CH. 13. 자율주행을 위한 딥러닝 모델

Section 01 자율주행 학습 모델

- 자율주행 자동차가 스스로 주행하려면 인지·판단·제어 등 세 가지 기능이 반드시 필요하다.
- 인지 기능은 카메라·레이더·라이다(LiDAR) 등 차체 내 센서 정보를 처리해 주변 환경 정보를 알아차리는 것, 판단 기능은 인지된 정보를 이용해 향후 벌어질 일을 예측한 후 가장 안전하고 빠른 차량 궤적을 생성하는 것, 제어 기능은 최종적으로 생성된 차량 궤적을 부드럽고 정확하게 따라갈 수 있도록 운전대·액셀러레이터·브레이크를 조작하는 것이다.
- 실제로 여러 곳에서 이 세 기능에 딥러닝 기술을 적용하는 연구가 활발하게 진행되고 있다.
- 지금껏 나온 여러 연구 결과만 봐도 딥러닝이 각각의 성능을 비약적으로 향상시킬 수 있단 사실을 알 수 있다.

Section 01 자율주행 학습 모델

- 특히 카메라를 이용한 주행 환경 인지 분야에서 딥러닝은 가장 중요한 기술로 자리 잡았다.
- 카메라를 통해 입력된 이미지에 딥러닝을 적용하면 자율주행 시스템에 필요한 정적(靜的) 환경 정보(차선·운전가능도로·교통표지판·교통신호 등)와 동적(動的) 환경 요소(차량·보행자·이륜차 등)를 전부 검출, 분류할 수 있다.
- 그 성능도 기존 머신러닝 알고리즘보다 월등히 뛰어나다.
- 또한 기존엔 인식이 불가능하다고 여겨졌던 영역의 성능까지 점차 개선되고 있다.
- 앞으로 카메라의 이미지 정보뿐만 아니라 레이더와 라이다(LiDAR)의 센서 정보, 그리고 차량 간 통신에서 오는 정보를 모두 딥러닝에 적용할 경우 인간의 인지 능력을 뛰어 넘는 자율주행 자동차의 인지 시스템을 구현할 수 있을 전망이다.

Section 01 자율주행 학습 모델

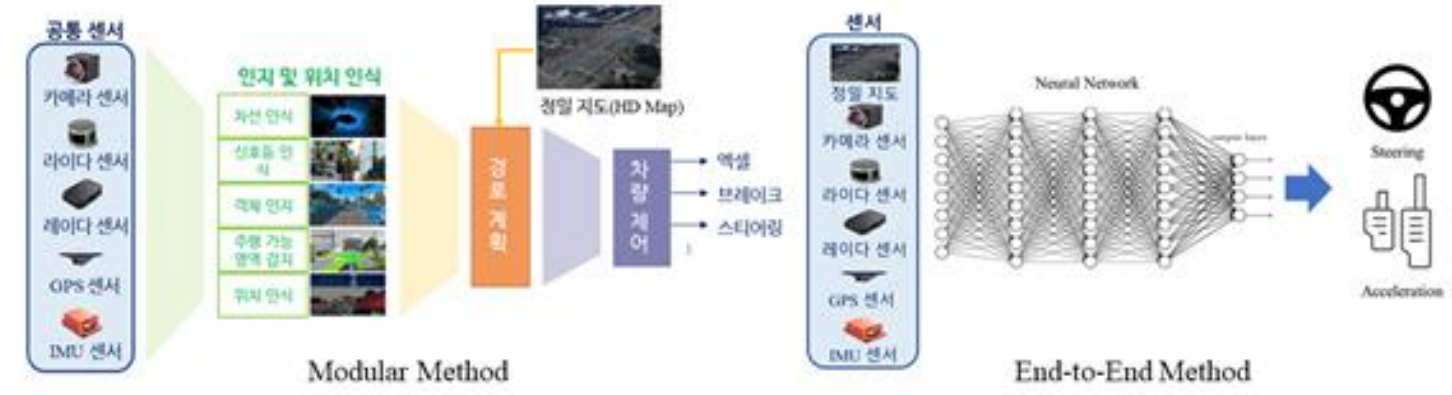
- 딥러닝은 차량의 궤적을 생성하는 판단 기능에도 사용될 수 있다.
- 미래의 움직임을 판단하고 결정하려면 다른 운전자의 움직임을 예측해야 한다.
- 하지만 다양한 운전 방식을 파악하고 이를 (정답이 딱 떨어지는) 수학 모델로 정의하긴 쉽지 않다.
- 다른 차량 운전자가 어떻게 운전할지 논리적으로 예측하는 건 불가능하기 때문이다.
- 이처럼 행동 방식을 논리적으로 파악하기 어려울 때 딥러닝은 매력적 솔루션이 된다.
- 다양한 운전 방식과 관련 센서 정보를 데이터로 입력한 후 이를 딥러닝 알고리즘으로 학습한다면 정확한 수학 모델 없이 데이터만으로도 다른 운전자의 운행을 예측할 수 있는 인공지능 구현이 가능하다.

Section 01 자율주행 학습 모델

- 차량 제어 분야에서도 딥러닝을 응용할 수 있다.
- 고도의 안정성이 요구되는 차량 제어 기능을 구현하려면 검증된 기존 기술을 사용하는 게 일반적이다.
- 하지만 탑승자의 승차감을 튜닝(tuning)하기 위한 제어 기능엔 딥러닝을 적용할 여지가 있다.
- 승차감은 개개인이 느끼는 감성적 요소인 만큼 '공학적 기준'을 만들어 적용하기엔 한계가 있다.
- 따라서 이 경우, 딥러닝 기술을 활용해 개개인의 운전 방식을 데이터화하면 인간 감성을 고려한 차량 제어가 가능해진다.
- 다시 말해 단순 자율주행 기능뿐 아니라 탑승자의 안색·음성·상태 등을 인식, 개별 탑승자에게 '맞춤형 편의 기술'을 제공하는 서비스에도 딥러닝이 적용될 수 있다.

Section 01 자율주행 학습 모델

- 자율주행 자동차의 주행 방식에는 모듈러(Modular) 방식과 종단간(End-to-End) 주행 방식이 있으며, 현재 대부분의 연구 개발은 모듈러 방식으로 진행하고 있다.
- 모듈러 방식은 사람이 직접 모든 교통 상황을 정의하여 모듈을 구성하므로, 예측 가능한 범위 내에서만 작동하고 사고 발생시 사고 경위를 쉽게 판단할 수 있다는 장점이 있다.
- 하지만, 알고리즘 설계의 복잡도가 높아서 모든 교통 상황에 대해서 대응을 할 수 없기 때문에 Level5 자율주행 자동차 개발은 불가하고, 현재는 Level 2~3단계 수준으로 상용화가 진행되고 있다.



Section 01 자율주행 학습 모델

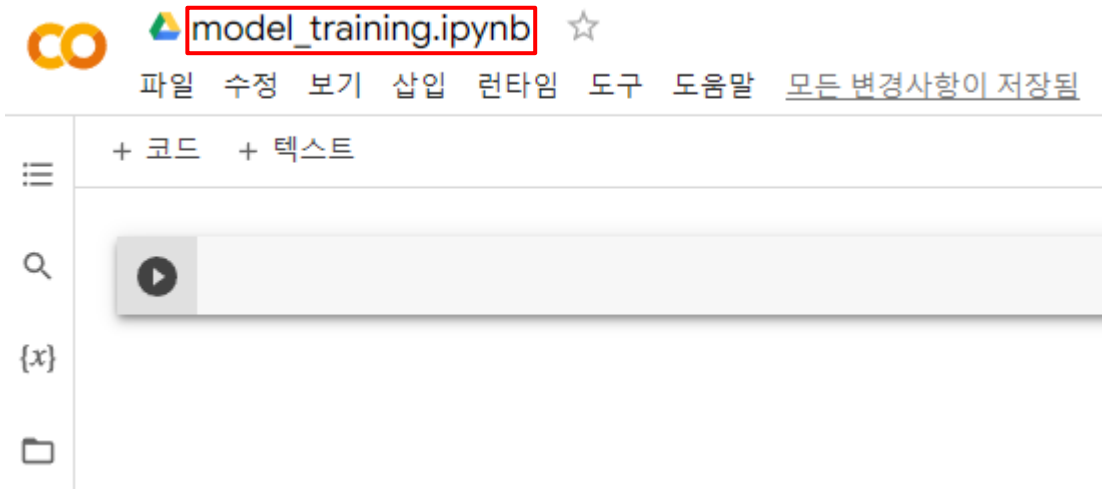
- 자율주행에 필요한 중간 기능 분류 없이 운전 과정 전체를 학습하는 방법을 종단간 (end-to-end) 자율주행이라고 한다.
- 엔드투엔드 자율주행 자동차는 운전에 필요한 센서 데이터를 직접 입력 받고 다양한 운행 상황에 대해 학습한 후 스티어링(steering)과 액셀러레이터, 브레이크 값을 직접 출력해낸다.
- 즉 딥러닝 알고리즘 안에서 인지·판단·제어 기능이 처음부터 끝까지 네트워크로 구현되는 것이다.
- 종단간 학습 모델은 다양한 도로 환경에 대한 충분한 데이터가 있다는 가정하에 입력값 간의 최적화된 조합을 찾아 다양한 교통상황에 대해서 모듈러 방식을 뛰어넘는 성능을 가질 수 있다.

Section 01 자율주행 학습 모델

- 종단간 학습 방식엔 운전자(사람)의 운행 방식을 데이터로 수집, 이를 모방하는 학습 방식과 시뮬레이터를 이용해 가장 최적화된 운전 방식을 스스로 학습하는 강화학습 기반 방식이 있다.
- 종단간 방식을 자율주행 자동차에 적용하면 새로운 운행 환경과 관련해 추가 기능이 필요할 때 인지·판단·제어 알고리즘을 매번 재설계하거나 변경하지 않고, 새로운 상황에 대한 데이터를 추가해 학습함으로써 자율주행 시스템을 구현할 수 있다.

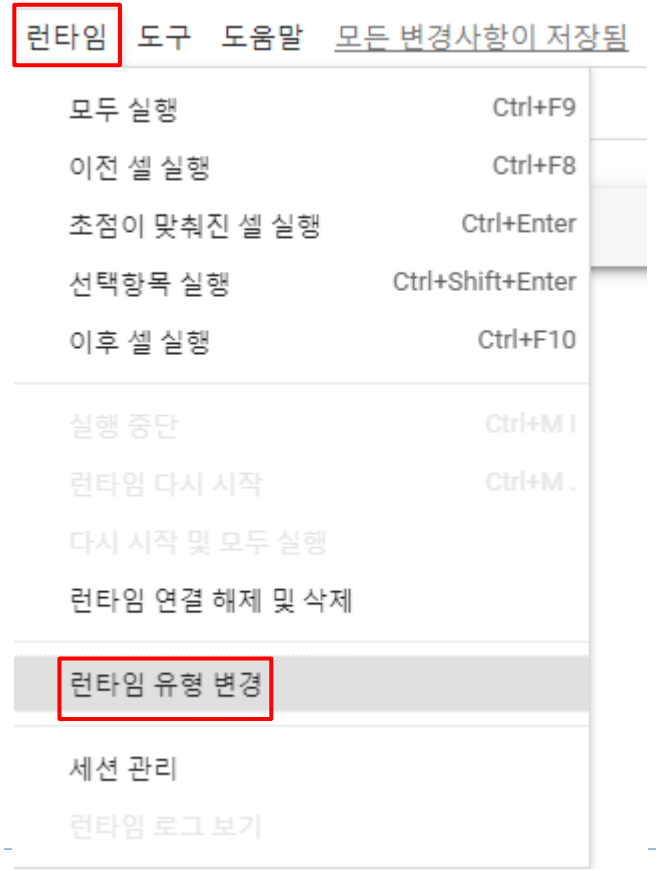
Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ [파일] -> [새 노트]를 생성하여 [model_training]으로 이름을 변경한다.



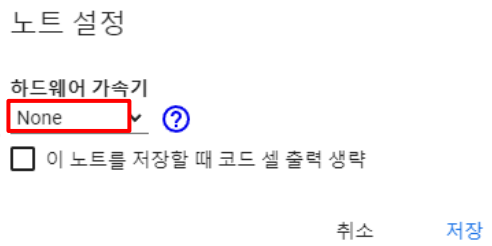
Section 02 자율주행을 위한 end-to-end learning 모델 학습

- 코랩은 딥러닝을 위한 하드웨어 가속기가 있어 빠르게 연산이 가능하다.
- [런타임] -> [런타임 유형 변경]을 클릭한다.

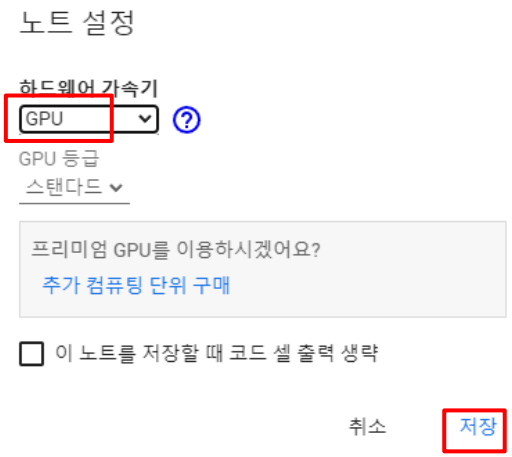


Section 02 자율주행을 위한 end-to-end learning 모델 학습

- ❑ 기본값인 [None] 으로 되어 있다.
- ❑ [None]부분을 클릭한다.



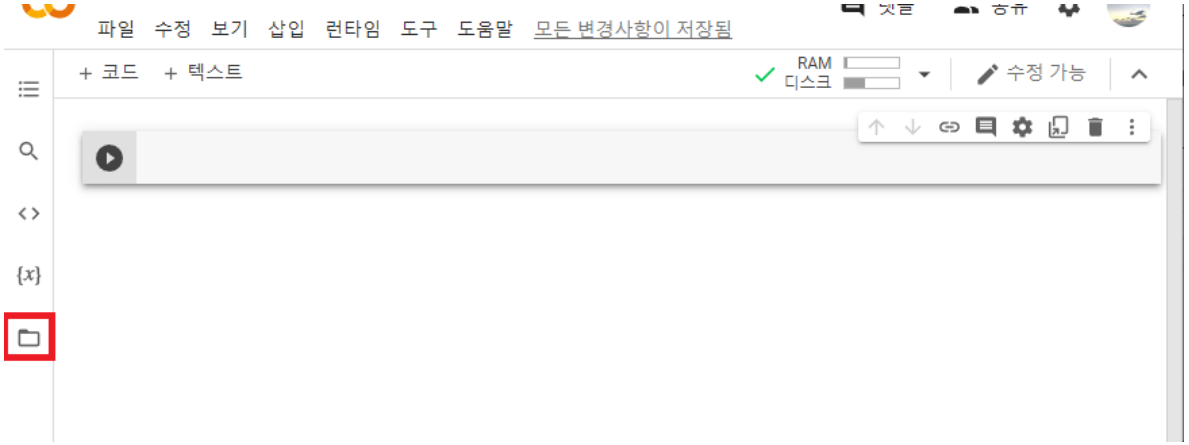
- ❑ [GPU]를 선택 후 [저장]을 눌러 빠져나온다.
- ❑ [TPU]는 일반적인 상황에서 [GPU]와 별 차이가 없고 코드도 변경해야 할 부분이 있기 때문에 [GPU]를 선택한다.



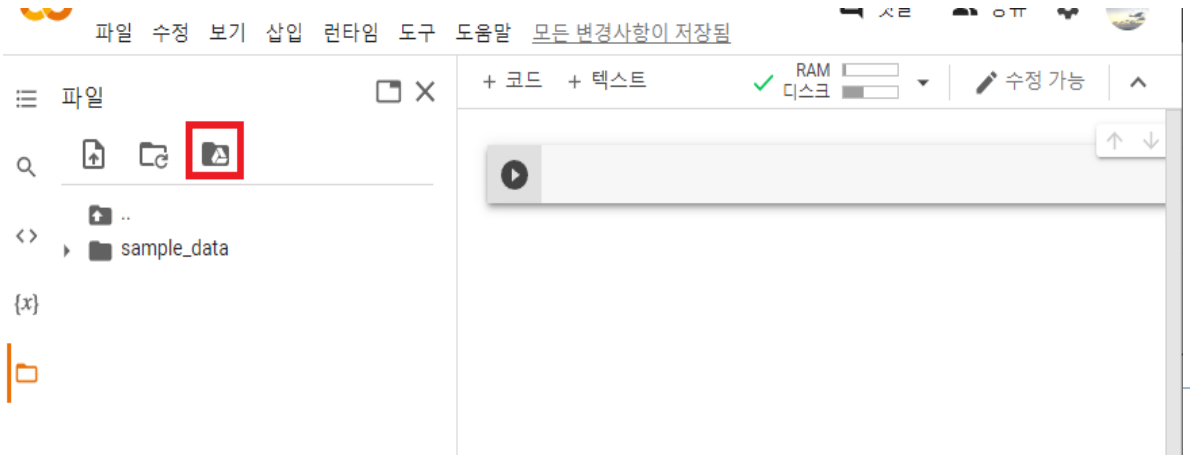
Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ 구글 드라이브 마운트

- 구글 코랩 노트에 왼쪽 중간의 '파일'버튼을 클릭한다.



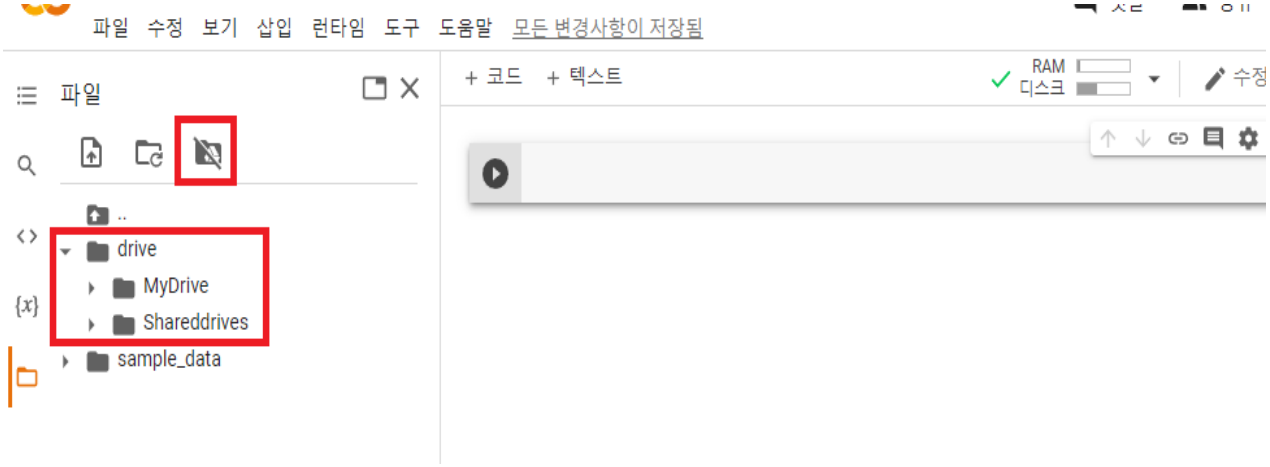
- 아래와 같은 화면이 나타나면 표시의 '드라이브 마운트' 아이콘을 클릭한다.



Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ 구글 드라이브 마운트

- '드라이브 마운트' 아이콘에 대각선이 생기며 구글 드라이브 연동이 완료된다.



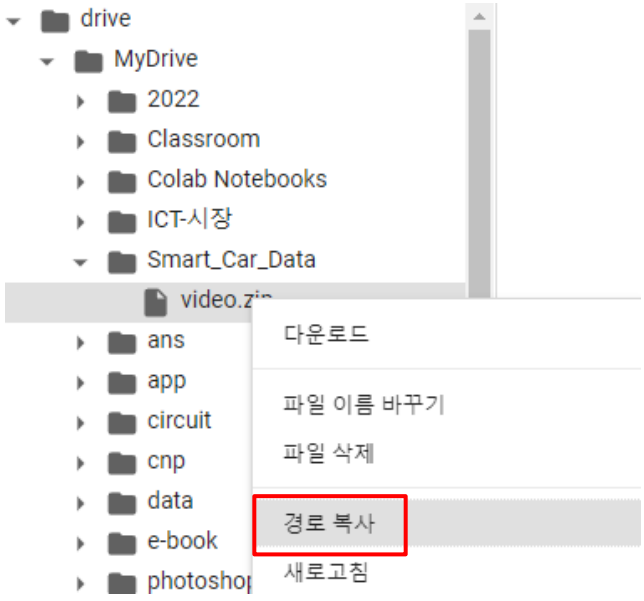
Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ 압축 풀기

- **구글 코랩 셀에 다음과 같이 입력하여 경로에 있는 파일의 압축을 풀어준다.**

```
!unzip -qq "/content/drive/MyDrive/Smart_Car_Data/video.zip"
```

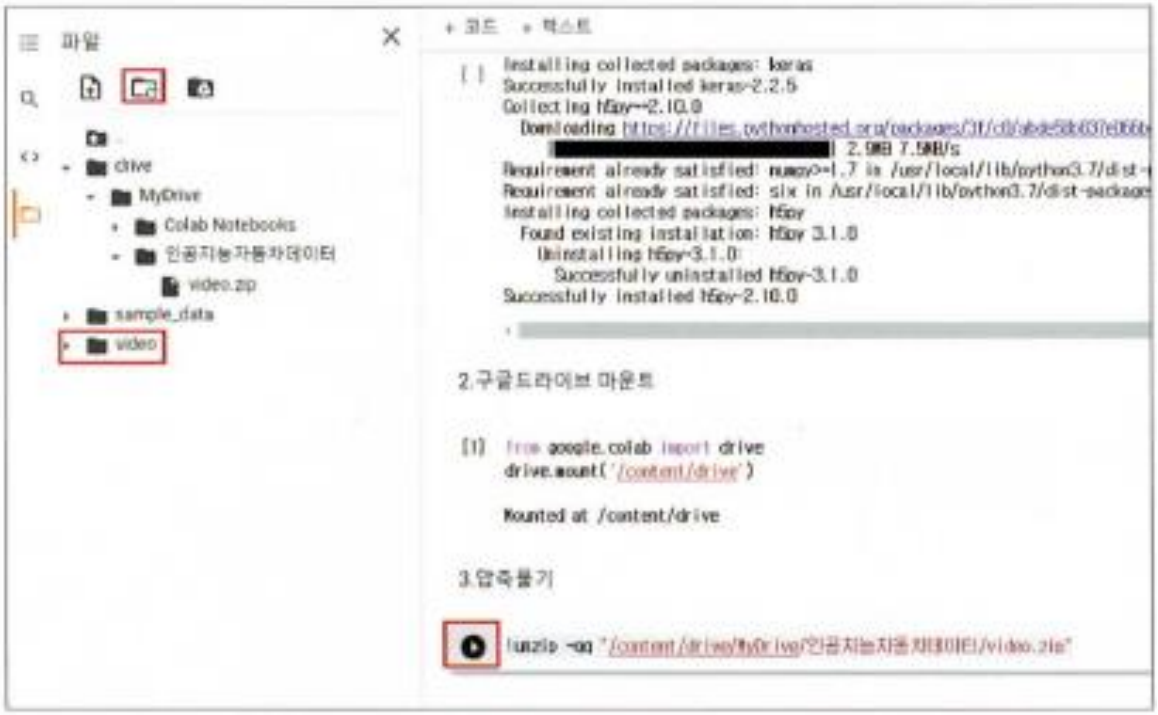
- video.zip **파일의 경로를 복사하는 방법은 코랩에서** video.zip **파일에 마우스 오른쪽 버튼을 클릭 후 [경로 복사] 버튼을 클릭하면 된다.**



Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ 압축 풀기

- [shift + enter]**키를 입력하여 구글 코랩 셀을 실행한다**.
- [**폴더 새로고침**] **아이콘을 클릭하면 압축이 풀린** [video] **폴더가 보여진다**.
- [video] **폴더 안에는 매우 많은 사진 데이터가 있다**.



Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ 라이브러리 불러오기

- 다음과 같은 코드를 추가하여 라이브러리를 불러온다.

```
1   import os
2   import random
3   import fnmatch
4   import datetime
5   import pickle
6
7   # data processing
8   import numpy as np
9   np.set_printoptions(formatter={'float_kind':lambda x: "%.4f" % x})
10
11  import pandas as pd
12  pd.set_option('display.width', 300)
13  pd.set_option('display.float_format', '{:,.4f}'.format)
14  pd.set_option('display.max_colwidth', 200)
15
```

Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ 라이브러리 불러오기

- 다음과 같은 코드를 추가하여 라이브러리를 불러온다.

```
16  # tensorflow
17  import tensorflow as tf
18  import tensorflow.keras
19  from tensorflow.keras.models import Sequential  # V2 is tensorflow.keras.xxxx, V1 is keras.xxx
20  from tensorflow.keras.layers import Conv2D, MaxPool2D, Dropout, Flatten, Dense
21  from tensorflow.keras.optimizers import Adam
22  from tensorflow.keras.models import load_model
23
24  print( f'tf.__version__: {tf.__version__}' )
25  print( f'.keras.__version__: {tensorflow.keras.__version__}' )
26
27  # sklearn
28  from sklearn.utils import shuffle
29  from sklearn.model_selection import train_test_split
30
```

Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ 라이브러리 불러오기

- 다음과 같은 코드를 추가하여 라이브러리를 불러온다.

```
31  # imaging
32  import cv2
33  from imgaug import augmenters as img_aug
34  import matplotlib.pyplot as plt
35  import matplotlib.image as mpimg
36  %matplotlib inline
37  from PIL import Image
```

Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ 데이터 불러오기

- **코랩 셀에 다음과 같은 코드를 추가하여 데이터를 불러온다.**

```
1   data_dir = '/content/video'
2   file_list = os.listdir(data_dir)
3   image_paths = []
4   steering_angles = []
5   pattern = "*.png"
6   for filename in file_list:
7       if fnmatch.fnmatch(filename, pattern):
8           image_paths.append(os.path.join(data_dir,filename))
9           angle = int(filename[-7:-4])
10          steering_angles.append(angle)
11
```

Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ 데이터 불러오기

- **코랩 셀에 다음과 같은 코드를 추가하여 데이터를 불러온다.**

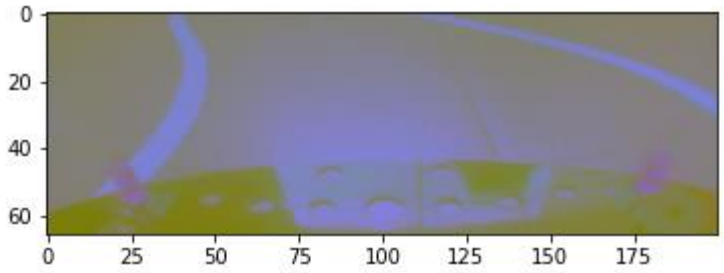
```
12  image_index = 20
13  plt.imshow(Image.open(image_paths[image_index]))
14  print("image_path: %s" % image_paths[image_index] )
15  print("steering_Angle: %d" % steering_angles[image_index] )
16  df = pd.DataFrame()
17  df['ImagePath'] = image_paths
18  df['Angle'] = steering_angles
```

Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ 데이터 불러오기

- 코랩 셀을 실행해서 데이터가 잘 불려왔는지 확인한다.

```
image_path: /content/video/train_01018_090.png
steering_Angle: 90
```



Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ 조향각 분포 확인하기

- **코랩 셀에 다음과 같은 코드를 추가한다.**

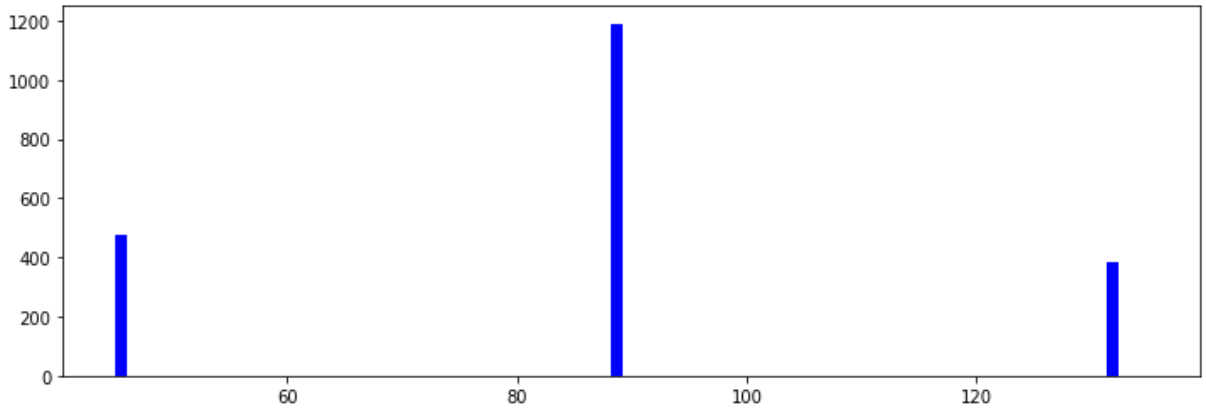
```
1  num_of_bins = 25
2  hist, bins = np.histogram(df['Angle'], num_of_bins)
3
4  fig, axes = plt.subplots(1,1, figsize=(12,4))
5  axes.hist(df['Angle'], bins=num_of_bins, width=1, color='blue')
```

Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ 조향각 분포 확인하기

- 코랩 셀을 실행하여 데이터의 빈도수를 확인한다.
- 너무 한쪽으로 몰려있다면 데이터가 편향적으로 학습될 것이다.
- 이 그래프를 참조하여 너무 한쪽으로 데이티가 몰리지 않도록 학습한다.
- 너무 한쪽으로 몰려있다면 데이터를 다시 모아야 한다.

```
(array([475.0000,   0.0000,   0.0000,   0.0000,   0.0000,   0.0000,   0.0000,   0.0000,
          0.0000,   0.0000,   0.0000,   0.0000, 1190.0000,   0.0000,   0.0000,   0.0000,
          0.0000,   0.0000,   0.0000,   0.0000,   0.0000,   0.0000,   0.0000,   0.0000,
        385.0000]),
 array([45.0000, 48.6000, 52.2000, 55.8000, 59.4000, 63.0000, 66.6000,
        70.2000, 73.8000, 77.4000, 81.0000, 84.6000, 88.2000, 91.8000,
        95.4000, 99.0000, 102.6000, 106.2000, 109.8000, 113.4000, 117.0000,
        120.6000, 124.2000, 127.8000, 131.4000, 135.0000]),
 <a list of 25 Patch objects>)
```



Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ 학습 데이터와 검증 데이터 분리하기

- **코랩 셀에 다음과 같은 코드를 추가한다.**

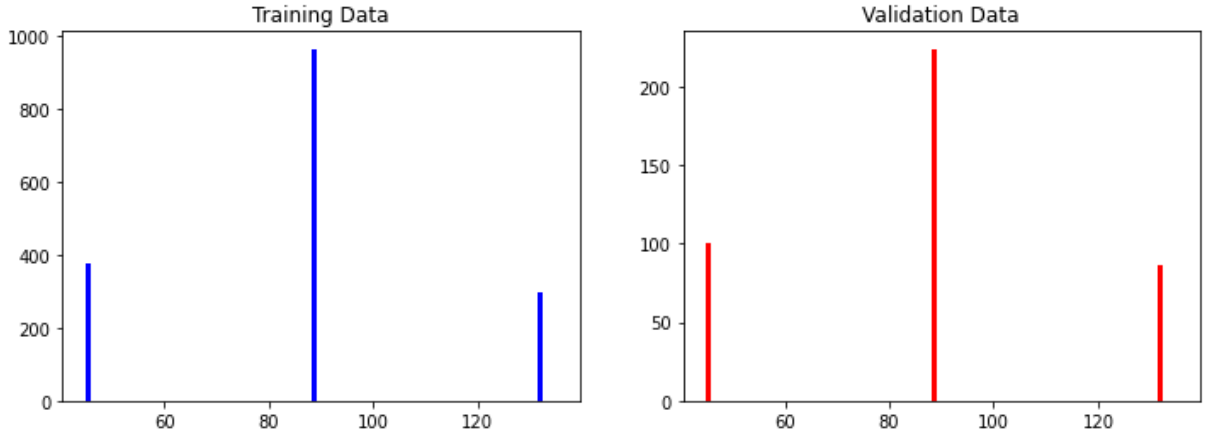
```
1  X_train, X_valid, y_train, y_valid = train_test_split( image_paths, steering_angles, test_size=0.2)
2  print("Training data: %d\nValidation data: %d" % (len(X_train), len(X_valid)))
3
4  fig, axes = plt.subplots(1,2, figsize=(12,4))
5  axes[0].hist(y_train, bins=num_of_bins, width=1, color='blue')
6  axes[0].set_title('Training Data')
7  axes[1].hist(y_valid, bins=num_of_bins, width=1, color='red')
8  axes[1].set_title('Validation Data')
```

Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ 학습 데이터와 검증 데이터 분리하기

- **코랩 셀을 실행하여 결과를 확인하면** 1,640**개의 학습 데이터와** 410**개의 검증 데이터로 분리되었다.**
- **학습 데이터는 파란색, 검증 데이터는 빨간색으로 비슷한 분포를 띄고 있다.**

```
Training data: 1640
Validation data: 410
Text(0.5, 1.0, 'Validation Data')
```



Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ 이미지 읽어오기 및 정규화하기

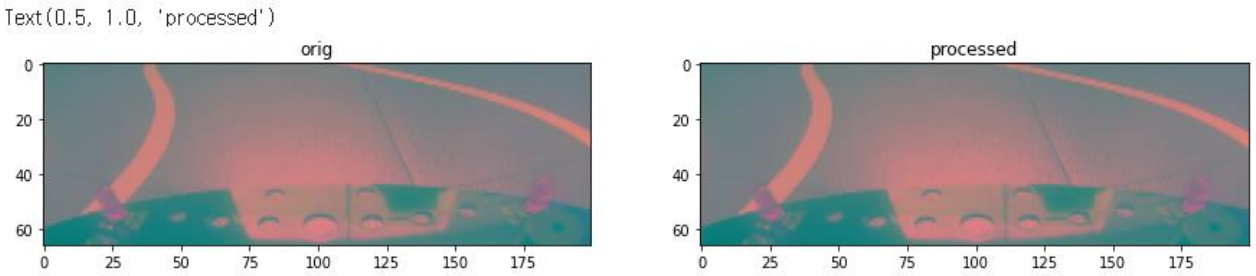
- **코랩 셀에 다음과 같은 코드를 추가한다.**

```
1   def my_imread(image_path):
2       image = cv2.imread(image_path)
3       return image
4
5   def img_preprocess(image):
6      image = image / 255
7      return image
8
9   fig, axes = plt.subplots(1, 2, figsize=(15, 10))
10  image_orig = my_imread(image_paths[image_index])
11  image_processed = img_preprocess(image_orig)
12  axes[0].imshow(image_orig)
13  axes[0].set_title("orig")
14  axes[1].imshow(image_processed)
15  axes[1].set_title("processed")
```

Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ 이미지 읽어오기 및 정규화하기

- **코랩 셀을 실행하여** orig **원본 이미지와** processed **정규화된 이미지이다. 정규화하였지만 눈으로 보여지는 차이는 없다.**
- **하지만 데이터를 학습할 때는 정규화하여 입력하면 학습 효과가 좋다.**



Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ end-to-end learning 모델 구성

- **코랩 셀에 다음과 같은 코드를 추가한다.**

```
1  def nvidia_model():
2      model = Sequential(name='Nvidia_Model')
3      model.add(Conv2D(24, (5, 5), strides=(2, 2), input_shape=(66, 200, 3), activation='elu'))
4      model.add(Conv2D(36, (5, 5), strides=(2, 2), activation='elu'))
5      model.add(Conv2D(48, (5, 5), strides=(2, 2), activation='elu'))
6      model.add(Conv2D(64, (3, 3), activation='elu'))
7      model.add(Dropout(0.2))
8      model.add(Conv2D(64, (3, 3), activation='elu'))
9
10     model.add(Flatten())
11     model.add(Dropout(0.2))
12     model.add(Dense(100, activation='elu'))
13     model.add(Dense(50, activation='elu'))
14     model.add(Dense(10, activation='elu'))
15
```

Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ end-to-end learning 모델 구성

- **코랩 셀에 다음과 같은 코드를 추가한다.**

```
16      model.add(Dense(1))
17      optimizer = Adam(lr=1e-3)
18      model.compile(loss='mse', optimizer=optimizer)
19
20      return model
21
22  model = nvidia_model()
23  print(model.summary())
```

Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ end-to-end learning 모델 구성

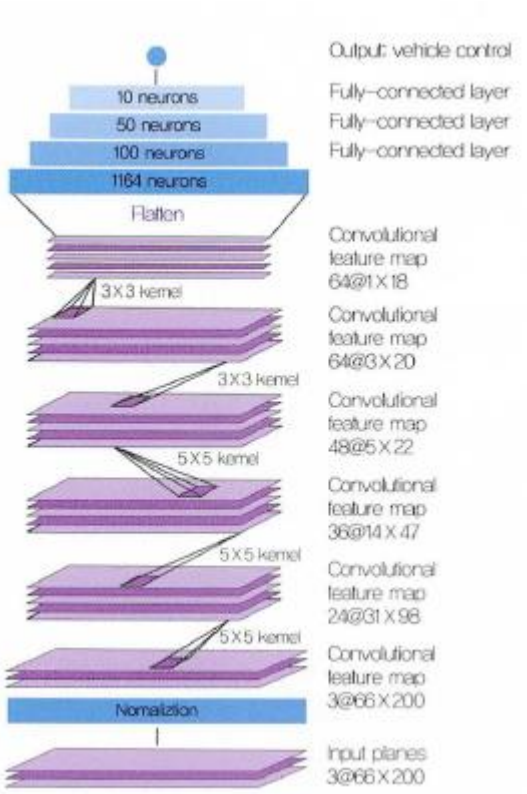
- **코랩 셀을 실행하여 결과를 확인하면, 총 252,219 파라미터로 모델을 생성하였음을 확인할 수 있다.**

```
WARNING:tensorflow:From /usr/local/lib/python3.7/dist-packages/tensorflow_core/python/ops/resource_variable_ops.py:1630: calling BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops) with constraint is deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
Model: "Nvidia_Model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 31, 98, 24)        1824
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 47, 36)        21636
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 5, 22, 48)         43248
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 20, 64)         27712
_________________________________________________________________
dropout (Dropout)            (None, 3, 20, 64)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 1, 18, 64)         36928
_________________________________________________________________
flatten (Flatten)            (None, 1152)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1152)              0
_________________________________________________________________
dense (Dense)                (None, 100)               115300
_________________________________________________________________
dense_1 (Dense)              (None, 50)                5050
_________________________________________________________________
dense_2 (Dense)              (None, 10)                510
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 11
=================================================================
Total params: 252,219
Trainable params: 252,219
Non-trainable params: 0
_________________________________________________________________
None
```

Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ end-to-end learning 모델 구성

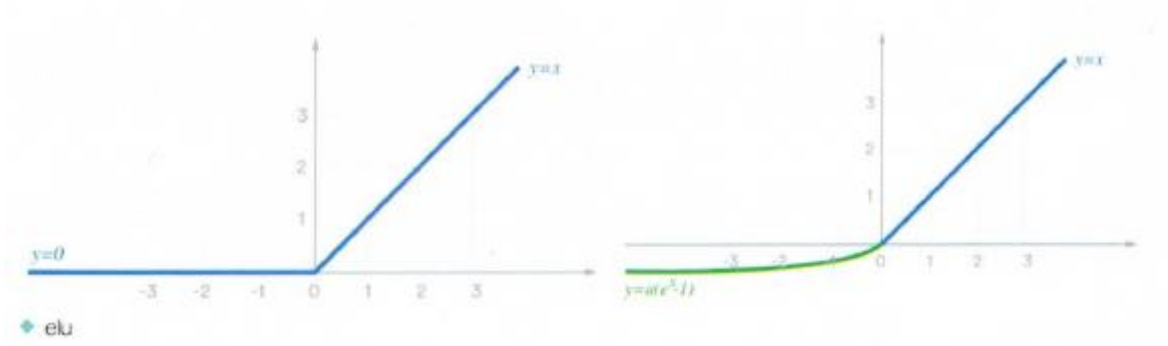
- end-to-end learning **모델은** Nvidia**에서 제공한 모델로 다음과 같이 모델이 구성된다**.



Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ end-to-end learning 모델 구성

- Nvidia**에서 적용한 모델을 충실하게 구성하였다**.
- **단 모델을 더욱더 견고하게 만들기 위해 드롭아웃 레이어를 추가하였다**.
- **우리는 익숙한** ReLU (Rectified Linear Unit) **활성화 함수 대신** ELU(Exponential Linear Unit) **활성화 함수를 사용했다**.
- ELU**는** x**가 음수일 때도 약간의 값을 다음 상태로 넘겨준다**.



Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ 학습 데이터 생성

- **코랩 셀에 다음과 같은 코드를 추가한다.**

```
1   def image_data_generator(image_paths, steering_angles, batch_size):
2     while True:
3       batch_images = []
4       batch_steering_angles = []
5
6       for i in range(batch_size):
7         random_index = random.randint(0, len(image_paths) - 1)
8         image_path = image_paths[random_index]
9         image = my_imread(image_paths[random_index])
10        steering_angle = steering_angles[random_index]
11
12        image = img_preprocess(image)
13        batch_images.append(image)
14        batch_steering_angles.append(steering_angle)
15
```

Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ 학습 데이터 생성

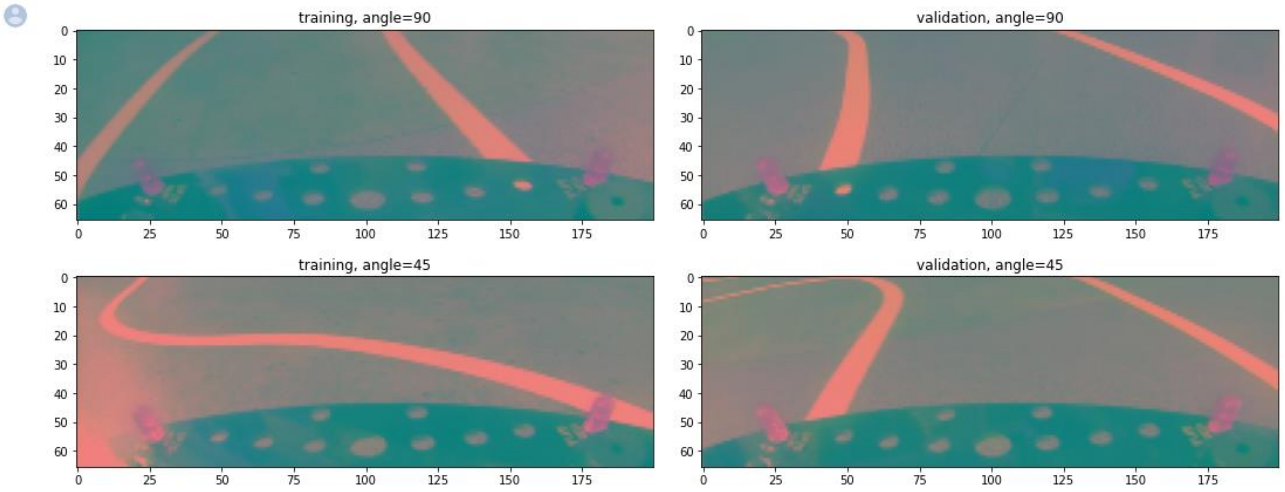
- **코랩 셀에 다음과 같은 코드를 추가한다.**

```
16        yield( np.asarray(batch_images), np.asarray(batch_steering_angles))
17
18    ncol = 2
19    nrow = 2
20
21    X_train_batch, y_train_batch = next(image_data_generator(X_train, y_train, nrow))
22    X_valid_batch, y_valid_batch = next(image_data_generator(X_valid, y_valid, nrow))
23
24    fig, axes = plt.subplots(nrow, ncol, figsize=(15, 6))
25    fig.tight_layout()
26
27    for i in range(nrow):
28      axes[i][0].imshow(X_train_batch[i])
29      axes[i][0].set_title("training, angle=%s" % y_train_batch[i])
30      axes[i][1].imshow(X_valid_batch[i])
31      axes[i][1].set_title("validation, angle=%s" % y_valid_batch[i])
```

Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ 학습 데이터 생성

- 코랩 셀을 실행하여 결과를 확인하면, 왼쪽의 학습 데이터와, 오른쪽의 검증 데이터가 잘 불러왔음을 확인할 수 있다.



Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ 모델 학습

- **코랩 셀에 다음과 같은 코드를 추가한다.**
- **모델 학습에는 대략 30분~1시간 가량 소요된다.**

```
1  model_output_dir = "/content/drive/MyDrive/Smart_Car_Data"
2
3  checkpoint_callback = tensorflow.keras.callbacks.ModelCheckpoint(filepath=os.path.join(model_output_
   dir,'lane_navigation_check.h5'), verbose=1, save_best_only=True)
4
5  history = model.fit_generator(image_data_generator( X_train, y_train, batch_size=100),
                                 steps_per_epoch=300,
                                 epochs=10,
                                 validation_data = image_data_generator( X_valid, y_valid, batch_size=100),
                                 validation_steps=200,
                                 verbose=1,
                                 shuffle=1,
                                 callbacks=[checkpoint_callback])
6
```

Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ 모델 학습

- 코랩 셀에 다음과 같은 코드를 추가한다.
- 모델 학습에는 대략 30분~1시간 가량 소요된다.

```
7   model.save(os.path.join(model_output_dir,'lane_navigation_final.h5'))
8   history_path = os.path.join(model_output_dir,'history.pickle')
9   with open(history_path, 'wb') as f:
10      pickle.dump(history.history, f, pickle.HIGHEST_PROTOCOL)
```

- 코랩을 실행하여 결과를 확인하면, Epoch 1/10 은 총 10개의 epoch 중 1단계를 실행하는 것을 의미한다.
- loss는 손실율로 학습이 진행되가면서 작아져야 한다.
- 코랩은 90분동안 아무런 입력이 없을 때 런타임이 종료된다.
- 학습이 완료 후 자동으로 구글 드라이브에 저장되므로 안심해도 된다.

Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ 모델 학습

- **최종결과 loss를 보면 학습이 진행될수록 loss가 줄어 들었음을 확인할 수 있다.**

```
Epoch 1/10
299/300 [============================>.] - ETA: 0s - loss: 398.0319
Epoch 00001: val_loss improved from inf to 274.91609, saving model to /content/drive/MyDrive/인공지능자동차데이터/lane_navigation_check.h5
300/300 [==============================] - 229s 764ms/step - loss: 397.6500 - val_loss: 274.9161
Epoch 2/10
299/300 [============================>.] - ETA: 0s - loss: 214.9280
Epoch 00002: val_loss improved from 274.91609 to 126.35032, saving model to /content/drive/MyDrive/인공지능자동차데이터/lane_navigation_check.h5
300/300 [==============================] - 233s 777ms/step - loss: 214.8591 - val_loss: 126.3503
Epoch 3/10
299/300 [============================>.] - ETA: 0s - loss: 116.4356
Epoch 00003: val_loss improved from 126.35032 to 120.28237, saving model to /content/drive/MyDrive/인공지능자동차데이터/lane_navigation_check.h5
300/300 [==============================] - 229s 763ms/step - loss: 116.3935 - val_loss: 120.2824
Epoch 4/10
299/300 [============================>.] - ETA: 0s - loss: 76.8087
Epoch 00004: val_loss improved from 120.28237 to 66.06465, saving model to /content/drive/MyDrive/인공지능자동차데이터/lane_navigation_check.h5
300/300 [==============================] - 229s 762ms/step - loss: 76.7342 - val_loss: 66.0646
Epoch 5/10
299/300 [============================>.] - ETA: 0s - loss: 60.9310
Epoch 00005: val_loss did not improve from 66.06465
300/300 [==============================] - 227s 758ms/step - loss: 60.9312 - val_loss: 73.4972
Epoch 6/10
299/300 [============================>.] - ETA: 0s - loss: 49.7026
Epoch 00006: val_loss improved from 66.06465 to 59.96464, saving model to /content/drive/MyDrive/인공지능자동차데이터/lane_navigation_check.h5
300/300 [==============================] - 223s 743ms/step - loss: 49.7647 - val_loss: 59.9646
Epoch 7/10
299/300 [============================>.] - ETA: 0s - loss: 43.5022
Epoch 00007: val_loss improved from 59.96464 to 46.77143, saving model to /content/drive/MyDrive/인공지능자동차데이터/lane_navigation_check.h5
300/300 [==============================] - 215s 717ms/step - loss: 43.5108 - val_loss: 46.7714
Epoch 8/10
299/300 [============================>.] - ETA: 0s - loss: 39.1087
Epoch 00008: val_loss did not improve from 46.77143
300/300 [==============================] - 215s 717ms/step - loss: 39.0674 - val_loss: 67.4741
Epoch 9/10
299/300 [============================>.] - ETA: 0s - loss: 35.6292
Epoch 00009: val_loss did not improve from 46.77143
300/300 [==============================] - 214s 714ms/step - loss: 35.6392 - val_loss: 50.6347
Epoch 10/10
299/300 [============================>.] - ETA: 0s - loss: 31.9293
Epoch 00010: val_loss improved from 46.77143 to 39.71617, saving model to /content/drive/MyDrive/인공지능자동차데이터/lane_navigation_check.h5
300/300 [==============================] - 217s 722ms/step - loss: 31.8768 - val_loss: 39.7162
```

Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ 모델 학습

- [구글 드라이브]에 결과 파일이 저장되었다.
- lane_navigation_final.h5 파일이 최종적으로 학습된 파일이다.

Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ 결과 확인

- **코랩 셀에 다음과 같은 코드를 추가한다.**

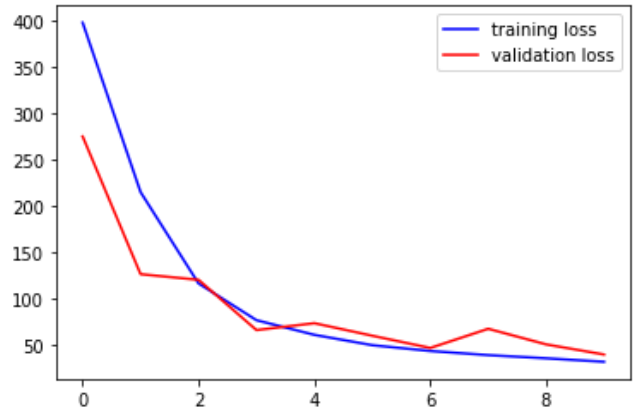
```
1  history.history
2
3  history_path = os.path.join(model_output_dir,'history.pickle')
4  with open(history_path, 'rb') as f:
5      history = pickle.load(f)
6
7  history
8  plt.plot(history['loss'],color='blue')
9  plt.plot(history['val_loss'],color='red')
10 plt.legend(["training loss", "validation loss"])
```

Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ 결과 확인

- **코랩 셀을 실행하여 결과를 확인한다.**

```
<matplotlib.legend.Legend at 0x7fd337a127d0>
```



Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ 검증

- **코랩 셀에 다음과 같은 코드를 추가한다.**

```
1   from sklearn.metrics import mean_squared_error, r2_score
2
3   def summarize_prediction(Y_true, Y_pred):
4
5       mse = mean_squared_error(Y_true, Y_pred)
6       r_squared = r2_score(Y_true, Y_pred)
7
8       print(f'mse  = {mse:.2}')
9       print(f'r_squared = {r_squared:.2%}')
10      print()
11
12  def predict_and_summarize(X, Y):
13      model = load_model(f'{model_output_dir}/lane_navigation_check.h5')
14      Y_pred = model.predict(X)
15      summarize_prediction(Y, Y_pred)
16      return Y_pred
```

Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ 검증

- **코랩 셀에 다음과 같은 코드를 추가한다.**

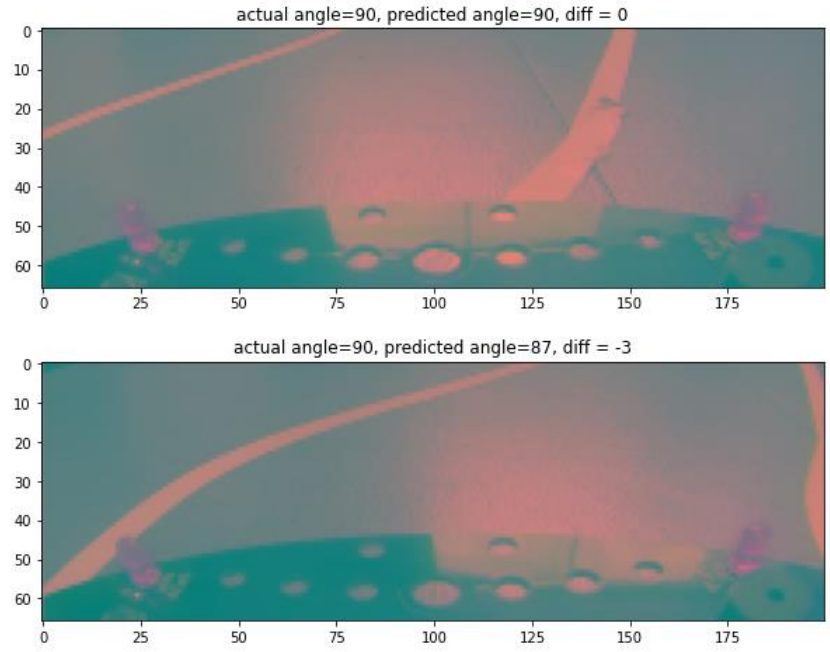
```
17
18  n_tests = 100
19  X_test, y_test = next(image_data_generator(X_valid, y_valid, 100))
20
21  y_pred = predict_and_summarize(X_test, y_test)
22
23  n_tests_show = 2
24  fig, axes = plt.subplots(n_tests_show, 1, figsize=(10, 4 * n_tests_show))
25  for i in range(n_tests_show):
26      axes[i].imshow(X_test[i])
27      axes[i].set_title(f"actual angle={y_test[i]}, predicted angle={int(y_pred[i])}, diff = {int(y_pred[i])-y_test[i]}")
```

Section 02 자율주행을 위한 end-to-end learning 모델 학습

❑ 검증

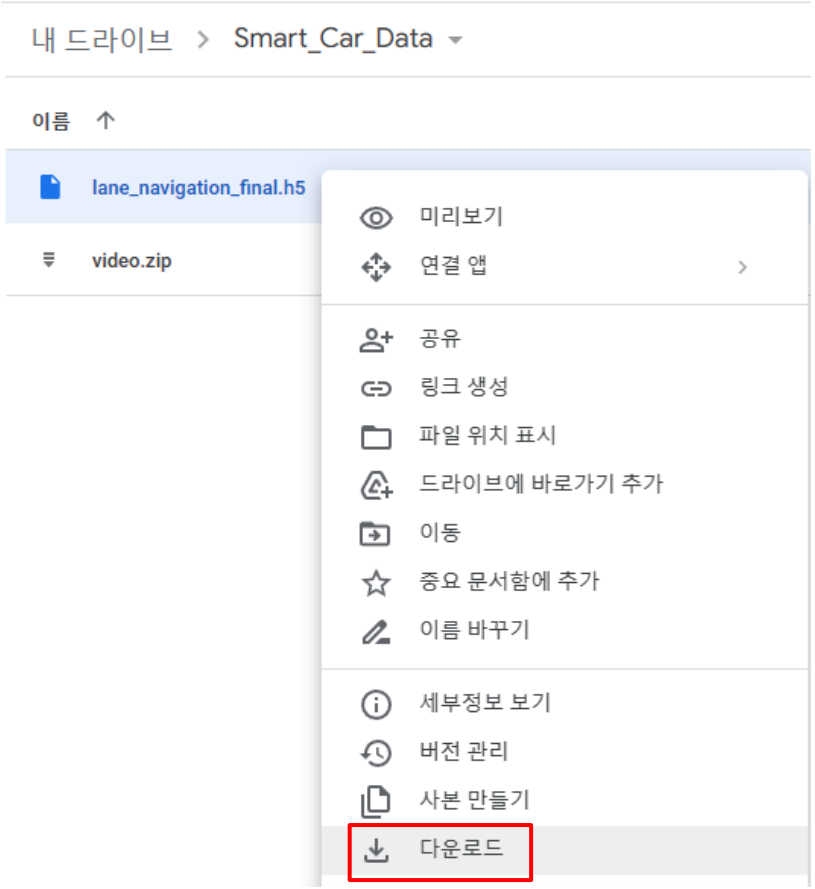
- **코랩 셀을 실행하면 학습 데이터의 성능 평가를 수행한다.**
- actual angle**은 실제 각도**, predicted angle**은 예측각도** diff**는 실제 각도와 예측각도의 차이이다.**
- diff**가 0에 가까울수록 학습이 잘되었다고 판단한다.**

```
mse       = 3.8e+01
r_squared = 96.01%
```



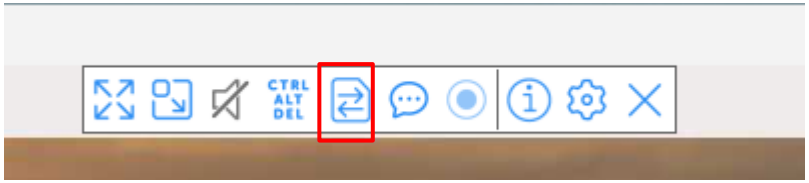
Section 03 학습모델 테스트

- [구글 드라이브]의 [Smart_Car_Data] 폴더에서 생성된 “lane_navigation_final. h5" 파일을 다운로드한다.
- 마우스 오른쪽 버튼을 클릭 후 [다운로드] 버튼을 클릭한다.

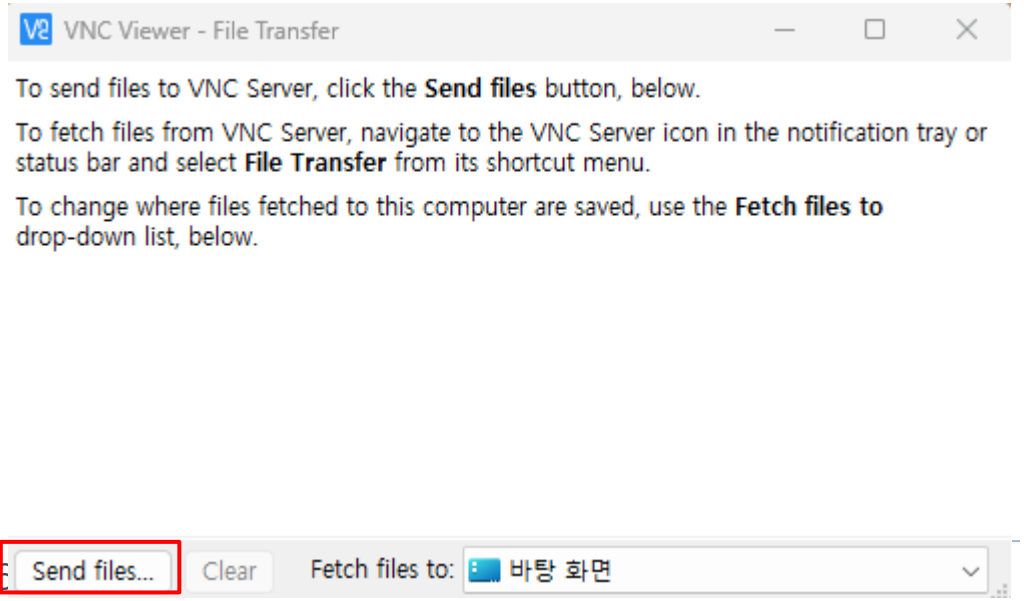


Section 03 학습모델 테스트

- 다운로드 폴더에 다운로드 되었다.
- lane_navigation_final.h5 파일을 라즈베리파이로 보내 보자.
- VNC Viewer에서 파일 전송 아이콘을 클릭한다.

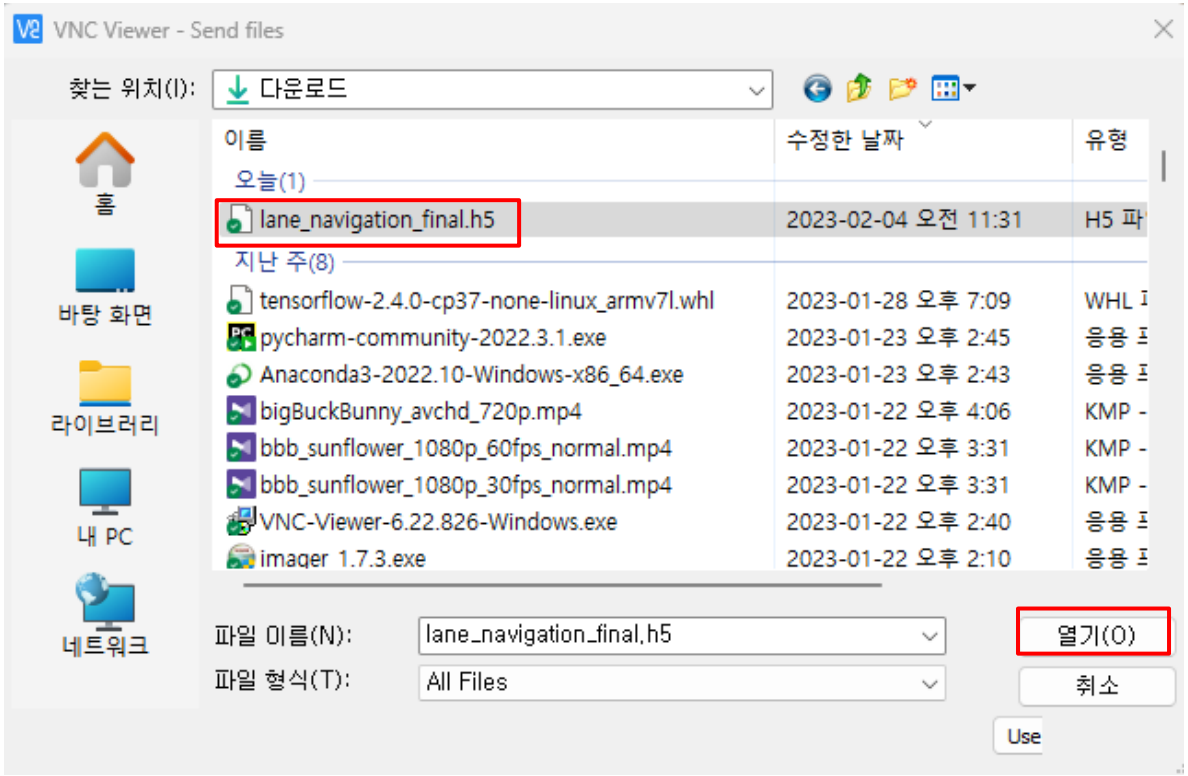


- [Send files ...]을 클릭한다.



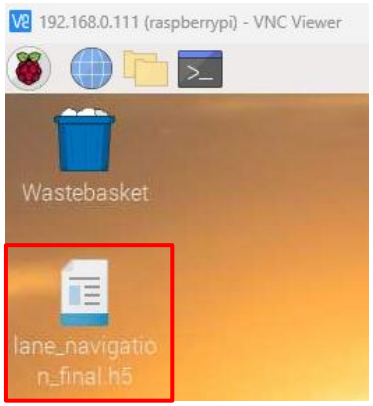
Section 03 학습모델 테스트

❑ [다운로드] 폴더에 있는 lane_navigation_final. h5 파일을 선택 후 [열기] 버튼을 눌러 전송한다.

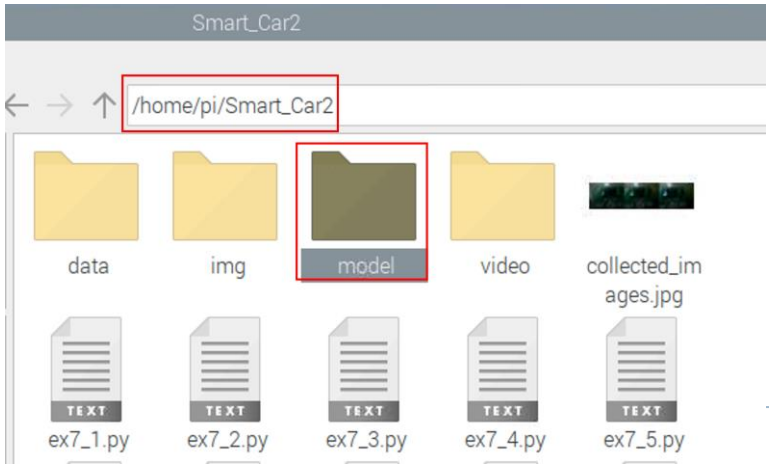


Section 03 학습모델 테스트

❑ 라즈베리파이의 바탕화면에 전송되었다.



❑라즈베리파이에서 [파일 매니저]를 열어 /home/pi/Smart_Car2 폴더에 [model] 폴더를 생성한다.



Section 03 학습모델 테스트

- model 폴더에 전송한 학습 모델 파일을 이동시킨다.
- 다음의 lane_navigation_final.h5의 학습된 결과 파일을 불러와 자동차의 조향각을 예측하는 코드를 만들어 보자.
- ex13_1.py

```
1   import cv2
2   import numpy as np
3   import tensorflow as tf
4   from tensorflow.keras.models import load_model
5
6   def img_preprocess(image):
7       height, _, _ = image.shape
8       image = image[int(height/2):,:,:]
9       image = cv2.cvtColor(image, cv2.COLOR_BGR2YUV)
10      image = cv2.GaussianBlur(image, (3,3), 0)
11      image = cv2.resize(image, (200,66))
12      image = image / 255
13      return image
14
```

Section 03 학습모델 테스트

❑ ex13_1.py

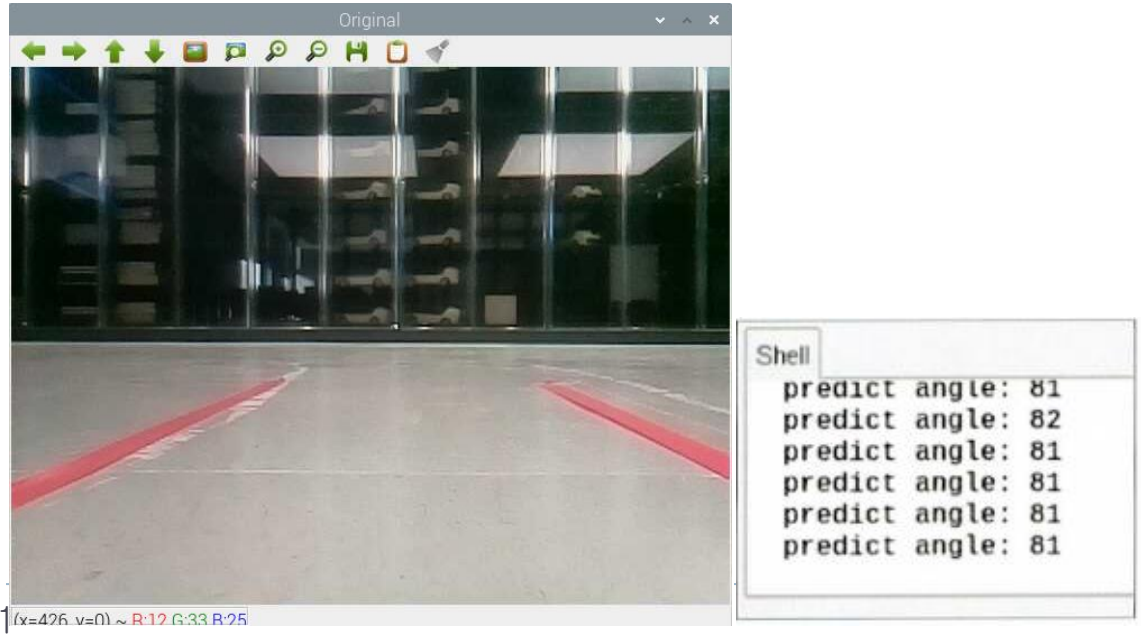
```
15  cam = cv2.VideoCapture(0)
16  cam.set(3, 640)
17  cam.set(4, 480)
18  model_path = '/home/pi/Smart_Car2/model/lane_navigation_final.h5'
19  model = load_model(model_path)
20
21  carState = "stop"
22
23  while( cam.isOpened()):
24      keyVal = cv2.waitKey(0)
25
26      if keyVal == ord('q') :
27          break
28
29      _, image = cam.read()
30      cv2.imshow('Original', image)
31
32      preprocessed = img_preprocess(image)
33      cv2.imshow('pre', preprocessed)
34
```

Section 03 학습모델 테스트

❑ ex13_1.py

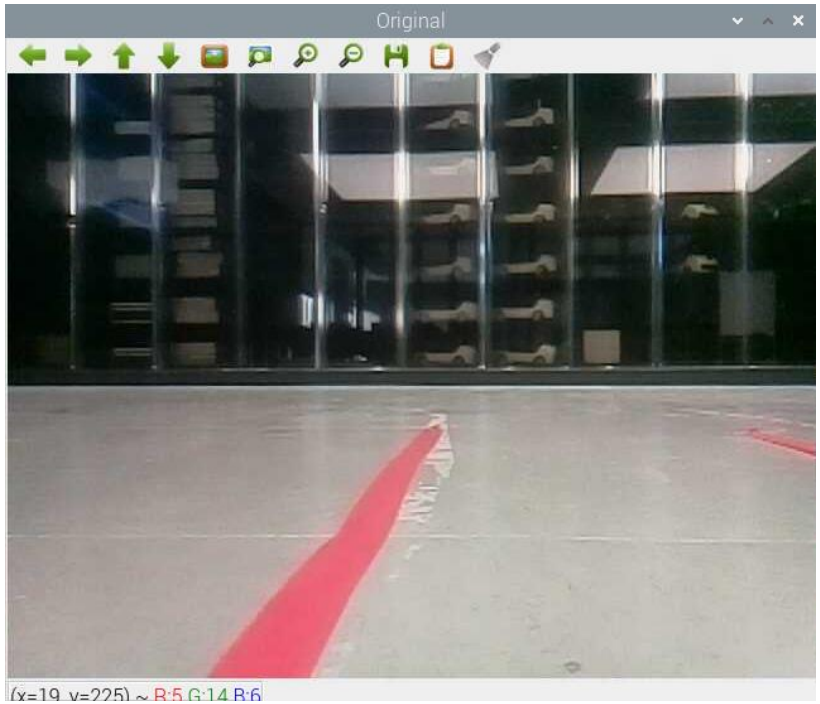
```
35      X = np.asarray([preprocessed])
36      steering_angle = int(model.predict(X)[0])
37      print("predict angle:",steering_angle)
38
39  cv2.destroyAllWindows()
```

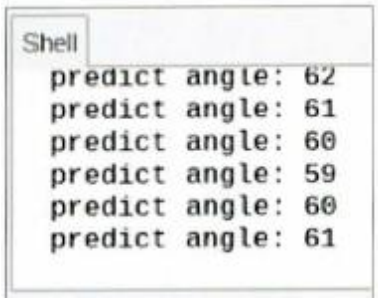
❑ 코드를 실행하여 결과를 확인하면, 선이 중간쯤으로의 조향각도는 81도를 예측하였다.



Section 03 학습모델 테스트

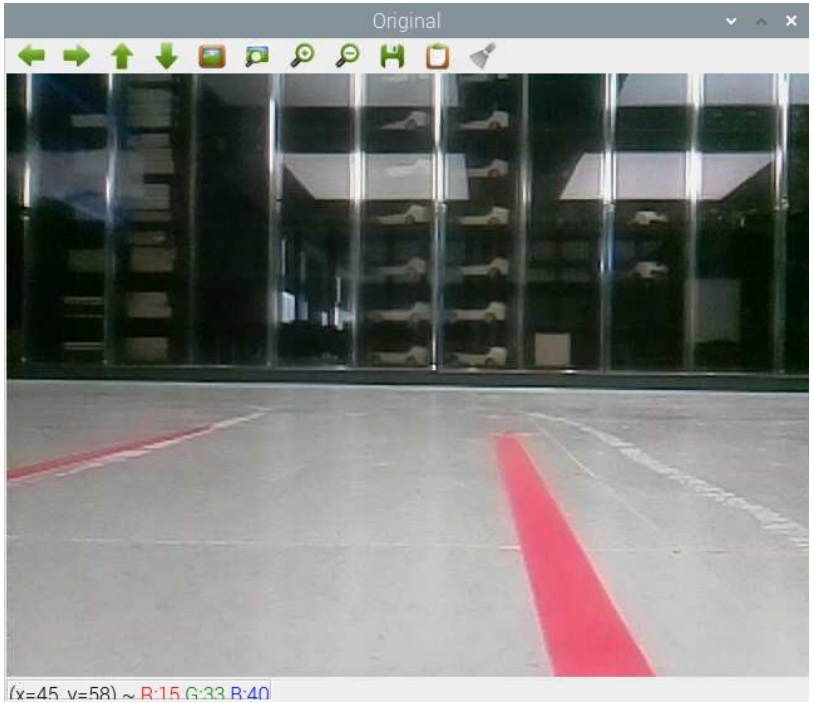
❑ 선이 왼쪽으로 치우쳐있어 자동차를 오른쪽으로 이동하기 위해 133도의 각도를 예측하였다.

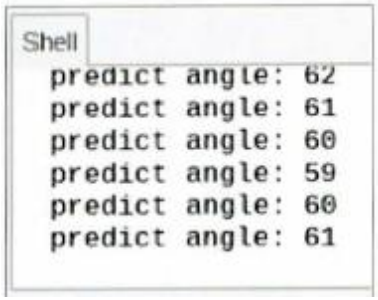




Section 03 학습모델 테스트

❑ 선이 오른쪽 치우쳐있어 자동차를 왼쪽으로 이동하기 위해 60도의 각도를 예측하였다.





Section 04 예측값을 통한 자동차의 진행방향 결정

❑ 다음과 같은 코드를 작성한다.

❑ Ex13_2.py

```
1   import cv2
2   import numpy as np
3   import tensorflow as tf
4   from tensorflow.keras.models import load_model
5
6   def img_preprocess(image):
7       height, _, _ = image.shape
8       image = image[int(height/2):, :, :]
9       image = cv2.cvtColor(image, cv2.COLOR_BGR2YUV)
10      image = cv2.GaussianBlur(image, (3,3), 0)
11      image = cv2.resize(image, (200,66))
12      image = image / 255
13      return image
14
```

Section 04 예측값을 통한 자동차의 진행방향 결정

❑ Ex13_2.py

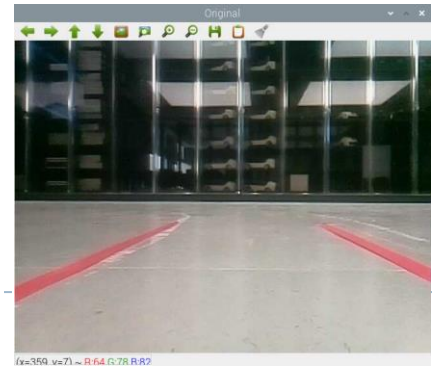
```
15  cam = cv2.VideoCapture(0)
16  cam.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
17  cam.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
18  model_path = '/home/pi/Smart_Car2/model/lane_navigation_final.h5'
19  model = load_model(model_path)
20
21  carState = "stop"
22
23  while( cam.isOpened()):
24      keyVal = cv2.waitKey(1)
25
26      if keyVal == ord('q') :
27          break
28
29      _, image = cam.read()
30      cv2.imshow('Original', image)
31
32      preprocessed = img_preprocess(image)
33      cv2.imshow('pre', preprocessed)
34
```

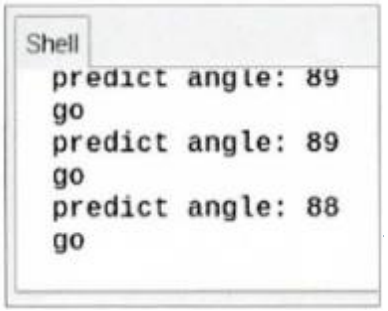
Section 04 예측값을 통한 자동차의 진행방향 결정

❑ Ex13_2.py

```
35      X = np.asarray([preprocessed])
36      steering_angle = int(model.predict(X)[0])
37      print("predict angle:",steering_angle)
38
39      if steering_angle >= 85 and steering_angle <= 95:
40          print("go")
41      elif steering_angle > 96:
42          print("right")
43      elif steering_angle < 84:
44          print("left")
45
46  cv2.destroyAllWindows()
```

❑ 코드를 실행하여 결과를 확인한다.





Section 05 자동차 자율주행 구현

❑ 다음과 같은 코드를 작성한다.

❑ Ex13-3.py

```
1   import cv2
2   from ctypes import *
3   import numpy as np
4   import tensorflow as tf
5   from tensorflow.keras.models import load_model
6   import os
7   import time
8
9   WiringPi = CDLL(“/home/pi/WiringPi/wiringPi/libwiringPi.so.2.70”, mode=RTLD_GLOBAL)
10  swcar = cdll.LoadLibrary(‘/home/pi/swcar/libswcar.so’)
11
12  motor_status = "STOP"
13  servo_status = "CENTER"
14
```

Section 05 자동차 자율주행 구현

❑ 다음과 같은 코드를 작성한다.

❑ Ex13-3.py

```
15  def motor_forward():
16      global motor_status
17      if (motor_status == "REVERSE") :
18          swcar.SIO_ForwardMotor(0)
19          time.sleep(0.1)
20      swcar.SIO_ForwardMotor(20)
21      motor_status = "FORWARD"
22
23  def motor_reverse():
24      global motor_status
25      if (motor_status == "FORWARD") :
26          swcar.SIO_ReverseMotor(0)
27          time.sleep(0.1)
28      swcar.SIO_ReverseMotor(20)
29      motor_status = "REVERSE"
30
31  def motor_stop():
32      global motor_status
33      swcar.SIO_ForwardMotor(0)
34      motor_status = "STOP"
35
```

Section 05 자동차 자율주행 구현

❑ 다음과 같은 코드를 작성한다.

❑ Ex13-3.py

```
36  def servo_left():
37      global servo_status
38      if(motor_status == "RIGHT"):
39          swcar.SIO_WriteServo(100, 50)
40          time.sleep(0.1)
41      swcar.SIO_WriteServo(100, 90)
42       servo_status = "LEFT"
43
44  def servo_right():
45      global servo_status
46       if(motor_status == "RIGHT"):
47          swcar.SIO_WriteServo(100, 50)
48          time.sleep(0.1)
49      swcar.SIO_WriteServo(100, 10)
50      servo_status = "RIGHT"
51
52  def servo_center():
53     global servo_status
54      swcar.SIO_WriteServo(100, 50)
55      servo_status = "CENTER"
56
```

Section 05 자동차 자율주행 구현

❑ Ex13-3.py

```
57  swcar.SIO_Init(0)
58
59  motor_stop()
60  servo_center()
61
62  def img_preprocess(image):
63      height, _, _ = image.shape
64      image = image[int(height/2):, :, :]
65      image = cv2.cvtColor(image, cv2.COLOR_BGR2YUV)
66      image = cv2.GaussianBlur(image, (3,3), 0)
67      image = cv2.resize(image, (200,66))
68      image = image / 255
69      return image
70
71  cam = cv2.VideoCapture(0)
72  cam.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
73  cam.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
74  model_path = '/home/pi/Smart_Car2/model/lane_navigation_final.h5'
75  model = load_model(model_path)
76
77  carState = "stop"
78
```

Section 05 자동차 자율주행 구현

❑ Ex13-3.py

```
79  while( cam.isOpened()):
80      _, image = cam.read()
81      cv2.imshow('Original', image)
82
83      keyValue = cv2.waitKey(1)
84      if keyValue == ord('q') :
85          motor_stop()
86          servo_center()
87          break
88      elif keyValue == 82 :
89          print("go")
90          carState = "go"
91      elif keyValue == 84 :
92          print("stop")
93          carState = "stop"
94
95      preprocessed = img_preprocess(image)
96      cv2.imshow('pre', preprocessed)
97
```

Section 05 자동차 자율주행 구현

```
98      X = np.asarray([preprocessed])
99      steering_angle = int(model.predict(X)[0])
100     print("predict angle:",steering_angle)
101
102     if carState == "go":
103        if steering_angle >= 85 and steering_angle <= 95:
104           print("go")
105           motor_forward()
106           time.sleep(0.3)
107           motor_stop()
108        elif steering_angle > 96:
109           print("right")
110            servo_right()
111            motor_forward()
112            time.sleep(0.3)
113            motor_stop()
114        elif steering_angle < 84:
115           print("left")
116           servo_left()
117           motor_forward()
118           time.sleep(0.3)
119           motor_stop()
120     elif carState == "stop":
121        motor_stop()
122
123  cv2.destroyAllWindows()
```

Q&A