Università degli Studi di Milano - Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di Laurea in Informatica

Heterogeneous Graph Neural Networks for Action Prediction on Egocentric Action Scene Graphs

Supervisor: Doct. Sorrenti Domenico

Co-supervisor: Prof. Ognibene Dimitri

Authored by:
Olmo Ceriotti
Student No. 886140

Academic Year 2023-2024

A mia madre, sempre pronta a supportarmi.
A mio padre, che mi ascolta sempre anche quando vado in loop.
A Lavinia, che mi ricorda di crederci anche quando non lo faccio più.
Ai miei amici, per le risate e le infinite discussioni.
Al Prof. Dimitri Ognibene per avermi aiutato a dar forma a questo lavoro.
A tutti coloro che ho incrociato negli ultimi tre anni, ad ogni bar, biblioteca o sala dove ho potuto studiare, all'open source e alla libera informazione.
Grazie.

"- Do you know what I'll do before I do it?
- Yes.
- What if I do something different?
- Then I don't know that."

Futurama

Abstract

This work explores the capabilities of heterogeneous graph neural networks to perform action prediction on the novel dataset Egocentric Action Scene Graphs. EASG presents a unique challenge for action prediction given its dynamic nature and structure. The main objective of this work is to navigate the possible architecture that can capture the deep connections between the nodes, finding the most suitable one and researching their capabilities. This approach highlights the possibilities given by heterogeneous GNNs on the field and creates a baseline for future work in relation to this approach.

Contents

List of Figures

Chapter 1

Introduction

Egocentric vision has earned a wide interest in the last decades. The possible applications in augmented reality, autonomous systems, and even healthcare help characterize it as one of the most prominent fields in the future technology landscape. Usually, egocentric vision applications focus on processing raw images, videos, and audio from a headset with a camera or a dedicated device [1], traditionally annotated with labels in the form of strings. Researchers from the University of Catania and Intel Labs took this notion a step further, producing a framework for creating deep and comprehensive annotations that could lead to a better understanding and representation of the ongoing scenes captured [2]. Egocentric Action Scene Graphs represent the current action through a graph encoding of its main actors, highlighting the underlying connection between them and keeping notes of their relationships and connections through time. The data is then easy to interpret for both humans and, as researched by the writers, LLMs such as GPTs.
Given the sequential nature of the dataset, trying to predict the shape, or at least the represented action, of the next graph seemed nearly natural. The original work regarding EASG already tried this task with GPT3 text-davinci-003 [3], which features almost 175 billion parameters, so it appeared possible to approach the problem with a lighter model.
Graph Neural Networks [4] seemed a more natural choice for processing the data present in this newly created dataset. This category of model is already in use in numerous fields such as Social Networks, Natural Language Processing, and Chemistry, presenting wonderful results. Heterogeneous GNNs [5] offer the possibility of processing heterogeneous graphs, graphs that present different types of nodes and edges and EASG fits the description perfectly. To perform the task, different types of Heterogenous GNNs were used, combining different aspects of graph processing. Neighborhood sampling with SAGE convolution [6], attention mechanisms with GAT layers [7], and even a graph-suited adaptation of the popular transformer were used [8].
Through this work, it was possible to taste the deep representation capabilities of the EASG dataset even with smaller and lightweight models.
The document is divided into 6 sections, including this introduction.
In the **second chapter**, it's possible to consult the theoretical foundations that made this work possible.
In the **third chapter**, the methods used for parsing the dataset and creating

the models are illustrated.
In the **fourth chapter**, the setup used for conducting the experiments is described, comprehensive of hardware and software tools.
In the **fifth chapter**, the most relevant results of the experiments are listed, along with a brief explanation.
In the **last chapter**, the results are then discussed and the future steps to continue this work are listed.

Chapter 2

Related Work

2.1 Egocentric Action Scene Graphs

Egocentric Action Scene Graphs (EASG) [2] is a new representation for long-form understanding of egocentric videos. While previous approaches to the annotation of egocentric videos employed simple representations such as verb-noun pairs, EASG provides a temporally evolving graph that aims at creating a better understanding of the actions performed during the duration of the clip. It's built upon the Ego-4D dataset [9], a massive-scale, egocentric dataset, that contains a wide variety of videos representing actions in egocentric vision. EASG annotates a chosen set of videos present in this dataset.
EASG is divided into scenes, sequences of actions extracted from an egocentric video. Each of these actions is represented by a graph $G(t) = (V(t), E(t))$ where $V(t)$ and $E(t)$ are the sets of vertices and edges that compose the graph. A graph $G(t)$ is composed of three different temporal realizations: the precondition PRE, the point of no return PNR, and the postcondition $POST$. These three temporal realizations represent the three main frames where an action is occurring. The frames are then associated to $G(t)$ as $F(t) = \{PRE_t, PNR_t, POST_t\}$. Every $G(t)$ has two fixed nodes: $v_{\text{cw}}(t)$ and $v_{\text{verb}}(t)$. The first one represents the camera wearer while the second represents the verb characterizing the action. The graph then includes a set of nodes $V_{\text{obj}}(t)$, encoding the objects involved. The final set of vertices $V(t)$ is given by the following expression

$$\{v_{\text{cw}}(t), v_{\text{verb}}(t)\} \cup V_{\text{obj}}(t)$$

. Every node has an assigned set of attributes through the function att. We define $att(v_{\text{cw}}(t)) = \emptyset$ and $att(v_{\text{verb}}(t)) = verb$ where $verb$ is the verb class. Each node $v_i(t)$ representing an object has associated with it a class attribute that defines the object class and three bounding boxes, one for each frame presented in the image of $F(t)$. The resulting att function for object nodes is defined as follows $att(v_i(t)) = (noun, box_{PRE}, box_{PNR}, box_{POST})$. The edges of the graph then describe the relationships between the different entities. An edge between the nodes $v_i(t)$ and $v_j(t)$ represents a relationship within the two entities at time t. It's represented in the graph as the tuple $(v_i(t), v_j(t)) \in E(t)$. The paper in question defines a function called e_t that returns a value r when called on two nodes related to each other and returns $\emptyset$ when they're not connected. A value

r is an element of R, a set of all the possible relationships between nodes.
There are two different types of relationships between objects and verbs. The first one is of the *direct object* type while the second is simply a preposition. Between object and object, there only could be a relationship characterized by a preposition. The dataset contains information about all the objects present in the scene but only objects involved in the action are represented by nodes.

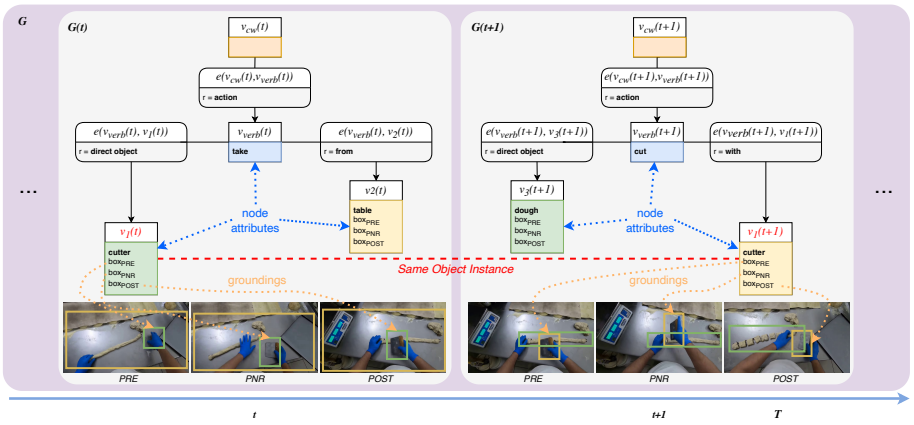


Figure 2.1: Example of two consecutive actions represented in the EASG format [2]

2.2 Heterogeneous Graph Neural Networks

While previous works proposed semantic-aware approaches [10], Heterogeneous Graph Neural Network (HetGNN)[11] proposes a model capable of handling processing over heterogeneous graphs. This work poses the base for heterogeneous graph neural network learning. It identifies the three main challenges of heterogeneous graph processing in neural networks(see Fig. 2.2) :

1. Sampling heterogeneous neighbors
2. Encoding heterogeneous content
3. Aggregating heterogeneous neighbors

Firstly, HetGNN addresses the challenge of neighbor sampling in heterogeneous environments. Using a random walk strategy stemming from a given node v, it generates a heterogeneous neighborhood $N(v)$, composed of different relationships and node types.
Then, the network utilizes a content embedding mechanism to process the just obtained information. Given the previously extracted neighborhood $N(v)$, for every node u the model extracts a set of features $h(u)$, modifying the process depending on the type of the node.
To accurately integrate the just obtained embedding, the model uses a two-step aggregation approach. The first step consists of using a type-specific aggregation function to aggregate the content features for each type of node. The second

step consists of using a recurrent neural network to combine all the type-specific embeddings.
This mechanism enables HetGNN to create a comprehensive representation of each node, incorporating all the different information from its neighborhood. From this baseline, it has been possible to proceed and include additional layers of processing to successfully create models capable of performing action prediction.

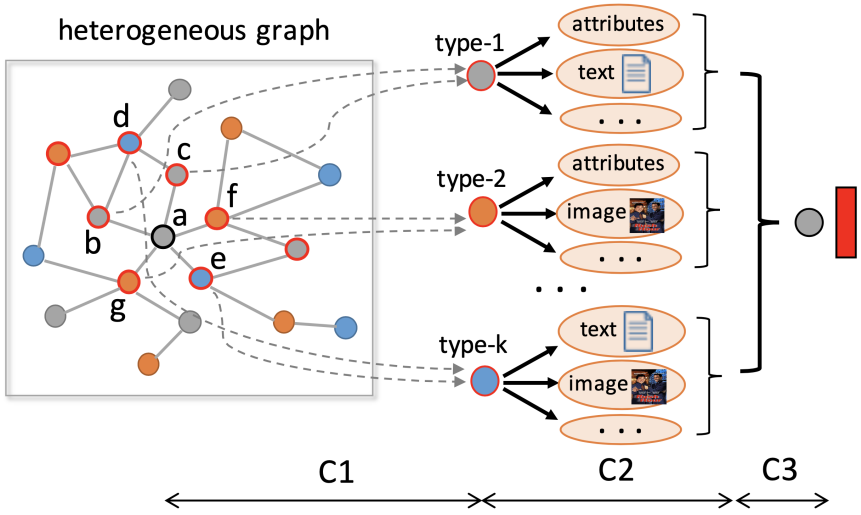


Figure 2.2: Challenges of graph neural network on heterogeneous graphs: C1 - sampling heterogeneous neighbors, C2 - encoding heterogeneous contents, C3 - aggregating heterogeneous neighbors [11]

2.3 PyTorch Geometric

PyTorch Geometric (PyG)[12] is a library for deep learning capable of handling irregularly structured data such as graphs and point clouds. It's built upon PyTorch and utilizes sparse GPU acceleration by providing dedicated CUDA kernels. All its methods support CPU and GPU computation and follow an immutable data flow paradigm. It has a wide variety of features dedicated to efficient graph structure handling and processing such as neighborhood aggregation, global pooling, and dataset processing.
PyG implements a wide variety of convolutional operators suitable for irregular domains usually called message passing techniques. To do so, the library offers a general MessagePassing interface for easily creating custom layers ready to be used. It also offers a large choice of already implemented convolutional layers presented in the latest papers, such as SAGE convolution[6], Graph Attention Networks[7], Graph Isomorphism Networks[13], and Dynamic Neighbourhood aggregation[14].
To properly support graph-level outputs, it offers a variety of readout functions such as global add, mean, or max pooling but also some more sophisticated

techniques such as sort pooling or global soft attention layer.
PyTorch Geometric also provides a consistent data format with a friendly interface for the creation and modification of datasets. Datasets can be created from an exhaustive range of files and be processed through the transform method, useful for performing data augmentation. Also, several of the most used datasets, e.g. PROTEINS[15] and IMDB-BINARY[16], are already provided through an easy-to-use API, that allows download and transforms.
PyTorch Geometric is a fundamental staple of the following work, offering interfaces and methods to deal with the complexity of heterogeneous graphs.

Chapter 3

Methodology

3.1 Task

Action prediction is the task performed by the model that will be presented and which around the dataset has been adapted. The goal, when performing action prediction, is to anticipate future actions based on a sequence of data describing previous behavior, such as a sequence of images, audio, or, in this case, graphs. To perform the task, subsequences of fixed length have been extracted from the dataset. The length of the extracted sequence has been defined as τ_o. This extracted sequence is composed of a fixed number T of temporal observation, or unique graphs g_t. The final goal is to be able to observe the T previous graphs and predict, with accuracy, the action present in the graph $T+1$ (see Fig. 3.1). An action is described by its verb and the verb's direct object. These two values must both be predicted simultaneously.
Since this task will not be performed on images, audio, or videos, given the fact that the dataset in question doesn't contain them, is important to define correctly in the scope of graph data. The task of action prediction can be defined as a task of graph classification. Given a graph, it will be classified as the next predicted action.

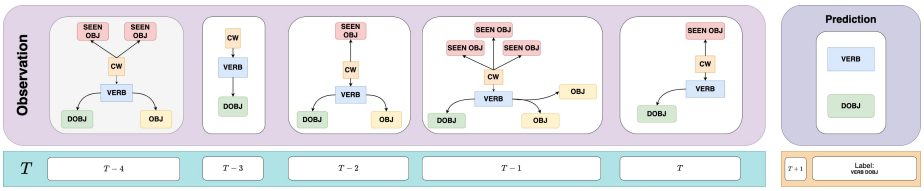


Figure 3.1: Example of a generic sequence and label

3.2 Dataset

The previously presented EASG [2] dataset needed some processing to be used inside a proper Graph Neural Network [4]. The two main steps in the processing were adapting it to the PyTorch Geometric [12] format and enhancing it to convey even more detailed information.

3.2.1 Data Cleaning

Before actually parsing the data, a major cleanup was needed to adapt the dataset to this specific task. A vast number of cleanup tasks were performed. The major ones were: creating a complete list of all the annotations used in the graphs (see Tab. A.1), removing corrupted graphs that presented no CW nodes or more than one verb, fixing spelling errors, and removing empty graphs.

3.2.2 Parsing

To be understood by a modern GNN the dataset needs to be expressed as a collection of tensors. To do so, the HeteroData [17] format native to the PyTorch geometric library has been used. The HeteroData format is the most suitable for the task since it allows encoding different types of nodes and edges in a single object. An HeteroData object has two main components:

1. The nodes, accessible through a string representing their name
2. The edges, accessible through a tuple in the following form (*node type*$_1$, *relationship*, *node type*$_2$)

The first step in the process was to encode the features of the nodes into a matrix of features. Every node inside the original object has an attribute x where this matrix is stored. Since there are only two different types of nodes inside the original dataset, the encoding was performed as follows. For verb nodes, the only features saved were the actual verbs they were representing. To do so, a list of all the possible verbs has been compiled so that the index of the specific nodes can be saved in the feature matrix. For object nodes, the bounding boxes in the three frames analyzed had to be stored too, so the resulting vector is in the form $(object\ category, box_{PRE}, box_{PNR}, box_{POST})$. To store the object category the same technique used for verbs was employed.

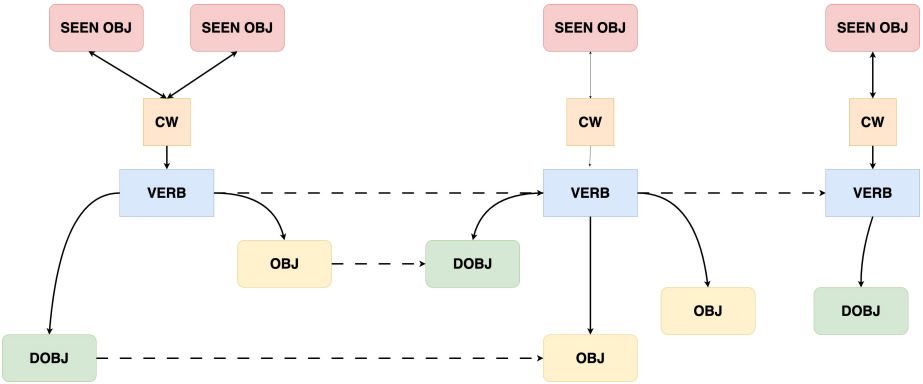


Figure 3.2: Example of three actions represented using the EASG format augmented trough the addition of edges bewtween verbs and nodes representing objects not present in the action

The second step, once nodes were successfully collected and represented, was to connect them between each other encoding edges. To do so, PyTorch Geometric

typically uses what's called an edge index, a tensor containing two different one-dimensional tensors of the same length. The first one stores the indexes of the source nodes while the second stores the indexes of the target nodes. To store the original edges from the source dataset, two edge indexes have been sufficient. The first one stores the connection between the verb and the objects, while the second represents connections between objects. Unfortunately, the edge index is not bound to store any edge feature, so another structure has been used to store the features representing the relationships between the two nodes.
To conclude the representation of the original dataset, it has been proven necessary to include the representation of edges in between nodes representing the same object in different and subsequent graphs. No features were necessary to characterize this relationship since there was no special relationship identifier involved. An edge is also been added in between subsequent verb nodes to avoid disconnected graphs (see Fig. 3.2).
In conclusion, parsing the dataset produced 221 HeteroData objects, representing sequences of graphs of a wide range of lengths.

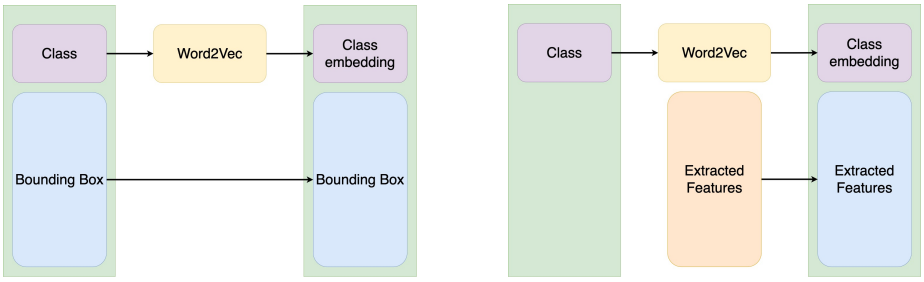
3.2.3 Sequence Extraction

To correctly perform the task previously exposed, it's necessary to define a way of extracting subgraphs of fixed length from the generated HeteroData objects, in consideration of their initial structure and characteristics. To do so, in the process of parsing, every node has been associated with a unique identifier to correctly assign it to its original graph. The process then consisted of selecting a span of the chosen length of subsequent graphs and only accepting nodes and edges from a particular sequence whose IDs were in the list. To do so PyTorch Geometric offered a useful interface, subgraph, capable of handling the process [17]. Then, the label for the graph, composed of a verb and its direct object, was extracted from the immediate next graph. All the original graphs that weren't long enough were previously removed to preserve the functionality of the process.

3.2.4 Data Enhancement

After the extraction of the subgraphs and the creation of the dataset, a wide range of techniques were employed to experiment and try to improve the model performance.
The first addition to the original data was to include, with a specific connection to the camera wearer node, all the objects present in the graphs not selected in the creation of the subgraph. This has been done because the wearer might have the intention to use objects that weren't seen in the last n graphs and through this representation they are kept in consideration.
The second technique consisted of including all those objects that were present in the scene but were not taking part in any action. As explained before, even if the original dataset didn't include them as nodes, these objects might be part of future actions and therefore worth keeping in consideration.
The next part of the data enhancement process consisted of including the features extracted from the original video in the Ego-4D dataset [9] through the use of a SlowFast model [18] inside the verb feature matrix. These features were used in the task of generating the original EASG graphs and they could prove

interesting to use.
Lastly, a Word2Vec [19] model has been employed to improve the representation of real-world objects, actions, and relationships. Since the method previously used consisted of a simple list of words identified by an index, a better semantic representation of words was needed. The model employed was the "word2vec-google-news-300" [20], trained on the Google news database of articles and returns a 300-dimensional tensor representing the input word.



(a) Process of enhancing object nodes (b) Process of enhancing verb nodes

Figure 3.3: Process for enhancing verbs and object nodes through the addition of Word2Vec embeddings and aggregating the pre-extracted features

Through these techniques, the dataset can be considered detailed enough (see Fig. 3.3) to be used for the task in hand.

3.3 Models

3.3.1 Graph Neural Networks

Graph Neural Networks [4] is a type of neural network specifically designed to process graph-structured data. The main idea behind this specific architecture is that each node in a graph is defined not only by its features but also by the other nodes in this neighborhood. We can represent this concept through the notion of state. Each node, before any computation, contains its features, the components of its initial state. Every iteration, using a technique called message passing, calculates a new state by combining the information from a specific edge and the information present in its neighborhood.
Message passing is defined as follows. Let a graph be $G = (V, E)$ and one-hop neighbourhoods be $N_u = \{v \in V | (v, u) \in E\}$. In addition, node features are collected in a matrix $X \in R^{|V| \times k}$ and the features of a single node u are expressed as x_u. Then the message passing procedure to compute the state h_u is

$$h_u = \phi(x_u, \oplus_{v \in N_u} \psi(x_u, x_v))$$

where $\psi : R^k \times R^k \to R^l$ is a message function, $\phi : R^k \times R^l \to R^m$ is readout function and $\oplus$ is a permutation-invariant aggregation function like $\sum$ or max. The message function is responsible for generating the message that nodes share. It accepts the features vector x_u and x_v and produces a message m_{uv}.

The readout function is the function responsible for aggregating all the information received from the neighborhood nodes and the original features inside the new state of a node u.
The permutation invariant aggregation function is then necessary to stack together all the messages received before the readout function computes the new representation of the node. Fundamentally, the aggregator is permutation invariant, since the messages could be received in any order, given the nature of a graph.
Inside this particular process lays the essence of a GNN. The readout function and the message function compose the learnable part of message-passing procedures, containing weights and biases responsible for shaping the behavior of the whole network.

3.3.2 Heterogeneous Graph Neural Networks

GNN responds to the need for Neural Networks that can gracefully process graph-like data but they are not capable of handling such data when the number of type of nodes and edges increases. As proposed, it's possible to process such graphs by modifying the underlying structure of the original GNN. To do so, instead of simply applying message-passing layers to the data which would not be possible since there are differences in the features matrix of different types of nodes, it's necessary to define a message-passing layer for every type of edge in the structure of the heterogeneous graph (see Fig. 3.4). This way, by iterating through the different edge types, the model can compute the resulting new states, obtained by aggregating the result of each layer. HetGNN [11] proposed the use of GRU [21] to perform such aggregation but PyG [12] provides a wide variety of other techniques. The one used was the 'mean' aggregation that, as the name suggests, computes the average of all the layers computing the state of any node.

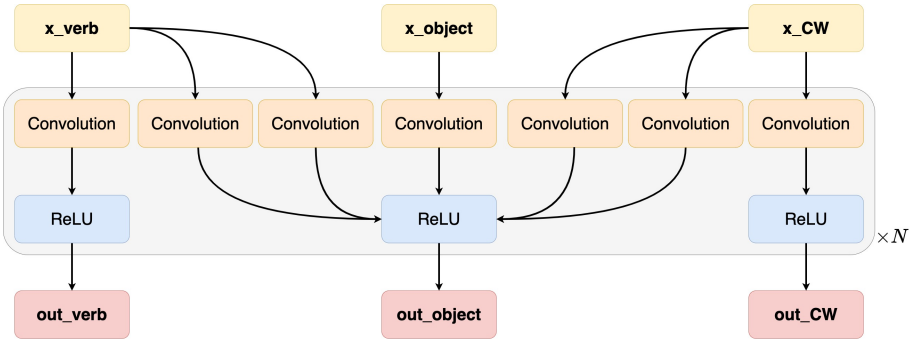


Figure 3.4: Illustration of the structure of a generic heterogeneous GNN operating on the EASG dataset. Not all edges were reported to increase readability.

3.3.3 Proposed Models

All the proposed model have two main components. A Heterogeneous Graph Neural Network and a classifier. The first component is the one that underwent the most variations. In the final iteration, three architectures were kept:

1. GraphSAGE
2. Graph Attention Networks (GAT)
3. Transformer

GraphSAGE

GraphSAGE [6] is a framework designed to learn node embeddings from a node's local neighborhood. It doesn't need the entire graph in training but it is still able to generalize to unseen nodes.
Through the use of a sampling strategy to generate node neighborhoods and a subsequent aggregation function, it is capable of combining information from node neighborhoods and computing node embeddings even for the nodes not present during training (see Fig. 3.5). The main operator of the model is

$$x'_i = W_1 x_i + W_2 \cdot \text{mean}_{j(i)} x_j$$

where x'_i is the next state of the node x_i, W_1 and W_2 are learnable weight matrices, $N(i)$ is a fixed-size obtained by sampling the neighbors of the node i and x_j is the state of one of those neighbors.
The mean function is not the only aggregator presented alongside the Graph-

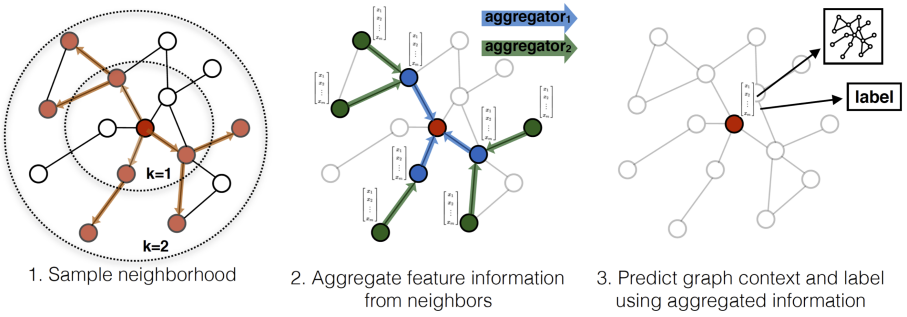


Figure 3.5: Visual illustration of the GraphSAGE sample and aggregate approach. [6]

SAGE model but it's the most straightforward and the one that lets the final model be lightweight. This architecture is usually employed for node classification and link prediction tasks, but its capability of bringing out the hierarchical structure in the final representation makes it suitable for graph classification tasks like the one in question.
In the experiment conducted for this work, GraphSAGE was applied through the use of the SAGEConv layer provided by PyTorch Geometric. Three layers were used, each of them followed by a ReLU activation function.

GAT

The Graph Attention Network (GAT) [7] was the first introduction of the concept of attention in the world of Graph Neural Networks. Previous works gave the same importance to every connection between nodes, applying a simple convolution as seen in GCN [22], while GAT uses the attention mechanism to weight

the relevance of connections between nodes and give a better representation of the graph. This weight is represented by an attention score each node has for its neighbor.

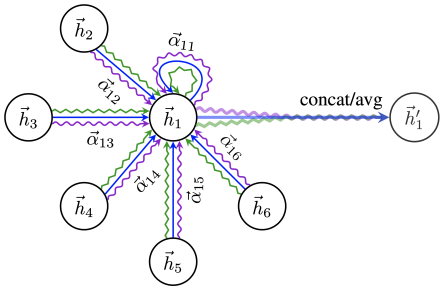


Figure 3.6: An illustration of multi- head attention (with 3 heads) by node 1 on its neighborhood [7]

This attention score is then fundamental to determine how much the central nodes depend on the representation of their neighbors. The core of the GAT architecture is

$$h_i^{(l+1)} = \sigma(\sum_{j \in N_{(i)}} \alpha_{ij}^{(l)} \times W^{(l)} \times h_j^{(l)})$$

where $h_j^{(l)}$ is the feature vector of node i at layer l, $N(i)$ represents the neighborhood of node i, σ is an activation function and α_{ij}^{l} are the attention scores for the layer l. The attention score α_{ij} are

$$\alpha_{ij} = \frac{\exp(\textbf{LeakyReLU}(\vec{a}^T[W\vec{h}_i \| W\vec{h}_j]))}{\sum_{k \in N(i)} \exp(\textbf{LeakyReLU}(\vec{a}^T[W\vec{h}_i \| W\vec{h}_j]))}$$

where $\vec{\mathbf{a}}^T$ is a vector parametrizing a single-layer feed-forward neural network used for calculate attention scores, W is a weight matrix and $\|$ is the concatenation operation. It's possible to note that a softmax function is being applied to make sure that the attention scores are thoroughly distributed and remain within the set boundaries. GAT also utilizes multiple attention heads to obtain a more clear and complete representation of the input data. The results of multiple heads are then concatenated or averaged (see Fig. 3.6).

Transformer

The graph transformer architecture [8] employs a similar mechanism to the one just presented for GAT. There still are some prominent differences that make the presence of both worthwhile for this work. The first, and most significant, difference is that the transformer computes the attention values for every single node, trying to grasp what relation may be present between two distant nodes. To do so, it applies a series of linear transformations to all the nodes of the graph to create the Query and Key vectors typical of the traditional transformer architecture [23] (see Fig. 3.7). The attention coefficient $\alpha_{i,j}$ are then given by

$$\alpha_{i,j} = \text{softmax}(q_i^\top \cdot (k_j + e_{ij}))$$

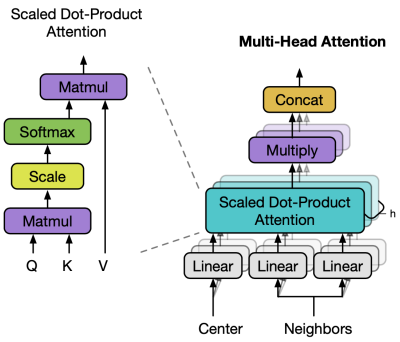


Figure 3.7: The structure of a graph transformer convolution layer. [8]

where q_i and k_j are the query and the key vector and e_{ij} are the edge features, if present, of the edge ij. To obtain the last vector typical of the transformer another linear transformation is applied to the nodes features vector. The final state of a node i is then obtained through

$$\hat{h}_i^{(l+1)} = \Big\|_{c=1}^{C} \Big[\sum_{j \in \mathcal{N}(i)} \alpha_{c,ij}^{(l)} (v_{c,j}^{(l)} + e_{c,ij}) \Big]$$

where $\|$ is the concatenations operation and C is the number of attention heads.

From homogeneous to heterogeneous

These three models were created with homogeneous graphs in mind but, through the use of the interface provided by PyTorch Geometric, it has been possible to convert them to accept heterogeneous graphs. A layer with the given architecture was deployed, as previously described, for every type of edge present in the heterogeneous data. The functionality of the layers was not affected.

Classifier

The classifier is the only part that remains consistent through all the implementations(see Fig. 3.8). Since Graph Neural Networks output a new representation of the nodes and their features, a classifier is then needed to extract the result of the prediction. The network operates in two branches one for verbs and one for objects. Since it has to predict an action, defined as the conjunction of verb and object, the first three fully connected layers of the processing are the same for both types of nodes. After each linear layer, ReLU is applied. The two node types are treated differently only in the final layer, where each of them has a dedicated processing. Once both of the nodes are ready, aggregation is done to compute the results of the prediction and Softmax is applied.

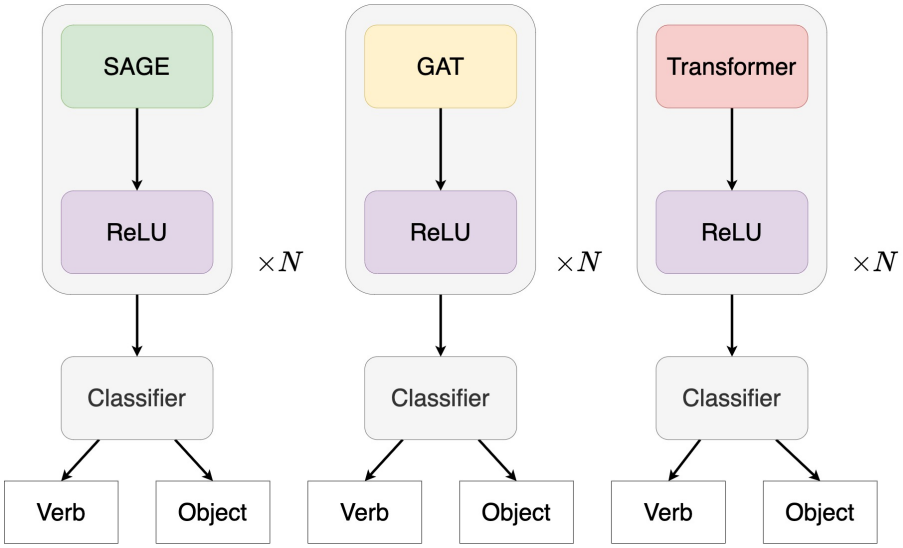


Figure 3.8: The structure of the proposed models.

Chapter 4

Experimental Setup

4.1 Tools and Hardware

As far as the tooling goes, the main software used was Google Colab [24], to train the models and parse the dataset, and Weight And Biases [25], as an experiment tracker for easy comparison and storage of the results. The model was trained on the processor offered by the coding platform, in particular: NVIDIA A100 40GB, NVIDIA T4 16GB, and NVIDIA L4 24GB.

4.2 Hyperparameter Settings and Model Configuration

The hyperparameters to consider during this training process were numerous.

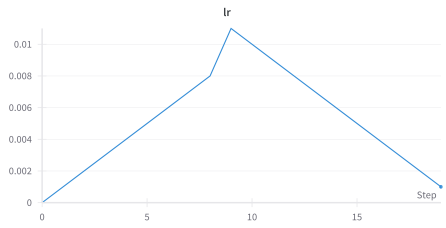


Figure 4.1: Value of learning rate at a given epoch

Firstly, as far as the loss function goes, the choice fell on the Cross-Entropy Loss function, since it's the most suitable for classification tasks. Since the classes to predict were two, two functions were used to quantify the error. To calculate the final loss two methods were adopted: the first one consisted of merely doing a weighted sum of the two losses, giving a slightly higher value to the object loss since the models generally had a bit more difficulty predicting them, while the second one consisted in weighting the sum with two learnable parameters so that the model could autonomously learn the importance of both predictions, as in Kendal et al.[26]. The formulation of the final loss L_{final} was then

$$L_{\text{final}} = \sum_{i \in L} w_i L_i$$

where L is the set containing the single losses and w_i is a learnable parameter, part of the set of weights W.
Stochastic Gradient Descent was used as the optimizer with a learning rate of 0.01 and a momentum of 0.9. A learning rate scheduler was also employed to gradually adjust the learning rate to avoid local minima and adapt to the different training stages. The adopted policy consisted of a linear warmup and decay. The learning rate was given by the product of the following function,

$$\text{lr\_coefficient} = \begin{cases} \frac{epoch}{peak\_epoch} & \text{if } epoch < peak\_epoch \\ \\ \frac{total\_epochs - epoch}{total\_epochs - peak\_epoch} & \text{otherwise} \end{cases}$$

where epoch represents the current epoch, peak_epoch is the index of the epoch where the learning rate is supposed to be maximum and total_epochs is the number of total epochs and the maximum learning rate. To increase training efficiency, He's weight initialization [27] was used in some experiments. He's initialization consists of initializing any linear layer's weight matrix as

$$W \sim \mathcal{N}(0, \sqrt{\frac{2}{n_{\text{in}}}})$$

where $\mathcal{N}$ is a normal distribution and n_{in} is the number of input units in the weight tensor. This technique also helps tackle the vanishing and exploding gradients.

Chapter 5

Results

5.1 General

The experiments were conducted on sequences of length of 20 single graphs. Since this work has a dual purpose, evaluating the changes done on the dataset and the capabilities of heterogeneous graph neural networks, a series of experiments took place to give the best possible panoramic. All three models were tested on the full augmented dataset and the results were compared against the inference done in Furnari et al. [2] with the GPT3 text-davinci-003 [3]. The augmentations in question are the inclusion of the pre-extracted features and the Word2Vec embeddings [19].
The metrics used to evaluate the models were two: Top-1 accuracy and Top-5 accuracy [28]. Top-1 accuracy measures the percentage of times the model's prediction with the highest probability is the correct one. Top-5 accuracy is based on the same concept but it takes into account the five highest probabilities. These two metrics are used in classification tasks but are especially useful in action prediction applications since they account for the inherited difficulties of such tasks. These two metrics were applied to the ability of the model to predict the correct verb, object, and action separately. The results obtained

model	**verb**		**object**		**action**	
	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
GPT	5.94	14.97	47.36	67.26	3.40	9.24
SAGE	14.67	35.33	10.89	10.92	**2.22**	5.33
GAT	13.78	**40.67**	**20.67**	**21.11**	1.56	**7.77**
Transformer	**14.89**	31.33	0.18	0.22	0	0

Table 5.1: Results of the tests on the dataset with all augmentations

(see Tab. 5.1) provide interesting insights.
Firstly, it's evident that, for most of the measured metrics, the baseline model performed better. While this proves that the models are not enough compared to GPT for grasping all the information that the dataset conveys, it shows an underlying potential that could be unveiled in the future. To support such claims, it's convenient to observe the results for the GAT model [7]. This model not only achieved comparable results concerning the action prediction metrics,

but it also exceeded the baseline posed by the GPT model on the verb prediction metrics. While this can't be considered a victory on all fronts, it indicates an unexpressed potential that with more research could be found and used.
Secondly, while in the baseline the object predictions were far more accurate than the verb predictions, the opposite was true for all the models examined. This might be caused by the fact that verbs were accompanied by pre-extracted features, giving way more information to the models. This may signify a need for pre-extracted features also for what regards objects.
Then, it's appropriate to analyze the dataset's characteristics, especially its label distribution. Observing the frequency of verb labels (see Fig. 5.1), it's

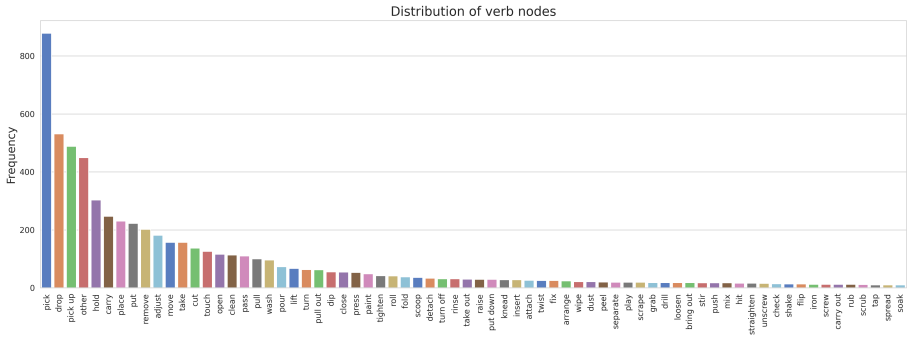


Figure 5.1: Distribution of verb labels. [2]

evident how a long tail distribution appears. This type of distribution occurs when a few in a wide range of values have a significantly higher probability than the rest. This is mainly caused by the different scenes included in the original EASG dataset. This distribution entails a wide range of problems for the training of deep learning models in general. For example, given the wildly different frequency of some classes, the model may be prone to overfitting when trained on this data. It also may incur some generalization problems given the underrepresentation of certain classes. This is a factor to keep in mind when evaluating a model trained on data with such distribution.
The same can be said about the frequency of object labels (see Fig. 5.2).
Lastly, the main reason for the baseline success has to probably be searched in

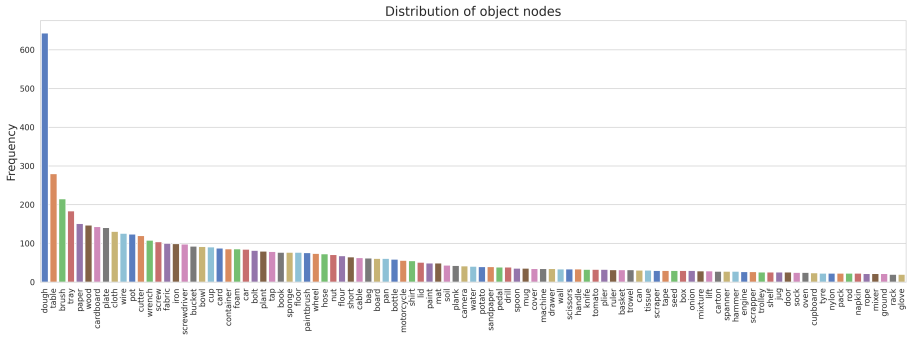


Figure 5.2: Distribution of object labels. [2]

its development. GPT [3] is, as the name conveys, a pre-trained model, trained on 570 gigabytes of text data from a wide range of sources. While it's true that the dataset augmentations used Word2Vec to try and convey semantic value to the verb and objects, it's also true that none of the examined models had access to the same quantity of data. This highlights the potential of these models, capable of achieving such results even without extensive pre-training procedures.

5.2 SAGE

The results relative to the SAGE model [6] provide interesting insights into the effectiveness of the dataset augmentations.
It's evident that the dataset with all the augmentations applied provides the most value and it's the most effective for the Sample and Aggregate architecture. It obtains the best results in 5 out of the 6 metrics, proving how the augmentation for this model was ultimately beneficial.
The model performs the worst when the pre-extracted features are not added to the data. This highlights the model's need for rich data, identifying the inadequacy of the Word2Vec features alone. Without the pre-extracted features, the model is not even capable of predicting any of the objects present in the action, once again indicating the strong relation between the two.
Lastly, the model achieves slightly better results when only the pre-extracted features are present, while still heavily underperforming. A little improvement had been seen in the object prediction field while remaining comparatively worse than the dataset with full augmentation.

	verb		**object**		**action**	
	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
Full	14.67	**35.53**	**10.89**	**10.92**	**2.22**	**5.33**
No pre-extracted	12.89	31.33	0	0	0	0
No Word2Vec	**15.11**	27.78	1.12	1.33	0.18	0.22

Table 5.2: Testing results for the SAGE model

5.3 GAT

The GAT architecture is by far the best-performing of the examined architectures. When trained on the dataset with full augmentation it achieves results close to the baseline model, while having significantly fewer parameters and training data.
When trained on the dataset without the pre-extracted features the model misses the target and achieves its worst performance, once again proving the need for rich features in such models. The performance is still better than the one reached by the SAGE model, demonstrating the power of the attention mechanism.
Without the Word2Vec embeddings, the results don't accuse much of a loss but

are still slightly inferior to the full augmented dataset.

	verb		**object**		**action**	
	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
Full	13.78	**40.67**	**20.67**	**21.11**	1.56	7.77
No pre-extracted	15.11	32.22	18.22	18.89	2.22	4.66
No Word2Vec	**15.33**	38.89	20.02	20.22	**2.66**	**8.44**

Table 5.3: Testing results for the GAT model

5.4 Transformer

The Transformer architecture [8] is by far the one that performed the worst. The reason may be found in its complexity, way above the order model proposed, and in the dataset structure. Is possible that the approach of the Transformer, based on finding the relationship between distant nodes, confused the data representation, masking the correct interpretation.
This model went against the trend that emerged in the previous analysis achieving its best performance in the training without the pre-extracted features. This further proves that the abundance of features confused the model, leading to its underwhelming results.
The other training runs, one with the full augmentation and one without the Word2Vec embeddings, both didn't predict a single accurate action, framing this architecture as the worst for this task and dataset.

	verb		**object**		**action**	
	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
Full	14.89	31.33	0.12	0.22	0	0
No pre-extracted	9.77	**42.44**	**10.82**	**11.33**	**1.77**	**4.89**
No Word2Vec	**15.56**	32.44	0	0.22	0	0

Table 5.4: Testing results for the Transformer model

Chapter 6

Conclusion

6.1 Final considerations

This work explored the application of heterogeneous graph neural networks on egocentric action scene graphs for the task of action prediction. The novel dataset treated in this work provided a unique perspective on egocentric vision, proposing an original method for annotating footage describing the action through its constituent methods.
The dataset was then augmented with a variety of techniques that, results in hand, proved useful and effective. The insights produced by the training on augmented dataset highlighted how heterogeneous graph neural networks thrive when given a large number of different features and don't shine when they're given an insignificant set of features. Given the increasing importance of such a task in the future landscape, this work constitutes a well-rounded panoramic of the situation.
The models tested achieved results comparable with the baseline that, given its increased training time and resources, constitute a significant achievement. The wildly different results obtained highlight the efficiency of Graph Attention Networks [7] compared to more classical convolution or well-known transformers. Low complexity model architectures gained better results compared to more complex models and attention mechanism still proved to be the best technique for the task.
Overall, the results paint a bright picture of the future potential of this specific task and set a heterogeneous graph neural network as a valid candidate to tackle it.

6.2 Future works

To proceed in this area of work a list of steps might be taken.
Firstly, additional augmentation may be needed to better semantically characterize the nodes in the dataset. Pre-extracted object features might be the best first step to do so, increasing the context available to the models. Also, improving the Word2Vec embedding mechanism [19] could result in a better representation of the elements of an action.
Then, different models could be tested on this dataset. To create a better

overview of heterogeneous graph neural network performances it's beneficial to try and apply different architectures to the task such as Residual Gated Graph Convolutional networks [29], GIN conv [13], or SplineConv [30]. There is also a wide set of other techniques that could prove effective such as pooling or dropout, both capable of reducing overfitting and improving performances.
Ultimately, an expansion of the dataset is due, since an increase in action representation means a better capability of generalization and more chances for better results.

References

[1] J. Engel, K. Somasundaram, M. Goesele, *et al.*, "Project aria: A new tool for egocentric multi-modal ai research," 2023. arXiv: 2308.13561 [cs.HC]. [Online]. Available: https://arxiv.org/abs/2308.13561.

[2] I. Rodin, A. Furnari, K. Min, S. Tripathi, and G. M. Farinella, "Action scene graphs for long-form understanding of egocentric videos," 2023. arXiv: 2312.03391 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2312.03391.

[3] T. B. Brown, B. Mann, N. Ryder, *et al.*, "Language models are few-shot learners," 2020. arXiv: 2005.14165 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2005.14165.

[4] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009. DOI: 10.1109/TNN.2008.2005605.

[5] X. Wang, H. Ji, C. Shi, *et al.*, "Heterogeneous graph attention network," 2021. arXiv: 1903.07293 [cs.SI]. [Online]. Available: https://arxiv.org/abs/1903.07293.

[6] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," 2018. arXiv: 1706.02216 [cs.SI]. [Online]. Available: https://arxiv.org/abs/1706.02216.

[7] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2018. arXiv: 1710.10903 [stat.ML]. [Online]. Available: https://arxiv.org/abs/1710.10903.

[8] Y. Shi, Z. Huang, S. Feng, H. Zhong, W. Wang, and Y. Sun, "Masked label prediction: Unified message passing model for semi-supervised classification," 2021. arXiv: 2009.03509 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2009.03509.

[9] K. Grauman, A. Westbury, E. Byrne, *et al.*, "Ego4d: Around the world in 3,000 hours of egocentric video," 2022. arXiv: 2110.07058 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2110.07058.

[10] Y. Dong, N. V. Chawla, and A. Swami, "Metapath2vec: Scalable representation learning for heterogeneous networks," pp. 135–144, 2017.

[11] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla, "Heterogeneous graph neural network," pp. 793–803, 2019.

[12] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," 2019. arXiv: 1903.02428 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1903.02428.

[13] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," 2019. arXiv: 1810.00826 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1810.00826.

[14] M. Fey, "Just jump: Dynamic neighborhood aggregation in graph neural networks," 2019. arXiv: 1904.04849 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1904.04849.

[15] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. V. N. Vishwanathan, A. J. Smola, and H.-P. Kriegel, "Protein function prediction via graph kernels," *Bioinformatics*, vol. 21, no. suppl$_1$, pp. i47–i56, Jun. 2005, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bti1007. eprint: https://academic.oup.com/bioinformatics/article-pdf/21/suppl_1/i47/524364/bti1007.pdf. [Online]. Available: https://doi.org/10.1093/bioinformatics/bti1007.

[16] P. Yanardag and S. Vishwanathan, "Deep graph kernels," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '15, Sydney, NSW, Australia: Association for Computing Machinery, 2015, pp. 1365–1374, ISBN: 9781450336642. DOI: 10.1145/2783258.2783417. [Online]. Available: https://doi.org/10.1145/2783258.2783417.

[17] "Heterogeneous graph learning." (2013), [Online]. Available: https://pytorch-geometric.readthedocs.io/en/latest/tutorial/heterogeneous.html (visited on 06/25/2024).

[18] C. Feichtenhofer, H. Fan, J. Malik, and K. He, "Slowfast networks for video recognition," 2019. arXiv: 1812.03982 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1812.03982.

[19] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013. arXiv: 1301.3781 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1301.3781.

[20] "Word2vec, tool for computing continuous distributed representations of words." (2013), [Online]. Available: https://code.google.com/archive/p/word2vec/ (visited on 06/28/2024).

[21] K. Cho, B. van Merrienboer, C. Gulcehre, *et al.*, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," 2014. arXiv: 1406.1078 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1406.1078.

[22] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2017. arXiv: 1609.02907 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1609.02907.

[23] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," 2023. arXiv: 1706.03762 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1706.03762.

[24] "Google colab." (2013), [Online]. Available: https://colab.research.google.com/ (visited on 06/29/2024).

[25] "Weight and biases." (2013), [Online]. Available: https://wandb.ai/home (visited on 06/29/2024).

[26] A. Kendall, Y. Gal, and R. Cipolla, "Multi-task learning using uncertainty to weigh losses for scene geometry and semantics," 2018. arXiv: 1705.07115 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1705.07115.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," 2015. arXiv: 1502.01852 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1502.01852.

[28] R. Wardle and S. Rowlands, "Deep-learning based egocentric action anticipation: A survey," Jul. 2023. DOI: 10.21203/rs.3.rs-3156532/v1.

[29] X. Bresson and T. Laurent, "Residual gated graph convnets," 2018. arXiv: 1711.07553 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1711.07553.

[30] M. Fey, J. E. Lenssen, F. Weichert, and H. Müller, "Splinecnn: Fast geometric deep learning with continuous b-spline kernels," 2018. arXiv: 1711.08920 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1711.08920.

Appendix A

Encoding details

A.1 HeteroData object

The HeteroData object representing a sub-sequence on which the training occurs has approximately the following shape:

```
HeteroData(
  extra_features={
    split='train',
    video_uid='41416e77-3b9a-4685-bd1b-ba26b5ba19b2',
    shape=[2],
    graph_uids=[20],
    label_uid='048
       de16984118093ecf33117058f3ce7075c899929b92ac71343e2c2e1e22e75
       ',
  },
  y_verb=[1],
  y_obj=[1],
  verb={
    x=[20, 2604],
    extra_features=[20],
  },
  object={
    x=[119, 312],
    extra_features=[119],
  },
  CW={
    num_nodes=20,
    extra_features=[20],
    x=[20, 1],
  },
  (verb, rel, object)={
    edge_index=[2, 24],
    edge_attr=[24, 300],
  },
  (verb, dobj, object)={ edge_index=[2, 20] },
```

```
  (object, rel, object)={
    edge_index=[2, 0],
    edge_attr=[0],
  },
  (object, time, object)={ edge_index=[2, 26] },
  (verb, next, verb)={ edge_index=[2, 19] },
  (CW, sees, object)={ edge_index=[2, 5] },
  (object, has_seen, CW)={ edge_index=[2, 1400] },
  (CW, has_seen, object)={ edge_index=[2, 1400] }
)
```

A.2 Labels

The following labels were used for objects and verbs:

Verbs	add, adjust, align, apply, arrange, attach, beat, bend, break, bring, bring-out, brush, carry, carry-out, carry-up, carve, change, check, chop, clean, clean-off, cleanse, clear, climb, close, close-back, coat, collect, connect, cover, crumple, cut, cut-off, cut-out, detach, dip, dip-in, disconnect, divide, drag, drill, drive, drop, drop-out, drop-up, dry, dust, empty, examine, fasten, feel, fetch, fill-up, fit, fix, fix-up, flap, flip, fold, force, glue, go, grab, grasp, grip, hammer, hang, hit, hold, hold-up, insert, inspect, iron, join, keep, knead, knit, lay, lay-down, leave, lift, lift-up, loose, loosen, loosen-out, lose, lower, mark, measure, mix, mount, move, move-off, move-up, open, operate, pack, paint, pass, peel, pet, pick, pick-out, pick-up, place, place-down, plaster, play, point, position, pour, pour-down, pour-off, pour-out, press, pull, pull-out, push, push-down, push-in, put, put-away, put-down, put-in, put-off, put-on, put-out, raise, read, release, remove, reposition, rest, return, rinse, roll, roll-up, rotate, rub, sand, scan, scoop, scoop-out, scrap, scrape, scratch, screw, screw-in, scrub, search, separate, set, sew, shake, shape, shave, shift, shove, shuffle, shut, slap, slice, slide, smoothen, soak, spin, split, spray, spread, spread-out, spread-up, sprinkle, squeeze, squeeze-out, stick, stir, store, straighten, straighten-out, stretch, sweep, swing, swirl, switch, switch-off, take, take-off, take-out, take-up, tap, taste, tear-off, test, throw, throw-away, tie, tight, tighten, tilt, touch, transfer, trim, turn, turn-off, turn-on, turn-out, turn-over, turn-up, twist, unfold, unhang, unlock, unplug, unscrew, untangle, untie, unwrap, uproot, use, wash, water, wear, wet, wipe, wipe-off, withdraw, wrap
Objects	adapter, apron, arm, art, artwork, bag, bar, basket, basin, bathtub, batter, battery, bed, belt-holder, bench, beverage, bicycle, bicycle-wheel, bike, bike-part, bin, block, board, bobbin, bolt, bolt-driver, book, booklet, books, bookshelf, both-hands, bottle, bow, bowl, bowls, box, boxer, brake, brake-shoe-pack, braking-system, branch, bread, brick, broom, broomstick, brush, bucket, bulb, bunch, button, cabbage, cabinet, cable, cables, caliper, camera, can, cap, car, car-part, carburetor, card, cardboard, carpet, cart, carton, case, casing, ceiling, cello, cement, center, chaff, chain, chair, charger, chip, chips, chisel, chocolate, chopping, chopping-board, clamp, cleaner, clip, clips, cloth, clothe, clothes, clothing-material, compartment, computer, connector, container, control, cooker, cooker-knob, comb, cord, cot, cover-lid, counter, countertop, cover, cracker, craft, crochet, crumbs, cup, cupboard, cutter, cutting-board, debris, derailleur, desk, detergent, dirt, dish, dishwasher, dog, door, dough, dough-strip, dough-strips, dough-machine, drawer, dress, drill, driller, drink, driver, drum, dry-stem, dust, dustbin, dustpan, egg, engine, eraser, fabric, fabrics, face, facemask, faucet, fence, file, filter, finger, fingers, floor, flour, flower, flower-pot, foam, food, fork, frame, frame-door, fridge, front, fuel, furniture, garbage, gasket, gauge, gear, generator, glass, glasses, glove, glue, gouge, grass, grater, grease, grinder, ground, guitar, hammer, hand, handle, hanger, heap, himself, hoe, hoes, holder, hook, hose, ice, ice-cubes, ice-tray, insulator, iron, iron-box, ironing-board, jack, jacket, jar, jug, keg, key, keyboard, knife, knob, knot, label, ladder, laptop, layer, leaf, left-hand, leg, lever, lid, lift, light, liquid, liquid-wash, lock, lubricant, machine, manual, marker, mask, mat, matchstick, material, measure, meat, metal, metal-board, metallic-object, milk, mirror, mixer, mixture, mop, mop-stick, motorbike, motorcycle, mouse, mouth, mower, mug, multimeter, nail, napkin, needle, net, newspaper, note, nozzle, nut, nylon, oil, onion, oven, other, pack, packet, pad, paddle, paint, paint-brush, paintbrush, palette, pan, pants, paper, papercraft, papers, part, pastry, patch, pedal, peel, peeler, pen, pencil, phone, photo, picture, piece, pieces, pile, piler, pin, pipe, pizza, plank, plant, planter, plastic-wraps, plate, plates, platform, platter, plier, pliers, plug, pocket, pole, polythene, pot, potato, pottery, pouch, pressure-hose, pruner, pruning-sheer, pump, purse, rack, rag, rail, railing, rake, refrigerator, remote, right-fist, right-hand, rim, ring, rod, rod-metal, roller, room, root, rope, ropes, rubber, ruler, sachet, salt, sand, sandpaper, sauce, saucer, saw, scaffold, scale, scarf, scissors, scoop, scooter, scourer, scraper, scrapper, screw, screwdriver, scrubber, seasoning, seat, seed, sellotape, serviette, sewing, shaft, shear, shears, sheet, shelf, shelve, shirt, shoe, short, shovel, side, sieve, sink, sink-faucet, slab, smartphone, soap, sock, socket, soil, spade, spanner, spatula, spice, sponge, sponge-foam, spoon, spray, spring, stack, stairs, stand, steel, steel-panel, stick, stool, stove, strand, string, structure, sugar, surface, switch, table, table-cloth, tablet, tag, tank, tap, tape, terminal, thread, tie, timber, timer, tin, tire, tissue, tissue-paper, tomato, toolbox, tools, top, torch, towel, toy, train, trash, tray, trey, trimmer, trolley, trouser, trowel, trunk, tub, tube, umbrella, undergarment, utensils, vacuum, vacuum-cleaner, valve, vase, vice, vine, waist, wall, wallet, wardrobe, wash, washer, washing, washing-machine, water, water-hose, weed, weighing-scale, weighing-machine, wheel, white-sheet, window, windshield, wipe, wiper, wire, wire-cutter, wires, wood, woods, wood-plank, wooden-block, wooden-rail, wooden-stand, workbench, worktable, worktop, wrapper, wrench, yarn, yarn-darner, yeast, zinc

Table A.1: Verbs and Objects labels