분석 멘토링 C조

7주차

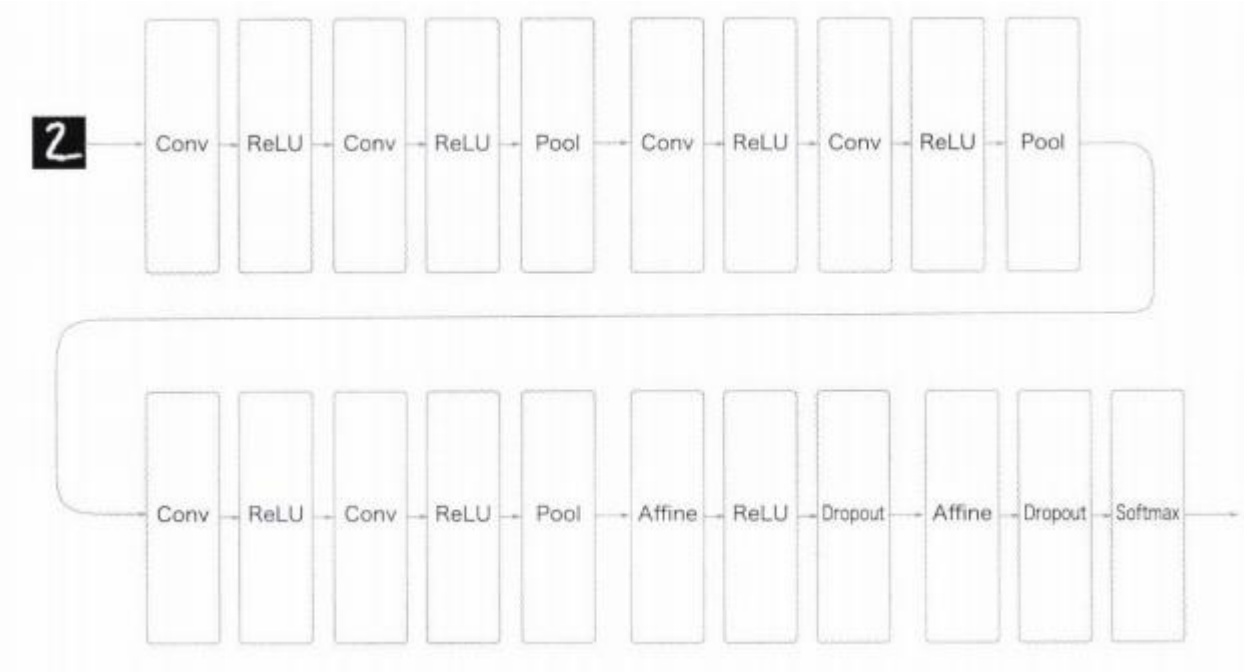
발표자: 김영민

INDEX

1-1. 간단한 손글씨 인식

DeepLearning 층을 더 깊게 만든 심층 신경망

VGG 기반 CNN



특징

- 3x3 필터 사용
- 층이 깊어지면서 채널 수 증가
16 -> 16 -> 32 -> 32 -> 64 -> 64
- Optimizier = Adam
- Activation Function = ReLU
- Fully Connected 뒤에 Drop Out
- 가중치 초기값 'He의 초기값'
- Pooling으로 공간 크기 줄임

1-1. 손글씨 구현 CNN

```
def __init__(self, input_dim=(1, 28, 28),
             conv_param_1 = {'filter_num':16, 'filter_size':3, 'pad':1, 'stride':1},
             conv_param_2 = {'filter_num':16, 'filter_size':3, 'pad':1, 'stride':1},
             conv_param_3 = {'filter_num':32, 'filter_size':3, 'pad':1, 'stride':1},
             conv_param_4 = {'filter_num':32, 'filter_size':3, 'pad':2, 'stride':1},
             conv_param_5 = {'filter_num':64, 'filter_size':3, 'pad':1, 'stride':1},
             conv_param_6 = {'filter_num':64, 'filter_size':3, 'pad':1, 'stride':1},
             hidden_size=50, output_size=10):
    # 가중치 초기화===========
    # 각 층의 뉴런 하나당 앞 층의 몇 개 뉴런과 연결되는가 (TODO: 자동 계산되게 바꿀 것)
    pre_node_nums = np.array([1*3*3, 16*3*3, 16*3*3, 32*3*3, 32*3*3, 64*3*3, 64*4*4, hidden_size])
    wight_init_scales = np.sqrt(2.0 / pre_node_nums)  # ReLU를 사용할 때의 권장 초깃값

    self.params = {}
    pre_channel_num = input_dim[0]
    for idx, conv_param in enumerate([conv_param_1, conv_param_2, conv_param_3, conv_param_4, conv_param_5, conv_param_6]):
        self.params['W' + str(idx+1)] = wight_init_scales[idx] * ₩
        np.random.randn(conv_param['filter_num'], pre_channel_num, conv_param['filter_size'], conv_param['filter_size'])
        self.params['b' + str(idx+1)] = np.zeros(conv_param['filter_num'])
        pre_channel_num = conv_param['filter_num']
    self.params['W7'] = wight_init_scales[6] * np.random.randn(64*4*4, hidden_size)
    self.params['b7'] = np.zeros(hidden_size)
    self.params['W8'] = wight_init_scales[7] * np.random.randn(hidden_size, output_size)
    self.params['b8'] = np.zeros(output_size)
    # 계층 생성===========
    self.layers = []
    self.layers.append(Convolution(self.params['W1'], self.params['b1'],
                       conv_param_1['stride'], conv_param_1['pad']))
    self.layers.append(Relu())
    self.layers.append(Convolution(self.params['W2'], self.params['b2'],
                       conv_param_2['stride'], conv_param_2['pad']))
    self.layers.append(Relu())
    self.layers.append(Pooling(pool_h=2, pool_w=2, stride=2))
    self.layers.append(Convolution(self.params['W3'], self.params['b3'],
                       conv_param_3['stride'], conv_param_3['pad']))
    self.layers.append(Relu())
    self.layers.append(Convolution(self.params['W4'], self.params['b4'],
                       conv_param_4['stride'], conv_param_4['pad']))
    self.layers.append(Relu())
    self.layers.append(Pooling(pool_h=2, pool_w=2, stride=2))
    self.layers.append(Convolution(self.params['W5'], self.params['b5'],
                       conv_param_5['stride'], conv_param_5['pad']))
    self.layers.append(Relu())
    self.layers.append(Convolution(self.params['W6'], self.params['b6'],
                       conv_param_6['stride'], conv_param_6['pad']))
    self.layers.append(Relu())
    self.layers.append(Pooling(pool_h=2, pool_w=2, stride=2))
    self.layers.append(Affine(self.params['W7'], self.params['b7']))
    self.layers.append(Relu())
    self.layers.append(Dropout(0.5))
    self.layers.append(Affine(self.params['W8'], self.params['b8']))
    self.layers.append(Dropout(0.5))
```

채널 수 증가, 필터 : 3x3

He initialization

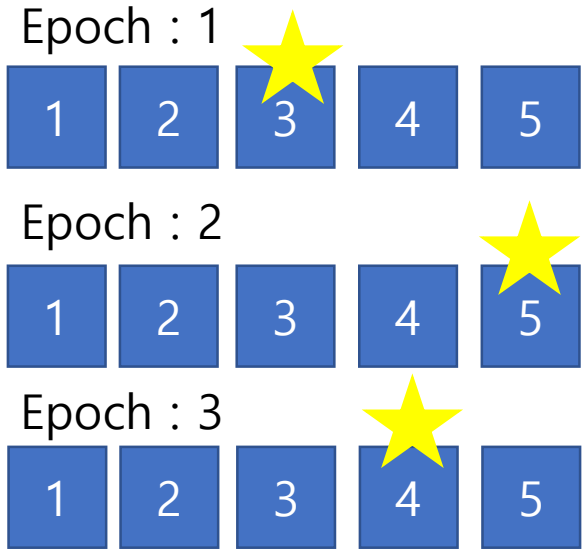
활성화 함수 : ReLU

Pooling으로 공간 줄임

Drop out 사용

1-2. 정확도를 높이는 방법

1. 앙상블 학습



1-3 모델로 예측

+

2-5 모델로 예측

+

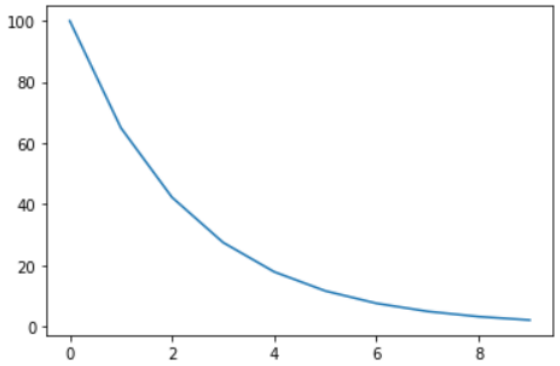
3-4 모델로 예측

AVG

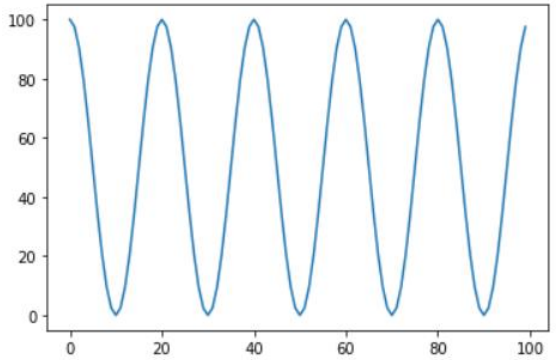
Final output

2. 학습률 감소

- Lambda LR



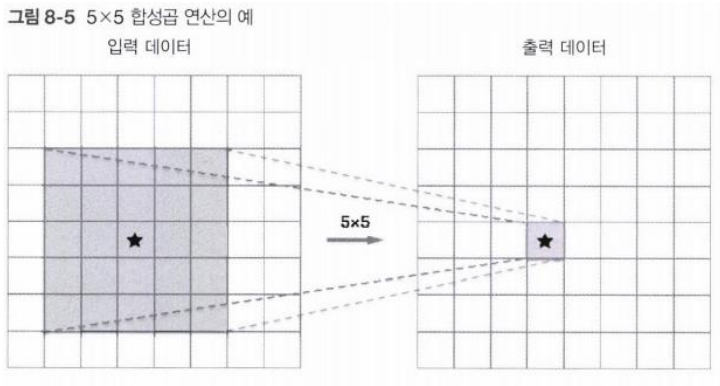
- CosineAnnealingLR



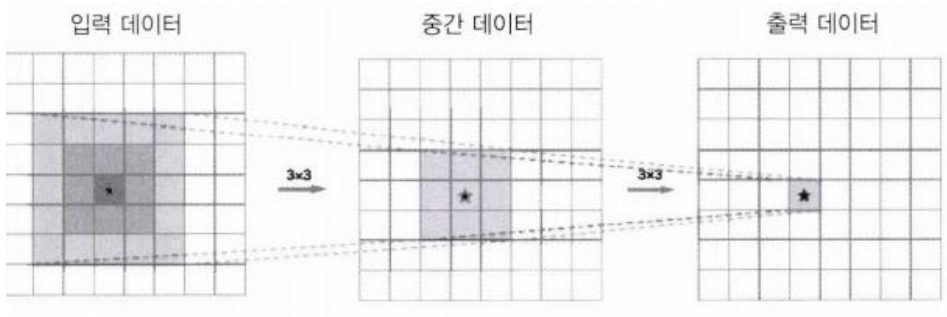
3. 데이터 확장

1-3. 깊게 하는 이유

1. 매개변수 수를 줄일 수 있다.



5x5 합성곱 연산 : 매개변수 25



3x3 합성곱 연산 2회 반복 : 매개변수(2x3x3) 18

2. 학습의 효율성 높일 수 있다.

문제를 계층적으로 분해 가능

= 처음 층에서 edge의 패턴을 학습

→ 풀기 쉬운 문제로 분해 가능

작은 필터를 겹쳐 신경망을 깊게 했을 때 장점

1. 매개 변수 수를 줄여 넓은 수용영역 소화 가능
2. 층을 거듭하며 활성화 함수를 합성곱 사이에 끼워 신경망 표현력 개선

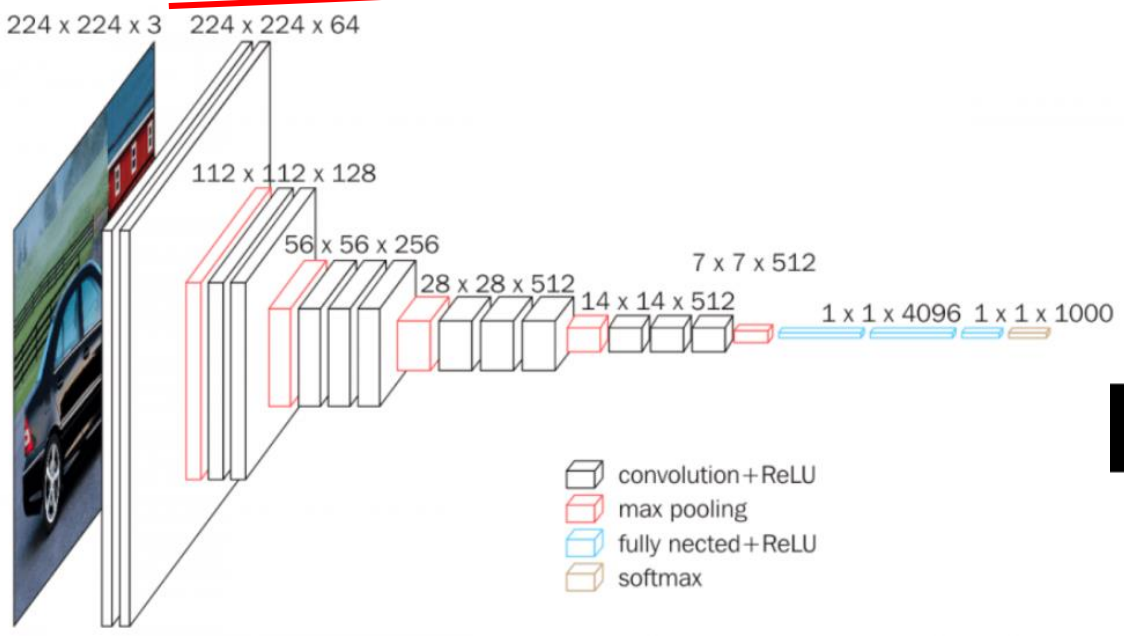
→ 비선형성이 증가하여 복잡한 표현도 가능

2. 여러 CNN 모델

2-1. VGGNet

VGG Architecture(VGG16)

Input_size 고정



1층 : 64개의 filter kernel로 input image convolution

4층 : 3x3x128로 convolution
2 x 2 max pooling을 stride 2로 적용해준다.

14층 : 7 x 7 x 512의 feature map flatten
Fully connected, Dropout 적용

16층 : softmax function 적용

3x3 필터 사용 → 파라미터의 개수를 줄여 효율적인 학습을 이뤄냄

Stride = 1 , zero padding,
Activation function = ReLU

2-1. VGGNet

Pytorch로 구현한 VGGNet(Simple version)

```
class VGG(nn.Module):
    def __init__(self, base_dim, num_classes=2):
        super(VGG, self).__init__()
        self.feature = nn.Sequential(
            conv_2_block(3,base_dim),
            conv_2_block(base_dim,2*base_dim),
            conv_3_block(2*base_dim,4*base_dim),
            conv_3_block(4*base_dim,8*base_dim),
            conv_3_block(8*base_dim,8*base_dim),
        )
        self.fc_layer = nn.Sequential(
            nn.Linear(8*base_dim * 7 * 7, 100),
            nn.ReLU(True),Inplace 연산 실행
            #nn.Dropout(),
            nn.Linear(100, 20),
            nn.ReLU(True),
            #nn.Dropout(),
            nn.Linear(20, num_classes),
        )

    def forward(self, x):
        x = self.feature(x)
        x = x.view(x.size(0), -1)
        x = self.fc_layer(x)
        return x
```

Convolution 연산 2번

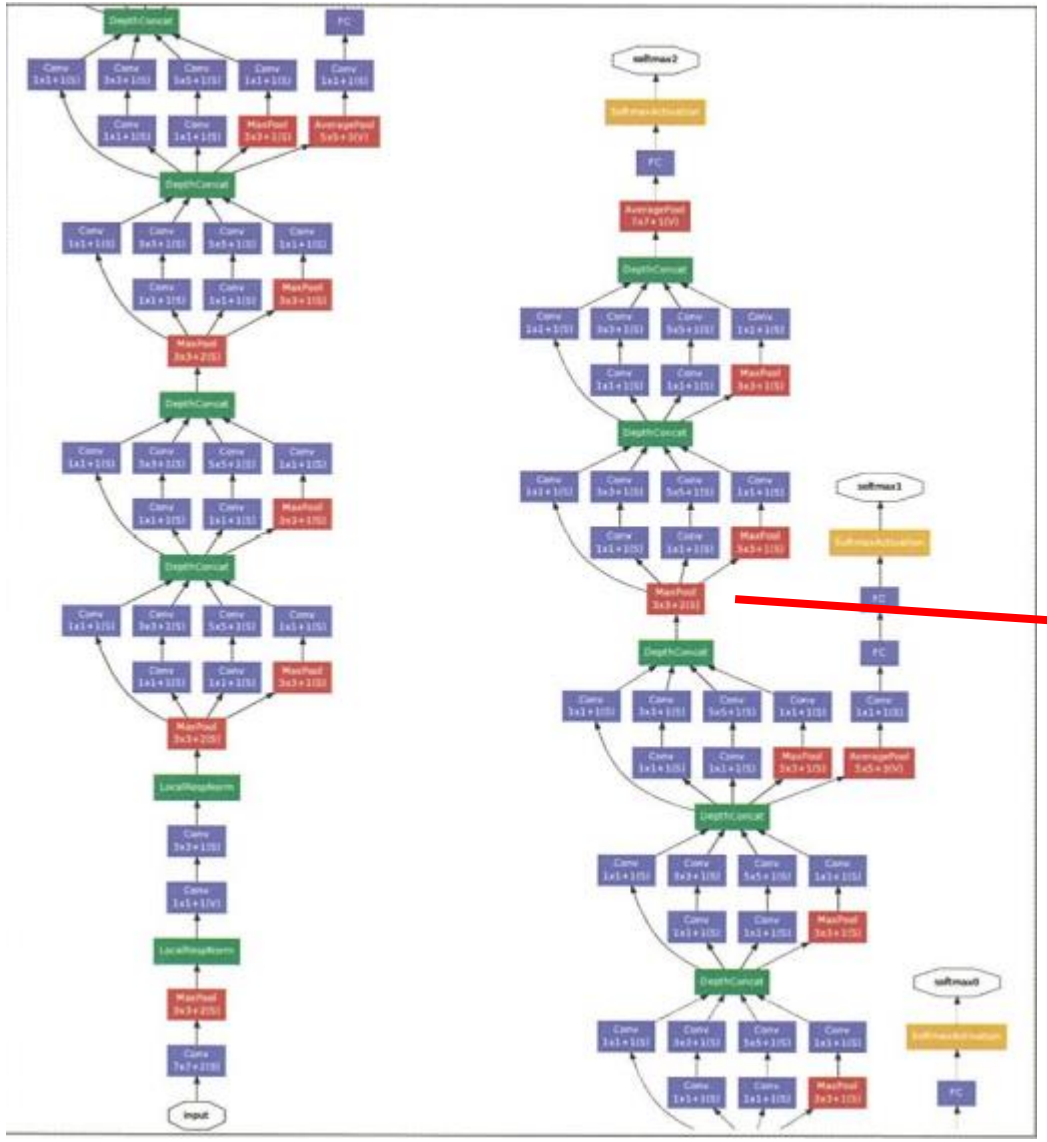
```
def conv_2_block(in_dim,out_dim):
    model = nn.Sequential(
        nn.Conv2d(in_dim,out_dim,kernel_size=3,padding=1),
        nn.ReLU(),
        nn.Conv2d(out_dim,out_dim,kernel_size=3,padding=1),
        nn.ReLU(),
        nn.MaxPool2d(2,2)
    )
    return model
```

Convolution 연산 3번

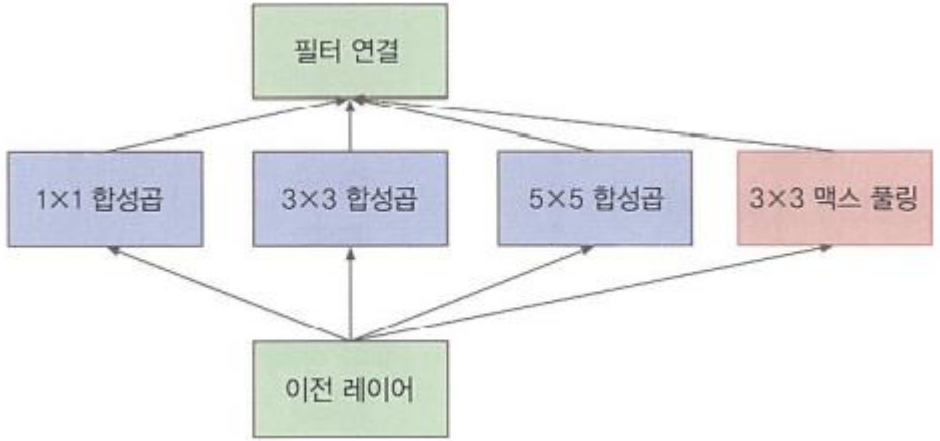
```
def conv_3_block(in_dim,out_dim):
    model = nn.Sequential(
        nn.Conv2d(in_dim,out_dim,kernel_size=3,padding=1),
        nn.ReLU(),
        nn.Conv2d(out_dim,out_dim,kernel_size=3,padding=1),
        nn.ReLU(),
        nn.Conv2d(out_dim,out_dim,kernel_size=3,padding=1),
        nn.ReLU(),
        nn.MaxPool2d(2,2)
    )
    return model
```

2-2. GoogLeNet

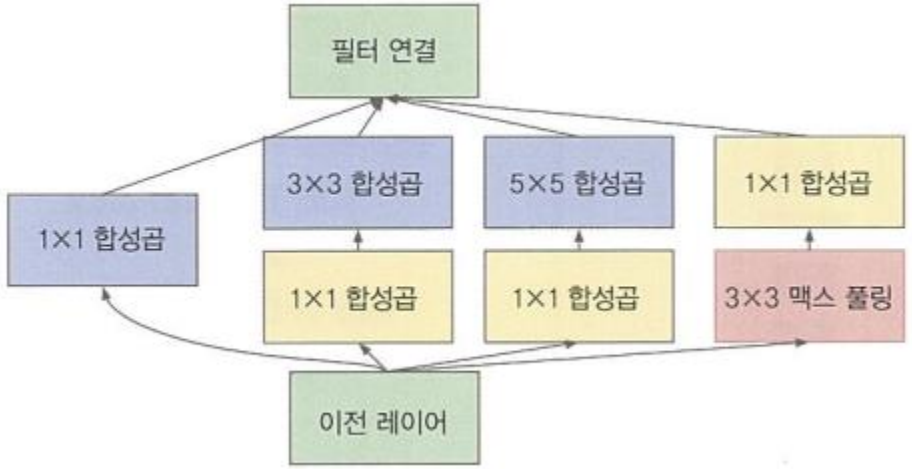
GoogleNet Architecture



기초적인 Inception 모듈



차원 감소를 더한 Inception 모듈



2-2. GoogLeNet

Pytorch로 구현한 GoogleNet Inception module

1x1 convolution

```
def conv_1(in_dim,out_dim):
    model = nn.Sequential(
        nn.Conv2d(in_dim,out_dim,1,1),
        nn.ReLU(),
    )
    return model
```

1x1 Conv -> 5x5 Conv

```
def conv_1_5(in_dim,mid_dim,out_dim):
    model = nn.Sequential(
        nn.Conv2d(in_dim,mid_dim,1,1),
        nn.ReLU(),
        nn.Conv2d(mid_dim,out_dim,5,1,2),
        nn.ReLU()
    )
    return model
```

1x1 Conv -> 3x3 Conv

```
def conv_1_3(in_dim,mid_dim,out_dim):
    model = nn.Sequential(
        nn.Conv2d(in_dim,mid_dim,1,1),
        nn.ReLU(),
        nn.Conv2d(mid_dim,out_dim,3,1,1),
        nn.ReLU()
    )
    return model
```

3x3 max pooling -> 1x1 Conv

```
def max_3_1(in_dim,out_dim):
    model = nn.Sequential(
        nn.MaxPool2d(3,1,1),
        nn.Conv2d(in_dim,out_dim,1,1),
        nn.ReLU(),
    )
    return model
```

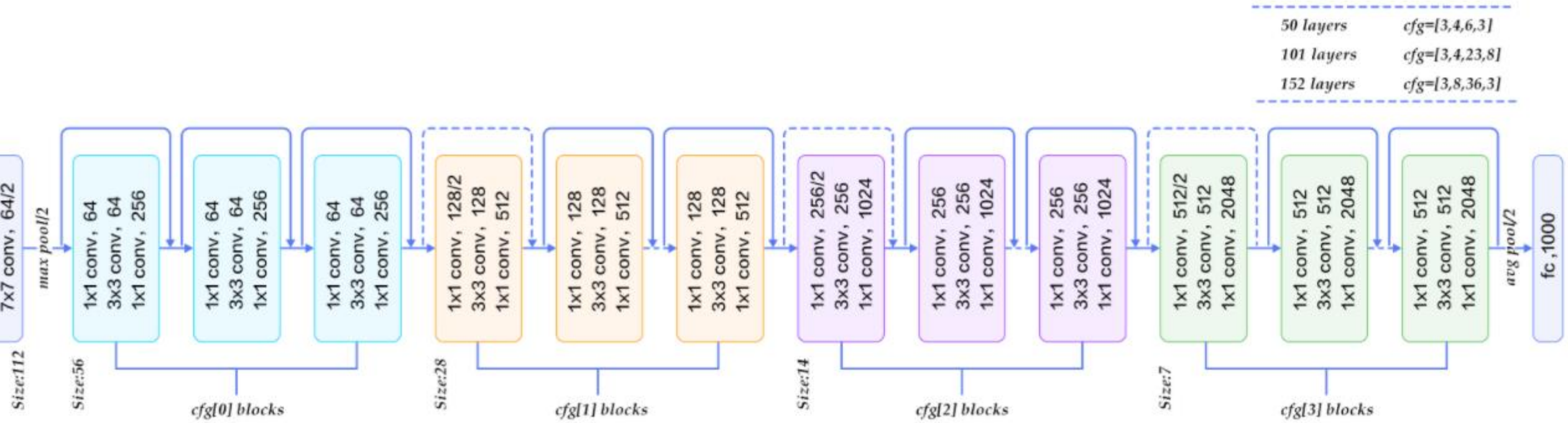
Inception Module

```
class inception_module(nn.Module):
    def __init__(self,in_dim,out_dim_1,mid_dim_3,out_dim_3,mid_dim_5,out_dim_5,pool):
        super(inception_module,self).__init__()
        self.conv_1 = conv_1(in_dim,out_dim_1)
        self.conv_1_3 = conv_1_3(in_dim,mid_dim_3,out_dim_3)
        self.conv_1_5 = conv_1_5(in_dim,mid_dim_5,out_dim_5)
        self.max_3_1 = max_3_1(in_dim,pool)

    def forward(self,x):
        out_1 = self.conv_1(x)
        out_2 = self.conv_1_3(x)
        out_3 = self.conv_1_5(x)
        out_4 = self.max_3_1(x)
        output = torch.cat([out_1,out_2,out_3,out_4],1)
        return output
```

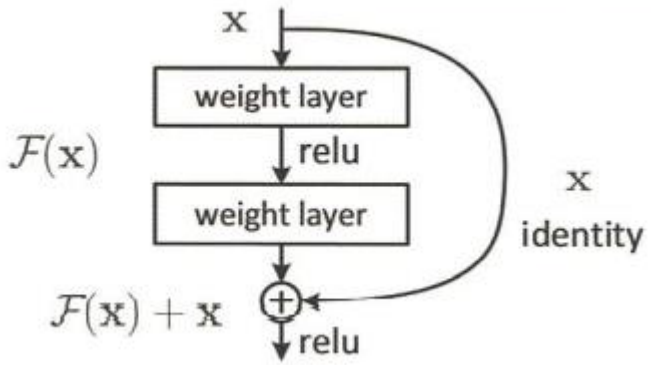
2-3. ResNet

ResNet Architecture



2-3. ResNet

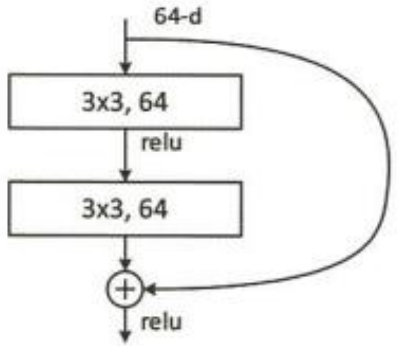
잔차 학습 블록

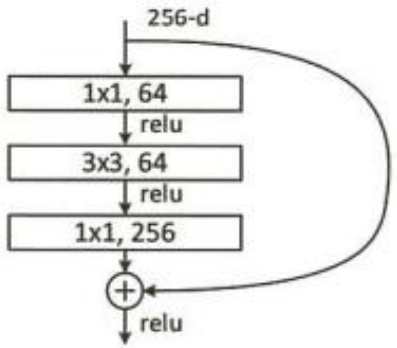


이전 단계의 특성 변형 X (단순 특성)

단순한 특성 + 복잡한 특성 모두 사용

BottleNeck





GoogleNet의 아이디어 차용

2-3. ResNet

Pytorch로 구현한 BottleNeck

```
class BottleNeck(nn.Module):
    def __init__(self,in_dim,mid_dim,out_dim,act_fn,down=False):
        super(BottleNeck,self).__init__()
        self.down=down

        # 특성지도의 크기가 감소하는 경우
        if self.down:
            self.layer = nn.Sequential(
              conv_block_1(in_dim,mid_dim,act_fn,2),
              conv_block_3(mid_dim,mid_dim,act_fn),
              conv_block_1(mid_dim,out_dim,act_fn),
            )
            self.downsample = nn.Conv2d(in_dim,out_dim,1,2)

        # 특성지도의 크기가 그대로인 경우
        else:
            self.layer = nn.Sequential(
                conv_block_1(in_dim,mid_dim,act_fn),
                conv_block_3(mid_dim,mid_dim,act_fn),
                conv_block_1(mid_dim,out_dim,act_fn),
            )

        # 더하기를 위해 차원을 맞춰주는 부분
        self.dim_equalizer = nn.Conv2d(in_dim,out_dim,kernel_size=1)
```

```
def forward(self,x):
    if self.down:
        downsample = self.downsample(x)
        out = self.layer(x)
        out = out + downsample
    else:
        out = self.layer(x)
        if x.size() is not out.size():
            x = self.dim_equalizer(x)
        out = out + x
    return out
```

Block을 통과했을 때 크기가 줄어드는지 여부

1x1 컨볼루션 -> 3x3 컨볼루션 -> 1x1 컨볼루션

Feature map이 감소할 때와 그대로일 때의 경우를 나눔

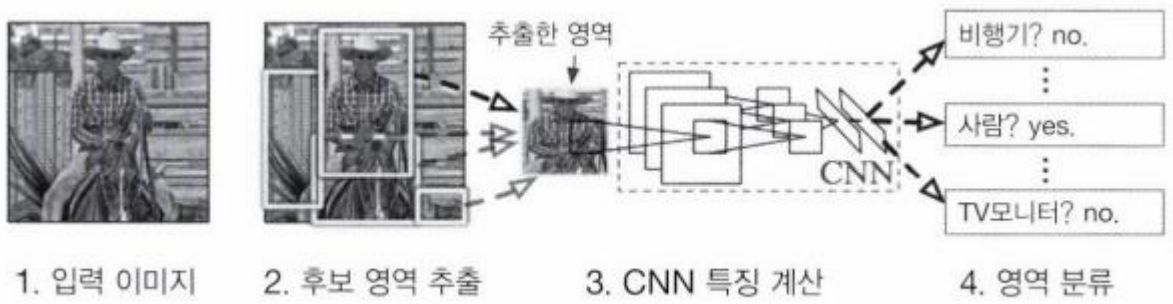
차원의 크기를 맞춰주는 부분

3. 딥러닝의 활용 및 미래

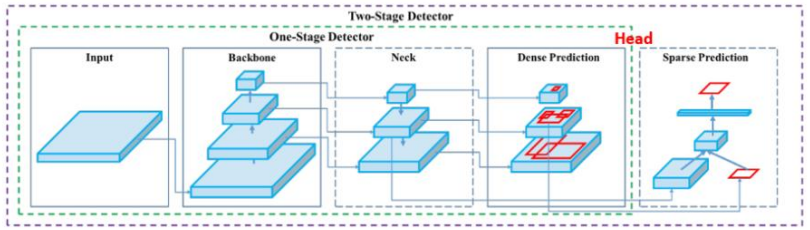
3-1-1. Object Detection

Object Detection 사물 검출 (ex. R-CNN, YOLO)

R-CNN



YOLO



Input: { Image, Patches, Image Pyramid, ... }

Backbone: { VGG16 [68], ResNet-50 [26], ResNeXt-101 [86], Darknet53 [63], ... } EfficientNet-B0/B7 [75], CSPResNeXt50 [81], CSPDarknet53 [81]

Neck: { FPN [44], PANet [49], Bi-FPN [77], ... } NAS-FPN [17], ASFF [48], SFAM [98] SPP [25], ASPP [5], RFB [47], SAM [85]

Head:

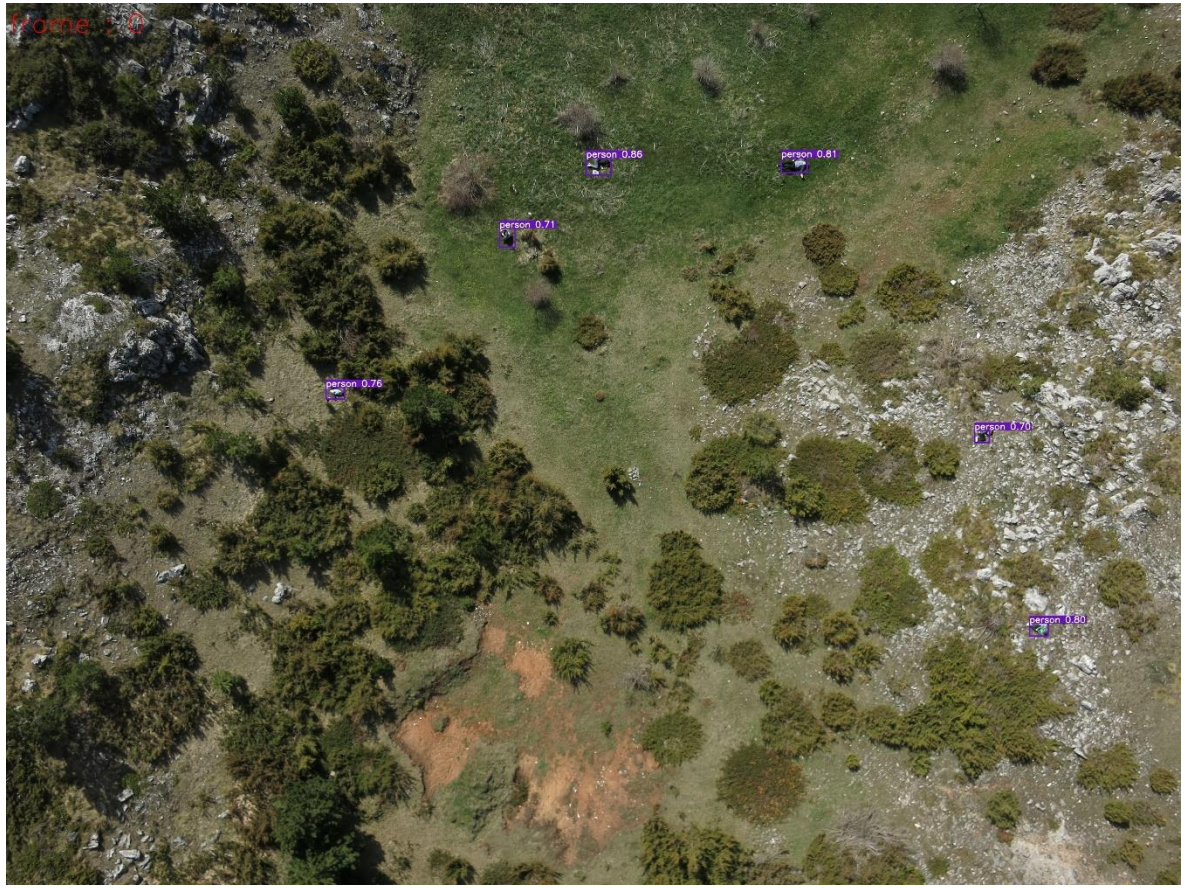
Dense Prediction: { RPN [64], YOLO [61, 62, 63], SSD [50], RetinaNet [45], FCOS [78], ... } CornerNet [37], CenterNet [13], MatrixNet [60]

Sparse Prediction: { Faster R-CNN [64], R-FCN [9], ... } Mask R-CNN [23] RepPoints [87]

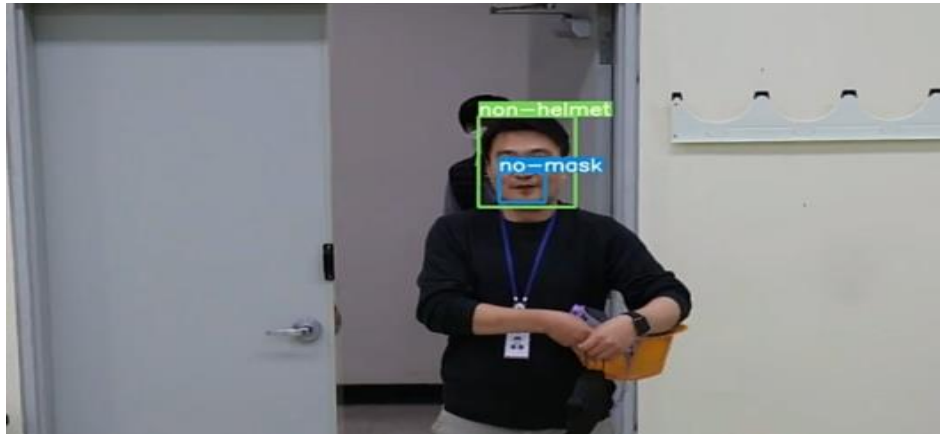
Figure 2: Object detector.

4-1-1. Object Detection

YOLOv5를 이용한 드론에서의 객체 검출



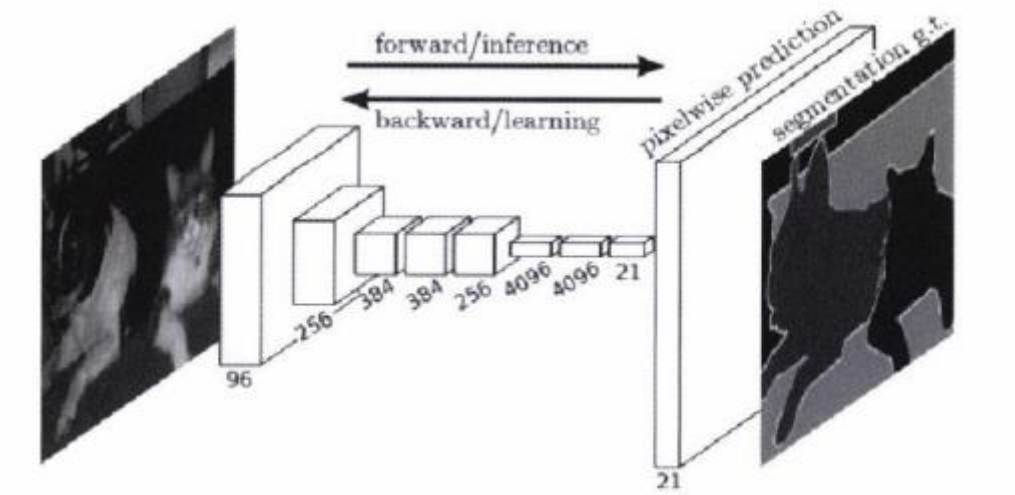
YOLOv5를 이용한 마스크 헬멧 검사



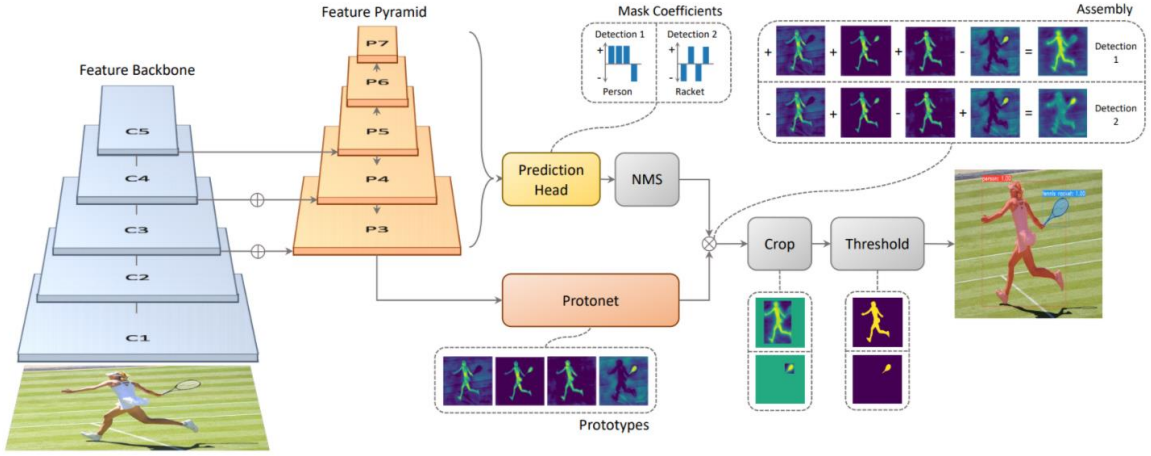
4-1-2. Segmentation

Image Segmentation 이미지를 픽셀 수준에서 분할

Fully-Convolution-Network



YOLACT



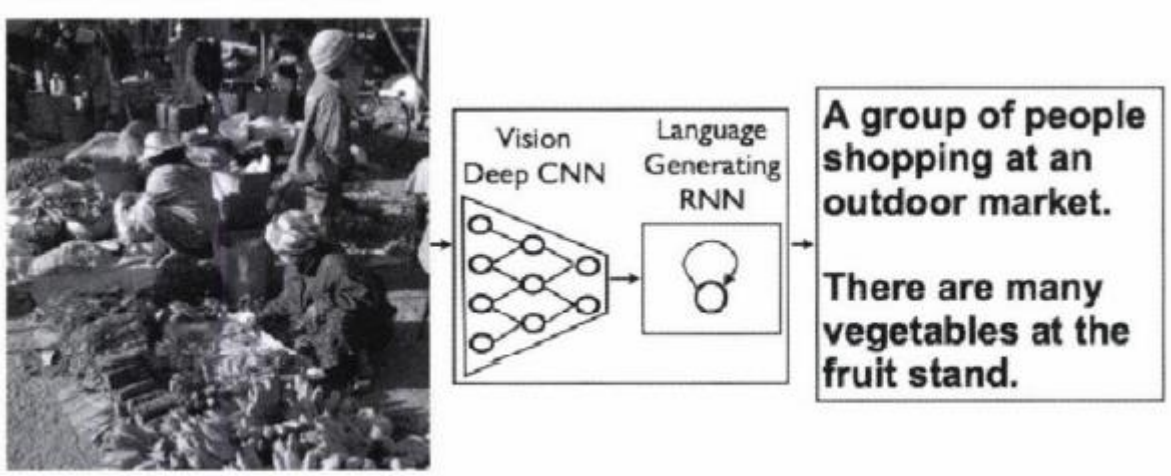
4-1-2. Segmentation

Detectron2를 활용한 Image Segmentation

4-1-2. 사진 캡션 생성

사진 캡션 생성

NIC(Neural Image Caption)



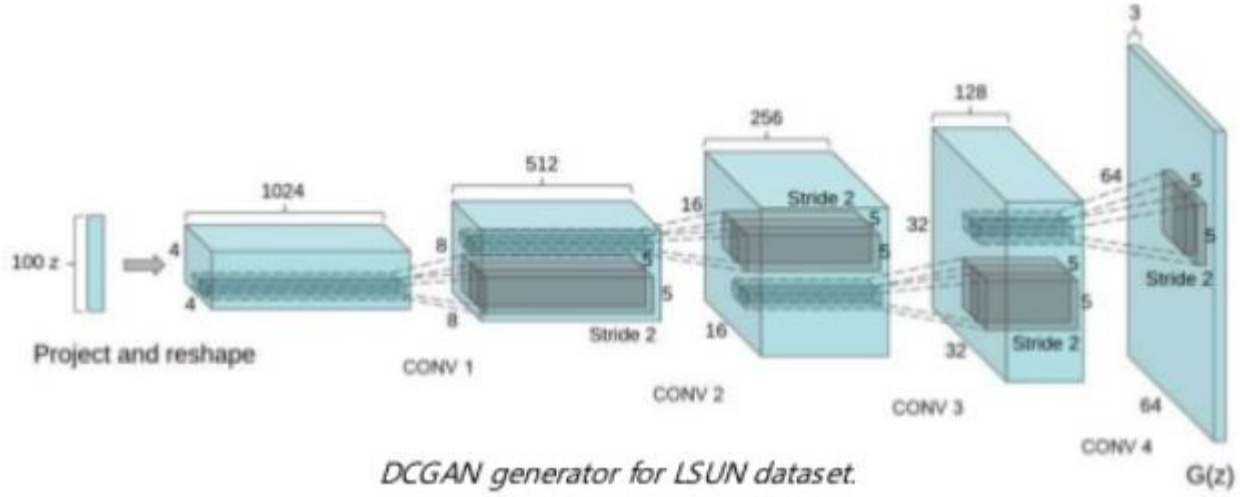
CNN + RNN(LSTM)

Encoder = CNN ⇨ Decoder = RNN

4-2-1. Image Generation

Image Generation 이미지를 생성하는 기술

DCGAN(Deep Convolution Generative Adversarial Network)



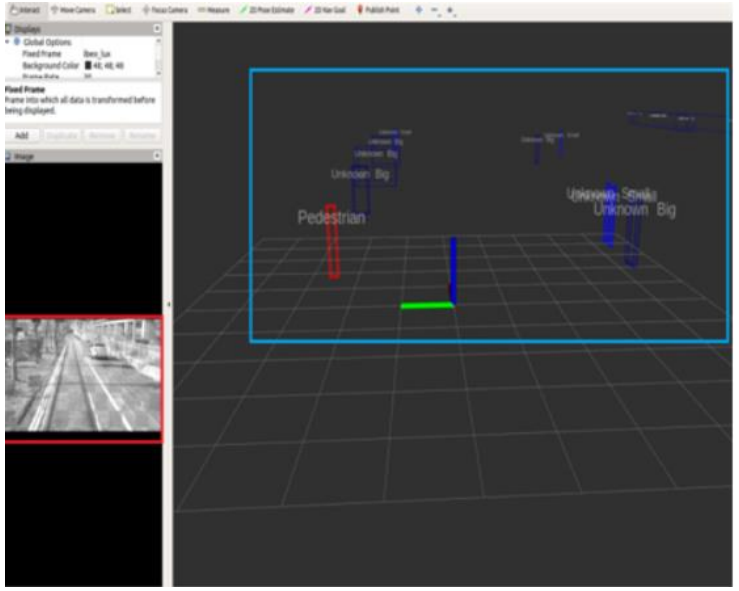
DCGAN generator for LSUN dataset.

Fully Connected를 CNN 구조로 대체
(기존 GAN과의 큰 차이점)

https://angrypark.github.io/generative%20models/paper%20review/DCGAN-paper-reading/

4-2-2. 자율주행

Self - Driving Lidar, Rader, IMU, Camera 등을 이용하여 운전자 없이 자율 주행을 이뤄냄



Lidar를 이용한 객체 인식

객체 인식 및 충돌 시간 및 지점 예측

4-2-2. 자율주행

Flow chart

Subscribe Image → Yolo Detection → Dense Optical flow → Remove Ego motion →

Behavior Estimation → Calculate TTC & Collision Point → Publish Image →

4-2-3. 강화학습

Reinforcement Learning 컴퓨터에게도 시행착오를 겪게 하면서 스스로 학습하게 함