한국IT교육원

EfficientDet과 U-Net

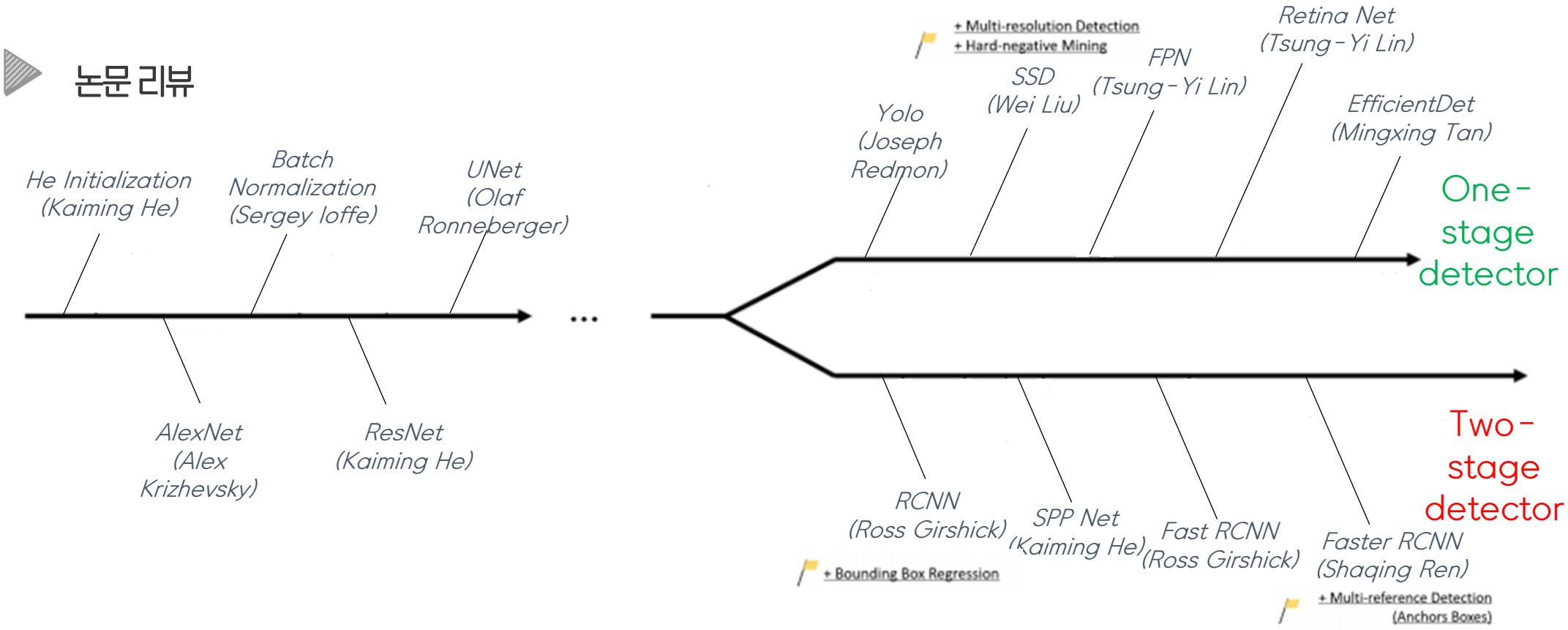
목차

00 프로젝트 수행을 위한 사전 스터디

프로그래밍 학습

- Python 및 TensorFlow, Keras 등 모델 생성 및 학습을 위한 프로그래밍 Tool 학습
- 사전 훈련된 모델을 사용하여 전이학습 및 Fine-Tuning 실습

논문 리뷰



프로젝트 팀 구성 및 역할

훈련생	역할	담당 업무
최현빈	팀장	• 모델 구현 • Data 수집 및 전 처리
김유연	팀원	• 모델 구현 • Data 수집 및 전 처리
김규현	팀원	• Data 수집 및 전 처리 • PPT 작성
윤도영	팀원	• 모델 구현
김형근	팀원	• 영상 기획 및 편집
조영재	팀원	• PPT 자료 작성
김희수	팀원	• 프로젝트 개요 작성
서동관	팀원	• 프로젝트 영상 검토

03 프로젝트 수행 절차 및 방법

구분	기간	활동	비고
프로젝트 기획	2/8 (수) ~ 2/10 (금)	• **프로젝트 기획 및 Model 선정** • **Dataset 선정**	• Model 선정 • 역할 분담 • 학습 Dataset 선정
데이터 수집 및 전 처리	2/13 (월) ~ 2/24 (금)	• **Dataset 및 각 종 Data 수집** • **모델 학습을 위한 전 처리**	• Dataset 수집 및 선별 같은 포멧으로 변환
두가지 모델 설계 및 구현	2/27 (월) ~ 3/6 (월)	• **EfficientDet 과 U-NET 모델 설계 및 구현**	• Model 구현
두가지모델 학습 및 Test	3/7 (화) ~ 3/16 (목)	• **Dataset을 이용한 모델 Training** • **학습 모델 Test 및 수정**	• 자체 중간평가 • Test 및 수정
결과 분석	3/17 (금)	• **모델 학습 데이터 및 결과 분석**	• 결과 분석
총 개발기간	2/8 (수) ~ 3/17 (금) (총 5주)		

1. 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 기반으로 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

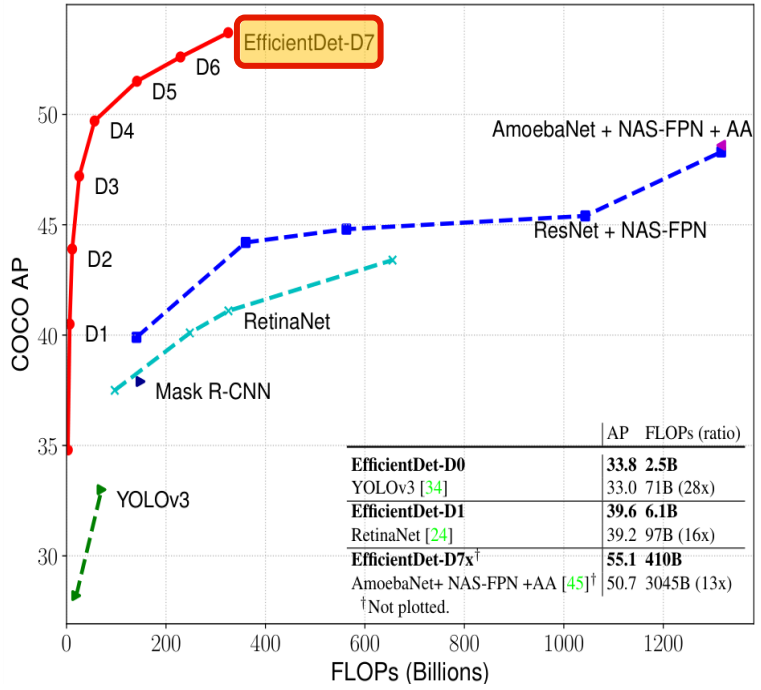
기획 의도 1

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

기존의 Object Detection 모델들보다 더욱 높은 정확도를 보여주면서도,

더욱 적은 연산량과 파라미터 수를 가지고 있어서 보다 **효율적인 모델 구현**이 가능합니다.

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



01 프로젝트 개요

기획 의도 2

- 최근 자율주행 차량에 대한 수요와 관심이 폭발적으로 증가하고 있으며, 완성차 업체마다 자율주행 기능이 탑재된 전기차를 경쟁적으로 출시
- 정부는 2025년까지 운전자 개입이 없는 완전 자율주행(Level 4)버스와 택시를, 2027년까지는 승용차를 출시하겠다는 계획안 발표
- 지난 수년간 자율주행에 필수적인 정확한 객체 인식(Object detection)을 위해 수많은 딥러닝 모델 (RCNN, Yolo, SSD 등) 및 알고리즘들이 제시 되어 왔음.
- U-Net은 생체의학 분야에서 이미지 분할(image segmentation)을 목적으로 제안된 모델로 자율주행에서 정확한 객체 인식을 위해 사용되어질 수 있음.

01 프로젝트 개요

기대 효과

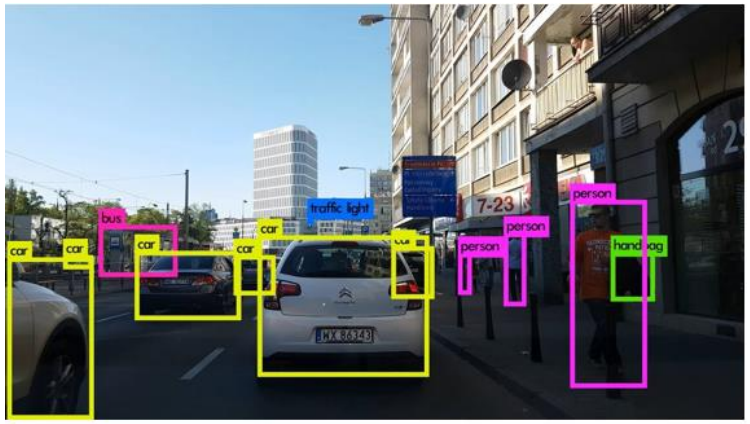
- EfficientDet 모델은 Object Detection 분야 뿐만 아니라,
  다양한 분야에서 응용 가능합니다.

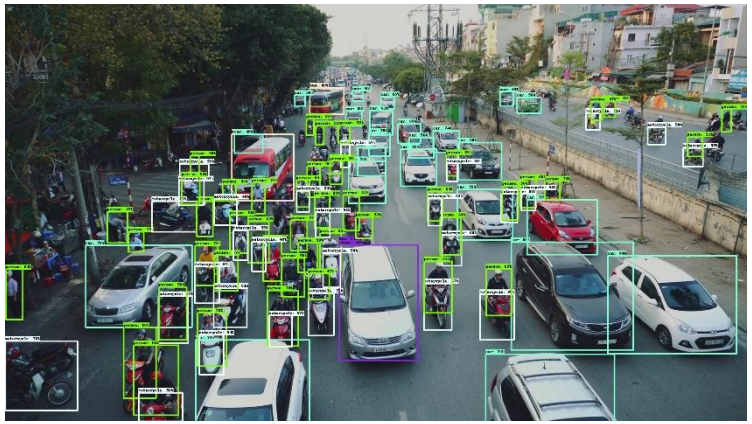
  예를 들어 자율주행, 보안 감시, 영상 인식 등에서 활용될 수 있습니다.

  자율주행차에서는 차량 및 보행자 등의 객체 검출에 사용되며,

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

- U-Met U-Net의 이미지 분할효과를 통해 더욱 정확하고 안전한
  자율주행 알고리즘 및 모델 구현에 도움을 줄 것으로 기대 할 수 있습니다.





출처: https://hoya012.github.io/blog/Tutorials-of-Object-Detection-Using-Deep-Learning-the-application-of-object-detection/, https://github.com/zylo117/Yet-Another-EfficientDet-Pytorch

01 프로젝트 개요

▶ 프로젝트 구조

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. **EfficientDet**은 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**

01 프로젝트 개요

활용 장비 및 재료(개발 환경 등)

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

03 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)


```xml
<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>
```

XML Format

```json
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
#                       }
#               }, ...
#       ]
# }
```

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.1bd393	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

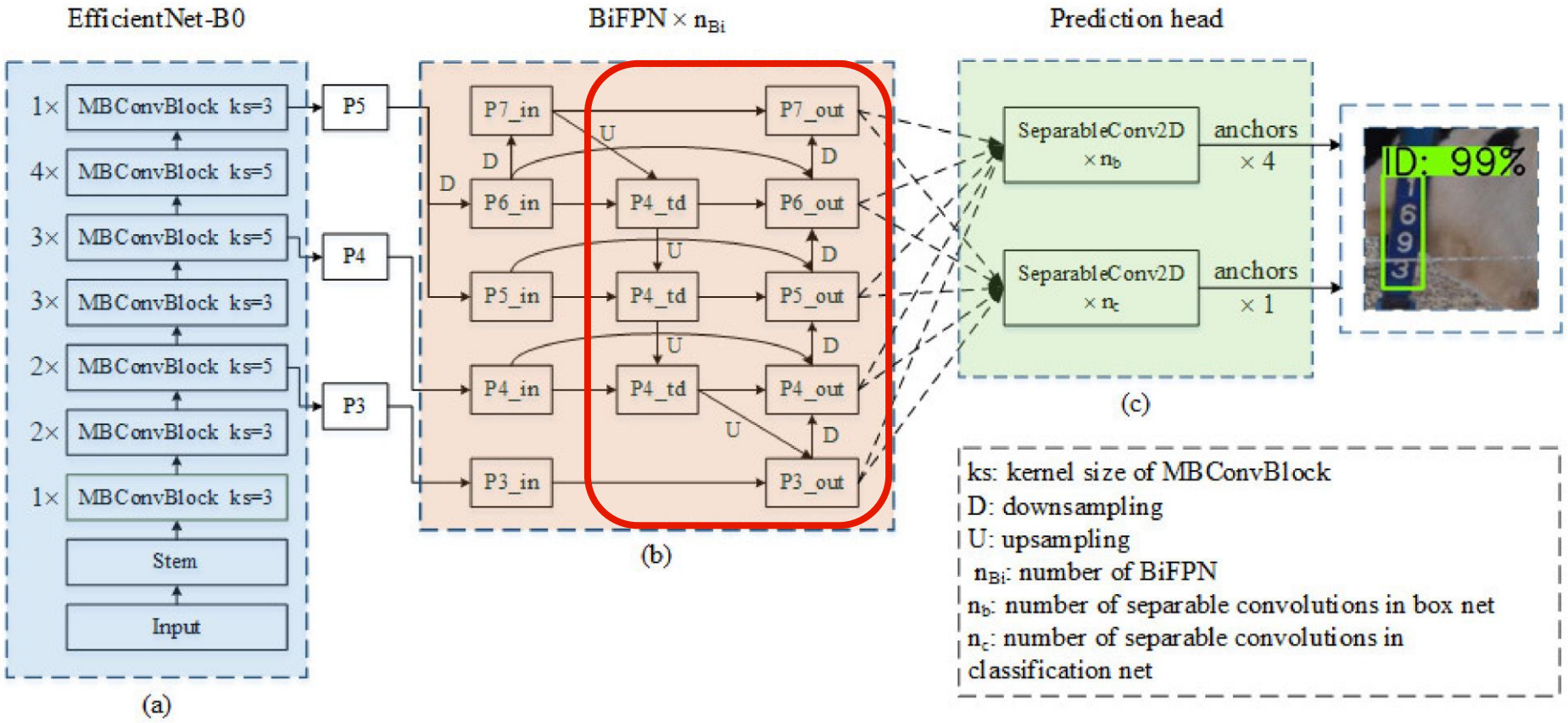
CSV Format

Datasets 출처: [https://aihub.or.kr], [https://universe.roboflow.com/]

03 프로젝트 수행 결과

결과 제시 ② 모델 개요

EfficientDet 모델 구조

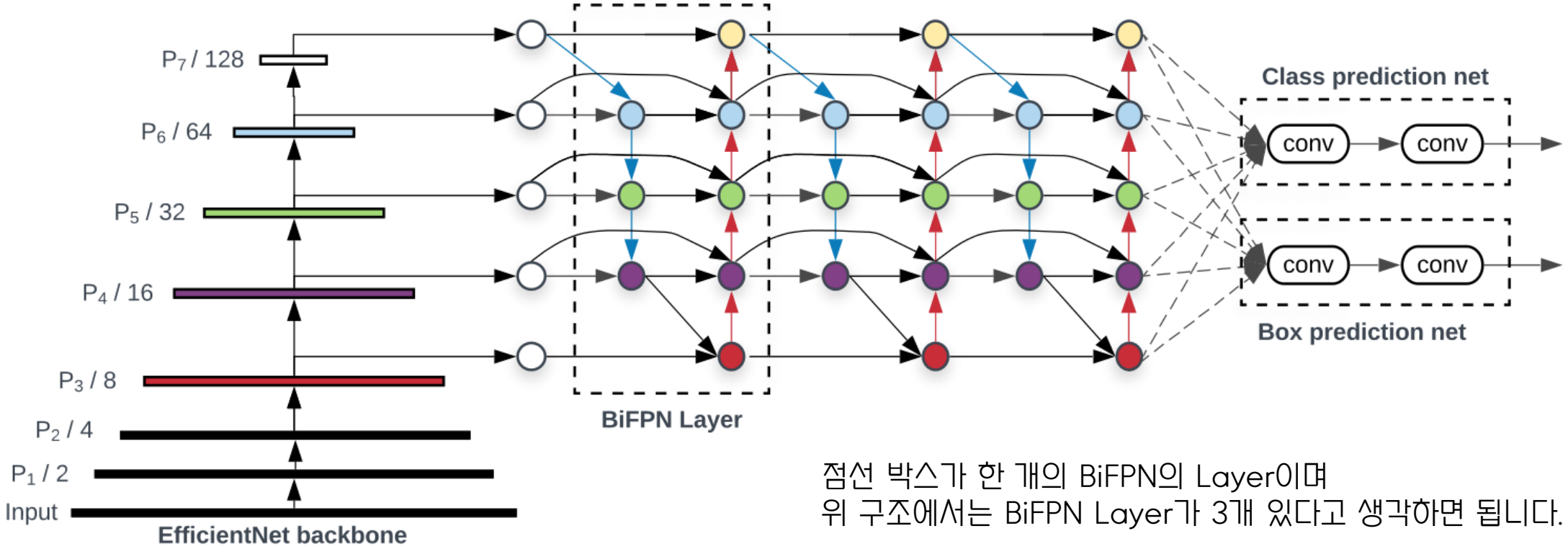


출처: EfficientDet: Scalable and Efficient Object Detection paper

03 프로젝트 수행 결과

결과 제시 ③ 모델 선정 및 분석

EfficientDet 분석



점선 박스가 한 개의 BiFPN의 Layer이며
위 구조에서는 BiFPN Layer가 3개 있다고 생각하면 됩니다.

출처: EfficientDet for fabric defect detection based on edge computing

03 프로젝트 수행 결과

결과 제시 ③ 모델 선정 및 분석

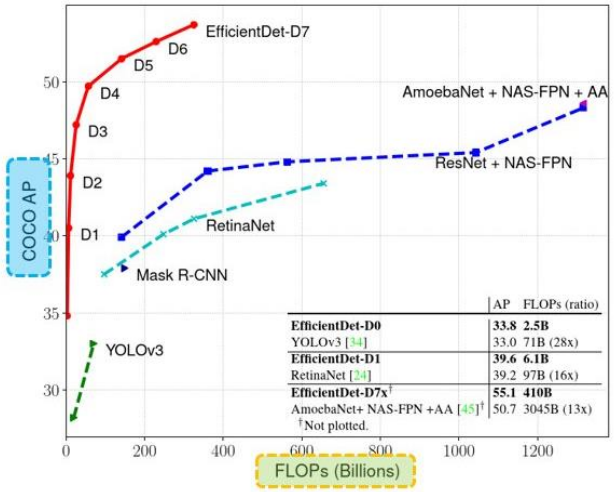
EfficientDet 코드 분석

```
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)
```

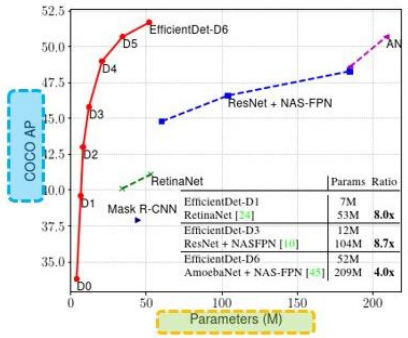
03 프로젝트 수행 결과

결과 제시 ③ 모델 선정 및 분석

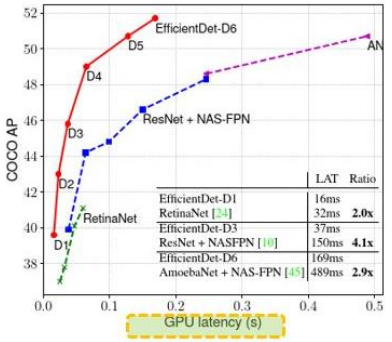


	AP	FLOPs (ratio)
EfficientDet-D0	**33.8**	**2.5B**
YOLOv3 [34]	33.0	71B (28x)
EfficientDet-D1	**39.6**	**6.1B**
RetinaNet [24]	39.2	97B (16x)
EfficientDet-D7x†	**55.1**	**410B**
AmoebaNet+ NAS-FPN +AA [45]†	50.7	3045B (13x)

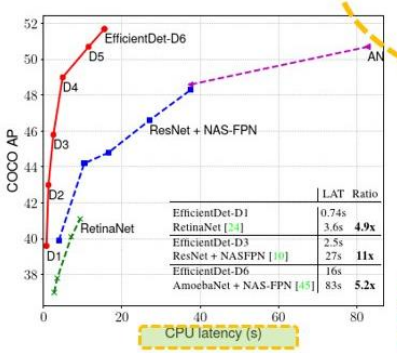
†Not plotted.



	Params	Ratio
EfficientDet-D1	7M	
RetinaNet [24]	53M	**8.0x**
EfficientDet-D3	12M	
ResNet + NASFPN [10]	104M	**8.7x**
EfficientDet-D6	52M	
AmoebaNet + NAS-FPN [45]	209M	**4.0x**



	LAT	Ratio
EfficientDet-D1	16ms	
RetinaNet [24]	32ms	**2.0x**
EfficientDet-D3	37ms	
ResNet + NASFPN [10]	150ms	**4.1x**
EfficientDet-D6	169ms	
AmoebaNet + NAS-FPN [45]	489ms	**2.9x**



	LAT	Ratio
EfficientDet-D1	0.74s	
RetinaNet [24]	3.6s	**4.9x**
EfficientDet-D3	2.5s	
ResNet + NASFPN [10]	27s	**11x**
EfficientDet-D6	16s	
AmoebaNet + NAS-FPN [45]	83s	**5.2x**

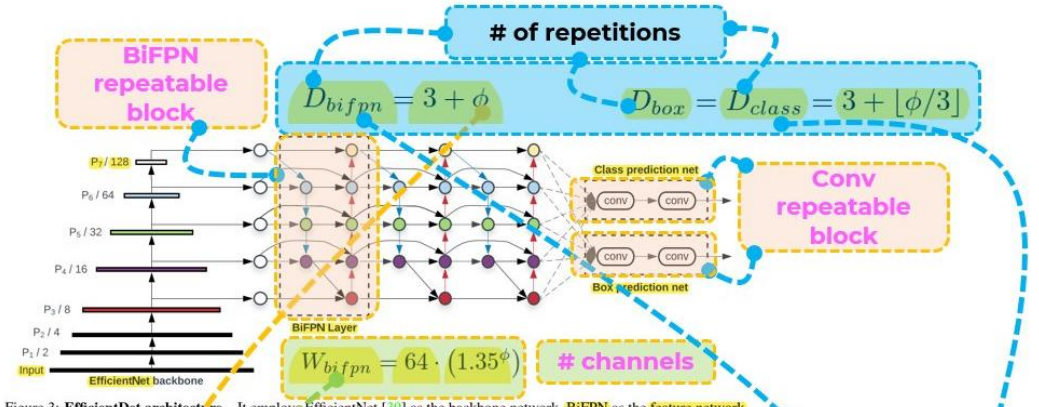


Figure 3: **EfficientDet architecture –** It employ EfficientNet [39] as the backbone network, BiFPN as the feature network, and shared class/box prediction network. Both BiFPN layers and class/box net layers are repeated multiple times based on different resource constraints as shown in Table 1.

	Input size R_{input}	Backbone Network	BiFPN #channels W_{bifpn}	BiFPN #layers D_{bifpn}	Box/class #layers D_{class}
D0 ($\phi = 0$)	512	B0	64	3	3
D1 ($\phi = 1$)	640	B1	88	4	3
D2 ($\phi = 2$)	768	B2	112	5	3
D3 ($\phi = 3$)	896	B3	160	6	4
D4 ($\phi = 4$)	1024	B4	224	7	4
D5 ($\phi = 5$)	1280	B5	288	7	4
D6 ($\phi = 6$)	1280	B6	384	8	5
D7 ($\phi = 7$)	1536	B6	384	8	5
D7x	1536	B7	384	8	5

Image size $R_{input} = 512 + \phi \cdot 128$

EfficientDet : Scalable and Efficient Object Detection

출처: https://pbs.twimg.com/media/FE-m3wgXEAM24As?format=jpg&name=large

03 프로젝트 수행 결과

결과 제시 ④ 모델 평가 및 개선

EfficientDet 모델 평가

EfficientDet은 기존의 Object Detection 모델들보다 더욱 높은 정확도와 더 적은 파라미터 수를 가진 모델입니다.
기존 모델들과 비교해 더 효율적인 성능을 보이고 있습니다.

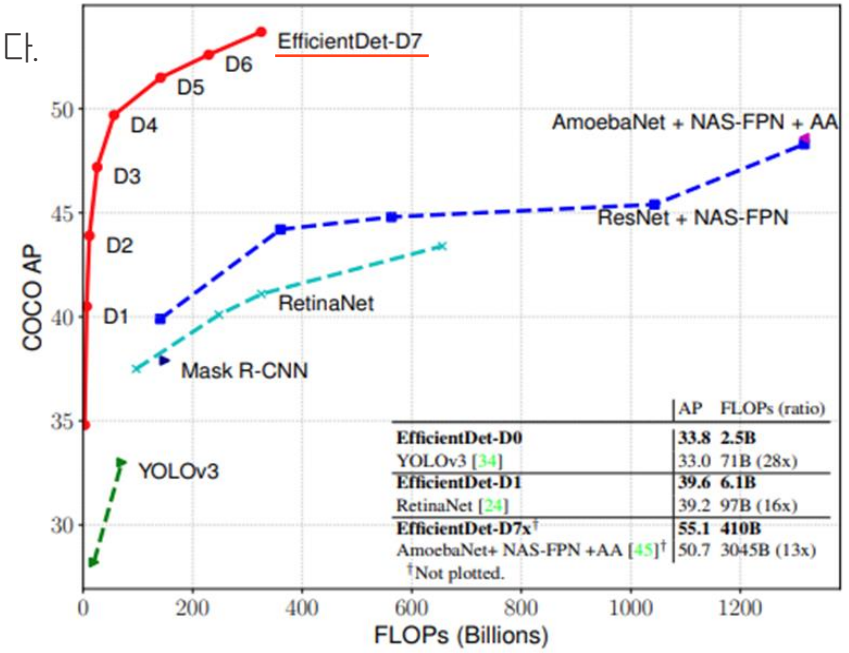
예를 들어, EfficientDet-D7 모델은 COCO 데이터셋에서 크기, 연산량이 작으면서 55.1 AP(Average Precision)를 달성했습니다.

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.

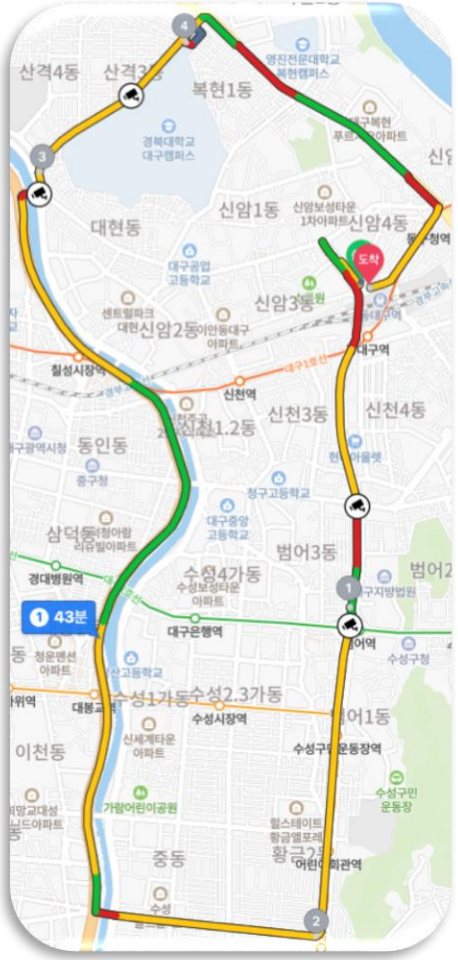
Method	Backbone	Size	FPS	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
EfficientDet-D5 [35]	EfficientNet-B5 [34]	1280	14*	51.5%	70.5%	56.7%	33.9%	54.7%	64.1%
ATSS [45]	R101-DCN [5]	800*	14	46.3%	64.7%	50.4%	27.7%	49.8%	58.4%
SABL [38]	R101 [11]	-	13	43.2%	62.0%	46.6%	25.7%	47.4%	53.9%
CenterMask [16]	V99-FPN [16]	-	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%



출처: EfficientDet: Scalable and Efficient Object Detection paper

03 프로젝트 수행 결과

결과 제시 ⑤ 시연 동영상

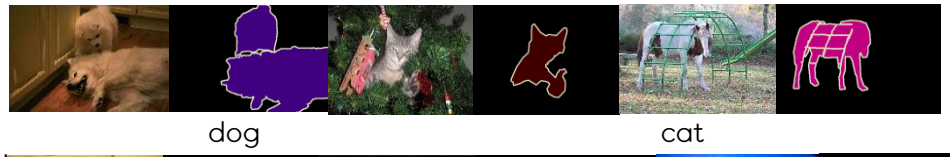


주행 경로

2. U-NET

01 프로젝트 수행 결과

학습 데이터 소개(Training)


dog cat


horse tv/monitor boat


bus train bottle airplane person


bird cow sofa dining table

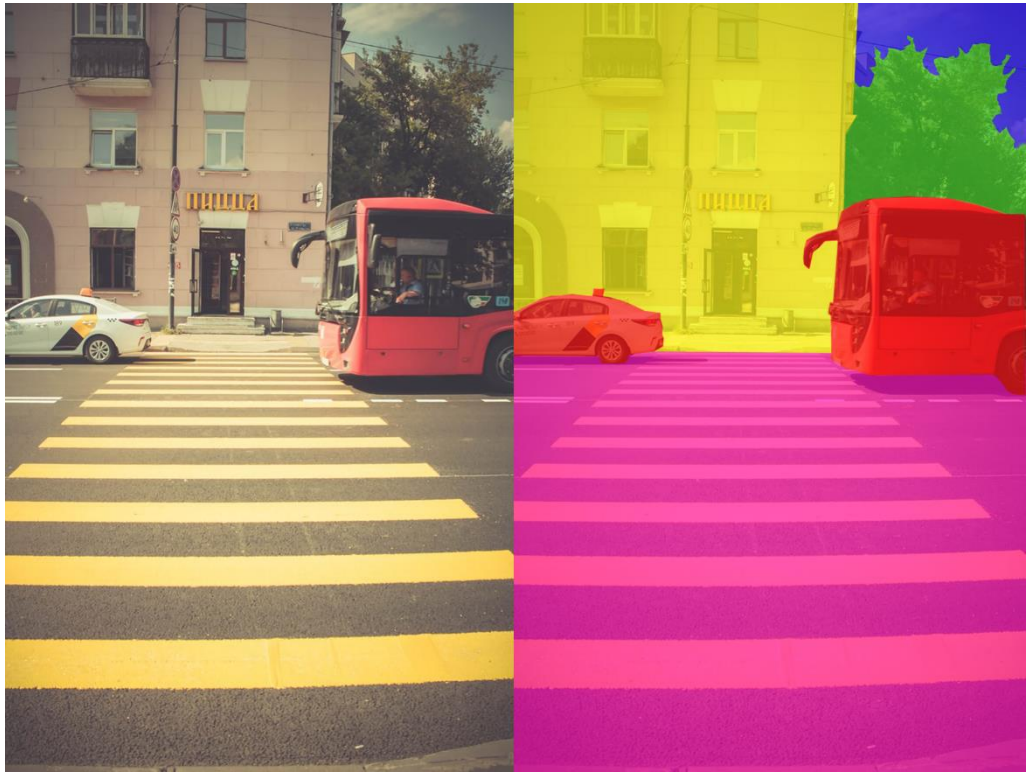

sheep bicycle potted plant motorbike car

사용 데이터셋 : VOCtrainval_11-May-2012

- Person : person
- Animal : bird, cat, cow, dog, horse, sheep
- Vehicle : airplane, bicycle, boat, bus, car, motorbike, train
- Indoor : bottle, chair, dining table, potted plant, sofa, tv/monitor

〈총 17,125개의 Data〉

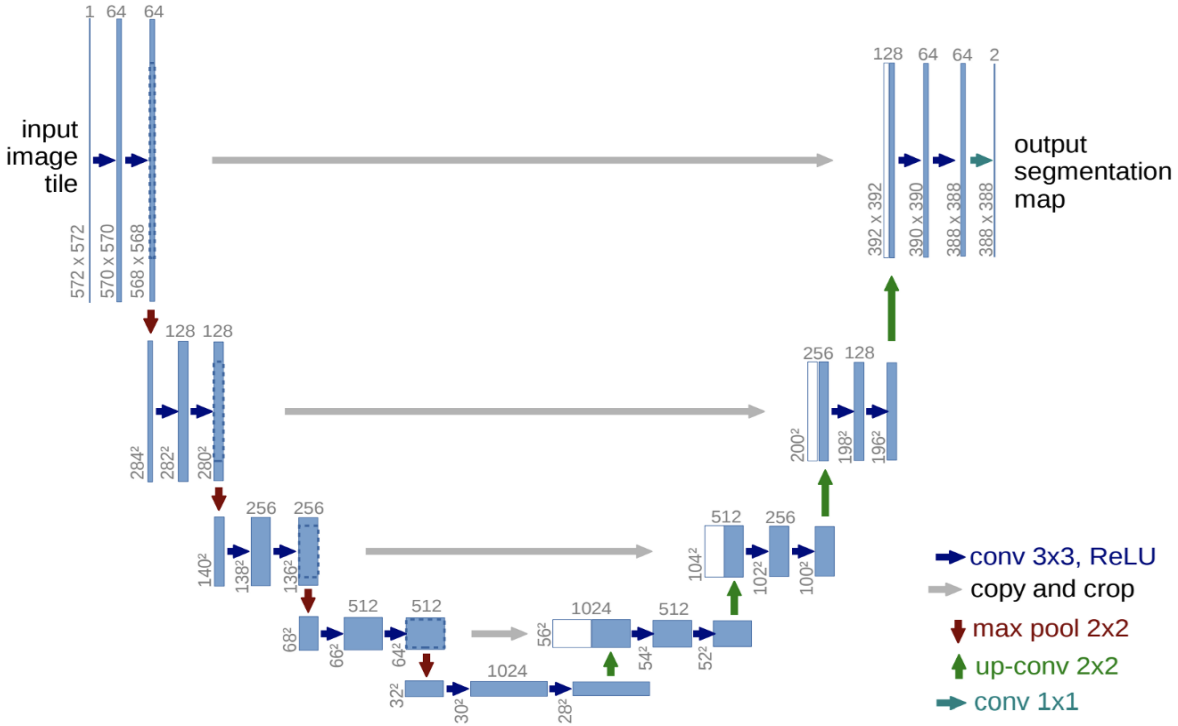
Image semantic segmentation

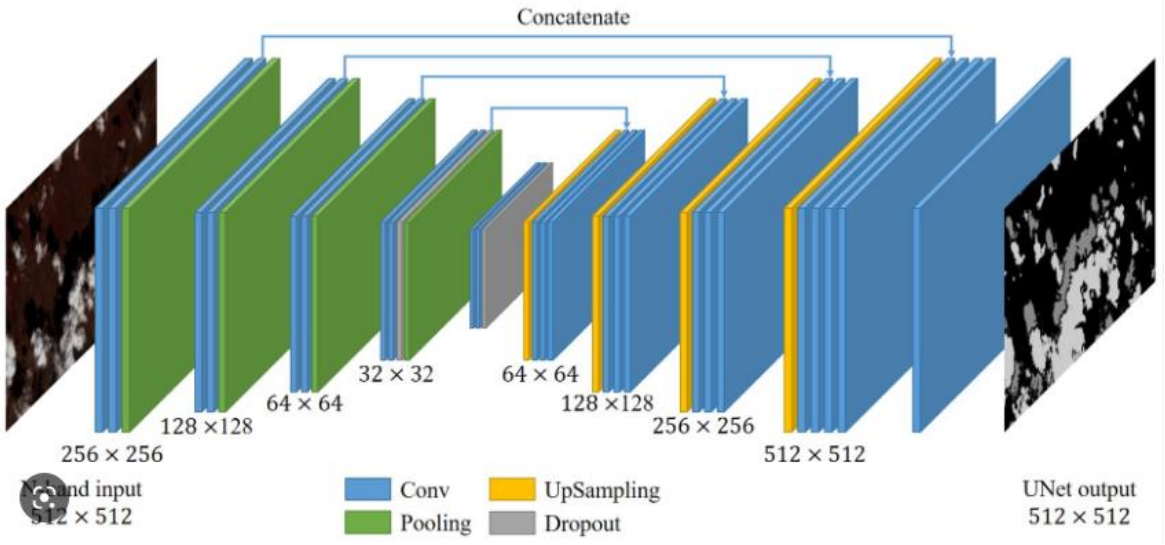


[출처] http://datagen.tech/guides/image-annotation/image-segmentation/#

[출처] https://thegradient.pub/semantic-segmentation

U-Net모델의 구조

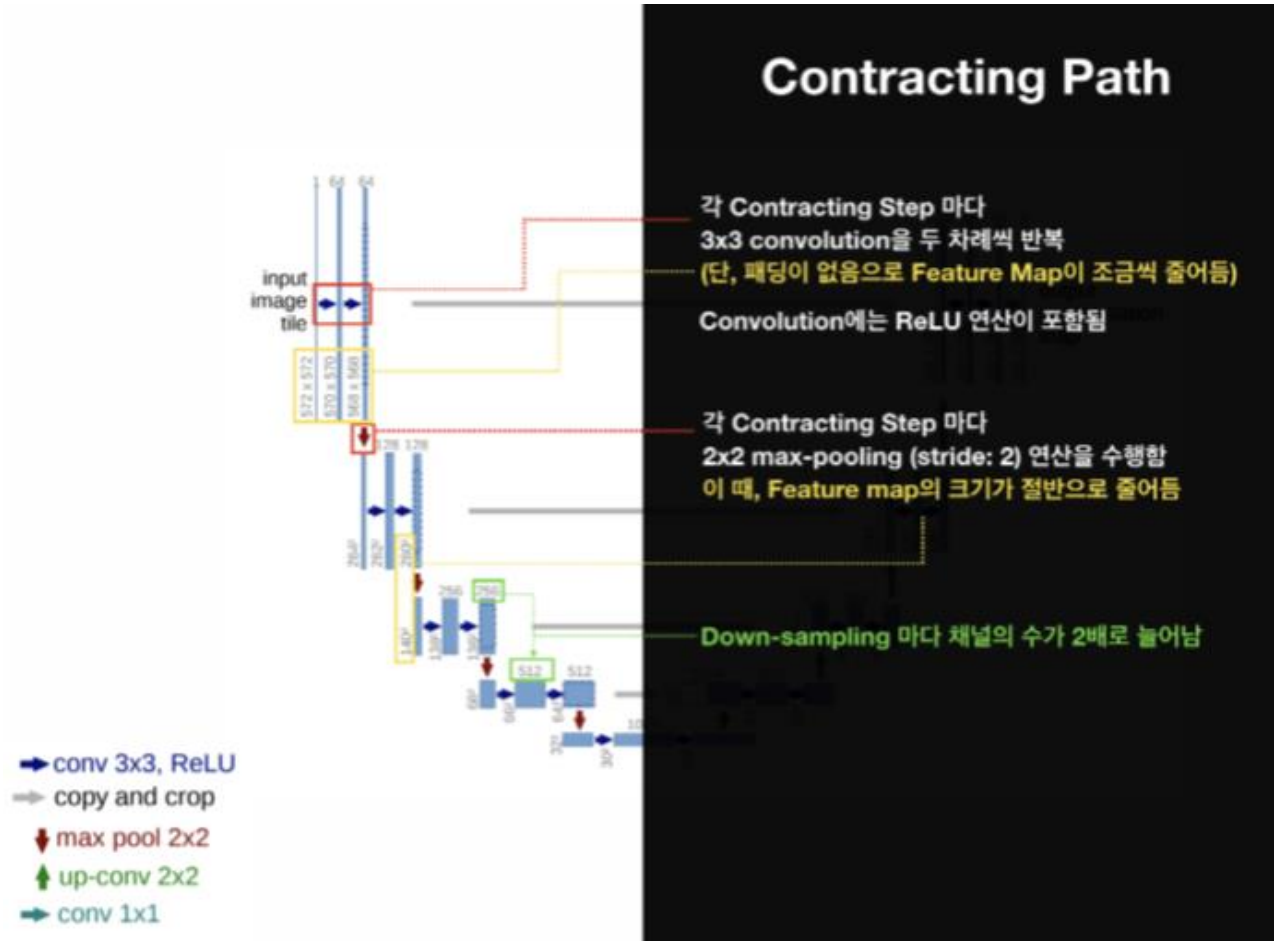




[출처] https://www.mdpi.com/2072-4292/12/12/2001

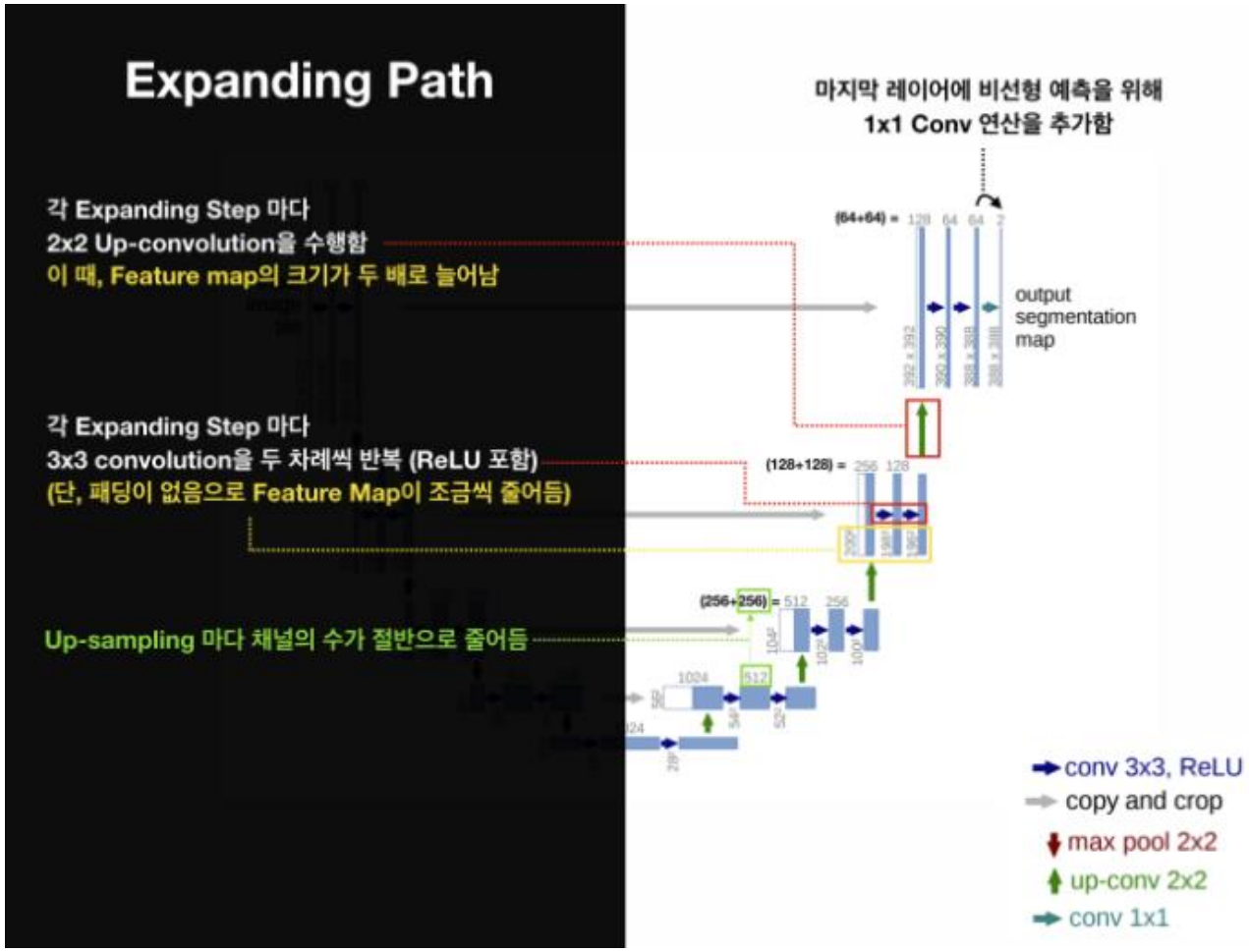
[출처] https://medium.com/@msmapark2/u-net-%EB%85%BC%EB%AC%B8-%EB%A6%AC%EB%B7%B0-u-net-convolutional-networks-for-biomedical-image-segmentation-456d6901b28a

U-Net모델의 구조



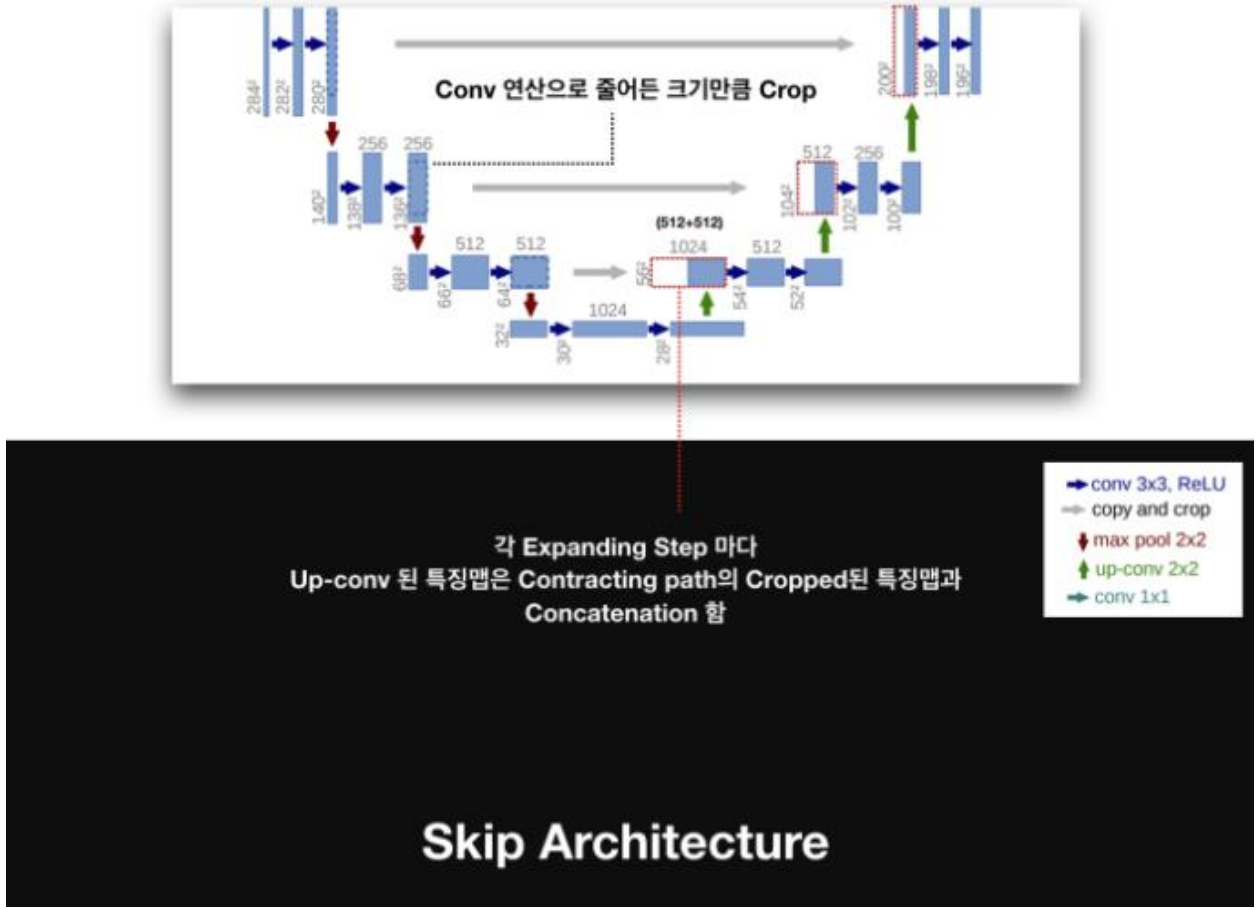
[출처] https://medium.com/@msmapark2/u-net-%EB%85%BC%EB%AC%B8-%EB%A6%AC%EB%B7%B0-u-net-convolutional-networks-for-biomedical-image-segmentation-456d6901b28a

U-Net모델의 구조



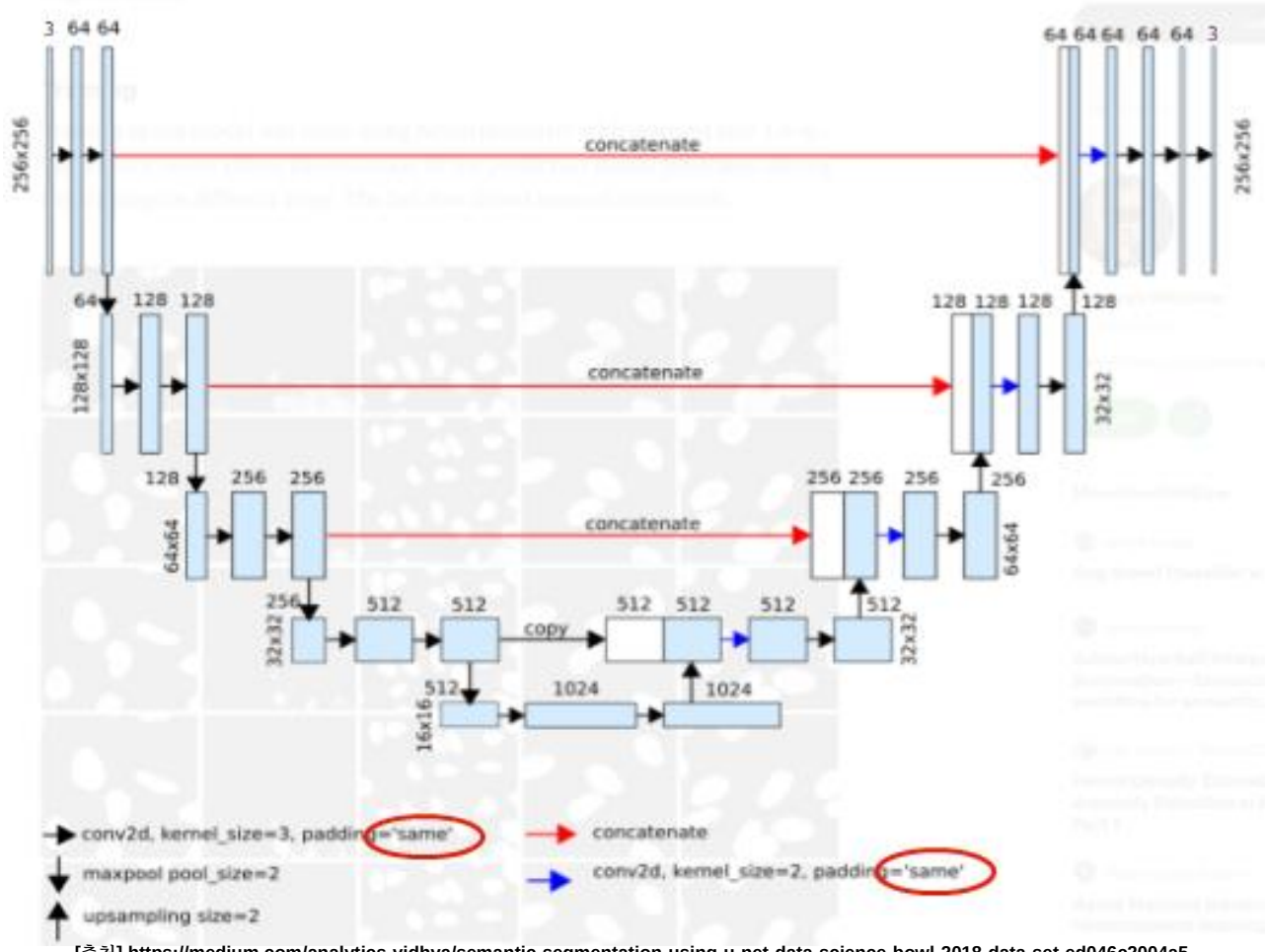
[출처] https://medium.com/@msmapark2/u-net-%EB%85%BC%EB%AC%B8-%EB%A6%AC%EB%B7%B0-u-net-convolutional-networks-for-biomedical-image-segmentation-456d6901b28a

U-Net모델의 구조



[출처] https://medium.com/@msmapark2/u-net-%EB%85%BC%EB%AC%B8-%EB%A6%AC%EB%B7%B0-u-net-convolutional-networks-for-biomedical-image-segmentation-456d6901b28a

프로젝트 구현 U-Net 모델(“same” convolution 적용)



[출처] https://medium.com/analytics-vidhya/semantic-segmentation-using-u-net-data-science-bowl-2018-data-set-ed046c2004a5

U-Net 모델 구현

```
import tensorflow as tf
import numpy as np

from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Conv2DTranspose
from tensorflow.keras.layers import concatenate

from test_utils import summary, comparator
```

Import module1

```
import os
import numpy as np
import pandas as pd

import imageio

import matplotlib.pyplot as plt
%matplotlib inline
```

Import module2

U-Net 모델 구현

```
path = ''
image_path = os.path.join(path, '/content/gdrive/MyDrive/Colab Notebooks/coursera programming practice/class4/week3/Image Segmentation with UNet/data/CameraRGB/')
mask_path = os.path.join(path, '/content/gdrive/MyDrive/Colab Notebooks/coursera programming practice/class4/week3/Image Segmentation with UNet/data/CameraMask/')
image_list = sorted(os.listdir(image_path)) #listdir : 경로에 있는 모든 파일명으로 리스트를 만듬
mask_list = sorted(os.listdir(mask_path))
image_list = sorted([image_path+i for i in image_list])
mask_list = sorted([mask_path+i for i in mask_list])
```

```
image_filenames = tf.constant(image_list) #image_list라는 리스트를 image_filenames라는 이름의 1차원 텐서로 만듬
masks_filenames = tf.constant(mask_list)

dataset = tf.data.Dataset.from_tensor_slices((image_filenames, masks_filenames)) #주어진 데이터소스를 여러 텐서로 자른다.

for image, mask in dataset.take(1):
    print(image)
    print(mask)
```

Dataset 객체 생성

U-Net 모델 구현

```python
def process_path(image_path, mask_path):
    img = tf.io.read_file(image_path)
    img = tf.image.decode_png(img, channels=3) #png image파일을 tensor로 변환
    img = tf.image.convert_image_dtype(img, tf.float32) #0~1값으로 normalize
    print("img.shape=", img.shape)

    mask = tf.io.read_file(mask_path)
    mask = tf.image.decode_png(mask, channels=3)
    print("mask.shape=", mask.shape)
    mask = tf.math.reduce_max(mask, axis=-1, keepdims=True)
    print("mask.shape=", mask.shape)
    return img, mask
```

```python
def preprocess(image, mask):
    input_image = tf.image.resize(image, (96, 128), method='nearest')
    input_mask = tf.image.resize(mask, (96, 128), method='nearest')

    return input_image, input_mask
```

```python
image_ds = dataset.map(process_path)

processed_image_ds = image_ds.map(preprocess)
```

학습에 사용 될 dataset 생성

U-Net 모델 구현

```
def conv_block(inputs=None, n_filters=32, dropout_prob=0, max_pooling=True):

    conv = Conv2D(n_filters,
                  3,
                  activation='relu',
                  padding='same',
                  kernel_initializer='he_normal')(inputs)
    conv = Conv2D(n_filters,
                  3,
                  activation='relu',
                  padding='same',
                  kernel_initializer='he_normal')(conv)

    if dropout_prob > 0:
        conv = Dropout(dropout_prob)(conv)

    if max_pooling:
        next_layer = MaxPooling2D(2,strides=2)(conv)

    else:
        next_layer = conv

    skip_connection = conv

    return next_layer, skip_connection
```

Contracting Block

```
def upsampling_block(expansive_input, contractive_input, n_filters=32):

    up = Conv2DTranspose(
                 n_filters,
                 3,
                 strides=2,
                 padding='same')(expansive_input)

    print("up=", up)
    print("contractive_input=", contractive_input)

    merge = concatenate([up, contractive_input], axis=3)
    print("merge=", merge)
    conv = Conv2D(n_filters,
                  3,
                  activation='relu',
                  padding='same',
                  kernel_initializer='he_normal')(merge)
    conv = Conv2D(n_filters,
                  3,
                  activation='relu',
                  padding='same',
                  kernel_initializer='he_normal')(conv)

    return conv
```

Expanding Block

U-Net 모델 구현

```
def unet_model(input_size=(96, 128, 3), n_filters=32, n_classes=23):

    inputs = Input(input_size)

    cblock1 = conv_block(inputs=inputs, n_filters=n_filters*1)
    cblock2 = conv_block(inputs=cblock1[0], n_filters=n_filters*2)
    cblock3 = conv_block(inputs=cblock2[0], n_filters=n_filters*4)
    cblock4 = conv_block(inputs=cblock3[0], n_filters=n_filters*8,dropout_prob=0.3)
    cblock5 = conv_block(inputs=cblock4[0], n_filters=n_filters*16,dropout_prob=0.3, max_pooling=False)

    ublock6 = upsampling_block(cblock5[0], cblock4[1], n_filters*8)
    ublock7 = upsampling_block(ublock6, cblock3[1], n_filters*4)
    ublock8 = upsampling_block(ublock7, cblock2[1], n_filters*2)
    ublock9 = upsampling_block(ublock8, cblock1[1], n_filters*1)

    conv9 = Conv2D(n_filters,
                3,
                activation='relu',
                padding='same',
                kernel_initializer='he_normal')(ublock9)

    conv10 = Conv2D(n_classes, 1, padding='same')(conv9)

    model = tf.keras.Model(inputs=inputs, outputs=conv10)

    return model
```

전체 모델 구조

```
img_height = 96
img_width = 128
num_channels = 3

unet = unet_model((img_height, img_width, num_channels))
```

모델 생성

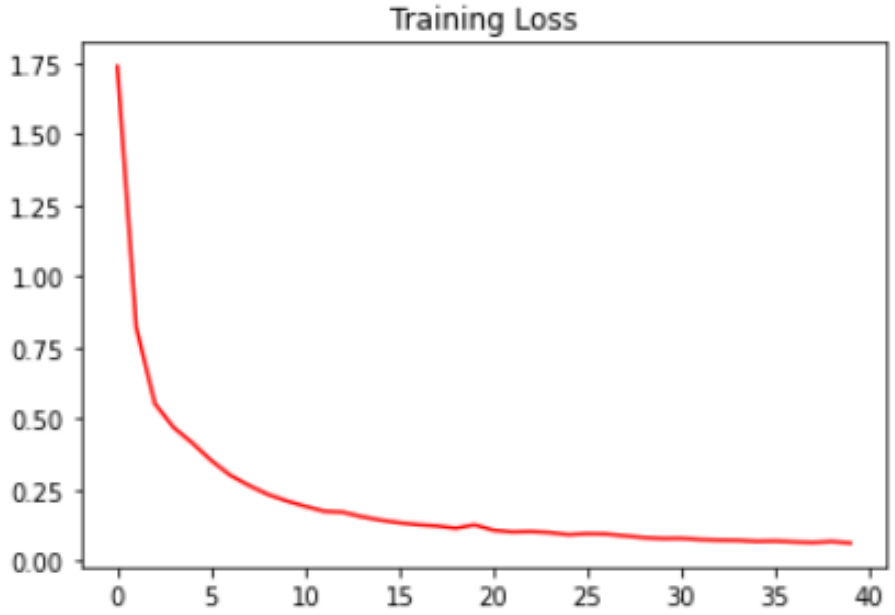
U-Net 모델 구현

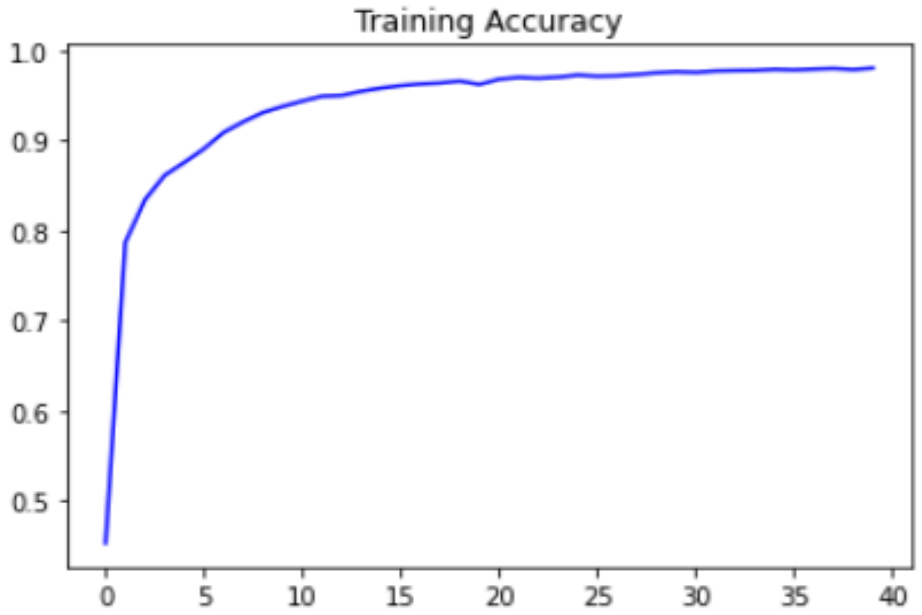
```
unet.compile(optimizer='adam',
             loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
             metrics=['accuracy'])
```

```
EPOCHS = 40
VAL_SUBSPLITS = 5
BUFFER_SIZE = 500
BATCH_SIZE = 32
processed_image_ds.batch(BATCH_SIZE)
train_dataset = processed_image_ds.cache().shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
print(processed_image_ds.element_spec)
model_history = unet.fit(train_dataset, epochs=EPOCHS)
```

모델 학습

U-Net 모델 구현

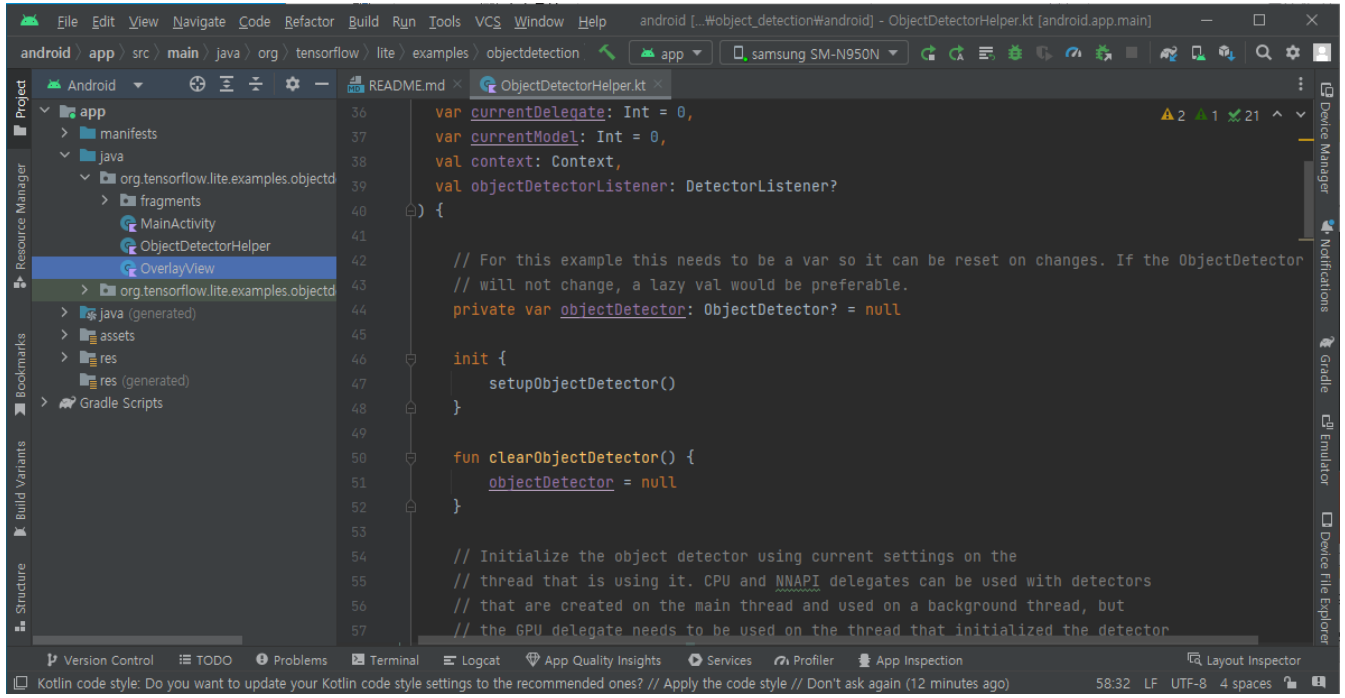

Training Loss
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
5
10
15
20
25
30
35
40

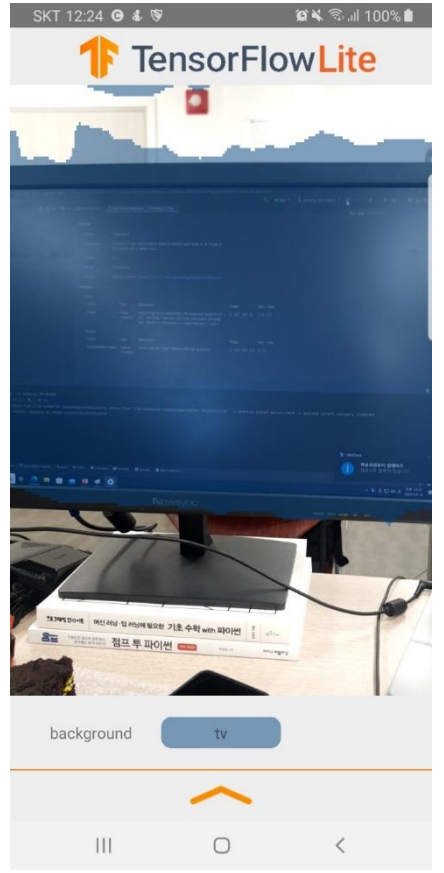

Training Accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0
5
10
15
20
25
30
35
40

04 프로젝트 수행 결과

Tensorflow Lite 모델 생성 및 Android studio 빌드

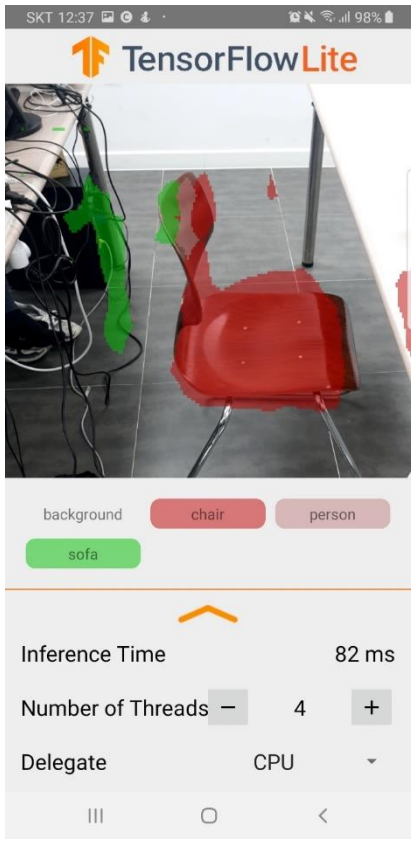
```
converter = tf.lite.TFLiteConverter.from_keras_model(unet)
tflite_model = converter.convert()

with open('unetmodel.tflite', 'wb') as f:
  f.write(tflite_model)
```

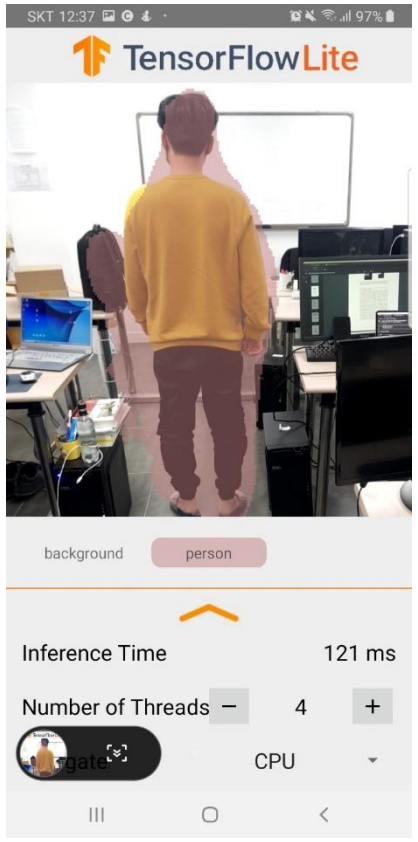


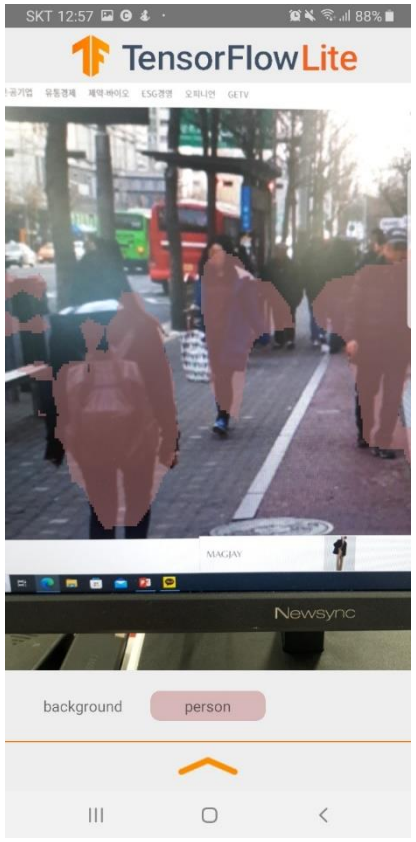


04 프로젝트 수행 결과

테스트 결과





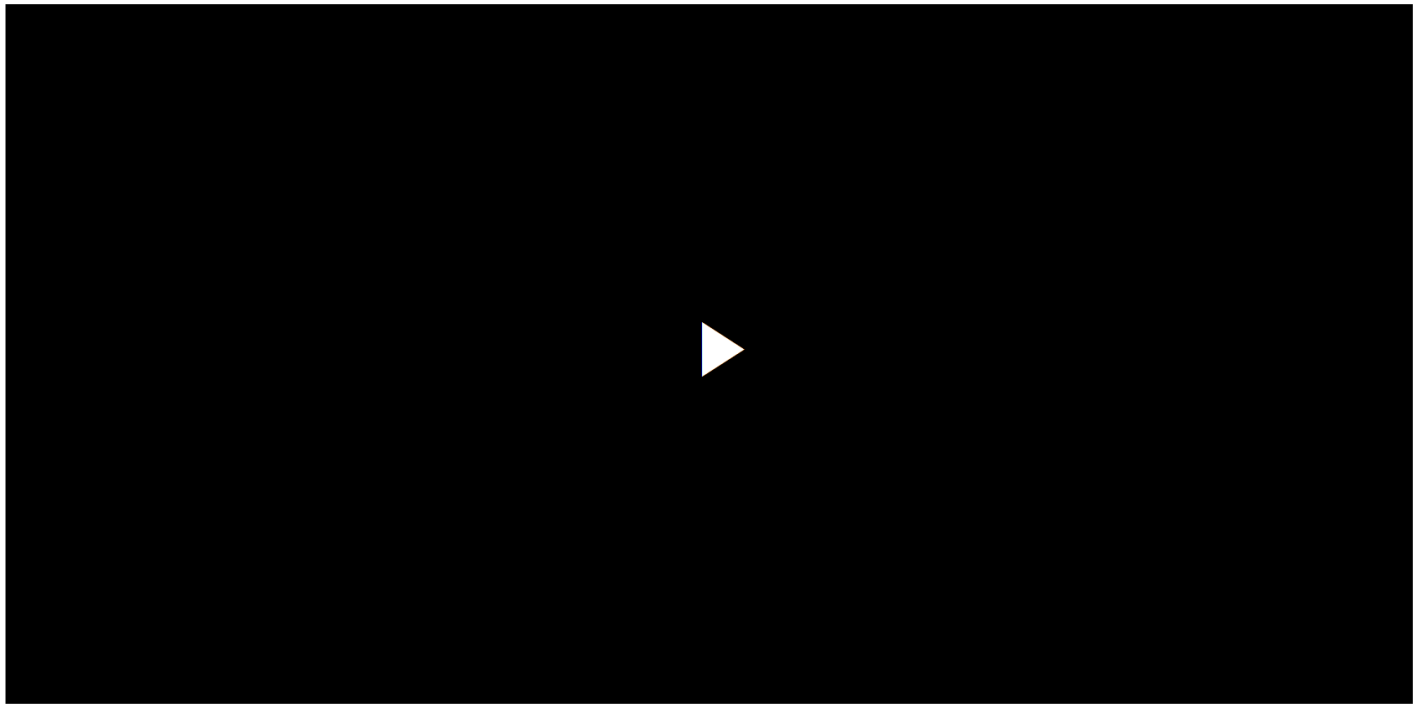


04 프로젝트 수행 결과

학습 데이터 소개(Training)

04 프로젝트 수행 결과

모델 시연 영상



04 자체 평가 의견

팀 평가 및 느낀 점

- **잘한 부분**

  각 팀원이 자신의 역할과 책임을 분명하게 이해하고, 프로젝트 전체의 목표와 일정을 공유하며,
  개인의 경험과 지식을 팀원들에게 공유해서, 서로에게 많은 도움을 주면서 프로젝트를 수행했습니다.

- **아쉬운 부분**

  Object Detection을 하지 못하는 Object들도 많이 있어서 높은 성능의 하드웨어와 많은 양의
  Dataset과 시간이 있었다면 "더 좋은 성능의 Object Detection을 할 수 있었겠다"라는 아쉬움이 있습니다.

- **프로젝트를 수행하면서 경험한 성과**

  최근까지의 Object Detection의 State of the Art Model인 EfficientDet에 대해서 알게 되었고
  Image-Segmentation U-NET을 통해서 Dataset 전 처리 방법과 학습 방법 그리고 Tensorflow Lite모델을
  안드로이드 플랫폼에 탑재해서 학습시킨 Dataset을 바탕으로 도로 주행환경 중 자율주행에 필요한 전방의 Object들을
  감지하는 능력을 키웠습니다.

감사합니다