- computer vision basics (in 3 parts)
  - detecting lane lines
  - convolution (sharpening, blurring)
  - edge detection
  - image transformation
  - region of interest masking
  - hough transform theory (feature extraction)
  - real-time lane line detection
  - template matching
  - corner detection
  - image pyramiding
  - feature matching
    - sift, surf, fast, orb
    - histogram of oriented gradients (hog)
  - camera calibration, distortion correction

course outline

Machine learning (2 parts)
- cross-validation
- linear & logistic regression
- decision trees
- naïve bayes
- support vector machines

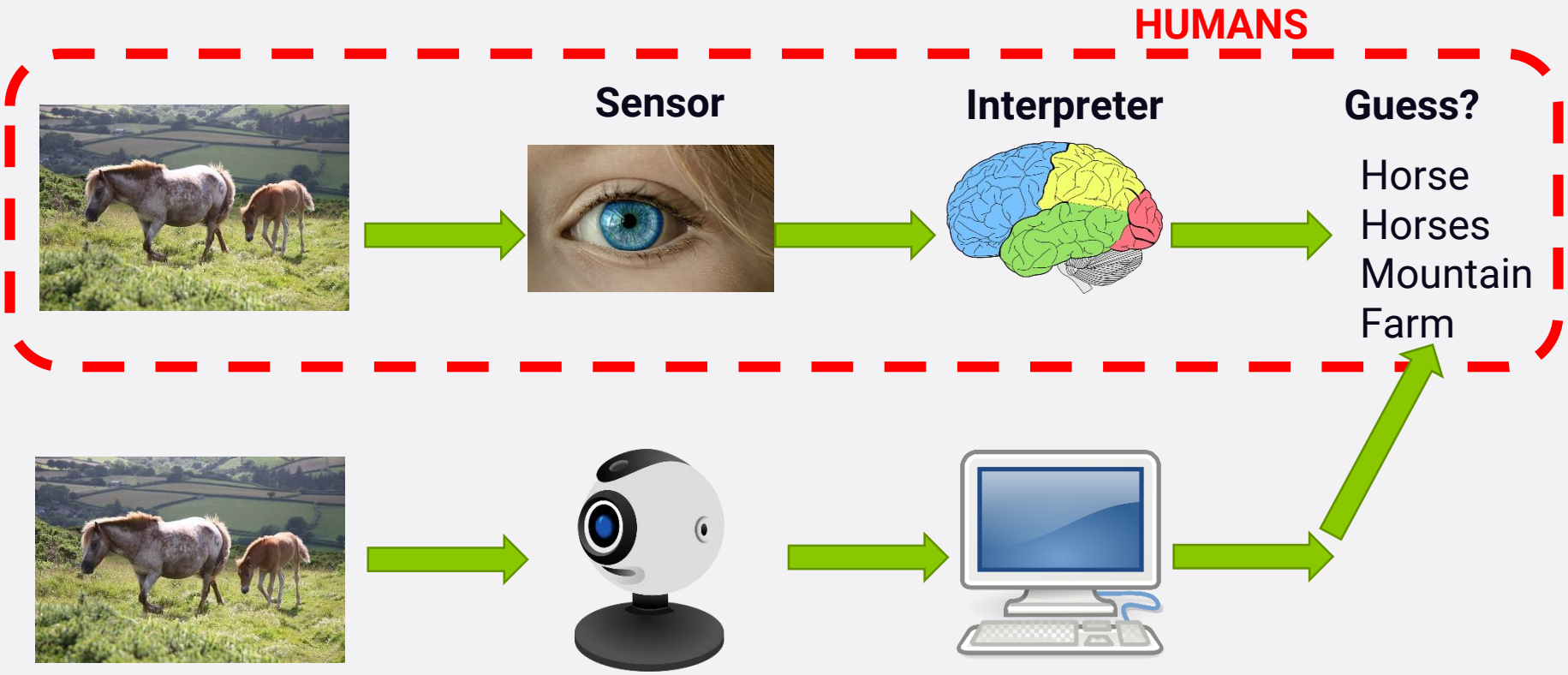
Artificial neural networks
- perceptrons
- multi-layer perceptrons
- activation and error functions
- gradient descent
- backpropagation

course outline

deep learning
- tensorflow
- dropout regularization
- convolutional neural networks
- LeNET CNN’s (classify traffic signs!)

INTRODUCTION
WHAT IS COMPUTER VISION?

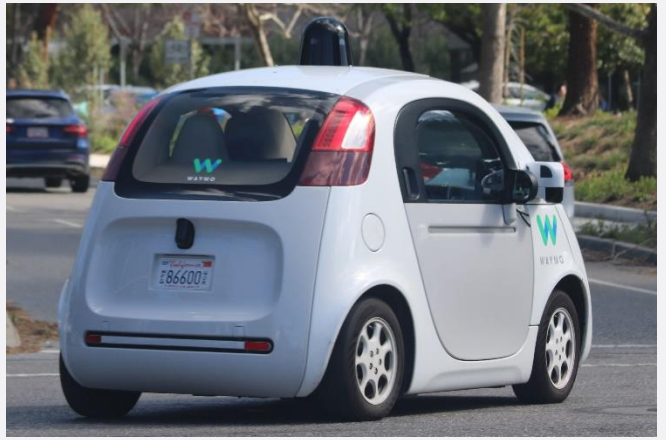
- Science that allows computers to understand images and video and determine what the computer "sees" or "recognizes".

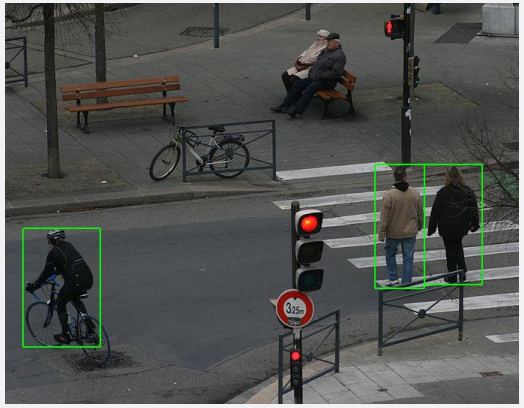


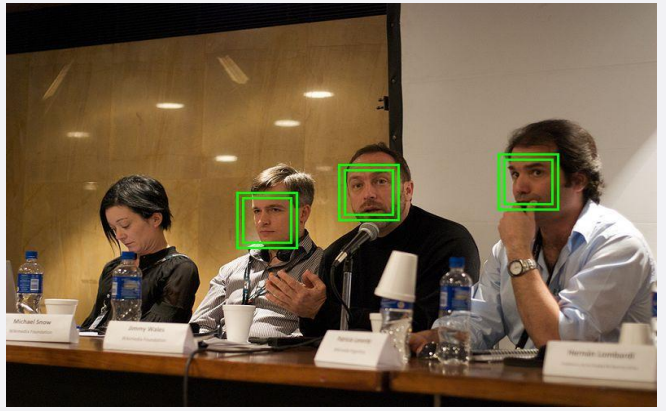
- **Photo credit:** https://commons.wikimedia.org/wiki/File:Wild_horses%3F.jpg (Brian Snelson)
- **Photo Credit:** https://pixabay.com/photos/eye-blue-eye-iris-pupil-face-1173863/
- **Photo Credit:** https://pixabay.com/illustrations/brain-lobes-neurology-human-body-1007686/
- **Photo Credit:** https://pixabay.com/en/camera-chat-internet-video-web-2022419/
- **Photo Credit:** https://commons.wikimedia.org/wiki/File:Gnome-computer.svg (GNOME icon artists)

INTRODUCTION
WHY IS IT IMPORTANT?

- Self Driving Cars – pedestrian/car detection
- Face recognition
- Object detection
- Handwriting Recognition
- License plate numbers detection
- Snapchat filters







Sundog Education

sundog-education.com

- **Photo credit:** https://pl.wikipedia.org/wiki/Plik:Waymo_self-driving_car_rear_three-quarter_view.gk.jpg (Grendelkhan)
- **Photo Credit:** https://commons.wikimedia.org/wiki/File:California_license_plate_Anarchist.jpg (Vards Uzvards)
- **Photo Credit:** https://fr.m.wikipedia.org/wiki/Fichier:D%C3%A9tection_de_personne_-_exemple_2.jpg (ll0zz @Flicker)
- **Photo Credit:** https://commons.wikimedia.org/wiki/File:Face_detection.jpg (Beatrice Murch)

INTRODUCTION
WHY IS IT SO CHALLENGING?

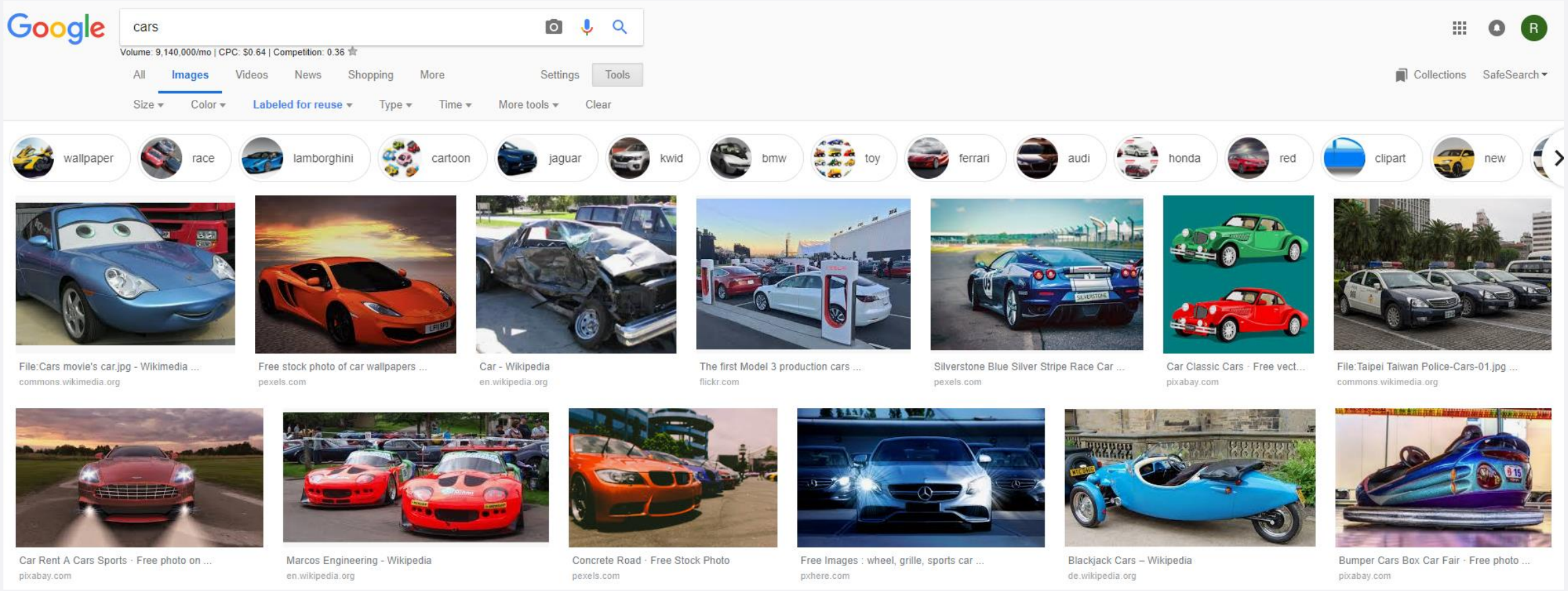


Photo credit: google search: cars!

INTRODUCTION
WHY IS IT SO CHALLENGING?

- **Viewpoints**
- Camera limitation
- Lighting
- Scaling
- Object variation

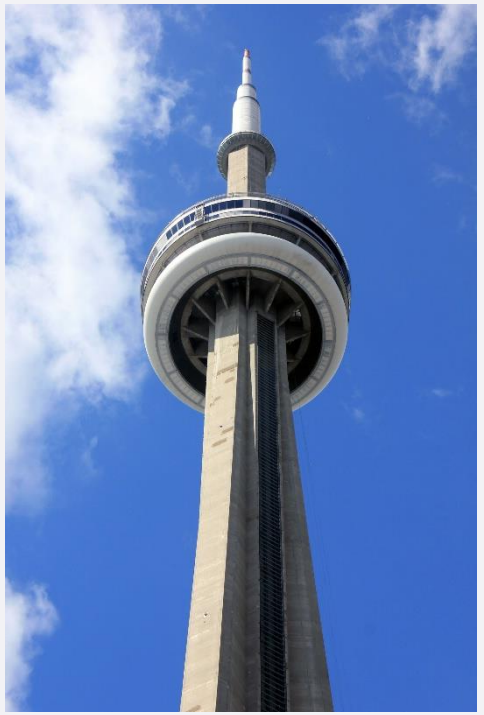


Photo credit: https://commons.wikimedia.org/wiki/File:Gfp-canada-ontario-toronto-cn-tower-up-close.jpg (*Creative Commons Public Domain Dedication)*
Photo Credit: https://commons.wikimedia.org/wiki/File:Toronto_-_ON_-_Toronto_Harbourfront7.jpg (Wladyslaw)
Photo Credit: https://pxhere.com/en/photo/1045265 (Creative Commons CC0)

INTRODUCTION
WHY IS IT SO CHALLENGING?

- Viewpoints
- **Camera limitation**
- Lighting
- Scaling
- Object variation

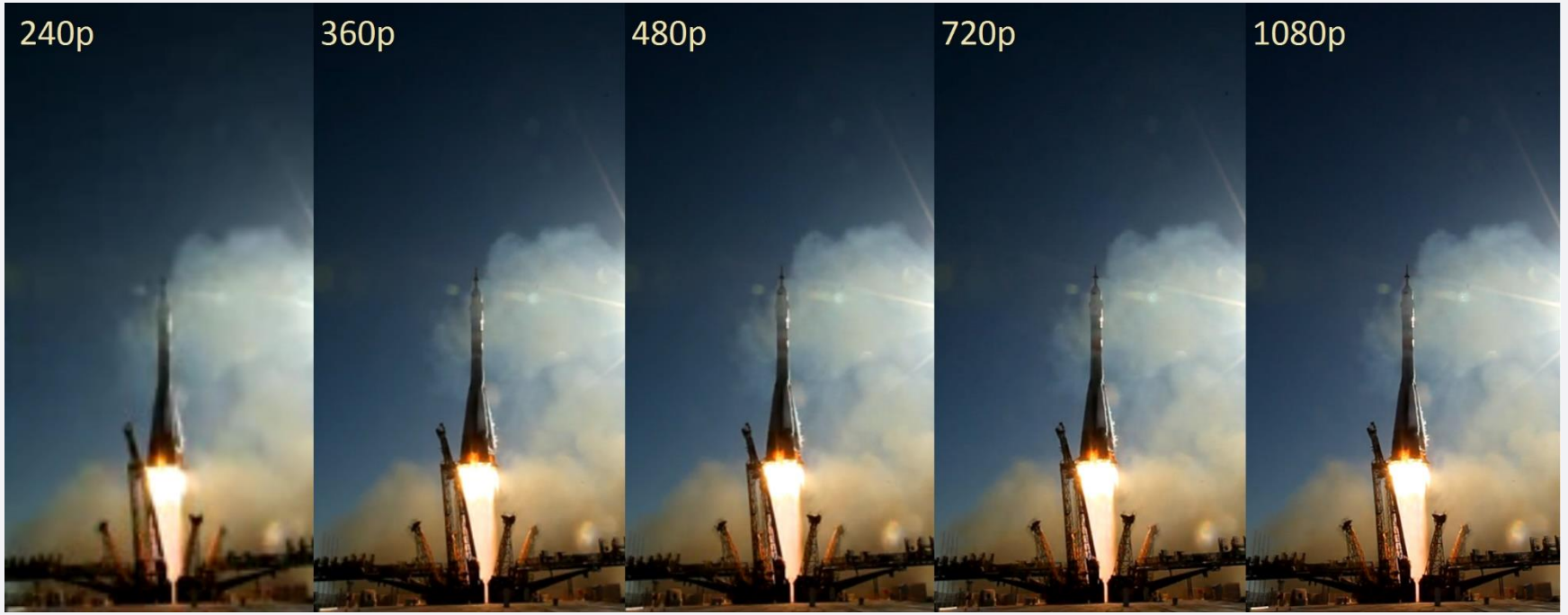


Photo credit: https://commons.wikimedia.org/wiki/File:YouTube-resolution-comparison.jpg

INTRODUCTION
WHY IS IT SO CHALLENGING?

- Viewpoints
- Camera limitation
- **Lighting**
- Scaling
- Object variation

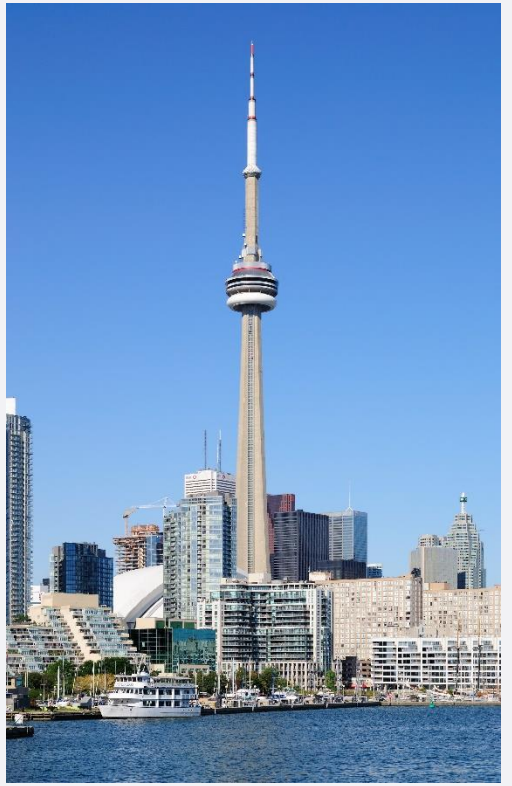


Photo credit: https://commons.wikimedia.org/wiki/File:Toronto_-_ON_-_Toronto_Harbourfront7.jpg (Wladyslaw)
Photo Credit: https://pxhere.com/en/photo/557866 (Creative Commons CC0)

INTRODUCTION
WHY IS IT SO CHALLENGING?

- Viewpoints
- Camera limitation
- Lighting
- **Scaling**
- Object variation

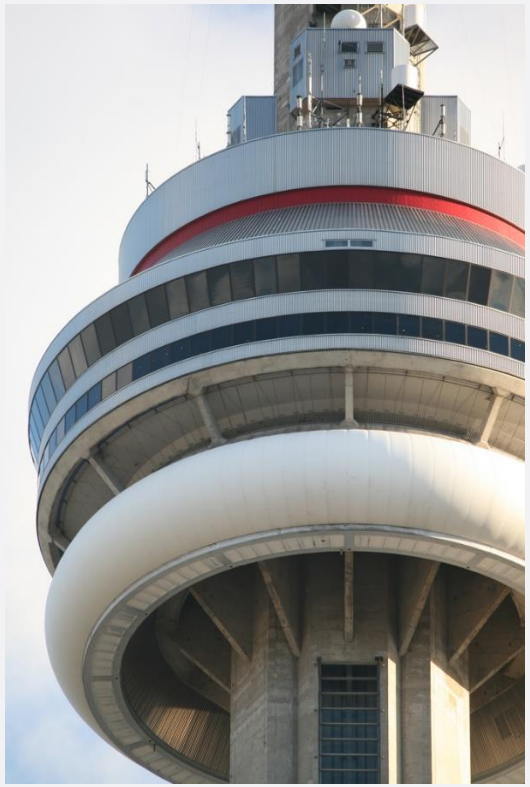


Photo credit: https://commons.wikimedia.org/wiki/File:Toronto_-_ON_-_Toronto_Harbourfront7.jpg (Wladyslaw)
Photo Credit: https://www.flickr.com/photos/fallout/3028182083 (Mihnea Stanciu)

INTRODUCTION
WHY IS IT SO CHALLENGING?

- Viewpoints
- Camera limitation
- Lighting
- Scaling
- **Object variation**

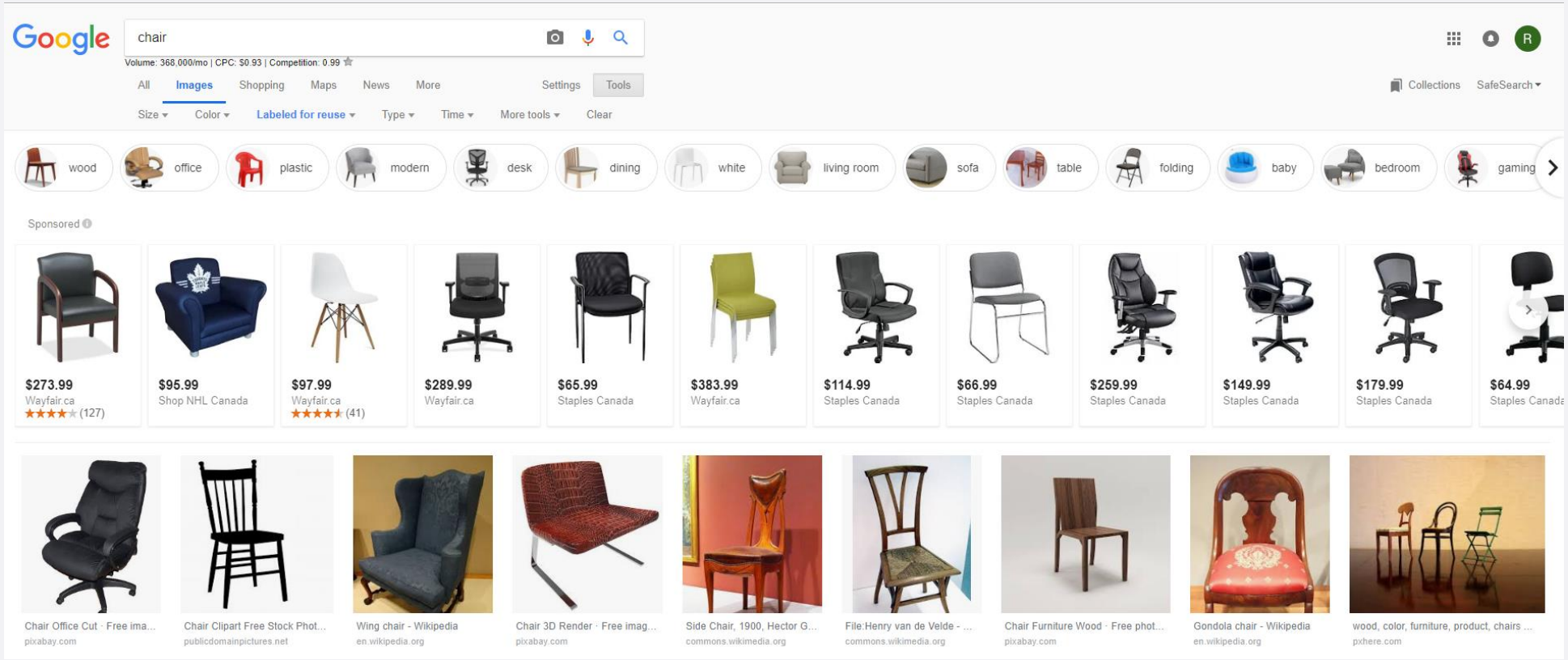


Photo credit: Google search “Chair”!

HUMANS VS. COMPUTER VISION SYSTEMS
WHICH ONE IS MORE ADVANCED?

- Let's assume we want to put **requirements** for an **artificial eye** to convert a conventional vehicle into autonomous one:

REQUIREMENT	HUMAN EYE
Must provide **360°** coverage around the vehicle.	3D vision for 130° of the Field of view 'blind-spot'
Must identify **objects** in 3D close and far from the vehicle.	Only central 50° FOV is 'high-resolution', outside central zone, perception drops
Must meet all requirements in various **lighting**/**weather** conditions.	Good performance in varying lighting conditions
Process data in **real-time**.	Equivalent video 'frame rate' good in central zone, poor at peripheral vision

- **Note:** assumptions above ignore age, disease and distractions
- **Conclusion:** human eyes provide acceptable vision in a very narrow field of view facing forward.

Photo Credit: https://commons.wikimedia.org/wiki/File:Google_driverless_car_at_intersection.gk.jpg (Grendelkhan)

HUMANS VS. COMPUTER VISION SYSTEMS
WHAT SENSORS DOES A SELF-DRIVING CAR HAVE?

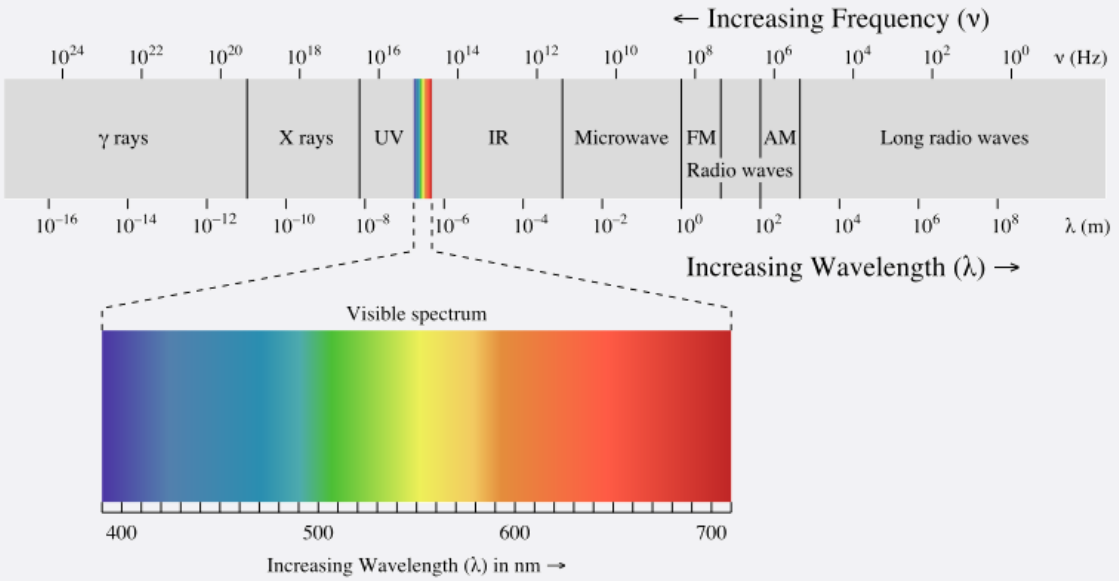
- Self driving cars have set of sensors that give perception of the world, and a machine learning algorithms to process sensors data and take actions.
  - **Cameras:** provides detailed images but require machine-learning algorithms to interpret 2-D images.
  - **LIDAR (**Light Detection and Ranging)**:** works like radar, but instead of sending radio waves it emits infrared laser beams to measure how long they take to come back. Creates a 3-D map of the world in real time.
  - **Radars:** sends radio waves to detect objects, reliable and cheaper than LIDAR systems.
  - **GPS:** Global positioning system provides information about current location of driverless car on the road using set of maps.

Check this out:
http://www.webpages.uidaho.edu/vakanski/how%20driverless%20cars%20work.html

Photo Credit: https://commons.wikimedia.org/wiki/File:Waymo_Chrysler_Pacifica_in_Los_Altos,_2017.jpg (Dllu)
Reference: https://www.ecnmag.com/blog/2016/04/how-autonomous-cars-will-deal-snow
Reference: http://www.webpages.uidaho.edu/vakanski/how%20driverless%20cars%20work.html

IMAGE
HOW DO HUMANS SEE COLORS?

- Waves are observed by the human eye and translated in the visual cortex into color.
- When you look at a banana, the wavelengths of reflected light determine what color you see.
- The light waves reflect off the banana and hit your eye has a wavelengths of 570 to 580 nanometers. These are the wavelengths of yellow light.



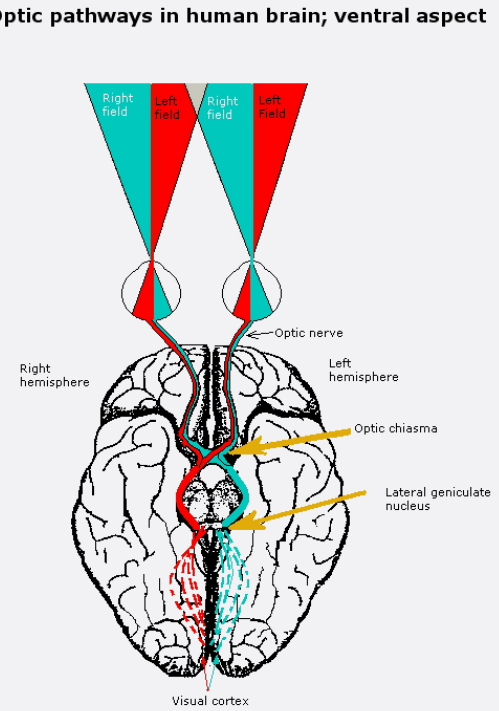
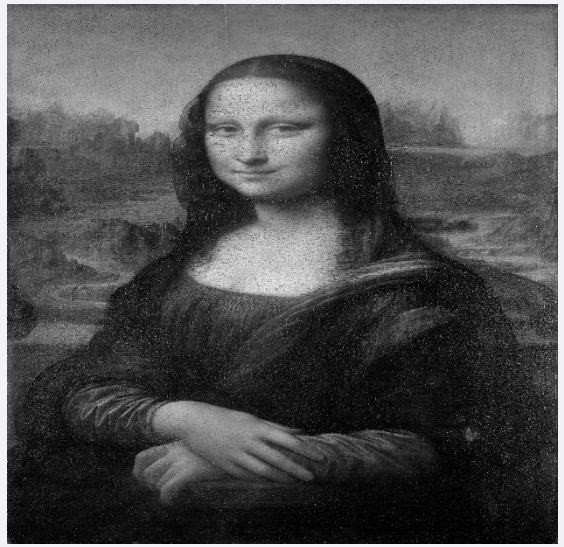


Photo Credit: https://commons.wikimedia.org/wiki/File:Optic_processing_human_brain.jpg (JonRichfield)
Photo Credit: https://de.wikipedia.org/wiki/Datei:EM_spectrum.svg (Philip Ronan)
Photot Credit: https://commons.wikimedia.org/wiki/File:Banana-Single.jpg (Evan-Amos)

IMAGE
HOW DO WE DIGITALLY REPRESENT A GRAYSCALE IMAGE?

- A greyscale image is system of 256 tones with values ranging from 0-255.
- '0' represents black and '255' represents white.
- Numbers in-between represents greys between black and white.
- Binary systems use digits '0' and '1‘ where '00000000' for black, to '11111111' for white (8-bit image).
- **Note:** binary value of '11111111' is equal to decimal value of '255'



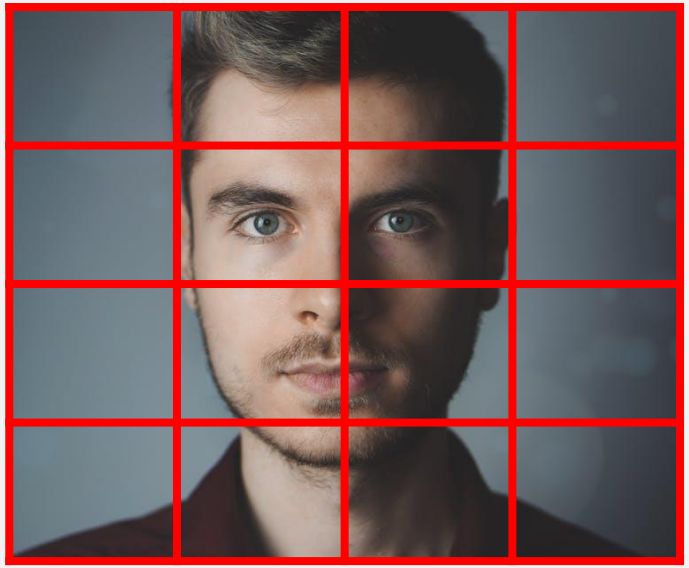
255	255	255
155	155	155
0	0	0

11111111	11111111	11111111
10011011	10011011	10011011
00000000	00000000	00000000

Photo Credit: https://www.pexels.com/photo/woman-art-painting-mona-lisa-40997/

IMAGE
HOW DO WE DIGITALLY REPRESENT A COLORED IMAGE?

- Each pixel coordinate (x, y) contains 3 values ranging for intensities of 0 to 255 (8-bit).
- RGB image is split into 3 matrices corresponding to red, green, and blue channels.
- Any other colour can be created by mixing several intensities of RGB.



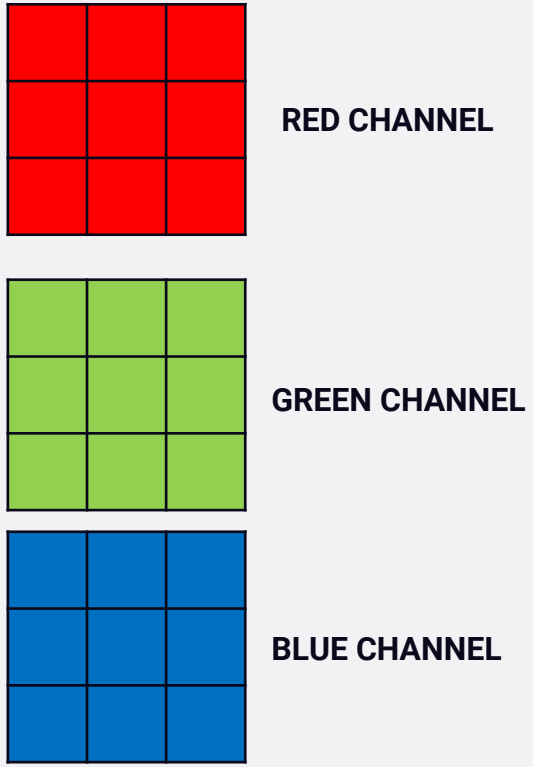
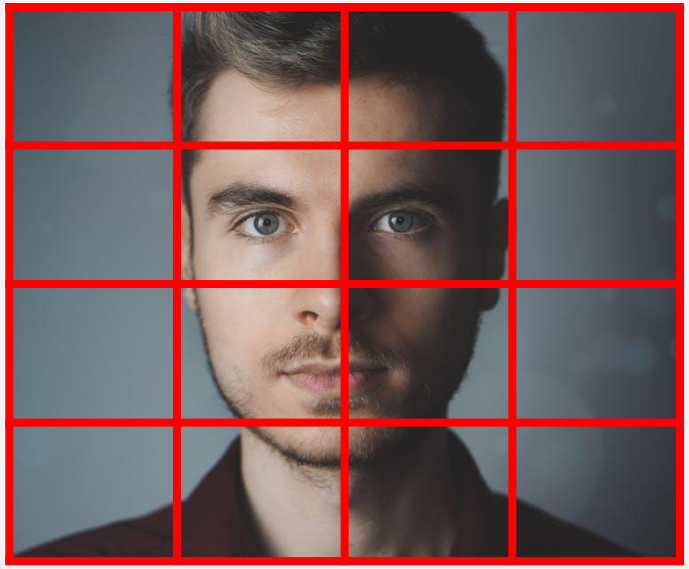


Photo Credit: https://commons.wikimedia.org/wiki/File:AdditiveColor.svg
Photo Credit: https://www.pexels.com/photo/face-facial-hair-fine-looking-guy-614810/ (Simon Robben)

IMAGE
HOW DO WE DIGITALLY REPRESENT A COLORED IMAGE?

- Each RGB pixel has three sets of 8-bit binary numbers which in turn translates into 24 bits of computer information in total, i.e.: '24-bit colour'.
- Assuming same number of pixels, RGB image is three times bigger in file size than a Greyscale one.



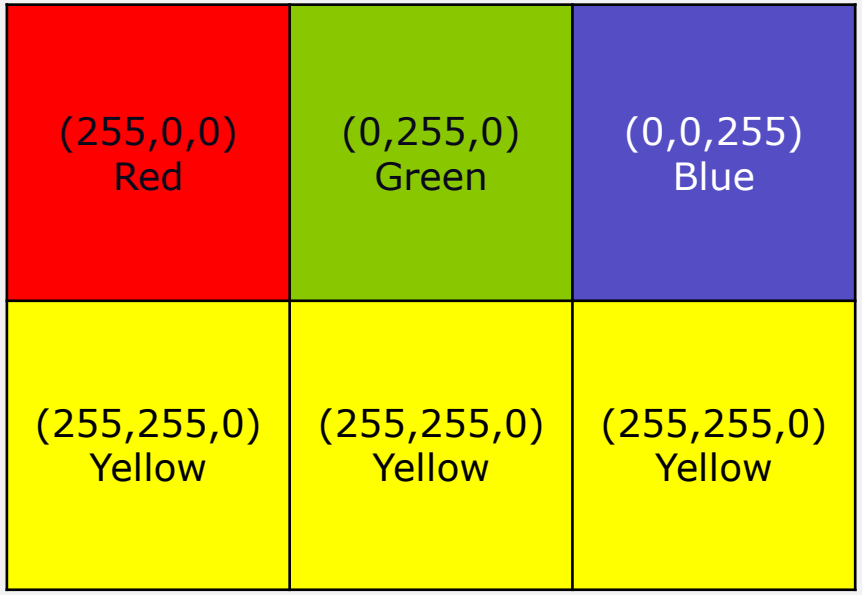


Photo Credit: https://www.pexels.com/photo/face-facial-hair-fine-looking-guy-614810/ (Simon Robben)

COLOR SELECTION:
WHAT ARE THE CHALLENGES OF COLOR SELECTION TECHNIQUE?

- What if the lane colour is not white?
- What if it is dark outside under different weather or lighting conditions?

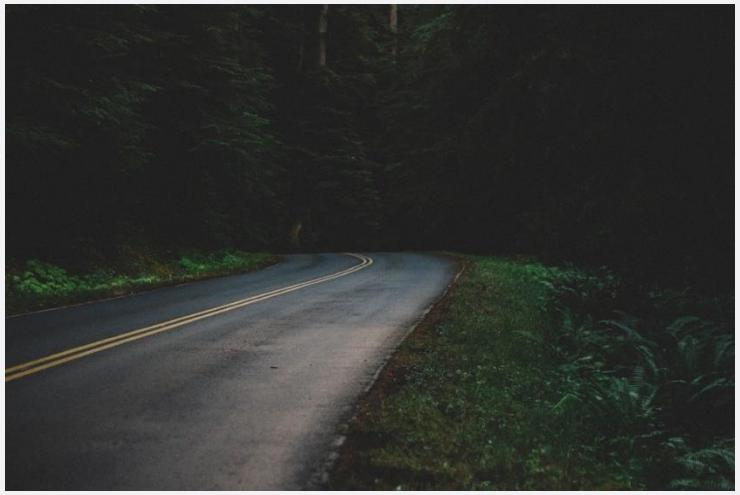


Photo Credit: https://pxhere.com/en/photo/1407123
Photo Credit: https://pixabay.com/en/road-street-snow-winter-trees-796932/
Photo Credit: https://pixabay.com/en/street-road-night-dark-asphalt-918882/

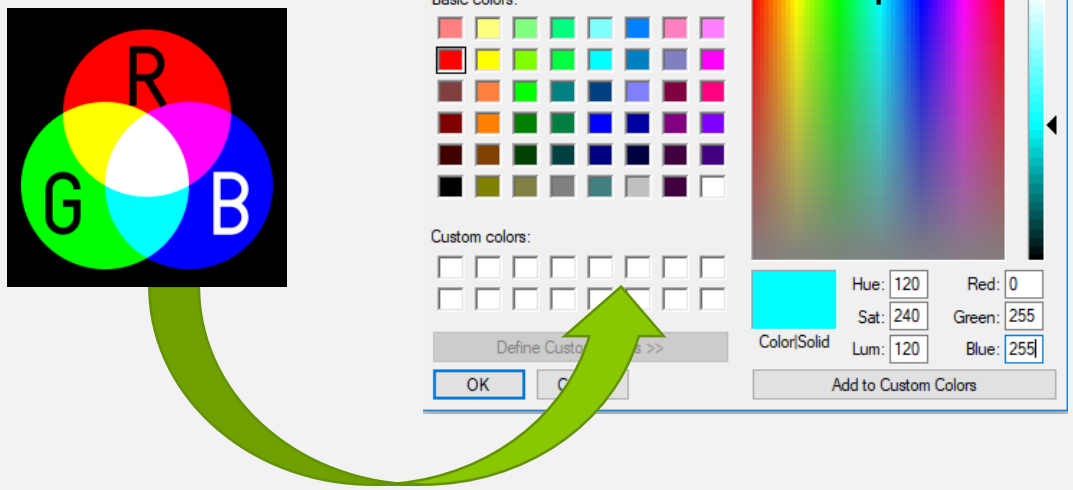
COLOR SELECTION: SOLUTIONS?

- More features has to be extracted from the camera image using more sophisticated computer vision techniques.
- Vehicles use LiDAR systems to create high resolution 3D digital maps of the surroundings.
- Example: in ideal weather conditions, the car collects 2.8 million laser points per second to create the LiDAR map
- This is equivalent to collecting up to 600 gigabytes of data per hour.
- The car can utilize the map created of the commute during summertime to be used as a reference during winter time.

Reference: https://www.ecnmag.com/blog/2016/04/how-autonomous-cars-will-deal-snow

COLOR SPACES:
RGB

- RGB color space is made of red, green, and blue and mixing them up can produce any color defined by those primary colors.
- OpenCV stores color in the BGR format.
- Let's try some colors in paint!



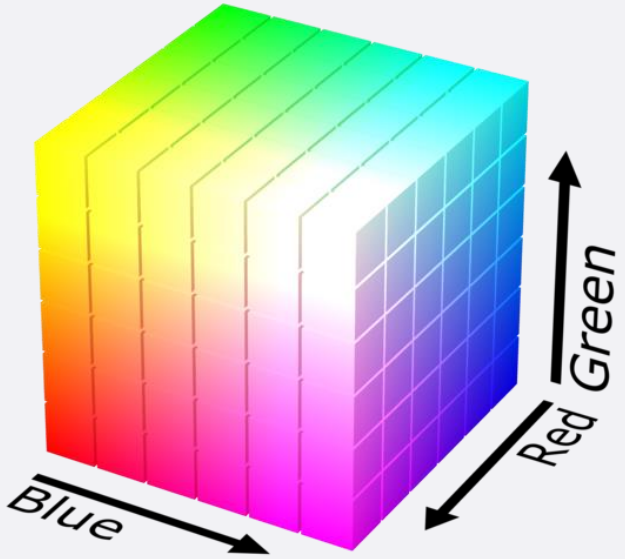


Photo Credit: https://commons.wikimedia.org/wiki/File:AdditiveColor.svg
Photo Credit: https://ar.wikipedia.org/wiki/%D9%85%D9%84%D9%81:RGB_color_solid_cube.png

COLOR SPACES:
HSV

- HSV (Hue, Saturation & Value/Brightness) is a color space that stores color information in a cylindrical representation.
- HSV color space aligns with the way human vision perceives color-making attributes.
  - Hue – Color Value (0 – 179)
  - Saturation – Vibrancy of color (0-255)
  - Value – Brightness or intensity (0-255)
- **Hue** is arranged in a radial slice, around a central axis of neutral colors ranging from black at bottom to white at the top.
- The **saturation** dimension resembling various shades of brightly colored paint.
- **Value** dimension resembling the mixture of those paints with varying amounts of black or white paint.

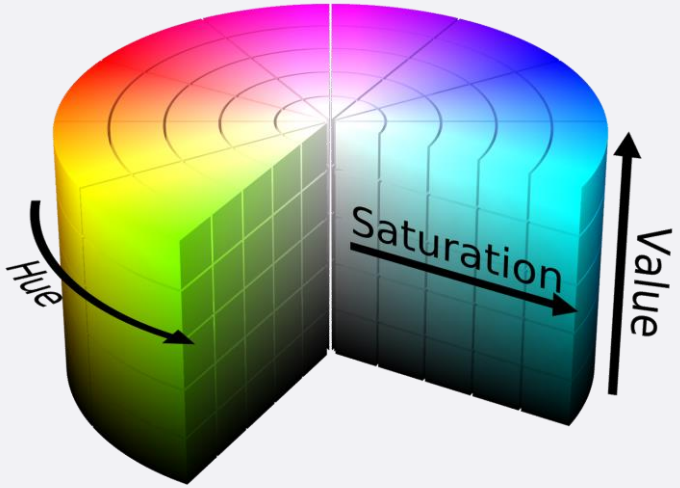
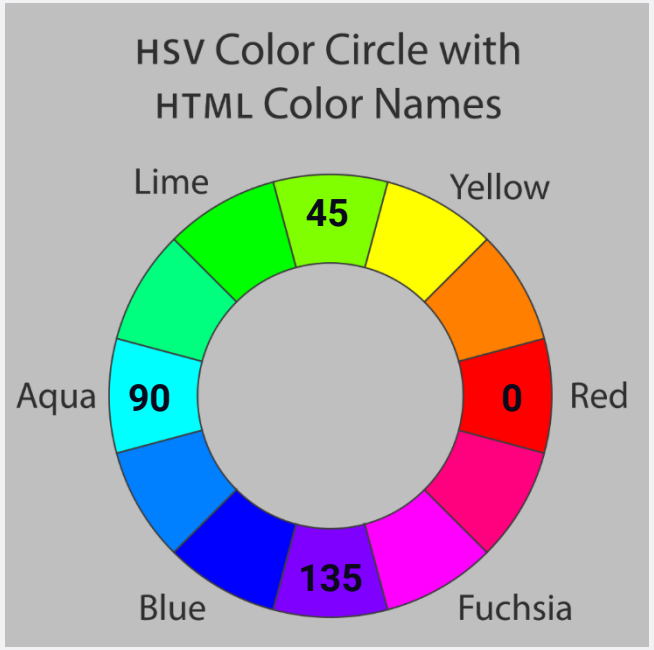


Photo Credit: https://fr.m.wikipedia.org/wiki/Fichier:HSV_color_solid_cylinder_alpha_lowgamma.png (SharkD)

COLOR SPACES:
RGB TO HSV

- The Hue color ranges from 0 to 180 (not 360) **in OpenCV**
  - Green – 45 to 75
  - Red – 165 to 15
  - Blue – 90 to 120
- Learn how to convert from RGB to HSV:
  http://www.javascripter.net/faq/rgb2hsv.htm
- Don't forget the ½ factor in Hue!



R: 0 G: 255 B: 0 Convert to HSV H: 120.0 S: 1.0000 V: 1.0000

R: 255 G: 0 B: 0 Convert to HSV H: 0.0 S: 1.0000 V: 1.0000

R: 0 G: 0 B: 255 Convert to HSV H: 240.0 S: 1.0000 V: 1.0000

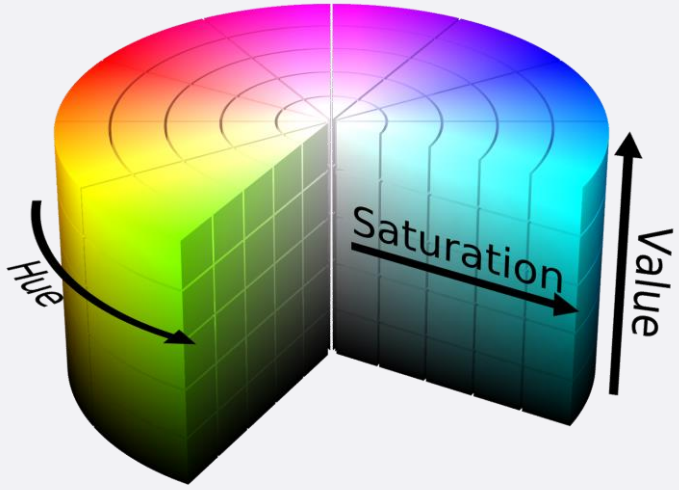


Photo Credit: https://fr.m.wikipedia.org/wiki/Fichier:HSV_color_solid_cylinder_alpha_lowgamma.png (SharkD)
Photo Credit: https://commons.wikimedia.org/w/index.php?curid=12405869 (By Jacobolus - Own work, CC BY-SA 3.0)

CONVOLUTIONS:
INTUITION

- Convolutions use a kernel matrix to scan a given image and apply a filter to obtain a certain effect.
- An image Kernel is a matrix used to apply effects such as blurring and sharpening.
- Kernels are used in machine learning for *feature extraction* to select most important pixels of an image.
- Convolution preserves the spatial relationship between pixels.

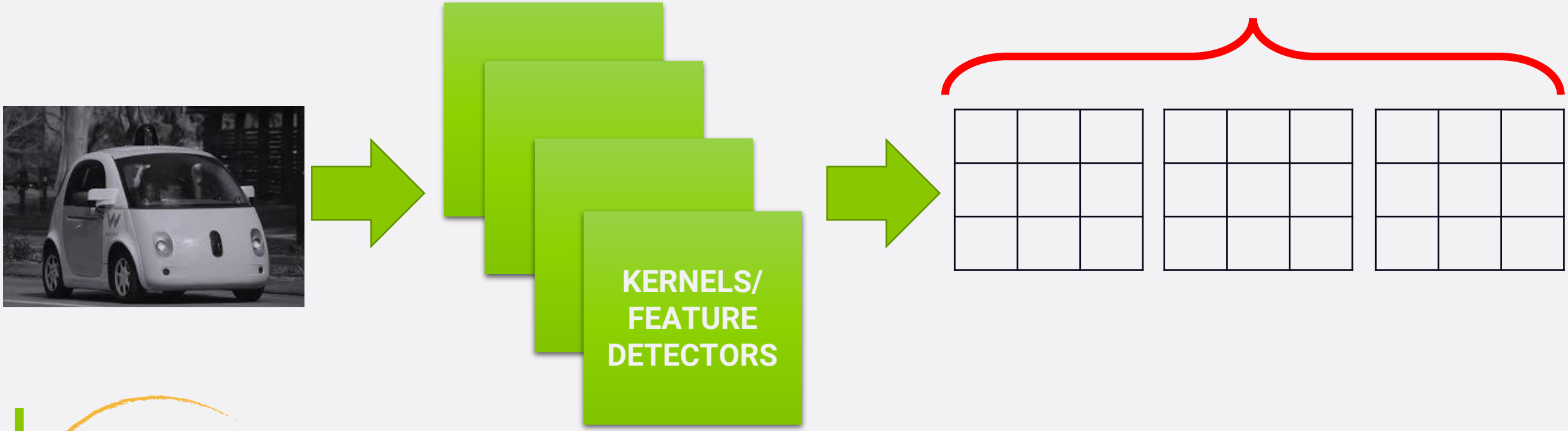


Photo Credit: https://commons.wikimedia.org/wiki/File:Waymo_self-driving_car_front_view.gk.jpg (Grendelkhan)

CONVOLUTIONS:
INTUITION

FEATURE DETECTOR

1	0	0
0	1	0
0	1	0

0	1	1	0	1
1	0	0	0	1
1	1	0	0	0
0	1	1	1	0
0	0	1	0	1

1	1	1
3	1	1
2	3	1

FEATURE MAP

GREYSCALE IMAGE

- Live Convolution: http://setosa.io/ev/image-kernels/

Photo Credit: https://commons.wikimedia.org/wiki/File:Waymo_self-driving_car_front_view.gk.jpg (Grendelkhan)

CONVOLUTIONS:
SHARPENING AND BLURRING

- The **sharpen** kernel emphasizes differences in adjacent pixel values which makes the image look more vivid.
- The technique is used to bring out detail in an image by enhancing the contrast of pixels on edges.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \qquad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

- The **blurring** kernel is used to blur a given image by averaging each pixel value and its neighbors.
- Blurring Kernel is an NxN matrix filled with ones. Normalization has to be performed.
- The values in the matrix has to sum to 1, if sum doesn't add to 1 then image will be brighter or darker

$$1/9 \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

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

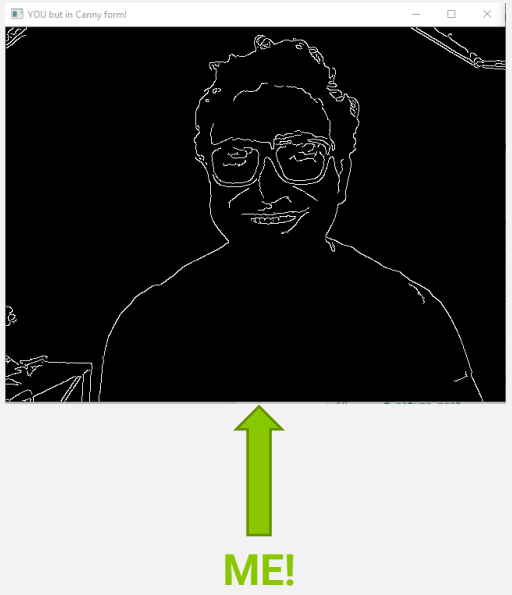
CONVOLUTIONS:
SHARPENING AND BLURRING

Operation	Kernel	Resulting Image
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$1/9\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$1/16\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Source: https://en.wikipedia.org/wiki/Kernel_(image_processing)
Photos Credit: https://commons.wikimedia.org/wiki/File:Vd-Blur1.png **(**Michael Plotke)

EDGE DETECTION AND GRADIENT CALCULATION INTUITION

- Edge detection is a tool that identifies points in a digital image at which the image brightness changes sharply or has discontinuities.
- Edge detection is important tool in computer vision especially for feature extraction/detection.
- We will apply a live edge detection transform of your face shortly!



EDGE DETECTION AND GRADIENT CALCULATION: SOBEL

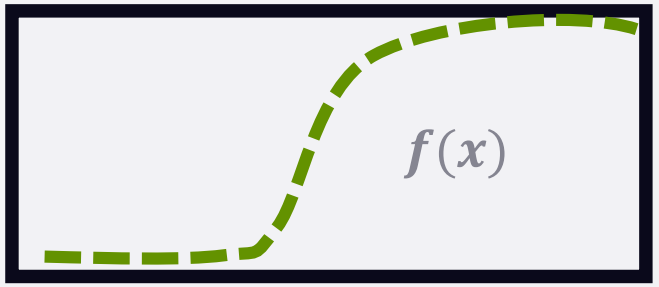
- Sobel edge detector is a gradient based method based on the first order derivatives.
- It calculates the first derivatives of the image separately for X and Y axes.
- Sobel uses two 3X3 kernels which are convolved with original image to calculate the derivatives.
- For image **A** , $\mathbf{G}_x$ and $\mathbf{G}_y$ are two images representing horizontal and vertical derivative approximations:

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

- Calculate:
  - Magnitude or "strength" of the edge: $\sqrt{G_x^2 + G_y^2}$
  - Approximate strength: $|G_x| + |G_y|$
  - The orientation of the edge: $\arctan(\frac{G_y}{G_x})$

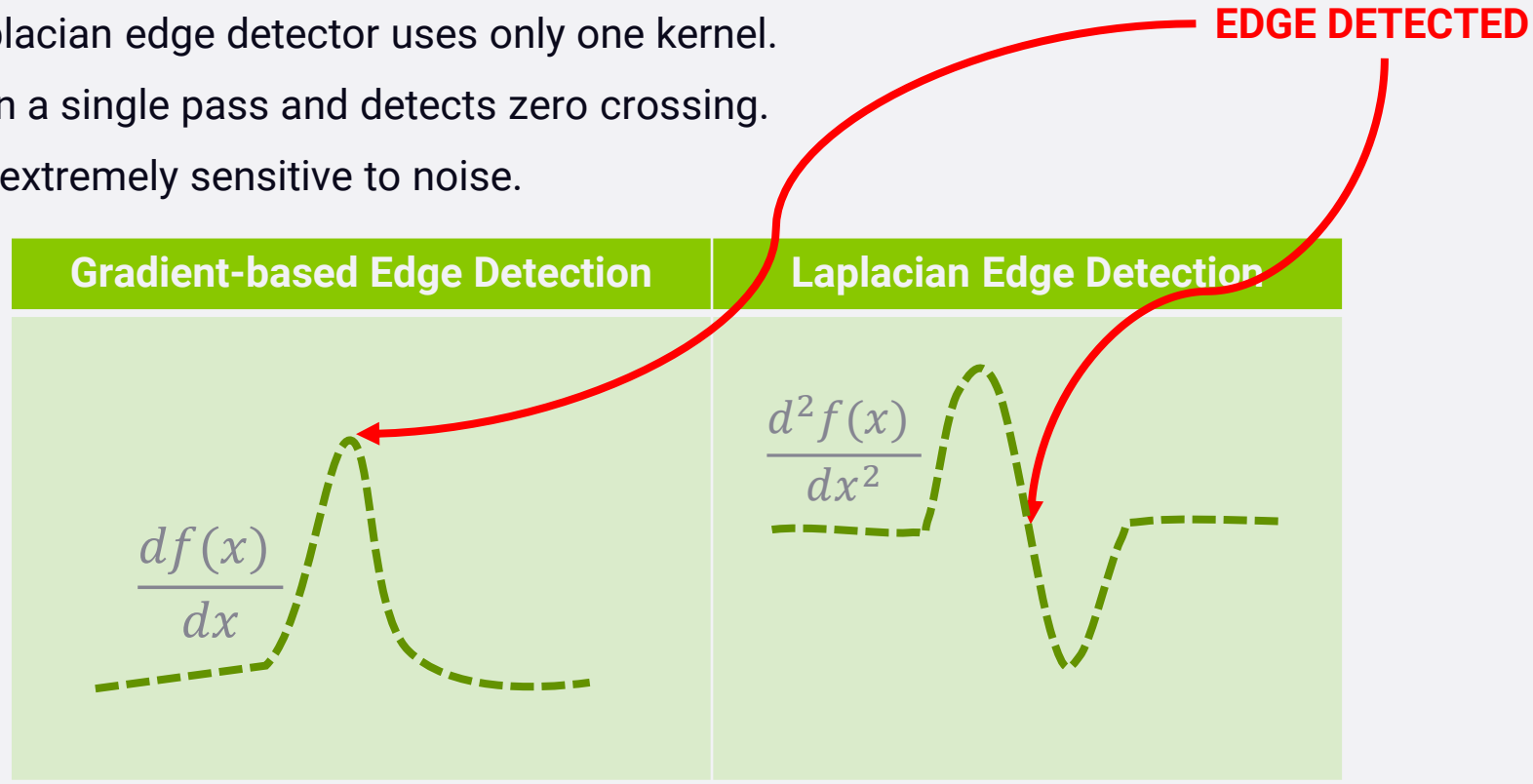
Reference:
http://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_Image_Gradient_Sobel_Laplacian_Derivatives_Edge_Detection.php
Reference: http://www.aishack.in/tutorials/sobel-laplacian-edge-detectors/

EDGE DETECTION AND GRADIENT CALCULATION: LAPLACIAN



- Unlike the Sobel edge detector, the Laplacian edge detector uses only one kernel.
- It calculates second order derivatives in a single pass and detects zero crossing.
- Second order derivatives are generally extremely sensitive to noise.
- The Laplacian kernel is:

0	-1	0
-1	4	-1
0	-1	0



EDGE DETECTION AND GRADIENT CALCULATION: CANNY

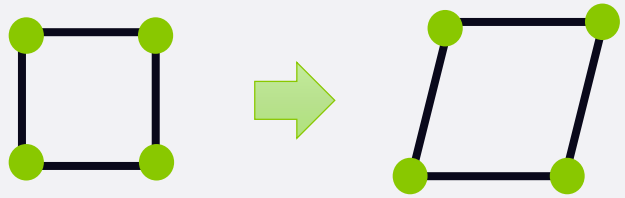
1. **Smoothing:** Smooth the image with a Gaussian filter with spread σ
2. **Gradient:** Compute gradient magnitude and direction at each pixel of the smoothed image
3. **Non-maximum suppression (thinning):** Zero out all pixels that are not the maximum along the direction of the gradient (look at 1 pixel on each side)
4. **Thresholding:** Threshold the gradient magnitude image such that strong edges are kept and noise is suppressed

Remove noise using Gaussian filter → Get gradient using Sobel → Non-maximum suppression → Hysteresis thresholding

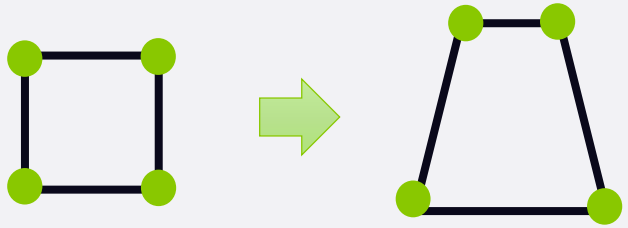
Photo Credit: https://commons.wikimedia.org/wiki/File:Waymo_self-driving_car_front_view.gk.jpg (Grendelkhan)

IMAGE TRANSFORMATIONS
INTRODUCTION

- Image transformations are used to correct distortions and/or change perspectives.
- Affine transformation preserves parallelism.
- Projective (non-affine) transformation does not preserve parallelism, length, and angle.



AFFINE TRANSFORMATION



PROJECTIVE (NON-AFFINE) TRANSFORMATION

IMAGE TRANSFORMATIONS
ROTATION

- OpenCV can perform rotations by using a rotation matrix M
- A rotation matrix is a matrix used to perform rotation in Euclidean space.
- It rotates points in the *xy*-plane counter clockwise through an angle θ about the origin.

M_rotation = getRotationMatrix2D(center, angle, scale)

rotated_image = cv2.warpAffine(image, M_rotation, (width, height))

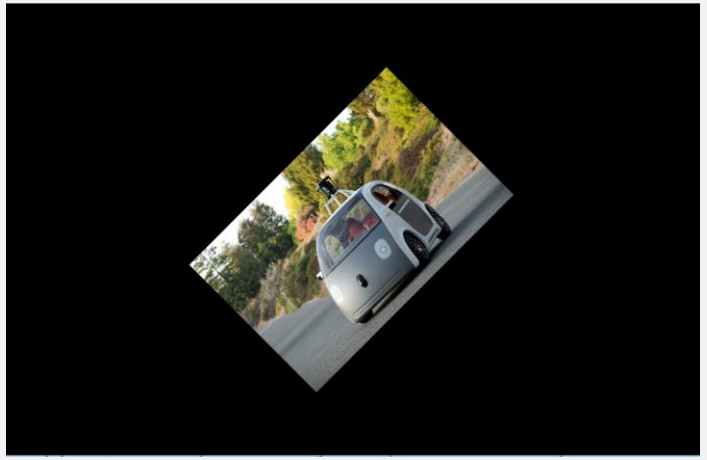


Photo Credit: https://www.flickr.com/photos/smoothgroover22/15104006386

IMAGE TRANSFORMATIONS
TRANSLATION

- Translation is the shifting of object's location in X and/or Y direction.
- OpenCV uses Translational matrix T as follows:

$$T = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \end{bmatrix}$$

Translational_Matrix = np.float32([[1, 0, Tx], [0, 1, Ty]])

translated_image = cv2.warpAffine(image, Translational_Matrix, (width, height))

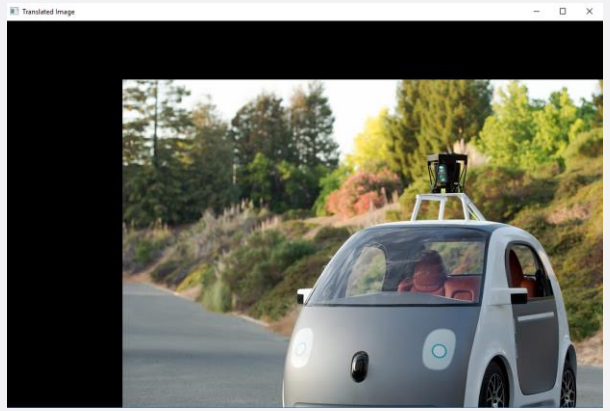


Photo Credit: https://www.flickr.com/photos/smoothgroover22/15104006386

IMAGE TRANSFORMATIONS
RESIZING

- Resizing using OpenCV can be performed using ***cv2.resize()***
- Preferable interpolation methods are ***cv.INTER_AREA*** for shrinking and ***cv.INTER_CUBIC*** for zooming.
- By default, interpolation method used is ***cv.INTER_LINEAR*** for all resizing purposes.

resized_image = cv2.resize(image, None, fx=3, fy=3, interpolation = cv.INTER_CUBIC)

Photo Credit: https://www.flickr.com/photos/smoothgroover22/15104006386

IMAGE TRANSFORMATIONS
PERSPECTIVE TRANSFORM

- Image transformations are used to correct distortions and/or change perspectives.
- Projective transformation (Non-affine) does not preserve parallelism, length, and angle.

M = cv2.getPerspectiveTransform(Source_points, Destination_points)

warped = cv2.warpPerspective(image, M, (width, height))

- ***M:*** Transformation matrix

SOURCE POINTS **DESTINATION POINTS**

Photo Credit: https://patch.com/massachusetts/somerville/somerville-blanket-sets-blanket-speed-limit-25-mph
Photo Credit: https://www.flickr.com/photos/chadelliott2012/5659144073

IMAGE TRANSFORMATIONS
CROPPING

- Image cropping can be performed as follows:

Image_cropped = image[h0:h1 , w0:w1]

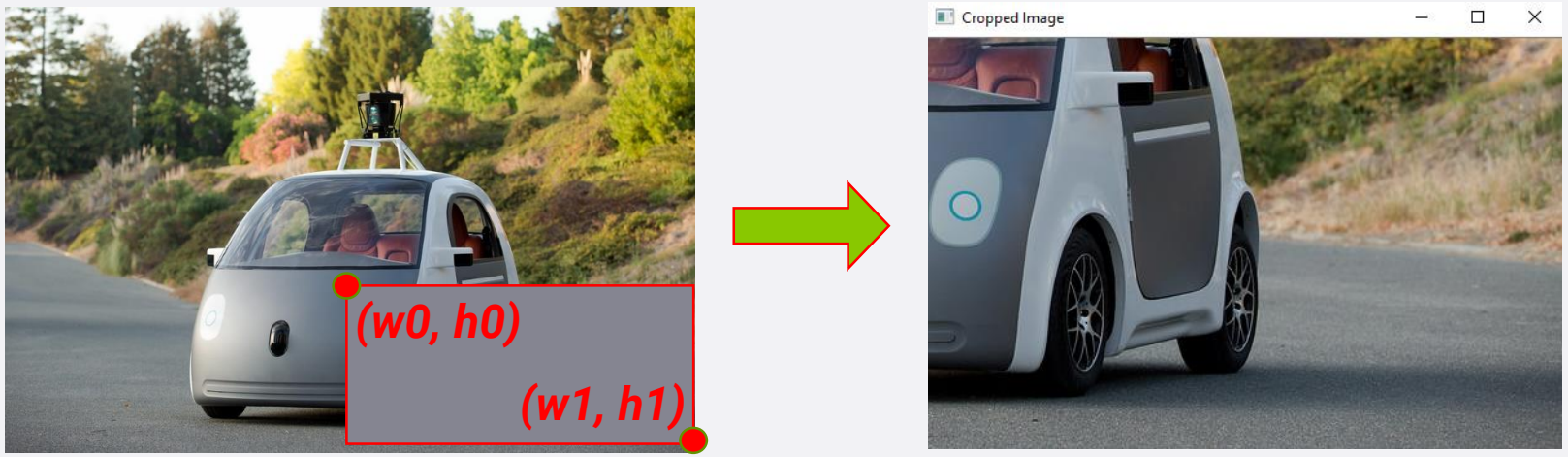


Photo Credit: https://www.flickr.com/photos/smoothgroover22/15104006386

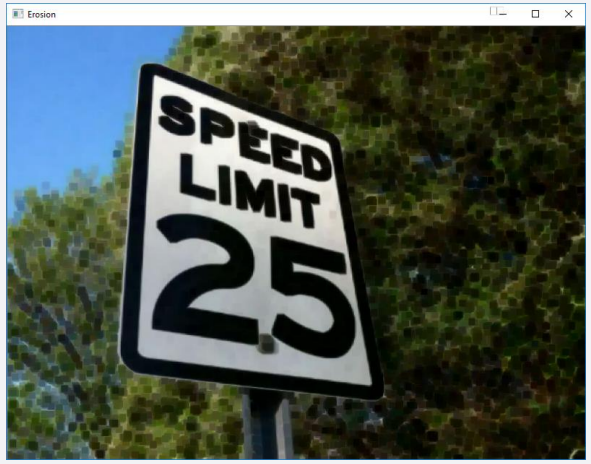
IMAGE TRANSFORMATIONS
DILATION AND EROSION

- **Dilation** means adding extra pixels to the boundaries of objects in an image
- **Erosion** means removing pixels at the boundaries of objects in an image
- Image dilation and erosion can be performed as follows:

kernel = np.ones((8,8), np.uint8)

image_erosion = cv2.erode(image, kernel, iterations=1)

image_dilation = cv2.dilate(image, kernel, iterations=1)



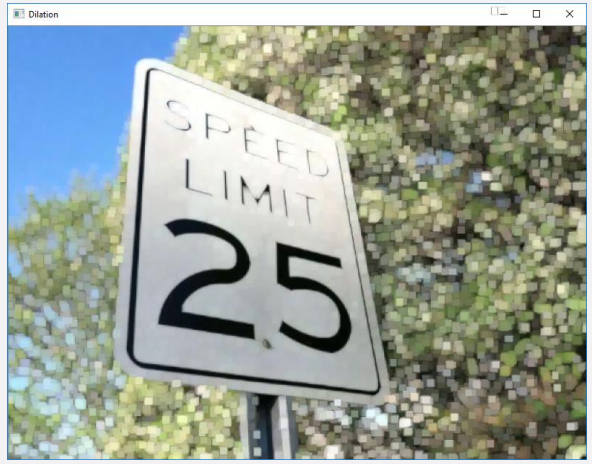
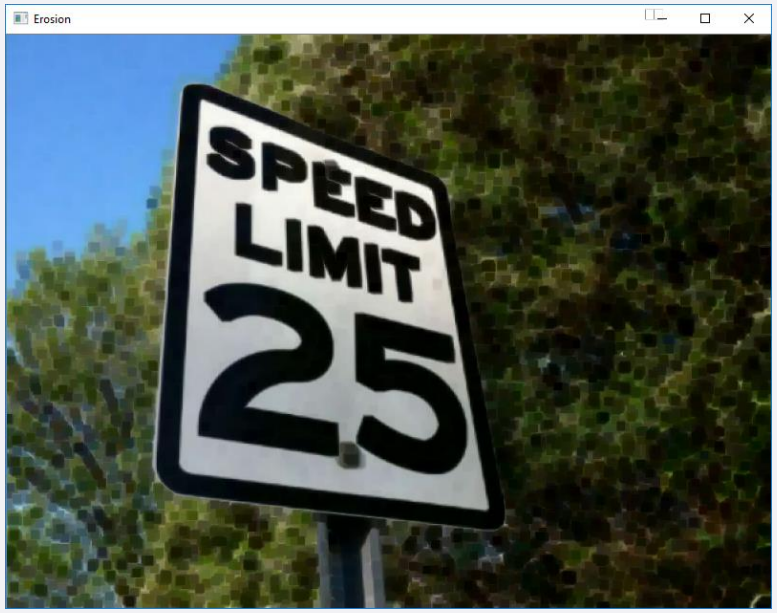


Photo Credit: https://patch.com/massachusetts/somerville/somerville-blanket-sets-blanket-speed-limit-25-mph
Photo Credit: https://www.flickr.com/photos/chadelliott2012/5659144073

IMAGE TRANSFORMATIONS
DO THEY LOOK REVERSED?!

- **Erosion** means removing pixels at the boundaries of objects in an image (removing pixels from the white background means increasing thickness of the letters!)
- **Dilation** means adding extra pixels to the boundaries of objects in an image (adding pixels to the white background so letters appear thinner!)



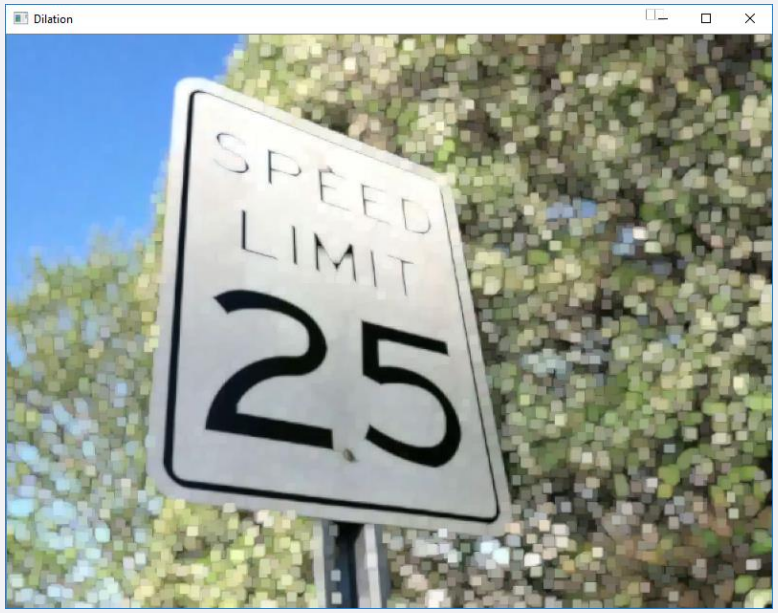
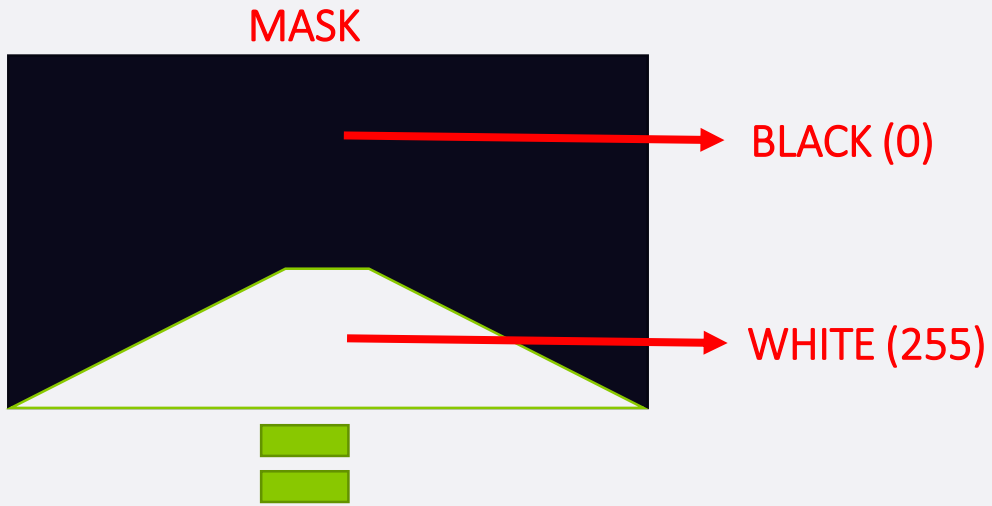


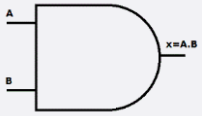
Photo Credit: https://patch.com/massachusetts/somerville/somerville-blanket-sets-blanket-speed-limit-25-mph
Photo Credit: https://www.flickr.com/photos/chadelliott2012/5659144073

REGION OF INTEREST MASKING
INTUITION

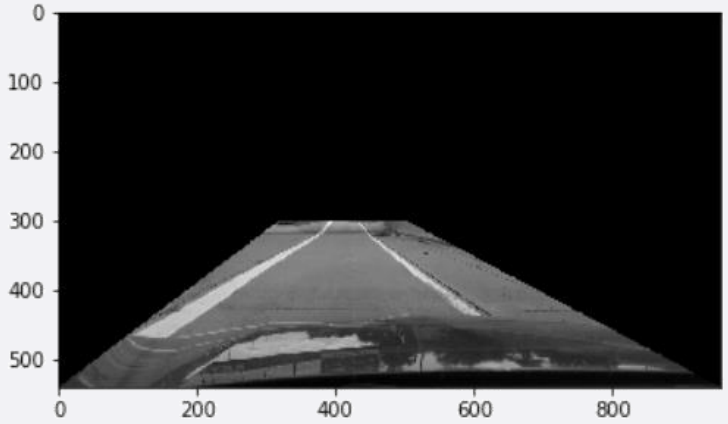
- Our goal is to mask the region of interest so we can focus our search efforts for the lane lines.

x	y	Z=x.y
0	0	0
0	1	0
1	0	0
1	1	1



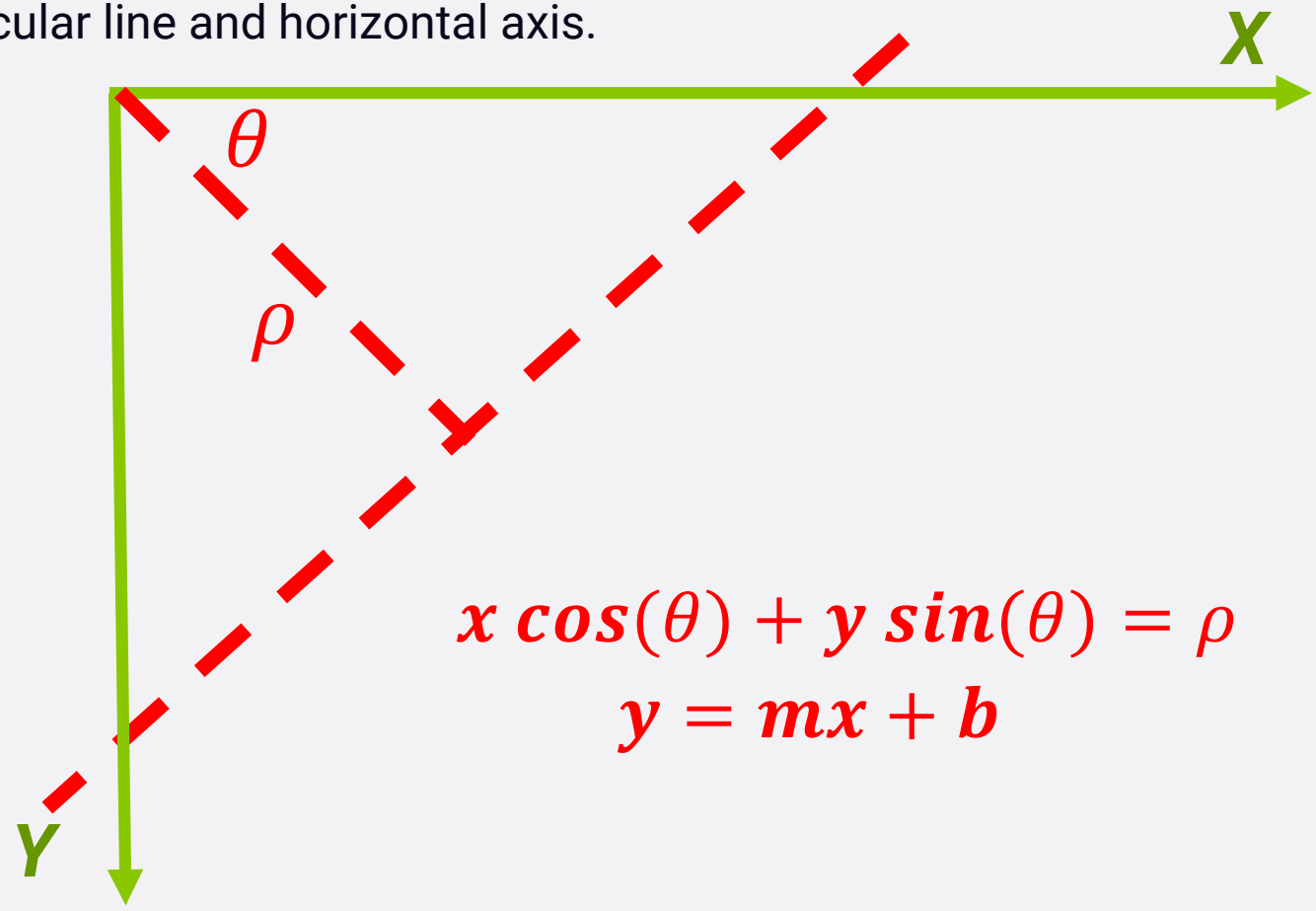


AND OPERATION



HOUGH TRANSFORM
HOW TO REPRESENT A LINE?

- A line can be represented with one of the two forms below.
- ρ is the perpendicular distance from origin to the line.
- θ is the angle formed by this perpendicular line and horizontal axis.



$$x\cos(\theta) + y\sin(\theta) = \rho$$

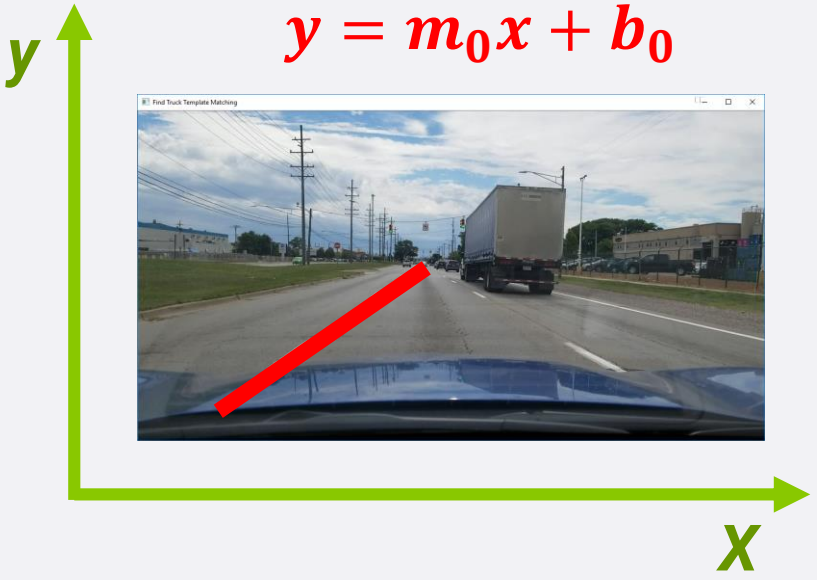
$$y = mx + b$$

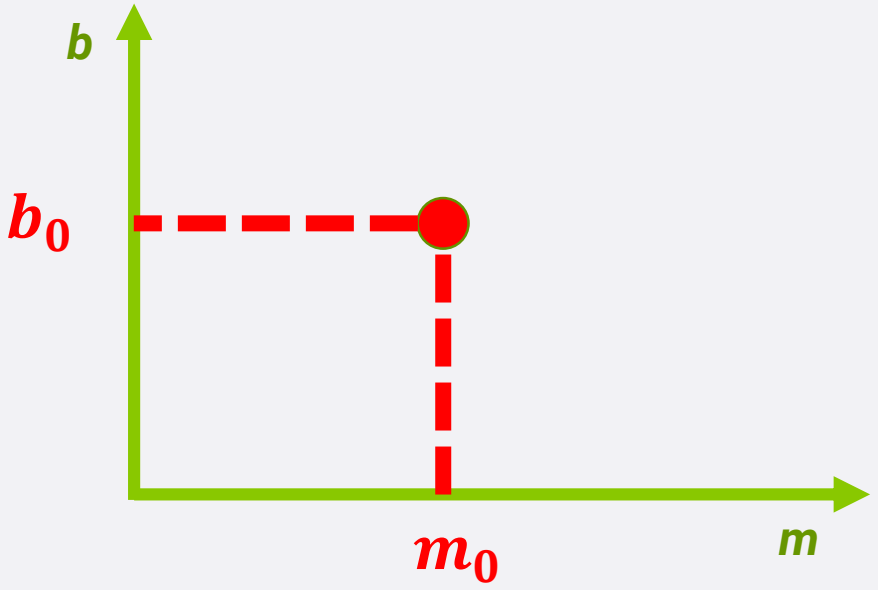
HOUGH TRANSFORM
INTUITION

- In image space, a line is plotted as x vs. y and modeled as $\boldsymbol{y = mx + b}$ or $\boldsymbol{x\,cos(\theta) + y\,sin(\theta) = \rho}$
- In parameter space (hough), a line is represented by a point in "m vs. b".
- Each line is represented as a single point with (m, b) coordinates or $(\rho,\ \theta)$ parameters.

$$\boldsymbol{x\,cos(\theta) + y\,sin(\theta) = \rho}$$

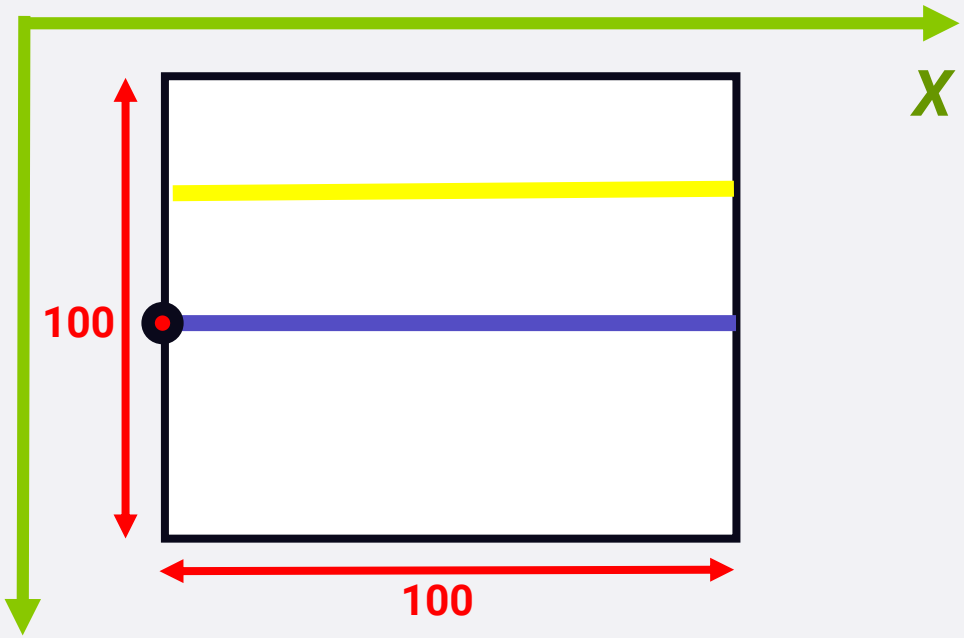
$$\boldsymbol{y = m_0 x + b_0}$$





HOUGH TRANSFORM
EXAMPLE

- Assume 100x100 image with horizontal line in the middle.
- First we create a 2D matrix or accumulator (to hold values of two parameters) and set it to 0.
- Select the first point on the line and try θ values from 0, 1, 2, ..180 and check the obtained value of ρ
- For every (ρ, θ) pair, increment an accumulator by 1 which means cell (50, 90) will be incremented by one
- Try next point, and repeat procedure.
- Cell (50, 90) will be voted up, the accumulator with maximum votes indicates a line!



$$x\cos(\theta) + y\sin(\theta) = \rho$$

HOUGH TRANSFORM
INTUITION

- The Hough Line Transform in openCV is ***cv2.HoughLines()*** is used to detect straight lines.

lines = cv2.HoughLines(image, ρ accuracy, θ accuracy, threshold)

lines = cv2.HoughLines(image, 1, np.pi/180, 240)

- To apply the Transform, first apply Canny edge detection pre-processing.
  - lines: A vector that will store the parameters (ρ, θ) of the detected lines
  - ρ : The resolution of parameter ρ in pixels.
  - θ : The resolution of the parameter θ in radians.
  - threshold: The minimum number of intersections to "detect" a line (minimum vote to be considered a line).

Reference: https://docs.opencv.org/3.1.0/d6/d10/tutorial_py_houghlines.html

IMAGE FEATURES
INTRODUCTION

- Image Features are important areas an image that are unique to a specific image.
- A feature is a piece of information in the image such as points, edges or objects that is different/unique.
- A feature may be a color or a detected edge.
- A good feature has to be repeatable, i.e.: if feature can be detected in two or more different images of the same scene.

IMAGE FEATURES
WHY ARE FEATURES IMPORTANT?

- Image Features are critical in machine learning and self-driving cars because they can be used to analyze, describe and match images.
- Features can be used to train a classifier to detect objects such as pedestrians and cars.

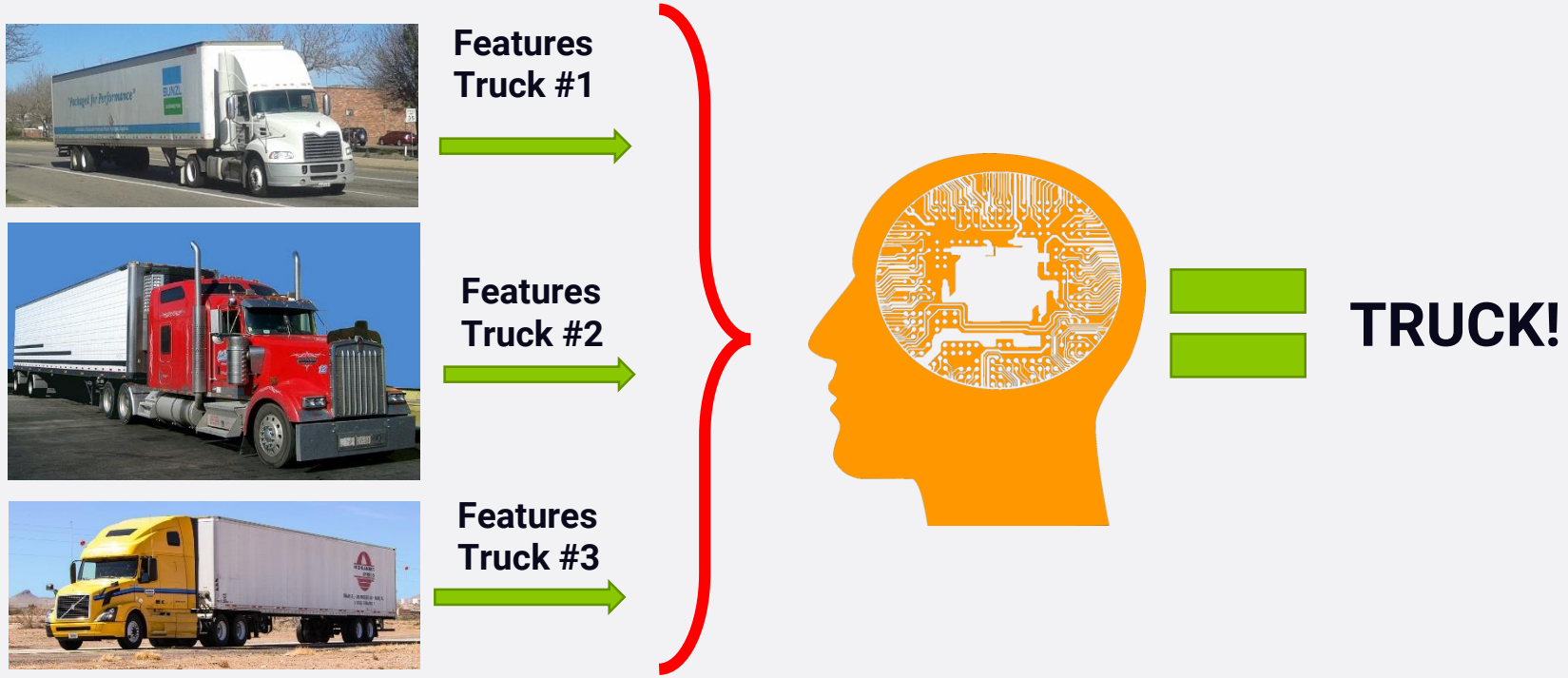


Photo Credit: https://svgsilh.com/ff9800/image/2099119.html
Photo Credit: https://pixabay.com/vectors/cranium-head-human-male-man-2099119/
Photo Credit: https://commons.wikimedia.org/wiki/File:Bunzl_truck,_Arvada,_CO.jpg (Xnatedawgx)
Photo Credit: https://commons.wikimedia.org/wiki/File:Kenworth_W900_semi_in_red.jpg (PRA)
Photo Credit: https://pixabay.com/en/truck-semi-truck-semi-truck-desert-1499377/

TEMPLATE MATCHING:
OBJECT (TRUCK) DETECTION

- Our goal is to find objects (truck) in this image using template matching.
- OpenCV has functions to perform this easily: **cv2.matchTemplate()**, **cv2.minMaxLoc()**

TEMPLATE

OBJECT DETECTED!

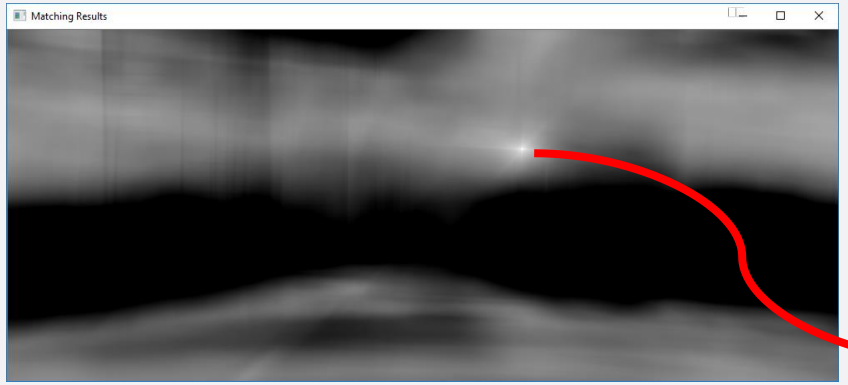
Sundog Education

sundog-education.com

Reference: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_template_matching/py_template_matching.html

TEMPLATE MATCHING:
OBJECT (TRUCK) DETECTION

- **cv2.matchTemplate()** simply slides the template image over the input image (as in 2D convolution) and compares the template and patch of input image under the template image.
- The function returns a grayscale image, where each pixel denotes how much does the neighbourhood of that pixel match with template.
- If input image is of size *(WxH)* and template image is of size *(wxh)*, output image will have a size of *(W-w+1, H-h+1)*.
- Once you got the result, **cv2.minMaxLoc()** function is used to find where is the maximum/minimum value. Take it as the top-left corner of rectangle and take *(w,h)* as width and height of the rectangle. That rectangle is the region of template.



TOP LEFT CORNER

TEMPLATE MATCHING:
OBJECT (TRUCK) DETECTION

- **cv2.matchTemplate()** simply slides the template image over the input image using one of the methods: ***method=CV_TM_CCORR_NORMED***
- The function slides through image **I**, compare it to the template **T** and generate a result image **R**
- The summation is done over template and/or the image patch: $x' = 0...w-1, y' = 0...h-1$

$$R(x,y) = \frac{\sum_{x',y'}(T(x',y') \cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

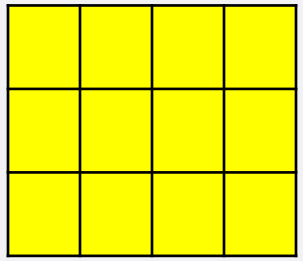
- **Check out the OpenCV documentation for other template matching methods:** https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_template_matching/py_template_matching.html

TEMPLATE MATCHING:
OBJECT (TRUCK) DETECTION

- Template has to be in the same orientation as in the original image (No rotation is allowed).
- Image sizing and scale is a challenge.
- Driving conditions such as weather, brightness and contrast.
- Perspective will challenge the technique.

IMAGE FEATURES
CORNERS AND EDGES AS FEATURES

- Edges are identified when change in intensity is noticed in one direction.
- Corners are identified when shifting a window in any direction over that point gives a large change in intensity in all directions.



EDGE DETECTED

CORNER DETECTED

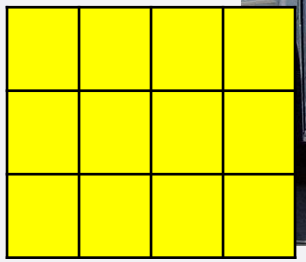


Photo Credit: https://commons.wikimedia.org/wiki/File:FedEx_truck_2.jpg

IMAGE FEATURES
CORNERS AND EDGES AS FEATURES

- Corners are regions in the image with large variation in intensity in all the directions.
- Harris corner detection finds the difference in intensity for a displacement of (u, v) in all directions.
- OpenCV has the function ***cv2.cornerHarris(img, block size, ksize, k)***
  - img - Input image, it should be grayscale and float32 type.
  - blockSize - It is the size of neighbourhood considered for corner detection
  - ksize - Aperture parameter of Sobel derivative used.
  - k - Harris detector free parameter in the equation (set to 0.1).

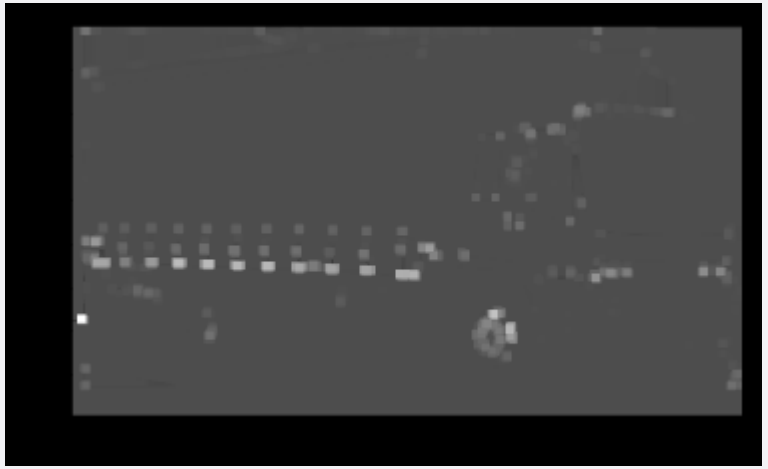
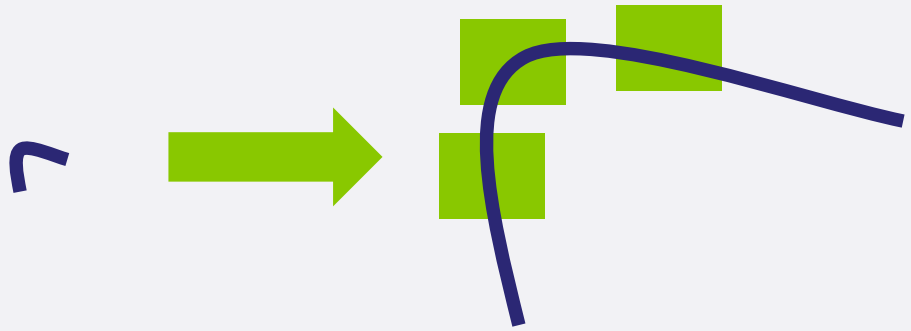


Photo Credit: https://commons.wikimedia.org/wiki/File:Truckrr.png (Limtu)

IMAGE FEATURES
CORNERS DETECTION LIMITATIONS

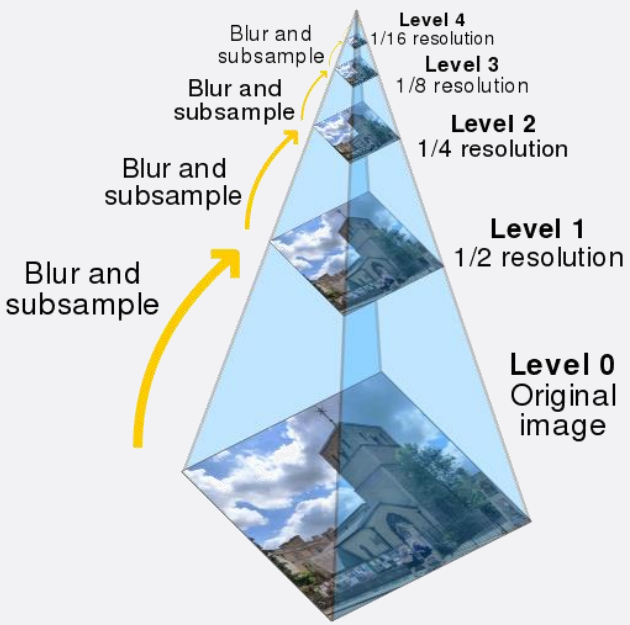
- Detecting corners as features in images can work well even of the image is:
  - Rotated, Translated and experienced changes in brightness
  - I.e.: even if the image is rotated, we can still find the same corners.
- The technique is challenged if the image is enlarged (scaling issues).
  - A corner may not be a corner if the image is scaled.
  - A corner in a small image would result in multiple corners in a zoomed-in larger image.



Reference: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html

IMAGE PYRAMIDING
INTUITION

- Image pyramiding refers to resizing the image by enlarging or shrinking.
- Pyramiding is important in object detection since it allows us to search for the object at various scales.
- An image pyramid is a collection of images - all arising from a single original image - that are successively down sampled until some desired stopping point is reached.
- By doing so, a M×N image becomes M/2×N/2 image. So area reduces to one-fourth of original area.



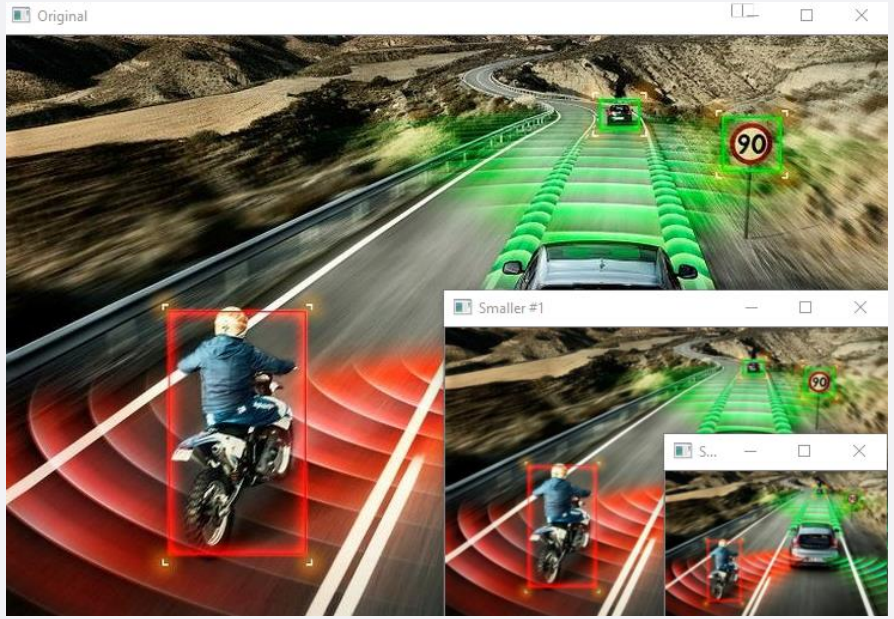


Photo Credit: https://commons.wikimedia.org/wiki/File:Image_pyramid.svg
Photo Credit: https://www.flickr.com/photos/eurosporttuning/16442169022

IMAGE PYRAMIDING
INTUITION

- Pyramiding is used for template matching by finding an object at different scales
- Gaussian pyramiding is one of the common types of image pyramiding and is used to down sample images
- Filtering the image, then subsample enhances the image quality (smoothness).
- Gaussian pyramids steps:
  (1)Convolve the image with a Gaussian kernel
  (2)Remove every even-numbered row and column

HISTOGRAM OF COLORS
INTUITION

- We can use the color histogram for feature detection, OpenCV has a histogram function as follows:

  cv2.calcHist([image], [channels], mask, [histSize], [ranges])

- images: source image
- channels: histogram index of channel, i.e.: [0], [1] or [2] for blue, green or red channel respectively.
- mask: put "None“ for the entire image.
- histSize : bin count, put [256] for full scale.
- ranges: put [0,256].

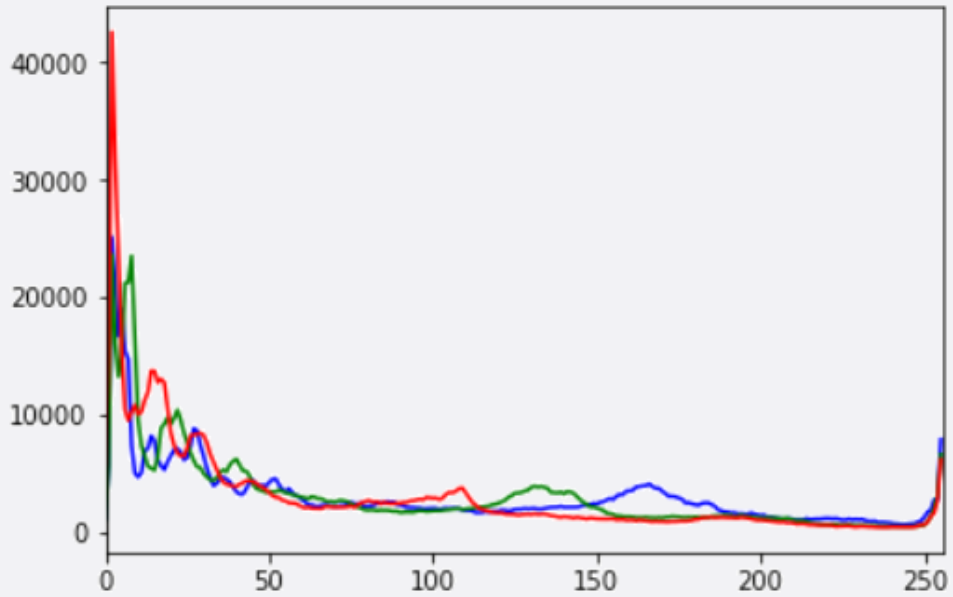
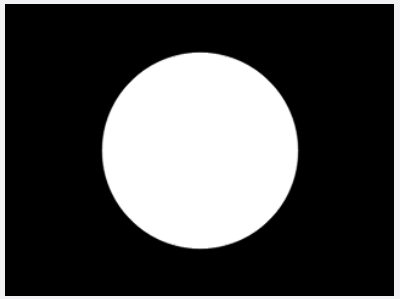


Photo Credit: https://www.flickr.com/photos/jurvetson/7408464122

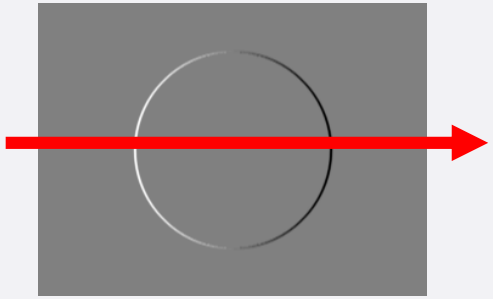
HISTOGRAM OF ORIENTED GRADIENTS
INTUITION

- For function $f(x, y)$, the gradient is the vector (f_x, f_y).
- An image is a discrete function of (x, y) so image gradient can be calculated as well.
- At each pixel, image gradient horizontal (x-direction) and vertical (y-direction) are calculated.
- These vectors have a direction $\text{atan}(\frac{f_y}{f_x})$ and a magnitude $(\sqrt{(f_x^2 + f_y^2)}$
- Gradient values are mapped to 0 - 255. Pixels with large negative change will be black, pixels with large positive change will be white, and pixels with little or no change will be gray.

ORIGINAL IMAGE



HORIZONTAL GRADIENT



VERTICAL GRADIENT

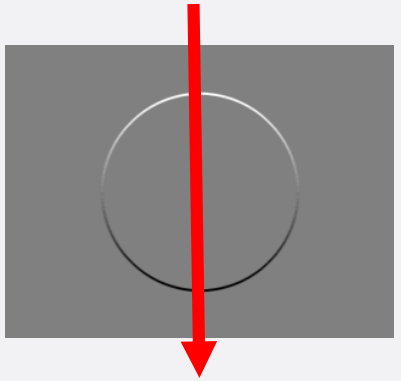
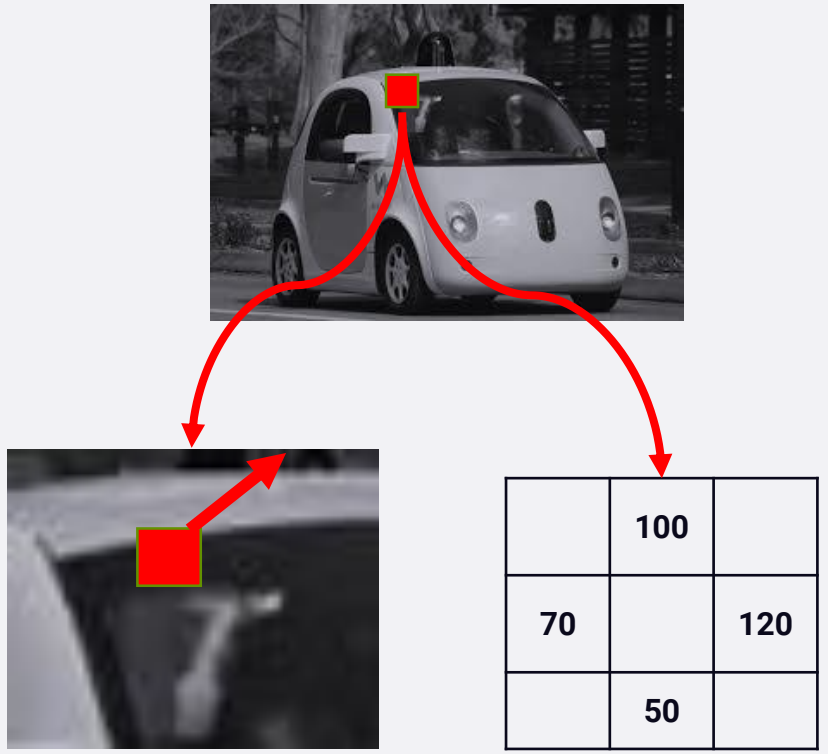


Photo Credit: https://stackoverflow.com/questions/19815732/what-is-gradient-orientation-and-gradient-magnitude/19816362#19816362

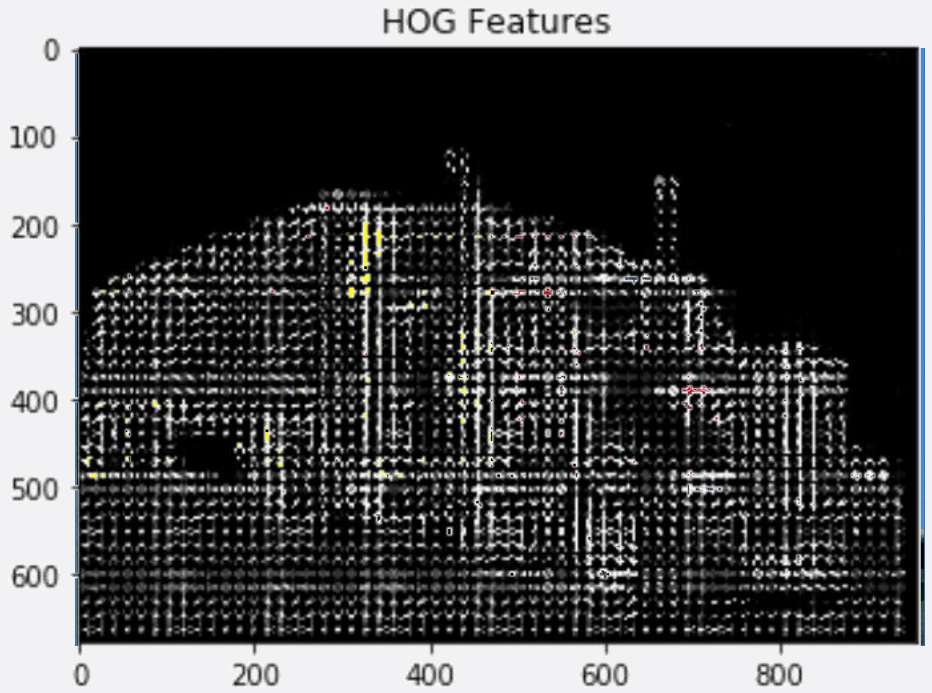
HISTOGRAM OF ORIENTED GRADIENTS
INTUITION

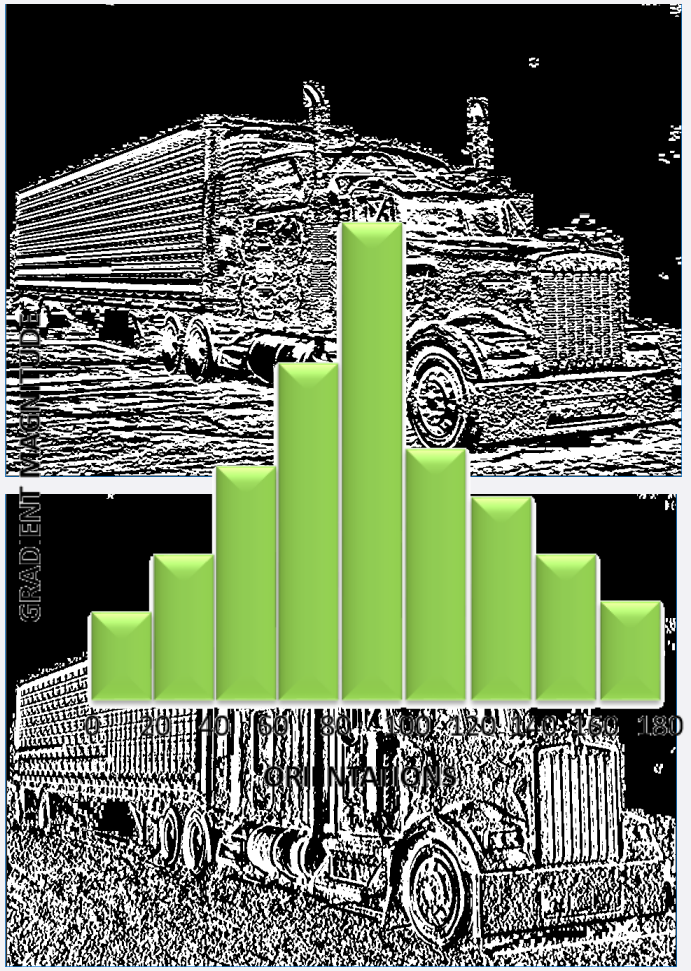
- The gradient value in the X-direction is 120-70=50
- Y-direction is 100-50=50.
- Putting it together we will have [50 50] feature vector.
- The magnitude and direction are calculated as follows:



Gradient Magnitude = $\sqrt{(50)^2 + (50)^2} = 70.1$
Gradient Angle = $\tan^{-1}(50/50) = 45°$

HISTOGRAM OF ORIENTED GRADIENTS
STEPS





Step 1:

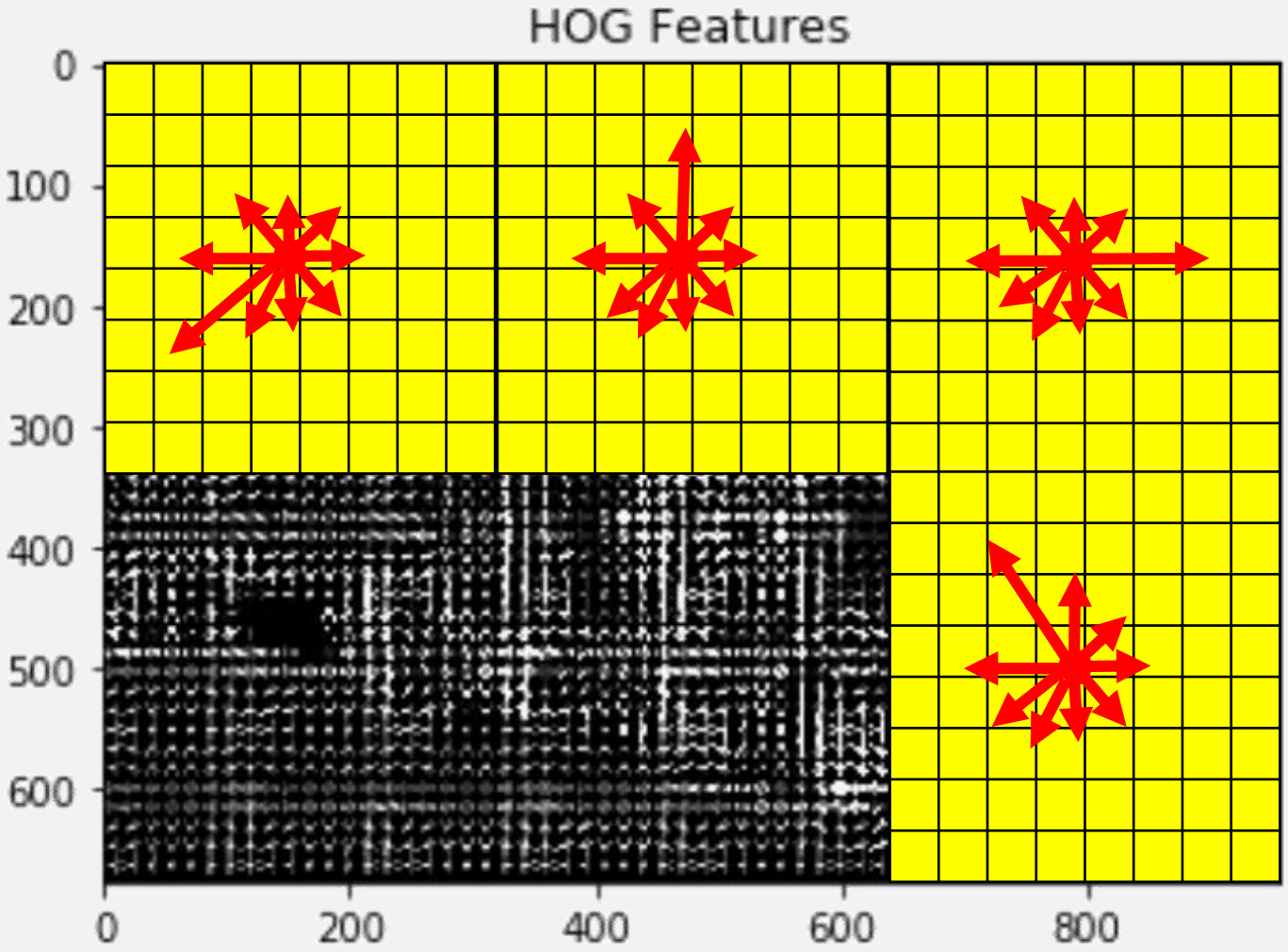
1. Using 8 x 8 pixel cell, compute gradient mag/direction

Step 2:

- Create a histogram of generated 64 (8 x 8) gradient vectors
- Each cell is then split into angular bins, each bin corresponds to a gradient direction (9 bins 0-180°) (20° each bin).
- This reduces 64 vectors to just 9 values.

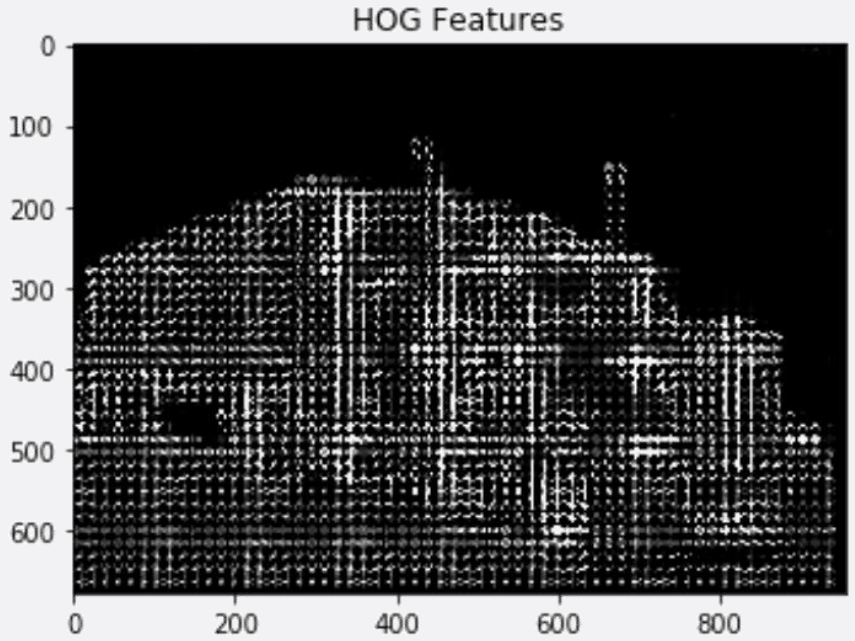
Photo Credit: https://commons.wikimedia.org/wiki/File:Red_truck_USA.JPG

HISTOGRAM OF ORIENTED GRADIENTS
EXPECTED HOG OUTPUT



HISTOGRAM OF ORIENTED GRADIENTS
SUMMARY

- HOGs are a feature descriptor used for object detection.
- HOGs along with SVM (support vector machine) classifiers work well for object detection.
- HOG applies a sliding window detector over an image and HOG descriptor is computed for each position.
- HOG takes care of the scaling issues by changing image scale (pyramiding).



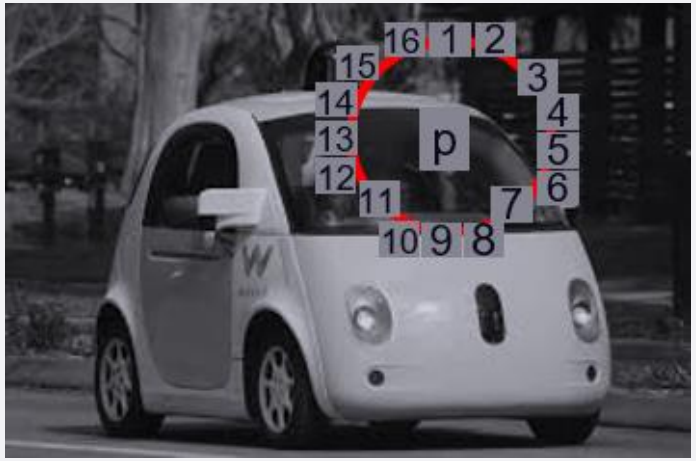
FAST/SURF/ORB/SIFT FEATURE DETECTORS
INTUITION

- FAST (Features from Accelerated Segment Test) algorithm was proposed by Edward Rosten and Tom Drummond in their paper “Machine learning for high-speed corner detection” in 2006 (Later revised it in 2010).

STEPS:

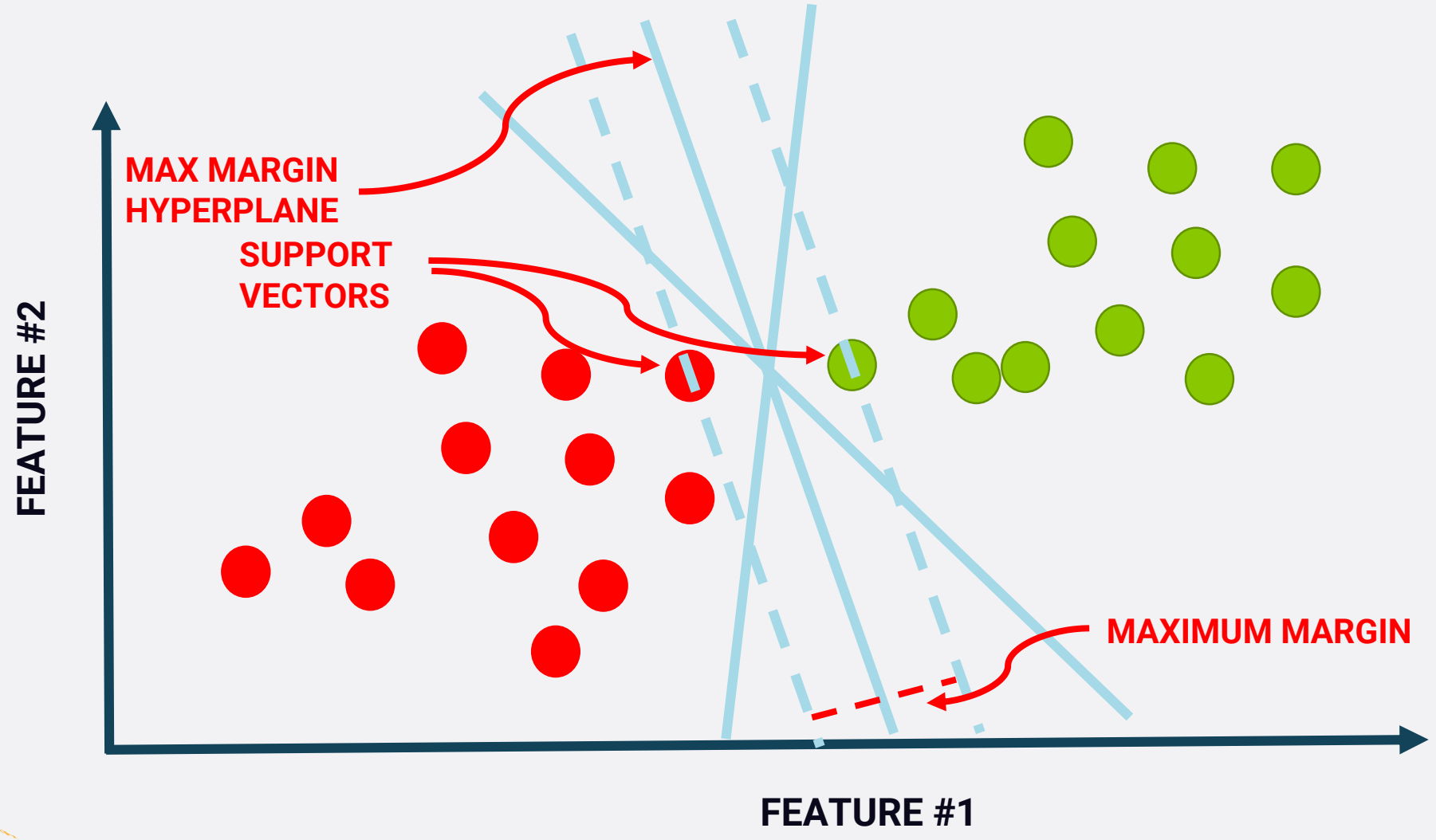
1. Select a pixel p in the image which is to be identified as an interest point
2. Get its intensity I_p.
3. Select appropriate threshold value t.
4. Consider a circle of 16 pixels around the pixel under test.
5. Pixel p is a corner if there exists a set of contiguous pixels in the circle which are all brighter than $I_p + t$, or all darker than $I_p - t$.
6. A high-speed test that only four pixels at 1, 9, 5 and 13. If p is a corner, then at least three of these must all be brighter than $I_p + t$ or darker than $I_p - t$. If neither of these is the case, then p cannot be a corner.

- Other feature detectors: SIFT, SURF, ORB (Oriented FAST and Rotated BRIEF an efficient alternative to SIFT or SURF proposed in 2011)

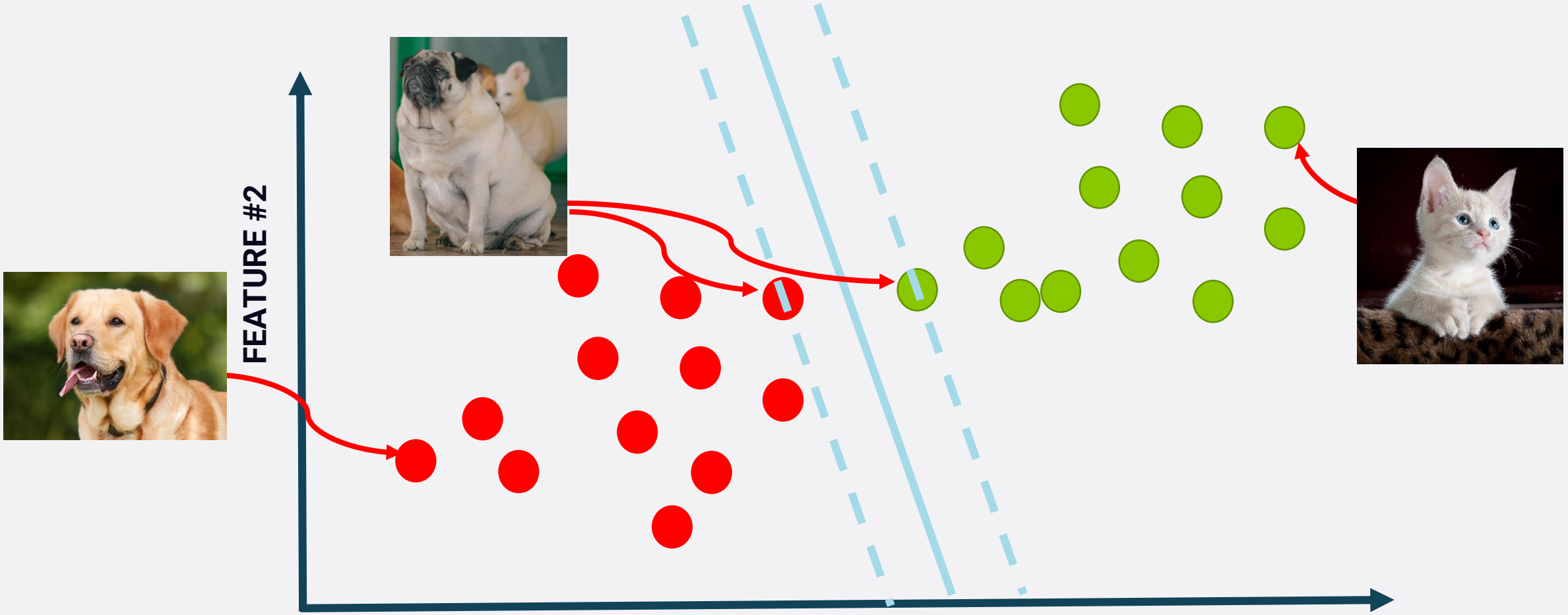


Reference: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_fast/py_fast.html
Photo Credit: https://commons.wikimedia.org/wiki/File:Waymo_self-driving_car_front_view.gk.jpg (Grendelkhan)

SUPPORT VECTOR MACHINE
INTUITION



SUPPORT VECTOR MACHINE
INTUITION

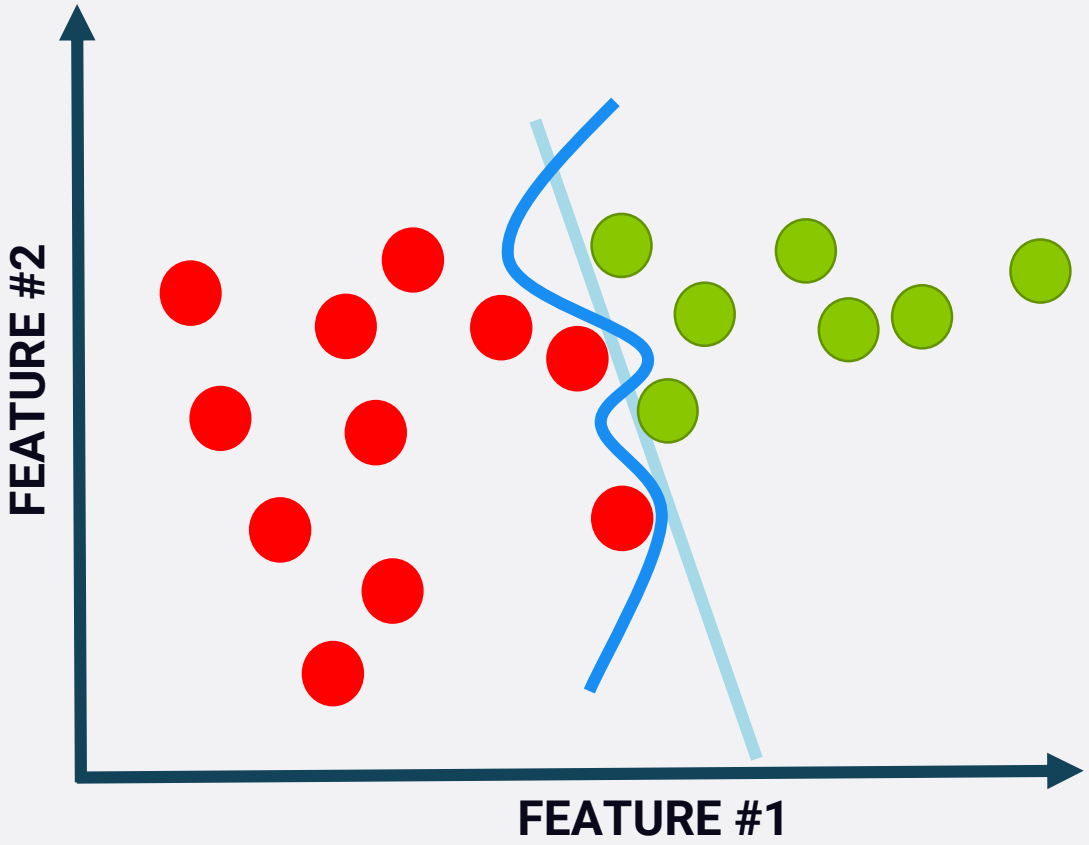


- Photo Credit: https://pixabay.com/photos/dog-labrador-light-brown-pet-1210559/
- Photo Credit: https://www.pexels.com/photo/animal-pet-cute-kitten-45201/
- Photo Credit: https://pxhere.com/en/photo/1424251

SUPPORT VECTOR MACHINE
PARAMETERS OPTIMIZATION

C parameter: Controls trade-off between classifying training points correctly and having a smooth decision boundary

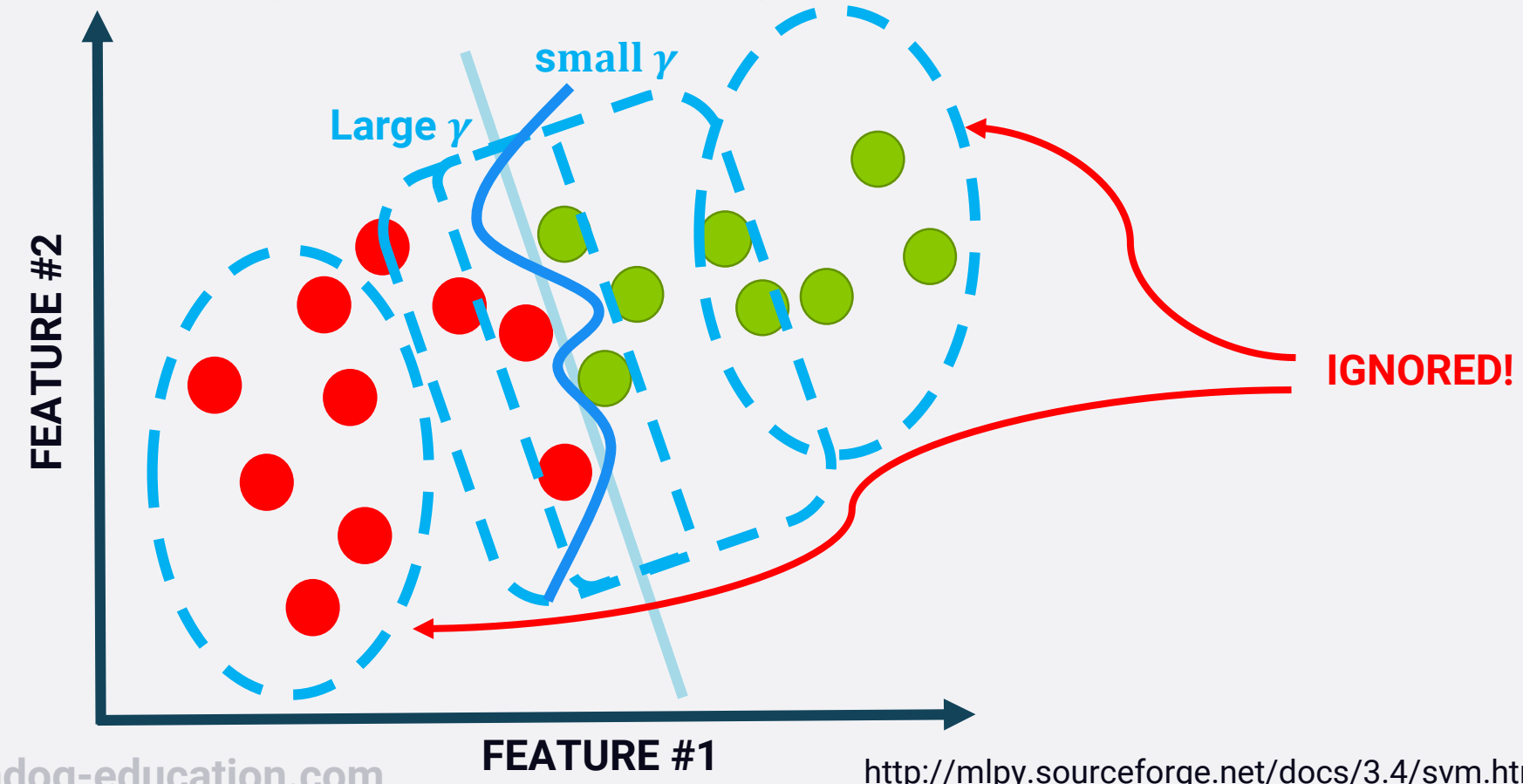
- **Small C (loose)** makes cost (penalty) of misclassification low (soft margin)
- **Large C (strict)** makes cost of misclassification high (hard margin), forcing the model to explain input data stricter and potentially over fit.



SUPPORT VECTOR MACHINE
PARAMETERS OPTIMIZATION

Gamma parameter: controls how far the influence of a single training set reaches

- **Large gamma:** close reach (closer data points have high weight)
- **Small gamma:** far reach (more generalized solution)



http://mlpy.sourceforge.net/docs/3.4/svm.html

what is machine learning?

training a model

machine learning systems **train** a **model** by feeding it **feature data** with known **labels**

features are the attributes used to inform your model's predictions
labels are the "correct answers" our model wants to be able to predict on its own

once the model is built, the model will **predict** labels just given feature data

features are **independent variables**, labels are **dependent variables**.

example: predicting sale price of a house

if we have a database of past house sales, we can **train** a **machine learning model** with it.

the **features** could be number of bedrooms, postal code, square footage, amount of land, number of bathrooms, etc.

the **labels** are the sale prices

once trained, this model can try to **predict** sale prices for houses that haven't sold yet, based only on their feature data.

example: classifying images of street signs

if we have a database of images taken from vehicles that contain known street signs, we can **train** our model with it.

the **features** in this case are the individual pixels from every image

the **labels** are the types of signs present in the image (school crossing, in this example)

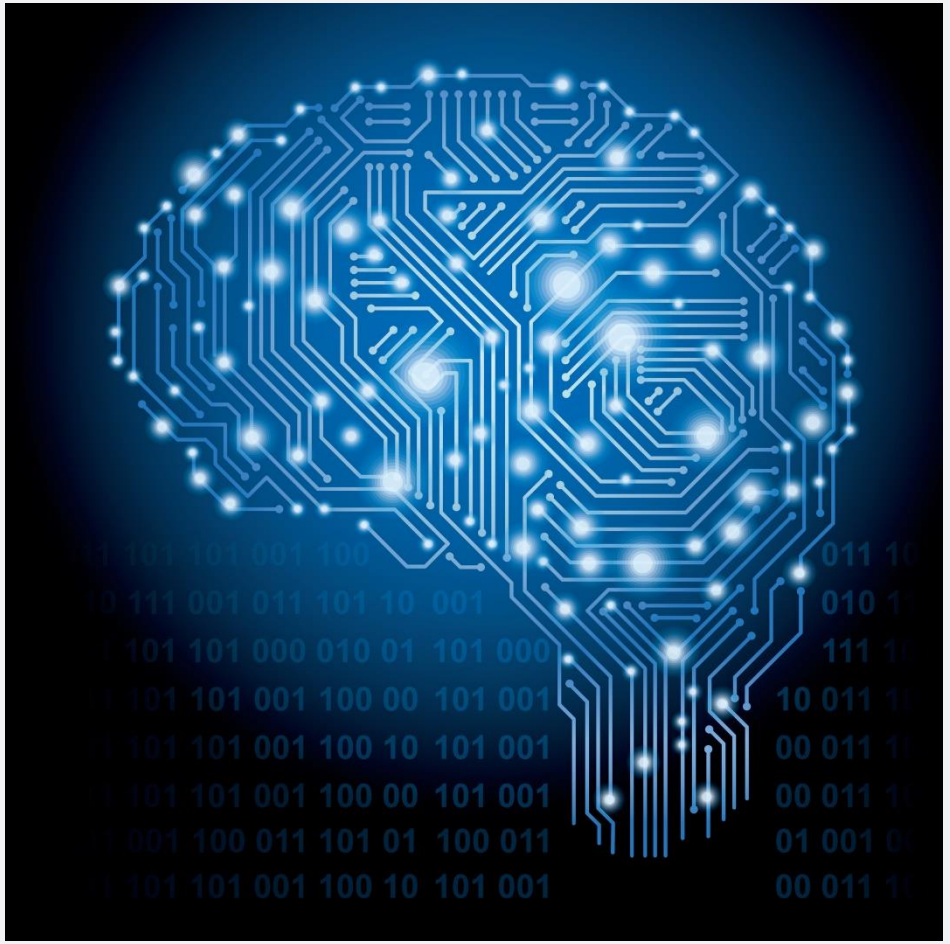
once trained, our model can identify what signs are present in a real-time image from a self-driving car, for example.

how do machine learning models work?

many different approaches, but generally speaking they tweak parameters of some mathematical model to minimize prediction errors as they are trained with new data.

neural networks, or “deep learning,” are just the latest approach, and are one of many machine learning techniques.

others include reinforcement learning, support vector machines, and decision trees.



evaluating machine learning systems

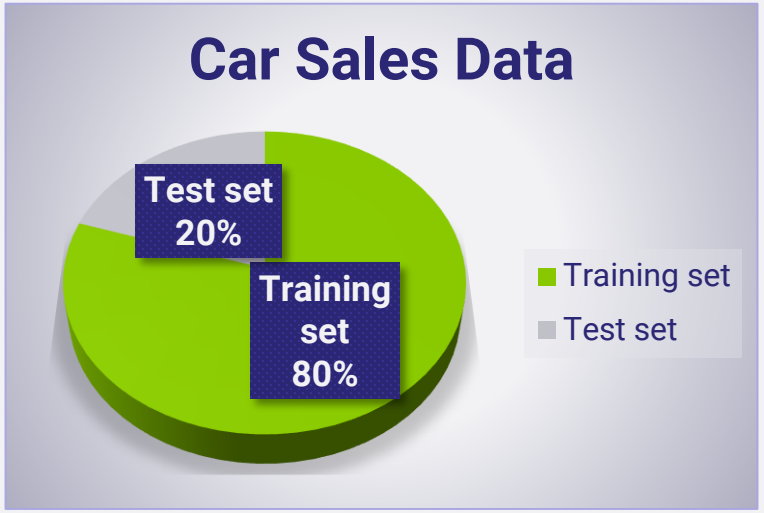
train/test

if you have a set of training data that includes the value you're trying to predict – you don't have to guess if the resulting model is good or not.

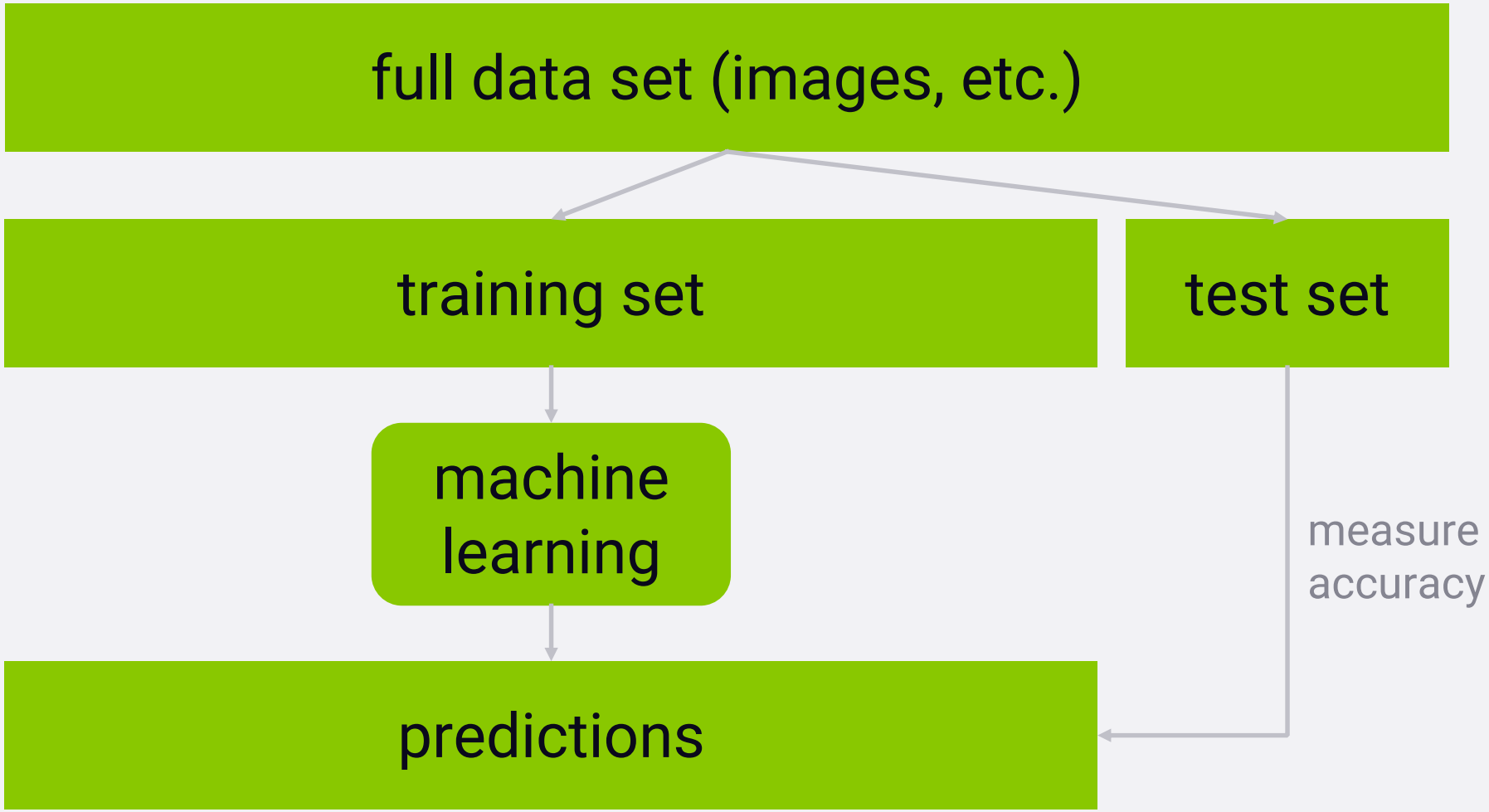
if you have enough training data, you can split it into two parts: a training set and a test set.

you then train the model using only the training set

and then measure (using r-squared or some other metric) the model's accuracy by asking it to predict values for the test set, and compare that to the known, true values.

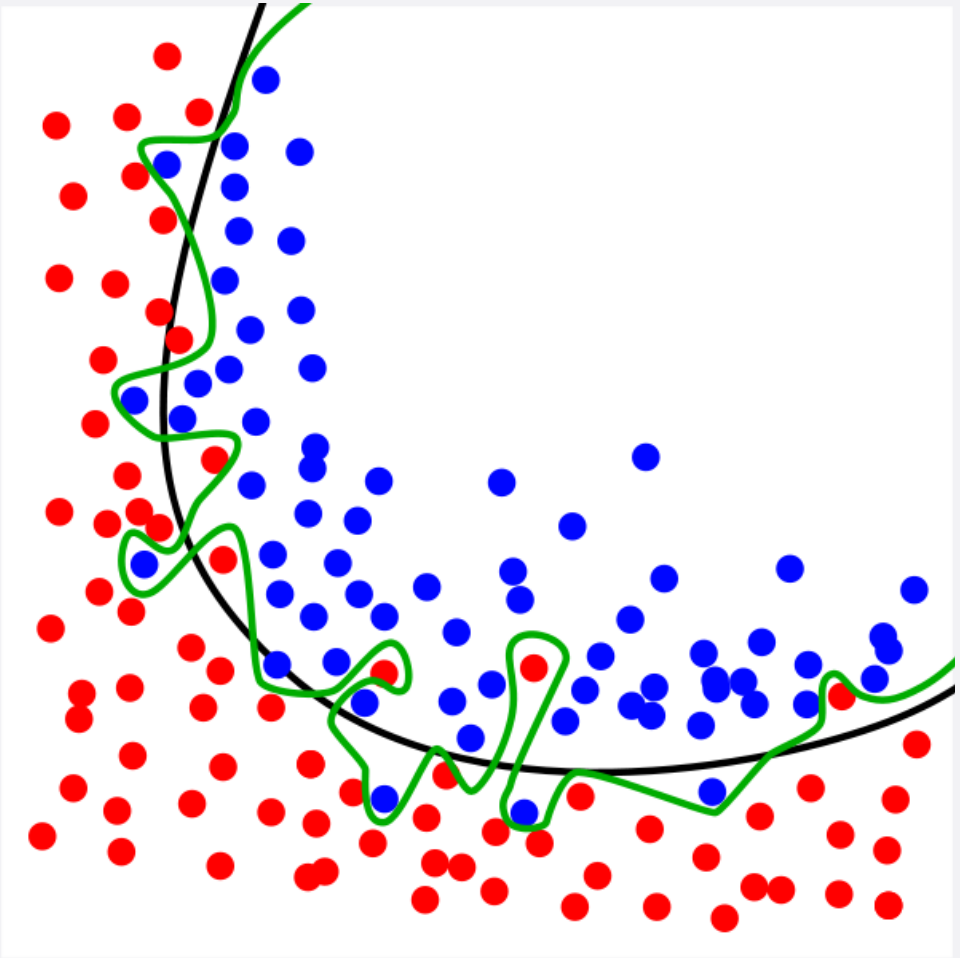


train/test


full data set (images, etc.)
training set
test set
machine learning
measure accuracy
predictions

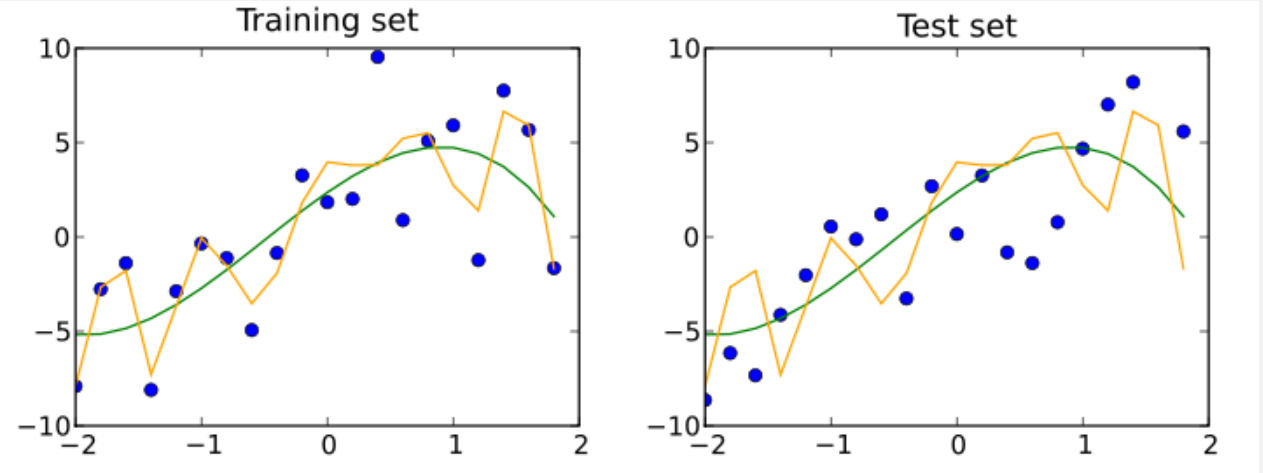
train / test in practice

- need to ensure both sets are large enough to contain representatives of all the variations and outliers in the data you care about
- the data sets must be selected randomly
- train/test is a great way to guard against *overfitting*



train/test is not infallible

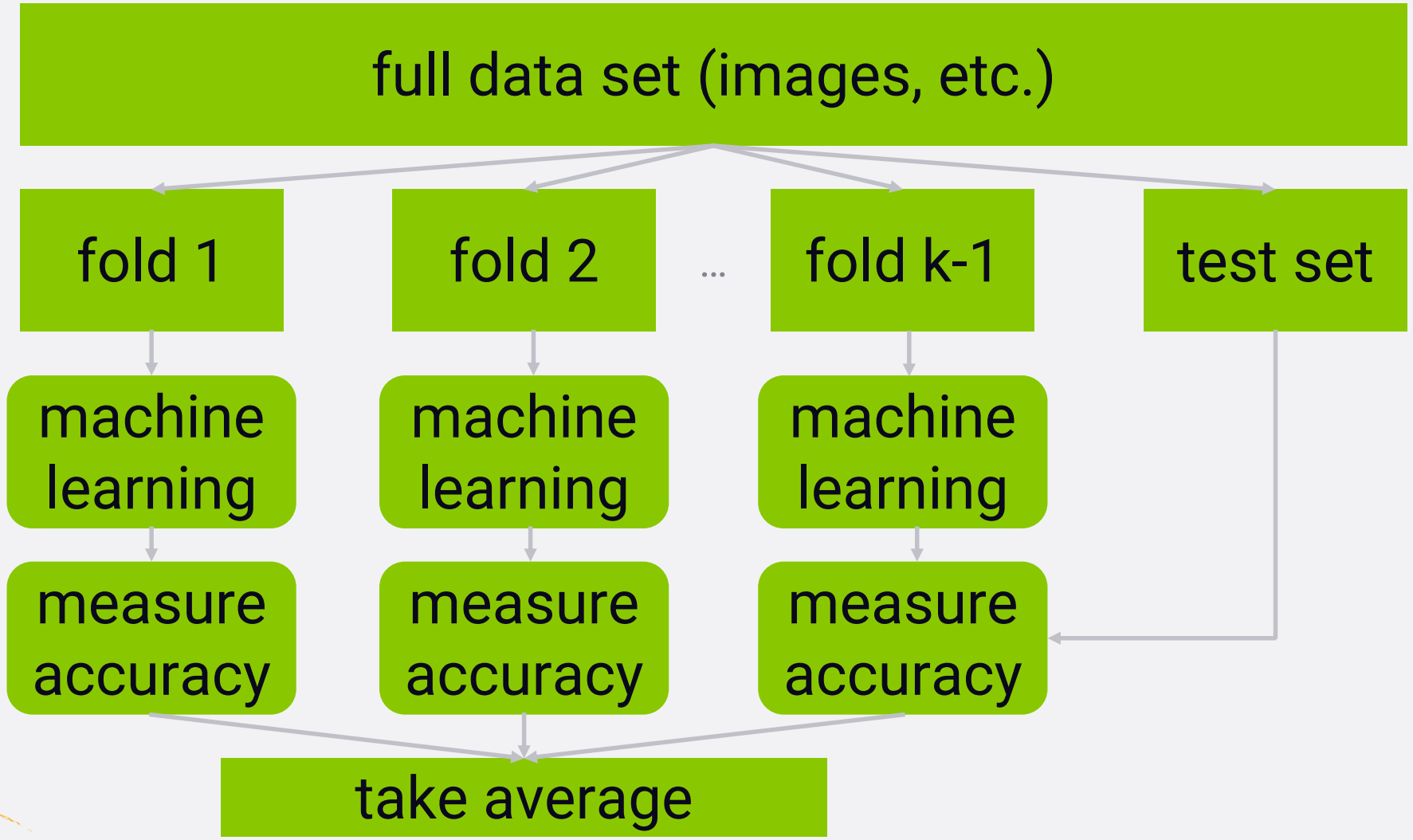
- maybe your sample sizes are too small
- or due to random chance your train and test sets look remarkably similar
- overfitting can still happen



k-fold cross validation

- one way to further protect against overfitting is *K-fold cross validation*
- sounds complicated, but it's a simple idea:
  - split your data into K randomly-assigned segments
  - reserve one segment as your test data
  - train on each of the remaining K-1 segments and measure their performance against the test set
  - take the average of the K-1 r-squared scores

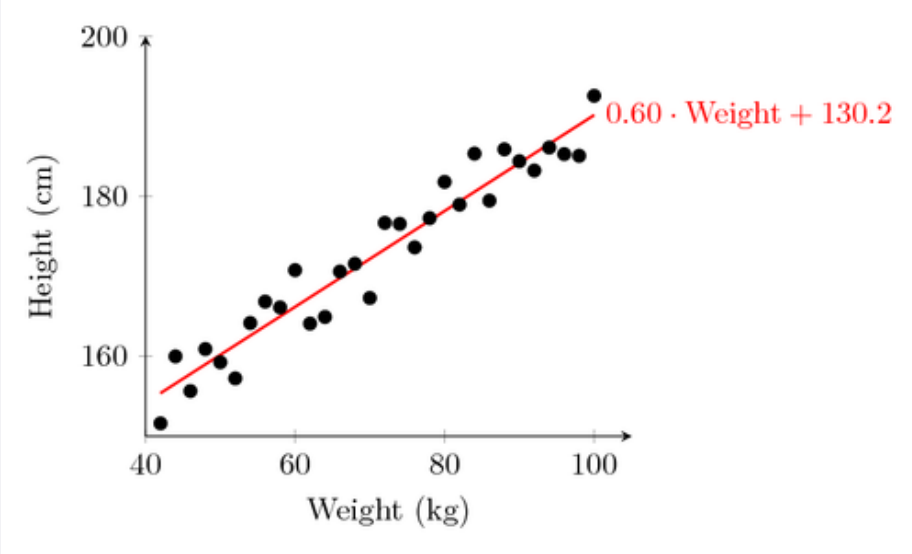
k-fold cross-validation



linear regression

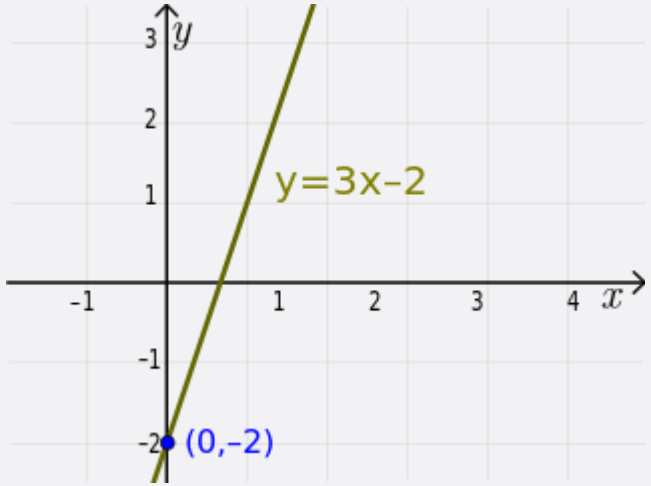
linear regression

- fit a line to a data set of observations
- use this line to predict unobserved values
- the term “regression” is confusing



linear regression: how does it work?

- usually using “ordinary least squares”
- minimizes the squared-error between each point and the line
- remember the slope-intercept equation of a line? y=mx+b
- the slope is the correlation between the two variables times the standard deviation in Y, all divided by the standard deviation in X.
  - neat how standard deviation how some real mathematical meaning, eh?
- the intercept is the mean of Y minus the slope times the mean of X
- but python will do all that for you.

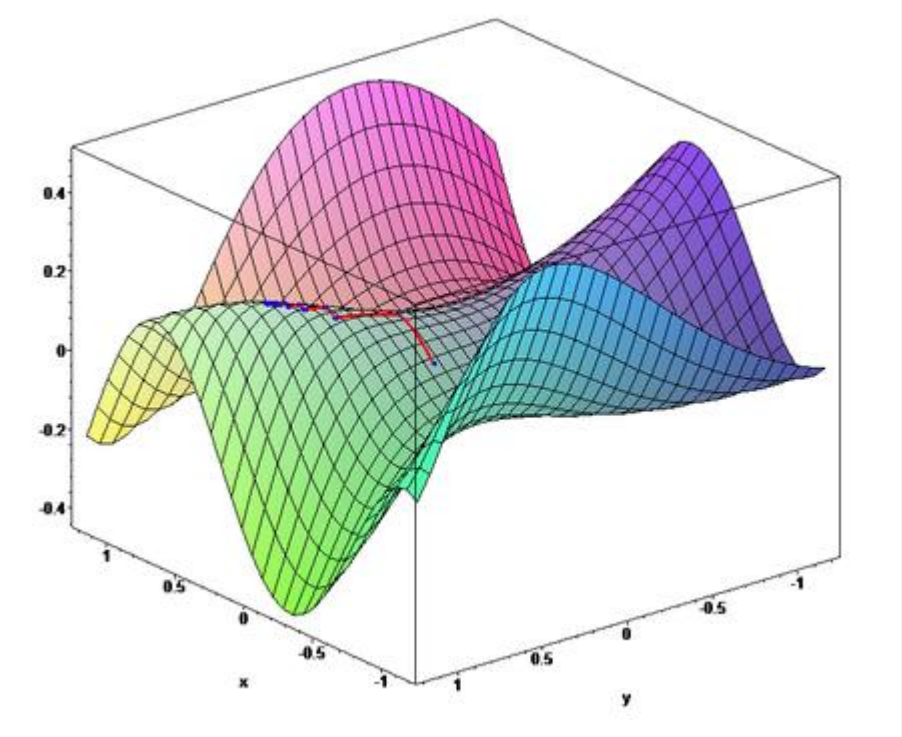


linear regression: how does it work?

- least squares minimizes the sum of squared errors.
- this is the same as maximizing the likelihood of the observed data if you start thinking of the problem in terms of probabilities and probability distribution functions
- this is sometimes called “maximum likelihood estimation”

more than one way to do it

- gradient descent is an alternate method to ordinary least squares.
- basically iterates to find the line that best follows the contours defined by the data.
- can make sense when dealing with multi-dimensional data
- popular in neural networks



measuring error with r-squared

- how do we measure how well our line fits our data?
- R-squared (aka coefficient of determination) measures:

The fraction of the total variation in Y that is captured by the model

computing r-squared

$$1.0 - \frac{sum\ of\ squared\ errors}{sum\ of\ squared\ variation\ from\ mean}$$

interpreting r-squared

- ranges from 0 to 1
- 0 is bad (none of the variance is captured), 1 is good (all of the variance is captured).

exercise

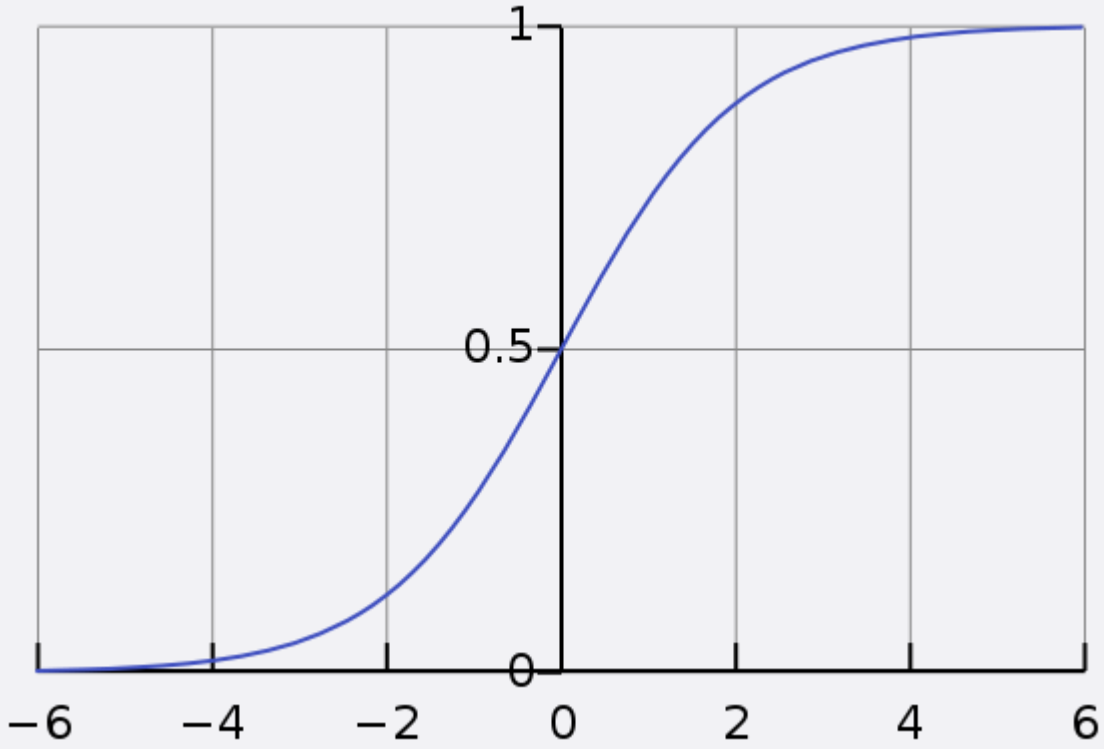
logistic regression

logistic regression

- regression for binary (yes/no) problems
- like linear regression, but final predictions are transformed into a 0-1 range

the sigmoid (logistic) function

$$y = \frac{1}{1 + e^{-(a+bx)}}$$



loss function for logistic regression

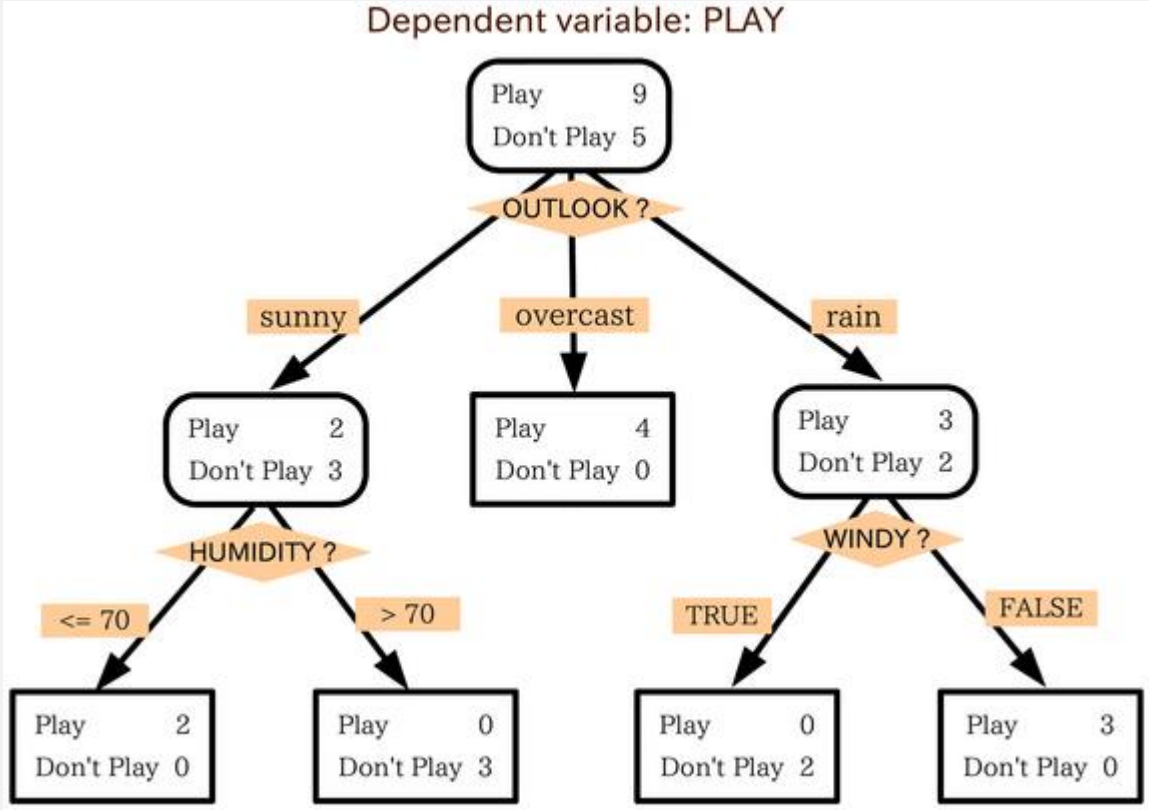
- least-squares won't work with the logistic curve
- instead we use gradient descent with maximum likelihood
- this just iterates to maximize the conditional probability of Y given X

exercise

decision trees

decision trees

- you can actually construct a flowchart to help you decide a classification for something with machine learning
- this is called a decision tree
- another form of supervised learning
  - Give it some sample data and the resulting classifications
  - Out comes a tree!



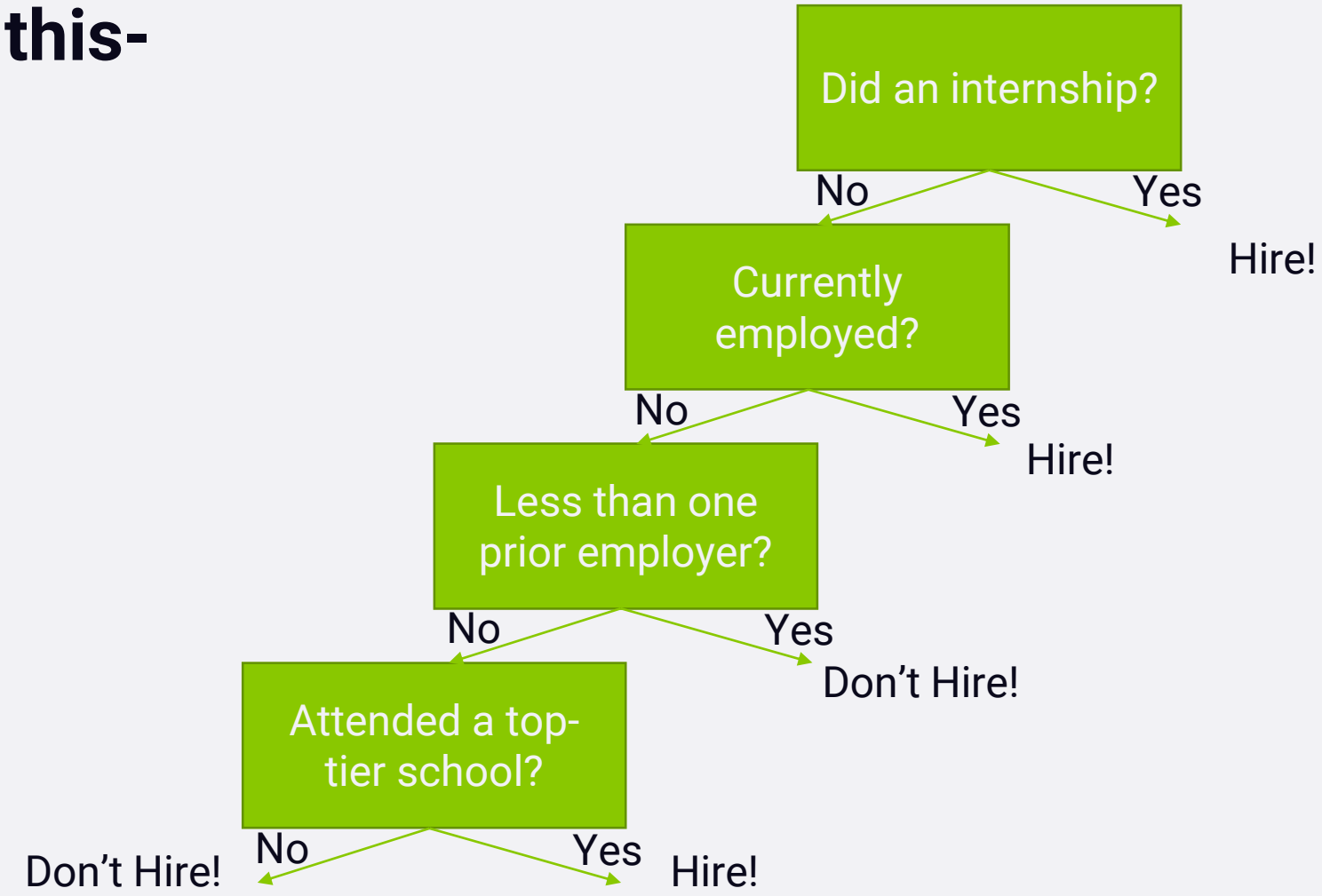
decision tree example

- you want to build a system to filter out resumes based on historical hiring data
- you have a database of some important attributes of job candidates, and you know which ones were hired and which ones weren't
- you can train a decision tree on this data, and arrive at a system for predicting whether a candidate will get hired based on it!

totally fabricated hiring data

Candidate ID	Years Experience	Employed?	Previous employers	Level of Education	Top-tier school	Interned	Hired
0	10	1	4	0	0	0	1
1	0	0	0	0	1	1	1
2	7	0	6	0	0	0	0
3	2	1	1	1	1	0	1
4	20	0	2	2	1	0	0

totally fabricated should-I-hire-this-person tree



how decision trees work

- at each step, find the attribute we can use to partition the data set to minimize the *entropy* of the data at the next step
- fancy term for this simple algorithm: ID3
- it is a *greedy algorithm* – as it goes down the tree, it just picks the decision that reduce entropy the most at that stage.
  - that might not actually result in an optimal tree.
  - but it works.

random forests

- decision trees are very susceptible to overfitting
- to fight this, we can construct several alternate decision trees and let them “vote” on the final classification
  - randomly re-sample the input data for each tree (fancy term for this: *bootstrap aggregating* or *bagging*)
  - randomize a subset of the attributes each step is allowed to choose from

exercise

naïve bayes

bayes' theorem

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)}$$

in english – the probability of A given B, is the probability of A times the probability of B given A over the probability of B.

the key insight is that the probability of something that depends on B depends very much on the base probability of B and A. people ignore this all the time.

bayes' theorem to the rescue

- drug testing is a common example. even a "highly accurate" drug test can produce more false positives than true positives.
- let's say we have a drug test that can accurately identify users of a drug 99% of the time, and accurately has a negative result for 99% of non-users. but only 0.3% of the overall population actually uses this drug.

bayes' theorem to the rescue

- event A = is a user of the drug, event B = tested positively for the drug.
- we can work out from that information that P(B) is 1.3% (0.99 * 0.003 + 0.01 * 0.997 – the probability of testing positive if you do use, plus the probability of testing positive if you don't.)
- $P(A|B) = \frac{P(A)P(B|A)}{P(B)} = \frac{0.003 * 0.99}{0.013} = 22.8\%$
- so the odds of someone being an actual user of the drug given that they tested positive is only 22.8%!
- even though P(B|A) is high (99%), it doesn't mean P(A|B) is high.

machine learning with naïve bayes

- $P(A|B) = \frac{P(A)P(B|A)}{P(B)}$
- let's use it for machine learning! i want a spam classifier.
- example: how would we express the probability of an email being spam if it contains the word "free"?
- $P(Spam \mid Free) = \frac{P(Spam)P(Free \mid Spam)}{P(Free)}$
- the numerator is the probability of a message being spam and containing the word "free" (this is subtly different from what we're looking for)
- the denominator is the overall probability of an email containing the word "free". (Equivalent to P(Free|Spam)P(Spam) + P(Free|Not Spam)P(Not Spam))
- so together – this ratio is the % of emails with the word "free" that are spam.

what about all the other words?

- we can construct P(Spam | Word) for every (meaningful) word we encounter during training
- then multiply these together when analyzing a new email to get the probability of it being spam.
- assumes the presence of different words are independent of each other – one reason this is called “naïve Bayes”.

sounds like a lot of work.

- scikit-learn to the rescue!
- the CountVectorizer lets us operate on lots of words at once, and MultinomialNB does all the heavy lifting on Naïve Bayes.
- we'll train it on known sets of spam and "ham" (non-spam) emails
  - So this is supervised learning!
- let's do this

exercise

support vector machines

support vector machines

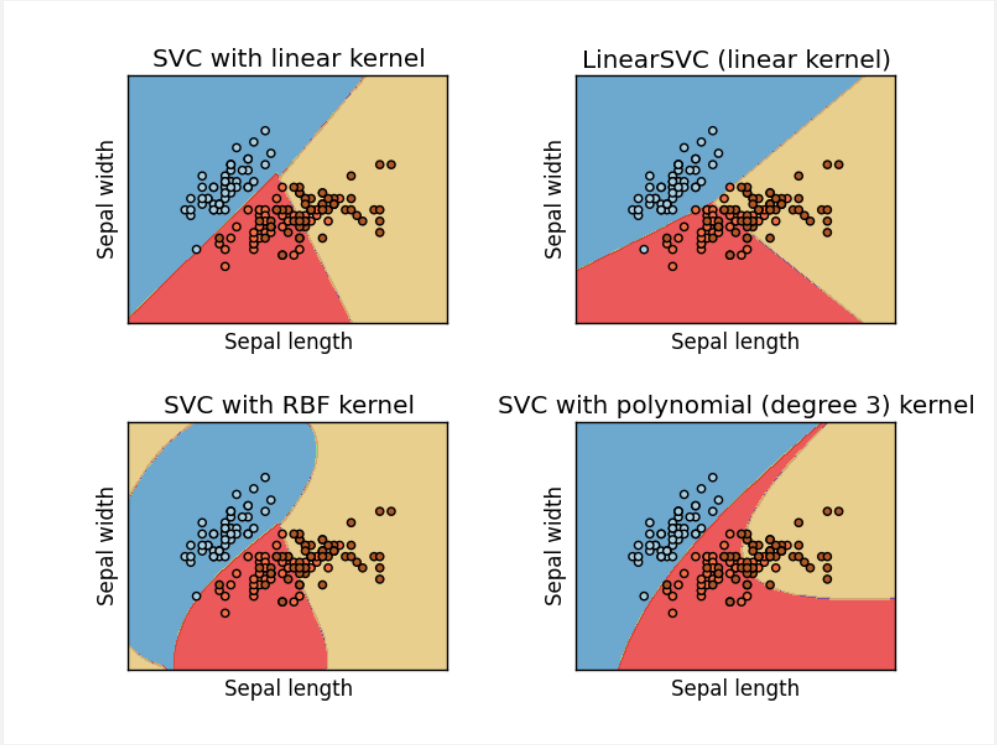
- works well for classifying higher-dimensional data (lots of features)
- finds higher-dimensional *support vectors* across which to divide the data (mathematically, these support vectors define hyperplanes. Needless to say I'm not going to get into the mathematical details!)
- uses something called the *kernel trick* to represent data in higher-dimensional spaces to find hyperplanes that might not be apparent in lower dimensions

higher dimensions? hyperplanes? huh?

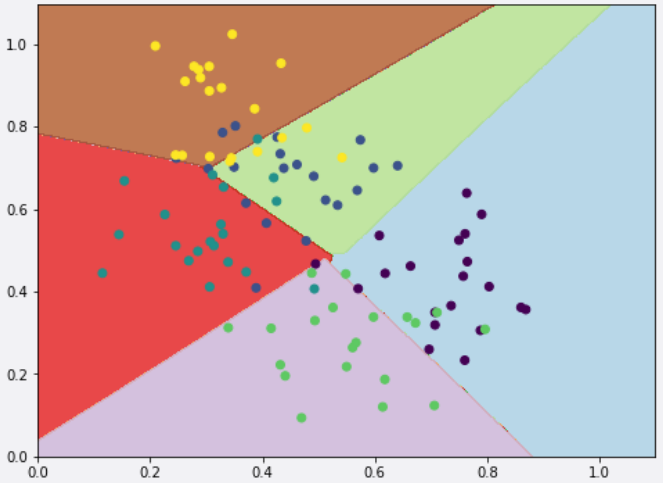
- the important point is that SVM's employ some advanced mathematical trickery to cluster data, and it can handle data sets with lots of features.
- it's also fairly expensive – the "kernel trick" is the only thing that makes it possible.

support vector classification

- in practice you'll use something called SVC to classify data using SVM.
- you can use different "kernels" with SVC. Some will work better than others for a given data set.

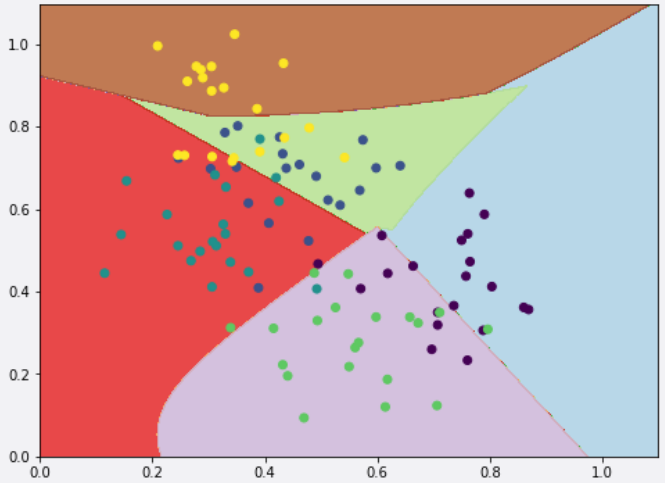


tuning kernels and gamma



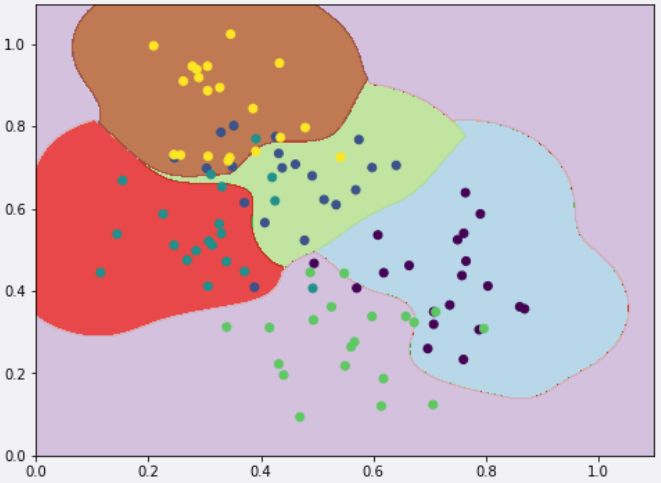
linear

$\langle x, x' \rangle$



poly

$(\gamma \langle x, x' \rangle + r)^d$

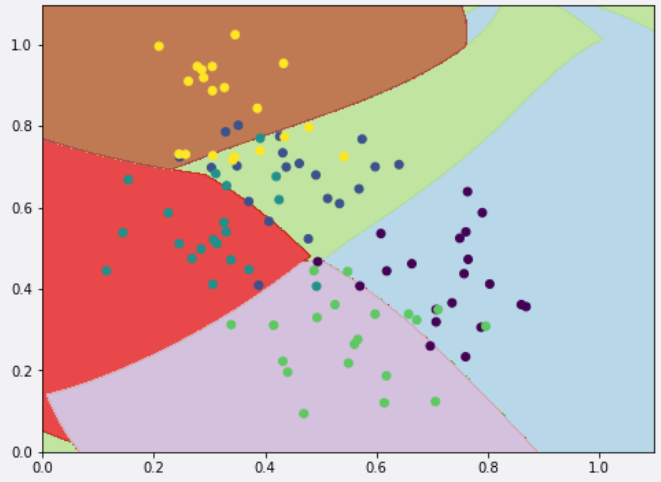


rbf

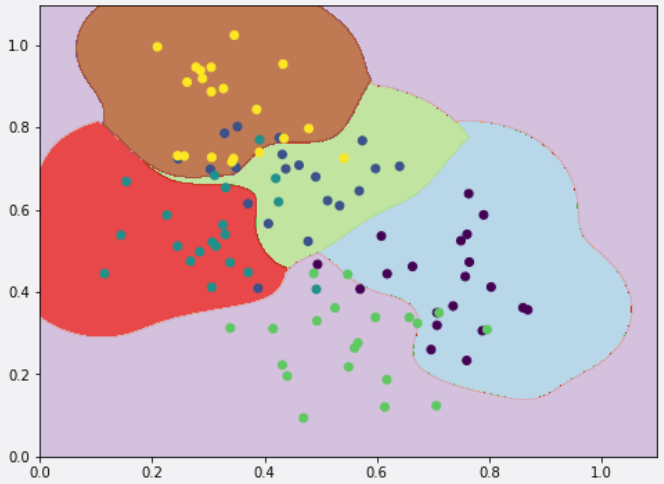
$e^{-\gamma \|x - x'\|^2}$

avoiding overfitting

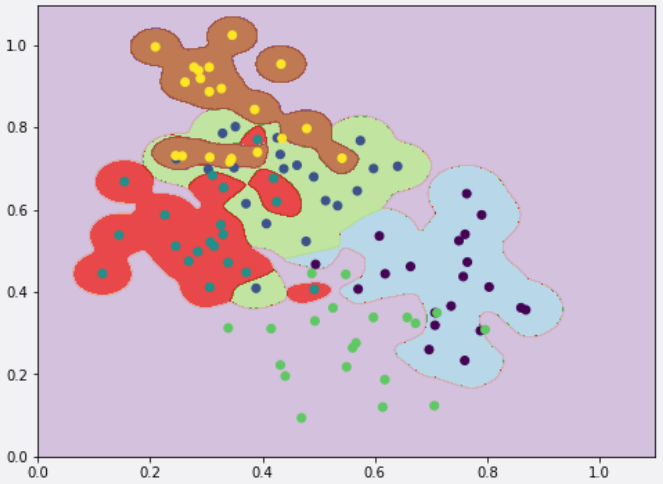
- don’t choose an overly complex kernel
- rbf is very sensitive to gamma values chosen
  - this specifies the area of influence of the support vectors
- use k-fold cross validation, GridSearchCV



$\gamma = 10.0$



$\gamma = 100.0$



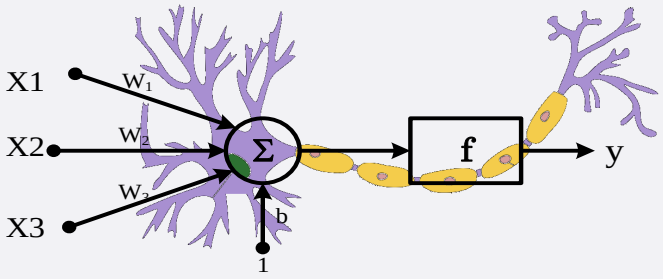
$\gamma = 1000.0$

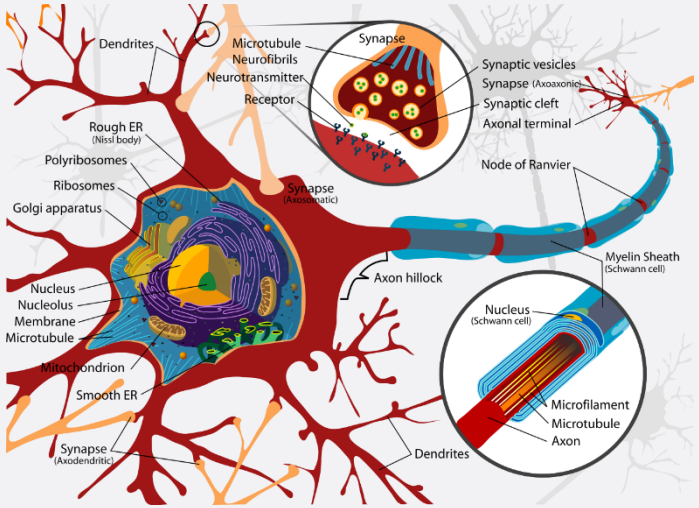
exercise

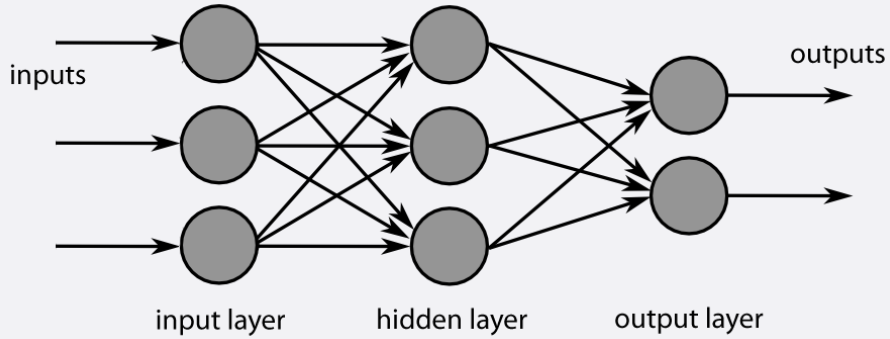
exercise

INTRODUCTION
WHAT IS AN ARTIFICIAL NEURAL NETWORK?

- ANNs are information processing models inspired by the human brain.
- The brain has over 100 billion neurons communicating through electrical and chemical signals.
- Neurons communicate with each other and help us see, think, and generate ideas.
- Human brain learns by creating connections among these neurons.







- **Photo Credit:** https://fr.m.wikipedia.org/wiki/Fichier:MultiLayerNeuralNetworkBigger_english.png
- **Photo Credit:** https://commons.wikimedia.org/wiki/File:Neuron_Hand-tuned.svg
- Photo Credit: https://commons.wikimedia.org/wiki/File:Complete_neuron_cell_diagram_en.svg

INTRODUCTION
HOW DO HUMANS LEARN? *SO WE CAN IMITATE THEM!*

- Humans learn from experience (by example).
- ANNs will learn in the same fashion, by showing several training samples over and over again ‘epochs’.

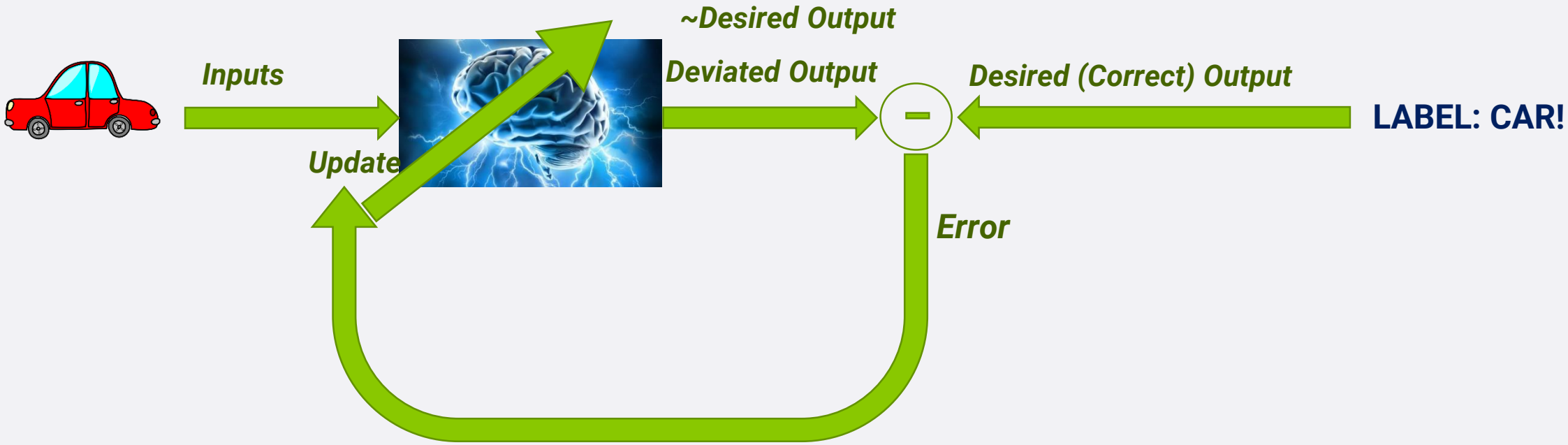
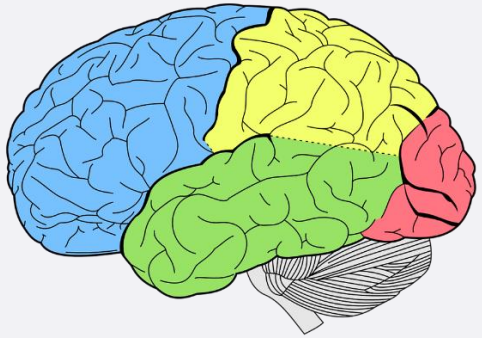


Photo Credit: https://www.flickr.com/photos/142299342@N06/32794072773
Photo Credit: https://www.modup.net/
Photo Credit: http://www.freestockphotos.biz/stockphoto/15685

INTRODUCTION
ANNs ADVANTAGES

- ANNs are capable of modeling relationships between a set of inputs and outputs.
- "*A neural network with sufficient number of neurons can approximately model any continuous function with an acceptable degree of accuracy*".
- Excellent generalization capability.



INTRODUCTION
ANNs LIMITATIONS

- Black-box structure
- Even though trained network can operate properly, the internal structure of the network often has no physical significance and very difficult to understand.

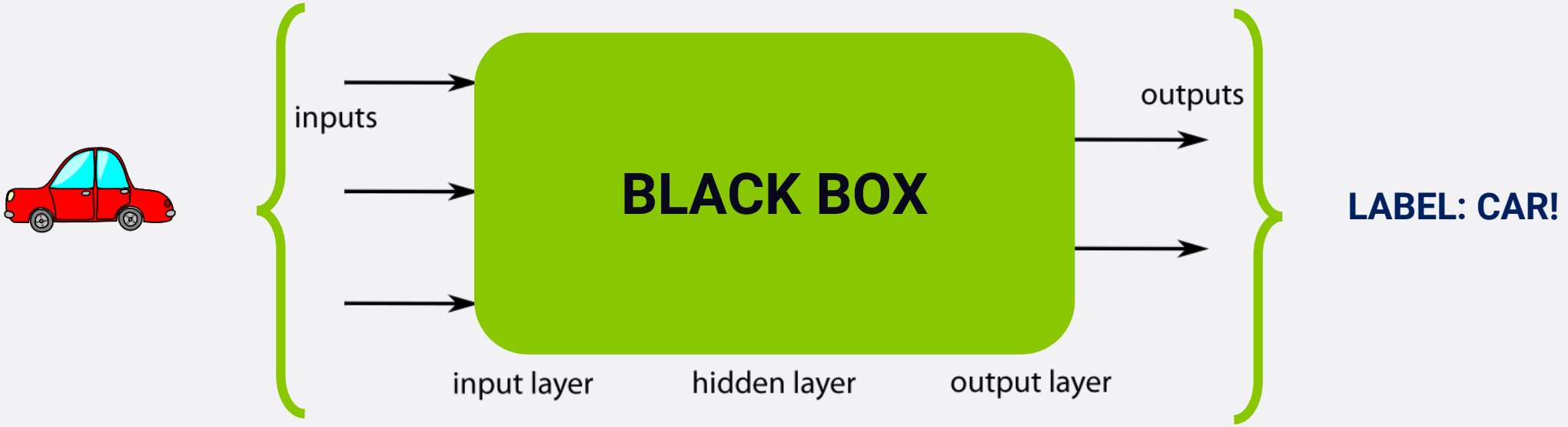
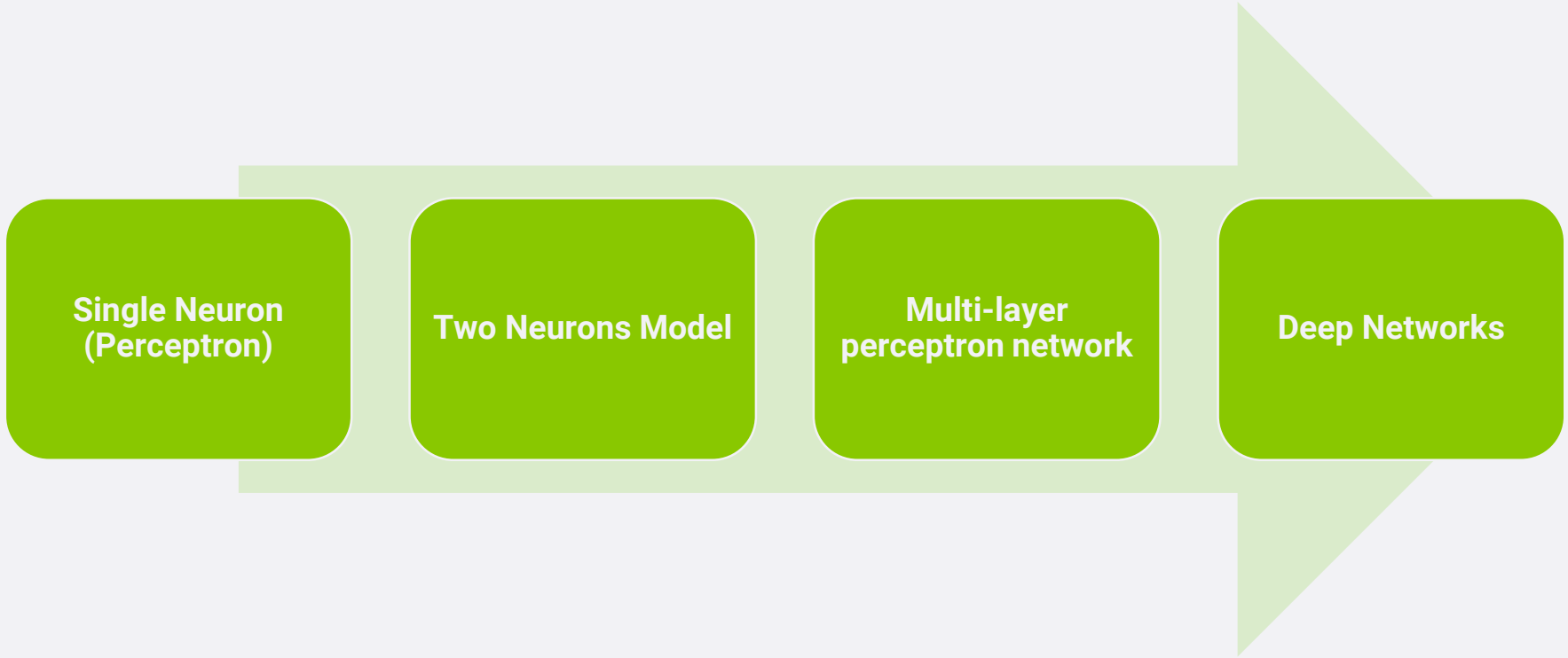


Photo Credit: http://www.freestockphotos.biz/stockphoto/15685

INTRODUCTION
ANNs APPLICATIONS

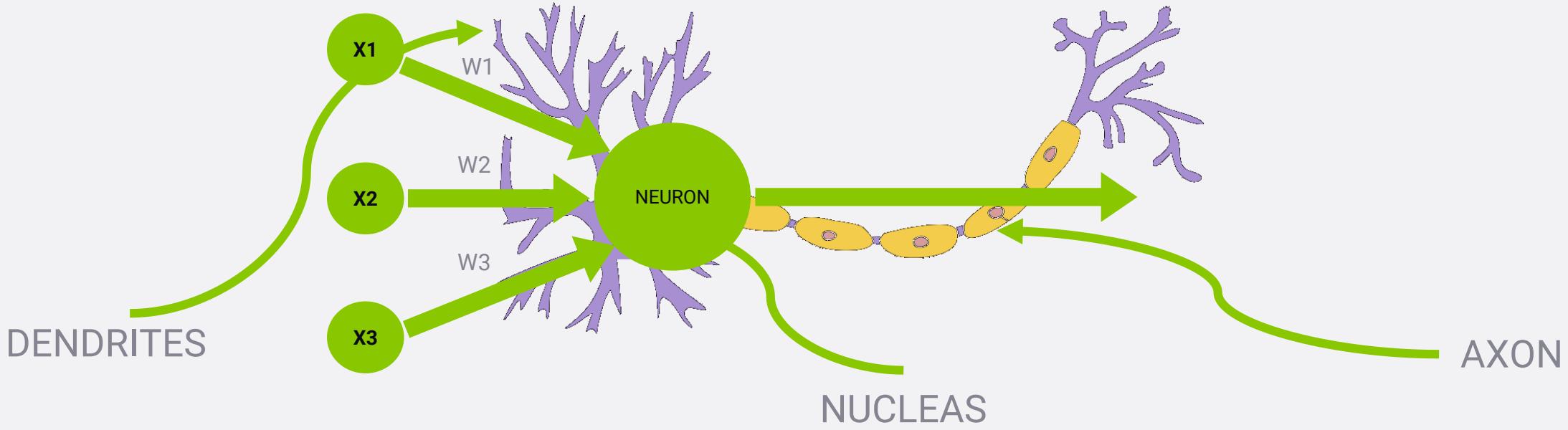
- Self-driving cars
- Image classification
- Fault detection and isolation
- Face and speech recognition
- System identification and control systems applications
- Finger prints recognition
- Handwritten character recognition
- Business applications such as bank failure prognosis and stock market predictions

INTRODUCTION
PLAN OF ATTACK!



SINGLE NEURON MODEL
NEURON MATHEMATICAL MODEL

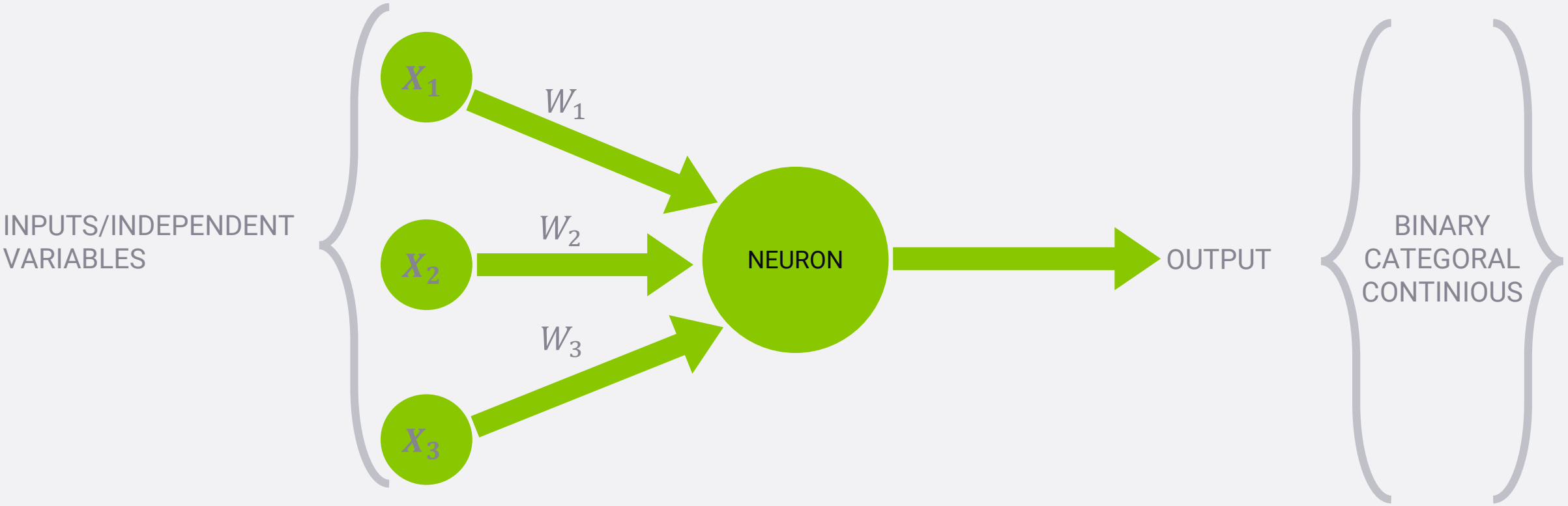
- The neuron collects signals from input channels named dendrites, processes information in its nucleus, and then generates an output in a long thin branch called the axon.



- **Photo Credit:** https://commons.wikimedia.org/wiki/File:Neuron_Hand-tuned.svg

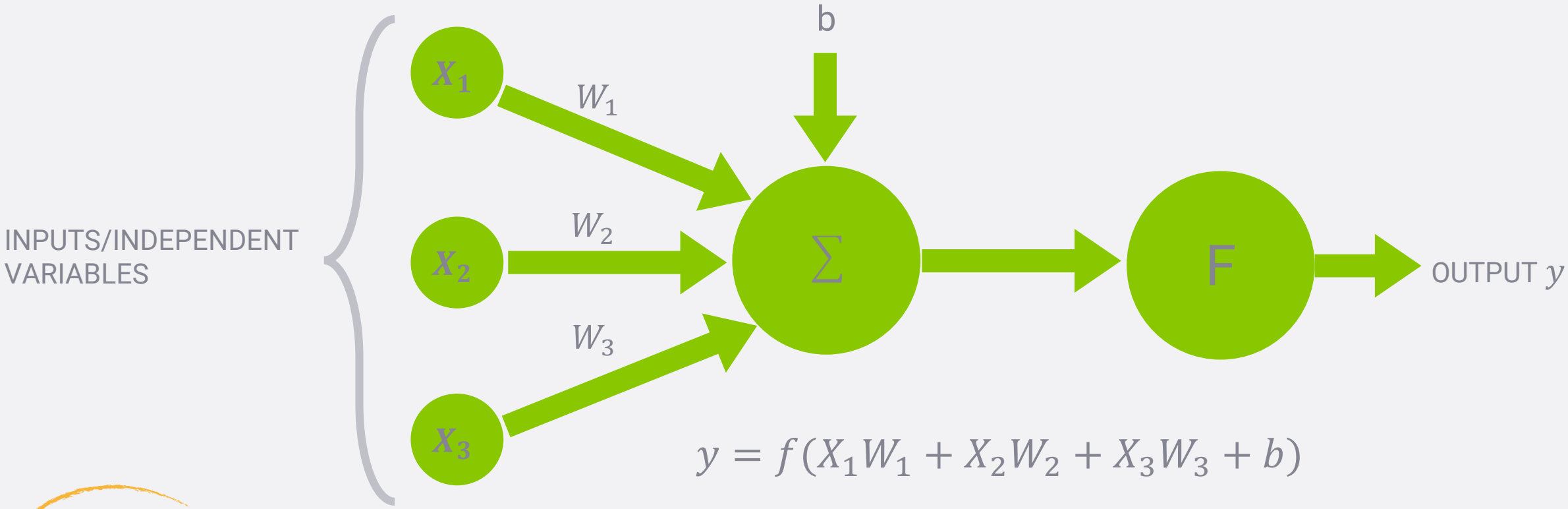
SINGLE NEURON MODEL
NEURON MATHEMATICAL MODEL

- The neuron collects signals from input channels named dendrites, processes information in its nucleus, and then generates an output in a long thin branch called the axon.
- Weights shows the strength of the particular node.



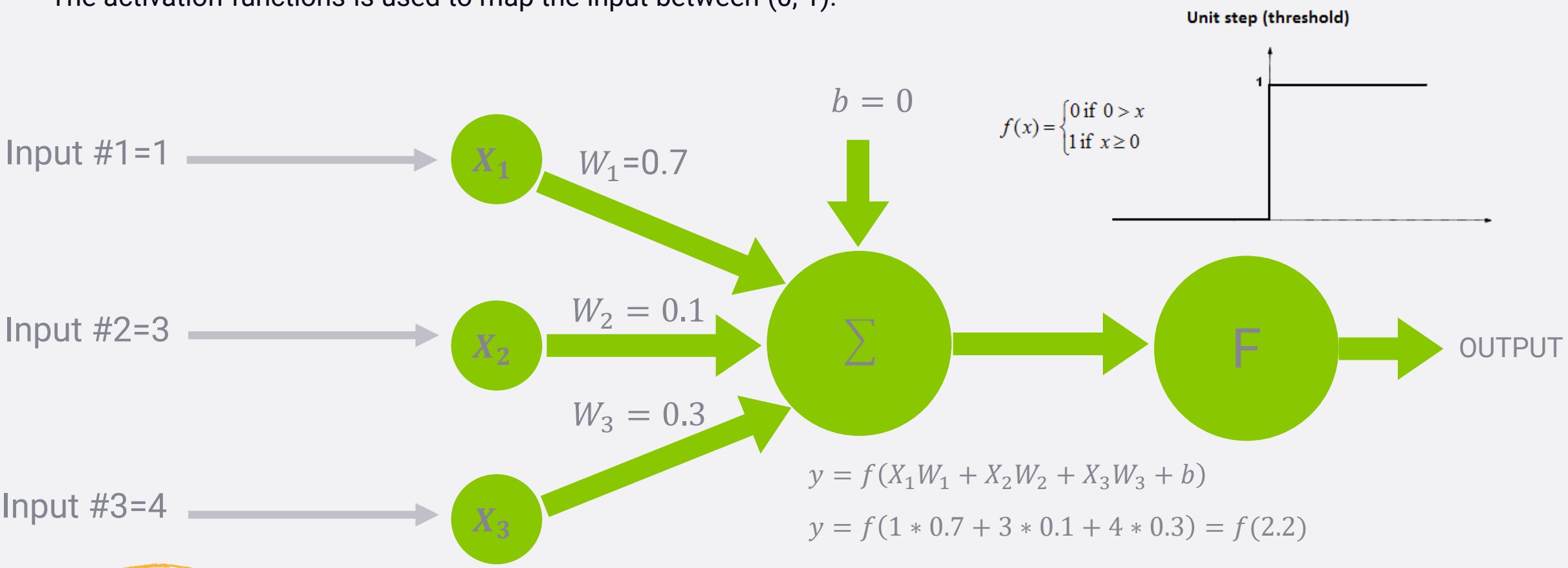
SINGLE NEURON MODEL
NEURON MATHEMATICAL MODEL

- bias allows to shift the activation function curve up or down.
- Number of adjustable parameters = 4 (3 weights and 1 bias).
- Activation function “F”.



SINGLE NEURON MODEL
NEURON MATHEMATICAL MODEL

- Let's assume that the activation function is a Unit Step Activation Function.
- The activation functions is used to map the input between (0, 1).



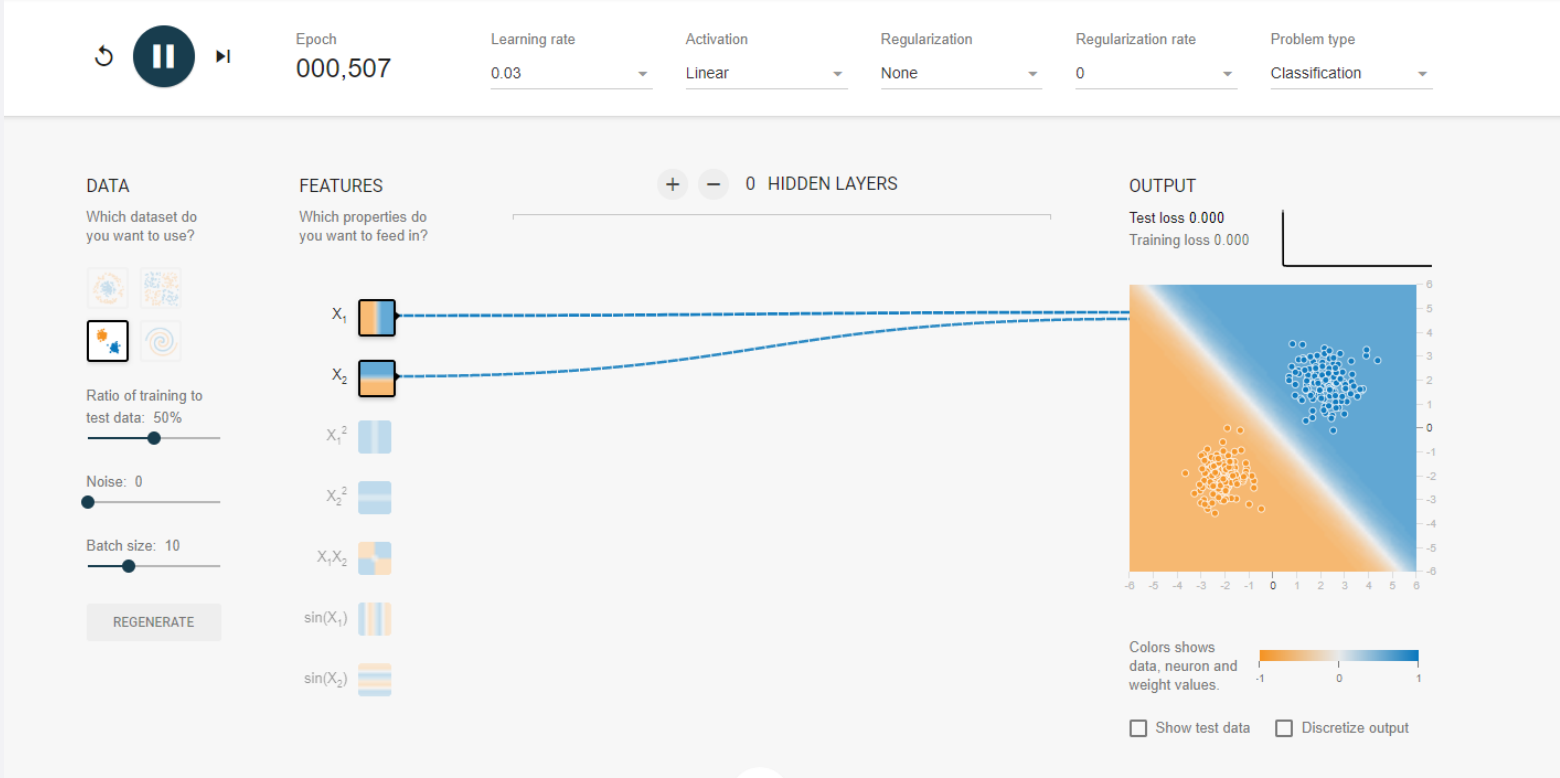
$$y = f(X_1W_1 + X_2W_2 + X_3W_3 + b)$$

$$y = f(1 * 0.7 + 3 * 0.1 + 4 * 0.3) = f(2.2)$$

$$y = 1 \text{ (because } 2.2 > 0)$$

SINGLE NEURON MODEL
FUN SIMULATION!

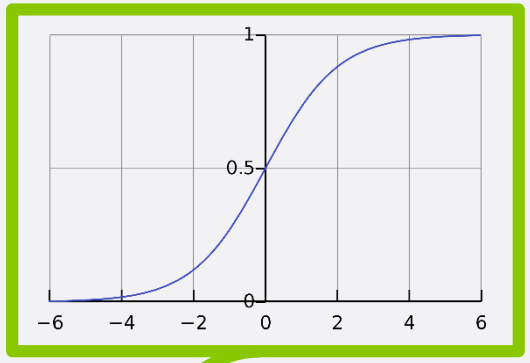
- Check this out: https://playground.tensorflow.org

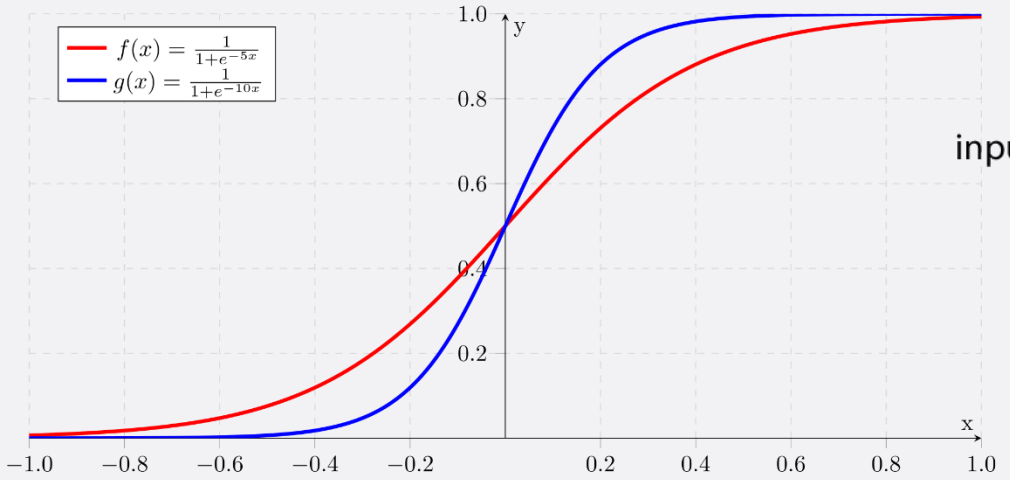


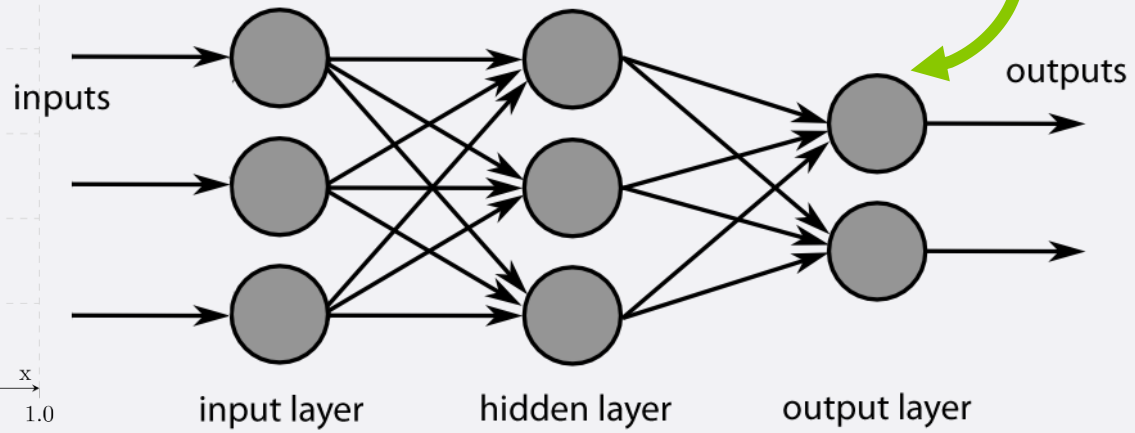
SINGLE NEURON MODEL
ACTIVATION FUNCTIONS

SIGMOID:

- Takes a number and sets it between 0 and 1
- Converts large negative numbers to 0 and large positive numbers to 1.
- Generally used in output layer.







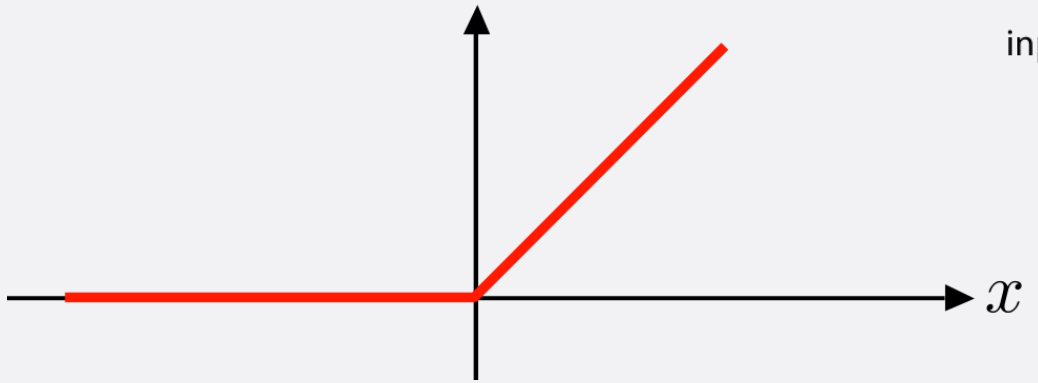
- **Photo credit:** https://commons.wikimedia.org/wiki/File:Sigmoid-function.svg
- **Photo Credit:** https://fr.m.wikipedia.org/wiki/Fichier:MultiLayerNeuralNetworkBigger_english.png
- **Photo Credit:** **https://commons.wikimedia.org/wiki/File:Logistic-curve.svg**

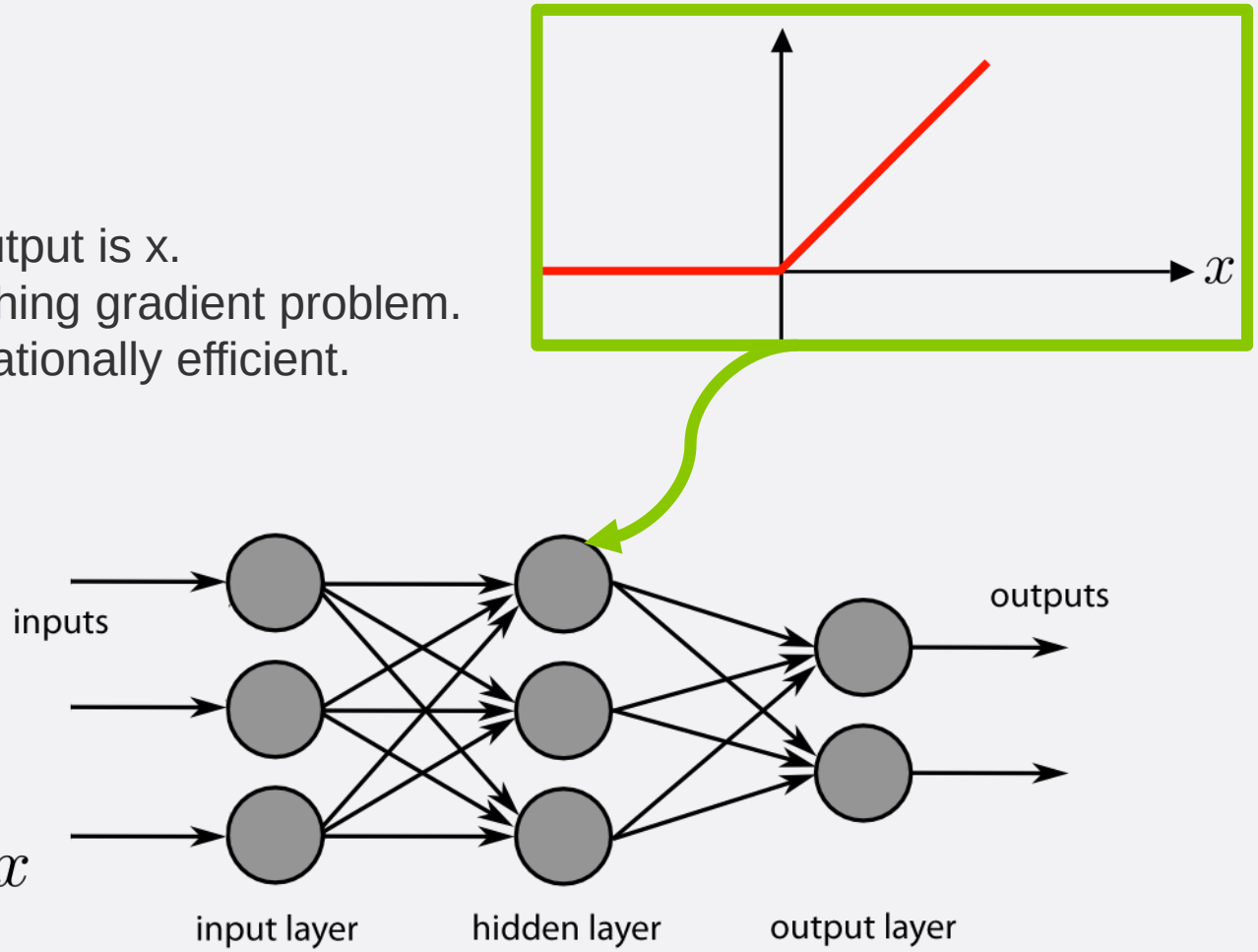
SINGLE NEURON MODEL
ACTIVATION FUNCTIONS

RELU (RECTIFIED LINEAR UNITS):

- if input x < 0, output is 0 and if x > 0 the output is x.
- RELU does not saturate so it avoids vanishing gradient problem.
- It uses simple thresholding so it is computationally efficient.
- Generally used in hidden layers

$$\mathrm{ReLU}(x) \triangleq \max(0, x)$$



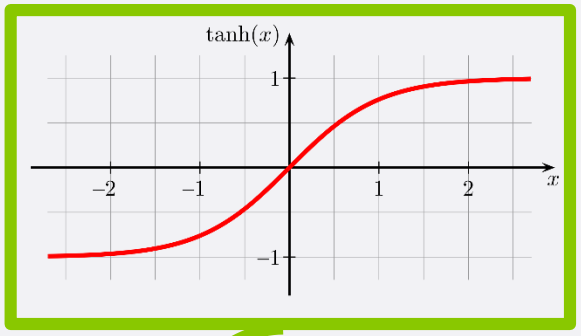


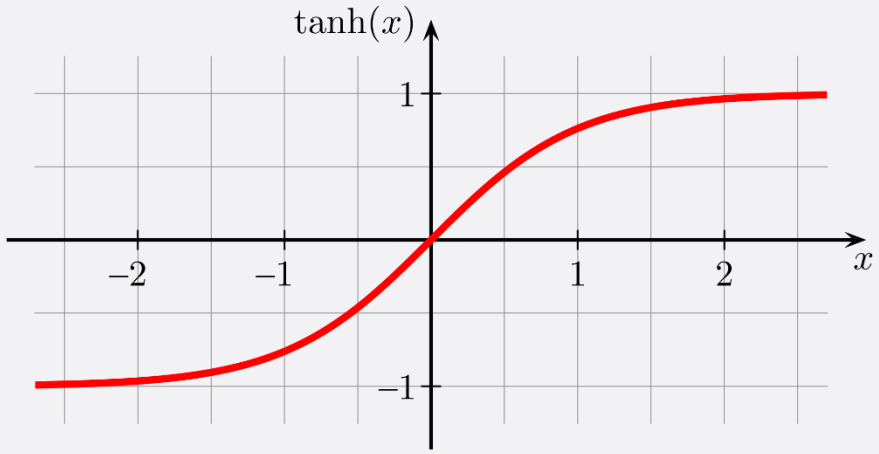
- **Photo credit:** https://commons.wikimedia.org/wiki/File:ReLU_and_Nonnegative_Soft_Thresholding_Functions.svg
- **Photo Credit:** https://fr.m.wikipedia.org/wiki/Fichier:MultiLayerNeuralNetworkBigger_english.png

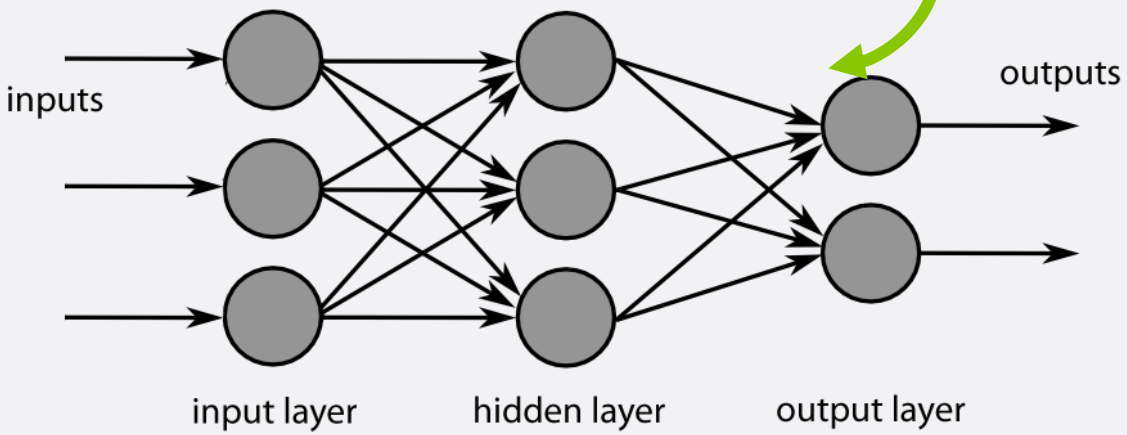
SINGLE NEURON MODEL
ACTIVATION FUNCTIONS

HYPERBOLIC TANGENT ACTIVATION FUNCTION

- "Tanh" is similar to sigmoid, converts number between -1 and 1.
- Unlike sigmoid, tanh outputs are zero-centered (range: -1 and 1).
- Tanh suffers from vanishing gradient problem so it kills gradients when saturated.
- In practice, tanh is preferable over sigmoid.







- **Photo credit:** https://commons.wikimedia.org/wiki/File:Hyperbolic_Tangent.svg
- **Photo Credit:** https://fr.m.wikipedia.org/wiki/Fichier:MultiLayerNeuralNetworkBigger_english.png

ANN TRAINING
HOW DO ANNs TRAIN?

- Like humans, ANNs can learn (or be trained) from experience (or by example) using training algorithms that can establish relationships between input-output datasets.
- During the learning (training) phase, ANNs can adaptively change their internal structure.
- Several ANNs training algorithms with various accuracy, speed, computational complexity, and memory requirements.

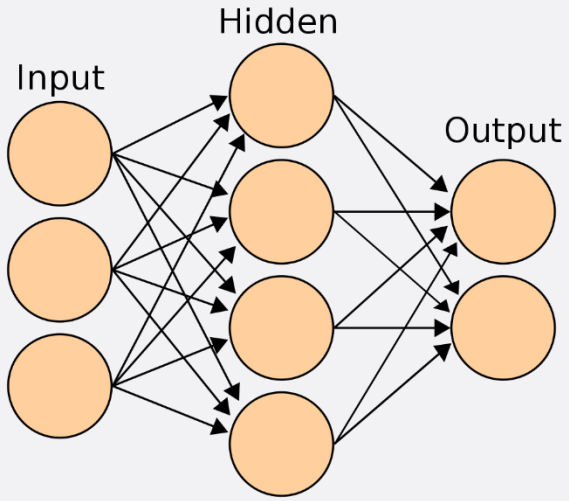


Photo Credit: https://commons.wikimedia.org/wiki/File:Artificial_neural_network.svg
Photo Credit: https://www.maxpixel.net/Father-Baby-Kid-Man-Girl-Child-People-Walking-2603224

ANN TRAINING
LEARNING STRATEGIES

Supervised learning

- Used if these is large set of test data with known labels (outputs).
- The learning algorithm evaluates output (i.e.: makes predictions), compares output against the label, and adjust network and repeat.

Unsupervised learning

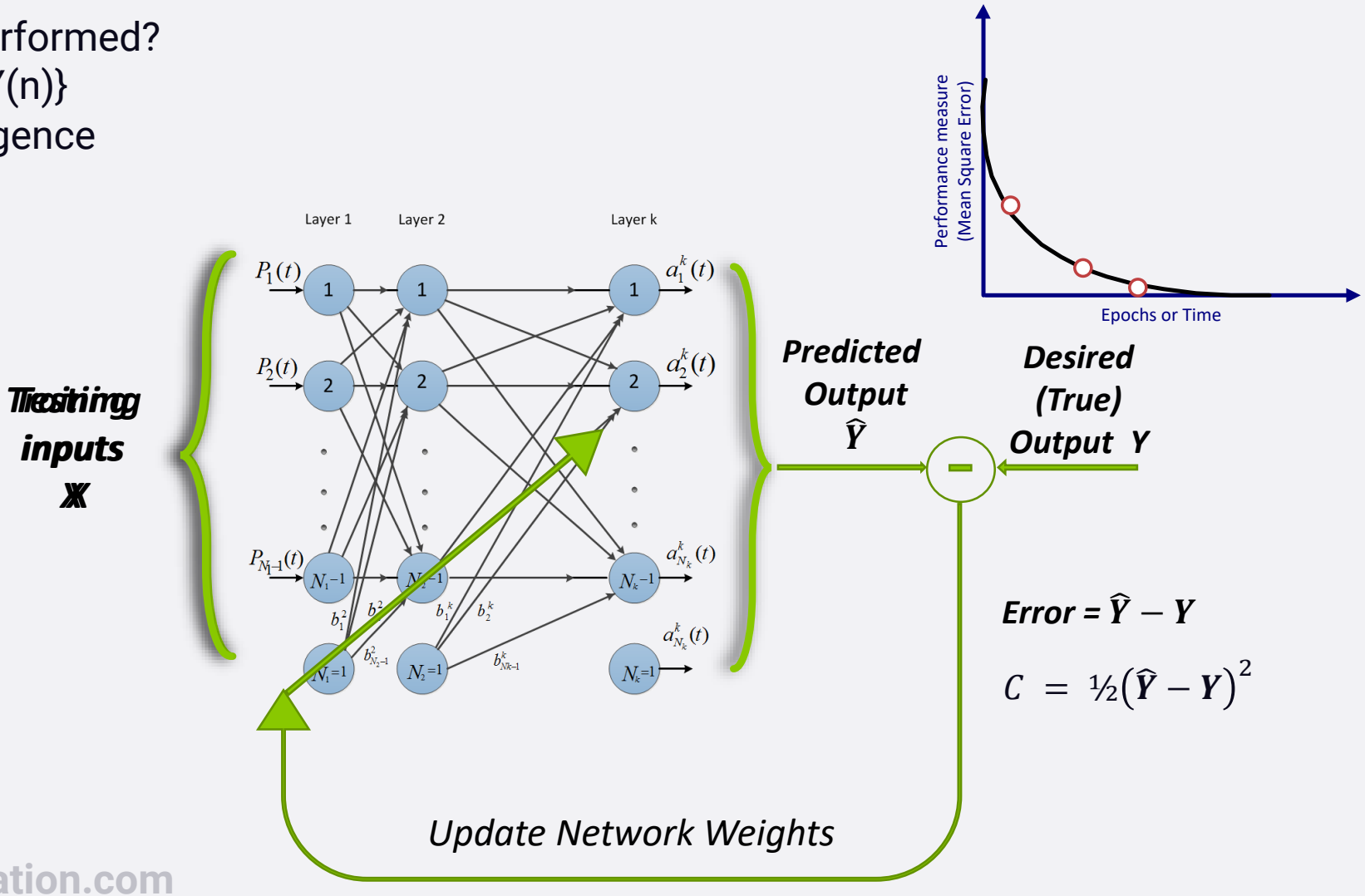
- Used with "unlabeled" data (not categorized) (Ex: k-means clustering).
- Since learning algorithm works with unlabeled data, there is no way to assess the accuracy of the structure suggested by the algorithm

Reinforced learning

- Learning algorithm takes actions that maximizes some notion of cumulative reward.
- Over time, the network learns to prefer the right kind of action and to avoid the wrong one.

ANN TRAINING
HOW DO ANNs TRAIN IN A SUPERVISED FASHION?

- How supervised training is performed?
- Assume training data: {X(n), Y(n)}
- Number of epochs till convergence
- Generalization capability



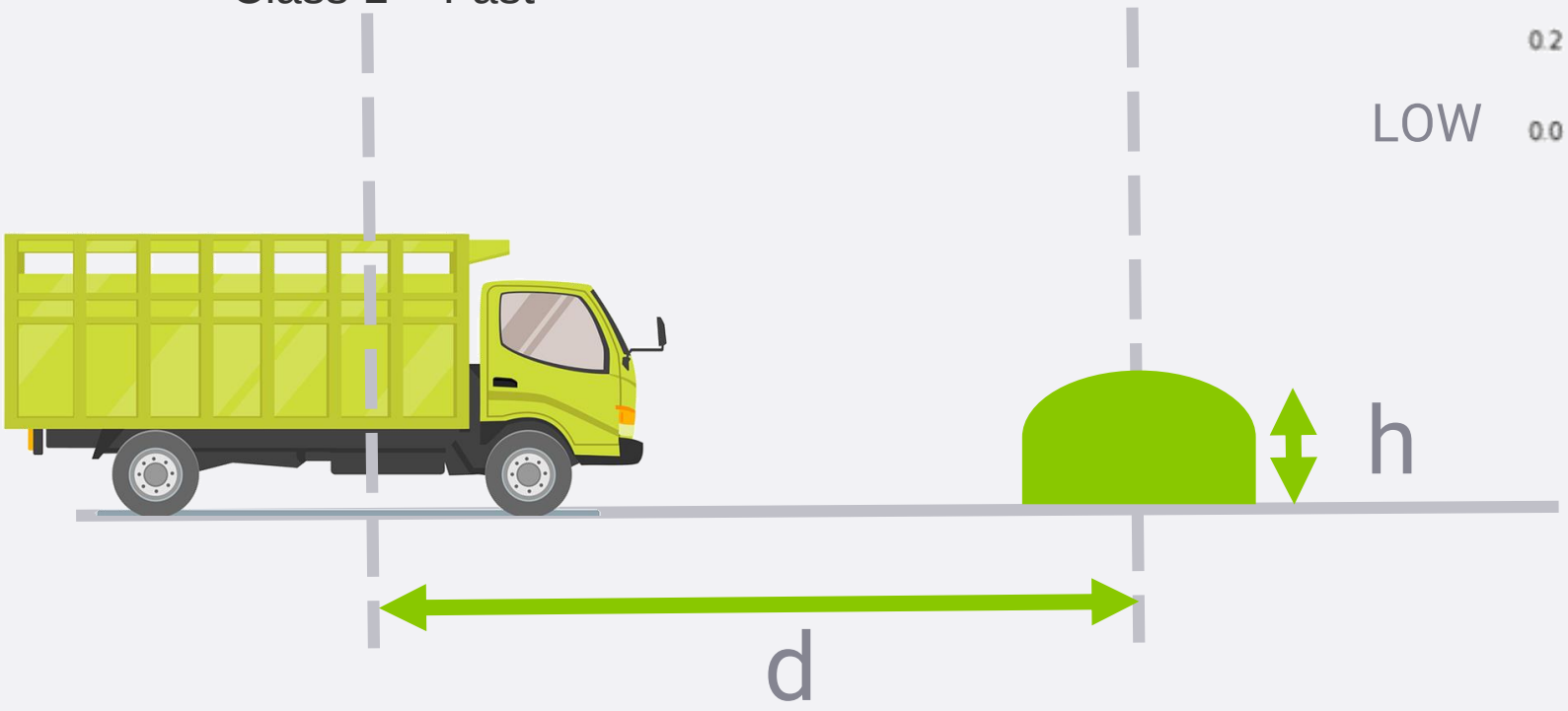
ANN TRAINING
HOW TO DIVIDE DATASETS?

- Data set is divided into 50%, 25%, 25% segments for training, validation, and testing, respectively.
- **Training set:** used for gradient calculation and weight update.
- **Validation set:** used for cross-validation which is performed to assess training quality as training proceeds. Cross-validation is implemented to overcome over-fitting (over-training).
  - Over-fitting occurs when algorithm focuses on training set details at cost of losing generalization ability.
  - Trained network MSE might be small during training but during testing, the network may exhibit poor generalization performance.
- **Testing set:** used for testing trained network.

PRACTICAL EXAMPLE
VEHICLE SPEED DECISION

Vehicle Speed decision:

- Class 0 = Slow
- Class 1 = Fast



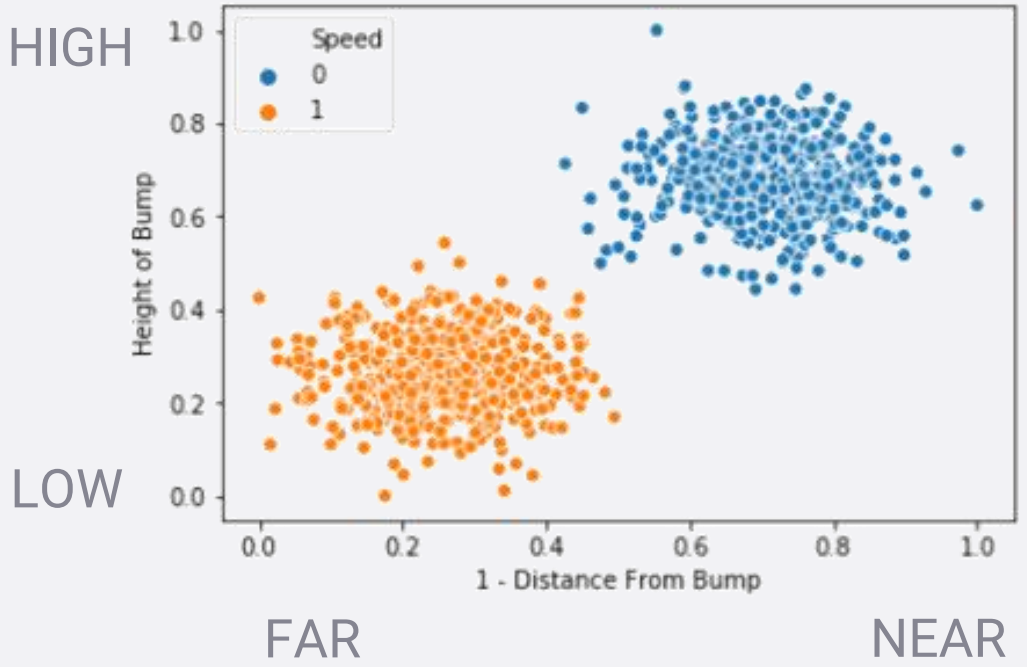
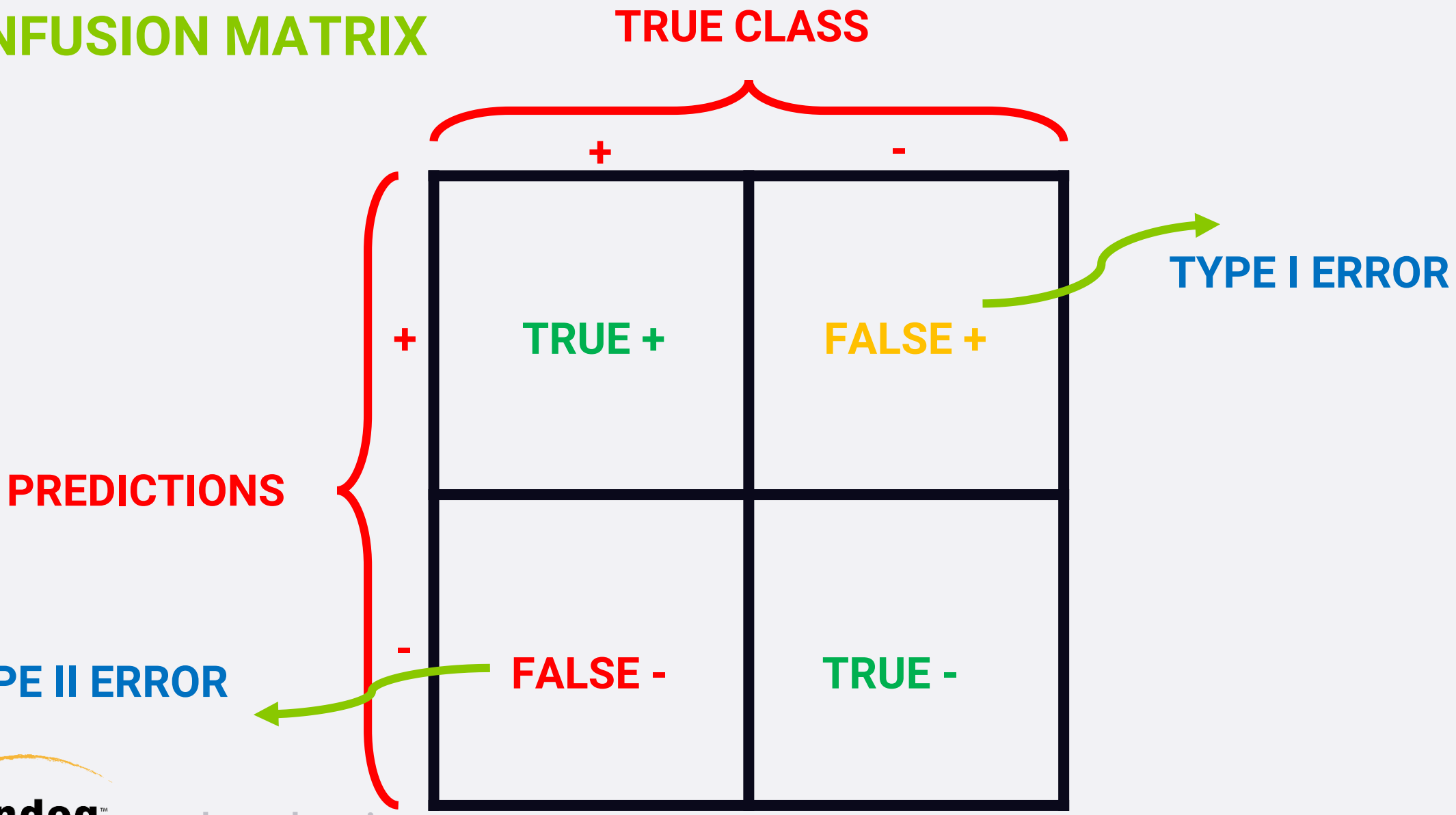


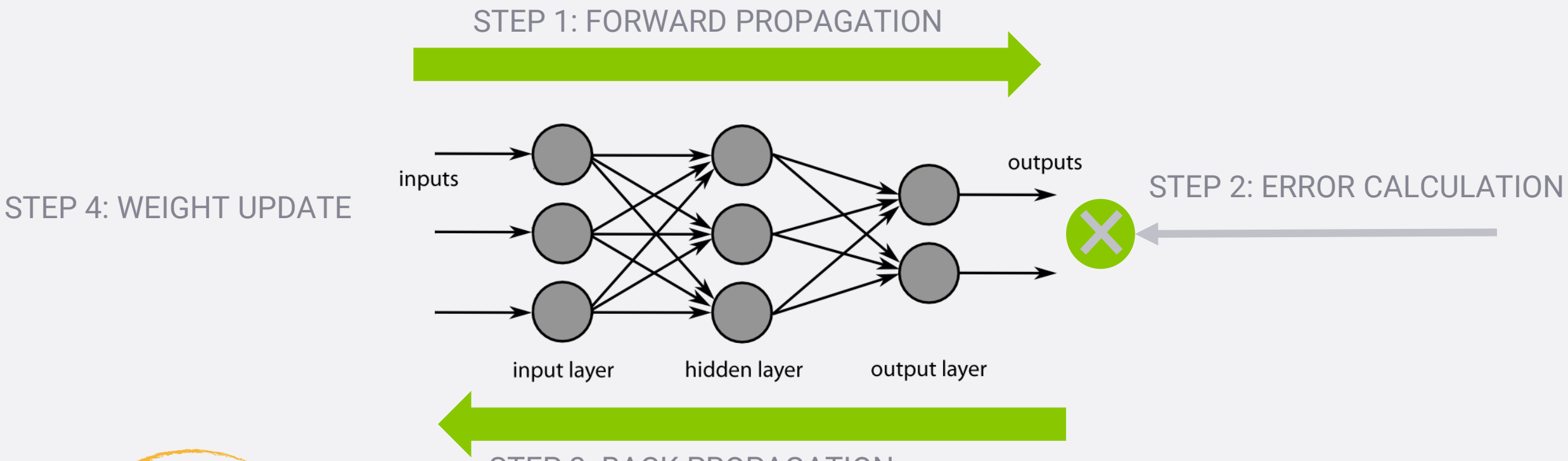
Photo Credit: https://pixabay.com/en/truck-car-vehicle-transportation-2178252/

PRACTICAL EXAMPLE
CONFUSION MATRIX



BACK PROPAGATION
INTUITION

- Backpropagation is a method used to train ANNs by calculating gradient needed to update network weights.
- It is commonly used by the gradient descent optimization algorithm to adjust the weight of neurons by calculating the gradient of the loss function.

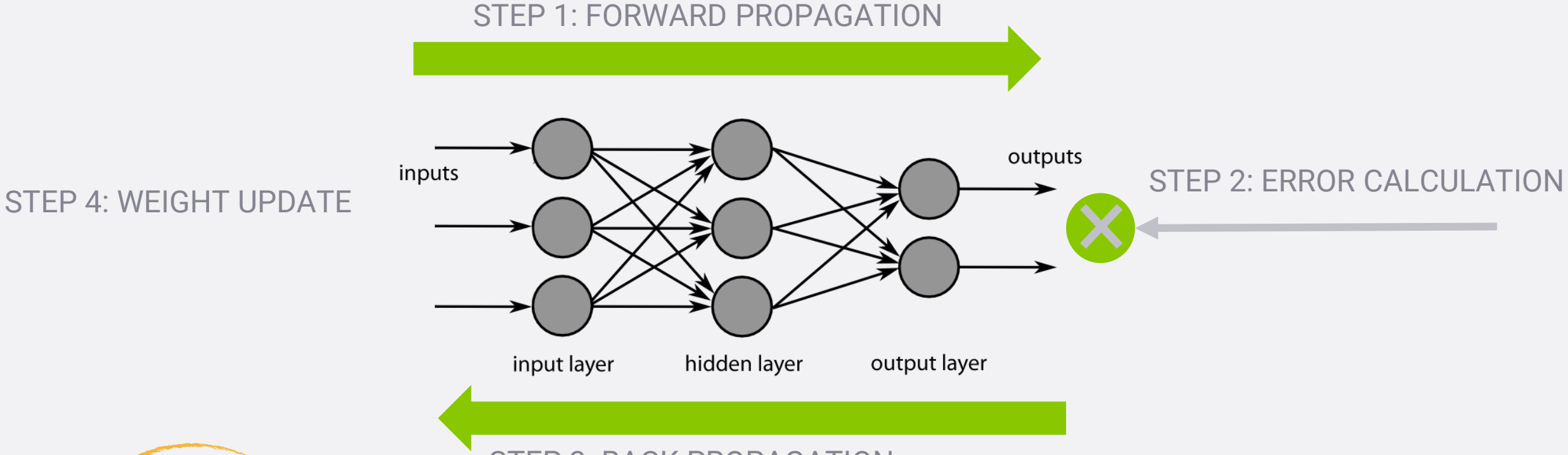


- **Photo Credit:** https://fr.m.wikipedia.org/wiki/Fichier:MultiLayerNeuralNetworkBigger_english.png

BACK PROPAGATION
INTUITION

Phase 1: propagation

1. Propagation forward through the network to generate the output value(s)
2. Calculation of the cost (error term)
3. Propagation of output activations back through the network using the training pattern target in order to generate the deltas (difference between targeted and actual output values)

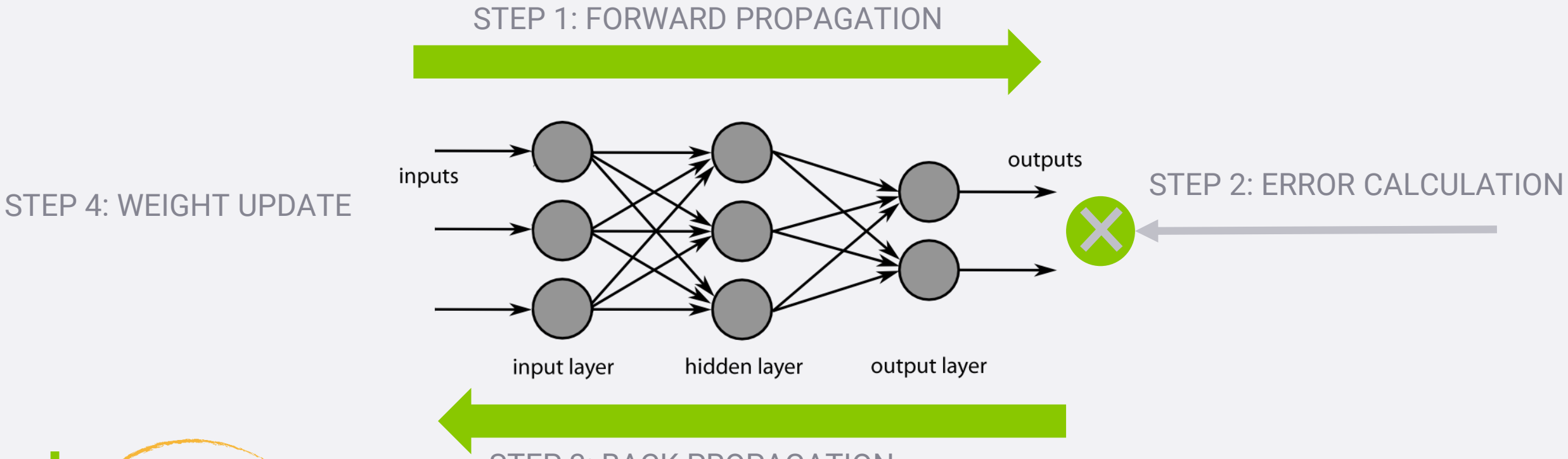


- **Photo Credit:** https://fr.m.wikipedia.org/wiki/Fichier:MultiLayerNeuralNetworkBigger_english.png

BACK PROPAGATION
INTUITION

Phase 2: weight update

- Calculate weight gradient.
- A ratio (percentage) of the weight's gradient is subtracted from the weight.
- This ratio (percentage) influences the speed and quality of learning; it is called the *learning rate*. The greater the ratio, the faster the neuron trains, but the lower the ratio, the more accurate the training is.



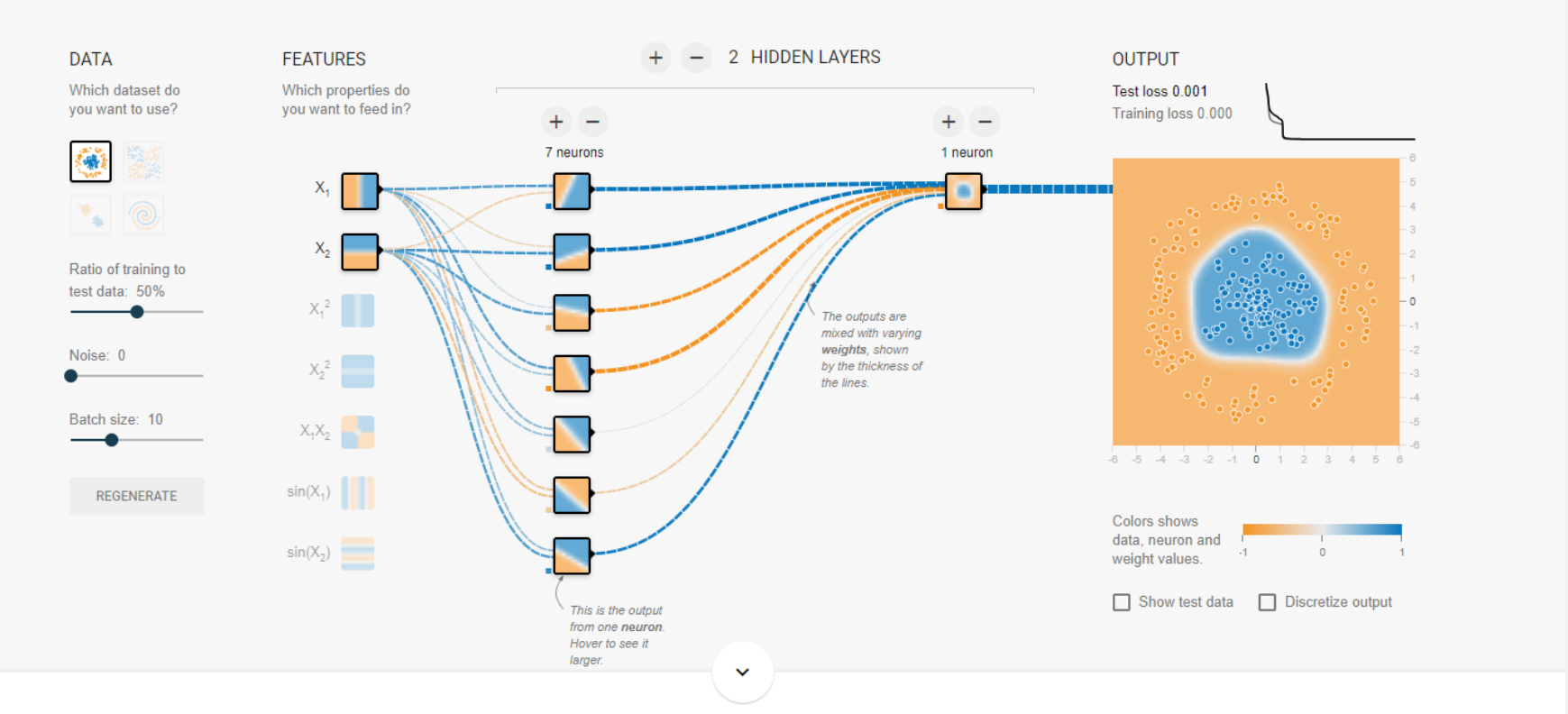
- **Photo Credit:** https://fr.m.wikipedia.org/wiki/Fichier:MultiLayerNeuralNetworkBigger_english.png

BACK PROPAGATION
READING MATERIAL

- “Backpropagation neural networks: A tutorial” by Barry J.Wythoff
- “Improved backpropagation learning in neural networks with windowed momentum”, International Journal of Neural Systems, vol. 12, no.3&4, pp. 303-318.

MULTILAYER PERCEPTRON ANN MODEL VISUALIZATION

- Check this out: https://playground.tensorflow.org



TWO-NEURON MODEL
MATRIX REPRESENTATION

- The network is represented by a matrix of weights, inputs and outputs.
- Total Number of adjustable parameters = 8:
  - Weights = 6
  - Biases = 2

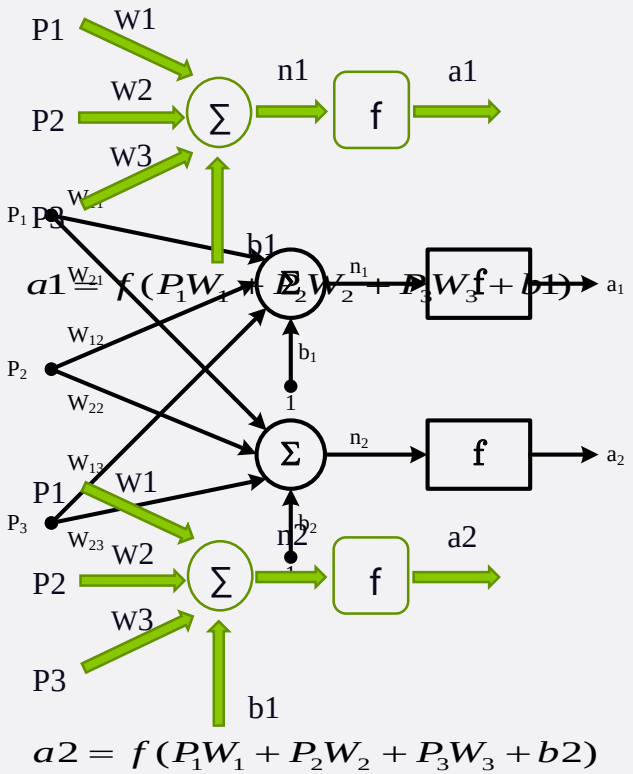
Matrix Representation

$$P = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

$$W = \begin{bmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \end{bmatrix}$$

$$b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

$$a = f(W \times P + b)$$



MULTILAYER PERCEPTRON ANN MODEL
NETWORK STRUCTURE

- A Multi-layer Perceptron (MLP) network consists of one input layer, hidden layers, and one output layer.
- Each layer consists of a number of neurons where each neuron is fully connected to all neurons in its preceding layer.
- In a feedforward MLP, signals propagate across layers in a forward fashion.

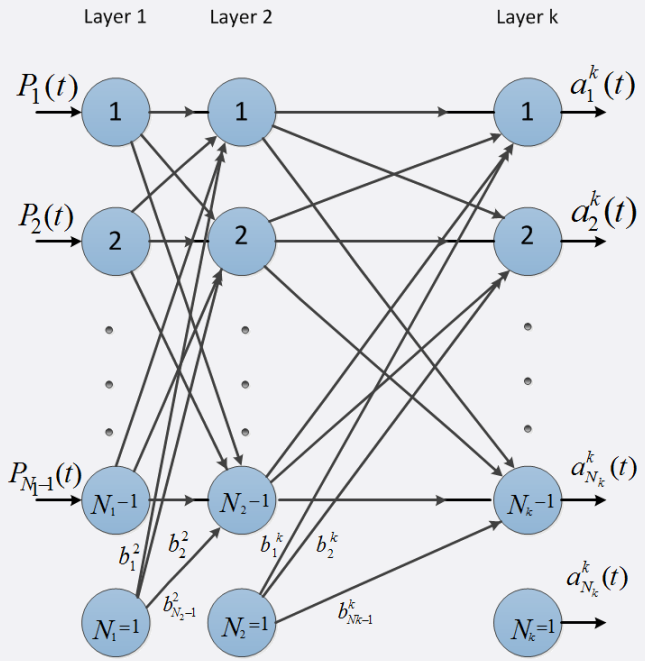
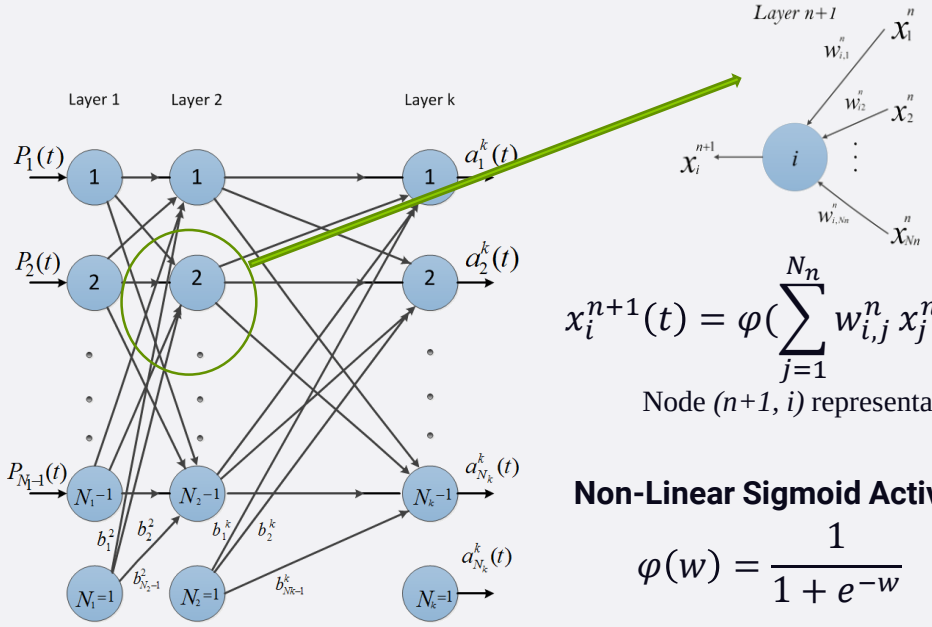


Photo Credit: Automotive Internal-Combustion-Engine Fault Detection and Classification Using Artificial Neural Network Techniques, IEEE Transactions on Vehicular Technology (2015)

MULTILAYER PERCEPTRON ANN MODEL
MATH (LIGHT!)

$$P = \begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_{N_1} \end{bmatrix}$$



$$x_i^{n+1}(t) = \varphi\left(\sum_{j=1}^{N_n} w_{i,j}^n \, x_j^n(t)\right)$$

Node *(n+1, i)* representation

$$\begin{bmatrix} W_{11} & W_{12} & \cdots & W_{1,N_1} \\ W_{21} & W_{22} & & W_{2,N_1} \\ \vdots & & \ddots & \vdots \\ W_{m-1,1} & W_{m-1,2} & \cdots & W_{m-1,N_1} \\ W_{m,1} & W_{m,2} & & W_{m,N_1} \end{bmatrix}$$

Non-Linear Sigmoid Activation function

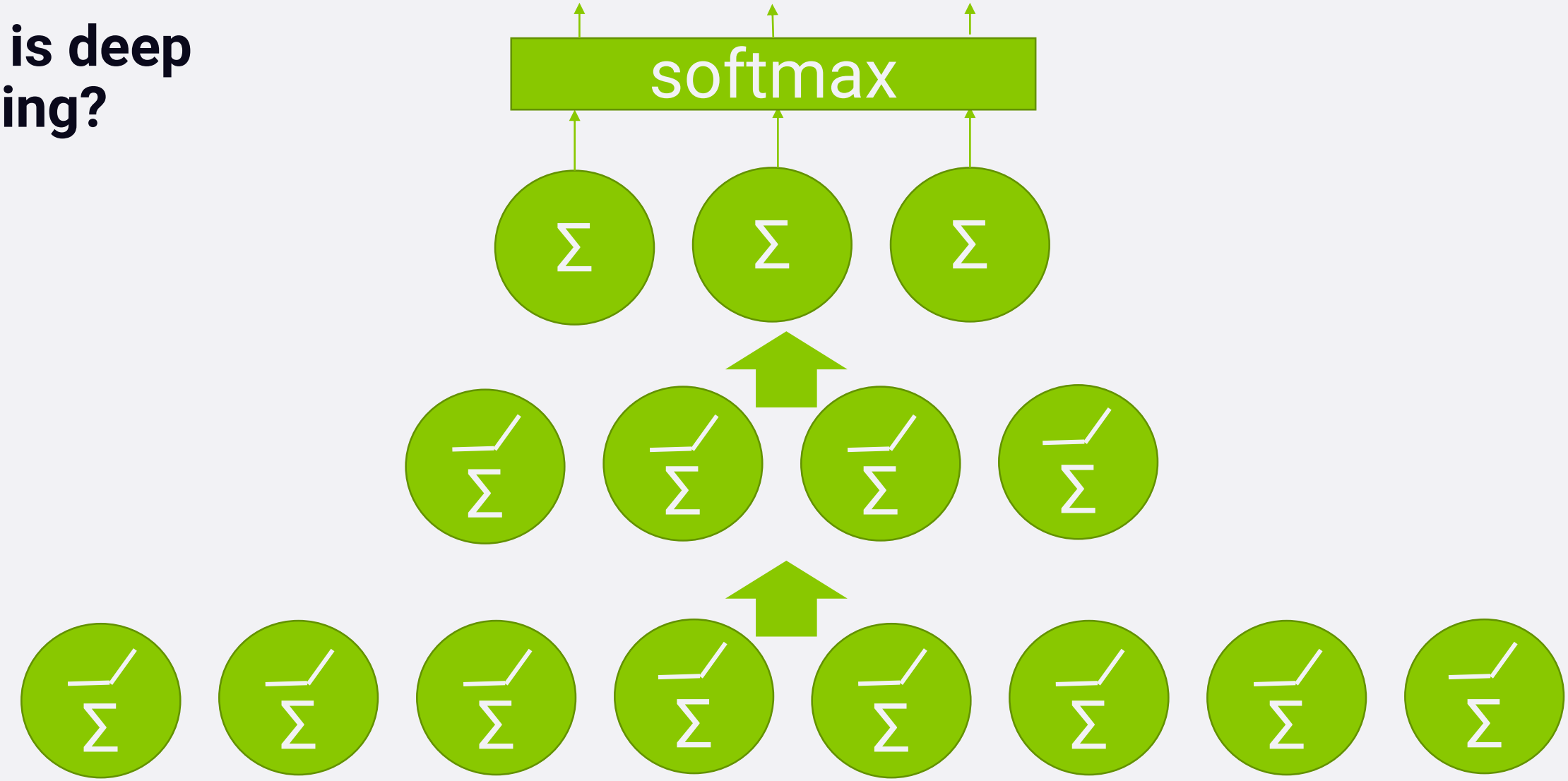
$$\varphi(w) = \frac{1}{1 + e^{-w}}$$

m: number of neurons in the hidden layer

N_1: number of inputs

Photo Credit: Automotive Internal-Combustion-Engine Fault Detection and Classification Using Artificial Neural Network Techniques, IEEE Transactions on Vehicular Technology (2015)

what is deep learning?



tensorflow

why tensorflow?

- it's not specifically for neural networks– it's more generally an architecture for executing a graph of numerical operations
- tensorflow can optimize the processing of that graph, and distribute its processing across a network
- it can also distribute work across GPU's!
  - can handle massive scale – it was made by Google
- runs on about anything
- highly efficient C++ code with easy to use Python API's

tensorflow basics

- install to Anaconda with pip install --ignore-installed --upgrade tensorflow
- or tensorflow-gpu (requires CUDA, cuDNN)
- a tensor is just a fancy name for an array or matrix of values
- to use Tensorflow, you:
  - construct a graph to compute your tensors
  - initialize your variables
  - execute that graph – nothing actually happens until then

World's simplest Tensorflow app:

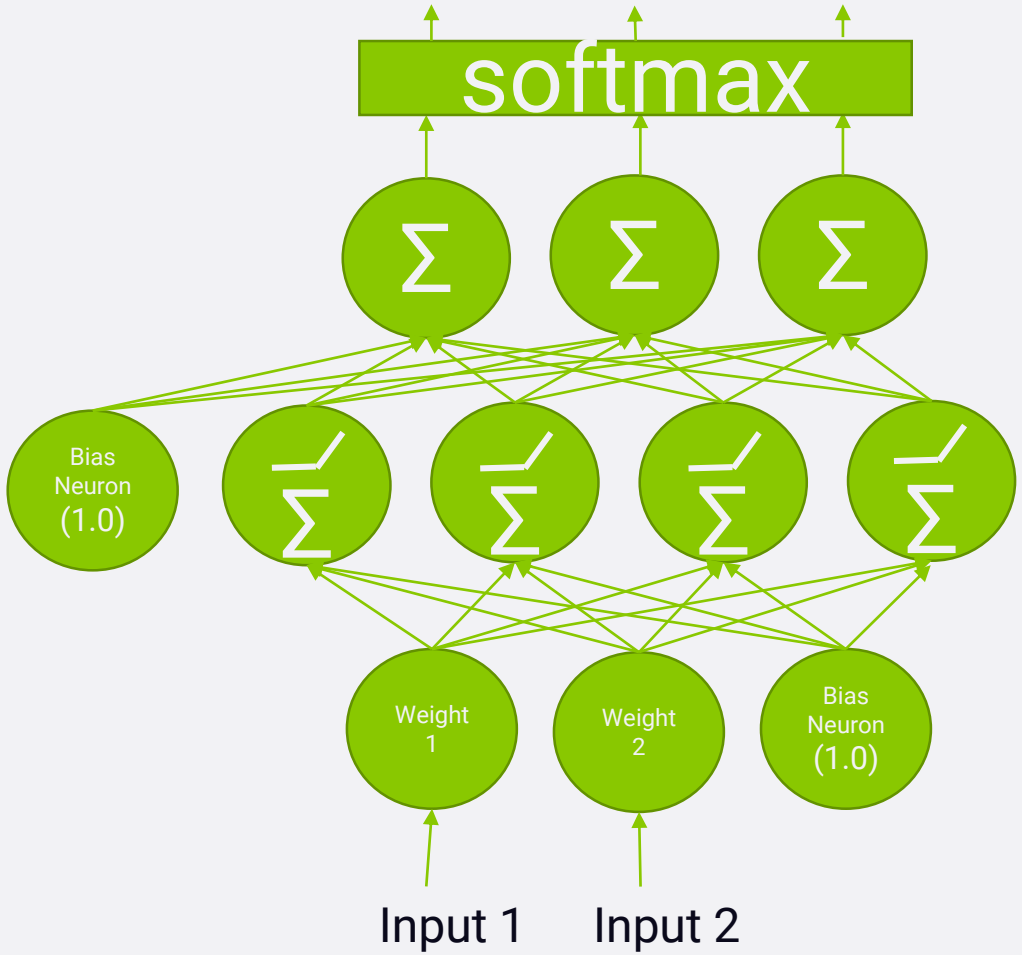
```
import tensorflow as tf

a = tf.Variable(1, name="a")
b = tf.Variable(2, name="b")
f = a + b

init = tf.global_variables_initializer()
with tf.Session() as s:
    init.run()
    print( f.eval() )
```

creating a neural network with tensorflow

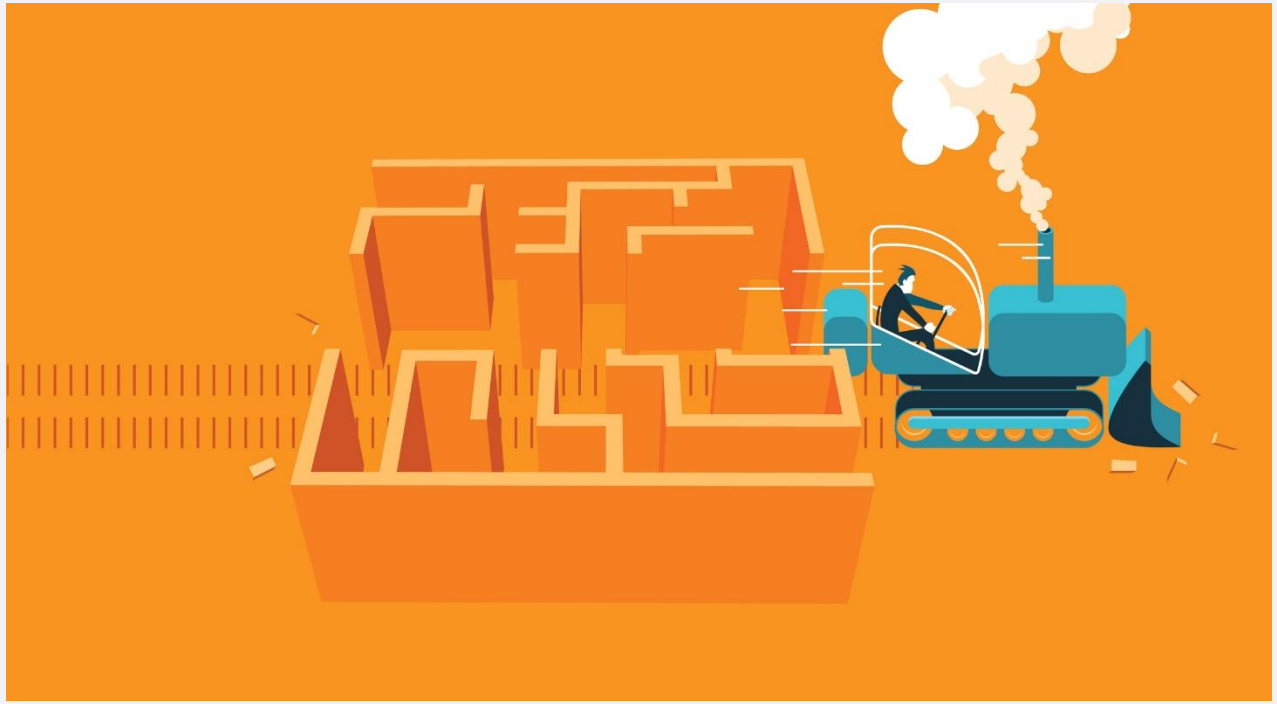
- mathematical insights:
  - all those interconnected arrows multiplying weights can be thought of as a big matrix multiplication
  - the bias term can just be added onto the result of that matrix multiplication
- so in Tensorflow, we can define a layer of a neural network as:

```
output = tf.matmul(previous_layer, layer_weights) + layer_biases
```

- by using Tensorflow directly we're kinda doing this the "hard way."



keras to the rescue



- easy and fast prototyping
  - higher-level API for Tensorflow
  - made for building deep neural nets
  - scikit_learn integration
  - less to think about – which often yields better results without even trying
  - this is really important! The faster you can experiment, the better your results.

example: multi-class classification

- image classification is an example of multi-class classification.

```
model = Sequential()

model.add(Dense(64, activation='relu', input_dim=20))
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9,
        nesterov=True)
model.compile(loss='categorical_crossentropy',
        optimizer=sgd, metrics=['accuracy'])
```

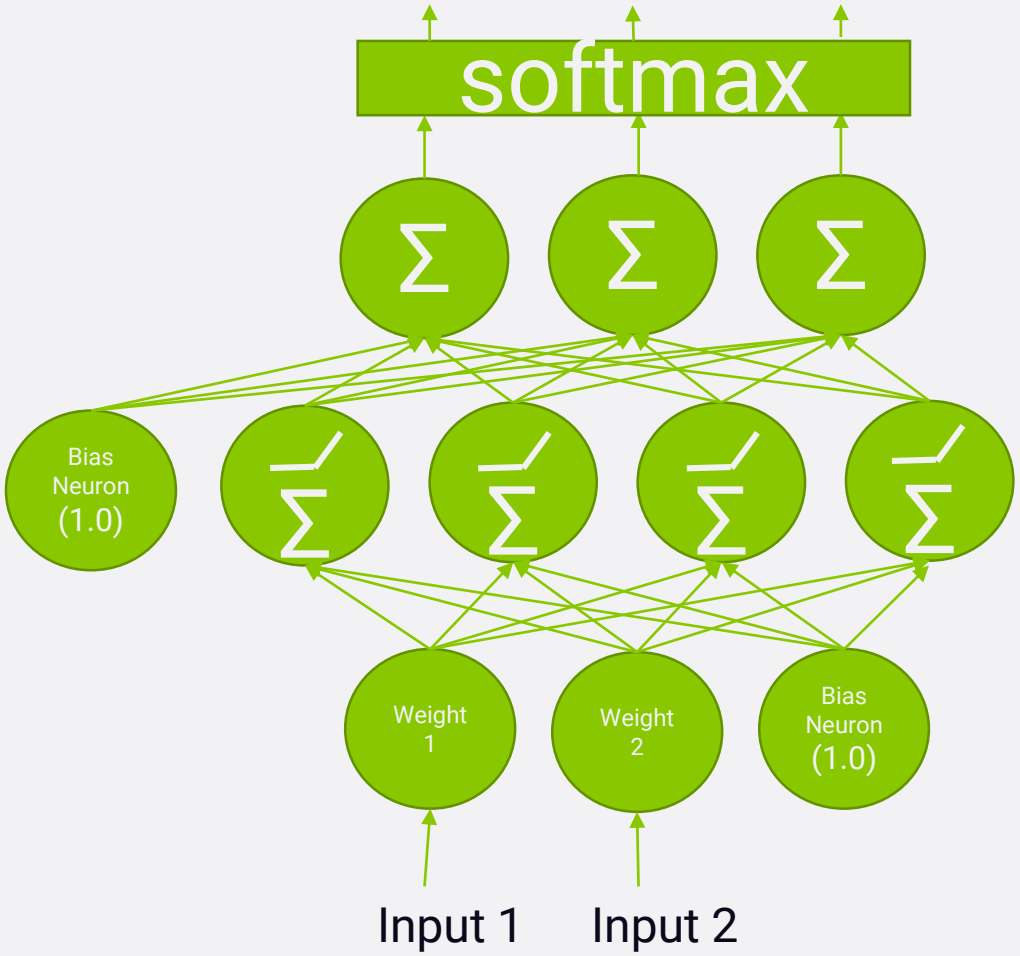
example: binary classification

```
model = Sequential()
model.add(Dense(64, input_dim=20, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy',
optimizer='rmsprop',  metrics=['accuracy'])
```

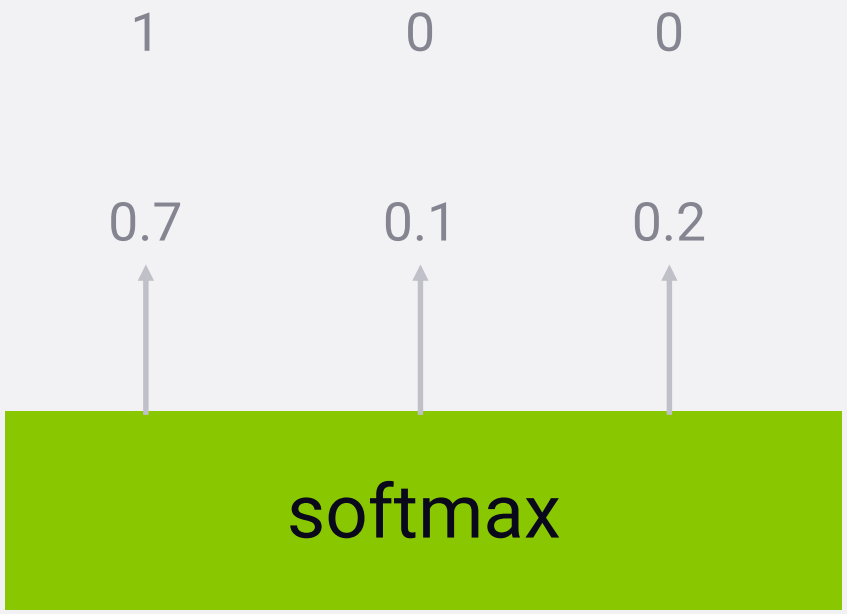
make sure your features are normalized

- neural networks usually work best if your input data is normalized.
  - That is, 0 mean and unit variance
  - The real goal is that every input feature is comparable in terms of magnitude
- scikit_learn's StandardScaler can do this for you
- many data sets are normalized to begin with

one-hot encoding



one-hot encoding



correct label: category 0

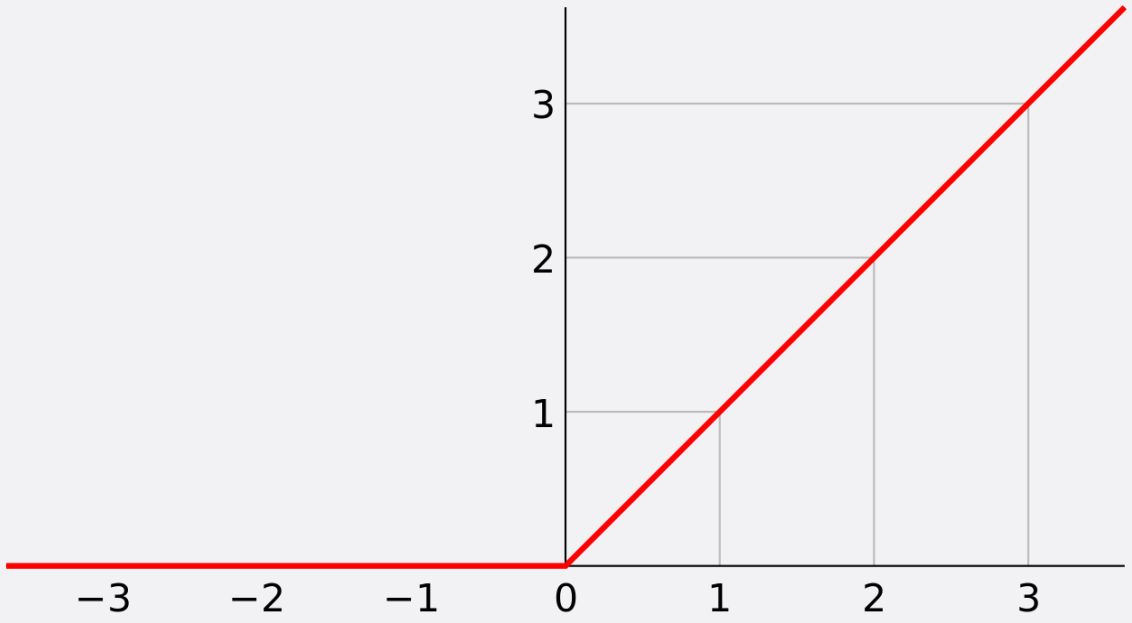
one-hot encoding

category	one-hot format
0	1 0 0
1	0 1 0
2	0 0 1

let’s try it out

activation functions (aka rectifier)

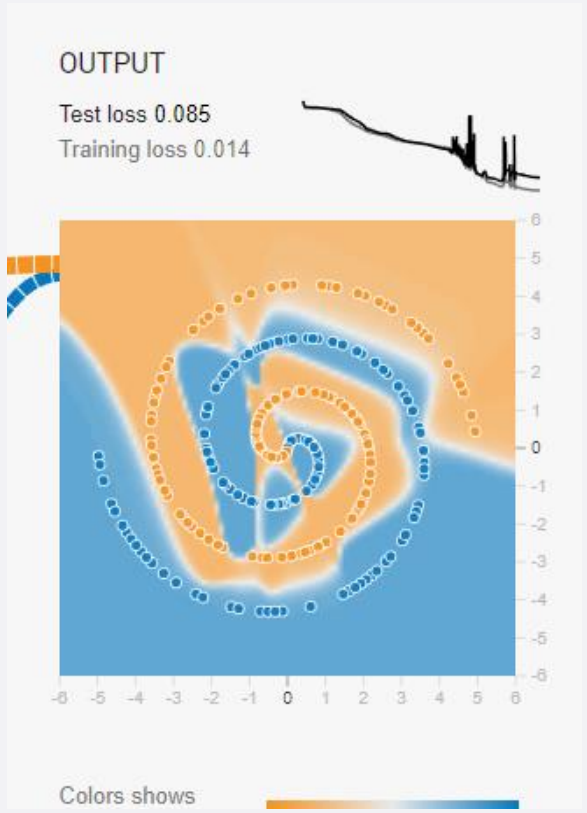
- Step functions don't work with gradient descent – there is no gradient!
  - Mathematically, they have no useful derivative.
- Alternatives:
  - Logistic (sigmoid) function
  - Hyperbolic tangent function
  - Exponential linear unit (ELU)
  - ReLU function (Rectified Linear Unit)
- ReLU is common. Fast to compute and works well.
  - Also: "Leaky ReLU", "Noisy ReLU"
  - ELU can sometimes lead to faster learning though.



ReLU function

avoiding overfitting with regularization

- With thousands of weights to tune, overfitting is a problem
- Early stopping (when performance starts dropping)
- Regularization terms added to cost function during training
- Dropout – ignore say 50% of all neurons randomly at each training step
  - Works surprisingly well!
  - Forces your model to spread out its learning

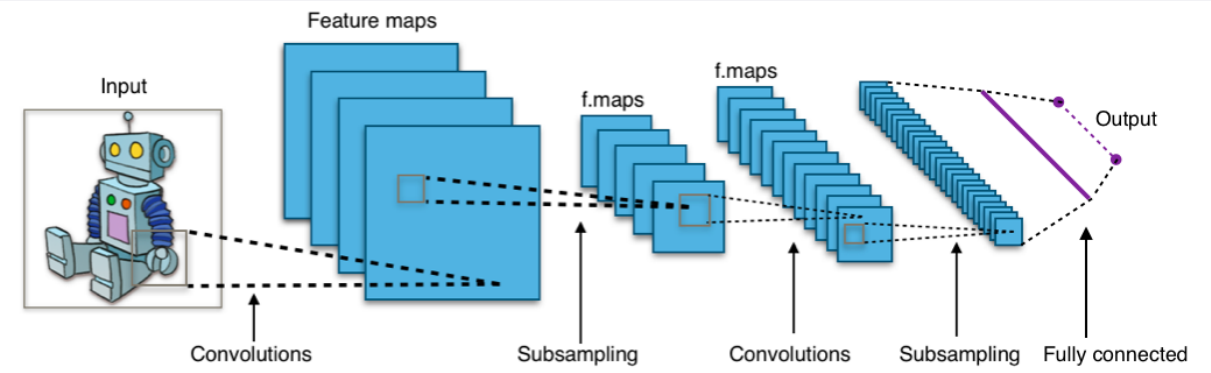


convolutional neural networks

cnn's: what are they for?

- when you have data that doesn't neatly align into columns
  - Images that you want to find features within
  - Machine translation
  - Sentence classification
  - Sentiment analysis
- they can find features that aren't in a specific spot
  - Like a stop sign in a picture
  - Or words within a sentence
- they are "feature-location invariant"

cnn's: how do they work?



- inspired by the biology of the visual cortex
  - local receptive fields are groups of neurons that only respond to a part of what your eyes see (subsampling)
  - they overlap each other to cover the entire visual field (convolutions)
  - they feed into higher layers that identify increasingly complex images
    - some receptive fields identify horizontal lines, lines at different angles, etc. (filters)
    - these would feed into a layer that identifies shapes
    - which might feed into a layer that identifies objects
  - for color images, extra layers for red, green, and blue

how do we “know” that’s a stop sign?

- individual local receptive fields scan the image looking for edges, and pick up the edges of the stop sign in a layer
- those edges in turn get picked up by a higher level convolution that identifies the stop sign’s shape (and letters, too)
- this shape then gets matched against your pattern of what a stop sign looks like, also using the strong red signal coming from your red layers
- that information keeps getting processed upward until your foot hits the brake!
- a CNN works the same way

cnn’s with keras

- source data must be of appropriate dimensions
  - ie width x length x color channels
- conv2D layer type does the actual convolution on a 2D image
  - Conv1D and Conv3D also available – doesn’t have to be image data
- flatten layers will convert the 2D layer to a 1D layer for passing into a flat hidden layer of neurons
- typical usage:
  - Convolve -> Dropout -> Flatten -> Dense -> Dropout -> Softmax

cnn’s are hard

- very resource-intensive (CPU, GPU, and RAM)
- lots of hyperparameters
  - kernel sizes, many layers with different numbers of units, amount of pooling… in addition to the usual stuff like number of layers, choice of optimizer
- getting the training data is often the hardest part! (As well as storing and accessing it)

specialized cnn architectures

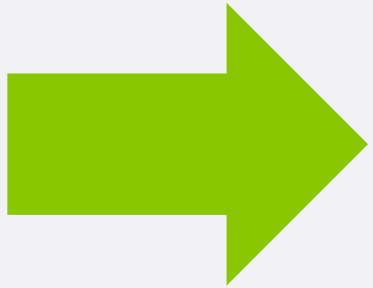
- defines specific arrangement of layers, padding, and hyperparameters
- LeNet-5
  - good for handwriting recognition
- AlexNet
  - image classification, deeper than LeNet
- GoogLeNet
  - even deeper, but with better performance
  - introduces *inception modules* (groups of convolution layers)
- ResNet (Residual Network)
  - even deeper – maintains performance via *skip connections*.

let’s try it out

max pooling with cnn's

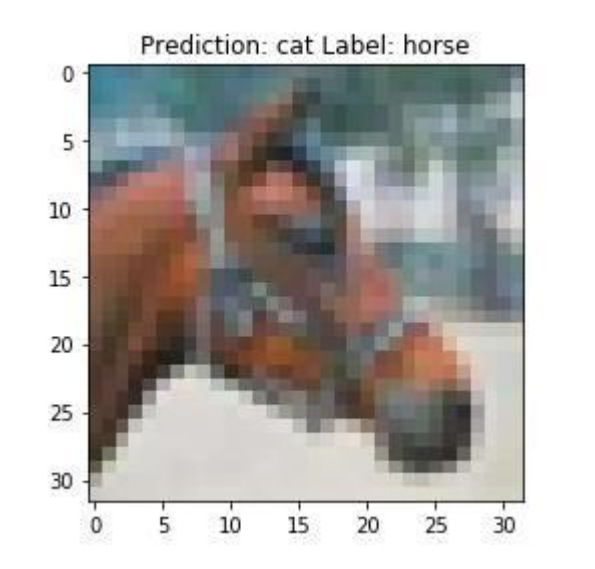
max pooling

1	0	2	3
4	6	6	8
3	1	1	0
1	2	2	4

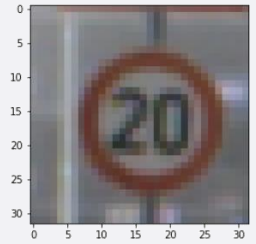


6	8
3	4

let’s improve on our image classifier.



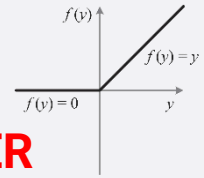
CONVOLUTIONAL NEURAL NETWORK
OVERVIEW



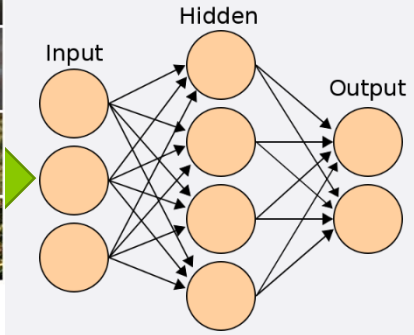
CONVOLUTION

KERNELS/
FEATURE
DETECTORS

FILTERS



CONVOLUTIONAL LAYER



POOLING LAYER
(DOWNSAMPLING)

YIELD

SPEED LIMIT
(20KM/H)

BUMPY ROAD

WILD ANIMALS

TURN LEFT AHEAD

Photo Credit: https://commons.wikimedia.org/wiki/File:Artificial_neural_network.svg
Data Source: J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In Proceedings of the IEEE International Joint Conference on Neural Networks, pages 1453–1460. 2011.
@inproceedings{Stallkamp-IJCNN-2011, author = {Johannes Stallkamp and Marc Schlipsing and Jan Salmen and Christian Igel}, booktitle = {IEEE International Joint Conference on Neural Networks},
title = {The {G}erman {T}raffic {S}ign {R}ecognition {B}enchmark: A multi-class classification competition}, year = {2011}, pages = {1453--1460}}

CONVOLUTIONAL NEURAL NETWORK
MAXPOOLING/FLATTENING

- Pooling or down sampling layers are placed after convolutional layers to **reduce feature map dimensionality**.
- This improves the computational efficiency while preserving the features.
- Pooling helps the model to generalize by avoiding overfitting. If one of the pixel is shifted, the pooled feature map will still be the same.
- Max pooling works by retaining the **maximum feature response** within a given sample size in a feature map.
- Live illustration : http://scs.ryerson.ca/~aharley/vis/conv/flat.html

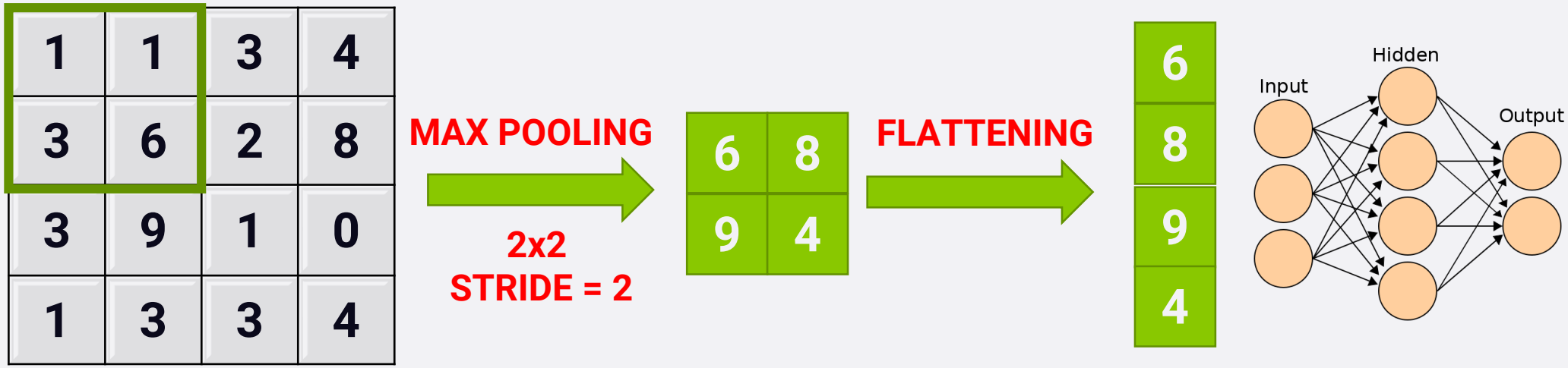
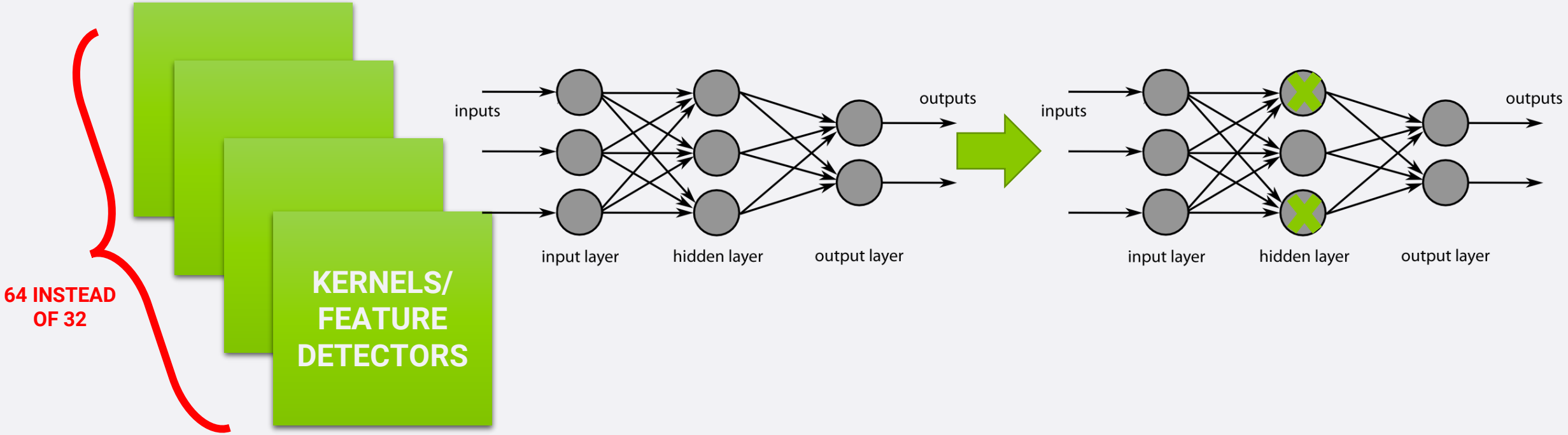


Photo Credit: https://commons.wikimedia.org/wiki/File:Artificial_neural_network.svg

CONVOLUTIONAL NEURAL NETWORK
INCREASE FILTERS/DROPOUT

- Improve accuracy by adding more feature detectors/filters or adding a dropout.
- Dropout refers to dropping out units in a neural network.
- Neurons develop co-dependency amongst each other during training
- Dropout is a regularization technique for reducing overfitting in neural networks.
- It enables training to occur on several architectures of the neural network



- **Photo Credit:** https://fr.m.wikipedia.org/wiki/Fichier:MultiLayerNeuralNetworkBigger_english.png