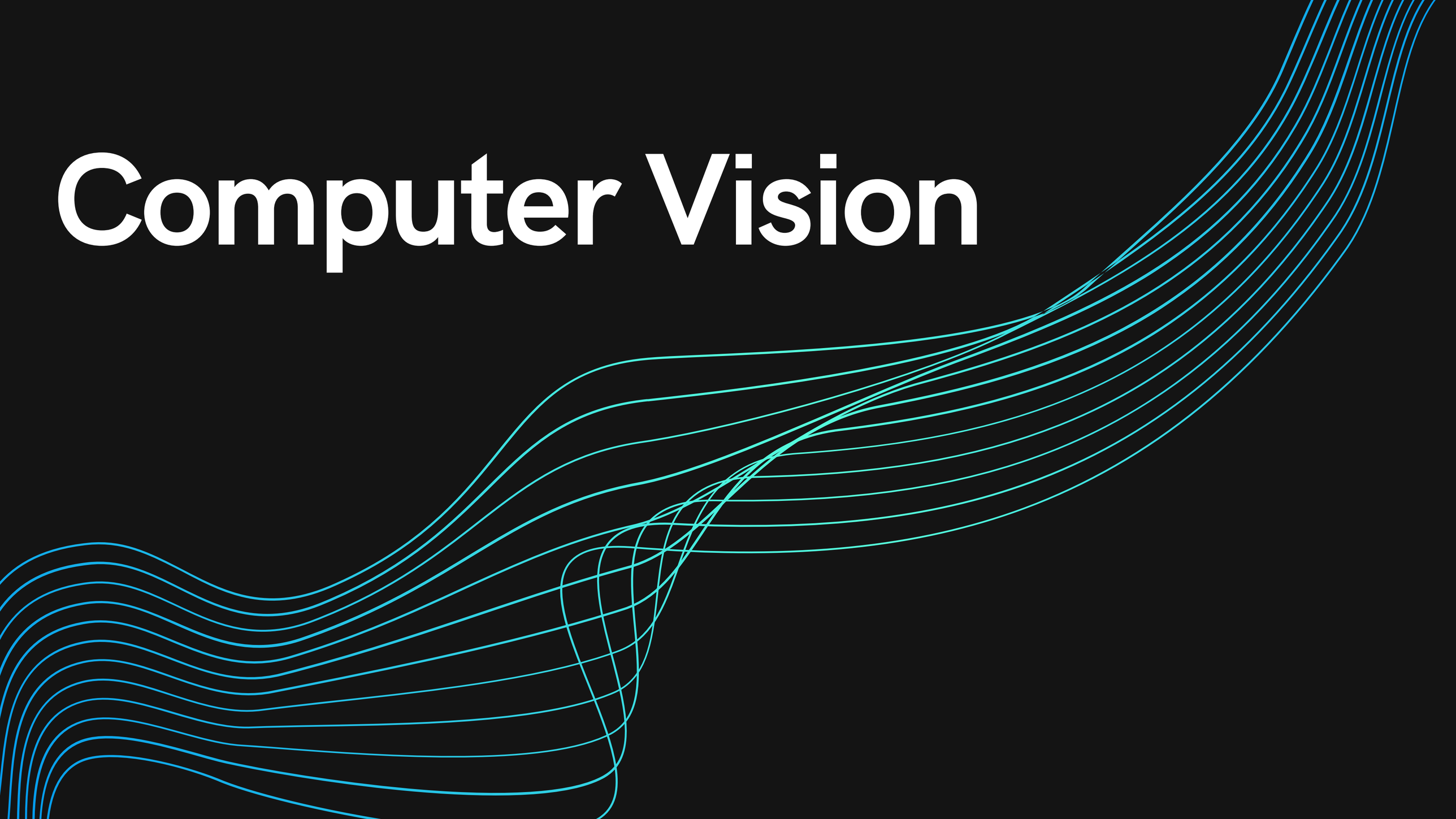
Computer Vision

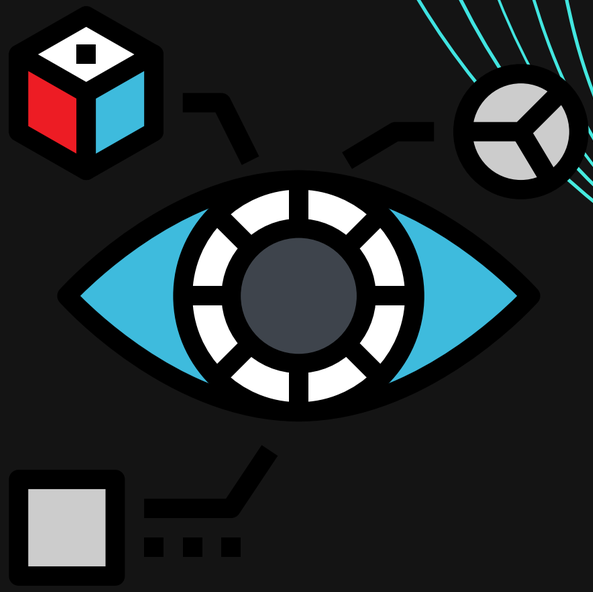


What is CV?

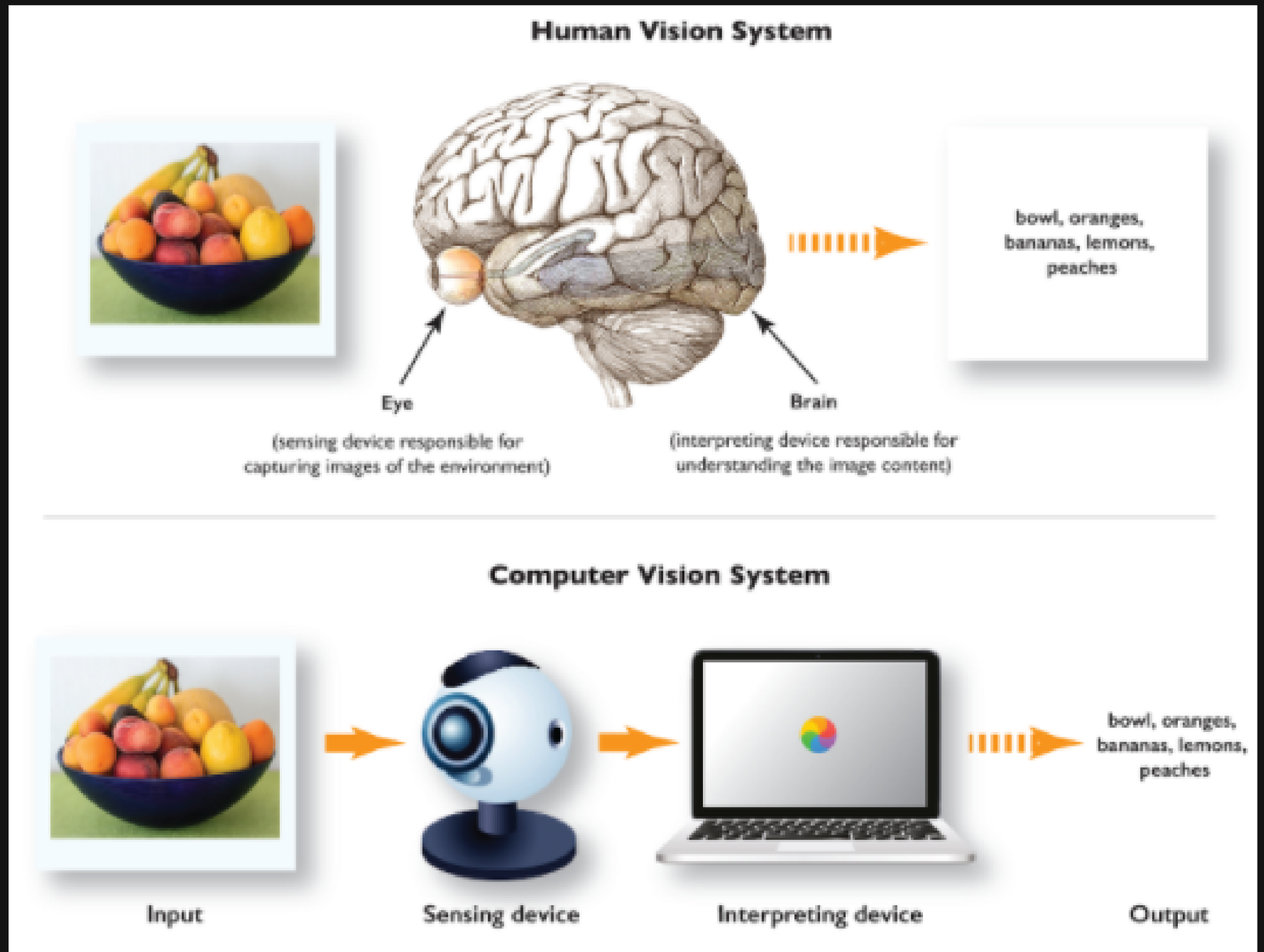
- An artificial intelligence workspace where we can collect information and extract features of images in digital media.
- Goal is to perform operations on images
- Draw parallels between human brain and computers

How do computers see?

- Computers use some algorithms to detect images in digital media.
- Images in digital media are made up of pixels.
- Pixels in any image have a color, a coordinate and each pixel has its own identity.
- On its ID, its writes coordinate and color information.
- This is how computers can detect and identify images.

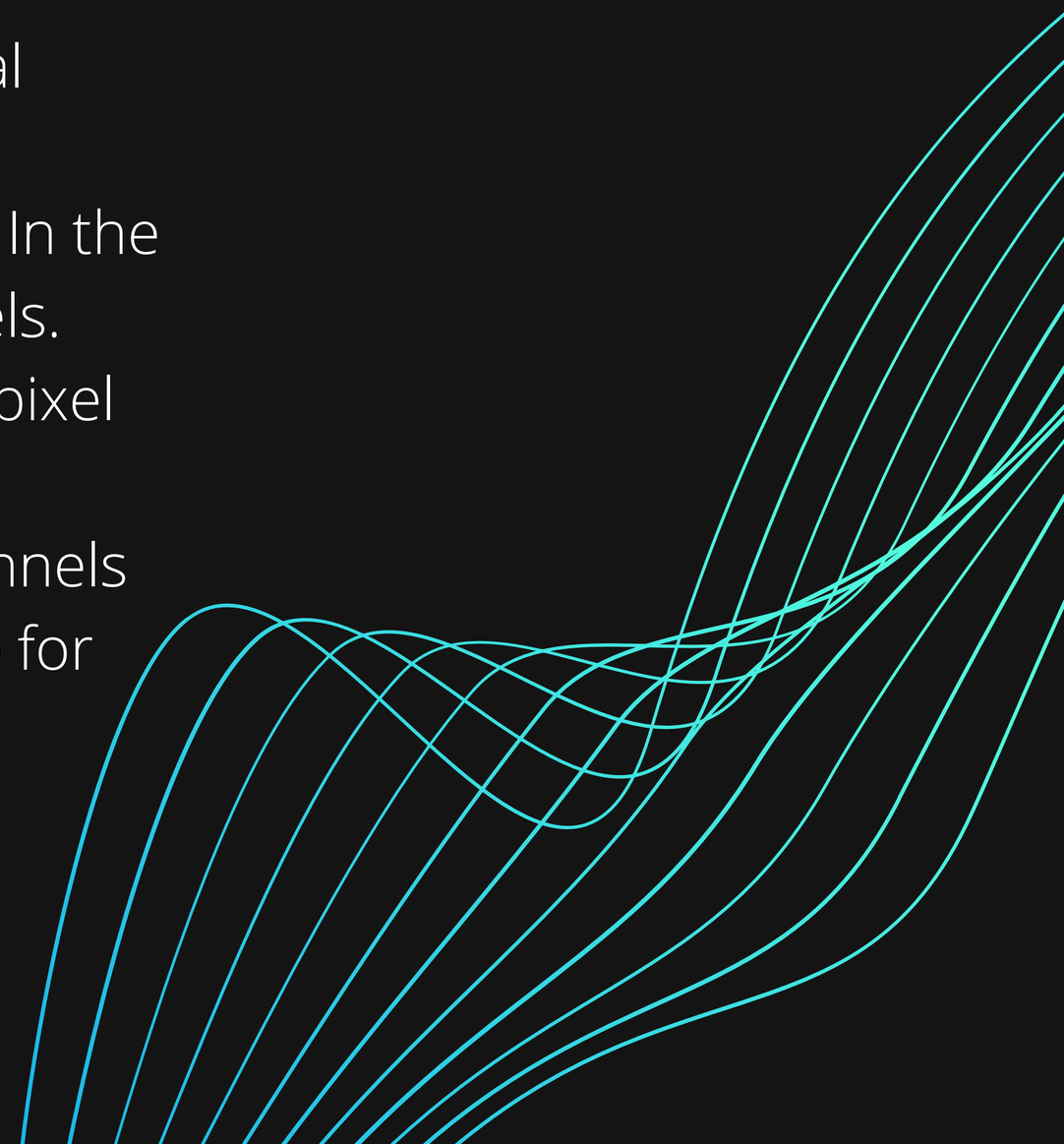


How do computers see?

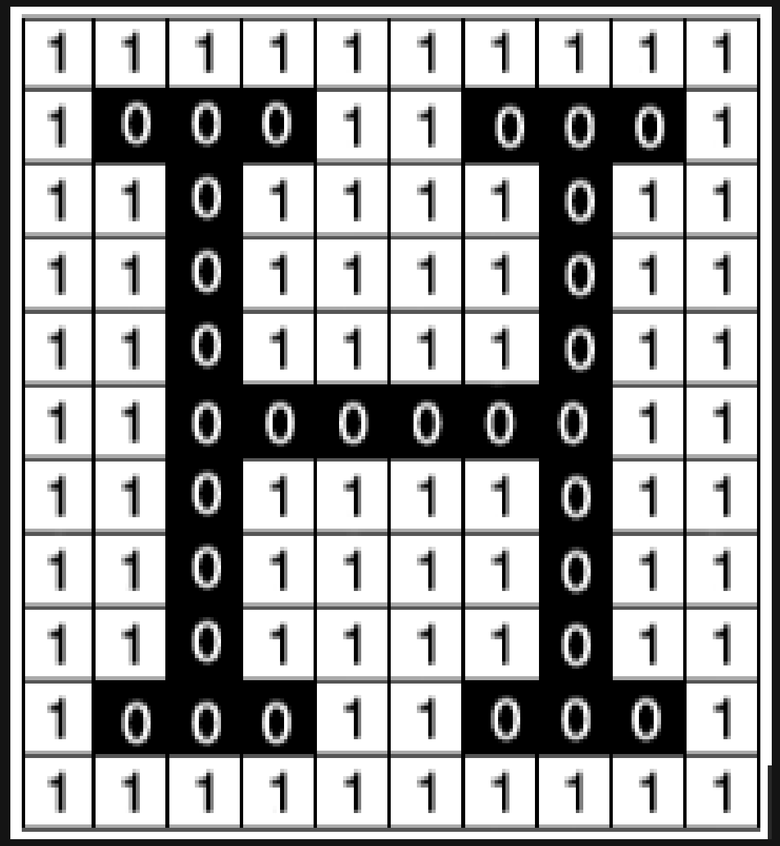


Understanding Images

- An image can be represented as a multidimensional array.
- A generic word is used called pixels or pixel values. In the case of color images we have three colored channels.
- Colored images will have multiple values for single-pixel values.
- The color values go from 0 to 255. These color channels are generally represented as Red Green Blue (RGB) for instance.



Understanding Images



How does CV work?

1. ACQURING IMAGE

Images are acquired in real-time through video, photos or 3D technology for analysis.

2. PROCESSING IMAGE

Models are trained using deep learning and labelled data

3. UNDERSTANDING IMAGE

Interpretative step - identify or classify image

APPLICATIONS

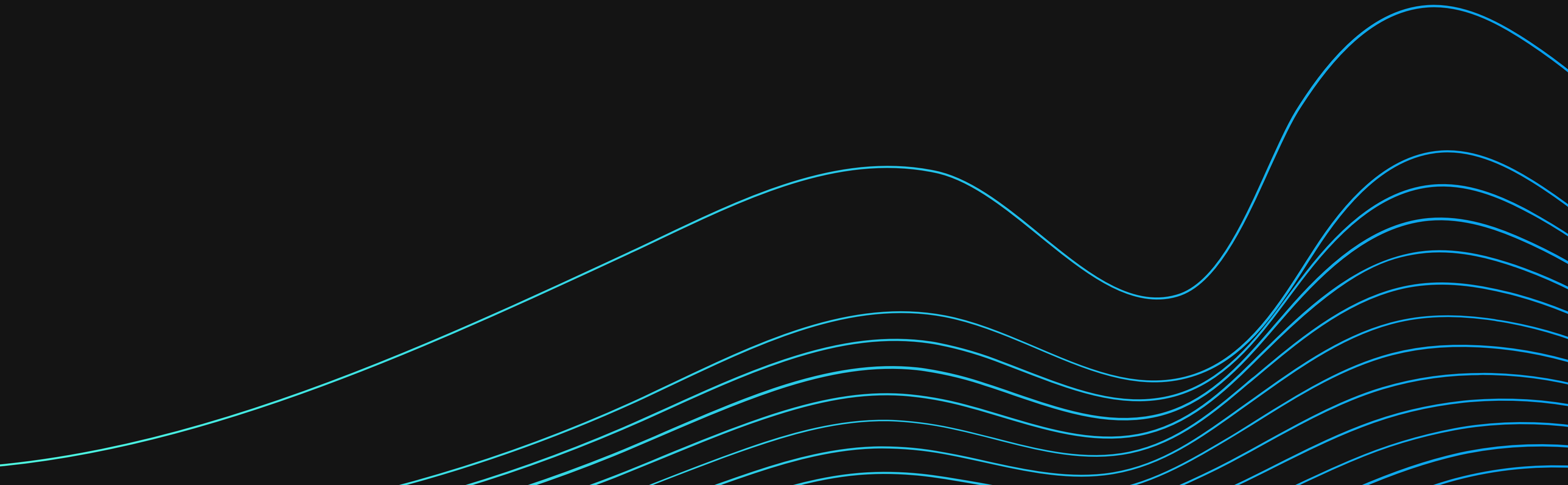
- FACE DETECTION
- CANCER DETECTION
- COVID 19 ANALYSIS
- IMAGE CLASSIFICATION
- TRAFFIC FLOW ANALYSIS

APPLICATIONS

- PARK OCCUPANCY DETECTION
- AUTOMATED LICENESEPLATE RECOGNITION
- BIOMETRIC ANALYSIS
- MOVEMENT ANALYSIS
- BIOTECHNOLOGY

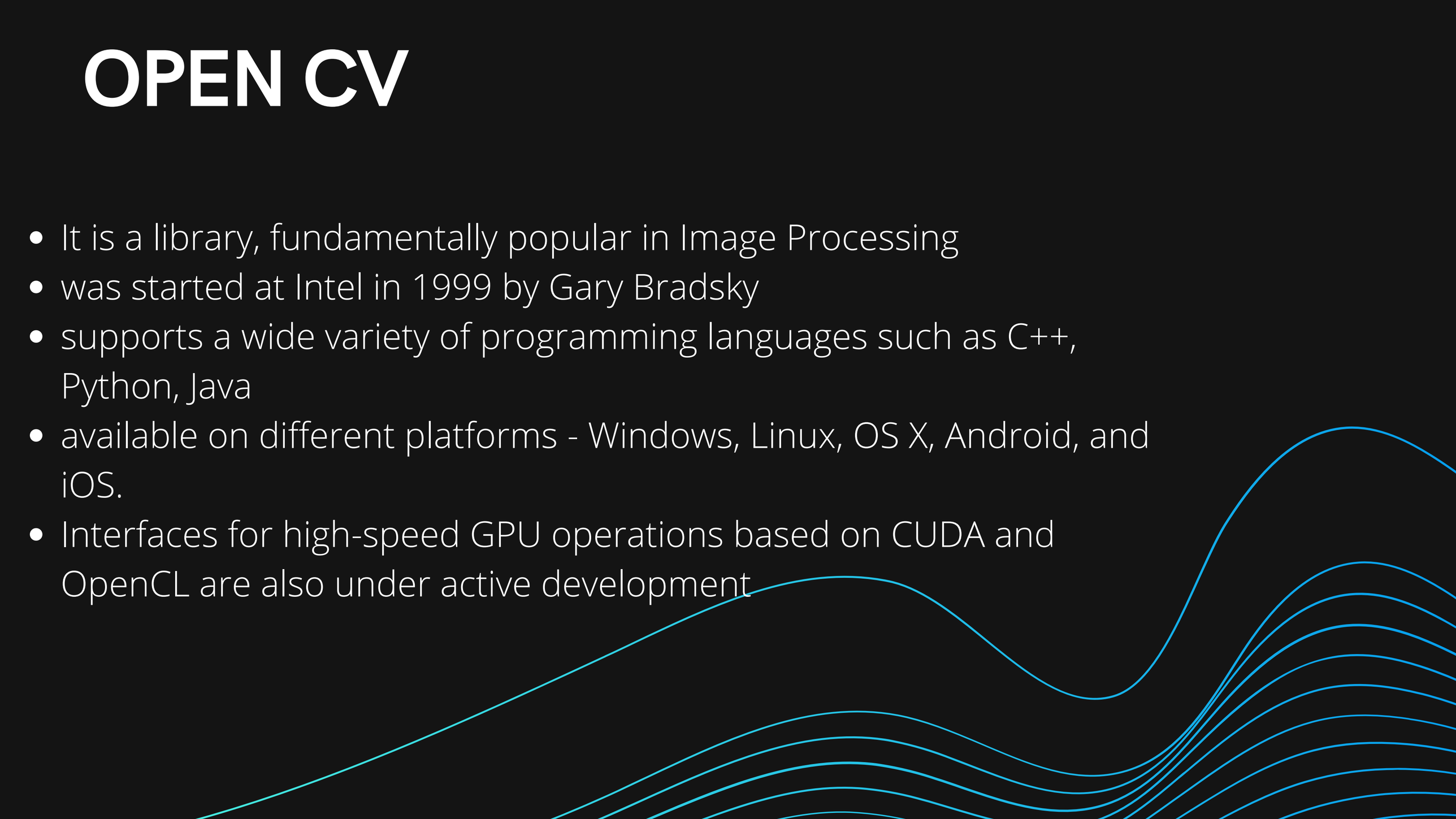
OPEN CV

OPEN SOURCE COMPUTER VISION LIBRARY



OPEN CV

- It is a library, fundamentally popular in Image Processing
- was started at Intel in 1999 by Gary Bradsky
- supports a wide variety of programming languages such as C++, Python, Java
- available on different platforms - Windows, Linux, OS X, Android, and iOS.
- Interfaces for high-speed GPU operations based on CUDA and OpenCL are also under active development

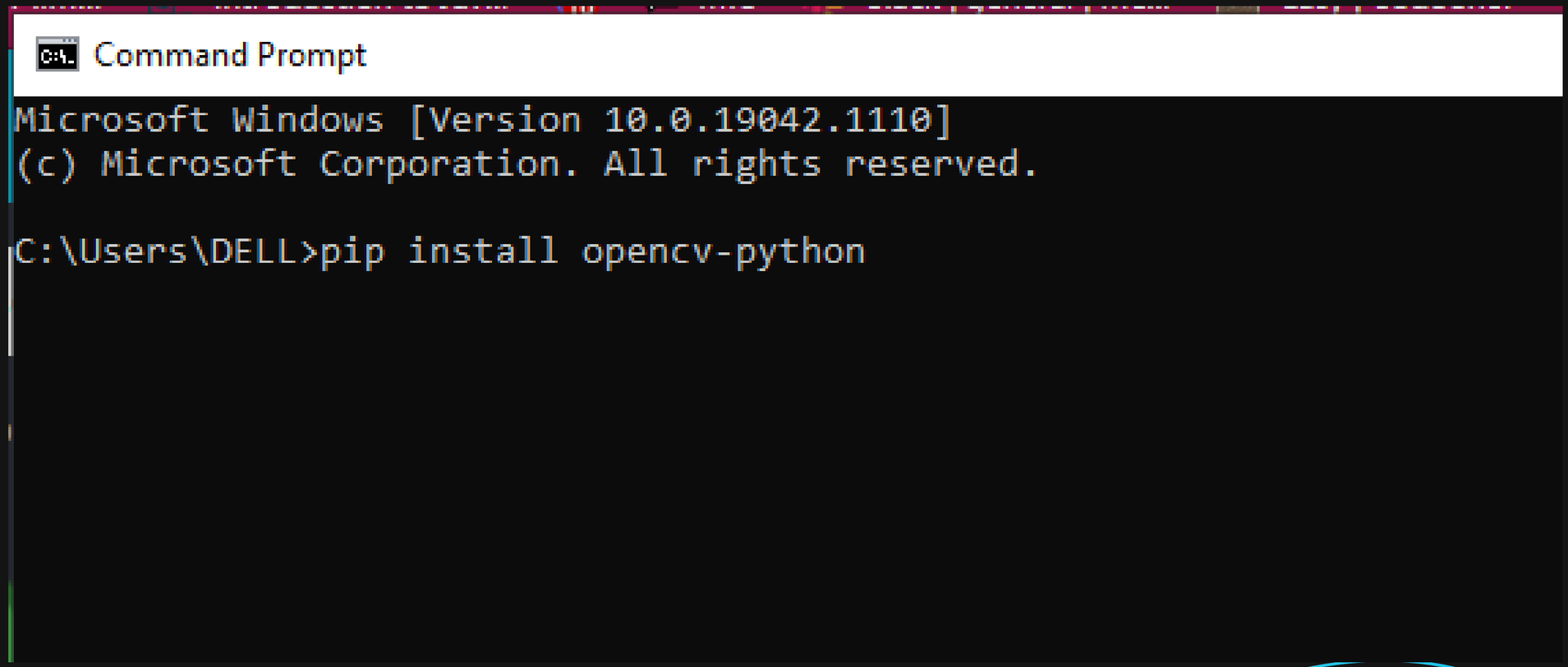


Installation

- Open the command terminal
- Enter the following command

```
pip install opencv-python
```

Installation



Reading an Image

- **cv2.imread()** - Used to read the image, function takes as an argument the path to the file from which we got the image
- **cv2.namedWindow()** - Image opens in a visual window, function takes the name of the window as its first argument.
- **cv2.imshow( ) -** used to display the current image on the screen, It takes two arguments. The first is the name of the visual we are going to show, and the second is the object it is registered in.
- **cv2.waitkey(0)** - This function takes the number value in milliseconds. When we write 0 here, it means we can close the window at any time.
- **cv2.destroyAllWindows()** - for advanced projects we can forget to close many windows that open on the screen. This function avoids this.

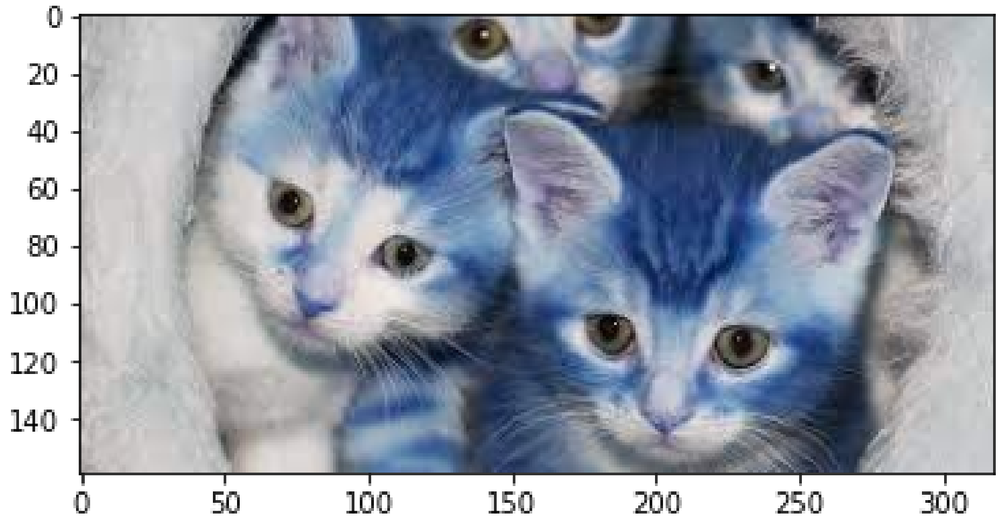
Reading an Image

```
In [13]: import cv2
         import matplotlib.pyplot as plt

In [14]: image = cv2.imread('cat.jpg')

In [15]: plt.imshow(image)

Out[15]: <matplotlib.image.AxesImage at 0x1b6c53ba3a0>
```



Reading an Image

We note that By default, the *imread* function reads images in the BGR (Blue-Green-Red) format.

We can read images in different formats using extra flags in the imread function:

- **cv2.IMREAD_COLOR:** Default flag for loading a color image.
- **cv2.IMREAD_GRAYSCALE**: Loads images in grayscale format**.**
- **WHY SWAP** - BGR by OpenCV, but Matplotlib’s plot expects RGB, for a correct display of the image, it is necessary to swap those channels. Also use cv2.cvtColor() for same

Reading an Image

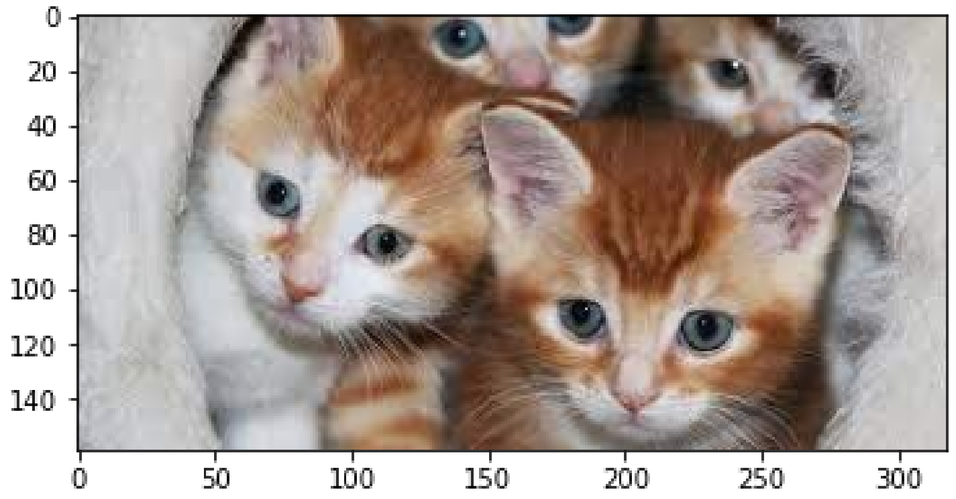
```
In [13]: import cv2
         import matplotlib.pyplot as plt

In [14]: image = cv2.imread('cat.jpg')

In [16]: img_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

In [17]: plt.imshow(img_rgb)

Out[17]: <matplotlib.image.AxesImage at 0x1b6c541f490>
```

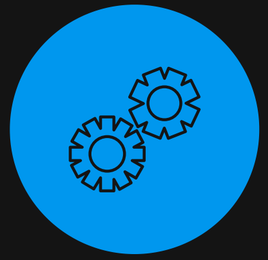


Important Tools of OpenCV

Geometry

Edge Detection

GrayScale



Blurring

Geometry

- Work around with geometry of images - resize, rotate, flip, etc

RESIZING

- We can resize our images by shrinking them in or zooming out
- cv2.INTER_AREA is used for shrinking
- cv2.INTER_CUBIC is used for zooming

RESIZING

```
In [20]: (hgt, wth) = img_rgb.shape[:2]

In [28]: res_img = cv2.resize(img_rgb, (int(wth / 2), int(hgt / 2)), interpolation = cv2.INTER_CUBIC)

In [29]: plt.imshow(res_img)

Out[29]: <matplotlib.image.AxesImage at 0x1b6c558de20>
```

IMAGE ROTATION/FLIPPING

- Data Augmentation allows us to generate more samples for training our model.
- It uses available data samples to produce the new ones, by applying image operations like rotation, scaling, translation, etc.
- IDuring the Data Augmentation technique Rotation or flip plays a significant role.
- It rotates the image at a specified angle by keeping labels the same.

IMAGE ROTATION/FLIPPING

```
# Along central x axis
flip_img = cv2.flip(img_rgb,0)
plt.imshow(flip_img)
```

```
<matplotlib.image.AxesImage at 0x1b6c55f09a0>
```

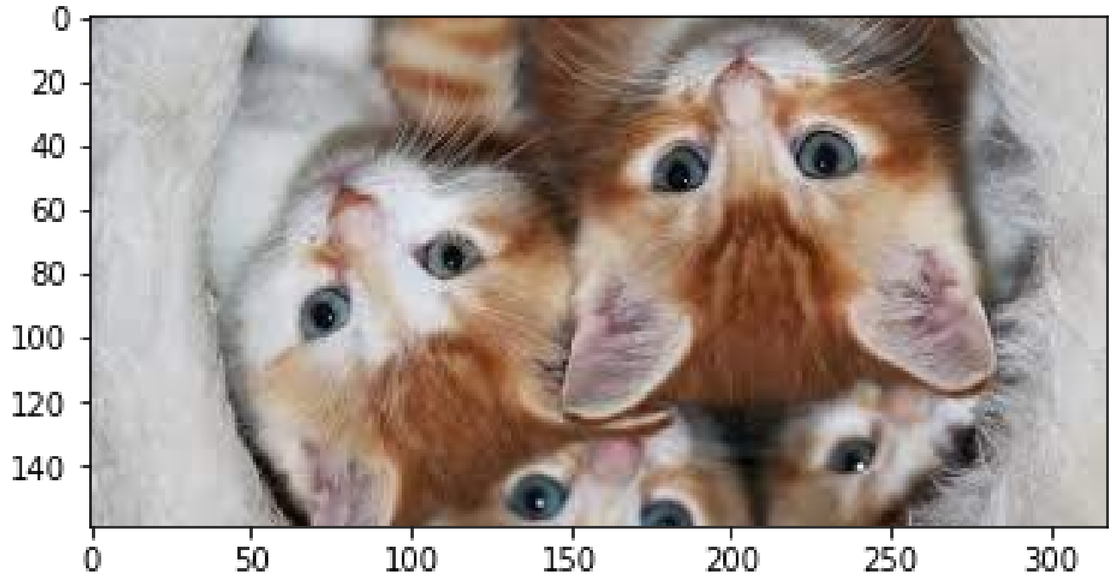
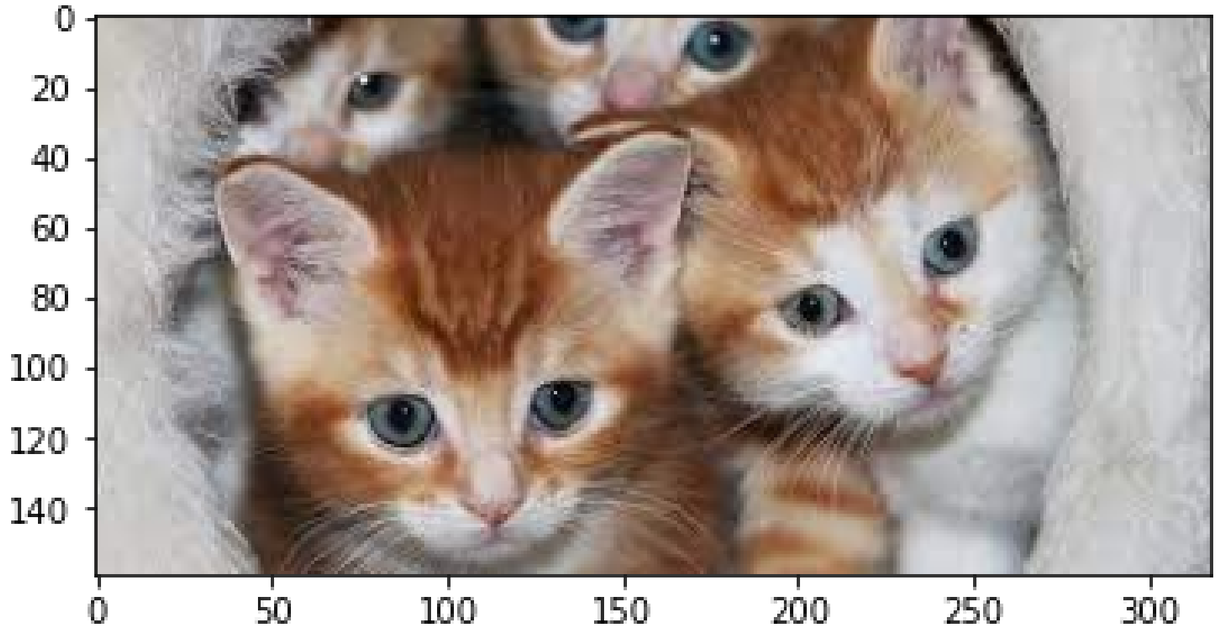


IMAGE ROTATION/FLIPPING

```
# Along central y axis
new_img_y = cv2.flip(img_rgb,1)
plt.imshow(new_img_y)
```

```
<matplotlib.image.AxesImage at 0x1b6c53e8df0>
```



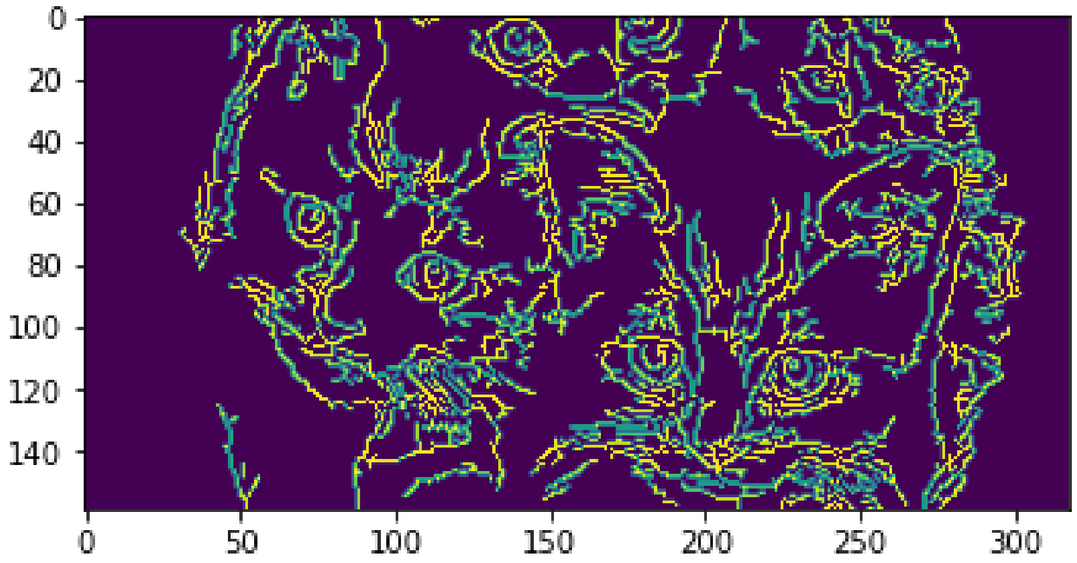
EDGE DETECTION

- The process of image detection involves detecting sharp edges in the image.
- This edge detection is essential in context of image recognition or object localization/detection.
- Many algorithms for detecting edges
- one such algorithm known as Canny Edge Detection.

EDGE DETECTION

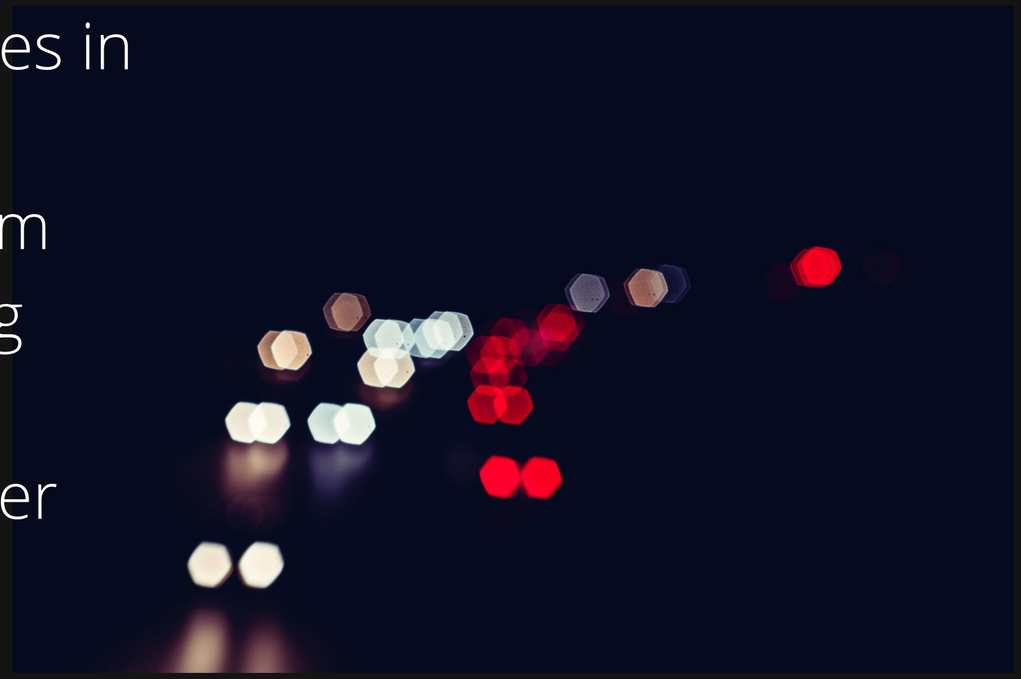
```
In [32]: edges = cv2.Canny(img_rgb, 100, 200)
         plt.imshow(edges)
```

```
Out[32]: <matplotlib.image.AxesImage at 0x1b6c5389910>
```



BLURRING AND SMOOTHING

- One of the most popular and common techniques in order to reduce noise in the image.
- Removes high-frequency content, like edges, from the image and commonly used image processing operation for reducing the image noise.
- obtained by convoluting the input image by a filter kernel having a low pass.



BLURRING ADVANTAGES

- It helps in Noise removal - as noise is considered as high pass signal so by the application of low pass filter kernel we restrict noise.
- Low intensity edges are removed.
- It helps in hiding the details when necessary.

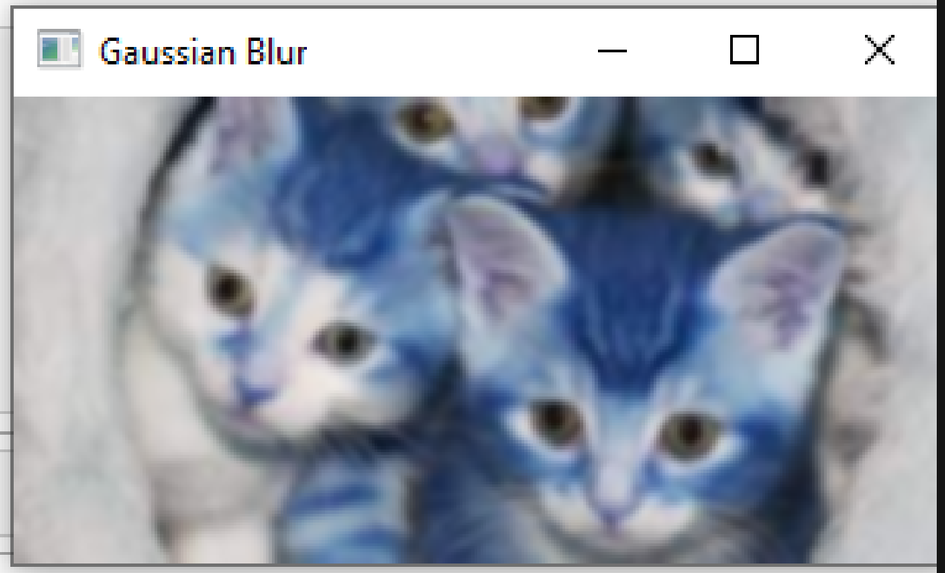
TYPES OF BLURRING

- **Gaussian Blurring:**
  - Gaussian blur is the result of blurring an image by a Gaussian function.
  - Widely used effect in graphics software, to reduce image noise and reduce detail
  - Also used as a preprocessing stage before applying our machine learning models.
  - E.g. of a Gaussian kernel(3×3)

TYPES OF BLURRING

```
[*]:
# Gaussian Blur
Gaussian_image = cv2.GaussianBlur(img_rgb, (7, 7), 0)
cv2.imshow('Gaussian Blur', Gaussian_image)
cv2.waitKey(0)
```

```
[ ]:
```

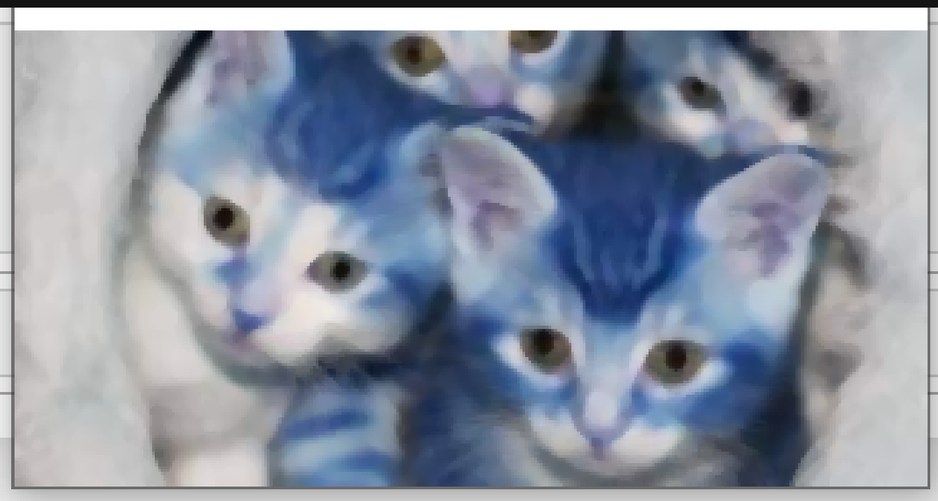


TYPES OF BLURRING

- **Median Blur:**
  - a non-linear digital filtering technique, often used to remove noise from an image or signal.
  - widely used in digital image processing because as it preserves edges while removing noise.

TYPES OF BLURRING

```
median = cv2.medianBlur(img_rgb, 5)
cv2.imshow('Median Blurring', median)
cv2.waitKey(0)
```

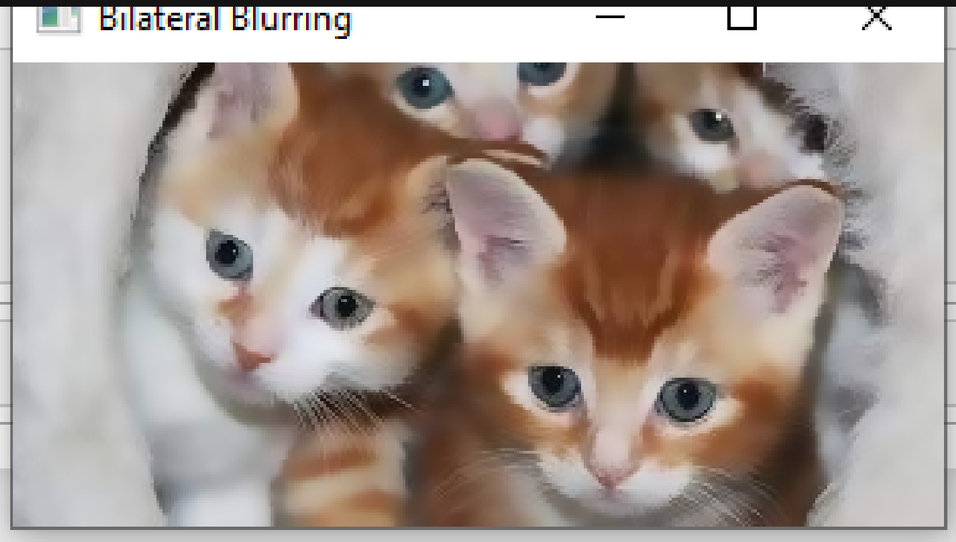


TYPES OF BLURRING

- **Bilateral Blur:**
  - A bilateral filter is a non-linear, edge-preserving, and noise-reducing smoothing filter for images.
  - It replaces the intensity of each pixel with a weighted average of intensity values from nearby pixels.
  - This weight can be based on a Gaussian distribution.
  - Thus, sharp edges are preserved while discarding the weak ones.
-

TYPES OF BLURRING

```
bilateral = cv2.bilateralFilter(image, 9, 75, 75)
cv2.imshow('Bilateral Blurring', bilateral)
cv2.waitKey(0)
```



Face Detection

- OpenCV is used to detect faces using a haar cascade based object detection algorithm.
- Haar cascades are basically trained machine learning classifiers model that calculates different features like lines, contours, edges, etc.

Face Detection

```
facecascade=cv2.CascadeClassifier(cv2.data.haarcascades+'haarcascade_frontalface_default.xml')
```

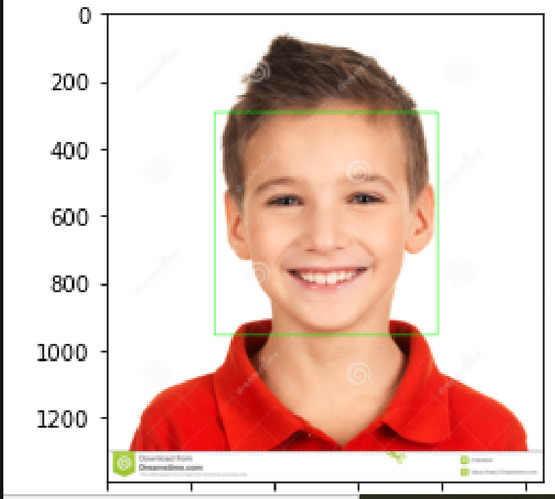
```
gray=cv2.cvtColor(happy_image,cv2.COLOR_BGR2GRAY)
```

```
faces=facecascade.detectMultiScale(gray,1.1,4)
```

```
for(x,y,w,h) in faces:
    cv2.rectangle(happy_image,(x,y),(x+w,y+h),(0,255,0),2)
```

```
plt.imshow(cv2.cvtColor(happy_image,cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x2ae34553ac0>
```



GrayScaling

- The process of converting an image from other color spaces to shades of gray. It varies between complete black and complete white.

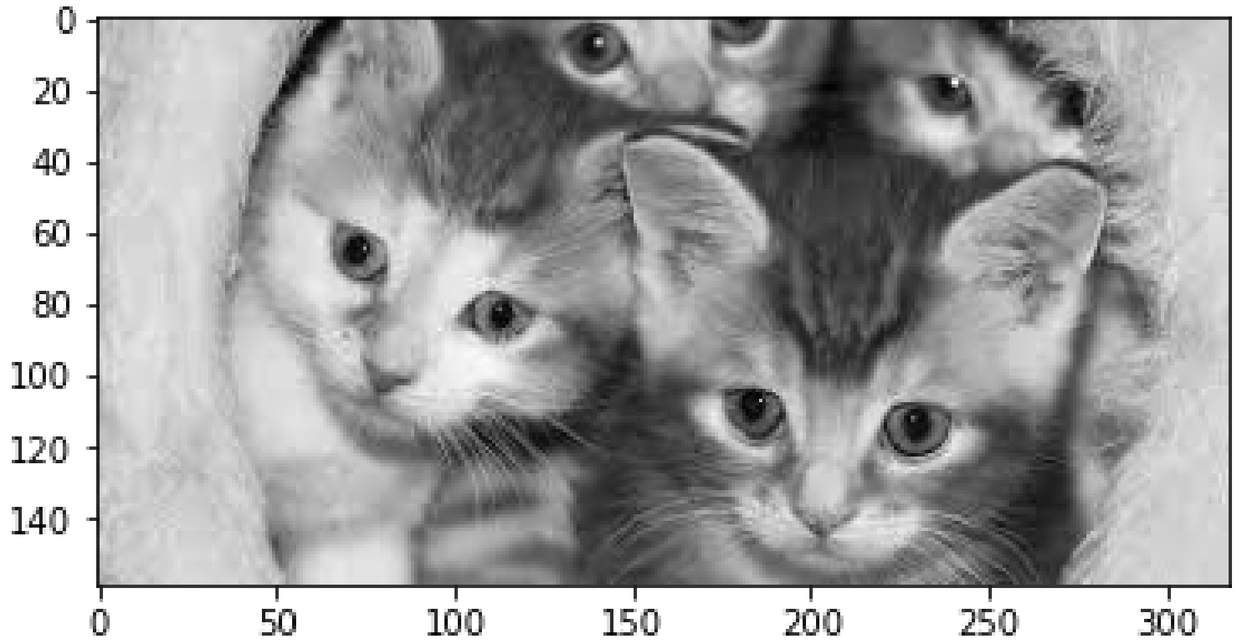
Importance of GrayScaling

- **Importance of grayscaling**
  - **Dimension reduction:** For example, In RGB images there are three color channels and has three dimensions while grayscale images are single-dimensional.
  - **Reduces model complexity:** Consider training neural article on RGB images of 10x10x3 pixel. The input layer will have 300 input nodes. On the other hand, the same neural network will need only 100 input nodes for grayscale images.
  - **For other algorithms to work:** Many algorithms are customized to work only on grayscale images e.g. Canny edge detection function pre-implemented in OpenCV library works on Grayscale images only.

GrayScaling

```
In [14]: gray_image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
         plt.imshow(gray_image, cmap = 'gray')
```

```
Out[14]: <matplotlib.image.AxesImage at 0x26b137c4d60>
```



DRAWING SHAPES

- We can use OpenCV to draw boundaries and shapes in our images
- Rectangle is most commonly used shape

Drawing A Rectangle

- Syntax: cv2.rectangle(image, start_point, end_point, color, thickness)
- Parameters:
  - image: It is the image on which rectangle is to be drawn.
  - start_point: It is the starting coordinates of rectangle. The coordinates are represented as tuples of two values i.e. (X coordinate value, Y coordinate value).
  -

Drawing A Rectangle

- end_point: It is the ending coordinates of rectangle. The coordinates are represented as tuples of two values i.e. (X coordinate value, Y coordinate value).
- color: It is the color of border line of rectangle to be drawn. For BGR, we pass a tuple. eg: (255, 0, 0) for blue color.
- thickness: It is the thickness of the rectangle border line in px. Thickness of -1 px will fill the rectangle shape by the specified color.
- Return Value: It returns an image.

Drawing A Rectangle

```
#window_name = 'Rect'

start_point = (50, 50)

end_point = (100, 100)

color = (255, 0, 0)

thickness = 2

image = cv2.rectangle(img_rgb, start_point, end_point, color, thickness)

plt.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x224d5170e50>
```