Deep Learning

# Semantic Segmentation

## (의미적 분할)

강사 양석환

# Semantic Segmentation

# Semantic Segmentation 이란?

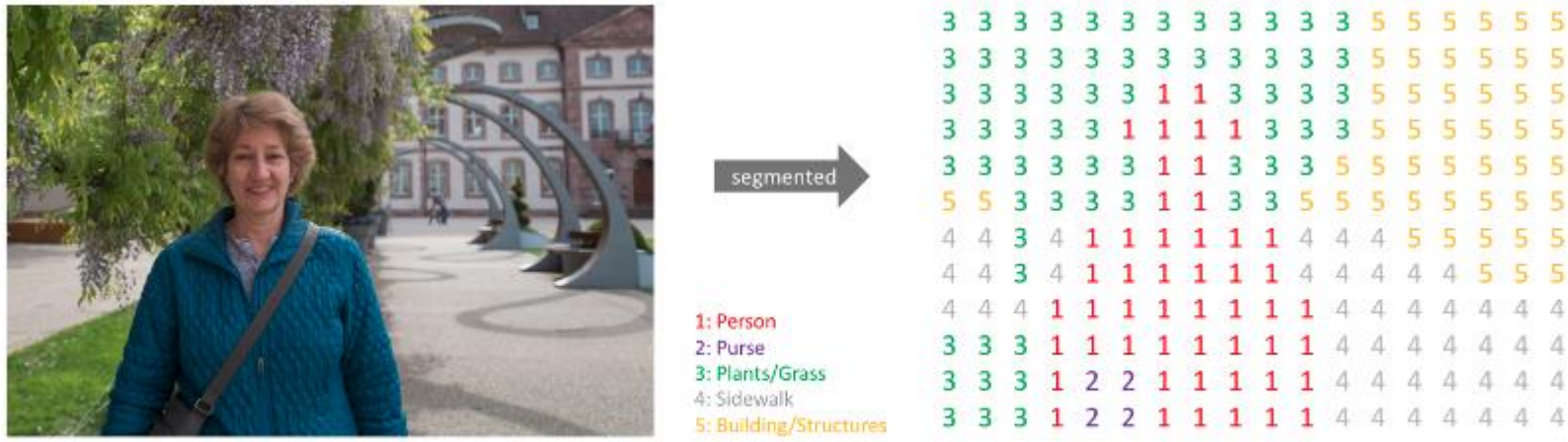
- **Semantic Segmentation(의미적 분할)**
  - 컴퓨터 비전 분야에서 가장 핵심적인 분야 중 하나
  - 디지털 이미지를 여러 개의 픽셀 집합으로 나누는 과정으로, 분할을 통해 이미지의 표현을 해석하기 쉬운 것으로 단순화하여 변환하는 것
  - 단순히 사진을 보고 분류하는 것에 그치지 않고 그 장면을 완벽하게 이해해야 하는 높은 수준의 문제라고 할 수 있음

Semantic Segmentation의 예
(출처: COCO 2020)

# Semantic Segmentation 이란?

- **Semantic Segmentation의 특징**
  - 객체 인식이 이미지 내 특정 영역에 대한 분류 결과를 보여준다면
  - Semantic Segmentation은 이미지 내 모든 픽셀에 대한 분류 결과를 보여줌

    → 이미지의 각 부분이 어떤 의미(사람, 길, 벽, 나무, 강, 하늘 등)를 가지고 있는지 구분할 수 있게 해 줌
  - Detection과 Classification을 동시에 수행할 수 있음

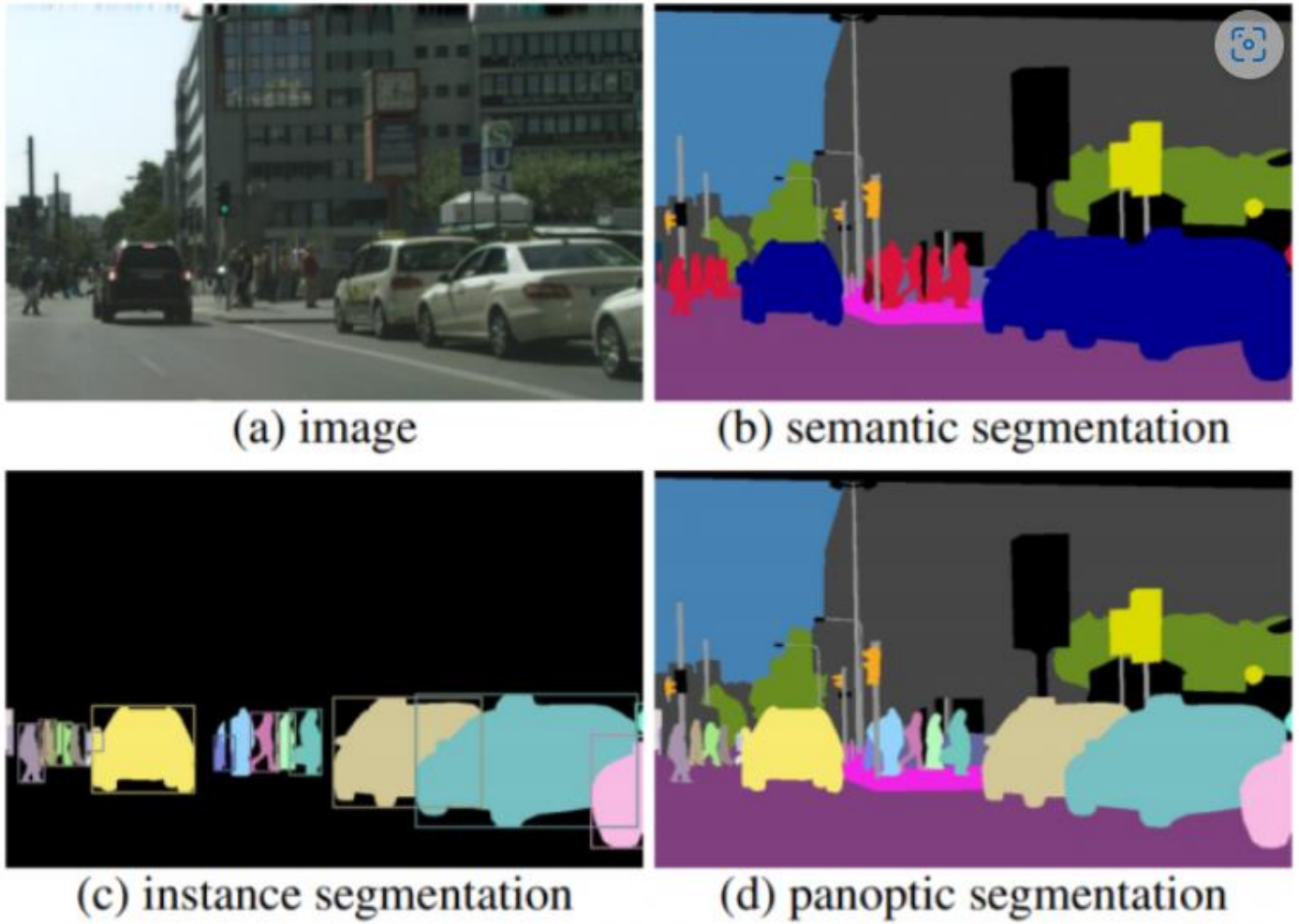


# Semantic Segmentation 이란?

- **Segmentation의 종류**
  - **Semantic Segmentation: 이미지의 모든 화소에 대해 클래스 라벨을 예측**
  - **Instance Segmentation**
    - 이미지의 모든 물체에 대해서 클래스 라벨을 예측하고 추가로 임의의 ID를 부여
    - Semantic Segmentation과의 차이
      - 겹쳐진 물체를 각각 검출함
      - 하늘이나 도로 등  정해진 형태가 없는 물체의 경우는 클래스 라벨을 부여하지 않음
      - 각 물체에 대해서 임의의 ID를 부여하기 위해 물체 별 구분을 인식함
  - **Panoptic Segmentation**
    - 두 가지의 segmentation을 결합한 형태(Semantic Segmentation + Instance Segmentation)
    - 이미지 안의 모든 화소에 대해 클래스 라벨을 예측하고 임의의 ID를 부여함

# Semantic Segmentation 이란?





(a) image (b) semantic segmentation

(c) instance segmentation (d) panoptic segmentation

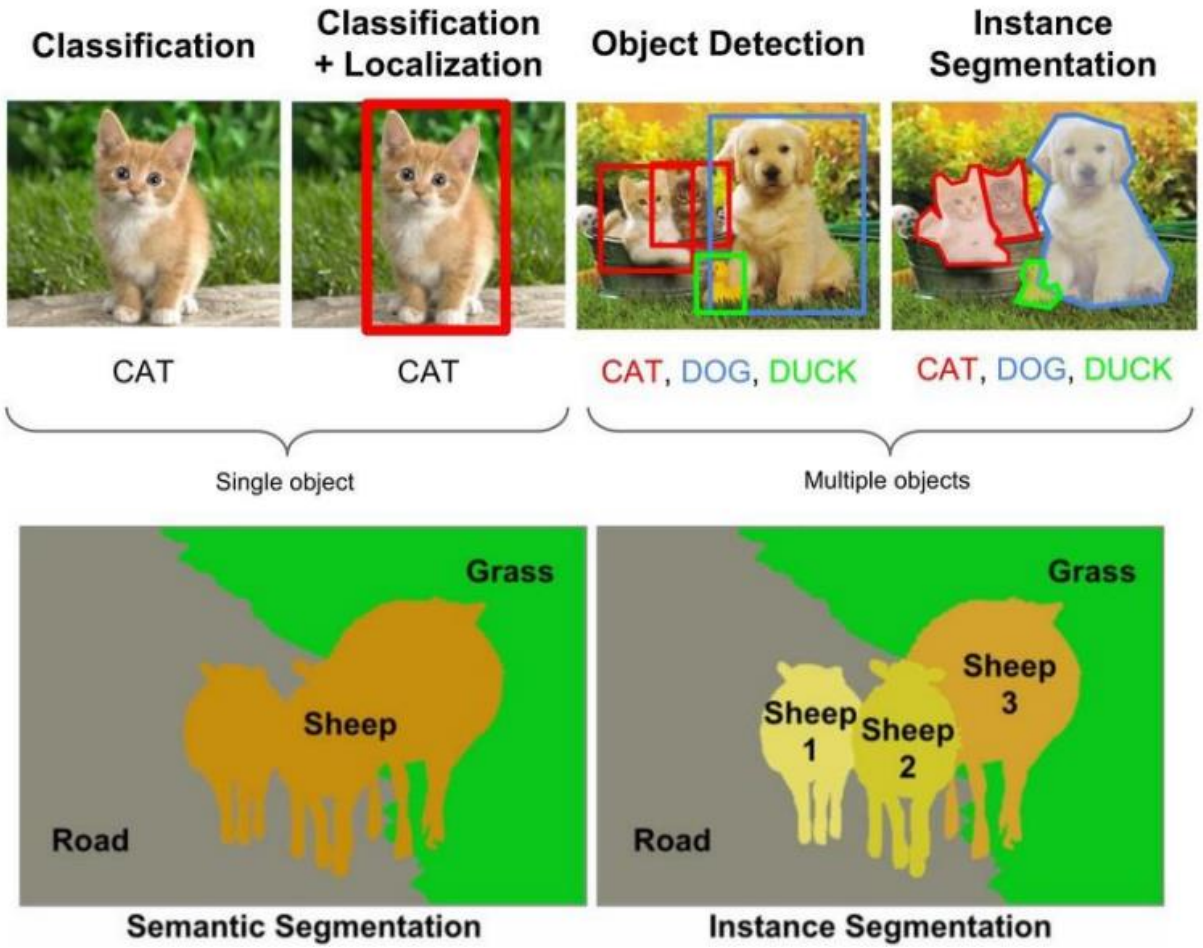
# Semantic Segmentation 이란?

- **Segmentation의 종류 별 특징**

| Semantic Segmentation | Instance Segmentation | Panoptic Segmentation |
|---|---|---|
| • 색칠 공부처럼 이미지 상의 모든 픽셀에 대해 클래스 카테고리를 부여<br>• 중첩은 분할하지 않고 인식<br>• 하늘이나 길과 같은 물체로 셀 수 없는 클래스의 영역 분할도 가능<br>• **U-NET**을 이용한 **Auto Encoder** 방식이 주된 사용 예 | • Object detection과 같은 물체의 인식을 픽셀 레벨에서 수행하는 태스크<br>• 중첩의 경우 분할하여 인식<br>• RoI 단위로 모든 것을 분할할 수는 없음<br>• **Mask R-CNN**이 주요 모델 예<br>  • Detection 기법을 사용<br>  • instance 영역을 취득 후, 각각의 영역에 대해 mask를 예측 | • Semantic Segmentation과 Instance Segmentation을 결합한 방식<br>• Thing 클래스<br>  • 셀 수 있는 클래스(자동차, 사람 등)<br>  • Instance Segmentation 수행<br>• Stuff 클래스<br>  • 셀 수 없는 클래스(하늘, 길 등)<br>  • semantic segmentation 수행 |

# Semantic Segmentation 이란?

- 컴퓨터 비전의 각 영역 비교



# CNN for Segmentation

- **CNN 모델**
  - **일반적으로 CNN은 Convolutional Layers, Non-Linear Activations, Batch Normalization, Pooling Layers 등으로 구성됨**
  - **중요한 점**
    - CNN의 비교적 앞쪽 부분의 layer에서 학습하게 되는 정보는 Low-Level Information
      - Low-Level Information : 이미지의 edges, colors 등과 같은 저 수준 정보들
    - CNN의 뒷쪽 부분의 layer에서 학습하게 되는 정보는 Higher-Level Information
      - Higher-Level Information : 객체에 관한 정보.즉 객체가 어떻게 다른지 분류할 수 있는 정보
    - CNN의 뒷쪽 부분의 layer로 갈 수록 Spatial Information(공간 정보)이 사라짐
      - 특히, Fully Connected Layers에서는 공간 정보는 거의 모두 없어짐 → by Down Sampling

# CNN for Segmentation

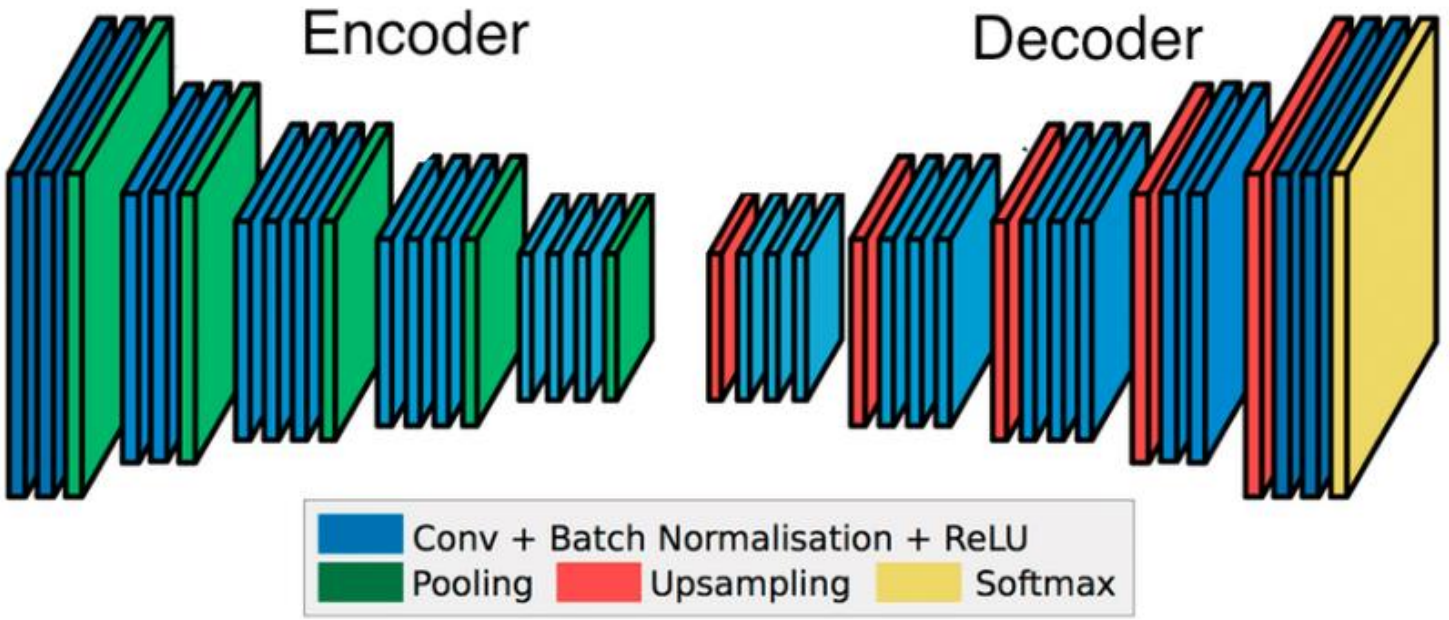
- 공간 정보가 Layer가 진행될수록 사라지는 이유
  - 입력 이미지의 W x H 는 Layer를 지나면서 Pooling Layer, Stride, Padding 등에 의해 계속해서 작아지게 됨(Down Sampling)
  - 이미지의 크기가 작아 짐 → 공간이 줄어 듦 → 공간 정보도 사라질 수 밖에 없음
  - Fully Connected Layer를 사용할 경우, 그나마 남아있는 공간정보도 모두 사라짐
    → 이런 이유로 Convolutional Layer를 1x1 필터로 적용하여
    Fully Connected Layer를 대체할 수 있음 → Fully Convolutional Network(FCN)
  - 결국 공간 정보(Spatial Information)을 어떻게 유지 할 것인가? 가 핵심 이슈가 됨
    → Encoder-Decoder 모델 적용

# Encoder-Decoder 모델

- 공간 정보 소실의 원인이 Down-Sampling이라면 Up-Sampling으로 해결 가능할까?
  - CNN 모델
    - Pooling Layer 등에 의해서 Down-Sampling 수행 → Higher Level 정보를 Feature Map 형태로 저장
      → Encoding 과정
  - Segmentation
    - W x H x 3의 이미지에서 픽셀 단위로 Class ID를 mapping → W x H의 matrix를 생성
    - Encoder를 통해 Down-Sampling를 수행하면 W x H가 줄어듦
    - 줄어든 W x H를 다시 키워 줌으로써 공간정보 회복 → Up-Sampling → Decoding 과정
    - 즉, Encoder를 통해 생성된 Feature Map을 다시 Input Size 만큼 키워서 Segmentation Map으로 재작성

# Encoder-Decoder 모델

- **Encoder-Decoder 구조**



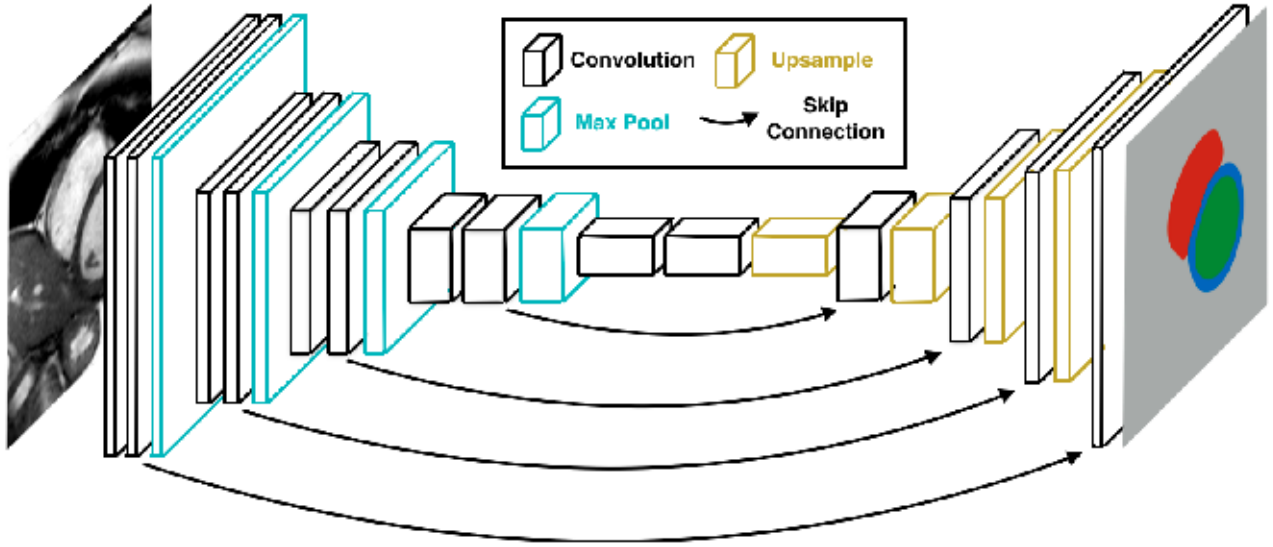

*Encoder-Decoder architecture*

- **그런데 이미 사라진 공간 정보를 다시 키운다고 제대로 복구될까? → 제대로 안됨**

  **→ Skip Connections 기술을 사용하여 극복**

# Skip Connections

- **Encoder-Decoder 구조를 사용하면**
  - **Higher level information만 가지고 Decoding 하기 때문에 Low-Level 정보가 손실됨**
    - Low-level 정보의 손실 → segmentation map을 생성했을 때, 그 경계가 모호하게 만들어짐
    - Skip Connections 기술은 Encoder의 각 Layer에서 나온 정보들을 Decoder에 적절하게 결합시켜 정확한 Boundaries를 생생하도록 도와 줌




*Encoder-Decoder with skip connections*

# 전이 학습과 미세 조정

- 전이 학습(Transfer Learning)
  - 한 분야의 문제를 해결하기 위해서 얻은 지식과 정보를 다른 문제를 푸는데 사용하는 방식
  - 다른 데이터 셋을 사용하여 이미 학습한 모델을 유사한 다른 데이터를 인식하는데 사용하는 기법
  - 새로 훈련시킬 데이터가 충분히 확보되지 못한 경우에 학습 효율을 높여 줌

# 전이 학습과 미세 조정

- **전이 학습의 방법**
  - **Fixed Feature Extractor 방식**
    - Fully Connected Layer 앞 단의 층들의 가중치를 모두 고정시키고 뒷 단의 Classifier만 다시 학습하는 방법
    - 보통은 Linear SVM 또는 Softmax Classifier를 사용하여 새로운 Dataset에 대하여 Classifier만 학습함
  - **Fine-tuning 방식**
    - 사전 학습된 모델(Pretrained Model)의 모든 Layers들을 다시 학습하는 방식
    - 이 때, 모든 층들의 가중치에 많은 변화를 주지 않고 미세하게 Tuning만 하는 방식으로 학습하는 방법

# 전이 학습과 미세 조정

- **Fixed Feature Extractor 방식과 Fine-Tuning 방식의 선택 기준**
  - **새로운 Dataset의 양이 적고 사전 학습 모델의 Original Dataset과 유사한 경우**
    - Fine-Tuning을 하게 되면 Dataset의 양이 작기 때문에 Overfitting이 발생
    - 새로운 Dataset과 Original Dataset이 유사하기 때문에 굳이 모든 층의 가중치를 다시 조정할 필요가 없음
    - 따라서 Fixed Feature Extractor 방식으로 앞 단 Feature Extractor의 가중치를 고정하고 뒤에 classifier 만 다시 학습하는 방식을 추천
  - **새로운 Dataset의 양이 많고 사전 학습 모델의 Original Dataset과 유사한 경우**
    - Dataset의 양이 많기 때문에 Overfitting의 우려가 적음
    - 따라서 Fine-Tuning 방식을 추천

# 전이 학습과 미세 조정

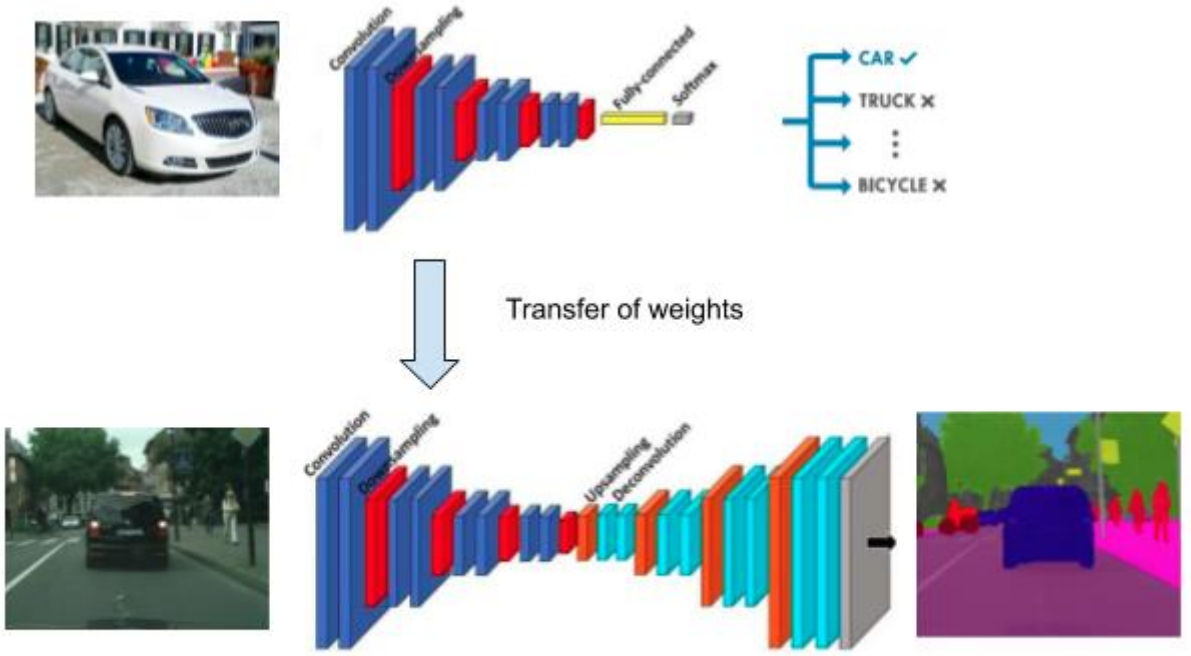
- **새로운 dataset의 양이 적고 사전 학습 모델의 Original Dataset과 전혀 다른 경우**
  - Fine-Tuning을 하게 되면 Dataset의 양이 적기 때문에 Overfitting이 발생
  - Fixed feature extractor 방식으로 하면 Dataset의 종류가 너무 다르기 때문에 성능을 보장할 수 없음
  - 따라서 Fixed Feature Extractor 방식을 사용하되, 뒷 단의 Higher Level 정보를 담고 있는 Features를 사용하지 않고 앞 단의 Low-Level 정보를 가진 Feature를 사용하여 학습을 진행하는 것을 추천 을 추천
  - Higher Level 정보는 Object를 구분하는 정보 → Dataset이 전혀 다른 경우, 성능에 악영향을 미칠 수 있음
  - 또한 Extractor의 뒷 단의 Higher Level 정보가 학습된 Layer부터 다시 Fine-Tuning 하기에는 Dataset가 매우 부족하기 때문에 Fine-Tuning은 비추천

# 전이 학습과 미세 조정

- **새로운 Dataset의 양이 많고 사전 학습 모델의 Original Dataset과 전혀 다른 경우**
  - 단순히 Scratch Training(새롭게 모든 layer를 다 학습함)을 하면 되지 굳이 Transfer Learning을 해야 할까?
  - 그러나 사전 학습 모델의 가중치로 초기화하여 학습하는 것이 Scratch Training 보다 유리하게 시작할 수 있음
  - 따라서 Fine-Tuning 방식을 적용하는 것을 추천

# 전이 학습과 미세 조정

- 왜 Segmentation에서 전이학습을 적용하는가?
  - 기존에 잘 학습된 모델을 Transfer Learning 기법을 사용하여 Encoder로 활용하게 되면 학습하고자 하는 Dataset을 보다 쉽게 학습할 수 있음




*Transfer learning for Segmentation*

# Segmentation에 사용할 모델의 선정

- **Base Model(=Backbone Network) 선택하기**
  - **Encoding 과정에서 쓰이는 모델을 Base Model로 보면 됨**
  - **Base Model은 ImageNet으로 사전 학습된 Model을 사용**
    - 대표적으로 ResNet, VGG-16, Mobilenet 등이 있음

직접 만들어도 되지만(Custom Network)
만들기도 어렵고 성능을 보장하기는 더 어려움
그냥 속 편하게 끌어와서 쓰는 것을 추천함..

# Segmentation에 사용할 모델의 선정

- **Base Model 후보들의 장단점 비교**
  - **ResNet**
    - 주로 50, 101 단의 Layer로 구성된 Network를 많이 사용
    - VGG-16이나 Mobilenet에 비해서 정확도가 높음
    - 속도나 GPU 메모리 측면에서 더 느리고 메모리를 많이 차지함

      → 여러 개의 모델을 사용해야 하는 Application이나 임베디드 환경 등에서는 사용하기 어려움
    - 단점의 극복을 위하여 ResNet을 10, 21 등 짧은 Layer까지만 사용하고 전이 학습을 적용하는 방법이 있음

# Segmentation에 사용할 모델의 선정



- **VGG-16**
  - ResNet 보다 속도는 약간 빠르고 정확도는 약간 떨어짐
  - 많은 Classifier 성능 비교 논문들을 보면 ResNet이 빠르다고 나와 있는데 Fully-Connected Layer를 사용하지 않으면 VGG가 더 빠르다고 함
  - 그렇다고 속도가 아주 빠른 것은 아니기 때문에 사용하기 애매함 → 따라서 비추천
    - ResNet이 나오기 전까지는 많이 사용됨

- **Mobilenet**
  - 속도은 앞의 2개의 모델보다 훨씬 빠름 → Mobile 환경의 CPU로 동작할 수 있게끔 설계되었기 때문
  - 정확도는 다소 떨어지지만 정확도로 인한 리스크가 크지 않은 Application이라면 추천

# Segmentation에 사용할 모델의 선정

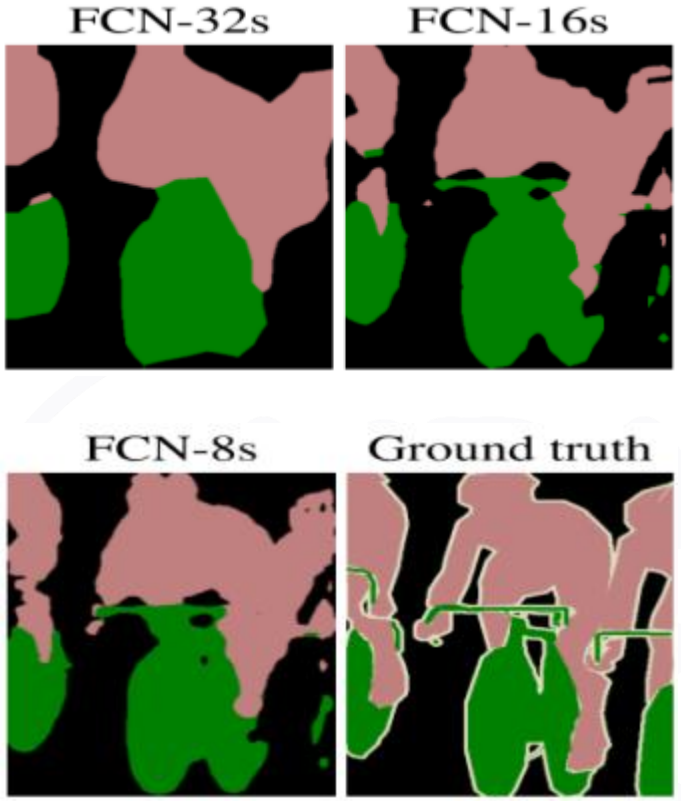
- **Segmentation Model 선택하기**
  - Base 모델과 Decoding 과정에서 쓰이는 모델을 포함한 Segmentation 전체 구조를 위한 모델
  - 대표적으로 FCN(Fully Convolutional Network), U-Net, SegNet, PSPNet 등이 있음

# Segmentation에 사용할 모델의 선정

- **Segmentation Model 후보들의 특징, 장단점 비교**
  - **FCN (Fully Convolutional Network): CVPR 2015**
    - 가장 처음 제안된 방법
    - End-to-End Semantic Segmentation 방법
    - Encoder로 VGG, AlexNet과 같은 Network를 사용
    - Decoding을 위해서 Fully Connected Layer를 삭제하고 1 x 1 Convolutional Layer를 사용함
    - Skip Connections을 위해 사용된 Features Map
      - FCN8, FCN16, FCN32

# Segmentation에 사용할 모델의 선정

- **Skip Connections을 위해 사용된 Features Map**
  - **FCN8**
    - 가장 뛰어난 성능을 보임
    - 마지막 feature map을 4배 upsampling
    - pool4의 output feature map을 2배 upsampling
    - pool3 output feature map과 concatenated
  - **FCN16**
    - 마지막 feature map을 upsampling
    - pool4의 output feature map과 concatenated
  - **FCN32**
    - 공간 정보가 많이 손상되기 때문에 FCN8과 같이 앞쪽의 Layer에서 얻은 Features Map을 사용하는 것이 좋음
    - 마지막 feature map만을 사용 (원본 대비 1/32까지 줄어든 feature map이라 32)



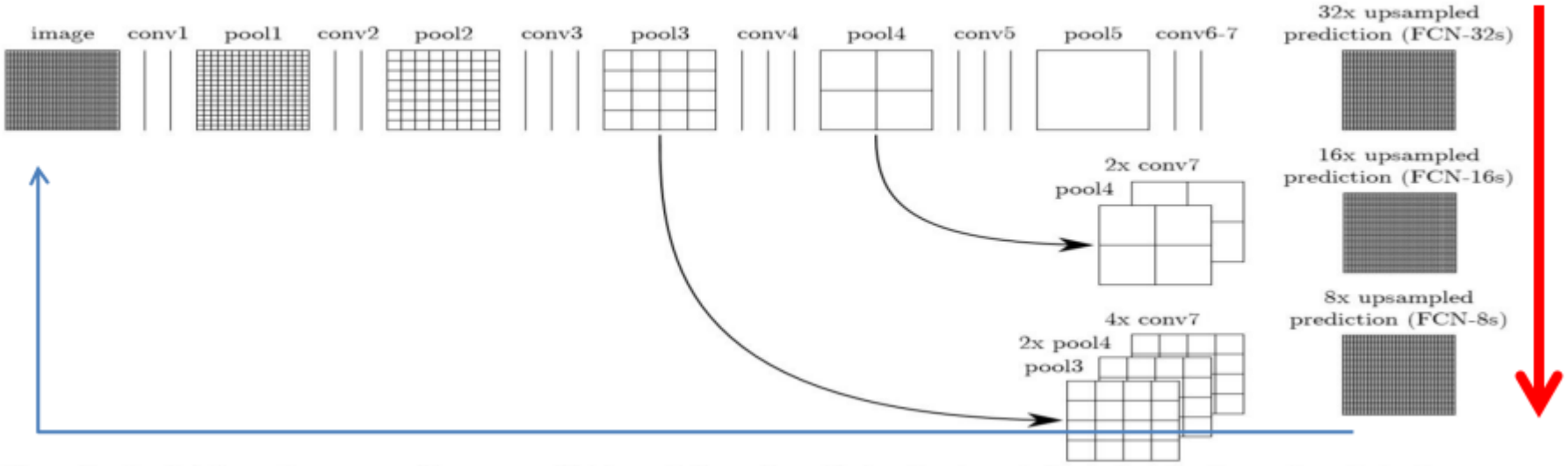
# Segmentation에 사용할 모델의 선정

- **FCN에서 Up-Sampling 방법으로 사용한 기술**
  - **Transposed Convolution**
    - 아래 그림에서 빨간 화살표 옆에 3개의 영상이 tranposed convolution에 의해 upsampling 된 이미지
    - 위에서부터 FCN 32, 16, 8
    - 마지막에 Input size에 맞게 변환하기 위해서 bilinear interpolation을 사용

Transposed convolution은 논문들에서 이름이 굉장히 다양하게 거론됨

- Strided Fractionally Convolution
- Deconvolution
- Transposed Convolution
- Backwards Strided Convolution
- Upconvolution

최근에는 거의 transposed convolution으로 수렴



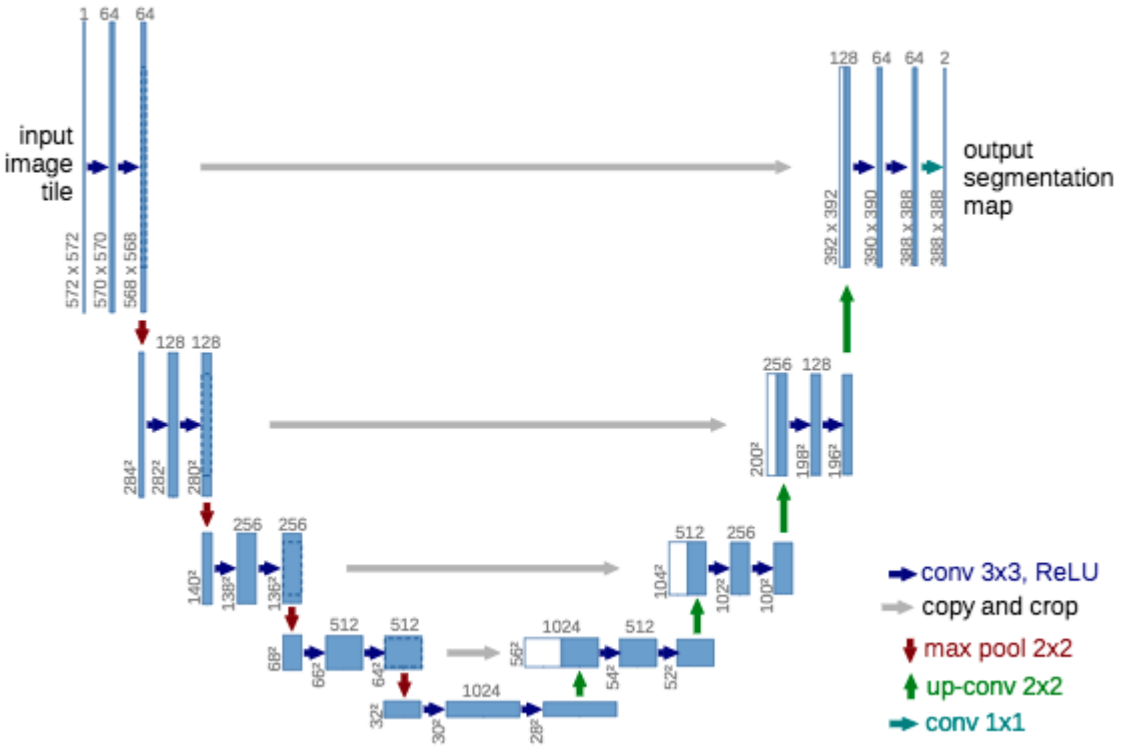
# Segmentation에 사용할 모델의 선정

- FCN 요약
  - Semantic segmentation을 위해 fully connected layer를 없애고 1 x 1 fully convolutional layer로 대체
  - 손실된 spatial information을 위해 앞단의 feature map을 사용 (FCN 8)
  - Learnable upsample convolution을 제안 (Transposed convolution)

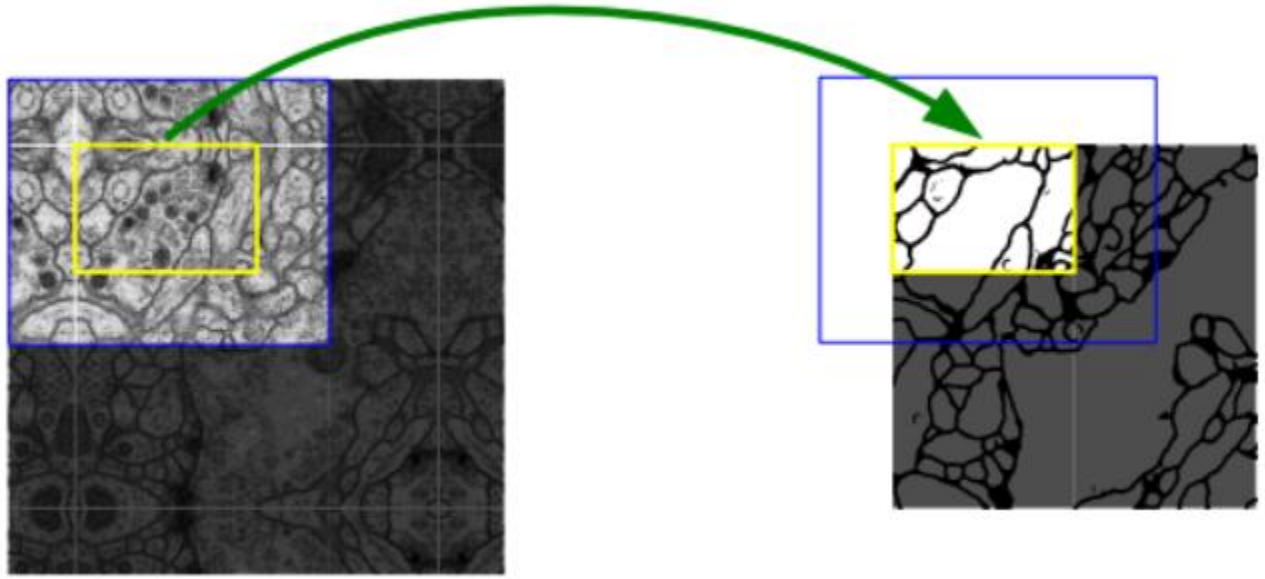
- FCN은 다른 segmentation model들에 비해서 비교적 가벼운 모델
  → dataset이 어렵지 않고 쉬운 케이스인 경우 많이 사용
- 다만, 오래된 모델이기 때문에 성능은 그다지 좋지 않을 수 있음

# Segmentation에 사용할 모델의 선정

- **U-Net: MICCAI 2015**
  - FCN이 Transposed Convolution을 통해 Decoding 하는 과정을 Backward Learning 으로 진행했다면
  - U-Net은 Encoder-Decoder Structure를 적용하고 Skip Connection을 추가로 사용함
  - Encoder와 Decoder의 Layer가 대칭을 이루며 그 모습이 U자와 같아서 U-Net이라고 부름
  - Skip Connection으로 원본 영상을 최대한 살리는 방향으로 과정이 진행됨
  - 의료 영상처리에서 주로 사용되던 Network
    - 의료 영상에서 부족한 데이터를 생성하는 용도로 많이 사용됨



# Segmentation에 사용할 모델의 선정



- 우측의 output 영상의 노란 부분을 출력하려면 파란 박스만큼의 영역이 필요한데 비어 있기 때문에 원본 input 영상에서 그대로 가져오는 방법(Mirroring)으로 진행함

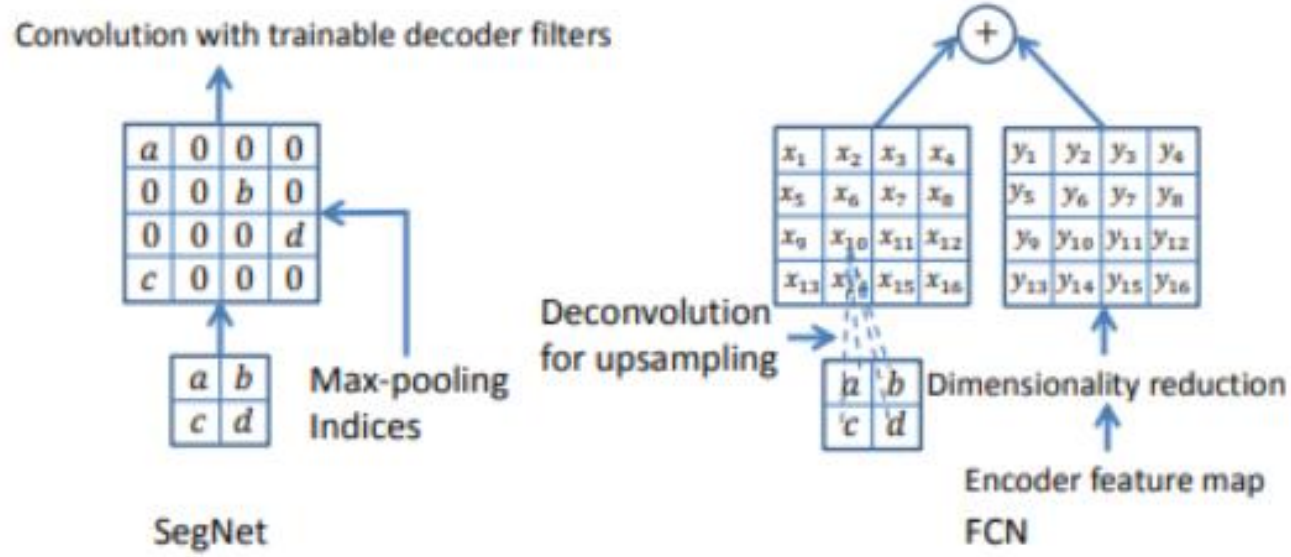
- U-Net은 skip connections을 통해서 영상의 디테일한 부분까지 살리는 모델이며
- 의료 영상을 대상으로 만들어졌기 때문에 굉장히 섬세한 부분을 살려낼 수 있음

# Segmentation에 사용할 모델의 선정

- **SegNet: TPAMI 2017**
  - Encoder-Decoder 형태의 구조를 Segmentation에 적용한 대표적인 Network
  - Skip Connections를 사용하지 않음
  - FCN과 달리 Up-Sampling 할 때, Down-Sampling 과정에서 저장해둔 Feature Map 전체를 사용하지 않음
  - Up-Sampling 진행 방법
    - Max Pooling 으로 Down-Sampling 할 때 Index 기억 → Index를 이용하여 Up-Sampling 진행
    - FCN은 Max Pooling 된 Feature Map을 전부 메모리에 저장해 두고 Up-Sampling 할 때 결합하는 방식인데
      그에 비해 SegNet은 인덱스만 기억해두고 해당 인덱스로 값을 배치하여 Up-Sampling 하기 때문에
      메모리를 효율적으로 사용할 수 있다는 장점을 가짐
    - SegNet은 Decoder에서 사용하는 필터를 학습함. FCN의 경우 따로 필터를 학습하지 않기 때문에 이 점에서 차이 발생

# Segmentation에 사용할 모델의 선정

- 필터 학습 시의 이점
  - 구체적으로 filter를 학습할 때 어떤 이점이 있는지는 명확하게 알려지지 않음
  - 실험 결과를 봤을 때, SegNet이 정확도가 더 좋고 메모리 효율에 대해서도 이점이 있음
- 정확도도 매우 좋은 것이 아니고 미세하게 좋기 때문에 SegNet이 더 좋은 모델인지 아닌지는 불명확함
- 메모리를 절약하고자 할 경우에는 SegNet이 좋다고 평가됨

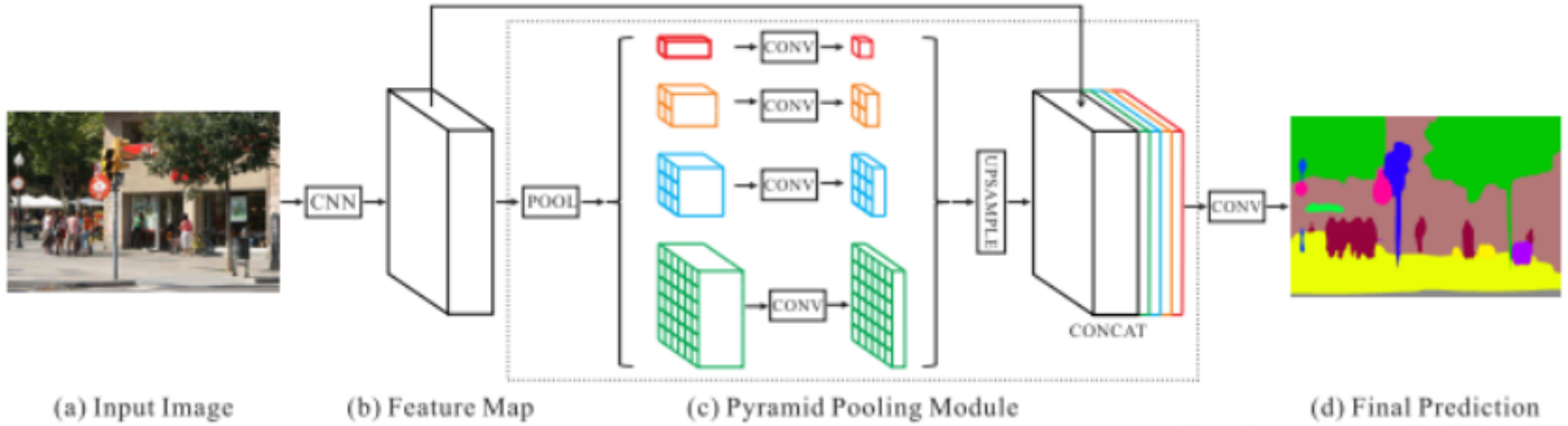


# Segmentation에 사용할 모델의 선정

- **PSPNet (Pyramid Scene Parsing Network): CVPR 2017**
  - Pyramid 구조로 되어 있음
  - 여러 Scale로 나누어 학습하기 때문에 Pyramid 방식이고 이를 다시 하나로 결합(Concat)하는 작업을 진행함 (Pyramid Pooling Module)
  - PSPNet은 Global Context를 더 잘 나타내기 위해 설계됨
    - 앞 단에 Encoder를 통해 만들어진 Feature Map을 여러 Scale로 Pooling(논문에서는 4개의 scale로 나누어 pooling)
    - 다음으로 해당 Feature Map을 1x1 Convolution Layer를 통해 채널 수를 조절
    - Up-Sampling 시 기존의 Feature Map 사이즈에 맞게 4개의 Feature Map을 Bilinear Interpolation으로 Resize 하여 Up-Sampling
    - 다음으로 원래의 Feature Map 뒤에 4개의 Feature Map을 결합(Concat)하여 사용함

# Segmentation에 사용할 모델의 선정

- **Pyramid Pooling Module이 Global Context를 더 잘 표현 할 수 있는 이유**




  - **Pyramid pooling module의 빨간색 scale에서**
    - scale이 작아 짐으로써, 작은 object는 사라지고 큰 object들이 나타남

      → scale이 작을 때는 큰 object를 중점적으로 나타내는 features가 됨
  - **Pyramid pooling module의 녹색 scale에서**
    - 작은 object를 잘 표현하기 위해서는 scale을 키워서 봐야 features를 잘 찾을 수 있음

      → 녹색 부분처럼 큰 scale에서 features를 추출

segmentation 하고자 하는 물체의 input size 대비 size을 추려서

적당한 scale로 feature map을 만들 수 있도록 pyramid를 설계하고

이를 바탕으로 큰 object, 작은 object 모두 잘 찾아 낼 수 있도록 하는 network

# Segmentation에 사용할 모델의 선정

- **PSPNet의 단점**
  - **input size가 최소 500 x 500은 되어야 그 효과가 나타남**
    - 너무 작은 size의 input을 사용하면 pyramid에서 scale이 축소되었을 때 문제가 될 수 있기 때문
  - **PSPNet은 U-Net과 다소 비슷함**
    - U-Net: 의료 영상처리를 위해 만들어 졌기 때문에 굉장히 작고 detail한 부분까지 찾아내는 데 초점이 맞춰 짐
    - PSPNet: 작은 object들까지 global하게 찾아내는데 초점이 맞춰 짐

    → 찾고자 하는 dataset들의 크기가 크고, 찾고자 하는 객체들의 수가 얼마 안되는 경우에는 U-Net이나 PSPNet은 너무 과도하게 찾는 network가 될 수 있음

    → 찾고자 하는 dataset들이 크기가 다양하고 대다수 작은 경우가 많으며, 그 수가 많은 경우는 PSPNet이나 U-Net을 사용하기에 적합함

# Segmentation에 사용할 모델의 선정

- Input Size(입력 데이터의 크기) 선정
  - Segmentation을 하기 위해서 모델 선택 이상으로 중요한 것이 Input Size의 선택
    - 일반적으로 input size는 200 x 200 ~ 600 x 600을 사용함
  - Input Size가 커지면 학습 할 때나 추론(Inference) 할 때 모두 GPU Memory가 많이 필요하게 됨
  - 내가 찾는 물체가 작은 사이즈 인 경우
    - 600 x 600 사용
      - 정확도는 높겠지만 그 상황에서 적은 리소스와 빠른 inference 속도를 보장할 수 있는 network가 과연 있는가...
    - 400 x 400 사용
      - 나름 괜찮은 정확도와 속도를 나타낼 수 있음
      - 그러나Segmentation 이외의 다른 Model들을 같이 사용해야 하는 경우라면 Momory의 제약으로 인해 힘든 경우가 발생할 수 있음