Skeleton Based Activity Recognition Using GCNs

Göksel Mert Çökmez
ETH Zürich
mcoekmez@student.ethz.ch

***Abstract*—This report details the pipeline of a Graph Convolutional Network called EfficientGCN and its adaptation to the hand skeleton as captured by Microsoft HoloLens. HoloLens and EfficientGCN is deployed in tandem to perform action recognition from the 3D pose of the human hand. This action recognition is performed in the context of a surgery room for tools used in spine surgeries. This report first discusses the pipeline of EfficientGCN and how it was modified in order to complete the task at hand. It then goes into the certain core ideas and data structures that are employed for this model to work, before benchmarking the action recognition accuracy across different configurations of the model. Lastly, the performance of the model is discussed with potential solutions recommended for its shortcomings.**

I. Introduction

Human pose and action recognition is a field on the rise, with novel methods and use cases coming into existence with an increasing rate. While there are already numerous methods for pose estimation such as conventional RGB based methods. However, skeleton based methods prove to be much more reliable since the location of the joints can be represented as a time series and thus we can make inferences using the dynamic changes in our data.

As a result, this project aims to establish a skeleton based method for robust action recognition using the captured hand pose data. When recording the hand pose data, most of the heavy lifting is done by Microsoft HoloLens, which offers accurate coordinates for the 26 joints defined in the human hand.

Thus, one of the main tasks of this project is choosing an already well functioning Graph Convolutional Network (GCN) based solution and adapting it for use in human hands instead of the whole body. In order to achieve this, the main candidate is MMSkeleton [2], which is a part of the open-mmlab project by the Multimedia Laboratory of The Chinese University of Hong Kong. However, due to compatibility and dependency issues with Windows 10, which the operating system for the workstation at hand, we opted to pursue another implementation of a GCN solution.

Our following candidate is EfficientGCN [3] as its dependencies work flawlessly with Windows 10. Moreover, compared to other State-Of-The-Art (SOTA) methods, EfficientGCN uses fewer parameters to achieve similar accuracy scores. This is especially helpful in reducing training time on the NTU RGB+D 60 dataset [1], which serves as a benchmark for numerous human pose recognition models.

The final step in the implementation of this project is recording various hand poses when using surgical equipment related to spine surgery, and converting the HoloLens data to the NTU dataset format, which is the original format used in training EfficientGCN. Once the dataset is converted to the NTU format, the joint indices and their relevant adjacency matrices are modified in order to accommodate the hand model seen in Figure 1, compared to the human skeleton joints in Figure 2.

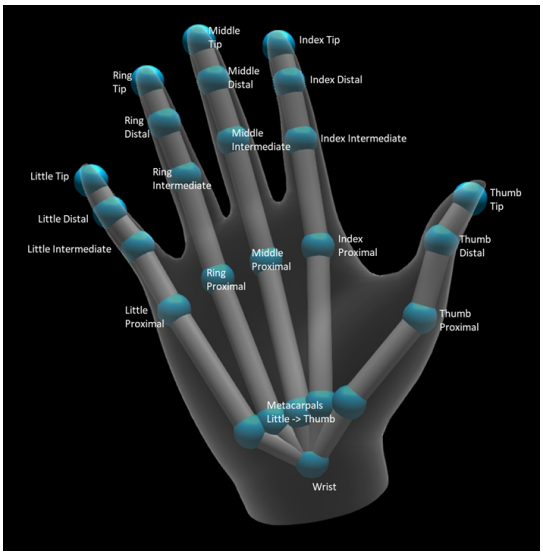

Fig. 1. Hand joints as captured by Microsoft HoloLens

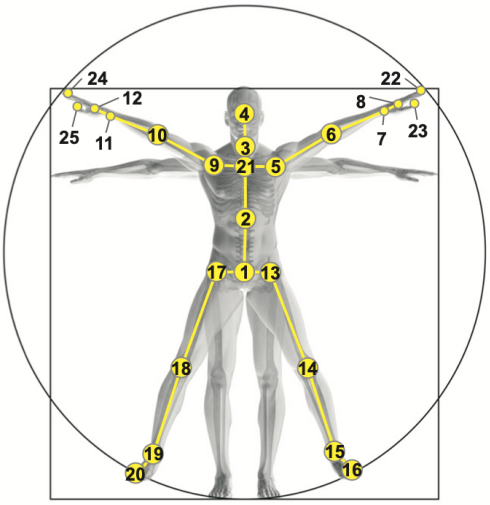

Fig. 2. Human skeleton joints as stated in the NTU RGB+D dataset [1]

II. Models and Methods

The main structure of this project stems from the pipeline of EfficientGCN. As a result, understanding the underlying architecture of EfficientGCN is imperative in order to explore the functionality of this project. Thus, we start with data preprocessing, alongside the explanation of the NTU format and the conversion to it and the form of the graph which diverge from the original implementation of the source code.

A. Data Preprocessing

In this implementation of EfficientGCN, the input is divided into three channels: one for joint positions, one for joint velocities and finally, one for bone features. We start off by normalizing the positions of each joint relative to a predefined center joint with index c. In the original implementation, the joint number 2 of the NTU skeleton is chosen as the center. For our hand pose recognition purposes, we normalize by the wrist index. We define the non-normalized joint coordinates $\mathcal{X} = \{x \in \mathbf{R}^{C_{in} \times T_{in} \times V_{in}}\}$ where C_{in} is the coordinate, T_{in} is the frame index and V_{in} is the joint index. Then, we can define the normalized coordinates $\mathcal{R} = \{r_i | i = 1, 2, 3, ..., V_{in}\}$ as:

$$r_i = x[:, :, i] - x[:, :, c] \tag{1}$$

The concatenation of the normalized and non-normalized 3D coordinates $\mathcal{X}$ and $\mathcal{R}$ form our first input of joint positions. We then go on to form two sets of velocities $\mathcal{F} = \{f_t | t = 1, 2, 3, ..., T_{in}\}$ and $\mathcal{S} = \{s_t | t = 1, 2, 3, ..., T_{in}\}$ by the following equations in order to form our velocity input:

$$\begin{aligned} f_t &= x[:, t+2, :] - x[:, t, :] \\ s_t &= x[:, t+1, :] - x[:, t, :] \end{aligned} \tag{2}$$

Consequently, we form the velocity input by concatenating the matrices $\mathcal{F}$ and $\mathcal{S}$. Last but not least, we need to form the bone feature input. For that, we define the bone length matrix $\mathcal{L} = \{l_i | i = 1, 2, 3, ..., V_{in}\}$ and the bone angles $\mathcal{A} = \{a_i | i = 1, 2, 3, ..., V_{in}\}$. The elements of these matrices are calculated as:

$$\begin{aligned} l_i &= x[:, :, i] - x[:, :, i_{adj}] \\ a_{i,w} &= arccos(\frac{l_{i,w}}{\sqrt{l_{i,x}^2 + l_{i,y}^2 + l_{i,z}^2}}) \end{aligned} \tag{3}$$

with i_{adj} denoting the adjacent joint to i and w denoting the 3D coordinates.

B. NTU Dataset Format

The NTU RGB+D is a popular dataset used to benchmark numerous skeleton based human pose estimation algorithms. It is published by the Rapid-Rich Object Search (ROSE) Lab of Nanyang Technological University. The dataset contains 1-4 people in each frame performing certain actions, alongside their associated 3D joint coordinates, RGB image captures and depth maps. Our main focus for our implementation is the files with `.skeleton` extension, which contain the 3D joint coordinates captured by Microsoft Kinect. These files are essentially text files with the total number of frames in the very beginning of the files with the following layout for each frame:

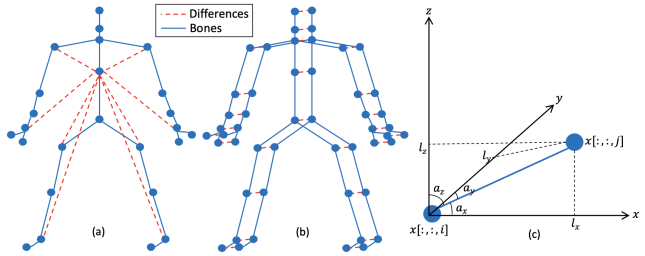

Fig. 3. The above figure indicates how the motion, velocity and bone feature inputs are calculated respectively.

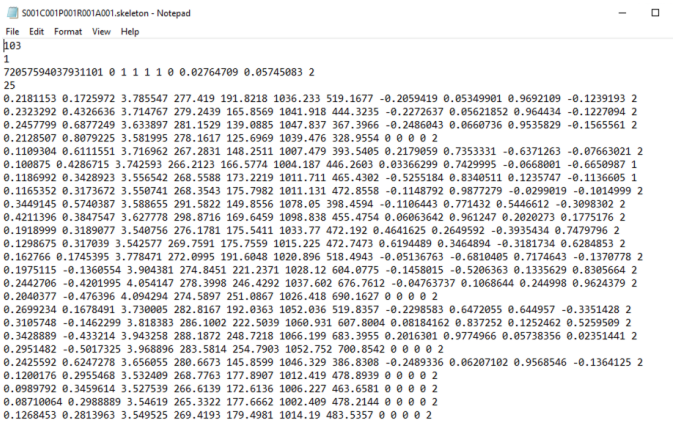

Fig. 4. NTU dataset format. The first line indicates the number of frames in this recording. The second line is the number of skeletons in that specific frame, with the third line acting as the identifier as the skeleton ID. Then the file goes on to list the number of joints visible in each frame and their coordinates.

As it can be observed in the NTU format, there are 11 coordinates associated with each joint. These stem from the recording format of Kinect v2, and they correspond to (in order):

- 3D x coordinate
- 3D y coordinate
- 3D z coordinate
- RGB camera 2D x coordinate
- RGB camera 2D y coordinate
- Depth camera x coordinate
- Depth camera y coordinate
- Quaternion w coordinate
- Quaternion x coordinate
- Quaternion y coordinate
- Quaternion z coordinate

However, a large part of this coordinates are redundant. Our script in HoloLens captures uniquely the 3D x, y and z coordinates, which is very useful since these three coordinates are the only ones used by EfficientGCN. Thus, when converting

the JSON file generated by the HoloLens to the NTU format, we end up with a similar layout, albeit with 26 joints and uniquely the 3D coordinates.

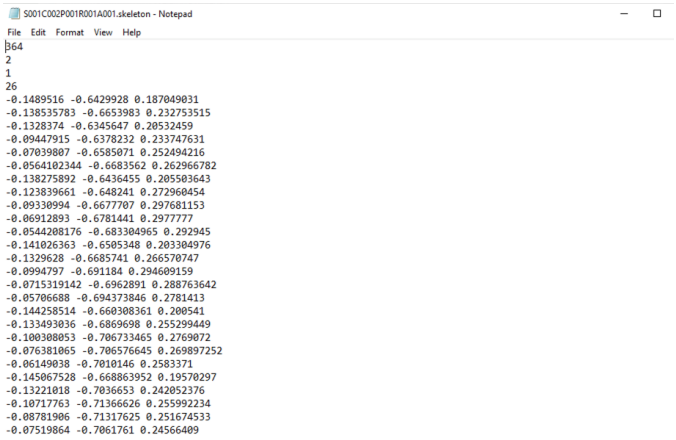

Fig. 5. Hand joint coordinates in the NTU format. Notice that the joint coordinates only list the 3D coordinates.

C. Graphs

Another important modification in the code is the inclusion of the aforementioned hand skeleton. In order to convey information regarding the skeletons, EfficientGCN stores a kind of adjacency matrix in which the positions of the joints are described relative to the center joint, which is the wrist joint in our case.

```
if self.dataset == 'hand':
    num_node = 26
    neighbor_1base = [(6, 5), (5, 4), (4, 3), (3, 1),
                      (11, 10), (10, 9), (9, 8), (8, 7), (7, 1),
                      (16, 15), (15, 14), (14, 13), (13, 12), (12, 1),
                      (21, 20), (20, 19), (19, 18), (18, 17), (17, 1),
                      (26, 25), (25, 24), (24, 23), (23, 22), (22, 1)]
    neighbor_link = [(i - 1, j - 1) for (i, j) in neighbor_1base]
    connect_joint = np.array([5,4,3,1,10,9,8,7,1,15,14,13,12,1,20,19,18,17,1,25,24,23,22,1]) - 1
    parts = [
        np.array([6, 5, 4]) - 1,          #thumb
        np.array([11, 10, 9, 8]) - 1,     #index
        np.array([16, 15, 14, 13]) - 1,   #middle
        np.array([21, 20, 19, 18]) - 1,   #ring
        np.array([26, 25, 24, 23]) - 1,   #pinky
        np.array([1,2,3,7,12,17,22]) - 1  #palm
    ]
```

Fig. 6. Data structure detailing the bones and the extremity of the bone closer to the center.

Notice the data structure outlined in Figure 6 that is found in the file $graph.py$. The $neighbor\_1base$ matrix contains the bones by specifying end joints, and $connect\_joints$ specifies which extremity of the bone is closer to the center, defining its orientation.

Moreover, similar to limbs, the hand skeleton is divided into six separate sections. There, we can see that joints are grouped under the following parts:

- Thumb
- Index
- Middle finger
- Ring finger
- Pinky
- Palm

III. Results

One thing that is very important to note regarding the benchmarks is the vast difference between accuracies of the 3 action and 4 actions datasets. The activities in question are the following:

- Piercing with the Lenki
- Poking with the Knopfsonde
- Tightening with the screwdriver
- Cutting with the scalpel

The main difference between our datasets is that the 3 action dataset only contains the first three action, whereas the 4 action dataset contains all actions. In the following tables, results from normalized and non normalized coordinates are compared alongside left hand only training compared to both hand training.

TABLE I
3 Action Dataset Accuracies

	Normalized	***Non-normalized***
Left hand training	$\mu = 0.83$ $\sigma = 0.0714$	$\mu = 0.82$ $\sigma = 0.1156$
Left and right hand training	$\mu = 0.78$ $\sigma = 0.0905$	$\mu = 0.89$ $\sigma = 0.0275$

TABLE II
4 Action Dataset Accuracies

	Normalized	***Non-normalized***
Left hand training	$\mu = 0.61$ $\sigma = 0.0917$	$\mu = 0.71$ $\sigma = 0.1067$
Left and right hand training	$\mu = 0.81$ $\sigma = 0.0249$	$\mu = 0.70$ $\sigma = 0.0265$

As we can see, the 4 action dataset performs significantly worse than the 3 action one. The main reason behind that is when data is well separated spatially, the model performs well. However, when we introduce another action that has similar temporal data to another existing action but differs temporally, the model struggles to differentiate between the two actions, leading to a significantly lower accuracy.

Apart from that, we can clearly observe that normalization leads to higher accuracies throughout. The only exception is the left hand normalized training with 4 action dataset, where the network fits on the location of the hand rather than its pose. However, this is a suboptimal fit. The location of the hand when performing the action should be irrelevant since we would like to focus on the pose instead. Since the scalpel and Knopfsonde are held in similar poses, this suboptimal fit increases the accuracy nonetheless. Lastly, while the use of both hands in training leads to a more accurate model, left hand training is not that far behind and offers a comparable accuracy.

IV. Discussion

A very evident observation from our results is that our model performs well with distinct spatial configurations, while

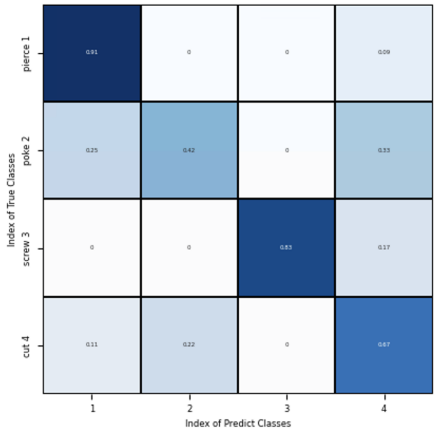

Fig. 7. The confusion matrix for the 4 action dataset. Notice the mix-ups between poking with Knopfsonde and cutting with scalpel. While these two actions differ temporally, they are very close spatially, leading to this state of confusion.

it suffers when the difference is limited to the temporal domain. A possible explanation for that is the limited frame window. In EfficientGCN, the datasets recorded with Kinect have around 60-100 frames while our dataset contains around 350-1000 frames due to the better capabilities of HoloLens. Since the maximum number of frames allowed by our GPU's memory is around 500 frames, the window cannot cover the whole action.

A direct approach to this issue would be simply increasing the number of frames the model is trained on, necessitating a larger memory for our GPU. However, since HoloLens records with such a high speed, we have around 60 frames covering one second in our recordings, which leads to bloated frame numbers. We could simply include one in five frames, effectively reducing the memory requirements of our model with very little information loss since a decrease from 60 FPS to 12 FPS should still retain the necessary temporal information for model fitting. As a result, the 500 frame window would be able to cover the whole duration of the action without the need of introducing more powerful hardware.

It should also be noted that our dataset is relatively small. In order to have more concrete information regarding the accuracy in the operating room, a larger dataset where the surgery tools are used my experts will yield a more accurate benchmark since our model is geared towards a very specific context that is the surgery room.

V. Summary

As a result of our benchmarks, we can conclude that GCNs are indeed reliable methods for skeleton based activity recognition. Even though the model was not extensively trained and it was trained on a small dataset, the resulting accuracies are already promising. This enables it to be potentially used for more niche tasks of which there are few datasets about. The model can be trained and deployed quickly for accurate results, and it is especially well suited for spatially distinct tasks. However, it should be noted that these benchmarks are also done in a very niche setting. In order to claim a wider use area, the model should be trained on a much larger dataset with more quotidian actions having different spatial and temporal configurations. A benchmark performed with this trained model would be more representaive of the model's capabilities in the bigger picture.

References

[1] Amir Shahroudy, Jun Liu, Tian-Tsong Ng, and Gang Wang. Ntu rgb+d: A large scale dataset for 3d human activity analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1010–1019, 2016.

[2] Jingbo Wang Dahua Lin Sijie Yan, Yuanjun Xiong. Mmskeleton. https://github.com/open-mmlab/mmskeleton, 2019.

[3] Yi-Fan Song, Zhang Zhang, Caifeng Shan, and Liang Wang. Efficientgcn: Constructing stronger and faster baselines for skeleton-based action recognition. *arXiv:2106.15125*, 2021.