Encoder-Agnostic Learned Temporal Matching for Video Classification

By

Darryl Ho

S.B., Massachusetts Institute of Technology (2021)

Submitted to the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2025

Authored by: Darryl Ho
Department of Electrical Engineering and Computer Science
January 24, 2025

Certified by: Samuel R. Madden
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by: Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Encoder-Agnostic Learned Temporal Matching for Video Classification

by

Darryl Ho

Submitted to the Department of Electrical Engineering and Computer Science on January 24, 2025 in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

ABSTRACT

In recent years, large transformer-based video encoder models have greatly advanced state-of-the-art performance on video classification tasks. However, these large models typically process videos by averaging embedding outputs from multiple clips over time to produce fixed-length representations. This approach fails to account for a variety of time-related features, such as variable video durations, chronological order of events, and temporal variance in feature significance. While methods for temporal modeling do exist, they often require significant architectural changes and expensive retraining, making them impractical for off-the-shelf, fine-tuned large encoders. To overcome these limitations, we propose DejaVid, an encoder-agnostic method that enhances model performance without the need for retraining or altering the architecture. Our framework converts a video into a variable-length temporal sequence of embeddings, which we call a multivariate time series (MTS). An MTS naturally preserves temporal order and accommodates variable video durations. We then learn per-timestep, per-feature weights over the encoded MTS frames, allowing us to account for variations in feature importance over time. We introduce a new neural network architecture inspired by traditional time series alignment algorithms for this learning task. Our evaluation demonstrates that DejaVid substantially improves the performance of a state-of-the-art large encoder, achieving leading Top-1 accuracy of 77.2% on Something-Something V2, 89.1% on Kinetics-400, and 88.6% on HMDB51, while adding fewer than 1.8% additional learnable parameters and requiring less than 3 hours of training time.

Thesis supervisor: Samuel R. Madden
Title: Professor of Electrical Engineering and Computer Science

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Video action recognition has seen remarkable progress with the rise of transformer-based video models [1–3]. To improve their performance, one of the most straightforward and effective approaches is to upscale the model and training data [4]. As such, the leaderboards of major action recognition benchmarks have come to be dominated by 100M–1B+ parameter large video transformers [5–11].

These large encoders process a video with a fixed frame count, height, and width, producing a single fixed-length embedding. When applying these encoders on video datasets with varying shapes, the common approach is to pass multiple temporal clips and spatial crops of each video through the encoder to generate several logit vectors, which are then averaged for class prediction [8–11]. However, while the height and width of videos in a dataset are usually constant, their duration can vary significantly [5]. Another important distinction between the temporal and spatial dimensions is that the order of information is much more critical temporally. For instance, it is common practice to flip images when training vision models [12], but reversing time often changes the semantic meaning of an action, such as distinguishing between opening and closing a door. When averaging logits from different temporal clips, sequential information in these clips is lost. Moreover, the significance of each feature within a video can change over time. For example, in a clip of a basketball player

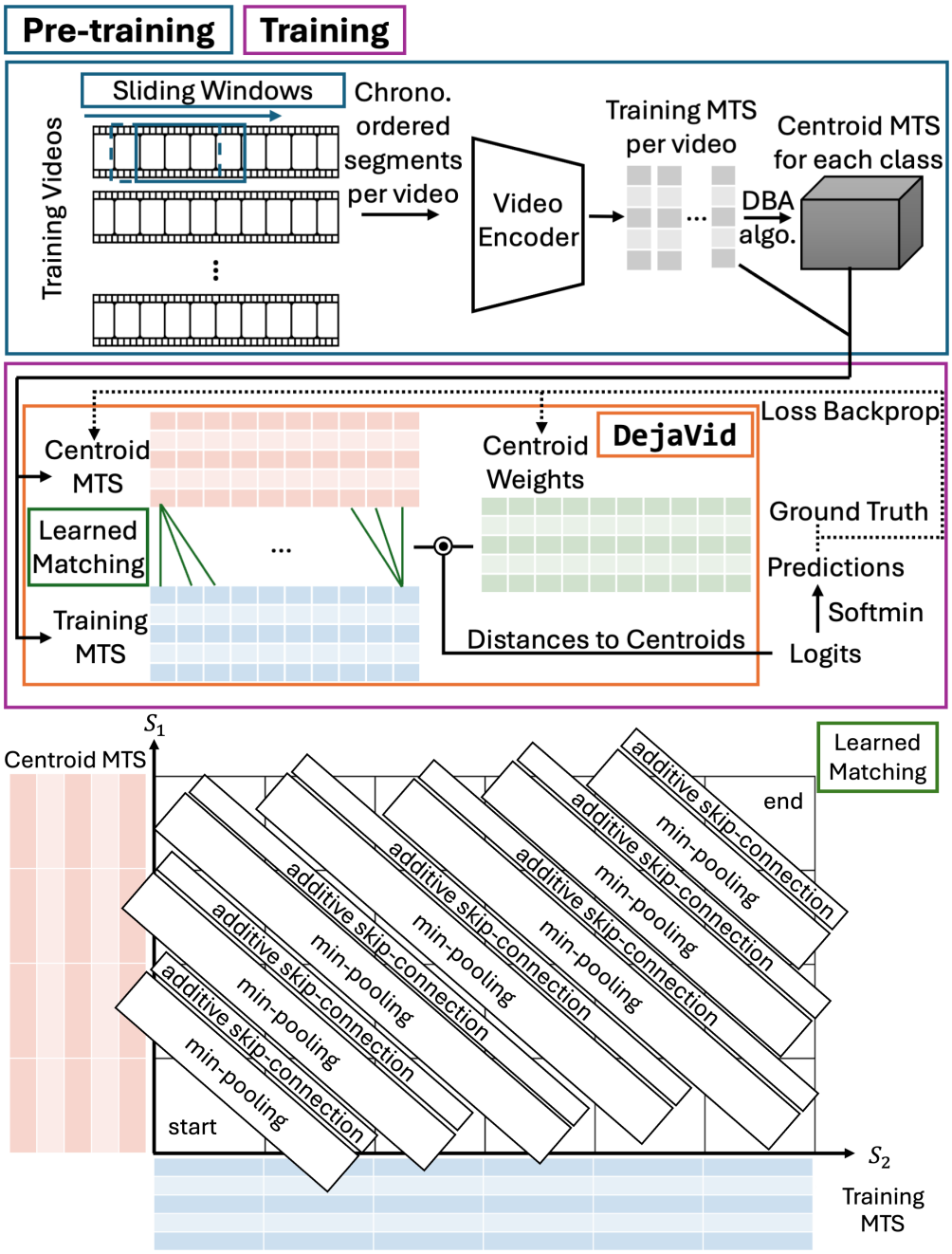

Figure 1.1: DejaVid system diagram.
We first transform a video into a variable-length temporal sequence of embeddings, which we call a multivariate time series (MTS). We then learn per-timestep, per-feature weights for the encoded MTS frames, enabling us to accommodate variations in feature importance over time, as well as a centroid MTS for each output class. For this task, we introduce a new neural network architecture inspired by traditional time series alignment algorithms. In testing, we freeze the post-training centroids and weights and use the same pre-training pipeline to generate MTSes to feed into DejaVid for predictions.

making a three-point shot, key features at the beginning might include whether the player is holding the ball or if their feet are touching the three-point line, while by the end, the most relevant feature could be whether the ball goes through the hoop. Although encoders might not explicitly model semantically interpretable features like these, temporal variation in feature importance can also occur for arbitrary features, which mean-pooling fails to capture adequately. This motivates a key observation of our work: by failing to carefully account for temporal dimensions, state-of-the-art large models leave significant accuracy gains on the table.

Improving the temporal performance of large encoders is not trivial. Existing temporal-focused methods require invasive changes to the model, such as inserting temporal-specific layers between transformer blocks [13, 14] or creating model duplications to focus on different input views [15, 16]. These approaches are costly to apply to off-the-shelf pre-trained encoders because they require extensive changes in implementation and costly re-training. These insights motivate DejaVid, a lightweight encoder-agnostic temporal approach that substantially boosts model classification performance. The simple yet crucial idea behind DejaVid is that a video is more effectively represented as a temporal sequence of embeddings, or a *multivariate time series* (MTS), rather than as a single embedding. This representation naturally preserves chronological order while supporting varying time lengths, addressing the aforementioned shortcomings. DejaVid is naturally able to use any pre-trained model by consuming a sliding-window of encoded video to generate an MTS, without requiring changes to the model's architecture, retraining, or fine-tuning. DejaVid is inspired by the well-known Dynamic Time Warping (DTW) algorithm [17], which dynamically aligns similar data points between two time series while maintaining their chronological order. In particular, DejaVid introduces a novel neural network architecture inspired by DTW but with learned per-timestep, per-feature weights to handle the temporal variance of feature importance. Additionally, we develop a custom high-performance CUDA kernel for efficient computation and backpropagation of the (time-weighted) warping distance that outperforms a naive

DTW-based implementation by 2 orders of magnitude.

We evaluate DejaVid by layering it on top of the largest (ViT-g) variant of VideoMAE V2 [8] (abbr. VideoMAE V2-g) as the backbone encoder to evaluate DejaVid on three popular action recognition datasets: Kinetics-400 [6], Something-Something V2 [5], and HMDB51 [7]. We show that DejaVid significantly improves on the strongest VideoMAE V2-g baseline, thereby achieving new state-of-the-art performance. In particular, DejaVid achieves Top-1 accuracy of 77.2% on Something-Something V2, 89.1% on Kinetics-400, and 88.6% on HMDB51. We also show that our proposed modifications to DTW, including the temporal weights, can further improve prediction performance.

Chapter 2

Related Work

State-of-the-Art in Action Recognition. Major action recognition benchmarks are topped by massive video transformers with hundreds of millions or billions of parameters. As of the time of writing, according to Papers with Code, the top performers on Kinetics-400 and Something-Something V2 with verifiable parameter counts have between 600M and 6B parameters [5, 6, 8–11]. However, their evaluations on these two datasets all apply spatial and temporal mean-pooling to the logits at the end, which suffers from the time-related shortfalls related to variable video durations, chronological order of events, and temporal variance in feature significance, as described in Chapter 1.

Temporal-focused Methods for Action Recognition. There have been various works on improving the temporal modeling capability of video transformers. One general approach is to introduce temporal-specific layers in between transformer blocks. For example, ILA employs a mask-based layer that predicts interactive points between frame pairs to implicitly align features [13], ATM applies basic arithmetic operations to pairs-of-frame features between ViT blocks [14], and TAM inserts an attention-based temporal adaptive module between 2D convolutions within a ResNet block [18]. Another line of work is to fuse multiple temporal views in different ways. For example, SlowFast networks feed two different frame rates of a video into separate encoders and add lateral connections between them to fuse information

across views [16], and MTV deploys a similar idea but for both spatial and temporal dimensions [15]. However, their performance is not competitive against the aforementioned large state-of-the-art models, partly due to their smaller size. Implementing these methods within the large models would involve considerable implementation changes and expensive retraining.

Learning Temporal Sequence Alignments. DTW is one of the most classic methods for calculating similarities between univariate time series or multivariate ones [17, 19]. Its traditional formulation is a dynamic programming scheme, but recent work has explored making it differentiable and then applying it to various fields. Soft-DTW [20] shows that DTW makes a good loss function when the min operator is replaced with soft-min. DTWnet [21] proposes using DTW learning for time series feature extraction, and D3TW, OTAM, and FTCL [22–24] similarly do this for weakly-supervised and few-shot video understanding. While these works have some similarities to our DTW-based temporal representation learning, our method is technically different and superior in three ways:

1. DejaVid models and learns the temporal variance in feature importance, which none of the other methods do.

2. We reformulate our DTW-inspired temporal learning algorithm as a new neural network architecture that boosts parallelizablity and facilitates differentiability.

3. We introduce changes to DTW to improve model stability, including removing the diagonal transition of DTW.

Chapter 3

Method

As described in Chapter 1, we represent a video as a temporal sequence of embeddings, or a multivariate time series (MTS). In this section, we assume all videos have been converted into MTSes, and operate in this MTS space. DejaVid feeds a video encoder a sliding-window stream of frames of an input video to generate an MTS, so the shape of an MTS is $T \times N_f$, where each index along T represents the encoding of a window of frames, and thus T can vary depending on individual video duration. N_f is the size of the embedding vector of the backbone encoder and is constant.

Our problem is to classify MTSes. Specifically, we are given a training set of MTSes with class labels among N_c classes, and our goal is to predict the classes of our validation MTSes. We solve this problem in three broad steps:

1. For each class, we calculate a centroid MTS from the training MTSes of that class.
2. We optimize the centroid MTSes according to some loss function w.r.t. the training MTSes.
3. For each validation MTS, we calculate which class's centroid MTS is the most similar to it.

Here, Steps 1 and 3 require a definition of distance (or similarity) between MTS pairs, and

Step 2 requires the distance metric to be backpropagateable. The following Secs. 3.1 and 3.2 describe how we solve these two problems.

3.1 Definitions

There are multiple ways in which one can define the distance between two MTSes [17, 25], but in this paper, we focus on a standard time series distance metric: Dynamic Time Warping (DTW), as given in Algorithm 1. DTW aligns two sequences that vary in time or speed by minimizing the distance between corresponding points. The two sequences are aligned by aligning each timestep in the first sequence to one or more consecutive timesteps in the second sequence while maintaining a monotonic relationship where a later timestep in the first must map to a later interval in the second. Monotonicity is maintained in Line 5 of Algorithm 1, where each step only moves forward in time for either sequence (from $i-1$ to i or $j-1$ to j). This alignment minimizes cumulative distance and is commonly visualized as a "warping path" on a 2D grid, which shows the optimal alignment of the two sequences, each depicted in one grid axis.

Algorithm 1 The Standard DTW Algorithm (without diagonal transition)

Input: Two MTSes $a \in \mathbb{R}^{n \times N_f}, b \in \mathbb{R}^{m \times N_f}$; a distance function $dist$
Output: Distance between a and b

1: $D \leftarrow \text{array}[n, m]$ {Out-of-bounds access returns $+\infty$}
2: $D_{0,0} \leftarrow dist(a_0, b_0)$
3: **for** i in $[0, n-1]$ **do**
4: **for** j in ($[0, m-1]$ if $i > 0$ else $[1, m-1]$) **do**
5: $D_{i,j} \leftarrow dist(a_i, b_j) + \min(D_{i-1,j}, D_{i,j-1})$
6: **end for**
7: **end for**
8: **return** $D_{n-1,m-1}$

In Algorithm 1, we use the Manhattan distance for the pointwise distance function $dist$. Also, although the original DTW algorithm includes a diagonal transition $D_{i-1,j-1}$ in the min clause, we omit this in DejaVid. We discuss this decision in more detail in Sec. 3.4.

Let $D_v(a, b)$ denote the DTW distance between MTSes a and b as returned by the standard Algorithm 1, and let $C \in \mathbb{R}^{N_c \times T_c \times N_f}$ represent the centroid MTSes for the N_c classes. Intuitively, we want to assign each MTS to the class it is nearest to in the DTW space. To do this, we define the predicted probability distribution $\hat{\mathbf{y}}_v$ for a training or validation MTS $m \in \mathbb{R}^{T_m \times N_f}$[1]and the loss $\mathcal{L}_v$ for m under centroids C and one-hot ground truth vector $\mathbf{y}$,:

$$\hat{\mathbf{y}}_v(C, m) := \underset{0 \le i < N_c}{\text{softmin}}(D_v(C_i, m))$$

$$\mathcal{L}_v(C, m) := \text{CrossEntropy}(\mathbf{y}, \hat{\mathbf{y}}_v(C, m))$$

However, as described in Chapter 1, we observe that the importance of each feature of a video can vary as time progresses. This observation motivates our time-weighted DTW distance metric, given in Algorithm 2 with modifications shown in red.

Algorithm 2 The Time-Weighted DTW Algorithm (without diagonal transition) (difference against Algorithm 1 in red)

Input: Two MTSes $a \in \mathbb{R}^{n \times N_f}, b \in \mathbb{R}^{m \times N_f}$; a distance function $dist_w$; $u \in \mathbb{R}_{>0}^{n \times N_f}$ temporal feature weights of a
Output: Distance between a and b

1: $D \leftarrow \text{array}[n, m]$ {Out-of-bounds access returns $+\infty$}
2: $D_{0,0} \leftarrow dist_w(u_0, a_0, b_0)$
3: **for** i in $[0, n-1]$ **do**
4: **for** j in ($[0, m-1]$ if $i > 0$ else $[1, m-1]$) **do**
5: $D_{i,j} \leftarrow dist_w(u_i, a_i, b_j) + \min(D_{i-1,j}, D_{i,j-1})$
6: **end for**
7: **end for**
8: **return** $D_{n-1,m-1}$

The difference is the additional weight input u, which denotes the change in feature importance over time for a. We also modify the pointwise Manhattan distance function to incorporate u: $dist_w(u_i, a_i, b_j) := \sum_{k=0}^{N_f - 1} u_{i,k} |a_{i,k} - b_{j,k}|$. The changes are colored in Lines 2 and 5 of Algorithm 2. Intuitively, when Algorithm 2 calculates the distance between two sequences, our time-weighted modification allows it to weight some timesteps more than

[1]In practice, we perform a LayerNorm operation before the soft-min, but this doesn't qualitatively change our approach.

some other timesteps in the first series, and different features can have different weights as well. Let $D_w(u, a, b)$ denote the time-weighted DTW distance between MTSes a and b as returned by the time-weighted Algorithm 2. We aim to learn the temporal variance of feature importance of the centroids C, so we define $U \in \mathbb{R}_{>0}^{N_c \times T_c \times N_f}$ of the same shape as C to be the time-varying feature weights. To maintain the positivity of distances, we constrain U to be positive. The time-weighted variant of predicted probability distribution $\hat{\mathbf{y}}_w$ and loss $\mathcal{L}_w$ are defined similarly as before[1]:

$$\hat{\mathbf{y}}_w(U, C, m) := \underset{0 \le i < N_c}{\text{softmin}}(D_w(U_i, C_i, m))$$

$$\mathcal{L}_w(u, C, m) := \text{CrossEntropy}(\mathbf{y}, \hat{\mathbf{y}}_w(u, C, m))$$

From now on, we focus on the time-weighted Algorithm 2, since one can set $U = \mathbf{1}_{N_c \times T_c \times N_f}$ to recover Algorithm 1.

3.2 Parallelizability & Differentiability

Algorithm 4 given an equivalent neural-network formulation for Algorithm 2; this formulation facilitates the use of backpropagation for optimizing weights and enhances parallelizability. Before providing the details of the algorithm, we provide some intuition as to how and why it works.

Fig. 3.1 depicts the 2D grid commonly used for representing DTW warping path(s) between two sequences, as descrbed in Sec. 3.1. We visualize each operation in the nested loop in Algorithm 2 onto the corresponding cell in the grid. We can see that each cell executes a min operation on the outputs of its left-adjacent and bottom-adjacent cells, followed by an addition of some pointwise weighted distance $dist_w$. Note that cells are executed in a row-major order in Algorithm 2. Therefore, the critical path length is quadratic ($O(nm)$).

However, we leverage a critical property of Algorithm 2 that enables a reformulation of

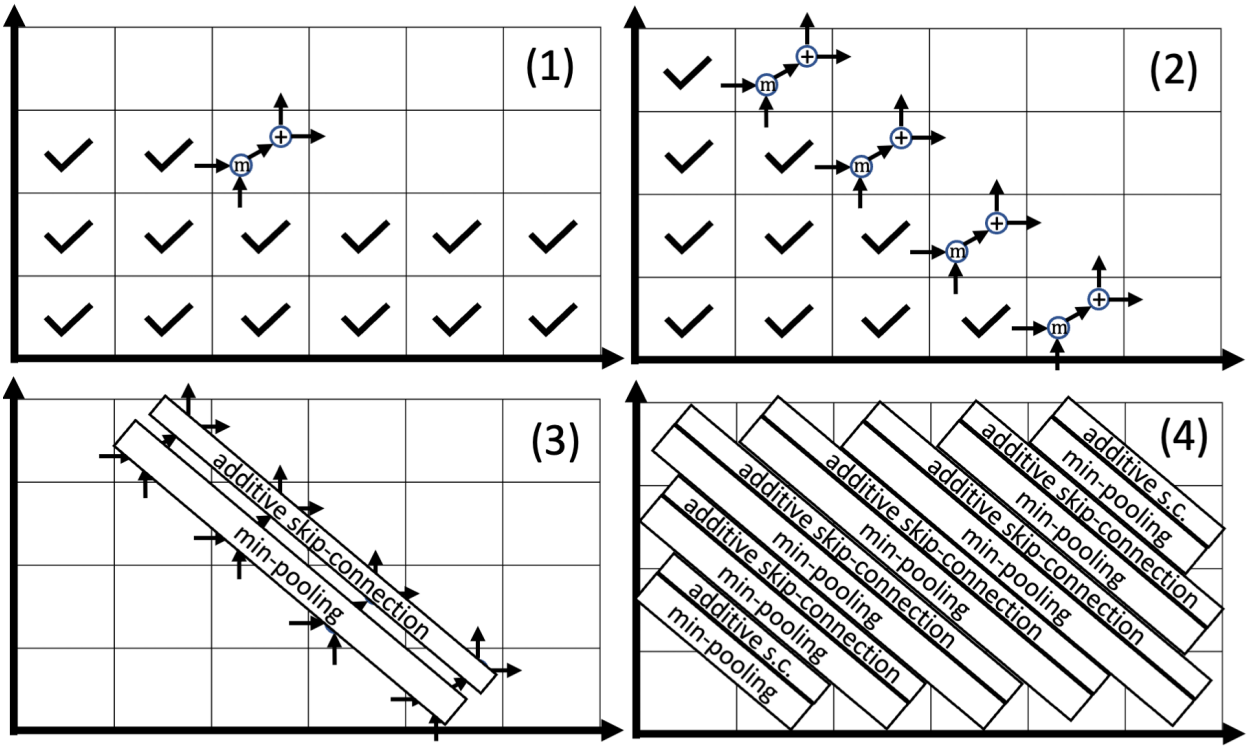

Figure 3.1: Illustration on DTW parallizability.
1. Each cell of the DTW 2D grid executes a *min* operation and then an *add* operation. Checkmarks show completed cells in row-major order. 2. In diagonal order, a diagonal can be executed as soon as the previous diagonal has finished. Checkmarks show completed cells in diagonal order. 3. We can view each diagonal as a min-pooling layer with kernel size 2 and stride 1, followed by an additive skip-connection layer. 4. We reformulate the algorithm into a serial stack of min-pooling layers interlaced with additive skip-connections.

the algorithm, which is that **each cell can be executed as soon as its left and bottom neighbors have finished.** Therefore, we reformulate Algorithm 2 to execute in **diagonal order** and observe that **the l-th diagonal can be executed as soon as the $(l-1)$-th diagonal has finished.** Here, we define the l-th diagonal to be the cells $\{(i,j) \mid i+j=l\}$. For example, the top-right of Fig. 3.1 shows the 4th diagonal. Algorithm 3 realizes the change in execution order from Algorithm 2.

Because each cell executes at Line 7 of Algorithm 3 a min operation on two consecutive cells of the previous diagonal and then adds some $dist_w$ term, we can express Algorithm 3 as a neural net where each diagonal is a min-pooling layer with kernel size 2 and stride 1 followed by an additive skip-connection layer. Each min-pooling layer is executed after its

Algorithm 3 The Time-Weighted DTW Algorithm, executed in diagonal order (without diagonal transition)

Input: Two MTSes $a \in \mathbb{R}^{n \times N_f}, b \in \mathbb{R}^{m \times N_f}$; a distance function $dist_w$; $u \in \mathbb{R}_{>0}^{n \times N_f}$ temporal feature weights of a
Output: Distance between a and b

1: $D \leftarrow \text{array}[n, m]$ {Out-of-bounds access returns $+\infty$}
2: $D_{0,0} \leftarrow dist_w(u_0, a_0, b_0)$
3: **for** l in $[1, n + m - 2]$ **do**
4: **for** i in $[0, n - 1]$ **do**
5: $j \leftarrow l - i$
6: **if** $j \in [0, m - 1]$ **then**
7: $D_{i,j} \leftarrow dist_w(u_i, a_i, b_j) + \min(D_{i-1,j}, D_{i,j-1})$
8: **end if**
9: **end for**
10: **end for**
11: **return** $D_{n-1,m-1}$

previous skip-connection and min-pooling are completed. So Algorithm 3, and therefore Algorithm 2 as well, is equivalent to a serial stack of min-pooling layers, with skip-connections interlacing in between. These descriptions are formalized in Algorithm 4, where one can see the execution of the interlaced layers at Lines 10 and 11. What makes the change in execution order powerful is that elements in the same diagonal are **mutually independent**, so the work within each diagonal is parallelized. So we have reduced the critical path length to the number of diagonals, obtaining the linear runtime $O(n + m)$ for Algorithm 4.

For differentiability, we rewrite the time-weighted DTW distance D_w as:

$$D_w := SC_{n+m-2} + MP_{n+m-2}(SC_{n+m-3} + MP_{n+m-3}(\ldots + SC_1 + MP_1(SC_0)\ldots))$$

where MP_l and SC_l are the l-th min-pooling and skip-connection as show in Algorithm 4. Because both min-pooling and additive skip-connections are standard neural network operations, loss gradients trivially propagate through them. In a nutshell, the loss backpropagates to only the $dist_w(u_i, a_i, b_j)$ functions on the optimal warping path, which enables gradient-based optimization on U and C. We leave the detailed formulas for loss gradients $\frac{\partial L_w}{\partial C}$ and $\frac{\partial L_w}{\partial U}$ in

Algorithm 4 Time-Weighted DTW Neural Network

Input: Two MTSes $a \in \mathbb{R}^{n \times N_f}, b \in \mathbb{R}^{m \times N_f}$; a distance function $dist_w$; $u \in \mathbb{R}_{>0}^{n \times N_f}$ temporal feature weights of a
Output: Distance between a and b

{Out-of-bounds access to D or SC returns $+\infty$}
$SC_{0,0} \leftarrow dist_w(u_0, a_0, b_0)$ {The 0-th skip-connection}
$D_{0,0} \leftarrow SC_{0,0}$
for l in $[1, n+m-2]$ **do**
    {The l-th diagonal}
    $\text{start}, \text{end} \leftarrow \max(0, l-m+1), \min(n-1, l)$
    $SC_l \leftarrow [dist_w(u_t, a_t, b_{l-t})]_{t=\text{start}}^{\text{end}}$
    $D_l \leftarrow \text{array}[\text{end} - \text{start} + 1]$
    $\text{kernel}, \text{stride} \leftarrow 2, 1$
    for i in $\text{range}(0, \text{end} - \text{start} + 1, \text{stride})$ **do**
        $D_{l,i} \leftarrow SC_{l,i} + \min_{s=0}^{\text{kernel}-1}(D_{l-1,i-s})$
    end for
end for
return $D_{n+m-2,0}$

Appendix A.

3.3 System Overview

With the definitions introduced in Secs. 3.1 and 3.2, we now formalize the 3-step solution outlined at the start of Chapter 3. Fig. 1.1 provides an overview of the DejaVid system. The video encoder takes as input a video clip of fixed duration and outputs an embedding $e \in \mathbb{R}^{N_f}$. Given any off-the-shelf video encoder, we apply a sliding window over each video to produce a sequence of clips, which we then feed into the encoder to produce a temporal sequence of embeddings $E = \{e_0, e_1, \ldots, e_{T-1}\} \in \mathbb{R}^{T \times N_f}$, which forms an MTS for each video sample in the dataset. Then, we produce a centroid MTS of shape $T_c \times N_f$ [2] for each class by feeding the training MTSes to the DBA (Dynamic Time Warping Barycenter Averaging) [26] algorithm, which calculates a reasonable "average" time series of the inputs while accounting for imperfect temporal alignment. Intuitively, each centroid MTS represents

[2] We set $T_c = 8$ for all experiments in this work.

how the class-wide average of the N_f features changes over time. Ultimately, we obtain many training MTSes, each with shape $T \times N_f$ for some varying T, and a stack of centroid MTSes C of shape $N_c \times T_c \times N_f$, where N_c is the number of classes. This concludes the pre-training stage of DejaVid.

In the training phase, to account for temporal variance in feature importance over time, each centroid MTS is paired with a positive weight tensor of the same shape $T_c \times N_f$ that represents the importance of each feature over time. The positivity is to maintain that the pointwise distance $dist_w$ is always positive. We denote the stack of weight MTSes U, with shape $N_c \times T_c \times N_f$. At initialization, the weight MTSes are either filled with ones or random positive values.

In each epoch, we calculate the weighted DTW distance D_w from each training MTS $m \in \mathbb{R}^{T \times N_f}$ to every centroid MTS to get the logit vector $\mathbf{z} = \{D_w(U_i, C_i, m)\}_{i=0}^{N_c-1} \in \mathbb{R}^{N_c}$, on which we apply softmin to get the class prediction $\hat{\mathbf{y}}$ and cross-entropy loss $\mathcal{L}_w$ w.r.t. the ground truth $\mathbf{y}$. The weights U and centroids C MTSes are updated during backpropagation. To accelerate forward and backward passes, we implement a custom DTW CUDA kernel that efficiently calculates a slope tensor $S \in \mathbb{R}^{N_c \times T_c \times N_f}$ and an intercept tensor $B \in \mathbb{R}^{N_c \times T_c \times N_f}$ such that $\texttt{torch.sum}(U \odot (C \odot S + B), \texttt{dim=(-2,-1)})$ equals the logit vector $\mathbf{z}$ and the effective gradients w.r.t. U and C equals $\frac{\partial L_w}{\partial U}$ and $\frac{\partial L_w}{\partial C}$. The implementation of the CUDA kernel, roughly speaking, leverages the fact that a weighted DTW distance is essentially the sum of many terms of $u \cdot |c - b|$, which equals $u \cdot (c \cdot (\pm 1) + (\mp 1)b)$, where the (± 1) term contributes to slope S and $(\mp 1)b$ to intercept B.

3.4 Modifications to Standard DTW

As mentioned in Sec. 3.1, unlike the original DTW algorithm, we exclude the diagonal transitions (i.e., transitioning from $D_{i-1,j-1}$ to $D_{i,j}$ in Algorithms 1 to 3 and from $D_{l-2,i-1}$ to $D_{l,i}$ in Algorithm 4) for DejaVid. The reason is due to model stability. When matching two

series of length n and m, by disallowing diagonal transitions, the warping path from $D_{0,0}$ to $D_{n-1,m-1}$ always takes $n+m-1$ steps, and so the DTW distance will always be the sum of $n+m-1$ step-wise distances. If diagonal transitions are allowed, the number of step-wise distances on the warping path can vary, destabilizing the model. Although beyond the scope of this paper, we note that there could be other ways to deal with diagonal transitions, such as assigning or learning a multiplicative or additive distance bias for diagonal vs. non-diagonal transitions, such as in Move-Split-Merge (MSM) [25], a more modern replacement of DTW that treats diagonal transitions differently from non-diagonal ones.

Another modification we made is to fix epoch centroids and weights. That is, at the beginning of an epoch, we make a frozen copy of the current centroids and weights, and then during the epoch, use them to calculate the warping path for each centroid-training MTS pair. Then, along the path, we calculate the pointwise distance using the (non-frozen) originals and perform backpropagation on them. Note that the frozen copies are not updated per batch but instead per epoch. The reason for this modification is again due to model stability. We found that if the warping path calculation mechanism is updated every batch, the distances become noisier from batch to batch, and the model has a harder time converging.

We discuss the effect of these modifications in Sec. 4.4.

Chapter 4

Evaluation

In this section, we describe our implementation and evaluation of DejaVid.

4.1 Datasets and Evaluation Metrics

We test DejaVid on three different popular action recognition datasets: Something-Something V2 [5], Kinetics-400 [6], and HMDB51 [7]. The first two are large-scale datasets: Something-Something V2 has 168,913 videos for training and 24,777 for validation, while Kinetics-400 has 240,436 training and 19,787 validation videos. In contrast, HMDB51 only has 3,570 training and 1,530 validation videos, but its smaller size can help us study the model's performance when the training set is smaller.

Something-Something V2 has 174 action classes that are object and motion-centric, which requires learning the subtle differences between motions and object interactions. Kinetics-400 covers 400 classes of everyday human actions derived from real-world scenarios on YouTube and can be human-object or human-human interactions. Its videos are also longer than those of Something-Something V2 on average ($\sim$ 300 frames vs. $\sim$ 40 frames), which can help differentiate the temporal modeling capabilities between models. HMDB51 has 51 human-dominated action classes, like Kinetics-400, containing some individual and some human-human behaviors.

For all these datasets, we train DejaVid on the training sets and test on the validation sets using the same evaluation protocols as those in VideoMAE V2 [8]. We report the top-1 and top-5 accuracy for every experiment on Something-Something V2 and Kinetics-400. Due to the smaller action set of HMDB51, we only report top-1 for experiments on it.

4.2 Implementation Details

Pre-training. We use the largest (ViT-g) model from VideoMAE V2 [8] (abbr. VideoMAE V2-g) as the video encoder [1], which by itself takes 16 frames of shape 224×224 as input and outputs a length-1408 representation and then a length-N_c logit vector, where N_c is the number of classes in the dataset. Thus the size of the encoder output N_f is $1408 + N_c$. For DejaVid, we apply a temporal sliding window across the input video, take the center crop, and resize to 224×224 to feed into the encoder. This gives us an MTS of shape $T \times N_f$ for some T. Then, for each action class, we randomly sample 50 MTSes, reshape each of them to $T_c \times N_f$ with linear interpolation, and then run 100 iterations of the DBA algorithm [26] to produce the centroid.

We now describe our choice of the temporal sliding window widths and strides. For Kinetics-400 and HMDB51, given a video, VideoMAE V2 temporally segments the video into 5 clips of the same length and takes 3 crops at the left, center, and right to produce $5 \times 3 = 15$ logit vectors, from which they then take the mean to produce the class prediction. Note that the temporal treatment is equivalent to a sliding window of width $\frac{|vid|}{5}$ and stride $\frac{|vid|}{5}$, where $|vid|$ is the video length. On the other hand, we only use the center crop, but deploy a sliding window of width $\frac{|vid|}{5}$ and stride $\frac{|vid|}{40}$, so we produce $(\frac{40}{5} \cdot (5-1)+1) \times 1 = 33$ logit vectors per video, with the resulting MTS having dimension $33 \times N_f$.

For Something-Something V2, unlike the other two datasets, VideoMAE V2 does not

[1] VideoMAE V2 publishes different finetuned weights for Kinetics-400 and Something-Something V2, and the script for finetuning on HMDB51, which we can afford to run thanks to the small dataset size. We use the respective finetuned weights for the pre-training stage of each dataset.

temporally segment but instead performs a strided slice on the frames with a step of 2. This means that the encoder is finetuned to an input window width of $|vid|$, which complicates our sliding window application. The vast majority of Kinetics-400 videos are of length ~ 300 frames, but videos in Something-Something V2 vary more in frame count, ranging from the teens to over a hundred, which means its encoder window width varies more too. In order to provide DejaVid with both constant-width and variable-width information, we apply four sliding windows with width $\{16, 32, 64, |vid|\}$ and stride 1 in parallel, and thus obtain for each video an MTS of dimension $|vid| \times (4 \cdot (1408 + N_c))$. The average video length in Something-Something V2 is ~ 40 frames, so on average, we produce $\frac{4 \cdot 40}{33} = 4.8$ times more embeddings per video than for Kinetics-400 and HMDB51.

Training & Testing. The pre-training stage gives us an MTS m for each training or validation video as well as the centroid MTSes C. The weight tensor U is either randomly initialized or one-initialized (i.e., filled with ones). To enforce positivity, we actually store the log of the weights and exponentiate them during forward passes. These log-of-weights are randomly initialized or zero-initialized (i.e., filled with zeros). We use the AdamW optimizer with betas $(0.9, 0.999)$ and no weight decay. The initial learning rate for the log-of-weights is 1e-3 for both Something-Something V2 and Kinetics-400, and 8e-3 for HMDB51. The learning rates for the centroid MTSes are one-third of the log-of-weights. We use batch size 48 and train for 36 epochs. We use a cosine scheduler across all 36 epochs. A validation run is performed after every epoch, and we report the top accuracy recorded across all epochs.

4.3 Main Results

This section compares our results with previous other temporal-related and state-of-the-art methods. The main baseline is VideoMAE V2-g, which is a billion-parameter video encoder that is state-of-the-art or near state-of-the-art on all evaluation datasets [5–8]. We test our method on each dataset with three different settings: random-initialization for the weight

MTSes U (denoted as *full learning; w-init-rnd* in the following charts), one-initialization for U (denoted as *full learning; w-init-one*), and finally one-initialization for U but freezing it throughout (i.e., disallowing `optimizer.step()` to act on U; denoted as *frozen weights; w-init-one*). Note that the last setting is for experimenting with the unweighted DTW distance as in Algorithm 1, so frozen weights with random initialization don't make sense.

Results on Kinetics-400. As shown in Tab. 4.1, our method achieves leading accuracy on Kinetics-400. We improve on the strongest VideoMAE V2-g baseline by 0.7%. DejaVid only requires 5.8M parameters, less than 0.6% of the encoder's parameter count, to achieve this gain. For comparison, the gain from VideoMAE V1 to VideoMAE V2 uses 380M extra parameters to increase the accuracy by only 0.4%. Note that our results are achieved without re-finetuning the encoder, so the training cost is minimal compared to training the encoder. In short, DejaVid is a comparatively tiny add-on that significantly boosts the performance of an off-the-shelf frozen model.

The performance gain of DejaVid is not only due to temporal supersampling. We test this by averaging the temporal dimension of each sample MTS to get a length-$(1408 + N_c)$ vector and using the last N_c elements for class prediction; the result is shown as VideoMAE V2 (temporal supersampling) Tab. 4.1. When compared against the original setting at VideoMAE V2 5×3, we see this alone does not improve the accuracy. Also, the temporal weight-learning does not provide additional benefits on top of our centroid-finetuning for Kinetics-400. We suspect the reason is that the temporal weights increase the expressivity of our method, so overfitting becomes more likely; one future work is to look into regularization for DejaVid.

Results on HMDB51. The results in Tab. 4.1 show that our model also achieves state-of-the-art accuracy on the HMDB51. We achieve a 0.5% performance gain against the strongest VideoMAE V2-g baseline. To the best of our knowledge and according to Papers with Code, this is the highest accuracy recorded on the HMDB51 action recognition benchmark to date (as of Nov, 2024). This shows that DejaVid works well in a training data-scarce scenario. We draw similar conclusions to those from Kinetics-400 in parameter efficiency –

Method	W.A.	Kinetics-400				Something-Something V2				HMDB51		
		Params	Clips×Crops	Top-1	Top-5	Params	Clips×Crops	Top-1	Top-5	Params	Clips×Crops	Top-1
Other State-of-the-Art Large Models												
TubeViT [11]	No	632M	4×3	90.9	98.9	307M	-	76.1	95.2	-	-	-
MVD [10]	No	633M	5×3	87.2	97.4	633M	2×3	77.3	95.7	-	-	-
InternVideo2-1B [9]	Yes[2]	1B	4×3	91.6	-	1B	2×3	77.1	-	-	-	-
InternVideo2-6B [9]	No	6B	4×3	92.1	-	6B	5×3	77.5	-	-	-	-
Other Temporal-Focused Methods												
ILA [13]	-	-	4×3	88.7	97.8	-	4×3	70.2	91.8	-	-	-
ATM [14]	-	-	4×3	**89.4**	**98.3**	-	2×3	74.6	94.4	-	-	-
TAM [18]	-	-	4×3	79.3	94.1	-	2×3	66.0	90.1	-	-	-
SlowFast [16]	-	-	10×3	79.8	93.9	-	-	61.7	-	-	-	-
MTV [15]	-	-	4×3	**89.1**	**98.2**	-	-	68.5	90.4	-	-	-
Main Baselines												
VideoMAE V1-H [8]	Yes	633M	5×3	88.1	-	-	-	-	-	-	-	-
VideoMAE V1-L (32-frame input) [8, 27]	Yes	-	-	-	-	305M	2×3	75.4	95.2	-	-	-
VideoMAE V2-g [8]	Yes	1013M	5×3	88.4	98.0	1013M	2×3	76.7[3]	95.8[3]	1013M	5×3	88.1
VideoMAE V2-g (temporal supersampling) [8]	Yes	1013M	33^4×1	88.1	97.8	1013M	$(4\|vid\|)^4$×1	76.2	95.7	1013M	33^4×1	88.0
Our Results												
Ours (frozen weights; w-init-one)	-	+5.8M		**89.1**	**98.2**	+8.8M		77.1	**96.3**	+0.60M		88.3
Ours (full learning; w-init-rnd)	-	+11.6M	33^4×1	88.9	98.1	+17.6M	$(4\|vid\|)^4$×1	**77.2**	**96.3**	+1.19M	33^4×1	**88.6**
Ours (full learning; w-init-one)	-	+11.6M		89.0	**98.2**	+17.6M		77.0	**96.3**	+1.19M		**88.6**

Table 4.1: Comparison with the VideoMAE V2-g baseline and other relevant variants, SOTA, or temporal methods on Kinetics-400, Something-Something V2, and HMDB51.
All accuracies are reported in percent (%). W.A. = Weights Available. Dashes (-) = N/A.

our parameter count is 1.19M, which is less than 0.12% of that of VideoMAE V2-g – and temporal supersampling alone does not improve performance.

Results on Something-Something V2. Tab. 4.1 compares our results on Something-Something V2 against other works. We outperform the top VideoMAE V2-g baseline with a performance gain of 0.5%. To the best of our knowledge and according to Papers with Code, DejaVid achieves the highest top-5 accuracy and one of the highest top-1 accuracies on the Something-Something V2 action recognition benchmark to date (as of Nov, 2024). With only 17.6M parameters, DejaVid uses fewer than 1.8% of the parameters required by VideoMAE V2-g to deliver this gain. When compared to the differences between VideoMAE V1 and VideoMAE V2, DejaVid again shows a much stronger accuracy gain per million extra parameters.

However, the performance gain is smaller than on Kinetics-400. One possible reason could be the difference in average video length: ~ 40 for Something-Something V2 and ~ 300 for Kinetics-400. Because Kinetics-400 videos are longer, there may be more temporal information for DejaVid to capture. Another reason may be that the encoder is trained with variable video durations, as described in Sec. 4.2, so one future work would be to re-finetune

the encoder with video clips of constant frame-count and retry our experiments.

4.4 Ablation Studies

Centroid vs. Weight Learning. In Tab. 4.2, we analyze the impacts of centroid and weight learning of DejaVid by freezing one component at a time and comparing the results. As seen, both frozen-centroid and frozen-weight variants yield meaningful accuracy gains against the VideoMAE V2-g baseline, and so both components are indeed beneficial. The centroid component has a greater performance gain on average across the three datasets, giving +0.43% top-1 accuracy gain vs. 0.25% for the weights, but the weight component seems to be more useful when training data is scarce. As for the training data-rich cases, the weight component seems to be more useful in Something-Something V2 than in Kinetics-400, potentially because we have 4 different feature sets from the different sliding window widths, and these features should naturally have different importance, so the one-initialized weights are further away from the global optimal.

Method	Sth-Sth2		K400		HMDB
	top-1	top-5	top-1	top-5	top-1
VideoMAE V2-g @ best [8]	76.7[3]	95.8[3]	88.4	98.0	88.1
Frozen centroids (w-init-rnd)	76.8	96.2	88.6	98.0	88.4
Frozen centroids (w-init-one)	76.9	96.2	88.8	98.1	88.4
Frozen weights (w-init-one)	77.1	**96.3**	**89.1**	**98.2**	88.3
Full learning (w-init-rnd)	**77.2**	**96.3**	88.9	98.1	**88.6**
Full learning (w-init-one)	77.0	**96.3**	89.0	**98.2**	**88.6**

Table 4.2: Ablation study on centroid & weight learning.

[2]We were unable to run InternVideo2 on our machine due to one of its required packages, `flash-attn`, being incompatible with our machine.

[3]The authors of VideoMAE V2 report 77.0 and 95.9 for top-1 and top-5 accuracy, respectively, but we measure only 76.7 and 95.8 using their published weights. We suspect the discrepancy is due to the input video loading mechanism. The official GitHub repo of VideoMAE V2 has inconsistent descriptions about the data preparation for Something-Something V2, but as of the time of writing, the source code assumes each video has been converted to a folder of jpeg images, and VideoMAE V2 loads these jpeg files instead of the source video. But when we do exactly that, the top-1 accuracy measures only 75.7. This is understandable since JPEG uses lossy compression. So we modified the source code to use opencv2 to directly load each video, which gives 76.7 and 95.8.

[4]The definition of these clips are described in Sec. 4.2.

Method	Sth-Sth2		K400		HMDB
	top-1	top-5	top-1	top-5	top-1
DT on, otherwise best	77.0	96.2	88.6	98.0	88.5
FECW off, otherwise best	77.1	96.2	89.0	**98.2**	88.5
Best (DT off, FECW on)	**77.2**	**96.3**	**89.1**	**98.2**	**88.6**

Table 4.3: Ablation studies on diagonal transition (DT) and fixed epoch centroids & weights (FECW).

Method	Sth-Sth2	K400	HMDB
Naive DTW implementation	4.79e4[5]	1.17e5[5]	294
Custom DTW CUDA kernel	**297**	**156**	**1.01**

Table 4.4: Ablation study on the custom DTW CUDA kernel.

Table shows runtime per training epoch in seconds.

Diagonal Transition of DTW. Tab. 4.3 presents a comparison between the best (highest top-1 accuracy[6]) setting of Tab. 4.2 for each dataset and the same setting but with diagonal transitions turned on. Enabling diagonal transitions consistently performs worse on all datasets. This justifies our choice of disabling diagonal transitions in DejaVid and verifies the insight behind this choice as described in Sec. 3.4.

Fixed Epoch Centroids & Weights for DTW Learning. We use Tab. 4.3 to compare the best setting (highest top-1 accuracy[6]) from Tab. 4.2 for each dataset with the same

[5]Estimated by running 3 batches.

[6]In the case of tied top-1 accuracies, we apply the same change to each tied setting and average their resulting accuracies after the change.

setting but disabling fixed epoch centroids & weights, where we do not make frozen copies of the centroids and weights for warping-path determination (shown as FECW off). Again, the inferior performance of this group compared to when FECW is on across all datasets justifies our design for DejaVid, aligning with the reasons detailed in Sec. 3.4 about model stability. However, the gap is less significant than that for the diagonal transition. This suggests there may be room for improvement by tweaking the path-determination mechanism, such as only updating every other epoch instead of each epoch.

Custom CUDA Kernel for Accelerating DTW. Tab. 4.4 summarizes the runtime per epoch of the best-performing settings on all datasets compared to using a naive Python implementation of DTW. All experiments are conducted on a single node with 8 NVIDIA H100 GPUs. We can see that our custom CUDA kernel provides a speedup between 161x and 750x. The most technically significant speedup comes from reducing the critical path length from quadratic to linear, as discussed in Sec. 3.2. Other factors contributing to the speedup include the Python/C++ speed difference and architecture and locality-aware, LibTorch-native implementation of the CUDA-kernel.

Chapter 5

Conclusion

We presented DejaVid, an encoder-agnostic method that boosts video classification performance without retraining the encoder or changing its architecture. Instead of converting a video into a single fixed-length embedding like most state-of-the-art encoders do, we feed a backbone encoder a sliding-window stream of frames to obtain a variable-length temporal sequence of embeddings, or a multivariate time series (MTS), which preserves temporal order and supports variable video durations. We then introduce a novel neural network architecture based on Dynamic Time Warping (DTW) with learnable, per-timestamp feature weights to address temporal variance in feature significance. Our ablation studies compare and justify various design choices and modifications to classical DTW, and our evaluation shows that DejaVid achieves state-of-the-art video classification results on a variety of benchmarks, significantly improves the accuracy of a cutting-edge large encoder, and outperforms other temporal methods. In particular, we achieve leading Top-1 accuracy of 77.2% on Something-Something V2, 89.1% on Kinetics-400, and 88.6% on HMDB51, representing a 0.5-0.7% accuracy gain over VideoMAE V2-g while adding less than 1.8% additional learnable parameters and less than 3 hours of training time.

Appendix A

Formulas for Loss Gradients

This section supplements Sec. 3.2 by proving the differentiability of the Algorithm 4 neural network of DejaVid, namely the detailed formulas for $\frac{\partial L_w}{\partial U}$ and $\frac{\partial L_w}{\partial C}$, which are omitted at the end of Sec. 3.2.

Recall from the end of Sec. 3.1 that we calculate the time-weighted distance from a training or validation MTS m to the centroid MTS C_i of each class i and then feed the class-wise distances to soft-min for class prediction. We first observe that before the soft-min, the distance calculations for each class are independent of each other; they do not share any elements of C or U, nor do they have any inter-class connections. So we can individually calculate $\frac{\partial L_w}{\partial U[c]}$ and $\frac{\partial L_w}{\partial C[c]}$ for each class c, then stack them together for the final $\frac{\partial L_w}{\partial U}$ and $\frac{\partial L_w}{\partial C}$.

Note that $\frac{\partial L_w}{\partial U[c]}$ and $\frac{\partial L_w}{\partial C[c]}$ are the combination of three components:

$$\frac{\partial L_w}{\partial U[c]} = \frac{\partial L_w}{\partial D_w[c]} \sum_l \frac{\partial D_w[c]}{\partial SC[c,l]} \frac{\partial SC[c,l]}{\partial U[c]}$$

$$\frac{\partial L_w}{\partial C[c]} = \frac{\partial L_w}{\partial D_w[c]} \sum_l \frac{\partial D_w[c]}{\partial SC[c,l]} \frac{\partial SC[c,l]}{\partial C[c]}$$

where l is the index of the diagonal, $D_w \in \mathbb{R}^{N_c}$ the distance from m to the centroid of each class, $SC[c,l]$ is the l-th skip-connection for class c as in Algorithm 4, and $C \in \mathbb{R}^{N_c \times T_c \times N_f}$, $U \in \mathbb{R}_{>0}^{N_c \times T_c \times N_f}$ are as defined in Sec. 3.1. The following tackles each of the three components respectively.

For $\frac{\partial L_w}{\partial D_w[c]}$, we use the standard cross-entropy and soft-min, so the derivative is well-known to be:

$$\frac{\partial L_w}{\partial D_w[c]} = \mathbf{y}[c] - p[c]$$

where $\mathbf{y}$ is the one-hot ground truth vector and $p[c]$ is the predicted probability of class c.

For $\frac{\partial D_w[c]}{\partial SC[c,l]}$, the standard trick for calculating loss gradients of a min-pooling layer is to define an indicator matrix. Note that the l-th min-pooling layer for class c has length $\|SC[c,l]\|$. Let $R[c,l]$ of shape $\|SC[c,l]\| \times \|SC[c,l-1]\|$ be the indicator matrix of the i-th

min-pooling for class c. We have:

$$R[c,l,a,b] = \begin{cases} 1 & \text{if } b \in \{a-1, a\} \text{ and} \\ & \text{the } a\text{-th output of the min-pooling} \\ & = \text{ the } b\text{-th input of the min-pooling} \\ 0 & \text{otherwise} \end{cases}$$

And since the min-pooling layers are chained, we have:

$$\frac{\partial D_w[c]}{\partial SC[c,l]} = \prod_{i=n+m-2}^{l+1} R[c,i]$$

Notably, $\|SC[c, n+m-2]\| = 1$, so the matrix product results in a shape of $\|SC[c, n+m-2]\| \times \|SC[c,l]\| = 1 \times \|SC[c,l]\|$.

Finally, for $\frac{\partial SC[c,l]}{\partial U[c]}$, first notice that for any given i, $U[c,i]$ can only contribute to $SC[c,l]$ at the entry with $dist_w(U[c,i], C[c,i], m[l-i])$. Denoting $\frac{\partial SC[c,l]}{\partial U[c]}$ as $dU_l[c] \in \mathbb{R}^{\|SC[c,l]\| \times T_c \times N_f}$, we thus have:

$$dU_l[c,i,j,f] = \begin{cases} |C[c, i+\text{start}, f] - m[j,f]| \\ \text{if } i + \text{start} + j = l \\ 0 \text{ otherwise} \end{cases}$$

where $\text{start} = \max(0, l - \dim_0(m) + 1)$ is the offset for the 0-th element of $SC[c,l]$, as in Line 6 of Algorithm 4.

Similarly for $\frac{\partial SC[c,l]}{\partial C[c]}$, first notice that for any given i, $C[c,i]$ can only contribute to $SC[c,l]$ at the entry with $dist_w(U[c,i], C[c,i], m[l-i])$. Denoting $\frac{\partial SC[c,l]}{\partial C[c]}$ as $dC_l[c] \in \mathbb{R}^{\|SC[c,l]\| \times T_c \times N_f}$, we thus have:

$$dC_l[c,i,j,f] = \begin{cases} U[c, i+\text{start}, f] \cdot \text{sign}(C[c, i+\text{start}, f] - m[j,f]) \\ \text{ if } i + \text{start} + j = l \\ 0 \text{ otherwise} \end{cases}$$

which concludes the formulas for loss gradients $\frac{\partial L_w}{\partial U}$ and $\frac{\partial L_w}{\partial C}$. This demonstrates the differentiability of the Algorithm 4 neural network of DejaVid, which enables optimization via backpropagation.

References

[1] 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*. 2017, pp. 6299–6308.

[2] A. Dosovitskiy. "An image is worth 16x16 words: Transformers for image recognition at scale". In: *arXiv preprint arXiv:2010.11929* (2020).

[3] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lučić, and C. Schmid. "Vivit: A video vision transformer". In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 6836–6846.

[4] X. Zhai, A. Kolesnikov, N. Houlsby, and L. Beyer. "Scaling vision transformers". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 12104–12113.

[5] R. Goyal, S. Ebrahimi Kahou, V. Michalski, J. Materzynska, S. Westphal, H. Kim, V. Haenel, I. Fruend, P. Yianilos, M. Mueller-Freitag, et al. "The" something something" video database for learning and evaluating visual common sense". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 5842–5850.

[6] W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, P. Natsev, et al. "The kinetics human action video dataset". In: *arXiv preprint arXiv:1705.06950* (2017).

[7] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. "HMDB: a large video database for human motion recognition". In: *2011 International conference on computer vision*. IEEE. 2011, pp. 2556–2563.

[8] L. Wang, B. Huang, Z. Zhao, Z. Tong, Y. He, Y. Wang, Y. Wang, and Y. Qiao. "Videomae v2: Scaling video masked autoencoders with dual masking". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 14549–14560.

[9] Y. Wang, K. Li, X. Li, J. Yu, Y. He, G. Chen, B. Pei, R. Zheng, J. Xu, Z. Wang, et al. "Internvideo2: Scaling video foundation models for multimodal video understanding". In: *arXiv preprint arXiv:2403.15377* (2024).

[10] R. Wang, D. Chen, Z. Wu, Y. Chen, X. Dai, M. Liu, L. Yuan, and Y.-G. Jiang. "Masked video distillation: Rethinking masked feature modeling for self-supervised video representation learning". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2023, pp. 6312–6322.

[11] A. Piergiovanni, W. Kuo, and A. Angelova. "Rethinking video vits: Sparse video tubes for joint image and video learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 2214–2224.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012).

[13] S. Tu, Q. Dai, Z. Wu, Z.-Q. Cheng, H. Hu, and Y.-G. Jiang. "Implicit temporal modeling with learnable alignment for video recognition". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 19936–19947.

[14] W. Wu, Y. Song, Z. Sun, J. Wang, C. Xu, and W. Ouyang. "What Can Simple Arithmetic Operations Do for Temporal Modeling?" In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 13712–13722.

[15] S. Yan, X. Xiong, A. Arnab, Z. Lu, M. Zhang, C. Sun, and C. Schmid. "Multiview transformers for video recognition". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 3333–3343.

[16] C. Feichtenhofer, H. Fan, J. Malik, and K. He. "Slowfast networks for video recognition". In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 6202–6211.

[17] D. J. Berndt and J. Clifford. "Using dynamic time warping to find patterns in time series". In: *Proceedings of the 3rd international conference on knowledge discovery and data mining*. 1994, pp. 359–370.

[18] Z. Liu, L. Wang, W. Wu, C. Qian, and T. Lu. "Tam: Temporal adaptive module for video recognition". In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 13708–13718.

[19] P. Senin. "Dynamic time warping algorithm review". In: *Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA* 855.1-23 (2008), p. 40.

[20] M. Cuturi and M. Blondel. "Soft-dtw: a differentiable loss function for time-series". In: *International conference on machine learning*. PMLR. 2017, pp. 894–903.

[21] X. Cai, T. Xu, J. Yi, J. Huang, and S. Rajasekaran. "Dtwnet: a dynamic time warping network". In: *Advances in neural information processing systems* 32 (2019).

[22] 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*. 2019, pp. 3546–3555.

[23] K. Cao, J. Ji, Z. Cao, C.-Y. Chang, and J. C. Niebles. "Few-shot video classification via temporal alignment". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 10618–10627.

[24] J. Gao, M. Chen, and C. Xu. "Fine-grained temporal contrastive learning for weakly-supervised temporal action localization". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 19999–20009.

[25] A. Stefan, V. Athitsos, and G. Das. “The move-split-merge metric for time series”. In: *IEEE transactions on Knowledge and Data Engineering* 25.6 (2012), pp. 1425–1438.

[26] F. Petitjean, A. Ketterlin, and P. Gançarski. “A global averaging method for dynamic time warping, with applications to clustering”. In: *Pattern recognition* 44.3 (2011), pp. 678–693.

[27] Z. Tong, Y. Song, J. Wang, and L. Wang. “Videomae: Masked autoencoders are data-efficient learners for self-supervised video pre-training”. In: *Advances in neural information processing systems* 35 (2022), pp. 10078–10093.