Multi-view Recognition for Human Action Based on Deep Learning and K-SVD

Pengyu Zhang, Graduate Student, University of Florida

Abstract—**Action recognition is one of the active divisions in computer vision fields. However, most of the former researches are mainly based on one view instead of multi-view imaging processing. Multi-view recognition can provide more information about the action, but there are still existing problems. One of them is the relatively low accuracy of classification due to the different angles of view. This project implements a deep learning method and also provides a related dictionary of each action. Firstly, the image sequence will be processed by convolutional neural networks (CNN), and long short- term memory (LSTM). Secondly, the feature matrix obtained from the deep network will be further processed by the K-SVD algorithm to create the corresponding dictionary for each view. The sparse dictionaries provide good representation for the features of human action and help to decompose high dimensional features. The Softmax classifier is used for deep neural network classification.**

***Index Terms*—Multi-view recognition, Convolutional neural network, Long short term memory**

I. Introduce

MULTI-VIEW recognition is an attractive research division because the images recorded from different views contain multiple features. Compared with multi-view recognition, the classification performances obtained by only implementing single-view recognition methods on multi-view recognition tasks are relatively not ideal. So, it's meaningful to implement a deep neural network with the capability of dealing with multi-view classification.

During the past years, researchers have developed algorithms to deal with multi-view recognition base on their presented single-view method. They focused on extracting features around the essential points to create feature vectors such as HOG and did get good results. However, with the large datasets have been set up. The INRIA Xmas Motion Acquisition Sequences (IXMAS) Multi-View Human Action Dataset, for example, meet with challenges that they require substantial computational resources when processing the essential points for feature vectors.

To solve the problems mentioned above, CNN and LSTM will be used to extract features from the input images and provide results of classification. Then the K-SVD algorithm will be utilized to decompose features with a goal of generating a dictionary for each action. The dataset will be used is IXMAS [1]. It is a public dataset contains 11 human actions from 5 different camera views. The original data is collected synchronized in the time domain from all 5 cameras with 25 frames per second. The action includes: check watch, cross arm, scratch head, sit down, get up, turn around, walk, wave, punch, kick, pick up.

The second section explains the neural network architecture. The third section focuses on experiments and evaluations. The fourth illustrates the dictionary I obtain by implementing K-SVD. The last chapter concludes the result obtained from section three, followed by discussing existing problems in this project.

II. System Architecture

A. *CNN and LSTM Architecture*

Convolutional Neural Network (CNN) and Long Short Term Memory (LSTM) are implemented in this project. The architecture block diagram for each view (There are fibe different views) is shown in figure 1:

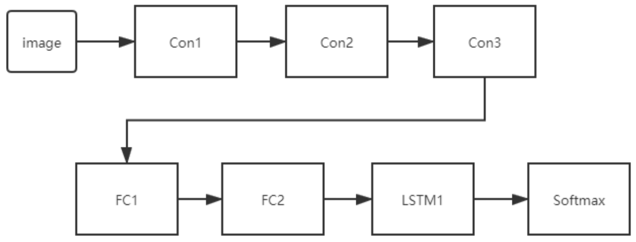


Fig. 1. CNN LSTM Architecture

It' clear to see that the CNN established by 3 convolutional layers (Con1, Con2, and Con3), followed by 2 fully connected layers (FC1, FC2). Then the output obtained from CNN will be passed to one LSTM layer for classification.

The input image is already preprocessed as a grayscale image with 64 x 48 pixels. The first convolutional layer is consists of 20 kernels, and each of them is 5 x 5. The stride is set as 1, followed by 0 paddings. It's easy to prove that a convolutional process on a c x m image with a kernel of size n x n under the condition that stride is 1 will produce an output image with a size of (c-n+1) x (m-n+1).

The first convolutional layer gives 20 channels output with a size of 60 x 44. A 2 x 2 max pooling will be further implemented on the output to generate a new output with a size of 30 x 22.

Then the output from Con1 is sent into Con2, established by 50 kernels with a size of 5 x 5. The same 2 x 2 max-pooling layer, as I implement in Con1, is utilized again. The Con2 will give an output image with a size of 13 x 9. The output image from Con2 is passed to the third convolutional layer with 50 channels kernels whose size is 2 x 2 instead to generate an output image with a size of 12 x 8. Then a third 2 x 2 max-pooling is implemented for the final process to create an image with a size of 6 x 4 (with a channel of 50). The output obtained from Con3 will be flattened and directly connected to the first fully connected layer (FC1) whose size is 1200, then a second full connected layer (FC2) with a size of 500 will be applied. The 1 x 500 vector is finally passed through the LSTM layer (LSTM1), whose size of hidden layers is set as 64. The last cell of LSTM is connected to another fully connected layer with a size of 11, which represents 11 different actions. Then a Softmax classifier is used to identify.

B. *Batch size of CNN and LSTM*

The system should have the capability to recognize different actions from a sequence of images. Therefore the batch size of the convolutional neural network is set as 50, which means the system process 50 images at a time. The batch size of CNN also has an influence on the input of LSTM. The formulas of LSTM is defined as follows:

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, X_t] + b_f\right) \tag{1}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, X_t] + b_i) \tag{2}$$

$$\tilde{c_t} = \tanh(W_c \cdot [h_{t-1}, X_t] + b_c) \tag{3}$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c_t} \tag{4}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, X_t] + b_o) \tag{5}$$

$$h_t = o_t \circ \tanh(c_t) \tag{6}$$

The f_t, i_t, o_t are the forget gate, input gate, output gate, respectively. h_{t-1} is the output at the current time t, which is precisely the output of the hidden layer in the memory cell. c_t represent the state of the cell at time t.

The choice of 50 as batch size will provide an output with a size of 50 x 1 x 500. Then the first dimension 50 indicates that the LSTM will implement 50 cells to transfer the memory of the former cell. Here I implement 1 as the value of the batch size of LSTM.

C. *Establishment of Dictionary*

After training the ConvNet & LSTM network, the feature vector of each action will be recorded. The next step is to fuse the feature vector as one feature matrix and further implement K-SVD algorithm to build a sparse representation for every action. The block diagram of feature fusion is shown in figure 2:

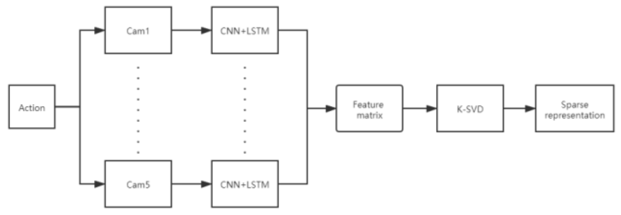


Fig. 2. Feature fusion and sparsification

The goal of K-SVD is to design an overcomplicated dictionary D. Each column of the dictionary is called an atom. A signal Y can be represented as Y = DX, which is the linear combination of these atoms. X is defined as the sparse coefficients matrix. The formula explains the algorithm is as follows:

$$\min_{D,X} \|Y - DX\|_F^2, s.t\ \forall \|X_i\| \leqslant T_0$$

The K-SVD algorithm Python code I used is open-source code. The link will be presented at the reference [2] of this project.

III. Experiments Design

A. *Data preprocessing*

The original data downloaded directly from IXMAS cannot be directly utilized as training data. There exist two problems. One is the frames of different videos vary a lot. Another is that considering the computational resources of my laptop, It’s not wise to use all of the data. In this project, I only select 122 videos from each camera. I firstly take advantage of OpenCV to split the videos into images. Because the frame of different videos might be different, I set up a threshold to automatically delete videos whose amount of frames is under 50, which is designed as the batch size of my neural network. And I also noticed that there exist some videos whose number of frames is more than 50, so I delete those videos as well. Finally, the dataset consists of 10 actions, and each action contains 93 videos, each video contains 50 frames. I split the dataset at a ratio of 1/3 to generate training data and testing data. Then the original is processed as grayscale images with mean subtraction and divided by stand variance.

B. *Experiments and evaluation*

I implement my code on Python 3.7.7. The neural network architecture is established based on PyTorch under the environment of Anaconda. The learning rate is set as 0.00005. I begin with only implementing a dropout value of 0.5 at the first fully connected layer. Here I only show the figures obtained from View 3. The learning curve and accuracy are shown in figure 3 and figure 4, respectively:

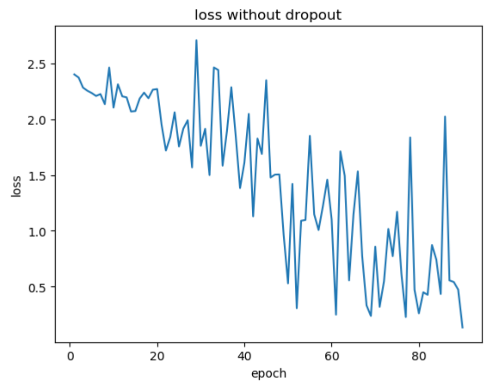
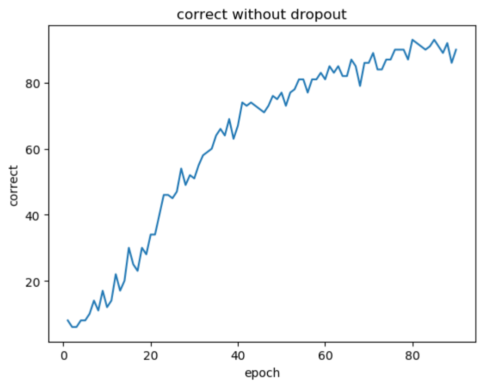


Fig. 3. Learning Curve

The value of learning curve fluctuates because I set the batch size of the input of LSTM as 1, which will always cause a non-smooth curve. But It’s clear to see that the trend of the loss is decreasing.



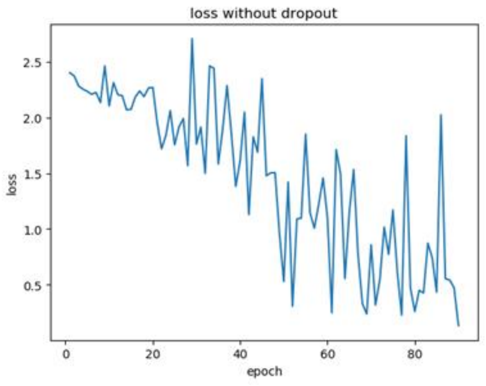


Fig. 4. Accuracy

The accuracy curve shows that the neural network keeps learning information with the increasing of epoch.

The results obtained from the last epoch are shown in table I :

TABLE I
TRAINING

View	Training Epoch	Training Accuracy
View1	90	0.9618
View2	90	0.9728
View3	90	0.9462
View4	90	1
View5	90	1

The results from training dataset look pretty good. However, the performance of the model on testing dataset is rather poor as table II shows:

TABLE II
TESTING

View	Training Accuracy
View1	0.03
View2	0.03
View3	0.06
View4	0.03
View5	0

The poor performances obtained from the testing dataset strongly indicate that the training process is overfitting. So I am starting thinking of the methodology of avoiding overfitting in a relatively big neural network. I have tried several methods: 1. Batch normalization with L2 norm; 2. Adding dropout to ConvNet architecture.

Due to the limitation of computational resources of my personal computer, and I need to train 5 different deep neural networks, I only experiment the method 2 with an epoch value of 90. I additionally add one dropout layer before the activation function of ConvNet. Then I implement a value of 0.3 as the dropout rate. The results from the training dataset and testing dataset are as follows:

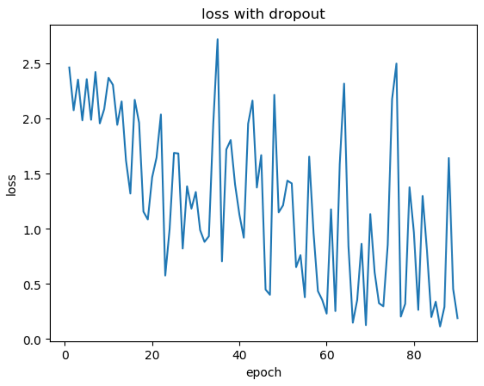


Fig. 5. Learning Curve with dropout rate = 0.3

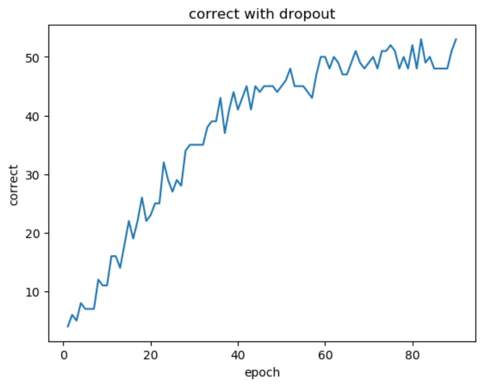


Fig. 6. Accuracy with drop out rate = 0.3

The learning curve remains similar properties as I did in the former experiment. However, a new problem occurs, which is the convergence of accuracy. Figure 6 shows that the accuracy already converges to around 55% at the epoch of 60. The testing results is as follows:

TABLE III
TESTING DROPOUT RATE = 0.3

View	Training Accuracy
View1	0.06
View2	0.09
View3	0.19
View4	0.06
View5	0.03

The performance of these five models on testing data is much better than the results in Table II, but it’s still not ideal. The new dropout rate helps to generalize the model. More experiments are needed to find the optimal dropout rate, as well as the trade-off between the cost of computational resources and accuracy.

IV. Dictionary Representation

Figure 7 shows the features extracted from the ConvNet and LSTM network and the reconstruction from the generated dictionary. Each action will be represented as a 64 x 5 matrix, each column of which is obtained from one view. The picture on the left-hand side is the original feature of action 1, the image on the right hand is the reconstructed version obtained from the corresponding dictionary with a value of 300 as the number of atoms and sparse coefficients.

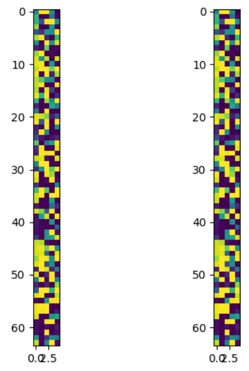


Fig. 7. Reconstruct from dictionary

V. Conclusion

A. *Performance of ConvNet and LSTM*

This project is finished basically under the guidance of Prof. Kulathumani's paper [3]. Additionally, I implement K-SVD to provide related sparse representation.

The deep neural network combined with CNN and LSTM can help to process a time sequence signal, which is specifically the action represented by one video consists of 50 sequential frames in this project. CNN is utilized as feature generator, followed by an LSTM network. The LSTM plays the role of trying to remember the state of the former hidden layers and adjust the current output. The property of utilized long term memory is appropriate for a time series input.

B. *The existing problem of my project*

The main problems is the contradiction between the high accuracy on training data and the low accuracy of testing data. This problem occurs when the network is overfitting. But in practice, increasing the dropout rate will decrease the convergence speed, which requires a higher capability of computation. The change of dropout rate may also cause underfitting if not selected properly.

C. *What I need to do in the future*

Based on the results of my experiments by now, I should keep thinking of modifying my code, that is, adjusting the dropout rate to approach better training results as well as better performance on the testing dataset. Or I could implement batch normalization to see whether the strategy works.

From the perspective of accelerating the training process, which bothers me a lot, I should seek for cloud server which can provide richer computational resources. The problem now I face is that every training process is time-consuming. Trying to modify the hyperparameters to obtain multiple results which can be used to evaluate the performance

Another work I should do in the future is analyzing the prediction accuracy of each view. In this project, I focus on the overall accuracy of different views instead of each action. Analyzing the accuracy of each action requires a larger dataset; that is to say, it requires a higher capability of computational resources.

References

[1] Weinland, Daniel and Ronfard, Remi and Boyer, Edmond Free Viewpoint Action Recognition using Motion History Volumes. Computer Vision and Image Understanding (CVIU), Volume 104, Number 2-3, 2006

[2] https://cloud.tencent.com/developer/article/1386025

[3] Rahul Kavi; Vinod Kulathumani; Fnu Rohit; Vlad Kecojevic J. of Electronic Imaging, 25(4), 043010 (2016). https://doi.org/10.1117/1.JEI.25.4.043010