An Optimized DGNN Architecture for Skeleton-Based Action Recognition

CS 7643 - Deep Learning (Fall 2023)

Georgia Institute of Technology

Sean Donnelly sdonnelly8@gatech.edu

Clay Dustin cdustin3@gatech.edu

Niko Lahanis nlahanis3@gatech.edu

Andrew Mueller amueller39@gatech.edu

Abstract

This paper addresses Skeleton-Based Action Recognition (SBAR) using Directed Graph Neural Networks (DGNN) in the context of computer vision, introducing two novel architectural enhancements to DGNNs— Attention Layers and Dropout Layers. These enhancements are aimed at improving SBAR model performance. Leveraging the widely-used NTU RGB+D 60 dataset, this research compares the proposed model against a baseline DGNN. Results reveal substantial accuracy gains, particularly with the inclusion of attention layers, without compromising generalization through the incorporation of dropout layers. The experimentation, conducted with careful consideration of computational constraints and dataset sizes, highlights the efficacy of the enhancements. Challenges in cloud resources and dataset magnitude are candidly discussed, providing insights for researchers facing similar constraints. The paper concludes with recommendations for scaling findings to larger datasets, exploring other frameworks, and proposing additional architectural adjustments for future SBAR research using DGNNs. This work contributes to the ongoing discourse on optimal neural network architectures for human motion recognition and sets a foundation for further advancements in the field.

1. Introduction/Background/Motivation

Over the past decade, the realm of Computer Vision (CV) has undergone a remarkable surge, fueled by innovative applications such as self-driving vehicles, facial recognition software, and dynamic object detection. These applications have evolved in tandem with the broader advancements in Artificial Intelligence (AI) infrastructure, which underpin their functionality. One of the most powerful AI architectures currently being used to perform these vision-based tasks are neural networks [9].

The use of a neural network architecture for computer vision allows the model to learn implicit features within the input data (in this case videos and images) without the explicit need to ”cherry-pick” important features. Neural networks more directly reduce the need for feature engineering to achieve optimal performance accuracy during model training.

While the use of neural networks in computer vision is a broad and fast-growing domain, our research chooses to focus on one specific subset of this area: Skeleton-Based Action Recognition (SBAR). While previous research has focused on the use of CNN in motion classification [3], some research has favored the use of Graph Networks [1]. The human body can be represented as a graph in which bones act as edges and joints function as vertices connecting these edges. Strong performance in SBAR has been noticed when using a specific type of Neural Network architecture called a Directed Graph Neural Network (DGNN) [7]. Previous research has shown that, when converting human movement from video/image data into a Directed Acyclic Graph (DAG) representing a human skeleton, neural networks are able to train on features of this graph and predict the motions being performed with high accuracy [5]. The combination of creating this Skeleton-Based DAG and subsequently running this representation through the layers of a DGNN establishes the foundation for state-of-the-art performance in human motion recognition tasks [4].

While existing DGNN models in literature achieve sufficient performance on large-scale human motion data-sets such as NTU RGB+D [6] and Skeleton-Kinetics (see 2.1 for detailed description of datasets), our research examines the following architectural additions to the DGNN framework.

1. **Attention Layers** - The inclusion of attention layers during our Graph updating process allows our DGNN architecture to selectively focus on essential nodes and learned features, while attenuating the significance of others.

2. **Dropout Layers** - While the selective focus of an attention layer can be beneficial to increase the predictive power of our model, this does put our model at an increased risk of bias, and in turn over-fitting. To counter this risk, we add dropout layers within the network to promote better generalization during model training.

Our research team implemented the outlined features to devise a novel DGNN architecture which we subsequently trained on the NTU RGB+D dataset. To assess its performance, we benchmarked the results against existing state-of-the-art SBAR DGNN models. [7] [4]

2. Approach

2.1. Input Data

Our model architecture is trained using ROSE Lab's 3D Skeleton Action Recognition dataset, one of the most widely used datasets for SBAR related tasks. Specifically, we used a subset of the data encompassing the first 60 actions, ”NTU RGB+D 60”.

Figure 1. Example Video Data Frames

This dataset contains 4 different modalities of data: RGB videos, depth map sequences, 3D skeleton data and infrared videos. Here, we only use the 3D skeleton data. NTU RGB+D 60 totals 113,945 samples over 60 classes captured from 106 distinct subjects and 32 different camera setups.

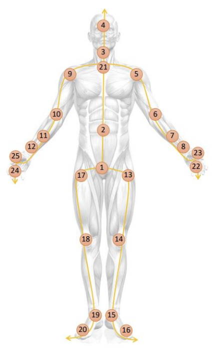


Figure 2. DAG Representation of Human Body

There are two separate benchmarks recommended by the original creators of the dataset [6]: Cross-Subject (The persons in the training and validation sets are different) and Cross-View (The horizontal angles of the cameras used in the training and validation sets are different).

Due to computational constraints, we opted to only train our model on Cross-View data, where 54,468 samples collected from half of the camera setups are used for training and the other 59,477 samples are used for testing.

This dataset is composed of two separate tracking components. Spatial data tracks the position of the skeletal graph, while motion data tracks the velocity of the skeleton over time.

2.2. Initial Model Architecture

Our research built upon the findings in, ”Skeleton-Based Action Recognition with Directed Graph Neural Networks”, which established a foundation for performance comparisons. [7].

The baseline neural network proposed by this team is primarily composed of Graph Temporal Convolution (GTC) Layers. Each GTC layer can be broken down into the following two components:

1. **DGN Block** - This block processes the spatial information of the skeleton data: feature aggregations are derived from linear transformations using the base target and source graphs for both the joint and bone information.
2. **TCN Block** - This block connects each time frame by applying a 1D convolution across the temporal dimension of the skeletal data. The aggregated joint and bone features are then individually passed through linear, batch normalization, and ReLU layers.

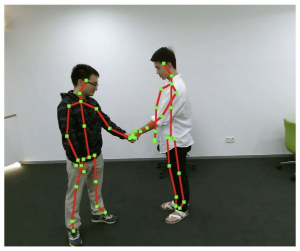


Figure 3. DAG Representation Overlaid on Video Data

It's important to note that, while these blocks combine to create one GTC layer, they inherently decouple the spatial and temporal dimensions of the skeletal data. This means the information flow of these dimensions are processed independently.

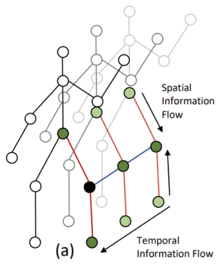
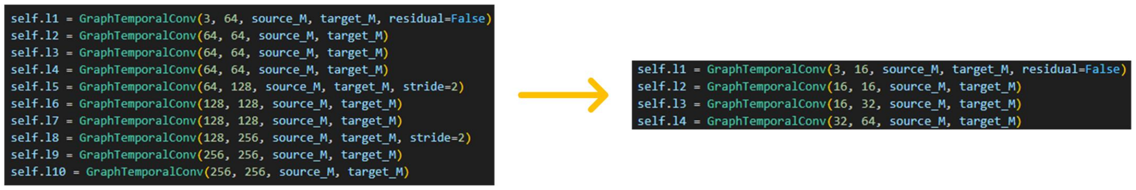


Figure 4. Information Flow - Baseline

The original architecture of the directed graph neural network (DGNN) has 10 units, each containing one DGN block and one TCN block. The output channels of the units are 64,64,64,128,128,128,256,256 and 256. The end of the network includes a global average pooling layer, followed by a softmax layer for class prediction. Due to computational constraints, we opted to reduce the sequence of layers in our baseline architecture.


```
self.l1 = GraphTemporalConv(3, 64, source_M, target_M, residual=False)
self.l2 = GraphTemporalConv(64, 64, source_M, target_M)
self.l3 = GraphTemporalConv(64, 64, source_M, target_M)
self.l4 = GraphTemporalConv(64, 64, source_M, target_M)
self.l5 = GraphTemporalConv(64, 128, source_M, target_M, stride=2)
self.l6 = GraphTemporalConv(128, 128, source_M, target_M)
self.l7 = GraphTemporalConv(128, 128, source_M, target_M)
self.l8 = GraphTemporalConv(128, 256, source_M, target_M, stride=2)
self.l9 = GraphTemporalConv(256, 256, source_M, target_M)
self.l10 = GraphTemporalConv(256, 256, source_M, target_M)
```

```
self.l1 = GraphTemporalConv(3, 16, source_M, target_M, residual=False)
self.l2 = GraphTemporalConv(16, 16, source_M, target_M)
self.l3 = GraphTemporalConv(16, 32, source_M, target_M)
self.l4 = GraphTemporalConv(32, 64, source_M, target_M)
```
Figure 5. DGNN Layers - Reduced Baseline

2.3. Attention Layer

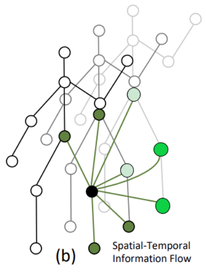


Figure 6. Information Flow w/ Attention Layer

Central to the DGNN network is the DGN block which updates vertex and edge features by aggregating adjacent components. In the case of a vertex updates, the DGN takes as input its current state and all incoming and outgoing edges (i.e. $v_i' = h([v_i, e_{in}, e_{out}])$). The original architecture used a fully-connected layer as the update function h, but we chose to experiment with different attention functions. [2]

1. Attention 1: Applies attention to a linear combination of the spatial features.

2. Attention 2: K, Q, and V matrices are computed for the entire bone and joint feature stack, solely targeting the spatial stream (bone and joint information over the graph). All the following methods were based off this implementation.

3. Attention 3: Applies attention to the sub-components of the bone and joint feature stacks.

4. Attention 4: Attention calculated for bone and joint weights using only the temporal stream.

5. Attention 5: Attention calculated over spatial and temporal streams.

6. Attention 6: Attention applied after the Bi-Temporal Convolution Layer.

To understand how attention is applied to our DGNN architecture, we focus on Attention 2, which bears the closest resemblance to attention methods learned in the course [8]. We specifically focus on how vertex weights are updated.

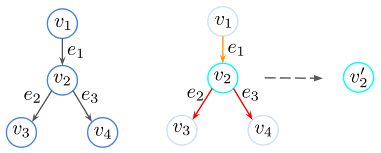


Figure 7. Vertex Weight Updates

In the diagram above we have a set of vertices and edges. Let the adjacency matrix A show the relationship between edge and vertex. In practice, there are both source and target adjacency matrices used to illustrate direction. The matrix multiplication of the edge feature matrix with A gives us a linear combination of ”adjacent” edge features for each vertex. Using the diagram above, the result of AE calculates a linear combination of e_1, e_2, and e_3 which results in an updated feature representation called v_2'. Some motions bring non-connected nodes together such as clapping, or tying a shoe. These motions require A to be able to learn possible cross body edge-vertex connections.

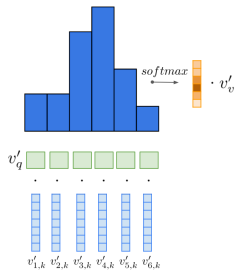


Figure 8. Self-Attention

We then apply self-attention to these updated vectors. Since v_2' is created using patterns from the adjacency matrix, the only vertices that attention is applied to are those vertices that are ”adjacent” to v_2. In figure 8, $v_1', v_2', ...v_6'$ are all judged to be ”adjacent” to the vector associated with v_q.

2.4. Dropout Layer

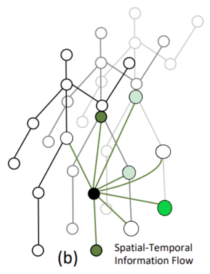


Figure 9. Information Flow w/ Dropout Layer

The attentive nature of the above features gives our architecture the ability to selectively focus on certain nodes and edges even when those nodes are far away from each other. Subsequent work offered an alternative way to understand compound motions that were carried out on different scales and distances [4]. However, attention is a much more efficient way of bridging this gap. Our research team hypothesized that, while this can lead to stronger predictive capabilities, the increased reliance on a smaller subset of features also puts our novel architecture at risk of over-fitting.

To increase the generalization capabilities of our neural network and reduce the risk of over-fitting, we included dropout layers throughout the model. This allows the network to randomly remove features so that it can more effectively distribute it's weighting across all learned features.

3. Experiments and Results

3.1. Training

Our main objective was to evaluate the addition of attention to the model. To start, we trained and tested each model for 20 epochs on 25% of the dataset. We then trained the highest performing attention model on a larger chunk of data and increased the epochs. The training data was increased to 50%, with epochs set at 60, along with the addition of two GCN layers. This was done to confirm that the attention's improvement over our baseline remained after simulating conditions resembling our target model.

3.2. Hyper-parameter Tuning

Given the high computational cost, it was difficult to experiment with hyper-parameters. We created a parameter to reduce the size of training data but this didn't seem to impact the performance of the model. In previous works the adjacency matrix became learnable after 10 epochs [9], this was to make sure the model could perform initial weight training efficiently. Since we trained on a fraction of the dataset we experimented with making the adjacency matrix learnable from the start of training. Additionally, we set a relatively high learning rate at 0.1 and chose to use Nesterov momentum for optimization given previous research covering its effectiveness in high learning rate training.

3.3. Testing

We used Cross-Entropy loss and Top K for accuracy calculations. In Top K, the model's top k guesses are used to evaluate against the true label. We used top 1, 3, and 5, but found that these metrics showed proportional improvements in accuracy, so we settled on evaluating with Top 3. The graph above shows that Attention 2 and Attention 3 outperform the other attention methods, as well as

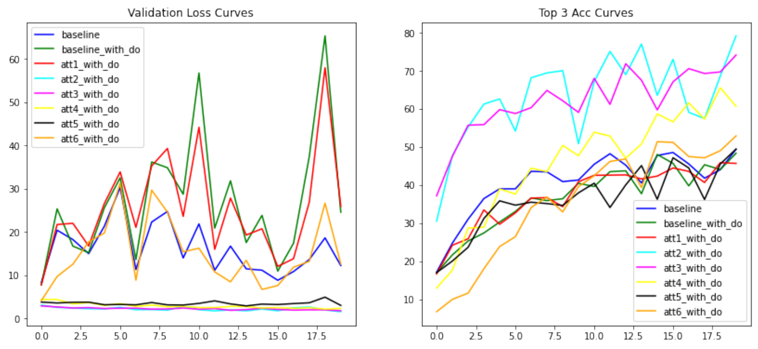


Figure 10. Testing Results

the baseline. We decided to use Attention 2 in our longer run of the model to compare against the baseline. The results of these runs are shown below.

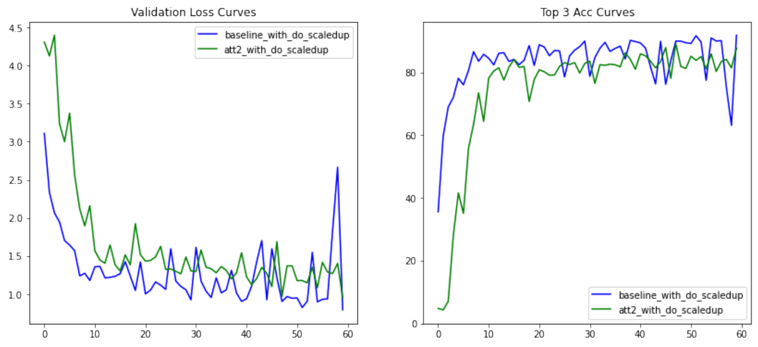


Figure 11. Testing Results

In the graph above, we see the baseline model outperforms attention 2 throughout training. Though attention out-performed the baseline when only three convolution layers were used, more layers and larger channel dimensions seem to have a positive effect on the baseline model results. Attention seems to recover features that are lost with less convolution layers. In this case, convolution layers with growing receptive fields do a better job modeling the multi-scale nature of motions than our attention methods.

3.4. Computational Demands

The implementation of our model architecture had specific computational requirements in order to properly train and observe results in a time and cost-effective manner.

The combination of a large dataset, and a computationally expensive architecture required our research team to pare down both the size of the training data, as well as reduce the numbers of layers in our model to allow for optimal run-times.

PyTorch is the Python-based Deep Learning library used for our implementation. This library allows for computational resources to be allocated to the GPU which significantly helped with run-time performance. The GPU used for training was an NVIDIA GeForce RTX 3080.

A preliminary cloud architecture was also replicated in Google Cloud's Vertex AI platform. In the cloud we were able to make use of larger CPUs and more powerful GPUs but they were costly. Due to budget constraints, we were unable to complete full runs in the cloud. However, the cloud did enable group members without GPU's to experiment with model development.

4. Experience

4.1. Challenges

1. **Cloud/Machine Limitations** - Our team quickly exhausted our allotted resource budget in Google Cloud, forcing us to resort to training on local machines which bottle-necked runtimes. Only half of the team had GPU capabilities requiring allocation to target group members to test implemented changes.

2. **Large Dataset Size** - The profoundly large size of the input data made training and testing this model a time-consuming task. This meant that we had to be very wary of any model changes made, and more importantly whether the change was worth prioritizing for a full model run.

3. **Model Ambiguity** - A common problem among existing deep learning frameworks is the inability to observe and visualize performance during run time which is something our team also experienced.

4.2. Approach Changes

Our initial attempts at creating a baseline deep learning framework for SBAR centered around the paper ”Disentangling and Unifying Graph Convolutions for Skeleton-Based Action Recognition” and Ken Liu's corresponding implementation in Github [4]. The architecture in this paper also proposes using DGNNs as it's base for human motion prediction, with the inclusion of a powerful feature extractor layer called ”MS-3GD” that utilizes a 3-Stream framework for human action recognition. Due to computational constraints aligned with the additional model complexity, we pivoted to the DGNN framework as a model baseline.

4.3. Successes

1. **Attention Layer Accuracy Increase** - Attention 2 and Attention 3 were able to achieve drastic performance increases compared to the baseline DGNN model. This indicates that providing a mechanism for the network to focus on specific subsets of the graph for classification is worth pursuing in further research.

2. **Increased Regularization** - The inclusion of a dropout layer in this deep learning framework showed that our data was able to achieve sufficient accuracy on test data, even with the complexity increases that came from the inclusion of an attention layer within our model.

5. Conclusion/Future Work

Our paper further motivates the inclusion of attention and dropout layers within DGNNs for Skeleton-Based Action Recognition. The inclusion of these layers into a baseline DGNN model showed stark accuracy improvements from the attention layer without sacrificing generalization capabilities with the additional inclusion of a dropout layer.

As a continuation of this research topic, our team advocates delving deeper into exploring the effectiveness of attention mechanisms within the DGNN framework, specifically when scaled up to encompass the entire NTU RGB + D action recognition dataset and the Kinetics 400 data-set. Our research was constrained by computational limitations, warranting a more comprehensive analysis to uncover the true efficacy of our results. Additionally, scaling these findings to larger datasets and other frameworks outside of DGNNs would allow for further result analysis. Our team recommends testing our model architecture at the scale and size of the architecture implemented in the paper to truly gauge effectiveness.

While the existing input data for this model is currently formatted in a two-stream skeleton based framework (one for position, and one for velocity), a future area of work may include a third stream that is already commonly used in analysis for engineering dynamics applications (acceleration).

References

[1] L.; Zhang X.; Lin L.; Tang G. Ahmad, T.; Jin. Graph convolutional neural network for human action recognition: A comprehensive survey. *IEEE Trans. Artif. Intell.*, 12, 2021. 1

[2] Dongkwan Kim and Alice Oh. How to find your friendly neighborhood: Graph attention design with self-supervision. In *International Conference on Learning Representations*, 2021. 3

[3] Jian Cheng Lei Shi, Yifan Zhang and Hanqing Lu. Non-local graph convolutional networks for skeleton-based action recognition. volume 12, 5 2018. 1

[4] Ziyu Liu, Hongwen Zhang, Zhenghao Chen, Zhiyong Wang, and Wanli Ouyang. Disentangling and unifying graph convolutions for skeleton-based action recognition. *CoRR*, abs/2003.14111, 2020. 1, 4, 6

[5] Senjian An Ferdous Ahmed Sohel Qiuhong Ke, Mohammed Bennamoun and Farid Boussad. A new representation of skeleton sequences for 3d action recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, page 4570–4579, 2017. 1

[6] Amir Shahroudy, Jun Liu, Tian-Tsong Ng, and Gang Wang. NTU RGB+D: A large scale dataset for 3d human activity analysis. *CoRR*, abs/1604.02808, 2016. 1, 2

[7] Lei Shi, Yifan Zhang, Jian Cheng, and Hanqing Lu. Skeleton-based action recognition with directed graph neural networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7904–7913, 2019. 1, 2

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, 2017. 4

[9] Zijie Ye, Haozhe Wu, and Jia Jia. Human motion modeling with deep learning: A survey. *AI Open*, 3:35–39, 2022. 1, 5

6. Appendix

6.1. Work Division

Members	Responsibilities
Sean Donnelly	Version Control, Model Implementation, PyTorch Geometric Investigation, and Model Training
Clay Dustin	Google Cloud Implementation, Model Training, and Model Evaluation/Results
Niko Lahanis	Google Cloud Implementation, Business Case Analysis, and Model Testing
Andrew Mueller	Attention Layer Design, Model Implementation, and Model Training

Table 1. Division of Work

In additional to the above individual job responsibilities, each team member also uniformly contributed to the following aspects of this project, including: multiple meetings per week, live coding, live editing, analysis, and paper reviews. Every team member shared an equal contribution to the finality of this deliverable.

6.2. Message Passing Through PyTorch Geometric

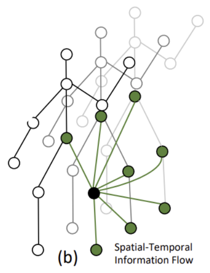


Figure 12. Information Flow w/ Message Passing

It's worth noting that the PyTorch Geometric library was influential in our model development. PyTorch Geometric is a library tailored to developing efficient implementations of graph networks. The core enhancements were centered around PyTorch Geometric's Message Passing interface, which abstracts the message passing process by defining "propagate" and "message" methods to allow for information flow between graph vertices. This allowed the model to aggregate and update node features while considering interrelations among joints (vertices) and bones (edges) in the skeletal data.