Computer Vision

Module 5. Deep Learning for Computer Vision

Task 5:

1. Paper review
2. CNN visualization
3. Experiment summary

5.1 Paper review

Title: STEP: Spatio-Temporal Progressive Learning for Video Action Detection

Authors: Yang, Xitong and Yang, Xiaodong and Liu, Ming-Yu and Xiao, Fanyi and Davis, Larry S. and Kautz, Jan

Link: http://openaccess.thecvf.com/content_CVPR_2019/papers/Yang_STEP_Spatio-Temporal_Progressive_Learning_for_Video_Action_Detection_CVPR_2019_paper.pdf (http://openaccess.thecvf.com/content_CVPR_2019/papers/Yang_STEP_Spatio-Temporal_Progressive_Learning_for_Video_Action_Detection_CVPR_2019_paper.pdf)

Tags: computer vision, deep learning, action detection

Year: 2019

Summary

What: The authors offer a progressive learning framework for spatio-temporal action detection in video - **Spatio-TEmporalProgressive(STEP) action detector**. Unlike existing methods that directly perform action detection in one run, the framework involves a multi-step optimization process that progressively refines the initial proposals towards the final solution. STEP consists of 2 stages: *spatial refinement* and *temporal extension*, where the former starts from sparse initial proposals and iteratively updates bounding boxes, and the latter gradually and adaptively increases sequence length to incorporate more related temporal context. STEP more effectively makes use of longer temporal information by handling the spatial displacement problem in action tubes (a sequence of bounding boxes of action). Extensive experiments on two benchmarks show that STEP consistently brings performance gains by usingonly a handful of proposals and a few updating steps.

How:STEP approach performs action detection at clip level, i.e., detection results are first obtained from each clip and then linked to build action tubes across a whole video. It is assumed that each action tubelet of a clip has a constant action label, considering the short duration of a clip, e.g., within one second. The target is to tackle the action detection problem through a few progressive steps, rather than directly detecting actions all at one run.

In order to detect the actions in a clip I_t with K frames, according to the maximum progressive steps S_{max}, we first extract the convolutional features for a set of clips $I = I_{t-Smax+1}, \ldots, I_t, \ldots, I_{t+Smax-1}$ using a backbone network such as VGG16 or I3D. The progressive learning starts with M predefined proposal cuboids $B^0 = \{b_i^0\}_{i=1}^M$ and $b_i^0 \in R^{K\times 4}$, which are sparsely sampled from a coarse-scale grid of boxes and replicated across time to form the initial proposals. These initial proposals are then progressively updated to better classify and localize the actions. At each steps, proposals are updated by performing the following processes in order:

• Extend: the proposals are temporally extended to the adjacent clips to include longer-range temporal context, and the temporal extension is adaptive to the movement of actions

• Refine: the extended proposals are forwarded to the spatial refinement, which outputs the classification and regression results

• Update: all proposals are updated using a simple greedy algorithm, i.e., each proposal is replaced by the regression output with the highest classification score:

$$b_i^s . = l_i^s(c*), c* = argmax p_i^s(c)$$

where c is an action class, $p_i^s \in R^{(C+1)}$ is the probability distribution of the ith proposal over C action classes plus background, $l_i^s \in R^{K\times 4\times C}$ denotes its parameterized coordinates (for computing the localization loss) ateach frame for each class, and $. =$ indicates decoding the parameterized coordinates.

Algorithm: STEP Action Detection for Clip I_t

Input: video clips I, initial proposals B^0, and maximum steps S_{max}

Output: detection results $\{(p_i^{S_{max}}, l_i^{S_{max}})\}_{i=1}^M$

```
1 extract convolutional features for video clips I
2 for s←1 to S_max do
3    if s==1 then:
4        //initial proposals
5        B'{s-1} ← B_0
6    else:
7        //temporal extension
8        B'{s-1} ← Extend(B{s-1})
9    end
10   // spatial refinement
11   {(p{S}_i,l^{S}_i)\}^{M}_{i=1} ← Refine(B'{s-1})
12   //update proposals
13   B{s} ← Update({(p{S}_i,l^{S}_i)\}^{M}_{i=1})
14 end
```

Results:

1. Comparison with the state-of-the-art methods on **UCF101** by frame-mAP and video-mAP under different IoU thresholds:

Method	frame-mAP@0.5 (mailto:frame-mAP@0.5)	video-mAP@0.05 (mailto:video-mAP@0.05)	video-mAP@0.1 (mailto:video-mAP@0.1)	video-mAP@0.2 (mailto:video-mAP@0.2)
MR-TS	65.7	78.8	77.3	72.9
ROAD	--	--	--	73.5
CPLA	--	79.0	77.3	73.5
RTPR	--	81.5	80.7	76.3
PntMatch	67.0	79.4	77.7	76.2
T-CNN	67.3	78.2	77.9	73.1
ACT	69.5	--	--	76.5
STEP	**75.0**	**84.6**	**83.1**	**76.6**

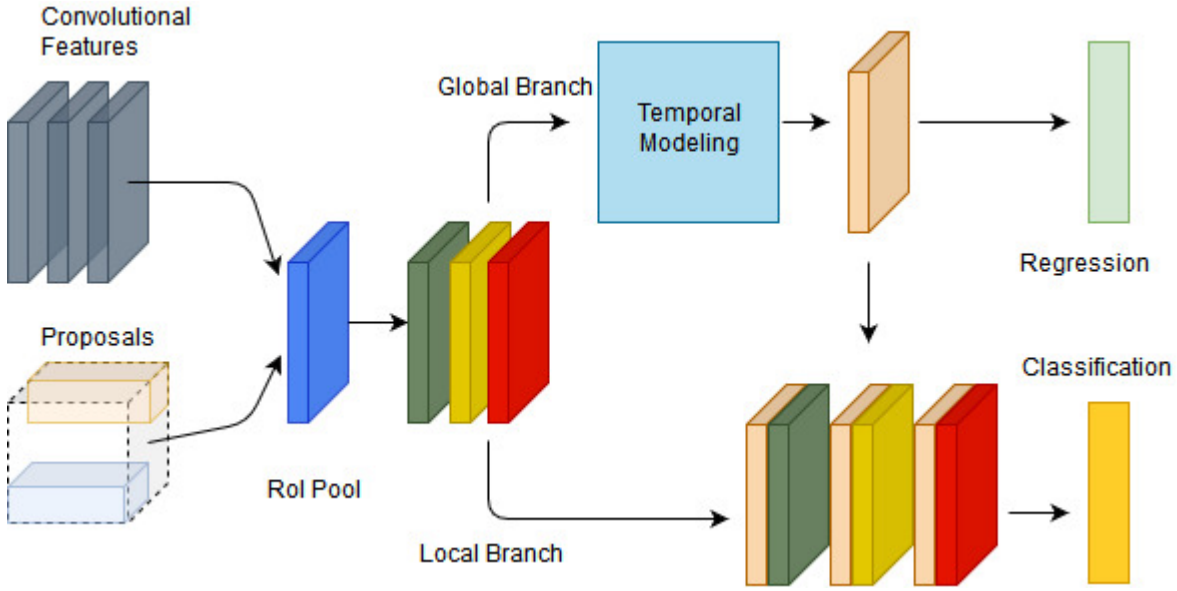
2. Comparison with the state-of-the-art methods on **AVA** by frame-mAP under IoU@0.5 (mailto:IoU@0.5). "*" means the results obtained by incorporating optical flow

Method	frame-mAP
Single Frame*	14.2
I3D	14.7
I3D*	15.6
ACRN*	17.4
STEP	**18.6**

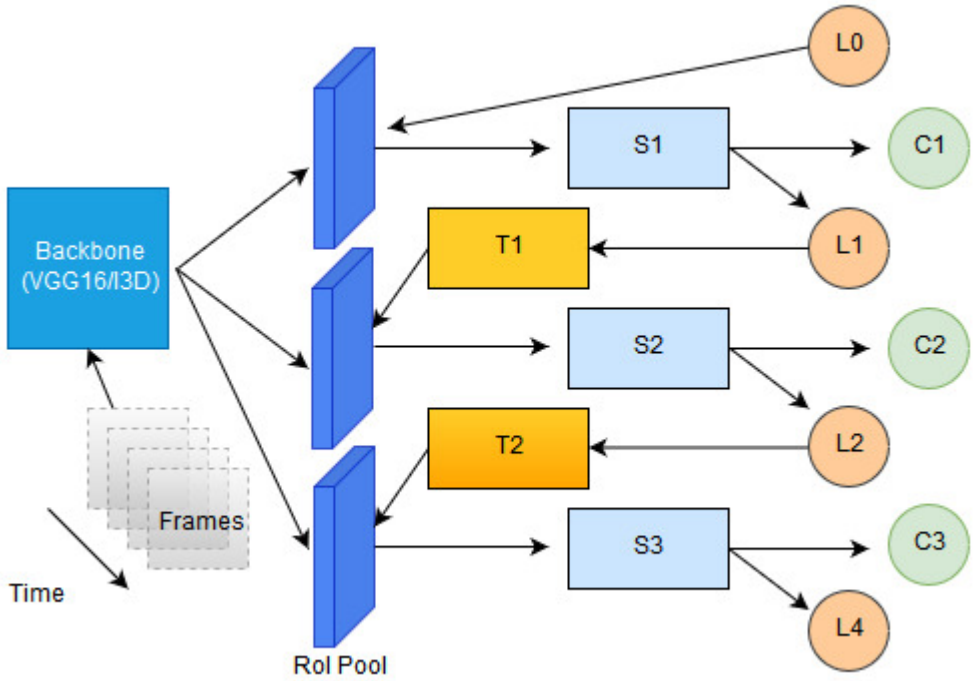
3. STEP runs at 21 fps using early fusion with 11 initial proposals and 3 steps on a single GPU, which is com-parable with the clip based approach (23 fps) and muchfaster than the frame based method (4 fps).

5.2 CNN visualization

Architecture of 2-branch network



STEP: progressive learning framework structure



Notations:

T - temporal extension

S - spatial refinement

C - classification

L - localization (L0 - initial proposals)

number - step

5.3 Experiment summary

The series of experiments were done based on a3_cifar10.ipynb (https://github.com/lyubonko/ucu2019/blob/master/assignments/a3_cifar10.ipynb (https://github.com/lyubonko/ucu2019/blob/master/assignments/a3_cifar10.ipynb)) using CIFAR10 dataset.

1. The initial model given in the notebook has the following structure:
   $[conv - relu - pool]x2 - [fc - relu]x3$
   with the initial parameters:

```
n_epoch = 5
batch_size = 4
lr=0.01
momentum = 0.9
optimizer = SGD+momentum
loss = nn.CrossEntropyLoss()
```

I tried to tune the parameters, however the obtained results in most cases were much worse than initial ones (seems the author spent enough time to pick-up such a splendit model). The intermediate results are shown bellow:

model	n_epoch	learning_rate	batch_size	optimizer	batch_norm	train_accuracy	test_accu
[conv-relu-pool]x2-[fc-relu]x3	5	0.001	4	SGD+momentum	-	-	
[conv-relu-pool]x2-[fc-relu]x3	10	0.001	4	SGD+momentum	-	-	
[conv-relu-pool]x2-[fc-relu]x3	5	0.01	4	SGD+momentum	-	-	
[conv-relu-pool]x2-[fc-relu]x3	5	0.0005	4	SGD+momentum	-	-	
[conv-relu-pool]x2-[fc-relu]x3	5	0.001	4	Adam	-	-	
[conv-relu-pool]x2-[fc-relu]x3	5	0.001	4	SGD+momentum	+	-	

model	n_epoch	learning_rate	batch_size	optimizer	batch_norm	train_accuracy	test_accu
[conv-relu-pool]x2-[fc-relu]x3	5	0.01	4	SGD+momentum	+	-	
[conv-relu-pool]x2-[fc-relu]x3	5	0.01	64	SGD+momentum	+	-	
[conv-relu-pool]x2-[fc-relu]x3	5	0.001	64	SGD+momentum	+	-	
[conv-relu-pool]x2-[fc-relu]x3	5	0.1	64	SGD+momentum	+	-	
[conv-relu-pool]x2-[fc-relu]x3	5	0.01	64	SGD+momentum	-	-	
[conv-relu-pool]x2-[fc-relu]x3	50	0.01	64	SGD+momentum	+	-	

After these series of experiments it could be proved that batch normalization positively influences accuracy (however, the batch_size should be big enough to see this changes).

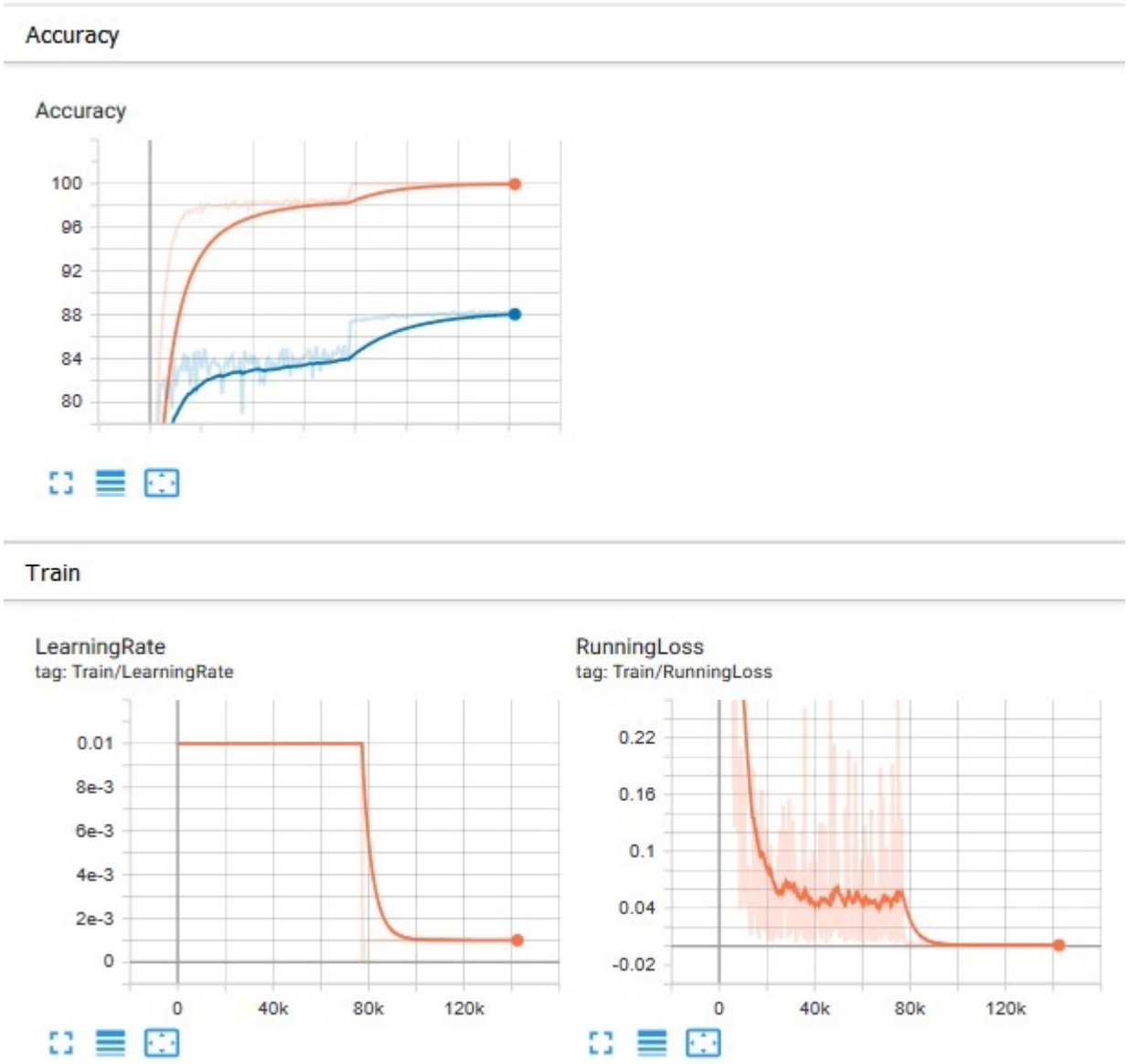
2. Having spent more then 10 hours only on parameters tunning using the original version of the notebook, I decided to rewrite a code a bit to be able to train the models using cuda in colab and check the accuracy on train and test dataset.

3. Assuming that parameter tunning would never provide with desirable "95% accuracy":D on cifar10 dataset, I decided to play with the famous CNN architecture such as VGG (16, 19), DenseNet and newly baked EfficientNet. The results are provided bellow:

model	n_epoch	learning_rate	batch_size	optimizer	batch_norm	train_accuracy	test_ac
efficientnet	5	0.1	32	SGD+momentum	+	55	
efficientnet	5	0.01	32	SGD+momentum	+	74	
efficientnet	5	0.001	32	SGD+momentum	+	70	
vgg16	5	0.1	32	SGD+momentum	+	64	
vgg16	5	0.01	32	SGD+momentum	+	82	

model	n_epoch	learning_rate	batch_size	optimizer	batch_norm	train_accuracy	test_ac
vgg16	5	0.001	32	SGD+momentum	+	88	
vgg16	5	0.1	**64**	SGD+momentum	+	**85**	
vgg19	5	0.1	32	SGD+momentum	+	52	
vgg19	5	0.01	32	SGD+momentum	+	78	
vgg19	5	0.001	32	SGD+momentum	+	87	
densenet	5	0.01	64	SGD+momentum	+	85	

4. To proceed with training on more n_epoch VGG16 was chosen, due to the speed and relatively one of the best performances after training on 5 epoch. (The same result was obtained for trained NN of Densenet architecture, however, the training was more time consuming). Here is the visualization of the training:



The schedule of learning rate was done using native torch function (learning rate starts with 0.01

value, and after 100 epoch, it decreseas to 0.001)

```
    optimizer = torch.optim.SGD(net.parameters(), lr=0.01, momentum=0.9,
 weight_decay=5e-4)
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=100,
gamma=0.1)
```

From the dependencies above we can see that having learning_rate = 0.01 the accuracy on the training dataset would come to plato of 84%. Since the trick with learning rate schedule was applied while traning NN, the it dramatically increased by 4 %. Thus the best obtained result of accuracy on test dataset is 88% (VGG16-architecture).

5. TODO (future):

```
 1) play around scheduling decresing learning rate over time
 2) add data augmentation
 3) check bigger batch size when batch norm applied
 4) use pretrained model
 5) check adviced in notebook architectures - to build a better feeli
ng of which blockes has more influence on model performance
```

Link to repo with experimental notebooks:
https://github.com/kasprova/CV_UCU/tree/master/module5/practice
(https://github.com/kasprova/CV_UCU/tree/master/module5/practice)