Day 13 Presentation

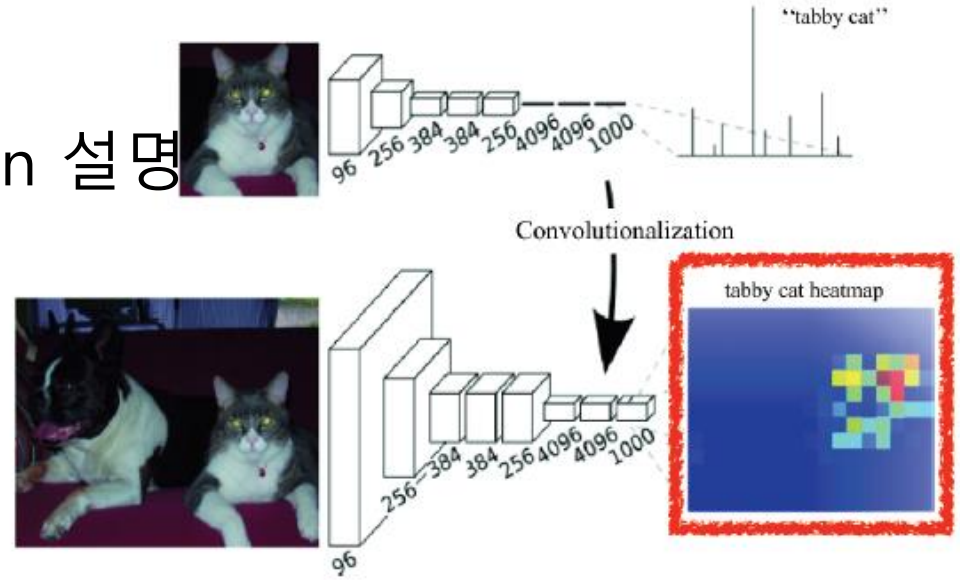
heatmap & sematic segmentation

그 연결고리를 찾아서... (아득)

강의에서 이해가 안됐던 부분

- 강의의 흐름:

1. semantic segmentation 설명
2. Fully Convolutional Network (FCN) 설명
3. FCN 만드는데 사용된 Convolutionalization 설명
4. Convolutionalization의 의의 설명
5. 그래서 히트 맵과 무슨 상관이..???
6. 안 그래도 누가 질문올림



위의 pdf 자료에서 저는 기존 CNN 구조를 이용하여 얻은 output은 사진이 속해있는 라벨의 확률이라고 생각했고
이 기존 CNN구조를 Fully Convolutional Network로 바꾸었을때 heatmap을 얻을 수 있다고 하는데
이 heatmap이 의미하는것과 왜 Fully Convolutional Network로 바꾸었을때 heatmap을 얻을수있는지 궁금합니다

연관 강의 [[DAY 13] Convolutional Neural Networks] | [DLBasic] Computer Vision Applications

heatmap과 semantic segmentation과의 연결고리는?

- **Fully Convolutional Networks for Semantic Segmentation (2015)**
- 이번 강의의 내용은 거의 ~~(전부)~~ 이 논문에서 나옴
- 포인트는 CNN과 FCN의 차이점으로 거슬러 올라감
- **왜 FCN을 쓰는가?**

앞으로 논문에서 인용한 부분은 " " 처리함

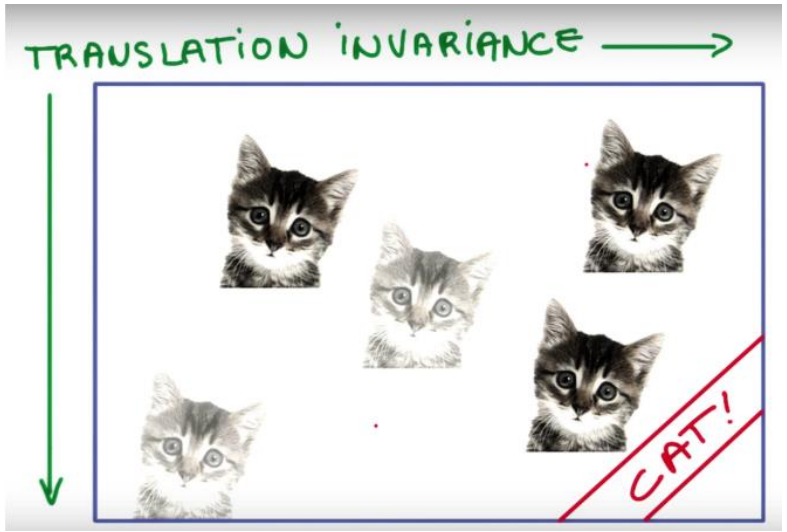
FCN을 써야하는 이유

- 데이터가 **이미지** 이기 때문에
- 이미지는 위치정보도 중요함. (Width & Height & color)
- 하지만 모든 CNN은 위치정보를 다 죽임 (fully-connected layer)
- **"Convnets are built on translation invariance"**

$$y_{ij} = f_{ks}\left(\{x_{si+\delta i, sj+\delta j}\}_{0 \le \delta i, \delta j \le k}\right)$$

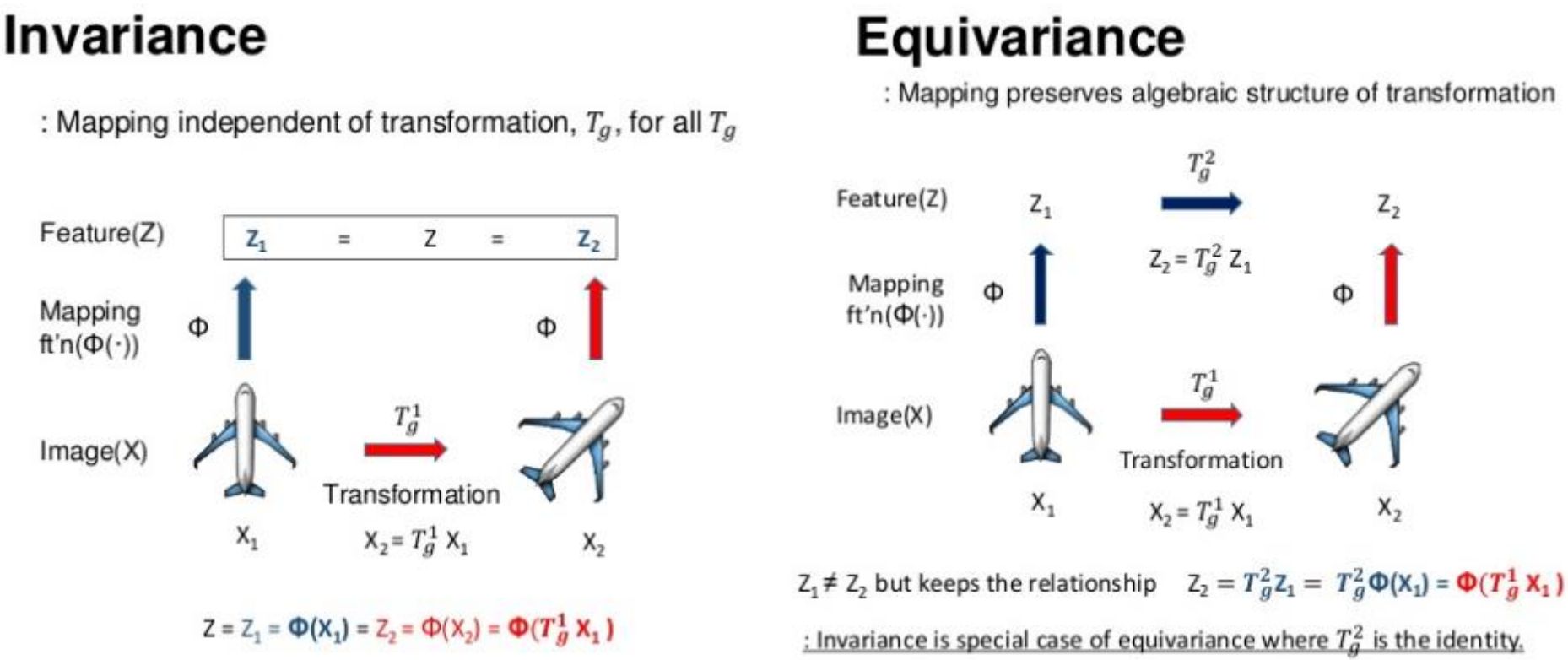
$$\ell(\mathbf{x}; \theta) = \sum_{ij} \ell'(\mathbf{x}_{ij}; \theta).$$

f_ks = nonlinear function with k,s
k = kernel size, s = stride
l = loss function



Invariance vs Equivariance (부가설명)

- input의 값이 달라졌을 때 output의 값이 다른지 vs 같은지



https://www.slideshare.net/ssuser06e0c5/brief-intro-invariance-and-equivariance

convolution layer는 위치 값을 갖는다!

- “However, these **fully connected layers can also be viewed as convolutions with kernels that cover their entire input regions.** Doing so casts them into fully convolutional networks that take input of any size and output classification maps.”
- 모든 input 에 대해 1*1*d convolution!
- 따라서 어떤 size의 이미지도 분류 문제에 적용할 수 있음.

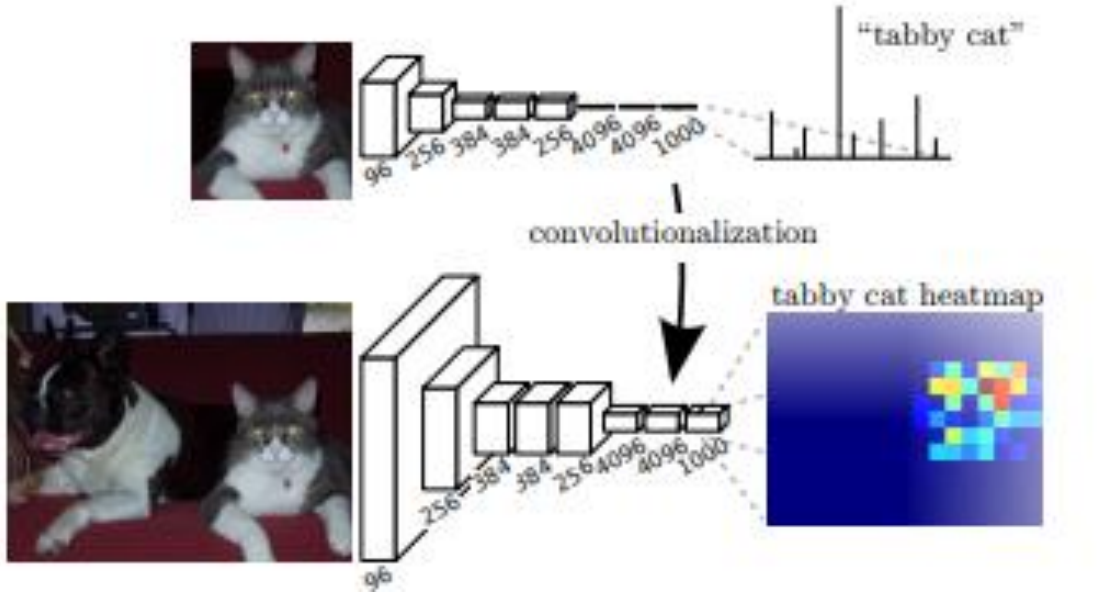
올해는 다르다!

nonlinear function -> nonlinear filter

$$f_{ks} \circ g_{k's'} = (f \circ g)_{k'+(k-1)s',ss'}.$$

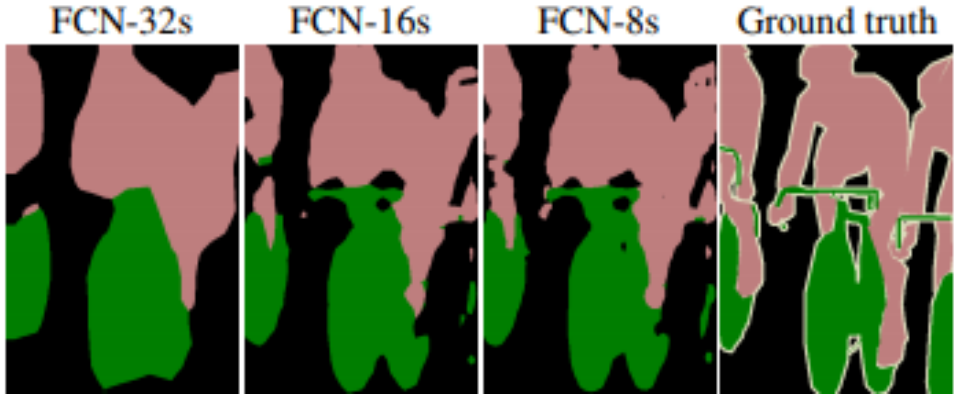
그래서 결국... heatmap의 의미는

- 각 1x1*d convolution은 위치 정보를 가짐
- heatmap은 이러한 convolution 들의 분류를 이용한 score map
- 각 class마다 heatmap이 다르게 나타남.



고양이 분류를 했으니 당연히 고양이 위치 위주로 class score가 쏠리는 모습

heatmap 만으로 충분?



- 당연히 아님
- 1x1xd 로 줄이는 과정에서 subscaling, 즉 픽셀이 줄어드는 현상이 발생했으므로 upscaling 과정을 통해서 픽셀 개수를 맞춰줌
- 그 과정이 강의에서 다룬 **deconvolution**
- “Note that the deconvolution filter in such a layer need not be fixed (e.g., to **bilinear upsampling**), but can be learned. **A stack of deconvolution layers and activation functions** can even learn a nonlinear upsampling”
- bilinear interpolation(initialization) & backwards strided convolution(= deconvolution = Transposed convolution)이 사용됨

참조한 블로그

- https://kuklife.tistory.com/11
- https://medium.com/@msmapark2/fcn-%EB%85%BC%EB%AC%B8-%EB%A6%AC%EB%B7%B0-fully-convolutional-networks-for-semantic-segmentation-81f016d76204

Pre-trained model

PyTorch에서 지원하는 model

TORCHVISION.MODELS

Classification

The models subpackage conta

- AlexNet
- VGG
- ResNet
- SqueezeNet
- DenseNet
- Inception v3
- GoogLeNet
- ShuffleNet v2
- MobileNet v2
- ResNeXt
- Wide ResNet
- MNASNet

Semantic Segmentation

The models subpackage contains definitions

- FCN ResNet50, ResNet101
- DeepLabV3 ResNet50, ResNet101

– Video classification

ResNet 3D

ResNet Mixed Convolution

ResNet (2+1)D

Object Detection, Instance Segmentation and Person Keypoint Detection

The models subpackage contains definitions for the following model architectures for detection:

- Faster R-CNN ResNet-50 FPN
- Mask R-CNN ResNet-50 FPN

PyTorch에서 지원하는 model

Network	Top-1 error	Top-5 error
AlexNet	43.45	20.91
VGG-11	30.98	11.37
VGG-13	30.07	10.75
VGG-16	28.41	9.62
VGG-19	27.62	9.12
VGG-11 with batch normalization	29.62	10.19
VGG-13 with batch normalization	28.45	9.63
VGG-16 with batch normalization	26.63	8.50
VGG-19 with batch normalization	25.76	8.15

- 같은 이름의 network여도 layer의 개수나 batch normalization같은 경우에 따라 종류가 다양
- 각 network의 error를 함께 보여줌

Pre-trained 모델 불러오기

```
import torchvision.models as models
vgg16 = models.vgg16(pretrained=True)
```

```
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /root/.cache/torch/hub/checkpoints/vgg16-3979
100% 528M/528M [00:11<00:00, 49.9MB/s]
```

- Pretrained=True라고 줌

```
def _vgg(arch: str, cfg: str, batch_norm: bool, pretrained: bool, progress: bool, **kwargs: Any) -> VGG:
    if pretrained:
        kwargs['init_weights'] = False
    model = VGG(make_layers(cfgs[cfg], batch_norm=batch_norm), **kwargs)
    if pretrained:
        state_dict = load_state_dict_from_url(model_urls[arch],
                                              progress=progress)
        model.load_state_dict(state_dict)
    return model
```

```
model_urls = {
    'vgg11': 'https://download.pytorch.org/models/vgg11-bbd30ac9.pth',
    'vgg13': 'https://download.pytorch.org/models/vgg13-c768596a.pth',
    'vgg16': 'https://download.pytorch.org/models/vgg16-397923af.pth',
    'vgg19': 'https://download.pytorch.org/models/vgg19-dcbb9e9d.pth',
    'vgg11_bn': 'https://download.pytorch.org/models/vgg11_bn-6002323d.pth',
    'vgg13_bn': 'https://download.pytorch.org/models/vgg13_bn-abd245e5.pth',
    'vgg16_bn': 'https://download.pytorch.org/models/vgg16_bn-6c64b313.pth',
    'vgg19_bn': 'https://download.pytorch.org/models/vgg19_bn-c79401a0.pth',
}
```

- Url로부터 다운받아옴

Pre-trained 모델 사용 시나리오

- 초기 설정으로 사용

  Network의 초기화를 정말 random한 값으로 초기화하는 것이 아닌
  큰 데이터셋으로 학습된 Pre-trained 모델로 초기화함으로써
  이후의 학습은 pre-trained된 weight에서 출발하여 좀 더 관심 있는 목표를 추출하게끔
  Finetuing되게 학습

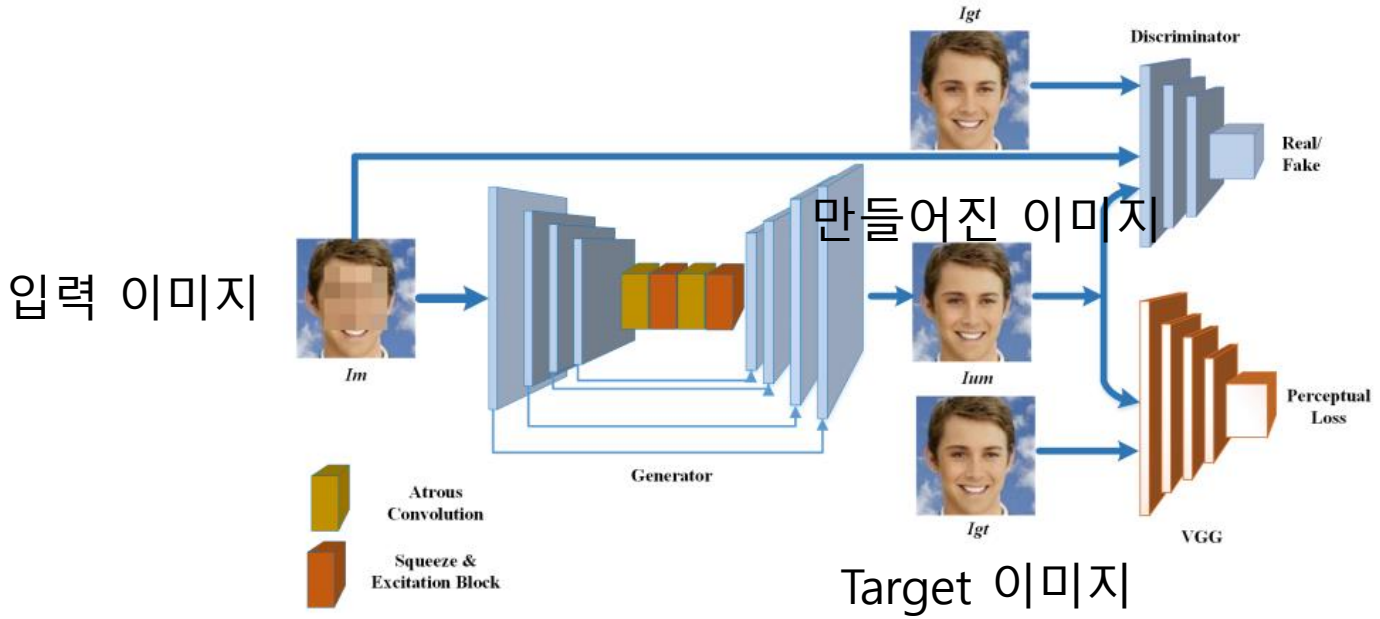
- 고정된 특징 추출기로 사용

  Fully connected layer를 제외한 모든 pre-trained weight를 고정하고
  Fully connected layer만을 random하게 초기화하여 이 부분만 학습
  ->이유 pre-trained된 모델은 해당 모델의 학습에 사용된 dataset의 class에 대해 분류
  그러나 우리가 분류하고자 하는 class의 개수는 다르므로

Pre-trained 모델 사용 시나리오

- Loss 계산에 사용(perceptual loss)

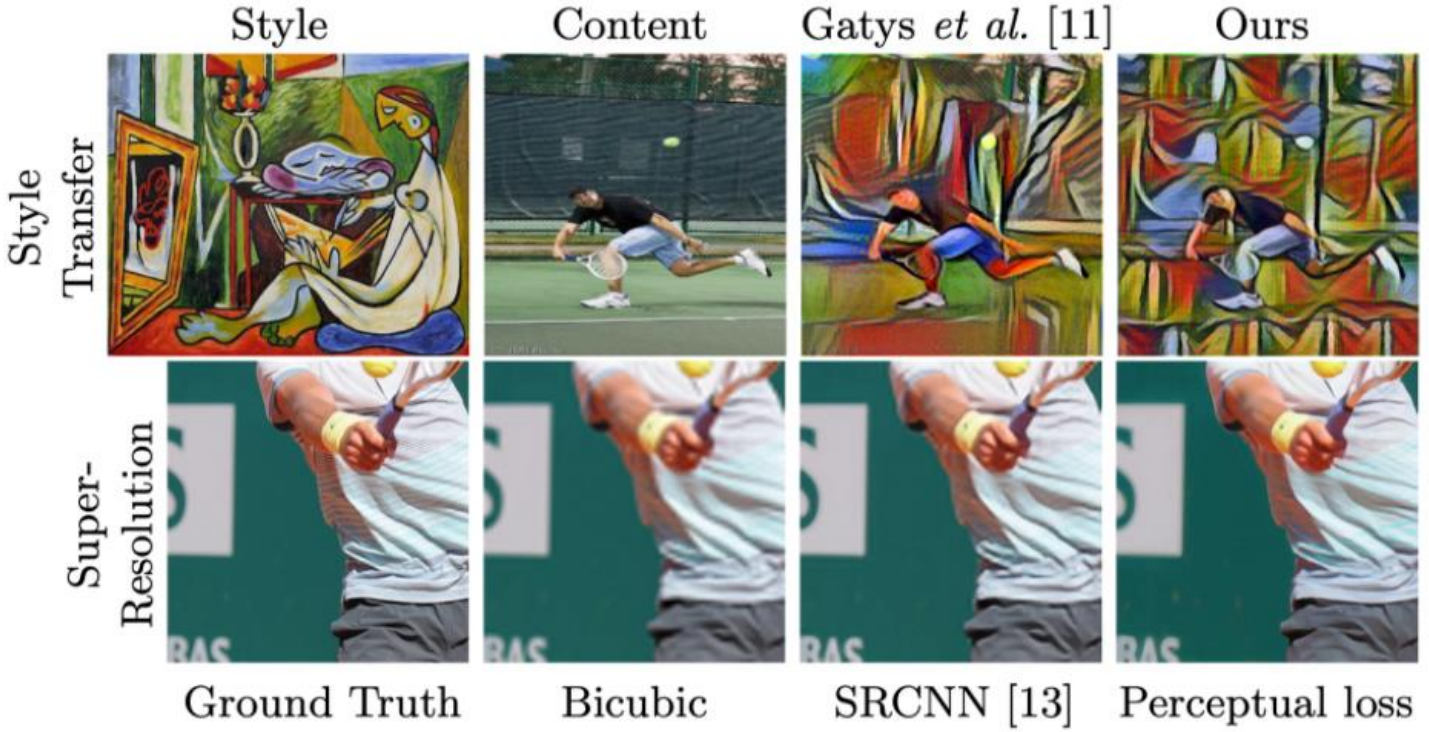
  Target 이미지와 만들어진 이미지를 pre-trained된 model에 입력으로 넣고
  그 결과로 나온 feature map 간의 loss를 계산
  Perceptual loss를 사용하면 결과의 detail이 좋아짐



Pre-trained 모델 사용 시나리오

- Loss 계산에 사용(perceptual loss)

  Target 이미지와 만들어진 이미지를 pre-trained된 model에 입력으로 넣고
  그 결과로 나온 feature map 간의 loss를 계산
  Perceptual loss를 사용하면 결과의 detail이 좋아짐



Reference

- Pretrined 모델 사용 시나리오 1,2 예제
- https://tutorials.pytorch.kr/beginner/transfer_learning_tutorial.html

- Pytorch.models
- https://pytorch.org/docs/stable/torchvision/models.html#classification

resnet.py

vision/torchvision/models/resnet.py 코드 분석

Models, Convolutions

```python
import torch
from torch import Tensor
import torch.nn as nn
from .utils import load_state_dict_from_url
from typing import Type, Any, Callable, Union, List, Optional
```

```python
__all__ = ['ResNet', 'resnet18', 'resnet34', 'resnet50', 'resnet101',
           'resnet152', 'resnext50_32x4d', 'resnext101_32x8d',
           'wide_resnet50_2', 'wide_resnet101_2']
```

```python
model_urls = {
    'resnet18': 'https://download.pytorch.org/models/resnet18-5c106cde.pth',
    'resnet34': 'https://download.pytorch.org/models/resnet34-333f7ec4.pth',
    'resnet50': 'https://download.pytorch.org/models/resnet50-19c8e357.pth',
    'resnet101': 'https://download.pytorch.org/models/resnet101-5d3b4d8f.pth',
    'resnet152': 'https://download.pytorch.org/models/resnet152-b121ed2d.pth',
    'resnext50_32x4d': 'https://download.pytorch.org/models/resnext50_32x4d-7cdf4587.pth',
    'resnext101_32x8d': 'https://download.pytorch.org/models/resnext101_32x8d-8ba56ff5.pth',
    'wide_resnet50_2': 'https://download.pytorch.org/models/wide_resnet50_2-95faca4d.pth',
    'wide_resnet101_2': 'https://download.pytorch.org/models/wide_resnet101_2-32ee1156.pth',
}
```

```python
def conv3x3(in_planes: int, out_planes: int, stride: int = 1, groups: int = 1, dilation: int = 1) -> nn.Conv2d:
    """3x3 convolution with padding"""
    return nn.Conv2d(in_planes, out_planes, kernel_size=3, stride=stride,
                     padding=dilation, groups=groups, bias=False, dilation=dilation)
```

```python
def conv1x1(in_planes: int, out_planes: int, stride: int = 1) -> nn.Conv2d:
    """1x1 convolution"""
    return nn.Conv2d(in_planes, out_planes, kernel_size=1, stride=stride, bias=False)
```

BasicBlock 클래스

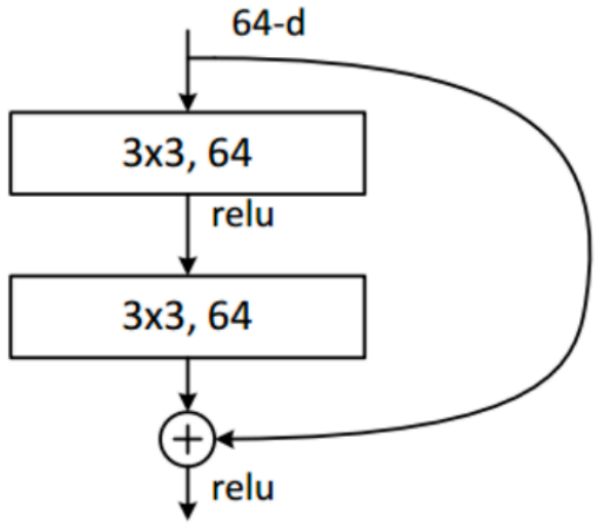
```python
class BasicBlock(nn.Module):
    expansion: int = 1

    def __init__(
        self,
        inplanes: int,
        planes: int,
        stride: int = 1,
        downsample: Optional[nn.Module] = None,
        groups: int = 1,
        base_width: int = 64,
        dilation: int = 1,
        norm_layer: Optional[Callable[..., nn.Module]] = None
    ) -> None:
        super(BasicBlock, self).__init__()
        if norm_layer is None:
            norm_layer = nn.BatchNorm2d
        if groups != 1 or base_width != 64:
            raise ValueError('BasicBlock only supports groups=1 and base_width=64')
        if dilation > 1:
            raise NotImplementedError("Dilation > 1 not supported in BasicBlock")
        # Both self.conv1 and self.downsample layers downsample the input when stride != 1
```

→ feature map 1배 확장

BasicBlock 클래스

```
self.conv1 = conv3x3(inplanes, planes, stride)
self.bn1 = norm_layer(planes)
self.relu = nn.ReLU(inplace=True)
self.conv2 = conv3x3(planes, planes)
self.bn2 = norm_layer(planes)
self.downsample = downsample
self.stride = stride
```

```
def forward(self, x: Tensor) -> Tensor:
    identity = x

    out = self.conv1(x)
    out = self.bn1(out)
    out = self.relu(out)

    out = self.conv2(out)
    out = self.bn2(out)

    if self.downsample is not None:
        identity = self.downsample(x)

    out += identity
    out = self.relu(out)

    return out
```



skip connection:

$$f(x) \rightarrow x + f(x)$$

Bottleneck 클래스

```
class Bottleneck(nn.Module):
    # Bottleneck in torchvision places the stride for downsampling at 3x3 convolution(self.conv2)
    # while original implementation places the stride at the first 1x1 convolution(self.conv1)
    # according to "Deep residual learning for image recognition"https://arxiv.org/abs/1512.03385.
    # This variant is also known as ResNet V1.5 and improves accuracy according to
    # https://ngc.nvidia.com/catalog/model-scripts/nvidia:resnet_50_v1_5_for_pytorch.

    expansion: int = 4
```

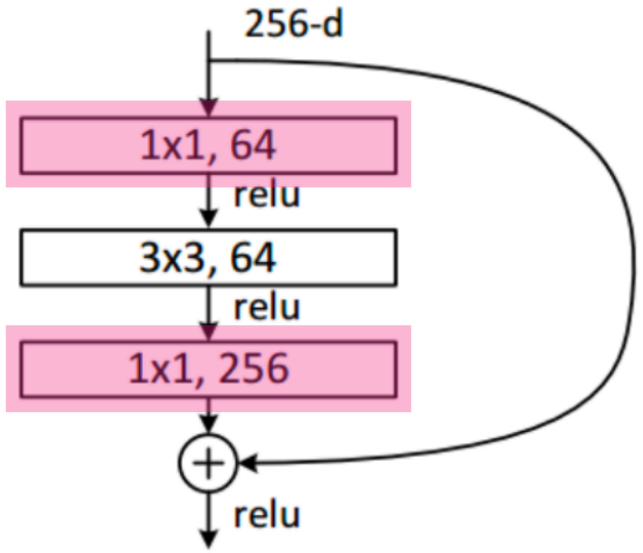
→ feature map 4배 확장

```
    def __init__(
        self,
        inplanes: int,
        planes: int,
        stride: int = 1,
        downsample: Optional[nn.Module] = None,
        groups: int = 1,
        base_width: int = 64,
        dilation: int = 1,
        norm_layer: Optional[Callable[..., nn.Module]] = None
    ) -> None:
        super(Bottleneck, self).__init__()
        if norm_layer is None:
            norm_layer = nn.BatchNorm2d
        width = int(planes * (base_width / 64.)) * groups
        # Both self.conv2 and self.downsample layers downsample the input when stride != 1
```

Bottleneck 클래스

```
self.conv1 = conv1x1(inplanes, width)
self.bn1 = norm_layer(width)
self.conv2 = conv3x3(width, width, stride, groups, dilation)
self.bn2 = norm_layer(width)
self.conv3 = conv1x1(width, planes * self.expansion)
self.bn3 = norm_layer(planes * self.expansion)
self.relu = nn.ReLU(inplace=True)
self.downsample = downsample
self.stride = stride
```

```
def forward(self, x: Tensor) -> Tensor:
    identity = x

    out = self.conv1(x)
    out = self.bn1(out)
    out = self.relu(out)

    out = self.conv2(out)
    out = self.bn2(out)
    out = self.relu(out)

    out = self.conv3(out)
    out = self.bn3(out)

    if self.downsample is not None:
        identity = self.downsample(x)

    out += identity
    out = self.relu(out)

    return out
```



skip connection:

$$f(x) \rightarrow x + f(x)$$

ResNet 클래스

```
class ResNet(nn.Module):

    def __init__(
        self,
        block: Type[Union[BasicBlock, Bottleneck]],
        layers: List[int],
        num_classes: int = 1000,
        zero_init_residual: bool = False,
        groups: int = 1,
        width_per_group: int = 64,
        replace_stride_with_dilation: Optional[List[bool]] = None
        norm_layer: Optional[Callable[..., nn.Module]] = None
    ) -> None:
```

```
super(ResNet, self).__init__()
if norm_layer is None:
    norm_layer = nn.BatchNorm2d
self._norm_layer = norm_layer

self.inplanes = 64
self.dilation = 1
if replace_stride_with_dilation is None:
    # each element in the tuple indicates if we should replace
    # the 2x2 stride with a dilated convolution instead
    replace_stride_with_dilation = [False, False, False]
if len(replace_stride_with_dilation) != 3:
    raise ValueError("replace_stride_with_dilation should be None "
                     "or a 3-element tuple, got {}".format(replace_stride_with_dilation))
```

ResNet 클래스

```
self.groups = groups
self.base_width = width_per_group
self.conv1 = nn.Conv2d(3, self.inplanes, kernel_size=7, stride=2, padding=3,
                       bias=False)
self.bn1 = norm_layer(self.inplanes)
self.relu = nn.ReLU(inplace=True)
self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
self.layer1 = self._make_layer(block, 64, layers[0])
self.layer2 = self._make_layer(block, 128, layers[1], stride=2,
                               dilate=replace_stride_with_dilation[0])
self.layer3 = self._make_layer(block, 256, layers[2], stride=2,
                               dilate=replace_stride_with_dilation[1])
self.layer4 = self._make_layer(block, 512, layers[3], stride=2,
                               dilate=replace_stride_with_dilation[2])
self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
self.fc = nn.Linear(512 * block.expansion, num_classes)
```

```
for m in self.modules():
    if isinstance(m, nn.Conv2d):
        nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
    elif isinstance(m, (nn.BatchNorm2d, nn.GroupNorm)):
        nn.init.constant_(m.weight, 1)
        nn.init.constant_(m.bias, 0)

# Zero-initialize the last BN in each residual branch,
# so that the residual branch starts with zeros, and each residual block behaves like an identity.
# This improves the model by 0.2~0.3% according to https://arxiv.org/abs/1706.02677
if zero_init_residual:
    for m in self.modules():
        if isinstance(m, Bottleneck):
            nn.init.constant_(m.bn3.weight, 0)  # type: ignore[arg-type]
        elif isinstance(m, BasicBlock):
            nn.init.constant_(m.bn2.weight, 0)  # type: ignore[arg-type]
```

ResNet 클래스

```
def _make_layer(self, block: Type[Union[BasicBlock, Bottleneck]], planes: int, blocks: int,
                stride: int = 1, dilate: bool = False) -> nn.Sequential:
    norm_layer = self._norm_layer
    downsample = None
    previous_dilation = self.dilation
    if dilate:
        self.dilation *= stride
        stride = 1
    if stride != 1 or self.inplanes != planes * block.expansion:
        downsample = nn.Sequential(
            conv1x1(self.inplanes, planes * block.expansion, stride),
            norm_layer(planes * block.expansion),
        )

    layers = []
    layers.append(block(self.inplanes, planes, stride, downsample, self.groups,
                        self.base_width, previous_dilation, norm_layer))
    self.inplanes = planes * block.expansion
    for _ in range(1, blocks):
        layers.append(block(self.inplanes, planes, groups=self.groups,
                            base_width=self.base_width, dilation=self.dilation,
                            norm_layer=norm_layer))

    return nn.Sequential(*layers)
```

```
def _forward_impl(self, x: Tensor) -> Tensor:
    # See note [TorchScript super()]
    x = self.conv1(x)
    x = self.bn1(x)
    x = self.relu(x)
    x = self.maxpool(x)

    x = self.layer1(x)
    x = self.layer2(x)
    x = self.layer3(x)
    x = self.layer4(x)

    x = self.avgpool(x)
    x = torch.flatten(x, 1)
    x = self.fc(x)

    return x

def forward(self, x: Tensor) -> Tensor:
    return self._forward_impl(x)
```

_resnet()

```
def _resnet(
    arch: str,
    block: Type[Union[BasicBlock, Bottleneck]],
    layers: List[int],
    pretrained: bool,
    progress: bool,
    **kwargs: Any
) -> ResNet:
    model = ResNet(block, layers, **kwargs)
    if pretrained:
        state_dict = load_state_dict_from_url(model_urls[arch],
                                              progress=progress)
        model.load_state_dict(state_dict)
    return model
```

`torch.hub.load_state_dict_from_url(`*`url`*`, `*`model_dir=None`*`, `*`map_location=None`*`, `*`progress=True`*`, `*`check_hash=False`*`, `*`file_name=None`*`)` [SOURCE]

Loads the Torch serialized object at the given URL.

If downloaded file is a zip file, it will be automatically decompressed.

If the object is already present in *model_dir*, it's deserialized and returned. The default value of *model_dir* is `<hub_dir>/checkpoints` where *hub_dir* is the directory returned by `get_dir()`.

`load_state_dict(`*`state_dict: Dict[str, torch.Tensor], strict: bool = True`*`)` [SOURCE]

→ 옵션을 받아서 ResNet 모델 반환

Copies parameters and buffers from `state_dict` into this module and its descendants. If `strict` is `True`, then the keys of `state_dict` must exactly match the keys returned by this module's `state_dict()` function.

여러 가지 모델들

```
def resnet18(pretrained: bool = False, progress: bool = True, **kwargs: Any) -> ResNet:
    r"""ResNet-18 model from
    `"Deep Residual Learning for Image Recognition" <https://arxiv.org/pdf/1512.03385.pdf>`_.

    Args:
        pretrained (bool): If True, returns a model pre-trained on ImageNet
        progress (bool): If True, displays a progress bar of the download to stderr
    """
    return _resnet('resnet18', BasicBlock, [2, 2, 2, 2], pretrained, progress,
                   **kwargs)


def resnet34(pretrained: bool = False, progress: bool = True, **kwargs: Any) -> ResNet:
    r"""ResNet-34 model from
    `"Deep Residual Learning for Image Recognition" <https://arxiv.org/pdf/1512.03385.pdf>`_.

    Args:
        pretrained (bool): If True, returns a model pre-trained on ImageNet
        progress (bool): If True, displays a progress bar of the download to stderr
    """
    return _resnet('resnet34', BasicBlock, [3, 4, 6, 3], pretrained, progress,
                   **kwargs)


def resnet50(pretrained: bool = False, progress: bool = True, **kwargs: Any) -> ResNet:
    r"""ResNet-50 model from
    `"Deep Residual Learning for Image Recognition" <https://arxiv.org/pdf/1512.03385.pdf>`_.

    Args:
        pretrained (bool): If True, returns a model pre-trained on ImageNet
        progress (bool): If True, displays a progress bar of the download to stderr
    """
    return _resnet('resnet50', Bottleneck, [3, 4, 6, 3], pretrained, progress,
                   **kwargs)
```

```
def resnet101(pretrained: bool = False, progress: bool = True, **kwargs: Any) -> ResNet:
    r"""ResNet-101 model from
    `"Deep Residual Learning for Image Recognition" <https://arxiv.org/pdf/1512.03385.pdf>`_.

    Args:
        pretrained (bool): If True, returns a model pre-trained on ImageNet
        progress (bool): If True, displays a progress bar of the download to stderr
    """
    return _resnet('resnet101', Bottleneck, [3, 4, 23, 3], pretrained, progress,
                   **kwargs)


def resnet152(pretrained: bool = False, progress: bool = True, **kwargs: Any) -> ResNet:
    r"""ResNet-152 model from
    `"Deep Residual Learning for Image Recognition" <https://arxiv.org/pdf/1512.03385.pdf>`_.

    Args:
        pretrained (bool): If True, returns a model pre-trained on ImageNet
        progress (bool): If True, displays a progress bar of the download to stderr
    """
    return _resnet('resnet152', Bottleneck, [3, 8, 36, 3], pretrained, progress,
                   **kwargs)


def resnext50_32x4d(pretrained: bool = False, progress: bool = True, **kwargs: Any) -> ResNet:
    r"""ResNeXt-50 32x4d model from
    `"Aggregated Residual Transformation for Deep Neural Networks" <https://arxiv.org/pdf/1611.05431.pdf>`_.

    Args:
        pretrained (bool): If True, returns a model pre-trained on ImageNet
        progress (bool): If True, displays a progress bar of the download to stderr
    """
    kwargs['groups'] = 32
    kwargs['width_per_group'] = 4
    return _resnet('resnext50_32x4d', Bottleneck, [3, 4, 6, 3],
                   pretrained, progress, **kwargs)
```

여러 가지 모델들

```
def resnext101_32x8d(pretrained: bool = False, progress: bool = True, **kwargs: Any) -> ResNet:
    r"""ResNeXt-101 32x8d model from
    `"Aggregated Residual Transformation for Deep Neural Networks" <https://arxiv.org/pdf/1611.05431.pdf>`_.

    Args:
        pretrained (bool): If True, returns a model pre-trained on ImageNet
        progress (bool): If True, displays a progress bar of the download to stderr
    """
    kwargs['groups'] = 32
    kwargs['width_per_group'] = 8
    return _resnet('resnext101_32x8d', Bottleneck, [3, 4, 23, 3],
                   pretrained, progress, **kwargs)


def wide_resnet50_2(pretrained: bool = False, progress: bool = True, **kwargs: Any) -> ResNet:
    r"""Wide ResNet-50-2 model from
    `"Wide Residual Networks" <https://arxiv.org/pdf/1605.07146.pdf>`_.

    The model is the same as ResNet except for the bottleneck number of channels
    which is twice larger in every block. The number of channels in outer 1x1
    convolutions is the same, e.g. last block in ResNet-50 has 2048-512-2048
    channels, and in Wide ResNet-50-2 has 2048-1024-2048.

    Args:
        pretrained (bool): If True, returns a model pre-trained on ImageNet
        progress (bool): If True, displays a progress bar of the download to stderr
    """
    kwargs['width_per_group'] = 64 * 2
    return _resnet('wide_resnet50_2', Bottleneck, [3, 4, 6, 3],
                   pretrained, progress, **kwargs)
```

```
def wide_resnet101_2(pretrained: bool = False, progress: bool = True, **kwargs: Any) -> ResNet:
    r"""Wide ResNet-101-2 model from
    `"Wide Residual Networks" <https://arxiv.org/pdf/1605.07146.pdf>`_.

    The model is the same as ResNet except for the bottleneck number of channels
    which is twice larger in every block. The number of channels in outer 1x1
    convolutions is the same, e.g. last block in ResNet-50 has 2048-512-2048
    channels, and in Wide ResNet-50-2 has 2048-1024-2048.

    Args:
        pretrained (bool): If True, returns a model pre-trained on ImageNet
        progress (bool): If True, displays a progress bar of the download to stderr
    """
    kwargs['width_per_group'] = 64 * 2
    return _resnet('wide_resnet101_2', Bottleneck, [3, 4, 23, 3],
                   pretrained, progress, **kwargs)
```

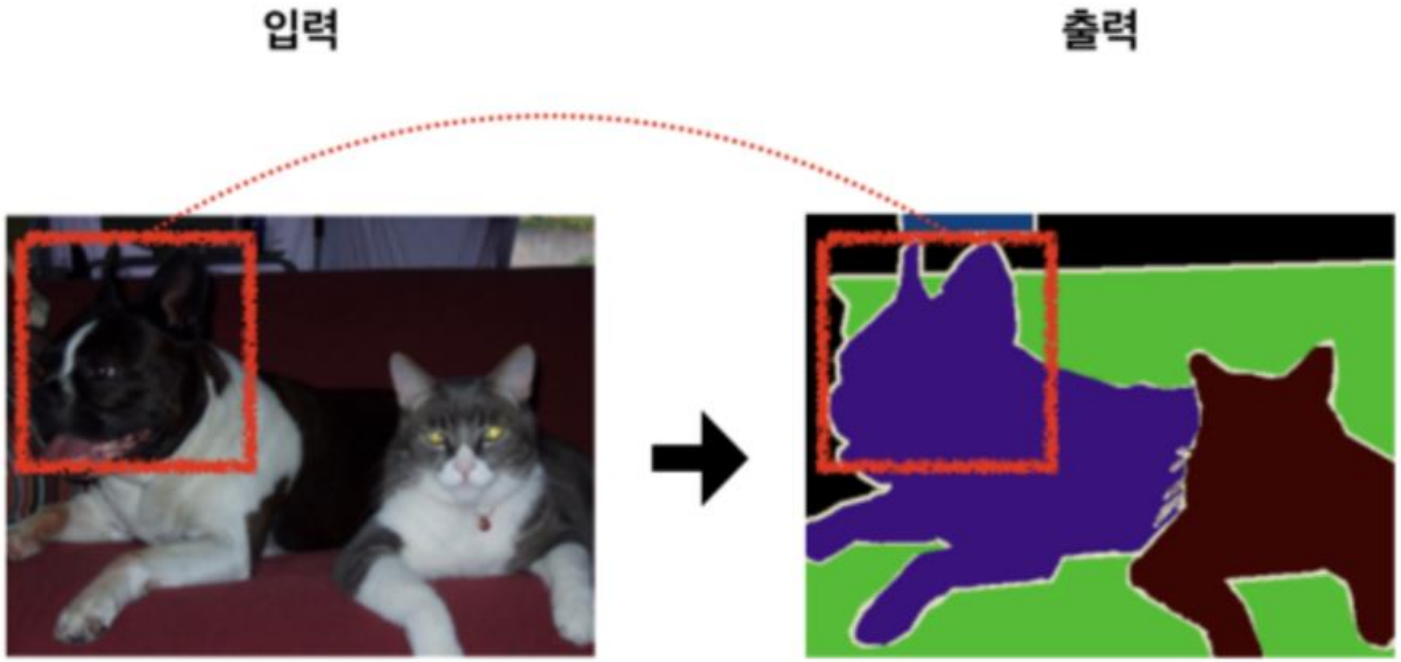
Reference

- https://github.com/pytorch/vision/blob/c5d6f1f3f921f4fc1be3797f891fe7ac473dca79/torchvision/models/resnet.py
- https://pytorch.org/docs/stable/generated/torch.nn.Module.html
- https://pytorch.org/docs/stable/hub.html
- https://discuss.pytorch.org/t/make-layers-in-torchvision-resnet-implementation/36260/2

FCN

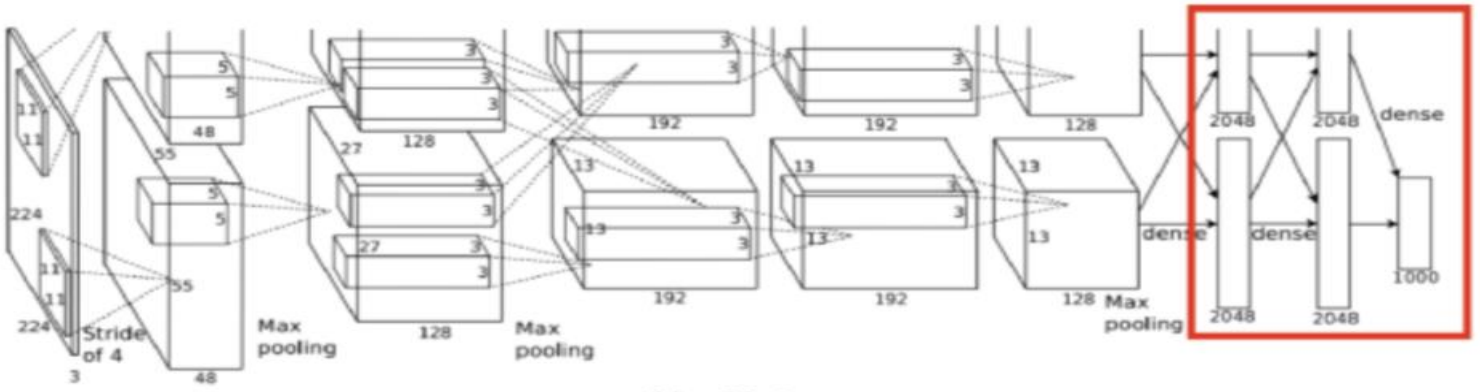
- Semantic Segmentation
- 기존 CNN의 한계
- Method

Semantic segmentation의 목적

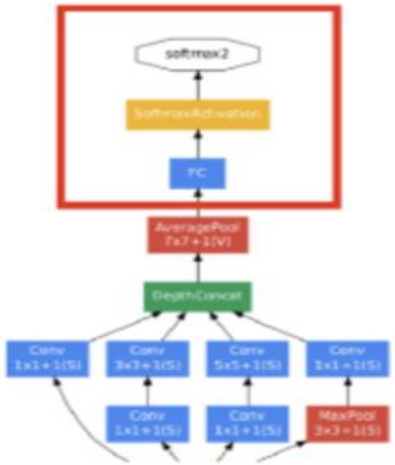


→ Segmentation은 입력 이미지의 위치 정보가 유지되어야 함.

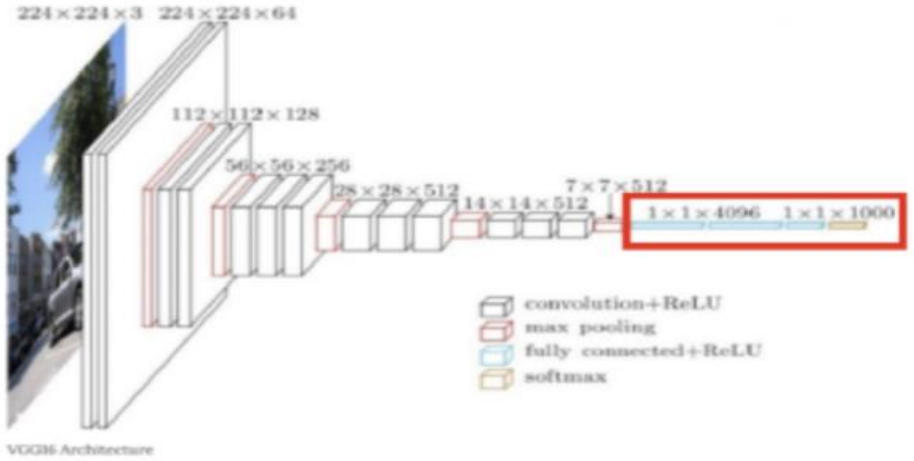
기존 classification process가 갖는 segmentation domain의 한계



AlexNet

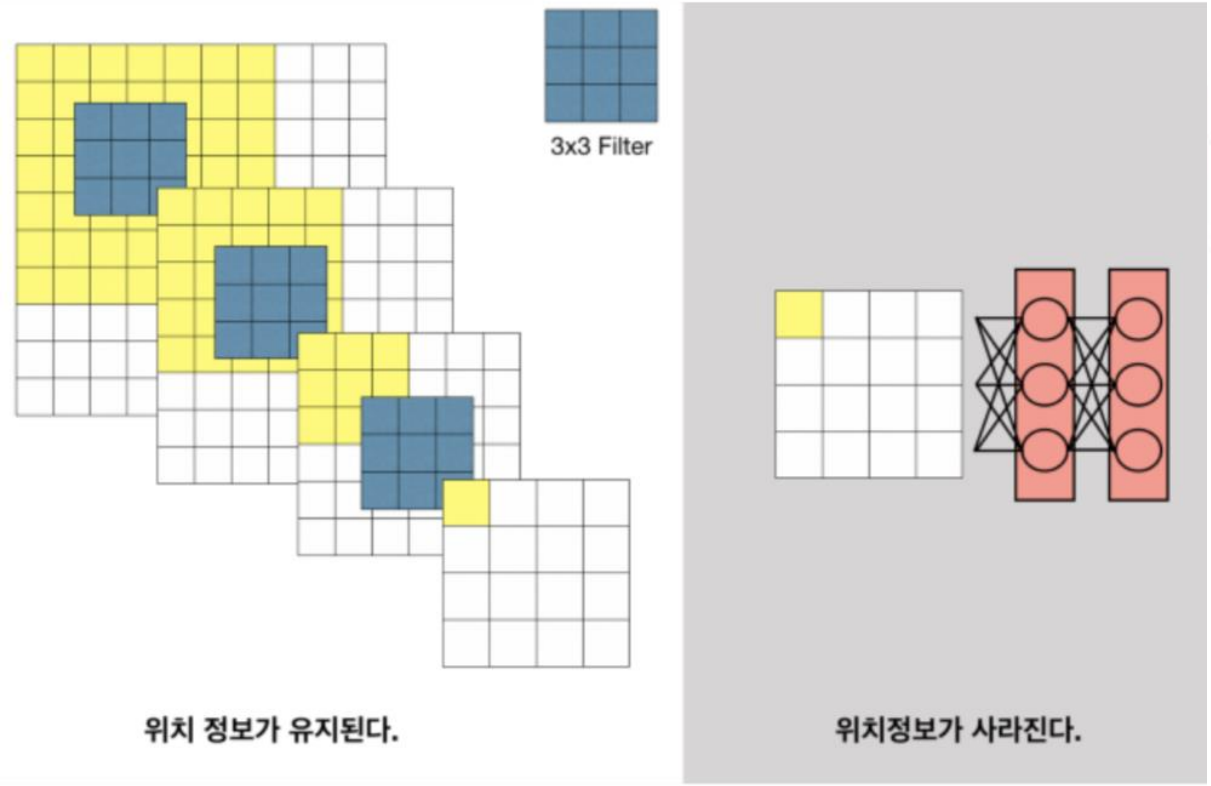


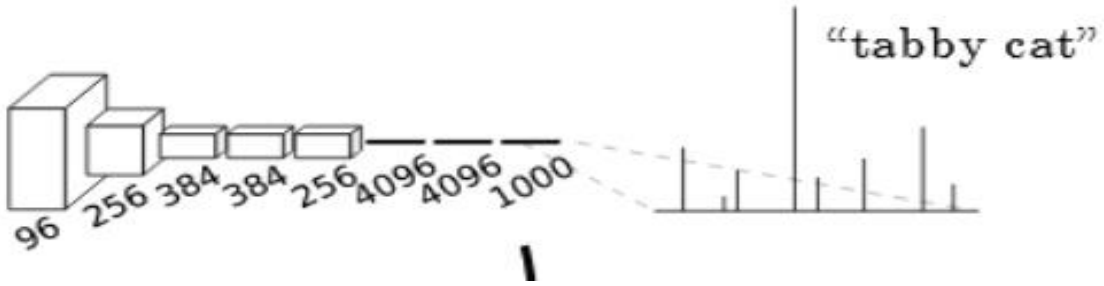
GoogLeNet



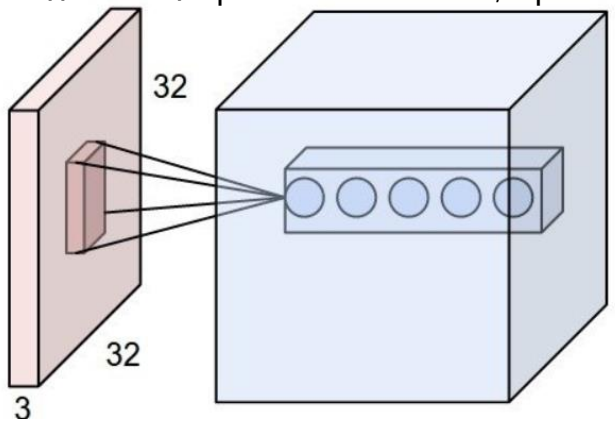
VGG16

한계 1: 위치정보 손실



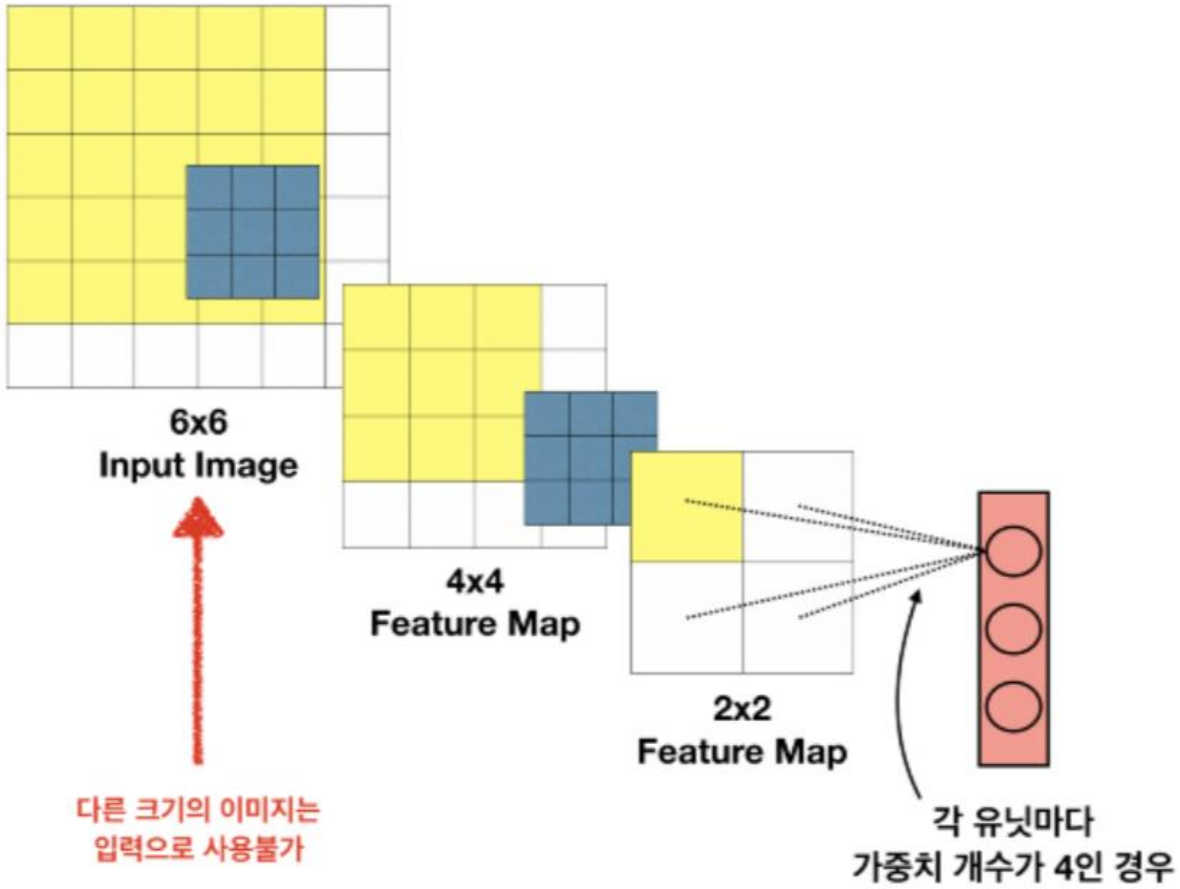


Receptive field: 커널의 크기로 이해, 일반적인 CNN에서 output layer의 뉴런 하나 당 영향을 끼치는 input layer 뉴런들의 spatial dimension, space를 의미함.



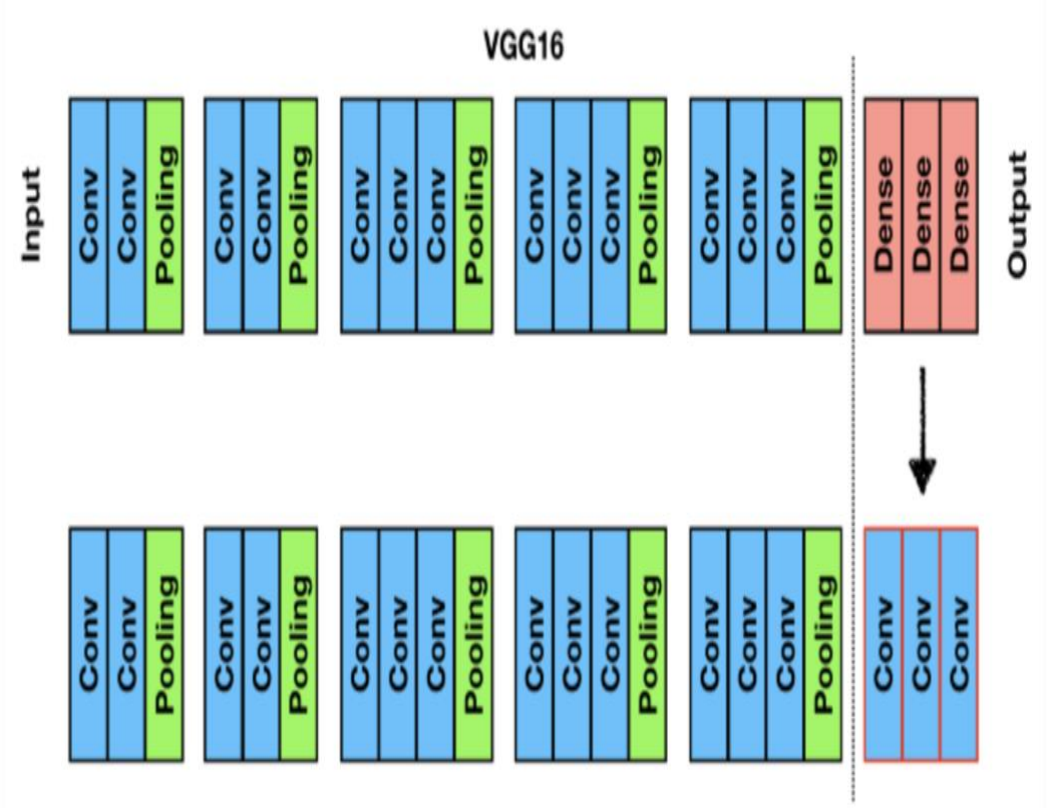
→ Conv 레이어를 거칠 때만 해도, 커널을 이용한 연산이라는 특징 때문에 receptive field라는 정보를 유지, 곧 위치 정보에 해당하는 연산 결과를 얻을 수 있었는데, FCL을 거치면 없어짐.

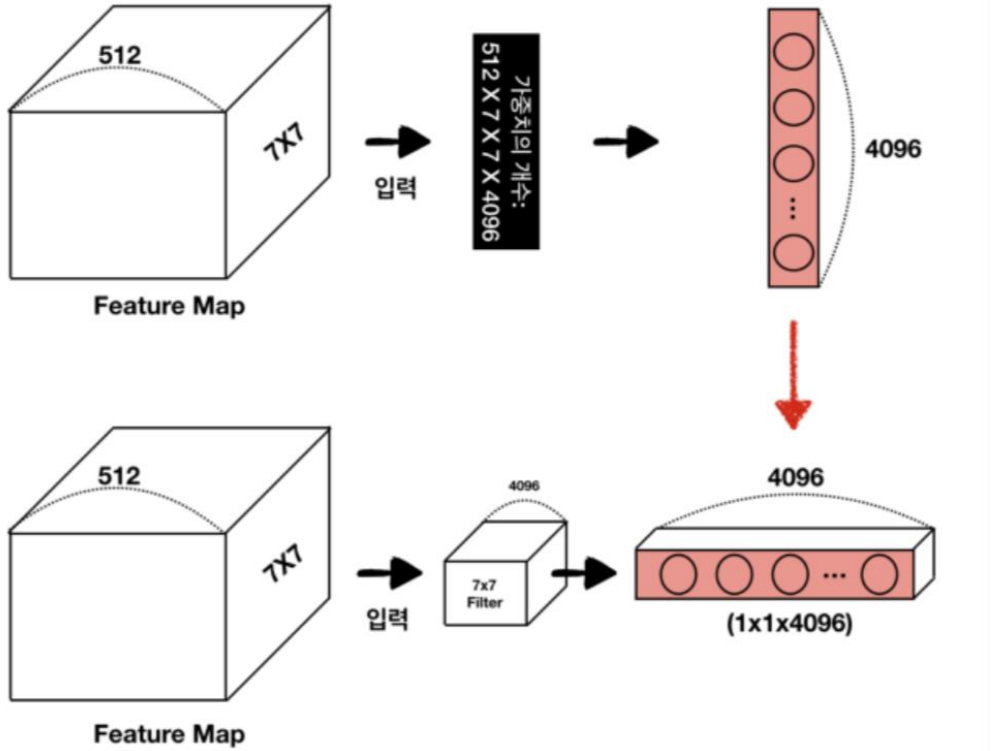
한계 2: 입력 이미지 크기가 고정된다.



- FCL로 넘어가기 전에 2x2 feature map을 flatten, 이후 FCL의 뉴런 개수를 우리가 임의대로 설정하겠지만, 그를 구성하는 가중치 개수는 우리가 설정해줘야 하고, 이 개수에 따라 연쇄적으로 feature map의 크기가 결정되다가 가장 초기의 input layer, 즉 input image 크기가 정해진다.

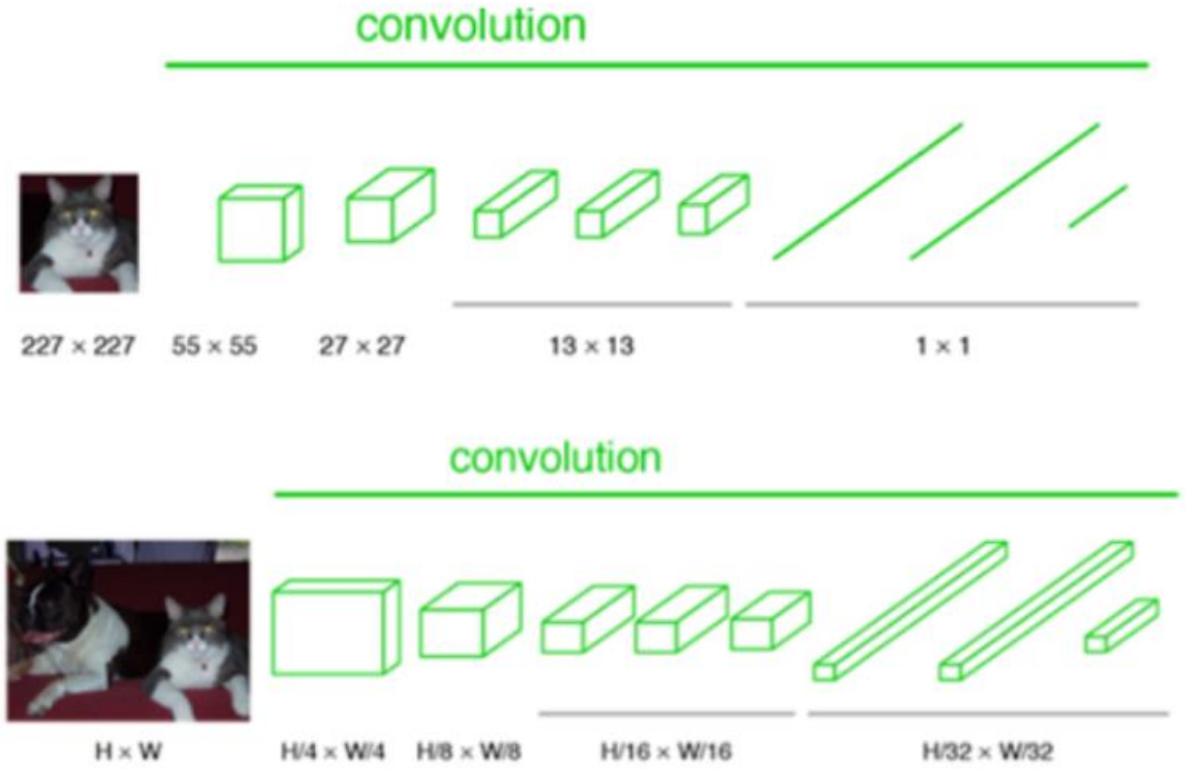
Method: Dense layers to Conv-layers





위의 예에서, FCL 대신 7x7x512 feature map에 7x7x512x4096 conv 연산 때리면 Parameter수가 동일 = 연산 모델은 같은건데 형태만 1d에서 이미지의 형태로 바뀜.

Method: 이전의 한계들은?

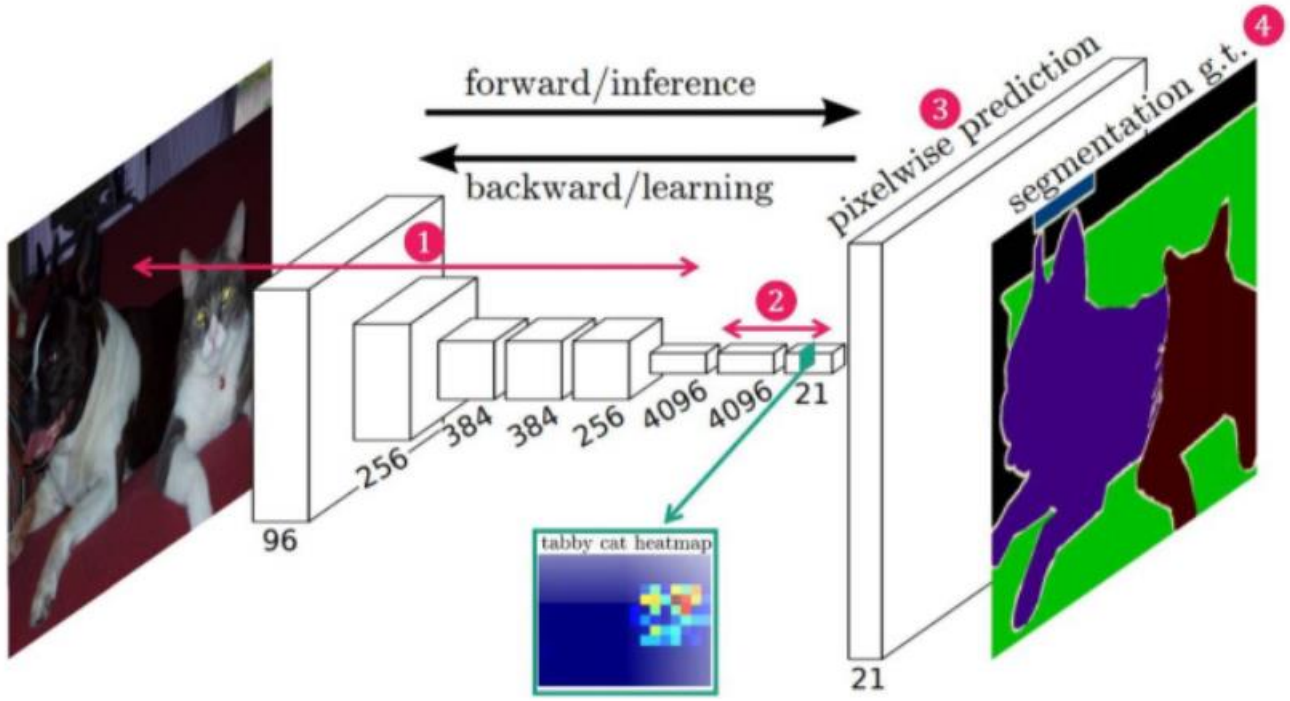


- FCL은 receptive field가 input의 전체 영역으로 보는 커널 conv layer라고 볼 수 있었는데, 이를 일반적인 conv layer로 바꿔줌으로써 위치 정보가 그대로 보존,

- 일반 CNN 대로, stride 2 max pooling이 layer마다 ½만큼 downsampling(?)하는 것뿐 --> 1*1 conv layers의 구성은 이전의 FCL이 가진 가중치 연쇄에 따른 고정 법칙을 탈피

FCN은 input dimension에 independent.

Heat map & Upsampling의 필요성



Heatmap 설명

- 21개의 heatmap : 21 label에 해당하는 heat map, feature map이 형성
→ 정답: 고양이 -> 개 레이블의 heat map에선 고양이 위치의 픽셀 값이 낮게 나옴.
- 다층의 Conv, Max pooling 연산에 의해 원래 이미지보다 굉장히 축소, 결과적으로 heat map에는 고차원의 정보가 압축되어 저장되어 있기에 (=Coarse) upsampling을 통하여 본래 목적이었던 픽셀 당 segmentation labeling을 위해 원래 이미지까지 늘리는게 요구됨.

Reference

- https://kuklife.tistory.com/117 , FCN 원리
- https://medium.com/@msmapark2/fcn-%EB%85%BC%EB%AC%B8-%EB%A6%AC%EB%B7%B0-fully-convolutional-networks-for-semantic-segmentation-81f016d76204 , FCN 논문 리뷰