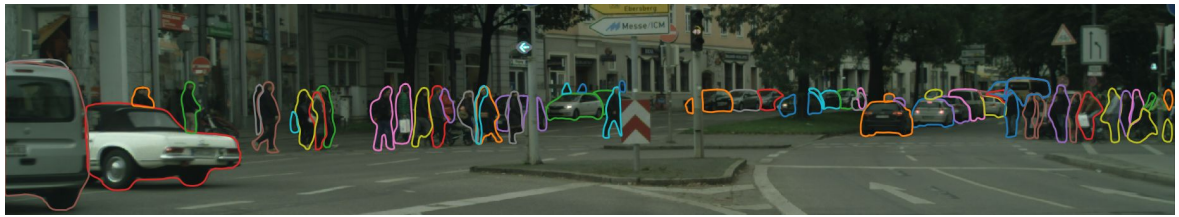
Deep Snake for Real-Time Instance Segmentation

Sida Peng[1] Wen Jiang[1] Huaijin Pi[1] Xiuli Li[2] Hujun Bao[1] Xiaowei Zhou[1*]

[1]Zhejiang University [2]Deepwise AI Lab



CVPR'20 Oral

DAVIAN Vision Study
2021.04.05
양소영

https://github.com/zju3dv/snake
https://www.youtube.com/watch?v=ZykJ9nyIPNI

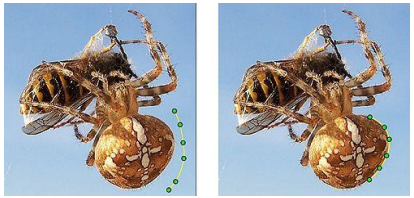
Contents

1.1 Snake algorithm, an active contour model (1988)

http://www.cs.ait.ac.th › cvreadings › Kass-Snakes PDF

Snakes: Active contour models

M KASS 저술 · 1988 · 24426회 인용 — A **snake** is an energy-minimizing spline guided by external constraint forces and influenced by image forces that pull it toward features such as lines and edges. **Snakes** are **active contour models**: they lock onto nearby edges, localizing them accurately.



The energy function of the snake is the sum of its external energy and internal energy, or

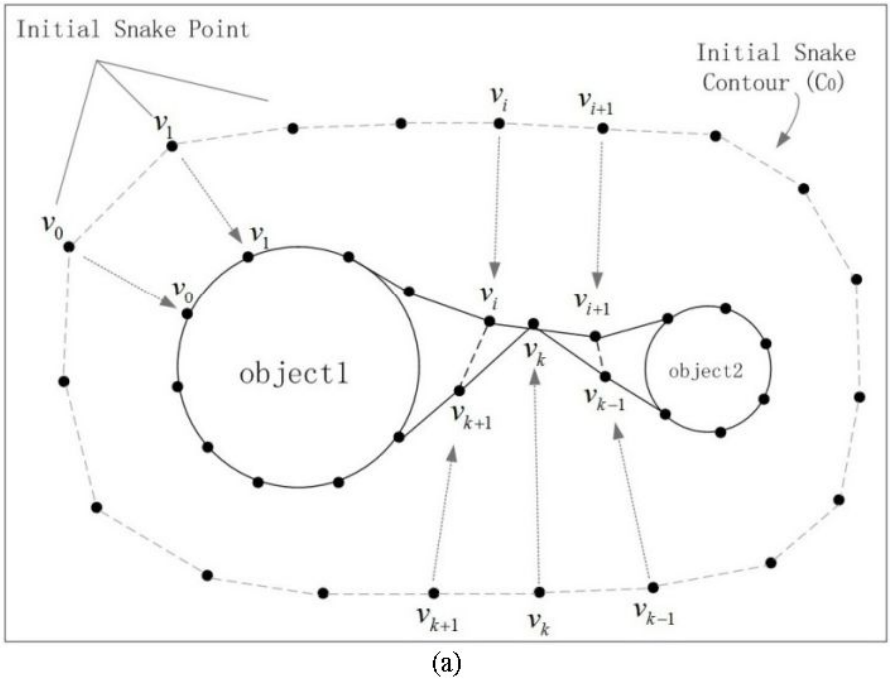
$$E^*_{\text{snake}} = \int_0^1 E_{\text{snake}}(\mathbf{v}(s))\,ds = \int_0^1 (E_{\text{internal}}(\mathbf{v}(s)) + E_{\text{image}}(\mathbf{v}(s)) + E_{\text{con}}(\mathbf{v}(s)))\,ds$$

- Snake is an active contour for representing object contours.
- The snake algorithm introduced by Kass et al. [1] has been widely applied in various object contour detection and object tracking
- an energy minimizing, deformable spline influenced by constraint and image forces that pull it towards object contours and internal forces that resist deformation.
- E_snake' : { int : snake 선의 모양 결정, image: 이미지 피쳐로 미는 힘*, cons: 사용자의 제약}

A SNAKE ALGORITHM FOR AUTOMATICALLY TRACKING MULTIPLE OBJECTS, Fang et al., IEEE International Conference on Image Processing, 2011

https://en.wikipedia.org/wiki/Active_contour_model

Iteration process of a snake contour



- Simply, snake algorithm moves its point towards the object by minimizing the energy (similar with Loss function).
- From initial snake contour (C_0), iteratively thightes the initial snake points to the object.

A SNAKE ALGORITHM FOR AUTOMATICALLY TRACKING MULTIPLE OBJECTS, Fang et al., IEEE International Conference on Image Processing, 2011

E_img

옛날엔 이미지 처리를 어떻게 했을까?

1 2 3

$$E_{\text{image}} = w_{\text{line}}E_{\text{line}} + w_{\text{edge}}E_{\text{edge}} + w_{\text{term}}E_{\text{term}}$$

1

- 이미지 라인 함수

$$E_{\text{line}} = I(x, y)$$

- 가장 간단하게 이용할 수 있는 함수로 영상의 밝기 자체를 이용.
- 가장 밝은 선과 어두운 선이 contour를 의미하므로, weight의 부호에 따라 밝은 선 혹은 어두운 선으로 Snake는 이동.

3

- Line segments와 corners의 끝을 찾기 위해 Smooth된 영상에서 line의 curvature를 이용.
  - 내부 에너지 함수의 2차 미분
- 종결 함수

$$C(x, y) = G_\sigma(x, y) * I(x, y)$$

$$\theta = \tan^{-1}(C_y / C_x) \qquad n = (con\theta, \sin\theta) \qquad n_\perp = (-\sin\theta, \cos\theta)$$

$$E_{\text{term}} = \frac{\partial\theta}{\partial\mathbf{n}_\perp} = \frac{\partial^2 C/\partial\mathbf{n}_\perp^2}{\partial C/\partial\mathbf{n}} = \frac{C_{yy}C_x^2 - 2C_{xy}C_xC_y + C_{xx}C_y^2}{(C_x^2 + C_y^2)^{3/2}}$$

2

- Edge 함수

$$E_{\text{edge}} = -|\nabla I(x, y)|^2$$

- 영상에 대한 1차 미분 결과로 영상에서 gradient가 큰 부분을 찾을 때 최소값을 갖는다.

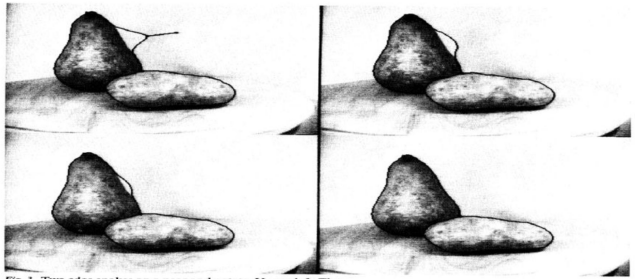


Fig. 3. Two edge snakes on a pear and potato. Upper-left: The user has pulled one of the snakes away from the edge of the pear. Others: After the user lets go, the snake snaps back to the edge of the pear.

http://slidesplayer.org/slide/14001590/

1.2 CenterNet

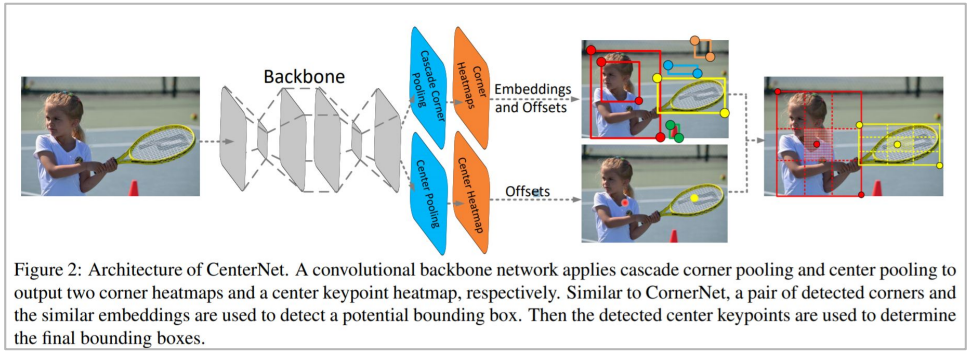
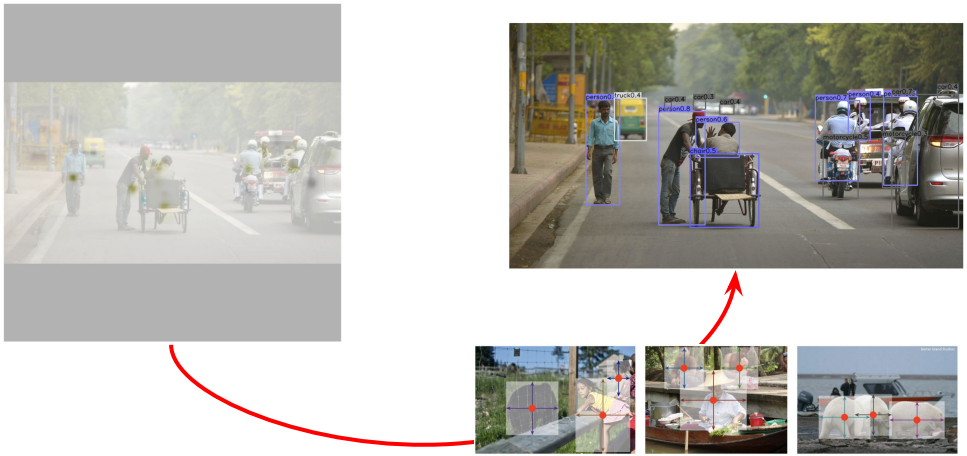


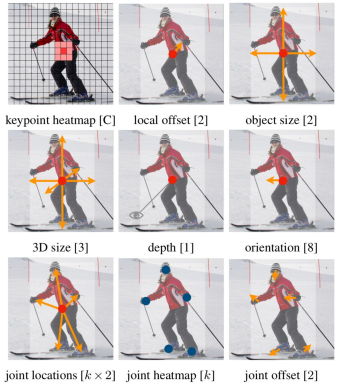
Figure 2: Architecture of CenterNet. A convolutional backbone network applies cascade corner pooling and center pooling to output two corner heatmaps and a center keypoint heatmap, respectively. Similar to CornerNet, a pair of detected corners and the similar embeddings are used to detect a potential bounding box. Then the detected center keypoints are used to determine the final bounding boxes.



- One-stage Detector
  - (X) box overlap (O) location → allocate “Anchor”
  - use only one anchor
  - high output resolution: output stride of 4.
- Single Keypoint Estimation 을 통해서 물체마다 단 하나의 keypoint인 중심점 추정
  - keypoint estimation 에서 요구되는 grouping, post-processing 과정이 필요 없게 됨.
  - 이를 기반으로 bounding box 얻어냄.
- Heatmap from Keypoint estimation
- → Object detection 결과

https://arxiv.org/pdf/1904.08189v3.pdf

1.2 CenterNet



object detection

3D object detection

Pose Estimation

	AP			AP_{50}			AP_{75}			Time (ms)			FPS		
	N.A.	F	MS	N.A.	F	MS	N.A.	F	MS	N.A.	F	MS	N.A.	F	MS
Hourglass-104	**40.3**	**42.2**	**45.1**	**59.1**	**61.1**	**63.5**	**44.0**	**46.0**	**49.3**	71	129	672	14	7.8	1.4
DLA-34	37.4	39.2	41.7	55.1	57.0	60.1	40.8	42.7	44.9	19	36	248	52	28	4
ResNet-101	34.6	36.2	39.3	53.0	54.8	58.5	36.9	38.7	42.0	22	40	259	45	25	4
ResNet-18	28.1	30.0	33.2	44.9	47.5	51.5	29.6	31.6	35.1	**7**	**14**	**81**	**142**	**71**	**12**

Table 1: Speed / accuracy trade off for different networks on COCO validation set. We show results without test augmentation (N.A.), flip testing (F), and multi-scale augmentation (MS).

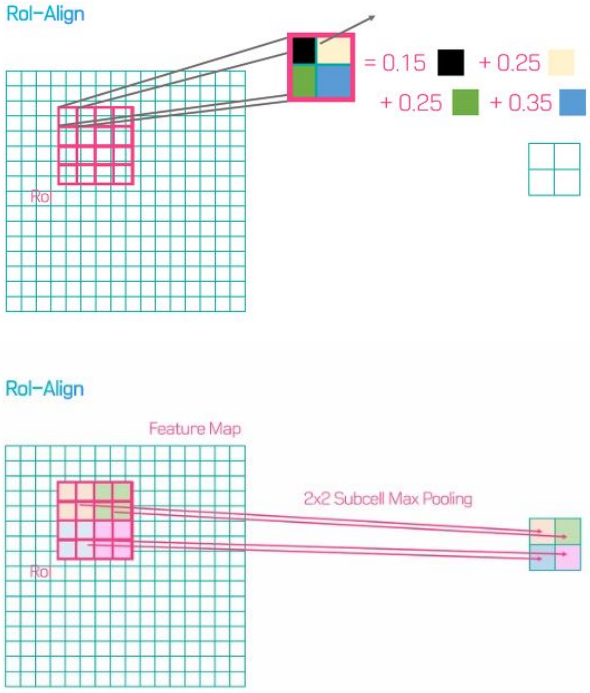
	Backbone	FPS	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
MaskRCNN [21]	ResNeXt-101	**11**	39.8	62.3	43.4	22.1	43.2	51.2
Deform-v2 [63]	ResNet-101	-	46.0	67.9	50.8	27.8	49.1	59.5
SNIPER [48]	DPN-98	2.5	46.1	67.0	51.6	29.6	48.9	58.1
PANet [35]	ResNeXt-101	-	47.4	67.2	51.8	30.1	**51.7**	60.0
TridentNet [31]	ResNet-101-DCN	0.7	**48.4**	**69.7**	**53.5**	**31.8**	51.3	**60.3**
YOLOv3 [45]	DarkNet-53	20	33.0	57.9	34.4	18.3	25.4	41.9
RetinaNet [33]	ResNeXt-101-FPN	5.4	40.8	61.1	44.1	24.1	44.2	51.2
RefineDet [59]	ResNet-101	-	36.4 / 41.8	57.5 / 62.9	39.5 / 45.7	16.6 / 25.6	39.9 / 45.1	51.4 / 54.1
CornerNet [30]	Hourglass-104	4.1	40.5 / 42.1	56.5 / 57.8	43.1 / 45.3	19.4 / 20.8	42.7 / 44.8	**53.9** / 56.7
ExtremeNet [61]	Hourglass-104	3.1	40.2 / 43.7	55.5 / 60.5	43.2 / 47.0	20.4 / 24.1	43.2 / 46.9	53.1 / 57.6
FSAF [62]	ResNeXt-101	2.7	**42.9** / 44.6	**63.8 / 65.2**	**46.3** / 48.6	**26.6 / 29.7**	**46.2 / 47.1**	52.7 / 54.6
CenterNet-DLA	DLA-34	**28**	39.2 / 41.6	57.1 / 60.3	42.8 / 45.1	19.9 / 21.5	43.0 / 43.9	51.4 / 56.0
CenterNet-HG	Hourglass-104	7.8	42.1 / **45.1**	61.1 / 63.9	45.9 / **49.3**	24.1 / 26.6	45.5 / **47.1**	52.8 / **57.7**

Table 2: State-of-the-art comparison on COCO test-dev. Top: two-stage detectors; bottom: one-stage detectors. We show single-scale / multi-scale testing for most one-stage detectors. Frame-per-second (FPS) were measured on the same machine whenever possible. Italic FPS highlight the cases, where the performance measure was copied from the original publication. A dash indicates methods for which neither code and models, nor public timings were available.

- Object Detection 외에도 3D object detection, multi-person human pose estimation 확장 가능
- Frame-per-second(FPS) 가 높음. ResNet-18 속도는 빠르지만 성능이 애매함..
- got a reject on a conference…

https://nuggy875.tistory.com/34

1.3 RoIAlign



- properly aligning the extracted features with the input.
- the results are not sensitive to the exact sampling locations, or how many points are sampled, as long as no quantization(e.g., 반올림 in `RoIPool`) is performed.
- bilinear interpolation & max pooling

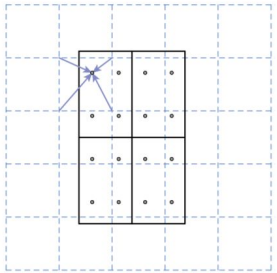


Figure 3. **RoIAlign:** The dashed grid represents a feature map, the solid lines an RoI (with 2×2 bins in this example), and the dots the 4 sampling points in each bin. RoIAlign computes the value of each sampling point by bilinear interpolation from the nearby grid points on the feature map. No quantization is performed on any coordinates involved in the RoI, its bins, or the sampling points.

Mask R-CNN, CVPR'17
https://mylifemystudy.tistory.com/82

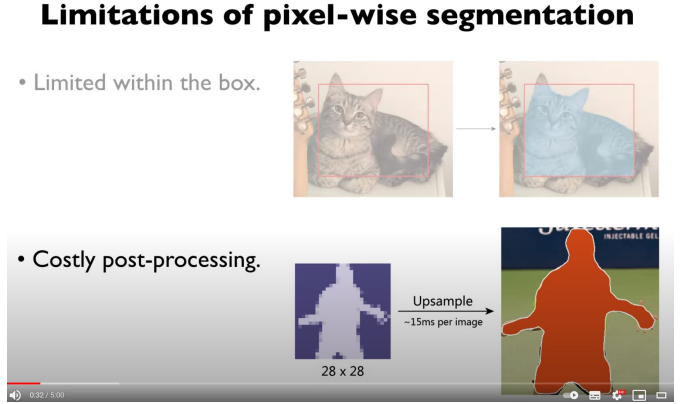
Deep Snake
for **Real-Time** Instance Segmentation

Contour-based segmentation model 로서 딥러닝 버전의 snake 알고리즘을 제시함.

1. snake 알고리즘이 object localization을 잘 하기 때문에 초반의 가벼운 detector(*CenterNet*) 으로도 좋은 성능이 나오고,
2. pixel-based 보다 파라미터가 적기 때문에 비싼 post-processing(e.g., mask upsampling) 필요X
3. 즉 가성비 좋은 알고리즘. **32.3 frame per seconds(fps) for 512x512** on a GTX 1080ti GPU

2. Problem Setting

- **Efficient Instance Segmentation in real-time**
- Pixel-wise segmentation
  - Most of SOTA models
  - sensitive to the inaccurate bounding box
  - representing an object shape as dense binary pixels → costly post-processing
- Object contour, as an alternative shape representation
  - contour: a set of vertices along the object silhouette
  - not limited within a bounding box and has fewer parameters than pixel one.
  - e.g., snake algorithm
- 이전: Learning-based contour-wise segmentation methods
  - regress the coordinates of contour vertices from an RGB image
  - pixel-based 만큼 성능이 좋지 않았음.
- 비교군: Curve-GCN
  - input: contour as graph , model: graph convolutional network(GCN)
  - annotation을 도와주기위해서 만든거지, 자동으로 instance segmentation해주는 완성된 파이프라인이 아님.



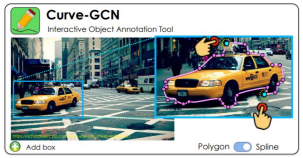
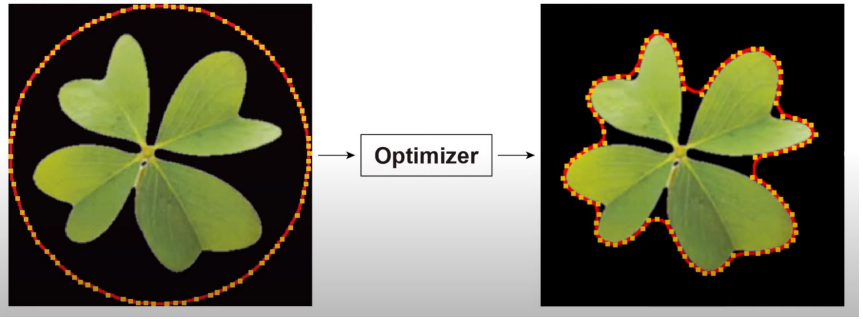


Figure 1: We propose Curve-GCN for interactive object annotation. In contrast to Polygon-RNN [7, 2], our model parametrizes

Novelty

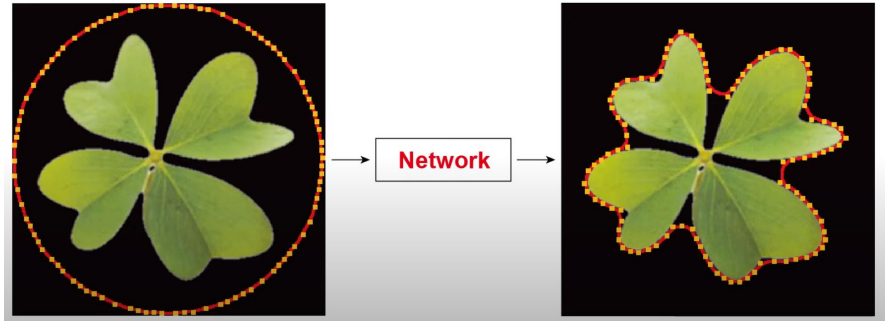
Snakes: active contour models

Segment objects by pulling a contour to match the object edge.



Deep Snake

Replace the hand-crafted optimizer with a neural network.



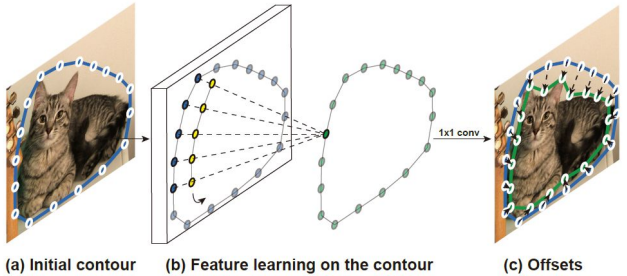
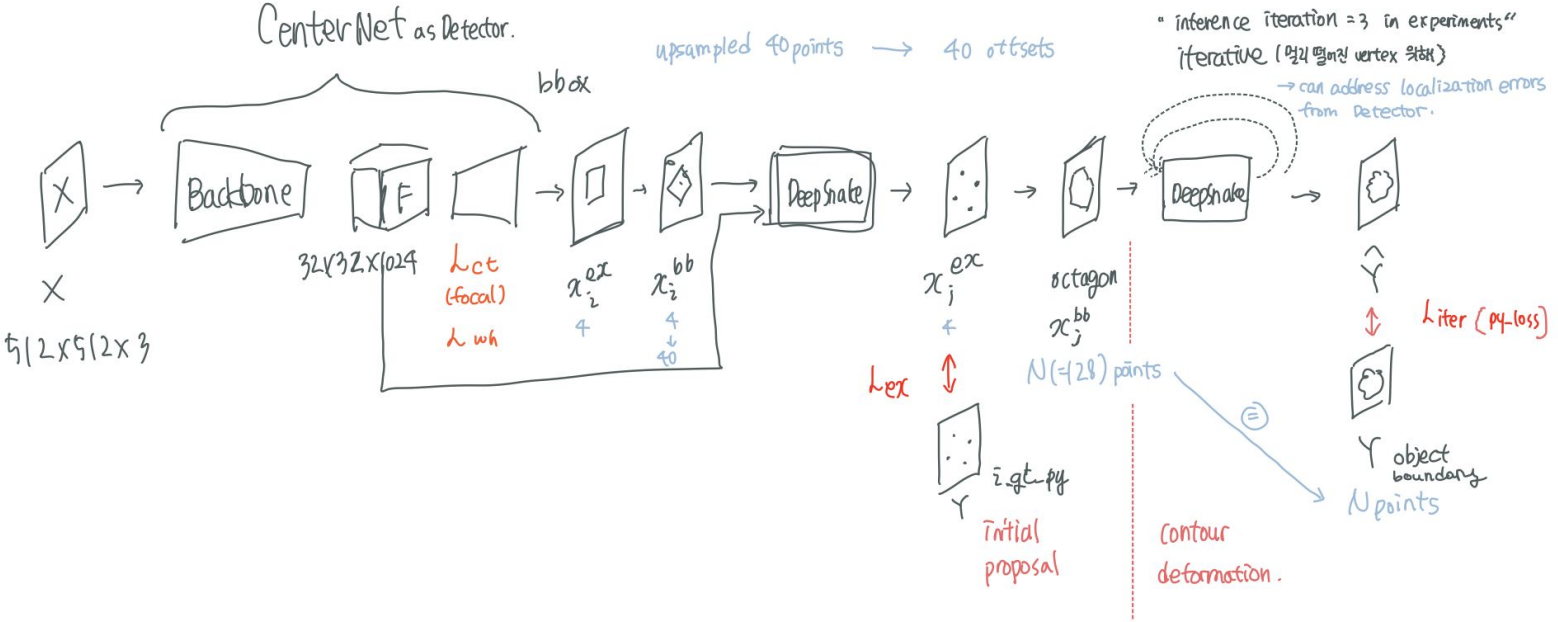
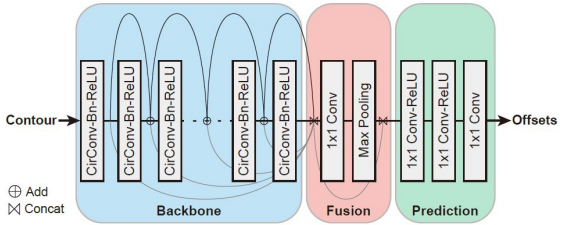


Figure 1. **The basic idea of deep snake.** Given an initial contour,

Overview


CenterNet as Detector.
bbox
upsampled 40 points → 40 offsets
" inference iteration = 3 in experiments"
iterative (점진 움직임 vertex 갱신)
→ can address localization errors from Detector.
X
Backbone
F
32x32x1024
Lct
(focal)
λ wh
DeepSnake
DeepSnake
octagon
Lex
N(=128) points
Liter (py-loss)
i_gt_py
object boundary
N points
Initial proposal
contour deformation.

3. Learning-based snake algorithm



(a) Deep snake

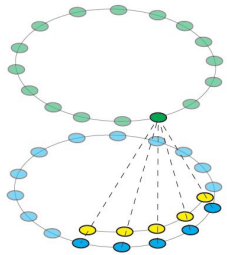


Figure 2. **Circular Convolution.** The blue nodes are the input features defined on a contour, the yellow nodes represent the kernel function, and the green nodes are the output features. The highlighted green node is the inner product between the kernel function and the highlighted blue nodes, which is the same as the standard convolution. The output features of circular convolution have the same length as the input features.

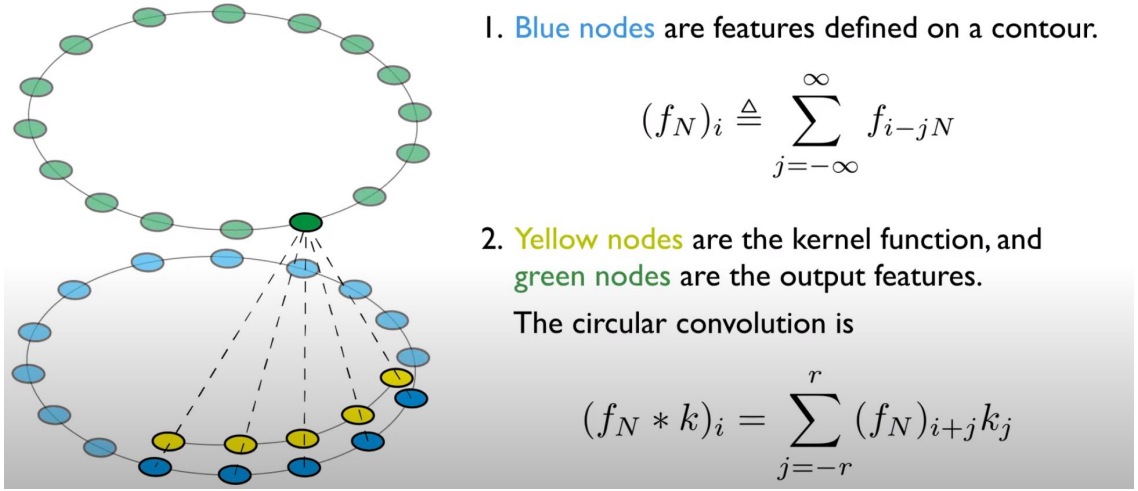
each vertex. The input feature f_i for a vertex $\mathbf{x}_i$ is a concatenation of learning-based features and the vertex coordinate: $[F(\mathbf{x}_i); \mathbf{x}_i]$, where F denotes the feature maps. The feature maps F are obtained by applying a CNN backbone on the input image. The CNN backbone is shared with the detector in our instance segmentation pipeline, which will be discussed later. The image feature $F(\mathbf{x}_i)$ is computed using the bilinear interpolation at the vertex coordinate $\mathbf{x}_i$. The

- N vertices {x_i | i = 1, … , N }
- input feature f_i for a vertex x_i = [F(x_i) ; x_i]

- feature maps F ← CNN backbone on the input image
  - F(x_i) ← bilinear interpolation at vertex coordinate x_i
    - torch.nn.grid_sample (interpolation 기법 중 하나)
  - x_i : to encode the spatial relationship among contour vertices

- subtract each dimension of x_i by the minimum value over all vertices, not to be affected by the translation of the contour in the image.
  - transformation = translation + rotation

3. Circular convolution → periodic 1D conv



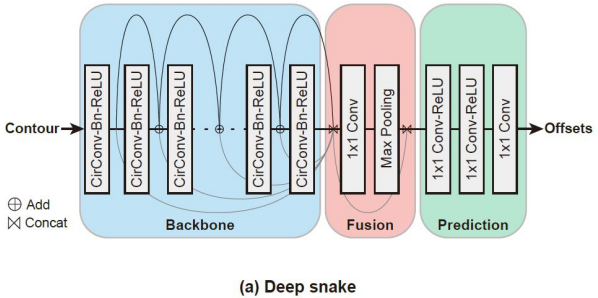
1. Blue nodes are features defined on a contour.

$$(f_N)_i \triangleq \sum_{j=-\infty}^{\infty} f_{i-jN}$$

2. Yellow nodes are the kernel function, and green nodes are the output features.

The circular convolution is

$$(f_N * k)_i = \sum_{j=-r}^{r} (f_N)_{i+j} k_j$$



(a) Deep snake

- where k : [-r, r] → R^D is a learnable kernel function
- and * is the standard convolution

3. Two-stage pipeline for instance segmentations

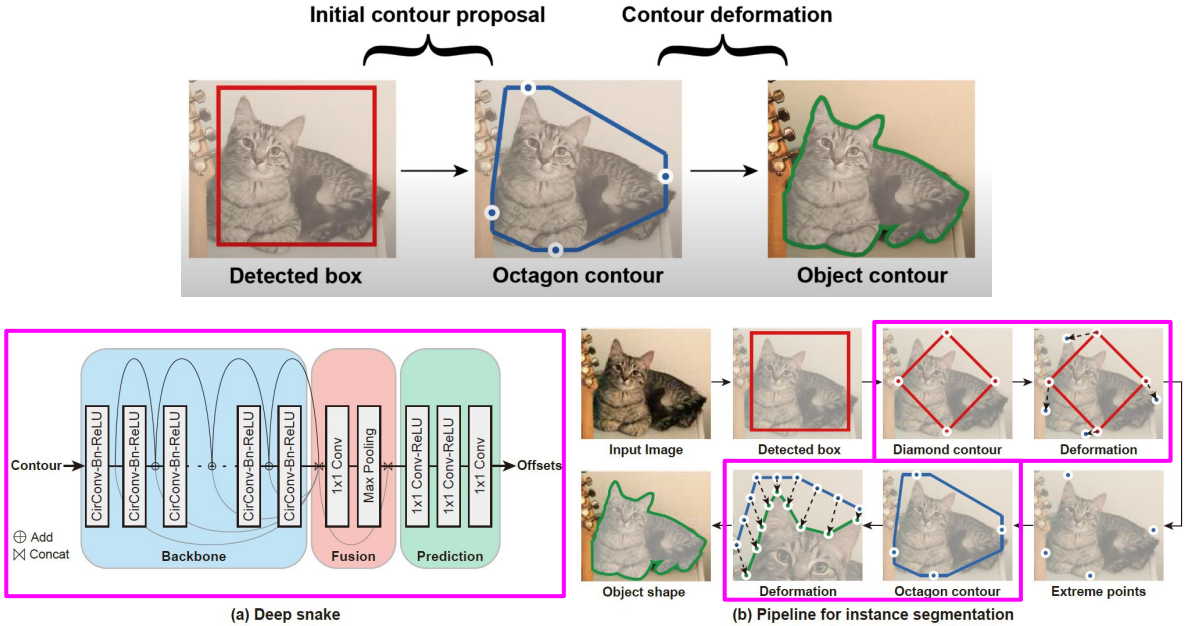


Figure 3. **Proposed contour-based model for instance segmentation.** (a) Deep snake consists of three parts: a backbone, a fusion block, and a prediction head. It takes a contour as input and outputs vertex-wise offsets to deform the contour. (b) Based on deep snake, we propose a two-stage pipeline for instance segmentation: initial contour proposal and contour deformation. The box proposed by the detector gives a diamond contour, whose four vertices are then shifted to object extreme points by deep snake. An octagon is constructed based on the extreme points. Taking the octagon as the initial contour, deep snake iteratively deforms it to match the object boundary.

Stage 1 : Initial contour proposal

1. input image
2. - **Detector ->** x^(ex)
3. connect center points from 2 → x^(bb)
   a. uniformly upsampled to 40points
4. - **Deep snake ->** 4 offsets

Stage 2 : Contour deformation

5. offsets from 4 → new x^(ex)
6. construct octagon contour
   a. uniformly sample N(=128) points
7. - **Deep snake ->** N offsets
8. object shape

3. Multi-component detection for fragmented components

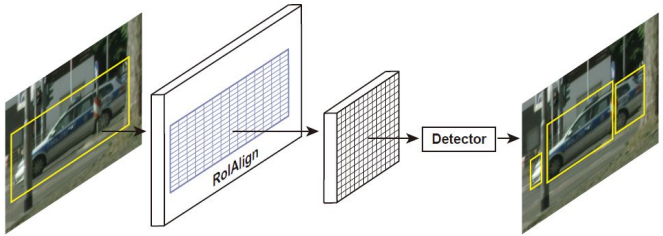
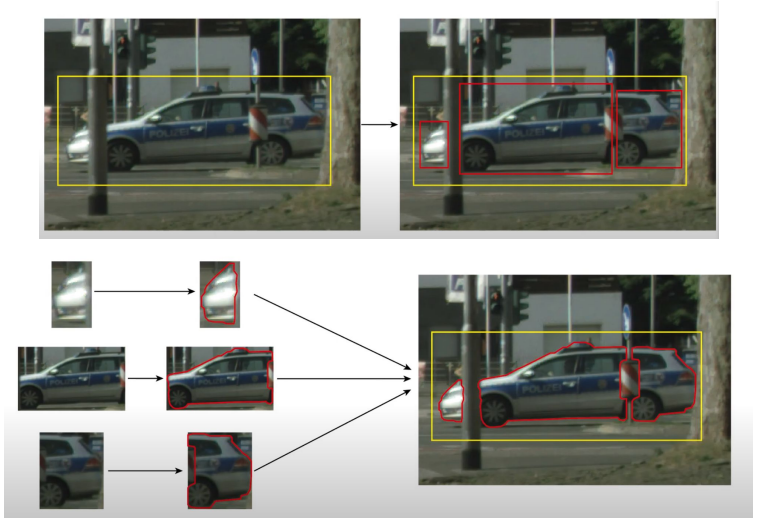


Figure 4. **Multi-component detection.** Given an object box, we perform RoIAlign to obtain the feature map and use a detector to detect the component boxes.



- 한 object가 다른 사물에 의해서 잘렸을 경우, RoIAlign 을 통해 피쳐맵을 뽑고,
- 각각 detection, deep snake를 한 이후 합침.
- 하지만 Cityscapes 데이터셋 제외, 다른 데이터셋은 거의 적용하지 않음.

Deep snake의 장점

Circular convolution

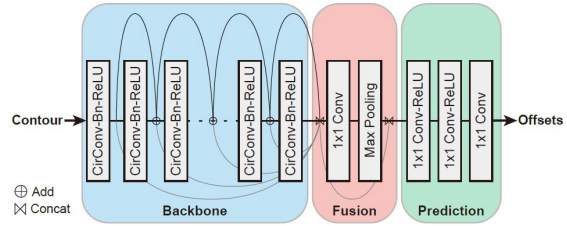
- better exploits the circular structure of the contour than the generic GCN.

Object-level structured prediction

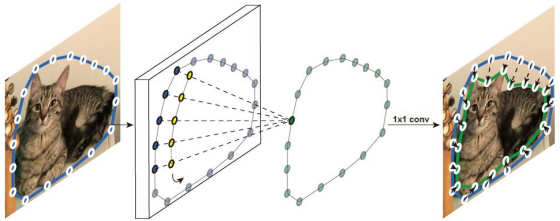
- the offset prediction at a vertex depends on other vertices of the same contour.
- therefore, the model will predict a more reasonable offset for a vertex located far from the object.
    - Standard CNNs may have difficulty in this case, as the regressed vector field may drive this vertex to another object which is closer.

- 따라서 object에서 먼 vertex에 대한 offset을 기존 CNN보다 잘 예측할 수 있음.
- 기존 CNN의 regressed vector field는 object 1에 해당한 vertex라도 더 가까운 object 2가 있다면 해당 vertex를 object2에 둘 수 있음.

3. Network architecture



(a) Deep snake

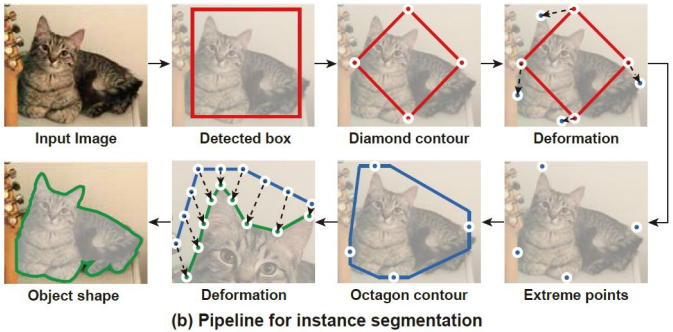


(a) Initial contour (b) Feature learning on the contour (c) Offsets

Figure 1. **The basic idea of deep snake.** Given an initial contour,

- Backbone
  - 8 layers of [Circular conv + Batch norm + ReLU]
  - residual skip connections for all layers
  - → feature for each vertex
- Fusion block
  - aims to fuse the information across all contour points at multiple scales.
  - 1. concatenates features from all layers in the backbone
  - 2. 1x1 conv
  - 3. Max pooling
  - → *fused_feature*
- Prediction
  - input : [feature of each vertex ; *fused_feature*]
  - three 1 x 1 convolution layers
  - output: vertex-wise offsets

4. Experiments: Implementation details



(b) Pipeline for instance segmentation

- Training strategy : smooth L1 loss in Fast R-CNN
  - for extreme point prediction : **Stage 1**

$$L_{ex} = \frac{1}{4}\sum_{i=1}^{4} \ell_1(\tilde{\mathbf{x}}_i^{ex} - \mathbf{x}_i^{ex}), \tag{3}$$

  - for iterative contour deformation : **Stage 2**
    - where ~x_i is the deformed contour point
    - x^(gt)_i is the ground-truth boundary point

$$L_{iter} = \frac{1}{N}\sum_{i=1}^{N} \ell_1(\tilde{\mathbf{x}}_i - \mathbf{x}_i^{gt}), \tag{4}$$

- Detector : *CenterNet*
  - 빠르면서 성능이 좋기 때문에 선택한 것으로 보임.
  - H x W x C feature map
  - → class-agnostic CenterNet
  - → H x W x **1** tensor representing the component center, and H x W x **2** tensor representing the box size.

4. Implementation details on each dataset

Cityscapes

- Multi-component detection tested at a single resolution of 1216x2432.
- No testing tricks are used.
- The detector is first trained alone for 140 epochs, and the learning rate starts from 1e-4 and drops by half at 80, 120 epochs.
- Then the detection and snake branches are trained end-to-end for 200 epochs, and the learning rate starts from 1e-4 and drops by half at 80, 120, 150 epochs.

KINS

- We train the detector and snake end-to-end for 150 epochs. The learning rate starts from 1e-4 and decays with 0.5 and 0.1 at 80 and 120 epochs, respectively.
- We perform multi-scale training and test the model at a single resolution of 768 2496.

SBD

- Since annotations of objects on SBD are mostly single-component, the multi-component detection strategy is not adopted. For fragmented instances, our approach detects their components separately instead of detecting the whole object.
- We train the detection and snake branches end-to-end for 150 epochs with multi-scale data augmentation. The learning rate starts from 1e-4 and drops by half at 80 and 120 epochs. The network is tested at a single scale of 512 512.

COCO

- the multi-component detection strategy is not adopted. The network is trained with multi-scale data augmentation and tested at the original image resolution without tricks (e.g., flip augmentation).
- The detection and snake branches are trained end-to-end for 160 epochs, where the detector is initialized with the pretrained model released by [44]. The learning rate starts from 1e-4 and drops by half at
- 80 and 120 epochs.

4.2. Dataset

1. Cityscapes
2. KINS
3. SBD
4. COCO

KINS

- aims to recover complete instance shapes even under occlusion

SBD

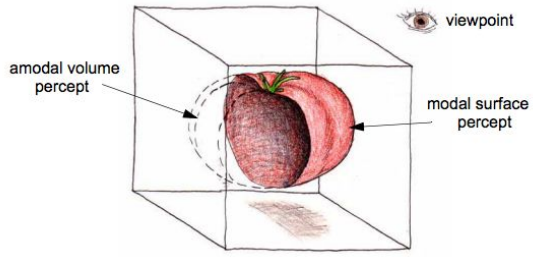
- re-annotates 11,355 images from the PASCAL VOC

(c)

(a)

(b)

Figure 1. Images in the KINS dataset are densely annotated with object segments and contain relative occlusion order. (a) A sample image, (b) the *amodal* pixel-level annotation of instances, and (c) relative occlusion order of instances. Darker color means farther instances in the cluster.



Amodal Instance Segmentation with KINS Dataset, Qi et al., CVPR'19

4.3 Ablation studies

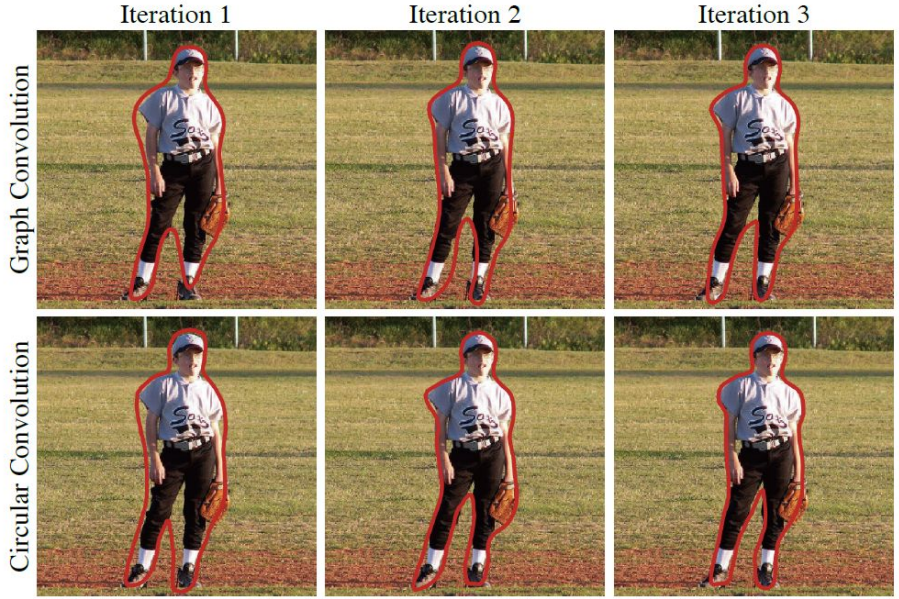


Figure 5. **Comparison between graph convolution (top) and circular convolution (bottom) on SBD.** The result of circular convolution with two iterations is visually better than that of graph convolution with three iterations.

4.3 Ablation studies

	AP_{vol}	AP_{50}	AP_{70}
Baseline	50.9	58.8	43.5
+ Architecture	52.3	59.7	46.0
+ Initial proposal	53.6	61.1	47.6
+ Circular convolution	**54.4**	**62.1**	**48.3**

Table 1. **Ablation studies on SBD val set .** The baseline is a direct combination of Curve-gcn [25] and CenterNet [44]. The second model reserves the graph convolution and replaces the network architecture with our proposed one, which yields 1.4 AP_{vol} improvement. Then we add the initial contour proposal before contour deformation, which improves AP_{vol} by 1.3. The fourth row shows that replacing graph convolution with circular convolution further yields 0.8 AP_{vol} improvement.

	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 5
Graph conv	50.2	51.5	53.6	52.2	51.6
Circular conv	50.6	54.2	**54.4**	54.0	53.2

Table 2. **Results of models with different convolution operators and different iterations** on SBD in terms of the AP_{vol} metric. Circular convolution outperforms graph convolution across all inference iterations. Furthermore, circular convolution with two iterations outperforms graph convolution with three iterations by 0.6 AP, indicating a stronger deforming ability. We also find that adding more iterations does not necessarily improve the performance, which shows that it might be harder to train the network with more iterations.

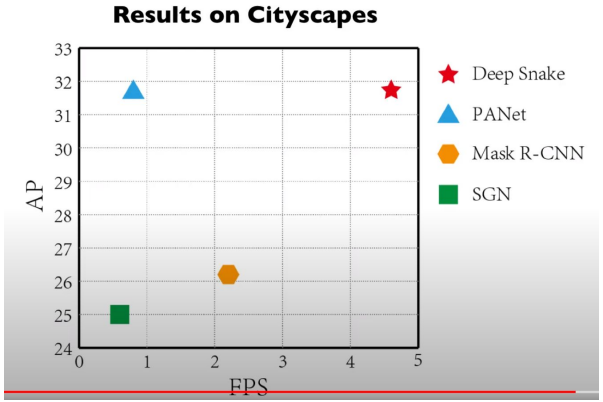
Table 1

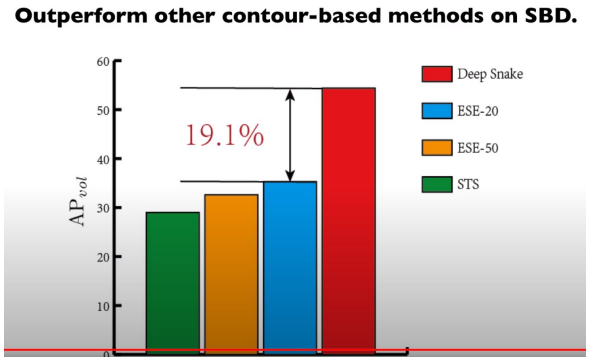
- baseline: combination of **Curve-gcn with CenterNet**
  - object box를 타원형으로 내뱉고 이를 바탕으로 Graph-ResNet 이 deformation 수행.
  - baseline의 경우 contour 를 graph로 처리함.
- 1 appending a global fusion block
  - before the prediction head.
- 2 proposal step generates an octagon initialization
  - by predicting four object extreme points
  - detection error 보완
  - object 에 더 가까이
- 3. graph convolution을 circular convolution으로 바꿨을 때

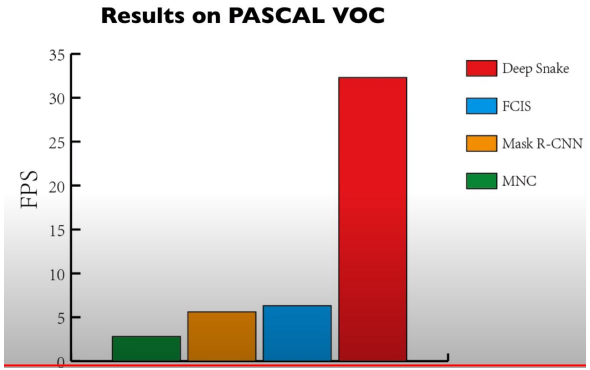
Table 2

- Circular의 두번째 iter가 graph conv의 3번째 iter 보다 0.6 AP_vol 높음.

4.4 Comparison with SOTAs







4.4. Results on AP, fps

	training data	fps	AP [val]	AP	AP_{50}	person	rider	car	truck	bus	train	mcycle	bicycle
SGN [26]	fine + coarse	0.6	29.2	25.0	44.9	21.8	20.1	39.4	24.8	33.2	30.8	17.7	12.4
PolygonRNN++ [1]	fine	-	-	25.5	45.5	29.4	21.8	48.3	21.1	32.3	23.7	13.6	13.6
Mask R-CNN [18]	fine	2.2	31.5	26.2	49.9	30.5	23.7	46.9	22.8	32.2	18.6	19.1	16.0
GMIS [28]	fine + coarse	-	-	27.6	49.6	29.3	24.1	42.7	25.4	37.2	**32.9**	17.6	11.9
Spatial [31]	fine	11	-	27.6	50.9	34.5	26.1	52.4	21.7	31.2	16.4	20.1	18.9
PANet [27]	fine	<1	36.5	**31.8**	57.1	36.8	**30.4**	54.8	27.0	36.3	25.5	**22.6**	**20.8**
Deep snake	fine	4.6	**37.4**	31.7	**58.4**	**37.2**	27.0	**56.0**	**29.5**	**40.5**	28.2	19.0	16.4

Table 3. **Results on Cityscapes val ("AP [val]" column) and test (remaining columns) sets.** Our approach achieves the state-of-the-art performance, which outperforms PANet [27] by 0.9 AP on the val set and 1.3 AP_{50} on the test set. In terms of the inference speed, our approach is approximately five times faster than PANet. The timing results of other methods were obtained from [31].

	detection	amodal seg	inmodal seg
MNC [9]	20.9	18.5	16.1
FCIS [23]	25.6	23.5	20.8
ORCNN [11]	30.9	29.0	26.4
Mask R-CNN [18]	31.1	29.2	×
Mask R-CNN [18]	31.3	29.3	26.6
PANet [27]	32.3	30.4	27.6
Deep snake	**32.8**	**31.3**	×

Table 4. **Results on KINS test set in terms of the AP metric.** The amodal bounding box is used as the ground truth in the detection task. × means no such output in the corresponding method.

	AP_{vol}	AP_{50}	AP_{70}
STS [20]	29.0	30.0	6.5
ESE-50 [42]	32.6	39.1	10.5
ESE-20 [42]	35.3	40.7	12.1
Deep snake	**54.4**	**62.1**	**48.3**

Table 5. **Results on SBD val set.** Our approach outperforms other contour-based methods by a large margin. The improvement increases with the IoU threshold: 21.4 in AP_{50} and 36.2 in AP_{70}.

	YOLACT [3]	ESE [42]	OURS
val (segm AP)	29.9	21.6	**30.5**
test-dev (segm AP)	29.8	-	**30.3**

Table 6. **Comparison with other real-time methods** on COCO.

method	MNC	FCIS	MS	STS	ESE	OURS
time (ms)	360	160	180	27	26	31
fps	2.8	6.3	5.6	37.0	38.5	32.3

Table 7. **Running time on the PASCAL VOC dataset.** "MS" represents Mask R-CNN [18] and "OURS" represents our approach. The last three methods are contour-based methods.

3. SOTA on Cityscapes.

- 성능 + 속도가 빠름!
- frame per seconds 에서 잘함
- 5x faster than PANet

4. KINS test set 에서도 잘함

5. 다른 contour-based 와 비교하면 큰 차이로 잘함

6. real-time 모델과 비교했을 때 segmentation 성능도 좋음

7. contour-based method 가 빠름

- Ours = 31 (ms)
  - CenterNet 18.4
  - initialization 3.1
  - each iter of deformation 3.3

4.4. Qualitative results

Figure 6. **Qualitative results on Cityscapes test and KINS test sets.** The first two rows show the results on Cityscapes, and the last row lists the results on KINS. Note that the results on KINS are for amodal instance segmentation.

4.4. Qualitative results

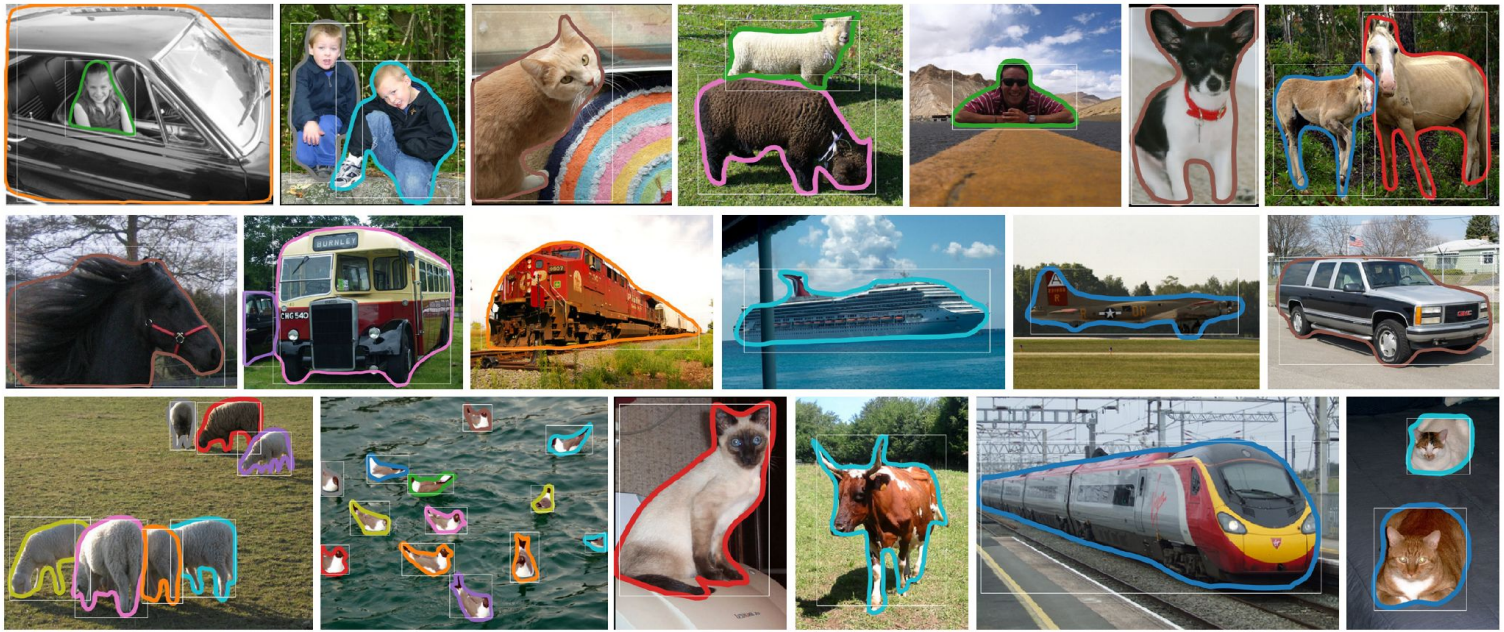


Figure 7. **Qualitative results on SBD val set.** Our approach handles errors in object localization in most cases. For example, in the first image, although the detected box doesn't fully enclose the car, our approach recovers the complete car shape. Zoom in for details.

5. Conclusion

1. 클래식한 이미지 프로세싱 기법을 기반으로, 노벨티 있는 알고리즘을 만들어냄
2. scalable vector graphic(SVG), 3D 등 vertex, contour를 활용하는 경우 유용할 것으로 보임
3. 실제로 Im2Vec (CVPR'21 oral) 의 경우 circular conv를 그대로 활용함
4. 리젝됐지만 성능이 좋고 목적에 부합한 모델이라면 (CenterNet), 활용가치 있음

Thanks : >