차량 모델 구분

14기 김인호

프로젝트 목차 예시

목차

1. 프로젝트 개요

넥스트랩에서 진행하는 공통 주제인 차량 브랜드 탐색을 진행

세부 과제로 차량의 색 구분, 방향 탐지 등을 진행하고 최종적으로 이를 종합한 브랜드 탐색을 목표로 한다.

예시 사진

1주차에 데이터 정리 및 색 구분, 2주차 부터 전체적인 브랜드 구분을 하려 했으나
데이터 전처리 자체에 어려움이 있어서 많은 시간이 소모되어 모델의 성능에 집중하지 못하고 구현자체에 의의를 두게 되었다.

02. 프로젝트 수행 절차 및 방법

구분	기간	활동
사전 준비	▶ 11/18(금) ~ 11/23(수)	▶ 데이터 저장 및 파악
데이터 정리	▶ 11/24(목) ~ 12/1(목)	▶ Labelme 데이터셋 구조, json파일 데이터 구조, yolo파일 구조 학습 및 annotation이용 관련 학습 ▶ 이미지 crop 코드
데이터 전처리	▶ 12/2(금) ~ 12/7(수)	▶ 데이터 정제 및 정규화
모델링	▶ 12/8(목) ~ 12/12(월)	**▶ 사용할 수 있는 모델 관련 기본 학습 및 case별로 적용해보기**
내용정리	▶ 12/12(월) ~ 12/13(화)	▶ 코드 정리 및 ppt준비
총 개발기간	▶ 11/18(금) ~ 12/12(월)	-

03. 프로젝트 수행 결과

결과 제시 ① 탐색적 분석 및 전처리

학습 데이터 소개 (Train dataset)

- 주어진 데이터셋은 AIHUB에서 제공하는 차량 외관 영상 데이터로 2021년까지 구축된 322,664건의 데이터이다.

차종은 총 100종으로 세부적으로 색상/연식 등에 따라 구분이 된다.

하지만 컴퓨터 성능으로 전체 데이터를 다 보기엔 컴퓨터의 성능이 좋지 않고 시간도 너무 지체 되어 다음과 같이 줄였다.

전체 데이터 셋에 대해서는 일괄적으로 CROP 진행(약 4만5천개)
-> 아우디, 벤츠, BMW, 쉐보레, 포드, 제네시스, 혼다, 현대 데이터셋으로 범위 축소
-> 그 중에서도 아우디, 벤츠, BMW, 제네시스, 현대의 데이터를 부분적으로 사용

같이 주어진 데이터셋으로는 JSON annotation파일이 있었는데 이는 labelme 형식, yolo에 사용할 수 있는 txt파일 등으로 변환하였다.

03. 프로젝트 수행 결과

결과 제시 ② 모델 개요

- **기본 CNN모델**
- **EfficientnetB0**
- **YOLOV4**

위와 같이 총 3개의 기본 모델에 대해 데이터를 분석 시도했으나
아직 모델에 대한 개념이나 활용면에서 숙지가 미숙하여 어려움을 겪어 모델의 성능을 높이는 것과 같은
부분은 생각할 수 없었고 구현 자체에 의미를 두었다.

따라서 모델을 돌린 것에 대해서도 유의미한 Loos나 acc를 파악하기는 힘들었다.

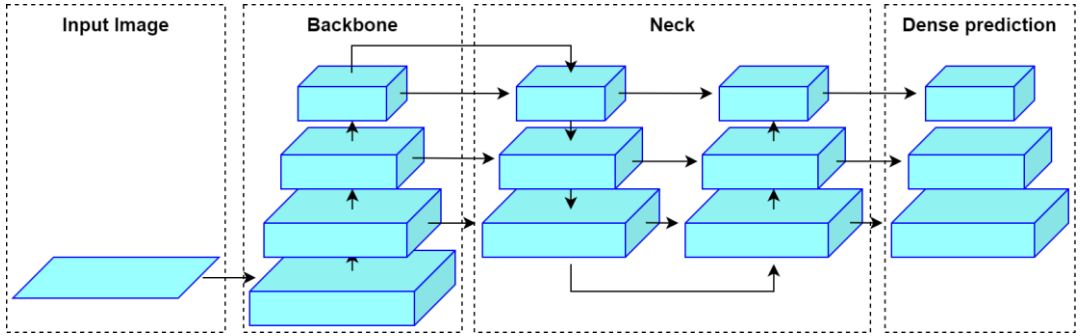
03. 프로젝트 수행 결과

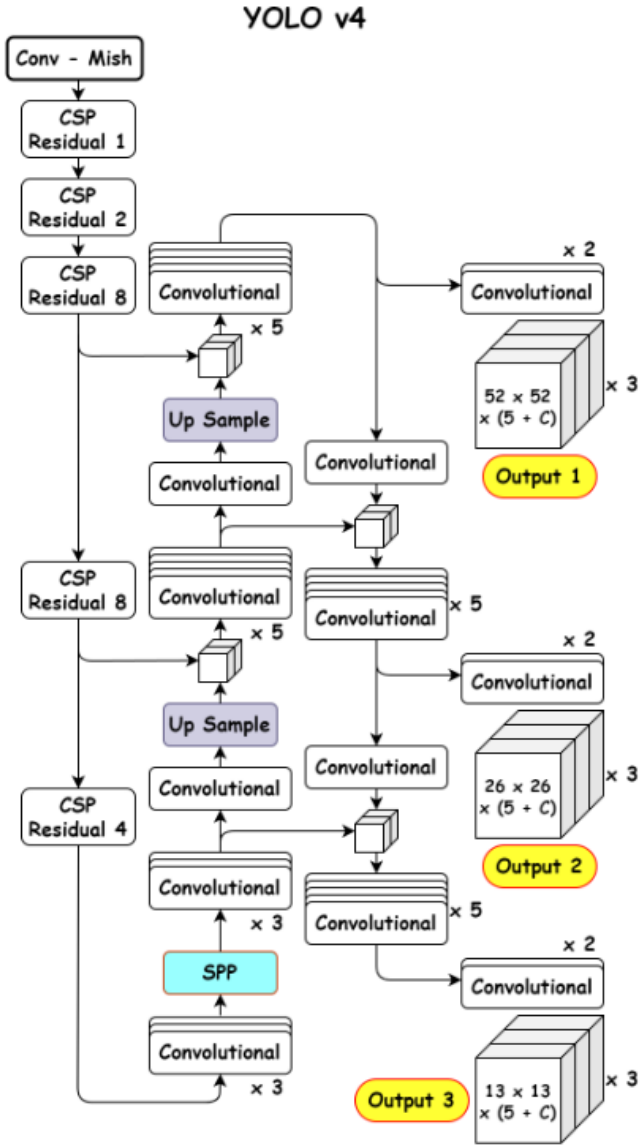
결과 제시 ② 모델 개요

YOLOV4

- One-stage 방식 의 모델로 구조를 단순화하여 빠르게 동작할 수 있도록 하는 모델
- 컨볼루션 계층 내에 작은 그룹을 적 용한 53개의 계층이 있는 DarkNet을 백본 (backbone)으로 사용하였으며 수용장을 늘리기 위 해 SPP(Spatial Pyramid Pooling)를 적용하였다.
  또한, 다양한 크기(multi-scale)의 객체를 추출하기 위 해 PANet을 사용하였고
  머리(Head) 부분에 YOLOv3를 적용하였다.
- ReLU 활성화 함수를 주로 사용한 이전 모델과 달리 YOLOv4는 연속적으로 미분 가능한 함 수인 Mish를 사용하였다.

- 객체 탐지는 객체의 위치를 찾아야 하는 지역화와 객체를 구별하는 분류작업으로 구성되어 있다.
  이를 동시에 수행하는 방 식이 One-stage 방식이고 순차적이며 거칠고 미세 한(coarse-to-fine) 방식으로 진행하는 방식이 Two -stage 방식이 있다.
- One-stage 방식은 YOLO, SSD, RetinaNet 등이 있고, Two-stage 방식은 R-CNN, Fast R-CNN, Faster R-CNN 등이 있다.





03. 프로젝트 수행 결과

결과 제시 ③ 모델 선정 및 분석

1. Yolov4 사용 / 차량 방향에 따른 차종 구분(아우디와 bmw의 crop데이터만 사용)

- 전면 : 2개의 헤드램프와 라디에이터 있는 경우
- 후면: 2개의 리어램프와 트렁크가 있는 경우
- 사이드: 위의 2가지 조건 만족하지 않는 경우

위와 같이 데이셋을 나눈후 yolov4에 적용가능한 txt 파일을 만듦

* 아래와 같이 클래스를 설정

```
component_list = [0 for i in range(len(annotations))]
for i in range(len(annotations)):
  component_list[i] = annotations[i]['classId']


class_data = []
for n in range(len(annotations)):


  class_data.append(int(re.sub(r'[^0-9]', ' ', component_list[n]).lstrip('0')))

if 0 in class_data:
    car_class = {"아우디_A4_2017":1,"아우디_A4_2018":2,"아우디_A6_2017":3,"아우디_A6_2018":4,"아우디_A6_2019":5,
                 "아우디_A6_2020":6,"아우디_Q5_2017":7,"아우디_Q5_2018":8,"아우디_Q5_2020":9,
                 "아우디_Q7_2019":10,"아우디_Q7_2021":11,"아우디_A7_2017":12,"BMW_3시리즈_2017":13,"BMW_3시리즈_2018":14,
                 "BMW_3시리즈_2019":15,"BMW_3시리즈_2020":16,"BMW_5시리즈_2017":17,"BMW_5시리즈_2018":18,"BMW_5시리즈_2019":19,"BMW_5시리즈_2020":20,
                 "BMW_5시리즈_2021":21,"BMW_7시리즈_2017":22,"BMW_7시리즈_2018":23,"BMW_7시리즈_2019":24,"BMW_7시리즈_2020":25,
                 "BMW_X3_2017":26,"BMW_X3_2018":27,"BMW_X3_2019":28,"BMW_X3_2020":29,
                 "BMW_X5_2017":30, "BMW_X5_2018":31,"BMW_X5_2019":32,"BMW_X5_2020":33,"BMW_X5_2021":34,
                },get(json_data['rawDataInfo']['MediumCategoryId']+'_'+json_data['rawDataInfo']['SmallCategoryId']+'_'+str(json_data['rawDataInfo']['yearId']))
```

03. 프로젝트 수행 결과

결과 제시 ③ 모델 선정 및 분석

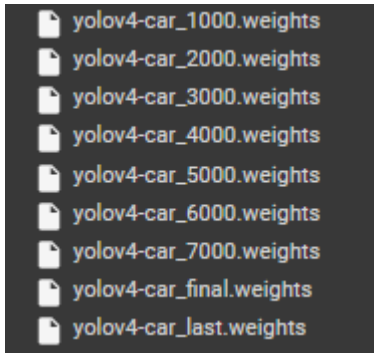
1. Yolov4 사용 / 차량 방향에 따른 차종 구분(아우디와 bmw의 crop데이터만 사용)

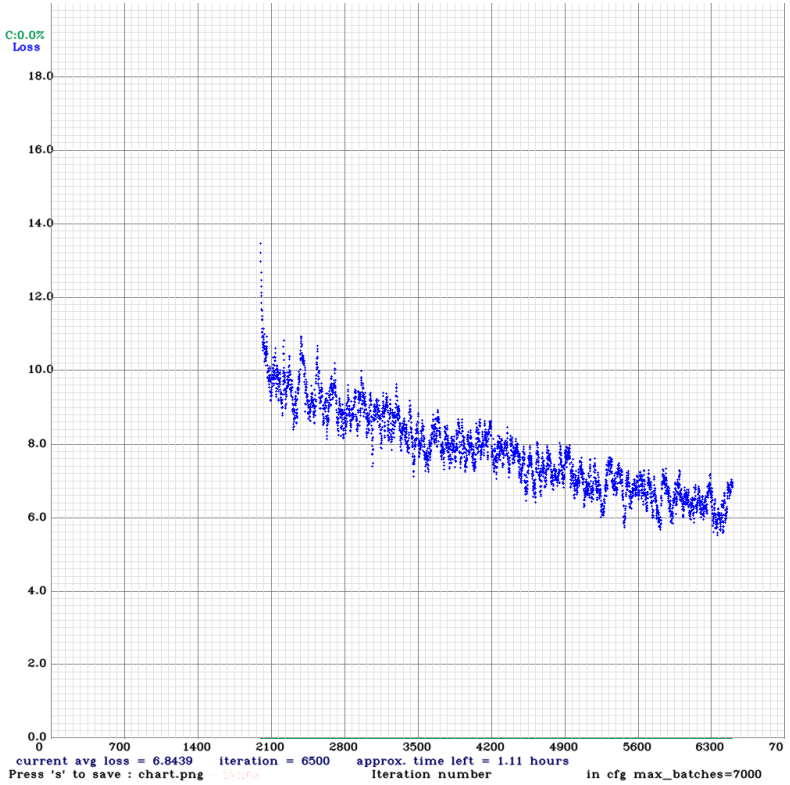
각각의 방향에 따른 모델을 만든 뒤 훈련을 하려 했으나 1000개의 데이터를 통해 전방을 훈련시키는데도 12시간이 걸림
따라서 일단 전방만 진행함

각각의 헤드램프2개, 라디에이터, 차량전체
-> 총 4개의 box를 통해 차량의 특성을 파악할 수 있도록 하려함

훈련을 시켜보았으나 유의미한 loss 감소가 일어나지 않았다.

Train과 Val 데이터 셋을 이용하여 acc와 loss를 비교한 뒤 원본 사이즈의 데이터를 통해 TEST를 하려 했으나 진행에 어려움을 겪어 일단 훈련된 모델을 통해 임의로 val에서 이미지를 골라 테스트 진행함





03. 프로젝트 수행 결과

결과 제시 ③ 모델 선정 및 분석

온전한 전면 방향이 아니라 그런지 바운딩 박스가 잘 처리도 되지 않은 것 같다.


```
/content/drive/MyDrive/yolov4_2/darknet/build/darknet/x64/data/FRONT/C_211207_BM_002_18_BK_A_T_02_011.jpg:
BMW_3시리즈_2018: 90%   (left_x:  116   top_y:  -13   width:  226   height:  218)
BMW_3시리즈_2018: 81%   (left_x:  221   top_y:  232   width:   87   height:   36)
BMW_3시리즈_2018: 78%   (left_x:  318   top_y:  236   width:   80   height:   35)
```

```
Detection layer: 139 - type = 28
Detection layer: 150 - type = 28
Detection layer: 161 - type = 28
/content/drive/MyDrive/yolov4_2/darknet/build/darknet/x64/data/FRONT/C_211202_BM_002_17_BL_A_T_02_001.jpg:
BMW_3시리즈_2017: 63%
tcmalloc: large alloc 1979711488 bytes == 0x558e0f28a000 @  0x7f832f1ab001 0x558da564c590 0x558da5680298 0x
```

```
[139] im_show('predictions.jpg')
```

front_val.txt | C_211202_BM_002_17_BL_A_T_02_001.jpg | C_211202_BM_002_17_BL_A_T_02_001.txt ×

```
1 13 0.378125 0.377083 0.218750 0.400000
2 13 0.434375 0.679167 0.101562 0.050000
3 13 0.381250 0.629167 0.054688 0.070833
4 13 0.539062 0.637500 0.053125 0.062500
5
```

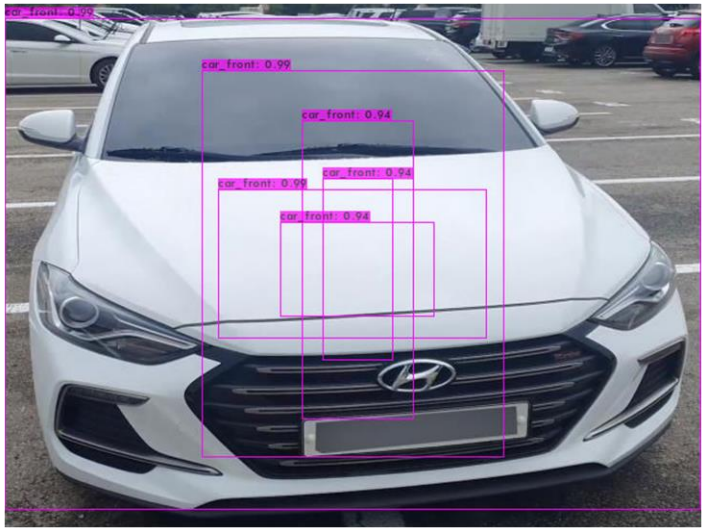
```
/content/drive/MyDrive/yolov4_2/darknet/build/darknet/x64/data/FRONT/C_211207_BM_038_18_GR_A_T_03_003.jpg:
BMW_X3_2018: 35%        (left_x:  132   top_y:   56   width:  201   height:  164)
BMW_X3_2018: 41%        (left_x:  349   top_y:  196   width:   63   height:   39)
BMW_X3_2018: 48%        (left_x:  357   top_y:   66   width:  203   height:  181)
```

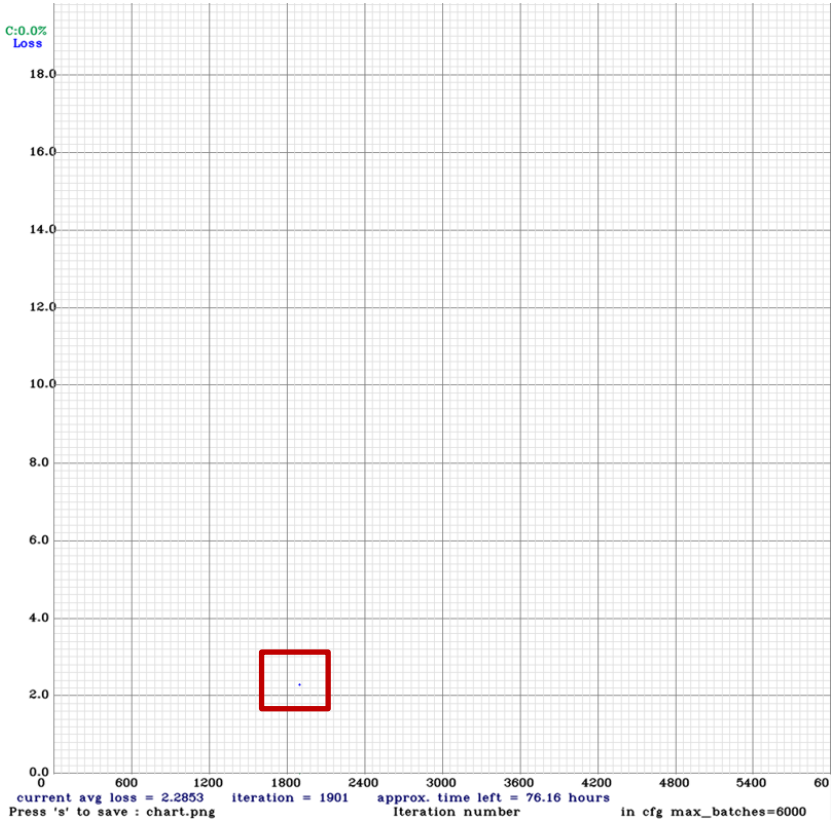
03. 프로젝트 수행 결과

결과 제시 ③ 모델 선정 및 분석

2. Yolov4 사용 / 차량 방향 구분(벤츠, BMW, 제네시스, 현대의 crop데이터만 사용, 약 4만)

훈련 시간이 부족해서 전체 batch를 못돌았더니 차량을 제대로 감지하지 못하였다.
따라서 자체적으로 모델을 통해 훈련시킨 weigh값을 사용하지 못하고
대표님께서 주신 모델의 weight를 사용했더니 다음과 같이 잘 탐지가 되었다.
전에 차량 객체 탐지 모델의 loss보다는 작게 점이 찍혀서 정상적으로 마무리 하면
나쁘지 않은 결과가 나올 것 같다.





03. 프로젝트 수행 결과

결과 제시 ③ 모델 선정 및 분석

3. 색상 구분 기본 CNN과 efficientnetB0사용 시도

아우디와 BMW의 원본이미지 + crop이미지(약4천개) 사용

총 4개의 색상: 검정/파랑/회색/흰색 이 있어서 다음과 같이 파일을 구분

Crop되지 않은 이미지는 원본인 1920*1080이고 crop된 이미지는 640*480 사이즈라 resize를 통해 640*480으로 통일 시킴

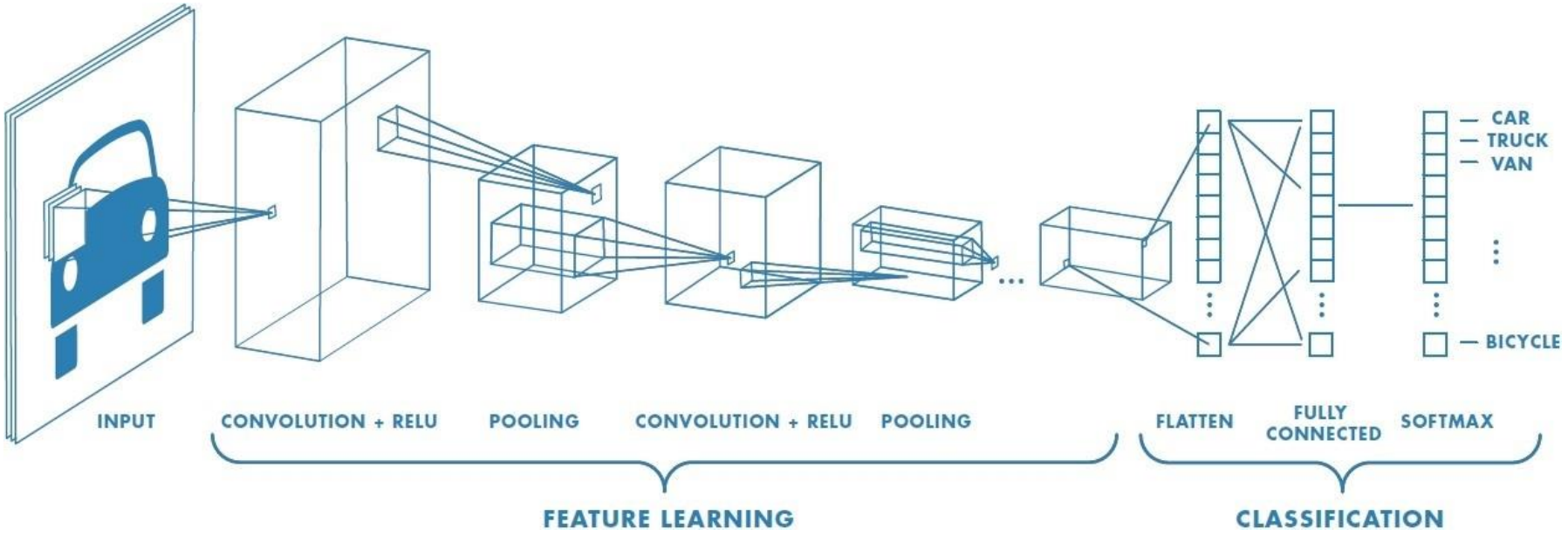

```
{'bk': 0, 'bl': 1, 'gr': 2, 'wh': 3}
```

03. 프로젝트 수행 결과

결과 제시 ② 모델 개요

CNN

- 이미지에서 행렬 곱 연산 대신 컨볼루션(convolution) 연산을 통해 특징을 추출하는 컨볼루션 계층과 특징맵의 크기를 줄이는 풀링(pooling) 계층으로 구성되어 있다. 특히 풀링 계층은 이미지의 크기를 줄이면서 특징을 그대로 유지하는 장점이 있지만, 원본 이미지의 크기가 줄어들어 위치 정보를 잃는 다는 단점이 존재한다. 반복 그리고 후반에 다층 퍼셉트론 층으로 이루어진다.



03. 프로젝트 수행 결과

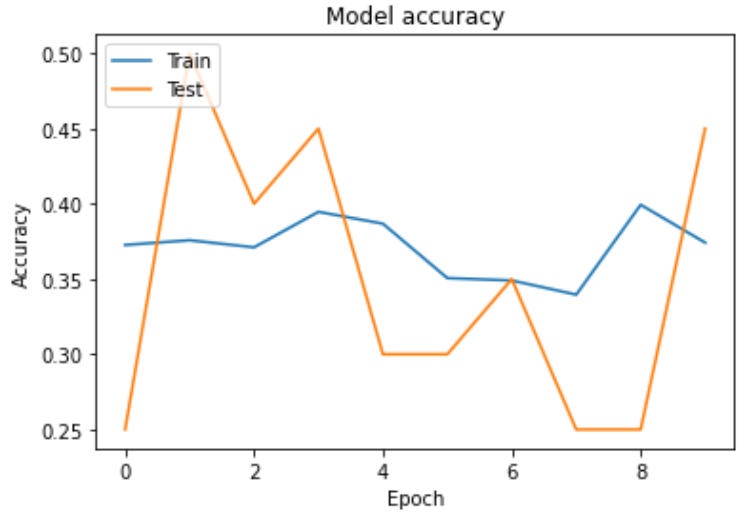
결과 제시 ③ 모델 선정 및 분석

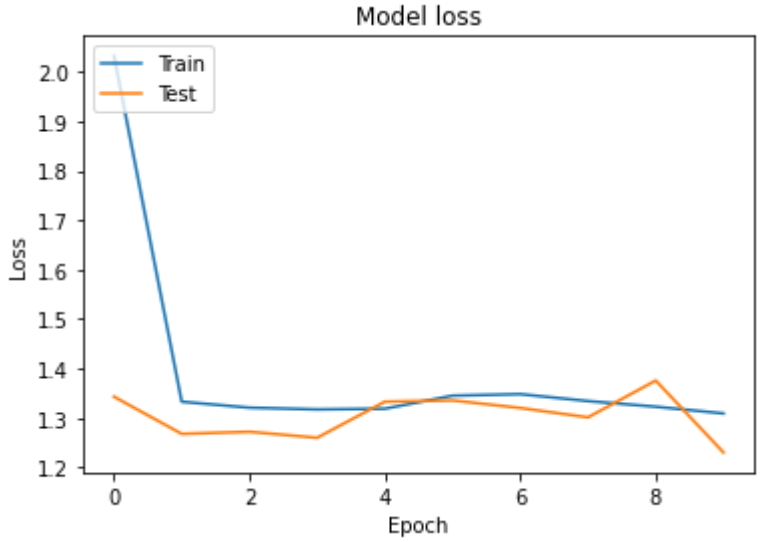
3. 색상 구분 기본 cnn과 efficientnetB0사용 시도

(시간 관계상 적은 epoch밖에 시도하지 못했다.)

cnn

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 480, 640, 32)      896

 conv2d_1 (Conv2D)           (None, 478, 638, 32)      9248

 max_pooling2d (MaxPooling2D  (None, 239, 319, 32)     0
 )

 dropout (Dropout)           (None, 239, 319, 32)      0

 conv2d_2 (Conv2D)           (None, 239, 319, 64)      18496

 conv2d_3 (Conv2D)           (None, 237, 317, 64)      36928

 max_pooling2d_1 (MaxPooling  (None, 118, 158, 64)     0
 2D)

 dropout_1 (Dropout)         (None, 118, 158, 64)      0

 flatten (Flatten)           (None, 1193216)           0

 dense (Dense)               (None, 512)               610927104

 dropout_2 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 4)                 2052

=================================================================
Total params: 610,994,724
Trainable params: 610,994,724
Non-trainable params: 0
_________________________________________________________________
```



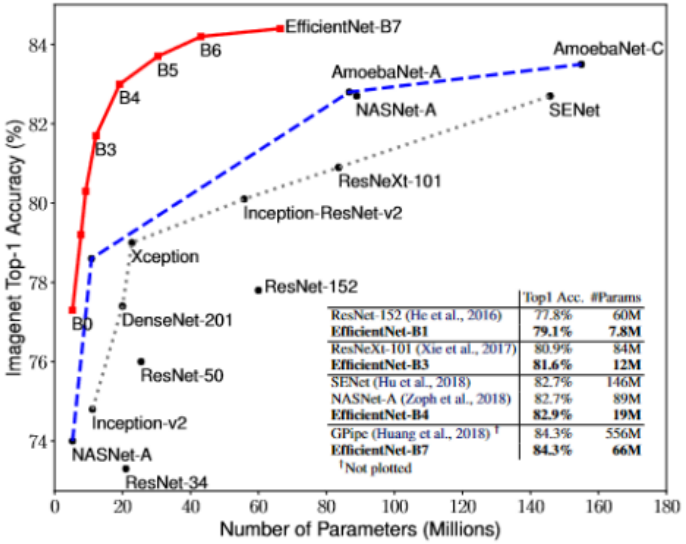


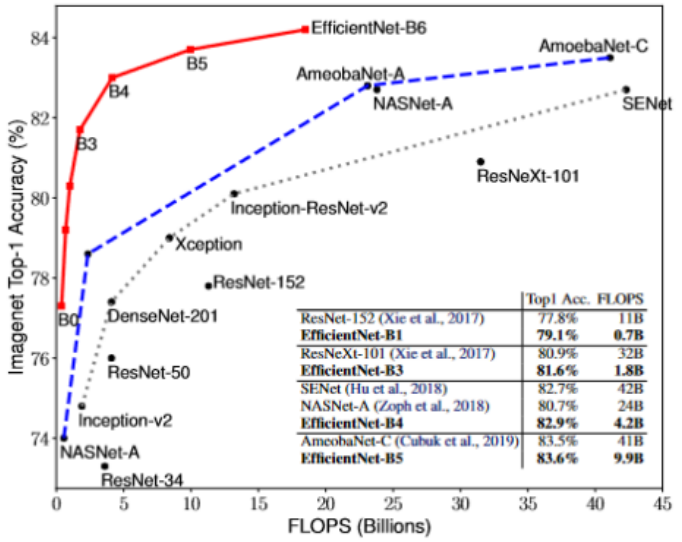
03. 프로젝트 수행 결과

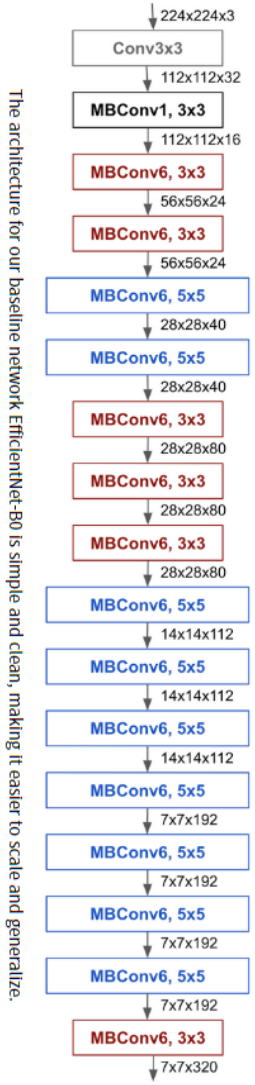
결과 제시 ② 모델 개요

EfficientnetB0

- **네트워크의 깊이(Depth), 필터 수(Width), 이미지 Resolution** 크기를 최적으로 조합하여 모델 성능 극대화하는 것
- B1~B7은 B0에서 Depth(Block의 개수)와 Width(Filter의 개수)를 증가시킨 모델
- 이는 기존 CNN의 feature map 개수, layer개수, resolution을 일정한 비율로 증가시키는 compound scaling 방법을 사용한 것으로 정확도와 효율성을 모두 높여서 매개 변수의 크기와 FLOPS(1초 동안 수행가능한 연산횟수)를 10배나 줄였다.







The architecture for our baseline network EfficientNet-B0 is simple and clean, making it easier to scale and generalize.

03. 프로젝트 수행 결과

결과 제시 ③ 모델 선정 및 분석

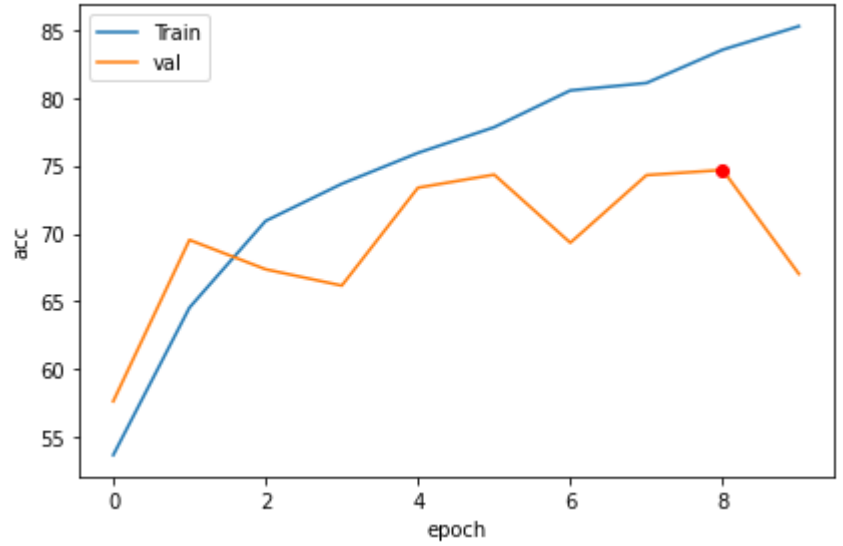
3. 색상 구분 기본 cnn과 efficientnetB0사용 시도

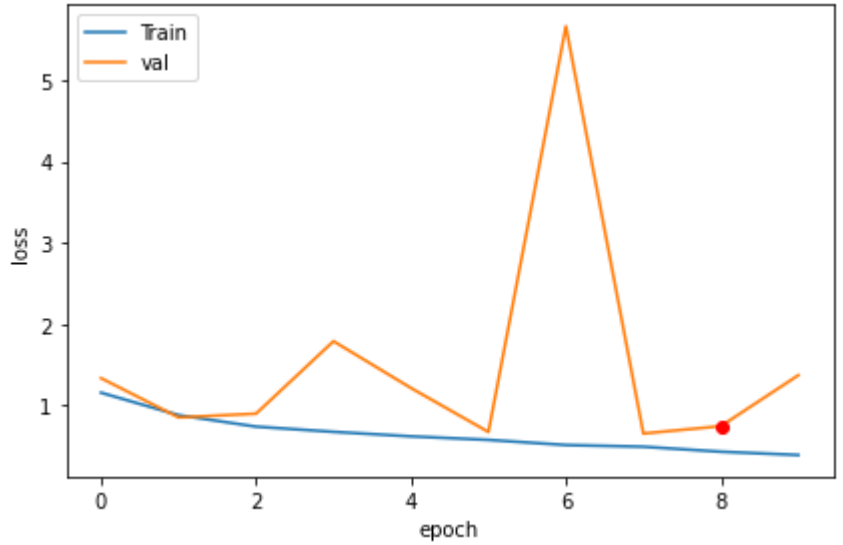
(시간 관계상 적은 epoch밖에 시도하지 못했다.)

efficientnetB0

```
### ImageFolder 작성
train_imgs = ImageFolder("/content/drive/MyDrive/car_color/class_output_resize/train",
                         transform=transforms.Compose([transforms.Resize((224, 224)),
                                                       transforms.ToTensor()]))

val_imgs = ImageFolder("/content/drive/MyDrive/car_color/class_output_resize/val",
                       transform=transforms.Compose([transforms.Resize((224, 224)),
                                                     transforms.ToTensor()]))

test_imgs = ImageFolder("/content/drive/MyDrive/car_color/class_output_resize/test",
                       transform=transforms.Compose([transforms.Resize((224, 224)),
                                                     transforms.ToTensor()]))
batch_size = 64
dataloaders, batch_num = {}, {}
dataloaders['train'] = data.DataLoader(train_imgs, batch_size, shuffle=True)
dataloaders['valid'] = data.DataLoader(val_imgs, batch_size, shuffle=True)
dataloaders['test'] = data.DataLoader(test_imgs, batch_size, shuffle=True)
batch_num['train'], batch_num['valid'], batch_num['test'] = len(dataloaders['train']), len(dataloaders['valid']), len(dataloaders['test'])

print('batch_size : %d,  tvt : %d / %d / %d' % (batch_size, batch_num['train'], batch_num['valid'], batch_num['test']))
```

```
batch_size : 64,  tvt : 160 / 46 / 23
```



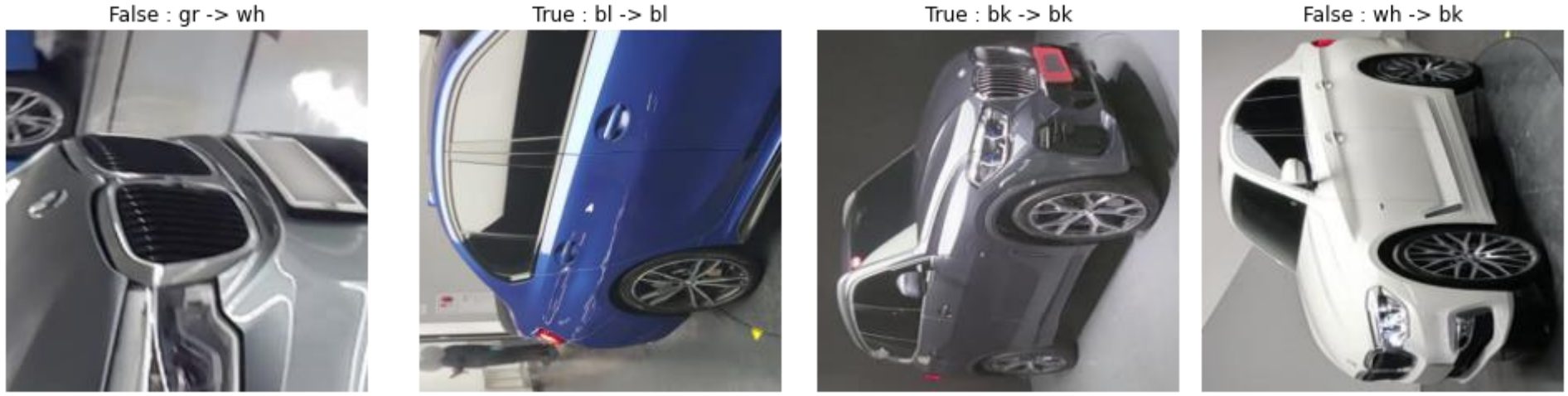


일단 총 10회의 epoch으로 돌린 결과 다음과 같은 추이를 보였으며
epoch수를 늘렸다면 조금 더 유의미한 수치가 나왔을 것 같다.

03. 프로젝트 수행 결과

결과 제시 ③ 모델 선정 및 분석

3. 색상 구분 기본 cnn과 efficientnetB0사용 시도



test: loss/acc : 0.73 / 75.5

1460개의 Test이미지를 사용해본 결과 75% 정도의 정확도를 보였다.

03. 자체 평가 의견

- 전체적으로 데이터 셋이 너무 크다 보니 데이터를 가져오는 처음부터 어려움을 겪었다.
그리고 json파일 구조 자체를 다루는 것에 대한 이해가 부족해 이를 여러가지 형태로 변환하는 과정에서 많은 시간이 걸렸다. 전체 json파일을 labelme에 적합한 파일로 바꾸는데 3일 넘게 걸리고 yolo모델에 적용하기 위해 txt로 변환시키는 과정에서도 많은 시간을 소요했다.
Class를 설정하는 부분에서도 이미지의 특징이 되는 부분이 아닌 전체 box에 대해 시도하는 등 시행착오를 많이 겪었다.

추가로 crop하는 코드를 구현하는데도 어려움이 있어서 모델을 구현하기 전까지의 시간이 너무 오래 걸려 모델에 대한 이해를 하고 실질적으로 모델을 돌려본 시간이 매우 적었다.
따라서 마지막주에는 모델의 성능에 관계없이 detection을 했는지 못했는지에 집중하게 되어 성능적인 부분을 전혀 고려하지 못한 부분이 아쉽다.
아마 돌려본 모델들의 성능 부분에 있어서는 resolution값이나 batch size설정에 있어서 문제가 있지 않았나 싶다.

처음부터 전체 데이터를 사용할 생각을 하지 말고 약 1000개 내외의 데이터만 뽑아내어 다양한 모델을 학습 및 구현해보고 점차 그 양을 확대해봤으면 좀 더 의미가 있는 프로젝트가 될 것 같은 아쉬움이 있다.