Università di Genova | **DIBRIS** DIPARTIMENTO DI INFORMATICA, BIOINGEGNERIA, ROBOTICA E INGEGNERIA DEI SISTEMI

Action Classification and Anticipation using Graph Neural Networks

Clara Mouawad

Master Thesis

Università di Genova, DIBRIS Via Opera Pia, 13 16145 Genova, Italy
https://www.dibris.unige.it/

Università di Genova

MSc Computer Science
Data Science and Engineering Curriculum

Action Classification and Anticipation using Graph Neural Networks

Clara Mouawad

Advisor: Nicoletta Noceti

Advisor: Federico Figari Tomenotti

Examiner: Matteo Moro

December, 2023

Abstract

Action recognition plays a pivotal role in understanding and interpreting human behavior within videos, finding applications across various domains such as surveillance, human-computer interaction, and robotics. Traditional methods predominantly rely on RGB images, treating actions as holistic events without capturing the intricate hierarchical part structures inherent in activities. In this context, Spatio-temporal Scene Graphs offer a transformative approach. Unlike conventional RGB-based approaches, Spatio-temporal Scene Graphs allow for the decomposition of actions into nuanced interactions between objects and their pairwise relationships, thereby capturing the temporal evolution of visual scenes during actions. This novel representation serves as a potent alternative to conventional RGB-based methods, offering a more detailed and interpretable depiction of actions. Moreover, recognizing the importance of forecasting actions before their completion, the thesis delves into the anticipation task. This task addresses the inherent challenge of understanding ongoing actions and holds significant potential for real-time applications, enabling systems to react proactively to unfolding events. The thesis delves into these aspects, providing insights into the efficacy of Spatio-temporal Scene Graphs for both Action Recognition and Anticipation tasks.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction and Motivation

In the rapidly evolving field of computer vision and artificial intelligence, the tasks of Action Recognition and Anticipation have emerged as pivotal domains with transformative implications. Action Recognition involves imparting machines with the ability to comprehend and interpret human actions depicted in images or videos, thereby enabling applications ranging from interactive interfaces to security surveillance and robotics. Anticipation, a more advanced facet, extends this capability by predicting actions before they fully unfold, thereby enhancing the proactive responsiveness of systems to dynamic scenarios. These tasks are not merely technical challenges but gateways to revolutionize human-computer interaction, fortify security measures, and optimize automation processes.

However, the journey towards achieving proficiency in Action Recognition and Anticipation is full of challenges. Action Recognition grapples with the intricacies of temporal dynamics, diverse environmental conditions, and the necessity to decipher actions within varying contextual settings [MBM14]. The temporal dynamics introduce challenges in capturing the nuanced progression of actions over time, making it difficult for traditional methods to discern complex interactions. [SNpS22] Additionally, diverse environmental conditions further compound the complexity, as actions may manifest differently depending on factors such as lighting, background, and occlusions [RYT14]. The need to decipher actions within varied contextual settings requires models to possess a contextual understanding that goes beyond simple recognition.

Anticipation, taking a step beyond, encounters additional hurdles [FF21] [HDL$^+$22]. Predicting actions before their full development demands the ability to discern subtle cues and infer potential outcomes. Understanding uncertainties associated with human behavior introduces challenges in accurately predicting actions in dynamic and unpredictable scenarios. Furthermore, adapting to rapidly changing scenarios poses a significant challenge, requiring models to exhibit a level of flexibility and adaptability to unforeseen circumstances.

Motivated by the need to overcome these challenges and to unlock the full potential of computer vision, and inspired by the work of others [JKFFN19] [Tri23] [RRF23], this thesis leverages the concept of Spatio-temporal as a foundational framework for addressing challenges in Action Recognition and extending it to the domain of Anticipation. Spatio-temporal Scene Graphs provide a comprehensive representation, capturing not only spatial relationships between objects but also the temporal evolution of these relationships over time. Our contribution lies in the distinct methodologies employed for Action Recognition and the novel extension of these scene graphs to the challenging task of Anticipation.

For Action Recognition, these graphs offer a structured framework that enriches the understanding of actions, allowing for nuanced modeling of intricate interactions. Leveraging Graph Convolutional Networks (GCNs), trained and optimized to enhance accuracy, becomes instrumental in capturing and learning complex relationships within the scene graphs, thereby improving the overall performance of Action Recognition [DLP21] [DMP21].

For Anticipation, the hierarchical nature of Spatio-temporal Scene Graphs becomes particularly crucial. This hierarchy facilitates context-aware recognition by explicitly modeling relationships between different elements, contributing to more accurate predictions of the final action. Although similar to Action Recognition in the utilization of GCNs, the scene graphs for Anticipation represent only a partial action. GCNs play a pivotal role in capturing and learning the temporal dependencies and intricate relationships within these scene graphs, enabling the model to make informed predictions about incomplete actions[DHJ+21] [TFL+22].

In essence, Spatio-temporal Scene Graphs, in conjunction with Graph Convolutional Networks, emerge as key contributors to advancing the accuracy of both Action Recognition and Anticipation, addressing the challenges posed by the intricate temporal dynamics, diverse environmental conditions, uncertainties in human behavior, and adaptability to rapidly changing scenarios. By providing a structured and comprehensive representation and harnessing the capabilities of GCNs, these approaches pave the way for more effective computer vision systems, enhancing their capabilities in understanding, predicting, and responding to human actions in diverse and dynamic environments. This thesis unfolds with the aim of delving into the theoretical foundations, presenting original contributions, and substantiating the effectiveness of the proposed model through experimental results and analysis.

1.1.1 Scope

The scope of this thesis is to advance the field of computer vision through a focused exploration of Action Recognition and Anticipation, leveraging Spatio-temporal Scene Graphs and Graph Convolutional Networks (GCNs). Building on existing knowledge and inspired by prior work [JKFFN19], our research extends the utility of Spatio-temporal Scene Graphs beyond Action Recognition to the challenging task of Anticipation. Through a carefully selected dataset and robust experimental protocols, our aim is to contribute novel methodologies that enhance the understanding, predictive accuracy, and responsiveness of computer vision systems in interpreting and anticipating human actions. This thesis seeks to provide valuable insights, bridging theoretical foundations with empirical evaluations to enrich the capabilities of contemporary computer vision models.

The main contributions of this thesis are summarized as follows:

- An analysis of the state-of-art models working on spatio-temporal scene graphs [JKFFN19]
- A detailed algorithm for producing spatio-temporal scene graphs from annotation files.
- An empirical evaluation for different structural choices in a graph neural network (choice of convolutional layers, hyperparameters and so on).
- An experimental evaluation for the tasks of action recognition and anticipation on a benchmark dataset using multiple metrics (precision, recall and mean average precision).

1.2 Structure of the Thesis

The remainder of this thesis is structured into several chapters, each systematically delving into the research topic to offer a comprehensive understanding of the proposed approach. The following provides an overview of the thesis structure:

- **Chapter 2:** This chapter spans critical concepts, including the fundamentals of Action Classification, Anticipation in Action Recognition, and the application of Neural Networks and Deep Learning in this domain. The discussion extends to Graph Neural Networks, particularly Graph Convolutional Networks and Graph Attention Networks, establishing the theoretical groundwork for the subsequent chapters.

- **Chapter 3:** This chapter provides an extensive review of related work, and state-of-the-art models, examining key areas such as Object Detection, Scene Graph Generation, Action Recognition, Graph Neural Networks and Anticipation. The synthesis of existing knowledge sets the stage for the original contributions presented in this thesis.

- **Chapter 4:** This chapter unveils the methodology adopted, encompassing the dataset used, the generation of Spatio-temporal Scene Graphs, the task of Action Recognition, and the novel Anticipation approach. The Graph Convolutional Network is strategically designed and experimented upon to attain optimal performance.

- **Chapter 5:** In this chapter, the experimental results unfold, showcasing the effectiveness of the proposed model in both Action Recognition and Anticipation tasks. The presentation includes a detailed analysis of the results on the dataset, offering insights into the model's capabilities.

- **Chapter 6:** The concluding chapter encapsulates the cumulative findings and summarizes its multifaceted contributions to Machine Learning and Computer Vision. It also presents future directions for research, proposing avenues for improvement and extending an invitation to future researchers to explore the proposed avenues, fostering continued progress in this dynamic field.

Chapter 2

Background

This chapter delves into the foundational concepts crucial for understanding the subsequent chapters. It explores the basics of Action Classification, the role of anticipation in Action Recognition, and provides an in-depth overview of Neural Networks and Deep Learning. Topics covered include key components such as layers, loss functions, and regularization techniques, along with an introduction to Convolutional Neural Networks (CNNs) and Graph Neural Networks (GNNs). The chapter concludes by discussing the applicability of GNNs to Action Classification and Anticipation.

2.1 Foundations of Action Classification

Action classification, within the realm of computer vision, is a fundamental task that involves the identification and categorization of human activities or movements within visual data, typically in the form of images or videos. This process is essential for machines to interpret and understand dynamic scenes, enabling them to make informed decisions or predictions based on observed actions. The foundations of action classification are integral to applications across diverse domains, ranging from surveillance to human-computer interaction and robotics.

Definition of Action Classification: In the context of computer vision, action classification refers to the computational process of recognizing and categorizing human activities or gestures depicted in visual data. This involves teaching machines to distinguish between various actions, allowing them to comprehend and respond to dynamic changes in their visual environment. Examples of actions may include walking, running, eating, drinking, or complex activities involving multiple interacting entities.

Importance Across Applications: The importance of action recognition extends across a spectrum of practical applications, each benefiting from the capability to understand and interpret human behavior:

- Video Surveillance: In surveillance systems, action classification enables the automatic identification of suspicious activities, facilitating real-time monitoring and threat detection.

- Human-Computer Interaction: Action classification plays a pivotal role in creating intuitive and responsive human-computer interfaces. Recognizing gestures or actions allows for natural interaction between users and devices, enhancing user experience.

- Robotics: In robotics, the ability to classify human actions is crucial for robots to navigate and interact with their surroundings. This is particularly relevant in scenarios where robots need to understand and respond to human commands or collaborate with human counterparts.

Challenges in Action Classification: While action classification holds great potential, it comes with inherent challenges that need to be addressed for robust and accurate performance

- Variability in Pose: Human actions can be performed in various poses and orientations, making it challenging to create models that generalize well across different body positions.

- Lighting Conditions: Changes in lighting conditions can significantly impact the appearance of actions, leading to difficulties in recognizing them accurately.

- Limited View: Depending on the type of images/ videos, it is very likely that the data is from one camera, and therefore lacking visibility from other view points.

- Temporal Dynamics: Actions unfold over time, introducing the temporal dimension to the classification task. Capturing and modeling temporal dependencies is crucial for accurately recognizing actions, especially in video sequences.

Understanding and mitigating these challenges are central to advancing the field of action classification and ensuring its applicability in real-world scenarios.

2.2 Anticipation in Action Recognition

Anticipation refers to the proactive ability of a system to predict and recognize actions before their completion, based on partial or incomplete information. In the realm of action recognition, this entails the foresight to identify and comprehend ongoing activities before they reach their conclusion. The anticipation framework aims to bridge the gap between the observed initial cues of an action and the full understanding of its intended or future trajectory. This section delves into the conceptualization of anticipation, its significance in real-time applications, and the inherent challenges and advantages it brings to action recognition systems.

Significance of Anticipating Actions:

Anticipating actions holds paramount importance, particularly in real-time applications and systems, for several compelling reasons:

- Enhanced Responsiveness: Anticipatory systems enable quicker responses to unfolding events by recognizing actions in their early stages. This is critical in time-sensitive applications where prompt decision-making is essential.

- Improved Interaction: In human-computer interaction and robotics, anticipating user actions allows systems to respond more intuitively and seamlessly. This enhances the overall user experience by creating interfaces that align more closely with natural human behavior.

- Predictive Analytics: Anticipation facilitates the integration of predictive analytics in various domains. By foreseeing actions, systems can make informed predictions, aiding in better planning, resource allocation, and decision-making.

Challenges of Anticipation in Action Recognition Systems:

Incorporating anticipation into action recognition systems comes with many challenges.

- Ambiguity in Early Cues: Early cues of an action might be ambiguous or insufficient, introducing challenges in accurately predicting the full action sequence.

- Temporal Variability: The temporal dynamics of actions, with varying speeds and durations, pose challenges in developing models that can effectively anticipate actions across different time scales.

- Uncertainty Handling: Anticipation introduces uncertainty, as predictions are made based on incomplete information. Effectively handling and quantifying this uncertainty is crucial for robust and reliable anticipatory systems.

Advantages of Anticipation in Action Recognition Systems: Even though Anticipation can be challenging, it brings many advantages.

- Improved Reaction Time: Anticipation significantly reduces reaction time by recognizing actions early, enabling systems to respond rapidly to dynamic environments and scenarios.

- Contextual Understanding: Anticipatory models inherently develop a contextual understanding of actions, considering not only the immediate visual cues but also the broader context in which actions unfold.

- Enhanced Performance: Anticipatory systems can enhance the overall performance of action recognition by providing a more comprehensive and timely interpretation of dynamic scenes.

Understanding the interplay between anticipation and action recognition is central to advancing the field and unlocking the full potential of intelligent systems.

2.3 Neural Networks and Deep Learning

Neural networks form the backbone of modern artificial intelligence, enabling machines to learn from data and make intelligent decisions. At the heart of this paradigm lies deep learning, a subfield of machine learning that leverages neural networks with multiple layers to extract intricate patterns and representations from complex data. This section provides a foundational understanding of neural networks and their evolution into deep learning.

2.3.1 Neural Networks

Defining Basic Components of Neural Networks: Neural networks are composed of interconnected layers, each consisting of nodes (or neurons) and weighted connections. The basic components include:

- Layers: Neural networks are organized into layers, each serving a specific purpose. The input layer receives data, hidden layers process information, and the output layer produces the final results.

- Nodes (Neurons): Nodes are the fundamental processing units within each layer. They receive inputs, apply a transformation using weights, and produce an output. The collective activity of nodes across layers forms the network's predictions.

- Weights: Weights represent the strength of connections between nodes. They are adjusted during the training process to optimize the network's performance by minimizing the difference between predicted and actual outcomes.

Mathematical Representation of Neural Networks: Neural networks are capable of capturing complex relationships in data through the iterative adjustment of weights during a training process. The output y_i of a node i in a neural network is calculated as follows:

$$y_i = \sigma(\sum_j w_{i_j}.x_j + b_i) \tag{2.1}$$

where:
- y_i is the output of node i,
- σ is the activation function,
- w_{i_j} is the weight connecting node j to node i,

- x_j is the input from node j,
- b_i is the bias term for node i.

2.3.2 Layers and Loss function

Important Neural Network Layers: Many layers can be used to build a network. We will focus on the ones needed for our work.

ReLU: Rectified Linear Unit also known as ReLU, is an activation function widely used in Neural Networks. It known for its simplicity and effectiveness. ReLU introduces non-linearity into the network, allowing it to learn complex relationships between inputs and outputs. Mathematically, ReLU is defined in Equation 2.2:

$$f(x) = max(0, x) \tag{2.2}$$

where x represents the input to the activation function, and $f(x)$ is the output. The ReLU function replaces all negative values in the input with zero and leaves positive values unchanged. This piece-wise linear function is computationally efficient and mitigates the vanishing gradient problem.

Dense Layers: Dense layers, also known as fully connected layers, constitute a fundamental building block in neural networks. These layers connect each neuron to every neuron in the preceding and succeeding layers, forming a dense matrix of connections. Mathematically, the output of a dense layer is calculated in Equation 2.3:

$$y = \sigma(W.X + b) \tag{2.3}$$

where X represents the input vector, W is the weight matrix, b is the bias vector, and σ is the activation function (such as ReLU). The dot product $W.X + b$ computes the weighted sum of inputs, and the activation function introduces non-linearity into the network. Dense layers play a crucial role in learning hierarchical representations of input data. Each neuron in a dense layer learns to capture specific features or patterns, and the collective behavior of neurons in subsequent layers enables the network to learn complex relationships. The number of neurons in a dense layer and the architecture of the network, including the number of layers, are hyperparameters that influence the model's capacity to learn and generalize.

Softmax: Softmax is an activation function and a fundamental component in neural networks, often used to convert raw output scores into probability distributions. Commonly employed in the output layer for multiclass classification tasks, softmax transforms the raw scores (logits) into a set of probabilities that sum to 1. The mathematical expression for softmax is given in Equation 2.4:

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \tag{2.4}$$

where $z = [z_1, z_2, ...z_k]$ represents the vector of logits for K classes. The function e^{z_j} exponentiates each logit, and the division by the sum of exponentiated logits ensures that the resulting values form a valid probability distribution.
Softmax introduces competition among the classes, emphasizing the prediction of the most probable class while suppressing others. The output probabilities can be interpreted as the model's confidence in assigning an input to each class. The softmax function is differentiable, making it compatible with gradient-based optimization algorithms like backpropagation.

Cross Entropy Loss: Cross Entropy Loss, or log loss, is a commonly used objective function in neural networks, particularly for classification tasks. It measures the difference between the predicted probability distribution and the true distribution of the target labels. Cross Entropy Loss is defined in Equation 2.5:

$$Cross\ Entropy\ Loss = -\sum_{i=1}^{N}\sum_{j=1}^{K} y_{ij}.log(p_{ij}) \tag{2.5}$$

where:
- N is the number of samples,
- K is the number of classes,
- y_{ij} is an indicator variable that equals 1 if the sample i belongs to class j,
- p_{ij} is the predicted probability of sample i belonging to class j.

The goal during training is to minimize the Cross Entropy Loss. Intuitively, this loss function punishes the model more severely when it confidently predicts the wrong class. The logarithmic term $log(p_{ij})$ amplifies the penalty for confidently incorrect predictions, making it a suitable choice for models aiming to output well-calibrated probabilities.

2.3.3 Evaluation Metrics

Precision: Precision is a fundamental metric in classification tasks, providing insights into the accuracy of positive predictions made by a model. It is defined as the ratio of true positive predictions to the total number of positive predictions, highlighting the model's ability to avoid false positives. Precision is particularly crucial in scenarios where the cost of false positives is high. Precision can be written as in Equation 2.6

$$Precision = \frac{True\ Positives}{True\ Positives\ +\ False\ Positives} \tag{2.6}$$

where True Positives represent the instances that were correctly predicted into their class, and False Positives represent the instances that were falsely predicted into that class.

Recall: Recall, also known as sensitivity or true positive rate, measures the ability of a model to capture all relevant instances of a class. It is the ratio of true positives to the total number of actual positive instances, emphasizing the model's completeness in identifying positives. Recall is vital in situations where missing positive instances can have severe consequences. Recall can be written as in Equation 2.7

$$Recall = \frac{True\ Positives}{True\ Positives\ +\ False\ Negatives} \tag{2.7}$$

where False Negatives represent the instances that should have been predicted into a class but were predicted as something else.

Mean Average Precision: Mean Average Precision (mAP) is an evaluation metric commonly used in object detection and information retrieval tasks. It combines precision and recall across multiple classes to provide a comprehensive measure of a model's performance. The process involves computing the Average Precision (AP) for each class and then taking the mean across all classes as shown in Equation 2.8

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i \tag{2.8}$$

The Average Precision for a single class is determined by calculating the area under the precision-recall curve. This curve represents the trade-off between precision and recall at different confidence score thresholds.

In summary, precision and recall offer insights into different aspects of a model's performance, addressing the trade-off between accuracy and completeness. mAP extends this evaluation to multiple classes, providing a holistic assessment of a model's effectiveness in handling diverse categories.

2.3.4 Deep Learning

Introduction to Deep Learning:

Deep learning extends the concept of neural networks to multiple layers, referred to as deep neural networks. The depth of these networks allows them to automatically learn hierarchical features from data. The training process involves forward and backward passes, where inputs are propagated through the network to produce predictions, and errors are backpropagated to adjust the weights. The backpropagation algorithm, coupled with optimization techniques like stochastic gradient descent, enables deep neural networks to iteratively improve their performance.

Mathematical Representation of Deep Learning: For a deep neural network with L layers, the output y is calculated through a series of transformations:

$$y = f_L(...f_2(f_1(x, W_1, b_1), W_2, b_2)..., W_{L-1}, b_{L-1}) \tag{2.9}$$

where:
- f_i is the activation function for layer i,
- W_i and b_i are the weight matrix and bias vector for layer i,
- x is the input.

Despite their power, deep neural networks face challenges such as overfitting, vanishing gradients, and the need for large amounts of labeled data. Regularization techniques, advanced activation functions, and transfer learning have been employed to address these challenges.

2.3.5 Backpropagation:

Backpropagation [RHW86], short for "backward propagation of errors", is a supervised learning algorithm used to train artificial neural networks by minimizing the error between predicted and actual outputs. The backpropagation algorithm is a key component in the optimization process of neural networks, enabling the adjustment of weights and biases to improve model performance.

The core idea of backpropagation is to iteratively update the model parameters (weights and biases) by computing the gradient of the loss function with respect to each parameter. This is achieved through the chain rule of calculus, allowing the algorithm to propagate the error backward through the network layers.

Mathematically, the backpropagation process involves two main steps: the forward pass and the backward pass.

- **Forward Pass:** During the forward pass, the input data is propagated through the neural network to generate predictions. For each layer, the weighted sum of inputs is passed through an activation function to produce the layer's output. The predicted output is then compared to the actual output using a loss function, such as cross-entropy loss.

- **Backward Pass:** The backward pass begins by calculating the gradient of the loss function with respect to the output of the network. This gradient is then propagated backward through the layers, computing the gradients of the loss function with respect to the parameters (weights and biases) of each layer. For a single-layer perceptron, the weight update rule during the backpropagation step can be expressed as in Equation 2.10:

$$\Delta w_{ij} = -\eta \frac{\delta E}{\delta w_{ij}} \tag{2.10}$$

where:
- $\Delta w_{ij}$is the change in weight connecting neuron i to neuron j,
- η is the learning rate,
- $\frac{\delta E}{\delta w_{ij}}$ is the gradient of the loss function with respect to the weight.

The weight update for multilayer networks involves a similar process, with the chain rule applied to calculate the gradient at each layer. The final weight updates are typically performed using optimization algorithms like Stochastic Gradient Descent (SGD) or its variants[Rud17].

2.3.6 Regularization

Regularization techniques in machine learning aim to prevent overfitting by introducing constraints on the model parameters.

Dropout: Dropout [SHK$^+$14] is a regularization technique used in neural networks to prevent overfitting by randomly deactivating a fraction of neurons during training. This process introduces robustness and helps the network generalize better to unseen data. Dropout is typically applied to fully connected layers, although it can be adapted for other layer types. Mathematically, dropout is implemented by randomly setting a fraction p of neurons to zero during each forward and backward pass. Dropout effectively creates an ensemble of neural networks by training different subnetworks for each mini-batch. This ensemble helps prevent co-adaptation of neurons and results in a more robust model. Dropout has been shown to be particularly effective in scenarios where overfitting is a concern, especially when dealing with limited training data.

2.4 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class of deep learning models designed for processing structured grid data, such as images. They have proven highly effective in computer vision tasks, demonstrating superior performance in tasks like image classification, object detection, and segmentation. CNNs are specifically engineered to recognize spatial hierarchies of patterns through the application of convolutional operations.

2.4.1 Key Components of CNNs:

Convolutional Neural Networks have many components, such as:

- **Convolutional Layers:** The fundamental building block of CNNs is the convolutional layer. It applies convolutional operations to the input data using learnable filters (kernels). Convolution involves sliding these filters over the input image and computing the dot product at each position. This operation captures local patterns and detects features like edges and textures.

- **Activation Functions:** Common activation functions, such as Rectified Linear Unit (ReLU), introduce non-linearity to the model. ReLU, for example, outputs the input for positive values and zero for negative values, enabling the model to learn complex relationships.

- **Pooling Layers:** Pooling layers, often implemented as max pooling or average pooling, downsample the spatial dimensions of the feature maps, reducing computational complexity and creating a degree of translation invariance.

- **Fully Connected Layers:** Fully connected layers follow the convolutional and pooling layers, transforming the high-level filtered information into a vector of probabilities for classification tasks.

Mathematical Representation: The convolution operation is represented in Equation 2.11:

$$F(i,j) = (X \star K)(i,j) = \sum_m \sum_n X(m,n).K(i-m, j-n) \tag{2.11}$$

where:
- X is the input image,
- K is the convolutional filter,
- F is the feature map,
- i and j represent the spatial dimensions of the feature map,
- m and n represent the filter dimensions.

2.4.2 Global Mean Pooling:

Global Mean Pooling (GMP) [LCY14] is a technique commonly used in neural network architectures, especially in convolutional neural networks (CNNs), for spatial dimension reduction. It provides a global context to the network by summarizing the spatial information of each feature map into a single value. Unlike traditional fully connected layers, which flatten the spatial dimensions, GMP retains the spatial structure while reducing dimensionality. This process is particularly effective in handling input data with varying spatial sizes. GMP is performed by computing the mean along each channel, represented in Equation 2.12

$$GMP(F) = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} F_{i,j} \tag{2.12}$$

where:
- $F \in \mathbb{R}^{H \times W \times C}$ is the Feature Map,
- H and W are the width and height dimensions respectively,
- C is the number of channels,
- $F_{i,j} \in \mathbb{R}^{C}$ represents the activation values at spatial position (i, j) across all channels.

The mean is calculated across all spatial positions, resulting in a global context representation for each channel.

Global Mean Pooling is often employed as a global feature extraction step before the final classification layer in CNNs. Its application can lead to more interpretable and robust representations while maintaining computational efficiency.

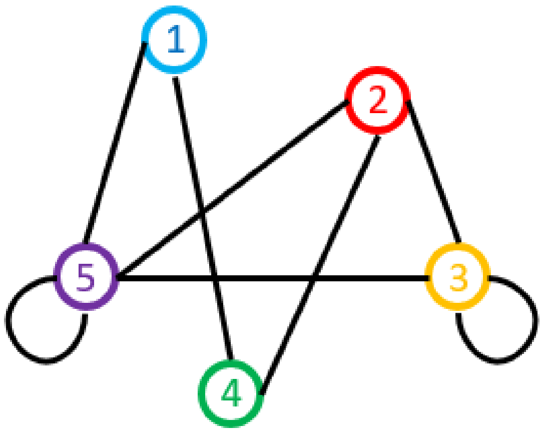
In summary, CNNs leverage convolutional operations, activation functions, pooling, and fully connected layers to automatically learn hierarchical features and relationships in input data, making them powerful tools for visual recognition tasks.

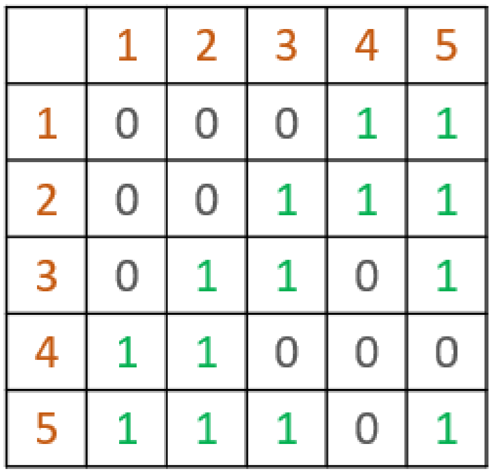
2.5 Graph Representation of Data

The representation of data as graphs introduces a versatile and expressive paradigm for modeling relationships and dependencies. In this section, we explore the conceptualization of representing data as graphs and the advantages this approach brings.

Structure of Graphs: A Graph $G(V, E)$ is a data structure containing a set of vertices (nodes) $i \in V$ and a set of edges $e_{ij} \in E$ connecting vertices i and j. These connections can be represented in an adjacency matrix. If the value of the matrix at row i and column j of the matrix is 0, it means there is no connection. If that value is 1 it means node i is connected to node j. Figure 2.1 shows a simple example of a Graph structure, and its corresponding adjacency matrix. However graphs can get more complex by adding node attributes, edge attributes, edge direction and so on.

Graphs for Action Recognition: In the context of action recognition, a graph-based representation involves defining nodes to represent objects or entities within a scene and edges to denote relationships or interactions between these entities. - Nodes: Nodes in





	1	2	3	4	5
1	0	0	0	1	1
2	0	0	1	1	1
3	0	1	1	0	1
4	1	1	0	0	0
5	1	1	1	0	1

Figure 2.1: Simple Example of Graph Structure and Adjacency Matrix

the graph correspond to the objects or entities present in the visual data. In the context of action recognition, these entities could be individuals, objects, body parts, or any relevant components contributing to the observed actions. Each node is associated with attributes that encapsulate information about the corresponding entity. These attributes may include visual features, spatial coordinates, or any relevant descriptors contributing to the understanding of the entity. - Edges: Edges in the graph represent the relationships or interactions between pairs of nodes. These relationships can encompass spatial connections, temporal dependencies, or any contextual associations that contribute to the overall understanding of the actions taking place. The edges may carry weights that signify the strength or significance of the relationships. For example, in the context of human actions, an edge weight could represent the degree of interaction or dependency between two body parts. The edges may also be associated with attributes representing the type of relationship between nodes.

An example of a simple scene graph representation is illustrated in Figure 2.2 where in this scenario we have a person in front of a bottle and chips, and touching a cup. The activity here could be "Drinking water".

Graph representations provide a natural way to encode complex relationships and interactions within structured data. Unlike traditional data representations, where entities are treated as independent and isolated, graphs allow us to model the interconnectedness between entities. This proves particularly valuable in domains such as action recognition,

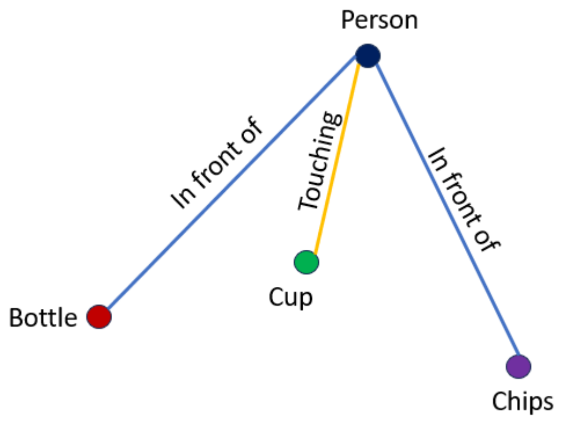


Figure 2.2: Example of Scene Graph Representation

where the understanding of relationships between objects or entities is crucial for a holistic interpretation of dynamic scenes.

2.6 Graph Neural Networks

While traditional neural networks operate on data organized in a sequential or grid-like fashion, graph structures introduce a more flexible and expressive representation. Graphs consist of nodes and edges, allowing for the modeling of intricate relationships between entities. In the context of neural networks, this entails considering not just the individual data points but also their connections and dependencies.

Fundamentals of GNNs: Graph Neural Networks (GNNs) extend the capabilities of traditional neural networks to handle graph-structured data. The key features of GNNs include:

- **Node Embeddings:** GNNs assign embeddings to nodes, capturing their features and contextual information. This allows nodes to represent entities in a graph, such as objects in an image or words in a document. These node features (embeddings) are the inputs to the GNN. every node i has its associated node features x_i.

- **Edge Attributes:** Edges can have features as well, such as the type of connection between the nodes. These features are represented as a_{ij}

- **Message Passing:** GNNs look for neighborhoods to understand why some nodes connect and others don't. The Neighborhood N_i of a node i is defined as the set of nodes j

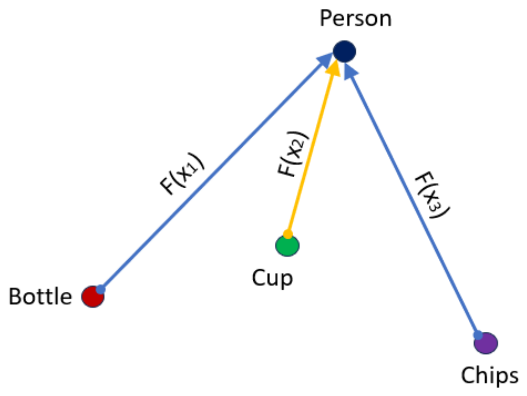


Figure 2.3: Example of Message Passing in a Neighborhood

connected to i by an edge. This is represented as $N_i = \{j \ : \ e_{ij} \ \in \ E\}$ [Ana22]. In a Graph Neural Network (GNN) layer, Message Passing involves taking the features of neighboring nodes, transforming them, and then transmitting these transformed features back to the source node. This iterative process is performed simultaneously for all nodes in the graph, ensuring that every neighborhood is thoroughly explored by the conclusion of this step. For the sake of simplicity, let's take the Graph in Figure 2.2 and consider just the node features. Message Passing for the neighborhood of node "Person" would look something similar to what is represented in Figure 2.3, where F could be any Function, such as a simple Neural Network or affine Transform. For the sake of simplicity, let us consider F to be $F(x_j) = W_j.x_j$

- **Aggregation:** The transformed messages have been passed from the neighbors of the node "Person" to it, they have to be aggregated (combined) in some way. There are many aggregation functions that can be used. These functions include:

$$Sum = \sum_{j \in N_i} W_j.x_j \tag{2.13}$$

$$Mean = \frac{Sum}{|N_i|} = \frac{\sum_{j \in N_i}}{|N_i|} W_j.x_j \tag{2.14}$$

$$Min = min_{j \in N_i}(W_j.x_j) \tag{2.15}$$

$$Max = max_{j \in N_i}(W_j.x_j) \tag{2.16}$$

Suppose the chosen Aggregation Function is denoted as Agg, the final Message Passing equation is denoted as:

$$\bar{m}_i = Agg(W_j.x_j : j \in N_i) \tag{2.17}$$

Using the aggregated messages, the GNN updates the source's features to include them. This is done using either a simple addition or a concatenation. This is represented as:

$$h_i = \sigma(K(H(x_i), \bar{m}_i)) \tag{2.18}$$

where :
- h_i are the new node features for node i,
- σ is the activation function (ReLU, Tanh ...),
- H is a simple Neural Network (MLP) or Affine Transformation,
- K is another MultiLayer Perceptron (MLP) to project the added vectors into another dimension.

Putting them together, and considering the aggregation function to be the Mean and F and H to be simple Feed-Forward layers, the final equation can be written as:

$$h_i = \sigma(W_1.h_i + \frac{\sum_{j \in N_i}}{|N_i|} W_2.h_j) \tag{2.19}$$

- **Adding Edge Features to the equation:** Suppose edges have features a_{ij}, the GNN should find a way to pass them too. This is done by factoring the embeddings of both nodes connected by the edge to update them at a particular layer l. This is represented as:

$$a_{ij}^l = T(h_i^l, h_j^l, a_{i,j}^{l-1}) \tag{2.20}$$

where T is a simple Neural Network that takes in the embeddings from connected nodes i and j as well as the edge features from the previous layer.

- **Stacking GNN Layers:** A Network is built by adding multiple layers. Let us consider a Network made out of 2 GNN layers. The input to the first GNN layers are the node features (embeddings) represented as X which are made of $x_{i:1->N}$. The output is the modified node embeddings H^1 which are made of $h^1_{i:1->N}$. This is the input to the next GNN layer. The final output is the modified node embeddings H^2 which is made of $h^2_{i:1->N}$. This is represented in Figure 2.4

Strengths of GNNs: The strength of GNNs lies in their inherent capacity to model relationships and dependencies within structured data. By considering the interconnectedness of entities in a graph, GNNs excel at tasks where understanding the context and interactions between elements is paramount. This is particularly valuable in action classification and anticipation, where actions unfold in dynamic scenes with intricate relationships between entities.

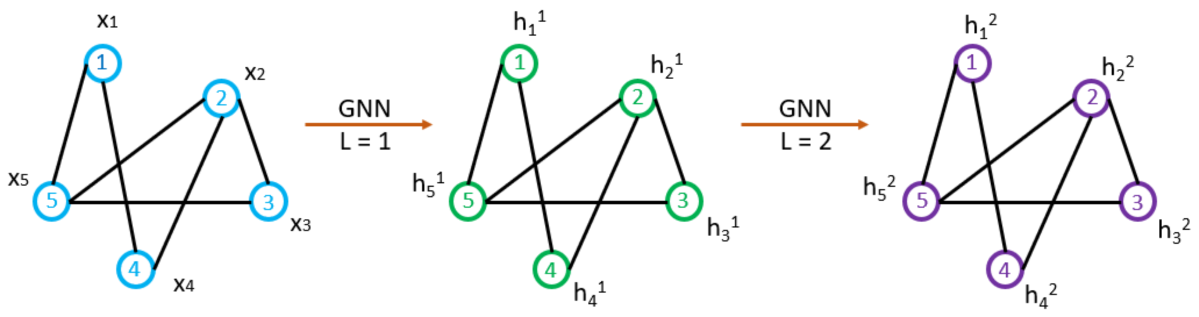


Figure 2.4: Stacked GNN layers

2.7 Graph Convolutional Networks

Graph Convolutional Networks (GCNs) are a class of Graph Neural Networks (GNNs) that have proven effective in modeling relationships and dependencies within graph-structured data. The essence of GCNs lies in their ability to perform convolutions on graphs, allowing them to capture complex relationships between nodes. Let's delve into the specifics of GCNs and understand how they operate on graph-structured data.

Propagation of Information: In GCNs, information is propagated through the graph by aggregating and combining information from neighboring nodes. The aggregation operation involves taking into account the features of connected nodes, capturing the dependencies between them.

Mathematical Representation: The output of a node in a GCN is calculated by aggregating the features of its neighbors. GCN has the following layer-wise propagation rule [KW17]:

$$H^{l+1} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}) \tag{2.21}$$

where:
- $\tilde{A} = A + I_N$ is the adjacency matrix of the graph G with added self connections and I_N is the identity matrix,
- $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ is a degree matrix, as a form of renormalisation to avoid numerical instabilities and exploding/vanishing gradients,
- $W^{(l)}$ is a layer-specific trainable weight matrix,
- σ is an activation function such as ReLU ,
- $H^{(l)}$ is the matrix of activations in the l^{th} layer; $H(0) = X$

Modeling Dependencies:
- Hierarchical Learning: GCNs enable hierarchical learning by capturing local dependencies

in the early layers and gradually incorporating information from distant nodes in deeper layers. This facilitates the modeling of complex relationships within the graph.
- Graph Convolution Operation: The graph convolution operation is fundamental to GCNs, allowing them to adapt their weights based on the local structure of the graph. This adaptability is crucial for capturing the nuances of dependencies in various applications, including action recognition.

2.8 Graph Attention Networks

Graph Attention Networks (GATs) [VCC+18] enhance the expressive power of GNNs by introducing attention mechanisms. Attention mechanisms enable nodes to selectively focus on relevant information from their neighbors, allowing for more flexible and adaptive learning. Let's explore the specifics of GATs and how attention mechanisms contribute to their effectiveness.

Attention Mechanisms: GATs assign importance weights to the edges connecting nodes, allowing each node to weigh the information from its neighbors differently. This attention mechanism is based on learned parameters, enabling the model to adaptively determine the importance of each neighbor.

Mathematical Representation: Edge weights are generated as follows:

$$\alpha_{ij} = Softmax(LeakyReLU(W_a^T.[Wh_i^l \oplus Wh_j^l])) \tag{2.22}$$

where:
- α_{ij} are the edge weights,
- $W_a^T \in \mathbb{R}^{2d'}$ and $W \subseteq \mathbb{R}^{d' \times d}$ are learned parameters and d is the embedding dimension,
- $\oplus$ is the vector concatenation operation.

The combined Message Aggregation and Update steps are a weighted sum over all the neighbours and the node itself:

$$h_i = \sum_{j \in N_i \cup \{i\}} \alpha_{ij}.Wh_j^l \tag{2.23}$$

Advantages of GATs:
- Selective Information Fusion: By allowing nodes to selectively focus on relevant neighbors, GATs enhance the model's ability to capture fine-grained dependencies. Nodes can adaptively attend to informative neighbors, leading to a more nuanced understanding of the graph.

- Improved Generalization: Attention mechanisms in GATs contribute to improved generalization by reducing the reliance on fixed neighborhood sizes. Nodes can dynamically adjust their attention to different neighbors based on the context, enhancing the model's flexibility.

2.9 Applicability of GNNs to Action Classification and Anticipation

Graph Neural Networks (GNNs) offer a powerful and versatile framework for tackling the challenges of action classification and anticipation. In this section, we delve into how GNNs can be effectively applied to these tasks, highlighting their unique strengths in modeling temporal dependencies and capturing contextual information within video sequences.

Modeling Temporal Dependencies with GNNs:
- Sequential Information: Actions unfold over time, and capturing the temporal dependencies between different frames is crucial for understanding the progression of an action. GNNs excel in contextualizing temporal information, allowing them to consider the dynamic evolution of actions.

- Dynamic Graph Adaptation: GNNs can adapt their graph structures dynamically to reflect the changing relationships between entities (nodes) across frames. This adaptability enables the model to learn and encode temporal dependencies inherent in video sequences.

- Temporal Graph Convolution: GNNs, particularly Graph Convolutional Networks (GCNs), can perform graph convolutions not only in the spatial domain but also in the temporal domain. This capability enables the modeling of complex temporal relationships between nodes, facilitating more accurate action recognition and anticipation.

- Hierarchical Learning: GNNs support hierarchical learning, allowing them to capture both short-term and long-term dependencies. This is essential for recognizing actions that may involve varying time scales and intricate temporal patterns.

Graph-Structured Data in Video Sequences:

- Nodes Representing Entities: In the context of action recognition, video frames can be represented as graphs where nodes correspond to objects, body parts, or entities present in each frame. GNNs enable the modeling of relationships between these nodes, capturing the spatial and temporal interactions.

- Edges Representing Relationships: Edges in the graph signify the spatial or temporal relationships between nodes. For example, edges may represent the spatial connections between body parts or the temporal dependencies between consecutive frames.

- Spatial Dependencies: GNNs inherently capture spatial relationships by considering the interactions between nodes within a single frame. This is particularly valuable in recognizing actions that involve complex spatial configurations, such as interactions between multiple objects or body parts.

- Temporal Dependencies: GNNs extend their spatial modeling capabilities to incorporate temporal dependencies, allowing them to recognize actions that evolve over time. The dynamic adaptation of graph structures enables the model to discern how entities interact across different frames.

Graph Attention Mechanisms:
- Selective Information Integration: Graph Attention Networks (GATs) within the GNN framework enhance the modeling of relationships by allowing nodes to selectively attend to relevant information. This attention mechanism is advantageous for discerning critical spatial and temporal cues, contributing to more accurate action recognition and anticipation.

- Adaptive Learning: GATs enable nodes to dynamically adjust their focus based on the context, leading to adaptive learning that is particularly beneficial in scenarios where certain entities or relationships may be more informative for specific actions.

In summary, GNNs, with their ability to model temporal dependencies and handle graph-structured data in video sequences, present a promising avenue for advancing action classification and anticipation. By leveraging the inherent relationships between entities and adapting dynamically to evolving contexts, GNNs offer a robust framework for understanding and interpreting dynamic actions in complex visual scenarios.

Chapter 3

Related Work

This chapter reviews existing literature relevant to the domains of Object Detection, Scene Graph Generation, Action Recognition, and Anticipation.

Within Object Detection, the discussion encompasses traditional methods, the evolution towards Convolutional Neural Networks (CNNs), and advancements in both region-based and single-stage detectors. The exploration of Scene Graph Generation includes an examination of its components, associated datasets, and diverse generation approaches. The section on Action Recognition covers techniques utilizing CNNs, Temporal Convolutional Networks (TCNs), Transfer Learning, Pretrained Models, and Attention Mechanisms. Additionally, the chapter investigates the application of Scene Graphs in the context of Action Recognition and provides insights into Graph Neural Networks, including specific architectures such as GraphSAGE, GraphConv, TransformerConv, GENConv, and GATv2Conv.

The comprehensive overview of related work sets the stage for the subsequent chapters, positioning the research within the broader landscape of current methodologies and advancements in the field.

3.1 Object Detection

Object detection is a fundamental computer vision task that involves locating objects and identifying them in an image or a video. It plays a crucial role in various applications, such as autonomous driving, surveillance, medical imaging, and augmented reality.

3.1.1 Traditional Methods:

Early object detection methods primarily relied on handcrafted features and traditional machine learning algorithms. Techniques like Histogram of Oriented Gradients (HOG) [DT05] and Haar-like features were widely used in combination with classifiers such as Support Vector Machines (SVM) and Cascade Classifiers, like the Viola Jones object detector [VJ01]. While effective for certain applications, these methods struggled with complex scenes, variations in lighting, and the presence of occlusions.

3.1.2 Convolutional Neural Networks:

The introduction of deep learning and Convolutional Neural Networks (CNNs) revolutionized object detection. The seminal work of AlexNet in 2012 [KSH12] and subsequent models, like VGG [SZ14b], GoogLeNet [SLJ$^+$14], and ResNet [HZRS15], greatly improved object detection accuracy. AlexNet is a convolutional neural network that is eight layers deep: five convolutional layers and three fully-connected layers. The architecture is visualized in Figure 3.1. There is a pretrained version of the network that can be loaded which is trained on more than a million images from the ImageNet database[1]. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. Some of the special features of the AlexNet are ReLU non linearity, multiple GPUs and overlapping Pooling [Wei19]. As a result, CNNs effectively learned rich feature representations for a wide range of images, making them well-suited for object detection tasks.

3.1.3 Region-Based Object Detection:

Region-based approaches, including Region Proposal Networks (RPN), Fast R-CNN [Gir15], and Faster R-CNN [RHGS15], emerged as significant advancements in object detection. **R-CNN** uses a region proposal method to generate about 2000 regions of interest (ROI). Different CNN networks are used for each region, followed by a fully connected layers to classify the object and refine the boundary box. However R-CNN is slow in training and the cost of repeating feature extractions 2000 times for different ROIs is expensive. **Fast R-CNN** solves this issue by using a feature extractor to extract features for the whole image first (single CNN for the regions). A region proposal feature map is collected, then max pooling is applied which reduces the dimensions of the feature maps. They are then transformed into a one-dimensional vector and fed to fully connected layers for classification and localization. Fast R-CNN is 25 times faster than R-CNN, but generating 2000 ROIs is still

[1]`https://www.image-net.org/`

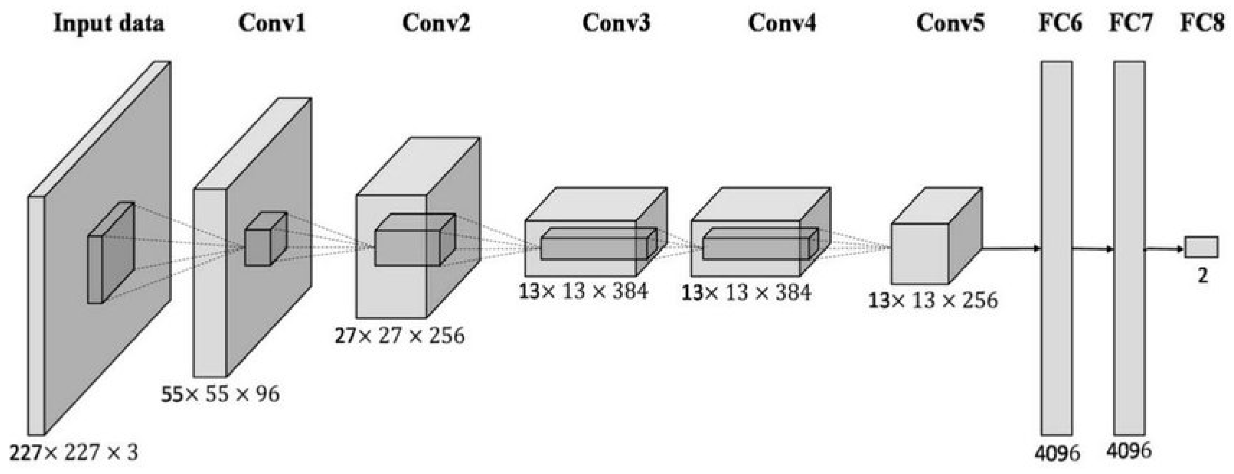


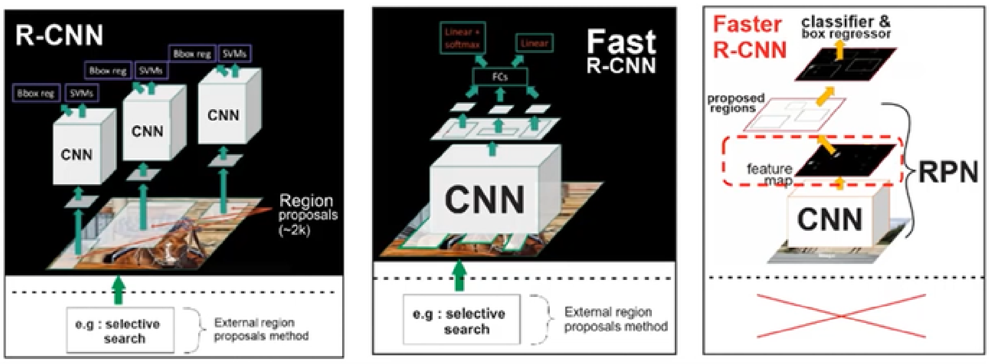
Figure 3.1: AlexNet Architecture from [TCAR20]

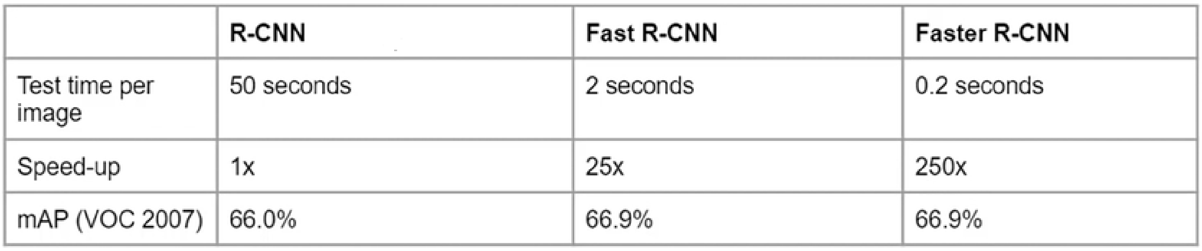
computationally expensive. **Faster R-CNN** replaces the region proposal method by an internal deep network and the ROIs are derived from the feature maps instead. This makes it around 10 times faster than Fast R-CNN, and 250 times faster than R-CNN, but still not fast enough for real time detection [Hui18] [Bul21]. The differences in architecture and performance between R-CNN, Fast R-CNN and Faster R-CNN are illustrated in Figure 3.2.

3.1.4 Single-Stage Detectors:

A single-stage detection framework refers to an integrated pipeline, like unified architectures, which directly predicts class probabilities and bounding box offsets from complete images in a single forward pass through a CNN network. This setup does not involve region proposal generation or post-classification. Single-stage Detectors, such as SSD (Single Shot Detector) [LAE$^+$15] and YOLO (You Only Look Once) [RDGF15], were designed for real-time object detection. They use a single feedforward neural network to simultaneously predict object bounding boxes and class probabilities for multiple objects in an image. These models excel in applications where speed is critical.

SSD have two components rather than one: a backbone model and an SSD head. The backbone model is a pretrained image classification network like ResNet or VGG16 which acts as a feature extractor, without their final classification layer. The SSD head is comprised of convolutional layers responsible for producing feature maps which in turn produces the bounding box and class predictions. [Ray22]. SSD only requires an input image and ground truth boxes during training. It evaluates default boxes of different aspect ratios at





	R-CNN	Fast R-CNN	Faster R-CNN
Test time per image	50 seconds	2 seconds	0.2 seconds
Speed-up	1x	25x	250x
mAP (VOC 2007)	66.0%	66.9%	66.9%

Figure 3.2: Architecture and Performance differences between R-CNN, Fast R-CNN and Faster R-CNN from [AGMM+21]

each location in several feature maps with varying scales. For each default box, it predicts shape offsets and confidences for all object categories. During training, it matches these default boxes to ground truth boxes, and the model loss is a weighted sum of localization loss and confidence loss.

YOLO treats detection as a regression problem, dividing the image into an S×S grid. For each grid cell, YOLO predicts B bounding boxes, their confidences, and C class probabilities. These predictions are encoded as a tensor. If the center of an object falls into a grid cell, that cell is responsible for detecting the object[AG19].

The difference between the SSD and YOLO architectures is illustrated in Figure 3.3.

3.2 Scene Graph Generation

Scene graph generation is a critical computer vision task that focuses on extracting structured representations of visual scenes. These representations, often referred to as scene graphs, encode the relationships between objects in an image, providing a deeper understanding of the visual content. The concept of scene graphs originated from earlier work in computer vision, where researchers focused on object detection and image captioning.

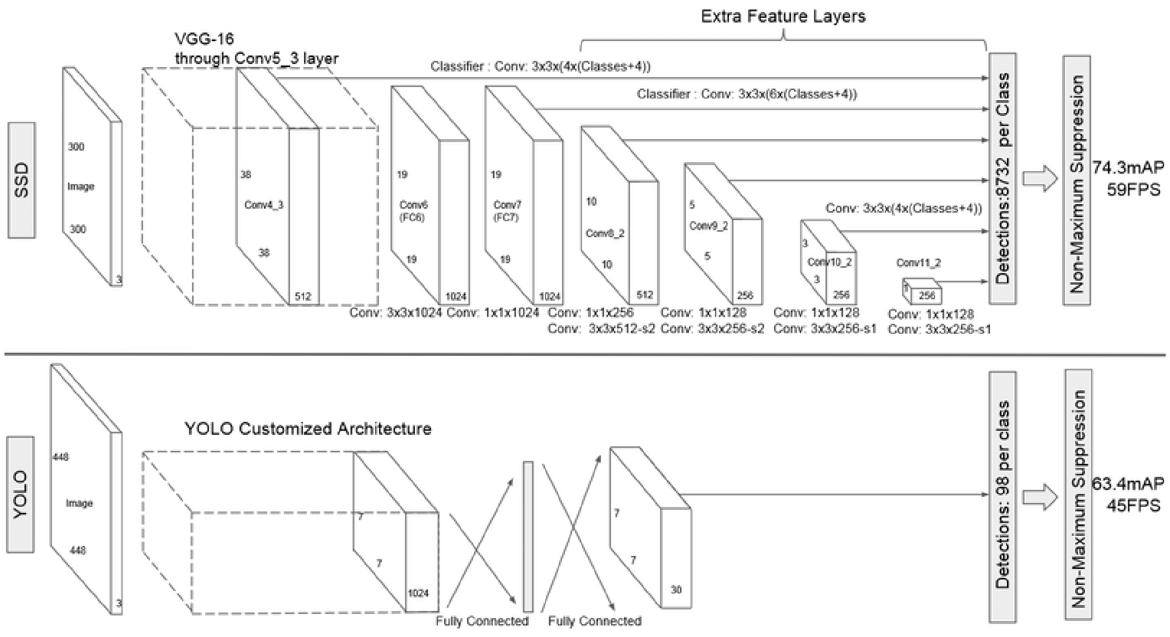

SSD
YOLO
VGG-16 through Conv5_3 layer
Extra Feature Layers
Classifier : Conv: 3x3x(4x(Classes+4))
Classifier : Conv: 3x3x(6x(Classes+4))
Conv: 3x3x(4x(Classes+4))
Detections:8732 per Class
Non-Maximum Suppression
74.3mAP 59FPS
YOLO Customized Architecture
Fully Connected
Fully Connected
Detections: 98 per class
63.4mAP 45FPS

Figure 3.3: Architectures of SSD and YOLO from [AG19]

Scene graphs emerged as a means to combine both object recognition and relationship modeling, enhancing the overall comprehension of visual content.

3.2.1 Scene Graph Components:

A typical scene graph comprises three key components: nodes representing objects, edges denoting relationships between objects, and attributes associated with both objects and relationships. Objects in the scene graph are labeled entities, relationships are labeled predicates, and attributes describe the properties of entities and relationships.

3.2.2 Scene Graph Datasets:

To train and evaluate scene graph generation models, researchers created datasets that contain annotated scene graphs.

- MS COCO: It does not contain any usable semantic information needed to make a machine learn how to generate scene graphs, but is the base dataset that most other datasets were built upon.

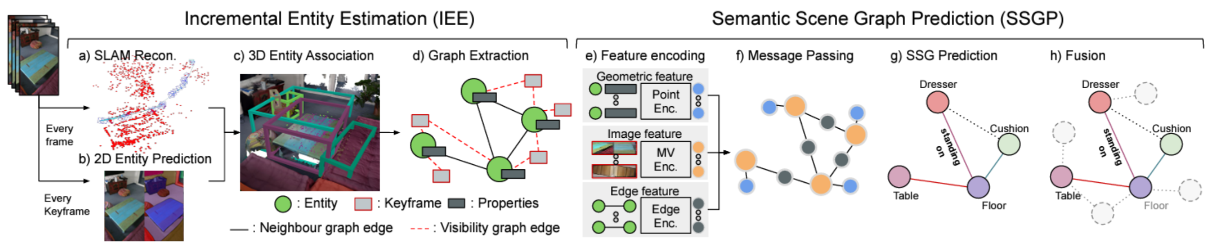


Figure 3.4: Semantic Scene Graph Generation model from [YCZ+20]

- VRD (Visual Relationship Detection) [LKBF16]: It is one of the first datasets developed during scene graph generation research. The VRD-Net leverages convolutional neural networks and recurrent neural networks to improve relationship detection accuracy.

- VG (Visual Genome)[2] [KZG+16]: This dataset introduces additional ground truths, including object relations and examples for visual question answering. VG leverages WordNet[Mil95], to identify objects and relations. However, due to the automatic generation of semantic data from crowd-sourced input, the dataset is inherently noisy, necessitating substantial preprocessing before utilization.

- HOMAGE (Home Action Genome[3] [RCJ+21]: It is a dataset for videos and it has played a significant role in advancing the field by providing large-scale, diverse data for training and testing. HOMAGE even created a challenge for the Scene Generation Task.

3.2.3 Scene Graph Generation Approaches:

Bottom up two-stage processes: Early scene graph generation models used a bottom up two-stage process, first identifying objects and relationships, and then constructing the scene graph. It's a process in which entities are grouped into triplets (subject, relation, object) which are connected to form the entire scene graph. The essence of the task is to detect the visual relationships. An interesting method is the one proposed in the paper "A Bottom-up Framework for Construction of Structured Semantic 3D Scene Graph" [YCZ+20] where they adopt visual perception to capture the semantic information and inference from scene priors to calculate the optimal parse graph. Afterwards, an improved probabilistic grammar model is used to represent the scene priors. Figure 3.4 illustrates their model. Given a sequence of RGB images, each frame is employed to reconstruct a sparse point map (a), while key frames are specifically used for estimating the 2D entities

[2]https://homes.cs.washington.edu/~ranjay/visualgenome/index.html
[3]https://homeactiongenome.org/

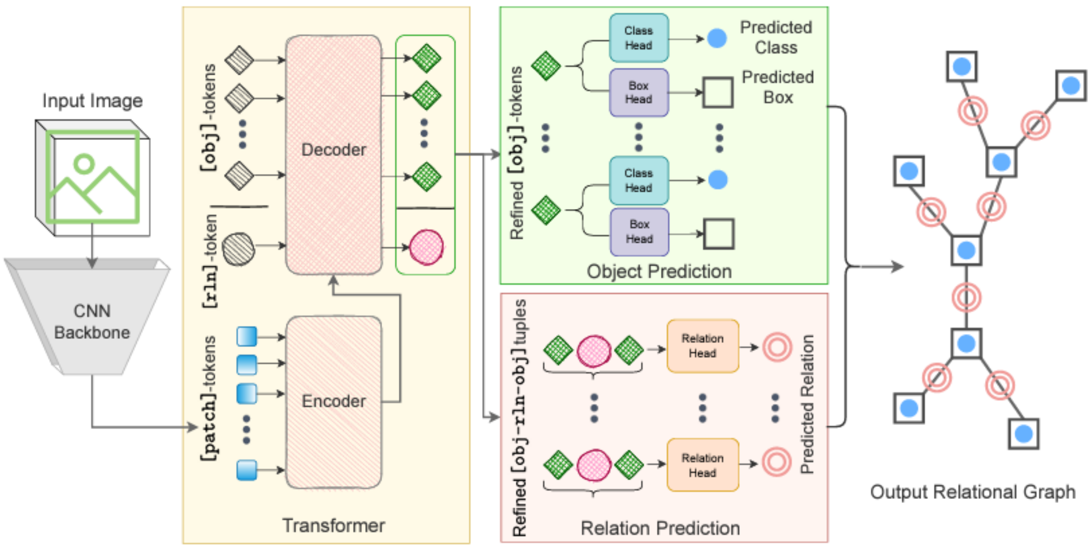


Figure 3.5: Relationformer architecture from[SKW+22]

(b). The results from (a) and (b) are then associated and merged to form a unified 3D map (c). Concurrently, graph properties are asynchronously extracted from the entity map (d) to estimate the Semi-Sparse Graph (SSG). Their network computes geometric, multi-view, and edge features (e). These features undergo mutual propagation through message passing (f), facilitating the prediction of an SSG (g). Subsequently, periodic SSGs are amalgamated to generate a comprehensive global 3D SSG (h).

One stage Approaches: In the traditional bottom up approach, 2 stages are required, one for detection and one for relation prediction. One stage approaches were introduced that jointly predict the objects and their relations. Relationformer [SKW+22] is a one-stage transformer-based framework that does that. It leverages direct set-based object prediction and incorporates the interaction among the objects to learn an object-relation representation jointly. The architecture of the model is illustrated in Figure 3.5

End-to-End Approaches: Recent advancements in scene graph generation involve end-to-end models such as SGTR [LZH22]. This SGG method formulates the task as a bi-partite graph construction problem. The problem is solved by the developing a transformer-based end-to-end framework that first generates the entity and predicate proposal set, followed by inferring directed edges to form the relation triplets. Their architecture is illustrated in Figure 3.6

Graph Neural Networks (GNNs): The application of graph neural networks to scene graph generation has gained popularity. GNNs enable direct modeling of object-object relationships and can leverage contextual information to improve predictions. Methods like Graph R-CNN [YLL+18] and Relational Graph Convolutional Networks (R-GCN)

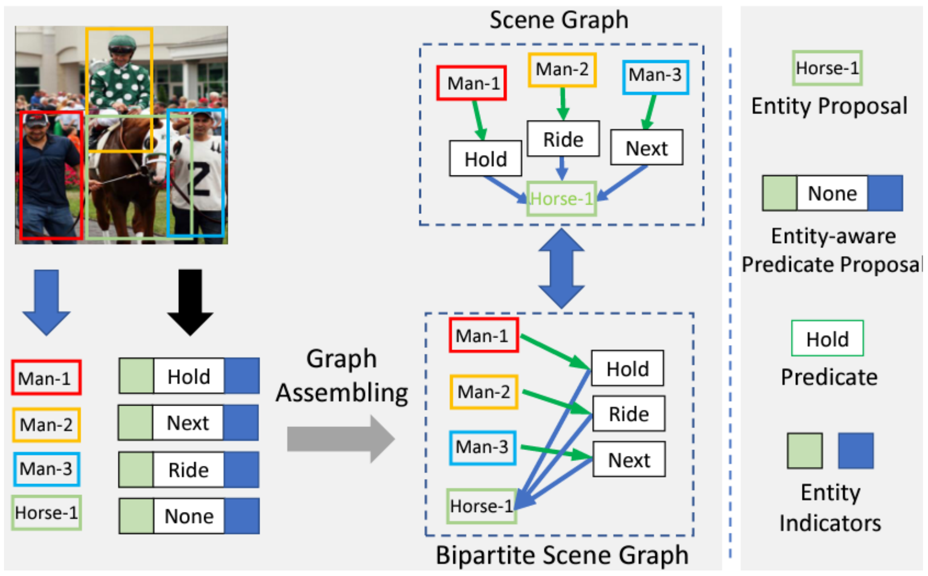


Figure 3.6: SGTR model Architecture from [LZH22]

[SKB+17] have showcased the potential of GNNs in this context. The Graph R-CNN model contains a Relation Proposal Network (RePN) that efficiently deals with the quadratic number of potential relations between objects in an image. The attentional Graph Convolutional Network (aGCN) proposed alongside the RePN effectively captures contextual information between objects and relations. Given an image, their model first uses RPN to propose object regions, and then prunes the connections between object regions through their relation proposal network (RePN). To incorporate contextual information from neighboring nodes in the graph, the Attentional Graph Convolutional Network (GCN) is applied. Finally, the scene graph is obtained on the right side. Their architecture is illustrated in Figure 3.7

3.3 Action Recognition

Action recognition is a crucial task in computer vision and artificial intelligence, enabling machines to understand and interpret human activities and gestures from visual data. Action recognition has found applications in various domains, including video surveillance, healthcare (patient monitoring and rehabilitation), sports analytics, autonomous robotics, and human-computer interaction. The ability to understand human actions from video data is essential for creating intelligent and interactive systems.

Early Approaches to Action Recognition:

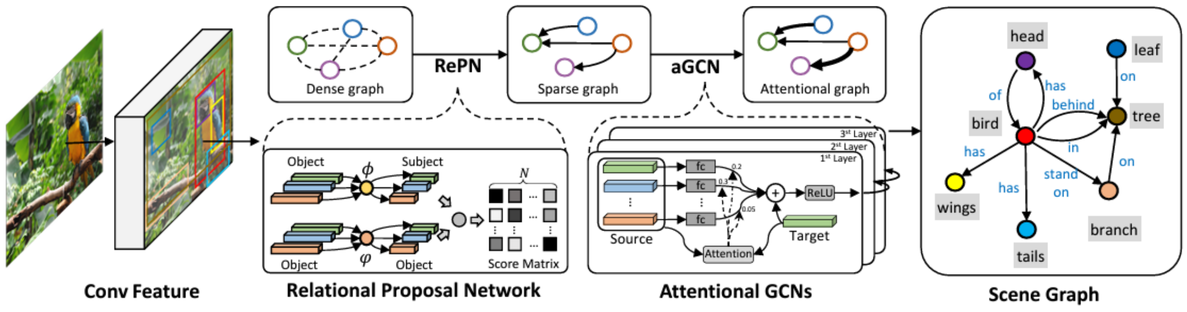


Figure 3.7: Graph R-CNN Architecture from [YLL+18]

Early action recognition methods primarily focused on handcrafted features and traditional machine learning algorithms. Techniques like Histogram of Oriented Gradients (HOG) [KS18], Local Binary Patterns (LBP) [MS09], and Space-Time Interest Points (STIP) [BGX09] were used to capture motion and appearance information in video frames. These methods lacked robustness to variations in lighting, viewpoint, and background clutter.

3.3.1 Convolutional Neural Networks (CNNs):

The introduction of deep learning, particularly Convolutional Neural Networks (CNNs), revolutionized action recognition. CNNs can automatically learn discriminative features from raw image data and video frames, leading to significant improvements in recognition accuracy. Notable models like Two-Stream CNNs and 3D CNNs have been developed to leverage spatiotemporal information effectively.

Two-Stream CNNs: [SZ14a] Two-Stream CNNs consist of two parallel networks: one processes spatial information (RGB frames), and the other processes temporal information (optical flow or stacked optical flow frames). A video can be naturally divided into two elements: the spatial component, represented by the appearance of individual frames, contains information about the scenes and objects depicted. The temporal component, expressed through motion across frames, communicates the movement of the observer (camera) and the objects within the video. Their architecture is divided into 2 streams and each stream is implemented using a deep ConvNet. This approach allows the network to capture both appearance and motion cues, enhancing action recognition accuracy. The architecture is illustrated in Figure 3.8

3D Convolutional Neural Networks (3D CNNs):[JXYY13] 3D CNNs extend the concept of 2D CNNs to video data, treating video frames as 3D volumes. They have been effective in modeling both spatial and temporal features, making them suitable for action

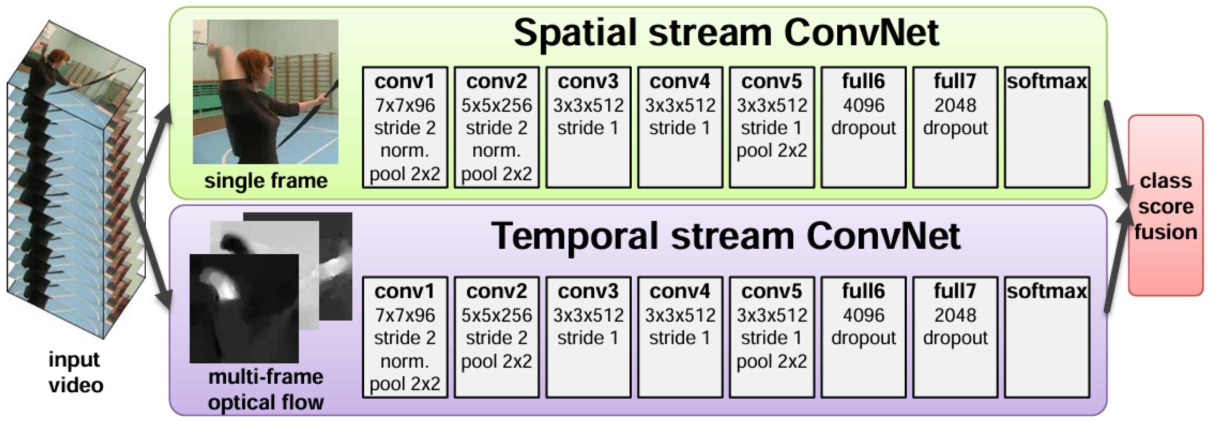


Figure 3.8: Two-Stream CNN Architecture from [SZ14a]

recognition in videos. C3D (Convolutional 3D) [TBF+15] and I3D (Inflated 3D) [CZ18] networks are prominent examples of 3D CNN architectures. The idea behind 3D CNNs lies in the fact that applying 2D convolution on wither an image, or a video volume results in an image; While applying 3D convolution on a video volume results in another volume, preserving temporal information of the input signal.

3.3.2 Temporal Convolutional Networks (TCNs):

Temporal Convolutional Networks (TCNs) are a class of models designed to capture long-range temporal dependencies in video sequences. TCNs have gained attention for their effectiveness in action recognition tasks, especially when dealing with long and complex actions. [KR17] The properties of a TCN are similar to those of a modern spatial Convolutional Neural Network(CNN) for recognition tasks. The network is built from stacked units of 1D convolution followed by a non-linear activation function. The 1D convolution is across the temporal domain.

3.3.3 Transfer Learning and Pretrained Models:

Transfer learning has become a standard practice in action recognition. Researchers often fine-tune models pretrained on large-scale video datasets like Kinetics or Sports-1M for specific action recognition tasks. This approach significantly reduces the need for extensive labeled data and improves recognition accuracy. In the paper [ZHL+22], they proposed a self-distillation guided transfer learning framework. At the beginning of each epoch, they

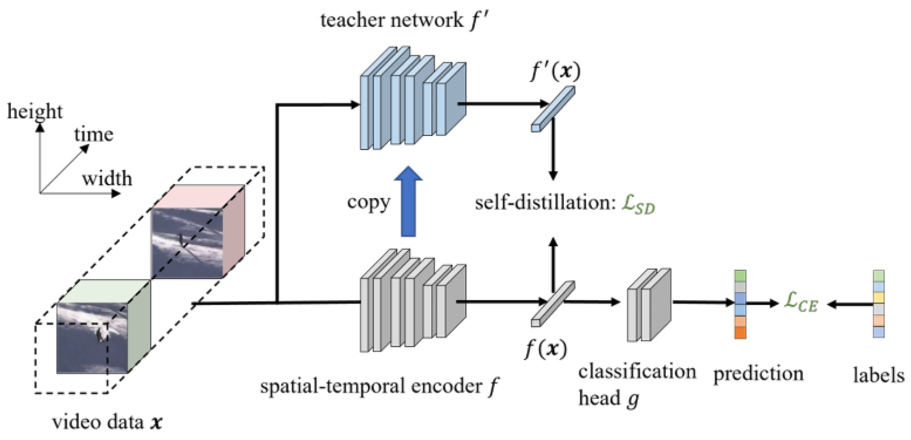


Figure 3.9: Transfer Learning Architecture from [ZHL+22]

make a copy of the spatial-temporal encoder from the last epoch and designate it as the teacher network. The regularization in transfer learning involves considering the Euclidean distance between representations generated by both encoders. Additionally, conventional cross-entropy loss is applied to the head network for action recognition. Their framework is illustrated in Figure 3.9

3.3.4 Attention Mechanisms:

Attention mechanisms have been integrated into action recognition models to focus on relevant spatiotemporal regions within video frames. These mechanisms enhance the ability to recognize actions in cluttered or complex scenes. In the paper [BTW+20] they propose an attention mechanism: In the first stage of this attention model, there's a view-specific attention process. Two LSTM encoders individually perform recognition for each view. The attention scores from these two views are then collected and input into the second stage. In the second stage, a Mutual-Aid RNN is employed to establish a collaborative attention mechanism. The Mutual-Aid block works frame-by-frame, facilitating cooperation between the two views. An attention module is applied to each view, and the results from multiple views are fused together.

3.4 Scene Graphs for Action Recognition

Using scene graphs for action recognition is an innovative approach that leverages structured representations of visual scenes to enhance the understanding of complex actions in videos. Several recent studies have explored this approach, and their contributions are

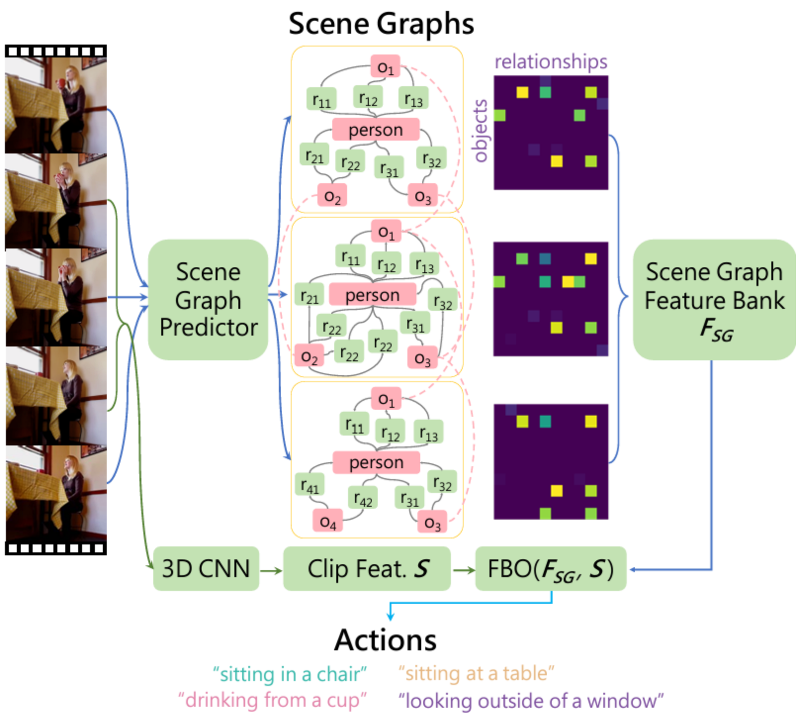


Figure 3.10: SGFB Model Architecture from [JKFFN19]

noteworthy.

- In the paper **"Action Genome: Actions as Composition of Spatio-temporal Scene Graphs"** [JKFFN19], they worked on the Action Genome dataset, created spatio-temporal scene graphs and proposed a method, named Scene Graph Feature Banks (SGFB), to incorporate spatio-temporal scene graphs into action recognition. They created a graph for each frame of the video and used the sequence of scene graphs as feature banks for the action recognition task. The feature vector is obtained by aggregating the information across all the scene graphs into a feature bank. The final representation is merged with 3D CNN features and used to predict action labels.
They also created the SGFB Oracle method, which assumes the availability of a perfect scene graph prediction method. The spatio temporal scene graph feature bank therefore, directly encodes a feature vector from ground truth objects and visual relationships for the annotated frames of the dataset.
With their SGFB model they reached an average precision of 44.3% on the validation set, and with their SGFB Oracle model they reached a 60.3% average precision.

- In the paper **"Explainable Action Prediction through Self-Supervision on Scene Graphs"** [KMOK23], they designed an encoder network that can learn embeddings from a sequence of scene graphs in a self-supervised manner on the "ROad event Awareness

Dataset (ROAD) for autonomous driving" [SAM+23]. Their encoder architecture allows for spatial and temporal modelling of scene graphs. It incorporates key spatial elements, including graph convolution layers, graph attention pooling layers, and graph readout layers. The model utilizes a Multi-Relational Graph Convolution (MR-GCN) approach, processing a sequence through multiple MR-GCN layers to acquire a K-hop spatial representation for each node concerning its neighbors. Drawing inspiration from graph isomorphism networks, this spatial representation is designed to capture relational information.
For hierarchical pooling, the model employs Self-Attention Graph Pooling (SAGPool), leveraging graph features, topology, and self-attention on the generated node embeddings to extract the most beneficial information for the learning task. A Global Readout operation is performed, involving the addition of each set of node embeddings to produce a full graph embedding of fixed dimensions. This output serves as a sequence of embeddings for each scene graph in the input.
The encoder's temporal attention component, LSTM-attn, plays a pivotal role in converting the sequence of graph embeddings into a single spatiotemporal embedding, often referred to as a context vector. Inspired by previous work, the model employs an attention mechanism to dynamically focus on embeddings in the sequence that are most relevant to the overall scene context. This involves computing weights for each embedding using a feed-forward layer on the LSTM output and final hidden state. The resulting spatiotemporal embedding, or context vector, is a weighted combination of hidden states over the entire sequence, serving as a crucial representation for downstream tasks such as classification or prediction.

- In the article **"Scene Graph Generation, Compression, and Classification on Action Genome Dataset"** [Li22], they used the Home Action Genome dataset to generate scene graphs and classify the action it represents. In their approach, they simplified the problem by having only 2 classes "using the phone" and "not using the phone".
First they divided the Scene Graph Generation task into 2 parts: node prediction and edge prediction.
For the node prediction they used an Object Detection Benchmark adapted from Mask R-CNN [MG18] to detect the object labels and their bounding box.
For the edge prediction they used a Scene Graph Benchmark [TZW+19] [Tan20] [TNH+20] model to predict the relationships between any two nodes as the edges. The combination of predicted nodes and edges form the complete scene graph.
Next they perform Graph Compression to create a smaller graph representation. To do this they apply the Multi-kernel Inductive Attention Graph Autoencoder (MIAGAE) [GPLI21] which has an encoder and a decoder. The encoder E learns to prune graph nodes and edges to create a compressed graph, then the decoder D learns to reconstruct the original graph by adding new nodes and edges. The encoder architecture includes Multi-kernel Inductive Graph convolution layers and Similarity Attention Graph Pooling layers. The decoder architecture includes inductive Un-pooling layers.

Finally they use a Graph Convolutional Network (GCN) consisting of 2 graph convolutional layers and 4 layers of multilayer perceptron (MLP) with batch normalization to train the model and output the final classification result. With this approach, the highest train accuracy reached was 0.60 and the highest test accuracy reached was 0.55.

3.5 Graph Neural Networks

Graph Neural Networks (GNNs) have garnered substantial attention in the field of machine learning, offering a powerful paradigm for learning and representation on graph-structured data. GNNs build upon the concept of neural networks but are tailored to handle graph data. Theoretical foundations of GNNs can be traced back to early works on graph-based semi-supervised learning. The propagation of information across nodes, often characterized by graph convolutions, lies at the core of GNNs. Torch geometric offers many models of GNNs that are ready to use in a network, with the most famous ones being GCNConv [KW17] and GATConv [VCC$^+$18], both of which we discussed in the Background section. Other state-of-the-art which are built upon the idea of the previously mentioned models surfaced recently including GraphSAGE [HYL18], GraphConv [MRF$^+$21], TransformerConv [SHF$^+$21], GENConv [LXTG20], GeneralConv [YYL21], and GATv2Conv [BAY22]. In this section I will explain these state-of the art layers and how they work.

3.5.1 GraphSAGE

GraphSAGE[HYL18], which stands for Graph Sample and Aggregated, is a graph neural network architecture designed for inductive learning on large graphs. GraphSAGE aims to generate embeddings for nodes in a graph by sampling and aggregating information from their local neighborhoods.
GraphSAGE starts by sampling a fixed-size neighborhood around each node in the graph. This neighborhood includes the central node itself and its adjacent nodes. For each node and its sampled neighborhood, GraphSAGE defines an aggregation function. This function aggregates information from the node and its neighbors to create a representative vector for the node. The aggregation function is parameterized, meaning it uses learnable parameters. The model learns how to aggregate information from different nodes based on their roles in the neighborhood. GraphSAGE learns embeddings for each node by iteratively sampling and aggregating information from its neighborhood. The embeddings are updated through training using backpropagation and optimization techniques.
One key feature of GraphSAGE is its inductive learning capability. Once the model is trained on a graph, it can generalize to unseen nodes during inference. This is achieved by using the learned aggregation functions, allowing the model to adapt to new nodes or

graphs.
In each layer, GraphSAGE performs a form of graph convolution operation by aggregating information from neighboring nodes. This allows the model to capture local graph structures and relationships. This is represented in Equations 3.1 and 3.2:

$$h^l_{N(i)} = AGGREGATE_l(\{h^{l-1}_j : j \in N(i)\}) \tag{3.1}$$

$$h^l_i = \sigma(F(h^{l-1}_i \oplus h^l_{N(i)})) \tag{3.2}$$

where:
- $\oplus$ is the vector concatenation operation.
- $N(i)$ is the uniform sampling function that returns a subset of all neighbours.
- σ is the activation function (ReLU),
- h^{l-1}_j is the node embedding of node j at layer l-1,
- F is a simple affine transform or neural network.

GraphSAGE typically consists of multiple aggregation layers. Each layer refines the node embeddings by sampling and aggregating information from increasingly larger neighborhoods. This hierarchical aggregation helps the model capture both local and global graph structures. In summary, GraphSAGE operates by iteratively sampling and aggregating information from local neighborhoods of nodes in a graph. This process allows it to generate embeddings that capture the structural and relational aspects of the graph. The use of parameterized aggregation functions and inductive learning makes GraphSAGE particularly effective for large-scale graph-based learning tasks.

3.5.2 GraphConv

GraphConv [MRF+21] is very similar to GCNConv [KW17]. The difference is that GraphConv preserves central node information by omitting neighborhood normalisation. So the equation for GraphConv is represented in Equation 3.3

$$h^{l+1}_i = W_1 h^l_i + W_2 \sum_{j \in N(i)} e^l_{j,i} . h^l_j \tag{3.3}$$

where $e^l_{j,i}$ denotes the edge weight from source node j to target node i at layer l.

3.5.3 TransformerConv

For TransformerConv [SHF+21], the schematic diagram of a layer is shown in Figure 3.11. It consists of the input with Positional Encodings (PEs), the multi-head attention mechanism with attention restricted to local neighbors, and the feed forward module. The attention mechanism is a function of the neighborhood connectivity for each node, which is shown in Equation 3.4

$$h_i^{l+1} = W_1 h_i^l + \sum_{j \in N(i)} \alpha_{i,j} W_2 h_j^l \tag{3.4}$$

where the attention coefficients $\alpha_{i,j}$ are computed via multi-head dot product attention shown in Equation 3.5

$$\alpha_{i,j} = Softmax(\frac{(W_3 h_i)^T (W_4 h_j)}{\sqrt{d}}) \tag{3.5}$$

Positional Encoding is represented by Laplacian PEs. The feed forward module employs batch normalization. Graph Transformer is extended to have edge representation. Two aspects are noteworthy in this extended-edge architecture [Dwi21]: the fusion of edge features with their corresponding pairwise implicit attention scores, and the presence of a dedicated edge feature pipeline at each layer. Equation 3.5 becomes Equation 3.6

$$\alpha_{i,j} = Softmax(\frac{(W_3 h_i)^T (W_4 h_j)}{\sqrt{d}} . W_5 e_{ij}) \tag{3.6}$$

where W_1, W_2, W_3, W_4, W_5 are learnable parameters and d represents the embedding dimension.

3.5.4 GENConv

GENConv (GENeralized Graph Convolution) [LXTG20] is a proposed graph convolutional model that aims to leverage the beneficial properties of invariance and equivariance in graph learning. It extends beyond traditional aggregation functions like mean, max, and sum. The model introduces generalized aggregation functions, which are permutation-invariant set functions parameterized by a continuous variable. This diversity provides flexibility in capturing different types of relationships within graph-structured data. These aggregation functions are continuous and differentiable with respect to the parameter. This property enables the use of optimization techniques during training and facilitates smooth transitions between different aggregation behaviors. GENConv emphasizes the ability to interpolate between parameter values to discover and adapt aggregation functions for specific tasks. This is crucial for tailoring the model's behavior to different data characteristics and learning requirements.
To empirically validate these properties, the model introduces two families of generalized mean-max aggregation functions: SoftMax aggregation and PowerMean aggregation.

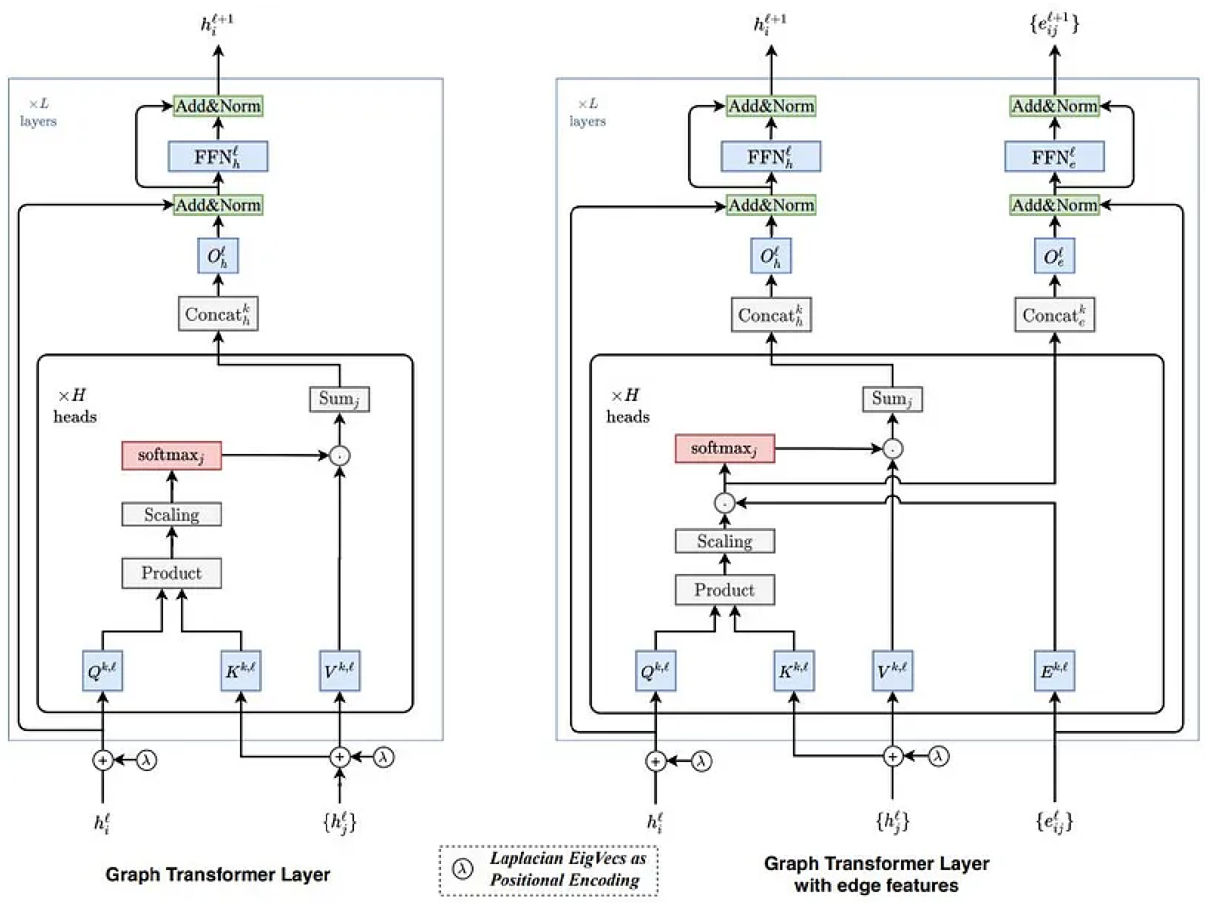


Figure 3.11: TransformerConv Architecture with and without edge features from [Dwi21]

- SoftMax aggregation, parameterized by an inverse temperature (β), exhibits mean-like behavior for low inverse temperatures and approaches max aggregation for high inverse temperatures.
- PowerMean aggregation, parameterized by a power factor (p), serves as a generalized mean-max aggregation function, including special cases like harmonic mean, geometric mean, arithmetic mean, and quadratic mean.
Its message construction is given by Equation 3.7

$$h_i^{l+1} = MLP(h_i^l + AGG(\{ReLU(h_i^l + e_{ji}^l) + \epsilon : j \in N(i)\}))) \tag{3.7}$$

where MLP is a MultiLayer Perceptron.

- **GeneralConv:** [YYL21] This is a General Convolutional layer. It expects a sparse adjacency matrix and computes the output shown in Equation 3.8

$$h_i^{l+1} = AGG(\{\sigma(Dropout(BN(h_j^l W + b))), j \in N(i)\}) \tag{3.8}$$

where $Dropout$ applies dropout to the node features and BN applies batch normalization to the node features.

3.5.5 GATv2Conv

GATv2Conv [BAY22] is based on the GATConv but it fixes the static attention problem of the standard GATConv layer. In the traditional GAT, linear layers are consecutively applied, leading to an unconditioned ranking of attended nodes with respect to the query node. In contrast, GATv2 allows each node the capability to attend to any other node. Similarly to GAT, the output is represented as 3.9

$$h_i = \sum_{j \in N_i \cup \{i\}} \alpha_{ij}.Wh_j^l \tag{3.9}$$

The difference lies in the calculation of the attention coefficients α_{ij}. For GATv2 , the equation is represented in Equation 3.10

$$\alpha_{ij} = Softmax(a^T LeakyReLU(W_1 h_i + W_2 h_j)) \tag{3.10}$$

Adding the edge attributes transform the equation into Equation 3.11

$$\alpha_{ij} = Softmax(a^T LeakyReLU(W_1 h_i + W_2 h_j + W_3 e_{i,j})) \tag{3.11}$$

To conclude, all of the previously discussed layers (GCNConv, GATConv, GraphSAGE, GraphConv, TransformerConv, GENConv, GeneralConv and GATv2Conv) are Graph Convolutional layers that work with Graph data. They take as parameters the input channels, the output channels and other optional parameters. The forward of these models takes the x tensor , which is the tensor of the node attributes, as well as the edge_index, which is the tensor of the list of edges in the graph. Some of these models are able to take additionally the edge_attr which is the tensor of the edge attributes. The parameters supported by each of these models are listed in the torch geometric GNN Cheatsheet[4].

3.6 Anticipation

Early recognition a.k.a anticipation is a crucial aspect of artificial intelligence and predictive analytics, enabling systems to forecast future events or conditions based on available data. Early recognition and anticipation find applications in various domains, including healthcare, finance, cybersecurity, manufacturing, and environmental monitoring.

Early Recognition vs. Traditional Prediction: Early recognition goes beyond traditional predictive analytics. While prediction typically aims to forecast an event or outcome, early recognition focuses on identifying signs or patterns that precede an event, allowing for proactive decision-making and intervention.

Anticipation in Action Prediction: Anticipation in action prediction allows systems to make proactive decisions. Instead of merely recognizing and reacting to current actions. It also improves the interaction between humans and intelligent systems. When AI systems can anticipate human actions, they can provide more intuitive and responsive interfaces. In domains like surveillance and security, anticipation is vital for early threat detection.

3.6.1 Techniques for Anticipation in Action Prediction

There are multiple techniques for performing Anticipation. We will introduce some of them in this section.

Pattern Recognition: Anticipation relies on recognizing patterns in previous action data. This often involves the use of machine learning and deep learning techniques, including recurrent neural networks (RNNs) and Long Short-Term Memory (LSTM) networks, to model temporal dependencies and predict future actions based on historical sequences. In the paper "Encouraging LSTMs to Anticipate Actions Very Early" [ASS$^+$17], the authors present a novel multi-stage Long Short-Term Memory (LSTM) architecture for ac-

[4]`https://pytorch-geometric.readthedocs.io/en/latest/notes/cheatsheet.html`

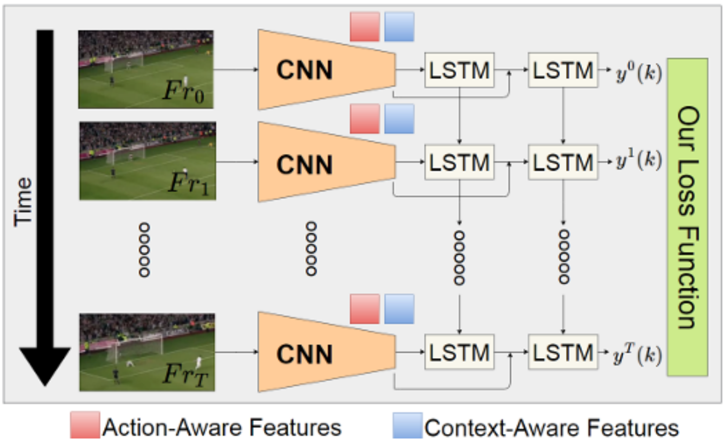


Figure 3.12: Model Architecture using LSTM for Anticipation from [ASS+17]

tion anticipation. The model efficiently extracts and simultaneously utilizes context- and action-aware features, contrasting with existing methods that either focus on global representations for the entire image or video sequence, neglecting the action itself, or localize feature extraction solely to the action through dense trajectories or optical flow, thereby failing to exploit contextual information. The proposed model refrains from relying on optical flow, making it significantly more efficient. On a single GPU, the model analyzes a short video (e.g., 50 frames) 14 times faster than comparable methods utilizing optical flow. The architecture consists of two stages: the first stage emphasizes global, context-aware information by extracting features from the entire RGB image, while the second stage combines these context-aware features with action-aware ones obtained from class-specific activations, typically corresponding to regions where the action occurs. Essentially, the model initially extracts contextual information and then merges it with the localized features. The state-of-the-art model architecture is illustrated in Figure 3.12

Temporal Modeling: Temporal modeling is essential for understanding the flow and order of actions. Techniques like Hidden Markov Models (HMMs) and Conditional Random Fields (CRFs) are used to model temporal relationships and anticipate future actions. These are more traditional techniques. For example, in the paper "Person Movement Prediction Using Hidden Markov Models" [GV06], they use HMMs to anticipate the next movement of a person. The system learns habits automatically and is able to predict the next location of the person based on previous behavior patterns.

Contextual Information: Anticipation is improved by considering contextual information. This involves incorporating data from sensors, environmental factors, and user context to refine action predictions. Context-aware systems can adapt to changing situations and make more accurate forecasts.

Multi-Modal Data Fusion: Anticipation is enhanced when combining data from multiple sources and modalities. For example, in autonomous vehicles, anticipating the actions of pedestrians involves fusing data from cameras, LIDAR, and radar sensors.

Reinforcement Learning: Reinforcement learning is a type of machine learning paradigm in which an agent learns to make decisions by interacting with an environment. The goal of the agent is to maximize a cumulative reward signal over time. In reinforcement learning, the agent takes actions within the environment, and the environment responds with feedback in the form of rewards or penalties. The agent learns to optimize its decision-making strategy to achieve long-term objectives. Reinforcement learning can be used to teach AI systems to anticipate actions by rewarding correct predictions and penalizing incorrect ones. This approach is particularly useful in robotics and control systems.

In the paper "Reinforcement Learning for Predicting Traffic Accidents" [CRKH22] the authors propose a model that focuses on predicting accidents as early as possible while maintaining high prediction accuracy. The task is framed within the context of existing research, leveraging dashcam video data as input. The main structure of the machine learning model is based on Markov Decision Process (MDP). The model predicts two actions, representing accident prediction and fixation prediction.
To efficiently process the entire video frame, the model employs a visual attention mechanism inspired by human perception, combining top-down and bottom-up attention. A convolutional neural network (CNN)-based saliency module processes input data for bottom-up attention, while top-down attention involves passing input data through the foveal vision module before reaching the saliency module. Both attention mechanisms are normalized and combined to form the observation environment for the reinforcement learning model.
The model's action space is defined by accident scores and fixation predictions, forming a concatenated policy. During training, the model uses two policy networks and two fully connected layers with ReLU activation to determine actions. A Long Short-Term Memory (LSTM) layer captures the temporal dependency of connected actions. The model addresses a multi-task problem, simultaneously performing accident anticipation and fixation prediction.
The reward functions are designed based on the accuracy of predictions, utilizing XNOR gates for accident scores and 2-D coordinates for fixation predictions. The weighting factor in the accident reward function is designed to decay exponentially, emphasizing early prediction. The model is trained using the Double Actors and Regularized Critics (DARC) [LMYL21] algorithm, allowing the critics to choose from two policies for accident scores and two for fixation predictions, resulting in four policies in total. This training approach contributes to the effectiveness of the reinforcement learning model in accident anticipation. The architecture is illustrated in Figure 3.13

Convolutional Neural Networks: Convolutional Neural Networks (CNNs) can be

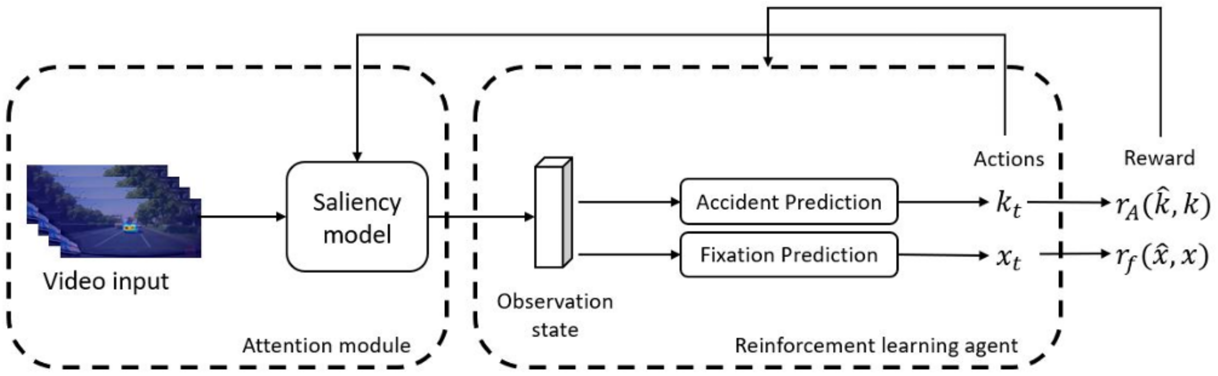


Figure 3.13: Reinforcement Learning Model Architecture from [CRKH22]

utilized for action anticipation in videos by sampling a limited number of frames for training. Many papers worked on CNNs for anticipation tasks [FRG18], [NM18], [WYW19]. The CNN, which is designed with a suitable architecture, incorporates temporal modeling components such as 3D CNNs or temporal layers in 2D CNNs. Training involves predicting future actions based on the sampled frames, and the model is evaluated on a separate video set for performance assessment. Fine-tuning and optimization strategies are employed to enhance accuracy, and the trained model can be deployed for real-time action anticipation on new video sequences, capturing spatial and temporal features to forecast actions before they occur.

Chapter 4

Methodology

This chapter details the methodology employed in this study. It begins by outlining the dataset used for experiments and proceeds to explain the processes of Spatio-temporal Scene Graph Generation, Action Recognition, and Anticipation. This chapter serves as a comprehensive guide to the procedures and techniques applied in the subsequent experimental analysis.

4.1 Dataset

The dataset used in this work is a section of the Home Action Genome dataset (Homage). The original dataset has 30 hours of videos, 70 classes of daily activities and 453 atomic actions. There are 86 object classes (excluding "person"), and 29 relationship classes in the dataset. The HOMAGE dataset has ego-view and 3rd-view for each video. Each video represents one activity and each activity is made of multiple atomic actions. Ground truth scene graph annotation files are present for all videos representing the objects, person and relationships between them present in each frame. A representation of the structure of Home Action Genome is represented in Figure 4.1

The HOMAGE dataset has 3 types of spatial relationship: one relating to attention, one to the relative position and one to the contact. The different types of relationships are presented in Table 4.1

The scene graph annotation files for all videos are formatted in the following way:

{"task": task name, "labels": [{"frame": frame number, "subject": {"name": "person", "rect": {"x": x, "y": y, "w": w, "h": h}}, "object": {"name": object name, "rect": {"x": x, "y": y, "w": w, "h": h}}, "relationships": [relationship1, relationship2...]}, ...]}

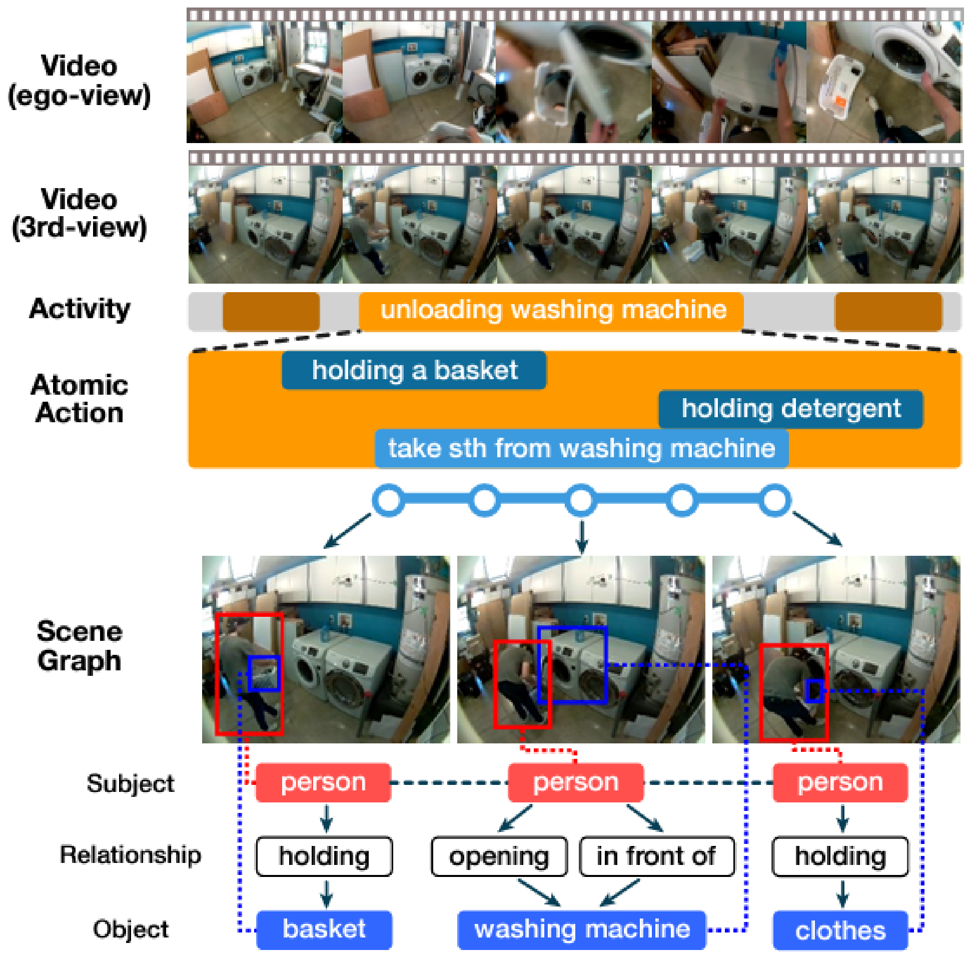


Figure 4.1: Home Action Genome Dataset from [RCJ+21]

Attention	Relative	Contact	
lookingat	infrontof	carrying	coveredby
notlookingat	behind	drinkingfrom	eating
unsure	onthesideof	haveitontheback	holding
	above	leaningon	lyingon
	beneath	notcontacting	sittingon
	in	standingon	touching
		twisting	wearing
		wiping	writingon

Table 4.1: Types of Relationships present in HOMAGE

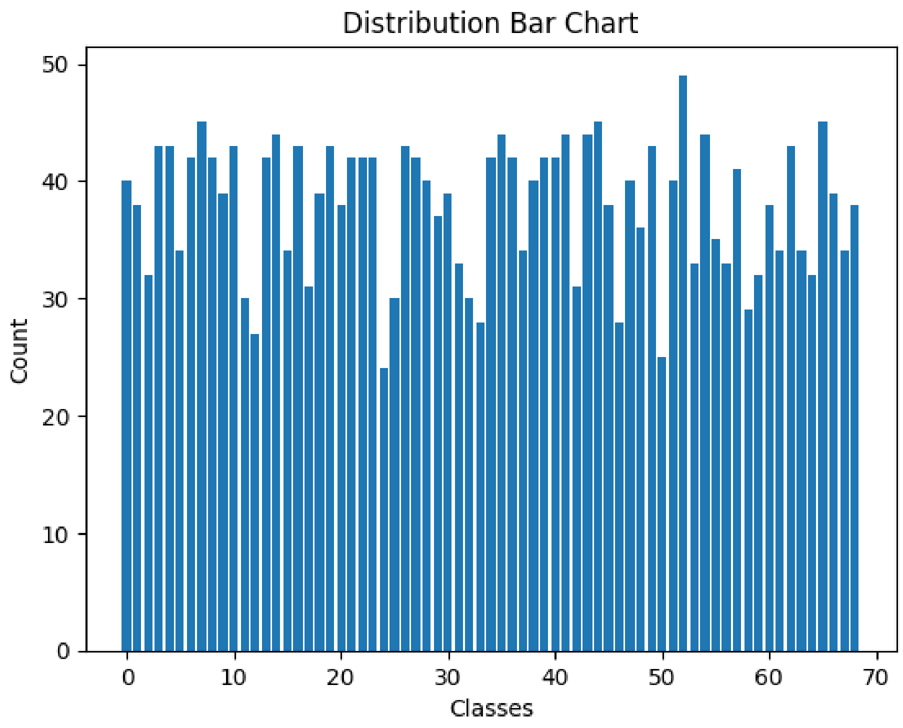


Figure 4.2: Data Distribution between classes

An example from the dataset would be:

{“task”: “p0003_r000_v002_a003-blowdry_hair”,“labels”:[{“frame”:1896, “subject”: {“name”: “person”, “rect”: {“x”: 475, “y”: 206, “w”: 189, “h”: 273 }},“object”: {“name”: ”mirror”, “rect”: {“x”: 199, “y”: 51, “w”: 193, “h”: 242}},“relationships”: [”notcontacting”]}, ...]}

The part of the dataset that is used in the thesis is made of a total of 2617 videos representing 69 daily activities. There are 85 object classes (excluding ”person”) so that’s a total of 86 with ”person”. There is a total of 25 relationship classes. In each frame there is only one subject (person) and one or multiple objects. The relationship is between the subject and an object. Multiple relationships could be present between the subject and an object, and multiple relationships could be present in the frame between the subject and multiple objects. Each video represents one activity from start to end. The distribution of the videos among the 69 activities is shown in Figure 4.2 . The classes are balanced enough, with most of them having a count between 25 and 45.

4.2 Spatio-temporal Scene Graph Generation

The first step to creating spatio-temporal scene graphs is creating spatial scene graphs. We do this by taking the annotation files, processing them to retrieve the desired information and reformat them in a way that can be used to create graphs using the networkx library. Nodes are the subject and the objects, and edges are the spatial relationships between them. Transforming the graphs into spatio-temporal is a matter of adding temporal connections between the subjects, and between each object to itself in consecutive frames.
Since the tasks are labeled only in the annotation files and with many prefixes (example: p0003_r000_v002_a003-blowdry_hair), the first step is to remove the prefixes and leave only the name of the activity (blowdry_hair). After doing that for all files, each activity is given a class (from 0 to 68); This is the class that should be predicted by the model.
Next is creating the actual graphs. The algorithm we implemented is shown in 1.

This is what the algorithm is doing: For the first frame of the video, the subject and object nodes are added, as well as the spatial relationships between them. The algorithm then checks if the next frame number is still the same as the previous one, and if so it means that there are additional objects to be added, and so it adds the object and connects it to the same previous subject with an edge based on the relationships between them. It keeps doing that until the frame number is different from the previous one, meaning it's a new frame. Since it's not the first frame the algorithm starts adding temporal connections between the node (subject or object) to itself in the previous frame. If many instances of the same object are present in the previous frame, the algorithm checks for minimal distance in the bounding box position under the assumption that in consecutive frames the object would not move too much. After creating the spatio-temporal graph, edge_index and edge_attr, x and y are specified as follows:
- edge_index is the torch tensor of the list of edges in the graph G.
- edge_attr is the torch tensor of the numerical edge relationship attributes of all edges in the graph G.
- x is the torch tensor of the node attributes of all nodes of the graph G, where node attributes are the object label and the position (x, y, w, h) of the object.
- y is the torch tensor of the numerical label of the task (activity done in the video).
Then data is created as a Data from torch_geometric.data, with parameters x, edge_index, edge_attr and y.

For visualization purposes, the spatio-temporal graph is reduced to just the first 5 frames. The position of the nodes is based on the center of the bounding box of the subject/ object, translated by a fixed amount for each timesteps for better visualization. Figure 4.3, Figure 4.4 and Figure 4.5 are all examples of spatio-temporal graphs generated from the Home Action Genome dataset and using the algorithm 1. Graphs defer in complexity based on the scene and its elements.

Algorithm 1 Algorithm for creating Spatio-Temporal Scene Graphs

```
Sort the frames in consecutive order
$node\_id, subject\_id \leftarrow 0$
$frame\_idx, start \leftarrow -1$
$first \leftarrow False$
$curr\_name,\ new\_name,\ curr\_id,\ new\_id \leftarrow [\ ]$
$G \leftarrow nx.Graph()$
for $frame\_data$ in $data$ do
    $fr \leftarrow frame\_data['frame']$
    if $frame\_idx \neq fr$ then
        if $(start == 1)$ & $(first == False)$ then
            $curr\_name \leftarrow new\_name$
            $curr\_id \leftarrow new\_id$
        end if
                                                        ▷ Add Subject Node
        $subject\_name \leftarrow frame\_data['subject']['name']$
        $subject\_rect \leftarrow frame\_data['subject']['rect']$
        $name\_label \leftarrow mapped\_label(subject\_name)$
        Create node with node_id, name_label, position
        $node_attributes \leftarrow name_label,\ x,\ y,\ w,\ h]$
        $subj\_id \leftarrow node\_id$
        $node\_id \leftarrow node\_id + 1$
                                                        ▷ Add Object Node
        $object\_name \leftarrow frame\_data['object']['name']$
        $object\_rect \leftarrow frame\_data['object']['rect']$
        $name\_label \leftarrow mapped\_label(object\_name)$
        Create node with node_id, name_label, position
        $node\_attributes \leftarrow name\_label,\ x,\ y,\ w,\ h]$
                                                        ▷ Add Relationship Edges
        for $i$ in $range(len(frame\_data['relationships']))$ do
            $rel\_label \leftarrow mapped\_rel(relationship)$
            $edge\_attributes \leftarrow rel\_label$
            Add edge between subj_id and node_id
        end for
```

```
if $start \neq -1$ then
    $first \leftarrow False$
    if $subject\_name\ in\ curr\_name$ then
        $index \leftarrow curr\_name.index(subject\_name)$
        $rel\_label \leftarrow mapped\_rel('temporal')$
        $edge\_attributes \leftarrow rel\_label$
        $Add\ edge\ between\ subj\_id\ and\ curr\_id[index]$
    end if
    if $object\_name\ in\ curr\_name$ then
        if $Count\ of\ object\_name\ in\ curr\_name\ ==\ 1$ then
            $index \leftarrow curr\_name.index(object\_name)$
            $rel\_label \leftarrow mapped\_rel('temporal')$
            $edge\_attributes \leftarrow rel\_label$
            $Add\ edge\ between\ node\_id\ and\ curr\_id[index]$
        else
            $Find\ distance\ between\ the\ object\ and\ all\ occurances$
            $min\_diff\_index \leftarrow\ index\ of\ the\ object\ in\ curr\_name\ with\ min\_dist$
            $rel\_label \leftarrow mapped\_rel('temporal')$
            $edge\_attributes \leftarrow rel\_label$
            $Add\ edge\ between\ node\_id\ and\ curr\_id[min\_diff\_index]$
        end if
    end if
    $new\_name \leftarrow [subject\_name, object\_name]$
    $new\_id \leftarrow [node\_id - 1, node\_id]$
else
    $start \leftarrow 1$
    $first \leftarrow True$
    $curr\_name \leftarrow [subject\_name, object\_name]$
    $curr\_id \leftarrow [node\_id - 1, node\_id]$
end if
$node\_id \leftarrow node\_id + 1$
$frame\_idx \leftarrow frame\_data['frame']$
```

```
        else
            $object\_name \leftarrow frame\_data['object']['name']$
            $object\_rect \leftarrow frame\_data['object']['rect']$
            $name\_label \leftarrow mapped\_label(object\_name)$
            $Create\ node\ with\ node\_id,\ name\_label,\ position$
            $node\_attributes \leftarrow name\_label,\ x,\ y,\ w,\ h]$
                                                            ▷ Add Relationship Edges
            for $i\ in\ range(len(frame\_data['relationships']))$ do
                $rel\_label \leftarrow mapped\_rel(relationship)$
                $edge\_attributes \leftarrow rel\_label$
                $Add\ edge\ between\ subj\_id\ and\ node\_id$
            end for
            if $first == True$ then
                $Append\ object\_name\ to\ curr\_name$
                $Append\ node\_id\ to\ curr\_id$
            else
                if $object\_name\ in\ curr\_name$ then
                    if $Count\ of\ object\_name\ in\ curr\_name\ ==\ 1$ then
                        $index \leftarrow curr\_name.index(object\_name)$
                        $rel\_label \leftarrow mapped\_rel('temporal')$
                        $edge\_attributes \leftarrow rel\_label$
                        $Add\ edge\ between\ node\_id\ and\ curr\_id[index]$
                    else
                        $Find\ distance\ between\ the\ object\ and\ all\ occurances$
                        $min\_diff\_index \leftarrow\ index\ of\ the\ object\ in\ curr\_name\ with\ min\_dist$
                        $rel\_label \leftarrow mapped\_rel('temporal')$
                        $edge\_attributes \leftarrow rel\_label$
                        $Add\ edge\ between\ node\_id\ and\ curr\_id[min\_diff\_index]$
                    end if
                end if
                $Append\ object\_name\ to\ new\_name$
                $Append\ node\_id\ to\ new\_id$
            end if
            $node\_id \leftarrow node\_id + 1$
        end if
    end for
```

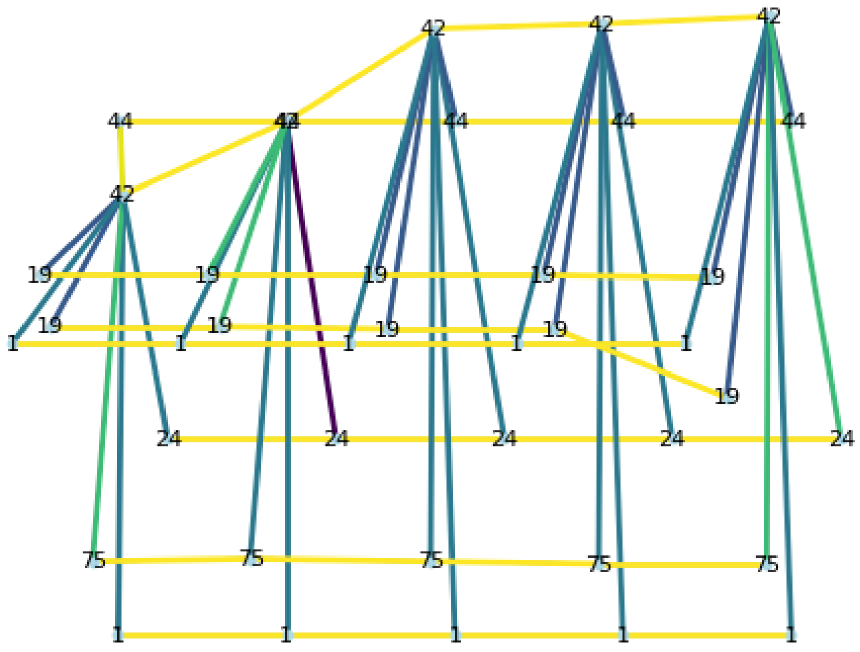


Figure 4.3: Example of Scene Graph for activity "Put Away Leftovers"

For example, Figure 4.3 shows the first 5 timeframes of the action "Put Away Leftovers". We can see that this scene graph has many nodes, including: 42: "person", 44: "countertop", 24: "fridge", 19: "food", 75: "microwave" and 1: "cabinet". Inside each frame the person is connected to the different objects with spatial relationships such as "lookingat", "notlookingat", "holding", "carrying", "touching", "onthesideof"... And successive frames are connected with temporal edges. We can see the movement of the person from the way the position of node 42 is changing.
Figure 4.4 has a less complicated graph. It represents the action of "load the dishwasher". The nodes present in this graph are 42: "person", 13: "dishwasher", 44: "countertop", 20: "detergent" and 2: "sink". As mentioned previously, the person is connected to the objects by spatial relationships and then the nodes are connected from one frame to another with temporal edges.
Figure 4.5 has a very simple scene graph with just 2 nodes: 42: "person" and 37: "laptop". The spatial relationships represented here are "infrontof", "lookingat" and "touching", and the activity represented here is "Use Laptop".
Naturally the full scene graphs of most videos are way bigger and more complex with a big number of nodes and relationships, depending on the number of frames present in a video.

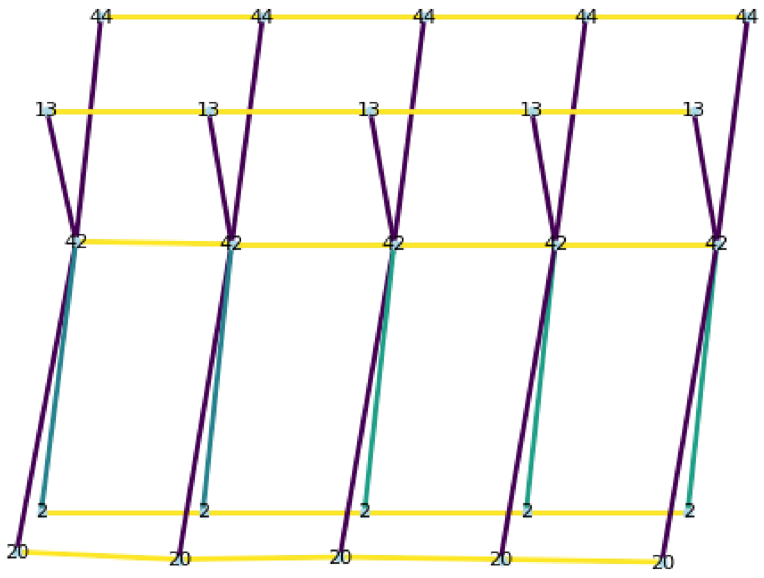


Figure 4.4: Example of Scene Graph for activity ”Load Dishwasher”

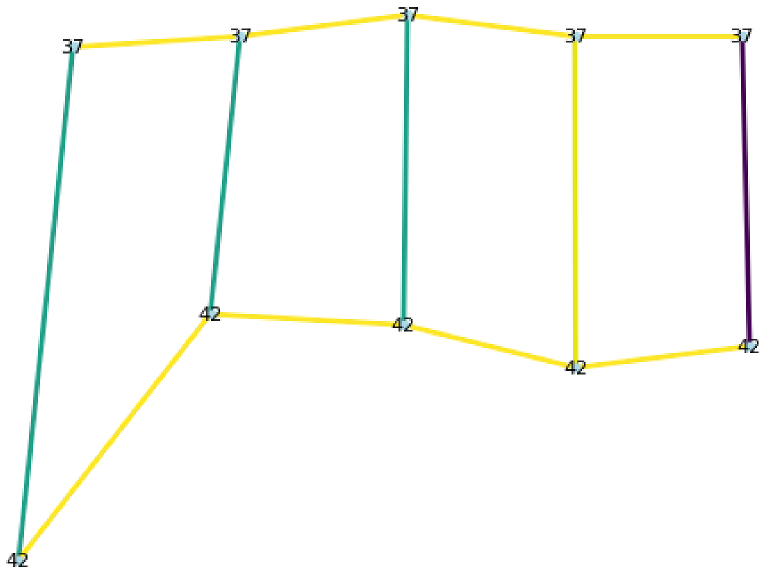


Figure 4.5: Example of Scene Graph for activity ”Use Laptop”

4.3 Action Recognition

The task at hand is action recognition based on scene graphs, which is why building a GNN (Graph Neural Network) makes perfect sense. There are many possible layers to choose from to build the network, all easily accessible through the torch geometric library. Because of the nature of the graphs being spatio-temporal, it is important to include edge attributes while training the model and not just the node attributes. This reduces the number of potential layers significantly as not all of them support edge attributes. 13 potential layers remain that are filtered using the torch geometric GNN cheatsheet, of which 5 take input channels and output channels in the same way that is readily available in the data. These 5 layers are: TransformerConv, GENConv, GeneralConv, GATConv and GATv2Conv. We perform preliminary tests on the five different layers at our disposal, investigate their strengths and weaknesses and compare their performance on the data. The input to the model is the scene graphs (inputted as x, edge_index and edge_attr) and the output is the class of the activity.

The input graphs have 5 node features (object label, x, y, w, and h) and 1 edge attribute (relationship label). The model comprises:

- 4 Graph Convolutional Layers, each with 64 channels and ReLU activation.
- Global mean pooling layer for spatial dimension reduction while retaining important information.
- Dropout of 20% to prevent overfitting.
- 2 Dense Layers to create a fully connected network.
- Softmax layer to transform the output into probabilities.

Since we have 2617 scene graphs, we divide them in the following way: 2000 for training and 617 for testing (23.5%). We use the torch_geometric.data DataLoader to create the train_loader and test_loader with a batch size = 200. Since we are dealing with a classification task, we use the Cross Entropy loss. We also use the Adam optimizer with a learning rate of 0.008.

For the training loop, we iterate in batches over the train loader, clear the gradients, get the predicted output of the model on this data, compute the loss between that predicted output and the ground truth labels, and use loss.backwards to compute the gradients of the model parameters with respect to the loss and perform backpropagation through the computational graph, calculating how much each parameter contributed to the error. optimizer.step then updates the model parameters based on the computed gradients. The optimizer adjusts the weights of the model to minimize the loss.

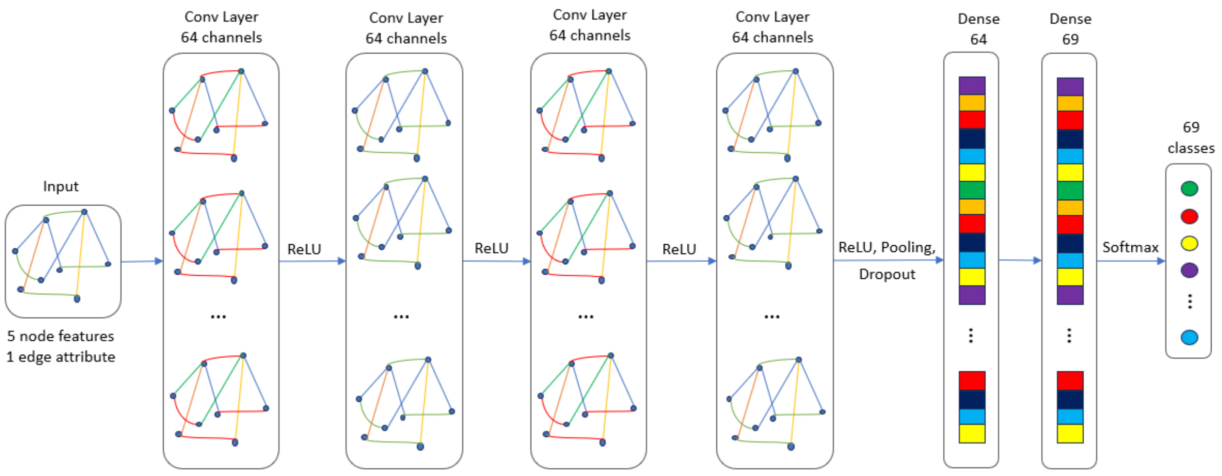


Figure 4.6: Test Network

The testing loop iterates through a loader (could be train or test), gets the predicted output and computes the loss and accuracy.

We then iterate through the number of epochs, in this case set to 200, train the model and then get the accuracy and loss for both training and testing.

4.4 Anticipation

The goal of anticipation is to predict the activity before the video is over. We do that by removing the nodes and edges of the last few frames of a video when creating the scene graphs, and training the best performing model on them. This way the model is forced to learn to recognize an action by its first few frames without having the luxury of processing the full video. In this section we first perform anticipation by removing the last 5, 10, 20, 30 and 50 frames of a video, and creating the scene graphs without them. If a video has a fewer number of frames they remain intact. The graphs are then trained on the best performing network for comparison. It is important to note that the same training/testing partition is used for all experiments so that the comparison is fair.

After plotting the distribution of the number of frames amongst the videos, illustrated in Figure 4.7, we can see that most videos have less than 50 frames, and so keeping them intact while removing 50 frames from videos that have more frames is not fair. The solution is to perform anticipation by removing a percentage of the frames from the end. This way is more fair for all videos. Videos with less than 10 frames are discarded in this part, and so the total number of videos used goes down from 2617 to 2432, and the train test split

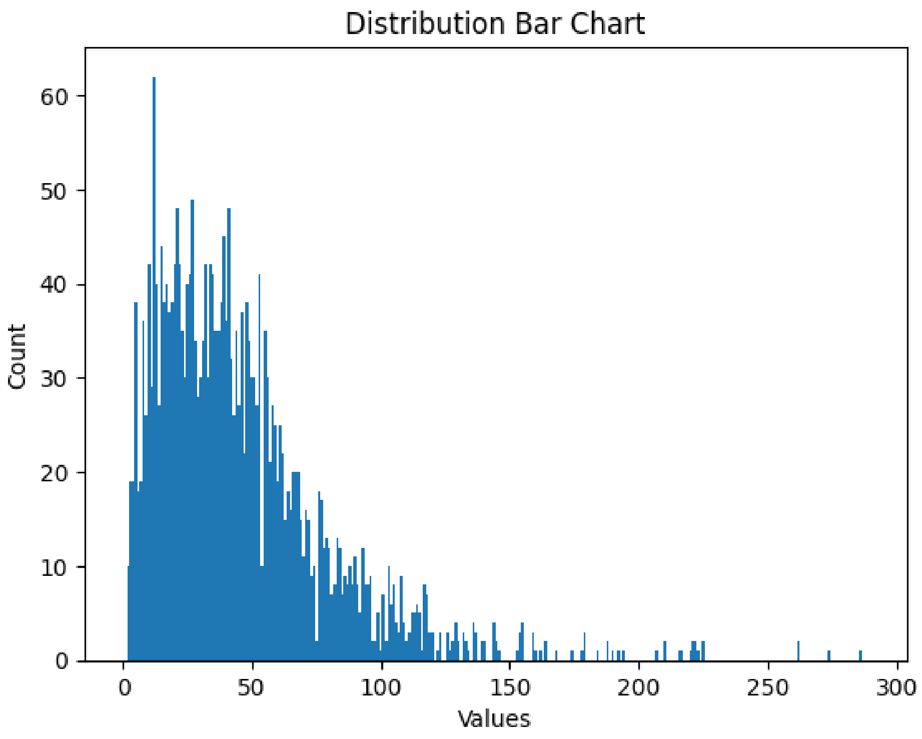


Figure 4.7: Distribution of Number of Frames in the videos

becomes 1900 for training and 532 for testing (27% for test). We first run the model on the reduced dataset without anticipation, and then we perform anticipation of 10%, 20%, 30% , 40%, 50%, 60% , 70%, 80%, 90% and compare and plot the results.

The algorithm for creating the spatio-temporal scene graphs for the anticipation task is very similar to that described as Algorithm 1 but with 2 main differences. First we need to add a counter to count what consecutive frame number we are on, and second we need to calculate how many frames to remove from the end of each video based on the percentage. We also add an if condition to add nodes and edges of only graphs that have more than 10 frames (to eliminate videos with few frames), and to stop adding nodes and edges once the frame counter reaches the desired number of frames for the anticipation task, calculated as total number of frames - number of frames to be eliminated from the end. The changes to the algorithm are presented in Algorithm 2.

Algorithm 2 Algorithm for creating Spatio-Temporal Scene Graphs for Anticipation

```
$counter \leftarrow 0$
$Sort\ the\ frames\ in\ consecutive\ order$
$node\_id, subject\_id \leftarrow 0$
$frame\_idx, start \leftarrow -1$
$first \leftarrow False$
$curr\_name,\ new\_name,\ curr\_id,\ new\_id \leftarrow [\ ]$
$G \leftarrow nx.Graph()$
$frames_r emoved \leftarrow \frac{anticipation\_percentagexnum\_frames}{100}$
for $frame\_data\ in\ data$ do
    $fr \leftarrow frame\_data['frame']$
    if $frame\_idx \neq fr$ then
        $counter++$                                   ▷ Increment frame counter
        if $num\_frames >= 10\ \&\ counter <= (num\_frames\ - frames\_removed)$ then
            $Continue\ the\ Scene\ Graph\ creation\ exactly\ as\ in\ Algorithm\ 1$
            $from\ line\ 10\ and\ on\ but\ inside\ this\ loop$
        end if
    end if
end for
```

Chapter 5

Experiments and Results

This chapter presents the outcomes of the conducted experiments in the areas of Action Recognition and Anticipation. The section on Action Recognition (5.1) provides a detailed analysis of the model's performance, offering insights into its accuracy and effectiveness in classifying actions, while section (5.2) provides the results on Anticipation. The discussion is accompanied by a thorough examination of the experimental setup and the parameters that influence the results, providing a comprehensive understanding of the model's behavior. The chapter serves as a critical evaluation of the proposed methodologies, contributing valuable insights to the fields of Action Recognition and Anticipation.

5.1 Action Recognition

In this section, we perform a preliminary evaluation of the GCN layers by trying the 5 different types of layers as the Convolutional layer in our network. The general network layout is the one shown in Figure 4.6, where we have 4 conv layers with ReLU activation then global mean pooling and dropout, then 2 dense layers and finally a softmax layer.

The Conv layers that we are evaluating are the following: TransformerConv, GENConv, GeneralConv, GATConv, GATv2Conv. All of these layers take as parameters the in_channels, which is the input dimension and out_channels, which is the output dimension, and some layers take other optional parameters such as the edge_dim, which is the edge dimension. The forward method of these layers take the x, edge_index and edge_attr.

The results of using the test network of Figure 4.6 with the different Conv layers and with dropout = 0.2, number of epochs = 200, learning rate = 0.008 and batch size = 200 are shown in Table 5.1

Conv Layer	Training Accuracy	Test Accuracy
TransformerConv	0.41	0.29
GENConv	0.88	0.67
GeneralConv	0.69	0.42
GATConv	0.65	0.44
GATv2Conv	0.41	0.18

Table 5.1: Results of using the test network on the different Conv Layers with dropout = 0.2, number of epochs = 200, learning rate = 0.008 and batch size = 200

We can clearly see that on the Test Network, GENConv outperforms all other layers with 0.88 train accuracy and 0.67 test accuracy. The second best model was GeneralConv with 0.69 train accuracy and 0.42 test accuracy which is a big difference in performance. However it is obvious that General Convolutions are better suited for the task at hand. GATv2Conv performed the worse with just 0.41 train accuracy and 0.18 test accuracy.

It is important to note that since we are working with 69 classes, it is fairly easy for the model to confuse two classes and hence get lower test results.

Since the GENConv performed the best on the test network, this is the layer to be used in the final model. The task now is to optimize this model by tuning the hyperparameters such as the learning rate and number of epochs, and changing the architecture of the network by adding and removing layers and hidden channels.

For the model optimization task, we fix the number of epochs to a reasonable number equal to 250, the dropout to 0.2 and the batch size to 200. The results of the experiments for finding the optimal model are shown in Table 5.2

Model	Learning Rate	Training Acc	Test Acc
GENConv(5,64), GENConv(64,64), GENConv(64,64), GENConv(64,64), dense(64,64), dense(64,69)	0.008	0.92	0.71
GENConv(5,64), GENConv(64,64), GENConv(64,64), GENConv(64,64), dense(64,64), dense(64,69)	0.01	0.87	0.67

GENConv(5,64), GENConv(64,64), GENConv(64,64), GENConv(64,64), dense(64,64), dense(64,69)	0.005	0.94	0.68
GENConv(5,64), GENConv(64,128), GENConv(128,128), GENConv(128,128), dense(128,64), dense(64,69)	0.008	0.87	0.70
GENConv(5,32), GENConv(32,64), GENConv(64,64), GENConv(64,64), dense(64,64), dense(64,69)	0.008	0.87	0.70
GENConv(5,64), GENConv(64,64), GENConv(64,128), dense(128,128), dense(128,69)	0.008	0.87	0.71
GENConv(5,64), GENConv(64,64), GENConv(64,64), GENConv(64,128), dense(128,128), dense(128,69)	0.008	0.87	0.66
GENConv(5,64), GENConv(64,64), GENConv(64,128), dense(128,64), dense(64,69)	0.008	0.93	0.75

GENConv(5,64), GENConv(64,64), GENConv(64,128), GENConv(128,128), dense(128,64), dense(64,69)	0.008	0.89	0.72

Table 5.2: Train and Test Accuracy on different models and configurations, to find the optimal hyperparameters

The first 3 rows are to experiment with the learning rate and determine which one works best. So we fixed the model to 4 GENConv layers and 2 dense layers and changed the learning rate from 0.008 to 0.01 and 0.005. The results show that a learning rate of 0.008 gives the best test accuracy. This is the learning rate that is used from the fourth row on.

Next we change the number of channels and use a different number of Conv layers. The model that gave the best results is the one made of 3 GENConv layers, going from 5 to 64 channels , then from 64 to 64 , and from 64 to 128, with a dense layer going from 128 to 64 neurons and a final dense layer from 64 neurons to the number of classes (69). This model gave a test accuracy of 0.75, which is a big improvement from the previous model. The structure of the best model is illustrated in Figure 5.1

5.2 Anticipation

For the Anticipation task, we follow 2 different strategies: anticipating by a fixed amount of frames, and anticipating by percentage.

Anticipation by fixed amount of frames: First we decided to perform the anticipation task by removing the last X frames of a video where X is 5, 10, 20 , 30 and 50 frames. Since the amount of frames vary from video to video and some of them do not have X frames, the strategy is to keep videos with number of frames less than X intact and remove frames from videos that have more than X frames. The results of this experiment are shown in Table 5.3

The test accuracy consistently decreases as more frames are removed, aligning with the anticipated trend. However, an intriguing deviation occurs between the removal of 30 to 50 frames, where the test accuracy surprisingly improves. This anomaly is rooted in the dataset's heterogeneous video lengths, ranging from as short as 10 frames to as long as 200 frames. Looking at Figure 4.7 we can see that a lot of videos have 30 to 50 frames. The challenge arises from the uniform removal of a fixed number of frames, disproportion-

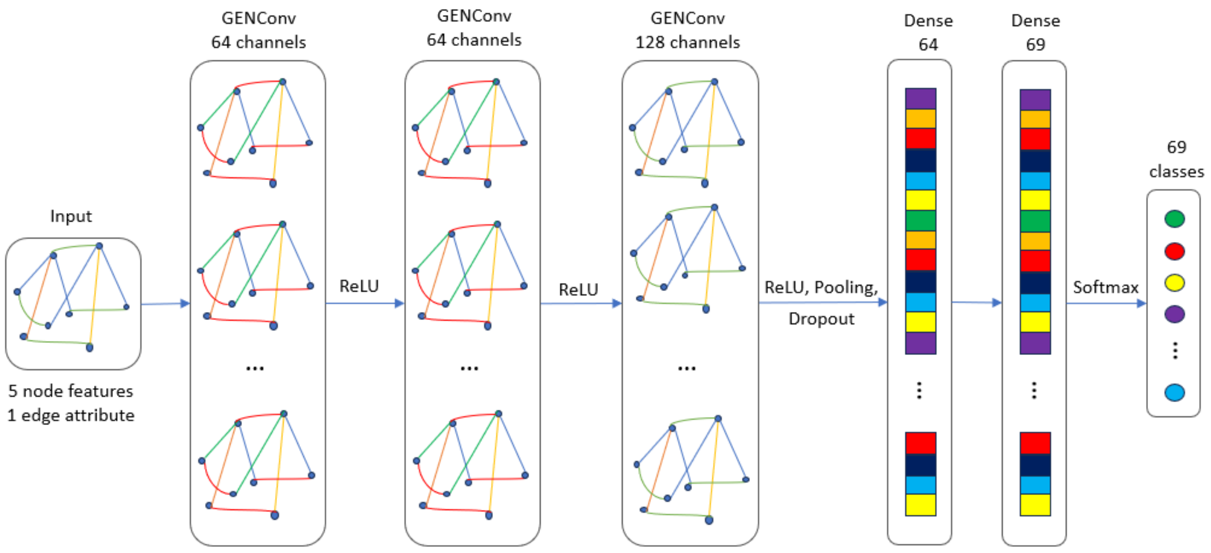


Figure 5.1: Best Network

Number of frames	Training Accuracy	Test Accuracy
0	0.93	0.75
5	0.94	0.71
10	0.93	0.70
20	0.92	0.67
30	0.92	0.65
50	0.93	0.68

Table 5.3: Anticipation Results for fixed number of Frames

ately affecting shorter videos compared to their lengthier counterparts. Consequently, a substantial portion of videos remains unaffected by frame removal, resulting in a dataset where more than half of the videos represent complete activities. In contrast, less than half undergo a significant reduction in frames. This imbalance introduces a dataset bias, as the model trains on scenarios where it encounters more instances of complete activities during training than those with substantial frame reduction. Consequently, this bias elucidates the observed phenomenon where removing 50 frames yields superior results compared to removing 30 frames. It underscores the critical influence of dataset composition on model performance and emphasizes the need for a balanced representation of full and partial activities in the training data. A better approach would be to consider a fixed percentage of frames to be removed and set a threshold for the minimal allowed number of frames in a video.

Anticipation by fixed percentage: Since the first approach was not very indicative and fair, the better approach is to use a percentage of frames P to be removed from the end of the video. This insures fairness across videos. We set the threshold for the minimal allowed number of frames in a video to be 10 frames. Meaning from this point on, videos with less than 10 frames are not considered anymore. After eliminating these videos, the total number of videos goes from 2617 to 2432. We change the train test partition to 1900 videos for training and 532 for testing (22%). We then create the scene graphs for these videos without considering the last P% amount of frames. So for example if a video has 10 frames, and the percentage P is 10% we create the scene graph with just the first 18 frames removing only 2 frames; while if a video has 200 frames and the percentage is 10%, the scene graph is created with the first 180 frames and thus removing 20 frames from the video.

We run this experiment for P = 10, 20, 30, 40, 50, 60, 70 , 80 and 90 to see the difference in results and how far can we anticipate while keeping a good accuracy. The results of this experiment are shown in Table 5.4, and visualized in Figure 5.2.

There is a few things to note from the results shown in Table 5.4 and Figure 5.2. First, the test accuracy without anticipation has increased from the previously best test accuracy of 0.75 to 0.79. The only thing that changed is the elimination of the 185 videos with less than 10 frames. This indicates that the model struggles to recognize patterns in videos with few numbers of frames and thus struggles to classify them, while using videos with a bigger amount of frames improves the results. So removing videos with few frames is a good preprocessing step to improve the quality of the data.

Next we can notice that the test accuracy for the anticipation with 10% is the exact same as the test accuracy without anticipation. This means that by 90% of the video, the model is able to give the same predictions it would give with the full 100% video, without any additional mistakes.
From 20% on the test accuracy starts to decrease, which is expected considering the model

Percentage of frames	Training Accuracy	Test Accuracy
0	0.95	0.79
10	0.96	0.79
20	0.96	0.76
30	0.93	0.73
40	0.96	0.71
50	0.93	0.70
60	0.91	0.67
70	0.94	0.61
80	0.90	0.57
90	0.88	0.54

Table 5.4: Results of Anticipation by Percentage

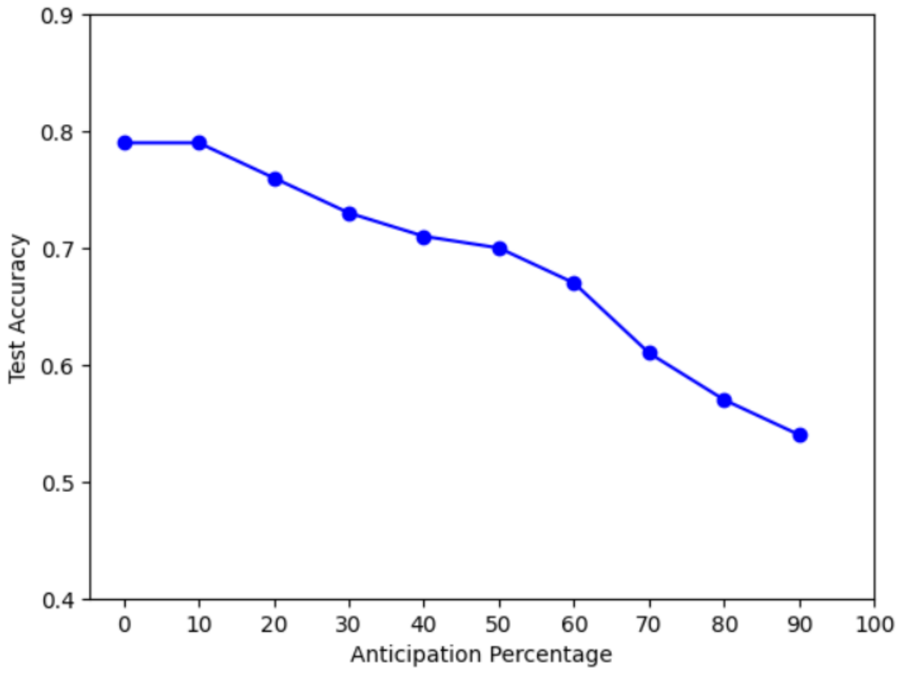


Figure 5.2: Test Accuracy of Anticipation by Percentage

is receiving less and less information each time.
At 20% the test accuracy is still higher than the one obtained using all of the videos without anticipation.
At 50% the test accuracy is still higher than 0.7 , meaning that the models only needs half of the video to be able to accurately predict 70% of the activities correctly. At 90% , meaning the model only has access to the first 10% of the video, it is still able to predict more than half of the classes correctly. Considering that we have 69 classes, this is a good accuracy.

5.3 Results on Reduced Dataset

Since eliminating the 185 videos that have less than 10 frames improved the test accuracy by more than 4% giving the best achieved accuracy as 0.79%, this is the dataset we will be using for this section.

We wanted to visualize the predictions on the test set class by class since we have 69 classes of activities. The results are illustrated in Figure 5.4. The blue bars represent the count of the correctly predicted instances of a class, while the red represent the additional instances of the class that were incorrectly predicted. We can see that the performance is better for some classes and worse for others. A few things to note from this illustration:

- There are 20 classes that were predicted 100% correct, such as 3: ”clean_stovetop”, 62: ”use_microwave” and 23: ”wash_vegetables”.
- 1 class was predicted 0% correctly, which is 50: ”prepare_for_shower”. This class was always confused with class 12: ”draw_bath”.
- The rest of the 69 classes (48 classes) were neither of the 2 extremes. Most of them were predicted correctly the majority of the time and incorrectly a few times.

The Recall of each class are illustrated in Figure 5.5. We can see that 20 classes have a recall of 100% and 1 has 0% while the rest are mostly between 50 and 90%. The median of the recall of all classes is 83%, while the average of the recall of all classes is 78.3%.

The Precision of each class are illustrated in Figure5.3. We can see that 17 classes have a precision of 100% and 1 has 0% while the rest are mostly between 50 and 90%. The median of the precision of all classes is 83%, while the average of the precision of all classes is 77.8%.

The mean Average Precision (mAP) is then calculated from the precision-recall curve, where each AP is the area under the curve and the mAP is the mean of all APs. We obtain a mAP = 63.1%.

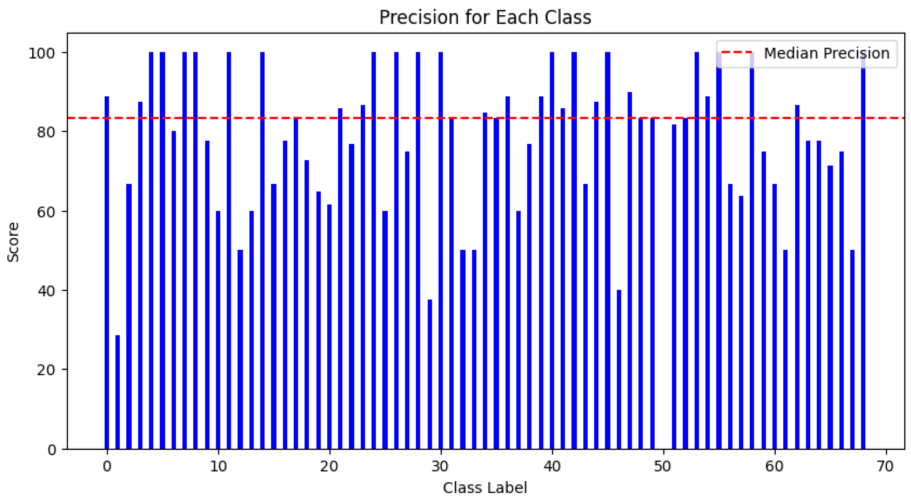


Figure 5.3: Precision percentage per class

The Confusion Matrix is plotted in Figure 5.6, showing the percentage of the predicted classes against the actual classes. We can note that the predictions are mostly along the diagonal which means that mostly the predictions were correct. A few examples where classes were confused are:

- class 1: "unpack from a suitcase" was confused 71% of the time with class 29: "pack a suitcase".
- class 29: "pack a suitcase" was confused 62% of the time with class 1: "unpack a suitcase".
- class 46: "listen to music" was confused 60% of the time with class 57: "use laptop".
- class 26: "serve dinner" was confused 44% of the time with class 10: "eat dinner" and 33% of the time with class 19: "set table"
- class 25: "load and run dryer" was confused 40% of the time with class 32: "unload drying machine".
- class 13: "work at table" was confused 25% of the time with class 60: "organize office supplies".

From the examples above, we can see that for the instances that the predictions were mostly incorrect, the predicted action was similar to the real one, and had similar components (such as pack and unpack suitcase or serve dinner and eat dinner).

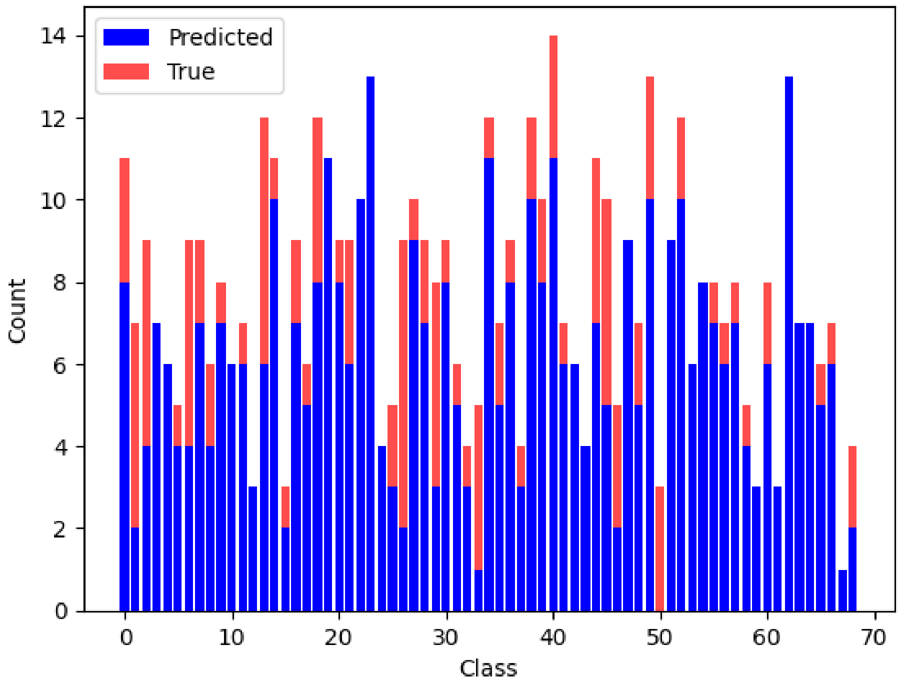


Figure 5.4: True Positives (Predicted) and False Negatives (True) on the test data

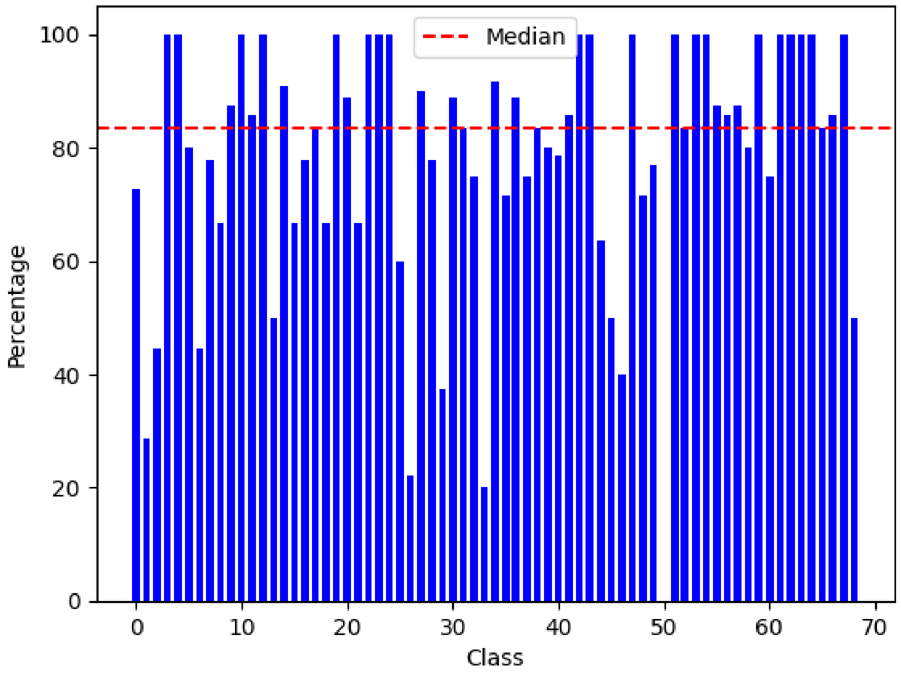


Figure 5.5: Recall percentage per class

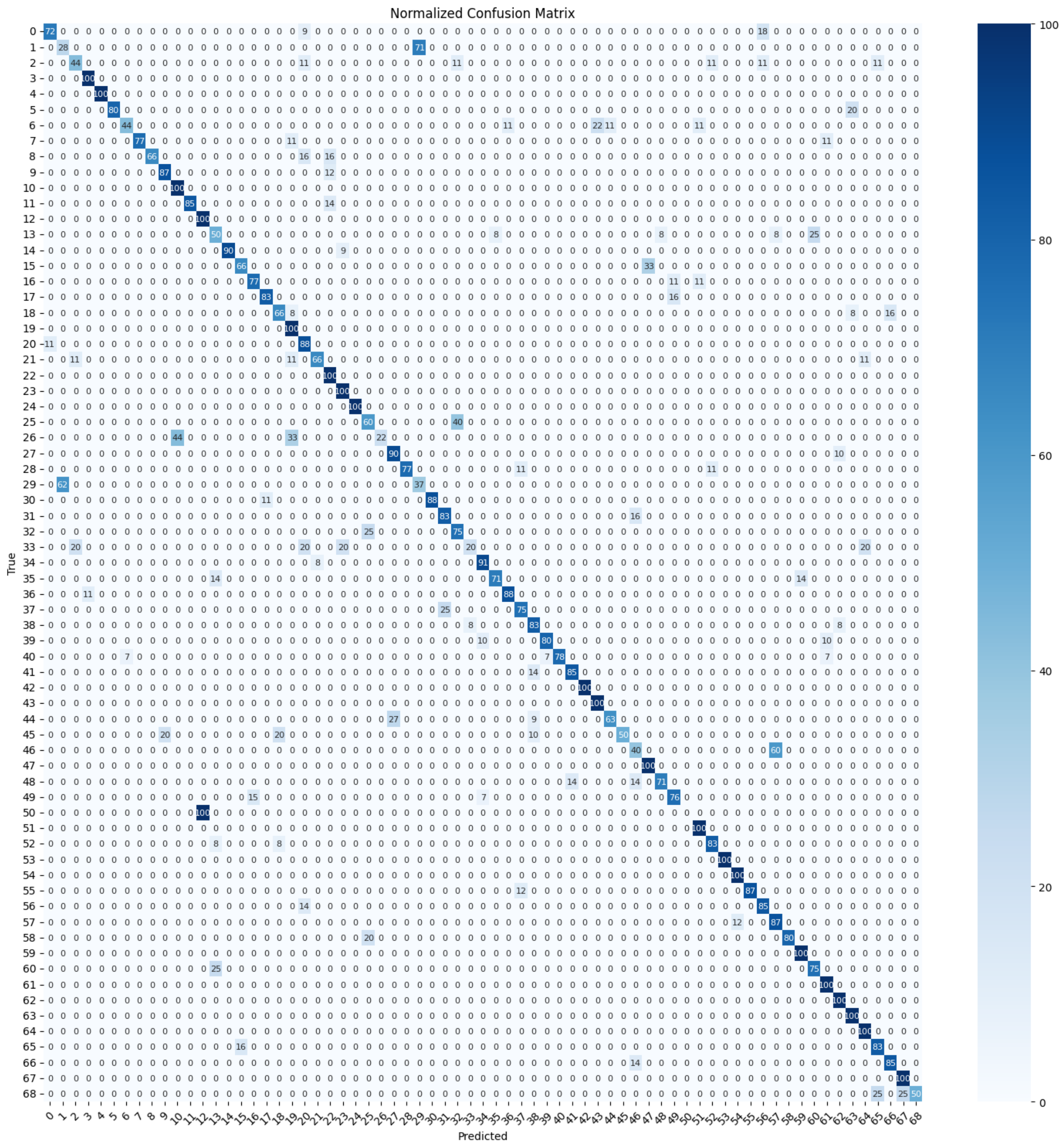


Figure 5.6: Confusion Matrix on Test Data

Chapter 6

Conclusion and Future Directions

In this thesis, we have presented a novel and approach for the Action Recognition and Anticipation tasks. By using Spatio-temporal Scene Graphs as inputs to a Graph Convolutional Network, we were able to classify with good accuracy the activities from the Home Action Genome dataset.

Throughout this research we were able to contribute in important areas. First we created an algorithm for converting spatial scene graphs on individual frames into one big spatio-temporal scene graph for the full video.

Next we experimented on different model architectures and evaluated different convolutional layers, thus optimizing the network with GENConv layers, dense layers, global mean pooling, dropout and Softmax. This model achieved an accuracy of 79% on our test data and a mean Average Precision of 63.1%[JKFFN19] for a classification over 69 classes .

For the anticipation task, we introduced a novel approach using the spatio-temporal graphs of a percentage of frames, and using the best performing GCN from the Action Recognition task. With 10% anticipation we achieved the exact same test accuracy as without anticipation, meaning that the model is able to give the same predictions by having 90% of the video as it would by having 100% of the video; and with anticipation of 50% we achieved an accuracy of 0.7.

The work done in this thesis resulted in good test accuracy on the Home Action Genome dataset, but there is always room for improvement. Here are some methods we propose that could potentially improve our model:

Using human skeleton: When creating the Scene Graphs, we are currently presenting the person as one node, positioned at the center of its bounding box. This can potentially be improved by representing the person by its skeleton instead, where joints are the nodes. This way relationships can be divided depending on the body parts that are

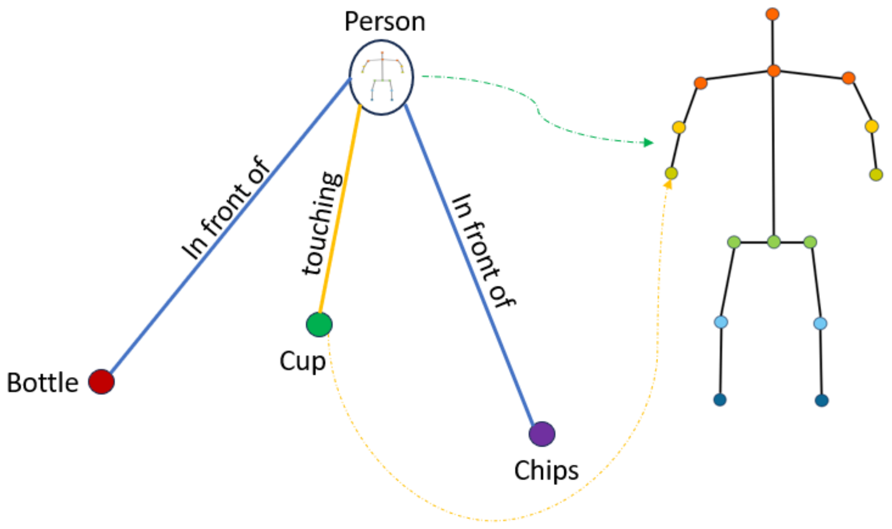


Figure 6.1: Nested Graphs using Skeleton

engaged. This can be done using nested Graphs, which means that the node person would further have a graph inside it for the skeleton. This is illustrated in Figure 6.1. In this case, contact relationships such as ”touching” and ”holding” can be directly connected to the hands, while relationships such as ”looking” can be connected to the head. This can give a deeper understanding of which joints the person uses for each action, however it is not a simple task as the connections are much deeper.

Adding more attributes: An easier method to improve the model is by adding more node and edge attributes. We used the label and position as attributes, but other attributes can be added too such as the velocity of the object (change in position from one frame to the next), or the type of spatial connection for the edges (contact- relative - attention)

Using edge weights: In this thesis we considered all edges to have equal weight, however a possible direction to take is to give different weights for different types of edges. For example contact relationships could be more important than relative relationships and so on. So assigning weights to the edges could potentially improve the model.

Other methods could include adding temporal attention mechanisms, use some graph refinement techniques , use data augmentation techniques, and reduce inference time to make the model more practical for applications that require quick action recognition and anticipation. Additionally creating Scene Graphs from scratch could enrich the task and make it more challenging. There are multiple ways to potentially improve the model, all of which require additional research and experimentation.

In conclusion the thesis contributed to many areas of research in Machine Learning and Computer Vision for Action Recognition and Anticipation tasks. The proposed future directions provide a roadmap for further research and development, opening up exciting opportunities for exploration and innovation in this evolving field.

Bibliography

[AG19] Almamon Abdali and Rana Ghani. *A proposed Intelligent Surveillance System for Smart Cities Using Microservice Architecture*. PhD thesis, University of Technology, 01 2019. `doi:10.13140/RG.2.2.17902.36161`.

[AGMM$^+$21] Khandakar Ahmed, Farid Ghareh Mohammadi, Manuel Matus, Farzan Shenavarmasouleh, Luiz Pereira, Ioannis Zisis, and M. Hadi Amini. Towards real-time house detection in aerial images using faster region-based convolutional neural network. *SSRN Electronic Journal*, 01 2021. `doi:10.2139/ssrn.3994191`.

[Ana22] Rishabh Anand. Math behind graph neural networks, Mar 2022. URL: `https://rish-16.github.io/posts/gnn-math/`.

[ASS$^+$17] Mohammad Sadegh Aliakbarian, Fatemeh Sadat Saleh, Mathieu Salzmann, Basura Fernando, Lars Petersson, and Lars Andersson. Encouraging lstms to anticipate actions very early, 2017. `arXiv:1703.07023`.

[BAY22] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks?, 2022. `arXiv:2105.14491`.

[BGX09] Matteo Bregonzio, Shaogang Gong, and Tao Xiang. Recognising action as clouds of space-time interest points. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1948–1955, 2009. `doi:10.1109/CVPR.2009.5206779`.

[BTW$^+$20] Yue Bai, Zhiqiang Tao, Lichen Wang, Sheng Li, Yu Yin, and Yun Fu. Collaborative attention mechanism for multi-view action recognition, 2020. `arXiv:2009.06599`.

[Bul21] Sema Zeynep Bulut. A brief overview of r-cnn, fast r-cnn and faster r-cnn, May 2021. URL: `https://medium.com/mlearning-ai/a-brief-overview-of-r-cnn-fast-r-cnn-and-faster-r-cnn-9c6843c9ffc0`.

[CRKH22] Injoon Cho, Praveen Kumar Rajendran, Taeyoung Kim, and Dongsoo Har. Reinforcement learning for predicting traffic accidents, 2022. arXiv:2212.04677.

[CZ18] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset, 2018. arXiv:1705.07750.

[DHJ+21] Liu D, Xu H, Wang J, Lu Y, Kong J, and Qi M. Adaptive attention memory graph convolutional networks for skeleton-based action recognition, 2021. doi:10.3390/s21206761.

[DLP21] Bruno Degardin, Vasco Lopes, and Hugo Proença. Regina - reasoning graph convolutional networks in human action recognition, 2021. arXiv:2105.06711.

[DMP21] Michael Duhme, Raphael Memmesheimer, and Dietrich Paulus. Fusion-gcn: Multimodal action recognition using graph convolutional networks, 2021. arXiv:2109.12946.

[DT05] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, 2005. doi:10.1109/CVPR.2005.177.

[Dwi21] Vijay Prakash Dwivedi. Graph transformer: Generalization of transformers to graphs, Mar 2021. URL: https://towardsdatascience.com/graph-transformer-generalization-of-transformers-to-graphs-ead2448cff8b.

[FF21] Antonino Furnari and Giovanni Maria Farinella. Rolling-unrolling lstms for action anticipation from first-person video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11):4021–4036, November 2021. URL: http://dx.doi.org/10.1109/TPAMI.2020.2992889, doi:10.1109/tpami.2020.2992889.

[FRG18] Yazan Abu Farha, Alexander Richard, and Juergen Gall. When will you do what? - anticipating temporal occurrences of activities, 2018. arXiv:1804.00892.

[Gir15] Ross Girshick. Fast R-CNN. *arXiv e-prints*, page arXiv:1504.08083, April 2015. arXiv:1504.08083, doi:10.48550/arXiv.1504.08083.

[GPLI21] Yunhao Ge, Yunkui Pang, Linwei Li, and Laurent Itti. Graph autoencoder for graph compression and representation learning. In *Neural Compression: From Information Theory to Applications–Workshop@ ICLR 2021*, 2021.

[GV06] Arpad Gellert and Lucian Vintan. Person movement prediction using hidden markov models. *Studies in Informatics and Control*, 15:17–30, 03 2006.

[HDL+22] Xuejiao Hu, Jingzhao Dai, Ming Li, Chenglei Peng, Yang Li, and Sidan Du. Online human action detection and anticipation in videos: A survey. *Neurocomputing*, 491:395–413, 2022. URL: `https://www.sciencedirect.com/science/article/pii/S0925231222003617`, `doi:10.1016/j.neucom.2022.03.069`.

[Hui18] Jonathan Hui. What do we learn from region based object detectors (faster r-cnn, r-fcn, fpn)?, Mar 2018. URL: `https://jonathan-hui.medium.com/what-do-we-learn-from-region-based-object-detectors-faster-r-cnn-r-fcn-f`

[HYL18] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018. `arXiv:1706.02216`.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv e-prints*, page arXiv:1512.03385, December 2015. `arXiv:1512.03385`, `doi:10.48550/arXiv.1512.03385`.

[JKFFN19] Jingwei Ji, Ranjay Krishna, Li Fei-Fei, and Juan Carlos Niebles. Action genome: Actions as composition of spatio-temporal scene graphs, 2019. `arXiv:1912.06992`.

[JXYY13] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231, 2013. `doi:10.1109/TPAMI.2012.59`.

[KMOK23] Pawit Kochakarn, Daniele De Martini, Daniel Omeiza, and Lars Kunze. Explainable action prediction through self-supervision on scene graphs, 2023. `arXiv:2302.03477`.

[KR17] Tae Soo Kim and Austin Reiter. Interpretable 3d human action analysis with temporal convolutional networks, 2017. `arXiv:1704.04516`.

[KS18] Seemanthini K and Manjunath S.S. Human detection and tracking using hog for action recognition. *Procedia Computer Science*, 132:1317–1326, 2018. International Conference on Computational Intelligence and Data Science. URL: `https://www.sciencedirect.com/science/article/pii/S1877050918307804`, `doi:10.1016/j.procs.2018.05.048`.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[KW17] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017. arXiv:1609.02907.

[KZG+16] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A. Shamma, Michael S. Bernstein, and Fei-Fei Li. Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations. *arXiv e-prints*, page arXiv:1602.07332, February 2016. arXiv:1602.07332, doi:10.48550/arXiv.1602.07332.

[LAE+15] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single Shot MultiBox Detector. *arXiv e-prints*, page arXiv:1512.02325, December 2015. arXiv:1512.02325, doi:10.48550/arXiv.1512.02325.

[LCY14] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network, 2014. arXiv:1312.4400.

[Li22] Tina Li. Scene graph generation, compression, and classification on action genome dataset, Jan 2022. URL: https://medium.com/stanford-cs224w/scene-graph-generation-compression-and-classification-on-action-genome-da

[LKBF16] Cewu Lu, Ranjay Krishna, Michael Bernstein, and Li Fei-Fei. Visual Relationship Detection with Language Priors. *arXiv e-prints*, page arXiv:1608.00187, July 2016. arXiv:1608.00187, doi:10.48550/arXiv.1608.00187.

[LMYL21] Jiafei Lyu, Xiaoteng Ma, Jiangpeng Yan, and Xiu Li. Efficient continuous control with double actors and regularized critics, 2021. arXiv:2106.03050.

[LXTG20] Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. Deepergcn: All you need to train deeper gcns, 2020. arXiv:2006.07739.

[LZH22] Rongjie Li, Songyang Zhang, and Xuming He. Sgtr: End-to-end scene graph generation with transformer, 2022. arXiv:2112.12970.

[MBM14] Sameh Megrhi, Azeddine Beghdadi, and Wided Mseddi. Trajectory feature fusion for human action recognition. 12 2014. doi:10.1109/EUVIP.2014.7018409.

[MG18] Francisco Massa and Ross Girshick. maskrcnn-benchmark: Fast, modular reference implementation of Instance Segmentation and Object Detection algorithms in PyTorch. `https://github.com/facebookresearch/maskrcnn-benchmark`, 2018. Accessed: [Insert date here].

[Mil95] George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, nov 1995. `doi:10.1145/219717.219748`.

[MRF$^+$21] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks, 2021. `arXiv:1810.02244`.

[MS09] Riccardo Mattivi and Ling Shao. Human action recognition using lbp-top as sparse spatio-temporal feature descriptor. In Xiaoyi Jiang and Nicolai Petkov, editors, *Computer Analysis of Images and Patterns*, pages 740–747, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[NM18] Nishant Nikhil and Brendan Tran Morris. Convolutional neural network for trajectory prediction, 2018. `arXiv:1809.00696`.

[Ray22] Ray. A simple guide to yolo and ssd, Mar 2022. URL: `https://medium.com/mlearning-ai/a-simple-guide-to-yolo-and-ssd-3172a0e876f`.

[RCJ$^+$21] Nishant Rai, Haofeng Chen, Jingwei Ji, Rishi Desai, Kazuki Kozuka, Shun Ishizaka, Ehsan Adeli, and Juan Carlos Niebles. Home Action Genome: Cooperative Compositional Action Understanding. *arXiv e-prints*, page arXiv:2105.05226, May 2021. `arXiv:2105.05226`, `doi:10.48550/arXiv.2105.05226`.

[RDGF15] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. *arXiv e-prints*, page arXiv:1506.02640, June 2015. `arXiv:1506.02640`, `doi:10.48550/arXiv.1506.02640`.

[RHGS15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *arXiv e-prints*, page arXiv:1506.01497, June 2015. `arXiv:1506.01497`, `doi:10.48550/arXiv.1506.01497`.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986. URL: `https://api.semanticscholar.org/CorpusID:205001834`.

[RRF23] Ramanathan Rajendiran, Debaditya Roy, and Basura Fernando. Modelling spatio-temporal interactions for compositional action recognition, 2023. arXiv:2305.02673.

[Rud17] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017. arXiv:1609.04747.

[RYT14] Manoj Ramanathan, Wei-Yun Yau, and Eam Khwang Teoh. Human action recognition with video data: Research and evaluation challenges. *IEEE Transactions on Human-Machine Systems*, 44(5):650–663, 2014. doi:10.1109/THMS.2014.2325871.

[SAM$^+$23] Gurkirt Singh, Stephen Akrigg, Manuele Di Maio, Valentina Fontana, Reza Javanmard Alitappeh, Salman Khan, Suman Saha, Kossar Jeddisaravi, Farzad Yousefi, Jacob Culley, Tom Nicholson, Jordan Omokeowa, Stanislao Grazioso, Andrew Bradley, Giuseppe Di Gironimo, and Fabio Cuzzolin. ROAD: The road event awareness dataset for autonomous driving. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1):1036–1054, jan 2023. URL: https://doi.org/10.1109%2Ftpami.2022.3150906, doi:10.1109/tpami.2022.3150906.

[SHF$^+$21] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification, 2021. arXiv:2009.03509.

[SHK$^+$14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[SKB$^+$17] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling Relational Data with Graph Convolutional Networks. *arXiv e-prints*, page arXiv:1703.06103, March 2017. arXiv:1703.06103, doi:10.48550/arXiv.1703.06103.

[SKW$^+$22] Suprosanna Shit, Rajat Koner, Bastian Wittmann, Johannes Paetzold, Ivan Ezhov, Hongwei Li, Jiazhen Pan, Sahand Sharifzadeh, Georgios Kaissis, Volker Tresp, and Bjoern Menze. Relationformer: A unified framework for image-to-graph generation, 2022. arXiv:2203.10202.

[SLJ$^+$14] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. *arXiv e-prints*, page arXiv:1409.4842, September 2014. arXiv:1409.4842, doi:10.48550/arXiv.1409.4842.

[SNpS22] Elham Shabaninia, Hossein Nezamabadi-pour, and Fatemeh Shafizadegan. Transformers in action recognition: A review on temporal modeling, 2022. arXiv:2302.01921.

[SZ14a] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos, 2014. arXiv:1406.2199.

[SZ14b] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv e-prints*, page arXiv:1409.1556, September 2014. arXiv:1409.1556, doi:10.48550/arXiv.1409.1556.

[Tan20] Kaihua Tang. A scene graph generation codebase in pytorch, 2020. https://github.com/KaihuaTang/Scene-Graph-Benchmark.pytorch.

[TBF+15] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks, 2015. arXiv:1412.0767.

[TCAR20] Vincenzo Taormina, Donato Cascio, Leonardo Abbene, and Giuseppe Raso. Performance of fine-tuning convolutional neural networks for hep-2 image classification. *Applied Sciences*, 10:6940, 10 2020. doi:10.3390/app10196940.

[TFL+22] Tsung-Ming Tai, Giuseppe Fiameni, Cheng-Kuang Lee, Simon See, and Oswald Lanz. Unified recurrence modeling for video action anticipation, 2022. arXiv:2206.01009.

[TNH+20] Kaihua Tang, Yulei Niu, Jianqiang Huang, Jiaxin Shi, and Hanwang Zhang. Unbiased scene graph generation from biased training. In *Conference on Computer Vision and Pattern Recognition*, 2020.

[Tri23] Subarna Tripathi. Spatio-temporal graphs for long-term video understanding, May 2023. URL: https://community.intel.com/t5/Blogs/Tech-Innovation/Artificial-Intelligence-AI/Spatio-Temporal-Graphs-for-Long-Term-Video-Understanding/post/1425258.

[TZW+19] Kaihua Tang, Hanwang Zhang, Baoyuan Wu, Wenhan Luo, and Wei Liu. Learning to compose dynamic tree structures for visual contexts. In *Conference on Computer Vision and Pattern Recognition*, 2019.

[VCC^{+}18] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018. `arXiv:1710.10903`.

[VJ01] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–I, 2001. `doi:10.1109/CVPR.2001.990517`.

[Wei19] Jerry Wei. Alexnet: The architecture that challenged cnns, Jul 2019. URL: `https://towardsdatascience.com/alexnet-the-architecture-that-challenged-cnns-e406d5297951`.

[WYW19] Dong Wang, Yuan Yuan, and Qi Wang. Early action prediction with generative adversarial networks. *IEEE Access*, 7:35795–35804, 2019. URL: `http://dx.doi.org/10.1109/ACCESS.2019.2904857`, `doi:10.1109/access.2019.2904857`.

[YCZ^{+}20] Bangguo Yu, Chongyu Chen, Fengyu Zhou, Fang Wan, Wenmi Zhuang, and Yang Zhao. A bottom-up framework for construction of structured semantic 3d scene graph. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8224–8230, 2020. `doi:10.1109/IROS45743.2020.9341228`.

[YLL^{+}18] Jianwei Yang, Jiasen Lu, Stefan Lee, Dhruv Batra, and Devi Parikh. Graph R-CNN for Scene Graph Generation. *arXiv e-prints*, page arXiv:1808.00191, August 2018. `arXiv:1808.00191`, `doi:10.48550/arXiv.1808.00191`.

[YYL21] Jiaxuan You, Rex Ying, and Jure Leskovec. Design space for graph neural networks, 2021. `arXiv:2011.08843`.

[ZHL^{+}22] Yang Zhou, Zhanhao He, Keyu Lu, Guanhong Wang, and Gaoang Wang. Preserve pre-trained knowledge: Transfer learning with self-distillation for action recognition, 2022. `arXiv:2205.00506`.

Acknowledgement

I would like to express my sincere gratitude to my supervisors, Professor Nicoletta Noceti and Federico Figari Tomenotti for their invaluable guidance, unwavering support, and insightful contributions throughout the journey of this thesis. Their expertise and encouragement have been instrumental in shaping the research and enriching my academic experience.

I extend heartfelt appreciation to my family, with profound thanks to my parents, Jamal and Najib, for their endless encouragement and sacrifices that paved the way for my academic pursuits. I am also grateful to my brother Assaad and my sister Marie for their support and understanding.

A special mention of gratitude is reserved for my sister Antonia. Her unwavering support, both emotionally and financially, played a pivotal role in enabling me to pursue and complete my master's degree in Italy. Her belief in my capabilities has been a driving force, and I am truly thankful for her generosity and encouragement.

I would also like to extend my gratitude to my partner Tommaso. His constant support, understanding, and encouragement have been a source of strength throughout this academic endeavor. His belief in my abilities and his unwavering support have made this journey more meaningful.

I am grateful to each of these individuals who, in their unique ways, contributed to the successful completion of this thesis. Their support has been a cornerstone of my academic and personal achievements.