03 Computer Vision With OpenCV

The next library we're going to look at is called Kraken, which was developed by Université PSL in Paris. It's actually based on a slightly older code base, **OCRopus**. You can see how the flexible open-source licenses allow new ideas to grow by building upon older ideas. And, in this case, I fully support the idea that the Kraken - a mythical massive sea creature - is the natural progression of an octopus!

03-01: Release the Kraken!

Purpose of Kraken*

What we are going to use Kraken for is to **detect lines of text as bounding boxes** in a given image. The biggest limitation of tesseract is the *lack of a layout engine inside of it*. Tesseract expects to be using fairly clean text, and gets confused if we don't crop out other artifacts. It's not bad, but Kraken can help us out be segmenting pages. Lets take a look.

*Please note that Kraken is only supported on Linux and Mac OS X, it is not supported on Windows. Documentation and Installation Notes can be found at: https://pypi.org/project/kraken/

Importing Kraken Module

In [1]:

```
# First, we'll take a look at the kraken module itself
import kraken
help(kraken)
```

```
Help on package kraken:

NAME
    kraken - entry point for kraken functionality

PACKAGE CONTENTS
    binarization
    ketos
    kraken
    lib (package)
    linegen
    pageseg
    repo
    rpred
    serialization
    transcribe

DATA
    absolute_import = _Feature((2, 5, 0, 'alpha', 1), (3, 0, 0, 'alp
ha', 0...
    division = _Feature((2, 2, 0, 'alpha', 2), (3, 0, 0, 'alpha',
0), 8192...
    print_function = _Feature((2, 6, 0, 'alpha', 2), (3, 0, 0, 'alph
a', 0)...

FILE
    /opt/conda/lib/python3.7/site-packages/kraken/__init__.py
```

Note the package contents in Kraken.

Importing `pageseg`

There isn't much of a discussion here, but there are a number of sub-modules that look interesting. I spend a bit of time on their website, and I think the pageseg module, which handles all of the page segmentation, is the one we want to use.

In [2]:

```
from kraken import pageseg
help(pageseg)
```

```
Help on module kraken.pageseg in kraken:

NAME
    kraken.pageseg

DESCRIPTION
    kraken.pageseg
    ~~~~~~~~~~~~~~

    Layout analysis and script detection methods.

FUNCTIONS
    detect_scripts(im, bounds, model='/opt/conda/lib/python3.7/site-
packages/kraken/script.mlmodel', valid_scripts=None)
        Detects scripts in a segmented page.

        Classifies lines returned by the page segmenter into runs of
scripts/writing systems.

        Args:
            im (PIL.Image): A bi-level page of mode '1' or 'L'
            bounds (dict): A dictionary containing a 'boxes' entry w
ith a list of
                           coordinates (x0, y0, x1, y1) of a text li
ne in the image
                           and an entry 'text_direction' containing
                           'horizontal-lr/rl/vertical-lr/rl'.
            model (str): Location of the script classification model
or None for default.
            valid_scripts (list): List of valid scripts.

        Returns:
            {'script_detection': True, 'text_direction': '$dir', 'bo
xes':
            [[(script, (x1, y1, x2, y2)),...]]}: A dictionary contai
ning the text
            direction and a list of lists of reading order sorted bo
unding boxes
            under the key 'boxes' with each list containing the scri
pt segmentation
            of a single line. Script is a ISO15924 4 character ident
ifier.

        Raises:
            KrakenInvalidModelException if no clstm module is availa
ble.

    segment(im, text_direction='horizontal-lr', scale=None, maxcolse
ps=2, black_colseps=False, no_hlines=True, pad=0, mask=None)
        Segments a page into text lines.

        Segments a page into text lines and returns the absolute coo
rdinates of
        each line in reading order.

        Args:
            im (PIL.Image): A bi-level page of mode '1' or 'L'
            text_direction (str): Principal direction of the text
                                  (horizontal-lr/rl/vertical-lr/rl)
            scale (float): Scale of the image
            maxcolseps (int): Maximum number of whitespace column se
```

```
parators
            black_colseps (bool): Whether column separators are assu
med to be
                                  vertical black lines or not
            no_hlines (bool): Switch for horizontal line removal
            pad (int or tuple): Padding to add to line bounding boxe
s. If int the
                                same padding is used both left and r
ight. If a
                                2-tuple, uses (padding_left, padding
_right).
            mask (PIL.Image): A bi-level mask image of the same size
as `im` where
                              0-valued regions are ignored for segme
ntation
                              purposes. Disables column detection.

        Returns:
            {'text_direction': '$dir', 'boxes': [(x1, y1, x2, y
2),...]}: A
            dictionary containing the text direction and a list of r
eading order
            sorted bounding boxes under the key 'boxes'.

        Raises:
            KrakenInputException if the input image is not binarized
or the text
            direction is invalid.

DATA
    __all__ = ['segment', 'detect_scripts']

FILE
    /opt/conda/lib/python3.7/site-packages/kraken/pageseg.py
```

So it looks like there are a few different functions we can call, and the segment function looks particularly appropriate.

The `kraken.pageseg` library is on the documentation front -- I can see immediately that we are working with *PIL.Image files*, and the author has even indicated that we need to pass in **either a binarized (e.g. '1') or grayscale (e.g. 'L') image**.

We can also see that the **return value is a dictionary object with two keys**, "text_direction" which will return to us...

- a string of the direction of the text,
- and "boxes" which appears to be a list of tuples,

where each tuple is a box in the original image.

Importing A MultiColumn Image With Text

This is an excerpt taken from an on-campus draft paper from the University of Michigan.

In [3]:

```
from PIL import Image
im=Image.open("readonly/two_col.png")
# Lets display the image inline
display(im)
# Lets now convert it to black and white and segment it up into lines with krake
n
bounding_boxes=pageseg.segment(im.convert('1'))['boxes']
# And lets print those lines to the screen
print(bounding_boxes)
```

CALEB CHADWELL
Daily Staff Reporter

In a statement Tuesday, the Department of Public Safety and Security wrote that DPSS was not aware until Saturday afternoon of an assault on a University lecturer last week, referred in testimony before Ann Arbor City Council Monday.

Khita Whyatt, lecturer of dance in the University of Michigan's School of Music, Theatre & Dance, said in an interview after her testimony on the incident that she did not immediately call the police because she was so shocked, but her department chair contacted the DPSS. Two days after the incident, Whyatt said she was interviewed by two DPSS officers. During her testimony to Council, she called on the University to release a crime report about how she was knocked down and intimidated by unknown assailants. The event follows similar incidents where crime alerts had not been released. The University has released two crime alerts of hate crimes on campus over the past two weeks.

Whyatt wrote in an email sent Tuesday afternoon to recipients including University President Mark Schlissel as well as The Michigan Daily that she waited until Saturday morning to report the assault to police because she was disoriented and did not know where to reach out.

"I did wait until Saturday morning to get in touch to report the incident," Whyatt wrote. "I was in shock and still processing what to do prior to reaching out ... it was also obvious that there was no way that these boys were going to be caught. Not being a student, I did not know who to report to. That must seem obvious by the fact that I

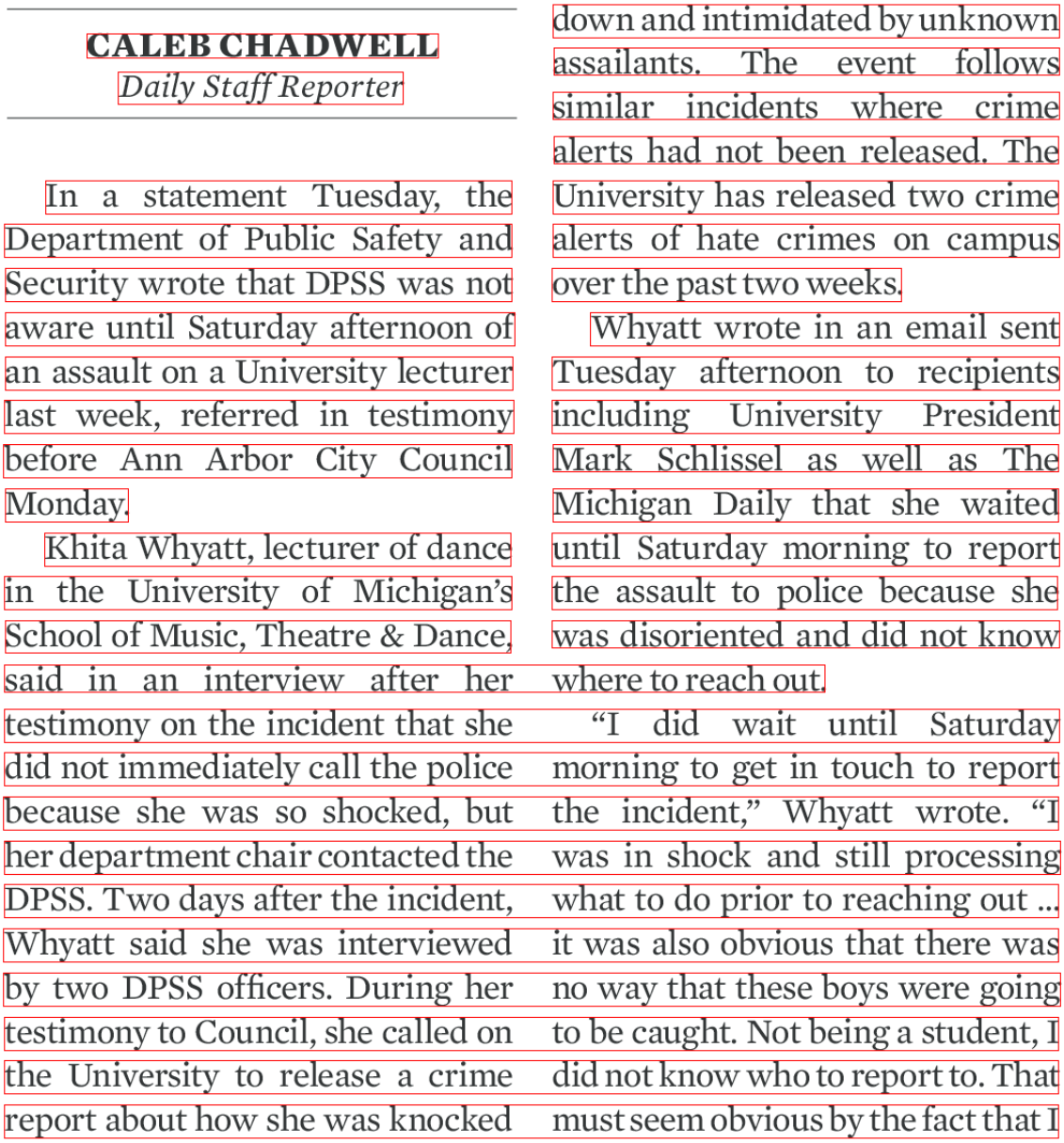
```
[[100, 50, 449, 74], [131, 88, 414, 120], [59, 196, 522, 229], [18,
239, 522, 272], [19, 283, 522, 316], [19, 327, 525, 360], [19, 371,
523, 404], [18, 414, 524, 447], [17, 458, 522, 491], [19, 502, 141,
535], [58, 546, 521, 579], [18, 589, 522, 622], [19, 633, 521, 665],
[563, 21, 1066, 54], [564, 64, 1066, 91], [563, 108, 1066, 135], [56
4, 152, 1065, 179], [563, 196, 1065, 229], [563, 239, 1066, 272], [5
62, 283, 909, 316], [600, 327, 1066, 360], [562, 371, 1066, 404], [5
62, 414, 1066, 447], [563, 458, 1065, 485], [563, 502, 1065, 535],
[562, 546, 1066, 579], [562, 589, 1064, 622], [562, 633, 1066, 660],
[18, 677, 833, 704], [18, 721, 1066, 754], [18, 764, 1065, 797], [1
7, 808, 1065, 841], [18, 852, 1067, 885], [18, 895, 1065, 928], [17,
939, 1065, 972], [17, 983, 1067, 1016], [18, 1027, 1065, 1060], [18,
1070, 1065, 1103], [18, 1114, 1065, 1147]]
```

Printing Bounding Boxes To Show Segregation

Let's write a little routine to try and see the effects a bit more clearly. Add a little documentation using triple quoted strings `''' '''`. Basically, this series of code will colour the boxes red.

In [4]:

```
def show_boxes(img):
    '''Modifies the passed image to show a series of bounding boxes on an image
 as run by kraken

    :param img: A PIL.Image object
    :return img: The modified PIL.Image object
    '''
    # Lets bring in our ImageDraw object
    from PIL import ImageDraw
    # And grab a drawing object to annotate that image
    drawing_object=ImageDraw.Draw(img)
    # We can create a set of boxes using pageseg.segment
    bounding_boxes=pageseg.segment(img.convert('1'))['boxes']
    # Now lets go through the list of bounding boxes
    for box in bounding_boxes:
        # An just draw a nice rectangle
        drawing_object.rectangle(box, fill = None, outline ='red')
    # And to make it easy, lets return the image object
    return img

# To test this, lets use display
display(show_boxes(Image.open("readonly/two_col.png")))
```



CALEB CHADWELL
Daily Staff Reporter

In a statement Tuesday, the Department of Public Safety and Security wrote that DPSS was not aware until Saturday afternoon of an assault on a University lecturer last week, referred in testimony before Ann Arbor City Council Monday.

Khita Whyatt, lecturer of dance in the University of Michigan's School of Music, Theatre & Dance, said in an interview after her testimony on the incident that she did not immediately call the police because she was so shocked, but her department chair contacted the DPSS. Two days after the incident, Whyatt said she was interviewed by two DPSS officers. During her testimony to Council, she called on the University to release a crime report about how she was knocked down and intimidated by unknown assailants. The event follows similar incidents where crime alerts had not been released. The University has released two crime alerts of hate crimes on campus over the past two weeks.

Whyatt wrote in an email sent Tuesday afternoon to recipients including University President Mark Schlissel as well as The Michigan Daily that she waited until Saturday morning to report the assault to police because she was disoriented and did not know where to reach out.

"I did wait until Saturday morning to get in touch to report the incident," Whyatt wrote. "I was in shock and still processing what to do prior to reaching out ... it was also obvious that there was no way that these boys were going to be caught. Not being a student, I did not know who to report to. That must seem obvious by the fact that I

Analysis of Kraken's Performance

Kraken isn't completely sure what to do with this two column format. In some cases, kraken has identified a line in just a single column, while in other cases kraken has spanned the line marker all the way across the page.

Using the `pageseg()` function's `black_colseps` parameter

If the `black_colseps()` is set to `True`, Kraken will assume that columns will be separated by black lines. This isn't the case here, but we have all the tools to go through and actually change the source image to have a black separator between columns.

First, update `show_boxes()` function by adding `black_colseps = True`

In [5]:

```
# The first step is that I want to update the show_boxes() function. I'm just go
ing to do a quick
# copy and paste from the above but add in the black_colseps=True parameter
def show_boxes(img):
    '''Modifies the passed image to show a series of bounding boxes on an image
 as run by kraken

    :param img: A PIL.Image object
    :return img: The modified PIL.Image object
    '''
    # Lets bring in our ImageDraw object
    from PIL import ImageDraw
    # And grab a drawing object to annotate that image
    drawing_object=ImageDraw.Draw(img)
    # We can create a set of boxes using pageseg.segment
    bounding_boxes=pageseg.segment(img.convert('1'), black_colseps=True)['boxes'
]
    # Now lets go through the list of bounding boxes
    for box in bounding_boxes:
        # An just draw a nice rectangle
        drawing_object.rectangle(box, fill = None, outline ='red')
    # And to make it easy, lets return the image object
    return img
```

Second, create an algorithm to detect a white column separator

In experimenting a bit I decided that I only wanted to add the separator if the space of was at least 25 pixels wide, which is roughly the width of a character, and six lines high. In determining these 2 properties, we need to determine:

- the width: which can be found by just assigning a variable to it at about 25 pixels.
- the height: which is harder since it depends on the height of the text. So write a routine to calculate the average height of a line.

In [6]:

```
def calculate_line_height(img):
    '''Calculates the average height of a line from a given image
    :param img: A PIL.Image object
    :return: The average line height in pixels
    '''
    # Lets get a list of bounding boxes for this image
    bounding_boxes=pageseg.segment(img.convert('1'))['boxes']
    # Each box is a tuple of (top, left, bottom, right) so the height is just to
p - bottom
    # So lets just calculate this over the set of all boxes
    height_accumulator=0
    for box in bounding_boxes:
        height_accumulator=height_accumulator+box[3]-box[1]
        # this is a bit tricky, remember that we start counting at the upper lef
t corner in PIL!
    # now lets just return the average height
    # lets change it to the nearest full pixel by making it an integer
    return int(height_accumulator/len(bounding_boxes))

# And lets test this with the image with have been using
line_height=calculate_line_height(Image.open("readonly/two_col.png"))
print(line_height)
```

```
31
```

Creating a Gap Box

Now after finding the line height, we want to scan through the image, look at each pixel to determine if there is **a block of whitespace**. How big of a block should we look for?

An appropriate box would be about 1 char width wide, and 6 lune heights tall, for instance.

In [7]:

```
# Lets create a new box called gap box that represents this area
gap_box=(0,0,char_width,line_height*6)
gap_box
```

Out[7]:

```
(0, 0, 25, 186)
```

Creating Function `gap_check()`

It seems we will want to have a function which, given a pixel in an image, can check to see if that pixel has whitespace to the right and below it. **Essentially, we want to test to see if the pixel is the upper left corner of something that looks like the gap_box.** If so, then we should insert a line to "break up" this box before sending to kraken

In [8]:

```
# Lets call this new function gap_check
def gap_check(img, location):
    '''Checks the img in a given (x,y) location to see if it fits the descriptio
n
    of a gap_box
    :param img: A PIL.Image file
    :param location: A tuple (x,y) which is a pixel location in that image
    :return: True if that fits the definition of a gap_box, otherwise False
    '''
    # Recall that we can get a pixel using the img.getpixel() function. It retur
ns this value
    # as a tuple of integers, one for each color channel. Our tools all work wit
h binarized
    # images (black and white), so we should just get one value. If the value is
0 it's a black
    # pixel, if it's white then the value should be 255
    #
    # We're going to assume that the image is in the correct mode already, e.g.
 it has been
    # binarized. The algorithm to check our bounding box is fairly easy: we have
a single location
    # which is our start and then we want to check all the pixels to the right o
f that location
    # up to gap_box[2]
    for x in range(location[0], location[0]+gap_box[2]):
        # the height is similar, so lets iterate a y variable to gap_box[3]
        for y in range(location[1], location[1]+gap_box[3]):
            # we want to check if the pixel is white, but only if we are still w
ithin the image
            if x < img.width and y < img.height:
                # if the pixel is white we don't do anything, if it's black, we
 just want to
                # finish and return False
                if img.getpixel((x,y)) != 255:
                    return False
    # If we have managed to walk all through the gap_box without finding any non
-white pixels
    # then we can return true -- this is a gap!
    return True
```

Drawing the column separator

Alright, we have a function to check for a gap, called gap_check. What should we do once we find a gap? For this, lets just draw a line in the middle of it.

This is important as this is where your pillow library comes in once again. Take a close look at the documentation of the function `draw_sep()` by the instructor.

In [9]:

```
# Lets create a new function
def draw_sep(img,location):
    '''Draws a line in img in the middle of the gap discovered at location. Note
that
    this doesn't draw the line in location, but draws it at the middle of a gap_
box
    starting at location.
    :param img: A PIL.Image file
    :param location: A tuple(x,y) which is a pixel location in the image
    '''
    # First lets bring in all of our drawing code
    from PIL import ImageDraw
    drawing_object=ImageDraw.Draw(img)
    # next, lets decide what the middle means in terms of coordinates in the ima
ge
    x1=location[0]+int(gap_box[2]/2)
    # and our x2 is just the same thing, since this is a one pixel vertical line
    x2=x1
    # our starting y coordinate is just the y coordinate which was passed in, th
e top of the box
    y1=location[1]
    # but we want our final y coordinate to be the bottom of the box
    y2=y1+gap_box[3]
    drawing_object.rectangle((x1,y1,x2,y2), fill = 'black', outline ='black')
    # and we don't have anything we need to return from this, because we modifie
d the image
```

Reprocessing the image to include black lines between columns

Now, lets try it all out. This is pretty easy, we can just iterate through each pixel in the image, check if there is a gap, then insert a line if there is.

Unfortunately, this has a time complexity of $O(mn)$,

for an image with m rows and n columns.

In [10]:

```
def process_image(img):
    '''Takes in an image of text and adds black vertical bars to break up column
s
    :param img: A PIL.Image file
    :return: A modified PIL.Image file
    '''
    # we'll start with a familiar iteration process
    for x in range(img.width):
        for y in range(img.height):
            # check if there is a gap at this point
            if (gap_check(img, (x,y))):
                # then update image to one which has a separator drawn on it
                draw_sep(img, (x,y))
    # and for good measure we'll return the image we modified
    return img

# Lets read in our test image and convert it through binarization
i=Image.open("readonly/two_col.png").convert("L")
i=process_image(i)
display(i)

# if you realised, this is O(n^2) compile time.
```

CALEB CHADWELL
Daily Staff Reporter

In a statement Tuesday, the Department of Public Safety and Security wrote that DPSS was not aware until Saturday afternoon of an assault on a University lecturer last week, referred in testimony before Ann Arbor City Council Monday.

Khita Whyatt, lecturer of dance in the University of Michigan's School of Music, Theatre & Dance, said in an interview after her testimony on the incident that she did not immediately call the police because she was so shocked, but her department chair contacted the DPSS. Two days after the incident, Whyatt said she was interviewed by two DPSS officers. During her testimony to Council, she called on the University to release a crime report about how she was knocked down and intimidated by unknown assailants. The event follows similar incidents where crime alerts had not been released. The University has released two crime alerts of hate crimes on campus over the past two weeks.

Whyatt wrote in an email sent Tuesday afternoon to recipients including University President Mark Schlissel as well as The Michigan Daily that she waited until Saturday morning to report the assault to police because she was disoriented and did not know where to reach out.

"I did wait until Saturday morning to get in touch to report the incident," Whyatt wrote. "I was in shock and still processing what to do prior to reaching out ... it was also obvious that there was no way that these boys were going to be caught. Not being a student, I did not know who to report to. That must seem obvious by the fact that I

Pass the Image Back Into Kraken

In [11]:

```
display(show_boxes(i))
```

CALEB CHADWELL
Daily Staff Reporter

In a statement Tuesday, the Department of Public Safety and Security wrote that DPSS was not aware until Saturday afternoon of an assault on a University lecturer last week, referred in testimony before Ann Arbor City Council Monday.

Khita Whyatt, lecturer of dance in the University of Michigan's School of Music, Theatre & Dance, said in an interview after her testimony on the incident that she did not immediately call the police because she was so shocked, but her department chair contacted the DPSS. Two days after the incident, Whyatt said she was interviewed by two DPSS officers. During her testimony to Council, she called on the University to release a crime report about how she was knocked down and intimidated by unknown assailants. The event follows similar incidents where crime alerts had not been released. The University has released two crime alerts of hate crimes on campus over the past two weeks.

Whyatt wrote in an email sent Tuesday afternoon to recipients including University President Mark Schlissel as well as The Michigan Daily that she waited until Saturday morning to report the assault to police because she was disoriented and did not know where to reach out.

"I did wait until Saturday morning to get in touch to report the incident," Whyatt wrote. "I was in shock and still processing what to do prior to reaching out ... it was also obvious that there was no way that these boys were going to be caught. Not being a student, I did not know who to report to. That must seem obvious by the fact that I

Looks like that is pretty accurate, and fixes the problem we faced. Feel free to experiment with different settings for the gap heights and width and share in the forums. You'll notice though method we created is really quite slow, which is a bit of a problem if we wanted to use this on larger text. But I wanted to show you how you can mix your own logic and work with libraries you're using. Just because Kraken didn't work perfectly, doesn't mean we can't build something more specific to our use case on top of it.

I want to end this lecture with a pause and to ask you to reflect on the code we've written here. We started this course with some pretty simple use of libraries, but now we're digging in deeper and solving problems ourselves with the help of these libraries. Before we go on to our last library, how well prepared do you think you are to take your python skills out into the wild?

03-02: Comparing Image Data Structures

OpenCV supports reading of images in most file formats, such as JPEG, PNG, and TIFF. Most image and video analysis requires converting images into grayscale first. This simplifies the image and reduces noise allowing for improved analysis.

Converting an Image To Grayscale using `cv.cvtColor()`

Let's write some code that reads an image of as person, Floyd Mayweather, and converts it into greyscale.

OpenCVs Documentation

Now, before we get to the result, lets talk about docs. Just like tesseract, opencv is an external package written in C++, and the docs for python are really poor. This is unfortunatly quite common when python is being used as a wrapper. Thankfully, the web docs for opencv are actually pretty good, so hit the website docs.opencv.org when you want to learn more about a particular function. In this case `cvtColor` converts from one color space to another, and we are convering our image to grayscale.

So now, we can change an image to grayscale in 3 ways:

- Using PIL's changing of color conventions
- Using PIL's Binarisation techniques
- Using OpenCVs _cv.cvtColor()__

In [12]:

```
# First we will import the open cv package cv2
import cv2 as cv
# We'll load the floyd.jpg image
img = cv.imread('readonly/floyd.jpg')
# And we'll convert it to grayscale using the cvtColor image
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

# Lets instpec this object that has been returned.
import inspect
inspect.getmro(type(gray))
```

Out[12]:

```
(numpy.ndarray, object)
```

Introducing the Numpy Library

We see that it is of type ndarray, which is a fundamental list type coming from the numerical python project. That's a bit surprising - up until this point we have been used to working with PIL.Image objects. **OpenCV, however, wants to represent an image as a two dimensional sequence of bytes**, and the ndarray, which stands for *n dimensional array*, is the ideal way to do this. Lets look at the array contents.

In [13]:

```
gray
```

Out[13]:

```
array([[ 40,  39,  39, ...,  77,  76,  75],
       [ 43,  42,  42, ...,  76,  75,  75],
       [ 39,  39,  39, ...,  76,  75,  74],
       ...,
       [ 21,  22,  24, ..., 219, 223, 209],
       [ 18,  20,  22, ..., 196, 206, 196],
       [ 16,  18,  20, ..., 168, 182, 176]], dtype=uint8)
```

The array is shown here as a list of lists, where the inner lists are filled with integers. The `dtype=uint8` definition indicates that each of the items in an array is an 8 bit unsigned integer, which is very common for black and white images.

So this is a pixel by pixel definition of the image.

Converting Image into PIL Object

PIL can take an array of data with a given color format and convert this into a PIL object. This is perfect for our situation, as the PIL color mode, "L" is just an array of luminance values in unsigned integers

In [14]:

```
from PIL import Image

image = Image.fromarray(gray, "L")
display(image)
```

Multidimensional Images: Introduction to Numpy & List of Lists

Lets talk a bit more about images for a moment. Numpy arrays are multidimensional. For instance, we can define an array in a single dimension:

In [ ]:

```
import numpy as np
single_dim = np.array([25, 50 , 25, 10, 10])
```

In an image, this is analagous to a **single row of 5 pixels each in grayscale**. But actually, all **imaging libraries tend to expect at least two dimensions**, a width and a height, and to show a matrix. So if we put the single_dim inside of another array, this would be a two dimensional array with element in the height direction, and five in the width direction

In [15]:

```
double_dim = np.array([single_dim])

double_dim
```

Out[15]:

```
array([[25, 50, 25, 10, 10]])
```

This should look pretty familiar, **it's a lot like a list of lists!** Lets see what this new two dimensional array looks like if we display it

In [16]:

```
display(Image.fromarray(double_dim, "L"))
```

-

Pretty unexciting - it's just a little line. Five pixels in a row to be exact, **of different levels of black.** We can check this by looking at the function `np.shape` . The shape attribute **returns a tuple** that shows the height of the image, by the width of the image.

In [17]:

```
# Pretty unexciting - it's just a little line. Five pixels in a row to be exact,
of different
# levels of black. The numpy library has a nice attribute called shape that allo
ws us to see how
# many dimensions big an array is. The shape attribute returns a tuple that show
s the height of
# the image, by the width of the image
double_dim.shape
```

Out[17]:

```
(1, 5)
```

In [18]:

```
# Lets take a look at the shape of our initial image which we loaded into the im
g variable
img.shape
```

Out[18]:

```
(416, 416, 3)
```

In [19]:

```
# This image has three dimensions! That's because it has a width, a height, and
 what's called
# a color depth. In this case, the color is represented as an array of three val
ues. Lets take a
# look at the color of the first pixel
first_pixel=img[0][0]
first_pixel
```

Out[19]:

```
array([33, 35, 53], dtype=uint8)
```

Here we see that the **color value is provided in full RGB** using an unsigned integer. This means that **each color can have one of 256 values**, and the total number of unique colors that can be represented by this data is 256 *256* 256 which is roughly 16 million colors.We call this 24 bit color, which is 8+8+8.

Changing Image into Grayscale via `reshape()`

One of the most common things to do with an ndarray is to reshape it -- to change the number of rows and columns that are represented so that we can do different kinds of operations. Here is our original two dimensional image

In addition, element-wise operations on the computer are done simultaneously, which means such operations are on average $O(1)" time.

In [20]:

```
print("Original image")
print(gray)
# If we wanted to represent that as a one dimensional image, we just call reshap
e
print("New image")
# And reshape takes the image as the first parameter, and a new shape as the sec
ond
image1d=np.reshape(gray,(1,gray.shape[0]*gray.shape[1]))
print(image1d)
```

```
Original image
[[ 40  39  39 ...  77  76  75]
 [ 43  42  42 ...  76  75  75]
 [ 39  39  39 ...  76  75  74]
 ...
 [ 21  22  24 ... 219 223 209]
 [ 18  20  22 ... 196 206 196]
 [ 16  18  20 ... 168 182 176]]
New image
[[ 40  39  39 ... 168 182 176]]
```

So, why are we talking about these nested arrays of bytes, we were supposed to be talking about OpenCV as a library. Well, I wanted to show you that often libraries working on the same kind of principles, in this case images stored as arrays of bytes, are not representing data in the same way in their APIs. But, by exploring a bit you can learn how the internal representation of data is stored, and build routines to convert between formats.

For instance, remember in the last lecture when we wanted to look for gaps in an image so

that we could draw lines to feed into kraken? Well, we use PIL to do this, using getpixel()

\ to look at individual pixels and see what the luminosity was, then ImageDraw.rectangle to

actually fill in a black bar separator. This was a nice high level API, and let us write routines to do the work we wanted without having to understand too much about how the images were being stored. But it was computationally very slow.

Creating GrayScale using Matrix Operation

Now, remember how slicing on a list works, if you have a list of number such as a=[0,1,2,3,4,5] then a[2:4] will return the sublist of numbers at position 2 through 4 inclusive - don't forget that lists start indexing at 0!

In [ ]:

```
# Instead, we could write the code to do this using matrix features within nump
y. Lets take
# a look.
import cv2 as cv
# We'll load the 2 column image
img = cv.imread('readonly/two_col.png')
# And we'll convert it to grayscale using the cvtColor image
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
```

In [21]:

```
# Now, remember how slicing on a list works, if you have a list of number such a
s
# a=[0,1,2,3,4,5] then a[2:4] will return the sublist of numbers at position 2 t
hrough 4
# inclusive - don't forget that lists start indexing at 0!
# If we have a two dimensional array, we can slice out a smaller piece of that u
sing the
# format a[2:4,1:3]. You can think of this as first slicing along the rows dimen
sion, then
# in the columns dimension. So in this example, that would be a matrix of rows
 2, and 3,
# and columns 1, and 2. Here's a look at our image.
gray[2:4,1:3]
```

Out[21]:

```
array([[39, 39],
       [37, 37]], dtype=uint8)
```

In [22]:

```
# So we see that it is all white. We can use this as a "window" and move it arou
nd our
# our big image.
#
# Finally, the ndarray library has lots of matrix functions which are generally
 very fast
# to run. One that we want to consider in this case is count_nonzero(), which ju
st returns
# the number of entries in the matrix which are not zero.
np.count_nonzero(gray[2:4,1:3])
```

Out[22]:

```
4
```

In [23]:

```
# Ok, the last benefit of going to this low level approach to images is that we
 can change
# pixels very fast as well. Previously we were drawing rectangles and setting a
 fill and line
# width. This is nice if you want to do something like change the color of the f
ill from the
# line, or draw complex shapes. But we really just want a line here. That's real
ly easy to
# do - we just want to change a number of luminosity values from 255 to 0.
#
# As an example, lets create a big white matrix
white_matrix=np.full((12,12),255,dtype=np.uint8)
display(Image.fromarray(white_matrix,"L"))
white_matrix
```

Out[23]:

```
array([[255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255],
       [255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255],
       [255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255],
       [255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255],
       [255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255],
       [255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255],
       [255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255],
       [255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255],
       [255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255],
       [255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255],
       [255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255],
       [255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 25
5]],
      dtype=uint8)
```

In [24]:

```
# looks pretty boring, it's just a giant white square we can't see. But if we wa
nt, we can
# easily color a column to be black
white_matrix[:,6]=np.full((1,12),0,dtype=np.uint8)
display(Image.fromarray(white_matrix,"L"))
white_matrix
```

|

Out[24]:

```
array([[255, 255, 255, 255, 255, 255,   0, 255, 255, 255, 255, 255],
       [255, 255, 255, 255, 255, 255,   0, 255, 255, 255, 255, 255],
       [255, 255, 255, 255, 255, 255,   0, 255, 255, 255, 255, 255],
       [255, 255, 255, 255, 255, 255,   0, 255, 255, 255, 255, 255],
       [255, 255, 255, 255, 255, 255,   0, 255, 255, 255, 255, 255],
       [255, 255, 255, 255, 255, 255,   0, 255, 255, 255, 255, 255],
       [255, 255, 255, 255, 255, 255,   0, 255, 255, 255, 255, 255],
       [255, 255, 255, 255, 255, 255,   0, 255, 255, 255, 255, 255],
       [255, 255, 255, 255, 255, 255,   0, 255, 255, 255, 255, 255],
       [255, 255, 255, 255, 255, 255,   0, 255, 255, 255, 255, 255],
       [255, 255, 255, 255, 255, 255,   0, 255, 255, 255, 255, 255],
       [255, 255, 255, 255, 255, 255,   0, 255, 255, 255, 255, 25
5]],
      dtype=uint8)
```

Speed Complexity Tradeoff

And that's exactly what we wanted to do. So, why do it this way, when it seems so much more low level? **Really, the answer is speed.** This paradigm of using matricies to store and manipulate bytes of data for images is much closer to how low level API and hardware developers think about storing files and bytes in memory.

How much faster is it? Well, that's up to you to discover; there's an optional assignment for this week to convert our old code over into this new format, to compare both the readability and speed of the two different approaches.

03-03: OpenCV as Face Detection

OpenCV comes with trained models for detecting faces, eyes, and smiles which we'll be using. You can train models for detecting other things - like hot dogs or flutes - and if you're interested in that I'd recommend you check out the Open CV docs on how to train a cascade classifier: https://docs.opencv.org/3.4/dc/d88/tutorial_traincascade.html However, in this lecture we just want to use the current classifiers and see if we can detect portions of an image which are interesting.

Load Training Data for OpenCV

After these classifiers are loaded, we want to try and detect a face, convert it to grayscale.

In [25]:

```
# First step is to load opencv and the XML-based classifiers
import cv2 as cv
face_cascade = cv.CascadeClassifier('readonly/haarcascade_frontalface_default.xml')
eye_cascade = cv.CascadeClassifier('readonly/haarcascade_eye.xml')
# Ok, with the classifiers loaded, we now want to try and detect a face. Lets pull in the
# picture we played with last time
img = cv.imread('readonly/floyd.jpg')
# And we'll convert it to grayscale using the cvtColor image
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
```

Using `.detectMultiScale()` classifier

In [26]:

```
# The next step is to use the face_cascade classifier. I'll let you go explore the docs if you
# would like to, but the norm is to use the detectMultiScale() function. This function returns
# a list of objects as rectangles. The first parameter is an ndarray of the image.
faces = face_cascade.detectMultiScale(gray)
# And lets just print those faces out to the screen
faces
```

Out[26]:

```
array([[158,  75, 176, 176]], dtype=int32)
```

In [27]:

```
faces.tolist()[0]
```

Out[27]:

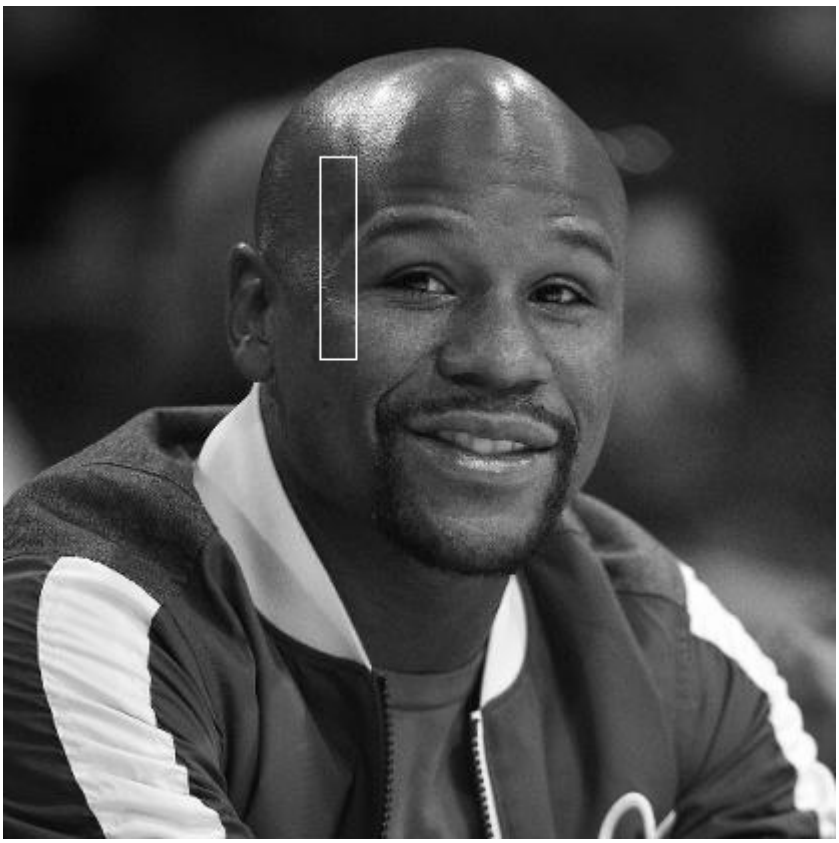
```
[158, 75, 176, 176]
```

Use PILLOW to Draw a Rectangle around the face

The resulting rectangles are in the format of **(x,y,w,h)** where x and y denote the upper left hand point for the image and the width and height represent the bounding box. We know how to handle this in PIL

In [28]:

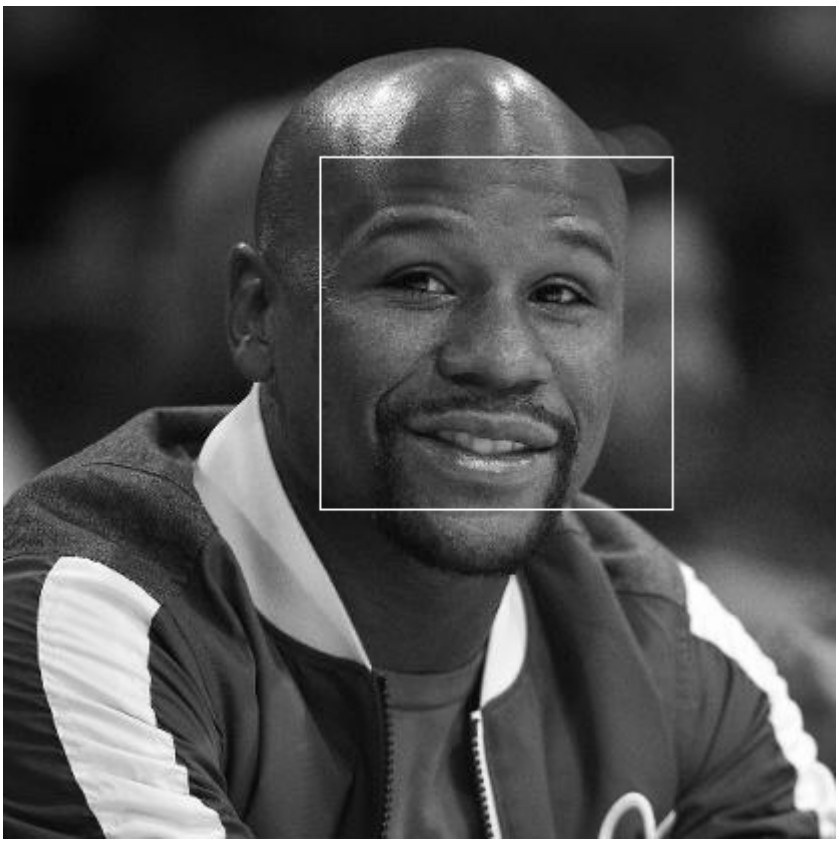
```
from PIL import Image

# Lets create a PIL image object
pil_img=Image.fromarray(gray,mode="L")

# Now lets bring in our drawing object
from PIL import ImageDraw
# And lets create our drawing context
drawing=ImageDraw.Draw(pil_img)

# Now lets pull the rectangle out of the faces object
rec=faces.tolist()[0]

# Now we just draw a rectangle around the bounds
drawing.rectangle(rec, outline="white")

# And display
display(pil_img)
```



So, not quite what we were looking for. What do you think went wrong? Well, a quick double check of the docs and it is apparent that OpenCV is return the coordinates as `(x,y,w,h)`, while PIL.ImageDraw is looking for `(x1,y1,x2,y2)`.

In [29]:

```
pil_img=Image.fromarray(gray,mode="L")
# Setup our drawing context
drawing=ImageDraw.Draw(pil_img)
# And draw the new box
drawing.rectangle((rec[0],rec[1],rec[0]+rec[2],rec[1]+rec[3]), outline="white")
# And display
display(pil_img)
```



Let's try this with a differnet set of data

In [30]:

```
img = cv.imread('readonly/msi_recruitment.gif')
# And lets take a look at that image
display(Image.fromarray(img))
```

```
---------------------------------------------------------------------
-------
AttributeError                            Traceback (most recent cal
l last)
<ipython-input-30-a6d9e3885cc1> in <module>
      4 img = cv.imread('readonly/msi_recruitment.gif')
      5 # And lets take a look at that image
----> 6 display(Image.fromarray(img))

/opt/conda/lib/python3.7/site-packages/PIL/Image.py in fromarray(ob
j, mode)
   2506     .. versionadded:: 1.1.6
   2507     """
-> 2508     arr = obj.__array_interface__
   2509     shape = arr['shape']
   2510     ndim = len(shape)

AttributeError: 'NoneType' object has no attribute '__array_interfac
e__'
```

Whoa, what's that error about? It looks like there is an error on a line deep within the PIL Image.py file, and it is trying to call an internal **private member** called _array*interface* on the img object, but this object is None

It turns out that the root of this error is that **OpenCV can't work with Gif images**. This is kind of a pain and unfortunate. But we know how to fix that right? One was is that we could just open this in PIL and then save it as a png, then open that in open cv.

In [31]:

```
# Lets use PIL to open our image
pil_img=Image.open('readonly/msi_recruitment.gif')
# now lets convert it to greyscale for opencv, and get the bytestream
open_cv_version=pil_img.convert("L")
# now lets just write that to a file
open_cv_version.save("msi_recruitment.png")
```

In [32]:

```
# Ok, now that the conversion of format is done, lets try reading this back into
opencv
cv_img=cv.imread('msi_recruitment.png')
# We don't need to color convert this, because we saved it as grayscale
# lets try and detect faces in that image
faces = face_cascade.detectMultiScale(cv_img)

# Now, we still have our PIL color version in a gif
pil_img=Image.open('readonly/msi_recruitment.gif')
# Set our drawing context
drawing=ImageDraw.Draw(pil_img)

# For each item in faces, lets surround it with a red box
for x,y,w,h in faces:
    # That might be new syntax for you! Recall that faces is a list of rectangle
s in (x,y,w,h)
    # format, that is, a list of lists. Instead of having to do an iteration and
then manually
    # pull out each item, we can use tuple unpacking to pull out individual item
s in the sublist
    # directly to variables. A really nice python feature
    #
    # Now we just need to draw our box
    drawing.rectangle((x,y,x+w,y+h), outline="white")
display(pil_img)
```

In [33]:

```
What happened here!? We see that we have detected faces, and that we have drawn
 boxes
around those faces on the image, but that the colors have gone all weird! This,
 it turns
out, has to do with color limitations for gif images. In short, a gif image has
a very
limited number of colors. This is called a color pallette after the pallette art
ists
use to mix paints. For gifs the pallette can only be 256 colors -- but they can
be *any*
56 colors. When a new color is introduced, is has to take the space of an old co
lor.
In this case, PIL adds white to the pallette but doesn't know which color to rep
lace and
thus messes up the image.

# Who knew there was so much to learn about image formats? We can see what mode
 the image
# is in with the .mode attribute
pil_img.mode
```

Out[33]:

```
'P'
```

In [34]:

```
# We can see a list of modes in the PILLOW documentation, and they correspond wi
th the
# color spaces we have been using. For the moment though, lets change back to RG
B, which
# represents color as a three byte tuple instead of in a pallette.
# Lets read in the image
pil_img=Image.open('readonly/msi_recruitment.gif')
# Lets convert it to RGB mode
pil_img = pil_img.convert("RGB")
# And lets print out the mode
pil_img.mode
```

Out[34]:

```
'RGB'
```

In [35]:

```
# Ok, now lets go back to drawing rectangles. Lets get our drawing object
drawing=ImageDraw.Draw(pil_img)
# And iterate through the faces sequence, tuple unpacking as we go
for x,y,w,h in faces:
    # And remember this is width and height so we have to add those appropriatel
y.
    drawing.rectangle((x,y,x+w,y+h), outline="white")
display(pil_img)
```

In [ ]:

```
# Awesome! We managed to detect a bunch of faces in that image. Looks like we ha
ve missed
# four faces. In the machine learning world we would call these false negatives
 - something
# which the machine thought was not a face (so a negative), but that it was inco
rrect on.
# Consequently, we would call the actual faces that were detected as true positi
ves -
# something that the machine thought was a face and it was correct on. This leav
es us with
# false positives - something the machine thought was a face but it wasn't. We s
ee there are
# two of these in the image, picking up shadow patterns or textures in shirts an
d matching
# them with the haarcascades. Finally, we have true negatives, or the set of all
possible
# rectangles the machine learning classifier could consider where it correctly i
ndicated that
# the result was not a face. In this case there are many many true negatives.
```

In [36]:

```
# There are a few ways we could try and improve this, and really, it requires a
 lot of
# experimentation to find good values for a given image. First, lets create a fu
nction
# which will plot rectanges for us over the image
def show_rects(faces):
    #Lets read in our gif and convert it
    pil_img=Image.open('readonly/msi_recruitment.gif').convert("RGB")
    # Set our drawing context
    drawing=ImageDraw.Draw(pil_img)
    # And plot all of the rectangles in faces
    for x,y,w,h in faces:
        drawing.rectangle((x,y,x+w,y+h), outline="white")
    #Finally lets display this
    display(pil_img)
```

In [38]:

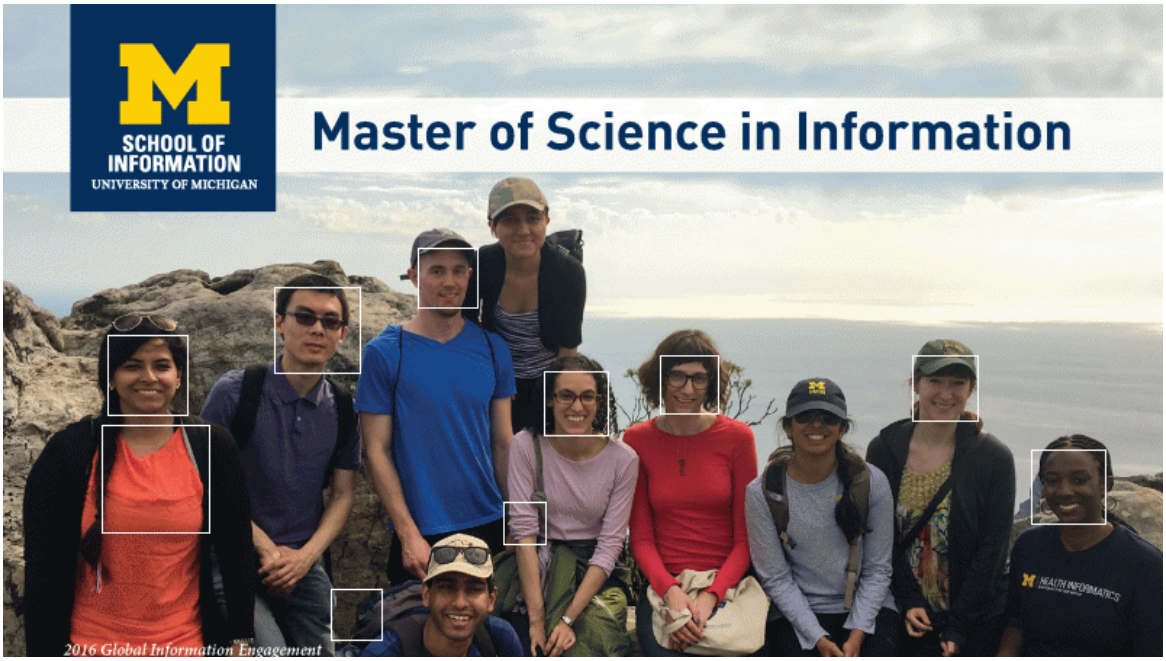
```
# Ok, first up, we could try and binarize this image. It turns out that opencv h
as a built in
# binarization function called threshold(). You simply pass in the image, the mi
dpoint, and
# the maximum value, as well as a flag which indicates whether the threshold sho
uld be
# binary or something else. Lets try this.
cv_img_bin=cv.threshold(img,120,255,cv.THRESH_BINARY)[1] # returns a list, we wa
nt the second value
# Now do the actual face detection
faces = face_cascade.detectMultiScale(cv_img_bin)
# Now lets see the results
show_rects(faces)
```

In [ ]:

```
# That's kind of interesting. Not better, but we do see that there is one false
 positive
# towards the bottom, where the classifier detected the sunglasses as eyes and t
he dark shadow
# line below as a mouth.
#
# If you're following in the notebook with this video, why don't you pause thing
s and try a
# few different parameters for the thresholding value?
```

In [39]:

```
# The detectMultiScale() function from OpenCV also has a couple of parameters. T
he first of
# these is the scale factor. The scale factor changes the size of rectangles whi
ch are
# considered against the model, that is, the haarcascades XML file. You can thin
k of it as if
# it were changing the size of the rectangles which are on the screen.
#
# Lets experiment with the scale factor. Usually it's a small value, lets try 1.
05
faces = face_cascade.detectMultiScale(cv_img,1.05)
# Show those results
show_rects(faces)
# Now lets also try 1.15
faces = face_cascade.detectMultiScale(cv_img,1.15)
# Show those results
show_rects(faces)
# Finally lets also try 1.25
faces = face_cascade.detectMultiScale(cv_img,1.25)
# Show those results
show_rects(faces)
```


SCHOOL OF
INFORMATION
UNIVERSITY OF MICHIGAN
Master of Science in Information
2016 Global Information Engagement


SCHOOL OF
INFORMATION
UNIVERSITY OF MICHIGAN
Master of Science in Information
2016 Global Information Engagement

In [ ]:

```
# We can see that as we change the scale factor we change the number of true and
# false positives and negatives. With the scale set to 1.05, we have 7 true posi
tives,
# which are correctly identified faces, and 3 false negatives, which are faces w
hich
# are there but not detected, and 3 false positives, where are non-faces which
# opencv thinks are faces. When we change this to 1.15 we lose the false positiv
es but
# also lose one of the true positives, the person to the right wearing a hat. An
d
# when we change this to 1.25 we lost more true positives as well.
#
# This is actually a really interesting phenomena in machine learning and artifi
cial
# intelligence. There is a trade off between not only how accurate a model is, b
ut how
# the inaccuracy actually happens. Which of these three models do you think is b
est?
```

In [40]:

```
# Well, the answer to that question is really, "it depends". It depends why you
 are trying
# to detect faces, and what you are going to do with them. If you think these is
sues
# are interesting, you might want to check out the Applied Data Science with Pyt
hon
# specialization Michigan offers on Coursera.
#
# Ok, beyond an opportunity to advertise, did you notice anything else that happ
ened when
# we changed the scale factor? It's subtle, but the speed at which the processin
g ran
# took longer at smaller scale factors. This is because more subimages are being
considered
# for these scales. This could also affect which method we might use.
#
# Jupyter has nice support for timing commands. You might have seen this before,
a line
# that starts with a percentage sign in jupyter is called a "magic function". Th
is isn't
# normal python - it's actually a shorthand way of writing a function which Jupy
ter
# has predefined. It looks a lot like the decorators we talked about in a previo
us
# lecture, but the magic functions were around long before decorators were part
 of the
# python language. One of the built-in magic functions in juptyer is called time
it, and this
# repeats a piece of python ten times (by default) and tells you the average spe
ed it
# took to complete.
#
# Lets time the speed of detectmultiscale when using a scale of 1.05
%timeit face_cascade.detectMultiScale(cv_img,1.05)
```

```
1.54 s ± 102 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [41]:

```
# Ok, now lets compare that to the speed at scale = 1.15
%timeit face_cascade.detectMultiScale(cv_img,1.15)
```

```
700 ms ± 49.4 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [ ]:

```
# You can see that this is a dramatic difference, roughly two and a half times s
lower
# when using the smaller scale!
#
# This wraps up our discussion of detecting faces in opencv. You'll see that, li
ke OCR, this
# is not a foolproof process. But we can build on the work others have done in m
achine learning
# and leverage powerful libraries to bring us closer to building a turn key pyth
on-based
# solution. Remember that the detection mechanism isn't specific to faces, tha
t's just the
# haarcascades training data we used. On the web you'll be able to find other tr
aining data
# to detect other objects, including eyes, animals, and so forth.
```

More Jupyter Widgets

In [42]:

```
# One of the nice things about using the Jupyter notebook systems is that there
 is a
# rich set of contributed plugins that seek to extend this system. In this lectu
re I
# want to introduce you to one such plugin, call ipy web rtc. Webrtc is a fairly
new
# protocol for real time communication on the web. Yup, I'm talking about chatti
ng.
# The widget brings this to the Jupyter notebook system. Lets take a look.
#
# First, lets import from this library two different classes which we'll use in
 a
# demo, one for the camera and one for images.
from ipywebrtc import CameraStream, ImageRecorder
# Then lets take a look at the camera stream object
help(CameraStream)
```

```
Help on class CameraStream in module ipywebrtc.webrtc:

class CameraStream(MediaStream)
 |  CameraStream(*args, **kwargs)
 |
 |  Represents a media source by a camera/webcam/microphone using
 |  getUserMedia. See
 |  https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/ge
tUserMedia
 |  for more detail.
 |  The constraints trait can be set to specify constraints for the
camera or
 |  microphone, which is described in the documentation of getUserMe
dia, such
 |  as in the link above,
 |  Two convenience methods are avaiable to easily get access to the
'front'
 |  and 'back' camera, when present
 |
 |  >>> CameraStream.facing_user(audio=False)
 |  >>> CameraStream.facing_environment(audio=False)
 |
 |  Method resolution order:
 |      CameraStream
 |      MediaStream
 |      ipywidgets.widgets.domwidget.DOMWidget
 |      ipywidgets.widgets.widget.Widget
 |      ipywidgets.widgets.widget.LoggingHasTraits
 |      traitlets.traitlets.HasTraits
 |      traitlets.traitlets.HasDescriptors
 |      builtins.object
 |
 |  Class methods defined here:
 |
 |  facing_environment(audio=True, **kwargs) from traitlets.traitlet
s.MetaHasTraits
 |      Convenience method to get the camera facing the environment
(often the back)
 |
 |      Parameters
 |      ----------
 |      audio: bool
 |          Capture audio or not
 |      kwargs:
 |          Extra keyword arguments passed to the `CameraStream`
 |
 |  facing_user(audio=True, **kwargs) from traitlets.traitlets.MetaH
asTraits
 |      Convenience method to get the camera facing the user (often
front)
 |
 |      Parameters
 |      ----------
 |      audio: bool
 |          Capture audio or not
 |      kwargs:
 |          Extra keyword arguments passed to the `CameraStream`
 |
 |  ----------------------------------------------------------------
------
 |  Data descriptors defined here:
```

```
 |
 |  constraints
 |      Constraints for the camera, see https://developer.mozilla.or
g/en-US/docs/Web/API/MediaDevices/getUserMedia for details.
 |
 |  ------------------------------------------------------------------
------
 |  Methods inherited from ipywidgets.widgets.domwidget.DOMWidget:
 |
 |  add_class(self, className)
 |      Adds a class to the top level element of the widget.
 |
 |      Doesn't add the class if it already exists.
 |
 |  remove_class(self, className)
 |      Removes a class from the top level element of the widget.
 |
 |      Doesn't remove the class if it doesn't exist.
 |
 |  ------------------------------------------------------------------
------
 |  Data descriptors inherited from ipywidgets.widgets.domwidget.DOM
Widget:
 |
 |  layout
 |      An instance trait which coerces a dict to an instance.
 |
 |      This lets the instance be specified as a dict, which is used
 |      to initialize the instance.
 |
 |      Also, we default to a trivial instance, even if args and kwa
rgs
 |      is not specified.
 |
 |  ------------------------------------------------------------------
------
 |  Methods inherited from ipywidgets.widgets.widget.Widget:
 |
 |  __del__(self)
 |      Object disposal
 |
 |  __init__(self, **kwargs)
 |      Public constructor
 |
 |  __repr__(self)
 |      Return repr(self).
 |
 |  add_traits(self, **traits)
 |      Dynamically add trait attributes to the Widget.
 |
 |  close(self)
 |      Close method.
 |
 |      Closes the underlying comm.
 |      When the comm is closed, all of the widget views are automat
ically
 |      removed from the front-end.
 |
 |  get_state(self, key=None, drop_defaults=False)
 |      Gets the widget state, or a piece of it.
 |
```

```
 |      Parameters
 |      ----------
 |      key : unicode or iterable (optional)
 |          A single property's name or iterable of property names t
o get.
 |
 |      Returns
 |      -------
 |      state : dict of states
 |      metadata : dict
 |          metadata for each field: {key: metadata}
 |
 |  get_view_spec(self)
 |
 |  hold_sync(self)
 |      Hold syncing any state until the outermost context manager e
xits
 |
 |  notify_change(self, change)
 |      Called when a property has changed.
 |
 |  on_displayed(self, callback, remove=False)
 |      (Un)Register a widget displayed callback.
 |
 |      Parameters
 |      ----------
 |      callback: method handler
 |          Must have a signature of::
 |
 |              callback(widget, **kwargs)
 |
 |          kwargs from display are passed through without modificat
ion.
 |      remove: bool
 |          True if the callback should be unregistered.
 |
 |  on_msg(self, callback, remove=False)
 |      (Un)Register a custom msg receive callback.
 |
 |      Parameters
 |      ----------
 |      callback: callable
 |          callback will be passed three arguments when a message a
rrives::
 |
 |              callback(widget, content, buffers)
 |
 |      remove: bool
 |          True if the callback should be unregistered.
 |
 |  open(self)
 |      Open a comm to the frontend if one isn't already open.
 |
 |  send(self, content, buffers=None)
 |      Sends a custom msg to the widget model in the front-end.
 |
 |      Parameters
 |      ----------
 |      content : dict
 |          Content of the message to send.
 |      buffers : list of binary buffers
```

```
 |          Binary buffers to send with message
 |
 |  send_state(self, key=None)
 |      Sends the widget state, or a piece of it, to the front-end,
if it exists.
 |
 |      Parameters
 |      ----------
 |      key : unicode, or iterable (optional)
 |          A single property's name or iterable of property names t
o sync with the front-end.
 |
 |  set_state(self, sync_data)
 |      Called when a state is received from the front-end.
 |
 |  ----------------------------------------------------------------
------
 |  Class methods inherited from ipywidgets.widgets.widget.Widget:
 |
 |  close_all() from traitlets.traitlets.MetaHasTraits
 |
 |  ----------------------------------------------------------------
------
 |  Static methods inherited from ipywidgets.widgets.widget.Widget:
 |
 |  get_manager_state(drop_defaults=False, widgets=None)
 |      Returns the full state for a widget manager for embedding
 |
 |      :param drop_defaults: when True, it will not include default
value
 |      :param widgets: list with widgets to include in the state (o
r all widgets when None)
 |      :return:
 |
 |  handle_comm_opened(comm, msg)
 |      Static method, called when a widget is constructed.
 |
 |  on_widget_constructed(callback)
 |      Registers a callback to be called when a widget is construct
ed.
 |
 |      The callback must have the following signature:
 |      callback(widget)
 |
 |  ----------------------------------------------------------------
------
 |  Data descriptors inherited from ipywidgets.widgets.widget.Widge
t:
 |
 |  comm
 |      A trait whose value must be an instance of a specified clas
s.
 |
 |      The value can also be an instance of a subclass of the speci
fied class.
 |
 |      Subclasses can declare default classes by overriding the kla
ss attribute
 |
 |  keys
 |      The traits which are synced.
```

```
 |
 |  model_id
 |      Gets the model id of this widget.
 |
 |      If a Comm doesn't exist yet, a Comm will be created automagi
cally.
 |
 |  ----------------------------------------------------------------
------
 |  Data and other attributes inherited from ipywidgets.widgets.widg
et.Widget:
 |
 |  widget_types = <ipywidgets.widgets.widget.WidgetRegistry object>
 |
 |  widgets = {}
 |
 |  ----------------------------------------------------------------
------
 |  Data descriptors inherited from ipywidgets.widgets.widget.Loggin
gHasTraits:
 |
 |  log
 |      A trait whose value must be an instance of a specified clas
s.
 |
 |      The value can also be an instance of a subclass of the speci
fied class.
 |
 |      Subclasses can declare default classes by overriding the kla
ss attribute
 |
 |  ----------------------------------------------------------------
------
 |  Methods inherited from traitlets.traitlets.HasTraits:
 |
 |  __getstate__(self)
 |
 |  __setstate__(self, state)
 |
 |  has_trait(self, name)
 |      Returns True if the object has a trait with the specified na
me.
 |
 |  hold_trait_notifications(self)
 |      Context manager for bundling trait change notifications and
cross
 |      validation.
 |
 |      Use this when doing multiple trait assignments (init, confi
g), to avoid
 |      race conditions in trait notifiers requesting other trait va
lues.
 |      All trait notifications will fire after all values have been
assigned.
 |
 |  observe(self, handler, names=traitlets.All, type='change')
 |      Setup a handler to be called when a trait changes.
 |
 |      This is used to setup dynamic notifications of trait change
s.
 |
```

```
 |      Parameters
 |      ----------
 |      handler : callable
 |          A callable that is called when a trait changes. Its
 |          signature should be ``handler(change)``, where ``change`
` is a
 |          dictionary. The change dictionary at least holds a 'typ
e' key.
 |          * ``type``: the type of notification.
 |          Other keys may be passed depending on the value of 'typ
e'. In the
 |          case where type is 'change', we also have the following
keys:
 |          * ``owner`` : the HasTraits instance
 |          * ``old`` : the old value of the modified trait attribut
e
 |          * ``new`` : the new value of the modified trait attribut
e
 |          * ``name`` : the name of the modified trait attribute.
 |      names : list, str, All
 |          If names is All, the handler will apply to all traits.
If a list
 |          of str, handler will apply to all names in the list.  If
a
 |          str, the handler will apply just to that name.
 |      type : str, All (default: 'change')
 |          The type of notification to filter by. If equal to All,
then all
 |          notifications are passed to the observe handler.
 |
 |  on_trait_change(self, handler=None, name=None, remove=False)
 |      DEPRECATED: Setup a handler to be called when a trait change
s.
 |
 |      This is used to setup dynamic notifications of trait change
s.
 |
 |      Static handlers can be created by creating methods on a HasT
raits
 |      subclass with the naming convention '_[traitname]_changed'.
Thus,
 |      to create static handler for the trait 'a', create the metho
d
 |      _a_changed(self, name, old, new) (fewer arguments can be use
d, see
 |      below).
 |
 |      If `remove` is True and `handler` is not specified, all chan
ge
 |      handlers for the specified name are uninstalled.
 |
 |      Parameters
 |      ----------
 |      handler : callable, None
 |          A callable that is called when a trait changes.  Its
 |          signature can be handler(), handler(name), handler(name,
new),
 |          handler(name, old, new), or handler(name, old, new, sel
f).
 |      name : list, str, None
 |          If None, the handler will apply to all traits.  If a lis
```

```
t
 |          of str, handler will apply to all names in the list.  If
a
 |          str, the handler will apply just to that name.
 |      remove : bool
 |          If False (the default), then install the handler.  If Tr
ue
 |          then unintall it.
 |
 |  set_trait(self, name, value)
 |      Forcibly sets trait attribute, including read-only attribute
s.
 |
 |  setup_instance(self, *args, **kwargs)
 |      This is called **before** self.__init__ is called.
 |
 |  trait_metadata(self, traitname, key, default=None)
 |      Get metadata values for trait by key.
 |
 |  trait_names(self, **metadata)
 |      Get a list of all the names of this class' traits.
 |
 |  traits(self, **metadata)
 |      Get a ``dict`` of all the traits of this class.  The diction
ary
 |      is keyed on the name and the values are the TraitType object
s.
 |
 |      The TraitTypes returned don't know anything about the values
 |      that the various HasTrait's instances are holding.
 |
 |      The metadata kwargs allow functions to be passed in which
 |      filter traits based on metadata values.  The functions shoul
d
 |      take a single value as an argument and return a boolean.  If
 |      any function returns False, then the trait is not included i
n
 |      the output.  If a metadata key doesn't exist, None will be p
assed
 |      to the function.
 |
 |  unobserve(self, handler, names=traitlets.All, type='change')
 |      Remove a trait change handler.
 |
 |      This is used to unregister handlers to trait change notifica
tions.
 |
 |      Parameters
 |      ----------
 |      handler : callable
 |          The callable called when a trait attribute changes.
 |      names : list, str, All (default: All)
 |          The names of the traits for which the specified handler
should be
 |          uninstalled. If names is All, the specified handler is u
ninstalled
 |          from the list of notifiers corresponding to all changes.
 |      type : str or All (default: 'change')
 |          The type of notification to filter by. If All, the speci
fied handler
 |          is uninstalled from the list of notifiers corresponding
```

```
to all types.
 |
 |  unobserve_all(self, name=traitlets.All)
 |      Remove trait change handlers of any type for the specified n
ame.
 |      If name is not specified, removes all trait notifiers.
 |
 |  ---------------------------------------------------------------
------
 |  Class methods inherited from traitlets.traitlets.HasTraits:
 |
 |  class_own_trait_events(name) from traitlets.traitlets.MetaHasTra
its
 |      Get a dict of all event handlers defined on this class, not
a parent.
 |
 |      Works like ``event_handlers``, except for excluding traits f
rom parents.
 |
 |  class_own_traits(**metadata) from traitlets.traitlets.MetaHasTra
its
 |      Get a dict of all the traitlets defined on this class, not a
parent.
 |
 |      Works like `class_traits`, except for excluding traits from
parents.
 |
 |  class_trait_names(**metadata) from traitlets.traitlets.MetaHasTr
aits
 |      Get a list of all the names of this class' traits.
 |
 |      This method is just like the :meth:`trait_names` method,
 |      but is unbound.
 |
 |  class_traits(**metadata) from traitlets.traitlets.MetaHasTraits
 |      Get a ``dict`` of all the traits of this class.  The diction
ary
 |      is keyed on the name and the values are the TraitType object
s.
 |
 |      This method is just like the :meth:`traits` method, but is u
nbound.
 |
 |      The TraitTypes returned don't know anything about the values
 |      that the various HasTrait's instances are holding.
 |
 |      The metadata kwargs allow functions to be passed in which
 |      filter traits based on metadata values.  The functions shoul
d
 |      take a single value as an argument and return a boolean.  If
 |      any function returns False, then the trait is not included i
n
 |      the output.  If a metadata key doesn't exist, None will be p
assed
 |      to the function.
 |
 |  trait_events(name=None) from traitlets.traitlets.MetaHasTraits
 |      Get a ``dict`` of all the event handlers of this class.
 |
 |      Parameters
 |      ----------
```

```
 |      name: str (default: None)
 |          The name of a trait of this class. If name is ``None`` t
hen all
 |          the event handlers of this class will be returned instea
d.
 |
 |      Returns
 |      -------
 |      The event handlers associated with a trait name, or all even
t handlers.
 |
 |  ----------------------------------------------------------------
------
 |  Data descriptors inherited from traitlets.traitlets.HasTraits:
 |
 |  cross_validation_lock
 |      A contextmanager for running a block with our cross validati
on lock set
 |      to True.
 |
 |      At the end of the block, the lock's value is restored to its
value
 |      prior to entering the block.
 |
 |  ----------------------------------------------------------------
------
 |  Static methods inherited from traitlets.traitlets.HasDescriptor
s:
 |
 |  __new__(cls, *args, **kwargs)
 |      Create and return a new object.  See help(type) for accurate
signature.
 |
 |  ----------------------------------------------------------------
------
 |  Data descriptors inherited from traitlets.traitlets.HasDescripto
rs:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

In [43]:

```
# We see from the docs that it's east to get a camera facing the user, and we ca
n have
# the audio on or off. We don't need audio for this demo, so lets create a new c
amera
# instance
camera = CameraStream.facing_user(audio=False)
# The next object we want to look at is the ImageRecorder
help(ImageRecorder)
```

```
Help on class ImageRecorder in module ipywebrtc.webrtc:

class ImageRecorder(Recorder)
 |  ImageRecorder(*args, **kwargs)
 |
 |  Creates a recorder which allows to grab an Image from a MediaStr
eam widget.
 |
 |  Method resolution order:
 |      ImageRecorder
 |      Recorder
 |      ipywidgets.widgets.domwidget.DOMWidget
 |      ipywidgets.widgets.widget.Widget
 |      ipywidgets.widgets.widget.LoggingHasTraits
 |      traitlets.traitlets.HasTraits
 |      traitlets.traitlets.HasDescriptors
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, format='png', filename='record', recording=False,
autosave=False, **kwargs)
 |      Public constructor
 |
 |  save(self, filename=None)
 |      Save the image to a file, if no filename is given it is base
d on the filename trait and the format.
 |
 |      >>> recorder = ImageRecorder(filename='test', format='png')
 |      >>> ...
 |      >>> recorder.save()  # will save to test.png
 |      >>> recorder.save('foo')  # will save to foo.png
 |      >>> recorder.save('foo.dat')  # will save to foo.dat
 |
 |  ----------------------------------------------------------------
------
 |  Data descriptors defined here:
 |
 |  format
 |      The format of the image.
 |
 |  image
 |      A trait whose value must be an instance of a specified clas
s.
 |
 |      The value can also be an instance of a subclass of the speci
fied class.
 |
 |      Subclasses can declare default classes by overriding the kla
ss attribute
 |
 |  ----------------------------------------------------------------
------
 |  Methods inherited from Recorder:
 |
 |  download(self)
 |      Download the recording (usually a popup appears in the brows
er)
 |
 |  ----------------------------------------------------------------
------
```

```
 |  Data descriptors inherited from Recorder:
 |
 |  autosave
 |      If true, will save the data to a file once the recording is
finished (based on filename and format)
 |
 |  filename
 |      The filename used for downloading or auto saving.
 |
 |  recording
 |      (boolean) Indicator and controller of the recorder state, i.
e. putting the value to True will start recording.
 |
 |  stream
 |      An instance of :class:`MediaStream` that is the source for r
ecording.
 |
 |  ----------------------------------------------------------------
------
 |  Methods inherited from ipywidgets.widgets.domwidget.DOMWidget:
 |
 |  add_class(self, className)
 |      Adds a class to the top level element of the widget.
 |
 |      Doesn't add the class if it already exists.
 |
 |  remove_class(self, className)
 |      Removes a class from the top level element of the widget.
 |
 |      Doesn't remove the class if it doesn't exist.
 |
 |  ----------------------------------------------------------------
------
 |  Data descriptors inherited from ipywidgets.widgets.domwidget.DOM
Widget:
 |
 |  layout
 |      An instance trait which coerces a dict to an instance.
 |
 |      This lets the instance be specified as a dict, which is used
 |      to initialize the instance.
 |
 |      Also, we default to a trivial instance, even if args and kwa
rgs
 |      is not specified.
 |
 |  ----------------------------------------------------------------
------
 |  Methods inherited from ipywidgets.widgets.widget.Widget:
 |
 |  __del__(self)
 |      Object disposal
 |
 |  __repr__(self)
 |      Return repr(self).
 |
 |  add_traits(self, **traits)
 |      Dynamically add trait attributes to the Widget.
 |
 |  close(self)
 |      Close method.
```

```
 |
 |      Closes the underlying comm.
 |      When the comm is closed, all of the widget views are automat
ically
 |      removed from the front-end.
 |
 |  get_state(self, key=None, drop_defaults=False)
 |      Gets the widget state, or a piece of it.
 |
 |      Parameters
 |      ----------
 |      key : unicode or iterable (optional)
 |          A single property's name or iterable of property names t
o get.
 |
 |      Returns
 |      -------
 |      state : dict of states
 |      metadata : dict
 |          metadata for each field: {key: metadata}
 |
 |  get_view_spec(self)
 |
 |  hold_sync(self)
 |      Hold syncing any state until the outermost context manager e
xits
 |
 |  notify_change(self, change)
 |      Called when a property has changed.
 |
 |  on_displayed(self, callback, remove=False)
 |      (Un)Register a widget displayed callback.
 |
 |      Parameters
 |      ----------
 |      callback: method handler
 |          Must have a signature of::
 |
 |              callback(widget, **kwargs)
 |
 |          kwargs from display are passed through without modificat
ion.
 |      remove: bool
 |          True if the callback should be unregistered.
 |
 |  on_msg(self, callback, remove=False)
 |      (Un)Register a custom msg receive callback.
 |
 |      Parameters
 |      ----------
 |      callback: callable
 |          callback will be passed three arguments when a message a
rrives::
 |
 |              callback(widget, content, buffers)
 |
 |      remove: bool
 |          True if the callback should be unregistered.
 |
 |  open(self)
 |      Open a comm to the frontend if one isn't already open.
```

```
 |
 |  send(self, content, buffers=None)
 |      Sends a custom msg to the widget model in the front-end.
 |
 |      Parameters
 |      ----------
 |      content : dict
 |          Content of the message to send.
 |      buffers : list of binary buffers
 |          Binary buffers to send with message
 |
 |  send_state(self, key=None)
 |      Sends the widget state, or a piece of it, to the front-end,
if it exists.
 |
 |      Parameters
 |      ----------
 |      key : unicode, or iterable (optional)
 |          A single property's name or iterable of property names t
o sync with the front-end.
 |
 |  set_state(self, sync_data)
 |      Called when a state is received from the front-end.
 |
 |  ----------------------------------------------------------------
------
 |  Class methods inherited from ipywidgets.widgets.widget.Widget:
 |
 |  close_all() from traitlets.traitlets.MetaHasTraits
 |
 |  ----------------------------------------------------------------
------
 |  Static methods inherited from ipywidgets.widgets.widget.Widget:
 |
 |  get_manager_state(drop_defaults=False, widgets=None)
 |      Returns the full state for a widget manager for embedding
 |
 |      :param drop_defaults: when True, it will not include default
value
 |      :param widgets: list with widgets to include in the state (o
r all widgets when None)
 |      :return:
 |
 |  handle_comm_opened(comm, msg)
 |      Static method, called when a widget is constructed.
 |
 |  on_widget_constructed(callback)
 |      Registers a callback to be called when a widget is construct
ed.
 |
 |      The callback must have the following signature:
 |      callback(widget)
 |
 |  ----------------------------------------------------------------
------
 |  Data descriptors inherited from ipywidgets.widgets.widget.Widge
t:
 |
 |  comm
 |      A trait whose value must be an instance of a specified clas
s.
```

```
 |
 |      The value can also be an instance of a subclass of the speci
fied class.
 |
 |      Subclasses can declare default classes by overriding the kla
ss attribute
 |
 |  keys
 |      The traits which are synced.
 |
 |  model_id
 |      Gets the model id of this widget.
 |
 |      If a Comm doesn't exist yet, a Comm will be created automagi
cally.
 |
 |  ----------------------------------------------------------------
------
 |  Data and other attributes inherited from ipywidgets.widgets.widg
et.Widget:
 |
 |  widget_types = <ipywidgets.widgets.widget.WidgetRegistry object>
 |
 |  widgets = {'cc401cd050ea4415bc5e87f7e3a0e0dd': CameraStream(cons
traint...
 |
 |  ----------------------------------------------------------------
------
 |  Data descriptors inherited from ipywidgets.widgets.widget.Loggin
gHasTraits:
 |
 |  log
 |      A trait whose value must be an instance of a specified clas
s.
 |
 |      The value can also be an instance of a subclass of the speci
fied class.
 |
 |      Subclasses can declare default classes by overriding the kla
ss attribute
 |
 |  ----------------------------------------------------------------
------
 |  Methods inherited from traitlets.traitlets.HasTraits:
 |
 |  __getstate__(self)
 |
 |  __setstate__(self, state)
 |
 |  has_trait(self, name)
 |      Returns True if the object has a trait with the specified na
me.
 |
 |  hold_trait_notifications(self)
 |      Context manager for bundling trait change notifications and
cross
 |      validation.
 |
 |      Use this when doing multiple trait assignments (init, confi
g), to avoid
 |      race conditions in trait notifiers requesting other trait va
```

```
lues.
 |      All trait notifications will fire after all values have been
assigned.
 |
 |  observe(self, handler, names=traitlets.All, type='change')
 |      Setup a handler to be called when a trait changes.
 |
 |      This is used to setup dynamic notifications of trait change
s.
 |
 |      Parameters
 |      ----------
 |      handler : callable
 |          A callable that is called when a trait changes. Its
 |          signature should be ``handler(change)``, where ``change`
` is a
 |          dictionary. The change dictionary at least holds a 'typ
e' key.
 |          * ``type``: the type of notification.
 |          Other keys may be passed depending on the value of 'typ
e'. In the
 |          case where type is 'change', we also have the following
keys:
 |          * ``owner`` : the HasTraits instance
 |          * ``old`` : the old value of the modified trait attribut
e
 |          * ``new`` : the new value of the modified trait attribut
e
 |          * ``name`` : the name of the modified trait attribute.
 |      names : list, str, All
 |          If names is All, the handler will apply to all traits.
If a list
 |          of str, handler will apply to all names in the list.  If
a
 |          str, the handler will apply just to that name.
 |      type : str, All (default: 'change')
 |          The type of notification to filter by. If equal to All,
then all
 |          notifications are passed to the observe handler.
 |
 |  on_trait_change(self, handler=None, name=None, remove=False)
 |      DEPRECATED: Setup a handler to be called when a trait change
s.
 |
 |      This is used to setup dynamic notifications of trait change
s.
 |
 |      Static handlers can be created by creating methods on a HasT
raits
 |      subclass with the naming convention '_[traitname]_changed'.
Thus,
 |      to create static handler for the trait 'a', create the metho
d
 |      _a_changed(self, name, old, new) (fewer arguments can be use
d, see
 |      below).
 |
 |      If `remove` is True and `handler` is not specified, all chan
ge
 |      handlers for the specified name are uninstalled.
 |
```

```
 |      Parameters
 |      ----------
 |      handler : callable, None
 |          A callable that is called when a trait changes.  Its
 |          signature can be handler(), handler(name), handler(name,
new),
 |          handler(name, old, new), or handler(name, old, new, sel
f).
 |      name : list, str, None
 |          If None, the handler will apply to all traits.  If a lis
t
 |          of str, handler will apply to all names in the list.  If
a
 |          str, the handler will apply just to that name.
 |      remove : bool
 |          If False (the default), then install the handler.  If Tr
ue
 |          then unintall it.
 |
 |  set_trait(self, name, value)
 |      Forcibly sets trait attribute, including read-only attribute
s.
 |
 |  setup_instance(self, *args, **kwargs)
 |      This is called **before** self.__init__ is called.
 |
 |  trait_metadata(self, traitname, key, default=None)
 |      Get metadata values for trait by key.
 |
 |  trait_names(self, **metadata)
 |      Get a list of all the names of this class' traits.
 |
 |  traits(self, **metadata)
 |      Get a ``dict`` of all the traits of this class.  The diction
ary
 |      is keyed on the name and the values are the TraitType object
s.
 |
 |      The TraitTypes returned don't know anything about the values
 |      that the various HasTrait's instances are holding.
 |
 |      The metadata kwargs allow functions to be passed in which
 |      filter traits based on metadata values.  The functions shoul
d
 |      take a single value as an argument and return a boolean.  If
 |      any function returns False, then the trait is not included i
n
 |      the output.  If a metadata key doesn't exist, None will be p
assed
 |      to the function.
 |
 |  unobserve(self, handler, names=traitlets.All, type='change')
 |      Remove a trait change handler.
 |
 |      This is used to unregister handlers to trait change notifica
tions.
 |
 |      Parameters
 |      ----------
 |      handler : callable
 |          The callable called when a trait attribute changes.
```

```
 |      names : list, str, All (default: All)
 |          The names of the traits for which the specified handler
should be
 |          uninstalled. If names is All, the specified handler is u
ninstalled
 |          from the list of notifiers corresponding to all changes.
 |      type : str or All (default: 'change')
 |          The type of notification to filter by. If All, the speci
fied handler
 |          is uninstalled from the list of notifiers corresponding
to all types.
 |
 |  unobserve_all(self, name=traitlets.All)
 |      Remove trait change handlers of any type for the specified n
ame.
 |      If name is not specified, removes all trait notifiers.
 |
 |  ----------------------------------------------------------------
------
 |  Class methods inherited from traitlets.traitlets.HasTraits:
 |
 |  class_own_trait_events(name) from traitlets.traitlets.MetaHasTra
its
 |      Get a dict of all event handlers defined on this class, not
a parent.
 |
 |      Works like ``event_handlers``, except for excluding traits f
rom parents.
 |
 |  class_own_traits(**metadata) from traitlets.traitlets.MetaHasTra
its
 |      Get a dict of all the traitlets defined on this class, not a
parent.
 |
 |      Works like `class_traits`, except for excluding traits from
parents.
 |
 |  class_trait_names(**metadata) from traitlets.traitlets.MetaHasTr
aits
 |      Get a list of all the names of this class' traits.
 |
 |      This method is just like the :meth:`trait_names` method,
 |      but is unbound.
 |
 |  class_traits(**metadata) from traitlets.traitlets.MetaHasTraits
 |      Get a ``dict`` of all the traits of this class.  The diction
ary
 |      is keyed on the name and the values are the TraitType object
s.
 |
 |      This method is just like the :meth:`traits` method, but is u
nbound.
 |
 |      The TraitTypes returned don't know anything about the values
 |      that the various HasTrait's instances are holding.
 |
 |      The metadata kwargs allow functions to be passed in which
 |      filter traits based on metadata values.  The functions shoul
d
 |      take a single value as an argument and return a boolean.  If
 |      any function returns False, then the trait is not included i
```

```
n
 |      the output.  If a metadata key doesn't exist, None will be p
assed
 |      to the function.
 |
 |  trait_events(name=None) from traitlets.traitlets.MetaHasTraits
 |      Get a ``dict`` of all the event handlers of this class.
 |
 |      Parameters
 |      ----------
 |      name: str (default: None)
 |          The name of a trait of this class. If name is ``None`` t
hen all
 |          the event handlers of this class will be returned instea
d.
 |
 |      Returns
 |      -------
 |      The event handlers associated with a trait name, or all even
t handlers.
 |
 |  ----------------------------------------------------------------
------
 |  Data descriptors inherited from traitlets.traitlets.HasTraits:
 |
 |  cross_validation_lock
 |      A contextmanager for running a block with our cross validati
on lock set
 |      to True.
 |
 |      At the end of the block, the lock's value is restored to its
value
 |      prior to entering the block.
 |
 |  ----------------------------------------------------------------
------
 |  Static methods inherited from traitlets.traitlets.HasDescriptor
s:
 |
 |  __new__(cls, *args, **kwargs)
 |      Create and return a new object.  See help(type) for accurate
signature.
 |
 |  ----------------------------------------------------------------
------
 |  Data descriptors inherited from traitlets.traitlets.HasDescripto
rs:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

In [46]:

```
# The image recorder lets us actually grab images from the camera stream. There
 are features
# for downloading and using the image as well. We see that the default format is
a png file.
# Lets hook up the ImageRecorder to our stream
image_recorder = ImageRecorder(stream=camera)
# Now, the docs are a little unclear how to use this within Jupyter, but if we c
all the
# download() function it will actually store the results of the camera which is
 hooked up
# in image_recorder.image. Lets try it out
# First, lets tell the recorder to start capturing data
image_recorder.recording=True
# Now lets download the image
image_recorder.download()
# Then lets inspect the type of the image
type(image_recorder.image)
```

Out[46]:

```
ipywidgets.widgets.widget_media.Image
```

In [47]:

```
# Ok, the object that it stores is an ipywidgets.widgets.widget_media.Image. How
do we do
# something useful with this? Well, an inspection of the object shows that there
is a handy
# value field which actually holds the bytes behind the image. And we know how t
o display
# those.
# Lets import PIL Image
import PIL.Image
# And lets import io
import io
# And now lets create a PIL image from the bytes
img = PIL.Image.open(io.BytesIO(image_recorder.image.value))
# And render it to the screen
display(img)
```

```
---------------------------------------------------------------------
-------
OSError                                   Traceback (most recent cal
l last)
<ipython-input-47-ba17cbdcfa82> in <module>
      8 import io
      9 # And now lets create a PIL image from the bytes
---> 10 img = PIL.Image.open(io.BytesIO(image_recorder.image.value))
     11 # And render it to the screen
     12 display(img)

/opt/conda/lib/python3.7/site-packages/PIL/Image.py in open(fp, mod
e)
   2685         warnings.warn(message)
   2686     raise IOError("cannot identify image file %r"
-> 2687                   % (filename if filename else fp))
   2688
   2689 #

OSError: cannot identify image file <_io.BytesIO object at 0x7f132b8
1daf0>
```

In [ ]:

```
# Great, you see a picture! Hopefully you are following along in one of the note
books
# and have been able to try this out for yourself!
#
# What can you do with this? This is a great way to get started with a bit of co
mputer vision.
# You already know how to identify a face in the webcam picture, or try and capt
ure text
# from within the picture. With OpenCV there are any number of other things you
 can do, simply
# with a webcam, the Jupyter notebooks, and python!
```