Portfolio

Le Hoang Giang

Table of Content

Traffic monitoring

Bachelor’s graduation project

Description

Description

- This is the bachelor's graduation project.
- We aimed to build a smart traffic camera system that can catch three types of violation:
  1. Speeding violation.
  2. Lane violation.
  3. Red-light running violation.
- In this portfolio, except for my contributions, I will just briefly write about other work that I inherited such as: yolov3 architecture, SORT algorithm, camera calibration, hue saturation value color space
- Project link: https://github.com/GiangHLe/traffic_monitoring
- Project duration: 1 year.

Proposed method

Detection and Tracking

- Our goal is to build a high-quality real-time system.
- For detection, we selected **YOLOv3** (https://arxiv.org/abs/1804.02767), **inference at 416 resolution** because it met our conditions:
  - Deep learning object detection model got high accuracy, 55.3 mAP on COCO dataset.
  - High-speed processing, one-stage object detection, 35 fps on Pascal Titan X.
  - Wide implementation for both Tensorflow and Pytorch.
- For Tracking, we selected **Simple Online and Realtime Tracking (SORT)** (https://arxiv.org/abs/1602.00763). SORT is shown as one of the best algorithms for tracking in both **speed and accuracy** at that time (Fig 1).

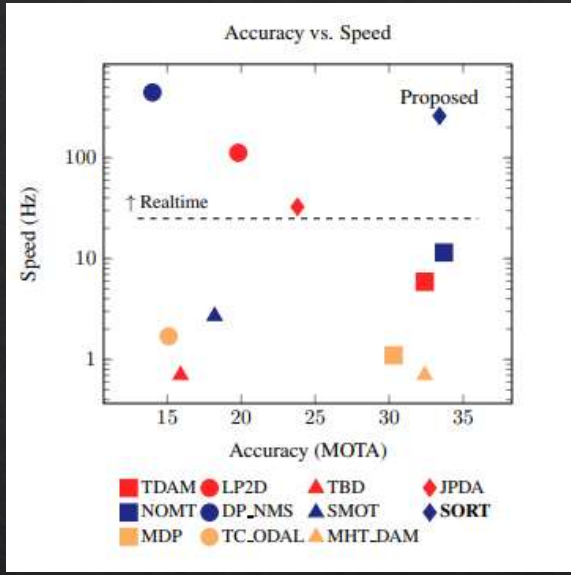
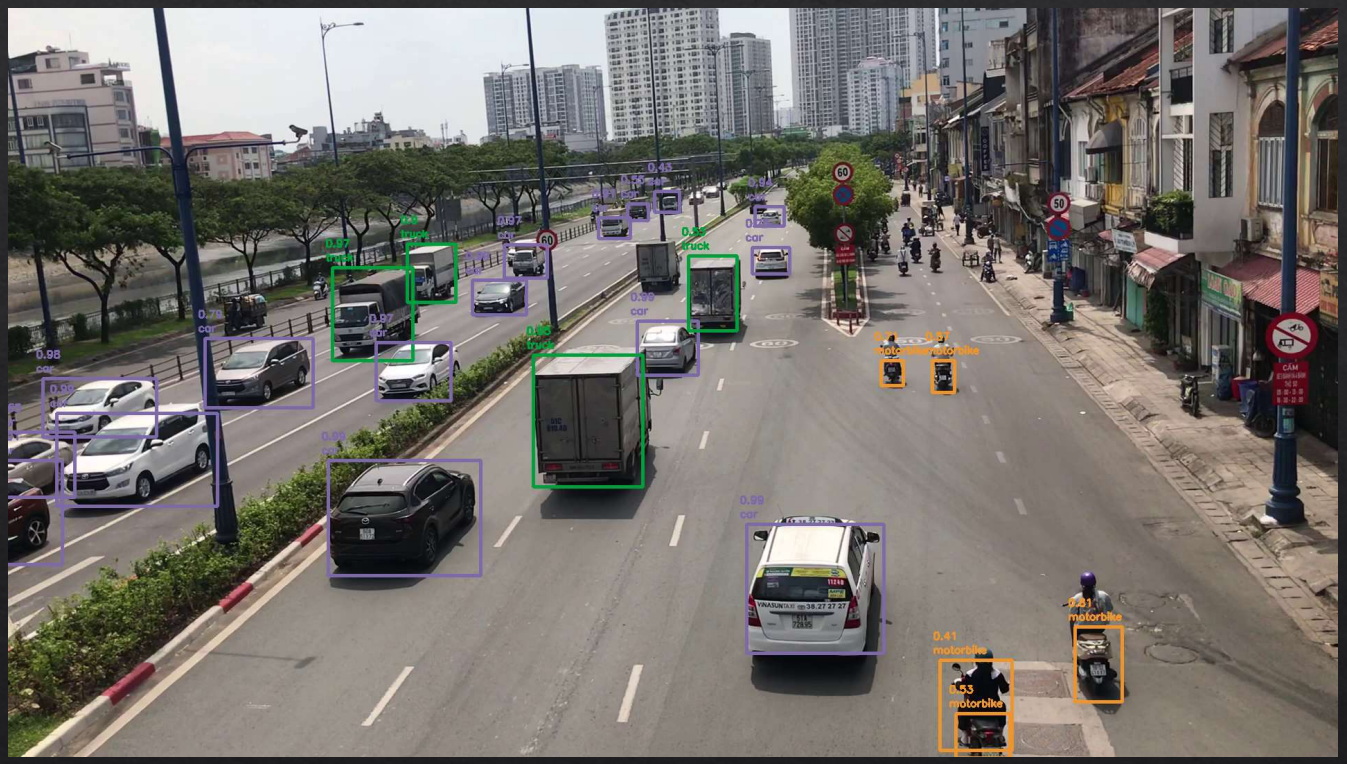


Fig 1. Benchmark performance of SORT in relation to several baseline trackers.

Detection and Tracking results



Speeding violation

- Camera calibration parameters are necessary to project the **distance per pixel on 2D image to 3D real-world units.**
- Traditionally, the camera calibration could collect based on many actions on the camera. However, we want **a flexible method** where we could directly apply it to any camera.
- Therefore, we borrow the work from (Sochor, CVIU17), using **diamond space algorithm** to find **the orthogonal vanishing points**. By then, we could find the camera parameters.

Diamond space

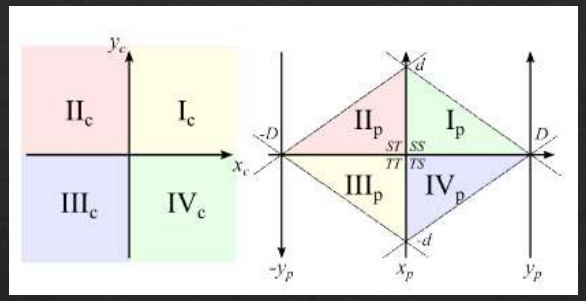


Fig 2. Casterian coordinate to parallel coordinate

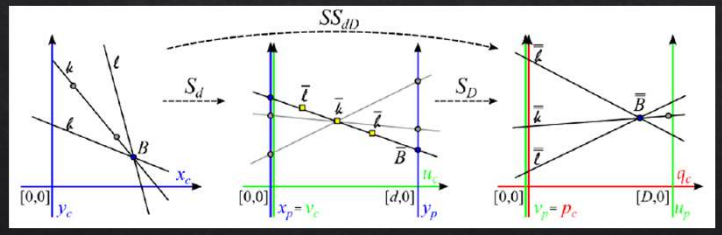


Fig 3. A line in 2D plane is project to point in diamond space

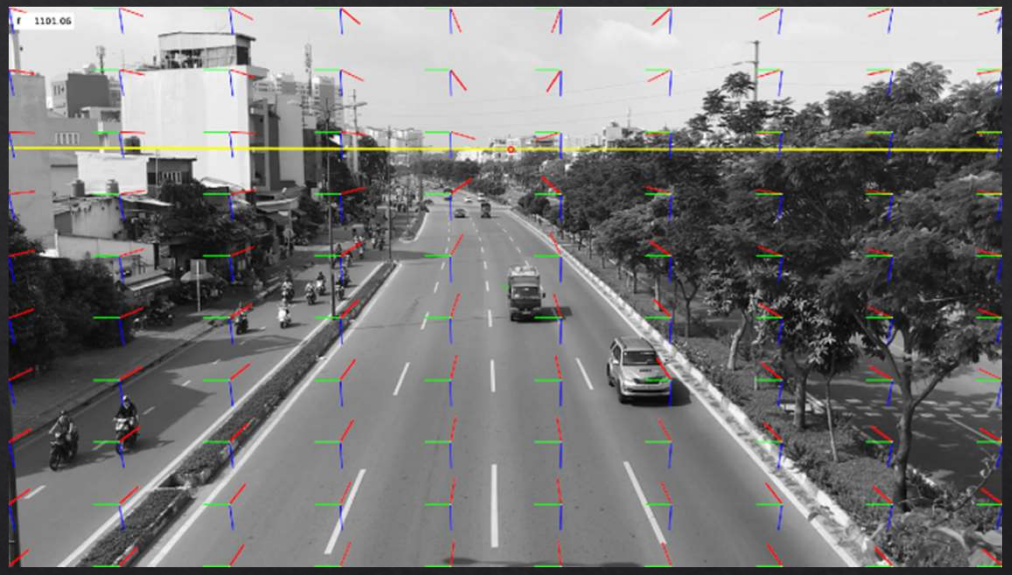


Fig 4. Three orthogonal vanishing points visualize as vector on 2D plane (experiment by Matlab)

2D image plane to real-world units

- With orthogonal vanishing points, we could find the focal length and the road plane, then it is able to project a 2D point in image plane to 3D point in real-world plane.
- However, the distance after projection need to be scaled to become meter units. To archive that, we measure the scale by:

$$\lambda = \frac{D_{real}}{D_{projection}}$$

  - D_{real}: the length of red line in Fig 5 in meter that we measure by tool.
  - $D_{projection}$: the length of red line in Fig 5 that we compute by Euclidean formula from two projecting points.

Fig 5. A frame from our dataset

Speeding violation

- Figure 5 shows the detection results on a vehicle. Call the center position of the bounding box is p, on 2D plane, it is represented by two values on x and y coordinate:
  - $p = [p_x, p_y]$
  - With focal length f from calibration: $\bar{p} = [p_x, p_y, f]$
  - Which could be converted to real-world 3D coordinate $P = [P_x, P_y, P_z]$
- The speed could be measure follow formula:

$$v = \lambda \times \frac{\|P_t - P_{t-\tau}\|}{T}(m/s)$$

  - λ: Scene scale
  - P_t: the position of vehicle in frame t
  - $P_{t-\tau}$: the position of vehicle in frame $t - \tau$
  - T: time different from frame $(t - \tau)$ to frame t. Example: a video 30FPS with $\tau = 5 \rightarrow T = \frac{5}{30} = 0.17s$

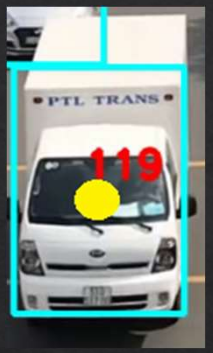


Fig 6. Detection and tracking result on one vehicle

Tricks

- To enhance the tracking quality, we implement the post processing **Non maximum suppression** method after detection.
- While evaluating, we found out that the auto-calibration does not work correctly in all areas of the image. We manually selected an area that works best and then only compute speed on it, normally it will be the center of camera.

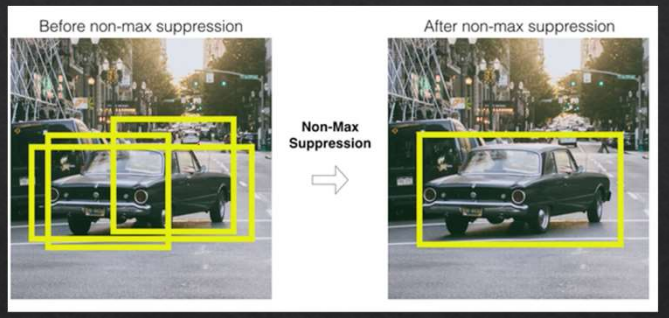


Fig 7. Non-max suppression example

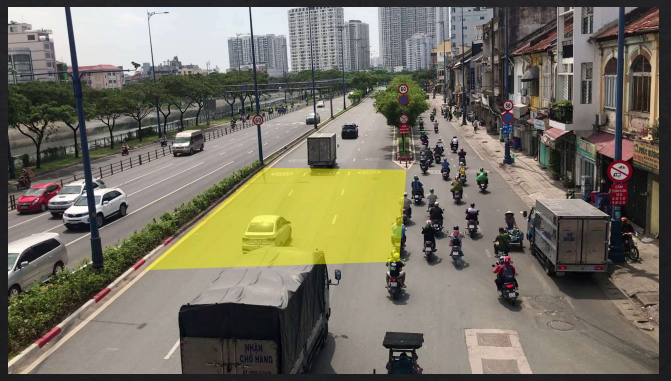


Fig 8. Area that work well for projection

Lane violation

- We divided the road to different lanes by lines as Fig 9.
- A vehicle is violated if it moves from one lane to another.
- Unfortunately, we could not find a way to automatically detect the line.
- Therefore, we manually determine the line equation on image by two random points in image plane on each line. (Fig 10).

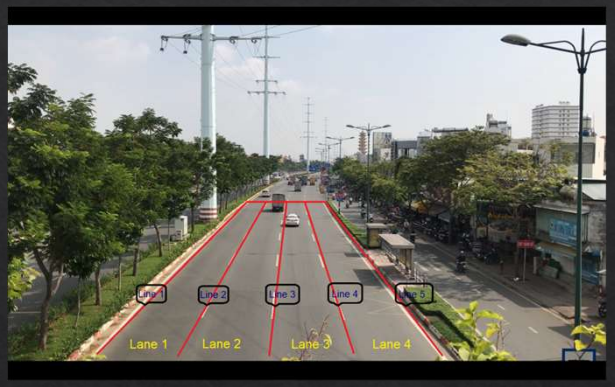


Fig 9. Divide road to lanes

Fig 10. 2 random points and normal vector of one line

Lane violation

- Let call the position of a car that archived from the detection module is P. We could visualize the relation between P with two normal vectors in Fig 11.
- With $P = (x_p, y_p)$ and two lines equation:

$$l_1: a_1 x - y + b_1 = 0$$
$$l_2: a_2 x - y + b_2 = 0$$

- It is possible to know if P is between those lines or not:

$$q(P, l_1, l_2) = (a_1 x_p - y_p + b_1) \times (a_2 x_p - y_p + b_2) \times (\overrightarrow{n_1} \bullet \overrightarrow{n_2})$$

- P is in the middle if $q(P, l_1, l_2) < 0$ and vice versa.

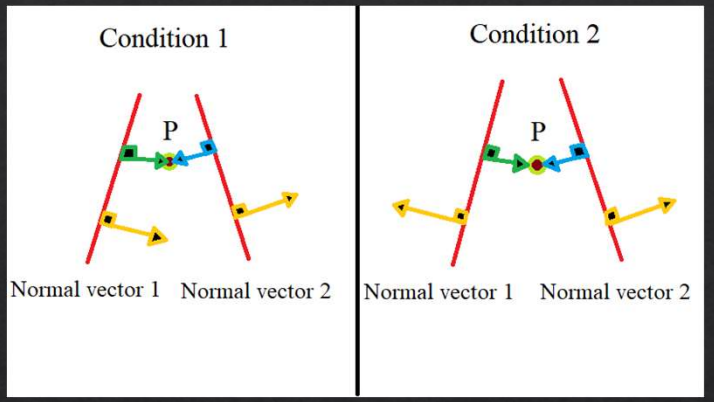


Fig 11. Vehicle position between two line

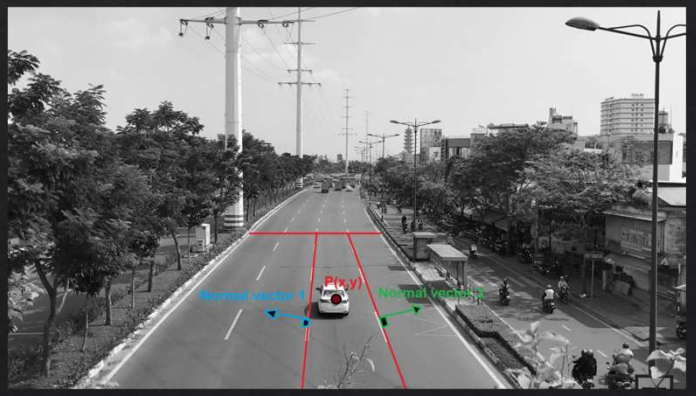


Fig 12. Vehicle position between two line on real image

Red-line running violation

- How we manage the traffic light status?
  - We captured the traffic light position by yolov3, then average the average bounding box positions during 3 first seconds and set it stable. (Fig 3)
  - Because digital colors are complex and difficult to define in the RGB color system. We convert images to a Hue-Saturation-Value color system and decide that red / magenta falls between 301 – 360 degrees.
  - A traffic light box is red if the number of red pixels greater than 1% of total pixels.

Fig 13. Traffic light box

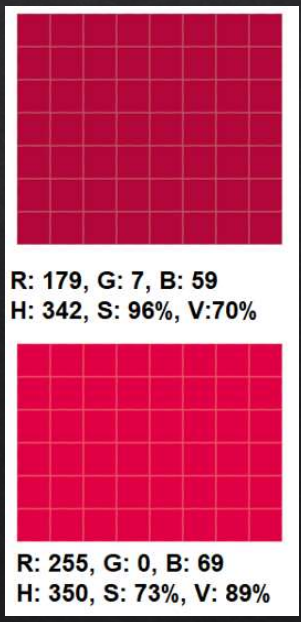


Fig 14. RGB and HSV comparison

Red-line running violation

- Normally, vehicles go from two directions, in order to reduce computational resources, we are only interested in vehicles going from the possible direction of the violation. (Fig 14)
- Using the method in lane violation part, we know which vehicle is above or under the line.
- Since we process all things in 2D and the recording angle is not orthogonal, we decide whether a vehicle crosses the line or not based on the intersection between its bounding box to the decision mask (Fig 15).

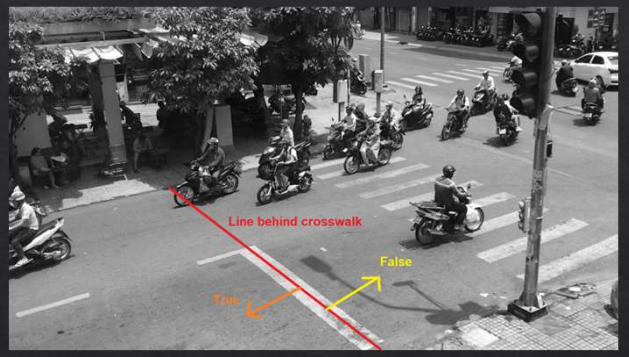


Fig 14. The decision line in red-line running violation

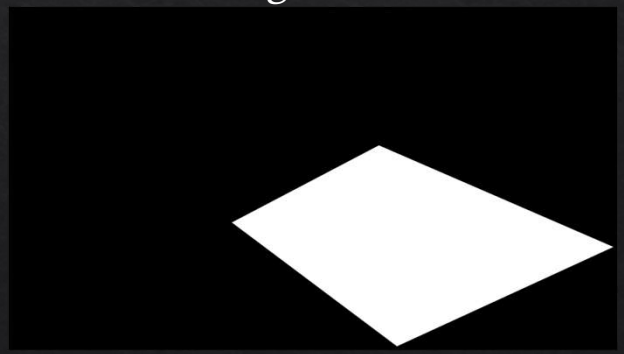


Fig 15. The decision mask in red-line running violation

Red-line running violation

- Figure 13 shows step-by-step how we can decide whether a vehicle is in violation.
- Firstly, the system only operates when the traffic light is red.
- Then the YOLOv3 and SORT return the tracking vehicle ID, if it new, check if it is from the correct side or not, if true, continue.
- Then we compute the intersection area, if the area is bigger or equal to 30, that ID is considered as violated.

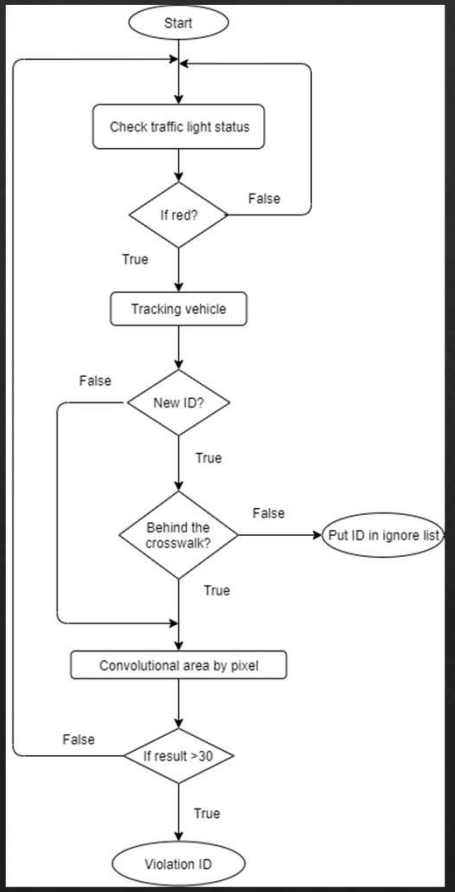


Fig 16. Red-line running violation detection process

Results and Conclusion

Dataset

- There are few datasets and stream links to support this project. Besides that, it is meaningless if this project could operate in my country. Therefore, we collect the data ourselves in Saigon, Vietnam.
- Data is captured by a 12 MP camera on device iPhone series 7 produced by Apple.
- We captured video in the middle of the overpass bridge for the highway dataset and on the rooftop of a coffee shop for the crossroad dataset.

Fig 17. Filming process and tools

Dataset

Fig 18. Highway dataset

Fig 19. Crossroad dataset

Speeding violation result

Fig 20. Speeding violation visualizing

- Yellow area is where the calibration work best.
- The speed limitation is 60km/h, so the vehicle in red is violated and green is vice versa.

Speeding violation result

- Fig 18 shows the report of vehicle speed that our system can catch:
  - ID: vehicle ID
  - FA: the number of frame that vehicle is in yellow area.
  - Time: FA/FPS
  - Speed output: Average speed result from our algorithm
  - Ground truth: Average speed result from measurement tool
- Evaluation metrics: Root mean square error (RMSE).

$$RMSE = \frac{1}{n}\sqrt{\sum_{i=1}^{n}\left(v_{gt,i} - v_{p,i}\right)^2} = 2.65$$

  - n : number of valid ID
  - $v_{gt,i}$: ground truth speed of ID i
  - $v_{p,i}$: predict speed of ID i

ID	FA	Time	Speed output	Ground truth
7	70	2.33	36.13	38.57
191	70	2.33	39.84	38.57
200	70	2.33	38.27	38.57
192	90	3.0	32.52	30.0
251	60	2.0	44.91	45.0
318	80	2.67	34.25	33.75
479	60	2.0	47.68	45.0
610	80	2.67	33.53	33.75
654	70	2.33	44.79	38.57
693	60	2.0	43.33	45.0
734	60	2.0	43.74	45.0
812	60	2.0	45.64	45.0
815	70	2.33	36.77	38.57
867	60	2.0	45.34	45.0
897	70	2.33	41.63	38.57
961	60	2.0	50.05	45.0
945	70	2.33	37.46	38.57
999	70	2.33	41.47	38.57
1054	70	2.33	42.67	38.57
1062	70	2.33	40.04	38.57
1092	60	2.0	39.69	45.0
1102	60	2.0	49.87	45.0
1136	70	2.33	44.06	38.57
1211	60	2.0	43.99	45.0
1234	60	2.0	43.55	45.0
1251	70	2.33	37.51	38.57
1243	100	3.33	26.44	27.0
1281	70	2.33	33.31	38.57
1280	100	3.33	24.28	27.0
1956	70	2.33	38.84	38.57
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
4360	60	2.0	44.92	45.0
4434	60	2.0	46.34	45.0
4458	70	2.33	38.25	38.57
4485	70	2.33	41.38	38.57
4608	60	2.0	46.21	45.0
4642	60	2.0	46.93	45.0
4657	60	2.0	45.24	45.0

Fig 21. Vehicle speed report

Lane violation result

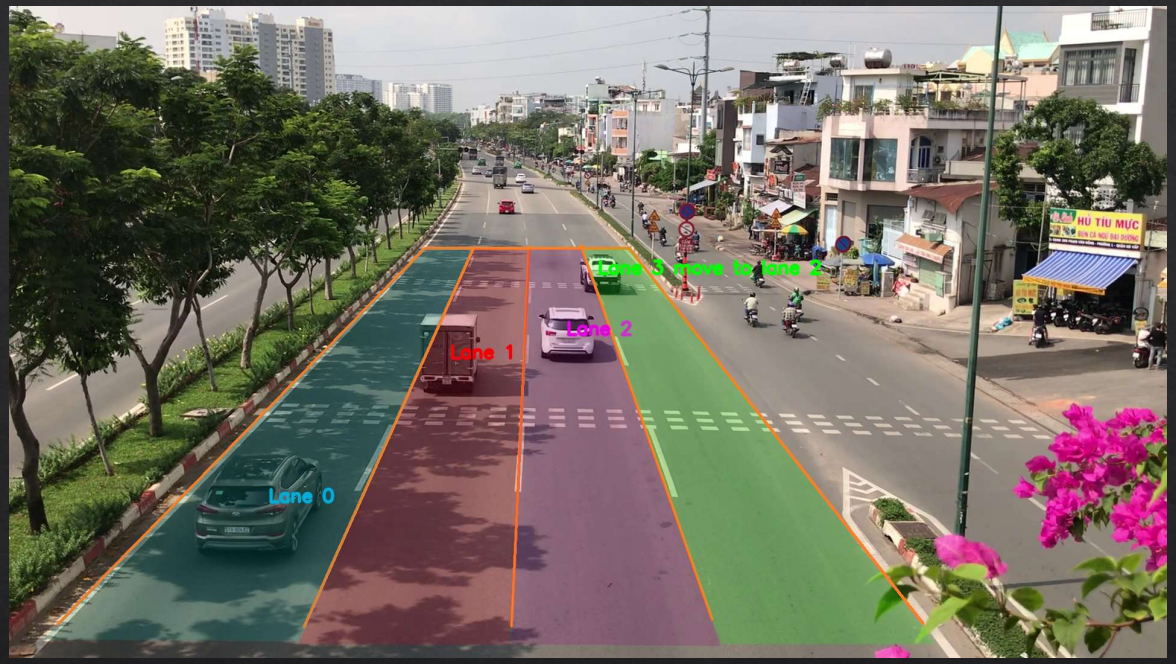


Fig 22. Lane violation visualizing

- This task performance is heavily relied on detection and tracking results, so we did not propose any evaluation metrics for it

Red-line running violation

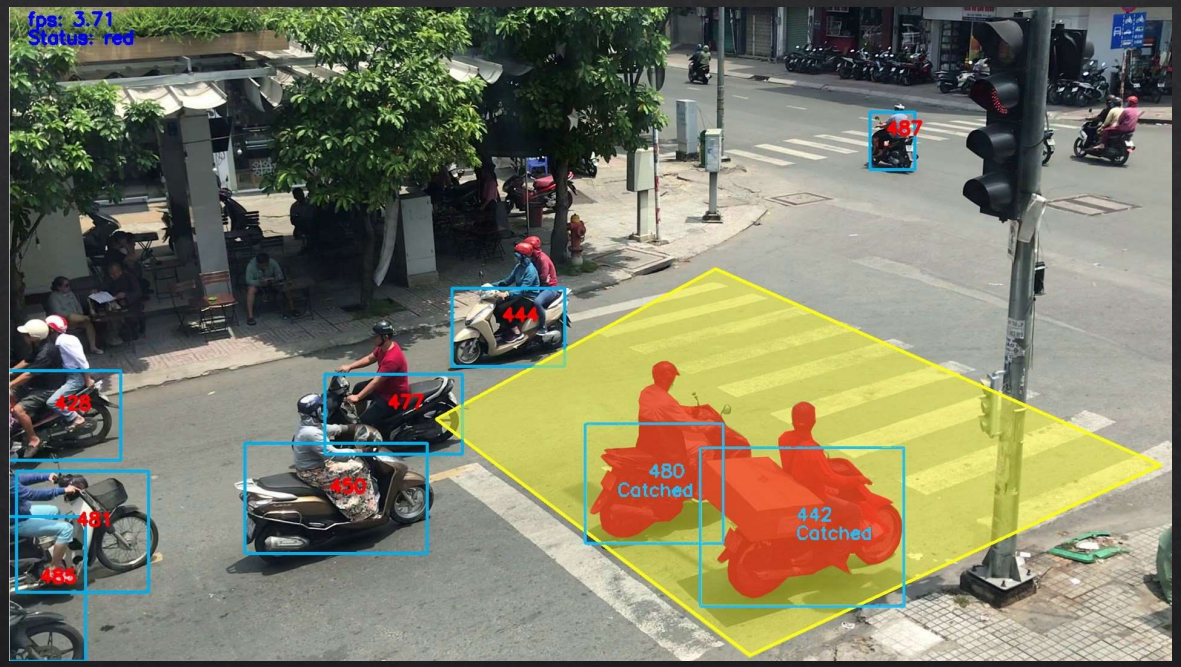


Fig 22. Red-line running violation visualizing

- Yellow area is the decision mask
- Vehicle in red is violated
- Accuracy: 37.5%

Processing speed

- Our hardware details:
  - CPU: Intel® Core™ i7-9700K Processor
  - RAM: 32GB
  - GPU: Geforce RTX 2070
  - Python: 3.5
  - Pytorch: 1.7.0
  - Operation system: Windows 10
- Speeding and lane violation processing speed: 6.5 FPS
- Red-line running violation processing speed: 5.5FPS

Conclusion

- We succeeded in building an automatic method to capture a vehicle's speed and lane. By that, we could decide which vehicle is violated. Besides that, we could also catch the cross-red line violation.
- The performance of the whole project heavily relies on the performance of tracking result.
- Our work has a problem when SORT loses track which causes low accuracy on red-line running violation task. We will have better result of we have better recording angle.

Auto control CPU wafer quality

Industry project

Done when worked in Emage Development

Description

Description

- This project belongs to Emage Development, so I can only show the general solution. Any related images or results in detail cannot be included.
- The client of this project is a factory in Thailand that manufactures wafers for the production of CPUs.
- During the covid-19 situation, the worker cannot come to the workspace, so our work is to build a solution to replace the worker. Our task includes:
  - On the zoom image of the wafer, find the known defect which is described in detail by the client.
  - Automatically control the machine to handle the defect.

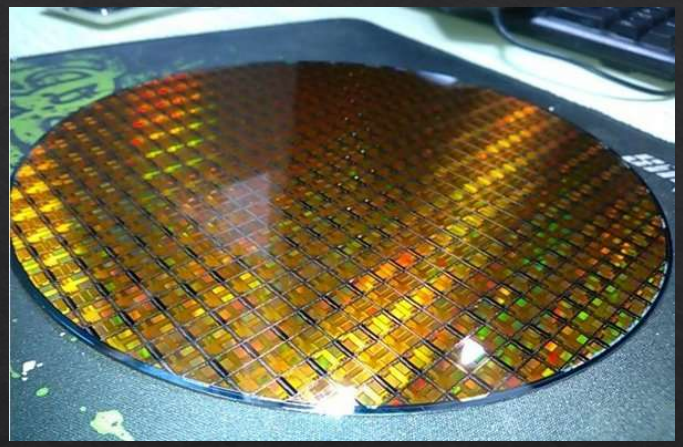


Fig 1. Wafer sample

Finding the defect on the wafer

Overall

- To archive the goal, the things we need to handle the model well:
  - Label
  - Model selection
  - Training strategy
  - Evaluation method
- Project attending time: 6 months.

Labeling

- Our customers provide detailed error descriptions and images dataset only.
- We build a labeling application by C# and then label by describing from the client.
- This process took about 3 months since we need the customer to confirm if our labels are correct or not because the deep learning performance heavily relies on labels.

Models and Evaluation method

- After discussing, we categorized this task as object detection, since the client also required processing speed (3 samples/ second at least) so YOLOv3 is our selected model.
- Normally, the standard evaluation metric of object detection is the mean Average Precision (mAP) which relies on the accuracy of the classification and the bounding box results.
- However, our team and client decided that we will bias the accuracy since it is more important to know which defect it is than its exact position of it.

Dataset

- Problem:
  - Imbalance: only 25% of the dataset has defects → We duplicate the number of defects images
  - Some parts of the image overlap each other → Center crop help improves a lot.
  - Since the label is from us with the description from the documents, we shred the hard label to soft label. Their value of them is then decided by the attribute of the image while we label it.
    - One hot label: [0,1,0] → [0.2, 0.7, 0.1]

Training strategy

- As our experience from real-world data competition as Kaggle and solutions from top developers, we decided to focus on processing the data than modifying the model.
- What we did:
  - Cross-validated is always a good solution to improve the score. However, it's only true for the competition because of the high computation cost (high capacity, high energy, high processing time).
  - We use the multiple image scale while training (256, 512 and 768).
  - Data augmentation, we tried both normal augmentation and random augmentation (which is claim in the paper is better than normal strategy). The normal augment performs much better so we guess the rand augment is better in academic datasets such as ImageNet or COCO.
  - Augments: HorizontalFlip, VerticalFlip, Random(Brightness, Contrast), Cutout.
  - Exponential moving average with decay 0.999

Training strategy

- What we did:
  - Scheduler: WarmpupExponentialDecay.
- Each method is proven to help the model improve since we tried to train two versions, with and without it per each method.

Automatically control the application

Overall

- Our team did not handle this part of the project so we did not know in detail.
- In the nutshell, they use the combination between the object detection model and reinforcement model to control the process.
- Our team then build a TCP/IP python script to send the signal from their result to the server to control the application.
- **I left the company before the project finished to come to Korea for a Master's degree. Therefore, I do not know the result of the project.**

Camera Image Quality Improvement AI Contest

Personal project

Dacon competition

Description

Description

- This is the competition on DACON, hosted by **LG AI Research**. Total price: 10,000,000 won. (Link: https://dacon.io/competitions/official/235746/overview/description).
- While taking photos, normally, the light source causes the glare phenomenon (Fig 1). The target of this task is to reduce the strength of the glare.
- Ranking: 10^{th} /228.
- 10 applicants from the top teams are recommended to join LG (document exemption)
- Evaluation method: PSNR

Fig 1. Problem example

My solution

Model selection

- The task is to interpolate the content of the glare area based on the structure and information of pixels around it.
- A segmentation solution with a U-Net architecture could handle this task well. Therefore, I use a model in that family which is pix2pixHD (https://github.com/NVIDIA/pix2pixHD)
- Applying this model naively to a competitive dataset yields a public score, PSNR: 16.94.

Pre-processing data: split image

- Images from dataset
- In training dataset, we have 2 different resolution, in width x height:
  - 3264 x 2448
  - 1632 x 1224
- The big image needs a bigger model to capture the local features that may cause overfitting.
- I split the image into minor patches size 512 square to let the model capture better local features as Fig 2.
- Using this method help me increase PNSR: 25.16

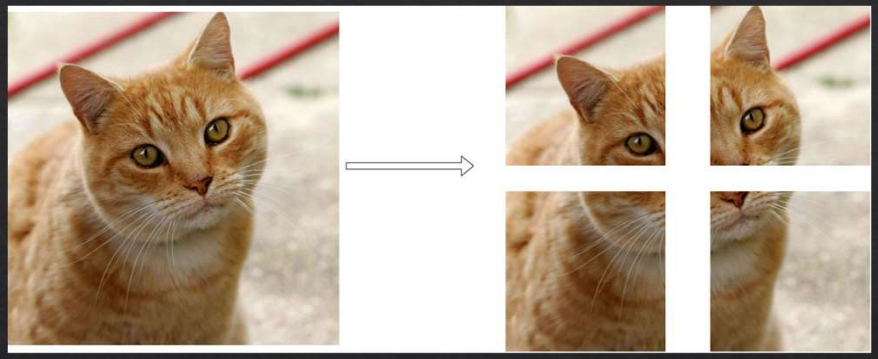


Fig 2. Image splitting

Pre-processing data: split image with overlap

- I realized that splitting images to perfectly fit patches would make the convolutional layer lose the information at the connection border.
- The solution to that problem is that the patches must overlap as Fig 3.
- This method help me reach 27.96.

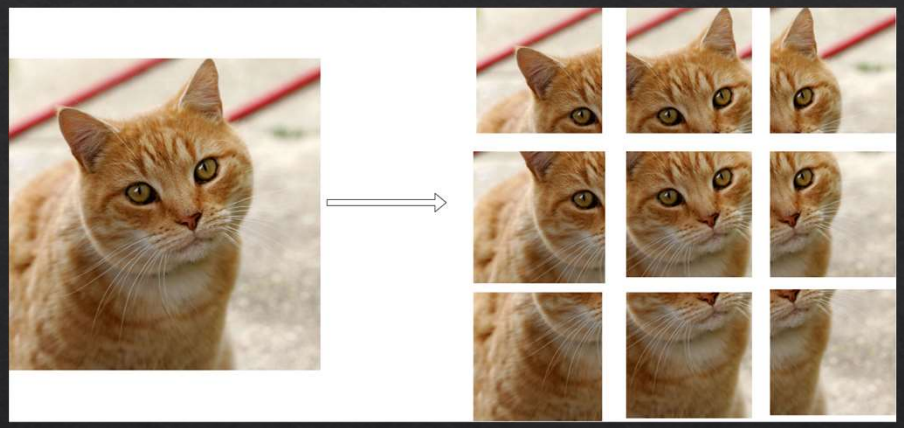


Fig 3. Image splitting with overlap

Pre-processing data: split image overlap with multiscale

- Shrinking image size would help the model catch local features but may cause problems in the global view and structure of the image.
- Fig 4, from left to right, original image, next is glared patches with different sizes from 128 to 64 then 32.
- In my implementation, I trained my model with the combination of 512-patch, 1024-patch, 2048-patch and full image.
- This method help me increase PSNR to 29.93

Fig 4. Multiscale patches

Data Augmentation for Image-translation

- Traditionally, augmentation is implemented for input images. In this task, that could make model transfer input image to the unwanted domain.
- Let call the input image is x and its target is y, the generator is G and the discriminator is D, we will have the fake image is $G(x)$. For each image I decide to apply data augmentation on it, I use the "color" (brightness, contrast and saturation) and translation (moving image up/down of left/right), and call this transformation matrix is T. The objective function of pix2pixHD have three part, I apply the transformation on all of them:
  - $D(x, y) \rightarrow D(T(x), T(y))$
  - $D\big(x, G(x)\big) \rightarrow D(T(x), T(G(x)))$
  - Different between $G(x)$ and y: apply augment for both $G(x) \rightarrow T(G(x))$ and $y \rightarrow T(y)$.
- Surprisingly, this method make my PSNR on public test decrease to 30.225.

Test time augmentation

- In test time, I implement augmentation while inference to diverge the output image.
- Augmentations: horizontal flip, random crop.
- Finally, my result on leaderboard:
  - Public set: 31.09.
  - Private set: 30.78.