차량공유업체의 차량 파손 여부 분류 프로젝트

김영우

1. 문제 정의

[차량 공유 업체의 고민]

“하루 평균 7~8만장의 차량 사진을 **일일히 수작업을 통해 파손된 차량을 식별하고** 차량 정비를 맡기는 업무가 너무 시간이 오래 걸려요”

“차량 이미지 사진을 통해 차량을 식별하고 알려주는 서비스를 원해요”

1. 문제 정의

[역할] DIGICO KT 관련 프로젝트 계획 및 개발 담당 사원

[업무] 차량공유업체에게 필요한 차량 파손 여부를 알려주는 서비스를 개발합니다.

2. 차량 파손 여부 분류에 사용할 데이터셋

[데이터명] 차량정보 데이터

[원본 데이터 확보 방법] 이미지 생성을 통한 이미지 수집

[제작 방법] 이미지 수집 후, 정상/파손 여부 수작업 분류(normal, abnormal 폴더로 수작업 분류)

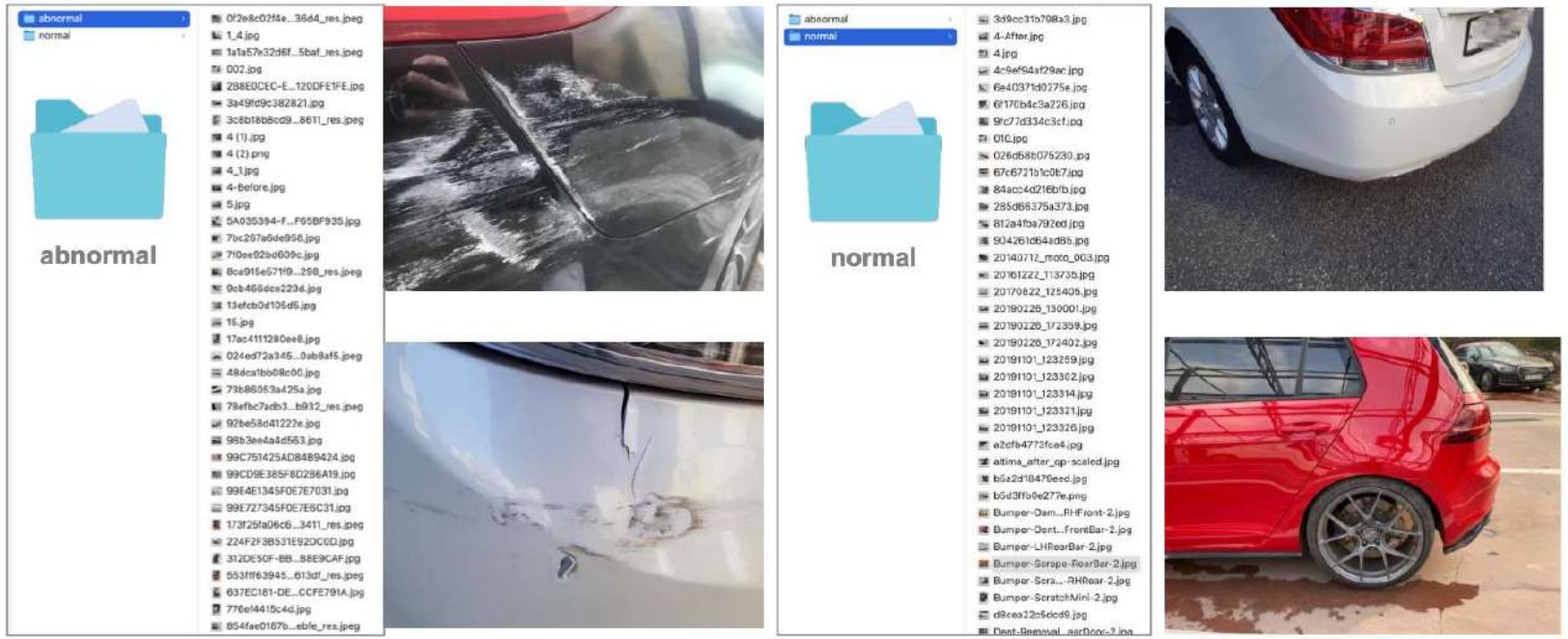
[구성 내용] 파일형태 : 이미지
파일건수 : 605개(정상 차량 이미지 302개, 파손 차량 이미지 303개)
파일용량 : 910MB

[데이터제공파일] Car_Image.zip

2. 데이터셋 - 상세

[데이터] Car_Image.zip

- 차량 파손 여부에 따라 abnormal(비정상, 파손 있음), normal(정상, 파손 없음) 폴더로 구분



3. 수행 절차 조건


Challenge
과제 수행 전략
데이터 전처리
다양한 이미지 데이터
▪ 차량의 정상/파손 개별 이미지
부족한 학습 이미지 데이터
▪ 이미지 모델링으로 적절한 성능을 얻기 위한 학습용 데이터 부족
모델링
모델 성능을 높이기
▪ 직접 설계 및 외부 모델 활용 중 어느 것이 모델 성능에 좋을까?
일관성 있는 데이터 구조 만들기
✔ 학습을 위해 통일된 폴더구조로 정리
✔ 학습용, 검증용, 테스트용 분류
직접 설계 모델 vs 성능개선(Data Augmentation)
✔ 직접 설계한 모델과 ImageDataGenerator를 활용하여 기존 데이터 증식하여 학습된 모델과 성능비교
직접 설계 모델 vs 성능개선(Transfer Learning)
✔ 직접 설계한 모델과 이미 생성된 모델(Transfer Learning)로 학습하고 성능비교

4. preview

- 본 프로젝트는 4가지 실험을 하였다.
  1. 차량 이미지를 Train, Validation, Test 셋으로 구분하여, CNN 모델링을 하였다. -> 모델링 1-1

  2. 차량 이미지를 Train, Valiadtion, Test 셋으로 구분하여,
     사전 학습된 모델 (VRR16, ResNet, Inception V3, EfficientNet)를 활용했다. -> 모델링 1-2

  3. 차량 이미지에 대한 Train과 Validation을 Image Augmentation(이미지 증강) 하여
     CNN 모델링을 하였다. -> 모델링 2-1

  4. 차량 이미지에 대한 Train과 Validation을 Image Augmentation(이미지 증강)하여
     사전 학습된 모델 (VRR16, ResNet, Inception V3, EfficientNet)를 활용했다. -> 모델링 2-2

5. 모델링 1을 위한 전처리

[모델링 1 을 위한 데이터 구조 만들기]

X : 이미지를 array 형태로 변환합니다.

y: 이미지 갯수만큼 normal – 0, abnormal – 1로 array를 만듭니다.

5. 모델링 1 을 위한 전처리

X – 파손 차량 이미지와 정상 차량 이미지를 concat 한다.

```
abnormal_images = os.listdir(f'{path}/abnormal') # 파손 차량 이미지들
normal_images = os.listdir(f'{path}/normal') # 정상 차량 이미지들

# 가져온 이미지 파일 리스트 출력
print(abnormal_images)  # abnormal 폴더 내의 파일 리스트 출력
print(normal_images)    # normal 폴더 내의 파일 리스트 출력
```

```
# 파손 차량 이미지와 정상 차량 이미지들을 합친다. 총 605개 이미지가 존재할 것이다.
car_images = abnormal_images + normal_images

print(car_images[0:5:1])
print(len(car_images))
```

```
['DALLíñE 2023-03-10 18.51.24 - scratched car.png', 'DALLíñE 2023-03-10 18.53.58 - slightly damaged car.png', 'DALLíñE 20
605
```

5. 모델링 1을 위한 전처리

X – 어디까지가 파손 차량 이미지인지, 정상 차량 이미지인지 확인한다.

```
car_images[0:303:1] # 여기까지가 파손 차량 이미지
```

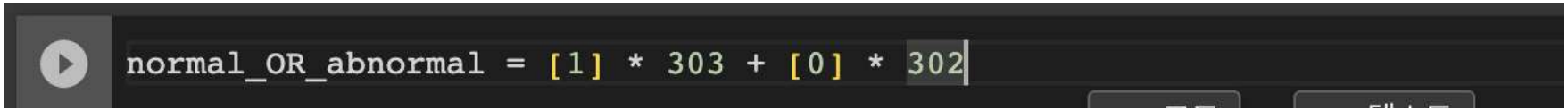
```
['DALLíñE 2023-03-10 18.51.24 - scratched car.png',
 'DALLíñE 2023-03-10 18.53.58 - slightly damaged car.png',
 'DALLíñE 2023-03-10 18.51.29 - scratched car.png',
 'DALLíñE 2023-03-10 18.51.32 - scratched car.png',
 'DALLíñE 2023-03-10 18.54.42 - slightly damaged car.png',
 'DALLíñE 2023-03-10 18.53.08 - scratched car.png',
 'DALLíñE 2023-03-10 18.54.17 - slightly damaged car.png',
 'DALLíñE 2023-03-10 18.54.19 - slightly damaged car.png',
 'DALLíñE 2023-03-10 18.51.26 - scratched car.png',
 'DALLíñE 2023-03-10 18.54.24 - slightly damaged car.png',
```

```
car_images[303::1] # 여기는 정상 차량 이미지
```

```
 'DALLíñE 2023-03-11 01.01.22 - photo of a part of car.png',
 'DALLíñE 2023-03-11 00.58.44 - photo of a part of car.png',
 'DALLíñE 2023-03-11 14.27.59 - part of a car.png',
 'DALLíñE 2023-03-11 14.35.20 - part of a car.png',
 'DALLíñE 2023-03-11 14.24.16 - part of a car.png',
 'DALLíñE 2023-03-10 23.29.18 - photo of a part of car without blemish.png',
 'DALLíñE 2023-03-10 23.35.37 - photo of a part of car without blemish.png',
 'DALLíñE 2023-03-11 01.36.47 - a part of a car.png',
 'DALLíñE 2023-03-11 00.51.38 - photo of a part of car.png',
 'DALLíñE 2023-03-11 00.16.32 - photo of a part of car without blemish.png',
 'DALLíñE 2023-03-11 14.33.28 - part of a car.png',
 'DALLíñE 2023-03-11 14.40.47 - photo of part of a car.png',
 'DALLíñE 2023-03-11 01.04.47 - photo of a part of car.png',
 'DALLíñE 2023-03-11 01.04.49 - photo of a part of car.png',
```

5. 모델링 1을 위한 전처리

y에 대한 normal – 0, abnormal – 1를 마련한다.



5. 모델링 1을 위한 전처리

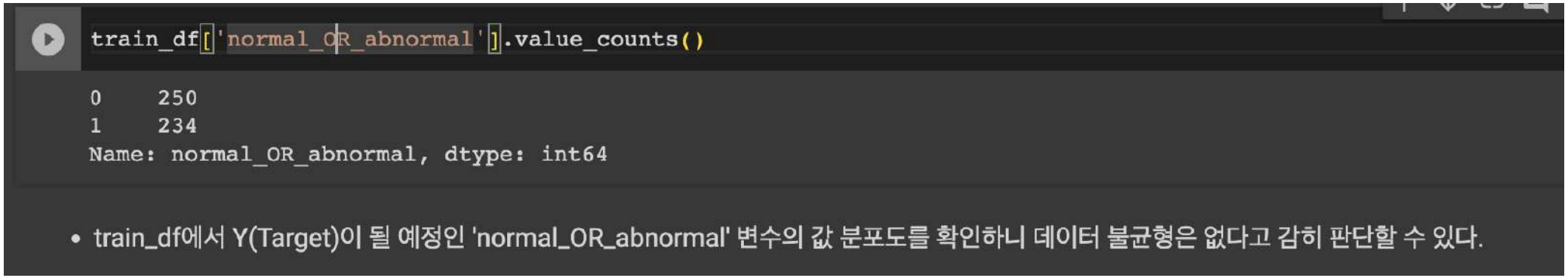
X와 Y로 구성된 데이터프레임(DataFrame)으로 만든 후, Train, Validation, Test 셋으로 구분한다.

```
[ ] from sklearn.model_selection import train_test_split

[ ] # 전체 데이터를 80% train set, 20% (test set + validation set)로 분리
    train_df, test_val_df = train_test_split(df, test_size=0.2, random_state=42)

    # (test set + validation set)를 test set과 validation set으로 분리 (각각 50%)
    test_df, val_df = train_test_split(test_val_df, test_size=0.5, random_state=42)

[ ] train_df.shape, val_df.shape, test_df.shape

    ((484, 2), (61, 2), (60, 2))
```

Train 데이터 484개, validation 데이터 61개, test 데이터 60개로 비교적 데이터 건수가 작은 것을 확인할 수 있다.
따라서 **Image Augmentation을 사용해볼까 하는 생각**을 할 수 있다.

5. 모델링 1을 위한 전처리

CNN 모델링에 활용할 Train 데이터셋을 보니, Y에 대한 0값이 250개, 1값이 234개
비교적 **데이터가 균형**하다는 것을 알 수 있다.
따라서 데이터 불균형 처리를 하기 위한 방법은 굳이 생각을 안해도 될 것 같다는 생각이 들었다.



5. 모델링 1을 위한 전처리

이미지(image)를 모델 입력형태로 적합한 np.array 형태로 변환해야 한다.
Train에 있는 이미지(image)를 np.array 형태로 변환한다.

- tran_df의 car_images에 있는 차량 파일 이름들(총 484개)을 가지고 np.array 형태로 만든다.

```
# 예시: 데이터프레임 내의 이미지 파일을 NumPy 배열로 변환하는 방법
import os
from keras.preprocessing.image import load_img, img_to_array
import numpy as np

image_file_names = train_df['car_images']  # 이미지 파일 경로가 있는 열
flag = train_df['normal_OR_abnormal'] # 0이냐 1이냐?

# 각 이미지를 NumPy 배열로 변환
image_arrays = []
for ifn, flag in zip(image_file_names, flag):
    if flag == 0: # 정상 차량 이미지인 경우
        img = load_img(f'{path}/normal/{ifn}', target_size=(280, 280)) # target_size를 (280, 280)만 줘도 된다고 한다.
    else: # 파손 차량 이미지인 경우
        img = load_img(f'{path}/abnormal/{ifn}', target_size=(280, 280)) # target_size를 (280, 280)만 줘도 된다고 한

    img_array = img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    image_arrays.append(img_array)

# 모든 이미지 배열을 하나의 배열로 결합
train_X = np.concatenate(image_arrays, axis=0)
```

본래 이미지는 (1024, 1024, 3) 이었는데, 이미지가 너무 크면, 학습 시간이 소요되고, 메모리 부족 현상이 발생할 수 있기 때문에 이미지 크기를 (280, 280, 3)으로 변환했다.

5. 모델링 1을 위한 전처리

이미지(image)를 모델 입력형태로 적합한 np.array 형태로 변환해야 한다.
Validation 에 있는 이미지(image)를 np.array 형태로 변환한다.

- val_df의 car_images에 있는 차량 파일 이름들(총 61개)을 가지고 np.array 형태로 만든다.

```
# 예시: 데이터프레임 내의 이미지 파일을 NumPy 배열로 변환하는 방법
import os
from keras.preprocessing.image import load_img, img_to_array
import numpy as np

image_file_names = val_df['car_images']  # 이미지 파일 경로가 있는 열
flag = val_df['normal_OR_abnormal'] # 0이냐 1이냐?

# 각 이미지를 NumPy 배열로 변환
image_arrays = []
for ifn, flag in zip(image_file_names, flag):
    if flag == 0: # 정상 차량 이미지인 경우
       img = load_img(f'{path}/normal/{ifn}', target_size=(280, 280)) # target_size를 (280, 280)만 줘도 된다고 한다. 1
    else: # 파손 차량 이미지인 경우
       img = load_img(f'{path}/abnormal/{ifn}', target_size=(280, 280)) # target_size를 (280, 280)만 줘도 된다고 한다.

    img_array = img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    image_arrays.append(img_array)

# 모든 이미지 배열을 하나의 배열로 결합
val_X = np.concatenate(image_arrays, axis=0)
```

본래 이미지는 (1024, 1024, 3) 이었는데, 이미지가 너무 크면, 학습 시간이 소요되고, 메모리 부족 현상이 발생할 수 있기 때문에 이미지 크기를 (280, 280, 3)으로 변환했다.

5. 모델링 1을 위한 전처리

이미지(image)를 모델 입력형태로 적합한 np.array 형태로 변환해야 한다.
Test 에 있는 이미지(image)를 np.array 형태로 변환한다.

- test_df의 car_images에 있는 차량 파일 이름들(총 60개)을 가지고 np.array 형태로 만든다.

```
# 예시: 데이터프레임 내의 이미지 파일을 NumPy 배열로 변환하는 방법
import os
from keras.preprocessing.image import load_img, img_to_array
import numpy as np

image_file_names = test_df['car_images']  # 이미지 파일 경로가 있는 열
flag = test_df['normal_OR_abnormal'] # 0이나 1이나?

# 각 이미지를 NumPy 배열로 변환
image_arrays = []
for ifn, flag in zip(image_file_names, flag):
    if flag == 0: # 정상 차량 이미지인 경우
        img = load_img(f'{path}/normal/{ifn}', target_size=(280, 280)) # target_size를 (280, 280)만 줘도 된다고 한다. l
    else: # 파손 차량 이미지인 경우
        img = load_img(f'{path}/abnormal/{ifn}', target_size=(280, 280)) # target_size를 (280, 280)만 줘도 된다고 한다.

    img_array = img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    image_arrays.append(img_array)

# 모든 이미지 배열을 하나의 배열로 결합
test_X = np.concatenate(image_arrays, axis=0)
```

본래 이미지는 (1024, 1024, 3) 이었는데, 이미지가 너무 크면, 학습 시간이 소요되고, 메모리 부족 현상이 발생할 수 있기 때문에 이미지 크기를 (280, 280, 3)으로 변환했다.

6. 모델링 1-1

- CNN 모델링을 하여 3개의 모델을 만들어서, 정확도를 확인한다.

6. 모델링 1- 1

1번째 모델 : Convolutional Layer 32 -> 64 -> 128로 점진적으로 증가
Fully Connected Layer 추가 후 한 층을 추가

```
clear_session()

# 모델 초기화
model1 = Sequential()

# Convolutional 레이어 추가 32 -> 64 -> 128로 진행
model1.add(Conv2D(32, (3, 3), activation='swish', input_shape=(280, 280, 3))) # 이미지 형태가 (280, 280, 3)이기 때문에 input_s
model1.add(MaxPooling2D((2, 2)))
model1.add(Conv2D(64, (3, 3), activation='swish'))
model1.add(MaxPooling2D((2, 2)))
model1.add(Conv2D(128, (3, 3), activation='swish'))
model1.add(MaxPooling2D((2, 2)))

# Fully Connected 레이어 추가 -> 층을 한번 쌓아봤습니다
model1.add(Flatten())
model1.add(Dense(128, activation='swish'))
model1.add(Dense(1, activation='sigmoid'))  # Binary classification이므로 마지막 레이어는 1개의 뉴런과 sigmoid 활성화 함수

# 모델 컴파일
model1.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Early Stopping 정의
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
```

6. 모델링 1- 1

정상 차량(0)로에 예측한 것이 실제로 정상 차량(0)으로 비교적 판단을 잘하고 있다.
파손 차량(1)로 예측한 것이 실제로 파손 차량(1)으로 절반 높게 잘 판단하고 있다.

```
[ ] # 모델 훈련
    history = model1.fit(train_X,
                         train_df['normal_OR_abnormal'],
                         epochs=30,
                         batch_size=32,
                         validation_data=(val_X, val_df['normal_OR_abnormal']),
                         callbacks=[early_stopping])

    # 훈련된 모델로 예측
    y_pred = model1.predict(test_X) # 총 60개 있을 것이구요.. 왜냐하면 test_X 60개 였죠?
                                    # 그래서 sigmoid()를 통해 0 ~ 1 값의 범위를 가질 것이구요

    # 예측값 0으로 확정지을지, 1로 확정지을지 판단
    y_pred_binary = (y_pred > 0.5).astype(int) # 우리가 기준을 둬야 합니다. 예를 들어 0.5 초과이면 1, 아니면 0 이런식으로 결정을 지어야 합니다..
```

```
Accuracy: 0.75
Precision: 0.9230769230769231
Recall: 0.6486486486486487
F1 Score: 0.7619047619047619
Confusion Matrix:
[[21  2]
 [13 24]]
```

6. 모델링 1- 1

2번째 모델 : Convolutional Layer 128 -> 64 -> 32로 점진적으로 진행
Fully Connected Layer 추가 후 두 층을 추가
Dropout 진행

```
clear_session()

# 모델 초기화
model2 = Sequential()

# Convolutional 레이어 추가 128 -> 64 -> 32로 진행, Dropout 진행
model2.add(Conv2D(128, (3, 3), activation='swish', input_shape=(280, 280, 3)))
model2.add(MaxPooling2D((2, 2)))
model2.add(Dropout(0.3))

model2.add(Conv2D(64, (3, 3), activation='swish'))
model2.add(MaxPooling2D((2, 2)))
model2.add(Dropout(0.3))

model2.add(Conv2D(32, (3, 3), activation='swish'))
model2.add(MaxPooling2D((2, 2)))
model2.add(Dropout(0.3))

# Fully Connected 레이어 추가 -> 층을 조금 더 쌓아보기로 했다.
model2.add(Flatten())
model2.add(Dense(128, activation='swish'))
model2.add(Dropout(0.3))
model2.add(Dense(64, activation='swish'))
model2.add(Dropout(0.3))
model2.add(Dense(1, activation='sigmoid'))  # Binary classification

# 모델 컴파일
model2.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Early Stopping 정의
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
```

6. 모델링 1- 1

파손 차량(1)로 예측했지만, 실제 정상 차량(0) 인 경우가 비교적 많아,
정확도가 떨어지는 것을 알 수 있다.

```python
# 모델 훈련
history = model2.fit(train_X,
                     train_df['normal_OR_abnormal'],
                     epochs=30,
                     batch_size=16,
                     validation_data=(val_X, val_df['normal_OR_abnormal']),
                     callbacks=[early_stopping])

# 훈련된 모델로 예측
y_pred = model2.predict(test_X) # 총 60개 있을 것이구요.. 왜냐하면 test_X 60개 였죠?
                                # 그래서 sigmoid()를 통해 0 ~ 1 값의 범위를 가질 것이구요

# 예측값 0으로 확정지을지, 1로 확정지을지 판단
y_pred_binary = (y_pred > 0.5).astype(int) # 우리가 기준을 둬야 합니다. 예를 들어 0.5 초과이면 1, 아니면 0 이런식으로 결정을 지어야 합니다.
```

```
Accuracy: 0.65
Precision: 0.6538461538461539
Recall: 0.918918918918919
F1 Score: 0.7640449438202248
Confusion Matrix:
[[ 5 18]
 [ 3 34]]
```

6. 모델링 1- 1

3번째 모델 : Convolutional Layer 256 -> 128 -> 64 -> 32로 점진적으로 진행
Fully Connected Layer 추가
Dropout 진행, BatchNormalization을 사용

```
clear_session()

# 모델 초기화
model3 = Sequential()

# Convolutional 레이어 추가 -> 512, 256, 128, 32 , BatchNormalization 활용
model3.add(Conv2D(256, (3, 3), activation='swish', input_shape=(280, 280, 3)))
model3.add(BatchNormalization())
model3.add(MaxPooling2D((2, 2)))
model3.add(Dropout(0.3))

model3.add(Conv2D(128, (3, 3), activation='swish'))
model3.add(BatchNormalization())
model3.add(MaxPooling2D((2, 2)))
model3.add(Dropout(0.3))

model3.add(Conv2D(64, (3, 3), activation='swish'))
model3.add(BatchNormalization())
model3.add(MaxPooling2D((2, 2)))
model3.add(Dropout(0.3))

model3.add(Conv2D(32, (3, 3), activation='swish'))
model3.add(BatchNormalization())
model3.add(MaxPooling2D((2, 2)))
model3.add(Dropout(0.3))

# Fully Connected 레이어 추가 -> 층을 안쌓아봤습니다.
model3.add(Flatten())
model3.add(Dense(1, activation='sigmoid'))  # Binary classification

# 모델 컴파일
model3.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Early Stopping 정의
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
```

6. 모델링 1- 1

정상 차량(0)로 예측했지만, 실제 파손 차량(1) 인 경우가 비교적 많아,
정확도가 떨어지는 것을 알 수 있다.

```
# 모델 훈련
history = model3.fit(train_X,
                     train_df['normal_OR_abnormal'],
                     epochs=30,
                     batch_size=16,
                     validation_data=(val_X, val_df['normal_OR_abnormal']),
                     callbacks=[early_stopping])

# 훈련된 모델로 예측
y_pred = model3.predict(test_X) # 총 60개 있을 것이구요.. 왜냐하면 test_X 60개 였죠?
                                # 그래서 sigmoid()를 통해 0 ~ 1 값의 범위를 가질 것이구요

# 예측값 0으로 확정지을지, 1로 확정지을지 판단
y_pred_binary = (y_pred > 0.5).astype(int) # 우리가 기준을 둬야 합니다. 예를 들어 0.5 초과이면 1, 아니면 0 이런식으로 결정을 지어야 합니다..
```

```
Accuracy: 0.6833333333333333
Precision: 0.8
Recall: 0.6486486486486487
F1 Score: 0.7164179104477612
Confusion Matrix:
[[17  6]
 [13 24]]
```

7. 모델링 1 - 2

- 모델링 1용 전처리했던 것을 사전 학습된 모델(VRR16, ResNet, Inception V3, EfficientNet)을 활용한다.

7. 모델링 1- 2

Vgg16 모델을 바탕으로 층을 더 쌓았다.

```
[ ] # 새로운 모델 구조 설계
    model9 = Sequential()
    model9.add(vgg16)            # VGG16 모델
    model9.add(Flatten())  # Flatten 층 추가
    model9.add(Dense(256, activation='swish'))  # Dense 층 추가
    model9.add(Dropout(0.3))  # Dropout 층 추가 (과적합 방지)
    model9.add(BatchNormalization())  # BatchNormalization 층 추가
    model9.add(Dense(128, activation='swish'))  # Dense 층 추가
    model9.add(Dropout(0.3))  # Dropout 층 추가 (과적합 방지)
    model9.add(BatchNormalization())  # BatchNormalization 층 추가
    model9.add(Dense(64, activation='swish'))  # Dense 층 추가
    model9.add(Dropout(0.3))  # Dropout 층 추가 (과적합 방지)
    model9.add(BatchNormalization())  # BatchNormalization 층 추가
    model9.add(Dense(1, activation='sigmoid'))  # 출력층 추가 (이진 분류를 위해 sigmoid 활성화 함수 사용)

    # 모델 컴파일
    model9.compile(optimizer='adam',
                   loss='binary_crossentropy',
                   metrics=['accuracy'])
```

7. 모델링 1 - 2

CNN 모델을 사용했을 때보다 정확도가 90대 초반으로 올라간 것을 알 수 있다.

```
[ ] history = model9.fit(train_X,
                         train_df['normal_OR_abnormal'],
                         epochs=30,
                         batch_size=32,
                         validation_data=(val_X, val_df['normal_OR_abnormal']),
                         callbacks=[early_stopping])

    # 훈련된 모델로 예측
    y_pred = model9.predict(test_X)  # sigmoid()를 통해 각각은 0 ~ 1 값의 범위를 가질 것이구요

    # 예측값 0으로 확정지을지, 1로 확정지을지 판단
    y_pred_binary = (y_pred > 0.5).astype(int) # 우리가 기준을 둬야 합니다. 예를 들어 0.5 초과이면 1, 아니면 0 이런식으로 결정을 지어야 합니다..
```

```
Accuracy: 0.9333333333333333
Precision: 0.9714285714285714
Recall: 0.918918918918919
F1 Score: 0.9444444444444445
Confusion Matrix:
[[22  1]
 [ 3 34]]
```

7. 모델링 1- 2

ResNet 모델을 바탕으로 층을 더 쌓았다.

```
# 새로운 모델 구조 설계
model10 = Sequential()
model10.add(resNet)            # VGG16 모델
model10.add(Flatten())  # Flatten 층 추가
model10.add(Dense(256, activation='swish'))  # Dense 층 추가
model10.add(Dropout(0.3))  # Dropout 층 추가 (과적합 방지)
model10.add(BatchNormalization())  # BatchNormalization 층 추가
model10.add(Dense(128, activation='swish'))  # Dense 층 추가
model10.add(Dropout(0.3))  # Dropout 층 추가 (과적합 방지)
model10.add(BatchNormalization())  # BatchNormalization 층 추가
model10.add(Dense(64, activation='swish'))  # Dense 층 추가
model10.add(Dropout(0.3))  # Dropout 층 추가 (과적합 방지)
model10.add(BatchNormalization())  # BatchNormalization 층 추가
model10.add(Dense(1, activation='sigmoid'))  # 출력층 추가 (이진 분류를 위해 sigmoid 활성화 함수 사용)

# 모델 컴파일
model10.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

7. 모델링 1 - 2

VRR16 모델을 사용했을 때보다 정확도가 높아져 90대 중반에 도달했음을 알 수 있다.

```
history = model10.fit(train_X,
                      train_df['normal_OR_abnormal'],
                      epochs=30,
                      batch_size=32,
                      validation_data=(val_X, val_df['normal_OR_abnormal']),
                      callbacks=[early_stopping])

# 훈련된 모델로 예측
y_pred = model10.predict(test_X)  # sigmoid()를 통해 각각은 0 ~ 1 값의 범위를 가질 것이구요

# 예측값 0으로 확정지을지, 1로 확정지을지 판단
y_pred_binary = (y_pred > 0.5).astype(int) # 우리가 기준을 둬야 합니다. 예를 들어 0.5 초과이면 1, 아니면 0 이런식으로 결정을 지어야 합니다..
```

```
Accuracy: 0.95
Precision: 0.9722222222222222
Recall: 0.9459459459459459
F1 Score: 0.9589041095890412
Confusion Matrix:
[[22  1]
 [ 2 35]]
```

7. 모델링 1- 2

Inception V3 모델을 바탕으로 층을 더 쌓았다.

```
# 새로운 모델 구조 설계
model11 = Sequential()
model11.add(inceptionV3)            # VGG16 모델
model11.add(Flatten())  # Flatten 층 추가
model11.add(Dense(256, activation='swish'))  # Dense 층 추가
model11.add(Dropout(0.3))  # Dropout 층 추가 (과적합 방지)
model11.add(BatchNormalization())  # BatchNormalization 층 추가
model11.add(Dense(128, activation='swish'))  # Dense 층 추가
model11.add(Dropout(0.3))  # Dropout 층 추가 (과적합 방지)
model11.add(BatchNormalization())  # BatchNormalization 층 추가
model11.add(Dense(64, activation='swish'))  # Dense 층 추가
model11.add(Dropout(0.3))  # Dropout 층 추가 (과적합 방지)
model11.add(BatchNormalization())  # BatchNormalization 층 추가
model11.add(Dense(1, activation='sigmoid'))  # 출력층 추가 (이진 분류를 위해 sigmoid 활성화 함수 사용)

# 모델 컴파일
model11.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

7. 모델링 1- 2

VRR16, ResNet에 비교적 정확도가 낮게 나오는 것을 알 수 있다.

```
history = model11.fit(train_X,
                      train_df['normal_OR_abnormal'],
                      epochs=30,
                      batch_size=32,
                      validation_data=(val_X, val_df['normal_OR_abnormal']),
                      callbacks=[early_stopping])

# 훈련된 모델로 예측
y_pred = model11.predict(test_X)  # sigmoid()를 통해 각각은 0 ~ 1 값의 범위를 가질 것이구요

# 예측값 0으로 확정지을지, 1로 확정지을지 판단
y_pred_binary = (y_pred > 0.5).astype(int) # 우리가 기준을 둬야 합니다. 예를 들어 0.5 초과이면 1, 아니면 0 이런식으로 결정을 지어야 합니다..
```

```
Accuracy: 0.8
Precision: 1.0
Recall: 0.6756756756756757
F1 Score: 0.8064516129032258
Confusion Matrix:
[[23  0]
 [12 25]]
```

7. 모델링 1- 2

EfficientNet 모델을 바탕으로 층을 더 쌓았다.

```
[ ]  # 새로운 모델 구조 설계
     model12 = Sequential()
     model12.add(efficientNetB4)            # VGG16 모델
     model12.add(Flatten())  # Flatten 층 추가
     model12.add(Dense(256, activation='swish'))  # Dense 층 추가
     model12.add(Dropout(0.3))  # Dropout 층 추가 (과적합 방지)
     model12.add(BatchNormalization())  # BatchNormalization 층 추가
     model12.add(Dense(128, activation='swish'))  # Dense 층 추가
     model12.add(Dropout(0.3))  # Dropout 층 추가 (과적합 방지)
     model12.add(BatchNormalization())  # BatchNormalization 층 추가
     model12.add(Dense(64, activation='swish'))  # Dense 층 추가
     model12.add(Dropout(0.3))  # Dropout 층 추가 (과적합 방지)
     model12.add(BatchNormalization())  # BatchNormalization 층 추가
     model12.add(Dense(1, activation='sigmoid'))  # 출력층 추가 (이진 분류를 위해 sigmoid 활성화 함수 사용)

     # 모델 컴파일
     model12.compile(optimizer='adam',
                     loss='binary_crossentropy',
                     metrics=['accuracy'])
```

7. 모델링 1- 2

Efficient 모델을 활용하여 정확도가 90대 중반에 도달했음을 알 수 있다.

```
history = model12.fit(train_X,
                      train_df['normal_OR_abnormal'],
                      epochs=30,
                      batch_size=32,
                      validation_data=(val_X, val_df['normal_OR_abnormal']),
                      callbacks=[early_stopping])

# 훈련된 모델로 예측
y_pred = model12.predict(test_X)  # sigmoid()를 통해 각각은 0 ~ 1 값의 범위를 가질 것이구요

# 예측값 0으로 확정지을지, 1로 확정지을지 판단
y_pred_binary = (y_pred > 0.5).astype(int) # 우리가 기준을 둬야 합니다. 예를 들어 0.5 초과이면 1, 아니면 0 이런식으로 결정을 지어야 합니다.
```

```
Accuracy: 0.95
Precision: 0.9722222222222222
Recall: 0.9459459459459459
F1 Score: 0.9589041095890412
Confusion Matrix:
[[22  1]
 [ 2 35]]
```

8. 모델링 2를 위한 전처리

[모델링 2 을 위한 데이터 구조 만들기]

Train, Validation, Test 데이터셋을 만들기 위해 각각 폴더를 생성하고,

하위 폴더 normal, abnoraml을 생성한다.

- 정상 이미지 저장소
  - project/Car_Images_train/normal/
  - project/Car_Images_val/normal/
  - project/Car_Images_test/normal/

- 파손 이미지 저장소
  - project/Car_Images_train/abnormal/
  - project/Car_Images_val/abnormal/
  - project/Car_Images_test/abnormal/

8. 모델링 2를 위한 전처리

정상 이미지 저장소, 파손 이미지 저장소에 이미지를 어떻게 저장시켰는지에 대해서 알아본다.
normal에 따라 Train과 Val 그리고 Test로 나눈다.
Abnormal에 따라 Train과 Val 그리고 Test로 나눈다.

```
# 이미지를 가져온다.
normal_car_images = get_image_list(f'{path}/normal/')
abnormal_car_images = get_image_list(f'{path}/abnormal/')
```

```
[ ] # 데이터 스플릿 (train: 80%, validation: 10%, test: 10%)
    train_normal, test_normal = train_test_split(normal_car_images, test_size=0.2, shuffle=True, random_state=61)
    val_normal, test_normal = train_test_split(test_normal, test_size=0.5, shuffle=True, random_state=12)

    train_abnormal, test_abnormal = train_test_split(abnormal_car_images, test_size=0.2, shuffle=True, random_state=24)
    val_abnormal, test_abnormal = train_test_split(test_abnormal, test_size=0.5, shuffle=True, random_state=35)
```

8. 모델링 2를 위한 전처리

정상 이미지 저장소, 파손 이미지 저장소에 이미지를 어떻게 저장시켰는지에 대해서 알아본다.
해당 폴더로 이동시킨다.

```
[ ]  # 이미지 파일을 복사하는 함수
     def copy_images(image_list, src_dir, dest_dir):
         for image in image_list:
             shutil.copy(os.path.join(src_dir, image), os.path.join(dest_dir, image))
```

```
# 이미지 파일 복사
copy_images(train_normal, f'{path}/normal', f'{path}/Car_Images_train/normal')
copy_images(val_normal, f'{path}/normal', f'{path}/Car_Images_val/normal')
copy_images(test_normal, f'{path}/normal', f'{path}/Car_Images_test/normal')

copy_images(train_abnormal, f'{path}/abnormal', f'{path}/Car_Images_train/abnormal')
copy_images(val_abnormal, f'{path}/abnormal', f'{path}/Car_Images_val/abnormal')
copy_images(test_abnormal, f'{path}/abnormal', f'{path}/Car_Images_test/abnormal')
```

8. 모델링 2를 위한 전처리

학습용, 검증용 데이터는 ImageGenerator()를 활용할 예정이기 때문에
테스트 데이터만 다음과 같은 작업을 한다.

```
[ ]  # 폴더 내의 모든 파일과 폴더 이름을 리스트로 가져오기
     test_normal_files = os.listdir(f'{path}/Car_Images_test/normal')
     test_abnormal_files = os.listdir(f'{path}/Car_Images_test/abnormal')
```

```
[ ]  test_files = test_normal_files + test_abnormal_files # test 용도 normal, abnormal 이미지들 합침
```

8. 모델링 2를 위한 전처리

학습용, 검증용 데이터는 ImageGenerator()를 활용할 예정이기 때문에
테스트 데이터만 다음과 같은 작업을 한다.

```
test_files[0:31:1] # 여기가 정상 차량 이미지들

['DALLíñE 2023-03-11 14.36.16 - photo of part of a car.png',
 'DALLíñE 2023-03-11 01.20.44 - a part of a car.png',
 'DALLíñE 2023-03-11 14.15.38 - part of a car.png',
 'DALLíñE 2023-03-10 23.43.01 - photo of a part of car without blemish.png',
 'DALLíñE 2023-03-11 01.04.26 - photo of a part of car.png',
 'DALLíñE 2023-03-11 01.03.58 - photo of a part of car.png',
 'DALLíñE 2023-03-10 22.23.20 - photo of a part of clean car.png',
 'DALLíñE 2023-03-11 01.06.36 - photo of a part of car.png',
 'DALLíñE 2023-03-11 00.52.55 - photo of a part of car.png',
 'DALLíñE 2023-03-11 14.28.22 - part of a car.png',
 'DALLíñE 2023-03-11 00.32.49 - photo of a part of car without blemish.png',
```

```
test_files[31::1] # 여기가 파손 차량 이미지들

['DALLíñE 2023-03-11 17.27.04 - slightly dented car.png',
 'DALLíñE 2023-03-11 18.45.56 - scratched car.png',
 'DALLíñE 2023-03-11 01.25.20 - slightly scratched car.png',
 'DALLíñE 2023-03-11 15.02.00 - dents of a car.png',
 'DALLíñE 2023-03-11 18.44.13 - slightly dented car.png',
 'DALLíñE 2023-03-11 18.45.52 - scratched car.png',
 'DALLíñE 2023-03-11 17.26.17 - slightly dented car.png',
 'DALLíñE 2023-03-11 18.43.03 - slightly dented car.png',
 'DALLíñE 2023-03-11 18.44.56 - slightly dented car.png',
 'DALLíñE 2023-03-11 17.18.00 - scratched car.png',
 'DALLíñE 2023-03-11 18.42.57 - slightly dented car.png',
```

```
[ ]  test_y = [0] * 31 + [1] * 31 # test_files에 매핑되는 0, 1 결정
```

8. 모델링 2를 위한 전처리

학습용, 검증용 데이터는 ImageGenerator()를 활용할 예정이기 때문에
테스트 데이터만 다음과 같은 작업을 한다.
테스트 데이터에 대한 Image를 모델 입력에 적합한 np.array 형태로 변환한다.

```python
# 예시: 데이터프레임 내의 이미지 파일을 NumPy 배열로 변환하는 방법
import os
from keras.preprocessing.image import load_img, img_to_array
import numpy as np

# 각 이미지를 NumPy 배열로 변환
image_arrays = []
for tf, ty in zip(test_files, test_y):
    if ty == 0: # 정상 차량 이미지인 경우
        img = load_img(f'{path}/Car_Images_test/normal/{tf}', target_size=(280, 280)) # target_size를 (280, 280)만 줘도 
    else: # 파손 차량 이미지인 경우
        img = load_img(f'{path}/Car_Images_test/abnormal/{tf}', target_size=(280, 280)) # target_size를 (280, 280)만 줘도

    img_array = img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    image_arrays.append(img_array)

# 모든 이미지 배열을 하나의 배열로 결합
test_files_vector = np.concatenate(image_arrays, axis=0)
```

8. 모델링 2를 위한 전처리

학습용, 검증용 데이터에 대해서 ImageGenerator()를 활용하여 Image Augmentation(이미지 증강) 한다.

```
datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

ImageGenerator()에 어떤 매개변수를 사용하느냐에 따라
Image Augmentation 하는 것이 약이 될 수 도 있고 독이 될 수 도 있다.
즉 이미지의 성격을 잘 파악해야 어떤 매개변수를 줄 것이며,
결과적으로 Image Augmentation 하는 것이 약이 될 수 있다.

8. 모델링 2를 위한 전처리

학습용 데이터에 대해서 Image Augmentation(이미지 증강) 한다.
알아서 0과 1 라벨링을 해주고, 이미지를 np.array 형태로 잘 변환해준다.

```
# 트레이닝 데이터용 generator 생성
train_generator = datagen.flow_from_directory(
    f'{path}/Car_Images_train',  # 트레이닝 이미지가 있는 폴더의 경로(Car_Images_train 까지만 준다.)
    target_size=(280, 280),       # 모든 이미지를 280 * 280크기로 리사이즈
    batch_size=64,                   # 배치 사이즈
    class_mode='binary',             # binary_crossentropy 손실을 사용하는 경우, 'binary'를 반환
    classes=['normal', 'abnormal'],    # 클래스 순서를 정의(normal를 0으로, abnormal를 1로 정의
    # save_to_dir=f'{path}/augmented_images',  # 증강된 이미지를 저장할 경로
    # save_prefix='aug'
)
```

8. 모델링 2를 위한 전처리

검증용 데이터에 대해서 Image Augmentation(이미지 증강) 한다.
알아서 0과 1 라벨링을 해주고, 이미지를 np.array 형태로 잘 변환해준다.

```
# 검증 데이터용 generator 생성 (일반적으로는 데이터 증강을 적용하지 않음)
validation_generator = datagen.flow_from_directory(
    f'{path}/Car_Images_val', # 검증 이미지가 있는 폴더의 경로(Car_Images_val 까지만 준다.)
    target_size=(280, 280),       # 모든 이미지를 280 * 280크기로 리사이즈
    batch_size=16,                    # 배치 사이즈
    class_mode='binary',              # binary_crossentropy 손실을 사용하는 경우, 'binary'를 반환
    classes=['normal', 'abnormal'],   # 클래스 순서를 정의(normal를 0으로, abnormal를 1로 정의
)
```

9. 모델링 2- 1

- CNN 모델링 했던 1개 모델을 차용하여 정확도를 확인한다.

9. 모델링 2- 1

```
clear_session()

# 모델 초기화
model4 = Sequential()

# Convolutional 레이어 추가 -> 256, 128, 64, 32 , BatchNormalization 활용
model4.add(Conv2D(64, (3, 3), activation='swish', input_shape=(280, 280, 3)))
model4.add(BatchNormalization())
model4.add(MaxPooling2D((2, 2)))
model4.add(Dropout(0.3))

model4.add(Conv2D(32, (3, 3), activation='swish'))
model4.add(BatchNormalization())
model4.add(MaxPooling2D((2, 2)))
model4.add(Dropout(0.3))

model4.add(Conv2D(8, (3, 3), activation='swish'))
model4.add(BatchNormalization())
model4.add(MaxPooling2D((2, 2)))
model4.add(Dropout(0.3))

# model4.add(Conv2D(32, (3, 3), activation='relu'))
# model4.add(BatchNormalization())
# model4.add(MaxPooling2D((2, 2)))
# model4.add(Dropout(0.3))

# Fully Connected 레이어 추가 -> 층을 안쌓아봤습니다.
model4.add(Flatten())
model4.add(Dense(1, activation='sigmoid'))  # Binary classification

# 모델 컴파일
model4.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Early Stopping 정의
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
```

9. 모델링 2- 1

모델을 학습하면서 Image Augmentation 한 것을 인자로 주며,
테스트 데이터를 이용해 정확도를 확인하니,
Image Augmentation 하지 않은 것보다 정확도가 비교적 떨어졌다.

```
# 모델 훈련
history = model4.fit(
    train_generator,
    steps_per_epoch=total_train_images // 64, # (Car_Images_train 폴더에서 Image Augmentation 했을 때 batch_size를 32로 줘서 3
    epochs=10,
    validation_data=validation_generator,
    validation_steps=total_validation_images // 16, # (Car_Images_val 폴더에서 Image Augmentation 했을 때 batch_size를 32로 주
)

# 훈련된 모델로 예측
y_pred = model4.predict(test_files_vector)  # sigmoid()를 통해 각각은 0 ~ 1 값의 범위를 가질 것이구요

# 예측값 0으로 확정지을지, 1로 확정지을지 판단
y_pred_binary = (y_pred > 0.5).astype(int) # 우리가 기준을 둬야 합니다. 예를 들어 0.5 초과이면 1, 아니면 0 이런식으로 결정을 지어야 합니다..
```

```
Accuracy: 0.6129032258064516
Precision: 0.6842105263157895
Recall: 0.41935483870967744
F1 Score: 0.5200000000000001
Confusion Matrix:
[[25  6]
 [18 13]]
```

10. 모델링 2- 2

- 모델링 2용 전처리했던 것을 사전 학습된 모델(VRR16, ResNet, Inception V3, EfficientNet)을 활용한다.

10. 모델링 2- 2

VRR16 모델을 활용했다.

```
# 새로운 모델 구조 설계
model5 = Sequential()
model5.add(vgg16)            # VGG16 모델
model5.add(Flatten())  # Flatten 층 추가
model5.add(Dense(256, activation='swish'))  # Dense 층 추가
model5.add(Dropout(0.3))  # Dropout 층 추가 (과적합 방지)
model5.add(BatchNormalization())  # BatchNormalization 층 추가
model5.add(Dense(128, activation='swish'))  # Dense 층 추가
model5.add(Dropout(0.3))  # Dropout 층 추가 (과적합 방지)
model5.add(BatchNormalization())  # BatchNormalization 층 추가
model5.add(Dense(64, activation='swish'))  # Dense 층 추가
model5.add(Dropout(0.3))  # Dropout 층 추가 (과적합 방지)
model5.add(BatchNormalization())  # BatchNormalization 층 추가
model5.add(Dense(1, activation='sigmoid'))  # 출력층 추가 (이진 분류를 위해 sigmoid 활성화 함수 사용)

# 모델 컴파일
model5.compile(optimizer='adam',
               loss='binary_crossentropy',
               metrics=['accuracy'])
```

10. 모델링 2 - 2

VRR16 모델을 이용하여 정확도 약 94도 정도 나왔으나,
Image Augmentation을 어떻게 했느냐에 따라 정확도가 더욱 높을 여지도 있고,
뚝 떨어질 가능성도 존재하는 것 같다.

```
history = model5.fit(train_generator,
                     steps_per_epoch=total_train_images // 64, # (Car_Images_train 폴더에서 Image Augmentation 했을
                     epochs=50,
                     validation_data=validation_generator,
                     validation_steps=total_validation_images // 16, # (Car_Images_val 폴더에서 Image Augmentation
                     callbacks=[early_stopping])

# 훈련된 모델로 예측
y_pred = model5.predict(test_files_vector)  # sigmoid()를 통해 각각은 0 ~ 1 값의 범위를 가질 것이구요

# 예측값 0으로 확정지을지, 1로 확정지을지 판단
y_pred_binary = (y_pred > 0.5).astype(int) # 우리가 기준을 둬야 합니다. 예를 들어 0.5 초과이면 1, 아니면 0 이런식으로 결정을 지어
```

```
Accuracy: 0.9354838709677419
Precision: 0.9090909090909091
Recall: 0.967741935483871
F1 Score: 0.9374999999999999
Confusion Matrix:
[[28  3]
 [ 1 30]]
```

10. 모델링 2- 2

ResNet 모델을 활용했다.

```
# Sequential 모델 생성
model6 = Sequential()

# 레이어 추가
model6.add(resNet)
model6.add(Flatten())  # Flatten 층 추가
model6.add(Dense(512, activation='swish'))
model6.add(BatchNormalization())
model6.add(Dropout(0.3))
model6.add(Dense(256, activation='swish'))
model6.add(BatchNormalization())
model6.add(Dropout(0.3))
model6.add(Dense(128, activation='swish'))
model6.add(BatchNormalization())
model6.add(Dropout(0.3))
model6.add(Dense(1, activation='sigmoid'))  # 출력층 추가 (이진 분류를 위해 sigmoid 활성화 함수 사용)

# 모델 컴파일
model6.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])
```

10. 모델링 2- 2

ResNet모델을 이용하여 정확도 약 95도 정도 나왔으나,
Image Augmentation을 어떻게 했느냐에 따라 정확도가 더욱 높을 여지도 있고,
뚝 떨어질 가능성도 존재하는 것 같다.

```
history = model6.fit(train_generator,
                     steps_per_epoch=total_train_images // 64, # (Car_Images_train 폴더에서 Image Augmentation 했을 때 batch
                     epochs=50,
                     validation_data=validation_generator,
                     validation_steps=total_validation_images // 16, # (Car_Images_val 폴더에서 Image Augmentation 했을 때 b
                     callbacks=[early_stopping])

# 훈련된 모델로 예측
y_pred = model6.predict(test_files_vector)  # sigmoid()를 통해 각각은 0 - 1 값의 범위를 가질 것이구요

# 예측값 0으로 확정지을지, 1로 확정지을지 판단
y_pred_binary = (y_pred > 0.5).astype(int) # 우리가 기준을 둬야 합니다. 예를 들어 0.5 초과이면 1, 아니면 0 이런식으로 결정을 지어야 합니다..
```

```
Accuracy: 0.9516129032258065
Precision: 0.9375
Recall: 0.967741935483871
F1 Score: 0.9523809523809523
Confusion Matrix:
[[29  2]
 [ 1 30]]
```

10. 모델링 2- 2

Inception V3 모델을 활용했다.

```
# 새 모델 정의
model7 = Sequential()

# InceptionV3 모델 추가
model7.add(inceptionV3)

# 추가 층들 정의
model7.add(GlobalAveragePooling2D())
model7.add(Dense(256, activation='swish'))
model7.add(BatchNormalization())
model7.add(Dropout(0.3))
model7.add(Dense(64, activation='swish'))
model7.add(BatchNormalization())
model7.add(Dropout(0.3))
model7.add(Dense(1, activation='sigmoid'))  # 이진 분류를 위한 출력층

# 모델 컴파일
model7.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

10. 모델링 2- 2

Inception V3 모델을 이용하여 정확도 약 76 정도 나왔으나,
Image Augmentation을 어떻게 했느냐에 따라 정확도가 더욱 높을 여지도 있고,
뚝 떨어질 가능성도 존재하는 것 같다.

```
history = model7.fit(train_generator,
                     steps_per_epoch=total_train_images // 64, # (Car_Images_train 폴더에서 Image Augmentation 했을 때 batch
                     epochs=50,
                     validation_data=validation_generator,
                     validation_steps=total_validation_images // 16, # (Car_Images_val 폴더에서 Image Augmentation 했을 때 b
                     callbacks=[early_stopping])

# 훈련된 모델로 예측
y_pred = model7.predict(test_files_vector)  # sigmoid()를 통해 각각은 0 ~ 1 값의 범위를 가질 것이구요

# 예측값 0으로 확정지을지, 1로 확정지을지 판단
y_pred_binary = (y_pred > 0.5).astype(int) # 우리가 기준을 둬야 합니다. 예를 들어 0.5 초과이면 1, 아니면 0 이런식으로 결정을 지어야 합니다..
```

```
Accuracy: 0.7580645161290323
Precision: 1.0
Recall: 0.5161290322580645
F1 Score: 0.6808510638297872
Confusion Matrix:
[[31  0]
 [15 16]]
```

10. 모델링 2- 2

Efficient 모델을 활용했다.

```python
# Efficient 모델 불러오기
efficientNetB4 = EfficientNetB4(weights='imagenet', include_top=False, input_shape=(280, 280, 3))
efficientNetB4.trainable = False  # 모델의 모든 레이어를 고정

# 새 모델 정의
model8 = Sequential()

# Efficient 모델 추가
model8.add(efficientNetB4)

# 추가 층들 정의
model8.add(GlobalAveragePooling2D())
model8.add(Dense(1, activation='sigmoid'))  # 이진 분류를 위한 출력층

# 모델 컴파일
model8.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

10. 모델링 2- 2

Efficient 모델을 이용하여 정확도 약 94 정도 나왔으나,
Image Augmentation을 어떻게 했느냐에 따라 정확도가 더욱 높을 여지도 있고,
뚝 떨어질 가능성도 존재하는 것 같다.

```
history = model8.fit(train_generator,
                     steps_per_epoch=total_train_images // 64, # (Car_Images_train 폴더에서 Image Augmentation 했을 때 batc
                     epochs=50,
                     validation_data=validation_generator,
                     validation_steps=total_validation_images // 16, # (Car_Images_val 폴더에서 Image Augmentation 했을 때
                     callbacks=[early_stopping])

# 훈련된 모델로 예측
y_pred = model8.predict(test_files_vector)  # sigmoid()를 통해 각각은 0 - 1 값의 범위를 가질 것이구요

# 예측값 0으로 확정지을지, 1로 확정지을지 판단
y_pred_binary = (y_pred > 0.5).astype(int) # 우리가 기준을 둬야 합니다. 예를 들어 0.5 초과이면 1, 아니면 0 이런식으로 결정을 지어야 합니다..
```

```
Accuracy: 0.9354838709677419
Precision: 1.0
Recall: 0.8709677419354839
F1 Score: 0.9310344827586207
Confusion Matrix:
[[31  0]
 [ 4 27]]
```

11. 결론

결론

- Image Augmentation을 적용한 것으로 모델을 돌렸을 때 기대한 바와 달리
  왜 굳이 Image Augmentation 이라는 절차를 거칠까 하는 의문이 들었다.
  그냥 Image Augmetation을 수행하지 않고 사전 학습된 모델로 돌려도 정확도가 90대 초반 ~ 90대 중반까지 정확도가 잘 나왔기 때문이다.
    - Image Augmentation이 데이터 자체를 증강하지만,
      어떤 방식으로 데이터를 증강시킬 것인가 하는 충분한 고려 없이 진행한다면
      오히려 데이터 품질이 좋아지지 않고 그 결과 정확도도 많이 떨어질 수 있다는 결론이 나왔다.
      즉 어떻게 각도를 정의하고, 어떤 방식으로 이미지를 증강해야, 데이터 품질이 한층 강화될까? 에 대한 생각이 필요할 것 같다.