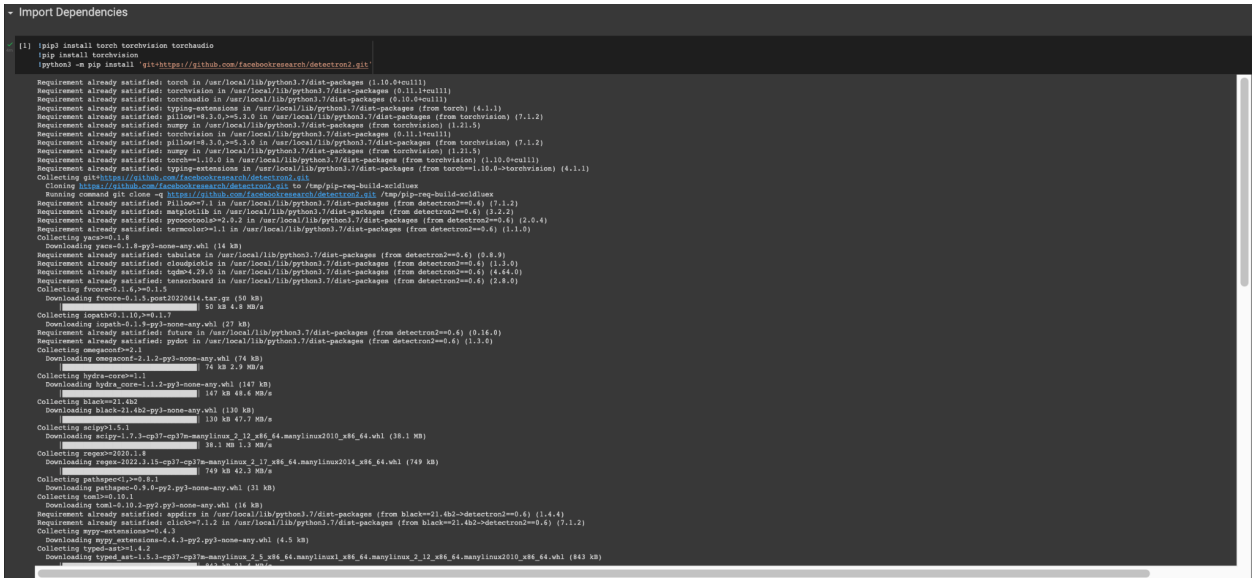


1. In this image, we are pip installing Torch, Torchvision, and Detectron2.

2. In this image we are installing the dependencies we will use in this notebook.

Code Snippet to use Webcam on Colab

```
from IPython.display import display, Javascript
from google.colab.output import eval_js
from base64 import b64decode

def record_video(filename='video.mp4'):

  # This function uses the take_photo() function provided by the Colab team as a
  # starting point, along with a bunch of stuff from Stack overflow, and some sample code
  # from: https://developer.mozilla.org/en-US/docs/Web/API/MediaStream_Recording_API

  js = Javascript("""
    async function recordVideo() {
      const options = { mimeType: "video/webm; codecs=vp9" };
      const div = document.createElement('div');
      const capture = document.createElement('button');
      const stopCapture = document.createElement("button");
      capture.textContent = "Start Recording";
      capture.style.background = "green";
      capture.style.color = "white";

      stopCapture.textContent = "Stop Recording";
      stopCapture.style.background = "red";
      stopCapture.style.color = "white";
      div.appendChild(capture);

      const video = document.createElement('video');
      const recordingVid = document.createElement("video");
      video.style.display = 'block';

      const stream = await navigator.mediaDevices.getUserMedia({video: true});
      let recorder = new MediaRecorder(stream, options);
      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);

      await new Promise((resolve) => {
        capture.onclick = resolve;
      });
      recorder.start();
      capture.replaceWith(stopCapture);
      await new Promise((resolve) => stopCapture.onclick = resolve);
      recorder.stop();

      let recData = await new Promise((resolve) => recorder.ondataavailable = resolve);
      let arrBuff = await recData.data.arrayBuffer();
      stream.getVideoTracks()[0].stop();
      div.remove();

      let binaryString = "";
      let bytes = new Uint8Array(arrBuff);
      bytes.forEach((byte) => {
        binaryString += String.fromCharCode(byte);
      })
```

```
      binaryString += String.fromCharCode(byte);
      })
      return btoa(binaryString);
    }
    """)
  try:
    display(js)
    data = eval_js('recordVideo({})')
    binary = b64decode(data)
    with open(filename, "wb") as video_file:
      video_file.write(binary)
    print(
        f"Finished recording video. Saved binary under filename in current working directory: {filename}"
    )
  except Exception as err:
      # In case any exceptions arise
      print(str(err))
  return filename
```

3. In these images, we are using the code snippet provided by colab to access webcams locally. This can be found through the snippets button available on Colab and inserted into the document.

This creates webcam record window

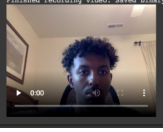
```
# Run the function, get the video path as saved in your notebook, and play it back here.
from IPython.display import HTML
from base64 import b64encode

video_width = 300

video_path = record_video()
video_file = open(video_path, "r+b").read()

video_url = f"data:video/mp4;base64,{b64encode(video_file).decode()}"
HTML(f"""<video width={video_width} controls><source src='{video_url}'></video>""")
```

```
Finished recording video. Saved binary under filename in current working directory: video.mp4
```



4. In this code chunk, we are creating a window to record and capture a video through the device's webcam. This will be saved as “video.mp4” in the working directory.

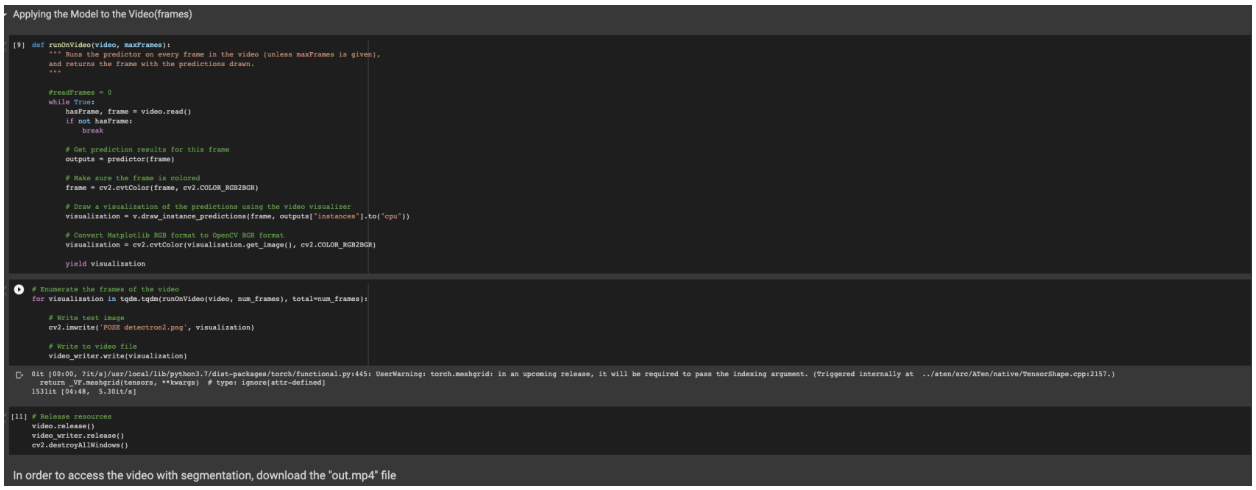
Accessing the components of the Video

```
[5] # Extract video properties
    video = cv2.VideoCapture('/content/video.mp4')
    width = int(video.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(video.get(cv2.CAP_PROP_FRAME_HEIGHT))
    frames_per_second = video.get(cv2.CAP_PROP_FPS)
    num_frames = int(video.get(cv2.CAP_PROP_FRAME_COUNT))
```

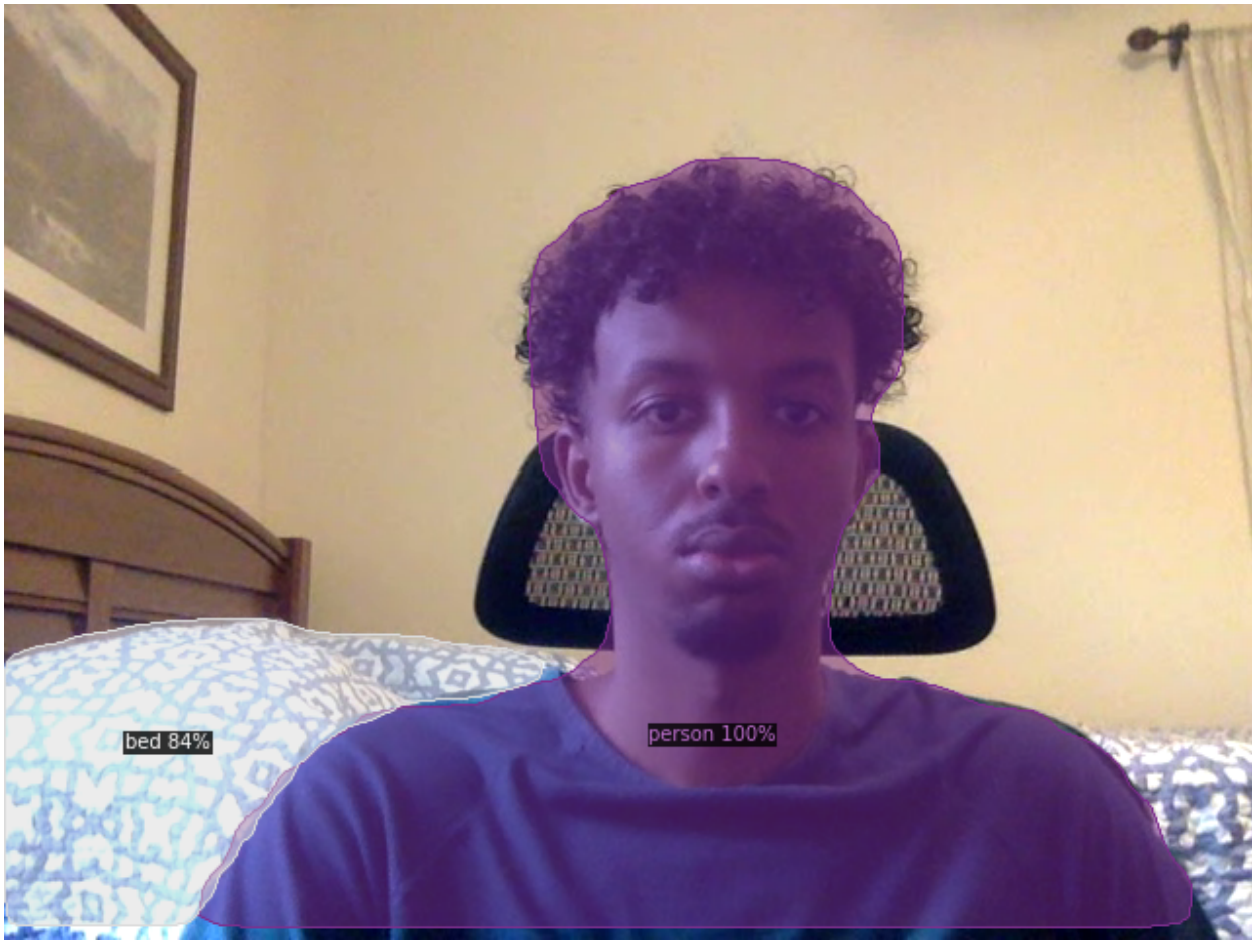
```
[6] # Initialize video writer
    #trying to change fps speed to 30 to make video normal
    video_writer = cv2.VideoWriter('out.mp4', fourcc=cv2.VideoWriter_fourcc(*'mp4v'), fps = 30, frameSize=(width, height), isColor=True)
```

5. In these code chunks, we are taking relevant information from the video we captured. Information such as frames per second, video dimensions, and the total number of frames in the video. In the next cell, we are initializing the Video writer. This is used to create the output file and will be utilized to apply the panoptic segmentation to the video that was captured. The parameters contain information on the video's format type, dimensions, color, and the number of frames per second desired (30). This can be edited to alter the output file, for example making it black and white we would set “isColor” to false.

6. The first cell of this code chunk is used to initialize the predictor. This calls the pre-trained model we would like to use, the weights we would like to use, and our threshold. The next cell is utilized to initialize the video visualizer, this is used to apply the detectron2 to video footage.



7. The first cell contains a function through the video frame by frame. This uses a while loop to read through all the frames and apply our model to predict the contents of the frame. The next cell is used to number the frames in the video to sequentially loop through the video frame by frame. Within this for loop is a “POSE detectron2.png” file that is created. This acts as a test frame to show the results of the detectron2. The final cell is used to close the resources used, this means closing the video, the video writer, and closing all open video capturing windows. The output video containing the panoptically segmented footage will be available in the working directory as “out.mp4”.



8. This is the results of the test frame “POSE detectron2.png”