Action Recognition

Chao Ni

Department of Theoretical and Applied Mechanics
Peking University

November 11, 2018

Outline

Motivation

- The signal sequence can have different temporal evolution along the fundamentally same path.
- Take the video as an example, in two videos representing same motion with different play speed distribution, the displaying effect may be different.
- Under a standard time scale, the comparison between signals will be with computation complex O(N).

Simulation

- Before Reparameterization (click to play)
- After Reparameterization (click to play)

Variational Calculus

- The problem is to minimize the integral of [1]

$$J = \int_0^1 ||X'(\tau)||^2 \dot{\tau}^2 dt \tag{1}$$

it has been proved this optimization problem has a global optimal solution $\tau(t)$

- The unique solution can be expressed as

$$F(x^*) = \frac{1}{c} \int_0^{x^*} ||X'(\sigma)|| d\sigma = t \tag{2}$$

its inverse $\tau = F^{-1}(t)$ is then the mapping function.

Gradient Descent Method

- Our goal is to find the reparameterization function $\tau(t)$. To find the global optimal solution, we utilize the numerical methods and divide the integral parts into N steps first.

$$
\begin{aligned}
J &= \frac{1}{N}\sum_{i=0}^{N-1} ||X'(\tau_i)||^2 \dot{\tau_i}^2 \\
&= \frac{1}{N}\sum_{i=0}^{N-1} (\frac{||X(\tau_{i+1}) - X(\tau_i)||}{\tau_{i+1} - \tau_i})^2 (\frac{\tau_{i+1} - \tau_i}{dt})^2 \\
&= N\sum_{i=0}^{N-1} ||X(\tau_{i+1}) - X(\tau_i)||^2
\end{aligned}
\tag{3}
$$

Gradient Descent Method

- The derivatives of the cost function can be drawn as follows:

$$\frac{\partial J}{\partial \tau_i} = \frac{N}{2\delta}(||X(\tau_{i+1}) - X(\tau_i + \delta)||^2 + ||X(\tau_i) - X(\tau_{i-1} + \delta)||^2 - ||X(\tau_{i+1}) - X(\tau_i - \delta)||^2 - ||X(\tau_i - \delta) - X(\tau_{i-1})||^2) \tag{4}$$

and the update approach is:

$$\tau_i = \tau_i - \alpha_i \frac{\partial J}{\partial \tau_i} \tag{5}$$

with the learning rate α

Simulation Input

- We simulate the input as an array of moving points with a random distribution.

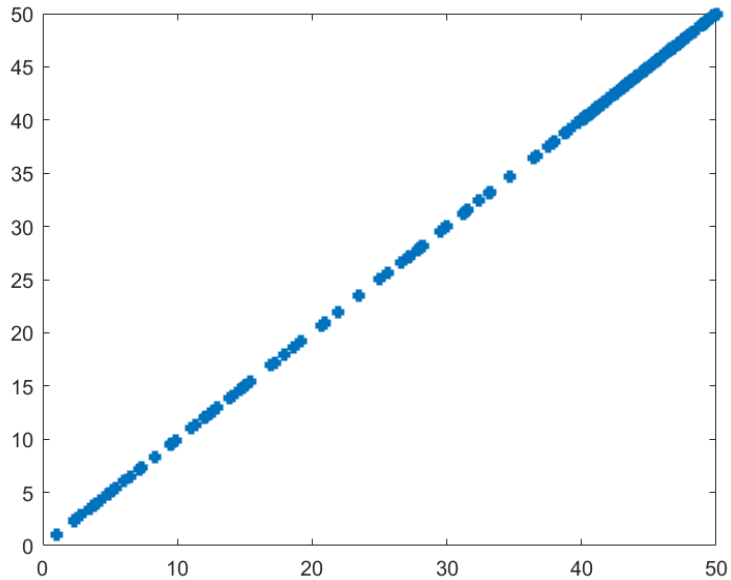
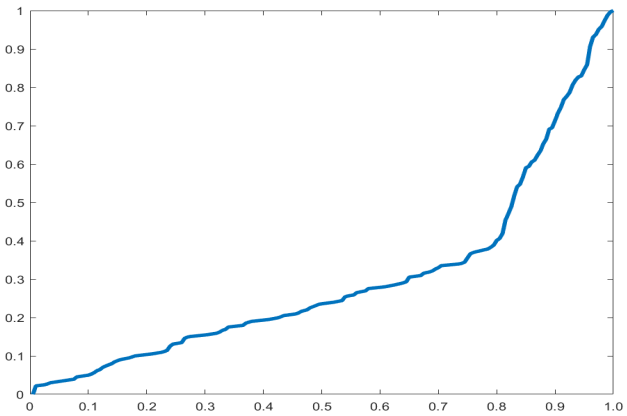
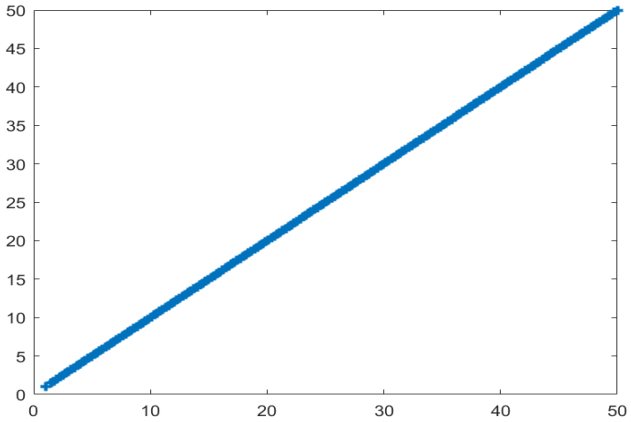


Figure: The simulation input signal, the sampled points distribute randomly, but are all on the same line.

Numerical Solution

- The reparameterization results forwarded by the numerical solution can be expressed as follows. The final cost is 0.0248.





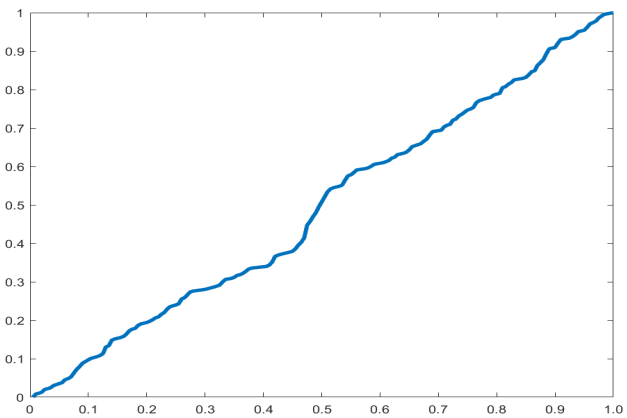
(a) The reparameterization function based on methods put forward in the paper: numerical result by "trapz" function embedded in MATLAB.

(b) The graph result under the reparameterization function.

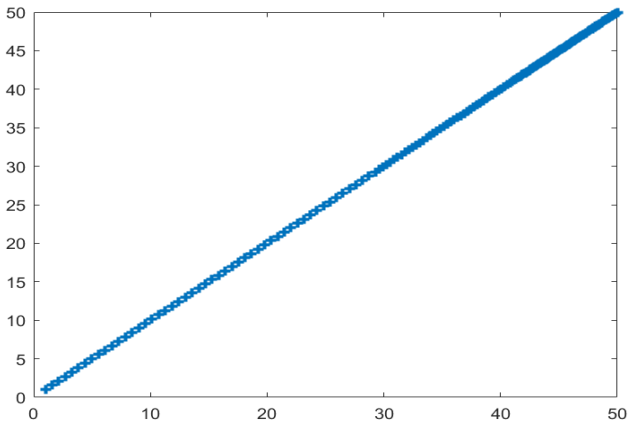
Figure: Results based on theory solution

Gradient Descent Method

- In the same time, I also plot the graph of the results derived by the gradient descent method, with a final cost 0.0044.



(a) The reparameterization function based on methods of gradient descent.



(b) The graph result under the reparameterization function.

Figure: Results based on gradient descent method

Cost-Iteration

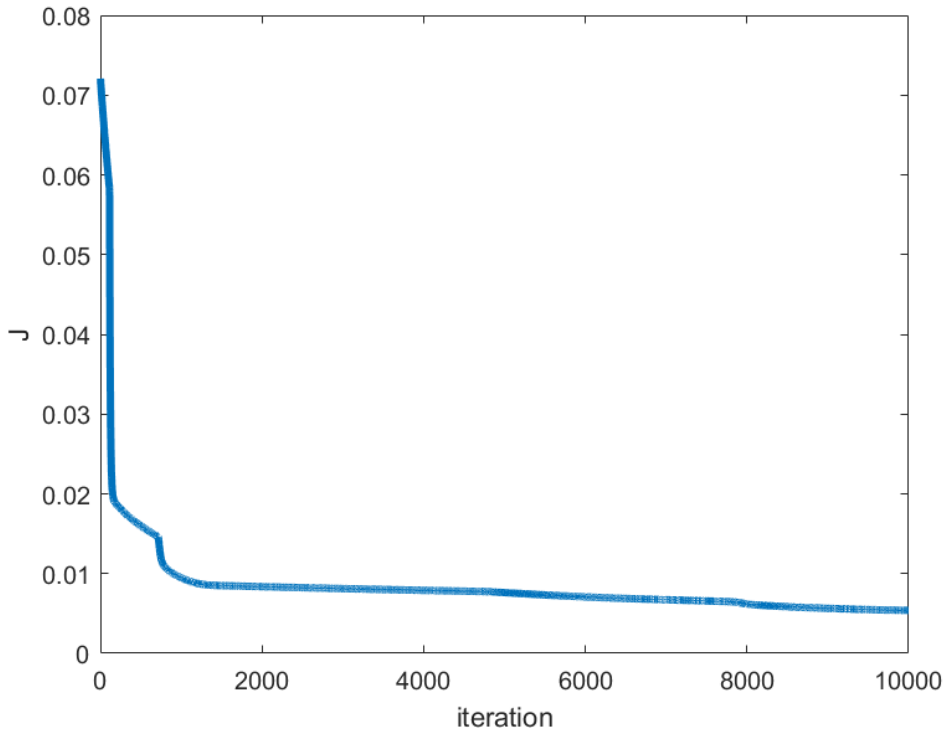


Figure: Cost function of J, it decreases as the iteration times increase with a final value 0.0044.

Comparison

- Fundamentally, the variational calculus with its numerical solution is correct, higher precision degree can be achieved by improvement in methods of integral and introducing more time steps.
- The gradient descent method can lead the solution fall into a local optimal. Because our initial mapping is randomly placed, and it turns out the change of the mapping is not large.
- If the initial signal sequence is nearly at a standard time scale, use gradient methods may be better as it achieve higher degree of computation precision.
- Gradient method can be used as a refinement to the numerical solution forwarded in the paper.

LRCN

A typical method for video action recognition is using architecture of long-term recurrent convolutional networks, which combines convolutional layers and long-range temporal recursion. [2] As is shown in Fig. 5

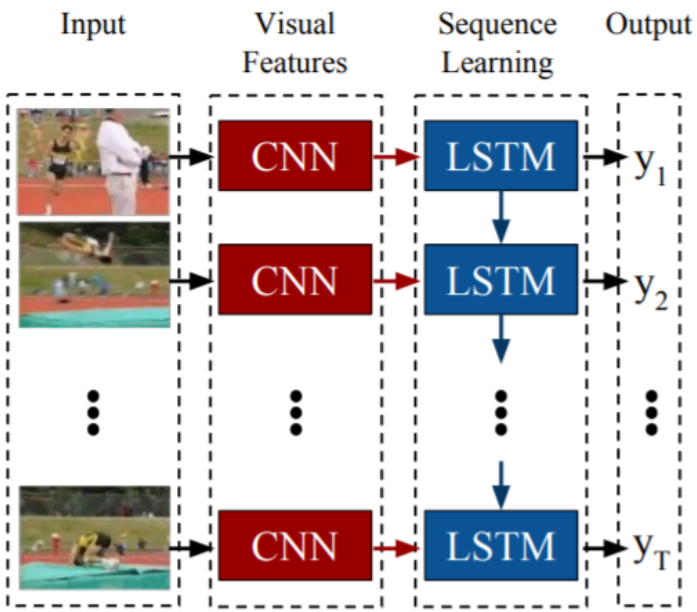


Figure: A typical action recognition network using long-term recurrent convolutional network

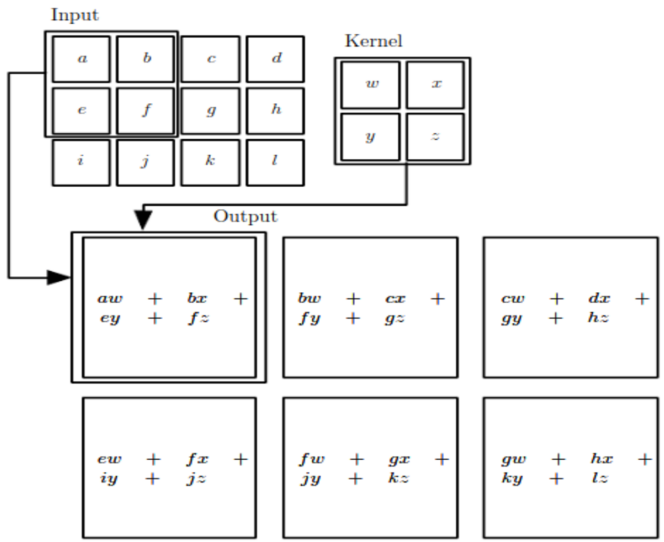
Temporal effect

- The two dimension convolutional operation can be expressed as follows:

$$S(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{6}$$

where I is the image input, K is the operation kernel and S the output. So in three-dimensional convolutional neural networks, the kernel has three dimensions.

- Here is the illustration of the convolution operation:



Model: C3D

- But one drawback of the naive LRCN model and other models like this is that, in practice, we don't make use of all frames of the total video. Once frames are chosen, the time gap between chosen frames are meaningless because this method destroy the temporal information after first convolutional layer. So the precision of LRCN is low compared to 3-dimensional convolutional neural networks.
- This paper [3] provides another way of action recognition for videos. As LRCN ignores the temporal information, this 3D convolutional networks add another dimension apart from 2D spatial features, the temporal dimension. The kernel then becomes a cube rather than a matrix. The model is as Fig. 6

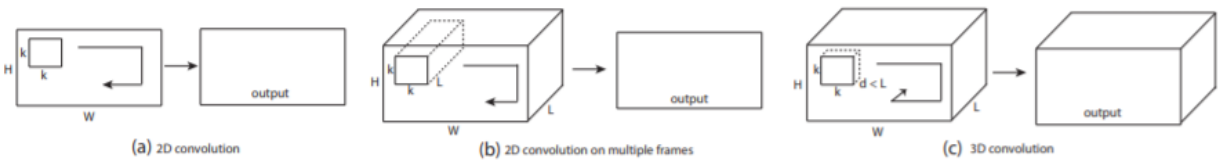


Figure: Applying 2D convolution results in an image, but applying 3D convolution results in a cube, thus preserving the temporal information.

dataset

- The training and testing code are basically from Github: https://github.com/kenshohara/3D-ResNets-PyTorch
- The software requirements are: Pytorch, OpenCV, GPU(the more, the better)
- The video dataset is from NTU dataset, as its background remains the same during the whole motion duration. http://rose1.ntu.edu.sg/datasets/actionRecognition.asp
- The video name is in the format of SsssCcccPpppRrrrAaaa (e.g. S001C002P003R002A013), for which sss is the setup number, ccc is the camera ID, ppp is the performer ID, rrr is the replication number (1 or 2), and aaa is the action class label. For now we mainly consider the label and the whole name as the video id.
- We need to extract out the information of the name and generate the annotation file for the model training first.

python files

- *extract.py*: extract out the video id(name of the video), label(from 1 to 60), convert video to frames(do some cropping and compressing space and speed consideration)
- *csv_generation.py*: generate the csv file containing label, video id, split name in list format.
- *number_to_list_name.py*: transfer the label name from number 1-60 to the strings, based on the motion list.
- *ntu_json.py*: To generate the annotation file for training the model.
- What we have already got:
  - *jpg_all* folder: Contain all jpg files converted from avi videos.
  - *ntu.json* file: Needed for training model.

problems

- The kinetics dataset is too big, it took me SIX hours to download a very limited number of videos (up to three videos for each label) The total dataset holds over 600GB volume of videos and the free space on the total thin6 server is 2.7TB...
- I trained on this small smaple, but found the training accuracy is very slow, and the loss keeps at a high level. I speculated that this was because there were too few samples and the model only learned a very specific feature, causing the fail of classification. (I have output the results after every epoch and found that every video in the same batch has almost the same output, it's like a randon output and no learning effect can be achieved.
- The problem of GPU volume. The paper has use a batch size of 128 to train the model, but the thin6 can hold up to a batch size of 16 if the frames are under the same cropping rule.

dataset

- We use the UCF101 data set, which is an action recognition data set of realistic action videos, collected from YouTube, having 101 action categories. This data set is an extension of UCF50 data set which has 50 action categories.
- We split the data set into training and testing set with 4-1 proportion.(It seems too small for the training set after my first training try)

c3d model

- we use *video_jpg_frame.py* to convert the videos to frames and put all frames under the folder *UCF101_frame*
- The model code is from the github: https://github.com/2012013382/C3D-Tensorflow-slim
- The neunual network has 8 convolutional layers and 3 fully connected layers. The dropout is intruduced in the network.

loss figure

- After 30 hours' training, we got a figure as follows:

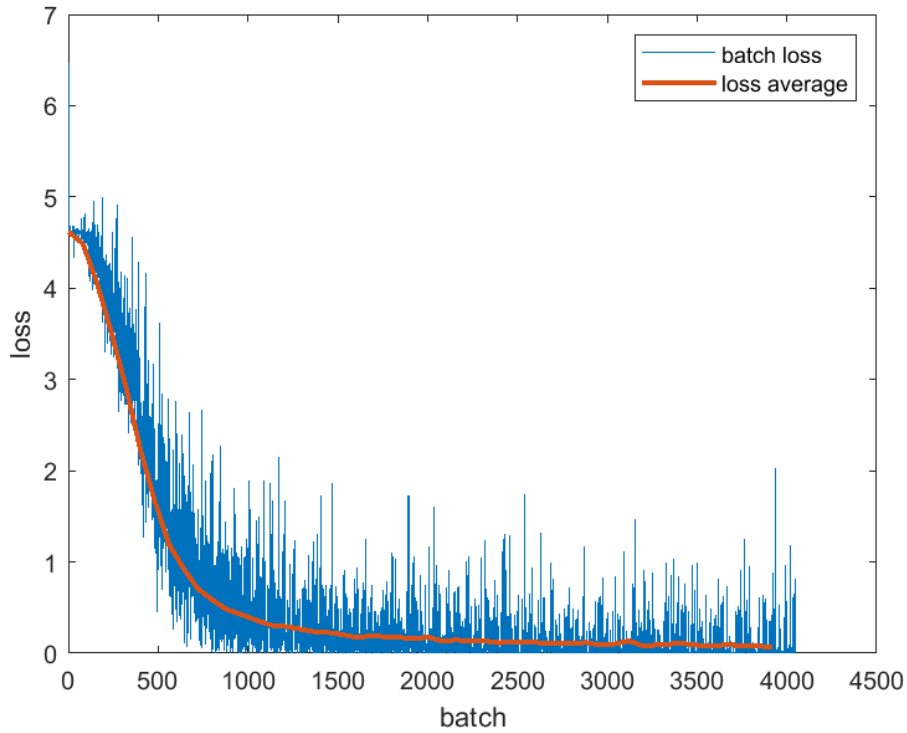


Figure: The x axis of the figure represents batch number, and the y axis the loss. We can see from the figure that the loss is decreasing when epoch/batch increases.

accuracy figure

- We plot the accuracy figure as follows:

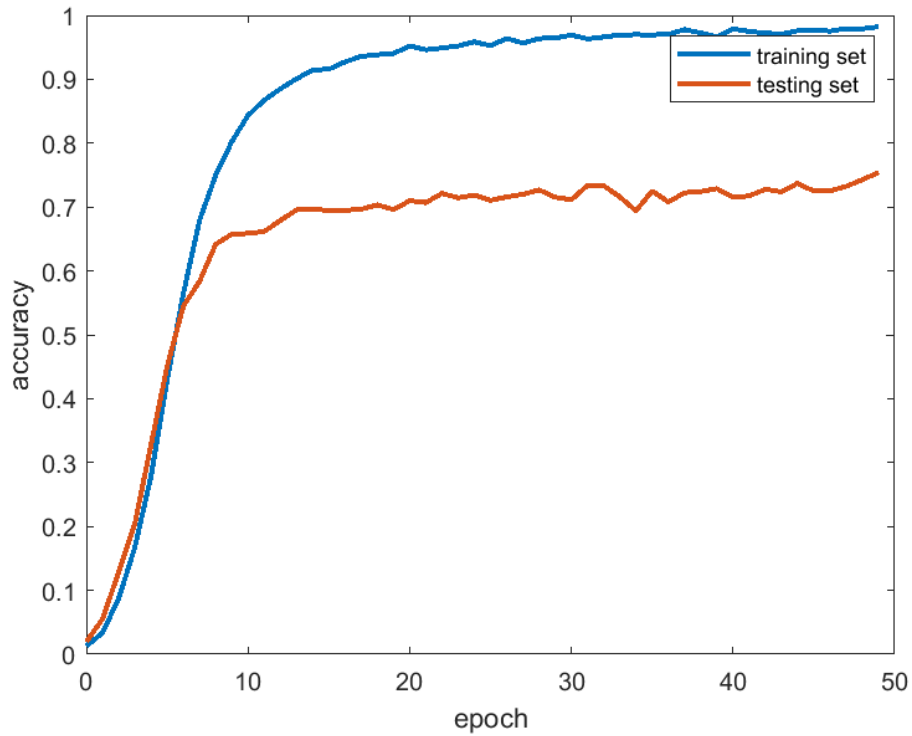


Figure: The blue line represents the accuracy of the training set while the yellow line the testing set. The final value of the training accuracy is 98.224%. but the testing accuracy is 75.423%. (The overfitting is severe)

Globally Optimal Reparameterization Algorithm (GORA)

- The problem is to minimize the integral of [1]

$$J = \int_0^1 ||X'(\tau)||^2 \dot{\tau}^2 dt \tag{7}$$

  it has been proved this optimization problem has a global optimal solution $\tau(t)$

- The unique solution can be expressed as

$$F(x^*) = \frac{1}{c} \int_0^{x^*} ||X'(\sigma)|| d\sigma = t \tag{8}$$

  its inverse $\tau = F^{-1}(t)$ is then the mapping function.

mapping example

- We take one video in the NTU as an example, it's under the label of "drinking water" and it has 250 frames in total. we divide the time period from 0 to 1 into 250 divisions, and give the corresponding mapping frame separately.

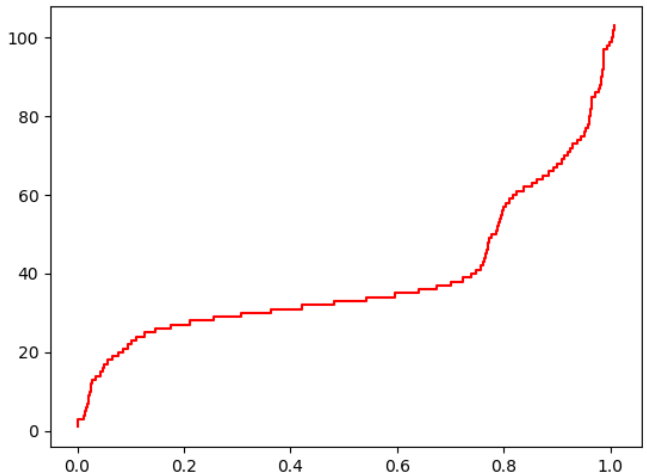


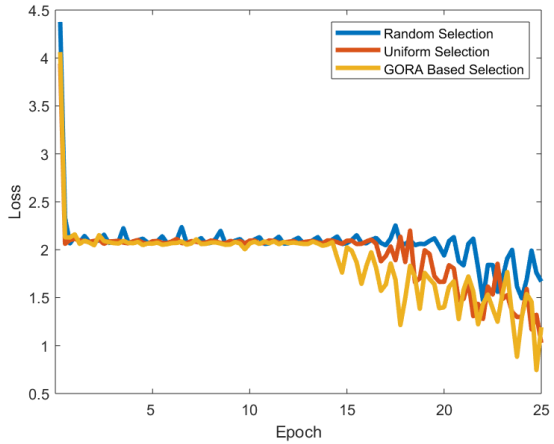
Figure: The x axis is the time projected into [0,1], the y axis represents the corresponding frame number. If we don't do the temporal reparameterization, it should be a straight line, and there is obvious difference after reparameterization.

frame sequence comparison

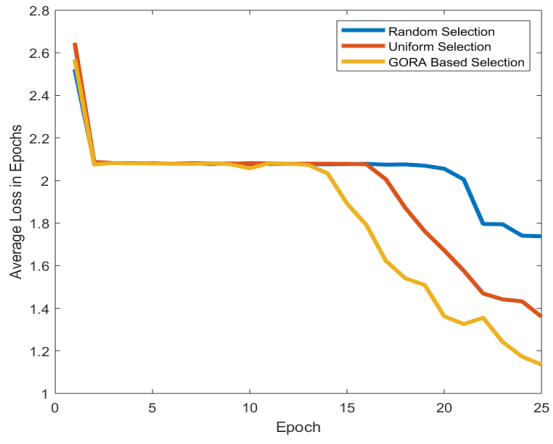
- In the projected time period, we chose 16 time instances uniformly distributed and chose the corresponding frame as the neural network input. For comparison, we plot two frame sequences as Fig. 10.

Figure: The frame sequence above represents the result input of the C3D architecture under the uniform selection method, the sequence below is the result of the reparameterization mapping. This figure shows the difference in the selected frames.

training performance comparison



(a) The loss figure along the training process in 25 epochs.



(b) The average loss along the training process in 25 epochs.

Figure: Loss tendency for three frame selection methods. We train the model for 25 epochs with batch size of 10. There are four bathes in every epoch. The experiment is reproduced several times and the result (tendency) is similar. Our GORA based selection achieves faster learning speed and higher recognition accuracy

training performance comparison

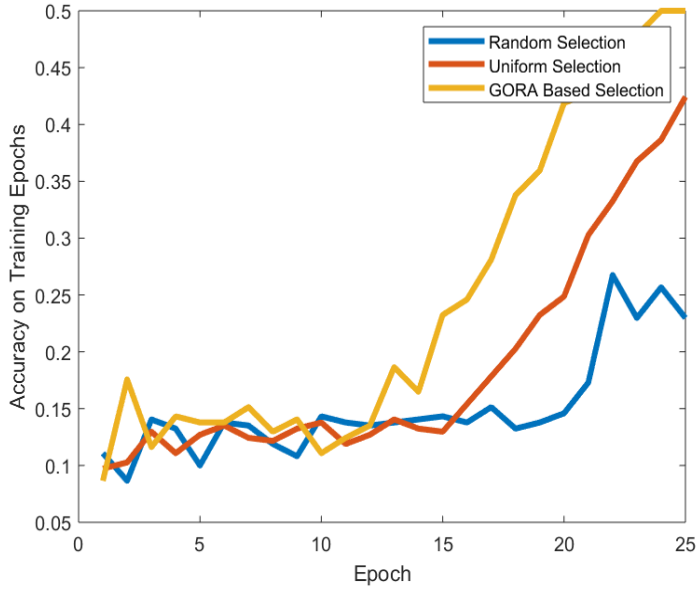


Figure: The tendency of training accuracy as the epoch increases. GORA based frame selection method achieve faster learning speed and higher accuracy in the first 25 epochs.

Fig. 12 shows the performance difference even more clearly. The GORA based frame selection method achieve better performance in training the model compared the other two.

training performance comparison

Background subtraction is also another efficient and broadly used method in dealing with video data. In our case, we apply our GORA preprocess mainly on the videos whose background remains still in the video periods, which means most of the information the frame contains are overlapped. We can then subtract the background, thus eliminating the redundant information and speeding up the training process.

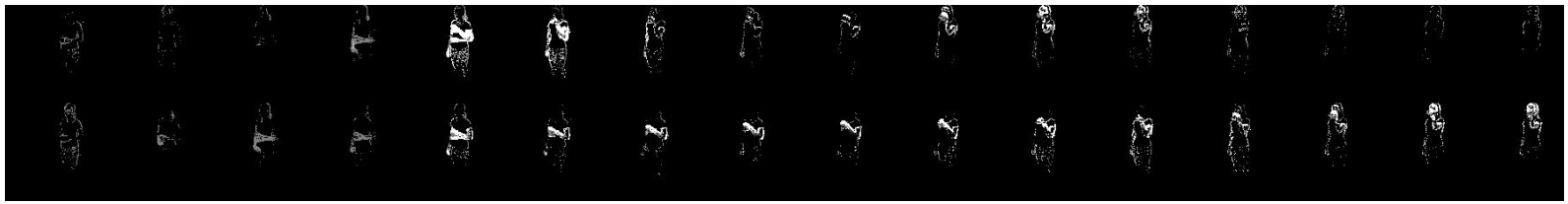


Figure: The frame sequence after the background subtraction.


Gregory S Chirikjian.
Signal classification in quotient spaces via globally optimal variational calculus.
arXiv preprint arXiv:1705.03744, 2017.


Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell.
Long-term recurrent convolutional networks for visual recognition and description.
In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634, 2015.


Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri.
Learning spatiotemporal features with 3d convolutional networks.
In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015.