Name: Nithin Dsouza

Batch ID: 05062021-9AM (weekend)

Topic: Object Detectors, Image Segmentation, Model Optimization and Inference

Required Libraries

```
import numpy as  np
import cv2
import os
```

Get the saved video file as stream

```
file_video_stream = cv2.VideoCapture(os.getcwd() + os.path.sep + "people.mp4")
```

Create a while loop

```
while True:
  get the current frame from video stream
  ret,current_frame = file_video_stream.read()
  use the video current frame instead of image
  img_to_detect = current_frame
  img_height = img_to_detect.shape[0]
  img_width = img_to_detect.shape[1]


   convert to blob to pass into model
  img_blob = cv2.dnn.blobFromImage(img_to_detect, 0.003922, (416, 416), swapRB=True, crop=False)
  recommended by yolo authors, scale factor is 0.003922=1/255, width,height of blob is 320,320
  accepted sizes are 320×320,416×416,608×608. More size means more accuracy but less speed
```

Person as a label Since Problem statement is only detecting person

```
  class_labels = ["person"]
```

Declare List of colors as an array

```
  Green, Blue, Red, cyan, yellow, purple
  Split based on ',' and for every split, change type to int
  convert that to a numpy array to apply color mask to the image numpy array
  class_colors = ["0,255,0","0,0,255","255,0,0","255,255,0","0,255,255"]
  class_colors = [np.array(every_color.split(",")).astype("int") for every_color in class_colors]
  class_colors = np.array(class_colors)
  class_colors = np.tile(class_colors,(16,1))
```

Loading pretrained model

```
  input preprocessed blob into model and pass through the model
  obtain the detection predictions by the model using forward() method
  yolo_model = cv2.dnn.readNetFromDarknet(os.getcwd() + os.path.sep + 'yolov4.cfg',os.getcwd() + os.path.sep + 'yolov4.weights')
```

Get all layers from the yolo network

```
  Loop and find the last layer (output layer) of the yolo network
  yolo_layers = yolo_model.getLayerNames()
```

```
yolo_output_layer = [yolo_layers[yolo_layer[0] - 1] for yolo_layer in yolo_model.getUnconnectedOutLayers()]

 input preprocessed blob into model and pass through the model
yolo_model.setInput(img_blob)
obtain the detection layers by forwarding through till the output layer
obj_detection_layers = yolo_model.forward(yolo_output_layer)
```

NMS Change 1

```
initialization for non-max suppression (NMS)
declare list for [class id], [box center, width & height[], [confidences]
class_ids_list = []
boxes_list = []
confidences_list = []
```

NMS Change 1 END

```
loop over each of the layer outputs
for object_detection_layer in obj_detection_layers:
 loop over the detections
  for object_detection in object_detection_layer:

    obj_detections[1 to 4] => will have the two center points, box width and box height
    obj_detections[5] => will have scores for all objects within bounding box
    all_scores = object_detection[5:]
    predicted_class_id = np.argmax(all_scores)
    prediction_confidence = all_scores[predicted_class_id]

    take only predictions with confidence more than 50%
    if prediction_confidence > 0.50:

      obtain the bounding box co-oridnates for actual image from resized image size
      bounding_box = object_detection[0:4] * np.array([img_width, img_height, img_width, img_height])
      (box_center_x_pt, box_center_y_pt, box_width, box_height) = bounding_box.astype("int")
      start_x_pt = int(box_center_x_pt - (box_width / 2))
      start_y_pt = int(box_center_y_pt - (box_height / 2))
```

NMS Change 2

```
      save class id, start x, y, width & height, confidences in a list for nms processing
      make sure to pass confidence as float and width and height as integers
      class_ids_list.append(predicted_class_id)
      confidences_list.append(float(prediction_confidence))
      boxes_list.append([start_x_pt, start_y_pt, int(box_width), int(box_height)])
```

NMS Change 2 END

NMS Change 3

```
Applying the NMS will return only the selected max value ids while suppressing the non maximum (weak) overlapping bounding boxes

Non-Maxima Suppression confidence set as 0.5 & max_suppression threhold for NMS as 0.4 (adjust and try for better perfomance)
max_value_ids = cv2.dnn.NMSBoxes(boxes_list, confidences_list, 0.5, 0.4)

loop through the final set of detections remaining after NMS and draw bounding box and write text
for max_valueid in max_value_ids:
    max_class_id = max_valueid[0]
    box = boxes_list[max_class_id]
    start_x_pt = box[0]
    start_y_pt = box[1]
    box_width = box[2]
    box_height = box[3]

    get the predicted class id and label
    predicted_class_id = class_ids_list[max_class_id]
    predicted_class_label = class_labels[predicted_class_id]
    prediction_confidence = confidences_list[max_class_id]
```

NMS Change 3 END

```
    obtain the bounding box end co-oridnates
    end_x_pt = start_x_pt + box_width
    end_y_pt = start_y_pt + box_height

    get a random mask color from the numpy array of colors
    box_color = class_colors[predicted_class_id]

    convert the color numpy array as a list and apply to text and box
    box_color = [int(c) for c in box_color]

     print the prediction in console
    predicted_class_label = "{}: {:.2f}%".format(predicted_class_label, prediction_confidence * 100)
    print("predicted object {}".format(predicted_class_label))

    draw rectangle and text in the image
    cv2.rectangle(img_to_detect, (start_x_pt, start_y_pt), (end_x_pt, end_y_pt), box_color, 1)
    cv2.putText(img_to_detect, predicted_class_label, (start_x_pt, start_y_pt-5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, box_color, 1)


cv2.imshow("Detection Output", img_to_detect)
terminate while loop if 'q' key is pressed
if cv2.waitKey(1) & 0xFF == ord('q'):
    break
```

Releasing the stream
Close all opencv windows

```
file_video_stream.release()
cv2.destroyAllWindows()
```