**# Project 2팀 모델 학습 테스트 이력**

- 모델 테스트 이력과 모델파일은 합쳐서 작성할 것

1. 학습용 데이터 셋

학습에 사용된 데이터셋의 구성은 다음과 같다

|  |  |  |  |
| --- | --- | --- | --- |
| 학습데이터셋 구성(417010장) | | | |
| train(375110장) | | validation(41900장) | |
| 주간 | 야간 | 주간 | 야간 |
| 262162 장 | 112948장 | 27700장 | 14200장 |
| 70% | 30% | 66% | 34% |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 학습 데이터 셋 수집 환경 분포 | | | | | | |
| 구분 | 주야 | | 날씨 | | 조명 | |
| 상태 | 주간 | 야간 | 정상 | 눈/비/안개 | 정상 | 역광 |
| 비율 | 70% | 30% | 94% | 6% | 87% | 13% |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 차선의 세부 속성 분포 | | | | | | | |
| 객체 | | | 선 종류 | | 색상 | | |
| 정지선 | 횡단보도 | 차선 | 실선 | 점선 | 황색 | 흰색 | 청색 |
| 6.5% | 8.5% | 85% | 57% | 43% | 14% | 83% | 3% |

학습용 데이터셋 파일 및 폴더 구성

* ../c\_1280\_720\_daylight\_train\_1 ~ ../c\_1280\_720\_daylight\_train\_8
  + /image
    - 번호.jpg
  + /json
    - 번호.json
* ../c\_1280\_720\_night\_train\_1 ~ ../c\_1280\_720\_night\_train\_4
  + /image
    - 번호.jpg
  + /json
    - 번호.json
* ../c\_1280\_720\_daylight\_validation\_1
  + /image
    - 번호.jpg
  + /json
    - 번호.json
* ../c\_1280\_720\_night\_validation\_1
  + /image
    - 번호.jpg
  + /json
    - 번호.json

이중 임의의 3000장의 이미지를 학습에 사용하였고 적합한 하이퍼파라미터를 찾아 해당 모델에 전체 데이터를 학습시켰다.

Table epoch

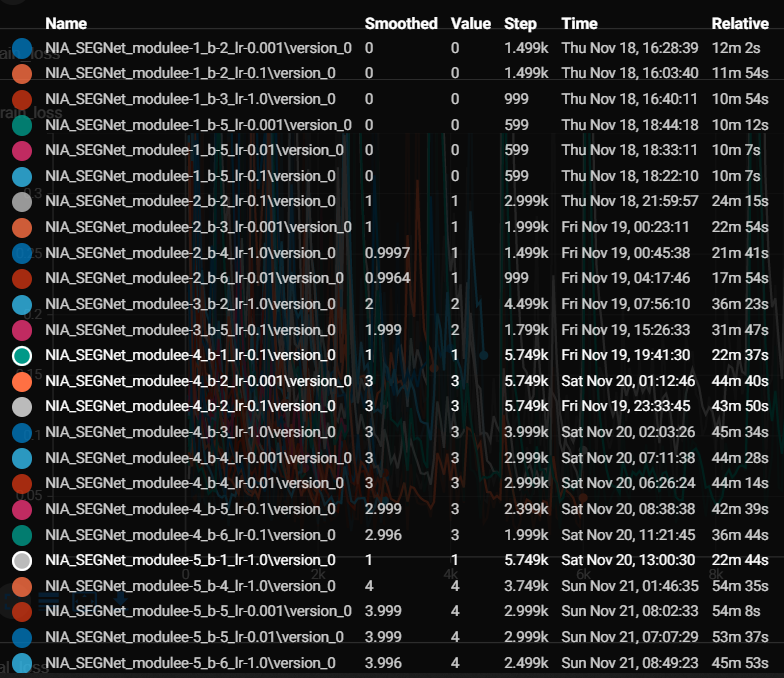
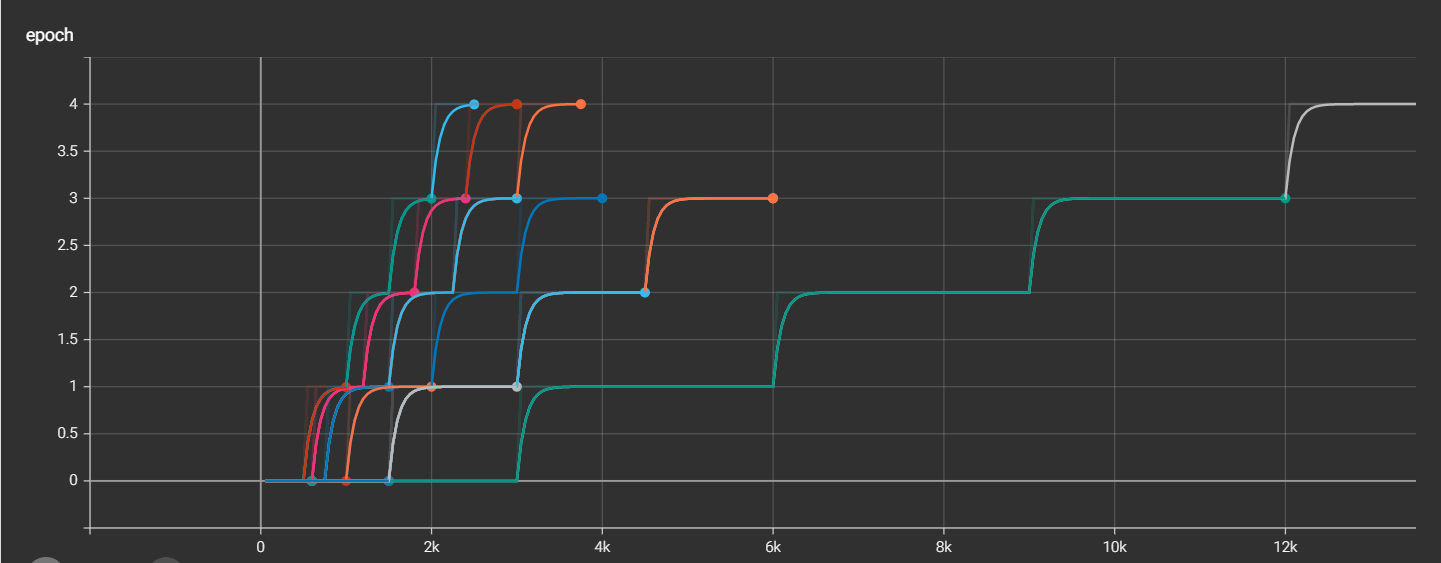


Table train\_loss

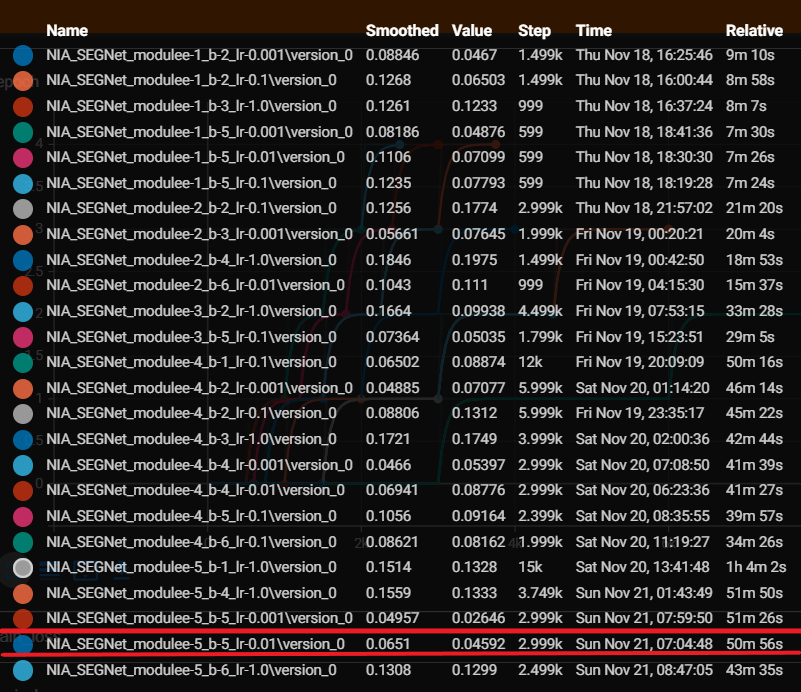
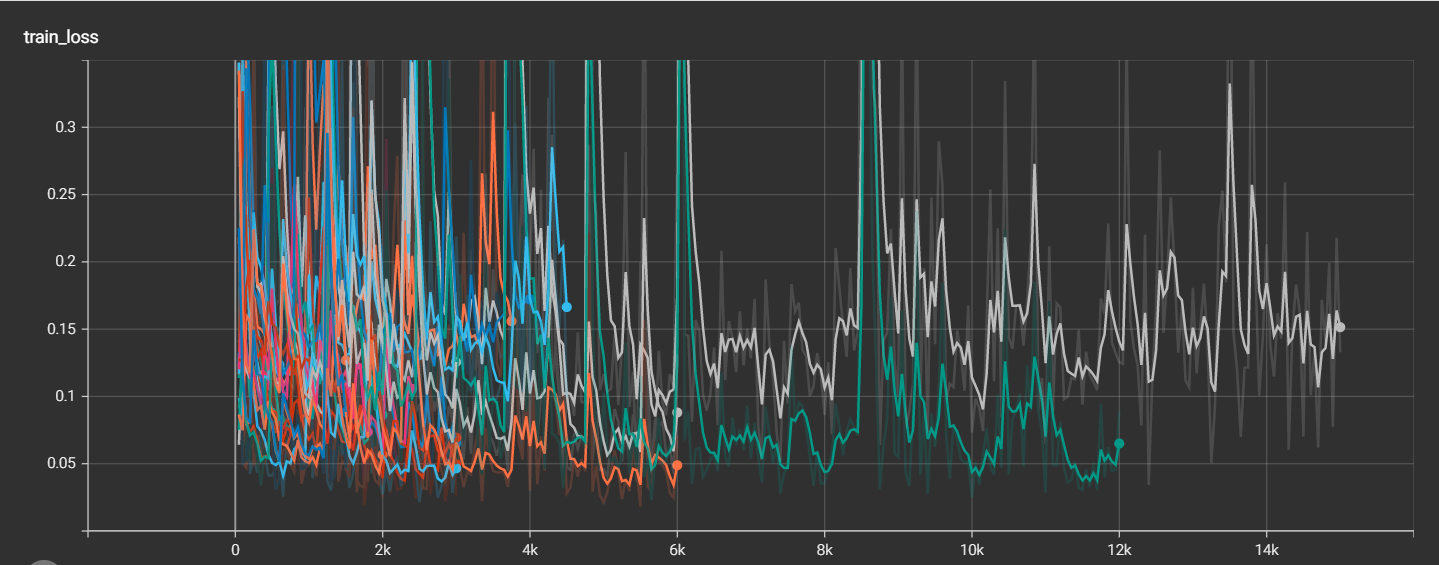


Table val\_loss

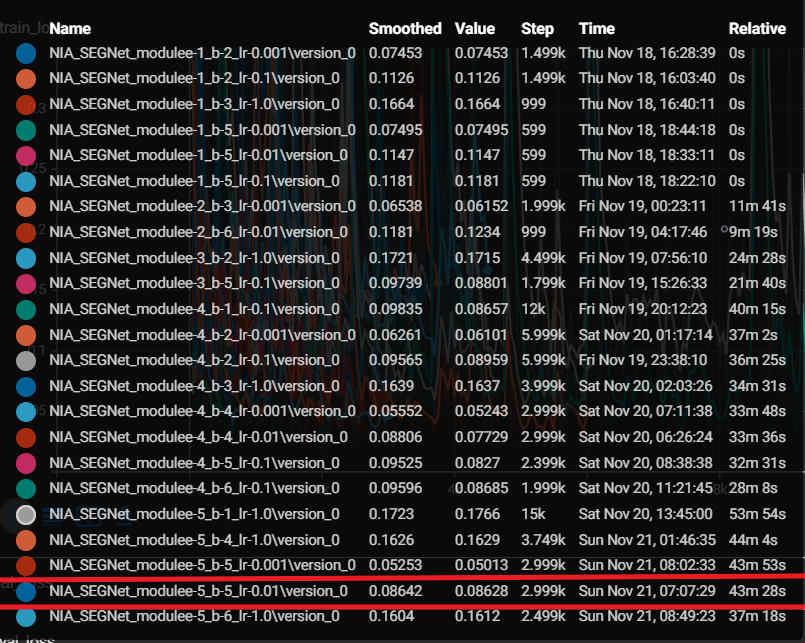
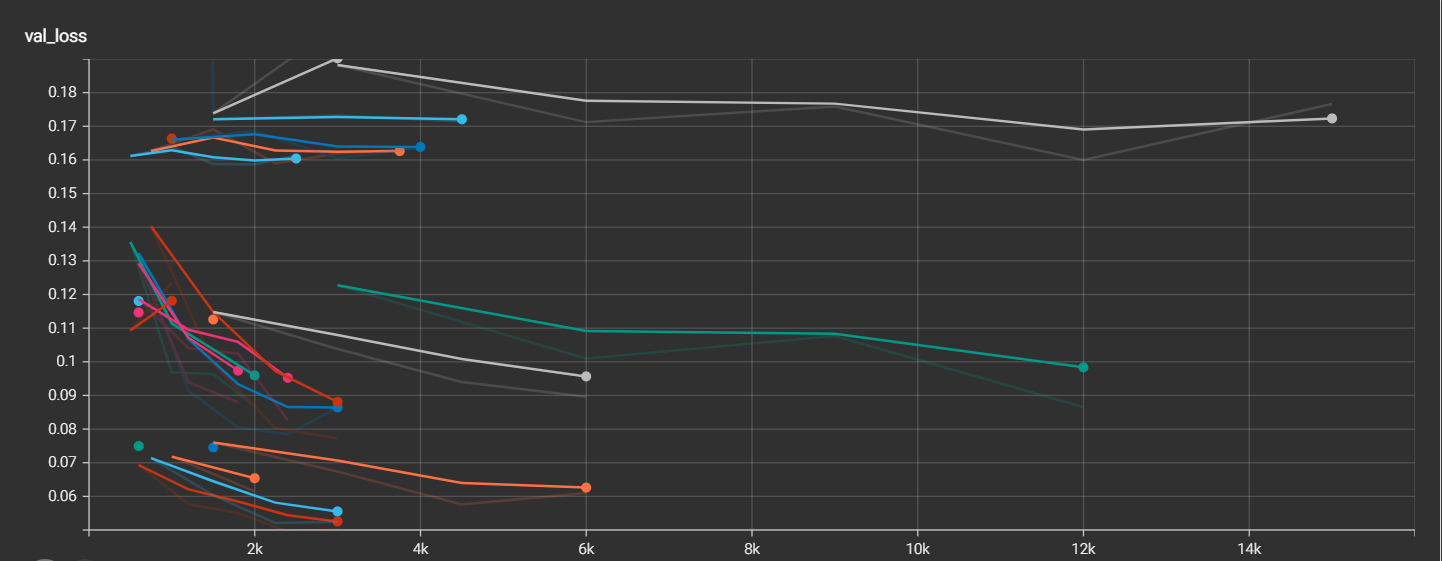
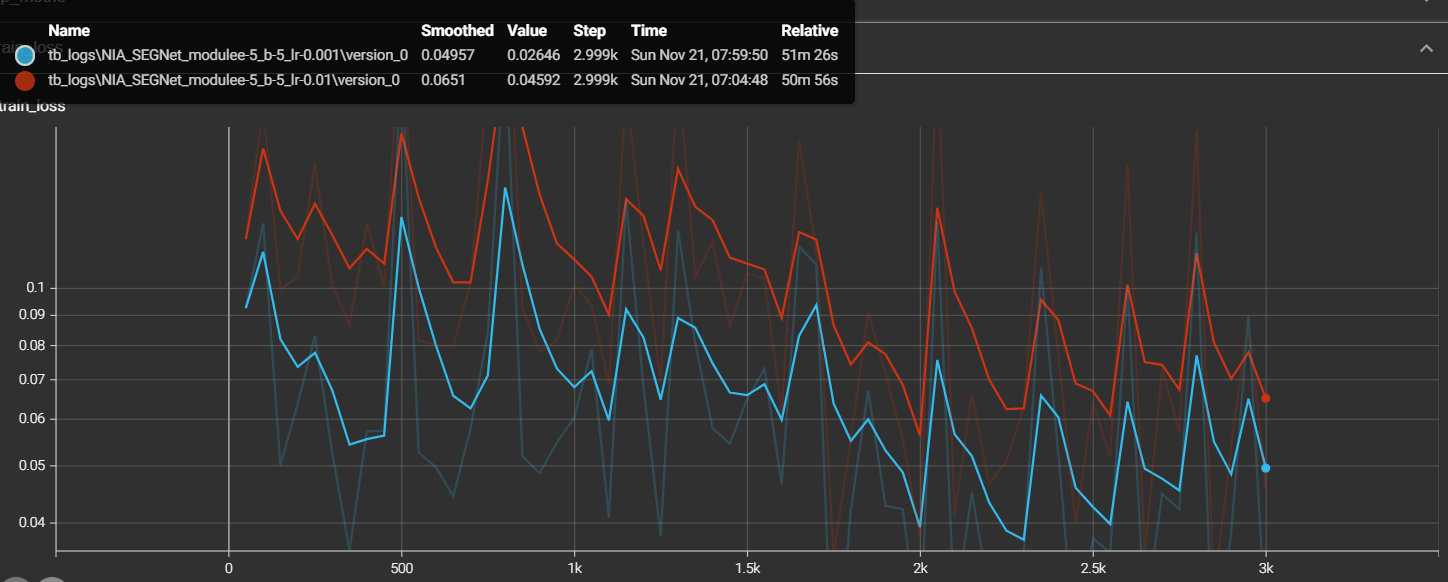
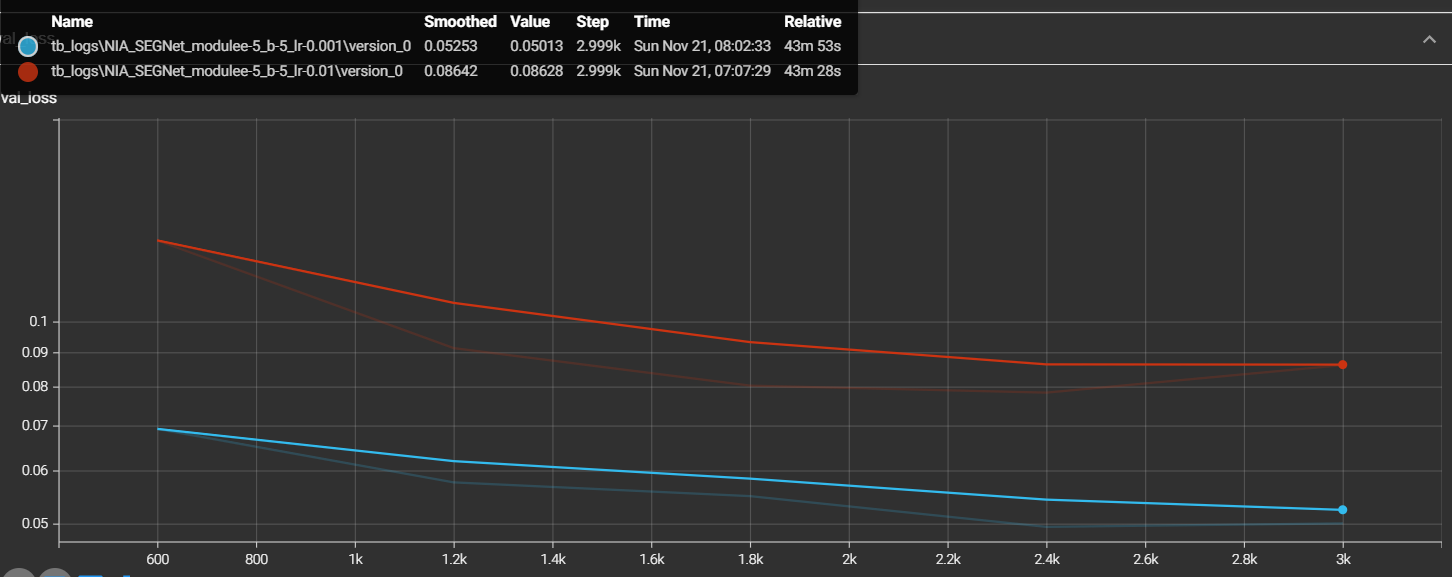


Figure 에폭5 배치사이즈5 학습률 0.01, 0.001 인 두 모델





사용될 모델은 에폭이 5, 배치사이즈가 5, 학습률이 0.001인 (파란색) 모델로 결정됨.

학습에 사용된 코드

arg\_parse.py

import argparse

parser = argparse.ArgumentParser(description='This is argparse example python program.')

parser.add\_argument('-f')

# 2. add arguments to parser

parser.add\_argument(

    '--epochs', '-e',

    type=int,

    default=1,

    help='Number of epoch to train.')

parser.add\_argument(

    '--batch\_size', '-b',

    type=int,

    default=1,

    help='Size of batch')

parser.add\_argument(

    '--learning\_rate', '-lr',

    type=float,

    default=0.1,

    help='Learning rate'

)

# 3. parse arguments

args = parser.parse\_args()

# 4. use arguments

print(args)

print('epochs :',args.epochs)

print('batch\_size :',args.batch\_size)

print('learning\_rate : ', args.learning\_rate)

e = args.epochs

b = args.batch\_size

lr = args.learning\_rate

hyp = "e-" + str(e)+ '\_' + "b-" + str(b) +'\_'+ "lr-" + str(lr)

print (hyp)

datasets.py

import sys

sys.path=[x for x in sys.path if not "python2.7" in x]

import os

import random

import numpy as np

import ujson as json

import matplotlib.pyplot as plt

from torch.utils.data import Dataset

from PIL import Image

from torchvision import transforms

import torch

# import sys

# if '/opt/ros/kinetic/lib/python2.7/dist-packages' in sys.path:

#     sys.path.remove('/opt/ros/kinetic/lib/python2.7/dist-packages')

import cv2

class LaneDataset(Dataset):

    I\_H = 512

    I\_W = 1024

    def \_\_init\_\_(self, data\_path):

        if not os.path.exists(data\_path):

            pths = ['/home/ubuntu/aiml/lane\_model/']

            if not os.path.exists("data"):

                os.mkdir("data")

            with open("data/train.txt","w") as train, open("data/val.txt","w") as val, open("data/test.txt","w") as test:

                for pth in pths: # data/.txt에 모델 과정별로 입력될 사진자료 디렉토리 주소 입력

                    for i in range(1,51): # 1~50 까지 만들어진 폴더를 정해진 규칙에 따라 txt파일에 써줌

                        if i % 10 == 1:

                            val.write(pth+str(i)+"\n")

                        elif i % 10 == 2:

                            test.write(pth+str(i)+"\n")

                        else:

                            train.write(pth+str(i)+"\n")

        with open(data\_path) as txt:

            lines = txt.readlines()  # .txt파일 내용 읽어오기

        self.img\_list\_1 = [] # .txt 경로를 따라가서 이미지 파일리스트를 저장할 리스트 변수 생성.

        self.img\_list = []

        for line in lines:

            self.img\_list\_1 += [line.strip()+f for f in os.listdir(line.strip()) if ".jpg" in f or '.png' in f]

        for i in range(3000) :

            index = random.randrange(0, len(self.img\_list\_1)-1)

            self.img\_list += [self.img\_list\_1[index]] # 대괄호로 안감싸주면  ㅂ,ㅈ,ㄷㄱ,ㅅ,ㅂ, 이렇게 저장됨

        # print([f for f in self.img\_list if not os.path.exists(f.replace('.jpg','.json').replace('.png','.json'))])

        self.len = len(self.img\_list)

        self.data\_path = data\_path

    def \_\_getitem\_\_(self, index): # 입력된 index에 해당하는 이미지를 출력

        while True:

            img\_path = self.img\_list[index]

            try:

                img = Image.open(img\_path)

                break

            except:

                with open("error\_files.txt",'a') as errlog:

                    errlog.write(img\_path+'\n')

                    index = index + 1

        w, h = img.size

        label\_path = img\_path.replace("image","json").replace('.jpg','.json').replace('.png','.json')

        with open(label\_path) as json\_file:

            json\_data = json.load(json\_file)

        img\_tensor = transforms.functional.to\_tensor(transforms.functional.resized\_crop(img, h-w//2, 0, w//2, w, (self.I\_H,self.I\_W)))

        target\_map = self.make\_gt\_map(json\_data, w, h)

        return img\_tensor, torch.LongTensor(target\_map), img\_path

    def \_\_len\_\_(self):

        return self.len

    def make\_gt\_map(self, json\_data, original\_w, original\_h):

        target\_map = np.zeros((self.I\_H, self.I\_W),dtype=np.int32)

        annotation = json\_data["annotations"]

        y\_offset = original\_h - original\_w // 2

        for item in annotation:

            obj\_class = item["class"]

            if obj\_class == "traffic\_lane":

                pos = item["data"]

                poly\_points = np.array([([pt["x"]\*self.I\_W/original\_w, (pt["y"] - y\_offset)\*self.I\_H/(original\_h-y\_offset)]) for pt in pos]).astype(np.int32)

                cv2.polylines(target\_map, [poly\_points], False, 1,10)

            if obj\_class == "stop\_line":

                pos = item["data"]

                poly\_points = np.array([([pt["x"]\*self.I\_W/original\_w, (pt["y"] - y\_offset)\*self.I\_H/(original\_h-y\_offset)]) for pt in pos]).astype(np.int32)

                cv2.polylines(target\_map, [poly\_points], False, 2,10)

            if obj\_class == "crosswalk":

                pos = item["data"]

                poly\_points = np.array([([pt["x"]\*self.I\_W/original\_w, (pt["y"] - y\_offset)\*self.I\_H/(original\_h-y\_offset)]) for pt in pos]).astype(np.int32)

                if len(poly\_points) == 0:

                    continue

                cv2.fillPoly(target\_map, [poly\_points], 3)

        return target\_map

if \_\_name\_\_ == "\_\_main\_\_":

    ld = LaneDataset(data\_path='data/val.txt')

    # ss = torch.zeros(4)

    # cnt = 0

    #

    for i,(img, target,path) in enumerate(ld):

        print(i)

        plt.imshow(img.permute((1,2,0)))

        plt.savefig("a.png")

        plt.imshow(target,vmax =3)

        plt.savefig("b.png")

        input()

module.py

import os

import torch

from torch import nn

import torch.nn.functional as F

from torchvision import transforms

from torch.utils.data import DataLoader

import pytorch\_lightning as pl

from datasets import LaneDataset

import matplotlib.pyplot as plt

from torchvision.models.segmentation import fcn\_resnet50

import numpy as np

from focal\_loss import FocalLoss

import datetime

import sys

import arg\_parse

import torchvision

class NIA\_SEGNet\_module(pl.LightningModule):

    def \_\_init\_\_(self):

        super(NIA\_SEGNet\_module, self).\_\_init\_\_()

        # self.save\_hyperparameters()

        self.fcn = fcn\_resnet50(pretrained=True)

        in\_channels = 2048

        inter\_channels = in\_channels // 4

        channels = 4

        dropout = 0.1

        self.fcn.classifier = nn.Sequential(

            nn.Conv2d(in\_channels, inter\_channels, 3, padding=1, bias=False),

            nn.BatchNorm2d(inter\_channels),

            nn.ReLU(),

            nn.Dropout(dropout),

            nn.Conv2d(inter\_channels, channels, 1)

        )

        self.f1 = 0

        self.f1cnt = 0

    def forward(self, x):

        out = self.fcn(x)

        return out

    def custom\_histogram\_adder(self):

        # iterating through all parameters

        for name,params in self.named\_parameters():

            self.logger.experiment.add\_histogram(name,params,self.current\_epoch)

    def training\_step(self, batch, batch\_idx):

        loss = self.get\_loss(batch)

        self.log('train\_loss', loss)

        return loss

    def get\_loss(self,batch):

        x, y, \_ = batch

        out = (self(x)['out'])

        ltype = "default"

        if ltype == "sqrt-frq":

            frequency\_weight = torch.Tensor([0.03055164, 0.15936654, 0.60941457, 0.20066725]).to(self.device)

            loss = F.cross\_entropy(out, y, weight=frequency\_weight)

        elif ltype == "focal":

            fl = FocalLoss()

            loss = fl(out, y)

        else:

            loss = F.cross\_entropy(out, y)

        return loss

    def test\_step(self, batch, batch\_idx):

        x, y, img\_path = batch

        out = torch.sigmoid(self(x)['out'])

        confusion\_mat = torch.zeros((4,4), device=self.device,dtype=torch.long)

        f1\_sum = 0

        f1\_cnt = 0

        # print(batch\_idx)

        acc = torch.tensor(0.0,device=self.device)

        # imshow = False

        imshow = True

        if imshow:

            for i, output in enumerate(out):

                final\_out = torch.argmax(output,0)

                img = x[i].cpu().permute((1,2,0)).numpy()

                # img = img[:,:,::-1]

                plt.imsave("input.png", img)

                plt.imsave("output.png", (final\_out.cpu()).int(),vmin=0,vmax=3)

                plt.imsave("target.png", (y[i].cpu()).int(),vmin=0,vmax=3)

                input()

        else:

            for i, output in enumerate(out):

                final\_out = torch.argmax(output,0)

                acc += torch.sum((final\_out==y[i]))/(512\*1024.0)

                for xx in torch.arange(4, device=self.device):

                    for yy in torch.arange(4, device=self.device):

                        confusion\_mat[xx,yy] += torch.sum((final\_out==xx)\*(y[i]==yy))

                aa,bb,cnt = 0,0,0

                for ii in range(4):

                    if torch.sum(confusion\_mat[ii,:]) !=0 and torch.sum(confusion\_mat[:,ii]) != 0:

                        aa += confusion\_mat[ii,ii]/torch.sum(confusion\_mat[ii,:]).float()

                        bb += confusion\_mat[ii,ii]/torch.sum(confusion\_mat[:,ii]).float()

                        cnt += 1

                aa /= cnt

                bb /= cnt

                # self.f1 += (2\*aa\*bb/(aa+bb))

                # self.f1cnt += 1

                f1 = (2\*aa\*bb/(aa+bb)).item()

                f1\_sum += f1

                f1\_cnt += 1

                print(img\_path, "F1 measure :", f1)

                file\_output = False

                # file\_output = True

                if file\_output:

                    if not os.path.exists("./output/"):

                        os.mkdir("./output/")

                    img = x[i].cpu().permute((1,2,0)).numpy()

                    folder\_path = "./output/"+str(batch\_idx\*self.batch\_size + i)

                    if not os.path.exists(folder\_path):

                        os.mkdir(folder\_path)

                    plt.imsave(folder\_path+"/input.png", img)

                    plt.imsave(folder\_path+"/output.png", final\_out.cpu()\*255/3)

                    plt.imsave(folder\_path+"/target.png", y[i].cpu())

            acc /= len(out)

            return confusion\_mat, f1\_sum, f1\_cnt

    def test\_epoch\_end(self,outputs):

        sum\_confusion\_mat = 0

        total\_f1 = 0

        total\_f1\_cnt = 0

        for confusion\_mat, f1\_sum, f1\_cnt in outputs:

            sum\_confusion\_mat += confusion\_mat

            total\_f1 += f1\_sum

            total\_f1\_cnt += f1\_cnt

        if total\_f1\_cnt > 0:

            print("total\_f1\_cnt",total\_f1\_cnt)

            print("average F1 measure", total\_f1/total\_f1\_cnt)

            print("total confusion matrix:\n", sum\_confusion\_mat.cpu().numpy())

        print("total\_f1\_cnt",total\_f1\_cnt)

        print("average F1 measure", total\_f1/total\_f1\_cnt)

        print("total confusion matrix:\n", sum\_confusion\_mat.cpu().numpy())

    def configure\_optimizers(self):

        optimizer = torch.optim.Adam(self.parameters(), lr=arg\_parse.args.learning\_rate)

        return optimizer

    def validation\_step(self, batch, batch\_idx):

        loss =  self.get\_loss(batch)

        self.log\_dict({'val\_loss': loss})

        return loss

    def validation\_epoch\_end(self, outputs):

        sum\_loss = 0

        for loss in outputs:

            sum\_loss += loss

    def val\_dataloader(self):

        dataset = LaneDataset(data\_path='/home/ubuntu/aiml/lane\_model/data/val.txt')

        train\_loader = DataLoader(dataset, batch\_size = self.batch\_size, num\_workers=28)

        return train\_loader

    def test\_dataloader(self):

        dataset = LaneDataset(data\_path='/home/ubuntu/aiml/lane\_model/data/sample.txt')

        train\_loader = DataLoader(dataset, batch\_size = self.batch\_size, num\_workers=12)

        return train\_loader

    def train\_dataloader(self):

        dataset = LaneDataset(data\_path='/home/ubuntu/aiml/lane\_model/data/train.txt')

        train\_loader = DataLoader(dataset, batch\_size = self.batch\_size, num\_workers=12,shuffle=True)

        #    DataLoader(dataset, batch\_size=1, shuffle=False, sampler=None,

        #    batch\_sampler=None, num\_workers=0, collate\_fn=None,

        #    pin\_memory=False, drop\_last=False, timeout=0,

        #    worker\_init\_fn=None)

        return train\_loader

train.py

import logging

from torch.utils import tensorboard

from pytorch\_lightning.loggers import TensorBoardLogger

import arg\_parse

import datetime

import platform

import psutil

# import cpuinfo

from module import NIA\_SEGNet\_module

import pytorch\_lightning as pl

import torch

import random

import time

from pytorch\_lightning.callbacks.early\_stopping import EarlyStopping

import shutil

from torch.utils.tensorboard import SummaryWriter

import numpy as np

#텐서보드 로거 객체 생성

hyp\_val=arg\_parse.hyp

logger  = TensorBoardLogger('tb\_logs', name='NIA\_SEGNet\_module'+hyp\_val, default\_hp\_metric=False)

#tensor board

writer = SummaryWriter('/home/ubuntu/aiml/lane\_model/src\_test/lightning\_logs')

writer.close()

param = arg\_parse.args

torch.manual\_seed(777)

random.seed(777)

model = NIA\_SEGNet\_module()

model.batch\_size =param.batch\_size

# print(model.fcn)

# print(flush=True)

trainer = pl.Trainer(gpus=[0],prepare\_data\_per\_node=True, distributed\_backend="ddp", callbacks=[EarlyStopping(monitor='val\_loss', patience=2)],  max\_epochs=param.epochs, logger=logger)

trainer.fit(model)

모델 하이퍼 파라미터의 변경 방법과 파라미터 별 결과

lane\_detect\_train\_test.bash를 실행 시켜서 매 반복마다 파라미터를 변경하여 train.py를 실행시킴

Table lane\_detect\_train\_tset.bash

|  |
| --- |
| lrate="1 0.1 0.01 0.001"  for ((e=1; e<=5; e++))  do  for ((b=1; b<=6; b++))  do  for lr in $lrate  do  python /home/ubuntu/aiml/lane\_model/src\_test/train.py -e=$e -b=$b -lr=$lr  done  done  done |