



HK NOVA: OPERATIONAL GUIDE

CHALLENGE 1: VEHICLE LICENSE PLATE RECOGNITION

PwC's Data-lympics 2019

HUMAN AND MACHINE

28-29 JANUARY 2019



ROHINI BANERJEE
BIKRAM ADITYA GANGWAR

TERENCE LEUNG
ABHISHEK PARYANI

CHALLENGE # 1 - License Plate Recognition

Problem Statement

Data Provided

Task Breakdown

Disclaimer and Limitations of Solution

Solution

Input

Model / Algorithm / Processing

Training

Output

Architecture

YOLOv2 Architecture

Pros of the solution:

Cons of the solution:

Edge Cases

Side Notes

Uniqueness

Deliverables: Files

How to Run the program

Pre-requisites: Packages, Libraries, Frameworks and other boilerplate:

Submission File(s)

References

CHALLENGE # 1 - License Plate Recognition

Problem Statement

Computer Vision: These days, many industries have a desire to understand the activities and interests of consumers for statistical purposes in order to develop marketing strategies and decide resource allocation.

By analysing a video clip, we would like you to develop a real time object detection algorithm which will be able to capture and recognise car registration number. The algorithm should be able to transform car registration number data into digitalised format and import either in database or export file as well as to be able to plot a graph to reflect live situation.

Data Provided

1. Training Data: Hong Kong Car Number Plates – which is a list of sample object images for training purpose.
2. Training Video (HD_video) – which is a sample training video clip as well as for the evaluate your result (485 MB x 2 files)

Task Breakdown

1. Research, study and analyze computer vision algorithms
2. Shortlist computer vision algorithms (open-source only)
3. Convert Video (2 videos : approximately 480mb+ each) to image frames using OpenCV (or similar tools)
4. Train the algorithm to detect multiple objects in an image frame
5. Allow for multiple input sources in the python code (console inputs)

Disclaimer and Limitations of Solution

1. The input files that have been tested were in the format of MP4 (video), which is then converted to jpg image frames by our solution (1 frame for every second).
2. The accuracy / ground truth was tested manually by observing images and video frames from the mp4 that were provided (1_converted.mp4 and 2_converted.mp4)
3. We did not use the train data set provided by PwC team due to lack of time to annotate and create the bounding box of the images provided.
4. Due to reflective surfaces of buildings (glass buildings specifically) and a few cars having tinted glasses, the object count suffers and therefore reduces the accuracy in some image frames.
5. There should be ample lighting to detect the vehicle number plate.
6. The detection is limited to the vehicle (car, bus, and other vehicles) and their respective number plates. Attributes such as the color of the vehicle, type of vehicle and others are not part of the detection algorithm.
7. The number plates used for training (as part of the videos provided) are in English only. Chinese and other language characters/scripts are not detected as part of our algorithm.

8. The arrows drawn on the roads are taken as the starting point of the plate recognition on the images. This has been achieved by cropping the images where the arrow and the divider on the road are located.
9. If a vehicle has two plates, only one plate (usually the one in yellow/white) gets detected accurately.
10. All limitations that apply to YOLOv2, apply to our final solution as well.

Solution

Object Detection is the backbone of many practical applications of computer vision such as autonomous cars, security and surveillance, and many industrial applications.

Our solution uses the following components that are open-source to solve the problem at hand:

1. YOLO V2
2. Darknet

Input

1. Video Files (MP4)

Model / Algorithm / Processing

1. YOLO V2
2. Darknet
3. OpenCV
4. Crop the video (so that only the divider is visible along with the arrow on the roads)
5. Remove Audio from Video files (to reduce the file size of the mp4)

Training

Around 200 images were used to train the model (CNN) for license plate detection.

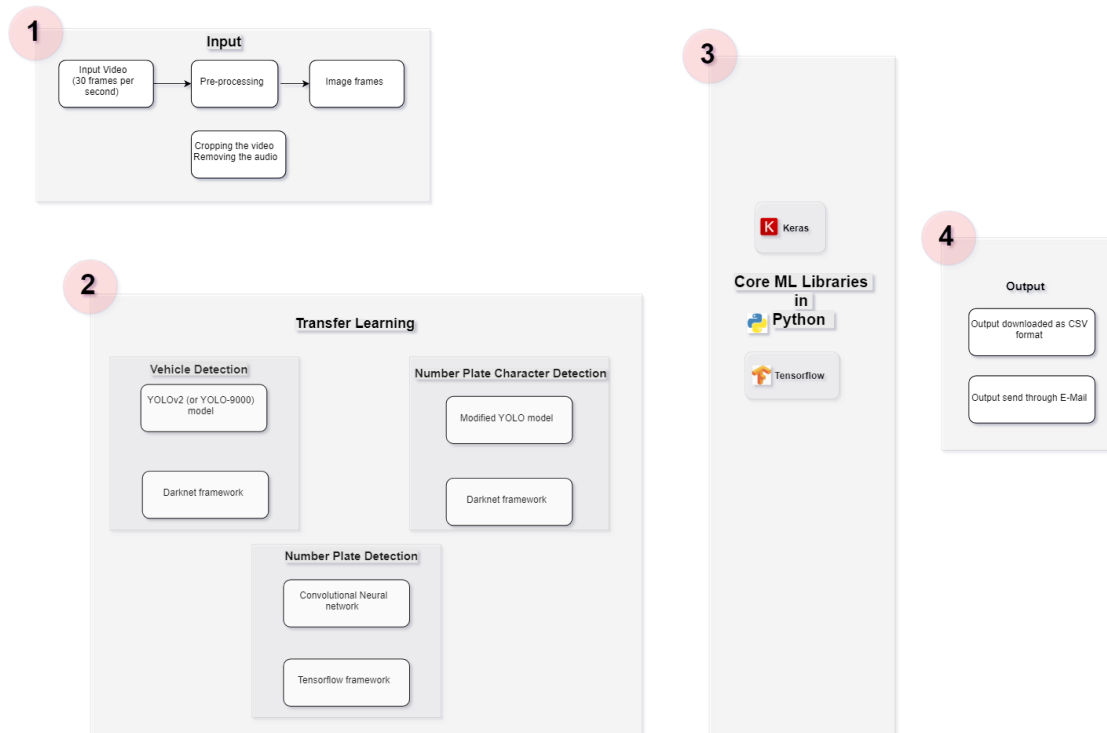
The images were captured from following sources -

- Stanford Cars Dataset
- SSIG SegPlate Dataset
- AOLP (Application-Oriented License Plate) Dataset

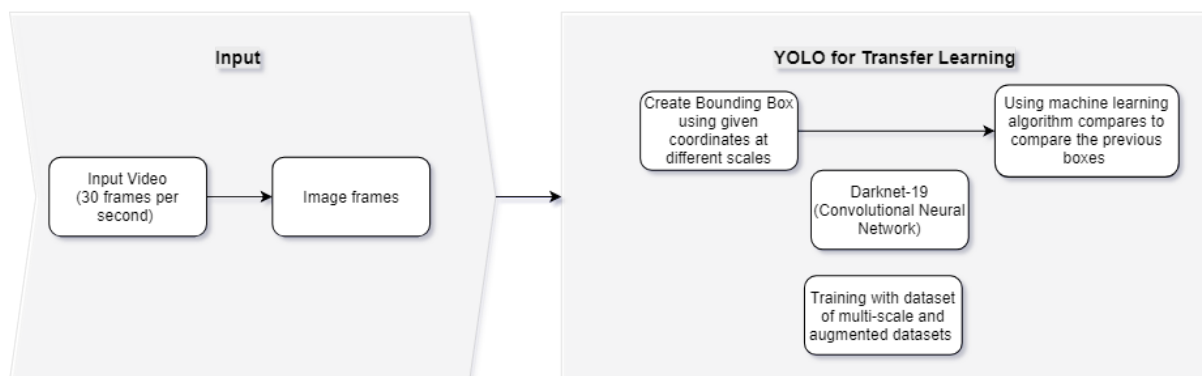
Output

1. CSV file in the format:
 - a. License Plate Number (Hong Kong Plate Number type only)
 - b. Timestamp
 - c. Object Count (one plate to one vehicle only)
2. Email Notification sent to a designated email account
3. Duplicate captures of number plates are removed in post-processing (pandas filter) so as to reduce the computing requirements while performing machine-learning on image/video.

Our Solution's Architecture



YOLOv2 Architecture



Pros of our solution:

1. Provided us with a **81%** accuracy
2. Runs faster than other similar models.
3. Model is trained on full images at multi-scale with no hard negative mining (to avoid false-positive detections).

Cons of the solution:

1. To train the model, a dataset of fewer than 200 images was used and annotated.
2. It only predicts 1 type of class in one sliding window. Hence, it struggles with very small objects and blurry license plate, which is not having a sharp pixel value transition.

Our solution meets the challenge # 1 requirements:

- ☐ Analyze a video clip
- ☐ Develop real-time object detection algorithm
- ☐ Capture and recognize car registration number
- ☐ Transform captured car registration number into digitalized format
- ☐ Import the digitalized format into database
- ☐ Plot a graph to reflect live situation

Besides the above, we set ourselves the following target:

1. The image processing must be performed locally
2. It must work with low quality images
3. It needs to be built using open source technology

Edge Cases (not part of test cases)

1. Vehicle having two number plates. In our model, only one number plate is detected.
2. Vehicle having a number plate that is not clean or is unclear or is not in focus due to camera quality reasons
3. Same number plate on two car in the same video

Test Cases

S.No #	Input	Output	Pass
1	Activate/execute the ALPR python3 file (<run.sh>) (Only run once the pre-requisites are installed)	App captures image or takes an upload from camera / photo library	Pass
2	Test Extraction	Desired output (digitizes the license number plate of HK vehicles)	Pass
3	Database Analysis	Output.CSV file has three columns: timestamp, object, number plate	Pass
4	Email Notification	Sends email to email address specified in input parameter	Pass

Side Notes

- Initially we planned to train the model (image annotations) and to generate the trained model Microsoft ONNX (an open format for machine learning (ML) model) was planned. But due to time constraints we could not complete the tagging of images. Azure Compute Vision (preview) ML service requires that each object have at least 50 images as part of the tags along with a probability and threshold.

- Additionally, the 200 images that were part of the training data set (provided by PWC), we did not have any false positive images to train our model. This would have costed us additional time to tag, annotate, and detect the differences between the ground truth.

Uniqueness

1. No over-fitting was done as part of model creation.
2. Manual feature engineering required to start processing and applying the model for training and testing (cropping, image transformation, and remove audio).
3. Works with blurry images (although this affects the accuracy) for license plate recognition.
4. Not computational expensive - uses one iteration for both recognition and detection of number plates and objects.

Deliverables: Files

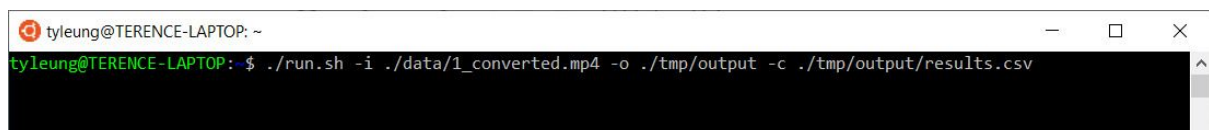
- Script
 - run.sh
- Python Files
 - extract-frames.py
 - vehicle-detection.py
 - license-plate-detection.py
 - license-plate-ocr.py
 - gen-output.py
 - results_formatter.py
 - email_notification.py
- Readme.pdf
- lpr_output.csv
- lpr_video_output.mp4

How to Run the program

In a bash terminal:

```
./run.sh -i <input_filepath> -o <ouput_dir> -c <csv_filepath>
```

where <input_filepath> is the filepath of the input video, <output_dir> is the output directory path, and <csv_filepath> is the output CSV file path.

A screenshot of a terminal window on a Linux system. The window title is 'tyleung@TERENCE-LAPTOP: ~'. The prompt is 'tyleung@TERENCE-LAPTOP: ~\$'. The command entered is './run.sh -i ./data/1_converted.mp4 -o ./tmp/output -c ./tmp/output/results.csv'. The command is highlighted in green. The terminal background is black, and the text is white. There are window control buttons (minimize, maximize, close) in the top right corner.

```
tyleung@TERENCE-LAPTOP: ~  
tyleung@TERENCE-LAPTOP: ~$ ./run.sh -i ./data/1_converted.mp4 -o ./tmp/output -c ./tmp/output/results.csv
```

The code is divided into two sections:

- 1) Exporting the Video into Images (through frames and CV2 package)
- 2) Running YOLOv2's DARKNET architecture and Warped Planar Convolutional Neural Network on the image data and the array data

The idea is that the video classification, given the time frame (less than 24 hours) and the limited resources (hardware and software wise), it is best to convert the video into image frames (still images - png/jpg). We have chosen convert each second into an image frame and output these into the output_dir directory.

The second, and most important point is to only collect / extract features using YOLO and Warped Planar object detection neural network system. The idea is that the features can then be applied (with transformation) to the test image dataset to classify the entire video (converted to image) objects and number plates.

Prerequisites: Packages, Libraries, Frameworks and other boilerplate:

- import os
- import cv2
- import numpy as np
- import sys
- from keras.preprocessing import image
- import keras.applications
- from keras.models import Model
- from keras.applications.resnet50 import ResNet50
- import csv
- from itertools import zip_longest
- from PIL import Image
- from PIL import ImageFilter
- from sklearn.preprocessing import StandardScaler
- from sklearn.decomposition import PCA
- from sklearn.ensemble import IsolationForest
- from sklearn import svm
- from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img
- import shutil
- import smtplib
- import email
- import matplotlib.pyplot

Content	Folder Name	Count
Train_images (converted from videos)	Output_dir	700 images

Submission File(s) - [ALL FILES HERE](#) (DropBox)

1. Source code and a list of library used by your solution (hknova-pwc-source-code.zip)
2. The expected output result of your solution (output.csv and lpr_output1.mp4 and lpr_output2.mp4)
3. Operation guide (current document) (Readme.pdf)

Sent to : data-lympic@hk.pwc.com

References

- 1.
2. <https://www.learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-python-c/>
3. http://openaccess.thecvf.com/content_cvpr_2017/papers/Redmon_YOLO9000_Better_Faster_CVPR_2017_paper.pdf
4. https://docs.opencv.org/3.0-beta/modules/imgcodecs/doc/reading_and_writing_images.html
5. <https://stackoverflow.com/questions/33311153/python-extracting-and-saving-video-frames>
6. <https://stackoverflow.com/questions/39622281/capture-one-frame-from-a-video-file-after-every-10-seconds>
7. <https://stackoverflow.com/questions/51474421/extracting-frames-every-second-of-all-videos-in-folder>
8. <https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html>
9. <https://medium.com/@14prakash/understanding-and-implementing-architectures-of-resnet-and-resnext-for-state-of-the-art-image-cf51669e1624>