Drivers License 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 an ID.** In order to detect the presence of identification, a YOLO Deep Learning Framework will be constructed and trained on the driver license 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.

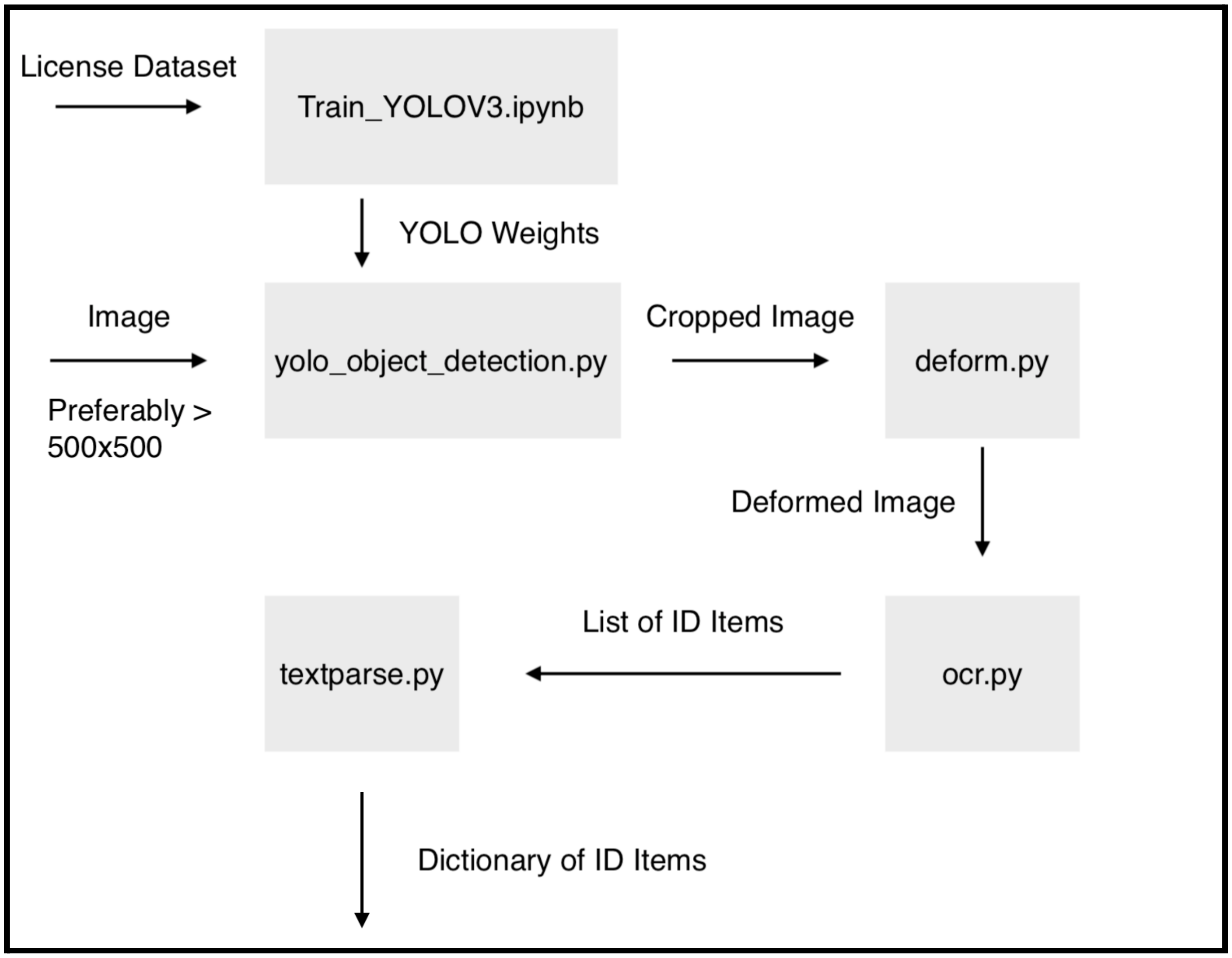
2. **Detect the spatial orientation of an ID and deform.** To optimize optical character recognition, the identification's image will be determined using Hough Transform Edge Detection and deformed such that it appears head-on with no rotation.

3. **Read and categorize information on the ID.** Once an ID is detected, and after some preprocessing, Tesseract OCR in Python will be used to extract text from the license. Generic Text Parsing will then be used to sort the information into desired categories, such as name, birth date, etc.

Overall Performance Table:

Step	Target Accuracy (or Loss)	Recorded Accuracy (or Loss)	Target Speed	Recorded Speed
YOLOv3	96%	99%	8ms	30ms
Deformation	<0.01	0.009	2ms	1ms
OCR	96%	96%	3ms	6ms
Text Parsing	96%	99%	3ms	<1ms
Overall	88.8%	94%	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 (**yolo_custom_detection/yolo_object_detection.py)** to run a network on a directory of images. The weights of after 5500 epochs are found as **yolov3_training.weights**, and the YOLOv3 configuration file is found as **yolov3_testing.cfg**. Earlier weights are also included in **prev_weights**.
- After training, the network will draw bounding boxes in input images around the detected driver licenses.

Testing Results:

- Average Loss: ~0.03

- Average IOU: ~0.85-0.95
- 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.

Deformation:

Description:

- The deformation algorithm takes the closely cropped image of the ID as an input. The images are then preprocessed with Canny Edge Detection to assist Hough Transform in detecting edges. The contours are then detected and filtered based off certain characteristics that are indicative of the ID's contour:
    - % Area: 75-100% of area
    - # of Corners: 4-5 corners
    - Aspect ratio: 0.5-0.6

  Once the ID's contour is detected, its corners are obtained, and the ID 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 **Driver License Dataset Documentation**, we can create custom transforms (and record them as the truth), apply them to flat images of IDs and paste them onto background images. These labeled images are then fed into the deformation algorithm to calculate a transform. The Loss will be calculated as the sum of the squared difference of all entries between the true transform and predicted transform: Loss = $\sum_n(\text{true-predicted})^2$
- Result:
    - Average Loss: ~0.009
    - Decision Speed: ~1ms

OCR

Description:

- The OCR algorithm will first preprocess the Post-YOLO deformed image using the OpenCV functions Binarize, Gaussian Blur, TopHat, BlackHat, and Threshold Local. It will then find contours, which are then labeled as 'Regions of Interest'. The python Tesseract OCR will then be run on the regions, and the text that is obtained will be returned in a list of text. The location of the text will also be recorded. **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 will first be fed through the YOLO and deformation algorithms. 200 'Regions of interest' (AOI) from these images are 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: 96%
    - 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.

Text Parsing

Description:

- The text parsing algorithm uses heuristics and regular expressions to categorize the received list of data from the OCR algorithm:
    - Sex: “F” or “M”
    - Dates:
        - Expiry: Date in Future
        - Birth: Date furthest in past
        - Issued: Date in Between
    - States:‘Nevada’,’California’,etc.
    - Address: Fill up Two - Three lines, ‘St’,’Rd’
    - Height: 4/5(')?-(0/1)?#(“)?
    - Weight: ###
    - Eye: BRO, etc.
    - Hair: BRO, BLK, etc.

  Locations of ambiguous items, such as Eye and Hair are referenced on an ‘Atlas of state IDs’ to categorize them. The main algorithm can be found in the file **textparse/textparse.py.**

Testing Protocol and Results:

- 1000 Custom Input lists of potential texts obtained, such as dates, names, states, sex, series of numbers, height, weight are automatically generated and fed into the algorithm to categorize items. The accuracy is determined as the number of correctly categorized items over the total number of items.
- Results
    - Average Accuracy: ~99%
    - Decision Speed: ~1ms