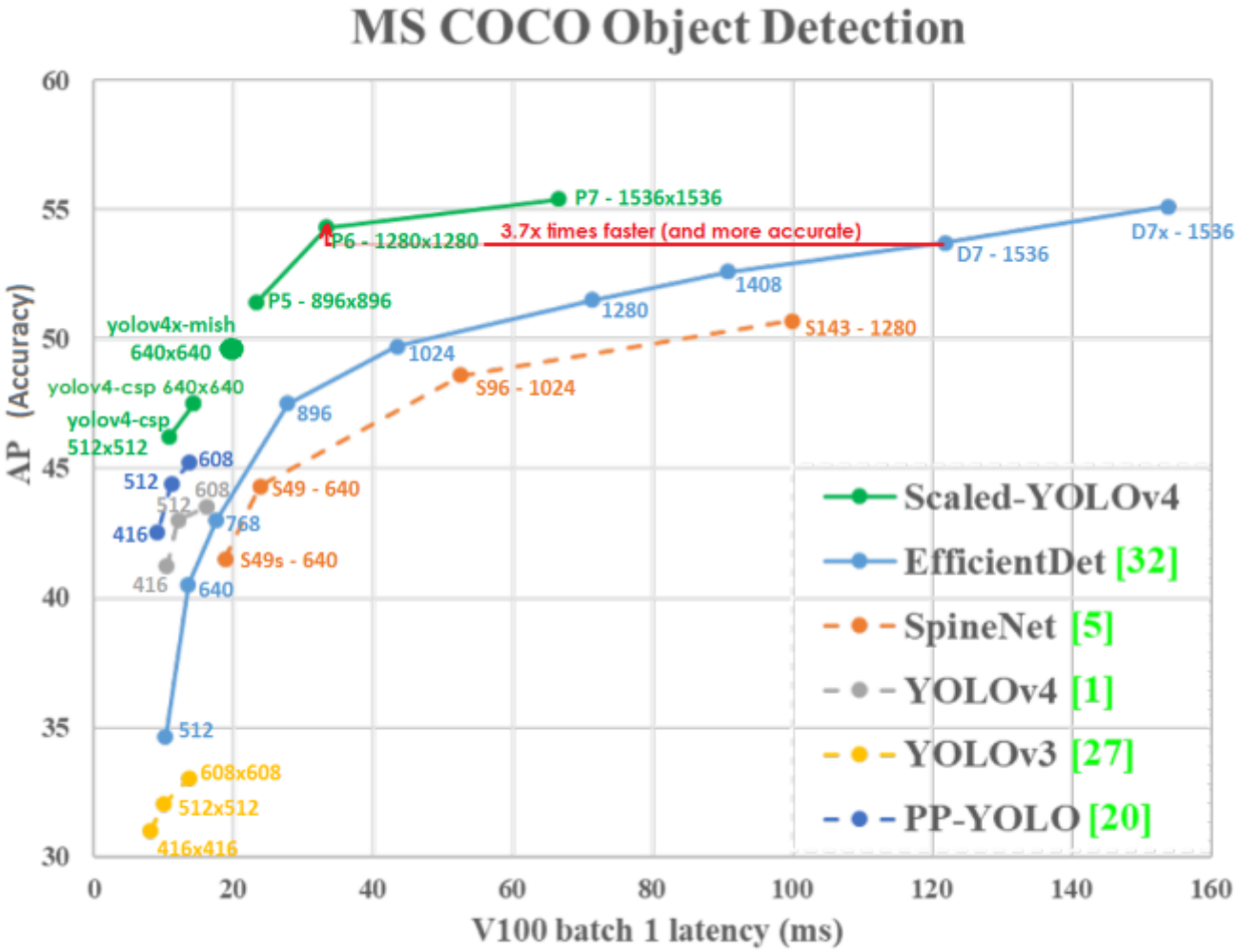
YOLOv4 Object Detection on Webcam In Google Colab

This notebook will walkthrough all the steps for performing YOLOv4 object detections on your webcam while in Google Colab. We will be using scaled-YOLOv4 (yolov4-csp) for this tutorial, the fastest and most accurate object detector there currently is.




```
# import dependencies
from IPython.display import display, Javascript, Image
from google.colab.output import eval_js
from google.colab.patches import cv2_imshow
from base64 import b64decode, b64encode
import cv2
import numpy as np
import PIL
import io
import html
import time
import matplotlib.pyplot as plt
%matplotlib inline
```

Cloning and Setting Up Darknet for YOLOv4

We will be using the famous AlexeyAB's darknet repository in this tutorial to perform YOLOv4 detections.

```
# clone darknet repo
!git clone https://github.com/AlexeyAB/darknet
```

```
Cloning into 'darknet'...
remote: Enumerating objects: 15313, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 15313 (delta 0), reused 3 (delta 0), pack-reused 15308
Receiving objects: 100% (15313/15313), 13.72 MiB | 18.10 MiB/s, done.
Resolving deltas: 100% (10400/10400), done.
```

```
# change makefile to have GPU, OPENCV and LIBSO enabled
%cd darknet
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile
!sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
!sed -i 's/LIBSO=0/LIBSO=1/' Makefile
```

```
/content/darknet
```

```
# make darknet (builds darknet so that you can then use the darknet.py file and ha
!make
```

```
./src/network_kernels.cu:379:7: warning: variable ‘l’ set but not used [-Wu
        layer l = net.layers[net.n - 1];
      ^
nvcc -gencode arch=compute_35,code=sm_35 -gencode arch=compute_50,code=[sm_
nvcc warning : The 'compute_35', 'compute_37', 'compute_50', 'sm_35', 'sm_3
g++ -std=c++11 -std=c++11 -Iinclude/ -I3rdparty/stb/include -DOPENCV `pkg-c
g++ -std=c++11 -shared -std=c++11 -fvisibility=hidden -DLIB_EXPORTS -Iinclu
In file included from src/yolo_v2_class.cpp:2:0:
include/yolo_v2_class.hpp: In member function ‘void track_kalman_t::clear_o
include/yolo_v2_class.hpp:879:50: warning: comparison between signed and un
                 if ((result_vec_pred[state_id].x > img_size.width) ||
include/yolo_v2_class.hpp:880:50: warning: comparison between signed and un
                     (result_vec_pred[state_id].y > img_size.height))
include/yolo_v2_class.hpp: In member function ‘track_kalman_t::tst_t track_
include/yolo_v2_class.hpp:900:30: warning: comparison between signed and un
         for (size_t i = 0; i < max_objects; ++i)
                            ~~^~~~~~~~~~~~
include/yolo_v2_class.hpp: In member function ‘std::vector<bbox_t> track_ka
include/yolo_v2_class.hpp:990:30: warning: comparison between signed and un
         for (size_t i = 0; i < max_objects; ++i)
                            ~~^~~~~~~~~~~~
include/yolo_v2_class.hpp: In member function ‘std::vector<bbox_t> track_ka
include/yolo_v2_class.hpp:1025:30: warning: comparison between signed and u
         for (size_t i = 0; i < max_objects; ++i)
                            ~~^~~~~~~~~~~~
src/yolo_v2_class.cpp: In member function ‘std::vector<bbox_t> Detector::tr
src/yolo_v2_class.cpp:439:40: warning: comparison between signed and unsign
         if (prev_bbox_vec_deque.size() > frames_story) prev_bbox_vec_deque
             ~~~~~~~~~~~~~~~~~~~~~~~~~~^~~~~~~~~~~~~~
```

```
src/yolo_v2_class.cpp:454:34: warning: comparison between signed and unsign
                    if (cur_dist < max_dist && (k.track_id == 0 || dist_ve
                                ~~~~~~~~~^~~~~~~~~
src/yolo_v2_class.cpp:478:40: warning: comparison between signed and unsign
        if (prev_bbox_vec_deque.size() > frames_story) prev_bbox_vec_deque
            ~~~~~~~~~~~~~~~~~~~~~~~~~~^~~~~~~~~~~~~
g++ -std=c++11 -std=c++11 -Iinclude/ -I3rdparty/stb/include -DOPENCV `pkg-c
In file included from src/yolo_console_dll.cpp:23:0:
include/yolo_v2_class.hpp: In member function ‘void track_kalman_t::clear_o
include/yolo_v2_class.hpp:879:50: warning: comparison between signed and un
                if ((result_vec_pred[state_id].x > img_size.width) ||
include/yolo_v2_class.hpp:880:50: warning: comparison between signed and un
                    (result_vec_pred[state_id].y > img_size.height))
include/yolo_v2_class.hpp: In member function ‘track_kalman_t::tst_t track_
include/yolo_v2_class.hpp:900:30: warning: comparison between signed and un
        for (size_t i = 0; i < max_objects; ++i)
                           ~~^~~~~~~~~~~~
include/yolo_v2_class.hpp: In member function ‘std::vector<bbox_t> track_ka
include/yolo_v2_class.hpp:990:30: warning: comparison between signed and un
        for (size_t i = 0; i < max_objects; ++i)

                           ~~^~~~~~~~~~~~
include/yolo_v2_class.hpp: In member function ‘std::vector<bbox_t> track_ka
include/yolo_v2_class.hpp:1025:30: warning: comparison between signed and u
        for (size_t i = 0; i < max_objects; ++i)
                           ~~^~~~~~~~~~~~
src/yolo_console_dll.cpp: In function ‘void draw_boxes(cv::Mat, std::vector
src/yolo_console_dll.cpp:192:46: warning: comparison between signed and uns
            int max_width = (text_size.width > i.w + 2) ? text_size.width
                             ~~~~~~~~~~~~~~~~^~~~~~~~
```

```
# get bthe scaled yolov4 weights file that is pre-trained to detect 80 classes (ob
!wget --load-cookies /tmp/cookies.txt "https://docs.google.com/uc?export=download&
```

```
--2021-09-30 17:00:29--  https://docs.google.com/uc?export=download&confirm=v
Resolving docs.google.com (docs.google.com)... 209.85.145.100, 209.85.145.102
Connecting to docs.google.com (docs.google.com)|209.85.145.100|:443... connec
HTTP request sent, awaiting response... 302 Moved Temporarily
Location: https://doc-0g-60-docs.googleusercontent.com/docs/securesc/duo2papc
--2021-09-30 17:00:29--  https://doc-0g-60-docs.googleusercontent.com/docs/se
Resolving doc-0g-60-docs.googleusercontent.com (doc-0g-60-docs.googleusercont
Connecting to doc-0g-60-docs.googleusercontent.com (doc-0g-60-docs.googleuser
HTTP request sent, awaiting response... 302 Found
Location: https://docs.google.com/nonceSigner?nonce=j05dkk0n26v9a&continue=ht
--2021-09-30 17:00:29--  https://docs.google.com/nonceSigner?nonce=j05dkk0n26
Connecting to docs.google.com (docs.google.com)|209.85.145.100|:443... connec
HTTP request sent, awaiting response... 302 Found
Location: https://doc-0g-60-docs.googleusercontent.com/docs/securesc/duo2papc
--2021-09-30 17:00:29--  https://doc-0g-60-docs.googleusercontent.com/docs/se
Connecting to doc-0g-60-docs.googleusercontent.com (doc-0g-60-docs.googleuser
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/octet-stream]
Saving to: ‘yolov4-csp.weights’

yolov4-csp.weights      [      <=>           ] 202.13M   176MB/s    in 1.2s

2021-09-30 17:00:30 (176 MB/s) - ‘yolov4-csp.weights’ saved [211944840]
```

Darknet for Python

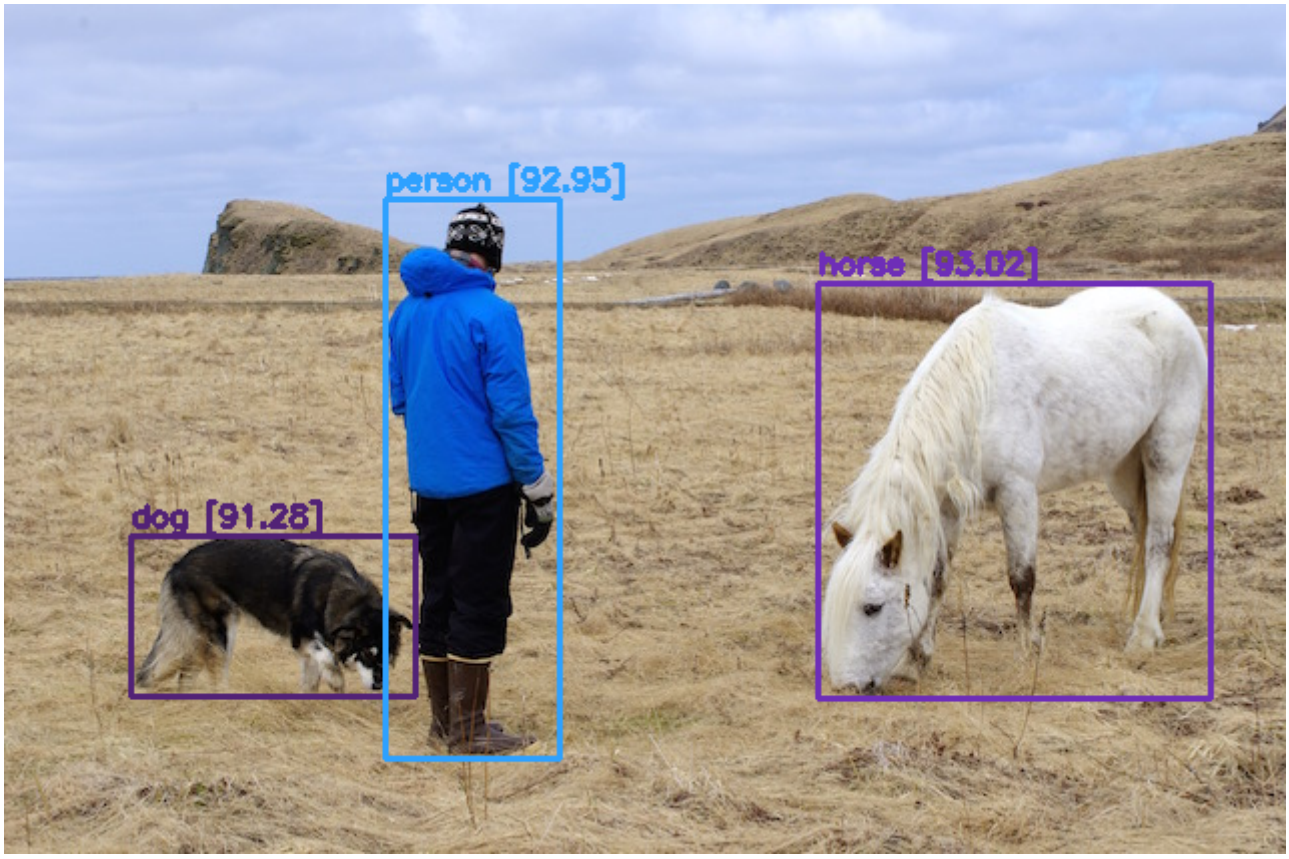
In order to utilize YOLOv4 with Python code we will use some of the pre-built functions found within darknet.py by importing the functions into our workstation. Feel free to checkout the darknet.py file to see the function definitions in detail!

```
# import darknet functions to perform object detections
from darknet import *
# load in our YOLOv4 architecture network
network, class_names, class_colors = load_network("cfg/yolov4-obj3.cfg", "data/obj
width = network_width(network)
height = network_height(network)

# darknet helper function to run detection on image
def darknet_helper(img, width, height):
  darknet_image = make_image(width, height, 3)
  img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
  img_resized = cv2.resize(img_rgb, (width, height),
                           interpolation=cv2.INTER_LINEAR)

  # get image ratios to convert bounding boxes to proper size
  img_height, img_width, _ = img.shape
  width_ratio = img_width/width
  height_ratio = img_height/height

  # run model on darknet style image to get detections
  copy_image_from_bytes(darknet_image, img_resized.tobytes())
  detections = detect_image(network, class_names, darknet_image)
  free_image(darknet_image)
  return detections, width_ratio, height_ratio
```

YOLOv4 Example on Test Image

Let's make sure our model has successfully been loaded and that we can make detections properly on a test image.

```
# run test on person.jpg image that comes with repository
image = cv2.imread("data/person.jpg")
detections, width_ratio, height_ratio = darknet_helper(image, width, height)

for label, confidence, bbox in detections:
  left, top, right, bottom = bbox2points(bbox)
  left, top, right, bottom = int(left * width_ratio), int(top * height_ratio), int
  cv2.rectangle(image, (left, top), (right, bottom), class_colors[label], 2)
  cv2.putText(image, "{} [{:.2f}]".format(label, float(confidence)),
                    (left, top - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                    class_colors[label], 2)
cv2_imshow(image)
```



Helper Functions

Here are a few helper functions defined that will be used to easily convert between different image types within our later steps.

```
# function to convert the JavaScript object into an OpenCV image
def js_to_image(js_reply):
  """
  Params:
          js_reply: JavaScript object containing image from webcam
  Returns:
          img: OpenCV BGR image
  """
  # decode base64 image
  image_bytes = b64decode(js_reply.split(',')[1])
  # convert bytes to numpy array
  jpg_as_np = np.frombuffer(image_bytes, dtype=np.uint8)
  # decode numpy array into OpenCV BGR image
  img = cv2.imdecode(jpg_as_np, flags=1)

  return img

# function to convert OpenCV Rectangle bounding box image into base64 byte string
def bbox_to_bytes(bbox_array):
  """
  Params:
          bbox_array: Numpy array (pixels) containing rectangle to overlay on vide
  Returns:
        bytes: Base64 image byte string
  """
```

```
  # convert array into PIL image
  bbox_PIL = PIL.Image.fromarray(bbox_array, 'RGBA')
  iobuf = io.BytesIO()
  # format bbox into png for return
  bbox_PIL.save(iobuf, format='png')
  # format return string
  bbox_bytes = '{}'.format((str(b64encode(iobuf.getvalue()),

  return bbox_bytes
```

YOLOv4 on Wecam Images

Running YOLOv4 on images taken from webcam is fairly straight-forward. We will utilize code within Google Colab's **Code Snippets** that has a variety of useful code functions to perform various tasks.

We will be using the code snippet for **Camera Capture** which runs JavaScript code to utilize your computer's webcam. The code snippet will take a webcam photo, which we will then pass into our YOLOv4 model for object detection.

Below is a function to take the webcam picture using JavaScript and then run YOLOv4 on it.

```
def take_photo(filename='photo.jpg', quality=0.8):
  js = Javascript('''
    async function takePhoto(quality) {
      const div = document.createElement('div');
      const capture = document.createElement('button');
      capture.textContent = 'Capture';
      div.appendChild(capture);

      const video = document.createElement('video');
      video.style.display = 'block';
      const stream = await navigator.mediaDevices.getUserMedia({video: true});

      document.body.appendChild(div);
      div.appendChild(video);
      video.srcObject = stream;
      await video.play();

      // Resize the output to fit the video element.
      google.colab.output.setIframeHeight(document.documentElement.scrollHeight, t

      // Wait for Capture to be clicked.
      await new Promise((resolve) => capture.onclick = resolve);

      const canvas = document.createElement('canvas');
      canvas.width = video.videoWidth;
      canvas.height = video.videoHeight;
      canvas.getContext('2d').drawImage(video, 0, 0);
      stream.getVideoTracks()[0].stop();
      div.remove();
      return canvas.toDataURL('image/jpeg', quality);
```

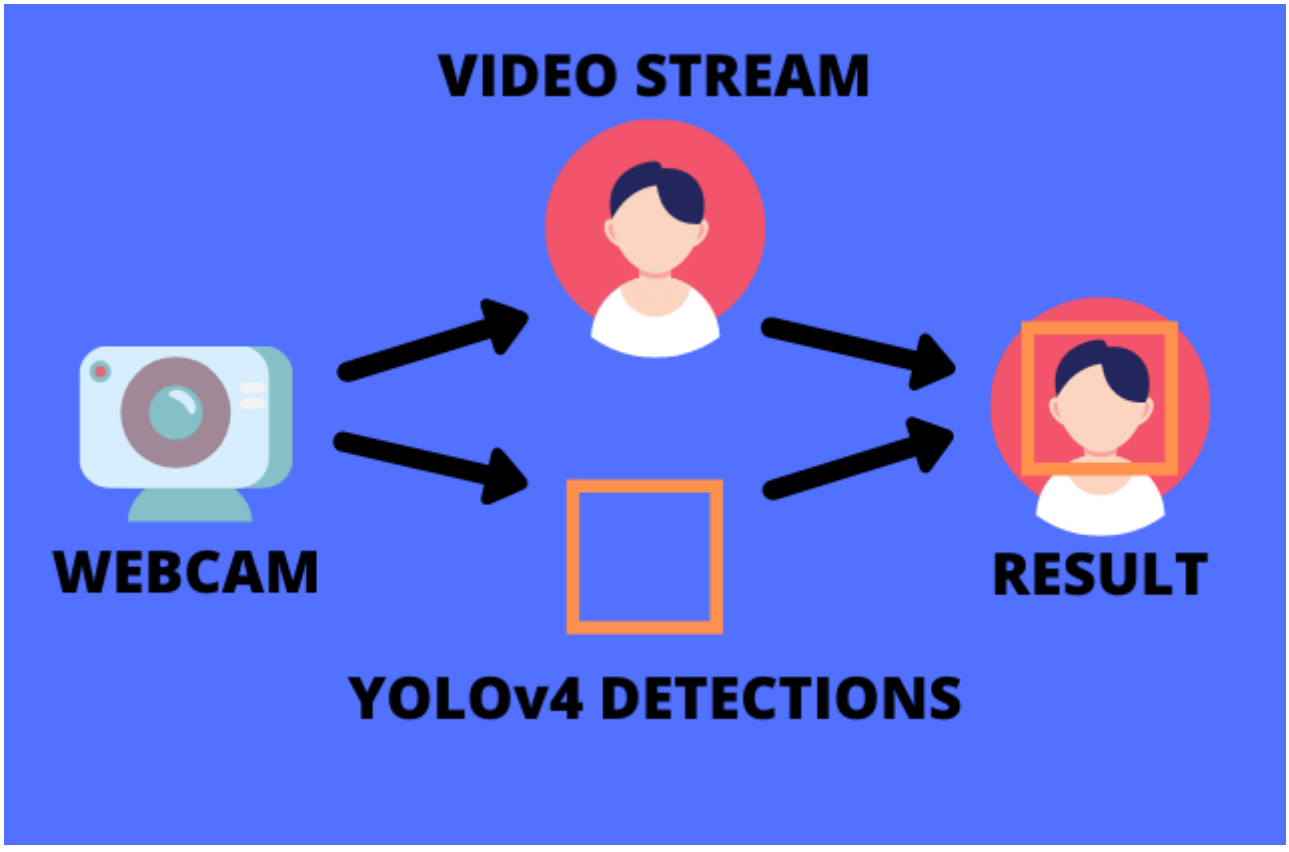
```
        }
        ''')
      display(js)

      # get photo data
      data = eval_js('takePhoto({})'.format(quality))
      # get OpenCV format image
      img = js_to_image(data)

      # call our darknet helper on webcam image
      detections, width_ratio, height_ratio = darknet_helper(img, width, height)

      # loop through detections and draw them on webcam image
      for label, confidence, bbox in detections:
        left, top, right, bottom = bbox2points(bbox)
        left, top, right, bottom = int(left * width_ratio), int(top * height_ratio), i
        cv2.rectangle(img, (left, top), (right, bottom), class_colors[label], 2)
        cv2.putText(img, "{} [{:.2f}]".format(label, float(confidence)),
                          (left, top - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                          class_colors[label], 2)
      # save image
      cv2.imwrite(filename, img)

      return filename


    try:
      filename = take_photo('photo.jpg')
      print('Saved to {}'.format(filename))

      # Show the image which was just taken.
      display(Image(filename))
    except Exception as err:
      # Errors will be thrown if the user does not have a webcam or if they do not
      # grant the page permission to access it.
      print(str(err))
```

```
Saved to photo.jpg
```

YOLOv4 on Webcam Videos

Running YOLOv4 on webcam video is a little more complex than images. We need to start a video stream using our webcam as input. Then we run each frame through our YOLOv4 model and create an overlay image that contains bounding box of detection(s). We then overlay the bounding box image back onto the next frame of our video stream.

YOLOv4 is so fast that it can run the detections in real-time!



Below is a function to start up the video stream using similar JavaScript as was used for images. The video stream frames are fed as input to YOLOv4.

```
# JavaScript to properly create our live video stream using our webcam as input
def video_stream():
  js = Javascript('''
    var video;
    var div = null;
    var stream;
    var captureCanvas;
    var imgElement;
```

```
        var labelElement;

        var pendingResolve = null;
        var shutdown = false;

        function removeDom() {
           stream.getVideoTracks()[0].stop();
           video.remove();
           div.remove();
           video = null;
           div = null;
           stream = null;
           imgElement = null;
           captureCanvas = null;
           labelElement = null;
        }

        function onAnimationFrame() {
          if (!shutdown) {
            window.requestAnimationFrame(onAnimationFrame);
          }
          if (pendingResolve) {
            var result = "";
            if (!shutdown) {
              captureCanvas.getContext('2d').drawImage(video, 0, 0, 640, 480);
              result = captureCanvas.toDataURL('image/jpeg', 0.8)
            }
            var lp = pendingResolve;
            pendingResolve = null;
            lp(result);
          }
        }

        async function createDom() {
          if (div !== null) {
            return stream;
          }

          div = document.createElement('div');
          div.style.border = '2px solid black';
          div.style.padding = '3px';
          div.style.width = '100%';
          div.style.maxWidth = '600px';
          document.body.appendChild(div);

          const modelOut = document.createElement('div');
          modelOut.innerHTML = "<span>Status:</span>";
          labelElement = document.createElement('span');
          labelElement.innerText = 'No data';
          labelElement.style.fontWeight = 'bold';
          modelOut.appendChild(labelElement);
          div.appendChild(modelOut);

          video = document.createElement('video');
          video.style.display = 'block';
```

```
      video.width = div.clientWidth - 6;
      video.setAttribute('playsinline', '');
      video.onclick = () => { shutdown = true; };
      stream = await navigator.mediaDevices.getUserMedia(
          {video: { facingMode: "environment"}});
      div.appendChild(video);

      imgElement = document.createElement('img');
      imgElement.style.position = 'absolute';
      imgElement.style.zIndex = 1;
      imgElement.onclick = () => { shutdown = true; };
      div.appendChild(imgElement);

      const instruction = document.createElement('div');
      instruction.innerHTML =
          '<span style="color: red; font-weight: bold;">' +
          'When finished, click here or on the video to stop this demo</span>';
      div.appendChild(instruction);
      instruction.onclick = () => { shutdown = true; };

      video.srcObject = stream;
      await video.play();

      captureCanvas = document.createElement('canvas');
      captureCanvas.width = 640; //video.videoWidth;
      captureCanvas.height = 480; //video.videoHeight;
      window.requestAnimationFrame(onAnimationFrame);

      return stream;
    }
    async function stream_frame(label, imgData) {
      if (shutdown) {
        removeDom();
        shutdown = false;
        return '';
      }

      var preCreate = Date.now();
      stream = await createDom();

      var preShow = Date.now();
      if (label != "") {
        labelElement.innerHTML = label;
      }

      if (imgData != "") {
        var videoRect = video.getClientRects()[0];
        imgElement.style.top = videoRect.top + "px";
        imgElement.style.left = videoRect.left + "px";
        imgElement.style.width = videoRect.width + "px";
        imgElement.style.height = videoRect.height + "px";
        imgElement.src = imgData;
      }

      var preCapture = Date.now();
```

```
      var result = await new Promise(function(resolve, reject) {
        pendingResolve = resolve;
      });
      shutdown = false;

      return {'create': preShow - preCreate,
              'show': preCapture - preShow,
              'capture': Date.now() - preCapture,
              'img': result};
    }
    ''')

  display(js)

def video_frame(label, bbox):
  data = eval_js('stream_frame("{}", "{}")'.format(label, bbox))
  return data
```

Running on Webcam Video

```
# start streaming video from webcam
video_stream()
# label for video
label_html = 'Capturing...'
# initialze bounding box to empty
bbox = ''
count = 0
while True:
    js_reply = video_frame(label_html, bbox)
    if not js_reply:
        break

    # convert JS response to OpenCV Image
    frame = js_to_image(js_reply["img"])

    # create transparent overlay for bounding box
    bbox_array = np.zeros([480,640,4], dtype=np.uint8)

    # call our darknet helper on video frame
    detections, width_ratio, height_ratio = darknet_helper(frame, width, height)

    # loop through detections and draw them on transparent overlay image
    for label, confidence, bbox in detections:
      left, top, right, bottom = bbox2points(bbox)
      left, top, right, bottom = int(left * width_ratio), int(top * height_ratio),
      bbox_array = cv2.rectangle(bbox_array, (left, top), (right, bottom), class_c
      bbox_array = cv2.putText(bbox_array, "{} [{:.2f}]".format(label, float(confi
                        (left, top - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                        class_colors[label], 2)

    bbox_array[:,:,3] = (bbox_array.max(axis = 2) > 0 ).astype(int) * 255
    # convert overlay of bbox into bytes
    bbox_bytes = bbox_to_bytes(bbox_array)
```

```
# update bbox so next frame gets new overlay
bbox = bbox_bytes
```

Status:**Capturing...**

When finished, click here or on the video to stop this demo

Hope You Enjoyed!

If you enjoyed the tutorial and want to see more videos or tutorials check out my YouTube channel HERE

Have a great day!

Executing (4m 23s) Cell › video_frame() › eval_js() › read_reply_from_input()