SKELETON BASED CHEO'S MOVEMENT RECOGNITION

Vu Huu Huan, Nguyen Van Hung, Tran Hai Long

Abstract: Cheo is a form of generally satirical musical theatre, often encompassing dance, traditionally performed by Vietnamese peasants in northern Vietnam and has fostered the national spirit through its lyrical content. However, the new Vietnamese generations have less care about this art, what need to carefully preserve. With the development of Deep Learning recently, especially Graph Convolution Networks, we aim to leverage the power of AI to recognize Cheo movements based on skeleton data. Through this task, we believe that it can open more future works like recognize Cheo movements in videos, combine more based movements to compose a new complicated movements, dance… which can help to preserve this beautiful art. In this project, we apply Spatial Temporal Graph Convolution Networks (ST-GCN) for this recognition task on our dataset.

1, Introduction

Practically, graph data is so popular in the real world, such as social networks, web link data, molecular structures, geographical maps, etc, so after graph network have introduced, it has received the attention of research society immediately due to their ability of representing the real world in a fashion that can be analysed objectively. Due to the expressiveness of graphs and a tremendous increase in the available computational power in recent times, a good amount of attention has been directed towards the machine learning way of analysing graphs, i.e. Graph Neural Networks and Graph Convolution Networks.

Hence, human action recognition based on skeleton data have become an active research area in recent years. Some new algorithms have been developed and reach impressive results. Because of that, we have decided to apply these algorithms for cheo's movement recognition. And the chosen algorithm is Spatial - Temporal Graph Convolution Networks (ST-GCNs) - what have introduced in 2018 and reach high accuracy on NTU-RGB+D datasets (81.5%, 88.3% on the cross-subject and cross-view benchmarks, respectively)

Cheo movements naturally can be represented by a time series of human joint locations, in the form of 3D coordinates. In our dataset, the movements are represented by 23 different human joints, each joint have the information about 3D locations. However, the scale of our dataset is too small (just have totally 210 samples and 38 classes), so in this project, we just focus on clearly explaining ST-GCN algorithms and how to implement it with Tensorflow instead of trying to reach high performance.

The rest of the report is organized as follows. In section 2, we briefly summarize some related work on ST-GCN and skeleton based

human action recognition. In section 3, we clearly explain about ST-GCN. Our experiments and the detail of the dataset are shown in section 4. The conclusion is in section 5 with some discussion for the future work.

2, Related work

Skeleton data is widely used in action recognition. Numerous algorithms are developed based on two approaches: the hand-crafted-based and the deep-learning-based. The first approach designs algorithms to capture action patterns based on the physical intuitions, such as local occupancy features, temporal joint covariances. On the other hand, the deep-learning-based approach automatically learns the action features from data. Obviously, graph-based approach is a branch of deep-learning-based approach with the initial idea is Graph Convolution Networks.

In this section, we cover the background material necessary for the rest of the report.

2.1, Graph Convolution Networks

As the name suggests, Graph Convolution Networks (GCN), draw on the idea of Convolution Neural Networks re-defining them for the graph domain. Given a graph *G*, a GCN takes as input:

- An input feature matrix V x F^0 , **X**, where V is the number of nodes, and F^0 is the number of features in each node.
- An V x V matrix representation of the graph structure such as the adjacency matrix **A** of G.

A hidden layer in the GCN can be written as $H^i = f(H^{i-1}, A)$ where $H^0 = \mathbf{X}$, and *f* is a propagation. Each layer $\mathbf{H^i}$ corresponds to a V × F^i feature matrix where each row is a feature representation of a node. At each layer, these features are aggregated to form the next layer's features using the propagation rule f.

A simple propagation rule. Let's consider the following very simple form of layer-wise propagation rule: $f(H^i, A) = \sigma(A H^i W^i)$, where σ is a non-linear activation function like the ReLU, W^i is the weight matrix for the i-th layer, A is the adjacency matrix of the graph G. Multiplication with A means that, for every node, we sum up all the feature vectors of all neighboring nodes but not the node itself. However, this propagation rule have 2 major limitation: the aggregated representation of a node does not include its own features; and nodes with large degree will have large values in their feature representation while nodes with small degrees will have small values, this can cause vanishing or exploding gradients.

To move beyond such limitations, we use the propagation rule introduced by Kipf & Welling at ICLR 2017 :

$$f(H^i, A) = \sigma(D^{-1/2}\hat{A}\,D^{-1/2} H^i W^i),$$

where $\hat{A} = A + I$, with I is the identity matrix having same shape with A (V x V), and D is the diagonal node degree matrix of $\hat{A}$. Adding an identity matrix I to A mean adding a self-loop to each node, this thing can help to resolve the first above problem, while multiply $D^{-½}$ with $\hat{A}$ mean normalizing A such that all rows sum to 1, hence we can avoid the second limitation.

In summary, if input of a GCN layer have shape V x F^0 , the output will have shape V x F^1 with F^1 equal the number of used filters in

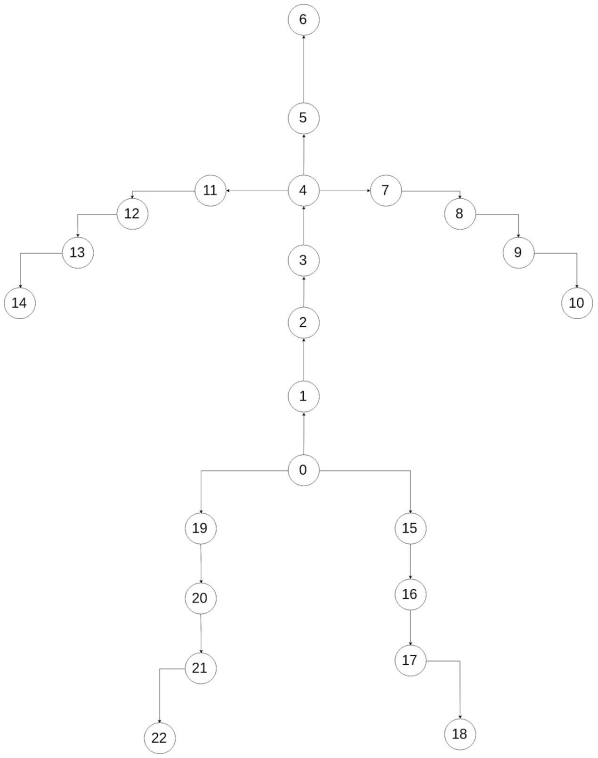

Figure 1: Skeleton graph construction

this layer. In other words, weight matrix have shape F^0 x F^1.

2.2, Skeleton Graph Construction

A skeleton sequence is usually represented by 2D or 3D coordinates of each human joint in each frame. In our dataset, the skeleton graph construction include 23 joints as shown in figure 1.

We construct the spatial temporal graph on the skeleton sequences in two steps. First, the joints within one frame are connected with edges according to the connectivity of human body structure. Then each joint will be connected to the same joint in the consecutive frame.

Overall, we represent a skeleton sequence by a four dimensional matrix with shape in_channels x T x V x M, where in_channels is the number of features in a node/ joint, T is the number of frames, V is the number of nodes in a frame and M is the number of subject/ human in sequence (in our case, M always equal one).

3, Spatial - Temporal Graph Convolution Networks (ST-GCN)

3.1, Pipeline overview

Skeleton based data can be obtained from motion-capture devices or pose estimation algorithms from videos. Usually the data is a sequence of frames, each frame will have a set of joint coordinates. Given the sequences of body joints in the form of 3D coordinates, we construct a spatial temporal graph with the joints as graph nodes and natural connectivities in both human body structures and time as graph edges. The input to the ST-GCN is therefore the joint coordinate vectors on the graph nodes. Multiple layers of spatial-temporal graph convolution operations will be applied on the input data and generating higher-level feature maps on the graph. After that, we apply a Global average pooling layer to this feature map, so the model can work with even different length sequence samples. It will then be classified by the standard SoftMax classifier to the corresponding action category. The whole model is trained in an end-to-end manner with backpropagation. We will now go over the components in the ST-GCN model.

3.2, Spatial Graph Convolution Networks

Before we dive into the full-fledged ST-GCN, we first look at the graph CNN model within one single frame. In this case, on a single frame, there will be V joint nodes, each node have in_channels features. Difference with natural images, which can be

treated as 2D grid, and the number of neighbor pixel always be the same when we use a K x K kernel on the image at any pixel of image, the skeleton graph data can be treated like that. The problem is when we slide the kernel from node 0 having 3 neighbor node is node 1, 15, 19 to the node 1 having just 2 neighbor node (node 0, 2), the number of neighbor nodes be changed. Because of that, we can not apply standard convolutional kernel on graph data. What we need now is a new **sampling function** to define what node is neighbor node, and a new **weight function** to map a node in the neighborhood to its subset label.

First, let's consider **sampling function**. On graphs, we can define the sampling function on the neighbor set

$B(v_{ti}) = \{v_{tj} \mid d(v_{ti}, v_{tj}) <= D\}$ of a node v_{ti}. Here $d(v_{tj}; v_{ti})$ denotes the minimum length of any path from v_{tj} to v_{ti}. Thus the sampling function p: can be written as $p(v_{ti}, v_{tj}) = v_{tj}$. In this work we use D = 1 for all cases, that is, the 1-neighbor set of joint nodes.

Second, we will define a new **weight function.** For graphs like the one we just constructed, there is no such implicit arrangement. So instead of giving every neighbor node a unique labeling, we simplify the process by partitioning the neighbor set $B(v_{ti})$ of a joint node v_{ti} into a fixed number of K subsets, where each subset has a numeric label. Thus we can have a mapping l_{ti}:

$$B(v_{ti}) \rightarrow \{0, \ldots, K-1\}$$

The simplest case is when we use Uni-labeling partitioning strategy, we will map all neighbor nodes of root node to just one label, it still mean that we just multiply all neighbor nodes to the same weight. We will discuss several partitioning strategies later.

Spatial Graph Convolution. Combine both sampling function and weight function, we can compute the output when convolve a node v_{ti} with the kernel by this equation:

$$f_{out}(v_{ti}) = \sum_{v_{tj} \in B(v_{ti})} \frac{1}{Z_{ti}(v_{tj})} f_{in}(v_{tj}) \cdot \mathbf{w}(l_{ti}(v_{tj})).$$

Spatial Temporal Modeling. Back to our data, the input will have temporal dimension, and to aggregate information of consecutive frames, we extend the concept of neighborhood to also include temporally connected joints as

$$B(v_{ti}) = \{v_{qj} | d(v_{tj}, v_{ti}) \leq K, |q - t| \leq \lfloor \Gamma/2 \rfloor\}.$$

Where the parameter Γ controls the temporal range to be included in the neighbor graph and can thus be called the temporal kernel size. To complete the convolution operation on the spatial temporal graph, we also need the sampling function, which is the same as the spatial only case, and the weight function, or in particular, the labeling map l_{ST} :

$$l_{ST}(v_{qj}) = l_{ti}(v_{tj}) + (q - t + \lfloor \Gamma/2 \rfloor) \times K,$$

where $l_{ti}(v_{tj})$ is the label map for the single frame case at v_{ti}.

For example, when we convolve the kernel on node 5 at 2-nd frame, this node will be a root node. Suppose we use Uni-labeling partitioning strategy, and K = 1, Γ = 2, so $l_{ti}(v_{25})$ will be 0 and $l_{ST}(v_{25})$ will be 1, $l_{ST}(v_{15})$ will be 0 and $l_{ST}(v_{35})$ will be 2.

Completed Spatial Temporal Convolution Block. Combine all above things, we will have a ST-GCN blocks, each block contain a

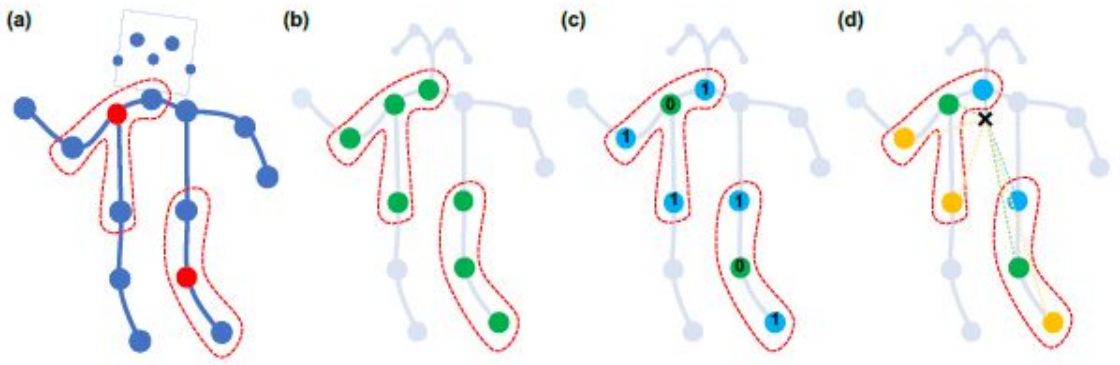

Figure 2: The proposed partitioning strategies for constructing convolution operations. From left to right: (a) An example frame of input skeleton. Body joints are drawn with blue dots. The receptive fields of a filter with D = 1 are drawn with red dashed circles. (b) Uni-labeling partitioning strategy, where all nodes in a neighborhood has the same label (green). (c) Distance partitioning. The two subsets are the root node itself with distance 0 (green) and other neighboring points with distance 1.(blue). (d) Spatial configuration partitioning. The nodes are labeled according to their distances to the skeleton gravity center (black cross) compared with that of the root node (green). Centripetal nodes have shorter distances (blue), while centrifugal nodes have longer distances (yellow) than the root node.

spatial graph convolution followed by a temporal convolution, which alternatingly extract spatial and temporal features.

3.3, Partition Strategies

There are 3 common partitioning strategy: Uni-labeling; Distance partitioning; Spatial configuration partitioning as show in figure 2. As we mentioned above, how to label the neighbor node is really importance, if we just label both neighbor node and root node a same label (Uni-labeling partitioning strategy), the model can not learn that are root node or neighbor node more useful to recognize action. The same issue happen with the second partitioning strategy: Distance partitioning, model just can learn that are root node or neighbor node more importance to recognize action, but in practice, the nodes located near human's center have the contribute differ than the nodes located far human's center. It's why we need the third strategy and it still be the most recommended strategy - Spatial configuration partitioning. In this strategy, we will compare the distance of the neighbor nodes to the joints seem be the center of body (in our case, we choose node 2 is the center) with the distance of the root node to the center. If the distance of neighbor node is longer than the distance of root node, this neighbor node will have label equal 2, and 1 in the other case, the root node always has label equal 0. Formally, it can be represented by following equation:

Number of total samples	Number of classes	Number of samples per classes	Number of frames per samples	Number of subjects/ actors
210	**38**	**~5.5**	**100 ~ 300**	**1**

Table 1. Summary of the dataset

$$l_{ti}(v_tj) = \begin{cases} 0 & \text{if } r_j = r_i \\ 1 & \text{if } r_j < r_i \\ 2 & \text{if } r_j > r_i \end{cases}$$

3.4, Learnable edge importance weighting

Although joints move in groups when people are performing actions, one joint could appear in multiple body parts. These appearances, however, should have different importance in modeling the dynamics of these parts. In this sense, we add a learnable maskMon every layer of spatial temporal graph convolution. The mask will scale the contribution of a node's feature to its neighboring nodes based on the learned importance weight of each spatial graph edge.

With this change, the spatial graph convolution is,

$$\mathbf{X}_{\text{out}} = \sum_{p \in \mathcal{P}} \mathbf{M}_{\text{st}}^{(p)} \circ \widetilde{\mathbf{A}^{(p)}} \mathbf{X}_{\text{in}} \mathbf{W}_{\text{st}}^{(p)},$$

where

$\widetilde{\mathbf{A}^{(p)}} = \mathbf{D}^{(p)^{-\frac{1}{2}}} \mathbf{A}^{(p)} \mathbf{D}^{(p)^{-\frac{1}{2}}}$ is the normalized adjacent matrix for each partition group, $\circ$ denote the Hadamard product, M_{st} and W_{st} are trainable weights for each partition group to capture edge weights and feature importance, respectively.

4, Experiments

4.1, Dataset

To collect skeleton movements data, we wore 23 special devices on actor to track the movements of 23 different joints while she was performing. These devices track the 3D locations of joints. The table 1 will show the summary about our dataset.

4.2, Experiment setup

Our experiments have been conducted using Python programing-language on Google Colab.

4.3, Result

To build the model for skeleton Cheo's movement recognition, we use 10 ST_GCN layers to extract high level feature maps, following by a Global average pooling and fully connected layer with the number of nodes equal the number of classes (38 classes). The used optimizer is Adam with default settings.

Because our dataset is too small to spilt into training and validation data, so we just train the model on the total dataset. After 10 epochs, the model quickly reach 75.2% training accuracy, but may be it cause overfitting.

5, Conclusion

In conclusion, we explore a common graph-based algorithm - ST-GCN - to Cheo's movement recognition. This algorithm has some advantages like low computational cost with pretty high performance; can leverage the link between nodes in the graph. However, it still have some disadvantages. The global average pooling make model can not learn anything about the length of samples. Moreover, if the length of data is large, these approaches turn out to be computationally infeasible.

To resolve these advantage, we can recommend a solution like adding the dilation to kernel on temporal dimension, it can help solving the second disadvantage when make the kernel become more parse.