Every one can be A Police man

Private Police system on Highway
using Image Segmentation

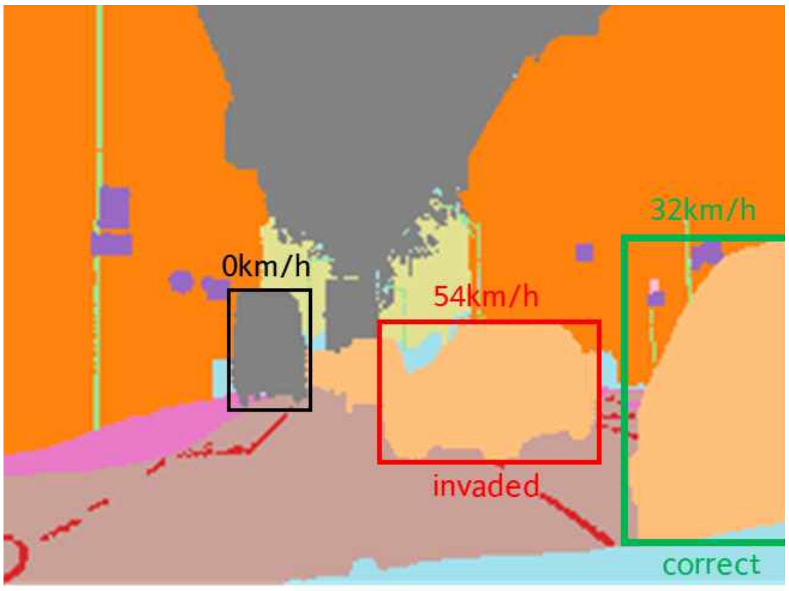
2020-2 DATA&AI Team 5
Sung Jae yeong / bui ngoc han

<Summary>

Speeding is the main tort on the road. It triggers big accident, especially on the highway. To prevent this, there is a camera to catch the speeding cars. But, we were wonder about the efficiency about the current system. It only catches in the specific section. There is also section crackdown system, but it calculates the speed by average so that it couldn't catch the momentary speeding.

So, we suggest the live catching system which measure in individual's vehicle using image segmentation. It is based on the representative learning model of Fully Convolutional Networks, U-Net. Also, we will talk about line invasion problem too. In addition, we will talk about the partition learning which we failed in process, but one way to improve the quality of segmentation.

Every application in this paper are based on fastai deep learning forum 2019's notebook. We also use the opencv and some mathematical calculation.



pic0 the sketch of the application

<Contents>

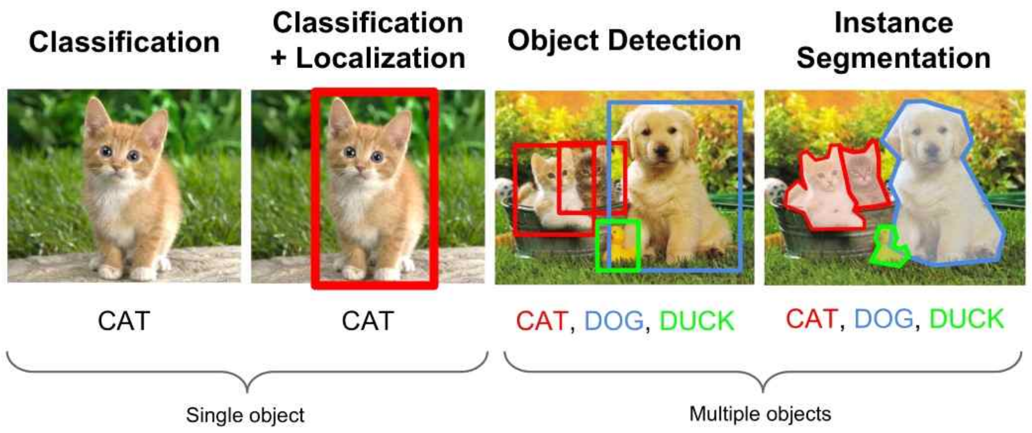
Part1. Reviewing & Labeling

Our project is based on 2019 fastai forum's lesson. We will focus on Lesson3's image segmentation problem. So, in this part, we will wrap up the Lesson 3's notebook and using it to labeling the source video which we will measure the speed and find lines.

A. Reviewing

1. Image segmentation : definition and models

In computer vision, there are three main problems. Classification, Localization/Detection, Segmentation. As you see in the pic1 images, segmentation is detecting the boundaries of multiple objects. So we can distinguish cat and dog or such things in specific boundaries.

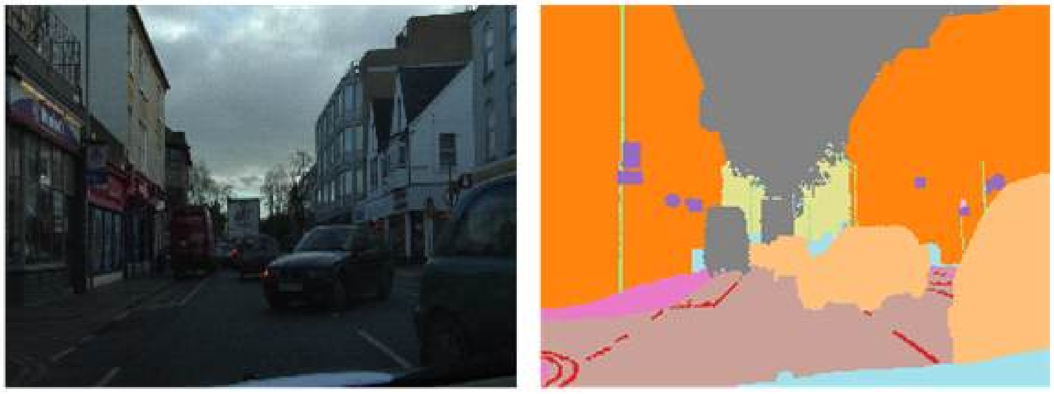


pic1 classification, localization/detection, segmentation

In this paper, we will focus on segmentation problem. The representative models are Fully Connected Network, SegNet, DeepLab. We will use one of the Fully Connected Network, U-Net.

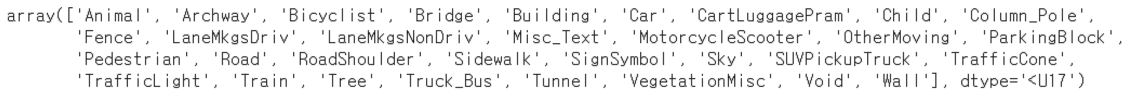
2. Reviewing notebook : with fastai's Lesson3-camvid notebook

In this part, we will wrap up the notebook in short way. For learning the model, we should prepare two kinds of images. One for original, the other for the labeled image. As you can see in the pic2 images, right one is already segmented. This is the ground truth image which is the corresponding answer image to the original one.



pic2 original image(left) and label image(right)

There are 32 labels we can detect. Each label is masked in grayscale value which is the same with the index of label (animal for 0, archway for 1..). Later, we will only use car(label 5) and landMkgsDriv(label 10).



```
array(['Animal', 'Archway', 'Bicyclist', 'Bridge', 'Building', 'Car', 'CartLuggagePram', 'Child', 'Column_Pole',
       'Fence', 'LaneMkgsDriv', 'LaneMkgsNonDriv', 'Misc_Text', 'MotorcycleScooter', 'OtherMoving', 'ParkingBlock',
       'Pedestrian', 'Road', 'RoadShoulder', 'Sidewalk', 'SignSymbol', 'Sky', 'SUVPickupTruck', 'TrafficCone',
       'TrafficLight', 'Train', 'Tree', 'Truck_Bus', 'Tunnel', 'VegetationMisc', 'Void', 'Wall'], dtype='<U17')
```

pic3 32 labels in camvid model

After preparing dataset and labels, we creates model and learn with the dataset. After learning, we can get the prediction image by insert image which we want to change it to segmented image. Labeling is that process.

B. Labeling

1. Extract frames : extracting frames from video using Opencv

Before getting segmented image, we have to prepare bunch of images to convert. But the source formation is video. So we have to extract image frames from the video. We used Opencv method to do it. Below is the source code for extracting using Pycharm editor.

```
import cv2

#https://www.youtube.com/watch?v=wculEdHXd5g
cap=cv2.VideoCapture('video.mp4')
ret, frame = cap.read()
```

```
count=0

while(ret):
      cv2.imwrite("frame%d.png"%count,frame)
      ret, frame = cap.read()
      count+=1
```

table1 Extracting frames from video

The length of source video was 10 seconds, and we got 327 images.

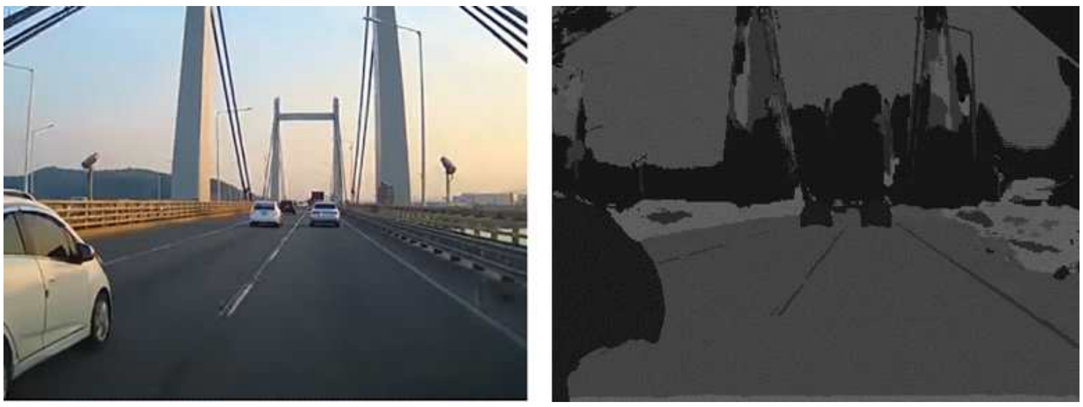
2. Labeling : labeling the prediction on video frames

After extracting, we have to insert 327 images into model which we made in camvid notebook. Below is the source code on Google colab.

```
for i in range(0,327):
   img = open_image(path/f'frame{i}.png')
   prediction = learn.predict(img)
   prediction[0] .save (path2/f'label{i}.png')
```

table2 Applying Prediction to prepared frames

Prediction[0] is corresponding to the segmented image. We saved the image as 'label{number}.png'. After labeling, we can get the 327 segmented images. pic4 is the one of them.



pic4 original frame(left) and correspoding label image(right)

As you see in the label image, the quality of segmentation is low. Fortunately, cars and the lanes are detected well, but the other labels doesn't. We guessed that it is because of the different environment in the

Korea and the Western road. So we tried to make label image which consists Korea environment, but it was difficult. So it should be improved later, but we will pass this problem in this paper.

Part2. Application

There was some problem in quality of label, but we successfully prepared original images by extracting frames from the video. Also prepared the correspoding label images by applying the camvid notebook to frame images.

Now, we will solve two main problems, one for the measuring speed and one for finding lanes. It consists some mathematical calculation. So, we will introduce the way to solve each problem first, and then we will introduce the source codes.

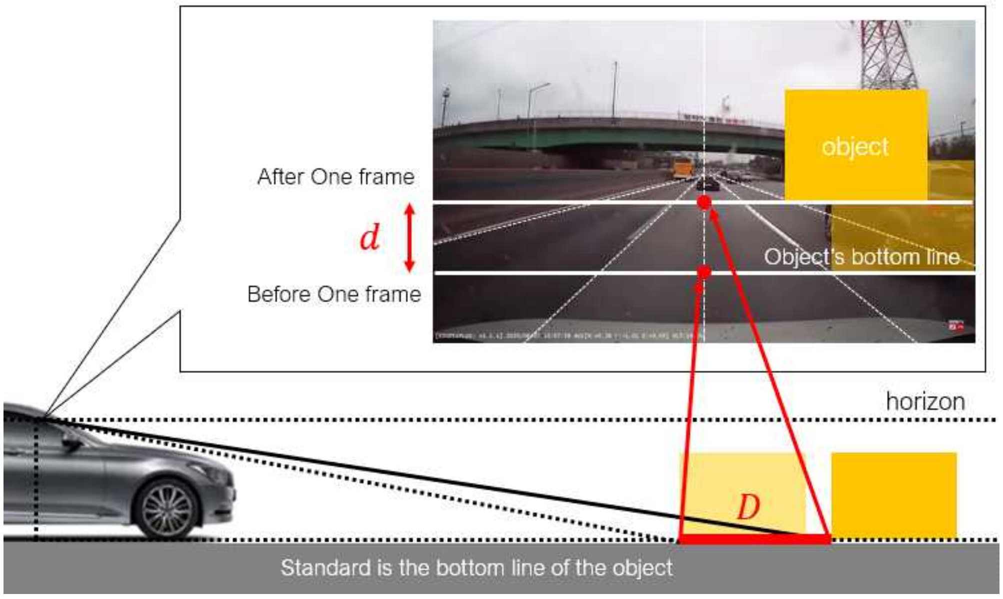
A. Measure speed

When we find object's velocity, we have to measure distance and corresponding duration. For duration, we define it as frame's duration. We extracted 327 images from video which is 10 seconds. So we can calculate the 1 frame's duration. After that, we will calculate the distance(D) by each continuous frames. Also, camera object is moving, so we have to add the self velocity to relative one. So, velocity would be below equation.

$$\begin{aligned} v &= v_{rel} + v_{cam} \\ &= \frac{D}{10s/327} + v_{cam} \end{aligned}$$

1. Perceptive calculation : measure distance

As you see on the above equation, we have to find the real distance from the plane images. Because of perceptive, the real distance is different from the screen distance which we can see. So, we made some supposition, and then created equation about real distance(D) and screen distance($\frac{d}{H}$, it should be the ratio between screen's height and the screen distance) (pic5).



pic5 relation between real distance D and screen distance d

We can guess that the real distance D has two factor. One is the screen distance and the other is the position of the bottom line of the object. Because of perceptive, the same distance is treated different in the back and front of the images. So the position of the bottom line is corresponding to it, and we will treat it as the distance from the horizon line.

So, we have to find the equation of D which affected by two factors. Distance for $\frac{d}{H}$, and the position for $\frac{y}{H}$ (Both parameters should be the ratio about the screen's half height). To define the equation, we made some suppositions below here.

1. Horizon line's position is half of the height of image.

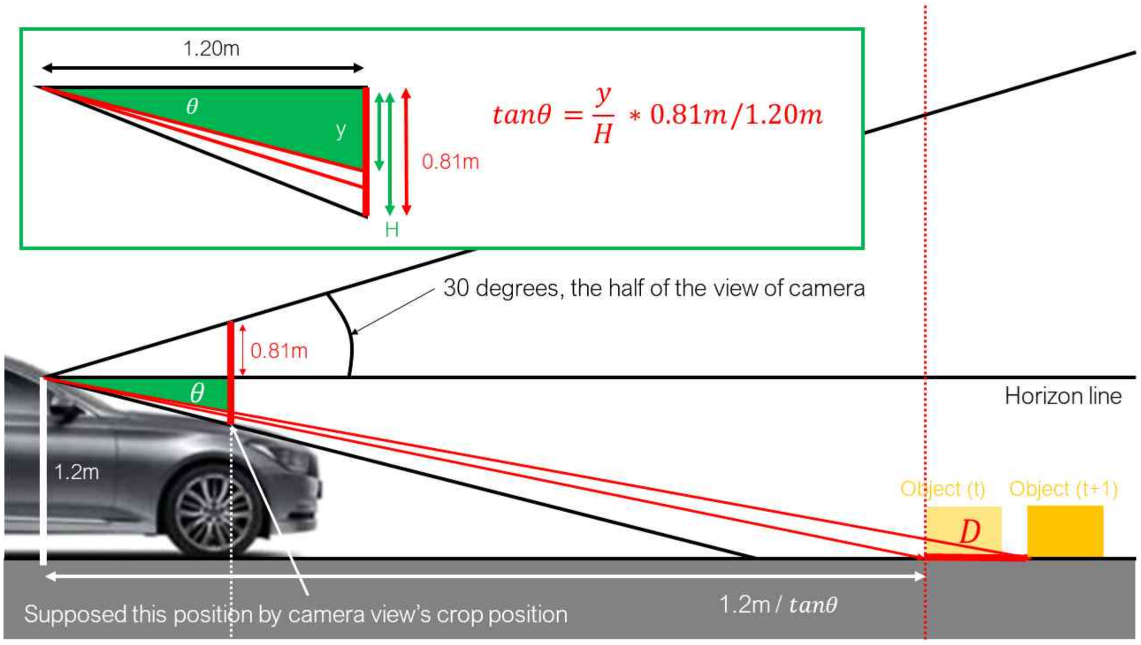
2. Model of car is Hyundai GENESIS G80. So distance standards of car's part are based on this model.

3. Camera has 60 degree of angle of view.

4. Roads are parallel to the camera's horizontal line. (It doesn't matter unless the road's horizontal formation is curve)

5. Camera's velocity is supposed as constant value. So after measuring relative velocity, we will add simple value to it (60km/h)

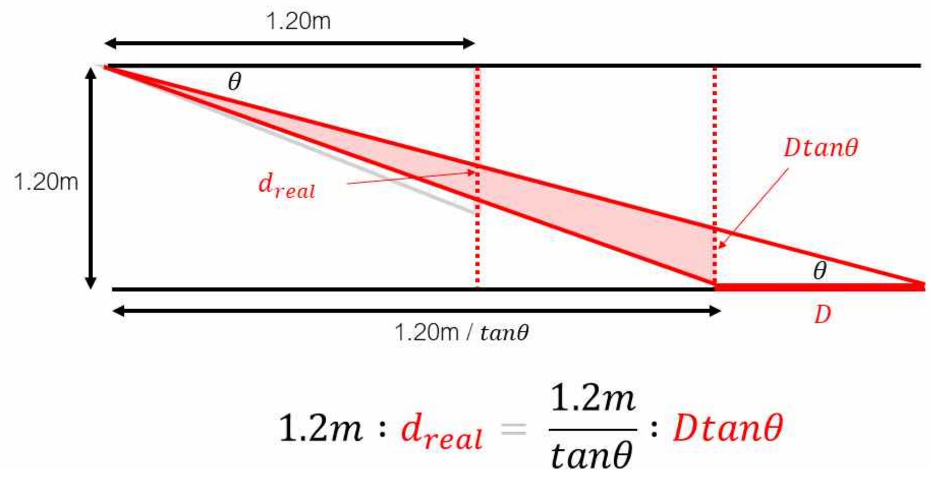
After supposition, we used proportional expression and trigonometric ratio to find the equation.



1.20m
θ
y
0.81m
H
$tan\theta = \frac{y}{H} * 0.81m/1.20m$
30 degrees, the half of the view of camera
0.81m
θ
Horizon line
1.2m
Object (t)
Object (t+1)
D
Supposed this position by camera view's crop position
1.2m / tanθ

pic6 Finding value of tangent

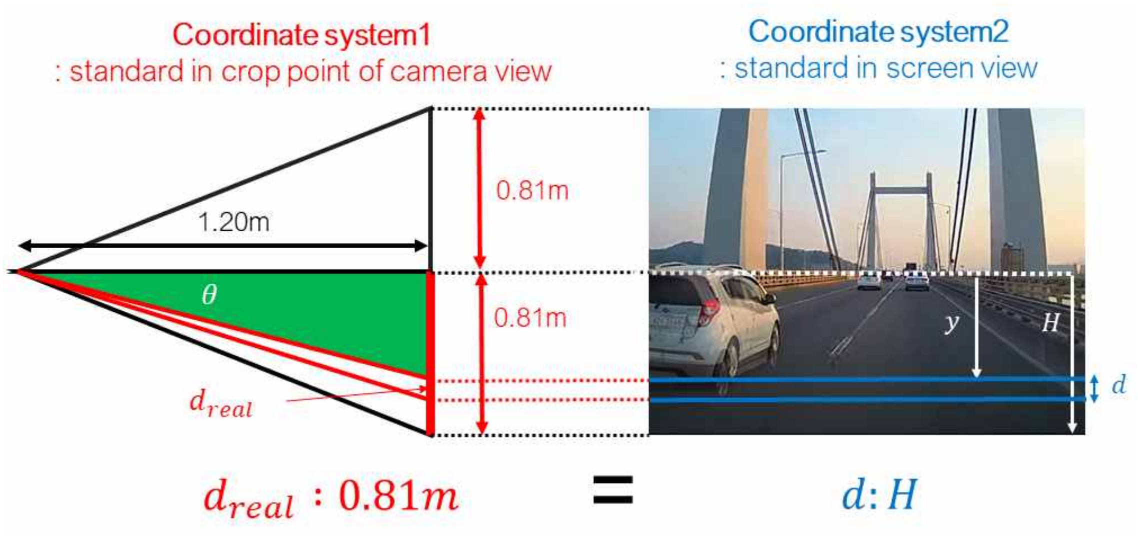
Based on measurement of model, we can get the tangent value of angle between horizon line and direct line to object's position (pic6).



$$1.2m : d_{real} = \frac{1.2m}{tan\theta} : Dtan\theta$$

pic7 Finding relation between D and d_real with tangent value

After that, we made equation between d_real and D with tangent value (pic7). Because of different coordinate system, d_real is the kind of correction of d to calculate. We can correct them by using proportional expression (pic8)


Coordinate system1
: standard in crop point of camera view
Coordinate system2
: standard in screen view
1.20m
0.81m
0.81m
θ
d_{real}
y
H
d
d_{real} : 0.81m = d: H

pic8 correction between d and d_real

Now, we can make the simple equation about D which is real distance on the road. We can make it by using three expression (pic6, pic7, pic8). Below is the equation which we created.

$$D = \frac{d_{real}}{tan^2\theta} = \frac{9}{5} * \frac{d}{H} * (\frac{H}{y})^2 \ (m)$$

As we defined on the above part, we can now calculate v_relative by using D and time duration.

$$v_{rel} = \frac{D}{10s/327} = 212 * \frac{d}{H} * (\frac{H}{y})^2 \ \text{(km/h)}$$

212 is kind of constant, and we can find that D is related to $\frac{d}{H}$ and $(\frac{H}{y})^2$. It is interesting that it is related to reverse position's square. We can guess that this term is representing preceptive.

2. Implement : with cv2.findContours

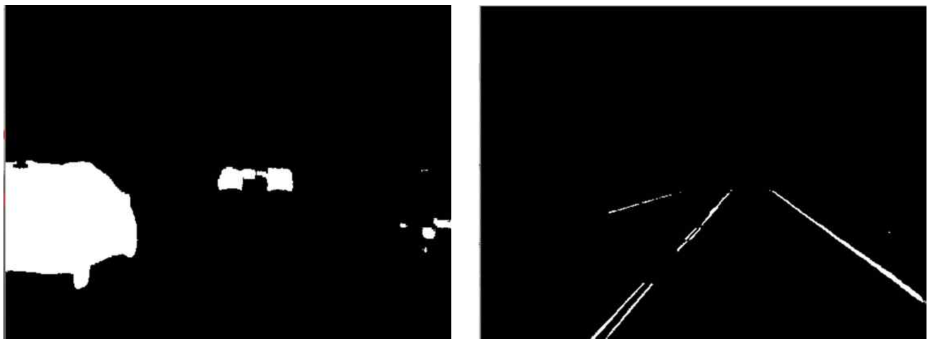
Now, we have to apply to our 327 label images to calculate speed. For that, we used opencv's FindContours method. It can find contours of the picture.

Before detecting, we will filtering the car label(number 5) and the lane label(number 10) from the label image. It is because that is precise way to find only cars. At this point, the image segmentation is being active.

```
black1 = cv2.imread("black.png")
black1=cv2.resize(black1,(480,360))
black2 = cv2.imread("black.png")
black2=cv2.resize(black2,(480,360))

label=cv2.imread("label/label%d.png"%(num))

for y in range(0,360):
  for x in range(0,480):
    if label[y][x][0]==10 and label[y][x][1]==10 and label[y][x][2]==10:
      black1[y][x]=[255,255,255]
    if label[y][x][0]==5 and label[y][x][1]==5 and label[y][x][2]==5:
      black2[y][x]=[255,255,255]
```

table3 extracting only cars(label 5) and only lanes(label 10)

As you can see in **table3**, we directly access to the RGB values of the label image, and change it in the for sentence. Remember that label images which we made has simple grayscale value (5 for cars and 10 for lanes). So, it would be work. For extracting only one label in one image, we made two empty black image. And then for each black image, we marked cars and lanes only. Lane-labeled image will be used in **B. Detect line invasion**.



pic9 extracting cars only(left) and lanes only(right) from label image

And then, we use cv2.findContours() method to find contours. After that, we can get the position of left top point of contour rectangle (x,y) and width, height(w,h) of the rectangle.

```
#find contours
car_a = cv.cvtColor(black2, cv.COLOR_BGR2GRAY)
_, car_b = cv.threshold(car_a, 127, 255, 0)
contours, hierarchy = cv.findContours(car_b, cv.RETR_LIST, cv.CHAIN_APPROX_SIMPLE)

max=0
for cnt in contours:
   x,y,w,h=cv.boundingRect(cnt)
   if (max<w*h):
      max=w*h
      cnt_max=cnt
```

table4 find contours(only max area contour)

When we consider all contours, it was difficult to calculate speed. Because contours were changed every frames so values are confused to be caculated. So, we will use only one contour which has max area(w*h) (table4).

```
v_rel = 212 * (abs(yh_present - yh_prior)) * 180 / ((abs(180 - (y + h))) ** 2)
v_rel = int(v_rel)
v=v_rel+60
text = str(v) + "km/h"
if (v_rel!=0):
   cv.putText(pic, text, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.8, color, 2)
cv.rectangle(pic, (x, y), (x + w, y + h), color, 2)
```

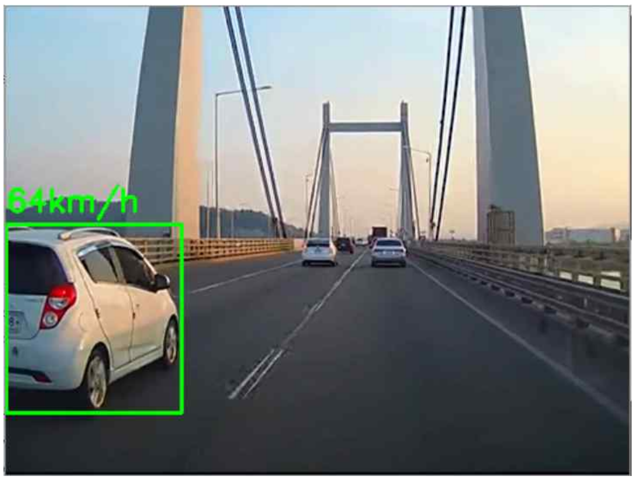
table5 draw contour rectangle and text the velocity

We calculate the distance on screen as difference of (y+h) value. It is corresponding to the bottom line’s vertical value, so that it can represent the move of the car.

So we made yh_present to represent current y+h value and yh_prior to represent past y+h value. So the abstract between them could be the distance on the screen (table5).

table 5 also consists the equation of velocity which we made. We added 60km/h for camera’s speed.

Now, we can measure the speed of moving cars. pic10 shows the contour rectangle's line and the speed of it.



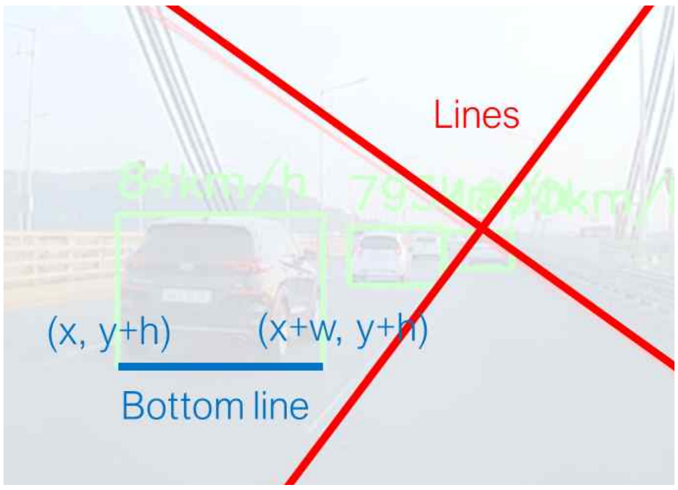
pic10 measuring speed

B. Detect Line invasion

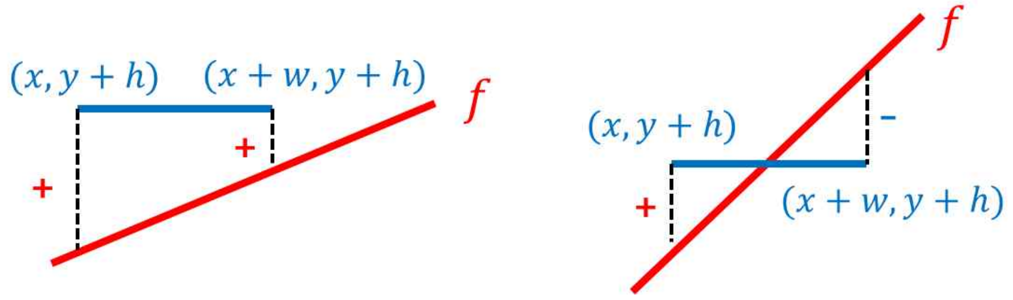
The reason why we find lanes is to detect line invasion. We will solve this problem with contour and the lanes which will be find with cv2.HoughLines() method.

1. Invasion calculation : line function

Before finding lines, we have to detect which situation is invaded or not. So we created simple logical equation of it.



pic11 using contour's bottom line to detect



pic12 detecting invasion. not invaded(left) and invaded(right)

We supposed that the bottom line of the contour rectangle represent the car's position, and using it to detect line invasion problem. The contour rectangle's two bottom positions are (x, y+h) and (x+w, y+h). The f is for the line function which we found by using cv2.HoughLines(). (pic11, pic12)
As you see in the pic12, when the two abstract distance has same sign, the car doesn't invade line.

$$\{y+h-f(x)\}^{+} \times \{y+h-f(x+w)\}^{+} > 0$$
$$\{y+h-f(x)\}^{-} \times \{y+h-f(x+w)\}^{-} > 0$$

But when the two abstract distance has different sign, the car does invade line.

$$\{y+h-f(x)\}^{+} \times \{y+h-f(x+w)\}^{-} < 0$$
$$\{y+h-f(x)\}^{-} \times \{y+h-f(x+w)\}^{+} < 0$$

So we can judge the invasion problem with detect the sign of the multiple of two distance.

2. Implement : with cv2.HoughLines()

As we did in A. Measure Speed, we will also use the pic9's extracted lanes. By using it, we will find the lanes on the label images. It would be more correct compare to the situation which we didn't extract only lanes.
The source codes which is using cv2. HoughLines() are below here(table 6). We controlled threshold parameter and min_theta, max_theta to correct the finding.

```
#find lines
line_a=cv.Canny(black1,50,200,None,3)
line_b = cv.cvtColor(line_a, cv.COLOR_GRAY2BGR)
lines = cv.HoughLines(image=line_a, rho=1, theta=np.pi / 180, threshold=50, lines=0,
srn=0,min_theta=np.pi*30/180, max_theta=np.pi*150/180)
```

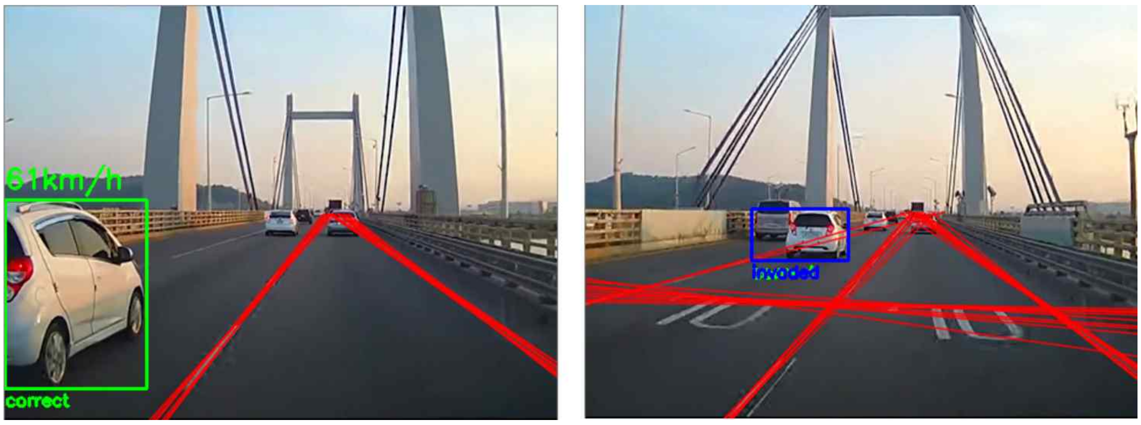
table6 using HoughLines() to find lines

After finding lines, we applied the equation which we made(**table7**). We texted "correct" for the correct lanes, and "invaded" for the invaded lanes.

```
f_x = math.tan(theta)*(x - x0) + y0
f_xw = math.tan(theta)*(x + w - x0) + y0
if ((y+h-f_x)*(y+h-f_xw))>0:
    cv.putText(pic, "correct", (x, y+h +15), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0),
2)
    color=(0,255,0)
else:
    cv.putText(pic, "invaded", (x, y+h + 15), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0,
0), 2)
    color=(255,0,0)
```

table7 apply equation of line invasion

You can see the result in **pic12**. It seems to be detected well, but some problems also exist. The lane direction is not correct, and contour's accuracy also low so that two cars are added.



pic12 detecting line correct and invasion

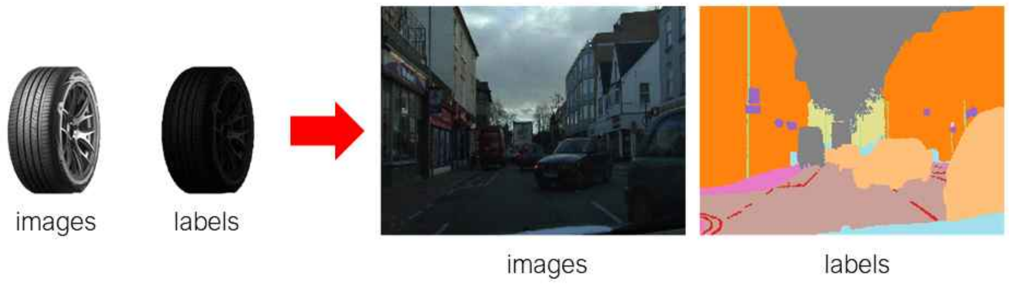
Part3. Addition task: partition learning

As you see in pic12, our main project's goal seems to be clear, but remain some problems either. We thought that main problems are occurred from the segmentation. So, we tried to find the way to improve the quality of segmentation. 'Partition learning' is the one of the way that we have tried.

A. Idea

In image segmentation or even general deep learning problem, making ground truth(label) dataset is a huge task to do. It is because we have to make it one by one.

In our project, same problem appeared. We wanted to add the label or re-learn the model, but there were no label images. So we have to make the label one by one (total 500~1000 images or more). It seems to be very difficult and boring task.



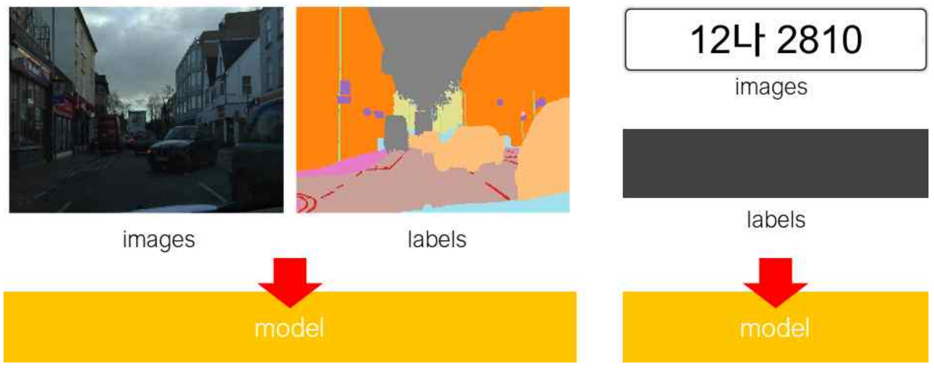
pic13 simple sketch of partition learning

To solve this problem, we thought about the 'partition learning'. Shortly, it means that learning by partition, not the whole label images. So, labeling would be easier because we just label them into one value. For example, if we want to segment the wheel of the car in the road images, we just learn with the single wheel images first, and then apply the original segmentation (pic13).

We have tried 3 formation of partition learning.

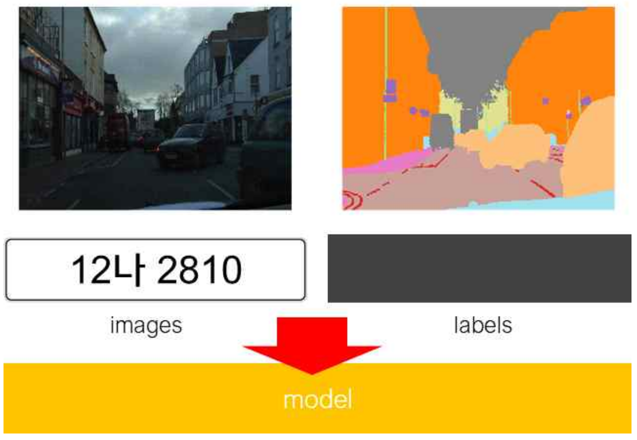
1. Learn by original dataset first, and then learn by partition dataset
2. Mix the original dataset and partition dataset, and then learn
3. Mixing by placing partition into random position of the original image

Here is the 3 way's conception. We will use number bar to test the partition learning.

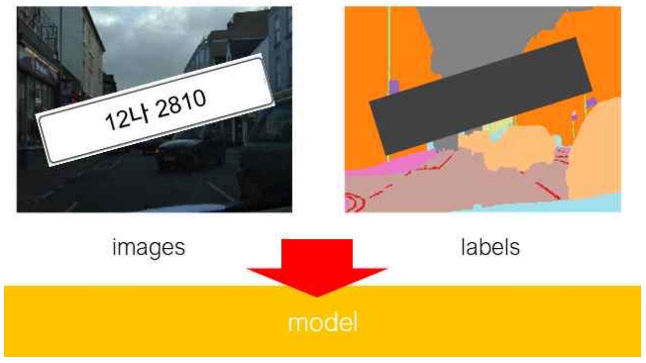


Try1 Separate-partition learning

Seperate-partion learning is only one that learns twice. Mixdata and Mixpixel-partion learning both learns once, but data set is different



Try2 Mixdata-partition learning



Try3 Mixpixel-partition learning

B. Tries

For every tries, we made data set about number bars with random numbers. We made 100 images and corresponding labels.

```
for i in range(0,100):
      img=bar.copy()
      text=str(random.randrange(10,99))+"XE"+str(random.randrange(1000,9999))
      cv.putText(img, text, (int(img.shape[1]*3/20),int(img.shape[0]*4/7)),
cv2.FONT_HERSHEY_SIMPLEX, 4.5, (0, 0, 0), 14)

      cv2.imwrite('bar_image/bar%d.png'%i, img)
      cv2.imwrite('bar_label/bar%d_P.png'%i,label)
```

table8 making number bar dataset

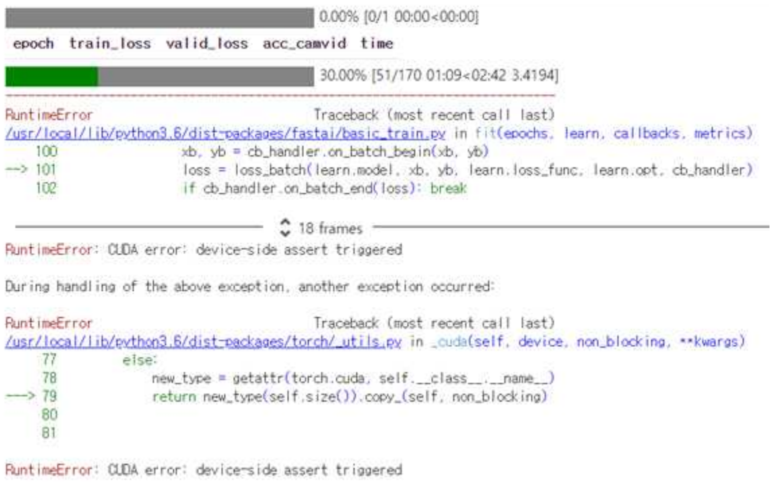
Try1: Separate-partition learning

We learned the model with original model first, and then learned with the numberbar datasets in Google colab. But after learning, it just judge the images by only second learning experience. It doesn't work.

The reason why it doesn't work is very simple. The model is initialized when the learner is defined (It is deserved fact). So, we moved to next try.

Try2: Mixdata-partition learning

For next try, we prepared mixing datasets to learn. 577 for the original images, and 100 for the numberbar images.



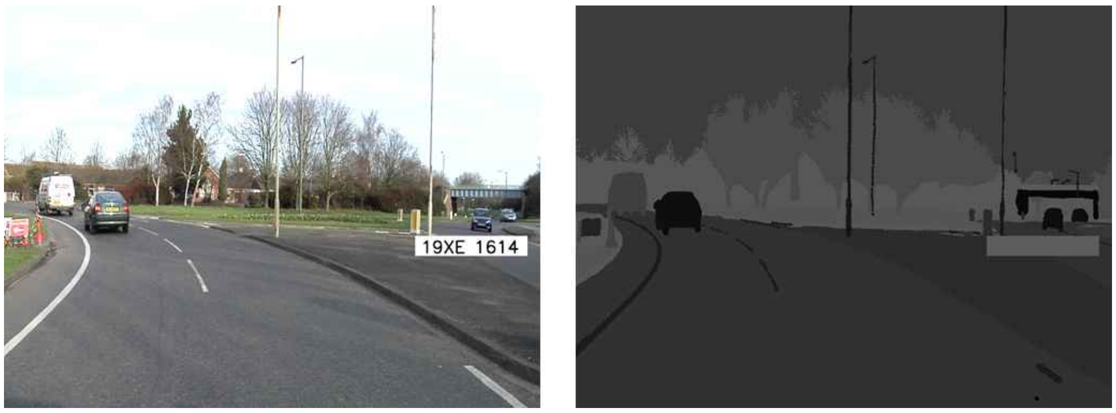
```
0.00% [0/1 00:00<00:00]
 epoch  train_loss  valid_loss  acc_camvid  time
30.00% [51/170 01:09<02:42 3.4194]
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
/usr/local/lib/python3.6/dist-packages/fastai/basic_train.py in fit(epochs, learn, callbacks, metrics)
    100             xb, yb = cb_handler.on_batch_begin(xb, yb)
--> 101             loss = loss_batch(learn.model, xb, yb, learn.loss_func, learn.opt, cb_handler)
    102             if cb_handler.on_batch_end(loss): break

                                 18 frames
RuntimeError: CUDA error: device-side assert triggered

During handling of the above exception, another exception occurred:

RuntimeError                              Traceback (most recent call last)
/usr/local/lib/python3.6/dist-packages/torch/_utils.py in _cuda(self, device, non_blocking, **kwargs)
     77         else:
     78             new_type = getattr(torch.cuda, self.__class__.__name__)
---> 79             return new_type(self.size()).copy_(self, non_blocking)
     80
     81

RuntimeError: CUDA error: device-side assert triggered
```

pic14 Error CUDA in Google colab

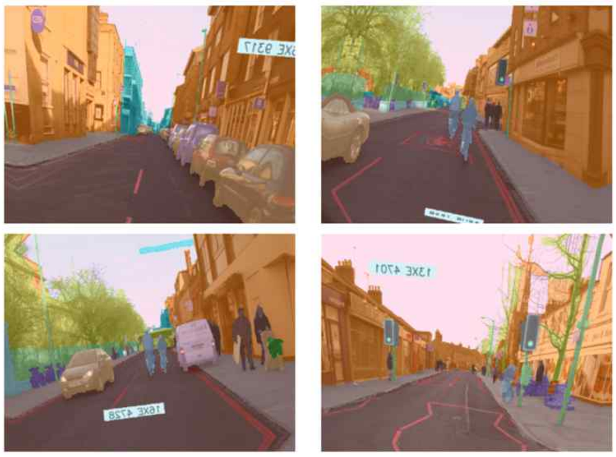
But as you see in pic14, there was error name CUDA when the learning process. We searched for that error and tried to fix it, but only reason we can guess is that there is problem in backpropagation because the dataset's formation is different between numberbar and original one. We couldn't find the exact reason. So, we moved to next try.

Try3: Mixpixel-partition learning

For Try3, we added numberbar into original image and corresponding dataset (pic15).



pic15 added numberbar into original image(left) and label image(right, brighten version)



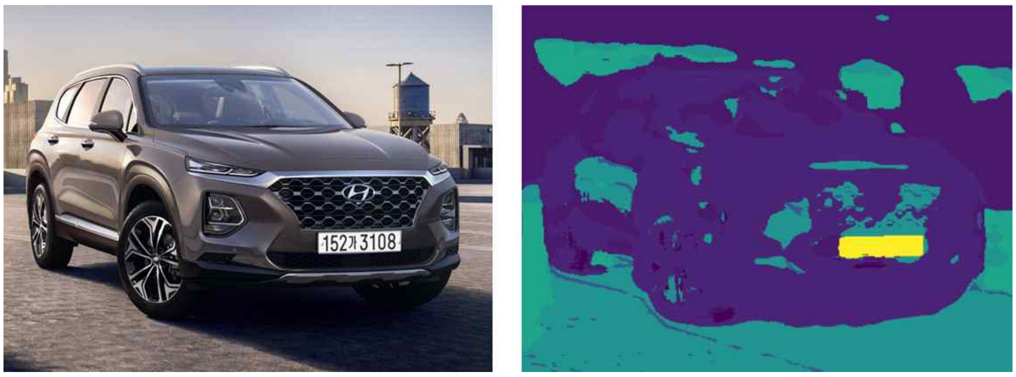
pic16 transforming of model

We were worried about the processing because the numberbar's formation is simple rectangle so that it would not detect other forms. But as you see in pic16, the model transforms the data so that the model which simply learned by horizontal forms of numberbars would detect the other shape of the numberbars. table9 is the source code which consists transforming process.

```
data = (src.transform(get_transforms(), size=size, tfm_y=True)
          .databunch(bs=bs)
          .normalize(imagenet_stats))
```

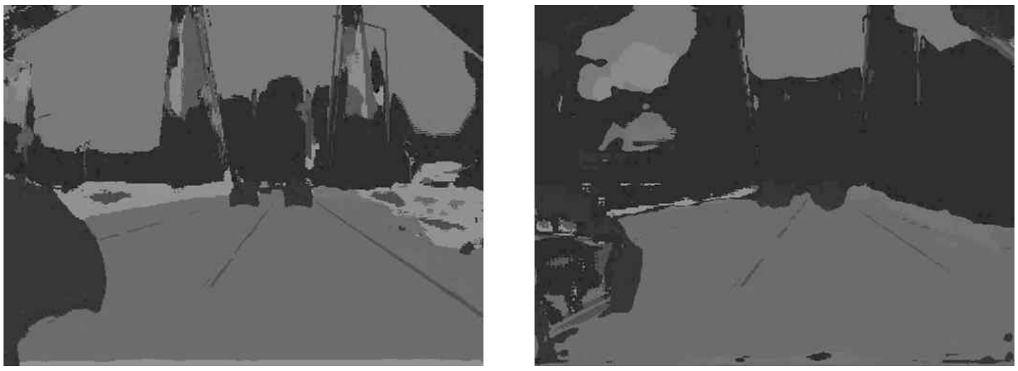
table9 tranform method in camvid notebook

Before checking with the frames which we finally apply, we applied it with easier image.



pic17 test image for Try3

As you can see in pic17, the model finds numberbar well. But the original segmentation seems to be wrong. The quality seems bad. So, after testing we applied the Try3 to image frames for checking the result.



pic18 Original labeling(left), Try 3's labeling(right)

As you can see in pic18, before mentioning the ability to find numberbars, the model's quality is low compared to the original model. We guessed that it is because of the deep learning model's sophistication. The learner doesn't segment the objects by using only each objects separately. Learner comprehensively consider the features, positions and configurations of objects.

So, thanks to model's intelligence, our partition-learning didn't proved by model.

Conclusion

We think that image segmentation is really interesting field in computer vision. How amazing that it could detect the objects well! So we used the image segmentation to clean the road without speeding problems. We wish that it could be commercialized on the road soon. Also earn money by catching speeding cars or invaded cars.

For edition task, we've tried to improve the quality of image segmentation by partition-learning, but it didn't work well. But now we know that the power of the machine learning networks.

Thanks to professor MoonRyul Jung so that we can create something fun and interesting application. Also thanks for the support for solve obstacles and teaching machine learning.