Lane Detection

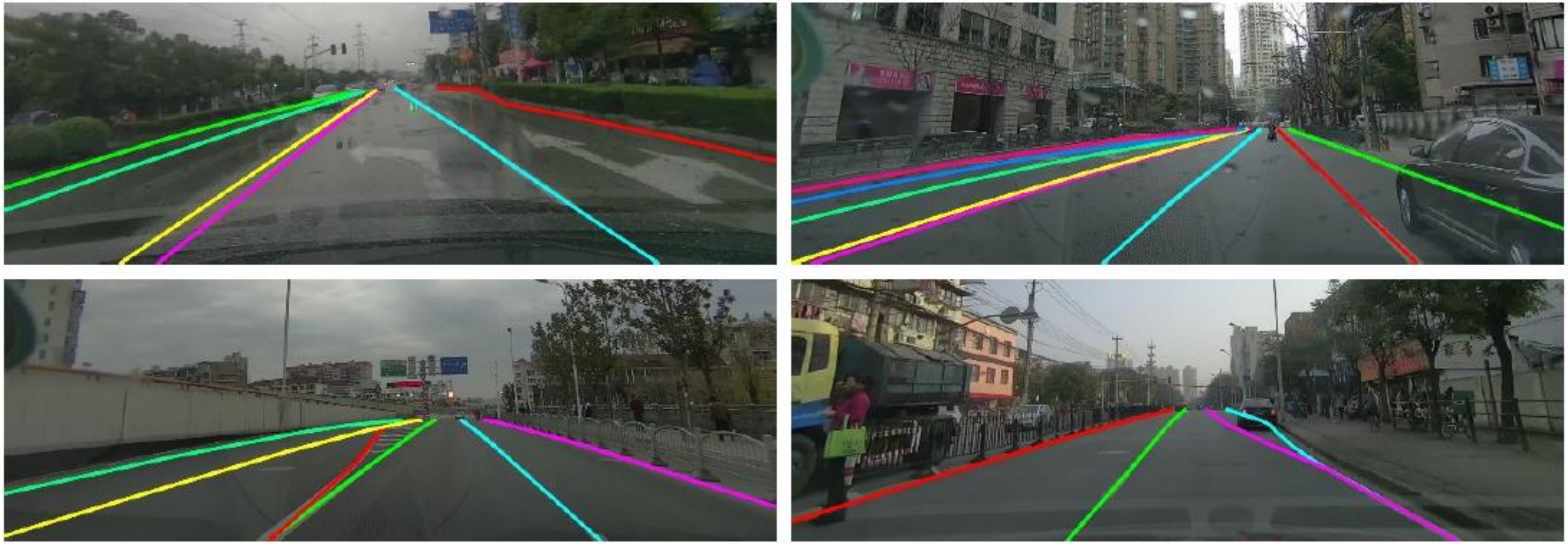
Chen Feng

2021.07.29

Content

- Task Introduction
- Segmentation-based
- Anchor-based
- Parameter-based
- Row-wise
- Conclusion and Idea
- Reference

Task Introduction

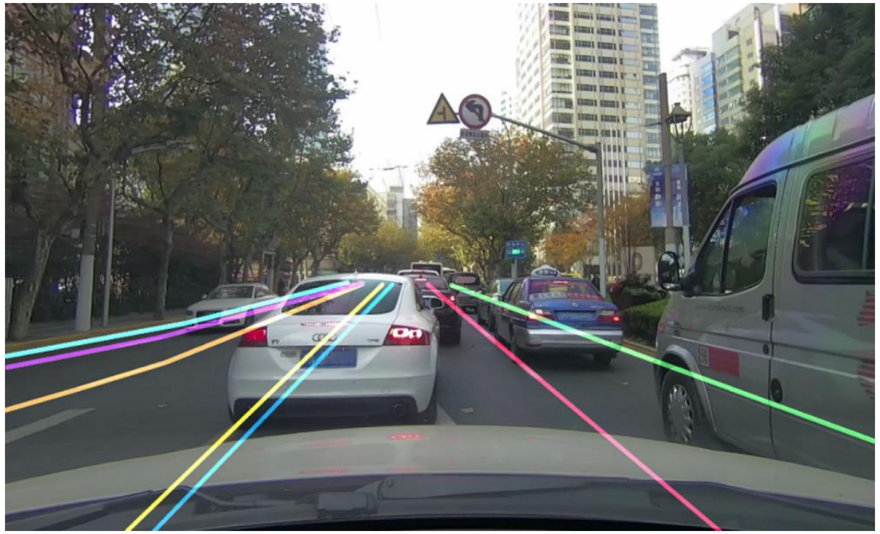


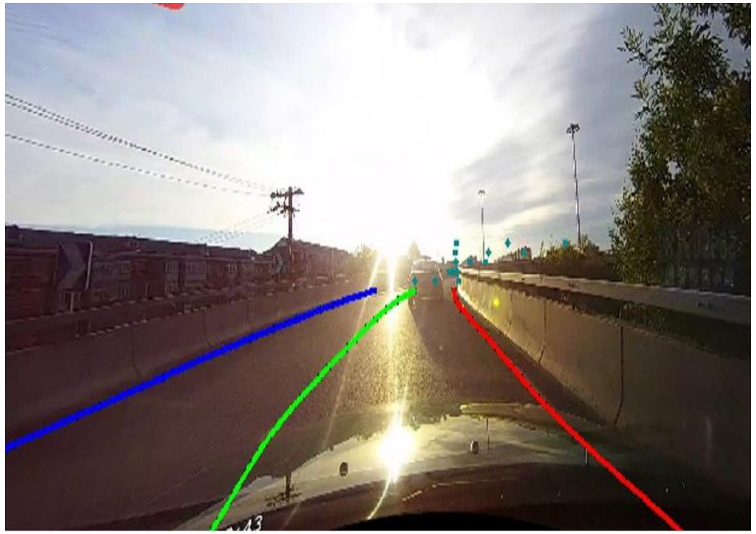
Input: image or video from vehicle perspective

Output: flexible quantity and precise lanes

Task objectives

- Classify foreground (lanes) and background precisely
- Detect lanes flexibly
- Work in complex cases, e.g. occlusion, dense, fork, curve, no-flatten, dazzle light, night, ambiguous scene





Dataset

- CULane: CULane Intro
- CurveLanes: CurveLanes Intro

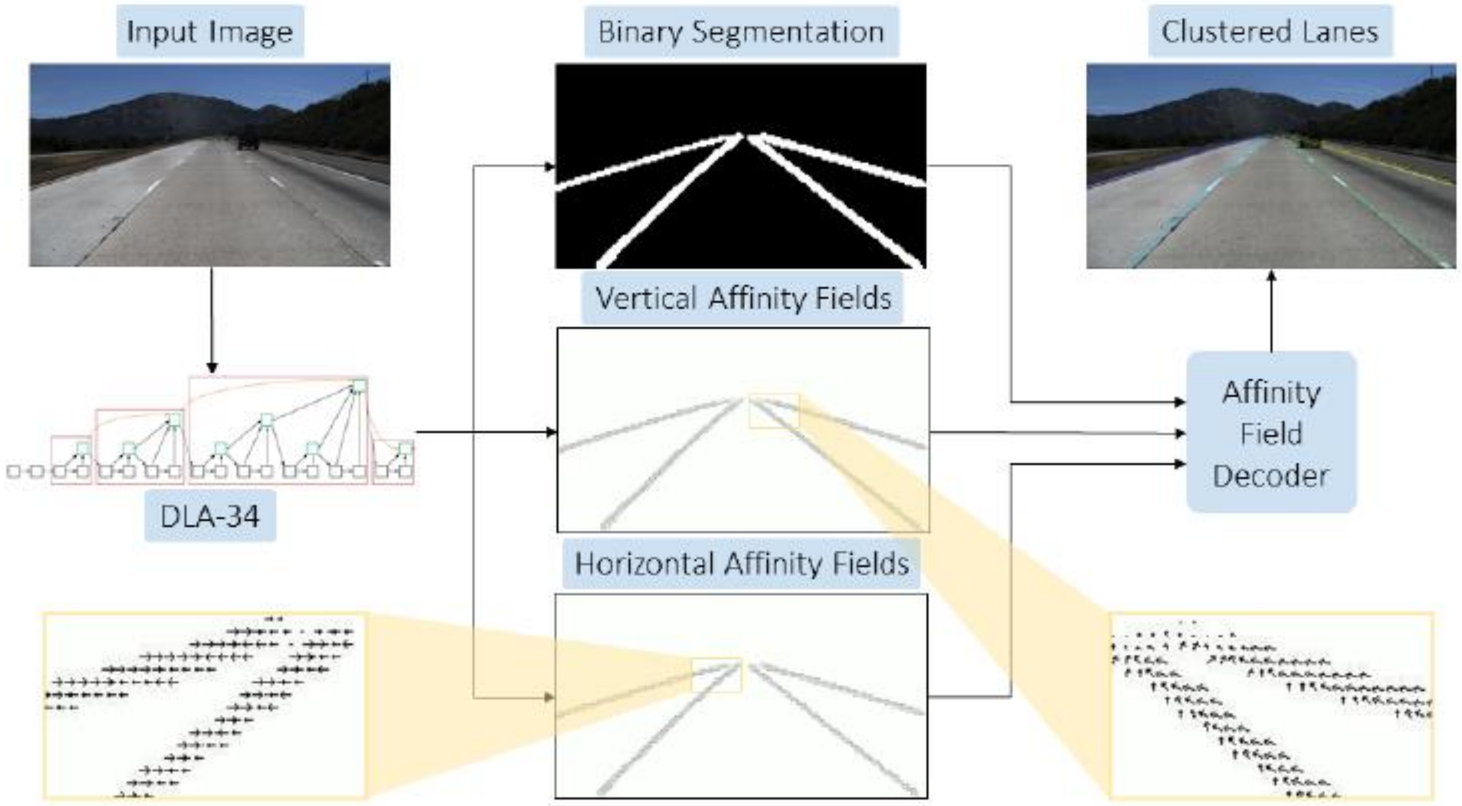
Segmentation-based

- Method: (2-stage) segmentation + clustering, as an instance segmentation task

- Problem: 1) pixel-wise → computational cost and slow;
  2) clustering needs hyper-param adjustment;
  3) local-based, occlusion and no visual clue (✕)

- Opt direction: 1) segmentation stage → feature interactive → aggregate global info;
  2) hyperparam-free clustering

Segmentation-based → LaneAF



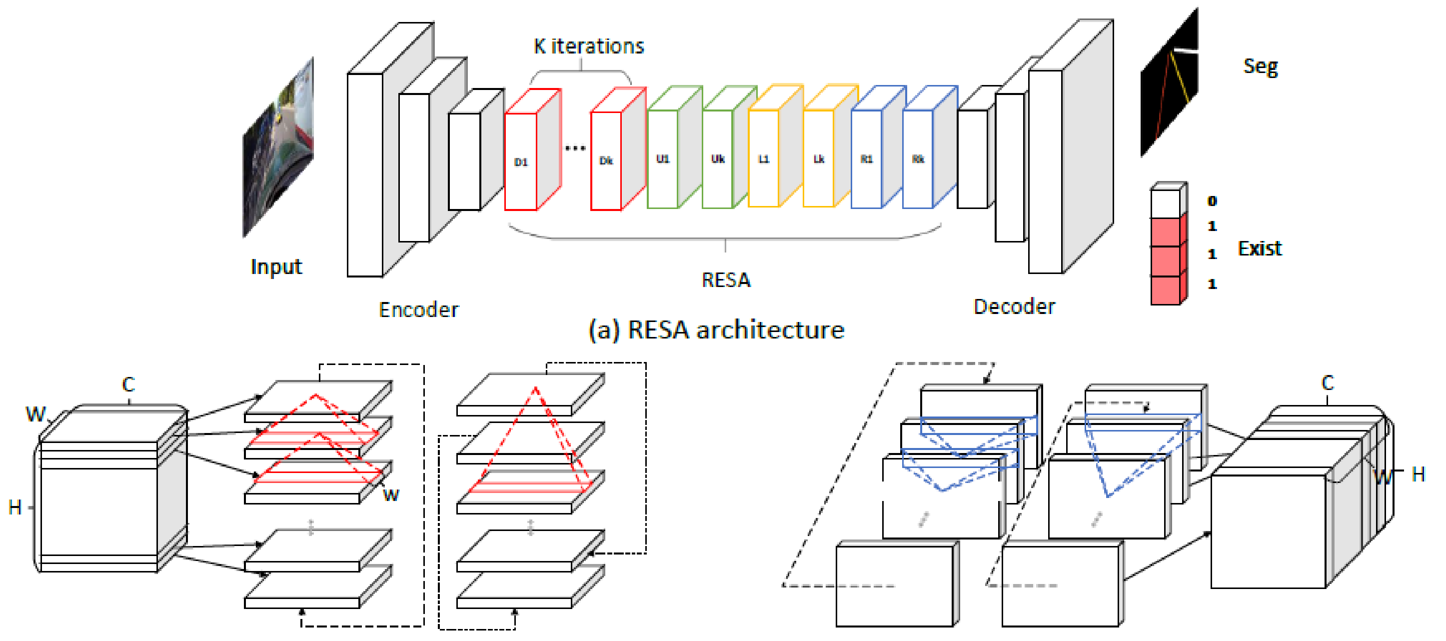
Predict 2 offset vectors on pixels in foreground: **HAF** and **VAF**

Clustering in embedding space → local to global by decoder

Opt: hyper-param-free clustering in an embedding space

[details]

Segmentation-based → RESA



(a) RESA architecture
(b) RESA_U
(c) RESA_R

Add **feature aggregator** between encoder and decoder → fuse global info

1. Multi strides: 1 and 2
2. Addition res: conv + element-wise addition to update ori_feature
3. Four directions: top-down (inverse), left to right (inverse)

Opt: strengthen interaction of local feature

detail

Anchor-based

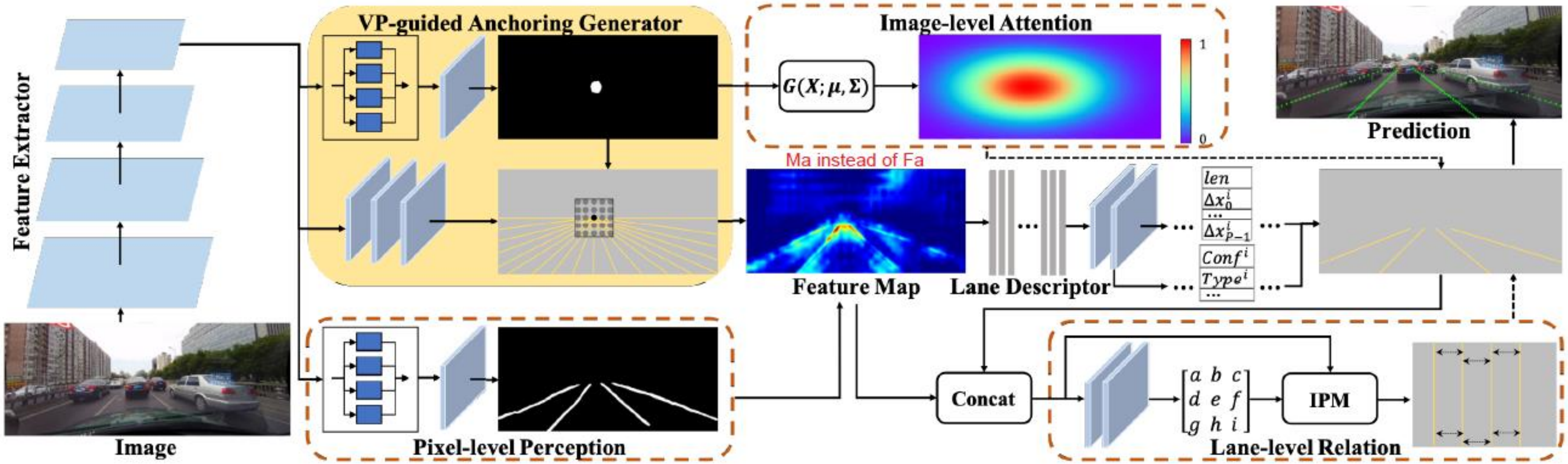
- Method: object detection task, thin and long anchor with offset map

- Problem: 1) dense anchors search → computational cost;
  2) precision (↓) with curvature (↑);
  3) with no global info for occlusion

- Opt direction: 1) global interaction of anchors;
  2) quantity of anchors (↓)

Anchor-based → SGNet

Opt: reduce anchor quantity and introduce prior geometry



Predict **vanishing point** (VP) → parallel constraint → 180° produce anchors [details]

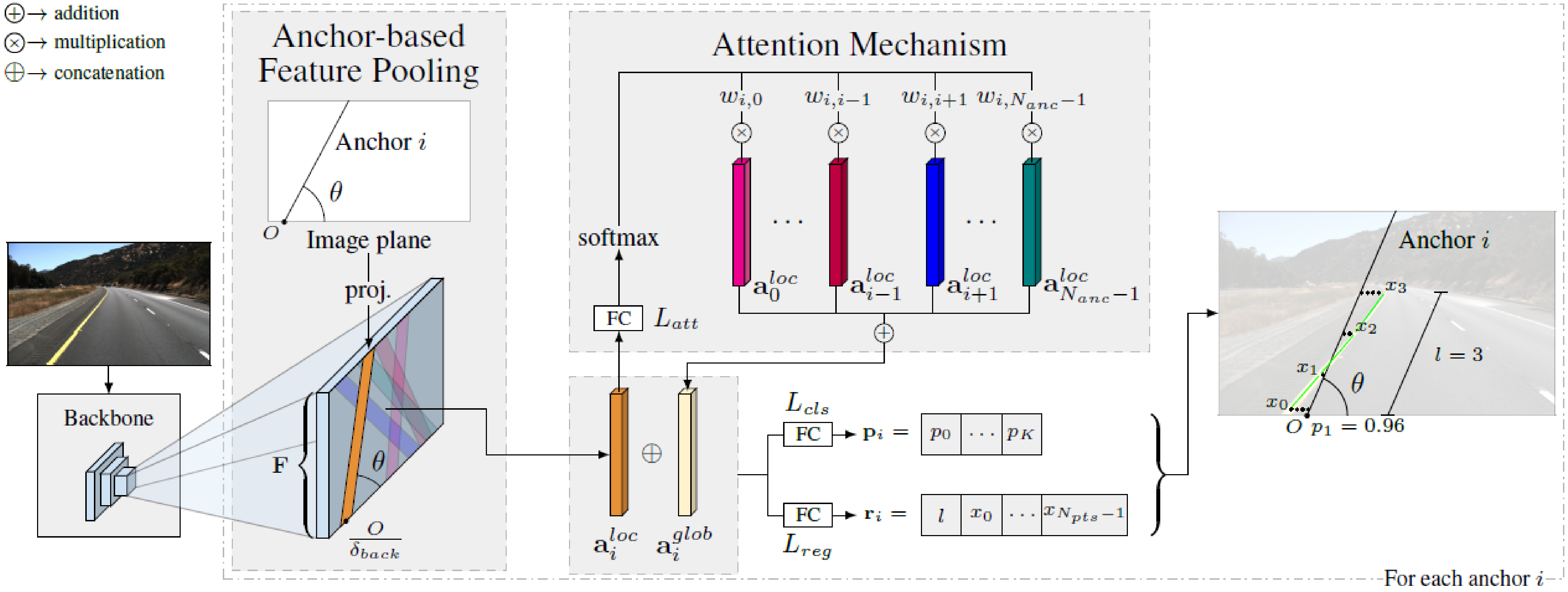
Fuse pixel-level perception → new feature map → lane descriptor ∈ $R^{H\times C}$ → anchor prediction

Inverse H Net supervision parallel lanes → lane-level relation

Attention mechanism for faraway pixels → image-level constraint

Anchor-based → LaneATT

Opt: anchor global interaction



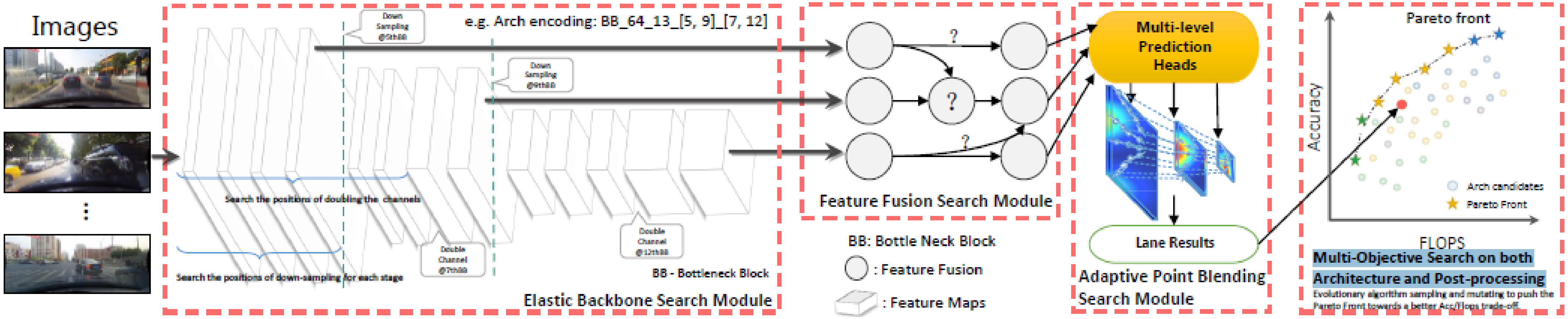
Anchors' starts from left, down and right → 3 attributes: start s, end e, direction a + offset map

AFP → anchor corresponding feature vector a_i^{loc} → attention → global features →(FC) cls + reg

[details]

Anchor-based → CurveLane-NAS

Opt: NAS-based methods to find optimal net by Pareto front

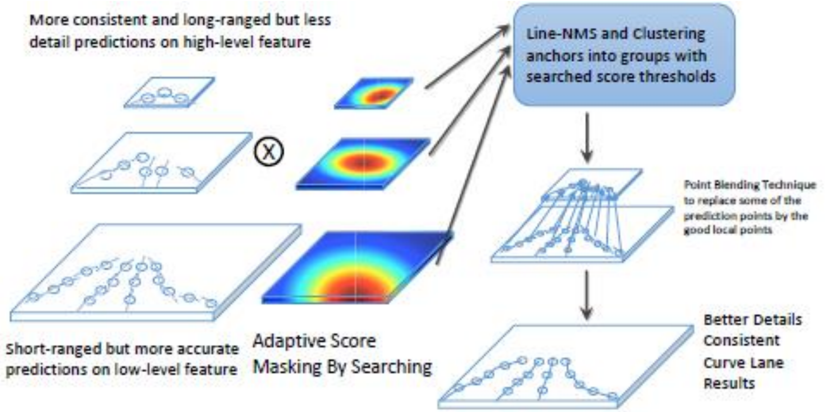


Build 3 search space based on NAS

1. Network structure NAS, like down-sampling position, doubling channel position etc.

2. Feature fusion search

3. Good local pts replace modification



[details]

Parameter-based

- Method: see a lane as multi-spline curve, actually **cubic** best → high-semantic info for 4 parameters
- Problem: upper limit too low, performance (✕)
- Representation model: 1) PolyLaneNet;
  2) LSTR (Transformer)

Pending…

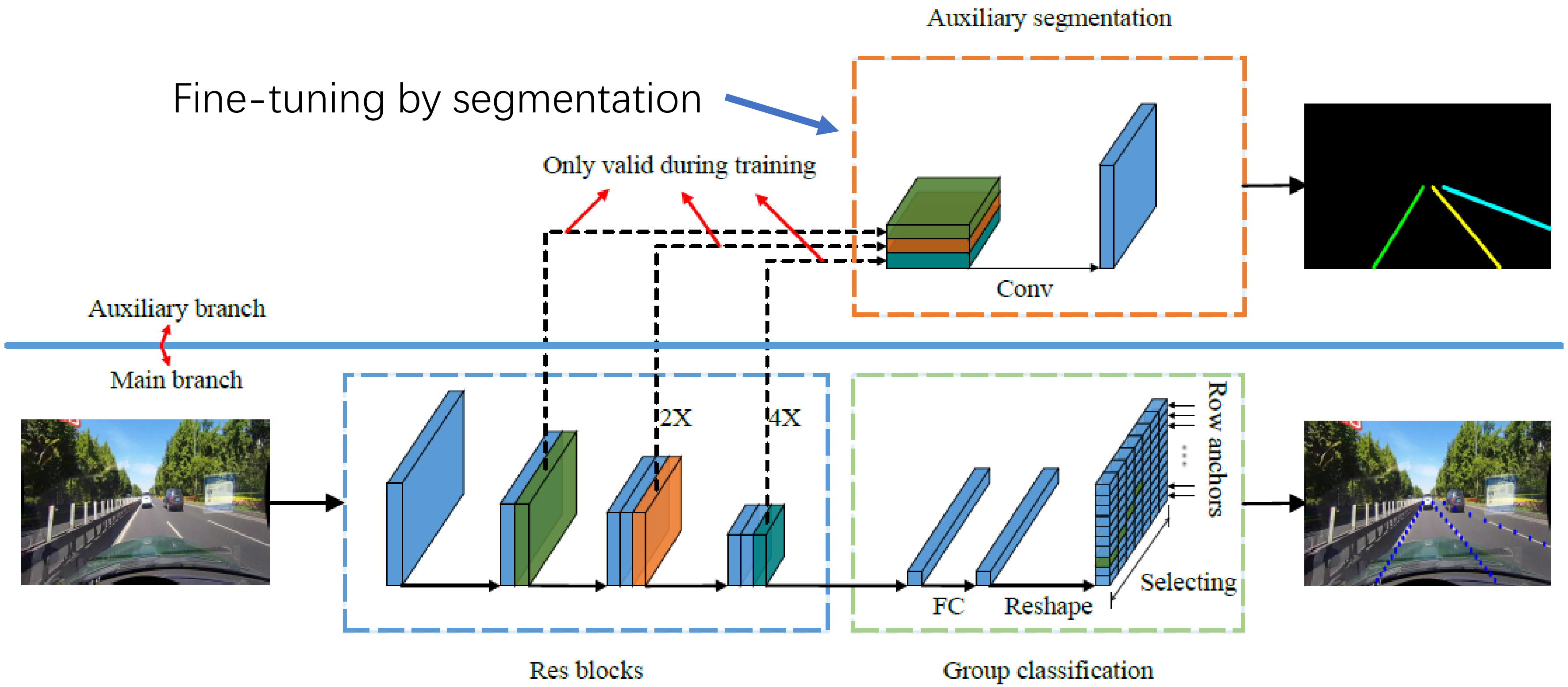
Row-wise (SOTA)

- Method: pixel-wise search → row-wise search → cost (↓)
  utilize continuous property of lanes → global info

- Problem: 1) need instance/channel-wise flexibly;
  2) row-wise → grid deviation → precision (↓);

- Opt direction: 1) predict quantity of lanes flexibly
  2) add grid offset map
  3) top-down → bottom-up

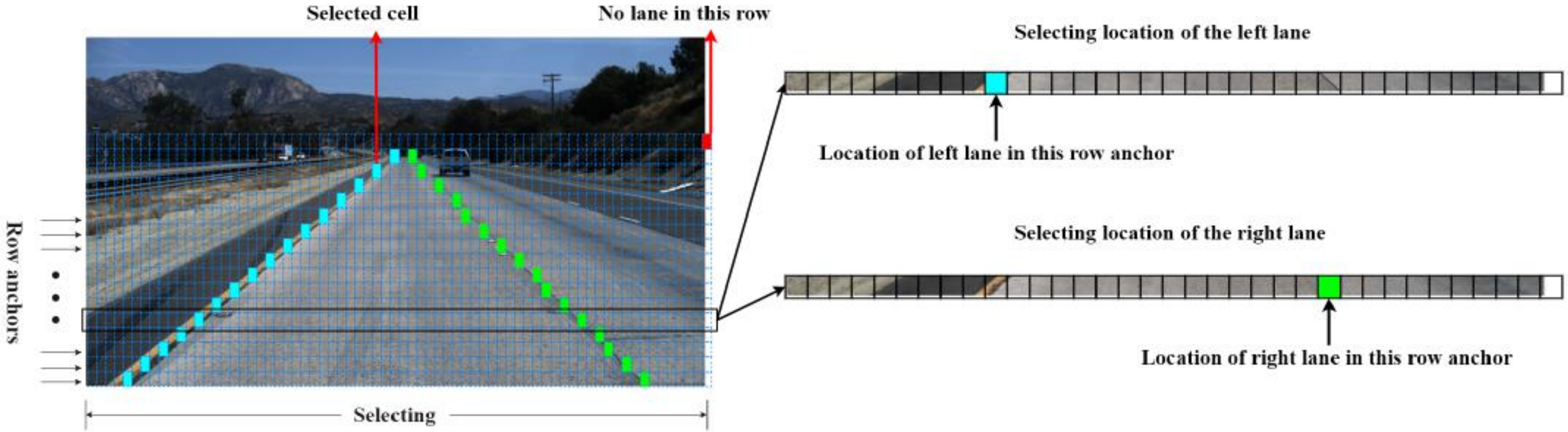
Row-wise → Ultrafast (baseline)



Output feature map $\in R^{H \times W \times C}$, C -- pre-defined hyper-parameter (**5**) for quantity of lanes **fixed !!!**

Channel-wise (lane-wise) → row-wise search for pixels area of lanes

How to separate lanes to C?

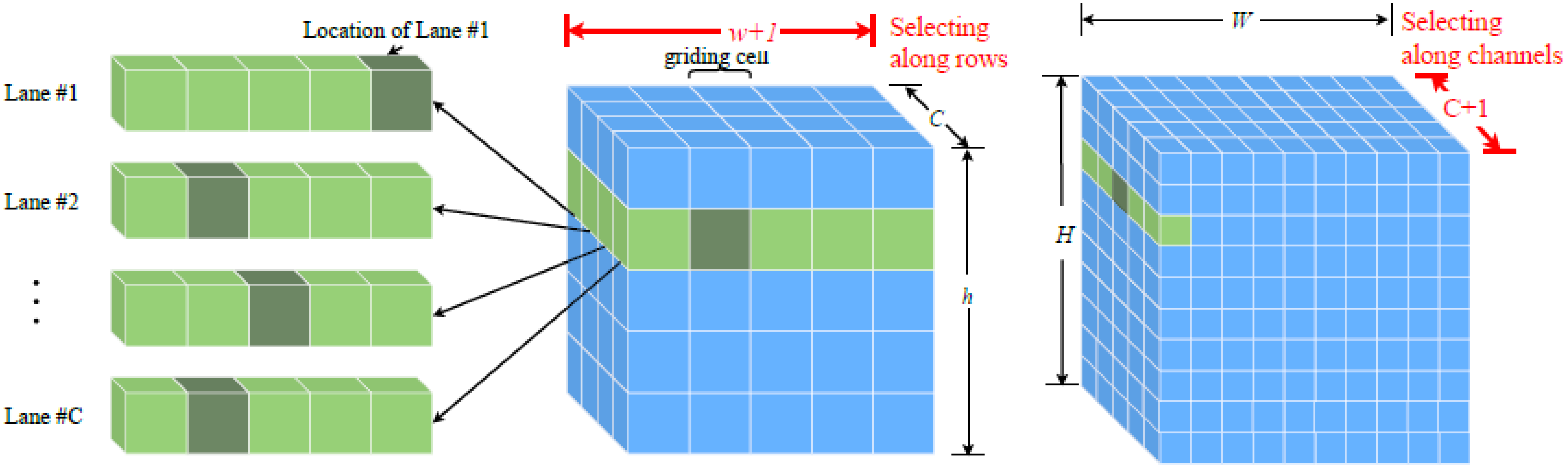


Set a channel, $h = 10, w = \frac{W}{100}$ → feature map grid on → $R^{h \times w \times C}$

Train a classifier : $P_{i,j,:} = f_{i,j}(X), s.t. i \in [1, C], j \in [1, h]$, where P is the $w + 1$ dims vector

$$L_{cls} = \sum_{i=1}^{C} \sum_{j=1}^{h} L_{CE}(P_{i,j,:}, T_{i,j,:}),$$

Why fast?



(a) Our formulation (b) Segmentation

Ultrafast: $C \times h$ classification problems, FLOPs: $C \times h \times (w + 1)$

Segmentation: $H \times W$ classification problems, FLOPs: $H \times W \times (C + 1)$

In summary, pixel-wise search → cell-wise search

Geometry constraint → 2 structural losses

- Continuous property → adjacent rows should be similar

$$L_{sim} = \sum_{i=1}^{C} \sum_{j=1}^{h-1} \|P_{i,j,:} - P_{i,j+1,:}\|_1$$

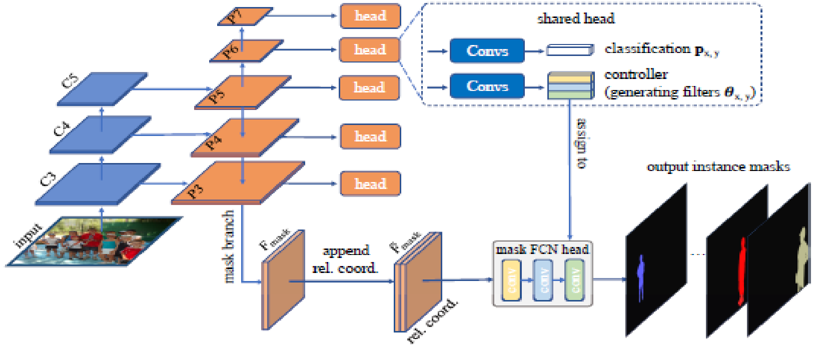
- Obtain location prediction

$$Prob_{i,j,:} = softmax(P_{i,j,1:w}), \ Loc_{i,j} = \sum_{k=1}^{w} k \cdot Prob_{i,j,k}$$

- Lane rigidity → the second difference constraint should be **0**

$$L_{shp} = \sum_{i=1}^{C} \sum_{j=1}^{h-2} \|(Loc_{i,j} - Loc_{i,j+1}) - (Loc_{i,j+1} - Loc_{i,j+2})\|_1,$$

How to solve fixed quantity? CondConv



w/o anchor object detector

FCOS + Controller Head

利用 FCOS

classification head , $H \times W \times C$, $P_{x,y}$ → binary classifier

center-ness head , $H \times W \times 1$, score → predict K instances.

box head , $H \times W \times 4$ (l, r, t, b) + NMS →

l, t, r, b

controller head , N × K . K mask heads

F_{mask} → $\frac{1}{8}$ input

$F_{mask} \in R^{H_m \times W_m \times C_m}$

$F_{mask} \xrightarrow{\text{coord conv}} \tilde{F}_{mask} \in R^{H_m \times W_m \times (C_m + 2)}$

加上 x, y 两个维度信息 (K instances center).

$\tilde{F}_{mask}$ → mask head 1 → instance 1

⋮

→ mask head K → K

mask head

Conv 1 (1×1) —8→ Conv 2 (1×1) —8→ conv 3 (1×1) —1→

10×1×1×8 + 8 (88) + 8×1×1×8 + 8 (72) + 8×1×1×1 + 1 (9) = 169 = N

$$L_{overall} = L_{fcos} + \lambda L_{mask},$$

$$L_{mask}(\{\theta_{x,y}\}) = \frac{1}{N_{pos}} \sum_{x,y} \mathbb{1}_{\{c^*_{x,y}>0\}} L_{dice}(MaskHead(\tilde{F}_{x,y};\theta_{x,y}), M^*_{x,y}), \quad (2)$$

instance prediction $H \times W \times k$

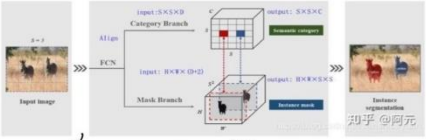
$H \times W \times C$

dice loss (实例与背景差异很大)

CondInst → channel-wise

SOLO v2 → cell-wise

H×W
⇓
S×S grids



SOLO architecture.

backbone
+
FPN

category branch : 语义分割分类器. S×S×C

Mask branch : 实例分割 M: H×W×S×S

稀疏
M is rebundant.

drawback: 1) M — large → cost

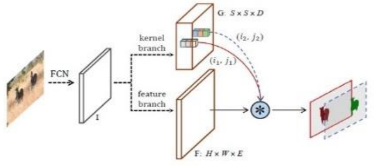
2) FPN → S different × shared

no need

To be more flexible. M = F * G

F – feature
G – filter

SOLO v2 divide Mask branch into { kernel branch. / feature branch.



(b) SOLOv2

kernel branch : S×S (at most) D – dims vector

↓ contain location information.

↓ assign

Kernel : input channel : E

output channel : 1

e.g. kernel : E×3×3×1 → D = 9E

S×S (sparse) only small quantity of S×S

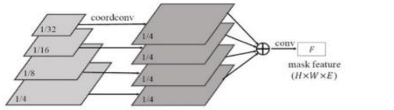


Figure 5 – Unified mask feature branch. Each FPN level (left) is upsampled by convolutions and bilinear upsampling until it reaches 1/4 scale (middle). In the deepest FPN level, we concatenate the x, y coordinates and the original features to encode spatial information. After element-wise summation, a 1×1 convolution is attached to transform to designated output mask feature $F \in \mathbb{R}^{H \times W \times E}$.

Feature branch

input img → FCN-backbone → FPN → { 1/32 → coordconv $\xrightarrow[\text{upsample}]{\text{conv}}$ 1/4

location-aware

1/16 → 1/4

1/8 → 1/4

1/4 → 1/4 }

$F \in \mathbb{R}^{H \times W \times E}$ ← conv Norm ReLU ← element-wise summation ⊕

generate Mask : H×W×E E×1×1×1

F * G_1

...

F * G_K

K << S×S

⇒ $Mask_1$ ⋮ $Mask_K$

Matrix NMS parallel accelerating
instead of Mask NMS
Fast +++...

Mask :

H×W×C

C < K

```
def matrix_nms(scores, masks, method='gauss', sigma=0.5):
    # scores: mask scores in descending order (N)
    # masks: binary masks (NxHxW)
    # method: 'linear' or 'gauss'
    # sigma: std in gaussian method

    # reshape for computation: Nx(HW)
    masks = masks.reshape(N, HxW)
    # pre-compute the IoU matrix: NxN
    intersection = mm(masks, masks.T)
    areas = masks.sum(dim=1).expand(N, N)
    union = areas + areas.T - intersection
    ious = (intersection / union).triu(diagonal=1)

    # max IoU for each: NxN
    ious_cmax = ious.max(0)
    ious_cmax = ious_cmax.expand(N, N).T
    # Matrix NMS, Eqn.(4): NxN
    if method == 'gauss': # gaussian
        decay = exp(-(ious^2 - ious_cmax^2) / sigma)
    else: # linear
        decay = (1 - ious) / (1 - ious_cmax)
    # decay factor: N
    decay = decay.min(dim=0)
    return scores * decay
```

Matrix NMS

Confidence : $decay_j = \min_{\forall s_i > s_j} \frac{f(iou_{i,j})}{f(iou_{\cdot,i})}$ f( ) — Gaussian decay

$s_j = s_j \cdot decay_j$

g granularity

CondLaneNet (Ultrafast++)

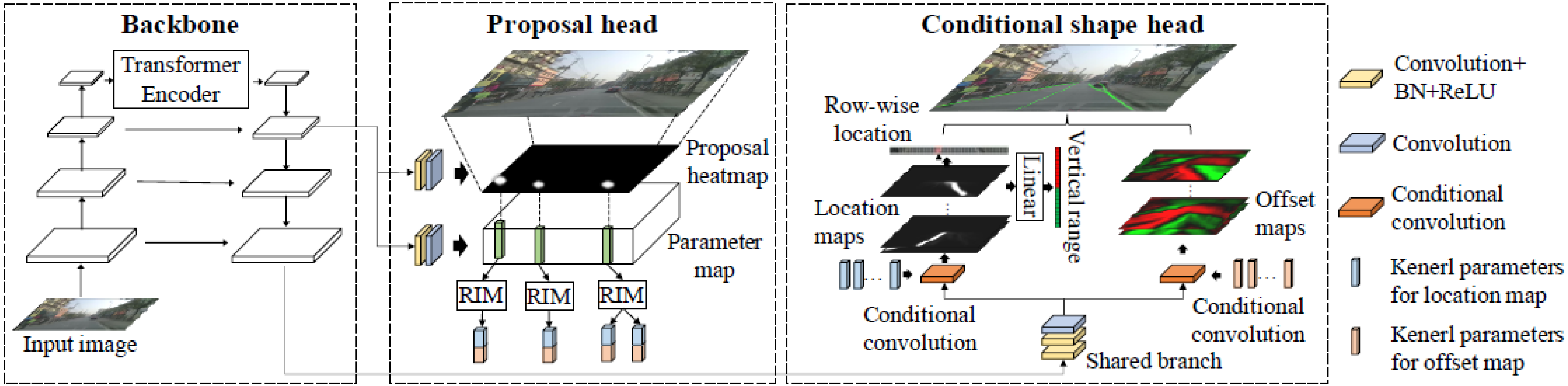
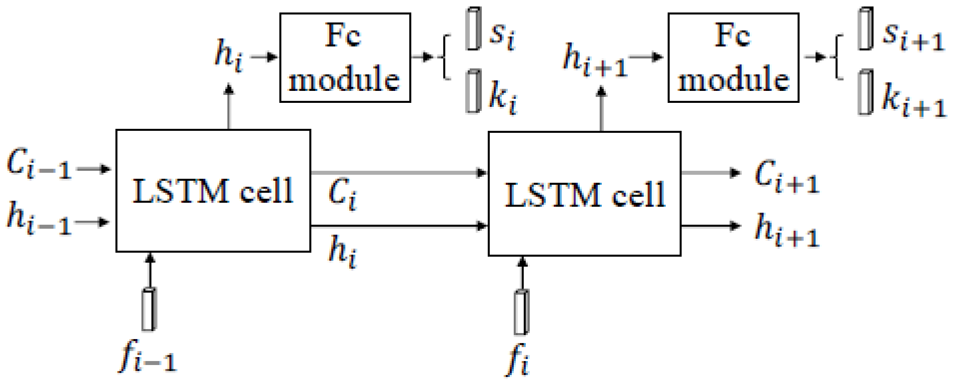


Figure 2. The structure of our CondLaneNet framework. The backbone adopts standard ResNet [8] and FPN [23] for multi-scale feature extraction. The transformer encoder module [37] is added for more efficient context feature extraction. The proposal head is responsible for detecting the proposal points which are located at the start point of the line. Meanwhile, a parameter map that contains the dynamic convolution kernels is predicted. The conditional shape head predicts the row-wise location, the vertical range, and the offset map to describe the shape for each line. To address the cases of dense lines and fork lines, the RIM is designed.

Opt: 1) flexible quantity prediction of lanes ← CondConv instance-wise
2) solve fork lanes case ← RIM module
3) improve precision granularity ← offset map in chosen cell

How to separate fork lanes?



Proposal heatmap → start pts (instead of center in OT task) + corresponding embedding vector f

Core unit: LSTM → state vector (2 dims), s + kernel parameters for location and offset

Inference stage: $\mathbf{argmax}(s) == 0$, return parameters to assign to CondConv kernel

Attention: Code implementation goes different.

```
for idx in range(pos_tensor.size()[0]):
                rnn_feat_input = rnn_params[idx:idx + 1, :]
                rnn_feat_input = rnn_feat_input.reshape(1, -1, 1, 1)
                hidden_h = rnn_feat_input
                hidden_c = rnn_feat_input
                rnn_feat_input = rnn_feat_input.reshape(1, 1, -1, 1, 1)

                if self.zero_hidden_state:
                    hidden_state = None
                else:
                    hidden_state = (hidden_h, hidden_c)
                num_ins_count = 0
                for _ in range(max_rtimes):
                    rnn_out, hidden_state = self.rnn_ceil(
                        inputs=rnn_feat_input,
                        hidden_state=hidden_state,
                        seq_len=1)
                    rnn_out = rnn_out.reshape(1, -1, 1, 1)
                    k_param, state = self.final_fc(rnn_out)
                    k_param = k_param.squeeze(-1).squeeze(-1)
                    state = state.squeeze(-1).squeeze(-1)
                    kernel_params.append(k_param)
                    num_ins_count += 1
                    if torch.argmax(state[0]) == 0:
                        break
                    rnn_feat_input = rnn_out
                    rnn_feat_input = rnn_feat_input.reshape(1, 1, -1, 1, 1)
                num_ins_per_seed.append(num_ins_count)

            num_ins = len(kernel_params)
            kernel_params = torch.cat(kernel_params, 0)
            mask_params = kernel_params[:, :self.num_mask_params]
            reg_params = kernel_params[:, self.num_mask_params:]
```

[code details](#)

Loss

$$\ell_{point} = \frac{-1}{N_p}\sum_{xy}\begin{cases}(1-\hat{P}_{xy})^{\alpha}log(\hat{P}_{xy}) & P_{xy}=1\\(1-P_{xy})^{\beta}(\hat{P}_{xy})^{\alpha}log(1-\hat{P}_{xy}) & otherwise\end{cases}$$

对于proposal point，为了解决背景与目标（车道线）样本数量差异大的问题，采用了focal loss监督。

$$\ell_{row} = \frac{1}{N_v}\sum_{i\in V}|E(\hat{x}_i)-x_i|$$

对于row-wise搜索分类，对每个channel利用每行车道线期望位置进行监督。

$$\ell_{range} = \sum_i(-y^i_{gt}log(v_i)-(1-y^i_{gt})log(1-v_i))$$

对于vertical range， 采用了softmax CE loss监督。

$$\ell_{offset} = \frac{1}{N_\Omega}\sum_{(j,i)\in\Omega}\left|\hat{\delta}_{ij}-\delta_{ij}\right|$$

对于offset，直接用GT offset监督学习。

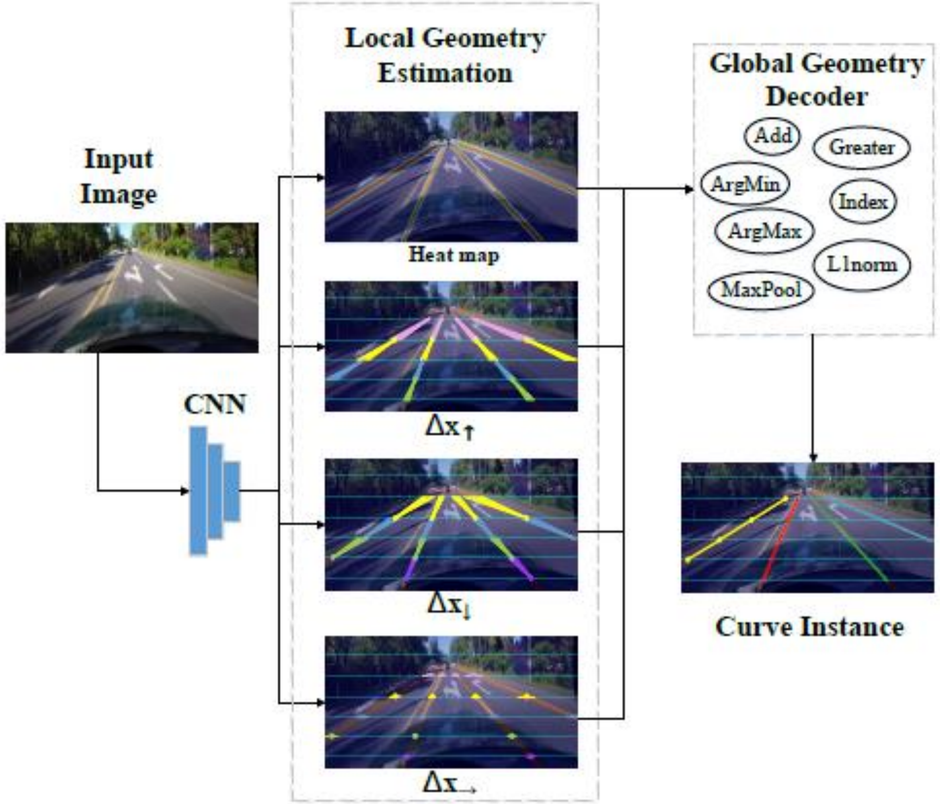
$$\ell_{state} = \frac{1}{N_s}\sum_i -[y_i\cdot log(s_i)+(1-y_i)\cdot log(1-s_i)]$$

reading details

对于RIM出来的state vector s_i ，利用GT监督RIM的学习。

$$\ell_{total} = \ell_{point}+\alpha\ell_{row}+\beta\ell_{range}+\gamma\ell_{offset}+\eta\ell_{state}$$

Row-wise → FOLOLane (bottom-up)



[reading details]

Whole structure: input → feature extraction → heat map+ offset map → global decoder → output

Four heads: 1) global Gaussian response instead of binary classification

2) upward offset prediction map

3) downward offset prediction map

4) horizontal offset prediction map

Global decoder: sampling keypoints row-wisly + offsets → predict lanes

Opt: more concise and elegant with **quasi-SOTA** result

How to train offset map?

$$\boldsymbol{Loss}_{\uparrow}(l) = \frac{1}{|N_{\sigma_g}(l)|}\Sigma_{p\in N_{\sigma_g}(l)}||\hat{\boldsymbol{p}}_{\uparrow}(p) - \varphi(l, f_y(p) - \Delta y)||_1,$$

$$\boldsymbol{Loss}_{\downarrow}(l) = \frac{1}{|N_{\sigma_g}(l)|}\Sigma_{p\in N_{\sigma_g}(l)}||\hat{\boldsymbol{p}}_{\downarrow}(p) - \varphi(l, f_y(p) + \Delta y)||_1,$$

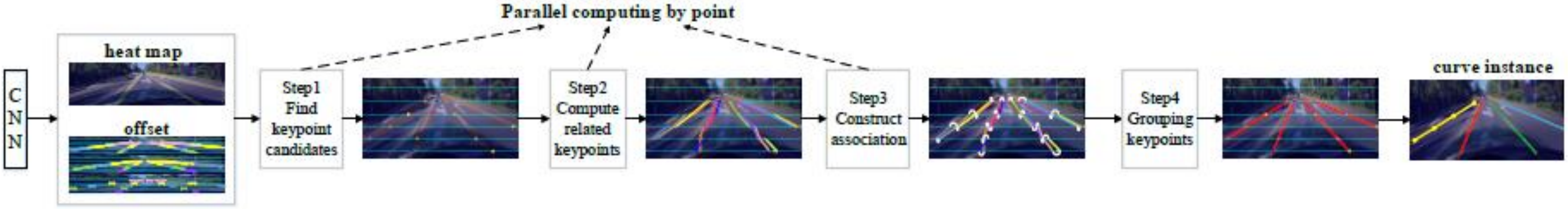
$$\begin{aligned}\boldsymbol{Loss}_{\rightarrow}(l) = &\frac{1}{2|N_{\sigma_g}(l)|}\Sigma_{p\in N_{\sigma_g}(l)}(\\ &||\hat{\boldsymbol{p}}_{\rightarrow}((\hat{\boldsymbol{p}}_{\uparrow}(p))) - \varphi(l, f_y(p) - \Delta y)||_1 + \\ &||\hat{\boldsymbol{p}}_{\rightarrow}((\hat{\boldsymbol{p}}_{\downarrow}(p))) - \varphi(l, f_y(p) + \Delta y)||_1),\end{aligned}$$

For each pixel p, given a **fixed** interval Δy, predict the offset comparing with the GT of last row and next row in the same lane in 3 directions.

Decoder → local to global

Greedy

Efficient

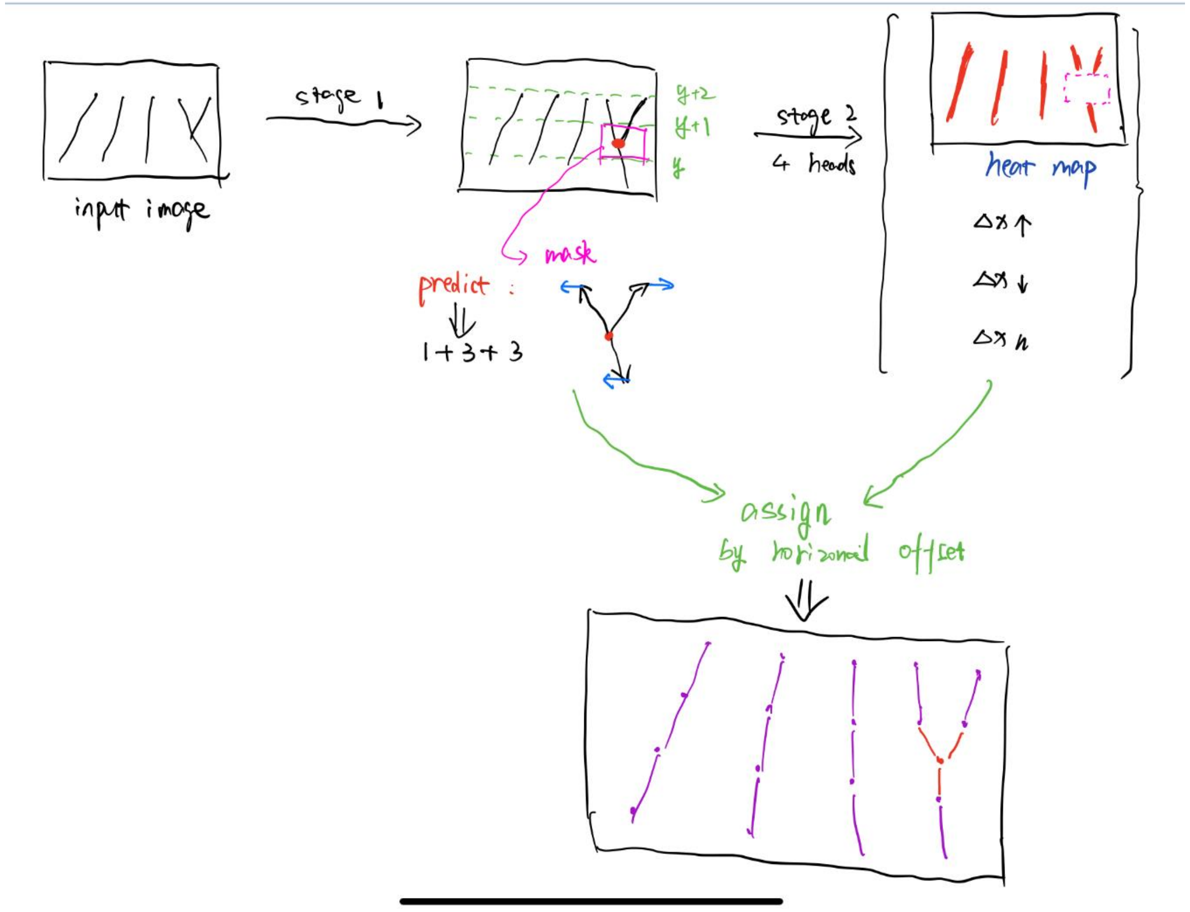


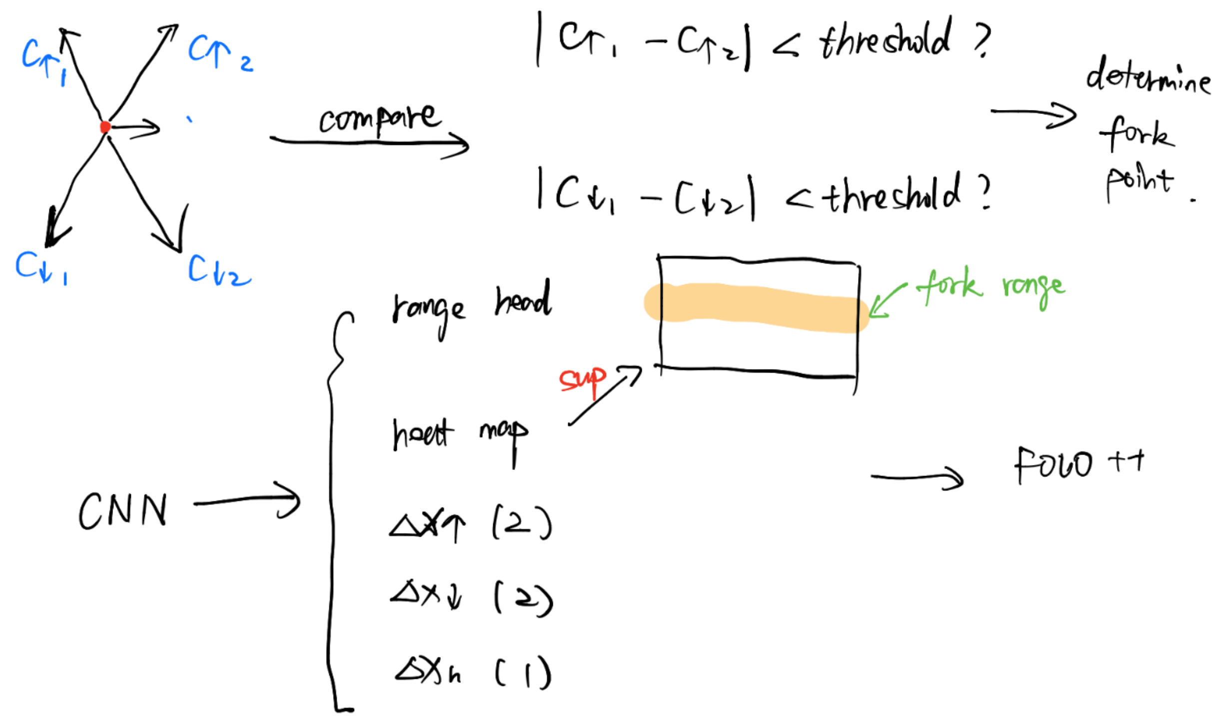
Conclusion

- More prior info: parallelism, continuity, rigidity of lanes and sequence (from bottom to up), equal width in most lane areas, etc.
- 2 branches: local → refinement, global → occlusion, etc.
- Offset map: improve precision and refinement

Idea → for fork lanes case

details





Reference

- [1] Hala Abualsaud, Sean Liu, David Lu, Kenny Situ, Akshay Rangesh, Mohan M. Trivedi, “LaneAF: Robust Multi-Lane Detection with Affinity Fields”, arXiv preprint arXiv: 2103.12040, 2021.
- [2] Tu Zheng, Hao Fang, Yi Zhang, Wenjian Tang, Zheng Yang, Haifeng Liu, Deng Cai, “RESA: Recurrent Feature-Shift Aggregator for Lane Detection”, arXiv preprint arXiv: 2008.13719, 2020.
- [3] Jinming Su, Chao Chen, Ke Zhang, Junfeng Luo, Xiaoming Wei, Xiaolin Wei, “Structure Guided Lane Detection”, IJCAI 2021, 2021.
- [4] Lucas Tabelini, Rodrigo Berriel, Thiago M. Paixão, Claudine Badue, Alberto F. de Souza, Thiago Oliveira-Santos, “Keep your Eyes on the Lane: Real-time Attention-guided Lane Detection”, CVPR 2021, 2021.
- [5] Hang Xu, Shaoju Wang, Xinyue Cai, Wei Zhang, Xiaodan Liang, Zhenguo Li, “CurveLane-NAS: Unifying Lane-Sensitive Architecture Search and Adaptive Point Blending”, ECCV 2020, 2020.
- [6] Zequn Qin, Huanyu Wang, Xi Li, “Ultra Fast Structure-aware Deep Lane Detection”, ECCV 2020, 2020.
- [7] Lizhe Liu, Xiaohao Chen, Siyu Zhu, Ping Tan, “CondLaneNet: a Top-to-down Lane Detection Framework Based on Conditional Convolution”, ICLR 2021, 2021.
- [8] Zhan Qu, Huan Jin, Yang Zhou, Zhen Yang, Wei Zhang, “Focus on Local: Detecting Lane Marker from Bottom Up via Key Point”, CVPR 2021, 2021.