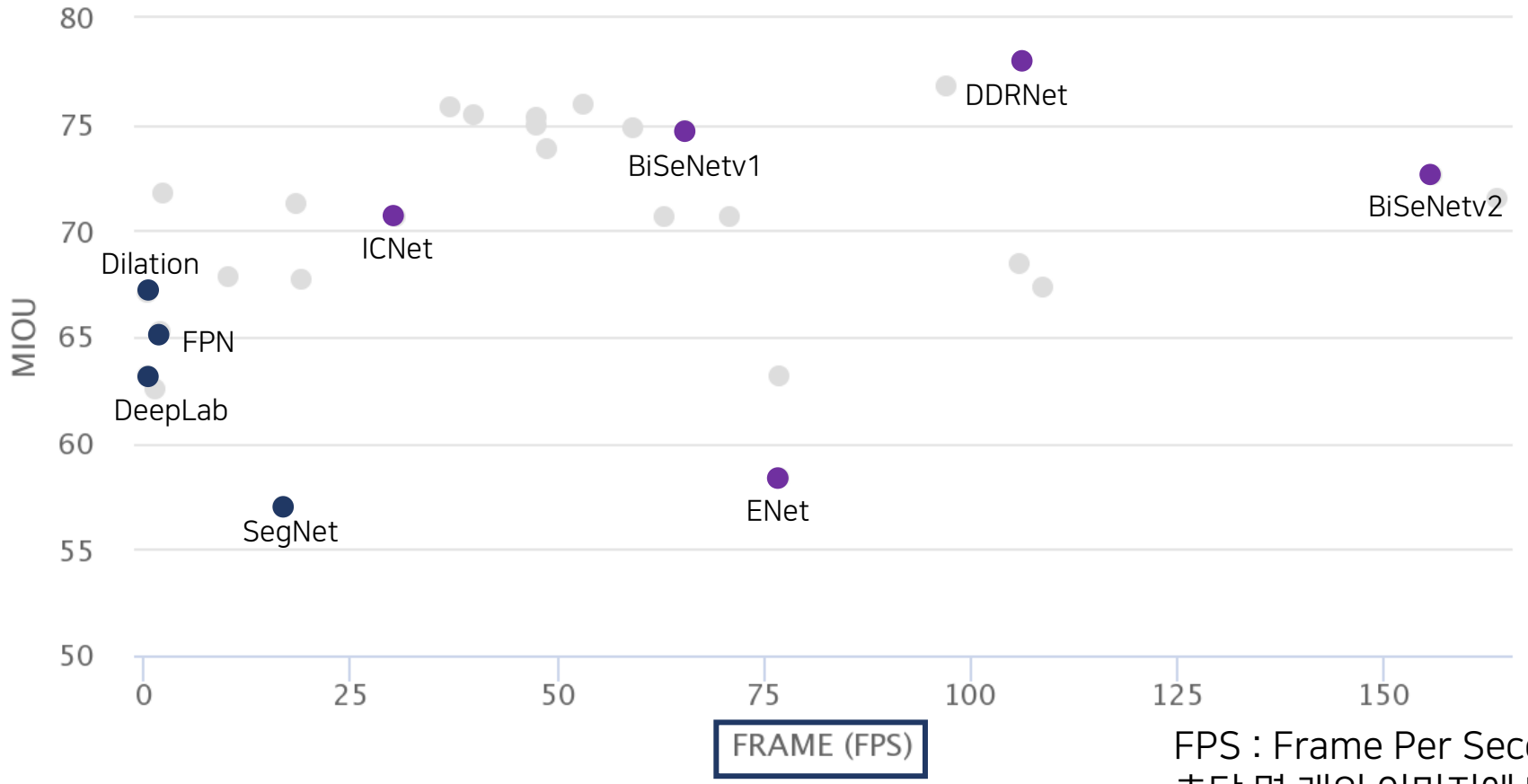
ENet

A Deep Neural Network Architecture for Real-Time Semantic Segmentation

0. Abstract

(1) Abstract

Real Time Semantic Segmentation



+ EspNet, ERFNet

FPS : Frame Per Second
초당 몇 개의 이미지에 대해서
Segmentation을 수행할 수 있는지
(Inference Time)

출처 : https://paperswithcode.com/sota/real-time-semantic-segmentation-on-cityscapes

0. Abstract

(1) Abstract

Real Time Semantic Segmentation

Table 2: Performance comparison.

Model	NVIDIA TX1						NVIDIA Titan X					
	480×320		640×360		1280×720		640×360		1280×720		1920×1080	
	ms	fps	ms	fps	ms	fps	ms	fps	ms	fps	ms	fps
SegNet	757	1.3	1251	0.8	-	-	69	14.6	289	3.5	637	1.6
ENet	47	21.1	69	14.6	262	3.8	7	135.4	21	46.8	46	21.6

Table 3: Hardware requirements. FLOPs are estimated for an input of $3 \times 640 \times 360$.

	GFLOPs	Parameters	Model size (fp16)
SegNet	286.03	29.46M	56.2 MB
ENet	3.83	0.37M	0.7 MB

18x Speed Up
75x Less FLOPs
79x Less parameters

ms : 하나의 사진을 Inference 하는데 걸리는 시간 (1/1000 s)
fps : 1s에 inference 하는 사진의 수

예) 1251 / 1000(ms) = 0.8 fps

GFLOPs : GPU Floating point Operations Per Second
Parameters : 파라미터의 수
Model size (fp16) : Quantization 처럼 16bit로 모델 저장해서 크기 잰 Size

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

1 Introduction

- (1) Real-time Semantic Segmentation

Real-Time Semantic Segmentation

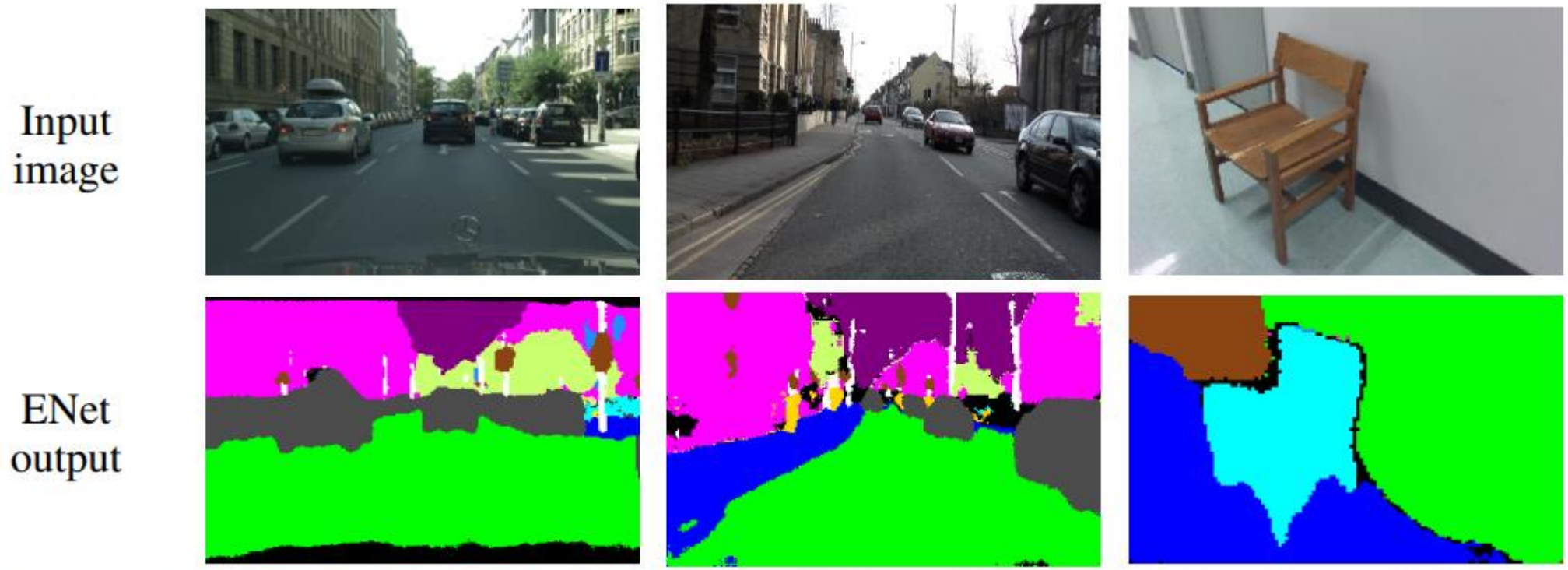


Figure 1: ENet predictions on different datasets (left to right Cityscapes, CamVid, and SUN).

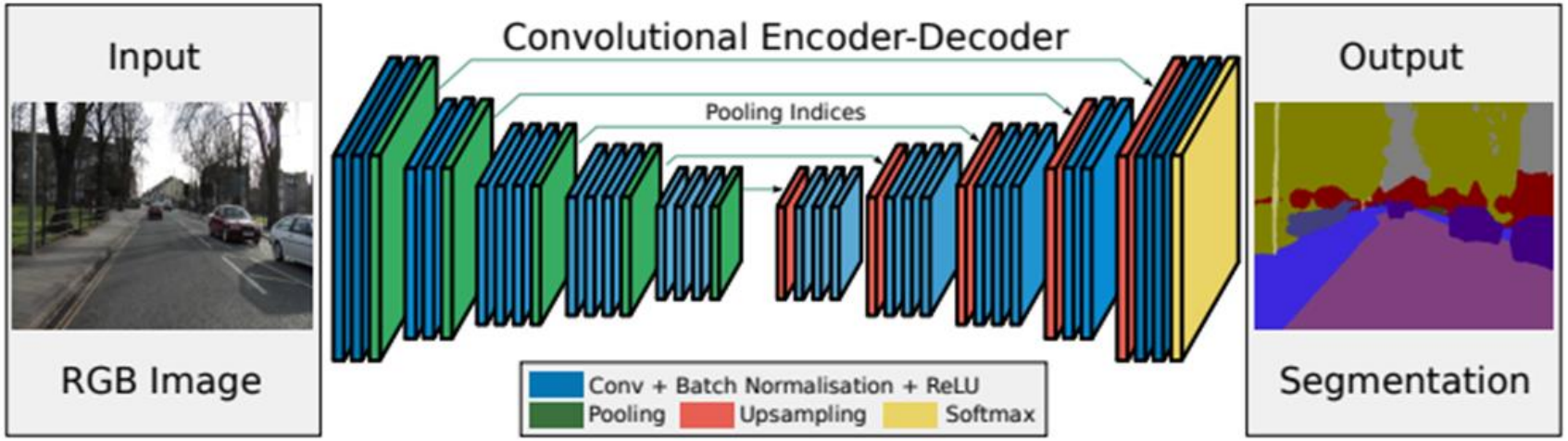
출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

2. Related Work

(1) Encoder-Decoder

Real-Time Semantic Segmentation

SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation (2015)



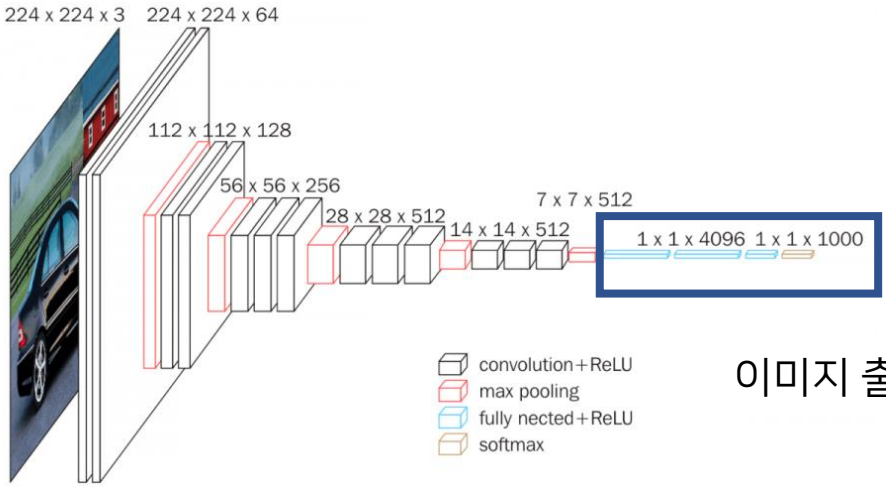
출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

2. Related Work

(1) Encoder-Decoder

Real-Time Semantic Segmentation

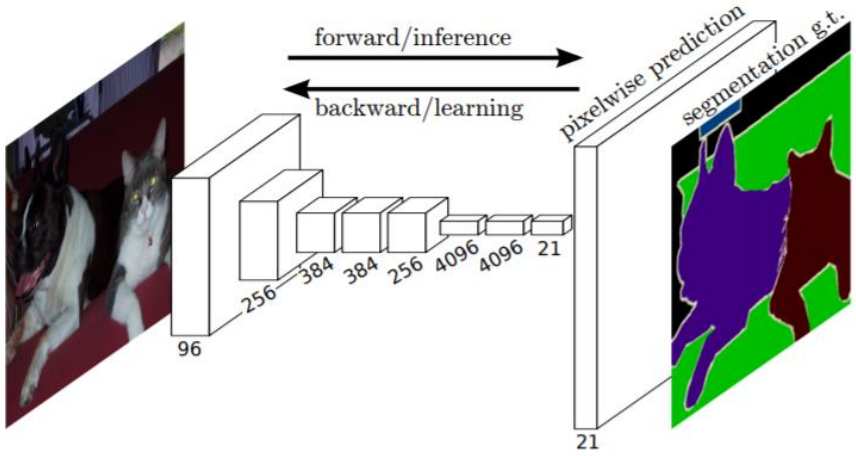
VGG16 – Convolutional Network for Classification and Detection (2014)



속도 상승을 위해서 VGG의 3개의 FC Layer를 제거한 형태 -> SegNet

이미지 출처 : https://neurohive.io/en/popular-networks/vgg16/

Fully Convolutional Networks for Semantic Segmentation (2015)



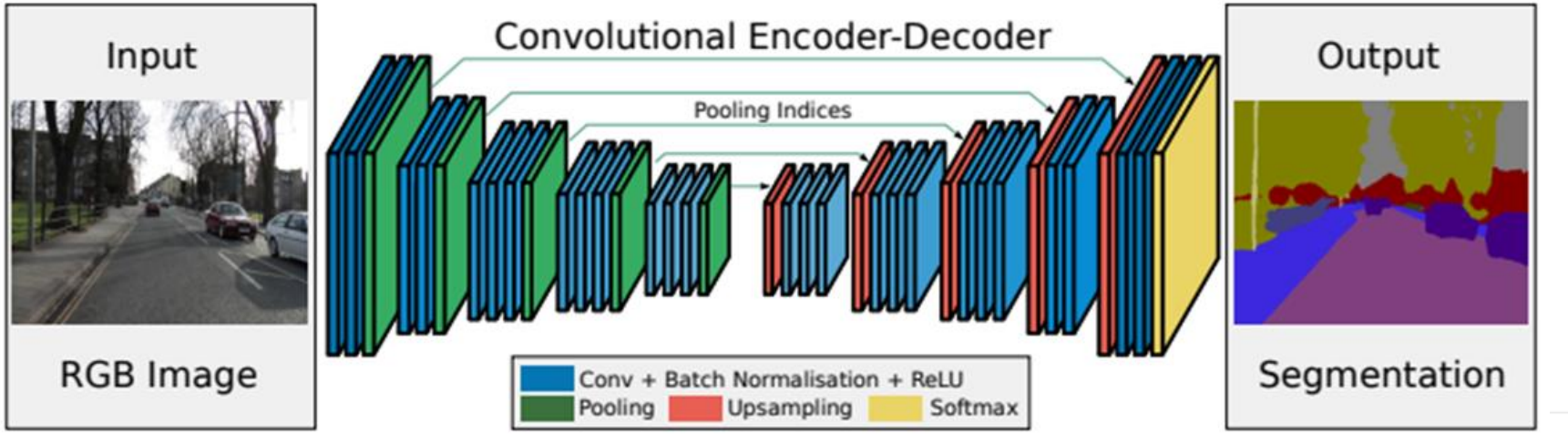
출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

2. Related Work

(1) Encoder-Decoder

Real-Time Semantic Segmentation

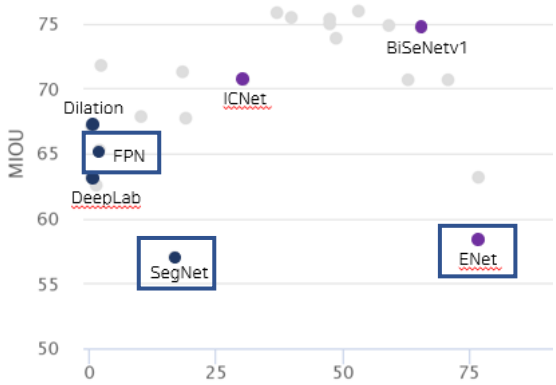
SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation (2015)



속도 상승을 위해서 VGG의 3개의 FC Layer를 제거한 형태 -> SegNet

- Floating point operations 수 감소
- Memory footprint 감소
- 파라미터의 수 감소
- 속도의 상승

-> 하지만, 아직 속도의 상승이 부족하고 성능이 너무 나오지 않는 한계가 있음



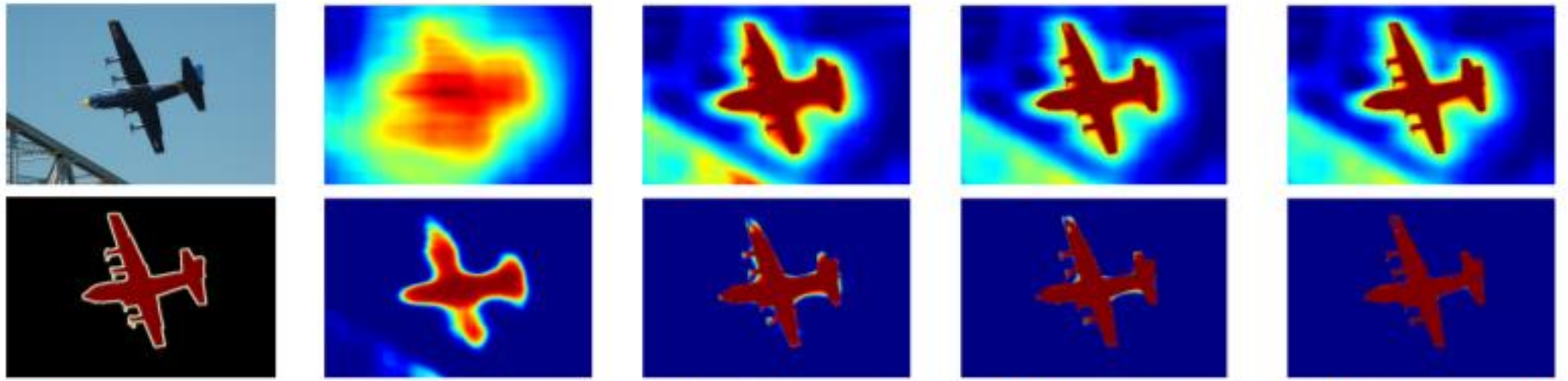
출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

2. Related Work

(2) Other's

Conditional Random Field

Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs (2015)



$$w_1 \exp\left(-\frac{||p_i - p_j||^2}{2\sigma_\alpha^2} - \frac{||I_i - I_j||^2}{2\sigma_\beta^2}\right) + w_2 \exp\left(-\frac{||p_i - p_j||^2}{2\sigma_\gamma^2}\right)$$

픽셀의 Position (p)정보와 Color (I)를 이용해서 후처리를 하는 방법

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

2. Related Work

(2) Other's

Recurrent Neural Network

Conditional Random Fields as Recurrent Neural Networks (CRF-RNN) (2015)

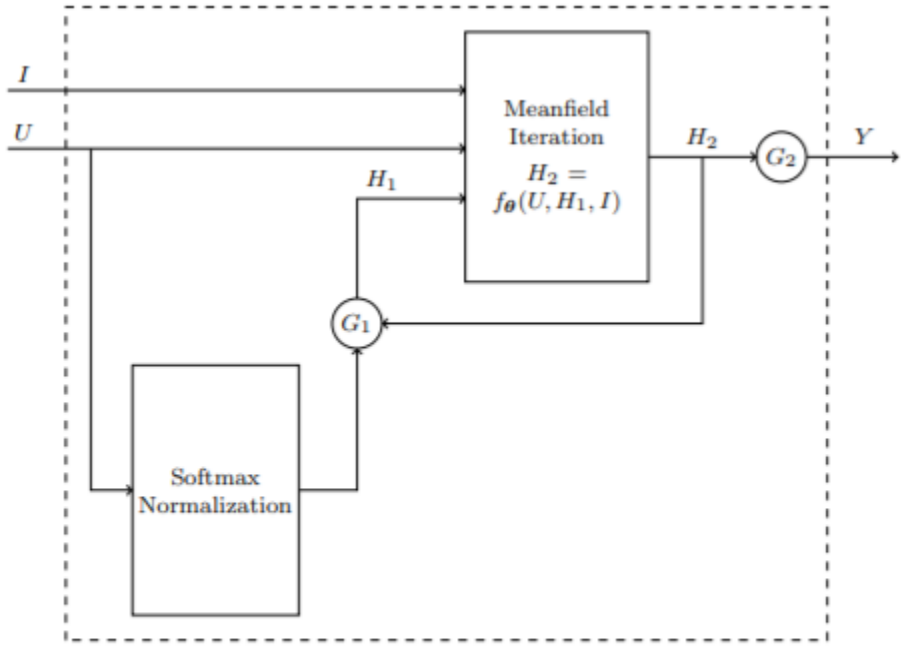


Figure 2. **The CRF-RNN Network.** We formulate the iterative mean-field algorithm as a Recurrent Neural Network (RNN). Gating functions G_1 and G_2 are fixed as described in the text.

$$E(x|I) = \sum_i \phi_u(x_i|I) + \sum_i \sum_{j \in \partial i} \phi_p(x_i, x_j|I)$$

Unary term　　Pairwise term

- X: a random field defined over a set of variables $\{X_1, \ldots, X_N\}$
  - Label of pixels (grass, bench, tree,..)
- I: a random field defined over a set of variables $\{I_1, \ldots, I_N\}$
  - Image (observation)

mean field approximation 해서 구하면,

$$w_1 \exp\left(-\frac{||p_i - p_j||^2}{2\sigma_\alpha^2} - \frac{||I_i - I_j||^2}{2\sigma_\beta^2}\right) + w_2 \exp\left(-\frac{||p_i - p_j||^2}{2\sigma_\gamma^2}\right)$$

출처 : http://swoh.web.engr.illinois.edu/courses/IE598/handout/fall2016_slide15.pdf

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

3. Network architecture

(1) Convolutional networks

- Recognition에서 Deep Learning 네트워크의 개발에 따른 다른 분야로의 확장

Table 1: ENet architecture. Output sizes are given for an example input of 512 × 512.

① Name	Type	Output size
initial		16 × 256 × 256
bottleneck1.0	downsampling	64 × 128 × 128
4× bottleneck1.x		64 × 128 × 128
bottleneck2.0	downsampling	128 × 64 × 64
bottleneck2.1	②	128 × 64 × 64
bottleneck2.2	dilated 2	128 × 64 × 64
bottleneck2.3	asymmetric 5	128 × 64 × 64
bottleneck2.4	dilated 4	128 × 64 × 64
bottleneck2.5		128 × 64 × 64
bottleneck2.6	dilated 8	128 × 64 × 64
bottleneck2.7	asymmetric 5	128 × 64 × 64
bottleneck2.8	dilated 16	128 × 64 × 64
Repeat section 2, without bottleneck2.0		
bottleneck4.0	upsampling	64 × 128 × 128
bottleneck4.1		64 × 128 × 128
bottleneck4.2		64 × 128 × 128
bottleneck5.0	upsampling	16 × 256 × 256
bottleneck5.1		16 × 256 × 256
fullconv		C × 512 × 512

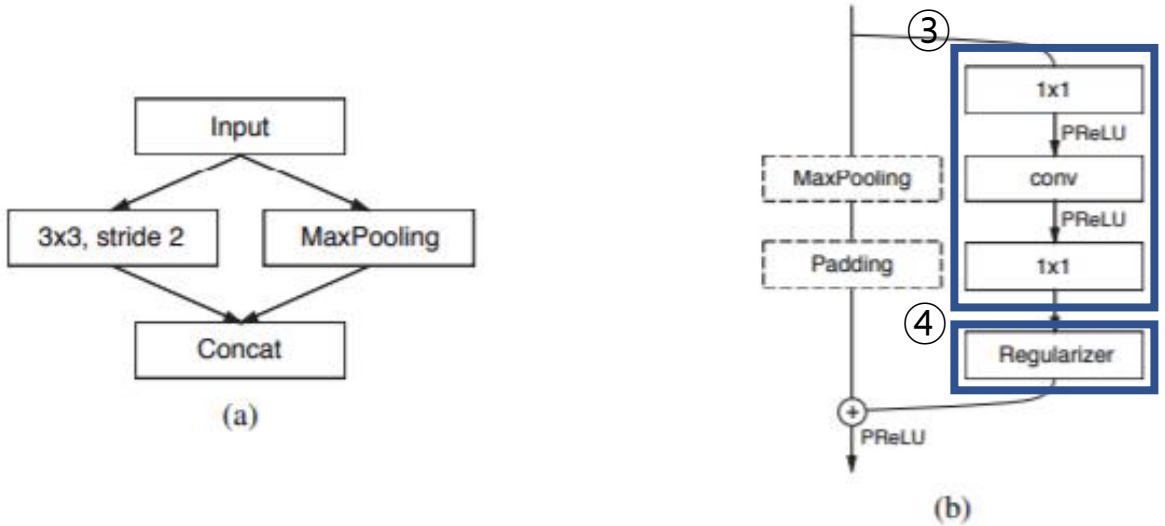


Figure 2: (a) ENet initial block. MaxPooling is performed with non-overlapping 2 × 2 windows, and the convolution has 13 filters, which sums up to 16 feature maps after concatenation. This is heavily inspired by [28]. (b) ENet bottleneck module. conv is either a regular, dilated, or full convolution (also known as deconvolution) with 3 × 3 filters, or a 5 × 5 convolution decomposed into two asymmetric ones.

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

3. Network architecture

(2) BottleNeck

Initial BottleNeck

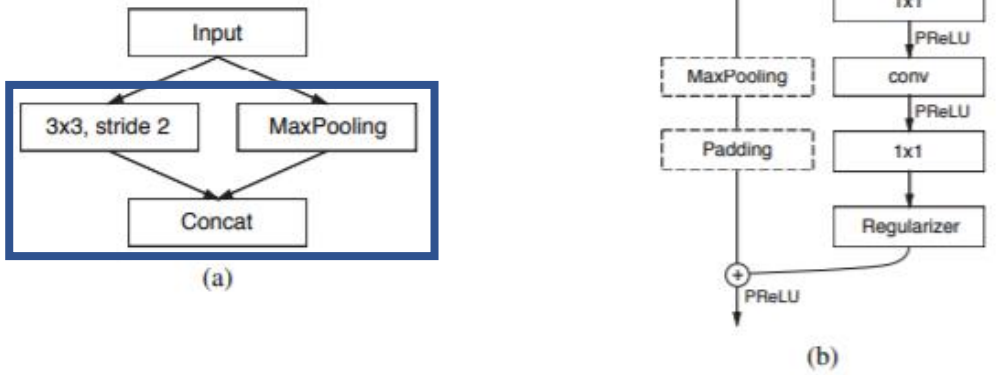


Figure 2: (a) ENet initial block. MaxPooling is performed with non-overlapping 2×2 windows, and the convolution has 13 filters, which sums up to 16 feature maps after concatenation. This is heavily inspired by [28]. (b) ENet bottleneck module. conv is either a regular, dilated, or full convolution (also known as deconvolution) with 3×3 filters, or a 5×5 convolution decomposed into two asymmetric ones.

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

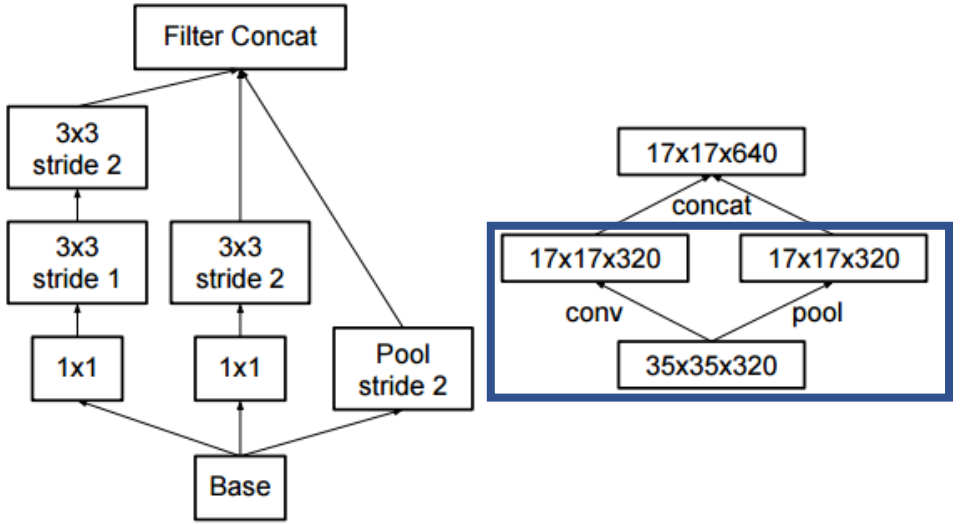


Figure 10. Inception module that reduces the grid-size while expands the filter banks. It is both cheap and avoids the representational bottleneck as is suggested by principle 1. The diagram on the right represents the same solution but from the perspective of grid sizes rather than the operations.

출처 : Rethinking the Inception Architecture for Computer Vision [28]

장점 : Inception module을 이용하면 Representation을 보존하면서도 연산량을 감소할 수 있음. (단순하게 Maxpooling을 이용하면 연산량은 감소하지만 피쳐의 Representation도 같이 감소하는 문제를 해결)

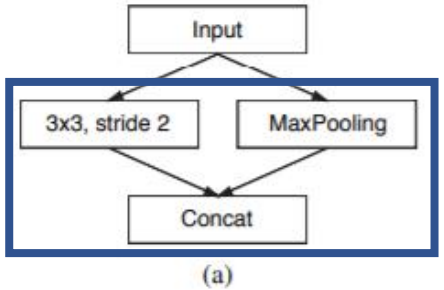
3. Network architecture

(2) BottleNeck

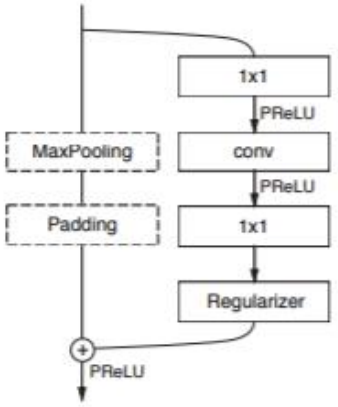
Initial BottleNeck

Table 1: ENet architecture. Output sizes are given for an example input of 512 × 512.

Name	Type	Output size
initial		16 × 256 × 256



(a)



(b)

Figure 2: (a) ENet initial block. MaxPooling is performed with non-overlapping 2 × 2 windows, and the convolution has 13 filters, which sums up to 16 feature maps after concatenation. This is heavily inspired by [28]. (b) ENet bottleneck module. conv is either a regular, dilated, or full convolution (also known as deconvolution) with 3 × 3 filters, or a 5 × 5 convolution decomposed into two asymmetric ones.

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

```python
class InitialBlock(nn.Module):
    def __init__(self, in_channels, out_channels, bias=False):
        super(InitialBlock, self).__init__()
        self.conv = nn.Conv2d(in_channels=in_channels, out_channels=out_channels-3,
                              kernel_size=3, stride=2, padding=1, bias=bias)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        self.bn = nn.BatchNorm2d(out_channels)
        self.prelu = nn.PReLU()

    def forward(self, x):
        conv = self.conv(x)
        maxpool = self.maxpool(x)
        out = torch.cat((conv, maxpool), axis=1)
        out = self.bn(out)
        out = self.prelu(out)
        return out
```

3. Network architecture

(2) BottleNeck

BottleNeck

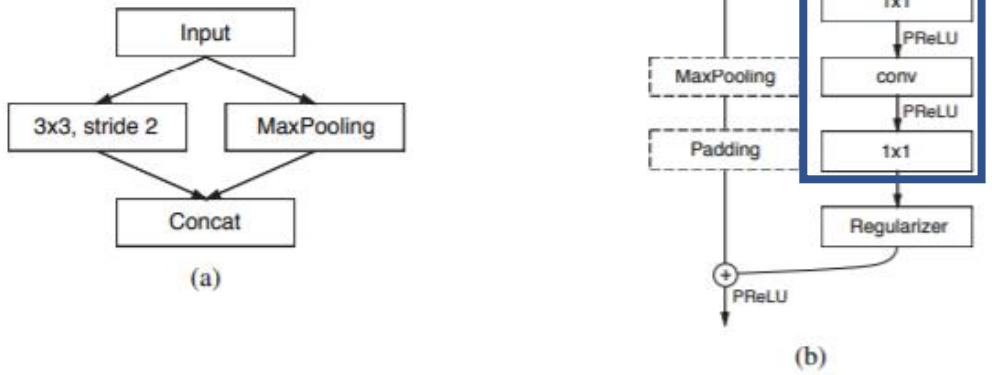
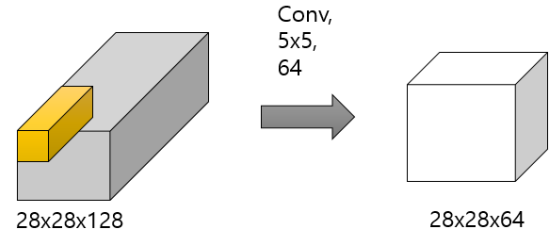


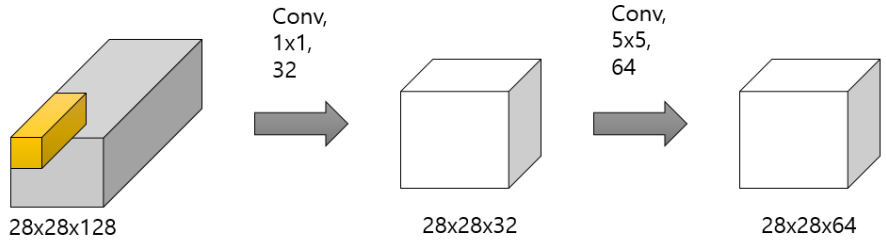
Figure 2: (a) ENet initial block. MaxPooling is performed with non-overlapping 2×2 windows, and the convolution has 13 filters, which sums up to 16 feature maps after concatenation. This is heavily inspired by [28]. (b) ENet bottleneck module. conv is either a regular, dilated, or full convolution (also known as deconvolution) with 3×3 filters, or a 5×5 convolution decomposed into two asymmetric ones.

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

참고자료 : https://stats.stackexchange.com/questions/194142/what-does-1x1-convolution-mean-in-a-neural-network



#params = 28x28x64x5x5x128 = 160M



#params = 28x28x32x128x1x1 = 4.8M

#params = 28x28x64x5x5x32 = 40M

#total = 44.8M

출처 : https://hwiyong.tistory.com/45

3. Network architecture

(2) BottleNeck

BottleNeck

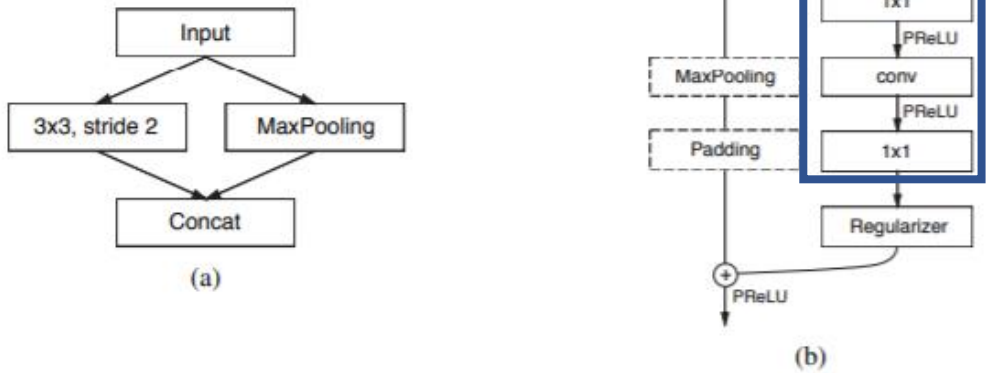
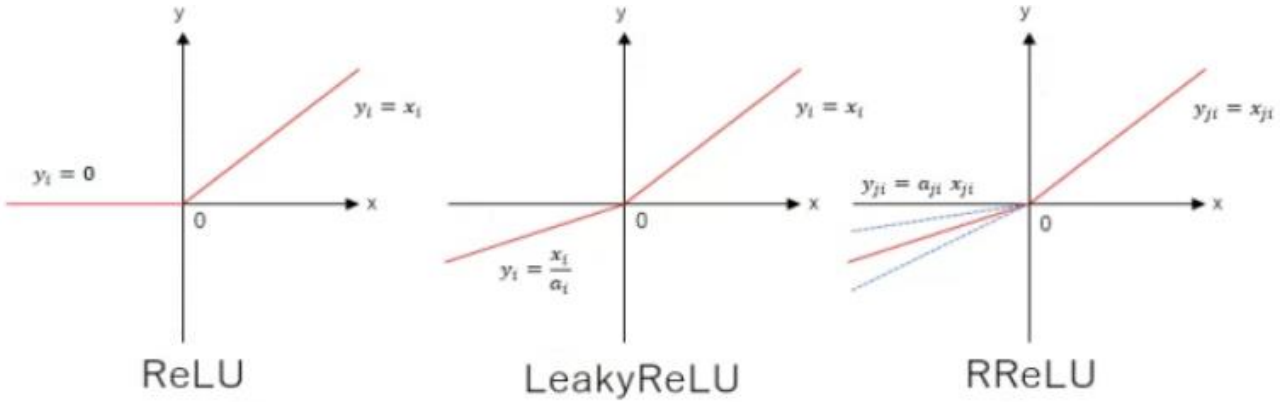


Figure 2: (a) ENet initial block. MaxPooling is performed with non-overlapping 2×2 windows, and the convolution has 13 filters, which sums up to 16 feature maps after concatenation. This is heavily inspired by [28]. (b) ENet bottleneck module. conv is either a regular, dilated, or full convolution (also known as deconvolution) with 3×3 filters, or a 5×5 convolution decomposed into two asymmetric ones.

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

PReLU : alpha : 파라미터로 두어서 모델이 스스로 학습하도록 하는 컨셉

참고자료 : Comparison of Performance by Activation Functions on Deep Image Prior



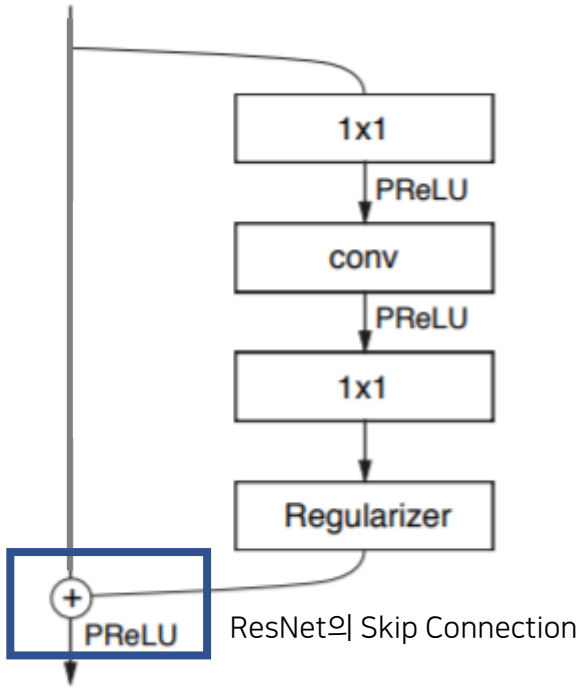

```
self.ext_branch = nn.Sequential(
    nn.Conv2d(in_channels, mid_channels, kernel_size=1,
              stride=1, bias=bias),
    nn.BatchNorm2d(mid_channels),
    nn.PReLU(),
    nn.Conv2d(mid_channels, mid_channels, kernel_size=3,
              stride=1, bias=bias, padding=1),
    nn.BatchNorm2d(mid_channels),
    nn.PReLU(),
    nn.Conv2d(mid_channels, out_channels, kernel_size=1,
              stride=1, bias=bias),
    nn.BatchNorm2d(out_channels),
    nn.PReLU(),
    nn.Dropout2d(p=prob)
```

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

3. Network architecture

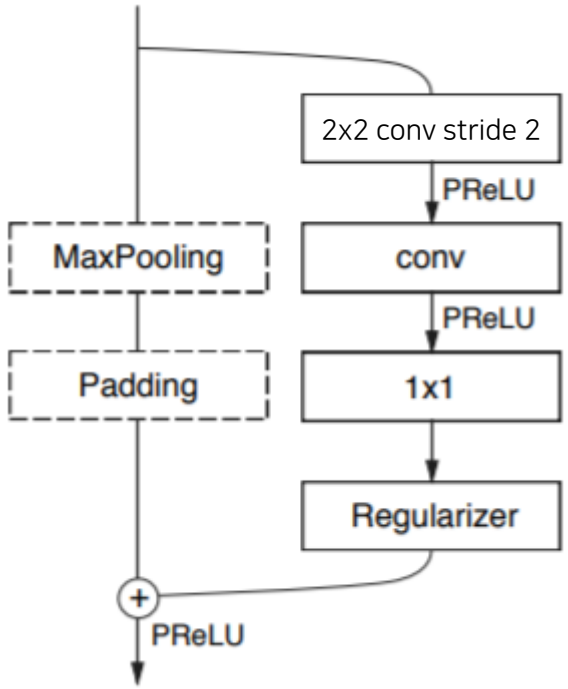
(2) BottleNeck

BottleNeck with Downsampling



ResNet의 Skip Connection

[기본적인 BottleNeck 구조]



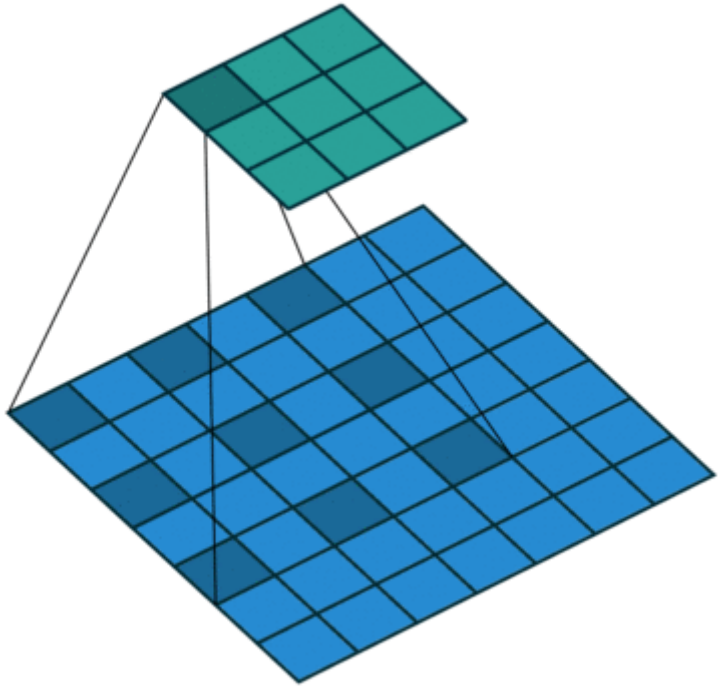
[Downsample시 BottleNeck 구조]

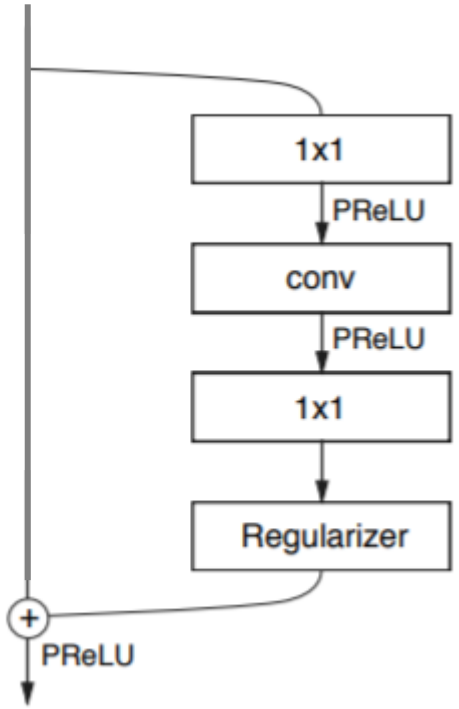
출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

3. Network architecture

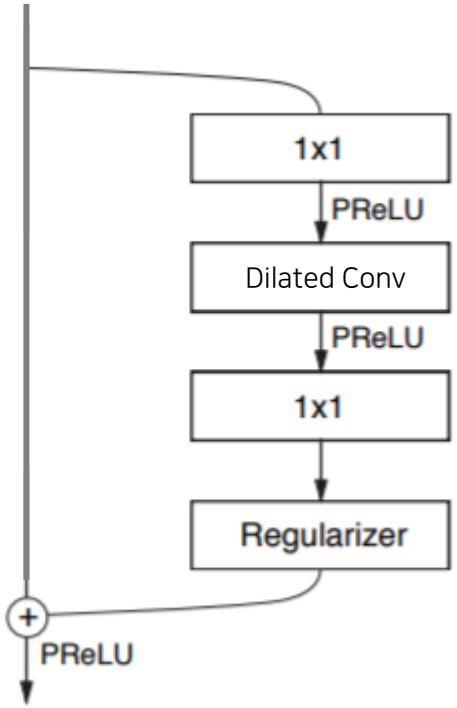
(2) BottleNeck

BottleNeck with dilated





[기본적인 BottleNeck 구조]



[Dilated BottleNeck 구조]

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

3. Network architecture

(2) BottleNeck

BottleNeck with asymmetric

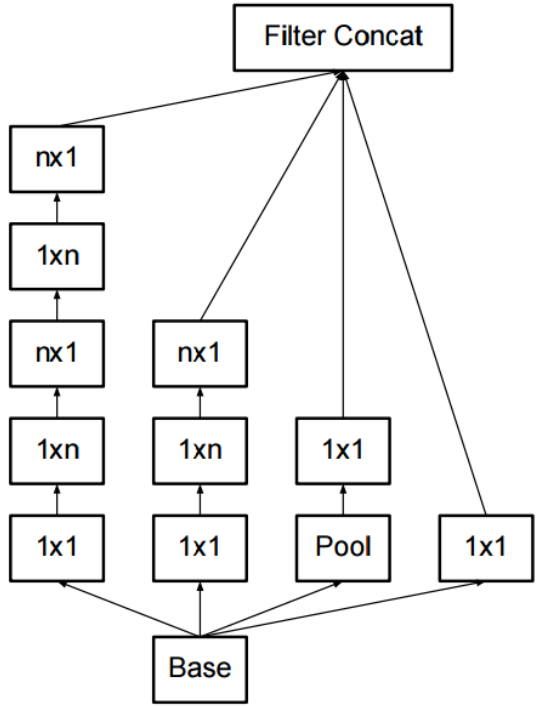
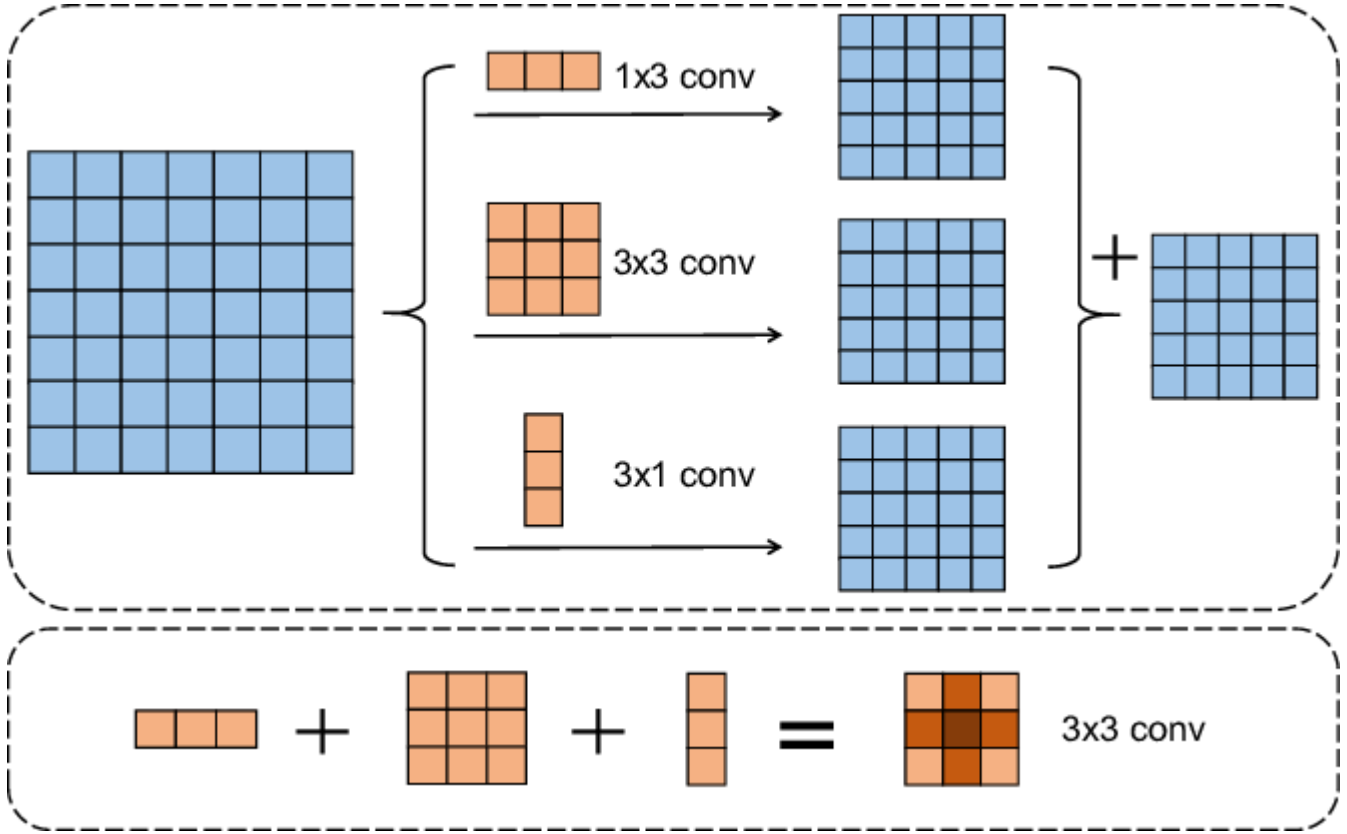


Figure 6. Inception modules after the factorization of the $n \times n$ convolutions. In our proposed architecture, we chose $n = 7$ for the 17×17 grid. (The filter sizes are picked using principle 3)

출처 : Rethinking the Inception Architecture for Computer Vision [28]

참고자료 : https://coding-yoon.tistory.com/122



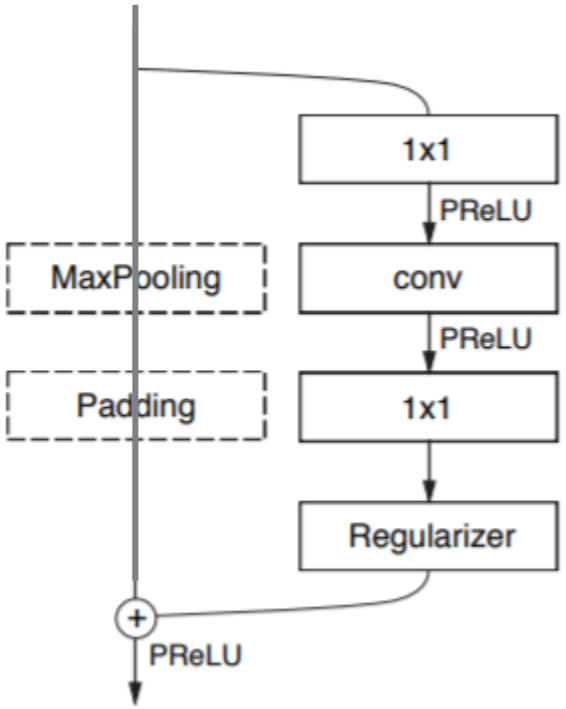
출처 : https://www.researchgate.net/figure/Illustration-of-asymmetric-convolution-blocks-traditional-3-3-layer-is-replace-with_fig2_344620670

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

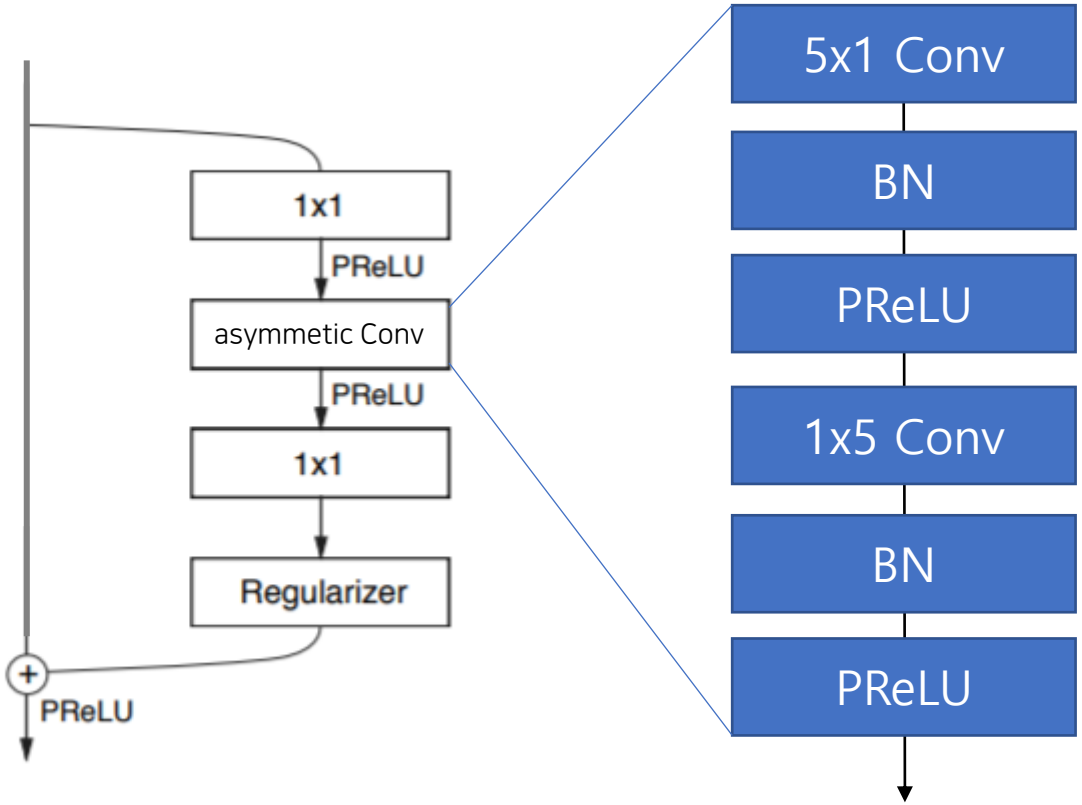
3. Network architecture

(2) BottleNeck

BottleNeck with asymmetric



[기본적인 BottleNeck 구조]



출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

3. Network architecture
(2) BottleNeck

Regularizer – Spatial Dropout

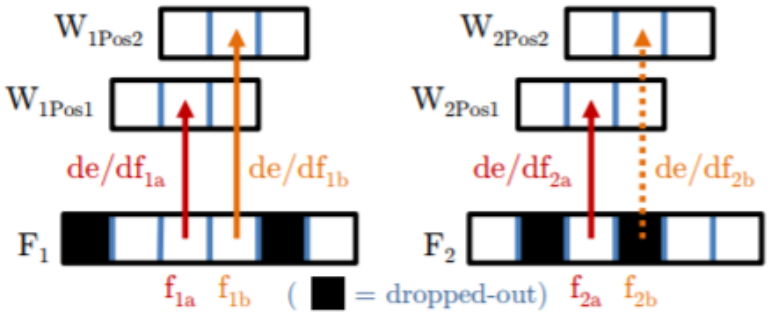


Figure 3: Standard Dropout after a 1D convolution layer

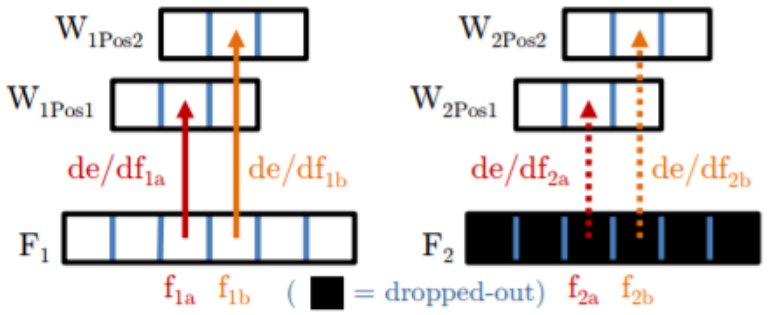


Figure 5: *SpatialDropout* after a 1D convolution layer

출처 : Efficient Object Localization Using Convolutional Networks

SpatialDropout : Feature Map 전체를 Dropout 시키는 방법

In initial experiments, we found that applying standard dropout (where each convolution feature map activation is “dropped-out” independently) before the 1 × 1 convolution layer generally increased training time but did not prevent over-training. **Since our network is fully convolutional and natural images exhibit strong spatial correlation, the feature map activations are also strongly correlated, and in this setting standard dropout fails. [27]**

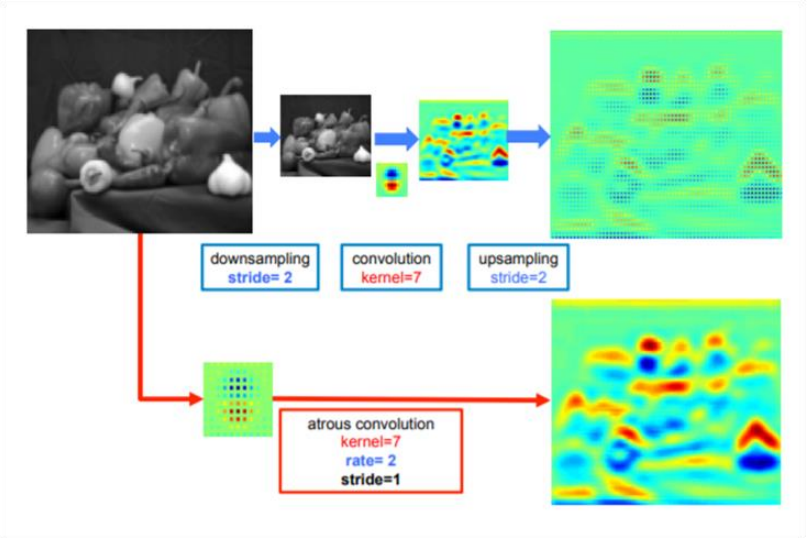
출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

4. Design choices
(1) Feature map resolution

Feature map resolution

Downsamping에 의한 두가지 한계

- Spatial information의 감소 (Edge)
- Output 자체는 Input하고 동일한 resolution을 가져야함

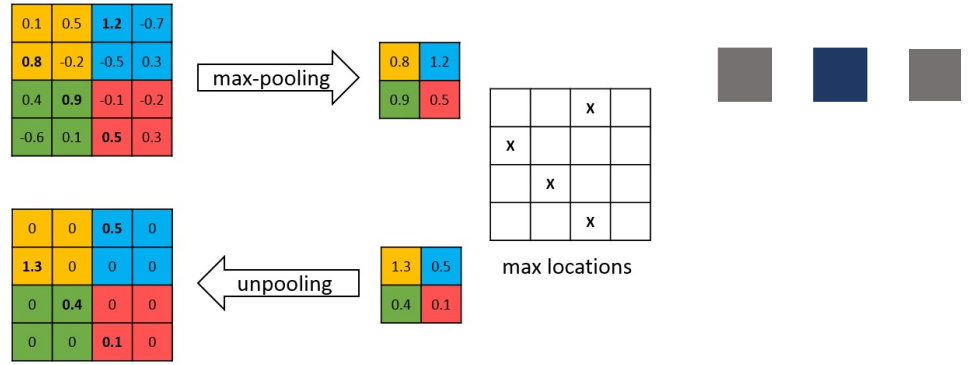


출처 : DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

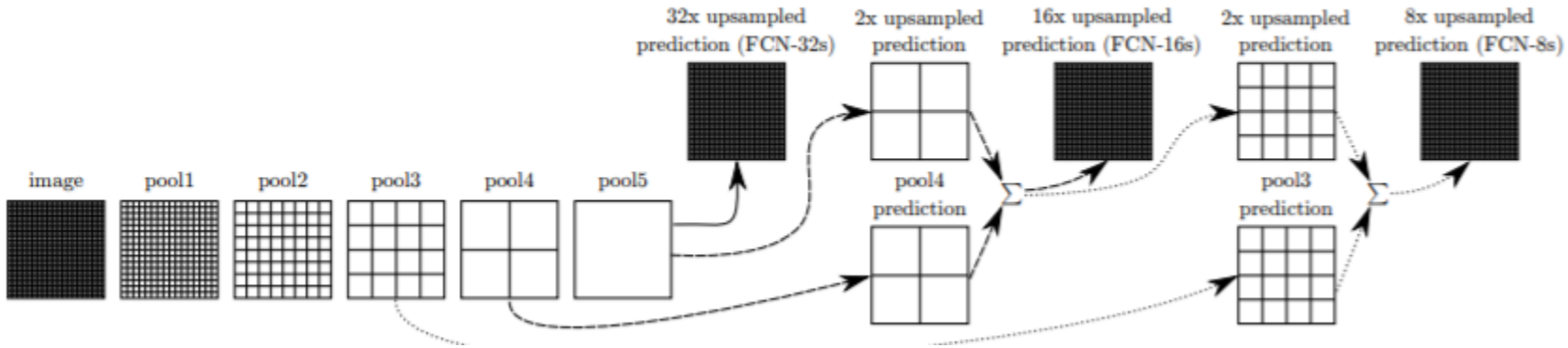
4. Design choices
(1) Feature map resolution

Feature map resolution



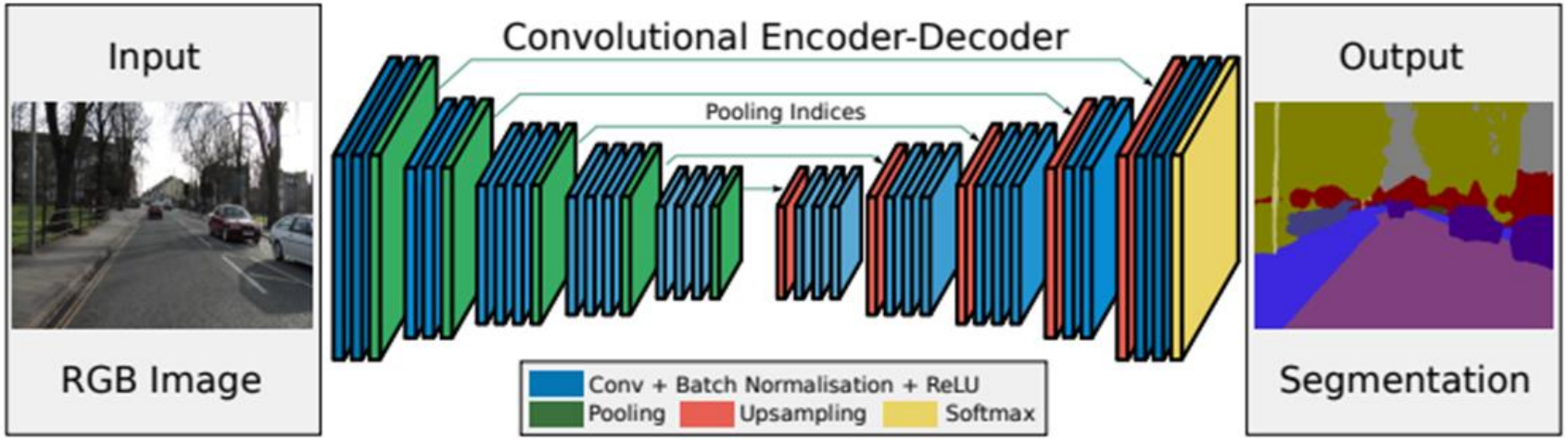
Unpooling 개념, 출처 : DeepPainter: Painter Classification Using Deep Convolutional Autoencoders

출처 : Fully Convolutional Networks for Semantic Segmentation



FCN은 이전 Feature Map을 Sum 하는 형태로 Downsampling에 의해서 잃어버린 spatial resolution을 복원

출처 : SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation



SegNet은 Unpooling이라는 컨셉과 Encoder-Decoder의 반복되는 Transposed Conv를 통해서 잃어버린 spatial resolution을 복원 **<- 하지만 아직 정확도는 낮음**

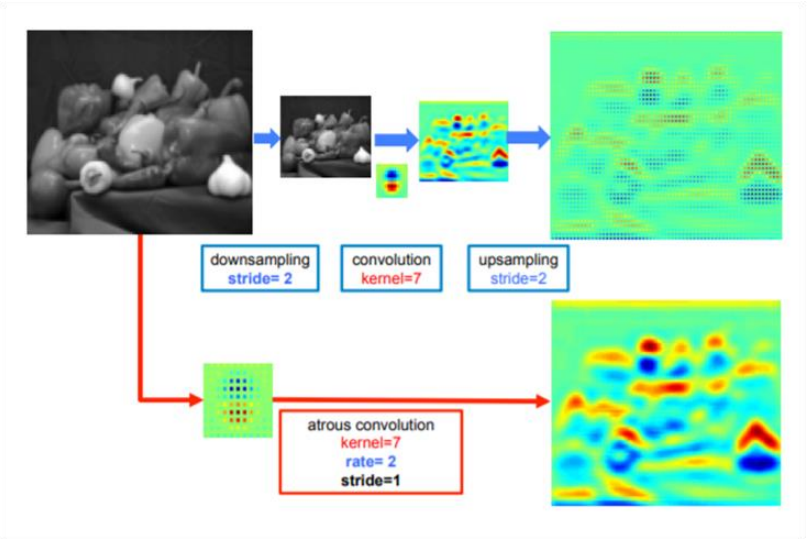
출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

4. Design choices
(1) Feature map resolution

Feature map resolution

하지만, Downsampling 자체가 성능은 해치지만 Segmentation에서 무시할 수 없는 두가지 장점이 존재

- 입력 이미지의 Resolution을 감소시켜서 Receptive Field를 넓힘 (도로와 라이더의 관계를 한번에 살펴봄으로서 객체간의 관계를 파악하는 장점이 있음)
- 메모리의 관점에서 거의 4배를 감소시키는 장점이 있음



DeepLabv2에서 언급된 것 처럼 Dilated Conv을 이용해서 이를 해결하려는 시도를 접목

출처 : DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

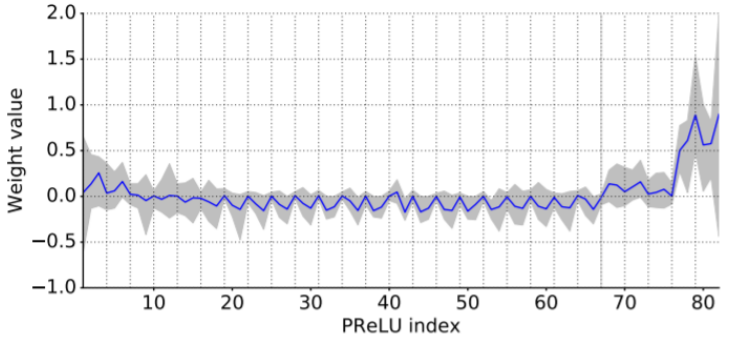
4. Design choices
(2) Early downsampling

Early downsampling

Initial Block에서 512x512 이미지를 256x256으로 감소시키면서 채널을 증가시키면서 real-time operation을 가능하게 만듬 (개인적인 생각으로는 이게 Contribution이 좋았던 것 같음)

Table 1: ENet architecture. Output sizes are given for an example input of 512 × 512.

Name	Type	Output size
initial		16 × 256 × 256



공간정보가 매우 중복되기에 압축해서 표현하는게 효율적이고, 초기 네트워크는 분류에 직접적으로 관여하는 대신에 피쳐 추출기의 역할을 하고 마지막 layer가 분류 역할을 수행

-> 이게 맞는지에 대한 의문점은 있음

-> 1. Feature map resolution에서 압축하는게 성능이 감소한다고 한 것 같은데 이거랑 상충

-> 2. 초기 네트워크가 분류에 직접적으로 관여하면 안되는지에 대한 직관적인 그게 와닿지 않음

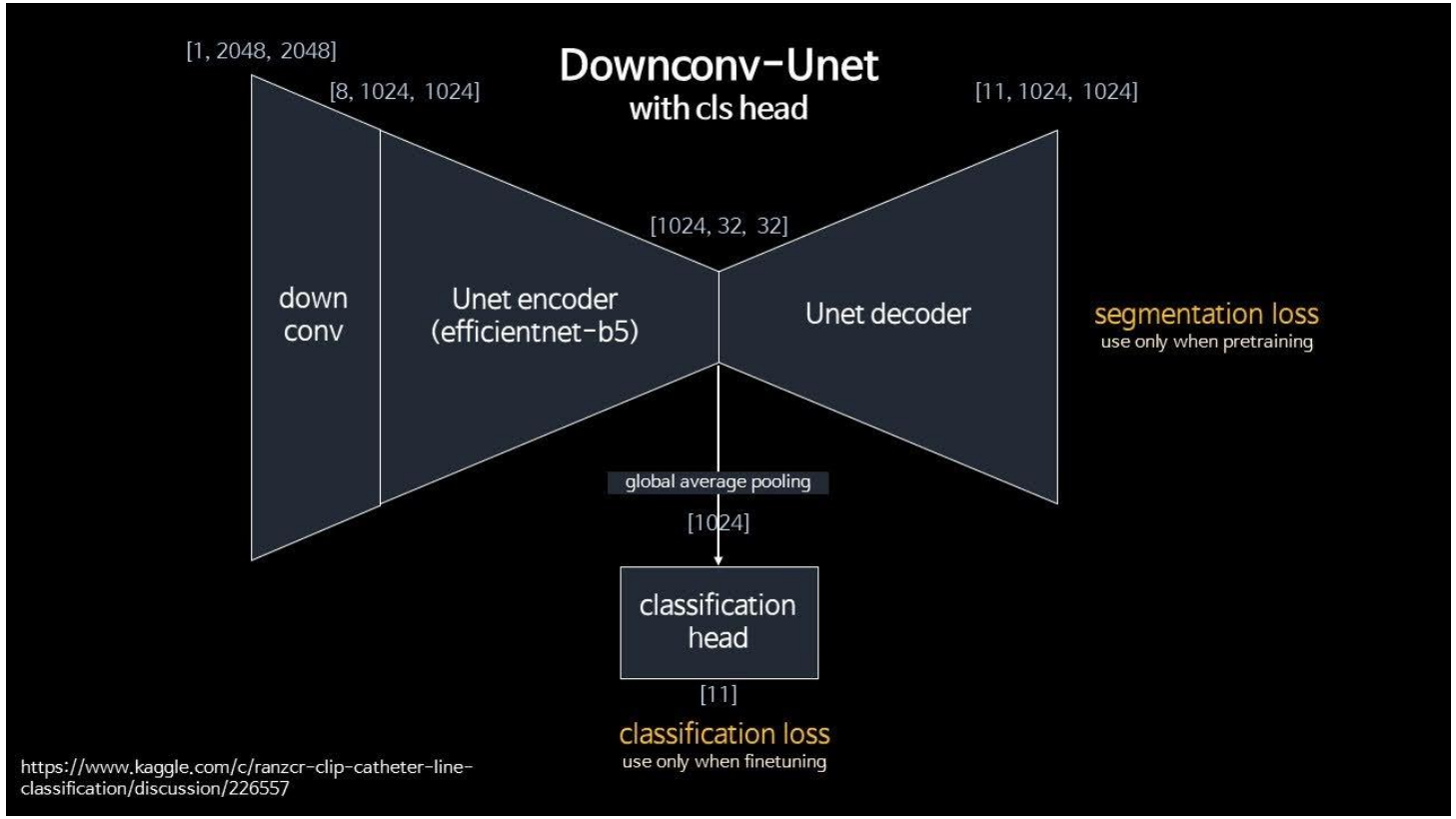
출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

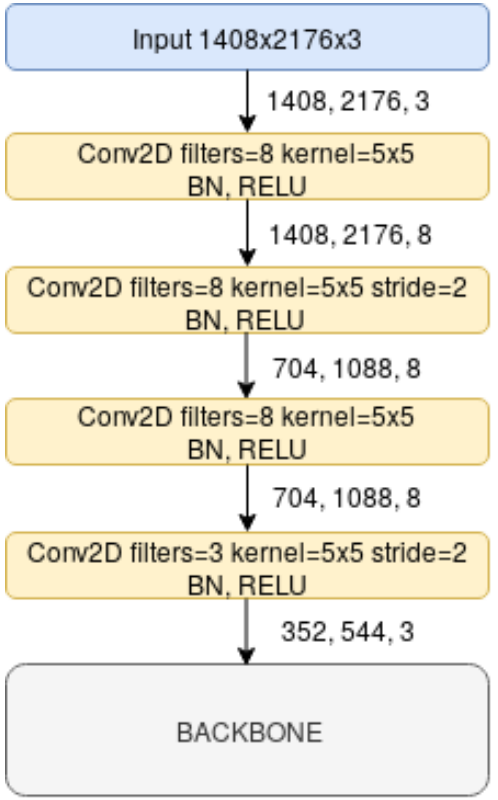
4. Design choices
(2) Early downsampling

Early downsampling

실용적인 측면 [Kaggle] – Ranzcr 11st, Clouds Sattellite 2nd Solution

출처 :
[왼쪽] https://www.kaggle.com/c/ranzcr-clip-catheter-line-classification/discussion/226557
[오른쪽] https://www.kaggle.com/c/understanding_cloud_organization/discussion/118255





출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

4. Design choices
(3) Decoder size

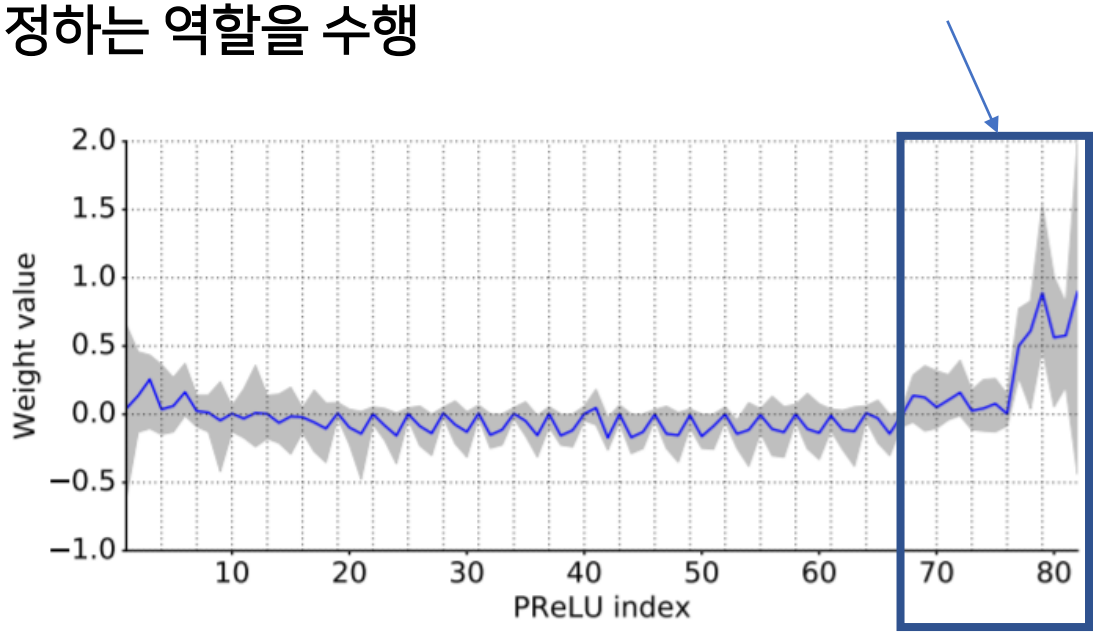
Decoder size

Table 1: ENet architecture. Output sizes are given for an example input of 512 × 512.

Name	Type	Output size
initial		16 × 256 × 256
bottleneck1.0	downsampling	64 × 128 × 128
4× bottleneck1.x		64 × 128 × 128
bottleneck2.0	downsampling	128 × 64 × 64
bottleneck2.1		128 × 64 × 64
bottleneck2.2	dilated 2	128 × 64 × 64
bottleneck2.3	asymmetric 5	128 × 64 × 64
bottleneck2.4	dilated 4	128 × 64 × 64
bottleneck2.5		128 × 64 × 64
bottleneck2.6	dilated 8	128 × 64 × 64
bottleneck2.7	asymmetric 5	128 × 64 × 64
bottleneck2.8	dilated 16	128 × 64 × 64
Repeat section 2, without bottleneck2.0		
bottleneck4.0	upsampling	64 × 128 × 128
bottleneck4.1		64 × 128 × 128
bottleneck4.2		64 × 128 × 128
bottleneck5.0	upsampling	16 × 256 × 256
bottleneck5.1		16 × 256 × 256
fullconv		C × 512 × 512

기존의 DeconvNet이나 SegNet, Unet 등의 연구와는 대비되게 Decoder와 Encoder가 대칭이 되지 않는 형태

- the encoder should be able to work in a similar fashion to original classification architectures, i.e. to operate on smaller resolution data and provide for information processing and filtering. Instead, the role of the the decoder, is to upsample the output of the encoder, only fine-tuning the details
- 인코더는 Feature 추출을 **디코더는 Upsample의 결과를 미세하게 조정하는 역할을 수행**



출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

4. Design choices

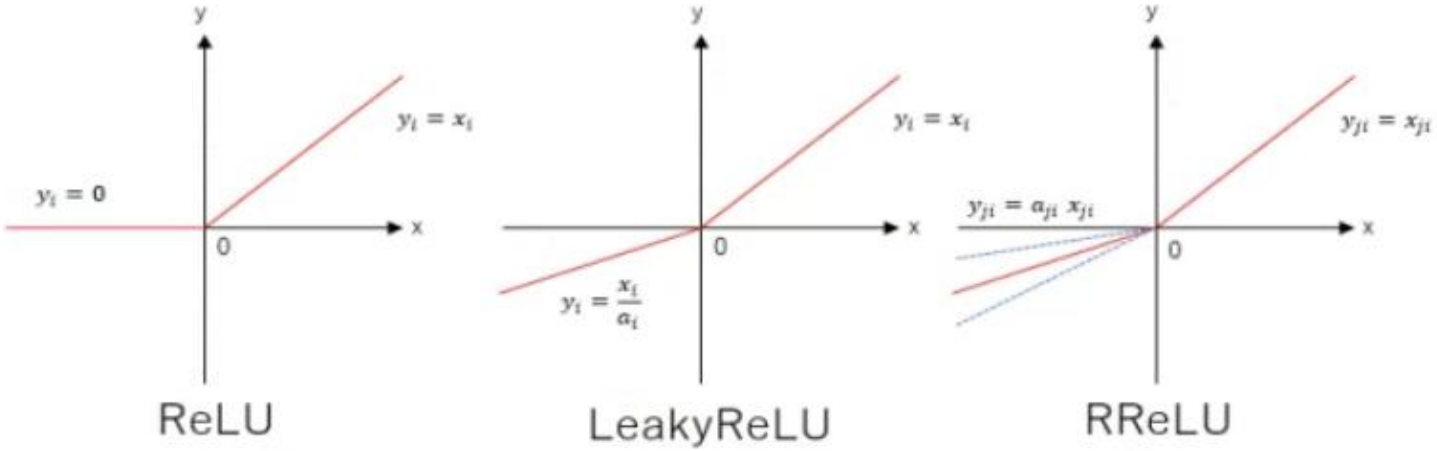
(4) Nonlinear operations

Nonlinear operations

*dying ReLU : ReLu에서 음수의 값을 가지는 값들이 들어와서 해당 노드에 0이 곱해져서 모두 죽어버리는 현상 (참고자료 : https://brunch.co.kr/@kdh7575070/27)

1. ReLU와 BN을 사용하는게 일반적으로 성능이 좋아진다고 알려졌지만 큰 영향이 없었음 (오히려 초기 Layer의 ReLU를 제거하는게 성능향상에 더 큰 영향을 미침)
2. ReLU 대신에 PReLU를 사용해서 이에 대한 Weight도 같이 학습하는게 성능향상에 더 좋았음 (goal of learning the negative slope of non-linearities)
   - Weight가 1에 가까우면 그대로 정보를 전달
   - 0에 가까우면 아예 전달을 안하는 형식
   - 음수인 경우에는 dying ReLU 문제를 해결하고 학습 속도가 빨라짐
   - (참고자료 : https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7)

참고자료 : Comparison of Performance by Activation Functions on Deep Image Prior

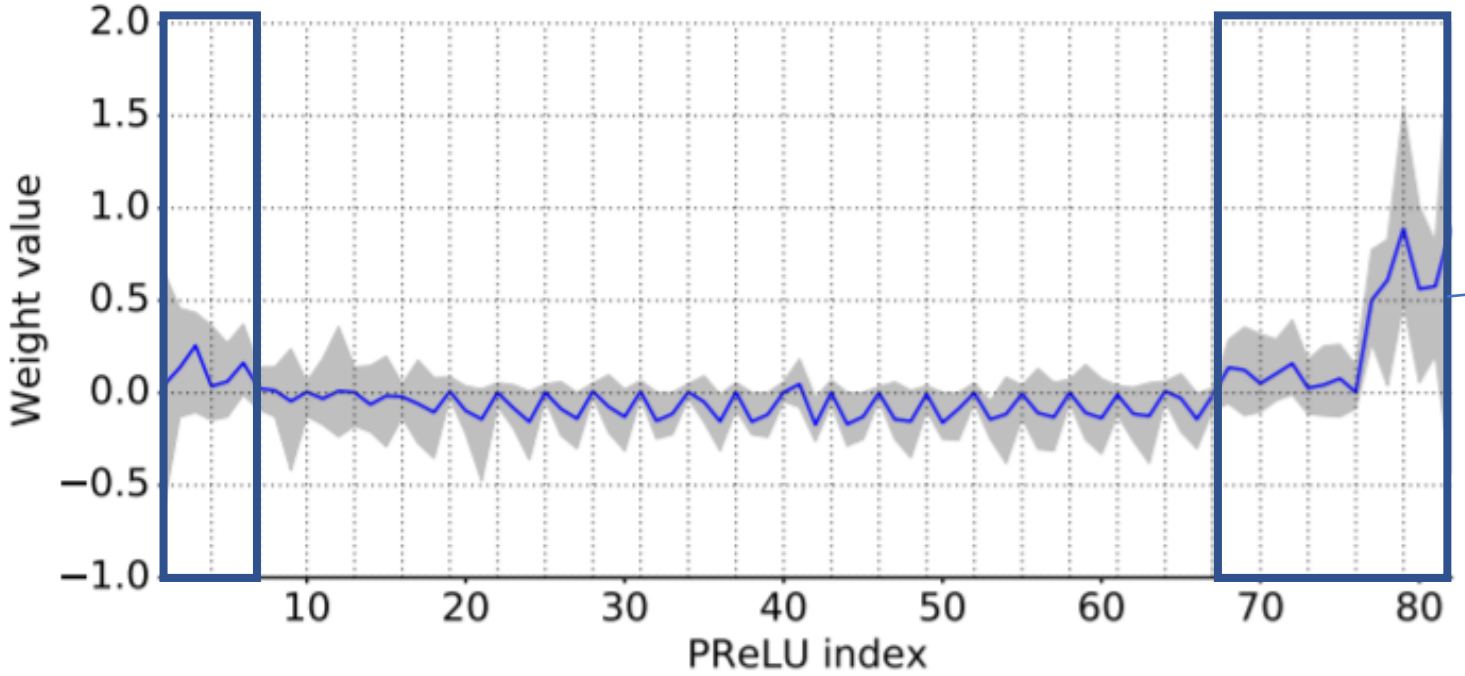


근데, 해당 문제에 대해서는 여러 의견이 있는데 별 성능 차이가 없다는 얘기가 더 많은 것 같습니다. 사용하는 경우도 많이 못 본 것 같구요.

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

4. Design choices
(4) Nonlinear operations

Nonlinear operations



디코더는 마지막은 Fine-Tune 역할

1. Encoder의 초기 Layer와 Decoder Layer의 경우 weight가 양수 값을 가지고 Encoder의 중간 부분부터 마지막은 0에서부터 음수의 값을 가짐
2. **Identity가 잘 작동하지 않고, 음수로 이렇게 작동하는 이슈가 생긴 이유는 ResNet과는 다르게 레이어가 너무 얕아서 발생한 이슈로 저자는 판단**

   -> 무슨 말인지 모르겠음..

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

4. Design choices

(5) Information-preserving dimensionality changes

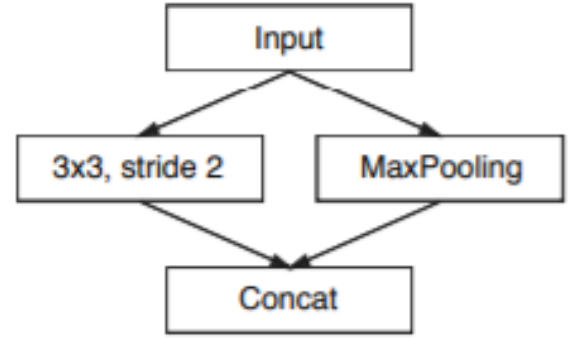
Information-preserving dimensionality changes

1. Pooling -> Conv : 비용은 적지만 *representational bottleneck 현상이 발생
2. Conv -> Pooling : Feature map depth는 늘어나지만, 비용이 상승하는 문제가 발생

-> 그래서, 병렬적으로 두가지를 수행해서 Concatnate 하는게 가장 효과적

*representational bottleneck : CNN에서 주로 사용되는 pooling으로 인해 feature map의 size가 줄어들면서 정보량이 줄어드는 것을 의미

출처 : https://hoya012.github.io/blog/deeplearning-classification-guidebook-2/



출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

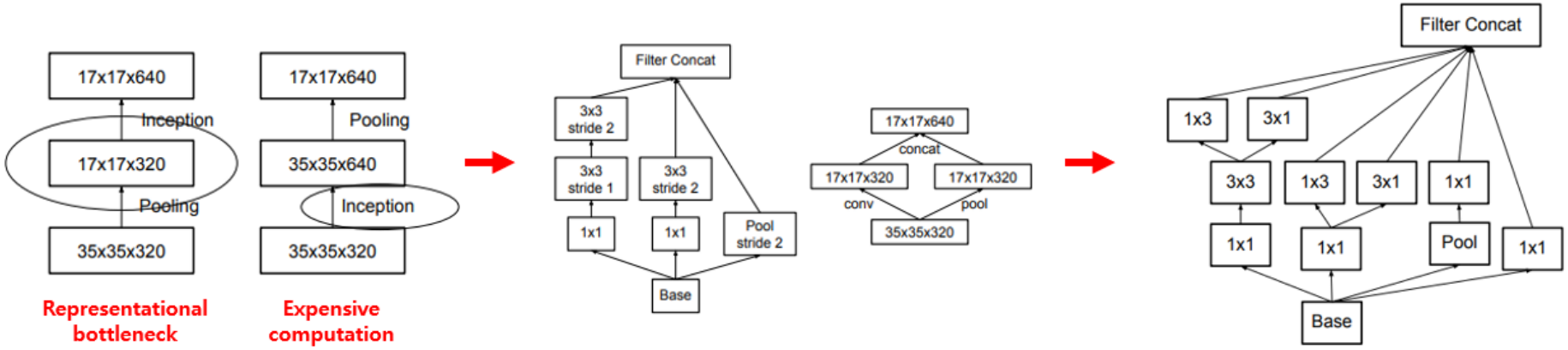


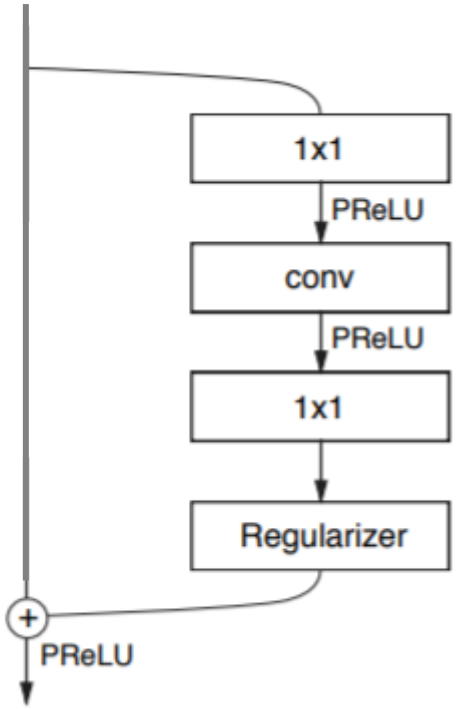
Figure 7

출처 : Rethinking the Inception Architecture for Computer Vision [28]
출처 : https://hoya012.github.io/blog/deeplearning-classification-guidebook-2/

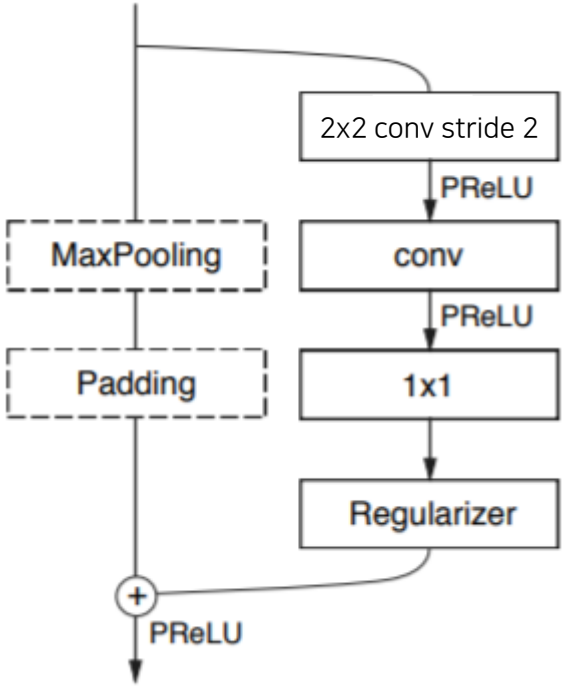
4. Design choices

(5) Information-preserving dimensionality changes

Information-preserving dimensionality changes



[기본적인 BottleNeck 구조]



[Downsampling시에 BottleNeck 구조]

1x1 projectio의 경우 정보의 75%가 사라지는 반면, 2x2는 100% 효과적으로 활용이 가능 (단 4배 만큼의 자원 소모가 늘어나지만, Downsampling이 2번 밖에 없어서 괜찮음)

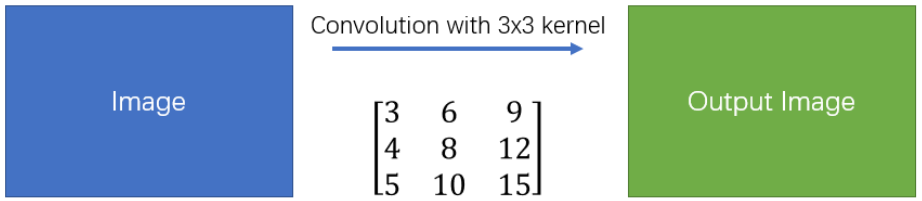
출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

4. Design choices
(6) Factorizing filters

Factorizing filters (asymmetric convolutions)

1. 3 x 3 convolutions -> 5 x 1 + 1 x 5 convolutions
   - 비슷한 weight의 개수를 가지지만 receptive field 관점에서는 더 증가한 형태를 가짐
   - *low-rank approximation 에 의해서 속도 상승을 가능하게함
   - 그리고 분리된 Convolution 사이에 non-linear operation을 넣을 수 있음

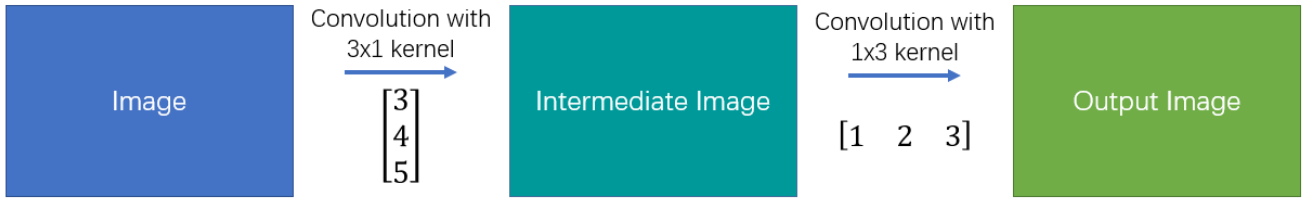
Simple Convolution



*low-rank approximation : filter를 decompose 하는 것 (kernel이 아니라 filter임을 유의하자. kernel은 (WxH)의 2D array이고, filter는 채널까지 포함한 3D array를 일컫는다.)

출처 : https://blogik.netlify.app/BoostCamp/U_stage/48_filter_decompostion/

Spatial Separable Convolution



출처 : https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728

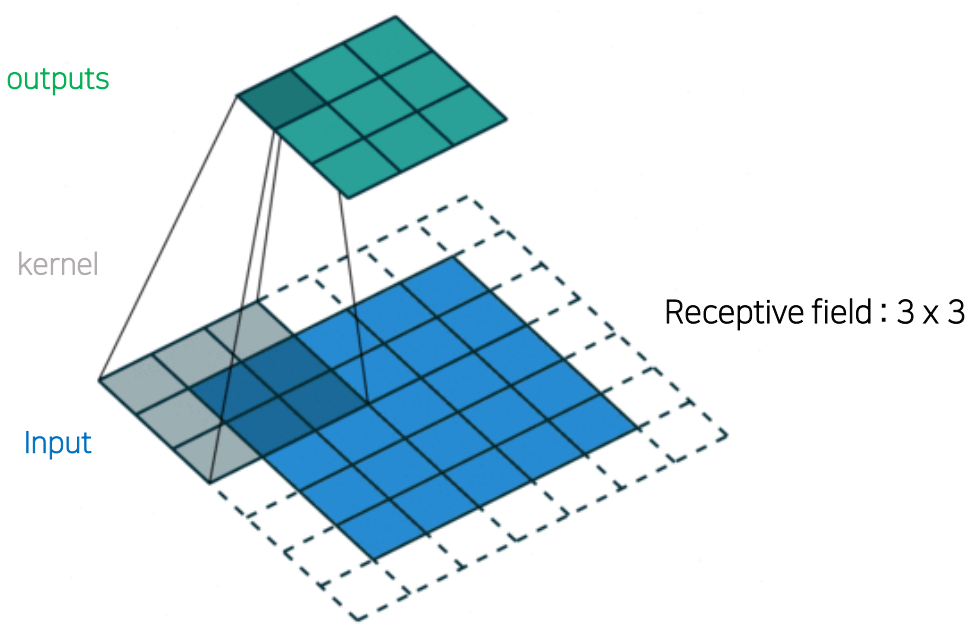
출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

4. Design choices
(7) Dilated convolutions

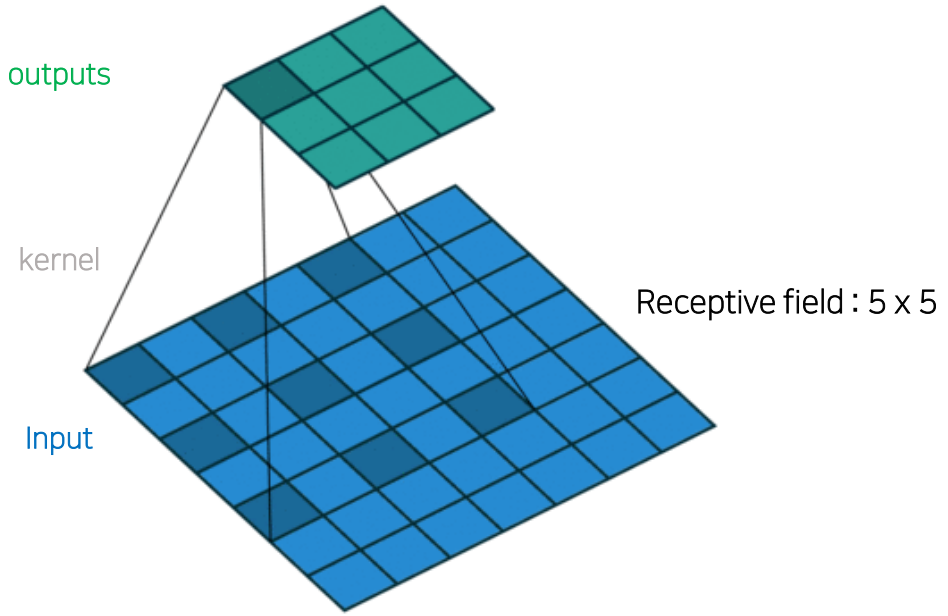
Dilated convolutions

1. Receptive Field를 넓혀서 넓은 context를 담을 수 있도록 시도

Standard Convolution



Dilated Convolution (a.k.a. atrous convolution)



출처 : A technical report on convolution arithmetic in the context of deep learning (https://github.com/vdumoulin/conv_arithmetic) (MIT License)

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

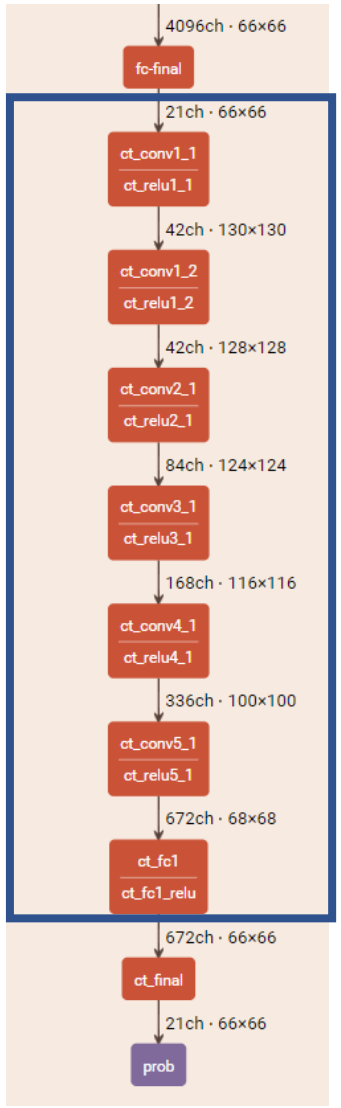
4. Design choices
(7) Dilated convolutions

Dilated convolutions

2. Sequence하게 두는 대신에 Bottleneck 내부에 Dilated Conv를 삽입하는 구조로 진행

Table 1: ENet architecture. Output sizes are given for an example input of 512 × 512.

Name	Type	Output size
initial		16 × 256 × 256
bottleneck1.0	downsampling	64 × 128 × 128
4× bottleneck1.x		64 × 128 × 128
bottleneck2.0	downsampling	128 × 64 × 64
bottleneck2.1		128 × 64 × 64
bottleneck2.2	dilated 2	128 × 64 × 64
bottleneck2.3	asymmetric 5	128 × 64 × 64
bottleneck2.4	dilated 4	128 × 64 × 64
bottleneck2.5		128 × 64 × 64
bottleneck2.6	dilated 8	128 × 64 × 64
bottleneck2.7	asymmetric 5	128 × 64 × 64
bottleneck2.8	dilated 16	128 × 64 × 64
Repeat section 2, without bottleneck2.0		
bottleneck4.0	upsampling	64 × 128 × 128
bottleneck4.1		64 × 128 × 128
bottleneck4.2		64 × 128 × 128
bottleneck5.0	upsampling	16 × 256 × 256
bottleneck5.1		16 × 256 × 256
fullconv		C × 512 × 512



출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

4. Design choices
(8) Regularization

Regularization

1. Pixel-wise segmentation dataset이 비교적 적음
   - 이유 : pixel wise annotation을 하기가 어려움 …
   - 문제 : Overfitting 발생하기가 쉬움

2. 위의 문제를 해결하기 위해서 Regularization 테크닉을 도입
   - L2 weight Decay
   - Dropout (x) -> Spatial Dropout (o)

$$Cost = \frac{1}{n}\sum_{i=1}^{n}\{L(y_i, \hat{y}_i) + \frac{\lambda}{2}|w|\}$$

$L(y_i, \hat{y}_i)$: 기존의 Cost function

[L1 Regularization]

$$Cost = \frac{1}{n}\sum_{i=1}^{n}\{L(y_i, \hat{y}_i) + \frac{\lambda}{2}|w|^2\}$$

[L2 Regularization]

출처 : https://light-tree.tistory.com/125

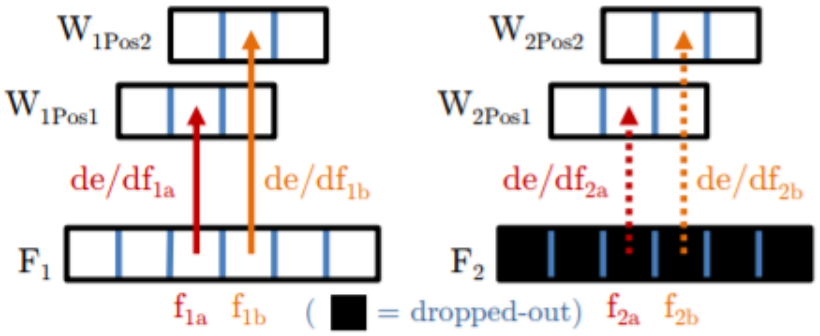


Figure 5: *SpatialDropout* after a 1D convolution layer

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

5. Results
(1) Performance Analysis

Inference Time

Table 2: Performance comparison.

Model	NVIDIA TX1						NVIDIA Titan X					
	480×320		640×360		1280×720		640×360		1280×720		1920×1080	
	ms	fps	ms	fps	ms	fps	ms	fps	ms	fps	ms	fps
SegNet	757	1.3	1251	0.8	-	-	69	14.6	289	3.5	637	1.6
ENet	47	21.1	69	14.6	262	3.8	7	135.4	21	46.8	46	21.6

Table 3: Hardware requirements. FLOPs are estimated for an input of $3 \times 640 \times 360$.

	GFLOPs	Parameters	Model size (fp16)
SegNet	286.03	29.46M	56.2 MB
ENet	3.83	0.37M	0.7 MB

18x Speed Up
75x Less FLOPs
79x Less parameters

ms : 하나의 사진을 Inference 하는데 걸리는 시간 (1/1000 s)
fps : 1s에 inference 하는 사진의 수

예) 1251 / 1000(ms) = 0.8 fps

GFLOPs : GPU Floating point Operations Per Second
Parameters : 파라미터의 수
Model size (fp16) : Quantization 처럼 16bit로 모델 저장해서 크기 잰 Size

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

5. Results
(2) Benchmarks

Settings

- Optimizer : Adam
- 2 stage의 학습을 진행
  - 1 stage : we trained only the encoder to categorize downsampled regions of the input image
  - 2 stage : we appended the decoder and trained the network to perform upsampling and pixel-wise classification

- Lr : 5e-4
- L2 weight decay : 2e-4
- Batch size : 10
- Class weight

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

5. Results
(2) Benchmarks

Cityscapes

Training : 2975 (19 classes)
Validation : 500
Test set : 1525

출처 : https://www.cityscapes-dataset.com/examples/

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

5. Results
(2) Benchmarks

Cityscapes

Table 4: Cityscapes test set results

Model	Class IoU	Class iIoU	Category IoU	Category iIoU
SegNet	56.1	34.2	79.8	**66.4**
ENet	**58.3**	**34.4**	**80.4**	64.0

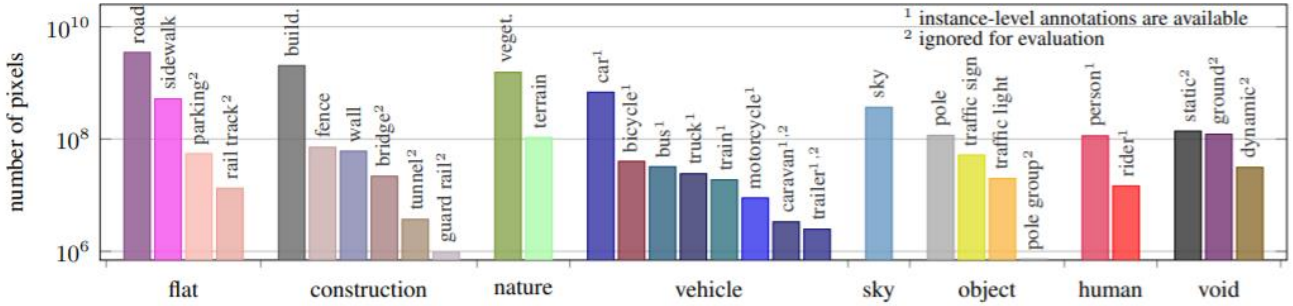


Figure 1. Number of finely annotated pixels (y-axis) per class and their associated categories (x-axis).

출처 : The Cityscapes Dataset for Semantic Urban Scene Understanding

Class IoU : 클래스별 IoU
Class iIoU : 클래스별 iIoU (iIoU는 Instance의 크기를 weight로 한 평가함수임)
Category IoU : 카테고리별 IoU
Category iIoU : 카테고리별 iIoU

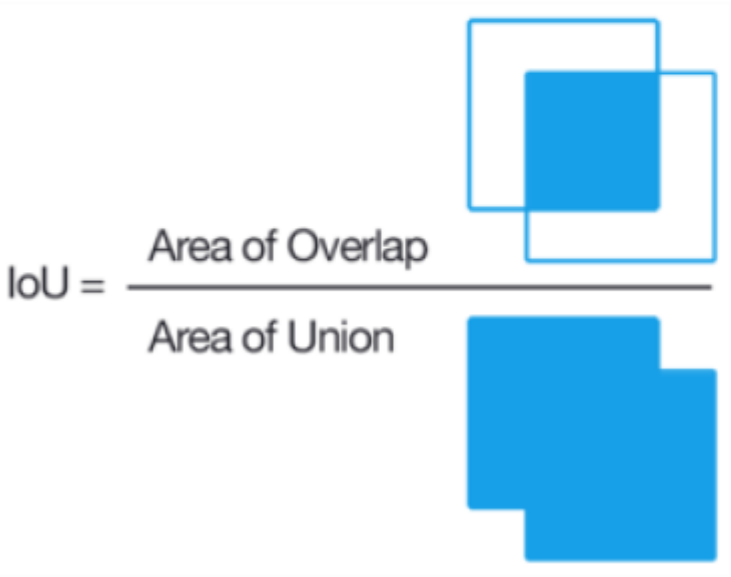
- IoU = TP / (TP+FP+FN)
- iIoU : instance-level intersection over union metric, which is IoU weighed by the average object size. iIoU = iTP / (iTP+FP+iFN). 참고로, iIoU는 IoU가 차지하는 Pixel이 큰 인스턴스에 편향되는 문제가 있어서 이를 해결하기 위한 메트릭임. 이때, i의 경우는 by weighting the contribution of each pixel by the ratio of the class' average instance size to the size of the respective ground truth instance (즉, class' average instance size / size of the respective ground truth instance 으로 계산)

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

5. Results
(2) Benchmarks

Cityscapes

$$\text{mean IU: } (1/n_{\text{cl}}) \sum_i n_{ii} / \left(t_i + \sum_j n_{ji} - n_{ii} \right)$$



출처 : https://towardsdatascience.com/metrics--toto--evaluateevaluate--youryour--semanticsemantic--segmentationsegmentation--modelmodel--6bcb99639aa26bcb99639aa2

5. Results
(2) Benchmarks

Cityscapes

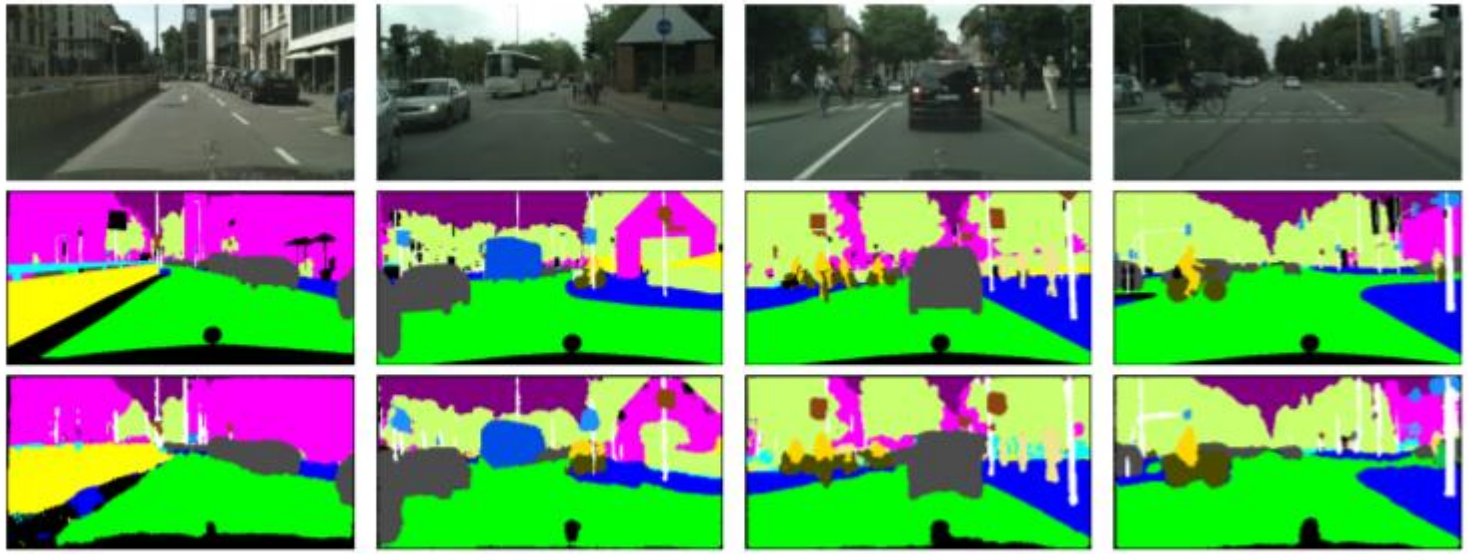


Figure 4: ENet predictions on Cityscapes validation set [14]

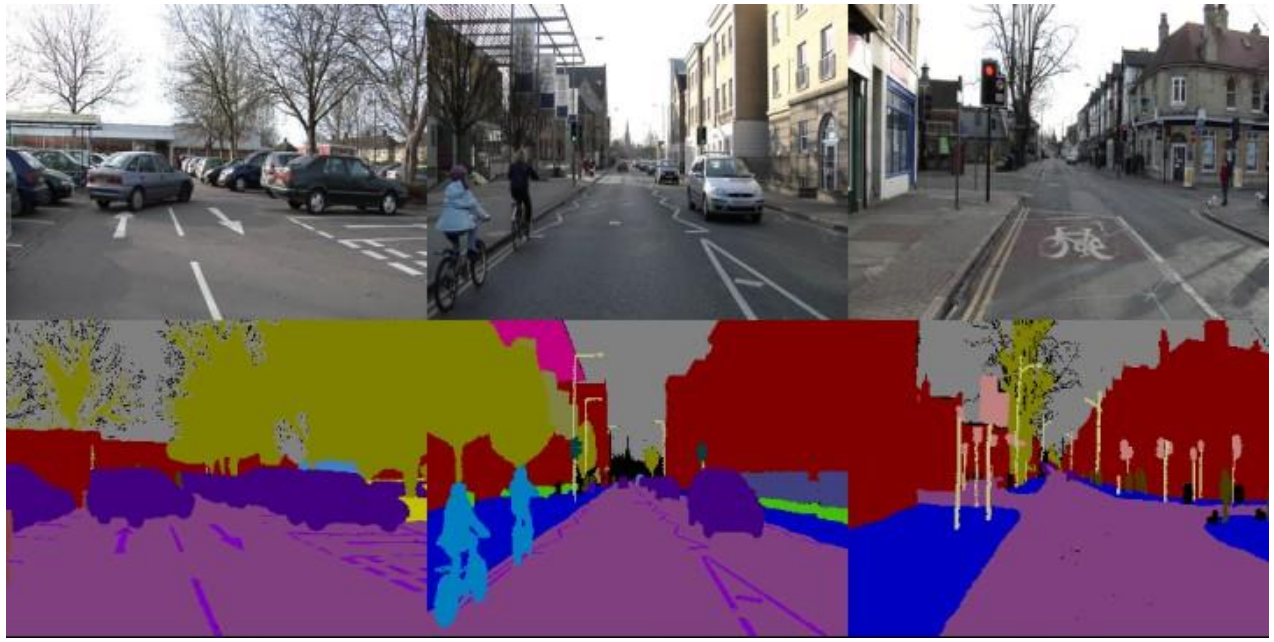
출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

5. Results
(2) Benchmarks

CamVid-Sturgess

데이터 출처 : P. Sturgess, K. Alahari, L. Ladicky, and P. H. Torr, "Combining appearance and structure from motion features for road scene understanding," in BMVC 2012-23rd British Machine Vision Conference. BMVA, 2009

Training : 367 (11 classes)
Valid : 100
Test set : 233
CAMVID를 Sturgess가 Train/Valid/Test로 각각 분류한 데이터라고 함. 클래스도 32개에서 11개로 줄임



출처 : http://mi.eng.cam.ac.uk/research/projects/VideoRec/CamVid/

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

5. Results
(2) Benchmarks

CamVid-Sturgess

Class Avg : 각 픽셀 Accuracy에 대한 평균 (Pixel-wise percentage accuracy)

Table 5: Results on CamVid test set of (1) SegNet-Basic, (2) SegNet, and (3) ENet

Model	Building	Tree	Sky	Car	Sign	Road	Pedestrian	Fence	Pole	Sidewalk	Bicyclist	Class avg.	Class IoU
1	75.0	84.6	91.2	**82.7**	36.9	93.3	55.0	47.5	**44.8**	74.1	16.0	62.9	n/a
2	**88.8**	**87.3**	92.4	82.1	20.5	**97.2**	57.1	49.3	27.5	84.4	30.7	65.2	**55.6**
3	74.7	77.8	**95.1**	82.4	**51.0**	95.1	**67.2**	**51.7**	35.4	**86.7**	**34.1**	**68.3**	51.3

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

5. Results
(2) Benchmarks

CamVid-Sturgess

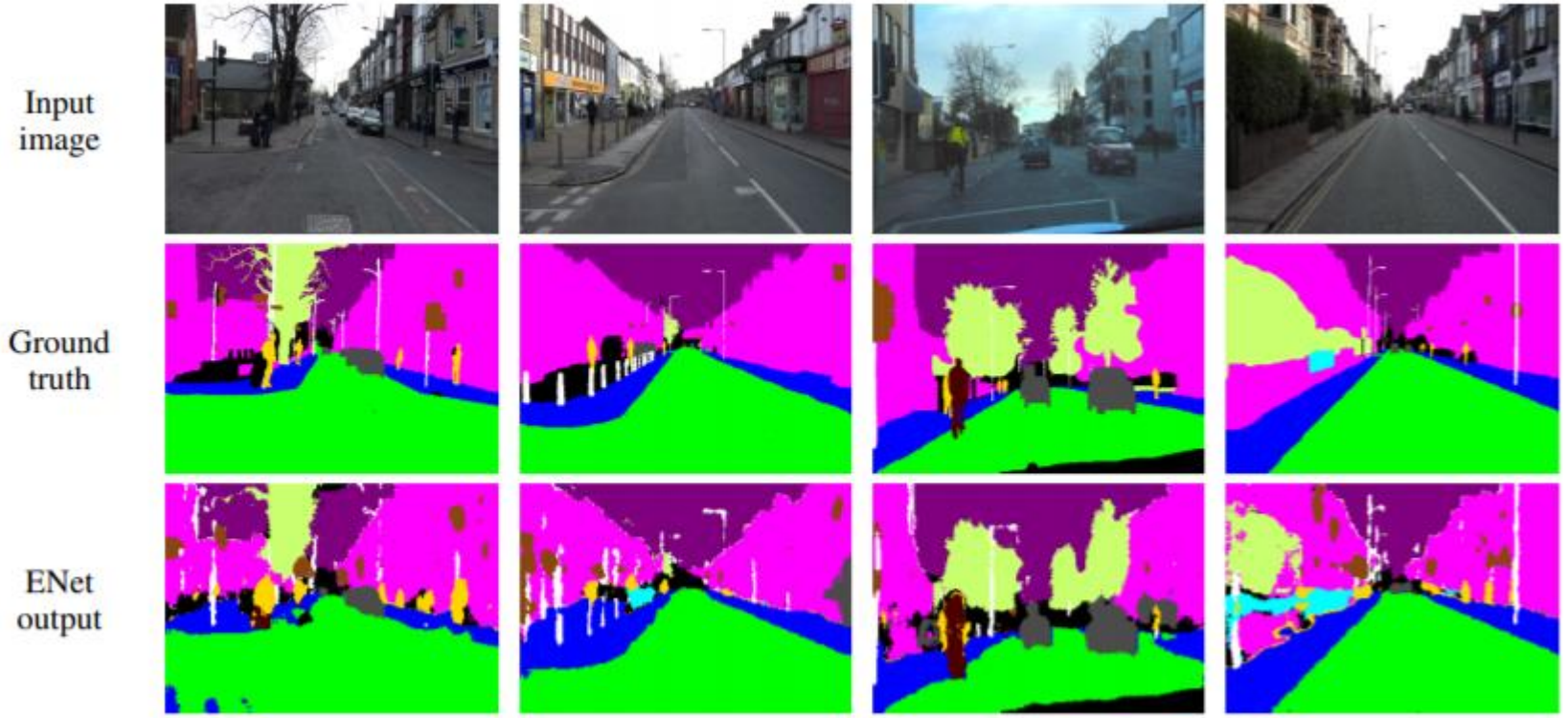


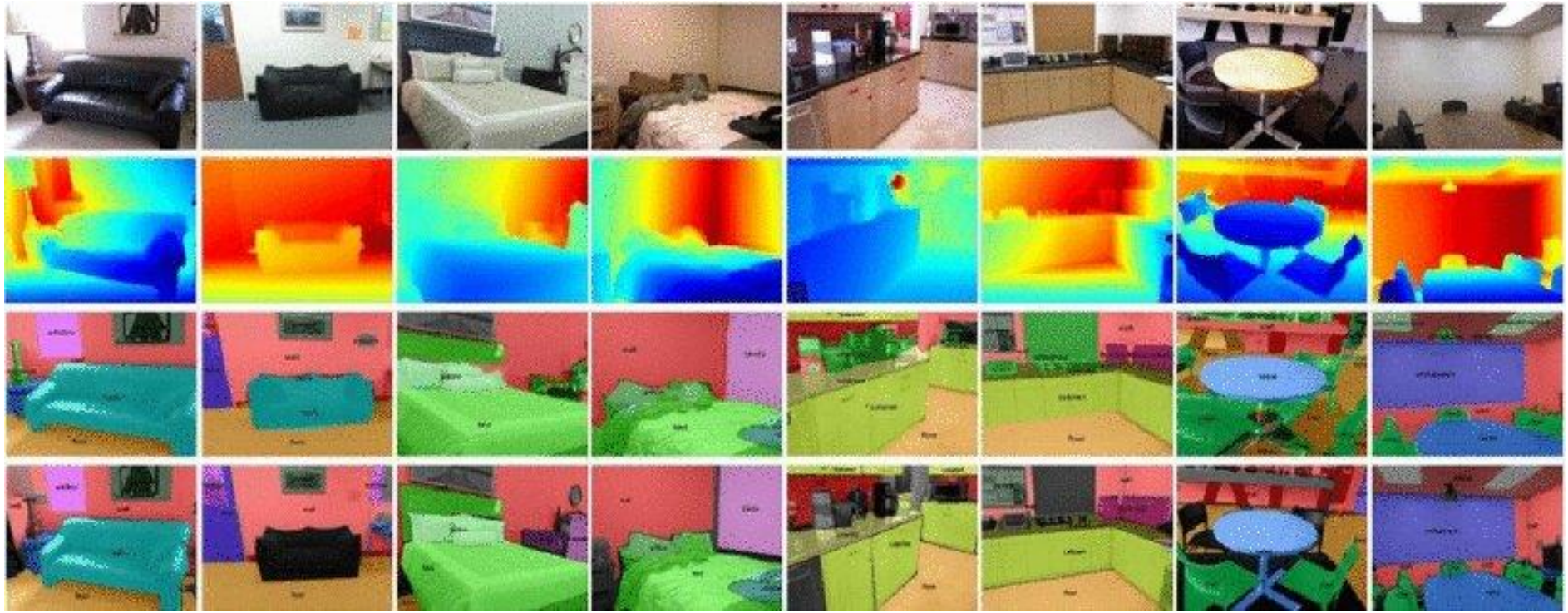
Figure 5: ENet predictions on CamVid test set [15]

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

5. Results
(2) Benchmarks

SUN RGB-D

Training : 5285 (37 classes)
Test set : 5050



출처 : Multi-class indoor semantic segmentation with deep structured model

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

5. Results
(2) Benchmarks

SUN RGB-D

Table 6: SUN RGB-D test set results

Model	Global avg.	Class avg.	Mean IoU
SegNet	**70.3**	**35.6**	**26.3**
ENet	59.5	32.6	19.7

Avg에 대한 자세한 설명이 안나와서 정확히 어떤 평가함수인지를 모르겠음.

Global avg : Pixel Accuracy
Class avg : Mean Class Pixel Accuracy
Mean IoU : MIoU

속도의 측면에서는 SegNet보다 20배 정도 빠른 속도를 보임

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

5. Results
(2) Benchmarks

SUN RGB-D

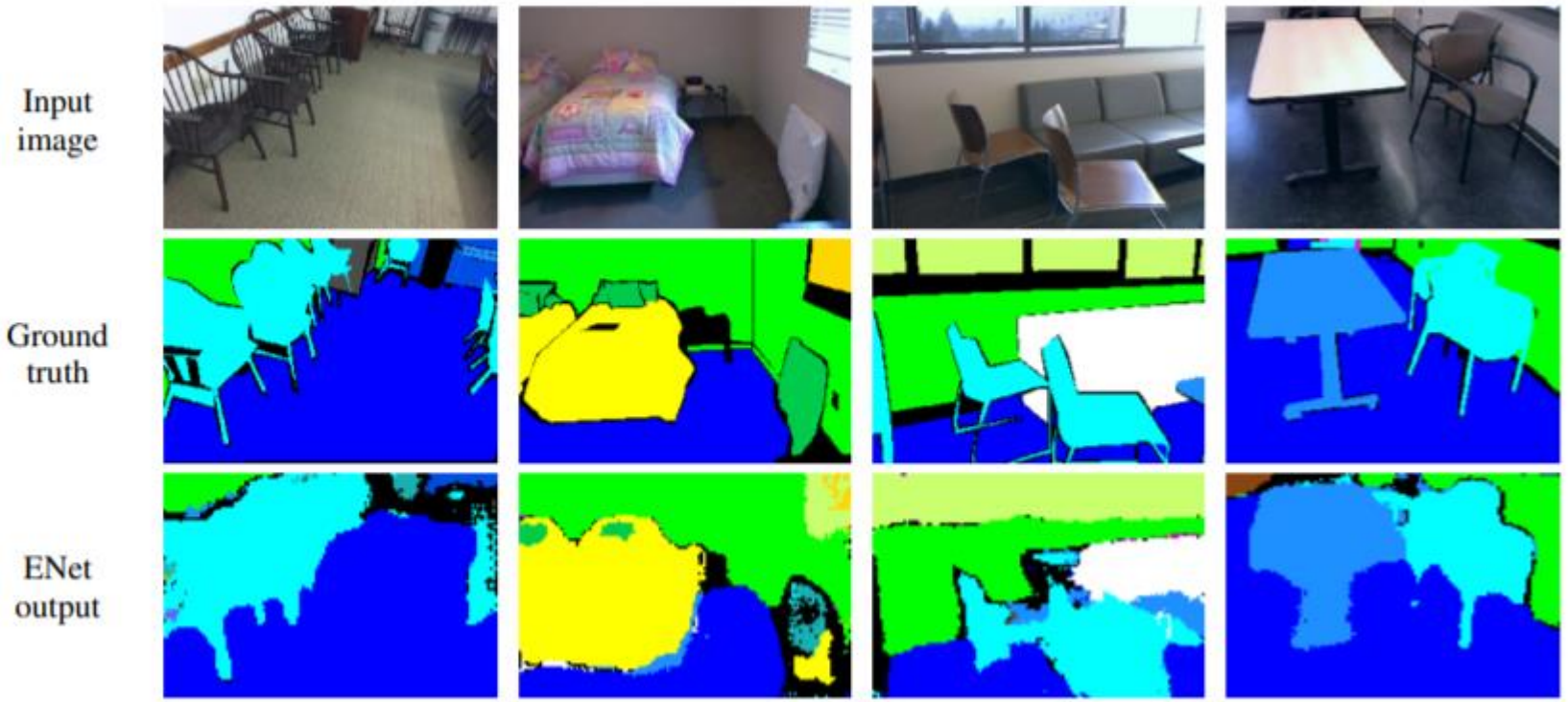


Figure 6: ENet predictions on SUN RGB-D test set [16]

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

6. Concolusion

(1) Advantages

1. 속도 측면에서 기존의 연구대비 20배의 속도 향상을 보인 논문

Table 2: Performance comparison.

Model	NVIDIA TX1						NVIDIA Titan X					
	480×320		640×360		1280×720		640×360		1280×720		1920×1080	
	ms	fps	ms	fps	ms	fps	ms	fps	ms	fps	ms	fps
SegNet	757	1.3	1251	0.8	-	-	69	14.6	289	3.5	637	1.6
ENet	47	21.1	69	14.6	262	3.8	7	135.4	21	46.8	46	21.6

2. 많은 논문들이 결합해서 좋은 성능을 보인 것 같음
   - Inception
   - ResNet
   - DilatedNet
   - PReLU
   - Spatial Dropout 등 ...

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

6. Concolusion

(2) Disadvantages

1. 속도의 향상은 있었지만 성능 측면에서는 많은 감소가 있었다는게 단점
   - 비록 SegNet 보다 Cityscape 데이터에서는 성능향상이 있었지만, CamVid와 SUNRGB-D의 데이터셋의 mIoU를 보면 segNet보다 성능이 떨어짐
   - mIoU 대비해서 Pixel ACC가 더 높은 경향을 보이는데, 이는 전체적인 형상을 잘 맞췄기 보다는 부분적인 영역으로 잘 맞춘게 아닌가 생각이 듬

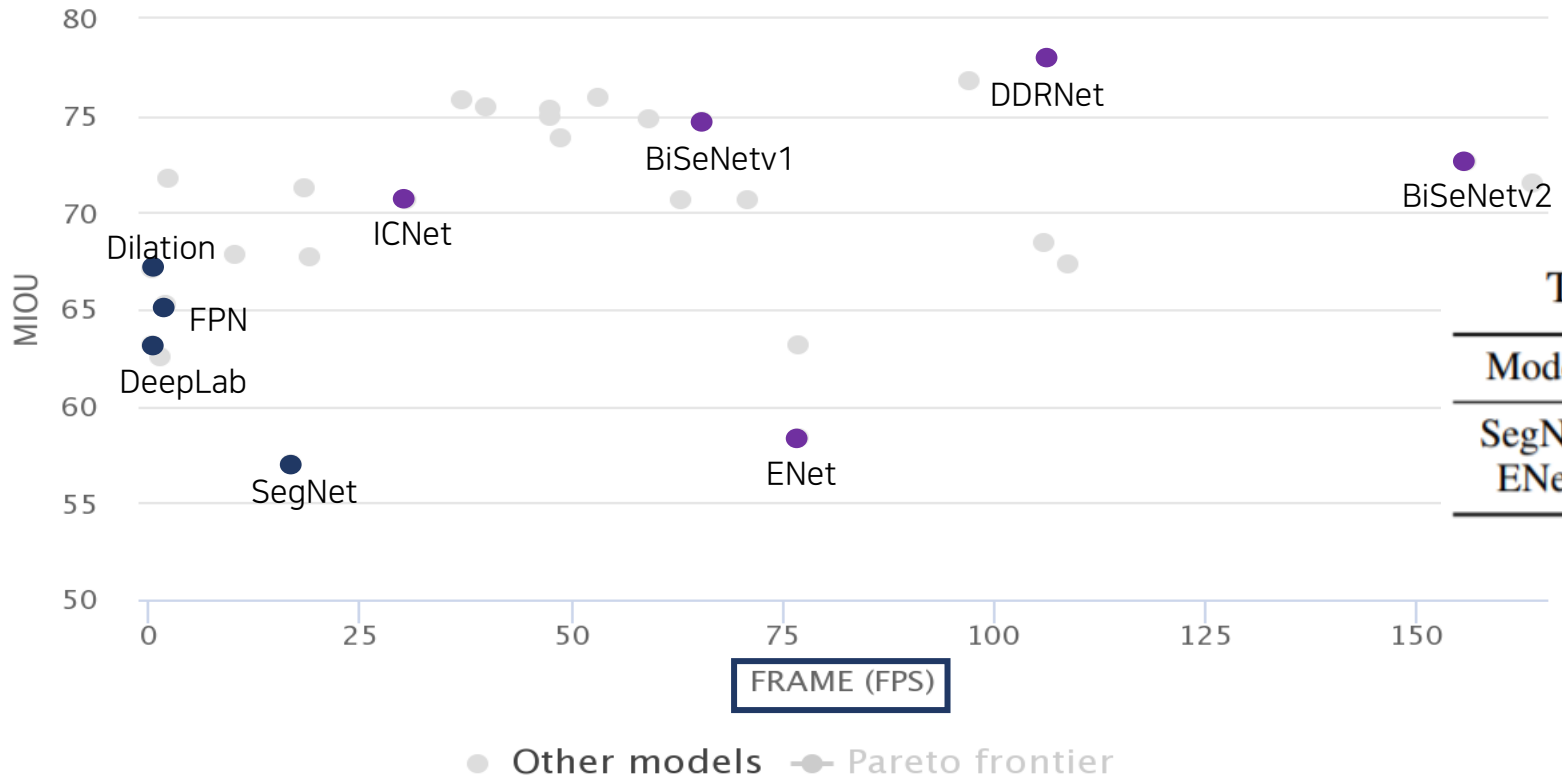


Table 6: SUN RGB-D test set results

Model	Global avg.	Class avg.	Mean IoU
SegNet	**70.3**	**35.6**	**26.3**
ENet	59.5	32.6	19.7

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

6. Concolusion

(2) Disadvantages

2. 논문 자체의 Contribution은 속도를 향상시키는 것이기는 하고 좋을 성능을 거두었지만, 자체적인 Novelty는 없던 것 같음
   - 속도 향상을 위한 시도 : 1x1 Conv, Dilated Conv, initial BottleNeck, BottleNeck <- 모두 이전에 나왔던 논문의 개념들을 한번에 접목시킨 것임

3. 적용되는 방법에 대한 설명이 충분하지 못한점이 아쉬움 (기존 논문들의 언급을 인용해서 바로 사용하거나 근거에 대한 설명이 추측으로 언급되는 부분이 아쉬움)
   - 예) PReLU가 잘 작동하는 부분에서 ResNet과 비교한 부분

> We hypothesize that identity did not work well in our architecture because of its limited depth. The reason why such lossy functions are learned might be that that the original ResNets [31] are networks that can be hundreds of layers deep, while our network uses only a couple of layers, and it needs to quickly filter out information.

   - 예) spatial Dropout, Decoder size 등에 대한 비교 (only fine-tuning 역할만을 한다는게 직관적으로 이해가 안감). 아래에 대한 추가적인 실험이 있었으면 좋았을 것 같음
     - Decoder Size가 클때랑 비교실험
     - Dropout을 했을때 왜 성능이 안좋아지는지 Feature map들의 형상을 시각적으로 비교

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

7. Questions & Answers

(1) Questions

[은경님 질문]

- ENet이 SegNet에서 경량화에 좀 더 신경을 썼다고 언급한 만큼, 다양한 시도와 실험을 한 거 같은데, PReLU를 사용한 부분에 있어서 "Figure 3"이 0에 수렴하면 ReLU가 선호되는데 대부분 0에 수렴(파란 가중치 평균선이 0 근처에서 -70대까지는 유지되니)하는 것으로 보입니다. 그렇다면 애초에 추가 hyper parameter 를 사용하는 PReLU를 사용하는 이유는 성능때문일까요? 사실 음수 가중치 학습을 위해서라면 Leaky ReLU 같은 형식도 사용해보는게 의미를 가질까요?

- Table1의 Fullconv 는 그냥 Fully connected convolution 이라 이해하면 될까요?

- 실제 적용 결과를 보면 확실히 Cityscape는 잘 된 느낌인데, Sun RGB-D는 경계선이 너무 뭉그러지는 느낌입니다. SegNet에서도 경계선에 대해 언급했고 ENet에서도 이를 위해 dilated convolutions을 썼음에도 이는 확실히 해결되지 않아보입니다. 아무래도 Sun RGB-D는 class 객체의 크기가 크기 때문일까요?

- 추가로 이제 실제 데이터로 하시는 분들께 여쭤보고 싶은 점이 지금 논문에서 benchmark 하는 데이터가 cityscape, CamVid 와 같이 도로위주의 데이터에, 아무래도 대충 형상이 일정한 class(사람, 차 등)인 경우가 많은데, 이렇게 일정한 형상이 아닌 Detail이 좀 더 요구되는 Task의 경우 대강이라도 그룹을 만들어서 유사한 형상끼리 class 를 정의해서 하시나요? 예를 들면, 도로의 금이나 땜빵 같은 경우처럼 일정한 형상이 아니라 중구난방의 모양을 갖는 경우의 연구를 어찌하고 계시는지도 궁금합니다

출처 : ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

감사합니다