Edge Detection

Canny Edge Detection & Richer Convolutional Features

Sogang University
Vision & Display Systems Lab, Dept. of Electronic Engineering

Presented By
임나현

Contents

- Introduction
- Canny Edge Detection
- Boundary Tracing
- Deep Learning
- Conclusion

Introduction

- Edge Detection

  Computer Vision에서 edge detection은 객체 인식, 자율 주행 등의 다양한 응용 분야에서 활용됨

  - Canny Edge Detection

    가장 널리 사용되는 edge detection 알고리즘으로, 연속적이고 명확한 경계선을 추출하기 위해 설계됨

    1) Gaussian Blurring

    2) Sobel Mask

    3) Non-Maximum Suppression

    4) Hysteresis Thresholding

- Rich Convolutional Features

  CNN의 다중 계층에서 추출한 특징을 통합하여 경계 검출의 정확성과 효율성을 향상시킨 기법

Introduction

- Input Image

Canny Edge Detection

- **Gaussian Blurring** ⇨ Sobel Mask ⇨ Non-Maximum Suppression ⇨ Hysteresis Thresholding

목적 : noise 제거 -> 부드러운 edge 와 edge detection 정확도 향상

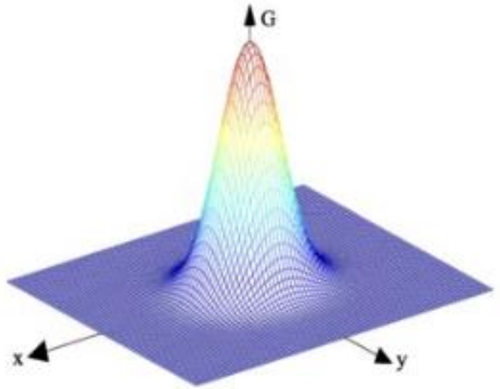
특징 : gaussian 분포 사용

중심 픽셀 - 높은 가중치, 주변 픽셀 - 낮은 가중치

filter size : truncate = 4.0

-> 평균(μ)에서 4σ(표준편차) 범위까지 커버

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$



(a) 3-D plot of the Gaussian function G.

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

$\times \frac{1}{273}$

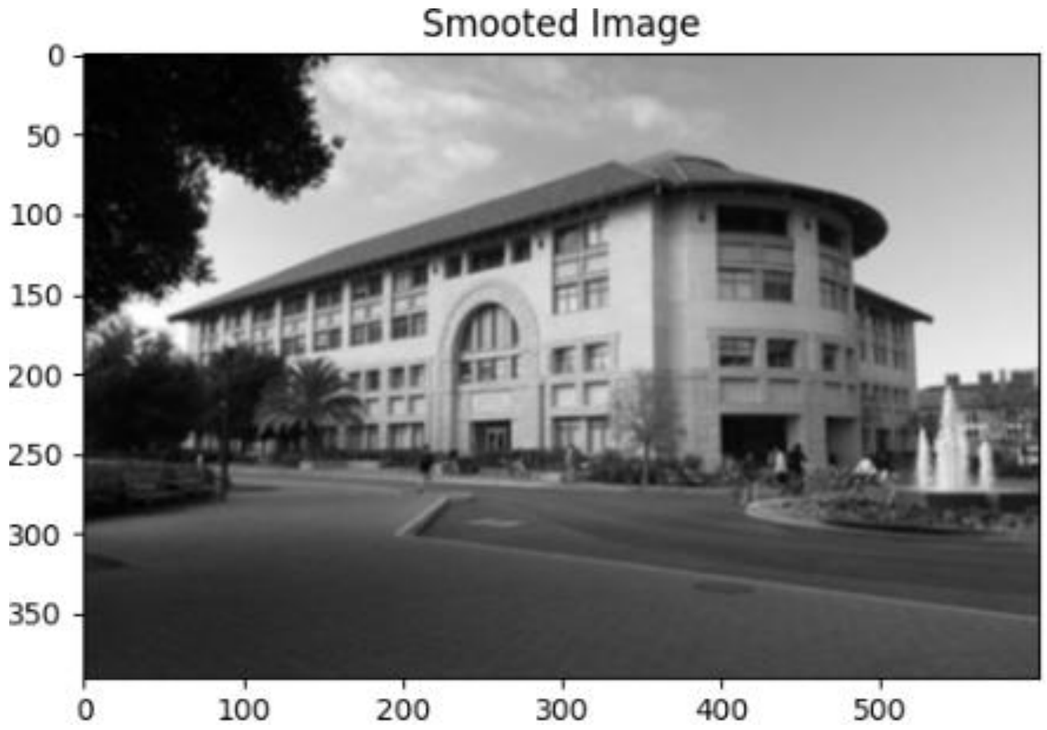
(b) A 5×5 Gaussian mask.

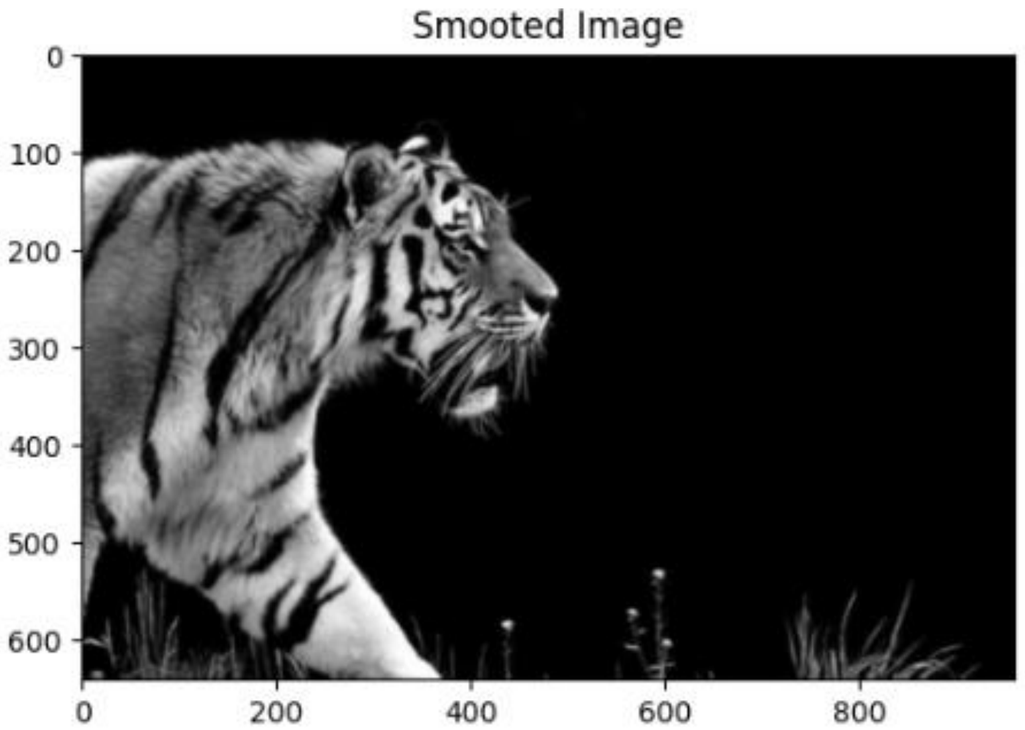
```
def gaussian(sigma):
  truncate = 4.0 #기본값
  kernel_size = 2*int(truncate * sigma) +1
  x, y = np.mgrid[-kernel_size//2:kernel_size//2, -kernel_size//2:kernel_size//2]
  gaussian = (1 / (2 * np.pi * sigma**2)) * np.exp(-(x**2 + y**2) / (2 * sigma**2)
  return gaussian / gaussian.sum()
```

Canny Edge Detection

- Gaussian Blurring

  방식 : 9x9 Gaussian mask (sigma = 1.0, truncate = 4.0)





Canny Edge Detection

- Gaussian Blurring ⇨ **Sobel Mask** ⇨ Non-Maximum Suppression ⇨ Hysteresis Thresholding

edge : 화소의 강도가 순간적으로 변하는 지점 -> 픽셀 값의 변화율을 계산하여 변화율이 큰 픽셀을 선택

순서 : 1) x축 방향, y축 방향의 편미분을 계산하여 gradient을 얻음

3x3 mask

-1	0	1
-2	0	2
-1	0	1

-1	-2	-1
0	0	0
1	2	1

2) mask를 적용할 때

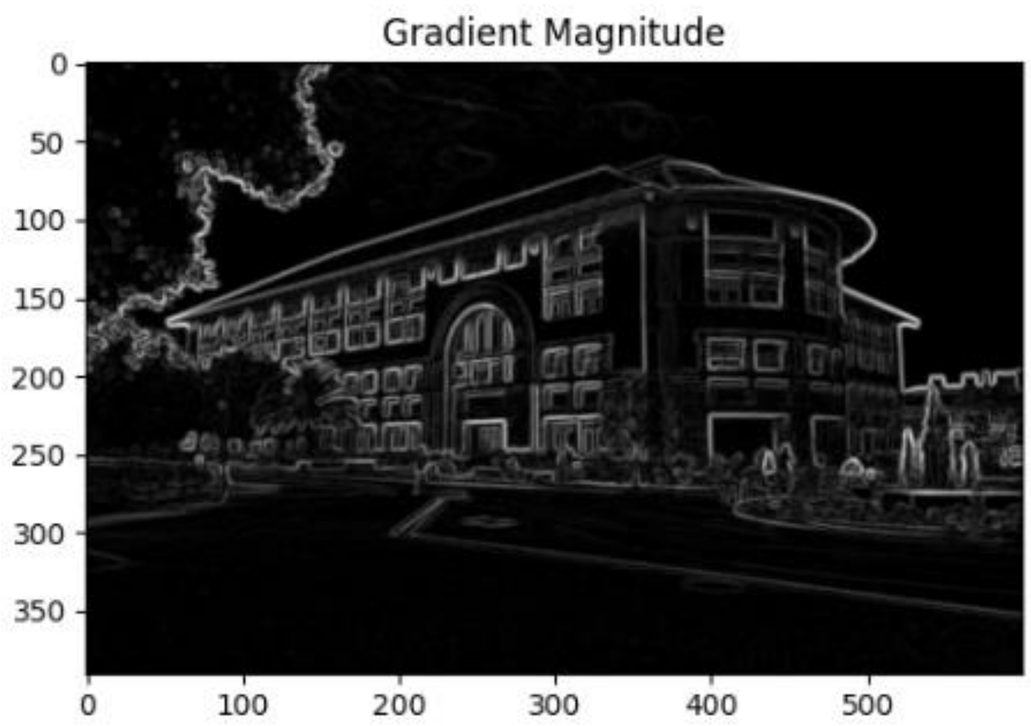
1차) zero padding : input image 경계에 zero padding을 사용하면, 경계 부분에서 값이 급격히 감소한 것으로 오인될 수 있음

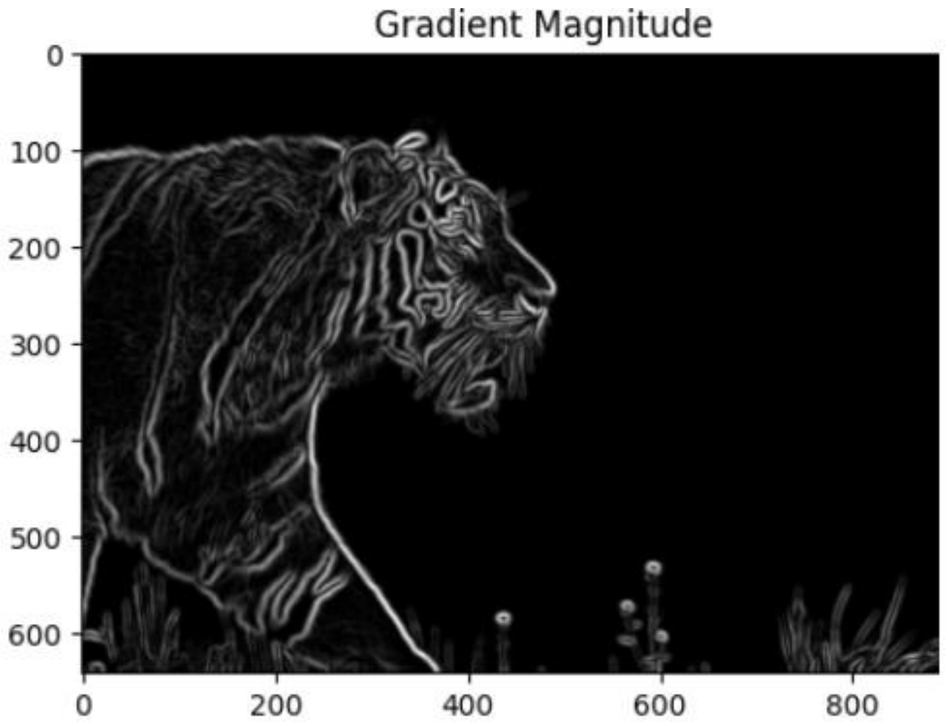
-> 2차) reflect padding

```
def my_convolve(image_array, kernel):
  h, w = image_array.shape
  kh, kw = kernel.shape
  pad_h, pad_w = (kh-1) // 2, (kw-1) // 2
  padded_image = np.pad(image_array, ((pad_h, pad_h), (pad_w, pad_w)), mode = 'reflect')
  result = np.zeros_like(image_array, dtype=np.float32)
  for i in range(h):
    for j in range(w):
      region = padded_image[i:i+kh, j:j+kw]
      result[i,j] = np.sum(region*kernel)
  return result
```

Canny Edge Detection

- Sobel Mask





Canny Edge Detection

- Gaussian Blurring ⇨ Sobel Mask ⇨ **Non-Maximum Suppression** ⇨ Hysteresis Thresholding

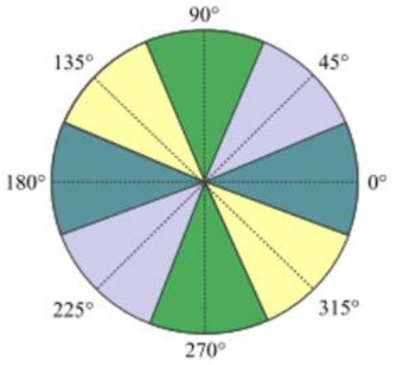
목적 : 강한 edge만을 남기고, 약하거나 중복된 edge는 억제

방식 : 주변 픽셀 값 중 최댓값은 유지하고 이외의 값은 0으로 제거

순서 : 1) gradient magnitude, angle 계산

$$Edge\_Gradient\,(G) = \sqrt{G_x^2 + G_y^2} \qquad Angle(\theta) = \tan^{-1}\left(\frac{G_x}{G_y}\right)$$

2) gradient 방향을 4구역으로 단순화



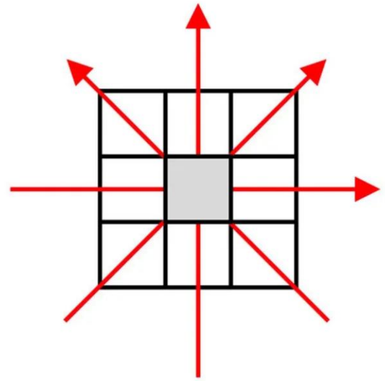

```
def angle_approximation(gradient_direction):
  h, w = gradient_direction.shape
  angle = (gradient_direction + 180) % 180
  for i in range(h):
    for j in range(w):
      if (22.5 < angle[i,j]< 67.5):
        angle[i,j] = 45
      elif (67.5 < angle[i,j] < 112.5):
        angle[i,j] = 90
      elif (112.5 < angle[i,j] < 157.5):
        angle[i,j] = 135
      elif (0 < angle[i,j] < 22.5) or (157.5 < angle[i,j] < 180):
        angle[i,j] = 0
  return angle
```

Canny Edge Detection

- Non-Maximum Suppression

순서 : 3) gradient 방향에 따라, 이웃한 픽셀 2개를 조사하여 local maximum 여부를 확인

Gradient 방향 (θ)	이웃 픽셀 1	이웃 픽셀 2
0° (수평)	$(i, j-1)$ (왼쪽)	$(i, j+1)$ (오른쪽)
45° (오른쪽 대각선)	$(i-1, j+1)$ (오른쪽 위 ↗)	$(i+1, j-1)$ (왼쪽 아래 ↙)
90° (수직)	$(i-1, j)$ (위쪽)	$(i+1, j)$ (아래쪽)
135° (왼쪽 대각선)	$(i-1, j-1)$ (왼쪽 위 ↖)	$(i+1, j+1)$ (오른쪽 아래 ↘)

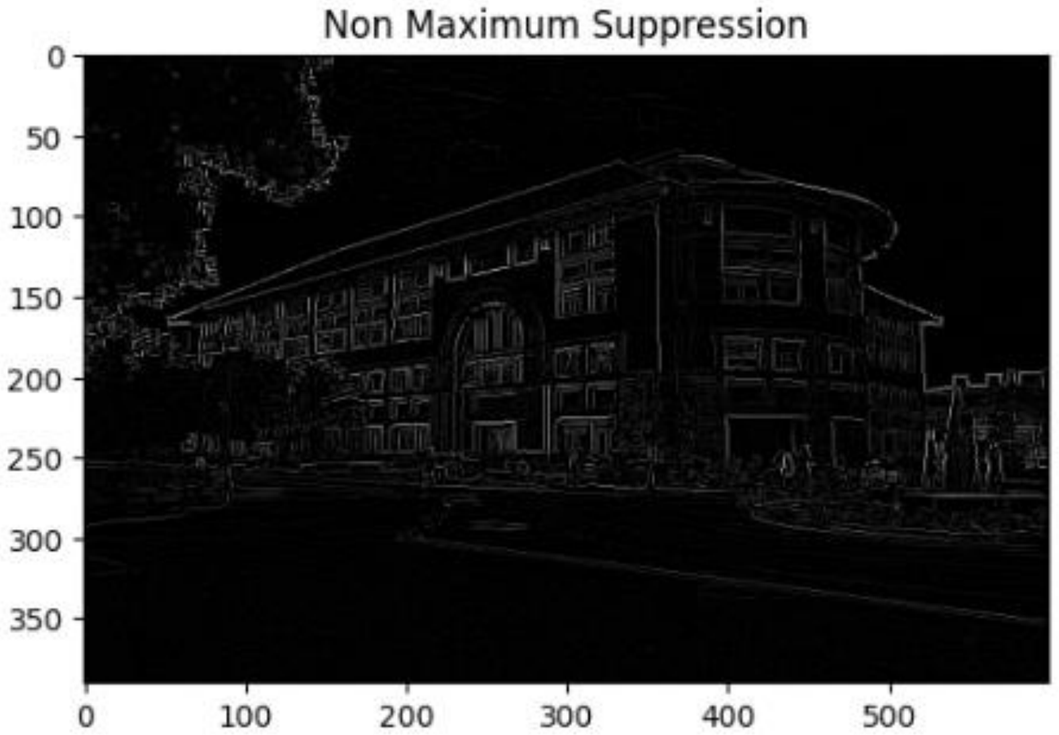


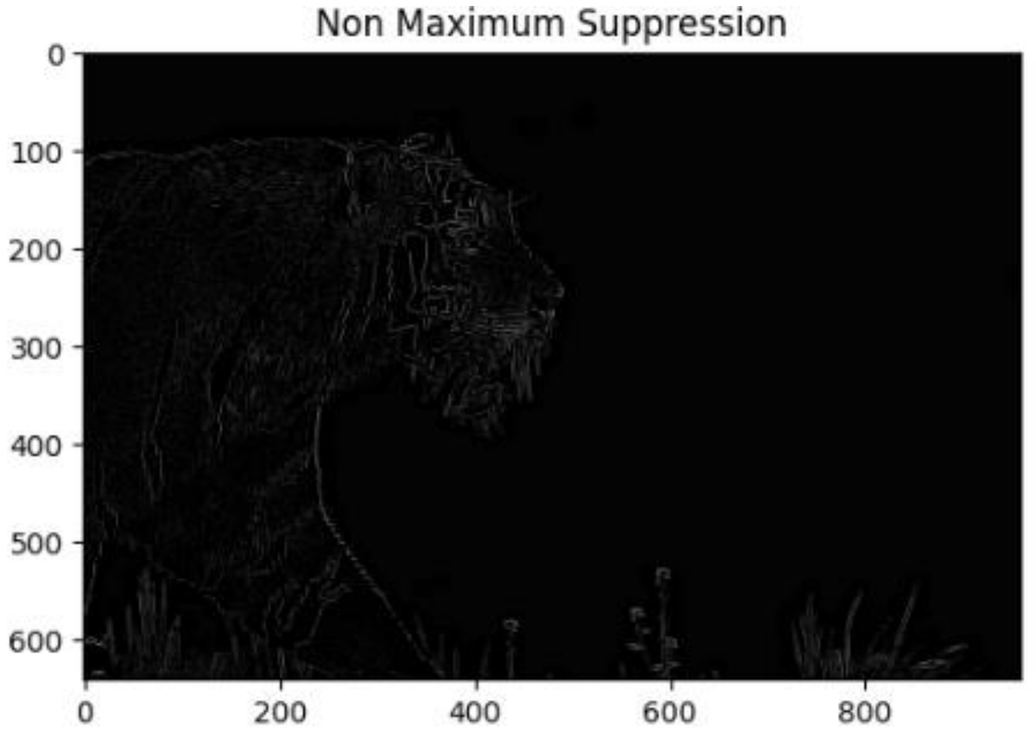
4) local maximum인 경우 유지하고, 이외의 값은 0으로 설정

```
nms = np.zeros((h,w), dtype = np.float32)
for i in range(1, h-1):
  for j in range(1, w-1):
    angle = gradient_approximate_direction[i,j]
    magnitude = gradient_magnitude[i,j]
    if angle == 0:
      max_value = max(gradient_magnitude[i, j-1], gradient_magnitude[i, j+1], magnitude)
    elif angle == 45:
      max_value = max(gradient_magnitude[i-1, j+1], gradient_magnitude[i+1, j-1], magnitude)
    elif angle == 90:
      max_value = max(gradient_magnitude[i-1, j], gradient_magnitude[i+1, j], magnitude)
    elif angle == 135:
      max_value = max(gradient_magnitude[i-1, j-1], gradient_magnitude[i+1, j+1], magnitude)
    else :
      nms[i,j] = magnitude
    if max_value == magnitude:
      nms[i,j] = max_value
```

Canny Edge Detection

- Non-Maximum Suppression





Canny Edge Detection

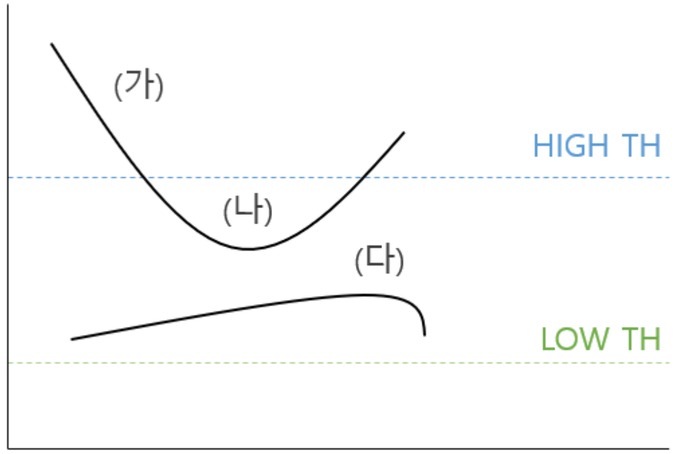
- Gaussian Blurring ⇨ Sobel Mask ⇨ Non-Maximum Suppression ⇨ **Hysteresis Thresholding**

원리 : 강한 edge는 유지하고 , 약한 edge는 강한 edge와 연결되어 있는 경우에만 유지

방식 : (가) high threshold보다 큰 값을 가지는 강한 edge는 유지

(나) low threshold보다 크지만 high threshold보다 작은 값을 가지는 edge는 주변 edge에 따라 선택

(다) low threshold보다 작은 값을 가지는 약한 edge는 제거

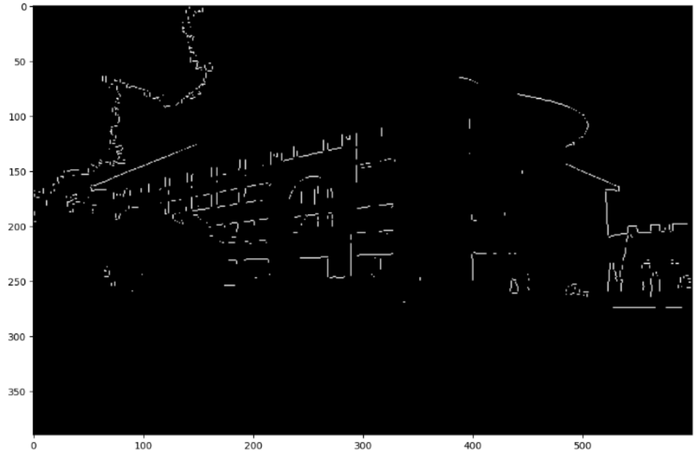


(가) (나) 유지
(다) 제거

```
for i in range(1, h-1):
  for j in range(1, w-1):
(가)  if nms[i,j] >= high_threshold :
        hysteresis[i,j] = 255
(나)  elif nms[i,j] >= low_threshold :
        if np.any(hysteresis[i - 1:i + 2, j - 1:j + 2] >= high_threshold):
          hysteresis[i,j] = 255
```

Canny Edge Detection

- Hysteresis Thresholding



High: 250
Low: 200

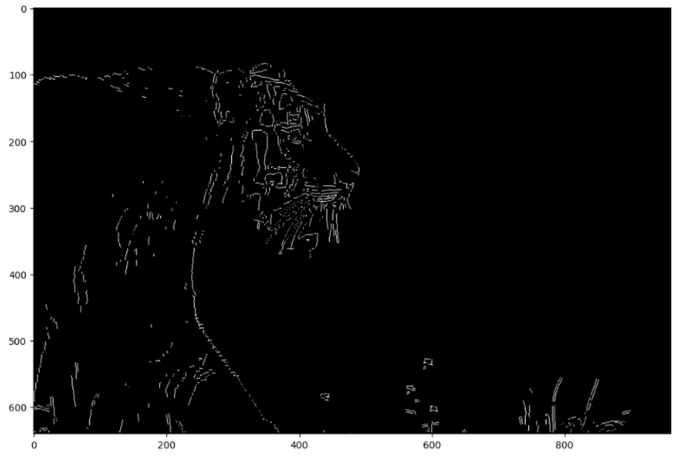
High : 150
Low : 100

High : 100
Low : 50

High: 40
Low: 30

Canny Edge Detection

- Hysteresis Thresholding



High : 200
Low : 150

High : 130
Low: 100

High : 100
Low : 50

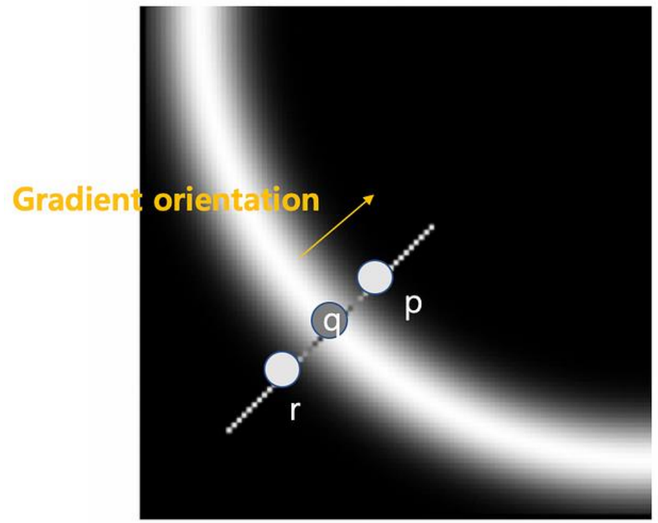
High : 60
Low : 40

Canny Edge Detection

- Non-Maximum Suppression vs Hysteresis Thresholding

Non-Maximum Suppression : edge의 가장 강한 부분(최댓값)만을 남기는 과정
-> edge가 더 얇고 명확하게 남도록 함

Hysteresis Thresholding : 임계점 이하의 약한 edge를 제거하는 과정
-> noise거나 유의미하지 않은 edge일 가능성이 있으므로 제거



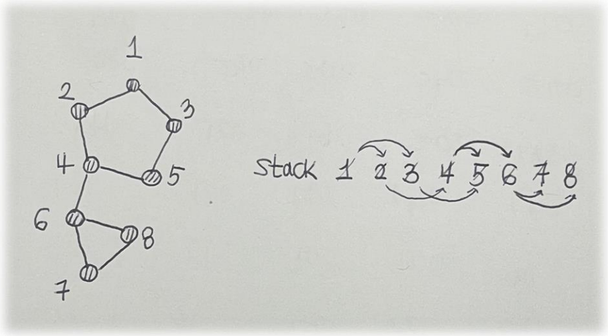
Boundary Tracing

- 8방향 Boundary Tracing

방식 : 임의의 픽셀에서 8방향 (상하좌우 및 대각선)을 따라가며 연결될 픽셀들을 찾아 객체의 외각선을 검출

1차) bfs : 한 점에서 여러 방향으로 뻗어 나가는 선들이 다른 경계에 속하더라도 하나의 stack에서 관리되면서 경계 혼합 문제 발생

-> 2차) dfs + backtracking 이용

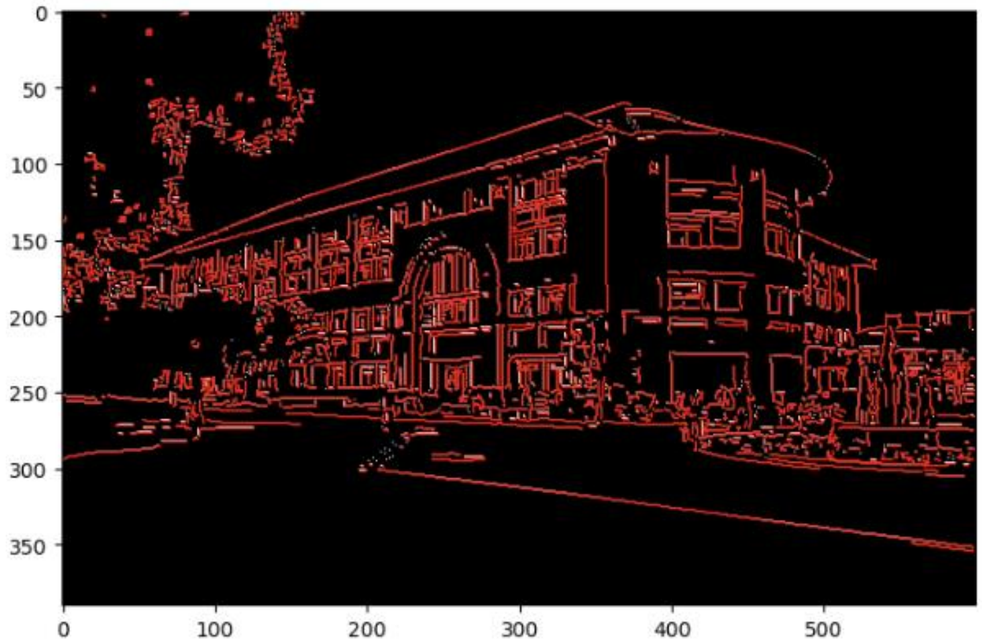


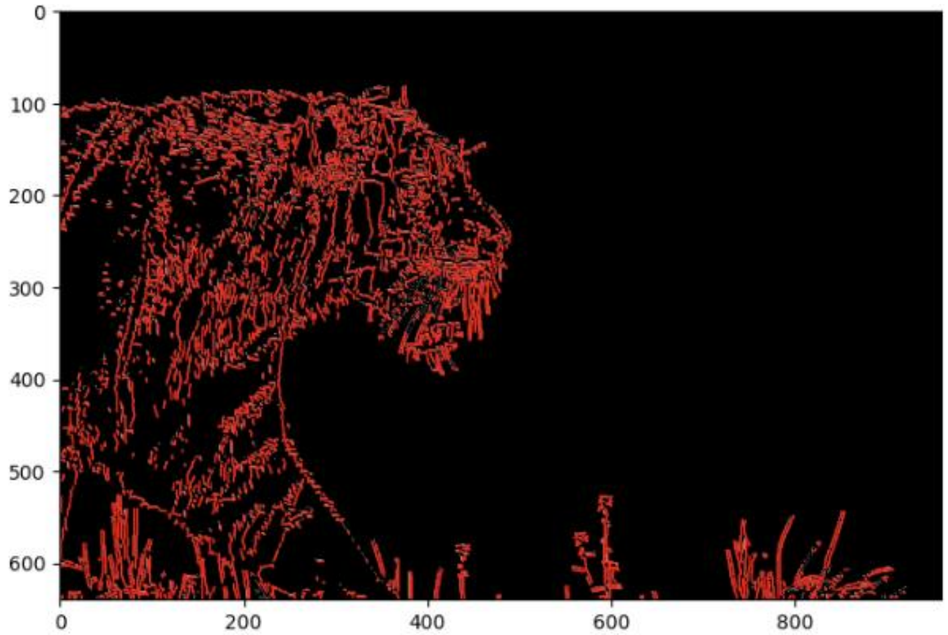

```
for i in range(h):
    for j in range(w):
      if hysteresis2[i,j] == 255:
        if visited[i,j] == 0:
          boundary = boundary_tracing(i, j, [], [])
          for path in boundary:
            path = np.array(path)
            plt.plot(path[:, 1], path[:, 0], linewidth=1, color='red')
```

```
def boundary_tracing(x, y, path, all_paths):
  path.append((x,y))
  visited[x,y] = 1
  flag = 0
  for dx, dy in directions:
    if x+dx < 0 or x+dx >= h or y+dy < 0 or y+dy >= w:
      continue
    if hysteresis2[x+dx, y+dy] == 255:
      if visited[x+dx,y+dy] == 0:
        flag = 1
        boundary_tracing(x+dx, y+dy, path, all_paths)
  if not flag:
    all_paths.append(path[:])
  path.pop()
  return all_paths
```

Boundary Tracing

- 8방향 Boundary Tracing





Introduction

- Deep Learning

weight & bias : 학습 과정에서 조정되는 파라미터 -> 모델이 데이터를 학습하며 최적의 값을 찾아나감

forward propagation : 입력 데이터를 신경망의 각 layer에 전달하여 최종 출력을 계산하는 과정

backward propagation : 모델의 예측 값과 실제 값의 차이인 손실 함수를 기반으로 gradient를 계산

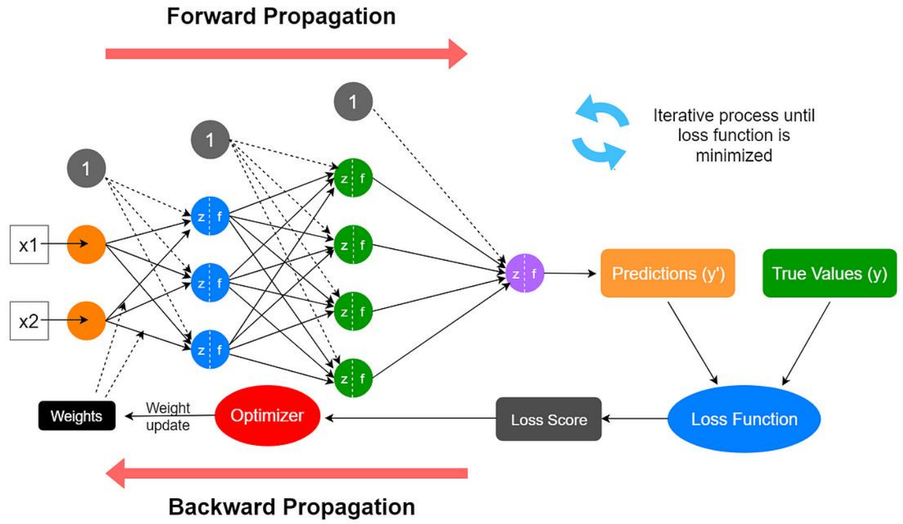
-> gradient를 역방향으로 전파하며 가중치를 업데이트

$$W = W - \eta \cdot \frac{\partial L}{\partial W}$$

Chain Rule

$$\frac{\partial f}{\partial x} = \frac{\partial q}{\partial x}\frac{\partial f}{\partial q}$$

Downstream Gradient / Local Gradient / Upstream Gradient



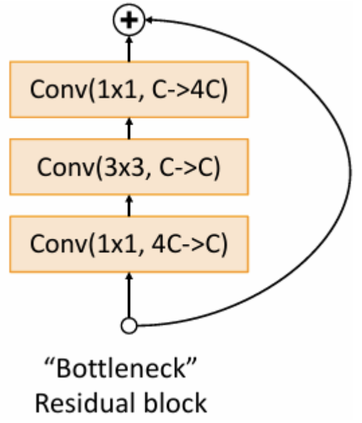
Introduction

- CNN model : ResNet

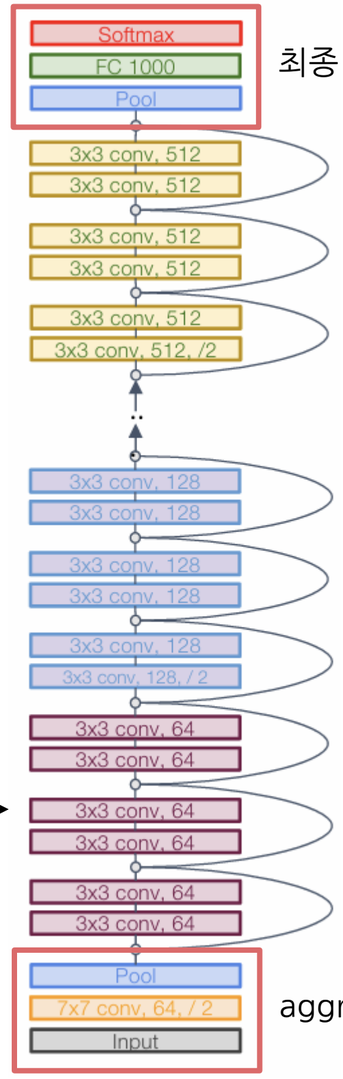
특징 : Residual Learning (입력 값을 그대로 전달)을 도입하여 기울기 소실 문제를 완화
-> 매우 깊은 network에서도 학습이 가능하며 성능이 매우 향상됨

방식 : Convolution Layer - CNN의 핵심 계층으로, 특징을 감지하고 feature map 출력
Fully Connected Layer - 입력된 특징을 기반으로 최종 출력을 생성
Pooling Layer - 이미지의 크기를 downsampling하여 계산량을 줄임
activation function - 비선형성을 부여하여 복잡한 관계를 학습하게 도움

```python
class Bottleneck(nn.Module):
    expansion = 4
    #첫 번째 convolution layer에서 채널 수를 줄이고, 마지막 convolution layer에서 채널 수를 다시 늘림

    def __init__(self, inplanes, planes, stride=1, downsample=None):
        super(Bottleneck, self).__init__()
        self.conv1 = conv1x1(inplanes, planes)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = conv3x3(planes, planes, stride)
        self.bn2 = nn.BatchNorm2d(planes)
        self.conv3 = conv1x1(planes, planes * self.expansion)
        self.bn3 = nn.BatchNorm2d(planes * self.expansion)
        self.relu = nn.ReLU(inplace=True)
        self.downsample = downsample
        self.stride = stride

    def forward(self, x):
        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)
        #만약에 stride가 2라면, output과 input의 크기가 맞지 않으므로 input을 output 크기에 맞춰서 변환해야 함
        out += identity
        out = self.relu(out)

        return out
```



"Bottleneck"
Residual block



최종 feature 추출

aggressive stem

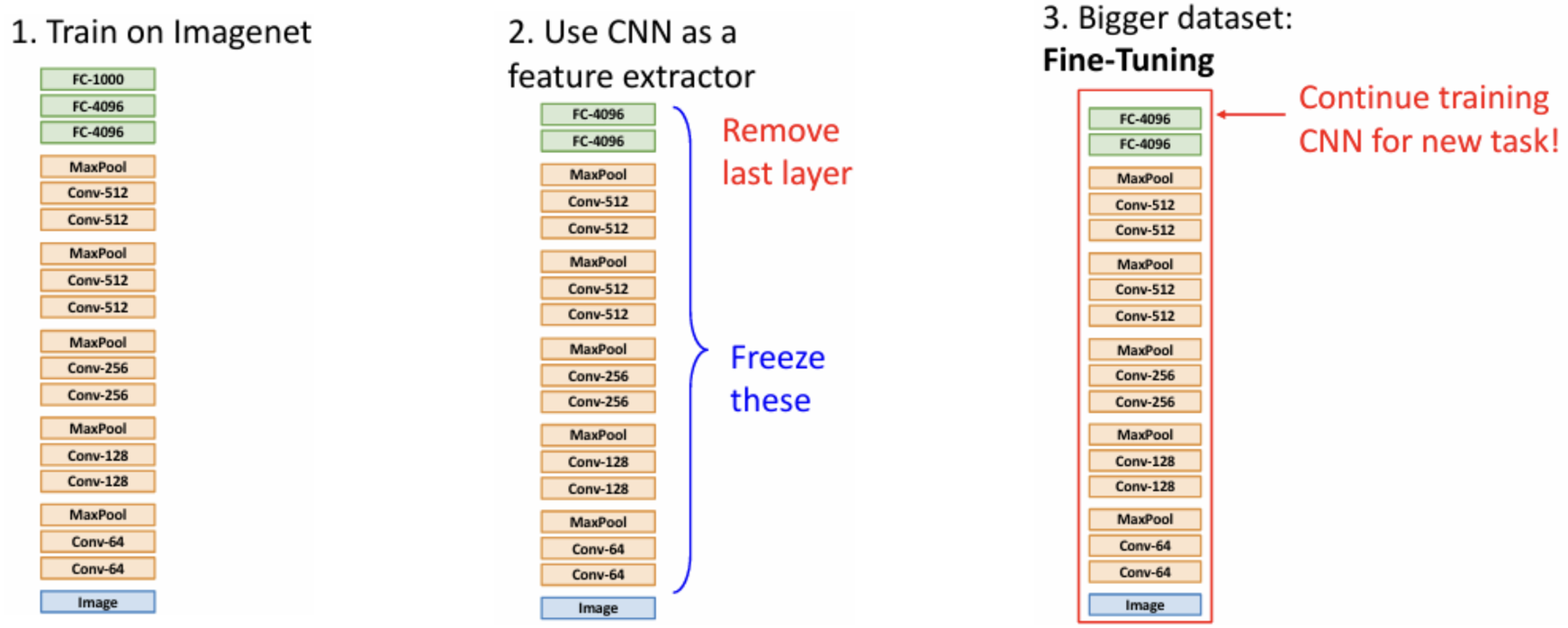
Introduction

- Transfer Learning

정의 : 사전 학습된 Neural Network의 지식을 활용하여 새로운 데이터셋에 적합하도록 조정하여 사용하는 방식

장점 : 학습 시간 단축, 적은 데이터로도 학습 가능, 학습 효율 향상

방식:



초기의 shallow layer : low-level features vs 후반의 deep layer : high-level features -> 특정 작업에 특화된 특징을 학습

Deep Learning

- Richer Convolutional Features

원리 : Multi-Scale Features - CNN의 각 계층에서 독립적으로 edge map을 생성하고, 이를 융합하여 최종 경계 map을 추출

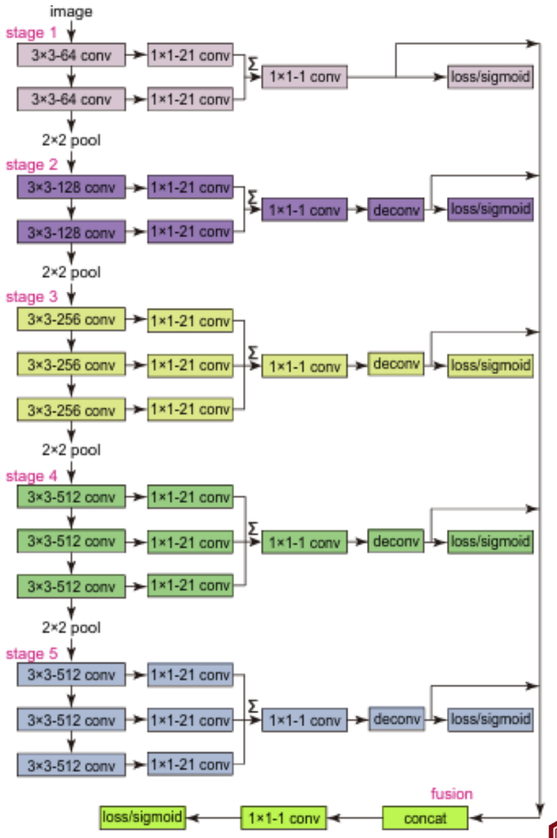
-> shallow layer의 low-level feature와 deep layer의 high-level feature를 동시에 활용하여 정밀하게 edge를 검출

- Richer Convolutional Features vs Holistically-Nested Edge Detection

공통 : 다중 계층 특징을 활용

RCF : 각 계층의 edge 맵에 가중치를 학습 가능한 형태로 설정하여, 기여도를 학습을 통해 얻음

HED : 각 계층의 edge 맵에 각 계층에서 생성된 edge map을 단순 가중합 방식으로 융합



Deep Learning

- Richer Convolutional Features

방식 : 1) ResNet의 각 layer에서 특징 맵 추출

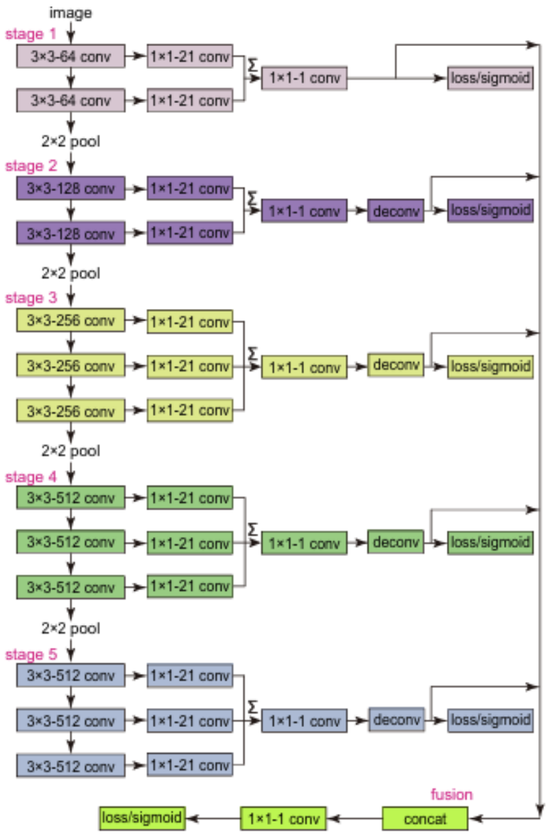
2) 각 특징 맵의 채널을 축소

3) 각 축소된 특징 맵의 채널을 1로 조정

4) 모든 특징 맵을 입력 이미지 크기로 복원

5) 복원된 output을 concatenate하여 최종 결과 생성

```
def forward(self, x, size):
    #x 1
    x = self.conv1(x)  #1/2
    x = self.bn1(x)
    x = self.relu(x)
    C1 = self.maxpool(x) #1/4
    C2 = self.layer1(C1) #1/4
    C3 = self.layer2(C2) #1/8
    C4 = self.layer3(C3) #1/16
    C5 = self.layer4(C4) #1/32


    R1 = self.relu(self.C1_down_channel(C1))
    R2 = self.relu(self.C2_down_channel(C2))
    R3 = self.relu(self.C3_down_channel(C3))
    R4 = self.relu(self.C4_down_channel(C4))
    R5 = self.relu(self.C5_down_channel(C5))

    so1_out = self.score_dsn1(R1)
    so2_out = self.score_dsn2(R2)
    so3_out = self.score_dsn3(R3)
    so4_out = self.score_dsn4(R4)
    so5_out = self.score_dsn4(R5)

    upsample = nn.UpsamplingBilinear2d(size)

    #입력 image 크기에 맞게 upsampling
    out1 = upsample(so1_out)
    out2 = upsample(so2_out)
    out3 = upsample(so3_out)
    out4 = upsample(so4_out)
    out5 = upsample(so5_out)

    fuse = torch.cat([out1, out2, out3, out4, out5], dim=1)
    final_out = self.score_final(fuse)

    results = [out1, out2, out3, out4, out5, final_out]
    results = [torch.sigmoid(r) for r in results]
    return results
```



Deep Learning

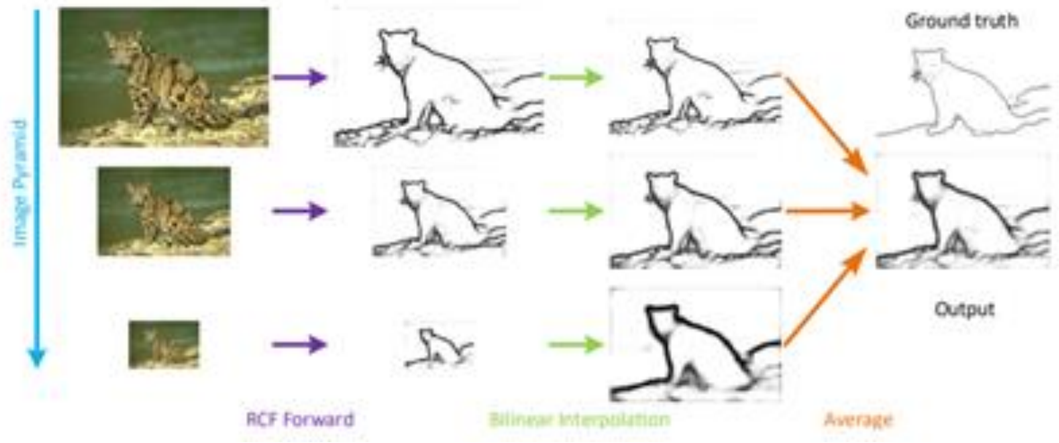
- Richer Convolutional Features

방식: 1) 사전 학습된 가중치를 load하여 사용

2) input image를 전처리 과정을 거쳐 신경망 모델에 전달

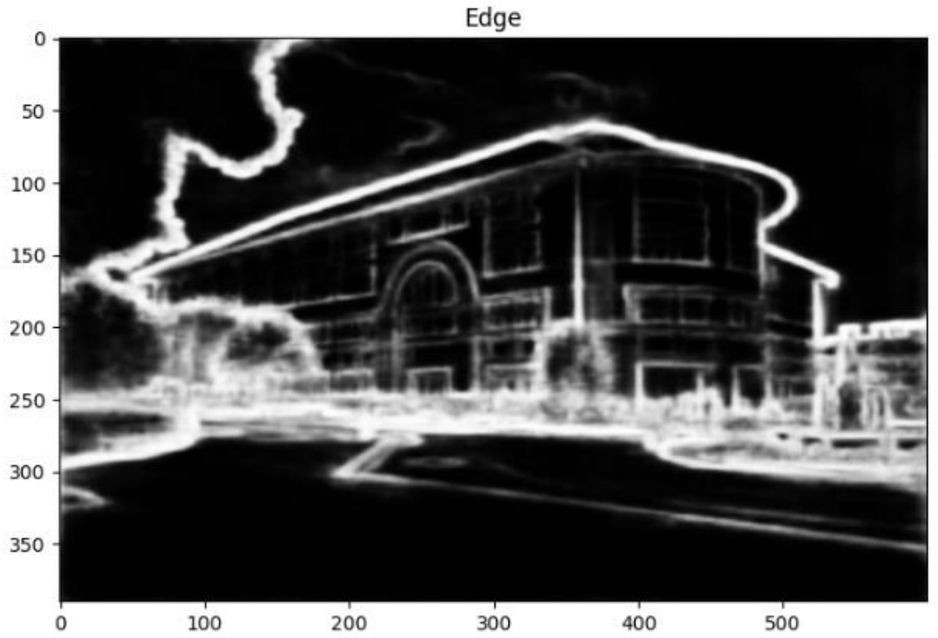
3) 신경망의 output을 edge map의 결과로 사용

```
PATH_WEIGHT = '/content/drive/MyDrive/VDSL/pretrained_weight.pth' #사전 학습되어 저장된 weight들
class RCF():

    def __init__(self, device='cpu'):
        tstamp = time.time()
        self.device = device
        device = torch.device(device)
        self.net = resnet101(pretrained=False)
        print('[RCF] loading with', device)
        self.net.load_state_dict(torch.load(PATH_WEIGHT, map_location=device))
        self.net.eval()
        print('[RCF] finished loading (%.4f sec)' % (time.time() - tstamp))

    def detect_edge(self, img):
        start_time = datetime.datetime.now()
        print('edge detection start : {}'.format(start_time))
        org_img = np.array(img, dtype=np.float32)
        h, w, _ = org_img.shape
        pre_img = self.prepare_image_cv2(org_img)
        pre_img = torch.from_numpy(pre_img).unsqueeze(0) # (C x H x W) -> (batch x C x H x W)
        outs = self.net(pre_img, (h, w))
        result = outs[-1].squeeze().detach().numpy()
        # result = (result * 255).astype(np.uint8)
        end_time = datetime.datetime.now()
        print('edge dection end: {}'.format(end_time))
        time_delta = end_time - start_time
        print('edge detection time : {} 초'.format(time_delta.seconds) + "₩n")
        return result
```



Deep Learning

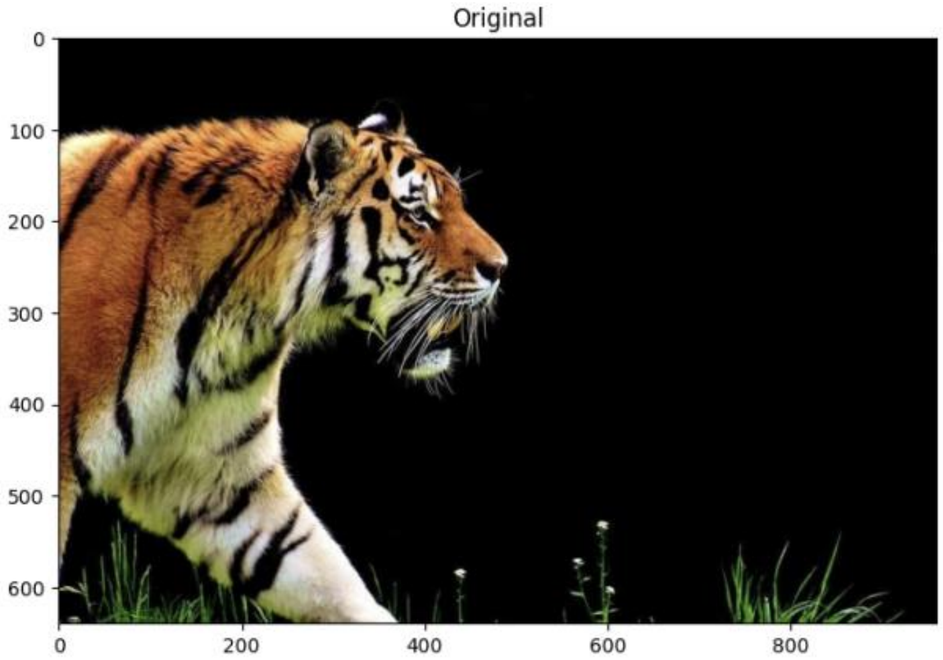
- Richer Convolutional Features

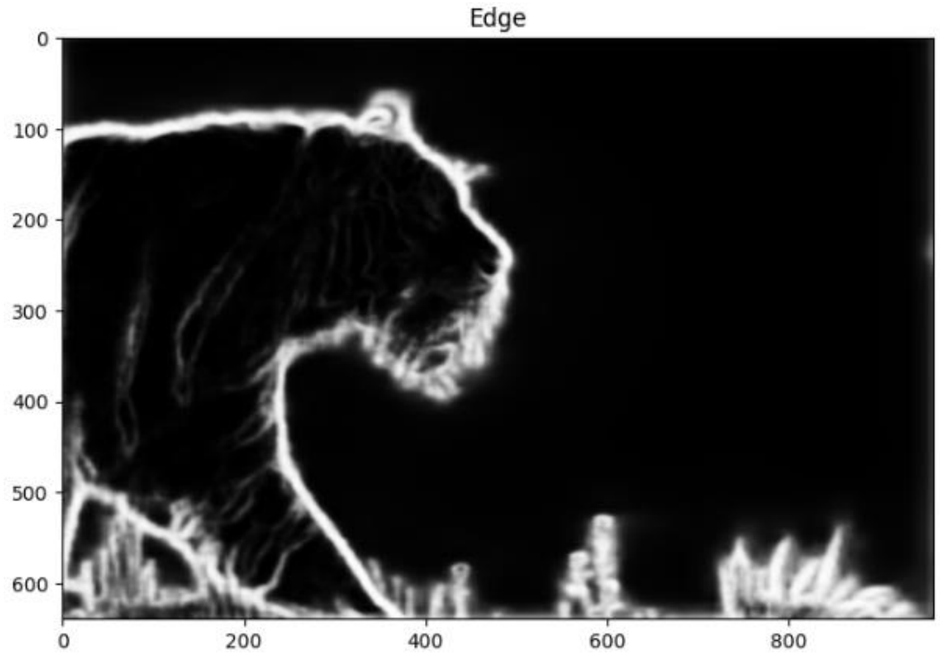



```
edge detection start : 2025-01-08 07:09:24.579958
edge dection end: 2025-01-08 07:09:32.012183
edge detection time : 7 초
```

Deep Learning

- Richer Convolutional Features






```
edge detection start : 2025-01-03 06:06:05.552462
edge dection end: 2025-01-03 06:06:13.325740
edge detection time : 7 초
```

Conclusion

- Edge Detection은 Computer Vision에서 핵심적인 역할을 하며, 다양한 응용 분야에 사용됨
- 입력 이미지의 특성에 따라 여러 edge detection 알고리즘 중 적절한 방법을 선택하여 적용할 수 있음
- 본 연구에서는 Edge Detection의 여러 알고리즘 중 Canny Edge Detection 알고리즘과 ResNet 기반의 Richer Convolutional Features의 동작을 분석함
- Canny Edge Detection : Gaussian Blurring → Sobel Mask → Non-Maximum Suppression → Hysteresis Thresholding 단계를 통해 연속적이고 명확한 edge를 검출
- RCF : ResNet 기반 CNN 모델의 다중 계층에서 추출한 특징을 통합하여 정밀하고 효율적으로 edge 검출