

한국IT교육원

EfficientDet

01 프로젝트 개요

▶ EfficientDet이란?

- EfficientDet 은 구글에서 개발한 객체 탐지 모델로, **작은 모델 사이즈**와 **높은 정확도**를 동시에 달성하는 것이 특징입니다. EfficientNet이라는 **경량화 된 네트워크 구조를 기반**으로 하여 다양한 크기의 모델을 만들어냅니다.
- 객체 탐지 문제를 해결하기 위한 아래와 같은 방법을 결합합니다.
 - Faster R-CNN Anchor box 방법과 RPN 구조.
 - MobileNet의 Depth-Wise 연산과 Bottleneck 구조.
 - RetinaNet의 Top-Down 연산.
 - ResNet의 Top-Down 연산.
- **BiFPN(Bi-directional Feature Pyramid Network)**이라는 새로운 특징 추출 방법을 사용하여 병렬적인 정보 흐름을 통해 정확도와 효율성 모두를 향상시키는 것을 목적으로 합니다. 이를 통해 작은 객체부터 큰 객체까지 정확하게 검출할 수 있습니다.

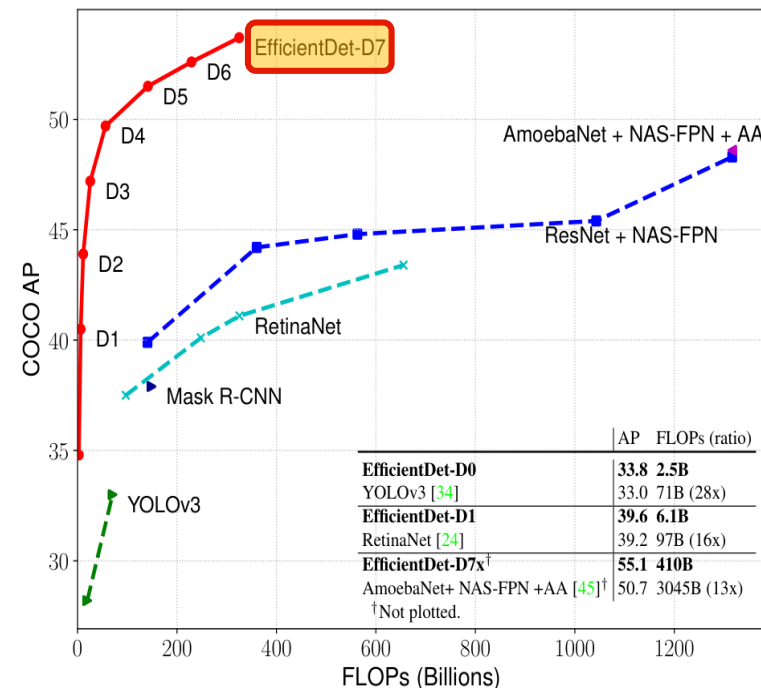
01 프로젝트 개요

▶ 기획 의도

EfficientDet은 최근 Object Detection 분야에서 매우 획기적인 발전을 이룬 딥 러닝 모델 SOTA(State-of-the-Art) 입니다.

기존의 Object Detection 모델들보다 **더욱 높은 정확도**를 보여주면서도, **더욱 적은 연산량과 파라미터 수**를 가지고 있어서보다 효율적인 모델 구현이 가능합니다.

이로 인해 Datasets 전 처리 및 EfficientDet 모델 구현과 학습을 통해서 최신의 Object Detection 모델을 학습하고자 합니다.



01 프로젝트 개요

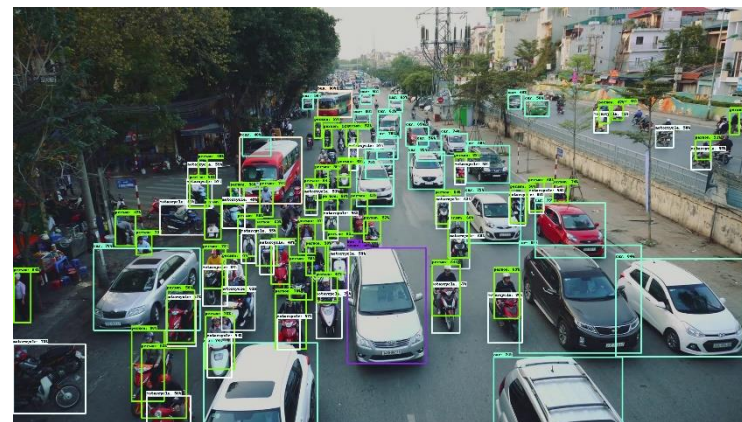
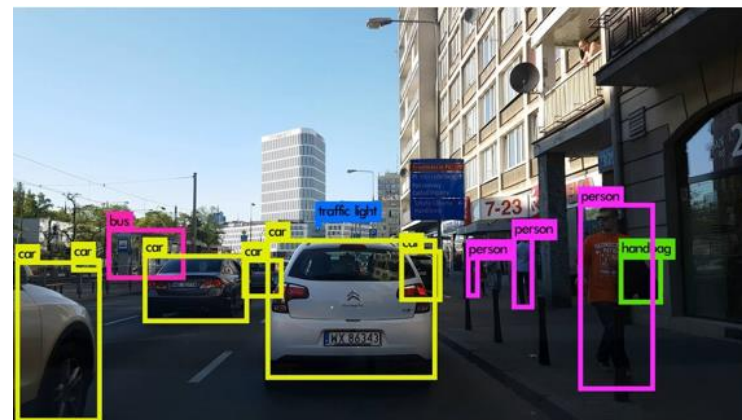
▶ 기대 효과

EfficientDet 모델은 Object Detection 분야 뿐만 아니라, 다양한 분야에서 응용 가능합니다.

예를 들어 자율주행, 보안 감시, 영상 인식 등에서 활용될 수 있습니다.

자율주행차에서는 차량 및 보행자 등의 객체 검출에 사용되며,

보안 분야에서는 인명 구조나 범죄 예방을 위한 CCTV 모니터링 등에 사용될 수 있습니다.



1. Datasets 수집 및 전 처리

2. Backbone Network

EfficientNet에서 사용된 MobileNetV2의 구조를 기반으로 한 EfficientNet 모델을 사용합니다.

3. Feature Pyramid Network

다양한 스케일의 특징 맵을 활용하여 Object Detection을 수행하는 Feature Pyramid Network(FPN)을 사용합니다.

EfficientDet은 EfficientNet의 네트워크 구조를 활용하여 FPN을 구성하였습니다.

4. BiFPN(Bi-directional Feature Pyramid Network)

BiFPN은 FPN에서 발생하는 문제를 해결하기 위해 도입되었습니다. EfficientDet은 EfficientNet의 네트워크 구조를 활용하여 BiFPN을 구성하였습니다. BiFPN은 상위 및 하위 레벨의 피라미드 기반 기능을 조합하여 높은 정확도와 빠른 속도를 동시에 보장합니다.

5. Dataset 학습 및 Test 후 평가

03 프로젝트 개요

▶ 활용 장비 및 재료(개발 환경 등)

- Python ver_3.11
- Tensorflow Lite
- Google Colaboratory
- Google Drive
- GALAXY S21 (android)

04 프로젝트 수행 결과

결과 제시 ① 탐색적 분석 및 전처리

▶ 학습 데이터 소개 (Train/dev set)

커스텀 데이터셋 (Custom Dataset)

A.I hub dataset , 사진(Labeling dataset)

```
<annotation>
  <folder></folder>
  <filename>free-video-854671_jpg.rf.99196cb032c78d9d459fd350bee181dd.jpg</filename>
  <path>free-video-854671_jpg.rf.99196cb032c78d9d459fd350bee181dd.jpg</path>
  <source>
    <database>roboflow.ai</database>
  </source>
  <size>
    <width>640</width>
    <height>640</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>car</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <occluded>0</occluded>
    <bndbox>
      <xmin>288</xmin>
      <xmax>410</xmax>
      <ymin>456</ymin>
      <ymax>641</ymax>
    </bndbox>
  </object>
  <object>
    <name>car</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <occluded>0</occluded>
    <bndbox>
      <xmin>456</xmin>
      <xmax>564</xmax>
      <ymin>341</ymin>
      <ymax>534</ymax>
    </bndbox>
  </object>
</annotation>
```

```
imagepath = "./images"

# read the JSON file into an in-memory object
# the JSON file contains data like this:
#
# {
#   "image1.jpg": [
#     {
#       "label": "label1",
#       "coordinates": {
#         "height": 243,
#         "width": 216,
#         "x": 174,
#         "y": 276
#       }
#     },
#     {
#       "label": "label2",
#       "coordinates": {
#         "height": 104,
#         "width": 217,
#         "x": 174,
#         "y": 450
#       }
#     },
#     ...
#   ],
#   "image2.jpg": [
#     {
#       "label": "label1",
#       "coordinates": {
#         "height": 173,
#         "width": 314,
#         "x": 324,
#         "y": 254
#       }
#     },
#     ...
#   ],
# }
# }
```

XML Format

JSON Format



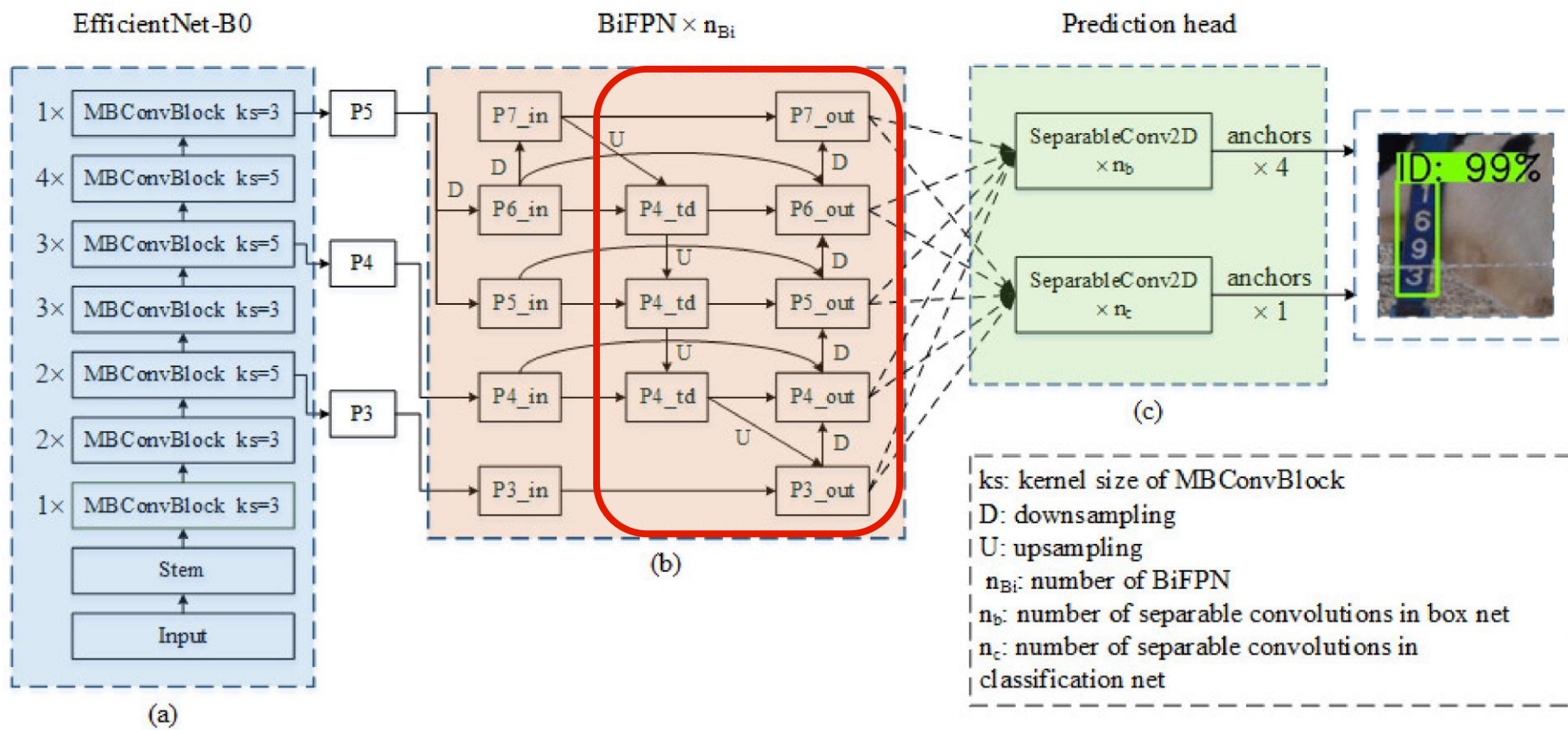
filename	width	height	class	xmin	ymin	xmax	ymax
00014_jpg.rf.011641...	416	416	car	9	144	413	412
00011_jpg.rf.99f968...	416	416	car	19	67	416	413
00009_jpg.rf.373cf8...	416	416	car	2	34	417	409
00012_jpg.rf.6a938f...	416	416	car	4	34	416	389
00006_jpg.rf.1e36fc...	416	416	car	185	265	388	408
00004_jpg.rf.1bd392...	416	416	car	109	96	321	351
00002_jpg.rf.34d3b5...	416	416	car	43	68	161	140
00002_jpg.rf.34d3b5...	416	416	car	363	78	416	130
00002_jpg.rf.34d3b5...	416	416	car	2	67	55	176
00002_jpg.rf.34d3b5...	416	416	car	33	84	397	371
00003_jpg.rf.4156bf...	416	416	car	43	91	407	324
000959_JPG.rf.378b...	817	829	motorcycle	523	667	605	773
000959_JPG.rf.378b...	817	829	person	607	596	639	681
000959_JPG.rf.378b...	817	829	car	45	590	176	722
000959_JPG.rf.378b...	817	829	car	253	648	452	830

CSV Format

04 프로젝트 수행 결과

결과 제시 ② 모델 개요

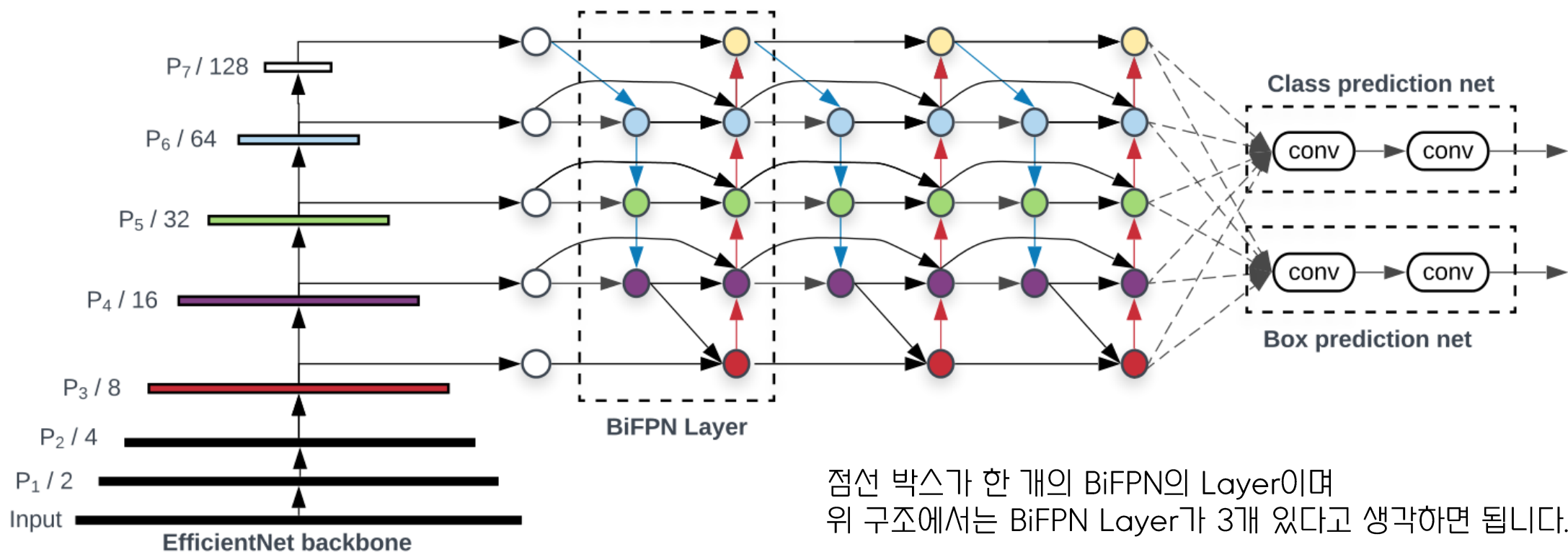
▶ EfficientDet 모델 구조



04 프로젝트 수행 결과

결과 제시 ③ 모델 선정 및 분석

▶ EfficientDet 분석



04 프로젝트 수행 결과

결과 제시 ③ 모델 선정 및 분석

▶ EfficientDet 코드 분석

```
def build_BiFPN(features, num_channels, id, freeze_bn=False):
    if id == 0:
        C3, C4, C5 = features
        P3_in = C3
        P4_in = C4
        P5_in = C5
        P6_in = layers.Conv2D(num_channels, kernel_size=1, padding='same', name='resample_p6/conv2d')(C5)
        P6_in = layers.BatchNormalization(momentum=MOMENTUM, epsilon=EPSILON, name='resample_p6/bn')(P6_in)
        # P6_in = BatchNormalization(freeze=freeze_bn, name='resample_p6/bn')(P6_in)
        P6_in = layers.MaxPooling2D(pool_size=3, strides=2, padding='same', name='resample_p6/maxpool')(P6_in)
        P7_in = layers.MaxPooling2D(pool_size=3, strides=2, padding='same', name='resample_p7/maxpool')(P6_in)
        P7_U = layers.UpSampling2D()(P7_in)
        P6_td = layers.Add(name=f'fpn_cells/cell_{id}/fnode0/add')([P6_in, P7_U])
        P6_td = layers.Activation(lambda x: tf.nn.swish(x))(P6_td)
        P6_td = SeparableConvBlock(num_channels=num_channels, kernel_size=3, strides=1,
                                   name=f'fpn_cells/cell_{id}/fnode0/op_after_combine5')(P6_td)
        P5_in_1 = layers.Conv2D(num_channels, kernel_size=1, padding='same',
                                name=f'fpn_cells/cell_{id}/fnode1/resample_0_2_6/conv2d')(P5_in)
        P5_in_1 = layers.BatchNormalization(momentum=MOMENTUM, epsilon=EPSILON,
                                             name=f'fpn_cells/cell_{id}/fnode1/resample_0_2_6/bn')(P5_in_1)
        # P5_in_1 = BatchNormalization(freeze=freeze_bn, name=f'fpn_cells/cell_{id}/fnode1/resample_0_2_6/bn')(P5_in_1)
        P6_U = layers.UpSampling2D()(P6_td)
        P5_td = layers.Add(name=f'fpn_cells/cell_{id}/fnode1/add')([P5_in_1, P6_U])
        P5_td = layers.Activation(lambda x: tf.nn.swish(x))(P5_td)
        P5_td = SeparableConvBlock(num_channels=num_channels, kernel_size=3, strides=1,
                                   name=f'fpn_cells/cell_{id}/fnode1/op_after_combine6')(P5_td)
        P4_in_1 = layers.Conv2D(num_channels, kernel_size=1, padding='same',
                                name=f'fpn_cells/cell_{id}/fnode2/resample_0_1_7/conv2d')(P4_in)
        P4_in_1 = layers.BatchNormalization(momentum=MOMENTUM, epsilon=EPSILON,
                                             name=f'fpn_cells/cell_{id}/fnode2/resample_0_1_7/bn')(P4_in_1)
        # P4_in_1 = BatchNormalization(freeze=freeze_bn, name=f'fpn_cells/cell_{id}/fnode2/resample_0_1_7/bn')(P4_in_1)
        P5_U = layers.UpSampling2D()(P5_td)
        P4_td = layers.Add(name=f'fpn_cells/cell_{id}/fnode2/add')([P4_in_1, P5_U])
        P4_td = layers.Activation(lambda x: tf.nn.swish(x))(P4_td)
        P4_td = SeparableConvBlock(num_channels=num_channels, kernel_size=3, strides=1,
                                   name=f'fpn_cells/cell_{id}/fnode2/op_after_combine7')(P4_td)
        P3_in = layers.Conv2D(num_channels, kernel_size=1, padding='same',
                                name=f'fpn_cells/cell_{id}/fnode3/resample_0_0_8/conv2d')(P3_in)
        P3_in = layers.BatchNormalization(momentum=MOMENTUM, epsilon=EPSILON,
                                             name=f'fpn_cells/cell_{id}/fnode3/resample_0_0_8/bn')(P3_in)
        # P3_in = BatchNormalization(freeze=freeze_bn, name=f'fpn_cells/cell_{id}/fnode3/resample_0_0_8/bn')(P3_in)
```

```
P4_U = layers.UpSampling2D()(P4_td)
P3_out = layers.Add(name=f'fpn_cells/cell_{id}/fnode3/add')([P3_in, P4_U])
P3_out = layers.Activation(lambda x: tf.nn.swish(x))(P3_out)
P3_out = SeparableConvBlock(num_channels=num_channels, kernel_size=3, strides=1,
                              name=f'fpn_cells/cell_{id}/fnode3/op_after_combine8')(P3_out)
P4_in_2 = layers.Conv2D(num_channels, kernel_size=1, padding='same',
                          name=f'fpn_cells/cell_{id}/fnode4/resample_0_1_9/conv2d')(P4_in)
P4_in_2 = layers.BatchNormalization(momentum=MOMENTUM, epsilon=EPSILON,
                                     name=f'fpn_cells/cell_{id}/fnode4/resample_0_1_9/bn')(P4_in_2)
# P4_in_2 = BatchNormalization(freeze=freeze_bn, name=f'fpn_cells/cell_{id}/fnode4/resample_0_1_9/bn')(P4_in_2)
P3_D = layers.MaxPooling2D(pool_size=3, strides=2, padding='same')(P3_out)
P4_out = layers.Add(name=f'fpn_cells/cell_{id}/fnode4/add')([P4_in_2, P4_td, P3_D])
P4_out = layers.Activation(lambda x: tf.nn.swish(x))(P4_out)
P4_out = SeparableConvBlock(num_channels=num_channels, kernel_size=3, strides=1,
                              name=f'fpn_cells/cell_{id}/fnode4/op_after_combine9')(P4_out)

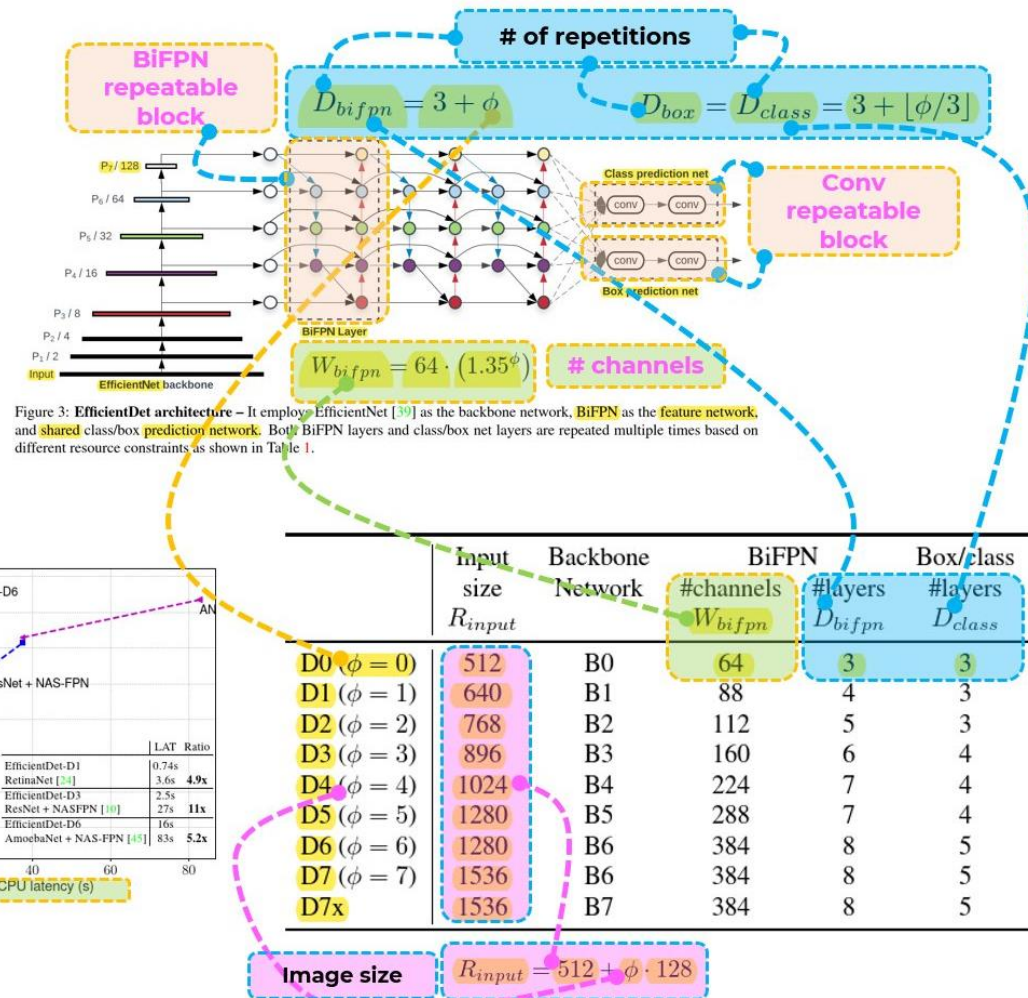
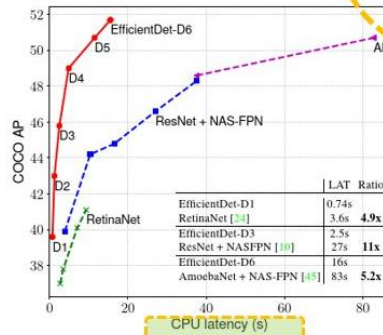
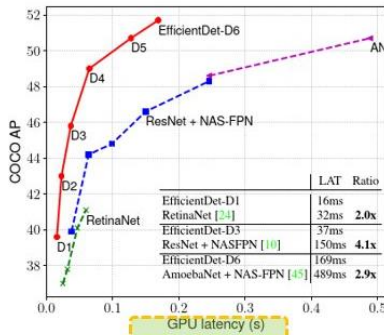
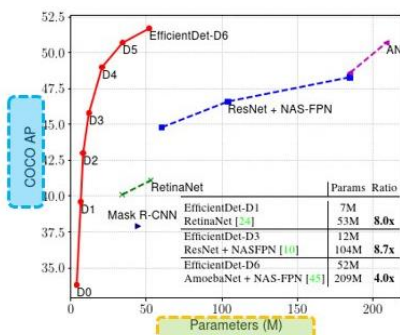
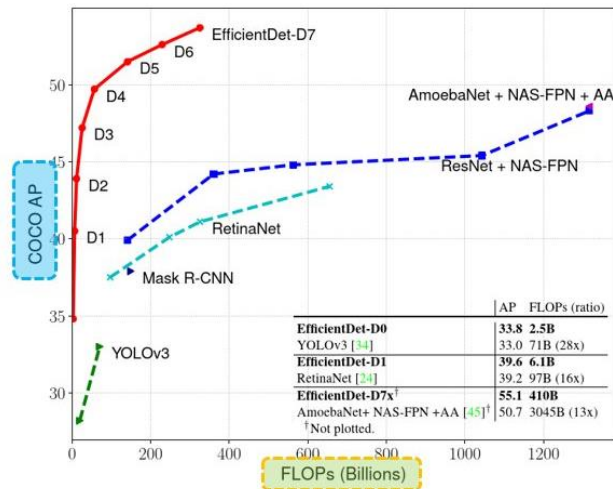
P5_in_2 = layers.Conv2D(num_channels, kernel_size=1, padding='same',
                          name=f'fpn_cells/cell_{id}/fnode5/resample_0_2_10/conv2d')(P5_in)
P5_in_2 = layers.BatchNormalization(momentum=MOMENTUM, epsilon=EPSILON,
                                     name=f'fpn_cells/cell_{id}/fnode5/resample_0_2_10/bn')(P5_in_2)
# P5_in_2 = BatchNormalization(freeze=freeze_bn, name=f'fpn_cells/cell_{id}/fnode5/resample_0_2_10/bn')(P5_in_2)
P4_D = layers.MaxPooling2D(pool_size=3, strides=2, padding='same')(P4_out)
P5_out = layers.Add(name=f'fpn_cells/cell_{id}/fnode5/add')([P5_in_2, P5_td, P4_D])
P5_out = layers.Activation(lambda x: tf.nn.swish(x))(P5_out)
P5_out = SeparableConvBlock(num_channels=num_channels, kernel_size=3, strides=1,
                              name=f'fpn_cells/cell_{id}/fnode5/op_after_combine10')(P5_out)

P5_D = layers.MaxPooling2D(pool_size=3, strides=2, padding='same')(P5_out)
P6_out = layers.Add(name=f'fpn_cells/cell_{id}/fnode6/add')([P6_in, P6_td, P5_D])
P6_out = layers.Activation(lambda x: tf.nn.swish(x))(P6_out)
P6_out = SeparableConvBlock(num_channels=num_channels, kernel_size=3, strides=1,
                              name=f'fpn_cells/cell_{id}/fnode6/op_after_combine11')(P6_out)

P6_D = layers.MaxPooling2D(pool_size=3, strides=2, padding='same')(P6_out)
P7_out = layers.Add(name=f'fpn_cells/cell_{id}/fnode7/add')([P7_in, P6_D])
P7_out = layers.Activation(lambda x: tf.nn.swish(x))(P7_out)
P7_out = SeparableConvBlock(num_channels=num_channels, kernel_size=3, strides=1,
                              name=f'fpn_cells/cell_{id}/fnode7/op_after_combine12')(P7_out)
```

04 프로젝트 수행 결과

결과 제시 ③ 모델 선정 및 분석



EfficientDet : Scalable and Efficient Object Detection

04 프로젝트 수행 결과

결과 제시 ④ 모델 평가 및 개선

▶ EfficientDet 모델 평가

EfficientDet은 기존의 Object Detection 모델들보다 **더욱 높은 정확도**와 **더 적은 파라미터 수**를 가진 모델입니다.
기존 모델들과 비교해 더 효율적인 성능을 보이고 있습니다.

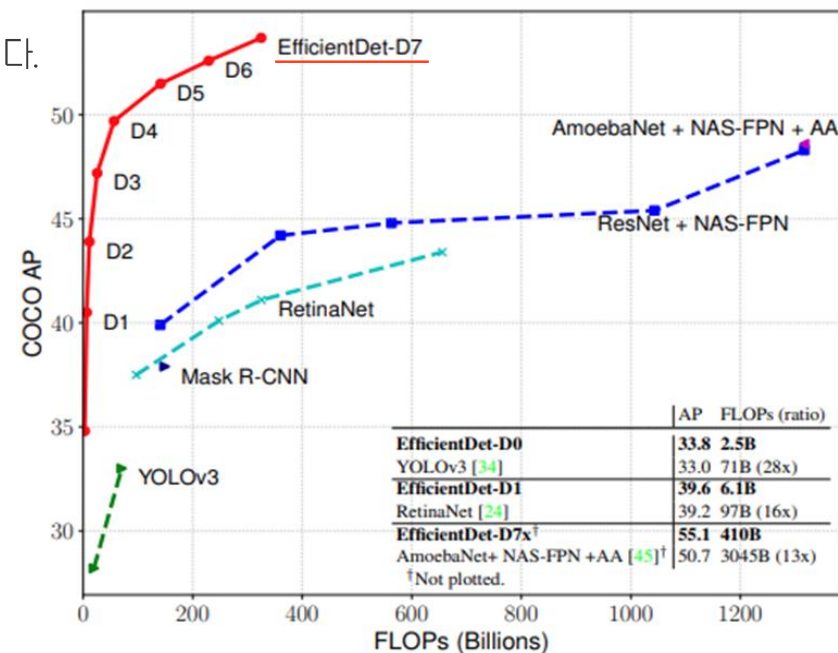
예를 들어, EfficientDet-D7 모델은 COCO 데이터셋에서 크기, 연산량이 작으면서 **55.1 AP(Average Precision)**를 달성했습니다.

이는 기존 모델인 Faster R-CNN, RetinaNet, SSD 등과 비교해 더 높은 정확도를 가지며, 파라미터 수가 적은 EfficientDet-D5 모델과 비교해 더욱 높은 성능을 보입니다.

예를 들어, EfficientDet-D4 모델은 RetinaNet-101 모델보다 빠르면서도 더 높은 정확도를 가집니다.
또한, EfficientDet은 기존 모델들과 비교해 속도가 빠른 편입니다.

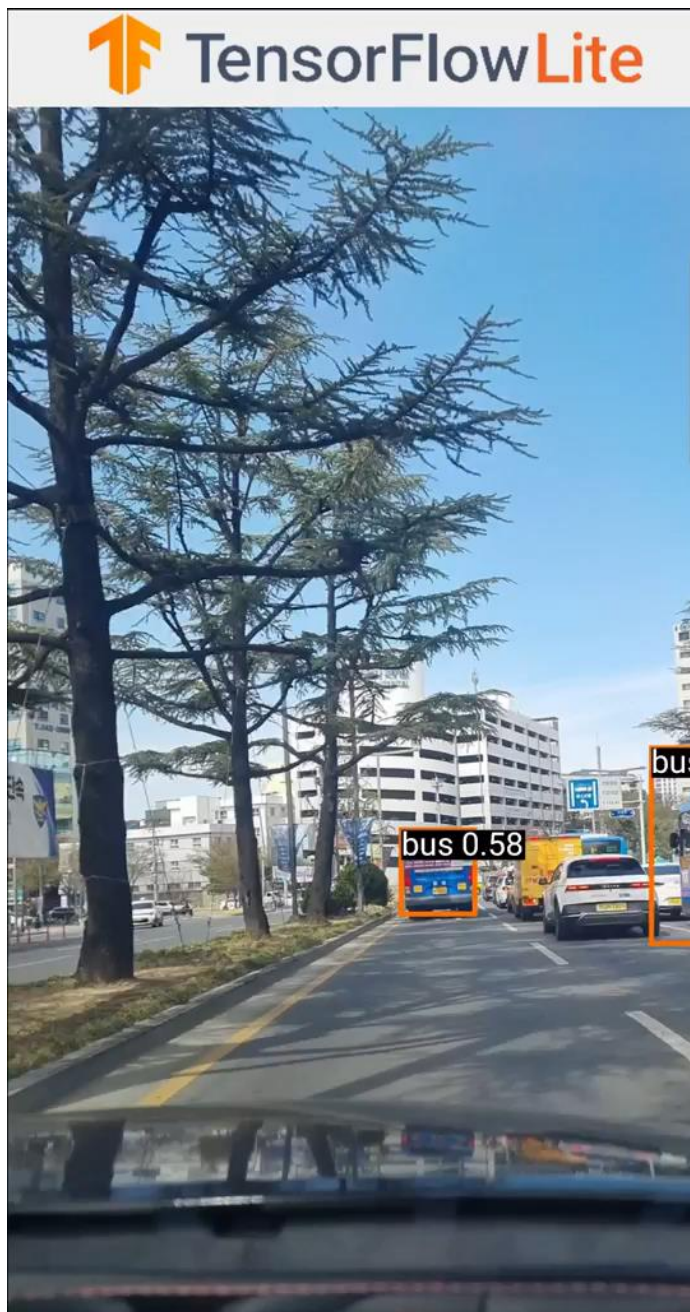
따라서, EfficientDet은 기존 모델들보다 더욱 효율적이며 정확도가 높은 모델로 평가될 수 있습니다.

Method	Backbone	Size	FPS	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L
EfficientDet-D5 [35]	EfficientNet-B5 [34]	1280	14*	51.5%	70.5%	56.7%	33.9%	54.7%	64.1%
ATSS [45]	R101-DCN [5]	800*	14	46.3%	64.7%	50.4%	27.7%	49.8%	58.4%
SABL [38]	R101 [11]	-	13	43.2%	62.0%	46.6%	25.7%	47.4%	53.9%
CenterMask [16]	V99-FPN [16]	-	13	46.5%	-	-	28.7%	48.9%	57.2%
EfficientDet-D6 [35]	EfficientNet-B6 [34]	1408	11*	52.6%	71.5%	57.2%	34.9%	56.0%	65.4%
RDSNet [39]	R101 [11]	800	11	38.1%	58.5%	40.8%	21.2%	41.5%	48.2%
RetinaNet [18]	S143 [6]	1280	10	50.7%	70.4%	54.9%	33.6%	53.9%	62.1%
SM-NAS: E5 [42]	-	1333*800	9.3	45.9%	64.6%	49.6%	27.1%	49.0%	58.0%
EfficientDet-D7 [35]	EfficientNet-B6 [34]	1536	8.2*	53.7%	72.4%	58.4%	35.8%	57.0%	66.3%
ATSS [45]	X-32x8d-101-DCN [5]	800*	7.0	47.7%	66.6%	52.1%	29.3%	50.8%	59.7%
ATSS [45]	X-64x4d-101-DCN [5]	800*	6.9	47.7%	66.5%	51.9%	29.7%	50.8%	59.4%
EfficientDet-D7x [35]	EfficientNet-B7 [34]	1536	6.5*	55.1%	74.3%	59.9%	37.2%	57.9%	68.0%
TSD [33]	R101 [11]	-	5.3*	43.2%	64.0%	46.9%	24.0%	46.3%	55.8%



04 프로젝트 수행 결과

결과 제시 ⑤ 시연 동영상



주행 경로