G-TAD: Sub-Graph Localization for Temporal Action Detection

M. Xu, C. Zhao, D. S. Rojas, A. Thabet, B. Ghanem

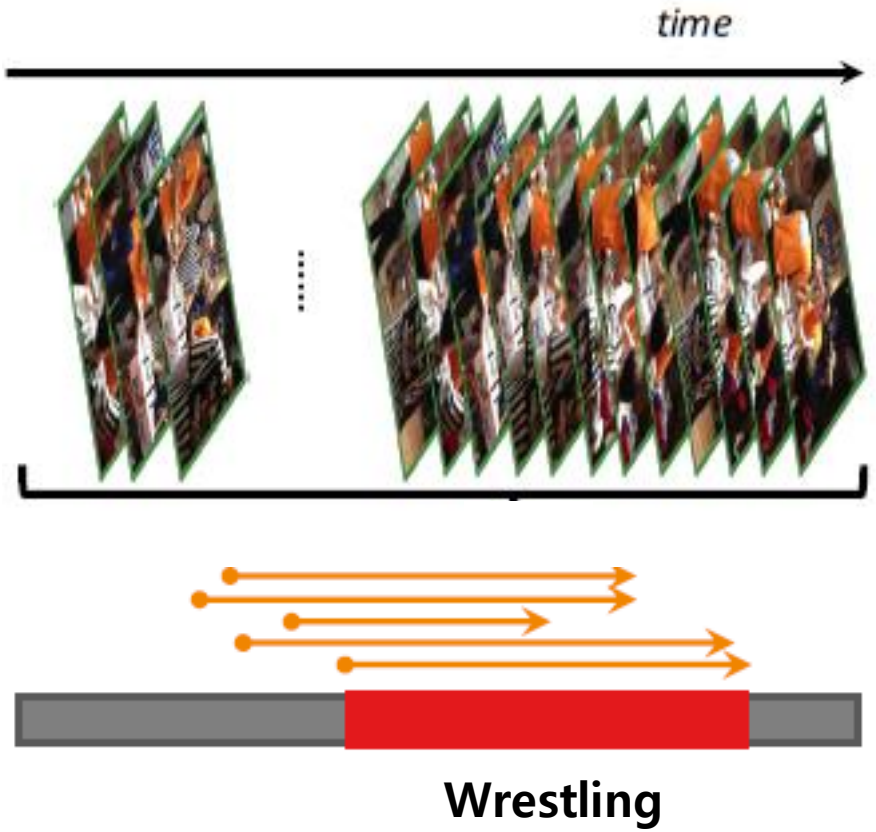
CVPR 2020

Jae Yoo Park

ECE, Seoul National University

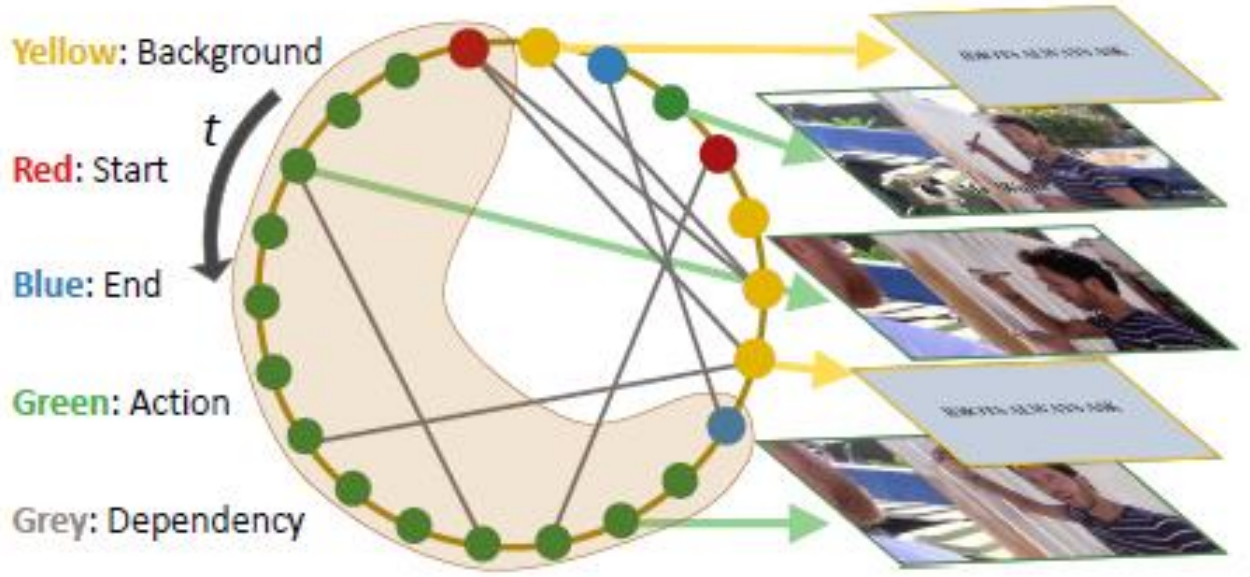
Task : Temporal Action Detection

- Recognize when & what type of the action happen in a given video
- 1-D version of Object Detection (e.g. R-CNN Series [1],[2])
- Previous works only consider temporal context via 1-D convolution on proposals(bounding boxes)
- Exploit semantic context even for background information via GCNs

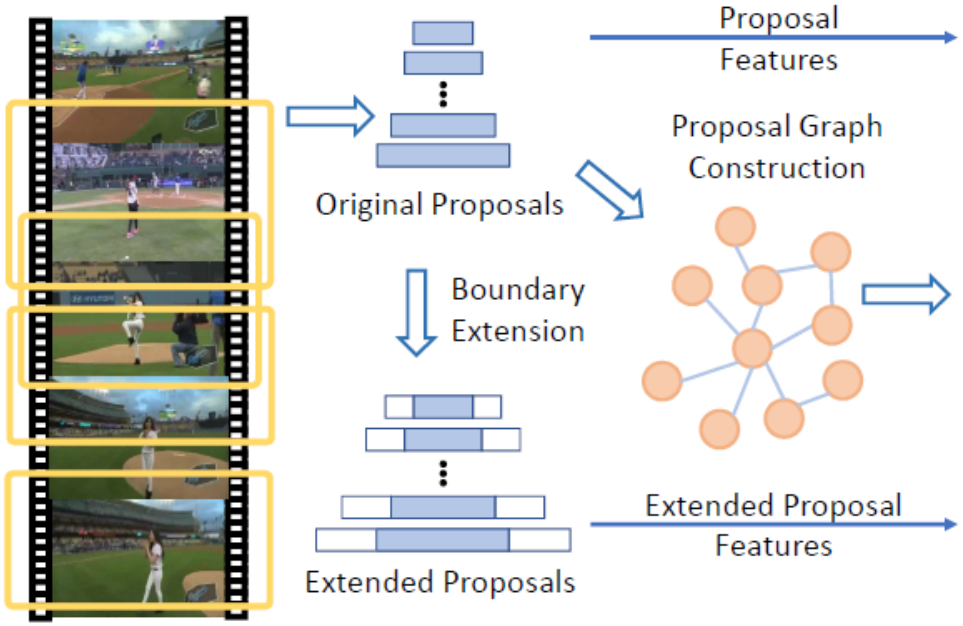


Constructing Graph

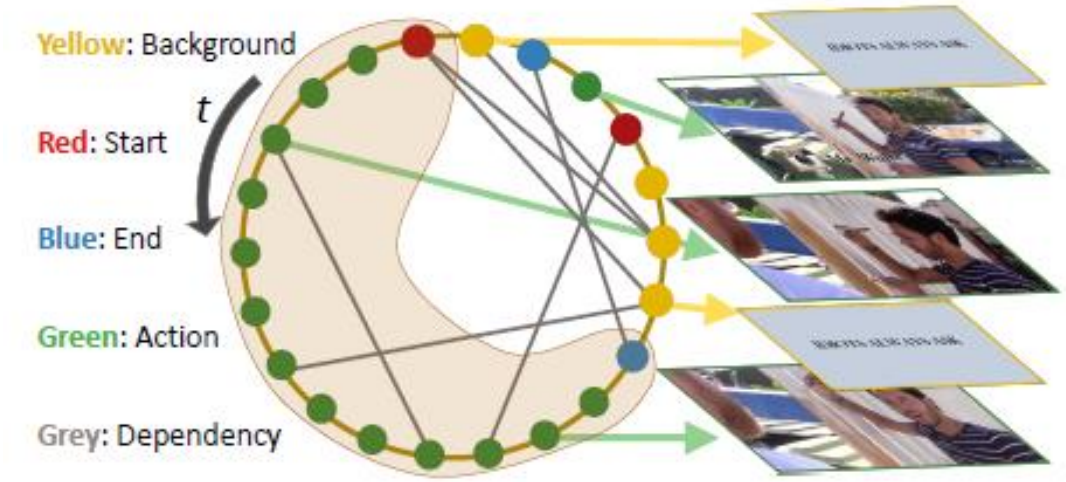
- Graph : Video
- Node : Video Snippet
- Edge : Snippet – Snippet
- Actions : Sub-Graph
- Temporal Action Detection : Sub-Graph Localization



P-GCN[3] VS GTAD



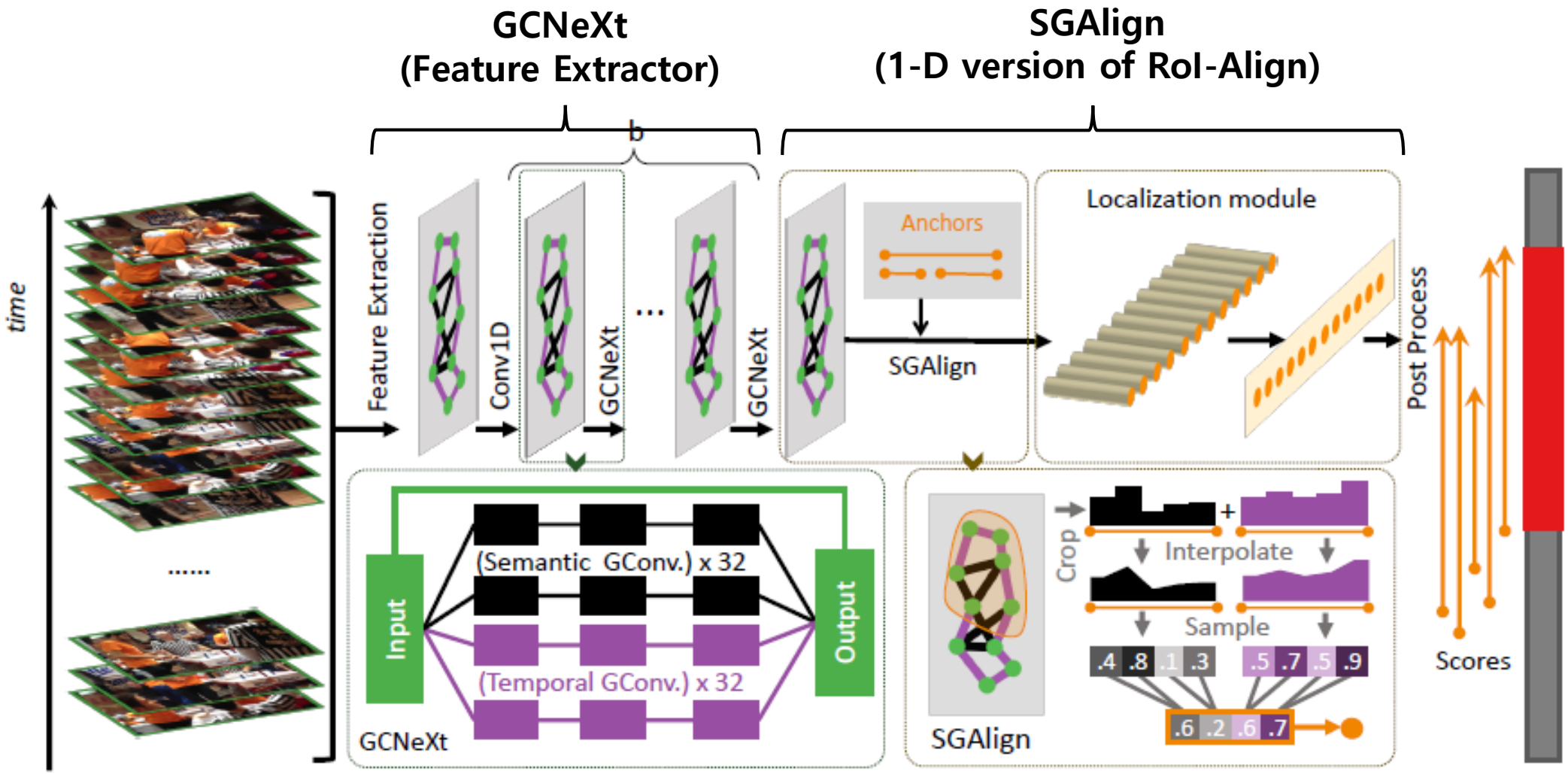
P-GCN[3]



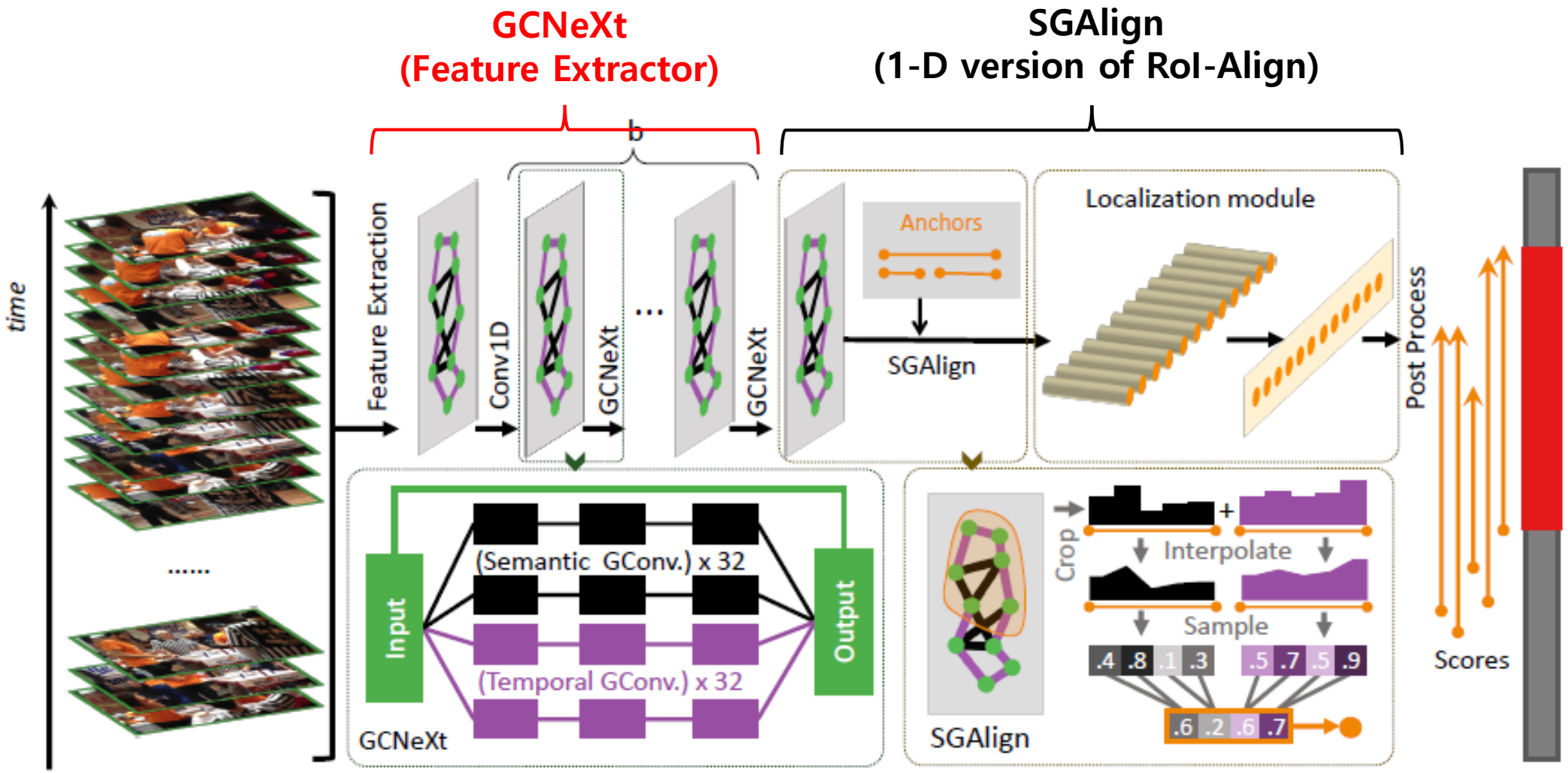
GTAD

- P-GCN : Graph based on the proposals
- GTAD : Graph based on the snippets
- P-GCN could not consider the background information effectively

Overall Architecture

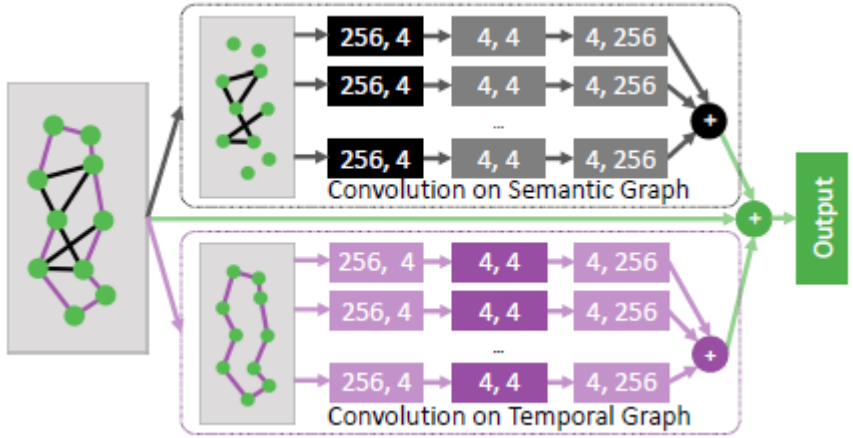


Overall Architecture



GCNeXt – Feature Extrator

- Edge : Snippet – Snippet -> Adjacency Matrix
  - Semantic Edge : KNN by Feature Similarity
  - Temporal Edge : Temporal order (Forward, Backward)
- Spatial Edge Convolution(DGCNN [4])



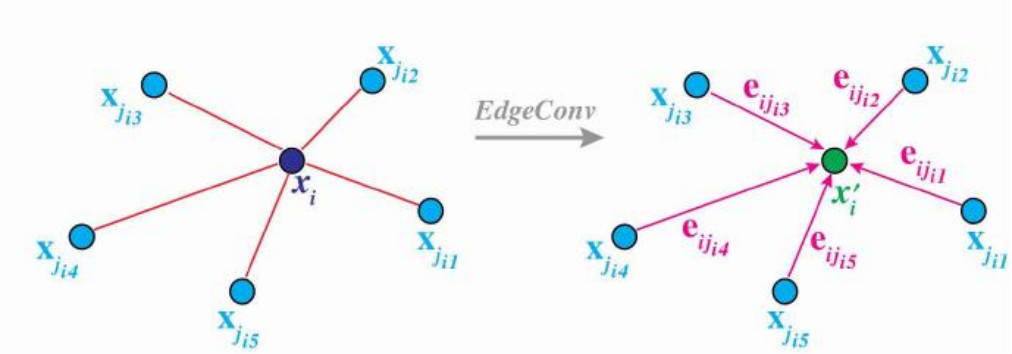
$$\mathcal{F}(X, A, W) = ([X^T, AX^T - X^T]W)^T$$

- GCNeXt
  - DeepGCNs [5] : Residual connection brings stability for stacking graph convolutions
  - ResNeXt [6] : Split-Transform-Merge

$$\mathcal{H}(X, A, W) = ReLU(\mathcal{F}'(X, A_t^f, W_t^f) + \mathcal{F}'(X, A_t^b, W_t^b) + \mathcal{F}'(X, A_s, W_s) + X)$$

GCNeXt – Feature Extrator

- Edge : Snippet – Snippet -> Adjacency Matrix
  - Semantic Edge : KNN by Feature Similarity
  - Temporal Edge : Temporal order (Forward, Backward)
- Spatial Edge Convolution(DGCNN [4])



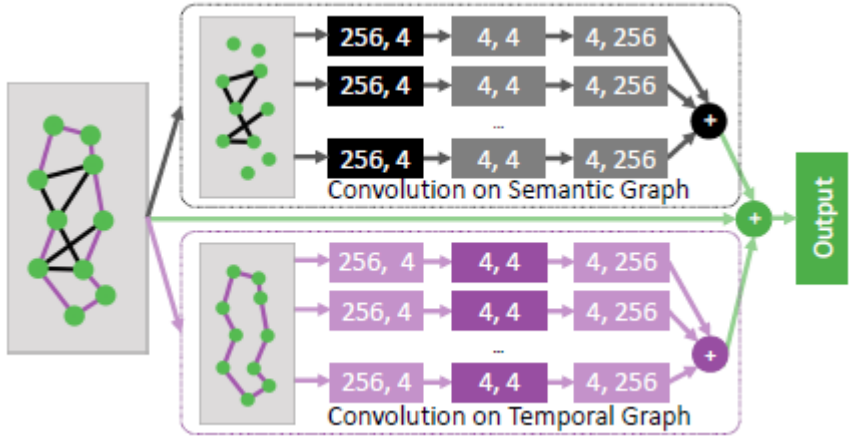
$$\mathcal{F}(X, A, W) = ([X^T, AX^T - X^T]W)^T$$

- GCNeXt
  - DeepGCNs [5] : Residual connection brings stability for stacking graph convolutions
  - ResNeXt [6] : Split-Transform-Merge

$$\mathcal{H}(X, A, W) = ReLU(\mathcal{F}'(X, A_t^f, W_t^f) + \mathcal{F}'(X, A_t^b, W_t^b) + \mathcal{F}'(X, A_s, W_s) + X)$$

GCNeXt – Feature Extrator

- Edge : Snippet – Snippet -> Adjacency Matrix
  - Semantic Edge : KNN by Feature Similarity
  - Temporal Edge : Temporal order (Forward, Backward)
- Spatial Edge Convolution(DGCNN [4])

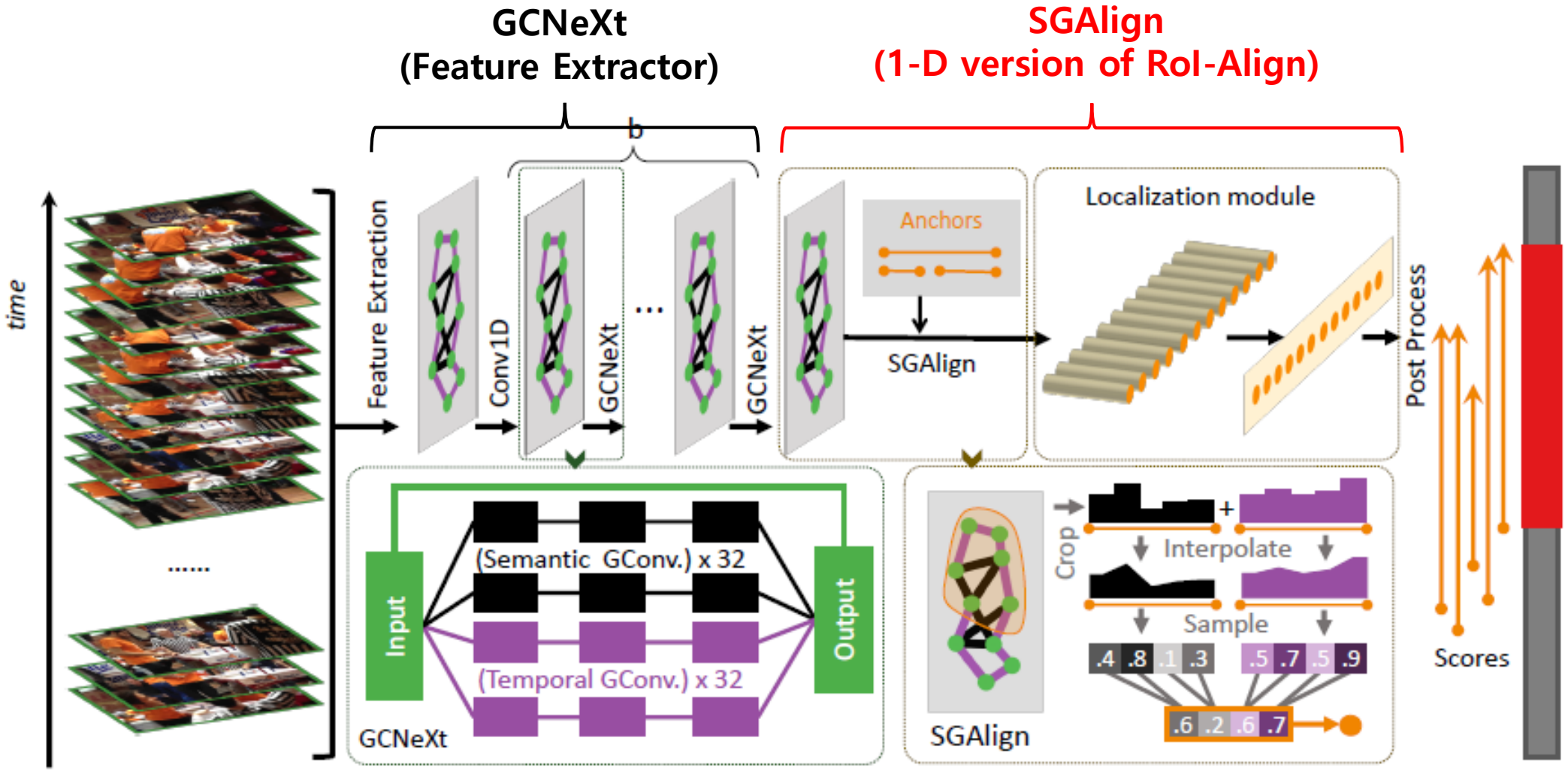


$$\mathcal{F}(X, A, W) = ([X^T, AX^T - X^T]W)^T$$

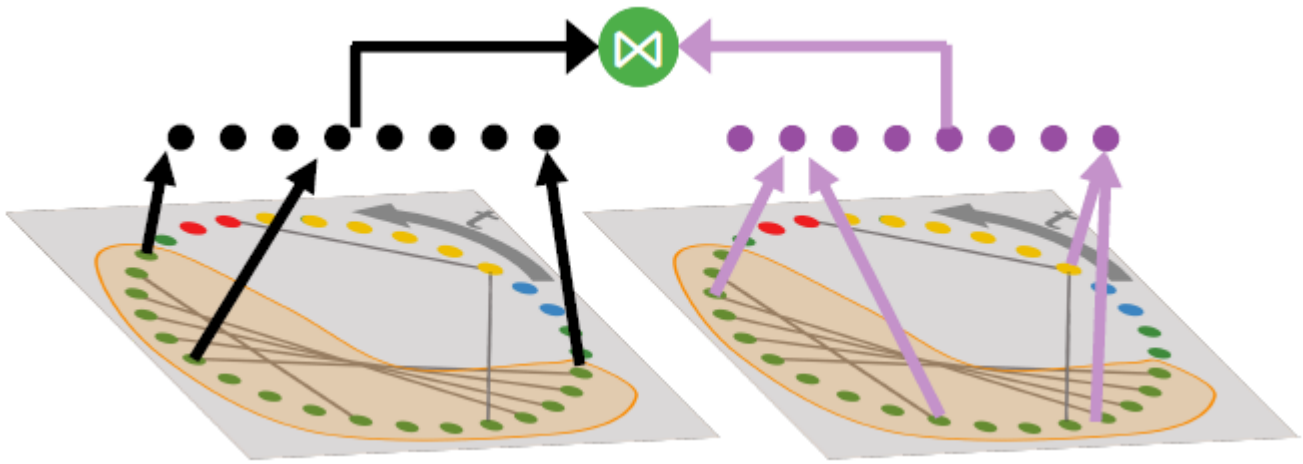
- GCNeXt
  - DeepGCNs [5] : Residual connection brings stability for stacking graph convolutions
  - ResNeXt [6] : Split-Transform-Merge

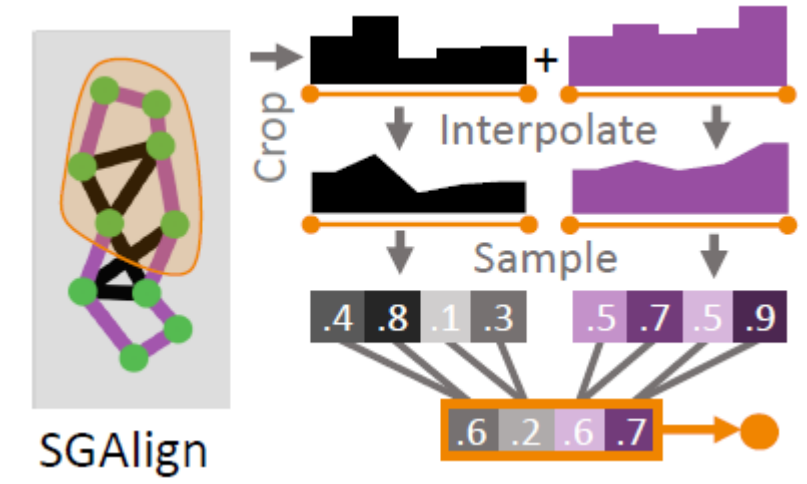
$$\mathcal{H}(X, A, W) = ReLU(\mathcal{F}'(X, A_t^f, W_t^f) + \mathcal{F}'(X, A_t^b, W_t^b) + \mathcal{F}'(X, A_s, W_s) + X)$$

Overall Architecture



SGAlign – Sub-Graph Align





- All possible pre-defined anchors -> Fixed size of the feature
- Inner Feature(Black) + Semantic Feature(Purple)
- Training :

$$L = L_g + L_n + \lambda_2 \cdot L_r$$

$$L_g = L_{wce}(p_{cls}, \mathbf{1}\{g_c > 0.5\}) + \lambda_1 \cdot L_{mse}(p_{reg}, g_c)$$

$$L_n = L_{wce}(p_s, g_{ns}) + L_{wce}(p_e, g_{ne})$$

Experiment Results

- Achieve state-of-the-art performance on two popular action detection benchmarks; ActivityNet 1.3 & THUMOS 14

Table 1. **Action detection results on validation set of ActivityNet-1.3**, measured by mAP (%) at different tIoU thresholds and the average mAP. G-TAD achieves better performance in average mAP than the other methods, even the latest work of BMN and P-GCN shown in the second-to-last block.

Method	0.5	0.75	0.95	Average
Wang *et al.* [46]	43.65	-	-	-
Singh *et al.* [40]	34.47	-	-	-
SCC [21]	40.00	17.90	4.70	21.70
CDC [36]	45.30	26.00	0.20	23.80
R-C3D [52]	26.80	-	-	-
BSN [30]	46.45	29.96	8.02	30.03
Chao *et al.* [9]	38.23	18.30	1.30	20.22
P-GCN [55]	48.26	33.16	3.27	31.11
BMN [29]	50.07	**34.78**	8.29	33.85
G-TAD (ours)	**50.36**	34.60	**9.02**	**34.09**

Table 2. **Action detection results on testing set of THUMOS-14**, measured by mAP (%) at different tIoU thresholds. G-TAD achieves the best performance for IoU@0.7, and combined with P-GCN, G-TAD significantly outperforms all the other methods.

Method	0.3	0.4	0.5	0.6	0.7
SST [5]	-	-	23.0	-	-
CDC [36]	40.1	29.4	23.3	13.1	7.9
TURN-TAP [15]	44.1	34.9	25.6	-	-
CBR [16]	50.1	41.3	31.0	19.1	9.9
SSN [56]	51.9	41.0	29.8	-	-
BSN [30]	53.5	45.0	36.9	28.4	20.0
TCN [11]	-	33.3	25.6	15.9	9.0
TAL-Net [9]	53.2	48.5	42.8	33.8	20.8
MGG [32]	53.9	46.8	37.4	29.5	21.3
DBG [28]	57.8	49.4	39.8	30.2	21.7
Yeung *et al.* [53]	36.0	26.4	17.1	-	-
Yuan *et al.* [54]	36.5	27.8	17.8	-	-
Hou *et al.* [22]	43.7	-	22.0	-	-
SS-TAD [4]	45.7	-	29.2	-	9.6
BMN [29]	56.0	47.4	38.8	29.7	20.5
G-TAD (ours)	54.5	47.6	40.2	30.8	**23.4**
BSN+P-GCN [55]	63.6	57.8	49.1	-	-
G-TAD+P-GCN	**66.4**	**60.4**	**51.6**	**37.6**	22.9

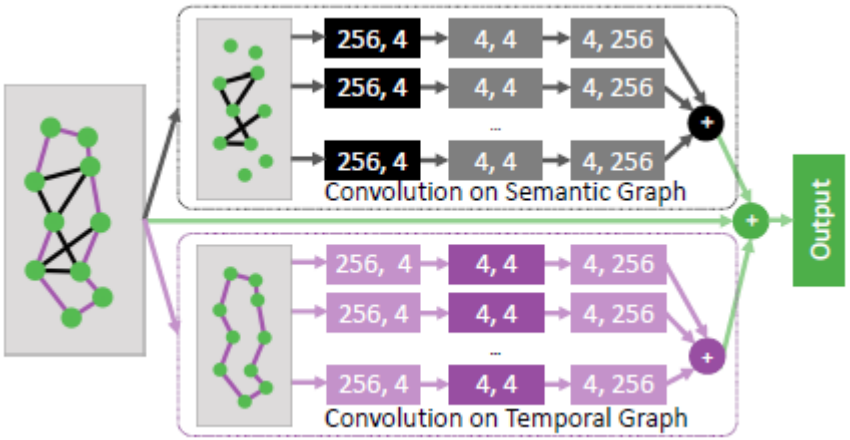
Ablation Study

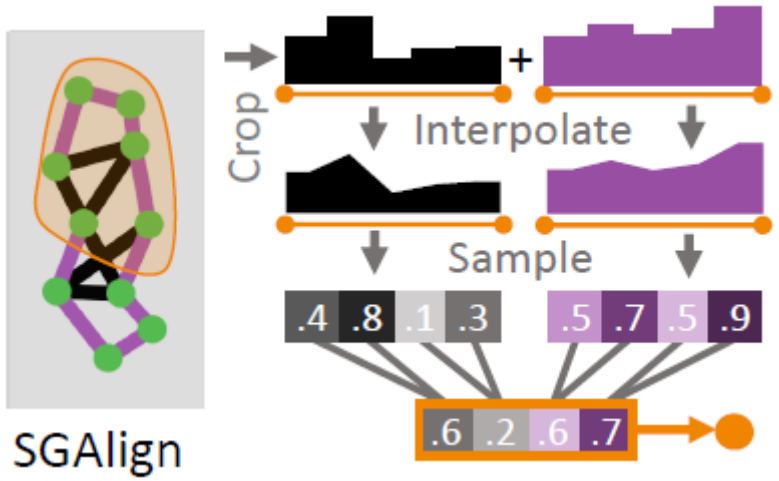
Table 3. **Ablating GCNeXt Components.** We disable temporal/semantic graph convolutions and set different cardinalities for detection on ActivityNet-1.3.

GCNeXt block			tIoU on Validation Set			
Temp.	Sem.	Card.	0.5	0.75	0.95	Avg.
✗	✗	1	48.12	32.16	6.41	31.65
✓	✓	1	50.20	**34.80**	7.35	33.88
✓	✗	32	50.13	34.17	8.70	33.67
✗	✓	32	49.09	33.32	8.02	32.63
✓	✓	32	**50.36**	34.60	**9.02**	**34.09**

Table 4. **Ablating SGAlign Components.** We disable the sample-rescale process and the feature concatenation from the semnantic graph for detection on ActivityNet-1.3. The rescaling strategy leads to slight improvement, while the main gain arises from the use of context information (semantic graph).

SGAlign		tIoU on Validation Set			
Samp.	Concat.	0.5	0.75	0.95	Avg.
✗	✗	49.84	34.58	8.17	33.78
✓	✗	49.86	**34.60**	**9.56**	33.89
✓	✓	**50.36**	**34.60**	9.02	**34.09**





Context of Background

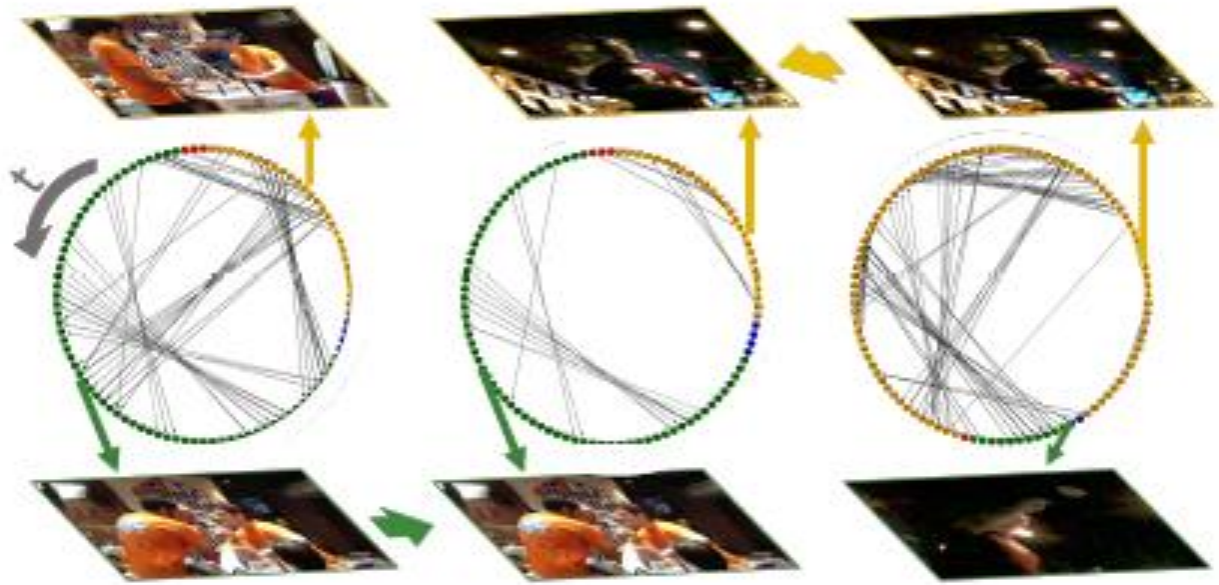


Figure 5. **Semantic graphs and Context.** Given two videos (left and right), we combine action frames of one video with background frames of another to create a synthetic video with no action context (middle). As expected, the semantic graph of the synthetic video contains no edges between action and background snippets.

Conclusion

- Formulate temporal action detection task as sub-graph localization problem
- Introduce GCNeXt layer, which is graph version of ResNeXt, by incorporating both temporal and semantic context
- Introduce SGAlign layer, which is 1-D version of RoI Align layer, while considering both of the inner and the context feature
- Achieve state-of-the-art performance on two popular action detection benchmarks; THUMOS 14 & ActivityNet 1.3

Reference

1. S. Ren et al., Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, NIPS, 2015
2. K. He et al., Mask R-CNN, ICCV, 2017
3. R. Zeng et al., Graph Convolutional Networks for Temporal Action Localization, ICCV, 2019
4. Y. Wang et al., Dynamic Graph CNN for Learning on Point Clouds, ACM Transactions on Graphics, 2018
5. G. Li et al., DeepGCNs : Can GCNs Go as Deep as CNNs?, ICCV, 2019
6. S. Xie et al., Aggregated Residual Transformation for Deep Neural Networks, CVPR, 2017

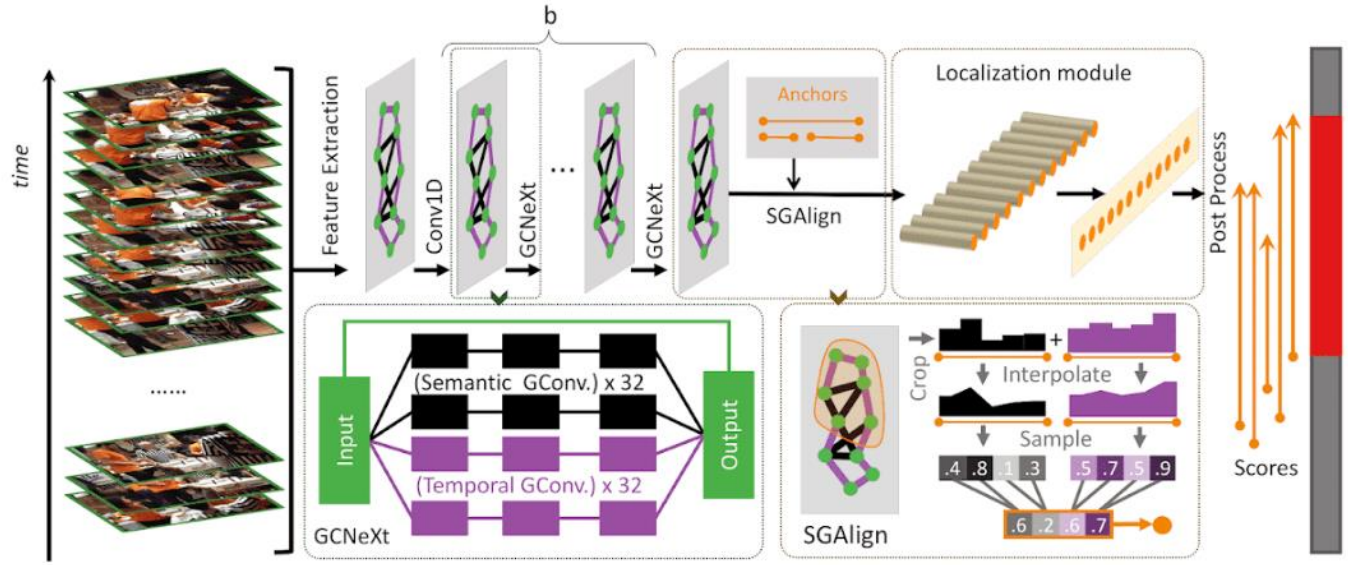
Code Experiment

- **Well-organized Official Repo** : https://github.com/Frostinassiky/gtad

G-TAD

Ranked #4 Temporal Action Localization on THUMOS'14

This repo holds the codes of paper: "G-TAD: Sub-Graph Localization for Temporal Action Detection", accepted in CVPR 2020.



Dependencies

- Python == 3.7
- Pytorch==1.1.0 or 1.3.0
- CUDA==10.0.130
- CUDNN==7.5.1_0

Code Experiment - Structure

- gtad
  - gtad_train.py – main file to train the model
  - gtad_inference.py – file to obtain raw test outputs
  - gtad_postprocess.py – file to post-process the raw test outputs to prediction results
  - gtad_lib – directory including the main modules
    - Align1D_cuda.cpp – cuda implementation of the SGAlign module
    - Align1D_cuda_kernel.cu – cuda implementation of the SGAlign module 2
    - align.py – main module to aggregate the cuda-implemented the SGAlign module
    - setup.py – file to compile the cuda-implemented the SGAlign module
    - dataset.py – file to load the datasets
    - loss_function.py – implementation of the loss function
    - models.py – implementation of the gtad model including GCNeXt
    - opts.py – configuration of the model
  - data – directory to save data & label
  - evaluation – directory including files to evaluate the model
  - output – directory to save the outputs

Code Experiment – gtad_train.py

- Load Model
- Set configuration
- Load Data

```
94 if __name__ == '__main__':
95     opt = opts.parse_opt()
96     opt = vars(opt)
97     if not os.path.exists(opt["output"]):
98         os.makedirs(opt["output"])
99 
00     model = GTAD(opt)
01 
02     model = torch.nn.DataParallel(model, device_ids=list(range(opt['n_gpu']))).cuda()
03     print('use {} gpus to train!'.format(opt['n_gpu']))
04 
05     optimizer = optim.Adam(model.parameters(), lr=opt["training_lr"],
06                            weight_decay=opt["weight_decay"])
07     train_loader = torch.utils.data.DataLoader(VideoDataSet(opt, subset="train"),
08                                                batch_size=opt["batch_size"], shuffle=True,
09                                                num_workers=8, pin_memory=True)
10 
11     test_loader = torch.utils.data.DataLoader(VideoDataSet(opt, subset="validation"),
12                                               batch_size=opt["batch_size"], shuffle=False,
13                                               num_workers=8, pin_memory=True)
14 
15     scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=opt["step_size"], gamma=opt["step_gamma"])
16     mask = get_mask(opt["temporal_scale"], opt['max_duration']).cuda()
17     for epoch in range(opt["train_epochs"]):
18       with autograd.detect_anomaly():
19         train(train_loader, model, optimizer, epoch, mask)
20         test(test_loader, model, epoch, mask)
21         scheduler.step()
22 
```

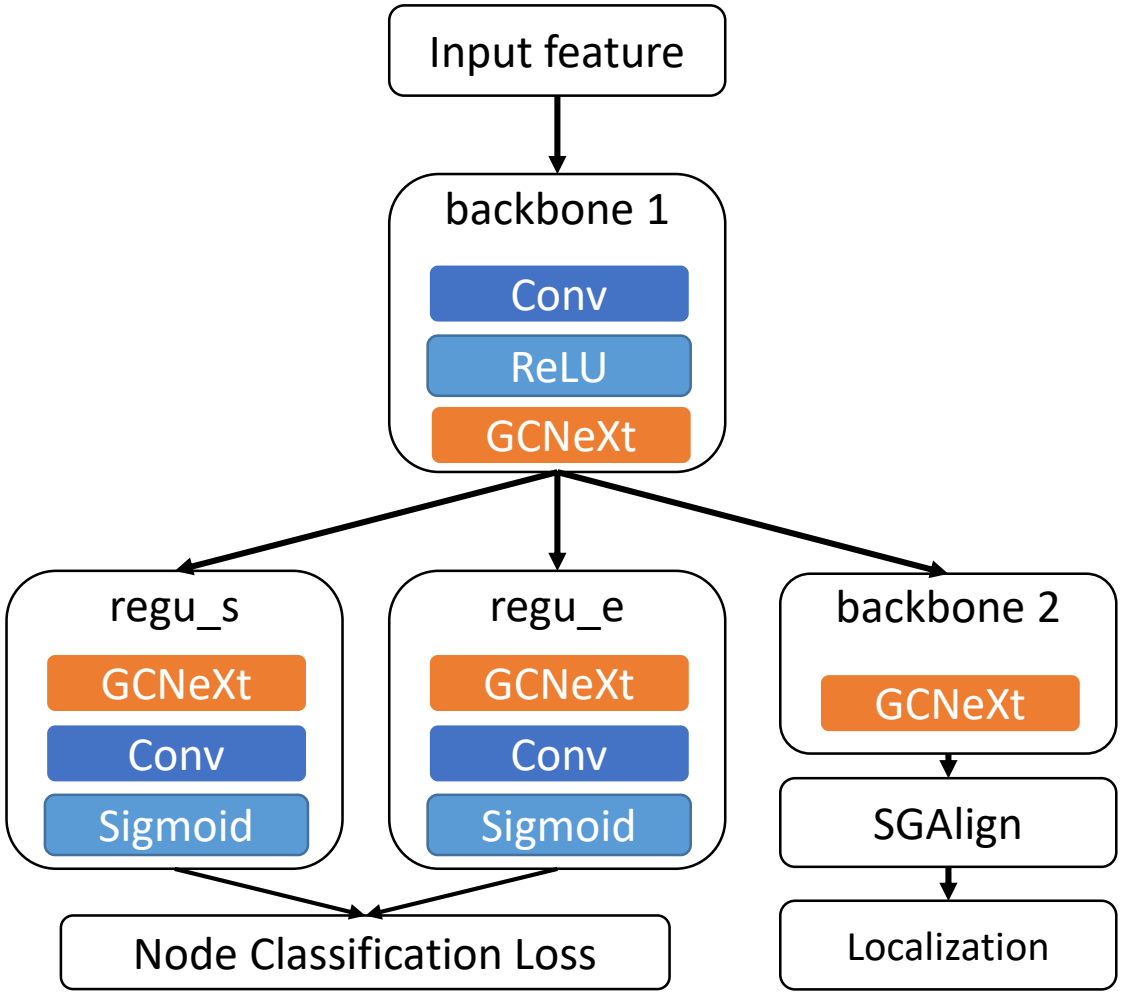
Code Experiment – gtad_train.py

- Train the model

```
37 # train
38 def train(data_loader, model, optimizer, epoch, bm_mask):
39     model.train()
40     total_am, subgraph_am, node_am = AverageMeter(), AverageMeter(), AverageMeter()
41     for n_iter, (input_data, label_confidence, label_start, label_end) in enumerate(data_loader):
42         # forward pass
43         confidence_map, start, end = model(input_data.cuda())
44         # loss
45         gt_iou_map = label_confidence.cuda() * bm_mask
46         subgraph_loss = subgraph_loss_func(confidence_map, gt_iou_map, bm_mask)
47         node_loss = node_loss_func(start, end, label_start.cuda(), label_end.cuda())
48         loss = subgraph_loss + node_loss
49 
50         # update step
51         optimizer.zero_grad()
52         loss.backward()
53         torch.nn.utils.clip_grad_norm_(model.parameters(), 1)
54         optimizer.step()
55         # update losses
56         total_am.update(loss.detach())
57         subgraph_am.update(subgraph_loss.detach())
58         node_am.update(node_loss.detach())
59 
60     print("[Epoch {0:03d}]\tLoss {1:.2f} = {2:.2f} + {3:.2f} (train)".format(
61         epoch, total_am.avg(), subgraph_am.avg(), node_am.avg()))
62
```

Code Experiment – models.py

- Define the model




```python
class GTAD(nn.Module):
    def __init__(self, opt):
        super(GTAD, self).__init__()
        self.tscale = opt["temporal_scale"]
        self.feat_dim = opt["feat_dim"]
        self.bs = opt["batch_size"]
        self.h_dim_1d = 256
        self.h_dim_2d = 128
        self.h_dim_3d = 512
        self.goi_style = opt['goi_style']
        self.h_dim_goi = self.h_dim_1d*(16,32,32)[opt['goi_style']]
        self.idx_list = []

        # Backbone Part 1
        self.backbone1 = nn.Sequential(
            nn.Conv1d(self.feat_dim, self.h_dim_1d, kernel_size=3, padding=1, groups=4),
            nn.ReLU(inplace=True),
            GCNeXt(self.h_dim_1d, self.h_dim_1d, k=3, groups=32, idx=self.idx_list),
        )

        # Regularization
        self.regu_s = nn.Sequential(
            GCNeXt(self.h_dim_1d, self.h_dim_1d, k=3, groups=32),
            nn.Conv1d(self.h_dim_1d, 1, kernel_size=1), nn.Sigmoid()
        )
        self.regu_e = nn.Sequential(
            GCNeXt(self.h_dim_1d, self.h_dim_1d, k=3, groups=32),
            nn.Conv1d(self.h_dim_1d, 1, kernel_size=1), nn.Sigmoid()
        )

        # Backbone Part 2
        self.backbone2 = nn.Sequential(
            GCNeXt(self.h_dim_1d, self.h_dim_1d, k=3, groups=32,idx=self.idx_list),
        )

        # SGAlign: sub-graph of interest alignment
        self.goi_align = GraphAlign(
            t=self.tscale, d=opt['max_duration'], bs=self.bs,
            samp=opt['goi_samp'], style=opt['goi_style']  # for ablation
        )

        # Localization Module
        self.localization = nn.Sequential(
            nn.Conv2d(self.h_dim_goi, self.h_dim_3d, kernel_size=1), nn.ReLU(inplace=True),
            nn.Conv2d(self.h_dim_3d, self.h_dim_2d, kernel_size=1), nn.ReLU(inplace=True),
            nn.Conv2d(self.h_dim_2d, self.h_dim_2d, kernel_size=opt['kern_2d'], padding=opt['pad_2d']), nn.ReLU(inplace=True),
            nn.Conv2d(self.h_dim_2d, self.h_dim_2d, kernel_size=opt['kern_2d'], padding=opt['pad_2d']), nn.ReLU(inplace=True),
            nn.Conv2d(self.h_dim_2d, 2, kernel_size=1), nn.Sigmoid()
        )

        # Position encoding (not used)
        self.pos = torch.arange(0, 1, 1.0 / self.tscale).view(1, 1, self.tscale)

    def forward(self, snip_feature):
        del self.idx_list[:]  # clean the idx list
        base_feature = self.backbone1(snip_feature).contiguous()  # (bs, 2048, 256) -> (bs, 256, 256)
        gcnext_feature = self.backbone2(base_feature)  #

        regu_s = self.regu_s(base_feature).squeeze(1)  # start
        regu_e = self.regu_e(base_feature).squeeze(1)  # end

        if self.goi_style==2:
            idx_list = [idx for idx in self.idx_list if idx.device == snip_feature.device]
            idx_list = torch.cat(idx_list, dim=2)
        else:
            idx_list = None

        subgraph_map = self.goi_align(gcnext_feature, idx_list)
        iou_map = self.localization(subgraph_map)
        return iou_map, regu_s, regu_e
```

Code Experiment – models.py

- GCNeXt
- Temporal Edge with Forward/Backward edge is equivalent to 1D Temporal Convolution
- Refer to Eq. (11) in the supplementary material

```python
# basic block
class GCNeXt(nn.Module):
    def __init__(self, channel_in, channel_out, k=3, norm_layer=None, groups=32, width_group=4, idx=None):
        super(GCNeXt, self).__init__()
        self.k = k
        self.groups = groups

        if norm_layer is None:
            norm_layer = nn.BatchNorm1d
        width = width_group * groups
        self.tconvs = nn.Sequential(
            nn.Conv1d(channel_in, width, kernel_size=1), nn.ReLU(True),
            nn.Conv1d(width, width, kernel_size=3, groups=groups, padding=1), nn.ReLU(True),
            nn.Conv1d(width, channel_out, kernel_size=1),
        ) # temporal graph

        self.sconvs = nn.Sequential(
            nn.Conv2d(channel_in * 2, width, kernel_size=1), nn.ReLU(True),
            nn.Conv2d(width, width, kernel_size=1, groups=groups), nn.ReLU(True),
            nn.Conv2d(width, channel_out, kernel_size=1),
        ) # semantic graph

        self.relu = nn.ReLU(True)
        self.idx_list = idx

    def forward(self, x):
        identity = x  # residual
        tout = self.tconvs(x)  # conv on temporal graph

        x_f, idx = get_graph_feature(x, k=self.k, style=1)  # (bs,ch,100) -> (bs, 2ch, 100, k)
        sout = self.sconvs(x_f)  # conv on semantic graph
        sout = sout.max(dim=-1, keepdim=False)[0]  # (bs, ch, 100, k) -> (bs, ch, 100)

        out = tout + identity + sout  # fusion
        if not self.idx_list is None:
            self.idx_list.append(idx)
        return self.relu(out)

```

Code Experiment – models.py

- SGAlign
- Apply Align1DLayer to the inner_feature and the context_feature
- Align 1D Layer is implemented in Cuda

```python
class GraphAlign(nn.Module):
    def __init__(self, k=3, t=100, d=100, bs=64, samp=0, style=0):
        super(GraphAlign, self).__init__()
        self.k = k
        self.t = t
        self.d = d
        self.bs = bs
        self.style = style
        self.expand_ratio = 0.5
        self.resolution = 16
        self.align_inner = Align1DLayer(self.resolution, samp)
        self.align_context = Align1DLayer(16,samp)
        self._get_anchors()

    def forward(self, x, index):
        bs, ch, t = x.shape
        # print('feature channel is', ch)
        if not self.anchors.is_cuda:  # run once
            self.anchors = self.anchors.cuda()

        anchor = self.anchors[:self.anchor_num * bs, :]  # (bs*tscale*tscal, 3)
        # print('first value in anchor is', anchor[0])
        feat_inner = self.align_inner(x, anchor)  # (bs*tscale*tscal, ch, resolution)
        #print('inner feature is with shape', feat_inner.shape)
        if self.style == 1: # use last layer neighbours
            feat, _ = get_graph_feature(x, k=self.k, style=2)  # (bs,ch,100) -> (bs, ch, 100, k)
            feat = feat.mean(dim=-1, keepdim=False)  # (bs, 2*ch, 100)
            feat_context = self.align_context(feat, anchor)  # (bs*tscale*tscal, ch, resolution//2)
            feat = torch.cat((feat_inner,feat_context), dim=2).view(bs, t, self.d, -1)
            # print('sg feature is with shape', feat.shape)
        elif self.style == 2: # use all layers neighbour
            feat, _ = get_graph_feature(x, k=self.k, style=2, idx_knn=index)  # (bs,ch,100) -> (bs, ch, 100, k)
            feat = feat.mean(dim=-1, keepdim=False)  # (bs, 2*ch, 100)
            feat_context = self.align_context(feat, anchor)  # (bs*tscale*tscal, ch, resolution//2)
            feat = torch.cat((feat_inner,feat_context), dim=2).view(bs, t, self.d, -1)
        else:
            feat = torch.cat((feat_inner,), dim=2).view(bs, t, t, -1)
        # print('shape after align is', feat_context.shape)

        return feat.permute(0, 3, 2, 1)  # (bs,2*ch*(-1),t,t)

    def _get_anchors(self):
        anchors = []
        for k in range(self.bs):
            for start_index in range(self.t):
                for duration_index in range(self.d):
                    if start_index + duration_index < self.t:
                        p_xmin = start_index
                        p_xmax = start_index + duration_index
                        center_len = float(p_xmax - p_xmin) + 1
                        sample_xmin = p_xmin - center_len * self.expand_ratio
                        sample_xmax = p_xmax + center_len * self.expand_ratio
                        anchors.append([k, sample_xmin, sample_xmax])
                    else:
                        anchors.append([k, 0, 0])
        self.anchor_num = len(anchors) // self.bs
        self.anchors = torch.tensor(np.stack(anchors)).float()  # save to cpu
        return  # anchors, anchor_num
```

Code Experiment – loss_function.py

- Objective Function
- Refer to Slide 11 for the detail
- Total Loss

$$L = L_g + L_n + \lambda_2 \cdot L_r$$

- Subgraph loss

$$L_g = L_{wce}(p_{cls}, 1\{g_c > 0.5\}) + \lambda_1 \cdot L_{mse}(p_{reg}, g_c)$$

- Node Loss

$$L_n = L_{wce}(p_s, g_{ns}) + L_{wce}(p_e, g_{ne})$$

```python
def subgraph_loss_func(pred_bm, gt_iou_map, bm_mask, lambda1=10):
    pred_bm_wcs = pred_bm[:, 0].contiguous()
    pred_bm_mse = pred_bm[:, 1].contiguous()

    wce_loss = pem_reg_loss_func(pred_bm_wcs, gt_iou_map, bm_mask)
    mse_loss = pem_cls_loss_func(pred_bm_mse, gt_iou_map, bm_mask)

    subgraph_loss = wce_loss + lambda1* mse_loss
    return subgraph_loss

def node_loss_func(pred_start, pred_end, gt_start, gt_end):
    def bi_loss(pred_score, gt_label):
        pred_score = pred_score.view(-1)
        gt_label = gt_label.view(-1)
        pmask = (gt_label > 0.5).float()
        num_entries = len(pmask)
        num_positive = torch.sum(pmask)
        ratio = num_entries / num_positive
        coef_0 = 0.5 * ratio / (ratio - 1)
        coef_1 = 0.5 * ratio
        epsilon = 0.000001
        loss_pos = coef_1 * torch.log(pred_score + epsilon) * pmask
        loss_neg = coef_0 * torch.log(1.0 - pred_score + epsilon)*(1.0 - pmask)
        loss = -1 * torch.mean(loss_pos + loss_neg)
        return loss

    loss_start = bi_loss(pred_start, gt_start)
    loss_end = bi_loss(pred_end, gt_end)
    loss = loss_start + loss_end
    return loss
```

Code Experiment – gtad_inference.py

- Inference
- Raw output from the model -> score for each proposal [start(xmin), end(xmax), classification score, regression score]

```python
if __name__ == '__main__':
    opt = opts.parse_opt()
    opt = vars(opt)
    if not os.path.exists(opt['output'] + "/results"):
        os.makedirs(opt['output'] + "/results")

    model = GTAD(opt)
    model = torch.nn.DataParallel(model, device_ids=[0]).cuda()
    checkpoint = torch.load(opt["output"] + "/GTAD_best.pth.tar")
    model.load_state_dict(checkpoint['state_dict'])
    model.eval()

    test_loader = torch.utils.data.DataLoader(VideoDataSet(opt, subset="validation", mode='inference'),
                                              batch_size=1, shuffle=False,
                                              num_workers=8, pin_memory=True, drop_last=False)

    print("Inference start")
    with torch.no_grad():
        for idx, input_data in test_loader:
            video_name = test_loader.dataset.video_list[idx[0]]
            offset = min(test_loader.dataset.data['indices'][idx[0]])
            video_name = video_name+'_{}'.format(math.floor(offset/250))
            input_data = input_data.cuda()

            # forward pass
            confidence_map, _, _ = model(input_data)

            clr_confidence = (confidence_map[0][1]).detach().cpu().numpy()
            reg_confidence = (confidence_map[0][0]).detach().cpu().numpy()

            # enumerate sub-graphs as proposals
            new_props = []
            for idx in range( opt["max_duration"]):
                for jdx in range(opt["temporal_scale"]):
                    start_index = jdx
                    end_index = start_index + idx+1
                    if end_index < opt["temporal_scale"]:
                        xmin = start_index * opt['skip_videoframes'] + offset  # map [0,99] to frames
                        xmax = end_index * opt['skip_videoframes'] + offset
                        clr_score = clr_confidence[idx, jdx]  # 64, 128
                        reg_score = reg_confidence[idx, jdx]
                        new_props.append([xmin, xmax, clr_score, reg_score])
            new_props = np.stack(new_props)

            col_name = ["xmin", "xmax", "clr_score", "reg_socre"]
            new_df = pd.DataFrame(new_props, columns=col_name)
            new_df.to_csv(opt["output"]+"/results/" + video_name + ".csv", index=False)

    print("Inference finished")
```

Code Experiment – gtad_postprocess.py

- Postprocess
- SoftNMS
- Compute IoU (Intersection over Union) with the groundtruth
- Compute mAP(Mean Average Precision)

```
if __name__ == '__main__':
    opt = opts.parse_opt()
    opt = vars(opt)
    opt["output"] = opt["output"]
    if not os.path.exists(opt["output"]):
        os.makedirs(opt["output"])
    opt_file = open(opt["output"] + "/opts.json", "w")
    json.dump(opt, opt_file)
    opt_file.close()

    print("Detection post processing start")
    gen_detection_multicore(opt)
    print("Detection Post processing finished")

    from evaluation.eval_detection import ANETdetection
    tious = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
    anet_detection = ANETdetection(
        ground_truth_filename='./evaluation/thumos_gt.json',
        prediction_filename=opt["output"] + '/detection_result.json',
        subset='test', tiou_thresholds=tious)
    mAPs, average_mAP = anet_detection.evaluate()
    for (tiou, mAP) in zip(tious, mAPs):
        print("mAP at tIoU {} is {}".format(tiou, mAP))
```

Code Experiment – Execution

- Datasets : THUMOS 14
- Loss = Subgraph_loss + node_classification_loss

```
[Epoch 000]    Loss 6.12 = 4.85 + 1.28 (train)
[Epoch 000]    Loss 5.63 = 4.33 + 1.30 (validation)
[Epoch 001]    Loss 4.76 = 3.57 + 1.19 (train)
[Epoch 001]    Loss 5.40 = 4.14 + 1.27 (validation)
[Epoch 002]    Loss 4.02 = 2.89 + 1.13 (train)
[Epoch 002]    Loss 5.42 = 4.19 + 1.23 (validation)
[Epoch 003]    Loss 3.58 = 2.49 + 1.08 (train)
[Epoch 003]    Loss 7.49 = 6.27 + 1.22 (validation)
[Epoch 004]    Loss 3.28 = 2.23 + 1.05 (train)
[Epoch 004]    Loss 5.22 = 4.02 + 1.20 (validation)
[Epoch 005]    Loss 2.77 = 1.74 + 1.03 (train)
[Epoch 005]    Loss 6.73 = 5.54 + 1.19 (validation)
[Epoch 006]    Loss 2.64 = 1.62 + 1.02 (train)
[Epoch 006]    Loss 7.58 = 6.39 + 1.19 (validation)
[Epoch 007]    Loss 2.58 = 1.56 + 1.02 (train)
[Epoch 007]    Loss 7.02 = 5.83 + 1.19 (validation)
[Epoch 008]    Loss 2.51 = 1.49 + 1.02 (train)
[Epoch 008]    Loss 7.36 = 6.16 + 1.19 (validation)
[Epoch 009]    Loss 2.46 = 1.45 + 1.01 (train)
[Epoch 009]    Loss 8.75 = 7.56 + 1.19 (validation)
```

- Final Performance
  - Slightly better than the reported one

```
Detection post processing start
Detection Post processing finished
mAP at tIoU 0.1 is 0.6523730039419201
mAP at tIoU 0.2 is 0.6263865611392926
mAP at tIoU 0.3 is 0.5815058801336764
mAP at tIoU 0.4 is 0.5161660141521802
mAP at tIoU 0.5 is 0.4315879779152393
mAP at tIoU 0.6 is 0.336516791890597
mAP at tIoU 0.7 is 0.2328141864616685
mAP at tIoU 0.8 is 0.12843067399141264
mAP at tIoU 0.9 is 0.030427392183438 66
2020.05.18-10.19.39
```

- Memory Usage : about 14.8GB

```
[0] TITAN Xp | 80'C, 100 % |  3713 / 12194 MB | sonic(3703M)
[1] TITAN Xp | 74'C, 100 % |  3701 / 12196 MB | sonic(3691M)
[2] TITAN Xp | 78'C, 100 % |  3701 / 12196 MB | sonic(3691M)
[3] TITAN Xp | 58'C, 100 % |  3701 / 12196 MB | sonic(3691M)
```

Thank You!