DLIP LAB4 : CNN Object Detection

Author : Min-Woong Han

Date : 2023 - 05 - 26

Class : Deep learning image processing

Purpose : Object detection using convolutional neural network (YOLO V5)

---

1. Introduction

This lab aims to employ a Convolutional Neural Network (CNN) called YOLO V5, a well-known model for object detection, and conduct supplementary procedures. Instead of creating a custom dataset, it is intended to utilize a pre-trained model from the official YOLO GitHub repository. The main objective is to utilize this model specifically for car recognition. Ultimately, this lab aims to implement additional processes to identify parked cars within a parking lot, determine the number of parked cars, and calculate the available parking spaces. This report will provide a comprehensive overview of the chosen object recognition model, and the approach in utilizing the model for recognition purposes, and the subsequent post-processing steps undertaken.

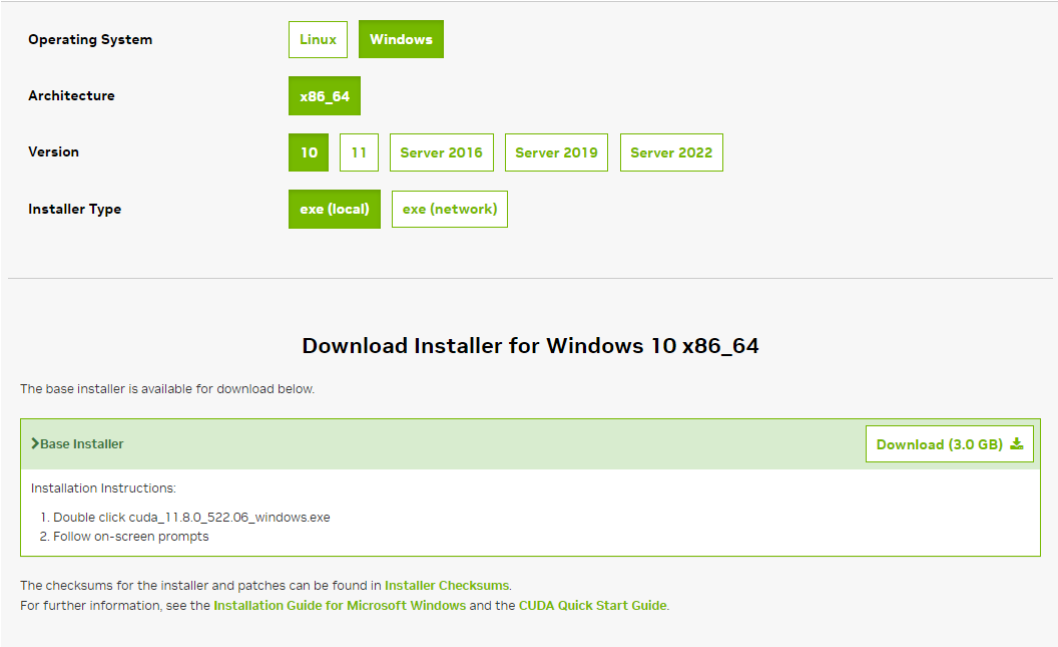
2. Requirement

- Python : 3.9.12
- Cuda : 11.8 version
- cuDNN : 8.6.0 (for Cuda version 11.x)
- Pytorch : 2.0.0 + cu118
- Torchvision : 0.15.0
- YOLO V5
- Anaconda version 3

Installation guide (reference)

2.1. cuda 11.8 installation

- download link : click here to install
- Download process



2.2. cuDNN 8.6.0 installation

- download link : click here to install

- Please refer to this link to get the further process : click here to get further process

2.3. pytorch installation

- Failure to install the appropriate version of Pytorch that is compatible with CUDA can lead to compatibility issues, such as encountering Torch backend errors and other related errors, based on my experience. Therefore, it is crucial to install the suitable versions of Torch and torchvision to ensure proper functionality.
- reference link : click here for reference

- Command

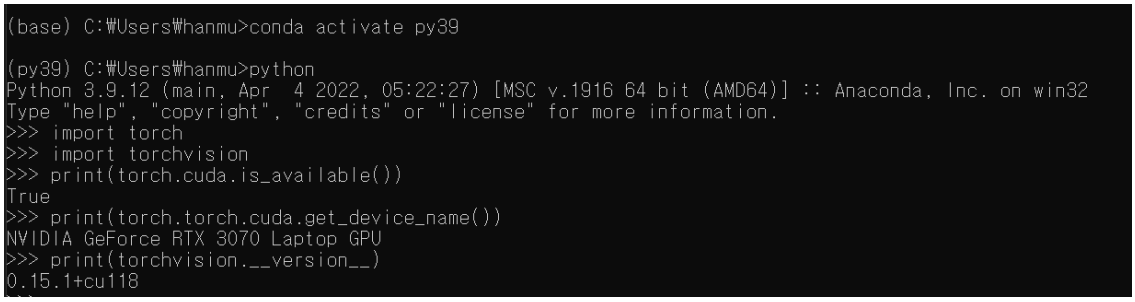
```
conda activate py39 (my environment)

pip install torch==2.0.0+cu118 torchvision==0.15.1+cu118
torchaudio==2.0.1 --index-url https://download.pytorch.org/whl/cu118
```

2.4. check for proper operation

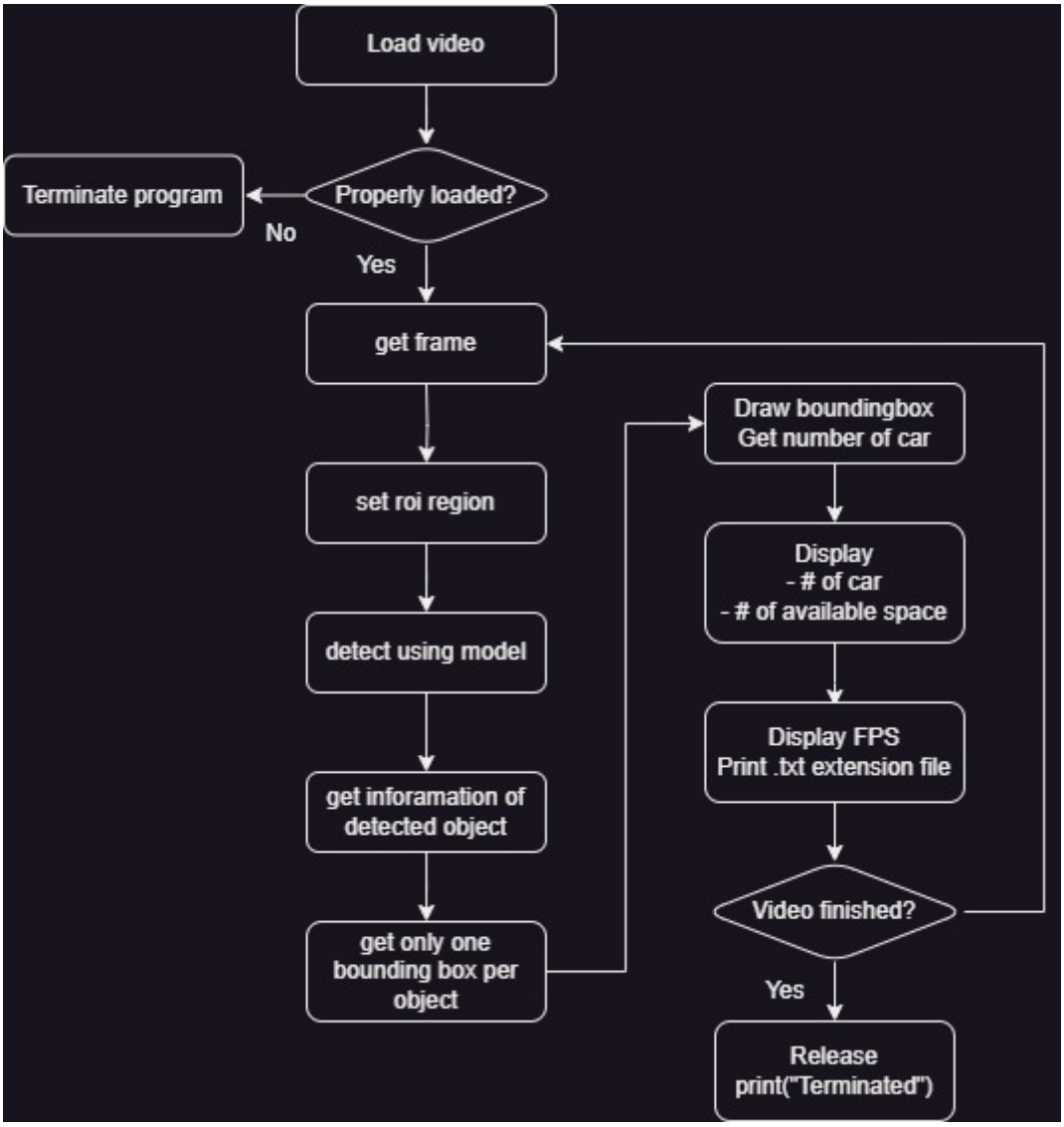
- With proper version compatibility, results below have to be shown.
- Command (anaconda)

```
conda activate py39
python
import torch
import torchvision
print(torch.cuda.is_available())
print(torch.torch.cuda.get_device_name())
print(torchvision.__version__)
```



3. Process

3.1. Flow chart



3.2. Model selection

YOLO V5 github : [click here for reference]

- Firstly, the size of the video is 1280x720. In this lab, while fast video processing is important, accurate video processing takes priority. Therefore, any resizing (downsizing) process of the source video was not performed. After that, it is needed to find an appropriate pre-trained model for object detection. Before using various pt files available on the official YOLO GitHub, it is important to consider the current size of the video and determine the purpose of the task in order to select the suitable object detection model.
- Figure below shows some pre-trained model shown in YOLO v5 github.

Pretrained Checkpoints

Model	size (pixels)	mAP^{val} 50-95	mAP^{val} 50	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.0	45.7	**45**	**6.3**	**0.6**	**1.9**	**4.5**
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7
YOLOv5n6	1280	36.0	54.4	153	8.1	2.1	3.2	4.6
YOLOv5s6	1280	44.8	63.7	385	8.2	3.6	12.6	16.8
YOLOv5m6	1280	51.3	69.3	887	11.1	6.8	35.7	50.0
YOLOv5l6	1280	53.7	71.3	1784	15.8	10.5	76.8	111.4
YOLOv5x6 + TTA	1280 1536	55.0 **55.8**	72.7 **72.7**	3136 -	26.2 -	19.4 -	140.7 -	209.8 -

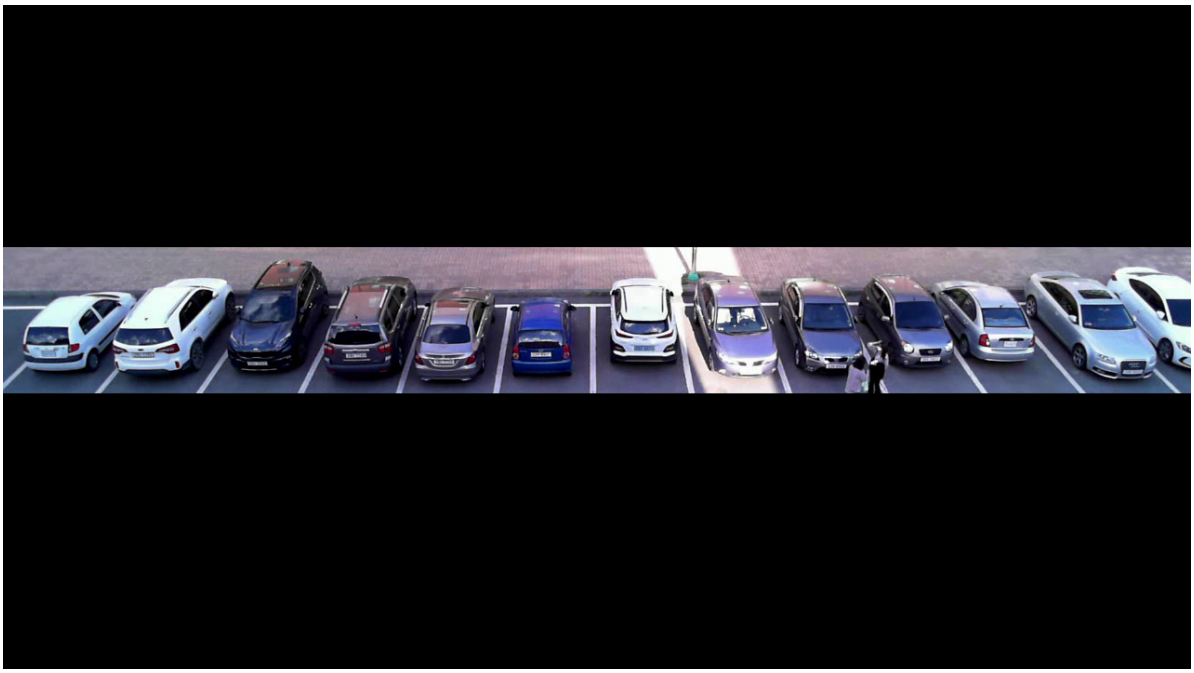
- The parameters mentioned with the model name in the figure above are explained as follows :
  - Size : the resolution of input source
  - mAP (mean average precision) : measurement of precision of object detection model
  - speed : how fast the model can be used in the CPU
  - params : the number of parameters in the model. Model gets complex with more parameters.
  - FLOPs : number of floating point operations required for the model to perform inference.
- Considering the size of source video (1280 x 720), and the capability of CPU embedded in the labtop, the model '**YOLOv5m6**' is selected for object detection.

3.3. code processing

- Since classic image processing techniques like edge detection those are sensitive to noise are not implemented in this lab, proper preprocessing like filtering is not applied.

3.3.1. setting roi region

- Considering that the goal is to count the number of parked cars rather than conducting detection for the entire video, it is necessary to set a region of interest (ROI) for the entire parking area and perform detection within that area for proper processing. The figure below shows the ROI region defined in the source video.



3.3.2. object detection & further process

- Datas stored in the variable `detResult` should be analyzed. The variable was transformed into pandas format. (`detresultPd`)
    - Each row includes below informations.
        - Location of bounding box (xmin, ymin, xmax, ymax)
        - Class of detected object
        - Confidence of detection
- With the datas with pandas format in the iteration loop, bounding box can be drawn for each detected object.
- Considering the occurrence of misclassifications where cars are mistakenly identified as trucks or buses, all cases where the model detects objects as car, truck, or bus classes were taken into account. This ensures that the count of cars includes instances where the model might have misclassified cars as trucks or buses.
- To ensure a clear distinction between vehicles entering or exiting the parking lot, a reliable decision-making process was implemented. The count of cars was increased and bounding boxes were drawn only when the y-coordinate of the center point of the bounding box was smaller than the pixel y-coordinate 386 (the center y-coordinate of the ROI).

- To address the issue of multiple bounding boxes being generated for a single object, leading to an overestimation of the number of cars detected, a solution was implemented during the lab. The YOLO model already incorporates the non-maximum suppression (NMS) technique by default, which ensures that only one bounding box is calculated for each detected object.

  However, in cases where multiple bounding boxes still occur for a single object, causing the detection of more cars than the actual count, a specific handling was applied. The x-coordinate of the center point for each bounding box was calculated. Then, a comparison was made between the x-coordinate of the current bounding box's center point and all previously stored x-coordinates. If the difference was not greater than 50 pixels, it was determined that there were no duplicate bounding boxes, and the count of cars was incremented accordingly.

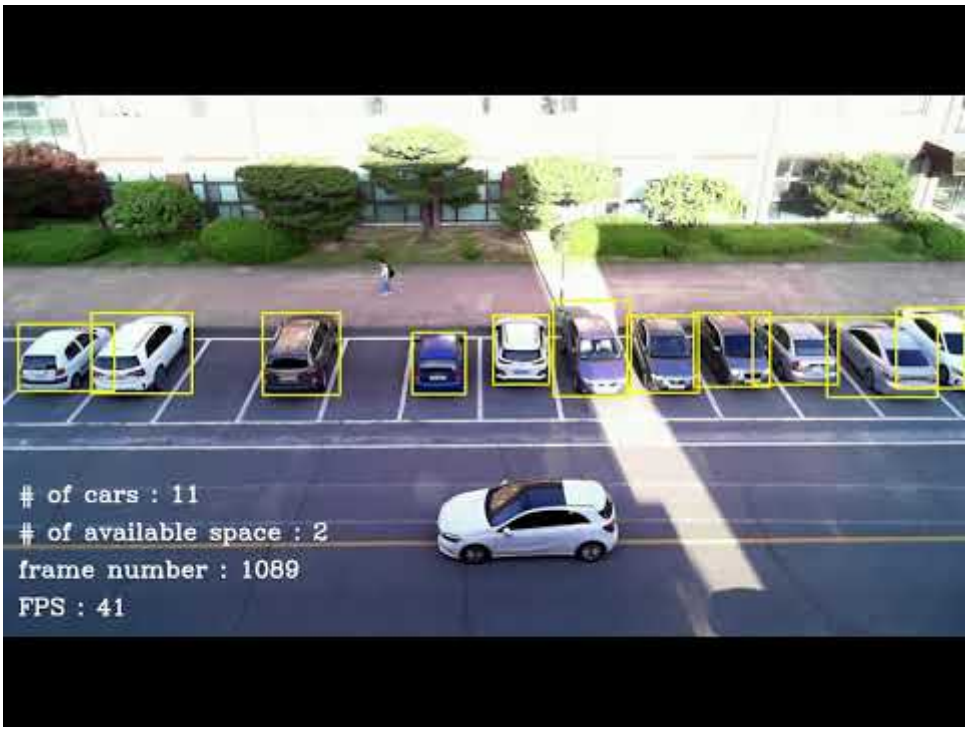
  This additional processing step helped mitigate the issue of duplicate bounding boxes and provided a more accurate estimation of the number of cars detected.

- The following result according to the application of above algorithm is as follows :

4. Result

4.1. Demo video

Video link : [demo video link]



4.2. Model evaluation

Finally, the number of cars recognized by the model is output as a txt file, allowing for performance analysis by comparing with the ground truth file. If the counts in the two files differ, this discrepancy is considered a count error, and the number of errors is incremented while the difference in values between the two files is counted. This process was executed with the MATLAB code provided, and the resulting table is as follows.

```
clear

groundTruth =
load('LAB_Parking_counting_result_answer_student_modified.txt');
model = load('counting_result.txt');
error = 0;

for Idx = 1 : 1501

    if groundTruth(Idx,2) ~= model(Idx,2)

        error = error + 1;
    end
end

numFrames = [error; 1500 - error; (1500 - error)/1500];
```

```
T = table(numFrames, 'RowNames', {'model ~= truth', 'model == truth',
'accuracy [%]'});
```

	1 numFrames
1 model ~= truth	11
2 model == truth	1489
3 accuracy [%]	0.9927

Upon reviewing the results, a difference between the ground truth and model-applied values can be observed in a total of 11 frames. Considering that the number of frames analyzed was 1500, the model demonstrates an accuracy of approximately 99.27%, assuming the ground truth as the correct answer. However, the accuracy may vary depending on how the algorithm is designed to confidently establish the presence of a vehicle in the parking lot as it enters and leaves. In other words, accuracy can change based on the developed algorithm counting the car in the parking space.

5. Dicussion and analysis

The key focus of this lab was to utilize a pre-trained YOLO model for object (car) detection and perform appropriate post-processing. As mentioned throughout the process, traditional image processing techniques that are sensitive to noise were not applied, which means that suitable filtering and blur techniques were not used. Additionally, it was crucial to select an appropriate model considering the video frame size and CPU performance to ensure effective object recognition. Furthermore, after applying the model, it was important to determine the return values and manage the data using a Pandas dataframe. By considering the returned bounding box information, class, confidence, and other factors, the management of parking spaces could be achieved. To address the issue of multiple bounding boxes for a single object (overlap problem), a solution was proposed. While various approaches exist to tackle the overlap problem, utilizing the x-coordinate of the bounding box helped resolve the issue, enabling a more accurate count of objects. Overall, this lab emphasized utilizing a pre-trained YOLO model for object detection, implementing appropriate post-processing techniques, considering model selection based on video frame size and CPU performance, managing data using a Pandas dataframe, and proposing a solution for the overlap problem to ensure an accurate count of objects.

6. Appendix

```
"""
* author    :   Min-Woong Han
* Date      :   2023 - 05 - 26
* Brief     :   DLIP LAB4 Parking management using deep learning
```

```
"""

from module import *

class DETECTION :

    def __init__(self) :

        self.colorYellow  = (0, 255, 255)
        self.colorGreen   = (0, 255, 0)
        self.colorPurple  = (238, 130, 238)
        self.colorWhite   = (255, 255, 255)
        self.colorBlack   = (0, 0, 0)
        self.colorBlue    = (255, 0, 0)
        self.colorRed     = (0, 0, 255)

        self.model        = torch.hub.load('ultralytics/yolov5',
'yolov5m6', pretrained = True)
        self.confThresh   = 0.2
        self.distThresh   = 50

        self.roiRegion    = [[0,260], [1280,260], [1280,410], [0, 410]]
        self.frameNum     = 0

        self.parkingMax   = 13
        self.carCnt       = 0
        self.centerX      = []

        self.timeStart    = 0
        self.timeFinal    = 0
        self.offsetThresh = 20

        self.userFont    = cv.FONT_HERSHEY_COMPLEX


    def getroiFrame(self, frame):

        mask = np.zeros_like(frame)
        cv.fillPoly(mask, np.array([self.roiRegion], dtype=np.int32),
self.colorWhite)

        return cv.bitwise_and(frame, mask)


    def printInfo(self, frame) :

        cv.putText(frame, f'# of cars : {self.carCnt}', (20, 540),
self.userFont, 1, self.colorWhite, 2)
        cv.putText(frame, f'# of available space : {self.parkingMax -
self.carCnt}', (20, 590), self.userFont, 1, self.colorWhite, 2)
```

```
        cv.putText(frame, f'frame number : {self.frameNum}', (20, 640),
self.userFont, 1, self.colorWhite, 2)
        cv.putText(frame, f'FPS : {round(1/(self.timeFinal -
self.timeStart))}', (20, 690), self.userFont, 1, self.colorWhite, 2)



    """
        - All process is handled in method 'mainProcess'
    """

    def mainProcess(self):

        cap = cv.VideoCapture('testVideo.avi')

        fourcc   = cv.VideoWriter_fourcc(*'XVID')
        outVideo = cv.VideoWriter('DLIP_LAB_PARKING_VIDEO_21800772_한민
웅.avi', fourcc, 20.0, (1280, 720))

        f = open("counting_result.txt",'w')

        if cap.isOpened() == False :
            print("Video is not loaded ...!")
            return

        while(cap.isOpened()):

            self.timeStart = time.time()

            ret, frame = cap.read()

            if not ret :
                print("Frame number is over ...!")
                break

            roiFrame = self.getroiFrame(frame)

            detResult = self.model(roiFrame)

            detresultPd = detResult.pandas()

            objectLen = len(detresultPd.xyxy[0])

            for Idx in range(objectLen) :

                xMin = round(detresultPd.xyxy[0].xmin[Idx])
                xMax = round(detresultPd.xyxy[0].xmax[Idx])
                yMin = round(detresultPd.xyxy[0].ymin[Idx])
                yMax = round(detresultPd.xyxy[0].ymax[Idx])

                if detresultPd.xyxy[0].confidence[Idx] > self.confThresh :
```

```
                    # Car, Truck, Bus ==> Car class
                    if detresultPd.xyxy[0].name[Idx] == "car" or
detresultPd.xyxy[0].name[Idx] == "truck" or detresultPd.xyxy[0].name[Idx]
== "bus" :

                        xCen = (xMin + xMax) / 2

                        if any([abs(xCen - prevcenX) < self.distThresh for
prevcenX in self.centerX]):
                            continue

                        self.centerX.append(xCen)

                        if (yMin + yMax)/2 < 386 :

                            self.carCnt += 1
                            cv.rectangle(frame, (xMin, yMin), (xMax, yMax),
self.colorYellow, 2)

            self.timeFinal = time.time()

            self.printInfo(frame)

            f.write(f"{self.frameNum} {self.carCnt}\n")

            self.centerX = []
            self.carCnt = 0
            self.frameNum += 1

            outVideo.write(frame)
            cv.imshow('Frame', frame)

            if cv.waitKey(1) == 27 :
                break

            if self.frameNum > 2500 :

                print("Frame number is overed ...!")
                break

        f.close()
        outVideo.release()
        cap.release()
        cv.destroyAllWindows()



if __name__ == "__main__" :

    detection = DETECTION()
```

```
    detection.mainProcess()
```