Universitat de Barcelona

Treball final de grau

GRAU DE MATEMÀTIQUES

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

Using Deep Learning for Fine-Grained Action Segmentation

Autor: Joaquim Yuste

Director:	**Dr. Albert Clapés i Dr. Sergio Escalera**
Realitzat a:	**Departament de Matemàtiques i Informàtica**

Barcelona, 20 de juny de 2021

Abstract

This project focuses on video action segmentation task, which aims to temporally segment and classify fine-grained actions in untrimmed videos. The development and refinement of this process is an important yet challenging problem, which can provide great improvements in work areas such as robotics, e-Health assistive technologies, surveillance, and beyond.

On the one hand, we will study the current state-of-the-art, as well as the metrics that are commonly used to evaluate an architecture on this kind of problems. On the other hand, we introduce two different attention-based modules that are capable of extracting frame-to-frame relationships, and a behaviour analysis will be performed by evaluating them over Georgia Tech Egocentric Activity (GTEA), which is an outstanding dataset. This dataset is focused on daily cooking activity videos, with fine-grained labels, and it has an egocentric point view. Eventually, we will compare the obtained results against the actual state-of-the-art scores, in order to discuss the effectiveness of each module.

Resum

Aquest projecte es basa en la detecció d'accions dins d'un vídeo, aquesta tasca té com a objectiu la generació de segments precisos al llarg de la dimensió temporal, on cadascun dels segments contindrà una acció concreta. El perfeccionament d'aquest procés aporta un gran benefici en els treballs realitzats en àmbits com la robòtica, la cuina i la vigilància, entre altres.

Primer de tot s'estudiaran els darrers estats de l'art en aquest sector, així com les mètriques més habituals que es fan servir a l'hora d'avaluar una arquitectura. A continuació es presentaran dos mòduls basats en mecanismes d'atenció que tenen, com a principal característica, la capacitat d'extreure relacions entre les dades introduïdes, i s'elaborarà una anàlisi del comportament avaluant-los dins de Georgia Tech Egocentric Activity (GTEA), un conjunt de vídeos centrats en la preparació d'aliments des d'un punt de vista egocèntric. Finalment, es compararan els resultats entre els nous mòduls i els actuals estats de l'art, amb l'objectiu de debatre l'eficàcia de cadascun.

Resumen

Este proyecto se centra en la detección de acciones sutiles dentro de un video, dicho problema tiene como objetivo generar segmentos precisos a lo largo de la dimensión temporal, donde cada segmento contendrá una acción concreta. El perfeccionamiento de este proceso es de gran utilidad para tareas realizadas en campos como la robótica, la cocina y la vigilancia, entre otros.

Primero se estudiarán los actuales estados del arte en este ámbito, así como las métricas más comunes que se utilizan a la hora de evaluar una arquitectura. A continuación se presentarán dos módulos basados en mecanismos de atención que poseen, como principal característica, la capacidad de extraer relaciones entre los datos introducidos, y se realizará un análisis del comportamiento evaluándolos dentro de *Georgia Tech Egocentric Activity*

(GTEA), un conjunto de videos centrados en actividades diarias en torno a la preparación de alimentos con una perspectiva egocéntrica. Por último, se compararán los resultados entre los nuevos módulos y los actuales estados del arte, con el objetivo de debatir la eficacia de cada uno de ellos.

Contents

Chapter 1

Introduction

1.1 Problem to address

Given an untrimmed video the task of *fine-grained temporal action segmentation* consists on basically determining the class label for each of the contained temporal subparts (actions). This is an immensely challenging task due to basically the often low inter-class but also large intra-class variability among video segments. In other words, the temporal action segments in one video are often much more alike compared to others from the same class in other videos. The emphasis on the study of this task leads to a significant improvement for the application in areas such as supermarket surveillance cameras [1], surgical phase recognition [2, 3, 4], food preparation activities [5, 6, 7], gymnastic activities [8] or even assembly processes [9, 10].

One of the main bottlenecks for the fine-grained temporal action segmentation is the difficulty of collecting high-quality large and densely-annotated datasets to correctly train deep neural networks in a supervised manner, for this reason, the current datasets available for this task nowadays are quite compact, so they can cause several over fitting problems on architectures with great amount of parameters.

However, with the amount of video hours being uploaded to the Internet every second, the task of automatically recognize and understand its underlying meaning has gained much attention.

As the task needs a frame-wise classification, the final prediction will be performed over the original temporal resolution, thus there are two possible approaches, either an encoder-decoder model [11, 12], or an architecture that can process a video maintaining its length unchanged [13, 14]. Although the first one has some benefits, like decreasing over-segmentation errors, the reduction of the temporal length in the encode stage can cause the loss of highly relevant information, which could compromise in the final prediction. The second one, and the most currently used, is commonly based on dilated convolutions, this kind of convolutions can acquire large receptive fields without increasing the number of parameters, neither shrinking the video's temporal length.

1.2 Objectives

The goal of this project is to acquire and assimilate the knowledge needed to understand the behavior behind a deep neural network, as well as being able to apply it, in order to analyze and learn from the results.

Specifically, what we aim for, is to modify a Multi-Stage Temporal Convolutional Network (MS-TCN), a concrete architecture of a convolutional deep neural network, by adding Self-Attention modules on top, in an attempt to improve the MS-TCN baseline score.

We will evaluate two different modules into this architecture, first, a Transformer-like structure, based on natural language processing methods, takes place, then, a fully convolutional Self-Attention will be used, in order to compare and analyze the results between both of them.

1.3 Summary of work done

First of all, an intensive study of the deep neural network background was made, for the purpose of learn the main concept. After that, the major objective was searching the most basics but relevant papers on video classification, to see how the convolutional neural networks were applied to work with video inputs.

Once assimilated, the next step was the research of the state-of-the-art in action detection. However, there are many task involving the classification of actions in a video, where each one has its own way to go, in this case, as the investigation progressed, the action segmentation task was attracting the most interest, consequently, the study was converging in that area, searching which models were used there.

After some more research the idea was clear, analyze a Temporal Convolutional Network, architecture that got the best results on the leaderboard, together with its variants, and evaluate it on GTEA dataset to obtain a baseline results.

The final step was adding and analyzing a Self-Attention mechanism. This module, popularized in [15] for natural language process, is getting great interest in different task, but was not applied for action segmentation yet, so the final procedure was introducing this mechanism into the architecture to determine if it can contribute in a better prediction results on action segmentation as well.

Chapter 2

Background

2.1 Artificial Neural Network

An artificial neural network (NN) is a machine learning technique, which was originally inspired by the behavior of biological neurons, the main functionality of these neurons is to collect and send signals from one to another.

The work around of this methodology, is to group several artificial neurons (also known as perceptrons or nodes), to generate a final prediction result. NN aims to find abstract relations between data to classify it, in order to achieve this, every neuron will be learning a set of parameters (weights) to aggregate the incoming signals from the neurons in the previous layer and produce a response. The responses of the different neurons in some arbitrary layer are sometimes referred to as features. In order to generate a prediction, the networkss include a last linear layer to project the features produced so far into a distribution of class scores.

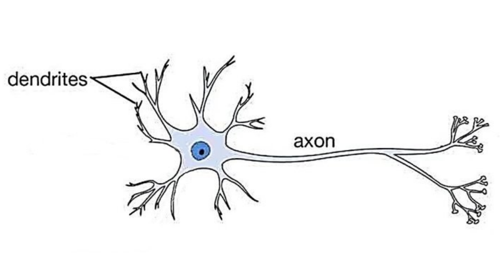
2.1.1 Perceptron

Biological neurons are made up of dendrites, the body cell and the axon 2.1 (a). Dendrites will capture a signal from a receptor cell or another neuron, then, this signal will be converted into an electrical pulse. Finally, according to the electrical signals that the neuron has received, this one will generate or not an output through the axon to another neurons.

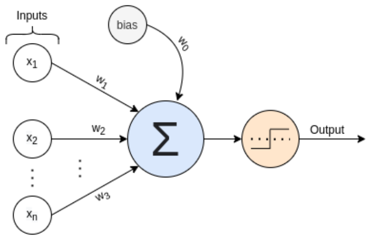
The Perceptron works mostly the same, given a set of inputs, each of these will be converted (multiplied) by a learned weights, and the results will be summed up. Eventually, the result will be passed through an activation function to determine if the Perceptron should be triggered up 2.1 (b). A bias value (commonly set to 1) is often used along with the entries as follows.

$$f(X) = w_0 * b + \sum_{i=1} w_i * x_i$$

This allows a shifting rate to the activation function



(a)



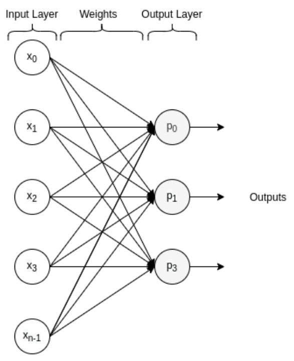
(b)

Figure 2.1: Differences between neurons: (a) Biological neuron; (b) Artificial neuron

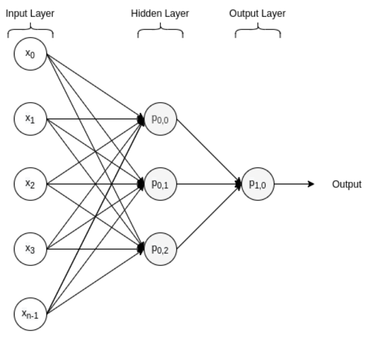
2.1.1.1 Multi-Layer Perceptron

A unique layer of Perceptrons can only carry out very simple tasks that are linearly separable, like logic gates (except XOR gate).

Thus, to overcome this limitation, a Multi-Layer Perceptron was introduced. Its structure consist of two or more layers, without counting the input layer, since it has no computational cost. The layers in between of input and output ones are called hidden layers. With only one hidden layer the network can approximate any continuous function. A Multi-Layer Perceptron with more than one hidden layer is called Deep Neural Networks, and increasing the number of hidden layers makes the network capable of retrieving higher degree functions more accurately.



(a)



(b)

Figure 2.2: Firsts Artificial Neural Networks: (a) Single-Layer Perceptron; (b) Multi-Layer Perceptron

2.1.2 Activation Functions

There are a few common activation functions to decide how the behavior of an output node will be displayed, depending on the task that a node will be performing, some functions will produce better results than others.

- **Binary Step**
  Is the most simple one, the Binary Step function makes use of a threshold so, every input that exceeds this value would lead into the activation of the node. It does not provide much information about the prediction, so we can not know if its a positive/negative prediction with great or low confidence rate.

$$f(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.1}$$

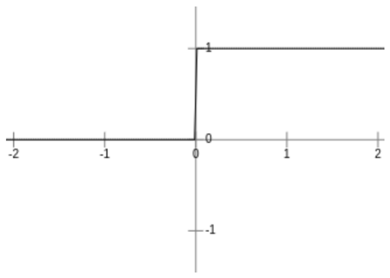


Figure 2.3: Representation of Binary Step function

- **ReLU**
  The Rectified Linear Unit is the most commonly used activation function in the hidden layers. As the Binary Step, it uses a threshold to determine if the neuron is activated, but in this case, if the value is higher than the threshold, the function will let the value pass unchanged. Since it has a low computational cost and its derivative is either 0 or 1, it's useful for both, very deep networks and error propagation (for the learning state).

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

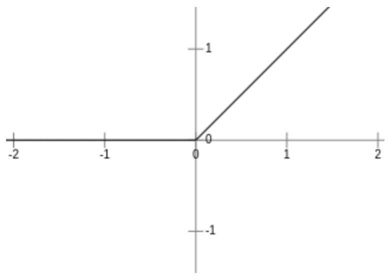


Figure 2.4: Representation of ReLU function

- **Sigmoid**
  The most intuitive way to represent a prediction confidence for the human brain is

using a probability distribution. Sigmoids offers an output which ranges between 0 and 1. Since ReLU function is mainly used in hidden layers, the probabilistic approach that provides the Sigmoid function makes it a regular function for output layers.

$$f(x) = \frac{1}{1 + e{-}x} \tag{2.3}$$

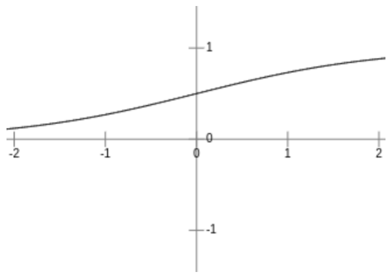


Figure 2.5: Representation of Sigmoid function

- **SoftMax**
  Although the above function produce a probability distribution output, can not be used for multi-class prediction, for the reason that results would not sum up to 1. In contrast, SoftMax is a function that operates over a given input vector by normalizing it and assigning a class probability to each value, in order to tackle multi-class classification tasks

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{2.4}$$

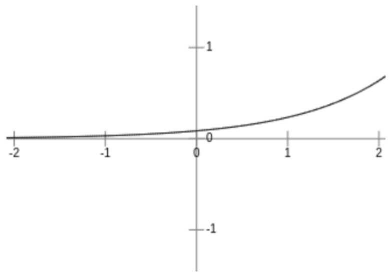


Figure 2.6: Representation of Softmax with [-2, -1, 0, 1, 2] as inputs

2.2 Training

The main advantage that a neural network has over other methodologies is stated by the Universal Approximation Theorem, which declares that any arbitrary function can be approximated by a Multi-Layer Perceptron, if it has at least one hidden layer and the enough amount of neurons within the same. Thus, a neural network have the ability to learn and solve almost any problem, regardless the complexity, since most of them can be thought as a function. However, this learning mechanism must be specified at some point. This section is focused on how a neural network can be taught to learn.

2.2.1 Loss Functions

There are some different learning strategies that can be applied to train an artificial neural network (i.e., to learn the proper weights for some predefined task), one of them is called fully-supervised learning. It consists of providing to the network inputs with its corresponding expected outputs, also known as ground truth. Then, on the training state, the network will generate predictions based on the given inputs and current weights, and will compare them with the also given ground truth labels. This is made by the Loss Function, which gives a numerical score that states how good was the prediction. Different learning tasks require using different loss functions. For classification and regression, typically, Cross-entropy and Mean Square Error Loss are used respectively.

2.2.2 Backpropagation

The learning capabilities relies on the optimization of the loss function. This optimization can be done by the gradient descent, which consists of updating the weights in the opposite direction of the obtained gradient using a learning rate, in order to minimize the loss function. However, extracting the gradient of every single node of the network is highly expensive, due to the number of parameters that a deep neural network can have at each layer.

To solve this problem, Backpropagation is an algorithm that recursively use the chain rule in order to compute the gradient with respect every weight. To do so, the transformations applied to the data must be divided into smaller steps, and the representation of this steps is known as computational graph. So, as an example, if we have some function $f(x, y, z, w) = (x + y)z + w$ with its computational graph shown in Figure 2.7, the partial derivative over the variable x would be as follows (2.5):

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial p}\frac{\partial p}{\partial x} \tag{2.5}$$

This way, the gradient obtained from a deeper layer can be reused (propagated) to calculate the new one of a higher layer, which is extremely efficient compared to computing the gradient of each neuron independently. Finally, the update of the weights will be decided by the optimization function, which determines how to use the extracted gradient.

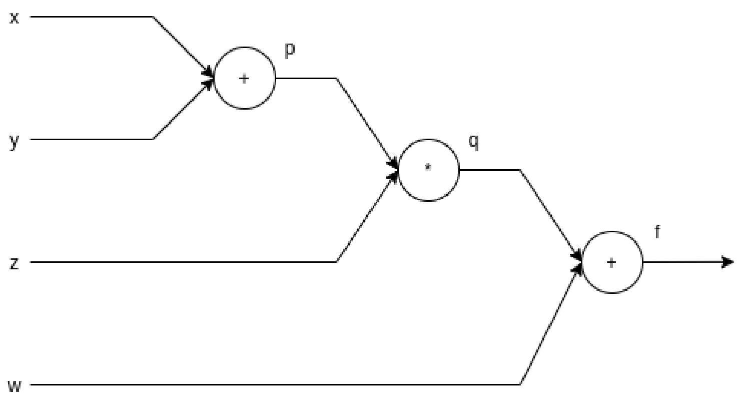


Figure 2.7: Computational graph of function $f(x, y, z, w) = (x + y)z + w$.

2.3 Convolutional neural network

Artificial Neural Networks can actually learn and resolve complex problems, however, does not have the capacity to understand highly correlated data inputs such as images, due to the fact that the input has to be flattened into a long vector, which can cause the loss of relevant local information. In addition, every neuron of an MLP layer, is fully connected to the outputs of the previous one, this kind of connections severely increases the number of parameters, e.g. given an image with 64x64 pixels, each neuron of the first hidden layer would have 4096 weights with 1 additional bias. This could imply some problems like redundancy or overfitting issues on deeper networks. To that end, it was introduced a convolutional version, which can process the input in a manner that the spatial structure of the data remains unchanged, and has the characteristic that its neurons make use of shared weights, this means that they have a limited amount of parameters, but are reused in different parts of the data. This technique makes the network easier to train, and brings the possibility to increase its depth, which allows the learning of more abstract features.

All of the above makes the CNN widely used for tasks involving inputs such as images, audio or even text in some cases. However, the convolution is not the only operation performed to the data, thus, a brief introduction of the layers most commonly used on this type of network will be presented below.

- **Convolutional Layer**
  The main one that gives to the network its name is the Convolutional Layer. In this type of layers, neurons are represented by filters, all parametrized by the same kernel size (receptive field) and stride. That is, a neuron of a convolutional layer only attends over a small regions of the input, although the same filter is convoluted through all the spatial positions of the input tensor by multiplying the input by the same (shared) weights, as it can be seen on Figure 2.8. The weights of the filters are adjusted during training to capture useful patterns in the data. Whereas filters on the first network layers will focus on simpler, more primitive patterns (like points or

lines), the ones in the latter layers will learn more complex abstract patterns (parts of objects or objects themselves).

The idea of a convolution is to multiply two functions, resulting on a maximum when both of them are the most similar. Therefore, a kernel with a learned pattern will be triggered up in those areas where the filter and the data are more resemble.

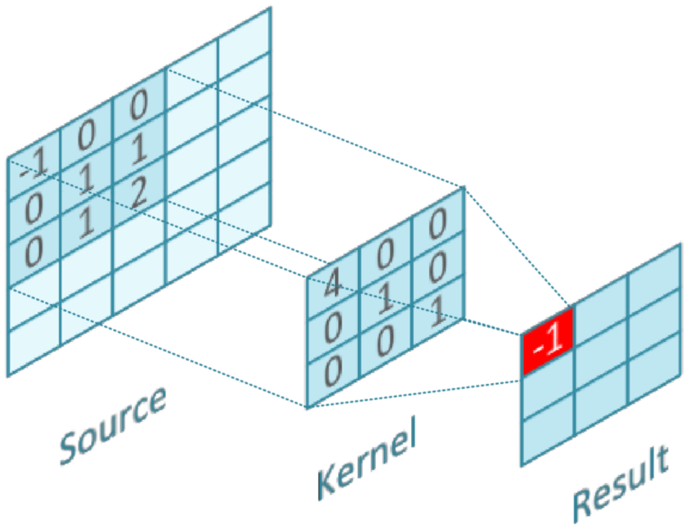


Figure 2.8: Scheme of a Convolution process. Image from [16]

The output tensor size of a convolutional layer is tied with the kernel size and the stride of its neurons, as the Figure 2.8 describes, which can be computed with the following equation (assuming that both input and kernel are square) $out_{size} = (in_{size} - k)/s + 1$, where k and s refers to kernel size and stride respectively. However, the addition of columns or rows to the input tensor allows to have more control on the output size regardless of the kernel size, this technique is known as *padding*, and the values of the new rows/columns are commonly initialized to zero.

There are three types of padding which are explained below, and Figure 2.9 shows how these paddings are applied to acquire each output size:

- Full: The output size is greater than the input size
- Same: The input and output have the same size
- Valid: The output has less size than the input

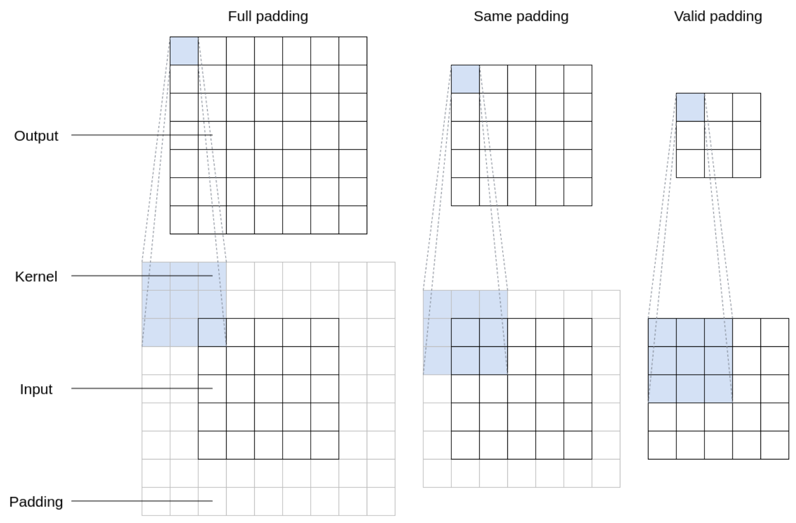


Figure 2.9: Employment of the padding technique.

- **Pooling Layer**
  Stacking several Convolutional Layers allows to learn from low-level features (edges or corners), to high-level ones (shadows or textures), however, these convolutions are localized on the input, and deep neural networks tends to memorize as much as possible, which means, that if the CNN finds some a feature in a concrete pixel of the image, it will try to find it again in the same pixel but other sample, in consequence, it will not be able to generalize and the prediction accuracy will decay.

  To solve this, Pooling Layers are typically applied on the tensors representing the filter responses output by convolutional layers. They consist of a simple predefined function with no extra learnable parameters, that operates on a subset of the given input and generates a single value for that subset. By doing so, and as shown in Figure 2.10, we are able to reduce the dimensionality, providing three benefits: (1) as the feature map is reduced, a computational performance is gained; (2) the amount of parameters to be trained are also decreased; and (3), more importantly, and the reason that these technique has a great impact to the network, is that obfuscates the exact location where a featured triggered up, hence is harder for the network memorize the data, and it is forced to generalize the obtained information.

  Regarding the function to be applied, Max Pooling and Average Pooling are commonly used, where the first one extracts the maximum value from the subset, while the second one computes the average result out of it.

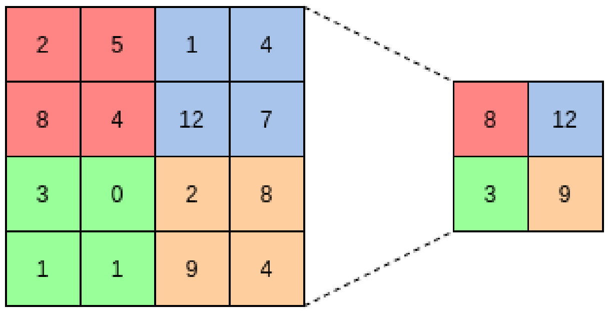


Figure 2.10: Reducing the data using Max Pooling.

- **Dense Layer**
  Is the most basic one, the Dense Layer, also known as Fully Connected Layer, has the particularity that the neurons that it contains receives every single output from the previous layer. A regular Artificial Neural Network as displayed in Figure 2.2 makes use of these layers as the main mechanism for feature extraction. CNN extracts local features using convolutions, however, is a common practice to apply a few (usually one or two) Dense Layers at the end of the network, this way, these layers can retrieve the most relevant features (extracted by the convolutions) independently of their spatial location in order to do the final classification.

2.4 Self-Attention

While an Artificial Neural Network tries to simulate the behavior of a biological brain, this technique focuses on a more concrete activity. Biological Attention is a cognitive process that discards irrelevant information and attends to more significant data. In this case, the attention mechanism seeks to generate relations between two inputs, in order to extract relevant dependencies, as can be seen on Figure 2.11(b). This has been widely applied in Neural Machine Translation task, where each input is a sentence from a different language.

However, this method can be applied with a single input, which is called Self-Attention. It aims to analyze the impact that holds each data with the rest, which is highly meaningful in image understanding [17, 18].

This mechanism consist on, given an input, generate three projections through learned linear transformations, which are named *Query*, *Key* and *Value*, then, in order to compute the attention, the first pair will be employed to obtain attention scores, which will determine the influence between each data from the input. Eventually, these scores are going to be used as attention weights for the *Value* projection, thus generating the final attentions. To do so, a dot product operation is applied between *Query* and *Key* projections, followed by a SoftMax activation, which will result in the aforementioned attention scores, and finally, one last dot product is applied between these scores and the *Value* projection, as the following equation describes (2.6):

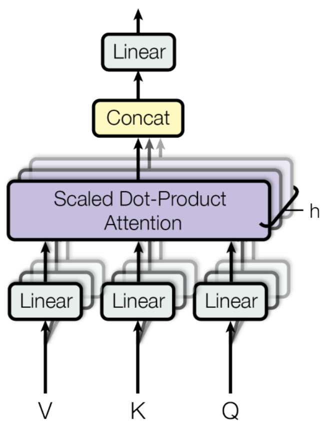
$$SelfAttention(In) = softmax(W_q(In) \cdot W_k(In)^\top) \cdot W_v(In) \tag{2.6}$$

Where W_q, W_k and W_v represents the linear transformations to generate *Query, Key* and *Value* projections respectively. Despite this, there are other ways to compute the attention such as scaled dot-product, where the partial result before the SoftMax is divided by the squared root of input's dimension size.

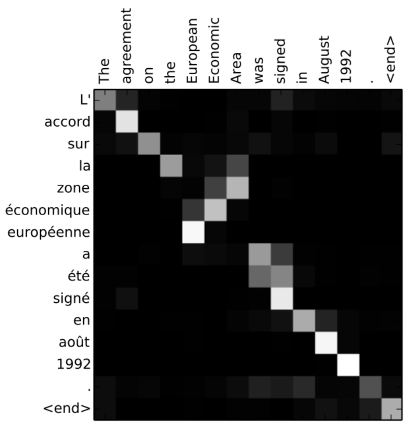
However, this operations are permutation invariant, which means that the self attention does not assume any temporal relation between the sequence, since each word of the same is fed simultaneously into the attention function. In an attempt to solve this, an encapsulated information about the position of each word is added to the input. This information, called Positional Encoding, makes it possible to the attention to be aware about the order of a sequence. The sinusoidal-based Positional Encoding is a well known function to represent the temporal order, it is computed as described in equation (2.7) where pos, f and D refers to sequence position, feature dimension and model dimension, respectively. However, there are many other functions to generate the Positional Encodings.

$$PE_{pos,2f} = sin\left(\frac{pos}{10000^{\frac{2f}{D}}}\right); PE_{pos,2f+1} = cos\left(\frac{pos}{10000^{\frac{2f}{D}}}\right) \tag{2.7}$$

The attention mechanism can also be computed in parallel, which is a strategy that may be beneficial to the performance. This technique receive the name of Multi-Head Attention, and it consists of splitting the input into h equal parts in the feature dimension, then, a learned projection for each part will be applied to generate *Query, Key* and *Value* sets. Each set of *Query, Key* and *Value*, will later be used as inputs for h independent attention functions (heads) to, eventually, concatenate the results and compute one last linear transformation to produce the final prediction, as shown in Figure 2.11(a).



(a)



(b)

Figure 2.11: (a) Scheme of a Multi-Head technique. Image from [15]; (b) Visualization of the attention scores. Image from [19]

Chapter 3

Related work

Some techniques used to tackle video understanding tasks are highly inspired in object detection methods [20, 21, 22, 23], since both of them involve deep frame analysis, moreover with the fact that many actions implies concrete object manipulations. Other approaches centralizes their efforts analyzing the differences between frames [24, 25], in order to capture the action's dynamic.

Two main workarounds can be distinguished here, classifying a trimmed video [26, 27] or localizing actions on untrimmed one [28, 29, 30]. In the recent years, researchers have focused on untrimmed videos, since they are a lot more suitable to real world applications. In this section we will introduce the most relevant contributions in two different tasks which are Action Detection and Action Segmentation, although this project is more focused in the second one, Action Detection has a great influence, since both of them have a similar goal.

3.1 Action Detection

This task not only aims to localize an action but also tries to predict the starting and ending frames. To achieve this there are two intuitive approaches, the first one would be predicting everything at once, which is quite challenging. The other possible solution would be a two-staged model, where the first stage would generate proposals, and the second one would refine and classify them. The research that has been made is more focused in the single stage approach.

Convolutional-De-Convolutional Networks (CDC) [31] adopts an encoder-decoder architecture. A C3D architecture [32], which is based in 3D ConvNets, is used to extract both temporal and spatial information, however, C3D model shrinks each dimension. The main goal of CDC is to predict a frame-level classes to generate a precises action localization so, in a way restore the temporal resolution, they use a transposed convolution (also known as Deconvolution) which is a technique to upsample the data in a learnable manner.

Boundary Sensitive Network (BSN) [33] uses a two stream model [24] as backbone to pre-extract spatio-temporal features, this backbone operates over a single RGB frame for the spatial stream, and a stacked optical flow frames for the temporal stream in order to explicitly capture motion information. Then, BSN employ each output from the backbone network as a frame embedding, concatenating them to further work in the temporal dimension by applying temporal convolutions, as a mean of learn low-range patterns.

Decoupled Single Shot Action Detection (Decoupled-SSAD) [29] has the same idea BSN, they extract the frame feature sequence employing a two stream network [24], and process the outputted data using temporal convolutions, but instead of doing it in a single stream, they divide the problem in two branches where each one will be specialized in a concrete task (i.e. proposal or classification), yet maintaining the main stream that will learn characteristics of both. The main idea of using this architecture is to allow the network learn concrete features for each subtask independently, while the main stream will provide information from the deeper layers to assist both branches by deconvolutional operations.

3.2 Action Segmentation

In contrast to the above task, Action Segmentation does no explicitly predict the initial and final frames of an action, here the main objective is to divide an untrimmed video into classified segments, where each action segment is highly dependent from its surroundings, so the network must have the capability to understand the video's underlying meaning. But the main difference concerning Action Detection, relies in how closely each class resembles to the other, which also makes this task a extremely challenging one.

Lea *et al.* [11] introduces two model approaches highly inspired in audio recognition works, where temporal convolutions are applied. Encoder-Decoder Temporal Convolutional Network (ED-TCN) is based in an encoder-decoder architecture as its names implies, here they use a fixed kernel size, while the temporal length is reduced by applying MaxPooling operations, in order to capture both short and long-range patterns, this kind of architectures also minimize the over-segmentation errors. Along with ED-TCN, a Dilated-TCN model is introduced, in this case, several dilated convolutions are applied, and the video's resolution is kept unchanged, dilated convolutions allow to capture long-range information without adding extra parameters.

Inspired on Pose Estimation task, Farha *et al.* [13] propose to stack several Dilated-TCNs to form a MultiStage-TCN (MS-TCN). They use a pre-extracted spatio-temporal features as input for the first stage, however, the following stages will receive probabilities without any additional information, in order to refine the final prediction. This methodology displays a significant enhancement over previous models.

Action Segment Refinement Framework (ASRF) [34] adapts MS-TCN architecture and incorporates two refinement branches. Similar to the approach made by the Decouple-SSAD of the previous section, each branch will be specialized in classify a frame or predict a segment boundary. Eventually the prediction of each branch will be merged to generate the final prediction. This method aims to minimize the over-segmentation errors and gets state-of-the-art results in this task.

3.3 Others

Symmetric Dilated Convolution (SD-TCN) [35] makes use of two TCNs with a bottleneck in between, in an encoder-decoder based architecture. On this bottleneck, they reduce the temporal resolution to pass it through a self-attention module, they assure that dilated convolutions only are capable of obtaining local neighbors relations, the objective of using this module is to generate frame-to-frame relationships in a non-local manner, which would assist in the final prediction.

Trans-SVNet [36] is build upon a two-staged TCN, but in this case, is used as an temporal feature extractor, which, at the same time, is fed with pre-extracted spatial features. Here a novel two-layer transformer method is proposed, which processes the data in a local manner. They claim that spatial features contain details that are lost when applying the temporal convolutions, in order to solve this, they employ the aforementioned transformer to aggregate both, spatial and temporal pre-extracted features, in short, they generating relations between those two features with a self-attention module to refine the predictions, while ensuring that the features are exploited as much as possible.

In contrast with these models, both applied in Surgical Phase Recognition, one of the modules proposed in this project consist of a transformer-based architecture, which could work over the whole video's resolution in a single step manner, with the purpose of generate global relationships (in SD-TCN the temporal length is shrinked, while in Trans-SVNet the transformer uses a convolutional method).

Pyramid Dilated Attention Network (PDAN) [37] have the same structure as a TCN, however, in this case the temporal convolutions are replaced with a novel dilated self-attention layer. They claim that a regular convolution gives the same importance to every frame on the kernel, meanwhile, the self-attention layer can extract better interactions between them, giving the proper relevance for each, especially when a large receptive field is applied.

We propose another module which is greatly inspired by PDAN, maintaining almost the same structure, but in this case it is attached to the output of a TCN to tackle Action Segmentation task for the first time, instead of Multi-Detection task, which is the one that PDAN is applied on. In addition, this module works with a larger kernel size, incorporates positional encoding, a head mechanism, and makes use of a residual feed-forward network, instead of the 1×1 convolutional bottleneck that is implemented in PDAN architecture.

Chapter 4

Methodology

In this project, we combine two different modules for temporal segmentation of actions in video, which are based on attention mechanisms applied across time. These modules will be built upon MS-TCN and ASRF arhictectures proposed by Farha *et al.* [13] and Ishikawa *et al.* [34], respectively, since both of them proved their video understanding capabilities by acquiring state-of-the-art results.

4.1 Proposed models

Given a sequence of per-frame representations $P = [p_1, \ldots, p_T] \in \mathbb{R}^{C \times T}$ outputted by a *Multi-Stage Temporal Convolutional Network* (MS-TCN), where T is the video sequence length (number of frames) and C the number of action classes, our main goal is refine those into $\hat{P} = [\hat{p}_1, \ldots, \hat{p}_T] \in \mathbb{R}^{C \times T}$ by identifying deep long-range inter-frame relationships.

The MS-TCN architecture is composed of S sequentially stacked single stages, each being a different Temporal Convolutional Network (TCN). A TCN consists of a given number of layers L_{TCN}, which we fixed to be the same across all stages. The first and last layers of each stage are 1D convolutional layers with kernel size of 1 (hereinafter we refer to this kind of convolution as 1×1 convolutional layer). The only difference between these two layers is that the first one projects the input features to the TCN dimension features D, whereas the last one projects them to C. The remaining, intermediate layers, are dilated convolutions with kernel size of 3 and D number of features for both, input and output sequences. Dilated convolutions have the particularity that the kernel is expanded by introducing fixed zeros between parameters, making it disregard the information in these zones where the zeros were applied, as it is displayed in Figure 4.1(b). Note that TCN layers apply the proper 0-padding to left and right sides of the sequence to guarantee the length of the input and output sequences remain unchanged.

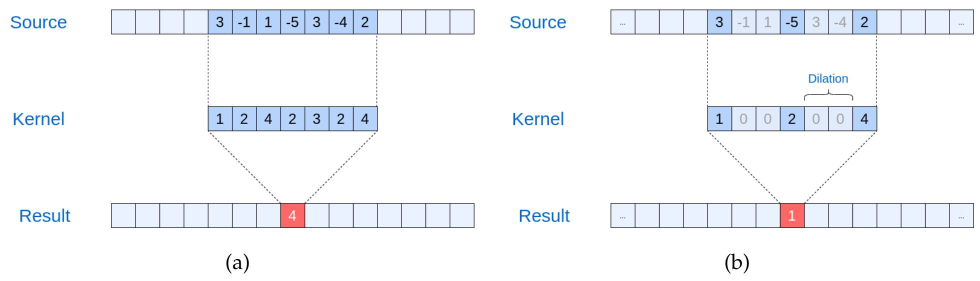


Figure 4.1: Examples of the different temporal convolutions: (a) Regular and (b) Dilated. For the sake of simplicity, each timestep of input and output sequences is represented by one single value (instead of C). In (a), the regular convolution densely multiplies each source position by its corresponding weight on the kernel and sums the weighted input values, i.e. $3 \cdot 1 + \ldots + 2 \cdot 4 = 4$. In (b), the operation is analogous, except for the more sparse (and, hence, faster) computation, leaving out greyed "Source"/"Kernel" positions by setting the dilated kernel positions to 0.

The dilation is able to provide to the temporal convolutions a larger receptive field, without increasing the number of learnable parameters. In particular, we double the dilation size as we go deeper in the TCN layers. Besides that, each dilated layer's output is linked with a residual connection to enhance the gradient calculation.

Note that each stage of the MS-TCN provides their own frame-level predictions that is then subsequently refined by the later stages. Precisely, the attention modules will be added after the very last stage prediction.

4.1.1 TCN with Transformer

The first goal of this project was the addition of a module based on [15], that could extract relations in a non-local manner, this way, a global relationship among frames can be built, acquiring frame-to-frame dependencies. A Transformer architecture can accomplish this objective by operating over the entire video, without increasing the number of parameters. Thus, a transformer module was added on top of a MS-TCN as shown in Figure 4.2

In this module we have a 1×1 convolutional layer, to project the dimensionlaity of the input features to D features. After that, a sinusoidal positional encoding is added to this new feature map as a mean to provide an order context to the frames. The result is used as input for the Multi-Head Attention layer followed by a point-wise feedforward layer, in order to process every self-attention's output individually. Both layers have a residual connection followed by a batch normalization. Eventually, another 1×1 convolutional layer is applied to reduce the feature map's dimensionality to the number of classes, thus yielding the final prediction by applying the SoftMax activation function.

In the self-attention part, the input is projected into *Query*, *Key* and *Value*, where the similarities between the first pair will be calculated and normalized, with a softmax function, to generate the attention scores, and use them as a weigthed matrix for the Value input 4.3.

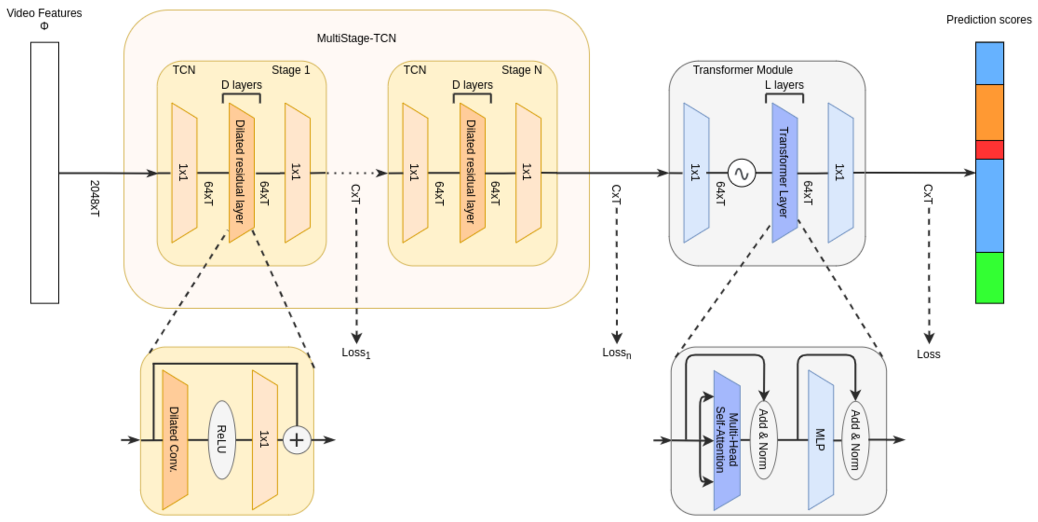


Figure 4.2: Multi-Stage TCN with Transformer module

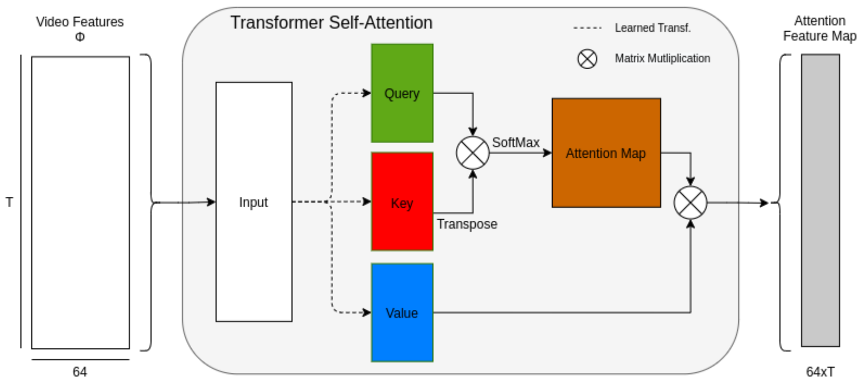


Figure 4.3: Transformer's attention layer behavior.

4.1.2 TCN with Convolutional Self-Attention

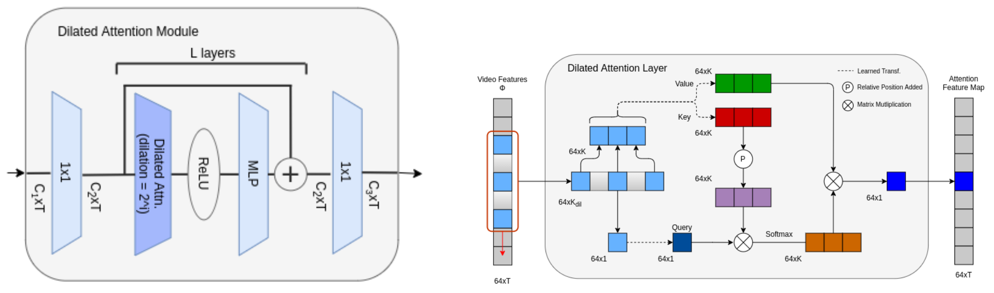
In this case, the structure of the attention module is almost the same as the one presented in the previous section. We have two 1×1 convolutional layers to change the number of features accordingly, with L attention layers in between as shown in Figure 4.4(a).

The common schedule of the transformer layer is replaced, instead, an adapted version of the Stand-Alone Self-Attention, presented in [38], is used. This new attention, can convolve over the temporal dimension. If an attention mechanism is applied to the whole video when this one has a large duration, distant frames are more likely to cause noise between them, due to the fact that their actions may be totally unrelated.

Stand-Alone Self-Attention applies relative positional encoding introduced in [39]. As we are applying the attention in a local manner, is hard for an absolute positional encoding to capture relative orders, so in this case, the addition of the positional information is added into the *Key* projection through a learnable vector. Sinusoidal Encoding uses an intuitive indexing solution, although not necessarily the most optimal. Parameterizing the positional encoding provides to the network the capability to decide how to understand the frame sequence order, since representing the position of something can be done in many ways.

Finally, the idea of dilating the convolutional attention, introduced in Pyramid Dilated Attention Networks [37], was implemented. This can highly improve the receptive field without increasing the number of parameters, also, with the compact kernel size, the noise that could be produced for irrelevant frames far away won't have such an impact in the results, due to the small samples that are taken. Figure 4.4(b) illustrates the behavior of the dilated convolutional attention.

The following formula is used in order to calculate the projection of the dilated kernel into the input vector: $k_{dil} = (d-1) * (k-1) + k$, where k_{dil} represents the resulting dilated kernel size, k is the actual kernel size, and d refers to the dilation rate.



(a) (b)

Figure 4.4: Convolutional Attention: (a) structure of the Convolutional Attention Stage; (b) behavior inside the Dilated Attention Layer

4.1.3 ASRF with Conv. Self-Attention

Action Segment Refinement Framework [34], is an architecture based on Multi-Stage TCN, where the main different is focused on the refinement stage, here, they introduce two branches of TCN, where each will specialize on prediction classification score or boundary probabilities, finally, the inference will be highly determined by the second branch. With the predicted boundaries outputted by this branch, a set of partial segment predictions will be defined, however, this segments are only specified by the boundaries so they do not have any class assigned yet. To do so, the classification branch is used, in this case we only look for the majority class predicted in the segments defined by the boundary branch, then, every other frame inside this segment that differs in class will be replaced for the predominant one. The idea behind this is to reduce the over-segmentation errors.

The attention module presented in the previous section is now added into the classification branch, in order to prove its positive effect over the final prediction.

Also, a study of the influence that the attention module has on each branch is made and presented in the results section.

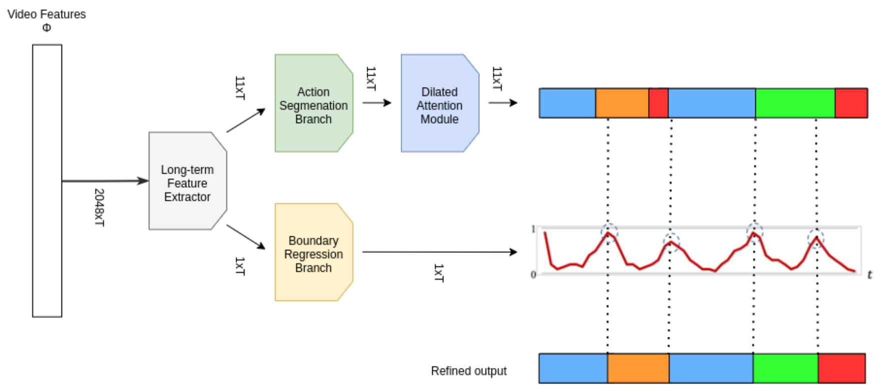


Figure 4.5: Convolutional Attention module applied on the classification branch of ASRF

Chapter 5

Results

In this section, we introduce the details of the dataset, the evaluation protocol used, and the relevant implementation details. Then we introduce all the ablation experiments for different method alternatives. And, finally, the results compared against the *state-of-the-art* scores.

5.1 Dataset

GTEA is a well-known and challenging dataset of fine-grained cooking actions [40]. The data is collected by a camera located on the actor's head, in order to acquire their egocentric point of view. There are 4 users, each performing 7 different activities, making a total of 28 unique videos. These videos are recorded at 15 frames per second, and their average length is about a minute long. The videos are labeled at frame level, each frame belonging to one of 11 different action categories, plus an extra one for background (no action). Since the number of instances for different categories, including background ones, are roughly the same, no investigation will be required to deal with the usual problem of learning from unbalanced data.

Instead of videos, we directly built on pre-extracted per-frame features already provided by Farha *et al.* [13]. These features were extracted utilizing the popular I3D model [26], which encodes the spatiotemporal information contained in the short video clip centered at each frame into a manageable 2,048 feature vector representation. Hence, given this more local spatiotemporal information, we can then entirely focus on modeling the more long range temporal dependencies among the frames, and also avoid the otherwise large computation requirements when dealing with frames directly. For a fair comparison between model alternatives, these same features will be used for all further experiments.

5.2 Evaluation Metrics

Temporal segmentation problems require of several evaluation metrics to assert the correct performance of proposed methods. Following the evaluation protocol proposed by [11], the evaluation metrics we report are:

- **Frame-wise accuracy**
  It represents the percentage of frames that were correctly labeled. Although the frame-level accuracy is widely used in segmentation tasks, it can not precisely capture over-segmentation errors. Note that if, during test, a groundtruth action segment is predicted as several action segments of this same class, but separated by some spurious missclassified frames between them, this would not be penalized.

- **Edit distance**
  This metric emphasizes the prediction order rather than precise temporal segmentation. Therefore, it is a representative measure that can provide information about the amount of over-segmentation errors that a network is generating. It is based on the Levenshtein algorithm, which counts the number of deletions, insertions and substitutions needed to match any two arbitrary finite strings; in our case, the string symbols representing action class labels. Because the algorithm returns a counting, we normalize, subtract it from 1, and convert it to a perceptual value. Given the groundtruth sequence of action classes G and the predicted sequence of action classes P:

  $$\text{Edit}(G, P) = \left(1 - \frac{\text{lev}(G, P)}{\max(|G|, |P|)}\right) * 100 \tag{5.1}$$

  where $|\cdot|$ is the length of a string/sequence. The Edit distance reported is the average among all the test sequences in the dataset.

- **Segmental F1 score**
  Is a common practice to use mean Average Precision (mAP) as a main metric in Action Detection task, however, Lea *et al.* [11] claimed that is not that efficient in fine-grained segmentation, since it is sensitive to the predicted confidence score that is assigned to each segment. To that end, they proposed F1 score as the segmental metric:

  $$F1 = 2 * \frac{precision * recall}{precision + recall}. \tag{5.2}$$

  Finally, a threshold is determined to decide if the corresponding prediction must be interpreted as a positive.

5.3 Implementation Details

Baselines In this project, four methodologies are adapted, where two of them (MS-TCN [13] and ASRF [34]), will be used as baseline architectures, and the other two (Transformer [15] and PDAN [37]), will be applied on top as final predictor modules. For the implementation of MS-TCN and ASRF, we based on the code from `http://github.com/yiskw713/asrf`.

Parameters For both architectures, MS-TCN and ASRF, the number of layers and the kernel sizes will be fixed to, respectively, $L_{\text{TCN}} = 12$ (i.e. 10 dilated temporal convolution layers) and 3. However, with the addition of the transformer and dilated self-attention modules, an analysis of the impact produced by the number of stages will be performed. The number of layers for each module will be reviewed as well. The models have a total of 64 filters for each layer.

Training The models were trained using four-fold leave-one-subject-out validation. For each testing fold, videos of the corresponding actor are used as testing set and the other three as training set. The parameters were learned by the ADAM optimizer [41] with a learning rate of 5e-4 and a batch size of 1, which was equivalent to an entire video per batch. Adam optimizer minimized Categorical Cross Entropy loss and/or Gaussian similarity-weighted loss (introduced in [34]). In the case of ASRF architecture, the Binary Logistic Regression loss function was introduced in order to train the boundary refinement branch.

5.4 Adding Transformer Module

The results of attaching a transformer module, at the output of a MS-TCN, will be presented in this section. Please note that every experiment is made upon a SingeStage-TCN unless otherwise specified.

5.4.1 Effect of the feature dimension

The first hyper-parameter that must be established is the number of features per frame, with which the transformer will be operating. The output of a SingleStage-TCN is the class score probability distribution over the C action classes, which leads into a feature map with dimensions $C \times T$, where T corresponds to the video's length and $C = 11$ to the number of classes. However, this same architecture internally uses $D_{\text{TCN}} = 64$ to work in each layer. So the first experiment will be focused on the selection of this two possible configurations for the transformer.

	F1@{10,25,50}			Edit	Acc
$D_{\text{Transf}} = 11$	46.13	43.69	**35.42**	37.17	**73.26**
$D_{\text{Transf}} = 64$	**48.40**	**43.79**	34.74	**38.26**	69.81

Table 5.1: Impact of the feature dimension (D_{Transf}) that the transformer works with.

Table 5.1 shows that, the usage of a higher feature dimensionality, performs slightly better, compared with just making use of the number of classes. Also, note the accuracy gets a better score when $D_{\text{Transf}} = C = 11$, while $D_{\text{Transf}} = 64$ outperforms almost every other metric. These results support the intuition that the transformer benefit from an increased number of features to internally model richer inter-frame relationships. We therefore fixed to $D_{\text{Transf}} = D_{\text{TCN}} = 64$ for the internal feature representation size of the transformer architecture.

5.4.2 Adding a Positional Encoding

Transformer layers compute a new representation for each input frame embedding by attending to all positions of the input without inherently relating temporal information. To that end, a Sinusoidal Encoding is added to the transformer's input[15]. With the addition of the positional encoding, the transformer is able to take into account the temporal order for computing frame interrelations. Table 5.2 displays its positive effect by improving every metric.

	F1@{10,25,50}			Edit	Acc
w/o PE	48.40	43.79	34.74	38.26	69.81
with PE	**52.32**	**48.68**	**38.26**	**44.10**	**70.07**

Table 5.2: Comparison between using positional encoding (PE) or not

5.4.3 Effect of the number of attention heads

Having several attention heads in each transformer layer is a strategy that makes it capable of learning different and complementary semantics in the output sequence [15]. Each head attends over a subspace of the input features $D_{\text{head}} = D_{\text{Transf}}/H$, where H is the number of attention heads. Then the outputs of the different heads are concatenated and passed through a feed-forward network.

	F1@{10,25,50}			Edit	Acc
H=1	52.32	**48.48**	38.26	**44.10**	70.07
H=2	46.63	42.91	35.01	41.26	71.36
H=4	48.94	44.89	35.54	40.29	70.39
H=8	**52.72**	48.57	**39.15**	43.44	**72.16**
H=16	45.07	41.97	32.50	34.99	70.72

Table 5.3: Effect of the number of heads (H)

Results displayed in Table 5.3 show that different choices of H do not have a significant impact to the final performance. This can be caused by the fairly low input embedding dimensionality ($D_{\text{Transf}} = 64$ against 512 from the original work [15]), which causes each head to learn in very small subspace. We ended up selecting $H = 8$, which is the one with best results for the most challenging metric (F1@50), for the following experiments.

5.4.4 Effect of the number of layers

This experiment sequentially stack additional transformer layers to learn more complex and abstract temporal semantics. However, as Table 5.4 shows, the performance substantially drops. This decrease of performance indicates that transformers with more than 1 layer may face problems to correctly understand GTEA semantics. Transformers are known of being data hungry and GTEA is a relatively small dataset. On the other hand, GTEA sequences are fairly long. Dai *et al.* [37] commented that very large receptive field may cause some noise to the attention scores, due to unrelated actions that may occur far away from a concrete frame.

Considering that the transformer is commonly applied in Neural Machine Translation task (NLP), where sentences have an average length of 30 words (in the case of WMT-17 dataset [42]), this could be a possible issue, since GTEA dataset has an average length of 1,000 frames per video. Transformer is trying to build frame-to-frame relations over the whole video's resolution, although perhaps the most relevant relationships are more localized in the nearer frames, while the frames far apart may be just introducing the aforementioned noise, that prevents the model to learn properly.

	F1@{10,25,50}			Edit	Acc
L=1	**52.32**	**48.48**	**38.26**	**44.10**	**70.07**
L=2	46.55	42.09	32.42	42.44	60.45

Table 5.4: Effect of the number of layers (L)

5.5 Adding Conv. Self-Attention Module

Based on the analysis of the results obtained by the Tranformer module, another approach is implemented, in order to tackle the issue that is allegedly causing the learning problems. Stand-Alone Self-Attention, introduced in [38], proposed a convolutional version of the attention mechanism. They claimed that this technique could replace the regular spatial convolutions while reducing the number of parameters.

Here, the transformer module is replaced by an adapted version of the Stand-Alone Self-Attention method, that can work in the temporal domain. The objective of this modification is the reduction of the attention's receptive field, which could minimize the prediction's noise. As performed in the previous section, a set of experiments have been accomplished and reported below, in order to select the correct hyperparameters.

5.5.1 Effect of the kernel size

The kernel size K will determine the receptive field that the self-attention module will be operating with, a small kernel size may not be adequate to capture the frame sequence context, but at the same time, a kernel size way to large may cause noise issues, so to begin with, a set of kernel sizes ranging between 7 and 15 weights were tested. The latter would represent up to a second long receptive field, since the videos have a frame rate of 15fps.

	F1@{10,25,50}			Edit	Acc
K=7	79.04	74.82	59.69	71.97	74.09
K=9	79.52	75.27	62.66	73.82	74.36
K=11	**80.33**	**76.97**	**62.82**	72.76	**75.48**
K=13	77.41	72.86	61.28	70.47	72.96
K=15	80.20	75.13	61.41	**72.98**	75.44

Table 5.5: Effect of the kernel size (K)

Although the results in Table 5.5 do not overly vary between different kernels, a slightly general improvement is visible when $K = 11$. The new proposed module achieves significantly better score results, compared to the transformer previously presented, even acquiring an absolute improvement of at least 24% in every segmental metric, as can be seen in Table 5.6. This results prove that the convolutional version is highly beneficial, and the assumption that the low results of the transformer module could be caused by its larger receptive filed is more conceivable. Therefore, the followings experiments will be realized with a fixed kernel size of 11.

Module	F1@{10,25,50}			Edit	Acc
Transformer ($L = 1$)	52.32	48.48	38.26	44.10	70.07
Conv. Self-Attn ($L = 1$,$K = 11$)	**80.33**	**76.97**	**62.82**	**72.76**	**75.48**

Table 5.6: Comparison between Transformer and Convolutional Self-Attention layers

5.5.2 Effect of the number of heads

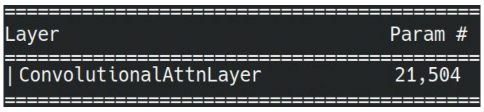
In a similar way to Multi-Head Attention technique, Stand-Alone Self-Attention has some head-like mechanism, in order to learn multiple distinct representations of the input. The heads in original Transformers are mimicked by grouped convolutional layers, which essentially followed the same strategy as described before: splitting the input's feature dimension in G segments, and grouping up the convolutional filters in also G different clusters, then, each cluster will perform the convolution operation over its assigned feature section. This methodology was firstly designed to improve the parallelization in multiple GPUs, but it provides additional benefits, like the one mention above (learn different representations of the input), or the reduction of parameters by having shared weights between groups.

For the original Transformer module, the analogous parameter H was set to 8. However, in order to ensure that this value is the best choice also for this newer model, we tested different options and presented them in Table 5.7. The obtained scores shows that this configurations actually acquire the higher results.

	F1@{10,25,50}			Edit	Acc
$G = 1$	73.45	71.09	58.88	66.86	**75.63**
$G = 2$	75.10	71.38	57.76	67.79	74.76
$G = 4$	78.35	74.47	60.14	70.10	74.32
$G = 8$	**80.33**	**76.97**	**62.82**	**72.76**	75.48
$G = 16$	78.40	74.82	61.39	72.73	74.68

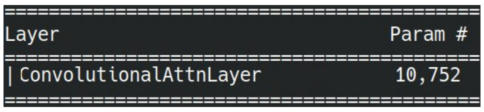
Table 5.7: Comparison choosing different number of groups (G. The number of layers and kernel sizes are $L = 1$ and $K = 11$ respectively

In addition, it is worth to emphasize the improvement over using a single group, since using more groups also reduces the number of learnable parameters, in this case, it is practically reduced by half, as can be seen in Figure 5.1.



```
==========================================
Layer                            Param #
==========================================
|ConvolutionalAttnLayer           21,504
==========================================
```

(a) $G = 1$



```
==========================================
Layer                            Param #
==========================================
|ConvolutionalAttnLayer           10,752
==========================================
```

(b) $G = 8$

Figure 5.1: Impact of that the number of groups have into the amount of parameters: (a) Convolutional Self-Attention with a single group; (b) Convolutional Self-Attention with a 8 groups.

5.5.3 Effect of the number of layers

Finally, we evaluate the best number of layers to include in the convolutional self-attention module. This will manifest whether an increment of the module's depth can provide enhanced results or it will yield into a worst prediction, as it did in the case of the transformer. There are two possible methodologies to work through in this section, the first one would be using regular convolutional attention layers, where each layer have the same receptive field, whereas the other would consist in including a dilated kernel in the convolutional self-attention layer, as in TCN.

5.5.3.1 Regular layers

In order to increase the receptive field when the regular convolutional layers are being used, the kernel size must be enlarged, which is unproductive for the self-attention module, as it was displayed in Table 5.5. On the other hand, keeping a small kernel makes unfeasible the learning of long-range dependencies, which is the main goal of TCN models. The results presented in the Table 5.8, shows how increasing this module's depth can actually help to achieve better results. Therefore, the reduced number of parameters in conv. self-attention w.r.t. Transformer's self-attention makes the training on smaller datasets such as GTEA easier. Besides, the long-range dependencies can be more easily captured by such hierarchy of convolutional self-attention layers, where deeper layers with increased kernel size will capture longer range dependencies from the more local information provided by more shallow layers.

	F1@{10,25,50}			Edit	Acc
$L = 1$	80.33	76.97	62.82	72.76	**75.48**
$L = 2$	80.51	76.99	63.69	74.53	73.27
$L = 3$	**82.09**	**79.93**	**66.36**	**76.81**	74.40

Table 5.8: Comparison of results among different numbers of convolutional self-attention layers L using standard convolutions

5.5.3.2 Dilated layers

Since the experiment made in the previous section claims that regular convolutional self-attention layers can bring promising scores, applying a dilated attention may lead into an improvement on the final predictions. As we already saw, enlarging the kernel may cause noise issues. However, long-range dependencies could provide relevant information, so in order to acquire this, a dilated version of the convolutional self-attention has been implemented, by inserting zeros in proper kernel positions, in order to expand its receptive field. Doing this, the attention module should be able to filter irrelevant frames without causing the aforementioned noise, since the number of samples that are taken is actually small (11 frames in this case), with the advantage that a greater receptive field can be applied. So, as done with the standard convolutional attention layers, some more depth levels were added, and the results displayed in Table 5.9, shows that this technique can also be beneficial for the performance.

	F1@{10,25,50}			Edit	Acc
$L = 1$	80.33	76.97	62.82	72.76	**75.48**
$L = 2$	81.56	78.64	64.40	75.89	74.89
$L = 3$	**82.99**	**80.75**	**67.14**	**77.71**	73.95

Table 5.9: Comparison of results among different numbers of convolutional self-attention layers L using dilated convolutions

Implementing the dilated kernels in attention layers was firstly proposed by Dai *et al.* in [37], which acquires state-of-the-art results in Multi-Action Detection task. Nonetheless, this is the first time that a convolutional attention module is applied in Action Segmentation (both versions, regular and dilated one). Finally, as the SS-TCN architecture does, the dilation factor is doubled at each layer.

5.5.3.3 Comparison between strategies

Both methodologies acquire great scores in every metric, however, dilated version achieve slightly better results as can be seen in Table 5.10. This can be attributed to the higher understanding of the context, due to the larger receptive field that this concrete module owns, for this reason, the dilated configuration will be used for the rest of the experiments.

	Standard Attn. Convolution					Dilated Attn. Convolution				
	F1@{10,25,50}			Edit	Acc	F1@{10,25,50}			Edit	Acc
$L = 1$	80.33	76.97	62.82	72.76	**75.48**	-	-	-	-	-
$L = 2$	80.51	76.99	63.69	74.53	73.27	81.56	78.64	64.40	75.89	74.89
$L = 3$	82.09	79.93	66.36	76.81	74.40	**82.99**	**80.75**	**67.14**	**77.71**	73.95

Table 5.10: Obtained scores for each tested strategies when the module depth is increased

5.5.4 Effect of the number of stages

Until now, every evaluation was carried out attaching the proposed modules at the output of a single stage TCN, in order to evaluate them in a more controlled environment. Yet the real potential of a TCN takes place when several stages are stacked one after the other in a MultiStage-TCN. This architecture relies in the refinement of the predictions as they pass through stages. Thus, in this section, we added multiple stages in the baseline architecture.

As the Table 5.11 displays, the addition of 1 more stage, boosts the final predictions and reduces the over-segmentation errors, as can be appreciated by the Edit score, however, incorporating several stages to the model, although it keeps improving the result, does not seem to have any relevant impact.

	F1@{10,25,50}			Edit	Acc
$S = 1$	82.99	80.75	67.14	77.71	73.95
$S = 2$	**86.04**	83.66	70.27	81.72	76.34
$S = 3$	85.16	81.29	64.15	80.18	74.17
$S = 4$	86.04	**84.10**	**70.57**	**82.41**	**76.58**

Table 5.11: Results with different number of stages (S) in the baseline network MS-TCN where $S = 1$ refers to a single stage TCN

5.6 Adding Conv. Self-Attention to ASRF

With the dilated attention module analyzed in MS-TCN, one last experiment is performed using Action Segment Refinement Network (ASRF) as the baseline network predictor. ASRF is an architecture which has almost the same structure as the MS-TCN, the main difference relying in the refinement phase. Here, instead of implementing a unique stream, which performs a per frame classification, an additional branch is added with the purpose of generating a segment boundary regression. This way, each branch can be specialized in a concrete task, achieving the actual state-of-the-art results.

The experiment consists of determining in which branch the dilated attention module can perform better, to that end, three different evaluations have been accomplished, and the results are shown in Table 5.12. Attaching the proposed module only into the classification branch, provided the best results. In contrast, if the same module is only added in the regression branch, the performance drops significantly, this could be due to the fact that the classification branch has a richer feature representation, since it outputs 11 class probabilities for each frame, while the boundary regression branch only outputs a singleton prediction per frame.

Branch	F1@{10,25,50}			Edit	Acc
Classification	**82.95**	**81.31**	**71.23**	76.42	**74.50**
Boundary	81.10	76.36	63.00	75.32	72.37
Both	82.56	79.73	67.11	**76.85**	73.21

Table 5.12: Scores obtained by attaching the dilated attention module on each branch

On the other hand, the model also acquires great results when the dilated attention module is added in both branches at the same time, however, only applying this module into the classification branch keeps getting the best scores, which may be caused by the same reason, the poorer context information that provides the boundary branch, probably makes it more difficult for the model to correctly learn to extract frame relationships.

Taking these results into account, the dilated attention module is finally applied only into the classification branch. For the last experiment, as it was made with MS-TCN architecture, an evaluation of the effect that implies the addition of different number of ASRF's stages is presented in Table 5.13. In contrast with Table 5.11, no boost can be seen in the prediction score when the second stage is added, although it seems that the performance keeps increasing along with the number of stages. Note the addition of the stages are implemented equally for each branch, which means that if there are 3 stages, the first one would be the initial predictor stage, and there will be 2 more refinement stages for each branch.

	F1@{10,25,50}			Edit	Acc
$S = 1$	82.95	81.31	71.23	76.42	74.50
$S = 2$	83.42	81.98	69.15	**78.61**	75.16
$S = 3$	**86.09**	**85.06**	**74.58**	78.45	75.91
$S = 4$	85.48	84.07	71.43	78.28	**76.57**

Table 5.13: Results with different number of stages (S) in the baseline network ASRF, where $S = 1$ refers to a single stage TCN.

5.7 Comparison with the State-of-the-Art

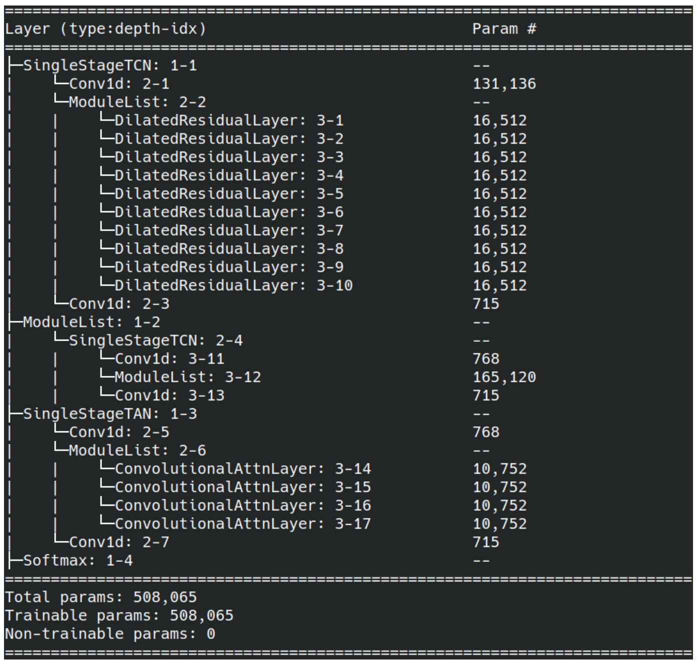
The proposed modules are built upon the state-of-the-art architectures, therefore, in this section will be presented the differences between applying them or not. This comparison is carried out on Georgia Tech Egocentric Activities dataset (GTEA). In addition, an evaluation of the baseline models is also performed, since Farha *et al.* [13] and Ishikawa *et al.* [34] do not provide results for different stages on GTEA dataset; moreover, ASRF used its own pre-extracted features, so the comparison would be unfair.

As can be seen in Table 5.14, attaching a transformer at the output of a MS-TCN, leads into a disadvantageous rather than beneficial behaviour, since the performance is greatly decreased. However, the dilated attention module can achieve competitive results in both architectures. Specifically, when this module is used upon a two-staged TCN, can obtain equivalent results compared to a MS-TCN with four stages. This is highly meaningful considering the fact that the first configuration has almost 300k less parameters, as it is displayed in Figure 5.2.

Finally, the addition of the dilated attention module also improves the results when it is applied right after the classification branch of ASRF, regardless of the number of stages. This certifies that the proposed method has the capability to extract relevant dependencies between frames.

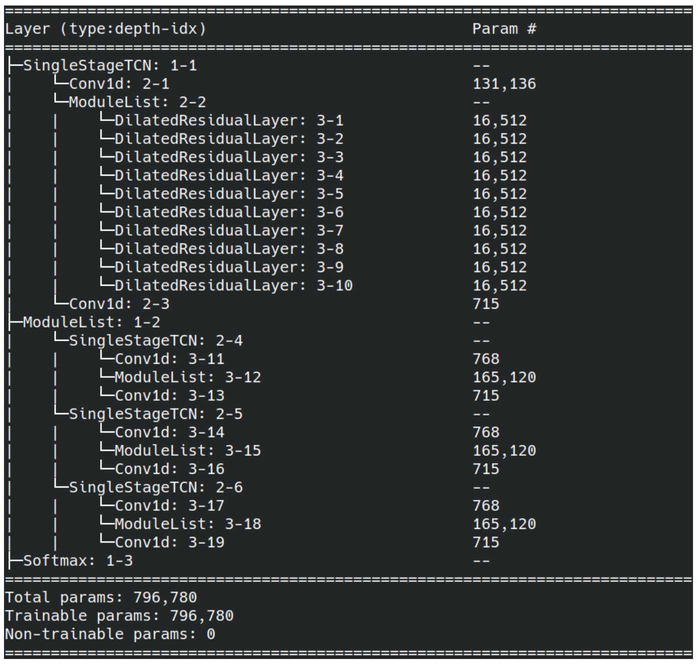
Model	Stages	F1@{10,25,50}			Edit	Acc
MS-TCN [13]	$S=1$	60.97	57.20	47.09	52.32	74.32
	$S=2$	80.53	77.53	64.08	76.51	74.31
	$S=3$	85.30	82.45	70.36	80.38	75.70
	$S=4$	**86.63**	84.15	69.44	82.00	74.91
ASRF [34]	$S=1$	84.36	81.26	67.86	79.09	74.34
	$S=2$	82.67	80.10	68.78	75.55	73.09
	$S=3$	84.60	83.27	72.22	76.35	75.04
	$S=4$	85.40	83.95	72.96	77.84	74.62
MS-TCN+Transf	$S=1$	52.32	48.48	38.26	44.10	70.07
MS-TCN+DAM	S=1	82.99	80.75	67.14	77.71	73.95
	$S=2$	86.04	83.66	70.27	81.72	76.34
	$S=3$	85.16	81.29	64.15	80.18	74.17
	$S=4$	86.04	84.10	70.57	**82.41**	**76.58**
ASRF+DAM	$S=1$	82.95	81.31	71.23	76.42	74.50
	$S=2$	83.42	81.98	69.15	78.61	75.16
	$S=3$	86.09	**85.06**	**74.58**	78.45	75.91
	$S=4$	85.48	84.07	71.43	78.28	76.57

Table 5.14: Comparison with the state-of-the-art on GTEA dataset where DAM refers to Dilated Attention Module



```
===========================================================================
Layer (type:depth-idx)                             Param #
===========================================================================
├─SingleStageTCN: 1-1                              --
│    └─Conv1d: 2-1                                 131,136
│    └─ModuleList: 2-2                             --
│    │    └─DilatedResidualLayer: 3-1              16,512
│    │    └─DilatedResidualLayer: 3-2              16,512
│    │    └─DilatedResidualLayer: 3-3              16,512
│    │    └─DilatedResidualLayer: 3-4              16,512
│    │    └─DilatedResidualLayer: 3-5              16,512
│    │    └─DilatedResidualLayer: 3-6              16,512
│    │    └─DilatedResidualLayer: 3-7              16,512
│    │    └─DilatedResidualLayer: 3-8              16,512
│    │    └─DilatedResidualLayer: 3-9              16,512
│    │    └─DilatedResidualLayer: 3-10             16,512
│    └─Conv1d: 2-3                                 715
├─ModuleList: 1-2                                  --
│    └─SingleStageTCN: 2-4                         --
│    │    └─Conv1d: 3-11                           768
│    │    └─ModuleList: 3-12                       165,120
│    │    └─Conv1d: 3-13                           715
├─SingleStageTAN: 1-3                              --
│    └─Conv1d: 2-5                                 768
│    └─ModuleList: 2-6                             --
│    │    └─ConvolutionalAttnLayer: 3-14           10,752
│    │    └─ConvolutionalAttnLayer: 3-15           10,752
│    │    └─ConvolutionalAttnLayer: 3-16           10,752
│    │    └─ConvolutionalAttnLayer: 3-17           10,752
│    └─Conv1d: 2-7                                 715
├─Softmax: 1-4                                     --
===========================================================================
Total params: 508,065
Trainable params: 508,065
Non-trainable params: 0
===========================================================================
```

(a) Structure and parameters of a Two-stage TCN with the Dilated Attn. Module added.



```
===========================================================================
Layer (type:depth-idx)                             Param #
===========================================================================
├─SingleStageTCN: 1-1                              --
│    └─Conv1d: 2-1                                 131,136
│    └─ModuleList: 2-2                             --
│    │    └─DilatedResidualLayer: 3-1              16,512
│    │    └─DilatedResidualLayer: 3-2              16,512
│    │    └─DilatedResidualLayer: 3-3              16,512
│    │    └─DilatedResidualLayer: 3-4              16,512
│    │    └─DilatedResidualLayer: 3-5              16,512
│    │    └─DilatedResidualLayer: 3-6              16,512
│    │    └─DilatedResidualLayer: 3-7              16,512
│    │    └─DilatedResidualLayer: 3-8              16,512
│    │    └─DilatedResidualLayer: 3-9              16,512
│    │    └─DilatedResidualLayer: 3-10             16,512
│    └─Conv1d: 2-3                                 715
├─ModuleList: 1-2                                  --
│    └─SingleStageTCN: 2-4                         --
│    │    └─Conv1d: 3-11                           768
│    │    └─ModuleList: 3-12                       165,120
│    │    └─Conv1d: 3-13                           715
│    └─SingleStageTCN: 2-5                         --
│    │    └─Conv1d: 3-14                           768
│    │    └─ModuleList: 3-15                       165,120
│    │    └─Conv1d: 3-16                           715
│    └─SingleStageTCN: 2-6                         --
│    │    └─Conv1d: 3-17                           768
│    │    └─ModuleList: 3-18                       165,120
│    │    └─Conv1d: 3-19                           715
├─Softmax: 1-3                                     --
===========================================================================
Total params: 796,780
Trainable params: 796,780
Non-trainable params: 0
===========================================================================
```

(b) Structure and parameters of a Four-stage TCN.

Figure 5.2: Comparison between architectures.

Chapter 6

Conclusions

The main objective of this project was to tackle the problem of temporal segmentation of actions observed in video data. For that, we first analyzed two different modules when they were applied into state-of-the-art architectures (MS-TCN [13] and ASRF [34]) to, eventually, compare the results and determine their effectiveness on the segmentation tasks. The first implemented module consisted of a transformer-based architecture which was prepared to handle the full video's resolution in a non-local manner. The purpose of this first approach, was to relate every single frame of a video with the rest. However, the obtained results were unsatisfactory, most likely due to the large receptive field and, consequently, a large amount inherent noise introduced.

Then, a convolutional self-attention module was introduced with essentially the same idea of TCNs: a dilated attention was implemented [37], and it allegedly helped to potentially minimize the noise by reducing the number of effective samples in larger receptive field. In contrast with the Transformer, the experiments made over the dilated attention layers revealed promising results, showing how the addition of a dilated convolutional self-attention on top of a MS-TCN provides to the model the capabilities to capture both, temporal patterns (TCN) and temporal relations (Attention).

As future work, we plan to carry out a more exhaustive evaluation of the proposed architectures in other datasets. Doing so, we expect to gather more insights about their effectiveness and, in particular, their ability to generalize over video data of different nature and with varying degrees of complexity in terms of temporal semantics.

Bibliography

[1] B. Singh, T. K. Marks, M. Jones, O. Tuzel, and M. Shao, "A multi-stream bi-directional recurrent neural network for fine-grained action detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1961–1970, 2016.

[2] Y. Gao, S. S. Vedula, C. E. Reiley, N. Ahmidi, B. Varadarajan, H. C. Lin, L. Tao, L. Zappella, B. Béjar, D. D. Yuh, *et al.*, "Jhu-isi gesture and skill assessment working set (jigsaws): A surgical activity dataset for human motion modeling," in *MICCAI workshop: M2cai*, vol. 3, p. 3, 2014.

[3] A. P. Twinanda, S. Shehata, D. Mutter, J. Marescaux, M. De Mathelin, and N. Padoy, "Endonet: a deep architecture for recognition tasks on laparoscopic videos," *IEEE transactions on medical imaging*, vol. 36, no. 1, pp. 86–97, 2016.

[4] O. Zisimopoulos, E. Flouty, I. Luengo, P. Giataganas, J. Nehme, A. Chow, and D. Stoyanov, "Deepphase: surgical phase recognition in cataracts videos," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 265–272, Springer, 2018.

[5] S. Stein and S. 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*, pp. 729–738, 2013.

[6] H. Kuehne, A. Arslan, and T. Serre, "The language of actions: Recovering the syntax and semantics of goal-directed human activities," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 780–787, 2014.

[7] Y. Li, Z. Ye, and J. M. Rehg, "Delving into egocentric actions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 287–295, 2015.

[8] D. Shao, Y. Zhao, B. Dai, and D. Lin, "Finegym: A hierarchical video dataset for fine-grained action understanding," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2616–2625, 2020.

[9] J. Jones, G. D. Hager, and S. Khudanpur, "Toward computer vision systems that understand real-world assembly processes," in *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 426–434, IEEE, 2019.

[10] Y. Ben-Shabat, X. Yu, F. Saleh, D. Campbell, C. Rodriguez-Opazo, H. Li, and S. Gould, "The ikea asm dataset: Understanding people assembling furniture through actions, objects and pose," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 847–859, 2021.

[11] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager, "Temporal convolutional networks for action segmentation and detection," in *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 156–165, 2017.

[12] P. Lei and S. Todorovic, "Temporal deformable residual networks for action segmentation in videos," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6742–6751, 2018.

[13] Y. A. Farha and J. Gall, "Ms-tcn: Multi-stage temporal convolutional network for action segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3575–3584, 2019.

[14] Z. Wang, Z. Gao, L. Wang, Z. Li, and G. Wu, "Boundary-aware cascade networks for temporal action segmentation," in *European Conference on Computer Vision*, pp. 34–51, Springer, 2020.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[16] B. Wicht, *Deep Learning feature Extraction for Image Processing*. PhD thesis, 01 2018.

[17] K. Chen, J. Wang, L.-C. Chen, H. Gao, W. Xu, and R. Nevatia, "Abc-cnn: An attention based convolutional neural network for visual question answering," *arXiv preprint arXiv:1511.05960*, 2015.

[18] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *International conference on machine learning*, pp. 2048–2057, PMLR, 2015.

[19] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[20] A. Gupta and L. S. Davis, "Objects in action: An approach for combining action understanding and object perception," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, IEEE, 2007.

[21] A. Fathi, A. Farhadi, and J. M. Rehg, "Understanding egocentric activities," in *2011 international conference on computer vision*, pp. 407–414, IEEE, 2011.

[22] C. Lea, A. Reiter, R. Vidal, and G. D. Hager, "Segmental spatiotemporal cnns for fine-grained action segmentation," in *European Conference on Computer Vision*, pp. 36–52, Springer, 2016.

[23] H. Xu, L. Yang, S. Sclaroff, K. Saenko, and T. Darrell, "Spatio-temporal action detection with multi-object interaction," *arXiv preprint arXiv:2004.00180*, 2020.

[24] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," *arXiv preprint arXiv:1406.2199*, 2014.

[25] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. Van Gool, "Temporal segment networks: Towards good practices for deep action recognition," in *European conference on computer vision*, pp. 20–36, Springer, 2016.

[26] J. Carreira and A. Zisserman, "Quo vadis, action recognition? a new model and the kinetics dataset," in *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6299–6308, 2017.

[27] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, and M. Paluri, "A closer look at spatiotemporal convolutions for action recognition," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 6450–6459, 2018.

[28] Y.-W. Chao, S. Vijayanarasimhan, B. Seybold, D. A. Ross, J. Deng, and R. Sukthankar, "Rethinking the faster r-cnn architecture for temporal action localization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1130–1139, 2018.

[29] Y. Huang, Q. Dai, and Y. Lu, "Decoupling localization and classification in single shot temporal action detection," in *2019 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1288–1293, IEEE, 2019.

[30] M. Xu, C. Zhao, D. S. Rojas, A. Thabet, and B. Ghanem, "G-tad: Sub-graph localization for temporal action detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10156–10165, 2020.

[31] Z. Shou, J. Chan, A. Zareian, K. Miyazawa, and S.-F. Chang, "Cdc: Convolutional-de-convolutional networks for precise temporal action localization in untrimmed videos," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5734–5743, 2017.

[32] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, pp. 4489–4497, 2015.

[33] T. Lin, X. Zhao, H. Su, C. Wang, and M. Yang, "Bsn: Boundary sensitive network for temporal action proposal generation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 3–19, 2018.

[34] Y. Ishikawa, S. Kasai, Y. Aoki, and H. Kataoka, "Alleviating over-segmentation errors by detecting action boundaries," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 2322–2331, 2021.

[35] J. Zhang, Y. Nie, Y. Lyu, H. Li, J. Chang, X. Yang, and J. J. Zhang, "Symmetric dilated convolution for surgical gesture recognition," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 409–418, Springer, 2020.

[36] X. Gao, Y. Jin, Y. Long, Q. Dou, and P.-A. Heng, "Trans-svnet: Accurate phase recognition from surgical videos via hybrid embedding aggregation transformer," *arXiv preprint arXiv:2103.09712*, 2021.

[37] R. Dai, S. Das, L. Minciullo, L. Garattoni, G. Francesca, and F. Bremond, "Pdan: Pyramid dilated attention network for action detection," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 2970–2979, January 2021.

[38] P. Ramachandran, N. Parmar, A. Vaswani, I. Bello, A. Levskaya, and J. Shlens, "Standalone self-attention in vision models," *arXiv preprint arXiv:1906.05909*, 2019.

[39] P. Shaw, J. Uszkoreit, and A. Vaswani, "Self-attention with relative position representations," *arXiv preprint arXiv:1803.02155*, 2018.

[40] A. Fathi, X. Ren, and J. M. Rehg, "Learning to recognize objects in egocentric activities," in *CVPR 2011*, pp. 3281–3288, IEEE, 2011.

[41] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[42] B. Ondrej, R. Chatterjee, F. Christian, G. Yvette, H. Barry, H. Matthias, K. Philipp, L. Qun, L. Varvara, M. Christof, *et al.*, "Findings of the 2017 conference on machine translation (wmt17)," in *Second Conference onMachine Translation*, pp. 169–214, The Association for Computational Linguistics, 2017.