```
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
```

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

```
Cloning into 'darknet'...
remote: Enumerating objects: 15851, done.
remote: Counting objects: 100% (18/18), done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 15851 (delta 5), reused 11 (delta 4), pack-reused 15833 (from 1)
Receiving objects: 100% (15851/15851), 14.42 MiB | 15.72 MiB/s, done.
Resolving deltas: 100% (10671/10671), done.
```

```
%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
```

```
!wget --load-cookies /tmp/cookies.txt "https://docs.google.com/uc?export=download&confirm=$(wget --quiet --save-cookies /tmp/cookies.txt --keep-session-cook
```

```
--2024-09-19 11:39:26--  https://docs.google.com/uc?export=download&confirm=&id=1V3vsIaxAlGWvK4Aar9bAiK5U0QFttKwq
Resolving docs.google.com (docs.google.com)... 142.251.175.100, 142.251.175.139, 142.251.175.113, ...
Connecting to docs.google.com (docs.google.com)|142.251.175.100|:443... connected.
HTTP request sent, awaiting response... 303 See Other
Location: https://drive.usercontent.google.com/download?id=1V3vsIaxAlGWvK4Aar9bAiK5U0QFttKwq&export=download [following]
--2024-09-19 11:39:27--  https://drive.usercontent.google.com/download?id=1V3vsIaxAlGWvK4Aar9bAiK5U0QFttKwq&export=download
Resolving drive.usercontent.google.com (drive.usercontent.google.com)... 74.125.130.132, 2404:6800:4003:c01::84
Connecting to drive.usercontent.google.com (drive.usercontent.google.com)|74.125.130.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2430 (2.4K) [text/html]
Saving to: ‘yolov4-csp.weights’

yolov4-csp.weights  100%[===================>]   2.37K  --.-KB/s    in 0s

2024-09-19 11:39:27 (32.7 MB/s) - ‘yolov4-csp.weights’ saved [2430/2430]
```

```
# 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-csp.cfg", "cfg/coco.data", "yolov4-csp.weights")
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
```

```
# 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(right * width_ratio), int(bottom * height_ratio)
  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)
```

```
# 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 to be overlayed on video stream
def bbox_to_bytes(bbox_array):
  """
  Params:
        bbox_array: Numpy array (pixels) containing rectangle to overlay on video stream.
  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()), 'utf-8')))

  return bbox_bytes


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, true);

      // 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), int(right * width_ratio), int(bottom * height_ratio)
    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))
```

Mostrar saída oculta

```
# 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


# 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), int(right * width_ratio), int(bottom * height_ratio)
      bbox_array = cv2.rectangle(bbox_array, (left, top), (right, bottom), class_colors[label], 2)
      bbox_array = cv2.putText(bbox_array, "{} [{:.2f}]".format(label, float(confidence)),
                        (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
```