Spatial-temporal Graph Convolutional Networks for Skeleton-based Action Recognition

Rajmund Nagy **Amrita Panesar** **Lívia Qian**

Abstract

In this project, we reimplement the Spatial-Temporal Graph Convolutional Network from 2018 which introduced graph convolutional networks to the area of human action recognition. We propose a novel partitioning strategy that performs on a comparable level with the original ones in the ST-GCN paper. With systematically designed experiments, we evaluate the importance of different components using the KTH Action Dataset. With no hyperparameter optimisation and limited training time, the network achieved a 80.47% test accuracy in our best run.

1 Introduction

With the rapid developments of the last decade in artificial intelligence, robotics and machine learning, we are already transitioning into an era where artificial agents will surround us in our home, at our workplace, on the road or even during a simple trip to the grocery store. These intelligent systems all face the challenge of recognising and understanding human intent, and much of modern AI research is focused on equipping them with the means of doing so. Our goal with this project was to reproduce the results an innovative paper from 2018 [9], aimed at solving human action recognition, a task within the domain of computer vision. The original authors introduce a graph convolutional network (GCN) architecture that operates on human joint location sequences (Fig. 1), which can either be captured by modern sensors such as Kinect, or extracted from videos by using keypoint detection frameworks. This publication was the first application of GCNs to the problem of action recognition on large-scale datasets, which spawned a new direction of research in this subfield.

To this day, the proposed ST-GCN model serves as an important baseline, and we hope that we can pique our fellow Master's students' interest in graph convolutional networks, a rapidly developing and truly exciting field with many open problems.

2 Related work

To provide proper context for the original paper, we will take a historical perspective and account for the developments that the ST-GCN builds on. We will discuss graph convolutional networks in detail in the Methods section.

In order to recognise human actions, multiple types of input can be used, for example *optical flows* [5, 7], *appearance* [1] and *depth maps* [10] are all competitive choices for input representation. The ST-GCN model operates on *dynamic human skeletons* represented by a time series of 2D or 3D human joint coordinates1. Most of the earlier methods simply treated these joint coordinates as feature vectors and performed temporal analysis on the input sequence, but they had very limited capability as they did not consider the spatial relationships between the joints. There were some recent methods that attempted to incorporate the spatial connections into the networks, but they relied on hand-crafted features or explicit rules driven by domain knowledge.

By the time ST-GCN was developed, deep learning approaches (mainly RNNs and temporal CNNs [11, 6]) were already popular for the other input modalities because they facilitate automatic feature extraction. While these methods were a natural fit for learning the temporal aspect of the input videos, they don't have an innate ability to learn spatial relationships between the joints.

The main innovation of the paper is the adaptation of graph convolutional networks to the action recognition task, which can effectively learn the spatial connections between different body parts without losing the ability to perform temporal analysis on the sequence.

3 Data

Since the original paper was trained on very large-scale datasets and we operated under severe time constraints, we decided to use the KTH action dataset [4] that contains only 6 types of human actions (walking, jogging, running, boxing, hand waving and hand clapping) performed by 25 subjects in 4 different scenarios: outdoors $s1$, outdoors with scale variation $s2$, outdoors with different clothes $s3$ and indoors $s4$, resulting in 600 videos in total that contain varying number of repetitions for an action. From the videos, we extracted the corresponding skeleton sequences with 25 joints using OpenPose [2].

3.1 Data augmentation

Since the KTH action dataset contains only 599 sequences, we apply data augmentation (similarly to the original paper) on the training set as a form of regularisation to help avoid overfitting. This involves modifying the training sequence at training time with probability 0.5 to have two out of the following transformations applied: horizontal flipping; rotation; translation; scaling. The amount of rotation, translation and scaling is chosen at random from a pre-chosen set of parameters.

4 Methods

4.1 Input representation

Each data point (video) is represented as a graph in which the joint locations are the vertices and the edges correspond to the connections between joints (let these edges be denoted by E_S). When it comes to inter-frame edges, corresponding joints in consecutive frames are connected (E_F). The number of vertices is fixed throughout the data set. In each frame of the video, the value associated with each vertex the position of the corresponding joint. Since we are working with 2D videos, we have to store 2 coordinates in each of these cases; the best way to deal with multiple dimensions is to regard them as separate channels. We are going to refer to the number of vertices as V and the number of frames as T for the rest of the paper (the number of time frames is not predefined).

4.2 Graph convolution

Convolutional graph neural networks redefine convolution for graph data. There are two distinct approaches for constructing the convolutional operator.

Spectral approaches (such as the famous ChebNet [3]) utilise the normalised graph Laplacian matrix as the mathematical representation of the undirected graph input. The Laplacian matrix is tremendously useful, because it possesses properties that enable us to apply the graph Fourier transform to it. As it is commonly known, the convolution operation is reduced to multiplication in the spectral-domain, yielding substantial speedups.

On the other hand, spatial methods follow traditional CNNs more closely. Since ST-GCN falls under this category, we will explicitly construct the notion of spatial graph convolution as follows.

First, consider a common application area of CNNs: recognising 2D images with c channels and pixel intensities ranging between $\{0, ..., Z-1\}$, using a $K \times K$ filter. Then, for each layer, the output of the 2D convolution operation at spatial location $\mathbf{x}$ for a single channel can be formulated as:

$$f_{out}(\mathbf{x}) = \sum_{h=1}^{K} \sum_{w=1}^{K} f_{in}(\mathbf{p}(\mathbf{x}, h, w)) \cdot \mathbf{w}(\ell(h, w)) \tag{1}$$

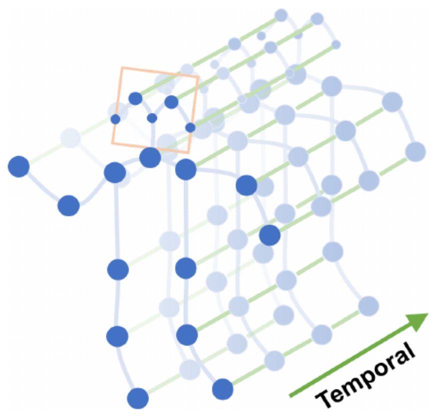

Figure 1: The spatiotemporal graph representation of the input skeleton sequence. The spatial connections are shows in light blue, and the temporal (self-) connections are shown in green. Figure taken from [9].

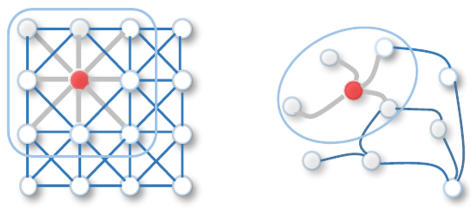

Figure 2: In images, the neighbourhood of a pixel is defined by its position on the 2D grid. In graphs, there is no inherent spatial order, and the neighbourhood is defined by the shared edges. Figure taken from [8].

where f_{in} is the input feature map, $\mathbf{p} : Z^2 \times Z^2 \to Z^2$ is the **sampling function** which enumerates the neighbours of location $\mathbf{x}$, $\ell : Z^2 \to \{0, ..., K^2 - 1\}$ is a **labelling function** that assigns up to K^2 different labels to the neighbours, which can be viewed as defining K^2 different types of distances from center pixel. Finally, $\mathbf{w} : \{0, ..., K^2 - 1\} \to \mathbb{R}^c$ is the **weight function** that provides a c-dimensional weight vectors based on the labels.

For 2D convolution on images, the sampling function $\mathbf{p}$ simply enumerates the pixels in the $K \times K$ neighbourhood centered around the pixel $\mathbf{x}$. Because the spatial order between pixels is inherently defined (conventionally, $x_{3,3}$ resides one pixel to the left and two pixels below $x_{4,5}$), this enumeration can be performed easily by iterating through the desired indices. The label of neighbour $x_{h,w}$ of the central pixel $\mathbf{x}$ with indices a, b is completely defined by the indices h and w of the neighbour: there are K^2 different distance tuples $(a - h, b - w)$, which can be mapped to labels. Therefore the labelling function can be omitted and the weight function reduces to a $K \times K$ real matrix, yielding the common formulation $f_{out}(\mathbf{x}) = f_{in} \odot \mathbf{W}$.

For graph inputs, the filter is going to be a K-dimensional vector instead of a matrix, because the V nodes in the input are not arranged in a 2D grid anymore. As such, the spatial connections cannot be defined solely by using indices, but we can enumerate the direct neighbours of the central node by accounting for all of its edges. The next step is to define the labelling function (which will assign up to K distinct labels because of the changed filter size), because the order of this enumeration is not well-defined (e.g. if we enumerate the edges clockwise starting from a certain point, then rotating the graph will change the order without changing the graph). Multiple strategies are discussed below. The filter weights will be defined by $\mathbf{w} : \{0, ..., K - 1\} \to \mathbb{R}^c$ — in practice, we store these values as a (c, K)-dimensional tensor.

4.3 Neighbourhood and partitioning

The concept of neighbourhood is introduced to emphasise the relationship between joints. For any node v_{ti} (where t is the temporal and i is the spatial index), the neighbourhood is

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

if both spatial and temporal features are considered. Γ is used for adjusting the temporal range; D is the size of the spatial neighborhood and is usually set to 1. We will use $D = 1$ for the entirety of the project.

A labelling function is used to categorise the elements in the neighbourhood. Given that we want to have K partitions, the labelling function is

$$l_{ST}(v_{qj}) = l_{ti}(v_{tj}) + (q - t + \lfloor \Gamma/2 \rfloor) \cdot K \tag{3}$$

where $l_{ti}(v_{tj}) : B(v_{tj}) \to \{0, ..., K-1\}$ corresponds to the spatial mapping of node v_{tj} given that v_{ti} is the root (centre) of the neighbourhood. There are multiple partitioning strategies that help create the label map. The authors of the paper recommend the following three strategies and introduce them for single-frame cases for the sake of simplicity (they can be extended to the spatial-temporal domain).

- **Uni-labelling** is the simplest strategy where every node has the same label (0) and $K = 1$.
- **Distance partitioning** is also simple; it partitions the neighbouring nodes according to their distance from the root node, which, if $D = 1$, means that the root nodes are always going to be 0 and everything else is 1. This way, the algorithm will be able to model local differential properties. $K = 2$ and $l_{ti}(v_{tj}) = d(v_{tj}, v_{ti})$.
- **Spatial configuration partitioning** is a bit more complicated. The neighbour set is divided into three categories: 1) the root itself; 2) centripetal group — the neighboring nodes that are closer to the centre of gravity than the root node; 3) the centrifugal group — any other node. The centre is defined as the average position of all joints within a frame. Based on this, we have

  $$l_{ti} = \begin{cases} 0 & \text{if } r_j = r_i \\ 1 & \text{if } r_j < r_i \\ 2 & \text{otherwise} \end{cases} \tag{4}$$

  where r_i is the average distance from the centre to joint i over all frames in the training set.

We also introduce a new partitioning scheme called symmetrical distance partitioning which takes the immediate neighbours in a similar fashion as distance partitioning but the symmetrical joints on the other side of the body as well.

4.4 Graph convolutional network

The network has a similar structure to a regular CNN; in our case, it consists of 9 + 1 convolutional layers, a global pooling layer and a Softmax classifier. The number of input and output channels for the convolutional layers look like this: (64, 64), (64, 64), (64, 64), (64, 128), (128, 128), (128, 128), (128, 256), (256, 256), (256, 256). The stride is 1 for all these layers except for the 4th and the 7th temporal convolution layers; for these, the stride is set to 2 as pooling layer. There is an additional graph convolutional layer at the beginning that increases the number of channels from 2 to 64. There are also other types of layers (see later sections).

4.4.1 Spatial convolution

Each convolutional layer consists of a spatial and a temporal convolutional layer. Spatial convolution is applied to all vertices, meaning that it runs over both the spatial and the temporal dimension. It should not, however, alter the temporal dimension, meaning that the filter size has to be in the form $(size_{spatial}, 1)$. The authors of the paper decided to perform a 1×1 convolution; this type is mainly used for changing the number of channels without having to change the amount of information carried by the data. Supposing that the number of input channels is C_{in} and the number of output channels is C_{out}, the depth of the 1×1 filters becomes C_{in} and there are C_{out} such filters (or $C_{out} * K$, if we take into account the fact that the number of partitions is K). The following formula was devised to summarise the operations needed for this kind of convolution:

$$f_{out} = \sum_{j=0}^{K-1} \Lambda_j^{-\frac{1}{2}} A_j \Lambda_j^{-\frac{1}{2}} f_{in} W_j \tag{5}$$

Here, f_{in} is the input of size (C_{in}, T, V) — or (N, C_{in}, T, V) when using batches — and W_j of size (C_{in}, C_{out}) is the weight matrix belonging to partition j. A_j is an adjacency matrix that represents single-frame intra-body edges and marks the nodes with label j each root is connected to; for example, $A_0 = I$ if self-connections are labelled with 0 (and nothing else). Finally, Λ_j is a diagonal matrix where $\Lambda_j^{ii} = \sum_k A_j^{ik} + \alpha$. We set α to 0.001 to avoid empty rows in A_j.

4.4.2 Temporal convolution

The temporal layer receives the output from the spatial layer, i.e., data of size (N, C_{out}, T, V). It is a standard 2D convolution that runs over the temporal dimension with kernel size $1 \times \Gamma$ and padding $\frac{\Gamma-1}{2}$ (so that the number of frames remains the same). The output of this layer should be (N, C_{out}, T, V).

4.4.3 Learnable edge importance weighting

There are joints that belong to multiple body parts, but their importance may vary depending on body part. To take this into consideration, a learnable mask M can be added to every layer of spatial-temporal graph convolution. The mask will scale the contributions of a node's feature to its neighboring nodes based on the learned importance weight of each spatial graph edge in E_S . This might improve the recognition performance and is straightforward to implement based on Eq. (5); A_j needs to be substituted with $A_j \otimes M$ where $\otimes$ denotes the element-wise product.

4.5 Residual block

Deep networks can often struggle to learn the underlying structure of data due to the vanishing gradient problem, causing information to be lost deeper in the network. We address this issue by adding an additional residual block to the original ST-GCN architecture. There is some evidence that applying batch normalisation with pre-activation achieves the best performance when residual connections are used, hence we opted for this approach. This type of residual structure, called full pre-activation, is illustrated in Figure 3.

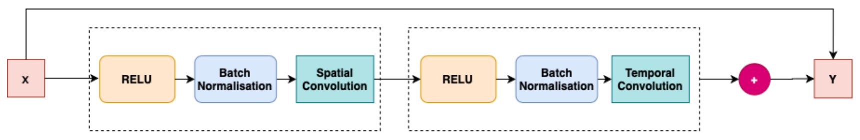

Figure 3: Residual block is made up of two blocks of activation-batch normalisation before each set of weights. Takes input X passes through two sets of pre-activated weights and the result is added to original X to produce output Y.

5 Experiments

In this section, we present the experiments we carried out to test different modifications to the original ST-GCN architecture.

5.1 Implementation details

We trained our STGCN using a Pytorch backend. We used the Adam optimiser to update the weights, batch size of 16. We used categorical cross-entropy as the loss function. The mask is initialised as a trainable parameter of 1's of the same dimensionality as the adjacency matrix when edge-importance weighting was used.

To evaluate our approach, we apply cross-subject training. This involves first training on 10 subjects and testing on a different set of 5 subjects (with a further 5 used as the validation set). This method is used to examine whether our model is learning general motion patterns in the joint sequences for actions that are independent of the subject.

5.2 Comparison of partitioning strategies

We compared the three partitioning strategies in the default setting with 1000 epochs and found that spatial configuration partitioning has the best performance on the test set. What is surprising is that distance partitioning performs much worse than uni-labelling despite having more expressive power. We decided to ignore this minor inconvenience because the other two produced the expected behaviour. Without regularisation, we found that the more complex a strategy is, the worse it performs.

We found that the new symmetrical partitioning strategy performs slightly worse than spatial configuration but is still fairly good, and beats distance labelling.

Table 1: Accuracies for 9-layer network with dropout rate of 0.5, no data augmentation and no edge importance weighting.

	Validation	Test
Uni-labelling	0.7109	0.6484
Distance	0.5156	0.4837
Spatial conf.	0.7188	0.7031
Symmetrical	0.625	0.6562

5.3 Learning rate

A high learning rate allows the model to learn faster, but can result in sub-optimal steps being taken in the loss landscape and thus a lower accuracy. In contrast, a smaller learning rate can enable the model to learn a more optimal set of weights, but at the expense of slower learning.

We investigated the effect of learning rate on validation accuracy and loss, for dropout of 0.5 and 0.25. As can be seen in the next section in Figure 4, a learning rate of $10e^{-4}$ was found to be most optimal (with dropout 0.5), hence we trained the final networks using this.

5.4 Regularisation

In order to avoid overfitting, we applied two commonly used regularisation techniques: dropout and data augmentation (as detailed in Section 3.1). We found data augmentation to produce smoother learning curves when the network was training longer.

The authors of [9] apply dropout at a rate of 0.5 in each STGCN-unit. We investigated the effect of dropout on performance to examine whether the small number of training sequences in the KTH Dataset as compared to the Kinetics database would require a higher dropout to perform well. Empirically, we found dropout rates of higher than 0.5 led to a higher validation loss and lower validation accuracy. Dropout 0.5 with a learning rate $\eta = 10^{-4}$ performed similarly to dropout 0.25 with the same learning rate, but produces a smoother validation curve during training.

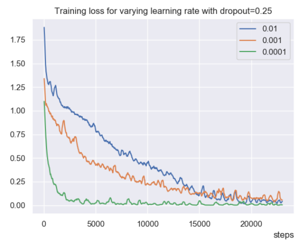

Figure 4: Dropout=0.25. Training loss decreases faster and is smoother with lower learning rates.

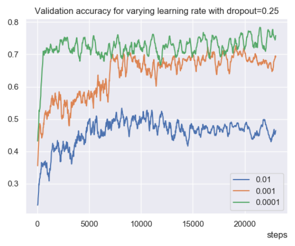

Figure 5: Dropout=0.25. Validation accuracy is highest for lower learning rate.

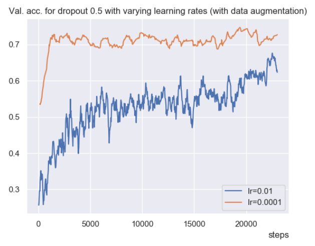

Figure 6: Dropout=0.5. Validation accuracy with is highest for learning rate 10^{-4}.

5.5 Batch normalisation

Batch normalisation is a method used between layers to improve the performance of neural networks. Often batch normalization layers are combined with activation layers; however, it is still argued whether having the activation before or after them is more effective. In [9], the authors mention using batch normalisation but are ambiguous over how many batch normalisation layers are used. We originally interpreted the paper as having only one batch normalisation layer (with an additional activation layer) in the entire network, right before passing the data to the convolutional layers. We experimented with it and found that it is bad to avoid the use of batch normalisation between convolutional layers (training loss turned out to be indefinite after every step).

Hence, we figured that they intended the batch normalisation layer to be used at the beginning of each graph convolutional layer. Another one of our experiments was targeted at finding out whether having batch normalisation at the end of each layer is necessary as this is the usual setting in most neural networks. We found that the network's performance is worse when there's batch normalisation only in the beginning.

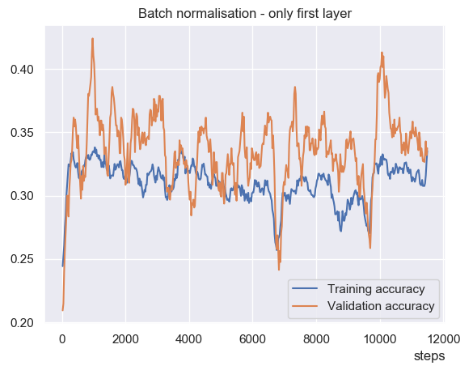

Figure 7: Accuracy for 500 epochs with batch size 500.

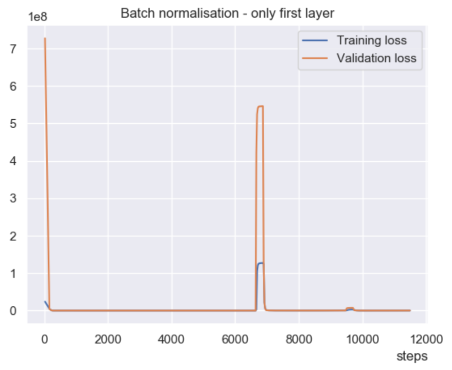

Figure 8: Loss for 500 epochs with batch size 500.

Our third experiment included changing the order of the ReLU activation layer and the batch normalisation layer. After running it for 200 epochs, we saw that its best validation accuracy was around 0.57. Due to time limits we decided to continue without investigating this option.

5.6 Residual block

We investigated whether adding a residual block could help performance, testing on the two best strategies thus far: unilabelling and spatial configuration. Empirically, we found that using a residual block improved performance with both strategies. We were able to build deeper networks and achieve a similar/better performance when using a residual block than without, however the performance dropped compared to the standard architecture. All experiments in this section were conducted with learning rate $\eta = 0.01$.

Presented in Figure 10 are the results of testing the residual block on the standard 9-layer architecture, as well as a slightly deeper 12-layer network with the following input-output channels: (64, 64), (64,64), (64,64), (64, 64), (64, 128), (128, 128), (128, 128), (128, 128), (128, 256), (256, 256), (256, 256), (256, 256). Our results show that a residual block improves accuracy, at least when 0 dropout is applied. The 9-layer network outperforms the deeper 12-layer network.

Experimenting with higher dropout 0.5 showed that validation accuracy stayed stagnant around the 0.2 mark. This is perhaps due to the residual block being unable to learn anything significant with the high proportion of dropped weights in the spatial and temporal convolution. For this reason, we carried out a final, longer run (see Section 5.7) with 0 dropout with the residual block enabled.

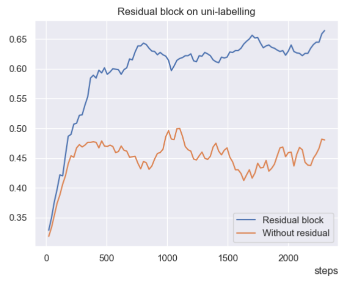

Figure 9: Uni-labelling on 9-layer network.

	Uni-labelling		Spatial Configuration	
	Original	W. Residual	Original	W. Residual
9-layer	0.5703	0.6796	0.5156	**0.7188**
12-layer	0.4688	0.5547	0.6719	0.6953

Figure 10: Validation accuracies for 9-layer and 12-layer network, experiments run for 100 epochs, 0 dropout. Highest performance is achieved for spatial configuration when a residual block was applied, on the 9-layer network.

5.7 Final model

In this section, we present the best results with the uni-labelling and spatial configuration partitioning strategies. Each run is with $\eta = 10^{-4}$, and with data augmentation switched on. We find the best performance is achieved when the spatial configuration partitioning strategy is used with the standard 9-layer network, with data augmentation and dropout = 0.5.

Given that data augmentation seemed to improve performance, we carried out two final longer training runs, one with the best residual block architecture we found thus far (0 dropout) and one without the residual block and dropout. As can be seen in Figure 5.7, the residual-block performance produced the best results, with an accuracy of 80.47%.

Strategy	Dropout	Residual Block	Edge-importance weighting	Test Accuracy
Uni-labelling	0.5	✗	✗	0.7109
Spatial configuration	0.5	✗	✗	0.7734
Spatial configuration	0.0	✓	✗	***0.8047***
Spatial configuration	0.0	✓	✓	0.703125

6 Conclusion

In this project, we present the application of spatial-temporal graph convolutional networks to skeletal-based activity recognition. We achieved good performance with the spatial configuration partitioning strategy, and tested several different modifications of the base network presented in [9].

Further work could perform further hyperparameter-optimisation and augmentation strategies, as we found it was easy for the network to overfit on the small KTH action dataset. It would also be interesting to explore applying a different temporal kernel size and input channel-output channel architectures for the convolution.

We chose to implement our network in Pytorch, with Pytorch Lightning used to provide a high-level interface for conducting research experiments. We hope this will aid other researchers in performing further optimisations, and have made our code available for public use here [1].

[1]https://github.com/nagyrajmund/st-gcn

References

[1] M. Al-Faris, J. Chiverton, L. Yang, and D. Ndzi. Appearance and motion information based human activity recognition. In *IET 3rd International Conference on Intelligent Signal Processing (ISP 2017)*, pages 1–6, 2017.

[2] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh. OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[3] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 3844–3852, Red Hook, NY, USA, 2016. Curran Associates Inc.

[4] Christian Schüldt, Ivan Laptev, and Barbara Caputo. Recognizing human actions: A local SVM approach. In *Proceedings - International Conference on Pattern Recognition*, volume 3, pages 32 – 36 Vol.3, 09 2004.

[5] Karen Simonyan and Andrew Zisserman. Two-Stream Convolutional Networks for Action Recognition in Videos. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, page 568–576, Cambridge, MA, USA, 2014. MIT Press.

[6] A. Ullah, J. Ahmad, K. Muhammad, M. Sajjad, and S. W. Baik. Action Recognition in Video Sequences using Deep Bi-Directional LSTM With CNN Features. *IEEE Access*, 6:1155–1166, 2018.

[7] M. Wrzalik and D. Krechel. Human Action Recognition Using Optical Flow and Convolutional Neural Networks. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 801–805, 2017.

[8] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, page 1–21, 2020.

[9] Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial Temporal Graph Convolutional Networks for Skeleton-Based Action Recognition, 2018.

[10] Xiaodong Yang and YingLi Tian. Super Normal Vector for Activity Recognition Using Depth Sequences. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, page 804–811, USA, 2014. IEEE Computer Society.

[11] Wentao Zhu, Cuiling Lan, Junliang Xing, Wenjun Zeng, Yanghao Li, Li Shen, and Xiaohui Xie. Co-Occurrence Feature Learning for Skeleton Based Action Recognition Using Regularized Deep LSTM Networks. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 3697–3703. AAAI Press, 2016.