module_3

May 13, 2023

0.1 Release the Kraken!

```
In [ ]: # The next library we're going to look at is called Kraken, which was developed by Uni
        # PSL in Paris. It's actually based on a slightly older code base, OCRopus. You can se
        # flexible open-source licenses allow new ideas to grow by building upon older ideas.
        # this case, I fully support the idea that the Kraken - a mythical massive sea creatur
        # natural progression of an octopus!
        #
        # What we are going to use Kraken for is to detect lines of text as bounding boxes in
        # image. The biggest limitation of tesseract is the lack of a layout engine inside of
        # expects to be using fairly clean text, and gets confused if we don't crop out other
        # 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
        # Documentation and Installation Notes can be found at: https://pypi.org/project/krake
```

```
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, 'alpha', 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, 'alpha', 0)...

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

```
In [2]: # There isn't much of a discussion here, but there are a number of sub-modules that lo
        # interesting. I spend a bit of time on their website, and I think the pageseg module,
        # handles all of the page segmentation, is the one we want to use. Lets look at it
        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.mlm
        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 with a list of
                           coordinates (x0, y0, x1, y1) of a text line 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', 'boxes':
            [[(script, (x1, y1, x2, y2)),...]]}: A dictionary containing the text
            direction and a list of lists of reading order sorted bounding boxes
            under the key 'boxes' with each list containing the script segmentation
            of a single line. Script is a ISO15924 4 character identifier.
```

```
        Raises:
            KrakenInvalidModelException if no clstm module is available.

    segment(im, text_direction='horizontal-lr', scale=None, maxcolseps=2, black_colseps=False,
        Segments a page into text lines.

        Segments a page into text lines and returns the absolute coordinates 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 separators
            black_colseps (bool): Whether column separators are assumed 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 boxes. If int the
                                same padding is used both left and right. 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 segmentation
                              purposes. Disables column detection.

        Returns:
            {'text_direction': '$dir', 'boxes': [(x1, y1, x2, y2),...]}: A
            dictionary containing the text direction and a list of reading 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
```

In [3]:
```
# So it looks like there are a few different functions we can call, and the segment
# function looks particularly appropriate. I love how expressive this library is on th
# documentation front -- I can see immediately that we are working with PIL.Image file
# and the author has even indicated that we need to pass in either a binarized (e.g. '
```

```
# 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.
#
# Lets try this on the image of text. I have a simple bit of text in a file called
# two_col.png which is from a newspaper on campus here
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 kraken
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, 5
```

```
In [4]: # Ok, pretty simple two column text and then a list of lists which are the bounding bo
        # lines of that text. Lets write a little routine to try and see the effects a bit mor
        # clearly. I'm going to clean up my act a bit and write real documentation too, it's a
        # practice
        def show_boxes(img):
            '''Modifies the passed image to show a series of bounding boxes on an image as run

            :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

In [ ]:
```
# Not bad at all! It's interesting to see that kraken isn't completely sure what to do
# two column format. In some cases, kraken has identified a line in just a single colu
# in other cases kraken has spanned the line marker all the way across the page. Does
# Well, it really depends on our goal. In this case, I want to see if we can improve a
#
# So we're going to go a bit off script here. While this week of lectures is about lib
# goal of this last course is to give you confidence that you can apply your knowledge
# programming tasks, even if the library you are using doesn't quite do what you want.
#
# I'd like to pause the video for the moment and collect your thoughts. Looking at the
```

```
        # with the two column example and red boxes, how do you think we might modify this ima
        # kraken's ability to text lines?

In [5]: # Thanks for sharing your thoughts, I'm looking forward to seeing the breadth of ideas
        # in the course comes up with. Here's my partial solution -- while looking through the
        # the pageseg() function I saw that there are a few parameters we can supply in order
        # segmentation. One of these is the black_colseps parameter. If set to True, kraken wi
        # columns will be separated by black lines. This isn't our case here, but, I think we
        # tools to go through and actually change the source image to have a black separator b
        #
        # The first step is that I want to update the show_boxes() function. I'm just going to
        # 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

            :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

In [6]: # The next step is to think of the algorithm we want to apply to detect a white column
        # In experimenting a bit I decided that I only wanted to add the separator if the spac
        # at least 25 pixels wide, which is roughly the width of a character, and six lines hi
        # width is easy, lets just make a variable
        char_width=25
        # The height is harder, since it depends on the height of the text. I'm going to write
        # to calculate the average height of a line
        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 top - bo
            # 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 left corn
        # 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
```

In [7]:
```
# Ok, so the average height of a line is 31.
# Now, we want to scan through the image - looking at each pixel in turn - to determin
# is a block of whitespace. How bit of a block should we look for? That's a bit more o
# than a science. Looking at our sample image, I'm going to say an appropriate block s
# one char_width wide, and six line_heights tall. But, I honestly just made this up by
# the image, so I would encourage you to play with values as you explore.
# 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)
```

In [8]:
```
# It seems we will want to have a function which, given a pixel in an image, can check
# if that pixel has whitespace to the right and below it. Essentially, we want to test
# if the pixel is the upper left corner of something that looks like the gap_box. If s
# we should insert a line to "break up" this box before sending to kraken
#
# 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 description
    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 returns thi
    # as a tuple of integers, one for each color channel. Our tools all work with bina
    # images (black and white), so we should just get one value. If the value is 0 it'
    # 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
    # binarized. The algorithm to check our bounding box is fairly easy: we have a sin
    # which is our start and then we want to check all the pixels to the right of that
    # 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 within
                if x < img.width and y < img.height:
                    # if the pixel is white we don't do anything, if it's black, we just w
                    # 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
        # then we can return true -- this is a gap!
        return True
```

In [ ]:
```
# Alright, we have a function to check for a gap, called gap_check. What should we do
# we find a gap? For this, lets just draw a line in the middle of it. Lets create a ne
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 image
    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, the top
    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 modified the
```

In [ ]:
```
# Now, lets try it all out. This is pretty easy, we can just iterate through each pixe
# in the image, check if there is a gap, then insert a line if there is.
def process_image(img):
    '''Takes in an image of text and adds black vertical bars to break up columns
    :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)

    #Note: This will take some time to run! Be patient!
```

In [ ]:
```
# Not bad at all! The effect at the bottom of the image is a bit unexpected to me, but
# sense. You can imagine that there are several ways we might try and control this. Le
# this new image works when run through the kraken layout engine
display(show_boxes(i))
```

In [ ]:
```
# Looks like that is pretty accurate, and fixes the problem we faced. Feel free to exp
# with different settings for the gap heights and width and share in the forums. You'l
# method we created is really quite slow, which is a bit of a problem if we wanted to
# this on larger text. But I wanted to show you how you can mix your own logic and wor
# libraries you're using. Just because Kraken didn't work perfectly, doesn't mean we c
# 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
# 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. B
# go on to our last library, how well prepared do you think you are to take your pytho
# skills out into the wild?
```

0.2 Comparing Image Data Structures

In [ ]:
```
# OpenCV supports reading of images in most file formats, such as JPEG, PNG, and TIFF.
# video analysis requires converting images into grayscale first. This simplifies the
# noise allowing for improved analysis. Let's write some code that reads an image of a
# Mayweather and converts it into greyscale.

# 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)

# Now, before we get to the result, lets talk about docs. Just like tesseract, opencv
# package written in C++, and the docs for python are really poor. This is unfortunatl
# when python is being used as a wrapper. Thankfully, the web docs for opencv are actu
# so hit the website docs.opencv.org when you want to learn more about a particular fu
# case cvtColor converts from one color space to another, and we are convering our ima
```

```
        # Of course, we already know at least two different ways of doing this, using binariza
        # color spaces conversions

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

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

```
In [ ]: # The array is shown here as a list of lists, where the inner lists are filled with in
        # The dtype=uint8 definition indicates that each of the items in an array is an 8 bit
        # integer, which is very common for black and white images. So this is a pixel by pixe
        # of the image.
        #
        # The display package, however, doesn't know what to do with this image. So lets conve
        # into a PIL object to render it in the browser.
        from PIL import Image

        # PIL can take an array of data with a given color format and convert this into a PIL
        # This is perfect for our situation, as the PIL color mode, "L" is just an array of lu
        # values in unsigned integers
        image = Image.fromarray(gray, "L")
        display(image)
```

```
In [ ]: # Lets talk a bit more about images for a moment. Numpy arrays are multidimensional. F
        # instance, we can define an array in a single dimension:
        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 ac
        # all imaging libraries tend to expect at least two dimensions, a width and a height,
        # show a matrix. So if we put the single_dim inside of another array, this would be a
        # dimensional array with element in the height direction, and five in the width direct
        double_dim = np.array([single_dim])

        double_dim
```

```
In [ ]: # This should look pretty familiar, it's a lot like a list of lists! Lets see what thi
        # two dimensional array looks like if we display it
        display(Image.fromarray(double_dim, "L"))
```

```
In [ ]: # Pretty unexciting - it's just a little line. Five pixels in a row to be exact, of di
        # levels of black. The numpy library has a nice attribute called shape that allows us
        # many dimensions big an array is. The shape attribute returns a tuple that shows the
```

```
        # the image, by the width of the image
        double_dim.shape

In [ ]: # Lets take a look at the shape of our initial image which we loaded into the img vari
        img.shape

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

In [ ]: # Here we see that the color value is provided in full RGB using an unsigned integer.
        # means that each color can have one of 256 values, and the total number of unique col
        # that can be represented by this data is 256 * 256 *256 which is roughly 16 million c
        # We call this 24 bit color, which is 8+8+8.
        #
        # If you find yourself shopping for a television, you might notice that some expensive
        # are advertised as having 10 bit or even 12 bit panels. These are televisions where e
        # the red, green, and blue color channels are represented by 10 or 12 bits instead of
        # ten bit panels this means that there are 1 billion colors capable, and 12 bit panels
        # capable of over 68 billion colors!

In [ ]: # We're not going to talk much more about color in this course, but it's a fun subject
        # lets go back to this array representation of images, because we can do some interest
        # with this.
        #
        # One of the most common things to do with an ndarray is to reshape it -- to change th
        # of rows and columns that are represented so that we can do different kinds of operat
        # Here is our original two dimensional image
        print("Original image")
        print(gray)
        # If we wanted to represent that as a one dimensional image, we just call reshape
        print("New image")
        # And reshape takes the image as the first parameter, and a new shape as the second
        image1d=np.reshape(gray,(1,gray.shape[0]*gray.shape[1]))
        print(image1d)

In [ ]: # So, why are we talking about these nested arrays of bytes, we were supposed to be ta
        # about OpenCV as a library. Well, I wanted to show you that often libraries working o
        # same kind of principles, in this case images stored as arrays of bytes, are not repr
        # data in the same way in their APIs. But, by exploring a bit you can learn how the in
        # 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 ima
        # that we could draw lines to feed into kraken? Well, we use PIL to do this, using get
        # to look at individual pixels and see what the luminosity was, then ImageDraw.rectang
        # actually fill in a black bar separator. This was a nice high level API, and let us w
        # routines to do the work we wanted without having to understand too much about how th
```

```
        # were being stored. But it was computationally very slow.
        #
        # Instead, we could write the code to do this using matrix features within numpy. Lets
        # 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 [ ]: # 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
        # 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 using ti
        # format a[2:4,1:3]. You can think of this as first slicing along the rows dimension,
        # in the columns dimension. So in this example, that would be a matrix of rows 2, and
        # and columns 1, and 2. Here's a look at our image.
        gray[2:4,1:3]

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

In [ ]: # Ok, the last benefit of going to this low level approach to images is that we can ch
        # pixels very fast as well. Previously we were drawing rectangles and setting a fill a
        # width. This is nice if you want to do something like change the color of the fill fr
        # line, or draw complex shapes. But we really just want a line here. That's really eas
        # 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

In [ ]: # looks pretty boring, it's just a giant white square we can't see. But if we want, we
        # 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

In [ ]: # And that's exactly what we wanted to do. So, why do it this way, when it seems so mu
        # more low level? Really, the answer is speed. This paradigm of using matricies to sto
        # and manipulate bytes of data for images is much closer to how low level API and hard
        # 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 assig
# 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.
```

0.3 OpenCV

In [ ]:
```
# Ok, we're just about at the project for this course. If you reflect on the specializ
# as a whole you'll realize that you started with probably little or no understanding
# progressed through the basic control structures and libraries included with the lang
# with the help of a digital textbook, moved on to more high level representations of
# and functions with objects, and now started to explore third party libraries that ex
# python which allow you to manipulate and display images. This is quite an achievemen
#
# You have also no doubt found that as you have progressed the demands on you to engag
# discovery have also increased. Where the first assignments were maybe straight forwa
# ones in this week require you to struggle a bit more with planning and debugging cod
# you develop.
#
# But, you've persisted, and I'd like to share with you just one more set of features
# we head over to a project. The OpenCV library contains mechanisms to do face detecti
# images. The technique used is based on Haar cascades, which is a machine learning ap
# Now, we're not going to go into the machine learning bits, we have another specializ
# Applied Data Science with Python which you can take after this if you're interested
# But here we'll treat OpenCV like a black box.
#
# OpenCV comes with trained models for detecting faces, eyes, and smiles which we'll b
# You can train models for detecting other things - like hot dogs or flutes - and if y
# interested in that I'd recommend you check out the Open CV docs on how to train a ca
# 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 c
# portions of an image which are interesting.
#
# 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')
```

In [ ]:
```
# Ok, with the classifiers loaded, we now want to try and detect a face. Lets pull in
# 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)
# The next step is to use the face_cascade classifier. I'll let you go explore the doc
# would like to, but the norm is to use the detectMultiScale() function. This function
# 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
```

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

In [ ]:
```
# The resulting rectangles are in the format of (x,y,w,h) where x and y denote the upp
# left hand point for the image and the width and height represent the bounding box. W
# how to handle this in PIL
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)
```

In [ ]:
```
# 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
# as (x,y,w,h), while PIL.ImageDraw is looking for (x1,y1,x2,y2). Looks like an easy f
# Wipe our old image
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)
```

In [ ]:
```
# We see the face detection works pretty good on this image! Note that it's apparent t
# not head detection, but that the haarcascades file we used is looking for eyes and a
# Lets try this on something a bit more complex, lets read in our MSI recruitment imag
img = cv.imread('readonly/msi_recruitment.gif')
# And lets take a look at that image
display(Image.fromarray(img))
```

In [ ]:
```
# Whoa, what's that error about? It looks like there is an error on a line deep within
# Image.py file, and it is trying to call an internal private member called __array_in
# 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.
# kind of a pain and unfortunate. But we know how to fix that right? One was is that w
```

```
        # just open this in PIL and then save it as a png, then open that in open cv.
        #
        # 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 [ ]:
```
        # Ok, now that the conversion of format is done, lets try reading this back into openc
        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 rectangles in (
            # format, that is, a list of lists. Instead of having to do an iteration and then
            # pull out each item, we can use tuple unpacking to pull out individual items in t
            # 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 [ ]:
```
        # What happened here!? We see that we have detected faces, and that we have drawn boxe
        # around those faces on the image, but that the colors have gone all weird! This, it t
        # out, has to do with color limitations for gif images. In short, a gif image has a ve
        # limited number of colors. This is called a color pallette after the pallette artists
        # use to mix paints. For gifs the pallette can only be 256 colors -- but they can be *
        # 256 colors. When a new color is introduced, is has to take the space of an old color
        # In this case, PIL adds white to the pallette but doesn't know which color to replace
        # thus messes up the image.
        #
        # Who knew there was so much to learn about image formats? We can see what mode the im
        # is in with the .mode attribute
        pil_img.mode
```

In [ ]:
```
        # We can see a list of modes in the PILLOW documentation, and they correspond with the
        # color spaces we have been using. For the moment though, lets change back to RGB, whi
        # 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
```

In [ ]:
```
# 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 appropriately.
    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 have mis
# four faces. In the machine learning world we would call these false negatives - some
# which the machine thought was not a face (so a negative), but that it was incorrect
# Consequently, we would call the actual faces that were detected as true positives -
# something that the machine thought was a face and it was correct on. This leaves us
# false positives - something the machine thought was a face but it wasn't. We see the
# two of these in the image, picking up shadow patterns or textures in shirts and matc
# them with the haarcascades. Finally, we have true negatives, or the set of all possi
# rectangles the machine learning classifier could consider where it correctly indicat
# the result was not a face. In this case there are many many true negatives.
```

In [ ]:
```
# 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 function
# 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 [ ]:
```
# Ok, first up, we could try and binarize this image. It turns out that opencv has a b
# binarization function called threshold(). You simply pass in the image, the midpoint
# the maximum value, as well as a flag which indicates whether the threshold should be
# binary or something else. Lets try this.
cv_img_bin=cv.threshold(img,120,255,cv.THRESH_BINARY)[1] # returns a list, we want the
# 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 positi
        # towards the bottom, where the classifier detected the sunglasses as eyes and the dar
        # line below as a mouth.
        #
        # If you're following in the notebook with this video, why don't you pause things and
        # few different parameters for the thresholding value?

In [ ]: # The detectMultiScale() function from OpenCV also has a couple of parameters. The fir
        # these is the scale factor. The scale factor changes the size of rectangles which are
        # considered against the model, that is, the haarcascades XML file. You can think of i
        # 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)

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 positives,
        # which are correctly identified faces, and 3 false negatives, which are faces which
        # 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 positives but
        # also lose one of the true positives, the person to the right wearing a hat. And
        # 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 artificial
        # intelligence. There is a trade off between not only how accurate a model is, but how
        # the inaccuracy actually happens. Which of these three models do you think is best?

In [ ]: # Well, the answer to that question is really, "it depends". It depends why you are tr
        # to detect faces, and what you are going to do with them. If you think these issues
        # are interesting, you might want to check out the Applied Data Science with Python
        # specialization Michigan offers on Coursera.
        #
        # Ok, beyond an opportunity to advertise, did you notice anything else that happened w
        # we changed the scale factor? It's subtle, but the speed at which the processing ran
        # took longer at smaller scale factors. This is because more subimages are being consi
        # 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 lin
```

```
        # that starts with a percentage sign in jupyter is called a "magic function". This isn
        # normal python - it's actually a shorthand way of writing a function which Jupyter
        # has predefined. It looks a lot like the decorators we talked about in a previous
        # 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 timeit, an
        # repeats a piece of python ten times (by default) and tells you the average speed 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)
```

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

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

0.4 More Jupyter Widgets

In [ ]:
```
        # 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 lecture 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 chatting.
        # 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)
```

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

In [ ]:
```
        # The image recorder lets us actually grab images from the camera stream. There are fe
        # for downloading and using the image as well. We see that the default format is a png
        # 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 call th
        # download() function it will actually store the results of the camera which is hooked
        # 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)

In [ ]: # Ok, the object that it stores is an ipywidgets.widgets.widget_media.Image. How do we
        # something useful with this? Well, an inspection of the object shows that there is a
        # value field which actually holds the bytes behind the image. And we know how to disp
        # 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)

In [ ]: # Great, you see a picture! Hopefully you are following along in one of the notebooks
        # 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 computer
        # You already know how to identify a face in the webcam picture, or try and capture te
        # from within the picture. With OpenCV there are any number of other things you can do
        # with a webcam, the Jupyter notebooks, and python!
```