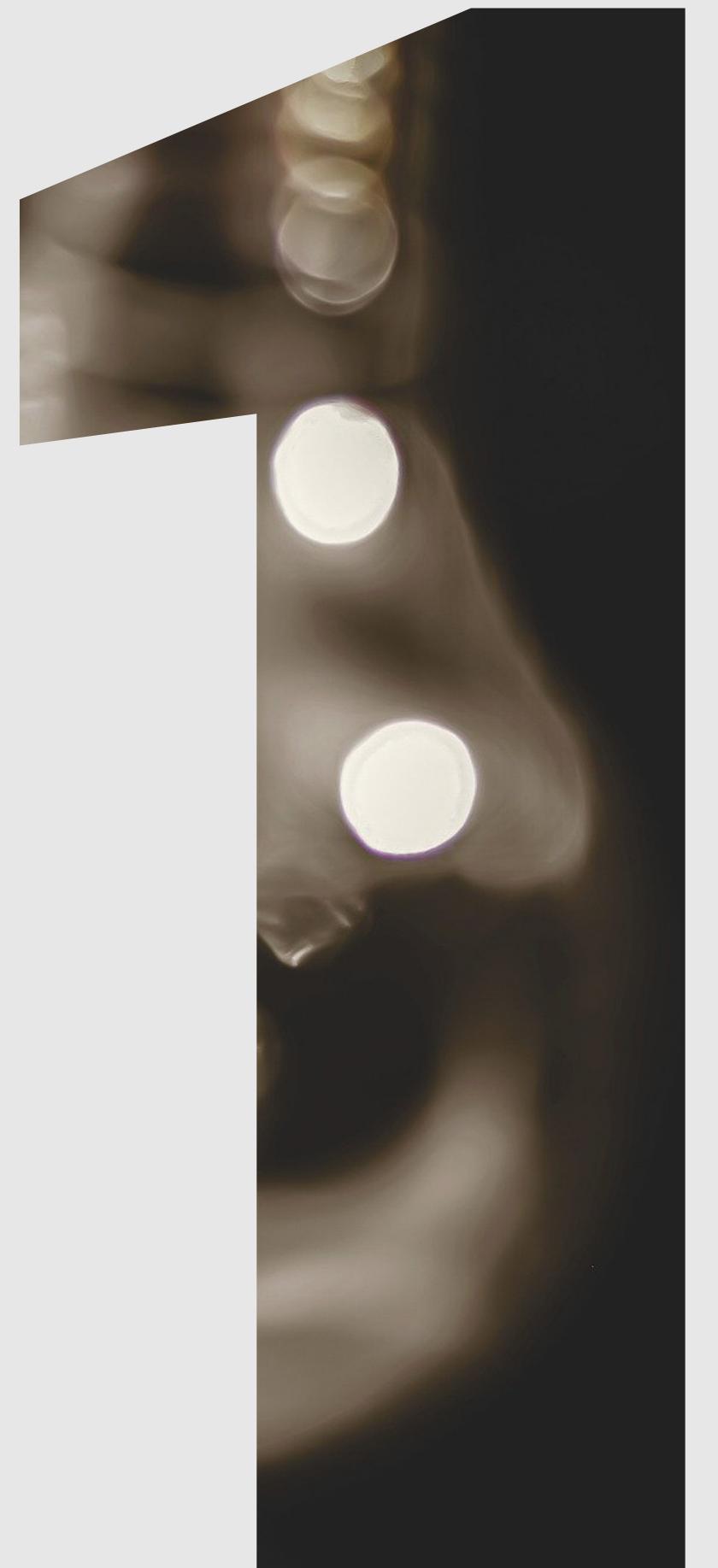


# 9-11차시. DeepLab



DeepLab

# 딥러닝을 도와주는 보조도구



<https://deepcognition.ai/>

딥러닝 GUI

<https://github.com/lutzroeder/netron>

딥러닝 GUI 분석기

# DeepLab



# DeepLab

## 논문

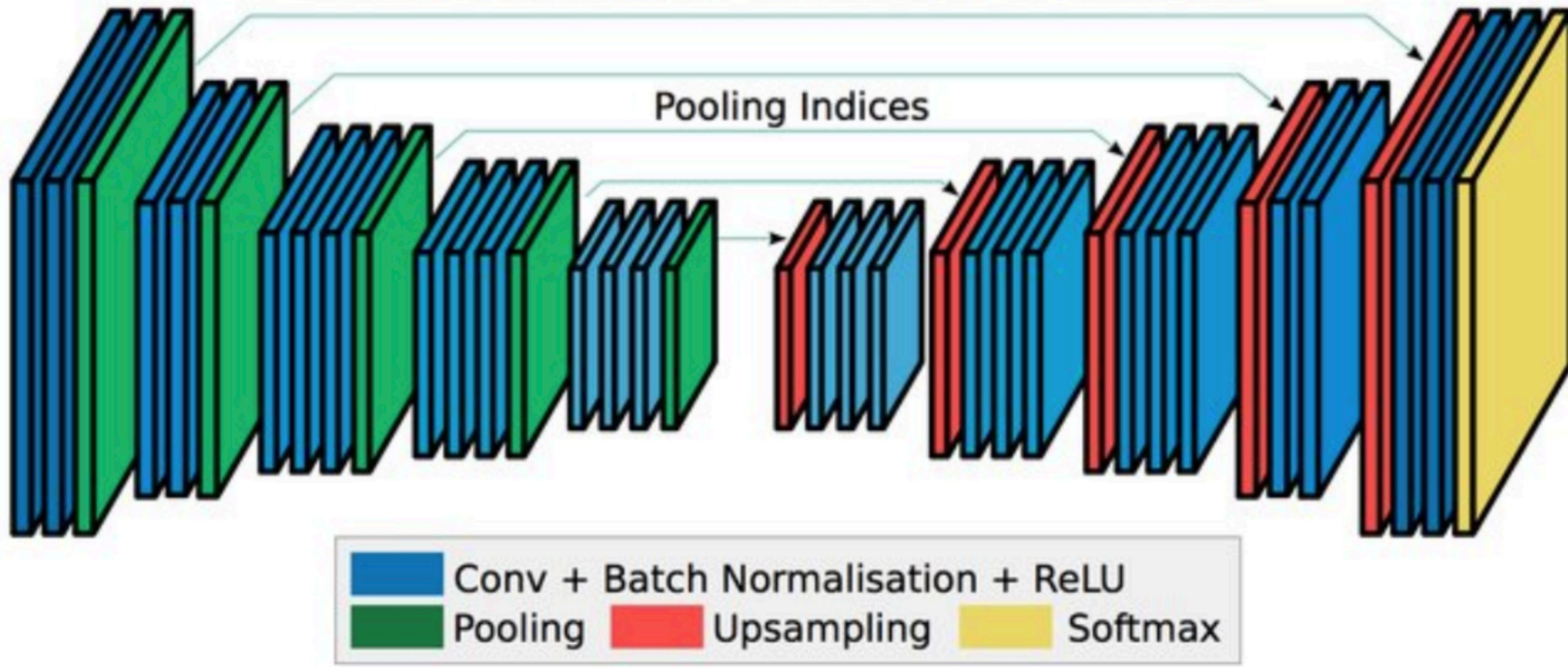
- DeepLab
  - [DeepLab-V1](<https://arxiv.org/abs/1412.7062>) : "Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs"
  - [DeepLab-V2](<https://arxiv.org/abs/1606.00915>) : "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs"
  - [DeepLab-V3](<https://arxiv.org/abs/1706.05587>) : "Rethinking Atrous Convolution for Semantic Image Segmentation"
  - [DeepLab-V3+](<https://arxiv.org/abs/1802.02611>) : "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation"
- V2에서 DeepLab 이라는 이름이 처음 나왔고, 현재 많이 일컬어지고 있는 것은 DeepLab-V3+ 이다.

# DeepLab

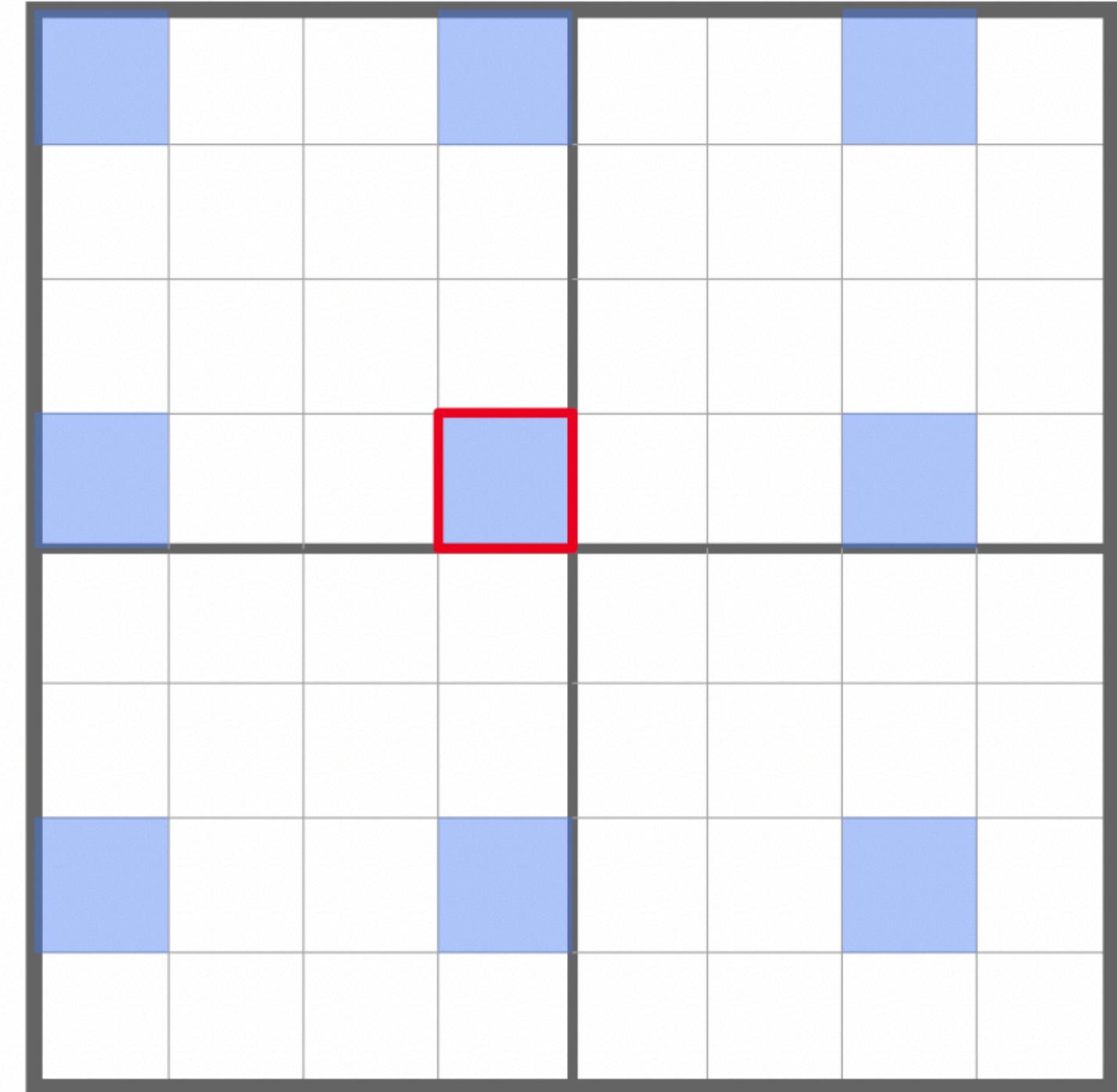
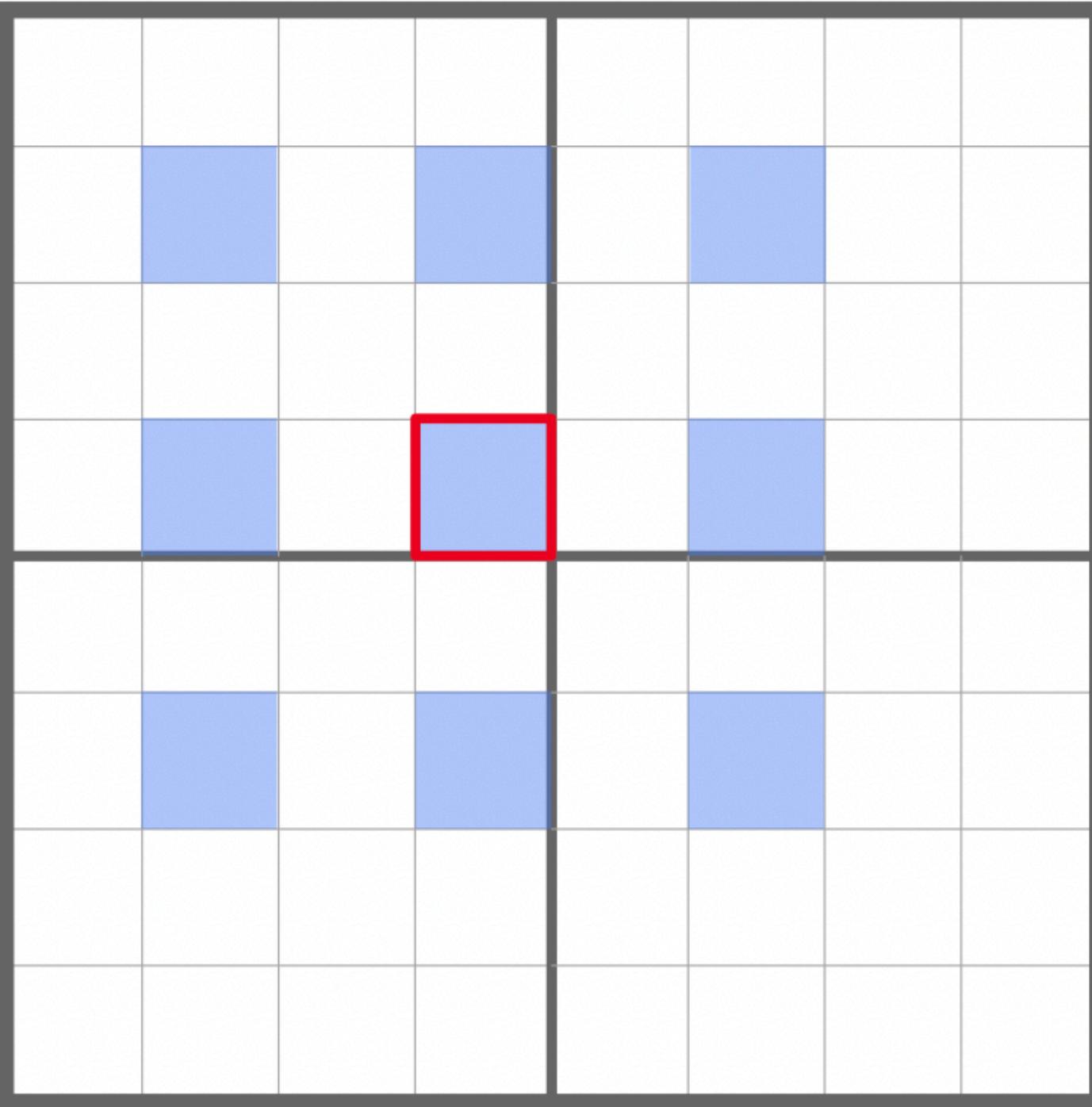
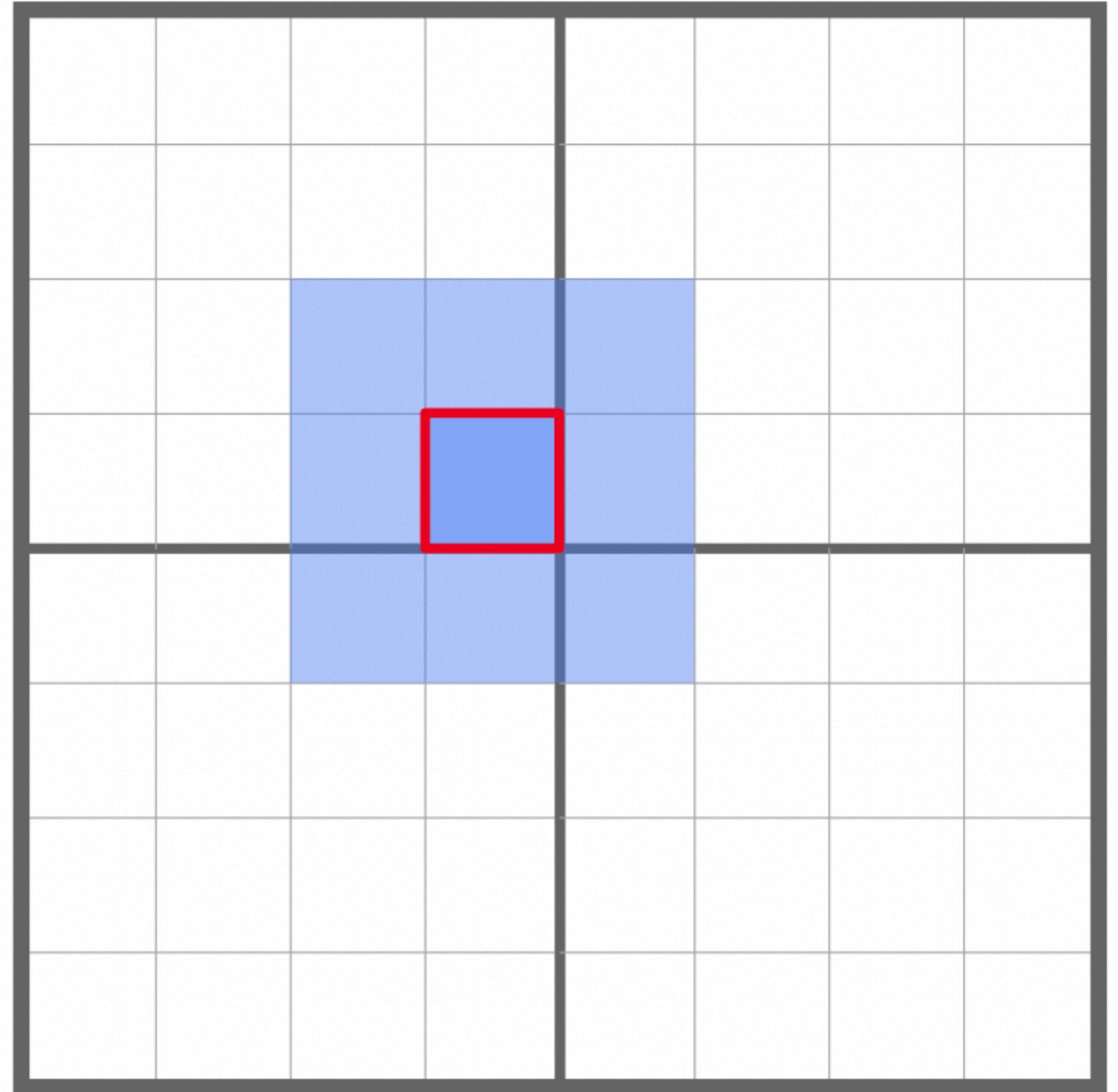
DeepLab 이란?

- atrous(dilated) 컨볼루션을 도입해서, CNN의 특성 응답의 해상도 컨트롤
- atrous spatial pyramid pooling (ASPP)의 사용으로, 서로 다른 객체 해상도 대응과 정확도 향상
- batch normalization의 도입
- decoder module로 세그먼테이션 결과 향상

# Convolutional Encoder-Decoder



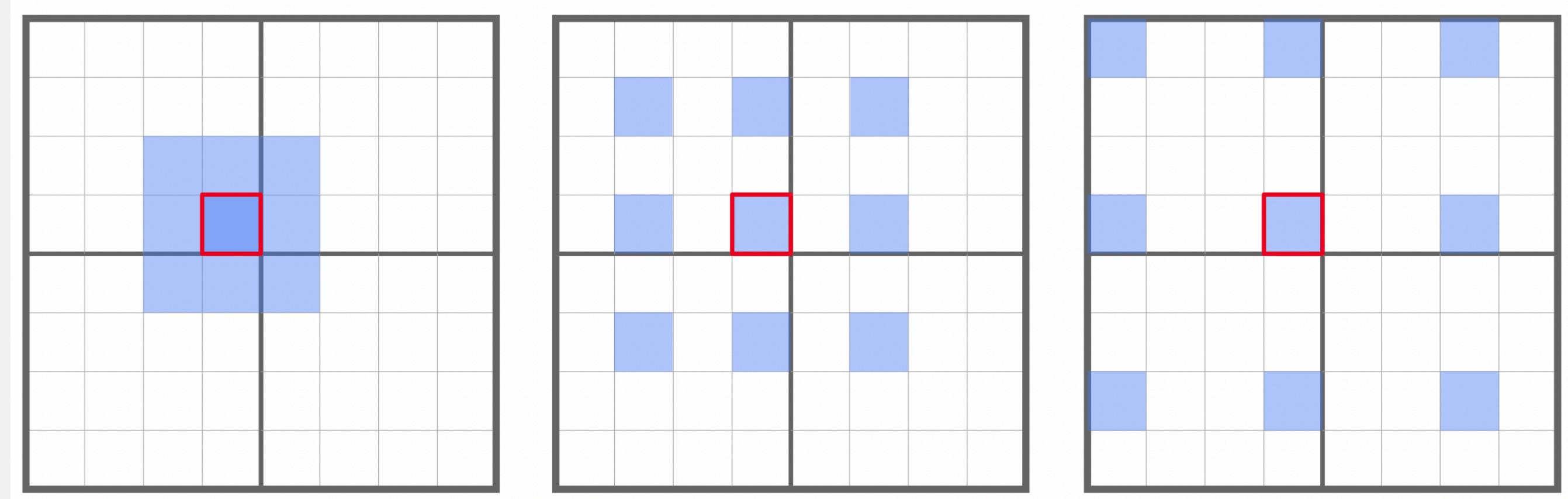
Encoder-Decoder 구조



# Atrous Convolution 구조

# Atrous Convolution 구조

- 이렇게 하면, 주변 영역을 보기 위해 max pooling과 같은 sub sampling을 할 필요가 없다.
- 같은 연산으로 더 큰 특징을 잡아낼 수 있다.
- 다양한 확장 비율을 가진 atrous convolution을 병렬로 사용할 수 있다.

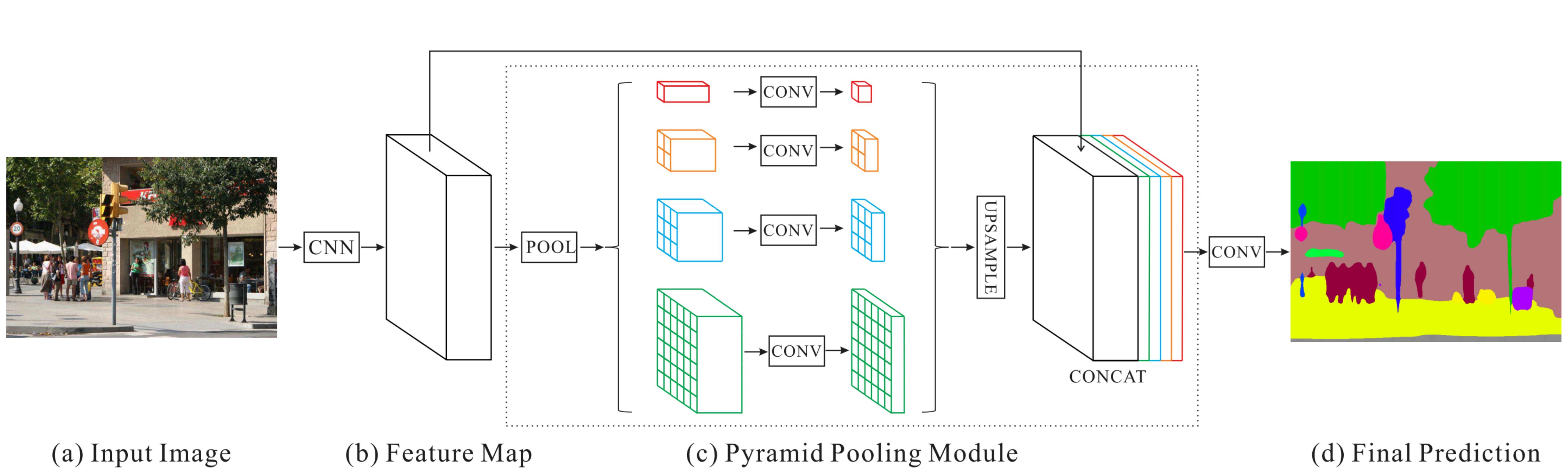


```
tf.keras.layers.Conv2D(  
    filters,  
    kernel_size,  
    strides=(1, 1),  
    padding="valid",  
    data_format=None,  
    dilation_rate=(1, 1), # 이 부분이 atrous 확장을 관리하는 부분이다.  
    ...  
)
```

# DeepLab

## 네트워크

- 인코더로 DeepLab V3 혹은 변형된 Xception 사용
  - PASCAL VOC 2012 데이터셋을 기준으로 ‘변형된 Xception’을 인코더로 쓸 때 성능이 좋았다고 한다.
- Atrous Convolution
  - 일반적인 인코더-디코더 구조에서는 불가능했던 인코더에서 추출된 특징맵의 해상도를 임의로 제어
- ASPP 모듈과 디코더 모듈에 Depth-wise Separable Convolution 을 적용
- PASCAL VOC 2012 와 Cityscapes dataset 에서 State-of-art를 달성

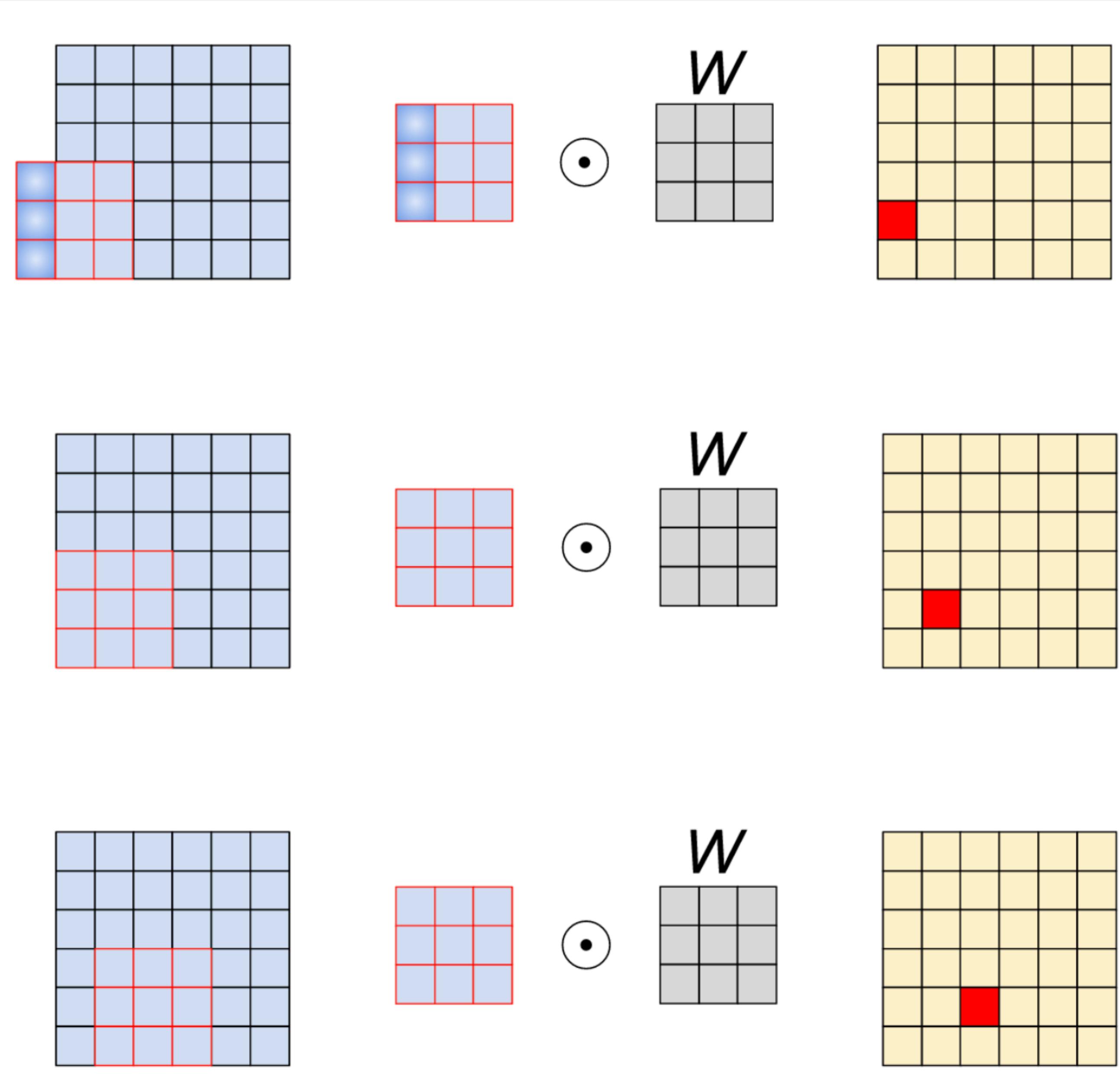


- ASPP : Atrous + Spatial Pyramid Pooling
  - DeepLab V3에서 소개된 Pooling
  - Pyramid Scene Parsing Network (PSPNetwork)에서 여러 해상도로 conv를 진행하고, 통합하는 모습. 인코더 부분에서 사용되었다.

## 개념 정리 - ASPP

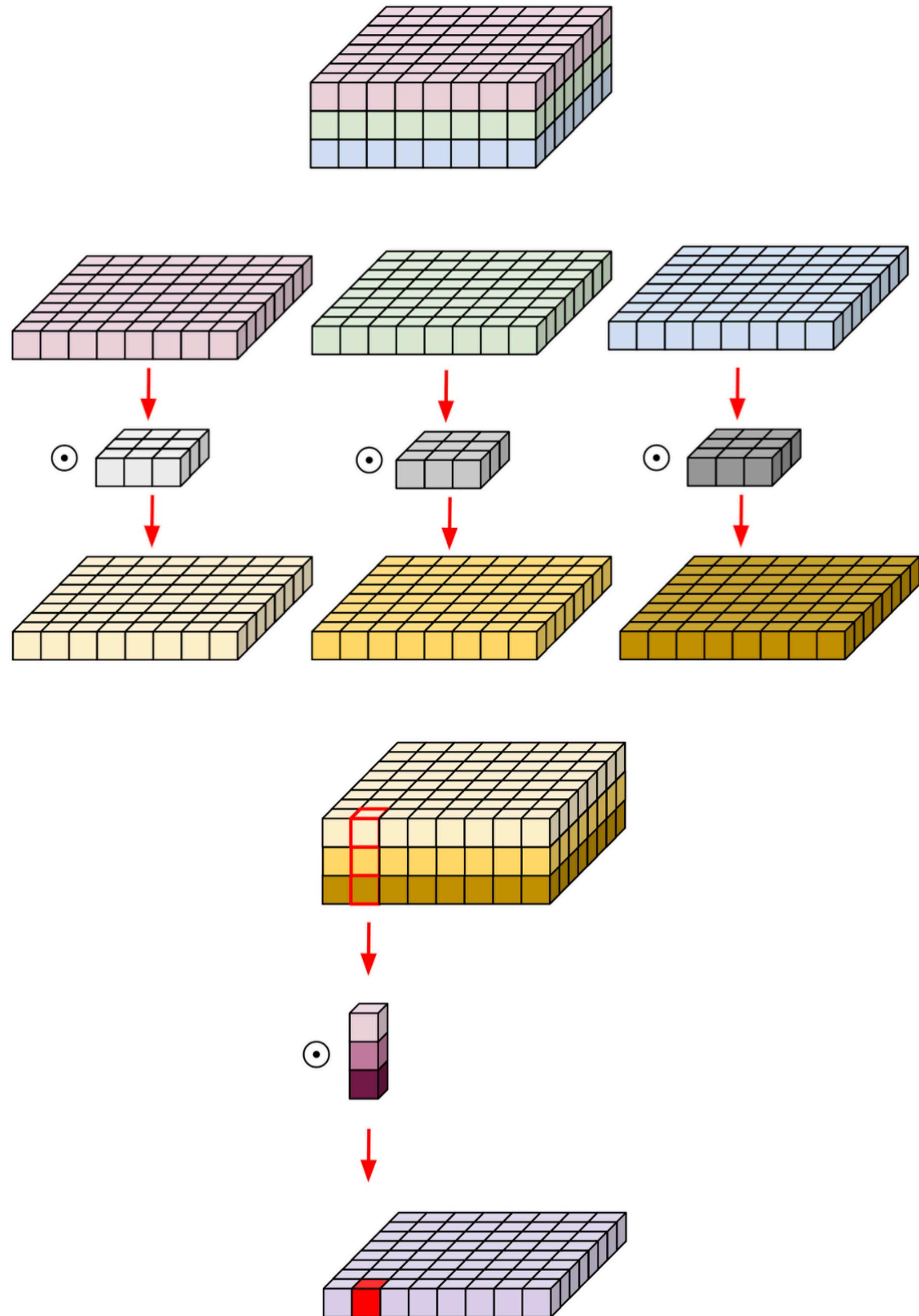
# 개념 정리 - Depthwise Separable Convolution

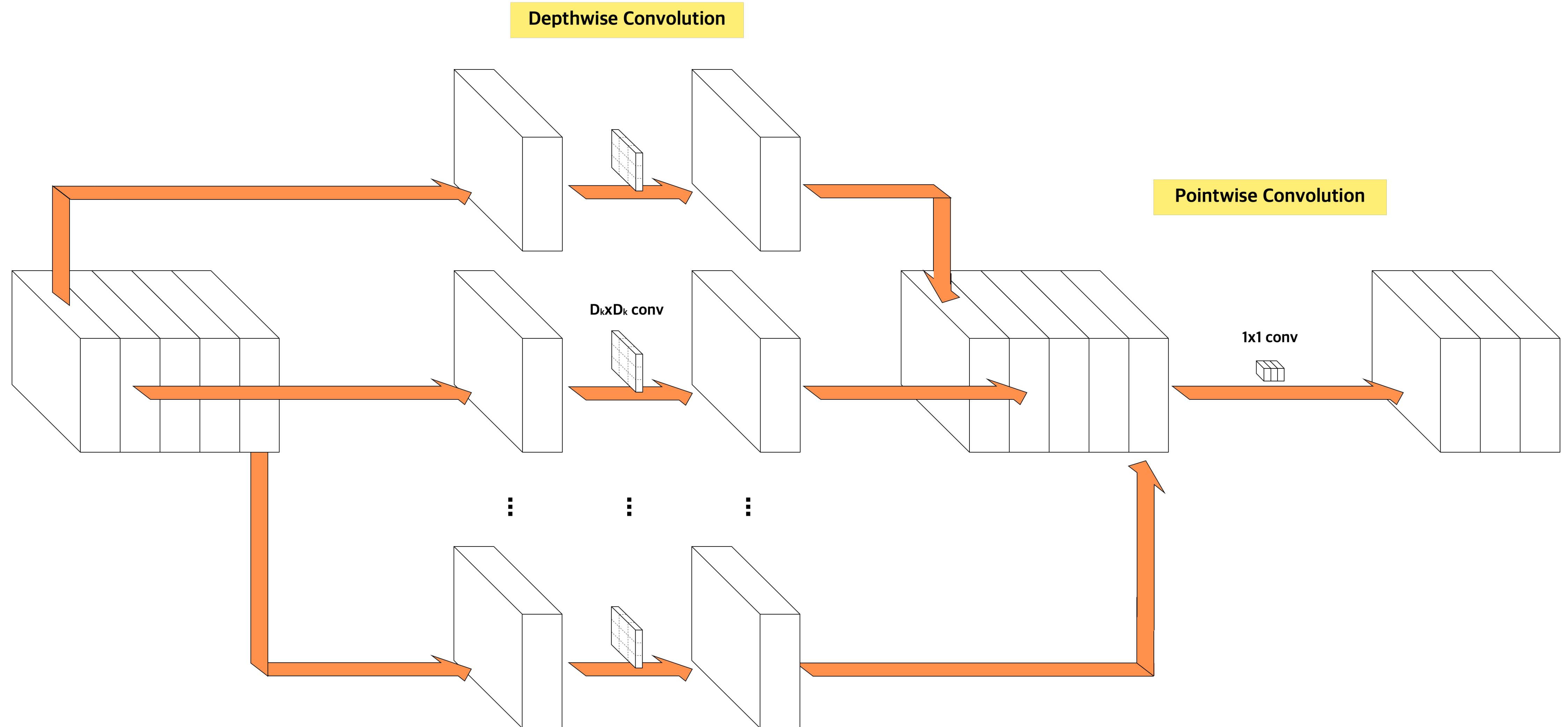
- 모델의 성능은 유지하면서 계산 비용과 parameter의 개수를 줄일 수 있는 방법
- Xception에서 사용
- 기존의 Convolution 이 두 개의 Convolution 으로 나눠져서 진행
  - Depth/Channel별로 convolution
  - Point 별로 convolution



# 개념 정리 - Depthwise Separable Convolution

- 3x3 2D 컨볼루션, 입력 채널 3, 출력 채널 16이고, 128의 가로, 세로
  - 기존의 컨볼루션
    - 파라미터 개수 :  $3 \times 3 \times 3 \times 16 = 432$ 
      - 파라미터 개수 = (필터 혹은 커널의 가로)\*(필터 혹은 커널의 세로)\*(입력 채널)\*(출력 채널)
    - 계산비용 :  $3 \times 3 \times 3 \times 16 \times 128 \times 128 = 7077888$ 
      - 계산비용 = (필터 혹은 커널의 가로)\*(필터 혹은 커널의 세로)\*(입력 채널)\*(출력 채널)\*(가로)\*(세로)
  - Depthwise Separable 컨볼루션
    - 파라미터 개수 :  $3 \times 3 \times 3 \times 1 + 1 \times 1 \times 3 \times 16 = 75$
    - 계산비용:  $3 \times 3 \times 3 \times 1 \times 128 \times 128 + 1 \times 1 \times 16 \times 128 \times 128 = 704512$

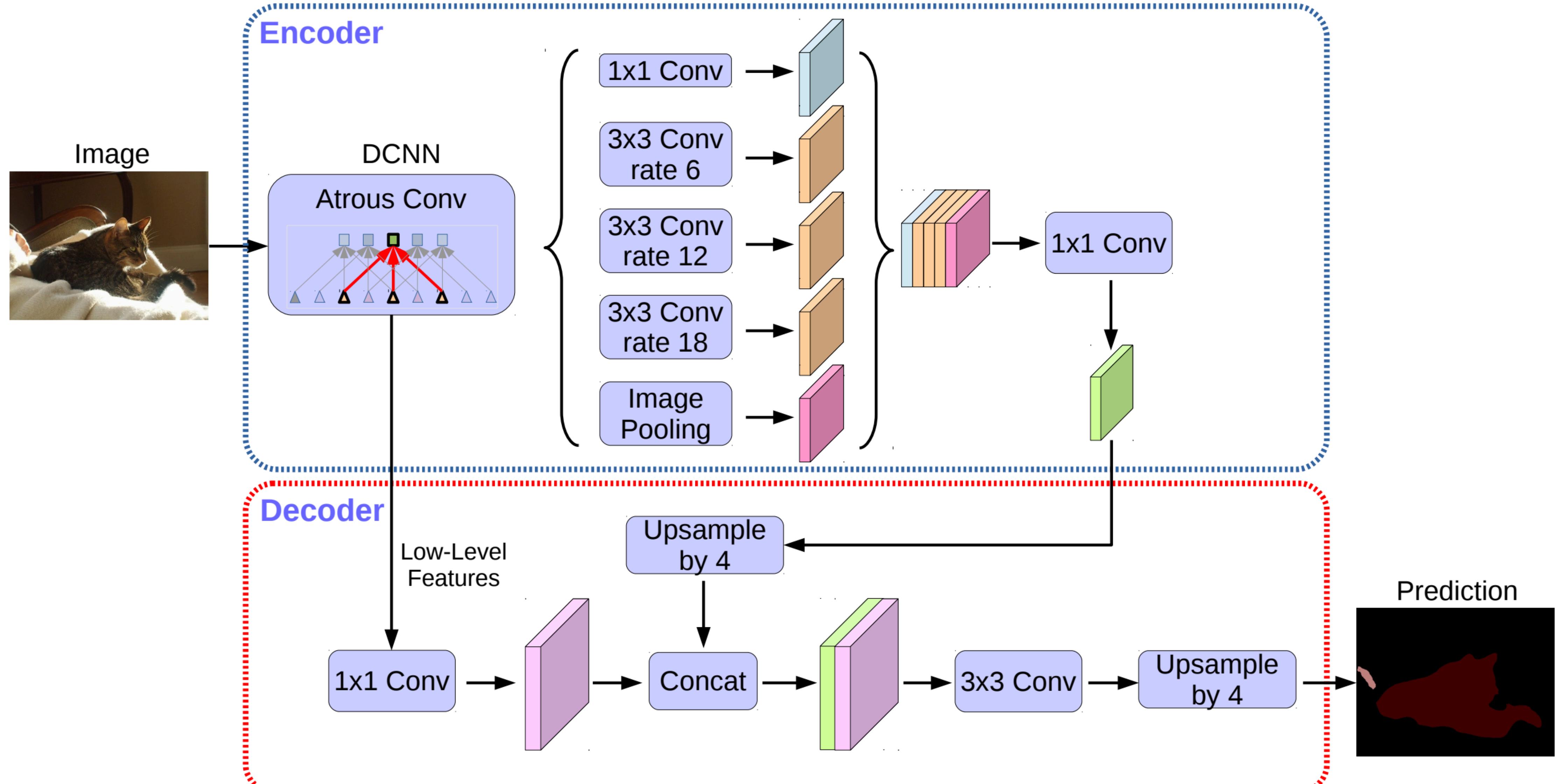




# DeepLab

## 개념 정리 - Output Stride

- 논문에서는 ‘Input 이미지 해상도’와 (Global Pooling을 하기 전) ‘Output 해상도’의 비율로 정의
- Strided Convolution 을 거치면 해상도가 점점 작아지면서 output stride 는 커짐
- 보통 Image Classification 의 마지막 특징맵은 input 보다 32배 정도 해상도가 줄어들지만, (이 경우 output stride: 32), Semantic Segmentation 더 빽빽한 특징맵이 필요하기 때문에 16 또는 8로 조절 (output stride가 작을수록 계산비용이 증가)
- 조절하는 방법으로는 마지막 block 에서 stride 를 없애거나, atrous convolution 으로 대체하는 등의 방법을 사용합니다.



모델 구조

- 인코더를 통해서 나온 특징맵은 원본 사진의 해상도보다 16배가 작다.
  - 효과적으로 이미지의 디테일을 회복하기 위해서 Decoding을 할 때 Low-Level Feature 를 합쳐서 진행
- 16으로 한 이유는 실험 결과 속도와 정확도의 좋은 trade-off point (여러 실험의 결과)
- Low-Level Feature 에 1x1 Conv 를 하는 이유는 인코더의 결과물과 채널의 줄이기 위해서
- 실험을 통해서 채널의 개수가 48개 일 때 가장 좋은 결과를 내어서 Low-Level Feature 의 채널수에 48을 사용

Encoder train OS	Decoder eval OS	MS COCO JFT					mIOU	Multiply-Add
		Flip	SC	COCO	JFT			
16	16						79.17%	68.00B
16	16	✓					80.57%	601.74B
16	16	✓	✓				80.79%	1203.34B
16	8						79.64%	240.85B
16	8	✓					81.15%	2149.91B
16	8	✓	✓				81.34%	4299.68B
16	16	✓					79.93%	89.76B
16	16	✓	✓				81.38%	790.12B
16	16	✓	✓	✓			81.44%	1580.10B
16	8	✓					80.22%	262.59B
16	8	✓	✓				81.60%	2338.15B
16	8	✓	✓	✓			81.63%	4676.16B
16	16	✓		✓			79.79%	54.17B
16	16	✓	✓	✓	✓		81.21%	928.81B
16	8	✓		✓			80.02%	177.10B
16	8	✓	✓	✓	✓		81.39%	3055.35B
16	16	✓		✓	✓	✓	82.20%	54.17B
16	16	✓	✓	✓	✓	✓	83.34%	928.81B
16	8	✓		✓	✓	✓	82.45%	177.10B
16	8	✓	✓	✓	✓	✓	83.58%	3055.35B
16	16	✓		✓	✓	✓	83.03%	54.17B
16	16	✓	✓	✓	✓	✓	84.22%	928.81B
16	8	✓		✓	✓	✓	83.39%	177.10B
16	8	✓	✓	✓	✓	✓	84.56%	3055.35B

# DeepLab V3를 인코더로 사용한 모델 - Residual

- DeepLab V3는 기본적으로 ResNet을 Backbone으로 사용
- 최종 배워야하는 것을  $H(x)$ , Stacking된 Layer의 Output을  $F(x)$ , 그리고 Input을  $x$ 라고 하면,
  - $H(x) = F(x) + x$
- 형태로 Block을 만든 것이다.
- 즉 Skip Connection이 없었던 것에 비해서  $F(x)$ 는 Input과의 '차이'만을 학습하면 되는 것이므로 이 때문에 Residual learning이라고 부른다.
  - $F(x) = H(x) - x$
- 논문에서는 이러한 방식이 기존의 방식에 비해 학습이 쉬울 것이라고 한다.

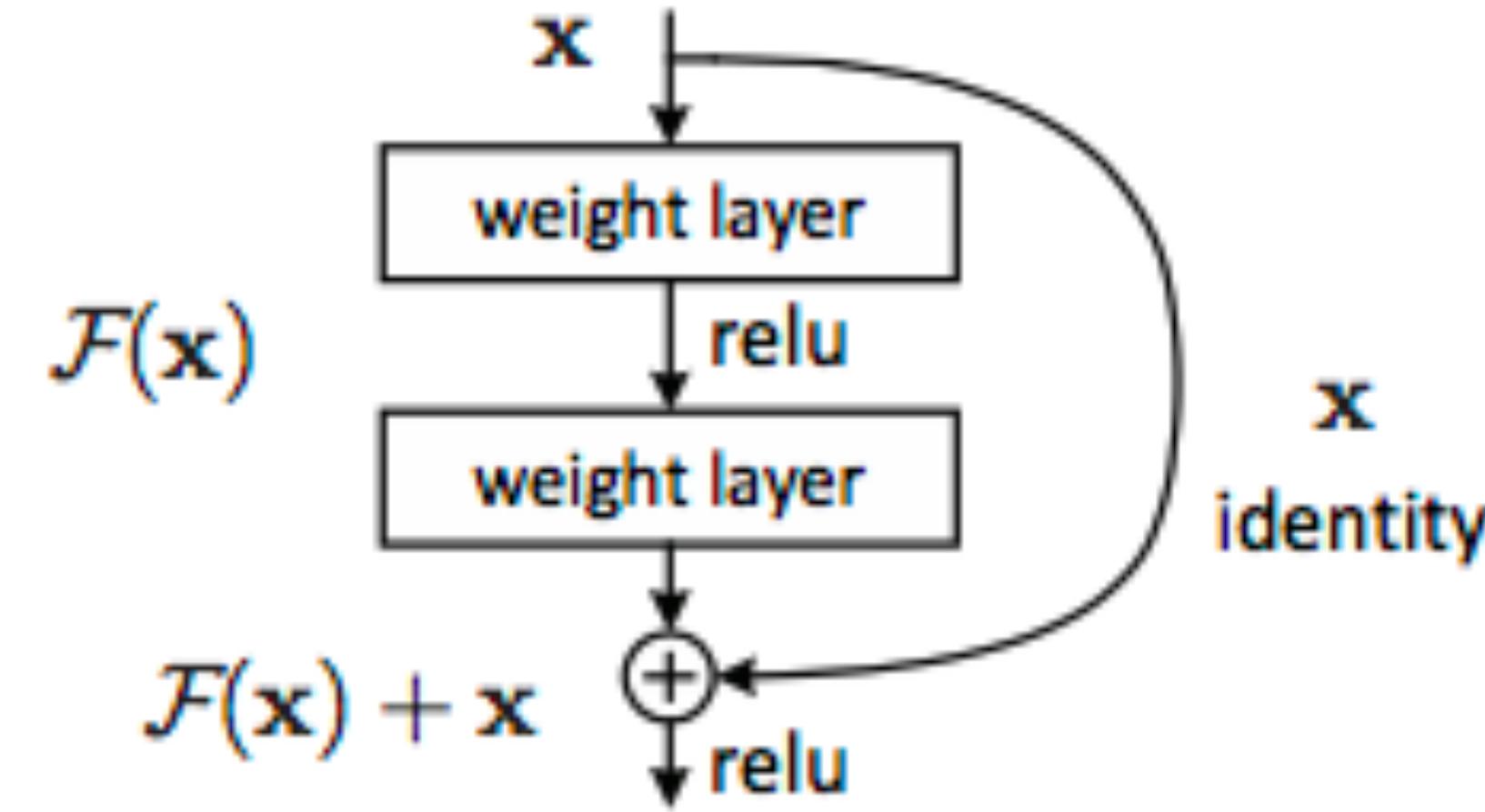
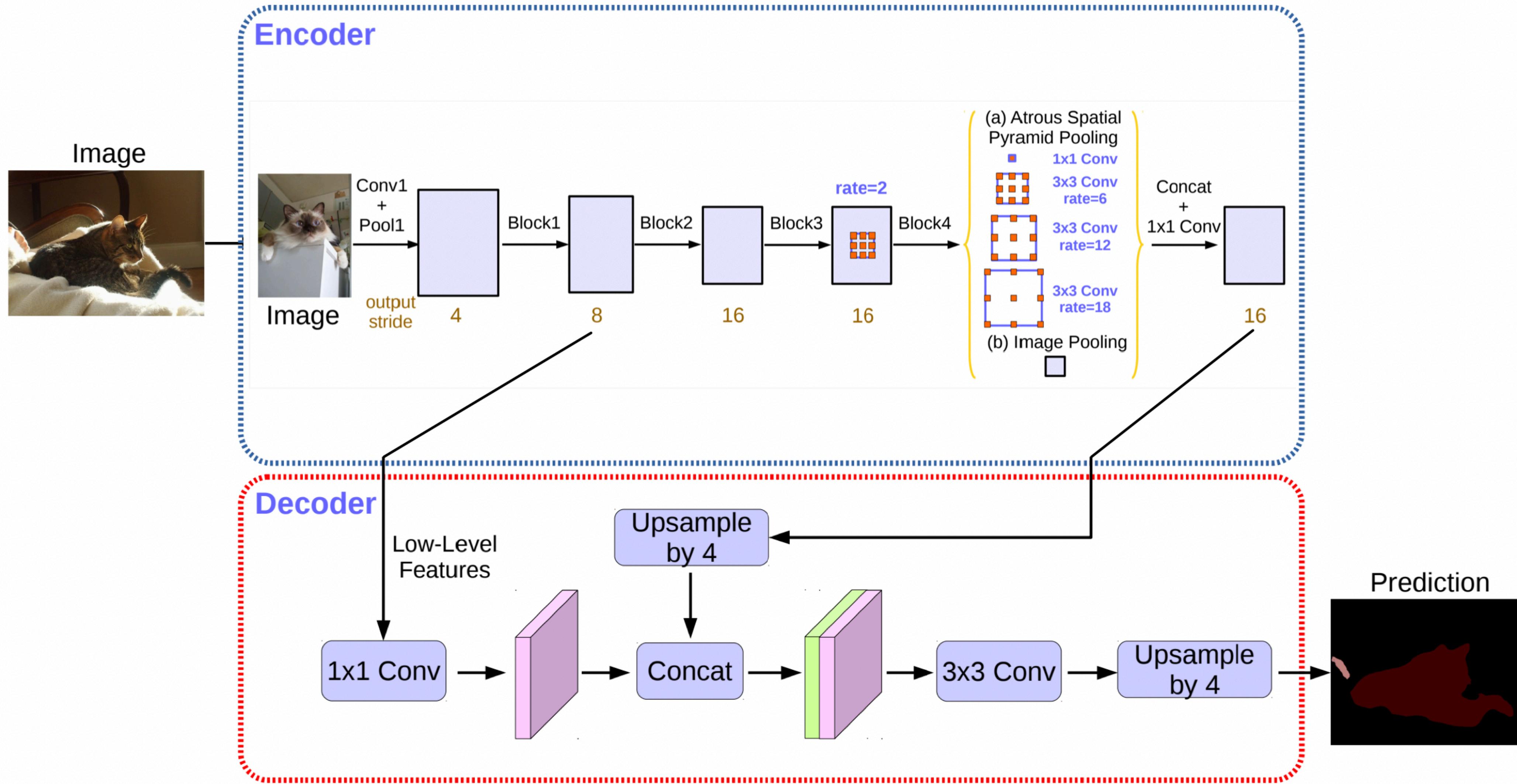
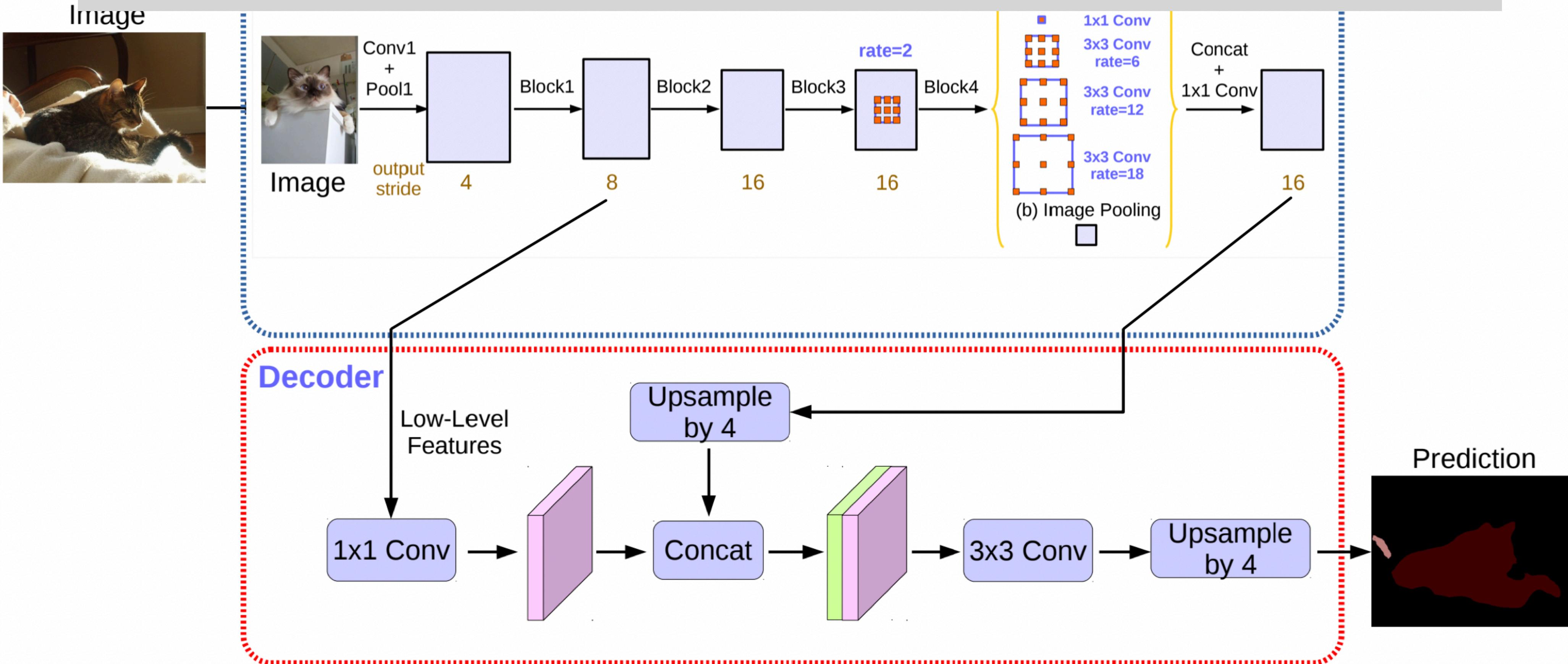


Figure 2. Residual learning: a building block.



DeepLab V3를 인코더로 사용한 모델

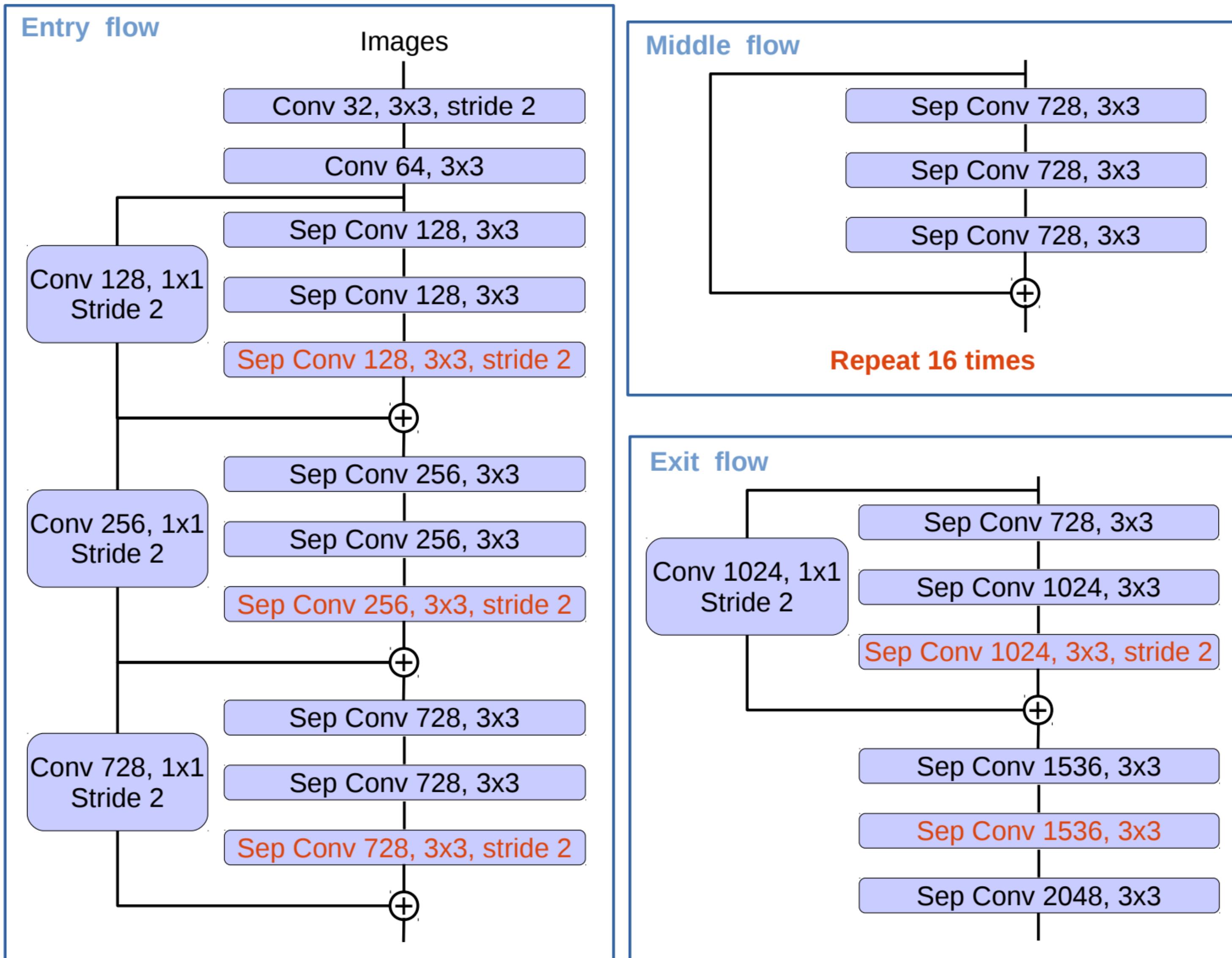
- Stride를 하기 전 Conv2 특징맵을 Low-Level 특징맵으로 사용
- 인코더의 결과물은 256개의 채널

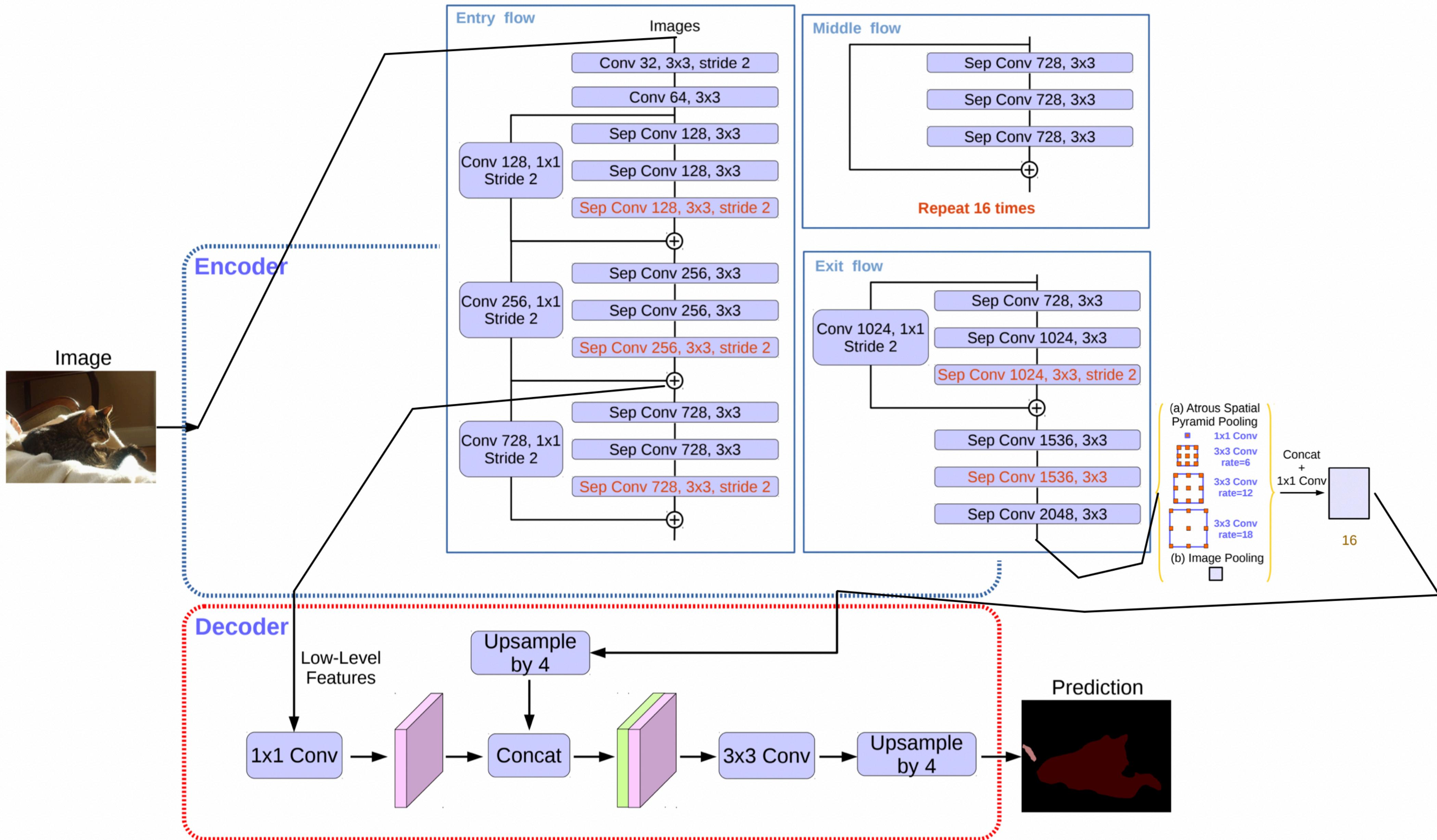


DeepLab V3를 인코더로 사용한 모델

# 변형된 Xception을 인코더로 사용한 모델

- 더 많은 레이어 (Entry flow에서의 변화를 제외하고는 MSRA의 수정과 같다)
- 모든 max pooling은 stride가 있는 depthwise separable convolutions로 변형
- batch normalization 및 ReLU가 각각의  $3 \times 3$  depthwise에 추가됨. MobileNet과 유사

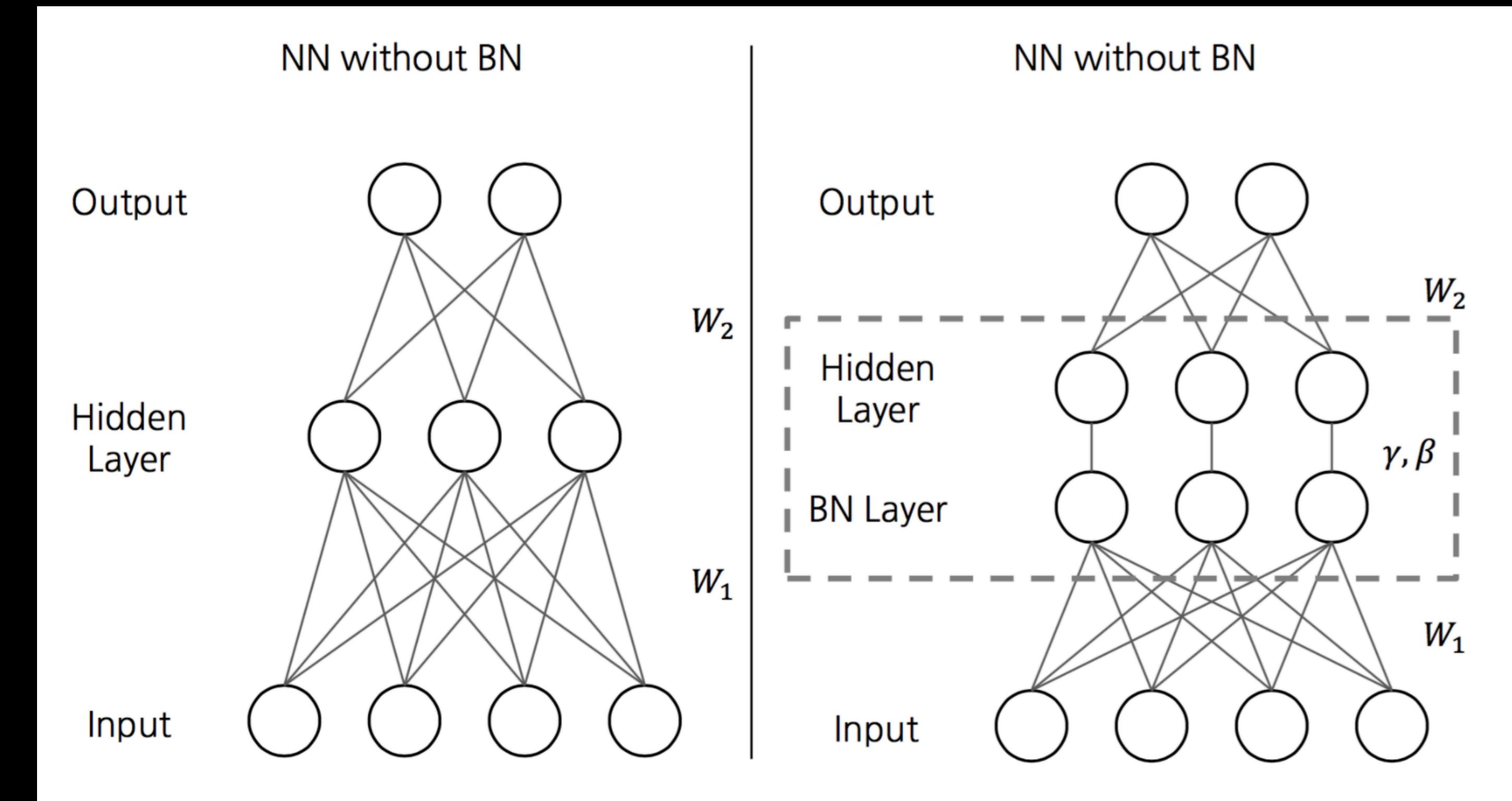




변형된 Xception 을 인코더로 사용한 모델

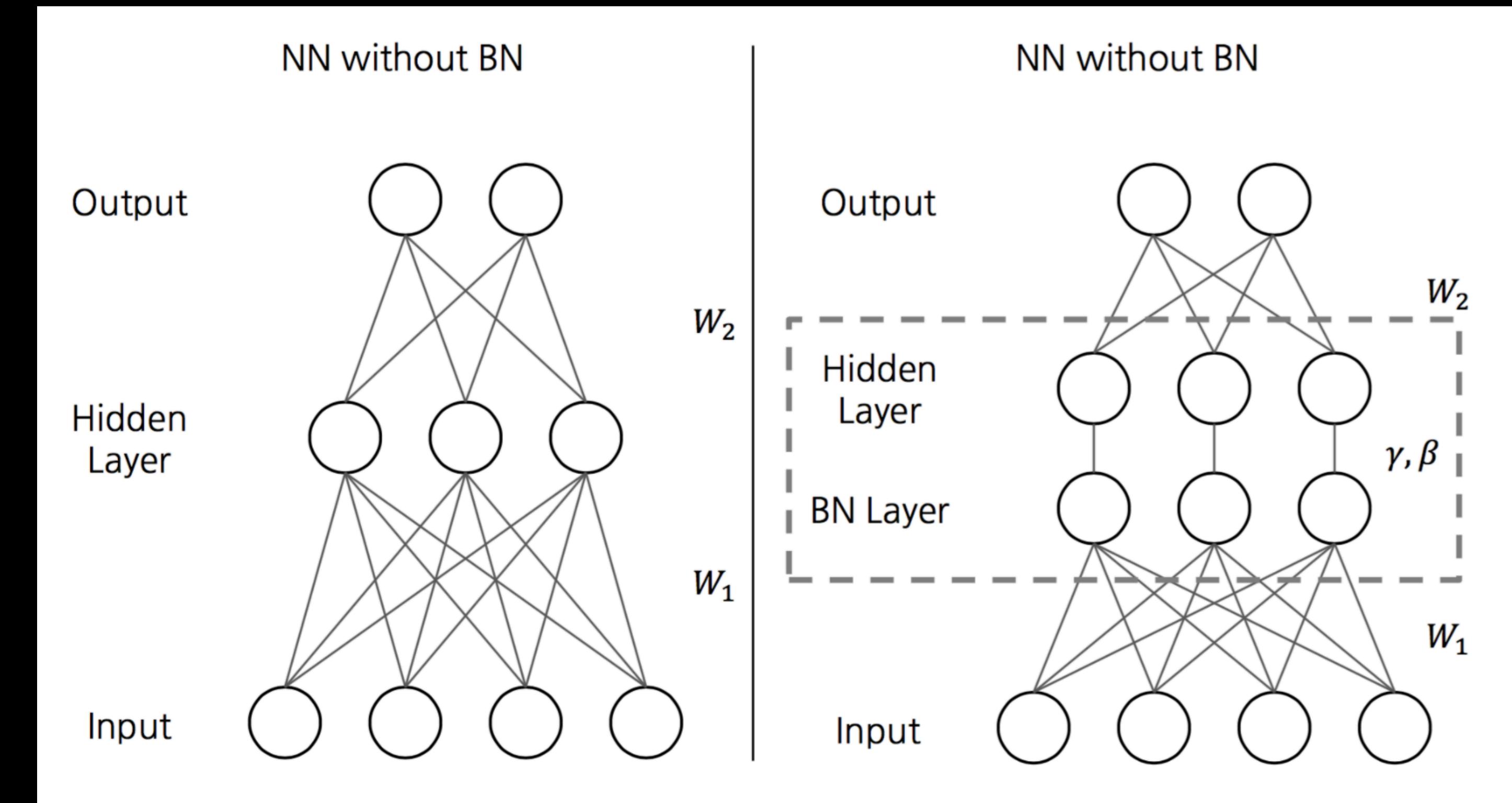
# 기타 개념 - 배치정규화

- 활성화함수의 활성화값 또는 출력값을 정규화(정규분포로 만든다)하는 작업
- 신경망의 각 layer에서 데이터(배치)의 분포를 정규화하는 작업
- 학습을 할 때마다 활성화값/출력값을 정규화하기 때문에 초기화(가중치 초기값) 문제에서 비교적 자유로워진다.



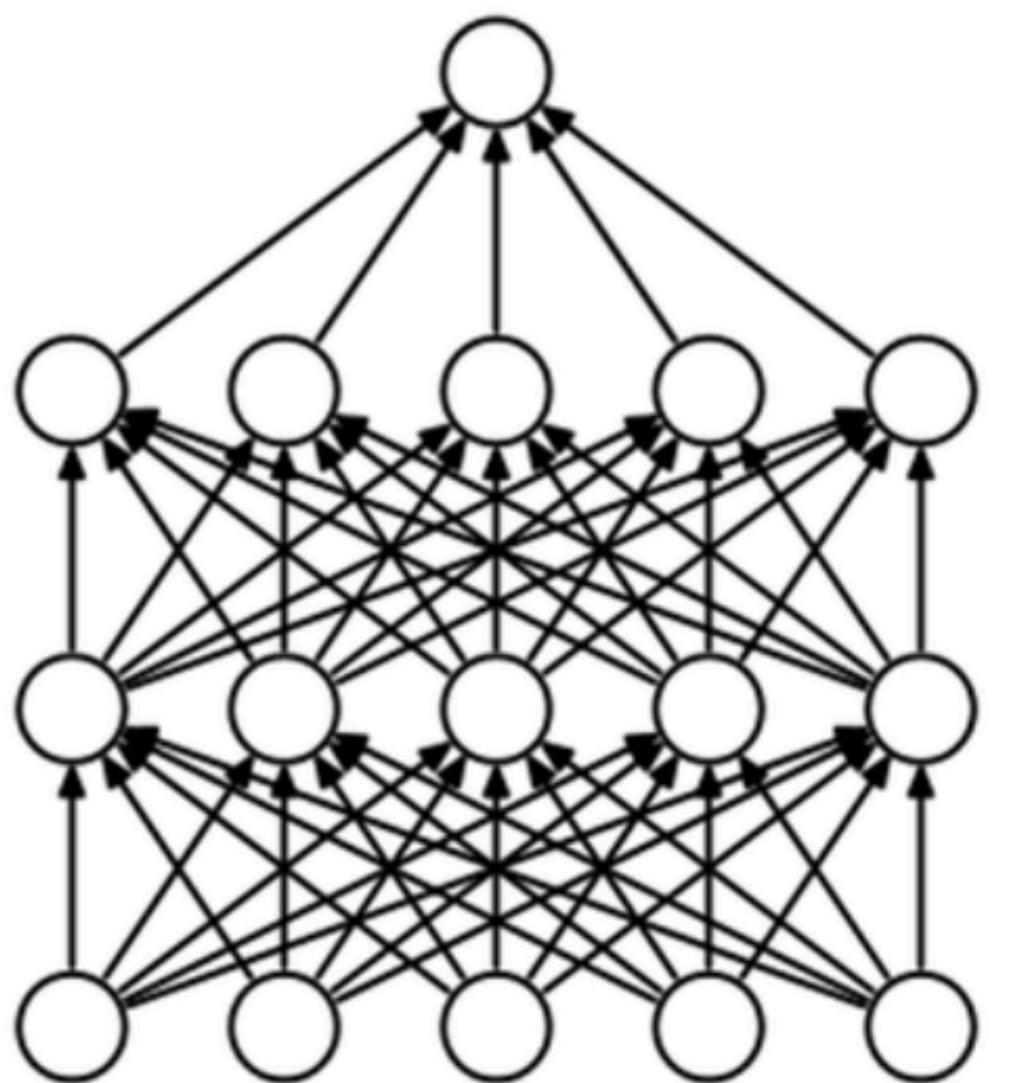
# 기타 개념 - 배치정규화

- 학습을 할 때 hidden layer의 중간에서 입력분포가 학습할 때마다 변화하면서 가중치가 엉뚱한 방향으로 갱신될 문제가 발생할 수 있다.
- 신경망의 층이 깊어질수록 학습 시에 가정했던 입력분포가 변화하여 엉뚱한 학습이 될 수 있다.
- training 과정 자체에서 학습을 안정화시키고 (vanishing, exploding 해결) 학습속도를 개선하는 방법

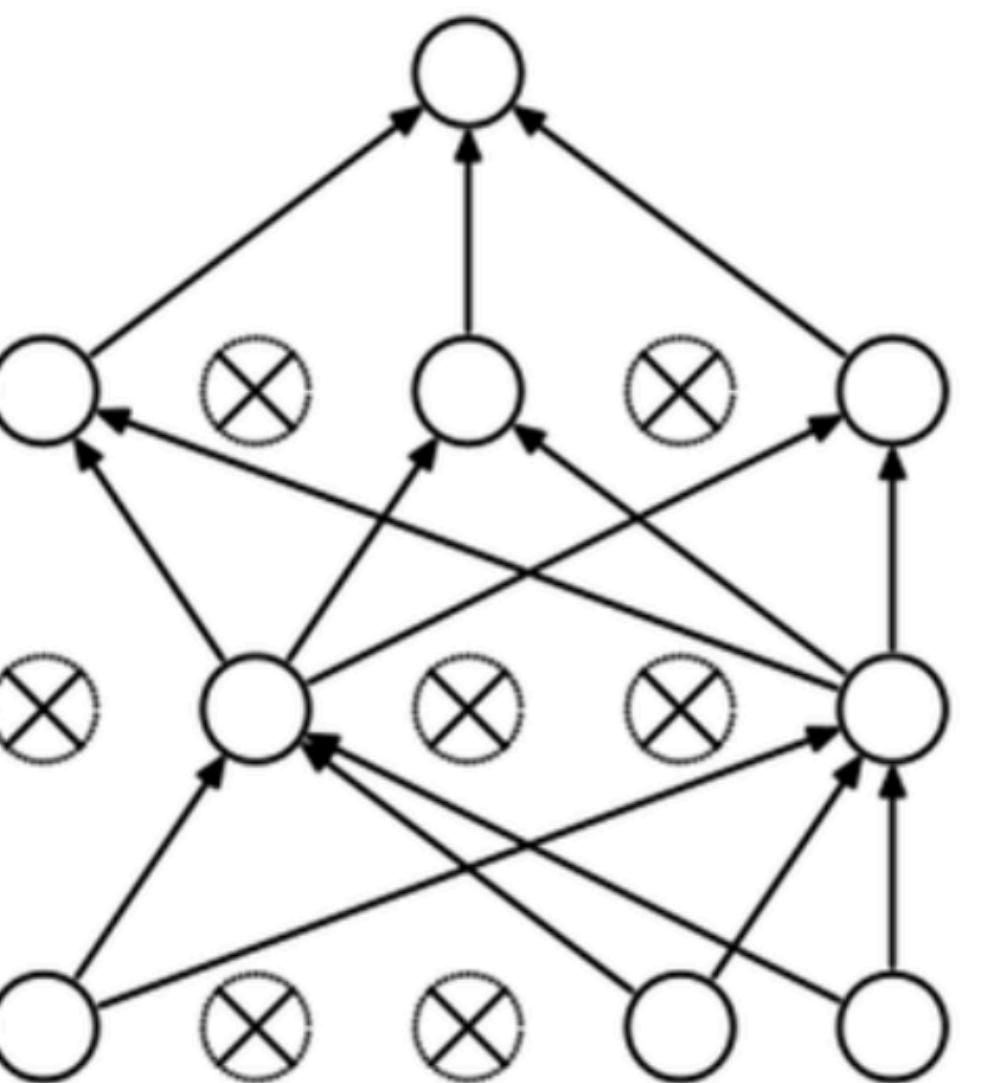


# 기타 개념

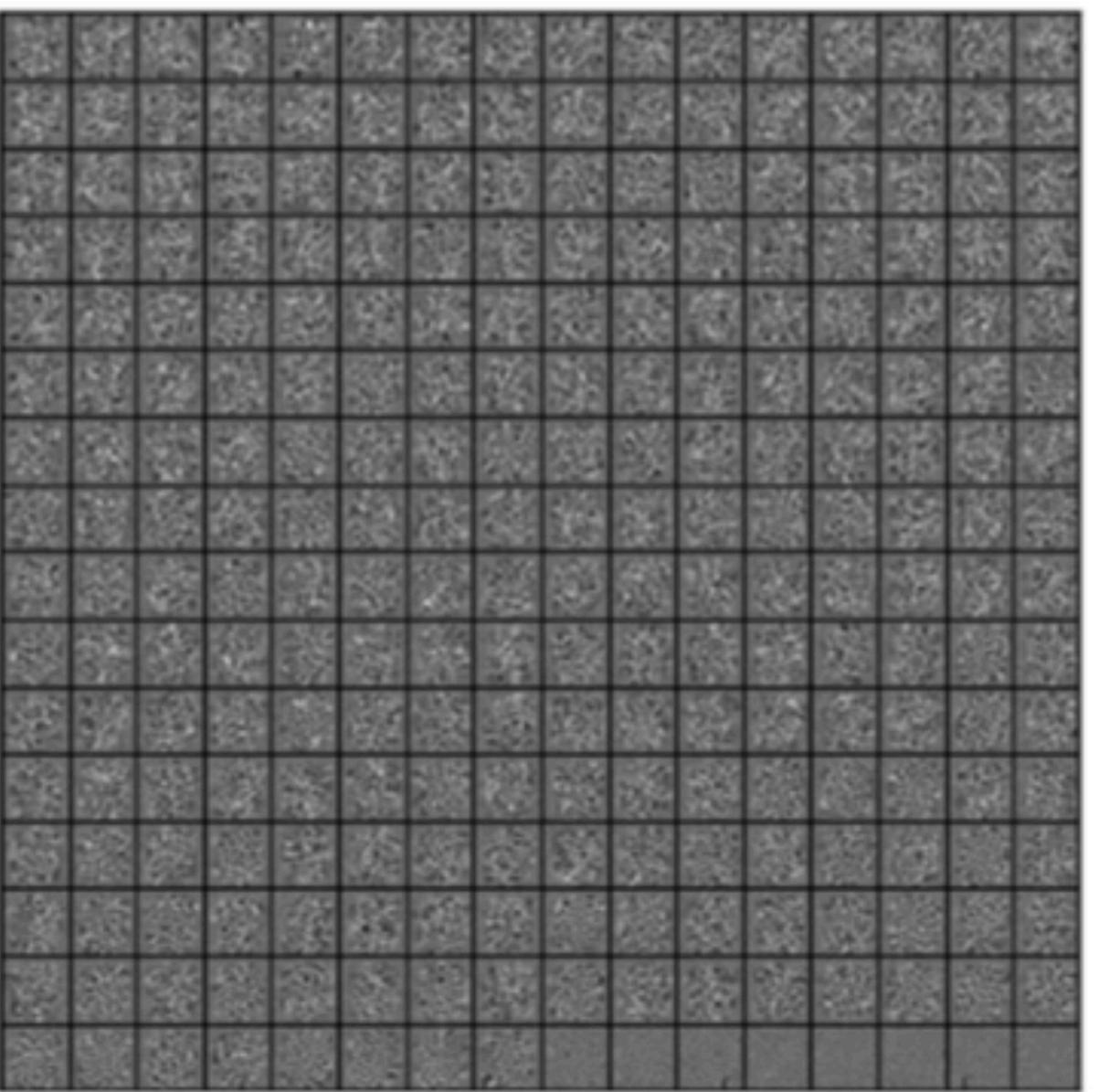
## Dropout



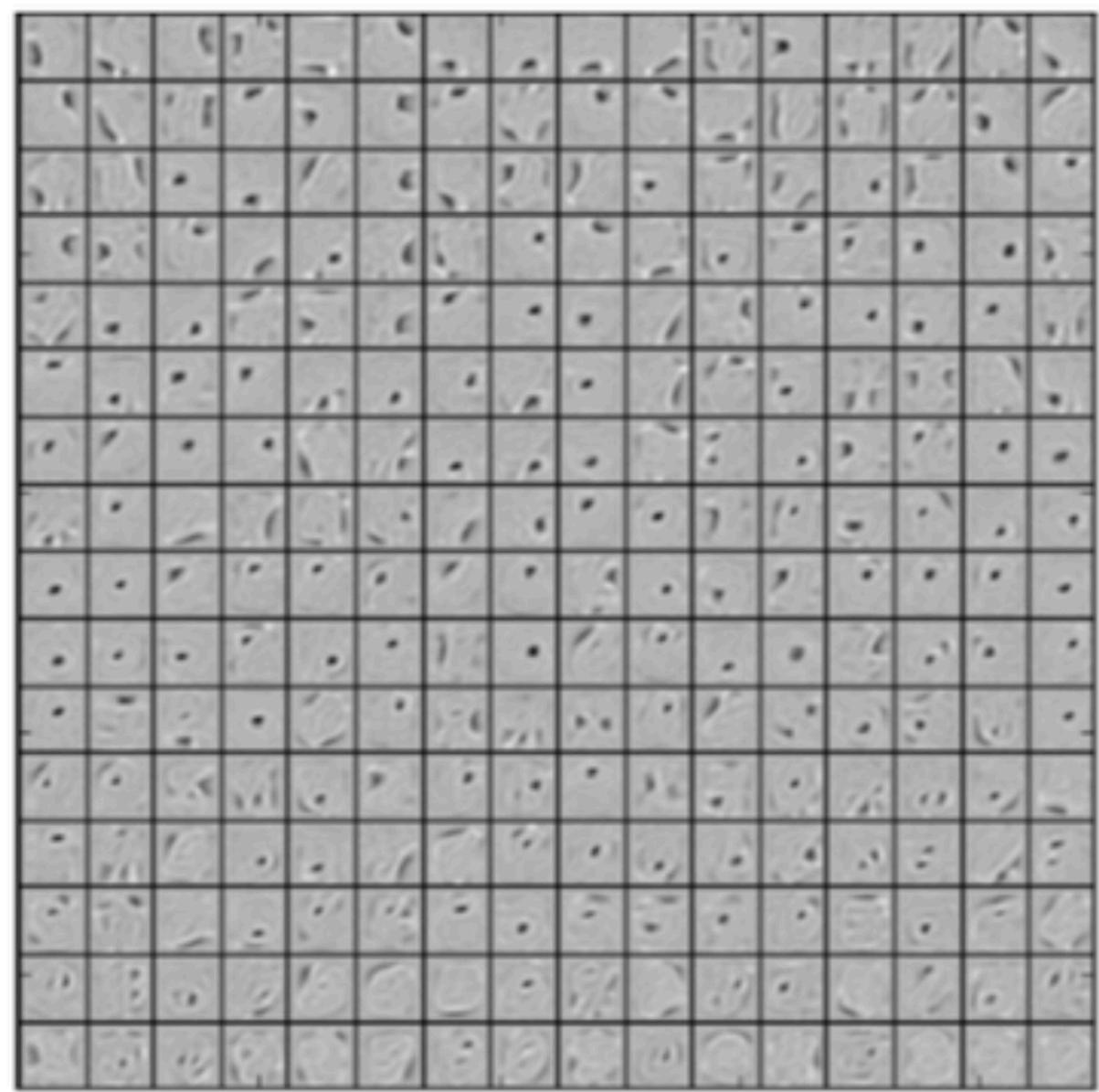
(a) Standard Neural Net



(b) After applying dropout.



(a) Without dropout

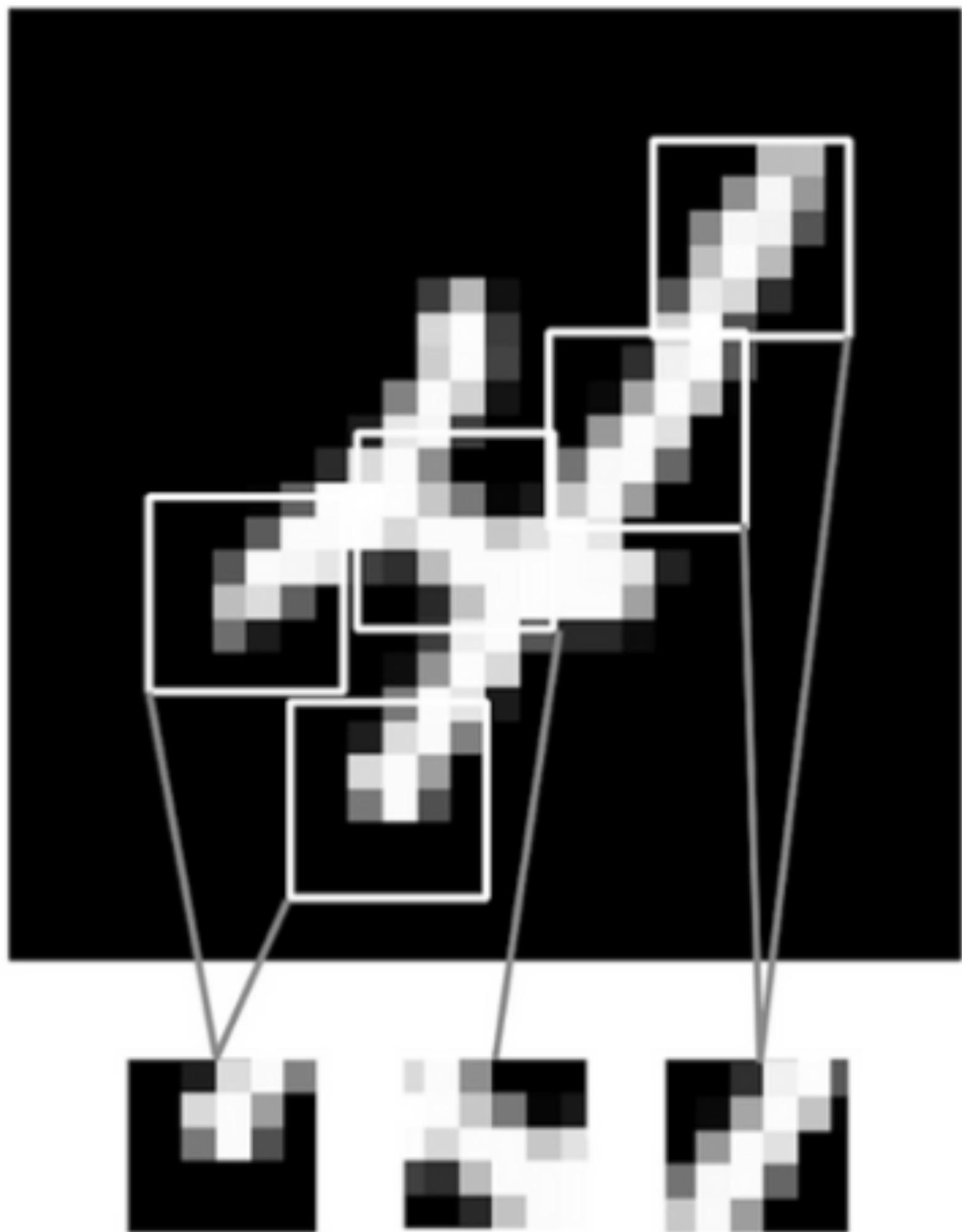


(b) Dropout with  $p = 0.5$ .

- ZFNet의 특성 시각화.

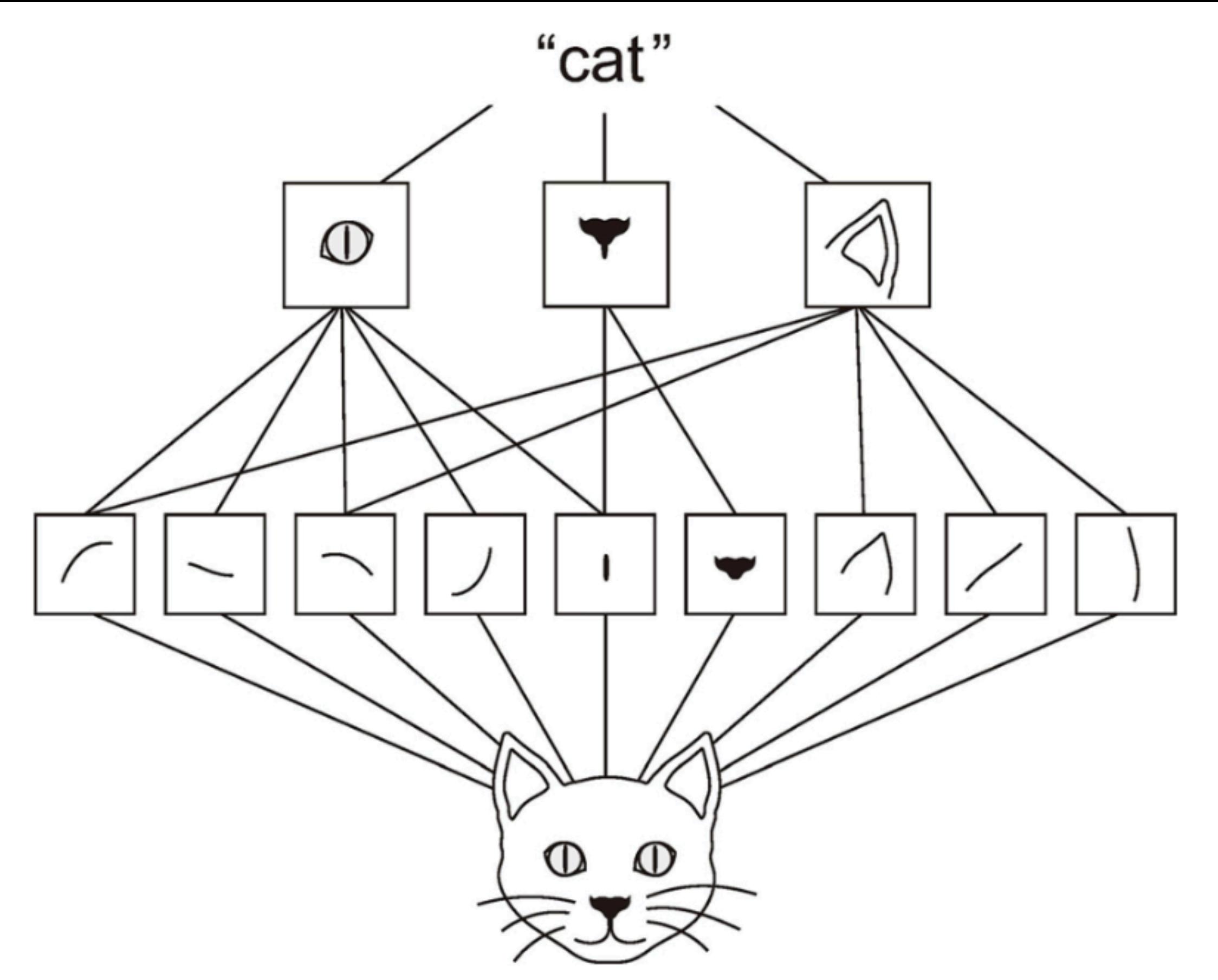
# Conv

- 합성곱은 지역 패턴을 학습

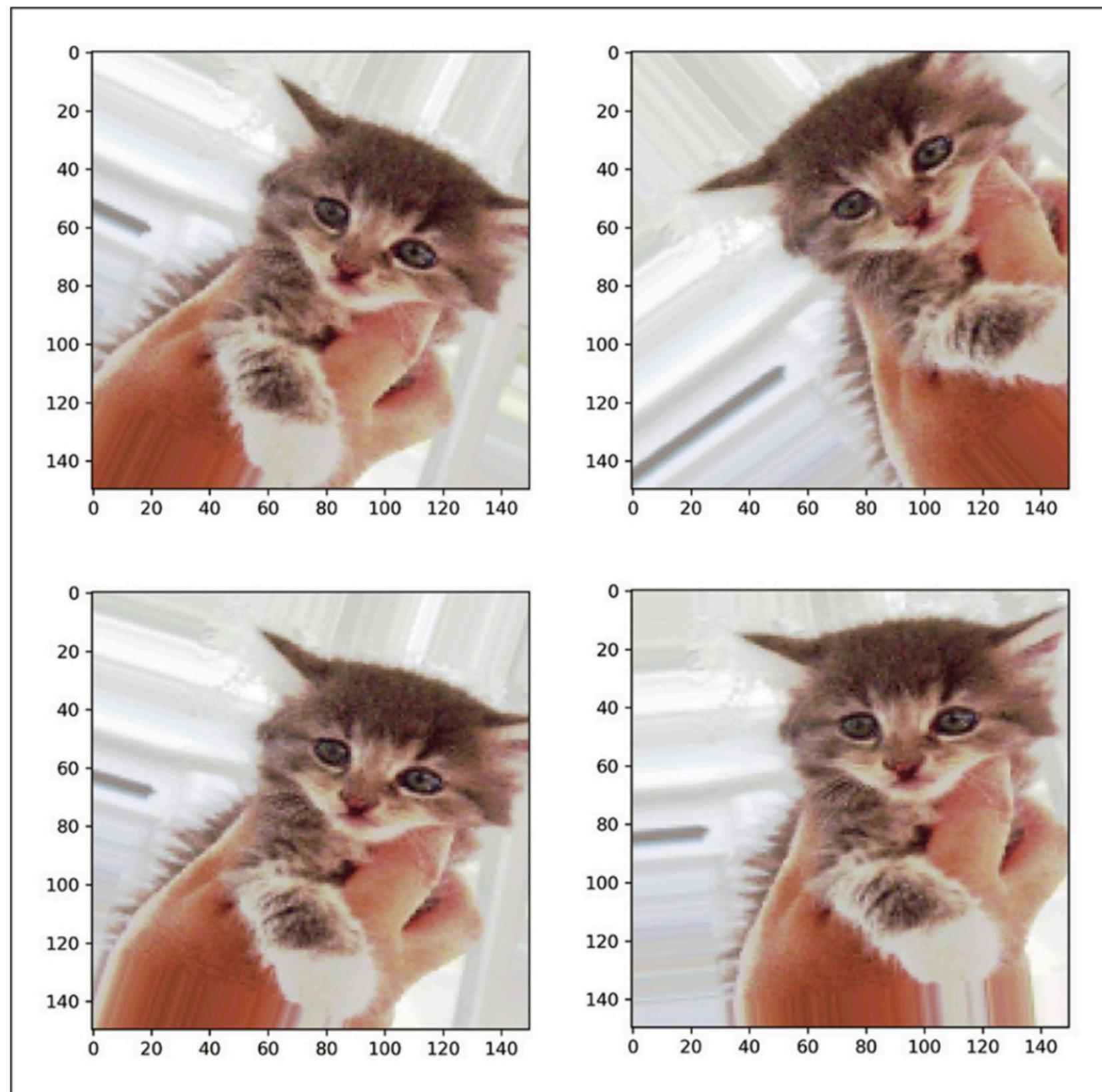


# Conv

- 컨브넷은 패턴의 공간적 계층 구조를 학습



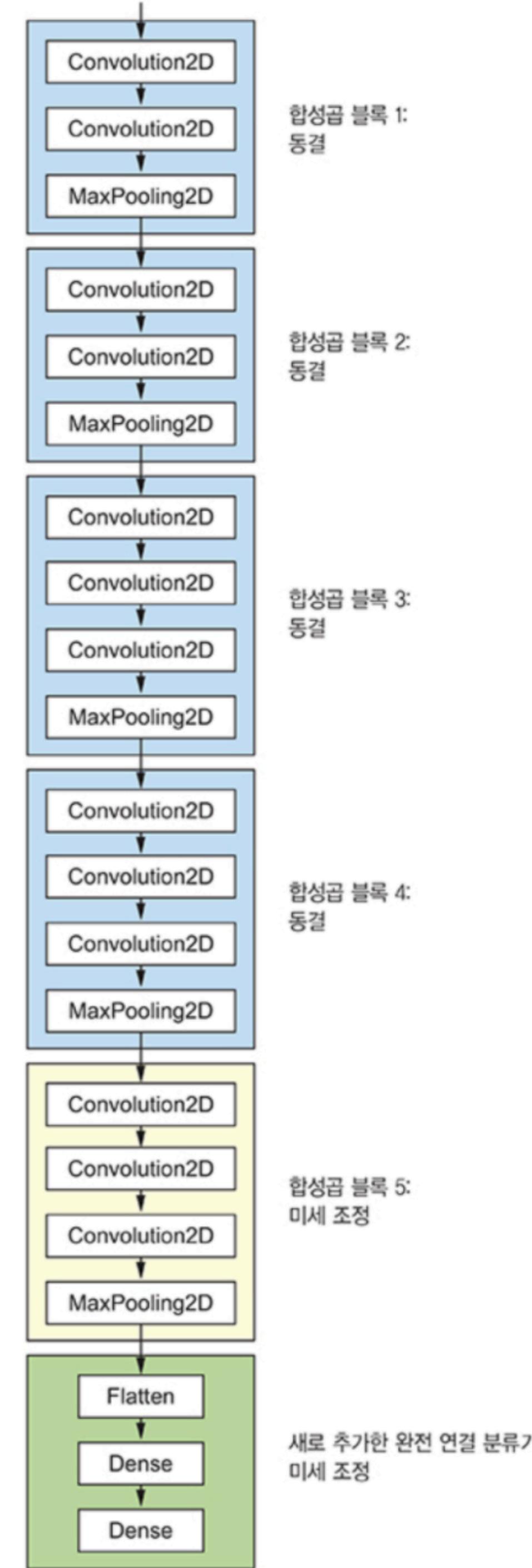
# 데이터 증식



```
datagen = ImageDataGenerator(  
    rotation_range=20,      # 랜덤하게 이미지 회전시킬 각도 -rotation_range ~ +rotation_range  
    width_shift_range=0.1,   # 이미지를 수평, 수직으로 랜덤하게 평행 이동 시킬 범위  
    height_shift_range=0.1,  # 1보다 큰 실수나 정수이면, 픽셀 값  
    shear_range=0.1,        # 랜덤하게 전단(shearing) 변환을 적용할 각도 범위  
    zoom_range=0.1,         # 랜덤하게 이미지를 확대할 범위 1-zoom_range ~ 1+zoom_range  
    horizontal_flip=True,   # 랜덤으로 이미지를 수평으로 뒤집는다. 수평 대칭을 가정할 수 있을 때 사용한다.  
    fill_mode='nearest')     # 회전, 이동으로 새롭게 생성해야 할 픽셀을 채울 전략 'nearest',  
    'constant', 'reflect', 'wrap'
```

```
from keras.preprocessing import image  
  
fnames = sorted([os.path.join(train_cats_dir, fname) for fname in os.listdir(train_cats_dir)])  
  
img_path = fnames[3]      # 증식할 이미지를 선택  
  
img = image.load_img(img_path, target_size=(150, 150))      # 이미지를 읽고 크기 변경  
  
x = image.img_to_array(img)      # (150, 150, 3) 크기의 넘파이 배열로 변환  
x = x.reshape((1,) + x.shape)    # (1, 150, 150, 3) 크기로 변환  
                                # flow() 메서드가 배치 데이터를 기대하기 때문에 샘플 데이터에 배치 차원을 추가하여 4D 텐서로 만든다.  
  
# 랜덤하게 변환된 이미지를 생성  
# 무한반복 되기 때문에 어느 시점에서 중지해야 한다.  
i = 0  
for batch in datagen.flow(x, batch_size=1):  
    plt.figure(i)  
    imgplot = plt.imshow(image.array_to_img(batch[0]))  
    i += 1  
    if i % 4 == 0:  
        break  
  
plt.show()
```

# 미세 조정



```
conv_base.trainable = True  
  
set_trainable = False  
for layer in conv_base.layers:  
    if layer.name == 'block5_conv1':  
        set_trainable = True  
    if set_trainable:  
        layer.trainable = True  
    else:  
        layer.trainable = False  
  
  
model.compile(loss='binary_crossentropy',  
              optimizer=optimizers.RMSprop(lr=1e-5),  
              metrics=['acc'])  
  
history = model.fit_generator(  
    train_generator,  
    steps_per_epoch=100,  
    epochs=100,  
    validation_data=validation_generator,  
    validation_steps=50)
```