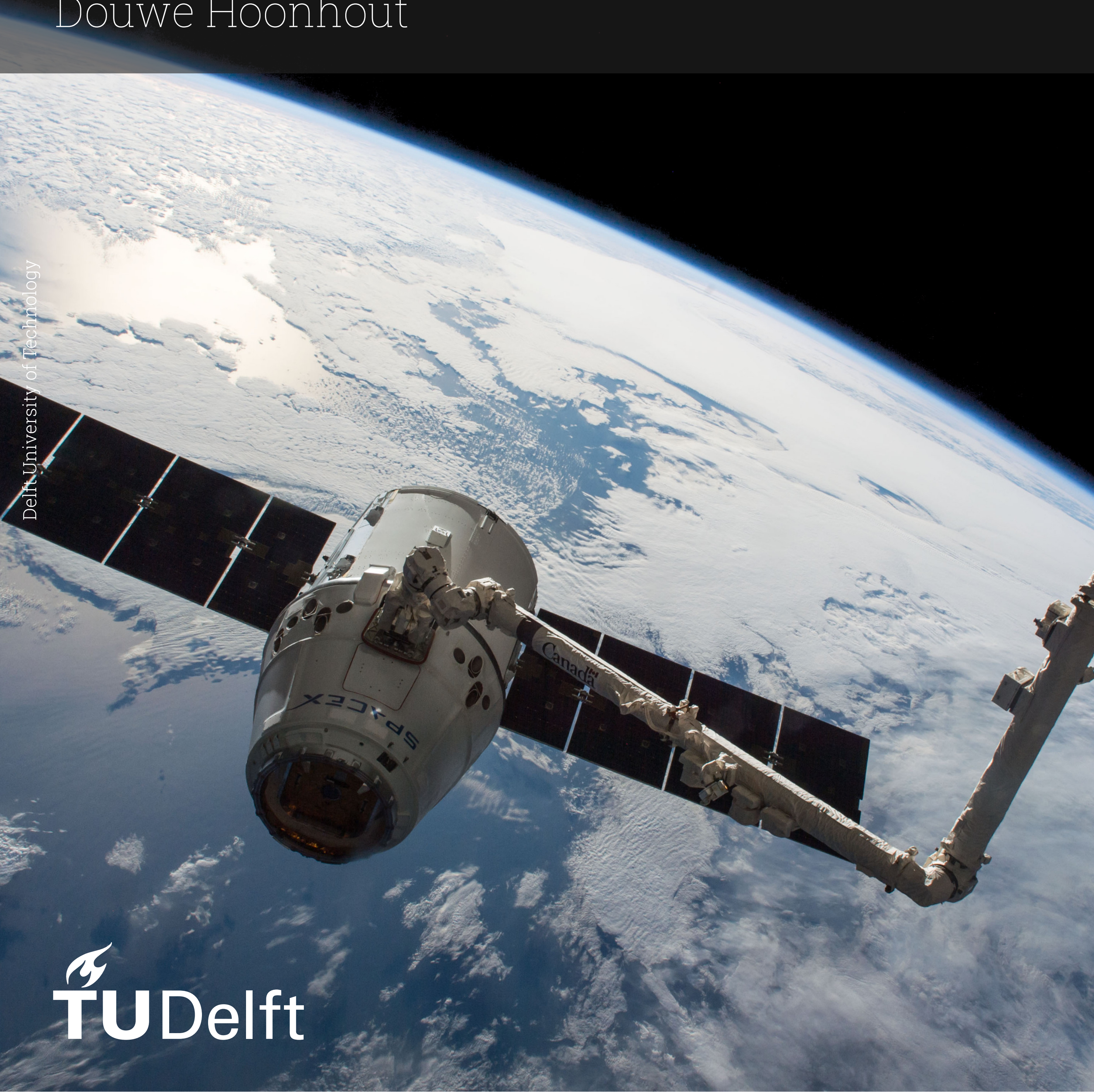
Segmenting actions by aligning video frames to learned prototypes

Douwe Hoonhout



Segmenting actions by aligning video frames to learned prototypes

by

Douwe Hoonhout

Thesis Defense Date:	19th July 2023, 10:30
Faculty:	Faculty of Electrical Engineering, Mathematics & Computer Science, Delft

Student number:	43393155	
Thesis committee:	Jan van Gemert	TU Delft Supervisor
	Silvia Pintea	Leiden University Medical Center Daily Supervisor
	Cynthia Liem	TU Delft External Supervisor

Preface

This report contains the work of my master thesis: ”Segmenting actions by aligning video frames to learned prototypes”. Hereby, I want to show my appreciation to all my supervisors that helped me during my master’s thesis. I would like to thank Silvia for her time every week for our daily meetings and for her patience when progress was hindered. I would like to thank Jan for his advice and his guidance to steer the project in the right direction. I would like to thank Cynthia for being an external supervisor during the thesis defense to give unbiased feedback on the work that has been done. Lastly, I would like to thank Jouke for making it possible to cooperate with Leiden University Medical Center and for making it possible to work at the LUMC location.

Douwe Hoonhout
Delft, July 2023

Contents

1 Introduction

The increase in video data has caused a growing interest in video understanding, leading to more research in tasks like localizing actions in long untrimmed videos. This task is also known as Temporal Action Segmentation and involves predicting the exact start and end boundaries of actions within videos. It requires framewise annotations for supervised models which is very labor-intensive. Therefore in this work, we focus on unsupervised methods. Current unsupervised methods focus on self-supervised and clustering methods that use the temporal structure in videos. The original features can be very noisy due to occlusions or lighting changes or because actions consist of a combination of motions. Combining them with temporal information can therefore result in better action segments. In this work, we propose the first alignment-based unsupervised model. Our goal is to learn informative action frames which we call prototypes. These prototypes will be used together with the original video in an alignment algorithm to obtain predicted segments of the actions within the video.

In chapter 2, a scientific article can be found that will explain most of the research that has been done. Furthermore, supplementary material can be found in chapter 3. This chapter includes some more background knowledge for the reader that are no experts on this topic and also provides some additional visualizations that did not fit in the scientific article.

2
Scientific Article

Segmenting actions by aligning video frames to learned prototypes

Douwe Hoonhout
Computer Vision Lab,
Delft University of Technology

Silvia L. Pintea
Division of Image Processing (LKEB),
Leiden University Medical Center

Jouke Dijkstra
Division of Image Processing (LKEB),
Leiden University Medical Center

Jan C. van Gemert
Computer Vision Lab,
Delft University of Technology

Abstract

Video temporal action localization is the task of identifying and localizing specific actions or activities within a video stream. Instead of only classifying which actions occur in the video stream, we aim to detect when an action begins and ends. In this work, we focus on solving this task without any supervision. Existing unsupervised methods solve this task by exploiting a combination of spatial and temporal information. We propose a new model that uses a MLP (multilayer perceptron to learn to sample prototype frames from a video. We use the distance between prototypes and video frames given by DTW (dynamic time warping) as a loss function to update the MLP. The sampled prototypes allow us to find the start and end boundaries of actions, when combined with DTW. Additionally, the prototype frames can be used for video summarization. We analyze our model in a controlled synthetic data setup, to show the weaknesses and strengths of our models. Additionally, we use the Breakfast *dataset, and* Cholec80 *surgery dataset to compare our model to the state-of-the-art models in a real scenario.*

1. Introduction

The increase in video data has caused an increasing interest in video understanding. One of the tasks in video understanding that is becoming more popular is localizing actions in long untrimmed videos. Videos are composed of multiple unitary tasks. For example, a video of a surgery consists of administering an anaesthetic, opening the patient, performing surgery, closing up and cleaning the operation room. Understanding duration and ordering of actions can have a wide range of applications from video retrieval to surveillance analysis [7, 18, 23, 31]. This task is referred to as “temporal action segmentation” and requires predicting exact start and end boundaries of actions in videos. In a supervised setting this means that we would require framewise action labels. Creating these framewise annotations is very labour intensive and therefore we focus on unsupervised methods.

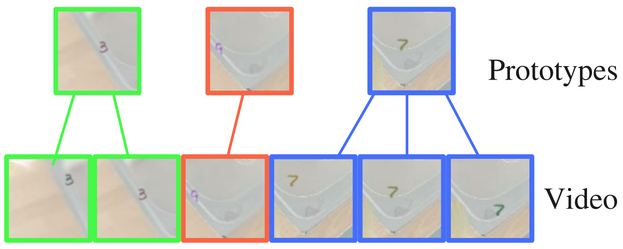


Figure 1. Our model learns to sample prototype frames. These prototypes can be used to align to the video and to label the video frames. The video-frames are from our synthetic dataset DigitVideos where moving digits represent actions.

Current unsupervised methods rely on auxiliary self-supervision task [1], and clustering methods [17, 27]. The original features can be very noisy due to occlusions, camera motion or lighting changes. Therefore, combining the features over time can result in better action segments [3, 17, 27]. Existing work finds temporal action boundaries by looking at rapid changes in features space [1, 11]. Other prior works [17, 27] assume that actions have a fixed order and therefore cannot deal with different action orderings or when actions reoccur in the videos.

In this work, we do not make any assumptions about the action ordering. We use the framewise features to learn to sample the most likely action prototypes via a simple MLP and a *Gumbel softmax*. The sampled prototypes and the input video are aligned using DTW (dynamic time warping) to obtain framewise prediction labels. Figure 1 shows an example of such an alignment. The prototype frames represent distinct actions and therefore align to different segments of the input video.

We make the following main contributions: (i) We propose an unsupervised video segmentation method that is trained over multiple videos and fine-tuned on each individual test video; (ii) Our model learns to sample informative and distinct video frames as task prototypes and then aligns the each input video frame to the prototypes via DTW; (iii) We perform extensive analysis and conclude that our method is competitive to the state-of-the-art on both synthetic video datasets, as well as the *Breakfast* [15], and *Cholec80* [29] surgery video dataset.

2. Related work

Video embedding for video segmentation. Video embeddings refer to numerical features that are extracted from deep learning architectures or other feature extraction methods. Common datasets on which unsupervised action localization is performed, already have pre-computed features [16]. This makes comparisons fair and consistent. Most of these features are obtained using video descriptors (HOG, HOF, MBH). In this work, synthetic data is used and therefore pre-computed features still need to be obtained. We follow [16] to obtain a reduced fisher vector representation of the improved dense trajectories [33], which builds on the method of dense trajectories [32]. We follow this framework because [17] showed that these feature representations perform better than pre-trained models [4, 14] in an unsupervised setting.

Weakly-supervised temporal action localization. There are primarily two distinct approaches to offer weak supervision for action segmentation: set-based [19, 21, 25, 22] and transcript-based [5, 6]. The main difference being that set-based methods do not know the action orderings within videos. Most set-based methods focus on generating action transcripts or pseudo-labels [19, 21, 25] to solve this task. A key recent observation for set-based labels is that videos with the same task follow a common action ordering which can be leveraged for effective learning [22].

In transcript-based supervision, orderings are already known and therefore most research leverage this information by using alignment methods. In [5] these transcripts are used in combination with DTW to learn distances between a video sequence and a transcript. More recently [6] used transcripts to learn prototypes for each action. It uses discriminative modeling by using positive and negative transcripts combined with DTW for training. Using the hinge-loss on the discrepancy values outputted by DTW, optimal prototypes can be learned. We use DTW in our loss function to align video frames to prototypes. However, our focus is on learning the prototypes in an unsupervised manner.

Unsupervised action segmentation. Unsupervised methods either rely on some form of self-supervision, some form of clustering using a combination of the framewise features with some temporal information or boundary change detection techniques. Traditional clustering algorithms suffer from noisy features and do not model any temporal structure of the videos. These problems can be solved by using a combination of feature space proximity and their respective positions in time. This was done in [27] which is an extension of First Integer Neighbor Clustering Hierarchy (FINCH) Algorithm [28]. Two main limitations of this algorithm are that it cannot deal with (1) alternating actions that have similar temporal positions and (2) equivalent repeated actions that have large temporal differences. Kukleva et al. [17] used an MLP to predict the timestamp of frames. The learned embedding of the last layer is clustered using K-Means. The average temporal location of the frames in the clusters are used together with Viterbi encoding to make a final prediction. This assumes a fixed transcript. Instead, a maximum a posteriori probability (MAP) estimate can remove this assumption as done by [20]. This method learns a action-level temporal feature embedding by using a new self-supervised method that detects whether action orderings are shuffled or not. Another approach that does not exploit temporal information, is looking at abrupt changes in feature space [1, 11]. LSTM + AL [1] is a self-supervised model that learns to predict the features of the next timestep. If the difference between the predicted features and the observed features is large then this might indicate a change of actions. A key observation is that abrupt changes in feature space can both indicate a change of action and also be a noisy outlier. However, noise and outliers can be mitigated by using smoothing filters on the features [11]. Using the smoothened features, similarity between frame t and t + 1 is calculated. If the frames are dissimilar this indicates a change of action. The most recent unsupervised method [3] learns a better feature representation of the videos by training a MLP that is optimised using a triplet loss. The positive anchors are established by the same observation from [27], which states that the similarity should be calculated by a combination of feature space similarity and temporal distance. In contrast to the above methods, our method is the only unsupervised method that uses an alignment based solution to solve the temporal structure in videos.

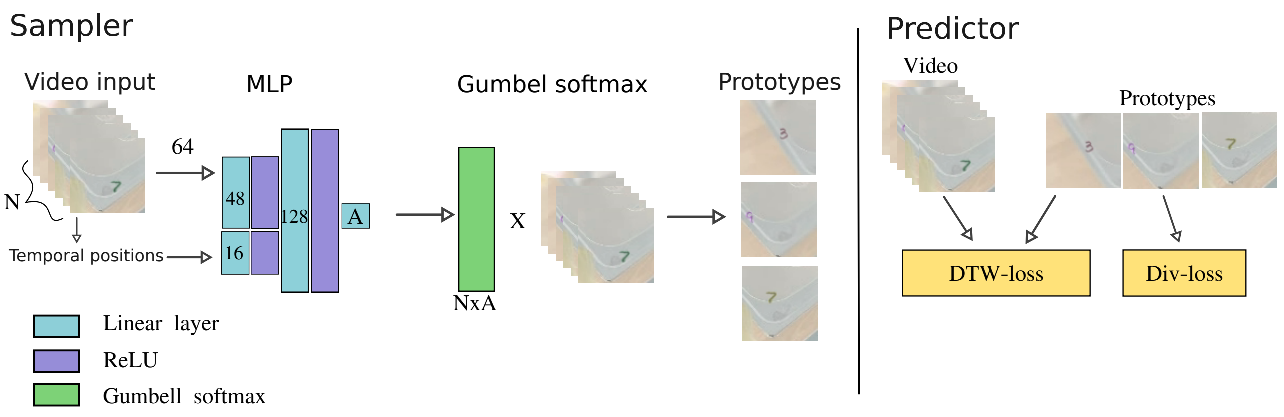


Figure 2. Overview of the model where prototypes are learned by sampling from the video. The network consists of a sampler network and a predictor network. The sampler samples prototypes by processing the video features via a MLP, and selecting informative prototype frames using *Gumbel softmax*. The prototypes are then used by the predictor to calculate the DTW optimal alignment between the input video. The distance between the aligned prototype and input video are used as a loss that propagates back to the MLP.

Dynamic time warping (DTW). DTW computes the discrepancy value between two sequences based on their optimal alignment from dynamic programming [26] and has many different applications. One way of applying it in the domain of machine learning is by using DTW between the test set and the training set combined with one nearest neighbor (1-NN) to find the right class label [10]. Or, if no labels are provided, DTW can be used as a time-series variant of k-means (Dynamic Time Warping Barycenter Averaging) [24]. DTW is also found in tasks such as video synchronization [12] and video retrieval. One problem with DTW is that it can result in degenerated results that align multiple actions to a single frame in the video. Therefore, in our work we use a modified version of DTW that is used to align action prototypes with videos.

3. Unsupervised prototype learning with DTW

An overview of the proposed model can be found in figure 2. The main components of the network are the sampler and the predictor. The sampler selects frames that could act as a prototype for a certain action. The predictor takes the prototypes that are selected by the sampler and uses these prototypes to align to specific regions of the video which results in a framewise prediction.

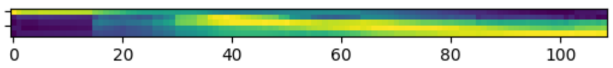


Figure 3. Heatmap of the logits after the MLP. The vertical axis represent the number of actions. The horizontal axis represent the number of frames. Heatmap shows learned regions for each action.

3.1. Sampler network

The sampler is responsible for finding frames that could potentially be a good prototype for an action. The input for the sampler are framewise features that are based on improved dense trajectories following the procedure in [16]. The sampler consists of an MLP with three linear layers followed by *Gumbel softmax* to do the sampling. The first layer takes as input features of 64 dimensions and the last layer of the network outputs the number of dimensions equal to the number of prototypes A the sampler needs to learn. The output of the MLP are the logits that indicate for each prototype which frames are most likely good candidates. Figure 3 shows a heatmap visualization of the logits after some iterations of learning. Each row reflect regions of interest for a single action prototype. The next step is to do the sampling of the prototype frames based on this heatmap. Using an *argmax* does not work here as *argmax* is not differentiable. Therefore, we use *Gumbel softmax* which is a differentiable approximation for sampling discrete data. The *Gumbel softmax* [13] takes logits from the MLP and uses these to sample a frame for each prototype.

The *Gumbel softmax* function starts from unnormalized logits π_i, and samples g_i from Gumbel distribution, and uses τ as the temperature hyperparameter. And it predicts sample probabilities y_i that approximate a one hot encoding for the sampled data:

$$y_i(\pi) = \frac{\exp\left(\frac{\log(\pi_i)+g_i}{\tau}\right)}{\sum_{j=1}^{k}\exp\left(\frac{\log(\pi_j)+g_j}{\tau}\right)}, \qquad (1)$$
$$g_i \to \text{Gumbel}(0,1)$$

When $\tau \to \infty$ the logits will be uniformly distributed whereas $\tau \to 0$ the output will output approximately one-hot encodings. Equation 2 shows an example of the *Gumbel softmax*. Here the first entry of the input logits π is sampled. Therefore the output approximates the one-hot encoding 1000.

We can now formulate the whole pipeline. We input video $v_i \in \mathbb{R}^{T_i \times M}$ as a batched input to the MLP. The video v_i has temporal length T and dimensionality M. The number of prototypes that need to be learned is A and therefore the number of nodes of the first layer of the MLP is M and the number of nodes in the last layer should be A. Let $L \in \mathbb{R}^{A \times T_i}$ be the output logits of the MLP. Here each row represent the probability logits to sample some frame for one prototype. Now we can apply the *Gumbel softmax* over each row to obtain the one-hot encoded indices of the frames that are sampled.

$$\pi = \begin{Vmatrix} 0.5 \\ 0.35 \\ 0.15 \\ 0.1 \end{Vmatrix} \xrightarrow{\text{Gumbel Softmax}} y(\pi) = \begin{Vmatrix} 0.880 \\ 0.059 \\ 0.030 \\ 0.031 \end{Vmatrix} \tag{2}$$

3.2. Predictor network

The predictor takes an input video $\mathbf{v}=\{v_i\}_{i\in\{1,..T\}}$ together with the prototypes $\mathbf{P}=\{p_j\}_{j\in\{1,..A\}}$. It considers all permutations of prototypes and chooses the permutation that best aligns to the video. This makes sure that the actions that are swapped do not impact the performance of our model. Then we will use DTW that gives an alignment, which can be used to find the different segments in the video.

DTW takes two temporal sequences as input and finds an optimal alignment that minimizes the distance by taking into account temporal shifts of the inputs. Furthermore, DTW is capable of computing distances even when the inputs have disparate temporal lengths. Figure 4 shows an example of two inputs with different temporal lengths. Here the distance is computed between the sampled prototypes and the video. The resulting alignment can be used to label the video. DTW is known to be efficiently solved using dynamic programming: it iteratively fills a distance table which memorizes previously calculated results. For both input sequences it starts at the first element and progresses towards the last elements. There are many different paths to get from the first elements to the last elements as DTW allows for temporal corrections by either stretching or compressing. Equation 3 shows the basic formula for filling the distance matrix. Here the minimum value is taken from three distinct cases. Either we temporally match, or we stretch or compress the time dimension. The current cost for aligning frame v_i with prototype p_j is the current distance cost at location (i, j) plus the minimum of previous points in the table.

$$\begin{aligned} DTW[i,j] = \text{distance}(i,j) + \min(&DTW[i-1,j-1], \\ &DTW[i,j-1], DTW[i-1,j]) \end{aligned} \tag{3}$$

One problem with this original formulation is that one video frame v_i can be aligned with multiple prototypes. The alignment path in figure 4 should be able to go diagonally, horizontally but not vertically. Thus, we remove $DTW[i-1, j]$ from equation 3 to arrive at our modified DTW function. This removes cases where it is unclear how to label a certain frame v_i, but also reduces the search space and therefore makes it computationally more efficient.

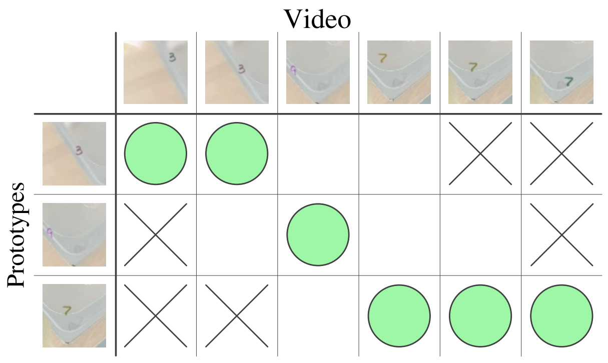


Figure 4. Optimal alignment path that is given by dynamic time warping using the prototypes as one input and the video as the second input. The original dtw algorithm is modified such that one video frame cannot align to multiple prototypes.

$$\begin{aligned} DTW[i,j] = \text{distance}(i,j) + \\ \min(DTW[i-1,j-1], DTW[i,j-1]) \end{aligned} \tag{4}$$

3.3. Loss function

To optimise the sampler MLP, we need to define a loss function. We want a large loss when frames are sampled that are uninformative, and a small loss when the sampled frames are good prototype candidates. The loss function should be based on the distance between the video and the prototypes, and therefore we use DTW for this as well. The MLP does not know a priori the ordering of the prototypes, and multiple tasks orderings can be present across videos. Therefore, for a given video $\mathbf{v}$, we consider all possible permutations of prototype orderings perm$(\mathbf{P})$ and evaluate DTW across these:

$$L_{dtw}(\mathbf{v}, \mathbf{P}(\theta)) = \min_{\mathbf{P}' \in \text{perm}(\mathbf{P}(\theta))} f_{DTW}(\mathbf{P}', \mathbf{v}), \tag{5}$$

where θ are the parameters of the MLP and f_{DTW} is the function implementing DTW as described in equation 4. We use the action ordering that in the end minimizes the DTW distance.

3.4. Diversity loss

During training one possible outcome is that the model will try to learn very average but adequate prototypes. We will evaluate the use of a diversity loss that will steer the model into learning more distinct prototypes. We follow the procedure of [2]. Essentially, we calculate the dot product Ω_{mn} between each pair of prototypes $(\mathbf{p}_m, \mathbf{p}_n)$ to evaluate how similar they are 7. Then the total diversity loss is calculated by simply summing the square of the dot product values 6, where we mask all dot-product values smaller than 0.5, using M_{mn}:

$$L_{div}(\mathbf{P}(\theta)) = \sum_{m=1}^{A}\sum_{n=1}^{A}\Big(\Omega_{mn}^2 M_{mn}\Big), \tag{6}$$

$$\Omega \in (\mathbf{P}(\theta)^T \times \mathbf{P}(\theta)), \text{ and } \Omega_{mn} = \mathbf{p}_m^T\mathbf{p}_n \tag{7}$$

$$M_{mn} = \begin{cases} 1, & \omega \leq \Omega_{mn} \leq 1 \\ 0, & i = j \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

where ω is set to 0.5. We now formulate the total loss as 9:

$$L_{total}(\mathbf{v}, \mathbf{P}(\theta)) = L_{dtw}(\mathbf{v}, \mathbf{P}(\theta)) + \beta L_{div}(\mathbf{P}(\theta)), \tag{9}$$

where β is a hyperparameter that needs to be tuned.

3.5. Temporal encoding

Some previous methods [17, 27] use temporal assumptions to improve the performance of the model. Therefore, we aim to evaluate whether adding normalized temporal positions of the frame features is beneficial. We follow the procedure of [30]:

$$PE(t, 2i) = \sin(t/10000^{2i/16}), \tag{10}$$

$$PE(t, 2i+1) = \cos(t/10000^{2i/16}), \tag{11}$$

where $t \in [0, 1]$ is the normalized temporal position of the frames, and i indexes the dimension of the embedding. The temporal information and the frame-wise features each go through an independent linear layer after which the two are concatenated. The first layer is replaced with a single 64 linear layer in experiments where no temporal positions are used.

3.6. Loss function with windowed videoclips

The permutation loss function 5 works well when the action ordering is swapped. However, it does not solve the problem when videos contain repeated tasks, because prototypes cannot be reused to align to a second video segment: i.e. each prototype aligns with one video segment. Therefore, we consider a second loss function that can handle repetitions of actions.

Instead of aligning the prototypes to the complete video, we use smaller video segments. The length of these segments is a hyperparameter that has to be tuned. Initially, the clip length can be set in such a way that clips include at most two different actions as the model is only able to use two prototypes per clip. Instead of using permutations of all prototypes as in equation 5, we use the Cartesian product of prototypes $\mathbf{P}(\theta)$. This gives us a set with all the different ordered pairs $(\mathbf{p}_n, \mathbf{p}_m)$ where $\mathbf{p}_n \in \mathbf{P}(\theta)$ and $\mathbf{p}_m \in \mathbf{P}(\theta)$. We will divide the video $\mathbf{v}$ into K clips where each clip has the same temporal length. Clip $\mathbf{c}_l$, $l \in \{1, ..K\}$ is the l-th clip of video $\mathbf{v}$. The DTW loss function becomes:

$$L_{dtw}(\mathbf{v}, \mathbf{P}(\theta)) = \sum_{l=1}^{K}\Bigg(\min_{(\mathbf{p}_m, \mathbf{p}_n)\in(\mathbf{P}(\theta)\times\mathbf{P}(\theta))} f_{DTW}((\mathbf{p}_m, \mathbf{p}_n), \mathbf{c}_l)\Bigg), \tag{12}$$

where θ are the MLP parameters.

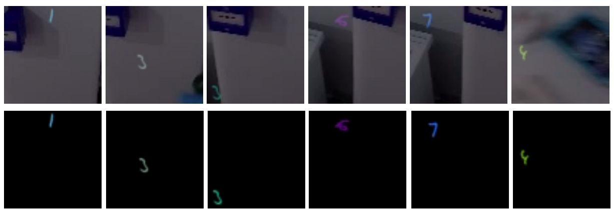


Figure 5. Some example frames of the synthetic dataset. The synthetic setup allows us to analyse our model in a full-controlled environment.

4. Experimental analysis

Datasets description. We created a synthetic dataset *DigitVideos* to have control over the number of actions and the ordering. It consists of moving MNIST [9] digits that represent action motions. Each digit is associated with a different specific motion. We only consider the digits 1, 3, 5, 7, 9 which correspond to horizontal, inverse diagonal, inverse horizontal, diagonal and vertical respectively. We put the moving digits on top of background frames taken from EPIC-KITCHEN [8] to bring the data closer to a real action segmentation dataset . This generates roughly five times as many trajectories and thus makes the features more noisy and realistic.

We consider three variations of this dataset to research whether models perform differently in different scenarios: one set of videos contains consistent action orderings — *DigitVideos-order*; one set contains videos where occasionally actions are swapped *DigitVideos-swap*; and one set contains videos where actions can be repeated multiple times – *DigitVideos-repeat.* Furthermore we do an ablation study on a normal action ordering with (*DigitVideos-bg*) and without (*DigitVideos-no-bg*) background frames to see the effect of the model enhancements in different setups. All variations of the dataset contain 200 videos with a resolution of 70x70. Figure 5 show some example frames.

Next to the *DigitVideo* dataset, we evaluate on *Breakfast* [15] which is a popular action segmentation dataset. In this dataset 52 participant are asked to execute 10 different cooking tasks: 'cereals', 'coffee', 'friedegg', 'juice', 'milk', 'pancake', 'salat', 'sandwich', 'scrambledegg', 'tea'. For each cooking activity a separate model is trained as the action motions across cooking activities is not consistent.

	DigitVideos-no-bg		
Method	MoF	F1	IoU
Our	0.793	0.769	0.660
Our + L_{div}	0.738	0.680	0.571
Our + *temp. embed.*	0.822	0.802	0.697
Our + L_{div} + *temp. embed.*	0.792	0.773	0.666
Our-window	0.743	0.731	0.615
Our-window + L_{div}	0.692	0.664	0.545
Our-window + *temp. embed.*	0.762	0.751	0.634
Our-window + L_{div} + *temp. embed.*	0.759	0.744	0.624

Table 1. Results of various model implementations on the synthetic dataset *DigitVideos-no-bg*. Since the videos do not contain background noise, the learned prototypes are already distinct. Therefor the diversity loss does not increase the performance of the model.

	DigitVideos-bg		
Method	MoF	F1	IoU
Our	0.683	0.639	0.524
Our + L_{div}	0.693	0.639	0.528
Our + *temp. embed.*	0.722	0.698	0.580
4 Our + L_{div} + *temp. embed.*	0.727	0.693	0.582
Our-window	0.674	0.651	0.530
Our-window + L_{div}	0.667	0.641	0.517
Our-window + *temp. embed.*	0.702	0.685	0.563
Our-window + L_{div} + *temp. embed.*	0.707	0.692	0.568

Table 2. Results of various model implementations on the synthetic dataset *DigitVideos-bg* with added background. Adding temporal information can significantly improve performance whereas diversifying the prototypes only increases the performance by a small margin.

The last dataset we evaluate on is *Cholec80* [29]. *Cholec80* is a endoscopic video dataset. It contains 80 videos of cholecystectomy surgeries performed by 13 different surgeons. For this dataset we use I3D [4] features with a dimensionality of 1024. Therefore the layers in the MLP are replaced with layers of size 1024, 256, 128. Furthermore the frames are downsampled to 0.5 frames per second to make it computationally feasible.

Evaluation metrics. Our model only segments the video into segments without any specific labelling. We follow [1, 27] and use Hungarian matching to obtain the optimal one to one mapping from prediction to ground truth labels. Using the matched labels we then follow [3, 17, 20, 27] and evaluate based on the three metrics that are used for this task: (1) *Mean over frames (MOF)* percentage of correctly predicted frames also referred to as accuracy; (2) *F1 score* computed over the predicted and ground truth frames; (3) *Intersection over Union (IoU)* also known as the Jaccard score.

Training procedure. We use all videos for training and

	DigitVideos-order		
Method	MoF	F1	IoU
K-means	0.651	0.620	0.500
Uniform	0.689	0.646	0.524
CTE [17]	0.670	0.569	0.462
TW-FINCH [27]	0.707	0.669	0.552
Our	0.727	0.693	0.582
Our-window	0.707	0.692	0.568

Table 3. Results of all models on the synthetic dataset *DigitVideos-order* with background noise. Actions always occur once and in the same order. Our model is slightly better compared to SOTA in this setup. Clip length is set to 70.

	DigitVideos-swap		
Method	MoF	F1	IoU
K-means	0.636	0.603	0.484
Uniform	0.686	0.640	0.518
CTE [17]	0.651	0.533	0.424
TW-FINCH [27]	0.695	0.652	0.533
Our	0.689	0.660	0.547
Our-window	0.673	0.660	0.531

Table 4. Results of all models on the synthetic *DigitVideos-swap* dataset. Actions always occur once and are swapped with a chance of 30%. Our model is on par with SOTA models. Clip length is set to 70.

update the model weights after having seen a video. A batch therefore contains all frames of a single video. If a video contains a large amount of frames we downsample the video by uniformly selecting frames. We train the model over 500 epochs using an ADAM optimizer with a learning rate of 10^{-3} and a weight decay of 10^{-4}. For the Gumbel-Softmax we use a temperature τ of 10^{-1}. The hyperparameter β for the diversity loss is set to 0.1. A low value for the diversity loss seemed to work better empirically.

4.1. Ablation study

We evaluate the different components of our model. Mainly we want to conclude whether using temporal features and diversifying the prototypes can increase the performance of the model. Table 1 shows the results on *DigitVideos-no-bg*. As actions have a consistent order in these videos using the temporal embedding increases the performance of the model. Diversifying the prototypes in this case does not seem to help. This is probably due to the features not being noisy and therefore there is no need to enforce that the prototype features should be far apart. Table 2 show the results on *DigitVideos-bg*. Here, the main improvement comes from the temporal embeddings. The diversity loss only slightly increases performance. This is the case with and without window-clips.

	DigitVideos-repeat		
Method	MoF	F1	IoU
K-means	0.569	0.545	0.400
Uniform	0.545	0.535	0.383
CTE [17]	0.510	0.468	0.327
TW-FINCH [27]	0.545	0.535	0.383
Our	0.540	0.519	0.373
Our-window	0.553	0.529	0.384

Table 5. Results of all models on the synthetic *DigitVideos-repeat* dataset. First and last action are identical. In the middle three actions are alternated and can occur one to three times. *K-means* is the best performing model in this setup. Action orderings are random and therefore models that make temporal assumptions do not perform optimally. *Our-window* clips is able to mitigate some temporal assumptions and therefore is the best performing model after *K-means*. Here the clip length is set to 60.

	Breakfast		
Method	MoF	F1	IoU
K-means	0.450	0.390	0.269
Uniform	0.628	0.478	0.378
CTE* [17]	0.443	0.217	0.125
TW-FINCH [27]	0.624	0.466	0.371
Our-window	0.627	0.409	0.325

Table 6. Results of all models on the *Breakfast* dataset. An asterisk means that performance is calculated over all videos instead of on single videos. *Uniform* sampling is a strong baseline. The data is noisy and contains different camera angels, thus our model is less likely to extract global information from videos. The model without windowed-clips is too computationally expensive, because of the large number of permutations. All obtained numbers are from own experiments.

	Cholec80		
Method	MoF	F1	IoU
K-means	0.430	0.374	0.258
Uniform	0.600	0.438	0.342
CTE [17]	0.588	0.424	0.323
TW-FINCH [27]	0.584	0.487	0.384
Our	0.608	0.399	0.321

Table 7. Results of all models on the *Cholec80* dataset. CTE performs well on this dataset because action orderings are fixed. *Uniform* is is the best on this data.

4.2. Comparison with the State-of-the-art

We benchmark our model on variants of the *DigitVideos* dataset and on the *Breakfast* dataset [15], and *Cholec80* [29] surgery videos. We compare with two state-of-the-art models: CTE [17] and TW-FINCH [27]. Additionally, we consider two simple baselines: *K-means* which simple

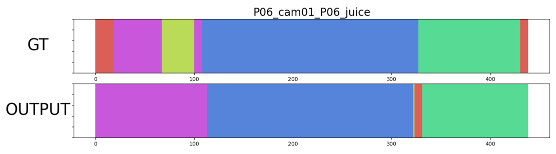


Figure 6. Predictions on *P06_juice* from *Breakfast*. The video contains repeated actions, whereas we train our model without clips. The model cannot learn repeated segments but instead finds a more coarse segmentation that does not contain repetitions.

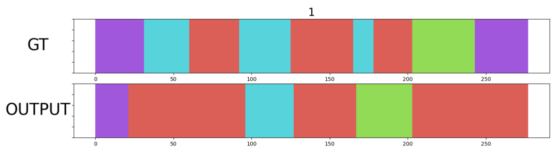


Figure 7. Our windowed model is not able to segment the video correctly. Our hypothesis is that the features of equivalent actions are not similar enough because of the background noise.

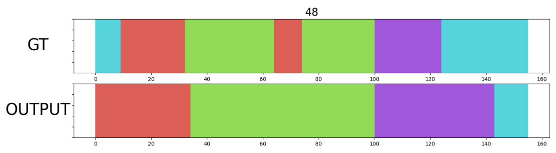


Figure 8. The repeated action is not recognised by the model. The clip length might be too small. However a smaller clip-length could also result in more incorrect segmentations as noise becomes more present.

selects the protoypes as cluster centres in the image domain, and *Uniform* which samples prototypes uniformly from the video and aligns them using DTW.

Comparisons on synthetic data. Tables 3, 4 and 5 show the results on different action orderings, swapped actions and repeated actions. On *DigitVideos-order* our model slightly outperforms the baselines and SOTA models. This could be because the action ordering is consistent and therefore the additional temporal information could be useful. Important is outperforming *Uniform* as this justifies that the model is indeed learning to sample better prototypes. On *DigitVideos-swap* our model is on par with the SOTA. The actions are sometimes swapped and therefore exploiting temporal information is less useful. In *DigitVideos-repeat* the action ordering is even more random. Therefore, the temporal embedding is also not used here. *K-means* is the best performing model as it does not make any temporal assumptions. Our windowed version of our model is able to outperform all other models. Interesting is that *Uniform* is actually a strong baseline. This suggests that DTW is able to take care of the alignment even when frames are sampled uniformly.

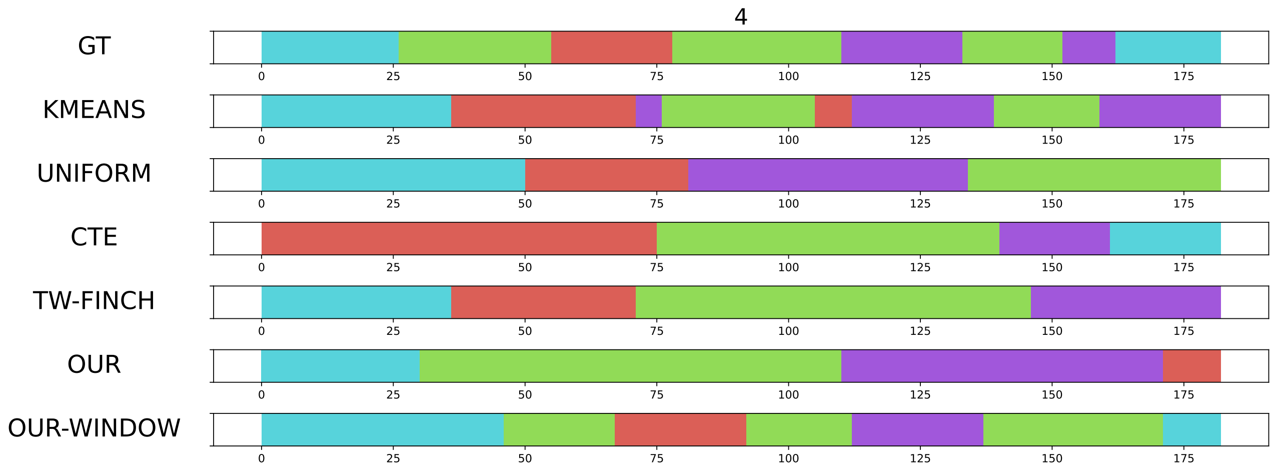


Figure 9. Predictions on a video from *DigitVideos-repeat*. Our windowed model has a good balance between temporal assumptions and being able to handle the repetitions of actions. Therefore it performs the best for this specific video.

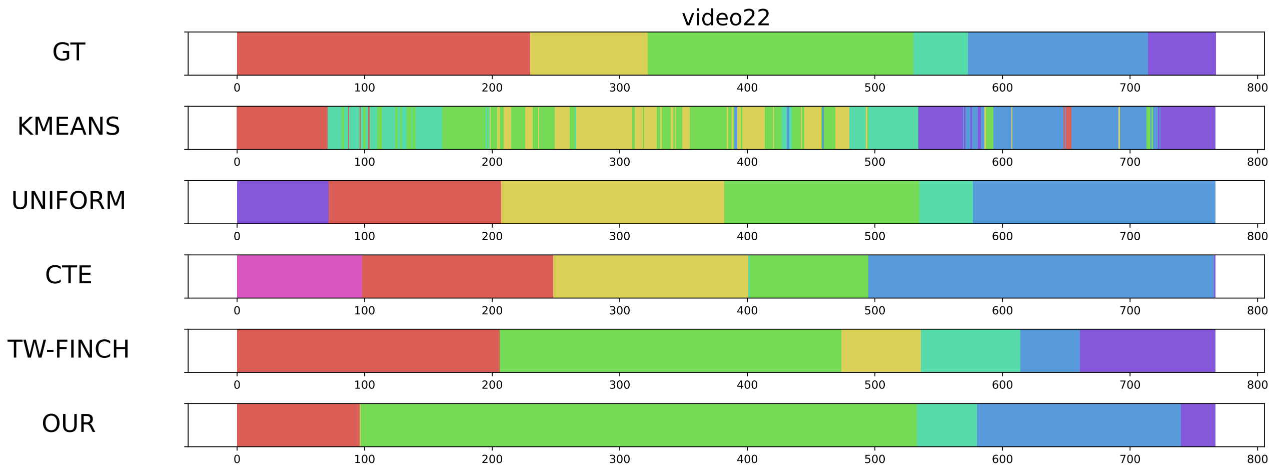


Figure 10. Predictions on a video from Cholec80. Our model over-segments the green part as this part is quite noisy. Our model best predicts the final part of the video but struggles in the other part of the video.

Comparisons on Breakfast video benchmark. Results on the *Breakfast* dataset are shown in table 6. Our model is difficult to train on this dataset because the number of unique actions within a set of cooking activity videos is inconsistent. Therefore we need to train a single model per video. Some videos also consist of many frames and therefore we uniformly sub-sample frames to ensure that the number of frames never exceeds 500 frames during training. The *Uniform* method outperforms the SOTA models by a small margin. This indicates that DTW can take care of the alignments by just uniformly sampling from the video. Because the *Uniform* baseline cannot deal with repeated actions, it seems that existing models also fail to recognize the repetitions of actions. Our model is on par with SOTA models. However, it does not learn any better prototypes than *Uniform* sampling based on the performance scores. This is partially because the large number of unique actions made the model computationally inefficient.

Comparisons on the medical Cholec80 dataset. Table 7 shows the results on the *Cholec80* dataset. *Uniform* is on par with SOTA models and also achieves the highest MoF score. *CTE* also performs well as actions in this dataset follow a fixed ordering. Our model underperforms. The main limitation of our model is that training for long videos became computationally infeasible, so we could not properly train our model.

5. Limitations and discussion

Firstly, the number of nodes in the last layer of the model is dependent on the number of prototype frames that need to be learned. Therefore, the model cannot handle video datasets that contain a varying number of unique actions per video. Secondly, our model is quite computationally expensive. This is mainly because the model needs to be able to handle different orderings of the actions. Therefore it needs to run DTW for all permutations of prototypes. Lastly, the clip length is static and therefore it cannot deal with cases where the variance in action duration is large. Figure 8 shows an example where a segment is missed because of varying action lengths. In the future, the model could perhaps be extended by generating pseudo-action orderings. Then there would be no need to consider all permutations in the DTW loss. In addition, the model could be improved by finding a way to incorporate a dynamic clip length.

6. Conclusion

This paper proposed a new approach to temporal action localization where a network learns to sample prototype frames of actions that are contained within a video. We introduced an unsupervised model that uses DTW to align prototypes to video segments and that uses DTW as a loss function to update the model. The model is on par with other SOTA models. However, it also has its limitations: (1) when introduced with a dataset that has an inconsistent number of action; (2) when actions have extreme variance in duration and (3) is computationally expensive. Therefore the model mainly performs well in cases where the number of actions is consistent and the variance in action duration is not too extreme.

References

[1] S. N. Aakur and S. Sarkar. A perceptual prediction framework for self supervised event segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1197–1206, 2019.

[2] B. O. Ayinde, K. Nishihama, and J. M. Zurada. Diversity regularized adversarial learning. *arXiv preprint arXiv:1901.10824*, 2019.

[3] E. Bueno-Benito, B. Tura, and M. Dimiccoli. Leveraging triplet loss for unsupervised action segmentation. *arXiv preprint arXiv:2304.06403*, 2023.

[4] 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*, pages 6299–6308, 2017.

[5] C.-Y. Chang, D.-A. Huang, Y. Sui, L. Fei-Fei, and J. C. Niebles. D3tw: Discriminative differentiable dynamic time warping for weakly supervised action alignment and segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3546–3555, 2019.

[6] X. Chang, F. Tung, and G. Mori. Learning discriminative prototypes with dynamic time warping. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8395–8404, 2021.

[7] R. T. Collins, A. J. Lipton, and T. Kanade. Introduction to the special section on video surveillance. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):745–746, 2000.

[8] D. Damen, H. Doughty, G. M. Farinella, A. Furnari, E. Kazakos, J. Ma, D. Moltisanti, J. Munro, T. Perrett, W. Price, et al. Rescaling egocentric vision: Collection, pipeline and challenges for epic-kitchens-100. *International Journal of Computer Vision*, pages 1–23, 2022.

[9] L. Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142, 2012.

[10] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. Querying and mining of time series data: experimental comparison of representations and distance measures. *Proceedings of the VLDB Endowment*, 1(2):1542–1552, 2008.

[11] Z. Du, X. Wang, G. Zhou, and Q. Wang. Fast and unsupervised action boundary detection for action segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3323–3332, 2022.

[12] I. Hadji, K. G. Derpanis, and A. D. Jepson. Representation learning via global temporal alignment and cycle-consistency. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11068–11077, 2021.

[13] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[15] 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*, pages 780–787, 2014.

[16] H. Kuehne, J. Gall, and T. Serre. An end-to-end generative framework for video segmentation and recognition. In *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1–8. IEEE, 2016.

[17] A. Kukleva, H. Kuehne, F. Sener, and J. Gall. Unsupervised learning of action classes with continuous temporal embedding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12066–12074, 2019.

[18] Y. J. Lee, J. Ghosh, and K. Grauman. Discovering important people and objects for egocentric video summarization. In *2012 IEEE conference on computer vision and pattern recognition*, pages 1346–1353. IEEE, 2012.

[19] J. Li and S. Todorovic. Set-constrained viterbi for set-supervised action segmentation. In *Proceedings of the*

IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 10820–10829, 2020.

[20] J. Li and S. Todorovic. Action shuffle alternating learning for unsupervised action segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12628–12636, 2021.

[21] J. Li and S. Todorovic. Anchor-constrained viterbi for set-supervised action segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9806–9815, 2021.

[22] Z. Lu and E. Elhamifar. Set-supervised action learning in procedural task videos via pairwise order consistency. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19903–19913, 2022.

[23] Y.-F. Ma, X.-S. Hua, L. Lu, and H.-J. Zhang. A generic framework of user attention model and its application in video summarization. *IEEE transactions on multimedia*, 7(5):907–919, 2005.

[24] F. Petitjean, A. Ketterlin, and P. Gançarski. A global averaging method for dynamic time warping, with applications to clustering. *Pattern recognition*, 44(3):678–693, 2011.

[25] A. Richard, H. Kuehne, and J. Gall. Action sets: Weakly supervised action segmentation without ordering constraints. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 5987–5996, 2018.

[26] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing*, 26(1):43–49, 1978.

[27] S. Sarfraz, N. Murray, V. Sharma, A. Diba, L. Van Gool, and R. Stiefelhagen. Temporally-weighted hierarchical clustering for unsupervised action segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11225–11234, 2021.

[28] S. Sarfraz, V. Sharma, and R. Stiefelhagen. Efficient parameter-free clustering using first neighbor relations. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8934–8943, 2019.

[29] 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*, 36(1):86–97, 2016.

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[31] S. Vishwakarma and A. Agrawal. A survey on activity recognition and behavior understanding in video surveillance. *The Visual Computer*, 29:983–1009, 2013.

[32] H. Wang, A. Kläser, C. Schmid, and C.-L. Liu. Dense trajectories and motion boundary descriptors for action recognition. *International journal of computer vision*, 103(1):60–79, 2013.

[33] H. Wang and C. Schmid. Action recognition with improved trajectories. In *Proceedings of the IEEE international conference on computer vision*, pages 3551–3558, 2013.

3 Supplementary Material

3.1. Dynamic time warping

Dynamic time warping is an algorithm that is used to measure the similarity between two time series. It was originally used for spoken word recognition where it solved problems of different speaking rates[11]. Now it is used in various research fields. Different speaking rates create temporal shifts in the data which makes Euclidean distance inadequate for measuring similarities. An example of this problem can be found in Figure 3.1. The Euclidean distance in this example is quite large whereas the two sequences in reality are very similar beside a temporally shift. Calculating the distance using dynamic time warping gives us a small distance instead. The idea behind dynamic time warping is to find an optimal alignment between the two time series such that the distance is minimal. Calculating the Euclidean distance using this alignment gives us the dynamic time warping distance.

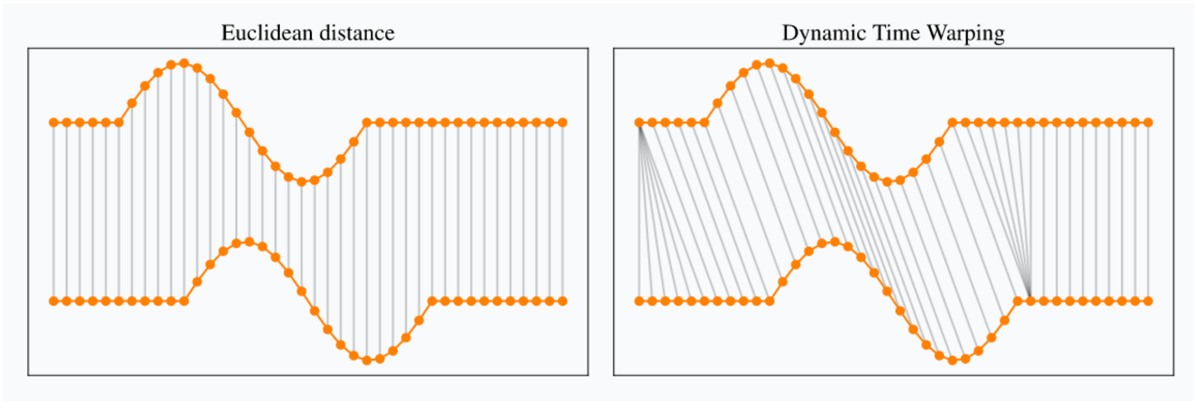
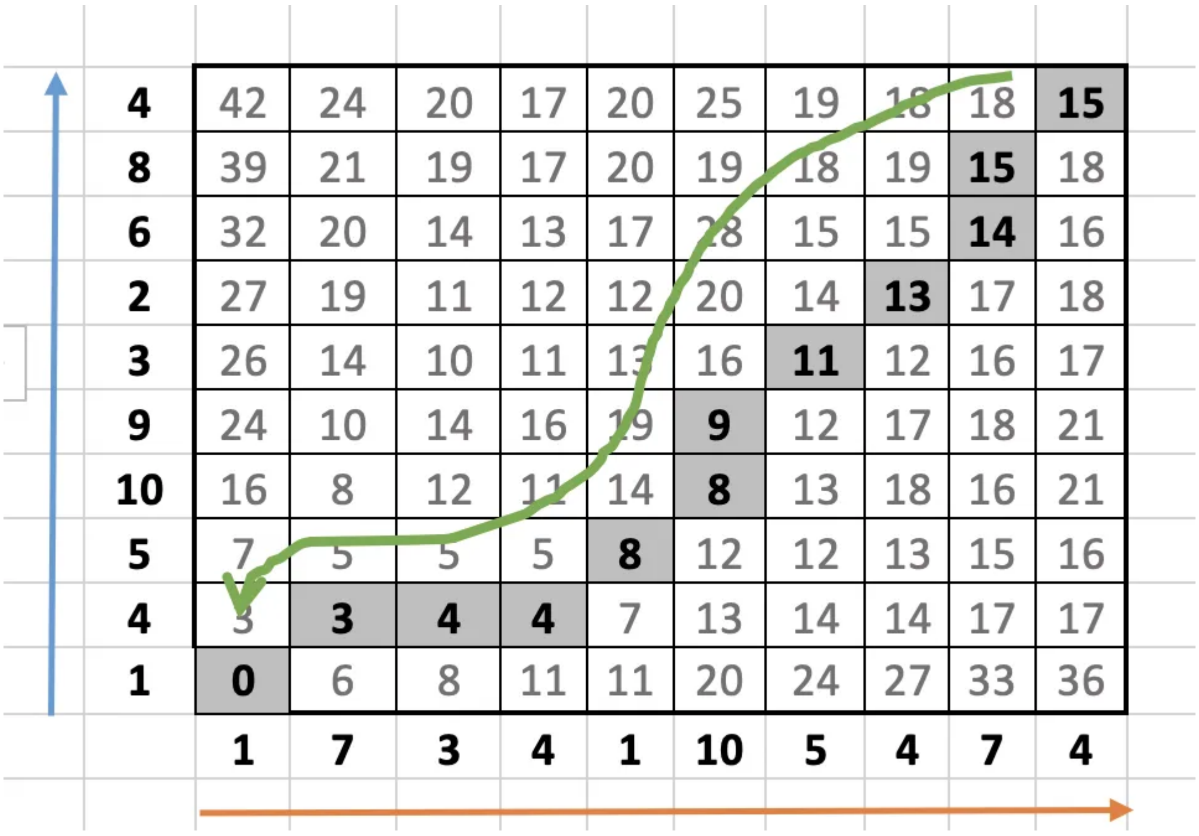


Figure 3.1: Visualization of a problem when calculating Euclidean distance of time series where one series is temporally shifted. Dynamic time warping solves this problem by finding the optimal alignment between the two series. Note that the two series are vertically shifted away from each other just to visualize the alignment. The two sequences are actually on top of each other. [13]

The algorithm of dynamic time warping iteratively computes a distance matrix D where each index $D_{i,j}$ represents the minimal cost of aligning i with j at that point as shown in figure 3.2. This process is done using dynamic programming which reuses computed values. This makes it computationally more efficient. The last entry of the distance matrix is the total cost of the optimal alignment. The optimal alignment can



4	42	24	20	17	20	25	19	18	18	**15**
8	39	21	19	17	20	19	18	19	**15**	18
6	32	20	14	13	17	28	15	15	**14**	16
2	27	19	11	12	12	20	14	**13**	17	18
3	26	14	10	11	13	16	**11**	12	16	17
9	24	10	14	16	19	**9**	12	17	18	21
10	16	8	12	11	14	**8**	13	18	16	21
5	7	5	5	5	**8**	12	12	13	15	16
4	3	**3**	**4**	**4**	7	13	14	14	17	17
1	**0**	6	8	11	11	20	24	27	33	36
	1	**7**	**3**	**4**	**1**	**10**	**5**	**4**	**7**	**4**

Figure 3.2: The distance table created by dynamic time warping to align two time series. The green line indicates the optimal alignment that gives the minimal distance between the two input time series.

be found by back-tracing back to the first entry. This is indicated by the green arrow in the figure. We start from the top right value and then the next entry is found by taking the minimal values of the surrounding entries. This is formulated as:
min$(D[i-1, j-1], D[i, j-1], D[i-1, j])$

3.2. Video descriptors

Unsupervised methods for temporal action segmentation do not work well on raw pixel values. Pixel values will constantly change as time progresses while the underlying action may be consistent. Instead, there will be a need to extract features from the videos. These feature extraction methods, also often referred to as video descriptors, are more traditional and were already researched before the era when deep learning became popular. One essential property that videos have which do not exist in plain images is that videos have a temporal structure. Therefore additional information can be exploited by tracking pixels over time which essentially is capturing motion information. This phenomenon is referred to as optical flow and is a concept that has already been around since the 1980s [10].

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \tag{3.1}$$

The formula shown in equation 3.1 is the main formula of optical flow and is the basis of all optical flow algorithms. It states that some pixel value at time t with position (x, y) is equal to some pixel value at dt with a different location. Note that for this equation to hold several conditions have to be met. Firstly, the brightness of the pixel needs to be consistent. Lighting can change the brightness of a pixel and therefore could cause problems. Secondly, occlusions might occur. Objects can move in front of each other which causes a pixel to be existing in the current frame but be occluded in the next frame. Lastly, also camera artifacts can play a role in altering pixel values.

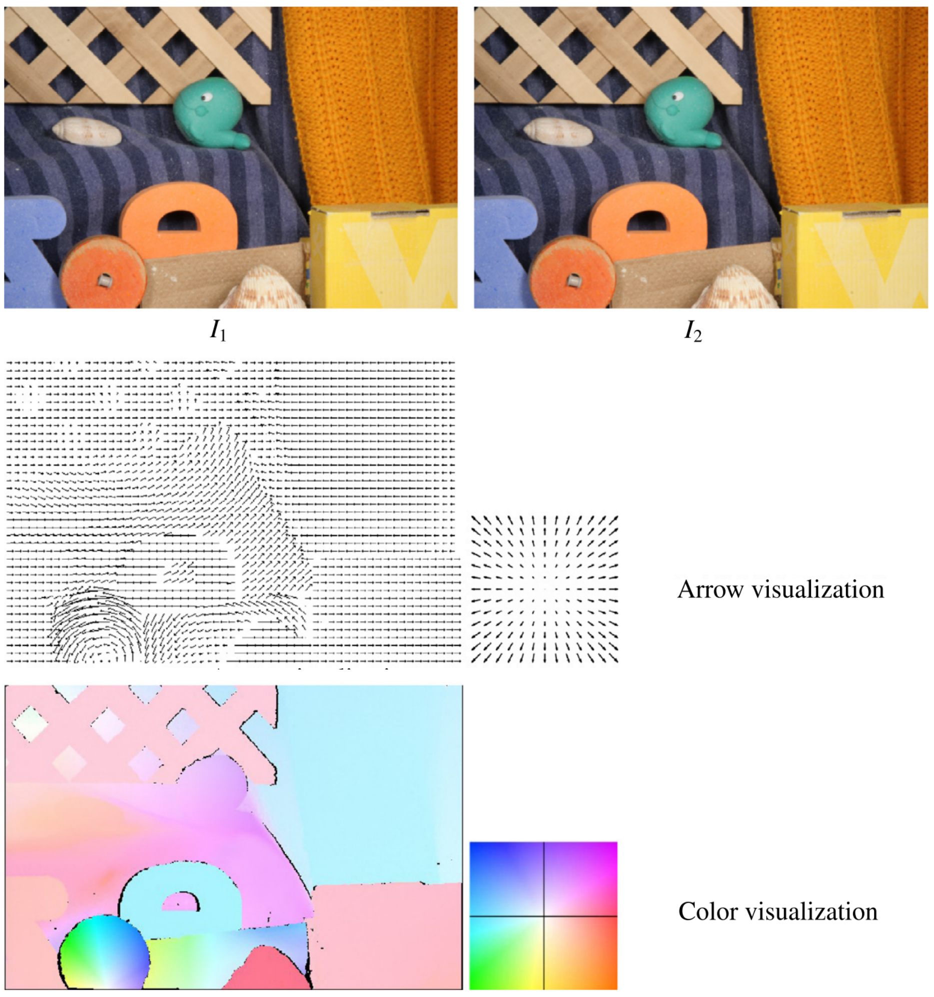


Figure 3.3: Two types of visualization of the motion field transforming I_1 in I_2. [4]

There are various implementations of optical flow algorithms and there are many ways to use the optical flow for various applications. The following subsections will describe some key applications of using optical flow.

3.2.1. Dense sampling

Optical flow tracks pixels over time in a video. This results in a large amount of data as frames can have a large resolution and the video can contain many frames. Therefore, research proposed to use Interest Points detectors which can detect points of interest. By only using optical flow information from these regions, the number of features can be reduced. In [16], many of these interest point detectors are compared to dense sampling on various datasets. It was concluded that dense sampling proves to be better performing and therefore all experiments that are found in this report use dense sampling [15].

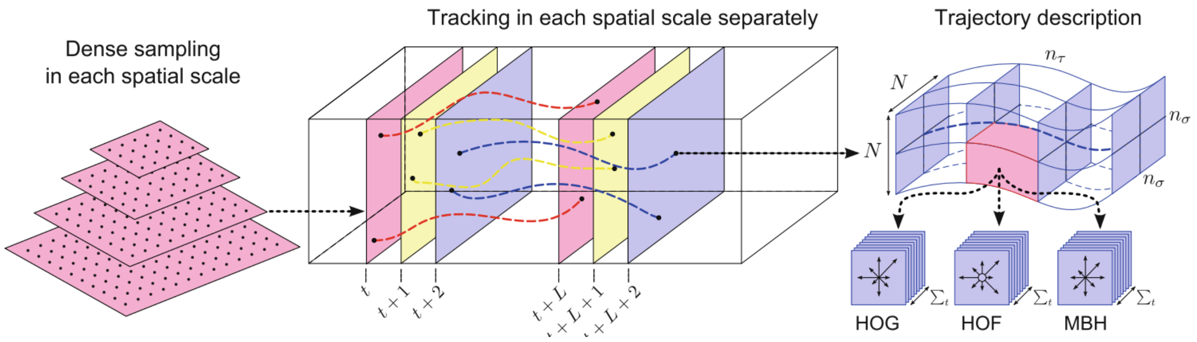


Figure 3.4: Visualization of the implementation of the dense trajectories from [15]

3.2.2. Histograms of Oriented Gradients (HOG)

The Histogram of Oriented Gradients is a video descriptor that represents gradient orientation in a localized portion of an image [2]. The key idea behind HOG is that the objects in localized regions can be represented by a distribution of intensity gradients. It is implemented by first dividing the image into smaller patches or cells. For each patch, we then calculate a local 1-D histogram of gradient directions. The calculation of the gradients is usually done using Sobel operations. Various filters give an approximation to the gradients but in our implementation, we use simple ones as shown in equation 3.2. Both filters are applied in a sliding window across the patch to obtain horizontal and vertical gradients for each pixel. It was shown that these simple filters performed better than more complex filters [2].

$$\begin{bmatrix}-1 & 0 & 1\end{bmatrix} \begin{bmatrix}-1 \\ 0 \\ 1\end{bmatrix} \tag{3.2}$$

Now that both the horizontal and vertical gradients are calculated, we can calculate the magnitude and angle of the gradients. These are shown in equations 3.3 and 3.4 respectively. For the angle of the gradients, we create 8 bins ranging from 0 to 360 degrees. These bins are used to create a histogram. The magnitude represents how much weight is added to the bin. Therefore, sharper edges that have more extreme gradients, add more weight to a bin than small gradients.

$$m = \sqrt{g_x^2 + g_y^2} \tag{3.3}$$

$$\theta = arctan(\frac{g_y}{g_x}) \tag{3.4}$$

As a last step, we need to normalize the values of the bins. We calculate L_2 normalization on the obtained values for each image patch. Figure 3.5 show a visualization of the calculated hog descriptor for each image patch. For our work we calculate HOG on the dense trajectories instead of on image patches [15].

3.2.3. Histogram of Optical Flow (HOF)

Histogram of Optical Flow is another video descriptor that also uses histograms as a feature descriptor. However, HOG focuses on the local shape and appearance of objects within an image. HOF focuses on motion patterns using optical flow [8]. It represents the optical flow orientations of the motion between consecutive frames.

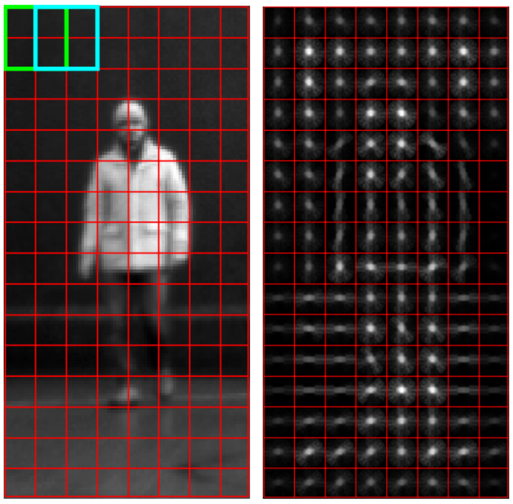


Figure 3.5: Left side: Grayscale test image. Right side: Hog visualization of the test image. [5]

The procedure is the same as for HOG. First, we will calculate the gradients of the optical flow to obtain an angle and a magnitude. This represents the direction of motion as well as the significance of the motion. For HOF we will have one additional bin which brings the total of bins to 9. The additional bin represents pixels for which the optical flow magnitudes are lower than a threshold. Also here L_2 normalization is used to obtain the final values.

3.2.4. Motion Boundary Histograms (MBH)

Optical flow captures all existing motion between consecutive frames. Therefore, it might also capture background motion that is not considered part of the action that we are trying to localize. We want to mitigate the disruptions of these background motions and therefore we can use motion boundary histograms. Motion Boundary Histograms are similar to HOG as it detects edges. HOG does this by looking at changes in pixel intensity but MBH does this by looking at relative motion. First, we calculate the horizontal and vertical optical flow gradients separately. These optical flow fields can be visualized in two new images. Using the procedure of HOG on top of these images gives us the boundaries that are present in the optical flow fields. These boundaries represent the relative motion between consecutive frames. Figure 3.6 shows a visualization of MBHx and MBHy together with optical flow and image gradient for comparison. Both the visualization of optical flow and image gradients captures a lot of background information. This background noise is less present in the visualization of the MBH. For our work, each trajectory contains 96 features for both MBHx and MBHy.

3.3. Dimensionality reduction

All of the described video descriptors are used as features to do action segmentation on a video. Concatenating all features of the video descriptors gives us 426 features in total for each trajectory: dense trajectories (30); Histogram of Oriented Gradients (96); Histograms of Optical Flow (108); Motion Boundary Histograms horizontal (96) and vertical (96). The obtained video information cannot be used directly because of two problems. Firstly, the synthetic dataset already produced around 250,000 trajectories

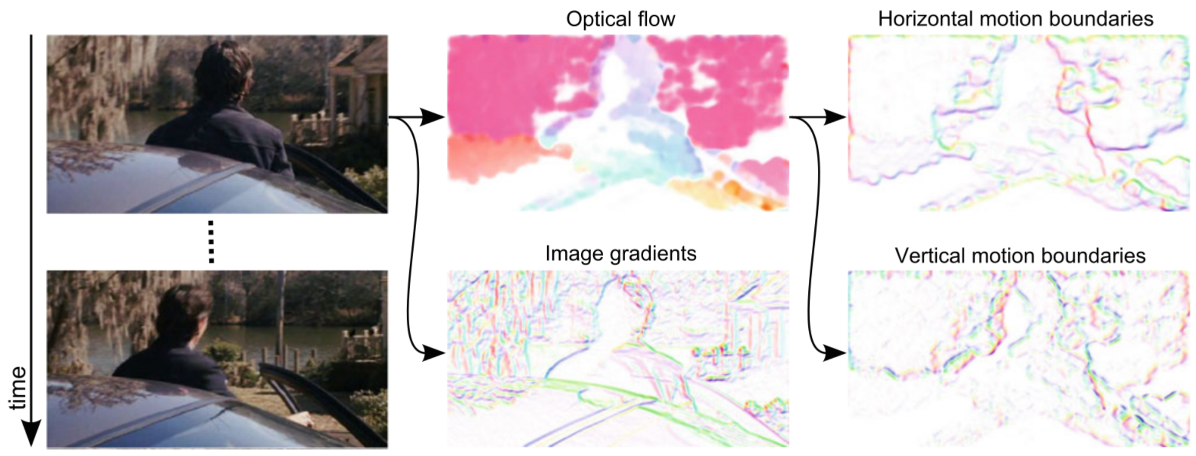


Figure 3.6: Left side: Two consecutive frames from a video. Middle: Visualizations of optical flow and image gradients which contain loads of background noise. Right: Visualizations of horizontal and vertical motion boundaries where background noise is mitigated [15].

for 200 videos which is a lot of data. As the number of features grow, the performance and effectiveness of algorithms might deteriorate. This phenomenon is referred to as the ”curse of dimensionality” [1]. Higher dimensional space grows exponentially as the number of features increase. To cover all that space a lot of data is needed and therefore data might become sparse. We will use dimensionality reduction techniques to reduce the number of dimensions while preserving the underlying information. The second problem is that we want frame-wise feature vectors that all have a constant dimensionality whereas now we only obtained trajectories that span fifteen frames by default. Figure 3.7 shows an overview of the procedure to resolve the problems [7]. First the dimensionality of the trajectory features are reduced from 426 to 64 using Principle Component Analysis (PCA). Then we randomly take 200,000 samples which we fit on a Gaussian mixture model (GMM). Using this GMM we can obtain a fisher vector representations of the data.

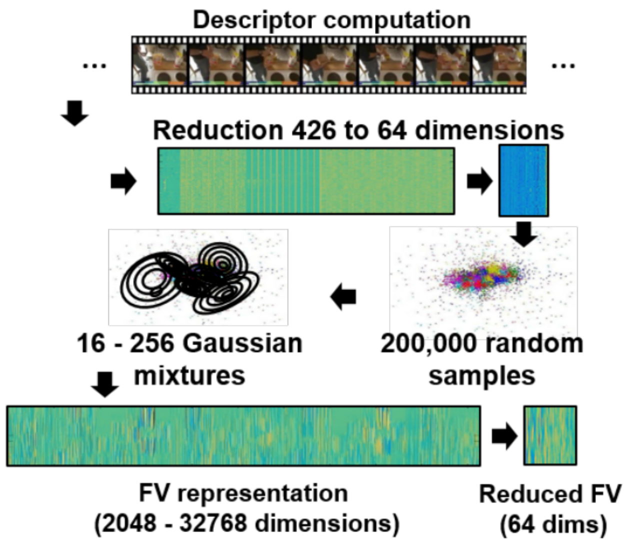
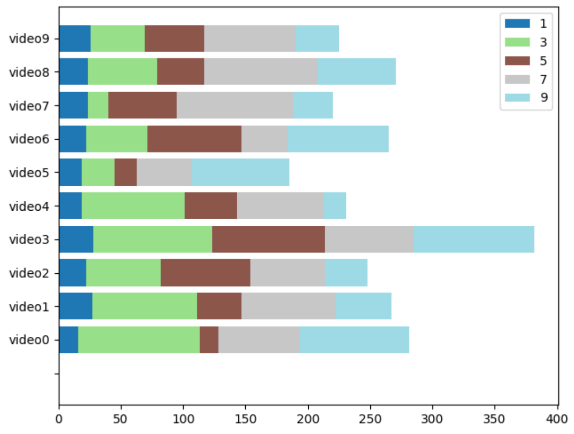
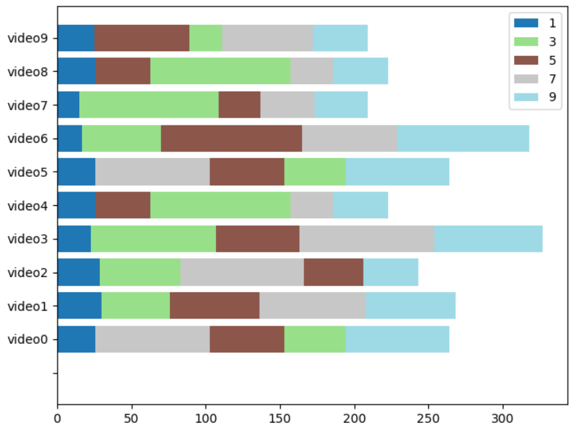

Descriptor computation
Reduction 426 to 64 dimensions
16 - 256 Gaussian mixtures
200,000 random samples
FV representation (2048 - 32768 dimensions)
Reduced FV (64 dims)

Figure 3.7: Pipeline of the dimensionality reduction process taken from [7].

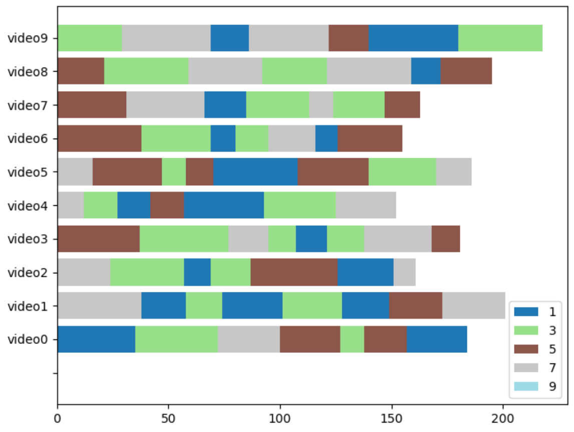
Fisher vectors showed to achieved good performance on Action Recognition [9]. The main idea behind Fisher Vectors is to get a representation of how much a feature deviates from a fitted probability distribution. In our case we have fitted a GMM with K components that gives us mean vectors μ_k and variances σ_k. These values will be used to calculate how much a feature set x deviates from the probability distribution by calculating derivatives to obtain $G^x_{\mu_k}$ and $G^x_{\sigma_k}$. It represents for each component of the GMM how the parameters of the model need to be changed to fit the given feature set x [12]. The dimensionality of $G^x_{\mu_k}$ and $G^x_{\sigma_k}$ depend on the number of components (K) used for the GMM and the dimensionality (D) of the features . In our experiments K and D are both set to 64 and therefore we will obtain a vector of $2 \times 64 \times 64 = 8192$. The feature sets are obtained per frame by applying a sliding window over the video trajectories. If a trajectory overlaps with the sliding window it is included in the feature set. By sliding the window across temporal space we now obtain 8192 feature values for each frame. The length of the sliding window is set to 20. As a last step we will again apply PCA to reduce the dimensionality to 64 and to mitigate the effect of the curse of dimensionality.



(a) *DigitVideos-order*



(b) *DigitVideos-swap*



(c) *DigitVideos-repeat*

Figure 3.8: Example videos of the three different set of synthetic videos.

3.4. Dataset description

3.4.1. Synthetic Data

We created a synthetic dataset to test the models in a controlled environment. It consists of three different versions: *DigitVideos-order*, *DigitVideos-swap*, *DigitVideos-repeat*. Each set of videos has a different complexity of action ordering ranging from less complex to more complex. The videos consist of moving digits that represent unique actions. Furthermore, the moving digits are put on top of video frames from EPIC-KITCHEN [3]. There are a total of five unique actions that are represented by moving digits: 1, 3, 5, 7, 9. These digits are moving horizontally, inverse diagonally, inverse horizontally, diagonally, and vertically respectively. In *DigitVideos-order* all videos have a consistent action ordering and therefore models that use temporal assumptions can perform well. The length of the actions within the video are not consistent and range between 15 and 90 frames. In *DigitVideos-swap* the first and last actions are consistent. The actions in between have a chance of 30% to be swapped and therefore action orderings are less consistent. In *DigitVideos-repeat*, the main challenge is the repetition of actions within one video. The first and last actions are always identical. Furthermore, actions in between are alternated and are allowed to reoccur at most three times. This set of videos is the most challenging as there is no pattern in terms of the temporal positions of the actions. Figure 3.8 shows ten example videos for each set of videos.

3.4.2. Breakfast

The Breakfast dataset [6] is a collection of video with 52 participants that perform ten different cooking activities in the kitchen: cereals, coffee, friedegg, juice, milk, pancake, salat, sandwich, scrambledegg, tea. There are a total of 48 subactions some of which span multiple actions. Figure 3.9 shows all the subactions together with the sample distribution. This dataset is very challenging as videos within a cooking activity do not have a consistent number of unique actions.

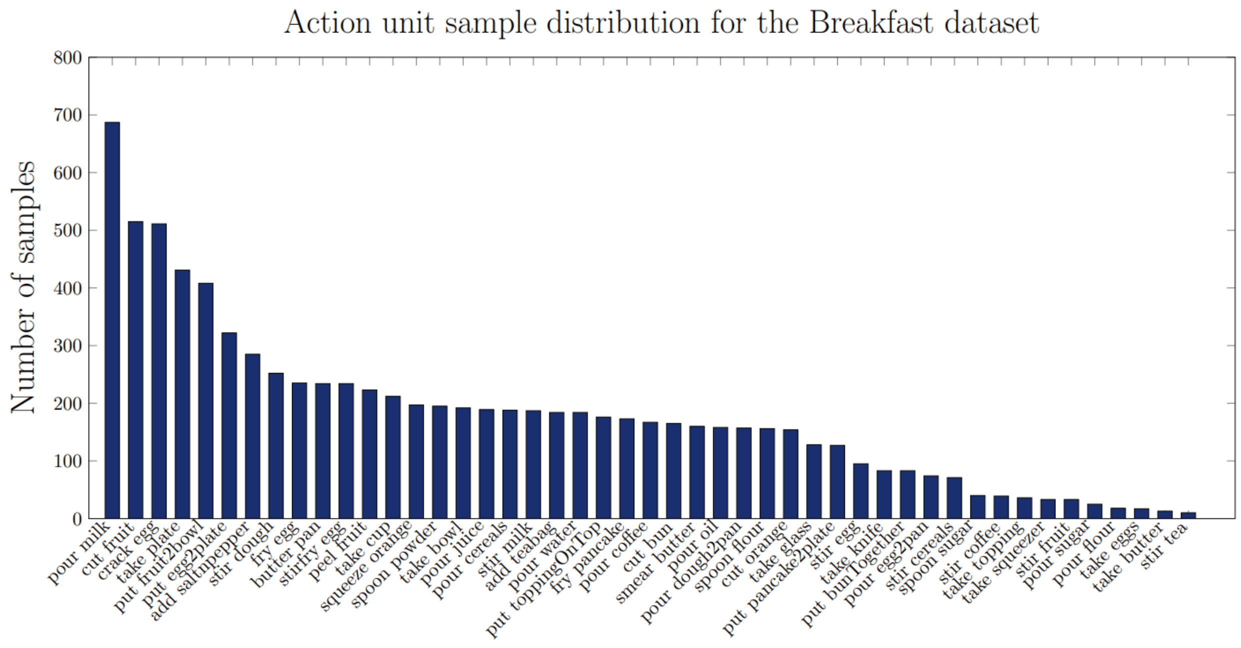


Figure 3.9: Distribution of the subactions from Breakfast taken from [6].

3.4.3. Cholec80

The Cholec80 [14] dataset consists of 80 endoscopic videos from cholecystectomy surgeries performed by 13 different surgeons. All videos contain similar actions and follow a fixed order. These include: preperation, calot triangle dissection, clipping cutting, gallbladder dissection, gallbladder packaging, cleaning coagulation and gallbladder retraction. Out of the 80 videos 15 of them do not include a preparation phase. Therefore we train a model on the set of video that do not contain a preparation phase and a model on the set of videos that do include a preparation phase. Figure 3.10 shows the duration of actions of ten different random videos.

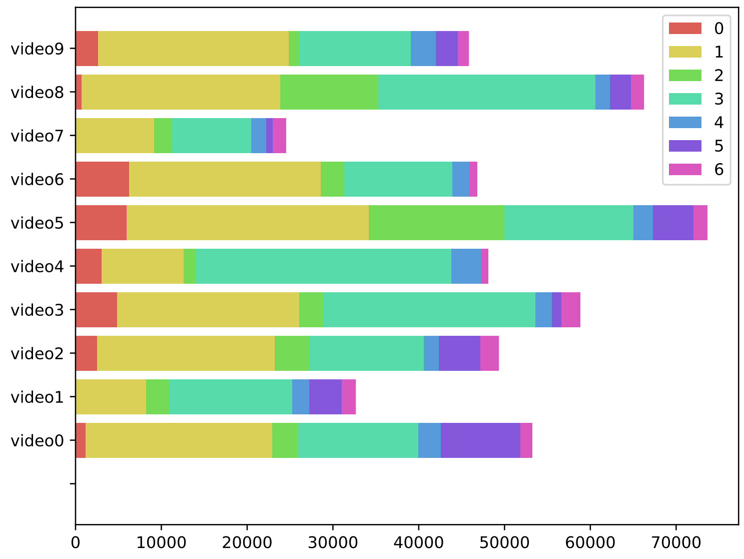


Figure 3.10: Visualization 10 random videos from cholec80. Action ordering is fixed but does not always include a preparation phase. Videos are recorded in 25 fps but are resampled to 0.5 for the experiments.

	DigitVideos-normal		
Method	MoF	F1	IoU
One model	0.727	0.693	0.582
Multiple models	0.553	0.391	0.310

Table 3.1: Results of training a single model over all videos versus training a model for each video.

3.5. Additional results and visualizations

3.5.1. Training on individual videos versus over all videos

There are several reasons why we train the model over all videos. It is more efficient to train a single model instead of a model for each video. Also we have the hypothesis that additional global information can be exploited. This is under the assumption that videos contain similar features and therefore have shared global information. We run experiments on the synthetic dataset to verify whether the hypothesis holds. Table 3.1 shows the results of these experiments. Training a single model over all the videos performs a lot better as training a single model on each video. The single model gets to see a lot more videos and therefore has more information. A single video can still be a bit noisy and therefore training over all videos might act as some form of denoising.

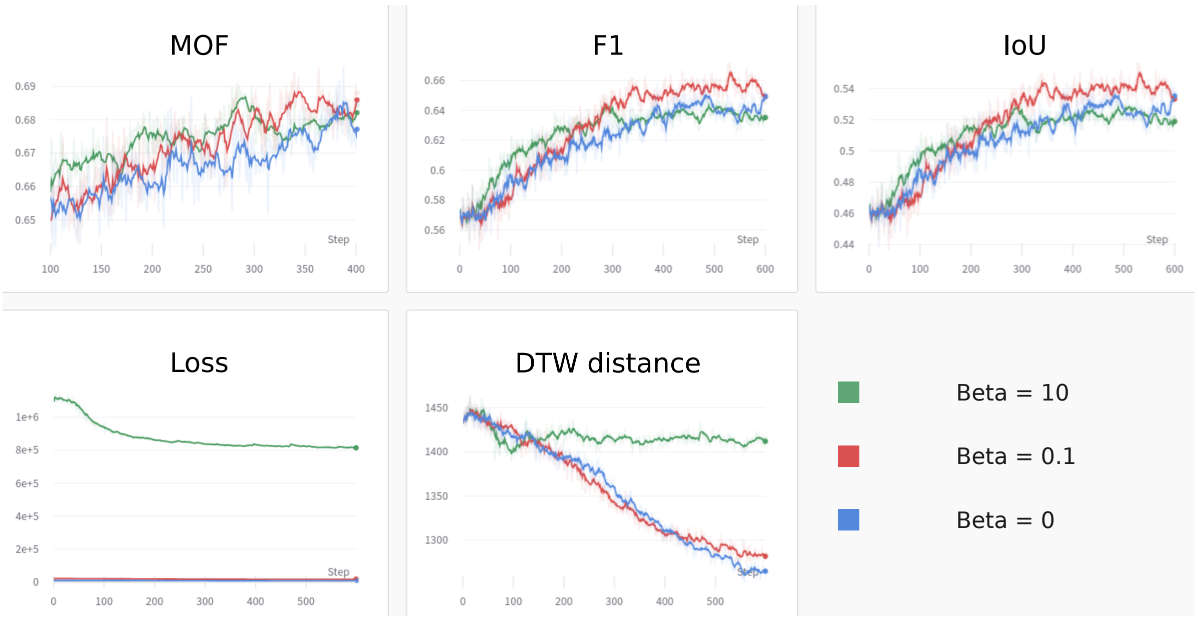


Figure 3.11: Visualization of training three models with different values of β to diversify the prototypes. Only optimizing using DTW distance does not guarantee to lead to the best possible results. Diversifying the prototypes also could help in generating better segments of the video. In this case a balance between DTW distance and diversifying gives us the best performance.

3.5.2. Impact of diversifying the prototypes

For the ablation study we evaluated the model with an additional diversity loss. For experiments three different values of β were used. One model is trained without diversity loss, one with a balance value and one with an extreme weight to diversifying the prototypes. This corresponds to the values: 0, 0.1, 10 respectively. Figure 3.11 shows a visualization of the training stage of the experiments. When β is set too high, the model will only optimize for diversifying the prototypes. This can be seen in the DTW distance graph. The green line initially decreases a little bit but does not continue to decrease just like the other lines. A sensible balance between the diversity loss and the DTW loss, gives us the best performance. This justifies that diversifying the prototypes helps the learning process.

3.5.3. Data visualization using t-SNE

Data visualization is important as it makes it easier to understand the complex data we are working with. One method that is widely used is t-SNE. It is an algorithm that can efficiently solve nonlinear data and can preserve the local and global structure of the data. One interesting visualization is the impact of background noise on the data. Figure 3.12 shows two t-SNE plots of the synthetic data with and without background noise respectively. Here, the colors represent the ground-truth action labels of the features. This helps us to understand whether features of similar action classes can be easily grouped. The visualization of the synthetic data without background noise shows that this is the case. However, when noise is added, the grouped clusters fuse more together and therefore are less easy to separate. This shows that the introduction of background noise was effective. However, it could also be the case that t-SNE is not able to visually group similar actions together because of the limitations by t-SNE.

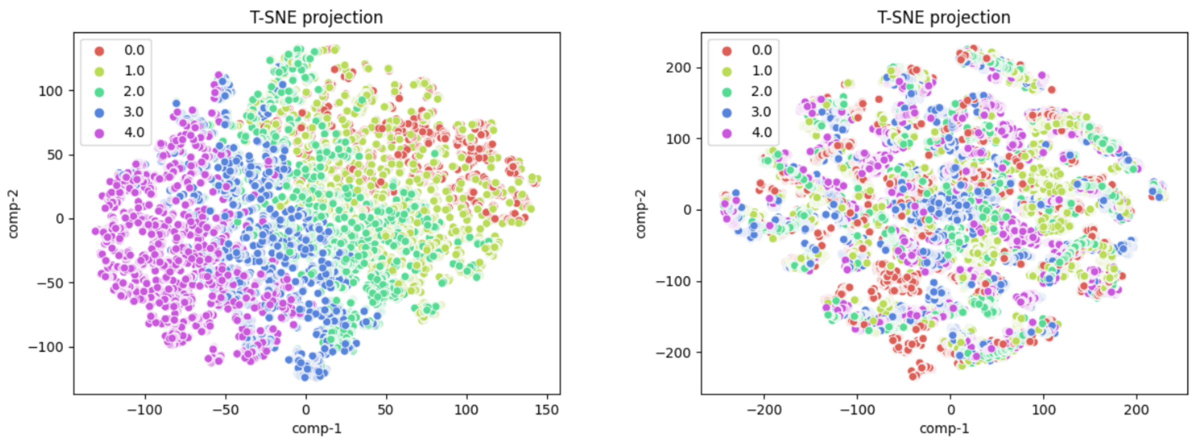


Figure 3.12: Data visualization of the synthetic data with and without background noise. The colors represent the ground-truth action labels. Left: Videos do not contain background noise and therefore the features are much easier to visually cluster. Right: Background noise is added which makes features harder to separate according to the ground-truth labels.

Another interesting visualization is to see whether equivalent reoccurring actions group well together. Figure 3.13 shows a t-SNE plot of a video from *DigitVideos-repeat*. Action 3 occurs twice and the two corresponding segments are close to each other. Action 5 occurs thrice but the corresponding segments are far away from each other. This indicates that repeated actions do not necessarily share the same features and therefore repeated actions might be hard to recognize.

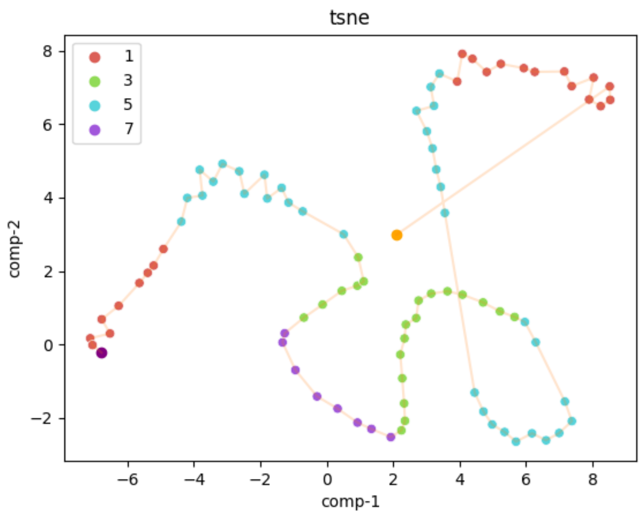


Figure 3.13: t-SNE visualization of the features of a video from *DigitVideos-repeat*. Colors represent different actions. Equivalent repeated actions do not group together and this might indicate that recognizing repeated actions is difficult.

References

[1] Richard Bellman. “Dynamic programming”. In: *Science* 153.3731 (1966), pp. 34–37.

[2] Navneet Dalal and Bill Triggs. “Histograms of oriented gradients for human detection”. In: *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR’05)*. Vol. 1. Ieee. 2005, pp. 886–893.

[3] Dima Damen et al. “Rescaling egocentric vision: Collection, pipeline and challenges for epic-kitchens-100”. In: *International Journal of Computer Vision* (2022), pp. 1–23.

[4] Denis Fortun, Patrick Bouthemy, and Charles Kervrann. “Optical flow modeling and computation: A survey”. In: *Computer Vision and Image Understanding* 134 (2015), pp. 1–21.

[5] Efstathios P Fotiadis, Mario Garzón, and Antonio Barrientos. “Human detection from a mobile robot using fusion of laser and vision information”. In: *Sensors* 13.9 (2013), pp. 11603–11635.

[6] Hilde Kuehne, Ali Arslan, and Thomas 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*. 2014, pp. 780–787.

[7] Hilde Kuehne, Juergen Gall, and Thomas Serre. “An end-to-end generative framework for video segmentation and recognition”. In: *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2016, pp. 1–8.

[8] Ivan Laptev et al. “Learning realistic human actions from movies”. In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2008, pp. 1–8.

[9] Xiaojiang Peng et al. “Action recognition with stacked fisher vectors”. In: *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer. 2014, pp. 581–595.

[10] K. Prazdny. “Motion and Structure from Optical Flow”. In: *Proceedings of the 6th International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI’79. Tokyo, Japan: Morgan Kaufmann Publishers Inc., 1979, pp. 702–704. ISBN: 0934613478.

[11] Hiroaki Sakoe and Seibi Chiba. “Dynamic programming algorithm optimization for spoken word recognition”. In: *IEEE transactions on acoustics, speech, and signal processing* 26.1 (1978), pp. 43–49.

[12] Jorge Sánchez et al. “Image classification with the fisher vector: Theory and practice”. In: *International journal of computer vision* 105 (2013), pp. 222–245.

[13] Romain Tavenard. *An introduction to Dynamic Time Warping*. `https://rtavenar.github.io/blog/dtw.html`. 2021.

[14] Andru P Twinanda et al. “Endonet: a deep architecture for recognition tasks on laparoscopic videos”. In: *IEEE transactions on medical imaging* 36.1 (2016), pp. 86–97.

[15] Heng Wang and Cordelia Schmid. “Action recognition with improved trajectories”. In: *Proceedings of the IEEE international conference on computer vision*. 2013, pp. 3551–3558.

[16] Heng Wang et al. “Evaluation of local spatio-temporal features for action recognition”. In: *Bmvc 2009-british machine vision conference*. BMVA Press. 2009, pp. 124–1.