Deep learning final project

2016124091 남윤찬,2018124094 성우령

<problem definition>

-딥러닝을 활용한 도로 위 교통상황에서의 위험 예측

도로 위 교통상황에서 도로이용자를 탐지하여 사고를 예측하고, 운전자에게 사고 위험성을 알리는 시스템을 설계하고자 한다.

큰 파이프라인은 다음과 같다. 1. 객체인식, 2. 객체의 미래 위치 예측, 3. 위험구역을 설정하여 사고 위험성 알리기

자동차에 장착된 단안 카메라를 사용하여 객체를 인식한 후, 현재 프레임에서의 객체 위치를 기반으로 미래 프레임에서의 객체 위치를 예측한다. 이때 객체 인식에는 YoloV5를 사용한다.

자동차에서 바라보는 프레임에서의 위험구역을 설정하여 객체의 미래 위치가 위험구역에 들어올 경우 운전자에게 사고 위험을 알린다. 위험구역은 자동차 진행 방향의 차선을 인식하여 설정한다.

사고 영상 데이터셋을 사용하여 모델을 학습시켜 사고를 예측하도록 한다.

<Existing source code as a starting point>

Yolov5: https://github.com/ultralytics/yolov5.git

Deep Sort: https://github.com/tensorturtle/classy-sort-yolov5.git

Train과 detection을 yolo를 통해 진행하고, deep sort를 이용해 미래위치를 예측함

<dataset>

Dataset: https://github.com/MoonBlvd/tad-IROS2019.git

위의 github에서 './datasets/A3D_urls.txt'에 있는 YouTube 링크를 통해 영상 데이터셋을 활용하여 train 진행

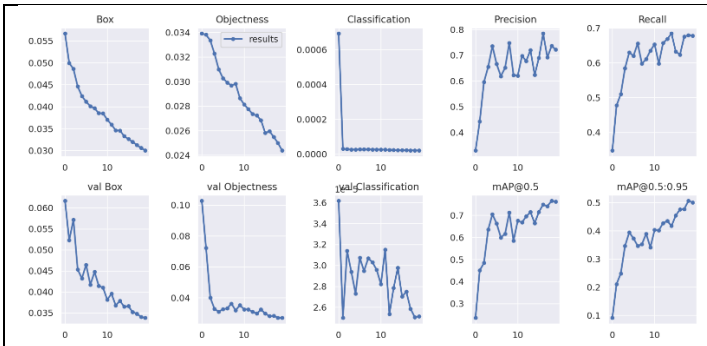
<Train>

위에서 받은 영상을 frame별로 나누어 차량에 대해 labeling을 진행한 뒤 train과 validation으로 나누어 yolo를 학습을 진행

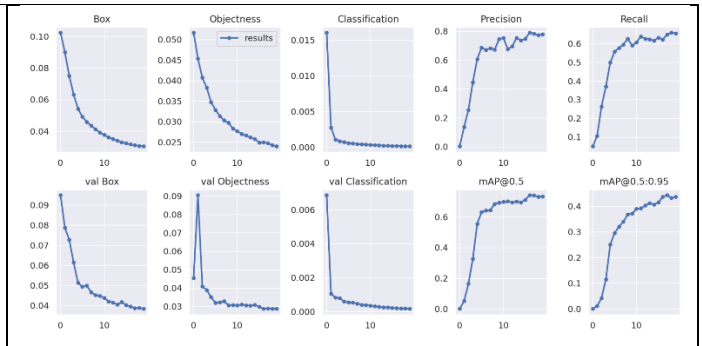
```
(pt18) guest0@cluster1:~/yolov5/dataset/images/train$ ls -l|grep ^-|wc -l
6015
```

```
(pt18) guest0@cluster1:~/yolov5/dataset/images/val$ ls -l|grep ^-|wc -l
1566
```

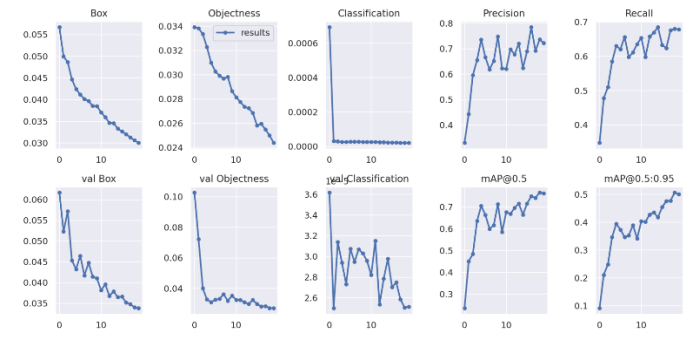
Train: 6015, Validation: 1566개의 축소 데이터로 모델 depth, optimizer, learning rate, pretrained weights 등을 바꿔가며 학습 결과가 가장 좋은 결과를 선택 (epochs = 20, batch size = 16으로 진행)



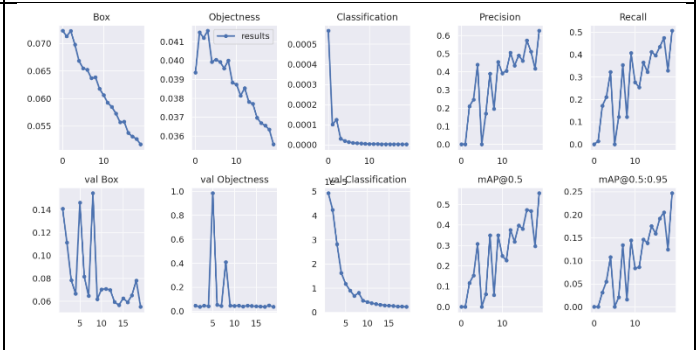
yolov5s, SGD, lr = 0.01



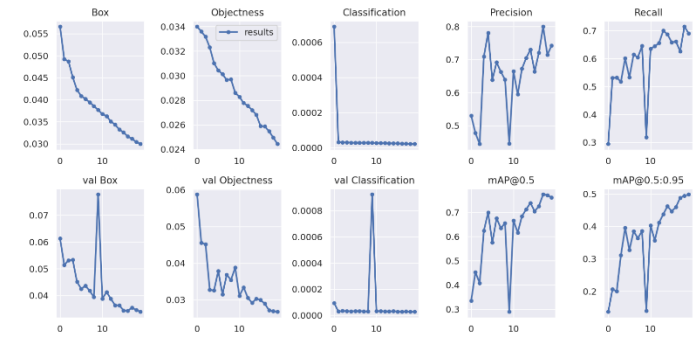
yolov5x, SGD, lr = 0.01



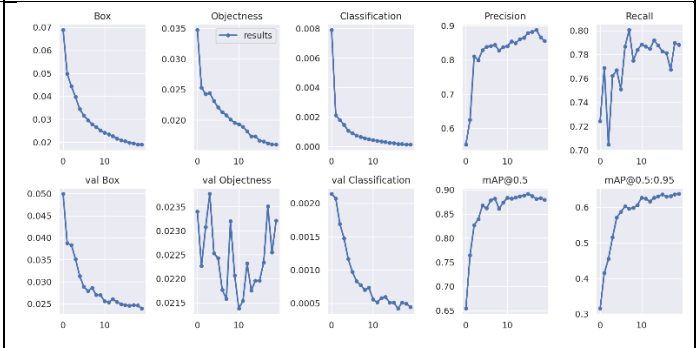
yolov5s, adam, lr = 0.01



yolov5s, adam, lr = 0.1



yolov5s, adam, lr = 0.001



yolov5l, SGD, lr= 0.01

- yolov5x, SGD, lr = 0.01로 train한게 가장 안정적인 지표를 보였고, yolov5l, SGD, lr= 0.01로 train 한것이 가장 높은 precision 점수를 보였다.

- yolov5 모델은 각 model이 총 yolov5s, yolov5m, yolov5l, yolov5x 4개로 이뤄져 있고 순서대로 depth가 깊고 사용하는 parameter도 많다. 네 개의 모델 중 하나를 선택하려면 train을 진행할 때 './models/'에 원하는 model의 구조가 지정되어 있는 '.yaml' 파일을 선택하여 사용한다. Yolov5 내부에서 자동으로 '.yaml'파일 내부에 있는 구조대로 backbone과 head의 architecture를 구성하여 train을 진행한다.

- yolov5에서 Training을 진행할 때 fine tuning을 위한 pretrained weights을 제공하는데 모델의 구조 개수와 마찬가지로 yolov5s, yolov5m, yolov5l, yolov5x 총 네 가지의 pretrained weights을 제공하여 train에 사용할 수 있다.

```
if opt.adam:
    optimizer = optim.Adam(pg0, lr=hyp['lr0'], betas=(hyp['momentum'], 0.999))
    #optimizer = optim.Adam(pg0, lr=1e-1, betas=(hyp['momentum'], 0.999))
else:
    optimizer = optim.SGD(pg0, lr=hyp['lr0'], momentum=hyp['momentum'], nesterov=True)
```

tarin.py

- 추가적으로 위의 코드에서 보다시피 train.py 내부에서 learning rate와 optimizer를 변경하여 train을 진행할 수 있다.

<Yolov5 Architecture>

- Yolov5의 **./models/common.py**에 구성 되어있는 yolov5의 backbone을 구성하는 핵심적인 부분들이다.

```
class Conv(nn.Module):
    # Standard convolution
    def __init__(self, c1, c2, k=1, s=1, p=None, g=1, act=True):  # ch_in, ch_out, kernel, stride, padding, groups
        super(Conv, self).__init__()
        self.conv = nn.Conv2d(c1, c2, k, s, autopad(k, p), groups=g, bias=False)
        self.bn = nn.BatchNorm2d(c2)
        self.act = nn.SiLU() if act is True else (act if isinstance(act, nn.Module) else nn.Identity())

    def forward(self, x):
        return self.act(self.bn(self.conv(x)))

    def fuseforward(self, x):
        return self.act(self.conv(x))
```

일반적인 convolution layer이다. Activation 함수로 SiLU를 사용했고 원하면 수정 가능하다.

```
class Bottleneck(nn.Module):
    # Standard bottleneck
    def __init__(self, c1, c2, shortcut=True, g=1, e=0.5):  # ch_in, ch_out, shortcut, groups, expansion
        super(Bottleneck, self).__init__()
        c_ = int(c2 * e)  # hidden channels
        self.cv1 = Conv(c1, c_, 1, 1)
        self.cv2 = Conv(c_, c2, 3, 1, g=g)
        self.add = shortcut and c1 == c2

    def forward(self, x):
        return x + self.cv2(self.cv1(x)) if self.add else self.cv2(self.cv1(x))
```

```
class BottleneckCSP(nn.Module):
    # CSP Bottleneck https://github.com/WongKinYiu/CrossStagePartialNetworks
    def __init__(self, c1, c2, n=1, shortcut=True, g=1, e=0.5):  # ch_in, ch_out, number, shortcut, groups, expansion
        super(BottleneckCSP, self).__init__()
        c_ = int(c2 * e)  # hidden channels
        self.cv1 = Conv(c1, c_, 1, 1)
        self.cv2 = nn.Conv2d(c1, c_, 1, 1, bias=False)
        self.cv3 = nn.Conv2d(c_, c_, 1, 1, bias=False)
        self.cv4 = Conv(2 * c_, c2, 1, 1)
        self.bn = nn.BatchNorm2d(2 * c_)  # applied to cat(cv2, cv3)
        self.act = nn.LeakyReLU(0.1, inplace=True)
        self.m = nn.Sequential(*[Bottleneck(c_, c_, shortcut, g, e=1.0) for _ in range(n)])

    def forward(self, x):
        y1 = self.cv3(self.m(self.cv1(x)))
        y2 = self.cv2(x)
        return self.cv4(self.act(self.bn(torch.cat((y1, y2), dim=1))))
```

Yolov5의 backbone의 핵심인 BottleneckCSP이다. cv1, cv4는 convolution + batch norm으로 되어있고, cv2, cv3는 convolution만 사용한다.

```
class Concat(nn.Module):
    # Concatenate a list of tensors along dimension
    def __init__(self, dimension=1):
        super(Concat, self).__init__()
        self.d = dimension

    def forward(self, x):
        return torch.cat(x, self.d)
```

단순히 2개의 convolution layer연산값을 합치는 모듈이다.

```
class SPP(nn.Module):
    # Spatial pyramid pooling layer used in YOLOv3-SPP
    def __init__(self, c1, c2, k=(5, 9, 13)):
        super(SPP, self).__init__()
        c_ = c1 // 2  # hidden channels
        self.cv1 = Conv(c1, c_, 1, 1)
        self.cv2 = Conv(c_ * (len(k) + 1), c2, 1, 1)
        self.m = nn.ModuleList([nn.MaxPool2d(kernel_size=x, stride=1, padding=x // 2) for x in k])

    def forward(self, x):
        x = self.cv1(x)
        return self.cv2(torch.cat([x] + [m(x) for m in self.m], 1))
```

Spatial Pyramid Pooling Layer로 k의 형태로 고정된 1차원형태의 배열을 생성하여 FC layer의 입력으로 들어갈 수 있게 해주는 모듈이다.

- 위에서 설명한 모듈 들이 어떻게 architecture로 만들어 주는 부분이다. /models/yolo.py 와 yolov5s.yaml과 같은 파일에서 찾아볼 수 있다.

```yaml
depth_multiple: 0.33  # model depth multiple
width_multiple: 0.50  # layer channel multiple

# anchors
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32

# YOLOv5 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Focus, [64, 3]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, C3, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 9, C3, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, C3, [512]],
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 1, SPP, [1024, [5, 9, 13]]],
   [-1, 3, C3, [1024, False]],  # 9
  ]
```

yolov5s.yaml

먼저 yolov5s.yaml에 저장되어 있는 모델의 구조이다. 총 4개의 깊이가 다른 모델이 있다고 했는데 yaml파일에 저장되어 있는 구조는 모두 동일하지만 depth_multiple, width_multiple 두 값이 다르다. 두 값이 클수록 depth가 깊어지고 width가 넓어지는 방식으로 자체적으로 모델을 구성한다.

```python
def parse_model(d, ch):  # model_dict, input_channels(3)
    logger.info('\n%3s%18s%3s%10s  %-40s%-30s' % ('', 'from', 'n', 'params', 'module', 'arguments'))
    anchors, nc, gd, gw = d['anchors'], d['nc'], d['depth_multiple'], d['width_multiple']
    na = (len(anchors[0]) // 2) if isinstance(anchors, list) else anchors  # number of anchors
    no = na * (nc + 5)  # number of outputs = anchors * (classes + 5)

    layers, save, c2 = [], [], ch[-1]  # layers, savelist, ch out
    for i, (f, n, m, args) in enumerate(d['backbone'] + d['head']):  # from, number, module, args
        m = eval(m) if isinstance(m, str) else m  # eval strings
        for j, a in enumerate(args):
            try:
                args[j] = eval(a) if isinstance(a, str) else a  # eval strings
            except:
                pass

        n = max(round(n * gd), 1) if n > 1 else n  # depth gain
        if m in [Conv, Bottleneck, SPP, DWConv, MixConv2d, Focus, CrossConv, BottleneckCSP, C3]:
            c1, c2 = ch[f], args[0]
            args = [c1, c2, *args[1:]]
            if m in [BottleneckCSP, C3]:
                args.insert(2, n)
                n = 1
        elif m is nn.BatchNorm2d:
            args = [ch[f]]
        elif m is Concat:
            c2 = sum([ch[-1 if x == -1 else x + 1] for x in f])
        elif m is Detect:
            args.append([ch[x + 1] for x in f])
            if isinstance(args[1], int):  # number of anchors
                args[1] = [list(range(args[1] * 2))] * len(f)
        else:
            c2 = ch[f]

        m_ = nn.Sequential(*[m(*args) for _ in range(n)]) if n > 1 else m(*args)  # module
        t = str(m)[8:-2].replace('__main__.', '')  # module type
        np = sum([x.numel() for x in m_.parameters()])  # number params
        m_.i, m_.f, m_.type, m_.np = i, f, t, np  # attach index, 'from' index, type, number params
        logger.info('%3s%18s%3s%10.0f  %-40s%-30s' % (i, f, n, np, t, args))  # print
        save.extend(x % i for x in ([f] if isinstance(f, int) else f) if x != -1)  # append to savelist
        layers.append(m_)
        ch.append(c2)
    return nn.Sequential(*layers), sorted(save)
```

yolo.py

Yolo.py에 있는 parse_model이라는 모듈이다. 이 모듈에서 모델 구조가 있는 yaml 파일을 읽어와 depth multiple,

width multiple 값을 이용해 위에서 설명한 모듈을 BottelneckCSP 모듈에서 bottleneck 부분을 몇 번을 더 반복할지를 정한다.

(Reference: https://ropiens.tistory.com/44)

<Explain Our Program>

A. Explain how to run your source code to test on unseen data. Please give sample command lines and test files that you did not use to train your model. So that we can easily figure out how to run your code.

(1) 리눅스 환경에서 risk_prediction directory로 이동하고 requirements를 설치합니다.

```
cd ./YourRoot/risk_prediction
pip install -r requirements.txt
```

(2) 테스트 영상으로 risk_pred.py를 실행합니다.

```
python classy_track.py --weights ./pre_weight.pt --source test_video.mp4   --save-img --save-txt
--classes 2
```

(3) ./inference/output 에서 결과 동영상을 확인하고 plot_risk.py를 실행하여 결과를 저장할 수 있습니다.

```
python plot_risk.py
```

★ Colab을 통해 간단하게 실행 해보실 수 있습니다. ★

https://colab.research.google.com/drive/1XH71W87KgdV3uZvCepnpjKIPGLBlyZN5?usp=sharing#scrollTo=CckiX6MoI6uE (git hub 용량 제한 때문에 colab에서는 pretrain weights를 적은 파라미터 수를 가지고 train한 weights을 사용해서 detection 성능이 비교적 좋지 않다는 점 양해 부탁드립니다.)

B-1. A description of any design decisions you made throughout the project. For example, how did you change you're the original source code to solve your problem?

```python
def draw_boxes(region,dx,dy,ds,img, bbox, identities=None, categories=None, names=None, offset=(0, 0)):
    score = 0
    score1 = 0
    for i, box in enumerate(bbox):
        x1, y1, x2, y2 = [int(i) for i in box]
        x1 += offset[0]
        x2 += offset[0]
        y1 += offset[1]
        y2 += offset[1]
        # box text and bar
        cx = round((x1+x2)/2) # box center 좌표
        cy = round((y1+y2)/2)
```

```
        cat = int(categories[i]) if categories is not None else 0
        id = int(identities[i]) if identities is not None else 0

        #color = compute_color_for_labels(id)
        label = f'{names[cat]} | {id}'
        t_size = cv2.getTextSize(label, cv2.FONT_HERSHEY_PLAIN, 2, 2)[0]

        w = x2 - x1
        h = y2 - y1

        # box 크기 변화율에 따라 근사화 하여 다음 box의 크기를 결정
        dx1 = round(int(ds[id-1])/(2*w + 1.6*h))
        dy1 = round(0.8*dx1)


        # 각 tracking id 마다 prediction한 cx,cy를 결정
        predy = cy + int(dy[id-1])
        predx = cx + int(dx[id-1])

        # box 크기 변화율에 따른 미래 box의 (x1,y1), (x2,y2) 예측
        x1_ = int(predx-w/2)-dx1
        y1_ = int(predy-h/2)-dy1
        x2_ = int(predx+w/2)+dx1
        y2_ = int(predy+h/2)+dy1

        # box내부로 들어오는 위험지역을 검출하여 점수로 계산
        pred_score = np.count_nonzero(region[y1_:y2_,x1_:x2_,0])
        real_score = np.count_nonzero(region[y1:y2, x1:x2,0])
        pred_risk_point = (pred_score/((x2_-x1_)*(y2_-y1_)+0.0000000001))*100
        real_risk_point = (real_score/((x2-x1)*(y2-y1)+0.0000000001))*100
        score1 += real_risk_point
        score += pred_risk_point

        # 위험지역에 들어온 box는 빨간색으로 표시하고, box 왼쪽위에 score 표시
        if x1_ < img.shape[1] and x2_ <img.shape[1] and y1_ < img.shape[0] and y2_ < img.shape[0]:
          if pred_risk_point == 0.0 and real_risk_point == 0.0:
                #cv2.rectangle(img, (x1, y1), (x2, y2),color, 1)
                cv2.rectangle(img, (x1_, y1_), (x2_, y2_),[0,255,0], 1)
                cv2.putText(img, f'{pred_risk_point}', (x1_, y1_ +
                                  t_size[1] + 4), cv2.FONT_HERSHEY_PLAIN, 1, [0, 255, 0], 1)
          else:
                #cv2.rectangle(img, (x1, y1), (x2, y2),[0,0,255], 1)
                cv2.rectangle(img, (x1_, y1_), (x2_, y2_),[0,0,255], 1)
                cv2.putText(img, f'{pred_risk_point}', (x1_, y1_ +
                                  t_size[1] + 4), cv2.FONT_HERSHEY_PLAIN, 1, [0, 0, 255], 1)

    # 점수 0~1로 정규화
    if score >= 50 :
        score = 1
    else:
        score = score/50

    if score1 >= 50 :
        score1 = 1
```

```
    else:
        score1 = score1/50
    return score, score1
```

risk_prediction.py 파일의 일부분이며 원본은 classy_track.py 입니다. 가장 핵심적으로 예측된 bounding box를 그리는 부분입니다. 원래는 id별로 box를 생성해내는 기능만 있던 모듈이지만 위험지역, box좌표의 변화량, box크기의 변화량을 추가적으로 입력으로 받아서 미래 box를 생성하고, score를 반환하도록 하였습니다.

```
##############################################
# 위험지역 설정
region = np.zeros((im0.shape))
region = lane_roi.lane_roi(region)
##############################################
```

risk_prediction.py 에서 위험지역을 설정하는 코드입니다.

```
import cv2
import numpy as np

def lane_roi(image):
  imshape = image.shape
  #gray = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
  vertices = np.array([[(0,imshape[0]),(0,imshape[0]*0.7),(imshape[1]/2-
80,imshape[0]*0.65),(imshape[1]/2+80,imshape[0]*0.65),(imshape[1],imshape[0]*0.7) ,(imshape[1],imshape[0])]], dtype =
np.int32)
  mask = np.zeros(image.shape, image.dtype)
  cv2.fillPoly(mask, vertices,(100,150,100))
  result = cv2.addWeighted(image,1,mask,0.4,0)
  return result
```

lane_roi.py에 직접 custom한 위험지역을 그리는 모듈을 사용하였습니다. 추가적으로 lane.py에 lane() 모듈을 사용하면 차선을 인식하여 위험지역으로 설정합니다. (차선인식 방법은 여러 상황에 따라 성능이 좋지 못해서 사용하지 않았습니다.)

```
  ##########################################################
  # sum을 통해서 각 id 마다 15 프레임 만큼의 변화율을 더해줌
  dx = torch.sum(u_d,axis = 1)
  dy = torch.sum(v_d,axis = 1)
  ds = torch.sum(s_d,axis = 1)
  ##########################################################
```

```
# tracking 알고리즘으로 부터 좌표 정보와 변화율 정보를 받아옴
if save_txt and len(tracked_dets) != 0:
    for j, tracked_dets in enumerate(tracked_dets):
        bbox_x1 = tracked_dets[0]
        bbox_y1 = tracked_dets[1]
        bbox_x2 = tracked_dets[2]
        bbox_y2 = tracked_dets[3]
        category = tracked_dets[4]
        u_overdot = tracked_dets[5]
        v_overdot = tracked_dets[6]
        s_overdot = tracked_dets[7]
        identity = tracked_dets[8]
        ##################################################################
        #
        u_d[int(identity)-1,pred_frame] = u_overdot
        v_d[int(identity)-1,pred_frame] = v_overdot
        s_d[int(identity)-1,pred_frame] = s_overdot
        cx = round((bbox_x1+bbox_x2)/2)
        cy = round((bbox_y1+bbox_y2)/2)
        cx_pred = torch.round(cx + dx[int(identity)-1])
        cy_pred = torch.round(cy + dy[int(identity)-1])
        ##################################################################
        with open(txt_path, 'a') as f:
            f.write(f'{frame_idx},{bbox_x1},{bbox_y1},{bbox_x2},{bbox_y2},{category},{u_overdot},{v_overdot},{s_overdot},{identity},\n')
```

risk_prediction.py 파일에 있는 tracking 알고리즘(deep sort)으로부터 좌표정도, 변화율 정보 등을 가져오는 코드입니다. 별도로 선언한 배열을 통해 각 id 별로 변화율 값을 프레임 별로 저장 받고 15프레임 정도의 값을 합하여 0.5초동안의 미래를 예측하고자 하였습니다.

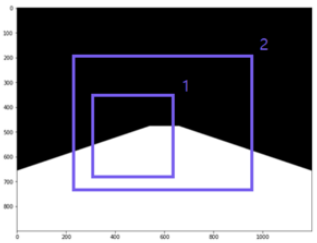
B-2. How did you build a dataset to train your model?

https://github.com/MoonBlvd/tad-IROS2019.git 에서 제공하는 A3D dataset에서 약 10개의 사고영상을 다운받아 frame 별로 나눈 뒤 성능이 가장 좋은 yolov5x로 detection을 수행하여 첫번째 labeling을 진행하고, 제대로 인식되지 않은 객체에 대해서는 roLabelImg (https://github.com/cgvict/roLabelImg.git) 프로그램을 이용해 yolo버전으로 labeling을 진행했다.

사실 yolo자체적으로 제공하는 pretrain weights을 사용하는 것도 성능이 나쁘지 않았지만 class가 80개나 되는 weights이기 때문에 detection하는데 실행시간이 오래 걸리는 단점이 있다.

B-3. What was your evaluation metric and why did you choose it?

Train에 대한 성능지표는 yolov5 자체적으로 여러 성능지표를 plot해주기 때문에 변경하지 않았고, 위험도 예측에 대한 evaluation을 진행했다.



bw, bh, w, h : Bbox와 image의 높이, 넓이

$B = \frac{x}{bw * bh}$: Bbox의 위험지역 비율

$I = \frac{x}{w * b}$: 전체 이미지에서 bbox에 들어온 위험지역 비율

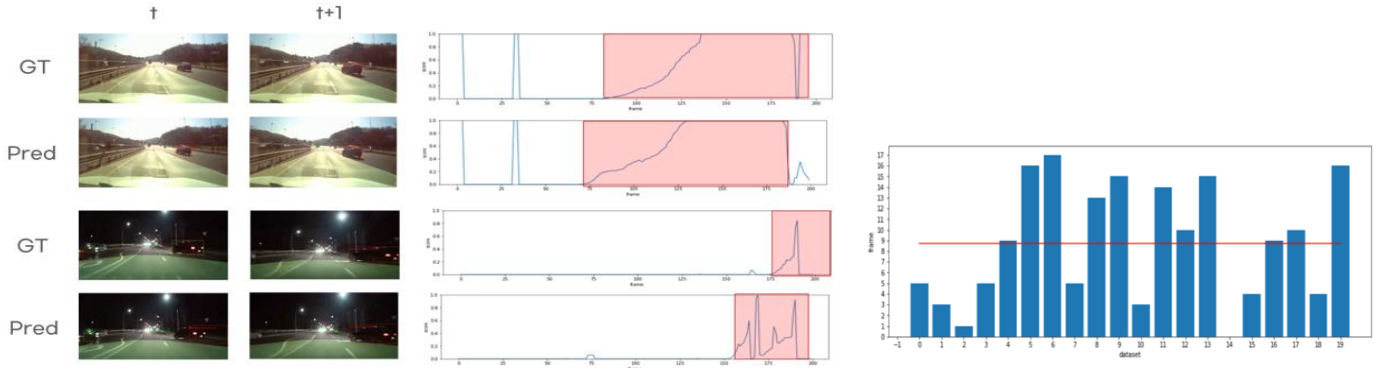
$$score = \lambda_1 B + \lambda_2 I$$

위와 같은 방법으로 미래위치의 정확도 보다는 위험을 얼마나 더 빠르게 예측하는가에 초점을 맞춰서 GT와의 성능을 비교했다. Object detection이 객체의 거리는 측정할 수 없기에 전체 이미지 크기에서 box 내부에 들어온 위험지역의 비율을 추가하여 계산하였다.

B-4. How did you optimize your model?

Yolov5의 학습 모델은 기본적으로 설정되어 있는 구성에서의 학습 정확도와 점수가 가장 좋았다. 학습을 진행하면서 모델의 크기를 변형하면서 가장 좋은 지표를 나타내는 것을 선택했고, lr, batch size, activation function등을 바꿔보면서 학습이 어떤 것이 가장 잘 되는지를 확인하였다. 그 결과 세번째로 큰 모델인 yolov5l 모델을 사용하고, SGD optimizer를 사용하고, lr과 activation function은 초기 설정을 변경하지 않는 것이 가장 좋은 결과를 나타냈다.

C. The results of your user evaluation experiment.



위험 예측에 대한 성능 평가는 위와 같다. 총 20개의 testset으로 youtube 사고영상을 사용했고, 6개 영상이 GT 보다 15frame이상 빠르게 위험을 예측했고, 평균 8.7frame 먼저 예측했다.

Train 정확도에 대한 evaluation은 위의 **<Train>** 목차에 자세하게 설명해 놓았습니다.

D. Give credit to any source of assistance (students with whom you discussed for this project, instructors, books, online sources, etc.)

(1) Yolov5: https://github.com/ultralytics/yolov5.git

(2) Deep Sort: https://github.com/tensorturtle/classy-sort-yolov5.git

(3) Dataset: https://github.com/MoonBlvd/tad-IROS2019.git

(4) labeling: https://github.com/cgvict/roLabelImg.git

(5) Yolov5 architecture: https://ropiens.tistory.com/44