Signal, Image and Video
- FACT in Motion: Testing TAS on Figure Skating Poses -

Edoardo Fiorentino [248444] Marco Ragusa [258336]

Contents

1 Introduction

Temporal Action Segmentation (TAS) is the task of assigning a class label to each frame in an untrimmed video, aiming to detect the temporal boundaries and categories of consecutive actions. Unlike traditional action recognition, which typically operates on pre-trimmed clips and predicts a single label, TAS must deal with long-range temporal dependencies, action transitions, and different durations. This makes TAS a considerably more complex and expressive problem, with wide applications in video understanding, surveillance, and sports analysis.

In recent years, transformer-based architectures have emerged as a powerful solution for TAS, thanks to their ability to capture global temporal context. In particular, the **Frame-Action Cross-attention Temporal Modelling (FACT)** model introduces a two-branch architecture that jointly processes frame-level and action-level representations, enabling bidirectional information exchange via cross-attention mechanisms.

In this project, we replicate and adapt the FACT model to a simplified scenario based on the FS-Jump3D dataset, which contains 3D pose sequences of figure skating jumps. Our objective is to evaluate how FACT behaves in a context where each video contains a single annotated action segment, without complex transitions or background actions. This setup allows us to observe how the model processes pose-only input and whether its temporal reasoning capabilities remain effective even in a constrained setting.

The idea of applying the FACT architecture to FS-Jump3D was inspired by a previous study[1], which explored this combination as a way to study temporal modeling over 3D pose sequences in figure skating. Starting from that, we implemented FACT from the official CVPR 2024 repository and adapted it to operate on FS-Jump3D using only coarse action labels and a simplified per-video structure.

Our project source code is available on GitHub[2].

2 Background

2.1 Temporal Action Segmentation vs Action Recognition

Temporal Action Segmentation (TAS)[3] differs from action recognition in its objective and complexity. While action recognition assigns a single label to an entire video clip—often under the assumption that the clip contains one dominant action—TAS aims to predict frame-level action labels for untrimmed videos that may contain multiple, consecutive actions.

TAS presents several challenges: it requires the detection of precise action boundaries, handling segments of variable-length, and modeling of both local motion and long-range temporal dependencies. iN addition, frame-based methods often suffer from over-segmentation, where predictions fluctuate frequently near transitions. Transformer-based approaches mitigate this by incorporating global temporal reasoning, though they often do so at high computational cost.

2.2 The FS-Jump3D Dataset

FS-Jump3D[4] is a dataset designed to study figure skating actions using 3D pose data. It contains pose sequences obtained from multiple athletes performing different types of figure skating jumps. Each video corresponds to a single jump trial and includes 3D coordinates for 86 body keypoints, resulting in 258-dimensional feature vectors per frame.

The dataset is structured hierarchically: it includes several athletes, each performing several jump types (such as Axel, Lutz, Flip, and Toe Loop), with each type repeated across multiple trials. Frame-level annotations are available in two formats:

- **Coarse labels:** a single action class indicating the type of jump (e.g., *Lutz*), applied only to the relevant segment.
- **Fine-grained labels:** optional subdivisions of the jump segment into *entry*, *jump*, and *landing* phases.

In this project, we use only the coarse annotations and omit the `None` (background) class, resulting in videos where all frames are assigned to a single action class. This simplification allows us to evaluate the model's behavior in a controlled setting, where each input sequence contains a single temporally bounded action without transitions or ambiguity.

2.3 The FACT Model

The **Frame-Action Cross-attention Temporal Modelling (FACT)** [5] is a model for temporal action segmentation that introduces a two-branch structure to process both detailed frame-level information and higher-level action representations. Unlike traditional approaches that either focus only on frames or use two separate stages, FACT combines both views in a single architecture that is updated step by step.

2.3.1 Dual-Branch Structure

The model is composed of two main branches:

- The **frame branch** takes in features for each frame and updates them using dilated temporal convolutions. These layers help the model understand local and medium-range temporal patterns. This branch also applies positional encodings and produces frame-level action predictions.
- The **action branch** uses a fixed number of learnable *action tokens*. Each token is designed to represent an action segment. These tokens are updated through transformer layers and can learn to focus on different parts of the video, depending on their interaction with the frame branch.

2.3.2 Cross-Attention and Feature Refinement

A key element of FACT is the use of **bidirectional cross-attention** to allow the two branches to communicate:

- In the **frame-to-action (F→A)** direction, the tokens focus on the frame features to learn which parts of the video they are responsible for.
- In the **action-to-frame (A→F)** direction, the frame features recive information from the tokens, allowing each frame to include global action-level context.

This interaction happens repeatedly across several update blocks, which gradually improve both the frame features and the token representations. Over time, the model learns to align each token with the correct temporal segment in the video.

2.3.3 Token Matching During Training

During training, the model receives a fixed number of tokens (e.g., $M = 10$). Only some of these tokens are matched to actual action segments; the rest may be unused. FACT supports two types of matching:

- **One-to-one matching**, where each token is linked to a single action segment. This is the setting used in our experiment.

- **One-to-many matching**, where one token can represent several repeated segments of the same class.

The token-to-segment assignments are computed using a similarity score that combines the predicted class and the overlap between the token's attention and the ground-truth segment. The matching is usually computed with the Hungarian algorithm.

2.3.4 Prediction and Fusion

At the final step of the model, both branches produce predictions:

- The frame branch gives a probability distribution over classes for each frame.
- The action branch predicts a class label for each token. These predictions are then spread across the frames using the attention weights learned during training.

To get the final prediction for each frame, the outputs of the two branches are combined using a weighted average:

$$P = w \cdot P^{\text{action}} + (1 - w) \cdot P^{\text{frame}}$$

where w is a parameter chosen based on validation performance. This fusion helps the model benefit from both local and global information.

2.3.5 Losses and Optimization

During training, FACT uses a combination of different loss functions to guide learning:

- A classification loss on each frame (cross-entropy),
- A loss for the action tokens, which helps them predict the correct class,
- A cross-attention loss, which encourages tokens to focus on the correct frames,
- A temporal smoothing loss, which reduces sudden changes in the predicted labels over time.

These losses are applied at every block of the model, allowing the network to improve its predictions and token assignments progressively throughout the layers.

3 Experimental Setup

3.1 Project Goal

The main goal of this project was to replicate the FACT[5] model on a new domain: figure skating action data represented as 3D pose sequences (FS-Jump3D). We aimed to test the behavior of the model in a simplified setting, using one action per video and pose-only input. This allowed us to focus on the model's ability to localize and classify a temporally bounded action without distractions from RGB or multimodal data.

3.2 Dataset Preprocessing

To prepare the FS-Jump3D dataset for training with the FACT model, we used a preprocessing script named `format.py`, adapted from the official dataset repository[4]. The dataset contains raw JSON files with 3D marker coordinates for different types of figure skating jumps performed by four athletes.

The script processes each file as follows. First, it loads the full list of markers and identifies the main temporal interval where all relevant markers are active. This step ensures that only the central part of the jump is extracted, removing unstable regions at the start or end of the sequence.

Next, the raw marker data are converted into a standardized joint-based representation. This is done using a rig mapping file that defines how groups of markers correspond to 17 anatomical joints. For each joint, the 3D coordinates are computed as the average of the associated markers. As a result, each frame is represented as a pose of shape $(17, 3)$, which is then flattened to a 51-dimensional vector.

The label for each sequence is inferred from the folder name in which the JSON file is stored (e.g., `Axel`, `Flip`). This label is assigned to all frames of the sequence, resulting in a uniform per-frame annotation. No background or "None" label is used in this setup.

After formatting, the following outputs are generated:

- A feature file in `.npy` format containing a $(T, 51)$ array of poses,
- A label file in `.txt` format with T action labels,
- A mapping file linking action names to numeric IDs,
- Three text files listing the samples in the training, validation, and test splits (70%, 15%, 15% respectively).

This preprocessing pipeline ensures that each jump trial is converted into a clean, standardized input format suitable for direct use with the FACT model.

We used the official FACT implementation as a submodule, applying changes to adapt it to our dataset and processing pipeline. Most of the model architecture was left unchanged. The input to the model consisted of 3D pose sequences with 51 features per frame, as obtained from the preprocessing step.

To configure the model for our dataset, we created a custom configuration file named `fsjump.yaml`, which defines all the necessary parameters for data loading, model structure, number of classes, number of action tokens, and training settings. This configuration file allowed us to run training and evaluation without modifying the base code of the model.

The number of action tokens was set to 10, which is more than sufficient given that each video contains only one action segment. We used one-to-one matching between the ground-truth segment and one of the action tokens. Since all frames in the sequence are labeled with the same action class, there was no need to supervise the remaining tokens using a null class. These unassigned tokens were simply left without supervision.

The frame branch of the model was configured to use dilated temporal convolutions, as in the original FACT design. The action branch relied on transformer layers to update token representations and to perform cross-attention with the frame features. Positional encodings were applied to both frames and tokens to preserve temporal structure and token identity.

No architectural modifications were made to the core update blocks, prediction logic, or loss computation modules.

3.3 Training and Evaluation

The model was trained using the standard FACT pipeline, adapted to the FS-Jump3D dataset through a dedicated configuration file (`fsjump.yaml`). Each input sequence consisted of 51-dimensional pose vectors extracted from preprocessed 3D marker data, with all frames labeled using the corresponding jump type.

We trained the model using the default loss structure of FACT, which combines framewise classification, token-level classification, attention alignment, and temporal smoothing. These losses are applied at every block of the model to allow progressive refinement of features and predictions.

A key objective of the training phase was to assess how the number of action tokens influences the model's performance in a simplified scenario, where each video contains only a single action and no background class. To this end, we conducted multiple training runs, varying the number of tokens

across the values $M = 10, 30, 50$, and 100. The rest of the training setup—including optimizer, learning rate, and number of update blocks—was kept constant to isolate the effect of token count.

The evaluation phase was based on standard metrics used in temporal action segmentation: accuracy (**Acc**), and frame-wise F1 scores at different temporal IoU thresholds (**F1@10**, **F1@25**, **F1@50**). These metrics allow us to capture both the per-frame correctness and the temporal consistency of the predictions.

The results obtained for each token configuration are presented and discussed in the next section, with particular attention to the relationship between token count and segmentation quality.

4 Results and Discussion

We evaluated the model's performance using several standard metrics for temporal action segmentation: framewise accuracy over action segments (Acc), edit score, and F1 scores at different intersection-over-union (IoU) thresholds (F1@10, F1@25, F1@50). All values are reported as percentages.

To assess the impact of the number of action tokens, we trained the model with different values of M ($M = 10, 30, 50, 100$). The results are reported in Table 1.

Tokens (M)	**Acc**	**Edit**	**F1@10**	**F1@25**	**F1@50**
10	17.93	19.35	17.14	17.14	17.14
30	17.93	19.35	19.35	19.35	19.35
50	**21.41**	**22.58**	**22.58**	**22.58**	**22.58**
100	17.93	19.35	19.35	19.35	19.35

Table 1: Performance metrics for different numbers of action tokens M.

The results show that increasing the number of tokens from 10 to 50 leads to a steady improvement across all metrics. The configuration with $M = 50$ achieves the highest values in framewise accuracy, edit score, and all F1 scores. This suggests that using a moderate number of tokens allows the model to better capture the temporal structure of the action, even in the simplified setting where each video contains only a single segment.

Interestingly, increasing the number of tokens further to $M = 100$ does not yield additional improvements and actually results in a slight drop in performance, bringing all metrics back to the levels observed at $M = 30$. This may indicate that too many tokens introduce redundancy or make the matching process less effective during training. Overall, the results highlight the importance of tuning the token count, even in constrained setups.

5 Conclusion

The results obtained in our experiments show relatively low accuracy and segmentation scores across all token configurations. This outcome is not entirely unexpected, considering the limitations of our setup. Unlike the original FACT training setting, which includes large-scale datasets with multiple actions per video, our training was performed using only FS-Jump3D, a smaller dataset where each video contains a single action. Moreover, the original study that inspired our work made use of multiple datasets to train the model, while our implementation relies solely on this one source of data. This likely limited the variety and generalizability of the patterns the model was able to learn.

Despite these constraints, we consider this project a meaningful step toward adapting a complex model like FACT to a more minimal and controlled setting. We were able to successfully configure the pipeline, from data preprocessing to evaluation, and observe the effect of increasing the number of action tokens on model performance. Even though the results are modest, they suggest that FACT maintains some capacity to learn temporal structure even when trained on short, single-action

sequences with pose-only input. The simplified structure of our setup also makes it easier to isolate model behavior and could serve as a useful testbed for further experimentation, such as attention analysis or multi-segment adaptation in future work.

References

[1] Ryota Tanaka, Tomohiro Suzuki, and Keisuke Fujii. "3D Pose-Based Temporal Action Segmentation for Figure Skating: A Fine-Grained and Jump Procedure-Aware Annotation Approach". In: *Proceedings of the 7th ACM International Workshop on Multimedia Content Analysis in Sports*. MM '24. ACM, Oct. 2024, pp. 17–26. DOI: 10.1145/3689061.3689077. URL: http://dx.doi.org/10.1145/3689061.3689077.

[2] *Project Repository*. URL: https://github.com/marcorags/SIV_UniTN_TAS_project.

[3] *TAS*. URL: https://github.com/nus-cvml/awesome-temporal-action-segmentation.

[4] *FS-Jump3D dataset*. URL: https://github.com/ryota-skating/FS-Jump3D.

[5] *FACT*. URL: https://github.com/ZijiaLewisLu/CVPR2024-FACT.