Deep Learning Project

May 20, 2020

1 Topic

In this project I will build a binary classification model and train it on a Google cloud VM to recognize the human action “Slipping” using the HMDB51 dataset and self-created videos. Then, I will compare and analysis performance among different classification model. In the end, the best one will be selected as final model for text video.

2 Dataset

2.1 Collect video

HMDB is a large human motion database, which is collected from various sources, mostly from movies, and a small proportion from public databases such as the Prelinger archive, YouTube and Google videos. The dataset contains 6849 clips divided into 51 action categories, each containing a minimum of 101 clips. Here I used stand, sit up, sit down videos as a part of training database.

Fig 1. HMDB51 dataset

Then, rest of videos are self-created videos, which contain slipping and not-slipping actions. The whole database has 668 video clips, including 430 negative clips and 238 positive clips. I simplified the slipping action clips, which makes the action more precise and generalized. Besides, I created videos by different mobile devices and different filming angle. The slipping action looks as Fig 2.

a) b) c)

Fig 2. Three types of slipping action

2.2 Extract rgb frames and TVL1 optical flow frames

Extracting rgb frames from video is easy and quick by OpenCV. For extracting TVL1 optical flow frames, I used the function of OpenCV – contrib. Based on cv2.optflow.DualTVL1OpticalFlow_create() and TVL1.calc(), I can extract optical flow frames, but it will take a very long time because of huge computation workload. The terrible running time will bring obstacles to the video test.

Therefore, I tried to find the method to accelerate this progress. Through reading the OpenCV document, I found an interesting class name UMat in OpenCV. A unified abstraction cv::UMat that enables the same APIs to be implemented using CPU or OpenCL code, without a requirement to call OpenCL accelerated version explicitly. These functions use an OpenCL-enabled GPU if exists in the system, and automatically switch to CPU operation otherwise. It becomes able to detect, load and utilize OpenCL devices and accelerated code automatically. All you need is changing Mat to UMat and a small adaptation to cv2 function. The time consumption comparison is shown as Fig 3. Obviously, UMat significantly improved processing speed. And the code is shown as follow.

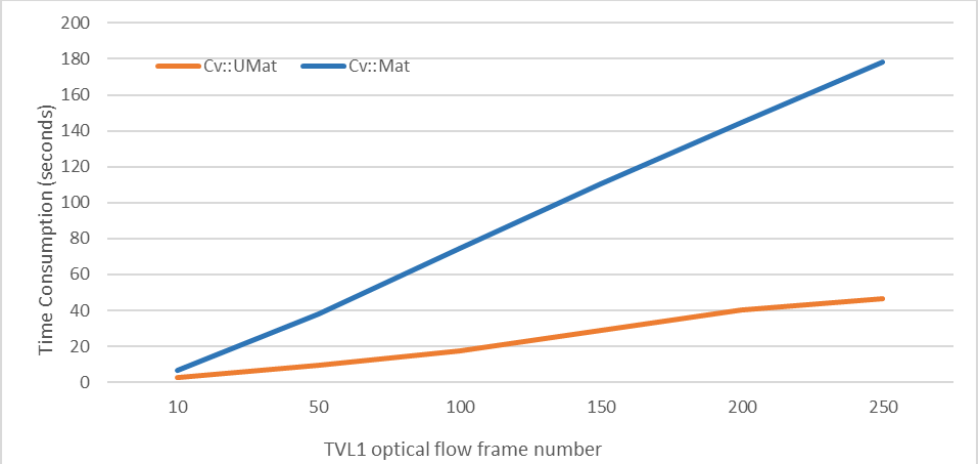


Fig 3. Time consumption comparison between UMat and Mat

```
def cal_for_frames(video_path, video_name, flow_path):
    frames = glob(os.path.join(video_path, '*.jpg'))
    frames.sort()
    if not os.path.exists(os.path.join(flow_path, video_name + '_u')):
        os.mkdir(os.path.join(flow_path, video_name + '_u'))
    if not os.path.exists(os.path.join(flow_path, video_name + '_v')):
        os.mkdir(os.path.join(flow_path, video_name + '_v'))
    prev = cv2.UMat(cv2.imread(frames[0]))
    prev = cv2.cvtColor(prev, cv2.COLOR_BGR2GRAY)
    for i, frame_curr in enumerate(frames[1:250]):
        curr = cv2.UMat(cv2.imread(frame_curr))
        curr = cv2.cvtColor(curr, cv2.COLOR_BGR2GRAY)
        tmp_flow = compute_TVL1(prev, curr)
        prev = curr
        cv2.imwrite(os.path.join(flow_path, video_name + '_u', "{:06d}.jpg".format(i+1)), tmp_flow[:, :, 0])
        cv2.imwrite(os.path.join(flow_path, video_name + '_v', "{:06d}.jpg".format(i+1)), tmp_flow[:, :, 1])
    return

def compute_TVL1(prev, curr, bound=15):
    """Compute the TV-L1 optical flow."""
```

```
    TVL1 = cv2.optflow.DualTVL1OpticalFlow_create()
    flow = TVL1.calc(prev, curr, None)
    flow = cv2.UMat.get(flow)
    assert flow.dtype == np.float32
    flow = (flow + bound) * (255.0 / (2 * bound))
    flow = np.round(flow).astype(int)
    flow[flow >= 255] = 255
    flow[flow <= 0] = 0
    return flow

def extract_flow(video_path, video_name, flow_path):
    cal_for_frames(video_path, video_name, flow_path)
    print('complete:' + flow_path + video_name)
    return
```

2.3 Improve Dataset

In the part 5, I have improved the dataset from 495 videos to 668 videos. And the best accuracy seems increase a little. However, if we look at pictures of model loss and accuracy, we will find that they are very chaotic. Although in some sample videos it got a beautiful time label, it is still an unstable model. Hence, I spent a long time trimming video frames so that they are able to achieve uniform specifications. During the process, I did two things. Firstly, make only one person is left in the frame. Secondly, convert all pictures to a width of 342 pixels and a height of 256 pixels. The process is shown as Fig 4.

Fig 4. Trimming processing

After that, deleting some not good video frames, which doesn't have any meaning in the picture. So the dataset changes as follow. And the model performance will be shown in the follow sections.

(Training video, Validation video) (357, 118) → (Training video, Validation video) (535, 133)

Next, increase dataset can solve two problems to a certain extent. First, detect videos which involve various sub-types of actions. Second, avoid detect be easily confused other actions. With this thought, I compare model with different dataset. The result is shown in Table 1.

Table 1. Performance comparison among different dataset

dataset	475	525	575	668
Spatial Accuracy	88.9831	89.3130	91.6084	96.2406
Motion Accuracy	92.3729	77.8626	81.8182	97.7444
Fusion Accuracy	94.0678	90.8397	93.0070	98.6425

From the Table 1, we will easily notice that 668 database has better performance in spatial accuracy, motion accuracy and fusion accuracy. Hence, I will use video_data_668 as the dataset for follow sections.

3 Two Stream Network vs Temporal Segment Network

3.1 Two Stream Network

In the previous work, I contributed the model based on two stream network. This network combines a spatial stream model and a motion stream model. Both models use the same CNN backbone. The difference between them is the input channel of network. Traditionally, the spatial stream uses rgb frames, so it is [1, 3, 224, 224]. The motion stream uses a stack of ten TVL1 optical frames, so it is [1, 20, 224, 224] for each video. However, be inspired by temporal segment network, I have revised the input of spatial stream model to three random rgb frames of each video, so the input size of network is [3, 3, 224, 224] for each video now. At first, the backbone network is resnet101. The architecture of resnet101 is shown as follow.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				

Fig 5. The architecture of resnet101

For selection of number of segment frame, I have compared separately extract 3, 5, 10 frames from video to build the model based on the old database. As a result, the trend of change of Acc and Loss is not very stable. Increase the number of segment frame is not really equal to achieve higher accuracy and lower loss. Hence, from the perspective of saving training time and enhancing model generalization, I choose 3 as the number of segment frame. And it also has a good classification performance in test video.

About categories of transfer learning, I chose finetuning the convnet instead of ConvNet as fixed feature extractor. Finetuning the convnet is to not only replace and retrain the classifier on top of the ConvNet on the new dataset, but to also fine-tune the weights of the pretrained network by continuing the backpropagation. The reason why I chose it is that my dataset is large and very different from the original dataset (ImageNet). Since the dataset is very large, I may expect that I can afford to train a ConvNet from scratch. However, in practice it is very often still beneficial to initialize with weights from a pretrained model. In this case, I would have enough data and confidence to fine-tune through the entire network.

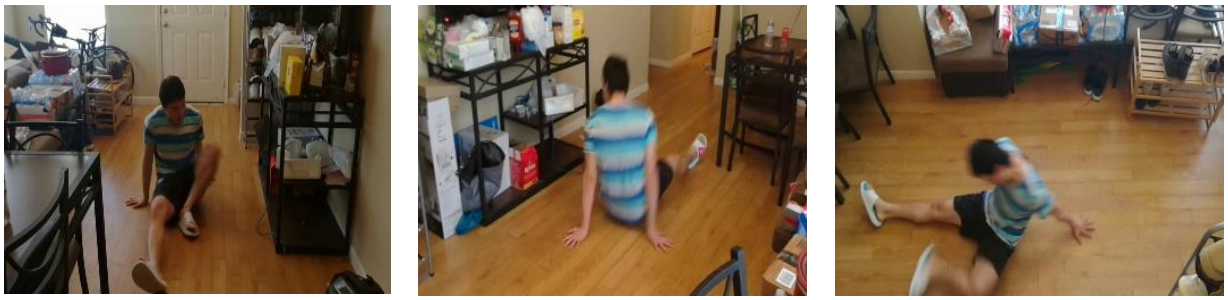
In part 5, I have tried to deploy multi-GPU for parallel model training, because I want to find whether large batch size can bring some benefits, such as improving parallelization efficiency, accelerating training speed, increasing accuracy and so on. I deployed 4 NVIDIA K80 GPU, and utilized the DataParallel tech of pytorch so that the batch size can be increased by 3 times. By contrary, too large batch size generates more terrible model, no matter in validation accuracy or example videos.

Besides, I have tried to change base CNN architecture from resnet101 to vgg16 in part 6. The performance comparison between resnet101 and vgg16 is shown as Table 2. Obviously, ResNet101 performs much better than VGG-16, no matter spatial accuracy or motion accuracy. The model based on

VGG-16 is not as good as [1] shown. There are several reasons, maybe the pretrained VGG-16 model in PyTorch is different from Caffe, maybe there are subtle bugs in my VGG-16 flow model.

Table 2. Performance comparison of VGG-16 and ResNet101				
Dataset	Video_475		Video_575	
Model	VGG-16	ResNet101	VGG-16	ResNet101
Spatial Accuracy	84.7458	88.9831	88.1356	91.6084
Motion Accuracy	62.7119	92.3729	64.3357	76.9231
Fusion Accuracy	86.4407	94.0678	79.0210	93.0070

In addition, we need to consider some special cases, such as bad illumination condition, only part of body in the video, same action from different angle and so on. For this cases, I have added more image transform during the model training. RandomCrop, ColorJitter, RandomRotation enhance the robustness of model. For example, after training, use new videos, which contain same action from different angle, to test model. The result is shown as Fig 6. It is not perfect for all angle, but for most of angle, it has a good performance.



a) 0.9824 b) 0.9756 c) 0.9852

Fig 6. Performance among different angle

Finally, I used average fusion accuracy between spatial stream model and motion model. The result is shown as Table 3.

Table 3. Performance of two stream network	
Spatial Accuracy	88.1356%
Motion Accuracy	92.3797%
Fusion Accuracy	94.9153%

3.2 Temporal Segment Network

After carefully reading the paper of temporal segment network, I found that it actually applied some interesting and useful tech methods to improve the performance of two stream network. The author didn't just propose the idea of temporal segment, he also contributed plenty of views for action recognition. Therefore, I attempt to add some model training tips as paper said into my model training.

Firstly, employ strategy called partial batch normalization. After initialization with pre-trained models, freeze the mean and variance parameters of all batch normalization layers except the first one. It can reduce the probability of over-fitting in the transferring process.

Secondly, exploit two new data augmentation techniques: corner cropper and scale jittering. In corner cropping technique, the extracted regions are only selected from the corners or the center of the image to avoid implicitly focusing on the center area of an image. In multi-scale cropping technique, present an efficient implementation of scale jittering. Fix the size of input image or optical flow fields as 256×342,

and the width and height of cropped region are randomly selected from {256, 224, 192, 168}. These cropped regions will be resized to 224 × 224 for network training. In fact, this implementation not only contains scale jittering, but also involves aspect ratio jittering. [2]

Thirdly, make use of RGB difference between two consecutive frames describe the appearance change, which may correspond to the motion salient region. Adding stacked RGB difference as another input modality.

In summary, we have three input modalities now. And each modality use the same backbone network. For each video, the input shape of tensor and output shape of tensor is shown in the Table 4. And the hyperparameters of model training is listed in the Table 5.

Table 4. Input shape and output shape for three modality			
Modality	RGB	OPT	RGBDiff
Input shape	[1, 9, 224, 224]	[1, 30, 224, 224]	[1, 48, 224, 224]
Output shape	[1, 2]	[1, 2]	[1, 2]

Table 5. hyperparameters of model training			
Modality	RGB	OPT	RGBDiff
Epochs	50	60	60
Batch size	24	16	12
Drop out	0.8	0.7	0.8
Learning rate	0.001	0.001	0.001
Momentum	0.9	0.9	0.9
Weight decay	0.0005	0.0005	0.0005

3.3 Compare different backbone architecture

The author of temporal segment network selected BNInception as the base network because of its good balance between accuracy and efficiency. With the continuous development of deep learning, many excellent deep learning models have emerged. Therefore, I implement some popular models and make a comparison among them. They are BNInception, ResNet101, ResNeSt101, InceptionV4, InceptionresnetV2. The InceptionresnetV2 is a deep learning model which combines residual module and inception module.

The ResNeSt101 is put forward this year, and the model and training code are released this month. It is a very fresh model. The author present a modular Split-Attention block that enables attention across feature-map groups. By stacking these Split-Attention blocks ResNet-style, he obtained a new ResNet variant called ResNeSt. [3] The paper declares that it significantly boosts the performance of downstream models at least 1% which based on ResNet. I read this paper recently and want to know if it is powerful as paper said.

The comparison result is shown in the Table 6. Obviously, ResNeSt101 is not as excellent as expected. The reason of that case maybe the training techniques I used cannot unleash all performance of this model. For example, I didn’t use rectified convolution layer and didn’t add too much groups for split attention blocks. But I think it is still a good try for model contribution.

Next, lets have a look at other models. The ResNet101 still have a good performance as before. Both the InceptionV4 and InceptionresnestV2 have better performance in the rgb frames. I think that is because their strong ability of understanding rgb image.

Table 6. Performance comparison among popular models					
Model	BNInception	Resnet101	Resnest101	InceptionV4	InceptionresnetV2
Rgb	92.46%	92.46%	86.47%	94.34%	93.18%
Opt	83.54%	91.70%	87.33%	86.31%	89.66%
Rgbdiff	85.61%	92.37%	90.55%	90.25%	92.72%

Then, I try to compute different average fusion accuracy to select best model. The result show in the Table 7. From the table we can see that the weighted average fusion accuracy is not equal to higher accuracy.

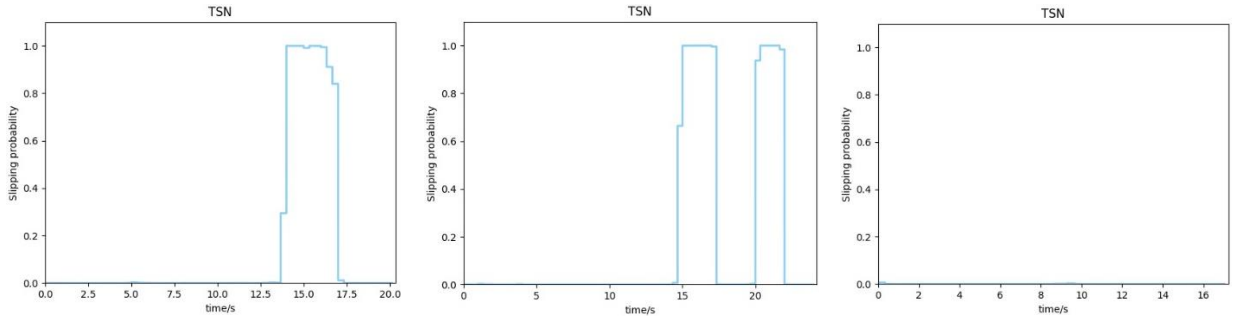
Table 7. Different average fusion accuracy					
Model	BNInception	Resnet101	Resnest101	InceptionV4	InceptionresnetV2
Rgb + Opt	91.5254%	94.0678%	89.8305%	94.9153%	94.9153%
Rgb + 1.5Opt	91.5254%	94.0678%	91.5254%	95.7627%	95.7627%
Rgb + Opt + Rgbdiff	90.6780%	83.8983%	94.9153%	93.2203%	94.9153%
Rgb + 1.5Opt + Rgbdiff	90.6780%	85.5932%	94.0678%	94.0678%	94.9153%

In summary, I choose InceptionV4 as the backbone architecture of my model. Because it is the second lightness model among the comparison and it has an excellent classification performance.

4 Improve model

However, does this model perform well in real videos? The answer is not. In fact, the accuracy of this InceptionV4 TSN model is even worse than resnet101 two stream model. Therefore, I came back to use resnet101 network as backbone of TSN. Besides, I improved the database in three ways. Firstly, increase the number of videos by adding more self-created videos. Secondly, simplify the slipping action clips, which makes the action more precise and generalized. Thirdly, creating videos by different mobile devices and different filming angle.

The result of accuracy of test dataset is shown before. Let' s look at the performance in real life videos. Obviously, the model has a good performance. The result is correct and accurate. Even the video contains two slipping actions, the time label can also be detected well. Therefore, this model is my final model.



a) positive slipping b) positive slipping c) negative slipping

Fig 7. Performance in real life videos

5 Conclusion

After improving database, revising model idea from two stream network to temporal segment network, adding more useful transform techniques, I have contributed a binary classification model for “Slipping”.

6 How to implement

6.1 Only test sample video

Run “test.sh” is enough.

6.2 Environment

VM: Google cloud deeplearning-vm

Framework: PyTorch-1.4

Based on: Debian GNU/Linux 9.11 (stretch) (GNU/Linux 4.9.0-11-amd64 x86_64\n)

Programming Language: Python 3.7 from Anaconda.

Machine type: n1-highmem-4 (4 vCPUs, 26 GB memory)

GPUs: 1 x NVIDIA Tesla K80

6.3 Prepare Steps

1. Clone code from github: git clone https://github.com/callmefish/TSN-simplify-pytorch.git

2. Download dataset and unzip them (if you don't want to train model, just ignore this step):

[video_data_668](https://storage.cloud.google.com/ucf101_for_rar/video_data_668.zip?authuser=1)

(gs://ucf101_for_rar/video_data_575.zip)(optional): A dataset of RGB frames and TVL1 optical flow frames extracted from 668 action videos.

3. Download sample video:

[sample_video](https://storage.cloud.google.com/ucf101_for_rar/sample_video.zip?authuser=1)

(gs://ucf101_for_rar/sample_video.zip): twelve sample videos.

4. Download model:

[668_resnet101_flow_model_best.pth.tar](https://storage.cloud.google.com/ucf101_for_rar/668_resnet101_flow_model_best.pth.tar?authuser=1)
(gs://ucf101_for_rar/668_resnet101_flow_model_best.pth.tar): Motion stream model.

[668_resnet101_rgb_model_best.pth.tar](https://storage.cloud.google.com/ucf101_for_rar/668_resnet101_rgb_model_best.pth.tar?authuser=1)
(gs://ucf101_for_rar/668_resnet101_rgb_model_best.pth.tar): Spatial stream model.

5. Before training model:

If you want to train your own model, you need to revise some 'parser.add_argument', such as train_list, val_list, root_path and so on. If you only need to test real life videos, being focused on win-test.py is enough.

6.4 Test model for your own

Run "test_model.py". You can change args in it.

7 Reference

[1] @misc{feichtenhofer2016convolutional,

title={Convolutional Two-Stream Network Fusion for Video Action Recognition},

author={Christoph Feichtenhofer and Axel Pinz and Andrew Zisserman},

year={2016},

eprint={1604.06573},

archivePrefix={arXiv},

primaryClass={cs.CV}}

[2] @misc{wang2016temporal,

title={Temporal Segment Networks: Towards Good Practices for Deep Action Recognition},

author={Limin Wang and Yuanjun Xiong and Zhe Wang and Yu Qiao and Dahua Lin and Xiaoou Tang and Luc Van Gool},

year={2016},

eprint={1608.00859},

archivePrefix={arXiv},

primaryClass={cs.CV}}

[3] @article{zhang2020resnest,

title={ResNeSt: Split-Attention Networks},

author={Zhang, Hang and Wu, Chongruo and Zhang, Zhongyue and Zhu, Yi and Zhang, Zhi and Lin, Haibin and Sun, Yue and He, Tong and Muller, Jonas and Manmatha, R. and Li, Mu and Smola, Alexander},

journal={arXiv preprint arXiv:2004.08955},

year={2020}

}