클래스 불균형 문제를 해결하기 위한 데이터 증강

안민혁

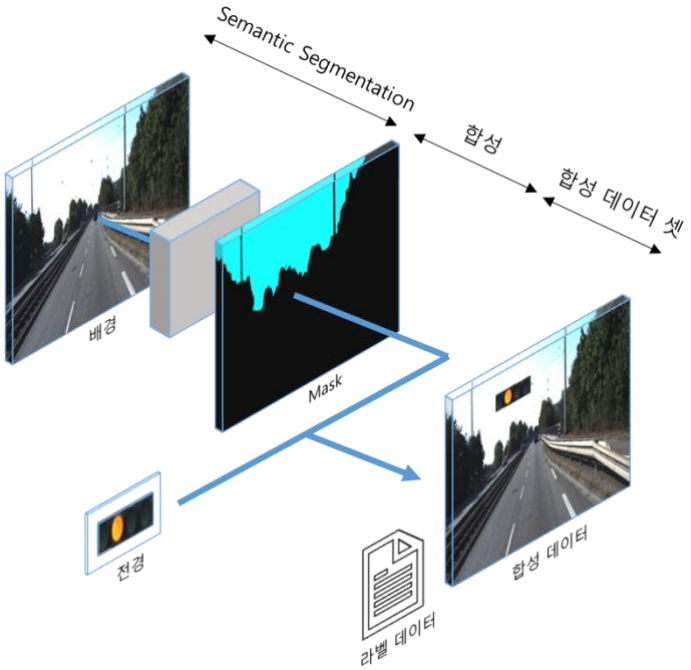

그림 1: 황색 신호 데이터 증강 프로그램 구조

1. 서론

1.1 배경

딥러닝 모델 학습에 있어서 클래스 불균형 문제는 아주 흔하게 접할 수 있는 문제다. 예를 들어, 신호등 상태 정보(적, 황, 녹)를 구분하기 위한 학습 데이터 셋을 구축하기 위해 도로에 나서서 신호등 영상을 촬영한다고 가정해보자. 적색 신호와 녹색 신호는 수십 초 이상 점등되는 반면 황색 신호는 단 3초밖에 점등되지 않기 때문에 전체 획득 데이터 중 황색 신호의 이미지 수의 비율은 현저하게 낮다. 이것을 클래스 불균형 문제라 하며, 클래스 불균형 상태에서 딥러닝 모델 학습 시 모델의 추론 결과가 예상대로 나타나지 않는 문제가 발생한다. 따라서 본 과제는 딥러닝 네트워크를 응용하여 흔하게 취득할 수 있는 배경 이미지와 그렇지 않은 황색 신호 이미지(이하 '전경'이라고 한다.) 입력으로부터 이미지 합성 결과물과 라벨 데이터를 포함한 합성 데이터 셋을 자동으로 생성하는 시스템을 구축하여 클래스 불균형 문제를 해소함과 동시에 데이터 구축 비용 절감 효과를 얻고자 한다. 그리고 합성 데이터만으로 모델을 학습한 뒤 모델의 성능을 평가하여 합성 데이터가 실제로 학습 데이터로써 가치가 있는지 확인해보고자 한다.

1.2 데이터 증강 프로그램 설계

보다 편리한 데이터 처리를 위해, 하나의 프로그램을 여러 단위로 세분화하여 각각이 독립적인 기능을 가지는 사용자 정의 함수로 정의하여 사용하고자 한다.

1.2.0 의존성 패키지 설치 및 가져오기

```
!pip install torchvision==0.8.2
!pip install timm==0.3.2
!pip install mmcv-full==1.2.7
!pip install opencv-python==4.5.1.48
%cd drive/MyDrive/SegFormer/
!pip install -e . --user
# 오류 발생 시 terminaltables 재설치
!pip uninstall terminaltables -y
!pip install terminaltables
```

그림 2: 의존성 패키지 설치

```
from argparse import ArgumentParser
from mmseg.apis import inference_segmentor, init_segmentor, show_result
_pyplot
from mmseg.core.evaluation import get_palette
import cv2
import numpy as np
import random
import os
count=0
fail=0
```

그림 3: 모듈 가져오기

파이썬 패키지는 pip install 명령어로 설치할 수 있다. 과제에 사용되는 프로그램은 그림 2와 같은 의존성 패키지를 설치하여 사용해야 하며 파이썬 코드 내에서 그림 3과 같이 모듈을 가져와 그 기능을 사용할 수 있게 된다.

1.2.1 입력 이미지 구성 및 불러오기

그림 4: 배경 이미지 예시

그림 5: 전경 이미지 예시

본 과제에서 사용한 배경 이미지는 KITTI 데이터셋[1]으로부터 샘플을 추출하여 사용했고 전경 이미지는 뒷배경과 분리 취득하여 사용하였으며 전경 이미지는 자연스러운 합성을 위해 배경이 제거된 PNG 포맷으로 제한했다. 또 전경 이미지의 크기가 라벨 데이터의 바운딩 박스 크기를 결정하므로 최대한 빈 공간이 없도록 제작했다.

```python
# 경로 지정을 위한 문자열 합치기
def fullfile(a,b):
  dir = []
  dir.append(a)
  dir.append(b)
  dir=''.join(dir)
  return dir
```

그림 6: 사용자 정의함수 fullfile()

여러 이미지를 불러올 때 경로를 지정하기 용이하도록 문자열을 합치는 함수 fullfile()를 정의했다.

1.2.2 Semantic Segmentation

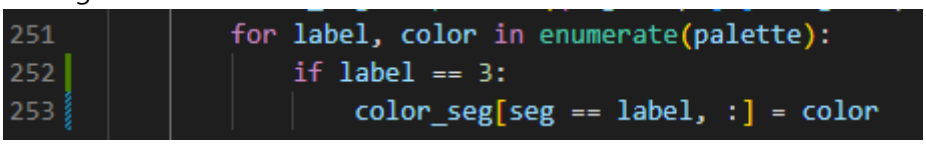

그림 7: SegFormer/mmseg/models/segmentors/base.py: 줄 252

현실 세계의 황색 신호등은 지면으로부터 일정 거리 떨어져 있는 곳에 고정되어 있기 때문에 이미지 합성 시 하늘을 등진 위치에 합성하는 것이 가장 자연스럽다. 그것을 구현하기 위해 프로그램에 Semantic Segmentation 모델을 적용하여 배경 이미지에 대한 Scene Understanding이 가능하도록 구현하고자 했다. 모델으로는 SegFormer[2]를 채택했으며 ADE20K 데이터셋[3]으로부터 학습된 Pretrained 가중치 파일을 사용하였다. 그림 7과 같이 모델 알고리즘 일부를 수정하여 카테고리 3번에 해당하는 'sky' 군집 영역에 대한 결과만 획득할 수 있도록 구현했다.

```python
# Segformer 모델 불러오기
model = init_segmentor('local_configs/segformer/B1/segformer.b1.512x512
.ade.160k.py', 'segformer.b1.512x512.ade.160k.pth', device='cuda:0')

# Segmentation 결과 Mask 얻기
def inference(bgimg, model):
  result = inference_segmentor(model, bgimg)
  mask=show_result_pyplot(model, bgimg, result, get_palette('ade'))

  return mask
```

그림 8: 사용자 정의 함수 inference().

그림 8은 모델을 불러오기 위한 코드와 모델 inference 결과를 얻는 기능을 하는 함수를 나타내고 있다.

1.2.3 전경 이미지 위치 및 사이즈 결정

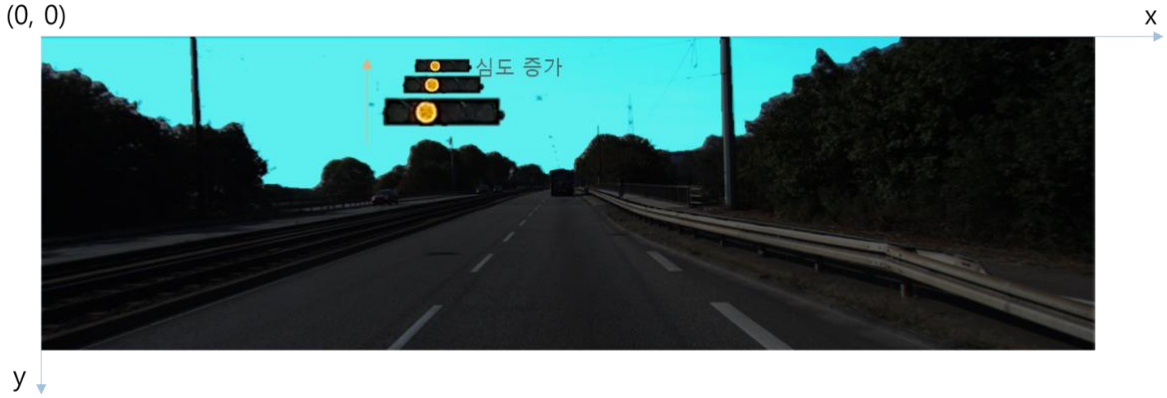

그림 9: Segmentation 기반 y 좌표에 따른 전경 사이즈 조절 예시.

1.2.2에서의 Segmentation Mask 데이터를 바탕으로 전경 이미지 합성이 가능한 위치와 사이즈를 결정할 수 있다. 전경 이미지의 위치는 'sky' 카테고리 영역으로 제한하며, 사이즈는 배경 이미지보다 크지 않아야 한다는 제한 조건을 설정했다. 그림 10의 함수를 이용하여 전경이 배경보다 크지 않도록 1차로 사이즈를 조정하고 그림 11의 함수를 이용하여 전경의 위치를 임의로 한 곳 선정한 뒤, 그 위치에 근거하여 그림 9과 같이 y 좌표에 따라 전경의 사이즈를 최종 결정할 수 있도록 구현했다.

```
# 전경 사이즈 1 차 조정
def size(obj, mask):
  k = 0.4 # 사이즈 조절 계수 (임의)
  bh, bw = mask.shape[:2]
  fh, fw = obj.shape[:2]
  ratio = fh/fw # 비율에 맞게
  #사이즈 조정
  if bh>=bw: # 세로가 더 긴 이미지
    out = cv2.resize(obj, dsize=(int(k*bw), int(k*bw*ratio)), interpola
tion=cv2.INTER_AREA)

  elif bw>bh: # 가로가 더 긴 이미지
    out = cv2.resize(obj, dsize=(int(k*bh/ratio), int(k*bh)), interpola
tion=cv2.INTER_AREA)

  return out
```

그림 10: 사용자 정의 함수 size()

```
# 전경 위치 결정
def position(obj,mask):
  bh, bw = mask.shape[:2]
  # Mask 이진화
  mask[mask>0] = 255
  a = np.array(mask)
  # 전경이 합성될 수 있는 위치 결정
  y,x = np.where(a==255)
  # 위치(2 차원 데이터) 중 랜덤으로 한 곳 결정
  rand = random.randint(0,len(y)-1)
  dx = x[rand]
  dy = y[rand]
  # 심도에 따른 전경 사이즈 최종 결정
  ratio_y = dy/bh
  out = cv2.resize(obj, dsize=(0,0), fx=ratio_y, fy=ratio_y, interpolat
ion=cv2.INTER_AREA)
  fh, fw, fc = out.shape
  # 사이즈 2 차 조정 후 위치 재확인
  dx_max = bw-fw-1
  dy_max = bh-fh-1
  dx = dx - int(fw/2)
  dy = dy - fh
  if dx>=dx_max:
    dx = dx_max
  elif dx <= 0:
    dx =0
  if dy>=dy_max:
    dy = dy_max
  elif dy <= 0:
    dy = 0
  return out,dx,dy
```

그림 11: 사용자 정의 함수 position()

1.2.4 영상 합성

그림 12와 같이 전경이 합성될 위치와 사이즈 정보를 바탕으로 배경과 전경을 합성한다. 마스크 기반 이미지 연산이 수행된다.

```
# 영상 합성하기
def compose(obj, bg, dx, dy):
  fh, fw = obj.shape[:2]
  bh, bw = bg.shape[:2]
  # 전경 마스크 생성
  _, mask = cv2.threshold(obj[:,:,3], 1, 255, cv2.THRESH_BINARY)
  mask_inv = cv2.bitwise_not(mask)
  obj = cv2.cvtColor(obj, cv2.COLOR_BGRA2BGR)
  # 전경이 합성될 위치 제한
  if dy+fh>=bh and dx+fw>=bw:
    roi = bg[dy:bh-1, dx:dx+fw]
  elif dx+fw>=bw:
    roi = bg[dy:dy+fh, dx:bw-1]
  elif dy+fh>=bh:
    roi = bg[dy:bh-1, dx:bw-1]
  else:
    roi = bg[dy:dy+fh, dx:dx+fw]
  masked_fg = cv2.bitwise_and(obj, obj, mask=mask)
  masked_bg = cv2.bitwise_and(roi, roi, mask=mask_inv)
  #전경 합성
  added = masked_fg + masked_bg
  composite = bg
  composite[dy:dy+fh, dx:dx+fw] = added
  return composite
```

그림 12: 사용자 정의 함수 compose()

1.2.5 라벨 데이터 생성

```
0 0.402979 0.117333 0.045894 0.058667
```

그림 13: 라벨 데이터 예시.

합성에 이용된 전경의 위치 및 사이즈 정보를 바탕으로 바운딩 박스 라벨 데이터를 생성한다. 그림 13의 첫 번째 열은 정수형 데이터로 카테고리 번호를, 2~5번째 열은 0 이상 1 이하의 float 형 값으로 이미지 너비와 높이에 따라 정규화 된 전경의 x 좌표, y좌표, 너비, 높이를 담고 있다. 본 과제에서는 합성에 사용된 전경의 카테고리 번호를 0으로 정해서 사용했다.

```
def txtgen(outtxt, clss, fw,fh, dx, dy, bh, bw): # 라벨 데이터 생성
  f = open(outtxt, 'w')
  cx = dx+fw/2
  cy = dy+fh/2
  list = [clss, str(round(cx/bw,6)), str(round(cy/bh,6)), str(round(fw/
bw,6)), str(round(fh/bh,6))]
  list=' '.join(list)
  f.writelines(list)
  print(list)
  f.close()
```

그림 14: 사용자 정의 함수 txtgen()

1.2.6 메인 함수

```
if __name__ == "__main__" :
  bg_list = os.listdir('KITTI') # 배경 이미지 리스트
  obj_list = os.listdir('yellow') # 전경 이미지 리스트

  for i in range(0, len(bg_list)): # 배경 이미지 수 만큼 반복
    bg_dir = fullfile("KITTI/", bg_list[i]) # 문자열 합쳐서 경로 지정
    bg_img = cv2.imread(bg_dir) # 배경 이미지 읽기
    mask = inference(bg_img, model) # 배경 segmentation 결과 저장
    mask=cv2.cvtColor(mask, cv2.COLOR_BGR2GRAY) # 구현이 편리하도록 3 채널(
RGB)-> 1 채널(GRAY) 변환
    for j in range(0, len(obj_list)): # 전경 이미지 수 만큼 반복
      bg_img = cv2.imread(bg_dir) # 배경 이미지 읽기
      obj_dir = fullfile("yellow/", obj_list[j]) # 문자열 합쳐서 경로 지정
      obj_img = cv2.imread(obj_dir, cv2.IMREAD_UNCHANGED) # 전경 이미지
읽기
      obj_img = size(obj_img, mask) # 전경 사이즈 1 차 조정
      try:
        obj_img, dx, dy = position(obj_img, mask) # 전경 위치 결정 및 사이
즈 2 차 조정
      except:
        fail=fail+1 # 전경 위치를 결정할 수 없다면 fail++
        continue
      composite = compose(obj_img, bg_img, dx, dy) # 영상 합성
      fh, fw = obj_img.shape[:2]
      bh, bw = mask.shape[:2]
      txtgen("out/"+str(count)+".txt", '0', fw,fh, dx, dy, bh, bw) # 라
벨 데이터 생성
      cv2.imwrite("out/"+str(count)+".jpg", composite) # 영상 저장
      count=count+1 # 합성 성공 시 count++
      print(count)
  print('Done.')
  print('Success:', count)
  print('No sky found:', fail)
```

그림 15: 메인 함수

그림 15의 메인 함수는 프로그램의 기본 함수로써 2중 반복문으로 설계되었으며, 저장소로부터 입력 데이터를 읽은 것을 사용자 정의 함수에 통과시켜 결과 데이터를 얻는 구조로 작성되어 있다.

1.2.7 프로그램 순서도

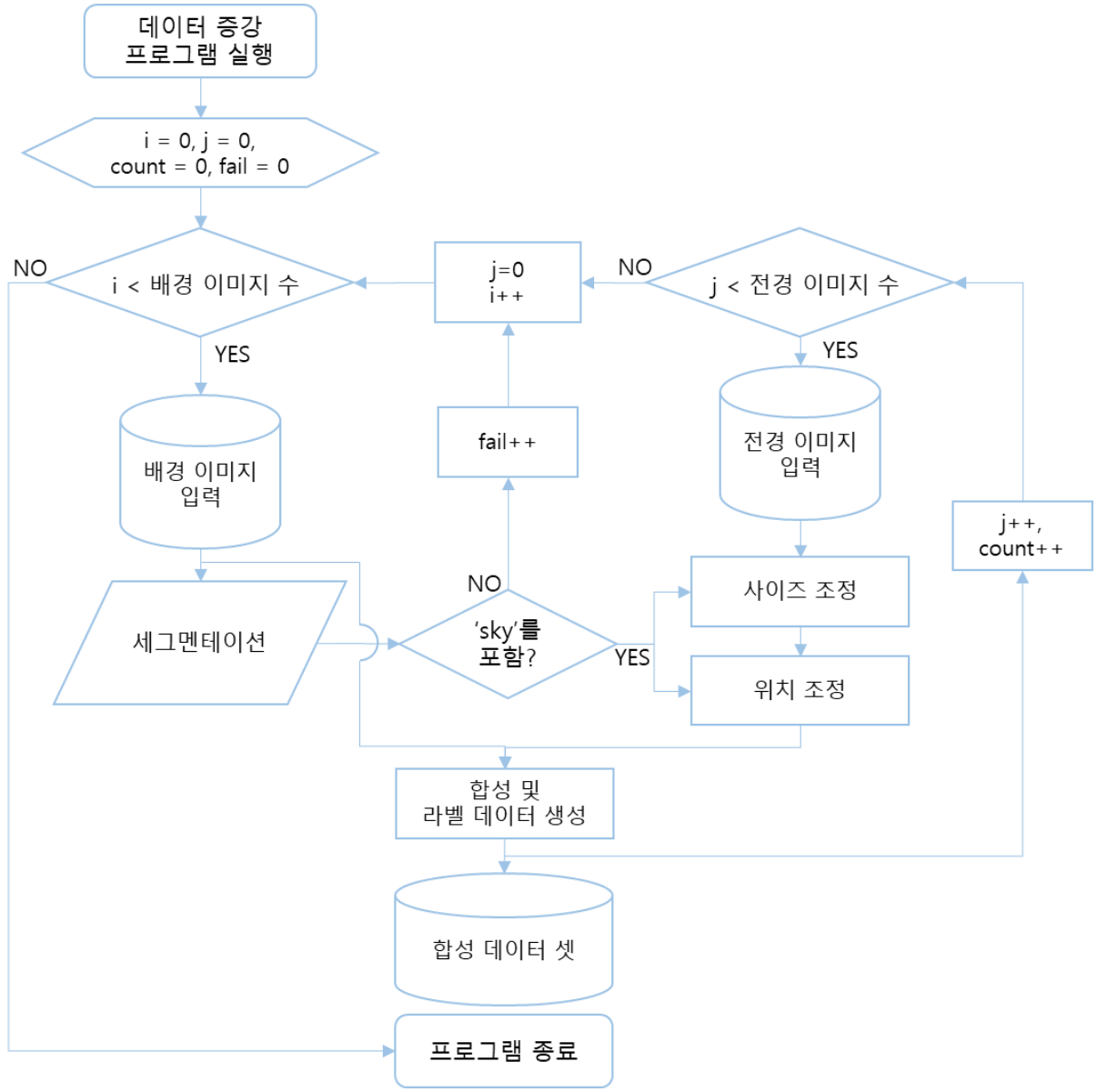

그림 16: 프로그램 순서도.

본 과제물의 1.2.1~1.2.6 절에서 정의한 사용자 정의 함수와 메인 함수를 기반으로 프로그램의 입력, 처리, 판단, 결과를 나타내어 그림 16과 같은 프로그램 순서도를 작성할 수 있다. 입력부는 배경 이미지, 전경 이미지를 입력으로 하며, 처리부는 세그멘테이션, 사이즈 조정, 위치, 조정, 합성 및 라벨데이터 생성 및 i, j, count, fail 변수 값 조정을 담당한다. 판단부는 조건에 따라 데이터를 참과 거짓으로 판별한다. 특히 세그멘테이션 결과 Mask의 내용을 검사하거나 반복문의 횟수를 검사하는 역할을 하여 알맞은 단계의 프로세스를 실행시킨다. 결과부는 합성 데이터와 라벨 데이터를 모아 저장소에 저장하는 역할을 한다.

$$\left(\textit{예상 반복 횟수}\right) = \textit{입력 배경 이미지 수} \times \textit{입력 전경 이미지 수} - \textit{배경에 하늘이 검출되지 않은 횟수}$$

프로그램이 이중 반복문을 빠져나와 프로그램을 종료하기까지 예상 반복 횟수는 위와 같다.

2. 본론

2.0 클래스 불균형 문제

신호등 예시에서의 클래스 불균형 문제를 확인해보고자 공개 데이터인 ETRI 신호등 데이터 1만장 샘플[5]을 받아 분석하고 라벨 데이터 수를 히스토그램으로 시각화해보고자 한다.

```
import os
import numpy as np
import matplotlib.pyplot as plt

basedir ='01 Traffic light (Sample)'
labeldir = 'labels_class_5'
histogram = np.zeros(5)
for k in range(0, 4):
    if (k==0):
        dir = basedir+'/20170110_01_Daejeon/'+labeldir
    elif (k==1):
        dir = basedir+'/20170117_01_Daejeon/'+labeldir
    elif (k==2):
        dir = basedir+'/20170208_01_Deajeon_Sejong/'+labeldir
    elif (k==3):
        dir = basedir+'/20190417_01_Seoul/'+labeldir

    items = os.listdir(dir)
    for i in range(0,len(items)):
        format = os.path.splitext(items[i])
        if format[1]==".txt":
            new_lines=[]
            filedir = dir+'/'+items[i]
            fd = open(filedir, 'r')
            lines = fd.readlines()
            if not lines:
                fd.close()
                #print(filedir)
                continue
            for line in lines:
                line_arr = line.split('\t')
                categor = int(line_arr[4])
                if (categor==1301 or categor==1401): #빨강
                    index = 0
                elif (categor==1300 or categor == 1400): #초록
                    index = 1
                elif (categor==1302 or categor == 1402): #주황
                    index = 2
                elif (categor==1403): #4구 빨 좌
                    index = 3
```

```
                elif (categor==1405): #4구 좌 직
                    index = 4
                else: continue
                histogram[index] = histogram[index] + 1
            fd.close()

print("histogram :", histogram)
x_data = ['Red', 'Green', 'Yellow', 'Red+Left','Green+Left']
plt.title("Number of bounding boxes by class distribution", fontsize=18
)
plt.xlabel("Class")
plt.ylabel("Nuber of bounding boxes")
plt.bar(x_data, histogram)
plt.show()
```

그림 17: 데이터 셋 분석 및 시각화 코드

그림 17은 개방 데이터의 라벨 데이터 파일(텍스트 파일)을 읽고 클래스 ID를 참고하여 histogram 변수에 저장한 뒤 시각화 하는 스크립트를 보여주고 있다. 라벨 데이터 파일로부터 적색신호, 녹색 신호, 황색 신호, 적색+좌회전 신호, 녹색+좌회전신호의 라벨 데이터 수를 histogram 변수에 저장하여 출력한다. 그 결과는 그림 18과 같다.

```
histogram : [7713. 5534.  295.  330. 1317.]
```

그림 18. 라벨 데이터 분석 결과

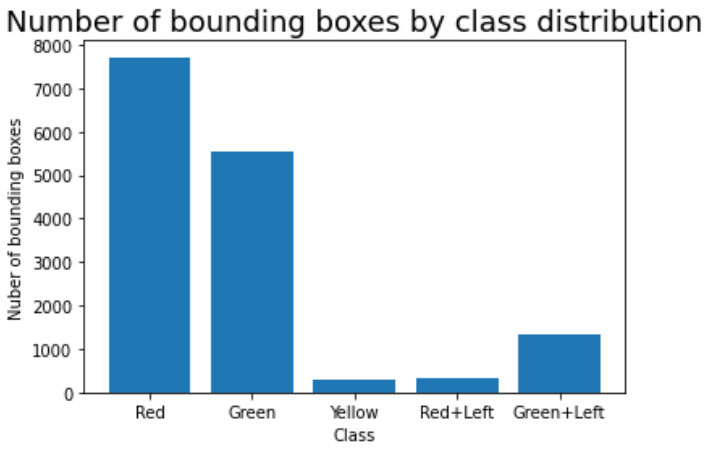

그림 19: 클래스 별 라벨 데이터 분포

그림 19는 ETRI 신호등 데이터 셋의 분석 결과를 시각화 한 것으로 x축은 카테고리 명, y축은 각 클래스 별 라벨 데이터의 수를 나타낸다. 적색 신호 클래스와 녹색 신호 클래스에 비해 나머지 클래스의 라벨 데이터 수는 현저히 낮은 수준인데, 1.1에서 언급했던 대로 클래스 불균형이 흔한 문제임을 확인할 수 있다. 특히, 총 1만 장의 영상 중에서 황색 신호가 촬영되어 라벨링 된 수는 단 295개임을 확인할 수 있고 데이터 증강이 반드시 필요한 경우라 볼 수 있다. 따라서 본 과제에서는 이렇게 흔한 클래스 불균형 문제를 해소하고자 데이터 증강 프로그램을 고안하여 합성 데

이터셋을 획득하고, 단독으로 모델에 학습시켜본 뒤 합성 데이터셋이 실제로 효과가 있는지 여부를 확인할 것이다.

2.1 합성 데이터 셋 획득 및 학습

본 절에서는 데이터 증강 프로그램을 구동하여 획득한 합성 데이터 셋을 바탕으로 학습한 YOLOv4 모델을 분석하여 합성 데이터 셋이 실제로 효과가 있는지 여부를 확인할 것이다.

```
0 0.402979 0.117333 0.045894 0.058667
1431
0 0.242351 0.082667 0.05153 0.042667
1432
0 0.752818 0.12 0.030596 0.058667
1433
0 0.80314 0.049333 0.016908 0.024
1434
0 0.180757 0.088 0.041063 0.042667
1435
0 0.226248 0.06 0.033816 0.029333
1436
0 0.035427 0.12 0.070853 0.058667
1437
0 0.415459 0.061333 0.027375 0.032
1438
Done.
Success: 1438
No sky found: 162
```

그림 20: 프로그램 결과 화면. 라벨 데이터와 합성 횟수

그림 20은 KITTI 데이터셋으로부터 추출한 200장의 배경 사진과 직접 제작한 8장의 전경 사진을 바탕으로, 직접 설계한 프로그램을 통해 총 1438 쌍의 합성 데이터 셋을 획득한 모습을 보여주고 있다.

그림 21: 합성 결과

최종적으로 그림 21과 같은 이미지와 라벨 데이터 1438 쌍이 저장되었다.

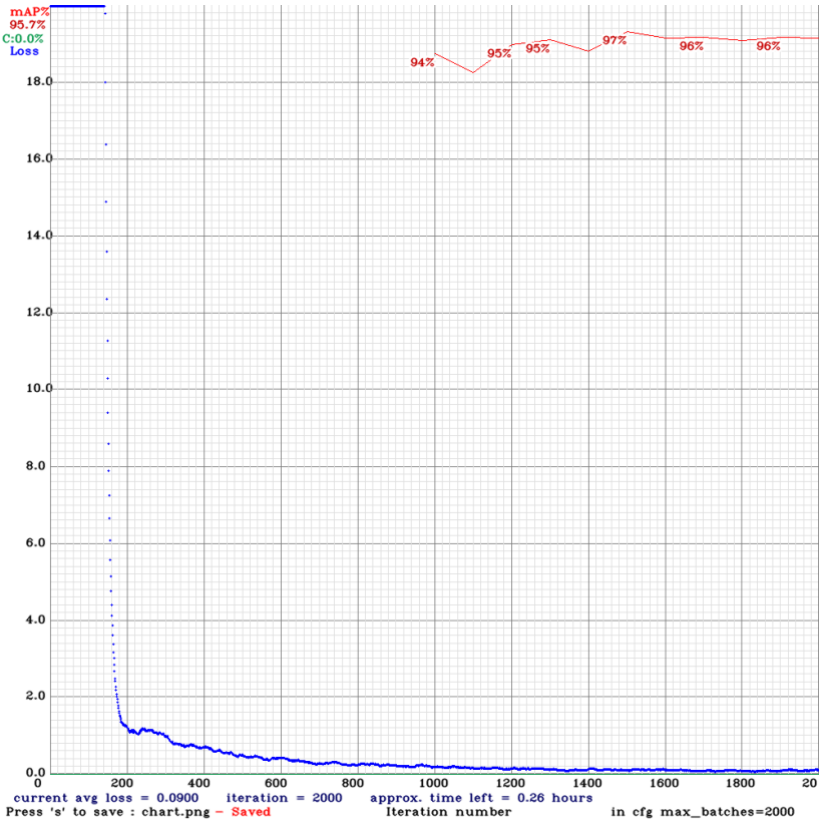

그림 22: YOLOv4 학습 차트

합성 데이터셋이 실제로 효과가 있는지 여부를 확인하기 위해 2.1에서 얻은 데이터 셋을 938 장 대 250 장 비율로 Train Set과 Validation Set으로 나누어 YOLOv4[4] 모델 기반 학습을 시킨 뒤, 객체 검출 테스트를 수행했다. 학습에 사용된 네트워크 해상도는 608x608이며 총 128,000 번의 학습이 이루어졌다. 모델의 평균 손실율은 약 0.0900이고 mAP는 약 95.7%의 결과를 얻었다.

그림 23: 모델 추론 결과

그림 23은 합성 데이터만으로 학습된 딥러닝 모델을 바탕으로 실제 신호등에 대해 객체인식을 수

행한 결과 문제없이 작동하는 모습을 보여주고 있다. 이는 데이터 증강 프로그램으로부터 획득한 합성 데이터 셋만으로 딥러닝 학습이 가능하고 객체 인식에 문제가 없는 것을 확인했으므로 모델 성능에 도움을 줄 수 있다는 의미로 해석할 수 있다.

3. 결론

본 과제에서는 클래스 불균형이 흔한 문제임을 확인하고, 쉽게 구할 수 있는 배경 이미지와 불균형 클래스 전경 이미지로부터 합성 데이터 셋 자동 생성 시스템을 구축하여 클래스 불균형 문제를 해소하는 과정을 보였다. 실험 결과, 합성 데이터 셋만으로 훈련된 딥러닝 모델이 문제없이 작동하는 모습을 보였는데, 이것은 합성 데이터가 실제로 학습 데이터로써 가치가 있다는 것을 의미했다. 즉, 부족한 데이터를 데이터 합성이라는 기법을 통해 클래스 불균형 문제를 해소하여 딥러닝 모델의 객체 인지 성능을 향상시키고 동시에 자동화를 통해 데이터 셋 구축 비용 절감을 기대할 수 있음을 보인 것이다.

4. 고찰

평소 의문이었지만 해결하지는 못했던, '합성데이터가 딥러닝에 정말 도움이 될까?'에 대한 의문을 전공 개념과 데이터 분석 기법을 결합하여 이미지 데이터 분석으로 해결하게 되면서 정말 재밌고 유익한 시간이 되었다고 생각한다. 특히 이전에는 잘 다루지 못했던 넘파이, 파일 시스템, 그래프 시각화를 다룰 줄 알게 되면서 이후 다른 프로그램 작성에 자신감이 생겼다.

[참고문헌]

1. Geiger, A., Lenz, P., Stiller, C., & Urtasun, R. (2013). Vision meets Robotics: The KITTI Dataset. International Journal of Robotics Research (IJRR).
2. Xie, E., Wang, W., Yu, Z., Anandkumar, A., Alvarez, J. M., & Luo, P. (2021). SegFormer: Simple and efficient design for semantic segmentation with transformers. Advances in Neural Information Processing Systems, 34.
3. Zhou, B., Zhao, H., Puig, X., Xiao, T., Fidler, S., Barriuso, A., & Torralba, A. (2019). Semantic understanding of scenes through the ade20k dataset. International Journal of Computer Vision, 127(3), 302-321.
4. Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). Yolov4: Optimal speed and accuracy of object detection. arXiv preprint arXiv:2004.10934.

5. This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No. 2017-0-00068, A Development of Driving Decision Engine for Autonomous Driving(4th) using Driving Experience Information / 2018-0-00327, Development of Fully Autonomous Driving Navigation AI Technology in high-precision map shadow environment)