Vitals Extraction Code Documentation

Within this document, we list and describe the various implementations used in the extraction of information from a drivers' license using camera feed. As an overview, there are three main steps in order to detect and extract information from identification from camera feed:

1. **Detect the presence of a monitor.** In order to detect the presence of a vitals monitor, a YOLO Deep Learning Framework will be constructed and trained on the vitals monitor dataset. YOLO was chosen due to its aptitude in processing large data at many frames per second. Furthermore, according to the problem, it is only necessary to detect whether one object is present in the frame, which YOLO excels at.

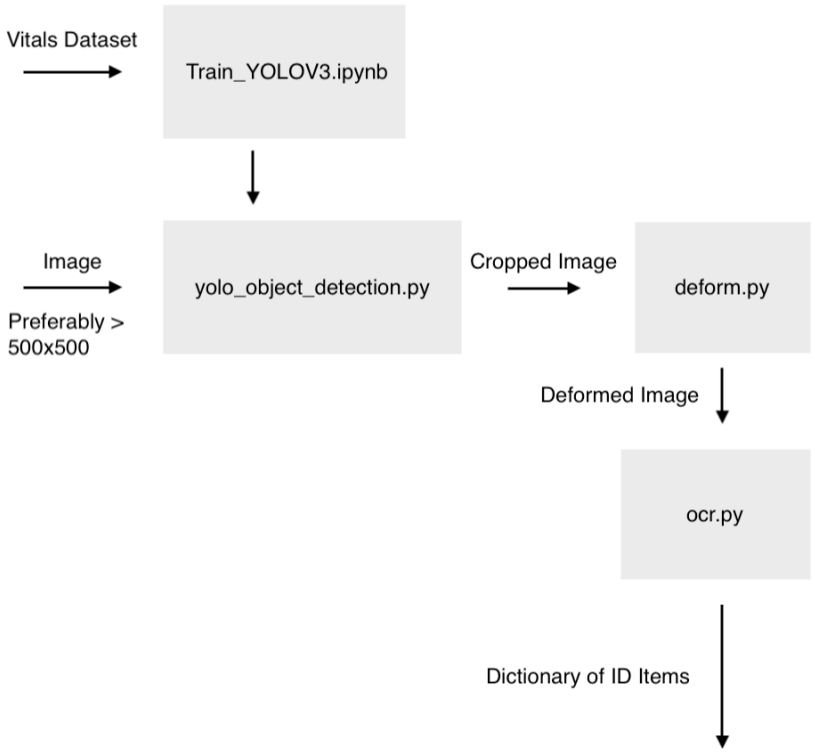
2. **Detect the spatial orientation of a monitor and deform.** To optimize optical character recognition, the identification's image will be determined using Hough Transform Edge Detection and deformed such that it appears head-on with no rotation.

3. **Read and categorize information on the vitals monitor.** Once a monitor is detected, and after some preprocessing, Tesseract OCR in Python will be used to extract text from the license. The location of numbers will then be used to sort the information into desired categories, such as heart rate, blood pressure, and oxygen saturation.

Overall Performance Table:

Step	Target Accuracy (or Loss)	Recorded Accuracy (or Loss)	Target Speed	Recorded Speed
YOLOv3	96%	99%	8ms	30ms
Deformation	<0.01	0.008	2ms	1ms
OCR and Location Parsing	96%	93%	3ms	6ms
Overall	88.8%	89.28%	16ms	37ms

Flow Table of Files



YOLOv3

Description:

- The YOLOv3 architecture and pre-weights were pulled from https://github.com/AlexeyAB/darknet. The implementation found in the files consists of a python notebook to train the architecture in Google Colab (**Train_YoloV3 .ipynb**), and a python file (**monitor_yolo/yolo_object_detection.py)** to run a network on a directory of images. The weights of after 2000 epochs are found as **yolov3_training.weights**, and the YOLOv3 configuration file is found as **yolov3_testing.cfg**.
- After training, the network will draw bounding boxes in input images around the detected driver licenses.

Testing Results:

- Average Loss: ~0.03
- Average IOU: ~0.80-0.95
- Average Accuracy: ~0.99
- Decision Speed: ~30ms
    - This decision speed is slow for video camera use, but can be run with separate threads for smoothness. It may also be possible to evaluate every other frame

rather than each frame without loss of significant data if we wish to run on one thread.

Deformation:

Description:

- The deformation algorithm takes the closely cropped image of the vitals monitor as an input. The images are then color separated in order to isolate the blackness of the vitals monitor that contains the vitals information. These new separated images are preprocessed with Canny Edge Detection and run through Hough Transform to detect the edges of the vitals monitor. The contours are then detected and filtered based off certain characteristics that are indicative of the vital monitor's contour:
    - % Area: 75-100% of area
    - # of Corners: 4-5 corners
    - Aspect ratio: 0.3-0.5

  Once the vital monitor's contour is detected, its corners are obtained, and the monitor is modeled as a rectangle. This rectangle is deformed to a head-on view. The main deform python file is found in **deform/deform.py**.

Testing Protocol and Results:

- Using the manual data augmentation demonstrated in the **Medical Device YOLOV3 Dataset Documentation**, we created custom transforms (and recorded them as the labeled truth), and applied them to a head-on view of the vitals monitor. These labeled images were then fed into the deformation algorithm to calculate a transform. The Loss was calculated as the sum of the squared difference of all entries between the true transform and predicted transform: Loss = $\sum_n$(true-predicted)^2
- Result:
    - Average Loss: ~0.008
    - Decision Speed: ~1ms

OCR

Description:

- The OCR algorithm first preprocesses the Post-YOLO deformed image using the OpenCV functions Binarize, Gaussian Blur, TopHat, BlackHat, and Threshold Local. It then finds contours, which are then labeled as 'Regions of Interest'. The python Tesseract OCR is then run on the regions, and the text that is obtained is returned in a list of text, paired with each text's location. Using this relative location data, we categorize what each number represents, as indicated in the figure below. **ocrv/ocr.py** contains the main OCR algorithm. **ocrv/stackchain** contains a few helpful functions, and **ocrv/test** contains a few representative test images.

Testing Protocol and Results:

- To create the testing dataset, images from the smart glasses camera were fed through the YOLO and deformation algorithms. 200 ‘Regions of interest’ (ROI) from these images were then labeled with their truth in a list. These same post-YOLO deform images are then fed into the OCR algorithm, and the output is compared with truth. The accuracy is determined as the number of ROIs correctly detected and read over the total number of ROIs.
- Result:
    - Average Accuracy: 93%
    - Decision Speed: 6ms
        - It may be best to have a hard limit on how quickly the OCR can be run. Instead of running the check on every frame that contains an ID, perhaps it should be run every three frames.