Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"
Corso di Laurea Magistrale in Data Science

Self-supervised learning for action segmentation using a Transformer architecture.

Relatore
Prof. **Lamberto Ballan**
Correlatrice
Dr. **Mariella Dimiccoli**

Candidato
Emanuele Mincato
Matricola
2019044

Anno Accademico 2022-2023
Data ufficiale di Laurea 22/09/2023

Abstract

The focus of this project is to address the problem of Temporal Action Segmentation (TAS), which consist in temporally segment and classify fine-grained actions in untrimmed videos. The enhancement of this procedure represents a significant albeit intricate challenge. Some of the main challenges for this problem are that different actions can occur with different speed or duration, also some of them can be ambiguous and overlap. Successfully addressing this challenge can yield substantial advancements in various domains of work, including robotics, medical support technologies, surveillance and many more.

Currently, the best performing *state-of-the-art* methods are fully-supervised. Consequently, they require huge annotation cost, are not scalable and not suited for applications where data collection is costly. To alleviate this problem, we propose a self-supervised transformer-based method for action segmentation, that does not require action labels, and demonstrate the effectiveness of the learned weights in a weakly-supervised setting. Precisely we built a Siamese architecture based on an improvement version of an already existing Transformer architecture. To validate our approach, we performed an ablation study and compared our results with the *state-of-the-art* to draw some conclusion.

All the work is done using Pythorch as deep learning framework. The reason for this choice are multiple like array-based programming, automatic differentiation to automate the calculation of derivatives, open source ecosystem, and of strong library as *'torchvision'* and *'torch'*.

Indice

Elenco delle figure

Capitolo 1

Introduction

The realm of artificial intelligence is an expanding domain that has captured the interest of numerous scholars and innovators due to its remarkable potentials. These capabilities have been enhanced by the increase in computational power and the abundance of available data.

Presently, computer vision stands out as a potent and captivating category of AI that finds utility in a broad spectrum of scenarios. Instances such as autonomous vehicles or the identification of cancer exhibit noteworthy and valuable applications of computer vision. Within this sphere lies a collection of relatively well-defined tasks pertaining to computer vision, encompassing techniques to process and comprehend digital images, generating meaningful insights. One of these tasks includes temporal action segmentation.

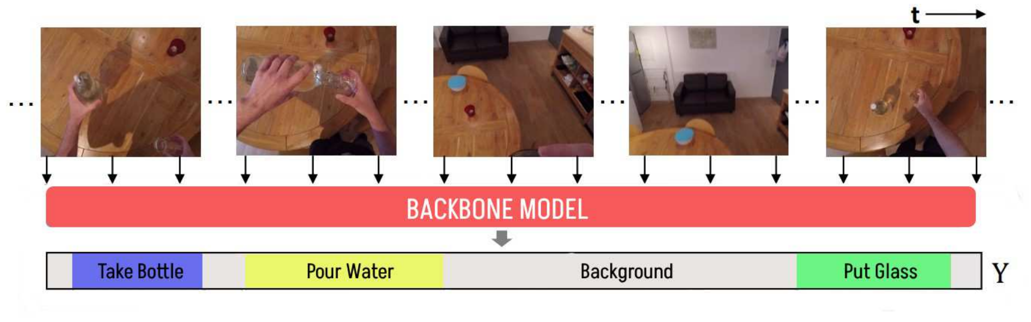

Figura 1.1: Example of Temporal Action Segmentation problem.

1.1 Temporal Action Segmentation

Temporal Action Segmentation (TAS) is a problem in the field of computer vision and video analysis. In essence the challenge of Temporal Action Segmentation revolves around automatically dividing a video into action segments each having precisely determined temporal boundaries. Finding a solution to this issue has implications, for video comprehension, content retrieval and various applications that heavily rely on accurate temporal action information. Some real-life scenario where these methods can be implemented are surveillance cameras[1], action recognition[2] and human computer interaction[3].

To formally define the Temporal Action Segmentation problem we consider an input video sequence V consisting of T frames. The objective is to detect and locate instances of actions within the video. Each action instance has its starting and ending timestamps indicating its temporal extent. The segmentation process entails dividing the video into overlapping segments with each segment representing a coherent action. It is crucial to determine the temporal boundaries of these segments in order to provide an accurate description of when each action occurs.

Addressing the Temporal Action Segmentation problem requires solving several challenges, including:

Action Variation: Actions can vary in terms of their appearance, speed, duration and context. Effective segmentation methods should consider these variations. Be able to apply across different instances of the same action.

Temporal Localization: Accurately determining the boundaries of actions is crucial for action segmentation. This involves identifying the frame or timestamp where an action starts and ends.

Overlapping Actions: Videos often feature actions happening at the same time or one after another causing segments to overlap. It is important to handle cases properly in order to achieve accurate segmentation.

Scalability: Real world videos can have lengths so a segmentation algorithm should be able to handle both short and long videos effectively.

Data Annotation: Creating training data for Temporal Action Segmentation usually involves annotating boundaries and assigning action labels for each segment. This process is time consuming. Requires expertise in the domain.

Researchers have tackled these challenges by employing techniques such as deep learning architectures, temporal modeling, motion analysis and attention mechanisms. Methods like convolutional networks (TCNs)[4] recurrent neural networks (RNNs)[5][6] and Transformers[7][8] have been utilized to capture temporal dependencies and patterns, in videos. Furthermore the evaluation and comparison of approaches, in terms of accuracy and efficiency rely on benchmark datasets. As the task needs a frame-wise classification, the models need to classify all the frames within a video and so the final prediction will be performed on the original temporal resolution. For this case are used mainly two approaches; the first is the use of an encoder-decoder model, while the second is the use of an architecture that can process a video maintaining its length unchanged. Both methods have some advantages and some drawbacks. The first one helps decreasing oversegmentation errors but the reduction of the temporal length, inside the encoder, can lead to loss of relevant information, which could compromise in the final prediction. The second one is commonly based on dilated convolutions, this models can acquire large receptive fields without increasing the number of parameters, neither shrinking the video's temporal length. All these methods will be discuss better later, on the Related works chapter.

1.2 Objectives

The goal of this project is to build a new model to perform Temporal Action Segmentation by minimizing the amount of labeled data required for this task. Specifically, the aim is to learn in an weakly-supervised fashion a representation space where we could use a simple clustering algorithms, like *K-means*, to classify each frame correctly. Moreover we wanted to test whether the Transformer architecture is a competitive model for solving the TAS problem.

1.3 Summary of work done

After a deep review of the actual *state-of-the-art* methods for Temporal Action Segmentation, we identified transformer-based approaches as the most efficient ones, and proposed a self-supervised transformer-based method to address the problem in an weakly-supervised fashion.

We did not only deeply analyze the *state-of-the-art* models, but we also dedicated some work to better understand the dataset used for this type of problem.

After this first part of studying the problem, we started building our solution, for the case of the weakly-supervised scenario. Initially we improved an already existing architecture[7], and then we used it as a backbone of our model. Then, to solve Temporal Action Segmentation problem, we built a Siamese architecture which scope is to distinguish between video of the same activities. The new features representation found by the Siamese architecture play a key role to solve this problem.

Moreover, to make our architecture work properly, we also applied *curriculum learning*, a way to make the training of the network more efficient[9][10].

Finally, we validated the architecture through an ablation study, optimized performance through hyper-parameter tuning and made a comparison to the actual *state-of-the-art*.

Capitolo 2

Background

This chapter introduces the history and the technical background of the modern deep learning algorithms with a focus on the Transformer architecture, a key model for this work.

2.1 Artificial Neural Network

Artificial neural networks (ANNs), sometimes abbreviated as neural networks (NNs) or neural nets, represent a subset of machine learning models that are built using principles of neuronal organization, discovered by connectionism in the biological neural networks constituting animal brains. The approach within this methodology involves grouping multiple artificial neurons (also referred to as perceptrons or nodes) to generate a final predictive outcome. Neural Networks aim to find abstract relationships within data, for classification purposes. To achieve this, each neuron learns a specific set of parameters (weights) to aggregate signals from neurons in the preceding layer, ultimately yielding a response. To generate predictions, networks incorporate a final linear layer that transforms the accumulated features into a distribution of class scores.

2.1.1 Perceptron

The perceptron was introduced by American psychologist and computer scientist Frank Rosenblatt in the late 1950s. It was inspired by the biological neuron and its simplified mathematical model. Rosenblatt aimed to develop a computational model that could mimic certain aspects of human brain function, such as pattern recognition and decision-making.

In 1958, Rosenblatt published a paper titled "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain"[11]. In this paper, he described the basic structure of the perceptron and its learning algorithm. The perceptron is a very simple algorithm consists of the following components: inputs, weights, summation function and activation function.

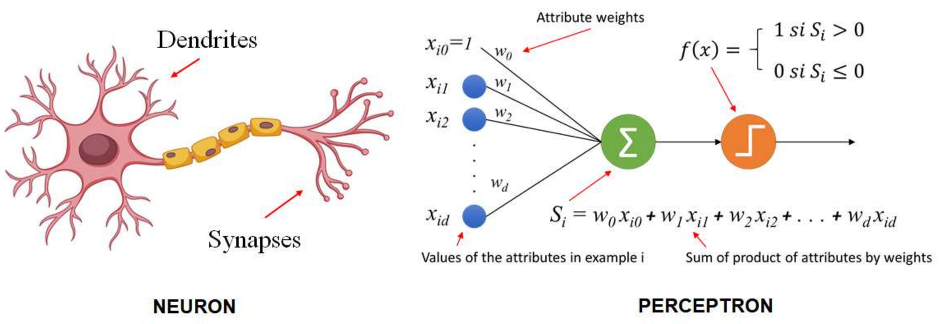
Dendrites
Synapses
NEURON
Attribute weights
$x_{i0}=1$
w_0
x_{i1}
w_1
x_{i2}
w_2
w_d
x_{id}
$f(x) = \begin{cases} 1 \; si \; S_i > 0 \\ 0 \; si \; S_i \leq 0 \end{cases}$
$S_i = w_0 x_{i0} + w_1 x_{i1} + w_2 x_{i2} + \ldots + w_d x_{id}$
Values of the attributes in example i
Sum of product of attributes by weights
PERCEPTRON

Figura 2.1: Perceptron architecture.

While the perceptron showed promise in solving simple classification tasks, it was limited to linearly separable problems. This limitation led to initial skepticism about the perceptron's capabilities and to a decline of interest in the 1960s after a research by Marvin Minsky and Seymour Papert[12] highlighted the its limitations and its inability to solve certain non-linear problems.

It wasn't until later advancements in neural network research, with the development of multi-layer networks (multi-layer perceptrons or MLPs) and more sophisticated activation functions, that neural networks regained attention and became the powerful machine learning models we know today[13][14].

To conclude, the perceptron is a foundational concept in neural network history, representing an early attempt to model artificial neurons and their learning capabilities.

2.2 Transformer

Nowadays there are a lot of powerful machine learning algorithms able to solve complex task, among these there is also the Transformer architecture, which is the core architecture used in this work.

The Transformer architecture is a groundbreaking deep learning model that revolutionized the field of natural language processing (NLP) and other sequence-to-sequence tasks. It was introduced in the paper "Attention Is All You Need" by Vaswani et al. [15] in 2017 and has since become the foundation for many *state-of-the-art* models in NLP.

Before the Transformer, recurrent neural networks and their variants, such as long short-term memory networks, were commonly used for sequence tasks like machine translation, text generation, and speech recognition. However, RNNs have limitations in parallelism and struggle to capture long-range dependencies in sequences. The Transformer architecture emerged as a solution to these challenges by introducing a novel approach to handling sequences through the use of self-attention mechanisms.

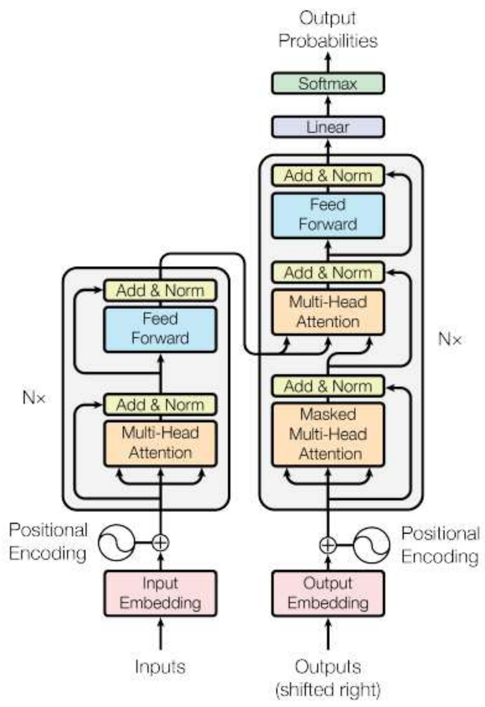

Figura 2.2: Original Transformer model architecture.

The main components of the Transformer Architecture, as we can see also from Figure 2.2, are:

- **Self-Attention Mechanism**: The core innovation of the Transformer is the self-attention mechanism, which allows the model to weigh the importance of different "words" in a sequence relative to each other. This enables the model to consider the context of each word in relation to all other words in the sequence, capturing long-range dependencies effectively. The formula for the self-attention, that in the main paper[15] is called "Scaled Dot-Product Attention", is the following:

  $$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}}) \cdot V$$

  Here Q, K, V stand for *Query*, *Key*, *Value* and they are linear projections of the input embeddings. Briefly, *queries* represent the positions for which we want to compute attention weights, *keys* help in determining how well a query aligns with different positions in the input and finally *values* contain information that will be combined according to the attention weights computed from the queries and keys.

- **Multi-Head Attention**: The Transformer employs multi-head attention, where the self-attention mechanism is applied multiple times with different learned weights. This allows the model to focus on different aspects of the input data in parallel, enhancing its ability to learn complex relationships.

- **Positional Encoding**: Since the Transformer does not have a built-in notion of sequence order (unlike RNNs), positional encoding is added to the input embedding to provide the model with information about the position of each token in the sequence. The positional encoding have the same dimension d_{model} as the embedding, so that the two can be summed. There are many choices of positional encoding, in the original paper[15] is used a positional encoding based on sine and cosine functions of different frequencies:

$$\overrightarrow{p}_t^{(i)} = f(t)^{(i)} = \begin{cases} \sin(\omega_k \cdot t) & : if \quad i = 2k \\ \cos(\omega_k \cdot t) & : if \quad i = 2k+1 \end{cases}$$

where

$$\omega_k = \frac{1}{10000^{2k/d}}$$

- **Encoder-Decoder Architecture**: The Transformer's original design includes both an encoder and a decoder. The encoder processes the input sequence, while the decoder generates the output sequence. This architecture is widely used for sequence-to-sequence tasks like machine translation.

- **Residual Connections and Layer Normalization**: The Transformer uses residual connections (skip connections) and layer normalization to stabilize and accelerate training. These techniques help alleviate the vanishing gradient problem and improve the flow of gradients during training.

- **Feed-forward Neural Networks**: Each layer of the Transformer contains feed-forward neural networks that operate independently on each position in the sequence. These networks contribute to the model's ability to capture complex patterns in the data.

Capitolo 3

Related work

There has been plenty of work done in the area of action segmentation for untrimmed videos. Most of them are based on a fully supervised scenario, which relies on a complete frame annotations, or weakly supervised approaches that use some form of metadata. However, to counter the dependence on annotated data, interest in unsupervised approaches has grown. All these approach and the most important *state-of-the-art* methodologies in this field are reported in the paper: "Temporal Action Segmentation: An Analysis of Modern Techniques"[16]. Here we will report only most efficient methods for each approach.

3.1 Fully-Supervised approaches

In a fully-supervised context, comprehensive action labels are assigned to each frame within training video sequences. The process of collecting these dense labels is the most time-consuming, requiring annotators to review all videos for their entirety. Alternatively, in a semi-supervised scenario, annotation effort is reduced by densely annotating only a subset of videos, while treating the remaining ones as unlabeled samples. In this context the main approaches are Temporal Convolutional Networks (TCNs), Improving Existing Architectures and the use of the Transformer architecture.

For what concern TCNs the idea behind these methods is to capture temporal patterns through a series of hierarchical convolutional layers. Lea et al. [17] implemented an encoder-decoder architecture with 1D temporal convolutional and deconvolutional kernels in order to capture long-range temporal patterns. Moreover they prove that TCN-based solutions are fast compared to other methods. The reasons because this method is fast is that, even if Lea et al. said that they work on a *fullresolution*, they applied strong downsampling to the frames of the videos to speed-up the process, but this type of preprocessing may cause the loss of important details. To overcome this issue Yazan Abu Farha and Juergen Gall[4] used a multi-stage architecture (MS-TCN) maintaining the temporal resolutions and expands the receptive field with progressively larger dilated convolutions. This strategy for sure increase the computational cost, but it allows to preserve better the temporal information especially the boundary information between actions. This two approaches are shown in Figure 3.2.

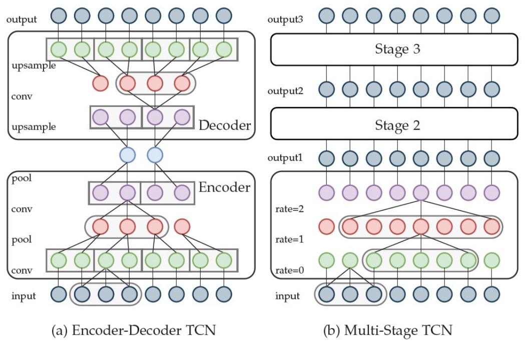

Figura 3.1: Temporal Convolutional Networks (TCN) for temporal action segmentation.

Other fully-supervised approaches, but much less explored and used, are the ones based on Transformer architecture. In this category there are two main works: UVAST[8] and ASFormer[7]. The reason Transformer architecture are implemented for Temporal Action Segmentation is due to attention mechanism. In fact thanks to the encoder, the transformer can attends frames within the inputs, which is often referred as self-attention (SA). Moreover the ability of the Transformer is confirmed by X. Wang et al.[18] where it compares the ability of his network to find non-local relationship (that is nothing more than temporal information) to the self-attention mechanism inside the Transformers. Transformers are experiencing a gradual acceptance in the context of TAS; however, their application remains constrained. The main reason for that is that transformers lack inductive biases, necessitating huge datasets (collection of videos) for optimal training. Furthermore, another problem found by [19] is that the self-attention mechanism could struggle to assign significant weights across extensive input ranges. F. Yi et al.[7] try to solve these problem with their ASFormer architecture. For this reason their architecture is very important for this project, and will be resumed later, in fact part of our work is to transform this model into an weakly-supervised method at activity level. Finally the last fully-supervised approaches are the ones which tries to improve the already existing architectures. The main work done in this case is Fast Inference Approximation for Action (FIFA) by Y. Souri et al.[20]. They were able to improve the results of most of the works mentioned above (MS-TCN, UVAST and ASFormer), in some cases they improved the accuracy up to 4%. But the most important features is that their methods is able to improve the speed of the previous ones, in some cases by more than 5 times while maintaining similar performance. The main core of their work is to defines a differentiable energy function to approximate the probabilities of possible segment alignments, instead of using the Viterbi algorithm that is much more computational demanding.

3.2 Weakly-Supervised approaches

Comparatively, weak labels necessitate less annotation effort compared to dense video labels. The weak labels encompass various forms, such as lists of actions or action sets for individual frames, as well as activity labels. Single-frame annotations, also referred to as time-stamp annotations, are sparsely labeled frames, essentially presenting an ordered list of actions associated with representative frames. The objective of weakly supervised techniques is to avoid intensive frame-level supervision. A. Yao et al.[16] divide this approaches in five main categories. Here, is briefly described each one of them, for further information consult the original paper.

- **Transcripts of ordered list of actions**:
  Transcript-based supervision involves only the actions occurring in a video along with their sequential arrangement. This method substantially cuts down the expenses associated with labeling videos, as it eliminates the need for annotating each frame individually. These approaches are divided into two overarching categories: iterative two-stage methods[21] and single-stage[22] solutions.

- **Action Set**:
  An action set entails a grouping of action labels provided for training purposes, without including information about their specific timing, sequence, or occurrence frequency. This form of labeling can resemble meta-tags attached to videos on platforms for sharing video content, such as video-sharing platforms. There are many works in this category like the one of A. Richard et al. [23], J. Li et al[24] and M. Fayyaz et al. [25]

- **Single-Frame Supervision**:
  In this case, instead of labelling all the frame of the videos, is labelled only one single frame for each action inside the video. There is not a specific strategy to choose the frame for each action, could it be done casually. But this will reduce a lot the effort of annotating each videos. Here the most important work of this scenario are the ones of Z. Li et al. [26] and A. Richard et al. [23]

- **Narrations & Subtitles**:
  This category refers to all methods which use textual data within the videos. In this case text data could be subtitles or some text which is describing the actions or the frame in the video. The main drawbacks of this category is that the methods assume that all the video and the text are correctly temporally aligned and, moreover, they need that all input videos have their owns text, not always the case. Nevertheless there are multiple works in this category such as the one of Sener et al.[27] and Fried et al. [28]

- **Activity Supervision**:
  The models inside this category assumes that the activity of which each video is part of is a given information. Despite being considering to be weakly supervised, these methods uses the same amount of information as the majority of unsupervised works. Also the work done during this project can been considered of this category.

3.3 Unsupervised approaches

Regarding the unsupervised setup, it revolves around collections of videos that showcase identical activities. However, it isn't entirely label-free, as it hinges on the presence of activity labels to assemble these video collections. Therefore, the unsupervised setup aligns with weak activity label guidance in terms of label information, yet these two setups diverge in the treatment of video collections during training. To elaborate, unsupervised approaches focus on one group of videos with the same activity at a time, whereas activity label supervision operates concurrently across videos from all activities. The unsupervised approach is the most difficult since it assumes that the videos are completely label-free, even if almost all of them are considering each activity per time, but is the the most useful since in real-life scenario most of the videos are unlabeled. Is worthy to mention that in this case no one used a Transformer architecture or at least no one was able to get competitive results with it.
Since these methods uses unlabeled data is needed to use the Hungarian matching to get the metrics like accuracy. This is a bit absurd because Hungarian matching needed the ground truth of the frame, but is one of the few solutions to compare different models. Also in this case there are a lot of works done, but most of them works at video levels[29] [30] [31] [32] , few of them at activity level [33] [34] (the scope of this work) and none of them at global level. The problem of working at global level in this scenario is that, if you classify a video in a wrong activity, is very likely to classify all the frames inside the video wrongly. The only way to classify them correctly is that the misclassify activity and the right activity share some common actions between them.
One of the first work done on unsupervised scenario is the one of Sener and Yao[35]. They proposed a general Mallow Model(gMM) to model the sequential structures of actions, the drawbacks is that the gMM cannot depict repeated actions since the ordering of actions is considered as a permutable sequence of steps. To improve the performance and especially the flexibility of the model, A. Kukleva et al.[36]; proposed a method that first learn continuous frame-wise temporal representation and afterwards each frame is clustered and finally the video order is found using the Viterbi algorithm.

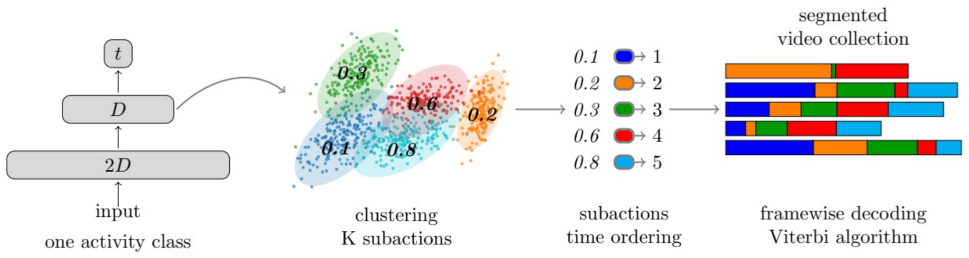

Figura 3.2: Proposed architecture by Kukleva et al.

The final method that is worth to cite in the unsupervised case is the one of M. Saquib Sarfraz et al.[32]. They proposed a model called Temporally-Weighted Hierarchical Clustering (TW-FINCH), the advantage of this model is that it doesn't require any training phase since it is directly applied to the precomputed features to find out the action boundaries. To work it applies temporally weighted hierarchical clustering to group together similar video frames. This method seems to outperform the previous ones, the only problem is that it work only at video level and it can not be extend to activity level, our focus for this project.

Capitolo 4

Methodology

In this chapter, we describe our contribution and the methodology used to build our proposed self-supervised approach. First, we describe a transformer-based architecture that we improved and used as building block to build a Siamese architecture that we used in a self-supervised fashion to learn action representations. We describe in details the structure of our architecture and how it works. Finally, we detail how we implemented curriculum learning to train the network in a efficient mode.

4.1 AS Former: problems and improvements

The first part of this work is to verify and see if it is possible and useful to implement a Transformer architecture for Temporal Action Segmentation in a weakly-supervised scenario. To do so we did not start from scratch but we started from an already existing architecture: the AS Former of F. Yi et al.[7].

The ASFormer utilizes an encoder-decoder structured Transformer. Given the pre-computed features of the video, the encoder will first predict the action probability of each frames. After that the output of the encoder is passed thought different decoders which scope is to refine the action prediction and improve the results. The original architecture can be seen in Fig. 4.1.

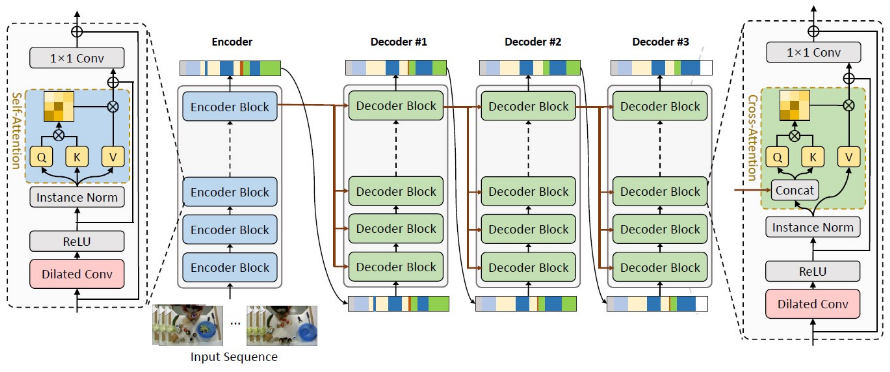

Figura 4.1: Original Transformer architecture proposed by F. Yi et al.

The main part of this architecture, as for every Transformer, are the encoder and the decoder. Here is briefly described their implementation since it will be used for this work.

Encoder

The input of the encoder are the pre-computed feature and they are passed in a fully connected layer, in order to adjust the dimension of the input feature. After this layer a series of encoder blocks are concatenated. Finally, a fully connected layer will compute the first output predictions that will be given to the decoder as input. The main difference in the encoder block, respect to the vanilla Transformer, is the use of a dilated temporal convolution instead of point-wise fully connected layer. This is due to give more importance to the temporal relationship between features, in fact every action occupies continued timestamps in the input video.

Decoder

The decoder here is used to refine the output of the encoder. The idea is to use the cross-attention to encode better the temporal relations among multiple action segments. The cross-attention mechanism allows every position in the decoder to attend over all positions in the refinement process by generating the attention weights. So the goal of the decoders is to reduce the weight of external information to avoid the problem of error accumulation. The use of the decoders in their case seems very important to boost their results, and the use of multiple decoders, for their case four, improve the results respect to using only one. For this work the decoder part is not implemented, this because it needs the ground truth of every frames to work properly and to do the refinement, an information that we assumed we did not have.

4.1.1 Problems

There are different problem of implementing a Transformer architecture for TAS. As F. Yi et al.[7] pointed out the following ones as the main concern about implementing a vanilla Transformer. Also in their work they try to find a solution for them and this is why we chose their model as backbone for our.

Lack of inductive biases

This problem is due to the small size of the (training) dataset. In fact the Transformer are data "hungry", especially compared to CNNs, and for this type of problem (TAS) usually the training set is relatively small. In fact it takes a lot of time to do all the data acquisition, for example the whole Breakfast dataset is ≈77 hours of video but it has only around 180 example for each activity. The lack of inductive biases makes difficult to learn a target function from a large hypothesis space.

Hard to form an effective representation

Instead this problem is related to the to the deficit of self-attention for the long input data (in this case videos). In fact a video can have up to 9741 frames, so it will be difficult for the self-attention mechanism to give attention to the whole input sequence in a meaningful way. At initialization, the self-attention layer give nearly uniform attention weights to all the elements in the sequence.

But due to the length of the videos, it is difficult for the self-attention layer to learn appropriate weights in meaningful locations. Moreover if the transformer block in the first stages are not able to find meaningful location where to put their attention it means that this task will be even more difficult for encoders block in the next stages.

Encoder-decoder architecture

Finally the last problem is intrinsic in the architecture of the vanilla Transformer. The main problem of an encoder-decoder architecture is in the preservation of the temporal information between frames and especially in the refinement process. Other architectures like CNNs and TCNs add additional information over the initial prediction to perform refinement, something that is not possible in the Transformer due to the decoder part, that is not designed for this purpose.

Solutions

To solve this problems they proposed simple but efficient solution that we will briefly describe here. First of all we do not need to deal with the third problem, since our problem is considering a weakly-supervised scenario instead of a fully supervised, as their case, so in our case there is not the decoder part, in fact it is not possible to do the refinement without the ground truth of the frames. To solve the first problem they relied on one property of the action segmentation task, in fact this problem has high locality of the features, this is because every action occupies continued timestamps. For this reason local inductive and temporal bias is very important to the action segmentation task. To enforce this bias they applied an additional temporal convolutions in each layer.

For the problem that with a Transformer architecture with serials of self-attention layers is hard to form an effective representation over long input sequence, they simply constrained each self-attention layer with a pre-defined 'window' of attention of increasing size. This way they obtained an hierarchical representation pattern, which forces the low-level self-attention layers to focus on local relations at first and then gradually enlarges their attention to capture longer dependencies. This process seems to solve the problem and, since the attention window gradually increase, the self-attention layer can cooperate better to achieve faster convergence speed and higher performance. This hierarchical representation has also another benefit, in fact it reduces the total space and time complexity to make our model faster and scalable. In Fig. 4.2 is reported an example of their self attention for a particular frames considering both the normal and hierarchical phase.

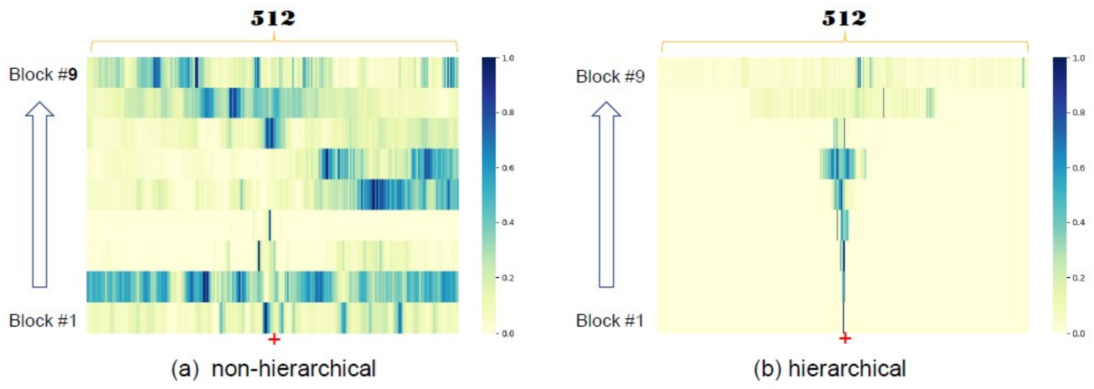

Figura 4.2: The visualization of the original attention weights for an anchor frame (+) in each encoder block.

4.1.2 Improved Self-Attention

Before using directly their model as a back-model for our, we decided to test their implementation to understand if there are some errors or possible improvement. Since here we just wanted to test their model, we decided to work in a fully-supervised scenario, like them. The first thing that we noticed is that the self-attention calculation is the bottle neck of their architecture. In fact is very time expensive and moreover the attention window for the frames in the border (starting and ending frames) is bad implemented, this because half of the attention window is wasted on padding frame. To solve these problem we implemented our version of the hierarchical self-attention, based on the following work [37]. A snippet of code with the most relevant part of our implementation of the sliding window self-attention is reported below:

```
    # Create windows around each token.
windows = torch.stack([x[:, :, i:i + self.w] for i in range(n)])
windows = windows.view(b * n, d, self.w)
    # Compute queries, keys, and values for each window.
q = self.q_linear(windows).view(b, n, self.n_heads, self.d_model // self.n_heads,
self.w).transpose(2, 4)
k = self.k_linear(windows).view(b, n, self.n_heads, self.d_model // self.n_heads,
self.w).transpose(2, 4)
v = self.v_linear(windows).view(b, n, self.n_heads, self.d_model // self.n_heads,
self.w).transpose(2, 4)
```

Here self.w is the dimension of the sliding window, that is double at each encoder block. In Tab. 4.1 are reported some comparison of the results obtained using ours and their implementation of the self-attention, considering different number of encoder block and considering the case with and without the decoder. Compare to the previous implementation ours is much more faster and the accuracy results are almost the same. From the Epoch time column we can notice that our implementation is almost 10 times faster compared to theirs, this is very important for this work. The only drawback is in the F1 score, where there is a big drop in performance. Nevertheless our implementation is way to fast compare to their and since time complexity is a problem, due to the size of the dataset, we decided to use our implementation.

# Layer	# Decoder	Type	Accuracy	F1	Epoch time (s)
8	0	Ours	**83.17%**	35.40	**6.69**
8	0	Their	81.79%	**46.28**	60.38
8	2	Ours	**82.94%**	**73.05**	**21.22**
8	2	Their	82.14%	68.67	174.62
9	0	Ours	**82.72 %**	40.26	**7.62**
9	0	Their	82.36%	**50.10**	62.27
9	2	Ours	81.69%	68.29	**22.81**
9	2	Their	**82.90%**	**70.51**	174.22
10	0	Ours	82.35%	33.33	**8.70**
10	0	Their	**82.99%**	**46.82**	68.85
10	2	Ours	81.91%	67.81	**25.16**
10	2	Their	**82.58%**	**68.29**	177.25

Tabella 4.1: Time comparison of our implementation of the self-attention and theirs.

Finally in the Fig. 4.3 we reported the self-attention weight, using our implementation, of different frames belonging to video of different activities. From the image we can see the attention windows that increase, double, at each encoder block. Also the color represent the attention weight to the neighbors frames, here the more blue the segment is (frames) the more attention the model is giving to that particular neighbor frame, vice-versa the more white it is the less attention is given to that frame.
Also is worth to notice the bottom right figure of the image. Here is represented the self-attention weight matrix respect to a frame in the end of the video. As we can see from this image most of the attention is put on the left part of the window, since the right part are just padding frames. This padding frames are added to fit the attention window also the 'border' frames, the ones that are in the start or at the end of a video.

Positional encoding

Another difference within the original ASFormer of [7] and our work is the use of the Positional Encoder. In fact in their work they did not use the positional encoder, typical of the vanilla Transformer architecture. The reason why they did so is because, and citing the original paper:
"The redundant absolute position encoding might be harmful to the temporal convolutions to learn the feature embedding". we tried to add the positional encoder to our implementation and we did not notice any drop in the performance and especially in the time required to train the network. So, since the positional encoder, is a key element of the vanilla Transformer we decided to used it also for our architecture.

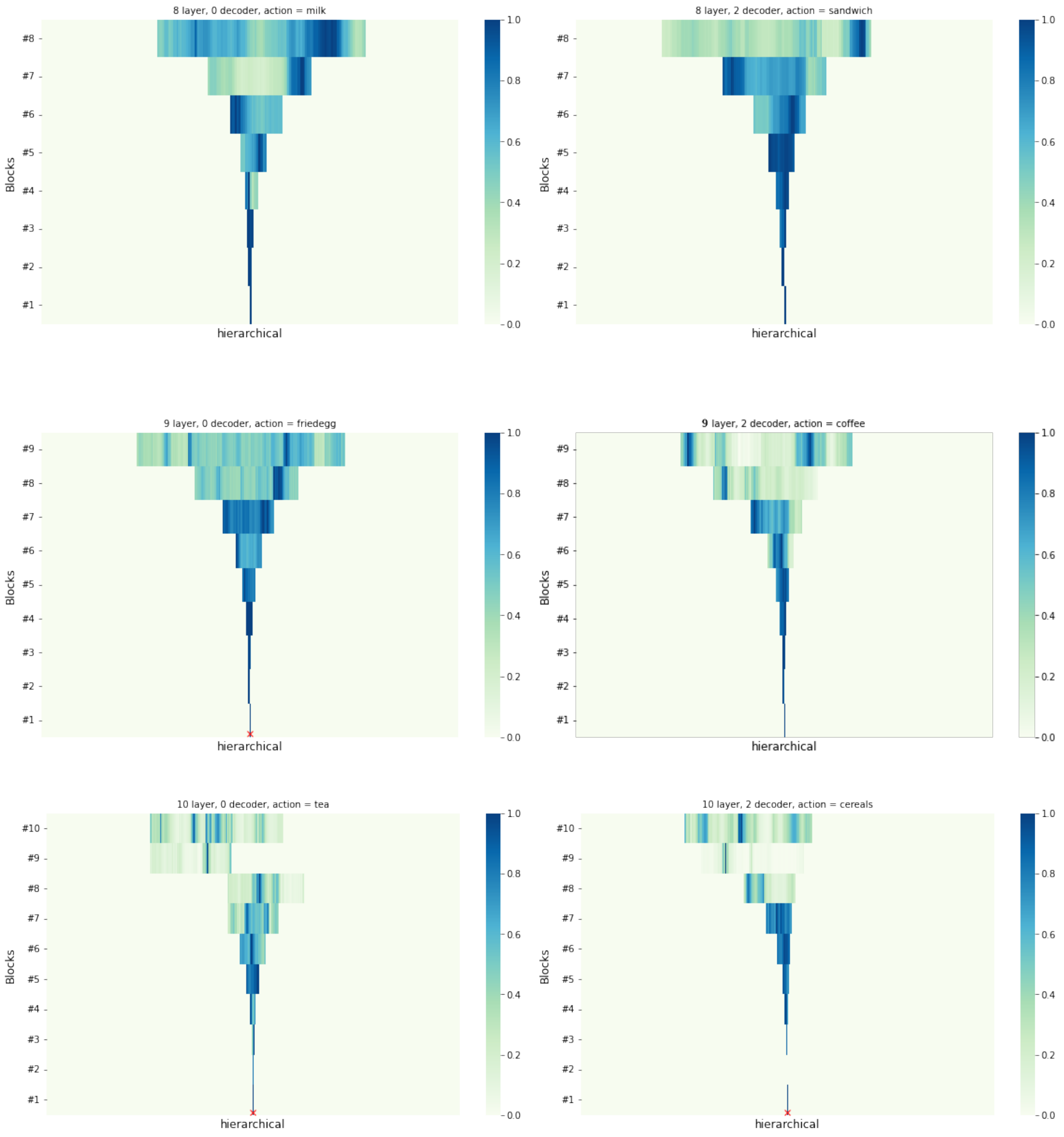

Figura 4.3: Self-attention weight using our implementation. For each row there is different number of encoder block. From the first row to the last are reported the results with 8, 9 and 10 encoder block. Instead the first column is considering 0 decoder while the second one 2 decoder blocks in series. Every self-attention is considering a frame from videos of different activity.

4.2 Siamese Architecture

Having implemented and improved the Transformer encoder block for our model, we can finally construct the main architecture to solve the Temporal Action Segmentation problem. We decided to build a Siamese architecture to distinguish between video of the same activity or different activities. Briefly a Siamese architecture is a type of neural network architecture used in tasks related to similarity or distance measurements between inputs.

In a Siamese architecture, two identical subnetworks are created, and they share the same parameters and weights. These subnetworks took two different inputs, often referred to as "anchor" and "comparison" inputs. The goal is to learn a similarity metric, meaning that the network aims to determine how similar or dissimilar the two inputs are.

For this particular problem the scope of the Siamese architecture is to decide if two video belongs to the same activity or not. The similarity metric is calculated through a constrastive loss. So the scope of the network is no longer to classify each frames of the video, as supposed for TAS problem, but is to make the features of the frames of the videos belonging to the same activity more closer (similar) and the features of the videos of dissimilar activities more distant (dissimilar). This will be explained better in the next section. The structure of the encoder block is reported in Fig.4.4, and for constructing the two subnetworks we used nine encoder blocks in series. This means that the maximum size of the 'window attention' that the architecture can give to a single frames is 512 (2^9). Therefore, in some video the self-attention of a single frame is calculated respect to all the other frames while in most of the cases the self-attention of a single-frames is related to just a portion of the video. This of course speed-up the training procedure.

In Fig. 4.5 is reported the final architecture of the Siamese model that was implemented to solve this problem.

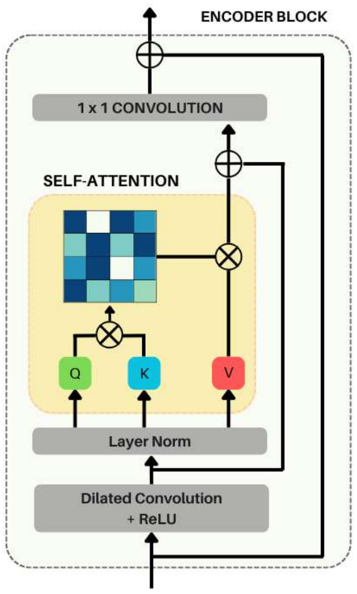

Figura 4.4: Encoder block inside the Siamese architecture.

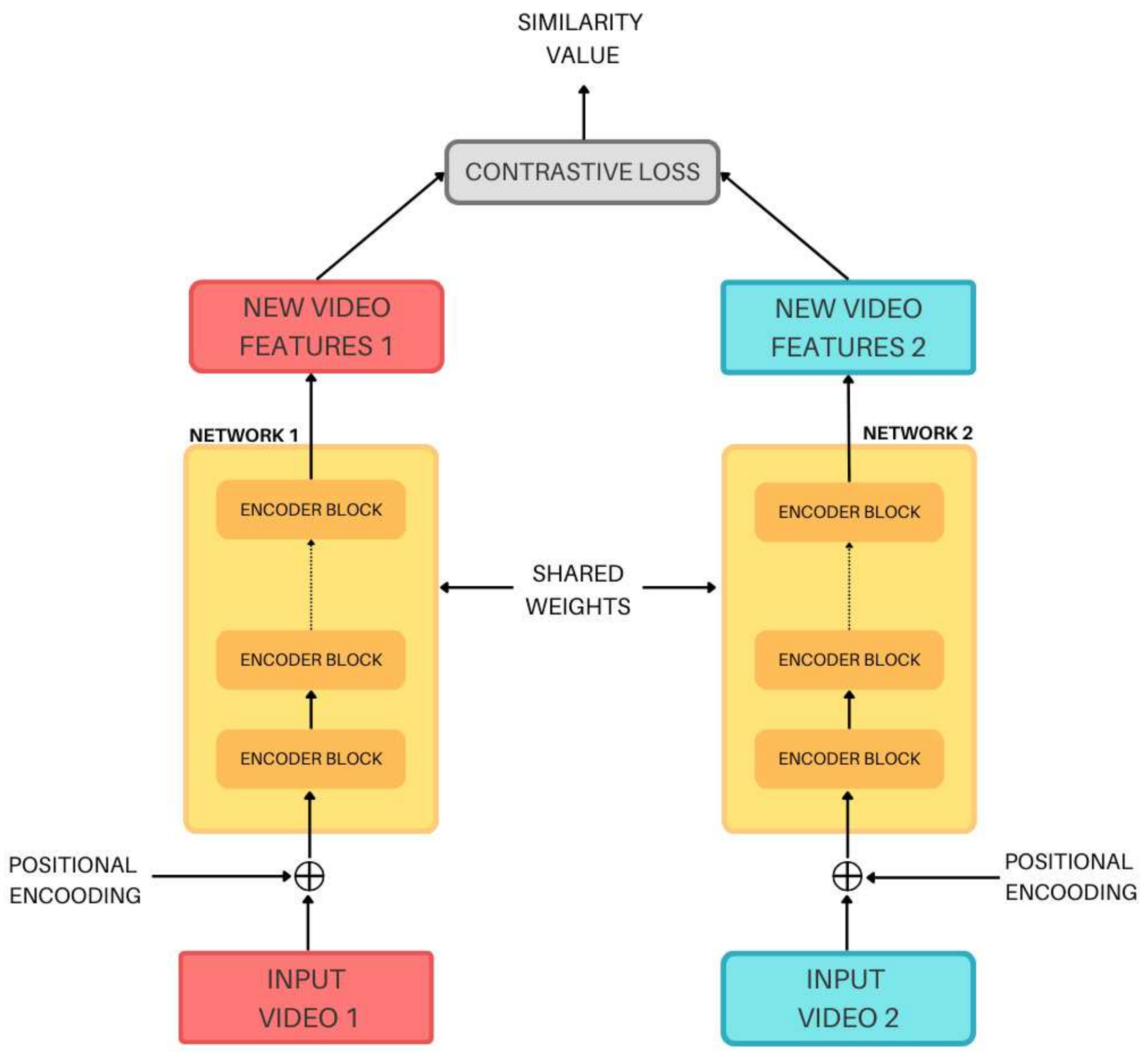

Figura 4.5: Siamese architecture.

4.2.1 Curriculum Learning

Curriculum learning[38] is a machine learning (ML) training strategy inspired by how humans learn. In the context of ML algorithms, curriculum learning involves gradually introducing training examples or concepts to the model in a structured order, making it easier for the model to learn progressively more complex patterns. Basically the idea is to train with the more easier classes or task initially, and when the model has started to learn those tasks, we can gradually insert more and more complexity into the training data. This strategy is used with a contrastive loss, explained in the next section, in order to train the simaese model.

First of all, if we want to apply he curriculum learning we need to define the stages, from the easiest to the difficult ones, to train progressively the architecture. Recall that the goal of this Siamese architecture is to discriminates between videos of same activity and videos that represent two different activity. To do so we added a flag value to all the couple of video given as input to the network, where '0' represent couple of video belonging to the same activity (positive cases) and '1' to different activities (negative cases). We need the positive and negative cases in order to apply the contrastive loss, but this will be explain better in the next section.

To create different stages we need to divide the couples of videos based on some common features of the video themselves. From the names of the videos it can be extracted the following information: person who take the video, kitchen where is taken the video and which camera was used. We relied only on those information to separate the couple of videos, of course watching every single video to extract more information is unfeasible due to the time required.

If two videos are in the same kitchen it means that the background is the same, so it will be easier for the architecture to distinguish between them. While if two activity are done by the same person it means that the order of action for completing it will be the same. Finally if two videos are taken with the same camera it means that the quality of the videos (and frames) are the same. Also is worth to mention that if a couple of video are taken in two different kitchen it means also that the person and the camera are different between the two videos.

In Tab.4.2 are reported the way we used these information to create different stages, from the easiest case to the most difficult one, both for the positive and negative case. From the table we can notice that the easiest combination of the positive case is the most difficult of the negative case and vice-versa. Finally we also considered an additional stage (Stage 5) that is simply the combination of all the previous stages.

Positive (Same Activity)		**Negative** (Different Activity)	
Stage	**Stage**	**Stage**	**# Reduced**
P1	same kitchen, same person, diff. camera	N1	different kitchen
P2	same kitchen, diff. person, same camera	N2	same kitchen, diff. person, diff. camera
P3	same kitchen, diff. person, different camera	N3	same kitchen, diff. person, same camera
P4	different kitchen	N4	same kitchen, same person, diff. camera
P5	Combination of the previous cases	N5	Combination of the previous cases

Tabella 4.2: Different stages for positive and negative pair of videos.

With these information we computed all the possible combination of couple of videos. In Tab.4.3 and Tab.4.4 we reported the number of the total combination for each stages both in the case of training phase and test phase. From these Tables we can notice two things. First that the number of possible combination is huge, especially in the negative case. In fact for the positive case we have a strong constriction that is: a pair of video must belong to the same activity. This will create some problems, in fact considering all the possible combination will take to much time during the training phase. Moreover, the number of total combination in positive and negative cases for the same stage differs a lot, creating an unbalanced dataset for the training and the testing phase related to a particular stage. To solve both of these problem we reduced, randomly, the number of possible combination for each stage in a way that the positive and negative case has more ore less the same number of combinations, this can be viewed in the Tables4.3 4.4.

Positive			**Negative**		
Stage	**# Original**	**# Reduced**	**Stage**	**# Original**	**# Reduced**
P1	1402	1402	N1	671392	2237
P2	1361	1361	N2	41876	2463
P3	4639	4639	N3	12155	6077
P4	74684	12399	N4	1325	12325
P5	82086	19801	N5	737748	23102

Tabella 4.3: Total number of combination on train phase.

Positive			**Negative**		
Stage	**# Original**	**# Reduced**	**Stage**	**# Original**	**# Reduced**
P1	150	150	N1	75327	251
P2	132	132	N2	3938	196
P3	423	423	N3	1284	642
P4	8400	1400	N4	1262	1262
P5	9105	2105	N5	81811	2351

Tabella 4.4: Total number of combination on test phase.

Finally in the Tab. 4.5 are reported the times required to train each stage of the architecture. These times are the best case scenario, when no other program is running in the server, otherwise it can require up to double the time for each stages to running the training phase. This is to highlight the importance of time management in this work, in fact to train the whole network in the wrong scenario it can require up to a full week. This is why it is important to reduce the number of possible combination for each stages. Finally in the Tab. 4.5 are reported also the accuracy result for the validation set. To do so we just put a threshold on the similarity result, produced by the contrastive loss, between the two video. The results are quite good, in fact in the last stage is able to reach 87.96% of accuracy. This mean that the Siamese architecture is able to produce a new representation of the videos (and so also of the frames) to distinguishes them between video of the same activity or not.

Recall that here we are in a supervised setting, since the network has the information if each pair of videos belong to the same activity or not.

	Time (sec)	**Time (min)**	**# Epochs**	**Total time (hours)**	**Accuracy**
Stage 1	≈ 139	≈ 2.31	80	< 4	93.01%
Stage 2	≈ 140	≈ 2.3	80	< 4	92.37%
Stage 3	≈ 417	≈ 6.95	60	< 7	84.22%
Stage 4	≈ 956	≈ 15.93	50	< 13	84.52%
Stage 5	≈ 1641	≈ 27.25	40	< 25	87.96%

Tabella 4.5: Times required to train the architecture for each stages.

4.2.2 Cropping and padding

As described before in the Dataset section, each video as a different length that can different a lot from each other.In fact for this dataset the shortest video has only 130 frames, while the longest 9741. To solve this problem and to have all the video at the same length we applied a cropping and padding strategy. We basically cropped videos, removing random frames within it, that are longer than the chosen length and padded the videos, adding zeros vector to the start and end, that are shorter than the chosen length. Cropping, and so reducing the video size, will also speed up the training process. Instead padding with zeros is not a problem since, inside the encoder blocks of the architecture, is present a mask that gives no attention (zero values) to all the zeros vector (padding frames), wasting no time on them. Finally we found out that the best padding/cropping value is 4096. We tried also to increase the possible length of the video in order to preserve more information but, already with a value of 8192, we had some problems since it overflow the GPU memory.

4.2.3 Contrastive Loss

Contrastive loss was first introduced in 2005 by Y. Le Cunn et al. [39] in this paper and its original application was in dimensionality reduction. Briefly, the goal of dimensionality reduction is to reduce the features space while preserving as much information as possible. Moreover, it should preserve neighborhood relationships between data points and been able to generalize to new unseen data.
Thanks to the contrastive loss the Siamese architecture is trained to map the

original features of the frames in such a way that the new embeddings of the frames of the videos belonging to the same activity are close to each others, while the embedding of frames of videos belonging to different activity are far from each other. Finally to verify if two videos are belonging to the same activity or not, you give them as input of the architecture and calculate the similarity between the obtained embeddings. If the similarity is small, the videos are likely to belong to the same activity, viceversa, if the difference is big enough, the two videos are likely to belong to two different activity.

More in detail the original formula of the contrastive loss[39] is the following one:

$$L(\overrightarrow{X_1}, \overrightarrow{X_2}) = (1 - Y)\tfrac{1}{2}(D_W)^2 + (Y)\tfrac{1}{2}\left\{max(0, m - D_W)\right\}^2$$

where m, is a hyperparameter, defining the lower bound distance between dissimilar samples. $\overrightarrow{X_1}$ and $\overrightarrow{X_2}$ are the features matrix of the two videos. Y is binary value that specify if the two videos belong to the same activity ($Y =$ 0) or to two different activities (Y =1). Instead D_W is the similarity metrics, in this architecture is used the Euclidean distance

The formula is highly similar of the Cross-entropy. The main difference is that Cross-entropy loss is a classification loss which operates on class probabilities produced by the network independently for each sample, instead the contrastive loss is a metric learning loss, which operates on the data points produced by the architecture and their similarity relative to each other.

Instead, to be more specific, "margin" is a minimal distance that dissimilar points need to keep. So it penalizes dissimilar samples for beings closer than the given margin. We used a margin value of 1.0. Y. LeCun et al.[40] prove that increase to much the value of margin is counterproductive.

An improvement of contrastive loss is triplet loss that outperforms the former by using triplets of samples instead of pairs[41].

Specifically, it takes as input an anchor sample I^α, a positive sample I^+ and a negative sample I^-. During training, the loss enforces the distance between the anchor sample and the positive sample to be less than the distance between the anchor sample and the negative sample:

$$L = \textstyle\sum_I^N \left\|X_I^\alpha - X_I^+\right\|_2^2 - \left\|X_I^\alpha - X_I^-\right\|_2^2 + m$$

When a model is trained with the triplet loss, it require fewer samples for convergence since it simultaneously update the network using both similar and dissimilar samples. That's why triplet loss is more effective than contrastive loss.

For this work we used the contrastive loss due to the way the architecture is constructed and because the model achieves good results anyway.

4.3 Other strategies

To improve the result of the Siamese architecture we tried also other strategies, which did not prove to be effective.

First of all we tried to calculate the self-similarity matrix of the input features, to understand from the raw data if there are some natural division between frames, basically to see if is possible to divide the frames into action directly from raw data. A similarity matrix is symmetric, with the diagonal elements representing the similarity of each data point with itself (usually set to the maximum similarity value). The off-diagonal elements capture the pairwise similarities between different data points.

We calculated the self-similarity matrix of the input features considering two different similarity measures, the euclidean distance and the cosine similarity. In Fig.4.6 are reported the self-similarity obtained for the features of a single video. In the image the vertical red dot lines are the boundary between the actions within the video. Already from these similarity matrix we can see that are not able to capture any meaningful information about that action boundary. In the cosine case we can notice high similarity in the top-left region, corresponding to the first action. Although, it is not sufficient since for the rest of the matrix e can not find any other region of high-similarity. Nevertheless we tried to multiply the similarity matrix to the input features to test empirically if it will improve the results, but it did not.

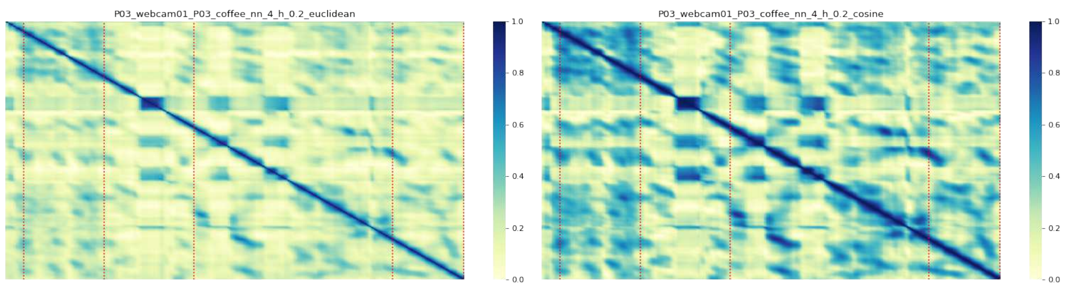

Figura 4.6: Similarity matrix considering euclidean distance and cosine similarity.

Another approach, to speed up the training of the network and to try to improve the results, is to apply directly to the input raw videos a down sampling. Briefly we removed randomly some frames of the video in order to get all the video to a specific length. The idea here is to reduce the redundancy of the frames next to each other. In fact, the videos are at taken at 15 fps and some actions can take minutes to accomplish them. This approach is not based only on an intuition but it follow also the work of Lei at al.[42], but in their case they applied it to a Temporal Convolutional Networks instead of a Transformer. We tried to implement also this downsampling strategy directly on the video and it did not work. There was a huge improvement in the time complexity but the MoF drops a lot.

Capitolo 5

Experimental setting

In this section, we introduce and analyze in details the dataset used for training and testing our architecture. We then explained how to use the Hungarian matching algorithm in order to calculate the evaluation metrics used to compare our model with the *state-of-the-art*. Finally we describe the algorithm used to classify each frame to their respective action cluster.

5.1 Dataset

There are a lot of available dataset concerning Temporal Action Segmentation problems. In this section there is a complete explanation of the datasets used in this work and also a brief explanation of the most important datasets for this kind of problem.

5.1.1 Breakfast dataset

The most important dataset for this work, and in which most of the experiments were done, is the Breakfast dataset[43]. The choice of this dataset is not random. First of all, most of the *state-of-the-art* models used this dataset to compare their results with the others. Secondly this dataset is well annotated and the quality of the videos is good enough. Moreover, for this dataset, the task of TAS is not trivial, especially in the *weakly-supervised* and *unsupervised* scenario, instead for the *fully-supervised* case the *state-of-the-art* models are able to achieve a Mean over Frame $\approx$76.0%. In Fig.5.1 are reported random frames from videos of this dataset, so we can have an idea of the type of images (frames) our model is gonna working with. To reduce the overall amount of data, all videos were down-sampled to a resolution of 320$\times$240 pixels with a frame rate of 15 fps.

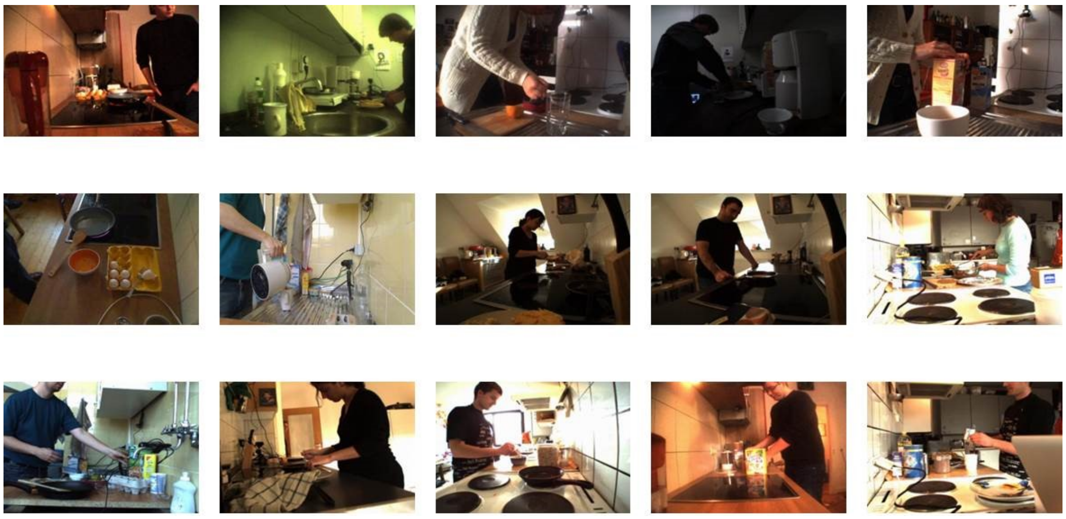
Figura 5.1: Frames from the videos of Breakfast dataset.

5.1.2 File names

A very important thing to understand for this work is the way every video is named. For this purpose we can check a random video name like the following one:

P25_cam02_P25_juice

First of all 'P25' represent the person that took the video. In total there are 52 persons and in this specific case the video is taken by person 25. After, 'cam02' is saying to us with which camera the video is taken, different cameras implies different quality of the videos and so also of the frames. And finally 'juice' told us which activity is performed in the video. Is not intuitive but the information of the person who is taken the video is very important for this problem, for two main reasons. The first is that each person can complete the activity on their own way, so they can take more or less time to complete it and also the order of action can differ. Moreover knowing which person is doing the activity give us another important information: the background. In fact different person did the activities (videos) in different kitchens and so, the background of the video can change a lot between them. To only way to find out to which kitchen each person corresponds, is to check manually at least one video for each person. In Tab. below5.1 are reported these correspondences.

5.1.3 Exploratory data analysis

The Breakfast Actions Dataset comprises of 10 actions related to breakfast preparation, performed by 52 different individuals in 18 different kitchens. The dataset is one of the largest fully annotated datasets available. The actions are recorded in real life scenario as opposed to a single controlled lab environment. It consists of over 77 hours of video recordings. Each activity is composed of different actions, in total there are 48 of them and they can belong among different activity. Moreover one of these action is the *Background*, this special action is when the frames in the videos cannot be assigned of any of the other

Kitchen ID	**People ID**
1	P03
2	P04
3	P05 - P06
4	P07
5	P08 - P09 - P10
6	P11
7	P12 - P13
8	P14 - P15
9	P16 - P17 - P18
10	P19 - P20
11	P21 - P22 - P23 - P24
12	P25 - P26 - P27
13	P28 - P29 - P30 - P31 - P32 - P33
14	P34 - P35
15	P36 - P37 - P38
16	P39 - P40 - P41 - P42 - P43
17	P44 - P45 - P46 - P47 - P48
18	P49 - P50 - P51 - P52 - P53 - P54

Tabella 5.1

actions. In Tab. 5.2 and Tab. 5.3 are reported the relations between the activity and their actions.

Activity	**# Actions**	**# Videos**	**avg length**
Cereals	5	214	704.08
Coffee	7	200	586.57
Friedegg	9	198	3119.84
Juice	8	187	1490.50
Milk	5	224	948.59
Pancake	14	173	5968.94
Salad	8	185	3429.51
Sandwich	9	197	1535.45
Scrambledegg	12	188	3117.33
Tea	7	223	716.20

Tabella 5.2: Number of action and videos for each activity.

As we can see from Tab. 5.2, every activity is composed by a different number of action and also the average length of the videos can change a lot. In fact *Coffee* has an average length of 586.57 while *Pancake* 5968.94. This suggest that different activity can be distinguished by each others by intrinsic properties of the activity itself, in fact some activities are more complicated than other since they require more time and they need more actions complete it. Also the shortest video is 'P51_webcam01_P51_coffee' with 130 frames and the longest is 'P42_stereo_01_P42_pancake' with 9741 frames. This huge different in the length could be a huge problem, since we need to define a model that is able to work with very different input length.

Activity	**Actions**
Cereals	take bowl - pour cereals - pour milk - stir cereals
Coffee	take cup - pour coffee - pour milk - pour sugar - spoon sugar - stir coffee
Friedegg	pour oil - butter pan - take egg - crack egg - fry egg - take plate add salt and pepper - put egg onto plate
Juice	take squeezer - take glass - take plate - take knife - cut orange - squeeze orange - pour juice
Milk	take cup - spoon powder - pour milk - stir milk
Pancake	take bowl - crack egg - spoon flour - pour flour - pour milk - stir dough - pour oil - butter pan - pour dough into pan - fry pancake - take plate - put pancake onto plate
Salad	take plate - take knife - peel fruit - cut fruit - take bowl - put fruit to bowl - stir fruit
Sandwich	take plate - take knife - cut bun - take butter - smear butter - take topping - add topping - put bun together
Scrambledegg	pour oil - butter pan - take bowl - crack egg - stir egg - pour egg into pan - stir fry egg - add salt and pepper - take plate - put egg onto plate
Tea	take cup - add teabag - pour water - spoon sugar - pour sugar - stir tea

Tabella 5.3: List of activity and corresponding actions inside the dataset.

Other useful metrics to analyze this dataset are the *repetition score*, the *order variation score* and the *imbalance ratio (IR)*. These metrics helps understand better the relationship between actions for each video/activity. The results are reported in Tab.5.4

Repetition score

The repetition score counts how many times unique actions are repeated inside a video.

$$r = 1 - \frac{u}{g}$$

where u is the number of unique action in a video and g is the number of action in that video. The value is between 0 and 1, where 0 means that no action are repeated within a video.

Order variation score

The order variation score gives us information on how much the order of actions change between video of the same activity. Value close to 1 means that the actions follow a strict order, with less variation between videos. On the other hand, a value near 0 indicates a high amount of ordering variations, making modeling the temporal relations between actions more difficult. The order variation score is defined as the average edit distance, between every pair of sequences. Then the result is normalized with respect to the maximum sequence length of the two.

$$v = 1 - e(S1, S2)/max(|S1|, |S2|)$$

Imbalance Ratio

The imbalance ratio [44] is a way to check the skewness of the input data, in this case of the actions distribution. This parameter is important to check because some metrics, like the Mean over Frame, are meaningful if the data are too unbalanced. To calculate it we need to compute the ratio between the number of frames in the head (action more frequents and with more frames) and the tail classes (actions with less frames).

Dataset	**Rep. score**	**Order var. score**	**Imb. Ratio**
Breakfast	0.11	0.15	639

Tabella 5.4

5.1.4 Pre-Computed Features

The owner of the dataset[43] provide also the pre-computed features of the input videos. This is very useful for multiple reasons. Firstly, because it save a lot of work and time since the the input features are already extracted from the videos. Secondly because it provides a common baseline, so is possible to compare our model with the *state-of-the-art*, since the results are obtained starting with the same feature and so they cannot be attribute on the way the features are extracted. The owners of the dataset provided two type of pre-computed features: Fisher Vector and Inflated 3D.

Fisher Vector

This methodology is based on Improve Dense Trajectory (IDT). The original dense trajectories features[45] are spatiotemporal features computed using optical flow while the improvement is a correction on camera motions. To been able to use this type of features for this problem the raw trajectories are encoded using Fisher Vectors (FV) [46]. In order to have further information on how the FV are extracted please consult the original paper by H. Kuehne et al.[47]. The main ideas of the work can be seen in Fig.5.2. Briefly the Improve Dense Trajectory features are computed and the corresponding descriptor is reduced to 64 dimensions. After a total of 200,000 features are randomly sampled and fitted to Gaussian Mixture Model, with different number of clustering (K). Then an FV representation is computed for each frame of the video. The corresponding representation is further reduced from 2048–32,768 (depending on the value of K) down to 64 dimensions.

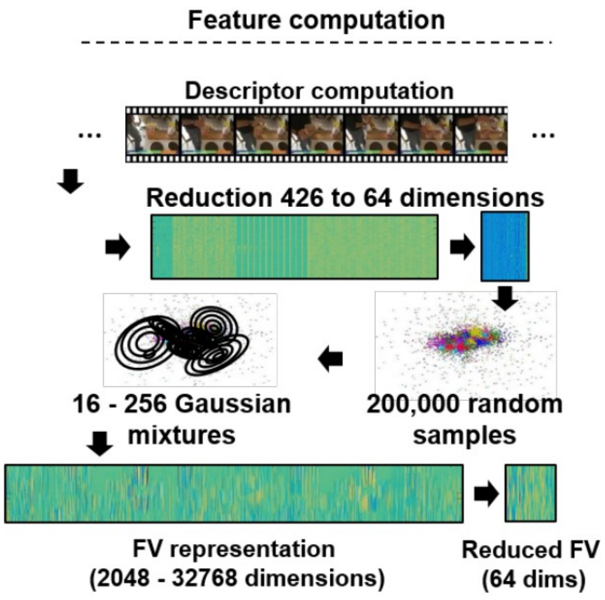
Feature computation
Descriptor computation
Reduction 426 to 64 dimensions
16 - 256 Gaussian mixtures
200,000 random samples
FV representation (2048 - 32768 dimensions)
Reduced FV (64 dims)

Figura 5.2: Original pipeline by H. Kuehne et al. to extract Fisher Vectors.

Inflated 3D ConvNet (I3D)

The features from videos are extracted using Inflated 3D ConvNet (I3D)[48] a *state-of-the-art* model to extract features from videos based on Inception-V1[49] architecture. The main idea is to inflates all NxN spatial kernels to NxNxN by replicating the original kernels N times and rescaling them with a temporal factor of N=1. The main problem with this features and the reason they were not chosen for this work is their dimension. In fact the folder containing the pre-computed features is ≈ 25.7 GB. This means that they are much more heavy to compute respect the Fisher Vector representation (≈ 2 GB) but the main problem was that there was not enough space in the server to stored them. For these reasons we worked with the Fisher Vector representation.

Other datasets

As we mention at the beginning of this section there are a lot of available datasets for this type problem. Here we will do a quick review of most of them, for further information check the reference.

- **GTEA**[50]: The Georgia Tech Egocentric Activities (GTEA) dataset contains seven types of daily activities such as making sandwich, tea, or coffee. Each activity is performed by four different people, for a total of 28 videos. For each video, there are about 20 fine-grained action instances such as take bread, pour ketchup, in approximately one minute.

- **50-SALADS**[51]: This dataset consists in 50 recorded videos, this collection features 25 participants engaged in the creation of two distinct mixed salads. The camera captures these videos from a top-down perspective, showcasing the participants' activities on the work surface. Each participant is furnished with a set of recipe steps, drawn randomly from a statistical recipe model.

- **CrossTask**[52]: CrossTask dataset contains instructional videos, collected for 83 different tasks. For each task is provided an ordered list of steps with manual descriptions. The dataset is divided in two parts: 18 primary and 65 related tasks. Videos for the primary tasks are collected manually and provided with annotations. Videos for the related tasks are collected automatically and don't have annotations.

- **COIN**[53]: The COIN dataset is alarge-scale dataset for COmprehensive INstructional video analysis. It consists of 11,827 videos related to 180 different tasks in 12 domains (e.g., vehicles, gadgets, etc.) related to our daily life. The videos are all collected from YouTube.

- **HA4M**[54]: the Human Action Multi-Modal Monitoring in Manufacturing (HA4M) dataset is a collection of multi-modal data relative to actions performed by different subjects. In particular, 41 subjects executed several trials of the assembly task, which consists of 12 actions. Data were collected in a laboratory scenario.

- **Epic-Kitchens**[55]: Epic-Kitchens dataset comprises a set of 432 egocentric videos recorded by 32 participants in their kitchens at 60fps with a head mounted camera. There is no guiding script for the participants who freely perform activities in kitchens related to cooking, food preparation or washing up among others. Each video is split into short action segments with specific start and end times and a verb and noun annotation describing the action.

- **Ikea-ASM**[56]: The IKEA ASM dataset is a multi-modal and multi-view video dataset of assembly tasks. It contains 371 samples of furniture assemblies and their ground-truth annotations. Each sample includes 3 RGB views, one depth stream, atomic actions, human poses, object segments, object tracking, and extrinsic camera calibration.

- **Meccano**[57]: MECCANO Multimodal comprises multimodal egocentric data acquired in an industrial-like domain in which subjects built a toy model of a motorbike. The multimodality is characterized by the gaze signal, depth maps and RGB videos acquired simultaneously.

- **YouCook2**[58]: YouCook2 is the largest task-oriented, instructional video dataset in the vision community. It contains 2000 long untrimmed videos from 89 cooking recipes; on average, each distinct recipe has 22 videos. The procedure steps for each video are annotated with temporal boundaries and described by sentences. The videos were downloaded from YouTube and are all in the third-person viewpoint.

Dataset	**Year**	**Duration**	**# Videos**	**# Activity**	**# Action**	**Domain**	**View**
Breakfast	2014	77h	1712	10	48	Cooking	3rd Person
GTEA	2011	0.4h	28	7	71	Cooking	Egocentric
50Salads	2013	5.5h	50	10	17	Cooking	Top-view
Epic-Kitchens	2020	200h	700	-	4053	Daily activity	Egocentric
Meccano	2021	0.3h	20	1	61	Assembly	Egocentric
HA4M	2022	6h	217	1	12	Manufacture	3rd Person
YouCookII	2018	176h	2k	89	-	Cooking	Mixed
CrossTask	2019	376h	4.7k	83	107	Mixed	Mixed
COIN	2019	476h	11.8k	180	778	Mixed	Mixed

Tabella 5.5: Summary and comparisons of procedural activity datasets.

5.2 Hungarian matching

The Hungarian Matching algorithm, also known as the Hungarian method or Kuhn-Munkres algorithm[59], is a combinatorial optimization algorithm used to solve the assignment problem in bipartite graphs. The assignment problem involves finding the optimal way to assign a set of agents to a set of tasks, with each agent being capable of performing certain tasks and having associated costs or values for performing those tasks. The goal is to find an assignment that minimizes the total cost or maximizes the total value.

Formally, let's consider a bipartite graph where there are n agents and m tasks, represented by a cost or value matrix C of dimensions n x m. The element C $[i]$ $[j]$ in the matrix represents the cost or value associated with assigning agent i to task j. The Hungarian algorithm seeks to find an assignment matrix X of the same dimensions, where $X\,[i]\,[j]$ is 1 if agent i is assigned to task j and 0 otherwise, such that the sum of the values of $C\,[i]\,[j]$ where $X\,[i]\,[j] = 1$ is minimized or maximized.
The algorithm follows these main steps:

1. Row Reduction
2. Column Reduction
3. Marking Zeros
4. Augmenting Paths
5. Augmenting Path Update
6. Line Removal
7. Iteration
8. Solution Construction

The Hungarian algorithm guarantees to find the optimal assignment in polynomial time complexity, making it a powerful tool for solving practical assignment problems. Its significance extends to various applications such as job scheduling, resource allocation, and matching problems in various domains. In

unsupervised TAS, Hungarian matching links the frames X of N clusters to the action label corpus Y of M classes.
But why is needed the use of Hungarian Matching? This work is in an unsupervised scenario (or weakly-supervised), this means that there are not a correlation between the estimated segments and ground truth actions. So, in order to compare our results with the *state-of-the-art* and to obtain the metrics like Mean of Frames (MoF) and Intersection over Unit (IoU), we needed to use the Hungarian matching.

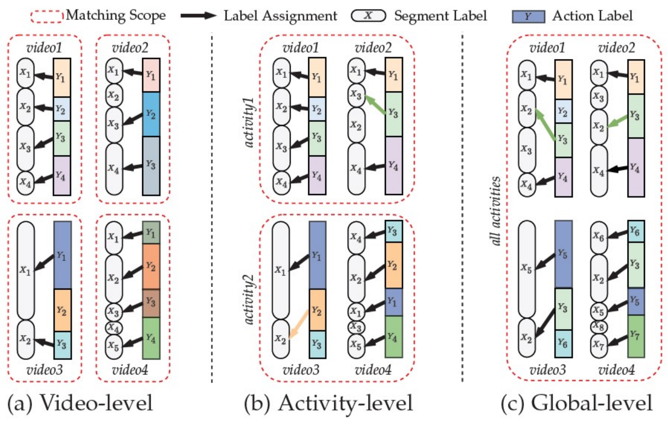

Figura 5.3: Hungarian matching applied at different levels.

As we can see from Figure 5.3 there are three possible levels for the application of Hungarian matching for TAS problem, and they are:

- **Video level**: This is the most trivial level. Here the scope is to match the labelled frames withing the ground truth frames of the same video. So the algorithm needs only to find relationship between the frames of the same video. This matching produces the best performance of the thre, due the fact that it is done per video.

- **Activity level**: In this case the Hungarian algorithm have to find the correspondence between frames and their ground truth considering all the videos of the same activity. This level is more challenging than the one before because here the proposed models need to find relationship also between different videos, related to the same activity. The work done for this thesis is considering this level for the Hungarian Matching algorithm.

- **Global level**: This one is the most difficult case. Here the algorithm need to find not only relationship between frames of different videos but also between videos concerning different activity, different activity means also different set of actions. There are not relevant works for unsupervised approaches for this level. The only one is the one of Kukleva et al.[36], but they assumed that each activity has a different set of actions that is not always true.

5.3 Evaluation metrics

The evaluation metrics for Temporal Action Segmentation are Mean of Frames (MoF), mean of Mean of Frames (mMoF), Edit Score, and F1-scores. These metrics are divided in two main categories: frame-based measure and segment-based measure. The first ones consider how any frames are predicted correctly while the second ones are evaluation metrics that focus on the segment errors. For the case of unsupervised and weakly-supervised learning, that are the ones of this work, only the MoF and F1 score are reported in most of the *state-of-the-art* models. Of course in these cases before computing the metrics we need to applied the Hungarian matching, as it was described in the previous section.

5.3.1 Frame-based metrics

The frame based metrics are Mean over Frames (MoF), that is basically the accuracy, and the mean Mean over Frames. The formula of MoF is the following one:

$$MoF = \frac{\#\,correct\ \ frames}{\#\,all\ \ frames}$$

The main problem with this metric is that it does not work well when the dataset is imbalanced, so when there are activity that has much more frames (because they are longer) then other ones, like in most of the case. To mitigate this problem was introduced the mean of Mean over Frame:

$$mMoF = \sum_a MoF(a)/|a|$$

where MoF(a) is the frame accuracy per class a and $|A|$ is the size of it.
The main drawbacks of these metrics is that they don't give any information on the quality of the segmentation of the video. This means that the MoF score might appear high even in cases where the segmentation results are actually disjointed. This phenomenon, where a continuous action is divided into multiple disconnected sub-segments, is known as over-segmentation. To check if the problem of over-segmentation is present in the results we can rely on segment based metrics.

5.3.2 Segment-based metrics

F1-score [17] and Edit Score [60] are the two segment-based methods used for this kind of problem. The F1-score evaluates the Intersection over Union (IoU) between each segment and its corresponding ground truth using a threshold of $\tau/100$. A segment is categorized as a true positive if its score surpasses the threshold in relation to the ground truth. In cases where multiple correct segments exist within the range of a single ground truth action, only one of them is counted as a true positive, while the remaining segments are designated as false positives.
The formula for F1-score is:

$$F1\_score = 2 * \frac{precision\ *\ recall}{precision\ +\ recall}$$

Usually the values for τ are: 10, 25, 50.

The Edit Score instead quantifies the similarity of two sequences. It is based on the Levenshtein distance and considers the minimum number of insertions, deletions, and replacement operations required to convert one segment sequence into another. The formula of the Edit score is:

$$Edit = \frac{1-e(X,Y)}{max(|X|,|Y|)} \cdot 100$$

This metric measures how well a model predicts the action segment ordering without requiring exact frame-wise correspondence to the ground truth.

Combining the frame-based and the segment-based methods we were able to have a clearer and more complete understanding on how well the model is performing.

5.4 Clustering

To cluster all the frames to their action we tried with some basics clustering algorithms. This because the scope of this work is to define an architecture powerful enough to learnt a new features representation for each actions so they will be easy to cluster them correctly. The methods that we tried are the following: *K-Means*, *Spectral clustering*, *Hierarchical clustering*, *DBSCAN*[61].

K-Means

This is the most common and intuitive algorithm to cluster data in an unsupervised way. The main advantages of this algorithm are that is easy to implement, and is computationally efficient. The problem is that it has many drawbacks. For example it assumes that the data points are distributed in a spherical shape, not always the case, and also is sensitive to the presence of outliers and noise in the data. But the main drawback, that is also in this work, is that requires the user to specify the number of clusters in advance.

Spectral clustering

Spectral clustering is an algorithm designed for cluster identification, leveraging the eigenvectors of a similarity matrix. This similarity matrix is crafted using a kernel function, quantifying the resemblance between data point pairs. Spectral clustering excels in scenarios where clusters exhibit non-linear structures, offering superior performance in handling noisy data compared to the k-means algorithm. Also it does not require the user to specify the number of clusters in advance.

Hierarchical clustering

This clustering algorithm creates a hierarchy of clusters, with each cluster being divided into smaller sub-clusters until all objects in the dataset are assigned to a cluster. The algorithm would start by treating each input data as a singular cluster, and then it would iteratively merge the closest pairs of clusters until all the data are grouped into a single hierarchy of clusters. The benefit compared to the k-means are that it can also data sets with varying densities and cluster sizes and that it does not require the number of cluster beforehand. Also this algorithm could be very useful for visualizing the structure of the data and identifying relationships between clusters.

DBSCAN

DBSCAN is a clustering algorithm that groups data points into clusters based on the density of the points. The algorithm works by identifying points that are in high-density regions of the data and expanding those clusters to include all points that are nearby. Moreover this algorithm is able to detect outliers and noisy point, that do not belong to high-density regions or that are not close to each other. Those point for this work will be automatically classified as 'background' action. There are many advantages over the k-means. First of all DBSCAN can handle data sets with varying densities and cluster sizes also it an identify clusters with arbitrary shapes, as it does not impose any constraints on the shape of the clusters. Finally, as the Hierarchical clustering algorithm, it does not require the number of clusters in advance.

5.4.1 Problems and application

All of these clustering algorithm are already implemented in 'sklearn.cluster' library, and in fact is the one that we used for this work. As stated previously, DBSCAN, Spectral and Hierarchical algorithms, theoretically, are much better than k-means, especially because they do not require the number of cluster in advance. Instead, practically, the only algorithm that is useful to use for this problem is k-means, this is because it is very computationally efficient and can work nicely also with huge amount of data. The problem here is that, since we are working at activity level, we need to perform a concatenation of all videos of the same activity before clustering each frames. This lead to a huge dimension of the input data (frames), that need to be clustered. In Tab. 5.6 are reported the total number of frames of all video concatenated for each activity and also are reported the number of cluster (action) for each activity.

Activity	**# frames**	**# cluster**
Cereals	129551	5
Coffee	97958	7
Friedegg	539733	9
Juice	241462	8
Milk	177387	5
Pancake	937125	14
Salad	558928	8
Sandwich	259495	9
Scrambledegg	517478	12
Tea	131782	7

Tabella 5.6: Number of total frames and clusters (actions) for each activity.

As we can see from the table, the number of concatenated frames is pretty high. For example, using DBSCAN all the frames are classified as -1, this means that are all considering as outliers or noisy data. So the algorithm is not able to find different area of density points and so it classify all the frames as *background* action. The Spectral clustering instead gave us memory issue, in fact when we tried to run it for the *Cereals* activity it gives the following error: " MemoryError: Unable to allocate 125. GiB for an array with shape (129551, 129551) and data type float64 " . Finally, also the Hierarchical clustering failed with this huge number of data. In fact, the algorithm does not converge, or at least in a feasible time, since it consider all the points as a cluster and then it started combining the in an hierarchical way, based on their distance. So, in the end, the only clustering algorithm that was possible to use for this problem was K-means.

Capitolo 6

Results

In this section we report quantitative results in terms of Mean of Frames and F1-score, and qualitative results through visualizations of the features space via t-SNE considering both video and activity level. Finally we reported the segmentation results of our model to understand better the strengths and weaknesses of our approach.

6.1 Quantitative results

Finally, the last thing to do is to test how well the model works for Temporal Action Segmentation. To do so we used the pretrained Siamese network, trained on the last stage of curriculum learning. We considered just one of the two streams, so its no longer a Siamese network, and pass all the video through it. Doing so we obtained a new and better representation of the videos frames. Finally all the video belonging to the same activity were concatenated. The last part consists of applying the K-means algorithm to classify the frames into their respective action clusters.

6.1.1 Ablation study and hyper-parameter tuning

To first thing we did to improve the results was tuning the hyperparameters of the Siamese architecture. The parameter that we tried to change are: optimizer, dropout, padding, number of layer, weight decay and learning rate. In Tab.6.1 are reported the possible choices for these parameter and in bold are the one that are the best within them. We considered the best parameter as the ones that minimize the validation loss. To check the performance of different combination of the parameters, we considered only the first stage of the curriculum learning and for 20 epochs. This of curse is due to time reason, as is not feasible to wait almost a week to have the results of only one combination.

Parameter	**Values**
num layers	**8**, 9, 10
padding	2097, **4096**
optimizer	'**adam**', 'SGD'
learning rate	0.01, **0.001**, 0.0001
dropout	0.1, **0.3**, 0.5
weight decay	**1e-8**, 1e-5

Tabella 6.1: Possible combination of hyperparameters.

To calculate all these possible combination we used a program called Weights & Biases [62] (W&B) . The reasons why we decided to use this tools are that is a popular platform for experiment tracking, hyperparameters optimization, and visualization of data in machine learning (ML) and deep learning.

In Fig.6.1 are reported all the runs, with different combination of hyperparameters, respect to the validation loss. In total we ran 139 combination of hyperparameters, chosen randomly between the ones in Tab.6.1; in the image is highlighted only the run with the lowest validation loss.

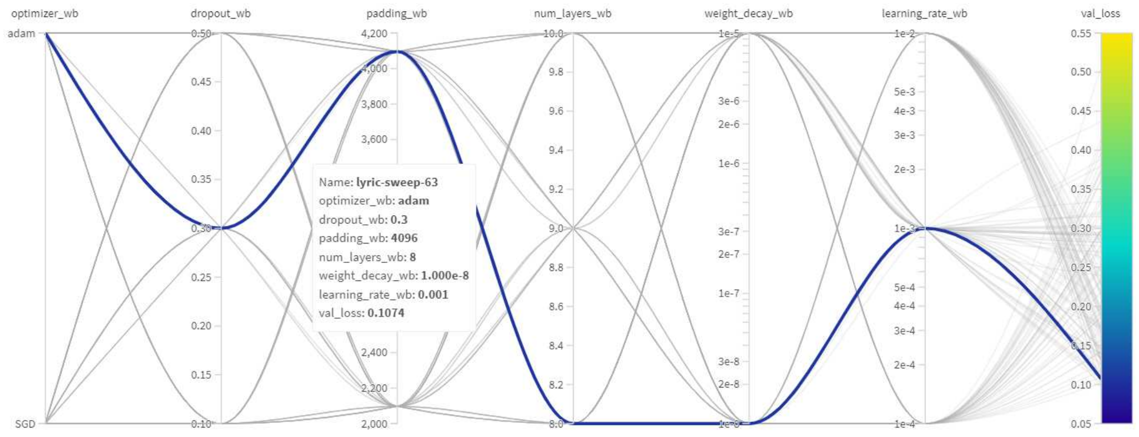
Name: **lyric-sweep-63**
optimizer_wb: **adam**
dropout_wb: **0.3**
padding_wb: **4096**
num_layers_wb: **8**
weight_decay_wb: **1.000e-8**
learning_rate_wb: **0.001**
val_loss: **0.1074**

Figura 6.1: Validation loss respect to possible combination of hyperparameters.

The difference with the previous architecture are that now the learning rate is set to 0.001 while before it was 0.0005. The weight decay is now 1e-8 and the dropout is 0.1 while previously they were 1e-5 and 0.3 respectively. Moreover the encoder block inside the Transformer architecture are now 8 instead of 9. This mean that the self-attention mechanism has a reduce attention window of a maximum of 256 instead of 512. Finally the padding size remains unchanged to 4096.

In Tab.6.2 we can notice the improvement in the results thanks to this new configuration of the architecture. This new architecture is able to improve the Mean over Frame of ≈ 12% respect the original features and ≈ 8% respect the features find by the previous configuration. Also the F1-score increase to 29.53 improving the previous results that are around 23. Moreover we can notice that this new architecture performs much better for certain activities respect to others. For example in the case of Cereals and Coffee it reaches a MoF of 55.6% and 56.25% respectively, while for activities such as Pancake and Scrambledegg it reaches only a MoF of 34.65% and 38.85%. This is due for two main reasons: the number of cluster and the total number of concatenated frames for each activity. In fact activities *Cereals* has only 5 possible actions (the lowest with Milk) and *Coffee* has the lowest number of total frames. Instead *Pancake* and *Scrambledegg* are the ones with more actions, respectively 14 and 12, and also *Pancake* is the one with the most total number of frames, 937125.

	Original		**New Features**		**Best Parameters**	
Activity	MoF	F1	MoF	F1	MoF	F1
Cereals	42.16	30.42	48.52	32.63	55.16	37.66
Coffee	32.48	19.56	38.48	19.76	56.25	27.98
Friedegg	27.69	18.75	30.78	19.74	43.10	26.04
Juice	37.25	27.55	44.72	28.82	54.65	33.15
Milk	46.30	36.76	37.06	24.38	48.77	34.64
Pancake	27.34	16.75	27.00	17.33	34.65	23.83
Salad	32.33	27.06	39.53	28.18	43.83	31.68
Sandwich	30.61	18.25	38.60	22.80	42.12	29.05
Scrambledegg	30.37	17.89	34.97	21.50	38.85	25.15
Tea	37.13	20.23	40.47	20.27	44.04	26.09
Total	34.37	23.32	38.01	23.54	**46.14**	**29.53**

Tabella 6.2: Results for every activity considering the original features, the first features extracted by the network (New Features) and the one extracted by the improved network (Best Parameters)).

After we found the best hyperparameters for our architecture, we wanted to test out if the hierarchical structure of the encoder block is essential for this problem. To do so we implemented the same architecture as the previous one but we fixed the size of self-attention window to 512, the maximal size of the previous architecture, for all encoder blocks. While we were training the Siamese network for each stage we noticed a worsening on the accuracy and of course an increment in the time required to train each epoch. For example for the last stage, the one considering the combination of the previous stages, this new architecture, without the hierarchical self-attention window, got an accuracy of 81.40% that is much worse than the previous case, in fact previously we got an accuracy of 87.96%. Moreover the time required increase a lot, now the architecture need ≈40 minutes to compute an epoch, that is almost the double if we considered the previous case, where it needed 'only' ≈27 minutes. Nevertheless we computed the new features of the videos, as done previously, in order to check the results without considering the hierarchical representation. With this modified architecture we obtained a MoF of 30.96% and a F1-score of 18.81. This results are much worse if we consider the previous case, where we obtained a MoF of 46.14% and a F1-score of 29.53. Moreover the time required to train an epoch for each stage is almost duplicated. This prove, also empirically, that the use of an hierarchical representation is essential to find out significant relations between frames of the same action.

6.1.2 Comparative results

Finally, we compare the results obtained by our model with those of *state-of-the-art* architectures. In Tab.6.3 are reported the Mof and the F1 score of the unsupervised and weakly-supervised models. First of all we can notice that the best results are obtained by CAD[34] for what concern MoF and by UDE[30] if we consider the F1-score. The probable reason for their results is that they used the Inflated 3D ConvNet (I3D) features representation, instead of the Improved Dense Trajectory (IDT) + Fischer Vector (FV) ones. As stated previously, we used the FV representation due to the time required to train the whole network and to compare to most of *state-of-the-art* methods that have been tested taking FV features as input. In fact, if we consider only the models that are using the FV features representation, our model reach similar results to the others, in fact is not the best in term of MoF and F1 score but is close enough.

Model	**Year**	**Input/Feature**	**F1**	**MoF**	**Scenario**
[35] Mallows	2018	IDT + FV	-	34.6	unsupervised
[63] Prism	2019	IDT + FV	-	33.5	unsupervised
[36] CTE	2019	IDT + FV	26.4	48.1	unsupervised
[31] JVT	2021	IDT + FV	29.9	41.8	unsupervised
[34] CAD	2021	IDT + FV	-	49.5	weakly-supervised
[34] CAD	2021	I3D	-	**53.1**	weakly-supervised
[30] UDE	2021	I3D	**31.9**	47.4	unsupervised
[64] TOT	2021	IDT + FV	31.0	47.5	unsupervised
Ours	2023	IDT + FV	29.5	46.1	weakly-supervised

Tabella 6.3: Performance of unsupervised and weakly-supervised methods evaluated on the Breakfast Actions dataset. Weakly-supervised methods use the same amount of information of unsupervised ones.

6.2 Qualitative results

For representing the new features representation we decided to use the t-distributed Stochastic Neighbor Embedding (t-SNE) algorithm. t-distributed Stochastic Neighbor Embedding is a dimensionality reduction algorithm developed by L. van der Maaten et al. in 2008 [65] To be more specific t-SNE is a nonlinear dimensionality reduction algorithm, and can be used instead of PCA. t-SNE is mostly used to understand high-dimensional data and project it into low-dimensional space, like in 2D. This algorithm is non-deterministic and iterative, so each time it runs it obtains different representations. For this reason it can be used to try to understand high-dimensional datasets but not to perform dimensionality reduction for ML training, like PCA.

In Fig. 6.2 and Fig. 6.3 are plotted difference t-SNE representation for different activity both in the case of video and activity level. The dimension of the features are reduced from 64 to 2 and each color in the plots represent a different action. The first difference we can notice between the two levels is

that the video level is much more easier respect to the activity one. Of course this is due to the fact that at video level there are much less frames respect considering all video from an activity and, moreover, because the frames are more similar, done by the same person in the same kitchen. Even for more difficult cases like 'P41_cam01_P41_pancake', top left in Fig. 6.2, clustering the action should not be to difficult since each action are well separated, especially in the case of the modified feature by our model.
The case is completely different if we check the t-SNE representation at activity level. From Fig.6.3 we can notice that algorithm is not able to find some pattern and divide correctly each action within their respect activity. All representation look like a random cloud of points, due to the huge amount of input data (frames). In some activity like 'coffee' and 'sandwich' (bottom right and top left) we can distinguish some area where some colors (actions) are more present than the others. But in cases like 'friedegg' (bottom left) is almost impossible to distinguish these area, due also to the 'background' action, light green in the plot, that is much more numerous than the other ones.

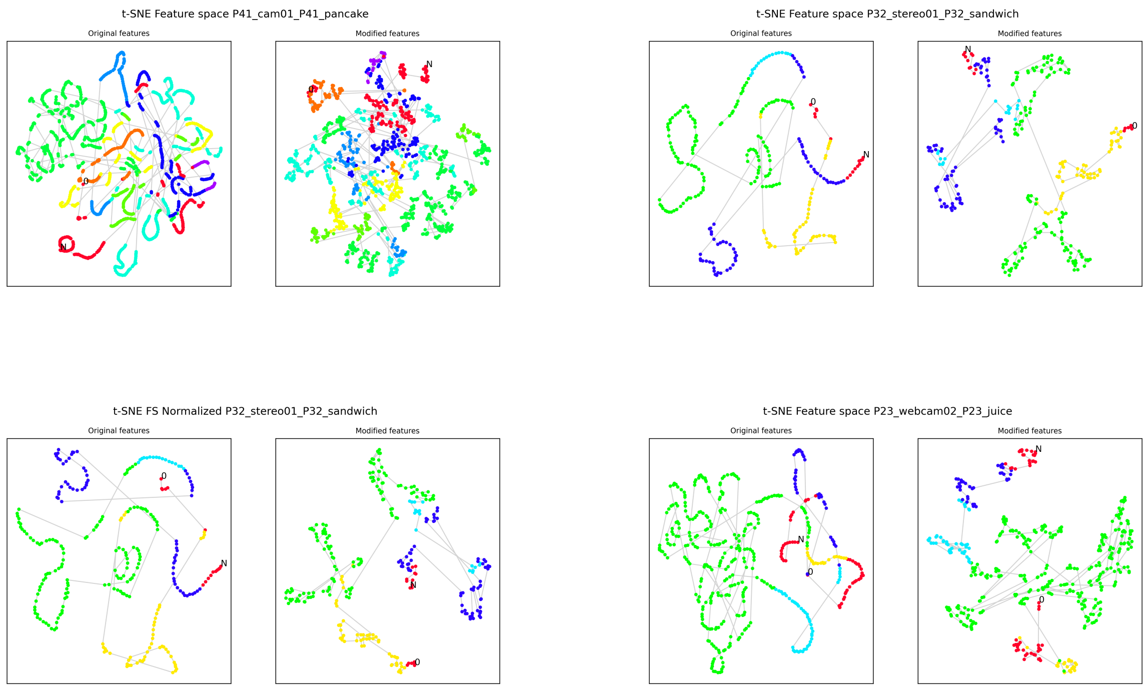

Figura 6.2: t-SNE features representation at video level.

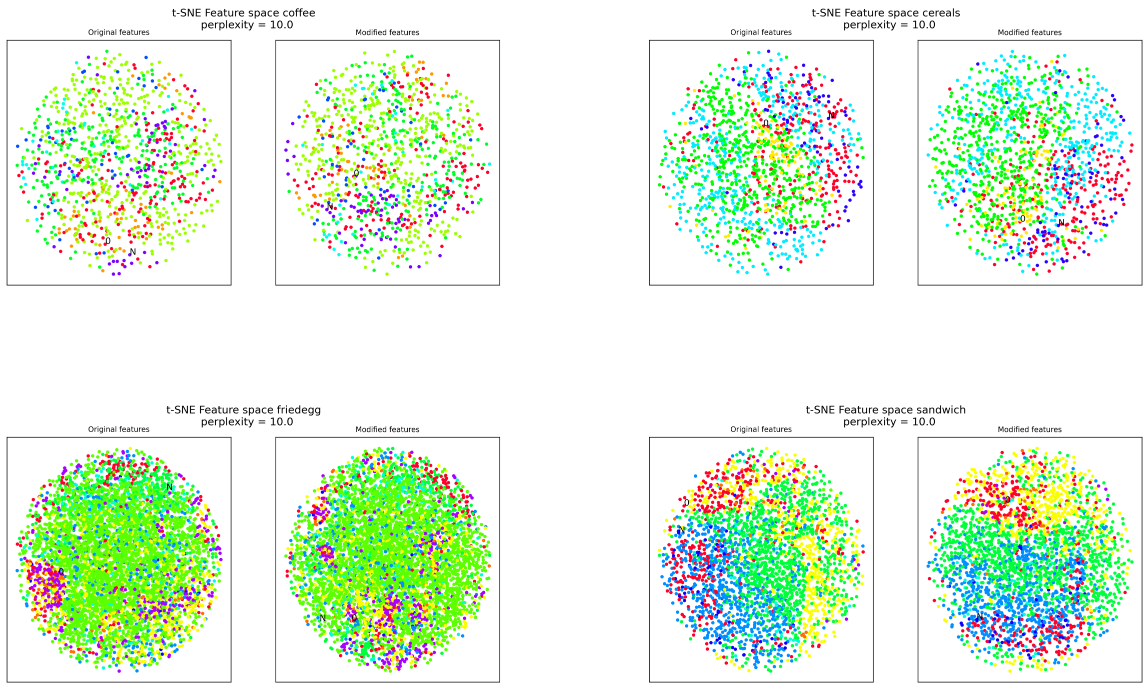

Figura 6.3: t-SNE features representation at activity level.

6.2.1 Segmentation result

To get a better understanding on how good our model works, we reported four action segmentation result in Fig.6.4. Four each of the four example are reported the action cluster (of a single video) based on three different feature representations. In fact, we have in the second line the result of the clustering obtained by applying K-means to the original features (ORI), in third line in the case of the features extracted by our model in the first run (FIRST), and finally in the final line are reported the actions cluster obtained with the features extracted by the network after the tuning of the hyperparameters. Moreover, in the first line are reported the ground truth (GT) and of course at each color corresponds a particular action. These bar plot gives us some useful insight information about the results, that can be more explicit than just the MoF and F1-score. The first thing that we can notice from the image is that the results obtained with the first new features representation learned by the network (FIRST) suffer a lot the problem of over-segmentation. In fact, even if the results are similar to the original features, from the image we can see that in this scenario the video is divided multiples times in small and different segments, suggesting that this features representation is not able to capture the temporal relationship between frames and so it lost the natural continuity of the actions. Instead, in the FINAL case we can notice that this problem of over-segmentation is less present, is still not as good as the ground truth but this may be the reason of the huge improvement in the results compared to the two other methods (ORI and FIRST).

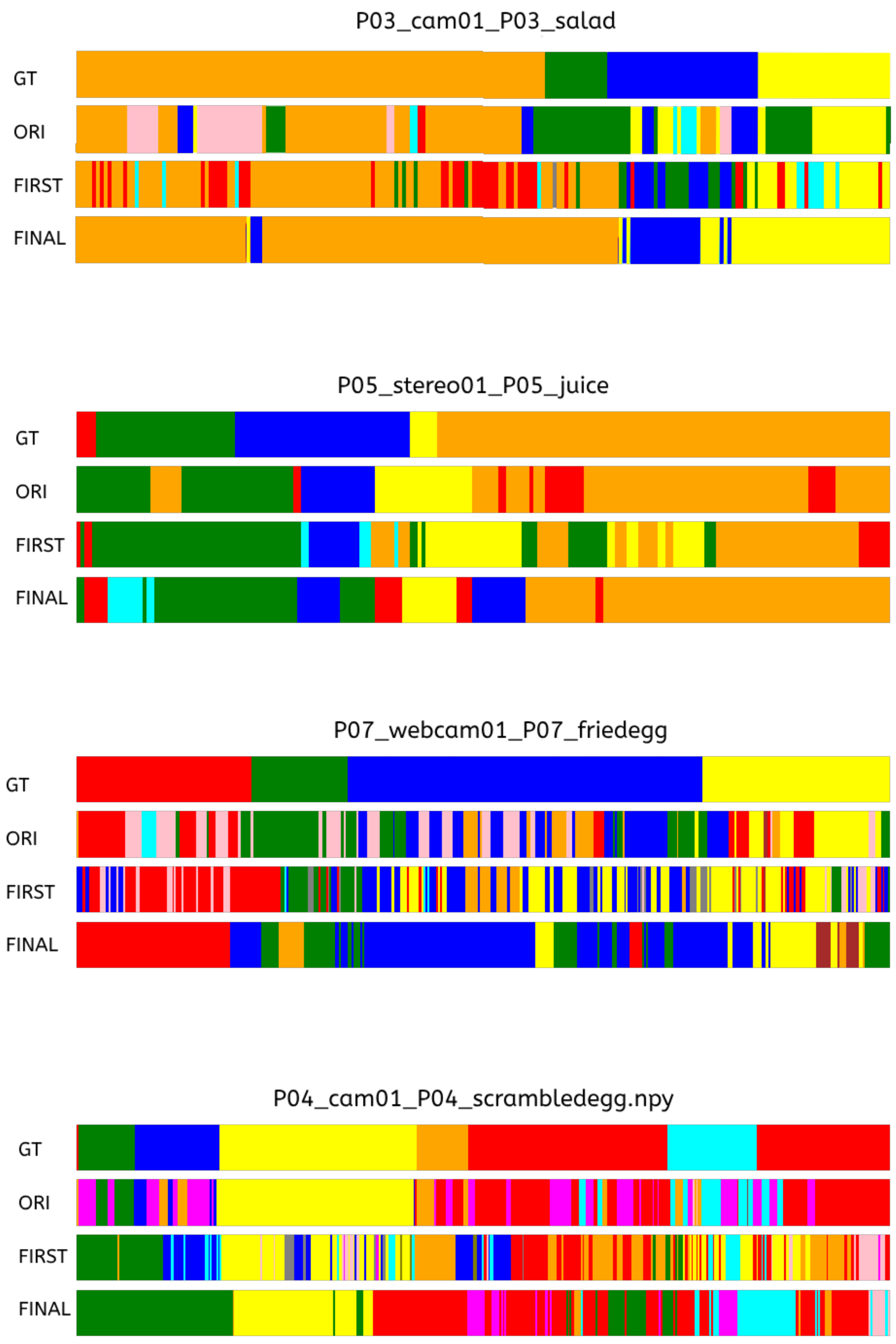

Figura 6.4: Segmentation results with different features representation. Here in the images GT represent the ground truth of each frames respect to their actions. ORI are the segmentation results using the original features. FIRST are the segmentation results using the first configuration of our architecture. FINAL are the segmentation results using the best hyperparameters combination for our architecture.

Capitolo 7

Conclusion

This work addressed the problem of Temporal Action Segmentation on video data, at activity level in a weakly-supervised fashion. The main contribution is a self-supervised transformer based architecture that learns action representations useful in a weakly-supervised scenario, hence significantly reducing the annotation cost of existing fully-supervised transformer-based architectures as ASFormer[7] and UVAST[27].

To achieve this result, we analyzed the *state-of-the-arts* architectures and we chose the ASFormer model as starting point for our project. The first part was focused on the understanding if this architecture was able to fit this problem in the weakly-supervised scenario. After changing and improved the ASFormer architecture we moved on the solution of the problem. We used our improved network to construct a Siamese architecture, which scope is a binary classification tasks, that has to distinguish between video of different activities and learn new frame representation. The key point for a good training of the Siamese architecture was to implement curriculum learning, that implied to divide the dataset (videos) in different stages, from the easiest pairs to the most complex ones. Finally all the new videos representations, of the same activity, were passed to a clustering algorithm, the k-means, to classify each frames. Compared to the *state-of-the-arts* the results obtained with our architecture are quite satisfactory. In fact there are not huge improvement on the actual results but they are align with the others, suggesting that even the Transformer architecture can be implemented for this kind of problem and, maybe, with some further studies on it, it can also surpass the actual *state-of-the-arts* results.

7.1 Future work

Since using the transformer-based architecture to solve the Temporal Action Segmentation problem is largely unexplored in the scenario of unsupervised and weakly-supervised learning, there are a lot of possibilities to improve the results.

First of all, the model should be tested on other datasets to get more insights about strengths and weaknesses points of the architecture. Also it can be useful to understand if the model is able to generalize and still work efficiently using video of different nature and with different degrees of complexity in terms of temporal semantics. Also, another possible strategy to improve the results, could be doing some boundary refinement to reduce the problem of over-segmentation. This strategy was introduced by Z. Wang et al. [66] for multi-stage segmentation algorithms. It is not designed for a Transformer architecture, but it could be adapted to work with it.

Finally, the last thing as possible future work and probably the most difficult one, is trying to work as a global level instead of activity level. For unsupervised and weakly-supervised scenario there are not relevant *state-of-the-arts* methods working at this level. The main difficulty is that the architecture needs to model the inter-activity association between actions. Moreover, if the model classifies the wrong activity for a video the model, it is likely to misclassify all the frames, since the action between activities are different.

Bibliography

[1] Bharat Singh, Tim K. Marks, Michael Jones, Oncel Tuzel, and Ming Shao. A multi-stream bi-directional recurrent neural network for fine-grained action detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1961–1970, 2016.

[2] Hong-Bo Zhang, Yi-Xiang Zhang, Bineng Zhong, Qing Lei, Lijie Yang, Xiang Du, and Duan-Sheng Chen. A comprehensive survey of vision-based human action recognition methods. *Sensors*, 19:1005, 02 2019.

[3] Harshala Gammulle, David Ahmedt-Aristizabal, Simon Denman, Lachlan Tychsen-Smith, Lars Petersson, and Clinton Fookes. Continuous human action recognition for human-machine interaction: A review, 2022.

[4] Yazan Abu Farha and Juergen Gall. MS-TCN: multi-stage temporal convolutional network for action segmentation. *CoRR*, abs/1903.01945, 2019.

[5] Alexander Richard, Hilde Kuehne, and Juergen Gall. Weakly supervised action learning with rnn based fine-to-coarse modeling, 2017.

[6] Yifei Huang, Yusuke Sugano, and Yoichi Sato. Improving action segmentation via graph-based temporal reasoning. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14021–14031, 2020.

[7] Fangqiu Yi, Hongyu Wen, and Tingting Jiang. Asformer: Transformer for action segmentation. *CoRR*, abs/2110.08568, 2021.

[8] Z. Kolter J. Gall M. Noroozi N. Behrmann, S. A. Golestaneh. Unified fully and timestamp supervised temporal action segmentation via sequence to sequence translation. In *European Conference on Computer Vision (ECCV)*, 2022.

[9] Petru Soviany, Radu Tudor Ionescu, Paolo Rota, and Nicu Sebe. Curriculum learning: A survey, 2022.

[10] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA, 2009. Association for Computing Machinery.

[11] Terence Sanger and Pallavi N. Baljekar. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

[12] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry.* MIT Press, Cambridge, MA, USA, 1969.

[13] Marvin L. Minsky and Seymour A. Papert. *Perceptrons: Expanded Edition.* MIT Press, Cambridge, MA, USA, 1988.

[14] Terrence J. Sejnowski. *The Deep Learning Revolution.* The MIT Press, 2018.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[16] Guodong Ding, Fadime Sener, and Angela Yao. Temporal action segmentation: An analysis of modern techniques, 2023.

[17] Colin Lea, Michael D. Flynn, Rene Vidal, Austin Reiter, and Gregory D. Hager. Temporal convolutional networks for action segmentation and detection, 2016.

[18] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks, 2018.

[19] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection, 2021.

[20] Yaser Souri, Yazan Abu Farha, Fabien Despinoy, Gianpiero Francesca, and Juergen Gall. Fifa: Fast inference approximation for action segmentation, 2021.

[21] Juergen Gall Hilde Kuehne, Alexander Richard. Weakly supervised learning of actions from transcripts, computer vision and image understanding. pages 78–89, 2017.

[22] De-An Huang, Li Fei-Fei, and Juan Carlos Niebles. Connectionist temporal modeling for weakly supervised action labeling, 2016.

[23] Alexander Richard, Hilde Kuehne, and Juergen Gall. Action sets: Weakly supervised action segmentation without ordering constraints, 2018.

[24] Jun Li and Sinisa Todorovic. Set-constrained viterbi for set-supervised action segmentation, 2020.

[25] Mohsen Fayyaz and Juergen Gall. Sct: Set constrained temporal transformer for set supervised action segmentation, 2020.

[26] Zhe Li, Yazan Abu Farha, and Juergen Gall. Temporal action segmentation from timestamp supervision, 2021.

[27] Ozan Sener, Amir Zamir, Silvio Savarese, and Ashutosh Saxena. Unsupervised semantic parsing of video collections, 2016.

[28] Daniel Fried, Jean-Baptiste Alayrac, Phil Blunsom, Chris Dyer, Stephen Clark, and Aida Nematzadeh. Learning to segment actions from observation and narration. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2569–2588, Online, July 2020. Association for Computational Linguistics.

[29] E. Bueno-Benito, B. Tura, and M. Dimiccoli. Leveraging triplet loss for unsupervised action segmentation, 2023.

[30] Sirnam Swetha, Hilde Kuehne, Yogesh S Rawat, and Mubarak Shah. Unsupervised discriminative embedding for sub-action learning in complex activities, 2021.

[31] Rosaura G. VidalMata, Walter J. Scheirer, and Hilde Kuehne. Joint visual-temporal embedding for unsupervised learning of actions in untrimmed sequences. *CoRR*, abs/2001.11122, 2020.

[32] M. Saquib Sarfraz, Naila Murray, Vivek Sharma, Ali Diba, Luc Van Gool, and Rainer Stiefelhagen. Temporally-weighted hierarchical clustering for unsupervised action segmentation, 2021.

[33] Sagie Benaim, Ariel Ephrat, Oran Lang, Inbar Mosseri, William T. Freeman, Michael Rubinstein, Michal Irani, and Tali Dekel. Speednet: Learning the speediness in videos, 2020.

[34] Guodong Ding and Angela Yao. Temporal action segmentation with high-level complex activity labels, 2022.

[35] Fadime Sener and Angela Yao. Unsupervised learning and segmentation of complex activities from video, 2018.

[36] Anna Kukleva, Hilde Kuehne, Fadime Sener, and Juergen Gall. Unsupervised learning of action classes with continuous temporal embedding, 2019.

[37] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020.

[38] Petru Soviany, Radu Tudor Ionescu, Paolo Rota, and Nicu Sebe. Curriculum learning: A survey, 2022.

[39] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742, 2006.

[40] Yann" "LeCun, Sumit" "Chopra, Raia" "Hadsell, M" "Ranzato, and F" "Huang. A tutorial on energy-based learning. 2006.

[41] Dhruv Patel, Abhinav Jain, Simran Bawkar, Manav Khorasiya, Kalpesh Prajapati, Kishor Upla, Kiran Raja, Raghavendra Ramachandra, and Christoph Busch. Srtgan: Triplet loss based generative adversarial network for real-world super-resolution, 2022.

[42] Peng Lei and Sinisa Todorovic. Temporal deformable residual networks for action segmentation in videos. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6742–6751, 2018.

[43] H. "Kuehne, A. B. Arslan, and T." Serre. "the language of actions: Recovering the syntax and semantics of goal-directed human activities". In *"Proceedings of Computer Vision and Pattern Recognition Conference (CVPR)"*, "2014".

[44] Ziwei Liu, Zhongqi Miao, Xiaohang Zhan, Jiayun Wang, Boqing Gong, and Stella X. Yu. Large-scale long-tailed recognition in an open world, 2019.

[45] Heng Wang, Alexander Kläser, Cordelia Schmid, and Cheng-Lin Liu. Action recognition by dense trajectories. In *CVPR 2011*, pages 3169–3176, 2011.

[46] Florent Perronnin, Jorge Sánchez, and Thomas Mensink. Improving the fisher kernel for large-scale image classification. volume 6314, pages 143–156, 09 2010.

[47] Hilde Kuehne, Juergen Gall, and Thomas Serre. An end-to-end generative framework for video segmentation and recognition, 2016.

[48] João Carreira and Andrew Zisserman. Quo vadis, action recognition? A new model and the kinetics dataset. *CoRR*, abs/1705.07750, 2017.

[49] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

[50] Alireza Fathi, Xiaofeng Ren, and James M. Rehg. Learning to recognize objects in egocentric activities. In *CVPR 2011*, pages 3281–3288, 2011.

[51] Sebastian Stein and Stephen J. McKenna. Combining embedded accelerometers with computer vision for recognizing food preparation activities. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '13, page 729–738, New York, NY, USA, 2013. Association for Computing Machinery.

[52] Dimitri Zhukov, Jean-Baptiste Alayrac, Ramazan Gokberk Cinbis, David Fouhey, Ivan Laptev, and Josef Sivic. Cross-task weakly supervised learning from instructional videos, 2019.

[53] Yansong Tang, Dajun Ding, Yongming Rao, Yu Zheng, Danyang Zhang, Lili Zhao, Jiwen Lu, and Jie Zhou. Coin: A large-scale dataset for comprehensive instructional video analysis, 2019.

[54] Grazia Cicirelli Roberto Marani, Laura Romeo. Ha4m - human action multimodal monitoring in manufacturing.

[55] Dima Aldamen, Davide Moltisanti, Evangelos Kazakos, Hazel Doughty, Jonathan Munro, William Price, Michael Wray, Tobias Perrett, and Jian Ma. Epic-kitchens-100, 2020.

[56] Yizhak Ben-Shabat, Xin Yu, Fatemehsadat Saleh, Dylan Campbell, Cristian Rodriguez-Opazo, Hongdong Li, and Stephen Gould. The ikea asm dataset: Understanding people assembling furniture through actions, objects and pose. 2020.

[57] Francesco Ragusa, Antonino Furnari, and Giovanni Maria Farinella. Meccano: A multimodal egocentric dataset for humans behavior understanding in the industrial-like domain, 2022.

[58] Luowei Zhou, Chenliang Xu, and Jason J. Corso. Towards automatic learning of procedures from web instructional videos, 2017.

[59] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[60] Colin Lea, Austin Reiter, Rene Vidal, and Gregory D. Hager. Segmental spatiotemporal cnns for fine-grained action segmentation, 2016.

[61] Martin Ester, Hans-Peter Kriegel, Jorg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

[62] Weights biases site. `https://wandb.ai/site`.

[63] Karan Goel and Emma Brunskill. Learning procedural abstractions and evaluating discrete latent temporal structure. In *International Conference on Learning Representations*, 2019.

[64] Sateesh Kumar, Sanjay Haresh, Awais Ahmed, Andrey Konin, M. Zeeshan Zia, and Quoc-Huy Tran. Unsupervised action segmentation by joint representation learning and online clustering, 2023.

[65] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[66] Zhenzhi Wang, Ziteng Gao, Limin Wang, Zhifeng Li, and Gangshan Wu. Boundary-aware cascade networks for temporal action segmentation. In *EC-CV (25)*, volume 12370 of *Lecture Notes in Computer Science*, pages 34–51. Springer, 2020.