결과 보고서

임베디드응용및실습.
13주차 실습 과제

과 목 명	임베디드응용및실습
학 번	2019161068
이 름	백승진
제 출 일	2024년 12월 02일

1. 1m 내외의 코스에서 직진 및 좌/우회전 자율주행이 가능하도록 코딩하시오.

1) 코드

```
# 경고 안내 출력 설정
import os
os.environ['CUDA_VISIBLE_DEVICES'] = '1'

# 필요한 모듈 임포트
import os
import random
import fnmatch
import datetime
import pickle
import cv2
import numpy as np
os.environ["HDF5_USE_FILE_LOCKING"] = "FALSE"

import pandas as pd
import tensorflow as tf
import tensorflow.keras
from tensorflow.keras import regularizers
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Conv2D, MaxPool2D, Dropout, Flatten, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint

from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split

#import colab
#from imgaug import augmenters as img_aug
import matplotlib.pyplot as plt
import matplotlib.image as mping
from PIL import Image

# 데이터 준비
data_dir = "crop_img"
file_list = os.listdir(data_dir)
image_paths = []
steering_angles = []
pattern ="*.png"
for filename in file_list:
    if fnmatch.fnmatch(filename, pattern):
        image_paths.append(os.path.join(data_dir, filename))
        angle = int(filename[-7:-4])
        steering_angles.append(angle)
print('image_paths', image_paths)
print('steering_angles', steering_angles)
image_index = 1
plt.imshow(Image.open(image_paths[image_index]))
print("image_path: ", image_paths[image_index])
print("steering_Angle: ", steering_angles[image_index])
df = pd.DataFrame()
df['ImagePath'] = image_paths
```

```
df['Angle'] = steering_angles

# 직진과 좌/우회전 데이터에 대한 히스토그램 생성
num_of_bins = 10
hist, bins = np.histogram(df['Angle'], num_of_bins)
print(hist)
print(bins)
fig, axes = plt.subplots(1,1,figsize=(10,4))
axes.hist(df['Angle'], bins=num_of_bins, width=1, color='blue')

# 각도를 범주형 데이터로 매핑
s = np.array(steering_angles)
new_angles = np.copy(steering_angles)
new_angles[np.where(s==0)] = 0
new_angles[np.where(s==45)] = 1
new_angles[np.where(s==135)] = 2
print(new_angles)
print(steering_angles)
print(len(np.where(new_angles==0)[0]))
print(len(np.where(new_angles==1)[0]))
print(len(np.where(new_angles==2)[0]))
df['Angle'] = new_angles

# 학습셋과 테스트셋으로 분리
X_train, X_valid, y_train, y_valid = train_test_split(image_paths, new_angles, test_size=0.2)
print("Training data: {}, Validation data : {}".format(len(X_train), len(X_valid)))

fig, axes = plt.subplots(1,2, figsize=(12,4))
axes[0].hist(y_train, bins=num_of_bins, width=1, color='blue')
axes[0].set_title("Training data")
axes[1].hist(y_valid, bins=num_of_bins, width=1, color='red')
axes[1].set_title("Validation data")

# 이미지 경로와 정답 클래스 쌍을 출력
for a, b in zip(X_train, y_train):
    print(a,b)

# 이미지 로드 및 전처리 (정규화)
def my_imread(image_path):
    image = cv2.imread(image_path)
    return image
def img_preprocess(image):
    image = np.float32(image) / 255.
    return image
fig, axes = plt.subplots(1,2,figsize=(15,10))
image_ori = my_imread(image_paths[image_index])
image_processed = img_preprocess(image_ori)
axes[0].imshow(image_ori)
axes[0].set_title("ori")
axes[1].imshow(image_processed)
axes[1].set_title("processed")
print(image_ori.shape)

# 딥러닝 모델 설계
```

```
def neural_net():
    model = Sequential()
    model.add(Conv2D(32, (5,5), input_shape=(66,200,3), activation='relu'))
    model.add(MaxPool2D((2,2)))
    model.add(Dropout(0.2))
    model.add(Conv2D(64, (5,5), activation='relu'))
    model.add(MaxPool2D((2,2)))
    model.add(Dropout(0.2))
    model.add(Conv2D(64, (5,5), activation='relu'))
    model.add(MaxPool2D((2,2)))
    model.add(Dropout(0.2))
    model.add(Conv2D(128, (3,3), activation='relu'))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dropout(0.5))
    model.add(Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
    model.add(Dropout(0.5))
    model.add(Dense(64, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
    model.add(Dropout(0.5))
    model.add(Dense(3, activation='softmax'))

    opt = Adam(lr=1e-3)
    loss_fn = tf.keras.losses.SparseCategoricalCrossentropy()
    model.compile(
        loss='sparse_categorical_crossentropy',
        optimizer=opt
    )

    return model

model = neural_net()
print(model.summary())

# 배치 데이터 생성
def image_data_generator(image_paths, steering_angles, batch_size):
    while True:
        batch_images = []
        batch_steering_angles = []
        len_imgs = len(image_paths)
        for i in range(batch_size):
            random_index = random.randint(0, len_imgs-1)
            image_path = image_paths[random_index]
            image = my_imread(image_path)
            steering_angle = steering_angles[random_index]

            image = img_preprocess(image)
            batch_images.append(image)
            batch_steering_angles.append(steering_angle)
        yield(np.asarray(batch_images), np.asarray(batch_steering_angles))

# 학습셋과 테스트셋의 샘플 시각화
ncol = 2
nrow = 2
```

```
print(len(X_train))

X_train_batch, y_train_batch = next(image_data_generator(X_train, y_train, nrow))
X_valid_batch, y_valid_batch = next(image_data_generator(X_valid, y_valid, nrow))
print(len(X_train_batch))

fig, axes = plt.subplots(nrow, ncol, figsize=(15,6))
fig.tight_layout()

for i in range(nrow):
    axes[i][0].imshow(X_train_batch[i])
    axes[i][0].set_title("training, angle={}".format(y_train_batch[i]))
    axes[i][1].imshow(X_valid_batch[i])
    axes[i][1].set_title("validation, angle={}".format(y_valid_batch[i]))

# 모델 저장용 디렉터리 생성
save_dir = datetime.datetime.now().strftime('%Y%m%d_%H%M')
model_output_dir = os.path.join(save_dir)
os.mkdir(model_output_dir)
print('model_output_dir ', model_output_dir)

# 배치 사이즈 설정
train_bSize = 16
valid_bSize = 128

# 체크포인트 콜백 설정
checkpoint = 'lane_navigation_'+ save_dir + '.h5'
checkpoint_callback = ModelCheckpoint(
    filepath=os.path.join(model_output_dir, checkpoint),
    verbose=1,
    save_best_only=True          # 가장 좋은 모델만 저장
)

# 모델 학습
history = model.fit_generator(
    image_data_generator(X_train, y_train, batch_size=train_bSize),
    steps_per_epoch=len(X_train) // train_bSize,
    epochs=150,
    validation_data=image_data_generator(X_valid, y_valid, batch_size=valid_bSize),
    validation_steps=len(X_valid) // valid_bSize,
    verbose=1,
    shuffle=True,
    callbacks=[checkpoint_callback]  # early_stopping 제거
)

# 모델 학습 히스토리 저장
history_path = os.path.join(model_output_dir, 'history.pickle')
with open(history_path, 'wb') as f:
    pickle.dump(history.history, f, pickle.HIGHEST_PROTOCOL)

# 예측 수행 및 결과 시각화
img_path = X_valid[0]
image = my_imread(img_path)
steering_angle = steering_angles[0]
```

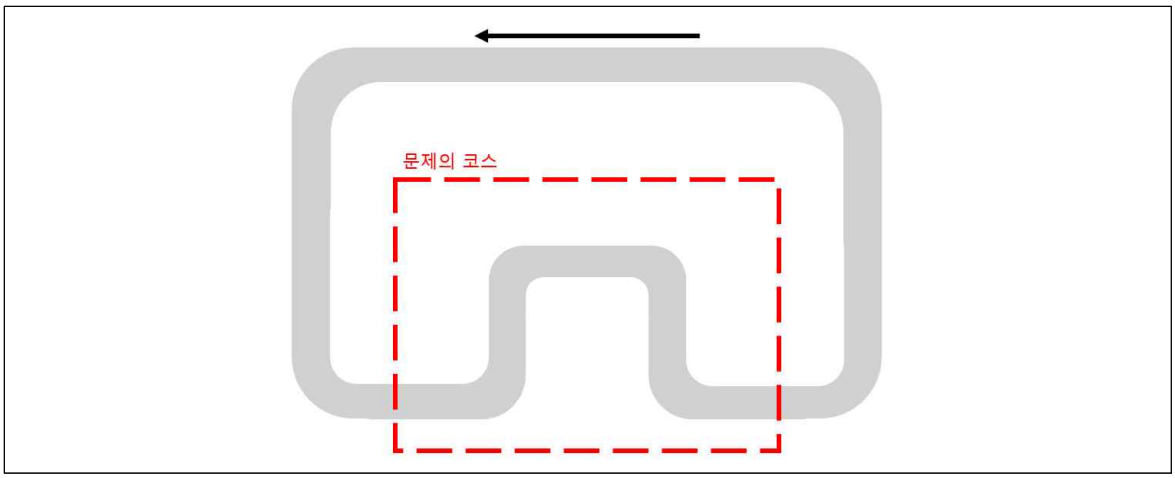
```
image = img_preprocess(image)
print(image.shape)
print(model.predict(np.expand_dims(image, 0))[0])
print(np.argmax(model.predict(np.expand_dims(image, 0))[0]))
plt.imshow(image)

# epochs 별 손실 변화 그래프 시각화
print(model_output_dir)

history_path = os.path.join(model_output_dir, 'history.pickle')
with open(history_path, 'rb') as f:
    history = pickle.load(f)

plt.plot(history['loss'], color='blue')
plt.plot(history['val_loss'], color='red')
plt.legend(['training loss', 'validation loss'])
```

2) 결과

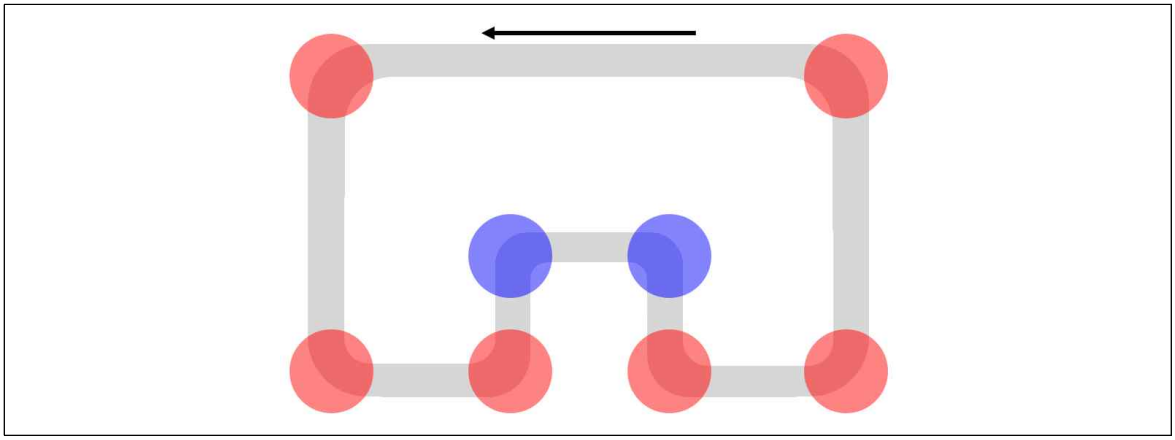
영상 참고

3) 해석

이번 과제에서는 데이터 수집을 위해 키보드 방향키를 이용한 수동 주행을 실시하였습니다. 사전에 받은 **drive_collect_textbook_video.py** 에서는 주행 중 방향키를 누를 때마다 해당 방향키에 매핑된 숫자의 파일명으로 자동 저장되도록 구현되어 있었습니다. 해당 데이터 수집 코드의 경우 방향키 설정 외에 특별히 수정한 점은 없습니다. 최종적으로 데이터는 **총 3,291장** 수집했습니다. 데이터 수집은 몇 차례 반복 진행되었습니다. 처음에는 어떤 조건에서도 동작할 수 있도록 시계/반시계 방향과 안쪽/바깥쪽 코스, 조명환경 등에 변화를 주며 8,000장 가까이 수집했으나, 결과는 직진조차 제대로 하지 못했습니다. 이는 조건에 비해 데이터양이 부족했기 때문이라고 생각하여, 조건에 제한을 두었고, 2차 수집 이후로는 반시계 방향, 바깥쪽 코스, 밝은 환경에서 수집했습니다. 두 번째로 수집한 데이터를 이용하여 만든 모델은 직진과 단순 좌회전 코스에서는 잘 동작했지만, 일부 커브가 잦은 코스에서는 잘 동작하지 않았습니다.



그래서 세 번째 데이터 수집 단계에서는 잘 동작하지 않은 문제의 코스를 중심으로 더 많이 학습시켰습니다. 1차와 2차 데이터를 수집할 때는 특정 코스를 분리하지 않고 전 코스만 몇 차례 반복해서 주행했지만, 3차 데이터를 수집할 때는 전 코스를 세 바퀴 정도 돌고, 문제의 코스를 추가로 10번 정도 주행하여 수집을 완료했습니다. 하지만 이 역시도 같은 코스에서 자율주행에 실패하였습니다. 이후 원인을 탐색하는 과정에서 *좌회전과 우회전의 비율에 차이가 꽤 크다는 것을 발견*하였습니다. 그럴 수밖에 없던 것이, 자율주행 코스가 좌회전이 총 6번, 우회전이 총 2번 나왔기에 아무리 우회전 코스만을 추가로 수집해도 좌회전에 비해 턱없이 부족하던 것이었습니다.



따라서 4차 데이터 수집 과정에서는 전 코스를 3번 주행하고, 기존대로 커브가 많은 문제의 코스를 10번 정도 주행하고, 추가로 우회전 코스를 10차례 더 주행했습니다. 그 결과 시작 위치나 방향에 상관없이 잘 동작하는 결과를 얻을 수 있었습니다.

데이터는 VSCode 상에 저장되는데, 모델 학습을 위해 해당 데이터를 로컬 환경으로 가져온 후 Backend.AI로 옮기는 작업을 진행했습니다. 이후 사전에 받은 **train_ai_car.ipynb** 파일을 실행시켜 학습을 진행했습니다. 해당 코드는 변경된 점이 몇 가지 있습니다. 내용은 아래 서술하겠습니다.

우선 Jupyter Notebook의 특성상 경고나 주의 문구가 자주 노출되는데, 이 부분이 결과 확인을 방해할 수 있으므로 출력을 off로 전환하는 코드를 맨 앞에 추가했습니다.

In [ ]:
```
# status initialize
import os

os.environ['CUDA_VISIBLE_DEVICES'] = '1'
```

다음으로 필요한 모듈을 임포트하고, 데이터를 준비했습니다. 데이터를 가져와 빈 리스트에 저장하였고, 데이터 파일명에서 방향 정보를 추출하여 steering_angle 리스트에 저장합니다. 데이터를 저장할 때, 파일명을 방향키에 대응되는 값으로 매핑하여 저장하였기 때문에, 별도로 레이블링 작업을 하지 않을 수 있었습니다. 이후 데이터 로드가 잘 되었는지 출력하고 시각화합니다.

(임포트 셀 이미지 생략)

In [ ]:
```
data_dir = "crop_img"
file_list = os.listdir(data_dir)
image_paths = []
steering_angles = []
pattern ="*.png"
for filename in file_list:
    if fnmatch.fnmatch(filename, pattern):
        image_paths.append(os.path.join(data_dir, filename))
        angle = int(filename[-7:-4])
        steering_angles.append(angle)
print('image_paths', image_paths)
print('steering_angles', steering_angles)
image_index = 1
plt.imshow(Image.open(image_paths[image_index]))
print("image_path: ", image_paths[image_index])
print("steering_Angle: ", steering_angles[image_index])
df = pd.DataFrame()
df['ImagePath'] = image_paths
df['Angle'] = steering_angles
```

다음으로 직진과 좌회전, 우회전의 분포를 확인하기 위해 히스토그램을 생성하여 시각화합니다.

In [ ]:
```
num_of_bins = 10
hist, bins = np.histogram(df['Angle'], num_of_bins)
print(hist)
print(bins)
fig, axes = plt.subplots(1,1,figsize=(10,4))
axes.hist(df['Angle'], bins=num_of_bins, width=1, color='blue')
```

다음은 방향 데이터를 범주형 데이터로 변환하는 작업을 수행하는 코드입니다. 이전에 방향 정보를 저

장한 steering_angles 리스트의 값이 만약 0이면 0으로, 45면 1로, 135면 2로 매핑합니다. 하지만 이번 과제에서는 0과 45, 135로 저장되지 않고 처음부터 0과 1, 2로 저장되었으므로 해당 코드는 의미가 없습니다. 따라서 삭제해도 동작에 지장이 없습니다.

```
In [ ]:
s = np.array(steering_angles)
new_angles = np.copy(steering_angles)
new_angles[np.where(s==0)] = 0
new_angles[np.where(s==45)] = 1
new_angles[np.where(s==135)] = 2
print(new_angles)
print(steering_angles)
print(len(np.where(new_angles==0)[0]))
print(len(np.where(new_angles==1)[0]))
print(len(np.where(new_angles==2)[0]))
#print(len(new_angles))
df['Angle'] = new_angles
```

다음으로 학습셋과 테스트셋으로 분리하는 작업을 진행합니다. 이 부분은 따로 수정하지 않고, 테스트셋의 비율을 20%로 유지하였습니다. 추가로, 학습셋과 테스트셋의 방향 정보 분포를 확인하기 위해 전과 동일하게 히스토그램을 시각화합니다.

```
In [ ]:
X_train, X_valid, y_train, y_valid = train_test_split(image_paths, new_angles, test_size=0.2)
print("Training data: {}, Validation data : {}".format(len(X_train), len(X_valid)))

fig, axes = plt.subplots(1,2, figsize=(12,4))
axes[0].hist(y_train, bins=num_of_bins, width=1, color='blue')
axes[0].set_title("Training data")
axes[1].hist(y_valid, bins=num_of_bins, width=1, color='red')
axes[1].set_title("Validation data")
```

다음으로 학습셋이 잘 준비되었는지 확인하기 위해 이미지 경로와 정답 클래스 쌍을 출력합니다.

```
In [ ]:
for a, b in zip(X_train, y_train):
    print(a,b)
```

다음은 전처리 작업을 진행합니다. 우선 이미지들을 모두 불러와, 각 픽셀값이 0~1 사이의 값을 갖도록 255로 나누는 정규화 작업을 진행합니다. 이후, 원본 이미지와 전처리한 이미지를 비교할 수 있도록 시각화합니다. 정규화 작업은 모든 입력값의 Scale을 유사하게 조정하여 학습을 안정화하기 위해 진행합니다.

```
In [ ]:
def my_imread(image_path):
    image = cv2.imread(image_path)
    return image
def img_preprocess(image):
    image = np.float32(image) / 255.
    return image
fig, axes = plt.subplots(1,2,figsize=(15,10))
image_ori = my_imread(image_paths[image_index])
image_processed = img_preprocess(image_ori)
axes[0].imshow(image_ori)
axes[0].set_title("ori")
axes[1].imshow(image_processed)
axes[1].set_title("processed")
print(image_ori.shape)
```

다음은 본격적으로 모델 설계에 들어갑니다. 이번 과제에 사용한 모델 구조는 기존 모델 구조와 매우 다릅니다. 기존 모델 구조와 하나씩 비교하자면, 우선 Kernel의 개수를 전체적으로 늘렸습니다. 왜냐하면, 몇 차례 학습이 잘 진행되지 않아 과적합의 위험을 감수하고 모델의 복잡도를 조금 늘릴 필요성을 느꼈기 때문입니다. 다행히 좋은 결과를 얻을 수 있었고, loss 그래프도 문제없었습니다. 다음으로, 과적합을 방지하기 위한 Dropout의 값을 0.5에서 0.2로 줄였습니다. 이유는 전과 마찬가지로 모델이 데이터의 패턴을 잘 학습하지 못한다고 판단했기 때문입니다. 따라서 과적합의 위험을 조금 감수하더라도 뉴런을 조금 더 사용하기로 결정했습니다. 대신 콘볼루션 레이어를 지나 마지막 Dense 레이어에서 가중치 정규화 작업을 함께 진행하여 과적합의 위험이 상쇄되도록 했습니다. 이번 과제에서는 L2 정규화를 진행하였는데, 이유는 이미지의 연속성이 강하기 때문에, 일부 가중치를 0으로 만들어 버리는 L1 정규화보다 L2 정규화가 더 적합하지 않을까 생각해서입니다. 다음으로 사소하긴 하지만, 가중치 정규화 작업과 동시에 Dense 레이어에서 출력 뉴런의 개수를 2의 제곱수인 128과 64로 변경하였습니다. 배치 사이즈를 포함하여 다양한 파라미터를 설정할 때는 2의 제곱인 수를 사용하는 것이 컴퓨터 메모리 접근 방식에 적합하다고 다른 강의에서 배운 적이 있기 때문입니다.

In [ ]:
```
def neural_net():
    model = Sequential()
    model.add(Conv2D(32, (5,5), input_shape=(66,200,3), activation='relu'))
    model.add(MaxPool2D((2,2)))
    model.add(Dropout(0.2))
    model.add(Conv2D(64, (5,5), activation='relu'))
    model.add(MaxPool2D((2,2)))
    model.add(Dropout(0.2))
    model.add(Conv2D(64, (5,5), activation='relu'))
    model.add(MaxPool2D((2,2)))
    model.add(Dropout(0.2))
    model.add(Conv2D(128, (3,3), activation='relu'))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dropout(0.5))
    model.add(Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
    model.add(Dropout(0.5))
    model.add(Dense(64, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
    model.add(Dropout(0.5))
    model.add(Dense(3, activation='softmax'))

    opt = Adam(lr=1e-3)
    model.compile(
        loss='sparse_categorical_crossentropy',
        optimizer=opt
    )

    return model

model = neural_net()
print(model.summary())
```

다음으로 모델 학습을 위한 배치 데이터를 생성하는 함수를 정의하는 코드입니다. 배치 방법은 모든 입력을 한 번에 메모리로 로드하지 않고, 필요한 만큼 즉, 설정한 배치 사이즈만큼 로드하는 방법으로 메모리 사용을 효율적으로 한다는 장점이 있습니다. 또한, 가중치를 더 자주 업데이트하므로 학습이 더 안정적으로 진행되는 장점도 있습니다.

In [ ]:
```
def image_data_generator(image_paths, steering_angles, batch_size):
    while True:
        batch_images = []
        batch_steering_angles = []
        len_imgs = len(image_paths)
        for i in range(batch_size):
            random_index = random.randint(0, len_imgs-1)
            image_path = image_paths[random_index]
            image = my_imread(image_path)
            steering_angle = steering_angles[random_index]

            image = img_preprocess(image)
            batch_images.append(image)
            batch_steering_angles.append(steering_angle)
        yield(np.asarray(batch_images), np.asarray(batch_steering_angles))
```

다음으로는 학습셋과 테스트셋의 샘플을 시각화하는 코드입니다. 학습셋과 테스트셋이 적절히 분리가 되었는지, 레이블링에 문제가 없었는지를 확인하기 위한 목적으로 보입니다.

In [ ]:
```
ncol = 2
nrow = 2

print(len(X_train))

X_train_batch, y_train_batch = next(image_data_generator(X_train, y_train, nrow))
X_valid_batch, y_valid_batch = next(image_data_generator(X_valid, y_valid, nrow))

print(len(X_train_batch))
fig, axes = plt.subplots(nrow, ncol, figsize=(15,6))
fig.tight_layout()

for i in range(nrow):
    axes[i][0].imshow(X_train_batch[i])
    axes[i][0].set_title("training, angle={}".format(y_train_batch[i]))
    axes[i][1].imshow(X_valid_batch[i])
    axes[i][1].set_title("validation, angle={}".format(y_valid_batch[i]))
```

다음으로 학습한 모델을 저장할 디렉터리를 생성합니다. 디렉터리명은 현재 날짜와 시간을 기준으로 생성됩니다. 하지만 이는 세계 시각 기준 즉, 영국의 런던 왕립 천문대 시각 기준으로 한국에서는 9시간 전인 시각 기준으로 생성됩니다.

```
In [ ]:
save_dir = datetime.datetime.now().strftime('%Y%m%d_%H%M')
model_output_dir = os.path.join(save_dir)
os.mkdir(model_output_dir)
print('model_output_dir ', model_output_dir)
```

저장할 디렉터리를 생성했으면, 이제 본격적으로 학습을 진행합니다. 학습셋과 테스트셋의 배치 사이즈는 각각 16과 128로 설정했습니다. 다음으로 체크포인트 콜백을 설정했습니다. 이는 가장 성능이 좋았던 모델만을 저장하기 위함입니다. 이후 모델 학습을 진행합니다. 기존 epochs는 100이었으나 150으로 늘렸다는 점 외에는 다른 점은 없습니다. 이렇게 학습이 진행되고 나면, 지금까지 학습한 히스토리도 함께 pickle 파일로 저장합니다. 히스토리는 epochs 별 손실 값이나 정확도 등 학습 결과를 분석하기 위해 사용됩니다. 원래 여기서 체크포인트 콜백 외에 Early-Stopping 콜백도 추가해봤으나, 결과적으로 자율주행이 잘 진행되지 않아 조금 오래 걸릴지라도 Early-Stopping 콜백은 제거하는 것으로 수정했습니다.

```
In [ ]:
train_bSize = 16
valid_bSize = 128

# 체크포인트 콜백 설정
checkpoint = 'lane_navigation_'+ save_dir + '.h5'
checkpoint_callback = ModelCheckpoint(
    filepath=os.path.join(model_output_dir, checkpoint),
    verbose=1,
    save_best_only=True   # 가장 좋은 모델만 저장
)

# 학습 시작
history = model.fit_generator(
    image_data_generator(X_train, y_train, batch_size=train_bSize),
    steps_per_epoch=len(X_train) // train_bSize,
    epochs=150,
    validation_data=image_data_generator(X_valid, y_valid, batch_size=valid_bSize),
    validation_steps=len(X_valid) // valid_bSize,
    verbose=1,
    shuffle=True,
    callbacks=[checkpoint_callback]  # early_stopping 제거
)

history_path = os.path.join(model_output_dir, 'history.pickle')
with open(history_path, 'wb') as f:
    pickle.dump(history.history, f, pickle.HIGHEST_PROTOCOL)
```

다음은 테스트셋의 첫 번째 이미지에 대해 예측을 수행하고 결과를 시각화하는 코드입니다.

```
In [ ]:
img_path = X_valid[0]
image = my_imread(img_path)
steering_angle = steering_angles[0]
image = img_preprocess(image)
print(image.shape)
print(model.predict(np.expand_dims(image, 0))[0])
print(np.argmax(model.predict(np.expand_dims(image, 0))[0]))
plt.imshow(image)
```

마지막으로 히스토리를 통해 학습한 결과를 시각화하는 코드입니다. 저장된 히스토리 파일을 불러와, 읽기 모드로 열어 epochs 별 학습 손실과 테스트 손실을 그래프로 표시합니다.

```
In [ ]:
print(model_output_dir)

history_path = os.path.join(model_output_dir, 'history.pickle')
with open(history_path, 'rb') as f:
    history = pickle.load(f)

plt.plot(history['loss'], color='blue')
plt.plot(history['val_loss'], color='red')
plt.legend(['training loss', 'validation loss'])
```

이번 과제에서 생각 이상으로 다양한 문제와 이를 해결하는 과정을 겪었습니다. 이를 통해 그만큼 많은 것을 배울 수 있었는데, 우선 데이터셋을 만 장 이상을 수집할 정도로 초기 단계부터 꽤 오랜 시간을 투자했지만, 데이터셋의 양과 결과가 항상 비례하지만은 않는다는 것을 배웠습니다. 또한, VSCode와 Backend.AI에서 대용량 파일을 처리하지 못한다는 사실과 이 데이터를 Command 명령어로 처리하는 방법을 배울 수 있었습니다. 그리고 같은 h5 확장자라도 tensorflow 버전이 다르면 저장 구조가 완전히 달라 실행되지 않는다는 점도 배웠습니다. 게다가, Backend.AI가 대용량 파일을 처리하게 되면 Kernel이 자주 죽는 현상으로 인해 tensorflow 1.15 버전을 다른 방법으로 사용하고자 가상 환경을 설치하는 과정도 겪었는

데, 이를 통해 가상 환경 세팅하는 방법도 처음 경험할 수 있었습니다. 그 외에도 모델의 구조를 변경하거나 코드를 수정하는 과정에서도 배움이 있었을 만큼, 이번 과제는 꽤 큰 의미가 있었습니다. 아직 스스로 이 코드를 작성할 수는 없겠지만, 작게나마 한층 성장했다고 생각합니다.