



GITHUB ID
BEATKRAQ

Since 2021

CP2 최종보고서

연식별 차종 분류

With Nextlab

김현우 2022.12.13

목차

01

개요

02

프로젝트
수행절차

03

프로젝트
수행결과

04

회고





Part 1

프로젝트 개요

프로젝트 개요 <프로젝트 목표>

프로젝트 목표

- 사진이나 영상 내의 차량을 탐지하고 차량의 모델 및 연식을 분류한다.
- 이 프로젝트는 실제 산업 현장에서의 응용 가능성을 탐색하고, 지속적으로 업데이트되는 차량 데이터에 대응할 수 있는 동적 학습 전략을 마련하는 것을 목표로 한다.

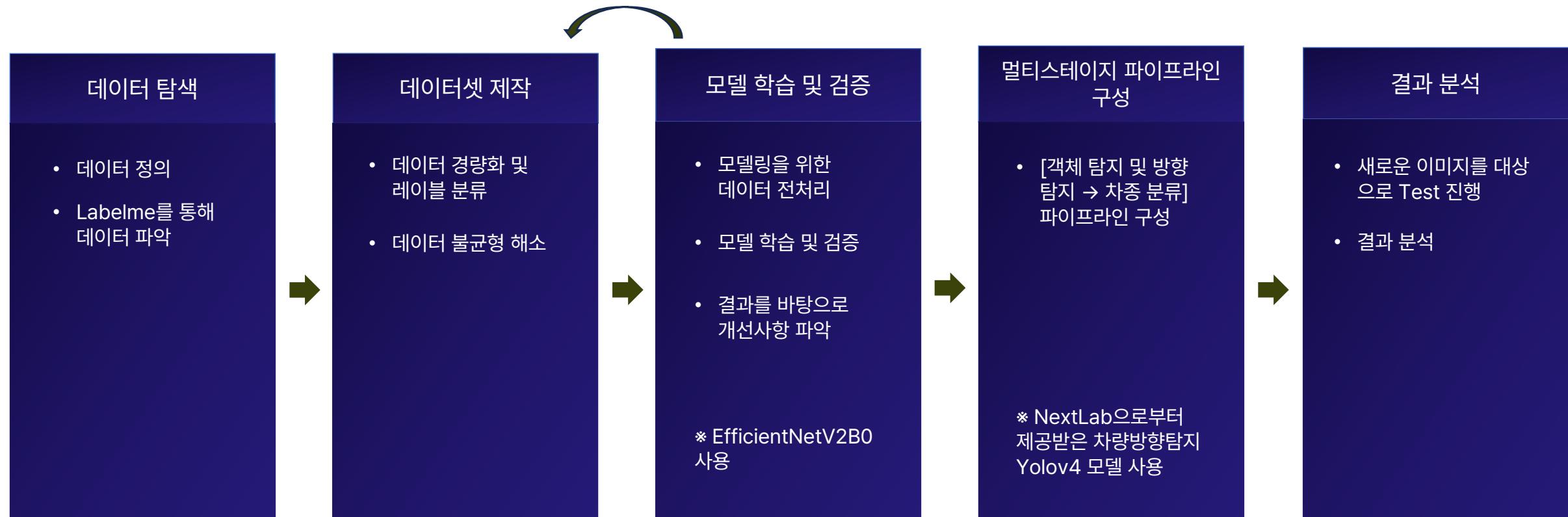


프로젝트 목적

- 실시간 객체 탐지 분야에서 가장 널리 사용되는 YOLO(You Only Look Once)의 사용법을 깊이 있게 이해한다.
- EfficientNet의 사용법을 숙지하여, 높은 성능과 효율성을 자랑하는 CNN 분류 모델을 효과적으로 활용하는 방법을 배운다.
- 멀티스테이지 파이프라인을 구성하여 YOLO와 EfficientNet을 활용, 객체 탐지(+방향인식)와 이미지 분류 작업을 통합한다.
- 이를 통해 다양한 실제 환경에서의 차량 식별과 분류 작업을 자동화하는 능력을 개발한다..



프로젝트 개요 <프로젝트 수행절차>





Part 2
프로젝트 수행절차

프로젝트 수행절차 <데이터 탐색>

데이터 정의

데이터 이름: 차량 외관 영상 데이터 (79.70GB)

출처: Aihub

개요: 차종, 연식, 색상, 트림과 14개 파트를 식별할 수 있는 학습용 데이터셋

데이터 구축 규모: 브랜드 21개, 차종 100종, 3099대 차량의 외관 학습데이터 322,644장

데이터 분포

- 전체 차량 이미지 110,285장 구축
- 차량 부분 이미지 213,569장 구축

데이터 영역	교통물류	데이터 유형	이미지
데이터 형식	JPG	데이터 출처	직접 촬영
라벨링 유형	바운딩박스(이미지)	라벨링 형식	JSON
데이터 활용 서비스	차량 동시 분석 트랙킹 시스템 활용/ 차량 외관 파손 모니터링 시스템 활용/ 차량의 차종, 컬러, 차량 외관의 파손 여부를 범죄 사용 차량 파악/ 자율주행 차량 인식 데이터셋 활용	데이터 구축년도/ 데이터 구축량	2021년/322,664건

• 종분류별 구축 수량

브랜드	BMW	기아	닛산	랜드로버	렉서스	르노	르노삼성
수집량	114	1012	7	39	11	13	194
데이터셋	11,868	105,564	732	4,069	1,158	1,365	20,384
브랜드	미니	벤츠	볼보	쉐보레	쌍용	아우디	제네시스
수집량	11	224	11	205	195	42	202
데이터셋	1,120	22,984	1,160	21,294	20,315	4,367	21,237
브랜드	지프	토요타	포드	폭스바겐	한국지엠	현대	혼다
수집량	4	10	42	36	10	710	7
데이터셋	344	1,037	4,440	3,643	1,048	73,799	736

프로젝트 수행절차 <데이터 탐색>

- 데이터 탐색 위해 labelme 프로그램 사용
- AIHUB에서 제공된 JSON파일을 labelme 형식으로 수정



labelme

이미지에 주석을 달아 객체 인식을 위한 데이터 세트를 생성할 수 있는 오픈 소스 그래픽 이미지 주석 툴(by MIT)입니다.

```

"learningDataInfo": {
  "path": null,
  "LearningDataId": "C_211222_AU_006_17_BK_A_P_01_002",
  "fileExtension": "json",
  "objects": [
    {
      "classId": "P01.프론트범퍼",
      "annotation": "bbox",
      "coords": {
        "tl": {
          "x": 185,
          "y": 327
        },
        "tr": {
          "x": 1417,
          "y": 327
        },
        "bl": {
          "x": 185,
          "y": 671
        },
        "br": {
          "x": 1417,
          "y": 671
        }
      },
      "left": 185,
      "top": 327,
      "width": 1232,
      "height": 344,
      "angle": 0
    }
  ]
}
  
```



```

{
  "label": "P01.프론트범퍼",
  "points": [
    [
      664.23,
      655.78
    ],
    [
      1160.605979987893,
      655.78
    ],
    [
      1160.605979987893,
      760.6099999999999
    ],
    [
      664.23,
      760.6099999999999
    ]
  ],
  "group_id": null,
  "shape_type": "polygon",
  "flags": {}
}
  
```

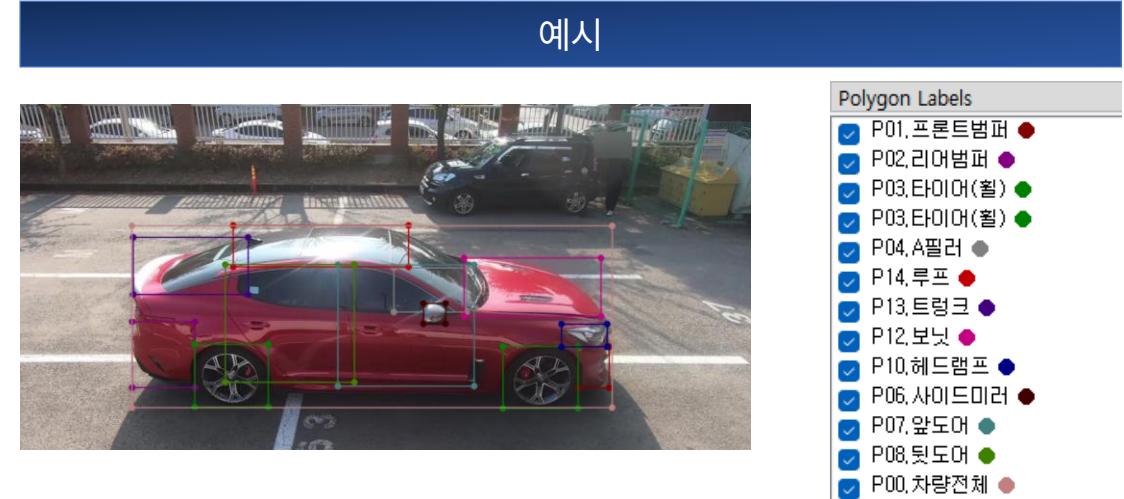
프로젝트 수행절차 <데이터 탐색>

(Continued)

- labelme로 데이터를 열어 데이터의 구조 파악
- 사진 데이터는 각각 angle, zoom별로 상이하게 촬영되었으며, 이에 따라 전체 14개의 파트 중 일부가 포함되었음
- 각 어노테이션에 대한 정보

P00: 차량전체
 P01: 프론트범퍼
 P02: 리어범퍼
 P03: 타이어(휠)
 P04: A필러
 P05: C필러
 P06: 사이드미러
 P07: 앞도어
 P08: 뒷도어

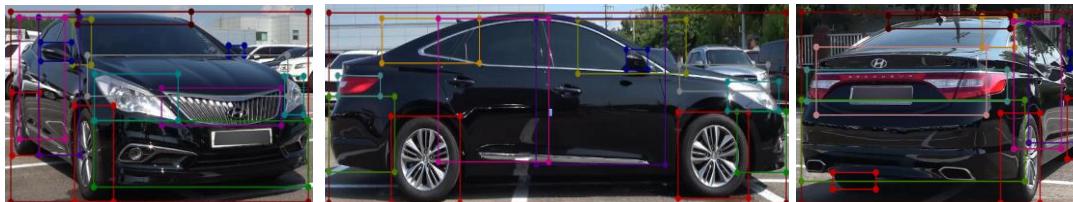
P09: 라디에이터그릴
 P10: 헤드램프
 P11: 리어램프
 P12: 보닛
 P13: 트렁크
 P14: 루프
 와 같으며, 차량 전체가 보이는 사진 중 리어, 헤드램프 유무 등으로 전/후/측면을 판단할 수 있을 것이라 생각하였음



프로젝트 수행절차 <데이터셋 제작>

데이터 경량화 및 레이블 분류

- 총 80GB의 자료를 분석하기에는 컴퓨팅 자원이 부족하여 경량화 진행
- 국내 브랜드인 [현대/기아](45GB)만을 대상으로 데이터 추출
- 전면: 헤드램프가 2개가 포함된 사진
- 후면: 리어램프 2개가 포함된 사진 + 리어램프 bounding box 비율이 1:6을 넘는 사진
- 측면: 헤드, 리어램프가 동시에 존재하는 사진



- 전체 차량 이미지만 사용 + P00: 차량전체 바운딩박스에 맞게 사진 크립
- 차량 연식으로 구분된 폴더를 수작업으로 연형으로 구분 진행
(동일한 외관에 따른 분류를 진행하기 위함)
- 최종데이터셋(4GB)

폴더구조

```

▼ 091.차량외관영상데이터
  ▼ 01.데이터
    ▼ 1.Training
      ▼ 라벨링데이터
        > TL1
        > TL2
        ▼ TL3
          > KI_기아
          > LA_랜드로버
    
```

상위폴더(차종)

```

026_K3
027_K5
028_K7
029_K9
043_니로
047_레이
051_모닝
052_모하비
056_봉고3
057_셀토스

```

하위폴더(연식+α)

```

2017_검정_트립A
2017_검정_트립B
2017_회색_트립A
2017_회색_트립B
2017_회색_트립C
2017_흰색_트립A
2017_흰색_트립B
2017_흰색_트립C
2018_검정_트립A
2018_검정_트립B

```

상위폴더 (앞뒤옆 분류)

```

angle.jpg
car_back
car_front
car_side

```

연식 분류

```

기아_K3_2017
기아_K3_2018
기아_K3_2019
기아_K3_2020
기아_K3_2021
기아_K5_2017
기아_K5_2018
기아_K5_2019
기아_K5_2020

```

연형 분류

```

기아_K3_17-18
기아_K3_19-21
기아_K5_17_MX
기아_K5_17_SX
기아_K5_18-19
기아_K5_20-21
기아_K7_17-19
기아_K7_20-21
기아_K9_17
기아_K9_18-21

```

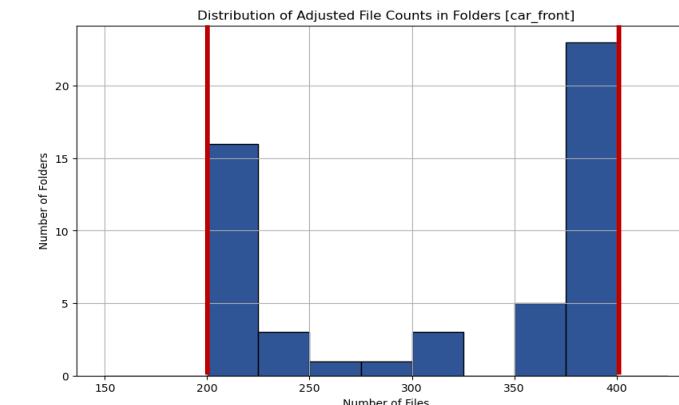
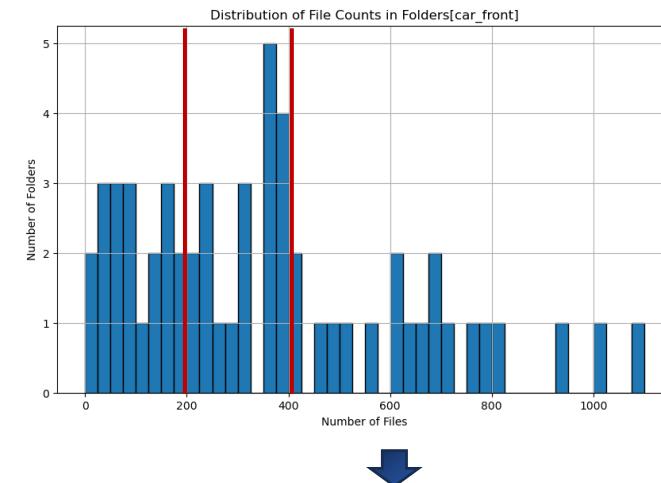
프로젝트 수행절차 <데이터셋 제작>

데이터 불균형 해소

- 각 클래스별 이미지 개수가 50개가 안되는 클래스를 drop
 - 또한, 데이터 불균형 완화를 위해 데이터 개수가 모자란 클래스에 대하여 Augmentation + 데이터가 많은 클래스에 대하여 undersampling (원저화 진행)
 - 전면: {'min': 200, 'max': 400},
후면 : {'min': 240, 'max': 480},
측면 : {'min': 270, 'max': 540}
- } 요약통계 및 분포 검토 후,
데이터 균형과 데이터 소실간의
밸런스를 고려하여 결정

```
# 데이터 증강 방법
datagen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.10,
    brightness_range=(0.8, 1.2),
    height_shift_range=0.10,
    fill_mode='nearest',
    horizontal_flip=True,
    rescale=1./255
)
```

예시 (전면)



- 클래스: 62
- 데이터: 17412

- 클래스: 57
- 데이터: 14901

프로젝트 수행절차 <EfficientNet 학습>

모델링을 위한 데이터 전처리

- Keras 내 ImageDataGenerator를 이용하여 데이터 전처리 진행
- Pixel 값을 0~1의 값을 갖도록 정규화
- Flow_from_directory를 통해 directory 구분에 따른 레이블 지정
- Target_size를 EfficientNetV2B0 분석에 맞는 (224, 224)로 지정

```
directory = "train_dir3/car_front"
n_class = len(os.listdir(directory)) # 57
batch_size = 32
img_size = (224, 224)

def make_dataset(train_dir, val_dir):
    train_datagen = ImageDataGenerator(rescale=1./255)
    validation_datagen = ImageDataGenerator(rescale=1./255)

    train_generator = train_datagen.flow_from_directory(
        train_dir, batch_size=batch_size, target_size=img_size, class_mode='sparse', shuffle=True)
    validation_generator = validation_datagen.flow_from_directory(
        val_dir, batch_size=batch_size, target_size=img_size, class_mode='sparse')

    return train_generator, validation_generator
```

모델 생성

- Keras 내 EfficientNet2B0 사용
- Include_top = False, weights = 'imagenet' 사용
- GlobalAveragePooling2D, Dropout을 적용한 뒤 구분하고자 하는 레이블의 개수로 Dense Layer 적용
- Optimizer: Adam(learning_rate = 0.00001)
- 학습률 0.0001 사용 시, val_acc가 수렴하지 못하고 뛰는 현상 발생
- Loss: sparse_categorical_crossentropy
- Metrics: Accuracy

```
def make_model():
    inputs = Input(shape=(224, 224, 3))
    x = EfficientNetB0(include_top=False, weights='imagenet')(inputs)
    x = GlobalAveragePooling2D()(x)
    x = Dropout(0.2)(x)
    outputs = Dense(n_class, activation='softmax')(x)
    model = Model(inputs=inputs, outputs=outputs)
    model.compile(optimizer=Adam(learning_rate=0.00001),
                  loss='sparse_categorical_crossentropy', metrics=['accuracy'])

    return model
```

프로젝트 수행절차 <EfficientNet 학습>

모델 학습

- ModelCheckpoint를 이용하여 val_acc가 가장 높은 best 결과만 저장

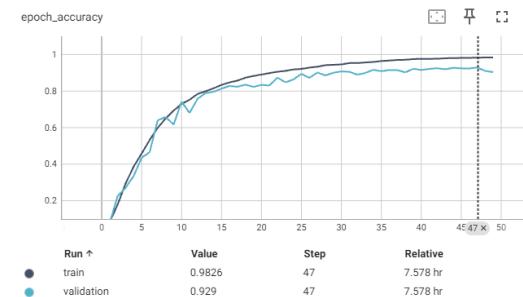
```
model_checkpoint_callback = ModelCheckpoint(
    filepath=model_path,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True,
    verbose=1
)
```

- Epoch = 50

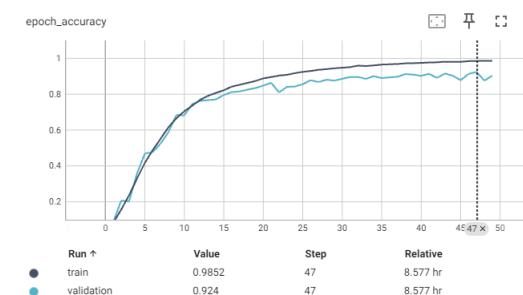
```
history = model.fit(
    train_generator,
    epochs=50,
    validation_data=validation_generator,
    callbacks=[
        model_checkpoint_callback,
        tensorboard_callback
    ]
)
```

모델 검증

- Front 학습 결과,
- Val_accuracy: 0.929/ val_loss: 0.3017

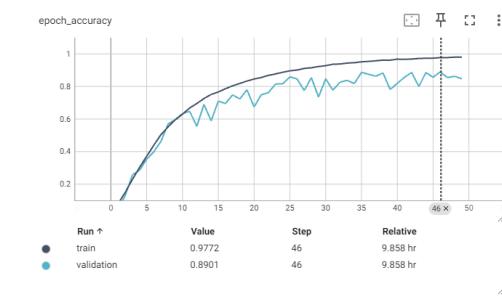


- Back 학습 결과,
- Val_accuracy: 0.924/ val_loss: 0.3178



모델 검증

- Side 학습 결과,
- Val_accuracy: 0.890/ val_loss: 0.346



- 평균 9시간 소요
- 전반적으로 25 Epoch 이후, 수렴하는 모습 (Early-stopping 필요성)
- Side에 해당하는 각도의 범위가 크고 차종을 구분할 수 있는 특징적인 부분들이 적다 보니, 학습시 변동성이 큰 모습을 보임 (조건을 추가해서 적합성을 높일 필요성)

프로젝트 수행절차 <EfficientNet 학습>

기존 조건 (차량의 파트만을 고려)

- 전면: 헤드램프가 2개가 포함된 사진
- 후면: 리어램프 2개가 포함된 사진 + 리어램프 bounding box 비율이 1:6을 넘는 사진
- 측면: 헤드, 리어램프가 동시에 존재하는 사진

```
def validate_conditions(json_data, folder_type):
    labels = [shape['label'] for shape in json_data['shapes']]
    if folder_type == 'car_back':
        # Conditions for "car_back"
        if "P00.차량전체" in labels and "P10.헤드램프" not in labels:
            rear_lamps = [shape for shape in json_data['shapes'] if shape['label'] == "P11.리어램프"]
            if len(rear_lamps) == 1:
                x1, y1, x2, y2 = get_crop_coordinates(json_data, "P11.리어램프")
                width = x2 - x1
                height = y2 - y1
                return width >= 6 * height
            elif len(rear_lamps) == 2:
                return True
            return False
        return False
    elif folder_type == 'car_front':
        # Conditions for "car_front"
        if "P00.차량전체" in labels and "P11.리어램프" not in labels:
            head_lamps = [shape for shape in json_data['shapes'] if shape['label'] == "P10.헤드램프"]
            if len(head_lamps) == 2:
                return True
            return False
        return False
    elif folder_type == 'car_side':
        # Conditions for "car_side"
        required_labels = {"P10.헤드램프", "P11.리어램프", "P00.차량전체"}
        counts = {label: sum(1 for shape in json_data['shapes'] if shape['label'] == label) for label in required_labels}
        # Check for exactly one of each required label
        if all(count == 1 for count in counts.values()):
            return True
        return False
    return True
```

추가된 조건 (차량의 파트 + 각도를 고려)

- 전면: 한 개의 헤드램프의 width가 다른 하나의 헤드램프의 width의 5배를 넘지 않는다
- 후면: 한 개의 리어램프의 width가 다른 하나의 리어램프의 width의 5배를 넘지 않는다
- 측면: 헤드램프의 width가 리어램프의 width의 4배를 넘지 않는다
(vice versa)

```
def validate_conditions(json_data, folder_type):
    labels = [shape['label'] for shape in json_data['shapes']]
    if folder_type == 'car_back':
        # Conditions for "car_back"
        if "P00.차량전체" in labels and "P10.헤드램프" not in labels:
            rear_lamps = [shape for shape in json_data['shapes'] if shape['label'] == "P11.리어램프"]
            if len(rear_lamps) == 1:
                x1, y1, x2, y2 = get_crop_coordinates(json_data, "P11.리어램프")
                width = x2 - x1
                height = y2 - y1
                return width >= 6 * height
            elif len(rear_lamps) == 2:
                widths = [get_crop_coordinates(json_data, lamp['label'])[2] - get_crop_coordinates(json_data, lamp['label'])[0] for lamp in rear_lamps]
                return not (widths[0] > 5 * widths[1] or widths[1] > 5 * widths[0])
            return False
        elif folder_type == 'car_front':
            # Conditions for "car_front"
            if "P00.차량전체" in labels and "P11.리어램프" not in labels:
                head_lamps = [shape for shape in json_data['shapes'] if shape['label'] == "P10.헤드램프"]
                if len(head_lamps) == 2:
                    widths = [get_crop_coordinates(json_data, lamp['label'])[2] - get_crop_coordinates(json_data, lamp['label'])[0] for lamp in head_lamps]
                    return not (widths[0] > 5 * widths[1] or widths[1] > 5 * widths[0])
                return False
            elif folder_type == 'car_side':
                # Conditions for "car_side"
                required_labels = {"P10.헤드램프", "P11.리어램프", "P00.차량전체"}
                counts = {label: sum(1 for shape in json_data['shapes'] if shape['label'] == label) for label in required_labels}
                # Check for exactly one of each required label
                if all(count == 1 for count in counts.values()):
                    head_lamp_width = get_crop_coordinates(json_data, "P10.헤드램프")[2] - get_crop_coordinates(json_data, "P10.헤드램프")[0]
                    rear_lamp_width = get_crop_coordinates(json_data, "P11.리어램프")[2] - get_crop_coordinates(json_data, "P11.리어램프")[0]
                    if not (head_lamp_width > 4 * rear_lamp_width or rear_lamp_width > 4 * head_lamp_width):
                        return True
                    return False
                return True
            return True
        return True
```

프로젝트 수행절차 <EfficientNet 학습>

추가 수정된 부분

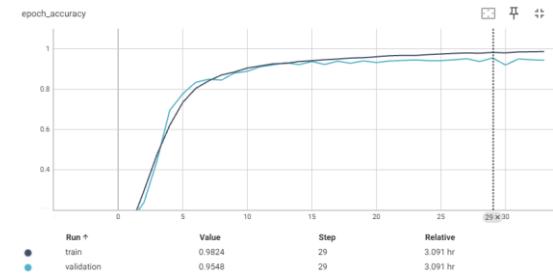
```
The average number of JPG files per folder in 'angle_jpg4/car_front' is: 237.55
The average number of JPG files per folder in 'angle_jpg4/car_back' is: 270.91
The average number of JPG files per folder in 'angle_jpg4/car_side' is: 253.60
```

- Augmentation+Undersampling을 통해 모든 클래스의 파일 개수를 250으로 맞춤 (전면, 후면, 측면)
- 모델 성능 증가 및 과적합 감소를 위해 validation_split을 0.2에서 0.1로 조정
- 과적합 방지 및 효율적인 자원 활용을 위해 early_stopping 도입

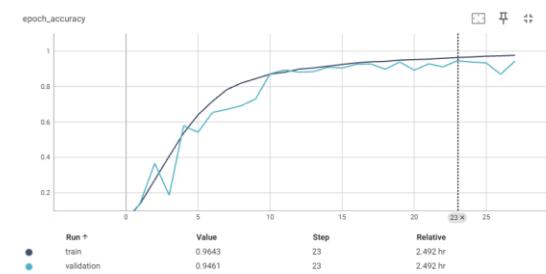
```
# Configure the EarlyStopping callback
early_stopping_callback = EarlyStopping(
    monitor='val_accuracy',
    patience=3,
    verbose=1,
    mode='max'
)
```

모델 검증

- Front 학습 결과,
- Val_accuracy: 0.954/ val_loss: 0.140

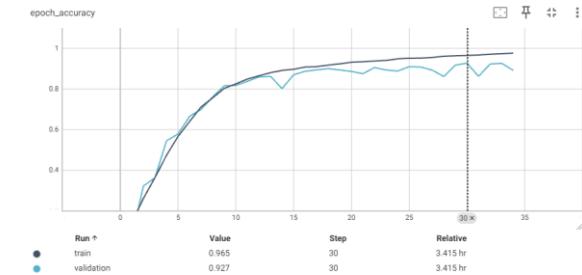


- Back 학습 결과,
- Val_accuracy: 0.946/ val_loss: 0.162



모델 검증

- Side 학습 결과,
- Val_accuracy: 0.927/ val_loss: 0.206



- 평균소요 시간: 9시간 → 3시간
- 평균 Best Epoch: 47 → 27
- 전면 val_acc: 0.929 → 0.954
- 후면 val_acc: 0.924 → 0.946
- 측면 val_acc: 0.890 → 0.927
- 조건을 추가하여 적합성을 높이면 학습 성능이 좋아지는 것을 확인할 수 있었음

프로젝트 수행절차 <멀티 스테이지 딥러닝 파이프라인>

[객체 탐지 및 방향 탐지 → 차종 분류] 파이프라인

- 객체 탐지 및 방향 탐지를 위해 NextLab로부터 제공받은 Yolo4 모델을 사용
- 모델의 설정이 저장된 "yolo4_car.cfg" file
- 모델의 가중치가 저장된 "yolo4-car_best_weights" file
- 모델의 클래스가 저장된 "car.names" file

```
# 학습된 YOLO모델 로드하기  
yolo_model = cv2.dnn.readNet("Model3/yolo_model/yolov4-car_best.weights", "Model3/yolo_model/yolov4-car.cfg")  
  
# EfficientNet 모델 로드하기  
model_front = load_model('Model3/car_front/eff_model.h5')  
model_back = load_model('Model3/car_back/eff_model.h5')  
model_side = load_model('Model3/car_side/eff_model.h5')  
  
# 차량 방향에 해당하는 모델 지정하기  
models = {'car_front': model_front, 'car_back': model_back, 'car_side': model_side}
```

- Yolo4 모델에 의해 차량 방향 클래스가 부여되면 그에 해당하는 EfficientNet 모델을 사용한다

프로젝트 수행절차 <멀티 스테이지 딥러닝 파이프라인>

[객체 탐지 및 방향 탐지 → 차종 분류] 파이프라인

```
def process_and_classify(img_path, classes_orientation, classes_brand):
    img, outputs, width, height = process_image(img_path)
    boxes, confidences, class_ids = get_boxes(outputs, width, height)
    detected_brands = [] # 감지된 브랜드 이름을 저장할 리스트

    # Non-Maximum Suppression 적용
    indexes = cv2.dnn.NMSBoxes(boxes, confidences, score_threshold=0.5, nms_threshold=0.4)

    # NMS를 통과한 인덱스만 사용
    final_boxes = [boxes[i] for i in indexes.flatten()]
    final_confidences = [confidences[i] for i in indexes.flatten()]
    final_class_ids = [class_ids[i] for i in indexes.flatten()]

    for i, box in enumerate(final_boxes):
        x, y, w, h = box
        crop_img = img[y:y+h, x:x+w] ← 이미지 안에서 각 객체 감지된 영역 추출
        orientation = classes_orientation[class_ids[i]]
        brand_index = classify_car(crop_img, orientation) # 예측된 브랜드의 인덱스 가져오기
        brand_name = classes_brand[brand_index] # 인덱스에서 브랜드 이름 가져오기
        detected_brands.append((brand_name, box)) # 브랜드 이름과 해당 박스 추가
        print(f"감지됨: {brand_name} 방향: {orientation}")

    draw_labels(final_boxes, final_confidences, final_class_ids, classes_orientation, detected_brands, img)
```

- I. process_image(img_path): 이미지 경로에서 이미지를 로드하고, YOLO 객체 감지 모델을 사용하여 이미지 내의 객체들을 감지한 후, 감지된 객체들의 정보 반환
- II. get_boxes(outputs, width, height): YOLO 객체 감지 모델의 출력에서 각 객체의 위치, 신뢰도, 클래스 ID를 추출하여 이미지 내의 객체를 정확하고 효율적으로 식별하고 위치를 파악
- III. Non-Maximum Suppression: 객체 감지 과정에서 중복된 감지 결과를 제거하여 각 객체를 정확히 하나의 박스로 표현
- IV. classify_car(crop_img, orientation): 차량의 방향에 최적화된 모델 (EfficientNet; 전면, 후면, 측면 中)을 사용하여 이미지 내 차량의 종류를 정확하게 분류
- V. Draw_labels(final_boxes, ...): 이미지 위에 바운딩 박스와 감지된 차량의 종류 텍스트 및 신뢰도를 추가하여 사용자에게 결과를 명확하고 직관적으로 제공

A photograph of a man in a striped polo shirt and cargo shorts taking a photograph of a bright yellow sports car. He is holding a camera with a lens attached. In the background, there are other people, a red car, and buildings with signs for "MOTOR CONDOS", "elm", and "AVALON".

Part 3

프로젝트 수행결과

프로젝트 수행결과

Test Image 수행 결과

- Train, Validation에는 포함되지 않는 새로운 이미지를 대상으로 Test 진행
- 총 128장의 사진 중 7장의 사진 CNN 분류 오류 발생
- 분류 정확도는 121/128. 약 93.8%

분류 오류 사례



- Train Dataset을 확인해보니, 넥쏘의 경우 골드 색상의 차량이 없어 전혀 학습되지 않았다.
색상이 차량 분류에 영향을 주는 듯하다. [데이터 다양성 부족으로 인한 오류]

프로젝트 수행결과

분류 오류 사례



데이터 부족

기아 니로 17-18년형을 21년형으로 분류하였다.
전체적인 차체의 형상이 유사하여 발생된 오류인 듯하다.



데이터셋 제작

기아 카니발 19-20년형을 17-18년형으로 다르게 분류.

후면의 경우, 19-20년형과 17-18년형을 리어 범퍼의 모양으로 구별할 수 있다. 그런데, 17-18년형의 Train Dataset에서 19-20년형의 사진을 여럿 볼 수 있었다.



바운딩박스 조건

현대 그랜저 _17의 Train Dataset을 확인해보니,
오리지널 데이터가 5장에 불과했음.
이로인한 과적합 발생이 된 듯함.

헤드램프가 하나일 경우, 바운딩박스 비율을 1:6에서 1:4로 변경하는 것이 좋아 보인다.



바운딩박스 조건

현대 그랜저 _17과 같은 이슈.
현대 그랜저 _18-19의 Train Dataset을 확인해보니,
오리지널 데이터가 11장에 불과했음.

프로젝트 수행결과

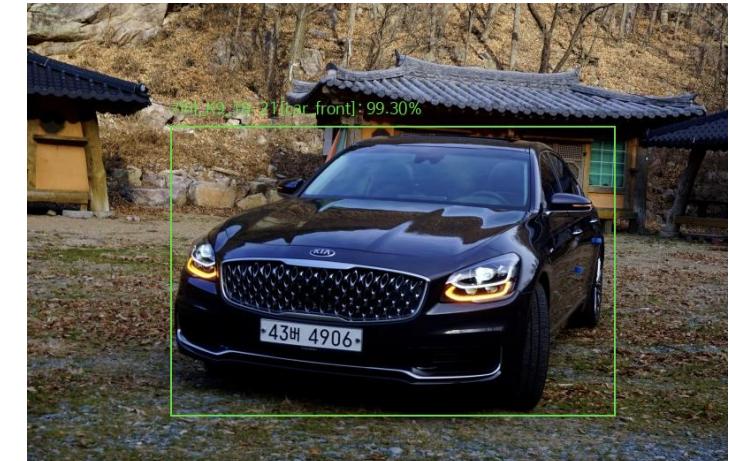
분류 성공 사례



<기아 K3 19-21>



<기아 K5 17 MX>



<기아 K9 18-21 MX>

- 헤드램프, 프론트 범퍼, 라디에이터 그릴 등 특징을 잘 캐치하여 기아의 K시리즈를 잘 구분하는 모습

프로젝트 수행결과

분류 성공 사례



<현대 아이오닉 17-20 전기차>



<현대 아이오닉 17-20 하이브리드>

- 현대 아이오닉 17-20 모델에서는 라디에이터와 같은 세부 차이를 기반으로 전기차와 하이브리드차의 구분이 정확하게 이루어짐

프로젝트 수행결과

분류 성공 사례



<현대 싼타페 17-18>



<현대 싼타페 19-21 Inspiration>

- 측면에서도 차체와 부품들을 정확하게 인식하여 현대 싼타페의 다양한 연형을 성공적으로 구분

프로젝트 수행결과

분류 성공 사례



<기아 카니발 17-18>



<기아 카니발 19-20>



<기아 카니발 21>

- 후면에서도 차체와 부품들을 정확하게 인식하여 기아 카니발의 다양한 연형을 성공적으로 구분



A symmetrical photograph of palm trees reflected in water under a cloudy sky. The image is split vertically by a central axis, with palm trees on both sides. A horizontal line with a blue bar above it cuts across the center of the image, containing the text "Part 4" and "프로젝트 회고".

Part 4
프로젝트 회고

프로젝트 회고

프로젝트를 진행하면서 겪은 문제점들

- Colab 사용 시, 압축 파일을 Google Drive에 업로드하는 것만으로도 24시간 이상이 걸려 진행이 불가했다. 그 뒤, 코랩 세션에서 압축파일을 풀어 진행하는 프로세스도 오랜 시간이 예상되었다.
- 효율적인 시간 활용과 코랩 컴퓨팅 자원 활용을 위하여 모델링 전 단계로 CPU를 활용하는 모든 단계는 로컬에서 진행하고, 데이터를 4GB 이하로 경량화 후 구글에 업로드한 후, 모델링 및 이후 작업은 코랩에서 진행하였다.
- 폴더를 구글 드라이브에 업로드할 때, 여러 세션에서 분할하여 업로드하면 시간을 세이브 할 수 있다. (구글드라이브 사용자별 트래픽 제한이 있어 4개 이상에선 문제가 생겼다.)
- 구글 코랩의 여러 세션에서 동시에 모델을 돌리면 리소스 고갈로 인해 강제 종료되는 문제가 생긴다.
- OpenCV의 경우, 한글과의 호환성이 좋지 않았다.
- Image에 Text를 입력하는 경우, 한글이 지원되는 PIL(pillow) 모듈 형식으로 변경한 뒤 진행.

- training과 validation에서 높은 정확률을 보여주는데 test만 하면 이상한 분류하는 문제가 있었다.
- os.listdir 함수는 디렉토리를 특정 순서대로 반환하는 것이 아니라서 꼭 sort 함수를 사용하여 알파벳 순서대로 정렬해야 된다.
- EfficientNet 모델의 학습 과정에서 검증 정확도(val_acc)가 비정상적으로 변동하는 현상이 관찰되었다.
- 트레이닝 데이터에 대한 학습 곡선이 매우 빠르게 1에 수렴하는 것을 보고, 학습률이 너무 높다고 판단하여 Adam 옵티마이저의 학습률을 0.0001에서 0.00001로 낮춘 후 이 문제가 해결되었다.

프로젝트 회고

배운점 및 느낀점

- YoloV5 및 Next_lab에서 제공한 가중치를 이용한 YoloV4 모델 제작을 통해 객체 탐지 모델 제작 과정 전반에 대한 이해를 얻었다.
- 다양한 CNN 기법을 공부하고, 기존 모델을 활용하여 사용자 정의 데이터셋에 전이 학습을 적용하는 방법을 숙달하게 되었다.
- 파이썬을 사용하여 원본 데이터에서 필요한 데이터를 추출하고 분류하는 과정을 통해 데이터 전처리의 중요성과 복잡성을 경험했다.
- CNN 모델을 직접 훈련해 보면서 훈련 데이터의 품질이 모델 성능에 큰 영향을 미친다는 것을 확인했다.

아쉬운점

- 대부분의 분류 오류 사례를 살펴보면 최종 데이터셋 제작에서 개선할 수 있는 부분이였다.
 - 후면 리어램프 하나일 경우, 데이터가 부족했던 문제 (바운딩박스 비율 1:6에서 1:4로 조정)
 - Train Dataset에서 사진이 잘못 분류되어있던 문제 (모든 사진 데이터를 살펴보며 더 정확한 클래스 분류)
 - 전체적인 데이터 부족 문제 (차량 전체 사진만 사용하지 않고 부분 사진이라도 핵심 부품이 포함되어 있다면 사용 eg. 전면의 경우: 헤드램프, 라디에이터, 프론트 범퍼 등)
- 이번 프로젝트는 80GB 데이터 중 45GB의 현대·기아 자동차 데이터만을 대상으로 진행되었다. 이로 인해 실제 현장에서의 활용 가능성은 낮으나, 추가 데이터셋 학습을 통해 보완할 수 있다.
- 앞으로 자동차 모델은 계속 추가될 것이니, 현재 방법으로는 새로운 모델이 추가될 때마다 학습을 다시 해야 하는 단점이 있다.



*thank
you*

감사합니다