

POLYTECHNIC OF TURIN

Master's Degree in Data Science and Engineering



Master's Degree Thesis

3DYogaSeg: A New Dataset and Benchmark for Skeleton-Based Action Recognition and Segmentation in yoga videos

Supervisors

Prof. Giuseppe Bruno AVERTA

Dott. Chiara PLIZZARI

CANDIDATE

Edoardo MARCHETTI

April 2024

Summary

In recent years, the popularity of online platforms for home exercise programs, particularly yoga, has increased significantly. Yoga involves various poses, known as ‘asanas’, where small differences can change the name and nature of the exercise. However, many video tutorials do not have accurate names or descriptions for these poses, which presents a challenge for beginners. Automated tools for identifying yoga poses offer critical support for users approaching these sessions. Although there is a vast literature on understanding videos, there is a significant gap in the recognition of specific exercises in yoga videos, largely due to a lack of adequate datasets. To solve this problem, we introduced a new dataset to identify and segment yoga poses using skeletal data. In fact, skeletal data are much less affected by changes in viewpoint and background than traditional RGB data, providing a more stable and reliable basis for analysis. We designed a linear interpolation process to merge the videos, creating a dataset that supports both action recognition and segmentation. In total, we collected 2115 videos from YouTube covering 58 different asanas, with an average of 34 sequences per pose and an average duration of 13 seconds per video. The skeletal sequences were extracted with the BlazePose model provided by Mediapipe, an open-source tool developed by Google for computer vision tasks. In our research, using the MS-GCN network-which combines the convolution on graphs of ST-GCN and the temporal segmentation capabilities of MS-TCN-we conducted an ablation study to analyze the performance impact of dataset features constructed by interpolation. In particular, we analyzed the impact of the number of exercise per video, the possibility of having an exercise repeated multiple times in the same video, and the number of transition frames used to connect two clips. The experiments showed that a dataset built with videos with contain fewer non-repeated exercises linked by a high number of transition frames helps to increase the frame-level accuracy of MS-GCN. We believe that using linear interpolation to join videos can be a solid starting point to further explore the conversion of datasets used for activity recognition task into datasets suitable for the task of temporal segmentation of videos.

Table of Contents

List of Tables	VI
List of Figures	VII
1 Introduction	1
1.1 Objectives and motivations for the project	2
1.2 Contributions	3
1.3 Thesis structure	5
2 DeepLearning: from Perceptron to Graph Convolution Neural Networks	7
2.1 The Perceptron	7
2.2 From neuron to "brain"	9
2.2.1 Multi-layer perceptron or Feed Forward Neural Network	9
2.2.2 Backpropagation	11
2.2.3 Learning Rate	11
2.2.4 Weight Initialization	12
2.2.5 Error Functions	13
2.2.6 Activation Function	14
2.2.7 Overfitting & Underfitting	15
2.3 Convolutional Neural Networks	16
2.3.1 Convolutional Layer	17
2.3.2 Pooling Layer	17
2.3.3 Fully Connected Layer	18
2.4 Graph Convolutional Neural Network	18
2.4.1 Representation Learning on Graph	19
2.4.2 Graph Convolutional Neural Networks	21
2.4.3 Spatial-based GCN	22
3 Skeleton-Based Human Activity Recognition	24
3.1 Skeleton-Based HAR	25

3.1.1	Pose estimation	25
3.1.2	Handcrafted based methods	26
3.1.3	Deep Learning based methods	27
3.2	ST-GCN	29
3.2.1	Graph Construction	29
3.2.2	Spatial Graph convolution and temporal modeling TGCN	29
3.2.3	Architecture	31
3.3	MS-AAGCN	32
3.3.1	Adaptive Graph convolutional Layer	32
3.3.2	STC - Attention module	33
3.3.3	Architecture	34
3.4	CTR-GCN	35
3.4.1	Channel-wise Topology Refinement Graph Convolution	35
3.4.2	Architecture	37
3.5	DG-STGCN	37
3.5.1	DG-GCN	38
3.5.2	DG-TCN	39
3.5.3	Uniform sampling	39
3.5.4	Architecure	39
4	Temporal Action Segmentation	40
4.1	Related task	40
4.2	Core techniques	41
4.2.1	Frame-Wise Representations	41
4.2.2	Temporal and sequential modeling	42
4.2.3	Over-segmentation	42
4.3	Level of supervision	43
4.4	MS-GCN	43
4.4.1	TCN	44
4.4.2	Multi-stage models	44
4.4.3	Architecture	45
5	3DYogSeg: a new dataset for pose recognition and segmentation	46
5.1	Yoga datasets	46
5.2	Video Collection	47
5.3	Skeleton extraction	49
5.4	Data transformation	51
5.5	Training & test sets	52
5.6	Synthetic dataset pipeline	52

6	Benchmark and experiments	55
6.1	Human Activity Recognition	55
6.1.1	Implementation details	55
6.1.2	Results	55
6.1.3	Failure Cases	56
6.1.4	Comparison with 3DYoga90	56
6.2	Temporal Action Segmentation	59
6.2.1	Experimantal settings	59
6.2.2	Results	60
6.2.3	Hierarchical results	67
6.2.4	Comparison with 3DYoga90	70
7	Conclusions and future developments	72
	Bibliography	75

List of Tables

5.1	Yoga lesson statistics. For each lesson, the video id, the duration in minutes and seconds, the number of tags within the lesson, the average length of a tag in seconds, and what are the available raw data sources are reported.	47
5.2	Dataset examples of 3DYogSeg. For each pose indicated in the header we reported both the RGB image and the relative skeletal representation.	48
5.3	Values parameters for synthetic dataset	54
6.1	3DYogSeg activity recongition benchmarks. MS-AAGCN outperforms the other sota entworks on our dataset.	56
6.2	Results comparison when MS-AAGCN is trained and tested on a different combination of dataset. The experiments are made only on common classes between the two datasets.	58
6.3	Yoga lesson statistics. For each lesson the are reported the video id, the duration in minute and seconds, the number of tags within the lesson and the average lenght of a tag in seconds.	59
6.4	Correlation matrix between syntethic dataset paramters and metrics.	63
6.5	MS-GCN quantitative results with syntethic dataset built on 3DYogSeg.	65
6.6	Hierarchical results for MS-GCN trained on a dataset built with paremeters (False,5,20). The table shows how raising the hierarchy the performances improve	68
6.7	MS-GCN trained on 3Dyoga90	70

List of Figures

2.1	Biological neuron. Image from [6]	8
2.2	Artificial Neuron	8
2.3	Domain of the XOR operator	10
2.4	Multi-Layer Perceptron	10
3.1	On the left the vitruvian man to which is aligned the BlazePose detector, while on the right the architecture of the BlazePose pose tracker. Images from [42]	26
3.2	The spatial temporal graph of a skeleton sequence used in ST-GCN. Blue dots denote the body joints. The intra-body edges (light-blue) between body joints are defined based on the natural connections in human bodies. The inter-frame edges (green) connect the same joints between consecutive frames. Image from [1]	30
3.3	Partion strategies. (a) An example of input skeleton with joints connections in blue and receptive filed with red dashed dots. (b) Uni-labeling, (c) Distance partitioning and (d) Spatial configuration. Image from [1]	31
3.4	Illustration of the adaptive graph convolutional layer (AGCL). There are two kinds of graphs in each layer, i.e., B_k and C_k . The orange box indicates that the part is the parameter of the network and is updated during the training process. θ and ϕ are two embedding functions whose kernel size is (1×1) . K_v denotes the number of subsets. \oplus denotes the element-wise addition. \otimes denotes the matrix multiplication. G is the gate that controls the importance of the two kinds of graphs. The residual box (dotted line) is only needed when C_{in} is not the same as C_{out} . Image from [2].	34
3.5	Illustration of the STC-attention module. Three sub-modules are arranged sequentially in the orders of SAM, TAM and CAM. \otimes denotes the element-wise multiplication. \oplus denotes the element-wise addition. Image from [2].	34

3.6	Illustration of the MS-AAGCN basic block. Convs represents the spatial adaptive graph convolutional layer (AGCL), and Conv _t represents the temporal AGCL, both of which are followed by a BN layer and a ReLU layer. STC represents the STC-attention module. Moreover, a residual connection is added for each block. Image from [2].	35
3.7	Framework of the channel-wise topology refinement graph convolution proposed in [3]. The channel-wise topology modeling refines the trainable shared topology with inferred channel-specific correlations. The feature transformation aims at transforming input features into high-level representations. Eventually, the output feature is obtained by channel-wise aggregation. Image from [3].	36
3.8	(a) The basic block of CTR-GCN. (b) CTR-GC with correlation modeling function $\mathcal{M}_1(\cdot)$ or $\mathcal{M}_2(\cdot)$. Image from [3].	37
3.9	The architecture of the dynamic group GCN (DG-GCN). Image from [4].	38
3.10	The architecture of the dynamic group TCN (DG-TCN). ‘D’ indicates dilation. Image from [4].	39
4.1	(a) a long short-term memory network (LSTM), (b) a temporal convolutional neural network (TCN), (c) a multi-stage temporal convolutional neural network (MS-TCN), (d) a spatial-temporal graph convolutional neural network (STGCN), and (e) a multi-stage spatial-temporal graph convolutional neural network (MS-GCN). Image from [5].	44
5.1	Number of clips for each class.	49
5.2	Average duration of clips per class.	50
5.3	Skeleton comparison. On the left the original skeleton returned by BlazePose, on the right our skeleton where only relevant joints are kept.	52
5.4	Synthetic dataset pipeline. Starting two RGB video we extract the 3D skeletal sequences C_1 (red) and C_2 (blue). Then, given the last frame of C_1 and the first frame of C_2 we reconstruct the transition frames through linear interpolation.	53
6.1	MS-AAGCN confusion matrix	57
6.2	Comparison between eagle and chair poses	57
6.3	Comparison between table top and cat-cow poses	58
6.4	MS-GCN segment-wise F1@0.5 trained on 3DYogSeg. From left to right each graph represents how the metric varies for with respect to transition frames number with $apv=5,10$, and 25 respectively. . .	61

6.5	MS-GCN frame-wise accuracy trained on 3DYogSeg. From left to right each graph represents how the metric varies for with respect to transition frames number with apv=5,10, and 25 respectively. . .	61
6.6	Corpse and Supine twist pose	63
6.7	Low lunge and twisted low lunge pose	64
6.8	Cat-cow and one legged table pose	64
6.9	MS-GCN qualitative results trained on 3DYogSeg with ar=False, apv=5 and tf=20	66
6.10	MS-GCN segment-wise F1@0.5 trained on 3DYogSeg. From top to bottom level 1, level 2 and exercise-level	68
6.11	MS-GCN frame-wise accuracy trained on 3DYogSeg. From top to bottom level 1, level 2 and exercise-level	69

Chapter 1

Introduction

In recent years, the efficiency and power of deep learning models have significantly increased due to the vast amount of data available. While the concept of neural networks dates back to the 1960s, it was not until the era of Big Data that algorithms capable of achieving human-comparable or even superior performance in terms of speed and accuracy became feasible.

The Perceptron, developed by Rosenblatt in 1957, was the first mathematical model to simulate a biological neuron. It is a binary linear classifier that divides samples into two half-planes based on linear comparability. However, to overcome the limitations of linear classification, artificial neural networks (ANNs) were introduced. ANNs consist of multiple neurons organized into layers, allowing for the learning of complex relationships between inputs and outputs. The training phase involves a feedforward pass, where predictions are generated, followed by backpropagation to update the weights based on prediction errors.

Convolutional Neural Networks (CNNs) were developed to process non-one-dimensional data such as images while preserving spatial information. In CNNs, neurons are organized in three dimensions: width, height, and depth. Convolutional layers apply filters to input data using the convolution operator to generate activation maps. The hierarchical features extracted by successive convolutional layers capture information from increasingly distant pixels in the original image.

Graph Neural Networks (GNNs) have gained popularity for processing graph-structured data, which is prevalent in various fields like social sciences and chemistry. Unlike images, graphs are non-Euclidean data structures and cannot directly apply conventional convolutions. GNNs address this challenge by iteratively exchanging information between nodes through aggregation and combination steps.

The application of Graph Convolutional Networks (GCNs) extends to various domains and tasks. In our work, we focus on their use in Skeleton-based Human Action Recognition (HAR) and Temporal Action Segmentation (TAS) tasks. HAR involves identifying and categorizing activities based on skeletal data, while TAS

entails segmenting temporally untrimmed video sequences and labeling each segment with predefined action labels. Both tasks find applications in video surveillance, human-computer interaction, and beyond.

Sports, like many other fields, are increasingly utilizing deep learning analysis methodologies to enhance athlete performance. At the professional level, data analysis is employed to optimize training loads and streamline scouting processes. Moreover, the availability of lightweight and efficient models on mobile devices has extended these capabilities to a wider range of users, including amateur athletes seeking to enhance their training experiences. Particularly, during lockdown periods due to COVID-19, there has been a surge in the popularity of home workout applications, many of which integrate AI to offer personalized feedback and guidance to users, simulating the experience of having a personal trainer.

Yoga, originating in India, is one such practice that has gained popularity as a home workout routine. It blends physical and spiritual well-being through various poses, known as 'asanas', which vary in difficulty. For beginners, learning these poses can be challenging. Automated AI tools that recognize yoga poses in videos offer two main benefits: they aid practitioners in memorizing poses and provide real-time feedback on posture correctness. The growing interest in AI applications for sports has led to the availability of more datasets in the yoga sector. However, many existing datasets have limitations, such as a restricted number of asanas or focusing solely on pose recognition in images rather than videos.

Our research has made significant contributions to the field of Human Action Recognition (HAR) by presenting a new dataset called 3DYogSeg. Unlike existing datasets, ours features both RGB videos and skeletal sequences and includes a comprehensive collection of 58 yoga asanas. Additionally, our study stands out as the first to delve into the segmentation of yoga videos, offering a pioneering method to repurpose datasets for Temporal Action Segmentation (TAS) from those originally designed for HAR. Furthermore, to evaluate the dataset's quality, we conducted extensive experiments on both the HAR and TAS tasks using state-of-the-art networks.

1.1 Objectives and motivations for the project

Our work focuses on addressing the tasks of Skeleton-based Human Activity Recognition (HAR) and Temporal Action Segmentation (TAS) within the domain of Yoga. We propose a novel dataset and a pipeline to create pseudo-lessons from videos containing single exercises.

The key aspects of these tasks involve the utilization of graphs, representing data with a non-Euclidean structure, and the handling of skeletal sequences, which contain both spatial and temporal information. Graphs are chosen due to their

ability to represent the human body as interconnected nodes, reflecting physical relationships such as bone connections and interactions between body parts in various poses. Additionally, skeletal sequences are represented as spatial-temporal graphs, where spatial features are captured within individual frames and temporal features are encoded in relations between frames.

The motivation for exploring these tasks in the context of yoga arises from the lack of datasets specifically designed for segmenting yoga video lessons based on timestamps. We believe that developing a tool capable of accurately segmenting video lessons can enhance the training experience for practitioners. For instance, providing users with information about the sequence and duration of exercises beforehand can help them prepare their training space effectively. Moreover, offering real-time feedback on pose execution during practice can improve technique. Additionally, categorizing video lessons into different levels of difficulty based on exercises, duration, and sequence can guide practitioners in their training progression.

Collecting a bespoke dataset for the TAS task, however, is time-consuming due to the diverse range of yoga poses. To overcome this challenge, we initially gathered videos from YouTube containing single yoga poses and extracted skeletal sequences. Subsequently, we evaluated the quality of our HAR dataset, 3DYogSeg, through benchmarking on four different Graph Convolutional Network (GCN)-based networks. We then developed a pipeline to concatenate clips of different poses using linear interpolation, thus creating pseudo-lessons. Using this approach, we constructed datasets for the TAS task by varying parameters such as the number of activities per video, transition frames between clips, and the inclusion of repeated exercises. Finally, we conducted segmentation experiments on real yoga video lessons obtained from YouTube. To demonstrate the versatility of our pipeline, we generated pseudo-lessons using another recent dataset, 3Dyoga90, and evaluated the performance of the MS-GCN network trained on this dataset.

1.2 Contributions

With this thesis, we introduce 3DYogSeg, a novel dataset containing 3D skeletal sequences for pose recognition. Our dataset comprises YouTube videos, identified using pose names in Italian, English, and Sanskrit (the official language of Yoga). We collected a total of 2115 videos, representing 58 distinct yoga asanas. On average, each exercise includes 34 clips, with an average duration of 12.91 seconds per clip. We extracted 3D skeletal sequences from these videos using the BlazePose model, which were further refined to include 15 joints. Compared to existing datasets in the Yoga domain, 3DYogSeg stands out as one of the first to provide publicly available skeletal sequences extracted from videos, rather than static

images. Moreover, our dataset exclusively contains clips showcasing the yoga poses, thereby minimizing extraneous movements such as preparation or exit phases. This deliberate curation choice simplifies the Human Action Recognition (HAR) task, as models are not exposed to patterns from similar asanas that might be performed in sequence. Moreover, in this thesis, we propose a pipeline to convert 3DYogSeg into a Temporal Action Segmentation (TAS) dataset. Leveraging linear interpolation, we demonstrate a method to concatenate two skeletal sequences, C_1 and C_2 , into a new sequence, C_3 , by iteratively adding transition frames. This approach allows us to create pseudo-video lessons characterized by parameters such as the number of activities per video (apv), the number of transition frames between clips (tf), and the presence or absence of repeated exercises (ar). We conducted experiments to establish benchmarks for the HAR task using four Graph Convolutional Network (GCN)-based networks [1, 2, 3, 4]. Despite the challenge posed by the task due to the high similarity between some poses, our results illustrate how our dataset can serve as an effective training resource for GCNs in yoga pose recognition. For the TAS task, we trained the MS-GCN [5] network on datasets generated using our proposed pipeline with various parameter settings. Both quantitative and qualitative analyses demonstrate that our methodology offers a strong baseline and is significantly less time-consuming than creating a bespoke dataset for segmentation. To our knowledge, this is the first study to address the segmentation of yoga video lessons. Finally, we compared the performance of our dataset with that of the 3Dyoga90 dataset, which shares similar construction methods, across both HAR and TAS tasks.

In summary, our work offers the following contributions:

- We introduce a novel dataset, 3DYogSeg, tailored for the Human Action Recognition (HAR) task within the domain of Yoga. This dataset comprises skeletal sequences extracted from YouTube videos, providing a valuable resource for researchers and practitioners interested in yoga pose recognition.
- We present a robust pipeline to convert a skeletal sequence dataset designed for HAR into a dataset suitable for the Temporal Action Segmentation (TAS) task. This methodology enables the generation of pseudo-video lessons, bypassing the need for labor-intensive dataset creation specific to segmentation.
- Our work includes benchmarks for the HAR task using four Graph Convolutional Network (GCN)-based networks. These benchmarks serve as a valuable reference point for evaluating the performance of GCNs in recognizing yoga poses using our dataset.
- We provide both quantitative and qualitative results of using the pseudo-lessons dataset for segmenting real video lessons. Our findings highlight the

efficacy of our methodology in segmenting yoga video content accurately and efficiently.

- We compare the performance of our 3DYogSeg dataset with that of 3Dyoga90, which shares similarities in construction methods. This comparison sheds light on the strengths and weaknesses of each dataset, providing insights for future research endeavors.

Overall, our work contributes to advancing research in both yoga pose recognition and temporal action segmentation, offering valuable resources and methodologies for researchers in these domains.

1.3 Thesis structure

The thesis is organized as follows:

- **DeepLearning: from Perceptron to Graph Convolution Neural Networks:** This chapter provides a historical overview of Deep Learning, tracing its evolution from the perceptron to graph convolutional neural networks. It introduces fundamental concepts essential for understanding our work.
- **Skeleton-Based Human Activity Recognition:** Chapter 3 introduces the Human Action Recognition task, with a focus on strategies based on skeletal sequences. It covers pose estimation techniques and the Blaze Pose architecture. Additionally, it describes four networks used for benchmarking on our dataset: ST-GCN, MS-AAGCN, CTR-GCN, and DG-STGCN.
- **Temporal Action Segmentation:** Chapter 4 delves into the Temporal Action Segmentation task, providing an overview of supervision methodologies, core techniques, and related tasks. It introduces the MS-GCN network utilized in our experiments.
- **3DYogSeg: a new dataset for 3D yoga pose recognition and segmentation:** The fifth chapter details the creation of the 3DYogSeg dataset and outlines the pipeline used to adapt it for Temporal Action Segmentation. It covers previous yoga datasets, video collection, skeleton extraction, data transformation, and dataset conversion.
- **Benchmark and experiments:** Chapter 6 presents benchmarks for activity recognition obtained using the aforementioned networks. It also showcases quantitative and qualitative results for the Temporal Action Segmentation task. Additionally, a comparison with similar existing yoga datasets is provided.

- **Conclusions and future developments:** The final chapter summarizes the key findings of our work and discusses potential future directions and areas for further research.

Chapter 2

DeepLearning: from Perceptron to Graph Convolution Neural Networks

This chapter presents the basic concepts of artificial neural networks and deep learning. Starting with the first form of a machine learning algorithm, the Perceptron, we will then go on to introduce the fundamental notions behind Neural Networks, and then discuss Convolutional Neural Networks. The last part of the chapter will focus on the use of non-Euclidean data and the networks used for their analysis, Graph Neural Networks, which are closely related to our work.

2.1 The Perceptron

The artificial neuron, called Perceptron, is a mathematical model that aims to replicate the behavior of the biological neuron (figure 2.1).

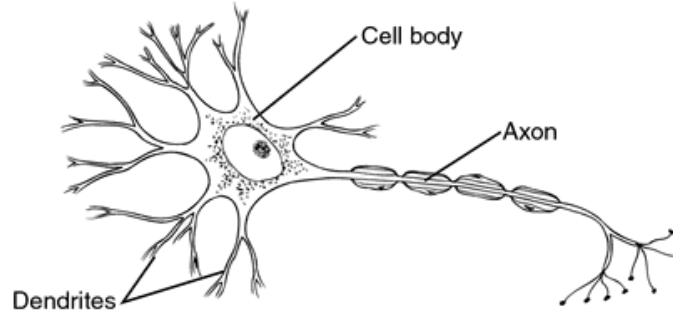


Figure 2.1: Biological neuron. Image from [6]

First introduced by Rosenblatt in 1957 [7], the perceptron receives numerical values instead of electrical signals. This input is then processed through the ‘weights’ obtaining a weighted sum also called the *activation value*. In the biological neuron, the processed input is then propagated through the axon if it reaches a minimum power. In the perceptron, this verification is done via the activation function. As in the human brain, the output of one neuron becomes the input of the next. The all process is represented by the figure below.

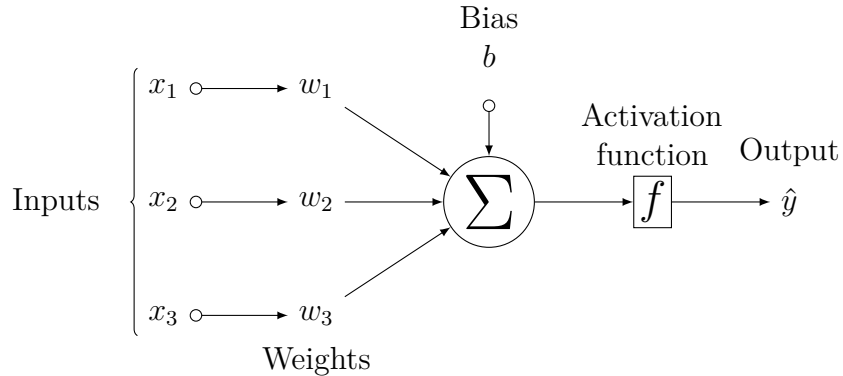


Figure 2.2: Artificial Neuron. Image from [8]

At a mathematical level, given an input x_i , with $i = 1, \dots, N$, the perceptron produces the output:

$$\hat{y} = g\left(w_0 + \sum_{i=1}^N x_i w_i\right) \quad (2.1)$$

In 2.1 w_i , for $i = 1, \dots, N$, are the weights, the activation values or weighed sum is represented by $\sum_{i=1}^N x_i w_i$, the bias b is equal to $w_0 * -1$ and the activation function, that could be a *step function*, a *linear function* or a *sigmoid* is g .

In case the activation function is a step function, the output can be rewritten as:

$$\hat{y}(x_1, \dots, x_N) = \begin{cases} +1, & \text{if } w_0 + w_1x_1 + \dots + w_Nx_N \geq 0. \\ -1, & \text{otherwise.} \end{cases} \quad (2.2)$$

As can be understood from 2.2, the perceptron divides space into two halves being a *Linear Binary Classifier*.

The weights, which are learned during training, are the component that causes the output to fall into the positive or negative semi-plane. During learning, the weights are updated according to this scheme:

- At the beginning, they are set with random values.
- Iteratively, given a set of desired inputs and outputs, the weights are maintained if the prediction is correct; otherwise, they are modified according to the *Hebbian Rule* (equation 2.3) where η is the learning rate, \hat{y} is the predicted output, and y is the desired output.

$$w_i = w_i + \eta(y - \hat{y})x_i \quad (2.3)$$

As just described, a single perceptron is a linear binary classifier, so its main limitation is the inability to solve nonlinear problems such as XOR (figure 2.3) due to the convergence theorem [9]: *the perceptron learning rule will converge to a set of weights that correctly represents the examples, as long as the examples represent a "linearly separable" function and η is sufficiently small.* To be able to separate nonlinear domains, it is possible to combine multiple perceptrons into neural networks.

2.2 From neuron to "brain"

2.2.1 Multi-layer perceptron or Feed Forward Neural Network

A Multi-layer Perceptron, or MLP, is a set of neurons organized according to topology. In fact, a neural network is divided into layers, i.e., neurons placed at the same distance from the input. The neural network (NN) can be divided into 3 main parts:

- *Input layer*: the neurons that receive the input;
- *Output layer*: the neurons that process the final result;

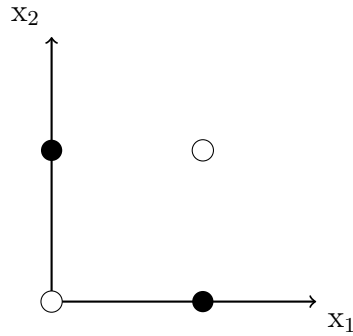


Figure 2.3: Non-linearly separable domain of the logical operator XOR. Image from [8]

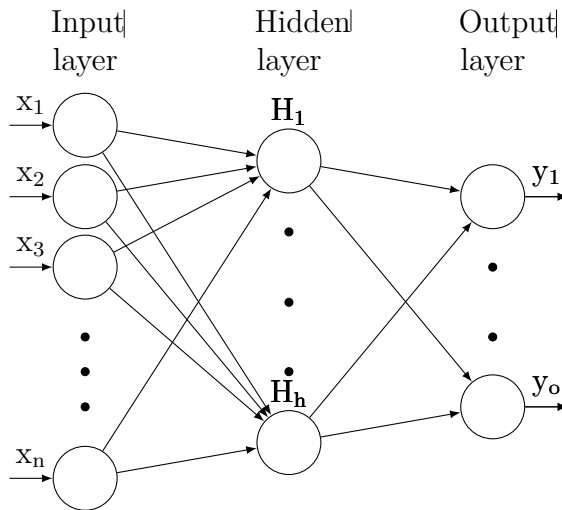


Figure 2.4: Artificial Neural Network with a single hidden layer and multiple outputs. Image from [8]

- *Hidden layers:* layers of neurons that process data between the input and output layers;

Figure 2.4 shows an example of a MLP with a single hidden layer, whose neurons are indicated with H_i , for $i = 1, \dots, h$, n input neurons and o output neurons.

In an MLP, each output \hat{y} can be defined as:

$$\hat{y} = g \left(\sum_{j=1}^J W_j + h \left(\sum_{i=1}^I w_{ji} x_i \right) \right) \quad (2.4)$$

where W_j is the weight for the connection of neuron j of the hidden layer to the output, and w_{ji} is the weight of input i that goes from neuron i to neuron j of the hidden layer. The functions g and h represent the activation functions which must have two important properties: *i*) not be linear so as to allow the generation of non-linear mappings between input and output, *ii*) they must be differentiable, a necessary condition for the learning strategy defined as **backpropagation** (section 2.2.2) and *iii*) they should be bounded in order to have a more stable gradient.

The propagation from the input to the output layer is defined as the *forwarding process*, from which the name Feed-Forward neural network is taken.

2.2.2 Backpropagation

As for the perceptron, in which the weights are updated according to 2.3, also in FNNs the learning rule, called backpropagation, aims to minimize the error or loss.

The basic idea is that the network learns from a set of inputs and the relative desired outputs based on the difference between the latter and the predictions produced. The error is then propagated back through the network to indicate which weights need to be changed.

Once the feedforward step is finished, the network generates outputs \hat{y} which are compared with the real outputs y . The algorithm then calculates the derivatives of the error \mathcal{L} , which indicates how much each weight impacts the error. Subsequently, through the update step, the weights are modified proportionally to the negative of the gradient as indicated in the equation 2.5, called the *gradient descent rule*.

$$\begin{aligned} w^{t+1} &\leftarrow w^t + \Delta w \\ \Delta w &= -\eta \cdot \frac{\delta \mathcal{L}}{\delta w} \end{aligned} \tag{2.5}$$

By iteratively repeating this process with multiple inputs, weights assume the values that minimize the error.

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) \tag{2.6}$$

As indicated by 2.5, the Δw is calculated separately for each w , so the updating of each weight depends only on its importance in the error.

2.2.3 Learning Rate

The learning rate, η in 2.5, is a hyperparameter that controls the step size at which the weights are updated during training. A higher learning rate leads to larger updates, resulting in faster convergence but can also cause instability or

overshooting. On the contrary, a lower learning rate leads to slower convergence but can result in stable training and better generalization ability. During training, various strategies can be used to determine the appropriate learning rate:

- *Fixed learning rate*: use a constant learning rate throughout the training. It is the simplest and easiest approach to implement but requires precise fine-tuning.
- *Learning rate schedules*: the learning rate is modified according to a linear or exponential schedule. This approach can strike a good balance between fast initial convergence and stabilization.
- *Adaptive learning rate*: some algorithms such as AdaGrad [10] and Adam [11] adapt the learning rate based on the feedback from the optimization process, taking into account previous gradients and second moments.

The choice of the learning rate or the strategy is often considered a fundamental and difficult step given its dependence on the dataset, model, and objectives.

2.2.4 Weight Initialization

Another fundamental aspect of training is the initialization of the weights. Incorrect initialization can lead to vanishing gradient, i.e., the gradient disappears and therefore the parameters are no longer updated, or exploding gradient, which can lead to divergence in the minimization of the loss. Two very common techniques as initialization strategies are:

- *Xavier Initialization* [12]: Through Xavier initialization, the variance of the gradient between the various layers of the network is kept almost constant. Once the number of input connections, n_{in} , and output connections, n_{out} , is fixed for each layer, the weights can be initialized via a uniform distribution (eq. 2.7) or via normalization with $\mu = 0$ and σ which depends on n_{in} and n_{out} (eq. 2.8).

$$\mathbf{W} \sim \mathcal{U}\left(-\frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}}\right) \quad (2.7)$$

$$\mathbf{W} \sim \mathcal{N}\left(0, \frac{2}{n_{in} + n_{out}}\right) \quad (2.8)$$

- *He initialization* [13]: Designed especially for activation functions that have rectified linear units (ReLU) [14] or its variations. In order to properly account for the features of ReLU activations, it sets the initial weights bigger than

those of Xavier initialization. In this case, the weight distribution follows a normal with zero mean and variance equal to $\frac{2}{n_{in}}$ (eq. 2.9).

$$\mathcal{W} \sim \mathcal{N}\left(0, \frac{2}{n_{in}}\right) \quad (2.9)$$

2.2.5 Error Functions

The error function, or loss function, quantifies the difference between the predicted output and the actual targets, providing a measure of how well the model performs on the training data. During training, the model aims to optimize the parameters so that the predictions are as similar as possible to the desired outputs. In general, we can represent the loss as $\mathcal{L}(\hat{y}, y)$, where \hat{y} is the prediction and y is the desired output. Below, we describe two examples of loss functions used for classification and regression tasks.

Mean Squared Error (MSE) The mean squared error, or MSE, is a loss used for the regression task where the target variable $Y \in \mathbb{R}$. MSE calculates the mean square of the differences between \hat{y}_i and y_i :

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N (\hat{y}_i - y_i)^2 \quad (2.10)$$

Here, N is the number of samples in the dataset, and \hat{y}_i and y_i represent the prediction and ground truth of the i -th sample respectively. MSE penalizes large deviations more than small ones, making it sensitive to outliers.

Cross-Entropy Loss Cross-entropy loss is one of the most common losses used in classification tasks such as Skeleton Based Human Activity Recognition (chapter 3), where the target variable Y assumes only discrete values.

$$\text{CrossEntropyLoss} = -\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^C y_{ij} \log(\hat{y}_{ij}) \quad (2.11)$$

In this equation, N is the number of training samples, C is the number of classes, \hat{y}_{ij} is the predicted probability for the i -th sample to belong to class c_j , and y_{ij} is the real probability that x_i belongs to c_j . This loss function encourages the model to assign higher probabilities to the correct classes. Cross-entropy loss is commonly used with the softmax activation function in the output layer of multiclass classification tasks.

2.2.6 Activation Function

Activation functions, as described in section 2.2.1, are the components of neural networks that enable the generation of mappings between input and output. Through these mappings, the model can learn complex relationships and representations of the input. As we anticipated in section 2.2.1 they should be bounded, differentiable and non-linear. Below, we describe some of the most common activation functions used.

Sigmoid Activation Function The sigmoid function, also known as the logistic function, squashes the input values into the range (0, 1). It is given by the formula:

$$\textit{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \quad (2.12)$$

The sigmoid function is commonly used in the output layer of binary classification tasks, where it interprets the output as a probability of belonging to one of the two classes.

ReLU (Rectified Linear Unit) Activation Function [14] The ReLU activation function is defined as the positive part of its argument, returning zero for negative inputs and the input value for positive inputs. It is given by the formula:

$$\textit{ReLU}(x) = \max(0, x) \quad (2.13)$$

ReLU is one of the most widely used activation functions in deep learning due to its simplicity and effectiveness. It helps alleviate the vanishing gradient problem and accelerates convergence in deep neural networks.

Softmax Activation Function The softmax function is used in the output layer of multi-class classification tasks to convert raw scores or logits into probability distributions over multiple classes. It computes the probability of each class as a normalized exponential function of the input values. The formula for softmax is given by:

$$\textit{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^C e^{x_j}} \quad (2.14)$$

Activation functions play a crucial role in determining the learning capacity of neural networks. Choosing an appropriate activation function depends on the specific characteristics of the task and the network architecture.

2.2.7 Overfitting & Underfitting

During the training of a network, the available data is usually divided into three subsets: *training data*, *validation data*, and *test data*. The training data is the subset through which the model learns, the validation set is the set of data used to verify performance and fine-tune the hyperparameters, while the test set simulates the use of the model in a real context where the data was never seen by the network. In the testing phase, to evaluate the generalization ability compared to training, the performances in training and testing are compared. If there is a very significant drop between training and testing performance or both are very low, it means that you have stumbled upon overfitting or underfitting, respectively.

Overfitting occurs when the network has adapted too much to the training data, also capturing noise and outliers, but without being able to generalize to new samples. This can happen when the model is too complex compared to the data available to train the network.

On the other hand, underfitting occurs when the model is too simple or the amount of data available is too little compared to the required task.

If in the first case we have high variability when training on different datasets, in underfitting the model presents a bias, i.e., it appears incapable of capturing the real relationships that exist between input and output. To obtain an optimal model that does not present either of the two problems, various methods such as Regularization, Dropout, Early stopping can be used.

Regularization Regularization involves the insertion of a term, called regularization, within the loss function, preventing the model from becoming too complex. In general, the loss function therefore becomes:

$$\mathcal{L}_{reg}(\theta) = \mathcal{L}(\theta) + \lambda \cdot \Omega(\theta) \quad (2.15)$$

where θ are the model parameters, $\mathcal{L}(\theta)$ is the original loss function, $\Omega(\theta)$ is the regularization term, and λ adjusts the intensity of the latter. This strategy is called *weight decay* and two examples are Lasso regularization, called L1, and Ridge regularization, called L2.

Lasso Regularization adds a regularization term equal to the sum of the absolute values of the parameters:

$$\text{Lasso Regularization term: } \sum_{i=1}^n |\theta_i| \quad (2.16)$$

Through this strategy, the model is pushed to have parameters with high sparsity, causing some of them to vanish.

Ridge regularization, on the other hand, adds a term proportional to the squared L2 norm of the parameters. The penalty term is given by:

$$\text{Ridge Regularization term: } \sum_{i=1}^n \theta_i^2 \quad (2.17)$$

Unlike Lasso, Ridge penalizes parameters that are too high without bringing them completely to zero.

Dropout Another technique that can be used to prevent the model from becoming too complex is Dropout. The idea, presented in [15], involves randomly deactivating some neurons at each training iteration. This reduces the interdependence that can be generated between the neurons, forcing the network to generalize better and be more robust. The neurons of the hidden layers are deactivated according to a Bernoulli distribution with probability p , called the *dropout rate*. Specifically, at each training iteration, a mask is generated according to this distribution, which is then applied to the activation values. This sets the activation values of the neurons to be deactivated to 0. The modified activation values are then propagated up to the output layer, allowing them to be included in the calculation of the loss and in the backpropagation of the gradients. At inference time, dropout is disabled so that the entire network can contribute to generating predictions. The choice of the value to assign to the drop out rate p is arbitrary and can be optimized through fine-tuning. However, in many cases a value of 0.5 has proven to be a valid solution.

Early Stopping Early Stopping is a third option to prevent overfitting. In this case, during training, the performance on the validation set is monitored iteratively. The moment these performances begin to worsen, indicating that further training could lead to overfitting, training is stopped. However, this strategy must be carefully balanced, as early stopping too anticipated can lead to the opposite problem, underfitting, resulting in a model still incapable of recognizing the real patterns between input and output.

2.3 Convolutional Neural Networks

A particular family of feedforward neural networks (FFNNs) are convolutional neural networks (CNNs), introduced for the first time in [16] and designed to better manage images. In fact, the layers of a CNN do not have neurons placed on a one-dimensional layer as represented in figure 2.4, but are organized in width (W), height (H), and depth (C). In this way, CNNs have two advantages compared to classic FNNs: they have a reduced number of parameters, and they manage to preserve the spatial information of the input.

2.3.1 Convolutional Layer

The convolutional layer is the fundamental part in which most of the computation is carried out within a convolutional neural network (CNN).

Each convolutional layer (ConvLayer) operates on the input through a set of learnable filters of size $F \times F \times C$, where F indicates the receptive fields of the filter and C the depth of the input. At each forward pass, the filters are spatially slid over the entire input. Through the convolution operation, the filter obtains a 2-D *activation map*. As can be understood from equation 2.18, the convolution operation replaces the elements of the input on which the filter is applied with the dot product between the latter and the filter parameters.

$$\text{conv}(p, q) = \sum_i \sum_j F_{(i,j)} x_{(i+p, i+q)} \quad (2.18)$$

Ideally, each filter extracts a local feature, and for this reason, a ConvLayer can have multiple filters. Consequently, the final output will also be three-dimensional, with depth C_{out} , where C_{out} is the number of filters in the layer. The height and width of the activation maps can be managed via two other parameters: the *padding* (P), which indicates how many zeros are inserted around the edge of the original input, and the *stride* (S), or the number of pixels by which the filter must shift between one convolution and another. The final dimensions will therefore be $H_{\text{out}} \times W_{\text{out}} \times C_{\text{out}}$, where H_{out} and W_{out} , for an input $H \times W \times C$, are:

$$H_{\text{out}} = \left\lfloor \frac{H + 2 \times (P) - F}{(S)} + 1 \right\rfloor \quad (2.19)$$

$$W_{\text{out}} = \left\lfloor \frac{W + 2 \times (P) - F}{(S)} + 1 \right\rfloor \quad (2.20)$$

2.3.2 Pooling Layer

The second layer that characterizes a convolutional neural network (CNN) is the *pooling layer* (PoolLayer). Often inserted between two convolutional layers, the pooling layer is exploited to perform downsampling on the generated activation maps.

The pooling layer operates via a filter similar to a ConvLayer. This time, however, the operator applied is not the convolution, but rather the Max operator or the Average operator. In the first case, for example, if the filter is 2×2 in size, for each passage of the filter, the four elements involved are replaced at the output by a single element containing the maximum value of the four. In the case of the average operator, the same four elements are replaced with a new element containing the average of the four values.

2.3.3 Fully Connected Layer

Downstream of the ConvLayer and the PoolLayer, one or more fully connected layers (FCL), i.e. the standard layers of a multi-layer perceptron, are inserted into a CNN, with the softmax as activation function. The purpose of this layer is to generate the probabilities of membership of each class for the general input based on the high-level features extracted from the convolutional layers and from the pooling layers.

In conclusion, a CNN is made up of the three layers just described. As written at the beginning of the section, CNNs, in addition to maintaining spatial information, use fewer parameters than a FFNN for the same input. This is possible due to the different topology of the two networks. In an FFNN, in fact, each neuron of the l layer is connected with all the neurons of the $l - 1$ layer, while in a CNN, a neuron at the l layer receives only a small part of the input of the $l - 1$ layer based on the size of the latter's filters.

Among the most famous CNNs, we find LeNet [16], AlexNet [17], and ResNet [18], the first network capable of improving the human error rate on ImageNet [19].

2.4 Graph Convolutional Neural Network

If convolutional neural networks (CNNs) were introduced for a type of data, images, for which feedforward neural networks (FFNNs) are inefficient, in the same way, Gori et al. [20] and subsequently Scarselli et al. [21] created Graph Neural Networks (GNNs), a family of neural networks suitable for managing data with non-Euclidean structures, *graphs*.

Graphs are used in many fields, from social sciences for the analysis of social networks to chemistry for the representation of molecules. Each graph can contain an unordered number of nodes, and therefore operators such as convolution, which are shift-invariant and designed for the regularity of structured data, are no longer directly applicable.

The idea that Gori et al. [20] and a year later Scarselli et al. [21] based GNNs on is that each node represents an object or concept, while the relationships are represented by the edges. Starting from the works cited above, Bruna et al. [22] introduced the concept of convolution on GNNs by proposing a spatial construction, where some properties of classical convolution are generalized on the graphs, and a spectral construction, which is designed based on the properties of convolution in the Fourier domain. If the second has shown limitations, also investigated in [23] and [24], spatial construction has proven to be a valid option, so much so that it is the technique on which the networks that are currently applied in human activity recognition and more are based.

In this section, we will describe how information is represented in graphs, then we will illustrate in more depth the theory behind GNNs, and finally, we will delve into spatial-based convolution.

2.4.1 Representation Learning on Graph

The main problem of deep learning applied to graphs is to identify a strategy for incorporating information on their structure. It is therefore necessary to identify a method to create an embedding of a node, or a (sub)graph, in which to insert the features of interest. In their work, Hamilton et al. [25] report an overview of some of the most recent strategies such as the shallow embedding approach and the more performing deep encoding method.

Before delving into the description of the strategies, it is necessary to define what a graph is at a mathematical level. A graph $\mathcal{G} = (V, E)$ is composed of a set of nodes $V = \{v_i, i = 1, \dots, N\}$ which are connected to each other via the edge set $E = \{e_{ij} = (v_i, v_j) \mid v_i \text{ and } v_j \in \mathcal{V}\}$. For each node v_i , we can define the set of its neighbors $N(v_i) = \{v_j \in V \mid \exists e_{ij} \in E\}$. If the edges have a direction, the graph is called directed. For each graph $\mathcal{G} = (V, E)$, with $|V| = n$ and $|E| = m$.

Shallow Embedding

Node embedding aims to map the position of the node v_i in the graph and its neighborhood $N(v_i)$ into a low-dimensional vector. The obtained embeddings can either be seen as projections of the nodes into a latent space where the edges are represented by geometric relations [26].

Most algorithms that work with node embedding are based on what is called *shallow embedding*, where an encoder and a decoder are combined to generate a node similarity measure. In particular, starting from an embedding function (equation 2.21), which maps each node into a vector $z_i \in \mathbb{R}^d$, a decoding function (equation 2.22) maps each pair of embeddings (z_i, z_j) into a measure of similarity. The target function in these contexts is defined by the equation 2.23:

$$ENC : V \rightarrow \mathbb{R}^d \tag{2.21}$$

$$DEC : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}^+ \tag{2.22}$$

$$\mathcal{L} = \sum_{(v_i, v_j \in V)} \ell(DEC(z_i, z_j) - s_{\mathcal{G}}(v_i, v_j)) \tag{2.23}$$

where ℓ evaluates the quality of the pairwise reconstruction.

If usually the encoding function is simply an ‘embedding lookup’, $ENC = Zv_i$, where $Z \in \mathbb{R}^{d \times |V|}$ is a matrix containing the embeddings of each node and $v_i \in \mathbb{I}_V$ is a one-hot indicator vector that identifies the column of Z corresponding to v_i , the decoding function varies based on the approach used:

- *Factorization-based approach*: The decoding function exploits matrix factorization. This family includes the Laplacian eigenmaps [27] and the inner product method [28, 29, 30].
- *Random walk approach*: This method aims to generate similar embeddings for two nodes if they co-occur in a short random walk. The difference with the factorization-based approach, methods like DeepWalk [31] and node2vec [32] use a stochastic node similarity.

The approaches mentioned above, however, lead to some disadvantages:

1. There are no shared parameters between nodes in the encoder and this is both computationally and statistically inefficient;
2. The node attributes are not exploited as the embeddings only consider the position of v_i and $N(v_i)$;
3. They are *transductive* [33], the embeddings are generated only for the nodes seen in training. This can be problematic in scenarios where graphs can evolve.

Deep encoding approaches

Deep encoding approaches, unlike the methods just described, replace the encoder with a deep neural network. These autoencoders generate node embeddings by aggregating neighborhood information. Furthermore, when provided, node feature vectors can be exploited directly. These methods are called convolutional as the node is represented as a function of the neighbors similarly to what is done in CNNs through the receptive fields of the filters. An algorithm that falls into this family involves two main steps:

1. *AGGREGATION*: during the encoding phase, at the k -th iteration for each node v in V an encoding $h_{N(v)}^k$ is generated as:

$$h_v^k = \text{AGG}_k(\{h_u^{k-1}, \forall u \in N(v)\}) \quad (2.24)$$

where h_u^{k-1} is the embedding of node u in the previous loop. For $k = 0$ the embedding $h_v^0 = x_v$, where x_v is the node feature vector.

2. *COMBINATION*: once the aggregation has been carried out, the final embedding for node v at iteration k is defined as:

$$h_v^k \leftarrow \sigma(W^k \cdot \text{COMB}(h_v^{k-1}, h_{N(v)}^k)) \quad (2.25)$$

where $\{W^k, \forall k \in [1, K]\}$ is the set of weight matrices for each iteration and σ an activation function.

The entire process can be repeated K times in an arbitrary manner.

Among the networks that follow this method there are the GNNs of which some examples are the GraphSAGE [33] and the Graph Convolutional Neural Networks [34, 35, 36] which are distinguished from each other by the operators used in the aggregation phase.

In summary, the deep encoding approach overcomes the three main limitations of shallow encoding as there is a split division of the parameters, W^k , and uses the same aggregation function to generate node embeddings. This ensures a reduction in the parameters to be learned, provides regularization, and allows generating embeddings for nodes not observed in training.

2.4.2 Graph Convolutional Neural Networks

Graph Neural Networks, or GNNs, are a family of networks that exploit the deep encoding approach using a message-passing algorithm as an aggregation function [8]. The first to formulate a version of GNN were Gori et al. [20], whose work was further developed by Scarselli et al. [21]. In their model, the network had to learn a state embedding h_v that incorporated the information from $N(v)$. This state embedding is calculated as:

$$h_v = \sum_{u \in N(v)} f(h_u^{t-1}, x_v, x_u) \quad (2.26)$$

where f is a differentiable mapping that generates a single vector from a set of vectors, while x_v and x_u are the feature vectors of v and u respectively. In [21], the function is defined as a *contraction map*, aiming to bring the nodes closer in the latent space into which they are projected. After an arbitrary number of iterations, the final embedding is defined as:

$$z_v = g(h_v^k) \quad (2.27)$$

where g is also a differentiable function. During training, the network learns the parameters of the two mapping functions f and g , and to do this, it uses a gradient-descent-based approach.

The iterative nature of this type of network, hence the name RecGNN, entails some limitations:

- Learning is inefficient as iterations continue until a point of convergence is reached.
- The network is unable to learn a hierarchy in the features, which is very useful in CNNs, as at each iteration the same set of parameters is used.

Although the GNNs just described operate directly on the feature matrix $X \in \mathbb{R}^{N \times d}$, the information contained in the edge feature vectors is not exploited.

To deal with the convergence problem, networks such as GatedGNN [37] exploit GRUs as an aggregation operator by creating embeddings such as:

$$h_v = \text{GRU}(h_v^{t-1}, \sum_{u \in N(v)} W h_u^{t-1}) \quad (2.28)$$

GCNs or Graph Convolutional Neural Networks [34, 35, 36], on the other hand, deal with the problem of hierarchical features. In fact, these networks use a fixed number of layers, each of which has its own set of weights. As already mentioned, two categories of convolution have been introduced, spectral-based and spatial-based. The latter is explored in greater detail in the following section.

2.4.3 Spatial-based GCN

Given a graph $G = (V, E)$ with N nodes, a spatial-based graph convolutional network (GCN) attempts to mimic what a classical convolutional neural network (CNN) does on an image. On the other hand, a grid of $\sqrt{N} \times \sqrt{N}$ pixels is nothing more than a graph with a well-defined structure, and the convolution operated through the filter is an aggregation function which, given a central node and its neighbors, maps the first with a new value containing the second's information. However, we cannot apply the same convolution as in CNNs in GCNs since in a graph, for each node v , we cannot be sure that $N(v)$ has a fixed cardinality, and furthermore, it is not possible to ensure the local invariance of CNNs.

With a view to overcoming these limits, some algorithms adapt the neighborhood aggregation function using the same W_k parameters for both node v and $N(v)$, generating the encoding h_v^k as:

$$h_v^k = \sigma \left(W_k \sum_{u \in N(v) \cup \{v\}} h_u^{k-1} \right) \quad (2.29)$$

Generalizing 2.29, we can define a GCN as a network whose input is the feature matrix $X \in \mathbb{R}^{N \times d}$, where d is the number of initial features associated with each node, and the adjacency matrix $A \in \mathbb{R}^{N \times N}$. Therefore, each hidden layer can be defined as $H^l = f(H^{l-1}, A)$, and the relative output will be a new feature matrix

of size $N \times d_l$. The equations (2.30) to (2.34) represent the propagation rule in layer l :

$$H^l = [h_1^l, \dots, h_N^l] \quad (2.30)$$

$$H^{l+1} = \sigma(AH^lW^l) \quad (2.31)$$

$$A = D^{-1/2}(\tilde{A})D^{-1/2} \quad (2.32)$$

$$\tilde{A} = A + I_N \quad (2.33)$$

$$D_{ij} = \sum_j (A_{ij}) \quad (2.34)$$

where \tilde{A} is the adjacency matrix to which the identity matrix I_N is added, generating a graph in which the self-connections are included, D is the matrix with the degrees of each node, W^l is the matrix of the weights of the l -th layer, and finally, σ represents the activation function. At the initial interaction, as was also underlined in the previous section, we have $H^0 = [x_1, \dots, x_N]$.

Based on the spatial-based method, Yan et al. [1] proposed the ST-GCN network, or Spatial-Temporal GCN, in 2018, which is the basis of many networks used in the context of skeleton-based human activity recognition.

Chapter 3

Skeleton-Based Human Activity Recognition

Human action recognition aims (HAR) to develop automatic systems for the recognition of activities carried out by people within images or videos [38]. Thanks to the development and diffusion of Deep Learning, these systems are increasingly applied in various fields, from video surveillance to sports.

The first versions of models capable of recognizing activity were based on the use of RGB images or combined the latter with depth data (RGB+D) [17, 39, 40]. However, the use of this type of data involves a high computational cost, but above all, it makes networks sensitive to noise. In fact, by processing the entire image, the network can create biases related to the background, for example. In recent years, thanks also to the development of technologies for pose estimation [41, 42], new architectures capable of working with skeletal data have been proposed.

The first algorithms based on skeletal data operated on hand-crafted features, which described the dynamism and statistical characteristics of the sequence [43, 44, 45, 46]. As can be easily understood, these techniques are expensive at the pre-processing level and are linked to the researcher's prior knowledge and the type of dataset used. Lately, architectures based on recurrent neural networks (RNNs) [47, 48], convolutional neural networks (CNN) [49, 50, 51], and graph convolutional neural networks (GCN) [1, 2, 3, 4] have been proposed, capable of extracting the relevant features autonomously.

In this chapter, we will retrace the salient stages of skeleton-based human action recognition and then delve deeper into the networks we used in our experiments. For a complete overview related to the topic of HAR, we recommend reading [38] from which passages have been extracted for the drafting of the next sections.

3.1 Skeleton-Based HAR

The use of skeletal data instead of images or RGB+D data is now preferred by researchers as it offers the possibility of training networks that are more focused on the movement and pose of the imaged subjects. Furthermore, the models at the computational level are more efficient. The explosion of their use has gone hand in hand with the development of hardware and software solutions that guarantee high accuracy in the extraction of skeletal sequences.

3.1.1 Pose estimation

The reconstruction of the skeleton is carried out starting from the estimate of the positions of the individual joints and defining the connections that exist between them. If initially the coordinates of the joints were obtained through manual labeling [52], thanks to Deep Learning this process is now automated.

2D pose estimation 2D human pose estimation can be divided into two categories:

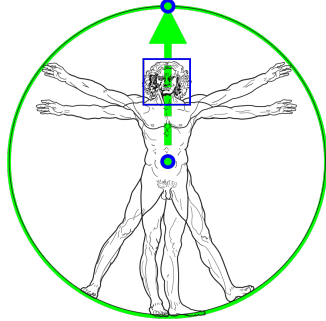
- *Direct regression approach* [53, 54]: regressing body keypoints directly from frames.
- *Heatmap-based approach* [55, 56]: the joint coordinates are obtained from heatmaps of the subject.

In case it is necessary to estimate the poses of several people at the same time, some algorithms [57] first identify all the people and then estimate their poses (top-down approach), while others [41] first detect all the joints in the image and then group them using clustering (bottom-up approach).

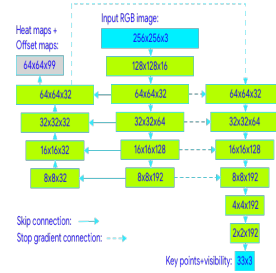
3D pose estimation Li et al. [58] were the first to apply Deep Learning to directly extract the 3D coordinates of the joints from the individual frames. Subsequent research [56] has shown how obtaining the coordinates of the joints starting from their heatmaps is a valid option even in the three-dimensional scenario. Other studies [59] have explored the possibility of extracting 3D skeleton sequences starting from 2D sequences with the aim of optimizing models based on direct regression and making them applicable to the real-time context.

Mediapipe & BlazePose In our work, we took advantage of a framework developed by Google, Mediapipe [60]. Mediapipe offers a set of Machine Learning and Deep Learning solutions that can also be used on mobile devices. In addition to image segmentation, object detection, and image segmentation tasks, in Mediapipe,

we find models capable of returning hand, face, and body landmark coordinates. To construct the sequences in our dataset, we exploited the BlazePose-based solution [42].



(a) Vitruvian man aligned via BlazePose detector vs. face detection bounding box.



(b) BlazePose architecture

Figure 3.1: On the left the vitruvian man to which is aligned the BlazePose detector, while on the right the architecture of the BlazePose pose tracker. Images from [42]

To compute the 3D positions of the joints, BlazePose first applies a pose detector to locate the subject within the image and then a pose tracker to follow its movement over time. The pose detector locates the person starting with the location of the face and then calculates other parameters such as the point at the center of the face, the size of the circle that circumscribes the person, and the incline (i.e., the angle between the lines connecting the two mid-shoulder and mid-hip points). Unlike networks such as YoloV7 [61], BlazePose activates the pose detector only when the person is not detected in the next frame. This strategy allows it not only to process at a higher fps, but also to decrease jittering. Once the person is identified, the pose tracker comes into play. The architecture consists of a combination of heatmap, offset, and regression approach. Losses from the first two are used only in the training phase to supervise the embeddings used by the regressor to calculate coordinates. During network training, it makes use of skip connections, but the regressor gradients are not extended in the heatmap trained features. It has been observed that this strategy improves the accuracy of both heatmaps and regression.

3.1.2 Handcrafted based methods

The handcrafted methods for Skeleton-based HAR consist of two steps: feature extraction and feature representation. Downstream of these two phases, a classifier, such as the SVM [43], is applied to generate the final predictions. Below is a short list of some research done in the field. Yang et al. in [44] proposed a Depth

Motion Map (DMM) which is obtained from depth information. Three motion history maps are generated by DMM by processing the spatio-temporal depth structure from the front, side, and top points of view and then describing the action through the concatenation of the extracted features. Vemulapalli et al. in [45], once having modeled the geometric connections, represent the movement through curves in the Lie group where each skeleton of the sequence represents a point of the curve. Papadopoulos et al. [46] through the analysis of the spherical angles between joints manage to obtain a real-time tracking system. In their work, the action is recognized with the help of a Hidden Markov Model (HMM) [62].

3.1.3 Deep Learning based methods

The methods based on deep learning are divided into 3 macro-categories: RNN-based, CNN-based and GCN-based. We describe these three in the following sections.

RNN Recurrent Neural Networks (RNNs) are networks widely used in natural language processing (NLP) or time-series analysis. Du et al., in [47], divided the skeleton into S parts, the 4 limbs plus the trunk and the head. Each of these sections is passed as input to bidirectional RNN units whose outputs are then fed to other bidirectional RNN units. The final structure of the model is a hierarchical RNN. Lee et al. [48] addressed the theme of feature representation by transforming the skeleton into another coordinate system to obtain robustness with respect to scaling, rotation, and translation. The model they proposed provides LSTMs with various time-step sizes to consider not only long-term dependencies but also medium- and short-term ones. In their work, Liu et al. [63] investigated how insignificant joints impact the deterioration of performance, suggesting a circular attention mechanism. Zhu et al. in [64] proposed a fully connected LSTM to structure the co-occurring features and improve the expressiveness of the network. Despite the evolution of various RNN-based models, this family of algorithms is limited in learning joint relations along the spatial dimension.

CNN The use of convolutional neural networks (CNN), whose operation is fully explained in Section 2.3, involves the reorganization of the skeleton sequences into pseudo-images. In [49], Du et al. use a CNN to which an image created by concatenating the joint coordinates rearranged in chronological order is passed and then normalized to handle sequences of variable lengths. Subsequently, Het et al. [50] proposed to organize the sequences into 3 grayscale images, one for each coordinate. Research such as that of Li et al. [65] have explored the possibility of exploiting a multi-stream CNN. In this case, the network is not only presented with the coordinates of the joints but also with the variation that this coordinate

undergoes between one frame and the next. Caetano et al. in [51] and [66] proposed further methods of skeleton representation: SkeletonMotion [51] encodes the temporal dynamics by explicitly calculating the magnitude and orientation of values of skeleton joints, TSRJI [66] (tree structure reference image) combines the use of reference joints and a tree structure skeleton. [67, 68] focused on using 3DCNN to find a solution to the long-time dependency problem as classical CNNs are not able to capture distant dependencies in time. Duan et al. proposed PoseConv3D [40] which manages to outperform even some GCN-based architectures. This network exploits 3D heatmap values guaranteeing more robustness of the GCN, and furthermore, the hierarchical features can be integrated with other modalities to further improve performance.

GCN Graph convolutional neural networks (GCN)-based methods, as they are designed for graphs, were found to be the most effective for the HAR task. In fact, a human skeleton can be seen as an undirected graph where the joints are the nodes and the edges are the ‘bones’ that join them. The first to propose a GCN for human action recognition (HAR) were Yan et al. [1]. In this algorithm, a space-time graph is constructed where the edges along the spatial dimension represent the intra-frame relationships, and those along the temporal dimension represent the inter-frame relationships. By combining multiple blocks that alternate GCN and temporal convolutional network (TCN), the network is able to process high-level features which are exploited by a softmax classifier to generate the final predictions. ST-GCN relies on a static and pre-defined adjacency matrix to describe the topology of the skeleton. Shi et al. [69] showed that if the matrix is left to be learned in a data-driven way by the network, the latter’s performance improves. Furthermore, with their study Shi et al. also explored the use of second-order features such as connection length. In a following work, with the MS-AAGCN network [2], the same authors introduced a multi-stream attention module to further improve performance. Realizing that the key to obtaining better predictions lies in the flexibility of being able to automatically learn the topology of the skeleton, Ye et al. [70] proposed Dynamic-GCN. In this model, each block for spatial modeling receives two inputs: G_{static} , a graph whose topology represents the physical connections, and G_i , a topology called dynamic and predicted by a context-encoding network. The latter is a convolutional neural network that learns in a data-driven way the dependencies between joints while also incorporating the contextual features coming from other joints. Other works have explored the possibility of developing a data-driven channel-specific refinement for the graphical structure. Inspired by the decoupling operation of CNNs, Cheng et al. [71] have proposed Decoupling-GCN, which divides the channels into decoupling groups, each of which has a trainable adjacency matrix, increasing the expressiveness of the network. Chen et al. [3] instead suggested the use of an architecture called CTR-GCN, which, starting from a shared topology,

learns a channel-specific adjacency matrix based on channel correlations. Duan et al., in [4], instead proposed two new modules: DG-GCN and DG-TCN. The first uses learned affinity matrices to capture dynamic graphical structures, while the other performs group-wise temporal convolution with varying receptive fields and incorporates a dynamic joint skeleton fusion module for adaptive multi-level temporal modeling. More recent work has explored possibilities of ‘assisting’ GCNs with LLM [72], or of adapting the transformer architecture to operate on graphs [73]. In the next sections, we will delve into the ST-GCN, MS-AAGCN, CTR-CGN, and DG-STGCN networks, which were used in the experiments to define benchmarks on our dataset.

3.2 ST-GCN

Yan et al. proposed ST-GCN to go beyond the limitations shown by RNN- and CNN-based models. Their network aims to capture both spatial and temporal information by operating directly on graphs. In this section we discuss the implementation of the architecture.

3.2.1 Graph Construction

In their work, Yan et al. defined a spatial-temporal graph. In particular, the graph $G = (V, E)$ built on a skeleton sequence with N joints and T frames contains both the intra-frame and inter-frame connections (see figure 3.2). The node set $V = \{v_{ti} | t = 1, \dots, T \text{ and } i = 1, \dots, N\}$ contains all the joints of the sequence, and for each node, the feature vector $F(v_{ti})$ contains coordinates and the estimation confidence. The network is capable of operating both with sequences extracted from 2D pose estimators and from three-dimensional sequences. The graph is constructed by first uniting all the joints within a single frame, i.e., once the frame t is fixed, the physical connections between the joints v_{ti} are defined for $i = 1, \dots, N$. Subsequently, each joint is connected with itself along the time dimension. Once the graph is completed, the set of edges is defined as $E = E_s \cup E_F$ where E_s are the intra-frame connections and E_F are the inter-frame ones.

3.2.2 Spatial Graph convolution and temporal modeling TGCN

Starting from the convolution operation (equation 3.1) on a 2D grid

$$f_{out}(x) = \sum_{h=1}^K \sum_{w=1}^K f_{in}(\mathbf{p}(x, h, w)) \cdot \mathbf{w}(h, w) \quad (3.1)$$

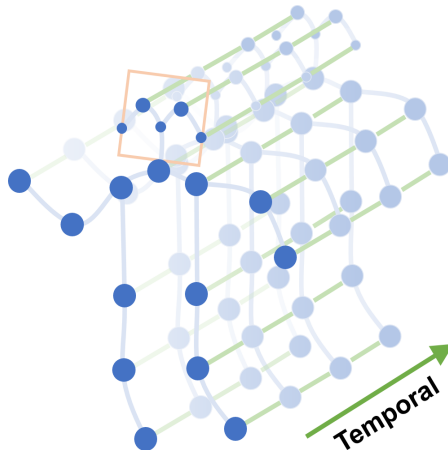


Figure 3.2: The spatial temporal graph of a skeleton sequence used in ST-GCN. Blue dots denote the body joints. The intra-body edges (light-blue) between body joints are defined based on the natural connections in human bodies. The inter-frame edges (green) connect the same joints between consecutive frames. Image from [1]

where f_{in} is the input feature map, \mathbf{p} is a sampling function $\mathbb{Z}^2 \times \mathbb{Z}^2 \rightarrow \mathbb{Z}^2$, h and w are the column and row indices of the kernel, and \mathbf{w} is a weight function $\mathbb{Z}^2 \rightarrow \mathbb{R}^c$, Yan et al. proposed an extension of the same equation for graphs. First of all, we must redefine the feature map f_{in}^t as a function $V_t \rightarrow \mathbb{R}^c$, i.e., a function that maps each node into a c -dimensional vector. Subsequently, the sampling function, which in CNNs is defined on the pixels around a central pixel x , was reformulated considering the neighbors of a node $B(v_{ti}) = \{v_{tj} | d(v_{ti}, v_{tj}) \leq D\}$, where $d(\cdot)$ is a distance measure indicating how many edges there are between the two nodes, while D is a threshold. Finally, for the weight function, the authors were inspired by the work of Niepert et al. [74] partitioning $B(v_{ti})$ into K subsets where each subset has a numeric label. By mapping each node of $B(v_{ti})$ with the label of the relevant subset, we obtain the labeling function $l_{ti} : B(v_{ti}) \rightarrow \{0, \dots, K - 1\}$ and the weight function $w(v_{ti}, v_{tj}) = w'(l_{ti}(v_{tj}))$. Three solutions (figure 3.3) are proposed for the partitioning of $B(v_{ti})$:

- *Uni-labeling:* $B(v_{ti})$ is composed of only a subset, itself. So we have that for each $v_{tj} \in B(v_{ti})$, $l_{ti}(v_{tj}) = 0$.
- *Distance partitioning:* $B(v_{ti})$ is divided based on the distance between v_{ti} and v_{tj} , therefore the labeling function is defined as $l_{ti}(v_{tj}) = d(v_{ti}, v_{tj})$.
- *Spatial-configuration:* $B(v_{ti})$ is divided into three subsets: the root-node v_{ti} , the centripetal group composed of the nodes closest to the center of gravity

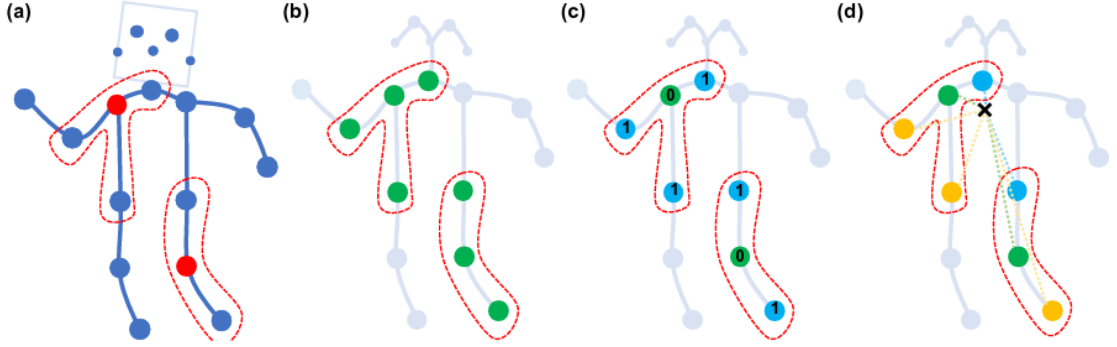


Figure 3.3: Partition strategies. (a) An example of input skeleton with joints connections in blue and receptive filed with red dashed dots. (b) Uni-labeling, (c) Distance partitioning and (d) Spatial configuration. Image from [1]

with respect to v_{ti} , and centrifugal group composed of the nodes further away from the center of gravity than v_{ti} . The labeling function assigns a value of 0, 1, and 2 respectively to the three subsets.

At this point, we can rewrite equation 3.1 as:

$$f_{out}(v_{ti}) = \sum_{v_{tj} \in B(v_{ti})} \frac{1}{Z_{ti}(v_{tj})} f_{in}(v_{tj}) \cdot \mathbf{w}(l_{ti}(v_{tj})) \quad (3.2)$$

where $Z_{ti}(v_{tj})$ is the cardinality of the subset to which v_{tj} belongs.

Given the way in which the graph is constructed, for the convolution on the temporal dimension, it is sufficient to redefine $B(v_{ti}) = \{v_{qi} | d(v_{tj}, v_{ti}) \leq K, |q - t| \leq \lfloor \Gamma/2 \rfloor\}$ where Γ controls the time range to include, q and t are the temporal indexes of the nodes and K is the temporal distance between the two nodes. The weight function becomes $l_{ST}(v_{qi}) = l_{ti}(v_{tj}) + (q - t + \lfloor \Gamma/2 \rfloor) \times K$.

3.2.3 Architecture

ST-GCN is made up of 9 layers divided into 3 groups with 64, 128, and 256 output channels respectively. The stride of the fourth and seventh blocks is set to 2 as the pooling layer. After the last block, a global average pooling is performed whose output is passed to a softmax classifier. The ResNet mechanism is applied to each block. Each ST-GCN block first performs a spatial convolution and then a temporal one. The spatial convolution is operated using the formulation proposed by Kipf and Welly [34]:

$$f_{out} = \mathbf{D}^{-\frac{1}{2}} (\mathbf{A} + \mathbf{I}) \mathbf{D}^{-\frac{1}{2}} f_{in} \mathbf{W} \quad (3.3)$$

$$D_{ii} = \sum_j (A_{ij} + I_{ij}) \quad (3.4)$$

where \mathbf{A} is the adjacency matrix of the undirected graph representing intra-body connections, \mathbf{W} is a trainable weight matrix, \mathbf{I} is the identity matrix and \mathbf{D} is the matrix containing the degree of the nodes.

The spatial convolution is nothing more than a 1x1 2D standard convolution to aggregate the node and the features of its neighbors through multiplication with the normalized adjacency matrix. To include partitioning strategies, 3.3 can be changed to 3.5 where A is dismantled into A_j adjacency matrices defined such that $A + I = \sum_j A_j$ e $D_j^{-1/2} = \sum_k A_j^{ik} + \alpha$.

$$f_{out} = \sum_j \mathbf{D}_j^{-\frac{1}{2}} \mathbf{A}_j \mathbf{D}_j^{-\frac{1}{2}} f_{in} \mathbf{W}_j \quad (3.5)$$

Yan et al. also propose to define a learnable edge importance matrix to multiply element-wise to the adjacency matrices.

The temporal convolution, on the other hand, is operated as a standard 1xF convolution on the dimensions (V,T) where V is the number of nodes, T is the number of frames and F is the receptive field, i.e., the number of frames to be considered.

3.3 MS-AAGCN

Given the static and pre-defined nature of ST-GCN adjacency matrices, Shi et al. [69, 2] proposed to replace it with topologies learned in a data-driven manner. They also suggested creating a multi-stream architecture so as to be able to also exploit second-order information such as the length of the edges between joints. Finally, in [2], they introduced an attention mechanism that operates on multiple levels. In this section, we will describe the MS-AAGCN architecture in detail as it also includes the previous version described in [69].

3.3.1 Adaptive Graph convolutional Layer

The proposed adaptive graph convolutional layer aims to make the network more flexible by deriving the adjacency matrix from the data.

$$f_{out} = \sum_j \mathbf{W}_j (f_{in} \bar{\mathbf{A}}_j) \quad (3.6)$$

Starting from the equation 3.5, which can be rewritten as reported in equation 3.6, where $\bar{\mathbf{A}}_j = \mathbf{D}_j^{-1/2} \mathbf{A}_j \mathbf{D}_j^{-1/2}$, Shi et al. inserted two new matrices, obtaining the following convolution:

$$f_{out} = \sum_j \mathbf{W}_j f_{in}(\mathbf{B}_j + \alpha \mathbf{C}_j) \quad (3.7)$$

The first subgraph, \mathbf{B}_j , is the global graph learned from the data. Initialized with \mathbf{A}_j , during training the matrix is updated without any constraints. \mathbf{B}_j is unique for each layer. The second subgraph, \mathbf{C}_j , however, is a graph that learns a unique topology for each sample. In particular, through a Gaussian function (equation 3.8) for each pair (v_i, v_j) , a similarity measure is calculated, verifying whether a connection exists between them and how strong this is.

$$f(v_i, v_j) = \frac{e^{\theta(v_i)^T \phi(v_j)}}{\sum_{j=1}^N e^{\theta(v_i)^T \phi(v_j)}} \quad (3.8)$$

In detail, given $f_{in} \in \mathbb{R}^{C_{in} \times T \times N}$, f_{in} is first embedded into the space $\mathbb{R}^{C_e \times T \times N}$ with two functions θ and ϕ represented by \mathbf{W}_{θ_j} and \mathbf{W}_{ϕ_j} . The embeddings obtained are reshuffled into $\mathbf{M}_{\theta_j} \in \mathbb{R}^{N \times C_e T}$ and $\mathbf{M}_{\phi_j} \in \mathbb{R}^{C_e T \times N}$, and subsequently multiplied together to obtain \mathbf{C}_j , which is finally normalized via SoftMax in the range $[0,1]$. The \mathbf{C}_j equation can therefore be summarized as:

$$\mathbf{C}_j = \text{SoftMax}(f_{in}^T \mathbf{W}_{\theta_j}^T \mathbf{W}_{\phi_j} f_{in}) \quad (3.9)$$

The matrix \mathbf{C}_j obtained with equation 3.9 is multiplied with a learnable parameter γ , unique for each layer, as it has been observed that this matrix is required with different intensities along the various layers.

3.3.2 STC - Attention module

Through the STC-attention module, the network is able to give different importance at joint, frame, and channel levels:

- *Spatial attention module* (SAM) allows the model to pay different levels of attention to joints. It computes the following $\mathbf{M}_s \in \mathbb{R}^{1 \times 1 \times N}$ as $\mathbf{M}_s = \sigma(g_s(\text{AvgPool}(f_{in})))$, where g_s is a 1×1 convolution and σ is the sigmoid function.
- *Temporal attention module* (TAM) is similar to SAM and derives the matrix $\mathbf{M}_t \in \mathbb{R}^{1 \times T \times 1}$ as $\mathbf{M}_t = \sigma(g_t(\text{AvgPool}(f_{in})))$.
- *Channel attention module* helps the model strengthen the discriminative features with the matrix $\mathbf{M}_c \in \mathbb{R}^{C \times 1 \times 1}$ as $\mathbf{M}_c = \sigma(\mathbf{W}_2(\delta(\mathbf{W}_1(\text{AvgPool}(f_{in}))))))$, where δ is a ReLU.

Through experiments, it has been noticed that concatenating the modules sequentially leads to better performance than their sum.

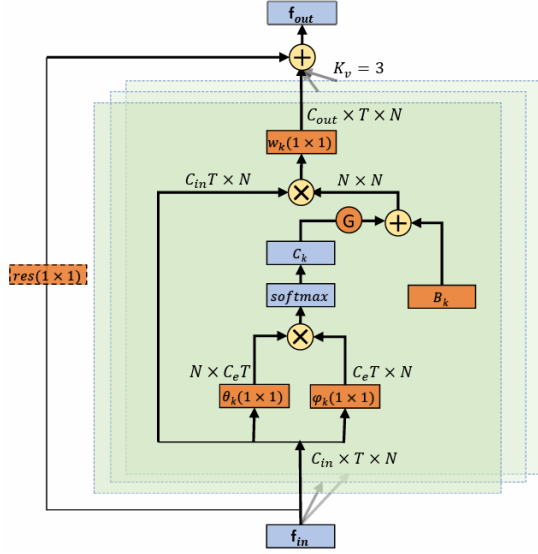


Figure 3.4: Illustration of the adaptive graph convolutional layer (AGCL). There are two kinds of graphs in each layer, i.e., B_k and C_k . The orange box indicates that the part is the parameter of the network and is updated during the training process. θ and ϕ are two embedding functions whose kernel size is (1×1) . K_v denotes the number of subsets. \oplus denotes the element-wise addition. \otimes denotes the matrix multiplication. G is the gate that controls the importance of the two kinds of graphs. The residual box (dotted line) is only needed when C_{in} is not the same as C_{out} . Image from [2].

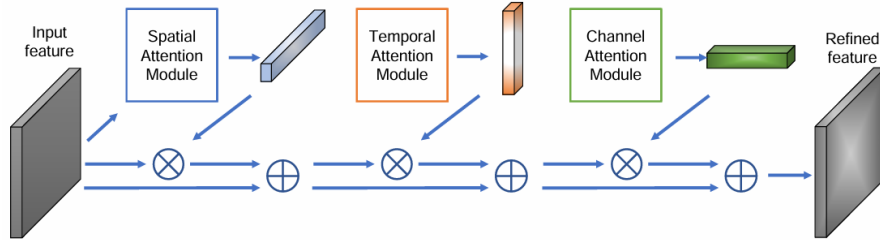


Figure 3.5: Illustration of the STC-attention module. Three sub-modules are arranged sequentially in the orders of SAM, TAM and CAM. \otimes denotes the element-wise multiplication. \oplus denotes the element-wise addition. Image from [2].

3.3.3 Architecture

The architecture used follows the ST-GCN setting with 9 layers divided into 3 groups per output channel and a global average pooling at the end to obtain the final output to be passed to the softmax classifier. Each layer of the network

(see figure 3.6) is made up of a spatial GCN module and TCN module followed by a Batch Normalization layer and a ReLU layer. The STC-module is inserted between the two modules. In the multi-stream scenario, the softmax scores are added together to obtain the final predictions.

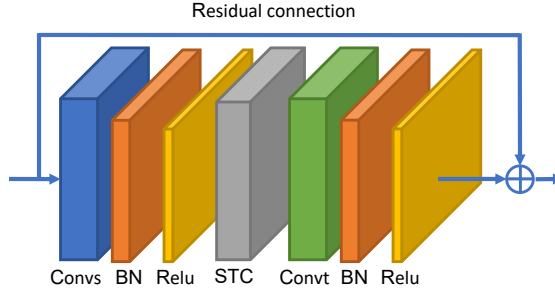


Figure 3.6: Illustration of the MS-AAGCN basic block. Convs represents the spatial adaptive graph convolutional layer (AGCL), and ConvT represents the temporal AGCL, both of which are followed by a BN layer and a ReLU layer. STC represents the STC-attention module. Moreover, a residual connection is added for each block. Image from [2].

3.4 CTR-GCN

To further increase the flexibility of the network Chen et al. with CTR-GCN they proposed a network which, starting from a common topology, creates a specific one for each channel by exploiting channel-specific correlations.

3.4.1 Channel-wise Topology Refinement Graph Convolution

The CTR-GC module framework, shown in figure 3.7, is divided into three parts. We describe those three in the following.

Feature transformation This module aims to transform the input into high-level features via a simple linear transformation.

$$\mathcal{T}(\mathbf{X}) = \mathbf{X}\mathbf{W} = \tilde{\mathbf{X}} \quad (3.10)$$

where $\mathbf{X} \in \mathcal{R}^{N \times C}$ is the transformed feature and $\mathbf{W} \in \mathcal{R}^{C \times C}$ is the weight matrix.

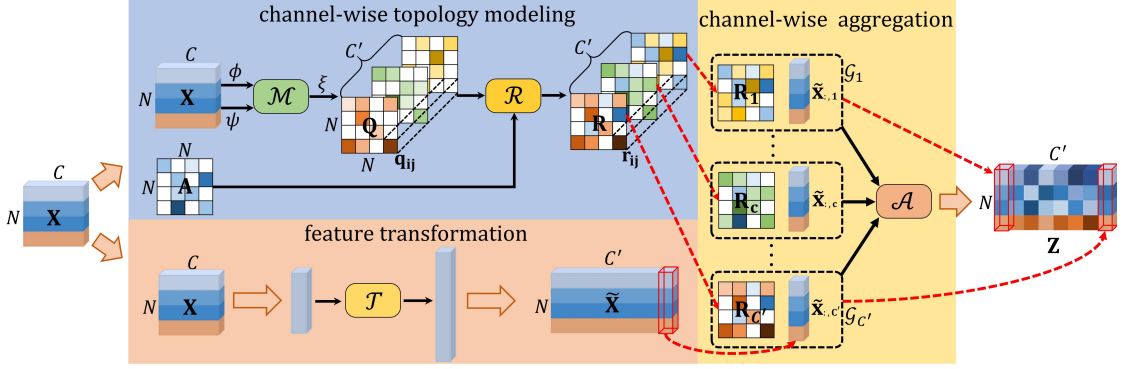


Figure 3.7: Framework of the channel-wise topology refinement graph convolution proposed in [3]. The channel-wise topology modeling refines the trainable shared topology with inferred channel-specific correlations. The feature transformation aims at transforming input features into high-level representations. Eventually, the output feature is obtained by channel-wise aggregation. Image from [3].

Channel-wise topology modeling Here, the adjacency matrix \mathbf{A} is used as topology shared between all channels and is approximated via backpropagation. In addition to this matrix, the matrix $\mathbf{Q} \in \mathbb{R}^{N \times N}$ containing the channel-specific correlations is learned. To obtain it, two linear transformations ψ and ϕ are applied to the input \mathbf{X} to reduce its dimensions before sending the input features into $\mathcal{M}(\cdot)$. Given a pair of vertices (v_i, v_j) and the corresponding features (x_i, x_j) , the authors designed two correlation functions. The first, 3.11, calculates the distance between $\psi(x_i)$ and $\phi(x_j)$ and then applies a non-linear function. The second, 3.12, instead concatenates the results of ϕ and ψ and then applies a multi-layer perceptron.

$$\mathcal{M}_1(\psi(x_i), \phi(x_j)) = \sigma(\psi(x_i) - \phi(x_j)) \quad (3.11)$$

$$\mathcal{M}_2(\psi(x_i), \phi(x_j)) = \text{MLP}(\psi(x_i) || \phi(x_j)) \quad (3.12)$$

For each pair (v_i, v_j) , the corresponding vector $q_{ij} \in \mathbb{R}^{C'}$ representing the channel-specific relationship between the two is calculated as:

$$q_{ij} = \xi(\mathcal{M}(\psi(x_i), \phi(x_j))) \quad (3.13)$$

where ξ is a linear transformation. The final channel-wise topology is calculated as $\mathbf{R} = \mathcal{R}(\mathbf{Q}, \mathbf{A}) = \mathbf{A} + \alpha \cdot \mathbf{Q}$, where α is an hyperparameter to control the strength of \mathbf{Q} .

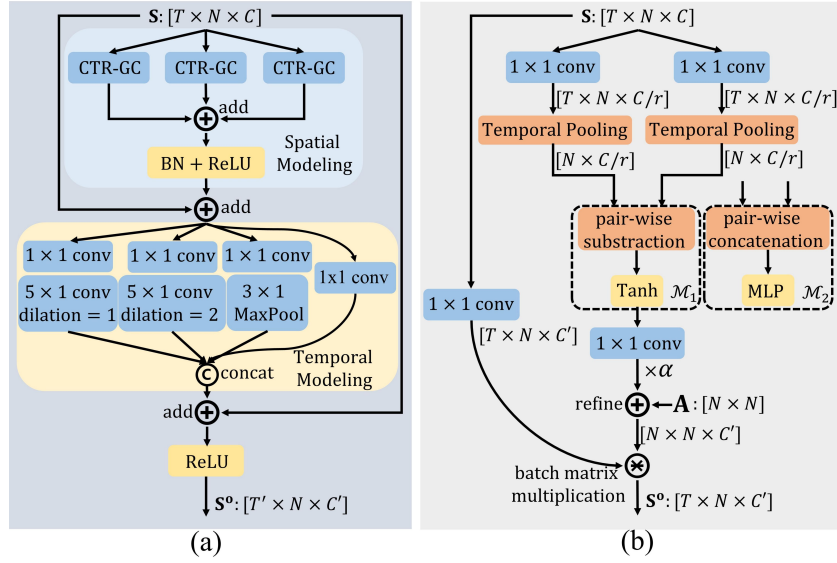


Figure 3.8: (a) The basic block of CTR-GCN. (b) CTR-GC with correlation modeling function $\mathcal{M}_1(\cdot)$ or $\mathcal{M}_2(\cdot)$. Image from [3].

Channel-wise aggregation Given \mathbf{R} and \tilde{X} , CTR-GC aggregates the features in a channel-wise manner and the final output is formulated as:

$$\mathbf{Z} = \mathcal{A}(\tilde{X}, \mathbf{R}) = [\mathbf{R}_1 \tilde{x}_{:,1} || \dots || \mathbf{R}_{C'} \tilde{x}_{:,C'}] \quad (3.14)$$

where $||$ indicates concatenation.

3.4.2 Architecture

The network consists of 10 blocks divided similarly to what was done in ST-GCN with the first 4 blocks having output channels equal to 64, the following 3 with output channels 128, and the last 3 with 256 output features. Each block (figure 3.8), along the time dimension, applies the CTR-GC module just described in parallel 3 times and sums the outputs obtained. To fit a skeletal sequence of size $S \in \mathbb{R}^{T \times N \times C}$, pooling along the time dimension is applied. To model actions with variable length, a multi-scale temporal module has been implemented in which different receptive fields are applied, and the results are chained together.

3.5 DG-STGCN

Duan et al. proposed the DG-STGCN architecture, comprising two modules: DG-GCN and DG-TCN, responsible for spatial and temporal convolutions, respectively.

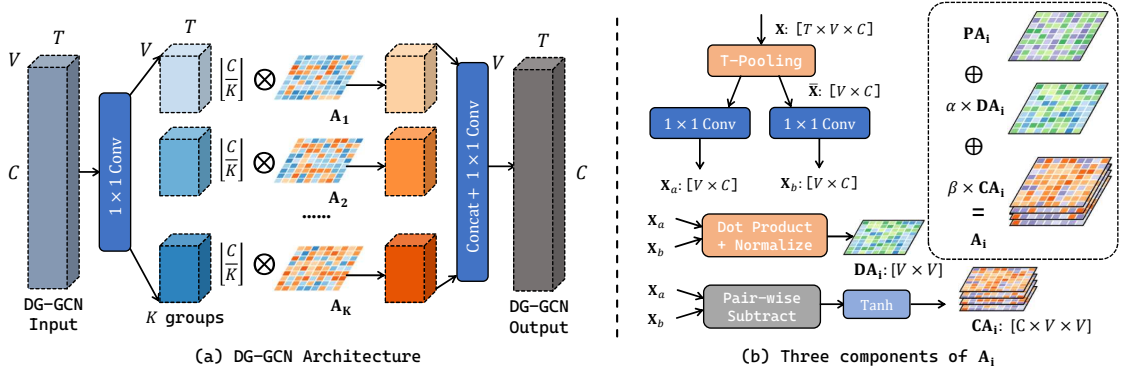


Figure 3.9: The architecture of the dynamic group GCN (DG-GCN). Image from [4].

Both modules segment the input into subgroups, which are processed independently before being merged at the output. Unlike [2] and [3], the key distinction lies in DG-GCN, which learns the adjacency matrix \mathbf{A}_k entirely without any prior assumptions. In this section we will look in detail at the two modules and the pre-processing strategy proposed by the authors.

3.5.1 DG-GCN

As mentioned earlier, while in [2] and [3], the network learns a refinement $\delta\mathbf{A}$ from an adjacency matrix \mathbf{A} , Duan et al. opted to make \mathbf{A} fully learnable by initializing it with a normal distribution. Once $\mathbf{A} \in \mathbb{R}^{K \times V \times V}$ has been initialized, where V is the number of nodes comprising the skeleton, the input of dimensions (C, V, T) is divided into K groups via a 1×1 convolution. Then, spatial modeling is carried out by multiplying the k -th group with \mathbf{A}_k , which is defined as:

$$\mathbf{A}_k = P\mathbf{A}_k + \alpha \times D\mathbf{A}_k + \beta \times C\mathbf{A}_k \quad (3.15)$$

where $P\mathbf{A}_k$ is referred to as the dataset-wise parameter and is a static element learned during training and also α and β are learnable parameters. The terms $D\mathbf{A}_k$ (the channel-agnostic component, equation 3.16) and $C\mathbf{A}_k$ (the channel-specific component, equation 3.17) are obtained from the input \mathbf{X} , to which a temporal pooling operation is applied to remove the temporal component, followed by two 1×1 convolutions.

$$D\mathbf{A}_k = \text{Softmax}(X_a X_b^T, \dim = 0) \quad (3.16)$$

$$C\mathbf{A}_k = \text{Tanh}(X_a - X_b) \quad (3.17)$$

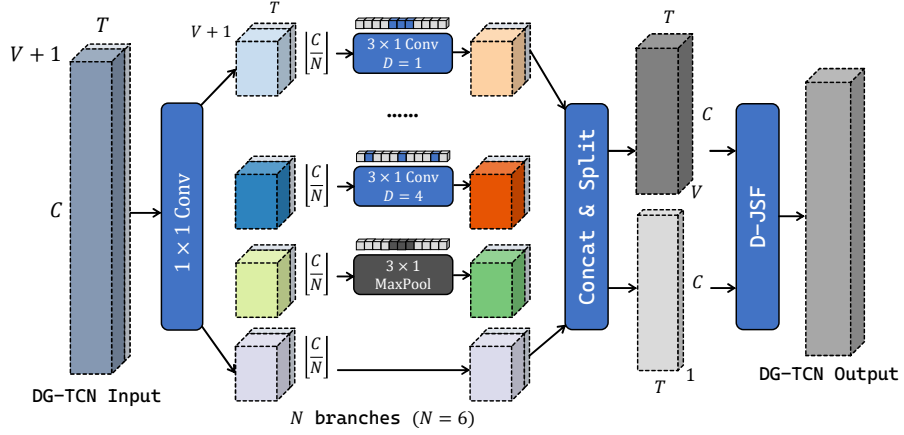


Figure 3.10: The architecture of the dynamic group TCN (DG-TCN). ‘D’ indicates dilation. Image from [4].

3.5.2 DG-TCN

The DG-TCN module 3.10 operates temporal modeling at both joint and skeleton levels simultaneously. To achieve this, an average pooling layer is applied to the input of the module along the V dimension to obtain the skeleton level features S . Afterward, S is concatenated to the X input along the V dimension. At this point, the input is divided into N groups, where N is the number of branches in the multi-group TCN, which generates the outputs X' and S' . The latter are fed to the D-JSF module, which generates the final output. Inside D-JSF, a learnable parameter $\gamma \in \mathbb{R}^V$ is used to calculate the i -th feature joint as $X'_i + \gamma S'_i$.

3.5.3 Uniform sampling

The uniform sampling technique allows generating more diverse samples than random cropping [3] or padding [75], while guaranteeing their integrity. To generate samples of N frames, the initial sequence of length T is divided into N non-overlapping windows, and then one frame is randomly selected for each window.

3.5.4 Architecture

In the DG-STGCN architecture, which builds upon the fundamental design of ST-GCN, a total of 10 GCN blocks are employed. Each GCN block comprises a DG-GCN (Dynamic Graph Convolutional Network) and a DG-TCN (Dynamic Graph Temporal Convolutional Network). The base channel width is set to 64. Notably, at the 5th and 8th GCN blocks, temporal pooling is applied to reduce the temporal length by half and simultaneously double the channel width.

Chapter 4

Temporal Action Segmentation

In computer vision, action recognition is a fundamental task for video understanding, wherein pre-trimmed video clips are classified with single semantic labels. While state-of-the-art methods can distinguish hundreds of classes, this approach is limiting as real-world video streams, such as surveillance feeds and autonomous vehicle data, are continuous and may contain actions spanning longer durations. Standard action recognition methods designed for short clips are not directly applicable to these scenarios. Temporal Action Segmentation (TAS) involves segmenting temporally untrimmed video sequences by labeling each segment with one of several predefined action labels. This task parallels semantic segmentation but operates in the temporal domain, replacing pixel-wise semantic labels with frame-wise action labels. By automatically segmenting untrimmed videos, it aids in understanding actions' timing, progression, environmental impact, and predicting future actions, facilitating various applications like assistive technologies, video surveillance, and human-robot interactions. In this chapter we give an overview based on '*Temporal Action Segmentation: An Analysis of Modern Techniques*' [76], a survey published by Ding et al., whose reading is strongly recommended for those who want to delve deeper into the topic.

The chapter is divided in related task (section 4.1), core techniques (section 4.2), level of supervision (section 4.3) and finally we describe the MS-GCN [5] network (section 4.4) which is used in the experiments

4.1 Related task

Formally, given a video or a sequence $x = (x_1, \dots, x_T)$ of length T with N actions, Temporal Action Segmentation (TAS) produces the following output:

$$s_{1:N} = \{(c_1, l_1), \dots, (c_N, l_N)\} \quad (4.1)$$

where $s_N = (c_N, l_N)$ represents a segment with length l_N to which a label c_N is assigned. The same task can be reformulated as frame-wise action segmentation:

$$y_{1:T} = \{y_1, \dots, y_T\} \quad (4.2)$$

where y_i is the label assigned to the i -th frame. Similarly to TAS, other tasks fall under the video understanding domains:

- *Temporal Action Detection & Localization (TAD\&L)* [77, 78], detects start and end of actions and predicts labels simultaneously. In contrast, TAS produces labels at the frame-wise level.
- *Sequence Segmentation (SS)*, popular in motion capture domain, aims to segment one [79] or more sequences at the same time [80].
- *Keyframe Detection (KFD)*, for each action, detects some key frames [81, 82].
- *Complex Activity Classification (CAC)* [83, 84], classifies procedural activity videos at the complexity activity level.
- *Generic Event Boundary Detection (GEBD)* [85], unlike TAS, it does not provide semantic labels but detects just the action changes.

4.2 Core techniques

Temporal action segmentation (TAS) leverages two fundamental techniques for its effectiveness: frame-wise representations, which extract informative features at the level of individual frames to capture both spatial and motion information, and temporal and sequential modeling, which integrates temporal dependencies and sequential context to enhance TAS accuracy.

4.2.1 Frame-Wise Representations

In TAS it is usual to use precomputed frame-wise features as input, rather than preferring end-to-end learning, mainly because of the substantial computational resources required for learning video features. Two most common frame-wise representations:

- *Fisher Vector Encoded IDT*: Before the rise of deep learning, Improved Dense Trajectories (IDTs) were commonly employed. Initially, IDTs comprised

spatiotemporal features derived from optical flow. The introduction of the Fish vector [86] enhanced IDTs by capturing both first- and second-order statistics of trajectories.

- *Inflated 3D ConvNets*: I3D [87] represent a state-of-the-art architecture for extracting features in video comprehension tasks. Pre-trained on the Kinetics dataset [88], the I3D model processes both optical flow and RGB video inputs, generating 2048D representations.

4.2.2 Temporal and sequential modeling

Temporal modeling at the frame-wise level extends the receptive fields over time and aggregates frame features, facilitating information exchange between frames. Various architectures have been employed for temporal modeling:

- *Recurrent Neural Networks (RNNs)* [89, 90, 37]: RNNs capture temporal dependencies using shared parameters, recursively processing input features in their temporal order. However, RNNs suffer from memory limitations when analyzing long time series and cannot process sequential data in parallel.
- *Temporal Convolutional Networks (TCNs)*: TCNs utilize 1x1 convolutions to extract temporal relationships in both causal and acausal modes. Two paradigms exist: encoder-decoder [91, 92] and multi-stage [93]. The latter maintains full temporal resolution and forms the foundation of MS-GCN.
- *Transformers*: Transformers leverage attention mechanisms [94] as their core technique. While some research demonstrates their suitability for TAS, their adoption is limited due to the extensive data requirements for training.

Sequential modeling is particularly effective at capturing features at the segment level, especially when actions exhibit a specific order or sequence. Several studies, such as [95, 96, 97], have explored the application of Hidden Markov Models and Markov Models for sequential modeling. However, as our focus is on temporal modeling in this work, we do not delve into sequential modeling in detail in this section. For those interested in further exploration, we recommend referring to [76] for more comprehensive insights.

4.2.3 Over-segmentation

In procedural actions, local continuity is crucial, implying that actions should change only at their boundaries. Researchers have aimed to refine resulting segments at these boundaries to enhance performance. Boundary refinement techniques, such as those proposed by Wang et al. [98], utilize local barrier pooling operations to smooth

boundary predictions, reducing ambiguity and over-segmentation. Additionally, some approaches supplement segmentation outputs with detected boundaries from separate network structures [99]. Gaussian smoothing, achieved through Gaussian kernel operations, enhances action continuity within narrow temporal windows, effectively improving segmentation metrics. While some methods directly apply smoothing to frame-wise action probabilities [100], others utilize sequential similarity scores between consecutive frames for more robust boundary detection [101]. Some studies suggest that soft action boundaries may enhance TAS performance compared to hard transitions [102, 103]. Variants of soft action boundaries, including linear and sigmoidal transitions, have been proposed and compared in terms of their effectiveness.

4.3 Level of supervision

In temporal action segmentation (TAS), various forms of supervision have been explored. The *fully-supervised* setting, as exemplified by works such as [93] and [104], involves dense action labels for every frame in training video sequences. However, this approach is labor-intensive, requiring annotators to view the entire video sequence. *Semi-supervised* methods, as discussed in [105] and [102], reduce annotation effort by densely annotating a subset of videos while treating others as unlabeled samples. Weak labels, requiring less annotation effort than dense video labels, encompass different forms such as *time-stamped annotations* [106], [107], *action lists* or *transcripts* [95], [108], and *action sets* [109], [110], [111], [112]. Recently, [100] proposed using complex activity labels at the video level, eliminating the need for action-level annotations altogether. In contrast, the *unsupervised* setting, as explored in works like [96], [113], [101], considers collections of videos performing the same activity. Although not label-free, this approach requires activity labels to form video collections. It is comparable to weak activity label supervision [100] in terms of label information, but differs in how video collections are processed during training. Specifically, unsupervised methods typically work with one group of same-activity videos at a time, while activity label supervision involves videos from all activities simultaneously.

4.4 MS-GCN

MS-GCN was proposed by Filtjens et al. [5] to address the Temporal Action Segmentation task. The basic idea is to combine the abilities to operate on graphs, exploit the spatial hierarchy between joints, of ST-GCN [1] and the over-segmentation error reduction capabilities of MS-TCN [93]. First, they modified ST-GCN blocks by including dilation in temporal graph convolution. Next, they

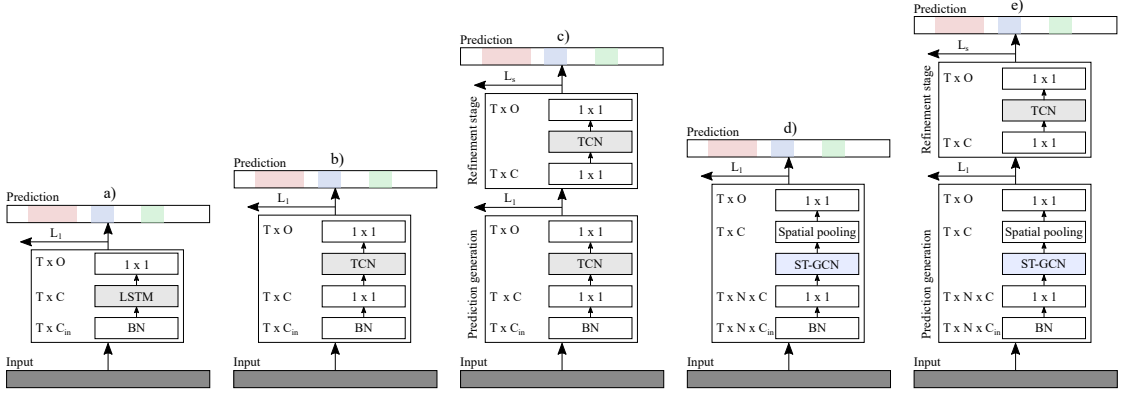


Figure 4.1: (a) a long short-term memory network (LSTM), (b) a temporal convolutional neural network (TCN), (c) a multi-stage temporal convolutional neural network (MS-TCN), (d) a spatial-temporal graph convolutional neural network (STGCN), and (e) a multi-stage spatial-temporal graph convolutional neural network (MS-GCN). Image from [5].

leveraged the decoupled prediction and refinement stages of MS-TCN by replacing the TCN blocks of the prediction stage with the modified ST-GCN blocks.

4.4.1 TCN

The authors, while presenting the structure of their proposed model, introduced the TCN-based model, which is a fundamental component of MS-TCN.

In a TCN-based model, the first layer consists of a 1×1 convolution that maps $f_{\text{in}} \in \mathbb{R}^{T \times C_{\text{in}}}$ to $f_{\text{adj}} \in \mathbb{R}^{T \times C}$, where f_{in} represents a MoCap sequence. Subsequently, f_{adj} is processed by multiple TCN blocks. Each block comprises a dilated temporal convolution [114], followed by batch normalization (BN) and a ReLU layer, and finally, a residual connection with the input. The final hidden output is defined as:

$$f_{\text{out}} = \text{ReLU}(\text{BN}(\mathbf{W} *_{d} f_{\text{adj}} + b)) + f_{\text{adj}} \quad (4.3)$$

where $*_{d}$ represents the dilated convolution.

4.4.2 Multi-stage models

Both TCN-based models and ST-GCN map an input sequence f_{in} to a hidden representation f_{out} . To obtain the final predictions, both models use a 1×1 convolution followed by a softmax:

$$\hat{Y} = \text{SoftMax}(W_1 * f_{\text{out}} + b), \quad (4.4)$$

where $\hat{Y} \in \mathbb{R}^{T \times L}$ encapsulates the probabilities for each frame to belong to class $l \in L$, $*$ the convolution operator, $b \in \mathbb{R}^C$ the bias term, and $\mathbf{W}_1 \in \mathbb{R}^{1 \times C_{in} \times C}$ the weights of the 1×1 convolution filter with C_{in} input feature channels and C equal to the number of feature channels. In multi-stage models, \hat{Y} obtained through ST-GCN or TCN is then used as input to refinement stages. Each stage s , consisting of multiple TCN blocks described in section 4.4.1, operates on the output of the previous stage according to the equation:

$$\hat{Y}^s = \Gamma(\hat{Y}^{s-1}) \quad (4.5)$$

where Γ denotes the single-stage TCN. If in MS-TCN the prediction stage to generate \hat{Y} is also composed of TCN blocks, in MS-GCN the ST-GCN blocks are exploited.

4.4.3 Architecture

The architecture of MS-GCN (figure 4.1e) follows the same settings as MS-TCN (figure 4.1c): one prediction stage and 3 refinement stages. Each stage consists of 10 layers with 64 filters and temporal kernel size of 3. The dilated receptive field of the i -th layer is equal to 2^i , with $i = 0, \dots, 9$.

The model is trained by minimizing a combination of cross-entropy loss (4.6) and MSE (4.7). This combination (4.8), suggested in [93], aims to minimize the over-segmentation error by means of the MSE, which casts the mean squared error over the sample-wise log probabilities.

$$\mathcal{L}_{cls} = \frac{1}{T} \sum_t -y_{t,l} \log(\hat{y}_{t,l}) \quad (4.6)$$

$$\mathcal{L}_{T-MSE} = \frac{1}{TL} \sum_{t,l} \hat{\Delta}_{t,l}^2 \quad (4.7)$$

$$\mathcal{L} = \sum_{s=1}^S \mathcal{L}_{s,cls} + \lambda \mathcal{L}_{s,T-MSE} \quad (4.8)$$

where y_{tl} and \hat{y}_{tl} the ground truth label and predicted probability for class l at sample t , respectively.

Chapter 5

3DYogSeg: a new dataset for pose recognition and segmentation

Given the lack of public datasets with 3D skeleton sequences for the task of activity recognition and video segmentation in the world of yoga, starting from summer 2023 we started collecting videos to build our personal dataset, 3DYogSeg. In this chapter we retrace the steps that led to the definition of the sequences and how we build a syntetic dataset for video action recognition from. In particular the chatper is divide as following: section 5.1 lists the yoga dataset available, section 5.2 illustrates how we collect the video, section 5.3 explain how we extracted the skeleton sequences, section 5.4 includes all the trasformation we perform to the row sequences, in 5.5 is described how the training and test are built, and finally in section 5.6 we describe the pipeline to assemble the synthetic dataset some statistics about the final dataset.

5.1 Yoga datasets

Thanks to the growing interest in health and physical well-being, in recent years there have been more and more studies linked to the world of sport and fitness. In parallel with this, the number and variety of yoga-related datasets is also growing (table 5.1). **Yoga82** [115] provides RGB images for 82 types of exercises distributed on a hierarchy of 3 levels based on the position the body takes. On average, approximately 347 samples per asana were collected via the Bing search engine, totaling more than 28.4K images. **YAR (Yoga Asana Recognition)** [116] was developed to train the YogNet network. The dataset contains videos of 16

participants who were filmed via a 2D web camera from 4 angles: frontal, from the back, and sideways. In total, 1206 videos were collected. **3D-Yoga** [117] consists of 3.8K action samples and 16,668 RGB-D keyframes. The dataset was constructed by filming 22 subjects from two different angles at the same time. Starting from the skeletons extracted from the two angles, the final skeletons were reconstructed. In total, there are 117 asanas for which a hierarchical annotation similar to Yoga82 is provided. **YogaTube** [118] consists of 5284 videos for 82 different exercises. The RGB videos were collected via Youtube and other search engines such as Bing. **3DYoga90** [119], built at the same time as our dataset and released at the end of 2023. It was designed starting from the Yoga82 hierarchy. For the same asanas plus 8 others, 6177 videos were collected from Youtube from which 5526 skeletal sequences were then extracted. Our dataset, like 3DYoga90, consists of skeletal sequences extracted from YouTube videos. What differentiates us from 3DYoga90 is that in our clips the poses are held for the entire duration without including the preparation for the pose and the exit. Also there are some poses that are not present in 3DYoga90.

Dataset	Asana	Samples	Type	Source available
Yoga-82	82	28,478	RGB-image	Images
YAR	20	1,206	RGB-video	Video
3D-Yoga	117	3,782	RGB-video\Skeleton	-
YogaTube	82	5,484	RGB-video	-
3DYoga90	90	6,177/5,526	RGB-video\Skeleton	Video url
3DYogSeg	58	2,115/2027	RGB-video\Skeleton	Video url

Table 5.1: Yoga lesson statistics. For each lesson, the video id, the duration in minutes and seconds, the number of tags within the lesson, the average length of a tag in seconds, and what are the available raw data sources are reported.

5.2 Video Collection

The dataset we proposed is composed of sequences extracted from videos on YouTube. Starting from the most common asanas, videos were searched for by names in English, Italian and Sanskrit, the ‘official’ language of yoga. Given the difficulty of identifying videos for which the pose was maintained for the entire duration of the video, we extracted clips which had to respect the following criteria to be considered valid:

1. Only one person must be filmed in the frame since in a video lesson usually only the teacher is filmed

2. The instructor maintains the pose throughout the clip, with minimal movement of non-essential body parts for demonstrating the asana. Given the breadth of variations that a pose can have is based on the position of a limb for example, it is important that during the training there is as little noise as possible with respect to the asana that the net is to learn.
3. All joints must be visible, with possible occlusion by the instructor’s body. In this way, the BlazePose model is able to determine with more certainty the true coordinates of the joints.

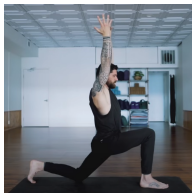







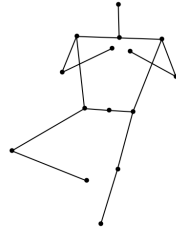
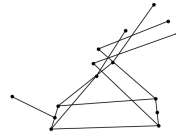
Low-lunge	Forward Fold	Facing downdog	Tree pose	Happy baby
				
				

Table 5.2: Dataset examples of 3DYogSeg. For each pose indicated in the header we reported both the RGB image and the relative skeletal representation.

Regarding the minimum or maximum duration of the clip, no stakes have been set since the same asana during a flow can be held for a few seconds or for several minutes. Furthermore, given that the ultimate objective is to segment video lessons during which the viewpoint is not always fixed, for each exercise we have collected some clips during which the point of view changes. Each collected clip is associated with a start and end time (in minute-second format) along with the video URL for downloading purposes, for a total of 2115 videos spanning 58 asanas were identified, with an average duration of approximately 12.91 seconds per clip. Figure 5.1 reports the distribution of the number of clips by class. The asana with the most clips collected is number 49 (*sphinx*) with 49 clips, followed by number 9 (*downward facing dog*) with 46 clips and number 1 (*reverse warrior*) with 45 clips. The poses with the least saved clips are 106 (*revolved high lunge*), 25 (*locust*) and 41 (*yogi squat*) with 12,19 and 20 samples respectively. From figure 5.2, instead, the average



Figure 5.1: Number of clips for each class.

duration of the clips for each class can be analyzed. From this point of view, classes 67 (*staff pose*), 95 (*yogis bicycle*) and 13 (*standing forward fold*) have the longest durations ranging from 25 to 36 seconds, while asanas 20 (*chaturanga*), 85 (*eagle*) and 51 (*thread the needle*) whose average number of clips vary between 3.5 and 4.5 seconds close the ranking. Table 5.2 shows some examples of poses included within the dataset with the relative RGB-frame and the skeleton.

5.3 Skeleton extraction

After acquiring the videos, we utilized the Mediapipe [60] library to extract skeletal data, a powerful open-source tool developed by Google for various computer vision tasks. Specifically, we employed BlazePose [42], a model tailored for real-time pose estimation. BlazePose provides 33 joints per frame, with their coordinates reported in two formats:

- **Pose landmarks:** The x and y coordinates are normalized to [0.0, 1.0] based on the image width and height respectively. The z coordinate represents the landmark depth, with the depth at the midpoint of hips serving as the origin. A smaller z value indicates closer proximity to the camera. The magnitude of z roughly aligns with the scale of x.

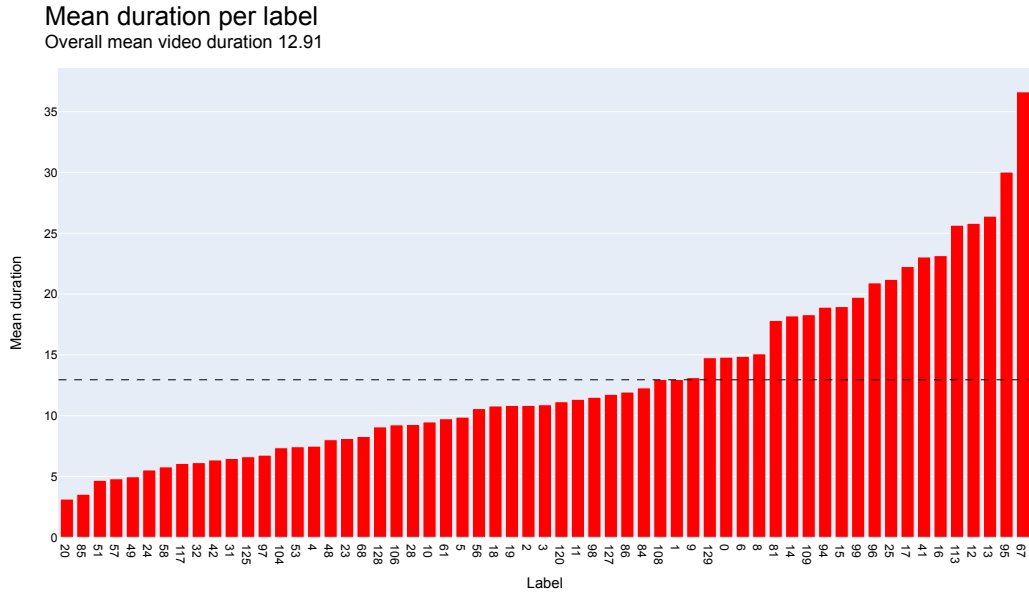


Figure 5.2: Average duration of clips per class.

- **Pose world landmarks:** These are real-world 3D coordinates in meters, with the origin positioned at the center between the hips.

In both cases, to each joint is assigned a visibility level in range $[0.0,1.0]$ which indicates the likelihood of the landmark being visible (present and not occluded) in the image. For our experiments, we opted for the *pose world landmark* format. BlazePose offers customization options to adjust precision levels through three parameters:

- **min_detection_confidence:** This sets the minimum confidence value (ranging from 0.0 to 1.0) from the person-detection model for successful detection.
- **min_tracking_confidence:** This establishes the minimum confidence value (ranging from 0.0 to 1.0) from the landmark-tracking model for successful tracking of pose landmarks. Failure to meet this confidence threshold triggers automatic person detection on the subsequent input image. Setting a higher value enhances solution robustness but may increase latency.
- **model_complexity:** This parameter determines the complexity of the pose landmark model, with options of 0, 1, or 2. Higher model complexities generally yield greater landmark accuracy but may also increase inference latency.

Observing the quality of the sequences extracted with various combinations of parameters and the time needed to obtain them, we decided to set the three parameters mentioned above to 0.8, 0.8 and 2 respectively. In this way we have more reliable sequences, where jittering is reduced, and which do not require too long times (around 7 fps).

5.4 Data transformation

Subsequently, to address frames where the skeleton was not identified, we implemented linear interpolation to estimate the joint coordinates. Specifically, when encountering an interval of empty frames, we calculated interpolation starting from the last frame before the interval and proceeding to the frame immediately following it according to the functions:

```
def linear_interpolation(frame1, frame2, alpha):
    interpolated_frame = (1 - alpha) * frame1 + alpha * frame2
    return interpolated_frame

def create_transition_frames(frame1, frame2, num_transition_frames):
    transition_frames = []
    for alpha in np.linspace(0, 1, num_transition_frames):
        interpolated_frame = linear_interpolation(frame1, frame2, alpha)
        transition_frames.append(interpolated_frame)
    return np.array(transition_frames)
```

where *num_transition_frames* is the number of empty frames detected, *frame1* and *frame2* are the last frame before the empty interval and the first one after the interval respectively, and *alpha* is the interpolation factor in $[0,1]$. In cases where the empty range occurred at the beginning or end of the dataframe, we simply removed the affected frames. The final data transformation step involved reducing the number of joints from 33 (figure 5.3a) to 15 (figure 5.3b) to reduce the use of useless joints (such as those of the face) with respect to pose recognition. Specifically, we:

- Derived the *Head* joint by averaging facial landmarks.
- Computed the *Neck* and *Pelvis* joints by determining the midpoints of the shoulders and hips respectively.
- Excluded all hand and foot joints except for the wrists and ankles.

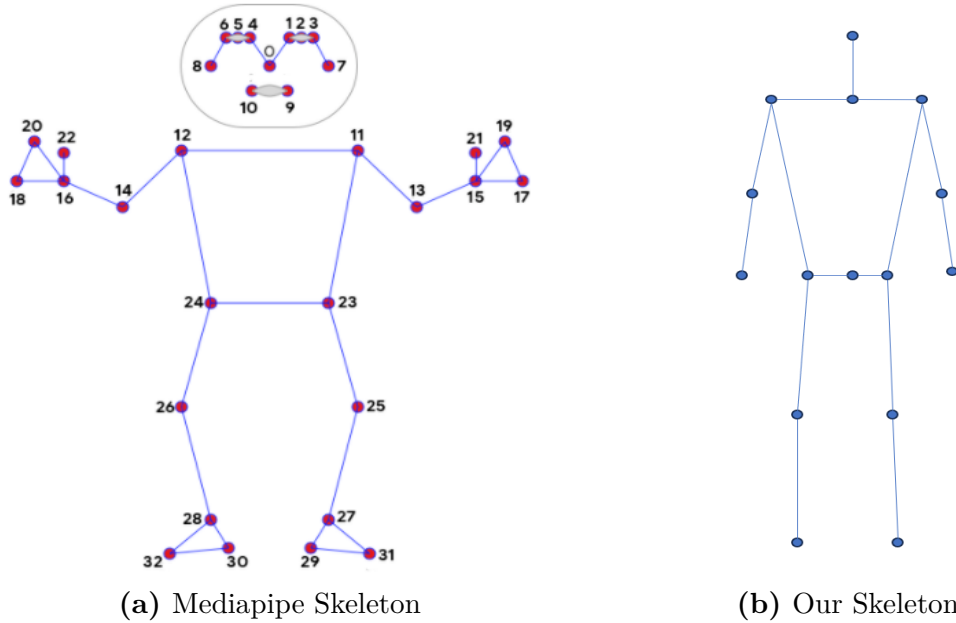


Figure 5.3: Skeleton comparison. On the left the original skeleton returned by BlazePose, on the right our skeleton where only relevant joints are kept.

5.5 Training & test sets

To create the training and testing splits in our dataset, we utilized YouTube video IDs. Specifically, we ensured that, for each asana, if there were multiple sequences extracted from the same video, they were all included in the same split. To achieve a 70/30 proportion between training and testing, we followed these steps:

1. Counted the sequences by video ID grouped by label.
2. Within each label, we sorted the video IDs in descending order by sequence count.
3. Calculated the cumulative sum of sequences.
4. Inserted into training the sequences of video IDs for which the cumulative sum was less than 70% of the total number of sequences associated with the label.

5.6 Synthetic dataset pipeline

The idea of adapting a dataset for human action recognition (HAR) into a dataset for the Temporal Action Segmentation (TAS) task was developed based on the

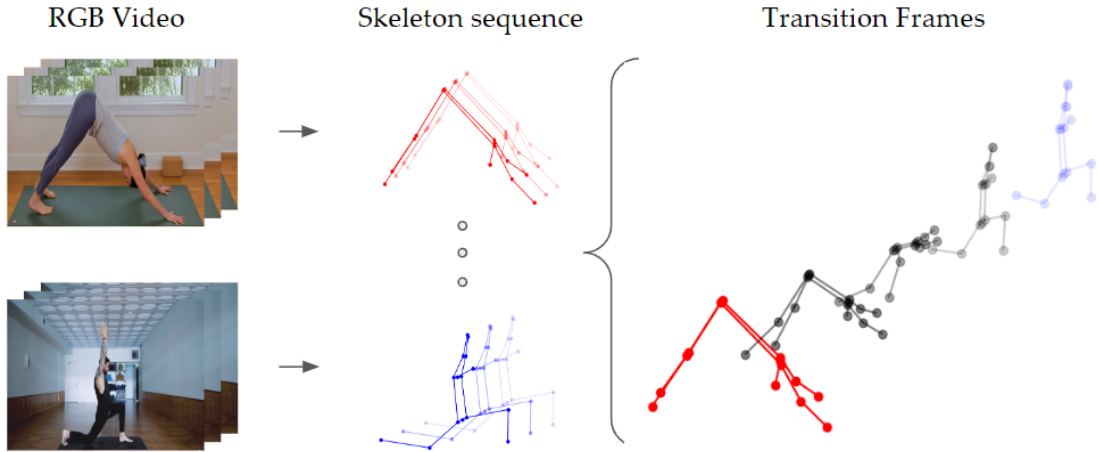


Figure 5.4: Synthetic dataset pipeline. Starting two RGB video we extract the 3D skeletal sequences C_1 (red) and C_2 (blue). Then, given the last frame of C_1 and the first frame of C_2 we reconstruct the transition frames trough linear interpolation.

strategy used in section 5.4 to reconstruct skeletons in frames where BlazePose failed to detect them. Given two clips $C_1 = \{f_{1i}, i = 1, \dots, n\}$ and $C_2 = \{f_{2j}, j = 1, \dots, m\}$, we can create a new clip $C_3 = C_1 \cup \{f_1, \dots, f_k\} \cup C_2$, where frames f_1, \dots, f_k are k frames calculated via linear interpolation starting from f_{c1n} and f_{c21} (figure 5.4). By repeating this operation, we can concatenate multiple videos sequentially, generating complete video lessons.

To build our datasets, we utilized three parameters whose values are reported in table 5.3:

- **activity_repeated** (ar): indicates whether an exercise can be repeated in a video lesson;
- **activity_per_video** (apv): number of exercises within the video lesson;
- **transition_frames** (tf): number of frames inserted between two clips via interpolation.

in the experiments section the impact of all the individual components on the performance of the segmentation network is studied.

Utilizing the training and testing split described in section 5.5, we constructed the video lessons according to the following steps:

1. Select the *apv* classes among those available with reinsertion if *ar = True* or without if *ar = False*.
2. For each *apv* class chosen, select a video ID and then a sequence associated with that video ID.

3. Concatenate the selected sequences via interpolation by inserting tf frames.

Each instance of the dataset consists of 1000 video lessons, with 700 built from the training clips and 300 from the test ones.

Param	Values
ar	[True, False]
apv	[5,10,15]
tf	[0,10,20,50]

Table 5.3: Values parameters for synthetic dataset

Chapter 6

Benchmark and experiments

To assess the effectiveness of our approach, we conducted numerous experiments in both the realms of Human Activity Recognition (HAR) and Temporal Action Segmentation (TAS). This chapter provides a detailed overview of our findings and their implications.

6.1 Human Activity Recognition

In our activity recognition experiments, as described in section 3.1.3, we utilized four networks: ST-GCN [1], MS-AAGCN [2], CTR-GCN [3], and DG-STGCN [4], to establish a benchmark on our dataset. In the following we explain the implementation details and the results that we obtained.

6.1.1 Implementation details

All experiments were conducted under identical conditions. Specifically, the networks were trained for 150 epochs using Stochastic Gradient Descent (SGD) with momentum set to 0.9 and weight decay to $5e^{-4}$. The initial learning rate was initialized to 0.1, and CosineAnnealing was employed as the scheduler.

6.1.2 Results

The results presented in Table 6.1 depict the performance achieved by training and testing on all available classes within our 3DYogSeg dataset. As anticipated, ST-GCN, often considered as the baseline in various works, exhibits the lowest performance among the networks evaluated. However, it is noteworthy that MS-AAGCN surprisingly outperforms the newer CTR-GCN and DG-STGCN models. The unexpected superiority of MS-AAGCN over DG-STGCN, which places third,

can be attributed to the dataset size. As discussed in [4], the advantage of the DG-GCN module becomes evident with larger datasets, whereas for smaller datasets (i.e., less than 1/4 of the NTU120-XSub training set), manually defined and refined topologies prove beneficial for spatial modeling of skeleton data. The advantage observed with MS-AAGCN over CTR-GCN may be attributed to the STC attention module, which enables the network to discern importance levels for joints, frames, and channels.

Model	Acc_top1	Acc_top5	F1_weighted	F1_macro
ST-GCN [1]	0.8558	0.9889	0.8380	0.8501
MS-AAGCN [2]	0.8986	0.9873	0.8850	0.8950
CTR-GCN [3]	0.8764	0.9937	0.8598	0.8760
DG-STGCN [4]	0.8685	0.9826	0.8520	0.8664

Table 6.1: 3DYogSeg activity recognition benchmarks. MS-AAGCN outperforms the other sota networks on our dataset.

6.1.3 Failure Cases

Among the poses that the network has the most difficulty in identifying are:

1. Eagle pose (85), which is classified 27% of the time as chair pose (2).
2. Table top pose (57), which is confused with the cat-cow (96) pose. This is probably due to the lack of a joint that identifies the movement of the spine, as the only difference between the two poses is in the movement of the head and the arching of the spine itself
3. Mountain pose (48) and mountain pose straches (128). Here too the difference is minimal, if in the first case the pose is completely static, in the second dynamism is inserted perhaps with the rotation of the neck or with a slow movement of the arms.

Case 1 and case 2 are reported in figures 6.2 and 6.3 respectively.

6.1.4 Comparison with 3DYoga90

The comparison between our dataset and the 3DYoga90 dataset revealed interesting insights when training the best performing network, MS-AAGCN, on common classes between the two datasets under four different scenarios. The results presented in Table 6.2 illustrate these scenarios. When the training and testing sets are

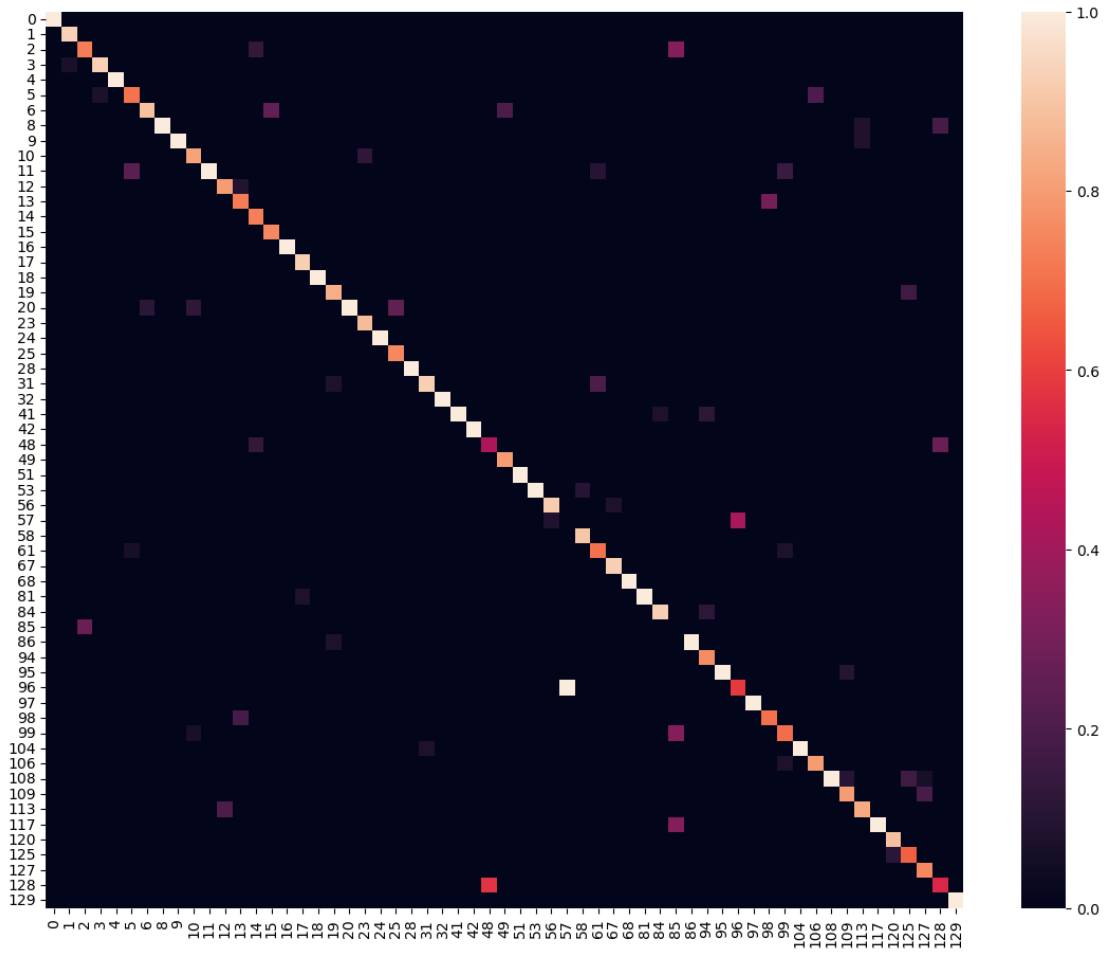


Figure 6.1: MS-AAGCN confusion matrix



Figure 6.2: Comparison between eagle and chair poses



Figure 6.3: Comparison between table top and cat-cow poses

train	test	acc_top1	acc_top5	F1_weighted	F1_macro
3DYogSeg	3DYogSeg	0.9390	0.9956	0.9185	0.9361
3DYoga90	3DYoga90	0.9146	0.9815	0.9032	0.9140
3DYogSeg	3DYoga90	0.6097	0.8846	0.6786	0.6063
3DYoga90	3DYogSeg	0.8597	0.9872	0.8566	0.8481

Table 6.2: Results comparison when MS-AAGCN is trained and tested on a different combination of dataset. The experiments are made only on common classes between the two datasets.

derived from the same dataset, we observe relatively similar performances, with accuracy top_1 scores of 0.94 and 0.92, and F1 weighted scores of 0.92 and 0.90, respectively, when trained on 3DYogSeg or 3DYoga90. However, notable differences emerge when training and testing are conducted on opposite datasets. Specifically, when the model is trained on 3DYoga90 and evaluated on 3DYogSeg samples, although there is a slight degradation in performance, the results remain relatively valid, with accuracy top1 and F1 weighted scores of 0.86. On the contrary, when the datasets are inverted, the accuracy drops significantly to 0.61 in top1 and 0.68 in F1 weighted. Further analysis of the videos present in the test sets of the five asanas with the largest drop in F1 score between the two scenarios revealed an interesting pattern. In the 3DYoga90 clips, a substantial portion of the footage comprised preparation for the pose, while our clips predominantly captured the person holding the pose from start to finish, with minimal preparation. For instance, in the Wild Things pose videos, MS-AAGCN predictions trained on 3DYogSeg and tested on 3DYoga90 were not entirely incorrect, considering the content of the video and the duration of the Wild Things pose compared to the predicted pose, which was still present in the video. This observation, coupled with the lower performances when both training and testing were conducted on 3DYoga90, suggests that the latter

dataset presents greater challenges compared to ours.

6.2 Temporal Action Segmentation

Through the Temporal Action Segmentation (TAS) experiments, we aimed to verify: i) whether linear interpolation could be used to adapt a Human Action Recognition (HAR) dataset for the TAS task, ii) what is the best combination of `activity_repeated`, `activity_per_video`, and `transition_frames` to build a synthetic dataset for optimizing the performance of the segmentation network. This section reports the experimental settings (section 6.2.1), quantitative and qualitative results (section 6.2.2), the results mapped on the hierarchy proposed by [115] (section 6.2.3) and finally the quantitative results obtained applying the same pipeline on 3DYoga90.

6.2.1 Experimental settings

The network used for the Temporal Action Segmentation (TAS) task is the Multi-stage Spatial-temporal graph convolutional neural network, or MS-GCN [5]. The architecture, proposed by Filtjens et al. in 2022, combines the convolutional capabilities of ST-GCN graphs with the segmentation capabilities of MS-TCN. The model is divided into two modules: the first generates an initial prediction for each frame by exploiting a modified version of ST-GCN blocks, while the second redefines the detected segments to reduce oversegmentation errors.

For each experiment, we created an instance of a synthetic dataset constructed according to the method described in the following section. As test videos, we segmented 6 video lessons obtained from YouTube, with their duration, number of poses, and average pose duration given in table 6.3.

Video id	Duration	Tags	Avg length
j7rKKpwdXNE	12:16	17	42,00
W2hRlGrgoUQ	8:39	22	21,13
B4kNiCWT17M	22:22	39	33,05
yetbSrCW1TQ	22:01	84	15,48
3zOwMzKV_gU	17:34	121	8,55
sOjXETbN32g	23:35	80	17,29

Table 6.3: Yoga lesson statistics. For each lesson the are reported the video id, the duration in minute and seconds, the number of tags within the lesson and the average length of a tag in seconds.

Metrics Following the methodology outlined in [5], we adopt the practice of quantitatively evaluating predictions against ground-truth annotations using a single evaluation metric per sample and another per segment. The sample-wise metric is the frame-wise accuracy and it is simply computed as the ratio of corrected labeled frames over the total number of frames. The segment-wise metric, instead, is the F1@k proposed by Lea et al. [104]. This metric has three characteristics:

1. it penalizes over-segmentation errors;
2. it does not penalize for minor temporal shifts between the predictions and ground truth, which may have been caused by annotator variability;
3. scores are dependent on the number actions and not on the duration of each action instance.

An action segment predicted by the model is first categorized as either a true positive (TP) or false positive (FP) by comparing its intersection over union (IoU) with the corresponding ground truth annotation. If the IoU surpasses a predefined threshold, it is labeled as a true positive segment (TP); otherwise, it is considered a false positive segment (FP). The count of false-negative segments (FN) in a trial is computed by subtracting the number of correctly predicted segments from the total number of segments marked by the experts. The final score is computed as:

$$F1 = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \quad (6.1)$$

The values reported in table 6.5 are computed as the average of the single metrics across the video lessons.

6.2.2 Results

Table 5.3 reports the values used for each parameter, while figures 6.5 and 6.4 reports the frame-wise accuracy and the segment-wise F1@0.5. The quantitative results are also reported in table 6.5. Below are some considerations on how the individual parameters impact performance.

Activity repeated Figure 6.5 shows how for a low number of asanas per video (apv=5), the network tends to prefer video lessons in which the poses are not repeated. By increasing the number of activities within the video, however, this difference tends to cancel out at the F1 level (figure 6.4, with apv=10 and apv=25) or even reverse when apv=25. We can therefore assume that by increasing the number of distinct exercises in the video lessons, performances tend to worsen.

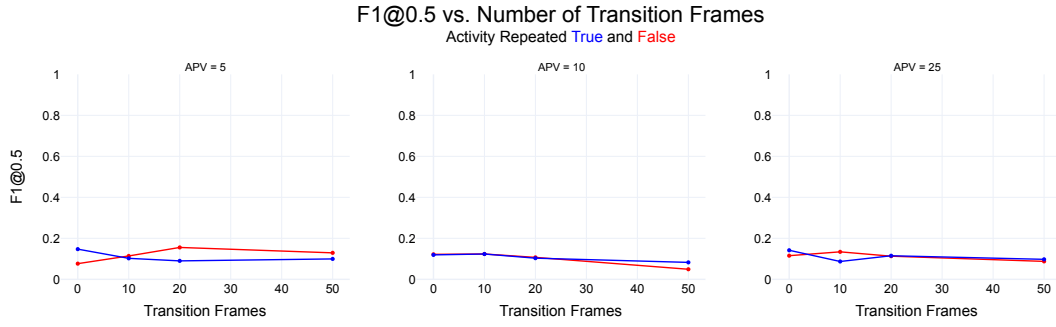


Figure 6.4: MS-GCN segment-wise F1@0.5 trained on 3DYogSeg. From left to right each graph represents how the metric varies for with respect to transition frames number with apv=5,10, and 25 respectively.

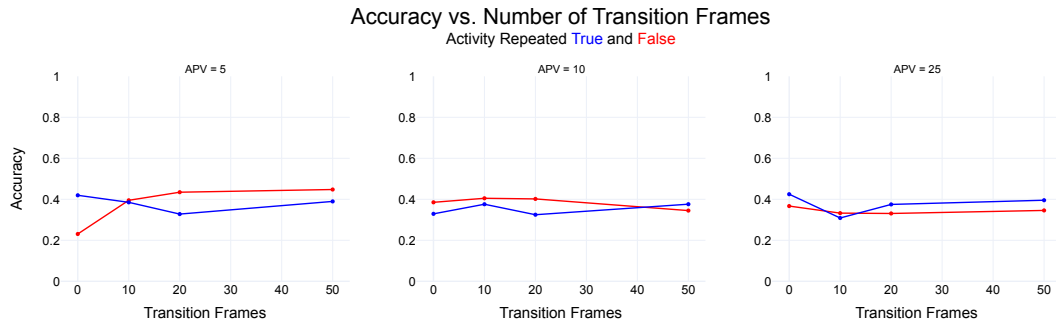


Figure 6.5: MS-GCN frame-wise accuracy trained on 3DYogSeg. From left to right each graph represents how the metric varies for with respect to transition frames number with apv=5,10, and 25 respectively.

Transition frames Observing how the performances change as the number of frames inserted between two clips varies, we can notice that:

- For apv=5, the graphs on the left in both Figure 6.5 and 6.4 show that increasing the number of tf results in better accuracy and F1@0.5 if ar=False. In particular, the highest metric values are reached for tf=20 and tf=50.
- For apv=10, the graphs in the center show that the model worsens in terms of F1@0.5 as the tf increases for both ar=False and ar=True (figure 6.4). In figure 6.5, however, we can see how for ar=False there is an initial improvement and then a worsening when tf increases, while for ar=True there is an oscillation between 0.37 and 0.32 (table 6.5).
- For apv=25, finally, it can be seen that in terms of F1@0.5, the performances

tend to worsen as tf increases, while in terms of accuracy for both values assumed by ar , the performances remain almost stable (0.35 for $ar=False$ and 0.39 for $ar=True$).

From this analysis we can understand how the network tends to prefer frames of smoother transitions for $apv=5$ and frames that immediately lead to the next asana if the video lesson contains many poses.

Activities per video Looking at figure 6.5, fewer activities per video seem to help the network achieve higher accuracy. In fact, once tf is fixed, we can see that, from left to right, a higher altitude is reached, especially when $ar=False$. If instead, we analyze figure 6.4, we can see that there is not a well-defined correlation between apv and $F1@0.5$. In fact, for both $ar=True$ and $ar=False$, given tf , the performances do not have a monotonically increasing or decreasing trend. However, as for accuracy, the highest peaks are reached for $apv=5$.

From our results and analyses, we observe that the optimal dataset configuration consists of pseudo-lessons containing a reduced number of exercises, with some degree of repetition, and an interval of transition frames ranging from 10 to 20. This approach strikes a balance between maintaining realistic body movements and simulating smooth transitions between poses, similar to those observed in actual yoga classes. During yoga video lessons, poses are typically performed without interruption, akin to a choreography. Additionally, YouTube fitness videos typically have frame rates between 24 and 30 frames per second (fps). Therefore, inserting a transition interval of 10 or 20 frames simulates a transition lasting approximately 0.4 or 0.8 tenths of a second, respectively. This duration captures semi-realistic movements while ensuring smooth transitions between poses. Furthermore, we analyzed the relationship between the number of unique exercises and the duration of video lessons. Comparing ratios obtained for both real and pseudo-lessons, we found that a smaller number of unique exercises within pseudo-lessons yields ratios more similar to those observed in real yoga lessons. Overall, our pipeline enables the creation of pseudo-lessons that closely resemble real yoga lessons. Consequently, datasets generated using our methodology serve as a practical alternative for the TAS task compared to datasets built entirely from scratch, which are time-consuming and challenging to collect. Our pipeline can generate 1000 pseudo-lessons in just half an hour, demonstrating its efficiency and scalability.

Figure 6.9 shows the qualitative results for each of the 6 test videoleasons, whose statistics are reported in table 6.3. In the image each label has been associated with a unique color, while in red are the background or unknown exercises in training. To make some considerations let us take in detail the lesson with id *B4kNiCWTU7M*, the third one shown in the figure 6.9. The network seems to be able to segment quite well where the poses are held for a long time for example

Parameter	Accuracy	F1@0.5
ar	0.0010	-0.0311
apv	-0.1526	0.0020
tf	0.1820	-0.4623

Table 6.4: Correlation matrix between syntethic dataset paramreters and metrics.

at the beginning and end of the lesson. However, there are a few cases where the segment is correctly identified but is associated with the wrong label. This is the case for the last segment (real 86, predicted 108, figure 6.6) or to the segment around frame 10,000 (real 61, predicted 99, figure 6.7). We can also find the same error in other segments, such as in lesson *j7rKKpwdXNE* (the fourth one in the figure) where from frame 10,000 to about frame 13,000 in ground truth we find poses 126 and 96, while in predictions only pose 96 (figure 6.8). As described in the next section, these are limiting cases since the poses are very similar to each other by global skeleton pose.



Figure 6.6: Corpse and Supine twist pose



(a) Low lunge pose



(b) Twisted low lunge pose

Figure 6.7: Low lunge and twisted low lunge pose



(a) Low lunge pose



(b) Twisted low lunge pose

Figure 6.8: Cat-cow and one legged table pose

ar	apv	tf	Accuracy	F1@0.5
False	5	0	0.2308	0.0765
False	5	10	0.3951	0.1141
False	5	20	0.4347	0.1556
False	5	50	0.4478	0.1292
False	10	0	0.3853	0.1219
False	10	10	0.405	0.1236
False	10	20	0.4019	0.1071
False	10	50	0.345	0.0492
False	25	0	0.3670	0.1154
False	25	10	0.3326	0.1341
False	25	20	0.3309	0.1129
False	25	50	0.3458	0.0876
True	5	0	0.4196	0.1473
True	5	10	0.3853	0.1024
True	5	20	0.3280	0.0900
True	5	50	0.3897	0.0996
True	10	0	0.3291	0.1193
True	10	10	0.3759	0.1235
True	10	20	0.3248	0.1032
True	10	50	0.3762	0.0827
True	25	0	0.4249	0.1418
True	25	10	0.3089	0.0871
True	25	20	0.3751	0.1150
True	25	50	0.3955	0.0976

Table 6.5: MS-GCN quantitative results with syntethic dataset built on 3DYogSeg.

Benchmark and experiments

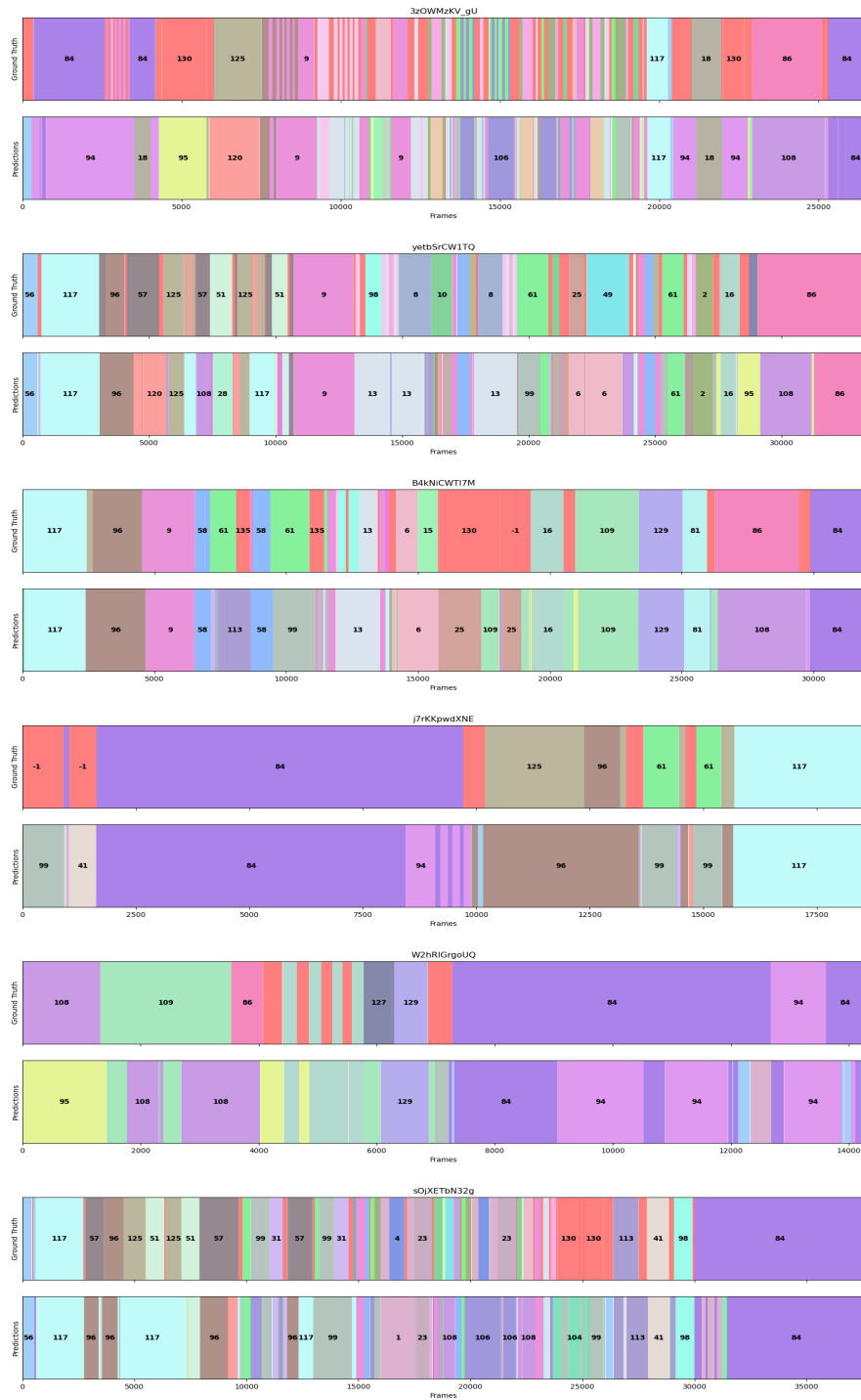


Figure 6.9: MS-GCN qualitative results trained on 3DYogSeg with $ar=False$, $apv=5$ and $tf=20$

6.2.3 Hierarchical results

Based on the recommendations from [115], [117], and [119], we organized the yoga exercises into macroclasses, which group poses based on their similarities. Figures 6.11 and 6.10 present the metrics for all three levels of the hierarchy, starting from the top down. Level 1 groups the poses in 6 classes [115]:

- Standing: The subject maintains an upright posture with either one or both legs on the ground. When only one leg is grounded, the other may be lifted, either held by one hand or freely in the air.
- Sitting: The subject is positioned on the ground with their hips resting on or very close to the ground, such as in the garland pose.
- Balancing: The subject balances their body weight on their palms, with both palms touching the ground and the rest of the body suspended in the air. The body is not in an inverted position.
- Inverted: The subject's body is positioned upside down, with the lower body either lifted in the air or close to the ground, as seen in poses like the plow pose.
- Reclining: The subject's body is reclined on the ground, with either the spine (upward-facing), stomach (downward-facing), or side of the body touching or very close to the ground. Alternatively, the subject's body may be positioned at a 180° angle alongside the ground, such as in plank poses.
- Wheel: The subject's body forms a half-circle or close to it on the ground. In poses where the body is facing upwards or downwards, both the palms and feet are in contact with the ground. In other variations, either only the hips or the stomach touches the ground.

Level 2 specialized the exercises in subgroups, while level 3 is the pose-level. The graphs depicted in Figures 6.10 and 6.11 illustrate the trend of F1@0.5 and accuracy metrics, respectively, as we vary the parameters of the synthetic dataset. It is evident that both metrics show an increasing trend as we move up the hierarchy. For instance, considering the triplet (False, 5, 20), which represents the best performance at the exercise level, we observe a notable improvement in accuracy from 0.44 to 0.62 and further to 0.73, while the F1@0.5 metric shows similar enhancement from 0.16 to 0.22 and ultimately to 0.30 (see Table 6.6). This indicates that as the dataset hierarchy becomes more refined, the model's performance improves accordingly. This observation suggests that even when the network misclassifies a segment, it tends to assign it a label that is not entirely dissimilar to the ground truth pose. This is consistent with the inherent similarity between yoga poses, making the segmentation task more challenging.

ar	apv	tf	accuracy	F1@0.5	level
False	5	20	0.730801	0.295597	1
False	5	20	0.618311	0.221597	2
False	5	20	0.434653	0.155561	3

Table 6.6: Hierarchical results for MS-GCN trained on a dataset built with parameters (False,5,20). The table shows how raising the hierarchy the performances improve

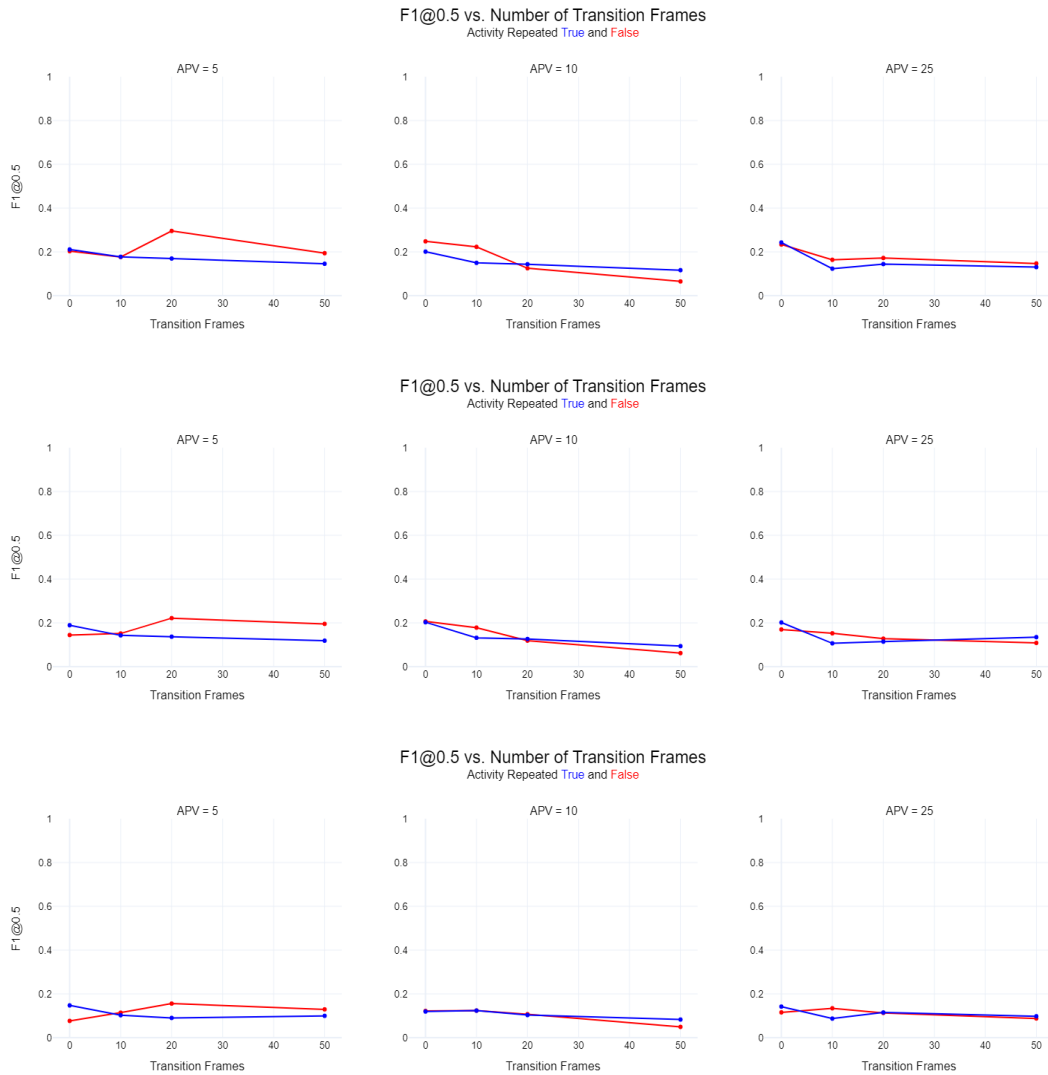


Figure 6.10: MS-GCN segment-wise F1@0.5 trained on 3DYogSeg. From top to bottom level 1, level 2 and exercise-level

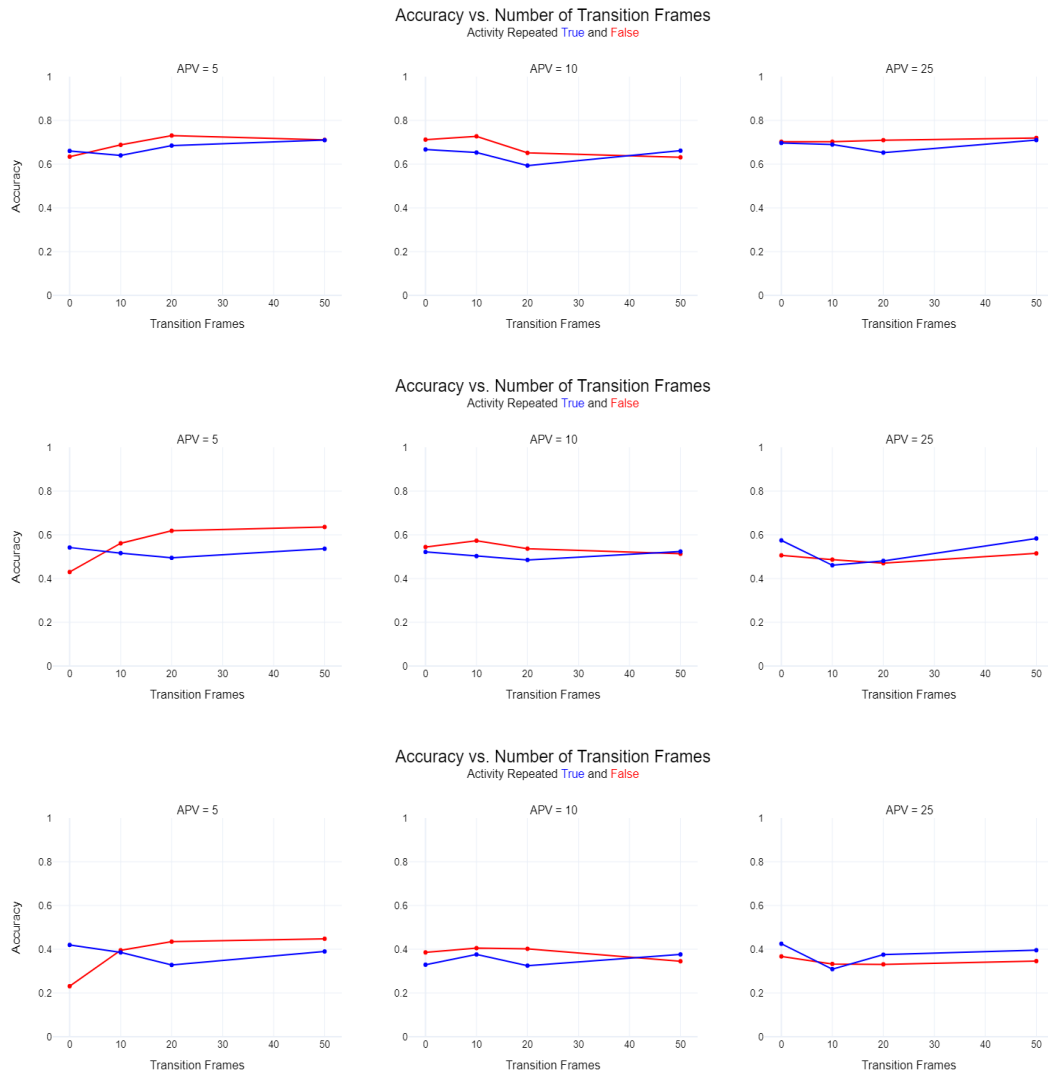


Figure 6.11: MS-GCN frame-wise accuracy trained on 3DYogSeg. From top to bottom level 1, level 2 and exercise-level

6.2.4 Comparison with 3DYoga90

ar	apv	tf	accuracy	F1@0.5
False	5	0	0.2042	0.0610
False	5	10	0.2052	0.0342
False	5	20	0.2339	0.0402
False	5	50	0.2277	0.0523
False	10	0	0.1898	0.0331
False	10	10	0.2152	0.0307
False	10	20	0.2431	0.0521
False	10	50	0.2320	0.0682
False	25	0	0.2557	0.0587
False	25	10	0.1867	0.0535
False	25	20	0.1439	0.0564
False	25	50	0.1791	0.0428
True	5	0	0.2304	0.0593
True	5	10	0.3307	0.0770
True	5	20	0.2216	0.0600
True	5	50	0.2821	0.0778
True	10	0	0.258	0.0561
True	10	10	0.2047	0.0882
True	10	20	0.1783	0.0250
True	10	50	0.232	0.0682
True	25	0	0.2405	0.0477
True	25	10	0.2486	0.0463
True	25	20	0.1691	0.0465
True	25	50	0.1782	0.0287

Table 6.7: MS-GCN trained on 3DYoga90

Due to the considerable resemblance between the datasets, we attempted to apply the human activity recognition (HAR) adaptation pipeline to temporal action segmentation (TAS) using the 3DYoga90 dataset. Table 6.7 presents results for the same set of parameters. However, the results obtained generally show lower performance compared to the synthetic datasets constructed from 3DYogSeg. Notably, the metrics reported exclude unknown exercises, which are more prevalent in 3DYoga90 compared to 3DYogSeg. This is attributed to the fact that 3DYogSeg was specifically curated to encompass all poses featured in the six video lessons. Moreover, it is important to consider that 3DYoga90 videos not only include the labeled asana but also encompass a preparation phase (often involving another exercise) and an exit phase. This additional complexity likely poses challenges for

the network in accurately identifying the start and end timestamps of the target exercise.

Nevertheless, it is interesting to observe that the highest accuracies and F1@0.5 scores are achieved with parameter sets aligned with the considerations outlined in section 6.2.2 for the results obtained with 3DYoga90. For instance, the highest accuracy is attained for the parameter triplet (True, 5, 10), where pseudo-video lessons feature a small number of distinct exercises (up to 5) relative to the class duration, interspersed with a limited number of transition frames (10). Similarly, the highest F1@0.5 score is obtained for the parameter set (True, 10, 10), where there are up to 10 distinct exercises, each separated by 10 frames.

These results, although inferior to those obtained with 3DYogSeg, suggest that once the characteristics of untrimmed videos are understood, it is feasible to construct reliable pseudo-videos using our approach. Additionally, the comparison of results across the two datasets indicates that initiating from clips containing only the reference pose facilitates the creation of synthetic datasets that enhance the performance of the segmentation network.

Chapter 7

Conclusions and future developments

With our research efforts, we have contributed to the field of Human Action Recognition (HAR) by introducing a novel dataset, 3DYogSeg, which differs from existing datasets in terms of data type (RGB video and skeletal sequences) and the number of yoga asanas included (58). Moreover, our study is the first to focus on the segmentation of yoga videos, proposing an innovative approach to create datasets for Temporal Action Segmentation (TAS) from those intended for HAR.

We collected YouTube videos based on pose names in multiple languages, resulting in 2115 clips. These clips were meticulously curated to ensure visibility of all joints, uninterrupted poses, and solitary individuals within the frame. Subsequently, we employed BlazePose to extract 3D skeletal sequences, yielding data for 58 unique asanas with an average clip duration of 12.91 seconds. Additionally, we compiled a text file for each asana containing video links and pose timestamps, facilitating independent image extraction and sequence creation.

Our pipeline for to adapt the HAR dataset to the TAS task involves linear interpolation to concatenate clips, thereby forming pseudo-lessons for TAS. By iteratively applying this process and customizing parameters such as *activity_per_video*, *transition_frames*, and *activity_repeated*, we can efficiently generate TAS datasets without the labor-intensive manual segmentation of videos.

To assess the dataset quality, we perform extensive experiments on both the HAR task and segmentation. As far as HAR, surprisingly, MS-AAGCN [2] outperformed ST-CGN [1], CTR-GCN [3], and DG-STGCN [4], despite expectations of its superiority over ST-GCN. We attribute these results to factors such as dataset size and the STC attention module implemented in MS-AAGCN.

Furthermore, we compared MS-AAGCN’s performance across four scenarios using exercises common to both 3DYogSeg and the recently published 3Dyoga90

dataset. While comparable results were obtained when training and testing on the same dataset, discrepancies arose when using different datasets, highlighting the dataset’s influence on model performance. Indeed, in the scenario where the model is trained on 3DYoga90 and tested on the same dataset, both accuracy and F1 weighted reach 0.86. However, there is a notable drop to 0.61 and 0.68, respectively, when the model is trained on our dataset and tested on 3DYoga90. Upon analyzing the videos from 3DYoga90, it was observed that, besides the labeled exercise, in some cases, another exercise occupies a significant portion of the clip. This discrepancy in labeling likely confuses the model trained on our dataset, leading it to predict an incorrect exercise, albeit one present in the clip.

On the other hand, experiments in the TAS domain have demonstrated that, it is feasible to construct synthetic datasets suitable for training a network, yielding good performances compared to the time required for manual dataset creation. Specifically, when employing a triplet of parameters that results in a few distinct exercises per lesson and a moderate number of transition frames (between 10 and 20), the MS-GCN network achieves optimal performance with frame-wise accuracy 0.4347 and F1@0.5 0.1556. To comprehend why a small number of distinct exercises and brief transitions yield superior results, we analyzed the traits of real video lessons. It was noted that these lessons also exhibited the same characteristics. Hence, it can be inferred that constructing the synthetic dataset to emulate the traits of real videos may enhance segmentation.

However, qualitative results reveal that while trained on pseudo-lessons from our dataset, the MS-GCN network occasionally correctly identifies the start and end timestamps but fails to recognize the correct exercise due to the high similarity between asanas.

In the following we suggest some possible future developments of our work.

Improving Transition Frame Construction One area for potential expansion involves enhancing the construction of transition frames within our adaptation pipeline. Currently, our methodology does not consider factors such as differences in body sizes between individuals when concatenating clips. To address this, we could create a standard skeleton to simulate lessons performed by a single person. Additionally, aligning all skeletons from individual clips with the same orientation could further enhance the realism of pseudo-lessons. By making these adjustments to closely mirror real scenarios, we anticipate potential improvements in segmentation network performance.

Validation of Adaptation Pipeline with Diverse Datasets Another avenue for further study is the validation of our adaptation pipeline using datasets from different domains. By analyzing this, we can better understand optimal guidelines for composing synthetic datasets. Moreover, if our methodology proves effective

across various domains, it could offer a less time-consuming alternative to traditional manual labeling. Additionally, testing our pipeline on domains where TAS datasets already exist would enable us to compare the performance of networks trained on original versus synthetic datasets.

Architecture Enhancement for MS-GCN A third variation of our work could involve modifying the architecture of MS-GCN [5]. We propose replacing blocks in the prediction stage, currently based on ST-GCN [1], with modules from MS-AAGCN [2]. This change aims to leverage MS-AAGCN’s superior pose recognition abilities, potentially leading to improved segmentation performance. Additionally, we suggest exploring a multi-stream prediction stage that utilizes not only first-order information such as joint coordinates but also connection lengths and speeds. Leveraging hierarchical annotations during the learning phase could further enhance segmentation performance, building upon the demonstrated effectiveness of such annotations in related tasks [115, 117, 119]

Acknowledgments Computational resources were provided by HPC@POLITO, a project of Academic Computing within the Department of Control and Computer Engineering at the Politecnico di Torino (<http://hpc.polito.it>)

Bibliography

- [1] Sijie Yan, Yuanjun Xiong, and Dahua Lin. «Spatial temporal graph convolutional networks for skeleton-based action recognition». In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018 (cit. on pp. 4, 23, 24, 28, 30, 31, 43, 55, 56, 72, 74).
- [2] Lei Shi, Yifan Zhang, Jian Cheng, and Hanqing Lu. «Skeleton-based action recognition with multi-stream adaptive graph convolutional networks». In: *IEEE Transactions on Image Processing* 29 (2020), pp. 9532–9545 (cit. on pp. 4, 24, 28, 32, 34, 35, 38, 55, 56, 72, 74).
- [3] Yuxin Chen, Ziqi Zhang, Chunfeng Yuan, Bing Li, Ying Deng, and Weiming Hu. «Channel-wise topology refinement graph convolution for skeleton-based action recognition». In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 13359–13368 (cit. on pp. 4, 24, 28, 36–39, 55, 56, 72).
- [4] Haodong Duan, Jiaqi Wang, Kai Chen, and Dahua Lin. «Dg-stgcn: Dynamic spatial-temporal modeling for skeleton-based action recognition». In: *arXiv preprint arXiv:2210.05895* (2022) (cit. on pp. 4, 24, 29, 38, 39, 55, 56, 72).
- [5] Benjamin Filtjens, Bart Vanrumste, and Peter Slaets. «Skeleton-based action segmentation with multi-stage spatial-temporal graph convolutional neural networks». In: *IEEE Transactions on Emerging Topics in Computing* (2022) (cit. on pp. 4, 40, 43, 44, 59, 60, 74).
- [6] W. Dorland. *Illustrated Medical Dictionary*. Saunders, Philadelphia, 1890 (cit. on p. 8).
- [7] Frank Rosenblatt. *The perceptron: A perceiving and recognizing automaton*. Report 85-460-1. Ithaca, New York: Project PARA, Cornell Aeronautical Laboratory, Jan. 1957 (cit. on p. 8).
- [8] CHIARA PLIZZARI. «Spatial temporal transformer networks for skeleton-based activity recognition». In: (2018) (cit. on pp. 8, 10, 21).
- [9] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA, USA: MIT Press, 1969 (cit. on p. 9).

- [10] John Duchi, Elad Hazan, and Yoram Singer. «Adaptive subgradient methods for online learning and stochastic optimization.» In: *Journal of machine learning research* 12.7 (2011) (cit. on p. 12).
- [11] Diederik P Kingma and Jimmy Ba. «Adam: A method for stochastic optimization.» In: *arXiv preprint arXiv:1412.6980* (2014) (cit. on p. 12).
- [12] Xavier Glorot and Yoshua Bengio. «Understanding the difficulty of training deep feedforward neural networks.» In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256 (cit. on p. 12).
- [13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. «Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.» In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034 (cit. on p. 12).
- [14] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. «Deep sparse rectifier neural networks.» In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 315–323 (cit. on pp. 12, 14).
- [15] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. «Dropout: a simple way to prevent neural networks from overfitting.» In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958 (cit. on p. 16).
- [16] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. «Gradient-based learning applied to document recognition.» In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324 (cit. on pp. 16, 18).
- [17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. «Imagenet classification with deep convolutional neural networks.» In: *Advances in neural information processing systems* 25 (2012) (cit. on pp. 18, 24).
- [18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. «Deep residual learning for image recognition.» In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778 (cit. on p. 18).
- [19] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. «Imagenet: A large-scale hierarchical image database.» In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255 (cit. on p. 18).
- [20] Marco Gori, Gabriele Monfardini, and Franco Scarselli. «A new model for learning in graph domains.» In: *Proceedings. 2005 IEEE international joint conference on neural networks, 2005*. Vol. 2. IEEE. 2005, pp. 729–734 (cit. on pp. 18, 21).

- [21] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. «The graph neural network model». In: *IEEE transactions on neural networks* 20.1 (2008), pp. 61–80 (cit. on pp. 18, 21).
- [22] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. «Spectral networks and locally connected networks on graphs». In: *arXiv preprint arXiv:1312.6203* (2013) (cit. on p. 18).
- [23] Mikael Henaff, Joan Bruna, and Yann LeCun. «Deep convolutional networks on graph-structured data». In: *arXiv preprint arXiv:1506.05163* (2015) (cit. on p. 18).
- [24] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. «Convolutional neural networks on graphs with fast localized spectral filtering». In: *Advances in neural information processing systems* 29 (2016) (cit. on p. 18).
- [25] William L Hamilton, Rex Ying, and Jure Leskovec. «Representation learning on graphs: Methods and applications». In: *arXiv preprint arXiv:1709.05584* (2017) (cit. on p. 19).
- [26] Peter D Hoff, Adrian E Raftery, and Mark S Handcock. «Latent space approaches to social network analysis». In: *Journal of the american Statistical association* 97.460 (2002), pp. 1090–1098 (cit. on p. 19).
- [27] Mikhail Belkin and Partha Niyogi. «Laplacian eigenmaps and spectral techniques for embedding and clustering». In: *Advances in neural information processing systems* 14 (2001) (cit. on p. 20).
- [28] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. «Distributed large-scale natural graph factorization». In: *Proceedings of the 22nd international conference on World Wide Web*. 2013, pp. 37–48 (cit. on p. 20).
- [29] Shaosheng Cao, Wei Lu, and Qionghai Xu. «Grarep: Learning graph representations with global structural information». In: *Proceedings of the 24th ACM international on conference on information and knowledge management*. 2015, pp. 891–900 (cit. on p. 20).
- [30] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. «Asymmetric transitivity preserving graph embedding». In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 2016, pp. 1105–1114 (cit. on p. 20).
- [31] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. «Deepwalk: Online learning of social representations». In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2014, pp. 701–710 (cit. on p. 20).

- [32] Aditya Grover and Jure Leskovec. «node2vec: Scalable feature learning for networks». In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 2016, pp. 855–864 (cit. on p. 20).
- [33] Will Hamilton, Zhitao Ying, and Jure Leskovec. «Inductive representation learning on large graphs». In: *Advances in neural information processing systems* 30 (2017) (cit. on pp. 20, 21).
- [34] Thomas N Kipf and Max Welling. «Semi-supervised classification with graph convolutional networks». In: *arXiv preprint arXiv:1609.02907* (2016) (cit. on pp. 21, 22, 31).
- [35] Thomas N Kipf and Max Welling. «Variational graph auto-encoders». In: *arXiv preprint arXiv:1611.07308* (2016) (cit. on pp. 21, 22).
- [36] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. «Modeling relational data with graph convolutional networks». In: *The semantic web: 15th international conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, proceedings* 15. Springer. 2018, pp. 593–607 (cit. on pp. 21, 22).
- [37] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. «Learning phrase representations using RNN encoder-decoder for statistical machine translation». In: *arXiv preprint arXiv:1406.1078* (2014) (cit. on pp. 22, 42).
- [38] Cailing Wang and Jingjing Yan. «A comprehensive survey of rgb-based and skeleton-based human action recognition». In: *IEEE Access* (2023) (cit. on p. 24).
- [39] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. «3D convolutional neural networks for human action recognition». In: *IEEE transactions on pattern analysis and machine intelligence* 35.1 (2012), pp. 221–231 (cit. on p. 24).
- [40] Haodong Duan, Yue Zhao, Kai Chen, Dahua Lin, and Bo Dai. «Revisiting skeleton-based action recognition». In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 2969–2978 (cit. on pp. 24, 28).
- [41] Daniil Osokin. «Real-time 2d multi-person pose estimation on cpu: Lightweight openpose». In: *arXiv preprint arXiv:1811.12004* (2018) (cit. on pp. 24, 25).
- [42] Valentin Bazarevsky, Ivan Grishchenko, Karthik Raveendran, Tyler Zhu, Fan Zhang, and Matthias Grundmann. «Blazepose: On-device real-time body pose tracking». In: *arXiv preprint arXiv:2006.10204* (2020) (cit. on pp. 24, 26, 49).

- [43] Chih-Chung Chang and Chih-Jen Lin. «LIBSVM: a library for support vector machines». In: *ACM transactions on intelligent systems and technology (TIST)* 2.3 (2011), pp. 1–27 (cit. on pp. 24, 26).
- [44] Xiaodong Yang, Chenyang Zhang, and YingLi Tian. «Recognizing actions using depth motion maps-based histograms of oriented gradients». In: *Proceedings of the 20th ACM international conference on Multimedia*. 2012, pp. 1057–1060 (cit. on pp. 24, 26).
- [45] Raviteja Vemulapalli, Felipe Arrate, and Rama Chellappa. «Human action recognition by representing 3d skeletons as points in a lie group». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 588–595 (cit. on pp. 24, 27).
- [46] Georgios Th Papadopoulos, Apostolos Axenopoulos, and Petros Daras. «Real-time skeleton-tracking-based human action recognition using kinect data». In: *MultiMedia Modeling: 20th Anniversary International Conference, MMM 2014, Dublin, Ireland, January 6-10, 2014, Proceedings, Part I 20*. Springer. 2014, pp. 473–483 (cit. on pp. 24, 27).
- [47] Yong Du, Wei Wang, and Liang Wang. «Hierarchical recurrent neural network for skeleton based action recognition». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1110–1118 (cit. on pp. 24, 27).
- [48] Inwoong Lee, Doyoung Kim, Seoungyoon Kang, and Sanghoon Lee. «Ensemble deep learning for skeleton-based action recognition using temporal sliding lstm networks». In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 1012–1020 (cit. on pp. 24, 27).
- [49] Yong Du, Yun Fu, and Liang Wang. «Skeleton based action recognition with convolutional neural network». In: *2015 3rd IAPR Asian conference on pattern recognition (ACPR)*. IEEE. 2015, pp. 579–583 (cit. on pp. 24, 27).
- [50] Qihong Ke, Mohammed Bennamoun, Senjian An, Ferdous Sohel, and Farid Boussaid. «A new representation of skeleton sequences for 3d action recognition». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 3288–3297 (cit. on pp. 24, 27).
- [51] Carlos Caetano, Jessica Sena, François Brémond, Jefersson A Dos Santos, and William Robson Schwartz. «Skelemotion: A new representation of skeleton joint sequences based on motion information for 3d action recognition». In: *2019 16th IEEE international conference on advanced video and signal based surveillance (AVSS)*. IEEE. 2019, pp. 1–8 (cit. on pp. 24, 28).
- [52] Catalin Ionescu, Fuxin Li, and Cristian Sminchisescu. «Latent structured models for human pose estimation». In: *2011 International Conference on Computer Vision*. IEEE. 2011, pp. 2220–2227 (cit. on p. 25).

- [53] Alexander Toshev and Christian Szegedy. «Deeppose: Human pose estimation via deep neural networks». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 1653–1660 (cit. on p. 25).
- [54] Joao Carreira, Pulkit Agrawal, Katerina Fragkiadaki, and Jitendra Malik. «Human pose estimation with iterative error feedback». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 4733–4742 (cit. on p. 25).
- [55] Yu Chen, Chunhua Shen, Xiu-Shen Wei, Lingqiao Liu, and Jian Yang. «Adversarial posenet: A structure-aware convolutional network for human pose estimation». In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 1212–1221 (cit. on p. 25).
- [56] Alejandro Newell, Kaiyu Yang, and Jia Deng. «Stacked hourglass networks for human pose estimation». In: *Computer Vision—ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part VIII 14*. Springer. 2016, pp. 483–499 (cit. on p. 25).
- [57] Hao-Shu Fang, Shuqin Xie, Yu-Wing Tai, and Cewu Lu. «Rmpe: Regional multi-person pose estimation». In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2334–2343 (cit. on p. 25).
- [58] Sijin Li and Antoni B Chan. «3d human pose estimation from monocular images with deep convolutional neural network». In: *Computer Vision—ACCV 2014: 12th Asian Conference on Computer Vision, Singapore, Singapore, November 1–5, 2014, Revised Selected Papers, Part II 12*. Springer. 2015, pp. 332–347 (cit. on p. 25).
- [59] Ching-Hang Chen and Deva Ramanan. «3d human pose estimation= 2d pose estimation+ matching». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7035–7043 (cit. on p. 25).
- [60] Camillo Lugaresi et al. «Mediapipe: A framework for building perception pipelines». In: *arXiv preprint arXiv:1906.08172* (2019) (cit. on pp. 25, 49).
- [61] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. «YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors». In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2023, pp. 7464–7475 (cit. on p. 26).
- [62] Lawrence Rabiner and Biinghwang Juang. «An introduction to hidden Markov models». In: *ieee assp magazine* 3.1 (1986), pp. 4–16 (cit. on p. 27).
- [63] Jun Liu, Gang Wang, Ling-Yu Duan, Kamila Abdiyeva, and Alex C Kot. «Skeleton-based human action recognition with global context-aware attention LSTM networks». In: *IEEE Transactions on Image Processing* 27.4 (2017), pp. 1586–1599 (cit. on p. 27).

- [64] Wentao Zhu, Cuiling Lan, Junliang Xing, Wenjun Zeng, Yanghao Li, Li Shen, and Xiaohui Xie. «Co-occurrence feature learning for skeleton based action recognition using regularized deep LSTM networks». In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 30. 1. 2016 (cit. on p. 27).
- [65] Chao Li, Qiaoyong Zhong, Di Xie, and Shiliang Pu. «Skeleton-based action recognition with convolutional neural networks». In: *2017 IEEE international conference on multimedia & expo workshops (ICMEW)*. IEEE. 2017, pp. 597–600 (cit. on p. 27).
- [66] Carlos Caetano, François Brémont, and William Robson Schwartz. «Skeleton image representation for 3d action recognition based on tree structure and reference joints». In: *2019 32nd SIBGRAPI conference on graphics, patterns and images (SIBGRAPI)*. IEEE. 2019, pp. 16–23 (cit. on p. 28).
- [67] Hong Liu, Juanhui Tu, and Mengyuan Liu. «Two-stream 3d convolutional neural network for skeleton-based action recognition». In: *arXiv preprint arXiv:1705.08106* (2017) (cit. on p. 28).
- [68] Jian Liu, Naveed Akhtar, and Ajmal Mian. «Skepxels: Spatio-temporal image representation of human skeleton joints for action recognition.» In: *CVPR workshops*. 2019, pp. 10–19 (cit. on p. 28).
- [69] Lei Shi, Yifan Zhang, Jian Cheng, and Hanqing Lu. «Two-stream adaptive graph convolutional networks for skeleton-based action recognition». In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 12026–12035 (cit. on pp. 28, 32).
- [70] Fanfan Ye, Shiliang Pu, Qiaoyong Zhong, Chao Li, Di Xie, and Huiming Tang. «Dynamic gcn: Context-enriched topology learning for skeleton-based action recognition». In: *Proceedings of the 28th ACM international conference on multimedia*. 2020, pp. 55–63 (cit. on p. 28).
- [71] Ke Cheng, Yifan Zhang, Congqi Cao, Lei Shi, Jian Cheng, and Hanqing Lu. «Decoupling gcn with dropgraph module for skeleton-based action recognition». In: *Computer Vision—ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIV 16*. Springer. 2020, pp. 536–553 (cit. on p. 28).
- [72] Haojun Xu, Yan Gao, Zheng Hui, Jie Li, and Xinbo Gao. «Language knowledge-assisted representation learning for skeleton-based action recognition». In: *arXiv preprint arXiv:2305.12398* (2023) (cit. on p. 29).
- [73] Dongjingdin Liu, Pengpeng Chen, Miao Yao, Yijing Lu, Zijie Cai, and Yuxin Tian. «TSGCNeXt: Dynamic-Static Multi-Graph Convolution for Efficient Skeleton-Based Action Recognition with Long-term Learning Potential». In: *arXiv preprint arXiv:2304.11631* (2023) (cit. on p. 29).

- [74] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. «Learning convolutional neural networks for graphs». In: *International conference on machine learning*. PMLR. 2016, pp. 2014–2023 (cit. on p. 30).
- [75] Ziyu Liu, Hongwen Zhang, Zhenghao Chen, Zhiyong Wang, and Wanli Ouyang. «Disentangling and unifying graph convolutions for skeleton-based action recognition». In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 143–152 (cit. on p. 39).
- [76] Guodong Ding, Fadime Sener, and Angela Yao. «Temporal action segmentation: An analysis of modern techniques». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2023) (cit. on pp. 40, 42).
- [77] Zheng Shou, Dongang Wang, and Shih-Fu Chang. «Temporal action localization in untrimmed videos via multi-stage cnns». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 1049–1058 (cit. on p. 41).
- [78] Tianwei Lin, Xiao Liu, Xin Li, Errui Ding, and Shilei Wen. «Bmn: Boundary-matching network for temporal action proposal generation». In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 3889–3898 (cit. on p. 41).
- [79] Jernej Barbič, Alla Safonova, Jia-Yu Pan, Christos Faloutsos, Jessica K Hodgins, and Nancy S Pollard. «Segmenting motion capture data into distinct behaviors». In: *Proceedings of Graphics Interface 2004*. Citeseer. 2004, pp. 185–194 (cit. on p. 41).
- [80] Emily B Fox, Michael C Hughes, Erik B Sudderth, and Michael I Jordan. «Joint modeling of multiple time series via the beta process with application to motion capture segmentation». In: (2014) (cit. on p. 41).
- [81] Ke Zhang, Wei-Lun Chao, Fei Sha, and Kristen Grauman. «Video summarization with long short-term memory». In: *Computer Vision—ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part VII 14*. Springer. 2016, pp. 766–782 (cit. on p. 41).
- [82] Dimitri Zhukov, Jean-Baptiste Alayrac, Ramazan Gokberk Cinbis, David Fouhey, Ivan Laptev, and Josef Sivic. «Cross-task weakly supervised learning from instructional videos». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 3537–3545 (cit. on p. 41).
- [83] Noureldien Hussein, Efstratios Gavves, and Arnold WM Smeulders. «Timeception for complex action recognition». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 254–263 (cit. on p. 41).

- [84] Noureldien Hussein, Efstratios Gavves, and Arnold WM Smeulders. «Pic: Permutation invariant convolution for recognizing long-range activities». In: *arXiv preprint arXiv:2003.08275* (2020) (cit. on p. 41).
- [85] Mike Zheng Shou, Stan Weixian Lei, Weiyao Wang, Deepti Ghadiyaram, and Matt Feiszli. «Generic event boundary detection: A benchmark for event segmentation». In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 8075–8084 (cit. on p. 41).
- [86] Florent Perronnin, Jorge Sánchez, and Thomas Mensink. «Improving the fisher kernel for large-scale image classification». In: *Computer Vision—ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5–11, 2010, Proceedings, Part IV 11*. Springer. 2010, pp. 143–156 (cit. on p. 42).
- [87] Joao Carreira and Andrew Zisserman. «Quo vadis, action recognition? a new model and the kinetics dataset». In: *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 6299–6308 (cit. on p. 42).
- [88] Luowei Zhou, Chenliang Xu, and Jason Corso. «Towards automatic learning of procedures from web instructional videos». In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018 (cit. on p. 42).
- [89] Alexander Richard, Hilde Kuehne, and Juergen Gall. «Weakly supervised action learning with rnn based fine-to-coarse modeling». In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2017, pp. 754–763 (cit. on p. 42).
- [90] Yifei Huang, Yusuke Sugano, and Yoichi Sato. «Improving action segmentation via graph-based temporal reasoning». In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 14024–14034 (cit. on p. 42).
- [91] Peng Lei and Sinisa Todorovic. «Temporal deformable residual networks for action segmentation in videos». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 6742–6751 (cit. on p. 42).
- [92] Dipika Singhania, Rahul Rahaman, and Angela Yao. «C2F-TCN: A framework for semi-and fully-supervised temporal action segmentation». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2023) (cit. on p. 42).
- [93] Yazan Abu Farha and Jurgen Gall. «Ms-tcn: Multi-stage temporal convolutional network for action segmentation». In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 3575–3584 (cit. on pp. 42, 43, 45).

- [94] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. «Attention is all you need». In: *Advances in neural information processing systems* 30 (2017) (cit. on p. 42).
- [95] Alexander Richard, Hilde Kuehne, Ahsan Iqbal, and Juergen Gall. «Neuralnetwork+ viterbi: A framework for weakly supervised video learning». In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2018, pp. 7386–7395 (cit. on pp. 42, 43).
- [96] Fadime Sener and Angela Yao. «Unsupervised learning and segmentation of complex activities from video». In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 8368–8376 (cit. on pp. 42, 43).
- [97] Anna Kukleva, Hilde Kuehne, Fadime Sener, and Jurgen Gall. «Unsupervised learning of action classes with continuous temporal embedding». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 12066–12074 (cit. on p. 42).
- [98] Zhenzhi Wang, Ziteng Gao, Limin Wang, Zhifeng Li, and Gangshan Wu. «Boundary-aware cascade networks for temporal action segmentation». In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXV 16*. Springer. 2020, pp. 34–51 (cit. on p. 42).
- [99] Yuchi Ishikawa, Seito Kasai, Yoshimitsu Aoki, and Hirokatsu Kataoka. «Alleviating over-segmentation errors by detecting action boundaries». In: *Proceedings of the IEEE/CVF winter conference on applications of computer vision*. 2021, pp. 2322–2331 (cit. on p. 43).
- [100] Guodong Ding and Angela Yao. «Temporal action segmentation with high-level complex activity labels». In: *IEEE Transactions on Multimedia* (2022) (cit. on p. 43).
- [101] Zexing Du, Xue Wang, Guoqing Zhou, and Qing Wang. «Fast and unsupervised action boundary detection for action segmentation». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 3323–3332 (cit. on p. 43).
- [102] Guodong Ding and Angela Yao. «Leveraging action affinity and continuity for semi-supervised temporal action segmentation». In: *European Conference on Computer Vision*. Springer. 2022, pp. 17–32 (cit. on p. 43).
- [103] Li Ding and Chenliang Xu. «Weakly-supervised action segmentation with iterative soft boundary assignment». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 6508–6516 (cit. on p. 43).

- [104] Colin Lea, Michael D Flynn, Rene Vidal, Austin Reiter, and Gregory D Hager. «Temporal convolutional networks for action segmentation and detection». In: *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 156–165 (cit. on pp. 43, 60).
- [105] Dipika Singhania, Rahul Rahaman, and Angela Yao. «Iterative contrast-classify for semi-supervised temporal action segmentation». In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 2. 2022, pp. 2262–2270 (cit. on p. 43).
- [106] Jun Li and Sinisa Todorovic. «Action shuffle alternating learning for unsupervised action segmentation». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 12628–12636 (cit. on p. 43).
- [107] Rahul Rahaman, Dipika Singhania, Alexandre Thiery, and Angela Yao. «A generalized and robust framework for timestamp supervision in temporal action segmentation». In: *European Conference on Computer Vision*. Springer. 2022, pp. 279–296 (cit. on p. 43).
- [108] Jun Li, Peng Lei, and Sinisa Todorovic. «Weakly supervised energy-based learning for action segmentation». In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 6243–6251 (cit. on p. 43).
- [109] Alexander Richard, Hilde Kuehne, and Juergen Gall. «Action sets: Weakly supervised action segmentation without ordering constraints». In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2018, pp. 5987–5996 (cit. on p. 43).
- [110] Zijia Lu and Ehsan Elhamifar. «Set-supervised action learning in procedural task videos via pairwise order consistency». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 19903–19913 (cit. on p. 43).
- [111] Mohsen Fayyaz and Jurgen Gall. «Sct: Set constrained temporal transformer for set supervised action segmentation». In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 501–510 (cit. on p. 43).
- [112] Jun Li and Sinisa Todorovic. «Set-constrained viterbi for set-supervised action segmentation». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 10820–10829 (cit. on p. 43).

- [113] Saquib Sarfraz, Naila Murray, Vivek Sharma, Ali Diba, Luc Van Gool, and Rainer Stiefelhagen. «Temporally-Weighted Hierarchical Clustering for Unsupervised Action Segmentation». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2021, pp. 11225–11234 (cit. on p. 43).
- [114] Fisher Yu and Vladlen Koltun. «Multi-scale context aggregation by dilated convolutions». In: *arXiv preprint arXiv:1511.07122* (2015) (cit. on p. 44).
- [115] Manisha Verma, Sudhakar Kumawat, Yuta Nakashima, and Shanmuganathan Raman. «Yoga-82: a new dataset for fine-grained classification of human poses». In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*. 2020, pp. 1038–1039 (cit. on pp. 46, 59, 67, 74).
- [116] Santosh Kumar Yadav, Aayush Agarwal, Ashish Kumar, Kamlesh Tiwari, Hari Mohan Pandey, and Shaik Ali Akbar. «YogNet: A two-stream network for realtime multiperson yoga action recognition and posture correction». In: *Knowledge-Based Systems* 250 (2022), p. 109097 (cit. on p. 46).
- [117] Jianwei Li, Haiqing Hu, Jinyang Li, and Xiaomei Zhao. «3D-Yoga: A 3D Yoga Dataset for Visual-Based Hierarchical Sports Action Analysis». In: *Proceedings of the Asian Conference on Computer Vision*. 2022, pp. 434–450 (cit. on pp. 47, 67, 74).
- [118] Santosh Kumar Yadav, Guntaas Singh, Manisha Verma, Kamlesh Tiwari, Hari Mohan Pandey, Shaik Ali Akbar, and Peter Corcoran. «YogaTube: a video benchmark for Yoga action recognition». In: *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2022, pp. 1–8 (cit. on p. 47).
- [119] Seonok Kim. «3DYoga90: A Hierarchical Video Dataset for Yoga Pose Understanding». In: *arXiv preprint arXiv:2310.10131* (2023) (cit. on pp. 47, 67, 74).